# Data Manipulation 2

STA 032: Gateway to data science Lecture 6

Jingwei Xiong

April 14, 2023

## Reminders

- HW 1 is due April 17th at 12pm
- HW 2 will be posted on the course website, due April 26 12pm.
- Please start the homework as soon as possible.
- Discussion will cover homework problems.

## Recap

- Data manipulation tools in `tidyverse`
  - `select()`
  - `arrange()`
  - `slice()`
  - `filter()`

> Remember, before using all tidyverse functions, you need to library(tidyverse) first!

## Today

- Data manipulation tools continue
  - `distinct()`: filter for unique rows
  - `mutate()`: adds new variables
  - `count()`: create frequency tables
  - `summarise()`: perform column summarization operations
  - `group_by()`: for grouped operations
  - `pull()`: access column data as a vector or a number
  - `rename()`: rename an existing column
  - `inner_join()`, `left_join()`: join together a pair of data frames based on a variable present in both data frames that uniquely identify all observations
- Data visualization introduction

## Data: Hotel bookings

- Data from two hotels: one resort and one city hotel
- Observations: Each row represents a hotel booking
- Goal for original data collection: Development of prediction models to classify a hotel booking's likelihood to be cancelled (Antonia et al., 2019)

```r
hotels <- readr::read_csv("data/hotels.csv")
```

Source: TidyTuesday

## First question: What is in the data set?

.tiny[

```
dplyr::glimpse(hotels)
```

```
Rows: 119,390
Columns: 32
$ hotel                          ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "Resort Hotel
$ is_canceled                    ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ lead_time                      ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 342, 737, 7,
$ arrival_date_year              ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 2015, 2015, 2
$ arrival_date_month             ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "July", "July
$ arrival_date_week_number       ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 27, 27, 27, 2
$ arrival_date_day_of_month      ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 1, 1, 1, 1, 1
$ stays_in_weekend_nights        ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ stays_in_week_nights           ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 1, 1, 2
$ adults                         ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 2, 2, 1, 1, 2
$ children                       ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ babies                         ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ meal                           ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "BB", "BB", "
$ country                        ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "PRT", "PRT",
$ market_segment                 ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "Direct", "Di
$ distribution_channel           ⊠[3m⊠[38;5;246m<chr>⊠[39m⊠[23m "Direct", "Di
$ is_repeated_guest              ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ previous_cancellations         ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
$ previous_bookings_not_canceled ⊠[3m⊠[38;5;246m<dbl>⊠[39m⊠[23m 0, 0, 0, 0, 0
```

## `distinct()` to filter for unique rows

```
hotels %>%
  distinct(market_segment)
```

```
# A tibble: 8 × 1
  market_segment
  <chr>
1 Direct
2 Corporate
3 Online TA
4 Offline TA/TO
5 Complementary
6 Groups
7 Undefined
8 Aviation
```

Recall: `arrange()` to order alphabetically

```
hotels %>%
  distinct(market_segment) %>%
  arrange(market_segment)
```

```
# A tibble: 8 × 1
  market_segment
  <chr>
1 Aviation
2 Complementary
3 Corporate
4 Direct
5 Groups
6 Offline TA/TO
7 Online TA
8 Undefined
```

## `distinct()` using more than one variable

```
hotels %>%
  distinct(hotel, market_segment) %>%
  arrange(hotel, market_segment)
```

```
# A tibble: 14 × 2
   hotel        market_segment
   <chr>        <chr>
 1 City Hotel   Aviation
 2 City Hotel   Complementary
 3 City Hotel   Corporate
 4 City Hotel   Direct
 5 City Hotel   Groups
 6 City Hotel   Offline TA/TO
 7 City Hotel   Online TA
 8 City Hotel   Undefined
 9 Resort Hotel Complementary
10 Resort Hotel Corporate
11 Resort Hotel Direct
12 Resort Hotel Groups
13 Resort Hotel Offline TA/TO
14 Resort Hotel Online TA
```

> dinstinct() is useful when you want to extract only the unique combinations of one or more columns in a data frame, and remove duplicate rows.

## `mutate()` to add a new variable

```
hotels %>%
  mutate(little_ones = children + babies) %>%
  select(children, babies, little_ones) %>%
  arrange(desc(little_ones))
```

```
# A tibble: 119,390 × 3
   children babies little_ones
      <dbl>  <dbl>       <dbl>
 1       10      0          10
 2        0     10          10
 3        0      9           9
 4        2      1           3
 5        2      1           3
 6        2      1           3
 7        3      0           3
 8        2      1           3
 9        2      1           3
10        3      0           3
# … with 119,380 more rows
```

What are these functions doing? How do to the same in base R?

