Data Handling: Import, Cleaning and Visualisation
Lecture 11: Visualisation and Dynamic Documents

Prof. Dr. Ulrich Matter
(University of St.Gallen)

16/12/2021

1 Data display

In the last part of a data pipeline we are typically dealing with the visualisation of data and statistical results for presentation/communication. Typical output formats are reports, a thesis (BA, MA, Dissertation chapter), interactive dashboards, and websites. R (and particularly RStudio) provides a very flexible framework to manage the steps involved in visualisation/presentation for all of these output formats. A first (low-level) step in preparing data/results for publication is the formatting of data values for publication. Typically, this involves some string operations to make numbers and text look nicer before we show them in a table or graph.

Consider, for example, the following summary statistics.

```r
# load packages and data
library(tidyverse)
data("swiss")
# compute summary statistics
swiss_summary <-
  summarise(swiss,
          avg_education = mean(Education, na.rm = TRUE),
          avg_fertility = mean(Fertility, na.rm = TRUE),
          N = n()
          )
swiss_summary
```

```
##   avg_education avg_fertility  N
## 1      10.97872      70.14255 47
```

Quite likely, we do not want to present these numbers with that many decimal places. The function `round()` can take care of this.

```r
swiss_summary_rounded <- round(swiss_summary, 2)
swiss_summary_rounded
```

```
##   avg_education avg_fertility  N
## 1         10.98         70.14 47
```

More specific formatting of numeric values is easier when coercing the numbers to character strings (text).[1] For example, depending on the audience (country/region) to which we want to communicate our results, different standards of how to format numbers are expected. In the English-speaking world it is quite common to use . as decimal mark, in the German-speaking world it is rather common to use ,. The `format()`-function provides an easy way to format numbers in this way (once they are coerced to character).

```
swiss_summary_formatted <- format(swiss_summary_rounded, decimal.mark=",")
swiss_summary_formatted
```

```
##   avg_education avg_fertility  N
## 1         10,98         70,14 47
```

R also provides various helpful functions to better format/display text strings. See, for example:

- Uppercase/lowercase: `toupper()`/`tolowe()`.
- Remove white spaces: `trimws()`,

```
string <- "AbCD "
toupper(string)
```

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

```
tolower(string)
```

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

```
trimws(tolower(string))
```

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

2 Data visualisation with `ggplot2`

A key technique to convincingly communicate statistical results and insights from data is visualisation. How can we visualise raw data and insights gained from statistical models with R? It turns out that R is a really useful tool for data visualisation, thanks to its very powerful graphics engine (i.e., the underlying low-level R functions that handle things like colors, shapes, etc.). Building on this graphics engine, there are particularly three R packages with a variety of high-level functions to plot data in R:

- The original `graphics` package (R Core Team (2018); shipped with the base R installation).
- The `lattice` package (Sarkar 2008), an implementation of the original Bell Labs 'Trellis' system.
- The `ggplot2` package (Wickham 2016), an implementation of Leland Wilkinson's 'Grammar of Graphics'.

While all of these packages provide well-documented high-level R functions to plot data, their syntax differs in some important ways. For R beginners it thus makes sense to first learn how to generate plots in R with *one* of these packages. Here, we focus on `ggplot2` because it is part of the `tidyverse`.

2.1 'Grammar of Graphics'

A few years back, Leland Wilkinson (statistician and computer scientist) wrote an influential book called 'The Grammar of Graphics'. In this book, Wilkinson develops a formal description ('grammar') of graphics used in statistics, illustrating how different types of plots (bar plot, histogram, etc.) are special cases of an underlying

---

[1]Note that this step only makes sense if we are sure that the numeric values won't be further analyzed or used in a plot (except for labels).

framework. In short, his idea was that we can think of graphics as consisting of different design-layers and thus can build and describe graphics/plots layer by layer (see here for an illustration of this idea).

This framework got implemented in R with the prominent `ggplot2`-package, building on the already powerful R graphics engine. The result is a user-friendly environment to visualise data with enormous potential to plot almost any graphic illustrating data.

2.2 `ggplot2` basics

Using `ggplot2` to generate a basic plot in R is quite simple. Basically, it involves three key points:

1. The data must be stored in a `data.frame`/`tibble` (in tidy format).
2. The starting point of a plot is always the function `ggplot()`.
3. The first line of plot code declares the data and the 'aesthetics' (e.g., which variables are mapped to the x-/y-axes):

```
ggplot(data = my_dataframe, aes(x= xvar, y= yvar))
```

2.3 Tutorial

In the following, we learn the basic functionality of `ggplot` by applying it to the `swiss` dataset.

2.3.1 Loading/preparing the data

First, we load and inspect the data. Among other variables it contains information about the share of inhabitants of a given Swiss province who indicate to be of Catholic faith (and not Protestant).

```
# load the R package
library(ggplot2)
# load the data
data(swiss)
# get details about the data set
# ?swiss
# inspect the data
head(swiss)
```

```
##               Fertility Agriculture Examination Education Catholic Infant.Mortality
## Courtelary        80.2        17.0          15        12     9.96             22.2
## Delemont          83.1        45.1           6         9    84.84             22.2
## Franches-Mnt      92.5        39.7           5         5    93.40             20.2
## Moutier           85.8        36.5          12         7    33.77             20.3
## Neuveville        76.9        43.5          17        15     5.16             20.6
## Porrentruy        76.1        35.3           9         7    90.57             26.6
```

As we do not only want to use this continuous measure in the data visualisation, we generate an additional factor variable called `Religion` which has either the value `'Protestant'` or `'Catholic'` depending on whether more then 50 percent of the inhabitants of the province are Catholics.

```
# code province as 'Catholic' if more than 50% are catholic
swiss$Religion <- 'Protestant'
swiss$Religion[50 < swiss$Catholic] <- 'Catholic'
swiss$Religion <- as.factor(swiss$Religion)
```

