

Session 5: Data Visualization

R for Stata Users

Rony Rodrigo Maximiliano Rodriguez-Ramirez

The World Bank – DIME | [WB Github](#)

16 November 2020



Table of contents



1. Introduction
2. Exploratory Analysis
3. {ggplot2}'s settings
4. Saving a plot
5. References and recommendations

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`.

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



Before we start

- Make sure you 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)
```



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 also `{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 not cleaned or 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.



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 customizaation (way better than Stata).
4. Community

{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.

{ggplot2}'s decomp



I will stick to Thomas Lin Pedersen's decomposition.



{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

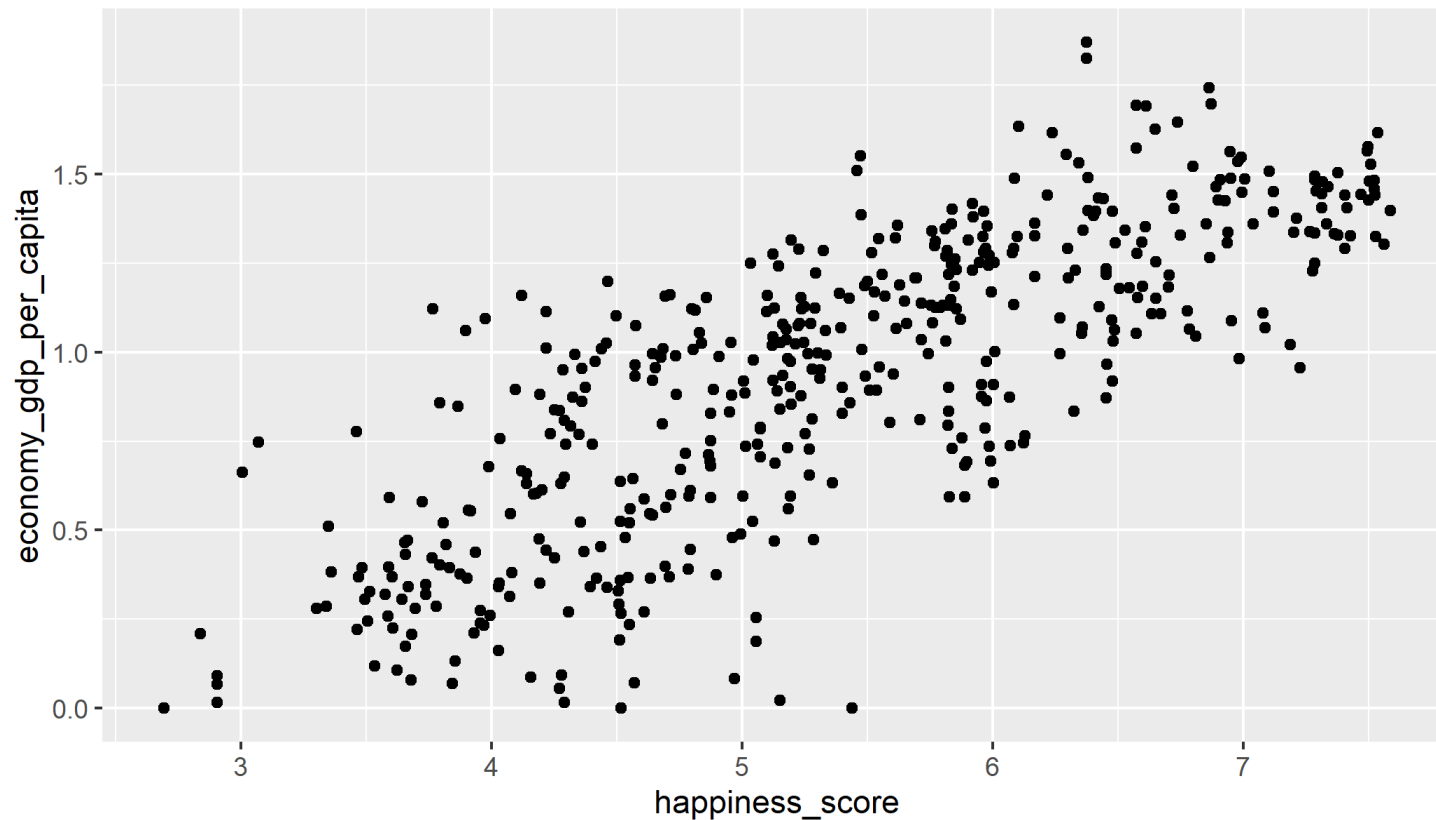
Exploratory Analysis

Exploratory Analysis



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

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

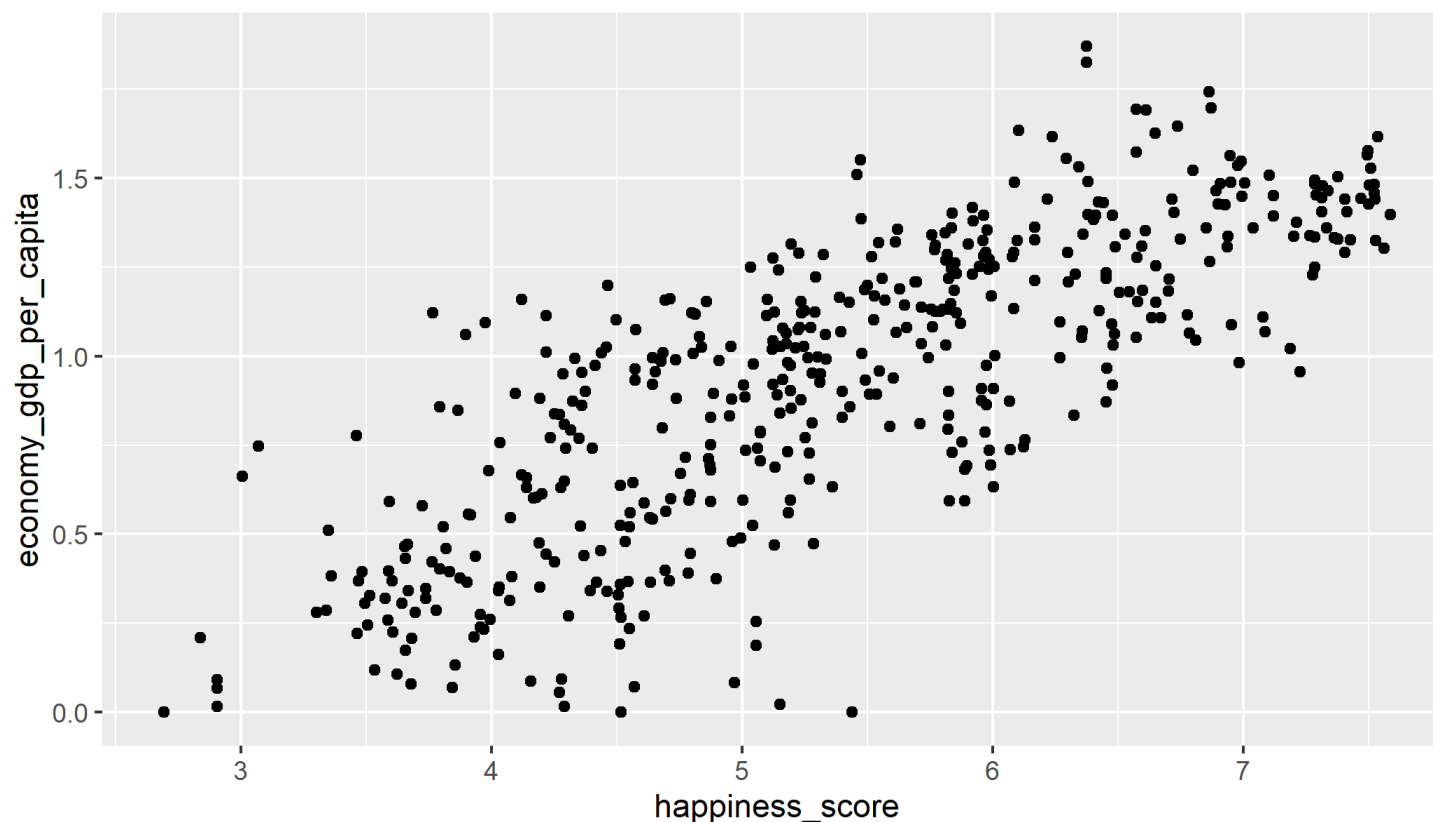


Exploratory Analysis



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

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

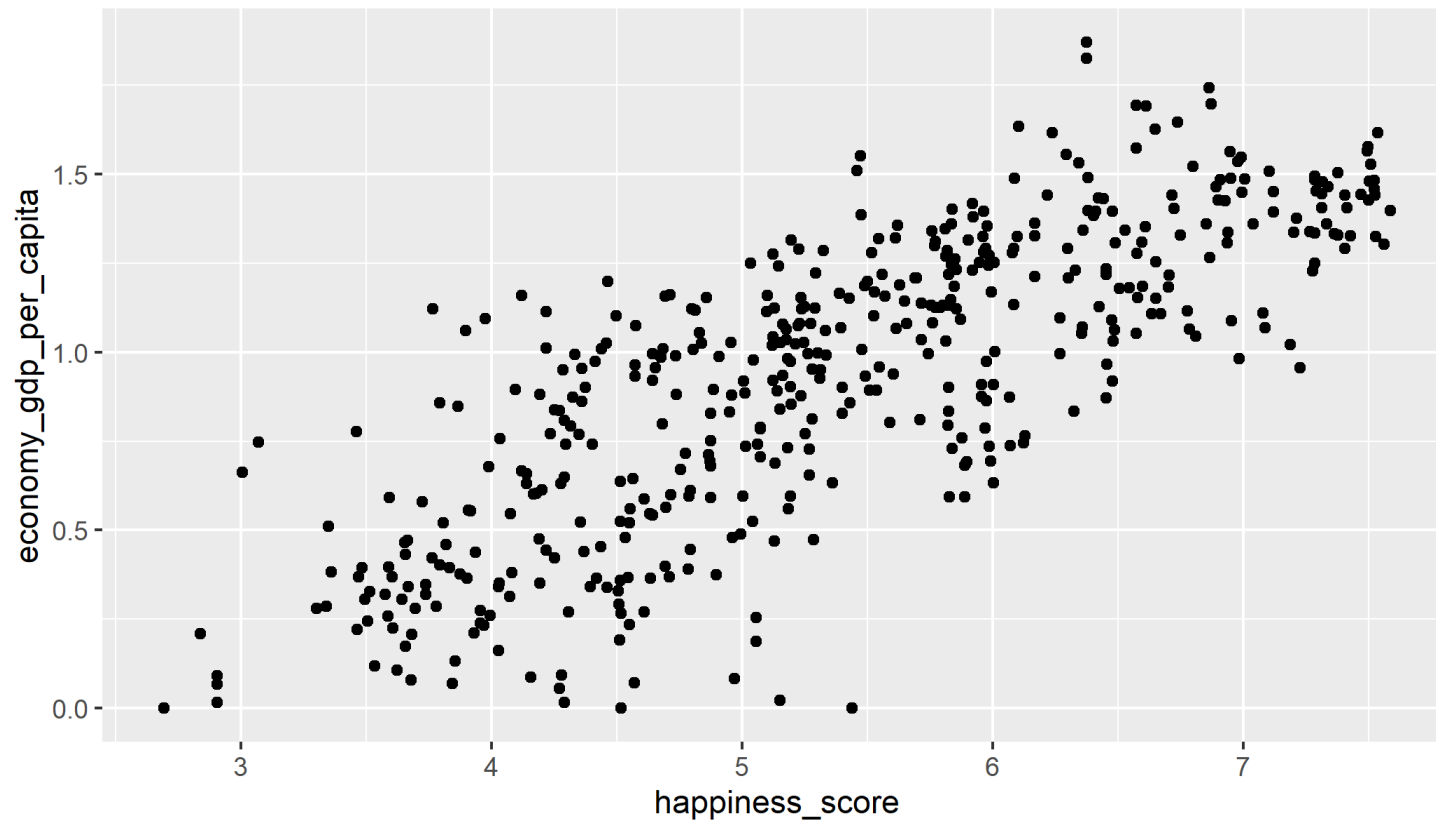


