# Introduction to R

## Lecture 1: Getting (slowly) Started

Swapnil Singh
Lietuvos Bankas | Course Link

# Lecture's Objectives

1. Why this course?
2. Getting started
3. What R can do?
4. Basic Introduction

# Why this course?

# What is R?

- R is a language and environment for statistical computing and graphics.
- **Open-source** and extensible.
- Built upon *S*, a statistical programming language.
- Rich set of packages for data manipulation (e.g., `tidyverse` environment build upon `tidyr`, `dplyr`, etc.).
- Comprehensive plotting libraries (e.g., `ggplot2`, `lattice`).
- Wide array of statistical tests.
- Facilitates machine learning algorithms.
- Excellent packages (e.g., `sf`, `terra`, etc.) for spatial datasets and GIS applications

# Comparison with other languages

## R vs Python

- R: More focused on statistical modeling and data visualization.
- R: Weaker on web-scraping
- Python: More general-purpose but has strong data science libraries (e.g., `pandas`, `scikit-learn`).

## R vs MATLAB

- R: Open-source and has a larger community for data science.
- MATLAB: Stronger in numerical simulation but less versatile for data manipulation.

# R vs STATA

## Flexibility and Extensibility

- **R**: Highly extensible through packages; good for custom statistical methods.
- **STATA**: More rigid but user-friendly for standard statistical tests.

## Data Handling

- **R**: More versatile data manipulation capabilities (`dplyr`, `tidyr`).
- **STATA**: Efficient for large datasets but less flexibility.

## Graphics and Visualization

- **R**: Advanced graphical capabilities (`ggplot2`).
- **STATA**: Basic graphs are easier to produce but less customizable.

## Pricing and Community

- **R**: Open-source, large and active community.
- **STATA**: Commercial software, smaller community focused on social sciences.

# So, Why This Course?

- Provide a basic introduction to R
- In future, extend the course to advance level (*depending upon demand*)
  - Causal inference methods
  - Big data (`vroom`)
  - Webscraping
  - Geospatial analysis
  - ...
- Many of these topics are not directly relevant for your specific work, but,

> **Knowledge is power!**

# Getting Started

# What you need?

## For this course:

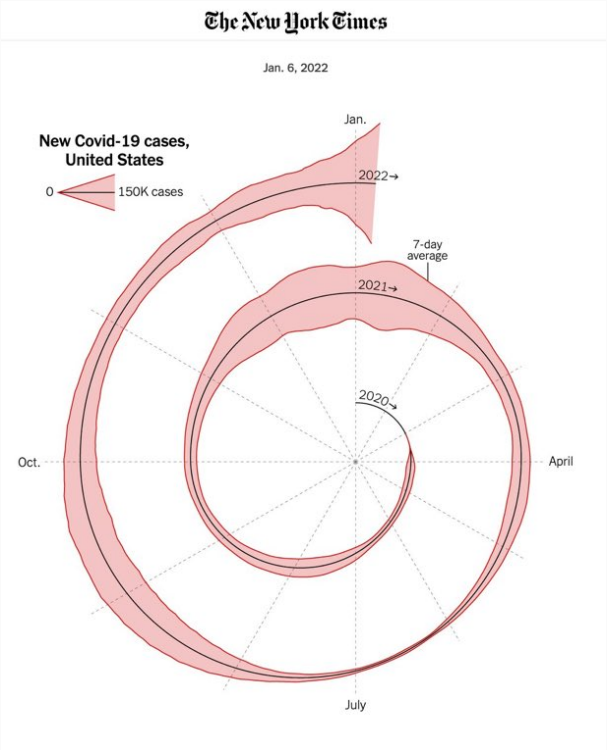
- Install `R`
- Install `RStudio`

## Additional components

- Install `Git`
- Create a `Github` account. Alternatively, you can also create account on `Gitlab` or `Gitbucket`, but `Github` is the most used one
- If working only with `RStudio`, follow this website to get things working: http://happygitwithr.com
- For general purpose use of `Git`, I will suggest to use `GitKraken`

# What R Can Do?

# Let's create some figures!




NY Times, 2022-01-06, Covid Cases Spiral, US

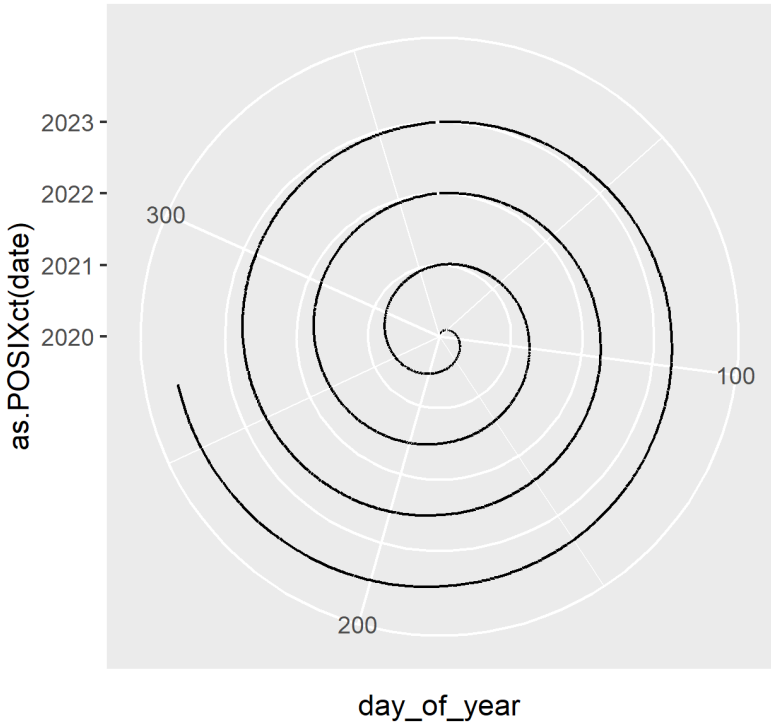
# Let's create some figures!

## Hats off to ByData blog!

```
pacman::p_load("tidyverse", "ggtext", "here", "lubridate")
owid_url <- "https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-
covid <- suppressMessages(read_csv(owid_url))

# get the data only for the US
covid_cases <- covid |>
  dplyr::filter(location == 'United States') |>
  dplyr::select(date, new_cases, new_cases_smoothed) |>
  dplyr::arrange(date) |>


  # add some additional data to complete the year
  dplyr::add_row(date = as_date("2020-01-01"), new_cases = 0, new_cases_smoothed = 0,
          .before = 1) %>%
  tidyr::complete(date = seq(min(.$date), max(.$date), by = 1),
           fill = list(new_cases = 0, new_cases_smoothed = 0)) %>%
  dplyr::mutate(day_of_year = yday(date),
         year = year(date)
         )
```

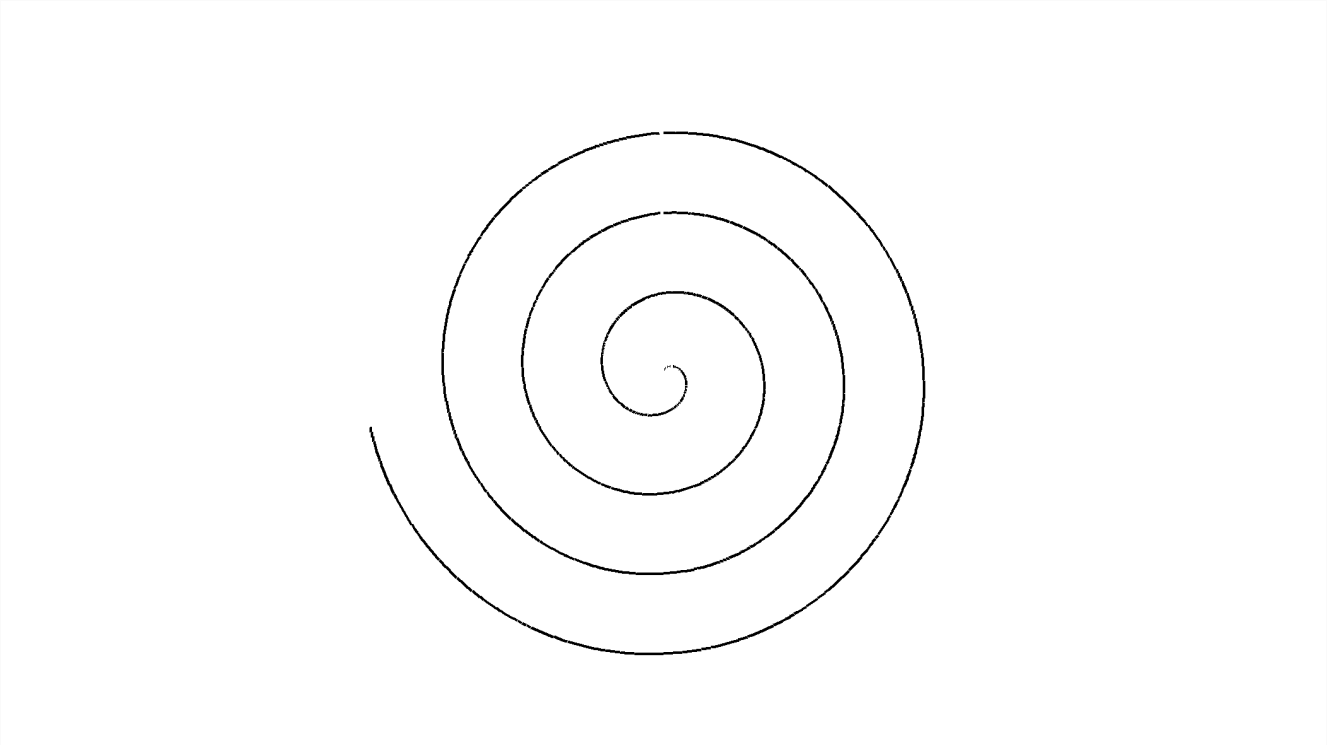
# How a basic plot looks like?

```
basic.plot ← covid_cases ▷
  ggplot() +
  geom_segment(aes(x = day_of_year, xend = day_of_year + 1,
                   y = as.POSIXct(date), yend = as.POSIXct(date))) +
  coord_polar()
basic.plot
```



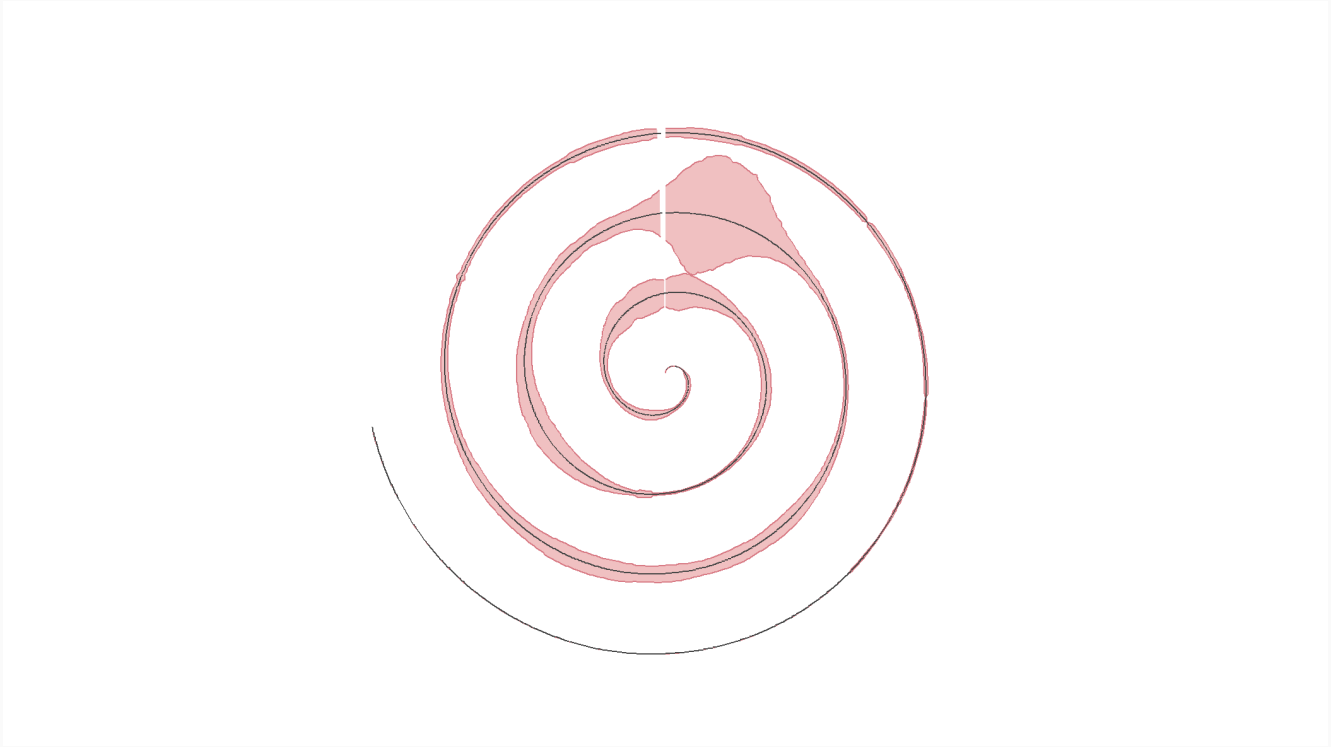
# How a basic plot looks like?

```
basic.plot + theme_void()
```