> Remember vector arithmetic? We can do similar things in homework 1 using `mutate()`

> Remember vector arithmetic? We can do similar things in homework 1 using `mutate()`

**HW1, Problem 4.1** | HW1, Problem 5.1 | HW1, Problem 7.5

```r
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)

city_temps %>% mutate(Celsius_temp = 5/9 * (temperature - 32))
```

```
            name temperature Celsius_temp
1        Beijing          35     1.666667
2          Lagos          88    31.111111
3          Paris          42     5.555556
4 Rio de Janeiro          84    28.888889
5       San Juan          81    27.222222
6        Toronto          30    -1.111111
```

> Remember vector arithmetic? We can do similar things in homework 1 using `mutate()`

HW1, Problem 4.1 | **HW1, Problem 5.1** | HW1, Problem 7.5

```r
library(dslabs)
data(murders)
murders %>% mutate(rate = total/population * 100000) %>% head()
```

```
       state abb region population total     rate
1    Alabama  AL  South    4779736   135 2.824424
2     Alaska  AK   West     710231    19 2.675186
3    Arizona  AZ   West    6392017   232 3.629527
4   Arkansas  AR  South    2915918    93 3.189390
5 California  CA   West   37253956  1257 3.374138
6   Colorado  CO   West    5029196    65 1.292453
```

> Remember vector arithmetic? We can do similar things in homework 1 using `mutate()`

HW1, Problem 4.1 | HW1, Problem 5.1 | **HW1, Problem 7.5**

```
murders %>% mutate(rank = rank(population)) %>%
  arrange(desc(rank)) %>% head()
```

```
         state abb        region population total rank
1   California  CA          West   37253956  1257   51
2        Texas  TX         South   25145561   805   50
3      Florida  FL         South   19687653   669   49
4     New York  NY     Northeast   19378102   517   48
5     Illinois  IL North Central   12830632   364   47
6 Pennsylvania  PA     Northeast   12702379   457   46
```

## `count()` to create frequency tables

```r
# alphabetical order by default
hotels %>%
  count(market_segment)
```

```
# A tibble: 8 × 2
  market_segment     n
  <chr>          <int>
1 Aviation         237
2 Complementary    743
3 Corporate       5295
4 Direct         12606
5 Groups         19811
6 Offline TA/TO  24219
7 Online TA      56477
8 Undefined          2
```

```r
# descending frequency order
hotels %>%
  count(market_segment,
        sort = TRUE)
```

```
# A tibble: 8 × 2
  market_segment     n
  <chr>          <int>
1 Online TA      56477
2 Offline TA/TO  24219
3 Groups         19811
4 Direct         12606
5 Corporate       5295
6 Complementary    743
7 Aviation         237
8 Undefined          2
```

- Base R version: `table()`

## count() and arrange()

```r
# ascending frequency order
hotels %>%
  count(market_segment) %>%
  arrange(n)
```

```
# A tibble: 8 × 2
  market_segment     n
  <chr>          <int>
1 Undefined          2
2 Aviation         237
3 Complementary    743
4 Corporate       5295
5 Direct         12606
6 Groups         19811
7 Offline TA/TO  24219
8 Online TA      56477
```

```r
# descending frequency order
# just like adding sort = TRUE
hotels %>%
  count(market_segment) %>%
  arrange(desc(n))
```

```
# A tibble: 8 × 2
  market_segment     n
  <chr>          <int>
1 Online TA      56477
2 Offline TA/TO  24219
3 Groups         19811
4 Direct         12606
5 Corporate       5295
6 Complementary    743
7 Aviation         237
8 Undefined          2
```

## `count()` for multiple variables

```
hotels %>%
  count(hotel, market_segment)
```

```
# A tibble: 14 × 3
   hotel        market_segment     n
   <chr>        <chr>          <int>
 1 City Hotel   Aviation         237
 2 City Hotel   Complementary    542
 3 City Hotel   Corporate       2986
 4 City Hotel   Direct          6093
 5 City Hotel   Groups         13975
 6 City Hotel   Offline TA/TO  16747
 7 City Hotel   Online TA      38748
 8 City Hotel   Undefined          2
 9 Resort Hotel Complementary    201
10 Resort Hotel Corporate       2309
11 Resort Hotel Direct          6513
12 Resort Hotel Groups          5836
13 Resort Hotel Offline TA/TO   7472
14 Resort Hotel Online TA      17729
```