Exploratory Analysis



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



The most common `geoms` are:

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

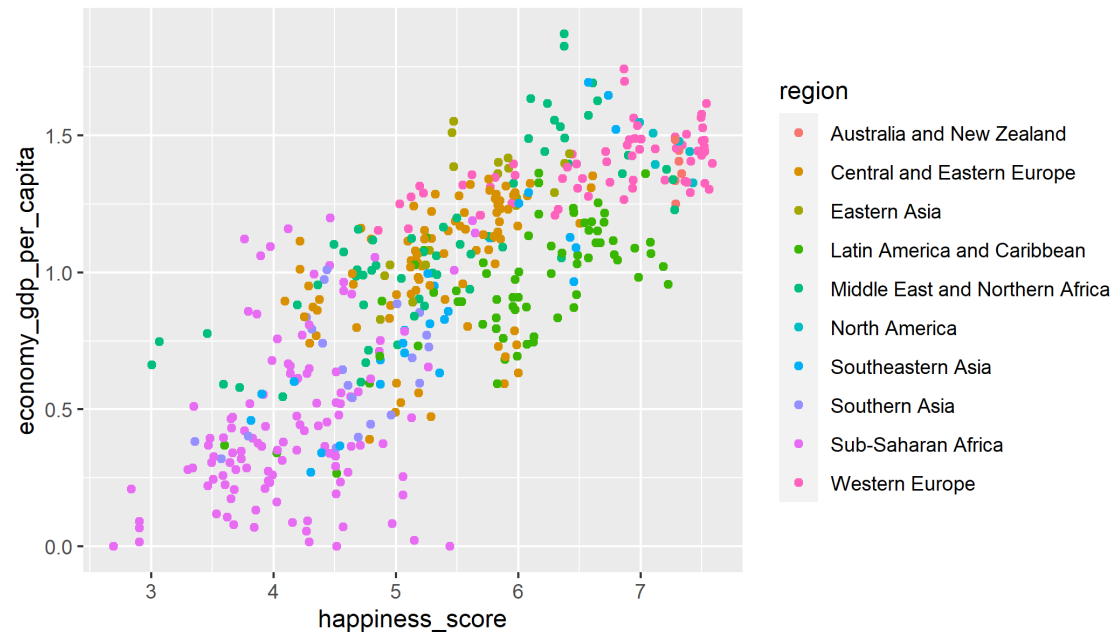
How the plot is going to be drawn.

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 = season))` to color points based on the variable `country`
- `geom_point(color = "grey")` to color all points in the same color.

Exploratory Analysis



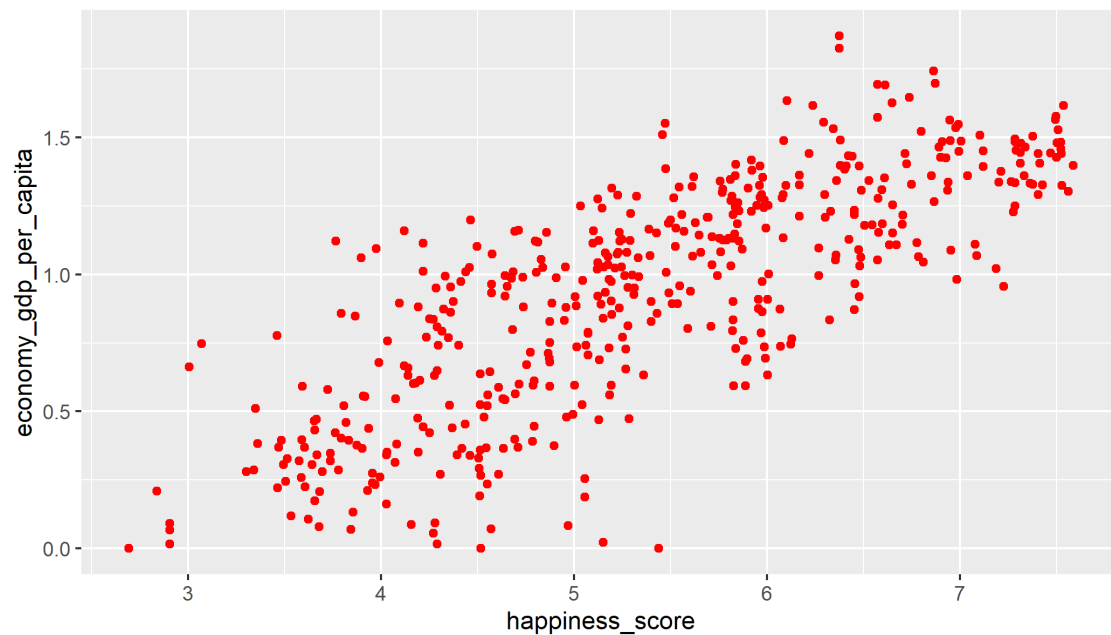
Let's modify our last plot.

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



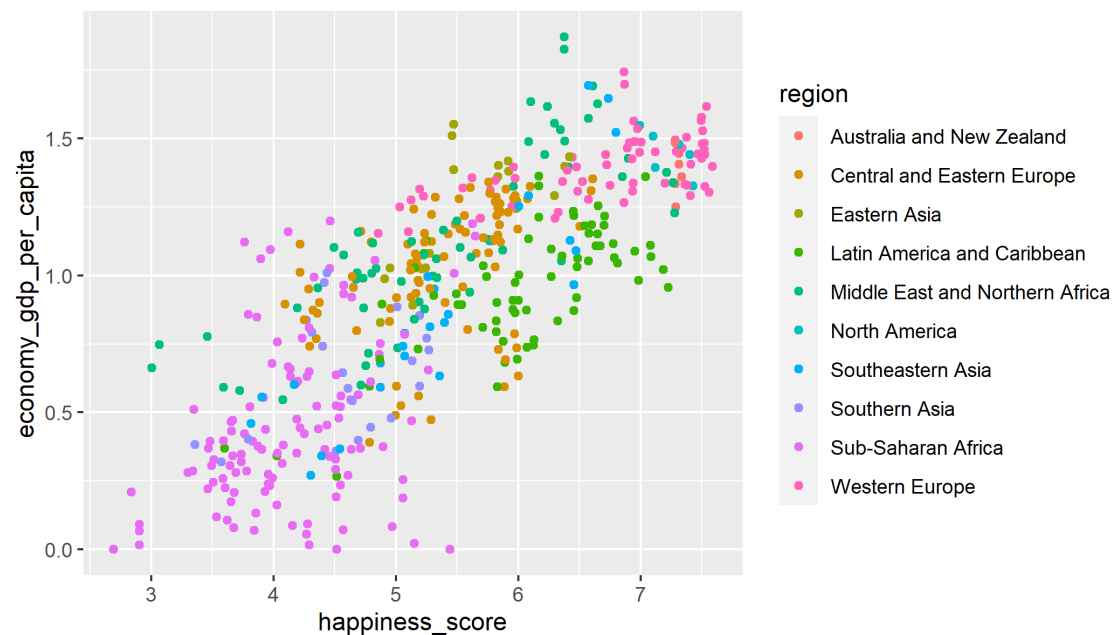
Let's modify our last plot.

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))
```