# Moving onto advanced level now!

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



# Moving onto advanced level now!

```
month_length ← c(31, 28, 31, 30, 31, 30,
                 31, 31, 30, 31, 30, 31)

month_breaks ← cumsum(month_length) - 30

basic.plot ←  basic.plot + scale_x_continuous(minor_breaks = month_breaks,
                     breaks = month_breaks[c(1, 4, 7, 10)],
                     labels = c("Jan.", "April", "July", "Oct.")) +
  theme(
    plot.background = element_rect(color = NA, fill = "white"),
    panel.grid.major.x = element_line(color = "grey70", size = 0.2, linetype = "dottec
    panel.grid.minor.x = element_line(color = "grey70", size = 0.2, linetype = "dottec
    axis.text.x = element_text(color = base_grey, size = 5, hjust = 0.5),
  )
```

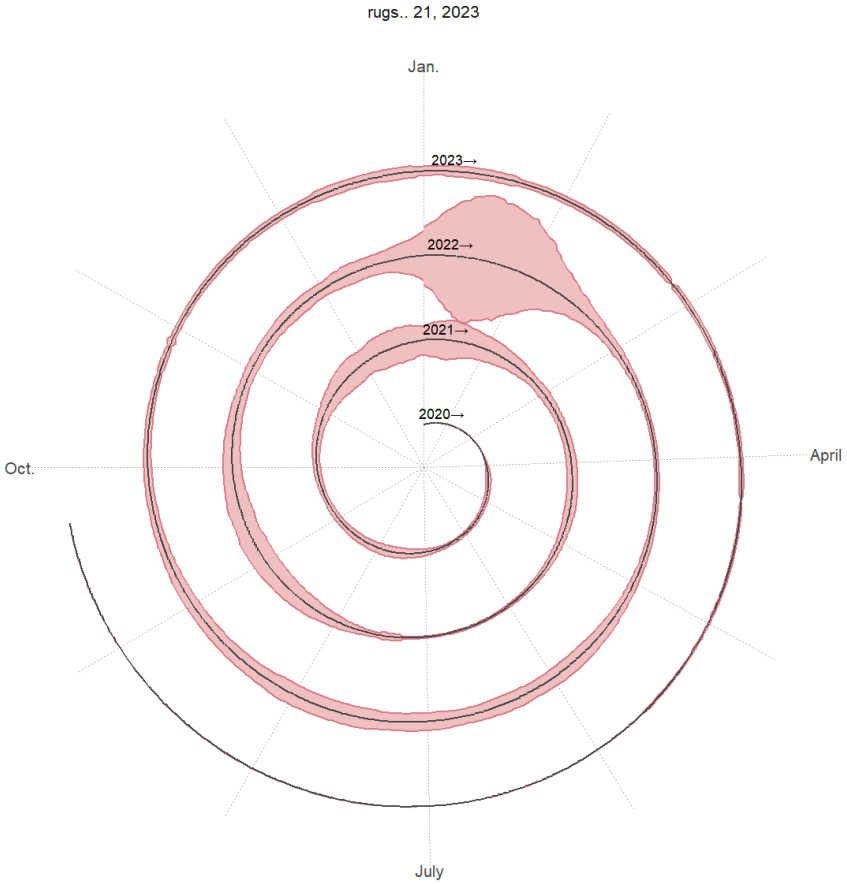
```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## ℹ Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

# Moving onto advanced level now!

- One truth about generating good figures: last 10 percent improvement takes 90 percent of time

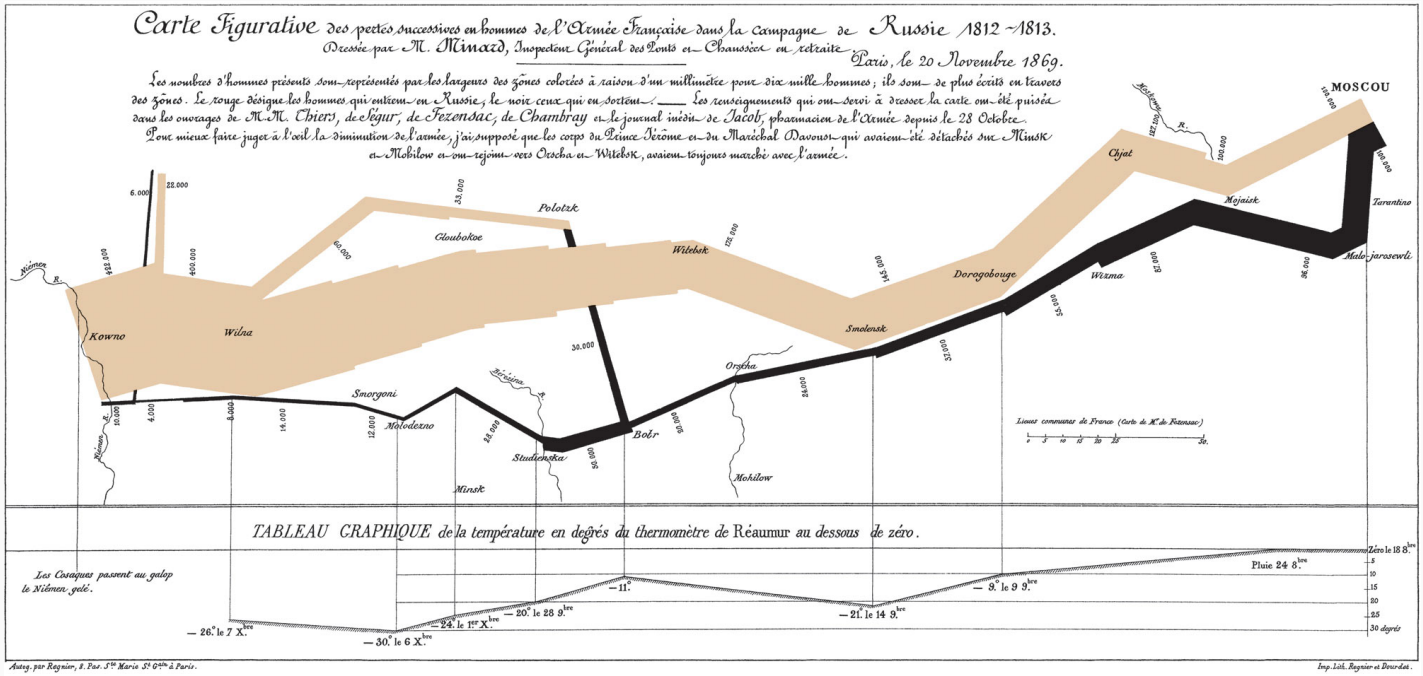
```
month_length ← c(31, 28, 31, 30, 31, 30,
                 31, 31, 30, 31, 30, 31)

month_breaks ← cumsum(month_length) - 30

basic.plot ←   basic.plot + scale_x_continuous(minor_breaks = month_breaks,
                    breaks = month_breaks[c(1, 4, 7, 10)],
                    labels = c("Jan.", "April", "July", "Oct.")) +
  theme(
    plot.background = element_rect(color = NA, fill = "white"),
    panel.grid.major.x = element_line(color = "grey70", size = 0.2, linetype = "dottec
    panel.grid.minor.x = element_line(color = "grey70", size = 0.2, linetype = "dottec
    axis.text.x = element_text(color = base_grey, size = 5, hjust = 0.5),
  )
```

# Advanced level now



# Minard's 1812 Plot

- Hats off to Andrew Heiss
- One lesson from this course: learn from best coders out there, rewrite and replicate their code



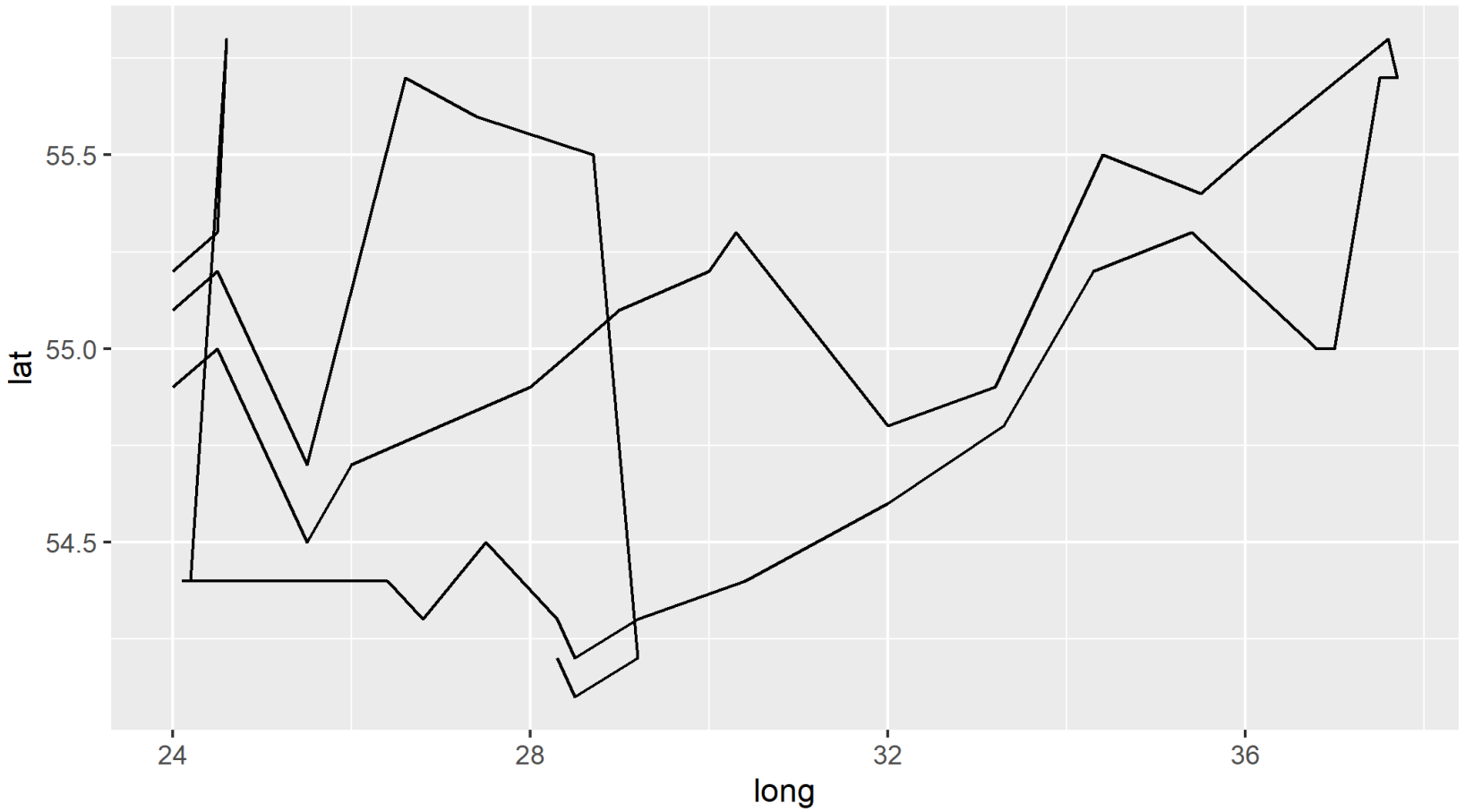

Forward and Retreat path of Napoleon's Army

# Minard's 1812 Plot: Basic Work

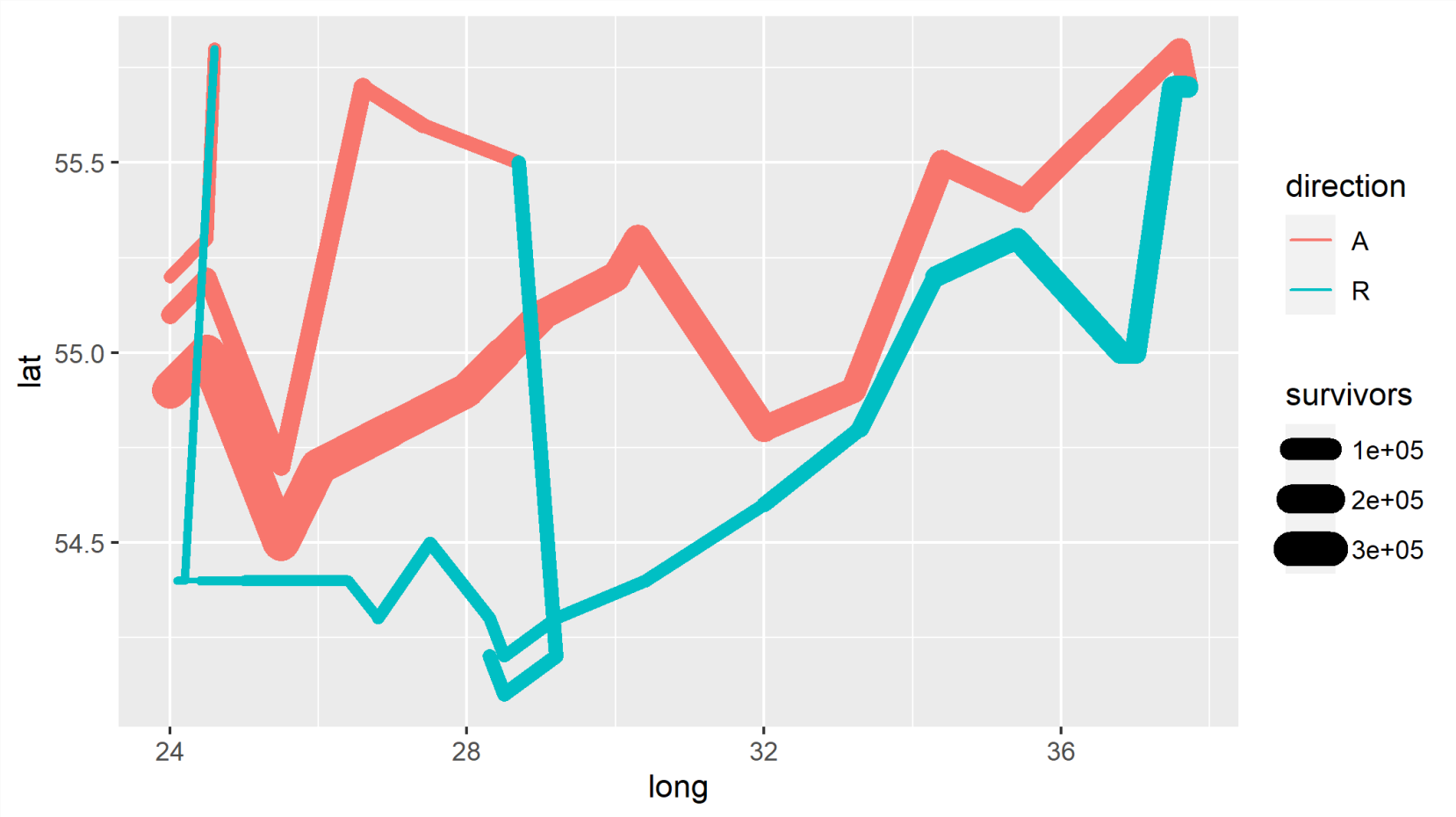
```
pacman::p_load(tidyverse,
               lubridate,
               ggmap,
               ggrepel,
               gridExtra,
               pander)