2.3.2   Data and aesthetics

We initiate the most basic plot with `ggplot()` by defining which data to use and in the plot aesthetics which variable to use on the x and y axes. Here, we are interested in whether the level of education beyond primary school in a given district is related with how well draftees from the same district do in a standardized army examination (% of draftees that get the highest mark in the examination).

```
ggplot(data = swiss, aes(x = Education, y = Examination))
```
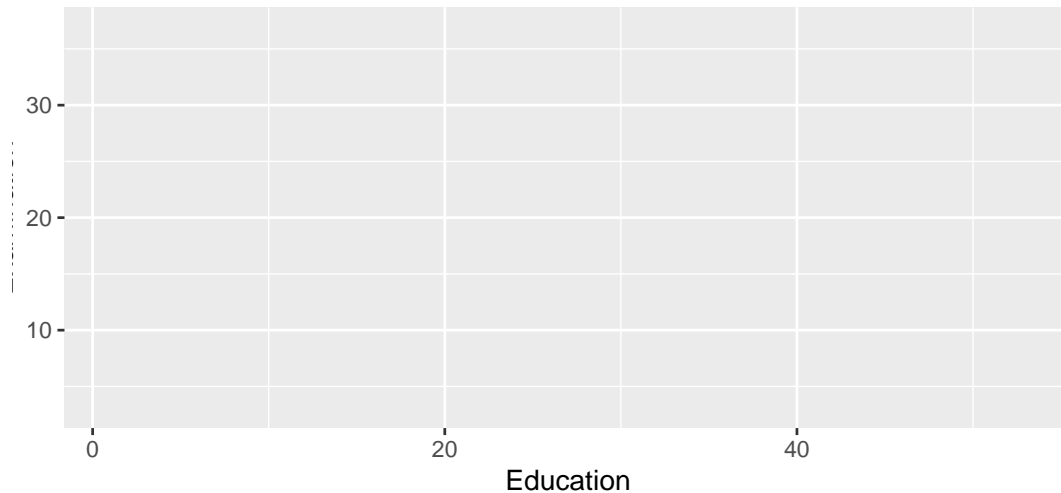

As we have not yet defined according to what rules the data shall be visualised, all we get is an empty 'canvas' and the axes (with the respective label and ticks indicating the range of the values).

2.4   Geometries (~ type of plot)

In order to actually plot the data we have to define the 'geometries', defining according to which function the data should be mapped/visualised. In other words, geometries define which 'type of plot' we use to visualise the data (histogram, lines, points, etc.). In the example code below, we use `geom_point()` to get a simple point plot.

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point()
```
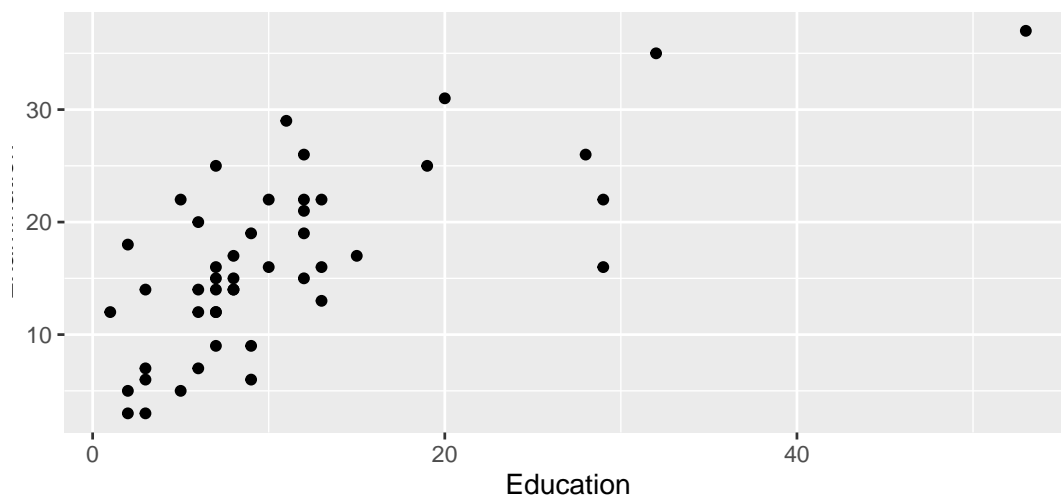
The result indicates that there is a positive correlation between the level of education and how well draftees do in the examination. We want to better understand this correlation. Particularly, what other factors could drive this picture.

2.4.1   Facets

According to a popular thesis, the protestant reformation and the spread of the protestant movement in Europe was driving the development of compulsory schooling. It would thus be reasonable to hypothesize that the picture we see is partly driven by differences in schooling between Catholic and Protestant districts. In order to make such differences visible in the data, we use 'facets' to show the same plot again, but this time separating observations from Catholic and Protestant districts:

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point() +
    facet_wrap(~Religion)
```
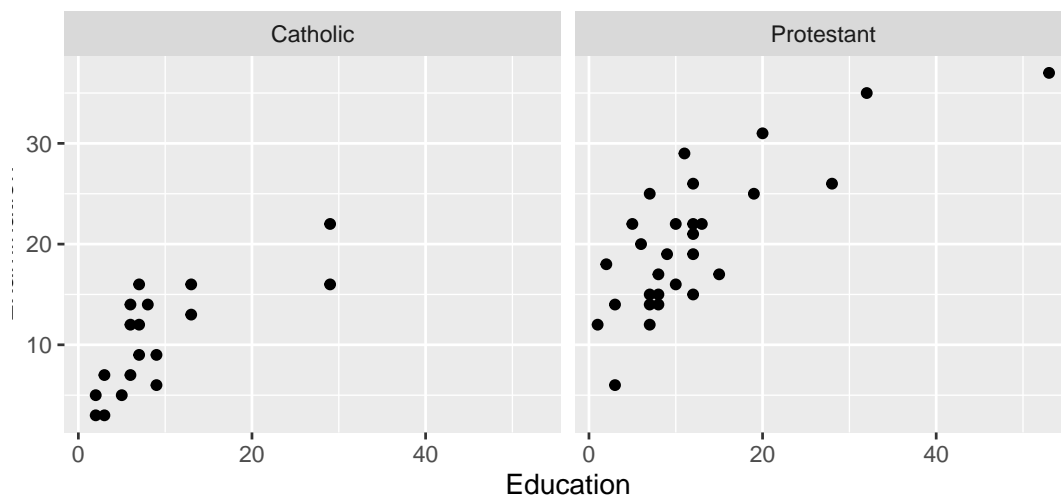
Draftees from protestant districts tend to do generally better (which might be an indication of better primary schools, or a generally stronger focus on scholastic achievements of Protestant children). However, the relationship between education (beyond primary schools) and examination success seems to hold for either type of districts.