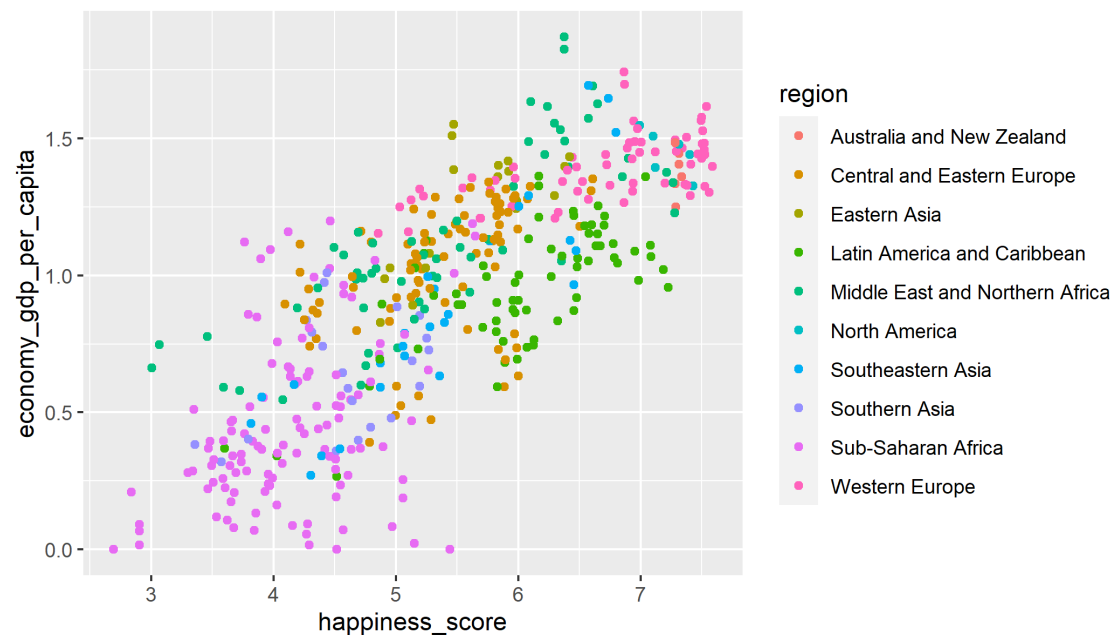


Exploratory Analysis



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 or 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.

