Lab 3: Data visualization

Data visualizations are graphics that represent quantitative or qualitative data. In PSTAT100 you'll be using the python visualization library Altair, which is built around the pandas dataframe. Altair creates visualizations by mapping columns of a dataframe to the various elements of a graphic: axes, geometric objects, and aesthetics.

Visualizations are essential tools in exploratory analysis as well as presentation. They can help an analyst identify and understand structure and patterns in a dataset at a high level and provide guidance for model development. They can be used to check assumptions and visualize model outputs. And they can be an effective means for conveying results to a general audience.

Constructing effective visualization is usually an iterative process: plot-think-revise-repeat. In exploratory visualization often it is useful to produce a large quantity of plots in order to look at data from multiple angles; in this context, speed is helpful and details can be overlooked. By contrast, presentation graphics are typically highly refined versions of one or two exploratory plots that serve as communication tools; developing them involves attention to fine detail.

Objectives

In this lab you'll become familiar with the basic functionality of Altair – the basic kinds of graphics it generates and how to construct these graphics from a dataframe – and get a taste of the process of constructing good graphics.

In Altair, plots are constructed by: 1. creating a *chart* 2. specifying *marks and encodings* 3. adding various *aesthetics*, and 4. resolving display issues through *customization*.

Technical tutorial. You'll get an introduction to each of these steps: * Creating a chart object from a dataframe * Encodings: mapping columns to graphical elements * Marks: geometric objects displayed on a plot (*e.g.*, points, lines, polygons) * Aesthetics: display attributes of geometric objects (*e.g.*, color, shape, transparency) * Customization: adjusting axes, labels, scales.

Visualization process. In addition, our goal is to model for you the process of constructing a good visualization through iterative revisions. * Identifying and fixing display problems * Discerning informative from non-informative graphical elements * Designing efficient displays

Background: elements of graphics

To understand why Altair (and other common visualization libraries like ggplot in R) works the way it does, it is helpful to have a framework for characterizing the elements of a graphic. Broadly speaking, graphics consist of sets of **axes**, **geometric objects** plotted on those axes, **aesthetic attributes** of geometric objects, and **text** used to label axes, objects, or aesthetics.

Altair constructs plots by mapping columns of a dataframe to each of these elements. A set of such mappings is referred to as an *encoding*, and the elements of a graphic that a dataframe column can be mapped to are called *encoding channels*.

Axes

Axes establish a reference system for a graphic: they define a space within which the graphic will be constructed. Usually these are coordinate systems defined at a particular scale, like Cartesian coordinates on the region (0, 100) x (0, 100), or polar coordinates on the unit circle, or geographic coordinates for the globe.

In Altair, axes are automatically determined based on encodings, but are customizable to an extent.

Geometric objects

Geometric objects are any objects superimposed on a set of axes: points, lines, polygons, circles, bars, arcs, curves, and the like. Often, visualizations are characterized according to the type of object used to display data – for example, the *scatterplot* consists of points, a *bar plot* consists of bars, a *line plot* consists of one or more lines, and so on.

In Altair, geometric objects are called *marks*.

Aesthetic attributes

The word 'aesthetics' is used in a variety of ways in relation to graphics; you will see this in your reading. For us, 'aesthetic attirbutes' will refer to attributes of geometric objects like color. The primary aesthetics in statistical graphics are color, opacity, shape, and size.

In Altair, aesthetic attributes are called *mark properties*.

Text

Text is used in graphics to label axes, geometric objects, and legends for aesthetic mappings. Text specification is usually a step in customization for presentation graphics, but often skipped in exploratory graphics. Carefully chosen text is very important in this context, because it provides essential information that a general reader needs to interpret a plot.

In Altair, text is usually controlled as part of encoding specification.

Data import: GDP and life expectancy

We'll be illustrating Altair functionality and visualization process using a dataset comprising observations of life expectancies at birth for men, women, and the general population, along with GDP per capita and total population for 158 countries at approximately five-year intervals from 2000 to 2019.

- Observational units: countries.
- Variables: country, year, life expectancy at birth (men, women, overall), GDP per capita, total population, region (continent), and subregion.

The data come from merging several smaller datasets, mostly collected from World Bank Open Data. The result is essentially a convenience sample, but descriptive analyses without inference are nonetheless interesting and suggestive.

Your focus won't be on acquainting yourself with the data carefully or on tidying. The cell below imports and merges component datasets.

```
# Load necessary libraries
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(readr)

#setwd('/Users/irene/Desktop')
# Import and format country regional information
countryinfo <- read_csv('data/country-info.csv') %>%
  select(3, 6, 7) %>%
  rename('Country Code' = 'alpha-3')
```

```
Rows: 249 Columns: 11
```

```
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (11): name, alpha-2, alpha-3, country-code, iso_3166-2, region, sub-regi...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#View(countryinfo)
```

```r
# Import and format GDP per capita
gdp <- read_csv('data/gdp-per-capita.csv', locale = locale(encoding = 'latin1')) %>%
  select(-'Indicator Name', -'Indicator Code') %>%
  pivot_longer(cols = -c('Country Name', 'Country Code'),
               names_to = 'Year',
               values_to = 'GDP_per_capita') %>%
  mutate(Year = as.integer(Year))
```

```
Rows: 264 Columns: 65
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): Country Name, Country Code, Indicator Name, Indicator Code
dbl (60): 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, ...
lgl  (1): 2020

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Import and format life expectancies
life <- read_csv('data/life-expectancy.csv') %>%
  rename(Life_Expectancy = All,
         Male_Life_Expectancy = Male,
         Female_Life_Expectancy = Female)
```

```
Rows: 733 Columns: 5
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): Country Name
dbl (4): Year, All, Male, Female

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#View(gdp)
#View(life)
```

```r
# Import population data
pop <- read_csv('data/population.csv', locale = locale(encoding = 'latin1')) %>%
  pivot_longer(cols = -c('Country Name', 'Country Code'),
               names_to = 'Year',
               values_to = 'Population') %>%
  mutate(Year = as.integer(Year)) %>%
  select(-c('Country Name'))
```

```
Rows: 264 Columns: 63
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): Country Name, Country Code
dbl (60): 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, ...
lgl  (1): 2020

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#View(pop)
```

```r
# Merge data
merge1 <- left_join(life, gdp, by = c('Country Name', 'Year'))
#View(merge1)
```

```r
merge2 <- left_join(merge1, countryinfo, by = 'Country Code')
merge3 <- left_join(merge2, pop, by = c('Country Code', 'Year'))

# Final data
data <- merge3 %>%
  drop_na() %>%
  select(-'Country Code')

# View the first few rows of the final dataset
head(data)
```

```
# A tibble: 6 x 9
  `Country Name`  Year Life_Expectancy Male_Life_Expectancy
  <chr>          <dbl>           <dbl>                <dbl>
1 Afghanistan     2019            63.2                 63.3
2 Afghanistan     2015            61.7                 61
3 Afghanistan     2010            59.9                 59.6
4 Albania         2019            78                   76.3
5 Albania         2015            77.8                 76.1
6 Albania         2010            76.2                 74.2
# i 5 more variables: Female_Life_Expectancy <dbl>, GDP_per_capita <dbl>,
#   region <chr>, `sub-region` <chr>, Population <dbl>
```

Life expectancy and GDP per capita

Here you'll see how marks and encodings work in a basic sense, along with some examples of how to adjust encodings.