cities <- read.table("data/cities.txt",
                     header = TRUE, stringsAsFactors = FALSE)

troops <- read.table("data/troops.txt",
                     header = TRUE, stringsAsFactors = FALSE)

temps <- read.table("data/temps.txt",
                    header = TRUE, stringsAsFactors = FALSE) %>%
  mutate(date = dmy(date))  # Convert string to actual date
```

# Minard's 1812 Plot: Basic Figure

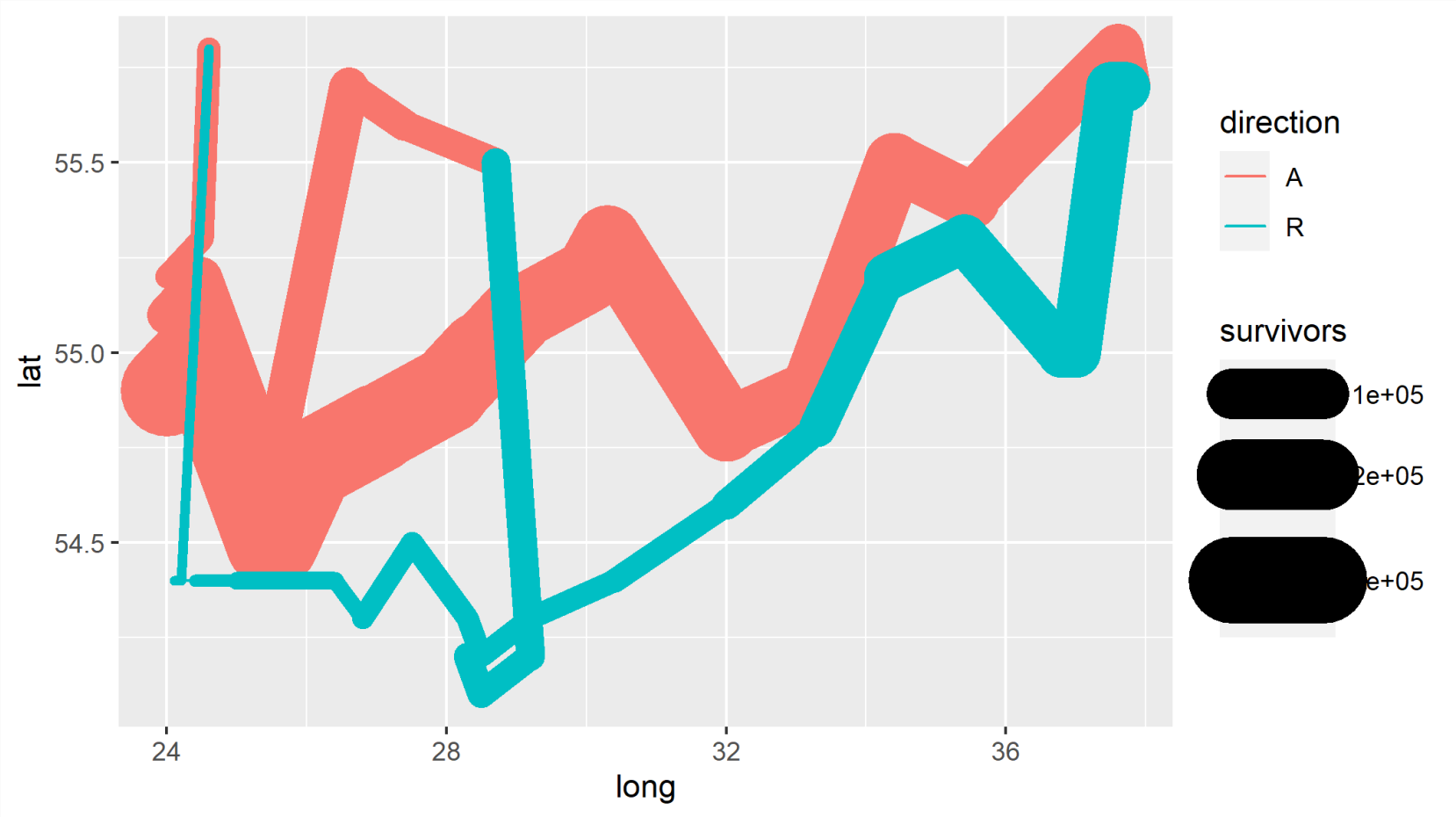
```
ggplot(troops, aes(x = long, y = lat, group = group)) +
  geom_path()
```



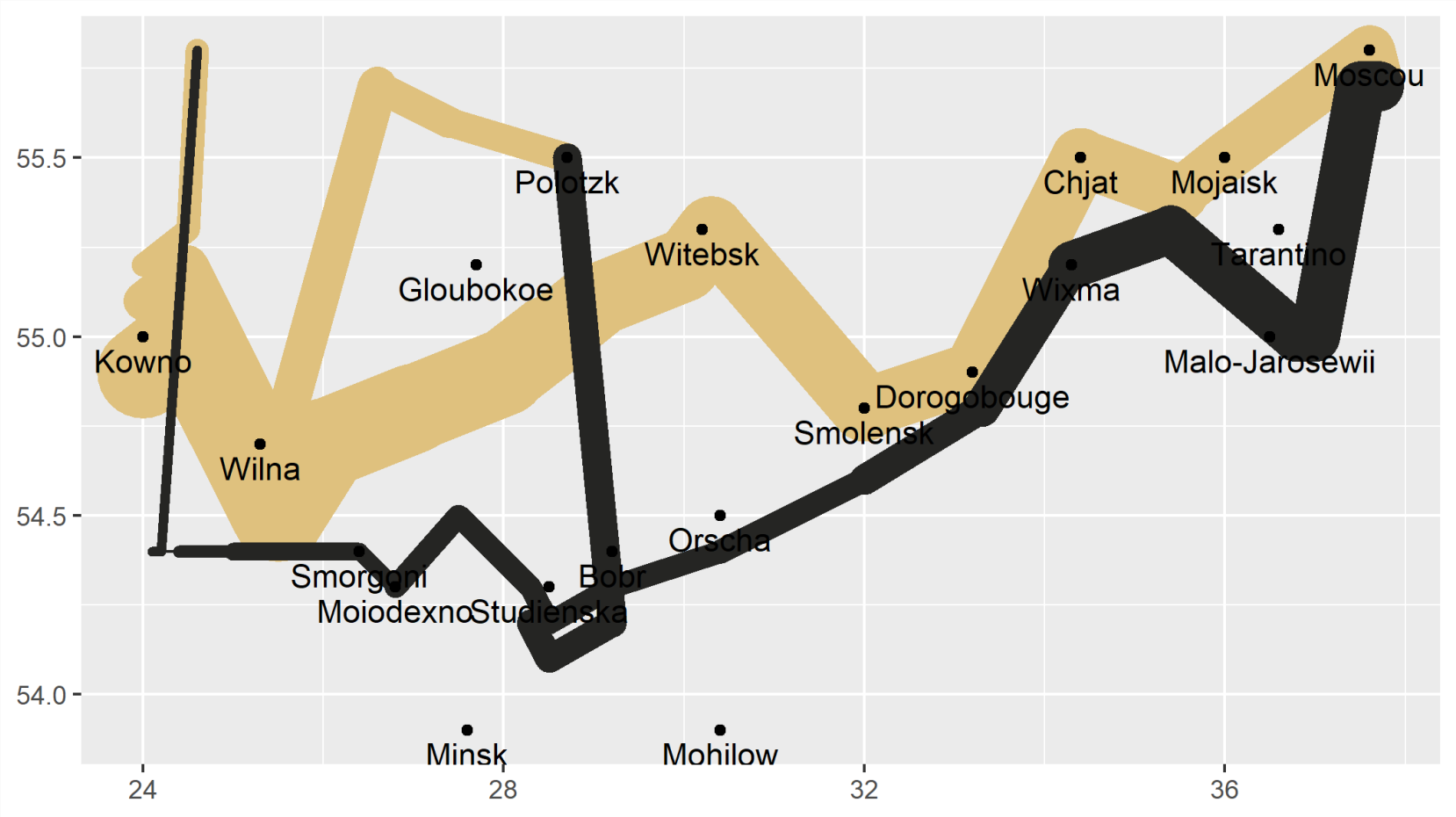
# Minard's 1812 Plot: Basic Figure



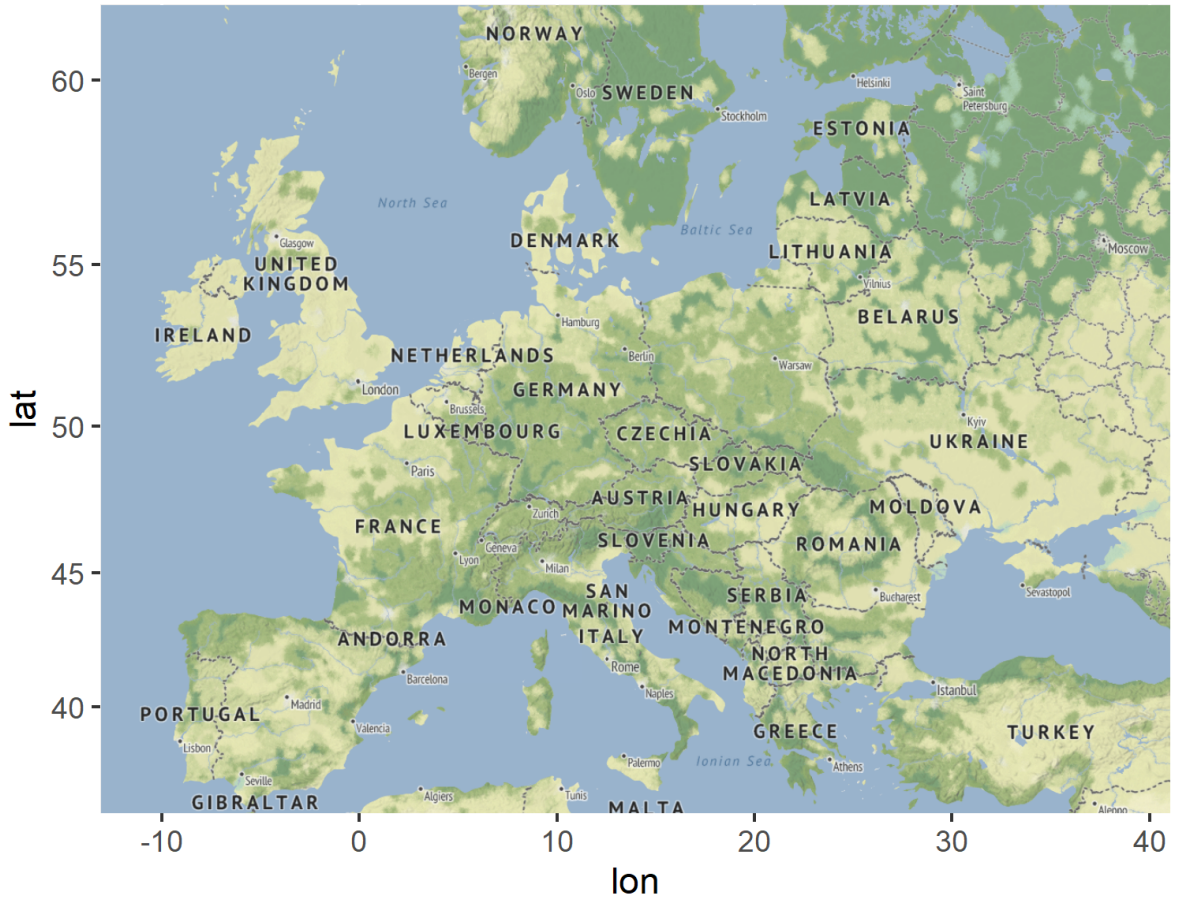
# Minard's 1812 Plot: Bit more



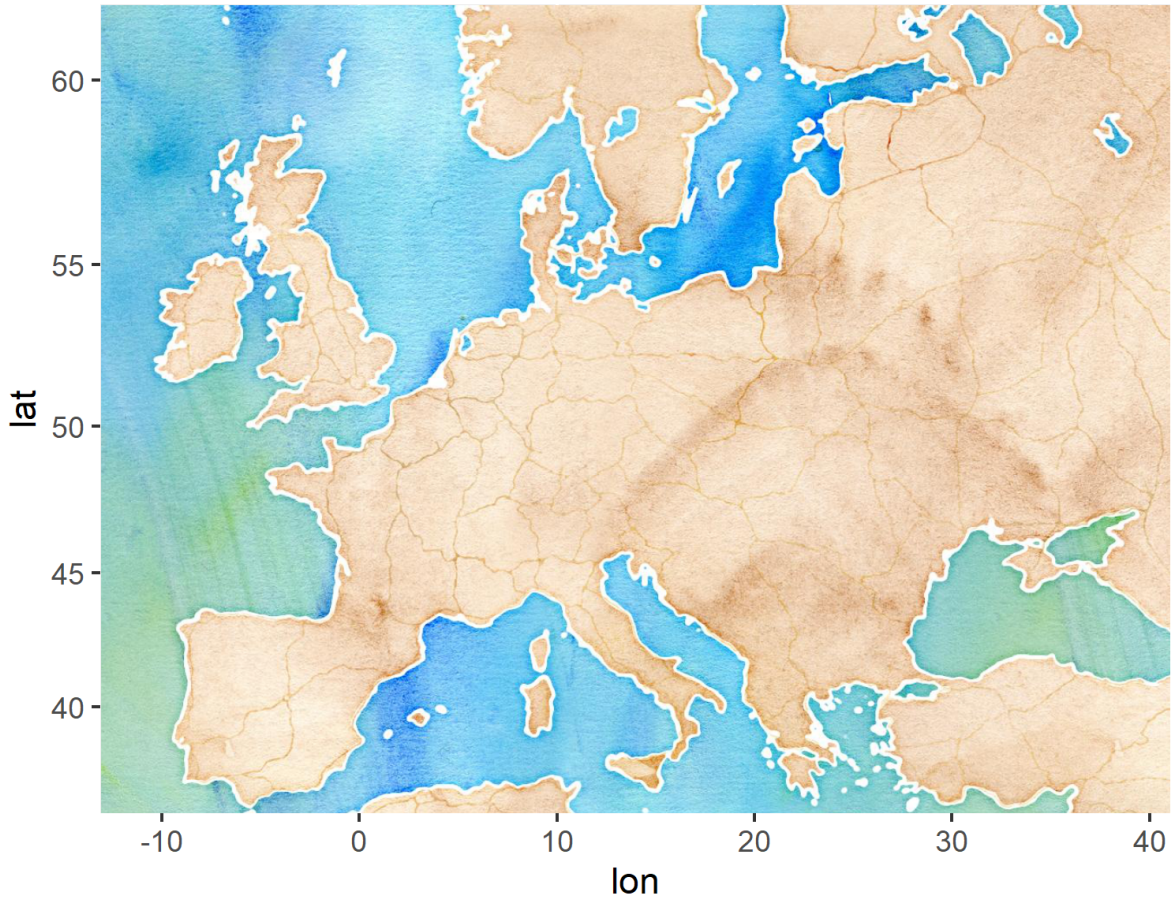
# Minard's 1812 Plot: Bit more



# Minard's 1812 Plot: Bring Spatial Map

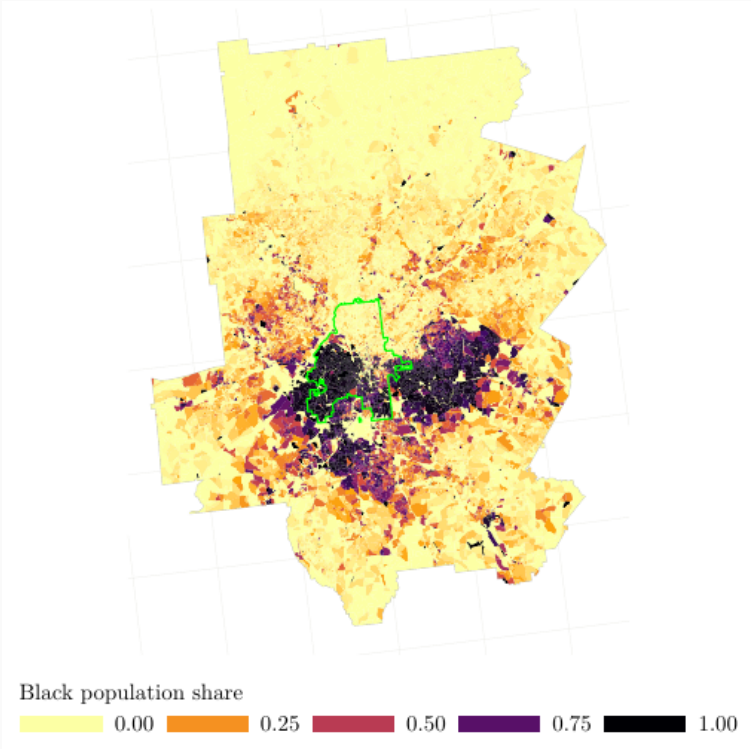


# Minard's 1812 Plot: Watercolor



# Minard's 1812 Plot: Overlay

# Spatial Map: My Own Plot




Atlanta Black population share at the block level

class: inverse, center, middle name: introduction

# What is R?

- R is a language and environment for statistical computing and graphics.
- **Open-source** and extensible.
- Built upon *S*, a statistical programming language.
- Rich set of packages for data manipulation (e.g., `tidyverse` environment build upon `tidyr`, `dplyr`, etc.).
- Comprehensive plotting libraries (e.g., `ggplot2`, `lattice`).
- Wide array of statistical tests.
- Facilitates machine learning algorithms.
- Excellent packages (e.g., `sf`, `terra`, etc.) for spatial datasets and GIS applications

# General Guidelines

- Use a different folder for each new project / assignment.
- Set the Working Directory for R at the beginning of each project
- Always use script files to keep a record of your work
- Initialize a Git repository in each project folder (unless the project is very small).

## My own example folder

- `root/`
  - *master.R*
  - `auxfiles/`
    - *auxfiles/script1.R*
    - *auxfiles/script2.R*
  - `Data/` (stores raw data, use *.gitignore* to ignore the content of this folder to *Git*)
    - *Data/rawdata1.dta*
    - ...
  - `Output/` (stores compiled, cleaned data, use *.gitignore*)
    - *Output/tables/* (stores estimation, descriptive tables)
    - *Output/Figures/* (stores figures, can use *.gitignore*)

# How `master.R` looks like?

- This file sets up the environment where your code will run

```
#clean up the memory
rm(list=ls())
gc()


#set up the language of your error message
Sys.setenv(lang='en')

#most important: set up your working directory
#all paths are relative to this
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

# How `master.R` looks like?

```r
#install pacman package, if not available always include this line, as your co-author
#might not have pacman
if (!require(pacman)) {
  install.packages('pacman', dependencies = T)