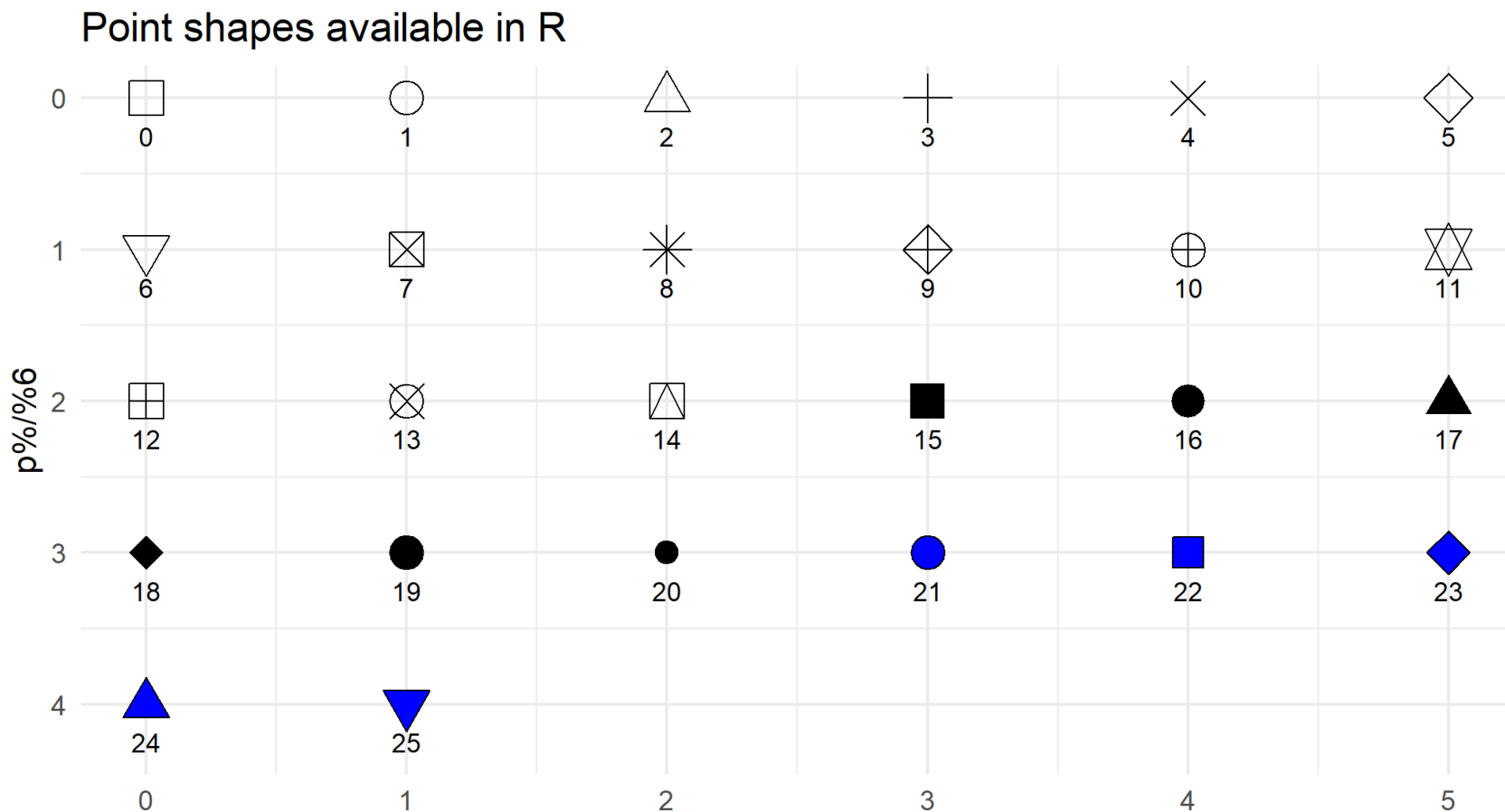
{ggplot2}: shapes



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



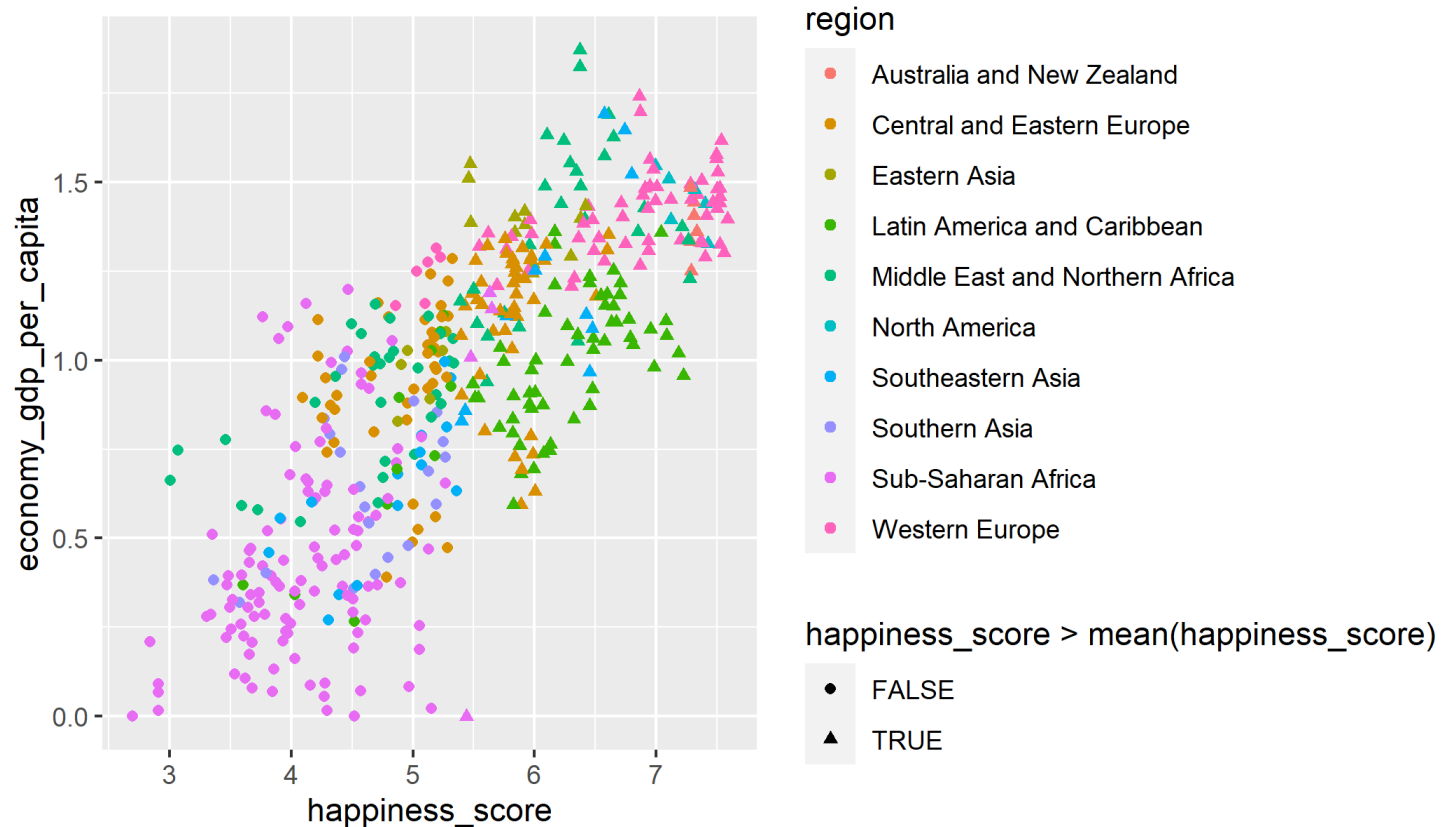
{ggplot2}: shapes



{ggplot2}: shapes and colors



