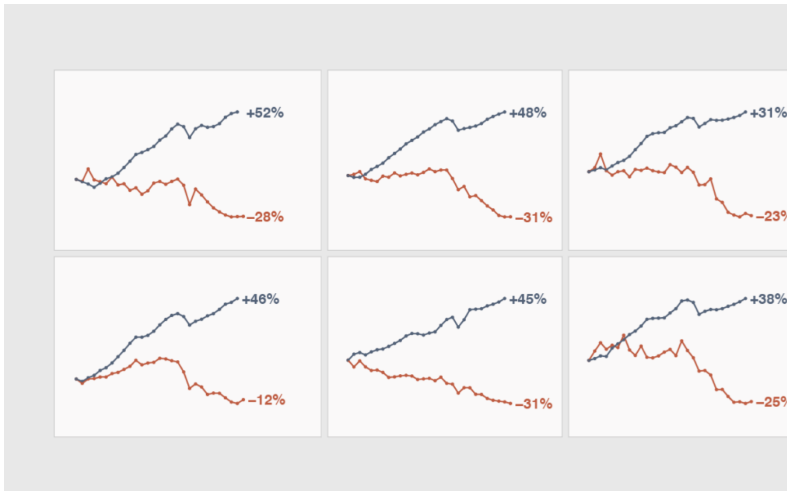
Members Only
Tutorials / ggplot2, R

How to Make Print-ready Graphics in R, with ggplot2

By Maarten Lambrechts

You don't have to use illustration software to polish your graphics. If keeping everything in R is your thing, this tutorial is for you.

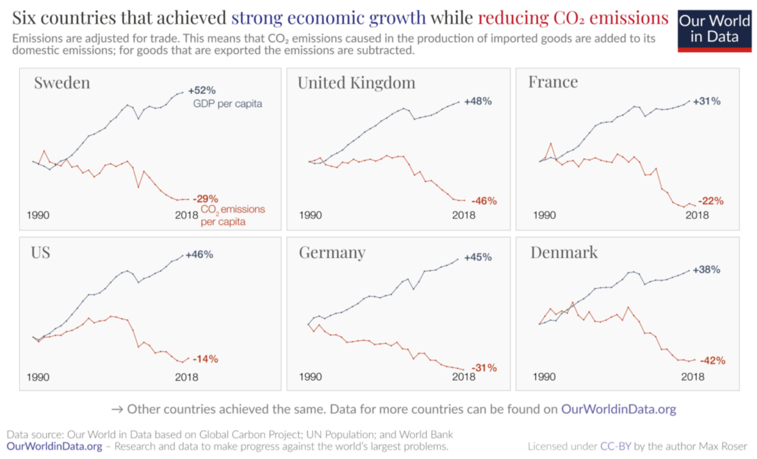
Download Source



ggplot2 provides sensible default settings for analysis, but when you make charts for a publication, you often need to match an existing style and shift focus to readability over exploration. Design around a message or results instead of leaving interpretation open-ended. Finally, you need to export your charts in the required file format with the correct dimensions and resolution.

In this tutorial you remake a chart published by Our World in Data, and learn to style, customize and export your visualization with ggplot2 and some additional packages.

Here is the original:



The charts show the decoupling of GDP and CO₂ emissions for 6 countries, meaning these countries grew their economy but reduced emissions.

Remaking this chart, you'll learn how to:

- Style the title, with a custom font, and apply color to parts of it
- Style the subtitle and wrap it over 2 lines
- Add a logo in the top right corner
- Compose 6 charts in a grid of 2 rows and 3 columns
- Make the first column of charts wider than the 2 others, to accommodate for the directly labelled time series
- Apply a background color to each chart
- Add a styled country name to each chart
- Add data labels to the most recent data points
- Add axis labels for the start and end year of the time series
- Apply custom colors to the lines and data labels
- Add a footer to the visualization
- Export the chart with the right dimensions, and sufficient resolution

This might seem like a long list of little things, but all these small elements add up to a much better chart on the whole.

Setup

You need the following packages:

- dplyr and tidyr to wrangle the data
- ggplot2 for plotting the data
- patchwork to arrange multiple plots together
- ggtext for advanced styling of text
- ragg for working with custom fonts and for marking up text
- magick for combining images, to add the logo

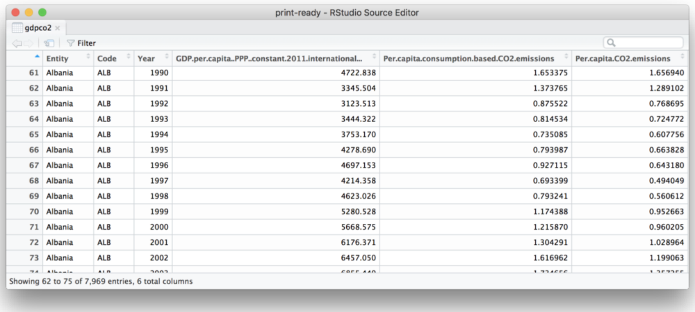
Start by installing and loading the the packages.

```
install.packages(c("dplyr", "tidyr", "ggplot2", "patchwork", "ggtext",
"ragg", "magick"))

library(dplyr)
library(tidyr)
library(ggplot2)
library(patchwork)
library(ggtext)
library(ragg)
library(magick)
```

Data

You can download the data from here, but is also included in the tutorial files. Load it, and inspect it with `View()`.

```
gdpco2 <- read.csv("co2-emissions-and-gdp.csv")
View(gdpco2)
```



	Entity	Code	Year	GDP.per.capita..PPP..constant.2011.international...	Per.capita.consumption.based.CO2.emissions	Per.capita.CO2.emissions
61	Albania	ALB	1990	4722.838	1.653375	1.656940
62	Albania	ALB	1991	3345.504	1.373765	1.289102
63	Albania	ALB	1992	3123.513	0.875522	0.768695
64	Albania	ALB	1993	3444.322	0.814534	0.724772
65	Albania	ALB	1994	3753.170	0.735085	0.607756
66	Albania	ALB	1995	4278.690	0.793987	0.663828
67	Albania	ALB	1996	4697.153	0.927115	0.643180
68	Albania	ALB	1997	4214.358	0.693399	0.494049
69	Albania	ALB	1998	4623.026	0.793241	0.560612
70	Albania	ALB	1999	5280.528	1.174388	0.952663
71	Albania	ALB	2000	5668.575	1.215870	0.960205
72	Albania	ALB	2001	6176.371	1.304291	1.028964
73	Albania	ALB	2002	6457.050	1.616962	1.199063

