TEXT AS DATA: WEEK 7

MATTHIAS HABER

27 OCTOBER 2021

GOALS FOR TODAY

GOALS

- Solution assignment 1
- ggplot2
- Visualizing text

SOLUTION TO ASSIGNMENT 1

REGULAR EXPRESSIONS

```
addresses <-c("221B Baker St., London", "1600 Pennsylvania Avenue,
        Washington D.C.",
              "742 Evergreen Terrace, Springfield")
sentences <- stringr::sentences[1:20]
field_names <- c("order_number", "order_date", "customer_email",
        "product_title", "amount")
email <- c("tom@hogwarts.com",
           "tom.riddle@hogwarts.com",
           "tom@hogwarts.eu.com",
           "potter@hogwarts.com",
           "harry@hogwarts.com",
           "hermione+witch@hogwarts.com")
files <- c(".bash_profile",
           "workspace.doc",
           "img0912.jpg",
           "updated_img0912.png",
           "documentation.html",
```

CONVERT ADDRESSES TO LOWER-CASE

```
library(stringr)
str_to_lower(string = addresses)
```

```
## [1] "221b baker st., london"
## [2] "1600 pennsylvania avenue, washington d.c."
## [3] "742 evergreen terrace, springfield"
```

EXTRACT DIGITS FROM ADDRESSES

```
str_extract(string = addresses, pattern = "[:digit:]+")
```

```
## [1] "221"  "1600" "742"
```

SPLIT ADDRESSES INTO TWO PARTS: STREET & CITY

```
str_split(string = addresses, pattern = ", ", simplify = T)
```

```
##      [,1]                       [,2]
## [1,] "221B Baker St."           "London"
## [2,] "1600 Pennsylvania Avenue" "Washington D.C."
## [3,] "742 Evergreen Terrace"    "Springfield"
```

SPLIT ADDRESSES INTO THREE PARTS: HOUSE NUMBER, STREET & CITY

```
str_split(string = addresses, pattern = "(?<=[:digit:].) |, ", simplify
        = T)
```

```
##      [,1]   [,2]                  [,3]
## [1,] "221B" "Baker St."           "London"
## [2,] "1600" "Pennsylvania Avenue" "Washington D.C."
## [3,] "742"  "Evergreen Terrace"   "Springfield"
```

FOR SENTENCES THAT END WITH THE LETTER “T” EXTRACT THE LAST WORD

```
str_extract_all(string = sentences, pattern = "[A-z]+t\\.$")
```

```
## [[1]]
## character(0)
##
## [[2]]
## character(0)
##
## [[3]]
## character(0)
##
## [[4]]
## character(0)
##
## [[5]]
## character(0)
##
## [[6]]
## character(0)
##
```

EXTRACT THE FIRST 30 CHARACTERS FROM EACH SENTENCE

```
str_trunc(string = sentences, width = 30, ellipsis = "...")
```

```
##  [1] "The birch canoe slid on the..." "Glue the sheet to the dark ...
##  [3] "It's easy to tell the depth..." "These days a chicken leg is...
##  [5] "Rice is often served in rou..." "The juice of lemons makes f...
##  [7] "The box was thrown beside t..." "The hogs were fed chopped c...
##  [9] "Four hours of steady work f..." "Large size in stockings is ...
## [11] "The boy was there when the ..." "A rod is used to catch pink...
## [13] "The source of the huge rive..." "Kick the ball straight and ...
## [15] "Help the woman get back to ..." "A pot of tea helps to pass ...
## [17] "Smoky fires lack flame and ..." "The soft cushion broke the ...
## [19] "The salt breeze came across..." "The girl at the booth sold ...
```

REPLACE ALL UNDERSCORES IN FIELD_NAMES WITH SPACES AND CAPITALIZE THE FIRST LETTER OF EACH WORD

```
str_replace_all(string = field_names, pattern = "_", replacement = " ")
        %>%
  str_to_title()
```

```
## [1] "Order Number"   "Order Date"     "Customer Email" "Product Title"
## [5] "Amount"
```

EXTRACT NAMES APPEARING BEFORE @ FROM EMAIL

```
str_extract(email, "^[\\w\\.\\+]*")
```

```
## [1] "tom"           "tom.riddle"     "tom"           "potter"
## [5] "harry"         "hermione+witch"
```

EXTRACT THE THREE IMAGES (.JPG, .PNG, .GIF) FROM FILES

```
str_extract(files, ".*\\.(jpg|png|gif)$")
```

```
## [1] NA                    NA                    "img0912.jpg"
## [4] "updated_img0912.png" NA                    "favicon.gif"
## [7] NA                    NA
```

WEB SCRAPING & TIDYTEXT

```
library(rvest)
library(tidytext)
```

EXTRACT THE TEXT OF J.K. ROWLING'S COMMENCEMENT SPEECH AT HARVARD UNIVERSITY

```
url <- "https://news.harvard.edu/gazette/story/2008/06/text-of-j-k-
        rowling-speech/"
speech <- read_html(url) %>%
  html_nodes(".article-body p") %>%
  html_text()
```

CONVERT THE TEXT TO TIBBLE, REMOVE THE FIRST AND 15TH LINE, ADD PARAGRAPH NUMBER

```
speech_df <- tibble(text = speech) %>%
 slice(c(2:14,16:n())) %>%
  mutate(paragraphs = row_number())
```

TOKENIZE THE TEXT INTO WORDS AND REMOVE STOP WORDS

```
speech_df <- speech_df %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

LIST THE 5 MOST FREQUENT WORDS

```
speech_df %>%
  count(word, sort = TRUE) %>%
  top_n(5)
```

```
## # A tibble: 5 × 2
##   word        n
##   <chr>   <int>
## 1 life       14
## 2 failure    13
## 3 day        10
## 4 people      9
## 5 parents     8
```

READTEXT AND QUANTEDA

```
library(readtext)
library(quanteda)
```

READ IN THE UK MANIFESTOS AND CREATE DOCUMENT NAMES FROM FILE NAMES

```
manifestos <- readtext("data/UK_manifestos/*.txt",
                       docvarsfrom= "filenames", dvsep = "-",
                       docvarnames = c("party", "year"))
```

CREATE A CORPUS OUT OF THE DOCUMENTS

```
manifesto_corpus <- corpus(manifestos)
summary(manifesto_corpus)
```

```
## Corpus consisting of 18 documents, showing 18 documents:
##
##          Text Types Tokens Sentences party year
##  Con-1918.txt   727   2084        56   Con 1918
##  Con-1922.txt   567   1665        42   Con 1922
##  Con-1923.txt   641   1715        59   Con 1923
##  Con-1924.txt  1016   3169        85   Con 1924
##  Con-1929.txt  1699   7103       222   Con 1929
##  Con-1931.txt   387    895        30   Con 1931
##  Lab-1918.txt   476   1241        47   Lab 1918
##  Lab-1922.txt   589   1401        54   Lab 1922
##  Lab-1923.txt   550   1259        39   Lab 1923
##  Lab-1924.txt   877   2495        69   Lab 1924
##  Lab-1929.txt   997   2720       116   Lab 1929
##  Lab-1931.txt   812   2267        91   Lab 1931
##  Lib-1918.txt   424    917        27   Lib 1918
##  Lib-1922.txt   467   1057        48   Lib 1922
##  Lib-1923.txt   863   2362        85   Lib 1923
```

OVERWRITE THE PARTY COLUMN WITH NEW LABELS

```
party <- c(rep("Conservatives", 6), rep("Labour", 6), rep("LibDems",
        6))
manifesto_corpus$party <- party
summary(manifesto_corpus)
```

```
## Corpus consisting of 18 documents, showing 18 documents:
##
##          Text Types Tokens Sentences        party year
##  Con-1918.txt   727   2084        56 Conservatives 1918
##  Con-1922.txt   567   1665        42 Conservatives 1922
##  Con-1923.txt   641   1715        59 Conservatives 1923
##  Con-1924.txt  1016   3169        85 Conservatives 1924
##  Con-1929.txt  1699   7103       222 Conservatives 1929
##  Con-1931.txt   387    895        30 Conservatives 1931
##  Lab-1918.txt   476   1241        47       Labour 1918
##  Lab-1922.txt   589   1401        54       Labour 1922
##  Lab-1923.txt   550   1259        39       Labour 1923
##  Lab-1924.txt   877   2495        69       Labour 1924
##  Lab-1929.txt   997   2720       116       Labour 1929
##  Lab-1931.txt   812   2267        91       Labour 1931
##  Lib-1918.txt   424    917        27      LibDems 1918
##  Lib-1922.txt   467   1057        48      LibDems 1922
##  Lib-1923.txt   863   2362        85      LibDems 1923
```

TOKENIZE AND PREPROCESS THE TEXT AND FIND CONTEXT FOR DISCUSSION AROUND "EUROPE"

```
manifesto_tokens <- tokens(manifesto_corpus, what = "word",
                           remove_numbers = TRUE,
                           remove_punct = TRUE) %>%
  tokens_tolower()
kwic(manifesto_tokens, "europe*")
```

```
## Keyword-in-context with 27 matches.
##    [Con-1918.txt, 76]                forever in the continent of |  eur
##   [Con-1918.txt, 203]                     to make the peace of |  eur
##   [Con-1918.txt, 277]               settle the political future of |  eur
##   [Con-1918.txt, 362]                    the foundations of a new |  eur
##  [Con-1922.txt, 1306]                         war we have lost in |  eur
##   [Con-1923.txt, 150] political and economic disorganisation of |  eur
##   [Con-1923.txt, 192]                          of a true peace in |  eur
##   [Con-1923.txt, 278]           the disorganisation and poverty of |  eur
##   [Con-1923.txt, 329]                   standard of living in many | euro
##   [Con-1923.txt, 783]             arising from the distractions of |  eur
##   [Con-1924.txt, 638]       restore stable economic conditions in |  eur
##   [Con-1924.txt, 989]                  settled state of affairs in |  eur
##  [Con-1929.txt, 1689]                      more upon them than any | euro
##  [Con-1929.txt, 5833]                   depends has been assured in |  eur
##  [Con-1929.txt, 5848]             security has been extended from |  eur
##  [Con-1929.txt, 6148]                       states of america as in | euro
```

REMOVE STOP WORDS AND CONVERT INTO BIGRAMS

```
manifesto_bigrams<- manifesto_tokens %>%
  tokens_remove(stopwords("en", source = "smart")) %>%
  tokens_ngrams(n=2)
```

CREATE A DOCUMENT FEATURE MATRIX FOR ONLY LIBERAL DEMOCRATS

```
manifesto_dfm <- dfm(manifesto_bigrams) %>%
  dfm_subset(., party == "LibDems")
```

FIND OUT HOW MANY BIGRAMS APPEAR ONLY ONCE