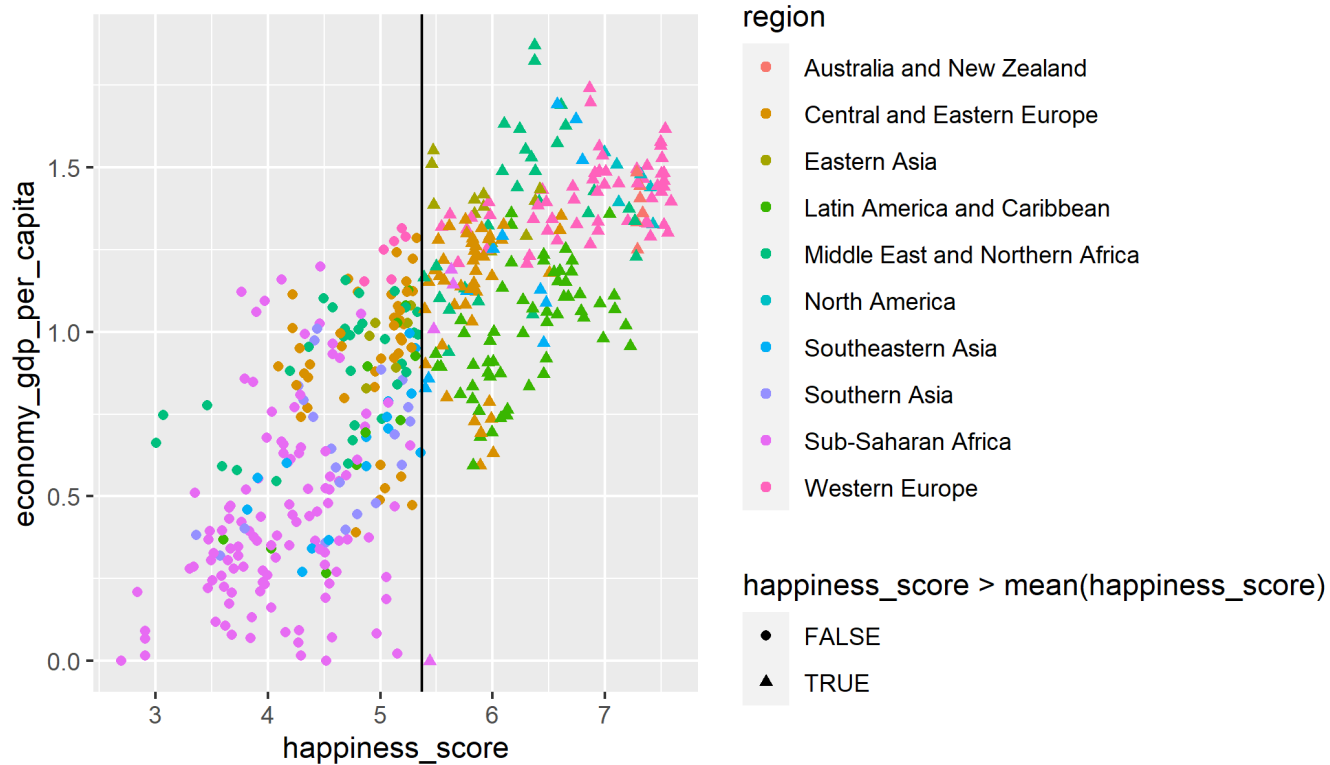
```
ggplot(data = whr_panel,  
       aes(x = happiness_score, y = economy_gdp_per_capita,  
           shape = happiness_score > mean(happiness_score),  
           color = region)) +  
  geom_point()
```