Showing 62 to 75 of 7,969 entries, 6 total columns

Filter out the data for the 6 countries you need, throw away the *Per.capita.CO2.emissions* column (you'll use the values corrected for import and export in the *Per.capita.consumption.based.CO2.emissions* column), and rename the GDP and CO_2 emissions column for convenience.

```
countries <- c("Sweden", "United Kingdom", "France", "United States",
"Germany", "Denmark")

gdpco2.six <- filter(gdpco2,  Entity %in% countries) %>%
  select(-Per.capita.CO2.emissions) %&>%
  rename(Emissions = Per.capita.consumption.based.CO2.emissions, GDP =
GDP.per.capita..PPP..constant.2011.international...)
```

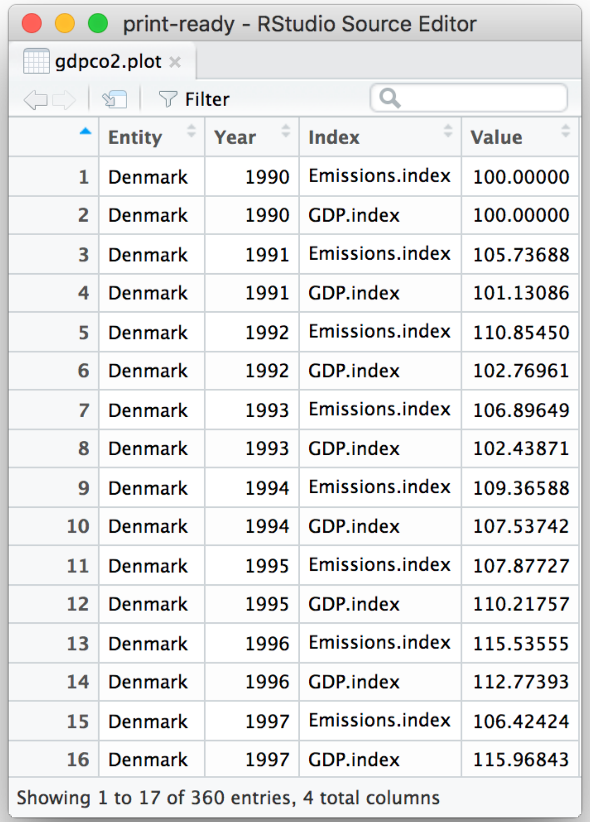
Next, you need to index the values. For both time series, the values are set to 100 in 1990, so you need to divide all values by the values of that year. Create a new data frame with the values from 1990, join it to the original data frame and calculate the indexed values with `mutate()`.

```
gdpco2.90 <- filter(gdpco2.six, Year == 1990) %>%
  select(Entity, Emissions.90 = Emissions, GDP.90 = GDP)

gdpco2.six.index <- left_join(gdpco2.six, gdpco2.90, by = "Entity") %>%
  mutate(Emissions.index = Emissions/Emissions.90*100) %>%
  mutate(GDP.index = GDP/GDP.90*100)
```

The last step in the data preparation is to pivot the data from wide format to the long format required by ggplot. Use `pivot_longer()` of the **tidyr** package for that.

```
gdpco2.plot <- select(gdpco2.six.index, Entity, Year, Emissions.index,
GDP.index) %>%
  pivot_longer(cols=3:4, names_to = "Index", values_to = "Value")
View(gdpco2.plot)
```

The data contains some missing values for 2018 and 2019 at this point. When you plot the data, ggplot will ignore these *NA* values, but it will generate a warning. You can safely ignore the warnings. If you would like to get rid of them, you can filter out the missing values with the *filter()* and *is.na()* functions.



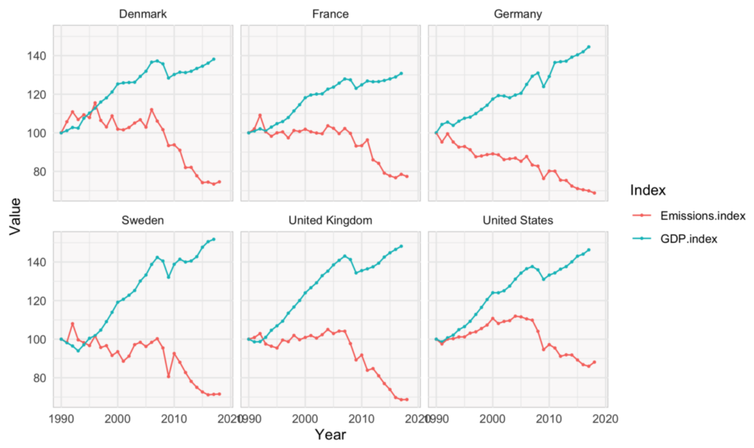
	Entity	Year	Index	Value
1	Denmark	1990	Emissions.index	100.00000
2	Denmark	1990	GDP.index	100.00000
3	Denmark	1991	Emissions.index	105.73688
4	Denmark	1991	GDP.index	101.13086
5	Denmark	1992	Emissions.index	110.85450
6	Denmark	1992	GDP.index	102.76961
7	Denmark	1993	Emissions.index	106.89649
8	Denmark	1993	GDP.index	102.43871
9	Denmark	1994	Emissions.index	109.36588
10	Denmark	1994	GDP.index	107.53742
11	Denmark	1995	Emissions.index	107.87727
12	Denmark	1995	GDP.index	110.21757
13	Denmark	1996	Emissions.index	115.53555
14	Denmark	1996	GDP.index	112.77393
15	Denmark	1997	Emissions.index	106.42424
16	Denmark	1997	GDP.index	115.96843

Showing 1 to 17 of 360 entries, 4 total columns

Plotting

One approach for composing 6 charts in a 2 by 3 grid, is to use ggplot2's native `facet_wrap()` function:

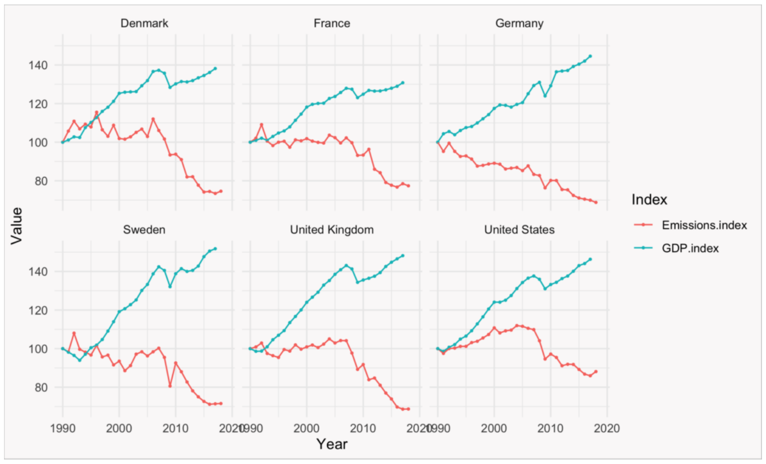
```
ggplot(gdpco2.plot, aes(x = Year, y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal() +
  facet_wrap(~Entity, ncol = 3) +
  theme(panel.background = element_rect(fill = "#FAF9F9", color =
"#CCCCCC", size = 0.5))
```



This is a good start, but 2 features that you need are not supported by ggplot and `facet_wrap()`: background color and column width.

You want some color applied to the chart background. The theming system in ggplot (see further down) offers 2 areas for styling: *plot.background* and *panel.background*. The *panel.background* is the area between the 2 axes (colored in light grey in the plots above) and the *plot.background* is the background of the whole plot. You need *plot.background*, because you want the 1990 and 2018 x-axis labels to be included in the colored area. But when you use `facet_wrap()`, *plot.background* is applied to the whole plot:

```
ggplot(gdpco2.plot, aes(x = Year, y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal() +
  facet_wrap(~Entity, ncol = 3) +
  theme(plot.background = element_rect(fill = "#FAF9F9", color =
"#CCCCCC", size = 0.5))
```



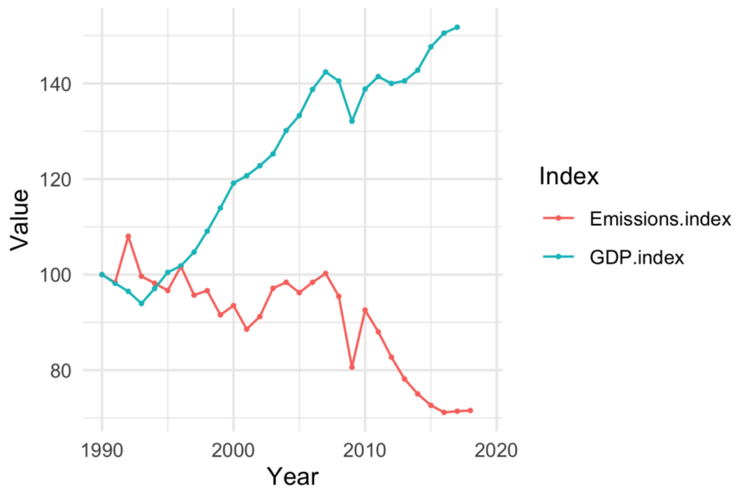
As for column width, you need the first column of charts to be wider than the two others. Column widths are not configurable in `facet_wrap()`.

As always, there are workarounds to both issues. But the **patchwork** package solves both issues in one go. **patchwork**'s goal is to make it easy to compose multiple ggplot plots into a single graphic. It supports the sizing of plots, and because every plot is independent from the others, you can apply some color to the *plot.background* of each chart, without setting the background color of the whole graphic.

A base plot for patchwork

Let's start with a base plot that you can reuse for each of the 6 charts in the patchwork that you are going to create. Because the style of the original charts is quite minimalistic, applying the built-in ggplot `theme_minimal()` is a good start for the theming.

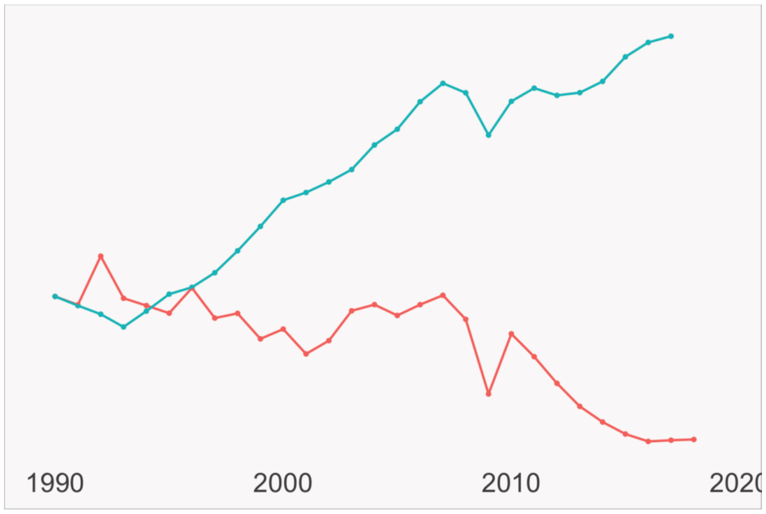
```
baseplot <- ggplot(filter(gdpco2.plot, Entity == "Sweden"), aes(x = Year,
y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal()
baseplot
```



Now customize the theming of the base plot. You can do that with ggplot's `theme()` function. When you want to change the styling of an element in a ggplot plot, you need to know how to address it in the ggplot theming system. You can think of the theming system like a tree, where leaves and smaller branches inherit theming from bigger branches, and bigger branches ultimately inherit from the root or stem.