2.4.2   Additional layers and statistics

Let's visualise this relationship more clearly by drawing trend-lines through the scatter diagrams. Once with the non-parametric 'loess'-approach and once forcing a linear model on the relationship between the two variables.
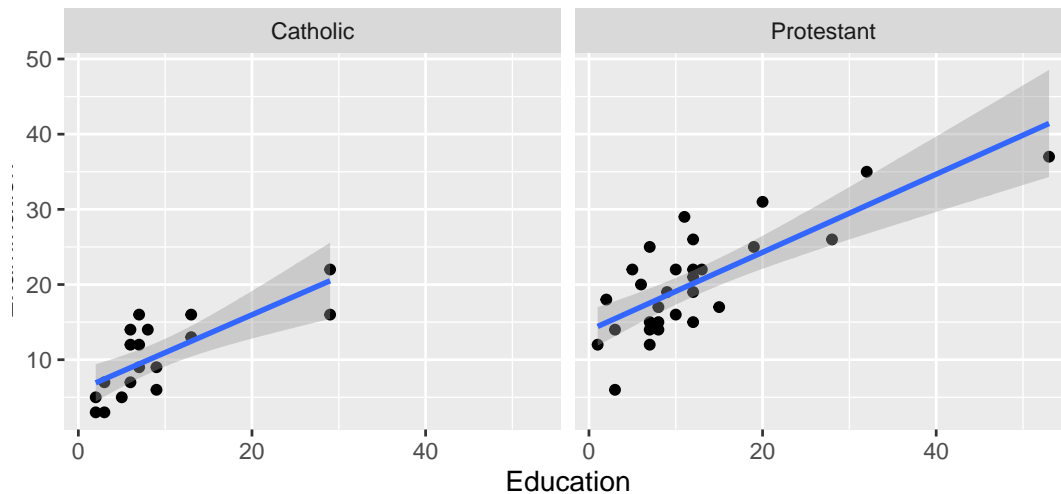
```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point() +
    geom_smooth(method = 'loess') +
    facet_wrap(~Religion)
```
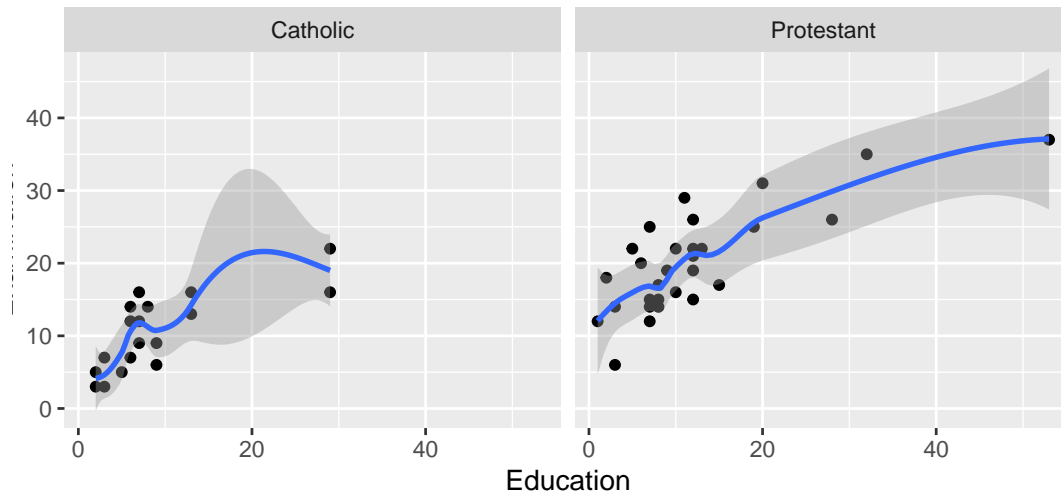
```
## `geom_smooth()` using formula 'y ~ x'
```

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point() +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion)
```

```
## `geom_smooth()` using formula 'y ~ x'
```


2.4.3 Additional aesthetics

Knowing a little bit about Swiss history and geography, we realize that particularly rural cantons in mountain regions remained Catholic during the reformation. In addition, cantonal school systems historically took into account that children have to help their parents on the farms during the summers. Thus in some rural cantons schools were closed from spring until autumn. Hence, we might want to indicate in the plot which point refers to a predominantly agricultural district. We use the aesthetics of the point geometry to color the points according to the 'Agriculture'-variable (the % of males involved in agriculture as occupation).

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point(aes(color = Agriculture)) +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion)
```

```
## `geom_smooth()` using formula 'y ~ x'
```
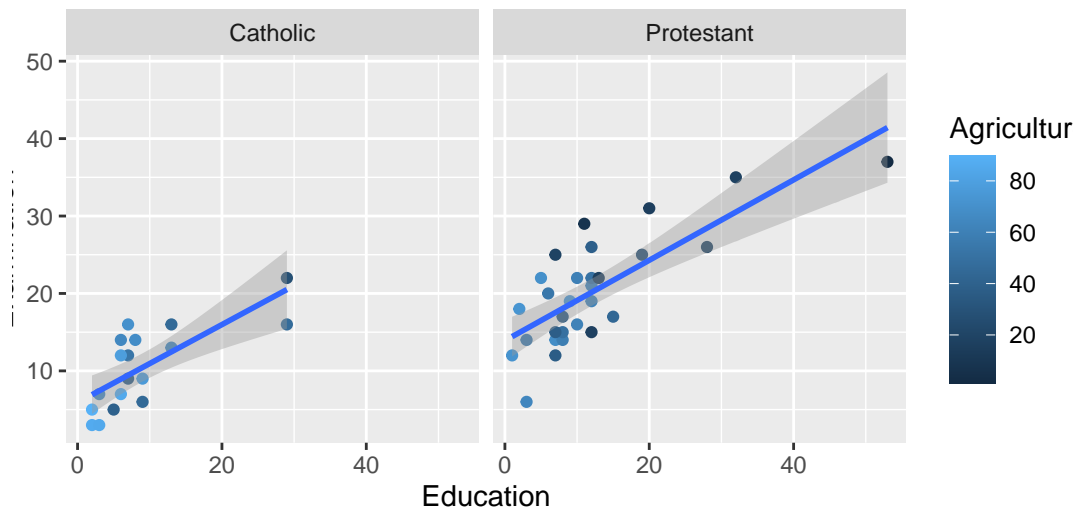