```
manifesto_dfm %>%
  dfm_trim(min_termfreq = 1, max_termfreq = 1) %>%
  nfeat()
```

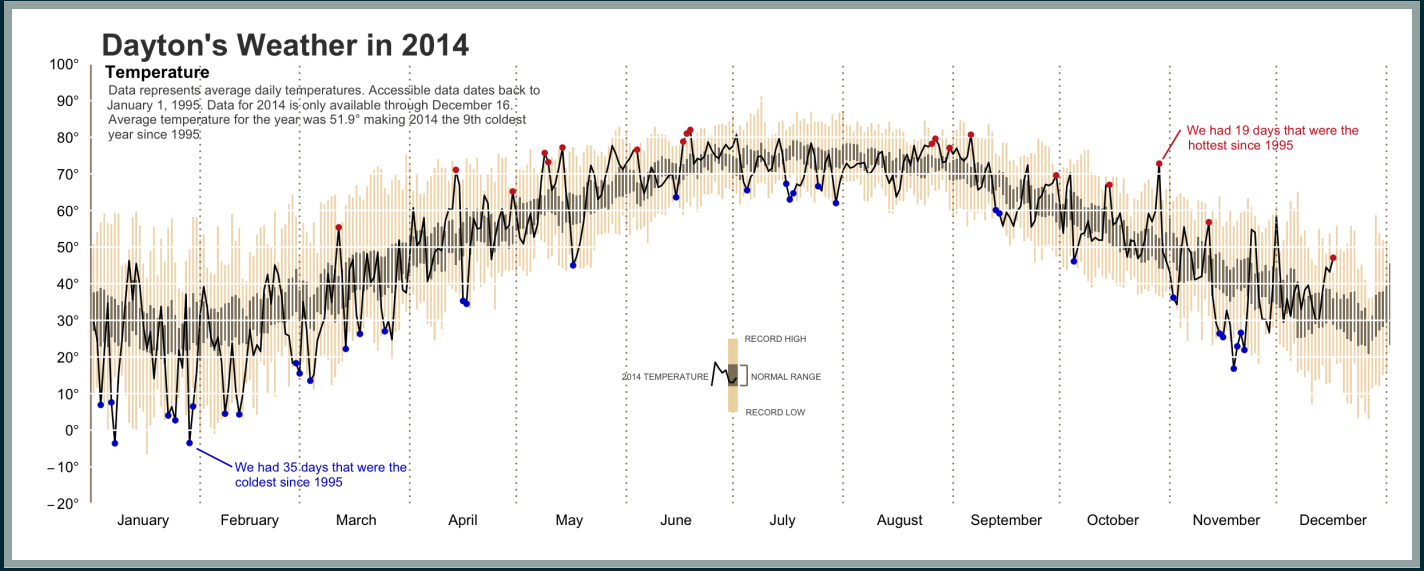
```
## [1] 3453
```

GGPLOT2

GGPLOT2

R has several systems for making graphs, but `ggplot2` is one of the most elegant and most versatile. `ggplot2` implements the grammar of graphics, a coherent system for describing and building graphs.

GGPLOT2 EXAMPLES

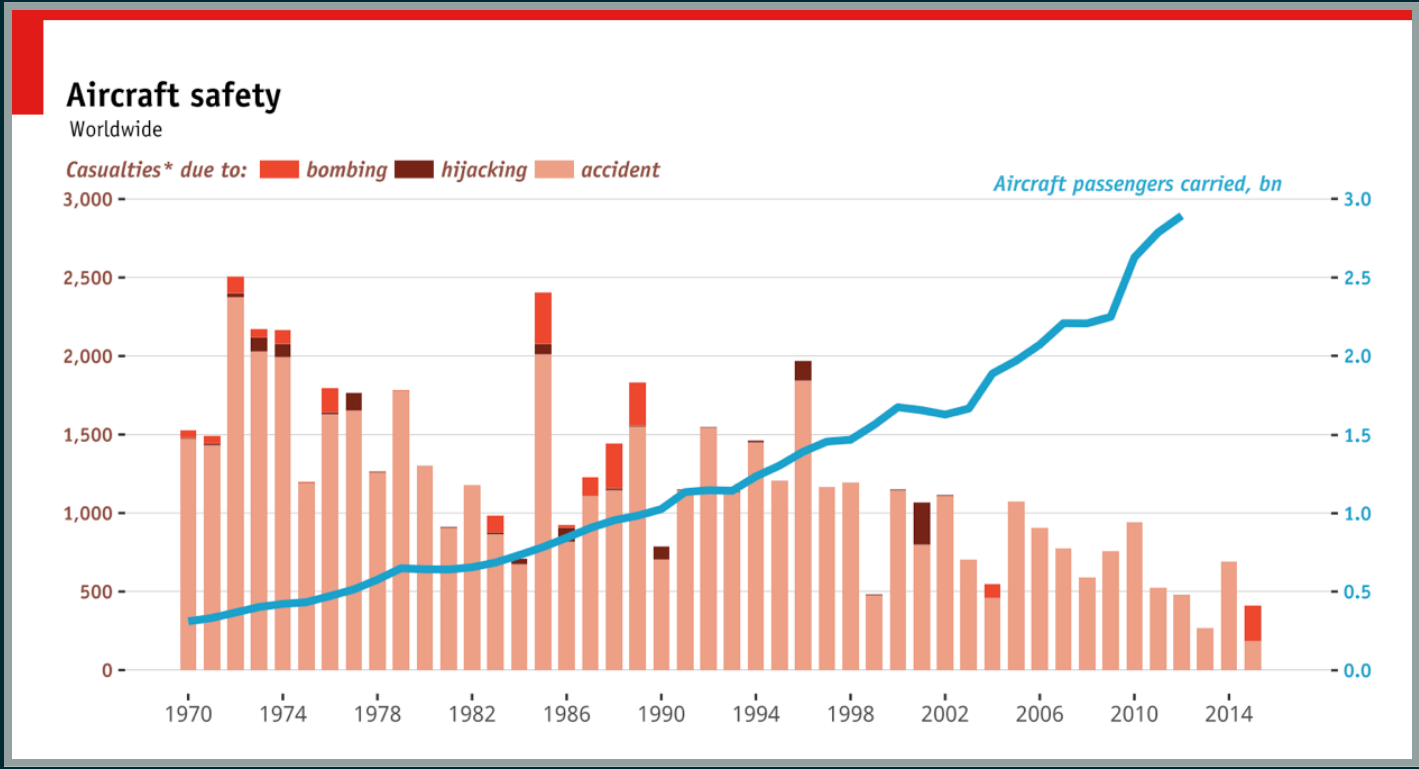


Source

GGPLOT2 EXAMPLES

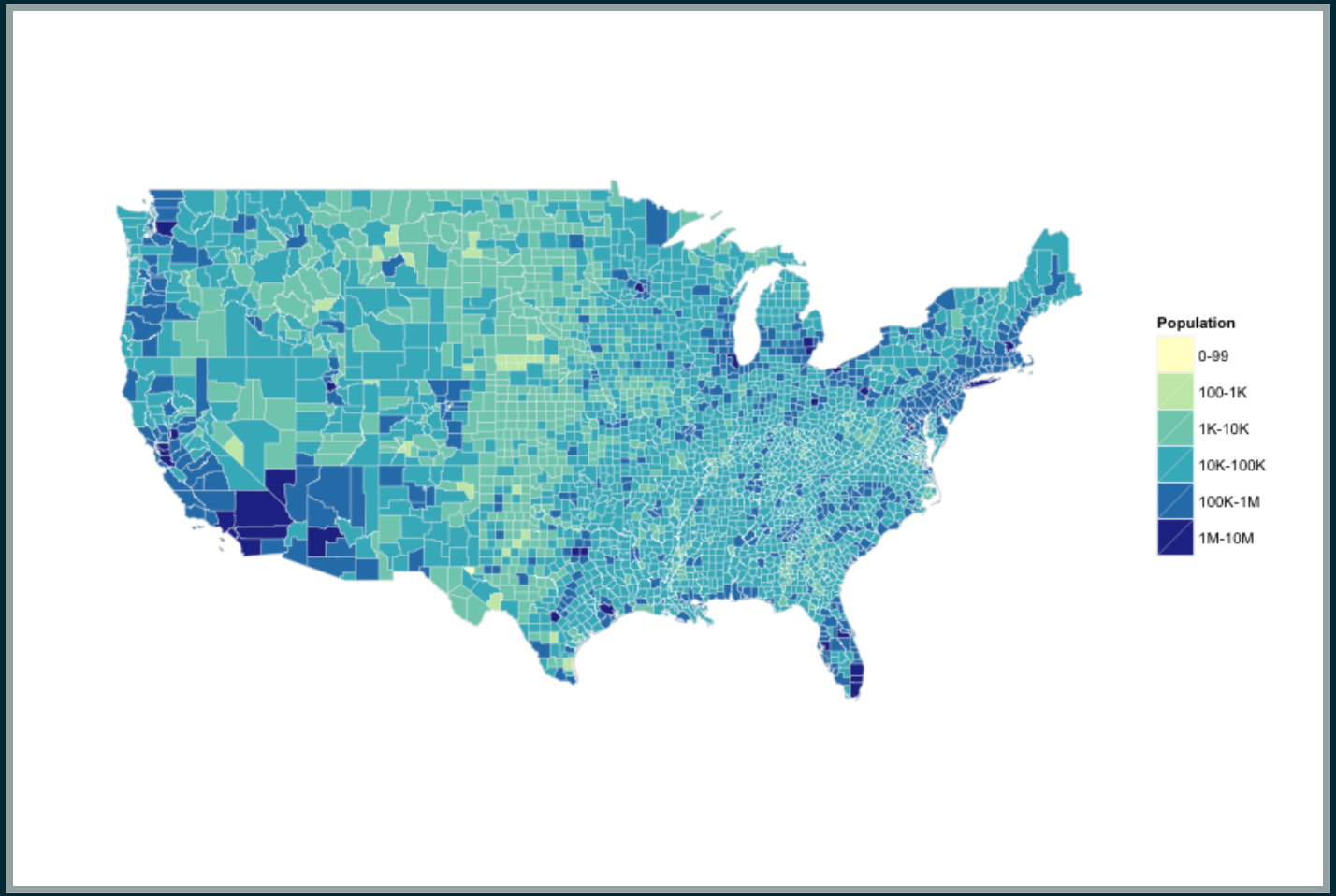
Source

GGPLOT2 EXAMPLES



Source

GGPLOT2 EXAMPLES



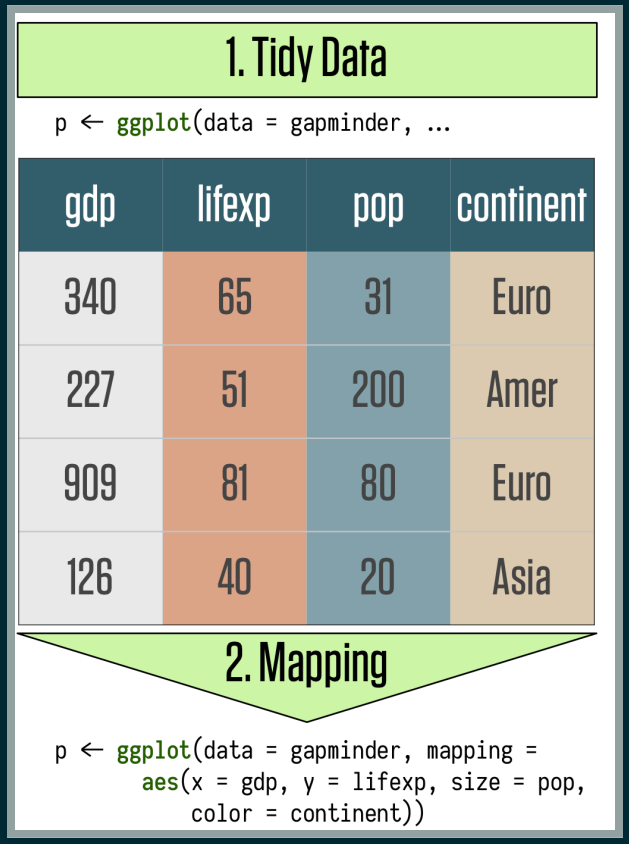
Source

THE GRAMMAR OF GRAPHICS

- Each plot is made of layers. Layers include the coordinate system (x-y), points, labels, etc.
- Each layer has aesthetics (`aes`) including x & y, size, shape, and color.
- The main layer types are called geometrics(`geom`) and include lines, points, etc.

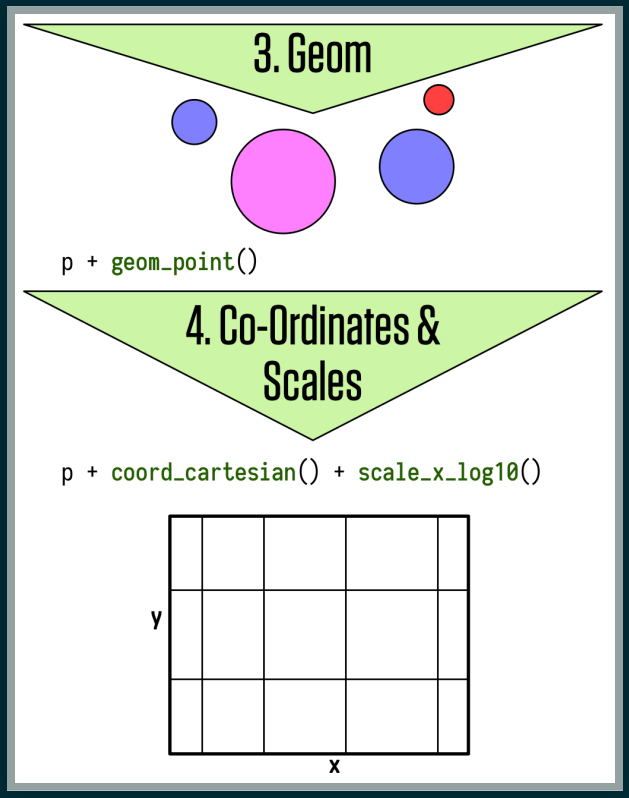
THE GRAMMAR OF GRAPHICS

A `ggplot` is build piece by piece

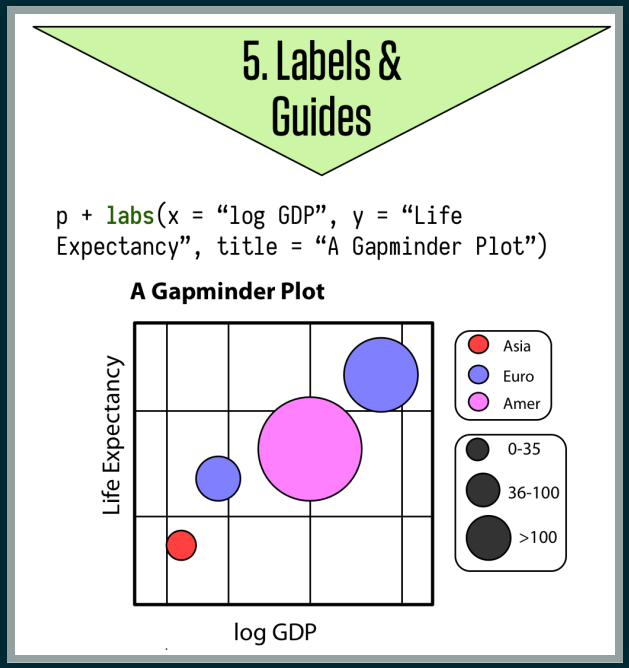


Source

THE GRAMMAR OF GRAPHICS



THE GRAMMAR OF GRAPHICS



GGPLOT WORKFLOW

1. Tell the `ggplot()` function what your data are.
2. Tell `ggplot` what relationships we want to see.
3. Tell `ggplot` how you want to see the relationships in your data.
4. Add additional layers to the p object one at a time.
5. Use additional functions to adjust scales, labels, tick marks.

COMPONENTS OF A GGPLOT2 GRAPH

- data: Variables mapped to aesthetic attributes
- aesthetic: Visual property of the plot objects
- geom: Geometrical object used to represent data
- stats: Statistical transformations of the data
- scales: Values mapped to aesthetic attributes
- coord: Coordinate system
- facets: Subplots that each display one subset of the data

MAPPING

We start creating a plot by telling `ggplot` what our data are and by storing the function in an object called `p`. For example, let's say we want to use the `gapminder` data to plot life expectancy against GDP per capita:

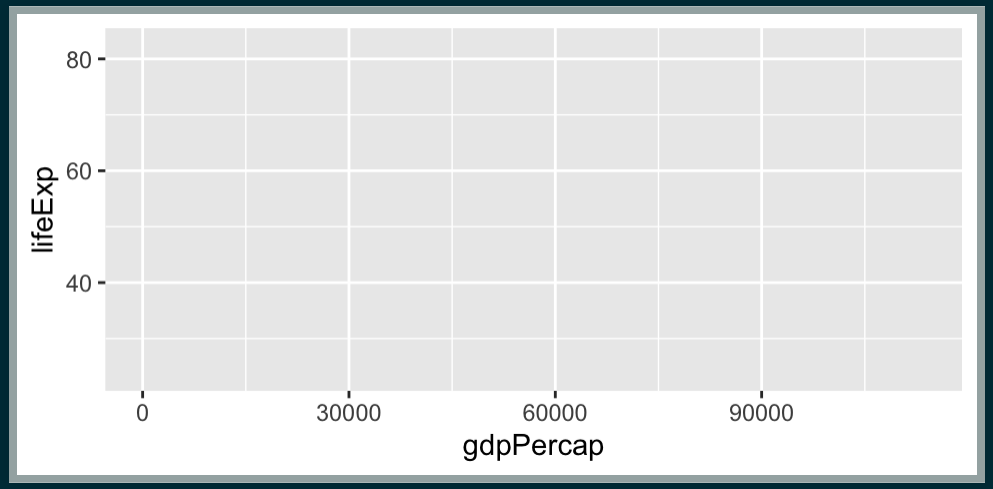
```
library(gapminder)
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
```

The `data` argument tells ggplot where to find the variables it is about to use. The `mapping = aes(...)` argument links variables to things you will see on the plot.

MAPPING

What happens if we just type `p` into the console at this point and hit return?

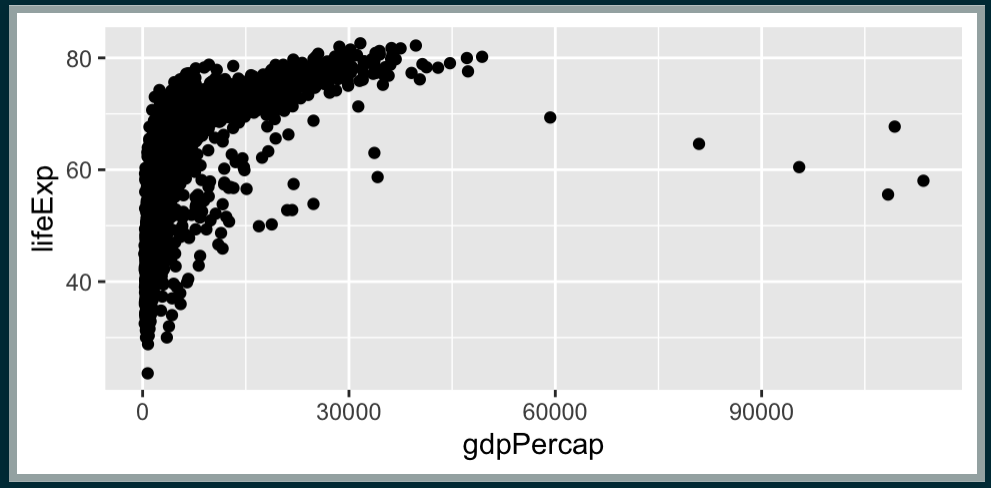
```
p
```



CHOOSING A GEOM

`p` already contains some information about out plot structure but we haven't given it any instructions yet about what sort of plot to draw. We need to add a layer to the plot by using the `geom_` function:

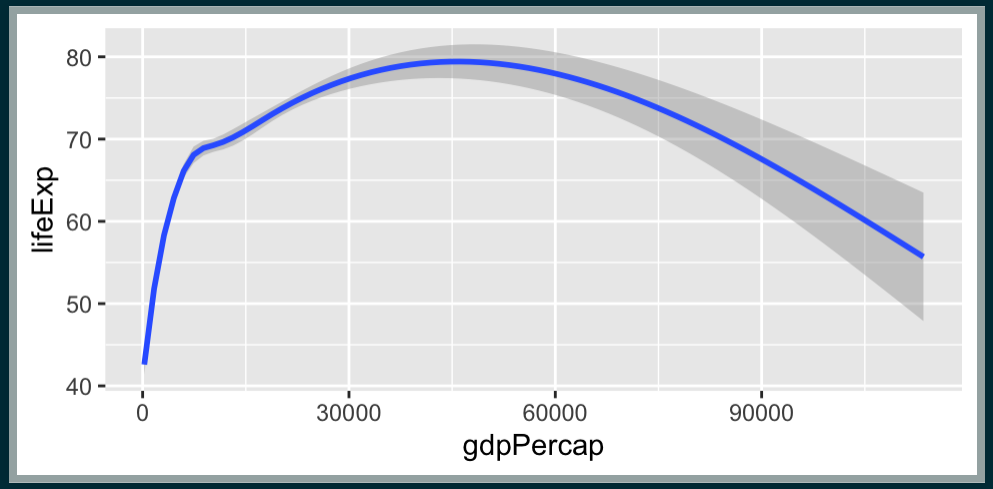
```
p + geom_point()
```



CHOOSING A GEOM

Let's try a different `geom_` and see what happens:

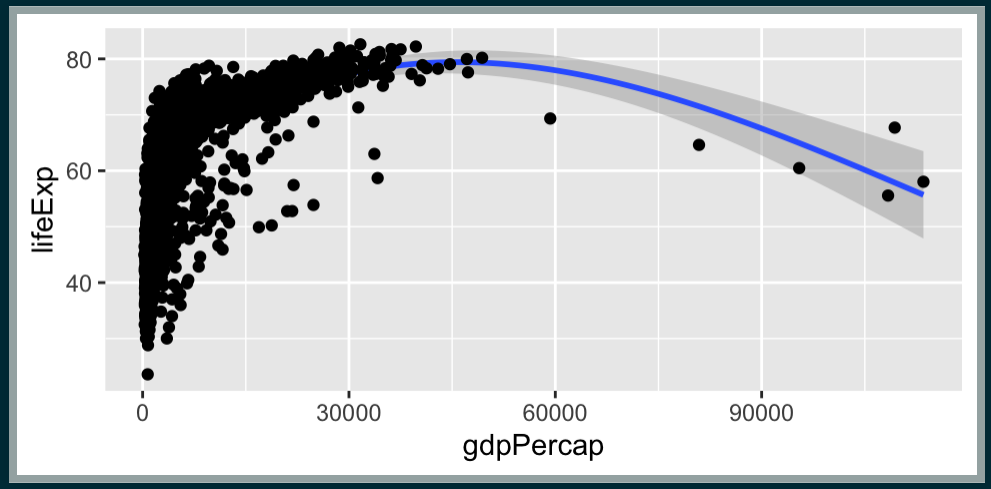
```
p + geom_smooth()
```



CHOOSING A GEOM

If we want to see data points and a smoothed line together we simply add `geom_point()` to the plot:

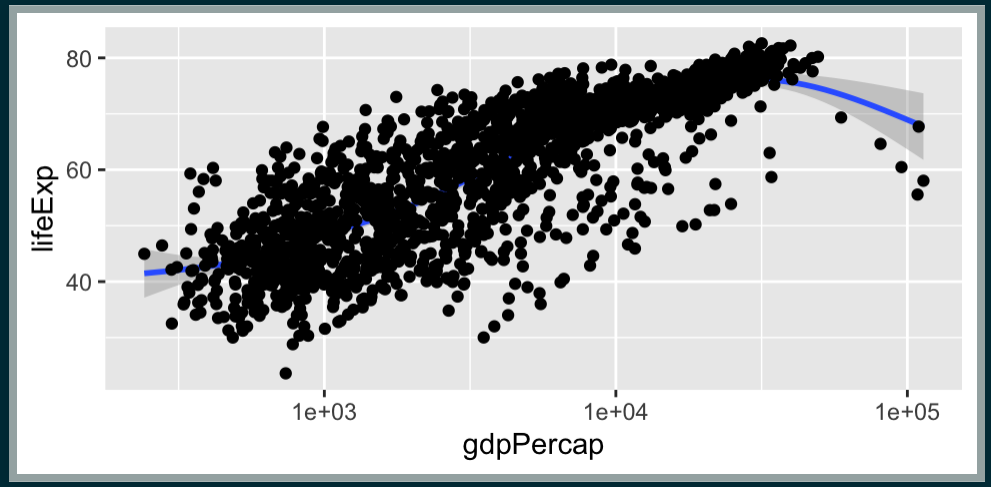
```
p + geom_smooth() + geom_point()
```



ADJUSTING THE SCALES

GDP pc does not seem to be normally distributed. We can account for that and transform the x-axis from a linear to a log scale by adding the `scale_x_log10()` function.