## Order affects output when you `count()`

```r
# hotel type first
hotels %>%
  count(hotel, market_segment)
```

```
# A tibble: 14 × 3
   hotel        market_segment     n
   <chr>        <chr>          <int>
 1 City Hotel   Aviation         237
 2 City Hotel   Complementary    542
 3 City Hotel   Corporate       2986
 4 City Hotel   Direct          6093
 5 City Hotel   Groups         13975
 6 City Hotel   Offline TA/TO  16747
 7 City Hotel   Online TA      38748
 8 City Hotel   Undefined          2
 9 Resort Hotel Complementary    201
10 Resort Hotel Corporate       2309
11 Resort Hotel Direct          6513
12 Resort Hotel Groups          5836
13 Resort Hotel Offline TA/TO   7472
14 Resort Hotel Online TA      17729
```

```r
# market segment first
hotels %>%
  count(market_segment, hotel)
```

```
# A tibble: 14 × 3
   market_segment hotel            n
   <chr>          <chr>        <int>
 1 Aviation       City Hotel     237
 2 Complementary  City Hotel     542
 3 Complementary  Resort Hotel   201
 4 Corporate      City Hotel    2986
 5 Corporate      Resort Hotel  2309
 6 Direct         City Hotel    6093
 7 Direct         Resort Hotel  6513
 8 Groups         City Hotel   13975
 9 Groups         Resort Hotel  5836
10 Offline TA/TO  City Hotel   16747
11 Offline TA/TO  Resort Hotel  7472
12 Online TA      City Hotel   38748
13 Online TA      Resort Hotel 17729
14 Undefined      City Hotel       2
```

## `summarize()` for summary stats

```r
# mean average daily rate for all bookings
hotels %>%
  summarize(mean_adr = mean(adr))
```

```
# A tibble: 1 × 1
  mean_adr
     <dbl>
1     102.
```

- `summarize()` changes the data frame entirely
- Rows are collapsed into a single summary statistic
- Columns that are irrelevant to the calculation are removed

## `summarize()` is often used with `group_by()`

- For grouped operations
- There are two types of `hotel`, city and resort hotels
- We want the mean daily rate for bookings at city vs. resort hotels

```
hotels %>%
  group_by(hotel) %>%
  summarize(mean_adr = mean(adr))
```

```
# A tibble: 2 × 2
  hotel        mean_adr
  <chr>           <dbl>
1 City Hotel      105.
2 Resort Hotel     95.0
```

- `group_by()` can be used with more than one group

## Multiple summary statistics

`summarize` can be used for multiple summary statistics as well.

```
hotels %>%
  summarize(
    n = n(), # frequencies
    min_adr = min(adr),
    mean_adr = mean(adr),
    median_adr = median(adr),
    max_adr = max(adr)
    )
```

```
# A tibble: 1 × 5
       n min_adr mean_adr median_adr max_adr
   <int>   <dbl>    <dbl>      <dbl>   <dbl>
1 119390   -6.38     102.       94.6    5400
```

## pull(): access column data as a vector or a number

After the summarize result, it is a data frame (or tibble). not a vector or number.

```
# mean average daily rate for all bookings
hotels %>%
  summarize(mean_adr = mean(adr))
```

```
# A tibble: 1 × 1
  mean_adr
     <dbl>
1     102.
```

If we want to access the number, we can use the `pull()`

```
hotels %>%
  summarize(mean_adr = mean(adr)) %>% pull(mean_adr)
```

```
[1] 101.8311
```

Another example:

```
hotels %>%
  group_by(hotel) %>%
  summarize(mean_adr = mean(adr)) %>% pull(mean_adr)
```

```
[1] 105.30447  94.95293
```

This can be useful when you want to assign a variable based on the result you calculated from the tidyverse workflow.

```
mean_adr = hotels %>%
  summarize(mean_adr = mean(adr)) %>% pull(mean_adr)
```

## rename(): rename an existing column

The syntax is `rename(new_name = old_name)`.

