

R Markdown Sample

Your Name

September 26, 2024

Prepare for analyses

I typically use the first chunk, right after the YAML header, to establish general chunk options and load necessary libraries and packages for the analysis. Since these procedures are not particularly interesting to report, I always configure this initial chunk with `include=FALSE`.

Getting started

To produce PDF file, you need \TeX files.

- Easy way: Install the `tinytex` package: `install.packages("tinytex")`. Then run `tinytex::install_tinytex()`.
 - *Warning*: Installing ‘tinytext’ will likely take few minutes.

RMarkdown

- Save the following [Cheat Sheet](#) for RMarkdown.
- If any of you is looking for an general introduction for RMarkdown, I suggest you to check [Chapter 27](#) from Wickham and Grolemund (2017) - **R for Data Science**.
- If you want a more comprehensive guide, then check Xie et al. (2021) - **R Markdown: The Definitive Guide**.
- Another, more applied, resource is Xie et al. (2022) - **R Markdown Cookbook**.

Basic console output

To insert an R code chunk, you can type it manually or just press Chunks - Insert chunks or use the shortcut key.

- Short cut in Windows: `Ctrl + Alt + I`
- Short cut in macOS: `Cmd + Option + I`

This will produce the following code chunk:

The code chunk input and output is then displayed as follows:

```
##      x      y
## 1    1 1.586
## 2    2 2.709
## 3    3 2.891
## 4    4 3.547
## 5    5 5.606
## 6    6 4.182
## 7    7 7.630
## 8    8 7.724
## 9    9 8.716
## 10  10 9.081
```

In the prose/text area (outside the console area), you can call the console to report quantities using 3.547. The in-text console also allows you to embed other R functions to customize the output, such as 8.72. In the previous line, I used the function `round()` to round the ninth element or observation from the vector `y` to two digits.

- Frequently used chunk options

Option	Description
include	If FALSE, knitr will run the chunk but not include the chunk in the final document
echo	If FALSE, knitr will not display the code in the code chunk above it's results in the final document.
error	If FALSE, knitr will not display any error messages generated by the code.
message	If FALSE, knitr will not display any messages generated by the code.
warning	If FALSE, knitr will not display any warning messages generated by the code.

Recommendation for homework

Option	HW setting
include	TRUE
echo	TRUE
error	FALSE
message	FALSE
warning	FALSE

Basic markdown functionality

For those not familiar with standard [Markdown](#), the following may be useful. See the source code for how to produce such points. However, RStudio does include a Markdown quick reference button that adequately covers this material.

Dot Points

Simple dot points:

- Point 1
- Point 2
- Point 3

and numeric dot points:

1. Number 1
2. Number 2
3. Number 3

and nested dot points:

- A
 - A.1
 - A.2
- B
 - B.1
 - B.2

Equations

Equations are included by using LaTeX notation and including them either between single dollar signs (inline equations) or double dollar signs (displayed equations). If you hang around the Q&A site [CrossValidated](#) you'll be familiar with this idea.

There are inline equations such as $y_i = \alpha + \beta x_i + e_i$.

And displayed formulas:

$$\frac{1}{1 + \exp(-x)}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\begin{aligned} X &= (x + a)(x - b) \\ &= x(x - b) + a(x - b) \\ &= x^2 + x(a - b) - ab \end{aligned}$$

More info: [LaTeX wiki](#)

Tables

Tables can be included using the following notation

A	B	C
1	Male	Blue
2	Female	Pink

Or you want to show nice regression tables

```
Res1 <- lm(formula = y ~ x, data = df)

Res2 <- lm(formula = y ~ x + I(x^2), data = df)

# notice option: results='asis'

stargazer(Res1, Res2)
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com % Date and time: Thu, Sep 26, 2024 - 6:33:06 PM

```
#For html
#stargazer(Res1, Res2, type = "html")
```

More info: [Cheat Sheet](#)

If you want to create a fancy table from data.frame, you can use “pander”

```
Table <- df %>%
  mutate(z = if_else(y>5, 1, 0)) %>%
  t()
```

Table

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x 1.000 2.000 3.000 4.000 5.000 6.000 7.00 8.000 9.000 10.000
## y 1.586 2.709 2.891 3.547 5.606 4.182 7.63 7.724 8.716  9.081
## z 0.000 0.000 0.000 0.000 1.000 0.000 1.00 1.000 1.000  1.000
```

With pander

```
Table %>% pander(caption = "Fancy Table")
```

Table 5: Fancy Table

x	1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	---	----

y	1.586	2.709	2.891	3.547	5.606	4.182	7.63	7.724	8.716	9.081
z	0	0	0	0	1	0	1	1	1	1

With kableExtra

```
Table %>% kableExtra::kable()
```

x	1.000	2.000	3.000	4.000	5.000	6.000	7.00	8.000	9.000	10.000
y	1.586	2.709	2.891	3.547	5.606	4.182	7.63	7.724	8.716	9.081
z	0.000	0.000	0.000	0.000	1.000	0.000	1.00	1.000	1.000	1.000

Intro to ggplot (with dplyr)

We are going to use the gapminder dataset to perform exploratory data analysis with dplyr and ggplot.

```
# load data
df <- read.csv("data/gapminder.csv")

# check first observations
head(df)

##      country continent year lifeExp gdpPercap
## 1 Afghanistan      Asia  1952   28.801   779.4453
## 2 Afghanistan      Asia  1957   30.332   820.8530
## 3 Afghanistan      Asia  1962   31.997   853.1007
## 4 Afghanistan      Asia  1967   34.020   836.1971
## 5 Afghanistan      Asia  1972   36.088   739.9811
## 6 Afghanistan      Asia  1977   38.438   786.1134

# what is the average income (gdpPercap) per continent?
aggregate(df$gdpPercap, list(df$continent), mean)

##      Group.1      x
## 1    Africa 2193.755
## 2 Americas 7136.110
## 3     Asia 7902.150
## 4   Europe 14469.476
## 5  Oceania 18621.609
```

In class, we have seen the function aggregate to aggregate quantities of interest, such as group means or variances.

