GGPLOT2

Title, subtitle, caption and tag in ggplot2

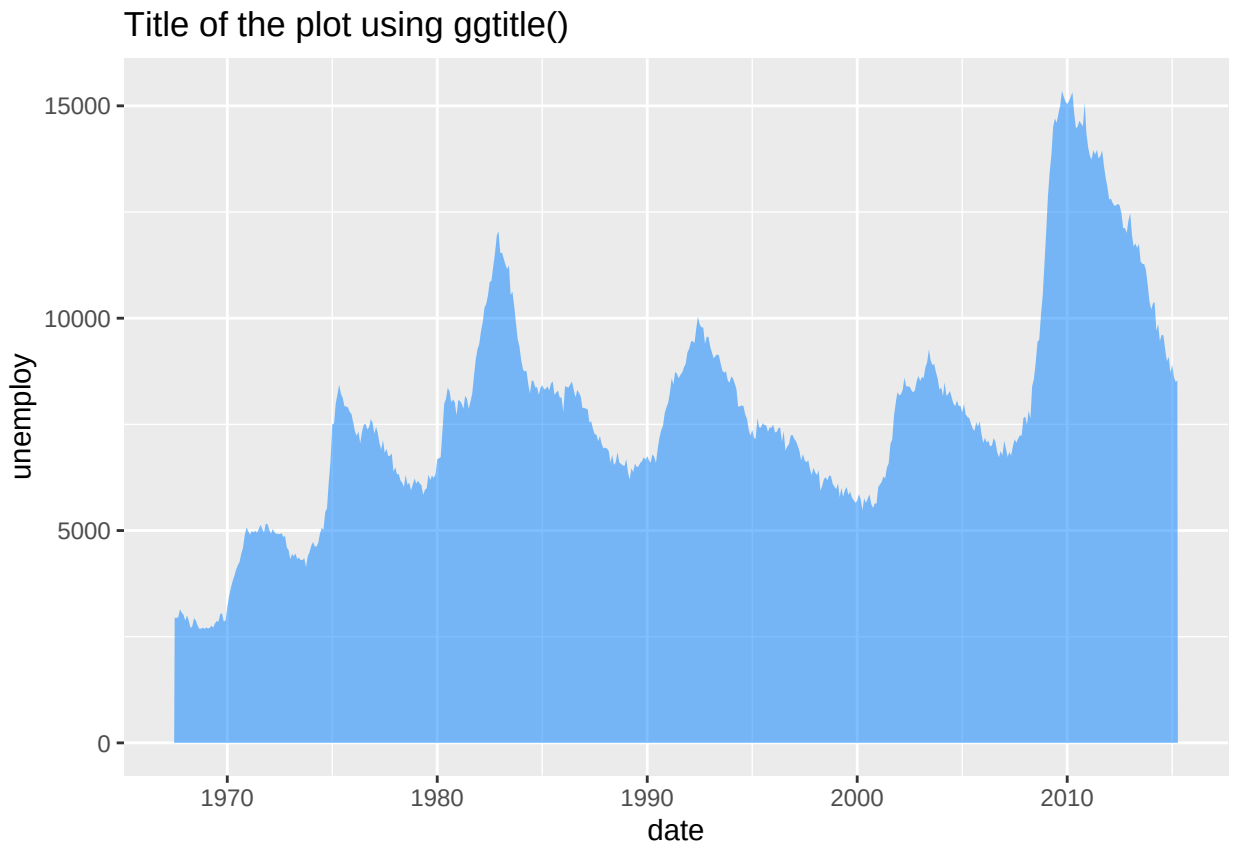
When using ggplot2 you can set a title, a subtitle, a caption and a tag. There are two ways to add titles: using ggtitle or labs function. The former is only for titles and subtitles and the latter also allows adding tags and captions.

Title

Option 1. Using ggtitle

Adding a title in ggplot2 with ggtitle

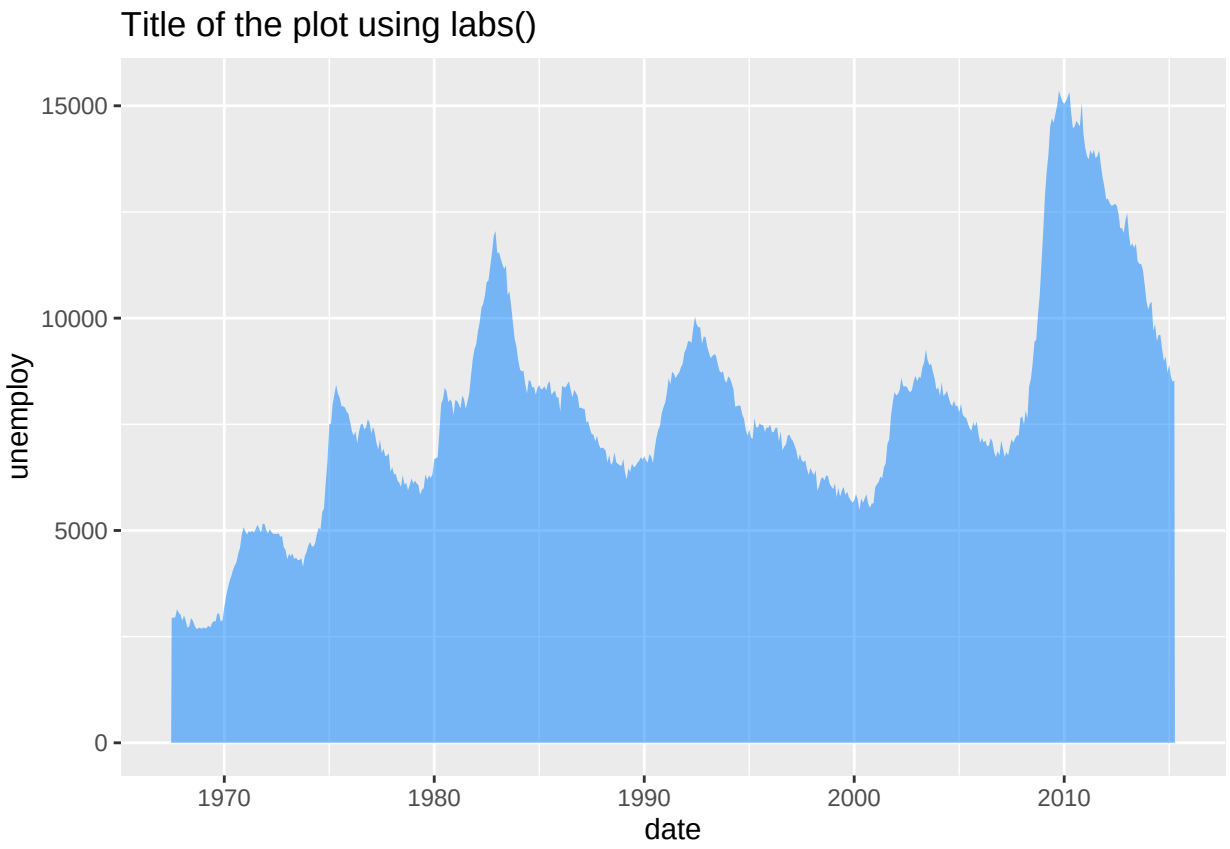
```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  ggtitle("Title of the plot using ggtitle()")
```



Option 2. Using labs

You can also use the labs function to add the title.

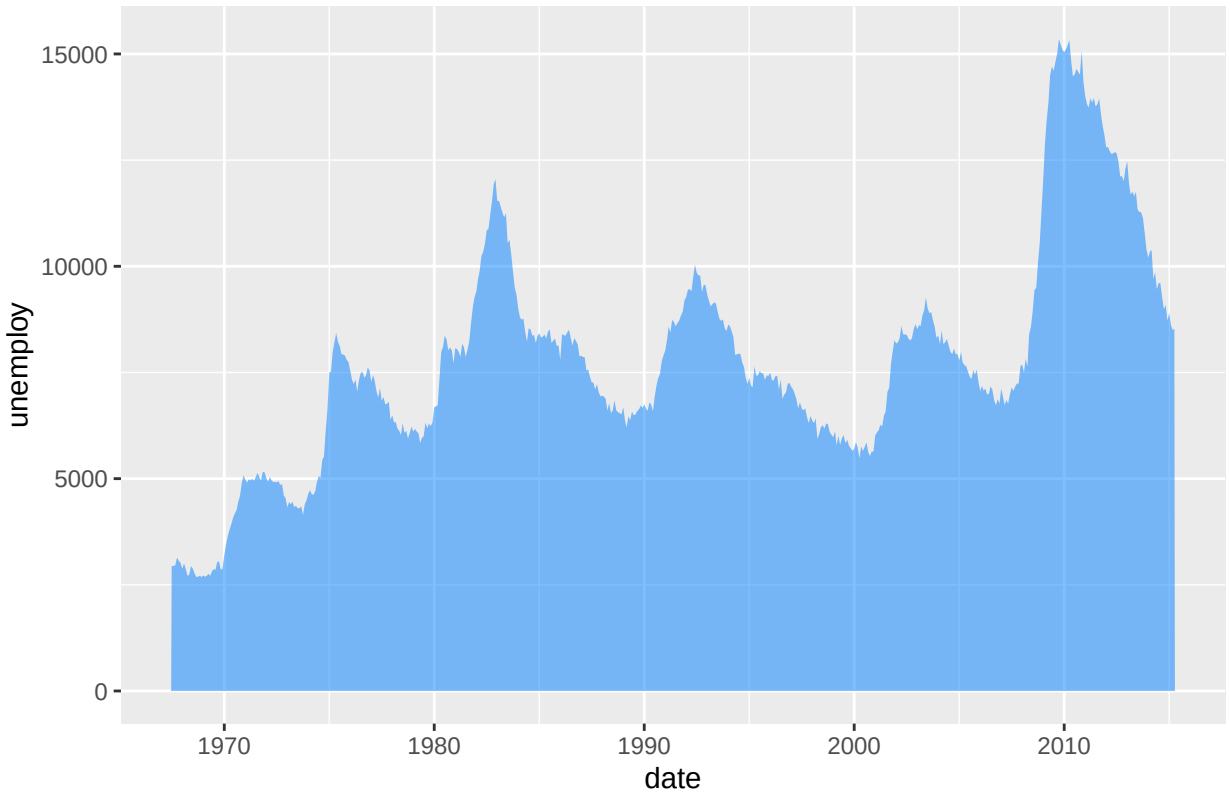
```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title of the plot using labs()")
```



Title position

The title position can be set respect to the whole plot instead of the panel with the plot.title.position component of the theme function. Default value is "panel". This configuration also applies to the subtitle.

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title on the plot margin") +
  theme(plot.title.position = "plot")
```



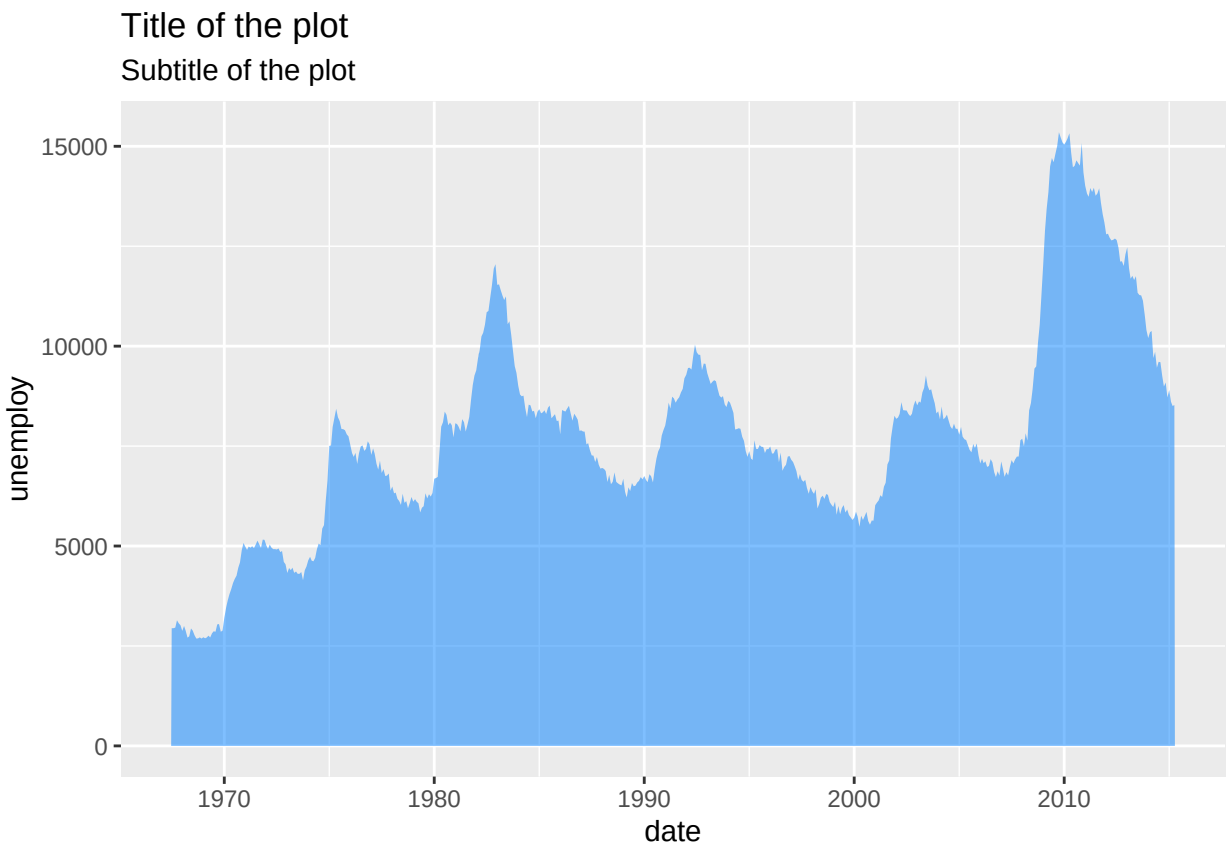
Subtitle

You can add a subtitle the same way you added the title, but with the subtitle argument.

Option 1. Using ggtitle

Add a subtitle in ggplot2

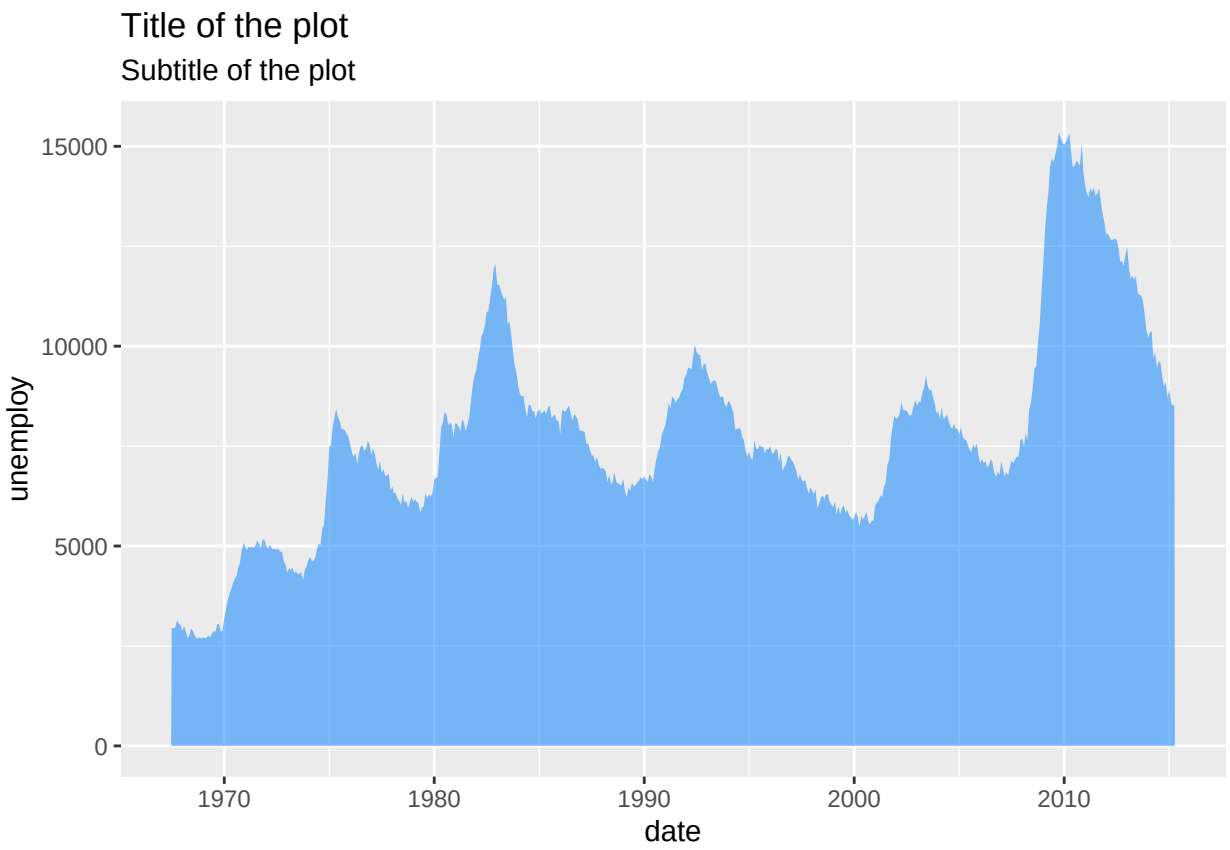
```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  ggtitle("Title of the plot",
          subtitle = "Subtitle of the plot")
```



Option 2. Using labs

Subtitle in ggplot2 using labs

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title of the plot",
       subtitle = "Subtitle of the plot")
```

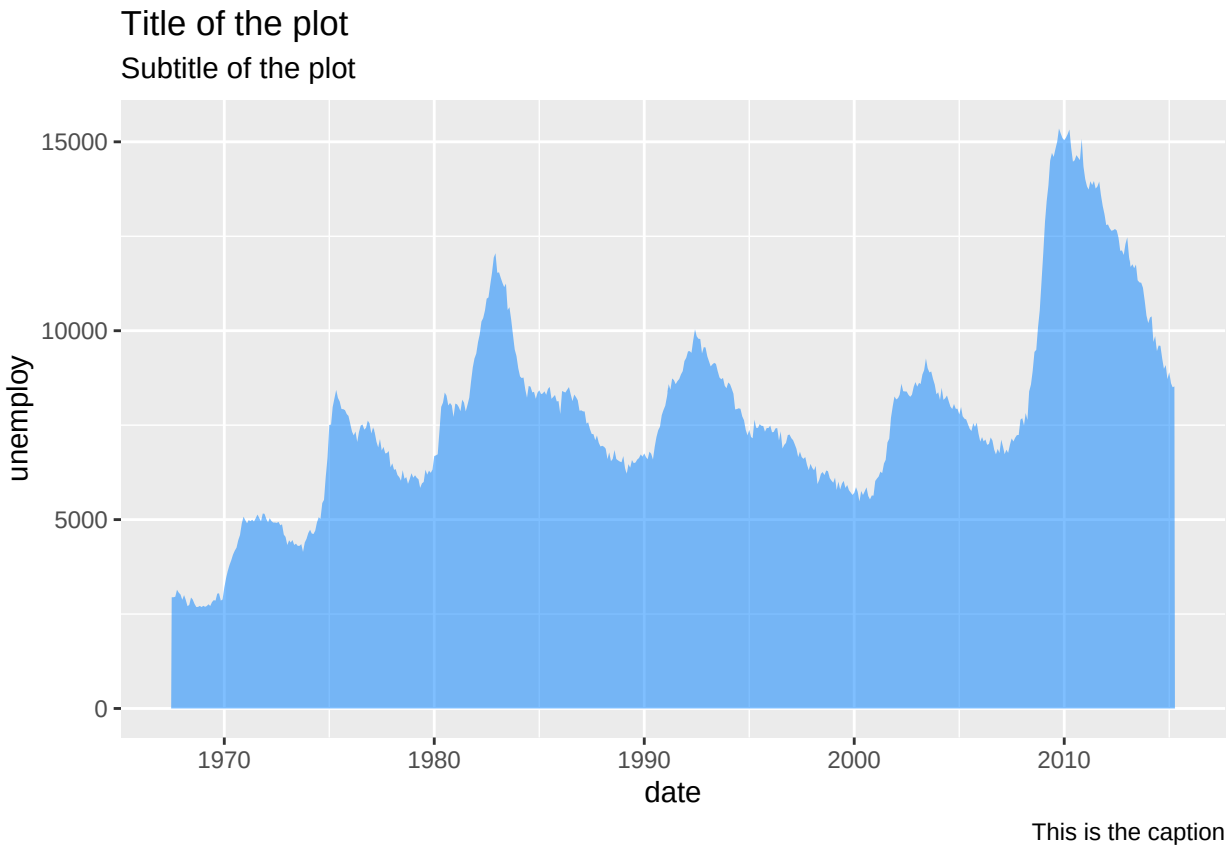


Caption

A caption can be used to describe the figure. You can add it with the caption argument of the labs function.

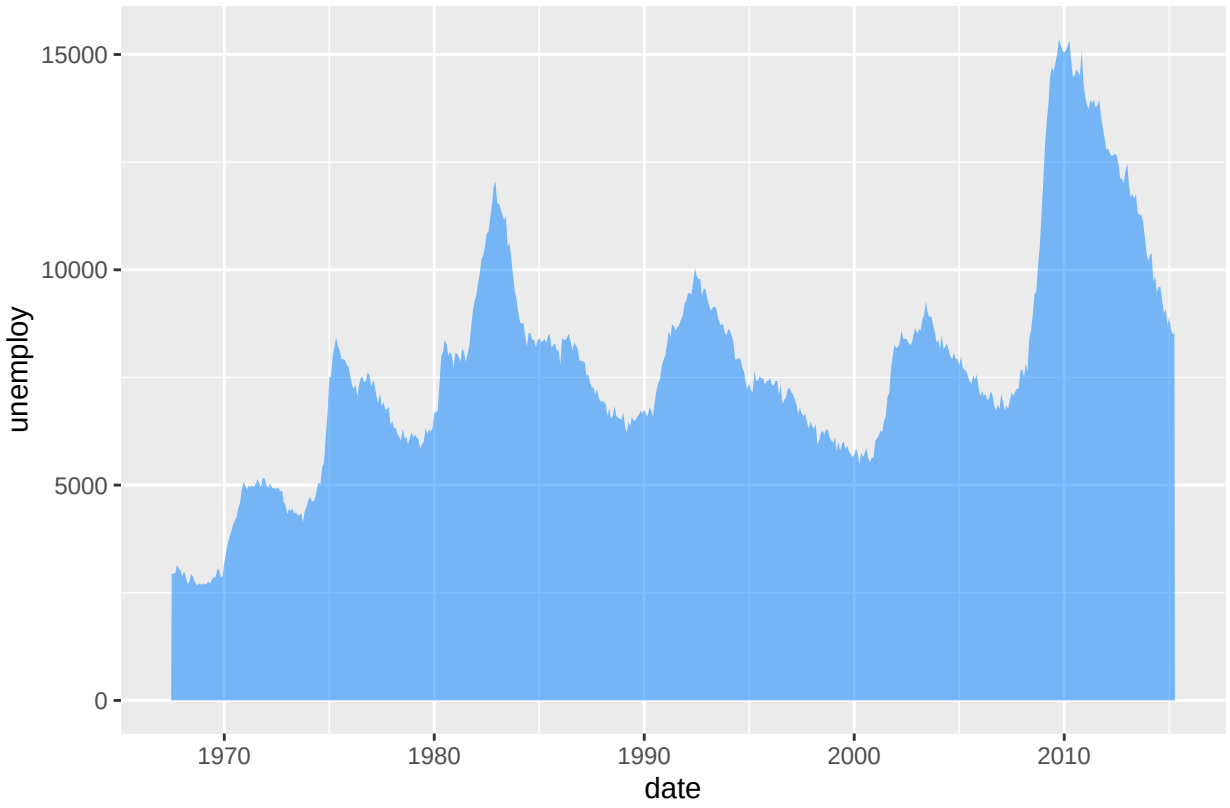
Adding a caption in ggplot2

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title of the plot",
       subtitle = "Subtitle of the plot",
       caption = "This is the caption")
```



The plot.caption.position of the theme function allows aliging the caption to the panel ("panel", default) or the whole plot ("plot").

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(caption = "This is the caption") +
  theme(plot.caption.position = "plot",
        plot.caption = element_text(hjust = 0))
```



This is the caption

Changing the caption position in ggplot2

Note that the default for the caption is right alignment, so you can set hjust = 0 to move the caption to the left of the whole plot.

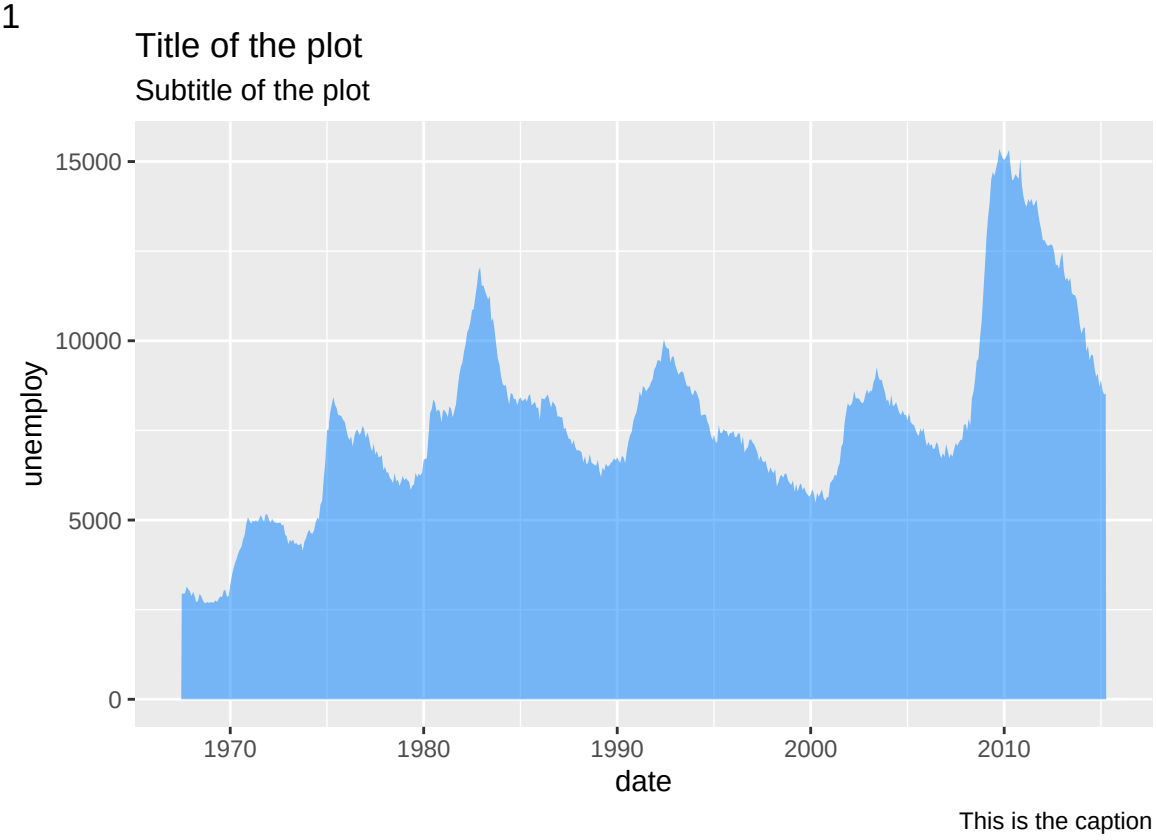
Tag

Adding a tag to the plot in ggplot2

Tags are useful to indicate the figure numbering. You can add it with the tag argument of the labs function.

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title of the plot",
       subtitle = "Subtitle of the plot",
       caption = "This is the caption",
       tag = "Fig. 1")
```

Fig. 1



Tag position

Changing the tag position in ggplot2

The position of the tag can be set with the plot.tag.position component of the theme function. Possible values are “topleft” (default), “top”, “topright”, “left”, “right”, “bottomleft”, “bottom” or “bottomright”.

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(tag = "Fig. 1") +
  theme(plot.tag.position = "bottomright")
```

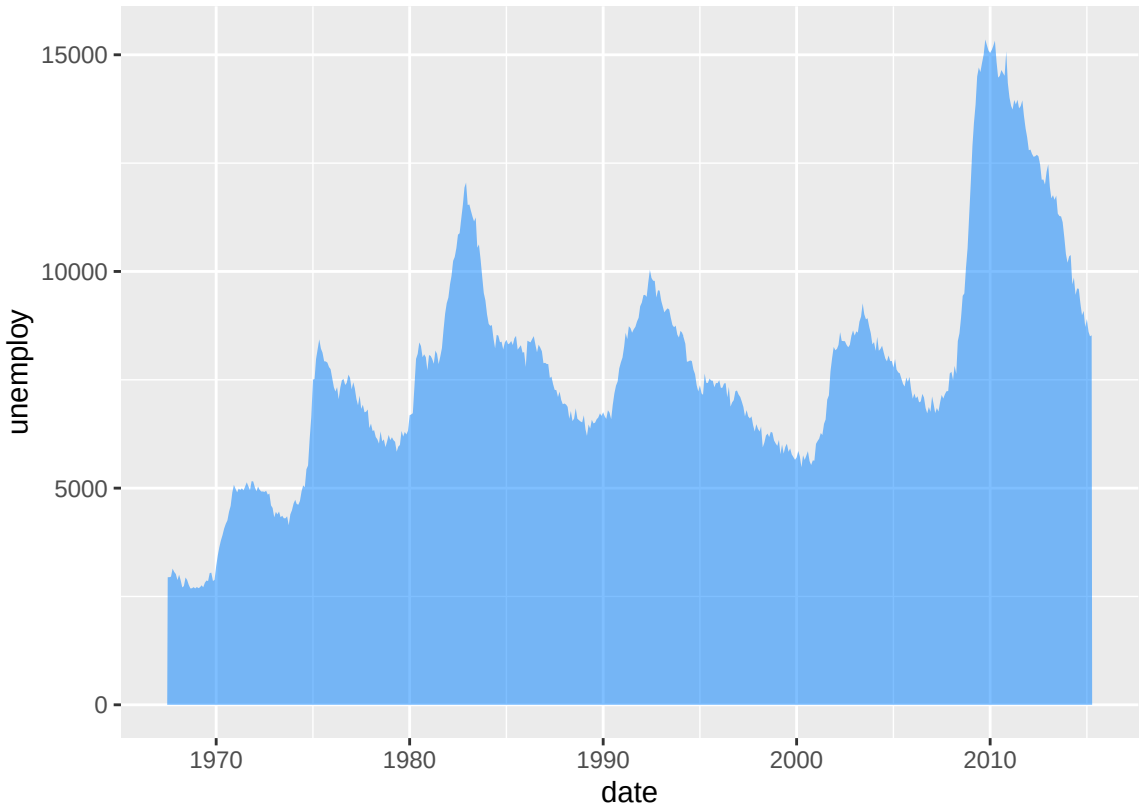


Fig. 1

Further styles and adjustments

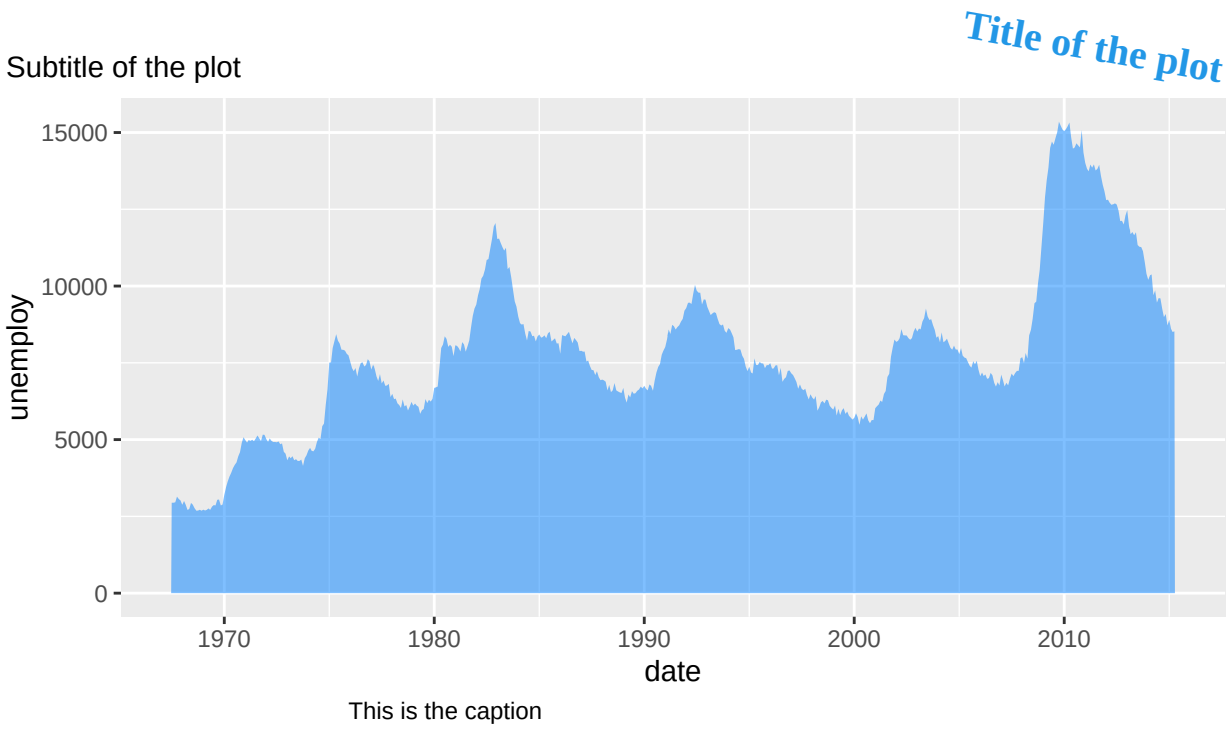
The titles, subtitles, captions and tags can be customized with the plot.title, plot.subtitle, plot.caption and plot.tag components of the theme function, making use of element_text. You can modify the color, the font family, the text size, the text face, the angle or the vertical and horizontal adjustment for each text as in the example below.

Styling the titles in ggplot2, such as modifying the color, size, face, family, lineheight and margins

```
# install.packages(ggplot2)
library(ggplot2)

ggplot(economics, aes(date, unemploy)) +
  geom_area(fill = rgb(0, 0.5, 1, alpha = 0.5)) +
  labs(title = "Title of the plot",
       subtitle = "Subtitle of the plot",
       caption = "This is the caption",
       tag = "Fig. 1") +
  theme(plot.title = element_text(family = "serif",              # Font family
                                  face = "bold",                 # Font face
                                  color = 4,                     # Font color
                                  size = 15,                     # Font size
                                  hjust = 1,                     # Horizontal adjustment
                                  vjust = 1,                     # Vertical adjustment
                                  angle = -10,                   # Font angle
                                  lineheight = 1,                # Line spacing
                                  margin = margin(20, 0, 0, 0)), # Margins (t, r, b, l)
```

```
        plot.subtitle = element_text(hjust = 0),                    # Subtitle customization
        plot.caption = element_text(hjust = 0.25),                  # Caption customization
        plot.tag = element_text(face = "italic"),                   # Tag customization
        plot.title.position = "plot",                               # Title and subtitle position ("plot"
        plot.caption.position = "panel",                            # Caption position ("plot" or "panel")
        plot.tag.position = "top")                                  # Tag position
```

Fig. 1



Text annotations in ggplot2

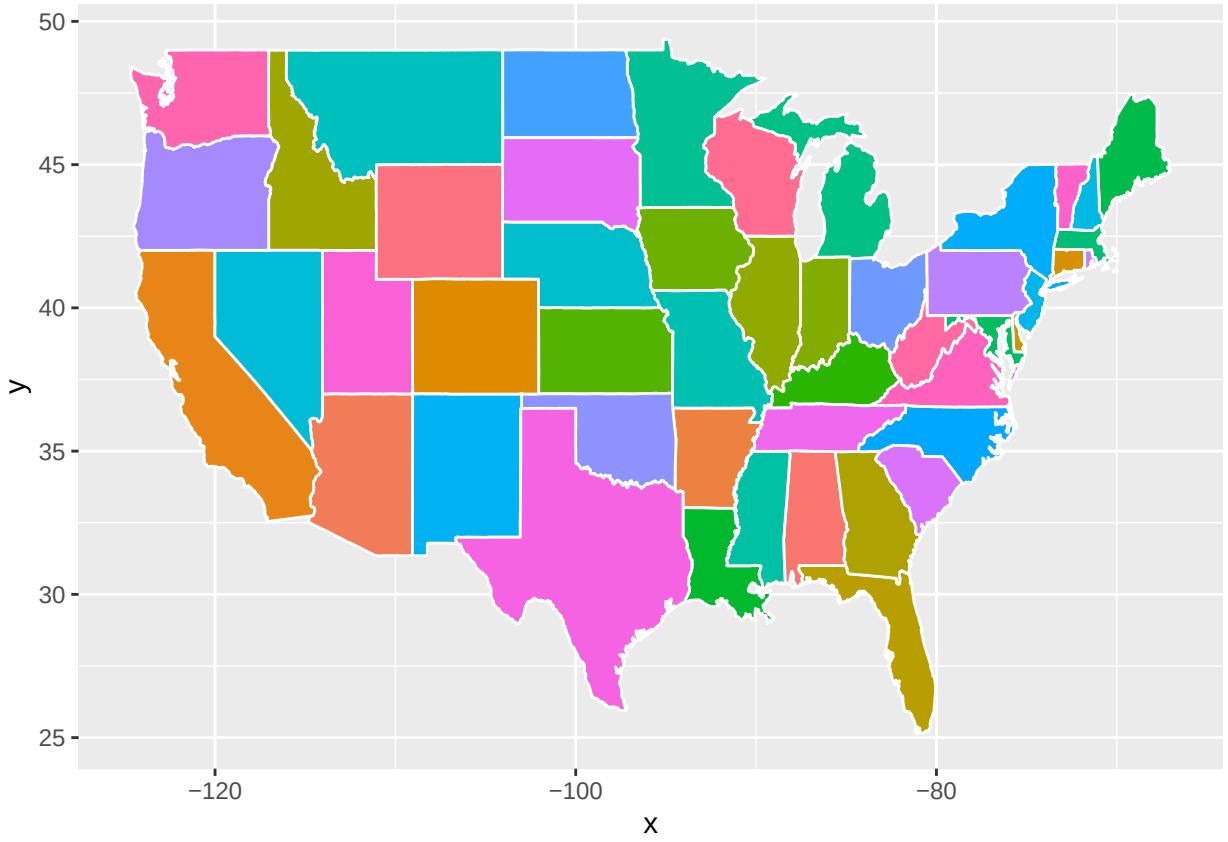
Adding text with geom_text Adding labels with geom_label Avoid overlapping with ggrepel Use markdown and HTML with ggtext The geom_text and geom_label functions allows adding text or labels, respectively, to plots created with ggplot2. You can add some annotations to some coordinates or label data points.

```
# install.packages("ggplot2")
library(ggplot2)

# install.packages("maps")
library(maps)

df <- data.frame(x = state.center$x, y = state.center$y,
                 state = state.name)

p <- ggplot(df, aes(x = x, y = y)) +
       geom_polygon(data = map_data("state"),
                    color = "white",
                    aes(x = long, y = lat,
```

```
                        fill = map_data("state")$region,
                        group = group)) +
      guides(fill = FALSE)
```

```
## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as
## of ggplot2 3.3.4.
```

```
p
```



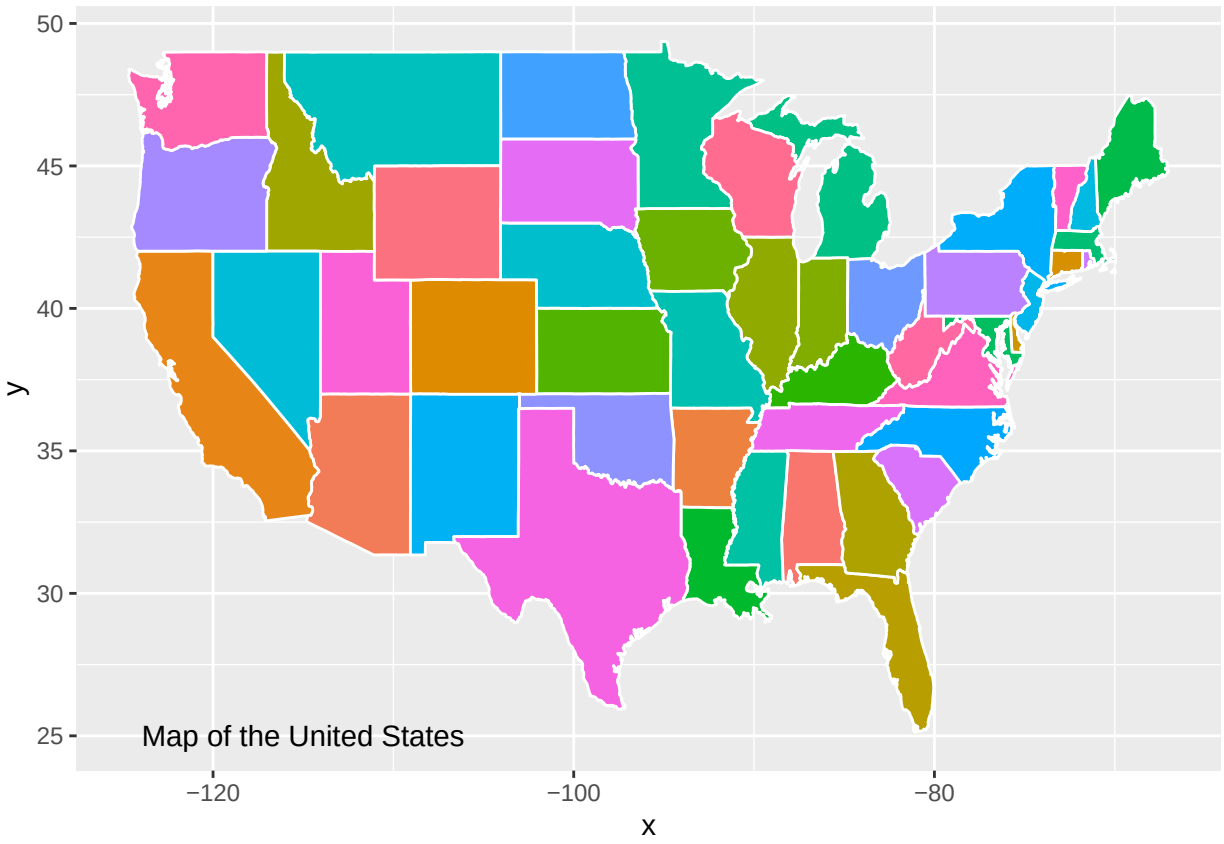
In this guide we are going to use the following example plot.

ggplot2 map

Adding text with geom_text Use the geom_text function to add texts in ggplot2

Adding text annotations

```
p +
  geom_text(aes(x = -115, y = 25,
                label = "Map of the United States"),
            stat = "unique")
```



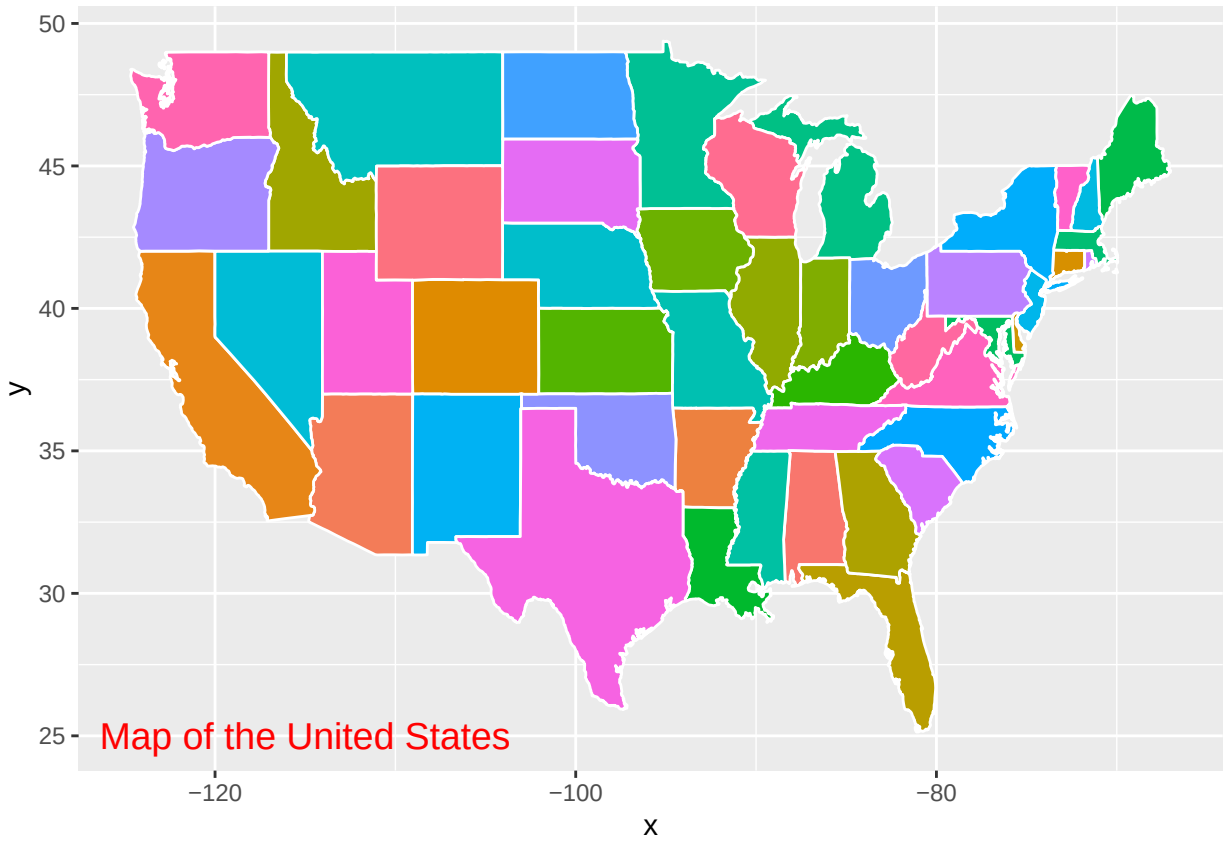
Set stat = “unique”, otherwise the label will be redrawn for each data point on your data frame.

Customize the size and the color of the text using geom_text

Customizing the annotations

There are several arguments that you can customize, such as the color or the size of the text.

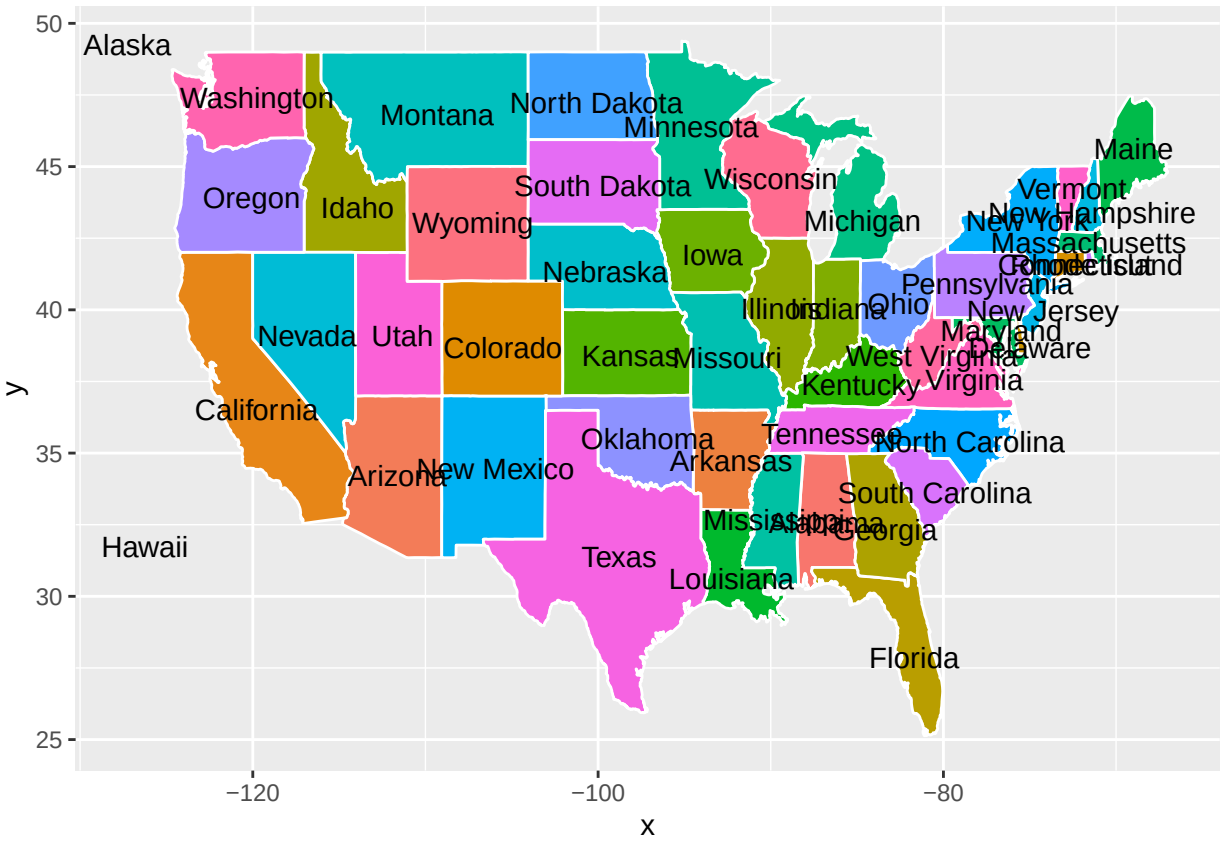
```
p +
  geom_text(aes(x = -115, y = 25,
                label = "Map of the United States"),
            stat = "unique",
            size = 5, color = "red")
```



Label observations in ggplot2

If your data set contains a variable with groups or labels you can pass it to the label argument.

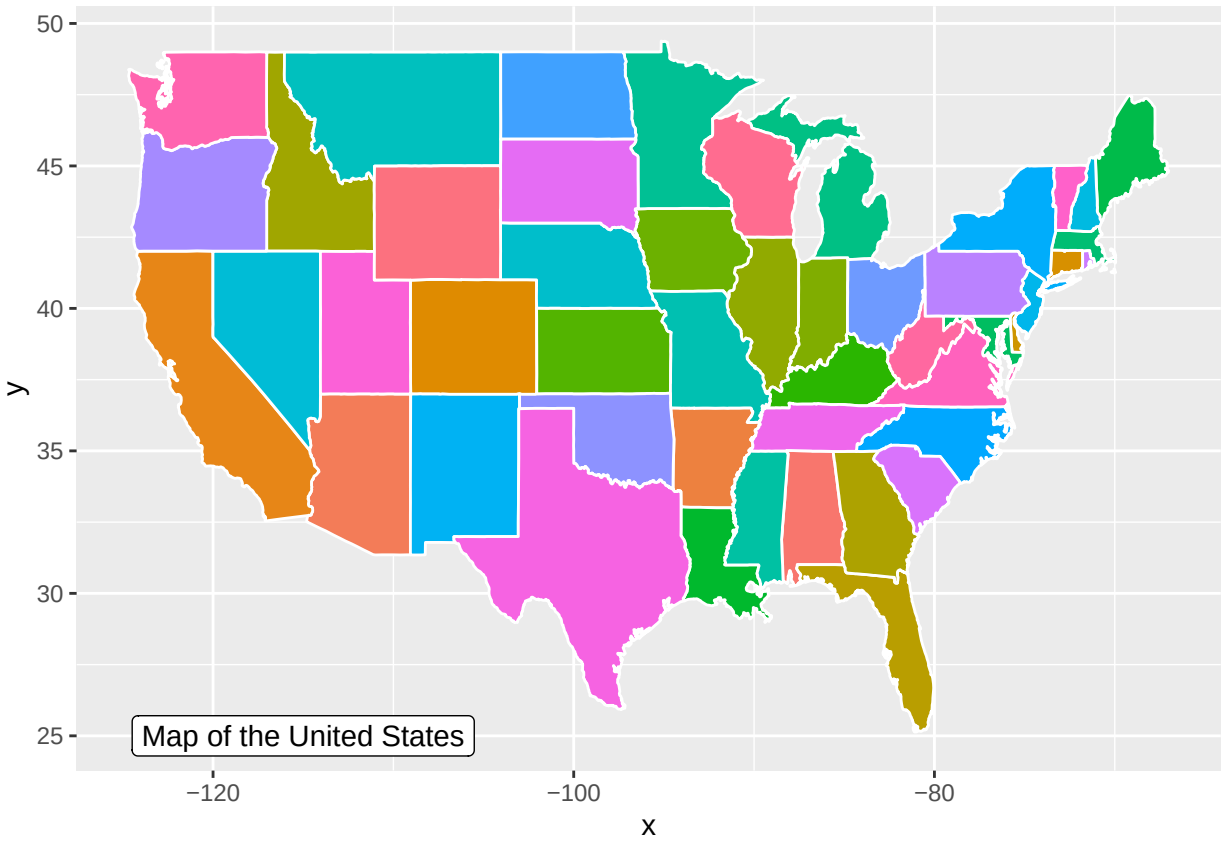
```
p +
  geom_text(aes(label = state))
```



Adding labels with geom_label If you prefer adding labels instead of raw text use geom_label. The function behaves the same as the previous but with a background, making the text easier to read.

Label annotation

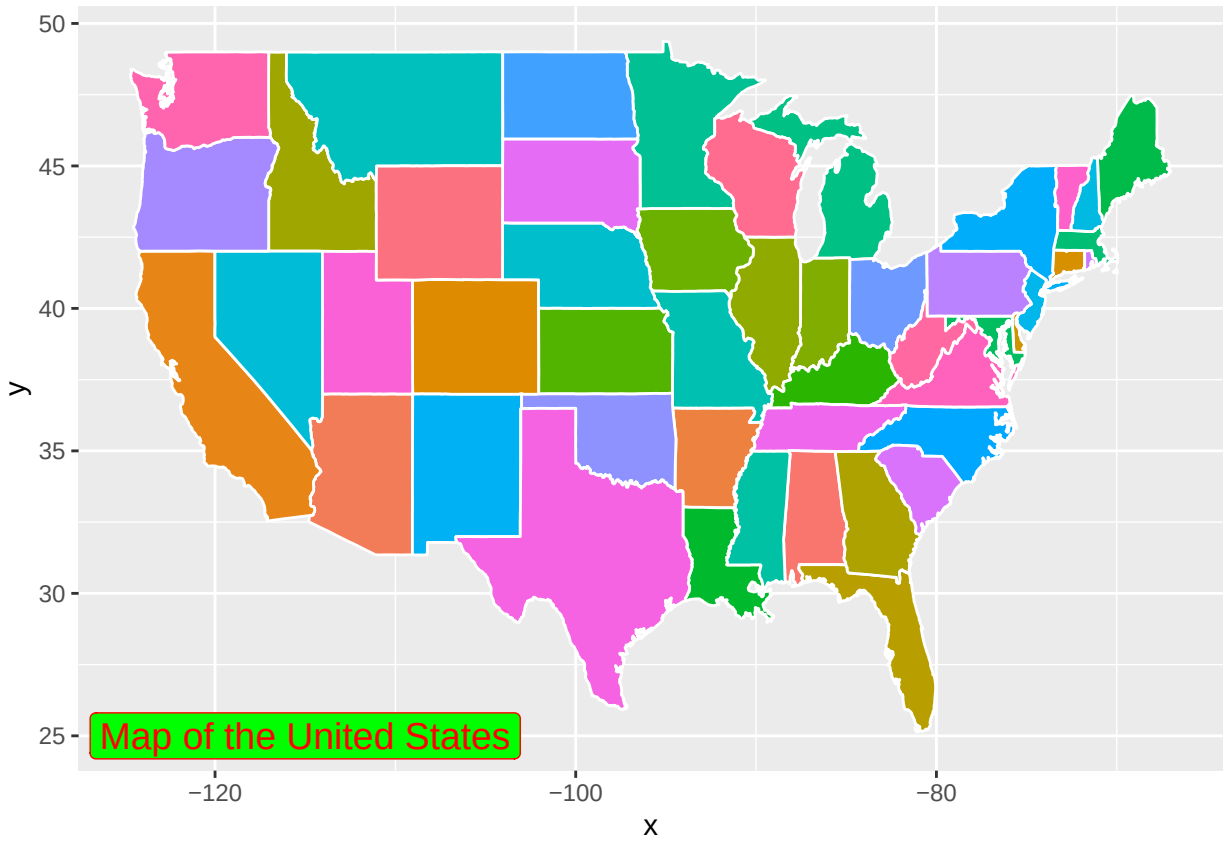
```
p +
  geom_label(aes(x = -115, y = 25,
                 label = "Map of the United States"),
             stat = "unique")
```



Use the geom_label function to add labels in ggplot2

Customizing the label

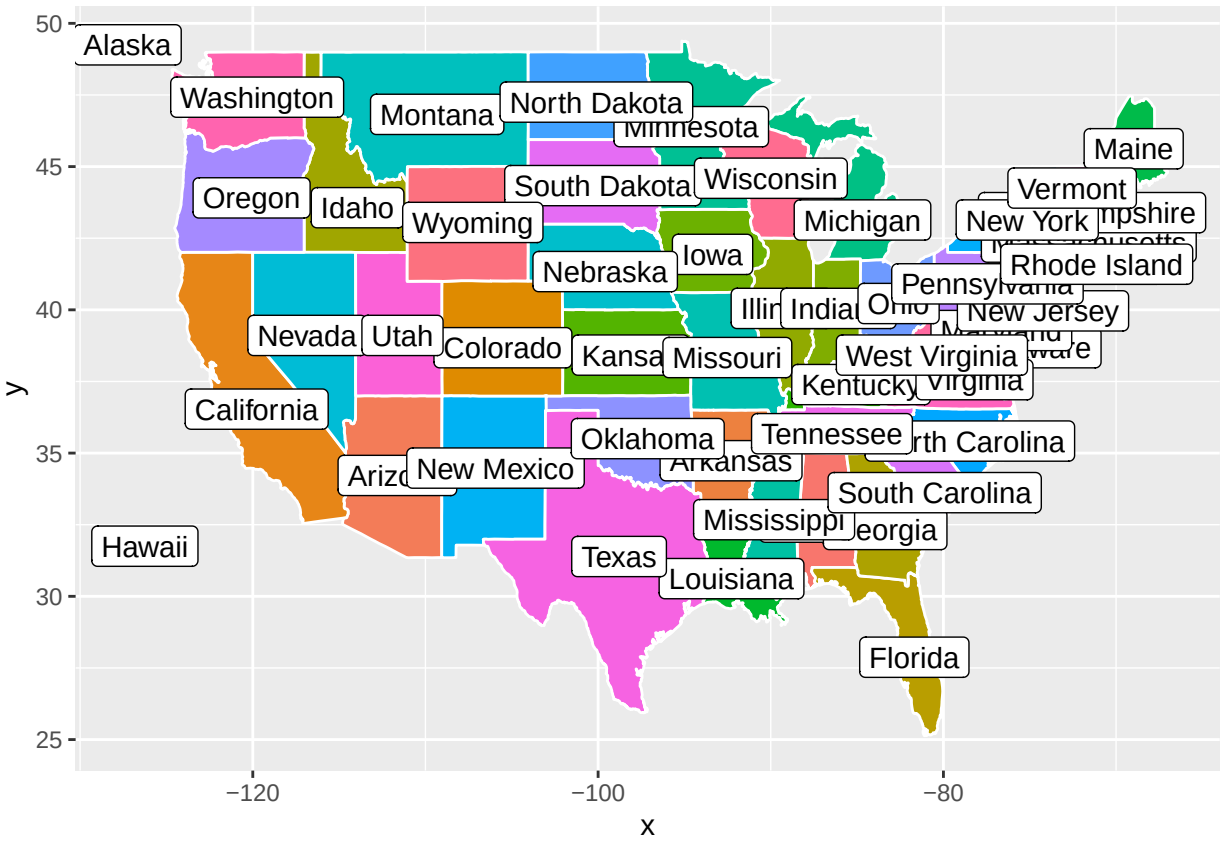
```
p +
  geom_label(aes(x = -115, y = 25,
                 label = "Map of the United States"),
             stat = "unique",
             size = 5, color = "red", fill = "green")
```



Customize the colors and size of geom_label

Labelling points

```
p +
  geom_label(aes(label = state))
```



Labelling observations in ggplot2 with geom_label

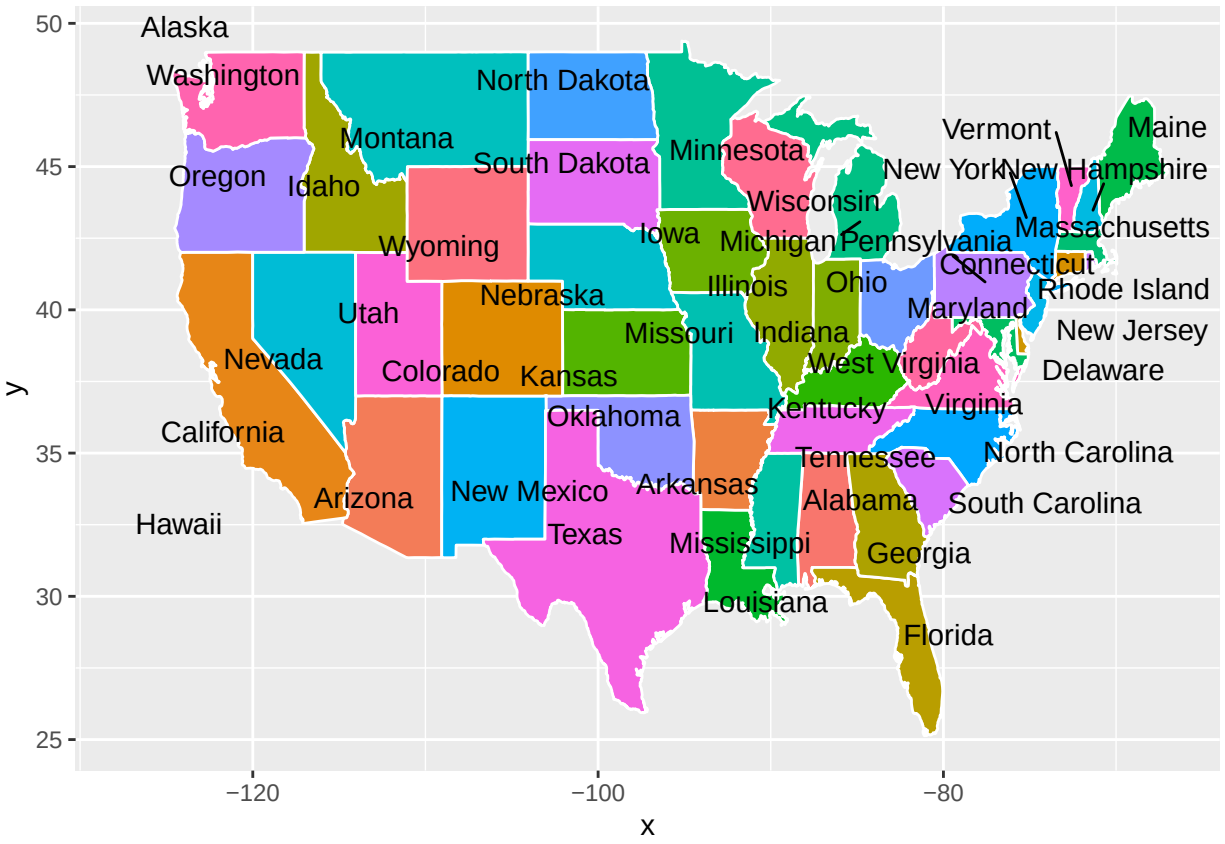
Use ggrepel to avoid overlapping of the texts or labels.

