

Tutorial for The R Graph Gallery

Yayun Luo & Yu Cheng

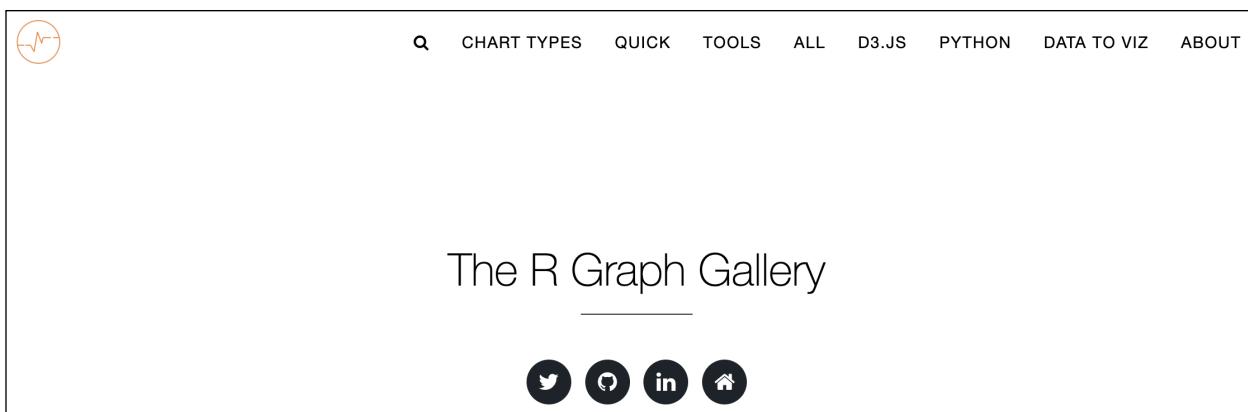
03/28/2022

• Introduction of The R Graph Gallery

The R graph gallery is a collection of charts made with the R programming language. The gallery provides hundreds of charts ,which made by packages ‘tidyverse’ and ‘ggplot2’. It is easy to make your own chart by using the reproducible code provided by the gallery. The R Graph Gallery is a good tool for the R programming beginners to make their own charts.

• Step 1: Go to the Website and Get Started

Here is the website: <https://r-graph-gallery.com/>



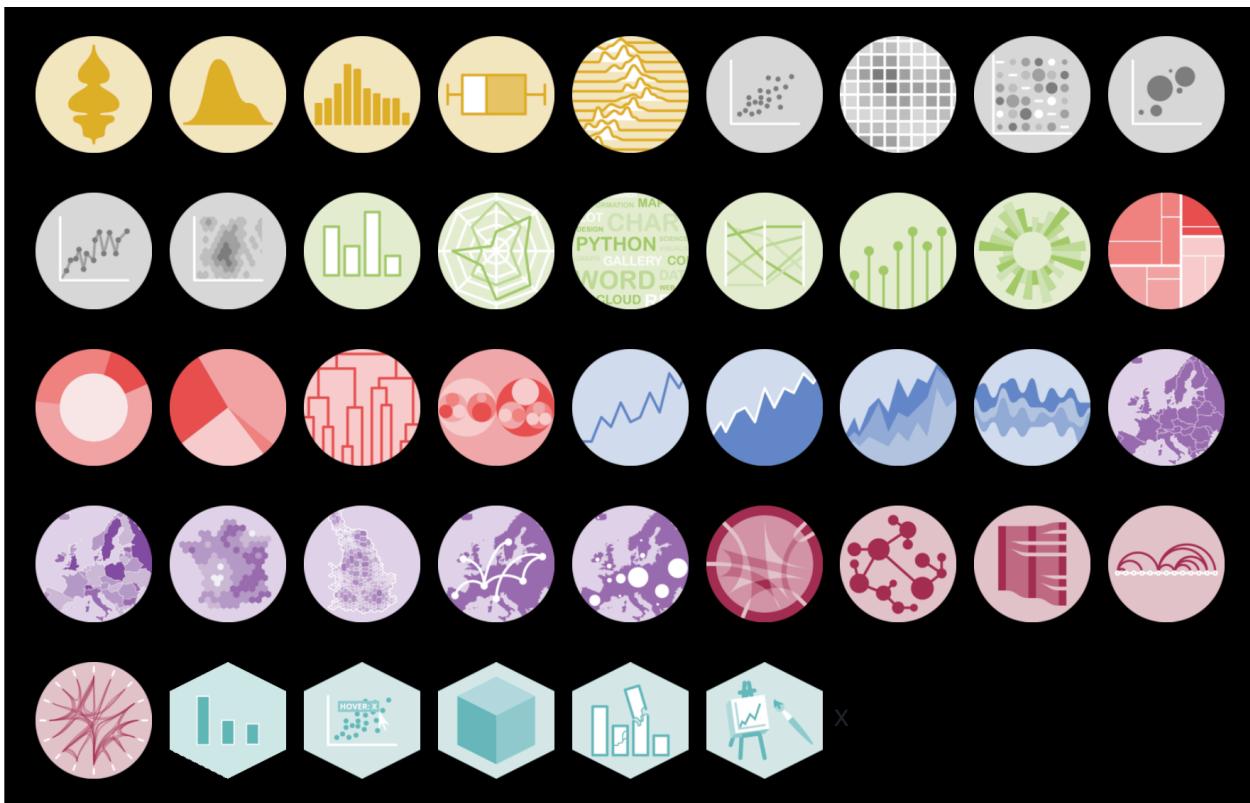
This is how the gallery looks like.