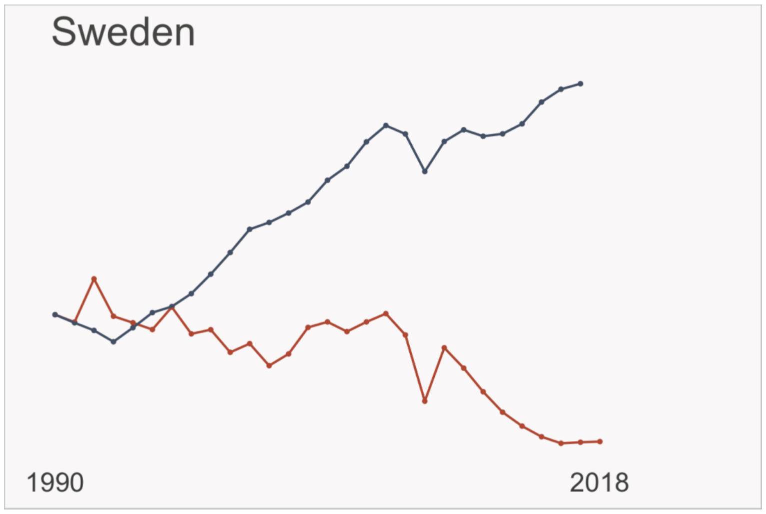
For example, you can address the axis titles of a plot with *axis.title* and set them to `element_blank()` to remove them. Axis labels are addressed with *axis.text*, but if you only want to remove the labels of the y axis, you can set *axis.labels.y* to `element_blank()`. Styling of text is done with `element_text()`, rectangle elements, like the *plot.background* are styled with `element_rect()`. Removing the legend is done by setting *legend.position* to "none".

```
baseplot <- ggplot(filter(gdpco2.plot, Entity == "Sweden"), aes(x = Year,
y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal() +
  theme(
    axis.title = element_blank(),
    axis.text.y = element_blank(),
    axis.text.x = element_text(size = 12),
    panel.grid = element_blank(),
    legend.position = "none",
    plot.background = element_rect(fill = "#FAF9F9", color = "#CCCCCC",
size = 0.5),
    plot.title = element_text(size = 18, color = "#565655", hjust = 0.05),
    )
```



Limit the labels on the x-axis to just 1990 and 2018 with the *breaks* parameter of `scale_x_continuous()`, and apply custom colors with `scale_color_manual()`. Let's also add a title with `ggtitle()` and add some styling for it with *plot.title* in the `theme()` function.

```
baseplot <- ggplot(filter(gdpco2.plot, Entity == "Sweden"), aes(x = Year,
y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal() +
  theme(
    axis.title = element_blank(),
    axis.text.y = element_blank(),
    axis.text.x = element_text(size = 12),
    panel.grid = element_blank(),
    legend.position = "none",
    plot.background = element_rect(fill = "#FAF9F9", color = "#CCCCCC",
size = 0.5),
    plot.title = element_text(size = 18, color = "#565655", hjust = 0.05),
    ) +
  scale_x_continuous(breaks = c(1990, 2018), limits = c(1990, 2024)) +
  scale_color_manual(values = c("#BE5B40", "#526177"))
baseplot + ggtitle("Sweden")
```

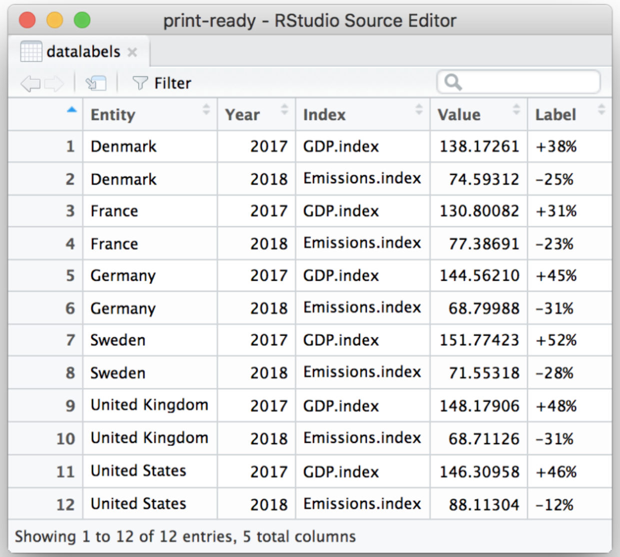


Data labels

The data labels, indicating the percent change between 1990 and the most recent data, are still missing.

The following snippet filters out the most recent data points (GDP from 2017, emissions from 2018), and creates a new column with the percentage change, with "%" appended, and a "+" prepended for the positive values of the GDP growth.

```
datalabels <- filter(gdpco2.plot, (Year == 2017 & Index == "GDP.index") |
(Year == 2018 & Index == "Emissions.index")) %>%
  mutate(Label = ifelse(Value - 100 < 0,
                        paste(round(Value - 100), "%", sep = ""),
                        paste("+", round(Value - 100), "%", sep = "")))

View(datalabels)
```

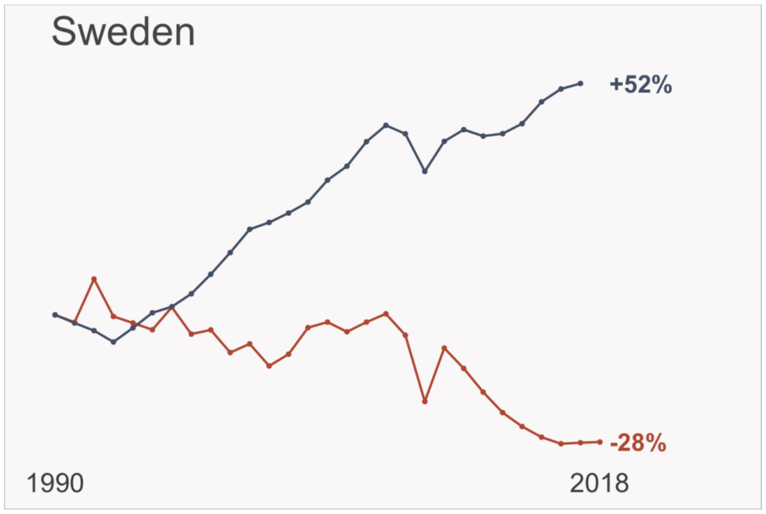


	Entity	Year	Index	Value	Label
1	Denmark	2017	GDP.index	138.17261	+38%
2	Denmark	2018	Emissions.index	74.59312	-25%
3	France	2017	GDP.index	130.80082	+31%
4	France	2018	Emissions.index	77.38691	-23%
5	Germany	2017	GDP.index	144.56210	+45%
6	Germany	2018	Emissions.index	68.79988	-31%
7	Sweden	2017	GDP.index	151.77423	+52%
8	Sweden	2018	Emissions.index	71.55318	-28%
9	United Kingdom	2017	GDP.index	148.17906	+48%
10	United Kingdom	2018	Emissions.index	68.71126	-31%
11	United States	2017	GDP.index	146.30958	+46%
12	United States	2018	Emissions.index	88.11304	-12%