Avoid overlapping with ggrepel The text and the labels are placed on the coordinates you set, but can overlap. The ggrepel package provides geom_text_repel and geom_label_repel functions, which make the labels repel away from each other as much as possible.

geom_text_repel function from ggrepel to avoid overlapping

geom_text_repel

```
# install.packages("ggrepel")
library(ggrepel)

p +
  geom_text_repel(aes(label = state))
```

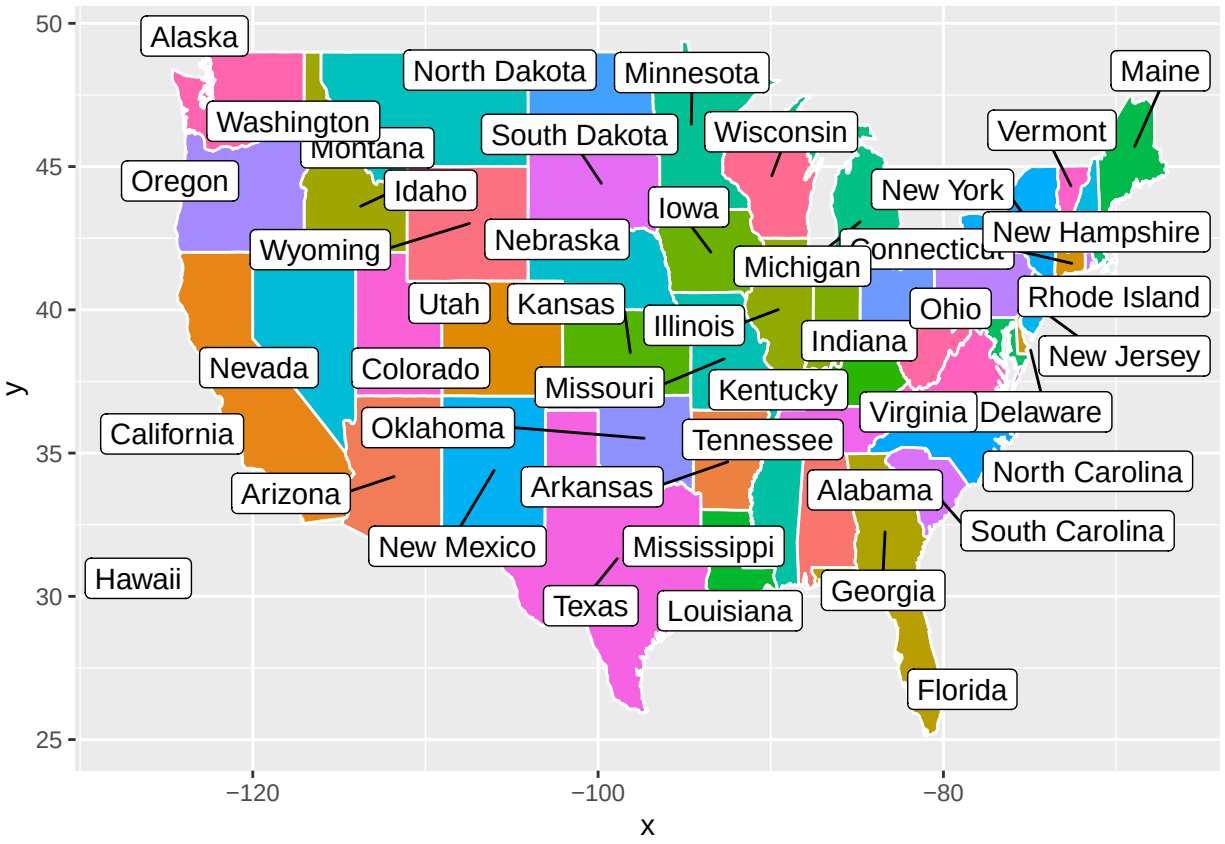


Use the geom_label_repel function to add labels without overlapping in ggplot2

geom_label_repel

```
# install.packages("ggrepel")
library(ggrepel)

p +
  geom_label_repel(aes(label = state))
```

```
## Warning: ggrepel: 4 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

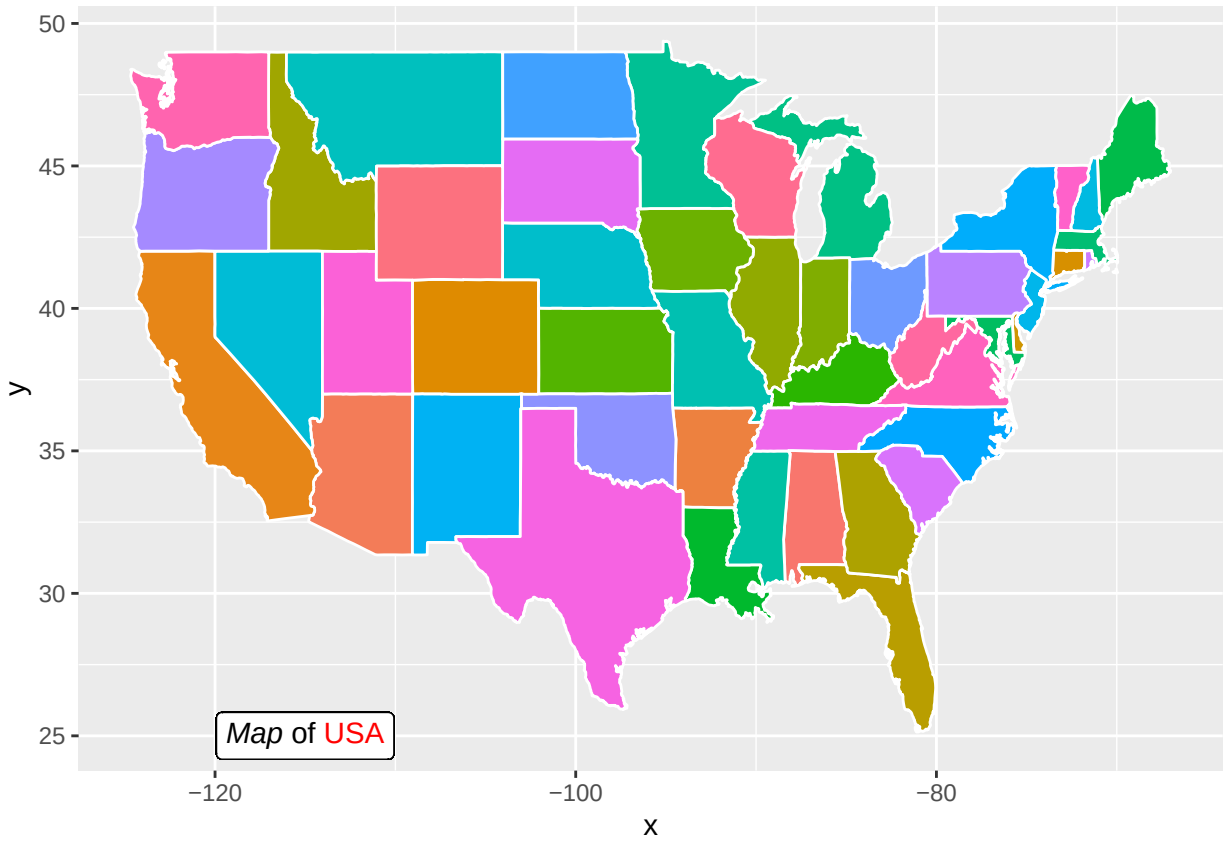


You can customize the colors, fonts and other arguments the same way as with geom_text or geom_label. See the package examples for more use cases.

Use markdown and HTML with ggtext If you want to fully customize you annotations use the geom_richtext function from ggtext, which allows you to add markdown and HTML formatting to your text annotations.

Add rich text

```
# install.packages("ggtext")
library(ggtext)

lab <- "*Map* of <span style = 'color:red'>USA</span>"
p +
  geom_richtext(aes(x = -115, y = 25,
                    label = lab))
```

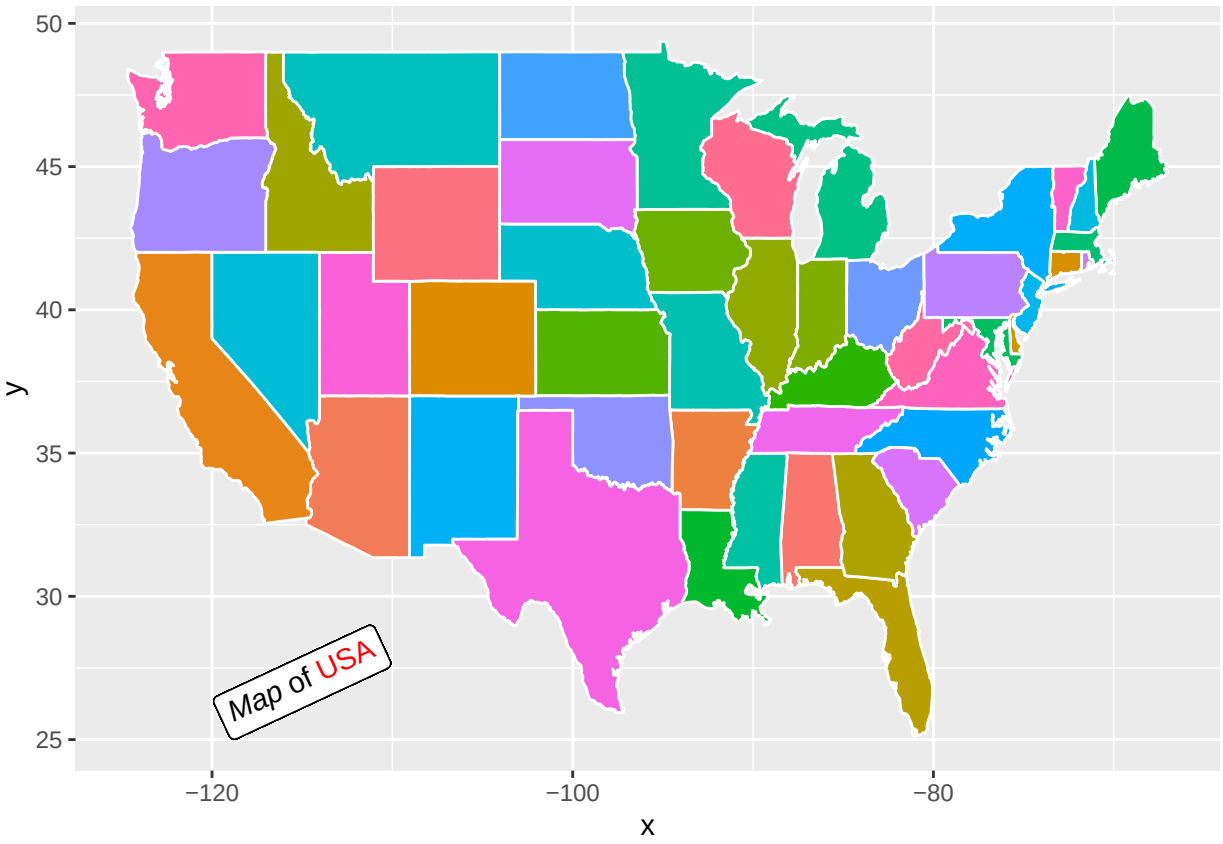


geom_richtext function to add rich text, HTML and markdown in ggplot2

Rotate the text

You can even rotate the text, a feature that is not available with default ggplot2.

```
# install.packages("ggtext")
library(ggtext)

lab <- "*Map* of <span style = 'color:red'>USA</span>"
p +
  geom_richtext(aes(x = -115, y = 27,
                    label = lab),
                angle = 25)
```



Rotate labels or texts in ggplot2

Background color in ggplot2

Panel background color

Panel border color

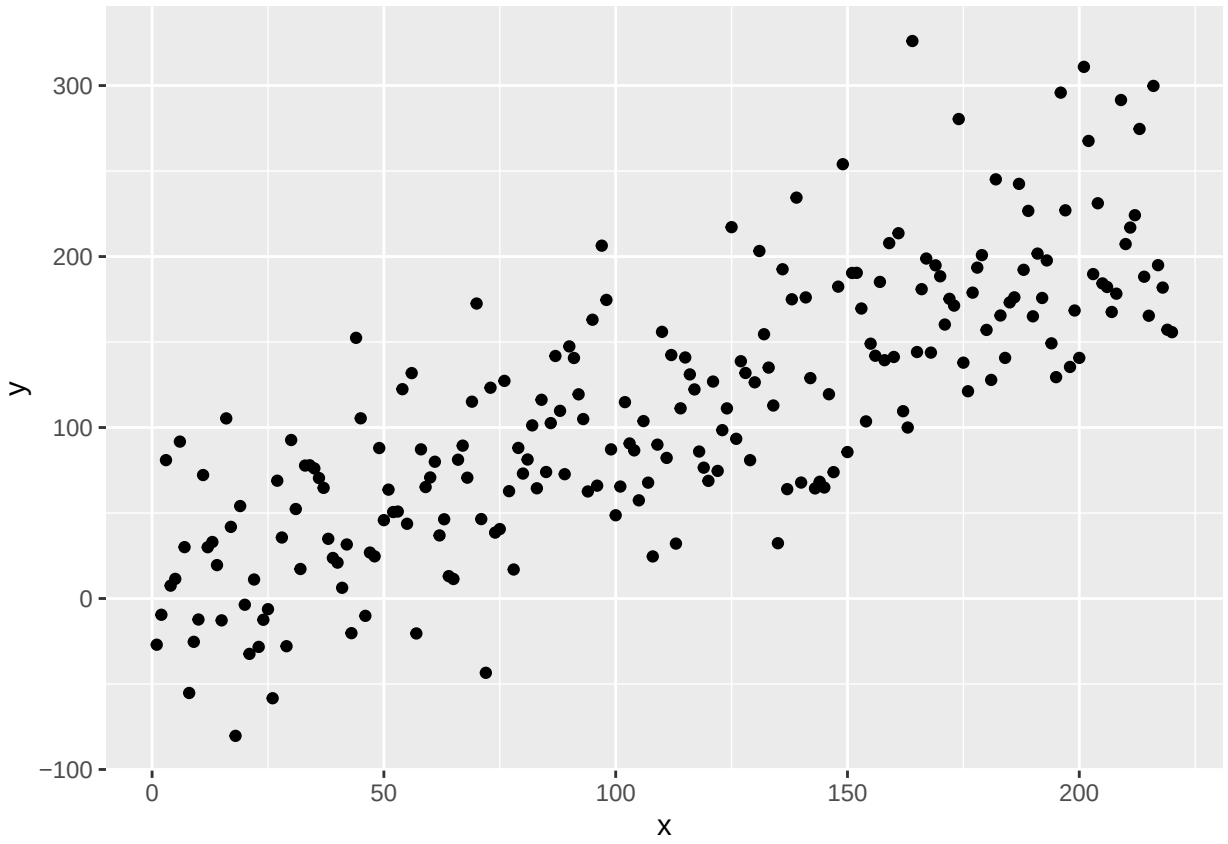
Plot background color

Plot border color

Changing the colors with themes Color picker Default plot

By default, ggplot2 plots have a gray panel and a white background.

```
library(ggplot2)

# Sample data
set.seed(123)
x <- 1:220
y <- x + rnorm(220, sd = 50)

df <- data.frame(x = x, y = y)

# Plot
ggplot(data = df, aes(x = x, y = y)) +
  geom_point()
```

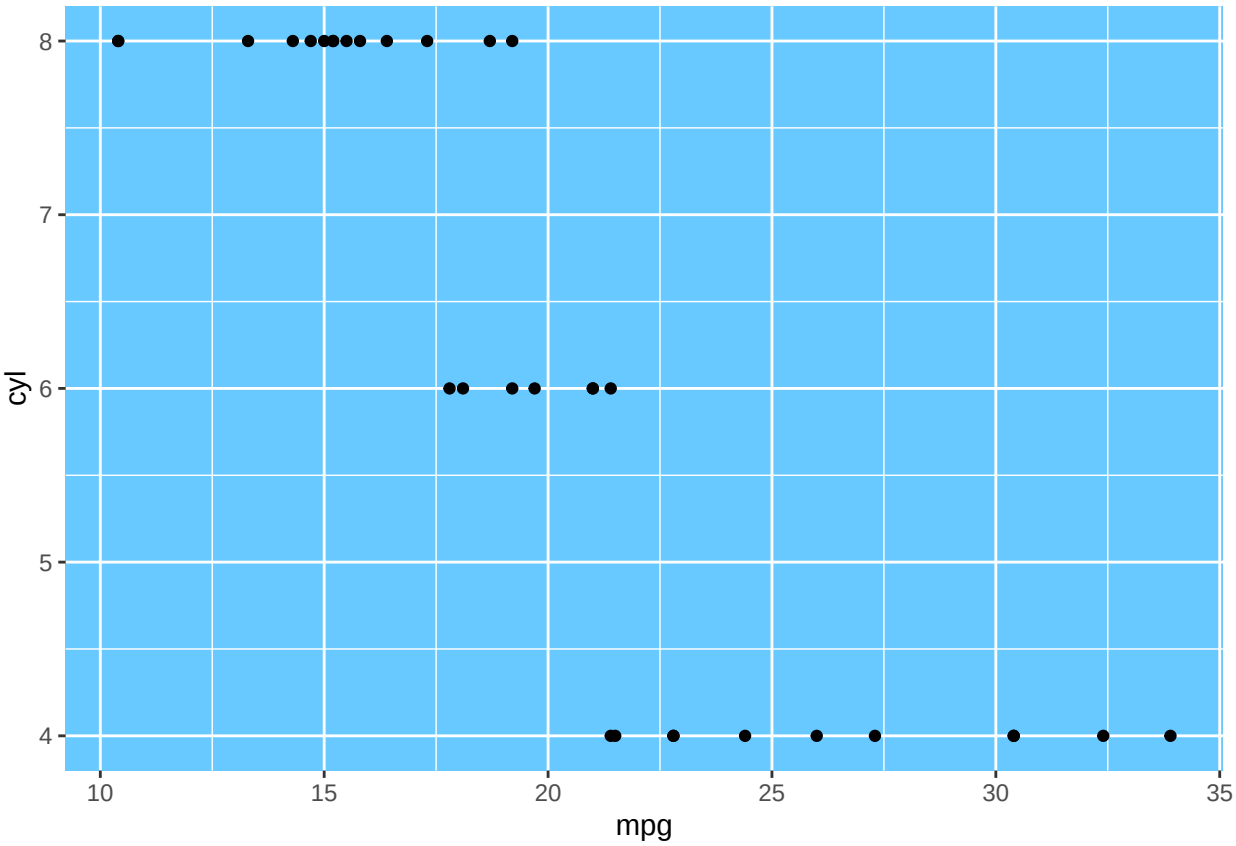


Default background color in ggplot2

Panel background color You can change the panel background color setting an element_rect in the panel.background component of the theme function as follows.

Change the background panel color in ggplot2

```
library(ggplot2)

ggplot(data = mtcars, aes(x = mpg, y = cyl))+
  geom_point() +
  theme(panel.background = element_rect(fill = "#67c9ff"))
```



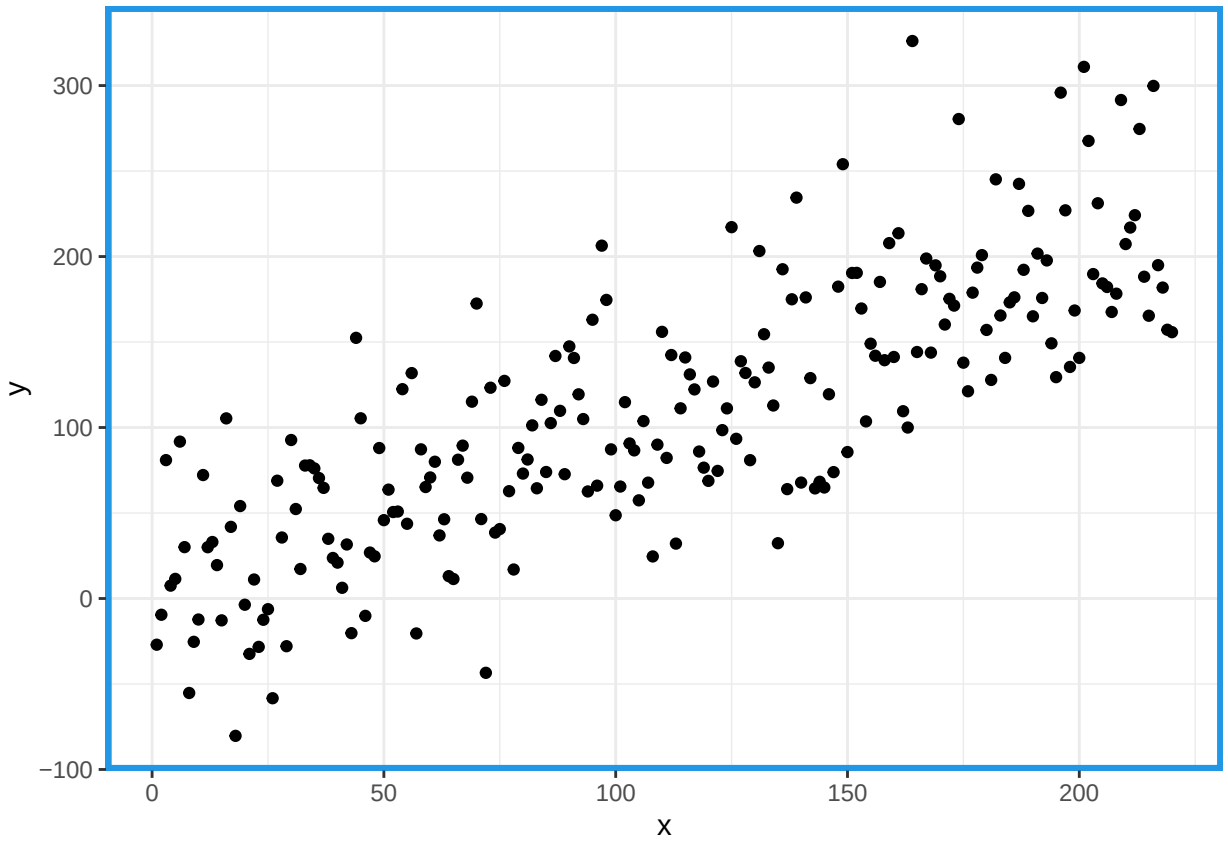
Panel border color Option 1

ggplot2 panel border color

The panel.border component of the theme function controls the color and width of the border of the panel with the color and size arguments. However, you will need to set fill = "transparent" to avoid hiding the data.

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_bw() +
  theme(panel.border = element_rect(fill = "transparent", # Needed to add the border
                                    color = 4,            # Color of the border
                                    size = 2))            # Border width
```

```
## Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
```

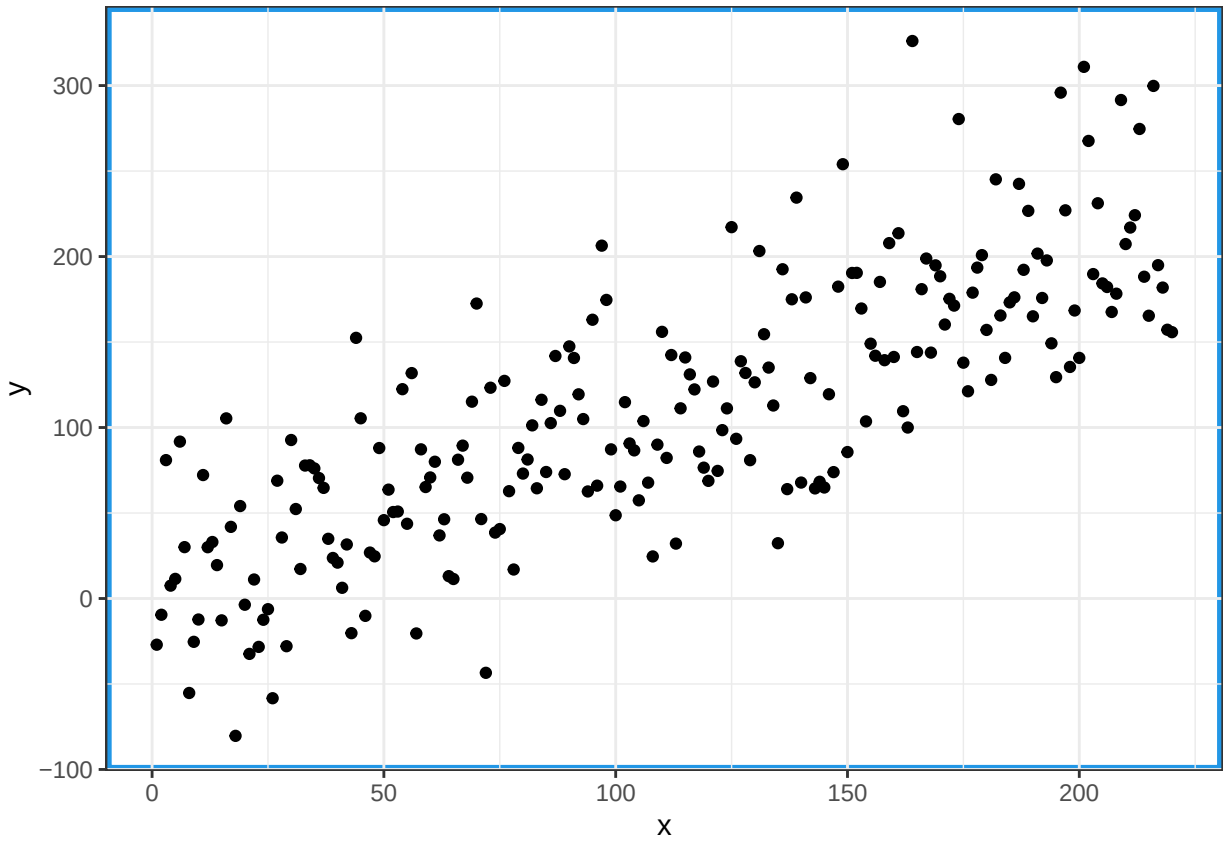


Option 2

Border color of the plot in ggplot2

You can also set a element_rect for the panel.background component and modify the border color with the color argument. However, this is not the recommended workflow, as it doesn't override the current border. You can check this with theme_bw (Note that the black border is behind the blue border).

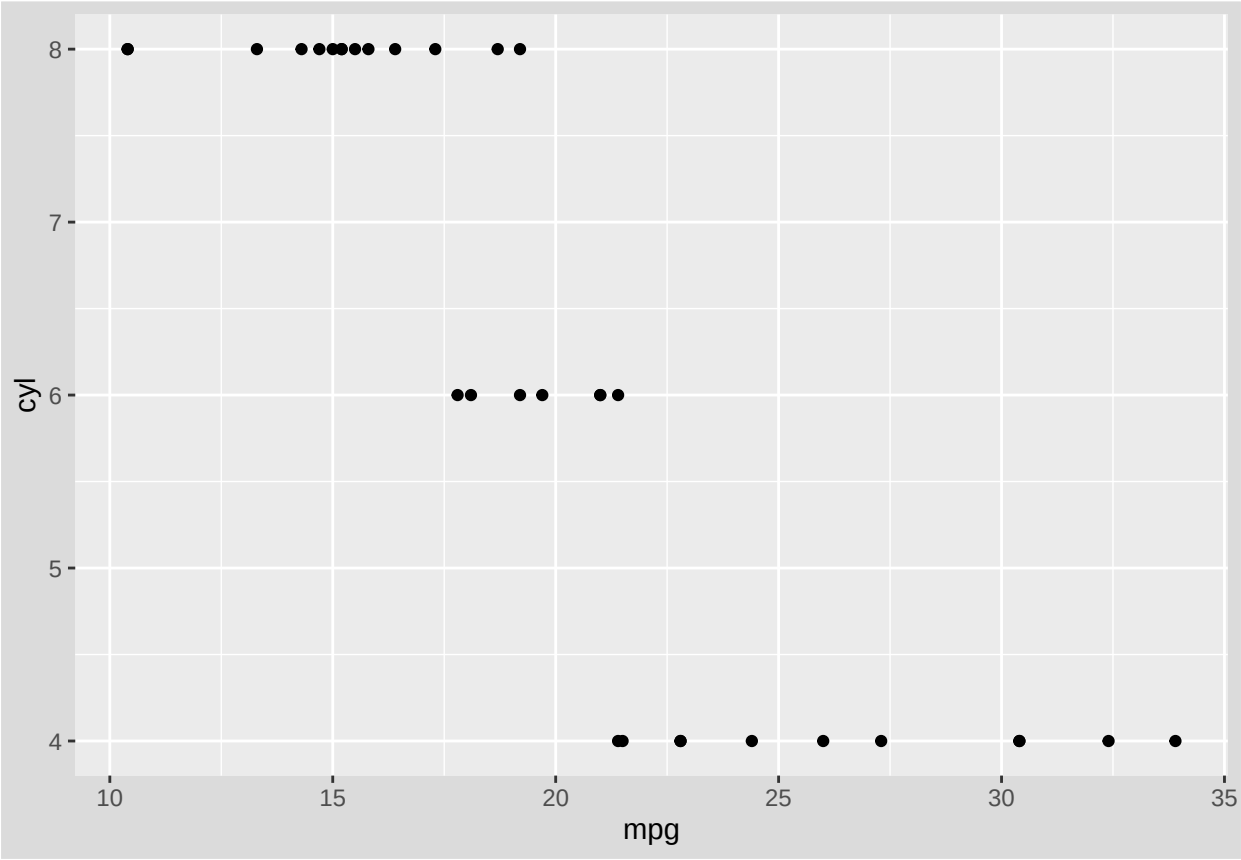
```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_bw() +
  theme(panel.background = element_rect(color = 4, # Color of the border
                                        size = 2)) # Border width
```



Plot background color The plot.background component of the theme function allows modifying the background color of the figure. Set the color inside the fill argument of an element_rect.

Plot background color in ggplot2

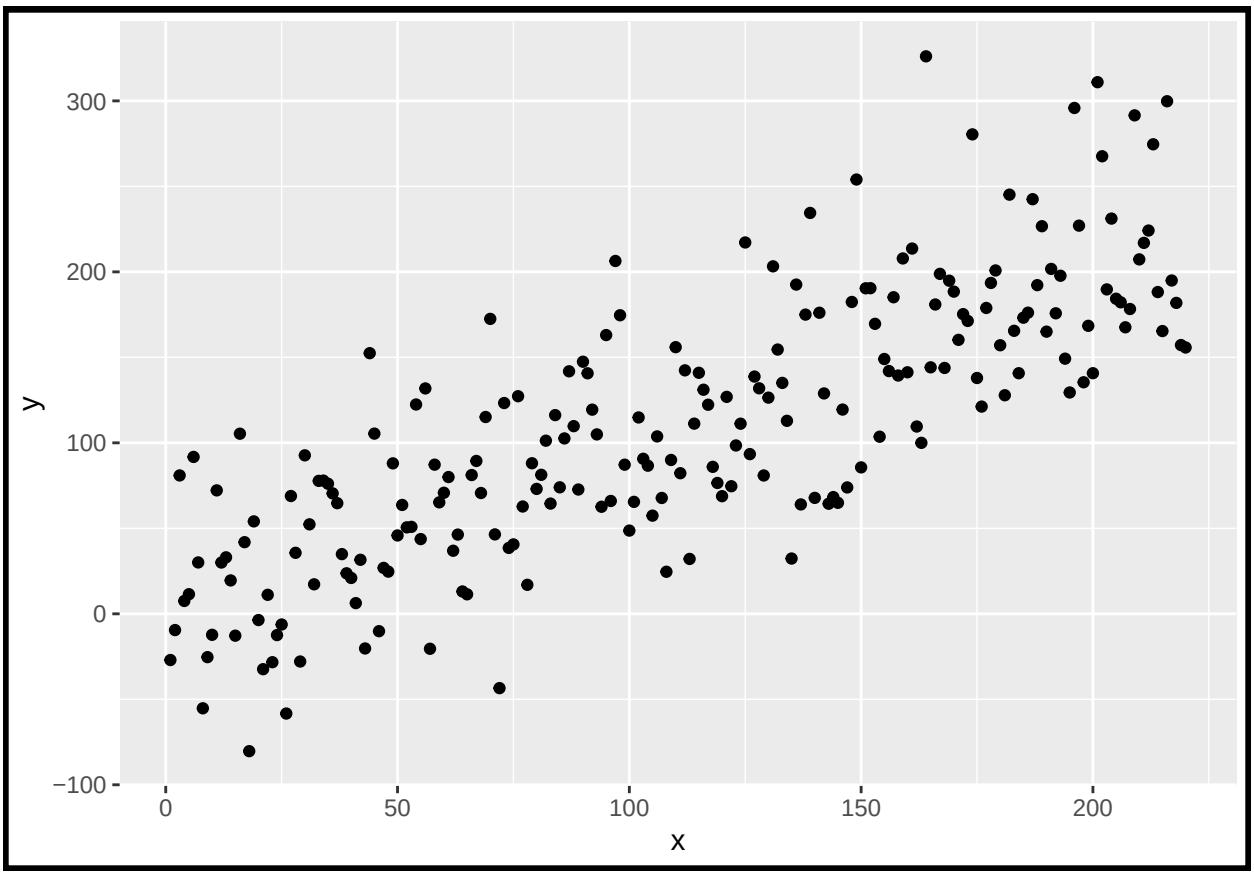
```
library(ggplot2)

ggplot(data = mtcars, aes(x = mpg, y = cyl))+
  geom_point() +
  theme(plot.background = element_rect(fill = "gray86")) # Background color of the plot
```



Plot border color Plot border color in ggplot

You can also set a border color for the whole figure. Just pass an element_rect to the plot.background component of the theme function and modify the color and the width of the border with the arguments color and size, respectively.

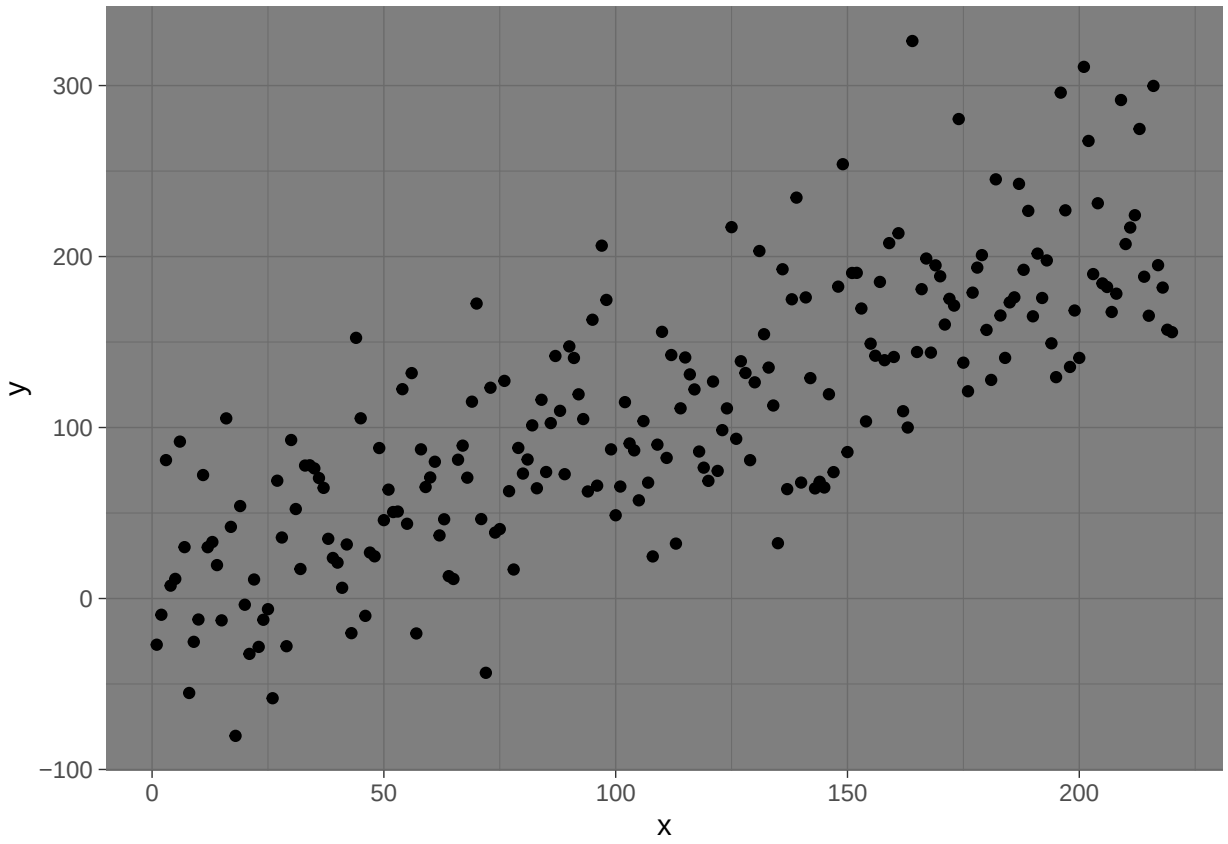
```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme(plot.background = element_rect(color = "black", # Border color
                                       size = 2))       # Border width
```



Changing the colors with themes It is worth mentioning that there are lots of ggplot themes available that provide different background colors.

In this example we are setting the theme_dark, which is one of the in-built ggplot2 themes.

```
library(ggplot2)

ggplot(data = df, aes(x = x, y = y)) +
  geom_point() +
  theme_dark()
```



Note that the default theme is theme_grey.

Change the color of a plot with a ggplot theme

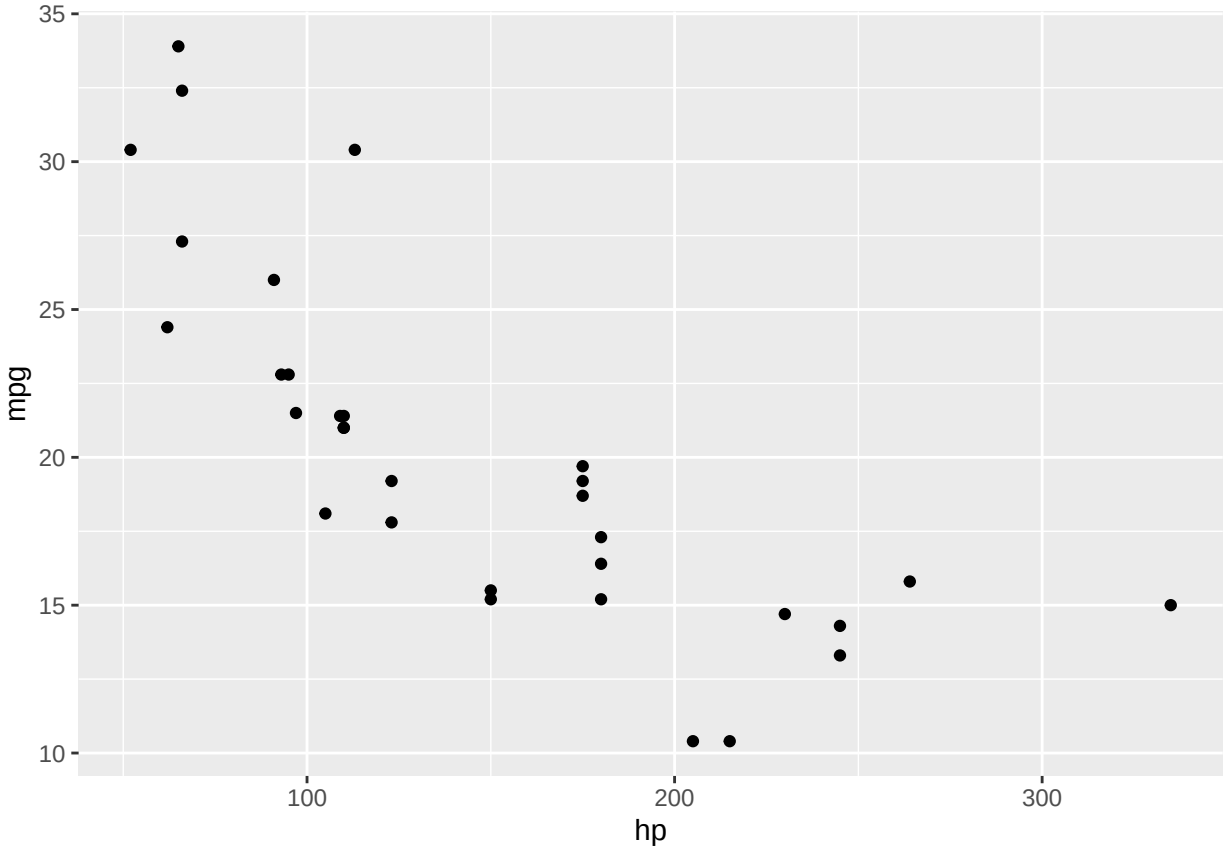
Color picker Use the color pickers to change the panel color (left) and the plot color (right), or to generate random colors pressing the blue button. Then you can copy the colors and use them in your plots.

#EBEBEB #FFFFFF Generate random

Grid customization in ggplot2

Grid customization Major grid Minor grid Custom grid breaks Remove grids By default, ggplot2 creates a major and a minor white grid as shown in the following figure.

```
library(ggplot2)
ggplot(data = mtcars, aes(x = hp, y = mpg))+
  geom_point()
```



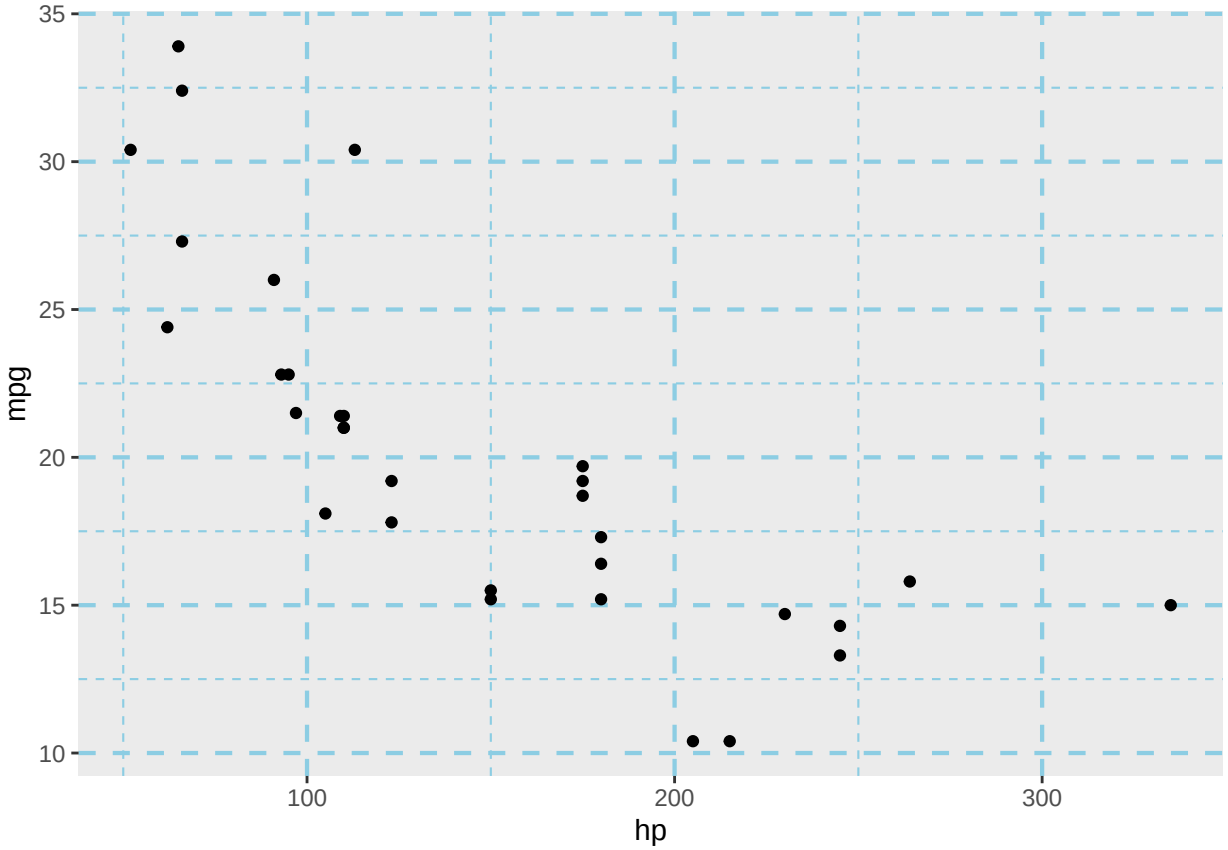
Default ggplot2 grid

Grid customization Grid customization in ggplot2

The grid aesthetics can be set with the panel.grid component of the theme function. Customize the color, line width and line type with the arguments of the element_line function.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid = element_line(color = "#8ccde3",
                                  size = 0.75,
                                  linetype = 2))
```

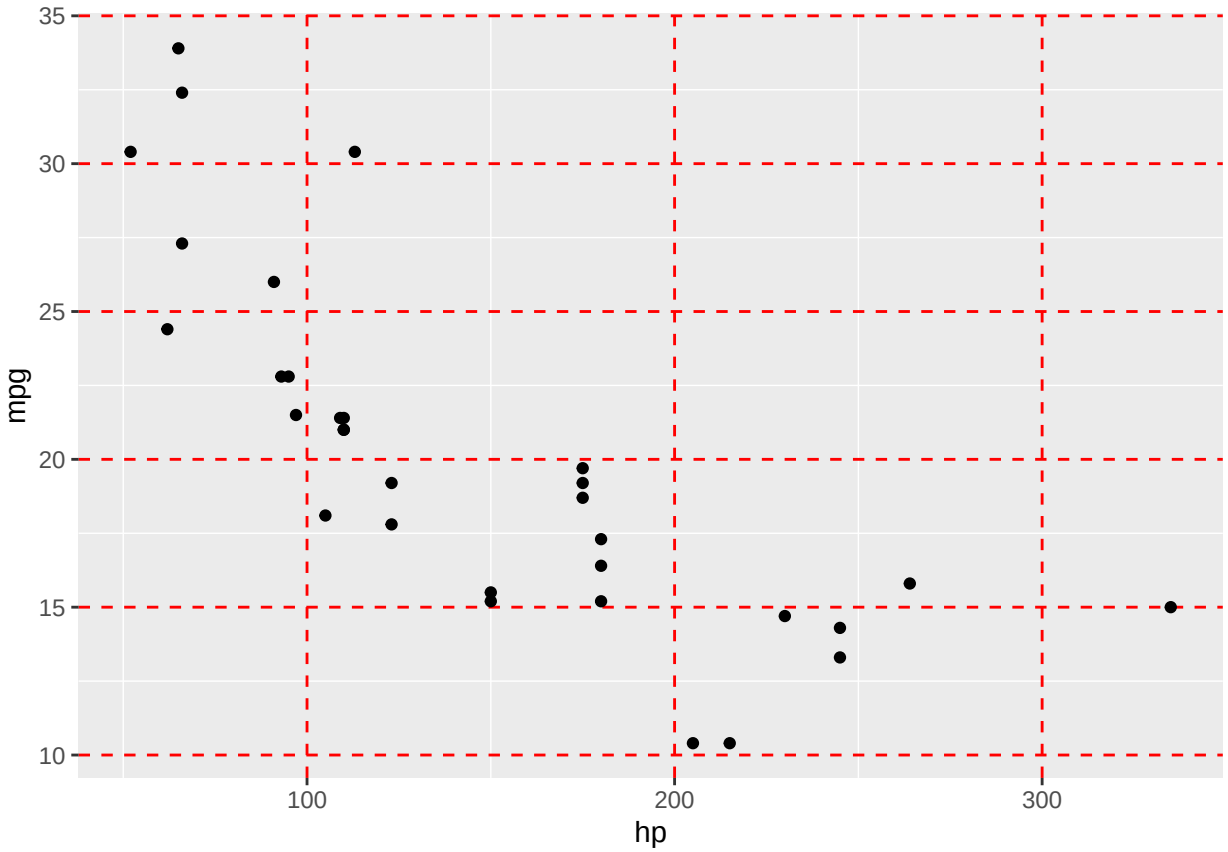
```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
```



You can also customize the major grid and the minor grid individually, as shown in the following sections.

Major grid The panel.grid.major allows you to customize the major grid of the panel.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major = element_line(color = "red",
                                        size = 0.5,
                                        linetype = 2))
```

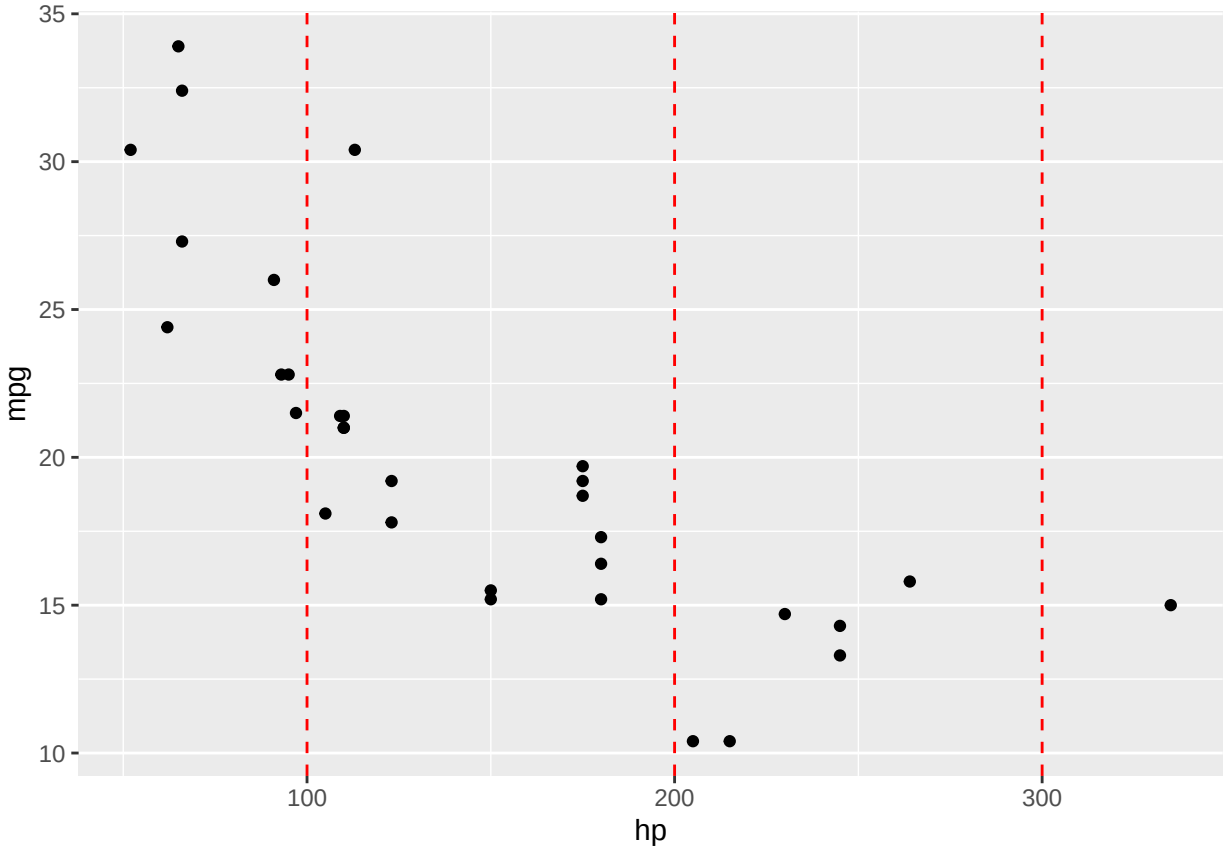


Major grid in ggplot2

Appending .x or .y to the previous component will allow customizing the vertical and horizontal lines of the major grid.