• Step 2: Find the Type of Chart You Need in the Gallery

Click “CHART TYPES”.



Then you need to choose the type you want from dozens of different chart types provided by the gallery.



For example, we choose “Histogram”.

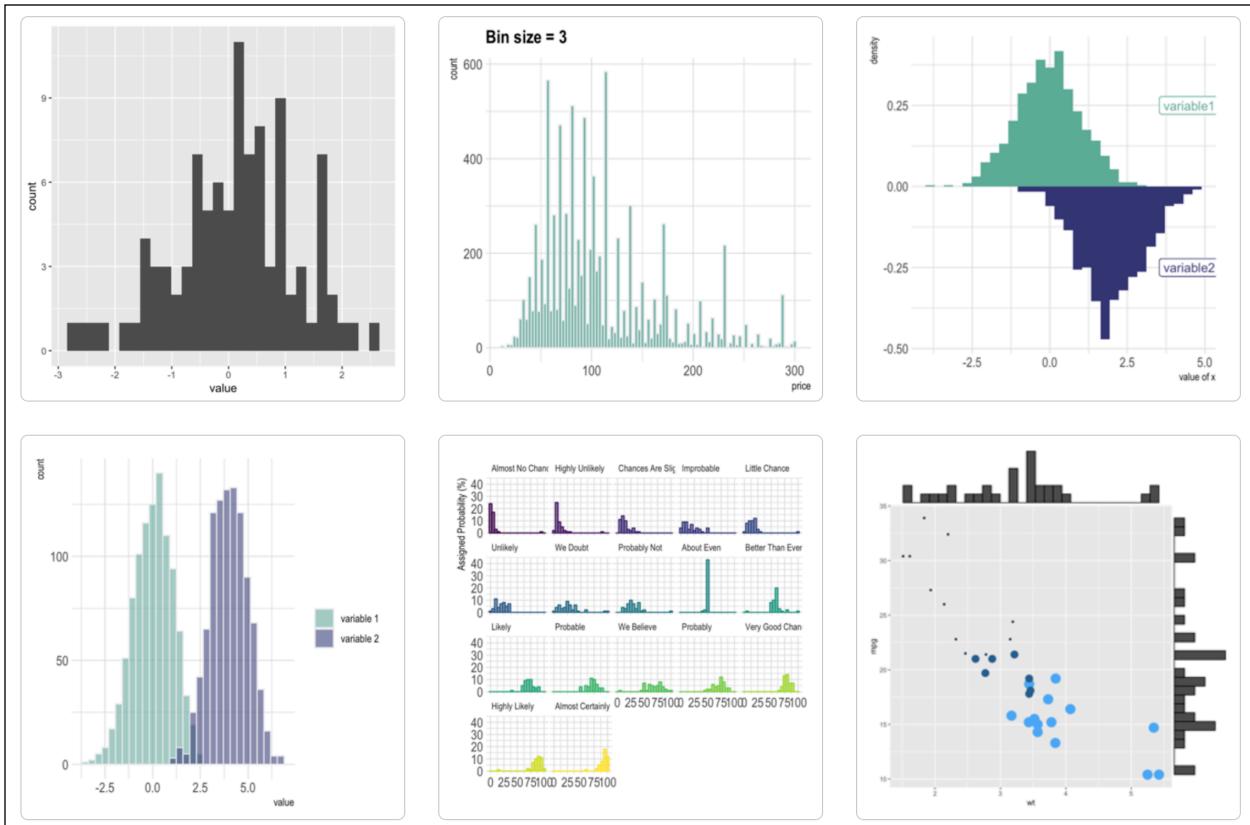
Then, you can see the introduction of the type of chart (Histogram) in the gallery.