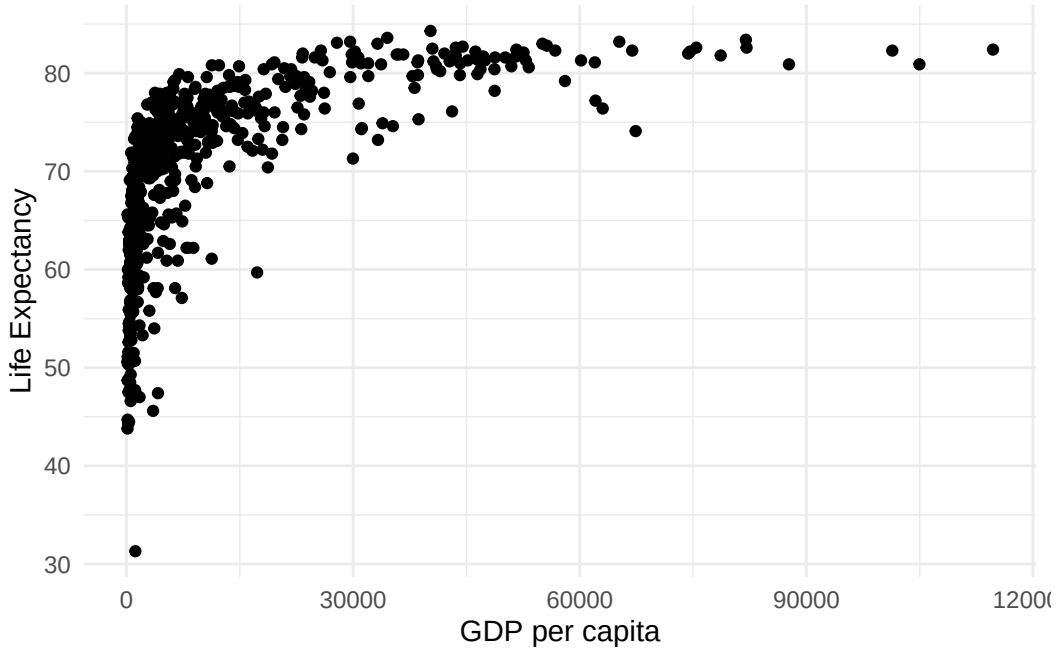
Basic scatterplots

The following code constructs a scatterplot of life expectancy at birth against GDP per capita; each point corresponds to one country in one year. The syntax works as follows:

ggplot(data, aes(…)) begins by constructing a ggplot object from the dataframe and specifies the aesthetic mappings (e.g., x-axis, y-axis, color, size). The result is passed to geom_point(), which adds points (circles) to the plot. Additional layers, such as scales (scale_x_log10() for

log scaling), themes (theme_minimal() for a clean appearance), and guides, are added to refine the visualization.

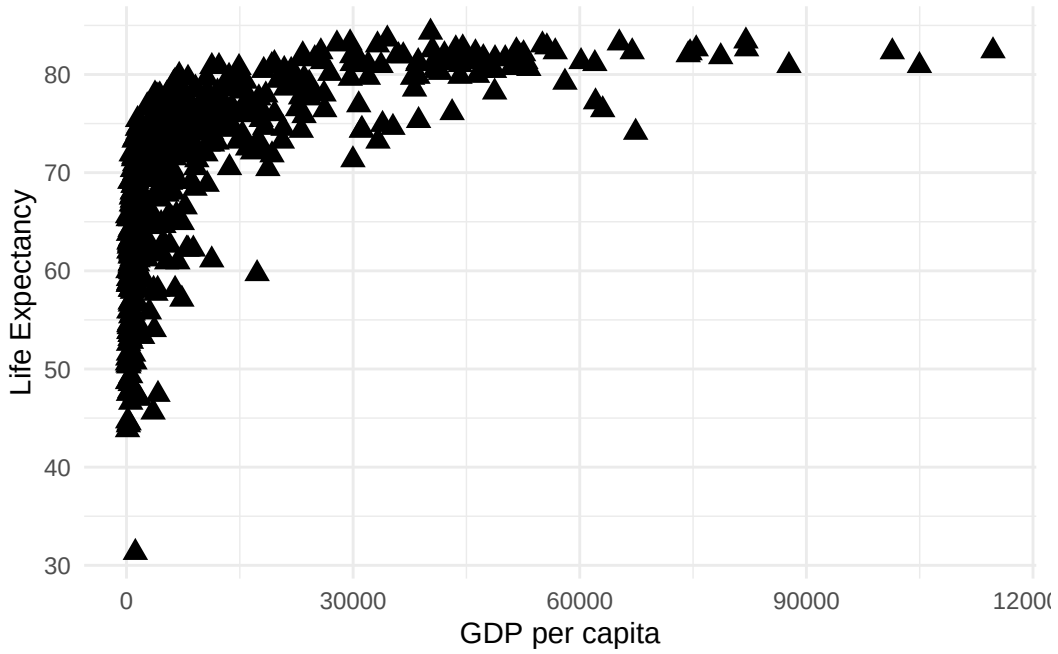
```
# Load necessary library
library(ggplot2)

# Basic scatterplot in R using ggplot2
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point() +
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy")
```



Question 1: Different marks

The code block below is a copy of the previous scatterplot. Have a look at the documentation on geom functions in ggplot2 for a list of possible geometries (marks) you can use, such as geom_point(), geom_line(), or geom_jitter(). Try out a few alternatives to see what they look like. Once you're satisfied, set the geometry back to geom_point() to use points.

```
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point(shape = 17, size = 3) +  # Triangle shape
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy")
```



Question 2: Mark properties

What is the difference between points and circles, according to the documentation?

Type your answer here, replacing this text. In ggplot2, points are a general geometric shape used in scatterplots, created by geom_point(). By default, points are rendered as circles, but their appearance can be customized using the shape aesthetic.

Points in geom_point(): Points are rendered as circular shapes by default. You can change the shape of points using the shape aesthetic, which supports a variety of symbols. Available shapes include circles, squares, triangles, crosses, and others.

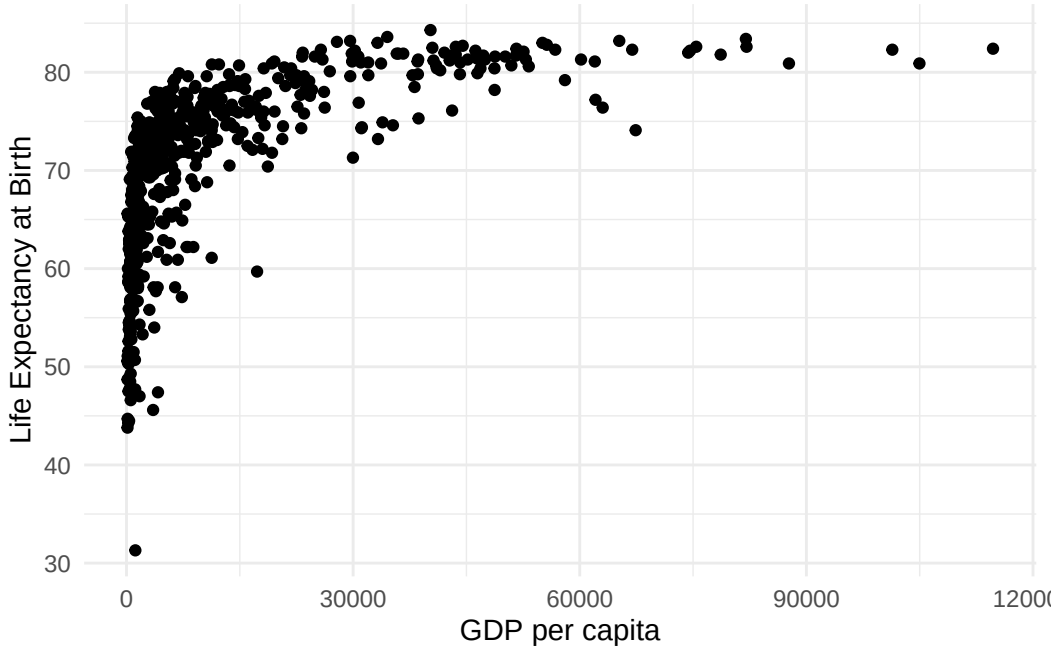
Circles in geom_point(): Circles are the default shape for points when no shape aesthetic is specified. You can explicitly use the shape corresponding to a circle (shape 16 in ggplot2's shape palette).

