# L03 – ggplot2

26. April 2021

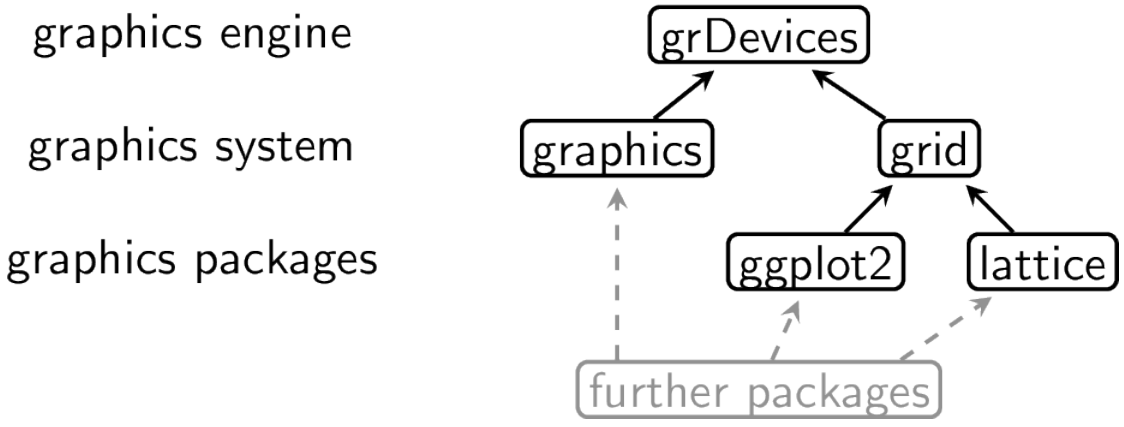
## Contents



## 1 Die Graphics Engine und das Paket ggplot2

Die Grundlage der grafischen Fähigkeiten von R bildet die **graphics engine** im Paket `grDevices`. Sie stellt fundamentale Infrastruktur wie Auswahl von Farbe und Auswahl des Ausgabeformates bereit.

Darauf bauen **graphics systems** wie die Pakete `graphics` oder `grid` auf. Diese stellen Funktionen zum Zeichnen von Objekten bereit.




`graphics` gehört zu Base-R (muss nicht extra geladen werden). Es wird auch mit *base graphics* bezeichnet. Wir haben es mittels der Funktion `plot()` in L01 verwendet.

```
search() # zeige aktuell geladene Pakete
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

Eine alternative zu den Base-Graphics ist das Paket `grid`. Es gehört nicht zu Base-R. Oft wird `grid` nicht direkt benutzt, sondern ein darauf aufbauendes Paket.

Das `lattice`-Paket baut auf `grid` auf und stellt eine Implementierung von sogenannten *Trellis plots* bereit.

Das Paket `ggplot2` baut auf `grid` auf und stellt eine Implementierung der sogenannten *Grammar of Graphics* bereit.

`graphics`, `lattice` und `ggplot2` bilden die wichtigsten grundlegenden Grafik-Pakete. Es gibt viele weitere Pakete, die den Funktionsumfang um spezielle Fähigkeiten erweitern, wie zB das Paket `ggmap` zum Plotten von Landkarten.

Hier beschäftigen wir uns mit `ggplot2`.

```
library(ggplot2)
library(tibble) # brauchen wir für Datensätze
```

## 1.1 Ein erstes Beispiel

Zur Illustration verwenden wir den Datensatz `ggplot2::mpg`. Dieser enthält verschiedene Eigenschaften von 234 Automodellen.

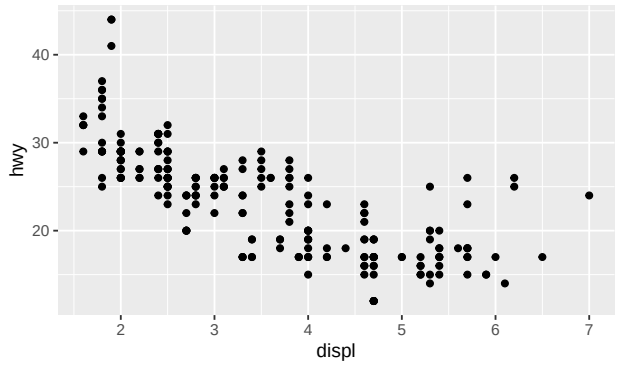
Die Eigenschaften sind unter anderem:

- `displ` (`double`) Hubraum in Liter (zwischen 1.6 und 7)
- `hwy` (`integer`) Verbrauch auf Highway in miles per gallon (12 - 44)
- `class` (`character`) Typ / Fahrzeugklasse (“compact”, “midsize”, “suv”, “2seater”, “minivan”, “pickup”, “subcompact”)