The resulting picture is in line with what we have expected. Overall, the districts with a lower share of occupation in agriculture tend to have rather higher levels of education as well as higher achievements in the examination.

2.4.4 Themes: Fine-tuning the plot

Finally, there are countless options to further refine the plot. For example, we can easily change the orientation/coordinates of the plot:

```r
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point(aes(color = Agriculture)) +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion) +
    coord_flip()
```

```
## `geom_smooth()` using formula 'y ~ x'
```
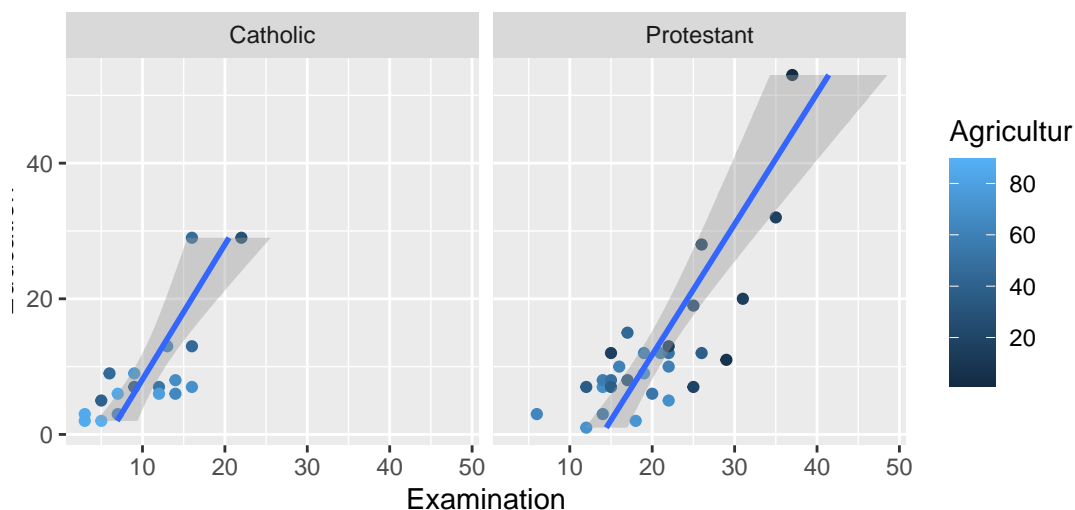

In addition, the `theme()`-function allows to change almost every aspect of the plot (margins, font face, font size, etc.). For example, we might prefer to have the plot legend at the bottom and have larger axis labels.

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point(aes(color = Agriculture)) +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion) +
    theme(legend.position = "bottom", axis.text=element_text(size=12) )
```
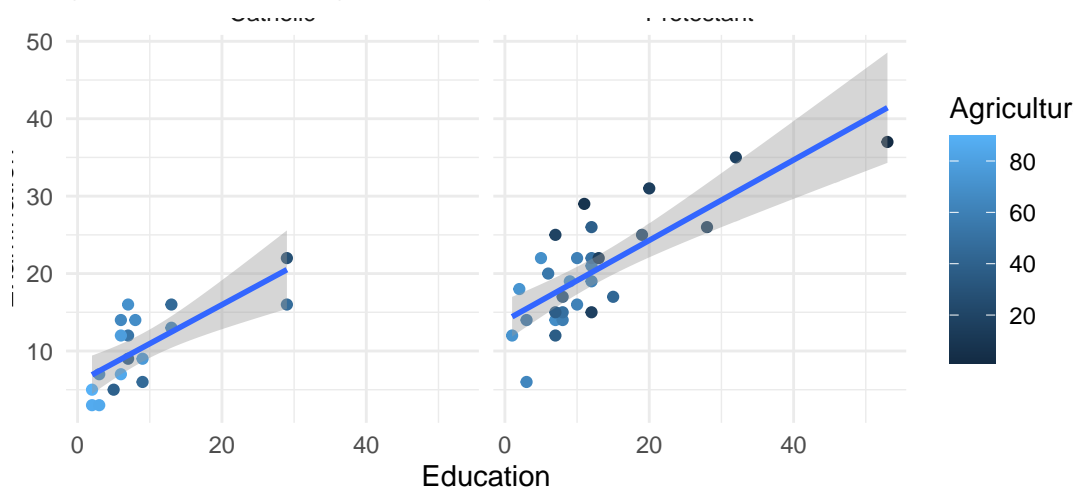
```
## `geom_smooth()` using formula 'y ~ x'
```
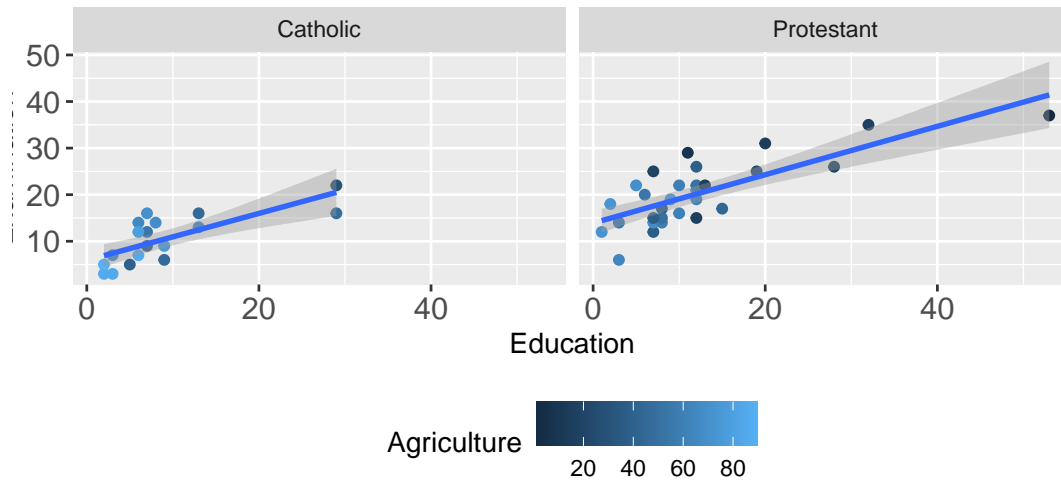

Moreover, several theme-templates offer ready-made designs for plots:

```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point(aes(color = Agriculture)) +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion) +
    theme_minimal()
```
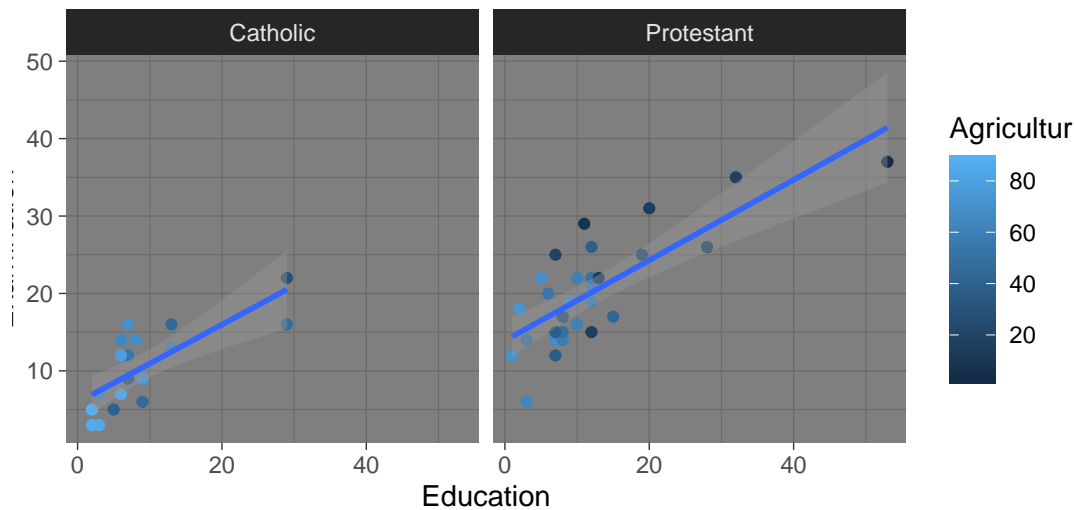
```
## `geom_smooth()` using formula 'y ~ x'
```



```
ggplot(data = swiss, aes(x = Education, y = Examination)) +
    geom_point(aes(color = Agriculture)) +
    geom_smooth(method = 'lm') +
    facet_wrap(~Religion) +
    theme_dark()
```