Showing 1 to 12 of 12 entries, 5 total columns

Add the data labels with a `geom_text()` layer with the percentage change.

```
baseplot +
  ggtitle("Sweden") +
  geom_text(data = filter(datalabels, Entity == "Sweden"), aes(label =
Label, x = 2018.5), hjust = 0, fontface = "bold")
```

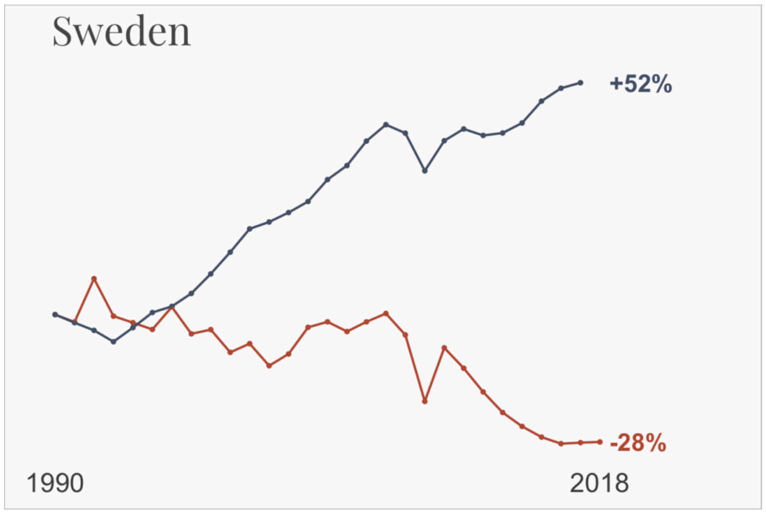


The base plot is ready, accept for 1 detail: you still need to customize the font of the title. Working with fonts outside the most common families (Times New Roman, Arial and Courier) in R used to be non-trivial. Luckily some packages are addressing this problem. A relatively new addition to the R packages working with fonts is the **ragg** package, which makes working with custom fonts a lot easier. With **ragg** installed, you get support for

- all your installed system fonts
- right-to-left and non-Roman scripts
- emoji's in text

The original Our World in Data visualization uses the Playfair Display font (freely available from Google Fonts). After installing **ragg**, using a custom font in your plots is as easy as installing the font on your system, and use it in your code.

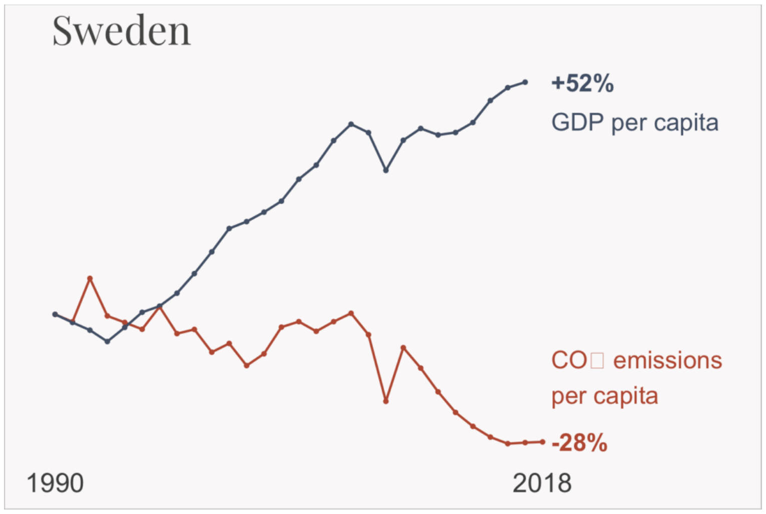
```
baseplot <- ggplot(filter(gdpco2.plot, Entity == "Sweden"), aes(x = Year,
y = Value, color = Index)) +
  geom_point(size = 0.5)  +
  geom_line() +
  theme_minimal() +
  theme(
    axis.title = element_blank(),
    axis.text.y = element_blank(),
    axis.text.x = element_text(size = 12),
    panel.grid = element_blank(),
    legend.position = "none",
    plot.background = element_rect(fill = "#FAF9F9", color = "#CCCCCC",
size = 0.5),
    plot.title = element_text(family = "Playfair Display", size = 18,
color = "#565655", hjust = 0.05)
    ) +
  scale_x_continuous(breaks = c(1990, 2018), limits = c(1990, 2024)) +
  scale_color_manual(values = c("#BE5B40", "#526177"))

baseplot +
  ggtitle("Sweden") +
  geom_text(data = filter(datalabels, Entity == "Sweden"), aes(label =
Label, x = 2018.5), hjust = 0, fontface = "bold")
```



Annotate with Rich Text

Now the base plot is ready, but in the original, the plot for Sweden contains an additional annotation layer, with the direct labels for both time series (“GDP per capita” and “CO_2 emissions per capita”). You can add these with ggplots `annotate()` function. To make the text fit, we extend the x scale a bit more to the right. Finding the right x and y position for the annotations requires a bit of trial and error.

```
sweden <- baseplot %+% filter(gdpco2.plot, Entity == "Sweden") +
  geom_text(data = filter(rates.plot, Entity == "Sweden"), aes(label =
Label, x = 2018.5), hjust = 0, fontface = "bold") +
  ggtitle('Sweden') +
  theme(plot.title = element_text(family = "Playfair Display", size = 18,
color = "#565655", hjust = 0.05)) +
  annotate("text", x = 2018.5, y = 143, label = "GDP per capita", hjust =
0, color = "#526177") +
  annotate("text", x = 2018.5, y = 86, label = "CO₂ emissions\nper
capita", hjust = 0, fill = NA, label.color = NA, color = "#BE5B40")  +
  scale_x_continuous(breaks = c(1990, 2018), limits = c(1990, 2028))
sweden
```

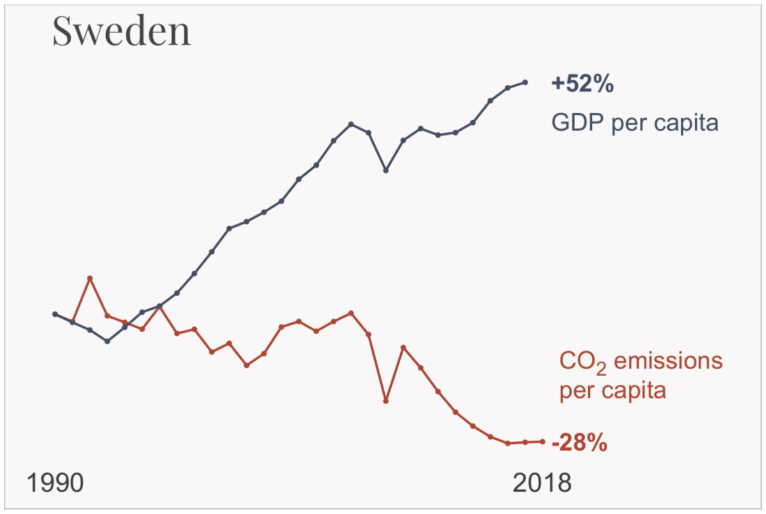


The annotation for the emissions is placed above the data label, instead of below it like in the original (in the original the annotation feels a bit cramped together with the data label and the “2018” label of the x-axis). The annotion contains a line break, which you can insert with “\n”.