Histogram

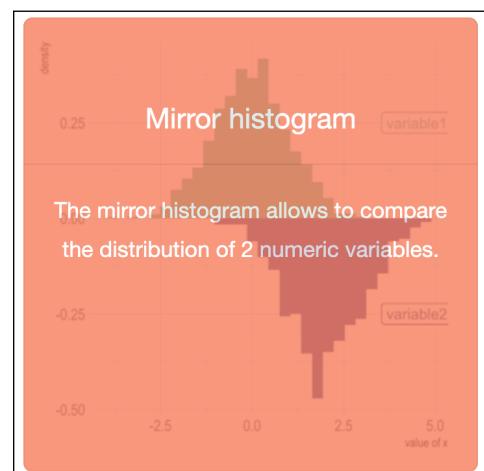


Welcome to the [histogram](#) section of the R graph gallery. A histogram is used to study the distribution of one or several variables, as explained in [data-to-viz.com](#). If you're looking for a simple way to implement it in R, pick an example below. This page focuses on [ggplot2](#) but base R examples are also provided. This [online course](#) can be a great addition to know more about [ggplot2](#).

The histogram can be built via the packages `ggplot2` and `tidyverse`, or it can be built just by base R. Codes for both way to build histograms are provided by the gallery.



Here are some kinds of histograms you can find in the gallery. To make your own histograms, you should just pick one you like and the codes will be there for you.



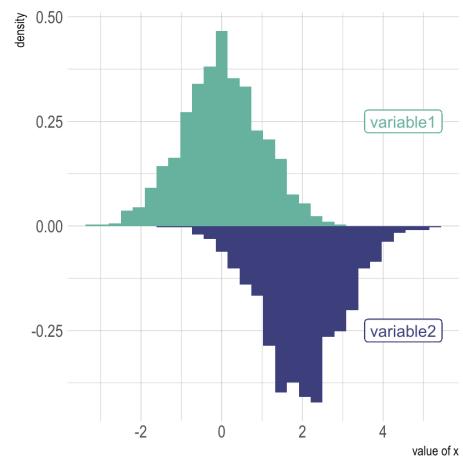
For example, we choose “Mirror Histogram”:

Histogram with `geom_histogram`

Of course it is possible to apply exactly the same technique using `geom_histogram` instead of `geom_density` to get a mirror histogram:

```
# Chart
p <- ggplot(data, aes(x=x)) +
  geom_histogram(aes(x = var1, y = ..density..), fill="#69b3a2") +
  geom_label(aes(x=4.5, y=0.25, label="variable1"), color="#69b3a2") +
  geom_histogram(aes(x = var2, y = -..density..), fill="#404080") +
  geom_label(aes(x=4.5, y=-0.25, label="variable2"), color="#404080") +
  theme_ipsum() +
  xlab("value of x")

#p
```



Then, you will find the codes for the Mirror histogram in the gallery. You just need to change the name of dataset, and the names of variables in order to build your own mirror histograms.

• Other Functions of the Gallery

← the R Graph Gallery Q CHART TYPES **QUICK** TOOLS ALL D3.JS PYTHON DATA TO VIZ ABOUT

ggplot2

- AXIS
- COLOR
- THEMES
- LEGEND
- ANNOTATION
- FACETING
- BACKGROUND
- RE-ORDERING
- FONTS

base r

- SHAPE PARAMETERS
- MARGIN
- COLOR

the R Graph Gallery Q CHART TYPES **QUICK** TOOLS ALL D3.JS **PYTHON** DATA TO VIZ ABOUT

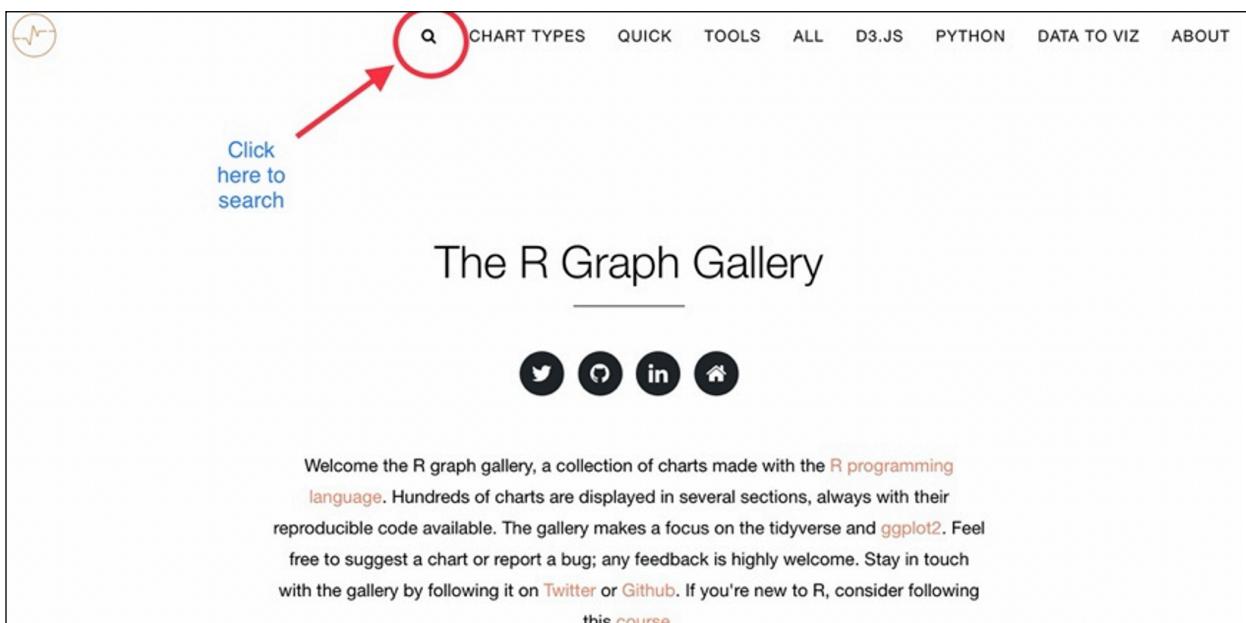
The gallery also provides PYTHON codes.