```
mpg
## # A tibble: 234 x 11
##    manufacturer model    displ  year   cyl trans   drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
##  1 audi         a4         1.8  1999     4 auto(l~ f        18    29 p     comp~
##  2 audi         a4         1.8  1999     4 manual~ f        21    29 p     comp~
##  3 audi         a4         2    2008     4 manual~ f        20    31 p     comp~
##  4 audi         a4         2    2008     4 auto(a~ f        21    30 p     comp~
##  5 audi         a4         2.8  1999     6 auto(l~ f        16    26 p     comp~
##  6 audi         a4         2.8  1999     6 manual~ f        18    26 p     comp~
##  7 audi         a4         3.1  2008     6 auto(a~ f        18    27 p     comp~
##  8 audi         a4 quat~   1.8  1999     4 manual~ 4        18    26 p     comp~
##  9 audi         a4 quat~   1.8  1999     4 auto(l~ 4        16    25 p     comp~
## 10 audi         a4 quat~   2    2008     4 manual~ 4        20    28 p     comp~
## # ... with 224 more rows
```

Als erstes Beispiel plotten wir Hubraum gegen Highway-Verbrauch der verschiedenen Modelle.

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



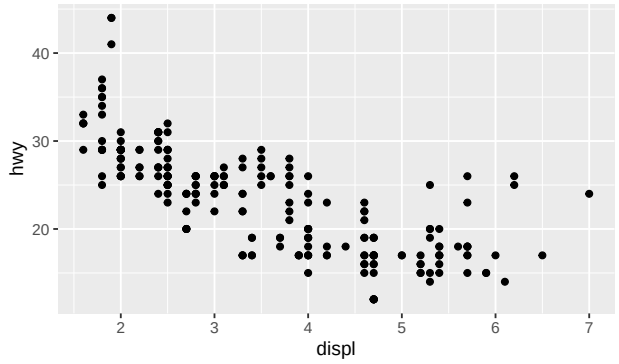

`ggplot()` erzeugt ein `ggplot`-Objekt. Das Argument `data` gibt an, dass wir Informationen aus dem Tibble `mpg` anzeigen wollen.

`geom_point()` erzeugt eine Zeichenebene (`Layer`), in die an gegebene (x,y)-Koordinaten Punkte gezeichnet werden. Daraus ergibt sich ein sogenannter Scatterplot.

Das Argument `mapping` beschreibt, welche Daten genau zur Bestimmung der ästhetischen Eigenschaften (x-Position, y-Position, Farbe, ...) — den sogenannten **aesthetics** — dienen.

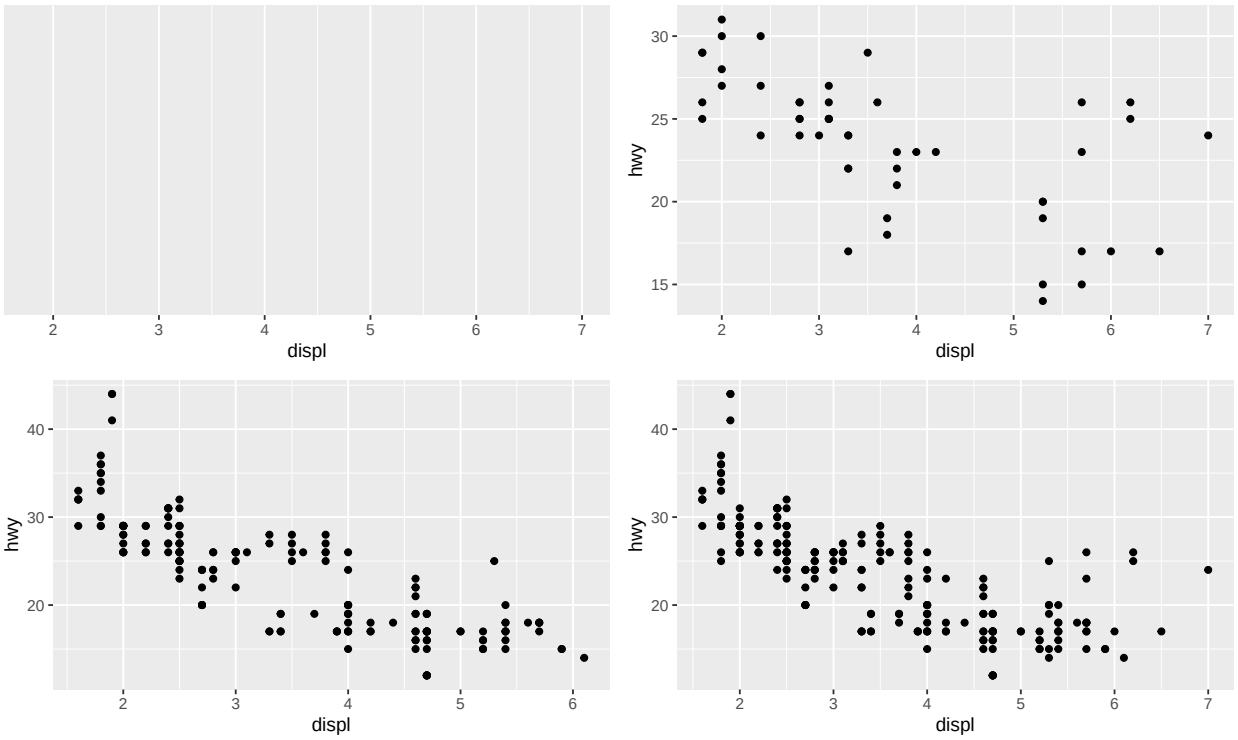
Das `ggplot`-Objekt und das Layer werden mit `+` verknüpft (zu einem neuen `ggplot`-Objekt). Dieses Objekt beschreibt den Plot. Um den Plot anzuzeigen, muss `print()` ausgeführt werden. Bei Eingabe in der Konsole wird `print()` automatisch ausgeführt. Dies steht im Gegensatz zu den Base-Graphics mit `plot()`, `points()`, `lines()`, die nach dem *painters model* erzeugt werden, dh ein grafischer Befehl wird sofort ausgeführt und "übermalt" ältere ggf graphische Ausgaben.