```
p + geom_smooth() + geom_point() + scale_x_log10()
```

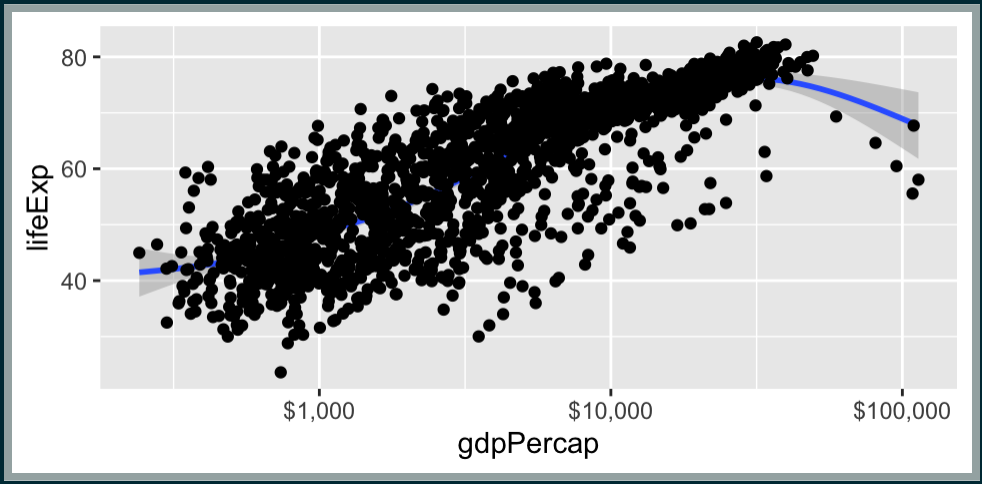


LABELS AND TITLES

Having created an interesting plot, we could now polish it up with nicer axis labels and titles. For example, let's swap the scientific notation on the x-axis for something more meaningful such as US dollars. The labels on the tick-marks can be controlled through the `scale_` functions. You can supply your own functions are use the pre-made functions from the handy `scales` library.

LABELS AND TITLES

```
p + geom_smooth() + geom_point() +
  scale_x_log10(labels = scales::dollar)
```

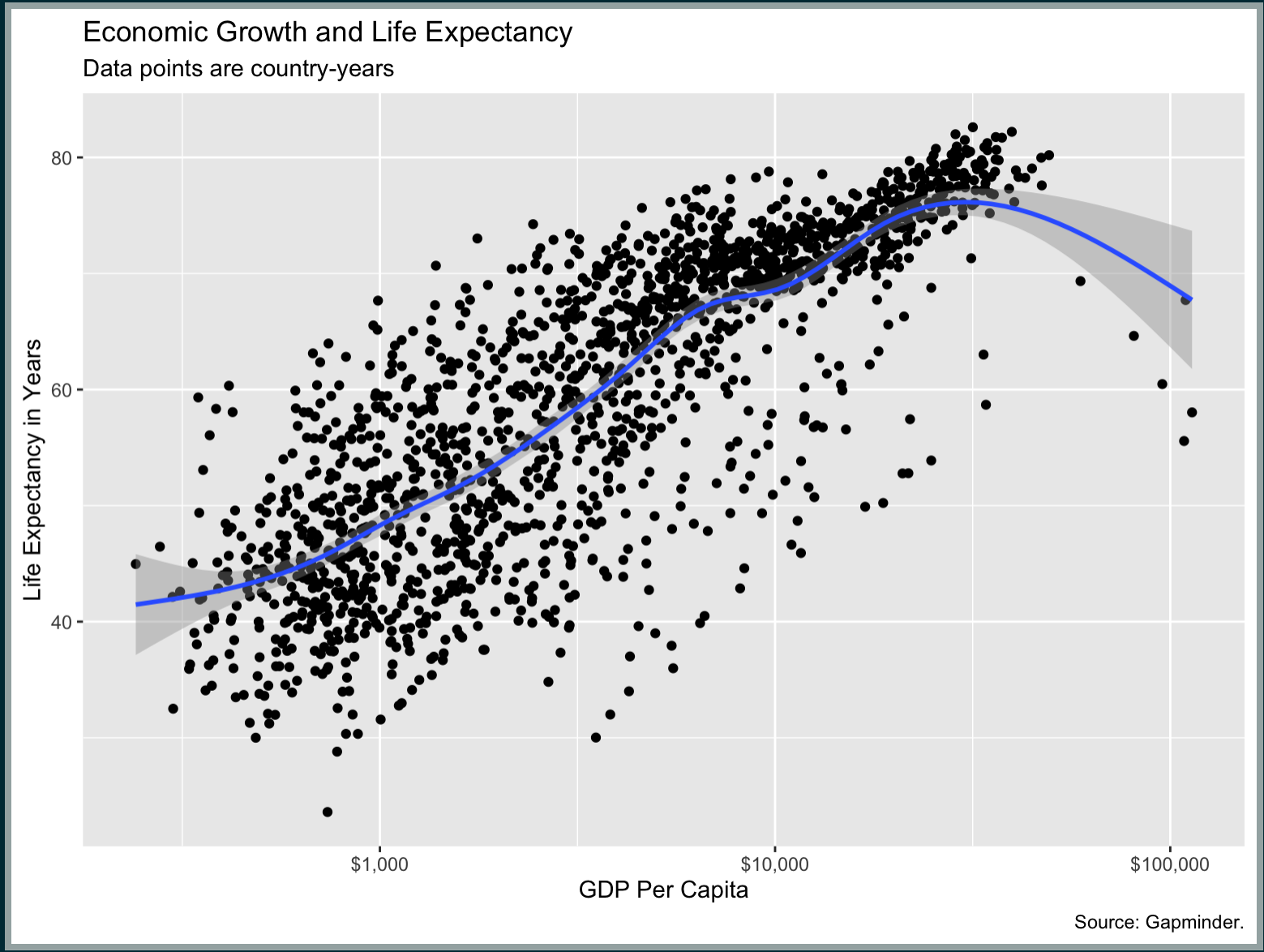


LABELS AND TITLES

We can adjust the axis labels and add plot titles using the `labs()` function:

```
p + geom_point() +
  geom_smooth() +
  scale_x_log10(labels = scales::dollar) +
  labs(x = "GDP Per Capita",
       y = "Life Expectancy in Years",
       title = "Economic Growth and Life Expectancy",
       subtitle = "Data points are country-years",
       caption = "Source: Gapminder.")
```

LABELS AND TITLES



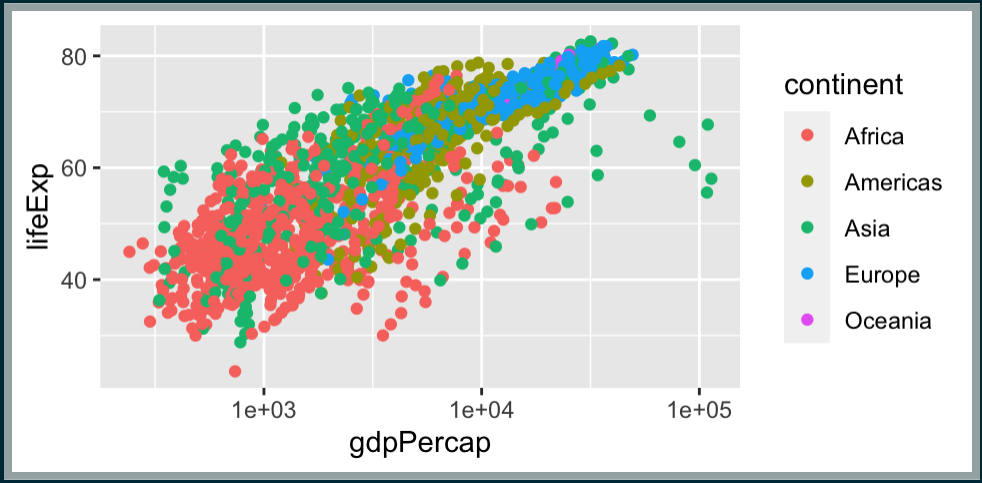
AESTHETICS MAPPING

We can easily map variables in our dataset to aesthetics such size, color, shape, and so on. For example, to map color to `continent`:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent))
```

AESTHETICS MAPPING

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent))
p + geom_point() + scale_x_log10()
```



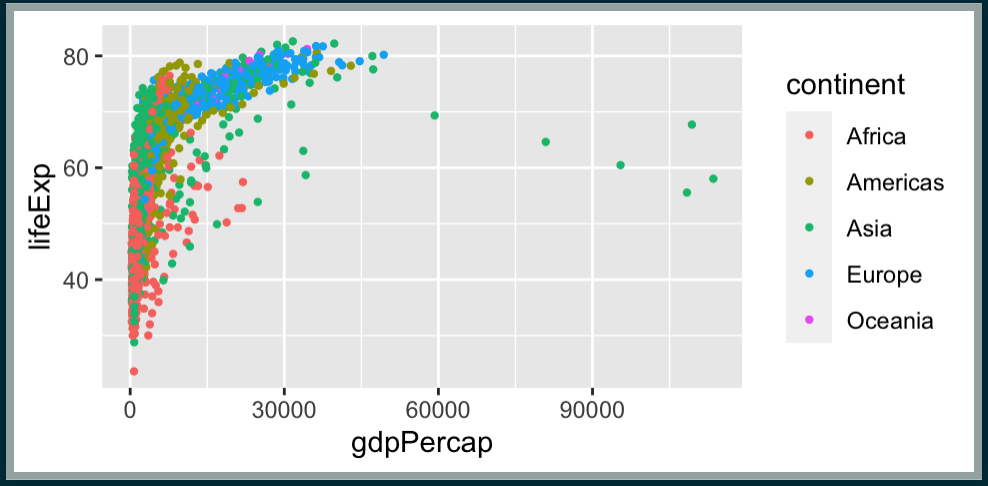
AESTHETIC MAPPINGS

`geom_` can take many other arguments that will affect how the plot looks. Some, such as color and size, have the same name as mappable arguments. Others are specific arguments only available for the `geom_` function. Let's look at a few examples:

AESTHETIC MAPPINGS

Change the size

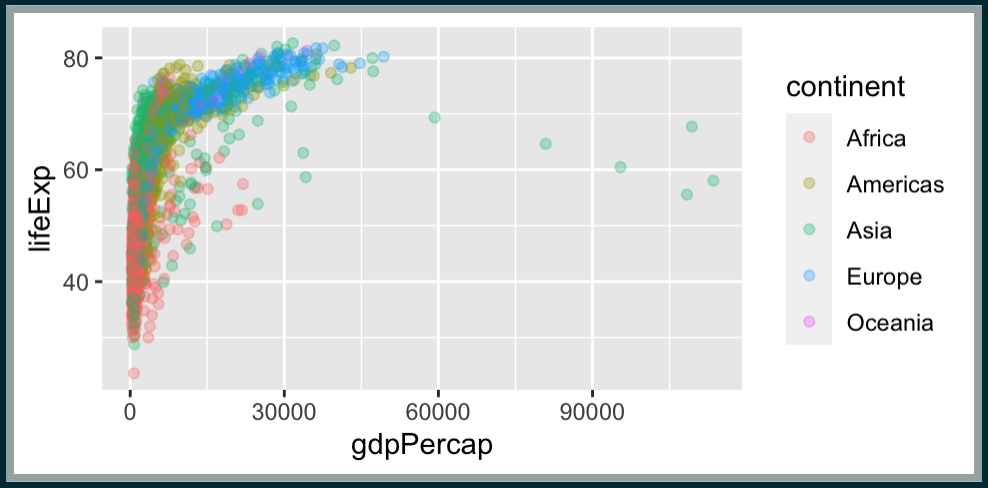
```
p + geom_point(size = 0.8)
```



AESTHETIC MAPPINGS

Adjust transparency

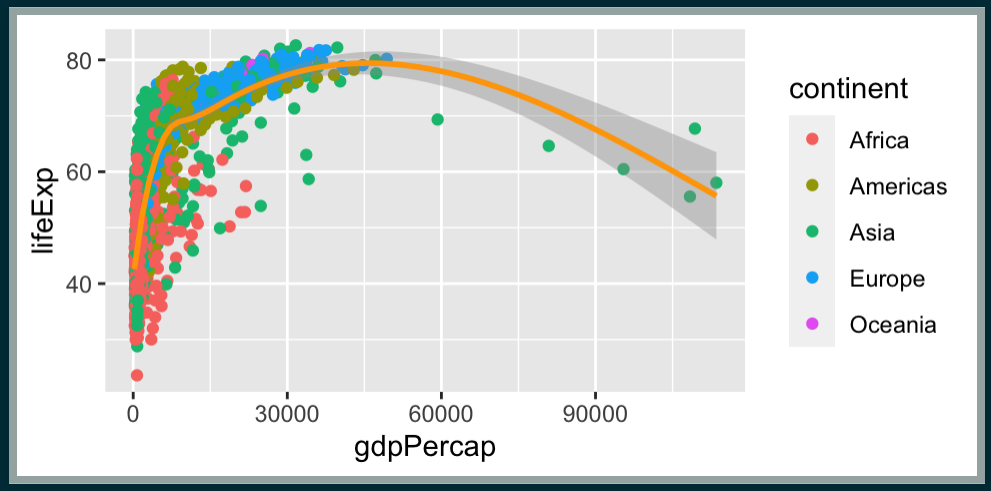
```
p + geom_point(alpha = 0.3)
```



AESTHETIC MAPPINGS

Change the color of the smoother

```
p + geom_point() +
  geom_smooth(color = "orange")
```

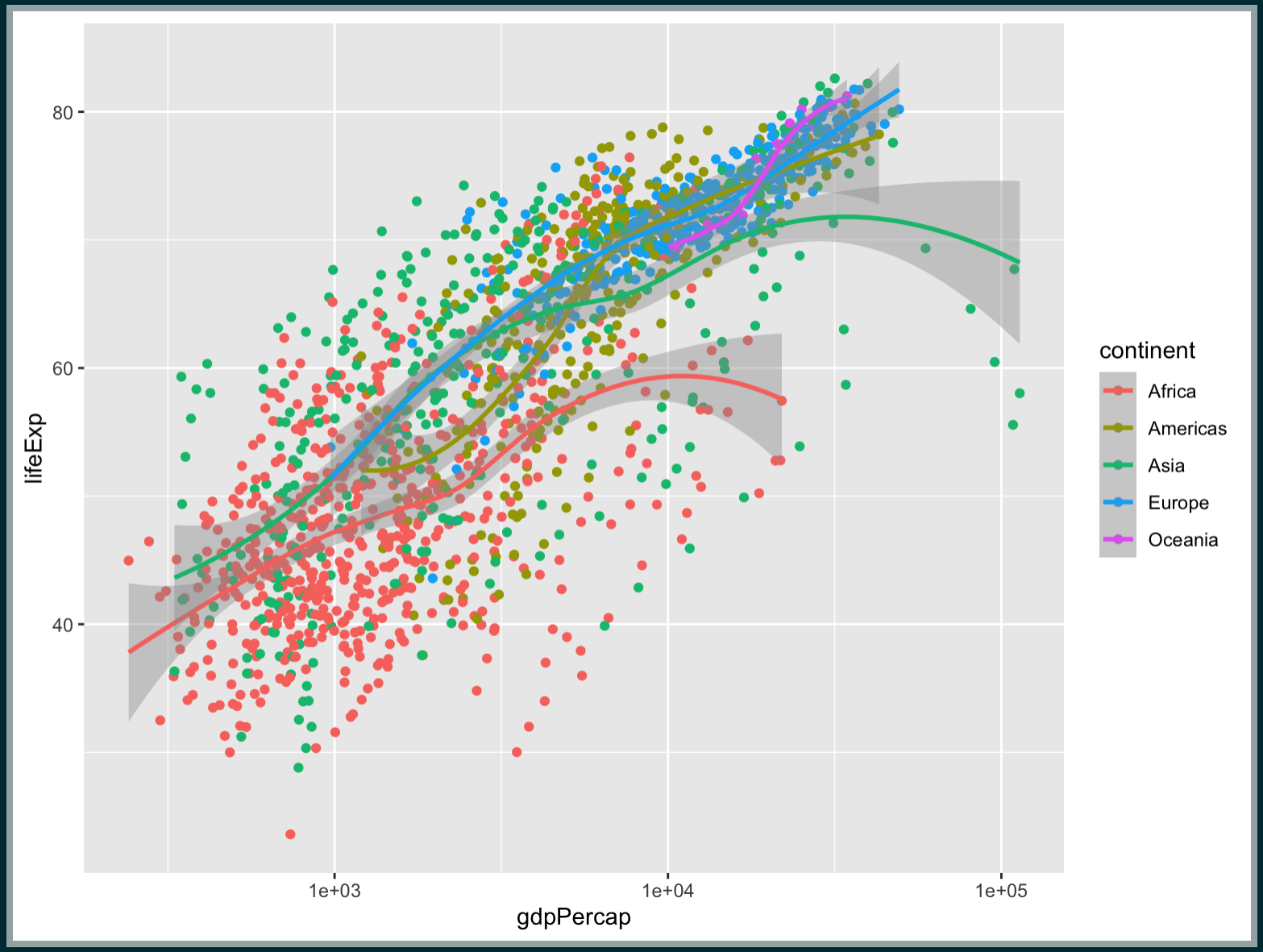


AESTHETIC MAPPINGS PER GEOM

Let's again map our `continent` variable to the color aesthetic. This time we also add a smoother.

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent))
p + geom_point() +
  geom_smooth() +
  scale_x_log10()
```

AESTHETIC MAPPINGS PER GEOM



AESTHETIC MAPPINGS PER GEOM

Both points and smoother are colored by continent. We can use `fill` inside `aes()` to color the interior of the smoother's standard error ribbon:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp,
                          color = continent,
                          fill = continent))