{ggplot2}: including more geoms



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

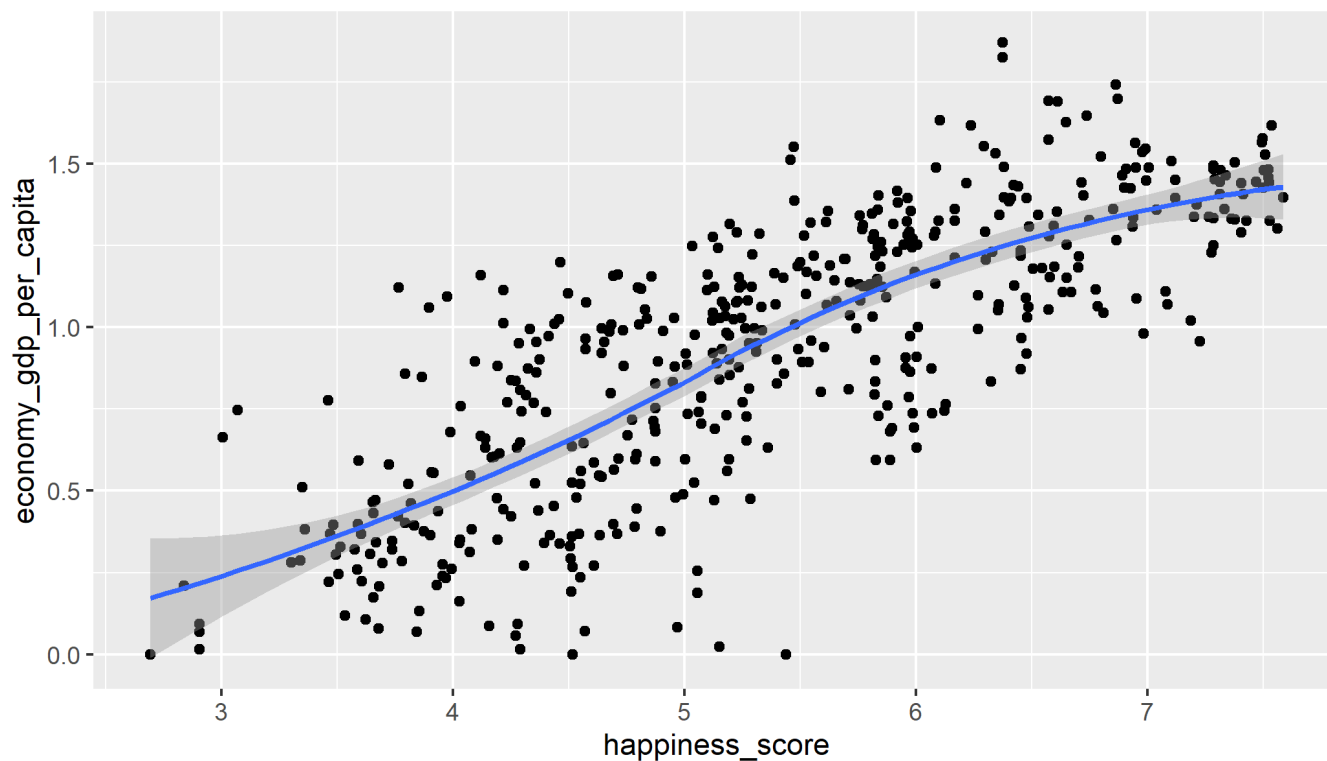


{ggplot2}: including more geoms



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

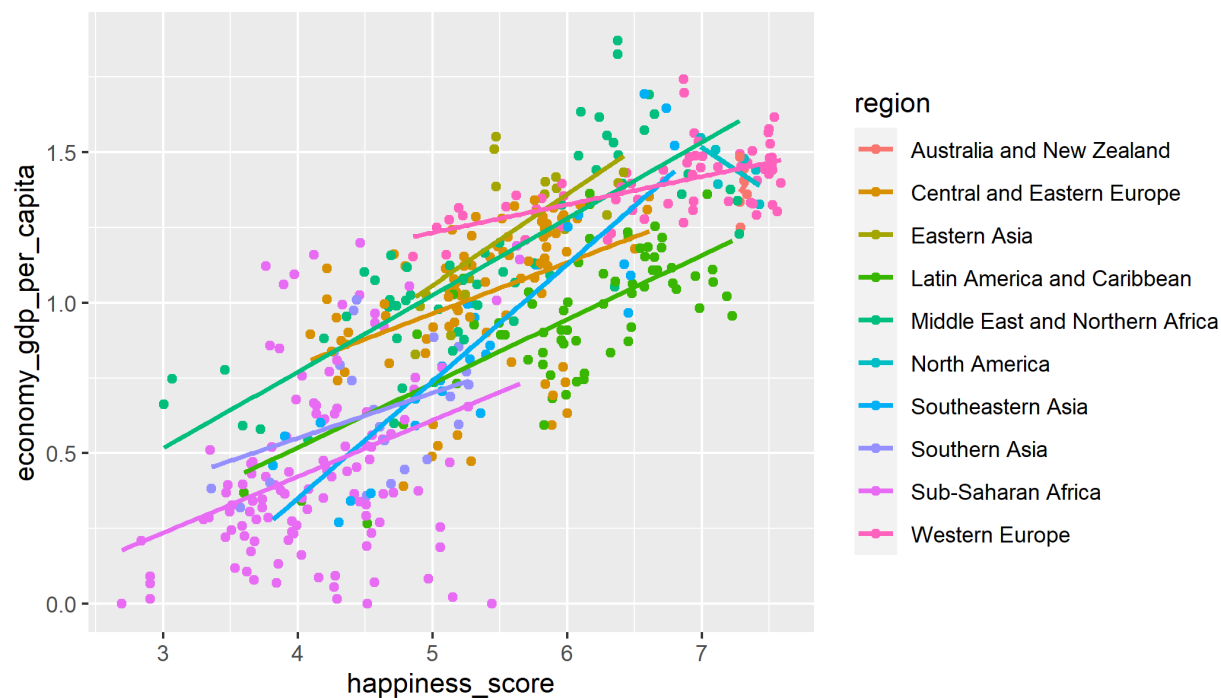
`geom_smooth()` using method = 'loess' and formula 'y ~ x'