• A Specific Example of Specific Uses of the R Graph Gallery

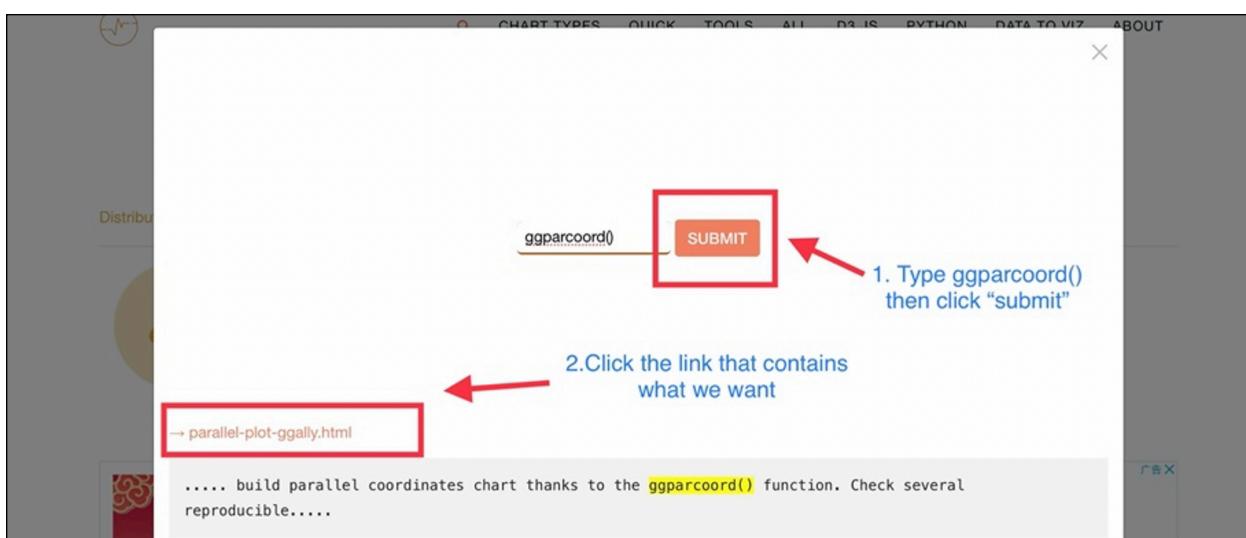
In problem set 3, we have one question is:

Create a static parallel coordinates plot using `GGally::ggparcoord()` showing state death rate patterns for these eight codes for all fifty states. Your graph should have one line for each state and the codes should be the axes of the graph. Include a conversation chart below displaying the cause of death for each of the eight Cause of death Code values since they're too long to fit in the plot. Apply alpha blending, splines, and rescaling to create the clearest version of the plot that you can. Discuss outliers, clusters, and correlations in detail.

This question tells us that we need to use the code **`ggparcoord()`** to showing state death rate patterns for these eight codes for all fifty states. Then we can search for **`ggparcoord()`** in R Graph Gallery website:



When we find the link containing the desired keyword, click on it.



Then we will get this page:

← the R Graph Gallery

CHART TYPES QUICK TOOLS ALL D3.JS PYTHON DATA TO VIZ ABOUT

Parallel coordinates chart with ggally

[Twitter](#) [GitHub](#) [LinkedIn](#) [Home](#)

ggally is a `ggplot2` extension. It allows to build parallel coordinates charts thanks to the `ggbparcoord()` function. Check several reproducible examples in this post.

[PARALLEL COORD SECTION](#) [ABOUT PARALLEL COORD. CHARTS](#)

If we click on the option on the left:

Parallel coordinates chart with ggally

[Twitter](#) [GitHub](#) [LinkedIn](#) [Home](#)

ggally is a `ggplot2` extension. It allows to build parallel coordinates charts thanks to the `ggbparcoord()` function. Check several reproducible examples in this post.