Adjusting Axis Scales, Labels, and Limits in R with ggplot2

In ggplot2, axis adjustments are made using functions like scale_x_*()* *and* *scale_y_*(). These functions allow you to modify:

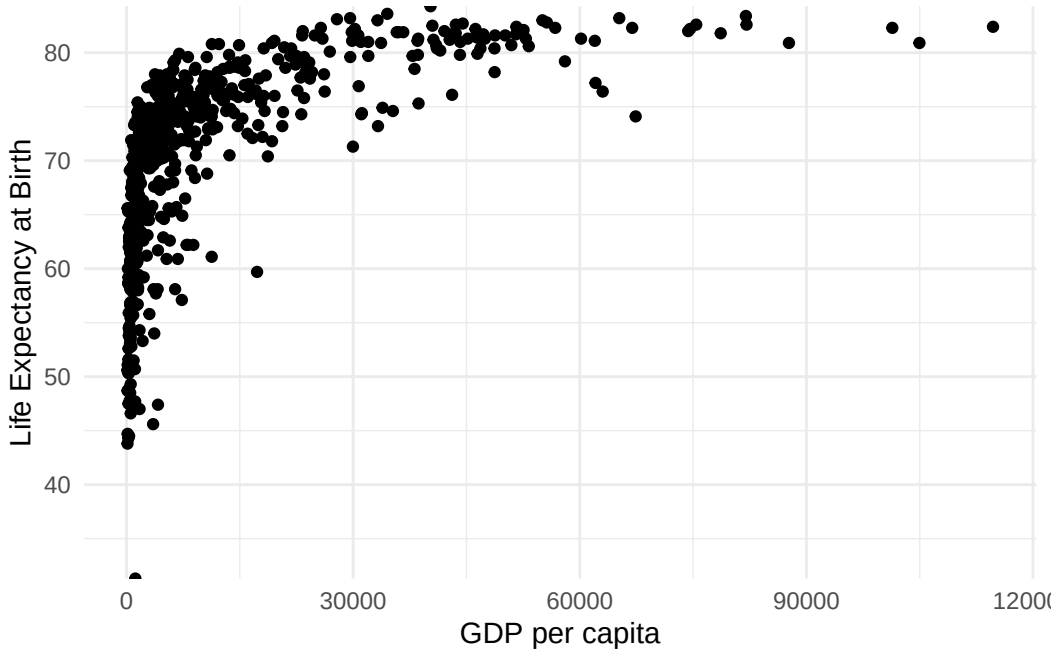
Scale types (e.g., logarithmic, continuous, categorical). Labels (e.g., axis titles). Limits (e.g., ranges for the axis). Ticks (e.g., number or formatting of tick marks).

```r
# change axis label
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point() +
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy at Birth")  # Custom y-axis title
```



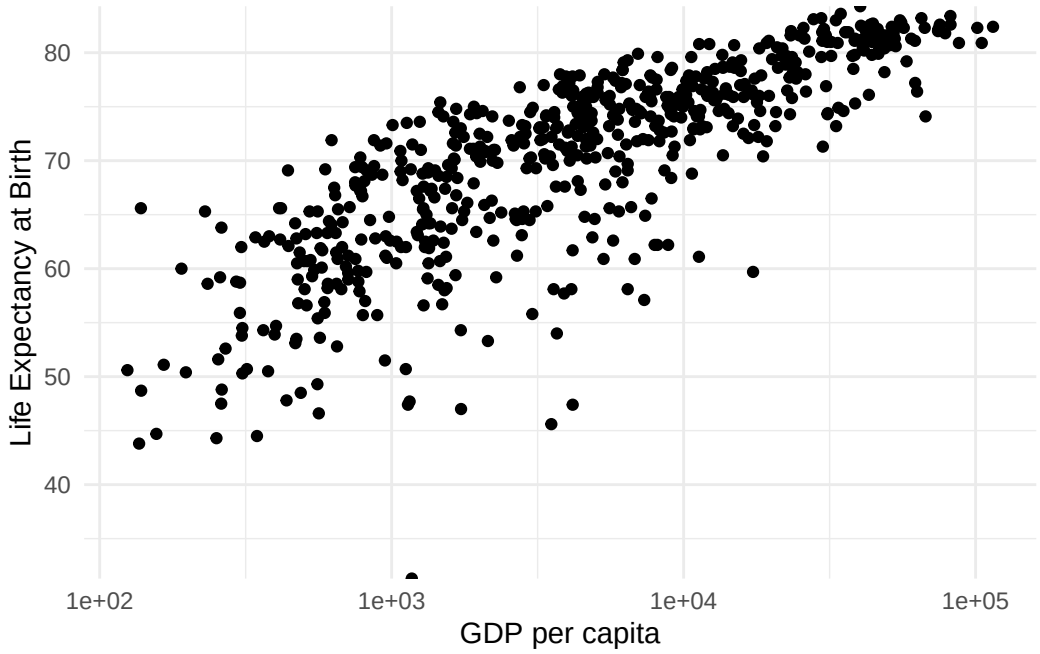
You can adjust the y-axis to start at a more appropriate value (e.g., 30 instead of 0) by setting the limits argument within the scale_y_continuous() function. This helps remove unnecessary whitespace and improves visualization.

```r
# don't start y axis at zero
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point() +
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy at Birth") +
  scale_y_continuous(expand = c(0, 0), limits = c(min(data$Life_Expectancy), NA))  # Remove
```



You can achieve this by applying a logarithmic transformation to the x-axis using scale_x_log10(). This transformation helps spread out data points clustered near 0 on the x-axis, improving the visibility of patterns.