In the tidyverse environment, dplyr has several functions that perform similar functions to aggregate and even more that will be very handy to explore your panel datasets.

Table 4:

	<i>Dependent variable:</i>	
	y	
	(1)	(2)
x	0.876*** (0.084)	0.728 (0.398)
I(x ²)		0.013 (0.035)
Constant	0.551 (0.523)	0.847 (0.952)
Observations	10	10
R ²	0.931	0.932
Adjusted R ²	0.922	0.913
Residual Std. Error	0.765 (df = 8)	0.810 (df = 7)
F Statistic	108.076*** (df = 1; 8)	48.343*** (df = 2; 7)
<i>Note:</i> *p<0.1; **p<0.05; ***p<0.01		

`group_by` for `summarize()` **data**.

- `group_by` sets up the data frame so that you can apply functions to each group separately using dplyr functions.
 - You shall call the function through **code chaining** using the pipe operator: `%>%`
 - Short cut in Windows: Ctrl + Alt + M
 - Short cut in macOS: Cmd + Option + M
- I recommend you to use data frames of class `tibble` instead of `data.frame`. See example below

In the example below, I show the difference between base R coding with `data.frame()` class and piping or chaining coding `%>%` with `tibble()` class.

```
# Looking at your data in the console with df
head(as.data.frame(df),2)
```

```
##      country continent year lifeExp gdpPercap
## 1 Afghanistan      Asia 1952  28.801  779.4453
## 2 Afghanistan      Asia 1957  30.332  820.8530
```

Looking at your data in the console with tibble

```
df %>%  
  as_tibble() %>%  
  head(2)
```

```
## # A tibble: 2 x 5  
##   country      continent  year lifeExp gdpPercap  
##   <chr>         <chr>    <int>  <dbl>    <dbl>  
## 1 Afghanistan Asia      1952   28.8     779.  
## 2 Afghanistan Asia      1957   30.3     821.
```

- `group_by` is a function in the `dplyr` package that allows you to group a data frame by one or more variables.
- When you use `group_by`, `dplyr` creates a new object that is a **grouped** version of the original data frame.

```
df %>%  
  as_tibble() %>%  
  group_by(country) %>%  
  head()
```

```
## # A tibble: 6 x 5  
## # Groups:   country [1]  
##   country      continent  year lifeExp gdpPercap  
##   <chr>         <chr>    <int>  <dbl>    <dbl>  
## 1 Afghanistan Asia      1952   28.8     779.  
## 2 Afghanistan Asia      1957   30.3     821.  
## 3 Afghanistan Asia      1962   32.0     853.  
## 4 Afghanistan Asia      1967   34.0     836.  
## 5 Afghanistan Asia      1972   36.1     740.  
## 6 Afghanistan Asia      1977   38.4     786.
```

`summarize()` is a function in the `dplyr` package that allows you to apply functions to each group in a grouped data frame and return a summary result.

```
df %>%  
  as_tibble() %>%  
  group_by(continent) %>%  
  summarize(gdp=mean(gdpPercap,na.rm=T),  
            life=mean(lifeExp,na.rm=T))
```

```
## # A tibble: 5 x 3  
##   continent      gdp  life  
##   <chr>         <dbl> <dbl>  
## 1 Africa      2194.  48.9  
## 2 Americas   7136.  64.7  
## 3 Asia       7902.  60.1
```

```
## 4 Europe      14469.   71.9
## 5 Oceania     18622.   74.3
```

You can comment your code after pipping it!

```
df %>%
  # transform the df as tibble
  as_tibble() %>%
  # group the data by continents
  group_by(continent) %>%
  # summarize group means
  summarize(gdp=mean(gdpPercap,na.rm=T),
            life=mean(lifeExp,na.rm=T)) %>%
  # generate a table output
  kableExtra::kable()
```