But notice that the subscript “2” in “CO_2” is missing. One option to add it, is to set the *parse* parameter to TRUE in `annotate()`, after which you can use the plotmath syntax for mathematical annotations (CO_2 would be written as CO[2]). But plotmath was designed for mathematical formulas, not so much for text. As a result, spaces and line breaks are not straightforward in plotmath.

This is where the **ggtext** package comes in handy. It allows you to use a subset of html in ggplot text elements. After loading the package, you should set the first argument of the annotate function to “richtext” instead of “text”, so the annotation uses `geom_richtext()` from the ggtext package instead of ggplots `geom_text()`. Then you can write subscript by using html *sub* tags, and *br* tags for line breaks.

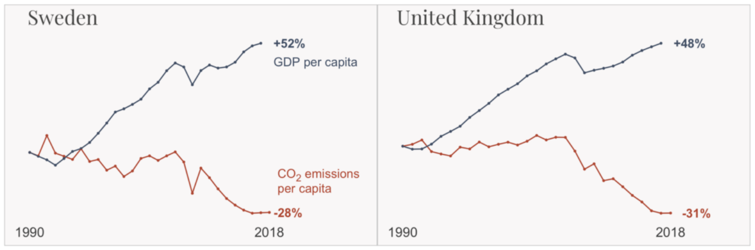
```
sweden <- baseplot %+% filter(gdpco2.plot, Entity == "Sweden") +
  geom_text(data = filter(rates.plot, Entity == "Sweden"), aes(label =
Label, x = 2018.5), hjust = 0, fontface = "bold") +
  ggtitle('Sweden') +
  theme(plot.title = element_text(family = "Playfair Display", size = 18,
color = "#565655", hjust = 0.05)) +
  annotate("text", x = 2018.5, y = 143, label = "GDP per capita", hjust =
0, color = "#526177") +
  annotate("richtext", x = 2018.5, y = 86, label = "CO<sub>2</sub>
emissions<br />per capita", hjust = 0, fill = NA, label.color = NA, color
= "#BE5B40") +
  scale_x_continuous(breaks = c(1990, 2018), limits = c(1990, 2028))
sweden
```



Patchwork

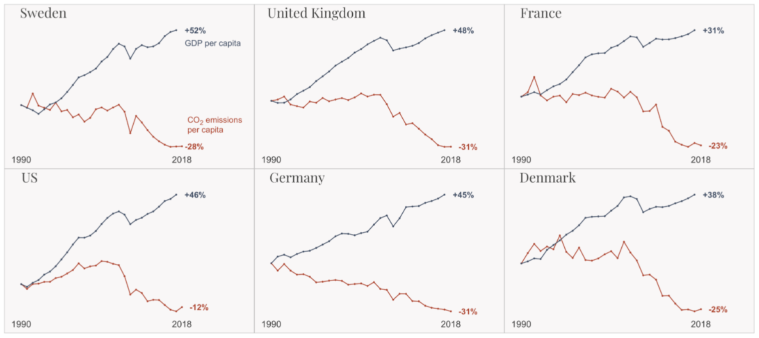
With that out of the way, it is time to compose the patchwork of charts. Composing 2 plots together is as simple as adding them together. Make a second plot with the baseplot and the data for the UK (you can use the %+% operator to update the data of a ggplot), and add it to the plot for Sweden:

```
uk <- baseplot %+% filter(gdpco2.plot, Entity == "United Kingdom") +
  geom_text(data = filter(datalabels, Entity == "United Kingdom"),
aes(label = Label), hjust = -0.4, fontface = "bold") +
  ggtitle('United Kingdom')

sweden + uk
```



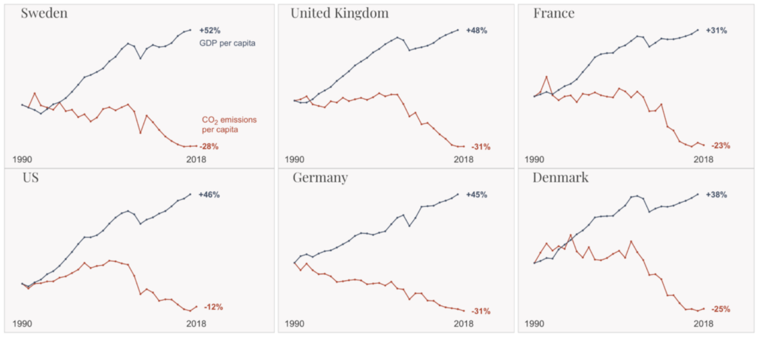
Rows in patchwork are separated with a slash (the plots for each country are generated in the same way as the one for the United Kingdom above):

```
(sweden + uk + france) /
(us + germany + denmark)
```



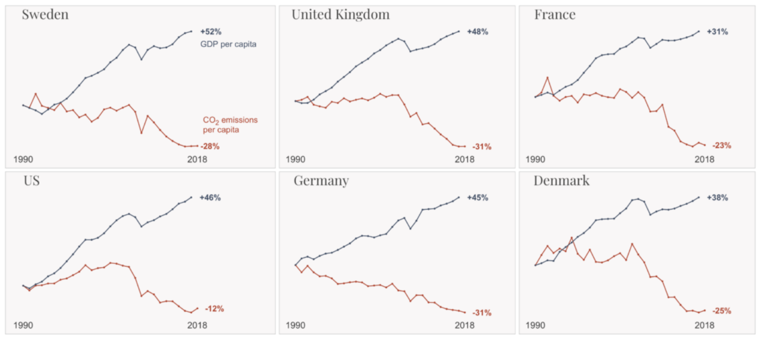
When your plots have no background color and no border, you could add spacing between the plots by setting the *plot.margin* in their `theme()` function. But here, you have to use a different approach. The `plot_spacer()` function adds an empty plot, of which you can set the *plot.margin* by adding a theme function. Than you can let it come together by adding a `plot_layout()` function that specifies the relative widths of each column.

```
(sweden +
    plot_spacer()  + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
    uk +
    plot_spacer() + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
    france  + plot_layout(widths = c(46,0.1,40,0.1,40))) /
  (us +
     plot_spacer()  + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
     germany +
     plot_spacer() + theme(plot.margin = unit(c(0,2,0,2), "pt"))+
     denmark  + plot_layout(widths = c(46,0.1,40,0.1,40)))
```