Vertical lines of the major grid

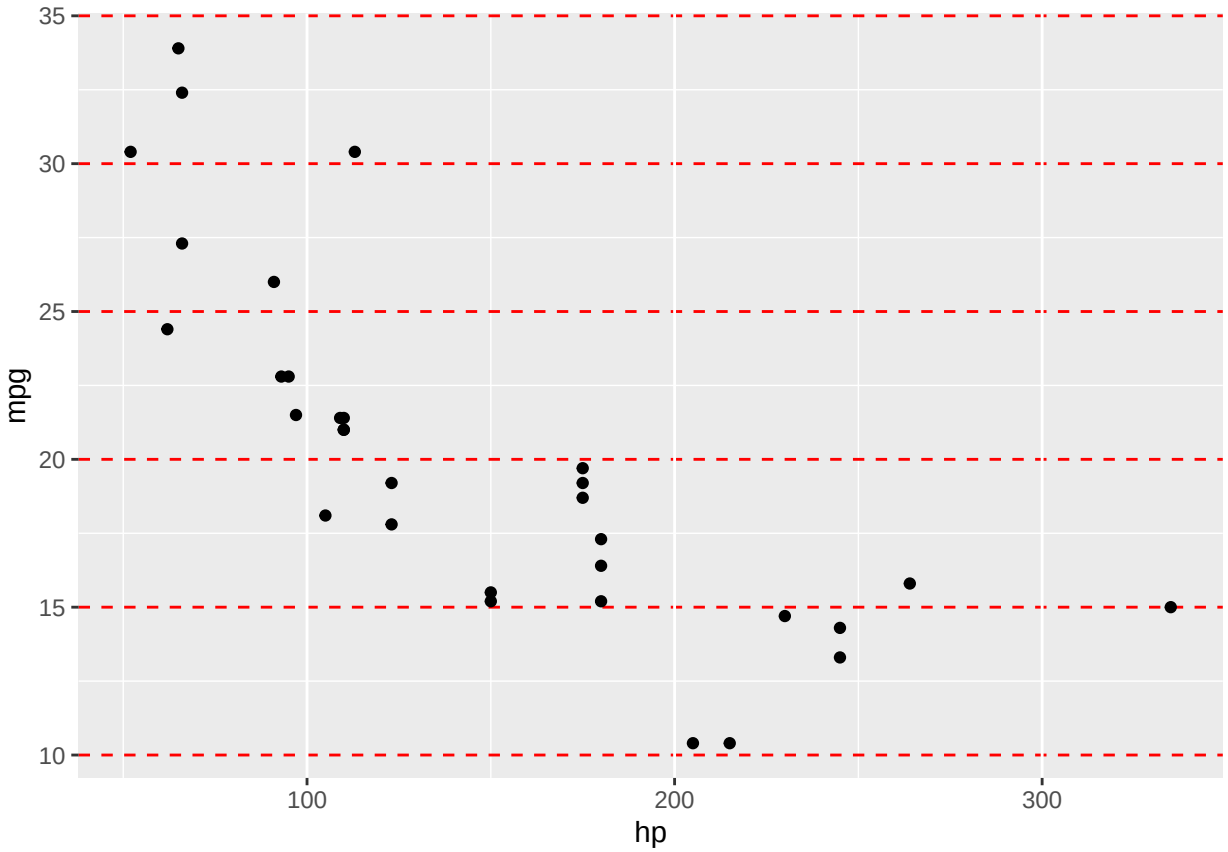
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major.x = element_line(color = "red",
                                          size = 0.5,
                                          linetype = 2))
```



Vertical lines of the major grid in ggplot2

Horizontal lines of the major grid

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major.y = element_line(color = "red",
                                          size = 0.5,
                                          linetype = 2))
```

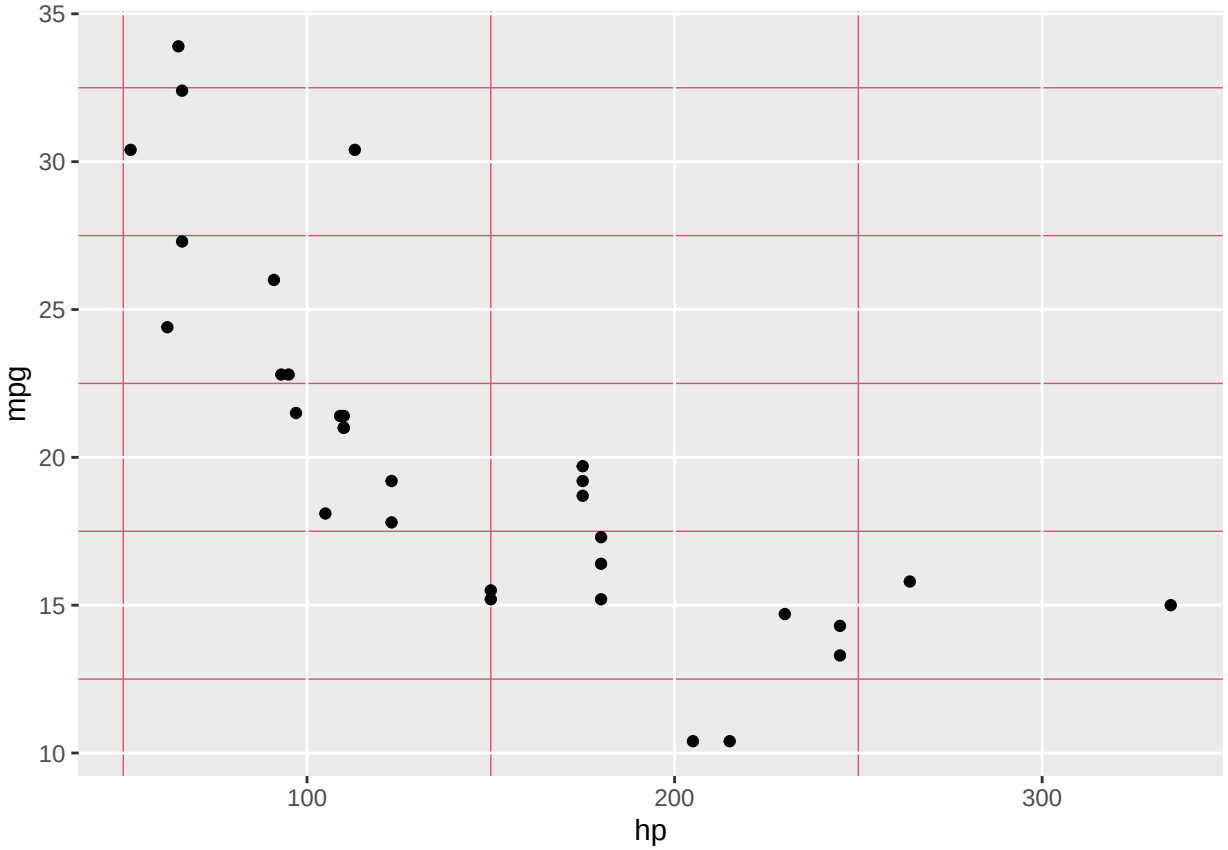


Horizontal lines of the major grid in ggplot

Minor grid Minor grid in ggplot2

The panel.grid.minor allows you to customize the minor grid of the panel.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor = element_line(color = 2,
                                        size = 0.25,
                                        linetype = 1))
```

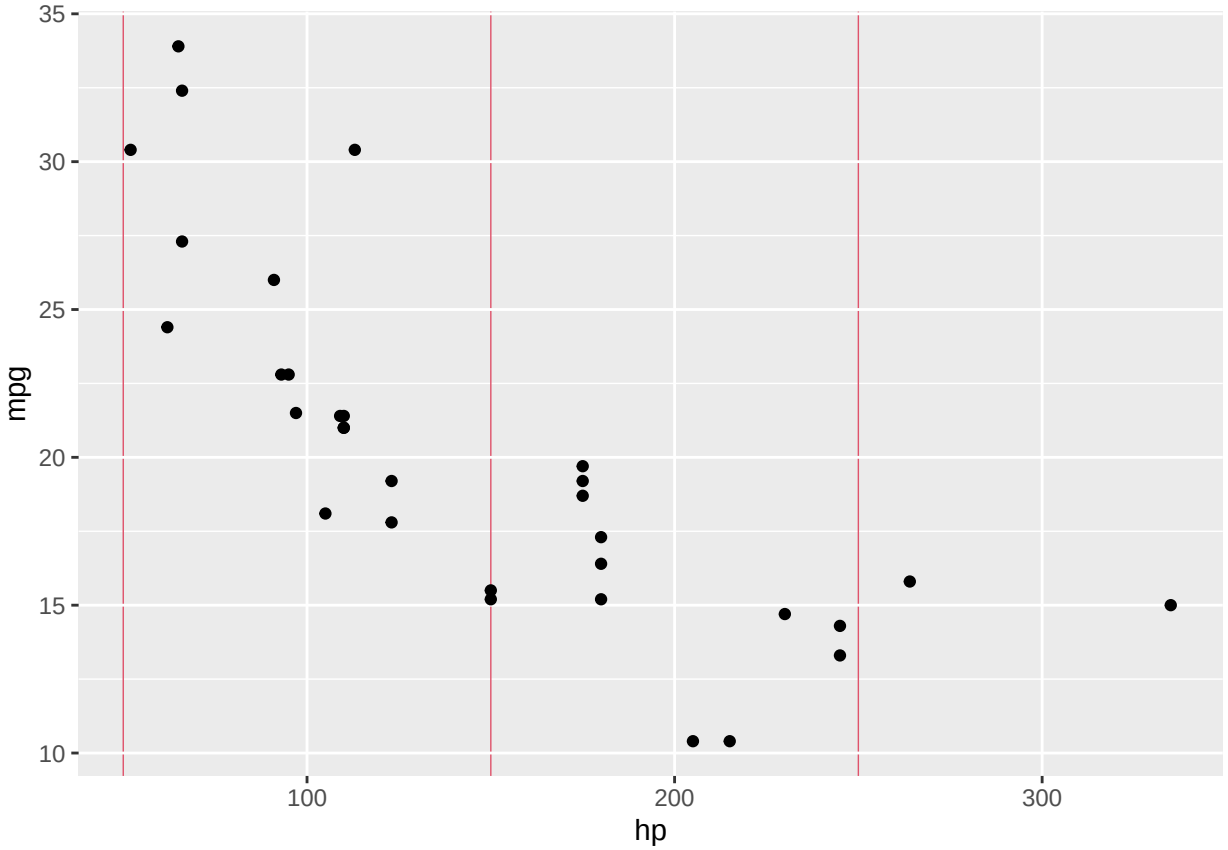


As in the previous section, you can append .x or .y to the theme element.

Minor grid of the X-axis in ggplot2

Vertical lines of the minor grid

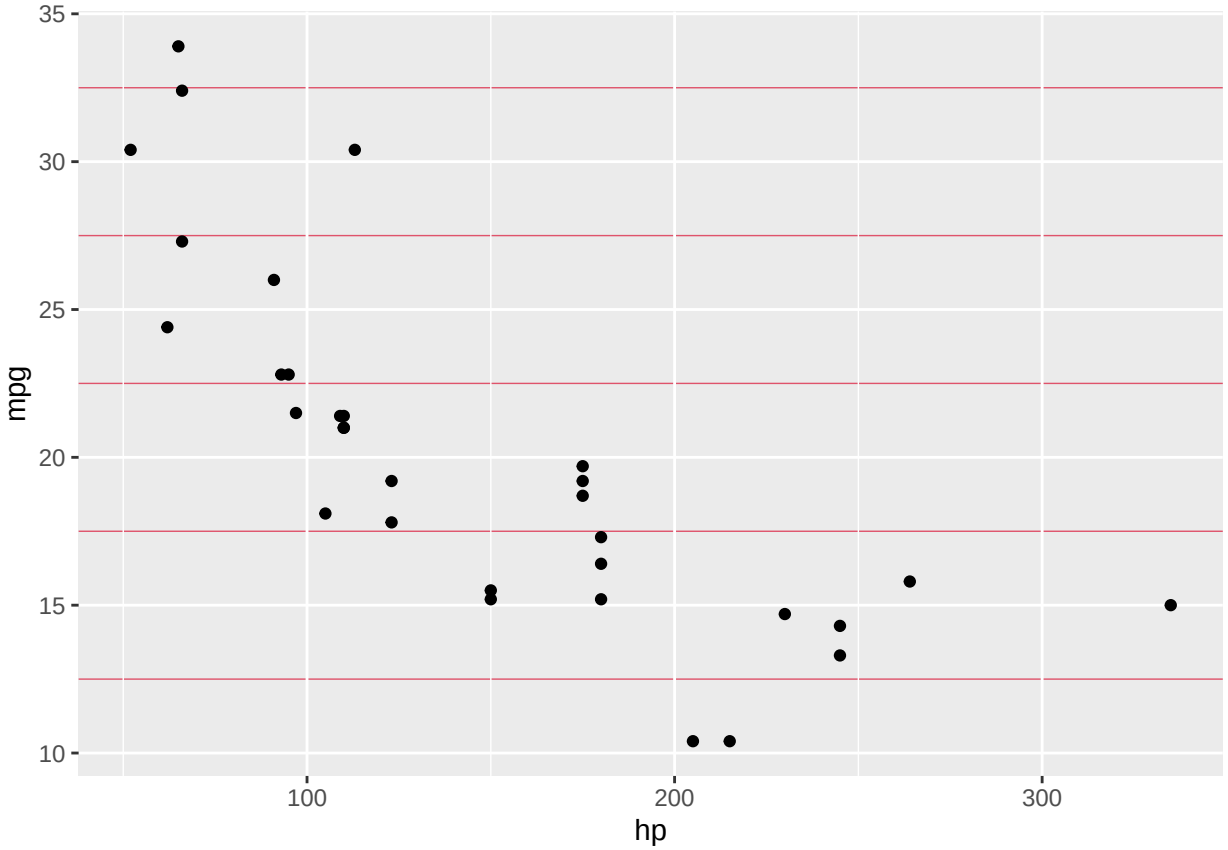
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor.x = element_line(color = 2,
                                          size = 0.25,
                                          linetype = 1))
```



Minor grid of the Y-axis in ggplot2

Horizontal lines of the minor grid

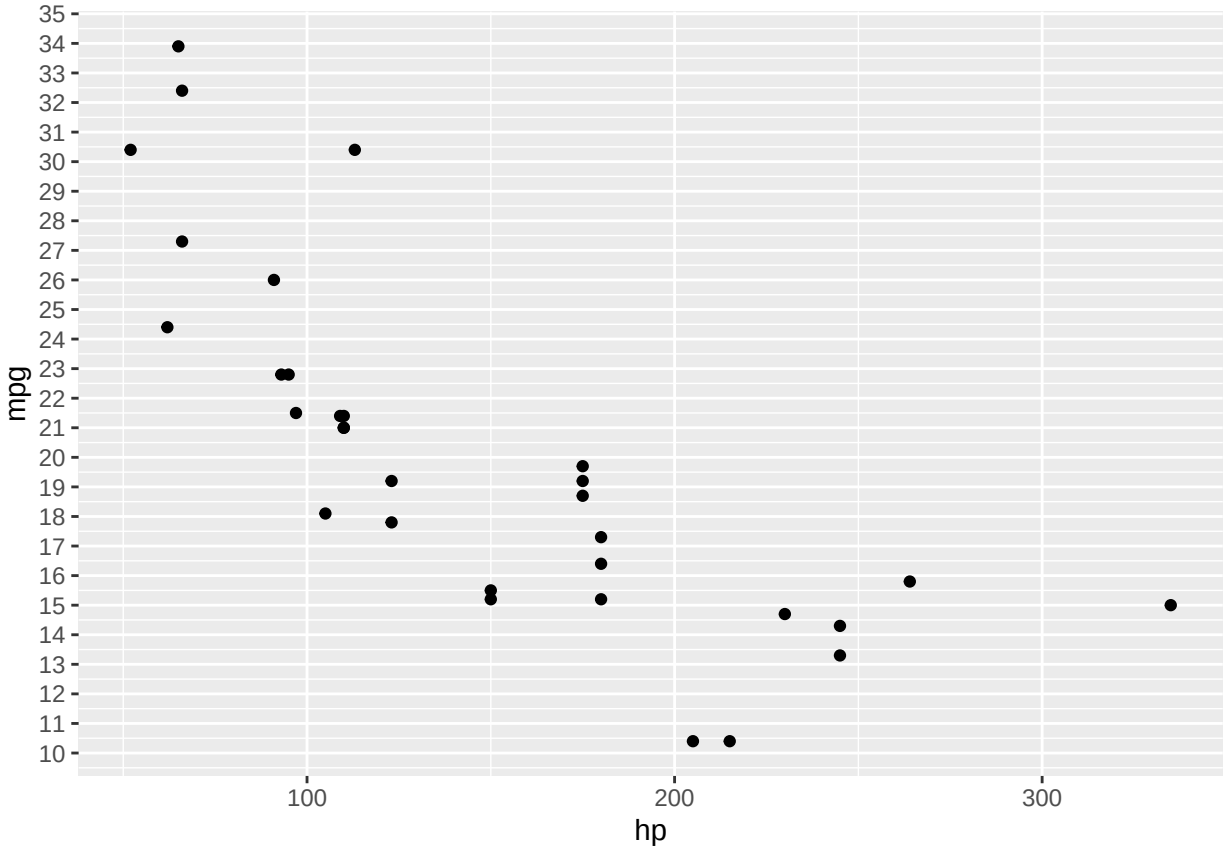
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor.y = element_line(color = 2,
                                          size = 0.25,
                                          linetype = 1))
```



Custom grid breaks The number of grid breaks can be customized in ggplot for each axis with the breaks argument of the scale_(x|y)*continuous or scale*(x|y)_discrete functions, depending on if the variable of the (x|y)-axis is continuous or discrete.

In this example you can customize the breaks of the Y-axis with the scale_y_continuous function as follows.

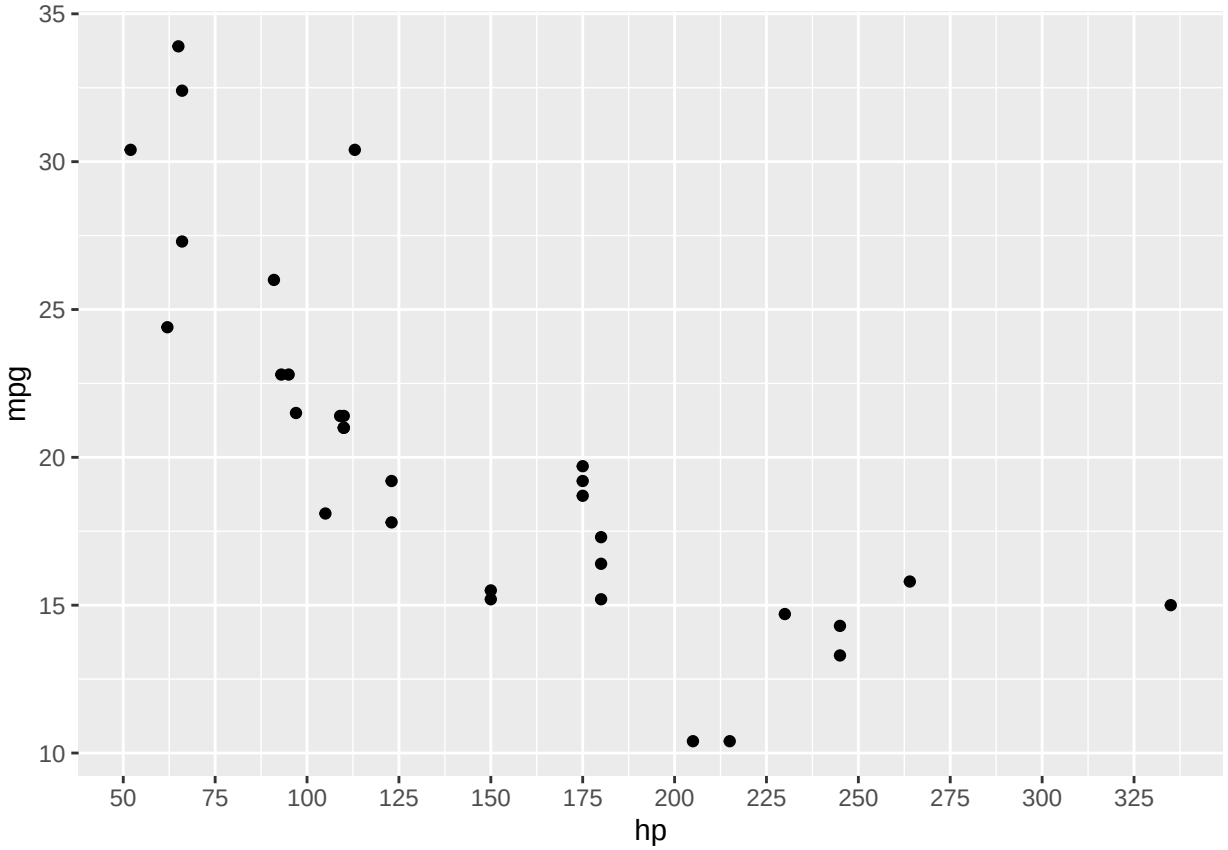
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_y_continuous(breaks = seq(10, 35, by = 1))
```



Custom grid breaks in ggplot2 with scale_y_continuous

As the X-axis is also continuous you can make use of the scale_x_continuous to customize the grid breaks of the X-axis.

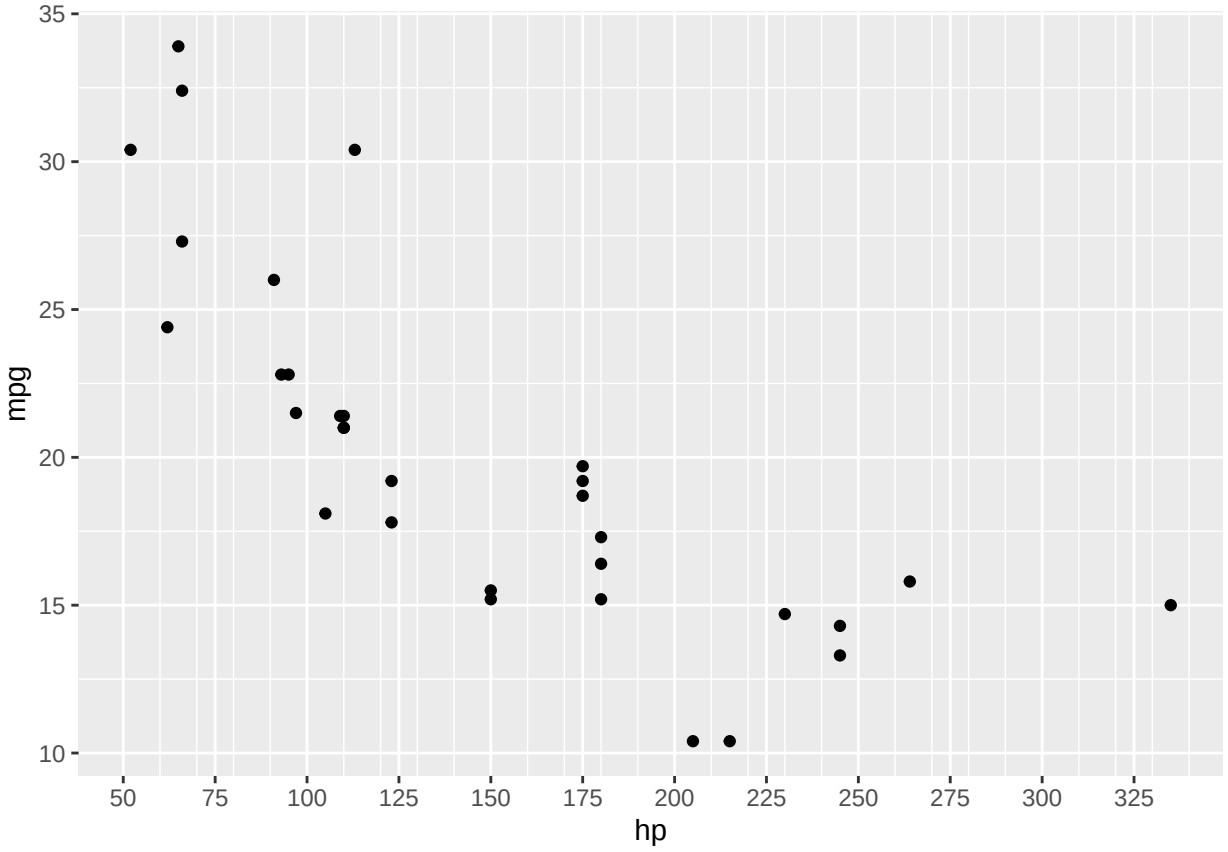
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by = 25))
```



Custom grid breaks X-axis with scale_x_continuous

Set minor breaks with minor_breaks argument.

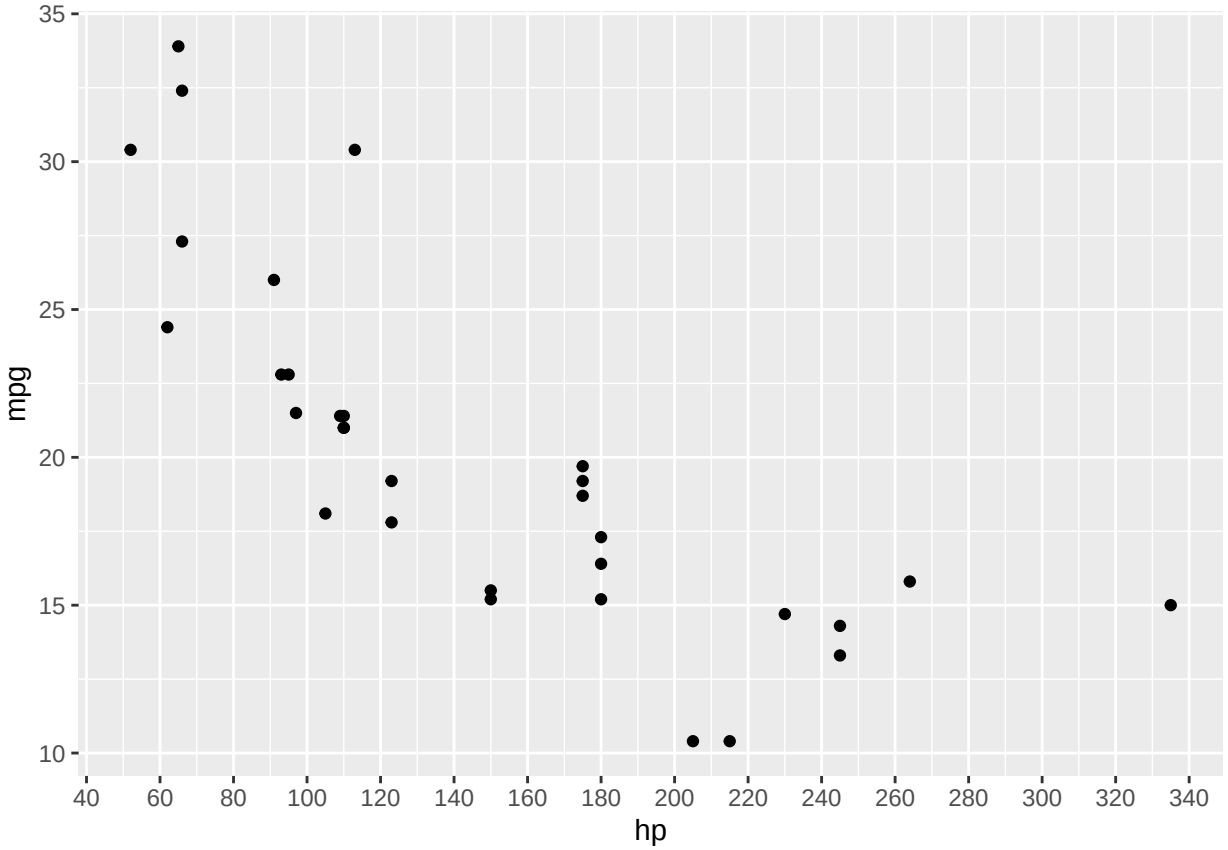
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by = 25),
                     minor_breaks = seq(50, 350, 10))
```



Minor breaks in ggplot2

Set the number of major breaks with n.breaks argument.

```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  scale_x_continuous(n.breaks = 20)
```



The algorithm behind the generation of major breaks may choose a different number than the specified to ensure nice break labels.

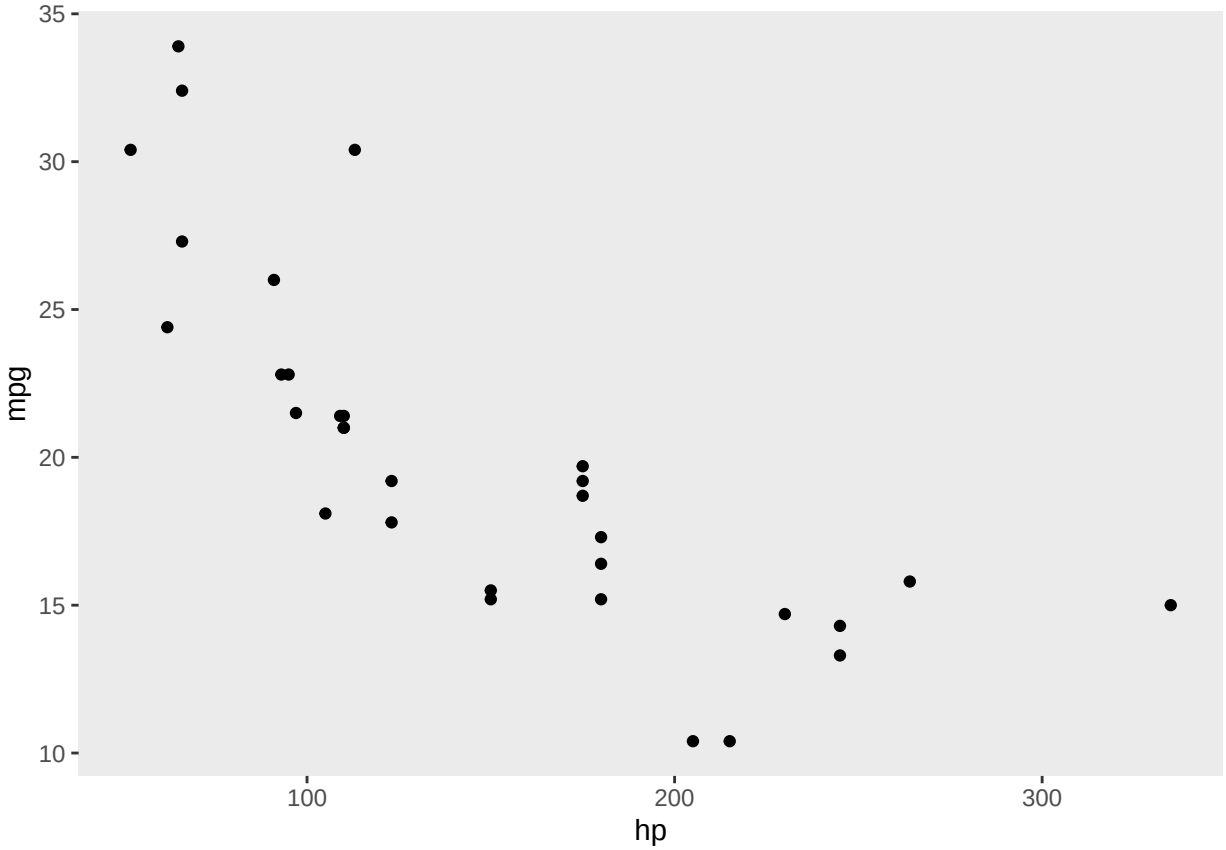
Number of breaks in ggplot

Remove grids The same way you customized each grid (panel.grid, panel.major, panel.major.x, panel.major.y, panel.minor, panel.minor.x, panel.minor.y) you can remove them but setting element_blank instead of element_line.

Remove grid in ggplot2

Remove all grids

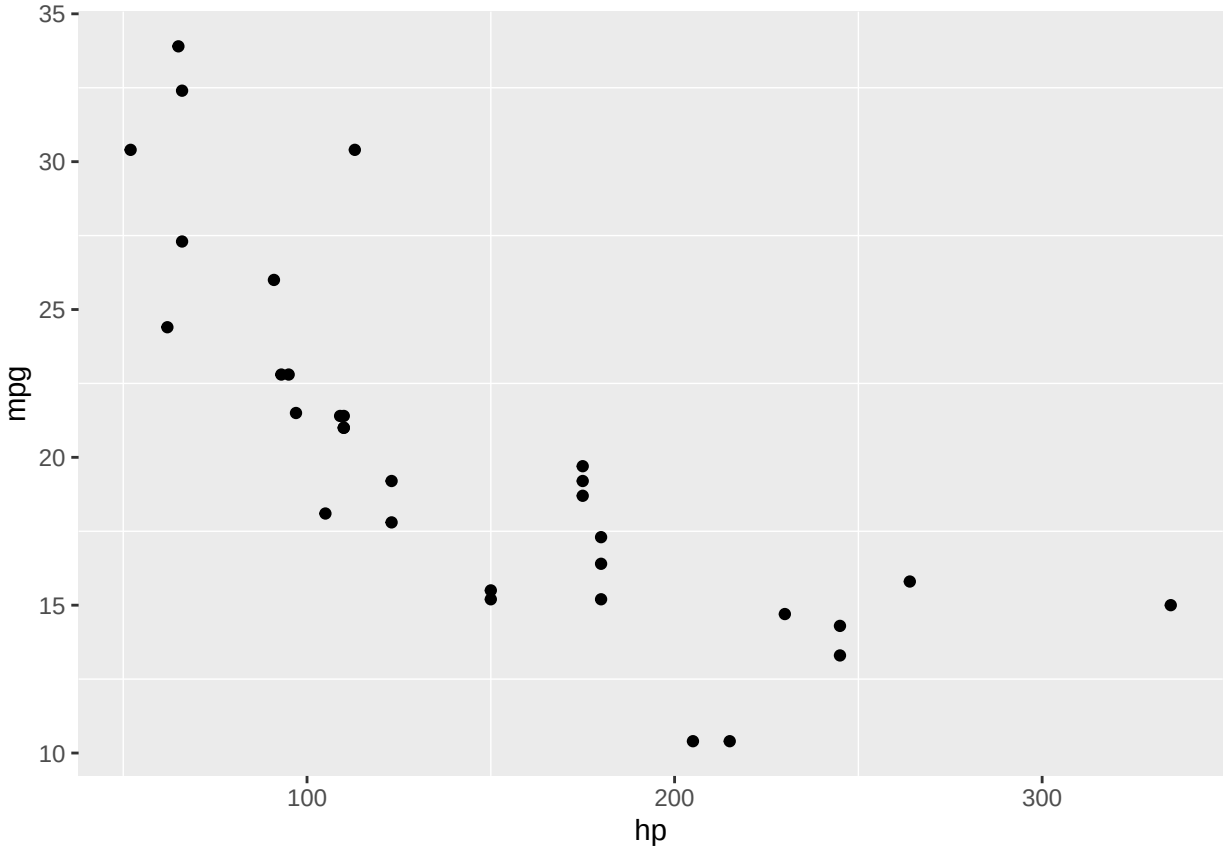
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid = element_blank())
```



Remove the major grid in ggplot2

Remove the major grid

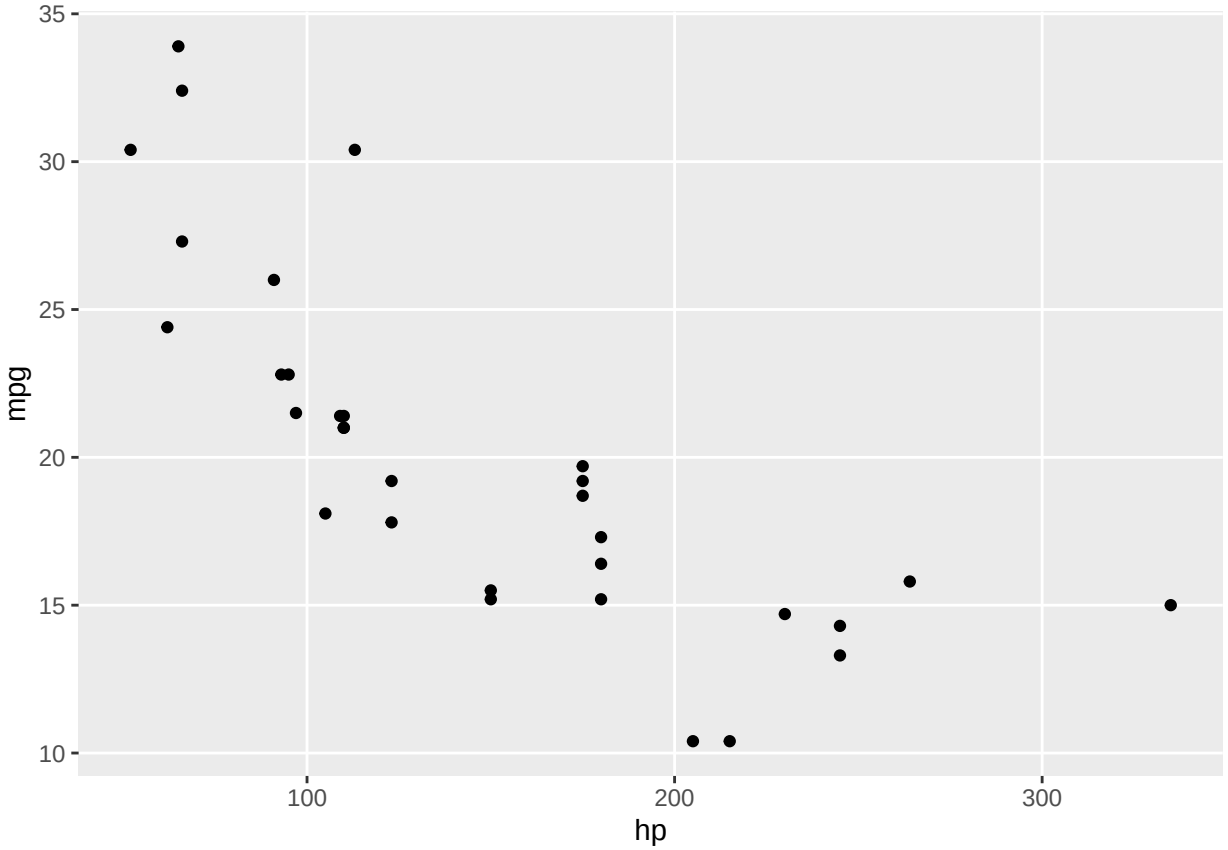
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.major = element_blank())
```



Remove the minor grid in ggplot2

Remove the minor grid

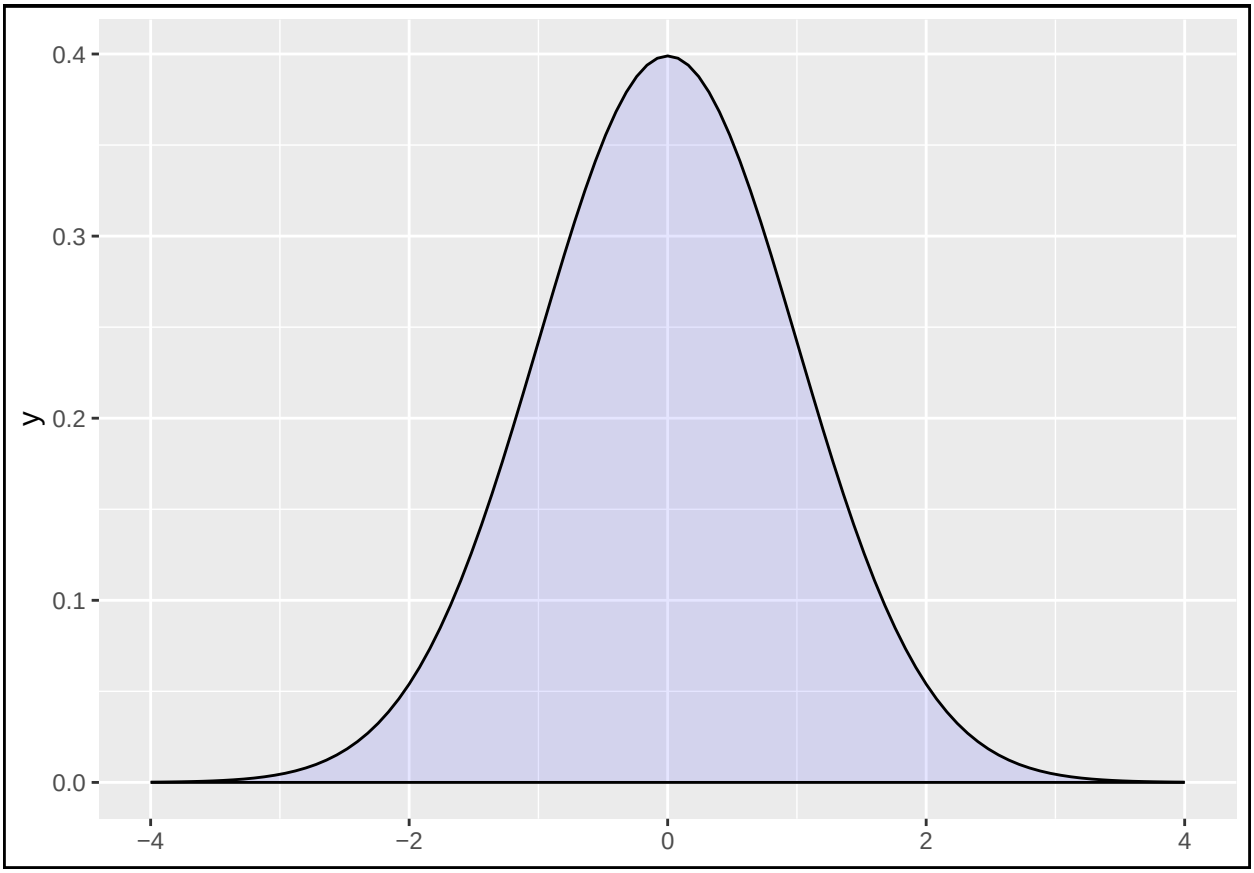
```
library(ggplot2)

ggplot(data = mtcars, aes(x = hp, y = mpg)) +
  geom_point() +
  theme(panel.grid.minor = element_blank())
```



Margins in ggplot2

Increase margins Remove margins The margins of the plots made with ggplot2 will adjust automatically to new layers, e.g. if you add a title. We have added a black box around the sample plot so you can see how margins change.

```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                       size = 1))
```

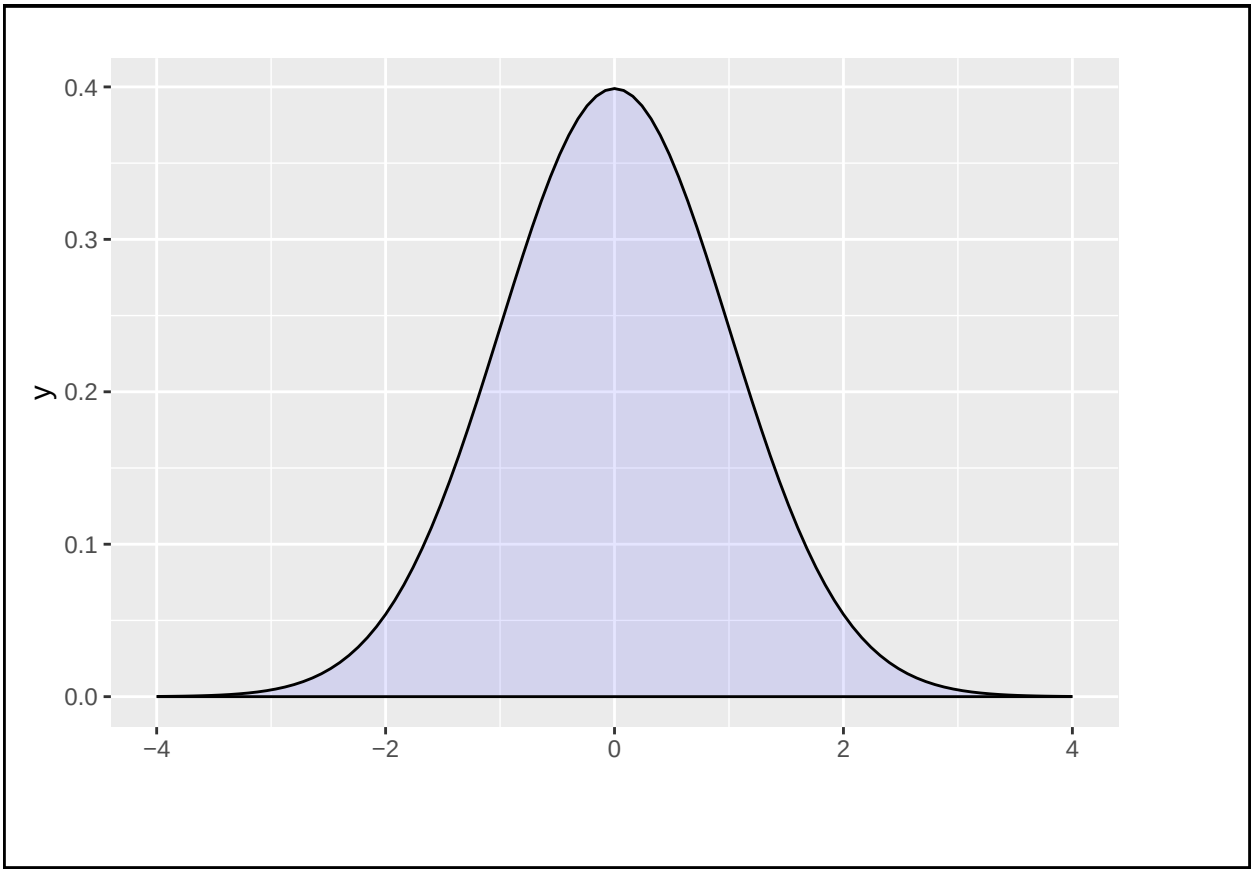


Default margins in ggplot2

Increase margins Increase the ggplot2 margins in points

In order to modify the plot margins set the margin function inside the plot.margin component of the theme function.

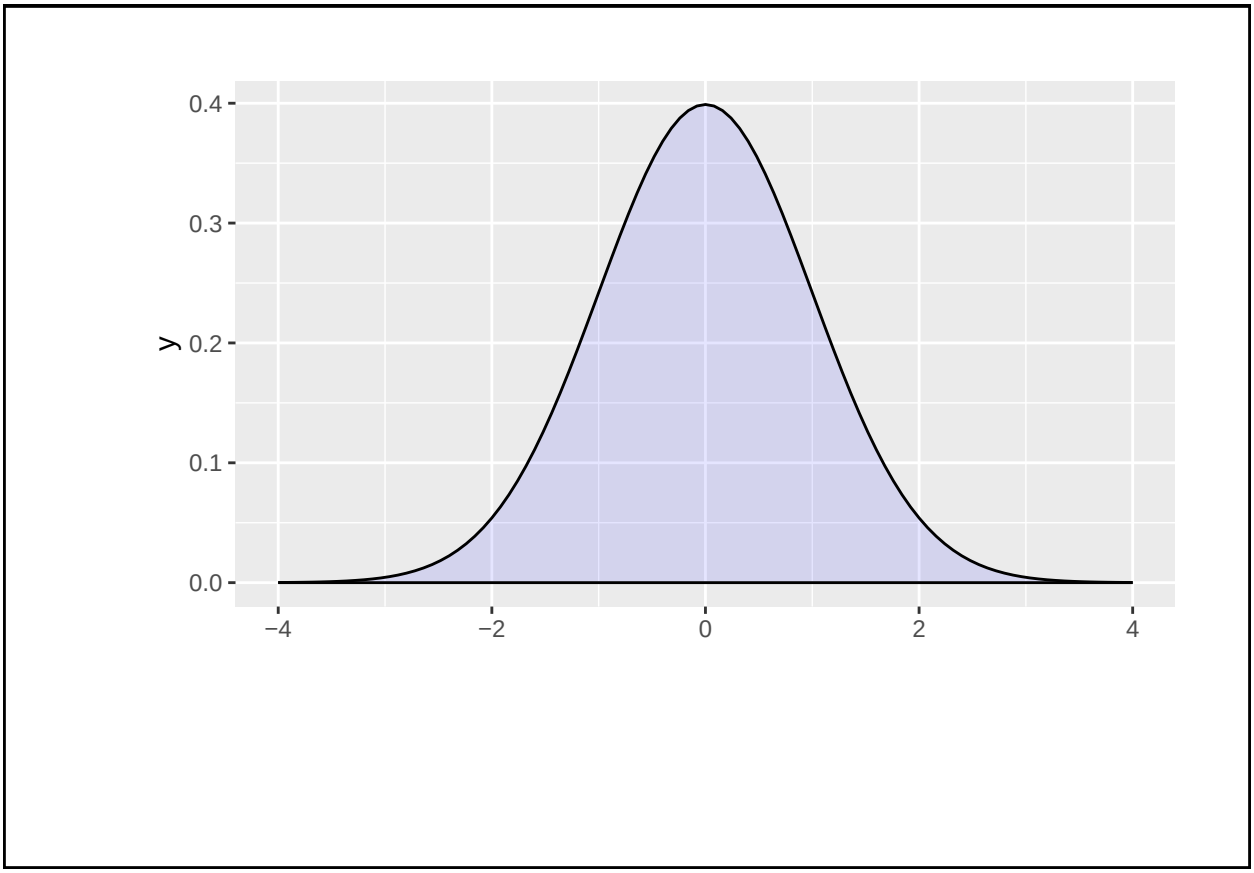
```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                       size = 1),
        plot.margin = margin(t = 20,  # Top margin
                             r = 50,  # Right margin
                             b = 40,  # Bottom margin
                             l = 10)) # Left margin
```



Increase the margins in centimeters in ggplot2

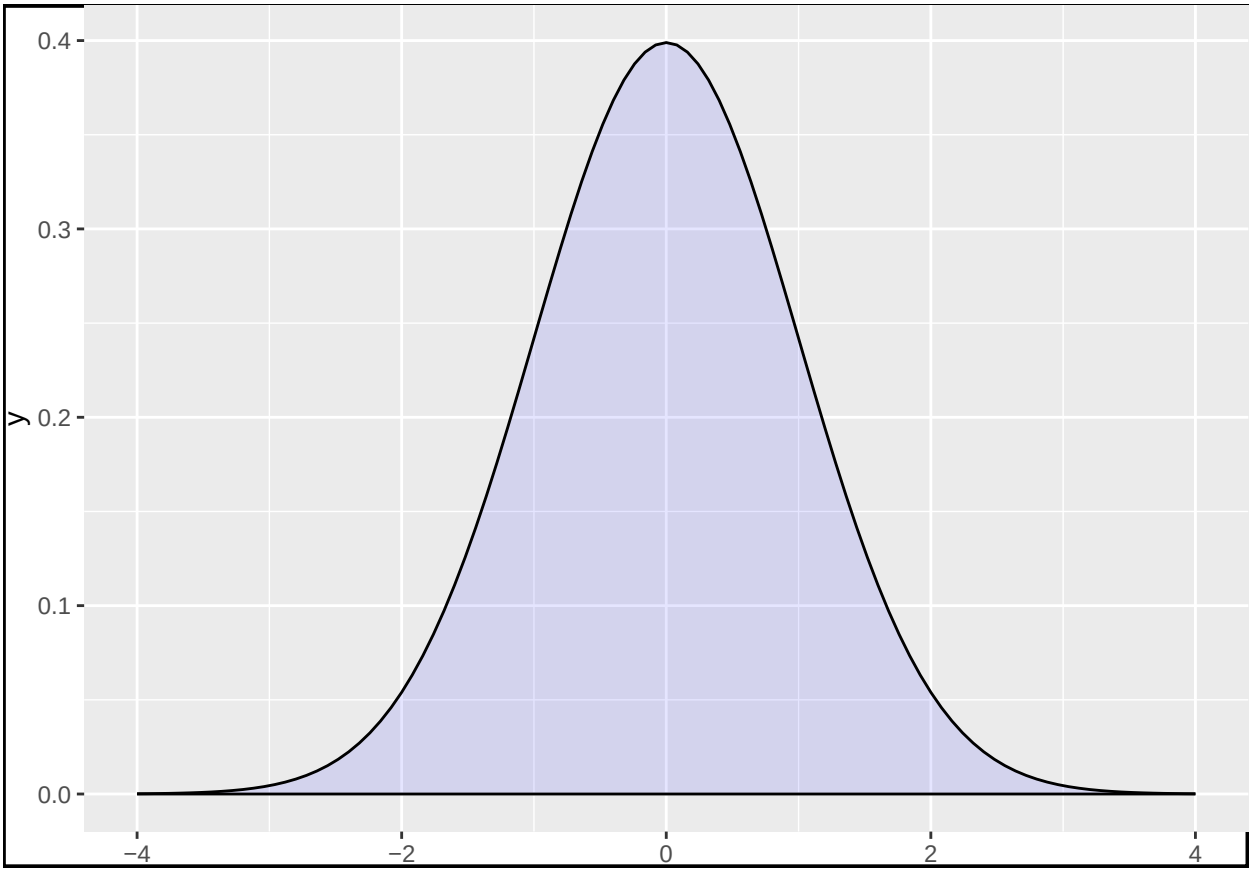
The margins are measured with points (“pt”), but you can use other unit measure in the unit argument, like centimeters. Type ?unit to see all the possible measures.

```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                       size = 1),
        plot.margin = margin(t = 1,  # Top margin
                             r = 1,  # Right margin
                             b = 3,  # Bottom margin
                             l = 2,  # Left margin
                             unit = "cm"))
```



Remove margins To remove the margins set all values to 0. Note that there is still space to fit all the elements of the plot. You can set negative values to reduce more the margins.

```
library(ggplot2)

ggplot() +
  stat_function(fun = dnorm, geom = "density",
                xlim = c(-4, 4),
                fill = rgb(0, 0, 1, 0.1)) +
  theme(plot.background = element_rect(color = 1,
                                       size = 1),
        plot.margin = margin(t = 0,  # Top margin
                             r = 0,  # Right margin
                             b = 0,  # Bottom margin
                             l = 0)) # Left margin
```



Remove the margins in ggplot2

You can remember the order of the arguments of the margin function (t, r, b, l) remembering the word trouble.

Legends in ggplot2

Sample data Adding a legend Change or remove the title of the legend Change or reorder the labels of the legend Change the position of the legend Legend customization Remove the legend Sample data In this tutorial we are going to use the following sample data categorized into two groups.

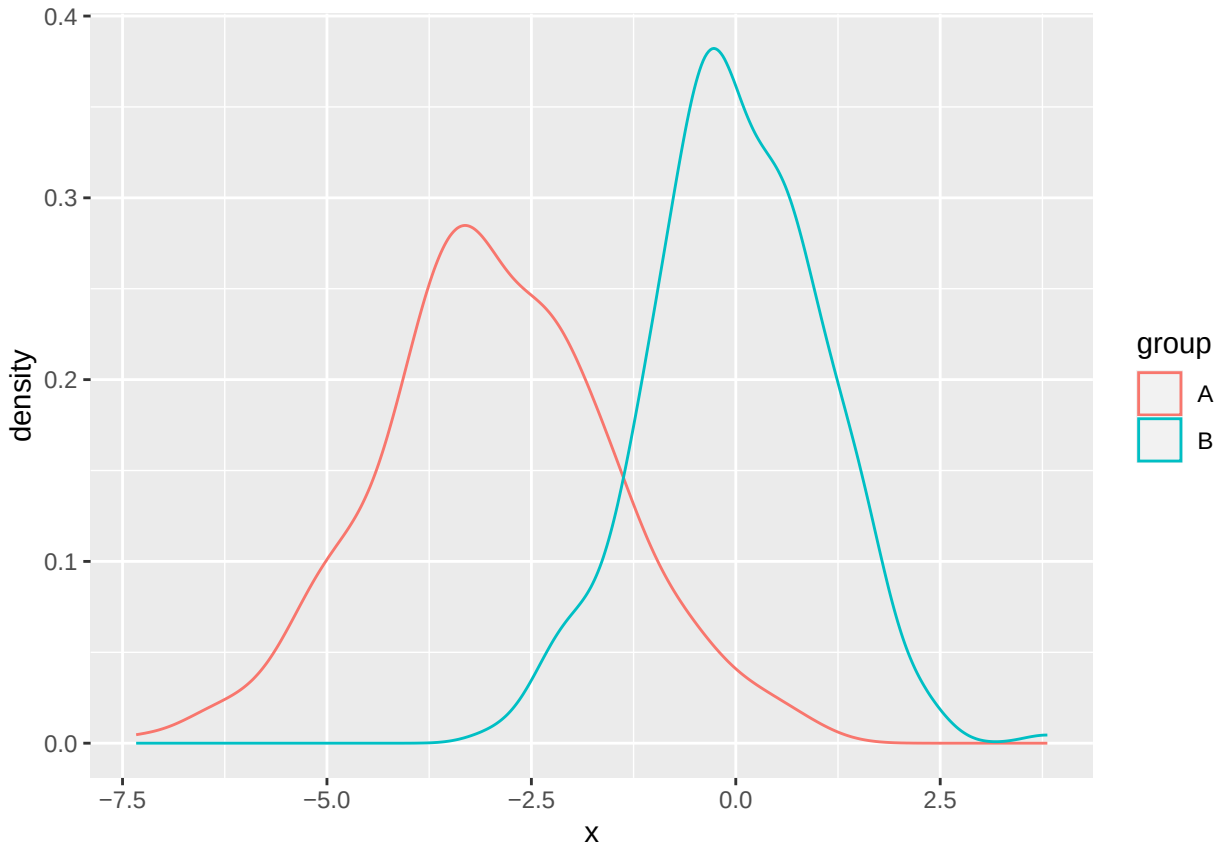
```
# Data
set.seed(1)
df <- data.frame(x = c(rnorm(300, -3, 1.5),
                       rnorm(300, 0, 1)),
                 group = c(rep("A", 300),
                           rep("B", 300)))
```

Adding a legend If you want to add a legend to a ggplot2 chart you will need to pass a categorical (or numerical) variable to color, fill, shape or alpha inside aes. Depending on which argument you use to pass the data and your specific case the output will be different.

Color

Passing the categorical variable to color inside aes the data will be plotted based on groups and an automatic legend will you up.

```r
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, color = group)) +
  geom_density(alpha = 0.5)
```



Plot with default legend in ggplot2

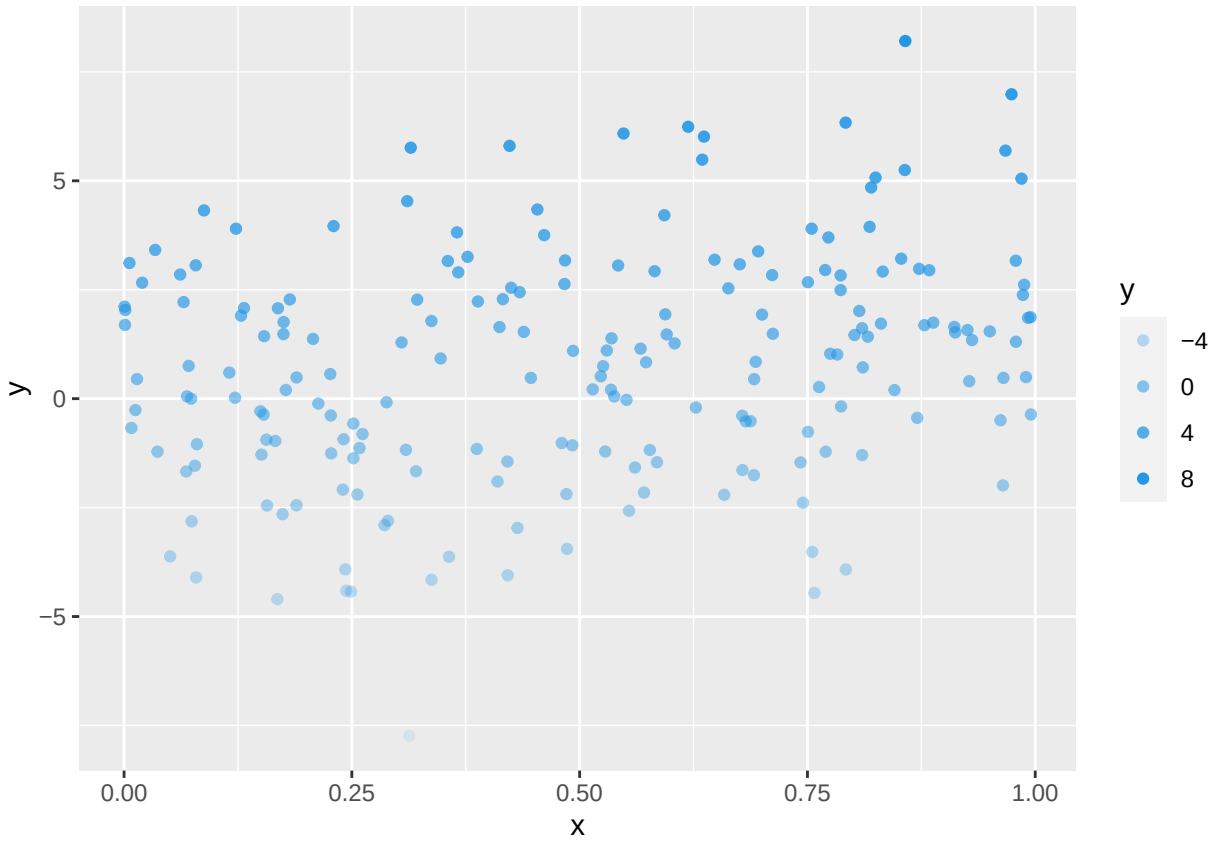
Note that this is the argument you should use if you want to create a plot that doesn't admit a fill color, such as a scatter plot. You can pass a categorical or a numerical variable.

```r
# install.packages("ggplot2")
library(ggplot2)

# Data
x <- runif(200)
y <- 3 * x ^ 2 + rnorm(length(x), sd = 2.5)
group <- ifelse(x < 0.5, "A", "B")
df2 <- data.frame(x = x, y = y, group = group)

ggplot(df2, aes(x = x, y = y, color = group)) +
  geom_point()
```

Scatter plot with legend in ggplot2

Alpha

```
# install.packages("ggplot2")
library(ggplot2)

# Data
x <- runif(200)
y <- 3 * x ^ 2 + rnorm(length(x), sd = 2.5)
group <- ifelse(x < 0.5, "A", "B")
df2 <- data.frame(x = x, y = y)

ggplot(df2, aes(x = x, y = y, alpha = y)) +
  geom_point(colour = 4)
```

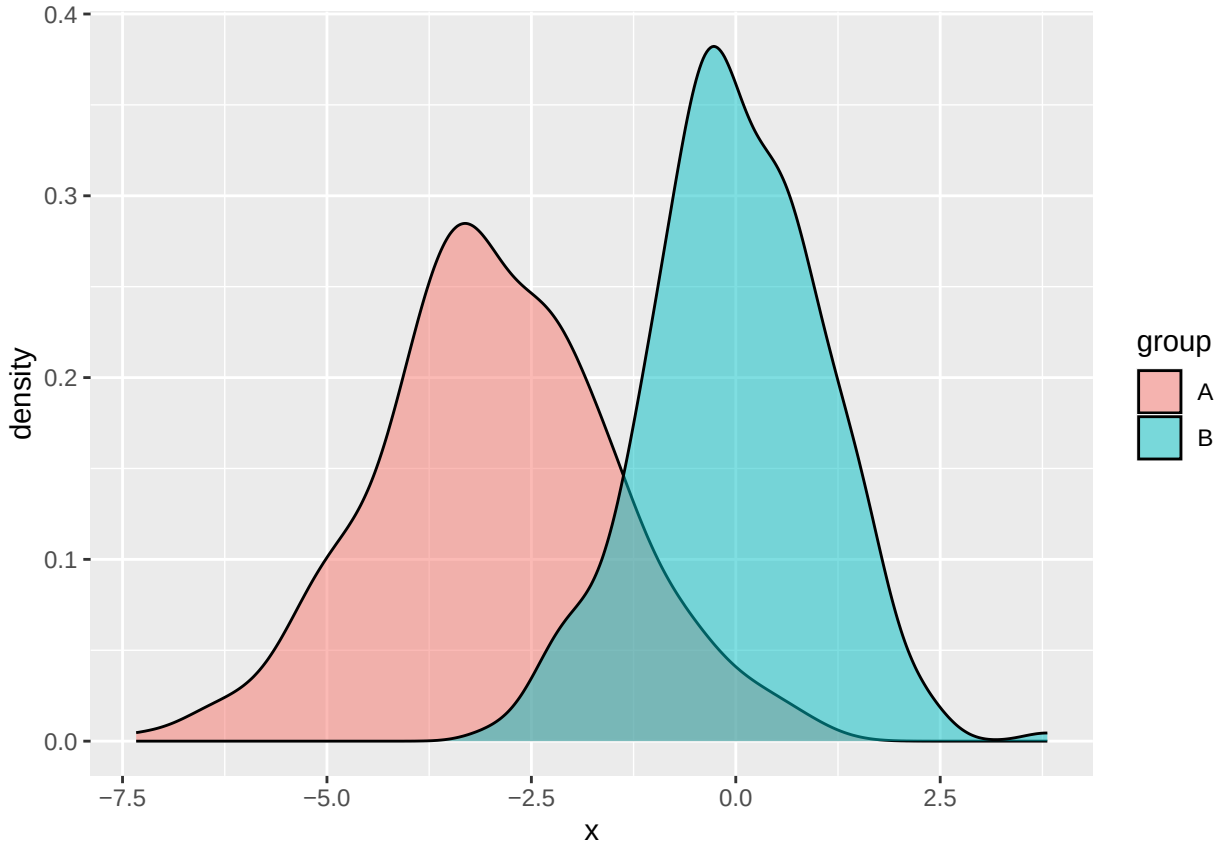


Scatter plot with legend based on transparency in ggplot2

Fill

If the plot you are creating allows adding a fill color you can use the fill argument inside aes, so the boxes of the legend will be colored.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5)
```

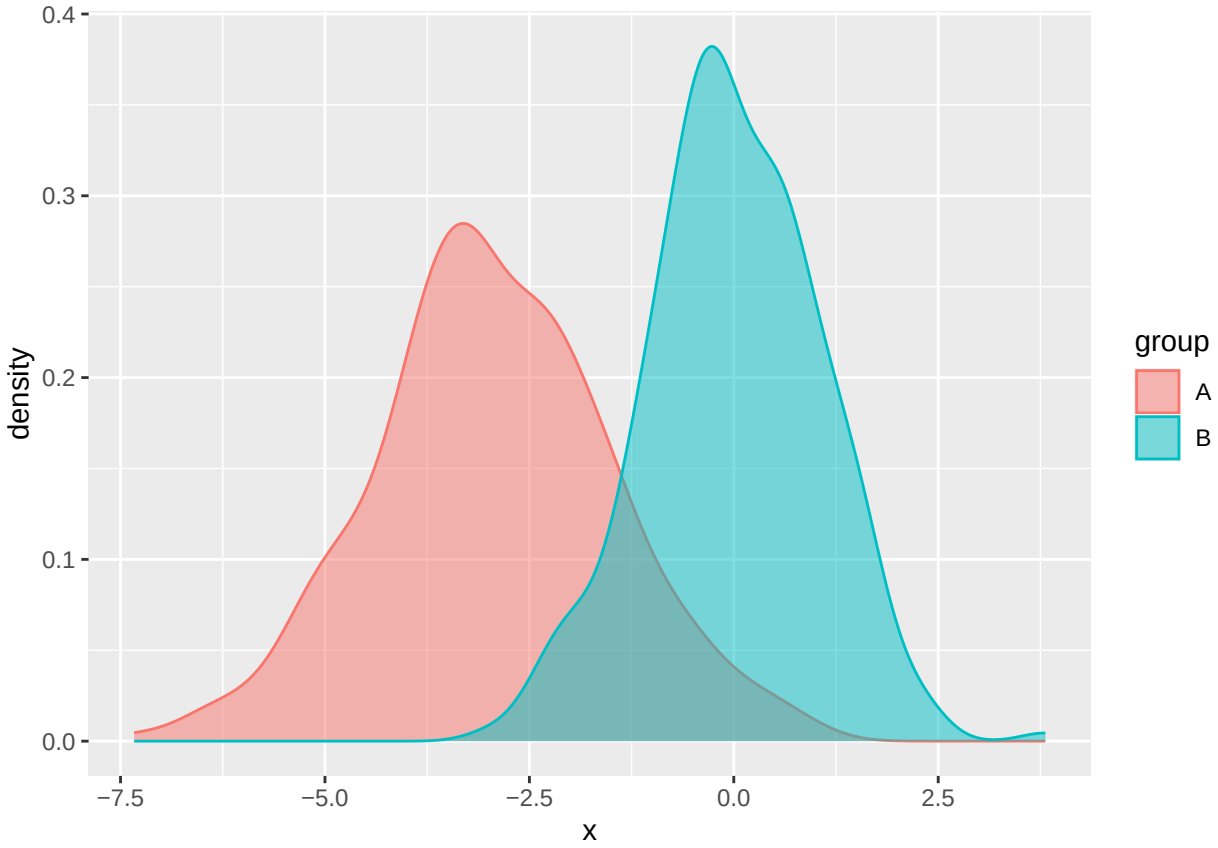


Legend by fill color in ggplot2

Color and fill at the same time

In the previous scenario you can also use both arguments (fill and color), so the legend will be a box filled with the correspoding color and the border will also have the corresponding color of the group.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group, color = group)) +
  geom_density(alpha = 0.5)
```



Fill and color ggplot2 legend

If you want to change the fill or border colors of the chart (and automatically of the legend) take a look to other tutorials, such as the density plot by group tutorial.