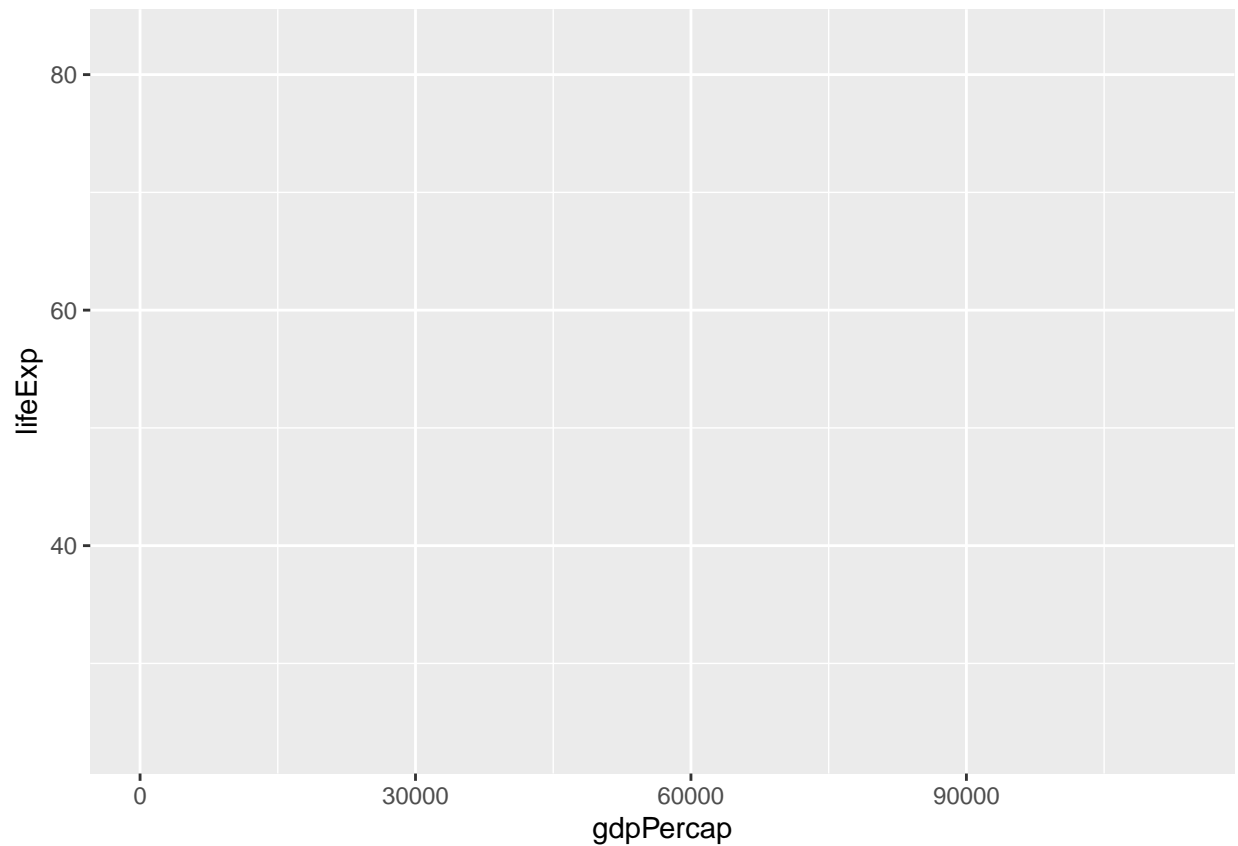
continent	gdp	life
Africa	2193.755	48.86533
Americas	7136.110	64.65874
Asia	7902.150	60.06490
Europe	14469.476	71.90369
Oceania	18621.609	74.32621

Grammar of graphics: ggplot2

- ggplot2: A *layered* grammar of graphics ([Wickham 2009](#)).
 - Build a graphic from multiple layers; each consists of some geometric objects or transformation
 - Use + to stack up layers
- *What* data do you want to visualize?
 - ggplot(data = ...)
- *How* are variables mapped to specific aesthetic attributes?
 - aes(... = ...)
 - * positions (x, y), shape, colour, size, fill, alpha, linetype, label...
 - * If the value of an attribute do not vary w.r.t. some variable, don't wrap it within aes(...)
- *Which* geometric shapes do you use to represent the data?
 - geom_{}:
 - * geom_point, geom_line, geom_ribbon, geom_polygon, geom_label...

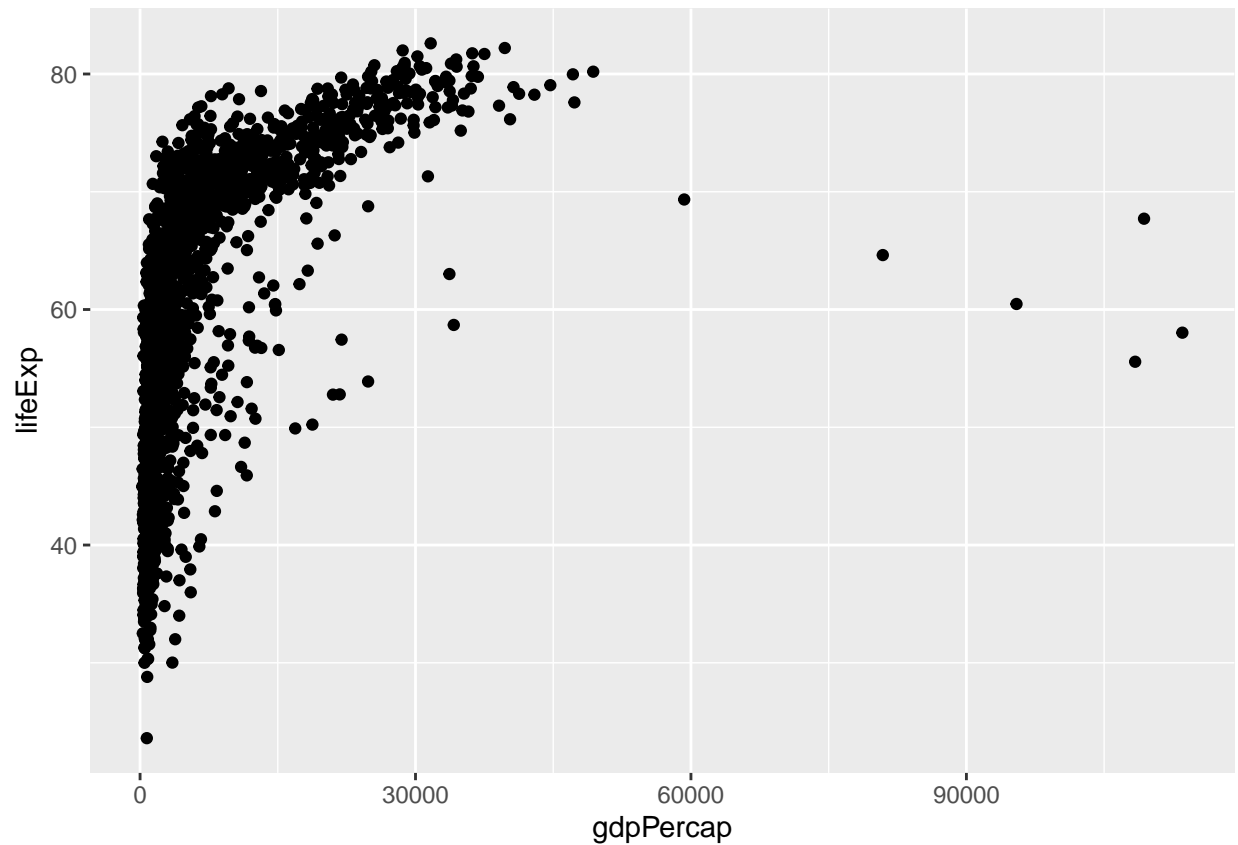
Step 1: Define a basic ggplot object with x and y aesthetics

```
ggplot(data=df,  
       aes(x=gdpPercap,  
           y=lifeExp))
```