The same procedure can be used to add vertical spacing between the 2 rows of charts: add an additional row with an empty plot:

```
## Row 1
patchwork.all <- (sweden +
    plot_spacer()  + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
    uk +
    plot_spacer() + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
    france +
    plot_layout(widths = c(46,0.1,40,0.1,40))) /
  ## Row 2 (spacing)
  (plot_spacer() + theme(plot.margin = unit(c(2,0,2,0), "pt"))) /
  ## Row 3
  (us +
     plot_spacer() + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
     germany +
     plot_spacer() + theme(plot.margin = unit(c(0,2,0,2), "pt")) +
     denmark +
     plot_layout(widths = c(46,0.1,40,0.1,40))) +
  ## layout for the whole plot
plot_layout(heights = c(40,0.1,40))
patchwork.all
```

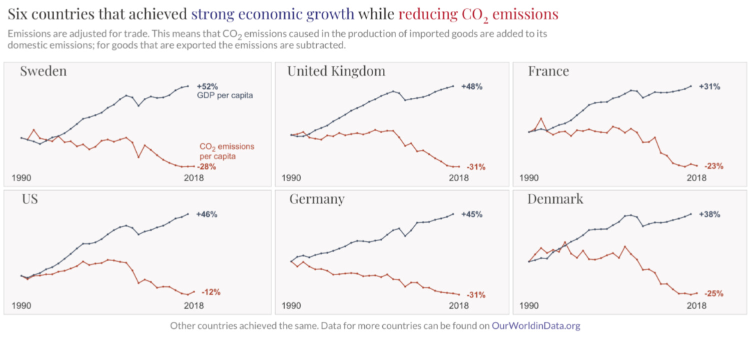


Title and subtitle

Just like with a normal ggplot plot, patchwork allows you to add a title, subtitle and caption, and style them. Adding these text elements is done by adding a `plot_annotation()` function.

You've used `geom_richtext()` of the ggtext package earlier, but ggtext offers more. Normally, text in a ggplot is styled with `element_text()` in the `theme()` function. But if you use ggtext's `element_markdown()` instead of `element_text()`, you can not only write markdown in the text, but also add html markup and styling. So you can change the color of parts of the title by wrapping those parts in *span* elements and set some styling on them. You also need this to add the subscript "2" in CO_2 in the title and subtitle.

```
patchwork.all + plot_annotation(
    title = "Six countries that achieved <span style='color:
#424590;'>strong economic growth</span> while <span style='color:
#C5303B;'>reducing CO<sub>2</sub> emissions</span>",
    subtitle = "Emissions are adjusted for trade. This means that
CO<sub>2</sub> emissions caused in the production of imported goods are
added to its<br/>domestic emissions; for goods that are exported the
emissions are subtracted.",
    caption = "Other countries achieved the same. Data for more countries
can be found on <span style='color: #726FAE;'>OurWorldinData.org</span>",
    theme = theme(
      plot.title = element_markdown(family = "Playfair Display", size =
20, color = "#413E3C"),
      plot.subtitle = element_markdown(family = "Lato", color = "#949393",
size = 13.5, margin = margin(t = 0, r = 0, b = 16, l = 0, unit = "pt")),
      plot.caption = element_markdown(family = "Lato", color = "#949393",
size = 13, hjust = 0.5, margin = margin(t = 10, r = 0, b = 0, l = 0, unit
= "pt")),
      plot.margin = margin(t = 10, r = 10, b = 10, l = 10, unit = "pt")))
```



Saving

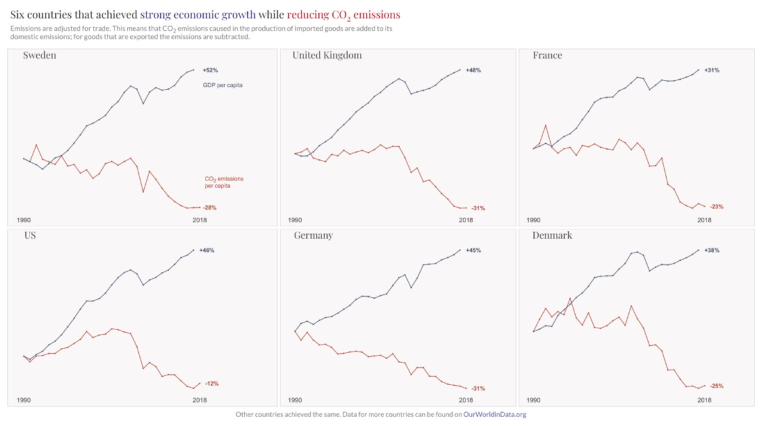
Apart from the logo in the top right corner, the visualization is ready. You are going to add the logo by merging the image file with the visualization together with the logo image, which means that you need to save the visualization image first.