[PARALLEL COORD SECTION](#) [ABOUT PARALLEL COORD. CHARTS](#)

Most basic

Click it

This is the most basic parallel coordinates chart you can build with `R`, the `ggally` packages and its `ggbparcoord()` function.

The input dataset must be a data frame with several numeric variables, each being used as a vertical axis on the chart. Columns number of these variables are specified in the `columns` argument of the function.

Species

- setosa
- versicolor
- virginica

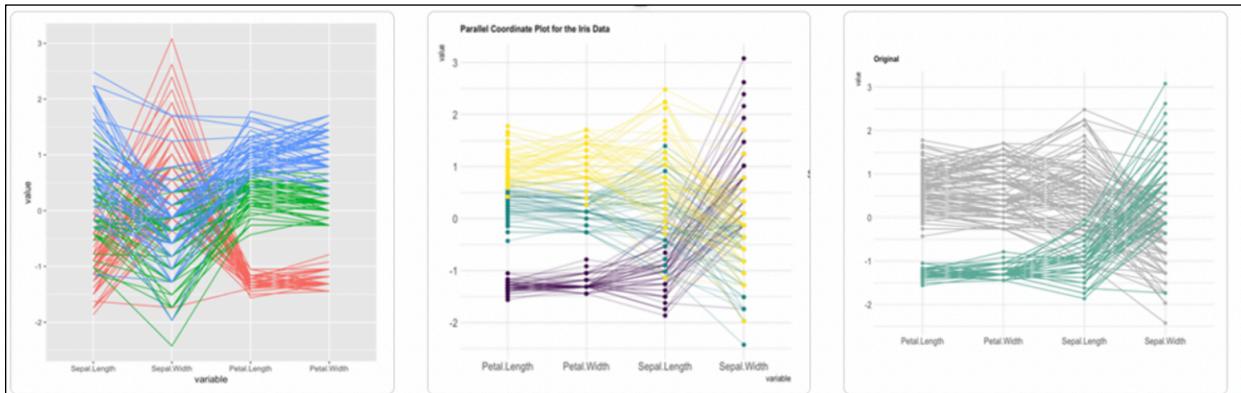
Then we can see many different styles of parallel coordinates charts.

Parallel Coordinates chart



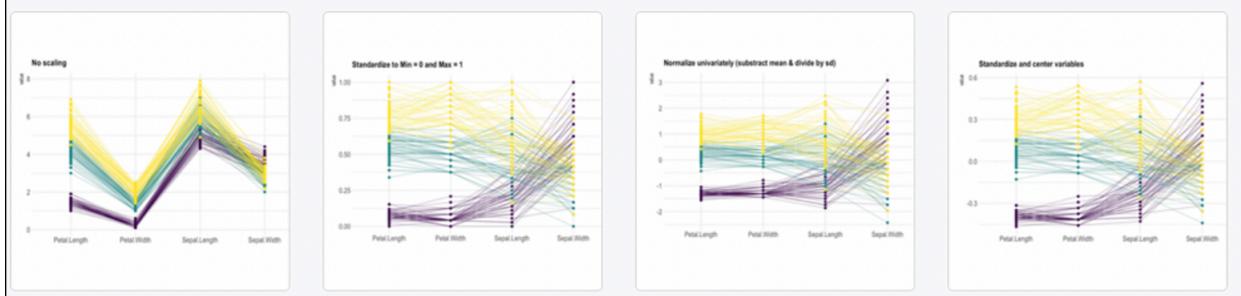
This is the [Parallel Coordinates chart](#) section of the gallery. If you want to know more about this kind of chart, visit [data-to-viz.com](#). If you're looking for a simple way to implement it in d3.js, pick an example below.

STEP BY STEP - THE [GGALLY](#) LIBRARY



A NOTE ON SCALING

Scaling variable is a crucial step to build a proper parallel coordinates chart. It transforms the raw data to a new scale that is common with other variables, and thus allow to compare them. The [ggally](#) package offers a `scale` option that computes the most common types of scaling:



If we click on the option on the right:

Parallel coordinates chart with ggally



`ggally` is a `ggplot2` extension. It allows to build parallel coordinates charts thanks to the `ggparcoord()` function. Check several reproducible examples in this post.

PARALLEL COORD SECTION

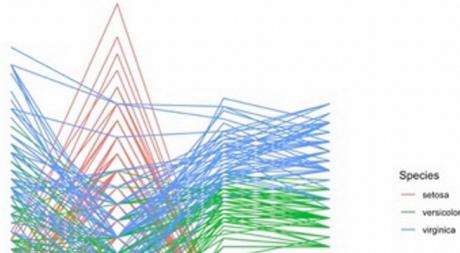
ABOUT PARALLEL COORD. CHARTS

Most basic

Click it

This is the most basic parallel coordinates chart you can build with `R`, the `ggally` packages and its `ggparcoord()` function.