Here we rename hotel column into hotel_name.

```
hotels %>% select(hotel:lead_time) %>%
    rename(hotel_name = hotel) %>% head()
```

```
# A tibble: 6 × 3
  hotel_name   is_canceled lead_time
  <chr>              <dbl>     <dbl>
1 Resort Hotel           0       342
2 Resort Hotel           0       737
3 Resort Hotel           0         7
4 Resort Hotel           0        13
5 Resort Hotel           0        14
6 Resort Hotel           0        14
```

## `join` family

Dplyr has a powerful group of join operations, which join together a pair of data frames based on a variable or set of variables present in both data frames that uniquely identify all observations. These variables are called keys.

- inner_join: Only the rows with keys present in both datasets will be joined together.
- left_join: Keeps all the rows from the first dataset, regardless of whether in second dataset, and joins the rows of the second that have keys in the first.
- right_join: Keeps all the rows from the second dataset, regardless of whether in first dataset, and joins the rows of the first that have keys in the second.
- full_join: Keeps all rows in both datasets. Rows without matching keys will have NA values for those variables from the other dataset.

## `join` family

To practice with the join functions, we can use a couple of built-in R datasets.

**Dataset** Inner join Left join Right join Full join

```
data(band_instruments2)
head(band_instruments2)
```

```
# A tibble: 3 × 2
  artist plays 
  <chr>  <chr> 
1 John   guitar
2 Paul   bass  
3 Keith  guitar
```

```
data(band_members)
head(band_members)
```

```
# A tibble: 3 × 2
  name  band   
  <chr> <chr>  
1 Mick  Stones 
2 John  Beatles
3 Paul  Beatles
```

## `join` family

To practice with the join functions, we can use a couple of built-in R datasets.

Dataset    **Inner join**    Left join    Right join    Full join

```
# Inner join
band_members %>% inner_join(band_instruments2, by = join_by(name == a
```

```
# A tibble: 2 × 3
  name  band    plays 
  <chr> <chr>   <chr> 
1 John  Beatles guitar
2 Paul  Beatles bass  
```

## `join` family

To practice with the join functions, we can use a couple of built-in R datasets.

Dataset    Inner join    **Left join**    Right join    Full join

```
# Left join
band_members %>% left_join(band_instruments2, by = join_by(name == ar
```

```
# A tibble: 3 × 3
  name  band    plays 
  <chr> <chr>   <chr> 
1 Mick  Stones  <NA>  
2 John  Beatles guitar
3 Paul  Beatles bass  
```

## `join` family

To practice with the join functions, we can use a couple of built-in R datasets.

Dataset | Inner join | Left join | **Right join** | Full join

```
# Right join
band_members %>% right_join(band_instruments2, by = join_by(name == a
```

```
# A tibble: 3 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith <NA>    guitar
```

## `join` family

To practice with the join functions, we can use a couple of built-in R datasets.

Dataset     Inner join     Left join     Right join     **Full join**

```
# Full join
band_members %>% full_join(band_instruments2, by = join_by(name == ar
```

```
# A tibble: 4 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```

## Introduction to data visualization

- Why we need data visualization?