```
## `geom_smooth()` using formula 'y ~ x'
```


3   Dynamic documents

Dynamic documents are a way to directly/dynamically integrate results of an analysis in R (numbers, tables, plots) in written text (a report, thesis, slide set, website, etc.). That is, we can write a report in so-called 'R-Markdown' format and place directly in the same document 'chunks' of R code which we want to be executed each time we 'knit' the report. Knitting the document means that the following steps are executed under the hood:

1. The code in the R chunks is executed and the results cached and formatted for print.
2. The formatted R output is embedded in a so-called 'Markdown'-file (`.md`).
3. The markdown-file is rendered as either a PDF, HTML, or Word-file (with additional formatting options, depending on the output format).

The entire procedure from importing, cleaning, analyzing, and visualising data can thus be combined in one document, based on which we can generate a meaningful output to communicate our results (in fact, the initial dashboard example, as well as all notes and slides of this course are generated in this way).

4   Tutorial: How to give the wrong impression with data visualization

4.1   Background and aim

Referring to the supposed trade-off between public health and economic performance in the context of the SARS-CoV-2 pandemic, Georgia Governor Brian Kemp has "reopened the state on April 24 [2020], citing a downward trend of COVID-19 cases being reported by the Georgia Department of Health [. . .] Georgians have been relying on this data (number of cases, hospitalizations and deaths in their areas) to determine whether or not it's safe for them to go out." (FIRSTCOAST NEWS) It later turned out that there was no downward trend at all but that the data on Covid cases was intentionally visualized in such a way to give the impression of a downward trend. In this tutorial we aim to replicate the doctored downward trend visualization, using the orignal data and `ggplot2`.[2]

---

[2]Note that the original data is not 100% identical with the data used in the original plot. The reason is that Covid-numbers are updated/corrected over time and the original plot use very recent data at the time.
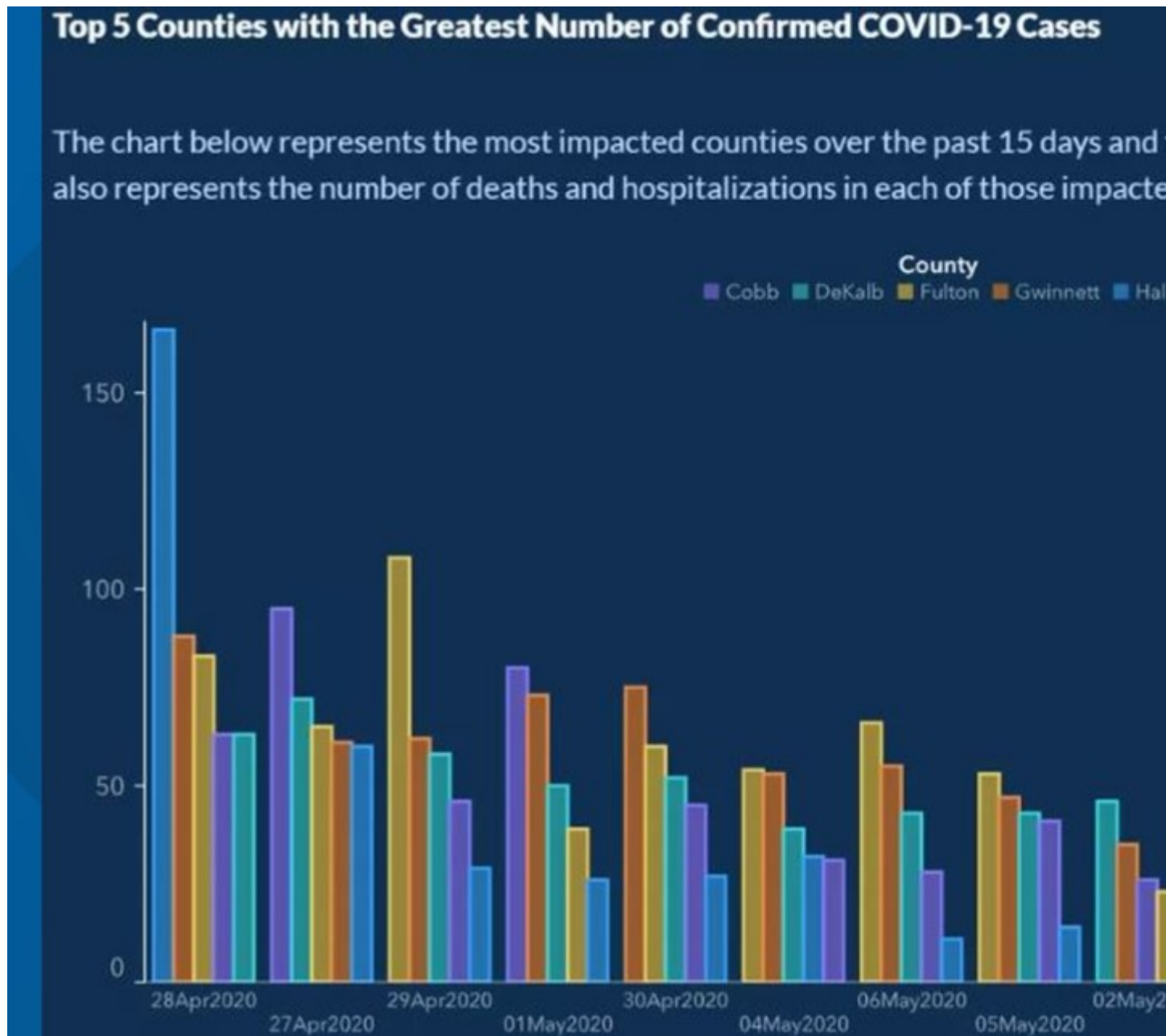
4.2 Data import and preparation

The original raw data is provided in a zip-file (compressed file). The code chunk below shows some tricks of how to download and import data from an online zip-file as part of a simple data analytics script. First, we generate a temporary file with `tempfile()`. We then download the compressed zip file containing the Covid-data to the temporary file via `download.file(DATA_URL, destfile = tmp)`. Finally, we use `unzip(tmp, files = "epicurve_rpt_date.csv" )` in order to decompress the one CSV-file in the zip-file we want to work with. After that, we can simply import the CSV-file the usual way (with the `read_cav()` function).

```
# SET UP ----------------

# load packages
```

```r
library(tidyverse)
library(ggplot2)

# fix vars
DATA_URL <- "https://ga-covid19.ondemand.sas.com/docs/ga_covid_data.zip"
COUNTIES <- c("Cobb", "DeKalb", "Fulton", "Gwinnett", "Hall")
MIN_DATE <- as.Date("2020-04-26")
MAX_DATE <- as.Date("2020-05-09")


# FETCH/IMPORT DATA ----------------------

# create a temporary file to store the downloaded zipfile
tmp <- tempfile()
download.file(DATA_URL, destfile = tmp)
# unzip the file we need
path <- unzip(tmp, files = "epicurve_rpt_date.csv" )
# read the data
cases <- read_csv(path)
```

```
## Parsed with column specification:
## cols(
##    measure = col_character(),
##    county = col_character(),
##    report_date = col_date(format = ""),
##    cases = col_double(),
##    deaths = col_double(),
##    cases_cum = col_double(),
##    death_cum = col_double(),
##    moving_avg_cases = col_double(),
##    moving_avg_deaths = col_double(),
##    antigen_cases = col_double(),
##    probable_deaths = col_double(),
##    antigen_cases_cum = col_double(),
##    probable_deaths_cum = col_double(),
##    total_cases = col_double(),
##    total_cases_cum = col_double(),
##    moving_avg_antigen_cases = col_double(),
##    moving_avg_probable_deaths = col_double(),
##    moving_avg_total_cases = col_double()
## )
```

```r
# inspect the data
head(cases)
```

```
## # A tibble: 6 x 18
##    measure county report_date cases deaths cases_cum death_cum moving_avg_cases moving_avg_deat~
##    <chr>   <chr>  <date>      <dbl> <dbl>  <dbl>     <dbl>     <dbl>            <dbl>
## 1 state_~ Georg~ 2020-02-01      0     0       0         0             0                0
## 2 state_~ Georg~ 2020-02-02      0     0       0         0             0                0
## 3 state_~ Georg~ 2020-02-03      0     0       0         0             0                0
## 4 state_~ Georg~ 2020-02-04      0     0       0         0             0                0
## 5 state_~ Georg~ 2020-02-05      0     0       0         0             0                0
## 6 state_~ Georg~ 2020-02-06      0     0       0         0             0                0
## # ... with 9 more variables: antigen_cases <dbl>, probable_deaths <dbl>, antigen_cases_cum <dbl>,
```

```
## #   probable_deaths_cum <dbl>, total_cases <dbl>, total_cases_cum <dbl>,
## #   moving_avg_antigen_cases <dbl>, moving_avg_probable_deaths <dbl>, moving_avg_total_cases <dbl>
```

Once the data is imported, we select/filter the part of the dataset used in the original bar plot. Note the tidyverse-functions introduced in previous lectures to efficiently prepare an analytic dataset out of the raw data. At one point, we use the `order()`-function to order the observations according to the report date of covid cases. This is what would be usually done when displaying the frequency of cases over time. This way, in the following plot the x-axis serves as a time axis, displaying the date of the corresponding reported case numbers from the beginning of the observation period on the left to the end of the observation period on the right).