```
plt_base <- ggplot(data = mpg) # zeichnet nicht
lay <- geom_point(mapping = aes(x=displ, y=hwy)) # zeichnet nicht
plt <- plt_base + lay # zeichnet nicht
plt # zeichnet, entspricht print(plt)
```




Default-Werte für `mapping` und `data` werden in `ggplot()` angegeben. Jedes Layer kann allerdings diese Argumente überschreiben oder ergänzen.

```
mpg_part1 <- mpg[1:50, ]
mpg_part2 <- mpg[-(1:50), ]
plt <- ggplot(data = mpg_part1, mapping = aes(x=displ)) # data und x aber nicht y angegeben
l1 <- geom_point(mapping = aes(y=hwy)) # y setzen
l2 <- geom_point(data = mpg_part2, mapping = aes(y=hwy)) # data ändern und y angeben
# grid.arrange() zur Ausgabe mehrerer Plots in einem Bild
gridExtra::grid.arrange(plt, plt + l1, plt + l2, plt + l1 + l2, nrow=2)
```

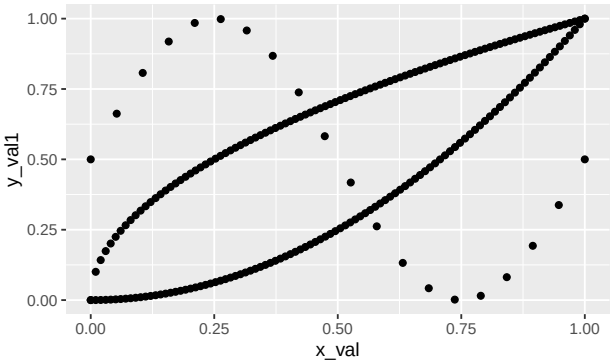



**Aufgabe:**

Wir erzeugen Tibbles mit (x,y)-Koordinaten zum Plotten von drei Funktionen.