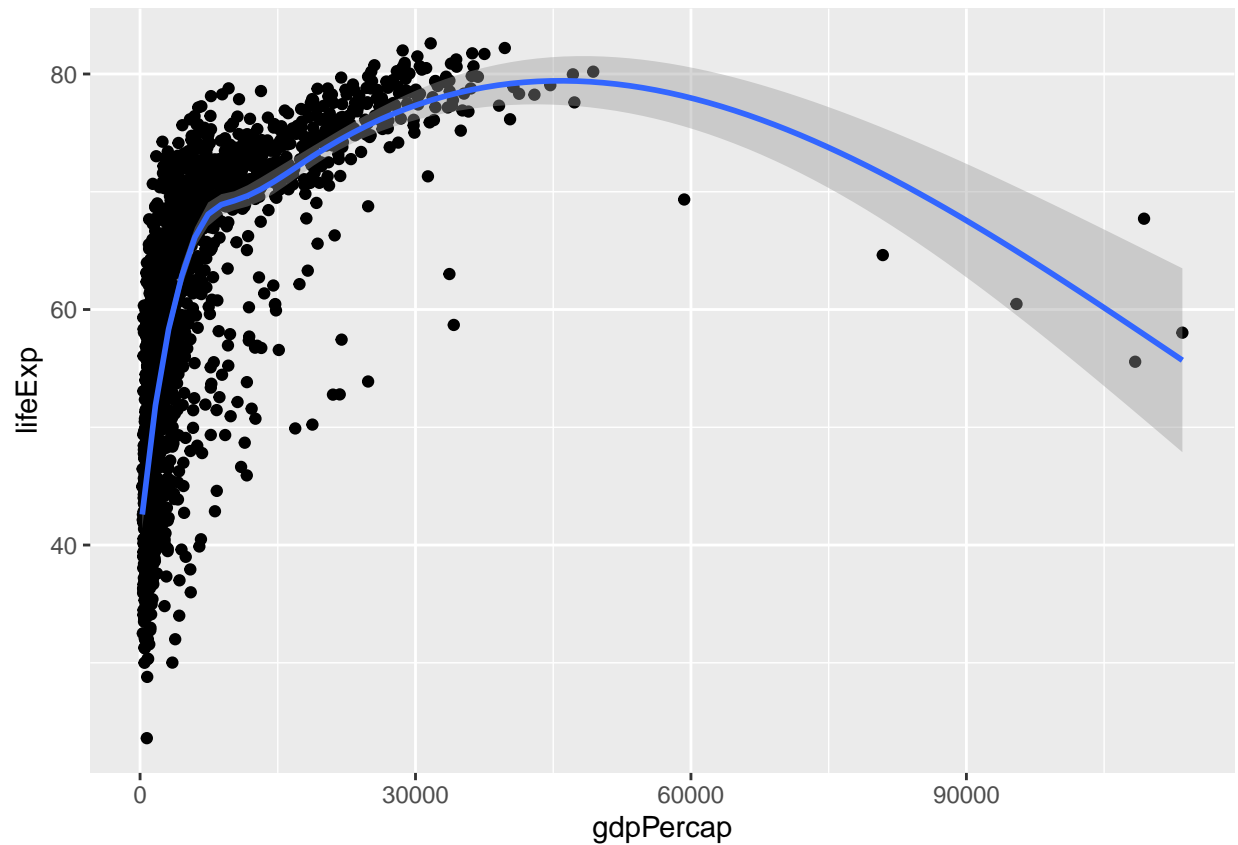
Step 2: Define a geometric shape

```
df %>%  
  ggplot(# aesthetics  
        aes(x=gdpPercap,  
            y=lifeExp)) +  
  # geometry  
  geom_point()
```



