

# Plotting II

Explorative Datenanalyse mit R

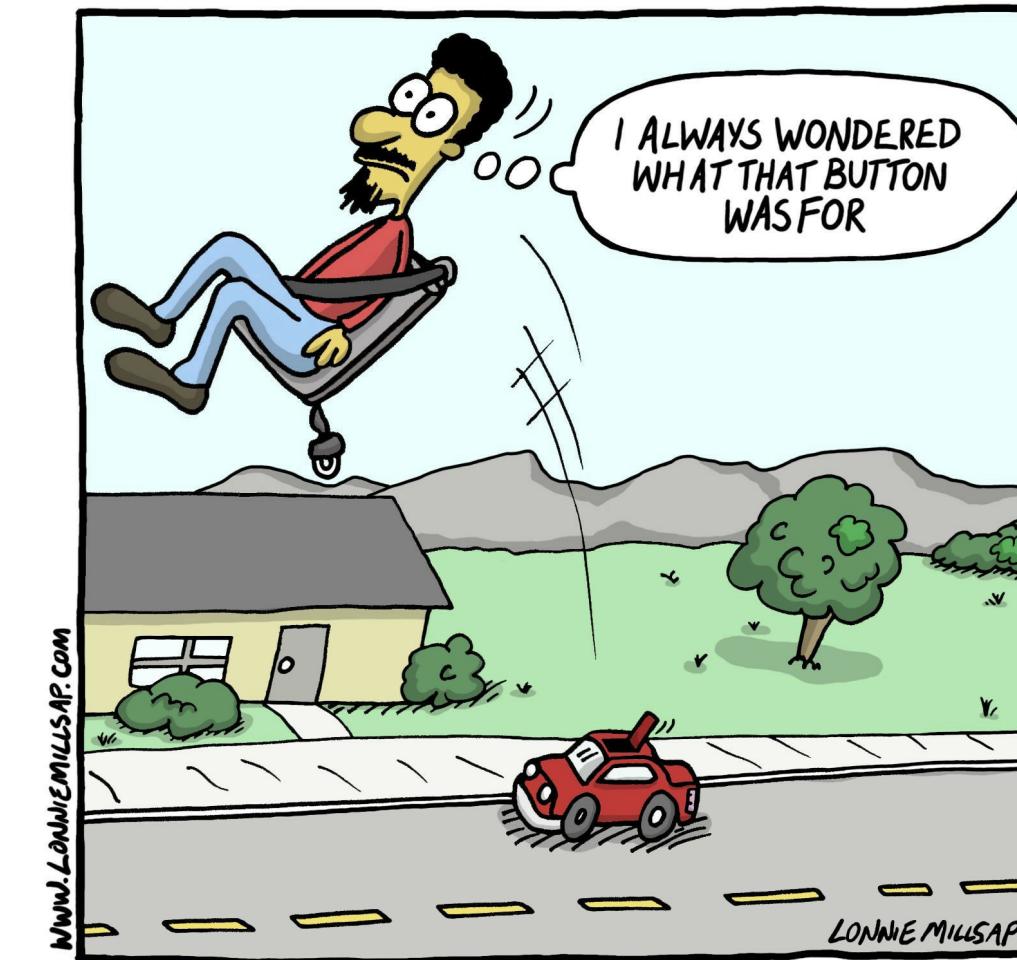
The R Bootcamp @ CSS



Dezember 2019

# Plots anpassen

- 1 Kreiere und verändere ein `gg` Plot-Objekt.
- 2 Teile den Plot auf mit `facets`
- 3 Nutze `themes` um einzelne Aspekte des Plots zu formatieren
- 4 Kreiere deine eigenen `themes`.
- 5 Speichere deinen Plot als `.pdf` oder `.png`.



from [lonniemillsap.com](http://www.lonniemillsap.com)

# Das gg Objekt

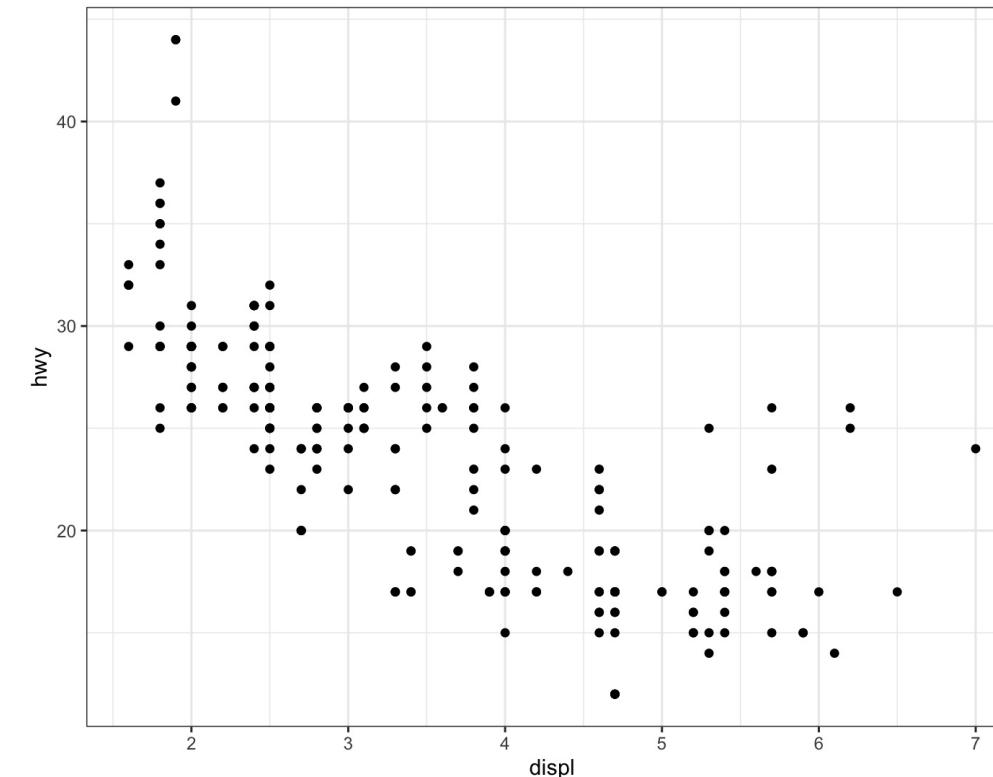
- 1 Die `ggplot` generiert ein **gg Objekt**, dass gespeichert werden kann.
- 2 **gg Objekte** können durch + beliebig **verändert/erweitert** werden
- 3 Ausführen des Objekts generiert den Plot.

```
# Weise Plot Objekt zu
mein_plot <- ggplot(data = mpg,
                     aes(x = displ, y = hwy)) +
  geom_point() + theme_bw()

# Zeige Klasse
class(mein_plot)
```

[1] "gg" "ggplot"

mein\_plot



# Das gg Objekt

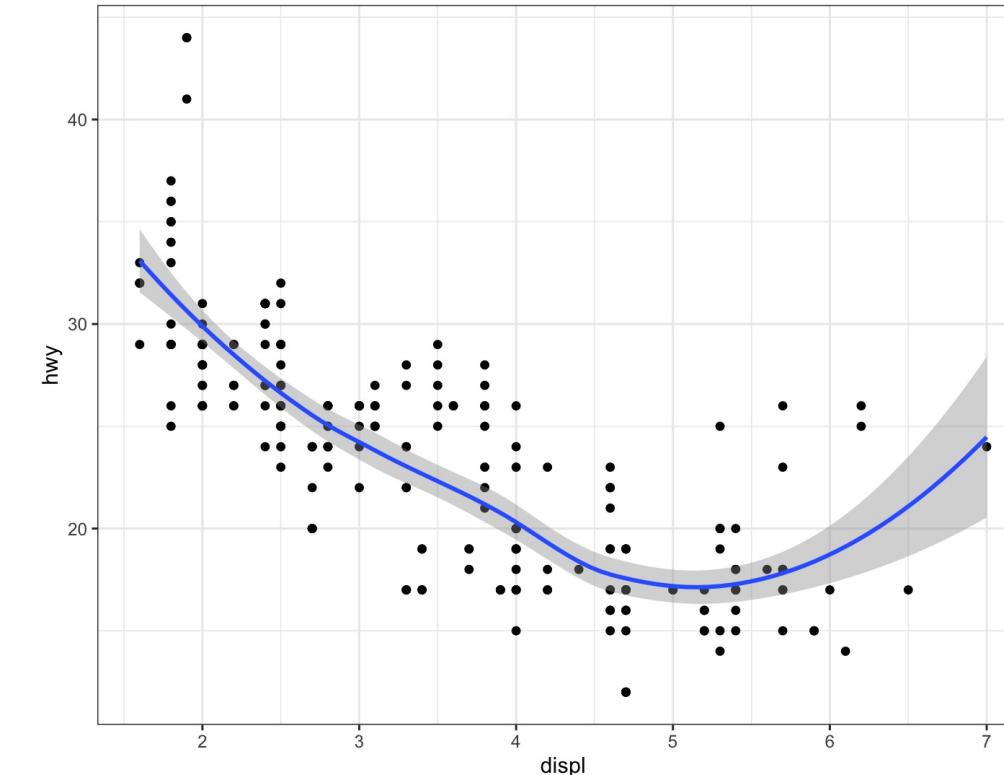
- 1 Die `ggplot` generiert ein **gg Objekt**, dass gespeichert werden kann.
- 2 **gg Objekte können durch + beliebig verändert/erweitert werden**
- 3 Ausführen des Objekts generiert den Plot.

```
# Weise Plot Objekt zu
mein_plot <- ggplot(data = mpg,
                     aes(x = displ, y = hwy)) +
  geom_point() + theme_bw()

# Zeige Klasse
class(mein_plot)
```