p + geom_point() +
  geom_smooth() +
  scale_x_log10()
```

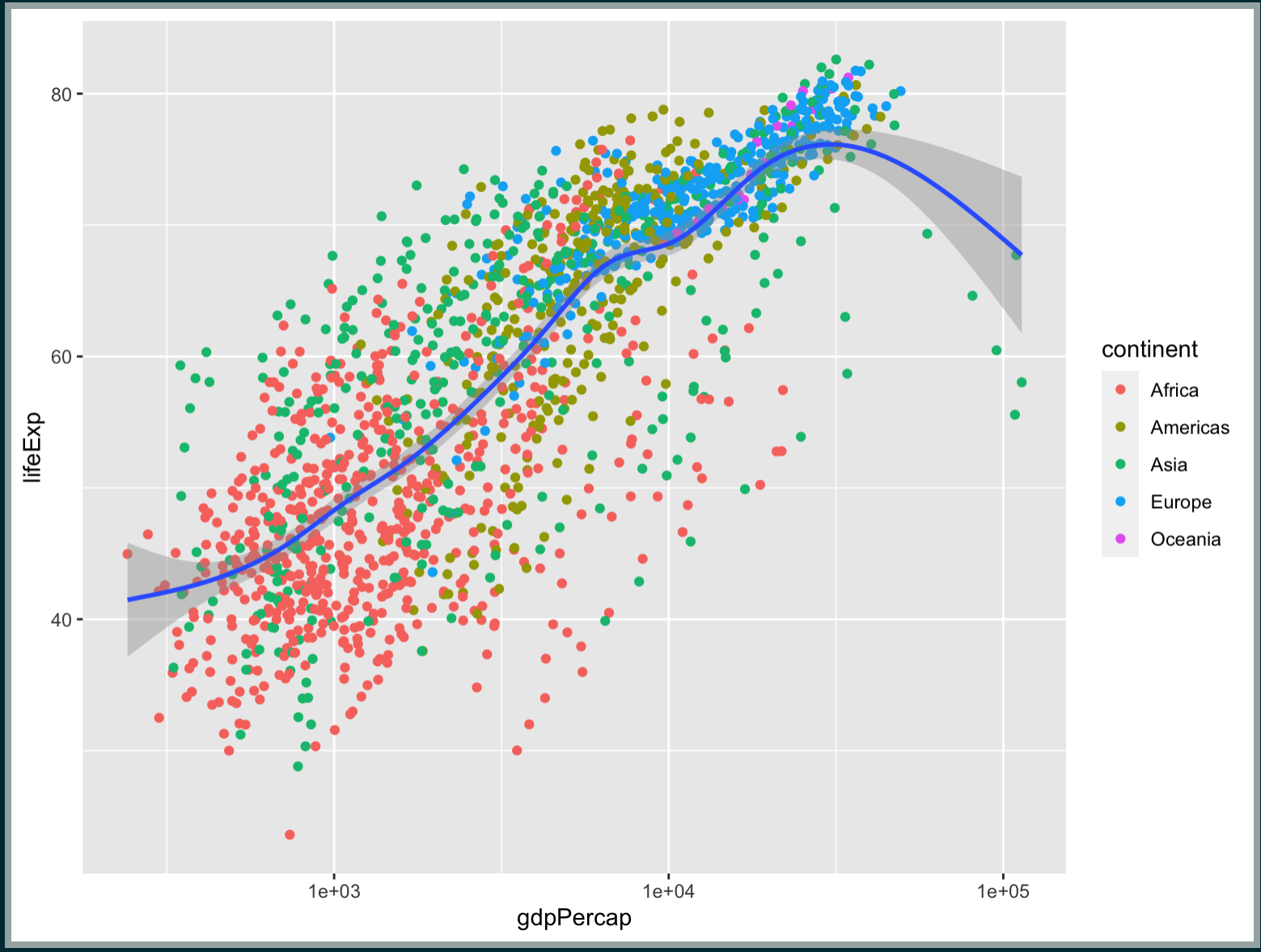
AESTHETIC MAPPINGS PER GEOM

AESTHETIC MAPPINGS PER GEOM

Having 5 different smoothers makes the plot difficult to read. If we just want one line but keep the colored points we can map the aesthetics we want only the `geom_` functions that we want them to apply to:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                          y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
  geom_smooth() +
  scale_x_log10()
```

AESTHETIC MAPPINGS PER GEOM



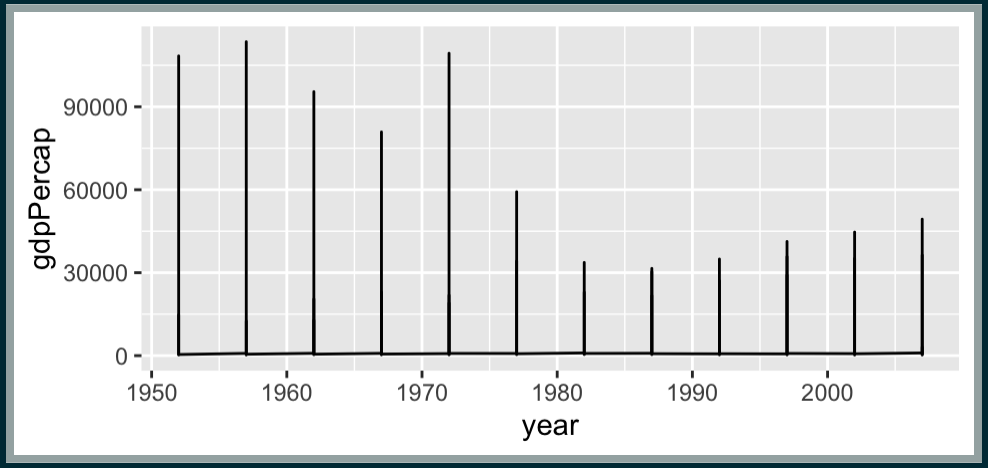
GROUP, FACET, TRANSFORM

There are a number of additional functions in `ggplot` that are frequently used to plot data. `group`, for examples, allows to learn more about the internal structure of your data ().
Let's say we wanted to plot the trajectory of economic development over time for each country. How would we do that?

GROUP

What's gone wrong here?

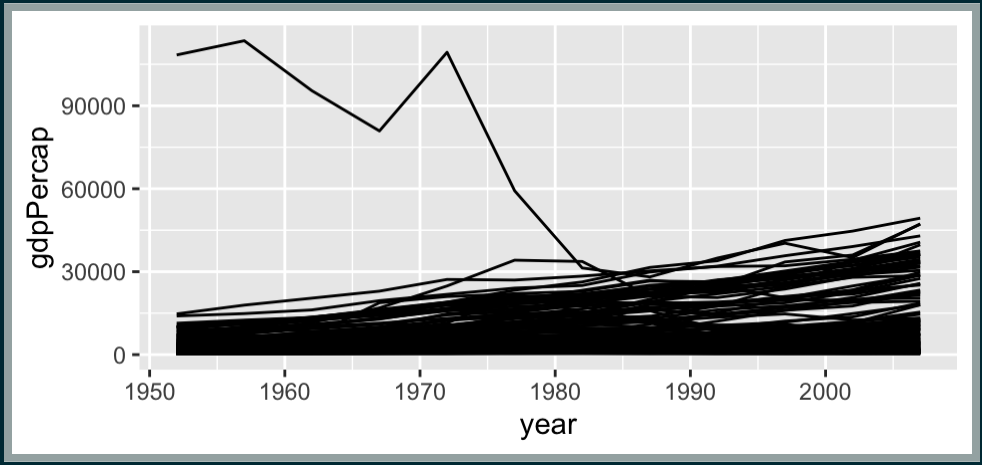
```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line()
```



GROUP

`ggplot` does not know that the yearly observations in the data are grouped by country. We have to tell it:

```
p + geom_line(aes(group = country))
```

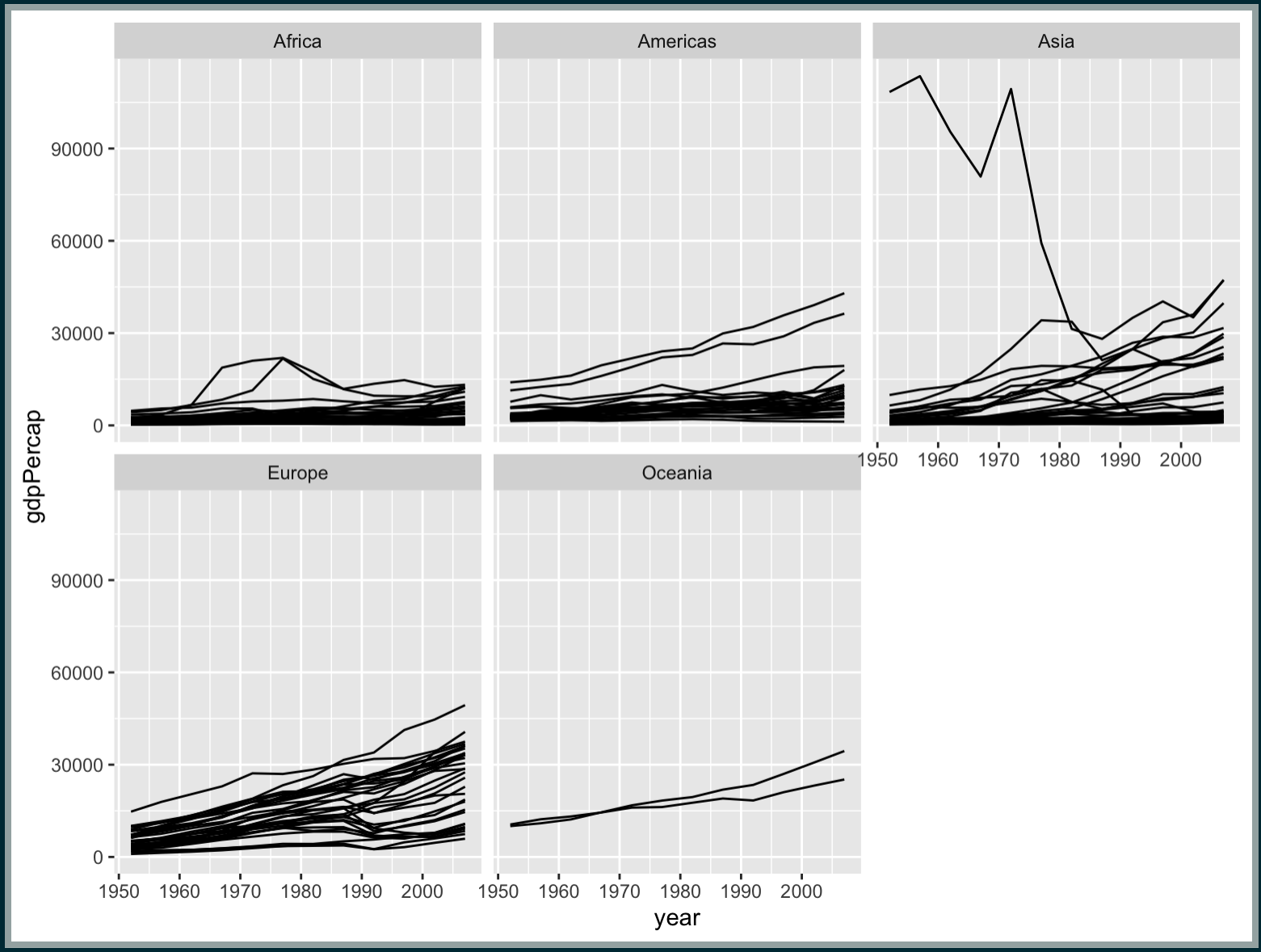


FACET

The plot we just made looks a little messy. To make the trend clearly we could *facet* the data by a third variable and plot the results in separate panels. We use `facet_wrap()` to split our plot by `continent`:

```
p <- ggplot(data = gapminder,
            mapping = aes(x = year,
                          y = gdpPercap))
p + geom_line(aes(group = country)) +
  facet_wrap(~ continent)
```

FACET

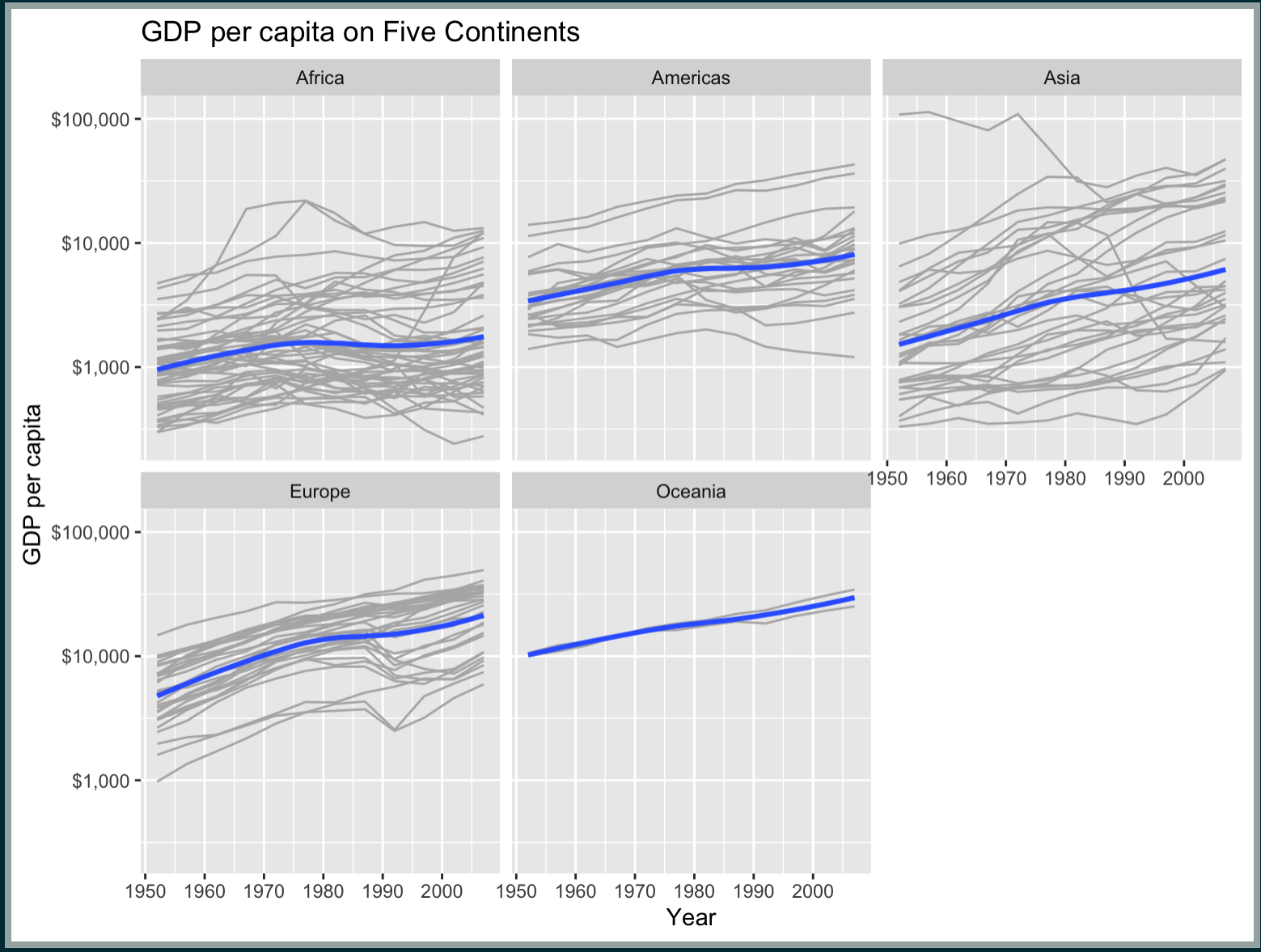

Africa
Americas
Asia
Europe
Oceania
gdpPercap
0
30000
60000
90000
1950
1960
1970
1980
1990
2000
year