```r
# only observations in the five major counties and during the relevant days
cases <- filter(cases, county %in% COUNTIES, MIN_DATE <= report_date, report_date <= MAX_DATE)
# only the relevant variables
cases <- select(cases, county, report_date, cases)
# order according to date
cases <- cases[order(cases$report_date),]
# we add an additional column in which we treat dates as categories
cases <- mutate(cases, date=factor(report_date))
```

4.3 Plot

First we show what the actual (honest) plot would essentially look like.

```r
ggplot(cases, aes(y=cases, x=date, fill=county)) +
  geom_bar(position = position_dodge2(), stat = "identity") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) # this avoids overlapping of x-axis labels (and is
```
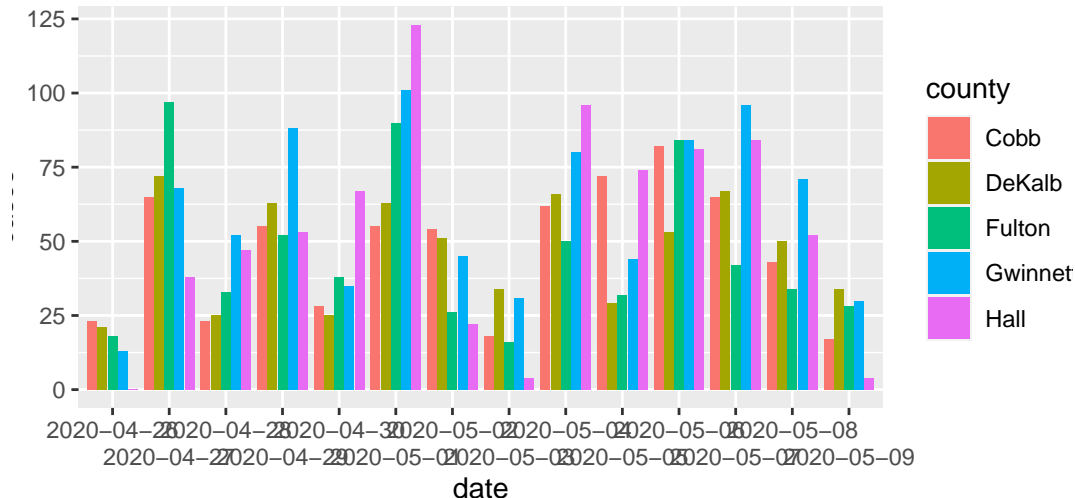

There is no obvious downward trend visible. So let's have a look at what steps are involved in manipulating the visualisation in order to give the wrong impression of a downward trend. Importantly, all the manipulation is purely done via the plotting of the data. We do not touch/manipulate the underlying data (such as removing observations or falsifying numbers).

When comparing the original plot with what we have so far, two things become aparent. First, the order of days on the X-axis is not chronological but seems to be based on the total number of cases per day. Second, the order of the bars of a given day does not follow the alphabetical order of the county names but the number of cases in each of the counties. Let's address the two aspects one at the time. To change the order on the x-axis we have to re-order the factor levels in `date`.

```
cases2 <- mutate(cases, date= fct_reorder(date, cases, sum,.desc = TRUE)) # re-order the dates based on

ggplot(cases2, aes(y=cases, x=date, fill=county)) +
  geom_bar(position = position_dodge2(), stat = "identity") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))   # this avoids overlapping of x-axis labels (and is
```
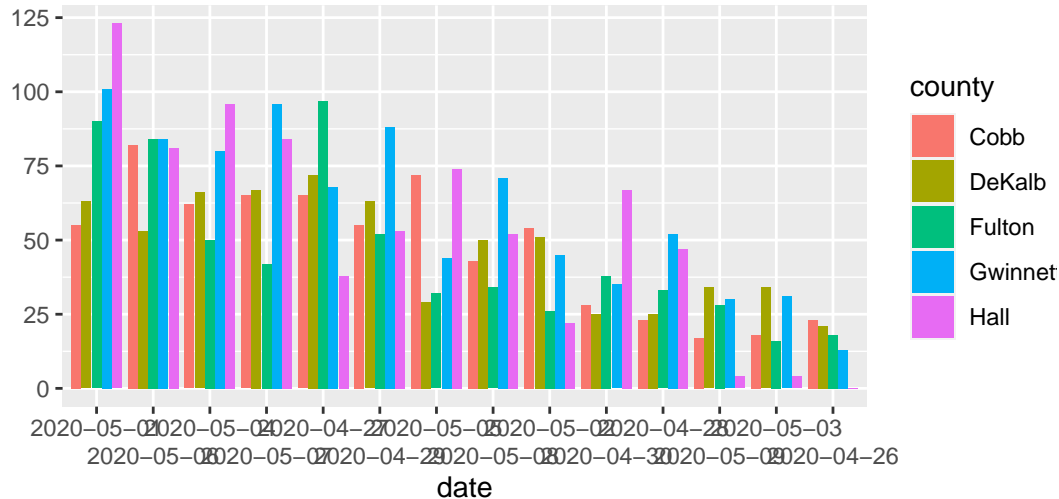

Note that the number of cases are not exactly the same as in the original. Quite likely the reason for this is that the numbers have been updated during May. Given the fact that the overall pattern is very similar, there is no reason to believe that the actual numbers underlying the original figure had been manipulated too. Now, let us address the second aspect (ordering of bars per date). For this, we use the `group` aesthetic, indicating to `ggplot` that we want the number of cases to be used to order the bars within each point in time (date). Setting `position_dodge2(reverse=TRUE)` simply means we want the bars per date to be ordered in decreasing order (per default it would be increasing; try out the difference by changing it to `position_dodge2(reverse=FALSE)` or simply `position_dodge2()`).

