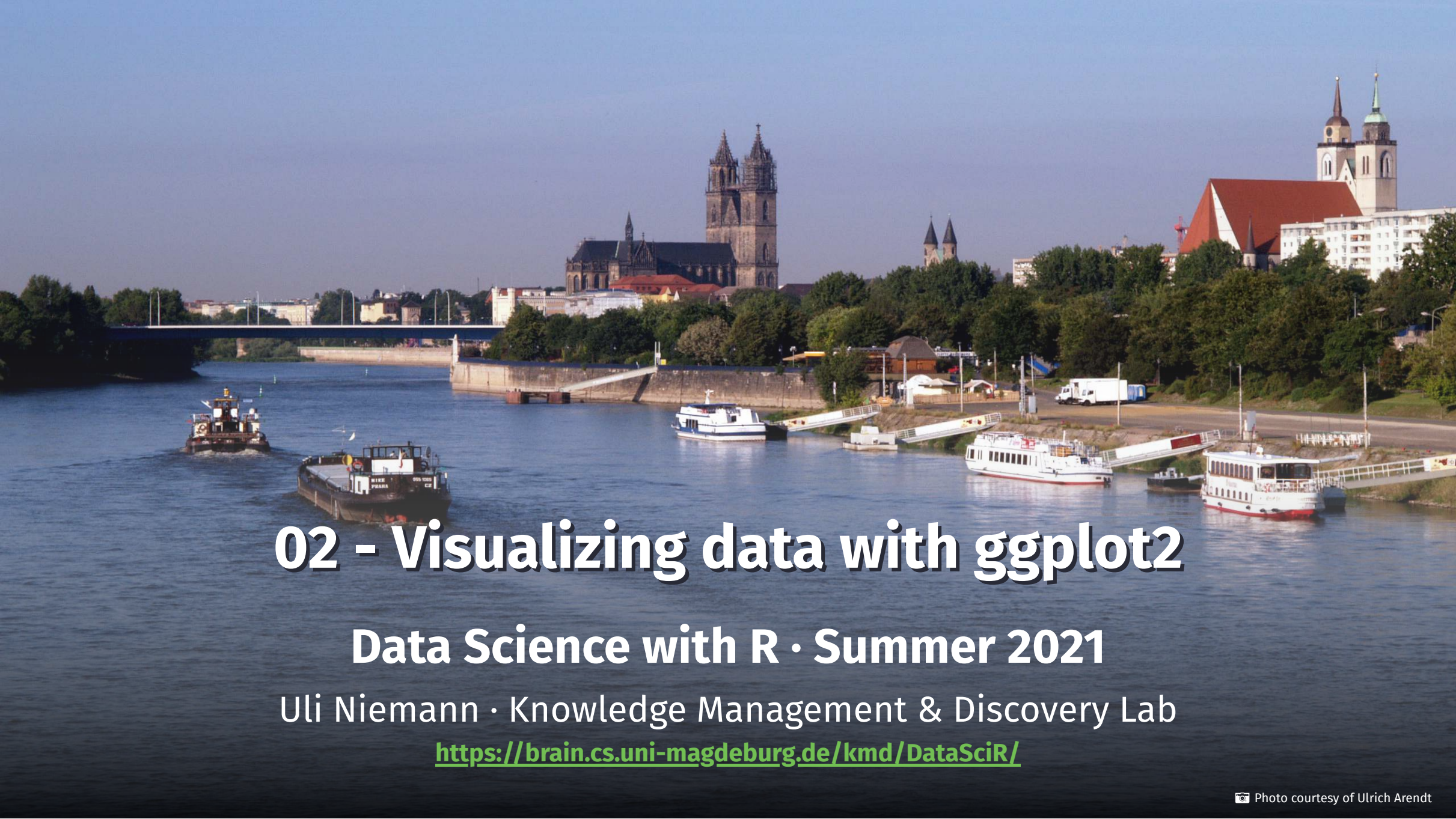


02 - Visualizing data with ggplot2

Data Science with R · Summer 2021

Uli Niemann · Knowledge Management & Discovery Lab

https://brain.cs.uni-magdeburg.de/kmd/DataSciR/

Photo courtesy of Ulrich Arendt

Datasets

In `R` most datasets come in the form of data frames:

- Each row is an **observation**.
- Each column is a **variable**.

```
library(gapminder)
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ... with 1,694 more rows
```

Example: Germany in 2007

- `country = "Germany"`
- `continent = "Europe"`
- `year = 2007`
- `lifeExp = 79.4` years
- `pop = 82400996` inhabitants
- `gdpPercap = 32170` USD

```
## # A tibble: 1 x 6
##   country continent  year lifeExp      pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Germany Europe     2007    79.4 82400996    32170.
```

What's in the Gapminder data?

- How many rows and columns does this dataset contain?
- What does each row represent?
- What does each column represent?

Take a `glimpse()` at the data:

```
library(dplyr)
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", "Afghanist~
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia~
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, 2002, 2007~
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.822, 41.674~
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12881816, 13~
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, 978.0114, ~
```

Consulting the dataset documentation

```
?gapminder
# alternative: place cursor within `gapminder` and press F1
```

gapminder {gapminder}

Gapminder data.

Description

Excerpt of the Gapminder data on life expectancy, GDP per capita, and population by country.

Usage

```
gapminder
```

Format

The main data frame `gapminder` has 1704 rows and 6 variables:

country

: factor with 142 levels

continent

: factor with 5 levels

year

: ranges from 1952 to 2007 in increments of 5 years

```
nrow(gapminder) # number of rows
```

```
## [1] 1704
```

```
ncol(gapminder) # number of columns
```

```
## [1] 6
```

```
dim(gapminder) # dimensions (row column)
```

```
## [1] 1704    6
```

Why visualize data?

Visualization is part of...

- **Exploratory data analysis**: understand distributions, identify outliers & missing data
- **Feature engineering**: discover relationships between two or more predictors and extract a new predictor to increase model performance
- **Model presentation**: show clusters, dimension reductions, etc.
- **Model evaluation**: graphically describe the performance of one or more inferential or predictive models
- **Storytelling**: convincingly communicate a data-driven finding

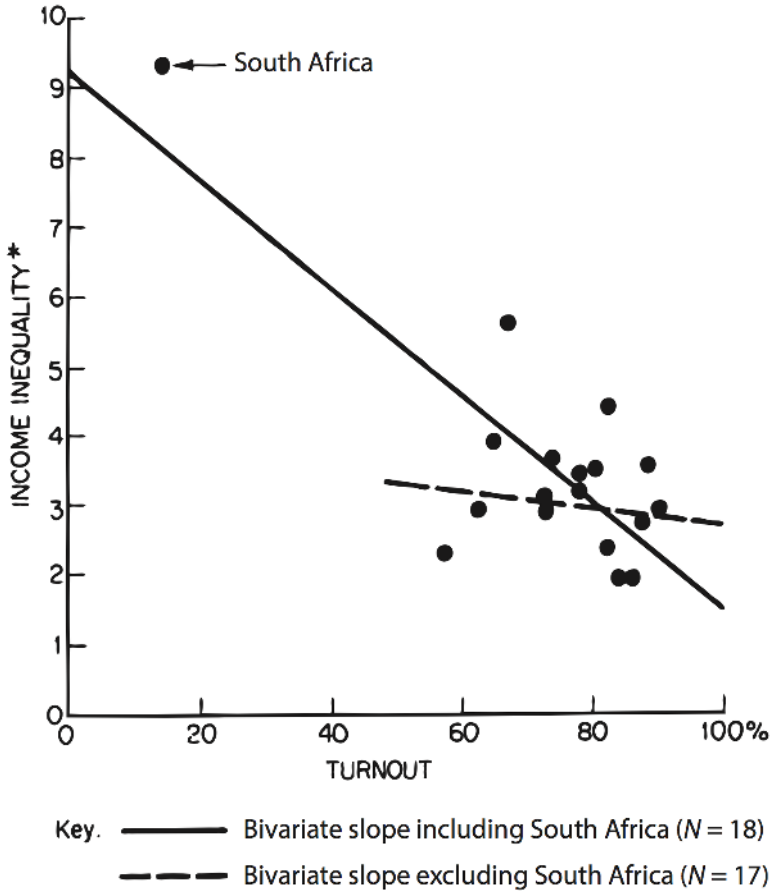


Figure source: Kieran Healy. "Data Visualization. A practical introduction". Princeton University Press, 2018.

Anscombe's quartett

```
##    group  x     y
## 1      1 10  8.04
## 2      1  8  6.95
## 3      1 13  7.58
## 4      1  9  8.81
## 5      1 11  8.33
## 6      1 14  9.96
## 7      1  6  7.24
## 8      1  4  4.26
## 9      1 12 10.84
## 10     1  7  4.82
## 11     1  5  5.68
## 12     2 10  9.14
## 13     2  8  8.14
## 14     2 13  8.74
## 15     2  9  8.77
## 16     2 11  9.26
## 17     2 14  8.10
## 18     2  6  6.13
## 19     2  4  3.10
## 20     2 12  9.13
## 21     2  7  7.26
## 22     2  5  4.74
```

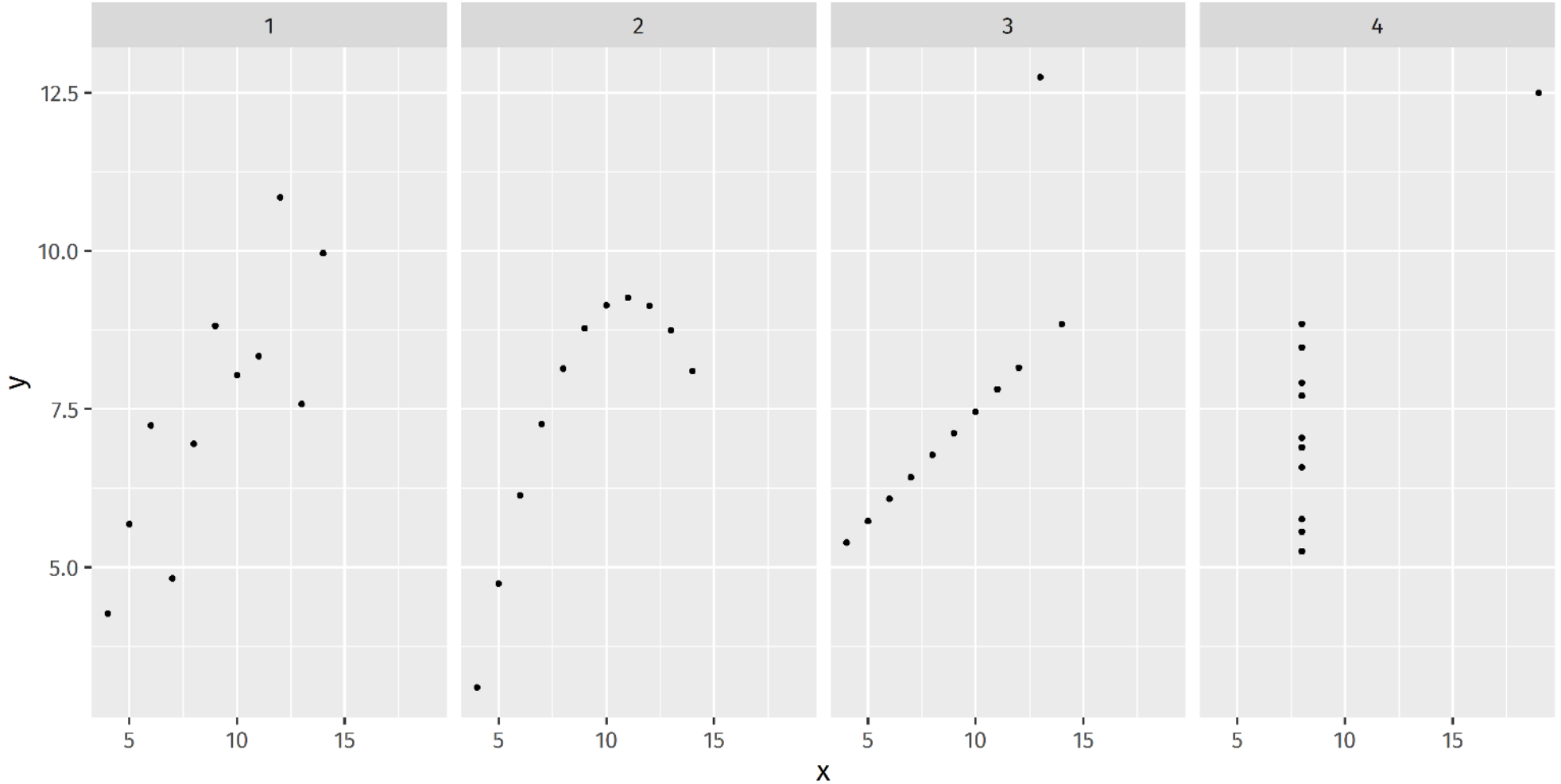
```
##    group  x     y
## 23     3 10  7.46
## 24     3  8  6.77
## 25     3 13 12.74
## 26     3  9  7.11
## 27     3 11  7.81
## 28     3 14  8.84
## 29     3  6  6.08
## 30     3  4  5.39
## 31     3 12  8.15
## 32     3  7  6.42
## 33     3  5  5.73
## 34     4  8  6.58
## 35     4  8  5.76
## 36     4  8  7.71
## 37     4  8  8.84
## 38     4  8  8.47
## 39     4  8  7.04
## 40     4  8  5.25
## 41     4 19 12.50
## 42     4  8  5.56
## 43     4  8  7.91
## 44     4  8  6.89
```

Summarizing Anscombe's quartet

```
ans %>%
  group_by(group) %>%
  summarize(
    n = n(),
    mean_x = mean(x),
    mean_y = mean(y),
    sd_x = sd(x),
    sd_y = sd(y),
    r = cor(x, y)
  )
```

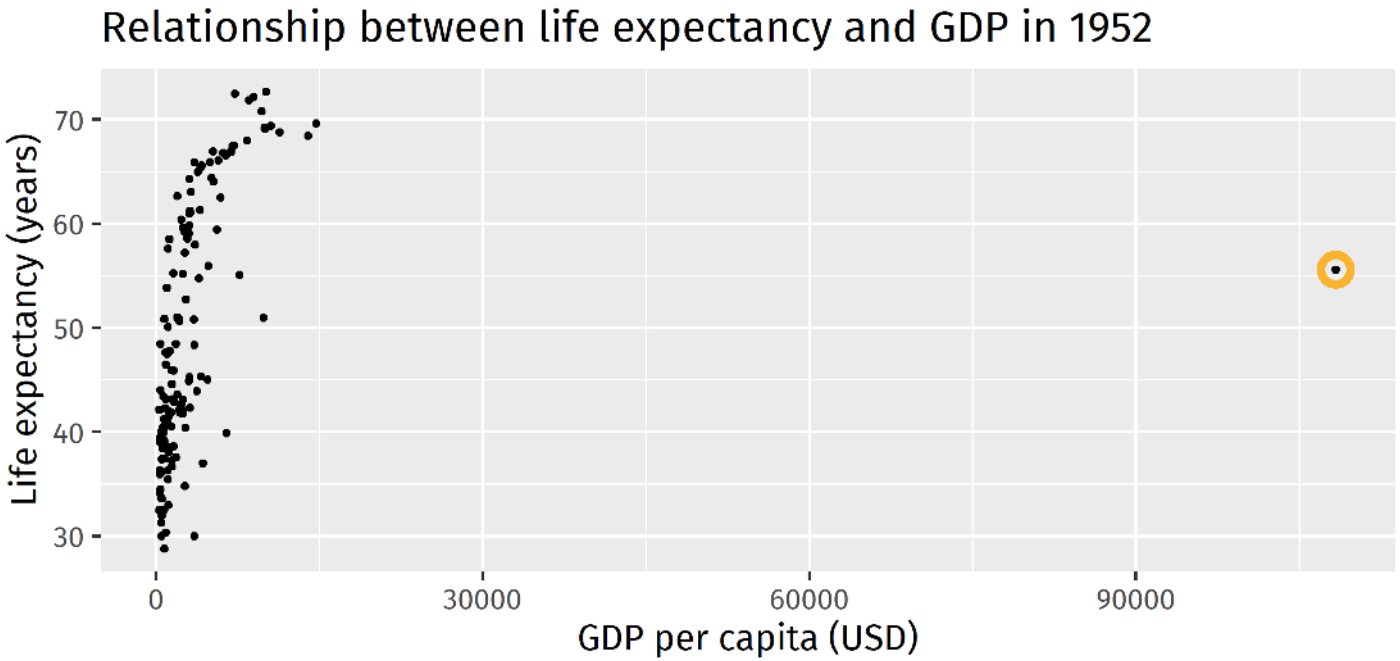
```
## # A tibble: 4 x 7
##   group     n mean_x mean_y  sd_x  sd_y     r
##   <chr> <int>  <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 1        11      9   7.50  3.32  2.03 0.816
## 2 2        11      9   7.50  3.32  2.03 0.816
## 3 3        11      9   7.5   3.32  2.03 0.816
## 4 4        11      9   7.50  3.32  2.03 0.817
```

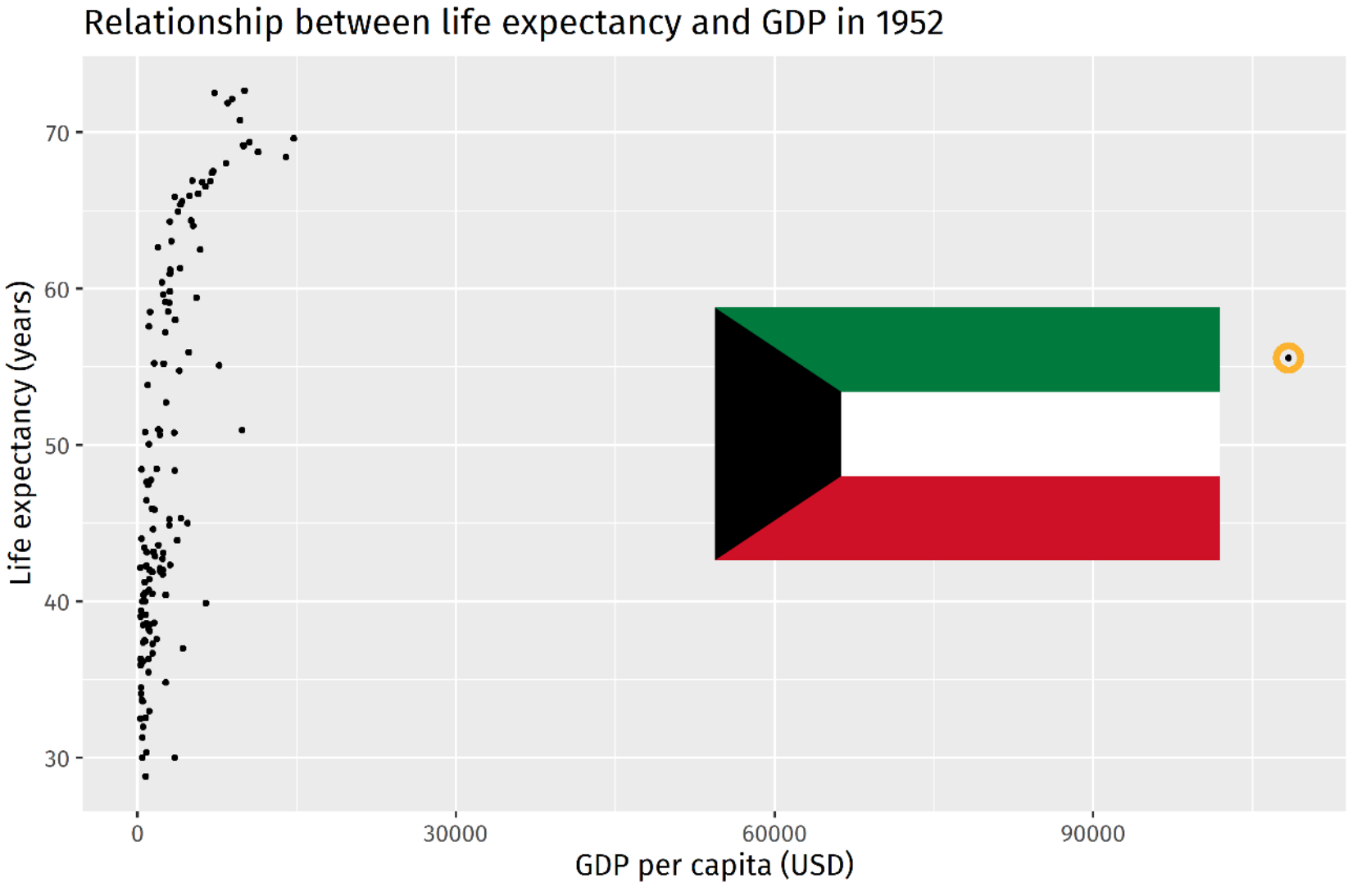
Visualizing Anscombe's quartet



Life expectancy vs. GDP

- How would you describe the relationship between life expectancy and GDP per capita in 1952?
- What other variables could have an influence on the shown trend?
- Which is the country with moderate life expectancy but extremely high GDP?

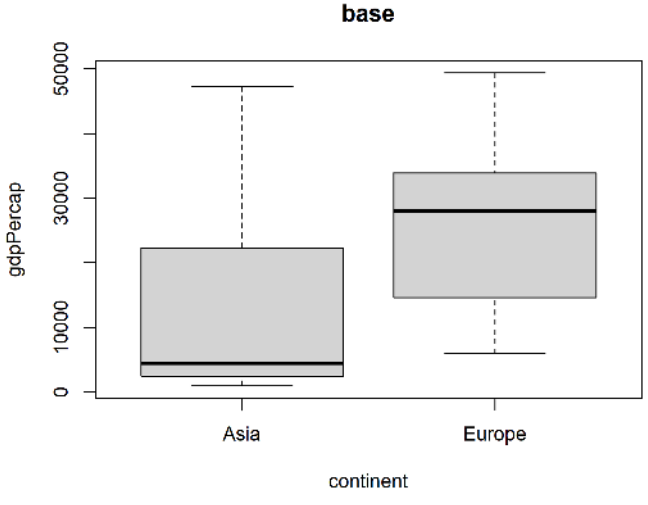


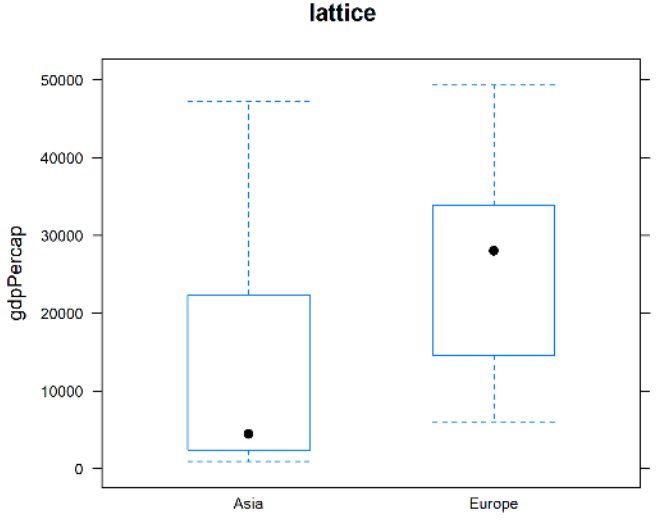

Relationship between life expectancy and GDP in 1952
Life expectancy (years)
70
60
50
40
30
0
30000
60000
90000
GDP per capita (USD)

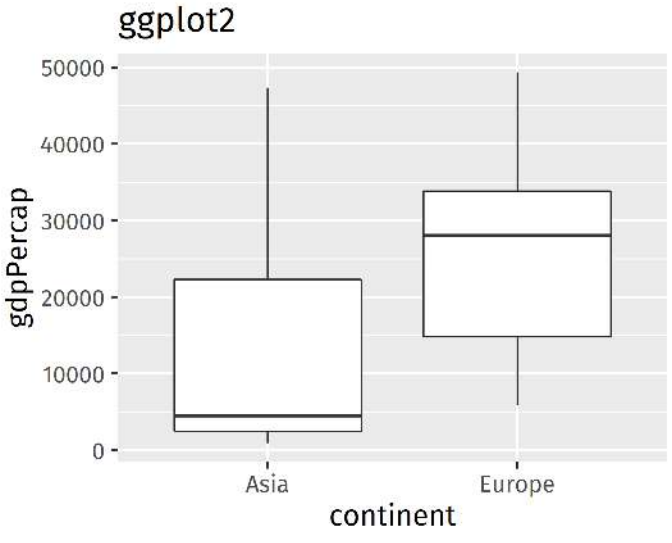
Data visualization

Data visualization = graphical representation of data

- There are many tools for visualizing data – D3, Microsoft Excel, Python, `R`, ...
- There are many systems within `R` for creating data visualizations – base, lattice, ggplot2



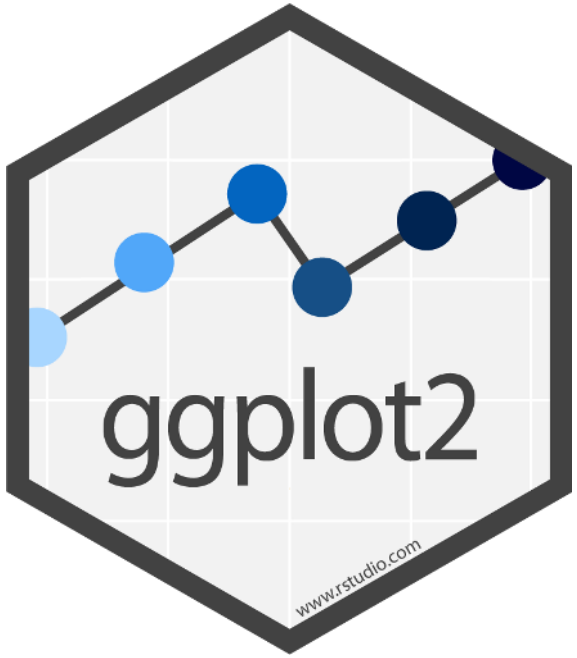




`ggplot2` 📈

`ggplot2` is a package for **data visualization** and part of the tidyverse.

- `ggplot2` is inspired by the **Grammar of Graphics**[1]
- idea: **break the graph into components** and **handle each component individually** → ensure versatility and control
- a `ggplot2` chart is built by stacking a **series of layers**
- advantage: build a **variety of different charts** with the same vocabulary → code that is easier to read and write



[1] Leland Wilkinson. *The grammar of graphics*. Springer Science & Business Media, 2006.

Components of a graphic

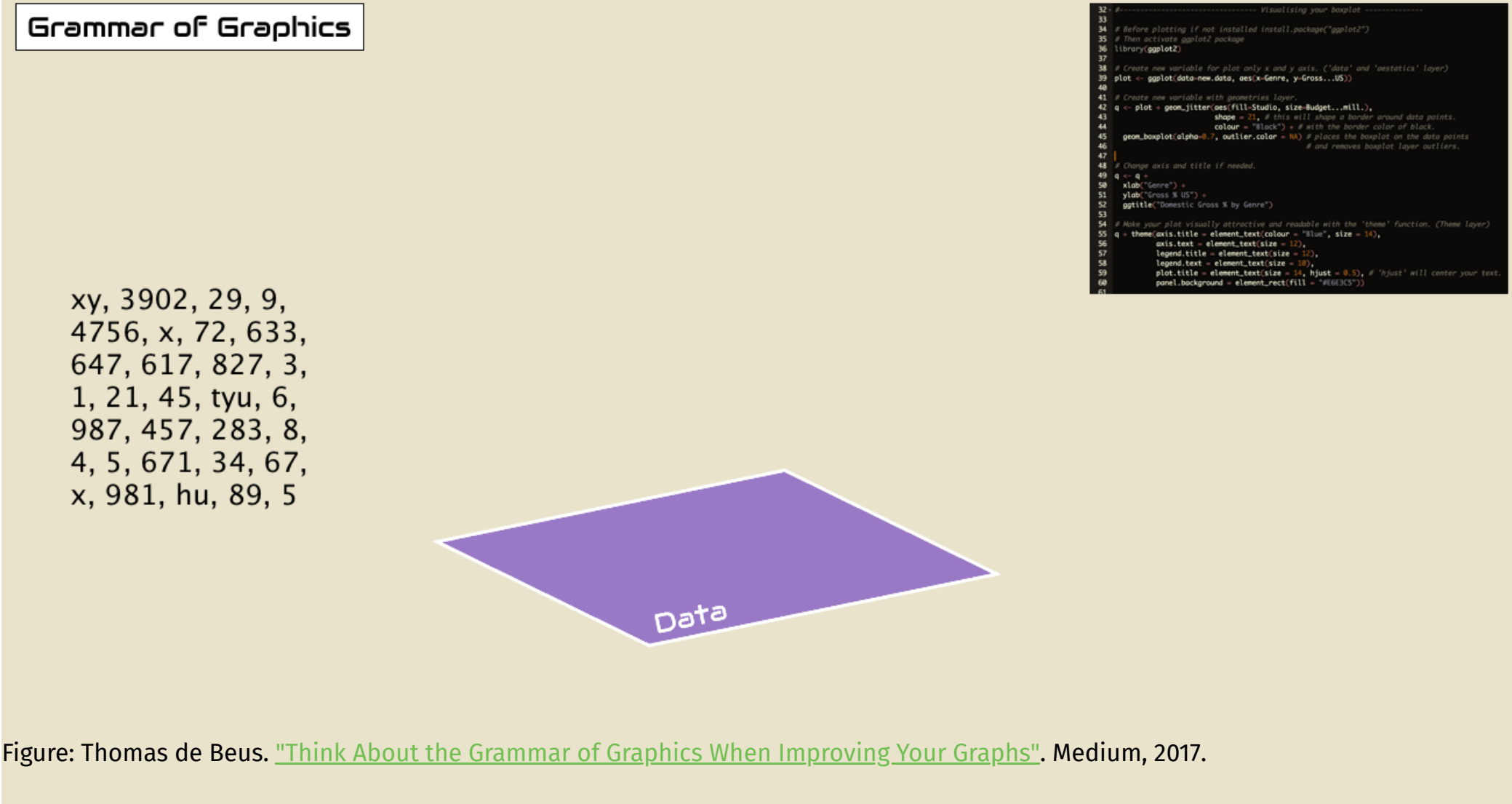


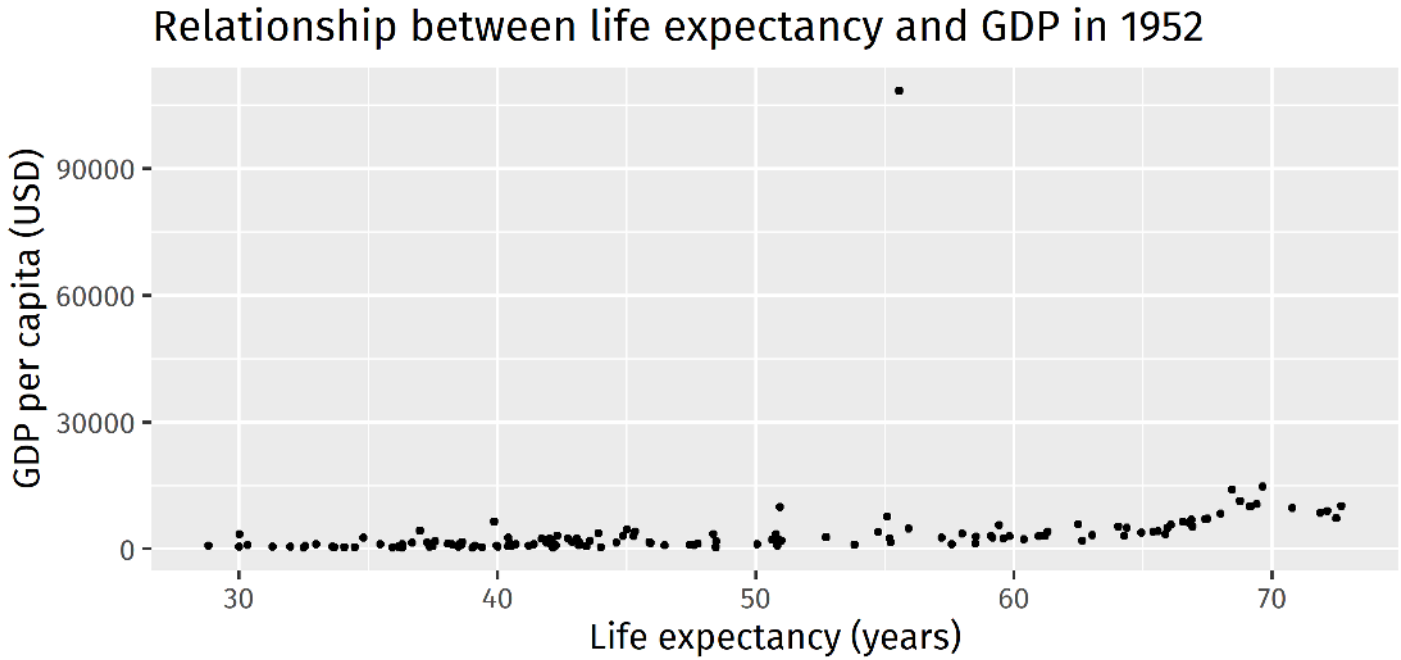
Figure: Thomas de Beus. "Think About the Grammar of Graphics When Improving Your Graphs". Medium, 2017.