```
x_plt1 <- seq(0,1,len=100)
x_plt2 <- seq(0,1,len=20)
plt_data1 <- tibble(
  x_val = x_plt1, # x-Werte für Funktion 1 und 2
  y_val1 = x_plt1^2, # y-Werte für Funktion 1
  y_val2 = sqrt(x_plt1)) # y-Werte für Funktion 2
plt_data2 <- tibble(
  x_val = x_plt2, # x-Werte für Funktion 3
  y_val = sin(2*pi*x_plt2)/2+0.5) # y-Werte für Funktion 3
```

Erzeuge folgenden Scatterplot durch Addieren von drei `geom_point`-Layern auf ein `ggplot`-Objekt. Verändere dabei nicht die Tibbles und erstelle auch keine neuen Tabellen, sondern nutze die Argumente der Plot-Funktionen wie oben gezeigt.




Ersetze nun `geom_point` durch `geom_line`, um einen Linienplot zu erhalten.

Füge nun für die Sinus-Kurve zusätzlich einen `geom_point`-Layer hinzu, sodass auch die Datenpunkte eingezeichnet werden.

**Hinweis:** Ein `ggplot`-Objekt kann mit `ggsave("output.png", plt)` als PNG gespeichert werden.

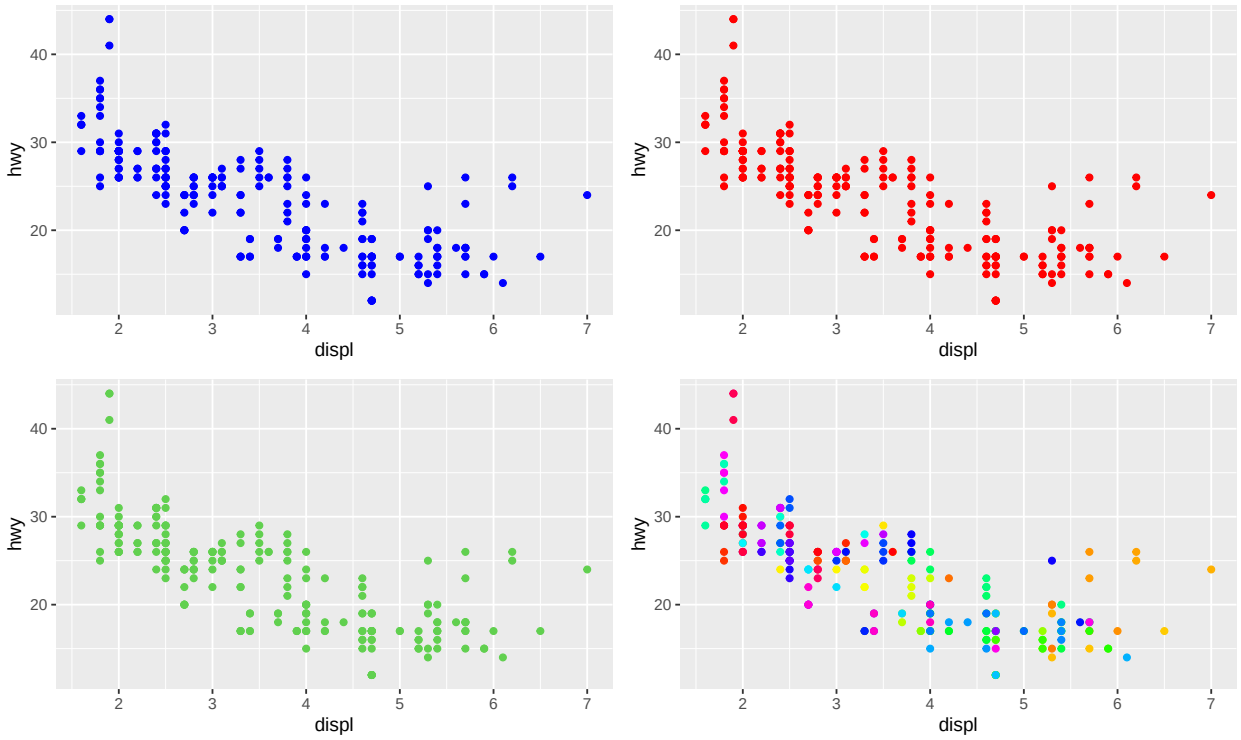
# 2 GeomPoint und GeomLine

## 2.1 Color

Mit dem Argument `color` setzten wir die Farbe. Der Wert ist eine Vektor der Länge 1 oder `nrow(data)`.

Wie bei Base-Graphics gibt es verschiedene Möglichkeiten Farben anzugeben.