```r
library(dslabs)
data(murders)
head(murders)
```

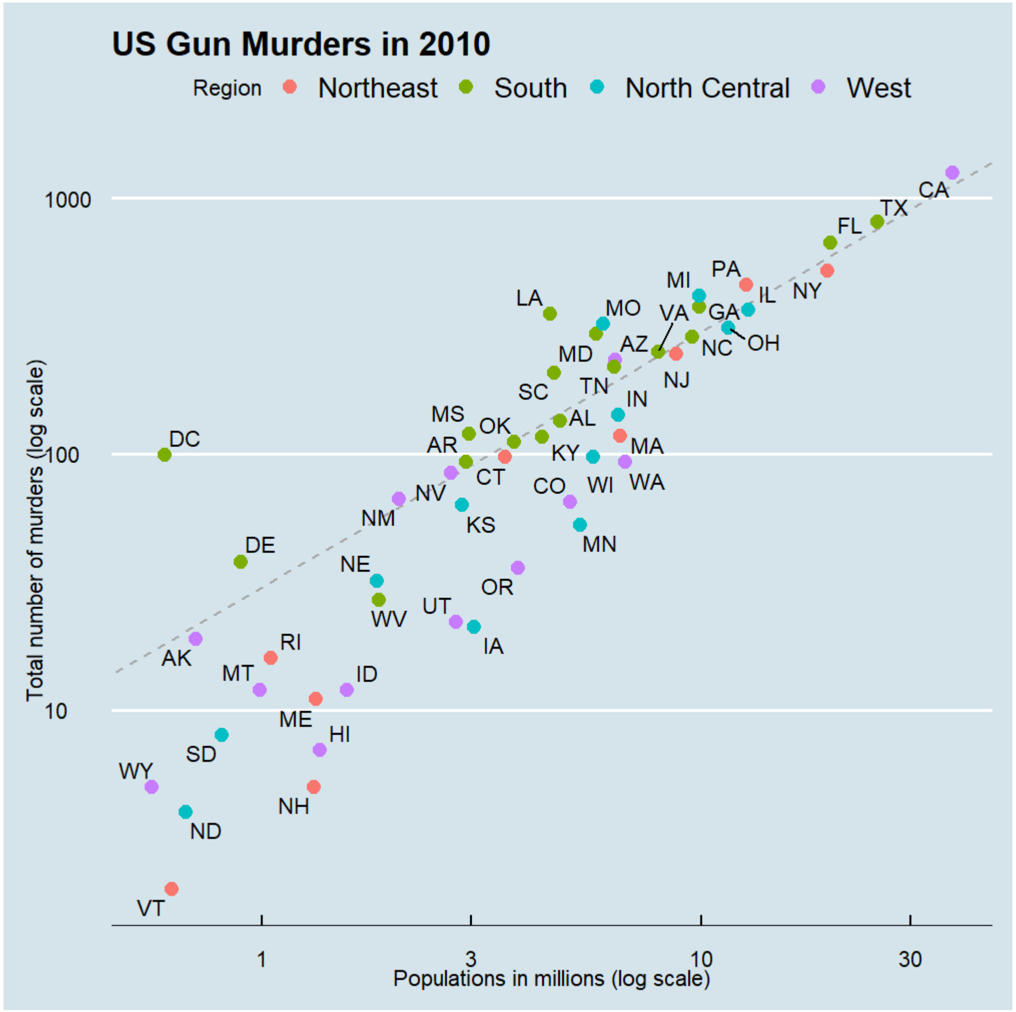
```
       state abb region population total
1    Alabama  AL  South    4779736   135
2     Alaska  AK   West     710231    19
3    Arizona  AZ   West    6392017   232
4   Arkansas  AR  South    2915918    93
5 California  CA   West   37253956  1257
6   Colorado  CO   West    5029196    65
```

- How is variable distributed?

- How can we identity patterns or relationships between variables

In contrast, the answer to all the questions above are readily available from examining this plot:

**Picture** Code explanation



In contrast, the answer to all the questions above are readily available from examining this plot:

Picture Code explanation