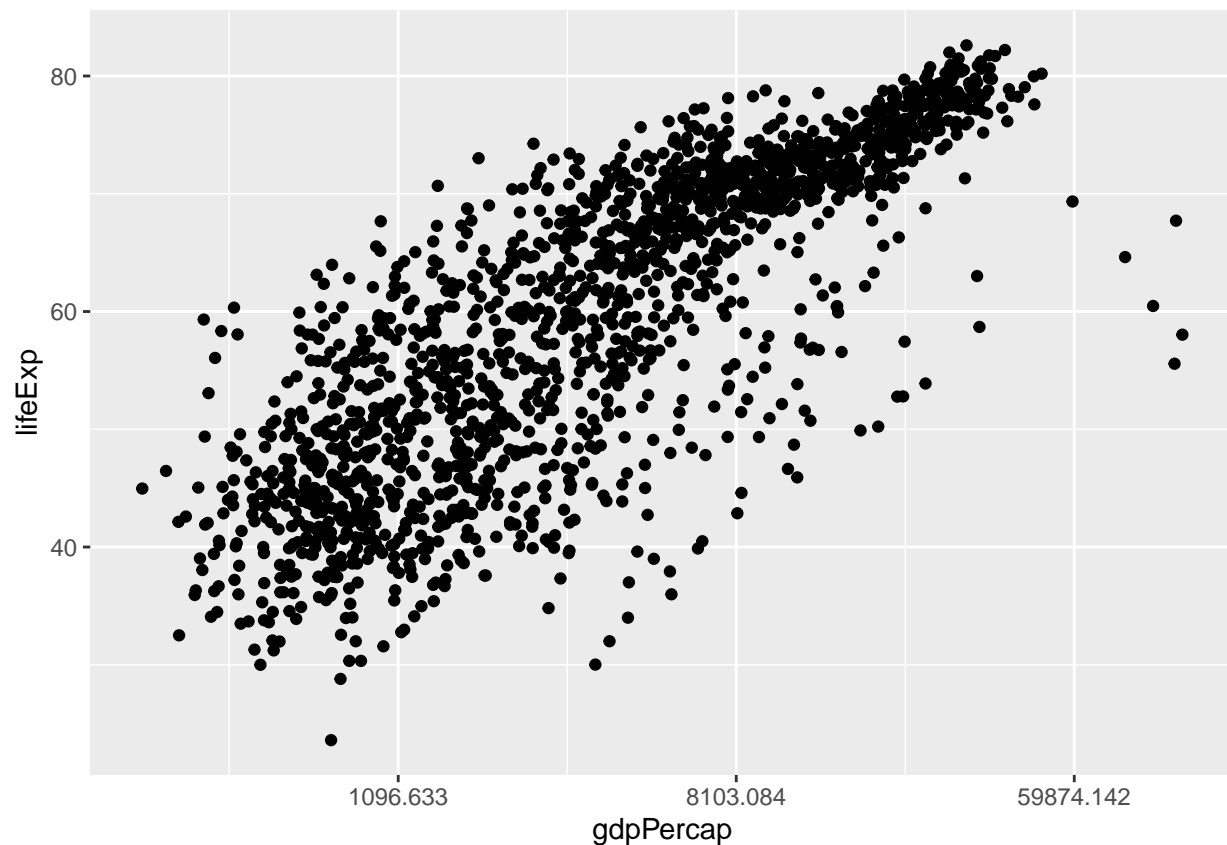
Note: we are not limited to have a single geometric form

```
df %>%  
  ggplot(# aesthetics  
    aes(x=gdpPerCap,  
        y=lifeExp)) +  
  # geometry point  
  geom_point() +  
  # geometry loess fit  
  geom_smooth()
```



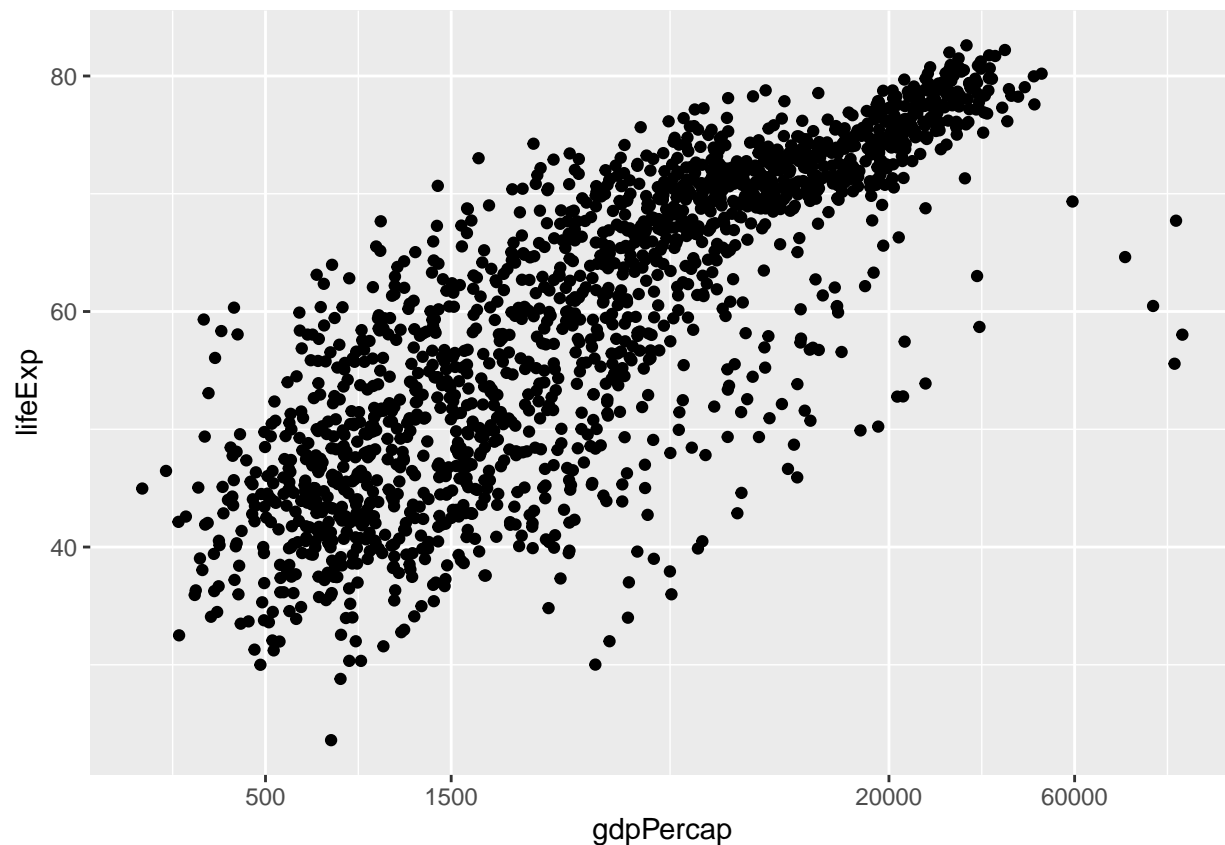
Step 3: modify coordinates or other arguments to customize the visual.

```
df %>%
  ggplot(# aesthetics
    aes(x=gdpPerCap,
        y=lifeExp)) +
  # geometry point
  geom_point() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log")
```