The input dataset must be a data frame with several numeric variables, each being used as a vertical axis on the chart. Columns number of these variables are specified in the `columns` argument of the function.

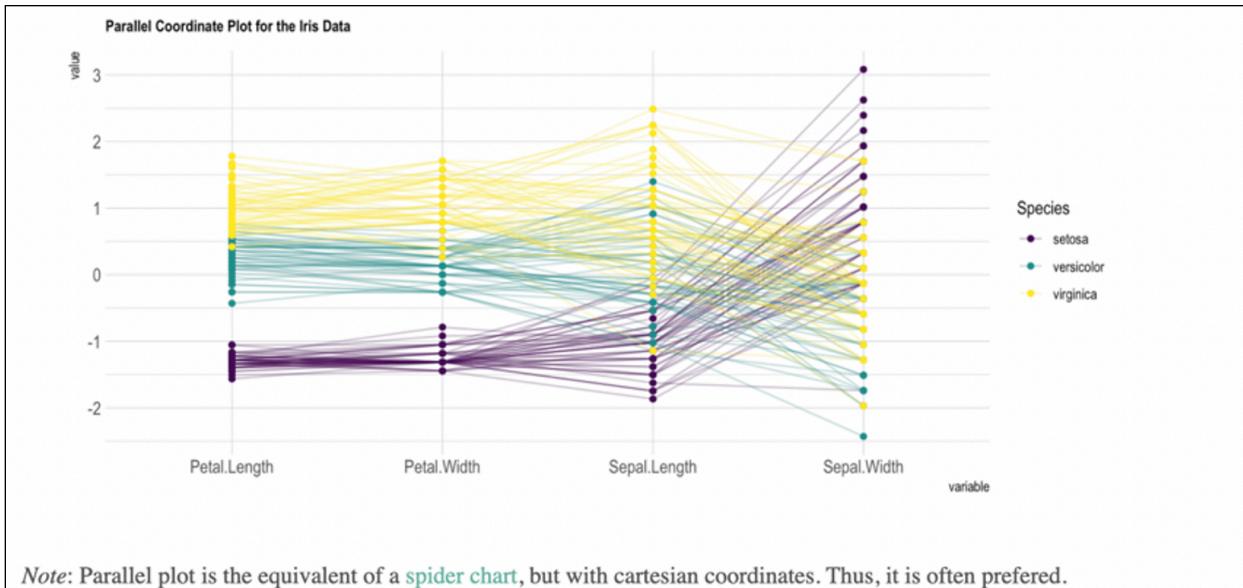


Then we will get a detailed description of parallel coordinates charts including definitions, roles, variation, common mistakes, related and how to build on ourselves by R or Python.

Definition

`Parallel plot` or parallel coordinates plot allows to compare the feature of several individual observations (`series`) on a set of numeric variables. Each vertical bar represents a variable and often has its own scale. (The units can even be different). Values are then plotted as series of lines connected across each axis.

The `iris` dataset provides four features (each represented with a vertical line) for 150 flower samples (each represented with a color line). Samples are grouped in three species. The chart below highlights efficiently that setosa has smaller Petals, but its sepal tends to be wider.