`ggplot2` vocabulary

- **data**: the actual data that is plotted as *tidy* data frame
- **aesthetics/mapping**: **map variables to visual properties**
  - x- and y-coordinates, color, shapes, transparency, line type
- **geoms** - geometric objects
  - points, bars, lines, histograms, etc.
- **stats** - data transformations (often implicit)
  - counts of categories for bar charts, summary statistics for a boxplot, regression parameters, etc.
- **scales** - translate between variable ranges and visual properties
  - which color should represent which category?, should the y-axis be log-transformed?
- **facets** - spread data onto multiple subplots/panels
- **coordinates** - change and adjust the coordinate system
  - cartesian, polar or cartographic coordinate system
- **themes** - additional visual settings not related to the data
  - font size or background color

First ggplot2 visualization

Plot | Code



First ggplot2 visualization

Plot | Code

- Which data subset is being plotted?
- What does each part of the code do?
- Which variables map to which **aes**thetical features of the plot?

```
ggplot(
  data = filter(gapminder, year == 1952),
  mapping = aes(x = lifeExp, y = gdpPercap)
) +
  geom_point() +
  labs(
    x = "Life expectancy (years)",
    y = "GDP per capita (USD)",
    title = "Relationship between life expectancy and GDP in 1952"
  )
```

First ggplot2 visualization

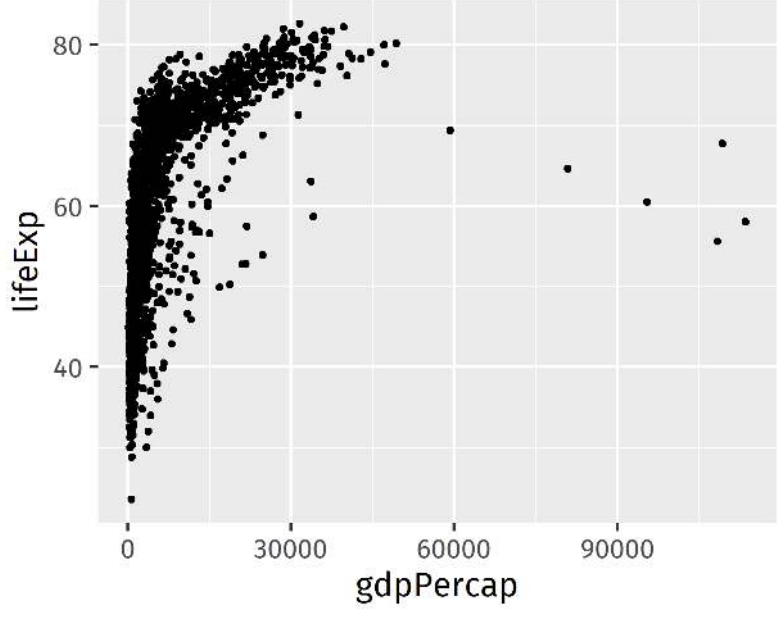
The first step in creating a `ggplot2` graph is to define a `ggplot` object with the `ggplot()` function. The main arguments are:

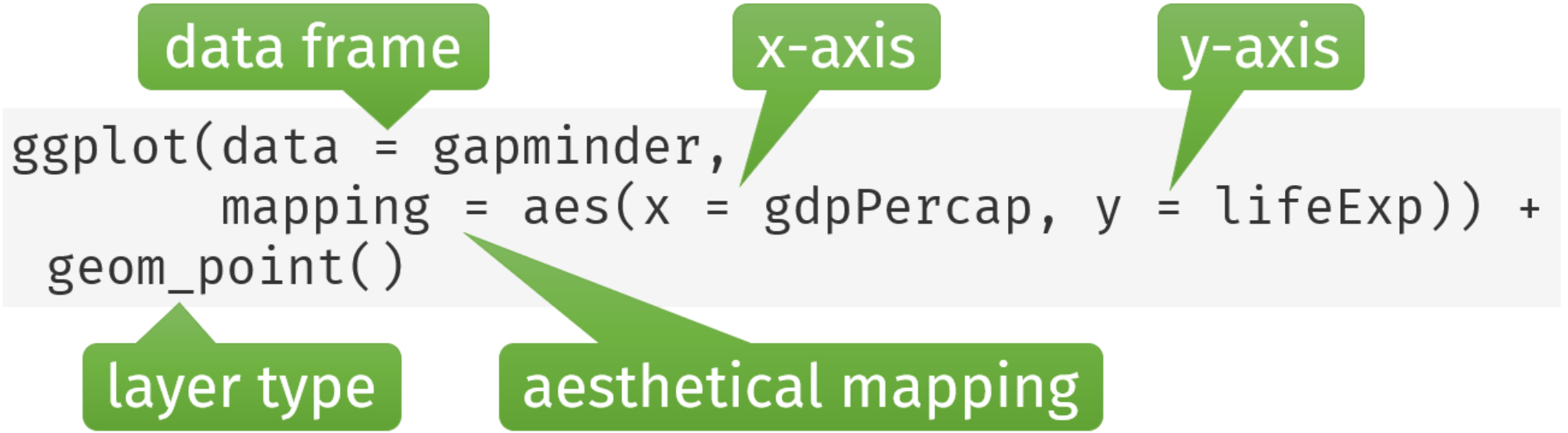
- `data`: the data frame associated with the graph
- `mapping`: the **aes**thetical mapping, i.e., which variables from the data will be mapped to the x- or y-position, color, shape, transparency, etc.

After initializing the graph, we continuously stack **layers** on top of (like LEGO blocks) with the `+` operator.

For example, we would like to create a graph from the Gapminder data, showing `gdpPercap` and `lifeExp` as scatterplot **geom**etry.

```
library(tidyverse) # loads also ggplot2
ggplot(
  data = gapminder,
  mapping = aes(x = gdpPercap, y = lifeExp)
) +
  geom_point()
```



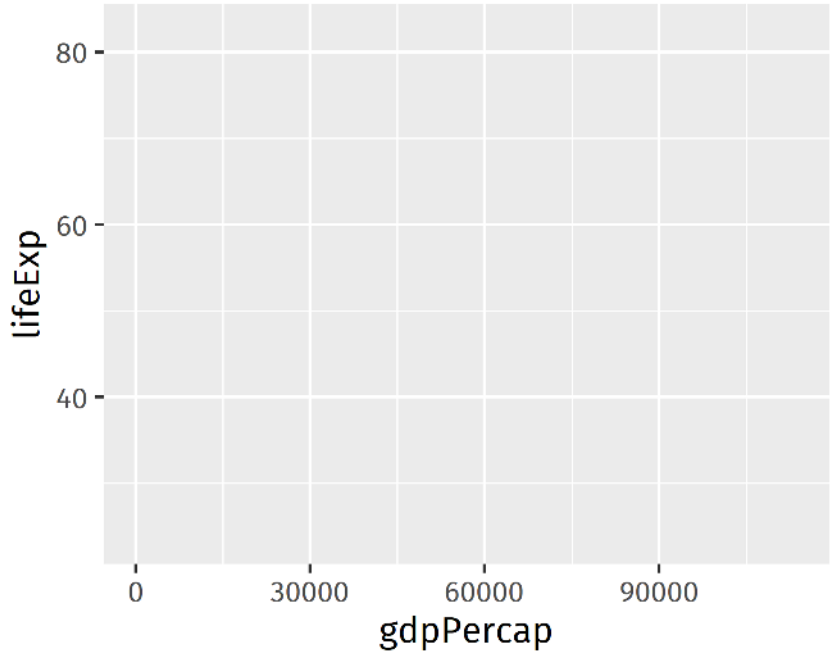

data frame
x-axis
y-axis
ggplot(data = gapminder,
mapping = aes(x = gdpPercap, y = lifeExp)) +
geom_point()
layer type
aesthetical mapping

Step 1 | Step 2 | Step 3a | Step 3b | Step 4

```
ggplot(data = gapminder)
```

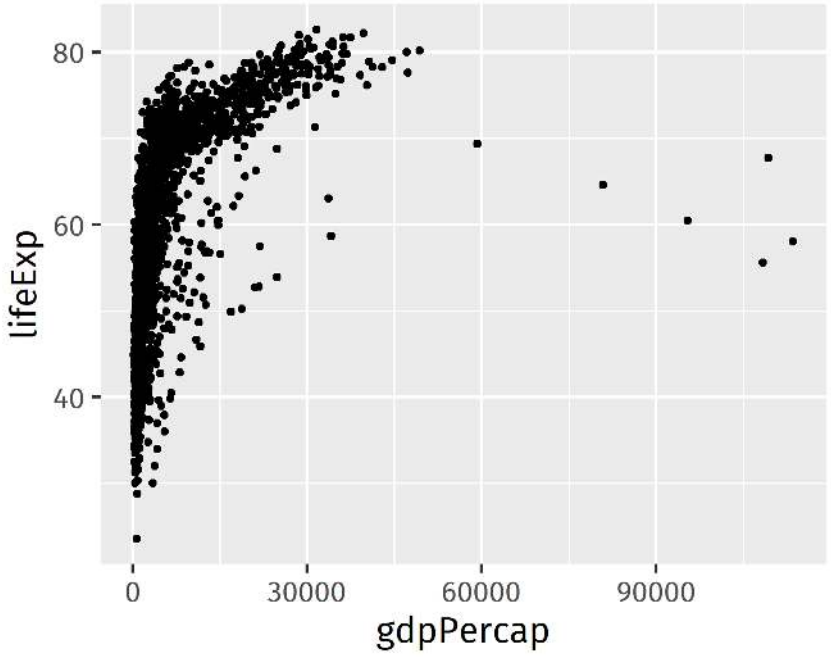
Step 1 | **Step 2** | Step 3a | Step 3b | Step 4

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp))
```



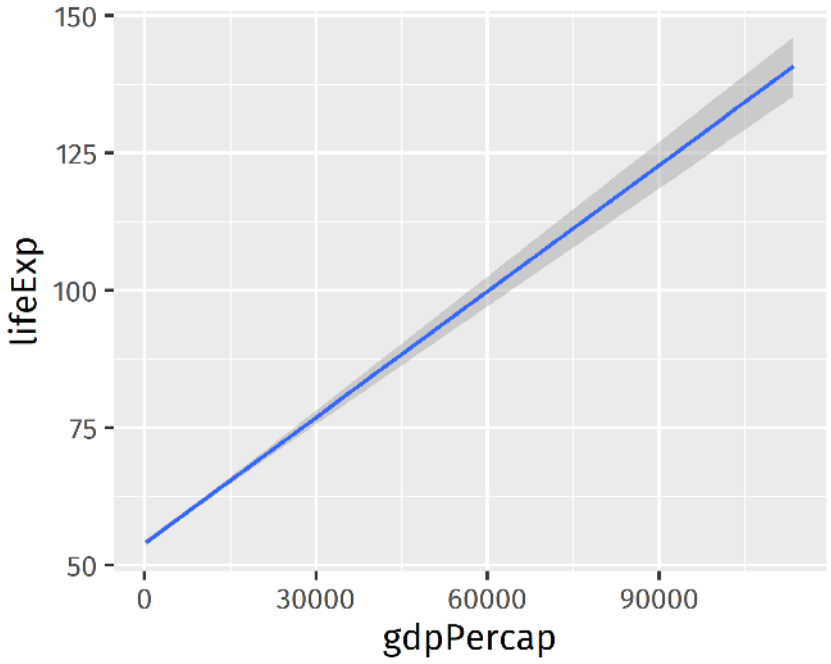
Step 1 | Step 2 | **Step 3a** | Step 3b | Step 4

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() # adds a scatterplot layer
```



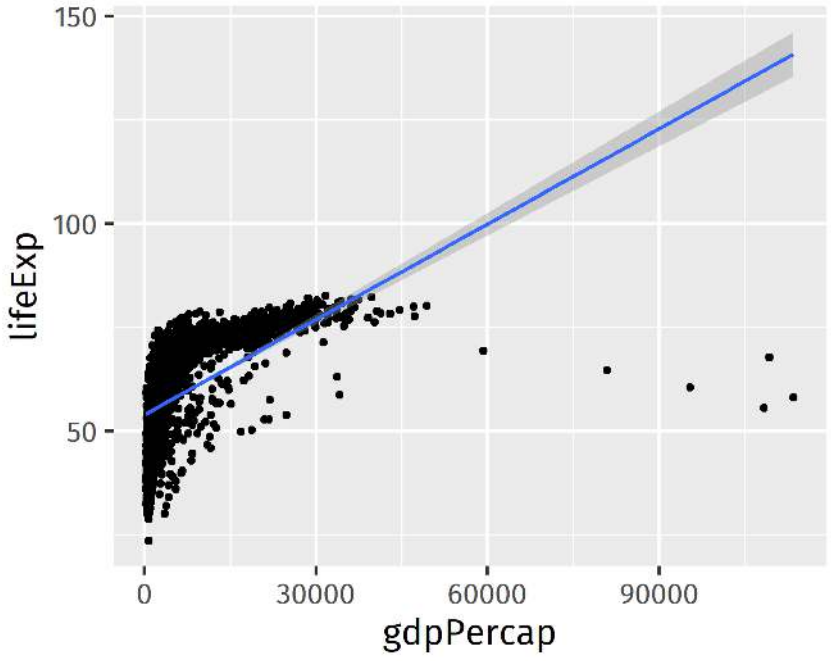
Step 1 | Step 2 | Step 3a | **Step 3b** | Step 4

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_smooth(method = "lm") # adds a trend line (lm = linear regression fit)
```



Step 1 | Step 2 | Step 3a | Step 3b | **Step 4**

```
# Add both scatterplot layer and trend line layer
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  geom_smooth(method = "lm")
```



Overview

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics (https://amzn.to/2ef1eWp). You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Installation

```
# The easiest way to get ggplot2 is to install the whole tidyverse:
install.packages (https://rdrr.io/r/utils/install.packages.html)("tidyverse")

# Alternatively, install just ggplot2:
install.packages (https://rdrr.io/r/utils/install.packages.html)("ggplot2")
```

Links

Download from CRAN at https://cloud.r-project.org/package=ggplot2 (https://cloud.r-project.org/package=ggplot2)

Browse source code at https://github.com/tidyverse/ggplot2/ (https://github.com/tidyverse/ggplot2/)

Report a bug at https://github.com/tidyverse/

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot(data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
stat = <STAT>, position = <POSITION>) +                   Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map) + **expand_limits**(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

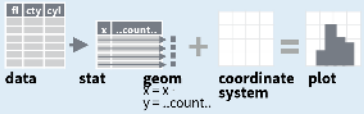
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

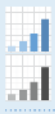
scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)



n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

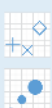
o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES



p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

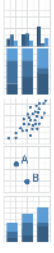
r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequa,a.area, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.



s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

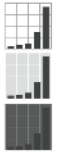
e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes



r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))

α^{c}	α^{d}	α^{e}	α^{p}	α^{r}

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

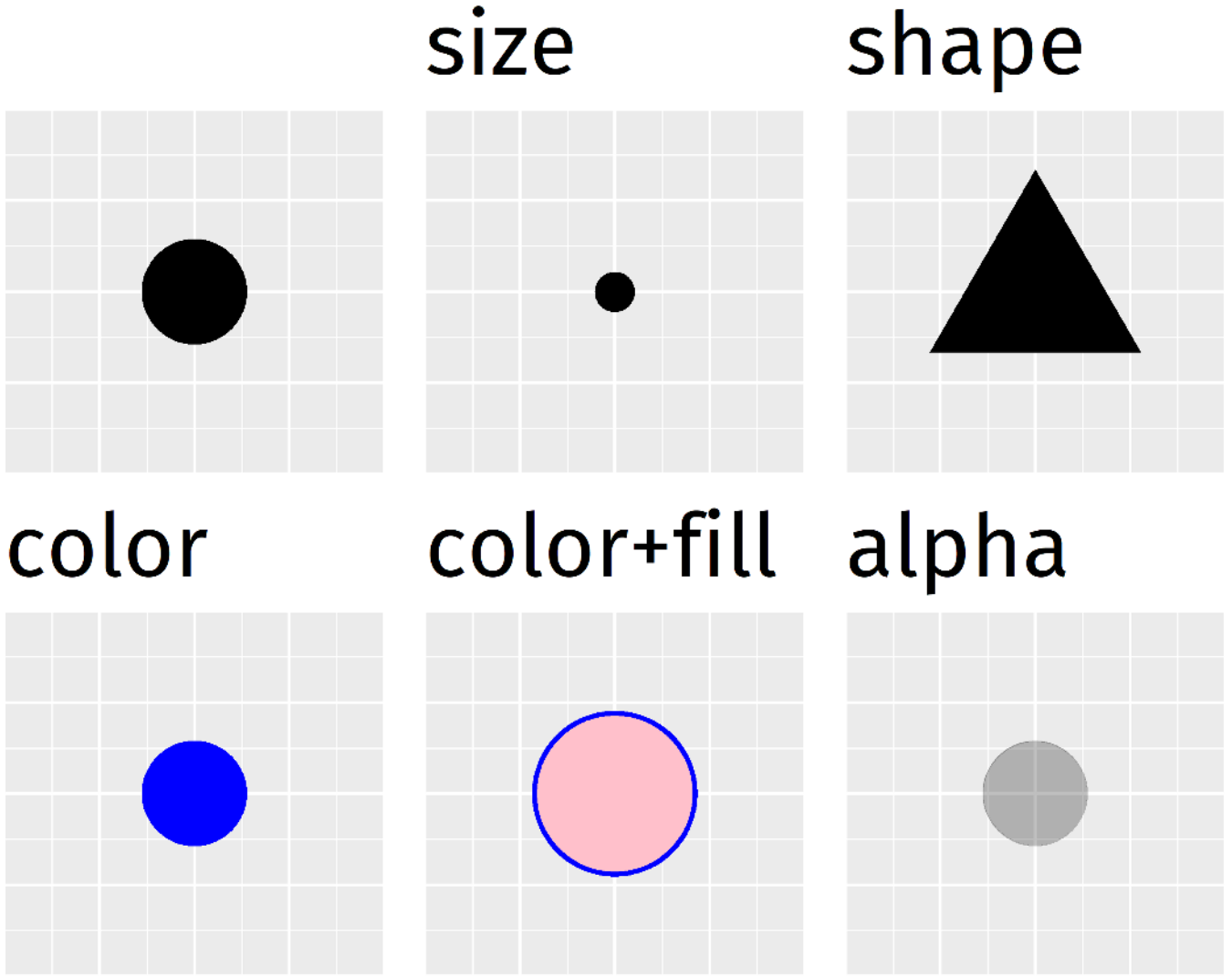
Zooming

Without clipping (preferred)
t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + xlim(0, 100) **+ ylim**(10, 20)

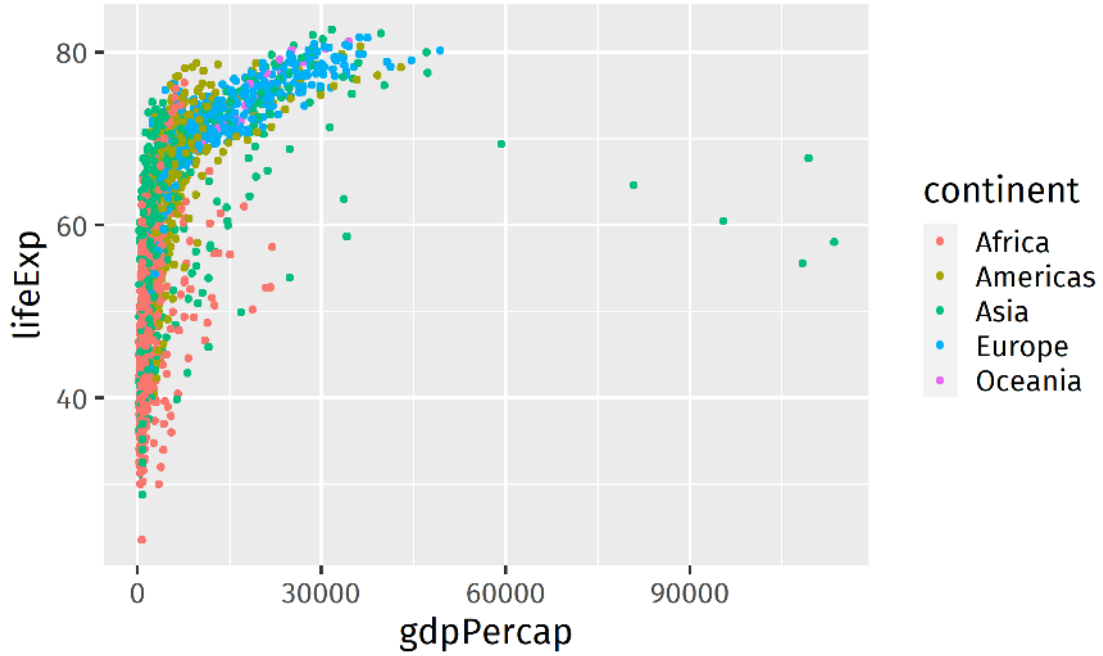
t + scale_x_continuous(limits = c(0, 100)) +
scale_y_continuous(limits = c(0, 100))

Further aesthetics



Further aesthetics

```
ggplot(
  gapminder,
  aes(
    x = gdpPercap, y = lifeExp,
    color = continent
  )
) + geom_point()
```



color		continent
red	↔	Africa
olive	↔	Americas
green	↔	Asia
blue	↔	Europe
pink	↔	Oceania

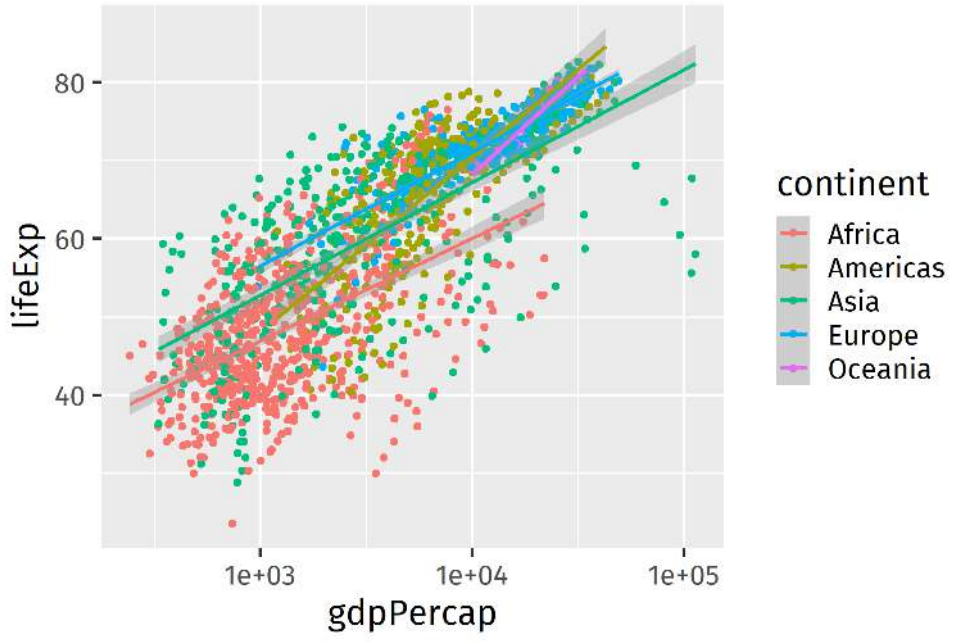
By default, `ggplot2` always creates a legend for mapping variables.

Global vs. local aesthetics

We can specify the aesthetic mapping either **globally** within the `ggplot()` function or **individually** for a specific layer within a `geom_*()` function.
If set globally, the aesthetic mapping takes effect on **all** geom layers.

Global:

```
# continent is mapped to color for
# all underlying layers
ggplot(
  data = gapminder,
  mapping = aes(
    x = gdpPercap,
    y = lifeExp,
    color = continent
  ))  +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10()
```

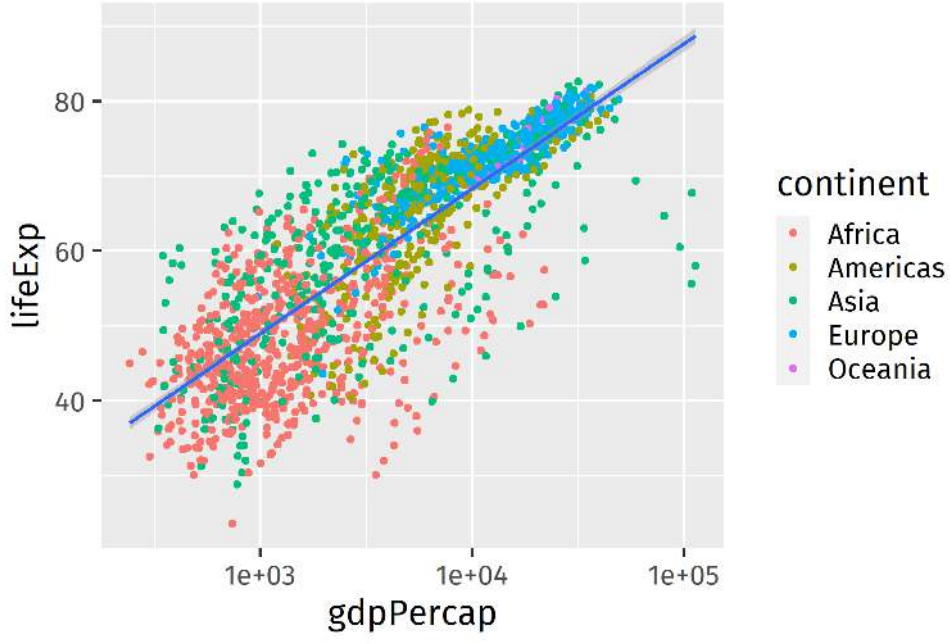


Global vs. local aesthetics

We can specify the aesthetic mapping either **globally** within the `ggplot()` function or **individually** for a specific layer within a `geom_*()` function.
If set globally, the aesthetic mapping takes effect on **all** geom layers.

Local:

```
# continent is mapped to color only
# for scatterplot layer
ggplot(
  data = gapminder,
  mapping = aes(x = gdpPercap, y = lifeExp)
) +
  geom_point(mapping = aes(color = continent)) +
  geom_smooth(method = "lm") +
  scale_x_log10()
```



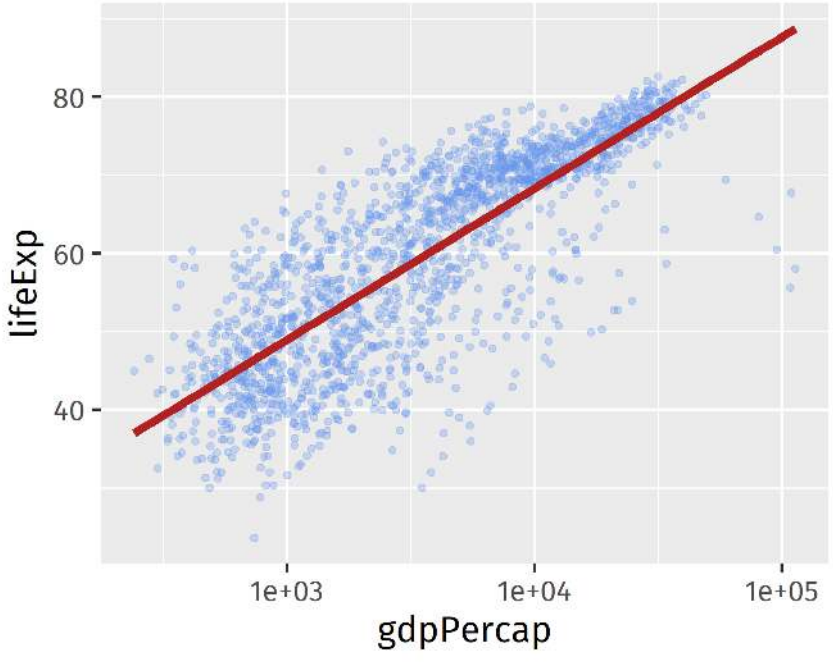
Note that the legend keys have also changed. Since the color mapping does not apply on the regression line anymore, the legend keys only show a point instead of a point and a line.

Setting layer arguments

Layer arguments that are **independent from the underlying data frame** are set outside of `aes()`.

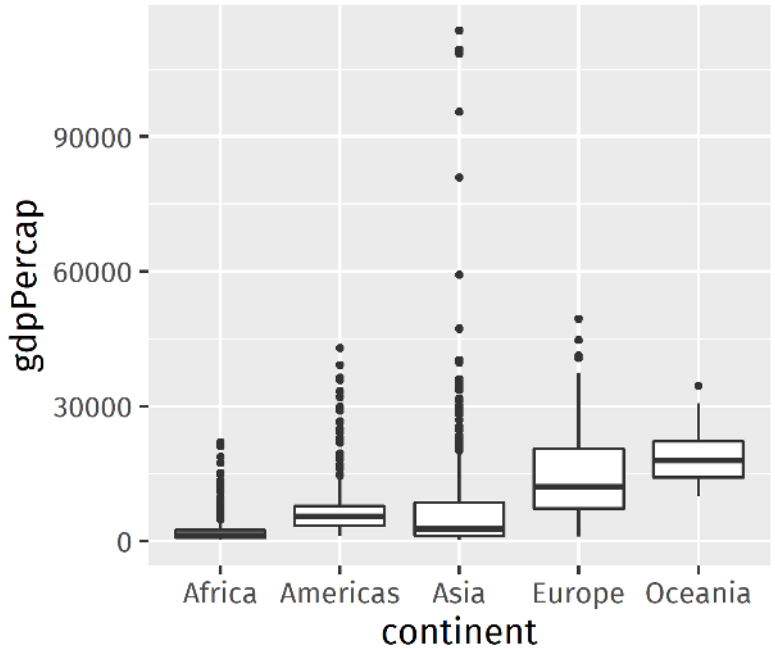
For example, we can make some cosmetic adjustments by setting the points' **color** and transparency (**alpha**), the line's color and **size**. Further, we remove the confidence interval (**se**) of the linear regression fit.