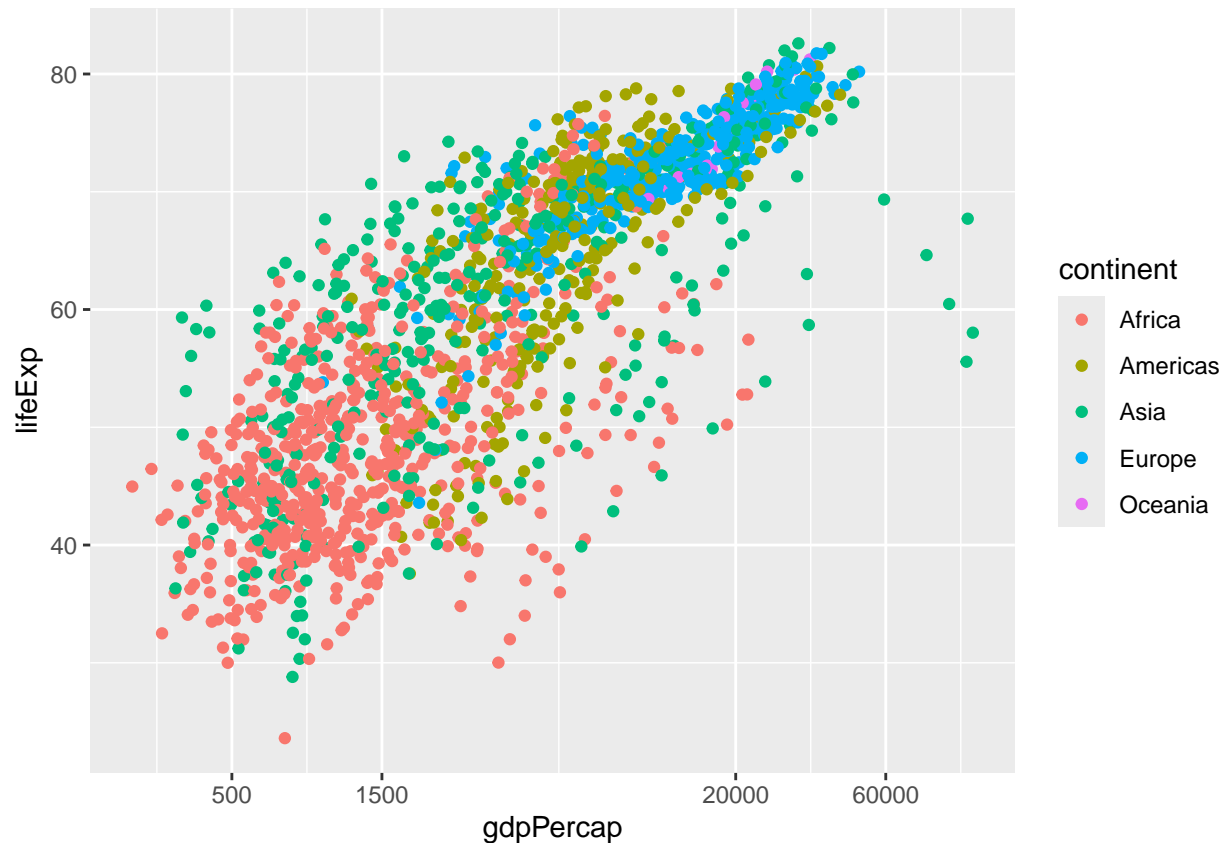
Step 3: scale or transform coordinates or other arguments to customize the visual.

```
df %>%
  ggplot(# aesthetics
    aes(x=gdpPerCap,
        y=lifeExp)) +
  # geometry point
  geom_point() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log",
    # introduce breaks
    breaks = c(500,
               1500,
               20000,
               60000))
```



Now, I am going to include a new aesthetic: **color**.

```
df %>%
  ggplot(# aesthetics
    aes(x=gdpPercap,
        y=lifeExp,
        # introduce color aesthetics
        color=continent)) +
  # geometry point
  geom_point() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log",
    # introduce breaks
    breaks = c(500,
               1500,
               20000,
               60000))
```



Step 5: Add x and y axis labels and a title

```
df %>%
  ggplot(# aesthetics
    aes(x=gdpPerCap,
        y=lifeExp,
        # introduce color aesthetics
        color=continent)) +
  # geometry point
  geom_point() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log",
    # introduce breaks
    breaks = c(500,
               1500,
               20000,
               60000)) +
  # Tittle
  labs(x="GDP per capita (log)",
       y="Life Expectancy",
       title="Income and life expectancy in the continents")
```



Note: sometimes, a geom can be fitted but the visual is not appealing.

```
df %>%
  ggplot(# aesthetics
    aes(x=gdpPerCap,
        y=lifeExp,
        # introduce color aesthetics
        color=continent)) +
  # geometry point
  geom_point() +
  # geometry smooth
  geom_smooth() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log",
    # introduce breaks
    breaks = c(500,
                1500,
                20000,
                60000)) +
  # Tittle
```

```
labs(x="GDP per capita (log)",
     y="Life Expectancy",
     title="Income and life expectancy in the continents")
```