[1] "gg" "ggplot"

```
mein_plot + geom_smooth()
```

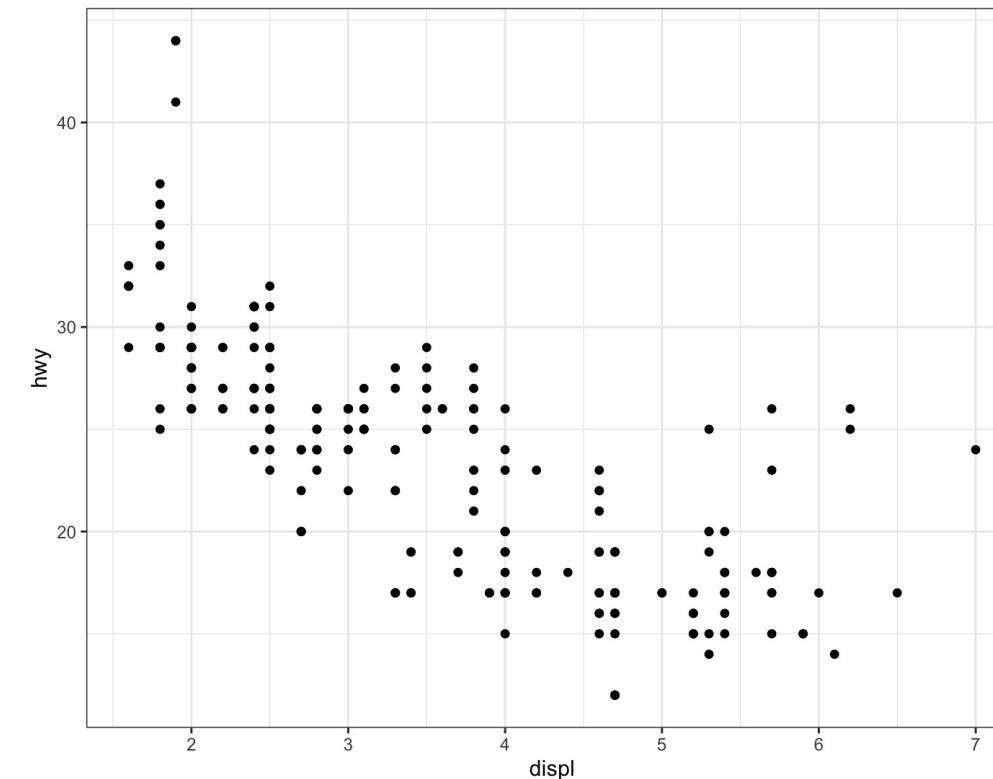


# facet\_\*

- 1 Facetting kreiert **denselben Plot für Gruppen** definiert durch dritte Variablen.
- 2 Facet Funktionen:

- `facet_wrap()`
- `facet_grid()`

```
# Ohne facetting
ggplot(data = mpg,
       mapping = aes(x = displ,
                      y = hwy)) +
  geom_point() + theme_bw()
```

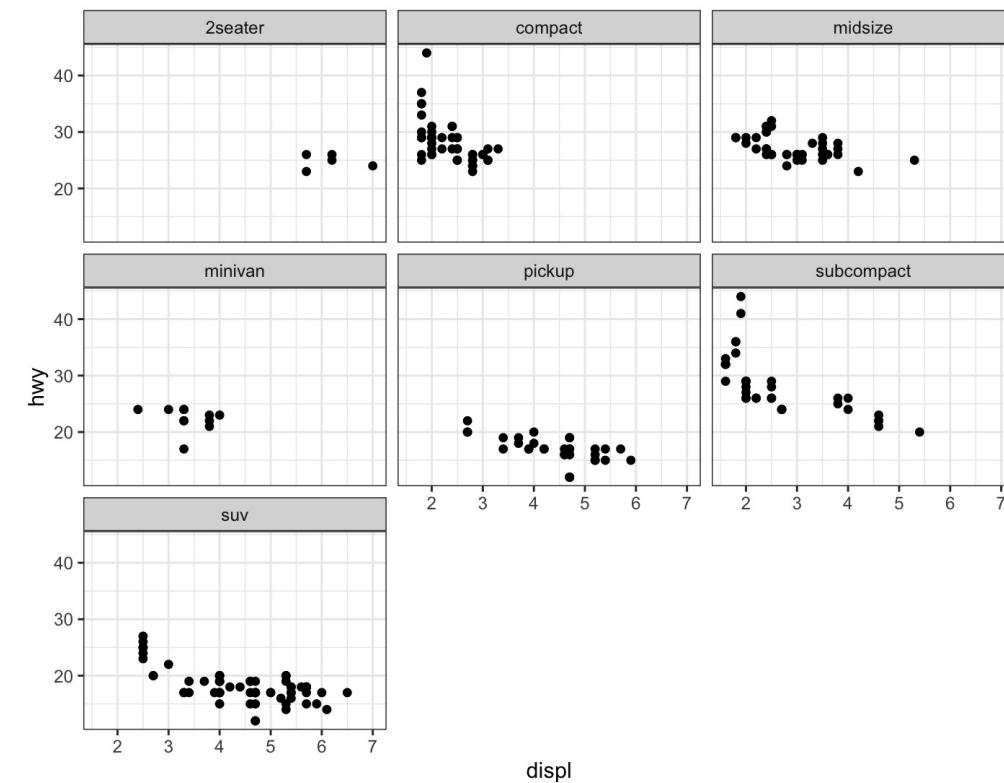


# facet\_wrap()

- 1 Facetting kreiert **denselben Plot für Gruppen** definiert durch dritte Variablen.
- 2 Facet Funktionen:

- `facet_wrap()`
- `facet_grid()`

```
# Ohne facetting
ggplot(data = mpg,
       mapping = aes(x = displ,
                      y = hwy)) +
  geom_point() + theme_bw() +
  facet_wrap(~ class)
```

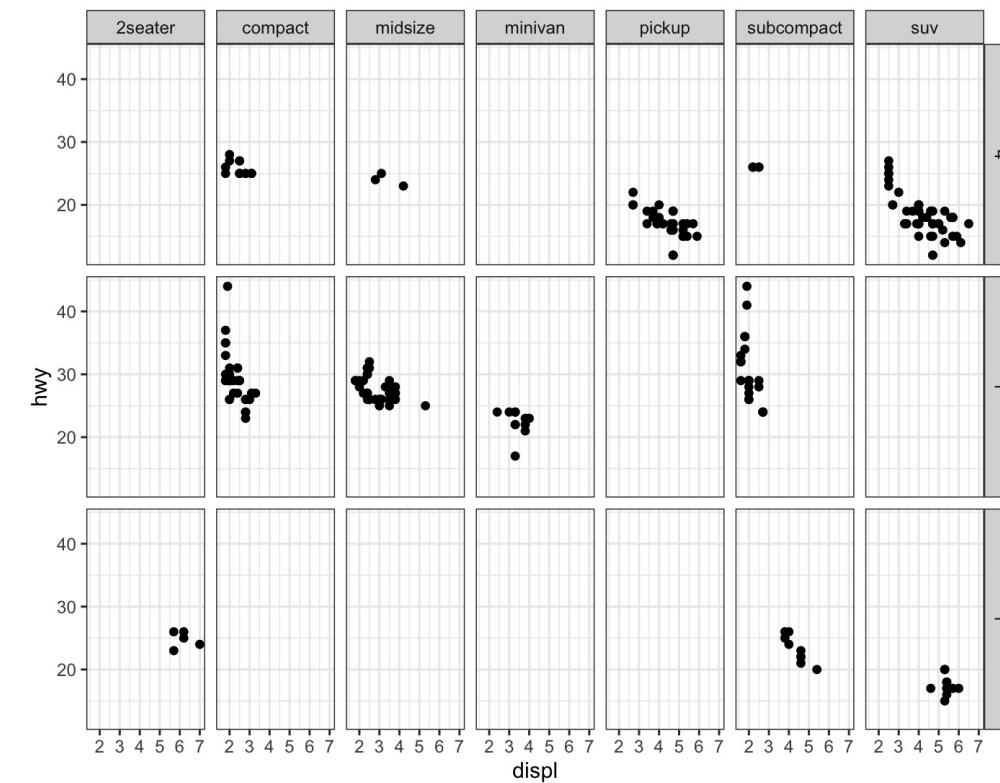


# facet\_grid()

- 1 Facetting kreiert **denselben Plot für Gruppen** definiert durch dritte Variablen.
- 2 Facet Funktionen:

- `facet_wrap()`
- `facet_grid()`

```
# Ohne facetting
ggplot(data = mpg,
       mapping = aes(x = displ,
                      y = hwy)) +
  geom_point() + theme_bw() +
  facet_grid(drv ~ class)
```



# theme()

1 Passt mit den **87 Argumenten von theme()** alle ästhetischen Aspekte deines Plots an.

2 Nutze hierzu vier Helferfunktionen:

- element\_rect() | für Flächen
- element\_line() | für Linien
- element\_text() | für Text
- element\_blank() | zum entfernen

```
# Verwendung von theme
mein_plot +
  theme(argument = element_(),
        argument = element_(),
        ...)
```

theme {ggplot2}

R Documentation

## Modify components of a theme

### Description

Use `theme()` to modify individual components of a theme, allowing you to control the appearance of all non-data components of the plot. `theme()` only affects a single plot: see [theme\\_update\(\)](#) if you want modify the active theme, to affect all subsequent plots.