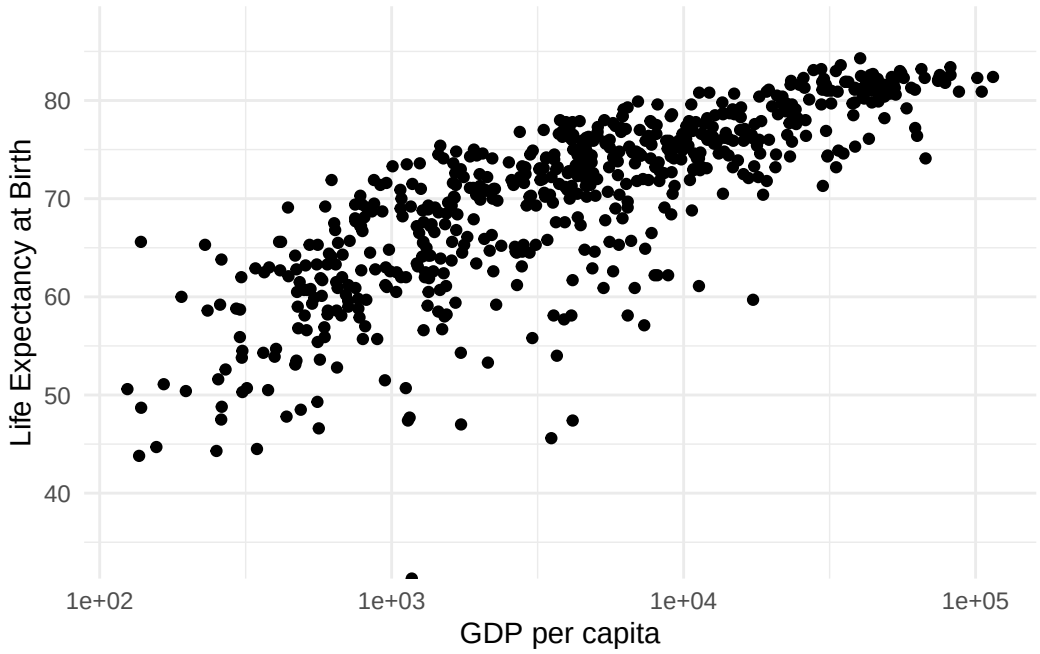
```
# log scale for x axis
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point() +
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy at Birth") +
  scale_x_log10() +  # Apply log scale to x-axis
  scale_y_continuous(expand = c(0, 0), limits = c(min(data$Life_Expectancy), NA))  # Prevent
```



Question 3: Changing axis scale

Try a different scale by modifying the scale type using the scale_*() function in the code below. Look at the ggplot2 documentation for a list of possible scale options and their parameters.

```
# try another axis scale
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy)) +
  geom_point() +
  theme_minimal() +
  labs(x = "GDP per capita", y = "Life Expectancy at Birth") +
  scale_x_log10() +  # Apply log scale to x-axis
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.1)))
```



Using aesthetic attributes to display other variables

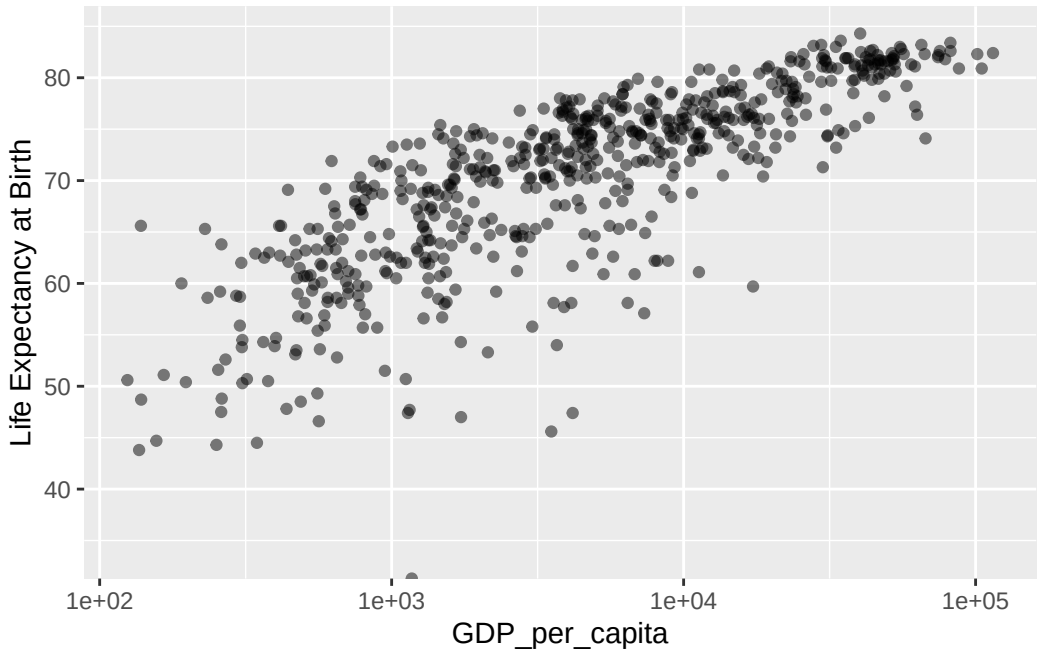
You can adjust the transparency of points globally by setting the alpha argument inside the geom_point() function. This is especially useful for scatterplots with tightly clustered points, as it reduces overplotting and reveals underlying patterns.

```
#View(data)
```

```
# change opacity globally to fixed value
library(scales)
```

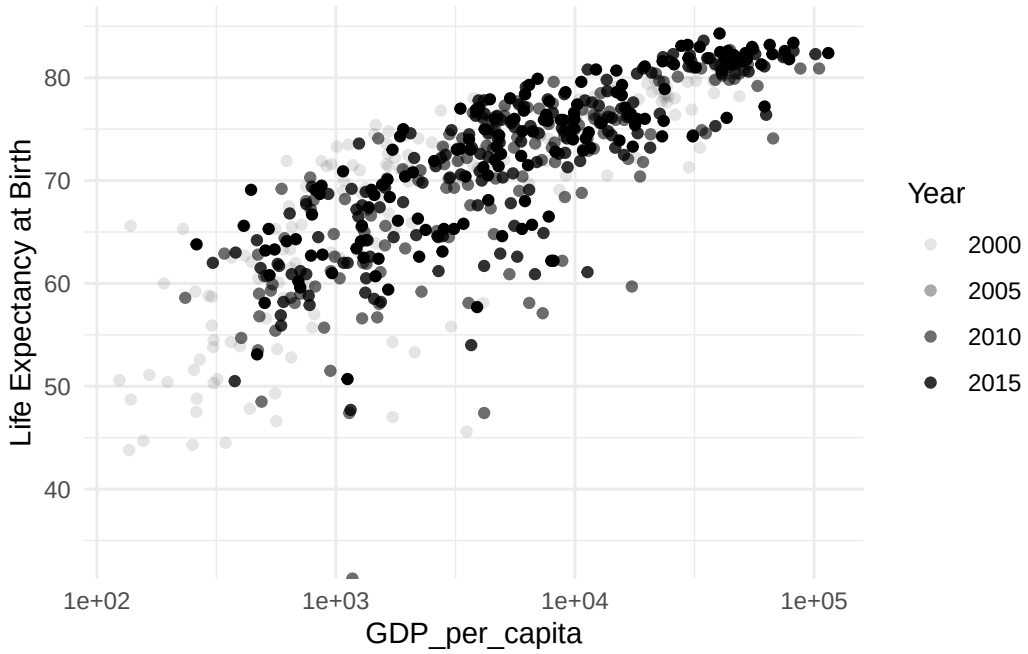
```
Attaching package: 'scales'

The following object is masked from 'package:readr':

    col_factor
```

```
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy))+
   geom_point(alpha = 0.5) +
   scale_x_log10() +
   scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.05)
```



You can display variable information through an aesthetic mapping using the aes() function in ggplot2. Instead of setting a global value (e.g., alpha = 0.5), you map the aesthetic to a column in the dataset. This enables the visualization to represent data dynamically.

```
# use opacity as an encoding channel
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, alpha = Year)) +
  geom_point() +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.05))
  theme_minimal()
```