```
ggplot(
  data = gapminder,
  mapping = aes(x = gdpPercap, y = lifeExp)
) +
  geom_point(
    alpha = 0.3,
    color = "cornflowerblue"
  ) +
  geom_smooth(
    method = "lm",
    color = "firebrick",
    se = FALSE,
    size = 2
  ) +
  scale_x_log10()
```



The following example shows an **incorrect** use of geom arguments. We would like to show multiple boxplots depicting the distribution of GDP/capita for each continent and adjust the boxplots' line size to `0.75`.

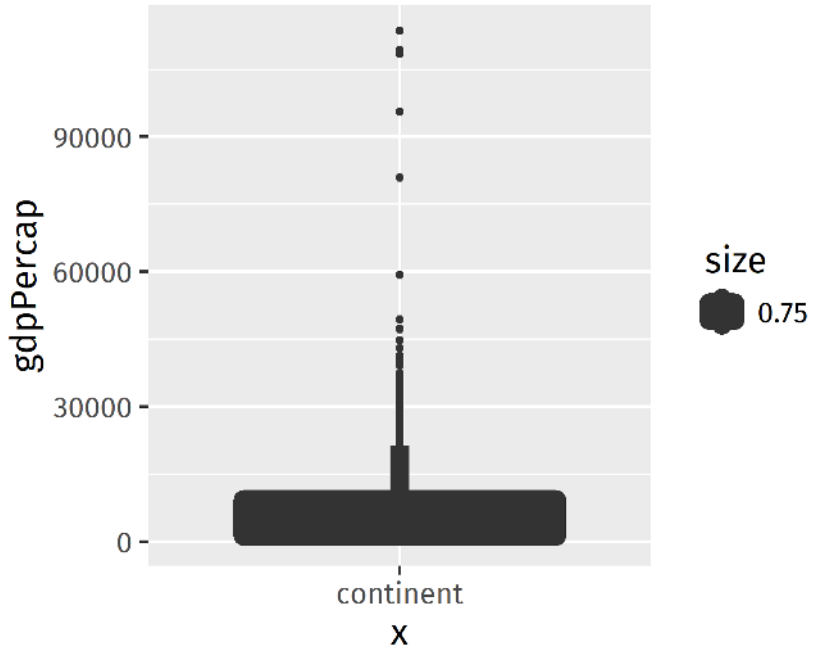
What we actually wanted | What we got



The following example shows an **incorrect** use of geom arguments. We would like to show multiple boxplots depicting the distribution of GDP/capita for each continent and adjust the boxplots' line size to `0.75`.

What we actually wanted | What we got

```
ggplot(gapminder) + geom_boxplot(
  aes(x = "continent", y = gdpPercap,
      size = 0.75)
)
```

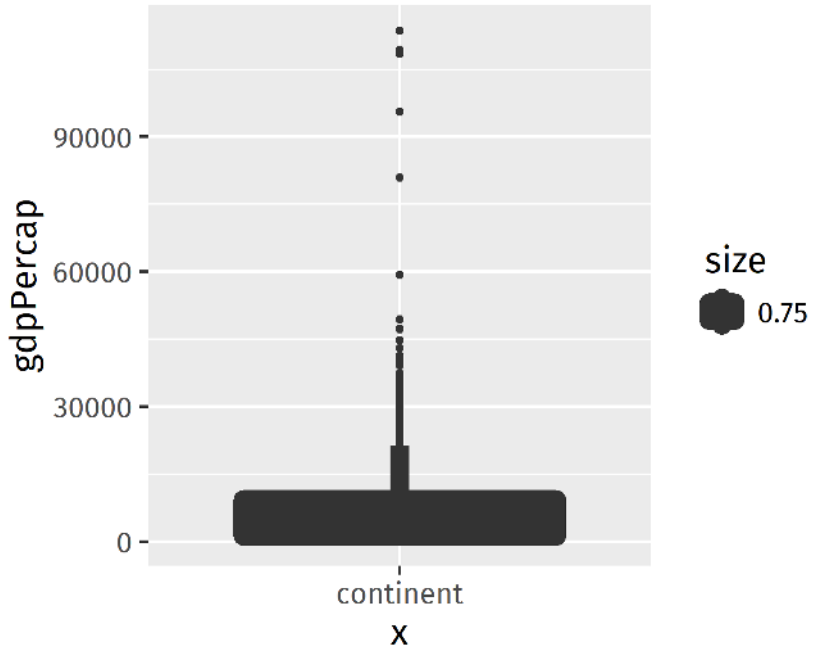


To fix the two problems of the code, we have to adhere to the following principles:

1. Within `aes()`, variables names must be passed as **expressions**, i.e., **without quotes**.
2. **Non-aesthetic arguments** must be set **outside** of `aes()`.

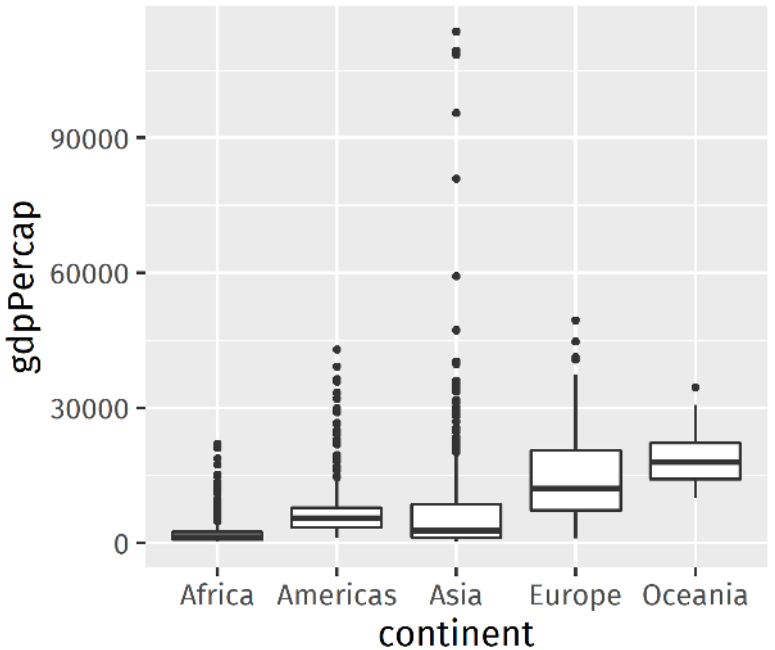
😭

```
ggplot(gapminder) + geom_boxplot(
  aes(x = "continent", y = gdpPercap,
      size = 0.75)
)
```



😊

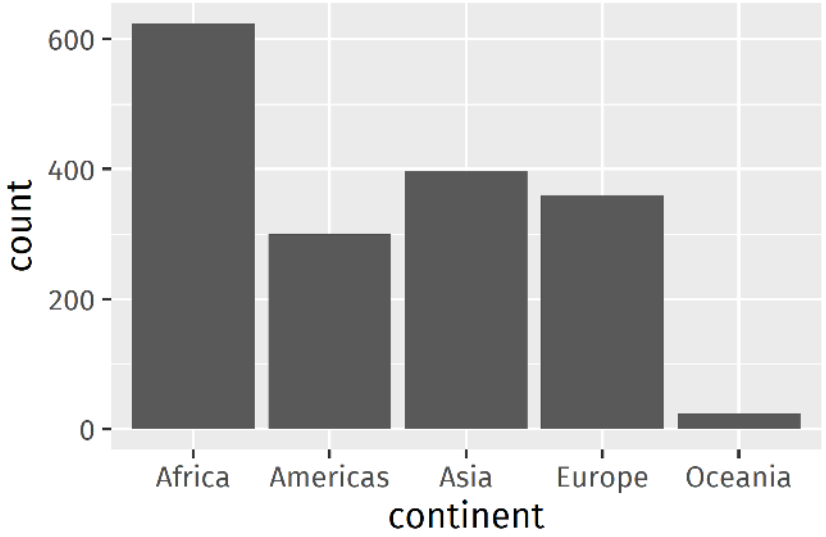
```
ggplot(gapminder) +
  geom_boxplot(
    aes(x = continent, y = gdpPercap),
    size = 0.75
  )
```



Stats

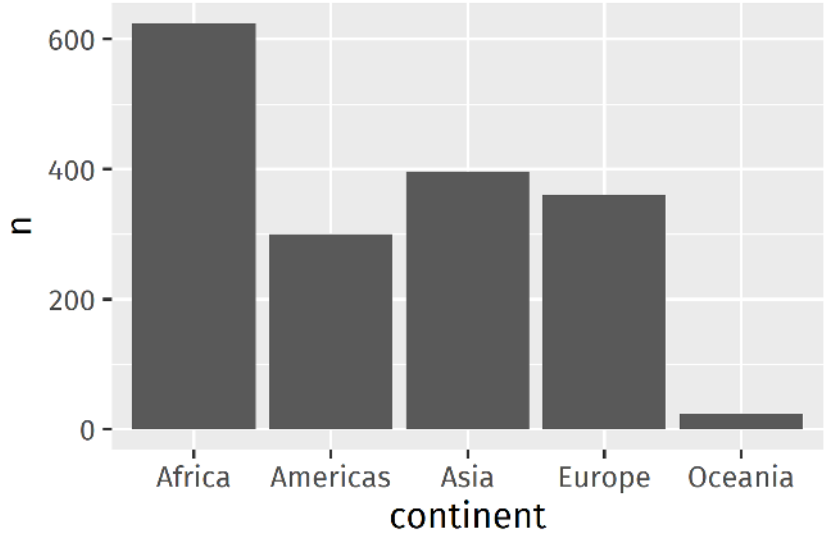
Stats are linked to geometries. Every geom has a default stat.

```
gapminder %>%
  ggplot(aes(x = continent)) +
  geom_bar(stat = "count") # default
```



`stat = "count"` automatically computes the number of observations for each category, which is the variable mapped to the x-aesthetic.

```
gapminder %>% count(continent) %>%
  ggplot(aes(x = continent, y = n)) +
  geom_bar(stat = "identity")
```

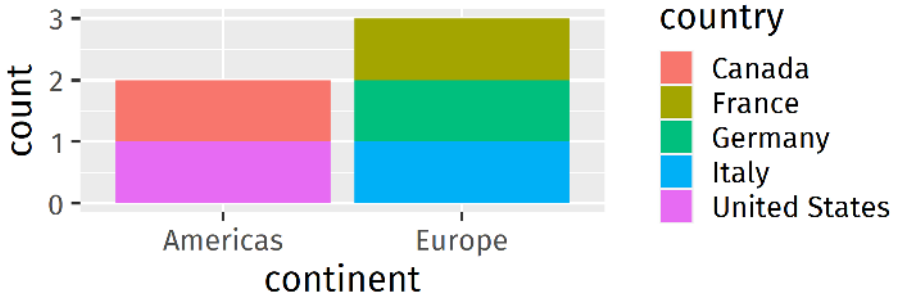


`stat = "identity"` requires to specify a variable that is mapped to `y` (bar height).

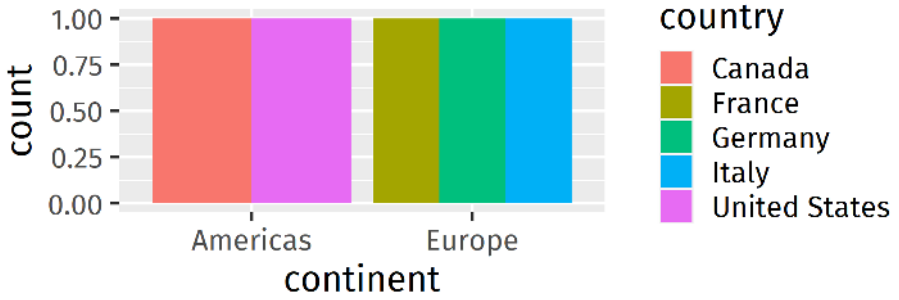
Position adjustment

```
selected_c <- c("Germany", "France", "Italy", "United States", "Canada")
s07 <- filter(gapminder, year == 2007, country %in% selected_c)
```

```
ggplot(s07, aes(continent, fill = country)) +
  geom_bar() # default: position_stack()
```




```
ggplot(s07, aes(continent, fill = country)) +
  geom_bar(position = position_dodge())
```




```
ggplot(s07, aes(continent, fill = country)) +
  geom_bar(position = position_stack())
```

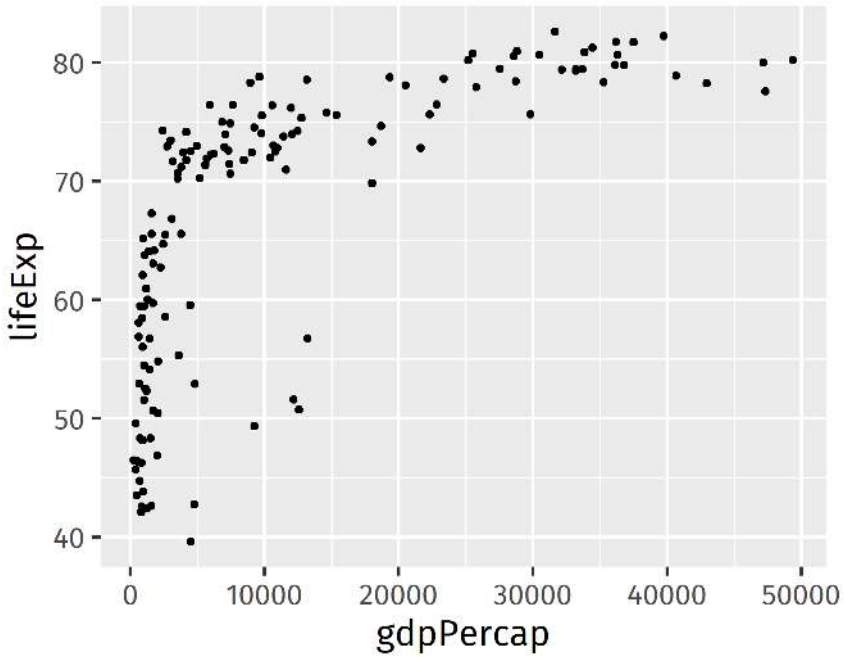
```
ggplot(s07, aes(continent, fill = country)) +
  geom_bar(position = position_fill())
```

Position adjustment

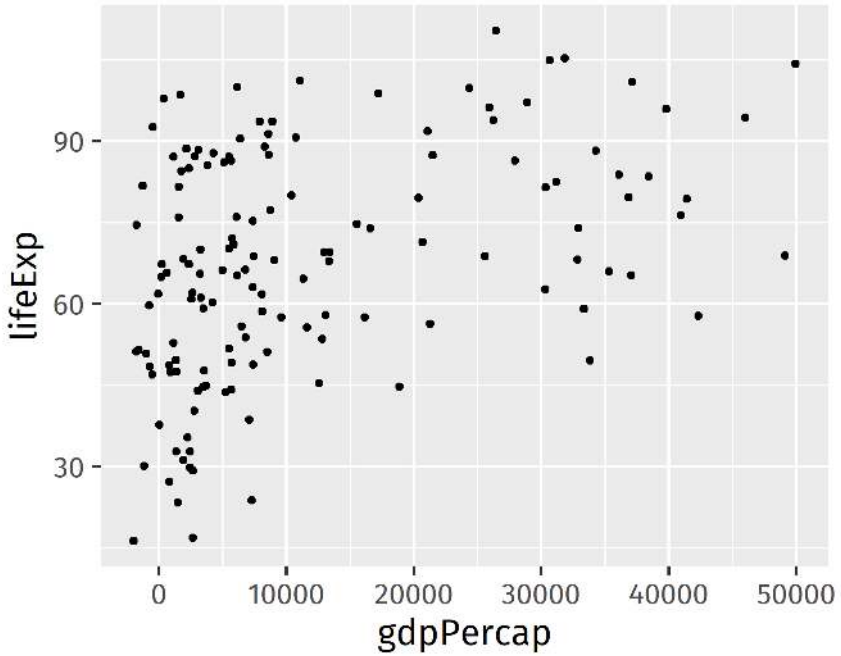
```
g07 <- filter(gapminder, year == 2007)
```

```
ggplot(g07, aes(gdpPercap, lifeExp)) +


  geom_point()
```




```
ggplot(g07, aes(gdpPercap, lifeExp)) +
 geom_point(
  position =
   position_jitter(width = 3000, height = 30)
  )
```



Scales

- Every aesthetical mapping given by `aes()` will have a scale
- If no **scale layer** is explicitly provided, a default scale will be used
- Scale function names follow an intuitive scheme: `scale<AES><TYPE>()`

Examples:

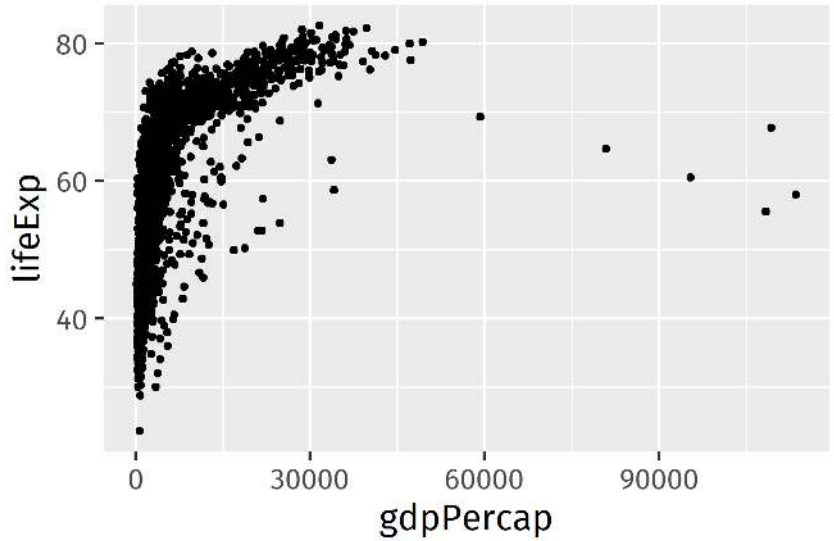
- continuous scale: `scale_<AES>_continuous()`
- discrete scale: `scale_<AES>_discrete()`
- scale with custom values: `scale_<AES>_manual()`
- scale with colors from the RColorBrewer package: `scale_{color,fill}_brewer()`
- scale with a color gradient `scale_{color,fill}_gradient()`
- ...

Except for x-/y-axis-scales, every scale will have its own **legend**.

Axis scales

Default | Explicit default | Customize scale | Log scale | Further customization

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```

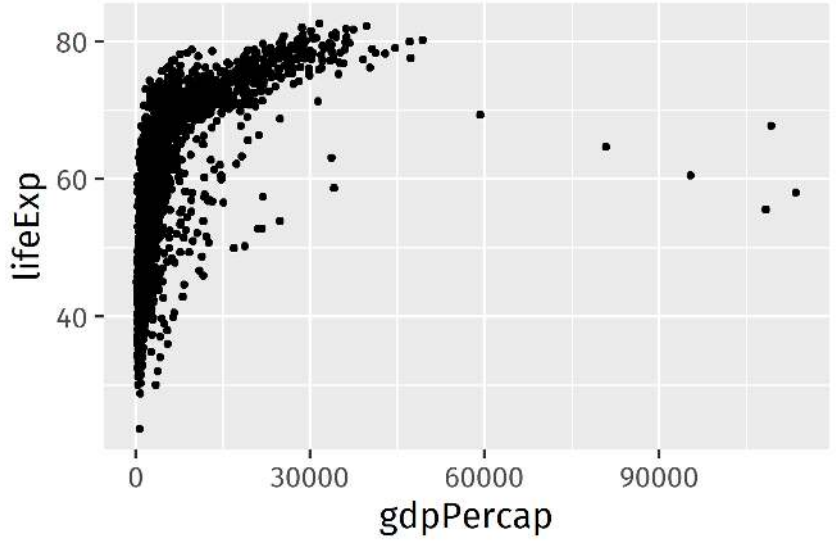


The x- and y-axis scales default to `scale_x_continuous()` and `scale_y_continuous()`, respectively. We do not need to explicitly add these layers to the graph.

Axis scales

Default | **Explicit default** | Customize scale | Log scale | Further customization

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_continuous()
```

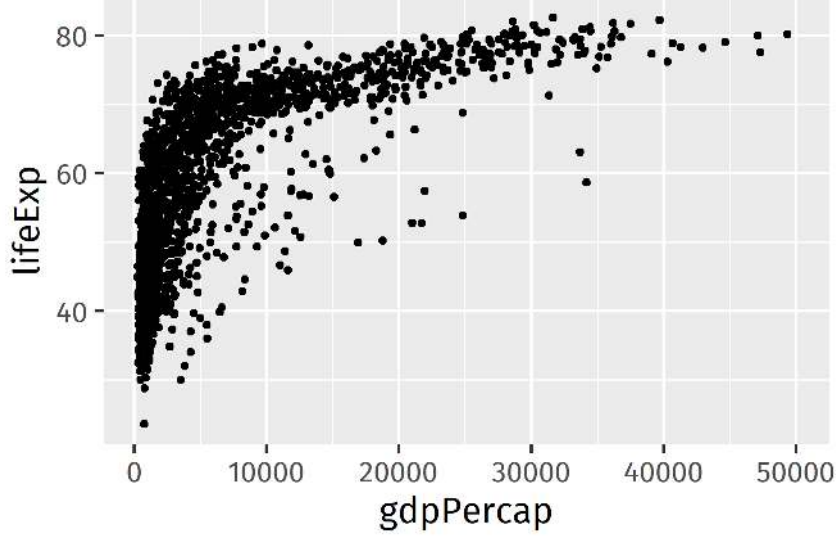


Axis scales

Default | Explicit default | **Customize scale** | Log scale | Further customization

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_continuous(limits = c(200, 50000))
```

```
## Warning: Removed 6 rows containing missing values (geom_point).
```

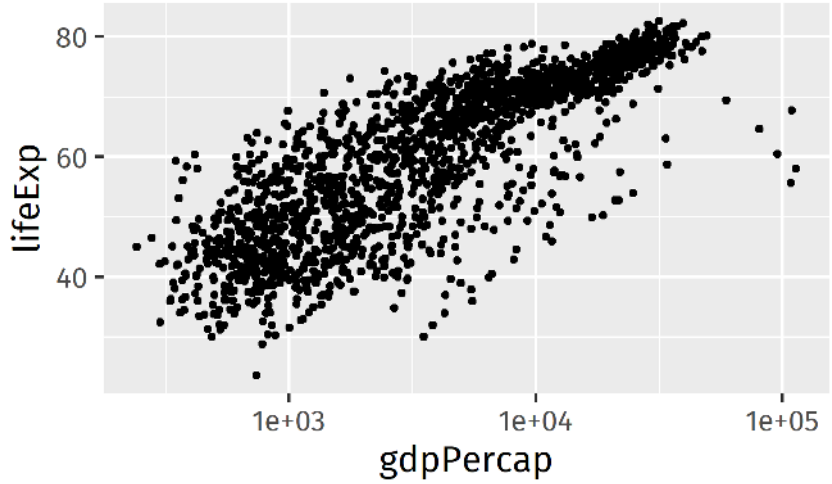


Note that we receive a warning. There are 6 observations that are outside the specified x-axis range.
Since the graph reveals a log relationship between GDP per capita and life expectancy, we may improve it by log-transforming the x-axis.

Axis scales

Default | Explicit default | Customize scale | **Log scale** | Further customization

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10()
```



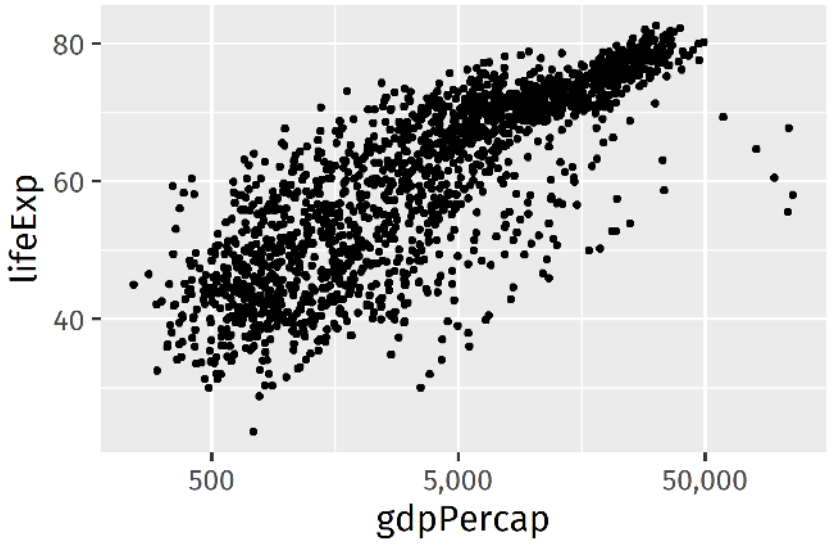
The x-axis labels in scientific location don't look particularly pretty.
We would like to make the following changes:

- make the x-axis labels more intuitive
- set custom axis breaks at 500, 5000 and 50000

Axis scales

Default | Explicit default | Customize scale | Log scale | **Further customization**

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  scale_x_log10(
    breaks = c(500, 5000, 50000), # ticks pos.
    labels = scales::comma # alternative labeling function
    # (put a comma before every three digits)
  )
```

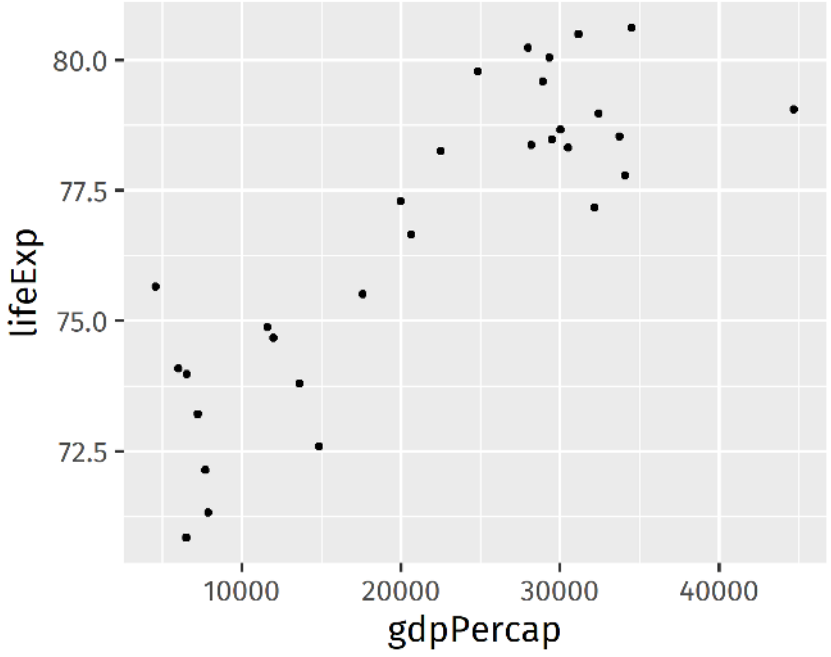


Further examples of scale adjustments

Initial plot | Custom breaks | Custom limits | Manual labels

```
ge7 <- gapminder %>% filter(year == 2002, continent == "Europe")
```

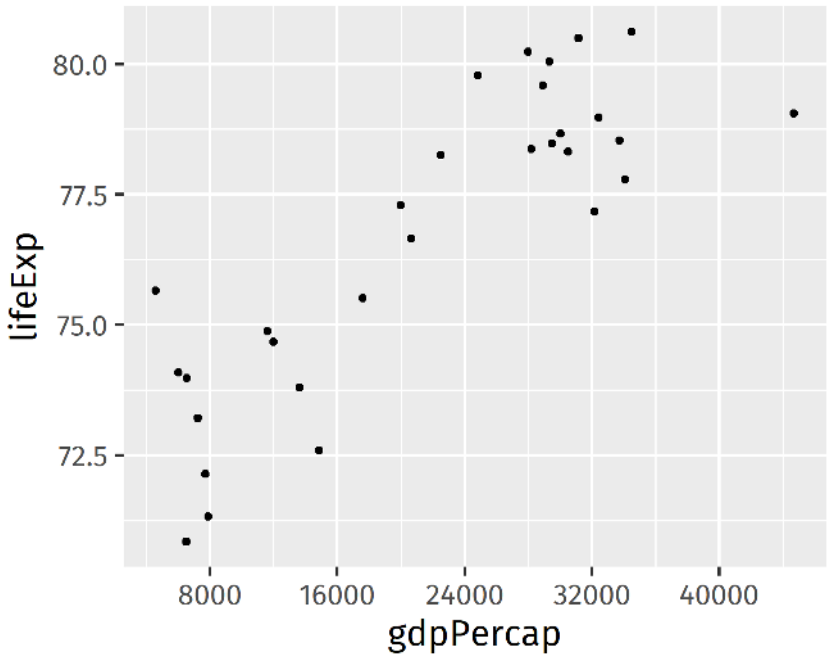
```
ggplot(ge7, aes(gdpPercap, lifeExp)) +
  geom_point()
```



Further examples of scale adjustments

Initial plot | **Custom breaks** | Custom limits | Manual labels

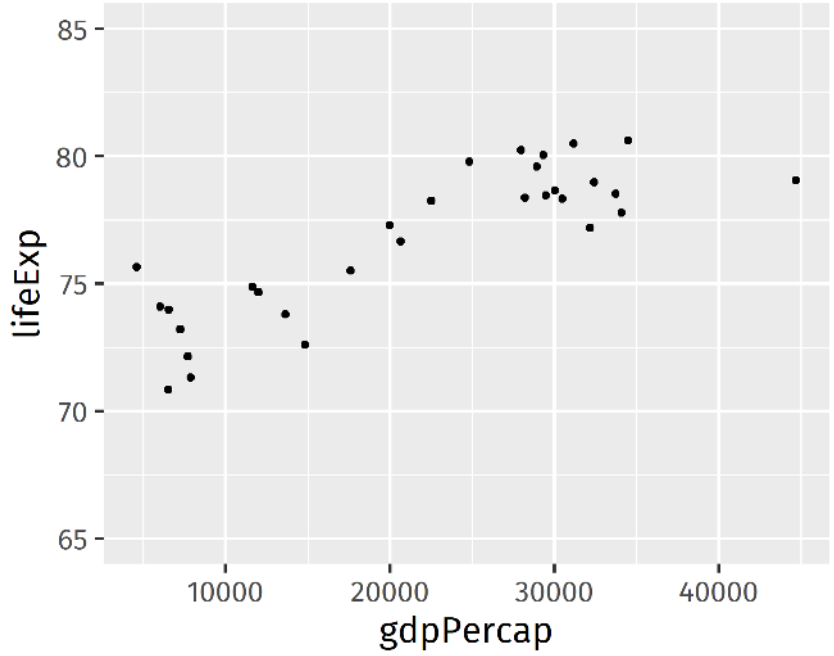
```
ggplot(ge7, aes(gdpPercap, lifeExp)) +
  geom_point() +
  scale_x_continuous(
    breaks = seq(8000, 40000, 8000)
  )
```



Further examples of scale adjustments

Initial plot | Custom breaks | **Custom limits** | Manual labels

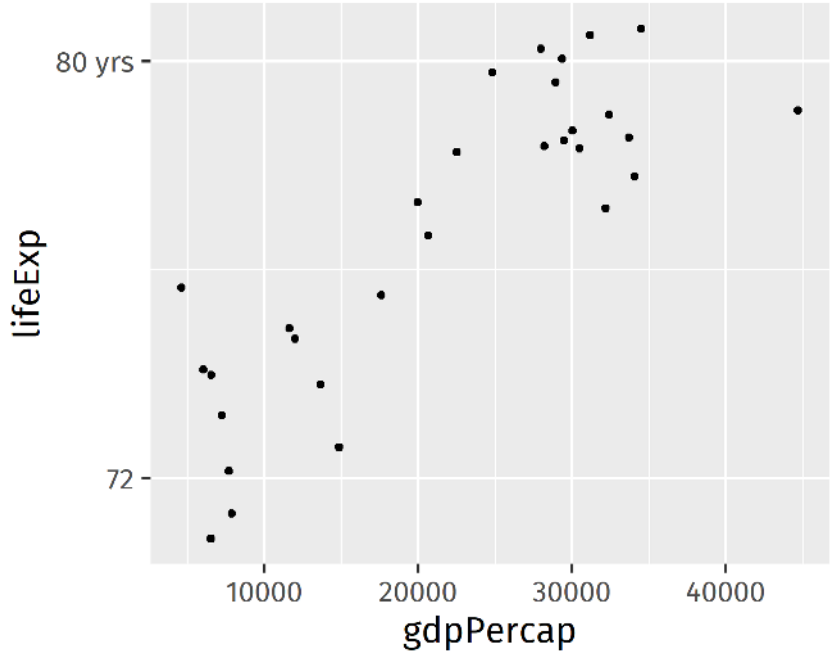
```
ggplot(ge7, aes(gdpPercap, lifeExp)) +
  geom_point() +
  scale_y_continuous(limits = c(65, 85))
```



Further examples of scale adjustments

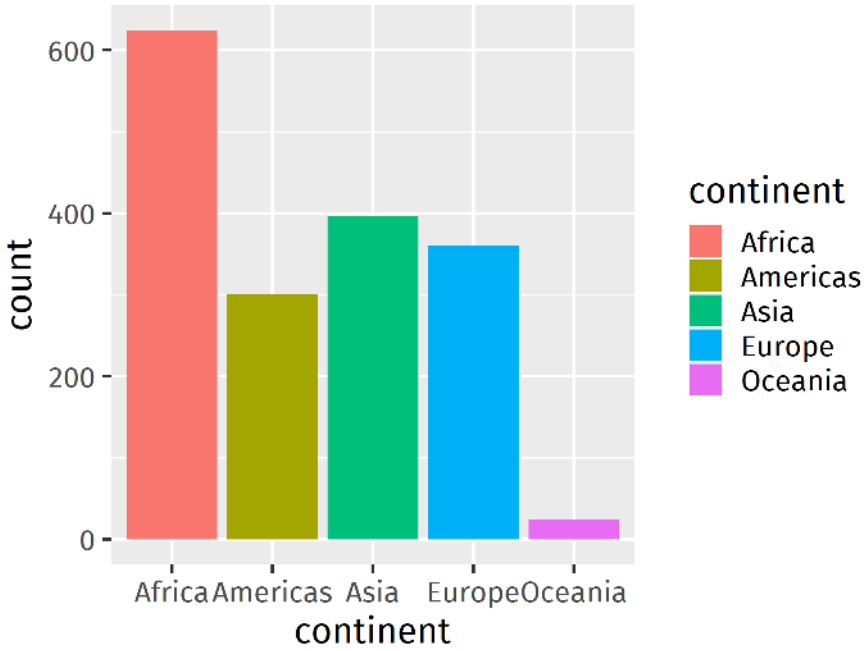
Initial plot | Custom breaks | Custom limits | **Manual labels**

```
ggplot(ge7, aes(gdpPercap, lifeExp)) +
  geom_point() +
  scale_y_continuous(breaks = c(72, 80), labels = c("72", "80 yrs"))
```



Color scales

```
ggplot(gapminder, aes(x = continent, fill = continent)) +
  geom_bar() # + scale_fill_discrete()
```

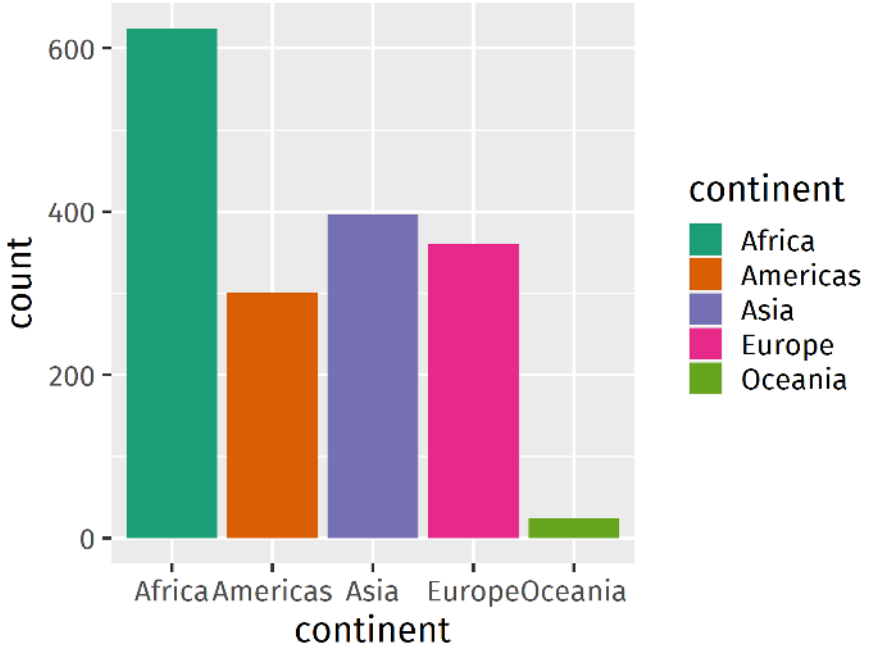


Color scales

We can replace this default color scale by adding a different `scale_fill_*` layer.