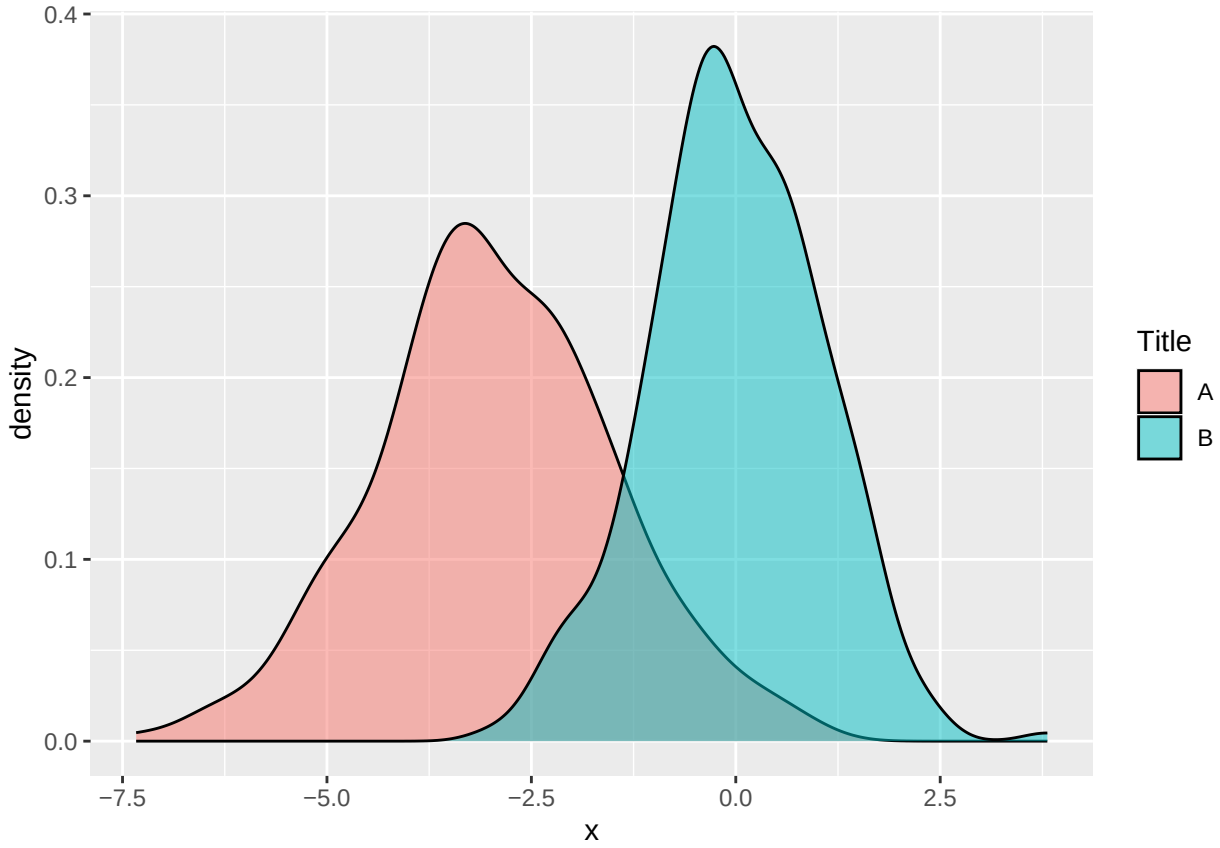
Change or remove the title of the legend There are several ways to change the title of the legend of your plot. Note that the chosen option will depend on your chart type and your preferences.

Changing the title of the legend in ggplot2 with guides

Option 1

The first option is using the guides function and passing guide_legend to fill or to color, depending on your plot.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  guides(fill = guide_legend(title = "Title"))
```

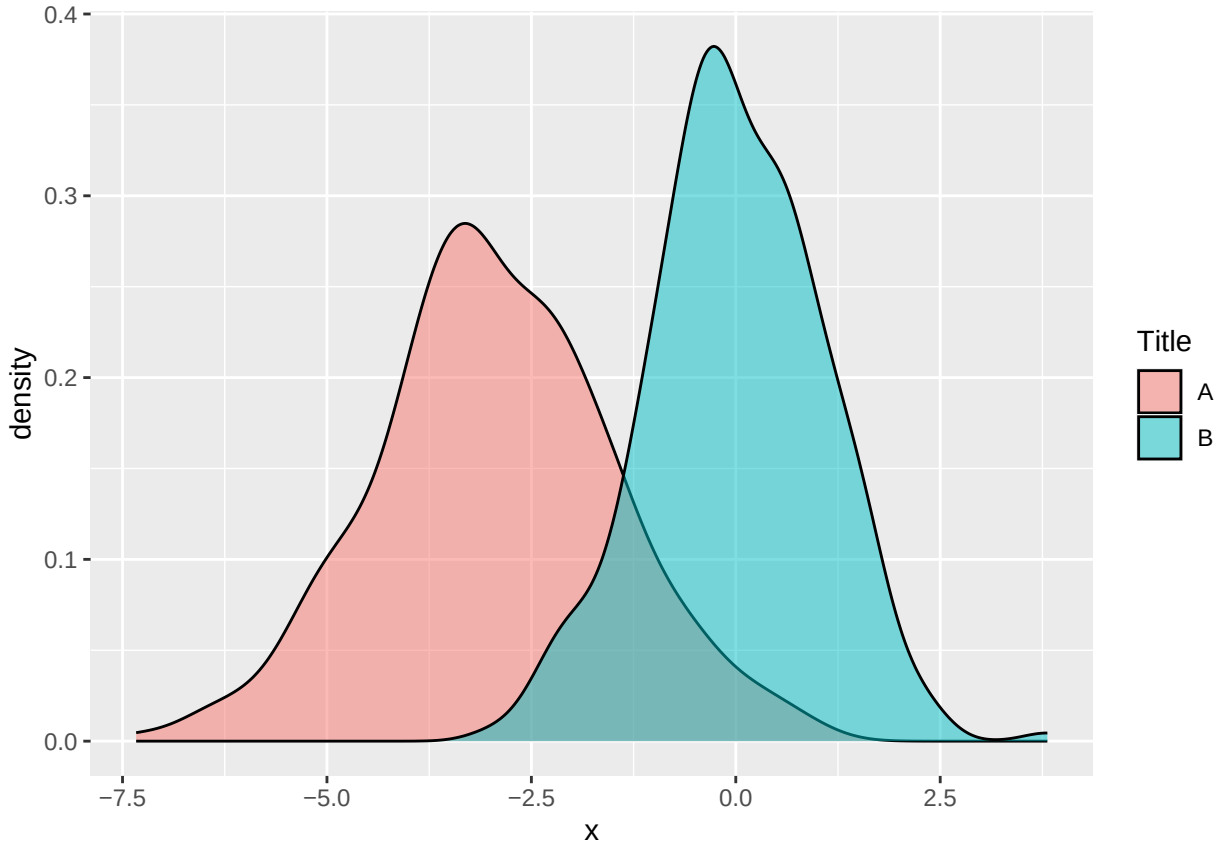


Modifying the title of the legend in ggplot2 with the labs function

Option 2

The second option is passing the new title to the fill or color argument of the labs function.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  labs(fill = "Title")
```

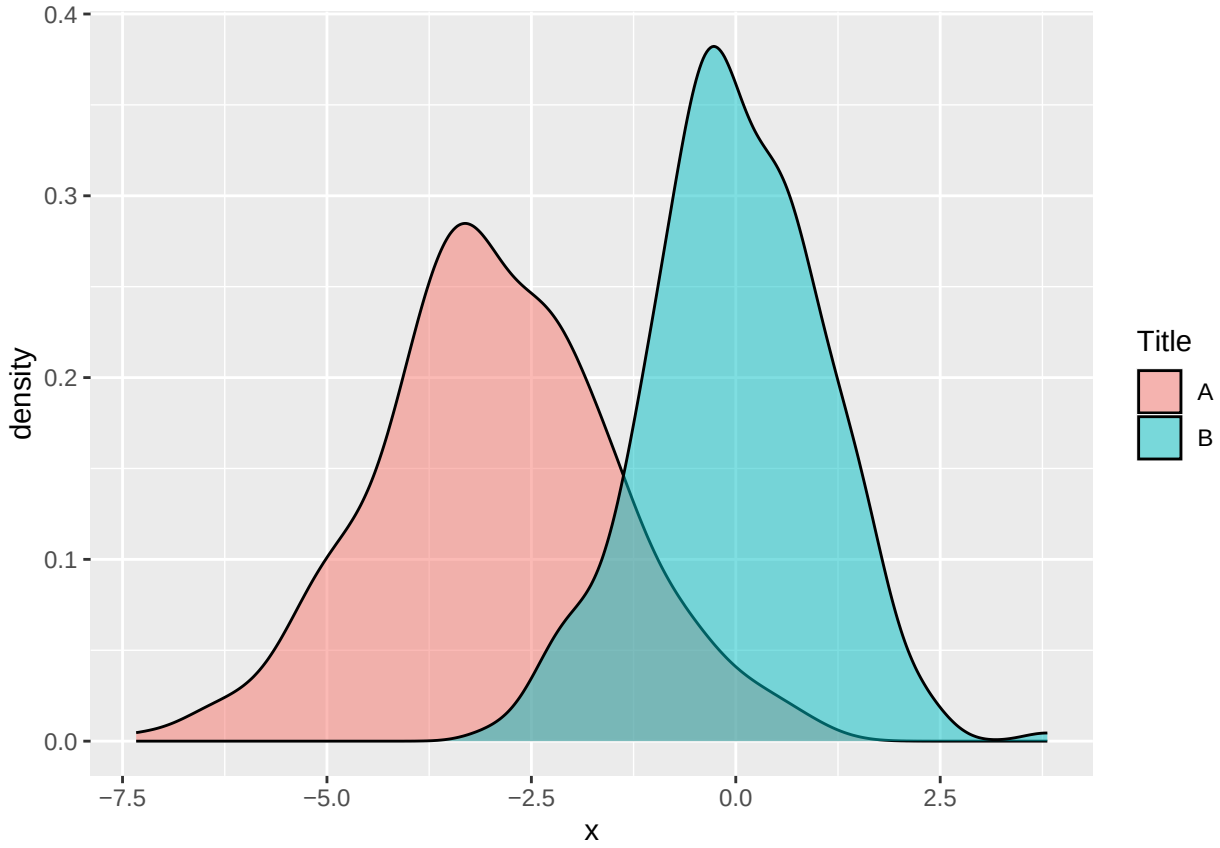


Customizing the legend title in ggplot2 with a scale

Option 3

The third option is passing the new title to the name argument of the corresponding "scale_x_y" functions, such as scale_fill_discrete or scale_color_discrete, if you are using them in your plot.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  scale_fill_discrete(name = "Title")
```

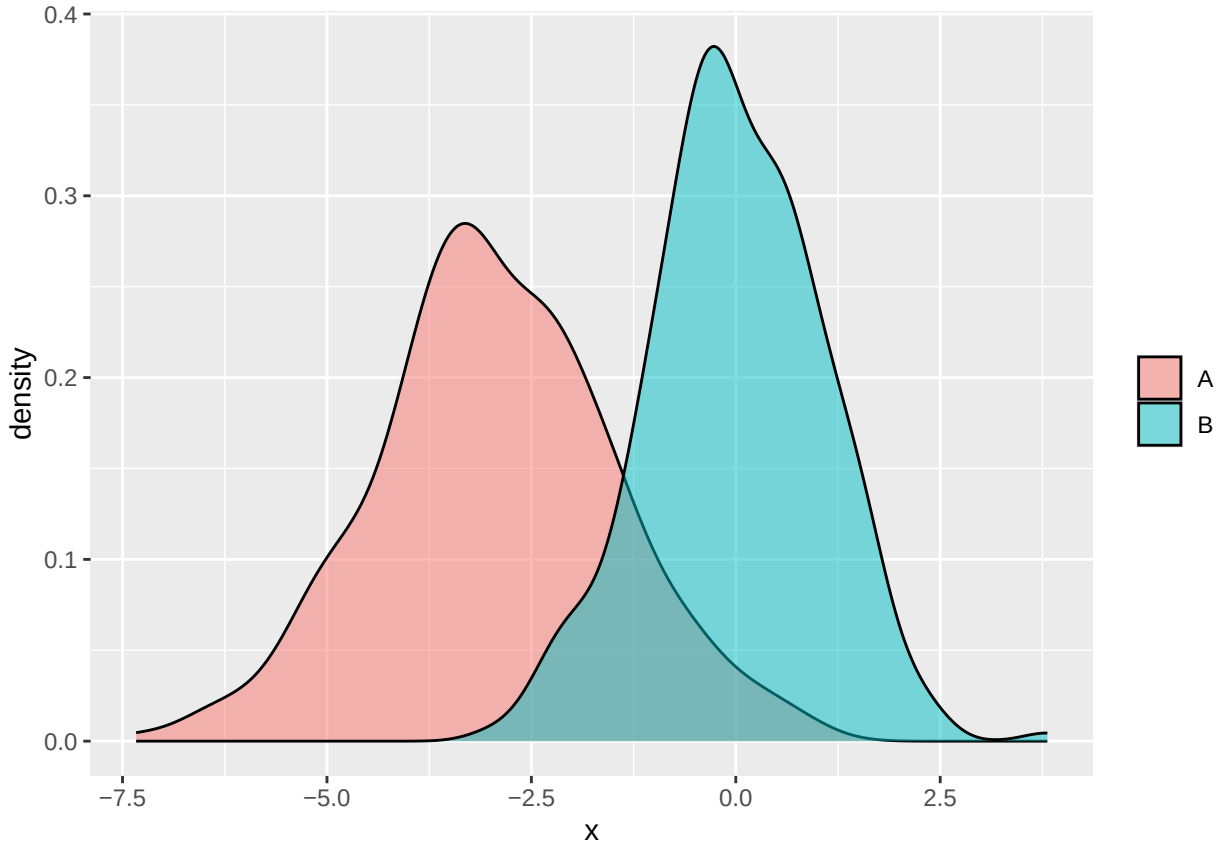


Remove the ggplot2 legend title

Removing the title

Finally, if you prefer removing the title of the legend just pass the element_blank function to the legend.title argument of theme.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.title = element_blank())
```

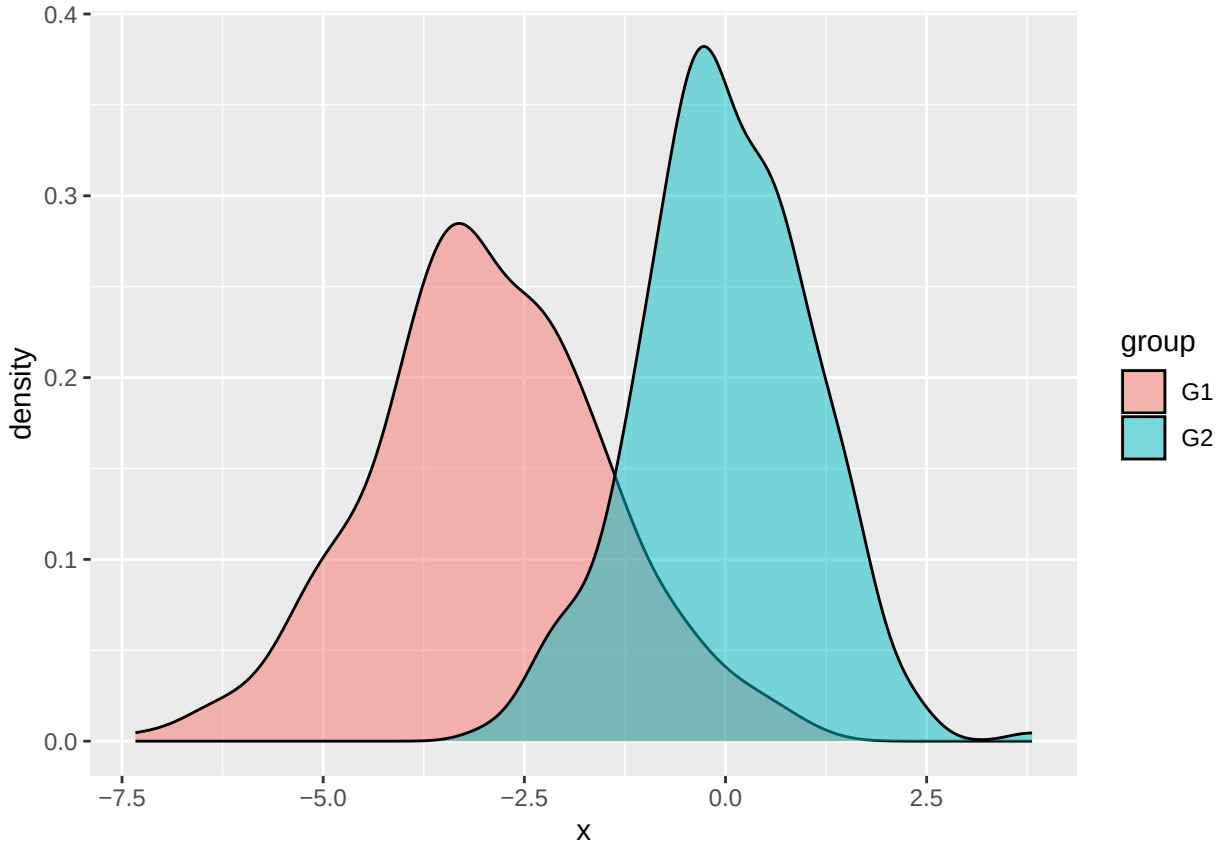


Change or reorder the labels of the legend If you want to change the group names of the legend you can customize the data frame column representing the groups.

Other option is passing the new labels to the labels argument of the scale_color_hue or scale_fill_hue functions to modify only the labels or using scale_color_discrete or scale_fill_discrete functions if you also want to change the colors.

New legend group labels

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  scale_fill_hue(labels = c("G1", "G2"))
```

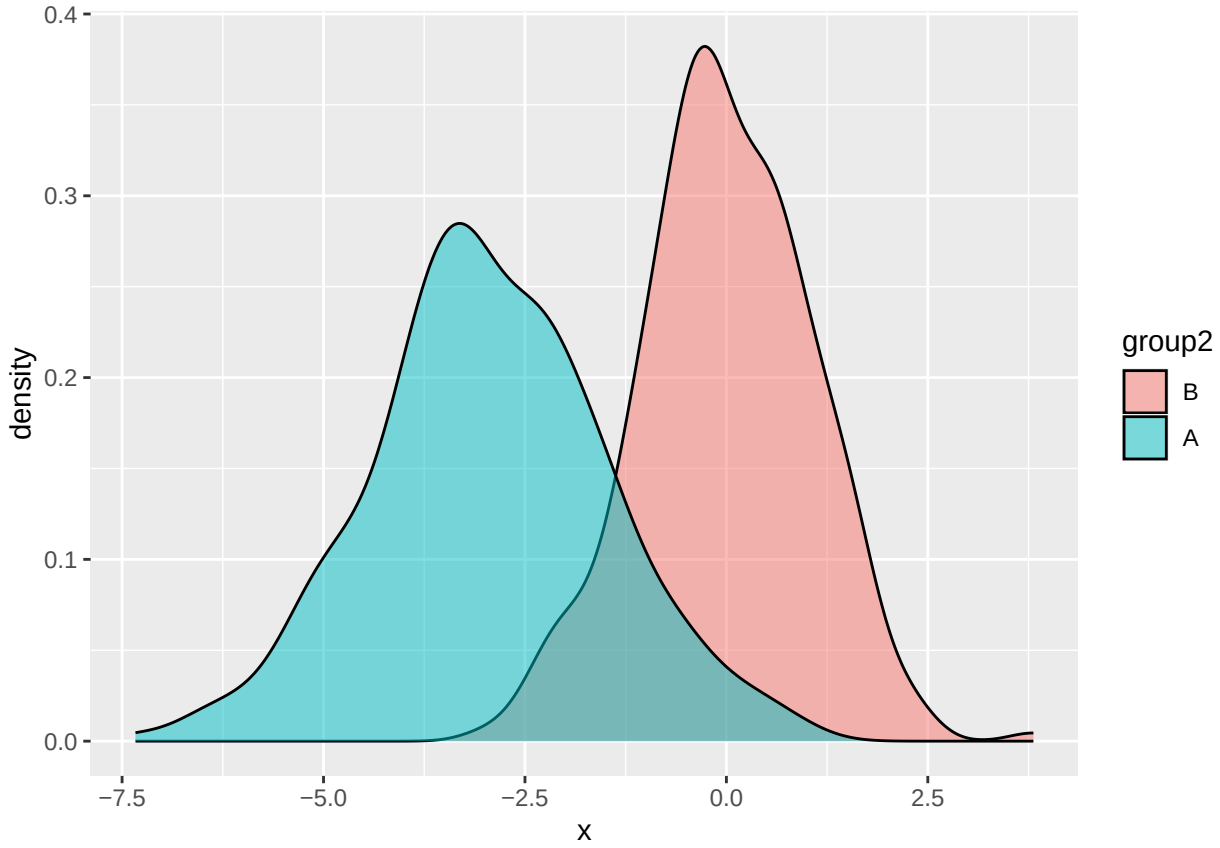


Customizing the legend labels in ggplot2

Reorder the labels

In case you want to reorder the labels of the legend you will need to reorder the factor variable. In this example we have created a new variable with the new order.

```
# install.packages("ggplot2")
library(ggplot2)

# Create a new variable with other levels
df$group2 <- factor(df$group,
                    levels = c("B", "A"))

ggplot(df, aes(x = x, fill = group2)) +
  geom_density(alpha = 0.5)
```



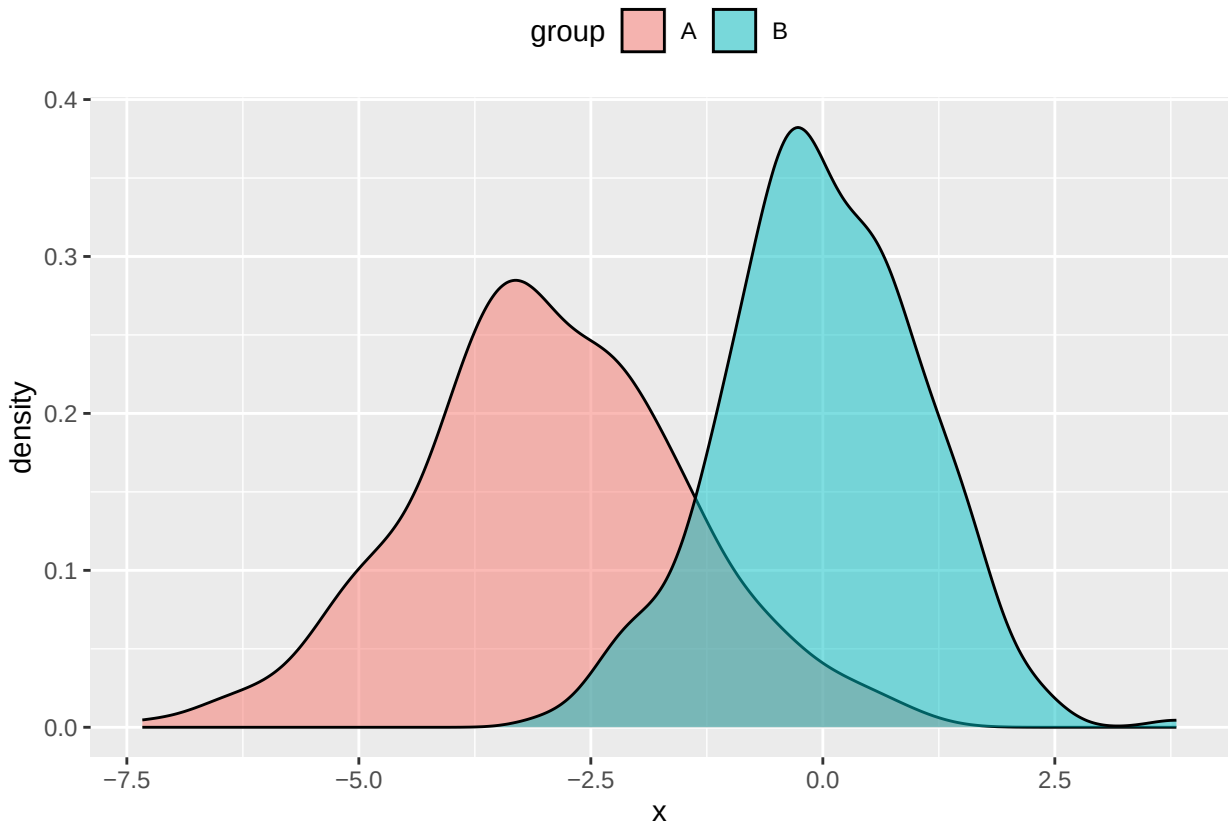
Reorder of the labels of the ggplot2 legend

Change the position of the legend By default, the automatic legend of a ggplot2 chart is displayed on the right of the plot. However, making use of the legend.position argument of the theme function you can modify its position. Possible values are “right” (default), “top”, “left”, “bottom” and “none”.

Legend at the top of the chart in ggplot2

Top

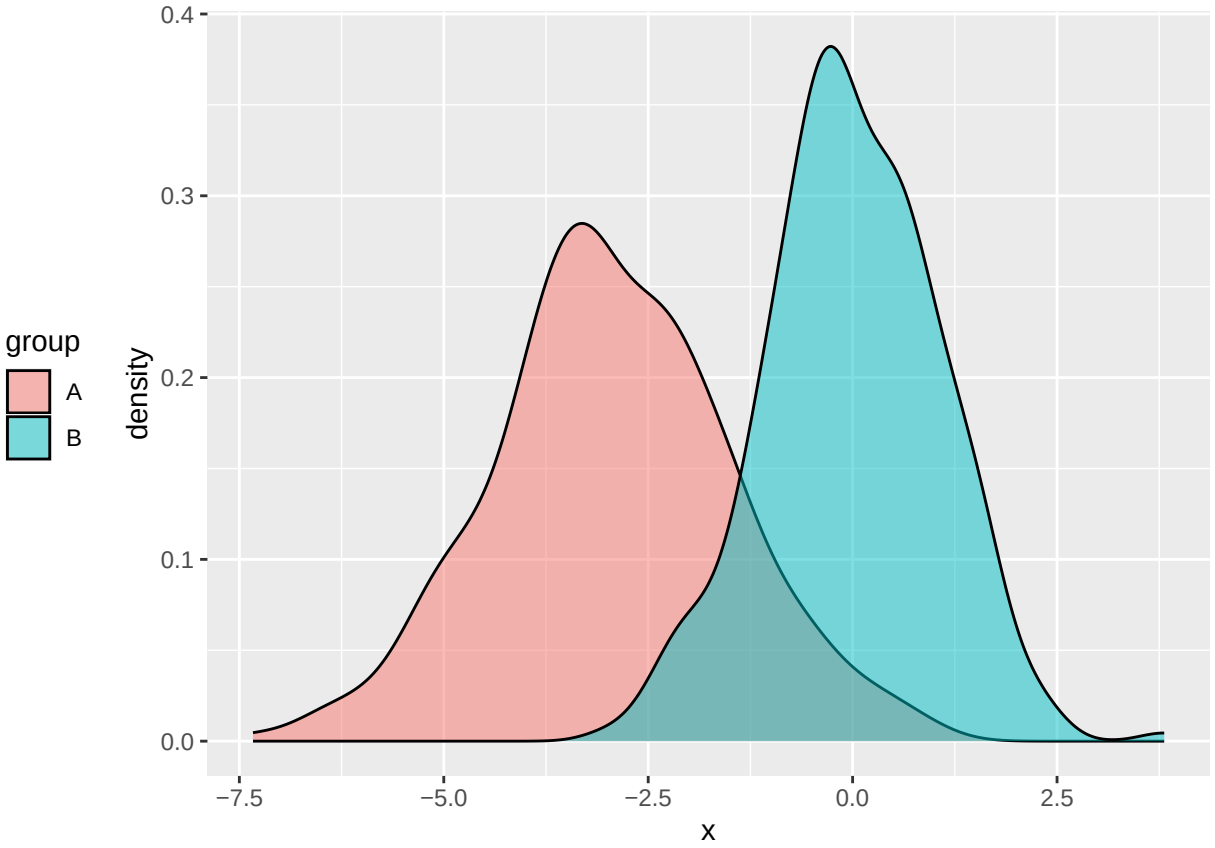
```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "top")
```



Legend at the left of the chart in ggplot2

Left

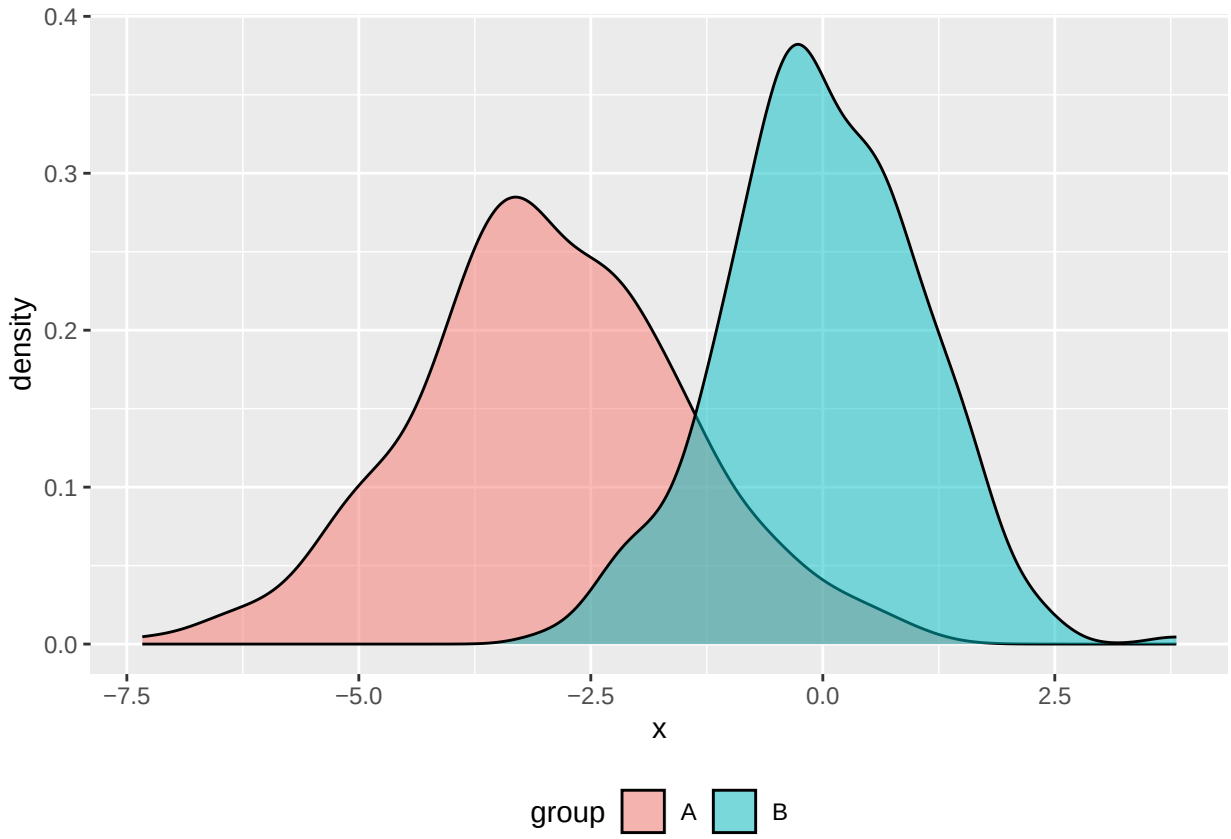
```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "left")
```



Legend at the bottom of the chart in ggplot2

Bottom

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "bottom")
```

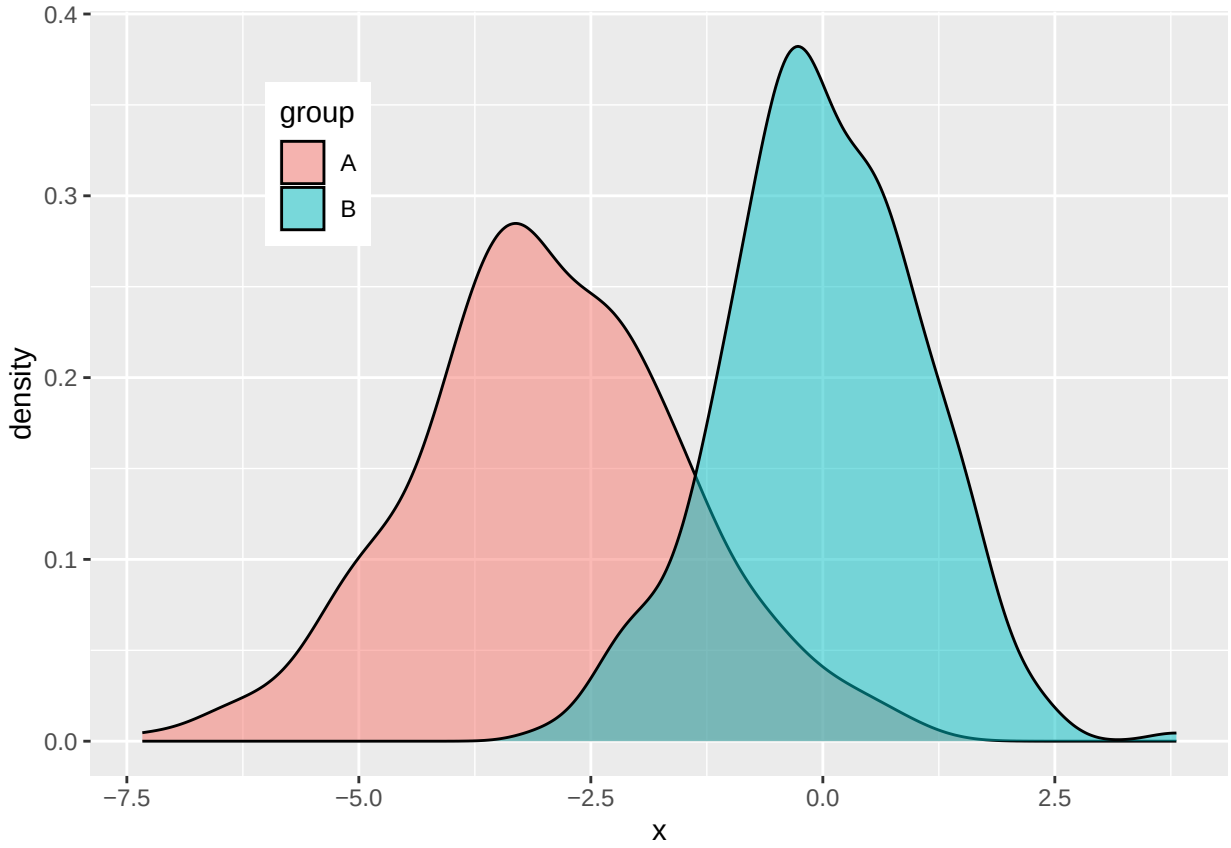


Legend inside a plot made with ggplot2

Custom position

Note that you can even set a custom position to place the ggplot2 legend inside the plot. You will need to set the coordinates between 0 an 1 with the legend.position argument of the theme function. Moreover, you can also change the background color of the legend with legend.background.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = c(0.2, 0.8),
        legend.background = element_rect(fill = "white"))
```



See also legend.justification, legend.direction and legend.box.just for further customization.

Legend customization There exist lots of arguments that allow customizing the ggplot2 legends. You will need to use the arguments of the theme function which starts with legend., such as legend.title, legend.background, legend.margin, . . .

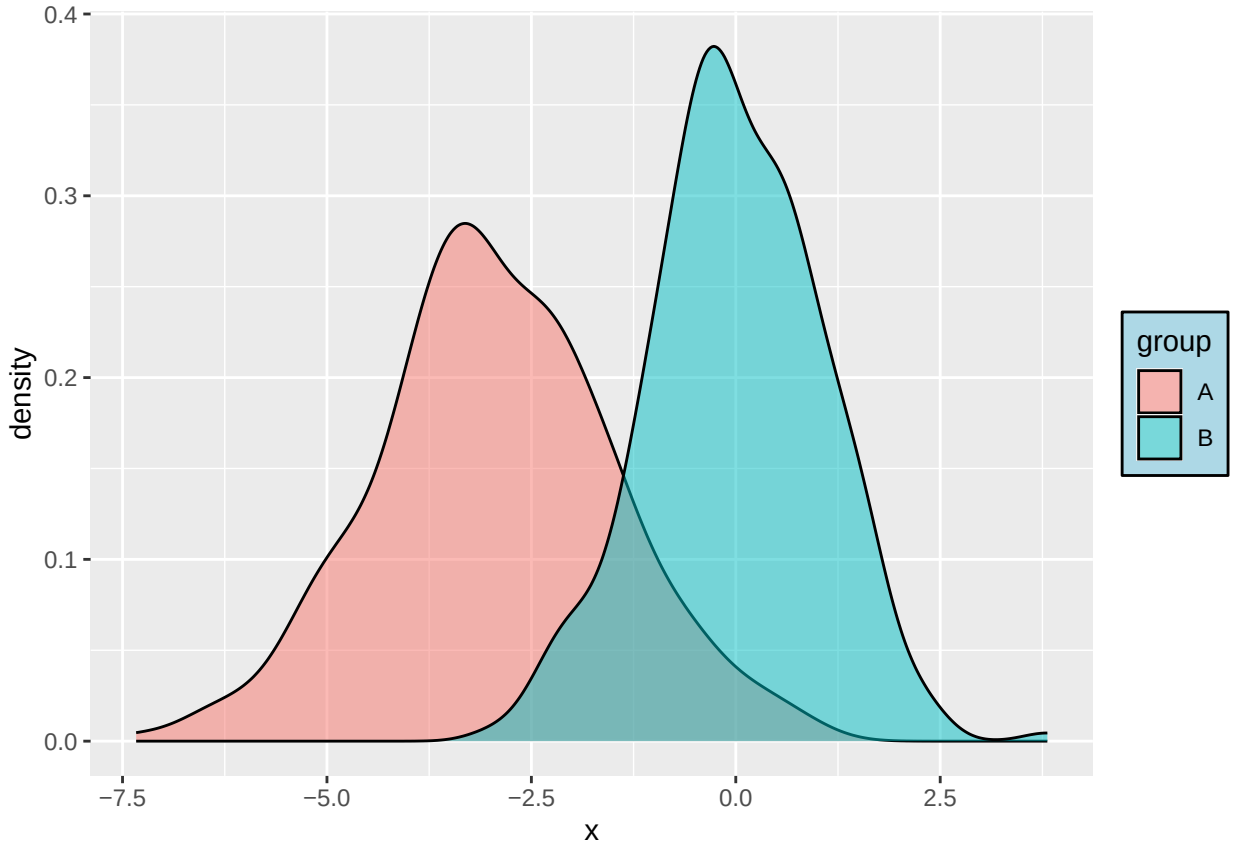
Customizing the title

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.title = element_text(family = "Roboto",
                                    color = "blue",
                                    size = 10,
                                    face = 2))
```

Legend title color and size customization in ggplot2

Customizing the legend background and border

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.background = element_rect(fill = "lightblue", # Background
                                         colour = 1))        # Border
```

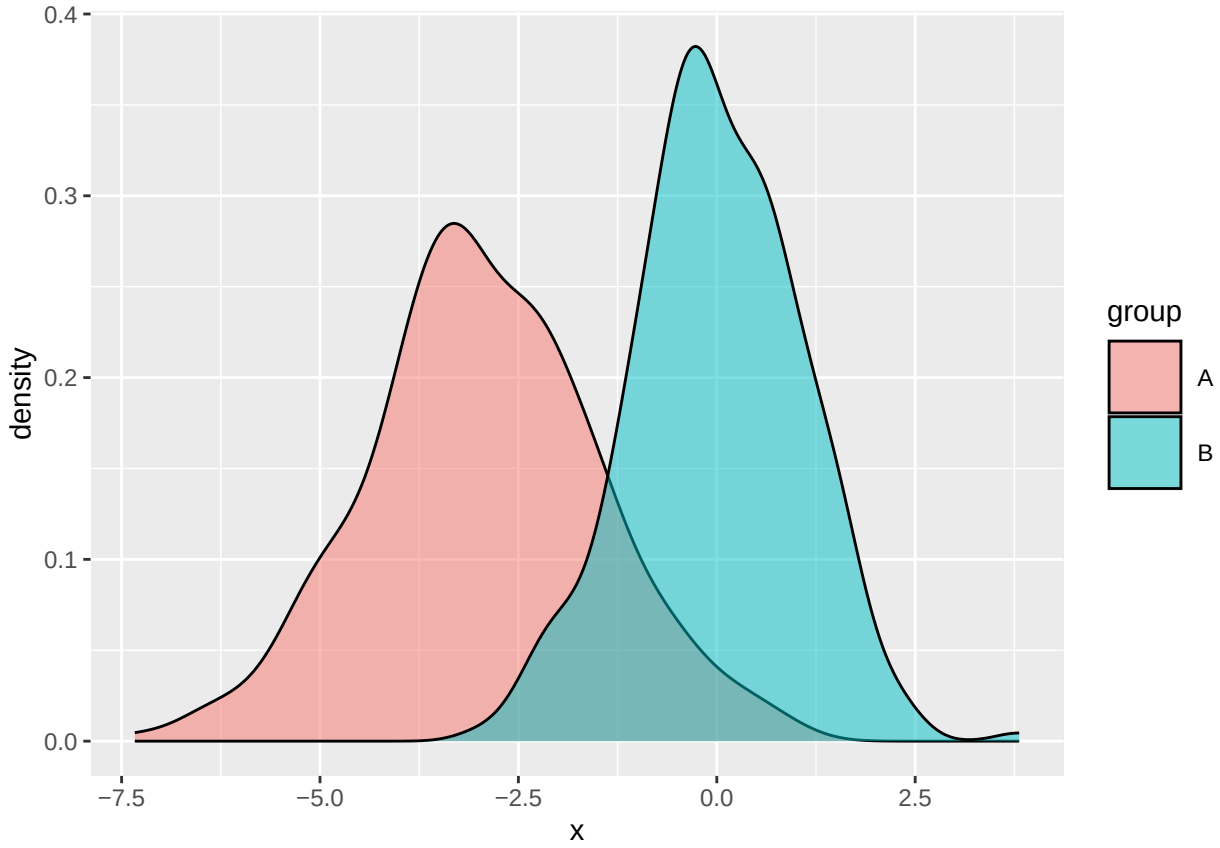


ggplot2 background and border color of the legend

Customizing the legend keys

In this example we are increasing the size of the boxes, but if you have for instance a scatter plot, you can change the size of the symbols with guides(color = guide_legend(override.aes = list(size = 4))), being 4 the new size.

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.key.size = unit(1, units = "cm"))
```



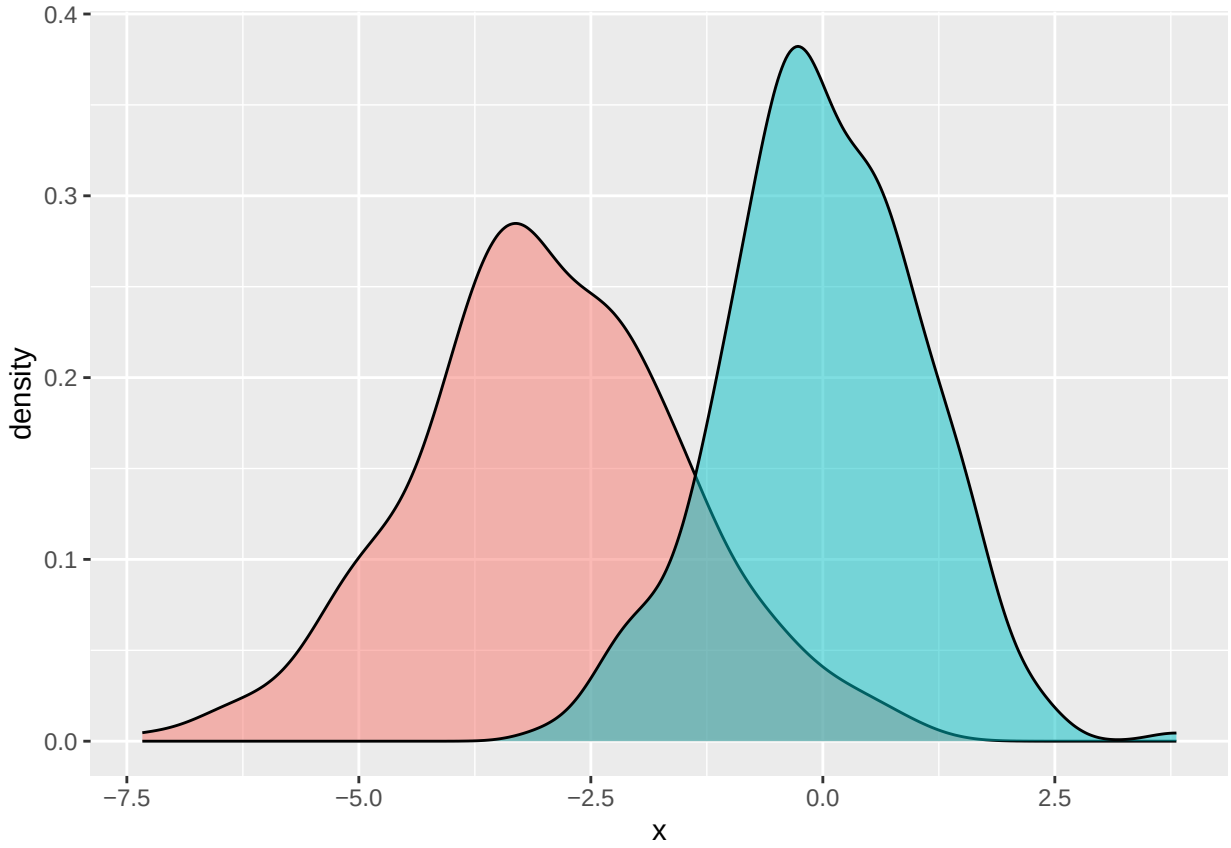
Size of the legend keys in ggplot2

Remove the legend Finally, in case you want to turn off the default legend you can set its position to “none”, as shown below:

Remove the ggplot2 legend

Removing the legend

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group)) +
  geom_density(alpha = 0.5) +
  theme(legend.position = "none")
```

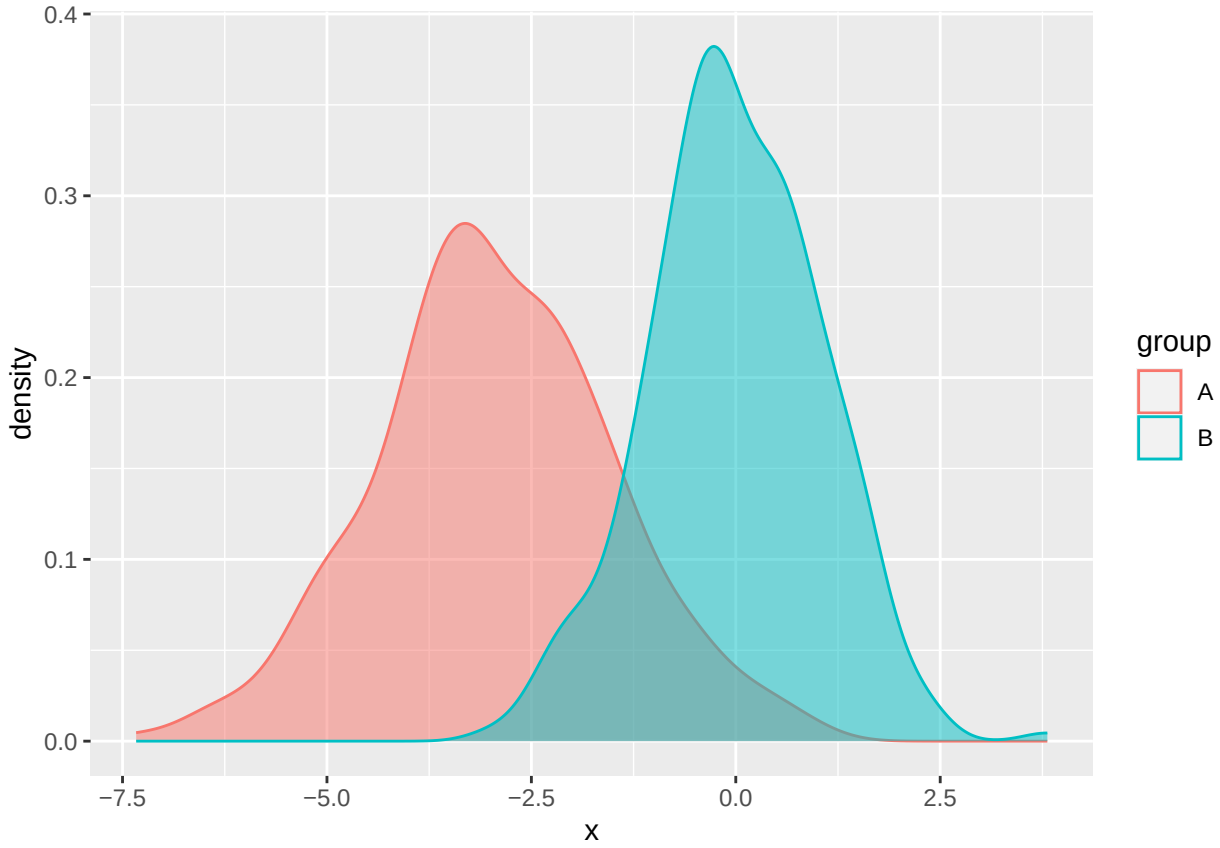


In addition, if your chart has several legends or you have mapped a variable to several arguments you can get rid of a specific legend setting that argument to "none" inside guides. In the following example we are removing the fill legend of the chart, letting only the color legend available.

Remove part of the ggplot2 legend

Removing a part of the legend

```
# install.packages("ggplot2")
library(ggplot2)

ggplot(df, aes(x = x, fill = group, color = group)) +
  geom_density(alpha = 0.5) +
  guides(fill = "none")
```



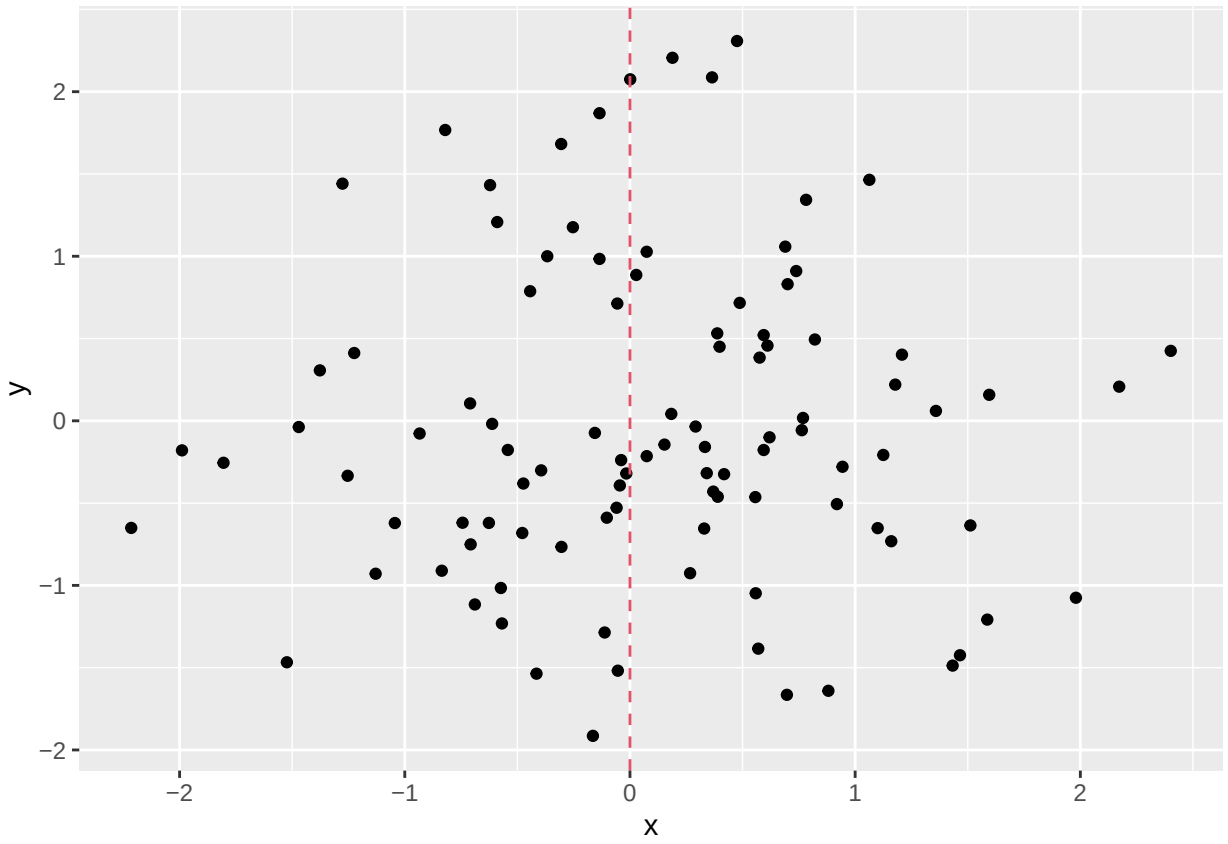
Reference lines, segments, curves and arrows in ggplot2

Vertical lines with geom_vline Horizontal lines with geom_hline Diagonal lines with geom_abline Segments with geom_segment Curves with geom_curve The ggplot2 package has several functions to add annotation layers to the plots such as reference lines (geom_vline, geom_hline and geom_abline), segments (geom_segment), curves (geom_curve) and arrows (arrows). In this tutorial we are going to review the most common use cases of these functions.

Vertical lines with geom_vline Considering that you have a plot made with ggplot2 you can add a new layer with geom_vline to add a single or multiple vertical lines. You just need to pass a single value or a numeric vector to the xintercept argument of the function where you want the lines to be displayed.

Note that you can also customize graphical arguments such as linetype, color or lwd.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_vline(xintercept = 0,
             linetype = 2,
             color = 2)
```