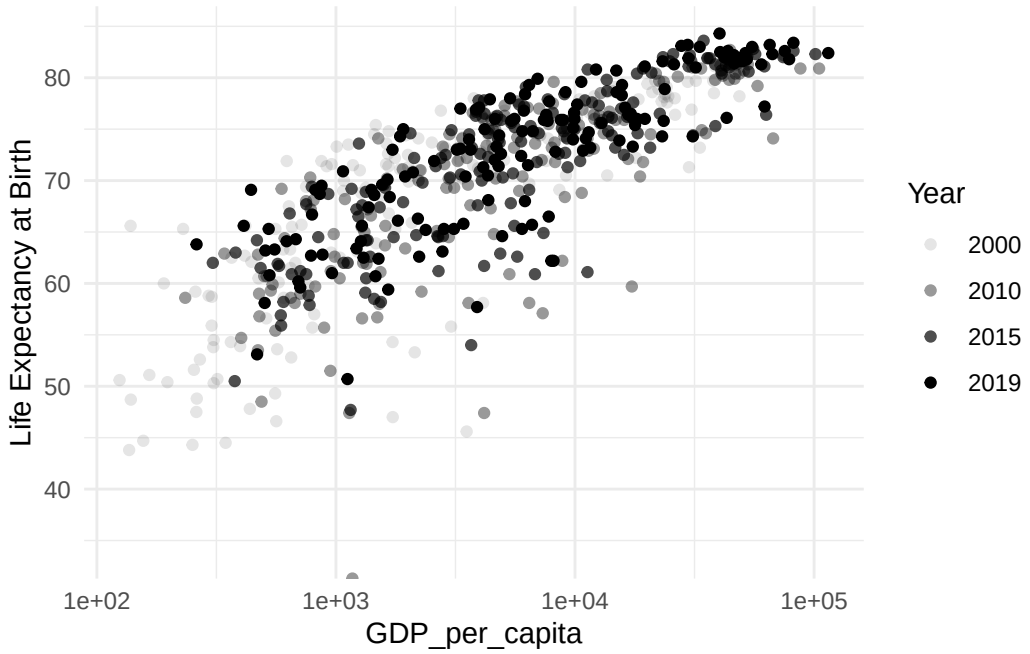
Notice that there's not actually any data for 2005. Isn't it odd, then, that the legend includes an opacity value for that year? This is because the Year variable is automatically treated as continuous due to its data type (integer). If we want to instead have a unique opacity value for each year (i.e., use a discrete scale), we can coerce the Year variable to a nominal type in R by converting it to a factor using as.factor().

Question 4: Coercing data types

In R, there isn't a direct syntax like :N as in Altair, but the equivalent operation is to convert the column to a factor, which is the R representation of a nominal (categorical) variable. Here's the code for mapping the Year column to a nominal data type:

```
data$Year <- as.factor(data$Year)

ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, alpha = Year)) +
  geom_point() +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.05))
  theme_minimal() +
  guides(alpha = guide_legend(title = "Year"))
```

```
Warning: Using alpha for a discrete variable is not advised.
```

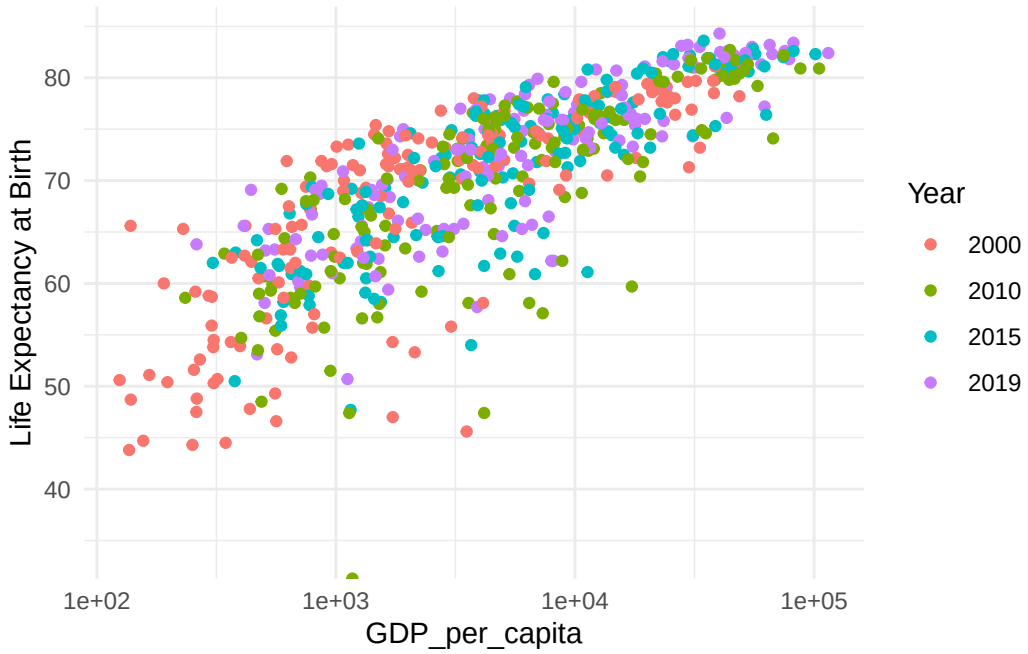


This displays more recent data in darker shades. Nice, but not especially informative. Let's try encoding year with color instead.

Question 5: Color encoding

Map `Year` to color and treat it as a nominal variable.

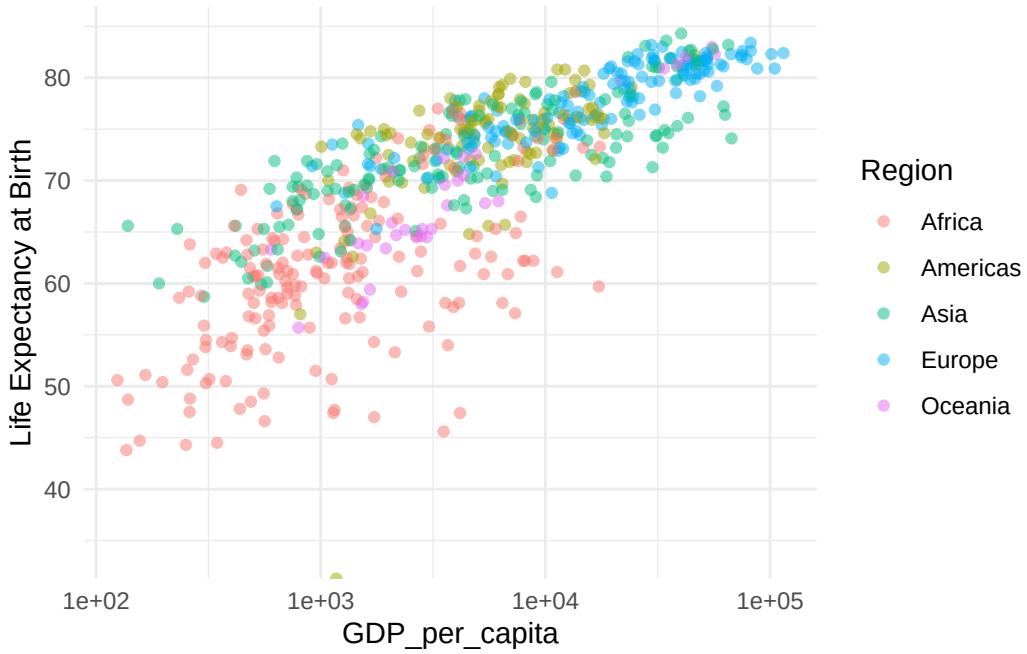
```
# map year to color
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = factor(Year))) +
  geom_point() +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.05))
  theme_minimal() +
  guides(color = guide_legend(title = "Year"))
```



Pretty, but there's not a clear pattern, so the color aesthetic for year doesn't make the plot any more informative than it was without color. This **doesn't** mean that year is unimportant; just that color probably isn't the best choice to show year.