```r
library(tidyverse)
library(ggthemes)
library(ggrepel)
library(ggplot2)
r <- murders |>
  summarize(pop=sum(population), tot=sum(total)) |>
  mutate(rate = tot/pop*10^6) |> pull(rate)
murders |> ggplot(aes(x = population/10^6, y = total, label = abb))
  geom_abline(intercept = log10(r), lty=2, col="darkgrey") +
  geom_point(aes(color=region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name="Region") +
  theme_economist()
```

In contrast, the answer to all the questions above are readily available from examining this plot:

Picture Code **explanation**

Each state in the dataset was identified in this plot as a colored point with a label next to it. The total number of murders is shown on the y axis in log scale, and the populations are shown on the x axis in millions. The state name is indicated by the text label next to the points, and the color designates the state region. The average murder rate in the US was added as a gray line (in millions).

## Why we need data visualization

We are reminded of the saying **"a picture is worth a thousand words"**. Data visualization provides a powerful way to communicate a data-driven finding.

Data visualization is the strongest tool of what we call *exploratory data analysis* (EDA). John W. Tukey, considered the father of EDA, once said,

> > "The greatest value of a picture is when it forces us to notice what we never expected to see."

Many widely used data analysis tools were initiated by discoveries made via EDA. EDA is perhaps the most important part of data analysis, yet it is one that is often overlooked.

The growing availability of informative datasets and software tools has led to increased reliance on **data visualizations** across many industries, academia, and government.

## Benefits of data visualization:

**Communication**: Data visualization provides a powerful way to communicate complex information to both technical and non-technical audiences.

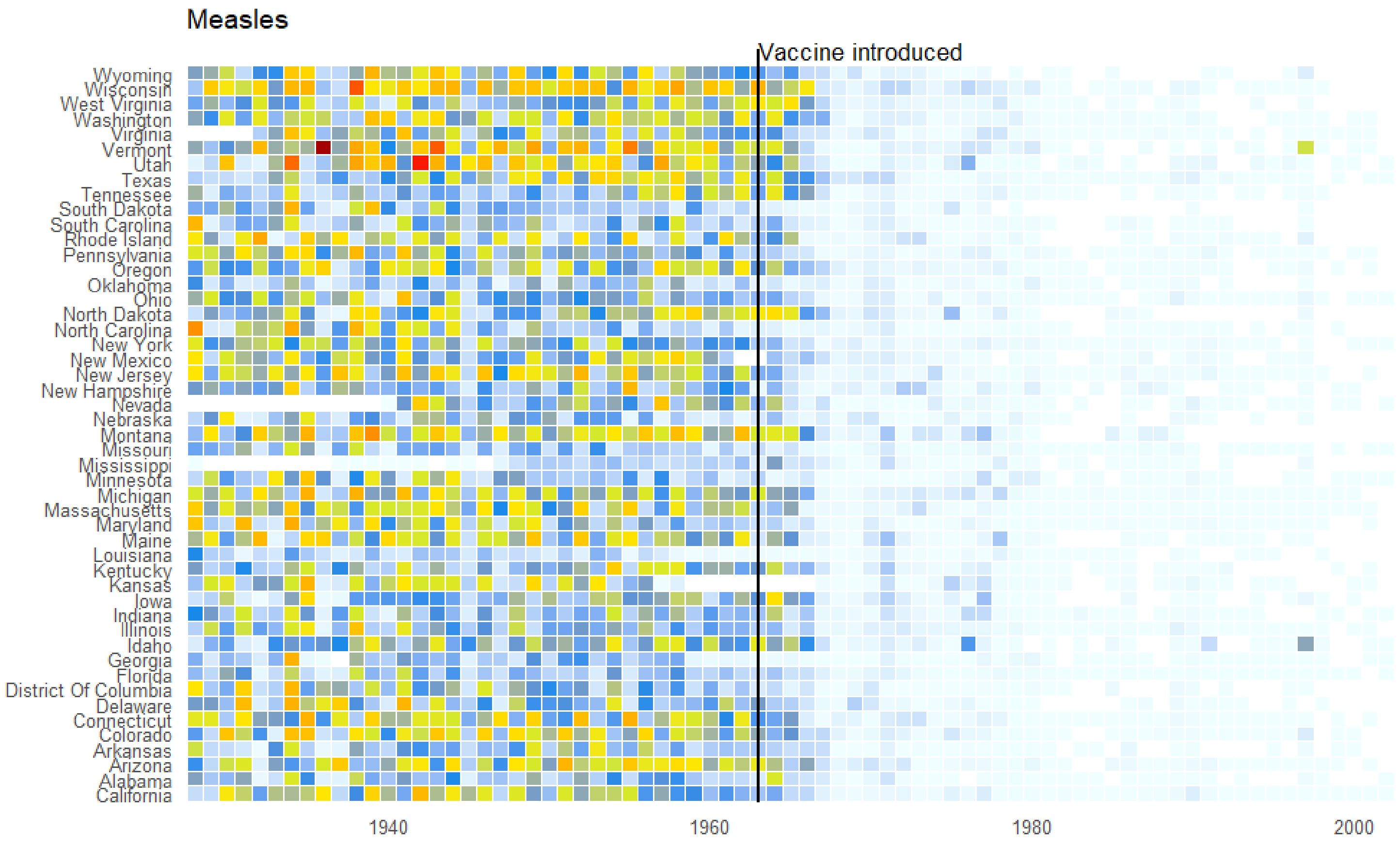
**Exploration**: Data visualization allows us to explore data and identify patterns or trends that may not be apparent from numerical summaries alone.

**Identification of errors or outliers**: Data visualization can help us identify potential errors or outliers in our data that may impact our analysis.

**Hypothesis generation**: Data visualization can help generate new hypotheses or questions for further investigation.

## Another example

Picture Code explanation



# Another example

Picture | **Code** | explanation