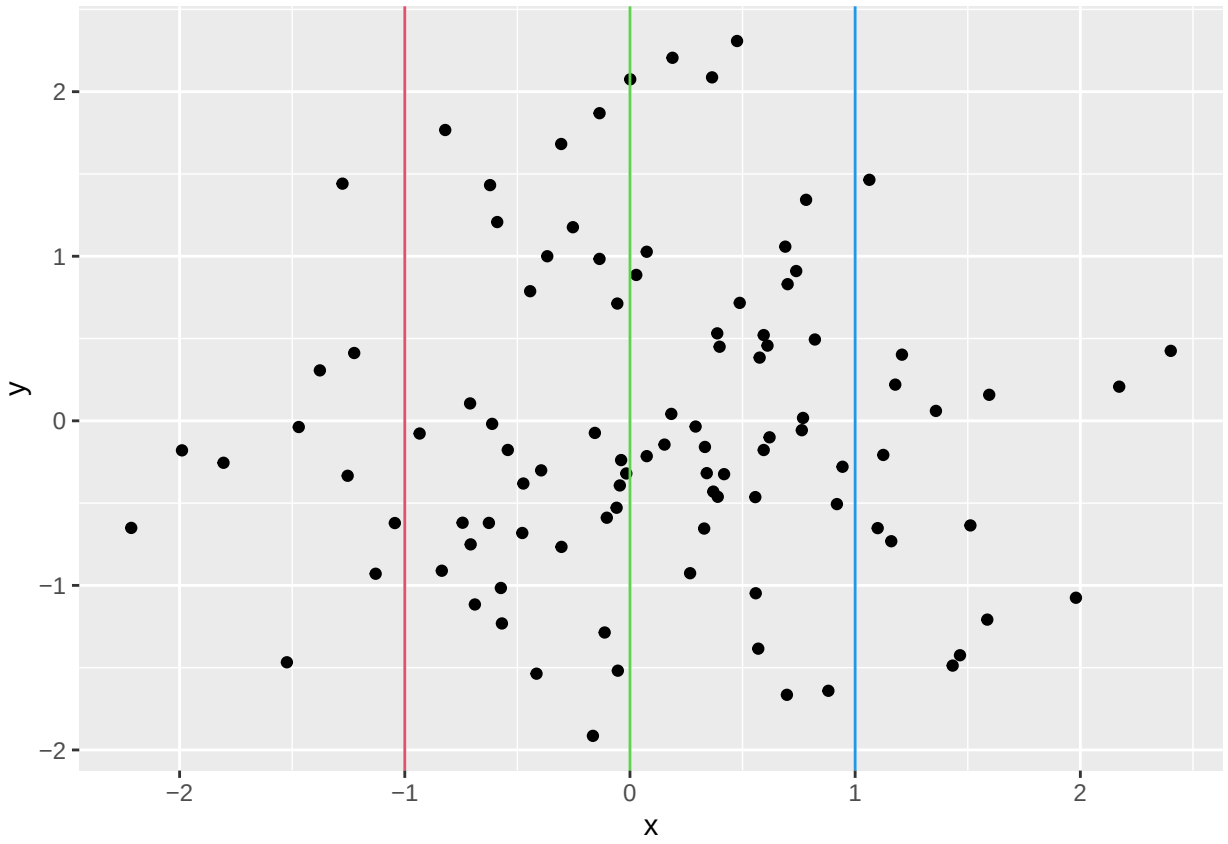


Vertical line in ggplot2 with geom_vline

Adding several lines at once

Note that all functions of this tutorial allow doing this.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_vline(xintercept = -1:1,
             linetype = 1,
             color = 2:4)
```



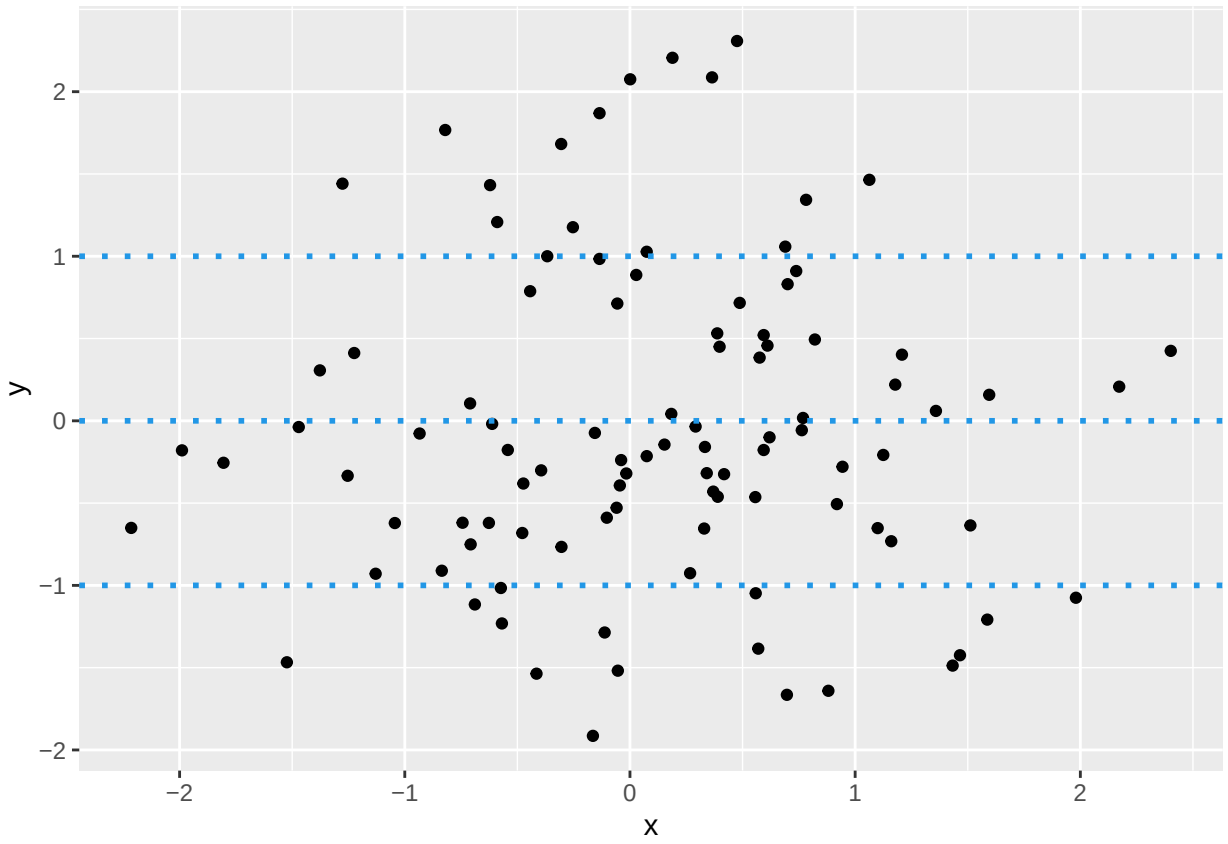
Adding multiple vertical lines in ggplot2 with geom_vline

Horizontal lines with geom_hline The geom_hline function behaves the same way as geom_vline. The only difference is that this function will add horizontal lines, so you will need to specify the yintercept argument instead of xintercept.

Horizontal lines in ggplot2 with geom_hline

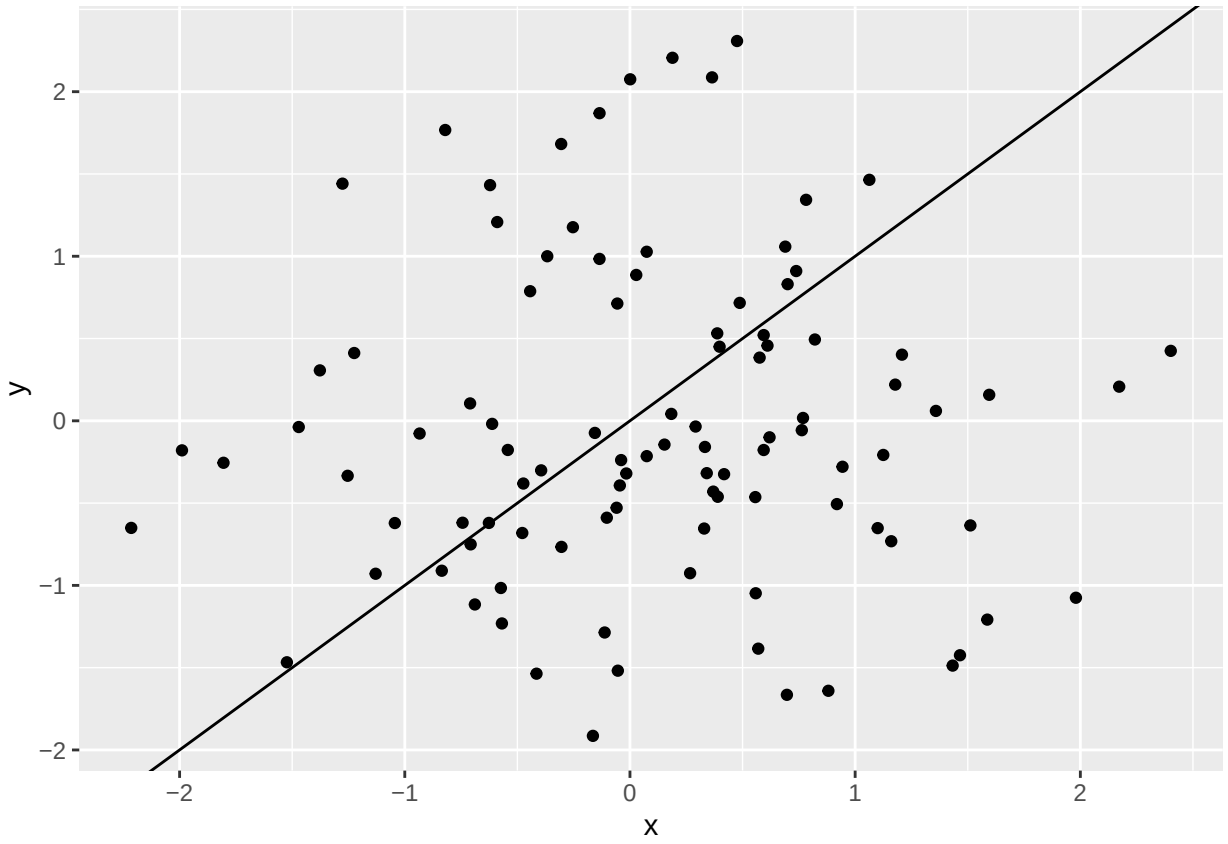
```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_hline(yintercept = -1:1,
             linetype = 3,
             color = 4,
             lwd = 1)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```



Diagonal lines with geom_abline The geom_abline function allows adding diagonal lines based on a intercept and a slope. The following example shows a diagonal line with intercept 0 and slope of 1.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1)
```



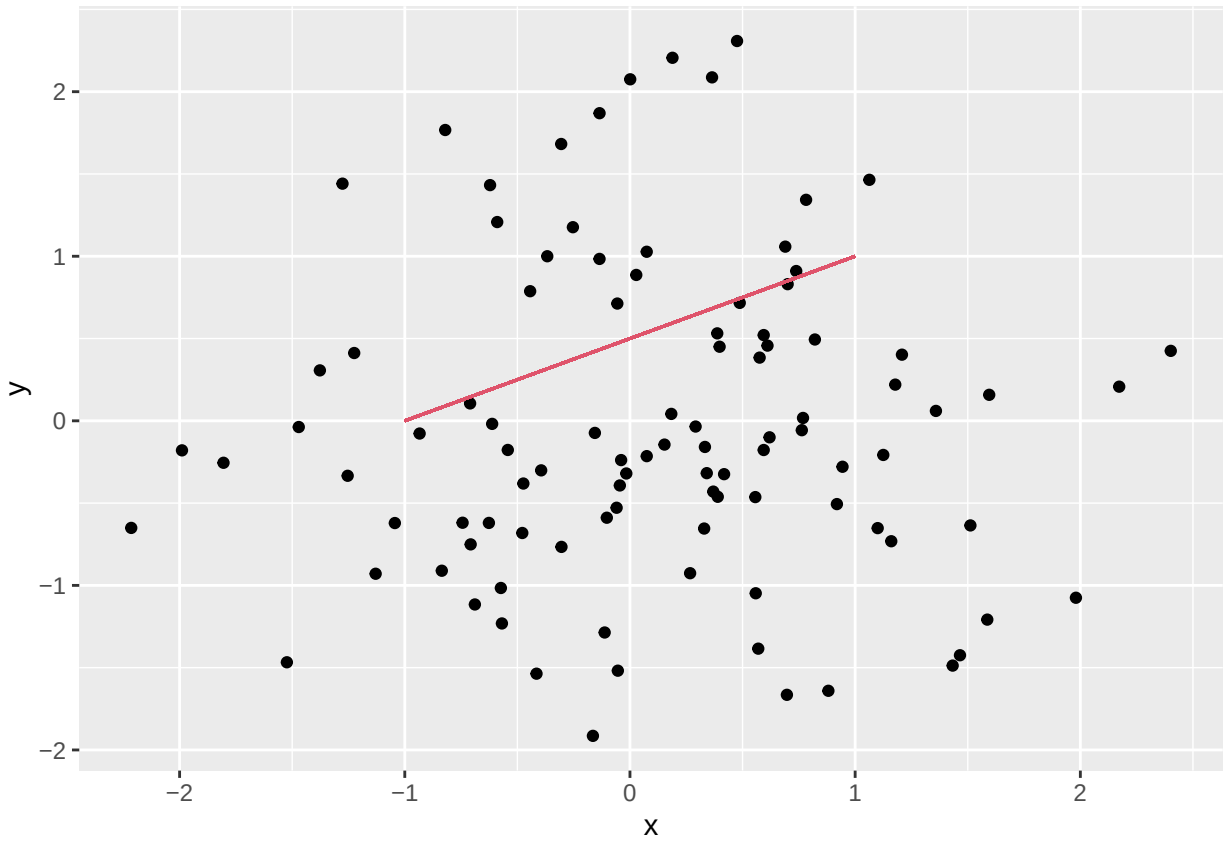
Adding a diagonal line in ggplot2 with geom_abline

Segments with geom_segment The previous functions can be used to add lines to the plots, but you cannot create segments with them. For that reason there exists the geom_segment function, which allows specifying the X and Y coordinates of the start and end of the desired segment with x, y (start) and xend, yend (end), respectively.

Note that if you pass the arrow function to the arrow argument you can create an arrow. See the arguments of the arrow function for further customization.

Adding segments in ggplot2 with geom_segment

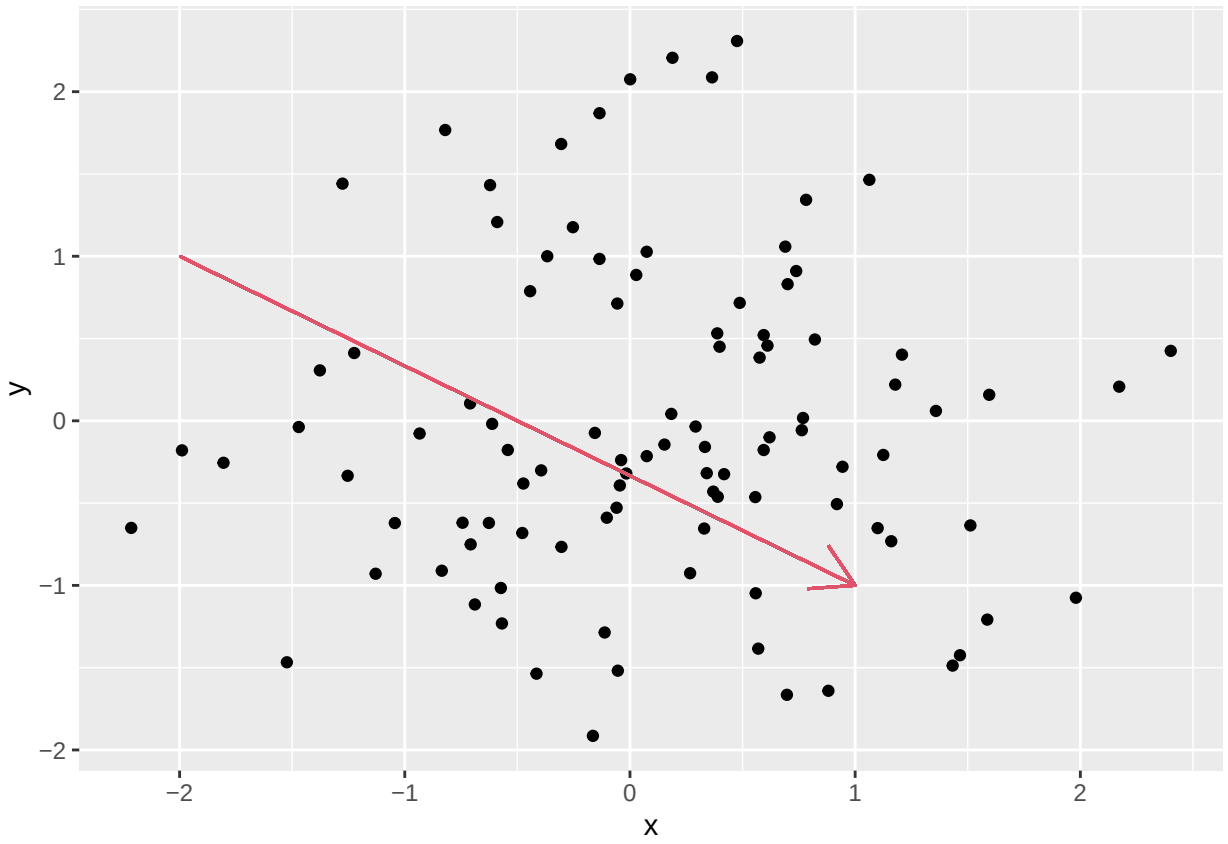
```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_segment(x = -1, y = 0,
               xend = 1, yend = 1,
               color = 2)
```



Adding an arrow to ggplot2 with geom_segment and the arrow function

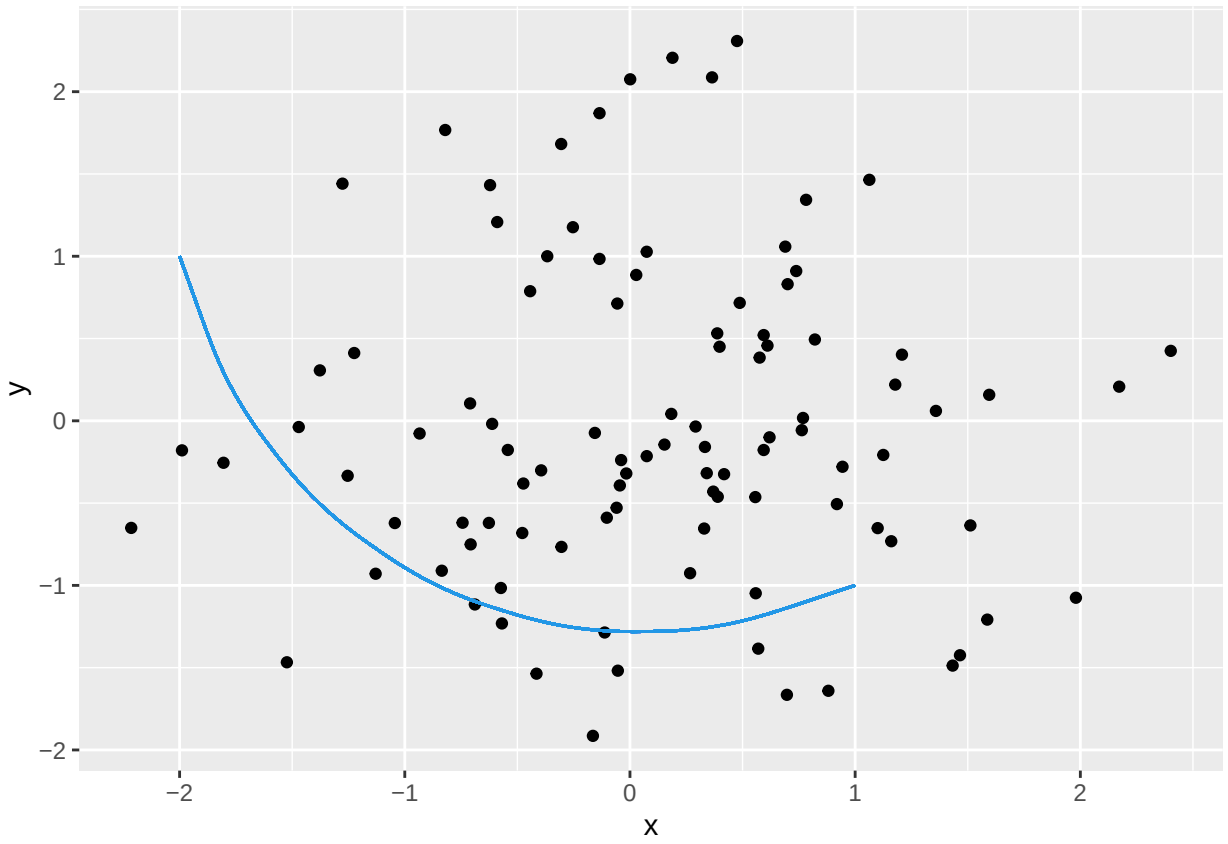
Annotate arrow

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_segment(x = -2, y = 1,
               xend = 1, yend = -1,
               color = 2,
               arrow = arrow())
```



Curves with geom_curve The geom_curve function behaves the same way as geom_segment. The only difference is that this function will create a curve instead of a straight line. You can also create an arrow using this function.

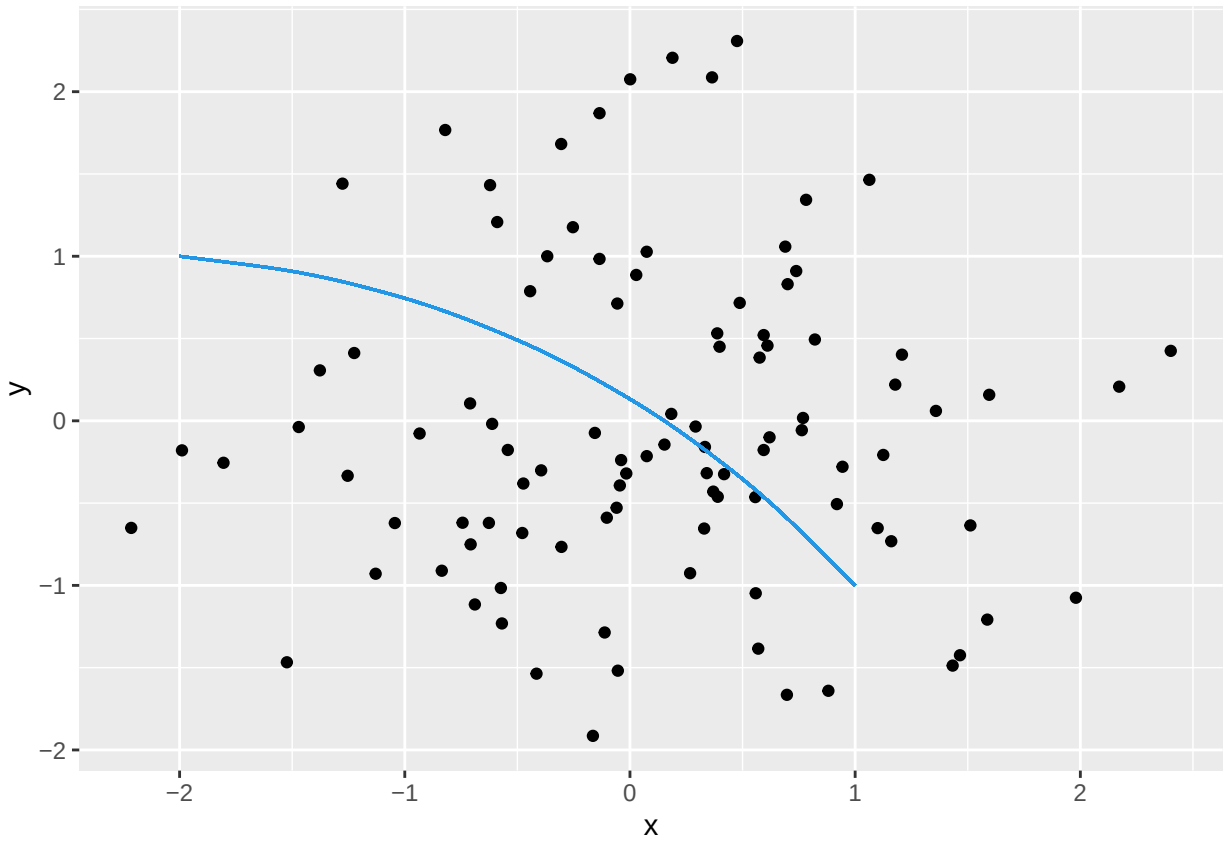
```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 1,
             xend = 1, yend = -1,
             color = 4)
```



Adding a curve to ggplot2 with geom_curve

Note that you can set the level of curvature passing a value the curvature argument of the function. A zero is a straight line, negative values will produce left-handed curves and positive values produce right-handed curves.

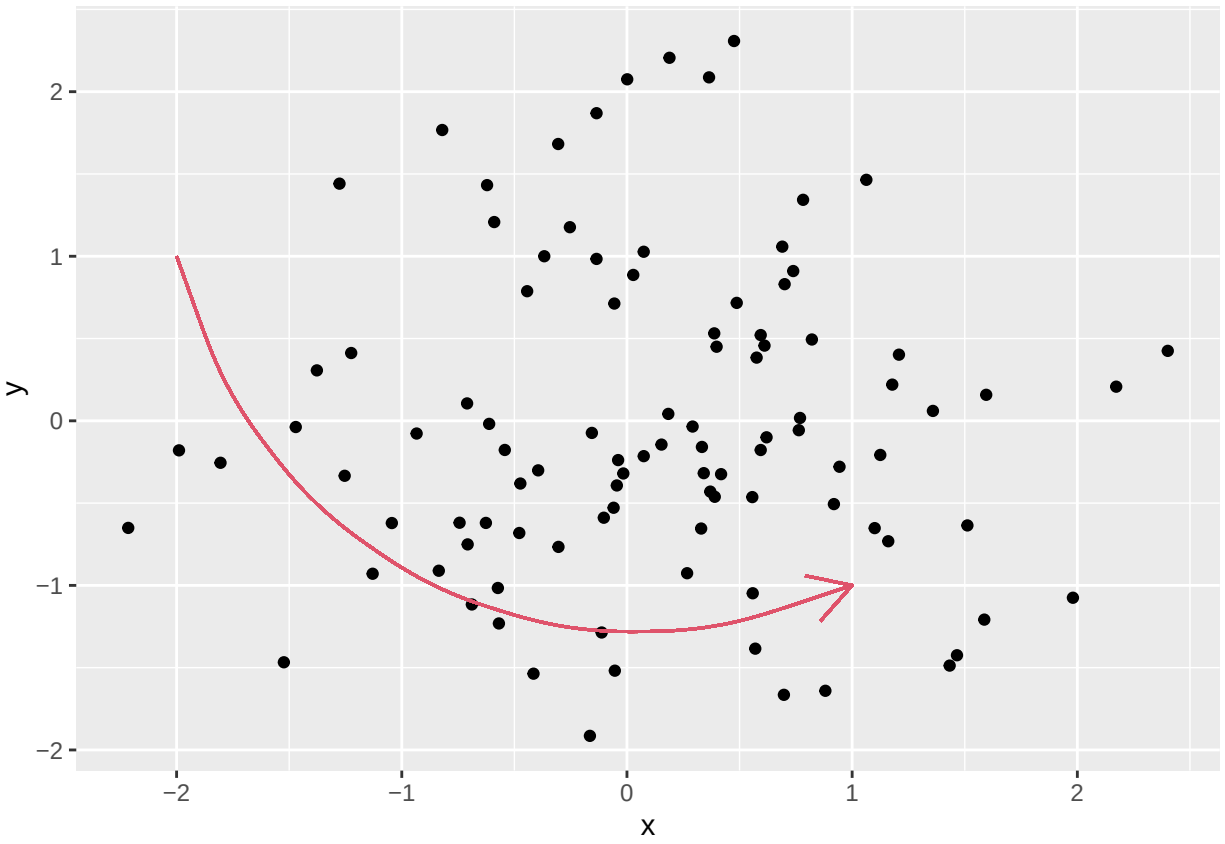
```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 1,
             xend = 1, yend = -1,
             color = 4,
             curvature = -0.2) # Level of curvature
```



Changing the level of curvature of the ggplot2 curve annotation

Annotate arrow

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(1)
df <- data.frame(x = rnorm(100),
                 y = rnorm(100))

ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_curve(x = -2, y = 1,
             xend = 1, yend = -1,
             color = 2,
             arrow = arrow())
```



Curved arrow in ggplot2

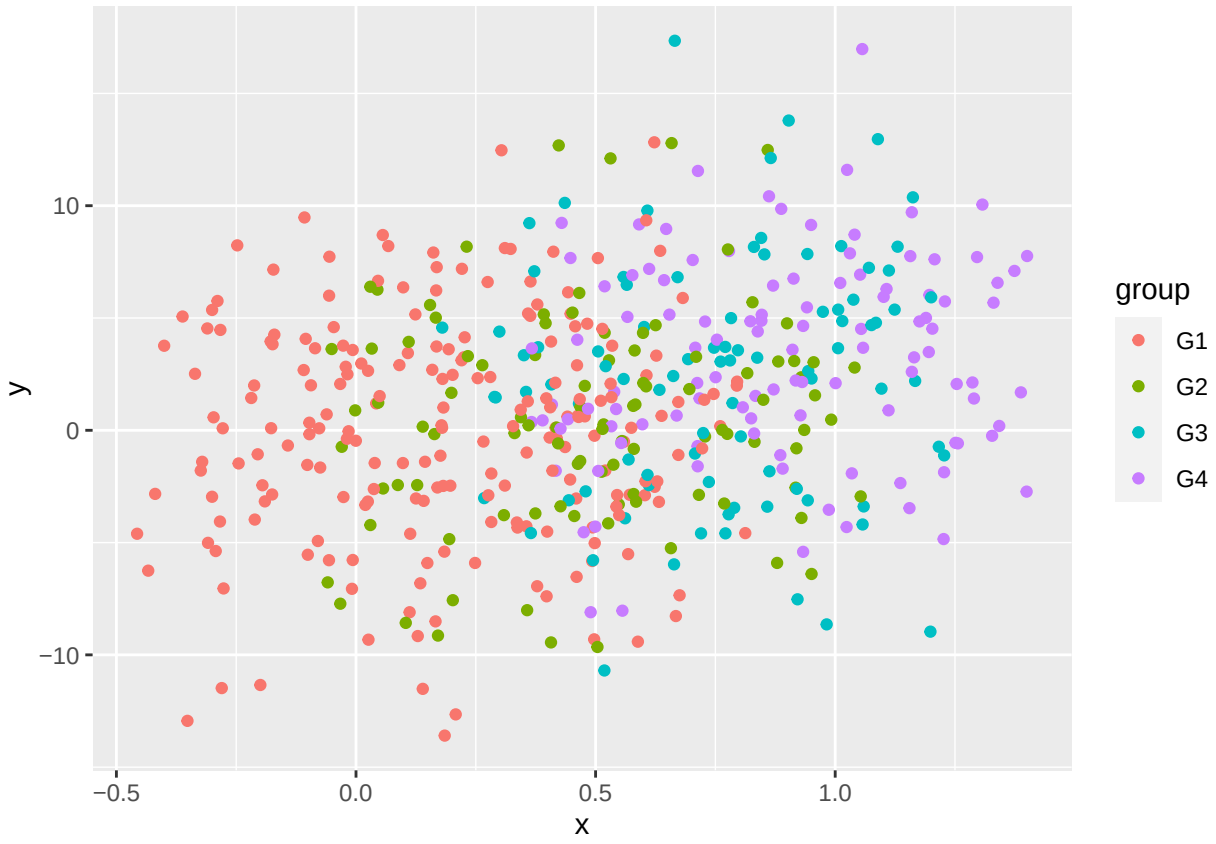
Faceting in ggplot2 with facet_wrap and facet_grid

One discrete variable: facet_wrap Two discrete variables: facet_grid Further customization of the strips and panels When using ggplot2 you can create multi panel plots, also known as Trellis plots or facets with the facet_grid or facet_wrap functions. These functions are similar, but there are some differences between them, as the former creates a matrix of panels based on two discrete variables (it also works with one, but its not recommended) while the latter creates a ribbon of plots based on a single discrete variable (it also works with two, but its not recommended).

One discrete variable: facet_wrap Consider, for instance, that you have a set of observations that belong to different groups. In this scenario, you can display them in the same graph and highlight the groups with a different color or marker.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)
```

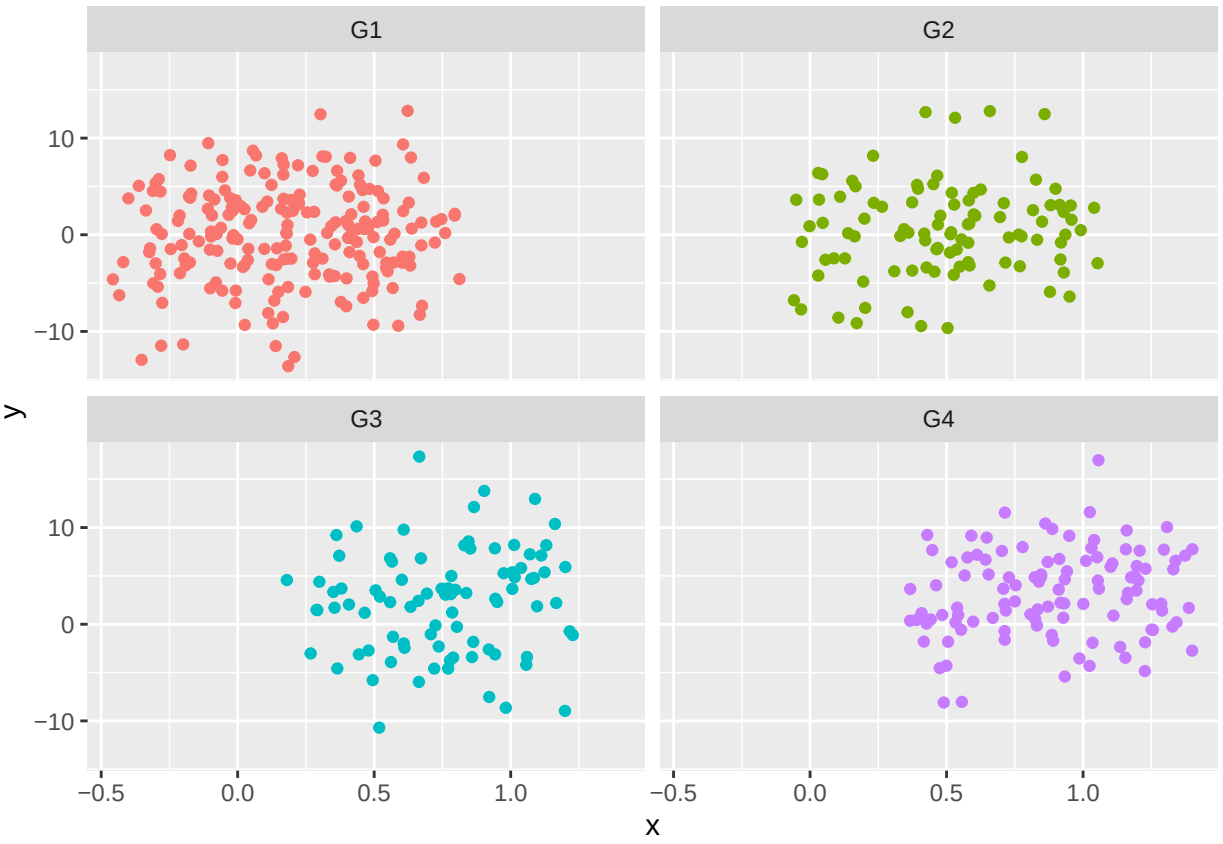
```r
# Scatter plot by group
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point()
```



Scatter plot with several groups in ggplot2

The problem is that sometimes the chart is not readable, so an alternative is to split the data into different plots to see each group individually. For this purpose you can make use of the facet_wrap function passing the grouping variable as formula, e.g. ~group.

```r
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group)
```



Facets in ggplot2 with facet_wrap

Change the order of the panels ordering the levels of the categorical variable with factor.

Number of rows and columns

The default number of rows and columns will depend on your data. If your categorical variable has four groups the facet_wrap function will create two columns and two rows, but in this example maybe having only one row is more readable, so you can set nrow = 1 or ncol = 4.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots (2 columns)
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group, nrow = 1) # Or ncol= 4
```

Multi panel plots in ggplot2 with the facet_wrap function

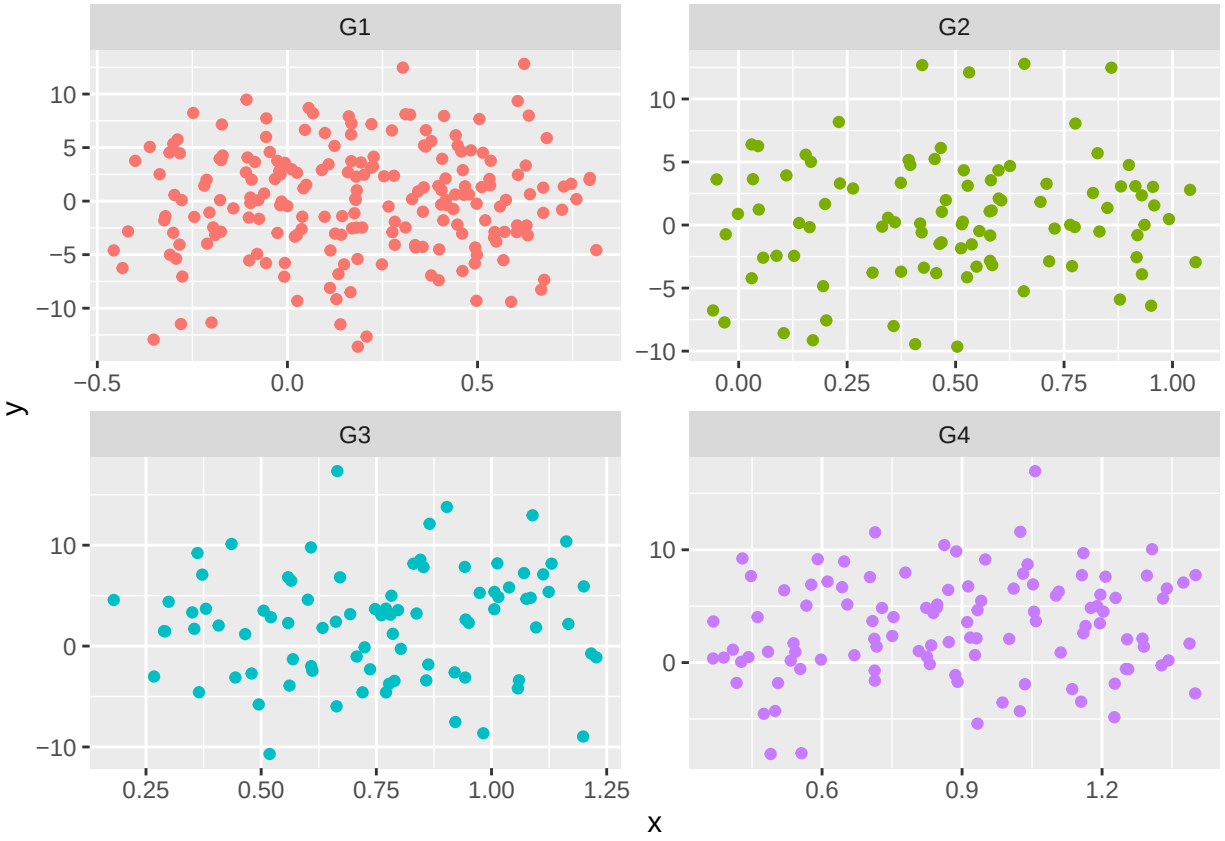
If instead of facet_wrap we use facet_grid, the same panels will be created, but there would always be only one row or only one column. That's why its use its not recommended and is better to use facet_wrap and be able to choose the number of rows or columns.

Axis scales

By default, all the panels will have the same scale along the axis, but when the data is in different scale you can set scales = "free", so each plot will have a different range. Setting the scales free makes easier to see patterns on each panel, while fixed makes easier to find patterns across panels. Note that you can also free a single axis with scales = "free_x" or scales = "free_y".

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
```

```
  facet_wrap(~group, scales = "free")
```

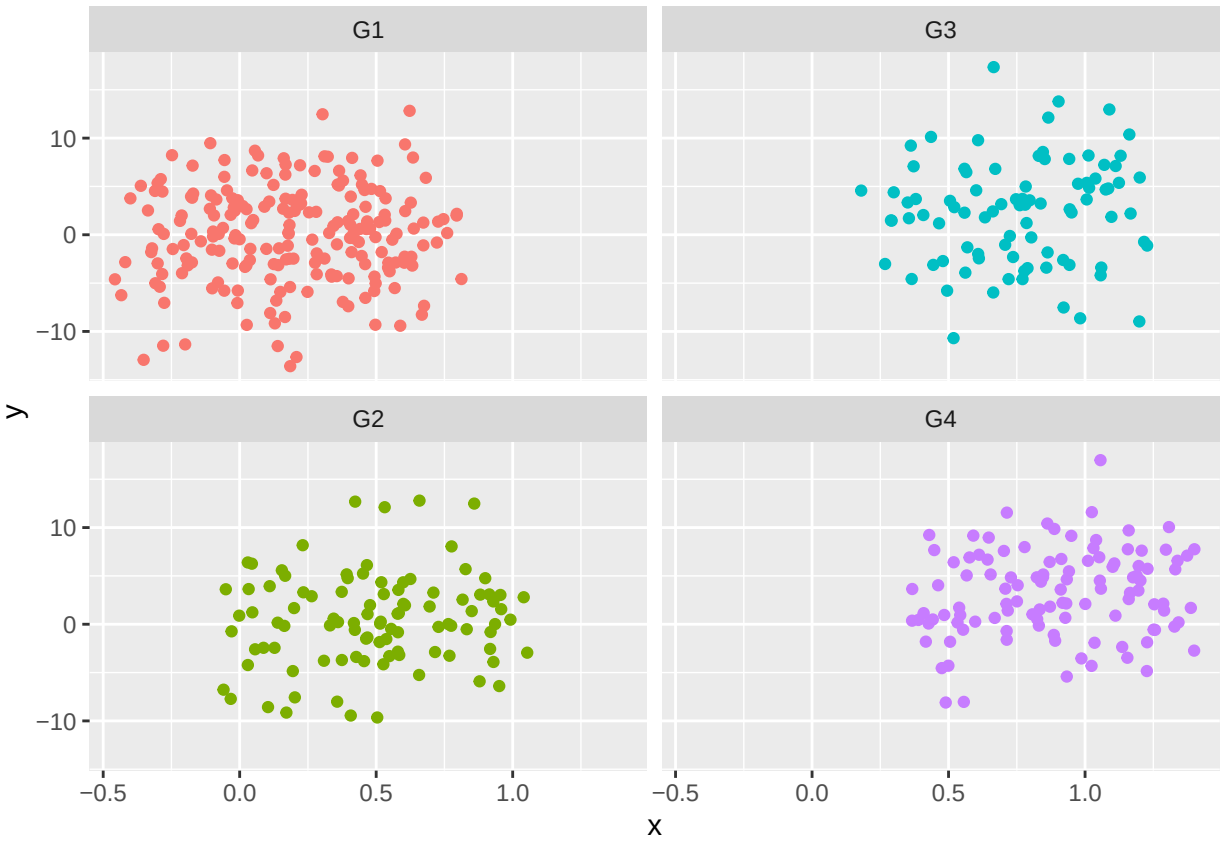


ggplot2 facets with free scales

Panels direction

The default direction for the panels is horizontal (dir = "h") but you can also set it to vertical with dir = "v". See that the upper right panel now corresponds to the third groups instead to the second.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Vertical ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group, dir = "v")
```

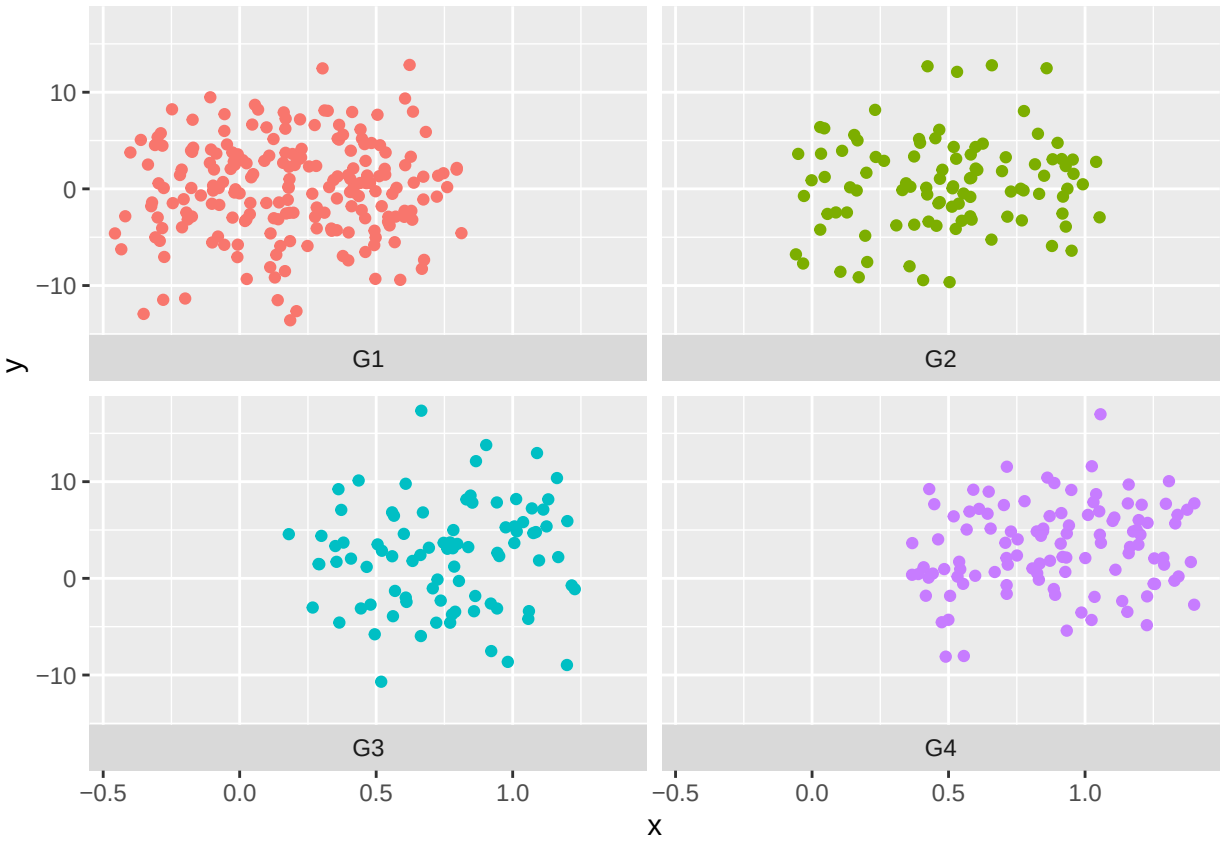


Vertical direction of the panels with facet_wrap

Labels position

The facet_wrap function provides an argument named strip.position that allows customizing the position of the labels. The default value is “top”, but it could also be “bottom”, “left” or “right”.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group, strip.position = "bottom")
```

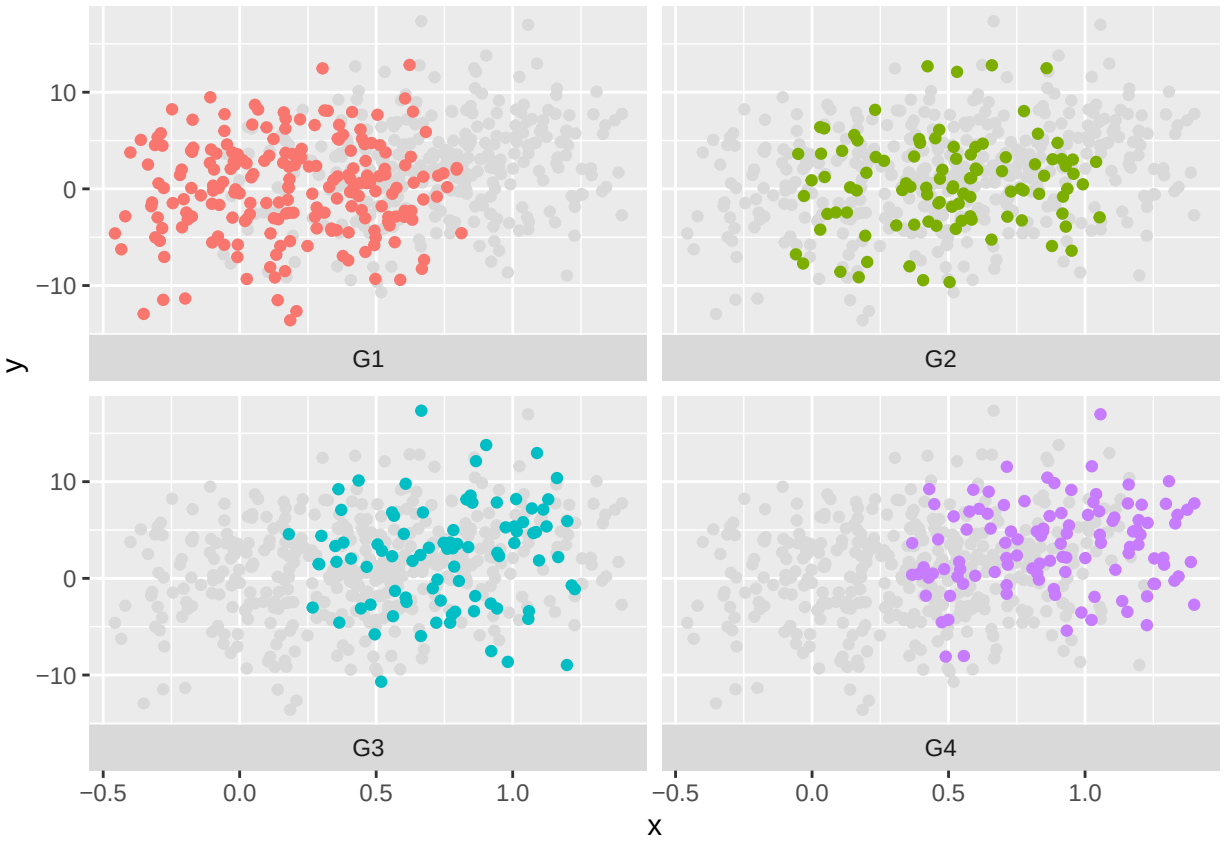


Position of the strips of the faceting panels in ggplot2

Highlight each group showing all the data behind

It is possible to add all the data to all panels and highlight the corresponding data for each one. For this purpose you will need to create a new layer passing as input the same data frame used before but removing the categorical variable (in this example is named group).

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(data = transform(df, group = NULL), colour = "grey85") +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group, strip.position = "bottom")
```

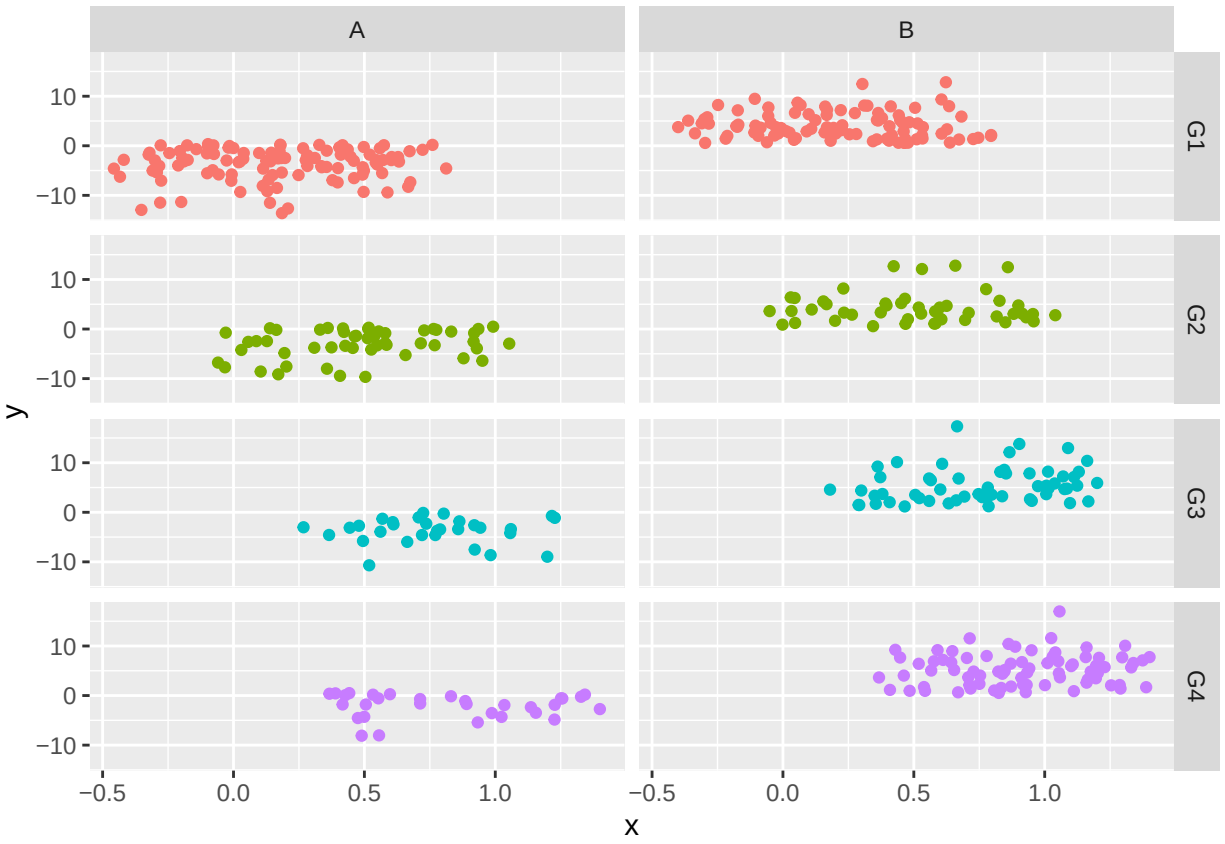


Highlight each group showing all the data behind in ggplot2 facets

Two discrete variables: facet_grid The facet_grid function is very similar to facet_wrap, but this function creates panels showing all the combinations of the data for two categorical variables. There are two ways to input data: the function can take a formula with the names of the categorical variables or you can pass the names of the variables to rows and cols arguments making use of the vars function.

Multi panelling in R with the facet_grid function from ggplot2

```r
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group1 <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
group2 <- ifelse(y < 0.5, "A", "B")
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group1 = group1, group2 = group2)

# Matrix of plots
ggplot(df, aes(x = x, y = y, color = group1)) +
  geom_point(show.legend = FALSE) +
  facet_grid(group1 ~ group2) # Or facet_grid(rows = vars(group1), cols = vars(group2))
```

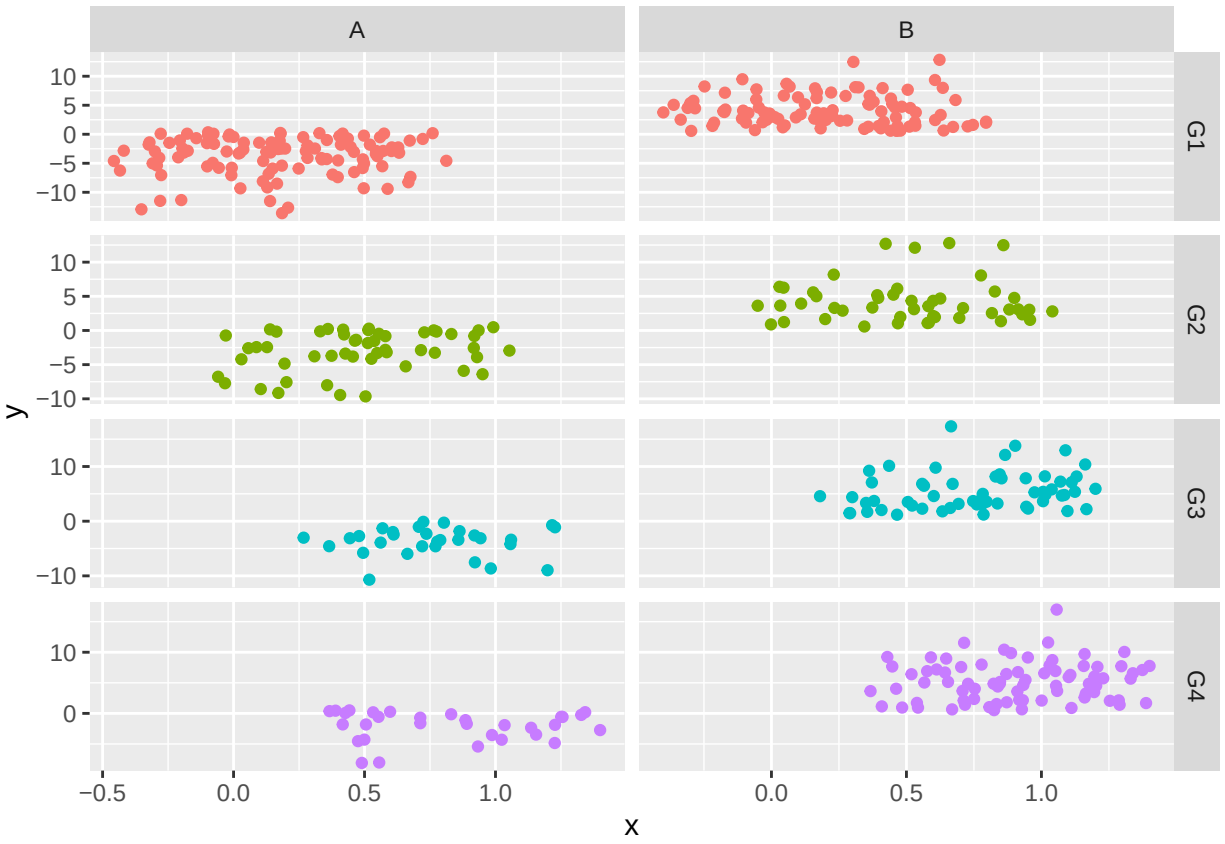


Axis scales

The facet_grid function also provides the scales argument to let the axis scales vary across rows and/or columns.

ggplot2 facet_grid free scales

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group1 <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
group2 <- ifelse(y < 0.5, "A", "B")
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group1 = group1, group2 = group2)

# Matrix of plots
ggplot(df, aes(x = x, y = y, color = group1)) +
  geom_point(show.legend = FALSE) +
  facet_grid(group1 ~ group2, scales = "free")
```

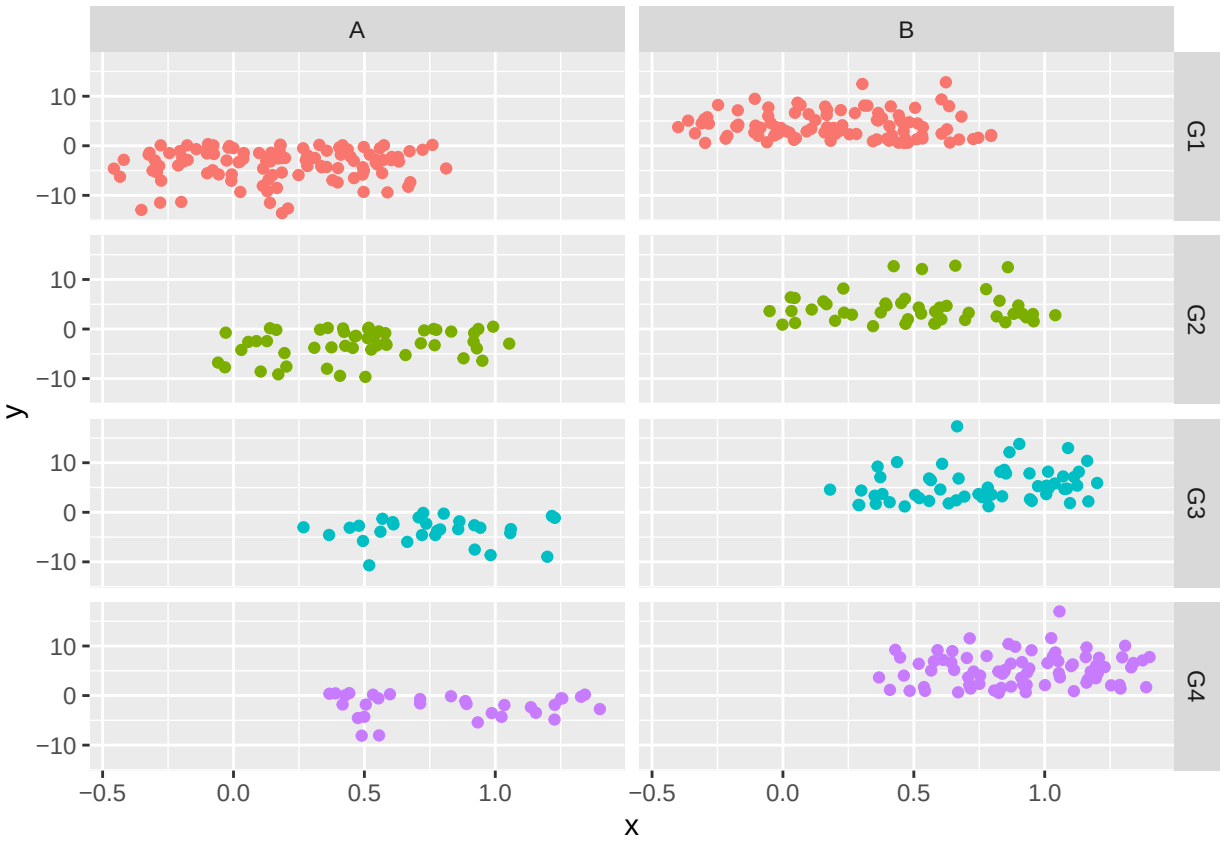


Panels size

If the panels scales vary, you can use the space argument to make the panels have different size. Possible values are “fixed” (default), “free_y” to free the heights, “free_x”, to free the widths and “free” to let both the width and height vary. Note that in this example the argument doesn’t make any difference as all panels have the same scale.

facet_grid function in R

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group1 <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
group2 <- ifelse(y < 0.5, "A", "B")
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group1 = group1, group2 = group2)