}

#load other packages you might need
pacman::p_load(tidyverse,
               hrbethemes,
               fontawesome,
               tidyverse,
               ggtext,
               lubridate,
               here,
               magick)
```

# What is a package?

- An R package is a collection of R functions, data, and documentation that can be installed and used by other R users.
    - Packages are a way to organize and distribute code, data, and documentation
- General working with packages:
    - `install.packages('tidyverse')`
        - This command already finds a mirror to download the package
    - `install.packages('tidyverse', repos='http://cran.us.r-project.org/', dependencies=TRUE)`
        - You are specifying a specific mirror
- I will suggest to use `pacman` (see the previous slide)
- Sometimes two different packages have a conflict:
    - they contain functions with same name
    - `R` will resolve it by selecting one function to *mask* another
    - If not careful, you can make error
- If loading two packages which have a conflict, try to use `::`

# Variables in *R*

- R uses symbolic variables: words, letters, and their combination
- These variables either *represent* or *store* values/objects
- **How it works?** Use of *assignment operator* ( `←` )

```
n ← 5
n
```

[1] 5

```
numVar ← 10
numVar
```

[1] 10

# Variables in *R*

- R uses symbolic variables: words, letters, and their combination
- These variables either *represent* or *store* values/objects
- **How it works?** Use of *assignment operator* ( `←` )
  - Don't use `=` for assignment
  - Most of the time it will work, but in `R` it is a bad practice

```
stringVar ← 'Hello World, I am R'
stringVar
```

[1] "Hello World, I am R"

```
boolVar ← TRUE
boolVar
```

[1] TRUE

```
newVar ←  boolVar + 3
newVar
```

[1] 4

# Variable naming etiquette

- We always work in a team
- Your code will be read by someone else
- Two rules for naming variables:
  - Give ***meaningful*** name to each variable
  - Make sure the name is ***readable***
- Read this: Google Style Guide for R Programming

# Functions

- We are not going to use `R` as a calculator
  - You can, but it is an inefficient use of `R`
- You are in this course to **up** your `Data` and `Statistics` work
- `R` allows you to do complicated things in an easy way
- We need `functions` for that
- *What is a function?*
  - A set of routine, you write once, use indefinitely
  - You can share this routine with other people also
    - Think of `package`: just a bunch of functions, packaged together, which can be used by anyone in the world

# Functions

- Generic representation of functions

```
function_name ← function(input1 , input2) {

  # Use `input1` and `input2` along with other functions and variable to
  # do your work


}  #end of function_name
```