You can save a plot (a single ggplot, or a patchwork of multiple plots) with the `ggsave()` function. The most important parameters of the ggsave function are

- *filename*: where to save the plot to. The file extension determines the file format (png, jpg, …)
- *plot*: the plot to save. Defaults to the last generated plot
- *units*: units for the width and height parameters. Values can be "in" (inches, the default), "cm" or "mm"
- *width* and *height*: the width and height of the saved image, in the specified *units*
- *dpi*: the resolution, in dots per inch

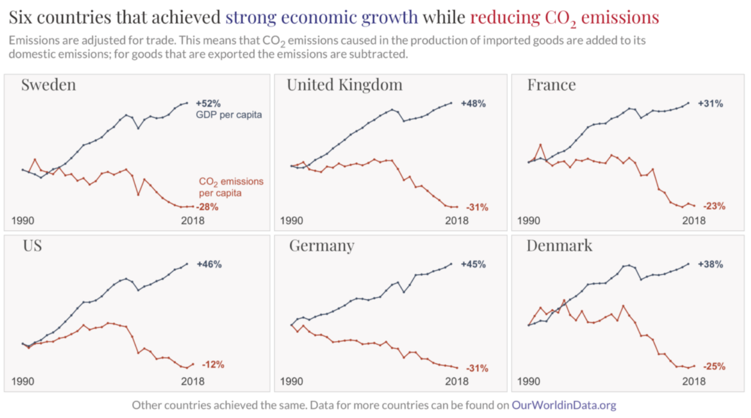
The following code generates a 6000 by 3300 pixel image. Can you see why?

```
ggsave(filename = "ourworldindata_noscale.png", width = 20, height = 11,
dpi = 300)
```



The resolution of the resulting image is a bit too high (the image size is 1 MB), and the font size is a bit too small compared to the other elements. You can use the *scale* parameter to scale the pixel dimensions. This will also scale the size of text elements relative to the image size: a *scale* < 1 will reduce the image size (in pixels) and "blow up" the size of the text, a *scale* > 1 will do the reverse

```
ggsave(filename = "ourworldindata_scaled.png", width = 20, height = 11,
dpi = 300, scale = 0.6)
```



Adding the logo

The only thing left to do, is to add the logo in the top right corner. There are ways to do add images directly in **ggplot** and **patchwork**, but the following method, using some functions of the **magick** package, is very generic and flexible.

First load the chart and logo image files with `image_read()`.

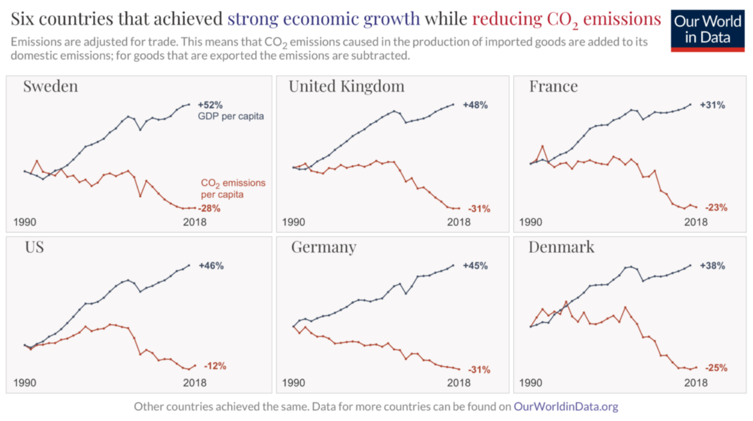
```
chart <- image_read("ourworldindata_scaled.png")
logo <- image_read("OurWorldinData-logo.png")
```

Next, scale the logo image, as the image is too large to fit in the top right corner. The second argument of the `image_scale()` function, "x200", scales the image 200 times down ("200x" would scale it 200 times up).

```
logo.scaled <- image_scale(logo, "x200")
```

Then you can combine both images with `image_composite()`, and offsetting the scaled logo 3,200 pixels in the horizontal direction, and 30 pixels in the vertical direction. Then save the resulting image with `image_write()`.

```
chart.logo <- image_composite(chart, logo.scaled, offset = "+3200+30")
image_write(chart.logo, path = "OWDcharts-logo.png", format = "png")
```



Wrapping up

By remaking a chart, you learned how to use the ggplot theming system to customize the styling of your visualizations, how to compose multiple plots together with patchwork, how to spice up text with markup and styling with ggtext, how to use custom fonts and how to add a logo and save your visualization.

This is already a big step towards making print-ready graphics with ggplot. But there is a lot more to say about this topic. If you want to dig a little deeper, here are some links to relevant packages, documentation and tutorials:

- the anatomy of a ggplot chart, and the ggplot theming system
- ggplot built-in themes and developing a custom ggplot theme
- working with fonts
- working with colors
- manual and automated labelling
- annotating plots
- adding graphical effects

About the Author

Maarten is a data journalist and data visualization consultant from Belgium. He likes maps, ggplot and a good story.

2 Comments (Add Yours)

- *Eulalia Claros* — November 30, 2021 at 8:29 am

  Big thank you to both of you, Nathan and Martin! you work wonders on the otherwise cumbersome details of the 'printing quality'stage!!

  It is so very useful to have this as reference, with all the relevant links for further digging..!

  Reply
  - *Nathan Yau* — December 1, 2021 at 10:02 am

    Eulalia, glad you found it helpful!

    Reply

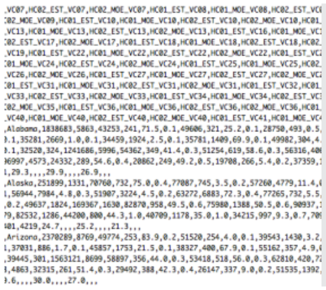
Add Comment

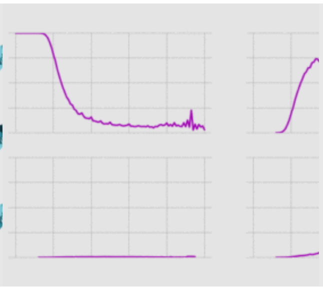
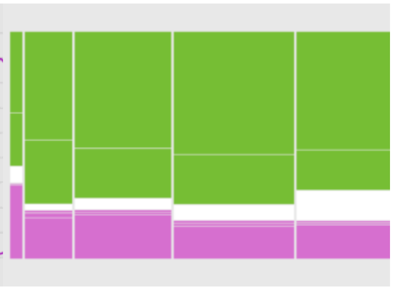
Logged in as cvidonne. Logout »

Comment

Submit Query

More Tutorials See All →



Loading Data and Basic Formatting **Small Maps and Grids** **How to Make Animated Line** **How to Make a Mosaic Plot in R**

in R

It might not be sexy, but you have to load your data and get it in the right format before you can visualize it. Here are the basics, which might be all you need.

Maybe you want to make spatial comparisons over time or across categories. Organized small maps might do the trick.

Charts in R

Sometimes it's useful to animate the multiple lines instead of showing them all at once.

Also known as a Marimekko diagram, the mosaic plot lets you compare multiple qualitative variables at once. They can be useful, sometimes.

- About
- Contact
- Twitter
- Newsletter
- RSS