```
plt <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy))
l1 <- geom_point(color = "blue")
l2 <- geom_point(color = "#FF0000")
l3 <- geom_point(color = 3)
l4 <- geom_point(color = rainbow(nrow(mpg)))
gridExtra::grid.arrange(plt + l1, plt + l2, plt + l3, plt + l4, nrow=2)
```

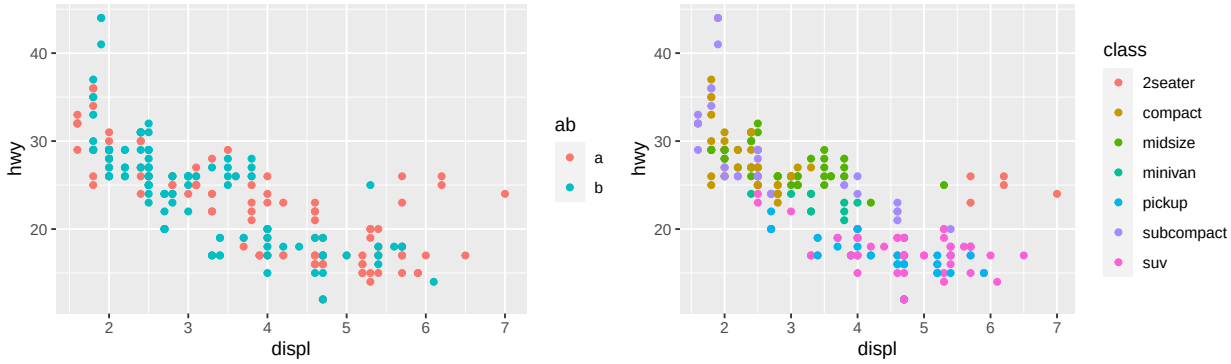



Mit `mapping` wird eine Verbindung zwischen Daten und einer aesthetic hergestellt. Zu den aesthetic zählt neben x- und y-Position auch die Farbe.

Die in `aes()` angegebenen Daten sind Vektoren der Länge 1 oder `nrow(data)`. Dabei können Spaltennamen aus `data` wie Variablen genutzt werden.

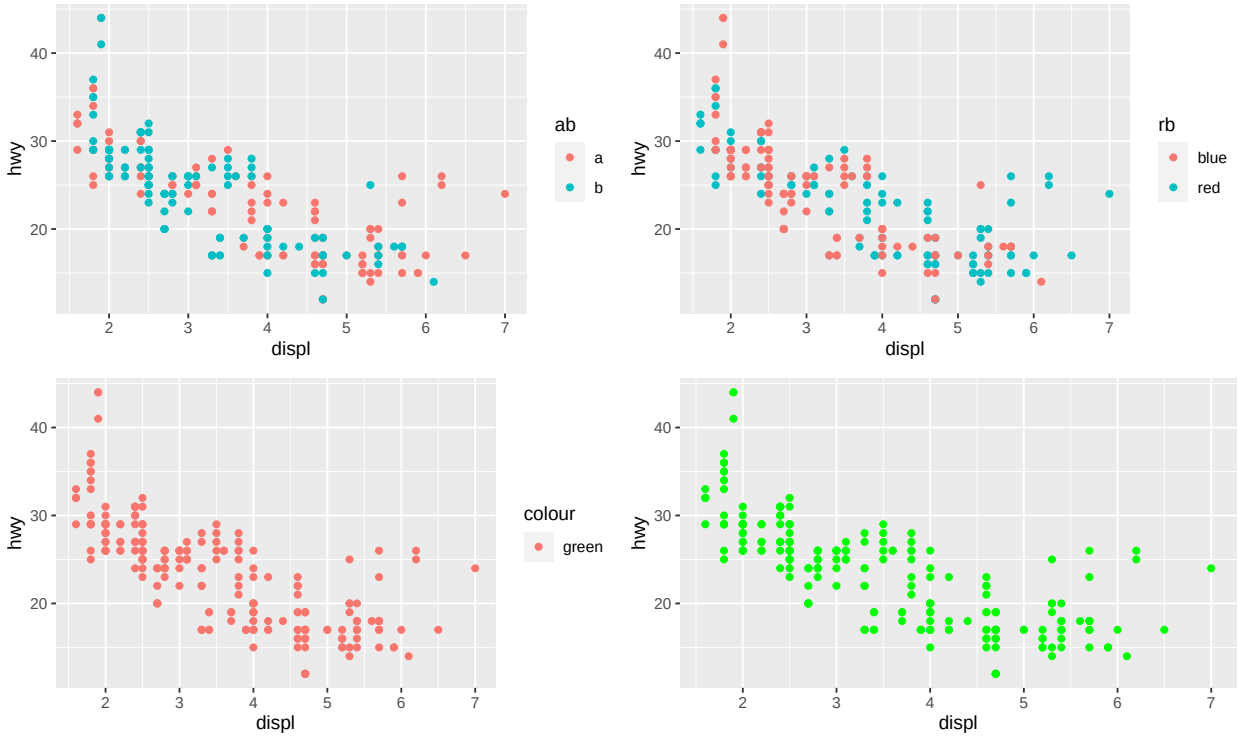
```
ab <- rep(c("a", "b"), each=nrow(mpg)/2)
# ab: externer Vektor
l1 <- geom_point(mapping = aes(color = ab))
# class: Spalte aus data=mpg
l2 <- geom_point(mapping = aes(color = class))
```