# Matrix of plots
ggplot(df, aes(x = x, y = y, color = group1)) +
  geom_point(show.legend = FALSE) +
  facet_grid(group1 ~ group2, space = "free_x")
```



The facet_grid function provides more arguments, such as shrink, labeller, as.table, switch, drop and margins. Recall to read the documentation for further details if needed.

Further customization of the strips and panels Using the arguments of the theme function you can customize the strips and panels. In the following examples we are going to review some use cases.

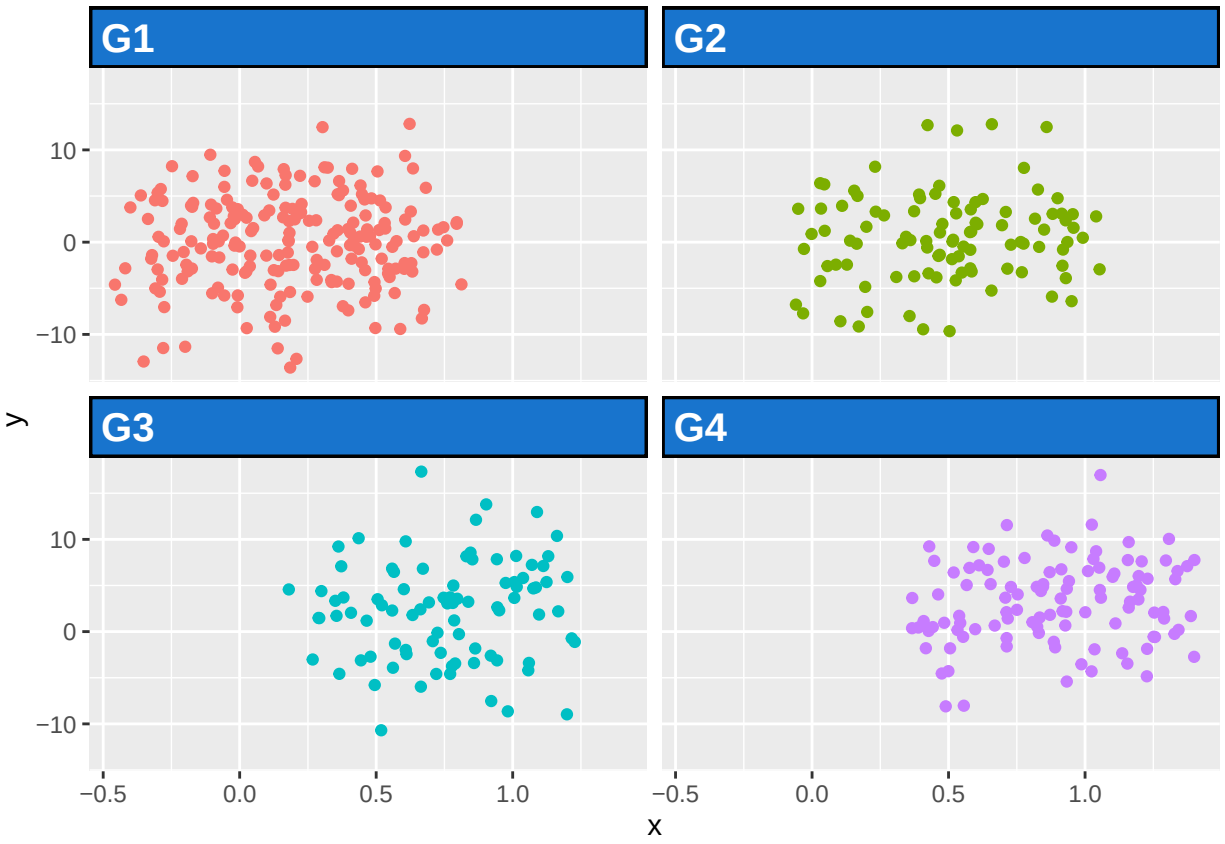
Customization of the strips of facet_wrap

You can customize the text styling of the labels with the strip_text argument. This argument takes the element_text function as input, where you can specify the different styles, such as the color, size, adjustment, etc. If you want to remove the labels use element_blank instead.

In order to customize the background color you will need to pass the element_rect function to strip.background and set the fill color with fill. In case you want to add a border you will need to set a line type, a color and a line width.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
```

```
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group) +
  theme(strip.text = element_text(face = "bold", color = "white", hjust = 0, size = 15),
        strip.background = element_rect(fill = "dodgerblue3", linetype = "solid",
                                        color = "black", linewidth = 1))
```



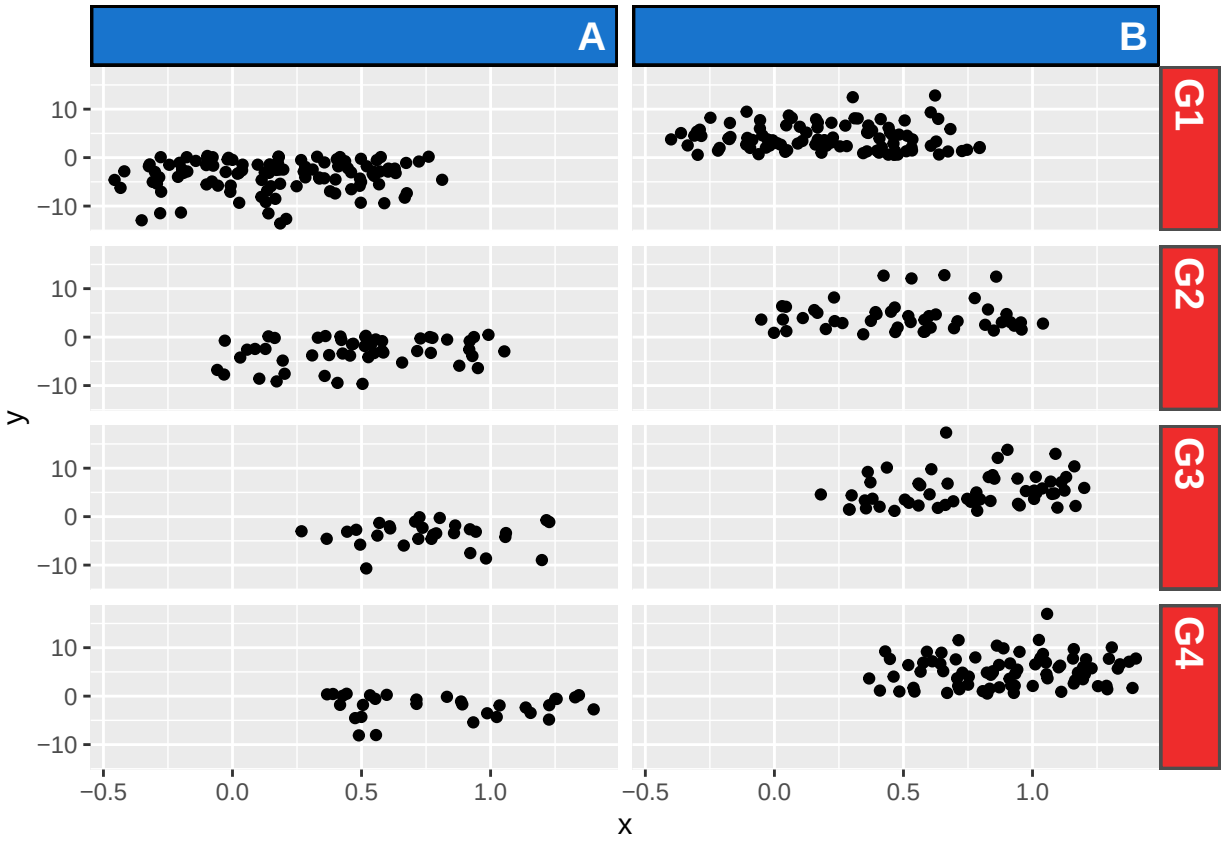
Background and text color of the strips of a ggplot2 facet plot made with facet_wrap

Customization of the strips of facet_grid

If you have created a multi panel plot with facet_grid you can customize the strips the same way as in the example before, but you can also customize the labels for each axis individually making use of strip.text.x, strip.text.y, strip.background.x and strip.background.y.

```
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group1 <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
group2 <- ifelse(y < 0.5, "A", "B")
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group1 = group1, group2 = group2)
```

```
# Ribbon of plots
ggplot(df, aes(x = x, y = y,)) +
  geom_point(show.legend = FALSE) +
  facet_grid(group1 ~ group2) +
  theme(strip.text.x = element_text(face = "bold", color = "white", hjust = 1, size = 15),
        strip.text.y = element_text(face = "bold", color = "white", hjust = 0, size = 15),
        strip.background.x = element_rect(fill = "dodgerblue3", linetype = "solid",
                                          color = "black", linewidth = 1),
        strip.background.y = element_rect(fill = "firebrick2", linetype = "solid",
                                          color = "gray30", linewidth = 1))
```



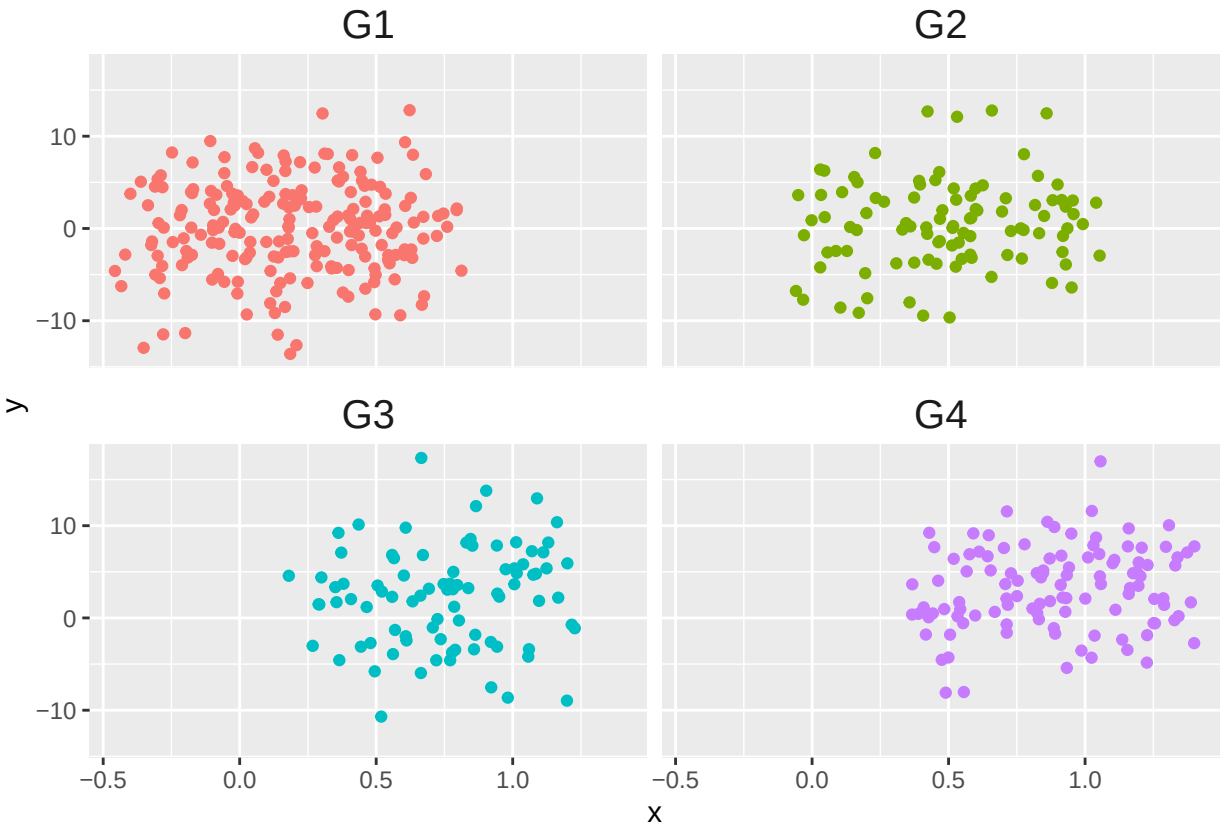
Customization of the strips of a panel plot made with facet_grid

Remove the background color of the strips

Note that it is possible to remove the background of the strips passing the element_blank function to strip.background.

```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)
```

```r
# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group) +
  theme(strip.text = element_text(size = 15),
        strip.background = element_blank())
```



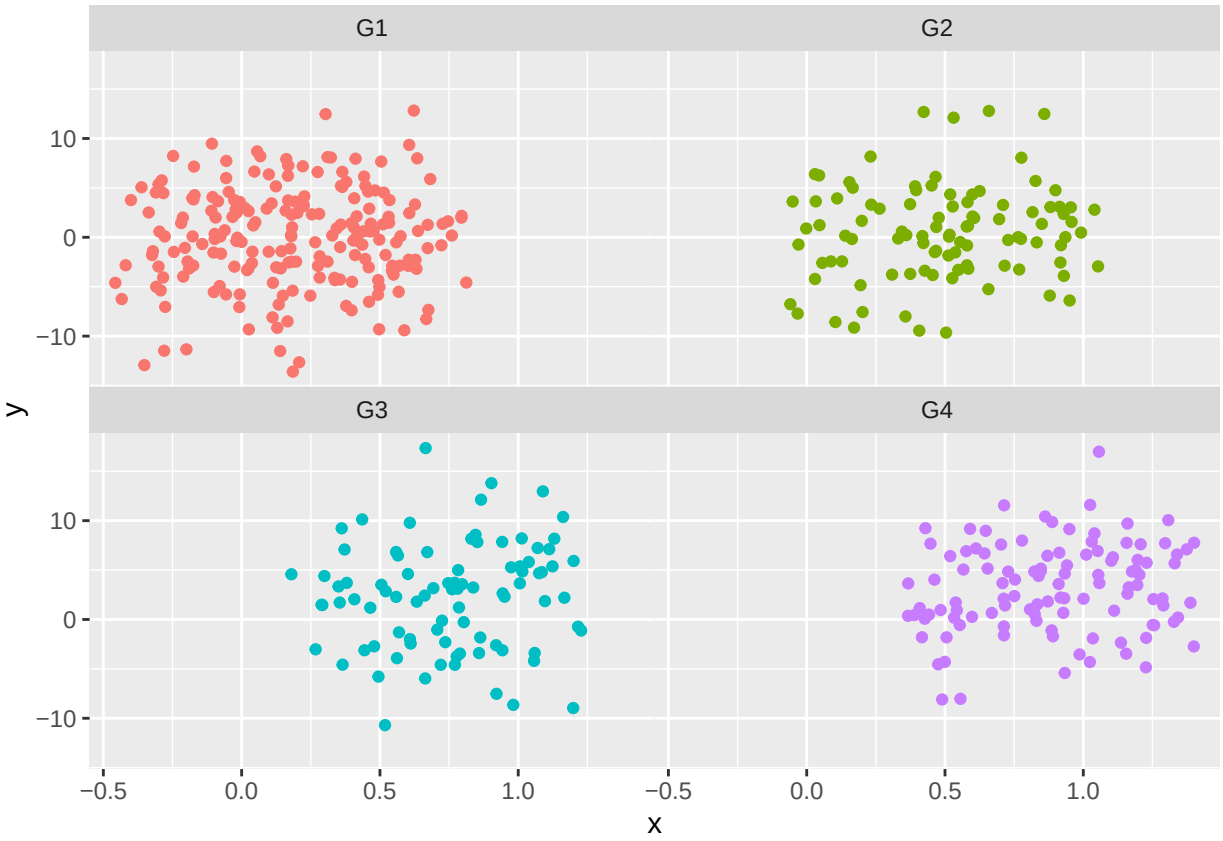
Remove the background color of the strips of a ggplot2 facet plot

Increase or remove the space between the panels

By default, there is a small space between the panels. You can increase or decrease that space passing a unit to the panel.spacing argument of the theme function.

```r
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)
```

```
# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~group) +
  theme(panel.spacing = unit(0, 'points'))
```



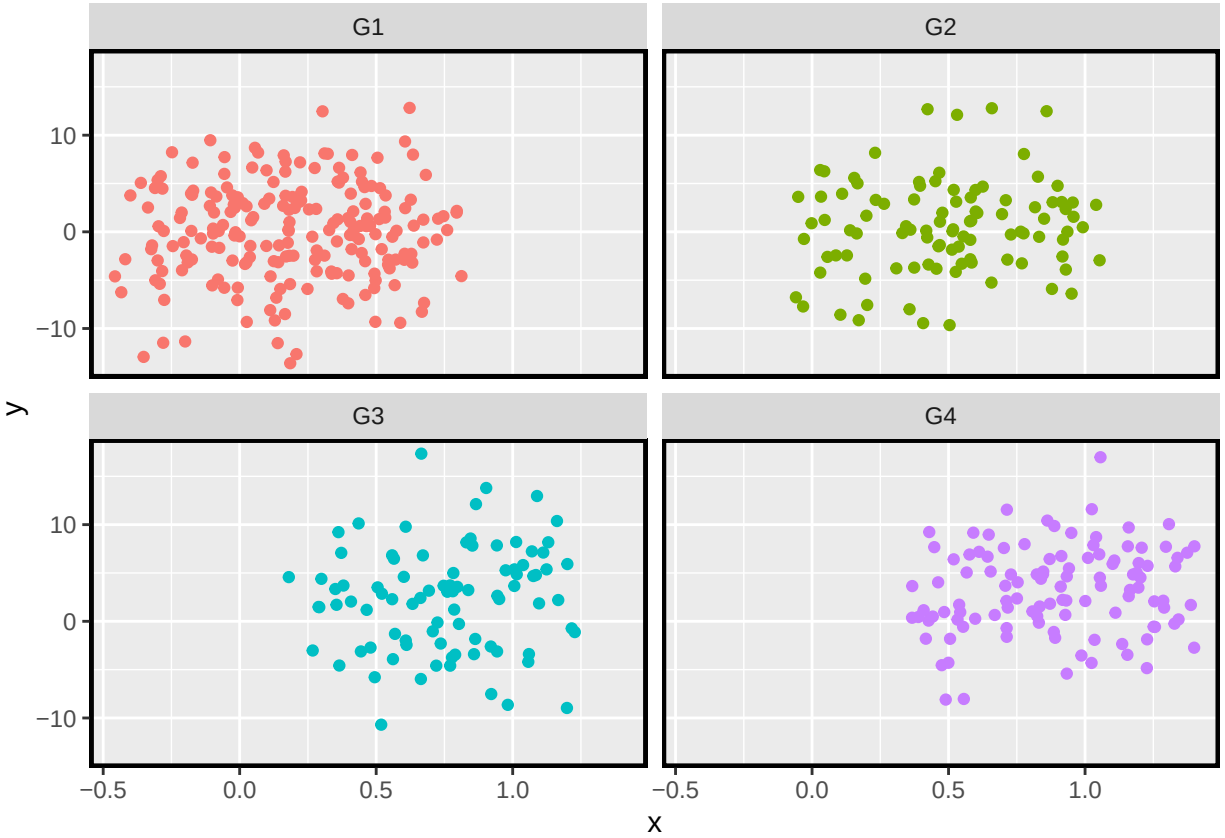
Remove or increase the space between facet panels in ggplot2

Add a border to the panels

Recall that you can also customize other parameters of the plots, such as adding a border to the panels. You can achieve this with panel.border, following the example below.

```
library(ggplot2)

# Data simulation
set.seed(4)
x <- runif(500)
y <- 4 * x ^ 2 + rnorm(length(x), sd = 5)
group <- ifelse(x < 0.4, "G1", ifelse(x < 0.6, "G2", ifelse(x < 0.8, "G3", "G4")))
x <- x + runif(length(x), -0.5, 0.5)

# Data frame
df <- data.frame(x = x, y = y, group = group)

# Ribbon of plots
ggplot(df, aes(x = x, y = y, color = group)) +
  geom_point(show.legend = FALSE) +
```

```
  facet_wrap(~group) +
  theme(panel.border = element_rect(fill = "transparent", # Needed to add the border
                                    color = "black", linewidth = 1.5))
```



Add a border to the panels of a facet plot in ggplot2

Coordinate systems in ggplot2

Cartesian coordiantes with coord_cartesian Fixed coordinates (equal scales) with coord_fixed Flipping the axes with coord_flip Transformations with coord_trans Polar coordinates with coord_polar Map projections with coord_quickmap and coord_map Coordinate systems in ggplot2 can be divided into two categories: linear (coord_cartesian, coord_fixed, coord_flip) and non-linear (coord_trans, coord_polar, coord_quickmap, coord_map) coordinate systems. These systems will be reviewed in this tutorial.

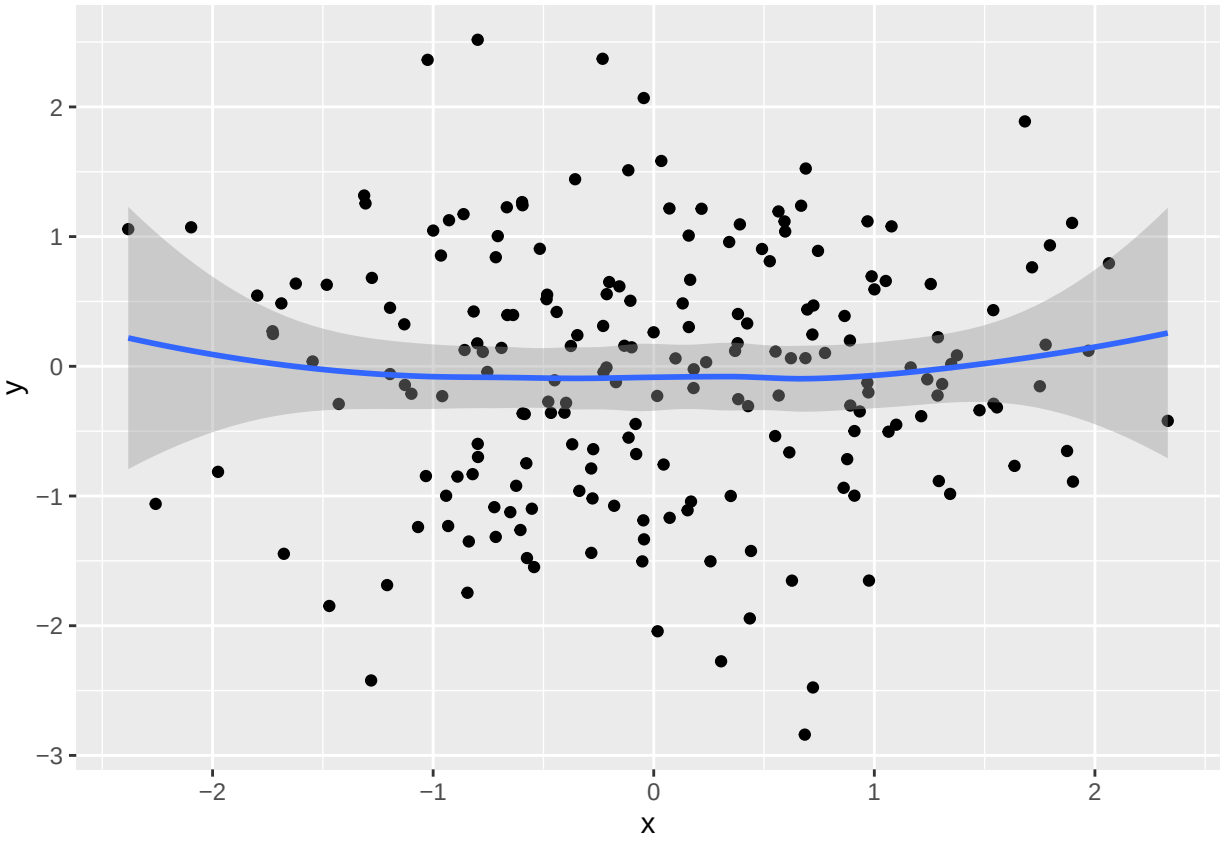
Cartesian coordiantes with coord_cartesian By default, ggplot2 charts have cartesian coordinate. However, the coord_cartesian function is very useful for zooming the plots, because if you use scale_x_continuous or scale_y_continuous the data will change the underlying data and hence the calculated stats, as shown below.

Default cartesian coordinates

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200),
                 y = rnorm(200))
```

```
p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()

p
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Default ggplot2 coordinates

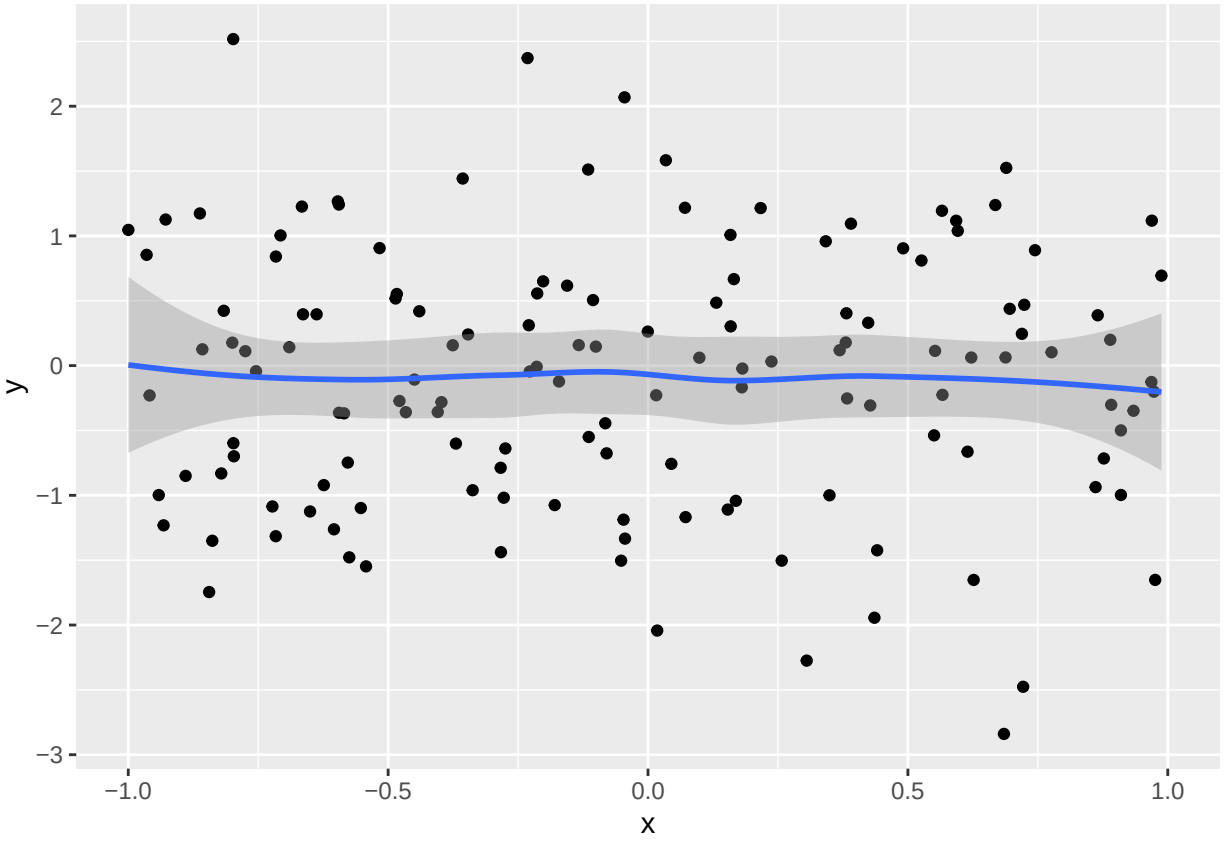
Using scale_x_continuous for zooming modifies the underlying data and the smooth estimation.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200),
                 y = rnorm(200))

p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()

p + scale_x_continuous(limits = c(-1, 1))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 59 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 59 rows containing missing values (`geom_point()`).
```

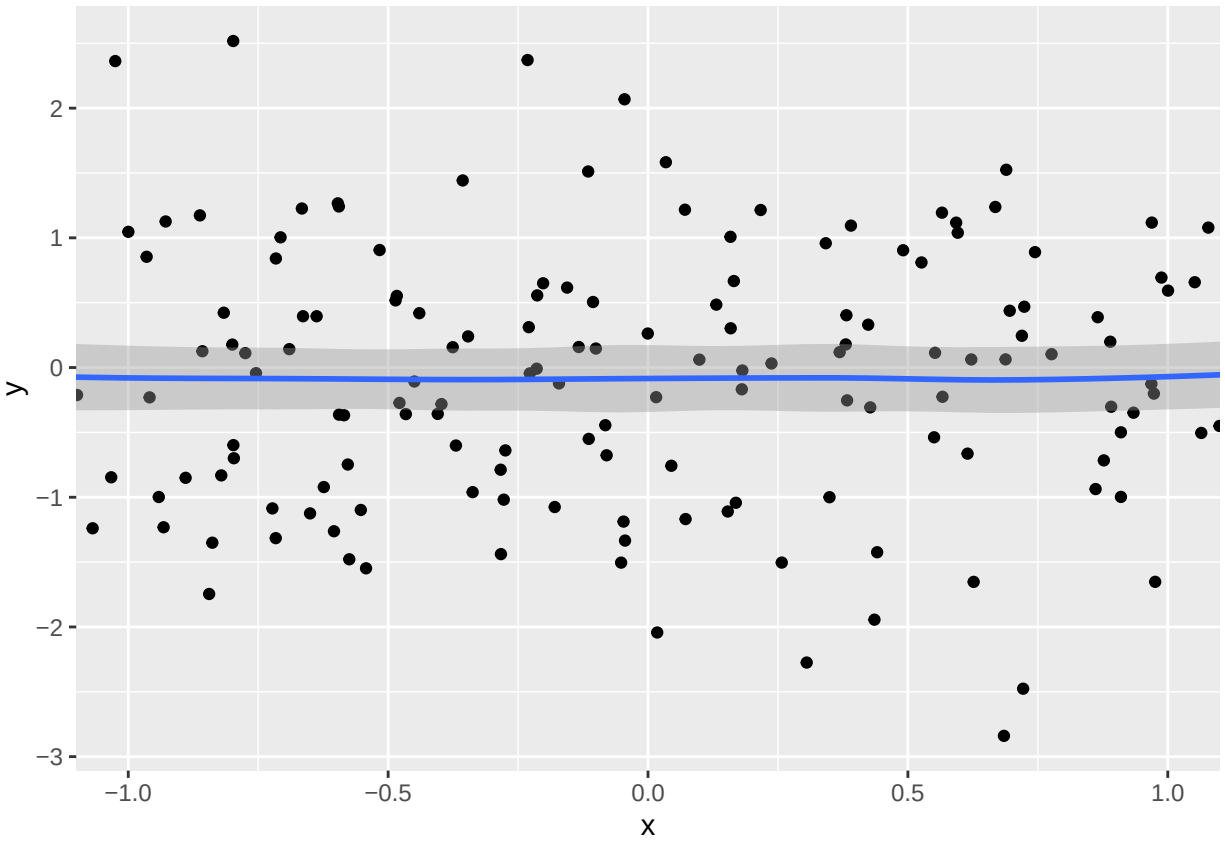


Zooming in ggplot2 with scale_x_continuous

However, if you use coord_cartesian you can set the xlim and ylim without modifying the original estimations and just zooming in or out.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200),
                 y = rnorm(200))

p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()

p + coord_cartesian(xlim = c(-1, 1))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



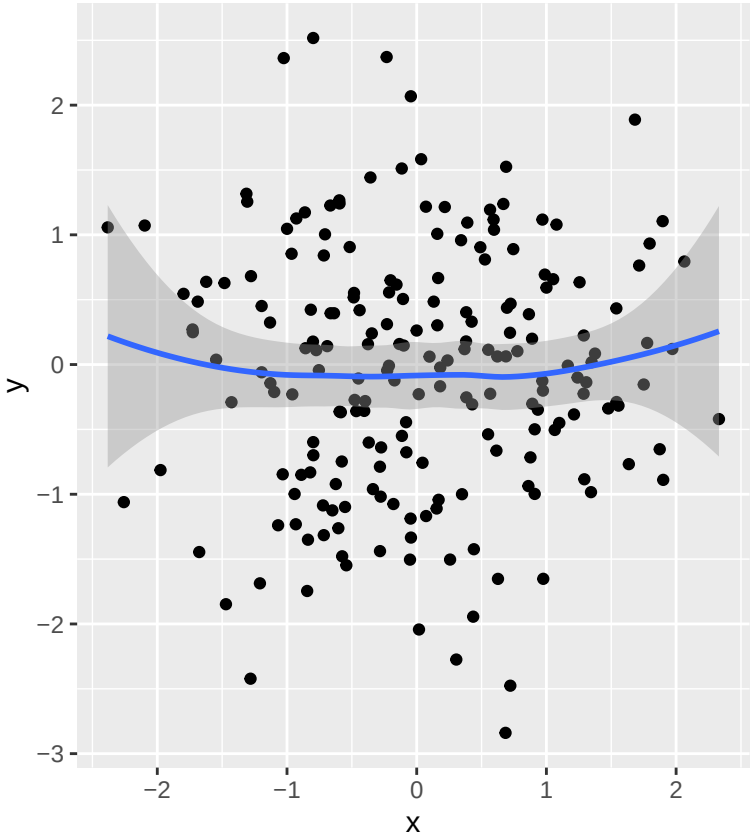
Zoom in ggplot2 with coord_cartesian

Fixed coordinates (equal scales) with coord_fixed The coord_fixed function is very useful in case you want a fixed aspect ratio for your plot regardless the size of the plotting device, this is, one unit along the X axis will be the same unit along the Y axis.

Fixed aspect ratio plot in ggplot2 with coord_fixed

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200),
                 y = rnorm(200))

p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()

p + coord_fixed()
```

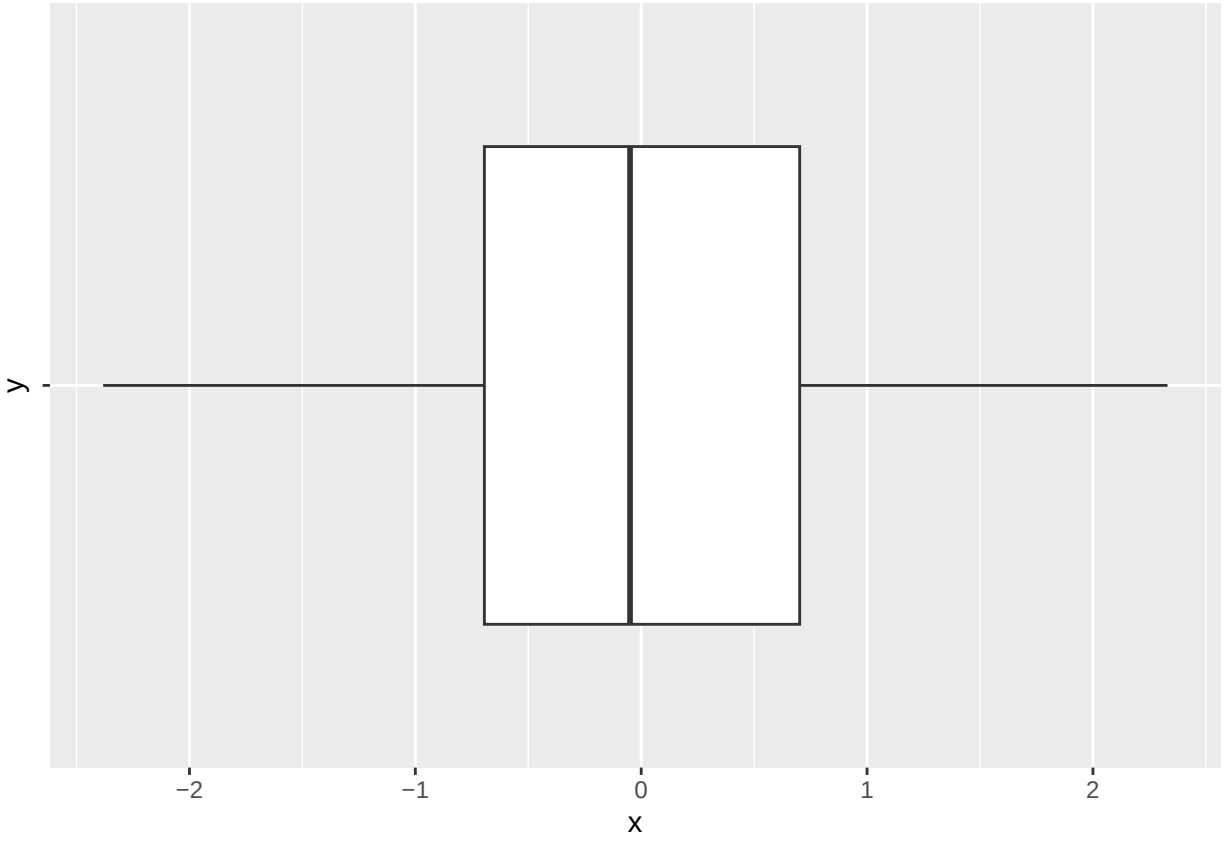
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Flipping the axes with coord_flip The coord_flip function rotates the axes in ggplot2, so if you have a vertical plot you can create the horizontal version and viceversa. This is specially useful for box plots, violin plots, ...

Default

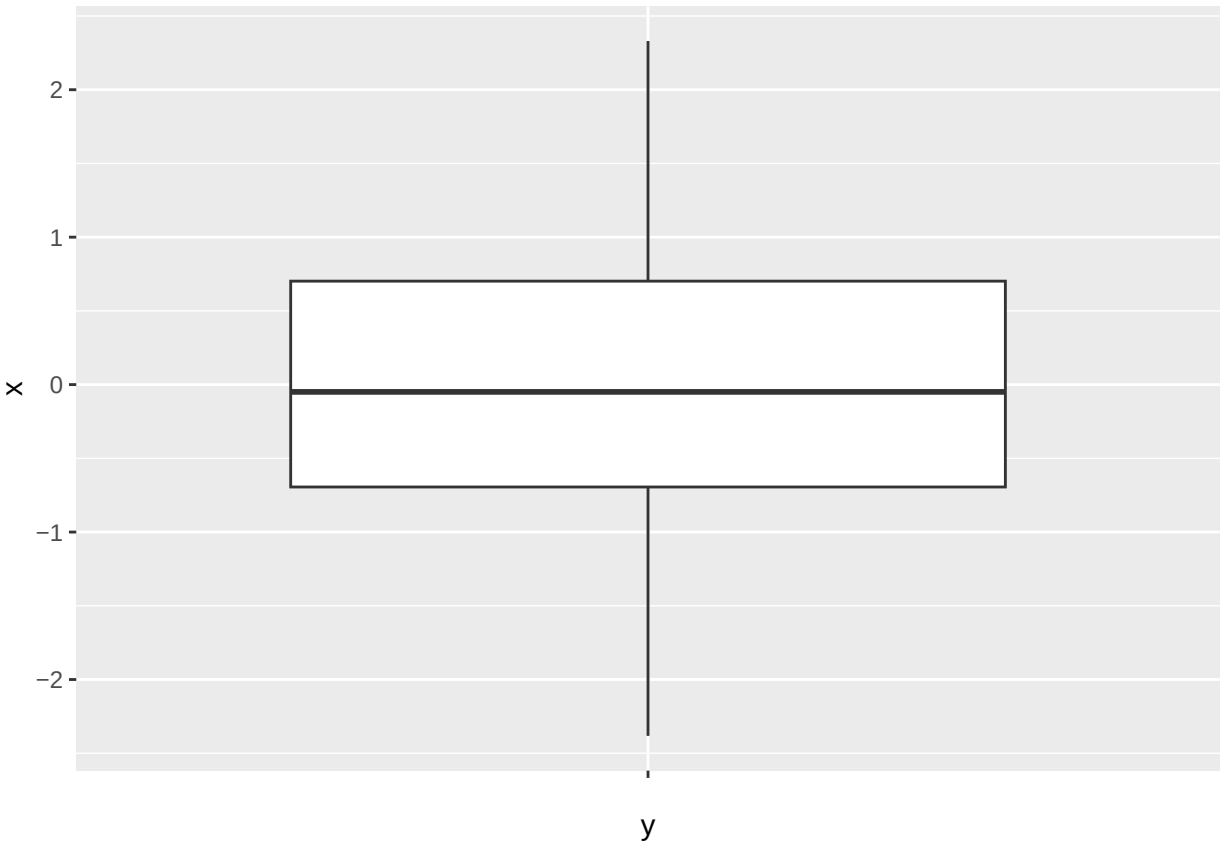
```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200))

p <- ggplot(df, aes(x = x, y = "")) +
  geom_boxplot()

p
```



Default ggplot2 orientation

Flipped axes

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = rnorm(200))

p <- ggplot(df, aes(x = x, y = "")) +
  geom_boxplot()

p + coord_flip()
```



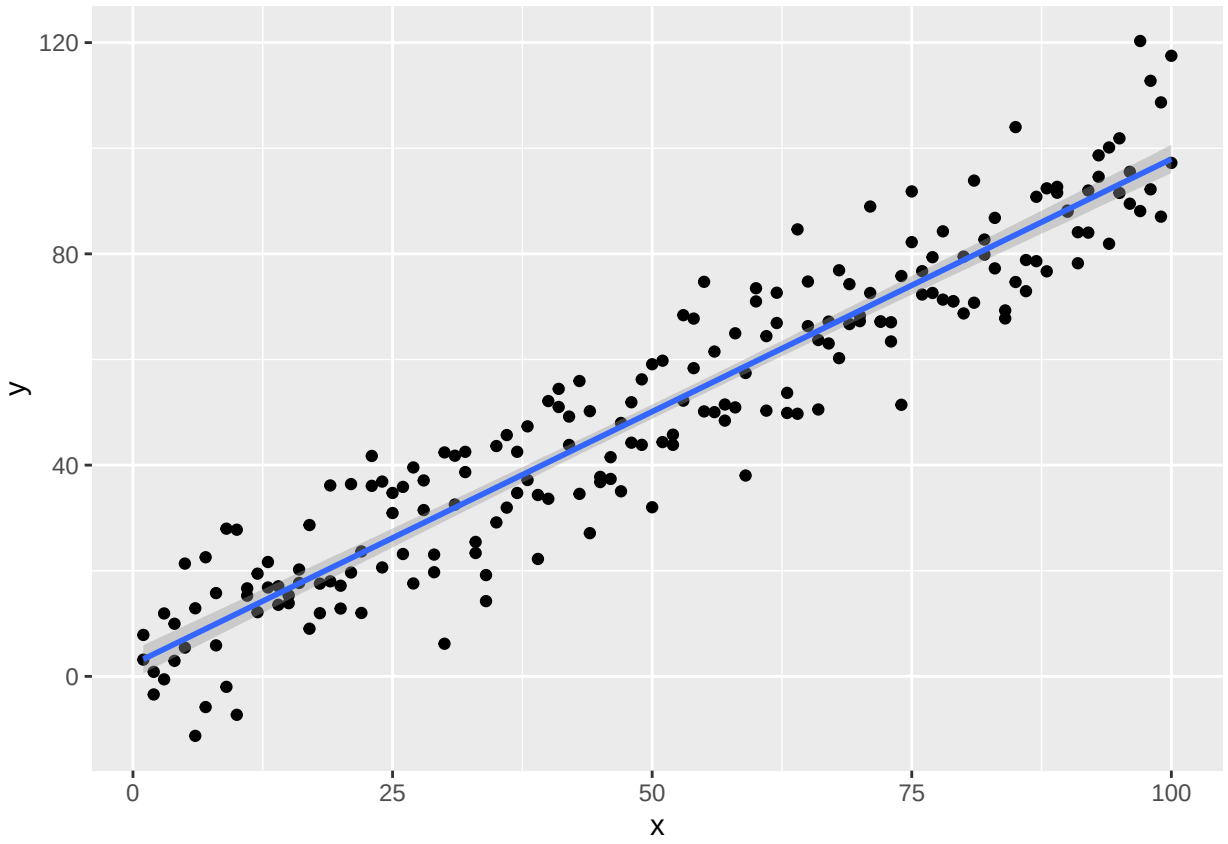
Flipping the axes in ggplot2 with coord_flip

Transformations with coord_trans The coord_trans function creates transformed cartesian coordinate systems. Note that using this functions is not the same as transforming the scale, due to when using coord_trans the transformation occurs after statistics, affecting the visual appearance of geoms. Consider the following sample plot:

Default ggplot2 transformation

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = 1:100,
                 y = 1:100 + rnorm(200, sd = 10))

p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm")

p
```

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

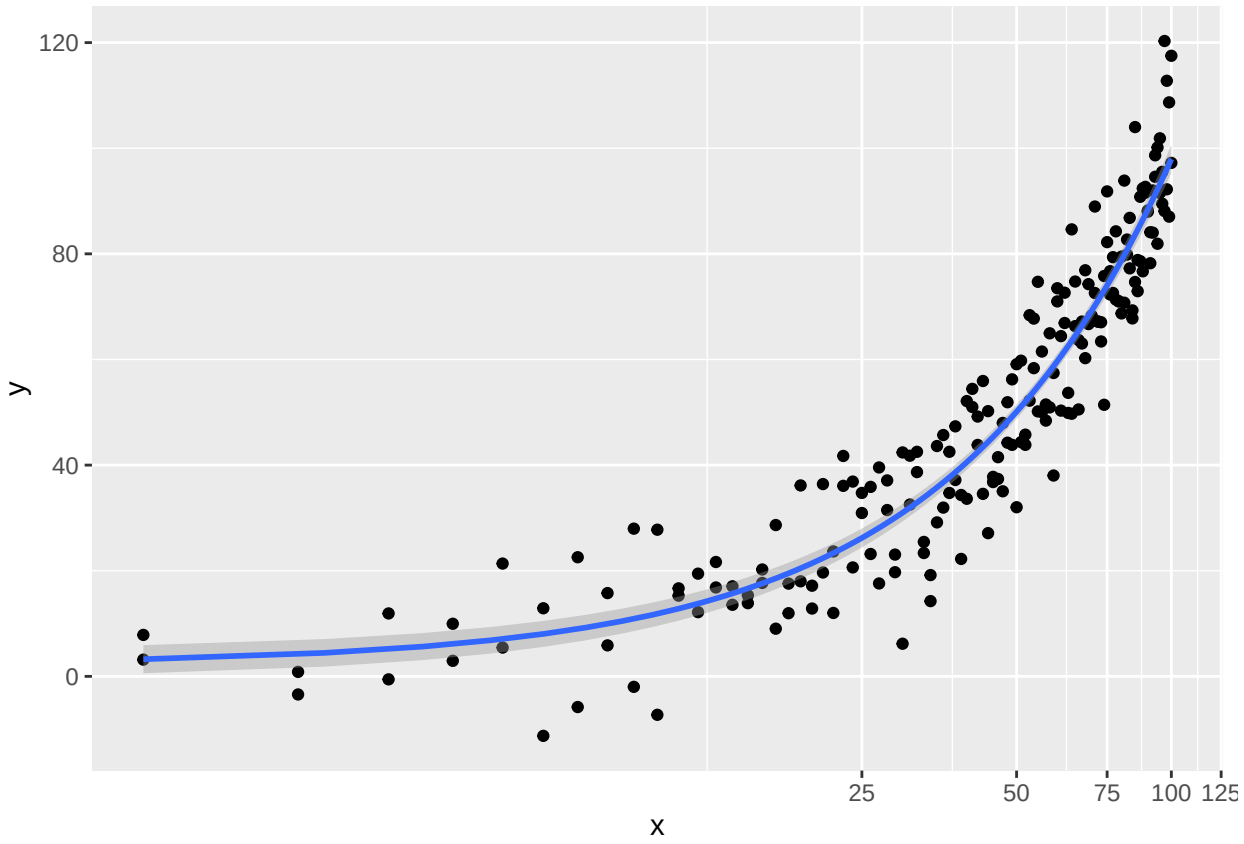


In the following example we are transforming the X-axis (x argument) with the log function, so the linear estimate will become a curve. Recall that you can pass any function to the axes as long as it makes sense.

Log transformation of the X-axis in ggplot2 with the coord_trans function

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = 1:100,
                 y = 1:100 + rnorm(200, sd = 10))

p <- ggplot(df, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm")

p + coord_trans(x = "log")
```

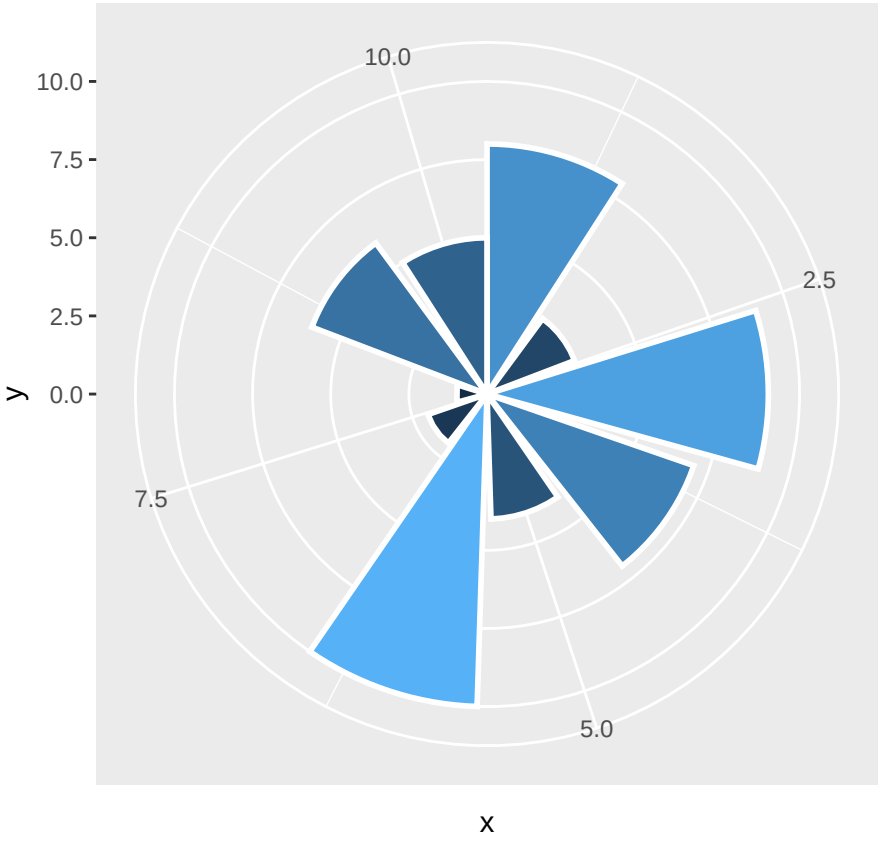
```
## `geom_smooth()` using formula = 'y ~ x'
```



Polar coordinates with coord_polar The polar coordinates can be created using coord_polar. This type of coordinates are commonly used for pie charts (which are stacked bar charts in polar coordinates), wind roses, radar charts, bullseye charts, ...

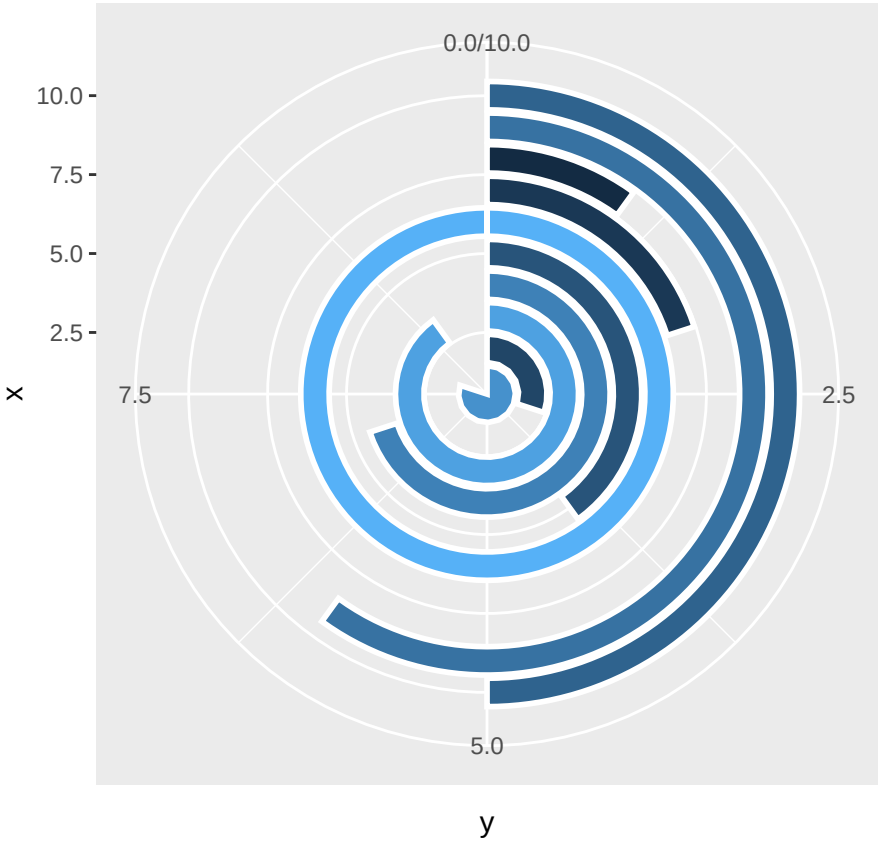
Note that by default the angle is mapped to the x variable, but you can set theta = "y" to map the angle to the y variable. You can also modify the direction of the plot with the direction argument (set -1 for anticlockwise). See the examples below for clarification.

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = 1:10,
                 y = sample(1:10))

p <- ggplot(df, aes(x = x, y = y, fill = y)) +
  geom_bar(stat = "identity", color = "white",
           lwd = 1, show.legend = FALSE)

p + coord_polar()
```



The coord_polar function in ggplot2

```
# install.packages("ggplot2")
library(ggplot2)

# Data
set.seed(4)
df <- data.frame(x = 1:10,
                 y = sample(1:10))

p <- ggplot(df, aes(x = x, y = y, fill = y)) +
  geom_bar(stat = "identity", color = "white",
           lwd = 1, show.legend = FALSE)

p + coord_polar(theta = "y")
```



Polar coordinates in ggplot2 with coord_polar

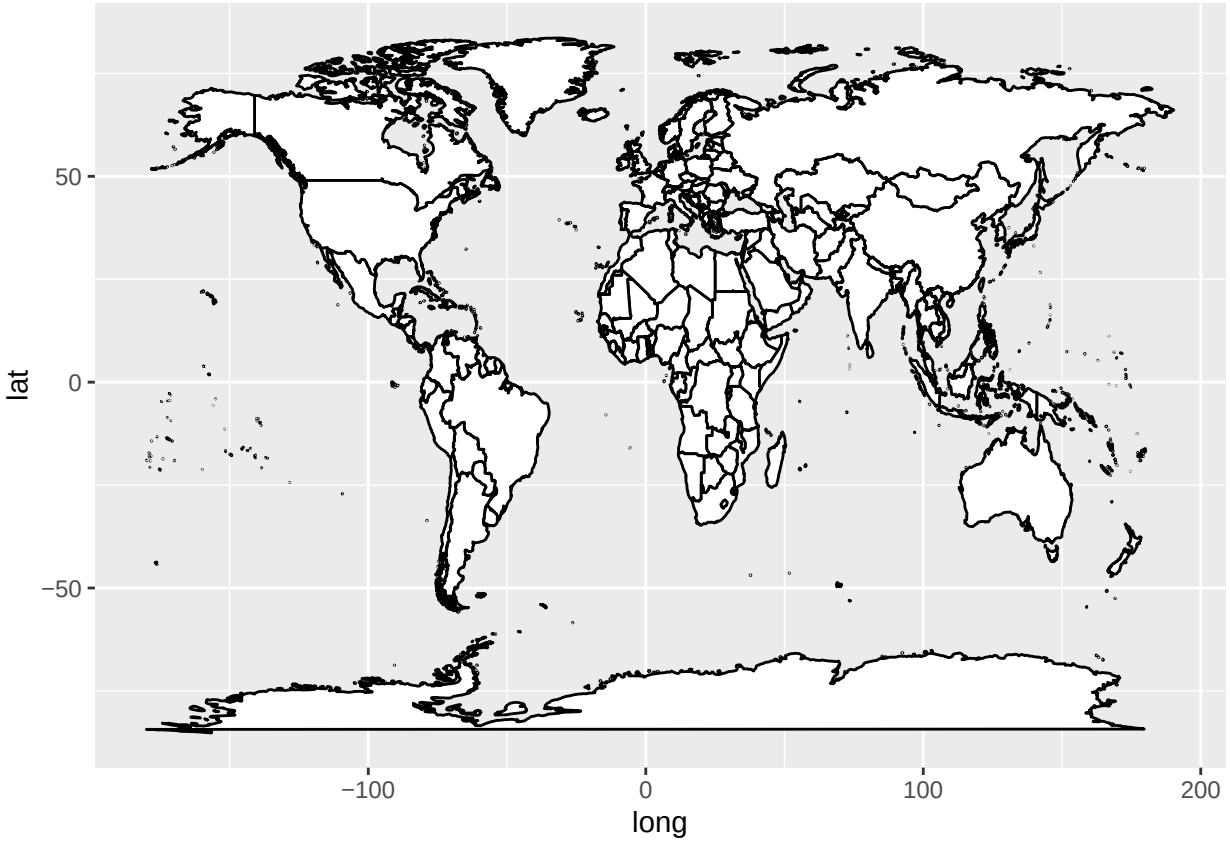
Map projections with coord_quickmap and coord_map The last system coordinates are related to map projections. You will need to have the mapproj package installed in your computer for applying the projections.

On the one hand, the coord_map function requires considerable computation, but will approximate the projection as much as possible. Note that there are lots of different projections available. Type ?mapproj::mapproject for the full list.