### Usage

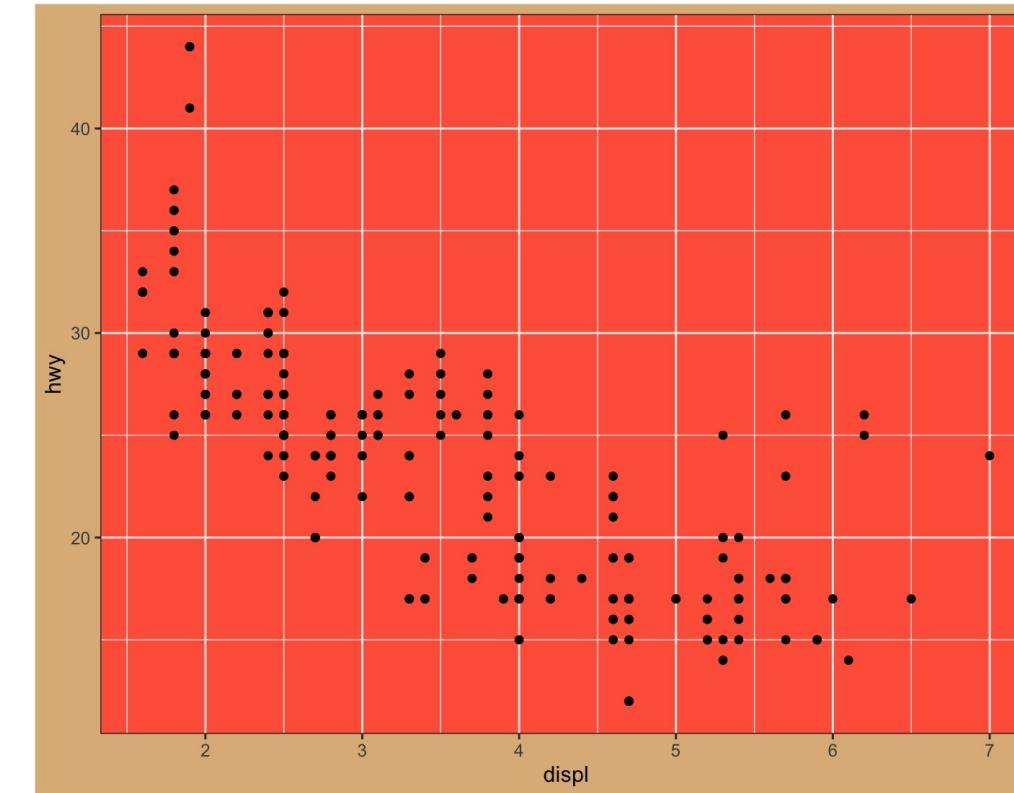
```
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x,
      axis.title.x.top, axis.title.x.bottom, axis.title.y, axis.title.y.left,
      axis.title.y.right, axis.text, axis.text.x, axis.text.x.top,
      axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right,
      axis.ticks, axis.ticks.x, axis.ticks.x.top, axis.ticks.x.bottom, axis.ticks.y,
      axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length, axis.line,
      axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y,
      axis.line.y.left, axis.line.y.right, legend.background, legend.margin,
      legend.spacing, legend.spacing.x, legend.spacing.y, legend.key,
      legend.key.size, legend.key.height, legend.key.width, legend.text,
      legend.text.align, legend.title, legend.title.align, legend.position,
      legend.direction, legend.justification, legend.box, legend.box.just,
      legend.box.margin, legend.box.background, legend.box.spacing,
      panel.background, panel.border, panel.spacing, panel.spacing.x,
      panel.spacing.y, panel.grid, panel.grid.major, panel.grid.minor,
      panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x,
      panel.grid.minor.y, panel.on top, plot.background, plot.title, plot.subtitle,
      plot.caption, plot.tag, plot.tag.position, plot.margin, strip.background,
      strip.background.x, strip.background.y, strip.place ment, strip.text,
      strip.text.x, strip.text.y, strip.switch.pad.grid, strip.switch.pad.wrap, ...,
      complete = FALSE, validate = TRUE)
```

# Hintergrund

## 1 Argumente für den Hintergrund:

- `panel.background` | für den inneren Hintergrund
- `plot.background` | für den äusseren Hintergrund

```
# Ändere den Hintergrund
mein_plot +
  theme(
    panel.background =
      element_rect(fill = 'tomato'),
    plot.background =
      element_rect(fill = 'burlywood'))
```



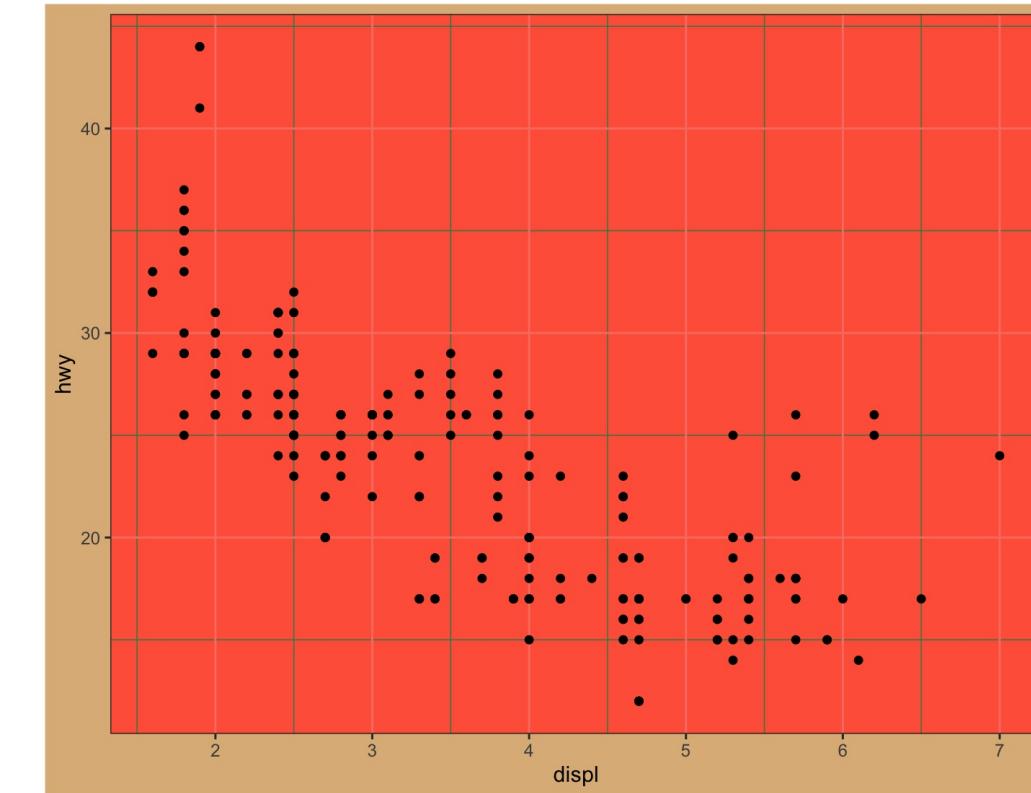
# Raster

1

Argumente für das Raster:

- `panel.grid.major` | grössere Rasterlinien
- `panel.grid.minor` | kleinere Rasterlinien

```
# Ändere das Raster
mein_plot +
  theme(
    panel.grid.major =
      element_line(colour = "salmon"),
    panel.grid.minor =
      element_line(colour = "seagreen"))
```



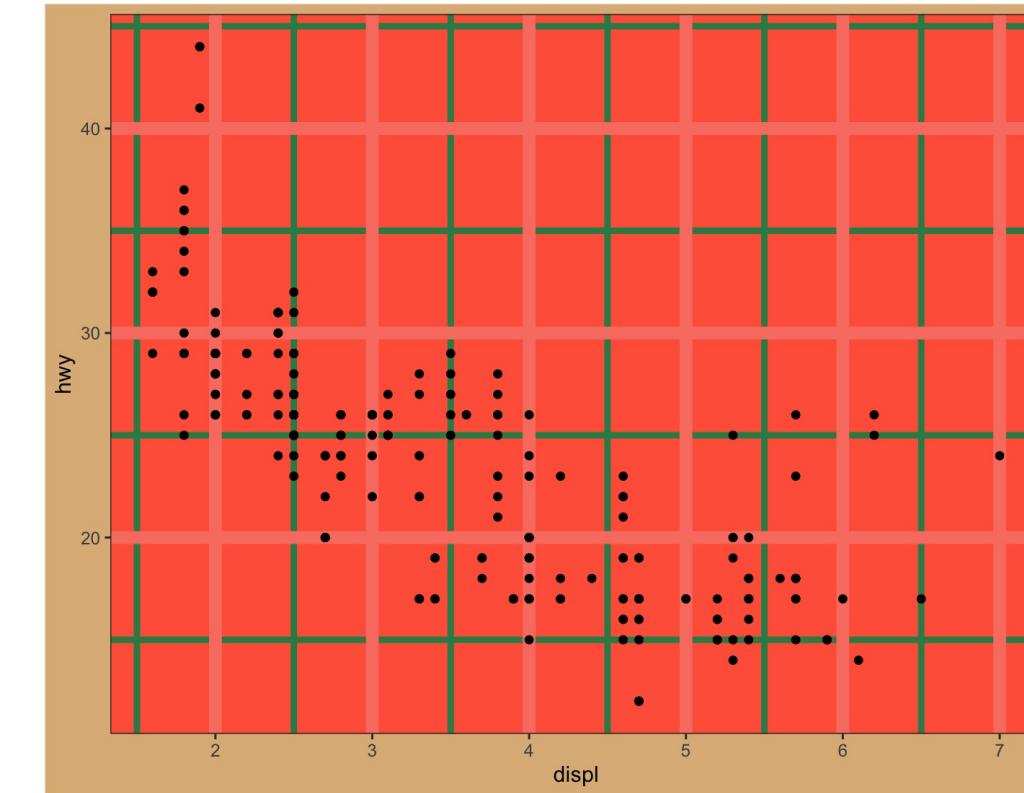
# Raster

1

Argumente für das Raster:

- `panel.grid.major` | grössere Rasterlinien
- `panel.grid.minor` | kleinere Rasterlinien

```
# Ändere das Raster
mein_plot +
  theme(
    panel.grid.major =
      element_line(colour = "salmon",
                   size = 3),
    panel.grid.minor =
      element_line(colour = "seagreen",
                   size = 1.5))
```

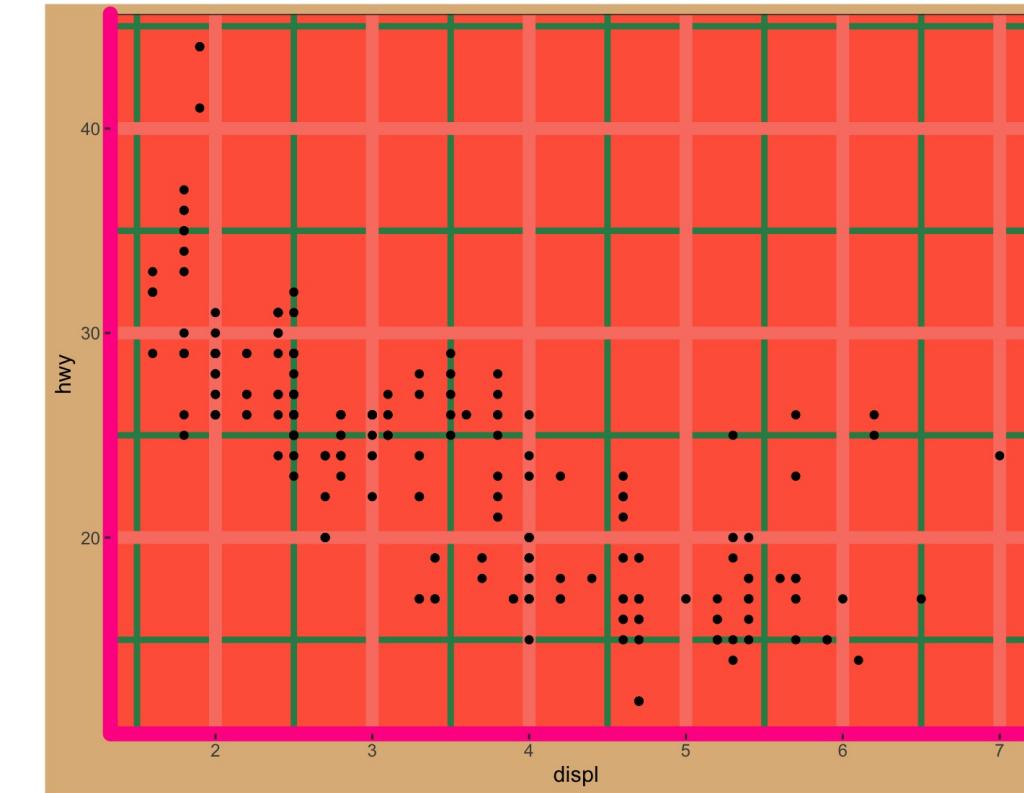


# Achsen

## 1 Argumente für die Achsen:

- o `axis.line.x` | x-Achse
- o `axis.line.y` | y-Achse
- o `axis.title.x` | x-Achse Titel
- o `axis.title.y` | y-Achse Titel

```
# Ändere die Achsen
mein_plot +
  theme(
    axis.line.x =
      element_line(colour = "deeppink",
                   size = 3.5,
                   lineend = "butt"),
    axis.line.y =
      element_line(colour = "deeppink",
                   size = 3.5))
```

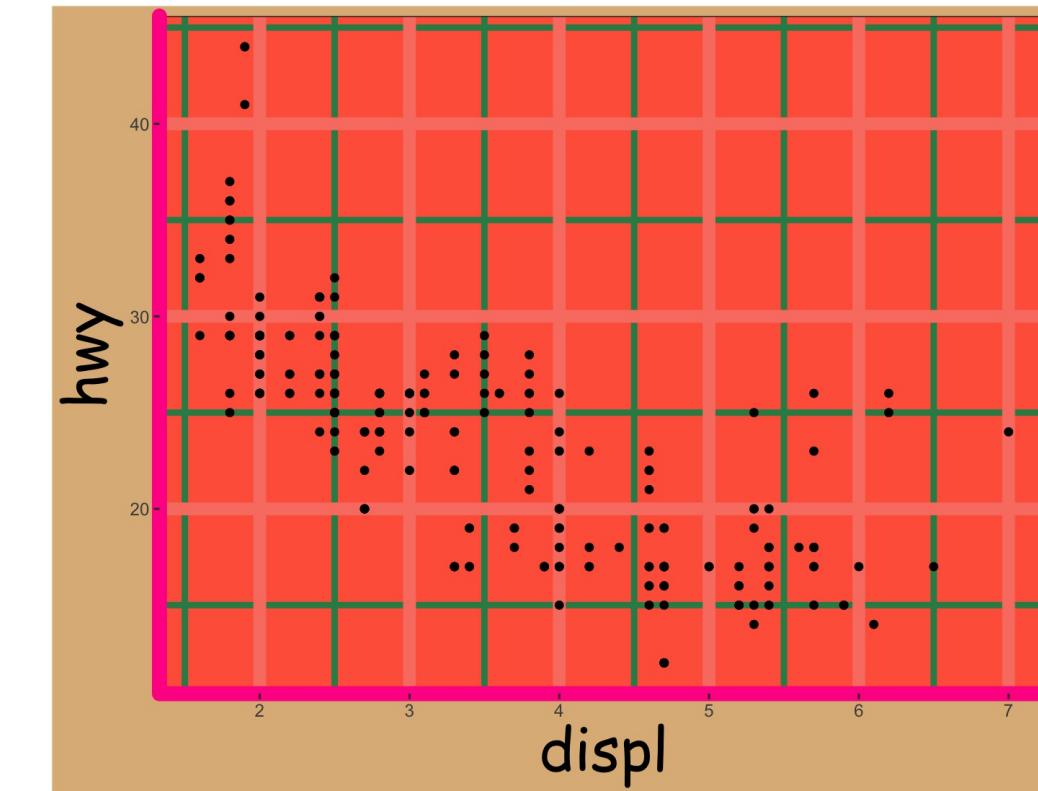


# Achsen

## 1 Argumente für die Achsen:

- `axis.line.x` | x-Achse
- `axis.line.y` | y-Achse
- `axis.title.x` | x-Achse Titel
- `axis.title.y` | y-Achse Titel

```
# change grid line color
mein_plot +
  theme(
    axis.title.x =
      element_text(family = "Comic Sans MS",
                   size = 30),
    axis.title.y =
      element_text(family = "Comic Sans MS",
                   size = 30))
```



# Weitere Argumente

(unvollständig)

## theme()

Argument	Beschreibung
axis.title.*	Alles betreffend Achsentitel
axis.ticks.*	Alles betreffend Achsenintervalle
axis.line.*	Alles betreffend Achsenlinien
legend.*	Alles betreffend Legenden
panel.*	Alles betreffend die innere Plotregion
plot.*	Alles betreffend die äussere Plotregion
strip.*	Alles betreffend die Facet Titel