```
ggplot(gapminder, aes(x = continent, fill = continent)) +
  geom_bar() +
  scale_fill_brewer(palette = "Dark2")
```

```
RColorBrewer::display.brewer.all()
```



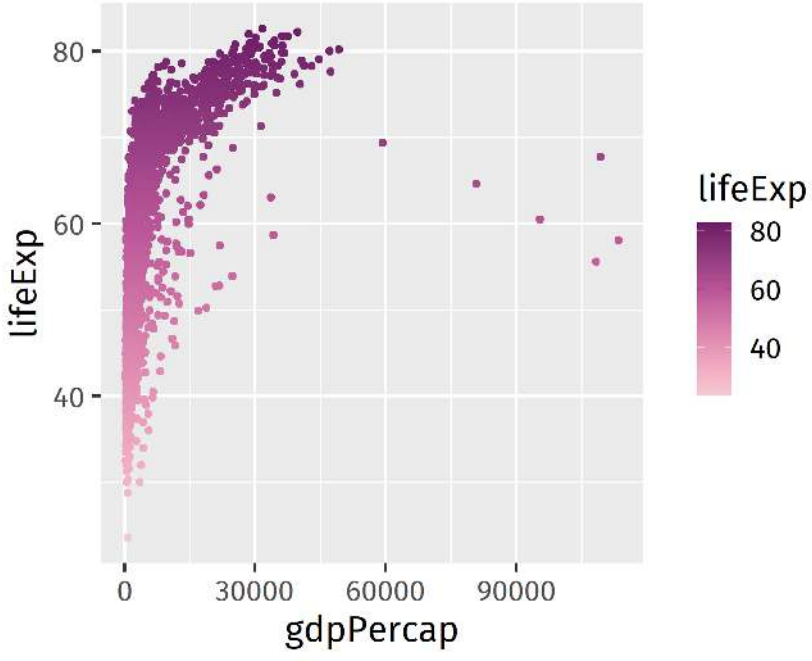
colorbrewer2.org

```
colorspace::hcl_palettes(plot = TRUE)
```

→ `colorspace::scale_<AES>_<TYPE>_<COLORSCALE>(palette = <PALETTE-NAME>)`

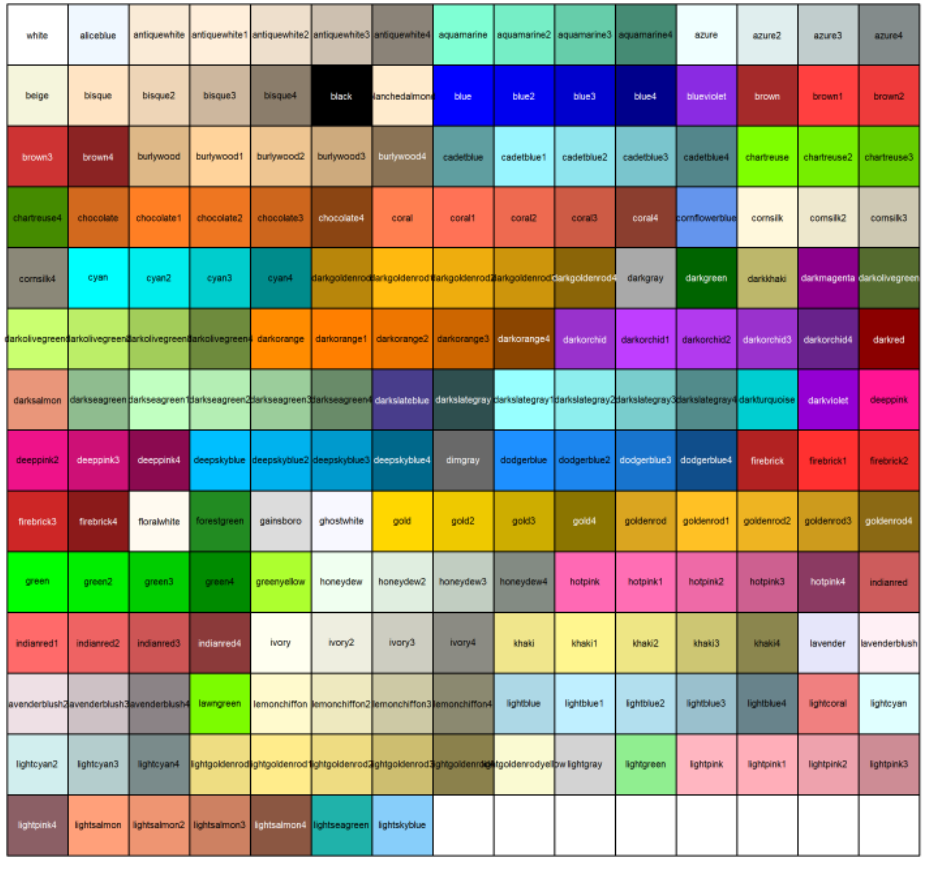
```
colorspace::scale_<AES>_<TYPE>_<COLORSCALE>(palette = <PALETTE-NAME>)
```

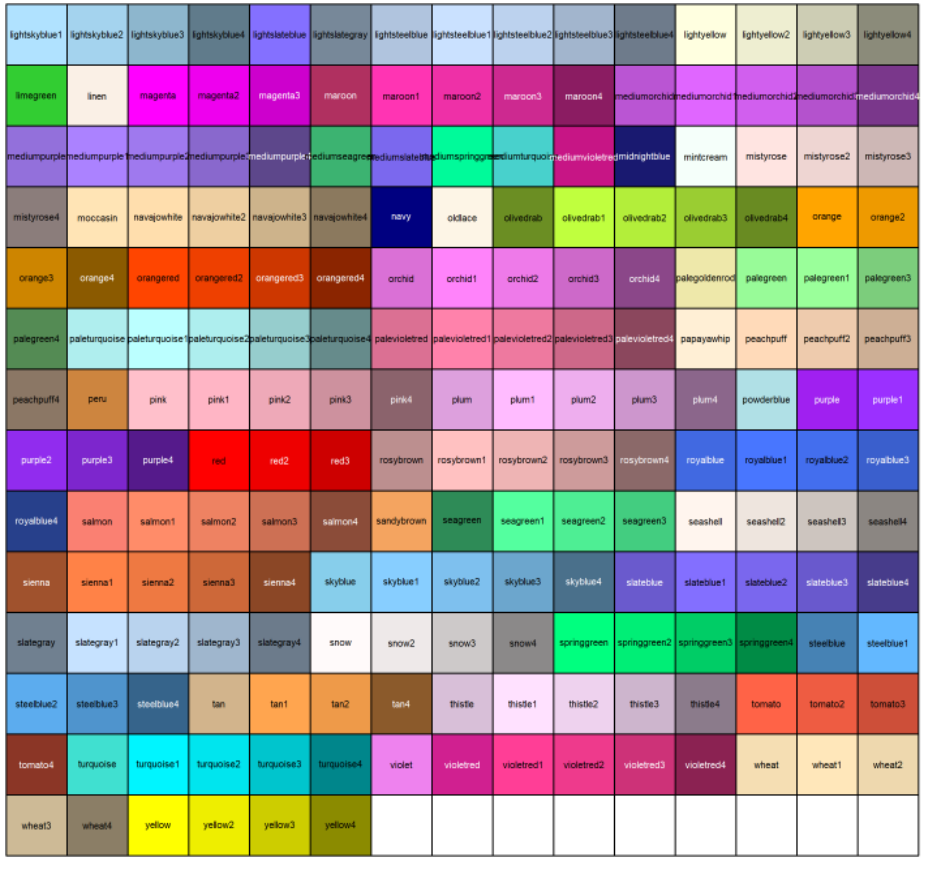
```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, color = lifeExp)) +
  geom_point() +
  colorspace::scale_color_continuous_sequential("Magenta")
```



The vector type of the variable that is mapped to the color aesthetic determines whether a color gradient is created (in case of a numeric variable) or whether a disrete color mapping is created (in case of a factor variable).

```
scales::show_col(colors()) # colors() returns the built-in color names
```





Facetting

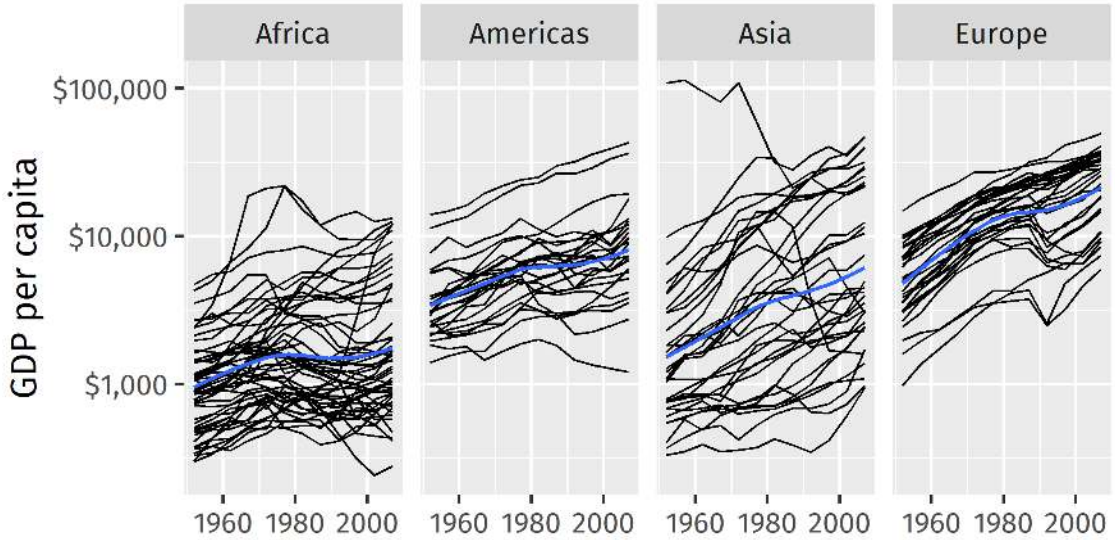
One of the highlights of `ggplot2` is the possibility to easily **facet** a plot, i.e. splitting the data onto multiple panels. Facetting allows to compactly present a lot of information by **stratifying by a third variable**. Also, faceting often is a remedy against **overplotting**.

The `facet_wrap()` function creates subpanels. Notation: `~`(tilde) comma-separated names of variables

```
ggplot(data = gapminder, mapping = aes(x = year, y = gdpPercap)) +
geom_line(aes(group = country)) +
  scale_y_log10() +
  facet_wrap(~ continent)
```

Facetting

```
p <- ggplot(data = gapminder %>% filter(continent != "Oceania"), aes(x = year,y = gdpPercap))
p + geom_line(aes(group = country)) +
  geom_smooth(method = "loess", se = FALSE) +
  scale_x_continuous(breaks = seq(1960, 2000, 20)) + scale_y_log10(labels = scales::dollar) +
  facet_wrap(~ continent, nrow = 1) +
  labs(x = NULL, y = "GDP per capita")
```



Instead of the formula notation (`~`), you can alternatively specify faceting variables with `vars()`, e.g. `facet_wrap(vars(continent))`

`facet_wrap()` vs. `facet_grid()`

- `facet_wrap()`: sequence of panels
- `facet_grid()`: matrix of panels

Show GDP development, stratified by life expectancy groups:

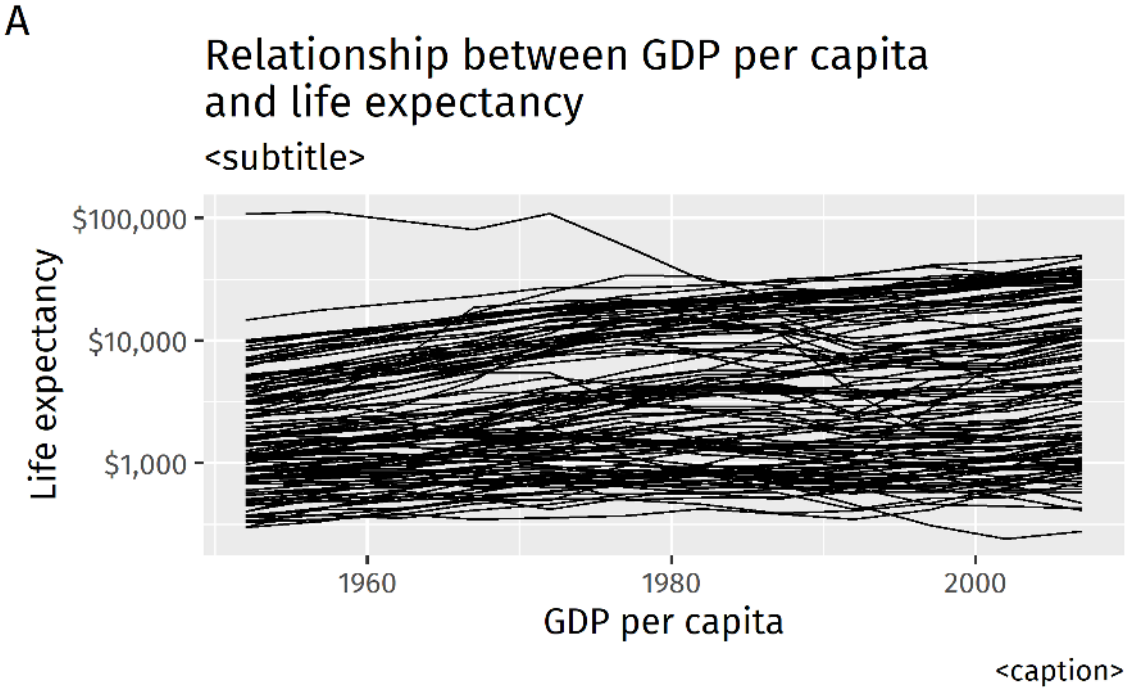
```
my_gapminder <- gapminder %>%  filter(continent %in% c("Africa", "Asia", "Europe")) %>%
  group_by(country) %>% mutate(lifeExp = if_else(max(lifeExp)<75, "lifeExp < 75", "lifeExp >= 75"))
p <- ggplot(data = my_gapminder, mapping = aes(x = year, y = gdpPercap)) +
  geom_line(aes(group = country)) +
  scale_x_continuous(breaks = seq(1960, 2000, 20)) +
  scale_y_log10(labels = scales::dollar)
```

```
p + facet_wrap(~ continent + lifeExp)
```

```
p + facet_grid(continent ~ lifeExp)
```

Labels

```
p +
  labs(
    x = "GDP per capita",
    y = "Life expectancy",
    color = "Continent",
    title = "Relationship between GDP per capita\nand life expectancy",
    subtitle = "<subtitle>",
    caption = "<caption>",
    tag = "A"
  )
```



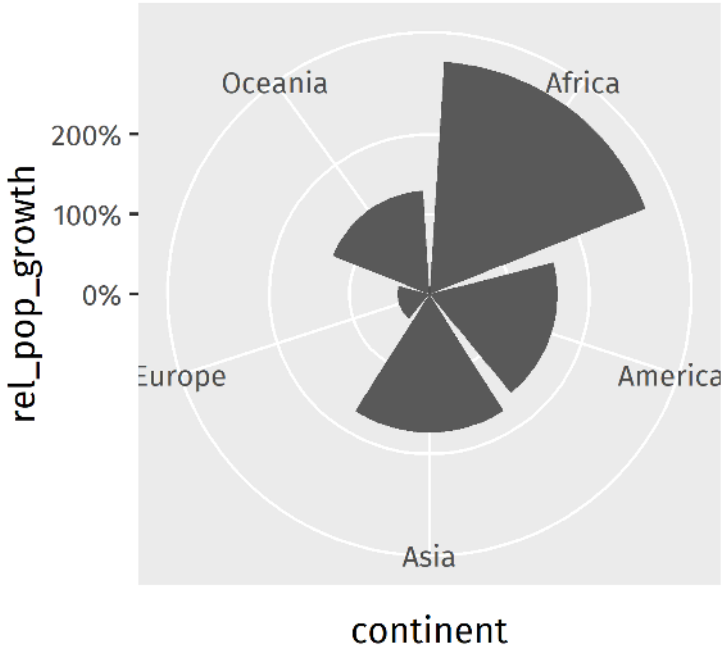
Coordinates

Specify on what type of canvas the data should be drawn on.

```
# Calculate average relative population growth
# from 1952 to 2007 per continent
(gc <- gapminder %>%
  filter(year %in% c(1952, 2007)) %>%
  group_by(continent, year) %>%
  summarize(avg_pop = mean(pop)) %>%
  group_by(continent) %>%
  summarize(rel_pop_growth =
              (avg_pop[2]-avg_pop[1]) /
              avg_pop[1]))
```

```
## # A tibble: 5 x 2
##   continent rel_pop_growth
##   <fct>               <dbl>
## 1 Africa              2.91
## 2 Americas            1.60
## 3 Asia                1.73
## 4 Europe              0.402
## 5 Oceania             1.30
```

```
ggplot(gc, aes(x=continent, y=rel_pop_growth)) +
  geom_col() +
  coord_polar() +
  scale_y_continuous(labels = scales::percent)
```



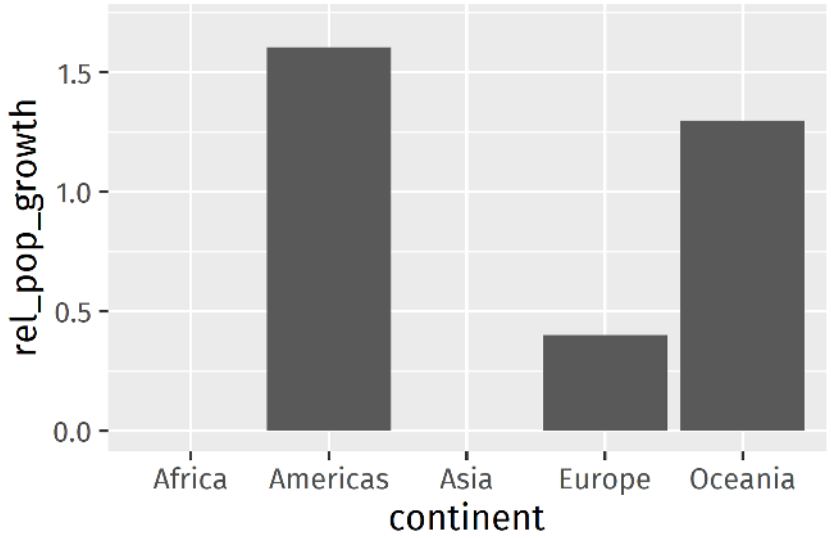
A polar coordinate system interprets x as **radius** and y as **angle**.

Coordinates

For **zooming**, use `coord_*` layers instead of `scale_*` layers.

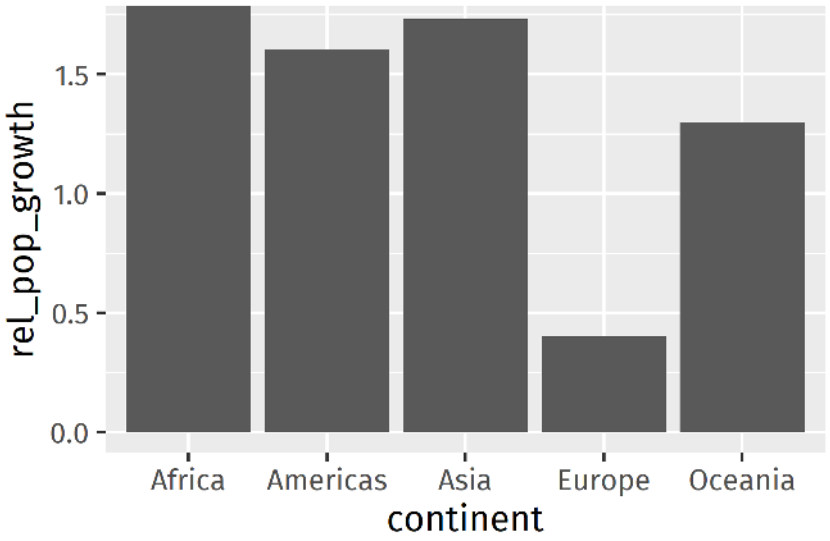
```
ggplot(gc, aes(x=continent, y=rel_pop_growth)) +
  geom_col() +
  scale_y_continuous(limits = c(0, 1.7))
```

```
## Warning: Removed 2 rows containing missing values
## (position_stack).
```



When using `scale_*`, data outside of the limits will be removed.

```
ggplot(gc, aes(x=continent, y=rel_pop_growth)) +
  geom_col() +
  coord_cartesian(ylim = c(0, 1.7))
```

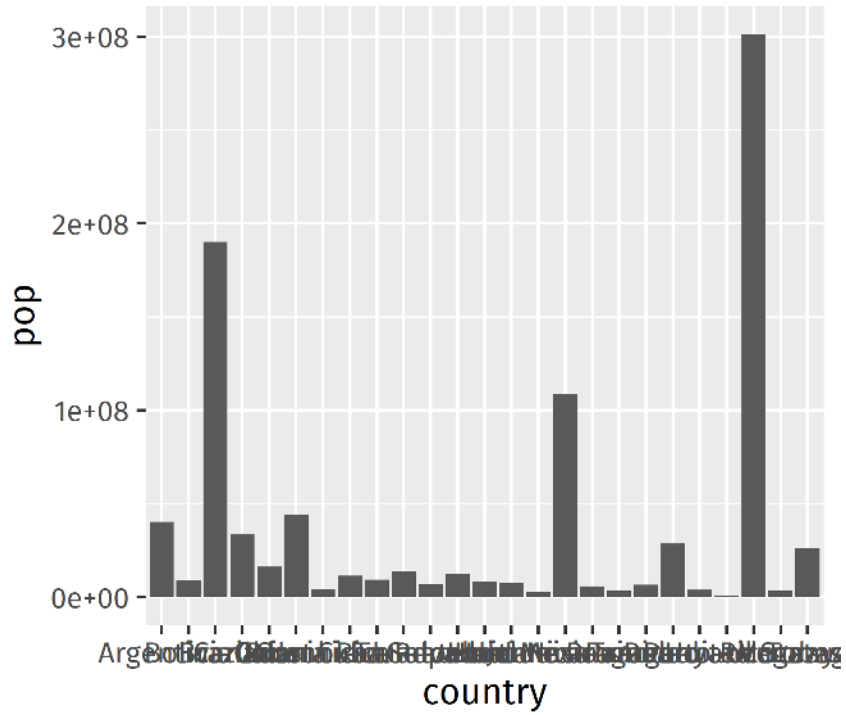


When using `coord_*`, data outside of the limits will not be removed.

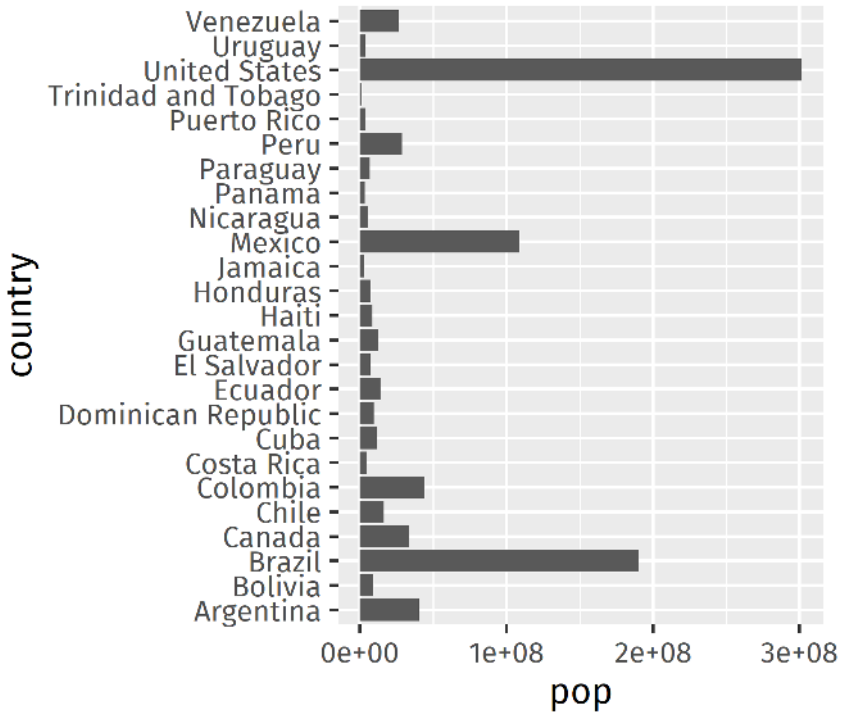
`coord_flip()` flips cartesian coordinates. It is very useful when you have a lot of categories on the x-axis or want to display very long labels.

```
gf <- filter(gapminder, year == 2007, continent == "Americas")
```

```
ggplot(gf, aes(country, pop)) +
  geom_col()
```




```
ggplot(gf, aes(country, pop)) + geom_col() +
  coord_flip()
```



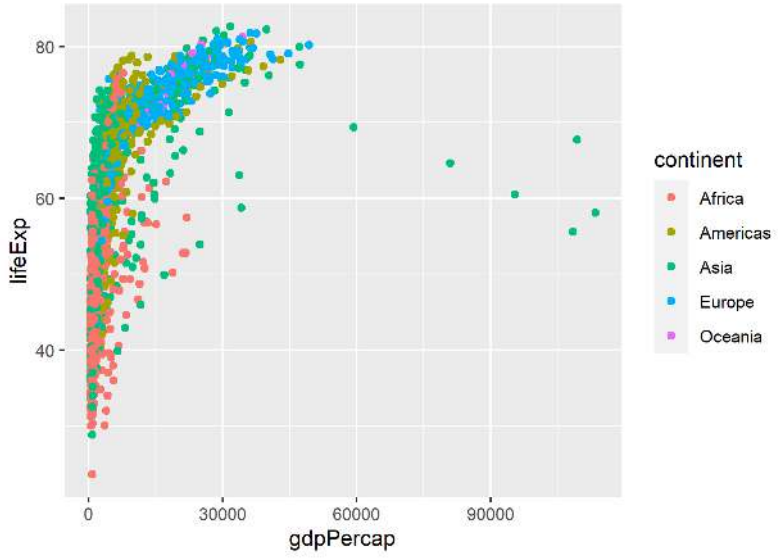
By swapping horizontal and vertical axes, the country names become readable.

Themes

Use a theme layer to change style aspects of the plot that are not related to the data.

Apply a build-in theme with `theme_<NAME>` to quickly change the overall appearance:

```
p <- ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, color = continent)) + geom_point()
p + theme_gray() # default theme
```

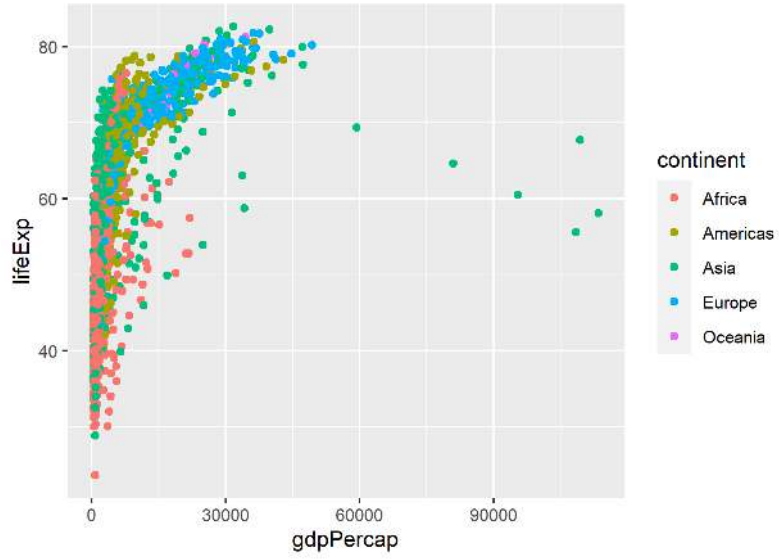


Alternative themes

theme_grey() | theme_bw() | theme_minimal() | theme_void()

> *The signature ggplot2 theme with a grey background and white gridlines, designed to put the data forward yet make comparisons easy. —* `?theme_grey`

```
p <- ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, color = continent)) + geom_point()
p + theme_gray() # default theme
```

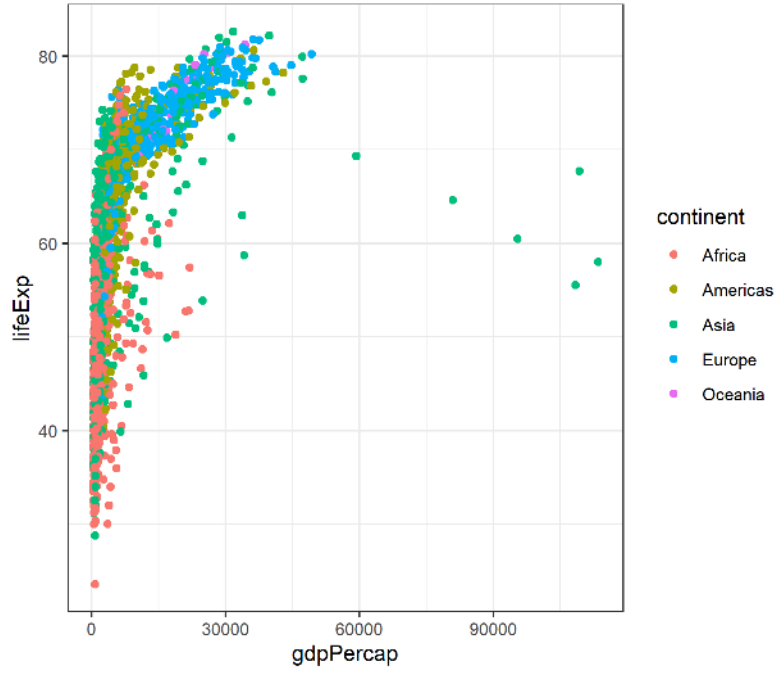


Alternative themes

theme_grey() | **theme_bw()** | theme_minimal() | theme_void()

> *The classic dark-on-light ggplot2 theme. May work better for presentations displayed with a projector. — `?theme_bw`*