# Functions

- **Example:**
  - `sum`, inbuilt function of `R`
  - takes numbers and returns their total

```
sum ← function(number1, number2) {

  total ← number1 + number2
  return(total)

} #end function sum
```

- *Note:* There is no default values for `number1` and `number2`

```
sumDefault ← function(number1=0, number2=1) {

  total ← number1 + number2
  return(total)

} #end function sumDefault
```

# Functions

- How about writing a function where we can pass a vector?

```
sumVector ← function(vector_to_sum) {

  n ← length(vector_to_sum)

  total ← 0

  for(iter in 1:n){
    total ← total + vector_to_sum[iter]
  }

  return(total)


} #end function sumVector
```

# R Data Structures

# Data *types* and *structures*

- `R` has six data types
  - character
  - numeric
  - integer
  - logical
  - complex
  - *raw* (rarely used)
- Combination of data types $\longrightarrow$ data structures
- Data structures can be
  - atomic vectors: the vector hold data only of a single type
  - non-atomic vectors i.e. `lists`: holds data of a multiple type
    - `lists` are one of the most powerful feature of `R`

# Data *types* and *structures*

- Functions provided by `R` to examine what does a `vector` contains
  - `class()`
  - `typeof()`
  - `length()`
  - `attributes()`
- **Example**

```
x ← 'swapnil singh'

class(x)
```

[1] "character"

```
typeof(x)
```

[1] "character"

# Data *types* and *structures*

- Functions provided by `R` to examine what does a `vector` contains
  - `class()`
  - `typeof()`
  - `length()`
  - `attributes()`
- **Example**

```
num ← c(1,2,3,4,5,6)

class(num)
```

[1] "numeric"

```
typeof(num)
```

[1] "double"

# Data *types* and *structures*

- Functions provided by `R` to examine what does a `vector` contains
  - `class()`
  - `typeof()`
  - `length()`
  - `attributes()`
- **Example**

```
num ← c(1,2,3,4,5,6)

length(num)
```

[1] 6

```
attributes(num)
```

NULL

# Vectors

- Two types of vectors:
  - ***atomic vector:*** contain element of same type
  - ***lists:*** the most important thing in *R*

# Atomic vector

```
#default initialization of vector
vector()
```

```
## logical(0)
```

```
#vector of type character
vector('character', length = 3)
```

```
## [1] "" "" ""
```

```
#use the character constructor, instead of vector
character(3)
```

```
## [1] "" "" ""
```

```
numeric(7)
```

```
## [1] 0 0 0 0 0 0 0
```

```
logical(5)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

# Atomic Vector

- Another way to create vector: use function `c`

```
numericVec ← c(0,1,3,4)
numericVec
```

```
## [1] 0 1 3 4
```

```
logicalVec ← c(TRUE,FALSE,FALSE, TRUE)
logicalVec
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

```
characterVec ← c('Swapnil', "R Course", 'LB')
characterVec
```

```
## [1] "Swapnil"  "R Course" "LB"
```

```
# Add element to vector
characterVec ← c(characterVec, 'October')
characterVec
```

```
## [1] "Swapnil"  "R Course" "LB"       "October"
```

# Atomic Vector - Missing Data

- A vector can contain missing data, reported by `NA`
- Does not matter the type of vector

```
presidentName ← c('Biden', 'Obama', NA, 'Bush')
presidentName
```

```
## [1] "Biden" "Obama" NA      "Bush"
```

```
#find the missing element
pnameNA ← is.na(presidentName)
pnameNA
```

```
## [1] FALSE FALSE  TRUE FALSE
```

```
whichNameNA ← which(is.na(presidentName))
whichNameNA
```

```
## [1] 3
```

# Lists

```
#unnamed list
unnamedList ← list('my name is Swapnil Singh',
                   35,
                   'economist')

unnamedList
```

```
## [[1]]
## [1] "my name is Swapnil Singh"
##
## [[2]]
## [1] 35
##
## [[3]]
## [1] "economist"
```

- Access elements of list

```
unnamedList[[1]][1]
```

```
## [1] "my name is Swapnil Singh"
```

# Lists

```
#unnamed list
unnamedList ← list(c('my name is Swapnil Singh', 'I work at LB'),
                   35,
                   'economist')

unnamedList
```

```
## [[1]]
## [1] "my name is Swapnil Singh" "I work at LB"
##
## [[2]]
## [1] 35
##
## [[3]]
## [1] "economist"
```

# Lists

```r
#named list
namedList ← list(intro = 'my name is Swapnil Singh',
                 age = 35,
                 profession = 'economist')
namedList
```

```
## $intro
## [1] "my name is Swapnil Singh"
##
## $age
## [1] 35
##
## $profession
## [1] "economist"
```

```r
namedList$intro
```

```
## [1] "my name is Swapnil Singh"
```

```r
namedList[['age']]
```

```
## [1] 35
```

# Data Frame

- Core of `data science` in `R`
- It is actually a `list` where each element of list has same length
- Let's take the previous example

```
#unnamed list
unnamedList ← list(c('my name is Swapnil Singh', 'I work at LB'),
                   35,
                   'economist')
#first element has different length than other two
length(unnamedList[[1]])
```

```
## [1] 2
```

```
length(unnamedList[[2]])
```

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

```
length(unnamedList[[3]])
```

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

# Data Frame

- Work with dataframe `CO2`

```
#ugly output
CO2
```

```
##    Plant        Type  Treatment conc uptake
## 1    Qn1      Quebec nonchilled   95   16.0
## 2    Qn1      Quebec nonchilled  175   30.4
## 3    Qn1      Quebec nonchilled  250   34.8
## 4    Qn1      Quebec nonchilled  350   37.2
## 5    Qn1      Quebec nonchilled  500   35.3
## 6    Qn1      Quebec nonchilled  675   39.2
## 7    Qn1      Quebec nonchilled 1000   39.7
## 8    Qn2      Quebec nonchilled   95   13.6
## 9    Qn2      Quebec nonchilled  175   27.3
## 10   Qn2      Quebec nonchilled  250   37.1
## 11   Qn2      Quebec nonchilled  350   41.8
## 12   Qn2      Quebec nonchilled  500   40.6
## 13   Qn2      Quebec nonchilled  675   41.4
## 14   Qn2      Quebec nonchilled 1000   44.3
## 15   Qn3      Quebec nonchilled   95   16.2
## 16   Qn3      Quebec nonchilled  175   32.4
## 17   Qn3      Quebec nonchilled  250   40.3
## 18   Qn3      Quebec nonchilled  350   42.1
```

# Data Frame

- Work with dataframe `CO2`

```
#better output: show first six rows or last six rows
head(CO2)
```

```
##   Plant   Type  Treatment conc uptake
## 1   Qn1 Quebec nonchilled   95   16.0
## 2   Qn1 Quebec nonchilled  175   30.4
## 3   Qn1 Quebec nonchilled  250   34.8
## 4   Qn1 Quebec nonchilled  350   37.2
## 5   Qn1 Quebec nonchilled  500   35.3
## 6   Qn1 Quebec nonchilled  675   39.2
```

```
tail(CO2)
```

```
##    Plant        Type Treatment conc uptake
## 79   Mc3 Mississippi   chilled  175   18.0
## 80   Mc3 Mississippi   chilled  250   17.9
## 81   Mc3 Mississippi   chilled  350   17.9
## 82   Mc3 Mississippi   chilled  500   17.9
## 83   Mc3 Mississippi   chilled  675   18.9
## 84   Mc3 Mississippi   chilled 1000   19.9
```

# Data Frame

```
#better output: dimensions of the data frame
#number of rows and columns
dim(CO2)
```

```
## [1] 84  5
```

```
nrow(CO2)
```

```
## [1] 84
```

```
ncol(CO2)
```

```
## [1] 5
```

```
names(CO2)
```

```
## [1] "Plant"     "Type"      "Treatment" "conc"      "uptake"
```

```
colnames(CO2)
```

```
## [1] "Plant"     "Type"      "Treatment" "conc"      "uptake"
```

# Class Exercise: Data Frame

- Load `LifeCycleSavings` data in your `R` work environment

```
dataLifeCycle ← LifeCycleSavings
dataLifeCycle
```

```
##                 sr pop15 pop75     dpi  ddpi
## Australia    11.43 29.35  2.87 2329.68  2.87
## Austria      12.07 23.32  4.41 1507.99  3.93
## Belgium      13.17 23.80  4.43 2108.47  3.82
## Bolivia       5.75 41.89  1.67  189.13  0.22
## Brazil       12.88 42.19  0.83  728.47  4.56
## Canada        8.79 31.72  2.85 2982.88  2.43
## Chile         0.60 39.74  1.34  662.86  2.67
## China        11.90 44.75  0.67  289.52  6.51
## Colombia      4.98 46.64  1.06  276.65  3.08
## Costa Rica   10.78 47.64  1.14  471.24  2.80
## Denmark      16.85 24.42  3.93 2496.53  3.99
## Ecuador       3.59 46.31  1.19  287.77  2.19
## Finland      11.24 27.84  2.37 1681.25  4.32
## France       12.64 25.06  4.70 2213.82  4.52
## Germany      12.55 23.31  3.35 2457.12  3.44
## Greece       10.67 25.62  3.10  870.85  6.28
## Guatamala     3.01 46.05  0.87  289.71  1.48
## Honduras      7.70 47.32  0.58  232.44  3.19
```

# Class Exercise: Data Frame

- Load `LifeCycleSavings` data in your `R` work environment
- Extract the column names and put it in a variable `cnames`

```
dataLifeCycle ← LifeCycleSavings
cnames ← colnames(dataLifeCycle)
cnames
```

```
## [1] "sr"    "pop15" "pop75" "dpi"   "ddpi"
```

# Class Exercise: Data Frame

- Load `LifeCycleSavings` data in your `R` work environment
- Extract the column names and put it in a variable `cnames`
- Show the first eight rows of the loaded data

```
dataLifeCycle ← LifeCycleSavings
head(dataLifeCycle, n = 8)
```

```
##             sr pop15 pop75     dpi ddpi
## Australia 11.43 29.35  2.87 2329.68 2.87
## Austria   12.07 23.32  4.41 1507.99 3.93
## Belgium   13.17 23.80  4.43 2108.47 3.82
## Bolivia    5.75 41.89  1.67  189.13 0.22
## Brazil    12.88 42.19  0.83  728.47 4.56
## Canada     8.79 31.72  2.85 2982.88 2.43
## Chile      0.60 39.74  1.34  662.86 2.67
## China     11.90 44.75  0.67  289.52 6.51
```

# Class Exercise: Data Frame

- Load `LifeCycleSavings` data in your `R` work environment
- Extract the column names and put it in a variable `cnames`
- Show the first eight rows of the loaded data
- Show the last three rows of the loaded data

```
dataLifeCycle ← LifeCycleSavings
tail(dataLifeCycle, n = 3)
```

```
##             sr pop15 pop75    dpi  ddpi
## Uruguay  9.24 28.13  2.72 766.54  1.88
## Libya    8.89 43.69  2.07 123.58 16.71
## Malaysia 4.71 47.20  0.66 242.69  5.08
```

# Class Exercise: Data Frame

- Load `LifeCycleSavings` data in your `R` work environment
- Extract the column names and put it in a variable `cnames`
- Show the first eight rows of the loaded data
- Show the last three rows of the loaded data
- Construct a new column `pop15_75` which is sum of `pop15` and `pop75`

```
dataLifeCycle ← LifeCycleSavings
dataLifeCycle$pop15_75 ← dataLifeCycle$pop15 + dataLifeCycle$pop75
head(dataLifeCycle)
```