## element\_rect()

Argument	Beschreibung
fill	Füllfarbe
colour	Randfarbe

## element\_line()

Argument	Beschreibung
size	Liniengrösse
linetype	Art der Linie

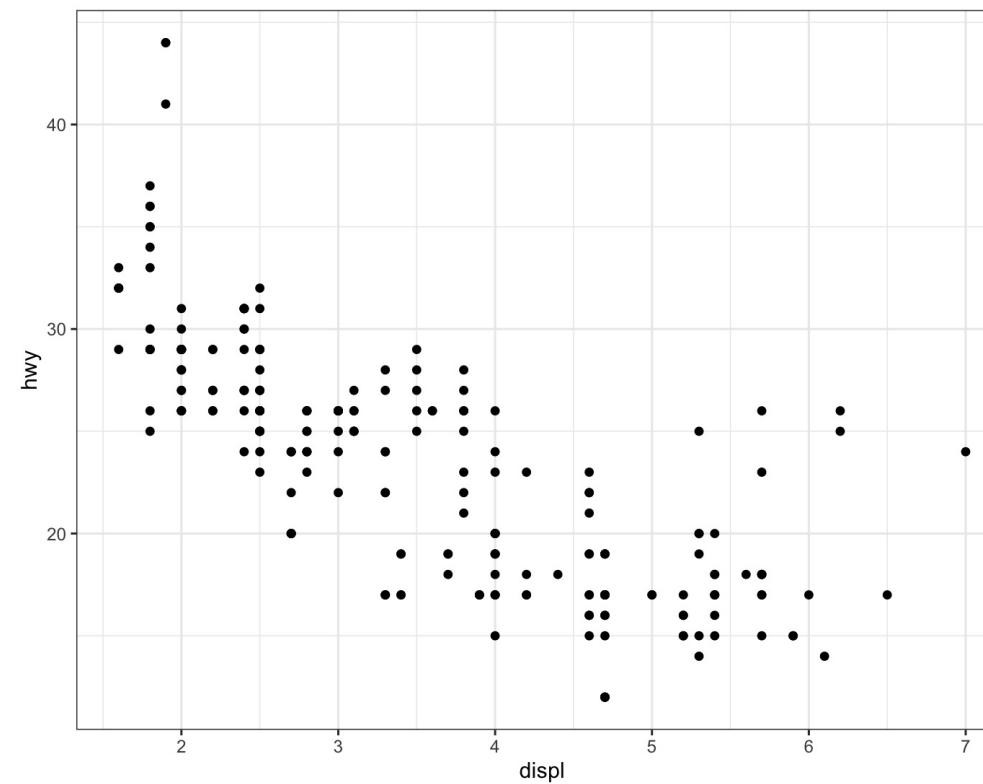
## element\_text()

Argument	Beschreibung
face	Schriftart
colour	Schriftfarbe

# Eigene themes

```
mein_theme <- theme(  
  panel.background =  
    element_rect(fill = 'tomato'),  
  plot.background =  
    element_rect(fill = 'burlywood'),  
  panel.grid.major = element_line(  
    colour = "salmon", size = 3),  
  panel.grid.minor = element_line(  
    colour = "seagreen", size = 1.5),  
  axis.line.x = element_line(  
    colour = "deeppink", size = 3.5,  
    lineend = "butt"),  
  axis.line.y = element_line(  
    colour = "deeppink", size = 3.5),  
  axis.title.x = element_text(  
    family = "Comic Sans MS", size = 30),  
  axis.title.y = element_text(  
    family = "Comic Sans MS", size = 30))
```

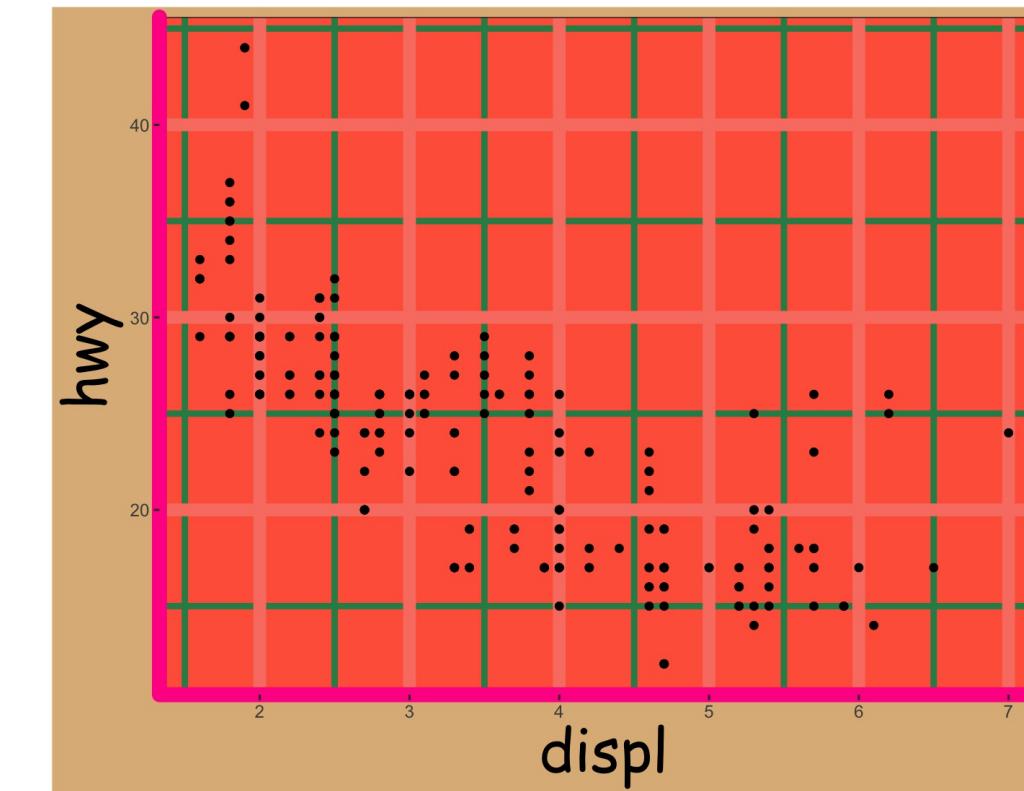
mein\_plot



# Eigene themes

```
mein_theme <- theme(  
  panel.background =  
    element_rect(fill = 'tomato'),  
  plot.background =  
    element_rect(fill = 'burlywood'),  
  panel.grid.major = element_line(  
    colour = "salmon", size = 3),  
  panel.grid.minor = element_line(  
    colour = "seagreen", size = 1.5),  
  axis.line.x = element_line(  
    colour = "deeppink", size = 3.5,  
    lineend = "butt"),  
  axis.line.y = element_line(  
    colour = "deeppink", size = 3.5),  
  axis.title.x = element_text(  
    family = "Comic Sans MS", size = 30),  
  axis.title.y = element_text(  
    family = "Comic Sans MS", size = 30))
```

```
mein_plot + mein_theme
```



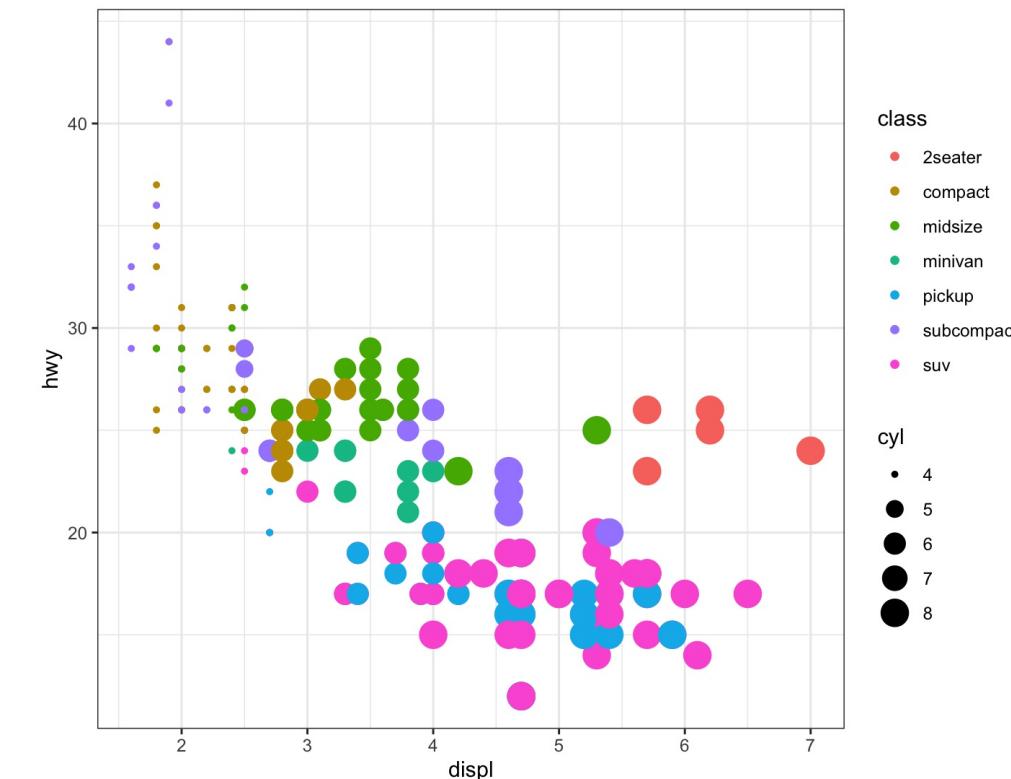
# scale\_\*

1 Mit `scale_*` Funktionen lassen sich alle möglichen Längen/Größen skalieren.

2 Klassen von `scale_*` Funktionen:

- o `scale_xy_*` | Skaliere Achsen
- o `scale_color_*`, `scale_alpha_*` | Skaliere Farben
- o `scale_size_*` | Skaliere Größen
- o `scale_alpha_*` | Skaliere Transparenzen
- o ...