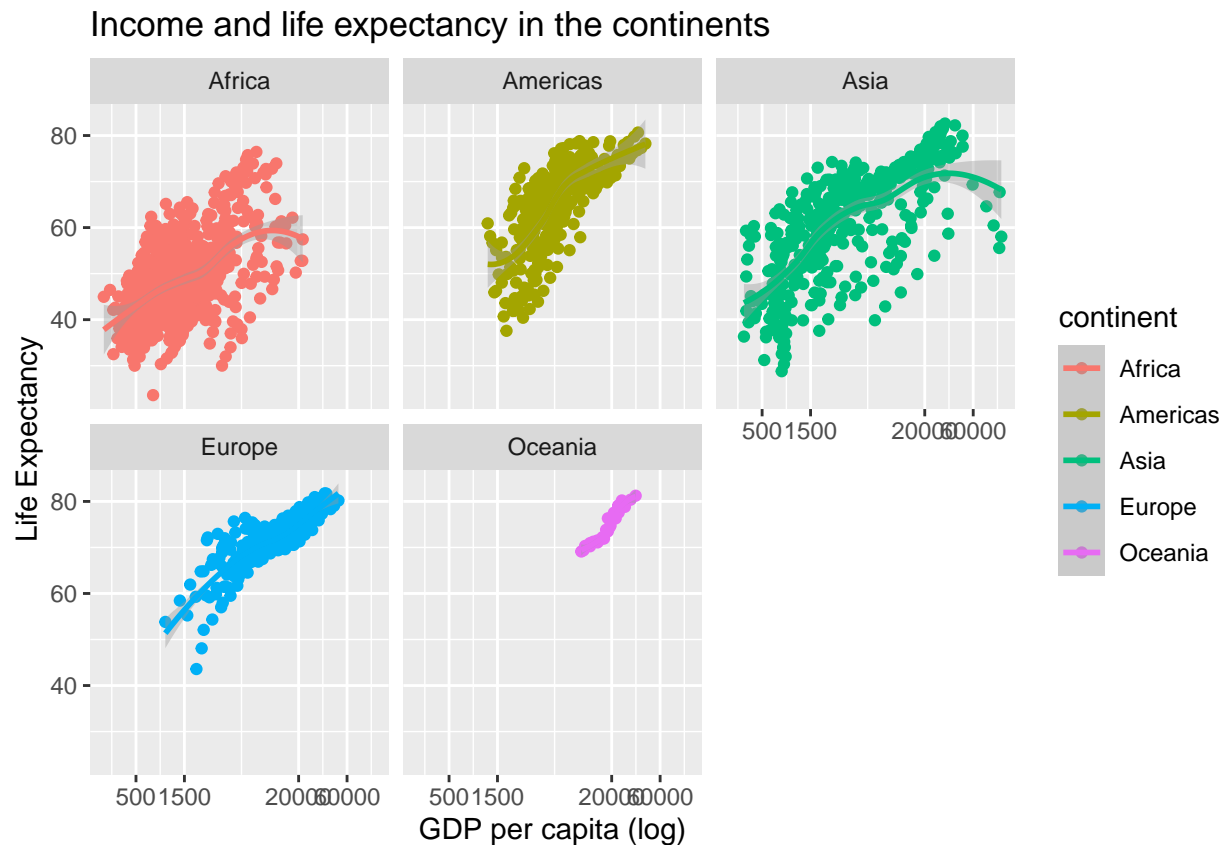
Step 6: create small multiples

```
df %>%
  ggplot(# aesthetics
        aes(x=gdpPerCap,
            y=lifeExp,
            # introduce color aesthetics
            color=continent)) +
  # geometry point
  geom_point() +
  # geometry smooth
  geom_smooth() +
  # Transform the scale of X variable
  scale_x_continuous(trans = "log",
                    # introduce breaks
                    breaks = c(500,
                              1500,
```

```

20000,
60000)) +
# Tittle
labs(x="GDP per capita (log)",
     y="Life Expectancy",
     title="Income and life expectancy in the continents") +
# Plot panels with small multiples
facet_wrap(~continent)

```



You can use the argument `alpha` to set the degree of transparency of a specific geom.

```

df %>%
  ggplot(# aesthetics
        aes(x=gdpPerCap,
            y=lifeExp,
            # introduce color aesthetics
            color=continent)) +
  # geometry point
  geom_point(alpha=0.1) +
  # geometry smooth
  geom_smooth() +
  # Transform the scale of X variable

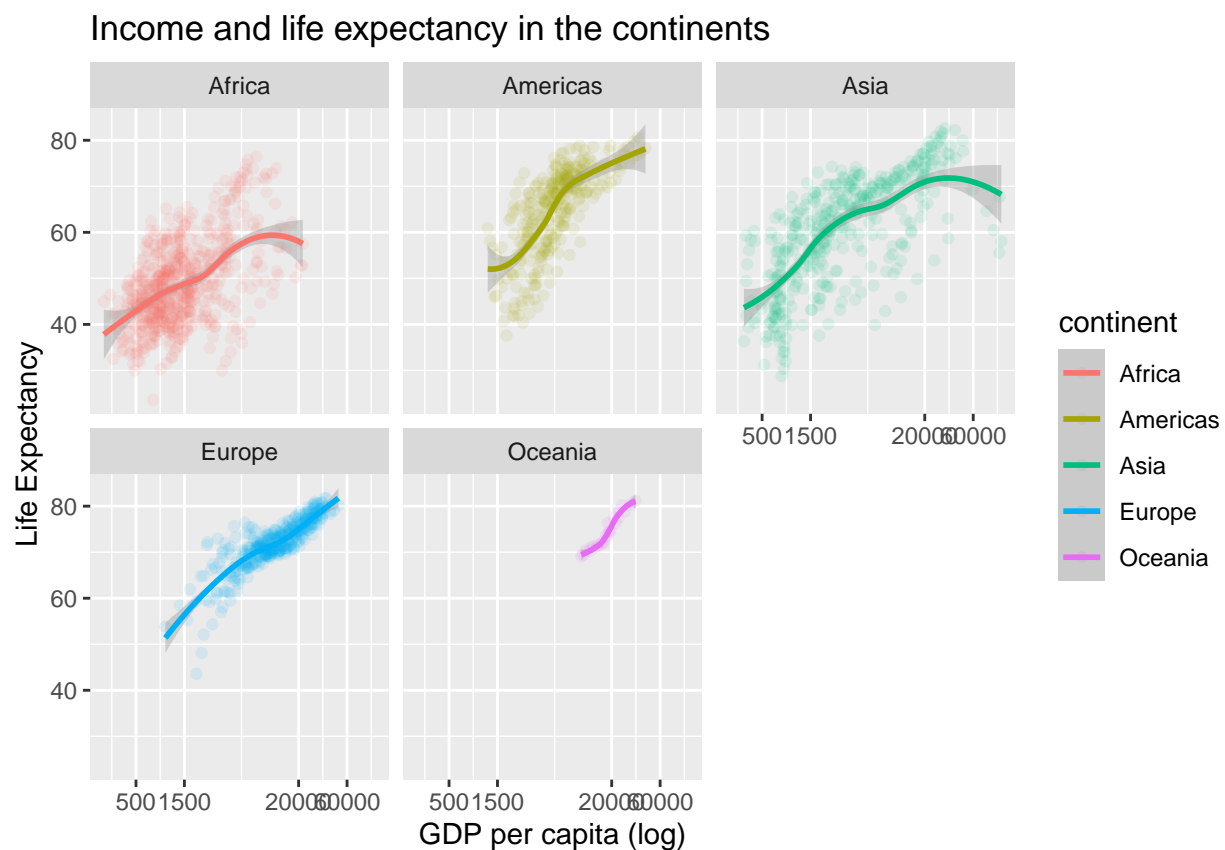
```

```

scale_x_continuous(trans = "log",
  # introduce breaks
  breaks = c(500,
             1500,
             20000,
             60000)) +

# Tittle
labs(x="GDP per capita (log)",
     y="Life Expectancy",
     title="Income and life expectancy in the continents") +
# Plot panels with small multiples
facet_wrap(~continent)