```
p + theme_bw()
```



Alternative themes

theme_grey() theme_bw() **theme_minimal()** theme_void()

> *A minimalistic theme with no background annotations. —* `?theme_minimal`

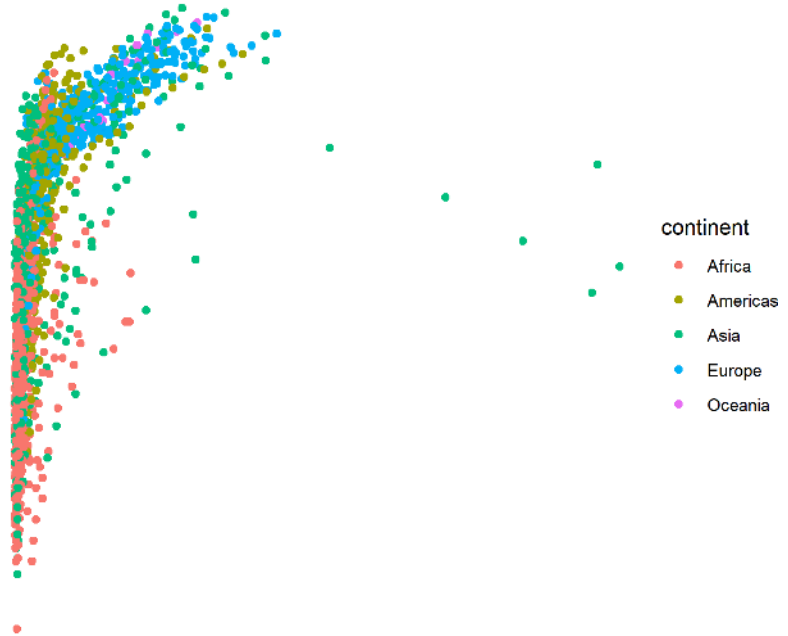
```
p + theme_minimal()
```

Alternative themes

theme_grey() theme_bw() theme_minimal() **theme_void()**

> *A completely empty theme. —* `?theme_void`

```
p + theme_void()
```



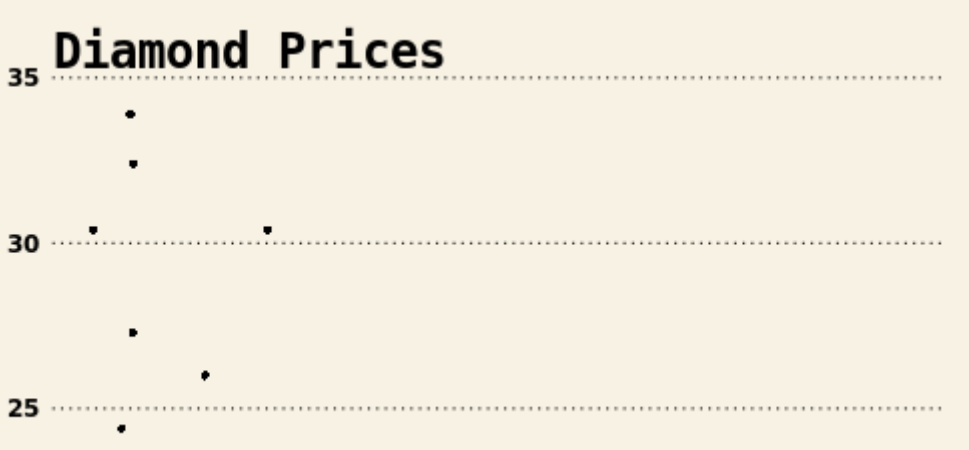
`ggthemes`

Overview | theme_base() | theme_excel_new()

ALL YOUR FIGURE ARE BELONG TO US

HOME › GGTHEMES

ggthemes

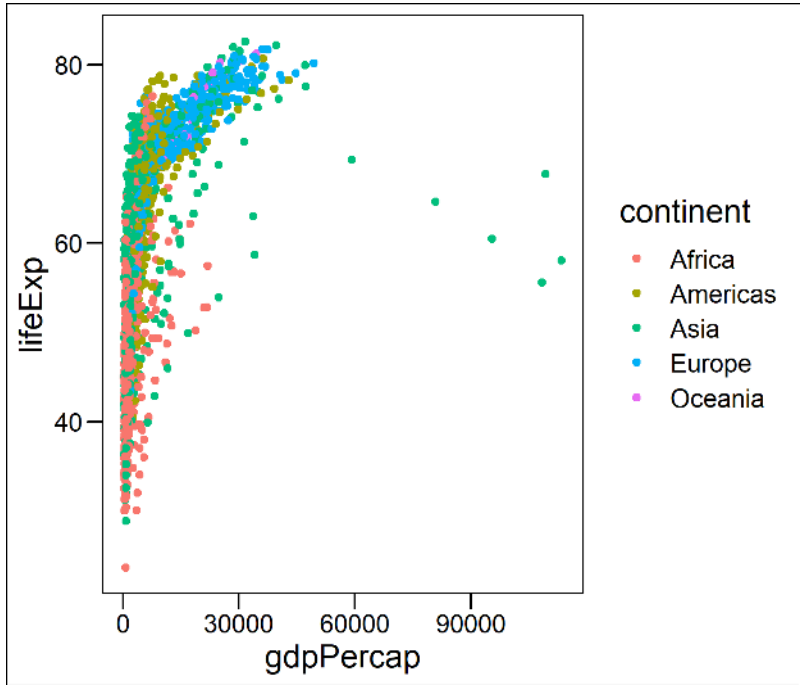


ggthemes

Overview | theme_base() | theme_excel_new()

> *Theme similar to the default settings of the 'base' R graphics. —* `?theme_base`

```
p + ggthemes::theme_base()
```

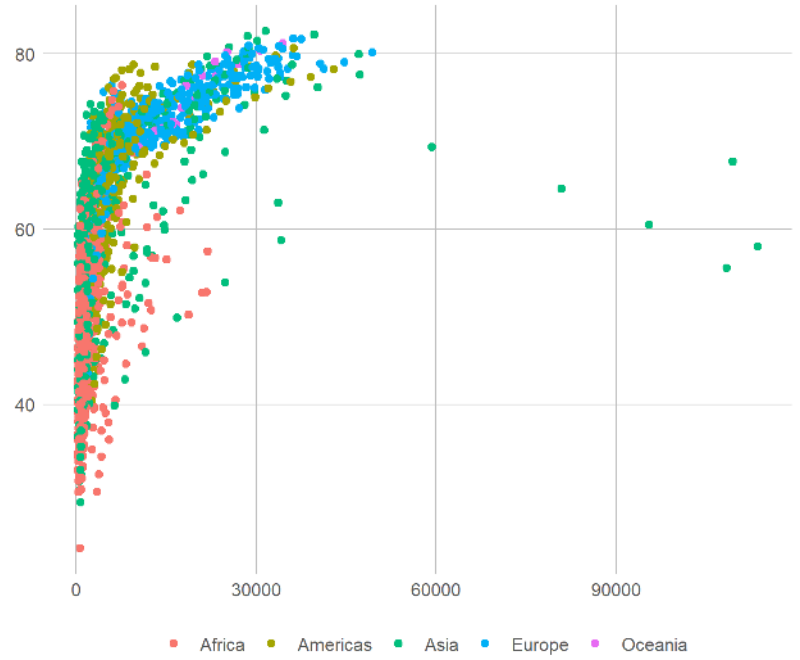


ggthemes

Overview | theme_base() | **theme_excel_new()**

> *Theme for ggplot2 that is similar to the default style of charts in current versions of Microsoft Excel. — `?theme_excel_new`*

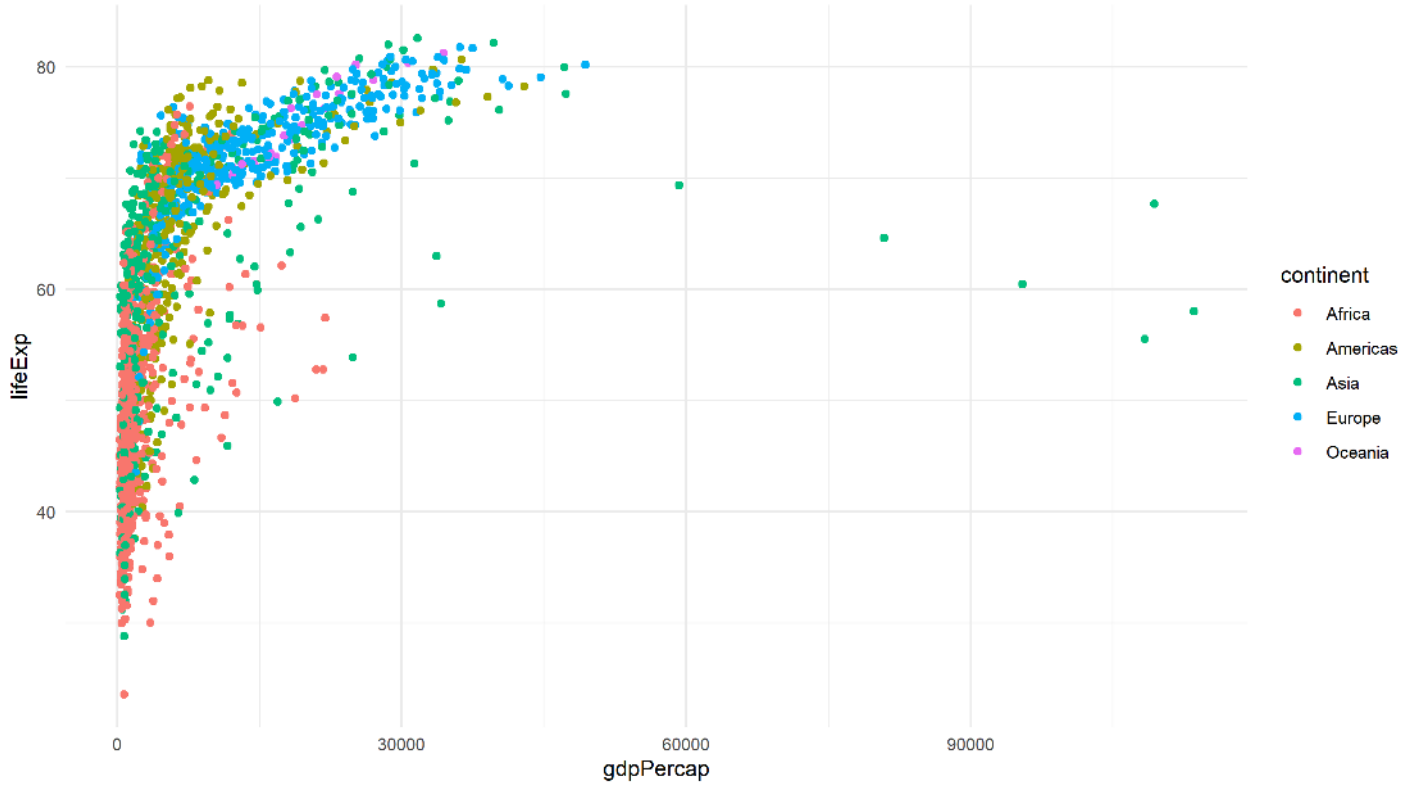
```
p + ggthemes::theme_excel_new()
```



Modifying base theme properties

default base settings | custom base settings

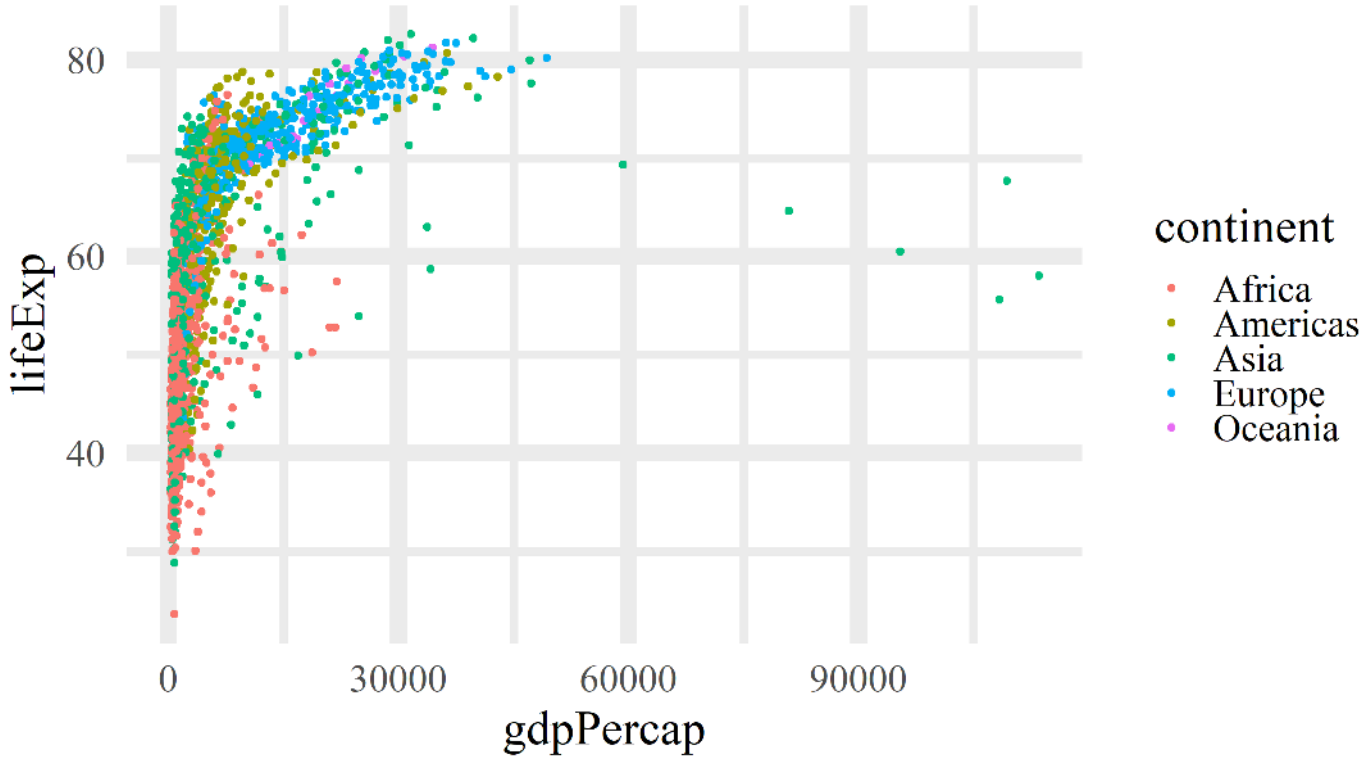
```
p + theme_minimal()
```



Modifying base theme properties

default base settings | custom base settings

```
p + theme_minimal(base_size = 24, base_family = "serif", base_line_size = 4)
```



Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | Lines | Borders & backgrounds | Remove elements | More | Get help

Make axes titles red and right-aligned.

```
p + theme_minimal(base_size = 24) +
  theme(axis.title = element_text(color = "red", hjust = 1))
```

Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | **Lines** | Borders & backgrounds | Remove elements | More | Get help

Add a y-axis line with arrow.

```
p + theme_minimal(base_size = 24) +
  theme(axis.line.y = element_line(arrow = arrow(type = "closed")))
```

Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | Lines | **Borders & backgrounds** | Remove elements | More | Get help

Make the legend box yellow and its border blue.

```
p + theme_minimal(base_size = 24) +
  theme(legend.background = element_rect(fill = "yellow", color = "blue"))
```

Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | Lines | Borders & backgrounds | **Remove elements** | More | Get help

Remove all grid lines.

```
p + theme_minimal(base_size = 24) +
  theme(panel.grid = element_blank())
```

Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | Lines | Borders & backgrounds | Remove elements | **More** | Get help

Put the legend above the plot.

```
p + theme_minimal(base_size = 24) +
  theme(legend.position = "top")
```

Modify indiv. theme elements: `theme(<ELEMENT> = ...)`

Text | Lines | Borders & backgrounds | Remove elements | More | **Get help**

`?theme` is your friend. 😎

theme {ggplot2} R Documentation

Modify components of a theme

Description

Themes are a powerful way to customize the non-data components of your plots: i.e. titles, labels, fonts, background, gridlines, and legends. Themes can be used to give plots a consistent customized look. Modify a single plot's theme using `theme()`; see `theme_update()` if you want modify the active theme, to affect all subsequent plots. Use the themes available in complete themes if you would like to use a complete theme such as `theme_bw()`, `theme_minimal()`, and more. Theme elements are documented together according to inheritance, read more about theme inheritance below.

Usage

```
theme(
  line,
  rect,
  text,
  title,
  aspect.ratio,
  axis.title,
  axis.title.x,
  axis.title.x.top,
  axis.title.x.bottom,
  axis.title.y,
  axis.title.y.left,
```

Save plots 💾

```
ggsave(
  filename = "filename.png", # or: pdf, svg, jpeg, eps, tiff, ...
  plot = p, # if not specified saves plot that was created last
  width = 8, height = 6,
  units = "cm",
  dpi = 300 # specifies resolution (dots per inch)
)
```

Visualizing numerical data

Number of variables involved

- **Univariate** data analysis: distribution of single variable
- **Bivariate** data analysis: relationship between two variables
- **Multivariate** data analysis: relationship between many variables at once, usually focusing on the relationship between two while conditioning for others

Types of variables

Type of variable		Description	Examples
Categorical/ Qualitative/	Nominal	Categories without inherent order.	Eye color, gender
	Ordinal	Categories with inherent order.	ratings $\{poor, fair, good\}$
Numerical/ Quantitative	Discrete	Only non-negative whole numbers.	Calendar dates, Bundesliga table standings
	Conti-nuous	A (theoretically) infinite number of values.	Temperature, monetary quantities, counts, age, mass, length

In this course, we use the terms *variable*, *attribute*, and *feature* synonymously.

IBM HR employee attrition & performance dataset

Artificial dataset from the IBM Watson Analytics Lab about factors that lead to employee attrition.

```
data(attrition, package = "modeldata")
attrition <- as_tibble(attrition)
glimpse(attrition)
```

```
## Rows: 1,470
## Columns: 31
## $ Age                      <int> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35, 29, 31, 34,~
## $ Attrition                <fct> Yes, No, Yes, No, No, No, No, No, No, No, No, No, No, N~
## $ BusinessTravel           <fct> Travel_Rarely, Travel_Frequently, Travel_Rarely, Travel~
## $ DailyRate                <int> 1102, 279, 1373, 1392, 591, 1005, 1324, 1358, 216, 1299~
## $ Department               <fct> Sales, Research_Development, Research_Development, Rese~
## $ DistanceFromHome         <int> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, 15, 26, 19, 24, 21~
## $ Education                <ord> College, Below_College, College, Master, Below_College,~
## $ EducationField           <fct> Life_Sciences, Life_Sciences, Other, Life_Sciences, Med~
## $ EnvironmentSatisfaction  <ord> Medium, High, Very_High, Very_High, Low, Very_High, Hig~
## $ Gender                   <fct> Female, Male, Male, Female, Male, Male, Female, Male, M~
## $ HourlyRate               <int> 94, 61, 92, 56, 40, 79, 81, 67, 44, 94, 84, 49, 31, 93,~
## $ JobInvolvement           <ord> High, Medium, Medium, High, High, High, Very_High, High~
## $ JobLevel                 <int> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, 1, 3, 1, 1, 4~
## $ JobRole                  <fct> Sales_Executive, Research_Scientist, Laboratory_Technic~
## $ JobSatisfaction          <ord> Very_High, Medium, High, High, Medium, Very_High, Low, ~
## $ MaritalStatus            <fct> Single, Married, Single, Married, Married, Single, Marr~
## $ MonthlyIncome            <int> 5993, 5130, 2090, 2909, 3468, 3068, 2670, 2693, 9526, 5~
## $ MonthlyRate              <int> 19479, 24907, 2396, 23159, 16632, 11864, 9964, 13335, 8~
## $ NumCompaniesWorked       <int> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1, 0, 5, 1, 0, 1, 2~
## $ OverTime                 <fct> Yes, No, Yes, Yes, No, No, Yes, No, No, No, No, Yes, No~
## $ PercentSalaryHike        <int> 11, 23, 15, 11, 12, 13, 20, 22, 21, 13, 13, 12, 17, 11,~
```

Variable types...

...for a selection of columns:

```
attrition %>%
  select(Age, Attrition, Gender, BusinessTravel, EducationField, JobLevel) %>%
  glimpse()
```

```
## Rows: 1,470
## Columns: 6
## $ Age            <int> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35, 29, 31, 34, 28, 29, 3~
## $ Attrition      <fct> Yes, No, Yes, No, No, No, No, No, No, No, No, No, No, No, Yes, No~
## $ Gender         <fct> Female, Male, Male, Female, Male, Male, Female, Male, Male, Male,~
## $ BusinessTravel <fct> Travel_Rarely, Travel_Frequently, Travel_Rarely, Travel_Frequentl~
## $ EducationField <fct> Life_Sciences, Life_Sciences, Other, Life_Sciences, Medical, Life~
## $ JobLevel       <int> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, 1, 3, 1, 1, 4, 1, 2, 1,~
```

Variable	Type
`Age`	numerical, discrete (here!)
`Attrition`	categorical, binominal
`Gender`	categorical, binominal (here!)
`BusinessTravel`	categorical, ordinal (frequently > rarely > non)
`EducationField`	categorical, multinominal (human resources, life sciences, marketing, etc.)
`JobLevel`	categorical, ordinal

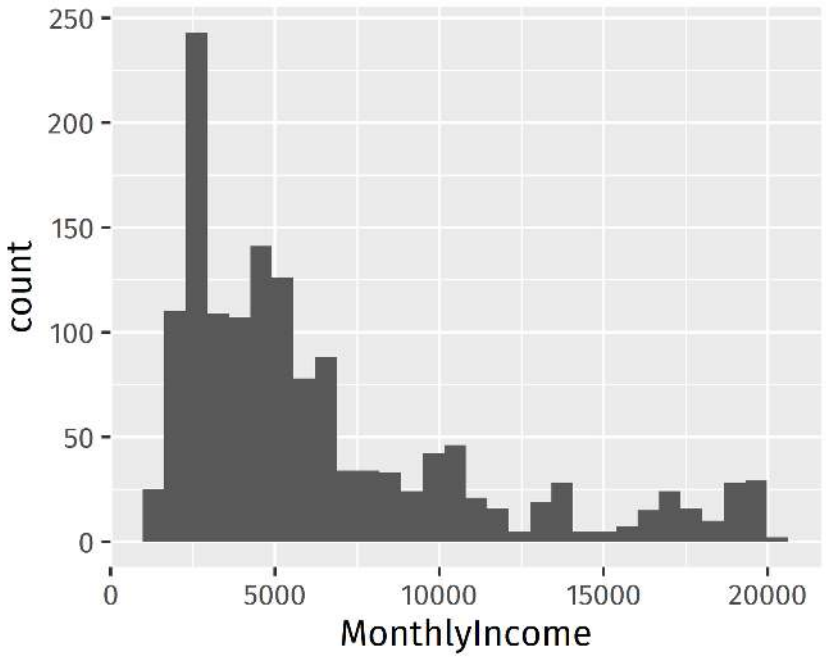
Describing shapes of numerical distributions

- **shape**:
  - **skewness**: left-skewed, right-skewed, symmetric
  - **modality**: unimodal, bimodal, multimodal, uniform
- **center**: mean (`mean()`), median (`median()`), mode (useful rather for categorical data)
- **spread**: range (`range()`), standard deviation (`sd()`), inter-quartile range (`IQR()`)
- unusual observations, i.e., **outliers**

Histogram

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram()
```

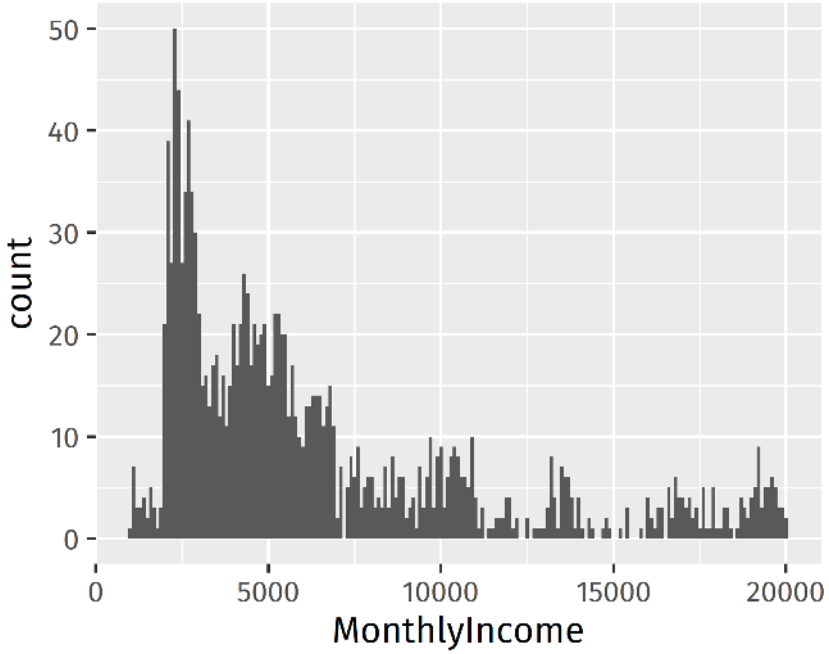
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Binwidth of histograms

binwidth = 100 | binwidth = 1000 | binwidth = 5000 | nbins = 15

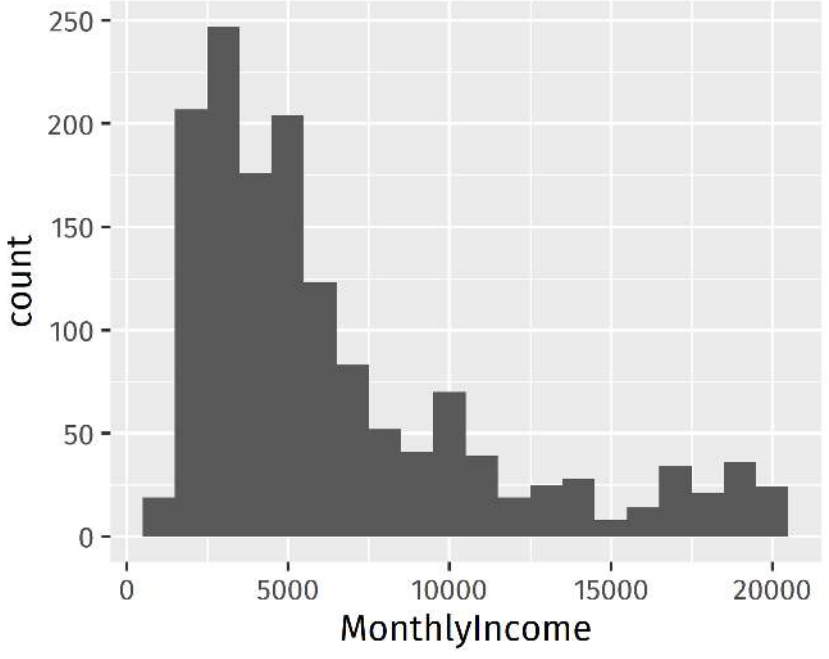
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram(binwidth = 100)
```



Binwidth of histograms

binwidth = 100 | **binwidth = 1000** | binwidth = 5000 | nbins = 15

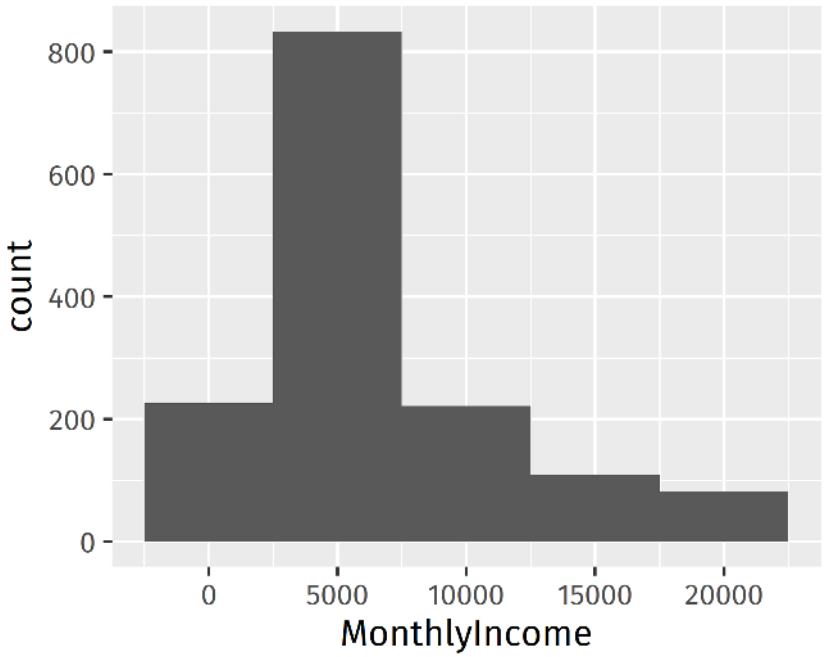
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram(binwidth = 1000)
```



Binwidth of histograms

binwidth = 100 | binwidth = 1000 | **binwidth = 5000** | nbins = 15

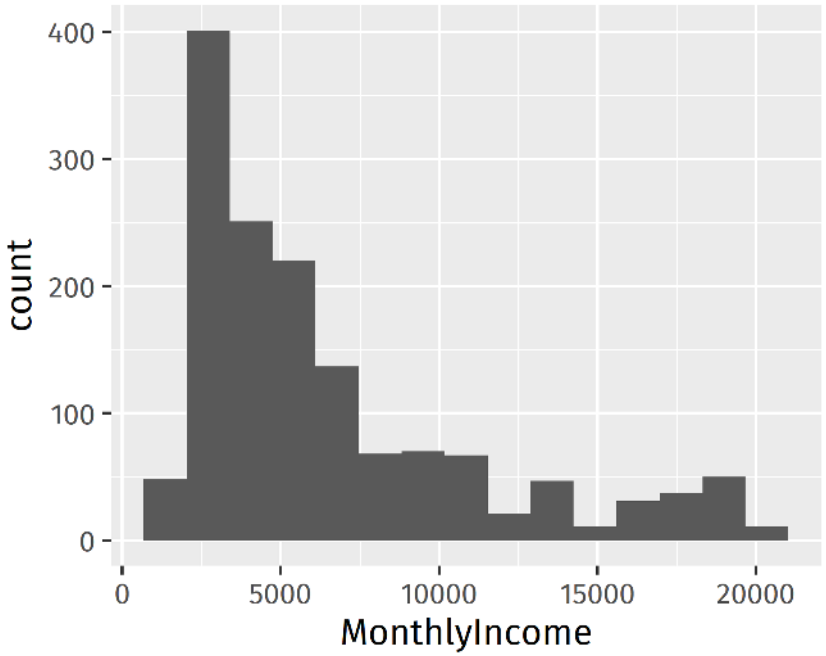
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram(binwidth = 5000)
```



Binwidth of histograms

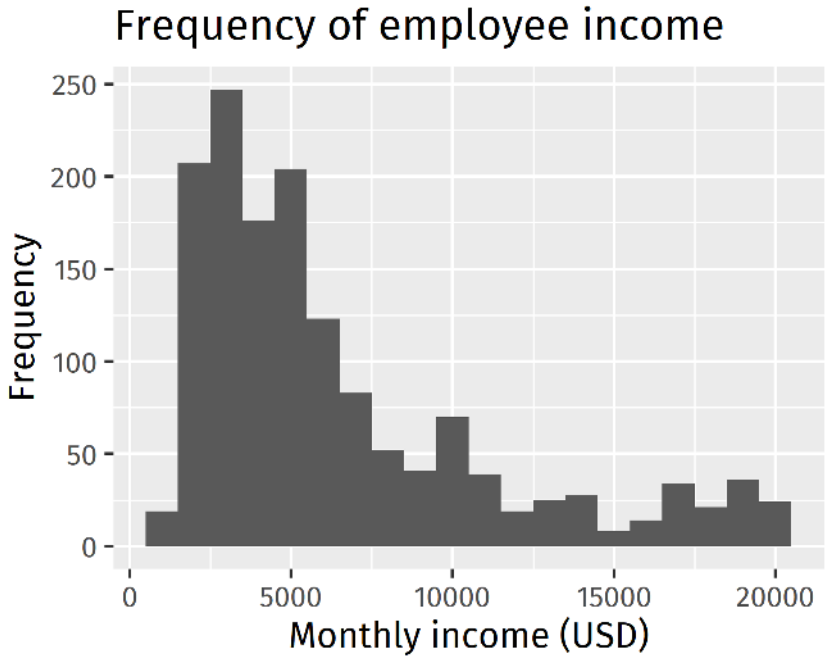
binwidth = 100 | binwidth = 1000 | binwidth = 5000 | **nbins = 15**

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram(bins = 15)
```