mein\_plot



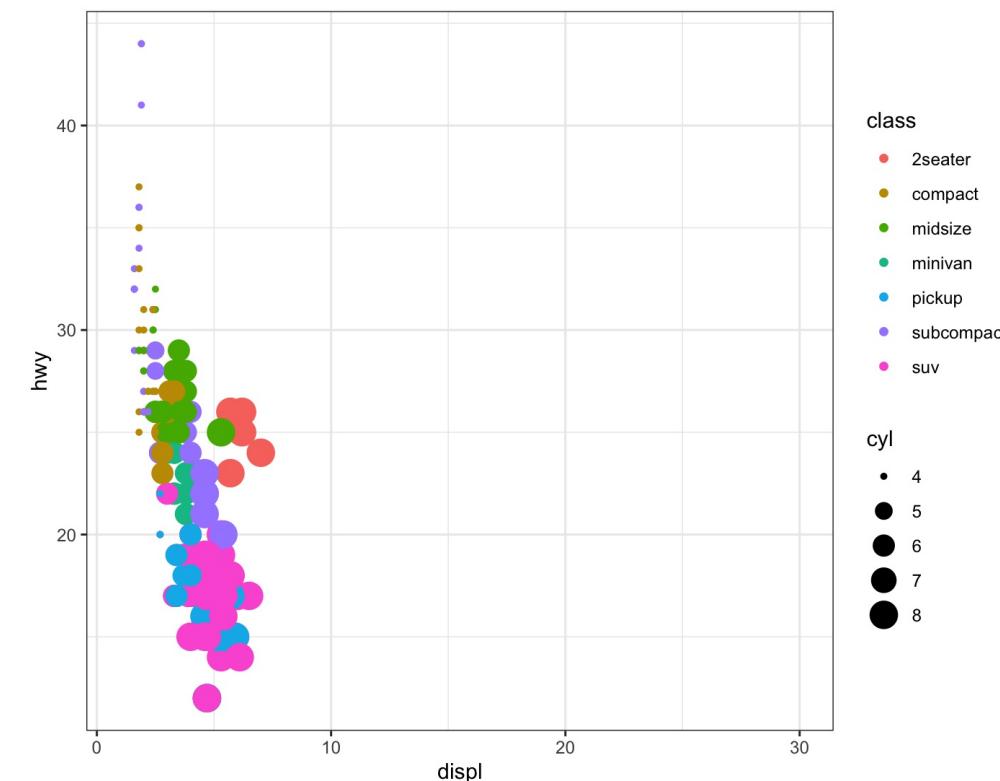
# scale\_x\_continuous()

1 Mit `scales_*`() Funktionen lassen alle möglichen Längen/Größen skalieren.

2 Klassen von `scales_*`() Funktionen:

- o `scale_xy_*` | Skaliere Achsen
- o `scale_color_*`, `scale_color_*` | Skaliere Farben
- o `scale_size_*` | Skaliere Größen
- o `scale_alpha_*` | Skaliere Transparenzen
- o ...

```
mein_plot +  
  scale_x_continuous(limits = c(1, 30))
```



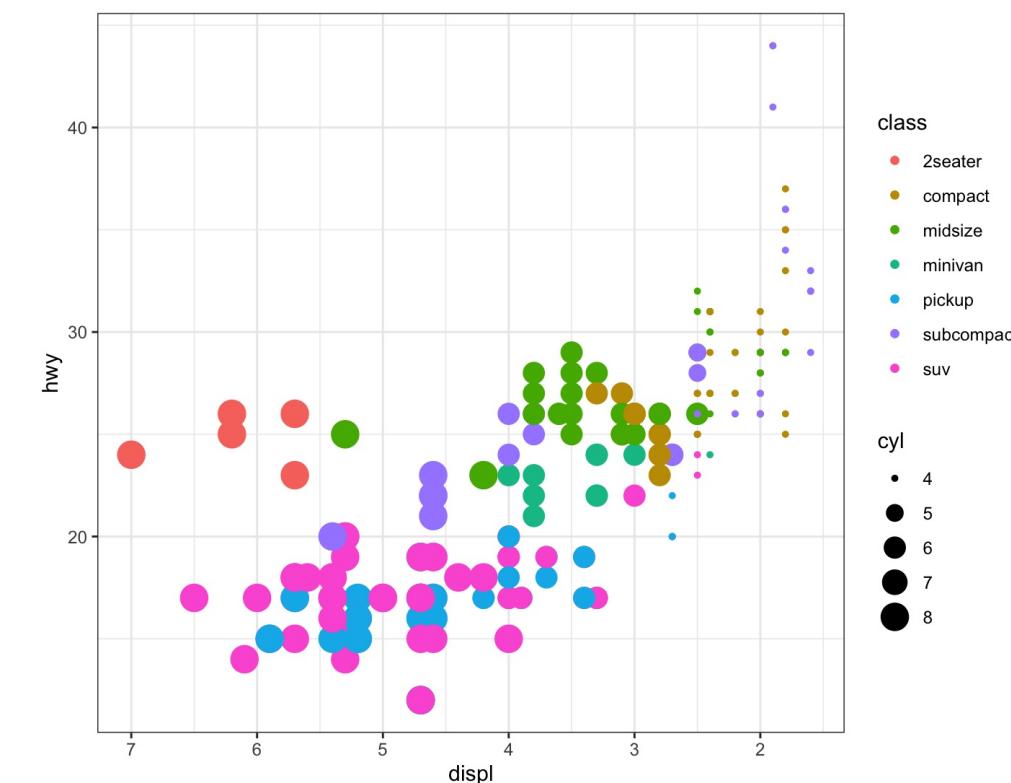
# scale\_x\_reverse()

1 Mit `scales_*`() Funktionen lassen alle möglichen Längen/Größen skalieren.

2 Klassen von `scales_*`() Funktionen:

- `scale_xy_*` | Skaliere Achsen
- `scale_color_*`, `scale_alpha_*` | Skaliere Farben
- `scale_size_*` | Skaliere Größen
- `scale_alpha_*` | Skaliere Transparenzen
- ...

```
mein_plot +  
  scale_x_reverse()
```



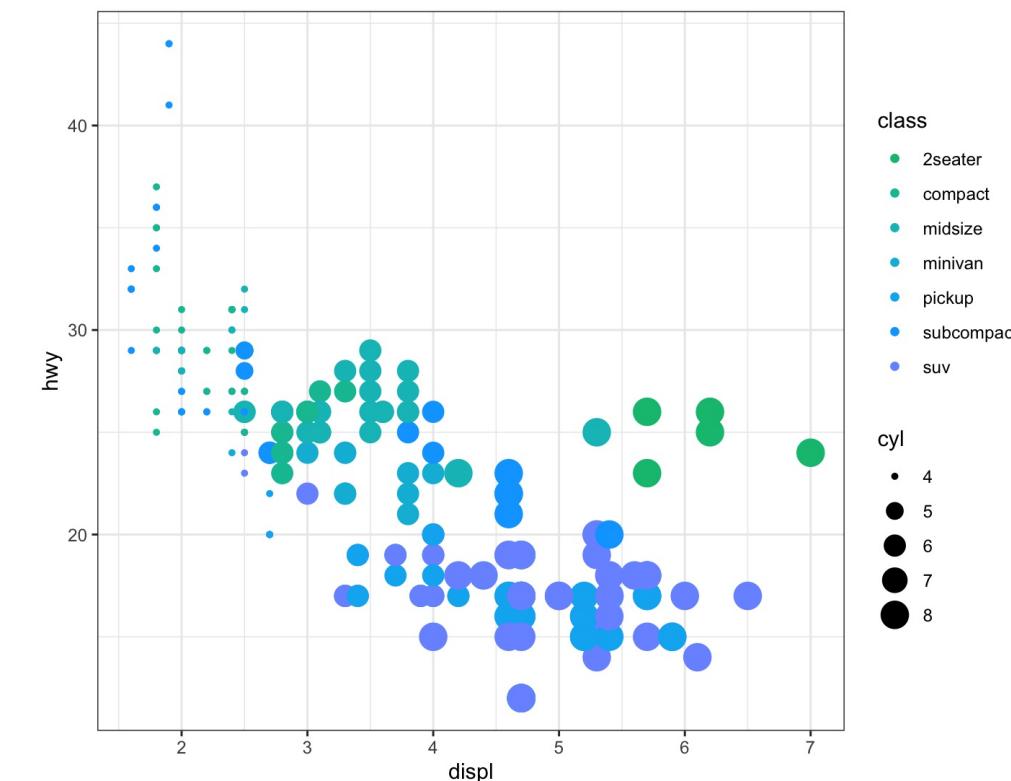
# scale\_color\_hue()

1 Mit `scales_*`() Funktionen lassen alle möglichen Längen/Größen skalieren.

2 Klassen von `scales_*`() Funktionen:

- `scale_xy_*` | Skaliere Achsen
- `scale_color_*`, `scale_color_*` | Skaliere Farben
- `scale_size_*` | Skaliere Größen
- `scale_alpha_*` | Skaliere Transparenzen
- ...

```
mein_plot +  
  scale_colour_hue(h = c(160, 260))
```