Default world map in ggplot2

Default map

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")

p <- ggplot(map_data("world"),
            aes(long, lat, group = group)) +
     geom_polygon(fill = "white", colour = 1)

p
```

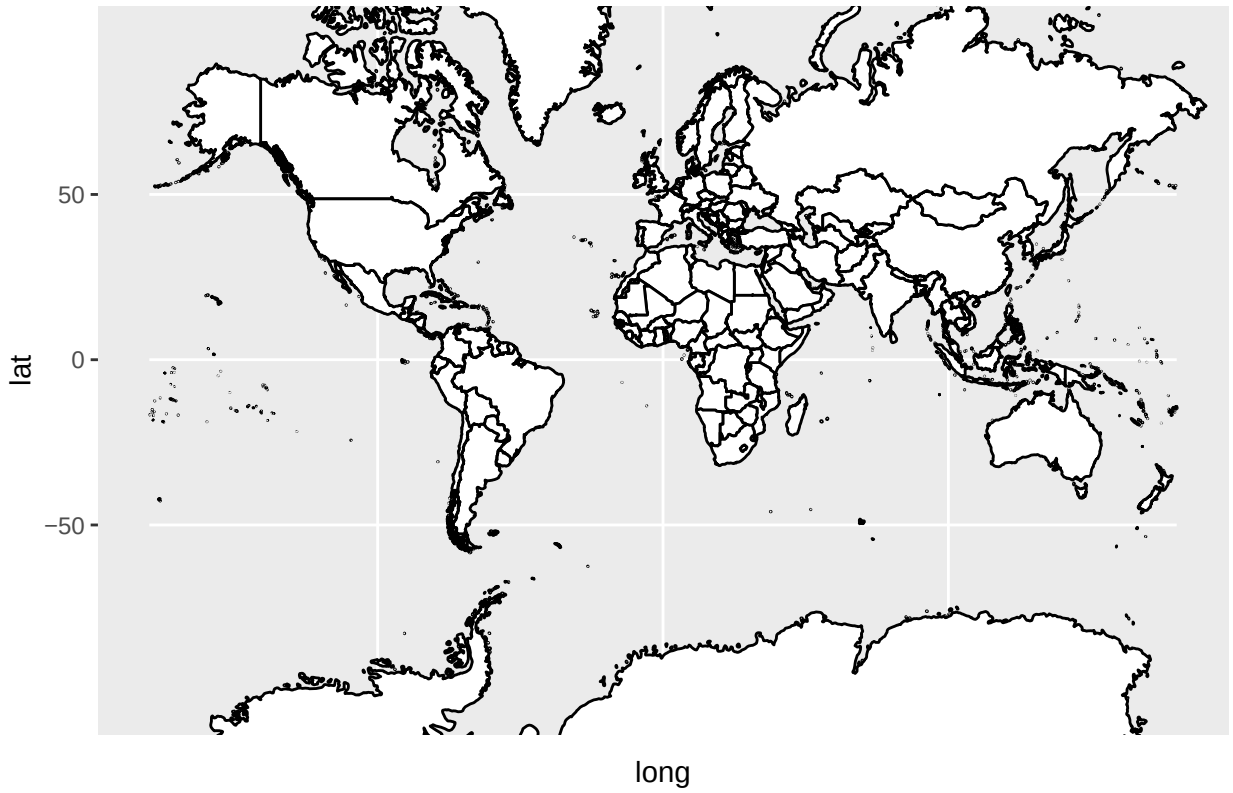


Mercator map projection in ggplot2 with coord_map

Mercator projection

The default projection is the well-known Mercator projection. Note that we modified the limits of the X-axis due to the function produces some unwanted horizontal lines. If you want to create this plot using coord_sf from sf package is a better solution.

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")

p <- ggplot(map_data("world"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p + coord_map(xlim = c(-180, 180))
```

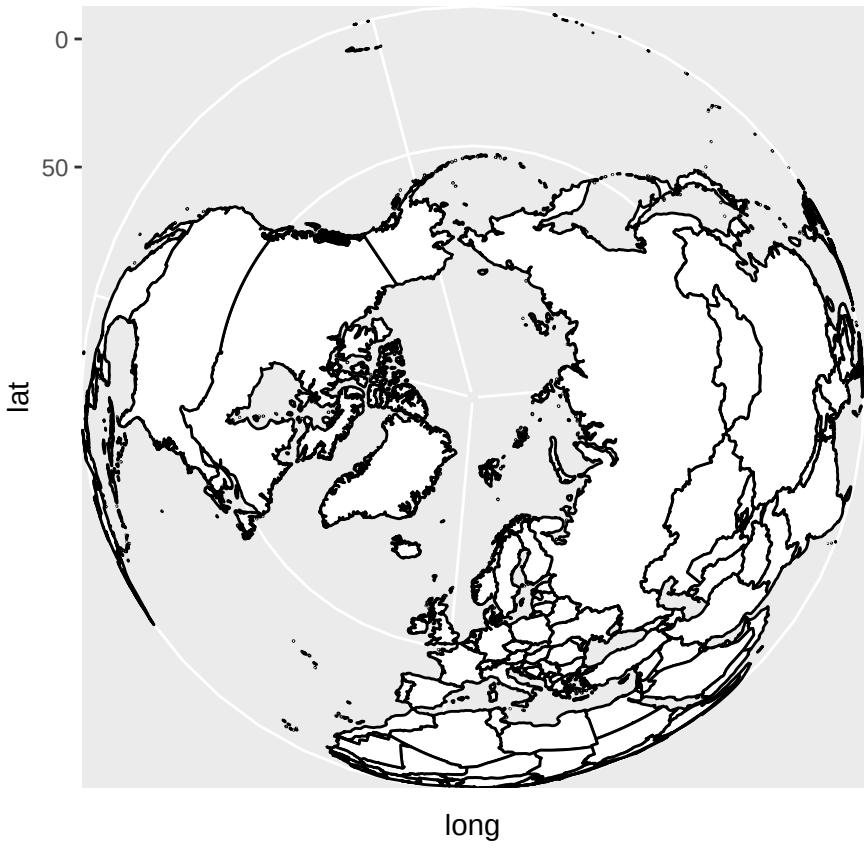



```
# p + sf::coord_sf() # Better solution
```

Orthographic map projection in ggplot2

Orthographic projection

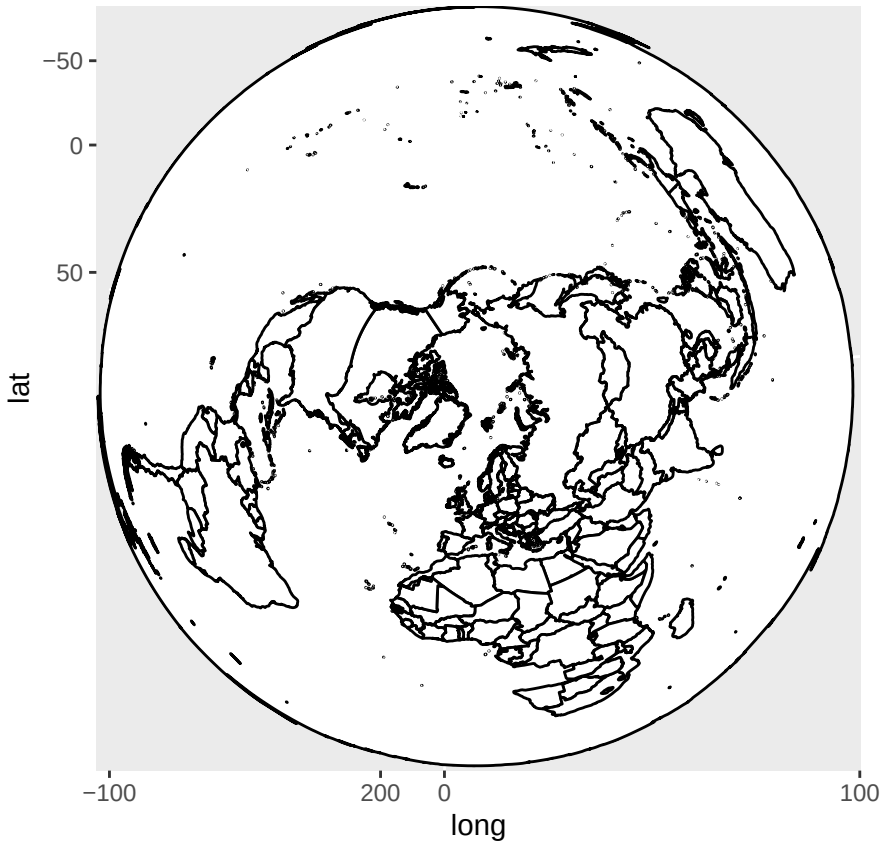
```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")

p <- ggplot(map_data("world"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p + coord_map("orthographic")
```



Fish eye map projection in ggplot2

Fish eye

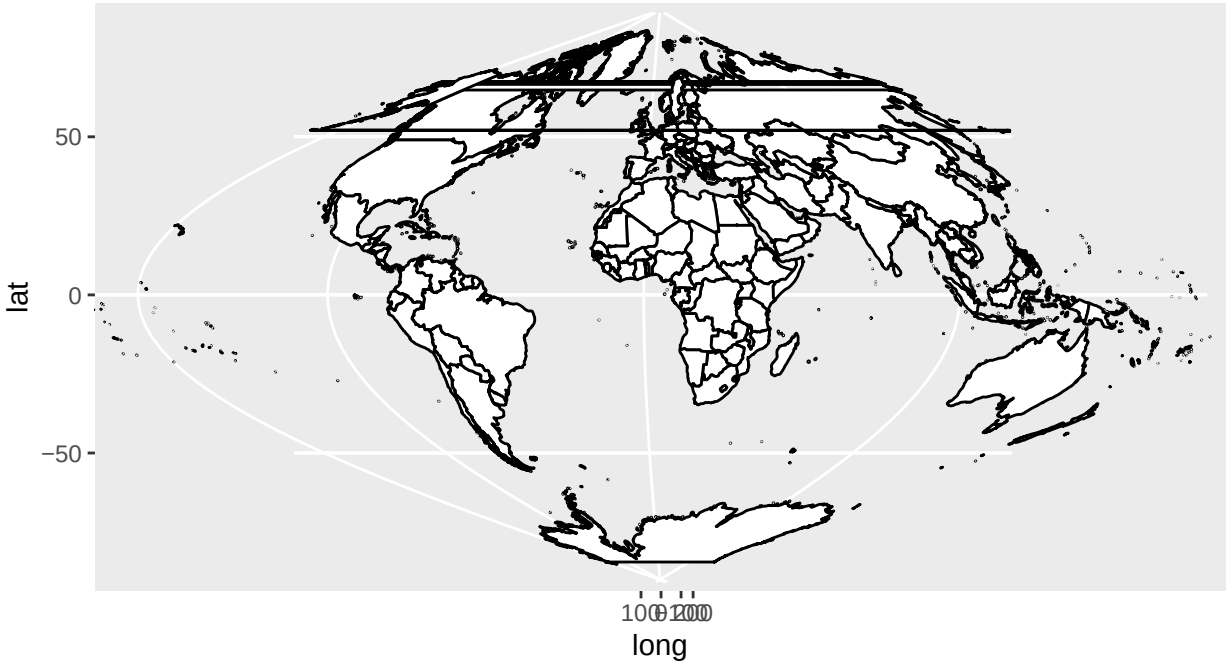
```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")

p <- ggplot(map_data("world"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p + coord_map("fisheye",
              n = 4) # Refractive index
```



Sinusoidal map projection in ggplot2

Sinusoidal

```
# install.packages("ggplot2")
library(ggplot2)
# install.packages("mapproj")

p <- ggplot(map_data("world"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p + coord_map("sinusoidal")
```

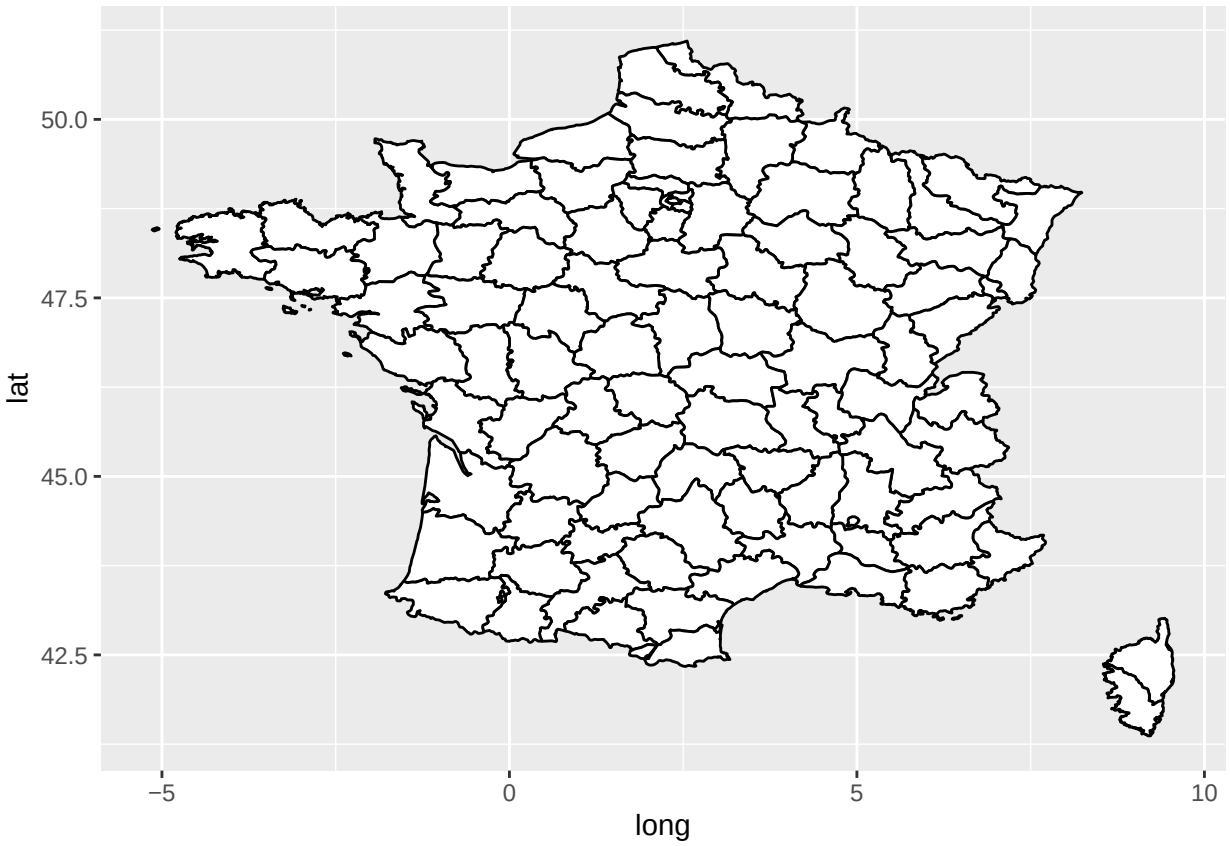


On the other hand, coord_quickmap also fixes the projection of a map, as shown below, but this function is a quick approximation for preserving straight lines which works best for small areas closer to the equator. Note that you can also apply any desired projection with the projection argument of the function, as in the previous examples.

Default map of France in ggplot2

Default map

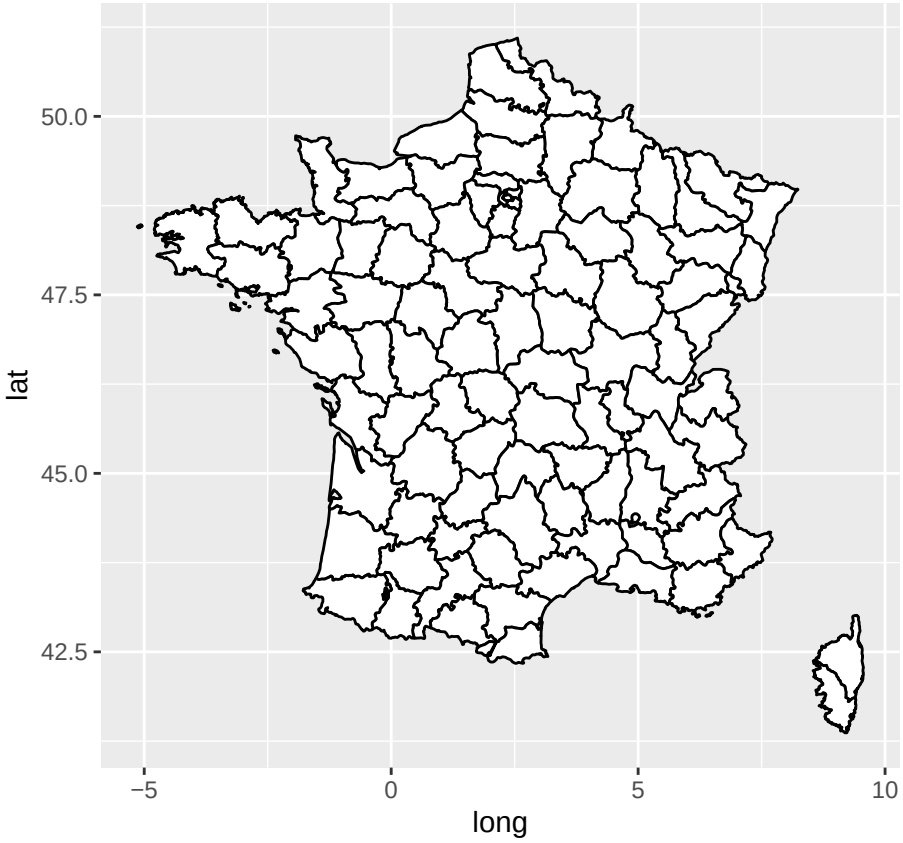
```
# install.packages("ggplot2")
library(ggplot2)

p <- ggplot(map_data("france"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p
```



Using the coord_quickmap function in ggplot2

Fixing the projection

```
# install.packages("ggplot2")
library(ggplot2)

p <- ggplot(map_data("france"),
            aes(long, lat, group = group)) +
      geom_polygon(fill = "white", colour = 1)

p + coord_quickmap()
```



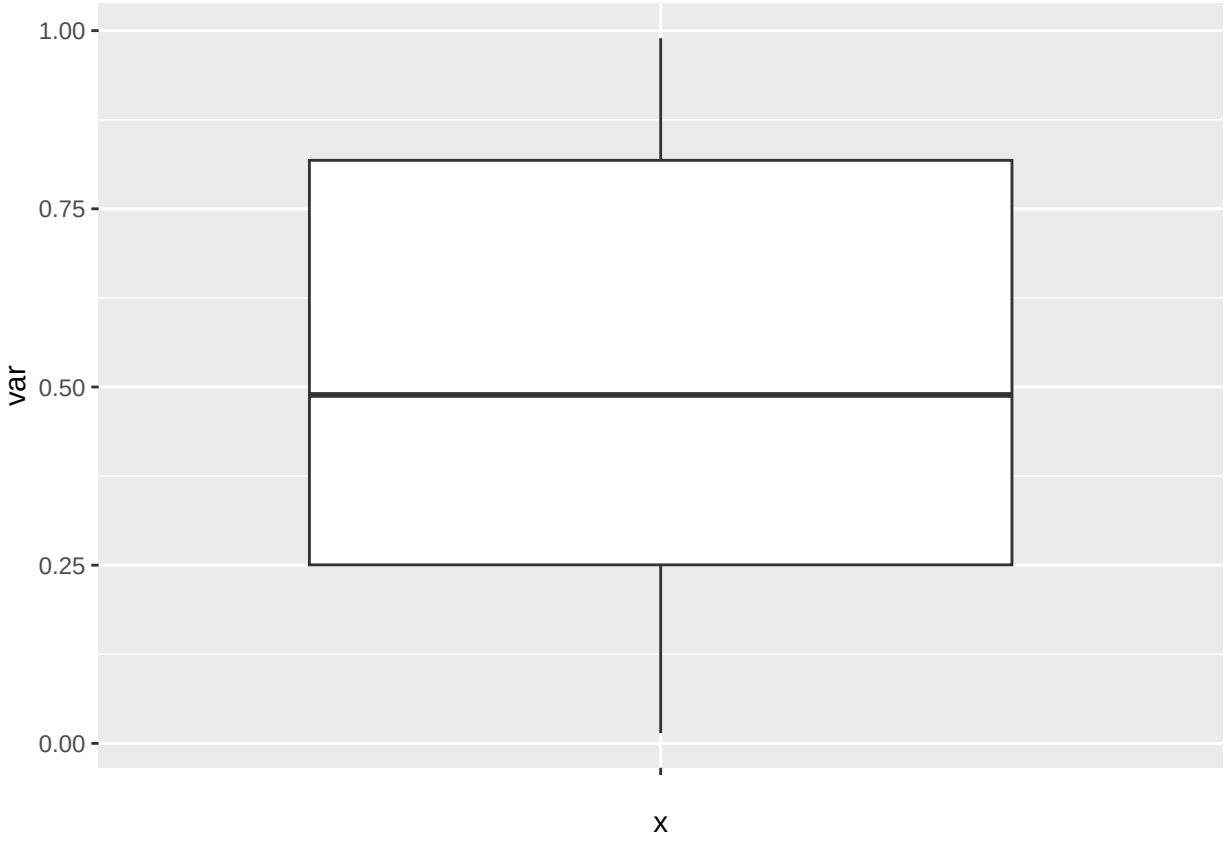
Combining plots in ggplot2

Sample plots patchwork Combining ggplot2 plots Controlling the layout Operators Titles and labels Adding more layers Adding base R plots, tables and text cowplot gridExtra The ggplot2 package doesn't provide a built-in functionality to arrange plots, but there are some packages that allow mixing ggplot2 objects into custom layouts. The most popular and easiest without doubt is patchwork, but there are some other alternatives, such as cowplot or gridExtra that we will also review in this tutorial.

Sample plots In order to make all the examples of this tutorial reproducible we are going to create four different plots and assign them to objects named p1, p2, p3 and p4. You can use any other names, but we will use these for convention.

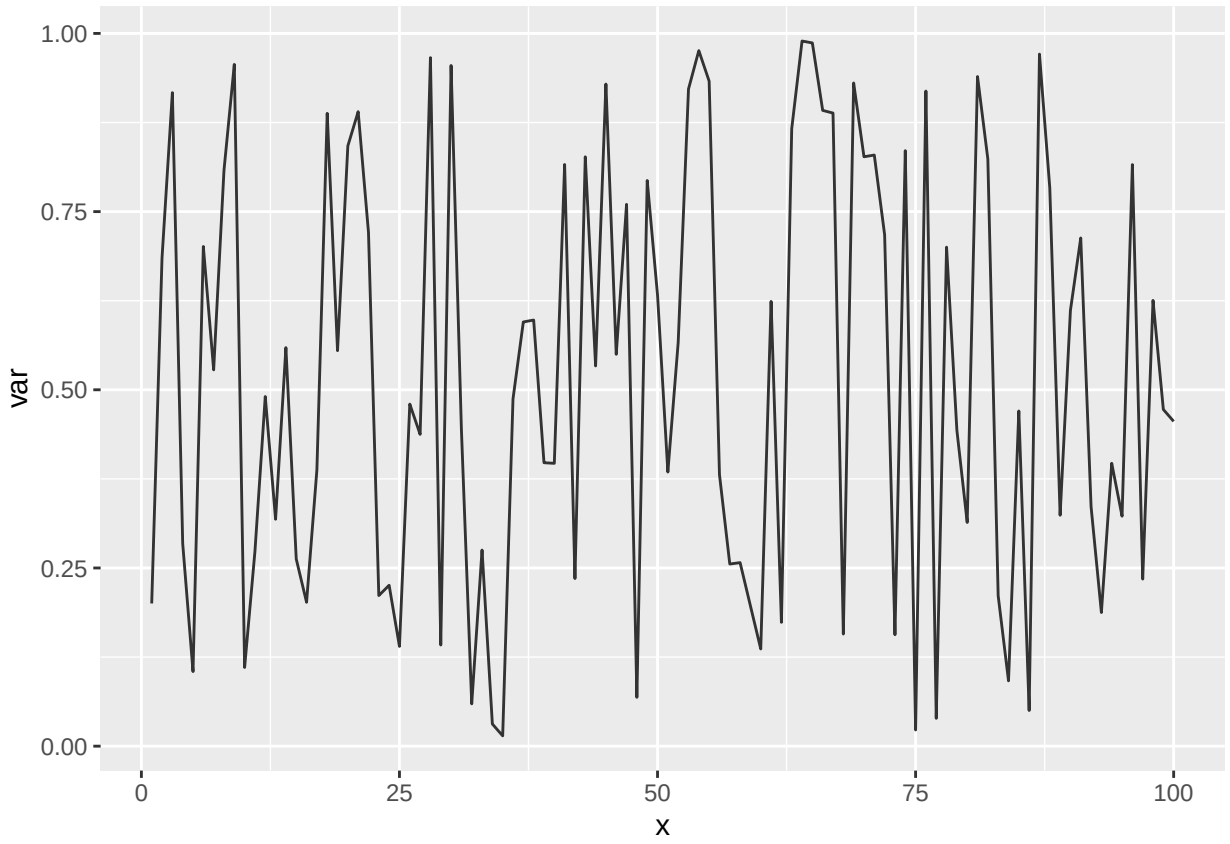
```
# install.packages("ggplot2")
library(ggplot2)

# Data simulation
set.seed(5)
x <- runif(100)
df <- data.frame(x = seq_along(x), var = x)

# Box plot
p1 <- ggplot(df, aes(x = "", y = var)) +
  geom_boxplot()

# Density plot
p2 <- ggplot() +
  stat_function(fun = dnorm, geom = "density",
```

```
                xlim = c(-3, 3), fill = "white")

# Line chart
p3 <- ggplot(df, aes(x = x, y = var)) +
  geom_line(color = "gray20")

# Scatter plot
p4 <- ggplot(df, aes(x = x, y = var)) +
  geom_point(color = "gray20")

# View the plots
p1
```

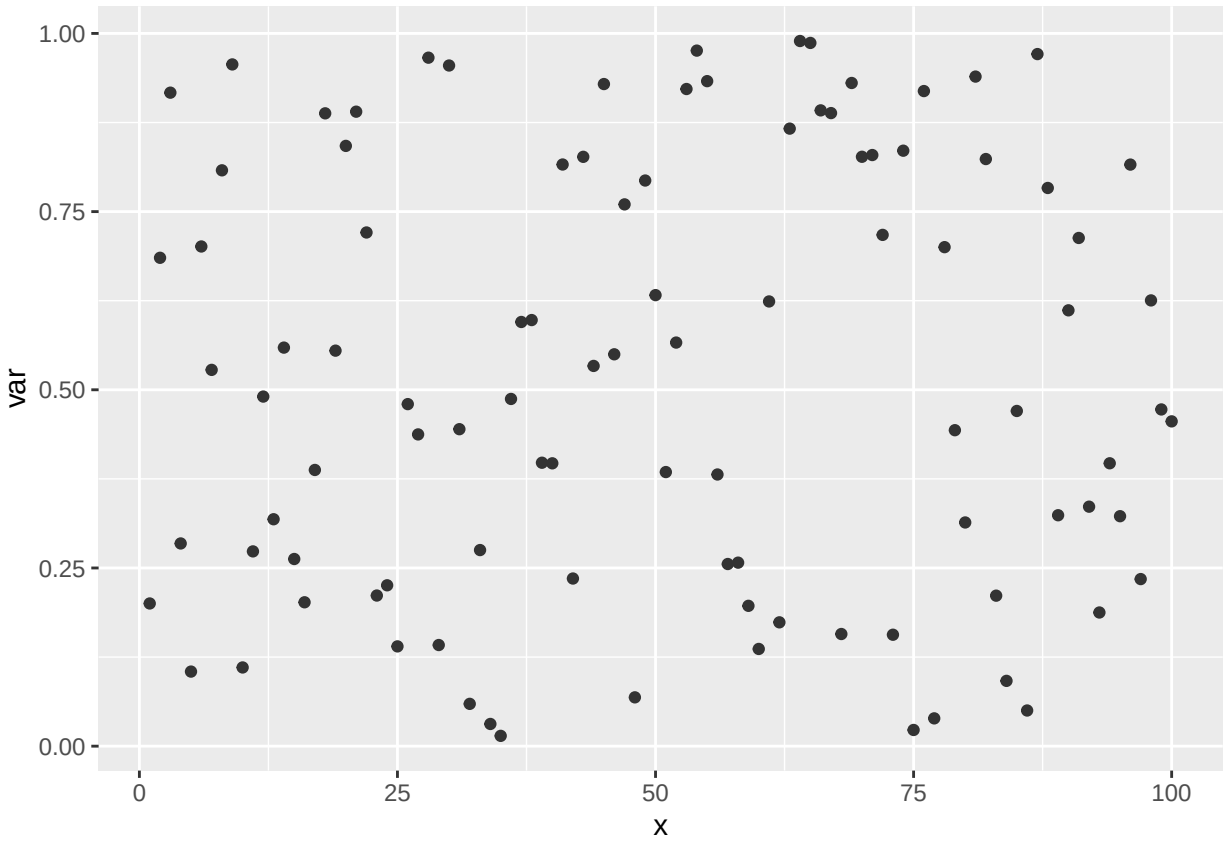



```
p2
```


```
p3
```



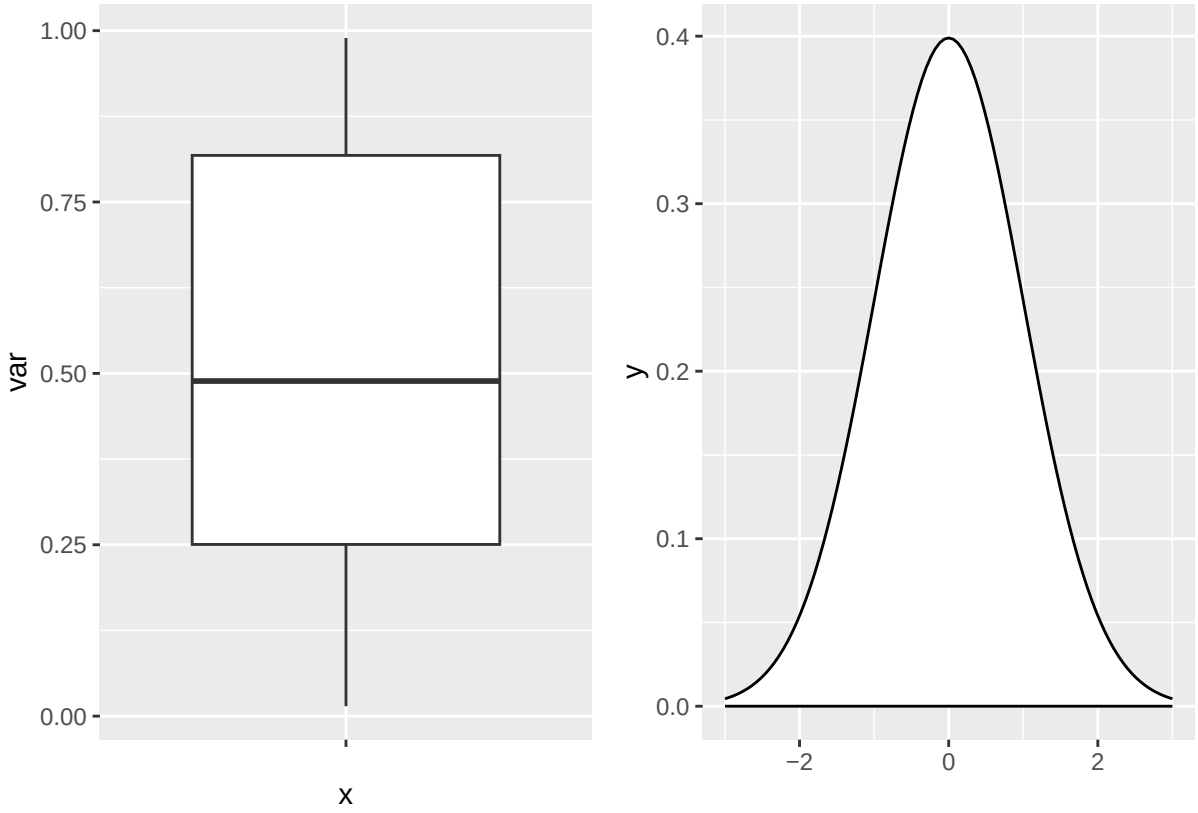

```
p4
```



Grid of ggplots

patchwork Combining ggplot2 plots patchwork is designed to combine ggplot2 plots into the same figure easily. The basic usage just consist on saving the plots into objects, loading the library and then using the + operator to combine the charts, as if you were adding a new layer.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Combine the plots
p1 + p2
```

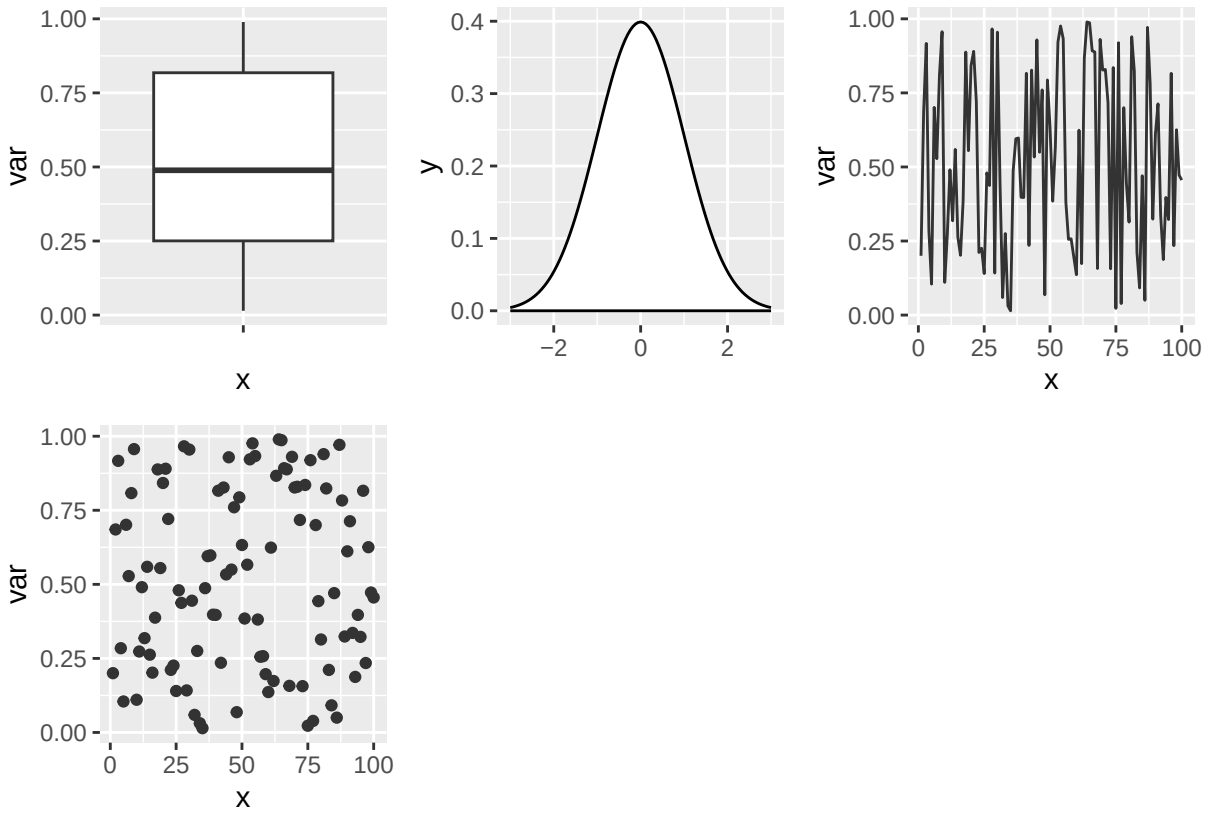


Combine two plots in ggplot2

patchwork will perfectly align the ggplot2 plots.

Controlling the layout When combining plots with +, patchwork will try to make the layout as square as possible, but if you want to customize the number of rows or columns of the figure you can use the plot_layout function. Note that you can also specify the relative widths and heights of the plots with widths and heights arguments.

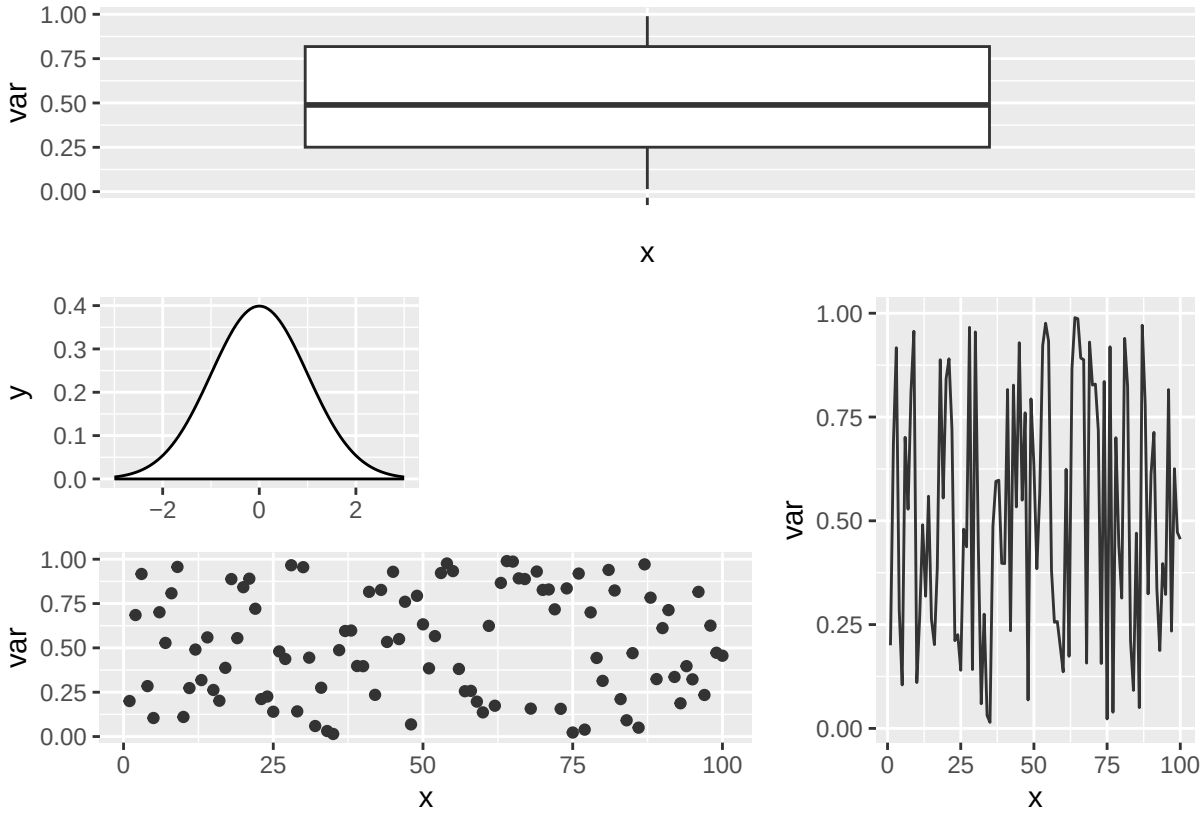
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Combine the plots
p1 + p2 + p3 + p4 +
  plot_layout(ncol = 3)
```



Arrange multiple plots with plot_layout

Probably the most interesting functionality of the plot_layout function is that you can create a custom layout design as shown in the example below, where 1, 2, 3 and 4 represents the locations for p1,p2, p3 and p4, respectively, and # represents an empty space. Recall that you can use numbers but also letters to represent the plot locations.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Custom design (# means empty area)
design <- "
  111
  2#3
  443
"

# Combine the plots with a custom layout
p1 + p2 + p3 + p4 +
  plot_layout(design = design)
```

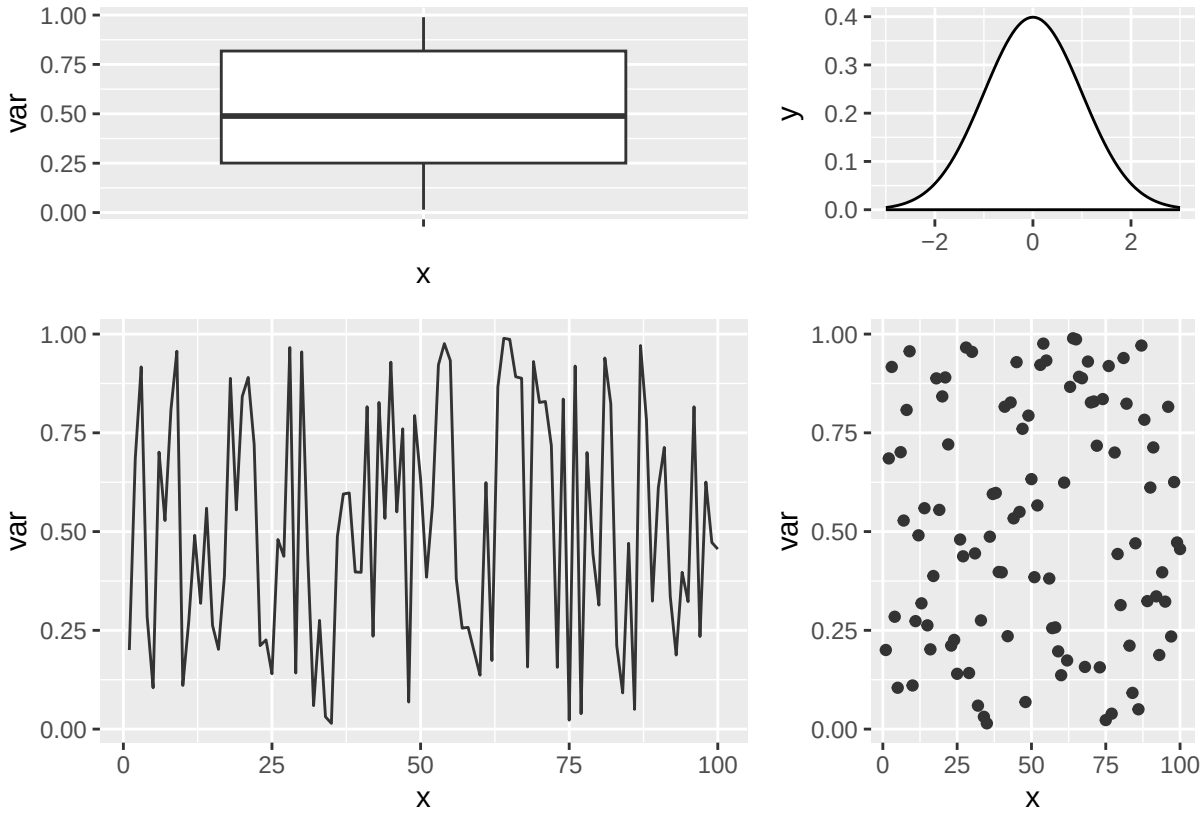


Set a custom layout design in ggplot2 with patchwork

The wrap_plots function

Sometimes you can't use the + operator programatically, so if you don't know the number of plots beforehand you can use the wrap_plots function and pass a list of plots to it. This function also allows specifying the number of rows and columns, the sizes and the custom layouts.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Combine the plots
wrap_plots(p1, p2, p3, p4,
           ncol = 2, nrow = 2,
           widths = c(1, 0.5), heights = c(0.5, 1))
```

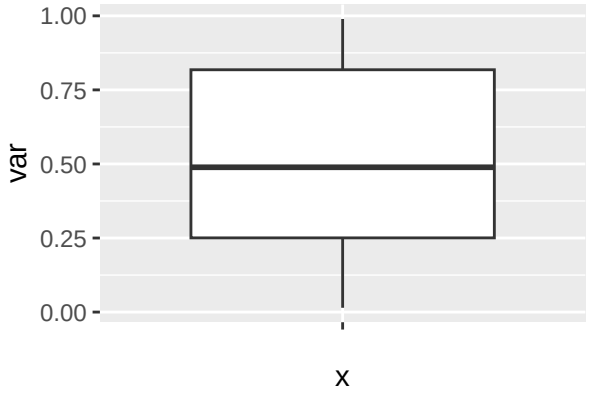


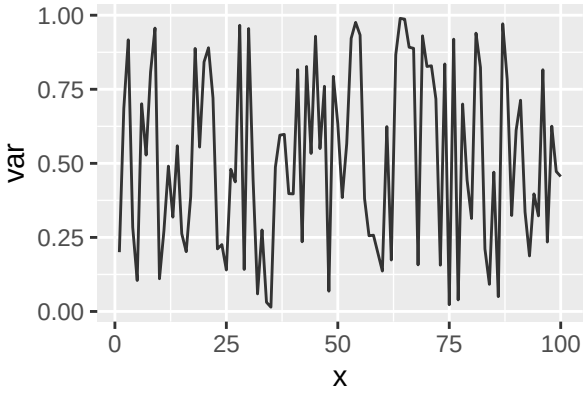
The wrap_plots function from patchwork

Adding spacers

When creating a custom layout you can use # to add spaces, as shown in one of the previous examples, but if you are using + there is also a function named plot_spacer to add spaces or gaps between plots.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Plots with spaces
p1 + plot_spacer() + plot_spacer() + p3
```





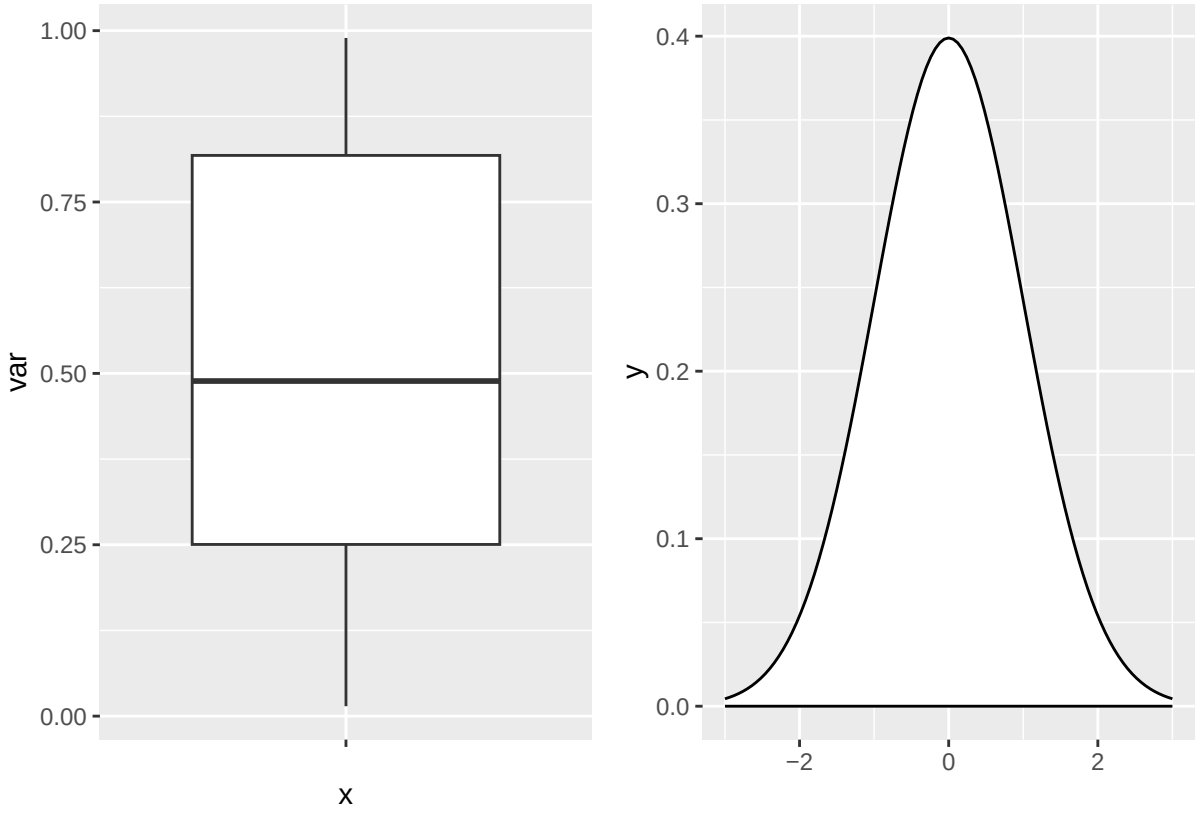
Add an space between a grid of ggplot2 plots

Operators The patchwork package also provides two handy operators to place plots beside each other or to stack them.

Arranging ggplot2 plots in rows (beside each other)

The | operator places plots in a row. This operator is similar to + when you have two plots but | will place all plots in a single row while + will try to create a square layout if possible.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Combine the plots in rows
p1 | p2
```



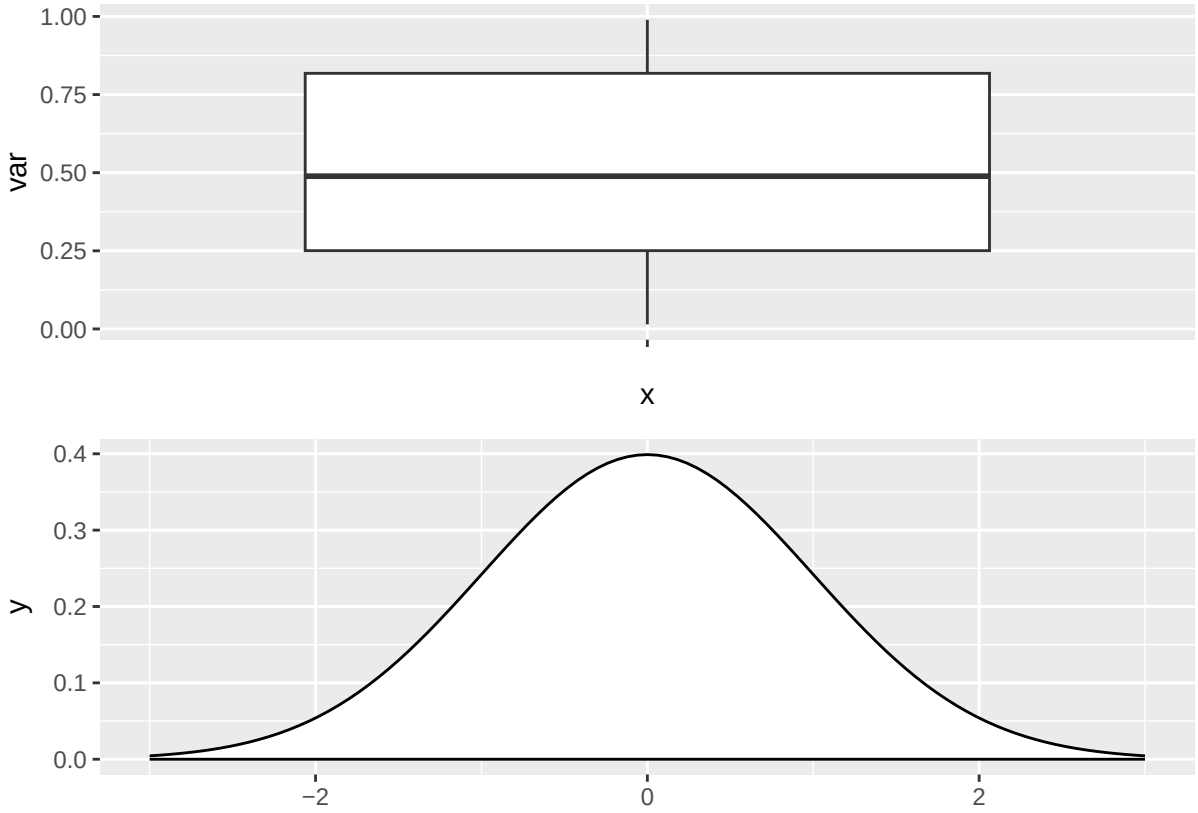

```
# Equivalent to:
# p1 + p2 + plot_layout(nrow = 1)
```

Arrange ggplot2 in rows beside each other

Arranging ggplot2 plots in columns (stacked)

The / operator stacks the ggplot2 plots into columns without the need of using the plot_layout function and specifying ncol = 1.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Combine the plots as column
p1 / p2
```



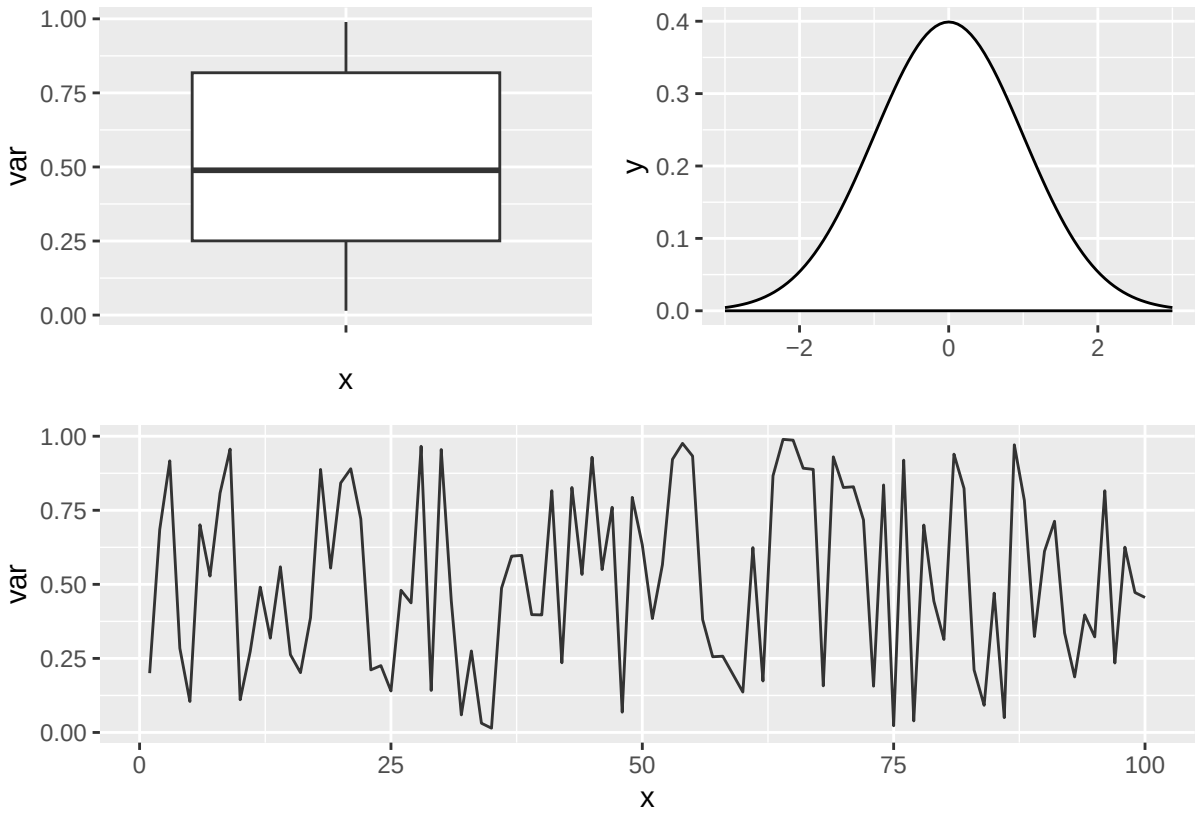

```
# Equivalent to:
# p1 + p2 + plot_layout(ncol = 1)
```

Stack ggplots using patchwork operators

Nesting operators

The previous operators can be nested to create complex layouts combining only a few operators. In the following example we are creating a layout with two plots at the top and one wider at the bottom.

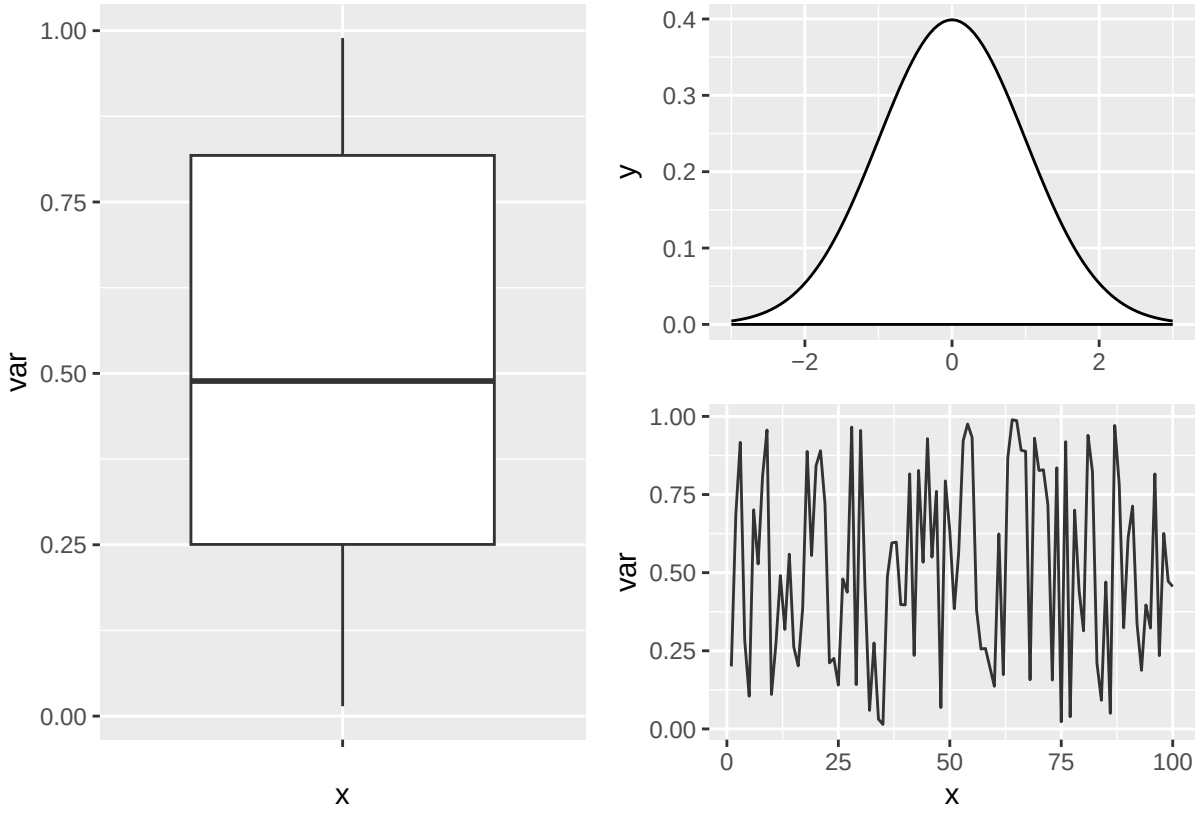
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Two plots on top and one at the bottom
(p1 | p2) / p3
```



Nest operators to create complex layouts of ggplot2 plots using the patchwork R package

The following example is similar to the previous, but with one plot at the left and two at the right.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# One plot at the left and two at the right
p1 | (p2 / p3)
```

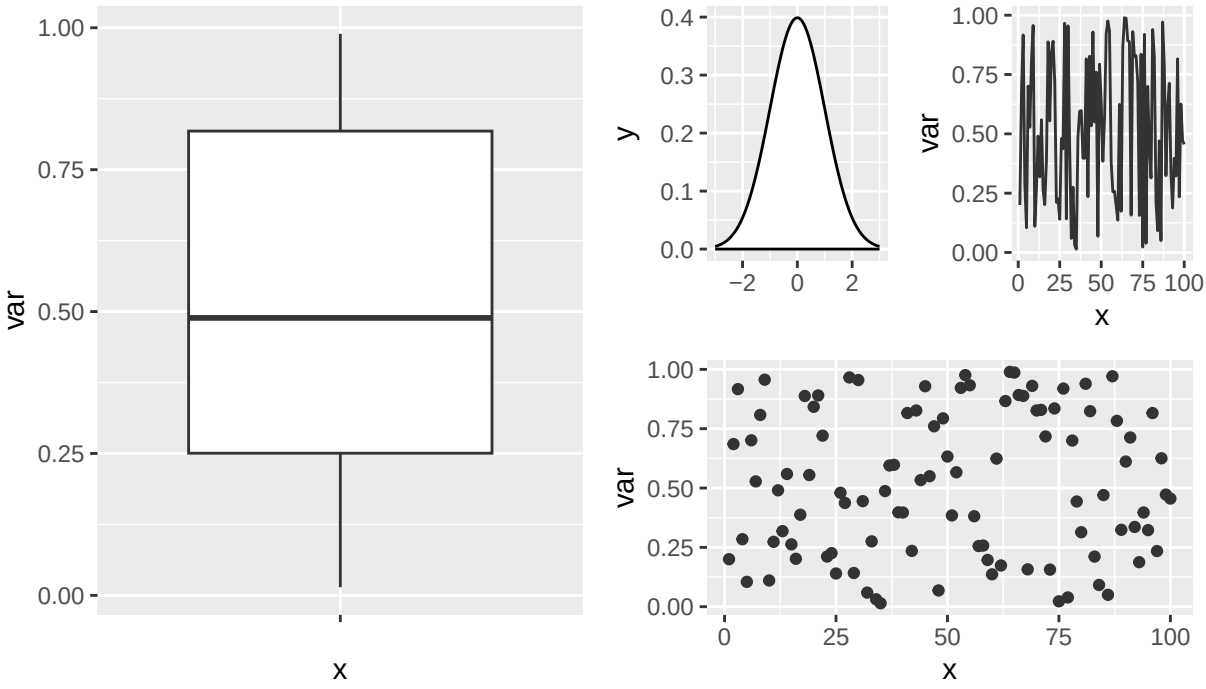


The patchwork R package

Titles and labels Title for all the plots

You can add a title to all plots with the plot_annotation function and the title argument. Note that you can also add a subtitle and a caption with the corresponding arguments.

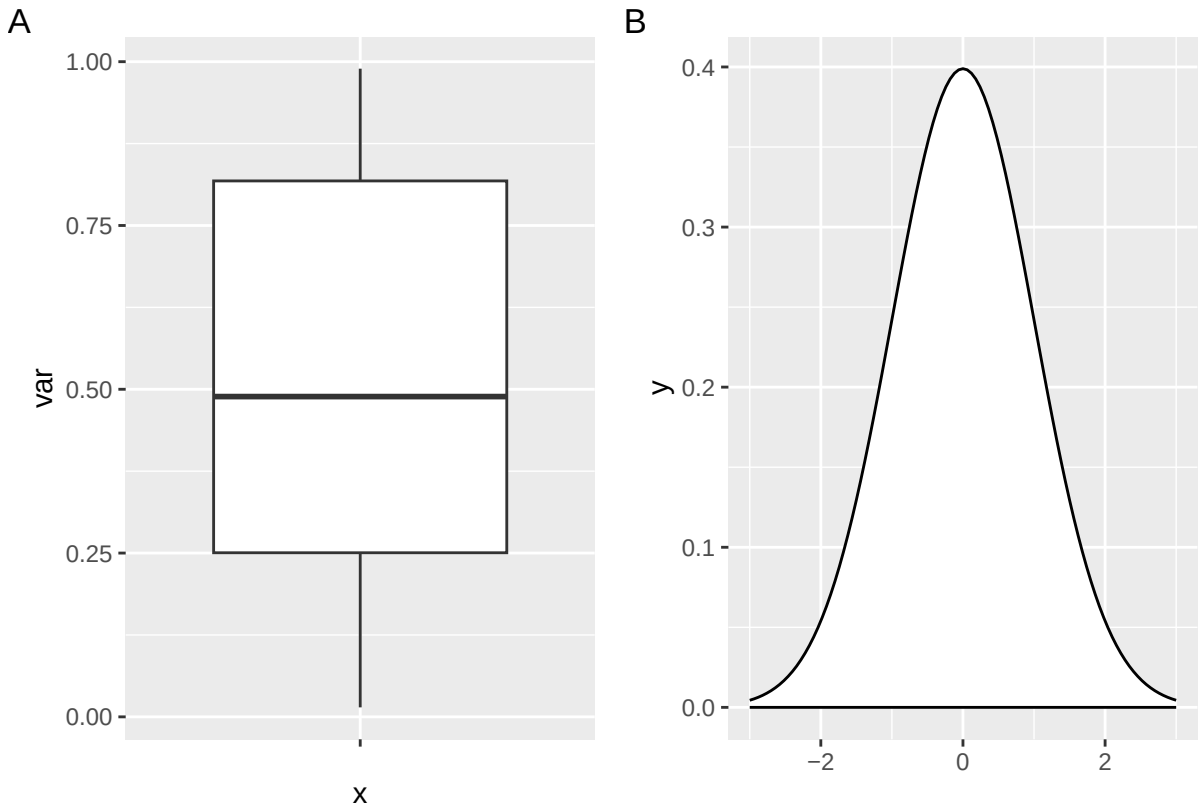
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Title for the combined plots
p1 + ((p2 | p3) / p4) +
plot_annotation(title = 'Title for all the plots',
                subtitle = "Subtitle",
                caption = "Caption")
```



Title for several plots in ggplot2

The plot_annotation function can also be used to label each plot individually with the tags_level argument. Possible options are “1” for numbers, “a” for lowercase letters, “A” for uppercase letters, “i” for lowercase Roman numerals, “I” for uppercase Roman numerals or a vector with your own tags.