```r
#knitr::include_graphics(file.path(img_path,"wsj-vaccines.png"))
data(us_contagious_diseases)
the_disease <- "Measles"
dat <- us_contagious_diseases |>
  filter(!state%in%c("Hawaii","Alaska") & disease == the_disease) |>
  mutate(rate = count / population * 100000 * 52 / weeks_reporting) |>
  mutate(state = reorder(state, rate))
jet.colors <-
colorRampPalette(c("#F0FFFF", "cyan", "#007FFF", "yellow", "#FFBF00", "orange", "red", "#7F0000"), bias
the_breaks <- seq(0, 4000, 1000)
dat |> ggplot(aes(year, state, fill = rate)) +
  geom_tile(color = "white", size=0.35) +
  scale_x_continuous(expand=c(0,0)) +
  scale_fill_gradientn(colors = jet.colors(16), na.value = 'white',
                       breaks = the_breaks,
                       labels = paste0(round(the_breaks/1000),"k"),
                       limits = range(the_breaks),
                       name = "") +
  geom_vline(xintercept=1963, col = "black") +
  theme_minimal() +
  theme(panel.grid = element_blank()) +
  coord_cartesian(clip = 'off') +
  ggtitle(the_disease) +
  ylab("") +
  xlab("") +
  theme(legend.position = "bottom", text = element_text(size = 8)) +
  annotate(geom = "text", x = 1963, y = 50.5, label = "Vaccine introduced", size = 3, hjust=0)
```

## Another example

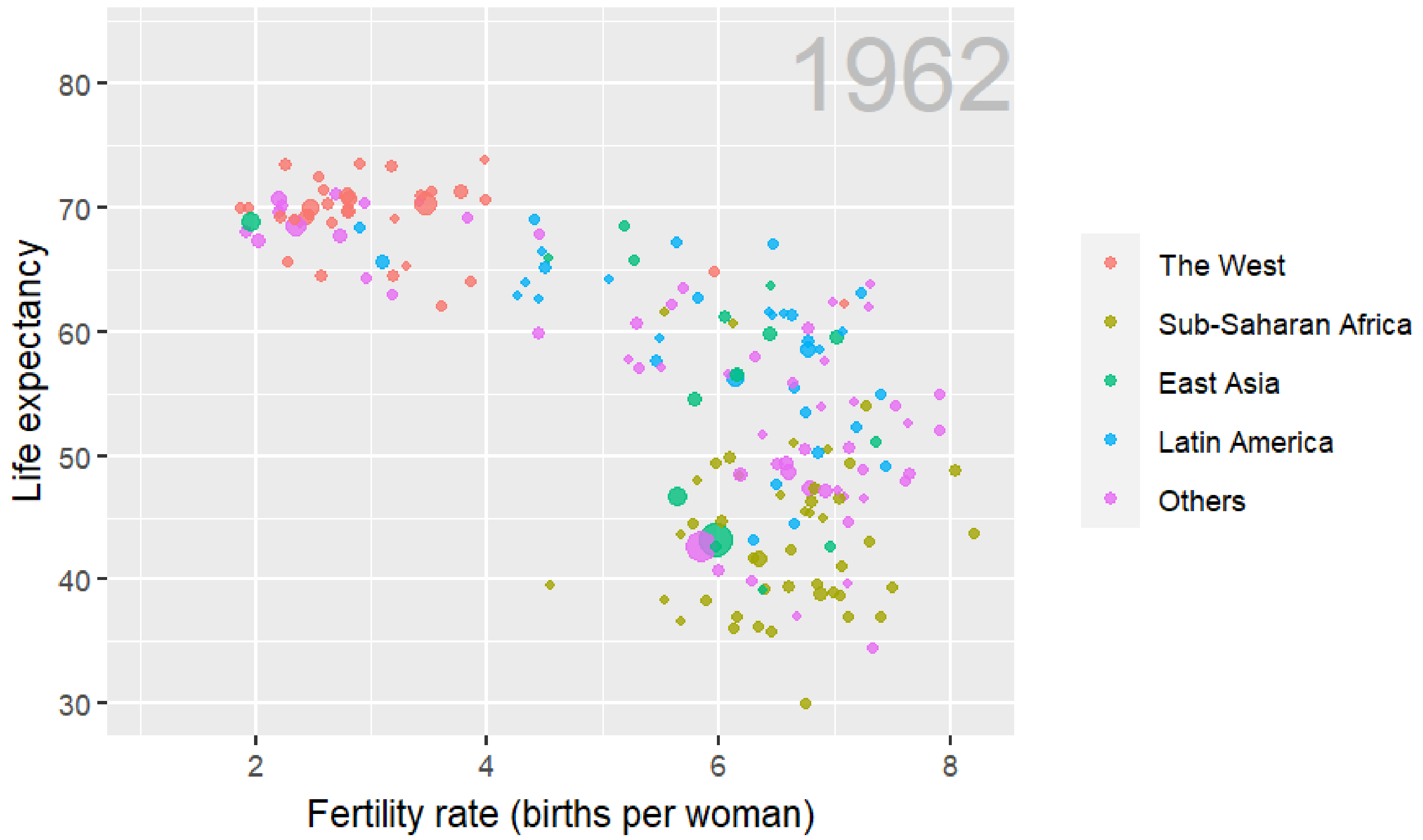
Picture Code explanation

A particularly effective example is a Wall Street Journal article showing data related to the impact of vaccines on battling infectious diseases.

One of the graphs shows measles cases by US state through the years with a vertical line demonstrating when the vaccine was introduced.

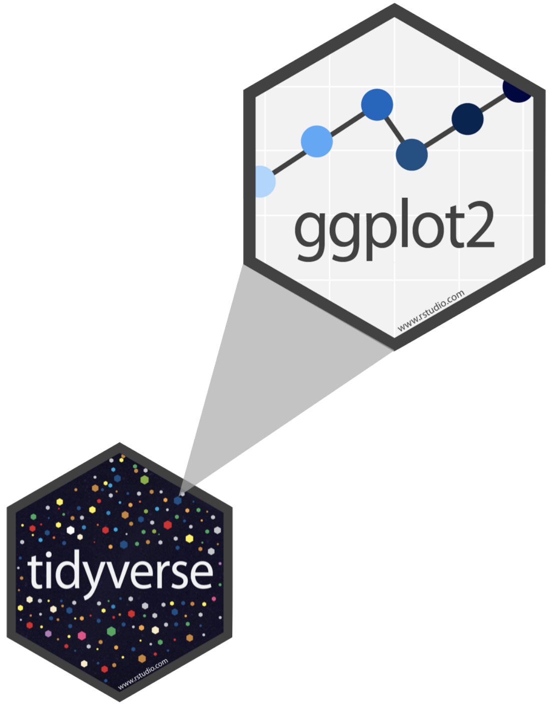
The plot shows the incidence rate of Measles in US states over time (years on the x-axis), represented by colored tiles for each state (on the y-axis). The incidence rate is calculated as the number of cases per 100,000 population per week, averaged over 52 weeks and adjusted for the number of weeks reporting data. States are sorted by their incidence rates, from lowest to highest, and are colored according to a gradient color scale, ranging from blue (low incidence rates) to red (high incidence rates). The plot includes a vertical line indicating the year when the Measles vaccine was introduced (1963). The plot is useful for visualizing how Measles incidence rates varied across US states over time, and how the introduction of the vaccine impacted the incidence rates.