```
gridExtra::grid.arrange(plt + l1, plt + l2, nrow=1)
```




Die Daten in `aes()` werden nicht direkt als Farbe interpretiert, sondern erst durch eine Skale (`scale`) in einen konkreten Farbwert übersetzt. Beachte den Unterschied zwischen `geom_point(mapping = aes(color = "green"))` und `geom_point(color = "green")`!

```
ab <- rep(c("a", "b"), each=nrow(mpg)/2)
l1 <- geom_point(mapping = aes(color = ab))
rb <- rep(c("red", "blue"), each=nrow(mpg)/2)
l2 <- geom_point(mapping = aes(color = rb))
l3 <- geom_point(mapping = aes(color = "green"))
l4 <- geom_point(color = "green")
gridExtra::grid.arrange(plt + l1, plt + l2, plt + l3, plt + l4, nrow=2)
```



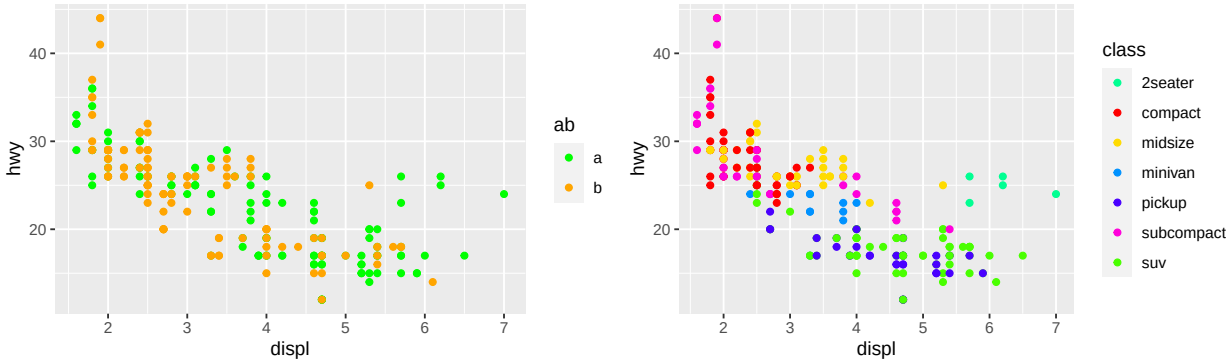

Um eine eigene Skale für Farben zu setzen, nutze `scale_color_manual()` (für diskrete Farbskalen) oder eine der Hilfsfunktionen die mit `scale_color_` beginnen.

Um eine Skale anzuwenden, wird sie auf das `ggplot`-Objekt addiert. Sie gilt dann für alle Layer.