FACET

We can also use the `ncol` argument to `facet_wrap()` to control the number of columns. We can add a smoother and a few cosmetic enhancements that make the graph a little more effective. Any geom that we include will be layered within each facet:

```
p + geom_line(color="gray70", aes(group = country)) +
    geom_smooth(size = 1.1, method = "loess", se = FALSE) +
    scale_y_log10(labels=scales::dollar) +
    facet_wrap(~ continent, ncol = 3) +
    labs(x = "Year",
         y = "GDP per capita",
         title = "GDP per capita on Five Continents")
```

FACET

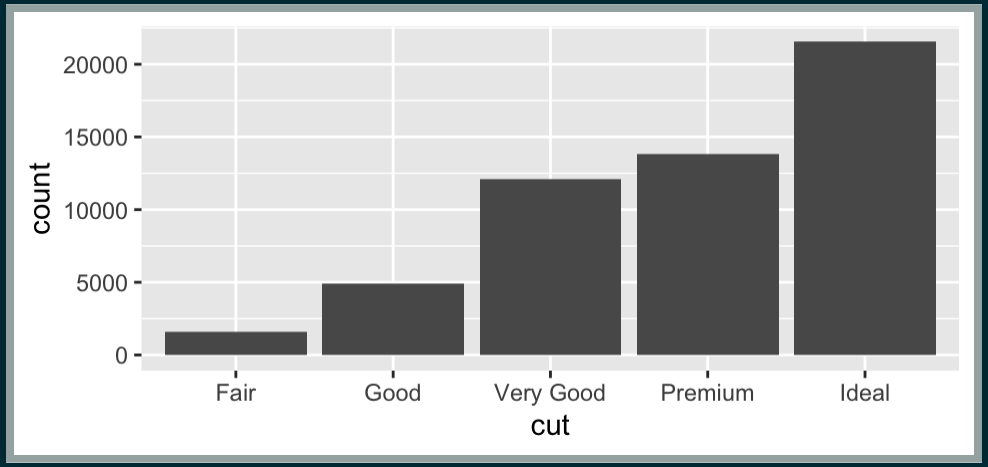

GDP per capita on Five Continents
Africa
Americas
Asia
Europe
Oceania
$100,000
$10,000
$1,000
GDP per capita
1950
1960
1970
1980
1990
2000
Year

FACET

`facet_wrap()` is best used when you want a series of small multiples based on a single categorical variable. If you want to cross-classify some data by two categorical variables you should try `facet_grid()` instead.

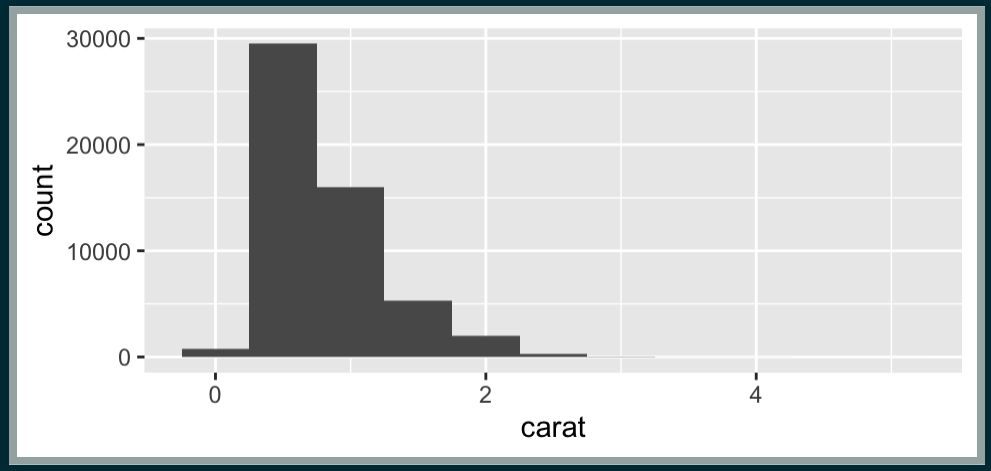
PLOTTING DISTRIBUTIONS: DISTRIBUTIONS OF CATEGORICAL DATA

```
ggplot(data=diamonds) +
  geom_bar(mapping = aes(x = cut))
```



PLOTTING DISTRIBUTIONS: DISTRIBUTIONS OF CONTINUOUS DATA

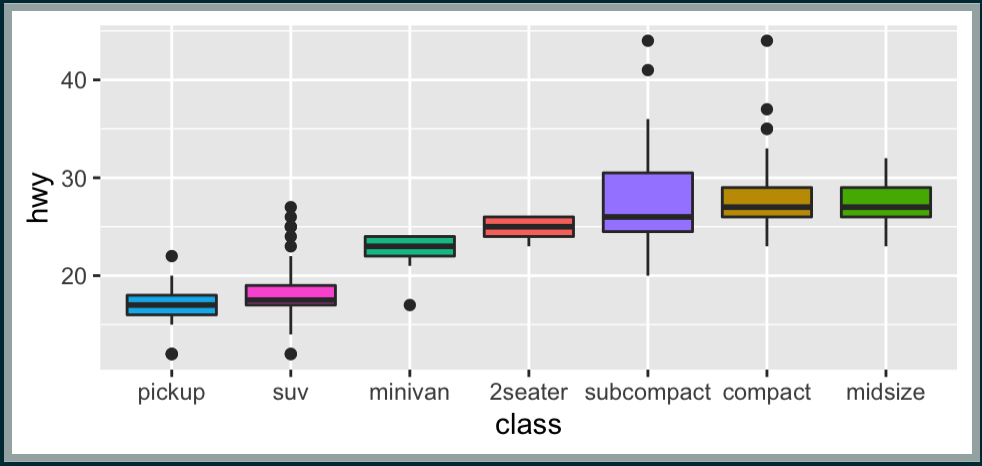
```
ggplot(data=diamonds) +
  geom_histogram(mapping = aes(x = carat),
                 binwidth = 0.5)
```



PLOTTING DISTRIBUTIONS: BOXPLOT

We can use `geom_boxplot()` to plot covariation between continuous and catagorical variables

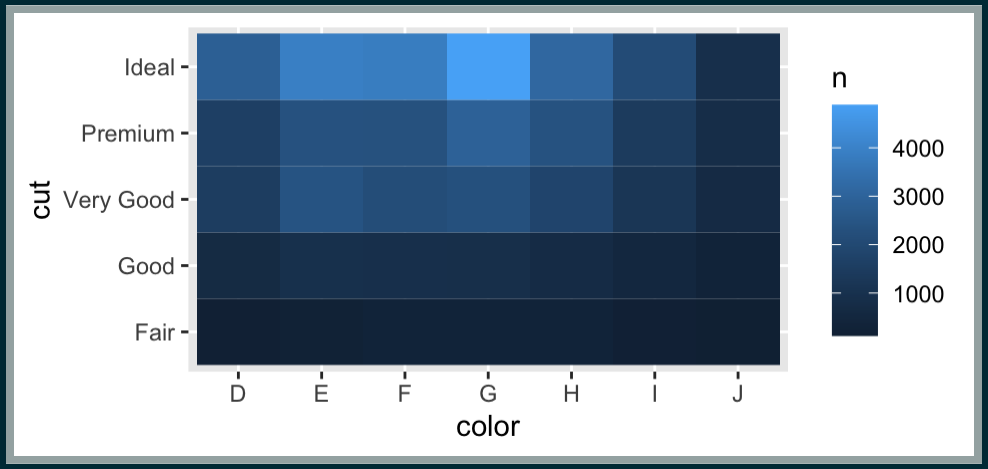
```
ggplot(data = mpg, aes(x = class, y = hwy, fill = class)) +
  geom_boxplot(aes(x=reorder(class, hwy,FUN = median),
                   y = hwy)) +
  theme(legend.position = "none")
```



PLOTTING DISTRIBUTIONS: TILE PLOT

We can use `geom_tile` to plot the covariation between two categorical variables

```
diamonds %>%
  count(color, cut) %>%
  ggplot(mapping = aes(x = color, y = cut)) +
    geom_tile(mapping = aes(fill = n))
```



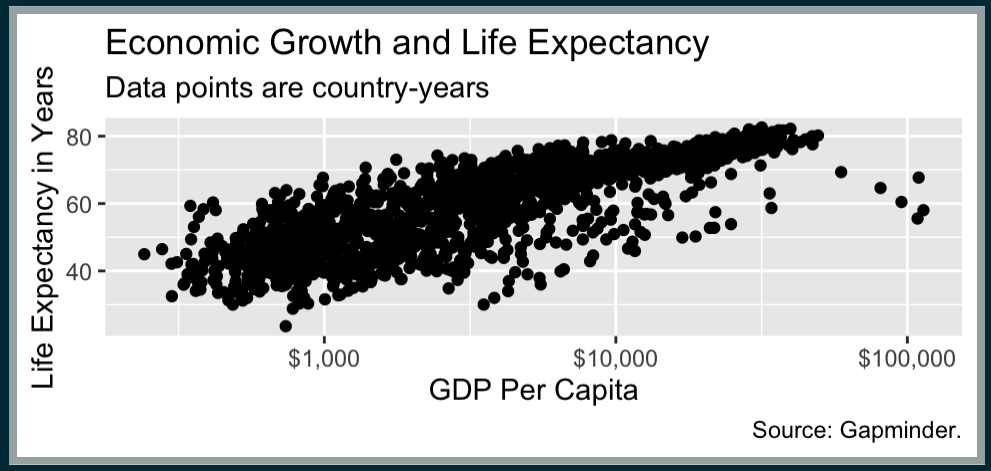
PLOTTING DISTRIBUTIONS: SCATTER PLOTS

The easiest way to visualize the covariation between two continuous variables is to draw a scatterplot with `geom_point()`.

```
p <- ggplot(data=gapminder, mapping = aes(x = gdpPercap,
                                          y = lifeExp)) +
  geom_point() +
  scale_x_log10(labels = scales::dollar) +
  labs(x = "GDP Per Capita",
       y = "Life Expectancy in Years",
       title = "Economic Growth and Life Expectancy",
       subtitle = "Data points are country-years",
       caption = "Source: Gapminder.")
```

PLOTTING DISTRIBUTIONS: SCATTER PLOTS

```
p
```



SAVING AND EXPORTING GGPLOT OBJECTS

`ggsave()` is a convenient function for saving the last plot that you displayed. It also guesses the type of graphics device from the extension. This means the only argument you need to supply is the filename.

```
ggsave("myplot.pdf") # save as pdf
ggsave("myplot.png", width = 4, height = 4) # save as png
```

MORE THEME PACKAGES

- ggpubr for publication-ready plots
- ggthemes for additional themes and scales, especially ones that match other software (Tableau)
- hrbrthemes my personal favorite theme package

GGPLOT2 RESOURCES

- General ggplot2 information
- R Graphics Cookbook
- Data Visualization: A Practical Introduction
- ggplot2: Elegant Graphs for Data Analysis
- RStudio visual data primer

VISUALIZING TEXT

LOADING & TOKENIZING MOBY DICK

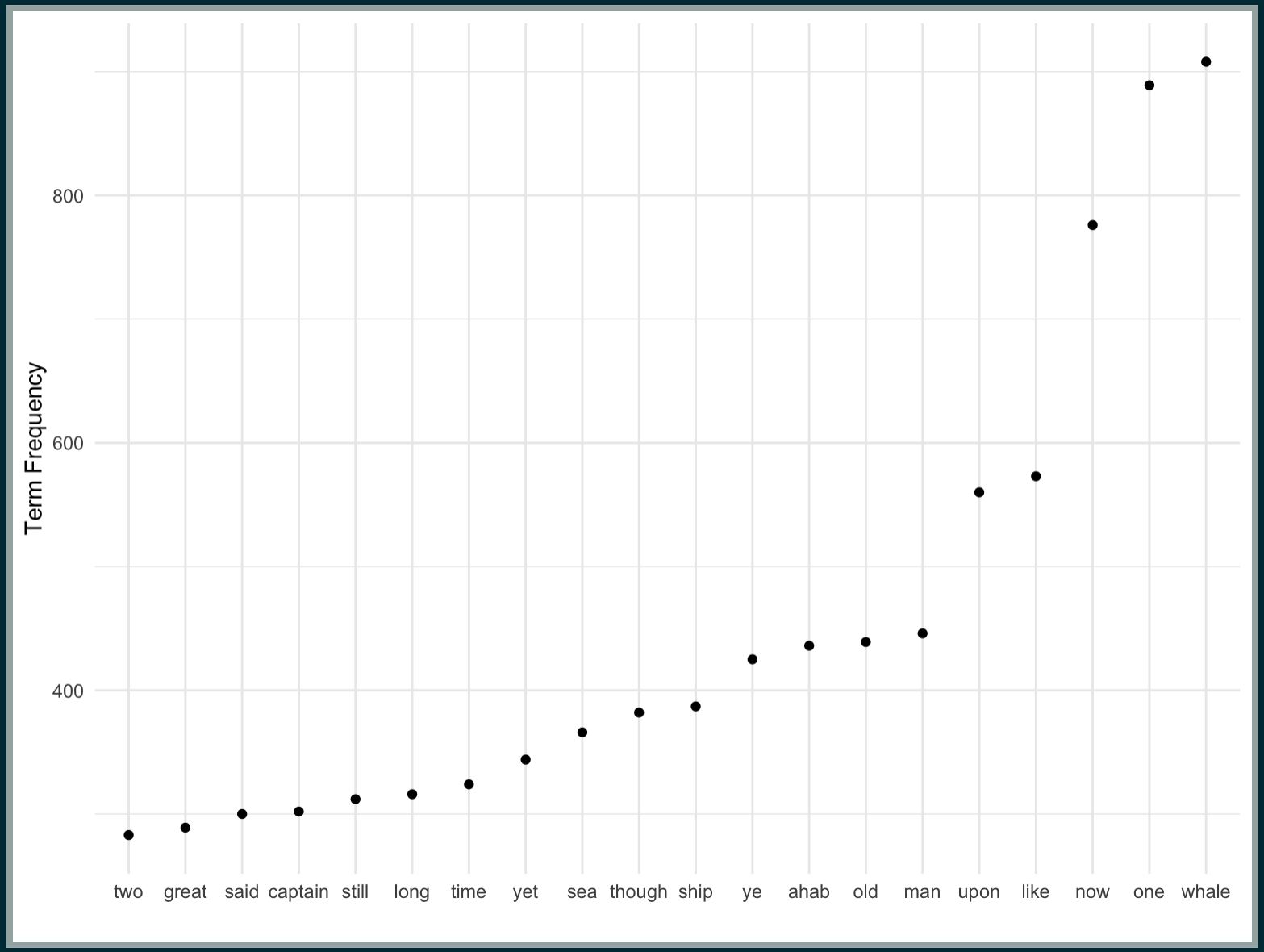
```
library(readtext)
moby_dick <- readtext("data/moby_dick.txt")
moby_dick_tokens <- tokens(moby_dick$text, what = "word",
                           remove_punct = TRUE) %>%
  tokens_tolower() %>%
  tokens_remove(stopwords("english"))
moby_dfm <- dfm(moby_dick_tokens)
```