Customizing histograms

Plot | Code



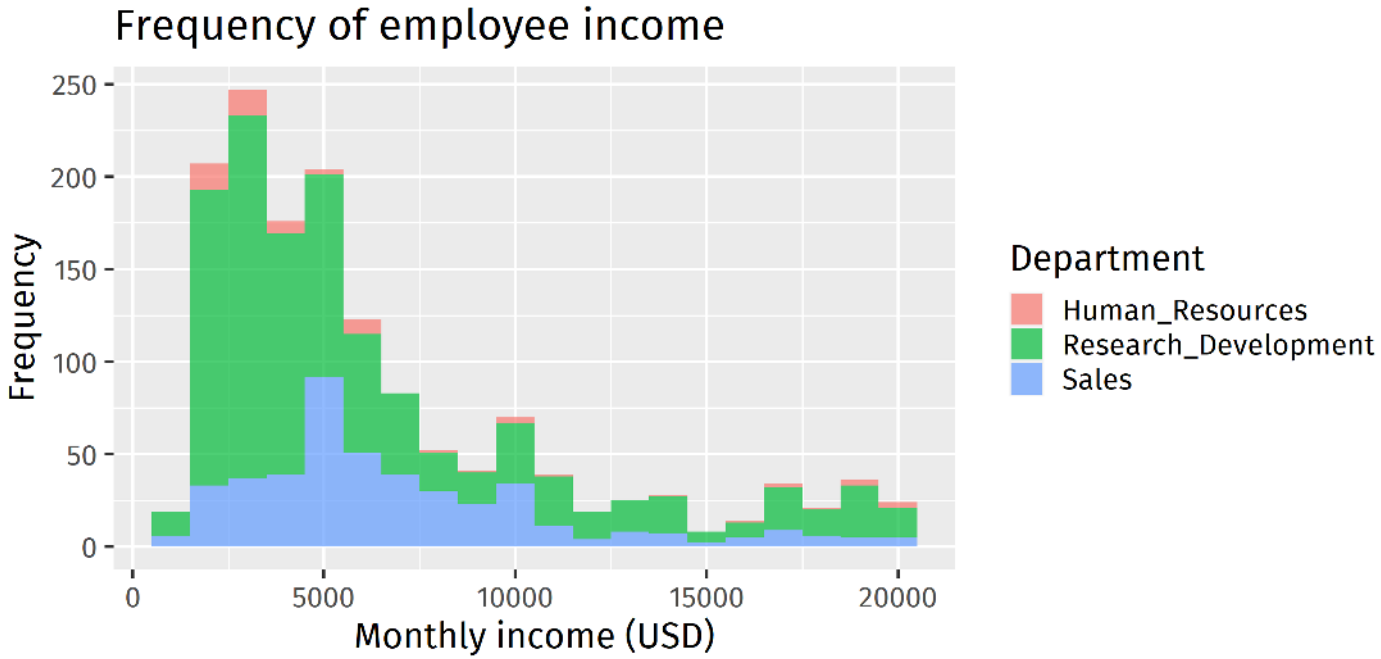
Customizing histograms

Plot | Code

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_histogram(binwidth = 1000) +
  labs(
    x = "Monthly income (USD)",
    y = "Frequency",
    title = "Frequency of employee income"
  )
```

Fill with a categorical variable

Plot | Code



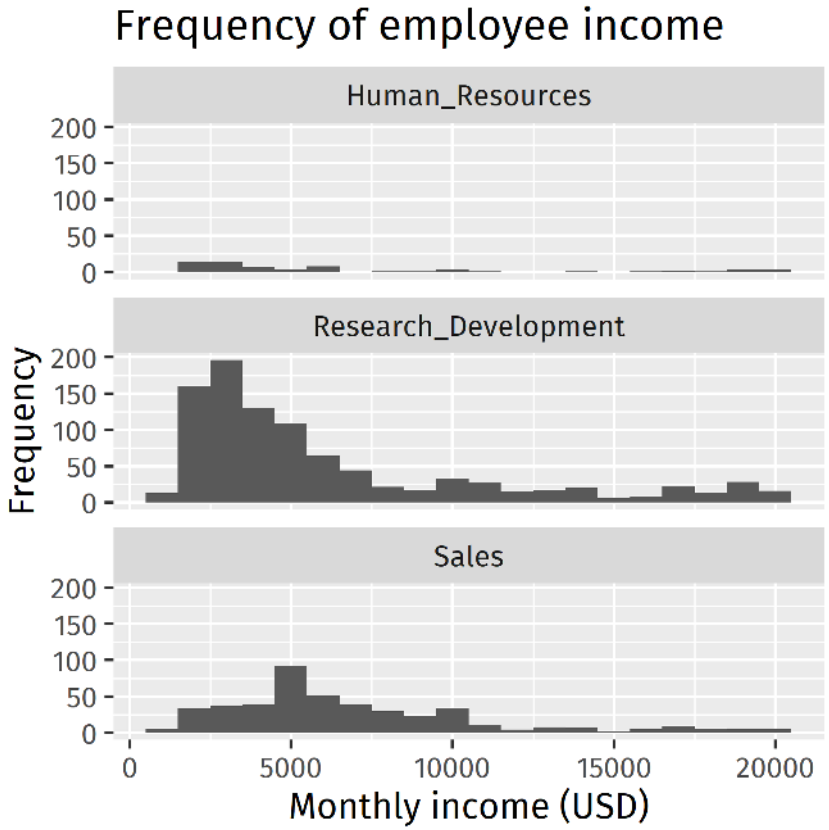
Fill with a categorical variable

Plot | Code

```
ggplot(
  attrition,
  aes(
    x = MonthlyIncome,
    fill = Department
  )
) +
  geom_histogram(
    binwidth = 1000,
    alpha = 0.7
  ) +
  labs(
    x = "Monthly income (USD)",
    y = "Frequency",
    title = "Frequency of employee income"
  )
```

Facet with a categorical variable

Plot | Code



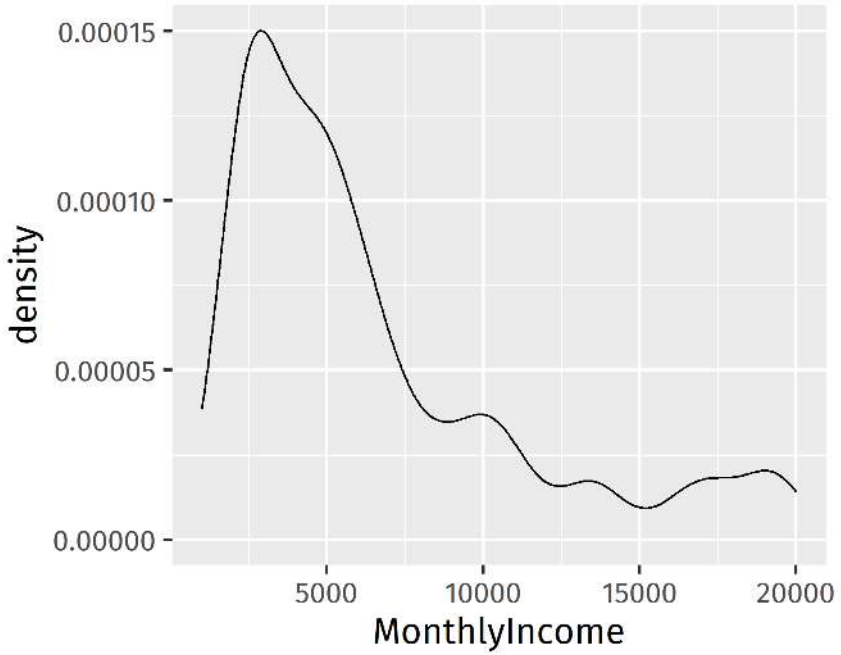
Facet with a categorical variable

Plot | Code

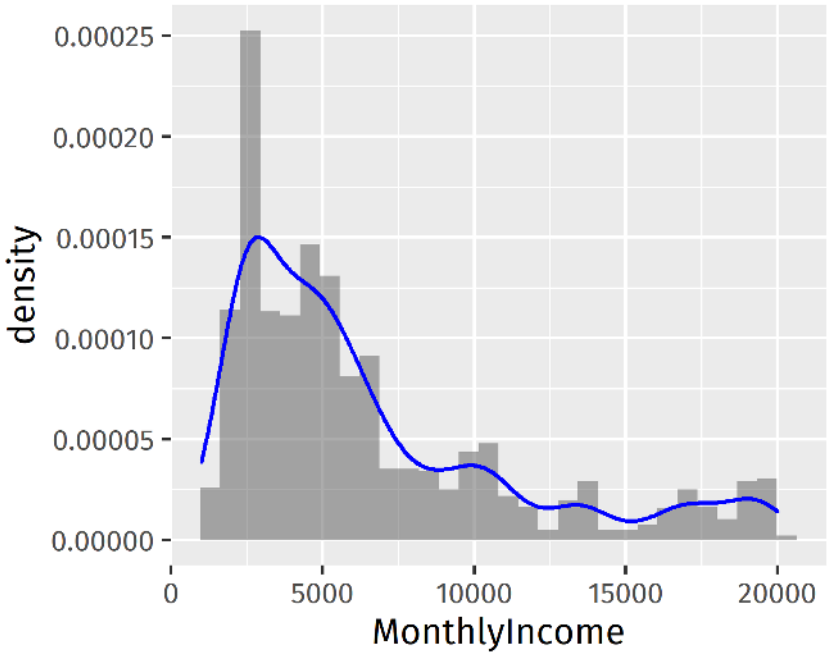
```
ggplot(
  attrition,
  aes(
    x = MonthlyIncome
  )
) +
  geom_histogram(
    binwidth = 1000
  ) +
  labs(
    x = "Monthly income (USD)",
    y = "Frequency",
    title = "Frequency of employee income"
  ) +
  facet_wrap(~ Department, nrow = 3)
```

Density plot

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_density()
```



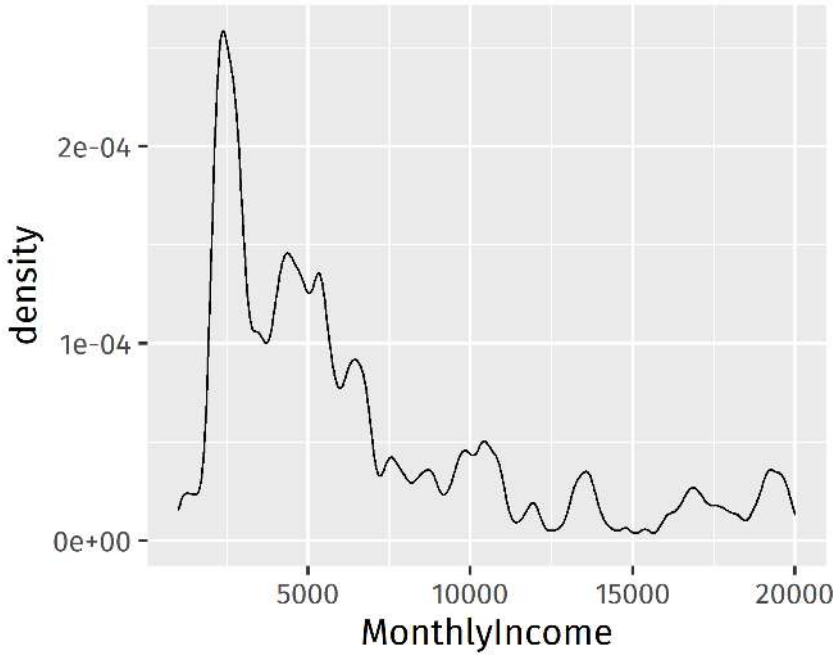
A density curve is like a smoothed representation of a histogram.



Adjusting the bandwith to control smoothness

adjust = 0.2 | adjust = 1 (default) | adjust = 2

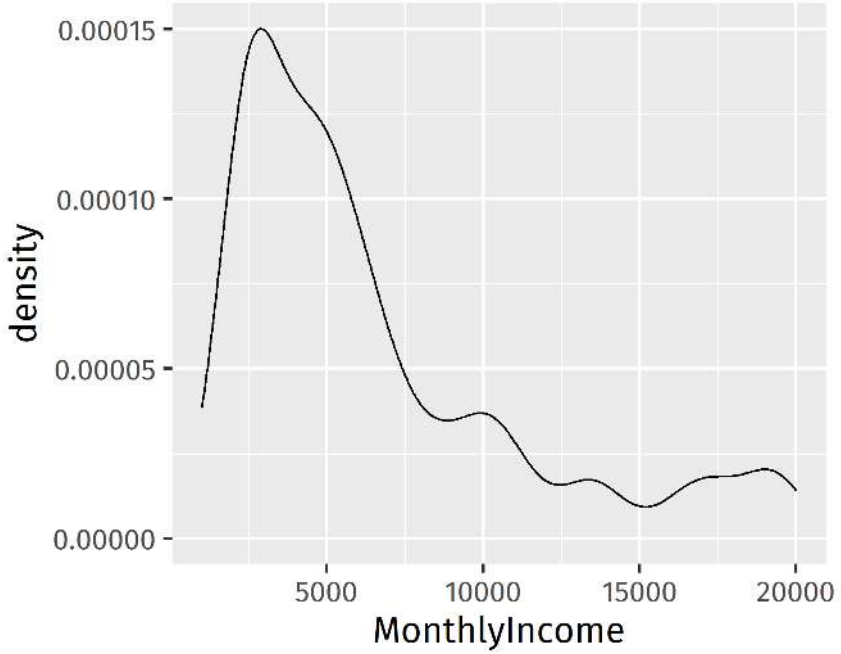
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_density(adjust = 0.2)
```



Adjusting the bandwith to control smoothness

adjust = 0.2 | **adjust = 1 (default)** | adjust = 2

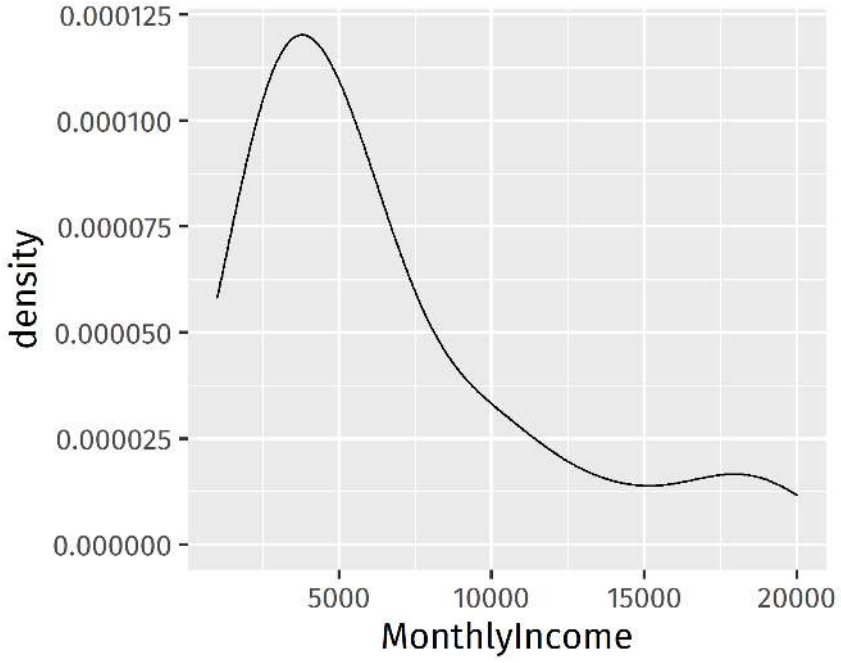
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_density(adjust = 1)
```



Adjusting the bandwith to control smoothness

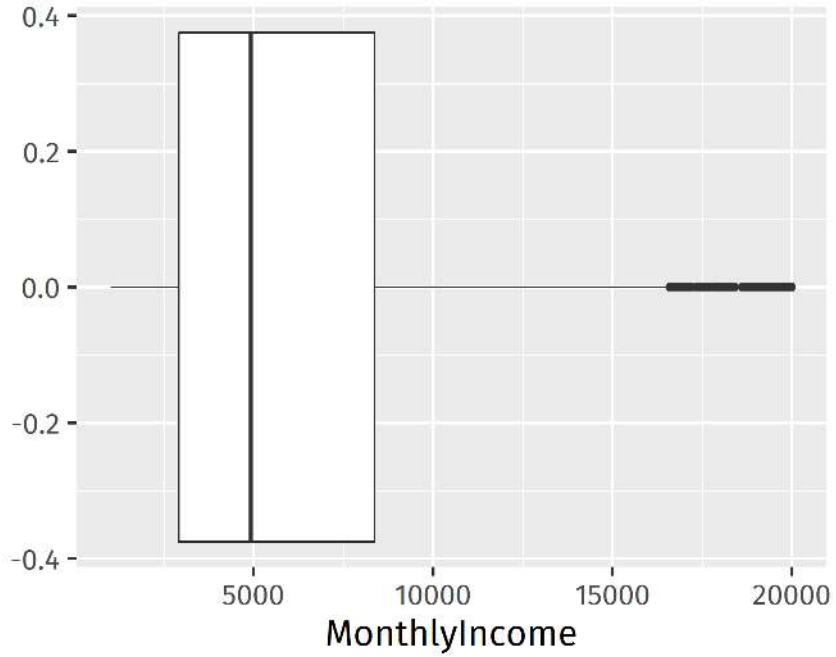
adjust = 0.2 | adjust = 1 (default) | adjust = 2

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_density(adjust = 2)
```



Boxplot

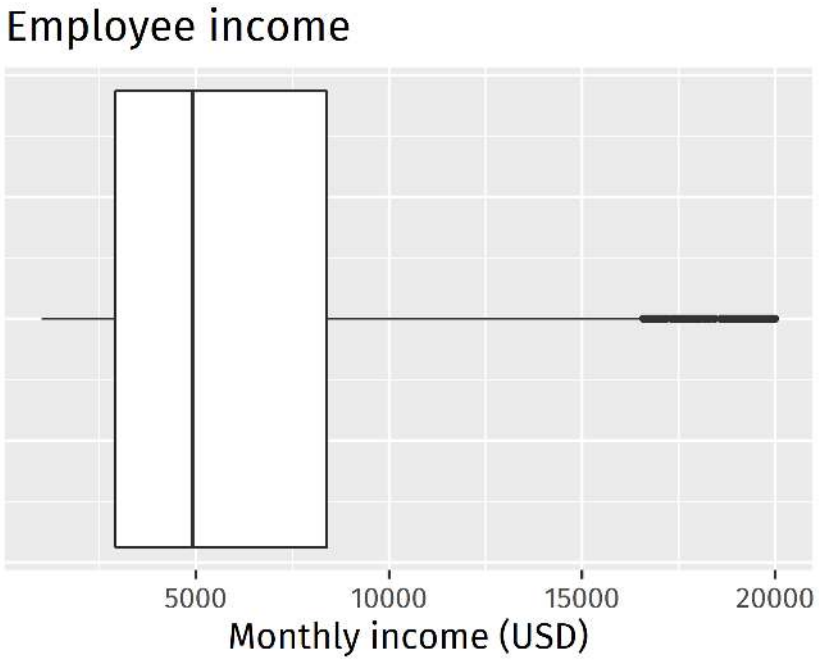
```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_boxplot()
```



The text on the y-axis isn't informative at all. Let's remove it.

Customizing boxplots

Plot | Code



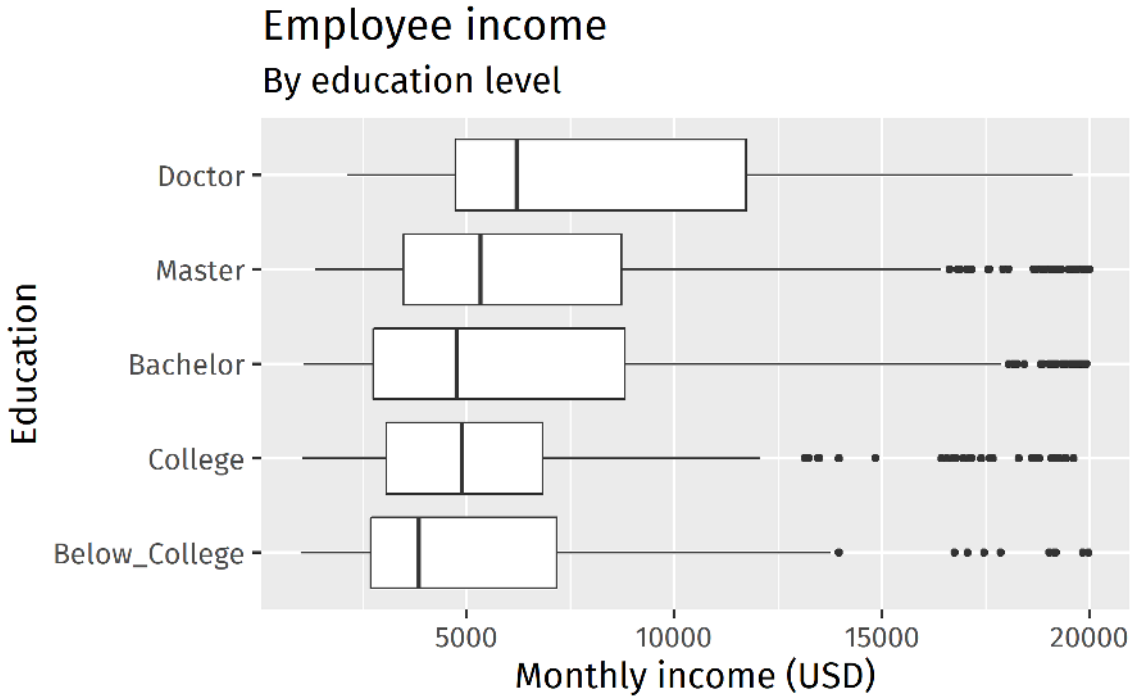
Customizing boxplots

Plot | Code

```
ggplot(attrition, aes(x = MonthlyIncome)) +
  geom_boxplot() +
  labs(
    x = "Monthly income (USD)",
    y = NULL,
    title = "Employee income"
  ) +
  theme(axis.text.y = element_blank()) +
  theme(axis.ticks.y = element_blank())
```

Adding a categorical variable

Plot | Code



Adding a categorical variable

Plot | Code

```
ggplot(attrition, aes(
  x = MonthlyIncome,
  y = Education
)
) +
  geom_boxplot() +
  labs(
    x = "Monthly income (USD)",
    y = "Education",
    title = "Employee income",
    subtitle = "By education level"
  )
```

Violin plots

Plot | Code

Violin plots

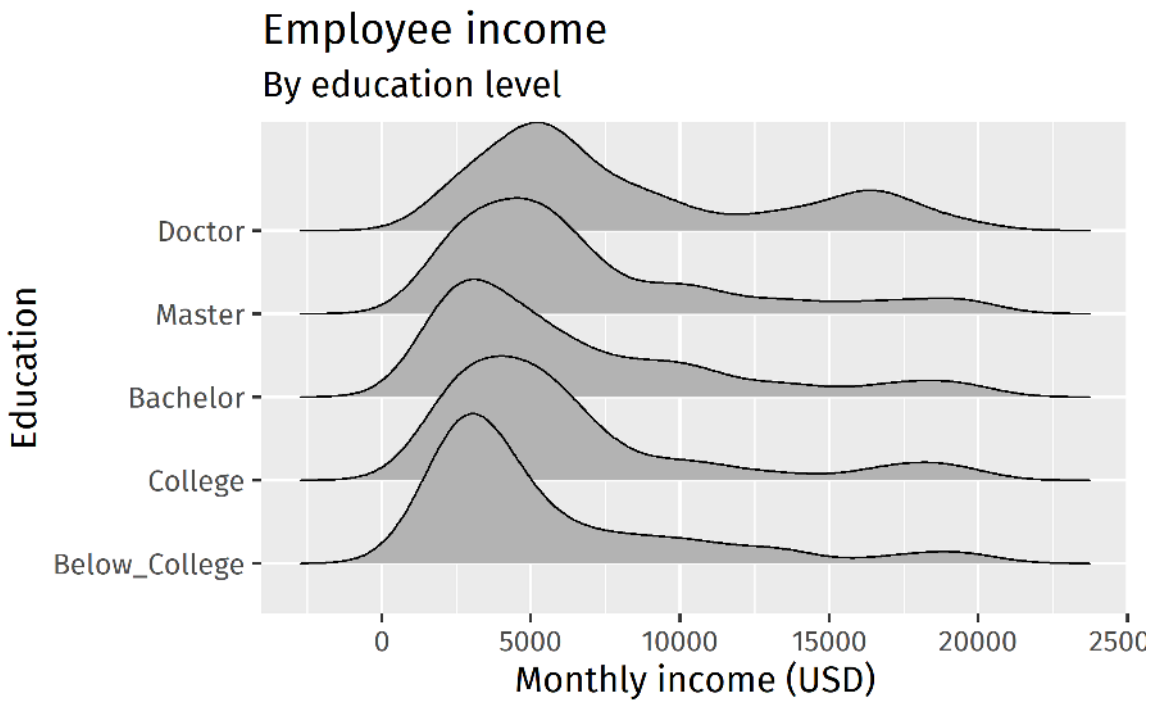
Plot | Code

```
ggplot(attrition, aes(
  x = MonthlyIncome,
  y = Education
)
) +
  geom_violin() +
  labs(
    x = "Monthly income (USD)",
    y = "Education",
    title = "Employee income",
    subtitle = "By education level"
  )
```

Ridgeline plots

Plot | Code

```
## Picking joint bandwidth of 1240
```



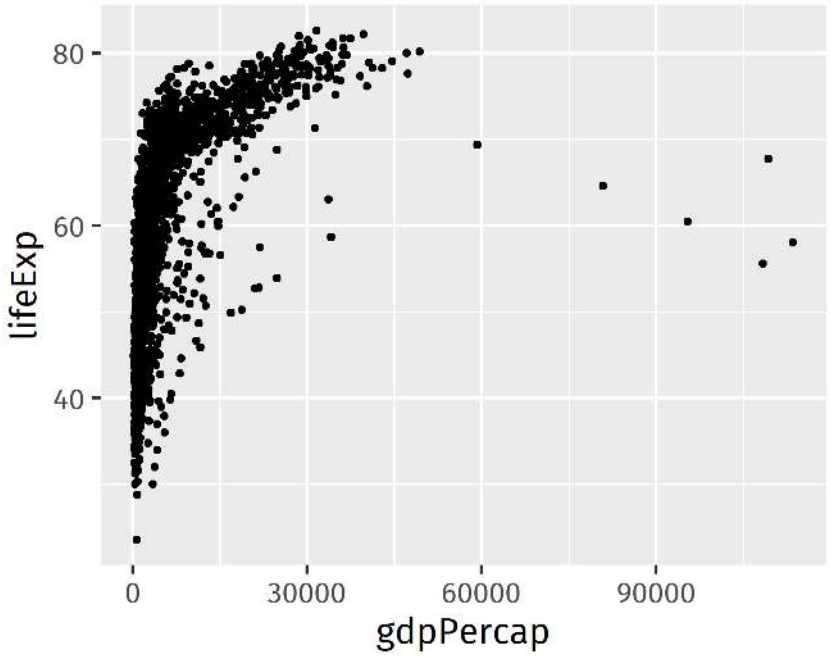
Ridgeline plots

Plot | Code

```
ggplot(attrition, aes(
  x = MonthlyIncome,
  y = Education
)
) +
  ggridges::geom_density_ridges() +
  labs(
    x = "Monthly income (USD)",
    y = "Education",
    title = "Employee income",
    subtitle = "By education level"
  )
```

Scatterplot

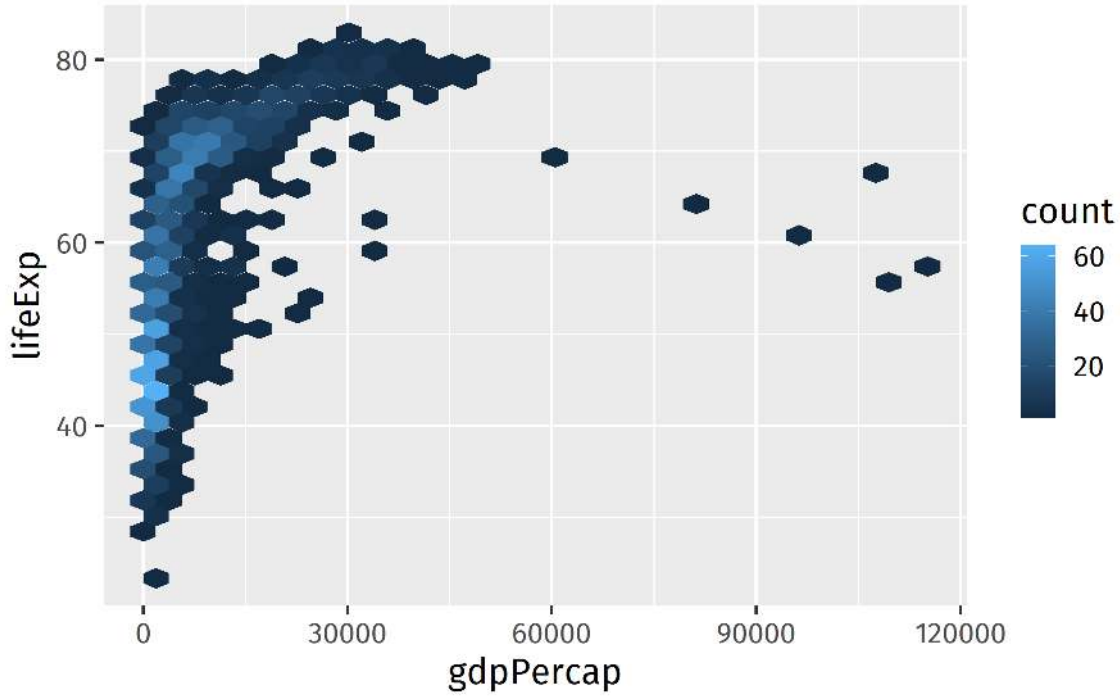
```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point()
```



There are a lot of overlapping points, which makes understanding of data density difficult.

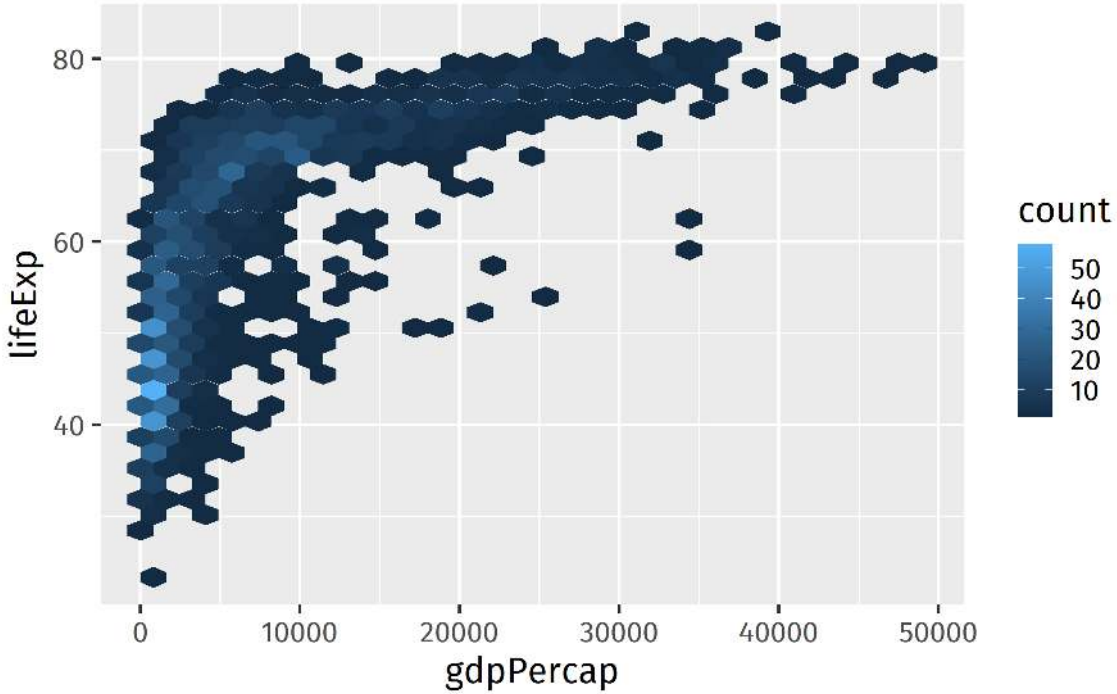
Hex plot