```
ab <- rep(c("a", "b"), each=nrow(mpg)/2)
l1 <- geom_point(mapping = aes(color = ab))
s1 <- scale_color_manual(values = c(a="green", b="orange"))
# Daten-Wert "a" wird zu Farbwert "green"
# Daten-Wert "b" wird zu Farbwert "orange"

l2 <- geom_point(mapping = aes(color = class))
cols <- rainbow(7)
names(cols) <- unique(mpg$class)
cols
##    compact    midsize        suv    2seater    minivan     pickup subcompact
##  "#FF0000"  "#FFDB00"  "#49FF00"  "#00FF92"  "#0092FF"  "#4900FF"  "#FF00DB"
s2 <- scale_color_manual(values = cols)

gridExtra::grid.arrange(plt + l1 + s1, plt + l2 + s2, nrow=1)
```



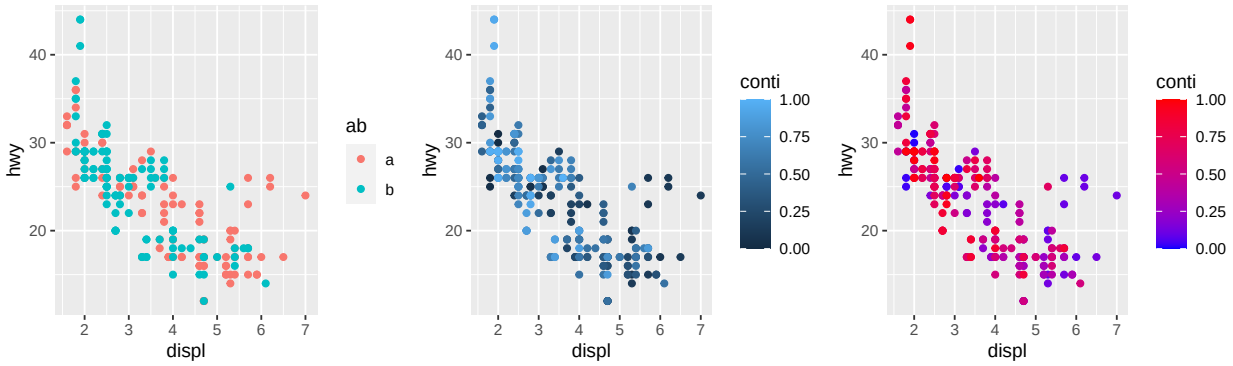

Wird keine Skale angegeben, wird abhängig vom Typ der ge*map*ten color-Daten eine diskrete oder kontinuierliche Farbskala gewählt. Außerdem werden die konkreten Übersetzungen zwischen Daten- und Farbwert automatisch gewählt.