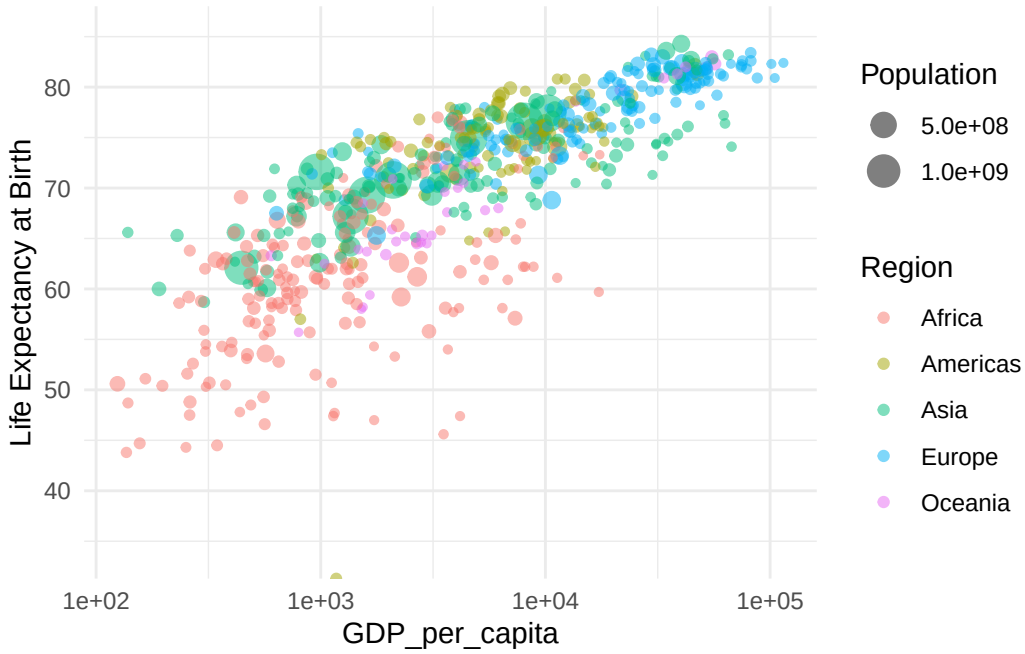
Let's try to find a color variable that does add information to the plot. When region is mapped to color, there is still substantial mixing but some apparent clustering. This communicates visually that there's some similarity in the relationship between GDP and life-expectancy among countries in the same region.

```
# map region to color
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = region)) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.05))
  theme_minimal() +
  guides(color = guide_legend(title = "Region"))
```



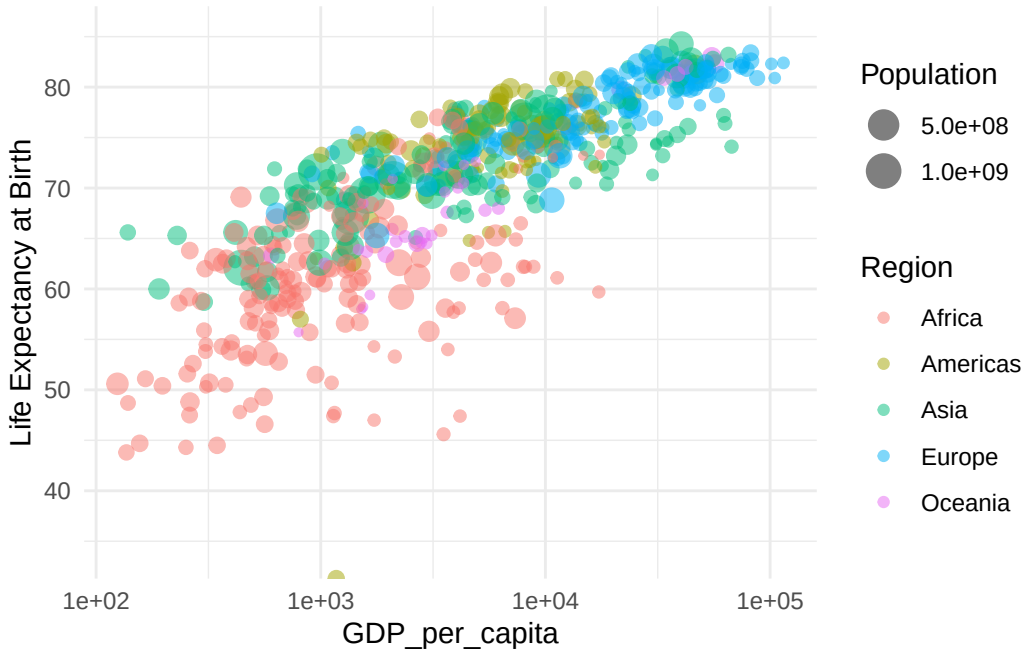
That's a little more interesting. Let's add another variable: map population to size, so that points are displayed in proportion to the country's total population.

```
# map population to size
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = region, size = Population))
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  theme_minimal() +
  guides(
    color = guide_legend(title = "Region"),
    size = guide_legend(title = "Population")
  )
```



Great, but highly populous countries in Asia are so much larger than countries in other regions that, when size is displayed on a linear scale, too many data points are hardly visible. Just like the axes were rescaled using `alt.X()` and `alt.Scale()`, other encoding channels can be rescaled, too. Below, size is put on a square root scale.

```
# rescale size
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = region, size = Population))
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region"))
```



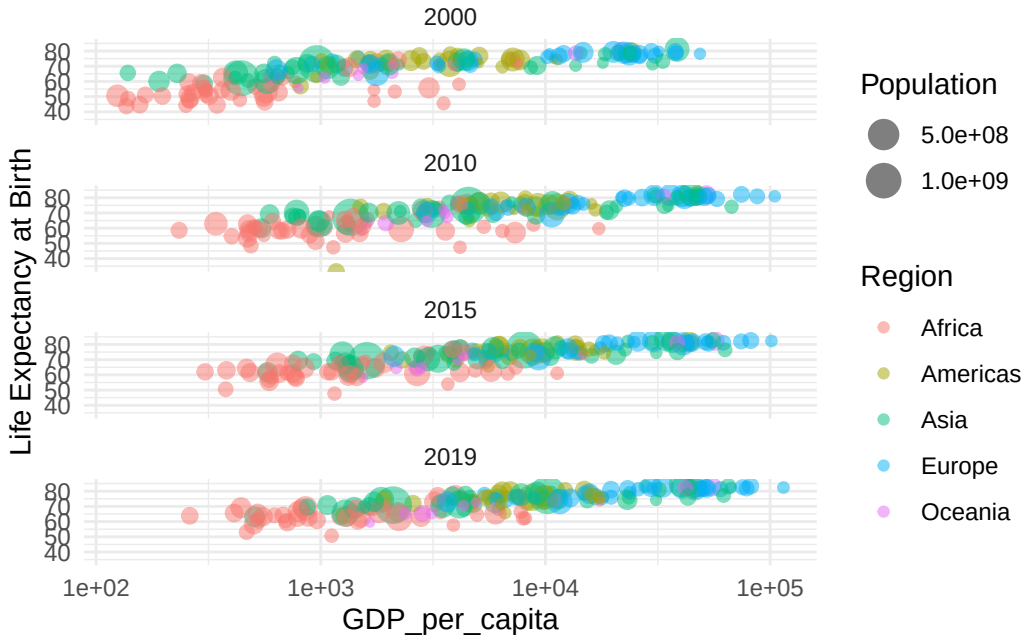
Not only does this add information, but it makes the regional clusters a little more visible!

Faceting

Your previous graphic looks pretty good, and is nearly presentation-quality. However, it still doesn't display year information. As a result, each country appears multiple times in the same plot, which is potentially misleading. Here we'll address that using faceting.