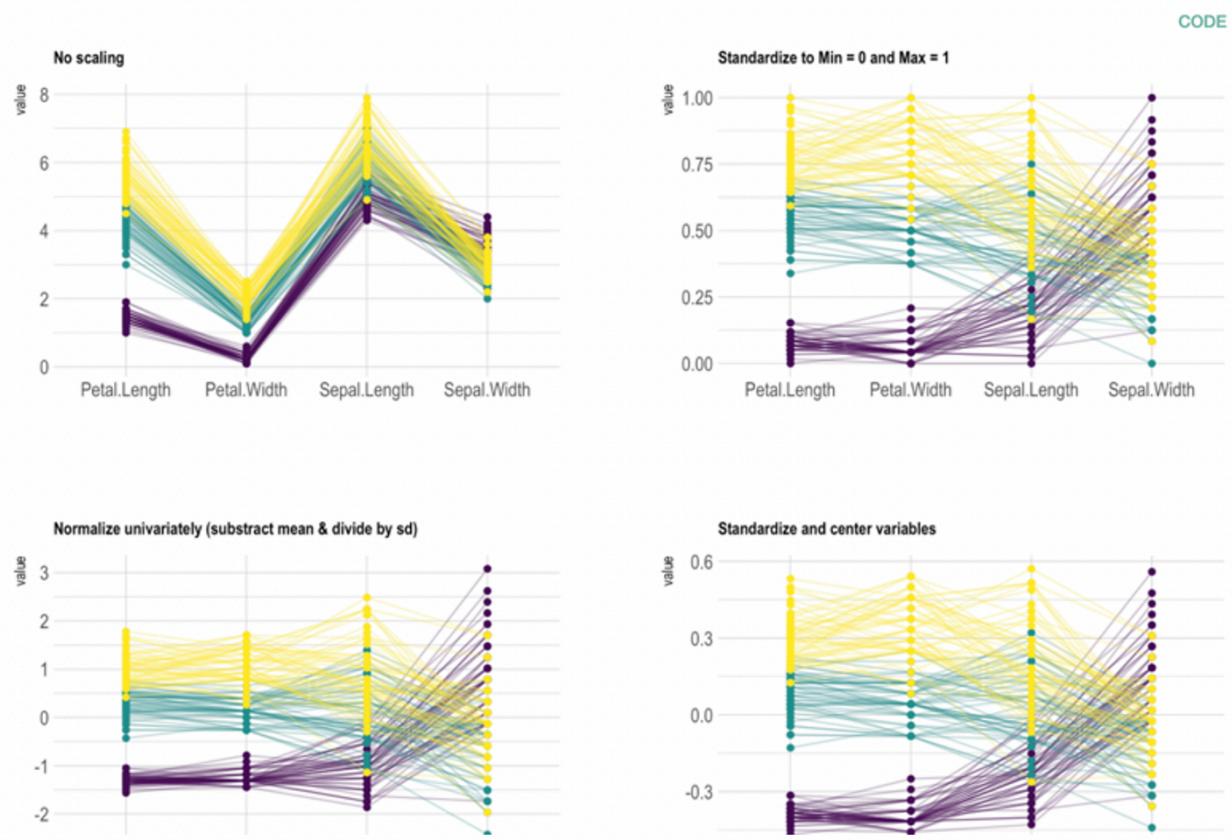


Note: Parallel plot is the equivalent of a [spider chart](#), but with cartesian coordinates. Thus, it is often preferred.

Variation

Here is an overview of the parallel coordinates features you can play with:

- *Scaling* - scaling transforms the raw data to a new scale that is common with other variables. It is a crucial step to compare variables that do not have the same unit, but can also help otherwise as shown in the example below:



Build your own

The [R](#) and [Python](#) graph galleries are 2 websites providing hundreds of chart example, always providing the reproducible code. Click the button below to see how to build the chart you need with your favorite programming language.

[R GRAPH GALLERY](#)[PYTHON GALLERY](#)

Go back to the site we originally searched out and keep scrolling down and we'll get the code for exactly how to draw parallel coordinates charts in R.

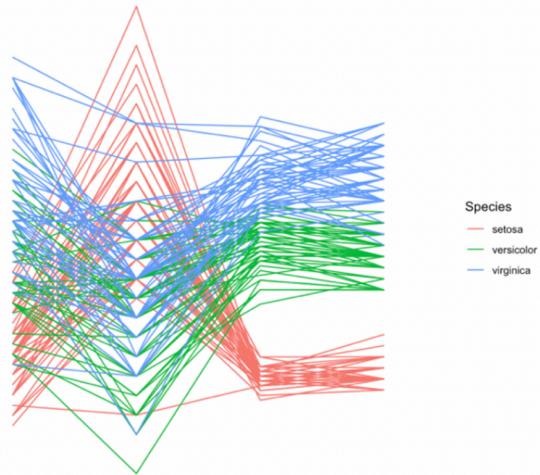
Firstly, we need to use `ggally` library and `ggparcoord()` function. We can also use `groupColumn` variable to categorize variable and color line.

Most basic

This is the most basic parallel coordinates chart you can build with `R`, the `ggally` packages and its `ggparcoord()` function.

The input dataset must be a data frame with several numeric variables, each being used as a vertical axis on the chart. Columns number of these variables are specified in the `columns` argument of the function.

Note: here, a categoric variable is used to color lines, as specified in the `groupColumn` variable.



```
# Libraries
library(GGally)

# Data set is provided by R natively
data <- iris

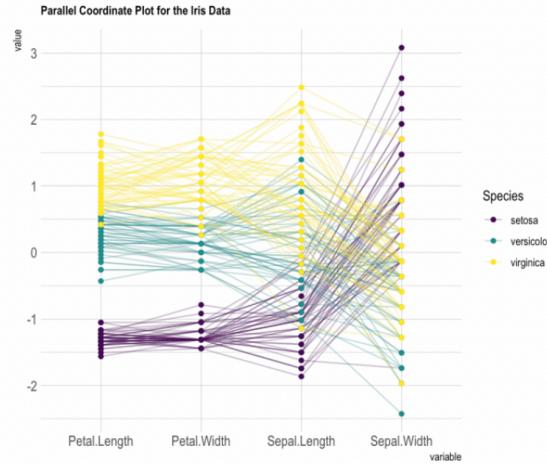
# Plot
ggparcoord(data,
            columns = 1:4, groupColumn = 5
)
```