```



Step 6: modify the theme_.

```

df %>%
  ggplot(# aesthetics
    aes(x=gdpPerCap,
        y=lifeExp,
        # introduce color aesthetics
        color=continent)) +

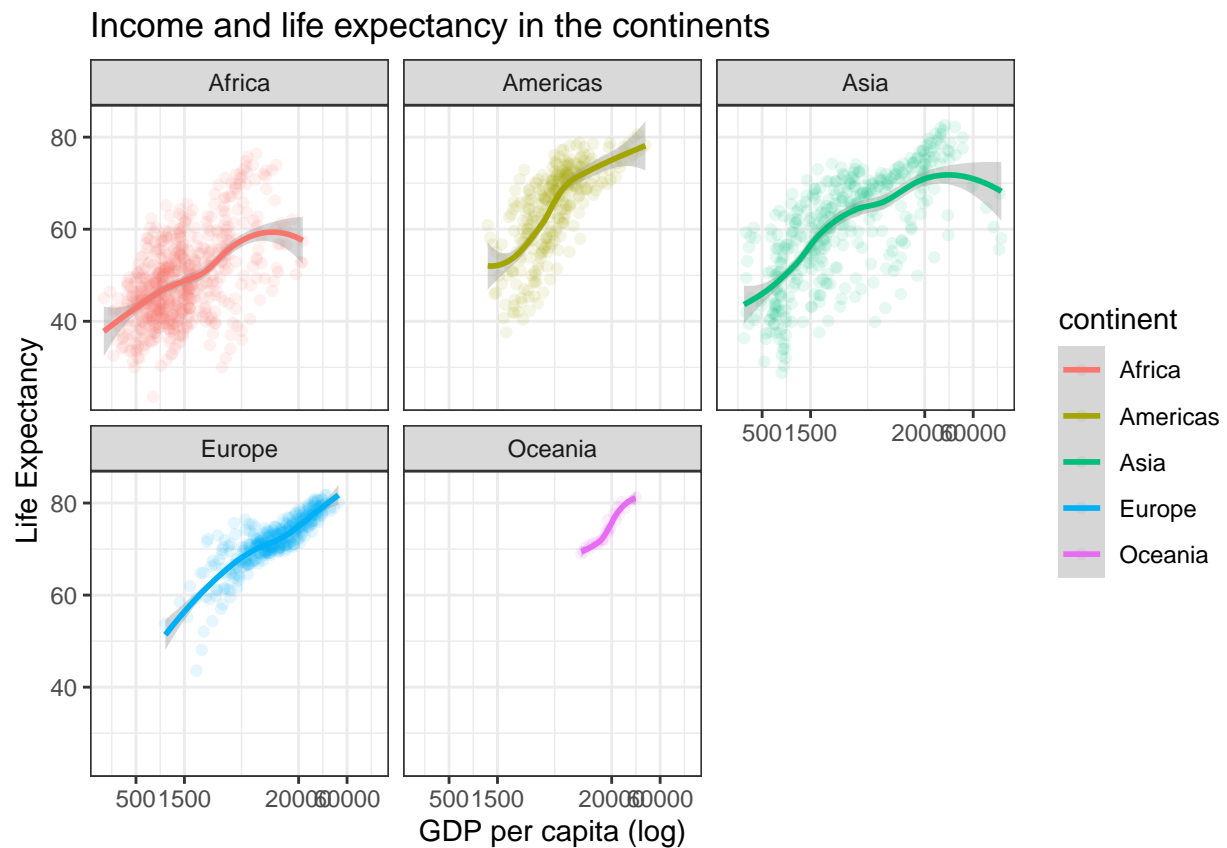
```

```

# geometry point
geom_point(alpha=0.1) +
# geometry smooth
geom_smooth() +
# Transform the scale of X variable
scale_x_continuous(trans = "log",
                    # introduce breaks
                    breaks = c(500,
                               1500,
                               20000,
                               60000)) +

# Tittle
labs(x="GDP per capita (log)",
     y="Life Expectancy",
     title="Income and life expectancy in the continents") +
# Plot panels with small multiples
facet_wrap(~continent) +
# theme for the plot
theme_bw()

```



Grammar of graphics in ggplot2

- ggplot2 is a powerful tool for creating professional visualizations.
- Search on the internet or ask ChatGPT for help with specific plot types using keywords based on geometries, such as **line plots**, **histograms**, **boxplots**, **coefficient plots**, etc.