Faceting is a technique in ggplot2 that creates multiple subplots (or panels) within a single visualization. Each panel represents a subset of the data, based on one or more categorical or temporal variables. This approach is especially useful for observing trends or patterns within specific groups or time periods.

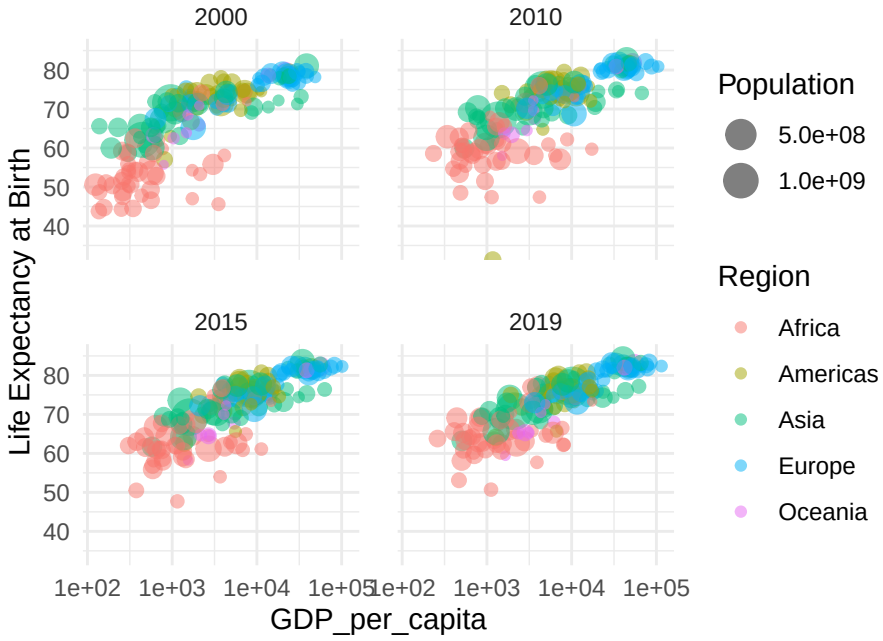
```
# facet by year
ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = region, size = Population))
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region")) +
  facet_wrap(~ Year, ncol = 1)  # Adjust the number of columns if needed
```



Question 6: Panel resizing

In R with ggplot2, resizing the individual facets is done using theme() to control the size of the plot area and its panels. You can adjust the size of individual facets using theme(panel.spacing) and controlling the overall plot dimensions with ggsave() or layout management.

```
# resize facets
# Create the faceted plot
p <- ggplot(data, aes(x = GDP_per_capita, y = Life_Expectancy, color = region, size = Popula
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region")) +
  facet_wrap(~ Year) +
  theme(
    panel.spacing = unit(1, "lines"),      # Adjust space between facets
    aspect.ratio = 0.8                     # Adjust aspect ratio of individual facets
  )
p
```




```
# Save the plot with specific dimensions
#ggsave("faceted_plot.png", plot = p, width = 12, height = 8)
```

Looks like life expectancy is increasing over time for lower-GDP nations, especially in Africa and Asia.

Can we also display the life expectancies for each sex separately? To do this, we'll need to rearrange the dataframe a little – untidy it so that we have one variable that indicates sex, and another that indicates life expectancy.

Question 7: Melt for plotting purposes

Drop the Life Expectancy column and reshape the Male Life Expectancy and Female Life Expectancy columns in data into a long format so that:

The values appear in a column called Life Expectancy at Birth. The variable names appear in a column called Group. Store the result as plot_df and print the first few rows. It may be helpful to refer to the pivot_longer documentation in tidyr for guidance. This is a pretty common operation for plotting purposes.

```
# Drop the 'Life Expectancy' column and reshape the data
plot_df <- data %>%
  select(-'Life_Expectancy') %>%                    # Drop the 'Life Expectancy' column
  pivot_longer(
```

```
    cols = c('Male_Life_Expectancy', 'Female_Life_Expectancy'), # Columns to melt
    names_to = "Group",                                          # New column for variable name
    values_to = "Life_Expectancy_at_Birth"                      # New column for values
  )

# Print the first few rows of the reshaped data
head(plot_df)
```

```
# A tibble: 6 x 8
  `Country Name` Year  GDP_per_capita region `sub-region`  Population Group
  <chr>          <fct>          <dbl> <chr>  <chr>              <dbl> <chr>
1 Afghanistan    2019            507. Asia   Southern Asia   38041754 Male_Life~
2 Afghanistan    2019            507. Asia   Southern Asia   38041754 Female_Li~
3 Afghanistan    2015            578. Asia   Southern Asia   34413603 Male_Life~
4 Afghanistan    2015            578. Asia   Southern Asia   34413603 Female_Li~
5 Afghanistan    2010            543. Asia   Southern Asia   29185507 Male_Life~
6 Afghanistan    2010            543. Asia   Southern Asia   29185507 Female_Li~
# i 1 more variable: Life_Expectancy_at_Birth <dbl>
```

You will need to complete the part above correctly before moving on. Check the result of the following cell (first few rows for each group) against the reference dataframe below – they should match exactly.

```
# Group by 'Group' and take the first 4 rows from each group
plot_df %>%
  group_by(Group) %>%
  slice_head(n = 4) %>%
  ungroup()
```

```
# A tibble: 8 x 8
  `Country Name` Year  GDP_per_capita region `sub-region`    Population Group
  <chr>          <fct>          <dbl> <chr>  <chr>                <dbl> <chr>
1 Afghanistan    2019            507. Asia   Southern Asia     38041754 Female_~
2 Afghanistan    2015            578. Asia   Southern Asia     34413603 Female_~
3 Afghanistan    2010            543. Asia   Southern Asia     29185507 Female_~
4 Albania        2019           5353. Europe Southern Europe    2854191 Female_~
5 Afghanistan    2019            507. Asia   Southern Asia     38041754 Male_Li~
6 Afghanistan    2015            578. Asia   Southern Asia     34413603 Male_Li~
7 Afghanistan    2010            543. Asia   Southern Asia     29185507 Male_Li~
8 Albania        2019           5353. Europe Southern Europe    2854191 Male_Li~
# i 1 more variable: Life_Expectancy_at_Birth <dbl>
```

```r
# check result
# Read the CSV file into a data frame
plotdf_check <- read.csv('data/plotdf-check.csv')

# View the first few rows
head(plotdf_check)
```