In the talks NewInsights on Poverty, Hans Rosling forces us to notice the unexpected with a series of plots related to world health and economics.



## Data visualization using `ggplot2`

Slide | Words 1 | Words 2



- ggplot2 is the tidyverse's data visualization package
- create relatively **complex** and **aesthetically pleasing** plots
- syntax is **intuitive** and comparatively easy to remember.
- gg in "ggplot2" stands for Grammar of Graphics
- Inspired by the book Grammar of Graphics by Leland Wilkinson

# Data visualization using `ggplot2`

Slide | **Words 1** | Words 2

Throughout the lecture, we will be creating plots using the **ggplot2**^[https://ggplot2.tidyverse.org/] package.

Many other approaches are available for creating plots in R. We chose to use **ggplot2** because it breaks plots into components in a way that permits beginners to create relatively **complex** and **aesthetically pleasing** plots using syntax that is **intuitive** and comparatively easy to remember.

One reason **ggplot2** is generally more intuitive for beginners is that it uses a grammar of graphics^[http://www.springer.com/us/book/9780387245447], the *gg* in **ggplot2**. This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a handful of verbs, nouns and adjectives without having to memorize each specific sentence. Similarly, by learning a handful of **ggplot2** building blocks and its grammar, you will be able to create hundreds of different plots.

## Data visualization using `ggplot2`

Slide Words 1 **Words 2**

Another reason **ggplot2** is easy for beginners is that it is possible to create informative and elegant graphs with relatively simple and readable code.

To use **ggplot2** you will have to learn several functions and arguments. These commands may be hard to memorize, but you can always return back to this tutorial and grab the code you want. Or you can simply perform an internet search for ggplot2 cheat sheet.

## Next week:

Learn how to use `ggplot2` to generate the first example!

## Readings

- Chapter 4:The tidyverse
- Data Wrangling with Tidyverse
- Chapter 7:Introduction to data visualization