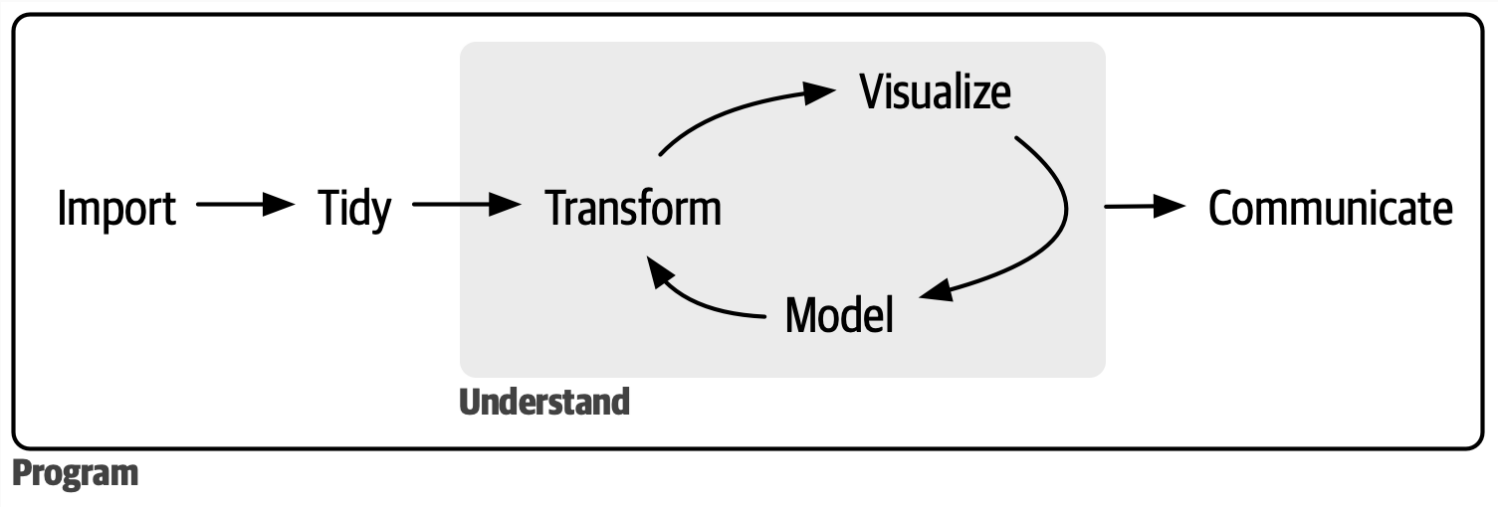
```
##              sr pop15 pop75     dpi ddpi pop15_75
## Australia 11.43 29.35  2.87 2329.68 2.87    32.22
## Austria   12.07 23.32  4.41 1507.99 3.93    27.73
## Belgium   13.17 23.80  4.43 2108.47 3.82    28.23
## Bolivia    5.75 41.89  1.67  189.13 0.22    43.56
## Brazil    12.88 42.19  0.83  728.47 4.56    43.02
## Canada     8.79 31.72  2.85 2982.88 2.43    34.57
```

# tidyverse + ggplot

# Introduction

- For reference: `R for Data Science (2e)`
- I will
  - Provide a quick tour of `tidy` environment
  - Introduce `pipes`
  - Spend a lot of time on data visualization
  - Spend sometime introducing `RMarkdown`
- I will **not** introduce statistical analysis and methods
  - No time

# Workflow




Source: R4DS 2e

- We will cover `Import`, `Tidy`, `Transform`, `Visualize`, and `Communicate`
- `Model` part will not be covered

# Start of the workflow

- You got the data
- First thing you do: try to `feel` it
  - Do some basic plots
  - Understand the type of variables it has
  - ...
- To plot, we will use `ggplot` package
  - loads with `tidyverse` package automatically
- `ggplot`: (G)rammar of (G)raphics plot
  - can only be used with `dataframes`

```
pacman::p_load(tidyverse,
               palmerpenguins,
               ggthemes)
```

- We use `palmerpenguins` data

# palmerpenguins data

- Description here: `palmer penguins data`
- Contains two datasets: `penguins` and `penguins_raw`

```
head(penguins)
```

```
## # A tibble: 6 × 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
## 1 Adelie  Torgersen           39.1          18.7               181        3750
## 2 Adelie  Torgersen           39.5          17.4               186        3800
## 3 Adelie  Torgersen           40.3          18                 195        3250
## 4 Adelie  Torgersen           NA            NA                  NA          NA
## 5 Adelie  Torgersen           36.7          19.3               193        3450
## 6 Adelie  Torgersen           39.3          20.6               190        3650
## # ℹ 2 more variables: sex <fct>, year <int>
```

```
dim(penguins)
```

```
## [1] 344   8
```

# palmerpenguins data

- Description here: `palmer penguins data`
- Contains two datasets: `penguins` and `penguins_raw`

```
glimpse(penguins)
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel…
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse…
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, …
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, …
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186…
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, …
## $ sex               <fct> male, female, female, NA, female, male, female, male…
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007…
```

# palmerpenguins data - plotting

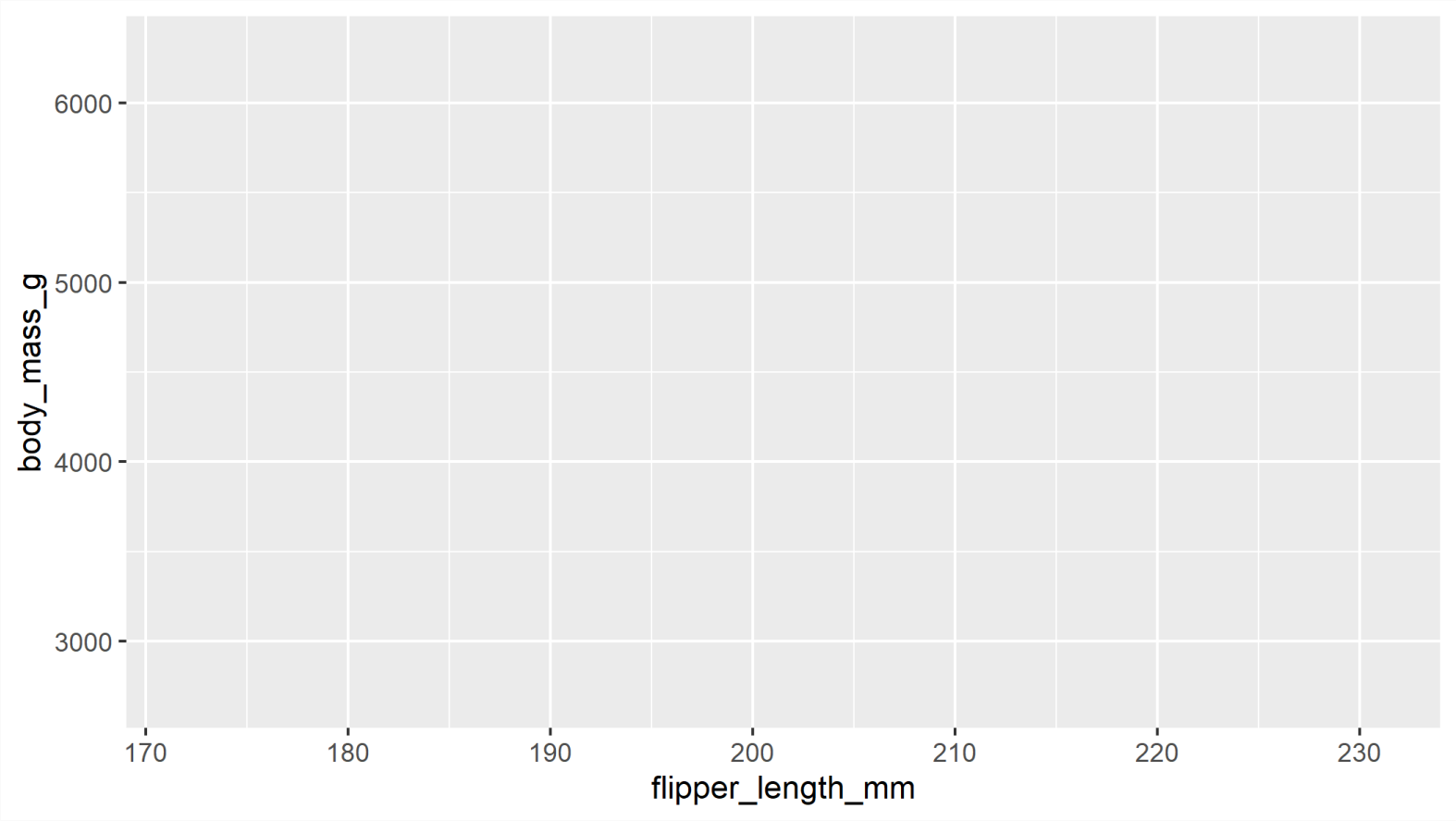
- What is the relationship between flipper length (`flipper_length_mm`) and body mass (`body_mass_g`) of penguins?
- Step by step, follow me by writing your own version
- `ggplot` works in layers

# palmerpenguins data - plotting

```
ggplot(data = penguins)
```

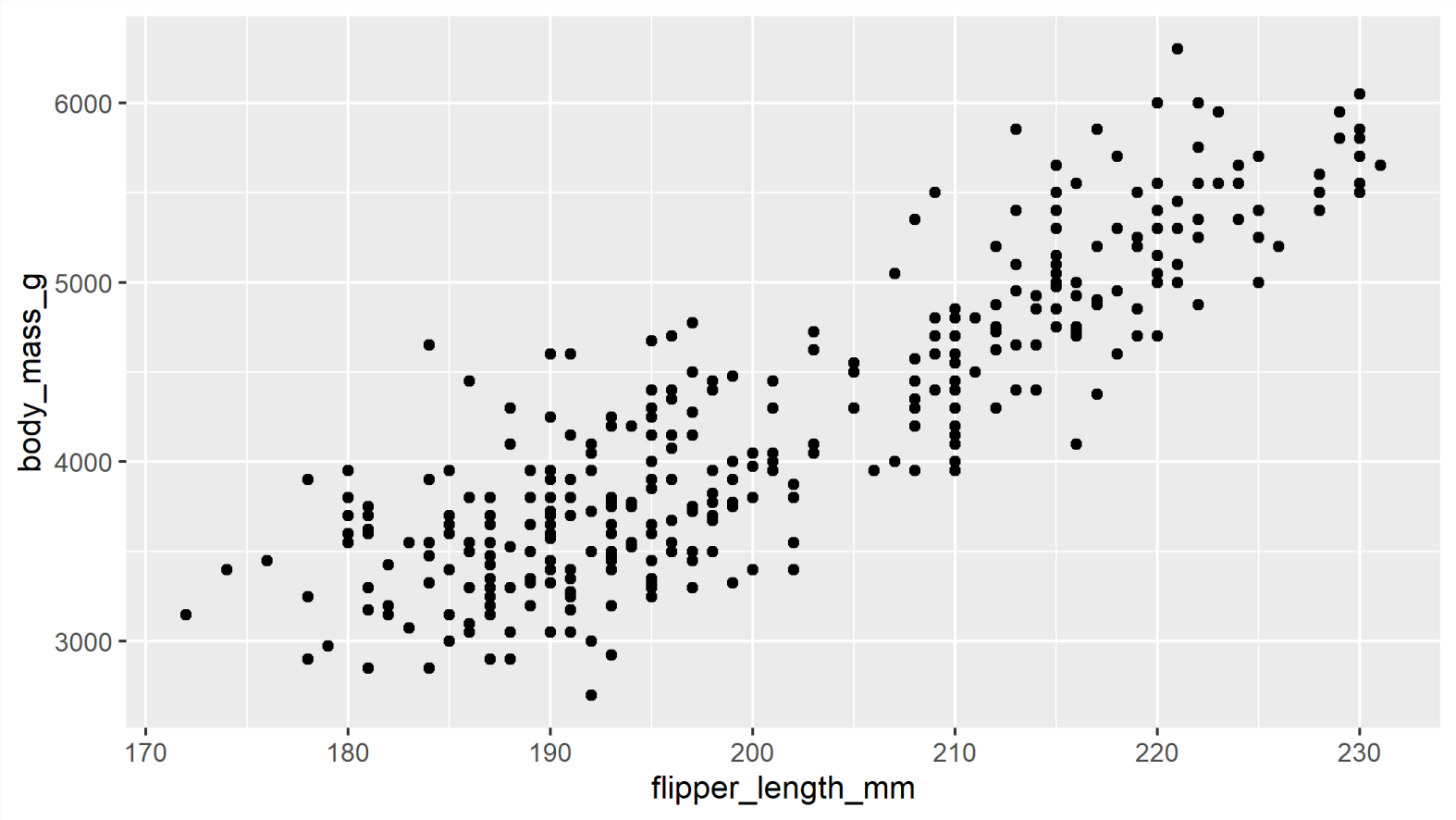
# palmerpenguins data - plotting

```
ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g))
```