```
ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_hex()
```



Hex plot

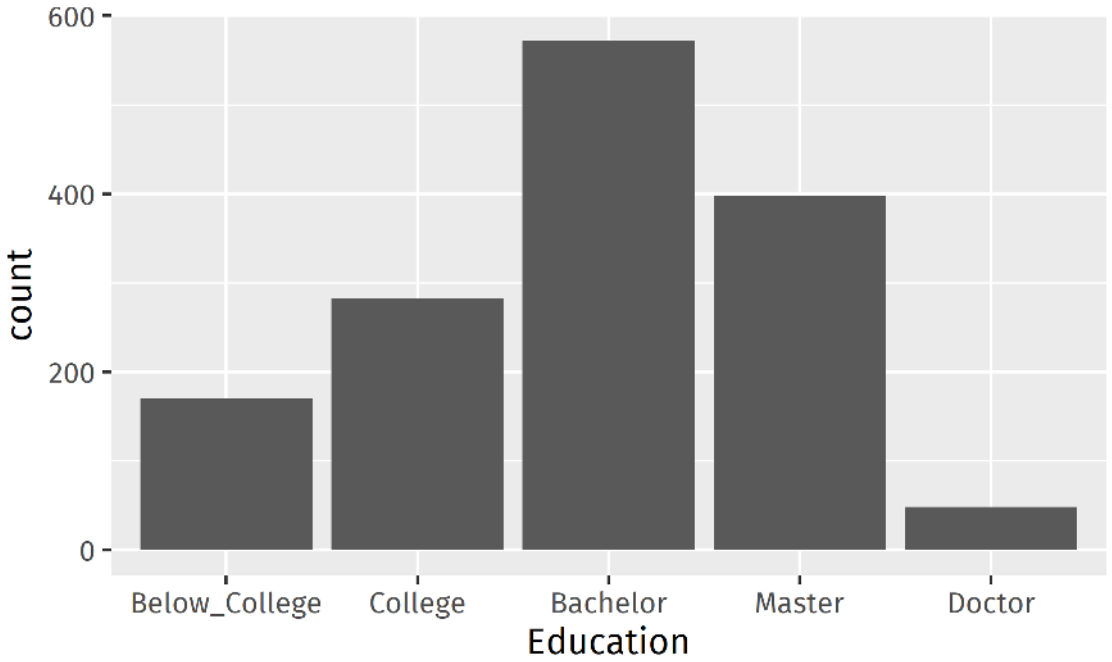
```
ggplot(gapminder %>% filter(gdpPercap < 50000),
       aes(x = gdpPercap, y = lifeExp)) +
  geom_hex()
```



Visualizing categorical data

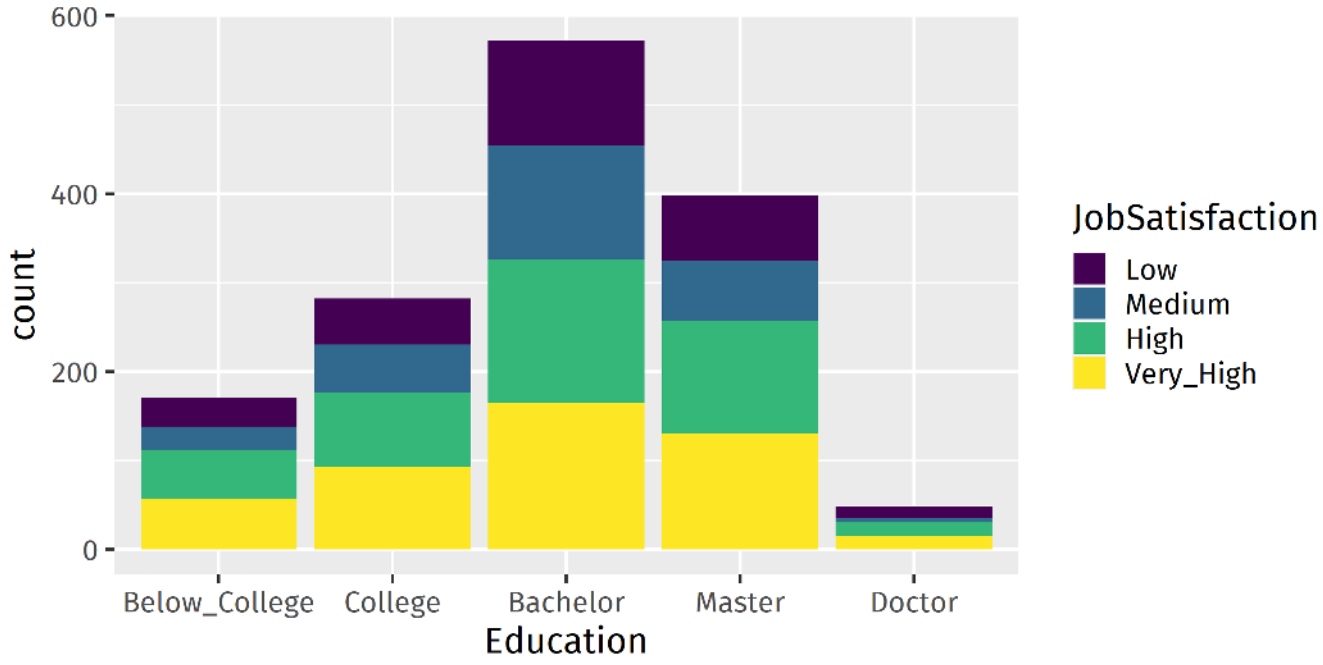
Bar plot

```
ggplot(attrition, aes(x = Education)) +
  geom_bar()
```



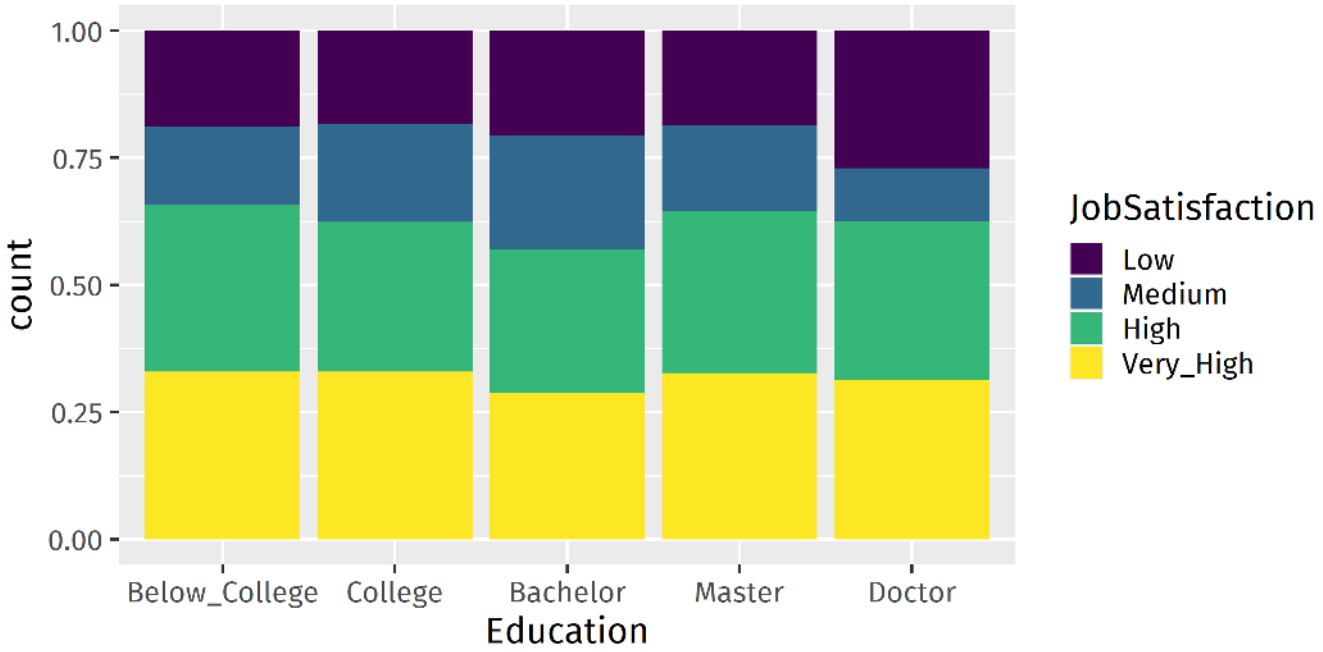
Segmented bar plot (absolute frequencies)

```
ggplot(attrition, aes(x = Education,
                      fill = JobSatisfaction)) +
  geom_bar()
```

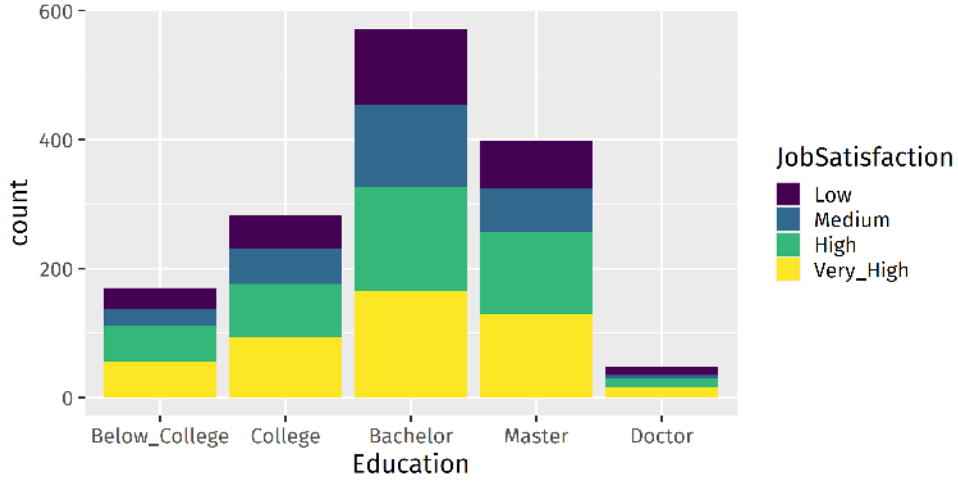


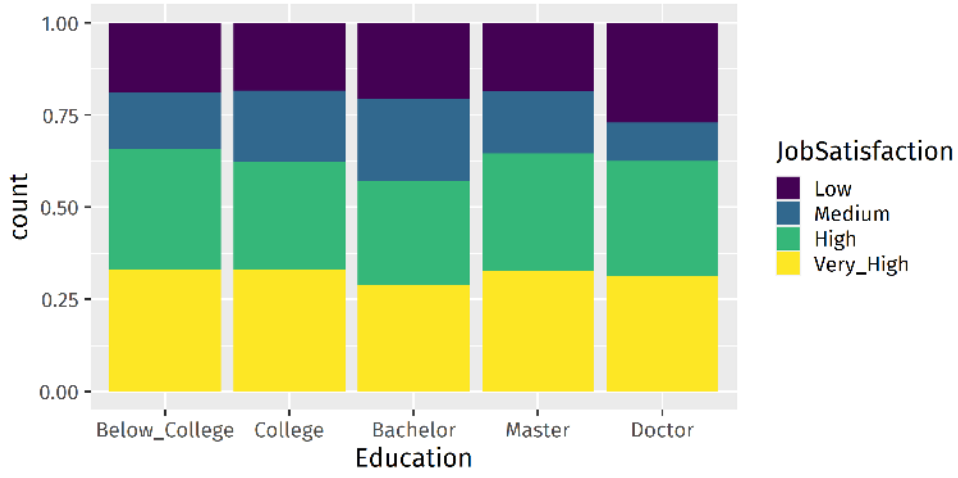
Segmented bar plot (relative frequencies)

```
ggplot(attrition, aes(x = Education, fill = JobSatisfaction)) +
  geom_bar(position = "fill")
```



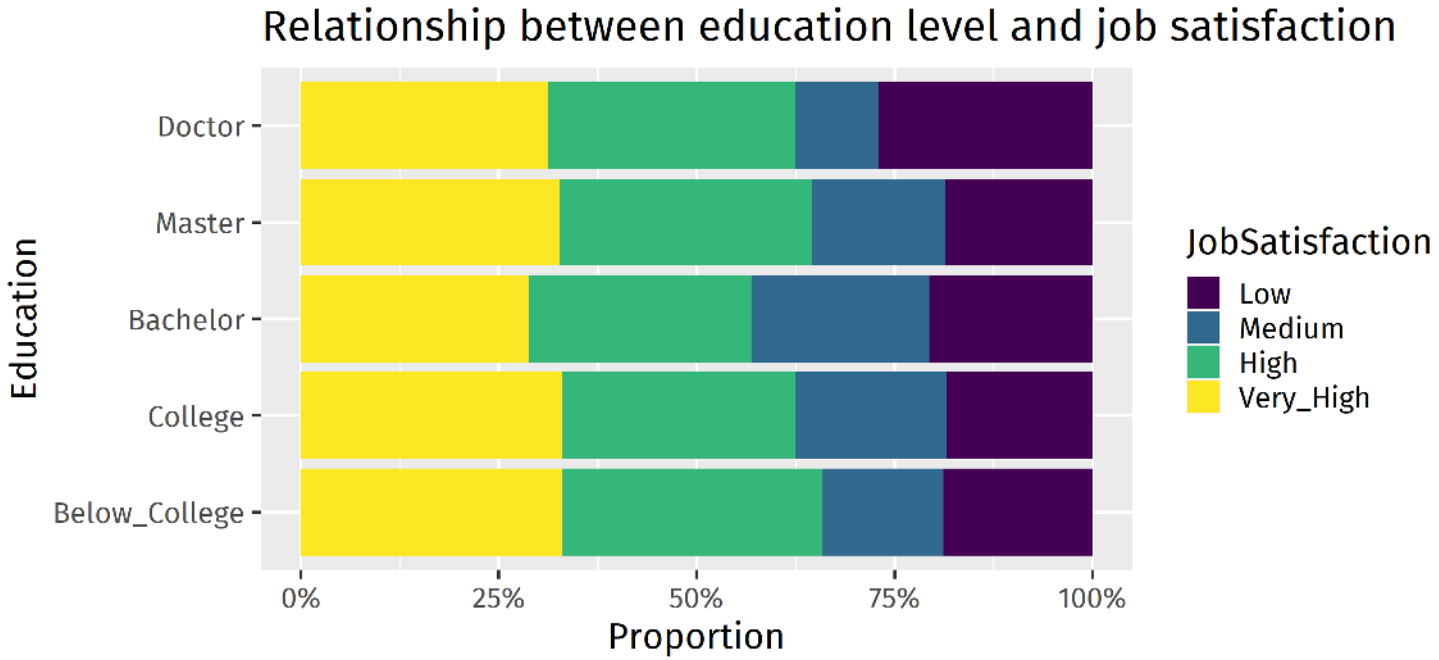
Which of the two bar plot variants is a more effective visualization for representing the relationship between education and job satisfaction?





Customizing bar plots

Plot | Code



Customizing bar plots

Plot | Code

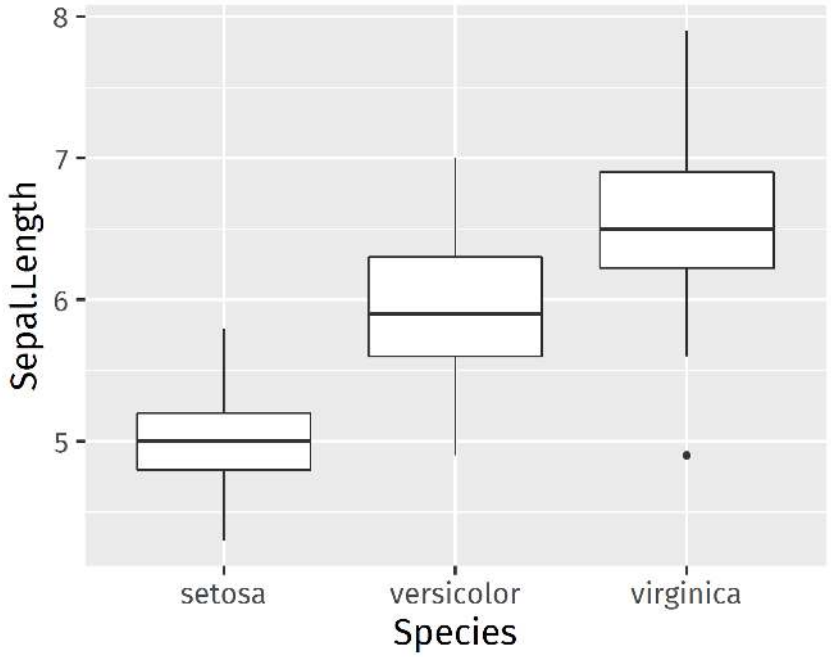
```
ggplot(attrition, aes(y = Education, fill = JobSatisfaction)) +
  scale_x_continuous(labels = scales::percent) +
  geom_bar(position = "fill") +
  labs(
    x = "Proportion",
    y = "Education",
    title = "Relationship between education level and job satisfaction"
  )
```

Advanced ggplot2 & extensions

Extracting plot details

One of the main reasons why `ggplot2` is easy to use, is that it makes the required computations for a lot of geoms by itself. For example, the boxplot geom automatically calculates the values for the 5-point summary and identifies possible outliers.

```
p <- ggplot(iris, aes(Species, Sepal.Length)) +
   geom_boxplot()
p
```




```
class(p)
```

```
## [1] "gg"     "ggplot"
```

🤔 *"I need to depict the summary statistics of the boxplot for my final project report. How can I extract them?"*

Extracting plot details

Whenever a ggplot object is "printed" to the screen, the function `ggplot_build()` is invoked internally to render the plot.

```
gr <- ggplot_build(p) # execute all necessary steps to render the plot
class(gr)
```

```
## [1] "ggplot_built"
```

```
names(gr)
```

```
## [1] "data"   "layout" "plot"
```

The three components of a `ggplot_built` object are:

- `data`: details for each plot layer, e.g. the 5-point summary of a boxplot.
- `layout`: axis information, e.g. breaks, ranges and labels
- `plot`: the rendered plot itself

Extracting plot details

We are interested in the 5-point summary of the 3 boxplots, which were automatically calculated by `ggplot2`. We extract the information from the `data` element of `gr`.

```
gr$data[[1]] # or: layer_data(p, i = 1)
```

```
##   ymin lower middle upper ymax outliers notchupper notchlower x flipped_aes PANEL group
## 1  4.3 4.800    5.0   5.2  5.8            5.089378   4.910622 1       FALSE     1     1
## 2  4.9 5.600    5.9   6.3  7.0            6.056412   5.743588 2       FALSE     1     2
## 3  5.6 6.225    6.5   6.9  7.9      4.9   6.650826   6.349174 3       FALSE     1     3
##   ymin_final ymax_final  xmin  xmax xid newx new_width weight colour  fill size alpha
## 1        4.3        5.8 0.625 1.375   1    1      0.75      1 grey20 white  0.5    NA
## 2        4.9        7.0 1.625 2.375   2    2      0.75      1 grey20 white  0.5    NA
## 3        4.9        7.9 2.625 3.375   3    3      0.75      1 grey20 white  0.5    NA
##   shape linetype
## 1    19    solid
## 2    19    solid
## 3    19    solid
```

Here, each row contains data for one of the three boxes. The first five columns are:

- `ymin`: lower end of lower whisker (= median - 1.5 * IQR)
- `lower`: lower end of box (= first quartile)
- `middle`: horizontal line within box (= median)
- `upper`: upper end of box (= third quartile)
- `ymax`: upper end of upper whisker (= median + 1.5 * IQR)

Maps

Example **choropleth maps** showing the poll results of the 2016 United States Presidential Elections:

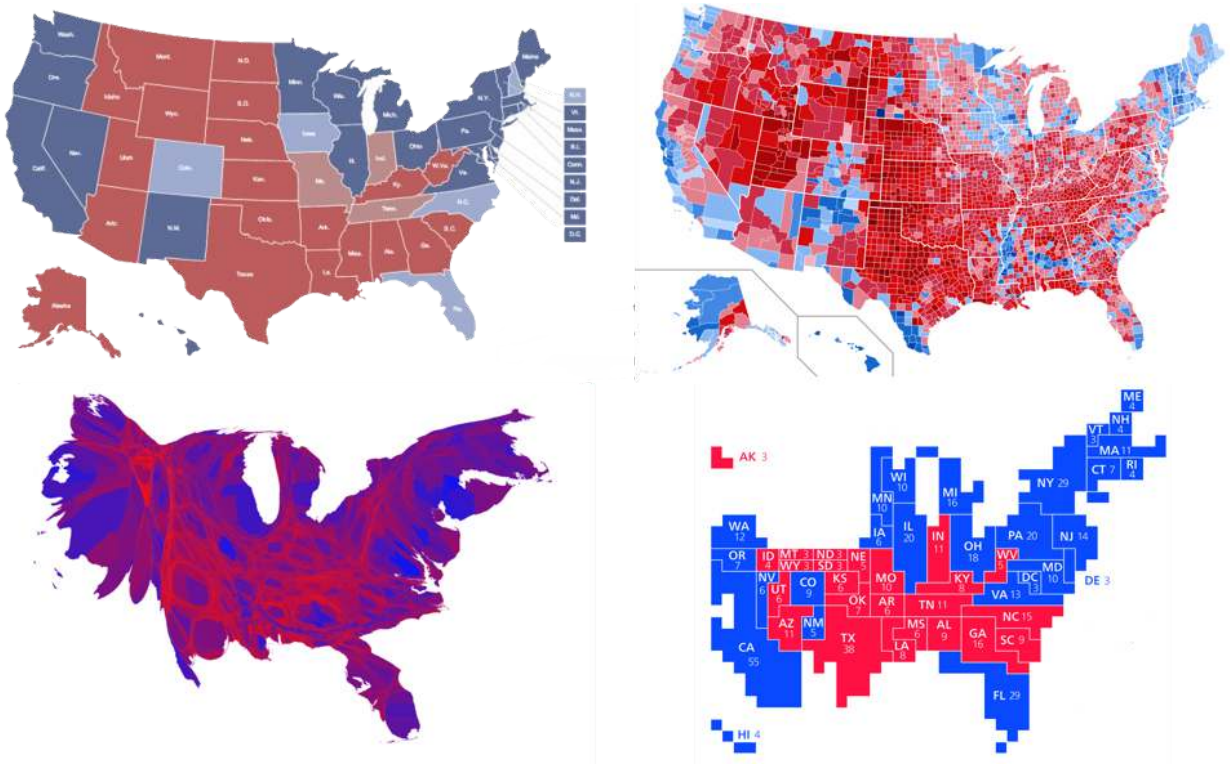


Figure source: Kieran Healy. "Data Visualization. A practical introduction". Princeton University Press, 2018.

Maps

Draw a map of the USA:

```
usa <- map_data("state")
str(usa)
```

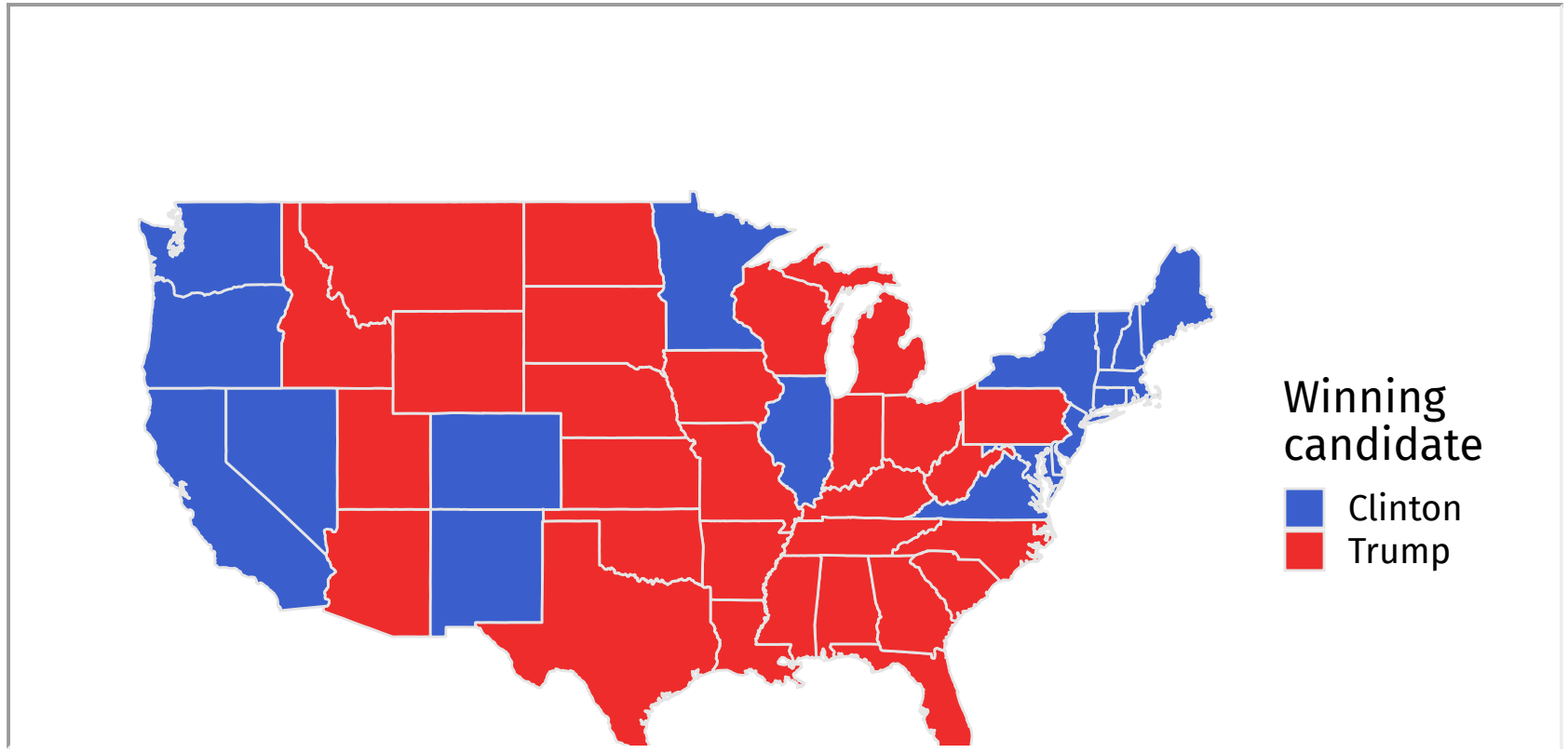
```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long     : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat      : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "alabama" "alabama" "alabama" "alabama" ...
##  $ subregion: chr  NA NA NA NA ...
```

```
ggplot(usa, aes(x = long, y = lat,
                group = group)) +
  geom_polygon(color = "black", fill = NA) +
  coord_map() +
  theme_void()
```

Interactive graphs with ggiraph

An interactive map that shows the 2016 US presidential election results. Hovering over a state lets a tooltip pop up, showing the percentages of each candidate of the Democratic and Republican parties.

Plot Code



Interactive graphs with ggiraph

An interactive map that shows the 2016 US presidential election results. Hovering over a state lets a tooltip pop up, showing the percentages of each candidate of the Democratic and Republican parties.

Plot | Code