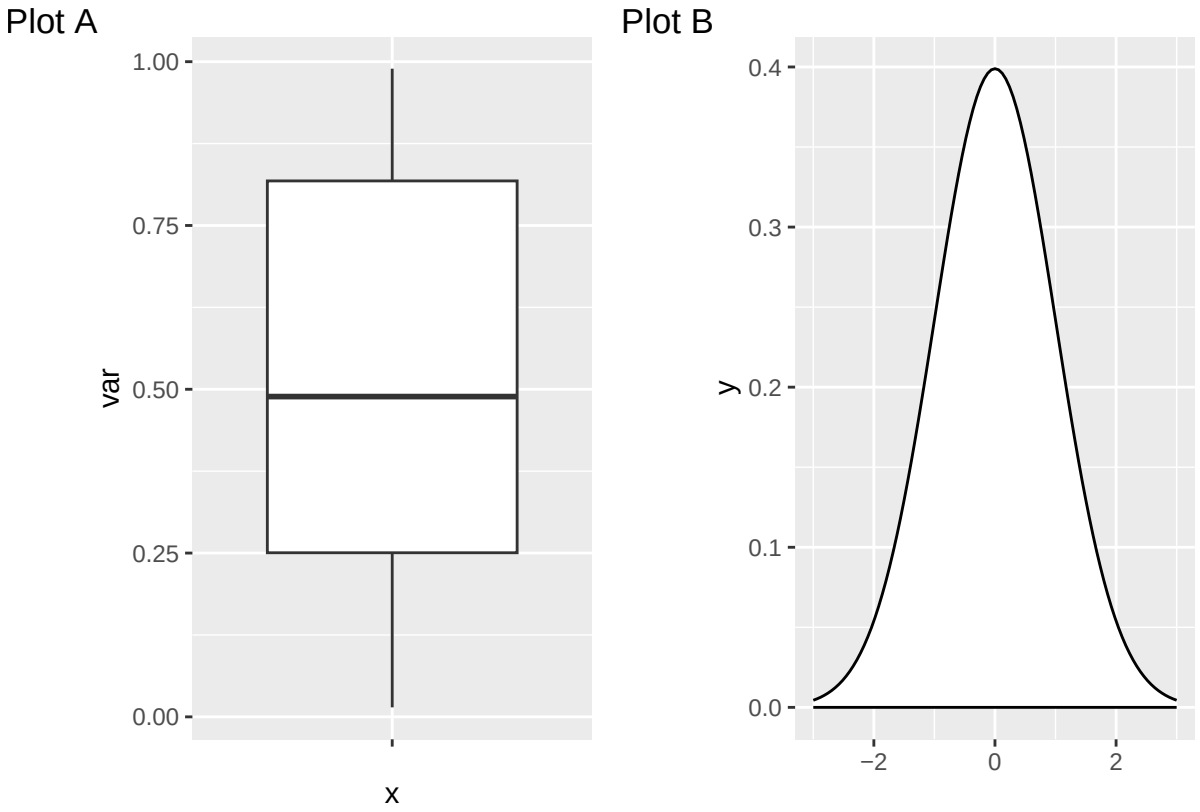
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Labels for each plot
p1 + p2 + plot_annotation(tag_levels = "A")
```



The plot_annotation function from patchwork

The labels can be customized with the tag_prefix, tag_suffix and tag_sep arguments. In the example below we add "Plot" as prefix of the labels.

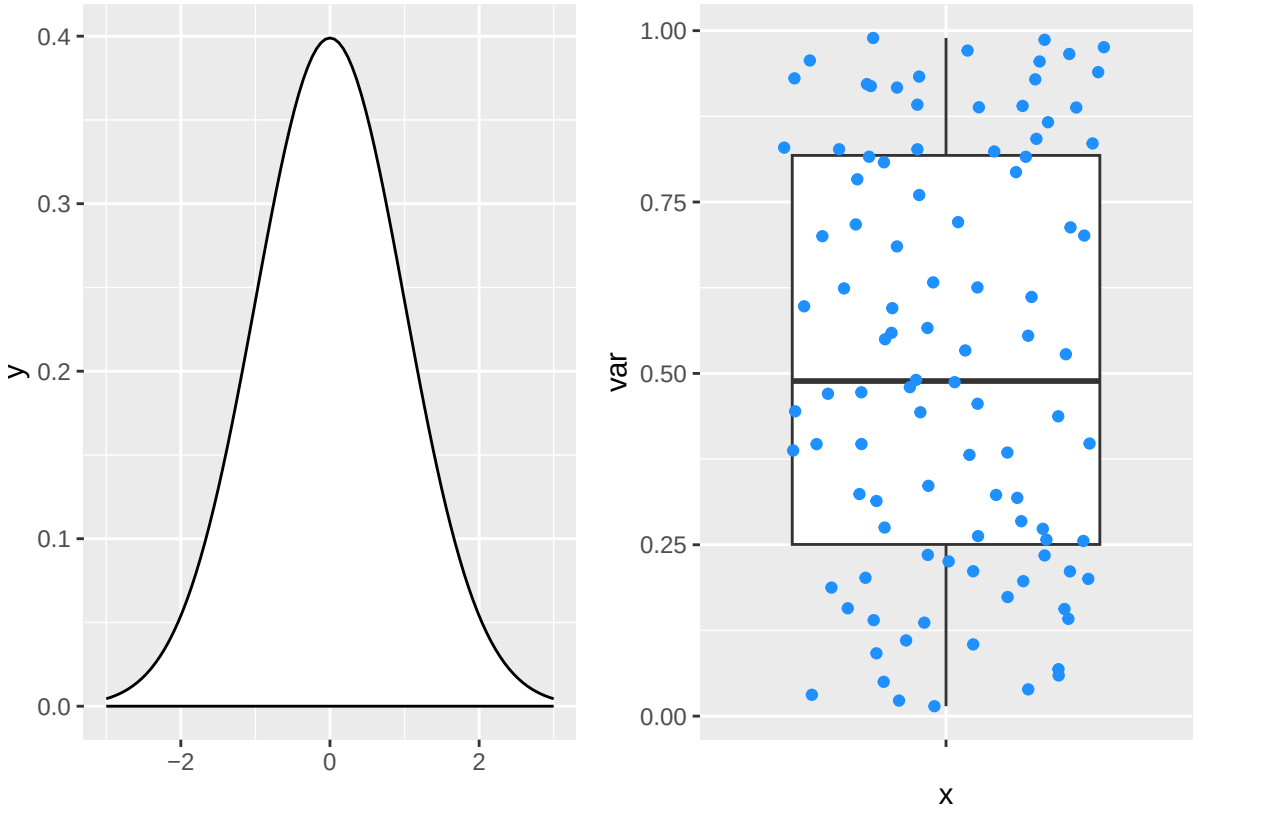
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Labels for each plot
p1 + p2 + plot_annotation(tag_levels = "A", tag_prefix = "Plot ")
```



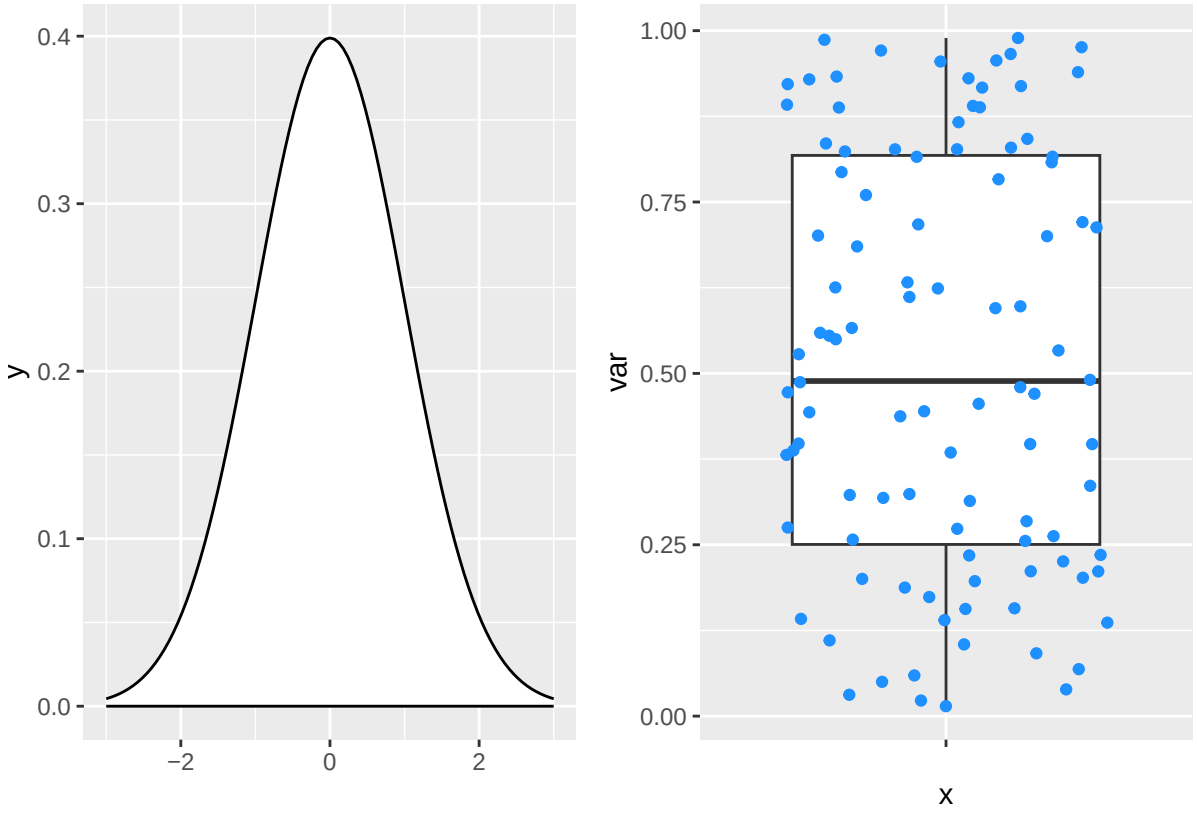
Mixing ggplots into a single figure and adding labels to each one

Adding more layers The figures created with patchwork behave the same way as a ggplot2 object, so you can add new layers as with normal plots, but the layer will be applied to the last added plot.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Add a new layer
p2 + p1 +
  geom_jitter(color = "dodgerblue1")
```



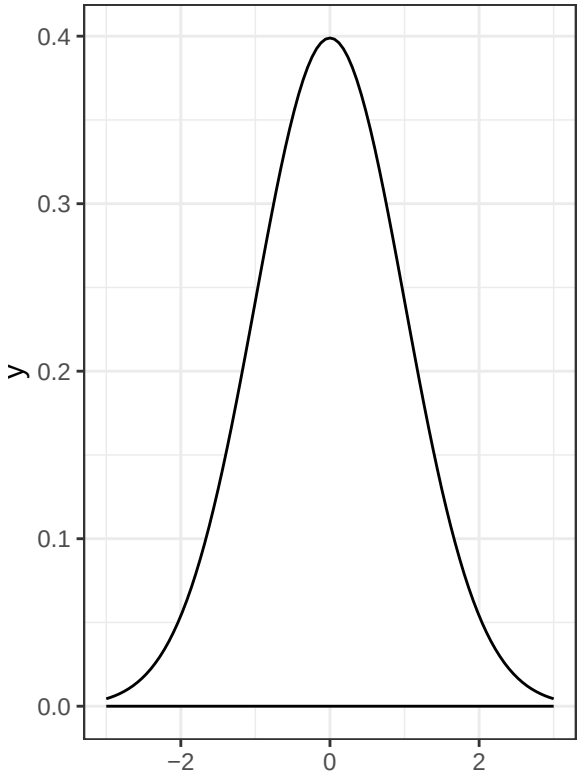

```
# Equivalent to:
patch <- p2 + p1
patch + geom_jitter(color = "dodgerblue1")
```

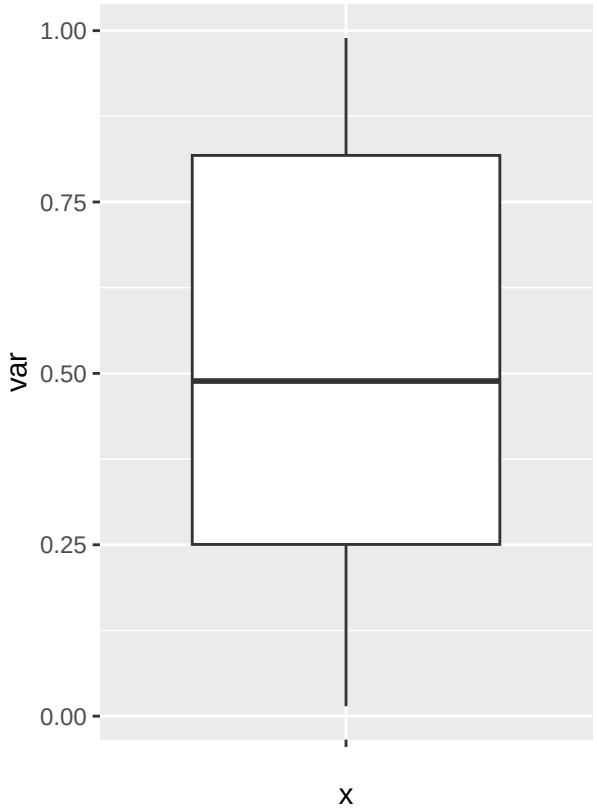


Add a new ggplot layer to a patchwork object

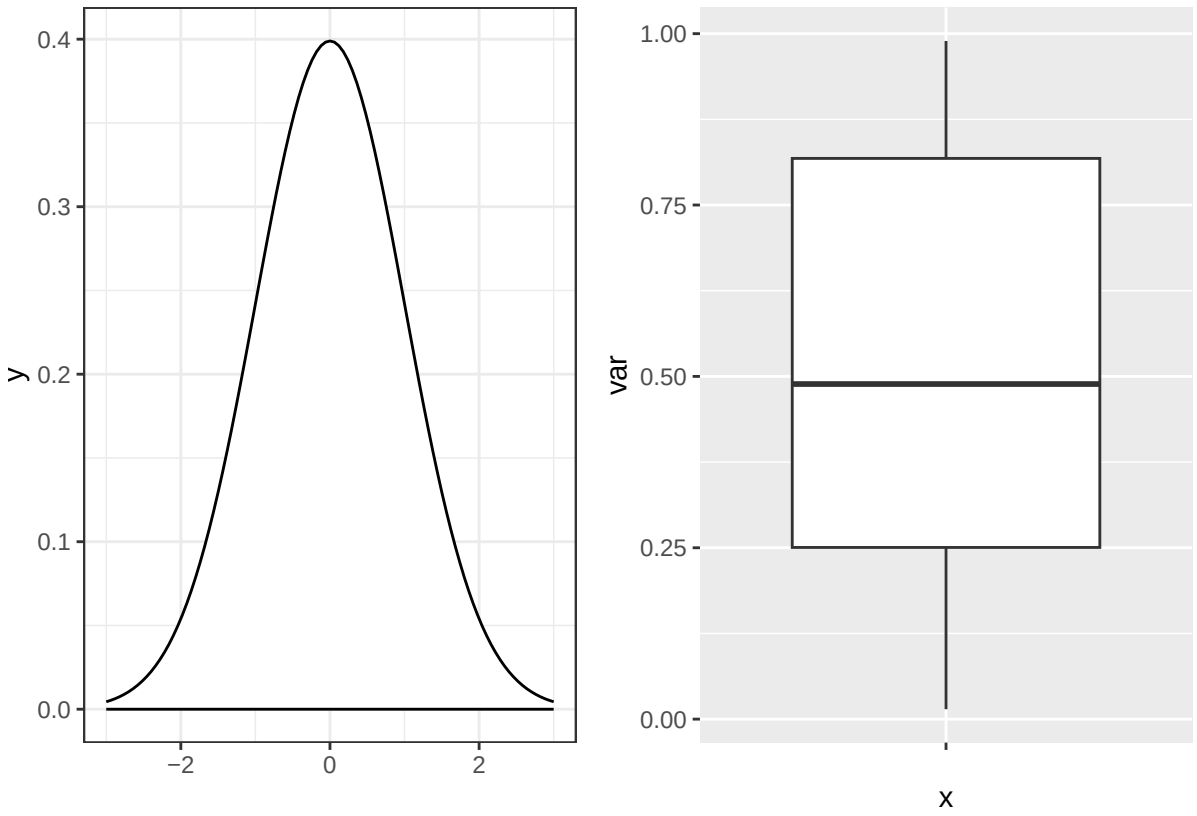
If you want to customize other than the last plot added you can add the new layer to it or save the patchwork, access the desired element and customize it, as shown in the following example.

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Add a new layer to the first plot
p2 + theme_bw() + p1
```






```
# Equivalent to:
patch <- p2 + p1
patch[[1]] <- patch[[1]] + theme_bw()
patch
```

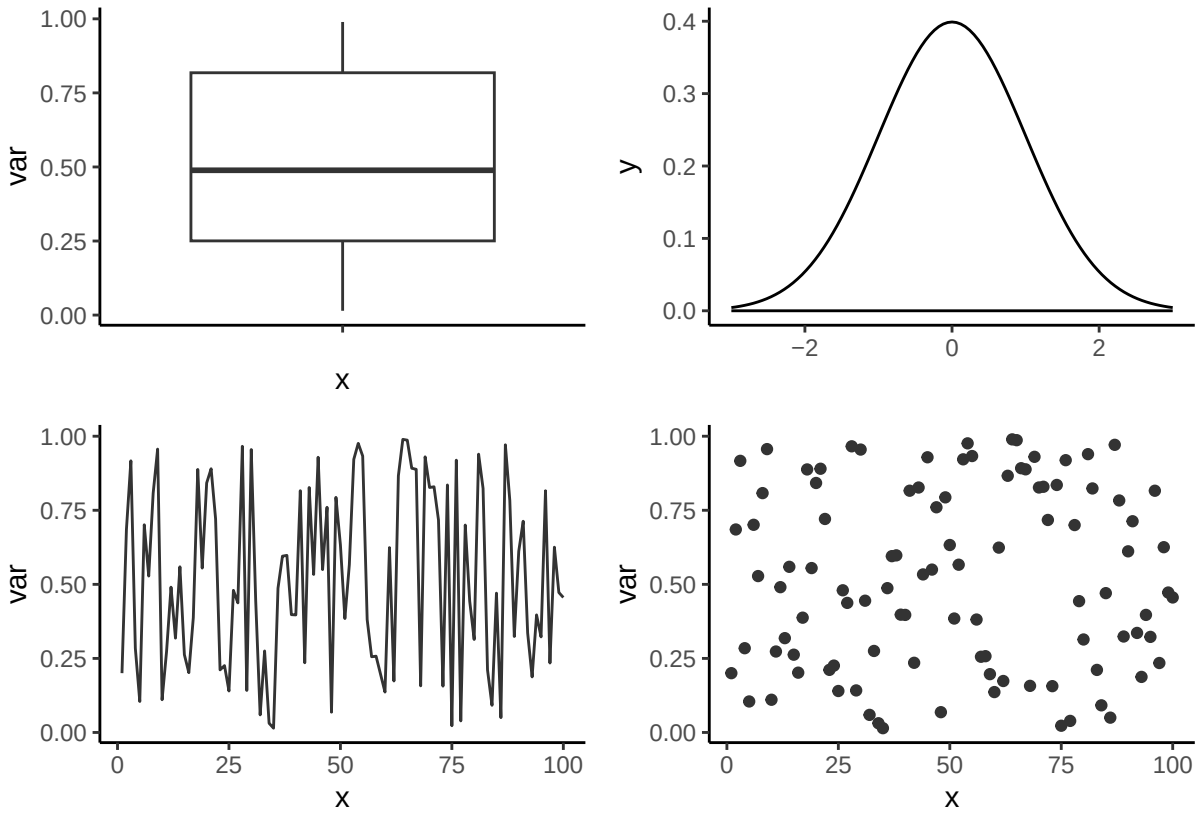


Add new layers to a plot composition made with patchwork

Modifying all plots at the same time

patchwork also provides the & operator to modify all the plots at the same time. This is very useful, for instance, to set the same theme for all plots at the same time, as shown in the example below.

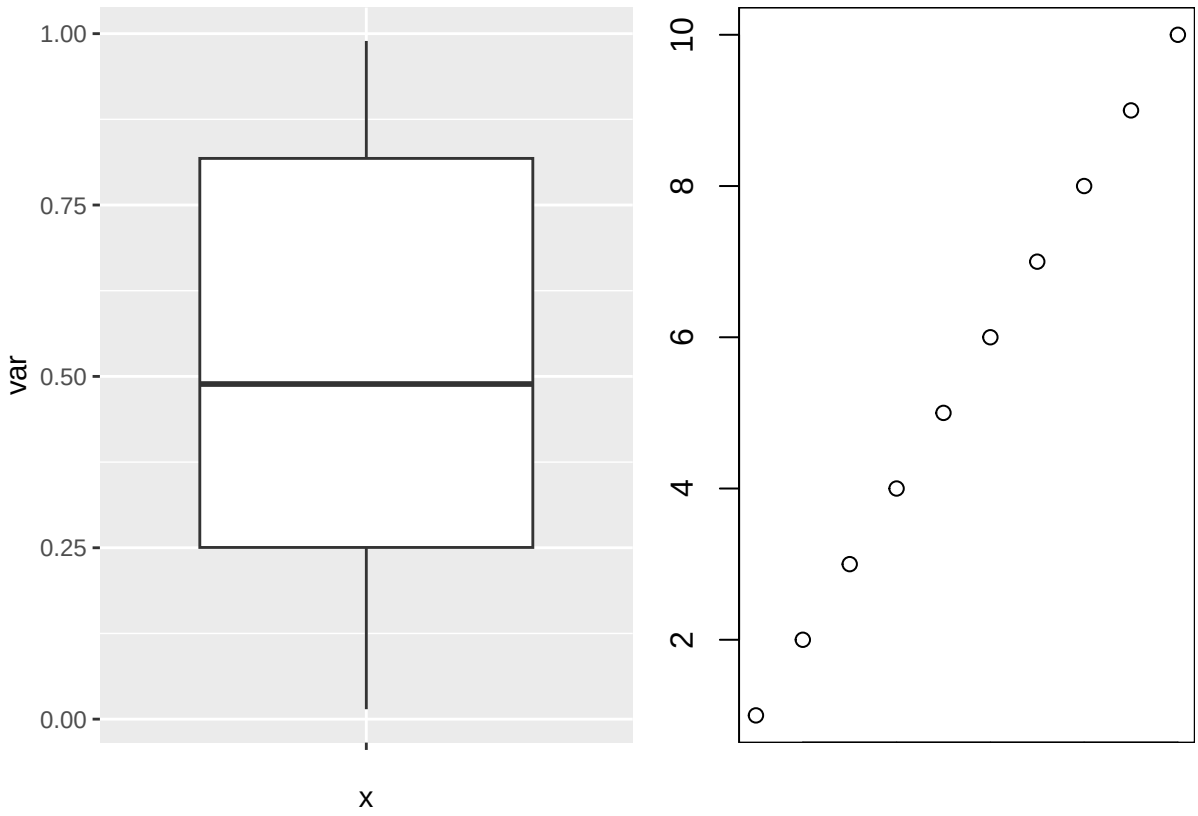
```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# Change the theme for all plots
p1 + p2 + p3 + p4 & theme_classic()
```



Apply a layer to all the ggplot2 plots of the layout at the same time

Adding base R plots, tables and text In the previous examples we have seen all the possibilities that patchwork provides to combine ggplot2 plots, but you can also create layout mixing ggplot2 plots, base R plots, tables and texts. If you want to combine a ggplot2 plot and a base R plot you can use the wrap_elements function and add the base R plot with a ~ as prefix, e.g. ~plot(x, y).

```
# install.packages("ggplot2")
# install.packages("patchwork")
library(ggplot2)
library(patchwork)

# ggplot2 and base R
old_par <- par()
par(mar = c(0, 2, 0, 0), bg = NA)
p1 + wrap_elements(panel = ~plot(1:10))
```




```
par(old_par)
```

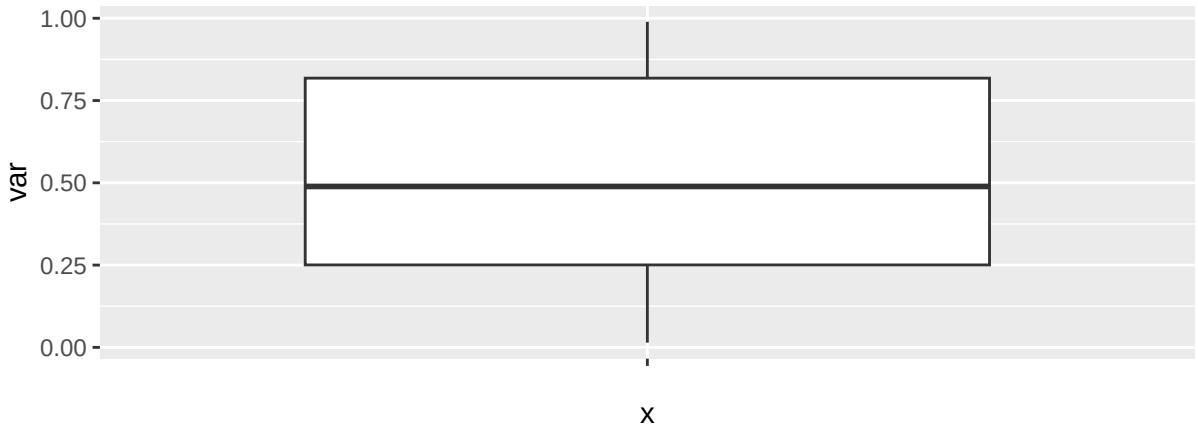
```
## Warning in par(old_par): graphical parameter "cin" cannot be set
## Warning in par(old_par): graphical parameter "cra" cannot be set
## Warning in par(old_par): graphical parameter "csi" cannot be set
## Warning in par(old_par): graphical parameter "cxy" cannot be set
## Warning in par(old_par): graphical parameter "din" cannot be set
## Warning in par(old_par): graphical parameter "page" cannot be set
```

Combining ggplot with plot in R

When adding base R plots and ggplot2 plots patchwork won't be able to align the plots, so you will need to customize the margins for one of the plots and try to fine tune the values until you reach a good alignment.

Making use of the tableGrob function from the gridExtra package that we will review in the last section of this tutorial you can add a table to a layout created with patchwork.

```
# install.packages("ggplot2")
# install.packages("patchwork")
# install.packages("gridExtra")
library(ggplot2)
library(patchwork)
library(gridExtra)

tab <- t(round(quantile(df$var), 2))

# ggplot2 with table
p1 / tableGrob(tab)
```

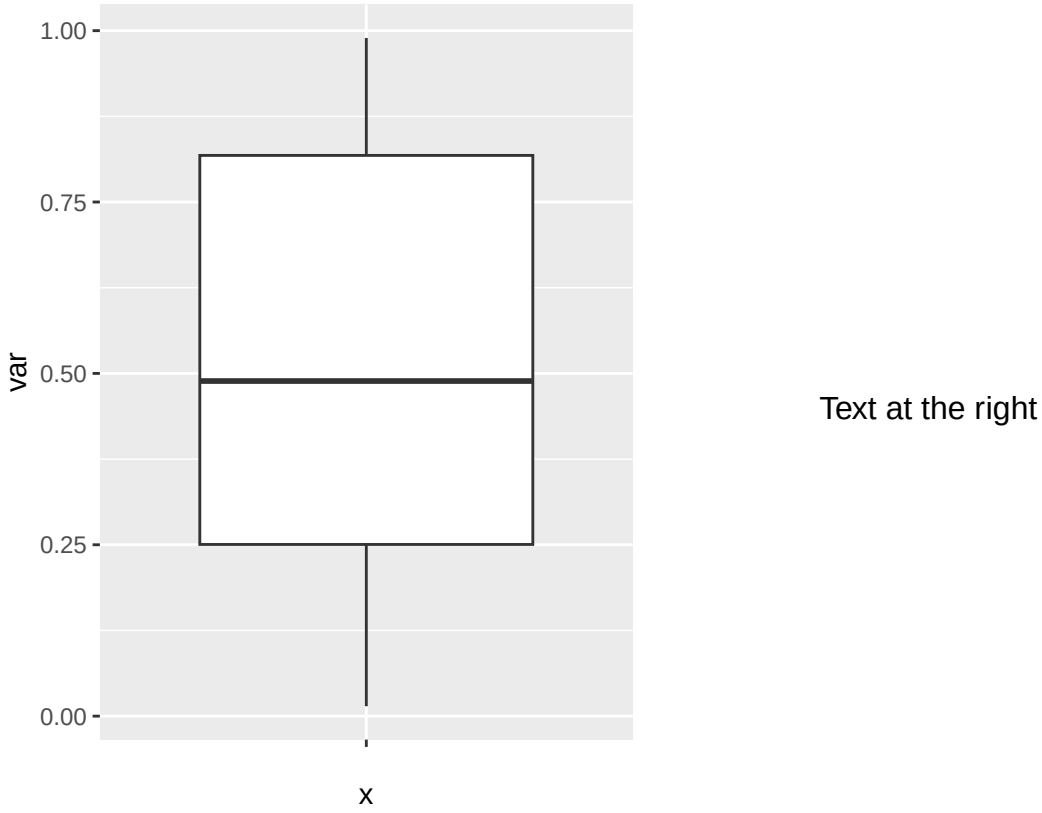


0%	25%	50%	75%	100%
0.01	0.25	0.49	0.82	0.99

ggplot2 with a table with tableGrob and patchwork

You can also use the textGrob function from gridExtra to add a text to the layout, but note that if you want the text to be the first element you will need to use the wrap_elements function.

```
# install.packages("ggplot2")
# install.packages("patchwork")
# install.packages("grid")
library(ggplot2)
library(patchwork)
library(grid)

# ggplot2 with text
p1 + textGrob('Text at the right')
```




```
# To put the text first use:
# wrap_elements(textGrob('Text at the left')) + p1
```

ggplot2 and text with the textGrob function from gridExtra

cowplot The cowplot package provides different add-ons to ggplot2, but in this tutorial we are going to review the plot_grid function, which allows combining plots passing objects to the function. Note that you can also add base R plots by using a ~ as with patchwork.

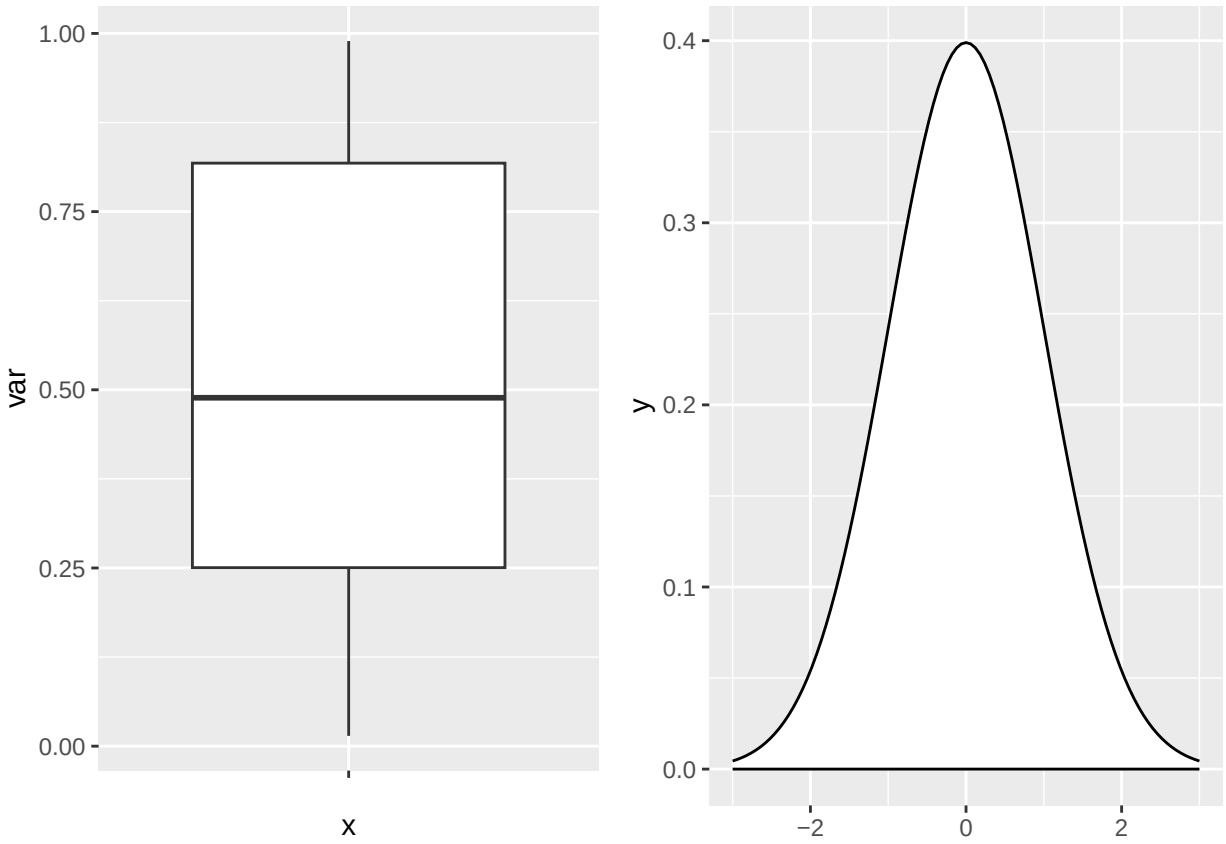
The cowplot package to mix ggplots

```
# install.packages("ggplot2")
# install.packages("cowplot")
library(ggplot2)
library(cowplot)
```

```
##
## Attaching package: 'cowplot'
```

```
## The following object is masked from 'package:patchwork':
##
##     align_plots
```

```
plot_grid(p1, p2)
```

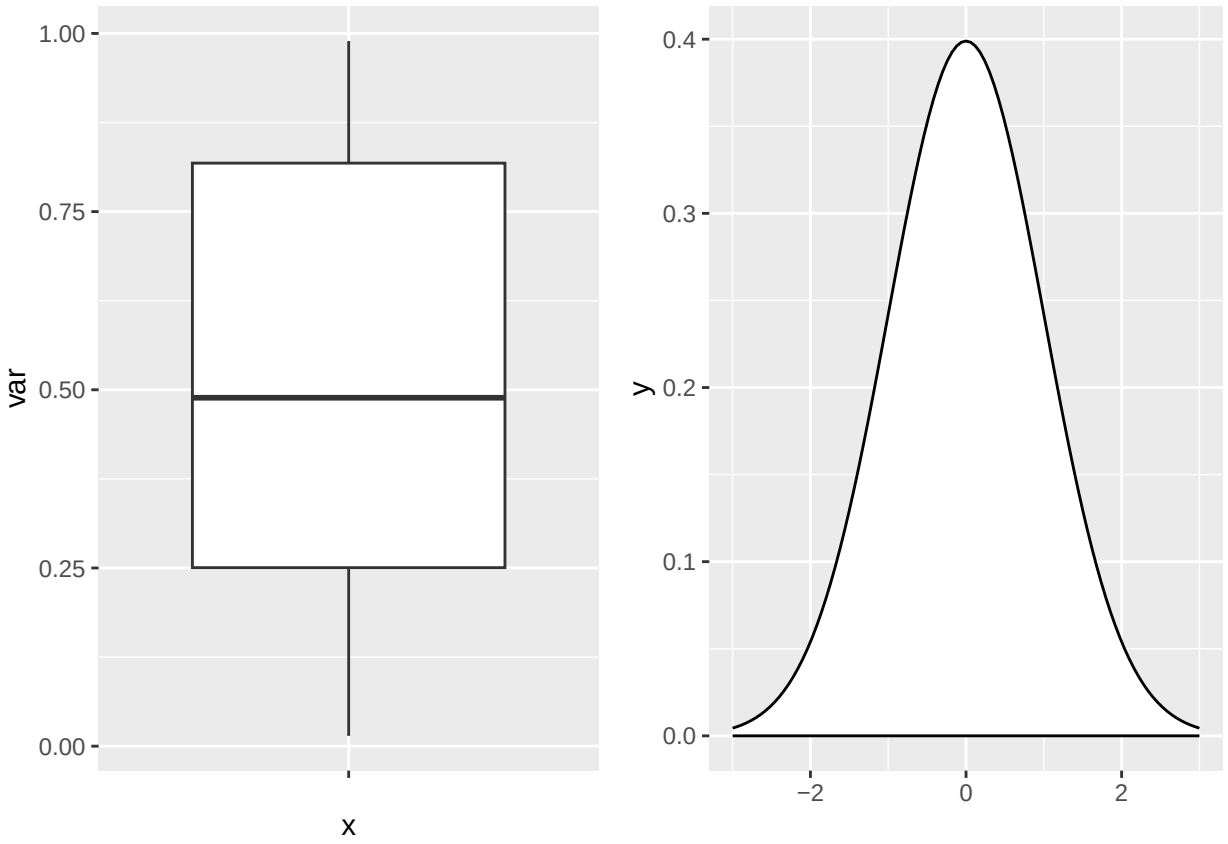


Aligning the plots

The function provides the align argument to set the desired alignment, which can be none (“none”, the default), horizontal (“h”), vertical (“v”) or aligned in both directions (“hv”).

Align plots in cowplot R package

```
# install.packages("ggplot2")
# install.packages("cowplot")
library(ggplot2)
library(cowplot)

plot_grid(p1, p2, align = "h")
```

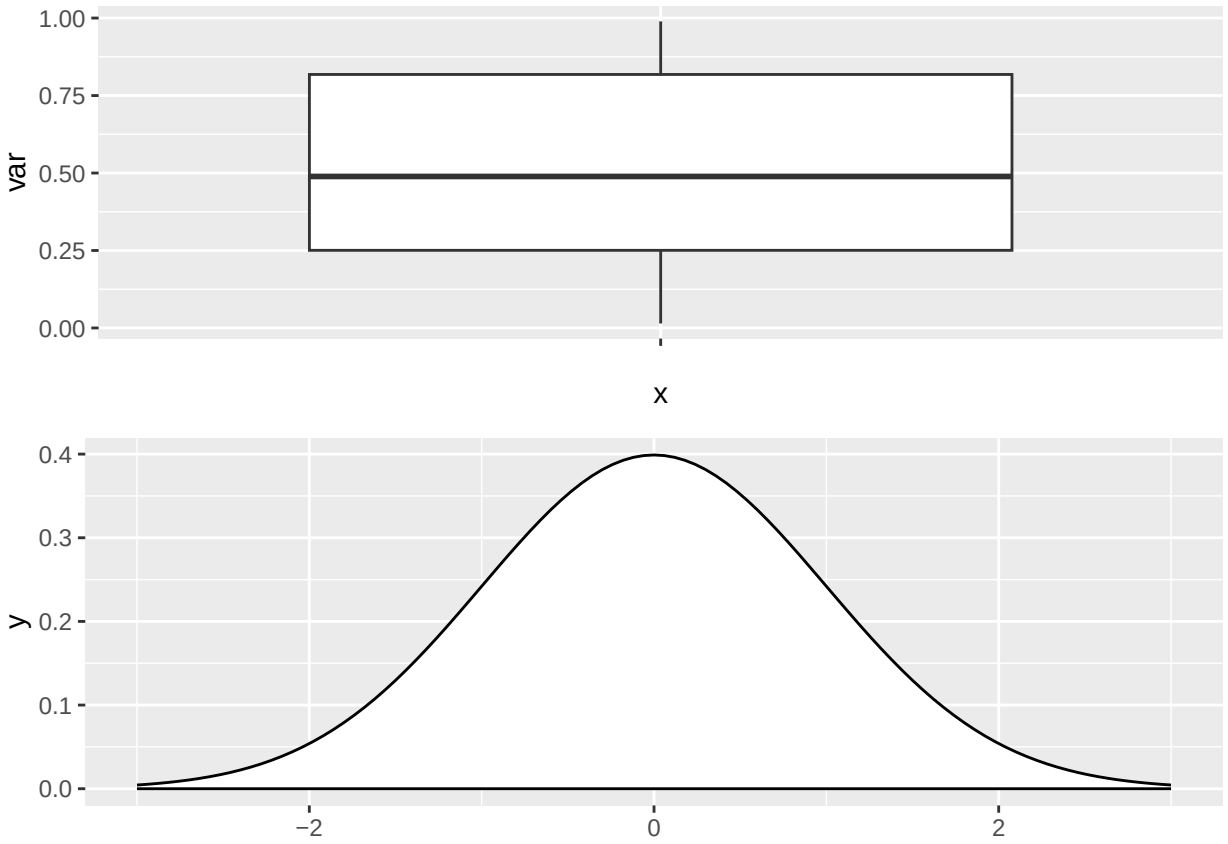


Number of rows and columns

The plot_grid function also let you choose the number of rows and columns used to create the grid of plots with the nrow and ncol arguments.

The plot_grid function from cowplot

```
# install.packages("ggplot2")
# install.packages("cowplot")
library(ggplot2)
library(cowplot)

plot_grid(p1, p2, ncol = 1)
```

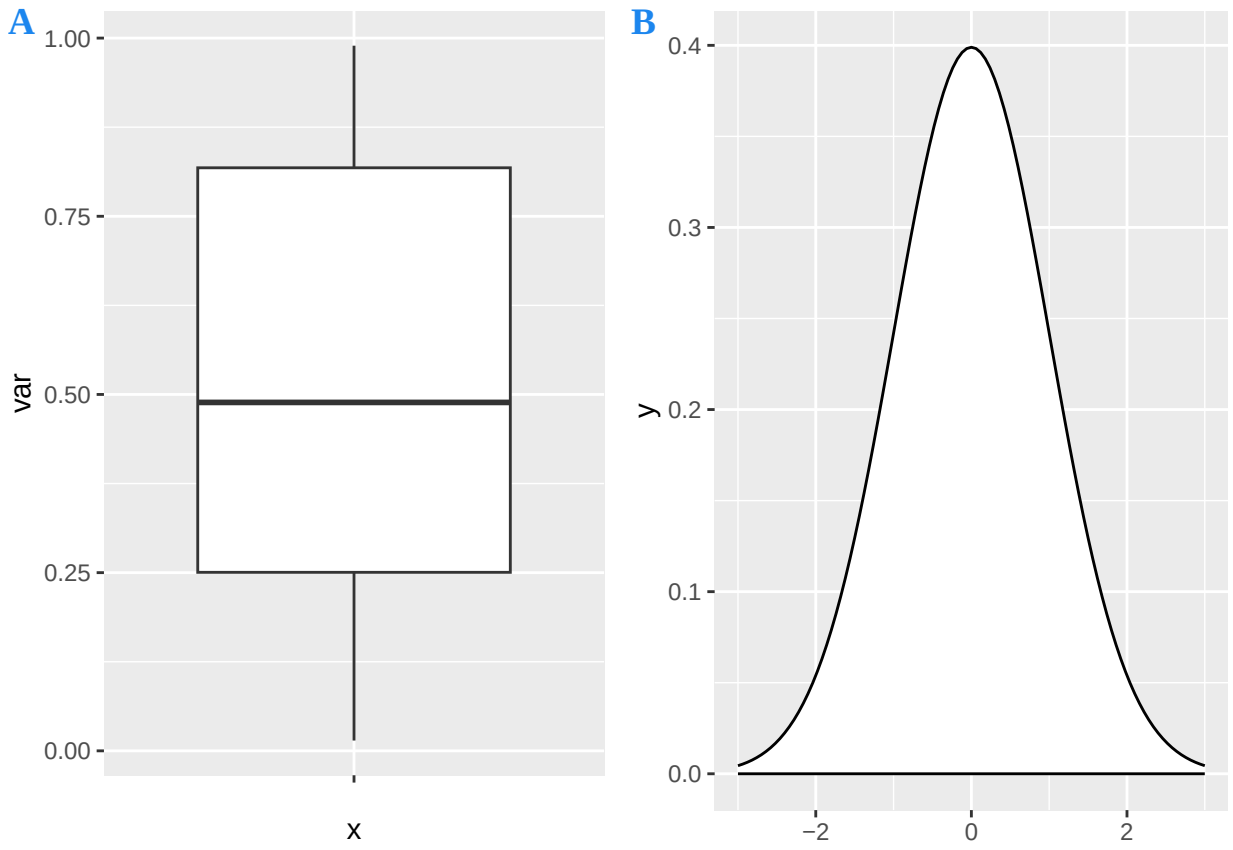


Adding labels to each plot

If you want to label each plot individually you can make use the labels argument of the function, where you can specify a vector of labels or use the "AUTO" or "auto" keywords for automatic labels in uppercase or lowercase, respectively. The function also provides several arguments to customize the style of the texts.

Add labels to the ggplot charts of the layout created with cowplot

```
# install.packages("ggplot2")
# install.packages("cowplot")
library(ggplot2)
library(cowplot)

plot_grid(p1, p2,
          labels = c('A', 'B'), # Or "AUTO" or "auto"
          label_fontfamily = "serif",
          label_fontface = "bold",
          label_colour = "dodgerblue2")
```

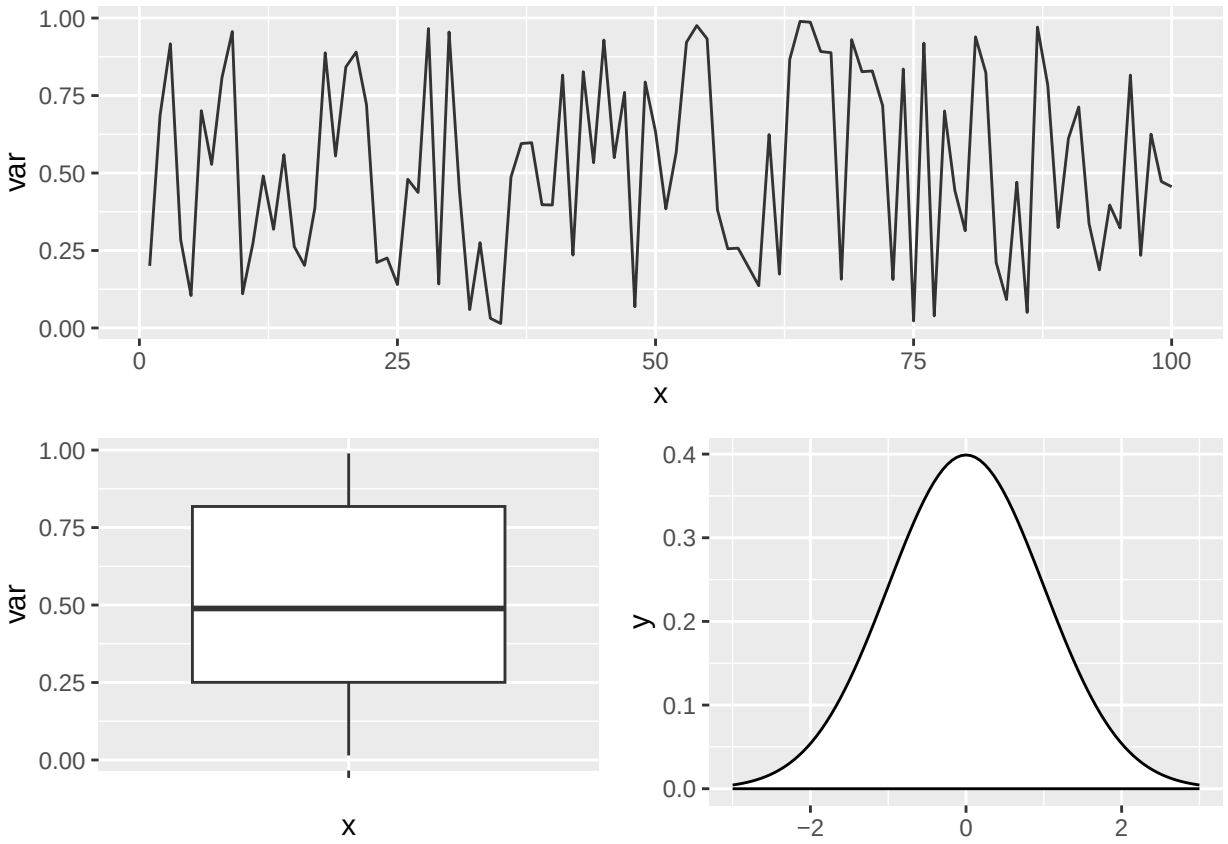


Mixing plots with cowplot

With cowplot you can also create more complex layouts combining plot_grid functions, as shown in the example below, where we are creating a layout with two plots at the bottom and one at the top.

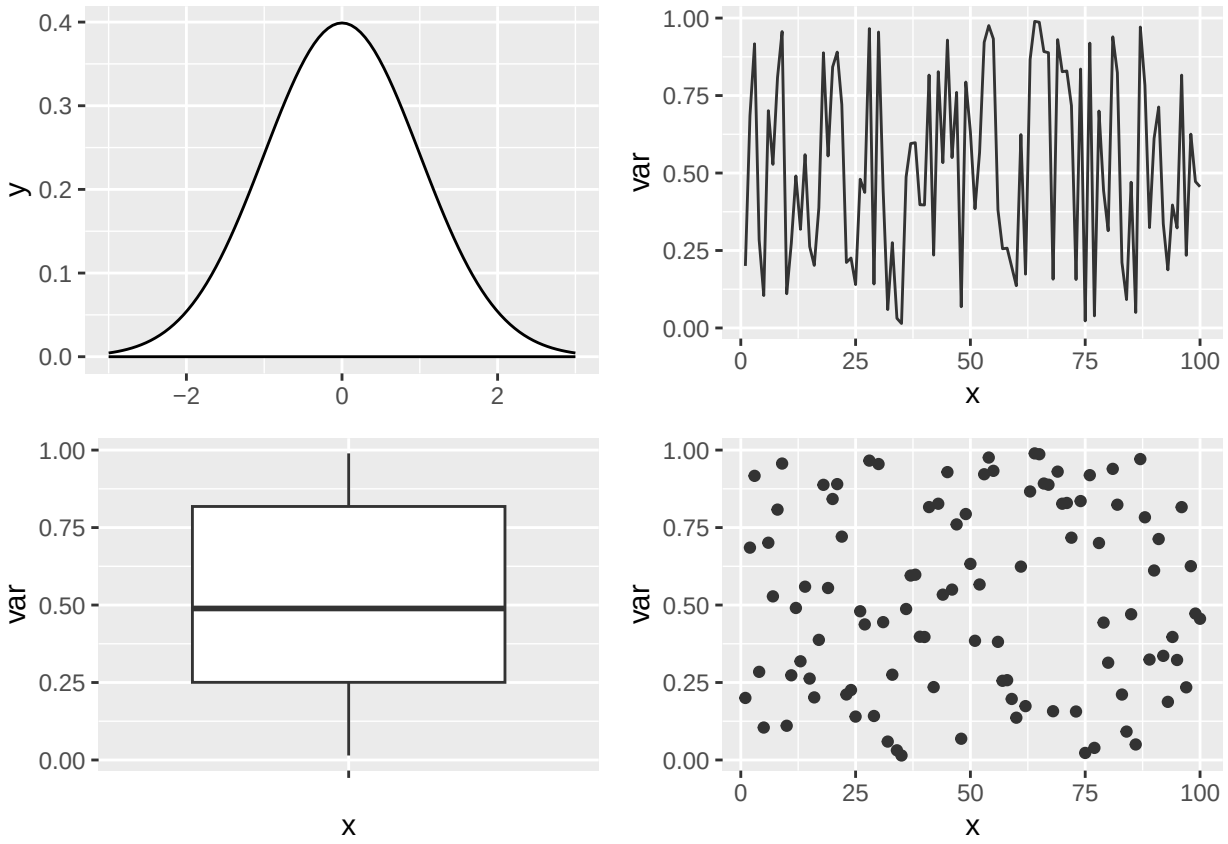
Combine several plots with the cowplot package

```
# install.packages("ggplot2")
# install.packages("cowplot")
library(ggplot2)
library(cowplot)

# Grid layout with cowplot
plot_grid(p3, plot_grid(p1, p2), ncol = 1)
```



gridExtra The gridExtra package provides the grid.arrange function to combine several plots (grobs, gtables, ggplot2 and trellis objects) on a single figure.

```
# install.packages("ggplot2")
# install.packages("gridExtra")
library(ggplot2)
library(gridExtra)

# Combine the plots with gridExtra
grid.arrange(p2, p3, p1, p4)
```

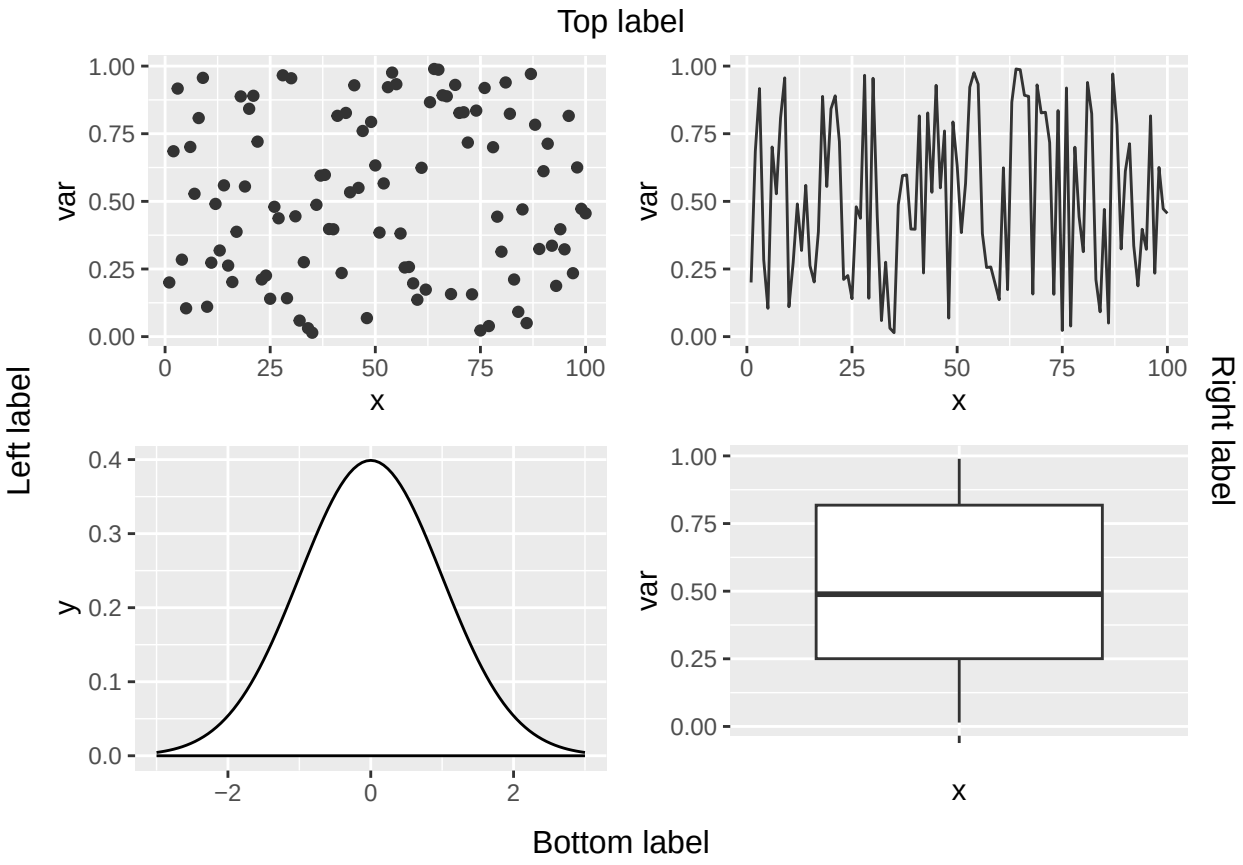


grid.arrange to mix several ggplots

You can also specify the number of rows with nrow, the number of columns with ncol, and the proportional sizes with widths and heights.

The function also provides arguments to add labels at the four sides of the figure named top, bottom, left and right.

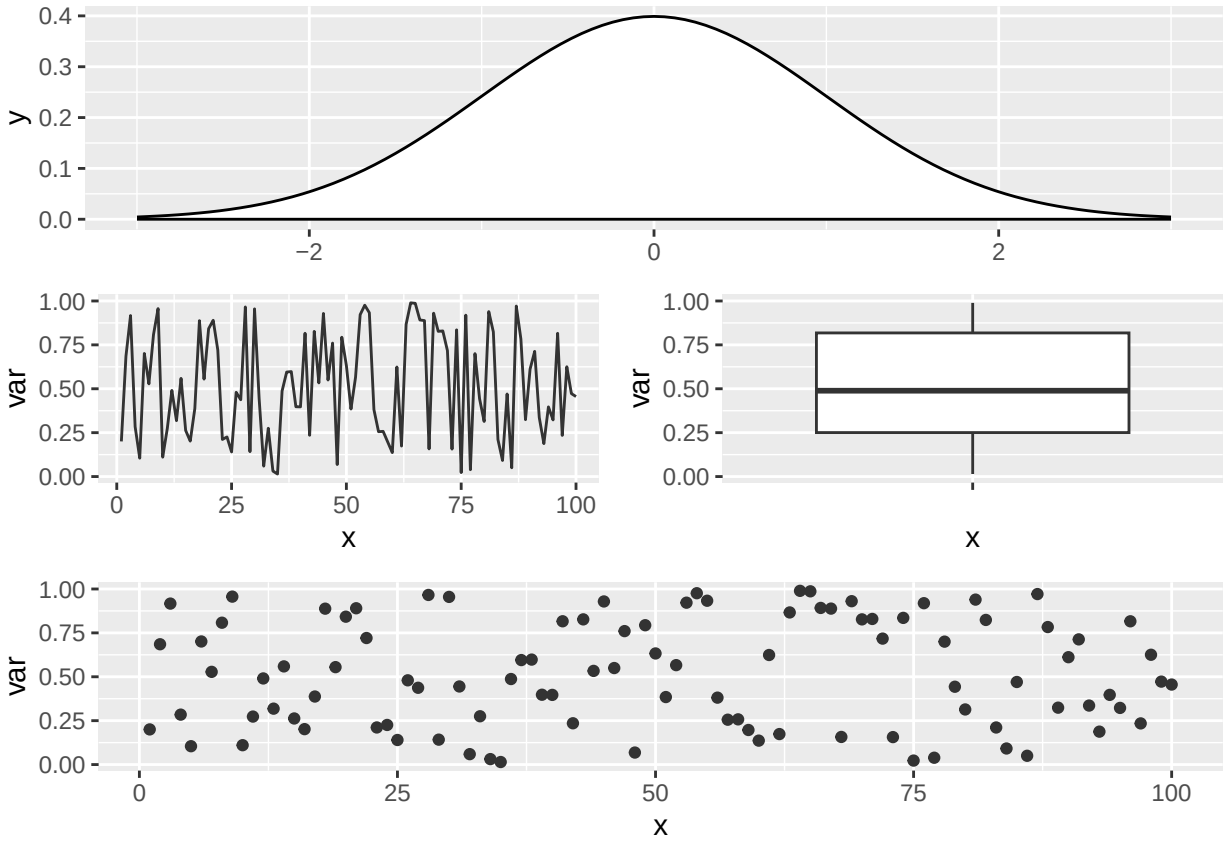
```
# install.packages("ggplot2")
# install.packages("gridExtra")
library(ggplot2)
library(gridExtra)

# Combine the plots
grid.arrange(p4, p3, p2, p1,
             top = "Top label", bottom = "Bottom label",
             left = "Left label", right = "Right label")
```



Add labels at the sides of a figure layout with the grid.arrange function from gridExtra

Similarly to the layouts created with patchwork you can create a matrix indicating the positions for each plot. Then you can pass this matrix to the layout_matrix argument in order to specify the desired layout.

```
# install.packages("ggplot2")
# install.packages("gridExtra")
library(ggplot2)
library(gridExtra)

# Custom layout
layout <- matrix(c(1, 1,
                   2, 3,
                   4, 4), ncol = 2, byrow = TRUE)

grid.arrange(p2, p3, p1, p4,
             layout_matrix = layout)
```



Custom layout of plots with the gridExtra package