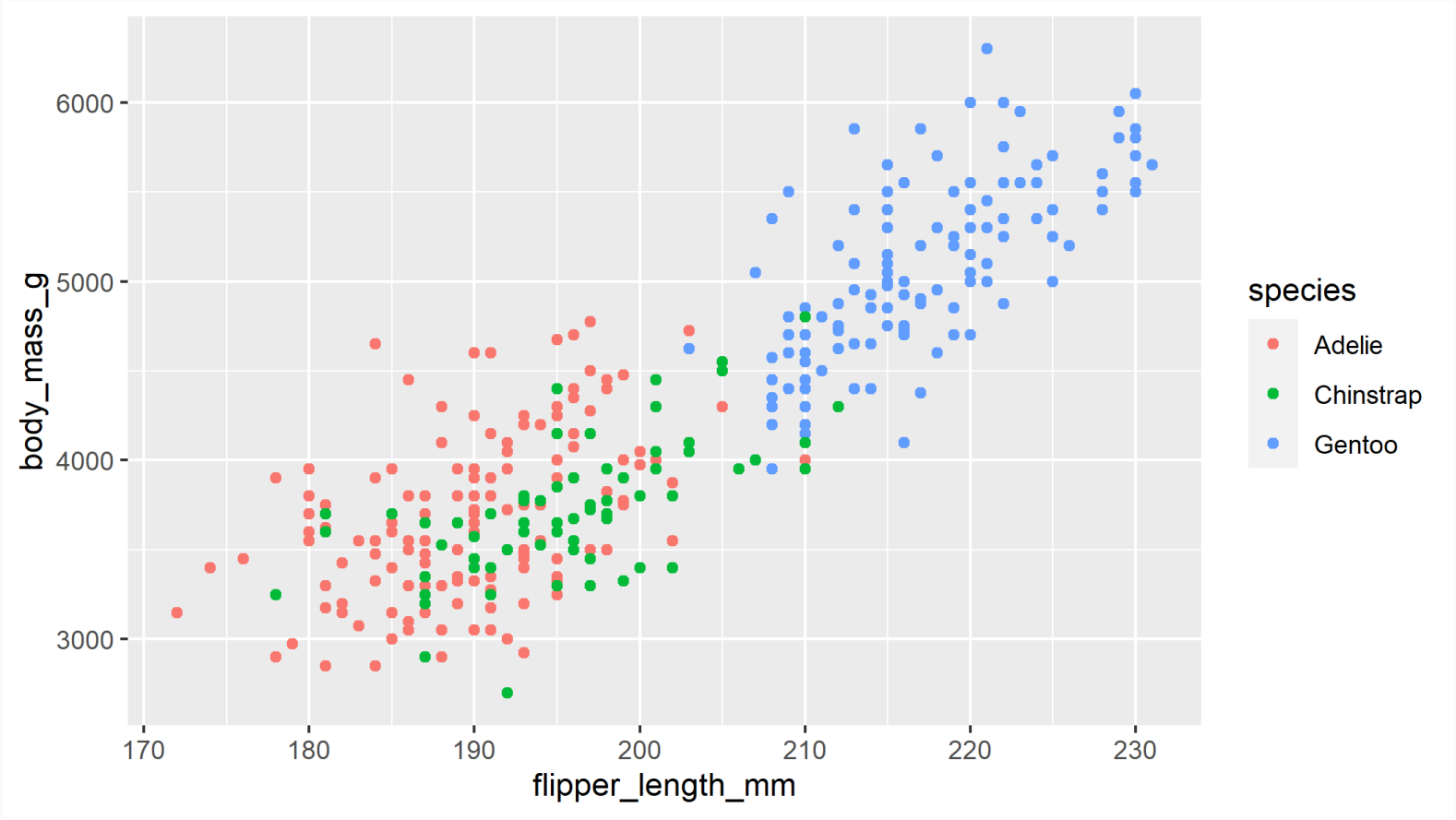
# palmerpenguins data - plotting

```
ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```



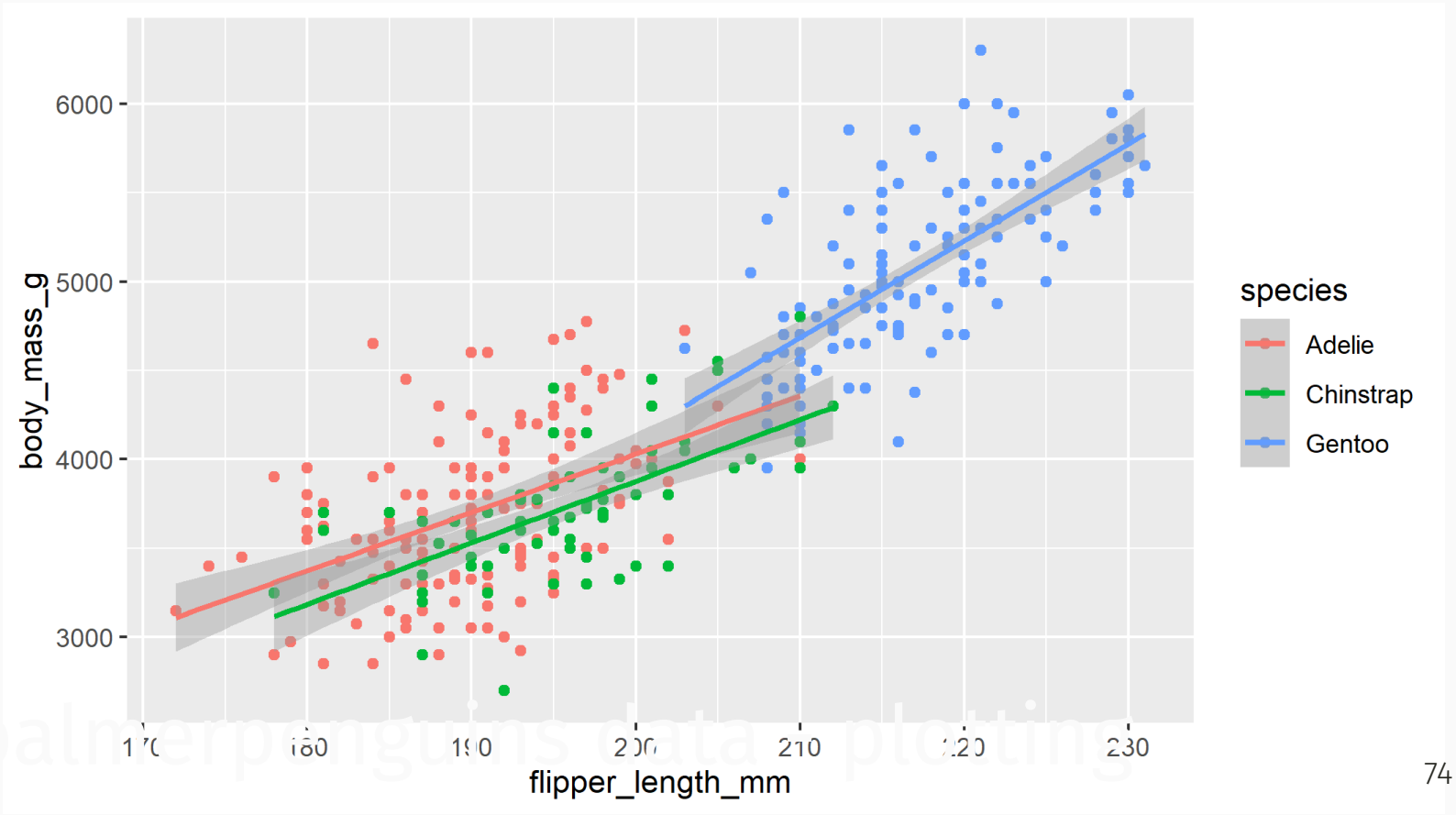
# palmerpenguins data - plotting

```
ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g, color=species)) +
  geom_point()
```



# palmerpenguins data - plotting

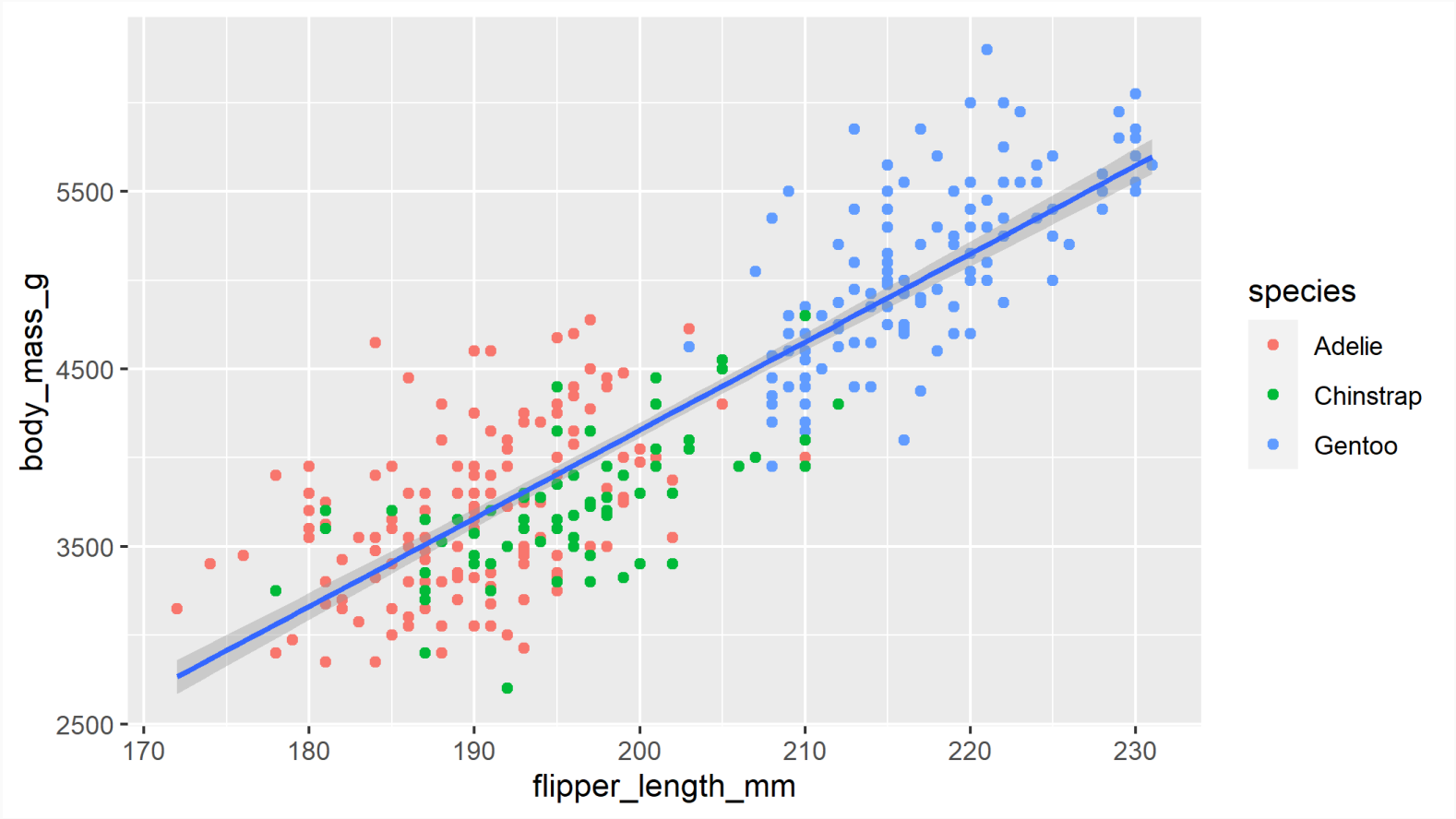
```
ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g, color=species)) +
  geom_point() +
  #add linear fit (three separate lines for each group)
  geom_smooth(method='lm')
```



# palmerpenguins data - plotting