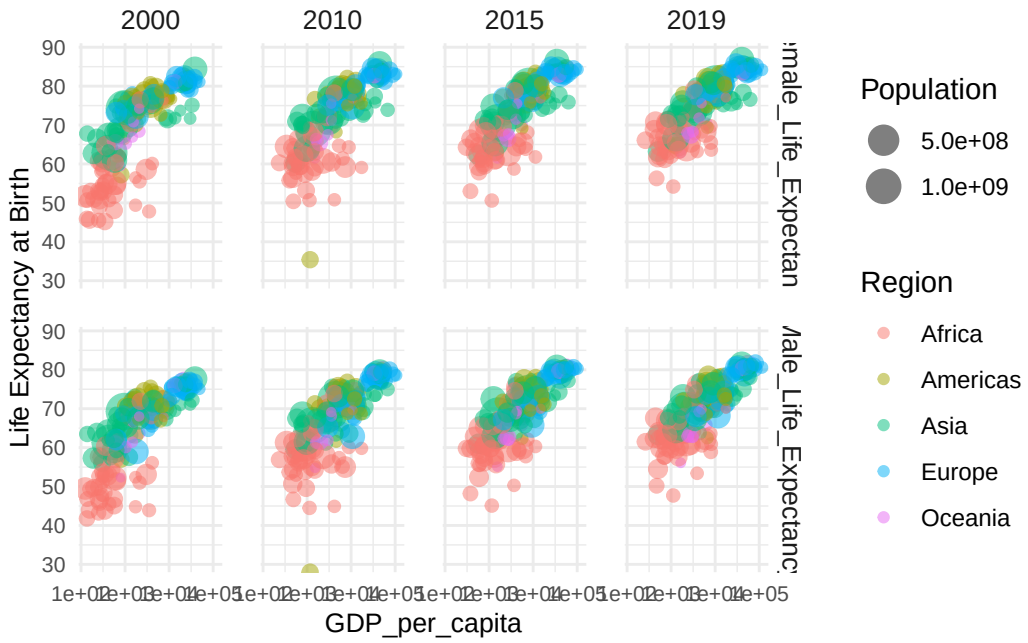
```
  Country.Name Year GDP.per.capita region      sub.region Population
1  Afghanistan 2019       507.1034   Asia   Southern Asia   38041754
2  Afghanistan 2015       578.4664   Asia   Southern Asia   34413603
3  Afghanistan 2010       543.3030   Asia   Southern Asia   29185507
4      Albania 2019      5353.2449 Europe Southern Europe    2854191
5  Afghanistan 2019       507.1034   Asia   Southern Asia   38041754
6  Afghanistan 2015       578.4664   Asia   Southern Asia   34413603
                  Group Life.Expectancy.at.Birth
1   Male Life Expectancy                    63.3
2   Male Life Expectancy                    61.0
3   Male Life Expectancy                    59.6
4   Male Life Expectancy                    76.3
5 Female Life Expectancy                    63.2
6 Female Life Expectancy                    62.3
```

To facet by both Year and Group (representing sex), you can use facet_grid() in ggplot2.

```r
# Filter out rows where Group is 'Life Expectancy'
filtered_plot_df <- plot_df %>%
  filter(Group != "Life Expectancy")

#View(filtered_plot_df)
```

```r
# Create the faceted plot
p1<- ggplot(filtered_plot_df, aes(x = GDP_per_capita, y = Life_Expectancy_at_Birth, color = r
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region")) +
  facet_grid(Group ~ Year) +  # Facet by Group (rows) and Year (columns)
  theme(
    panel.spacing = unit(1, "lines"),  # Adjust spacing between facets
```

```
    strip.text = element_text(size = 10),  # Adjust facet label text size
    axis.text = element_text(size = 8),    # Adjust axis text size
    axis.title = element_text(size = 10)   # Adjust axis title size
  )
p1
```




```
#ggsave("faceted_plot.png", plot = p1, width = 12, height = 8)
```

Question 8: Adjusting facet layout

It's a little hard to line up the patterns visually between sexes because they are aligned on GDP per capita, not life expectancy – so we can't really tell without moving our eyes back and forth and checking the axis ticks whether there's much difference in life expectancy rates by sex. Switching the row/column layout gives a better result. Modify the cell below so that facet columns correspond to sex and facet rows correspond to years.

To modify the layout so that facet columns correspond to Group (sex) and facet rows correspond to Year, we simply switch the order in the facet_grid() function in R. Here's the updated code:

```
# Filter out rows where Group is 'Life Expectancy'
filtered_plot_df <- plot_df %>%
  filter(Group != "Life Expectancy")
```

```r
# Create the faceted plot with switched row/column layout
ggplot(filtered_plot_df, aes(x = GDP_per_capita, y = Life_Expectancy_at_Birth, color = regio
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Life Expectancy at Birth", expand = expansion(mult = c(0, 0.07))
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region")) +
  facet_grid(Year ~ Group) +  # Facet by 'Year' (rows) and 'Group' (columns)
  theme(
    panel.spacing = unit(1, "lines"),  # Adjust spacing between facets
    strip.text = element_text(size = 10),  # Adjust facet label text size
    axis.text = element_text(size = 8),   # Adjust axis text size
    axis.title = element_text(size = 10)  # Adjust axis title size
  )
```

So life expectancy is a bit lower for men on average. But from the plot it's hard to tell if some countries reverse this pattern, since you can't really tell which country is which. Also, the panel is a bit cumbersome. Take a moment to consider how you might improve these issues, and then move on to our suggestion below.

The next parts will modify the dataframe `data` by adding a column. We'll create a copy `data_mod1` of the original dataframe `data` to modify as to not lose track of our previous work:

```
data_mod1 = data
```

Question 9: Data transformation and re-plotting

A simple data transformation can help give a clearer and more concise picture of how life expectancy differs by sex. Perform the following steps: * append a new variable `Difference` to `data_mod1` that gives the difference between female and male (F - M) life expectancies in each country and year; * modify the your plot of general life expectancy against GDP per capita by year to instead plot the difference in life expectancies at birth against GDP per capita by year.

When modifying the example, be sure to change the axis label appropriately.

```
# Create a copy of the original dataset
data_mod1 <- data_mod1 %>%
  mutate(Difference = Female_Life_Expectancy - Male_Life_Expectancy)

# View the first few rows of the modified data
head(data_mod1)
```

```
# A tibble: 6 x 10
  `Country Name` Year  Life_Expectancy Male_Life_Expectancy
  <chr>          <fct>           <dbl>                <dbl>
1 Afghanistan    2019             63.2                 63.3
2 Afghanistan    2015             61.7                 61
3 Afghanistan    2010             59.9                 59.6
4 Albania        2019             78                   76.3
5 Albania        2015             77.8                 76.1
6 Albania        2010             76.2                 74.2
# i 6 more variables: Female_Life_Expectancy <dbl>, GDP_per_capita <dbl>,
#   region <chr>, `sub-region` <chr>, Population <dbl>, Difference <dbl>
```

```
# Plot the difference in life expectancies
ggplot(data_mod1, aes(x = GDP_per_capita, y = Difference, color = region, size = Population)
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(name = "Difference in Life Expectancy (F - M)", expand = expansion(mult
  scale_size_continuous(trans = "sqrt", guide = guide_legend(title = "Population")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Region")) +
  facet_wrap(~ Year, ncol = 1) +  # Facet by Year in a single column
```

```r
theme(
  panel.spacing = unit(1, "lines"),  # Adjust spacing between facets
  strip.text = element_text(size = 10),  # Adjust facet label text size
  axis.text = element_text(size = 8),   # Adjust axis text size
  axis.title = element_text(size = 10)  # Adjust axis title size
)
```

Question 10: Interpretation

Note in the last graphic that (1) each panel shows an increasing trend and (2) one region shows the opposite trend. Interpret these observations in context.

Type your answer here, replacing this text. The general trend across most regions suggests that wealthier countries tend to exhibit larger gender disparities in life expectancy, favoring women. However, the anomaly in one region highlights the importance of examining local cultural, socioeconomic, and policy factors that influence health outcomes. Understanding these regional differences can help policymakers design targeted interventions to address inequities in health and longevity.

Submission

1. Save the notebook.

2. Restart the kernel and run all cells. (**CAUTION**: if your notebook is not saved, you will lose your work.)
3. Carefully look through your notebook and verify that all computations execute correctly and all graphics are displayed clearly. You should see **no errors**; if there are any errors, make sure to correct them before you submit the notebook.
4. Download the notebook as an `.ipynb` file. This is your backup copy.
5. Export the notebook as PDF and upload to Gradescope.