PLOT FREQUENT WORDS

To plot the frequency of often used words in the novel you first need to install and load the `quanteda.textstats`library.

```
library(quanteda.textstats)
theme_set(theme_minimal())
textstat_frequency(moby_dfm, n = 20) %>%
  ggplot(aes(x = reorder(feature, -rank), y = frequency)) +
  geom_point() +
  labs(x = "", y = "Term Frequency")
```

PLOT FREQUENT WORDS



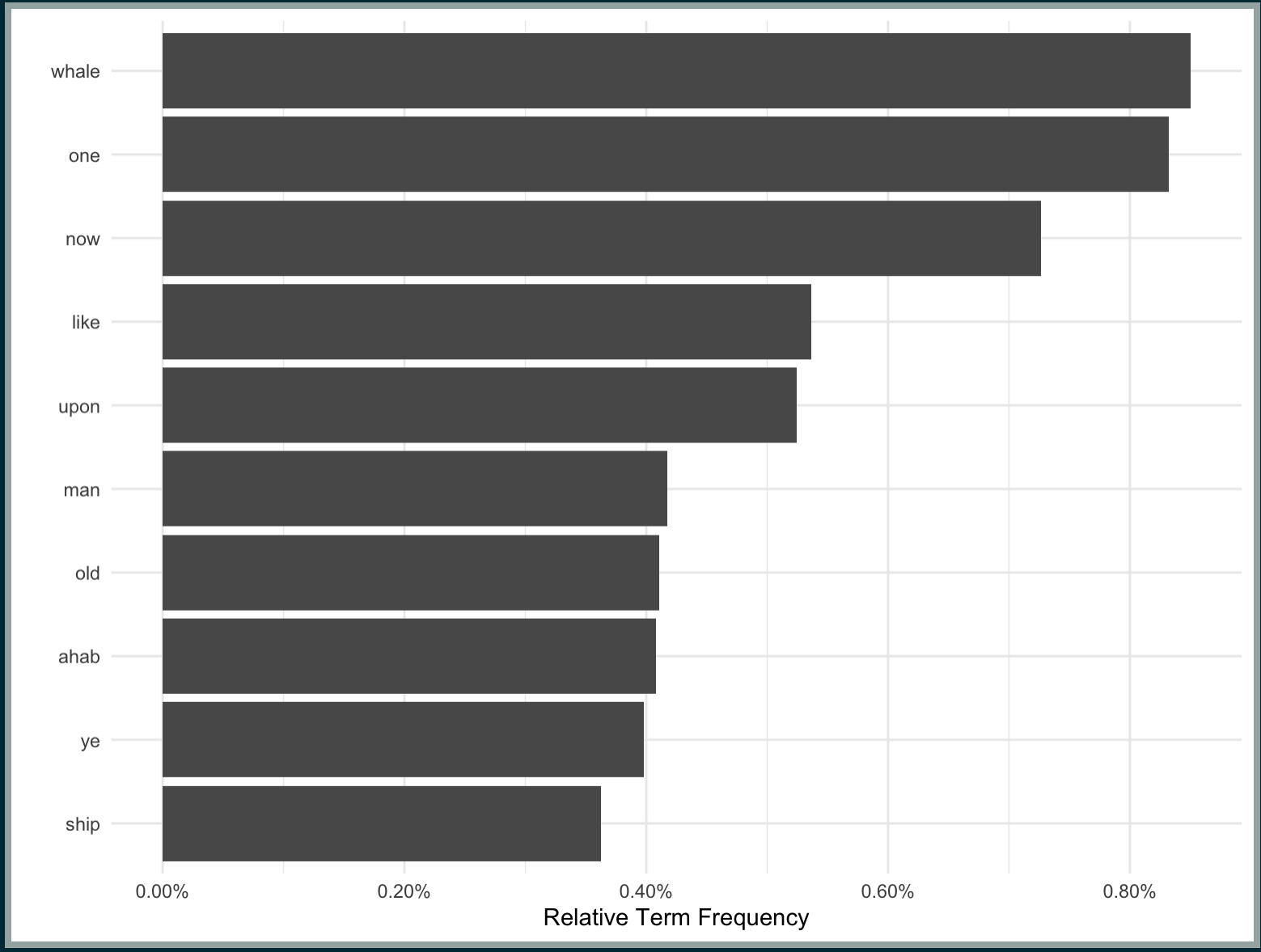
PLOT RELATIVE WORD FREQUENCIES

The total number of occurrences of a word is not very useful. To plot the frequency as a percentage instead we first need to created a new dfm with the `dfm_weight` function and `scheme = "prop"`.

```
moby_dfm_pct <- dfm_weight(moby_dfm, scheme = "prop")
```

```
textstat_frequency(moby_dfm_pct, n = 10) %>%
  ggplot(aes(x = reorder(feature, -rank), y = frequency)) +
  geom_col() + coord_flip() +
  scale_y_continuous(labels = scales::percent) +
  labs(x = "", y = "Relative Term Frequency")
```

PLOT RELATIVE WORD FREQUENCIES



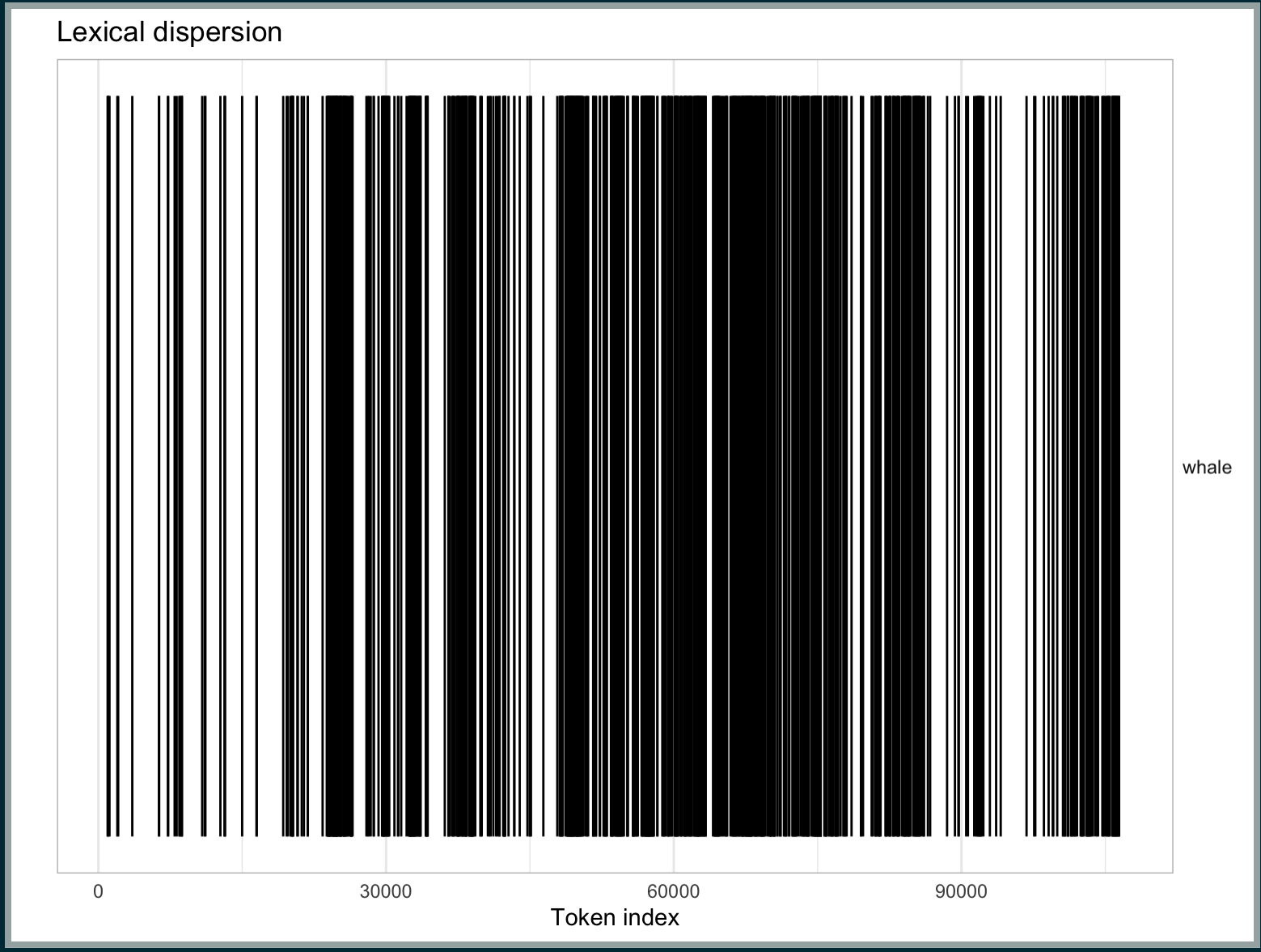
LEXICAL DISPERSION

Quanteda also has a number of plotting functions that are based on ggplot2 and are contained inside the `quanteda.textplots` package.

For example, to visualize the occurrences of certain words throughout the novel we can create a dispersion plot. The great thing is that we can create the plot directly from the object returned by the `kwic()` function.

```
library("quanteda.textplots")
textplot_xray(kwic(moby_dick_tokens, pattern = "whale")) +
    ggtitle("Lexical dispersion")
```

LEXICAL DISPERSION

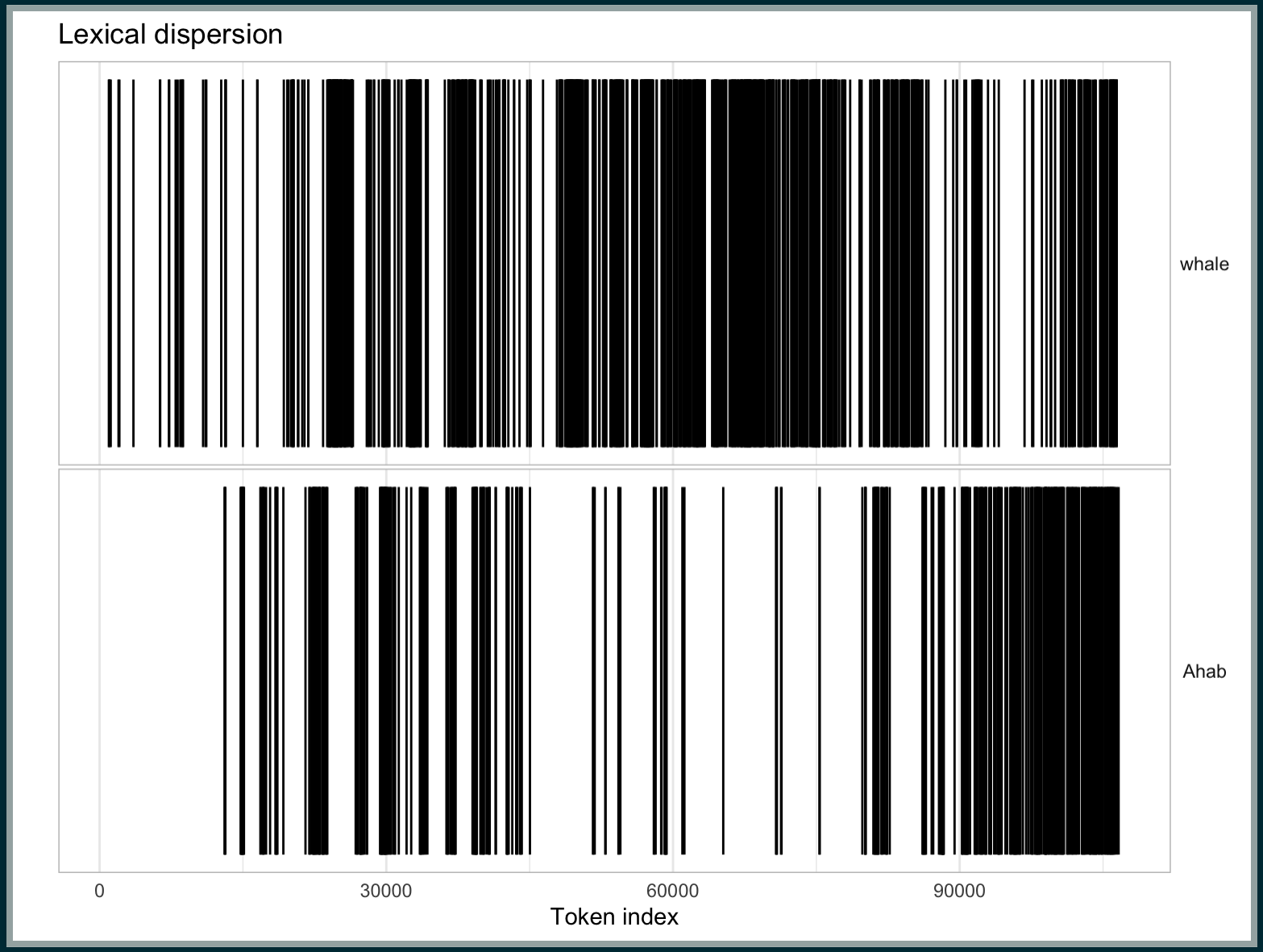


LEXICAL DISPERSION

We can also create dispersion plots for multiple words:

```
textplot_xray(
    kwic(moby_dick_tokens, pattern = "whale"),
    kwic(moby_dick_tokens, pattern = "Ahab")) +
    ggtitle("Lexical dispersion")
```