```
p ←  ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +

  # notice how `aes` for geom_point shifted from above to here
  geom_point(mapping = aes(color=species)) +

  #add linear fit (one line across all three groups)
  geom_smooth(method='lm')
```

# palmerpenguins data - plotting



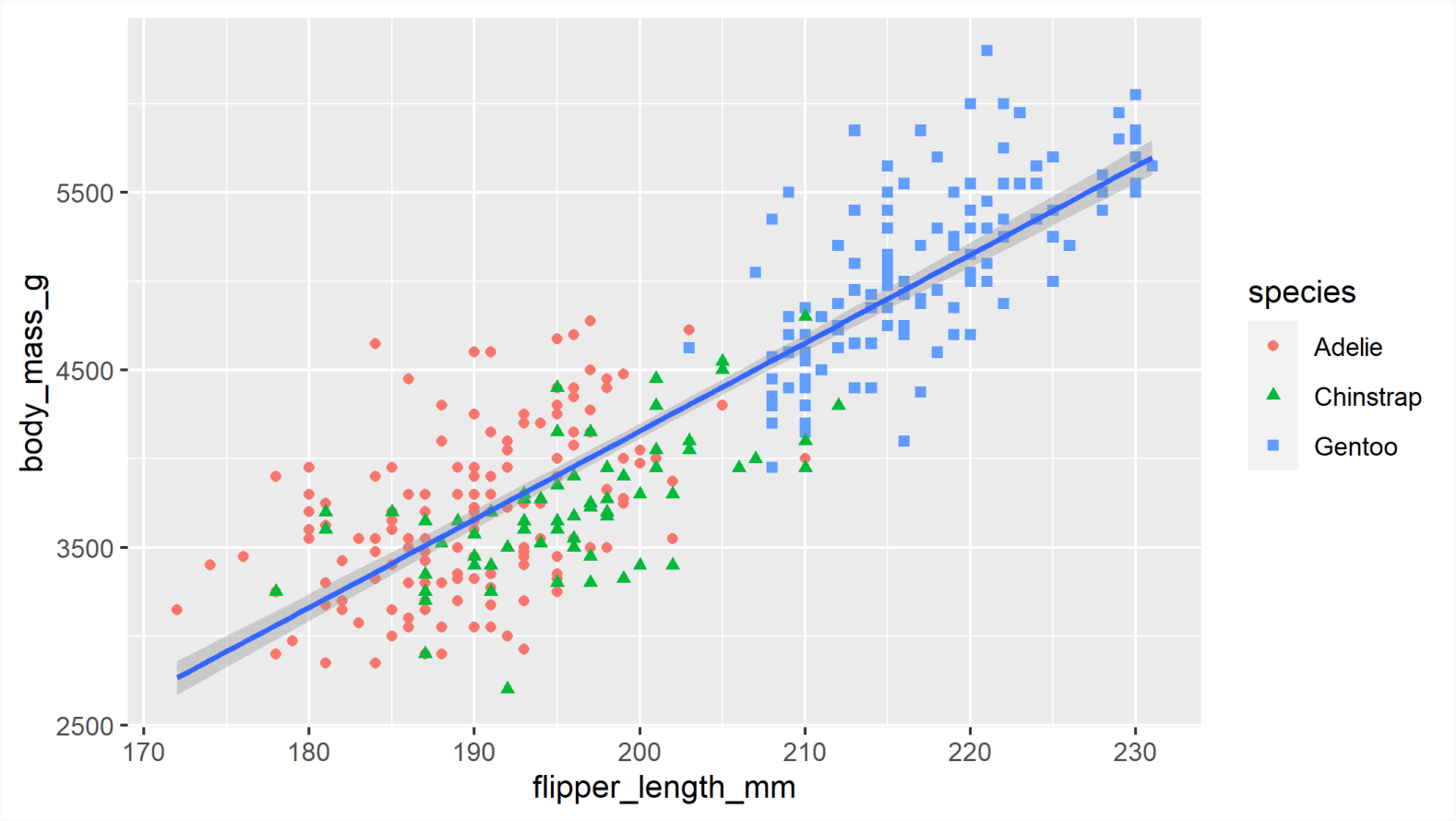
# palmerpenguins data - plotting

- Different shapes for each species

```
p ←  ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +

  # notice how `aes` for geom_point shifted from above to here
  geom_point(mapping = aes(color=species, shape=species)) +

  #add linear fit (one line across all three groups)
  geom_smooth(method='lm')

p
```

# palmerpenguins data - plotting

- Different shapes for each species



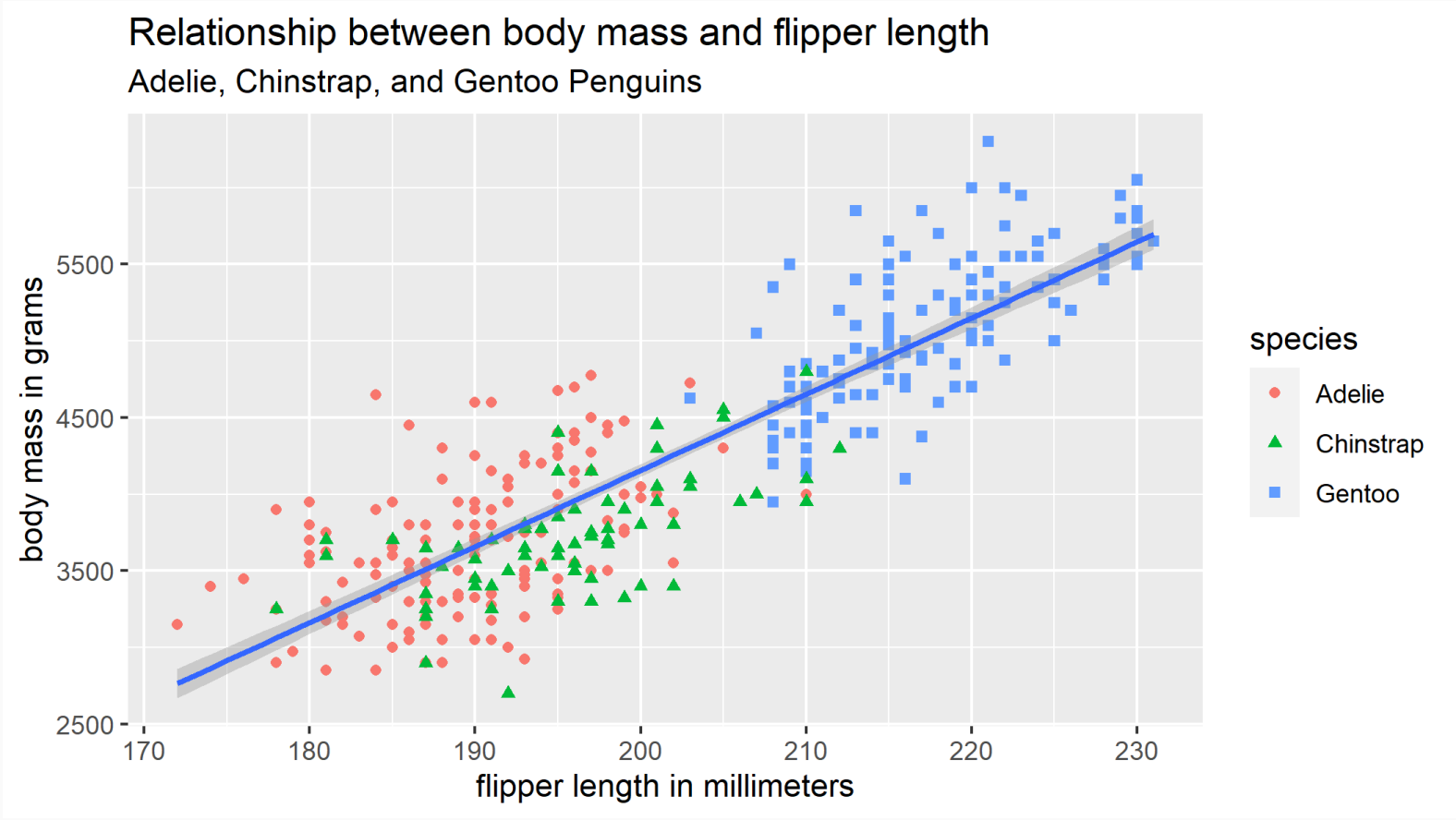
# palmerpenguins data - plotting

- Add a new layer for axis and plot titles

```
p ←  ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +

  # notice how `aes` for geom_point shifted from above to here
  geom_point(mapping = aes(color=species, shape=species)) +

  #add linear fit (one line across all three groups)
  geom_smooth(method='lm')  +

  labs(
    title = "Penguins: relationship between body mass and flipper length",
    subtitle = "Adelie, Chinstrap, and Gentoo Penguins",
    x = "flipper length in millimeters",
    y = "body mass in grams",
    color = "species",
    shape = "species"
  )

p
```

# palmerpenguins data - plotting

- Add a new layer for axis and plot titles



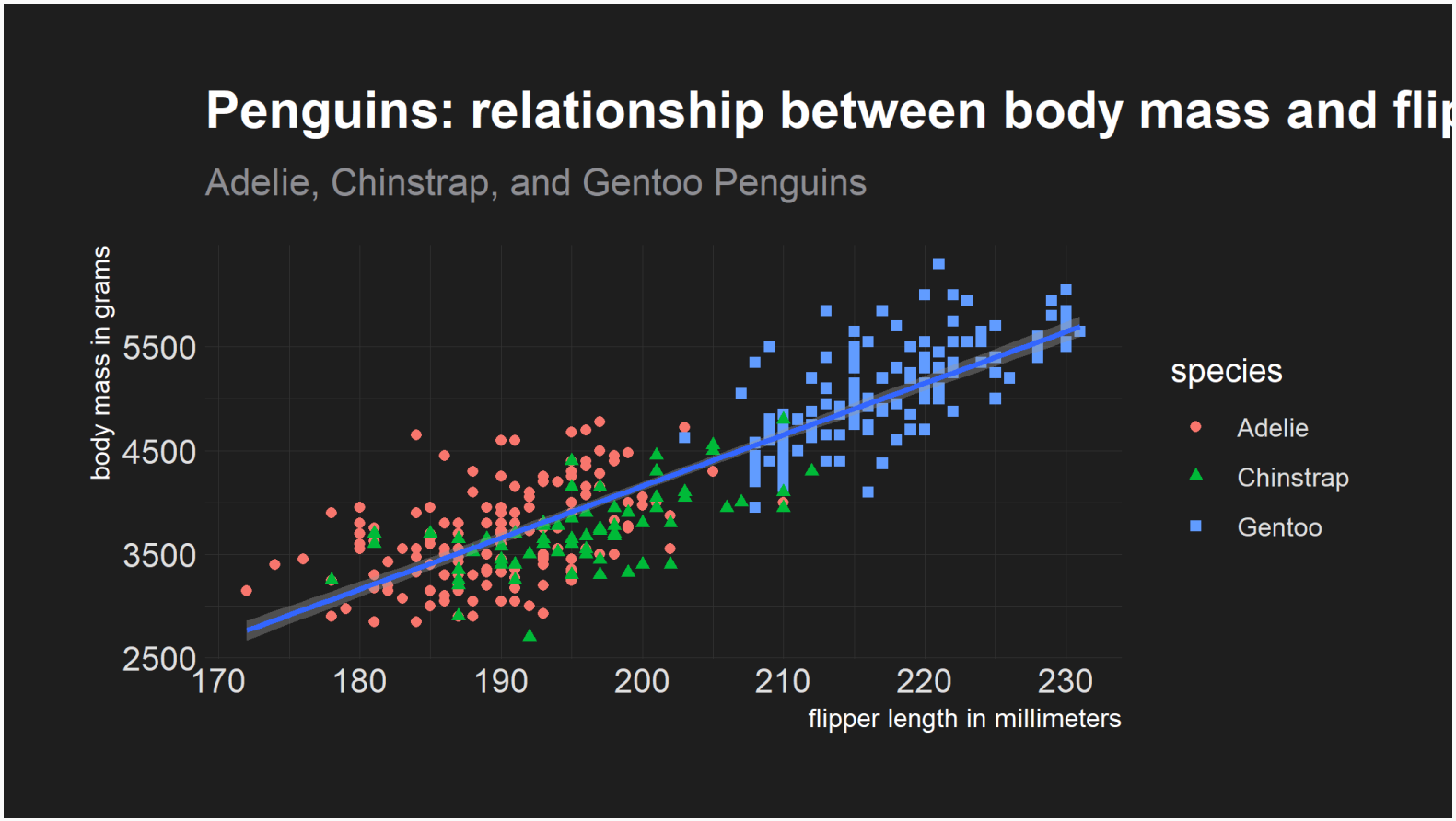
# palmerpenguins data - plotting

- Change the theme of the plot

```
p <-  ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +

  # notice how `aes` for geom_point shifted from above to here
  geom_point(mapping = aes(color=species, shape=species)) +


  #add linear fit (one line across all three groups)
  geom_smooth(method='lm')  +

  #modern theme
  theme_modern_rc() +

  labs(
    title = "Penguins: relationship between body mass and flipper length",
    subtitle = "Adelie, Chinstrap, and Gentoo Penguins",
    x = "flipper length in millimeters",
    y = "body mass in grams",
    color = "species",
    shape = "species"
  )

p
```

# palmerpenguins data - plotting

- Change the theme of the plot



# End of Lecture 1

- We learned
  - basic principles of `R`
  - aesthetic (`aes`) mapping of `ggplot`
  - basic understanding of lists
- Next class
  - digging deeper into `tidyverse` environment
  - learn about `pipes`
  - understanding the concept of `tidy` data
    - `pivot`