```
ggplot(cases2, aes(y=cases, x=date, fill=county, group=cases)) +
  geom_bar(aes(group=cases), position = position_dodge2(reverse=TRUE), stat = "identity") +
    scale_x_discrete(guide = guide_axis(n.dodge = 2))   # this avoids overlapping of x-axis labels (and
```
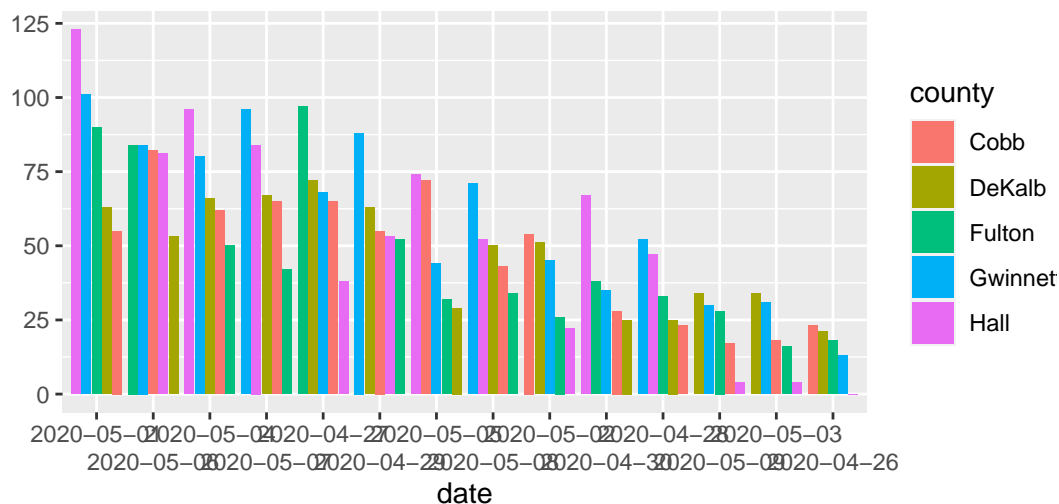

This already looks much more like the original plot.

4.4   Cosmetics: theme

Finally, we tweak the plot's theme in order to look more similar to the original. The following code is a first shot at addressing the most obvious aspects to make the plot more similar to the original. More steps might be needed to make it essentially identical (consider, for example the color-scheme of the bars). This part of the tutorial is a nice illustration of how versatile the `ggplot2-theme()`-function is regarding the tweaking of every detail of a plot.

```r
ggplot(cases2, aes(y=cases, x=date, fill=county, group=cases)) +
  geom_bar(aes(group=cases), position = position_dodge2(reverse=TRUE), stat = "identity") +
  ggtitle("Top5 Counties with the Greatest Number of Confirmed COVID-19 Cases") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +  # this avoids overlapping of x-axis labels (and
  guides(fill=guide_legend(title = "County",
                           title.position = "top",
                           direction = "horizontal",
                           title.hjust = 0.5)) +
  theme(legend.position = "top",
        plot.title = element_text(colour = "white", hjust = -0.1),
        legend.key.size = unit(0.5,"line"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill="#132a52"),
        legend.background = element_rect(fill="#132a52"),
        plot.background = element_rect(fill="#132a52"),
        legend.text = element_text(colour = "#536b8d"),
        legend.title = element_text(colour = "white"),
        axis.text = element_text(colour = "#536b8d"),
        axis.ticks = element_line(colour = "#71869e"),
        axis.line = element_line(colour="#71869e"),
        axis.title = element_blank())
```
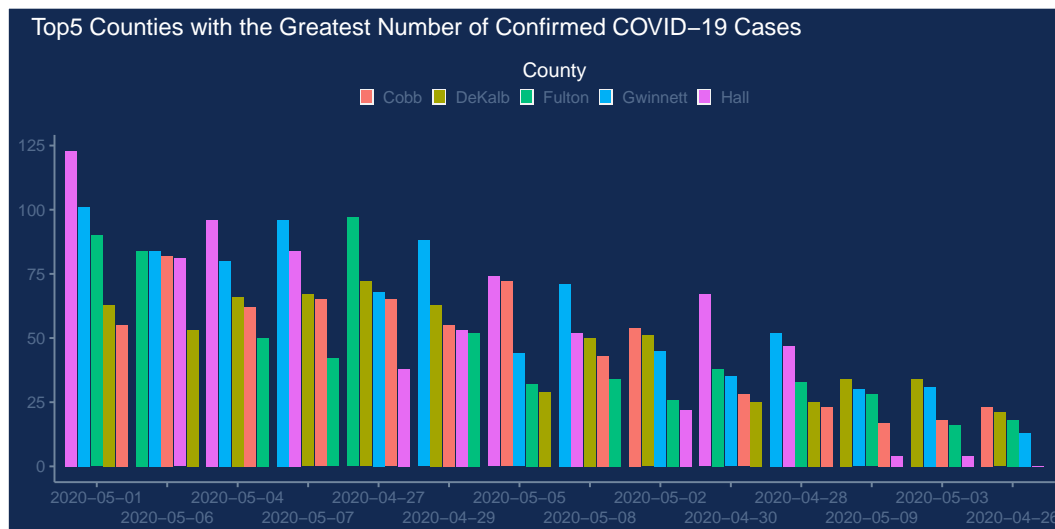

References

R Core Team. 2018. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R*. New York: Springer. http://lmdvr.r-forge.r-project.org.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.