```
library(ggiraph)
p <- usa %>% rename(state = region) %>%
  mutate(state = stringr::str_to_title(state)) %>%
  mutate(state = if_else(state == "District Of Columbia", "District of Columbia", state)) %>%
  left_join(socviz::election %>% select(state, winner, pct_clinton, pct_trump), by = "state") %>%
  mutate(tooltip = paste0(winner, " won ", state, "\nClinton: ", pct_clinton, "%\nTrump: ", pct_trump, "
  ggplot(aes(long, lat, group = group)) +
  geom_polygon_interactive(aes(fill = winner, data_id = state, tooltip = tooltip), color = "gray90") +
  scale_fill_manual(values = c("royalblue3", "firebrick2")) +
  labs(fill = "Winning\ncandidate") +
  coord_map() +
  theme_void(base_family = "Fira Sans", base_size = 18)
girafe(ggobj = p)
```

Draw maps from shape files

The `maps` package contains map data only for a handful of countries, including USA, France, Italy and New Zealand, as well as 2 world maps.

Generally, **shapefiles** are more flexible in accessing geographic and political boundaries than built-in maps. A shapefile is **geospatial vector data format** for geographic information system (GIS) software.

Shapefiles actually consist of several sub-files, see e.g. Wikipedia.

Extensions & Alternatives

https://exts.ggplot2.tidyverse.org/gallery

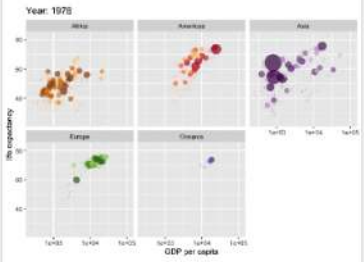
ggplot2 extensions - gallery

Add Your Extension! ggplot2-exts.org

53 registered extensions available to explore

Sort: Github stars | Text Filter | Author Filter | Tag Filter | CRAN Only

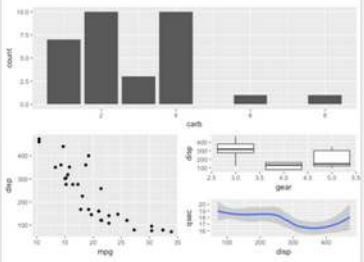
Showing 53 of 53



gganimate ★ Star 1235

A Grammar of Animated Graphics.

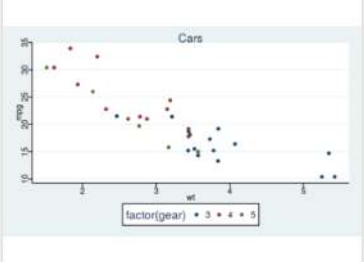
- author: thomasp85
- tags: visualization, general
- js libraries:



patchwork ★ Star 1051

Easy composition of ggplot plots using arithmetic operators

- author: thomasp85
- tags: visualization, composition
- js libraries:



ggthemes ★ Star 981

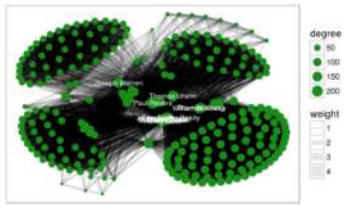
Some extra geoms, scales, and themes for ggplot.

- author: jrnold
- tags: visualization, general
- js libraries:

esquisse ★ Star 647

Explore and Visualize Your Data Interactively with `ggplot2`

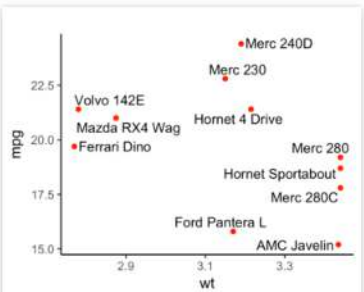
- author: dreamrs
- tags: visualization, interface
- js libraries:



ggraph ★ Star 633

ggraph is tailored at plotting graph-like data structures (graphs, networks, trees, hierarchies...).

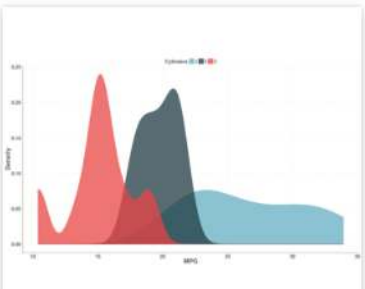
- author: thomasp85
- tags: visualization, general
- js libraries:



ggrepel ★ Star 620

Repel overlapping text labels away from each other.

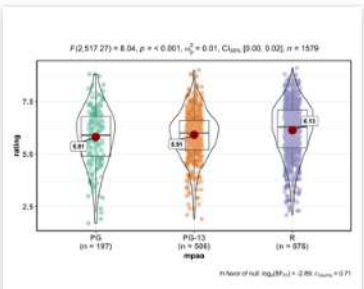
- author: slowkow
- tags: visualization, general
- js libraries:



ggthemr ★ Star 544

Themes for ggplot

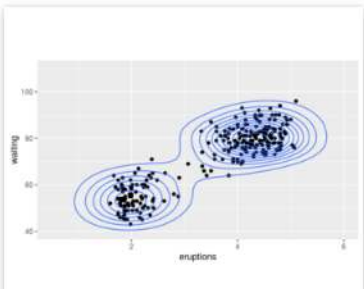
- author: cttobin
- tags: visualization, general
- js libraries:



ggstatsplot ★ Star 492

'ggstatsplot' provides a collection of functions to enhance 'ggplot2' plots with results from statistical tests.

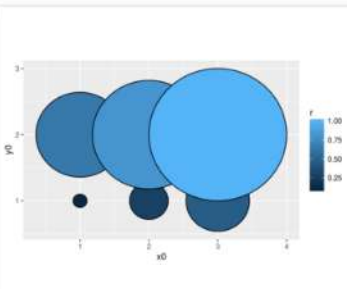
- author: IndrajeetPatil
- tags: visualization, statistics
- js libraries:



ggalt ★ Star 430

A compendium of 'geoms', 'coords' and 'stats' for 'ggplot2'.

- author: hrbrmstr
- tags: visualization, general
- js libraries:



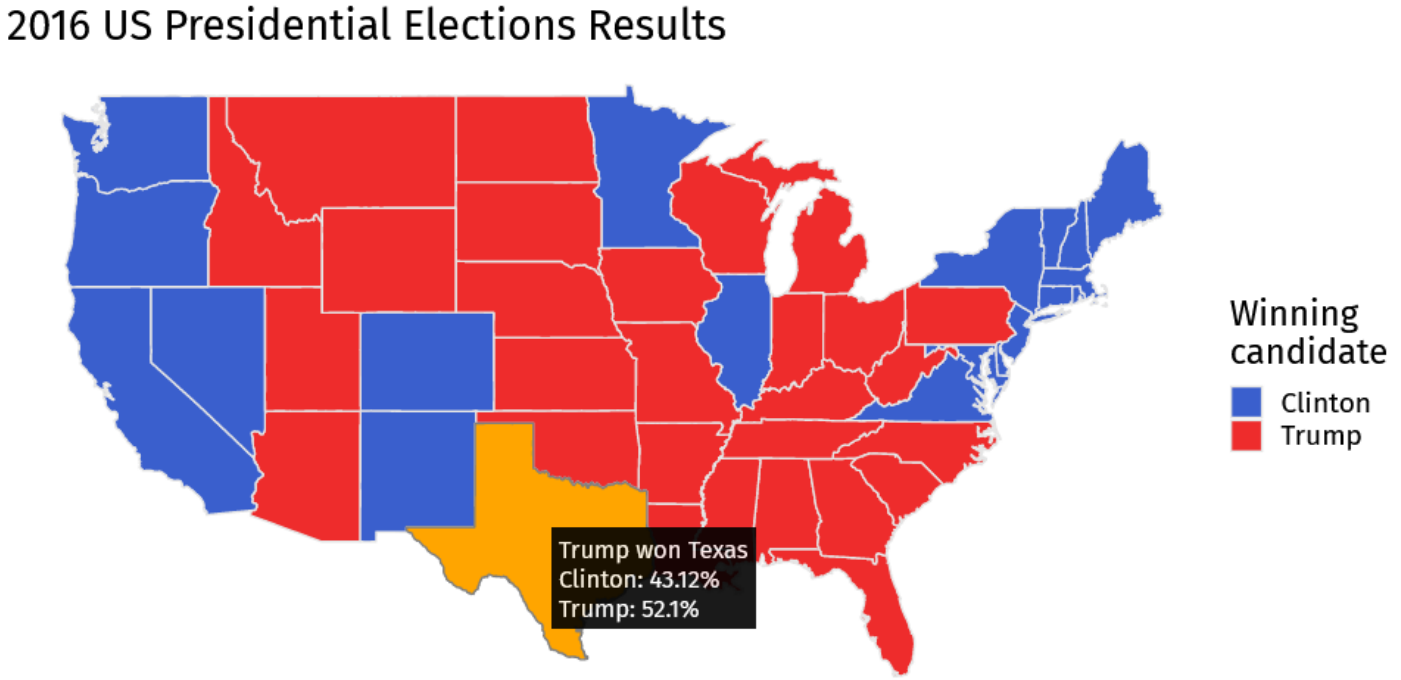
ggforce ★ Star 399

ggforce is aimed at providing missing functionality to ggplot2 through the extension system introduced with ggplot2 v2.0.0.

- author: thomasp85
- tags: visualization, general
- js libraries:

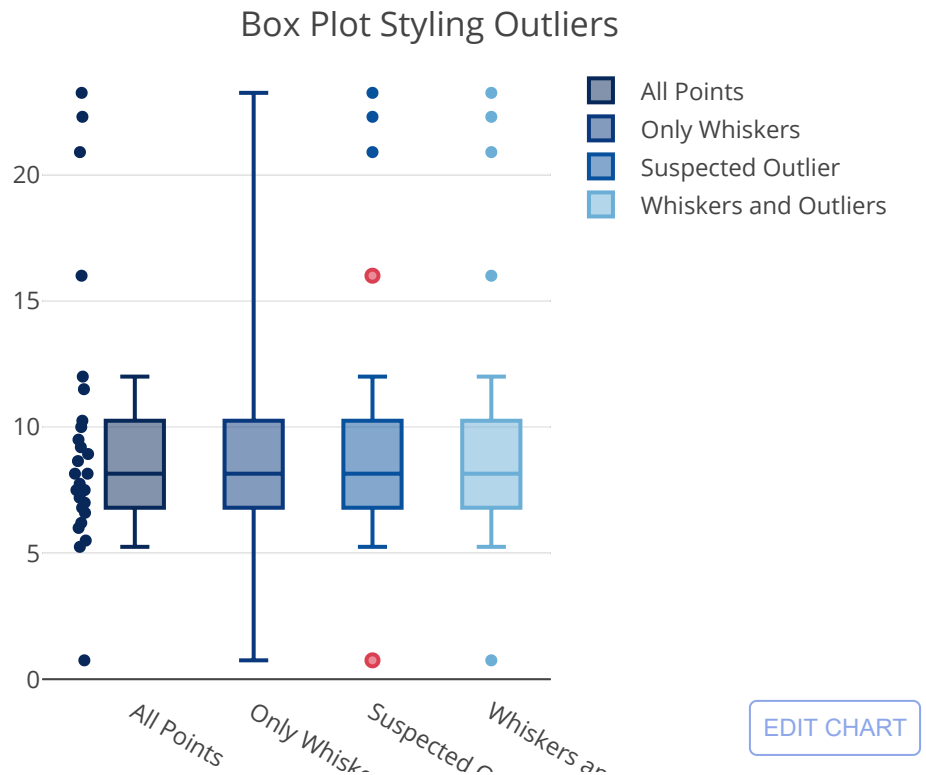
ggiraph

`ggiraph`: htmlwidget to extend `ggplot2` with `D3.js` to generate **animated** graphs



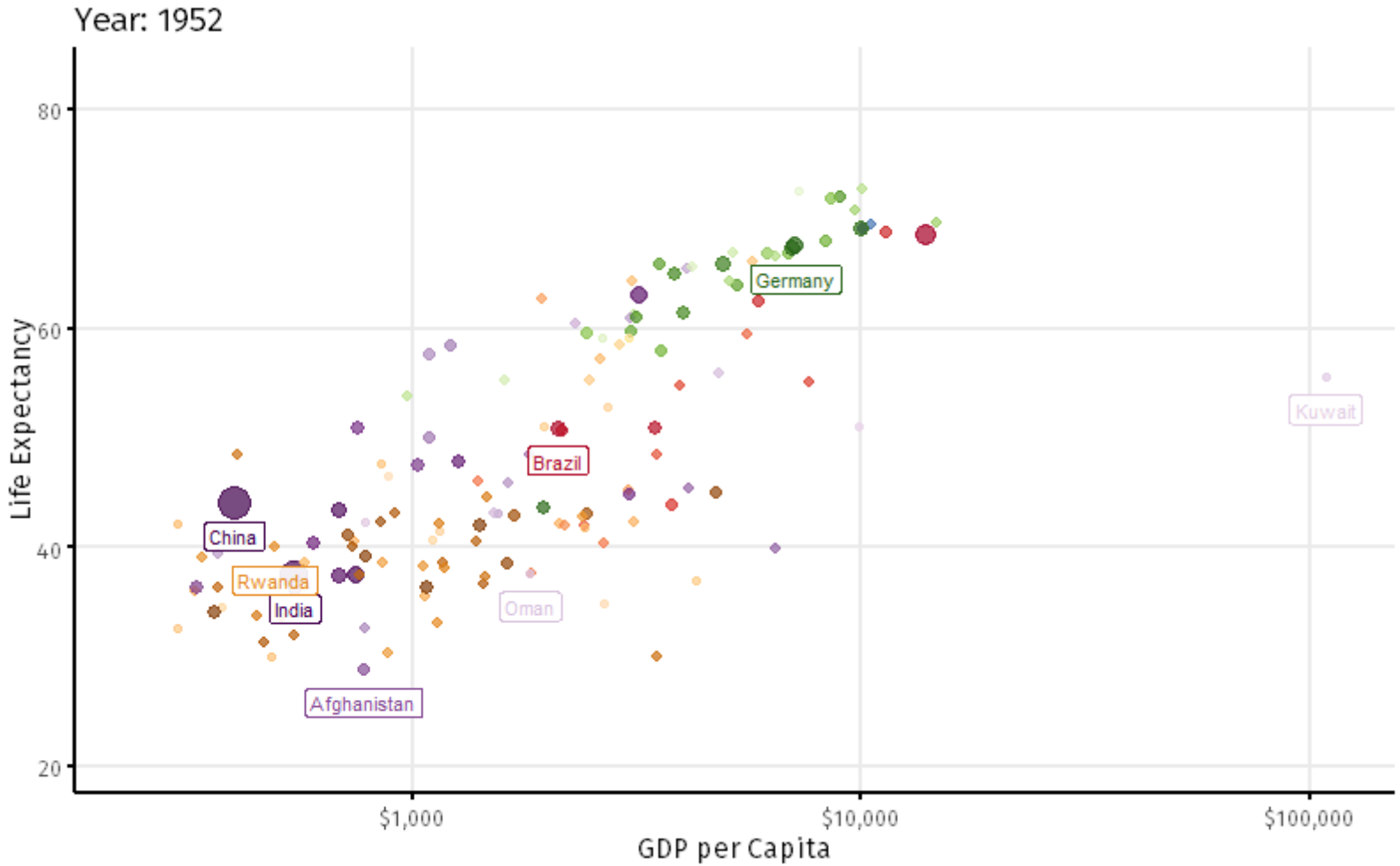
plotly

`plotly`: interface to eponymous Javascript library to create interactive graphs. The comfort function `ggplotly()` converts a `ggplot2` plot into a `plotly` graph



gganimate

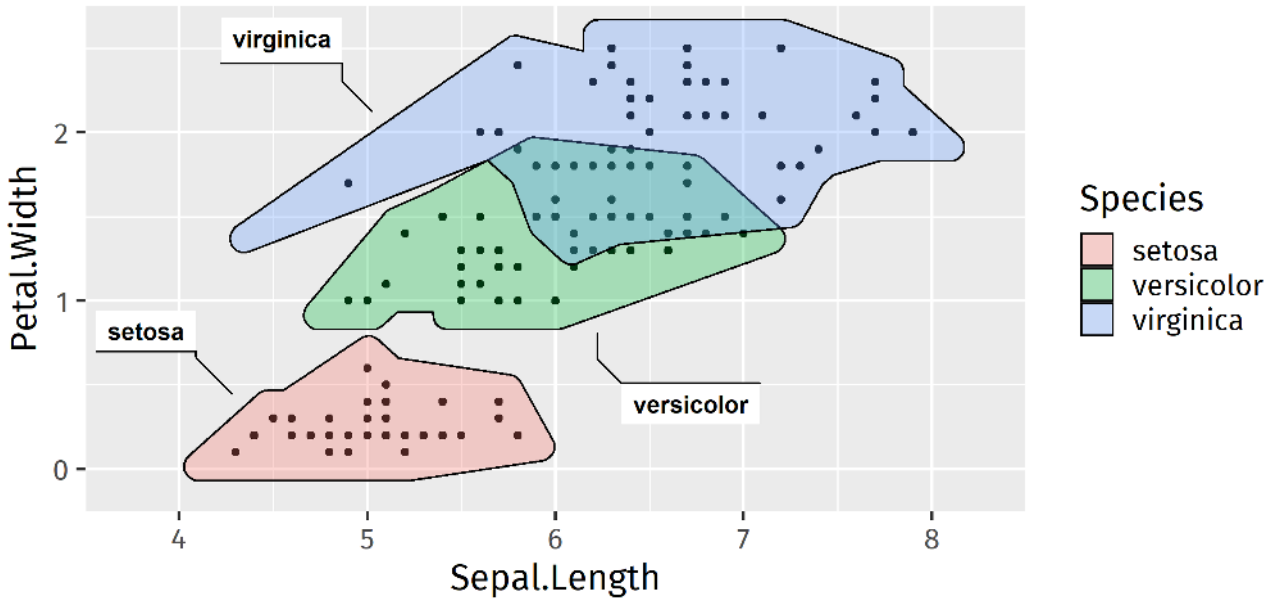
`gganimate`: animated `ggplot2` plots



ggforce

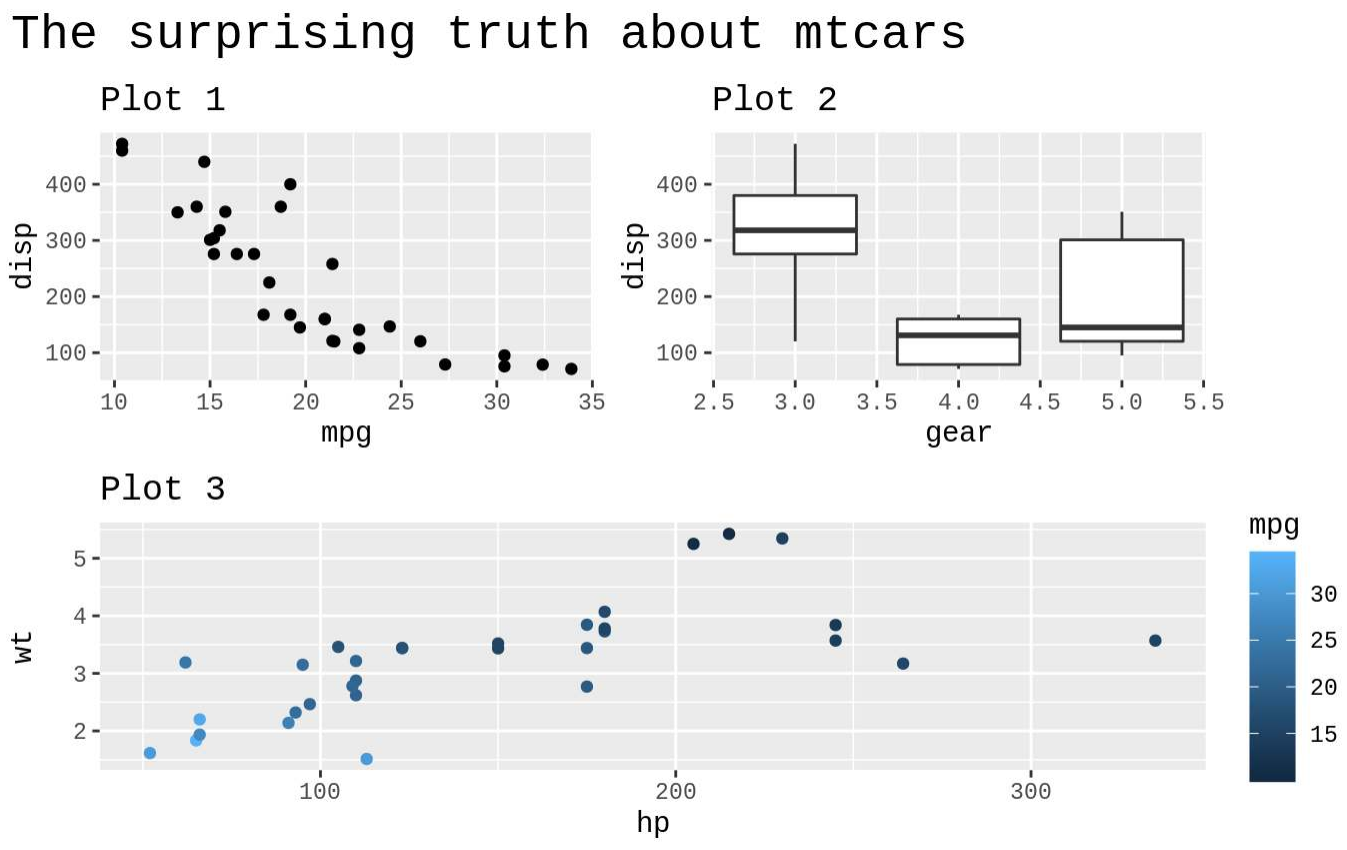
`ggforce`: various additional extensions to `ggplot2`

```
library(ggforce)
ggplot(iris, aes(Sepal.Length, Petal.Width)) +
  coord_cartesian(xlim = c(3.5,8.5), ylim = c(-0.25,2.75),
                  expand = F) +
  geom_point() +
  geom_mark_hull(aes(fill = Species, label = Species),
                 concavity = 3)
```



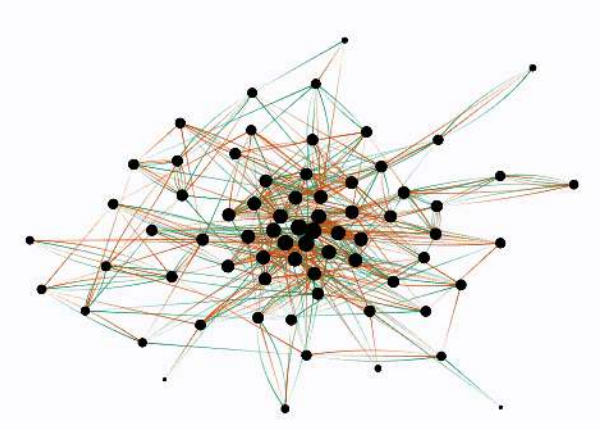
patchwork

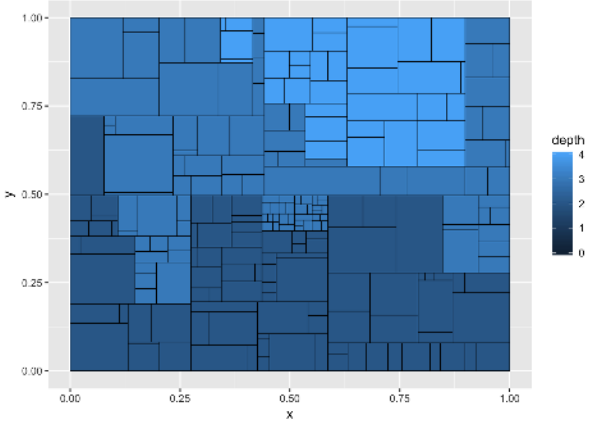
`patchwork`: combine multiple different ggplot2 graphs into a composite plot

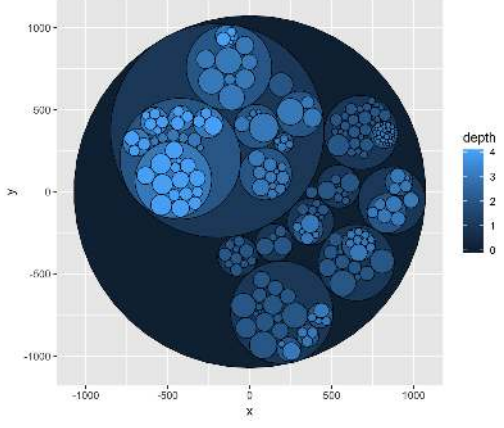


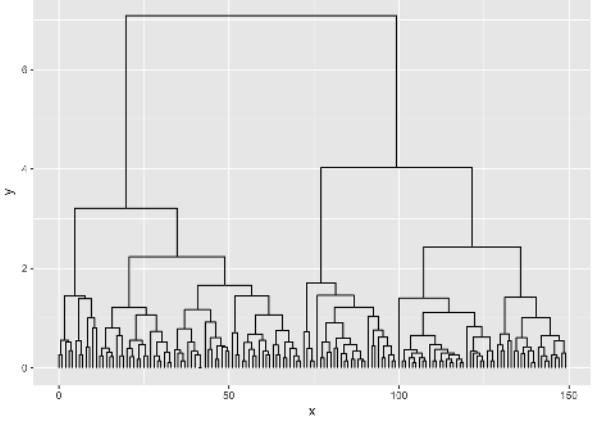
ggraph

`ggraph`: graph and network visualizations[1]



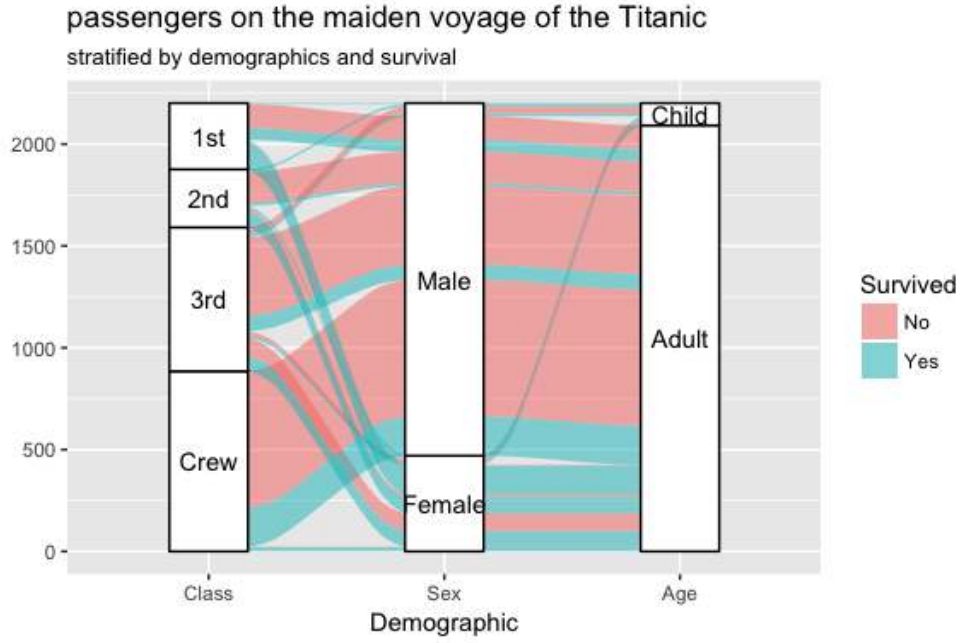


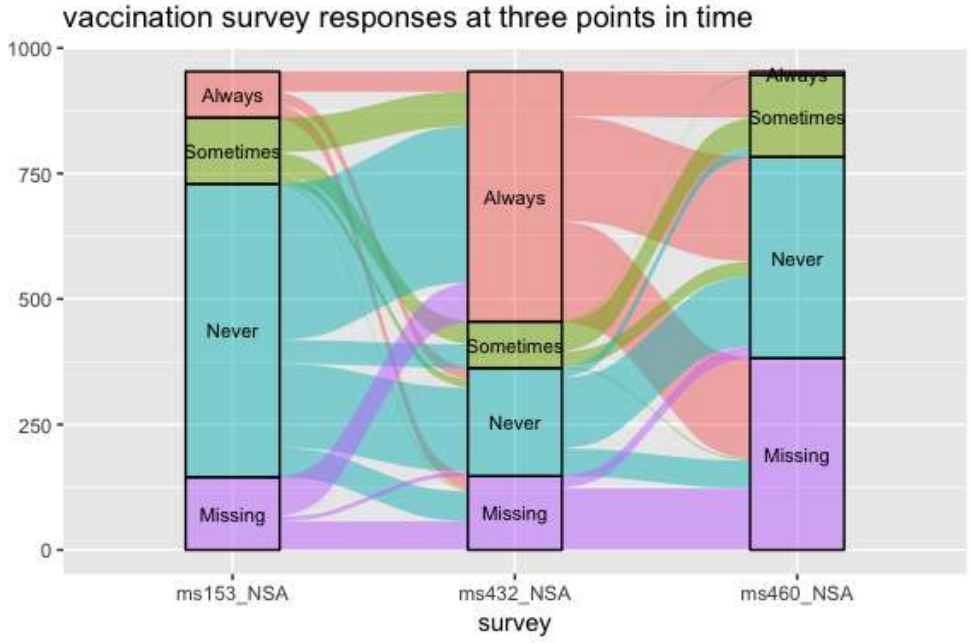




ggalluvial

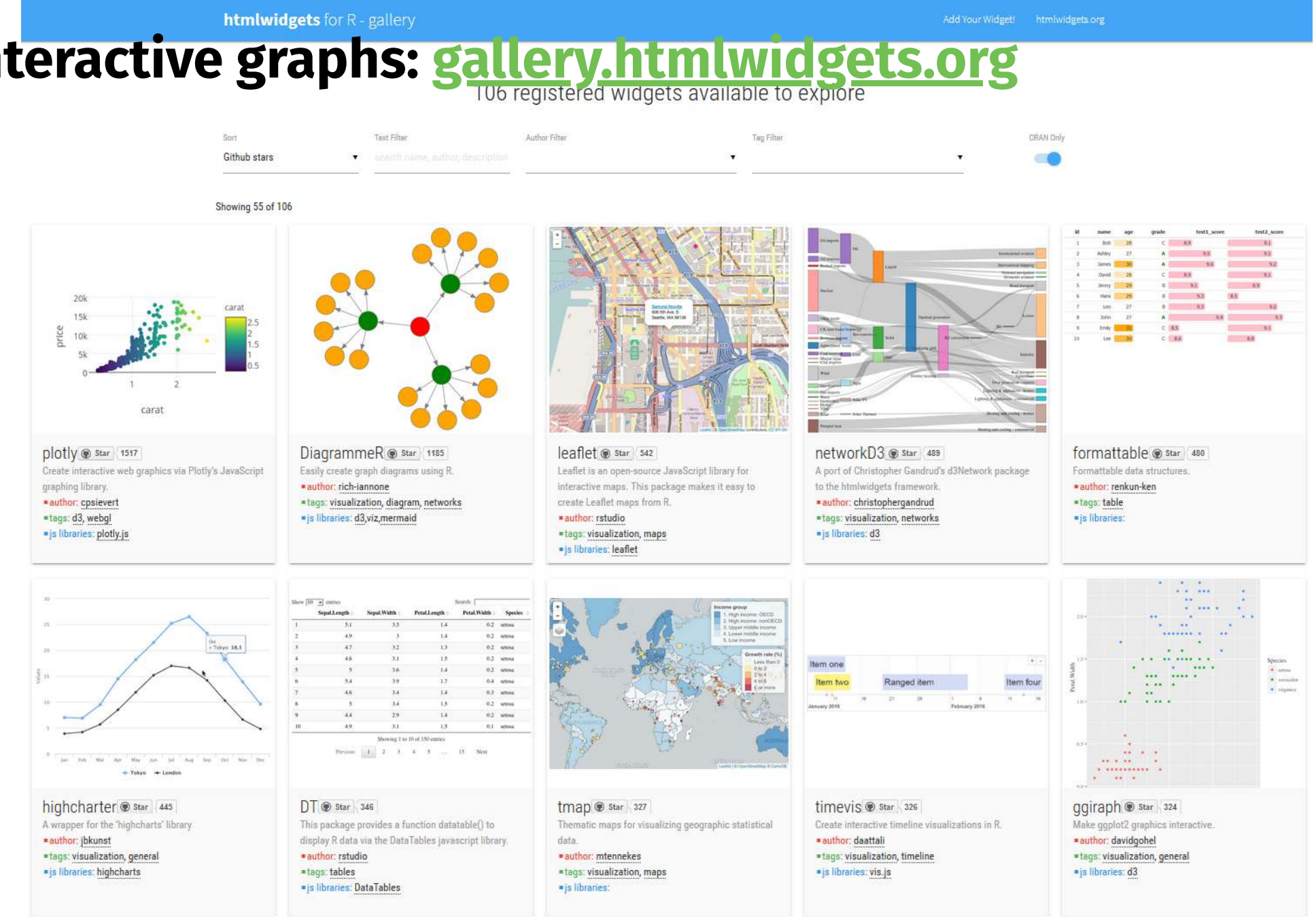
`ggalluvial`: graphs for multi-dimensional categorical count data or repeated categorical measurement data[1] (Sankey diagrams)





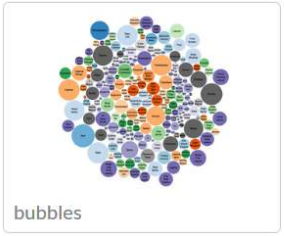
[1] URL: https://cran.r-project.org/web/packages/ggalluvial/index.html

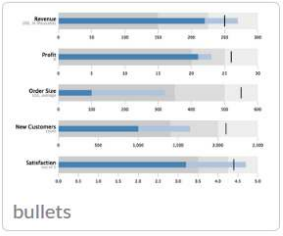
Interactive graphs: gallery.htmlwidgets.org



r2d3

`r2d3`: `R` interface to D3 Javascript library.


bubbles

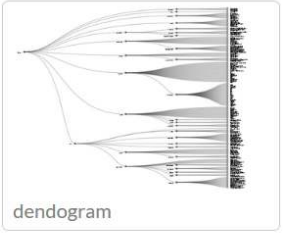

bullets

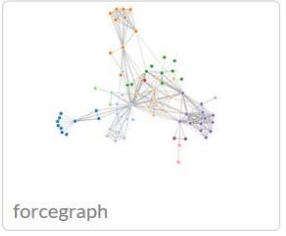

calendar

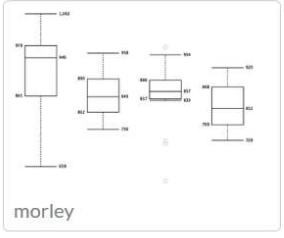

cartogram

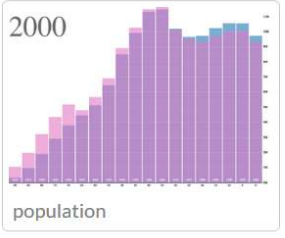

chord


circlepacking

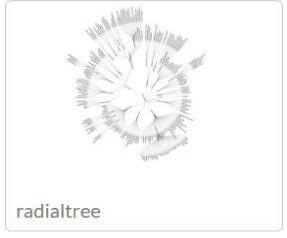

dendogram

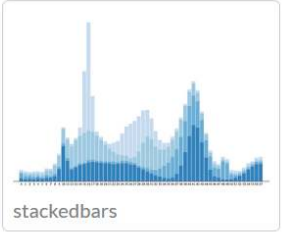

forcegraph

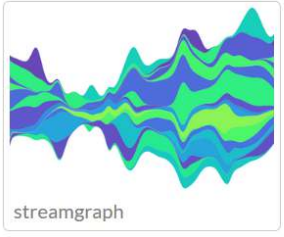

morley

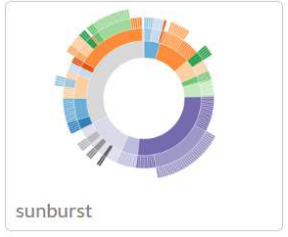


population


radialtree


stackedbars


streamgraph


sunburst


treemap


voronoi

Abstraction in Software

Easy things are awkward

Hard things are straightforward

Really hard things are doable

Easy things are trivial

Hard things are really awkward

Really hard things are impossible

D3

Grid

ggplot

Stata

Excel

Kieran Healy. A Practical Introduction to Data Visualization with ggplot2 Workshop. rstudio::conf 2020.

Session info

```
##  setting  value
##  version  R version 4.0.4 (2021-02-15)
##  os       Windows 10 x64
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  English_United States.1252
##  ctype    English_United States.1252
##  tz       Europe/Berlin
##  date     2021-03-31
```

package	version	date	source
dplyr	1.0.5	2021-03-05	CRAN (R 4.0.4)
forcats	0.5.1	2021-01-27	CRAN (R 4.0.3)
gapminder	0.3.0	2017-10-31	CRAN (R 4.0.3)
ggforce	0.3.2	2020-06-23	CRAN (R 4.0.2)
ggplot2	3.3.3	2020-12-30	CRAN (R 4.0.3)
patchwork	1.1.1	2020-12-17	CRAN (R 4.0.3)

package	version	date	source
purrr	0.3.4	2020-04-17	CRAN (R 4.0.2)
readr	1.4.0	2020-10-05	CRAN (R 4.0.3)
stringr	1.4.0	2019-02-10	CRAN (R 4.0.2)
tibble	3.1.0	2021-02-25	CRAN (R 4.0.3)
tidyr	1.1.3	2021-03-03	CRAN (R 4.0.4)
tidyverse	1.3.0	2019-11-21	CRAN (R 4.0.2)


Thank you! Questions?
-->
Photo courtesy of Stefan Berger