{ggplot2}: including more geoms and colors



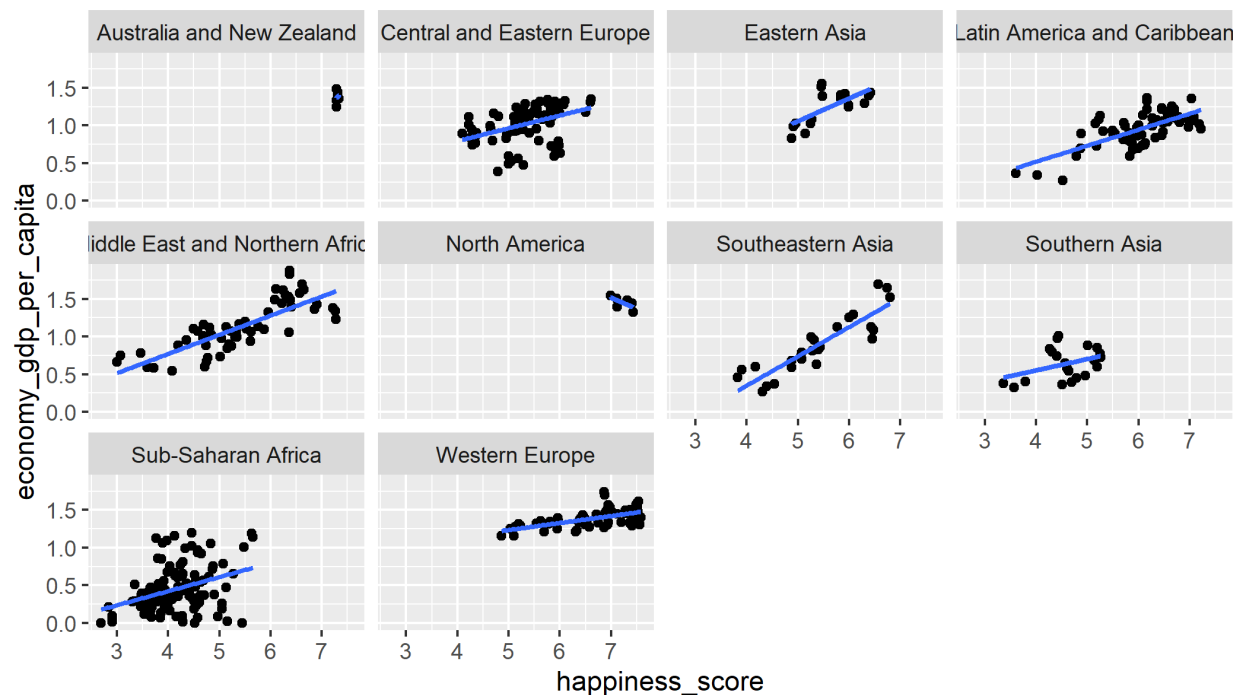
```
ggplot(data = whr_panel,  
       aes(x = happiness_score, y = economy_gdp_per_capita,  
           color = region)) +  
  geom_point() +  
  geom_smooth(aes(group = region),  
             method = "nls", formula = y ~ a * x + b, se = FALSE,  
             method.args = list(start = list(a = 0.1, b = 0.1)))
```



{ggplot2}: including more geoms and colors



```
ggplot(data = whr_panel,  
       aes(x = happiness_score, y = economy_gdp_per_capita)) +  
  geom_point() +  
  geom_smooth(aes(group = region),  
             method = "nls", formula = y ~ a * x + b, se = FALSE,  
             method.args = list(start = list(a = 0.1, b = 0.1))) +  
  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

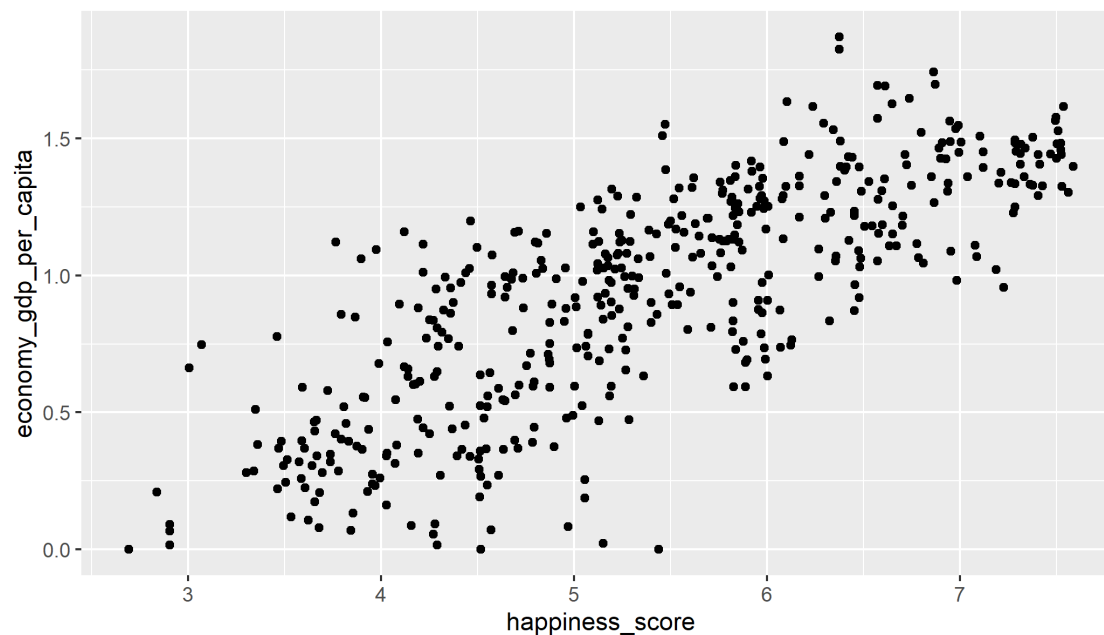
```
whr_panel %>%  
  mutate(  
    latam = ifelse(region == "Latin America and Caribbean", TRUE, FALSE)  
  ) %>%  
  ggplot(aes(x = happiness_score, y = economy_gdp_per_capita,  
            color = latam)) +  
  geom_point()
```

{ggplot2}: 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. Change our theme.
2. Add correct labels.
3. Add some annotations.
4. Modify our 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() +  
  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



Let's save the 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.

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

Saving a plot



Let's save the 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.

```
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)
```

References and recommendations

References and recommendations



- **Websites:**

- Interactive stuff : <http://www.htmlwidgets.org/>
- The R Graph Gallery: <https://www.r-graph-gallery.com/>
- Ggplot 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 and Garrett Grolemund

Thank you~