Then we can also know how to Custom color, theme, general appearance:

Custom color, theme, general appearance

This is pretty much the same chart as the previous one, except for the following customizations:

- color palette is improved thanks to the `viridis` package
- title is added with `title`, and customized in `theme`
- dots are added with `showPoints`
- a bit of transparency is applied to lines with `alphaLines`
- `theme_ipsum()` is used for the general appearance



```
# Libraries
library(hrbrthemes)
library(GGally)
library(viridis)

# Data set is provided by R natively
data <- iris

# Plot
ggparcoord(data,
            columns = 1:4, groupColumn = 5, order = "anyClass",
            showPoints = TRUE,
            title = "Parallel Coordinate Plot for the Iris Data",
            alphaLines = 0.3
          ) +
  scale_color_viridis(discrete=TRUE) +
  theme_ipsum() +
  theme(
    plot.title = element_text(size=10)
  )
```

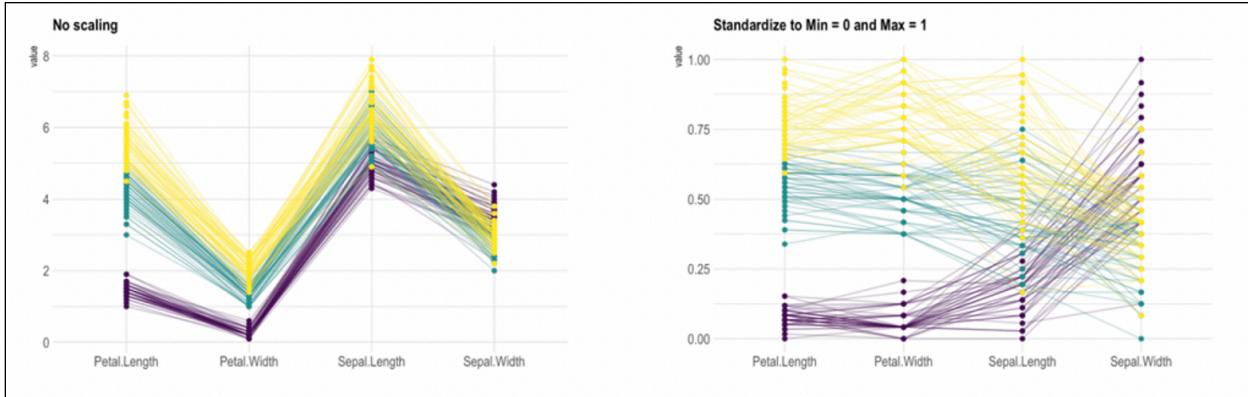
How to scaling:

Scaling

Scaling transforms the raw data to a new scale that is common with other variables. It is a crucial step to compare variables that do not have the same unit, but can also help otherwise as shown in the example below.

The `ggally` package offers a `scale` argument. Four possible options are applied on the same dataset below:

- `globalminmax` → No scaling
- `uniminmax` → Standardize to Min = 0 and Max = 1
- `std` → Normalize univariately (subtract mean & divide by sd)
- `center` → Standardize and center variables



```

ggparcoord(data,
  columns = 1:4, groupColumn = 5, order = "anyClass",
  scale="globalminmax",
  showPoints = TRUE,
  title = "No scaling",
  alphaLines = 0.3
) +
scale_color_viridis(discrete=TRUE) +
theme_ipsum()+
theme(
  legend.position="none",
  plot.title = element_text(size=13)
) +
xlab("")

ggparcoord(data,
  columns = 1:4, groupColumn = 5, order = "anyClass",
  scale="uniminmax",
  showPoints = TRUE,
  title = "Standardize to Min = 0 and Max = 1",
  alphaLines = 0.3
) +
scale_color_viridis(discrete=TRUE) +
theme_ipsum()+
theme(
  legend.position="none",
  plot.title = element_text(size=13)
) +
xlab("")

```

Based on these codes provided on the website, we can modify them according to the requirements of the topic and get parallel coordinates chart we need.

```

qlc <- qlb_temp[qlb_temp$State != "District of Columbia",] %>%
  select(c(1,3,4)) %>%
  spread(.,key = Cause.of.death.Code,value =death_rate)

ggparcoord(data = qlc,columns = 2:9,alphaLines = 0.6,showPoints = F, scale = "globalminmax",boxplot = F) +
  theme(panel.grid.minor = element_blank()) +
  theme(panel.grid.major.y = element_blank()) +
  xlab(NULL)

```