LEXICAL DISPERSION



ANALYZING CHAPTERS

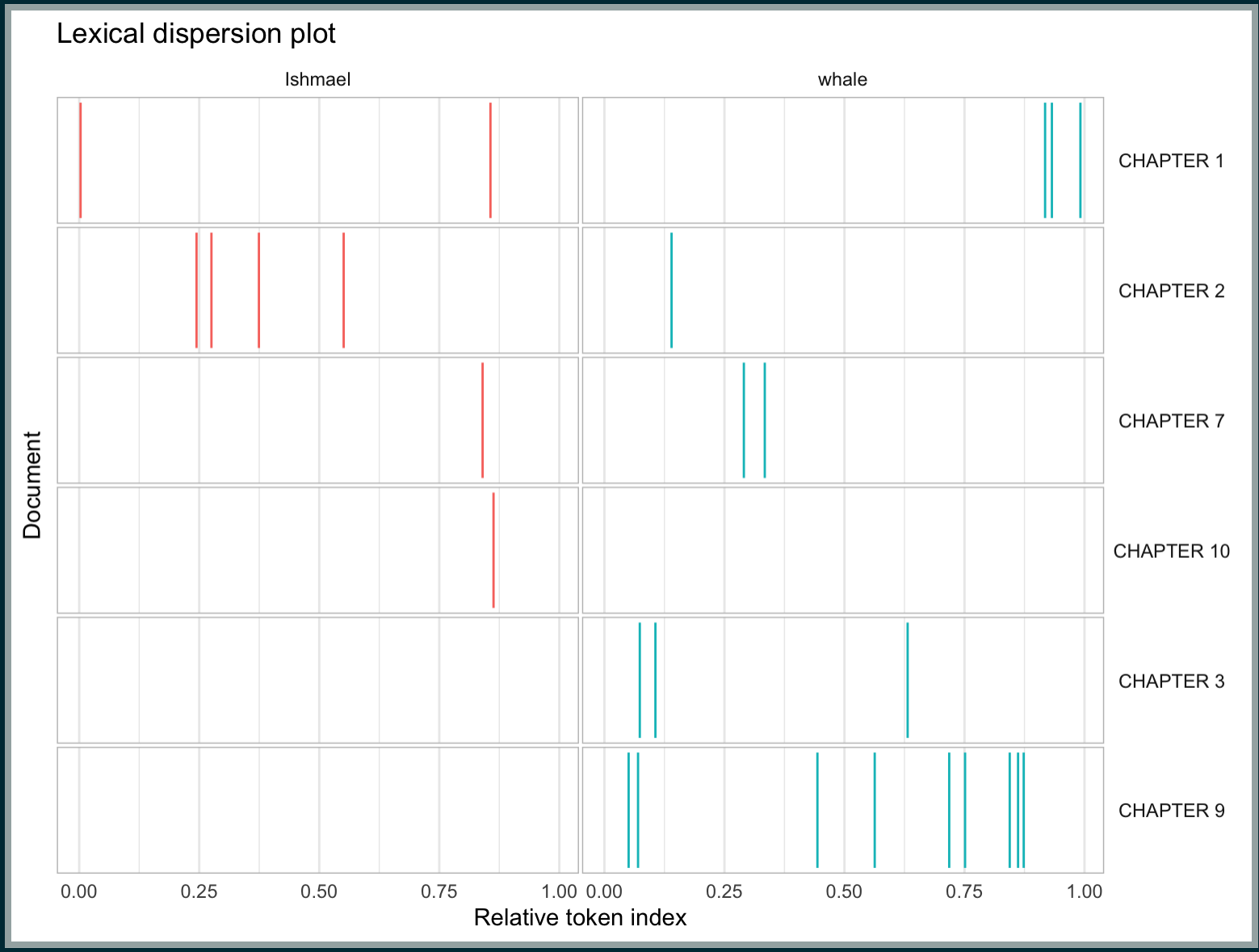
If we are interested in analyzing the book's individual chapters in more detail we first need to create a corpus object and supply them as document names.

```
moby_chapter <- corpus(moby_dick) %>%
  corpus_segment(pattern = "CHAPTER \\d+", valuetype = "regex")
docnames(moby_chapter) <- docvars(moby_chapter, "pattern")
moby_chapter_tokens <- tokens(moby_chapter, remove_punct = TRUE) %>%
  tokens_tolower() %>%
  tokens_remove(stopwords("english"))
```

ANALYZING CHAPTERS

```
textplot_xray(
    kwic(moby_chapter_tokens[1:10], pattern = "Ishmael"),
    kwic(moby_chapter_tokens[1:10], pattern = "whale")) +
  aes(color = keyword) +  theme(legend.position = "none")
```

ANALYZING CHAPTERS

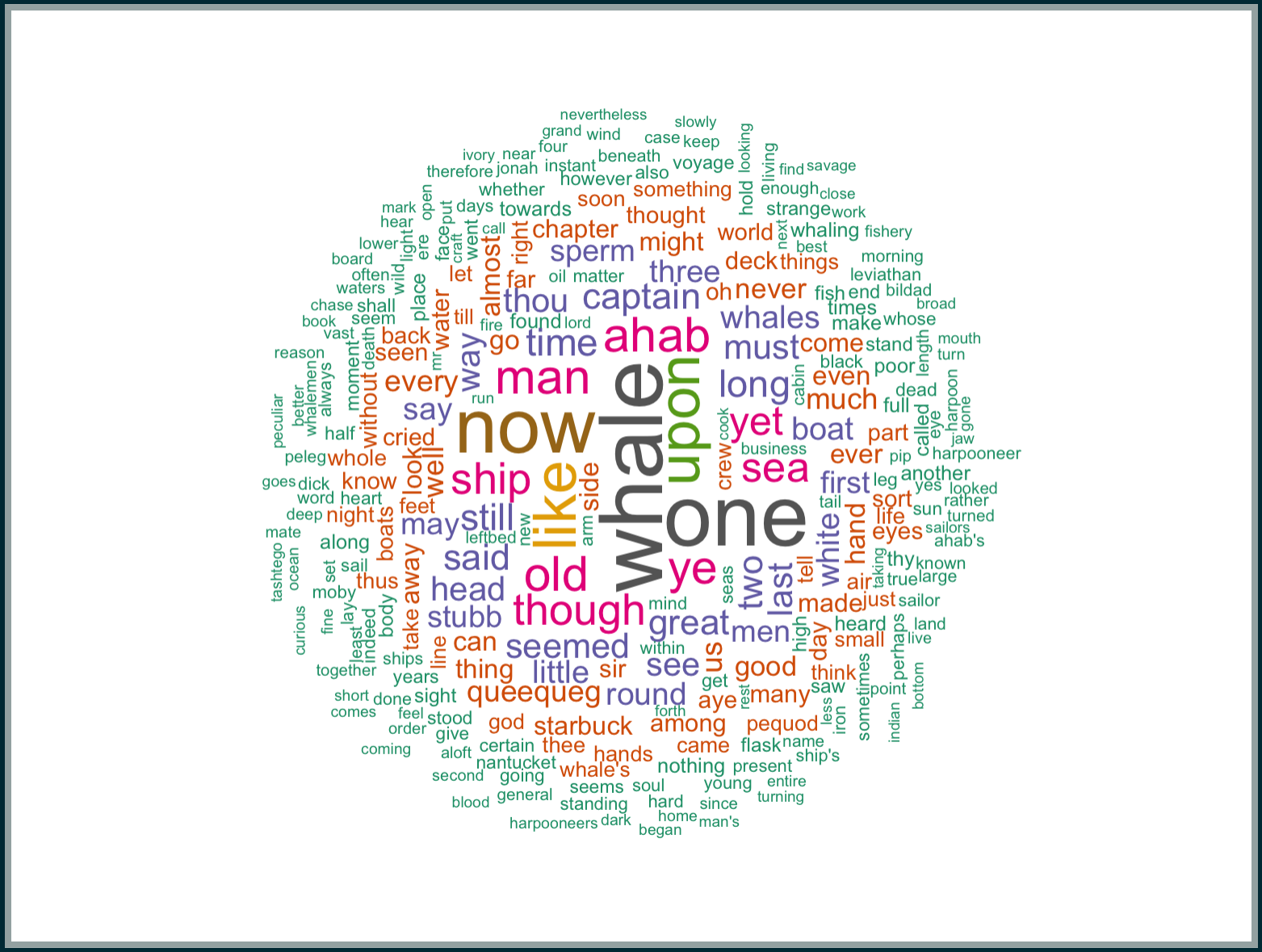


PLOTTING WORD CLOUDS

Although there are a lot of things wrong with wordclouds you can easily create them with the `textplot_wordcloud` function.

```
textplot_wordcloud(moby_dfm, min_count = 50, random_order = FALSE,
                   rotation = .25,
                   color = RColorBrewer::brewer.pal(8,"Dark2"))
```

PLOTTING WORD CLOUDS

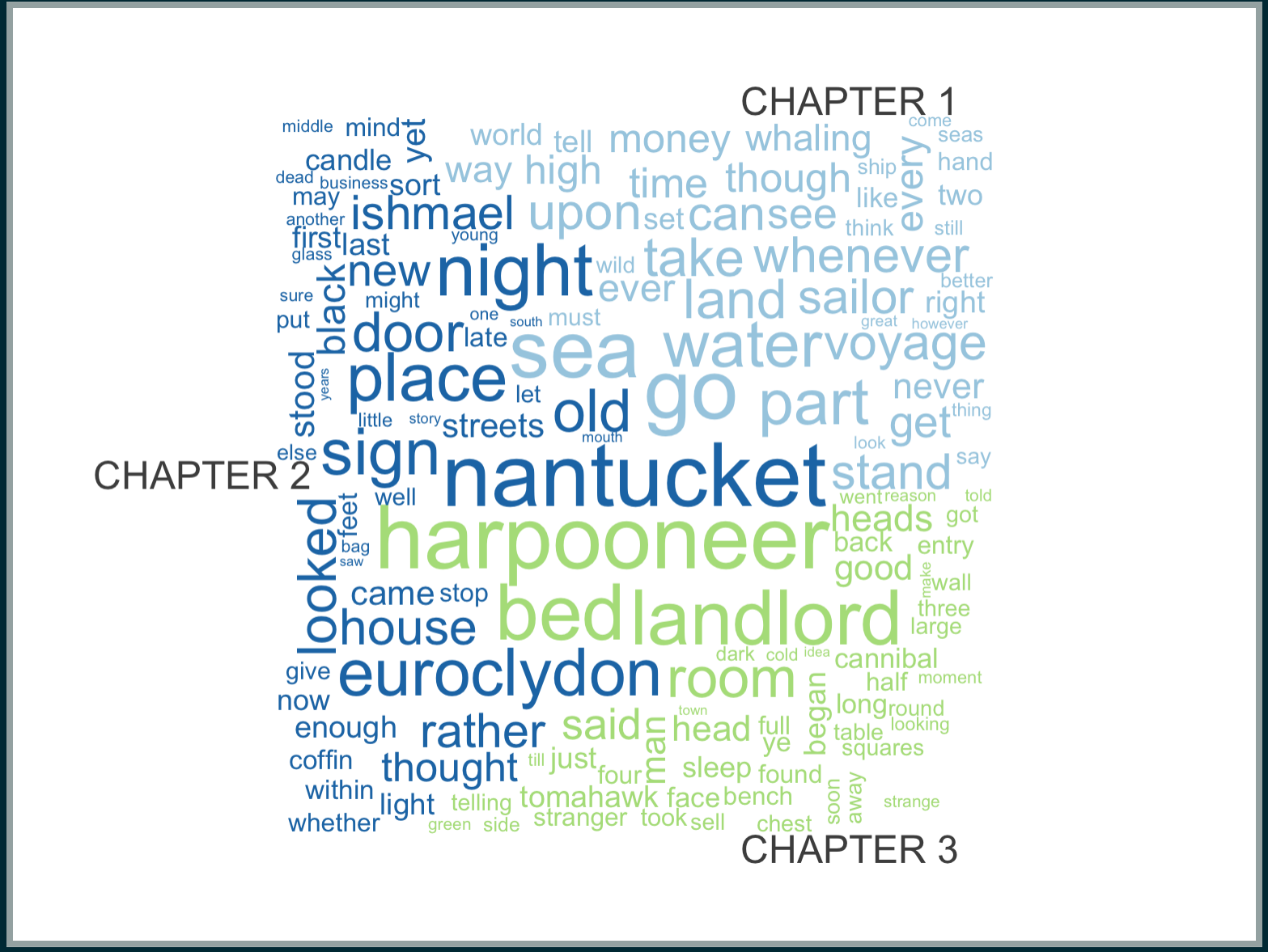


PLOTTING WORD CLOUDS

You can even create grouped wordclouds using `dfm_group`.

```
moby_chapter_tokens[1:3] %>%
  dfm() %>%
  dfm_group(groups = pattern) %>%
  dfm_trim(min_termfreq = 5, verbose = FALSE) %>%
  textplot_wordcloud(comparison = TRUE)
```

PLOTTING WORD CLOUDS



WRAPPING UP

QUESTIONS?

OUTLOOK FOR OUR NEXT SESSION

- Next week we'll learn how to create custom dictionaries to label and cluster texts.

THAT'S IT FOR TODAY

Thanks for your attention!