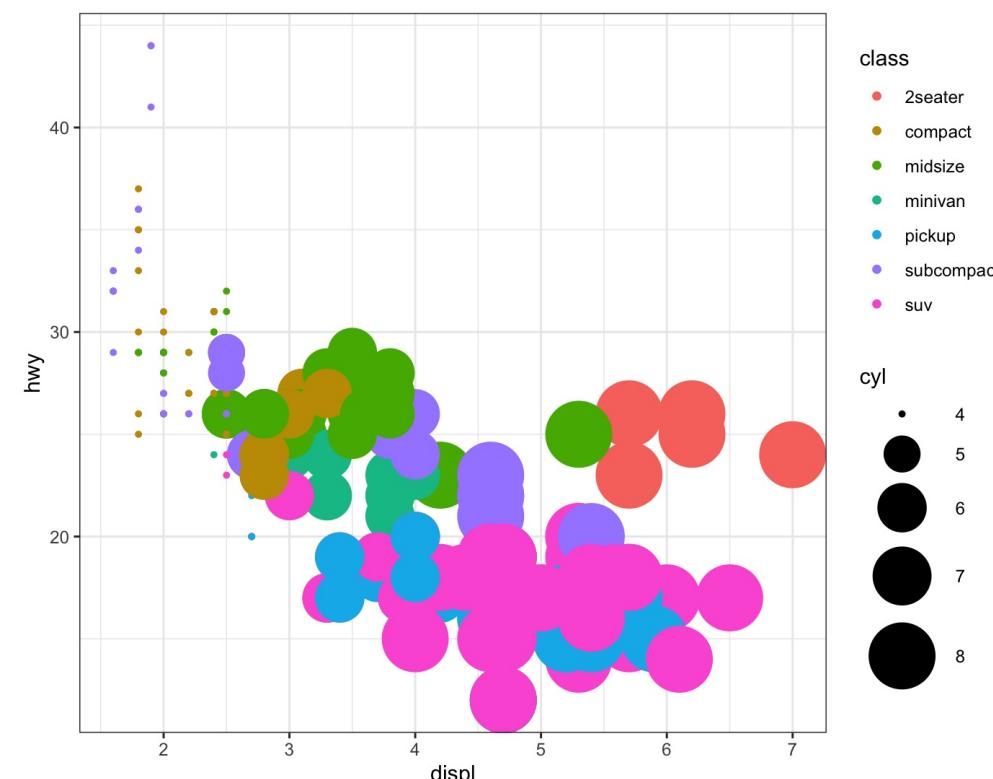
# scale\_size()

1 Mit `scales_*`() Funktionen lassen alle möglichen Längen/Größen skalieren.

2 Klassen von `scales_*`() Funktionen:

- o `scale_xy_*` | Skaliere Achsen
- o `scale_color_*`, `scale_alpha_*` | Skaliere Farben
- o `scale_size_*` | Skaliere Größen
- o `scale_alpha_*` | Skaliere Transparenzen
- o ...

```
mein_plot +  
  scale_size(range = c(1, 15))
```



# Multiple Plots

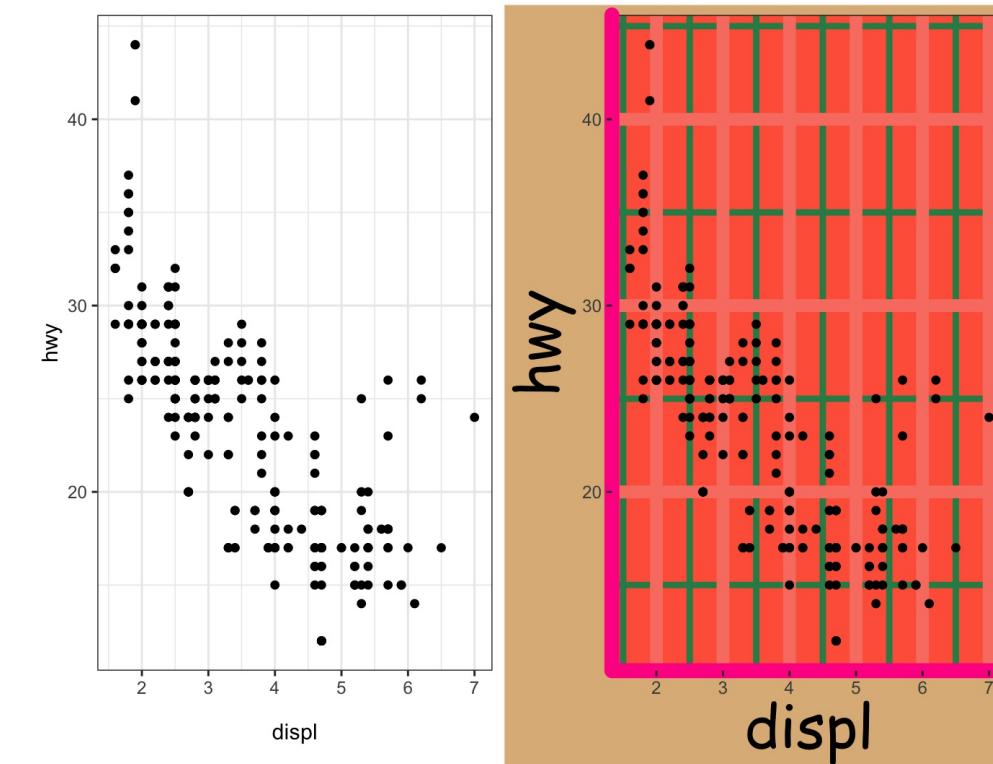
1 Das patchwork Paket liefert eine einfache Syntax um Plots zu verknüpfen.

2 patchwork Syntax:

- + | generisch zusammefügen
- | | nebeneinander stellen
- / | untereinander stellen
- () | zusammenfassen
- & | auf alle anwenden

```
# Speichere plots
schoen <- mein_plot
unschoen <- mein_plot + mein_theme
```

schoen + unschoen



# Multiple Plots

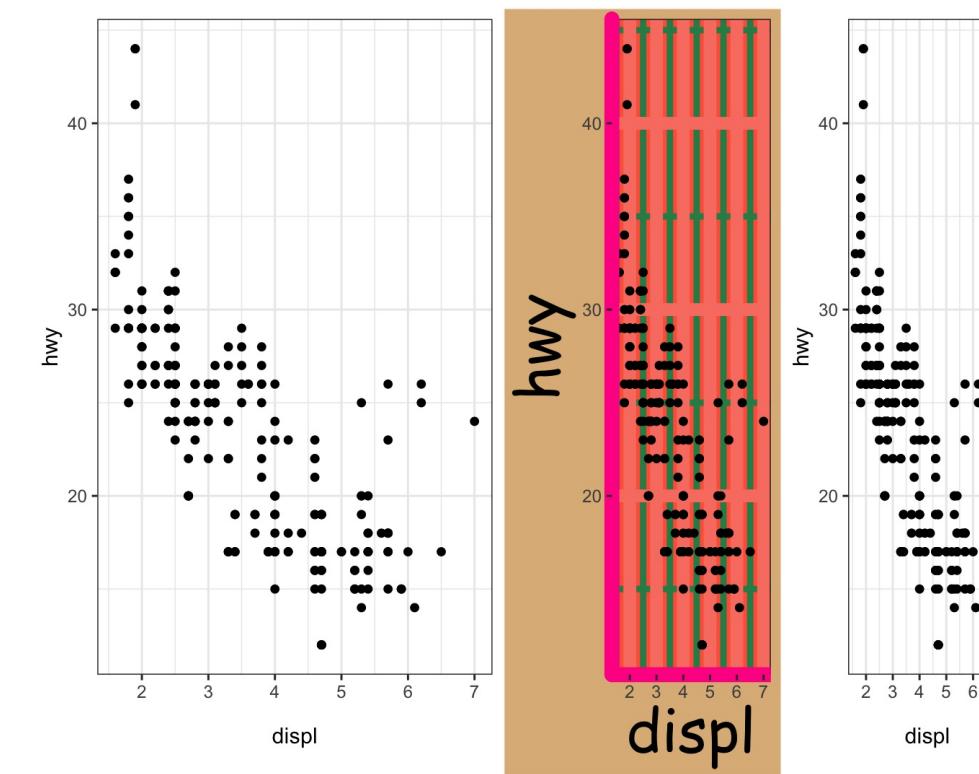
1 Das patchwork Paket liefert eine einfache Syntax um Plots zu verknüpfen.

2 patchwork Syntax:

- + | generisch zusammefügen
- | | nebeneinander stellen
- / | untereinander stellen
- () | zusammenfassen
- & | auf alle anwenden

```
# Speichere plots
schoen <- mein_plot
unschoen <- mein_plot + mein_theme
```

schoen | unschoen + schoen



# Multiple Plots

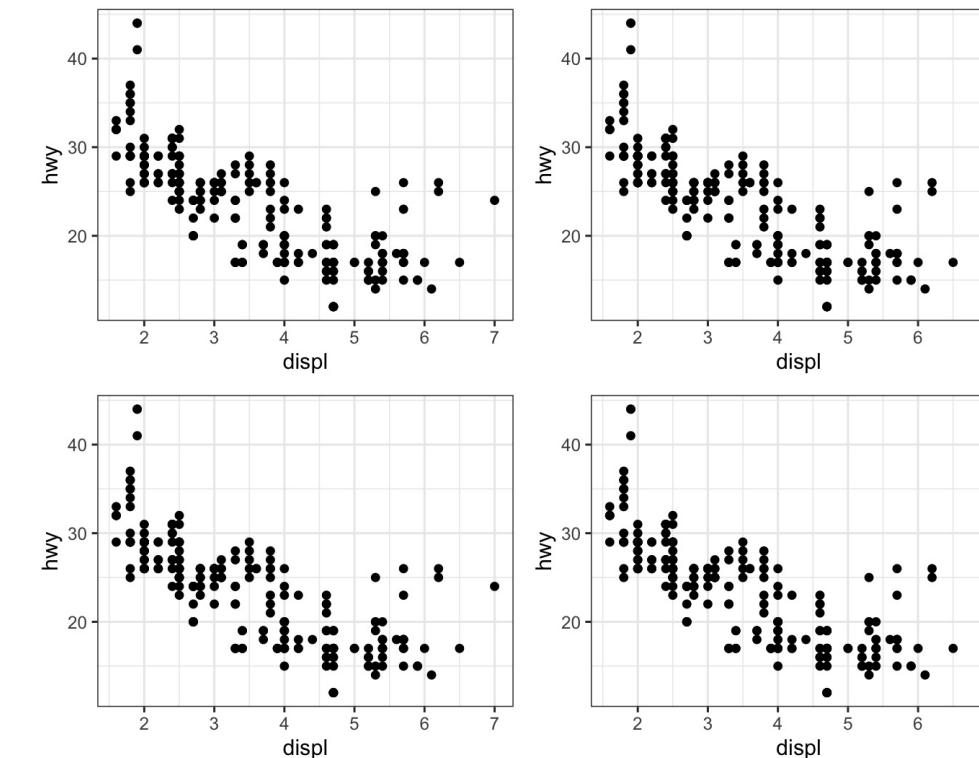
1 Das patchwork Paket liefert eine einfache Syntax um Plots zu verknüpfen.

2 patchwork Syntax:

- + | generisch zusammefügen
- | | nebeneinander stellen
- / | untereinander stellen
- () | zusammenfassen
- & | auf alle anwenden

```
# Speichere plots
schoen <- mein_plot
unschoen <- mein_plot + mein_theme
```

(schoen+schoen) / (schoen+schoen)



# Multiple Plots

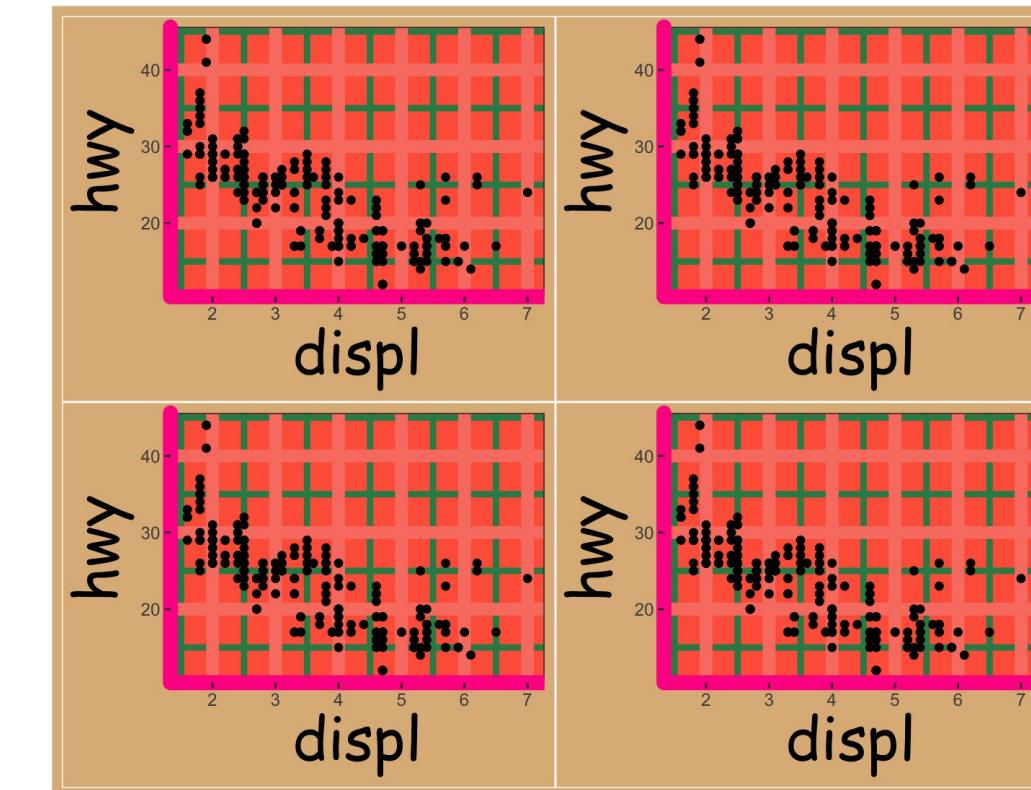
1 Das patchwork Paket liefert eine einfache Syntax um Plots zu verknüpfen.

2 patchwork Syntax:

- + | generisch zusammefügen
- | | nebeneinander stellen
- / | untereinander stellen
- () | zusammenfassen
- & | auf alle anwenden

```
# Speichere plots
schoen <- mein_plot
unschoen <- mein_plot + mein_theme
```

(schoen+schoen) / (schoen+schoen) & mein\_theme



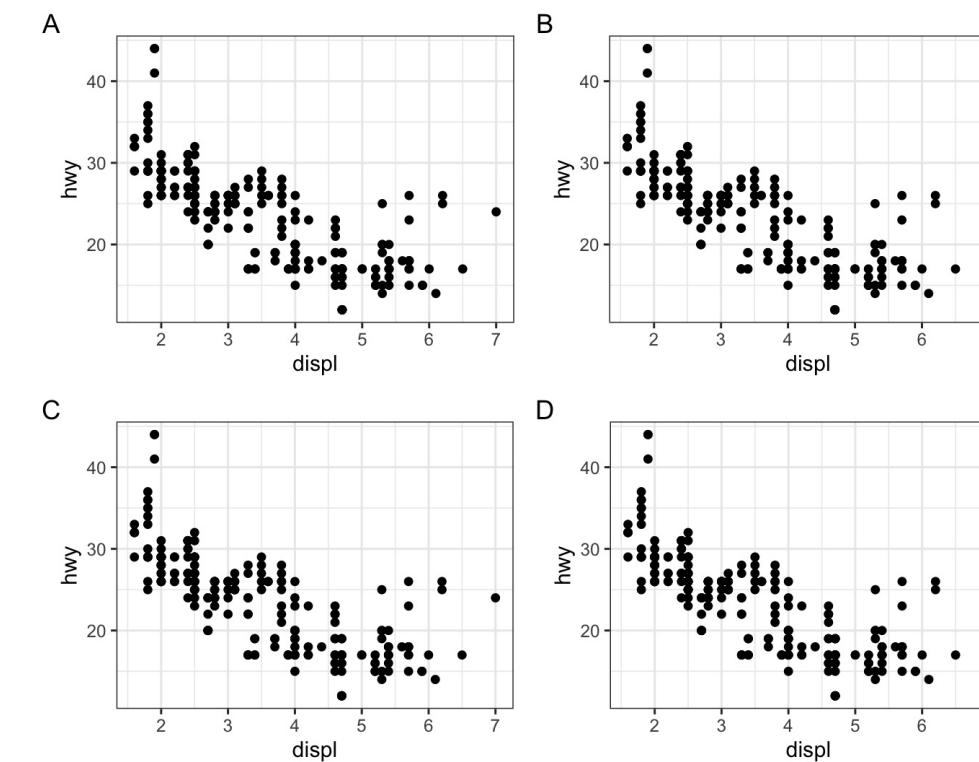
# Multiple Plots

- 1 Das patchwork Paket liefert eine einfache Syntax um Plots zu verknüpfen.
- 2 patchwork Syntax:

- + | generisch zusammefügen
- | | nebeneinander stellen
- / | untereinander stellen
- () | zusammenfassen
- & | auf alle anwenden

```
# Speichere plots
schoen <- mein_plot
unschoen <- mein_plot + mein_theme
```

```
(schoen+schoen) / (schoen+schoen) +
  plot_annotation(tag_levels = "A") &
  theme(legend.position = "none")
```



# ggsave()

- 1 Kreiere **Bilddateien** mit ggsave().
- 2 ggsave Argumente:
  - o filename | Dateipfad
  - o device | z.B. ".pdf" oder ".png"
  - o path | Pfad zum Ordner
  - o height, width | Höhe, Breite
  - o unit | Einheit für Höhe, Breite
  - o dpi | Auflösung pro Einheit

```
# Kreiere meinen Plot
mein_plot <- ggplot(data = mpg,
                     aes(x = displ, y = hwy)) +
  geom_point() +
  mytheme

# Kreiere "mein_plot.pdf"
ggsave(filename = "mein_plot",
       plot = mein_plot,
       device = "pdf",
       path = "figures",
       width = 6,
       height = 4)
```

# ggsave()

- 1 Kreiere **Bilddateien** mit ggsave().
- 2 ggsave Argumente:
  - o filename | Dateipfad
  - o device | z.B. ".pdf" oder ".png"
  - o path | Pfad zum Ordner
  - o height, width | Höhe, Breite
  - o unit | Einheit für Höhe, Breite
  - o dpi | Auflösung pro Einheit

```
# Kreiere meinen Plot
mein_plot <- ggplot(data = mpg,
                     aes(x = displ, y = hwy)) +
  geom_point() +
  mytheme

# Kreiere "mein_plot.png"
ggsave(filename = "mein_plot",
       plot = mein_plot,
       device = "png",
       path = "figures",
       width = 6,
       height = 4)
```

# Practical