```
# character -> diskrete scale
l1 <- geom_point(mapping = aes(color = ab))

# double -> kontinuierliche scale
conti <- seq(0, 1, len=nrow(mpg))
l2 <- geom_point(mapping = aes(color = conti))

# manual continuous scale
s <- scale_color_gradient(low = "blue", high = "red")

gridExtra::grid.arrange(plt + l1, plt + l2, plt + l2 + s, nrow=1)
```



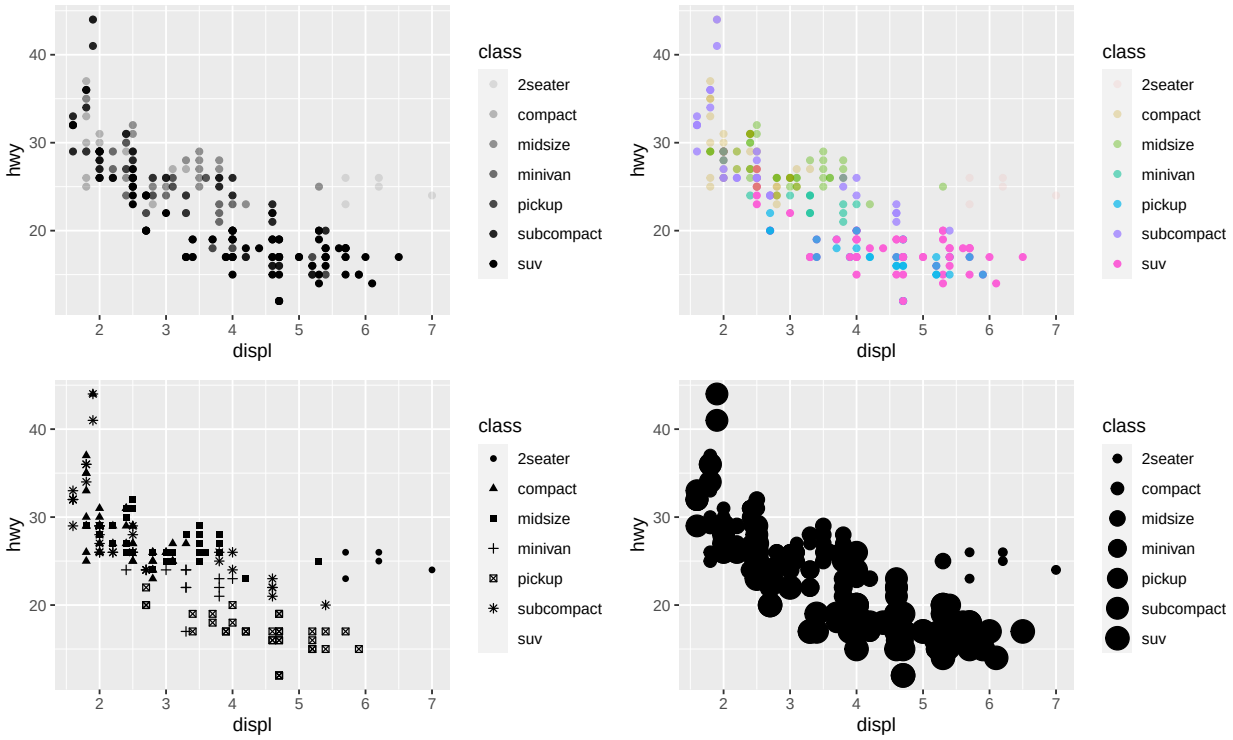

## 2.2 Weitere aesthetics

`geom_point()` hat unter anderem folgende aesthetics:

- **x** (notwendig)
- **y** (notwendig)
- alpha: Transparenz (Default: keine Transparenz)
- color: (Default: schwarz)
- shape: (Default: Kreis)
- size

```
l1 <- geom_point(mapping = aes(alpha = class))
l2 <- geom_point(mapping = aes(color = class, alpha = class)) # können kombiniert werden
l3 <- geom_point(mapping = aes(shape = class))
l4 <- geom_point(mapping = aes(size = class))
gridExtra::grid.arrange(plt + l1, plt + l2, plt + l3, plt + l4, nrow=2)
## Warning: Using alpha for a discrete variable is not advised.

## Warning: Using alpha for a discrete variable is not advised.
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
## Warning: Removed 62 rows containing missing values (geom_point).
## Warning: Using size for a discrete variable is not advised.
```

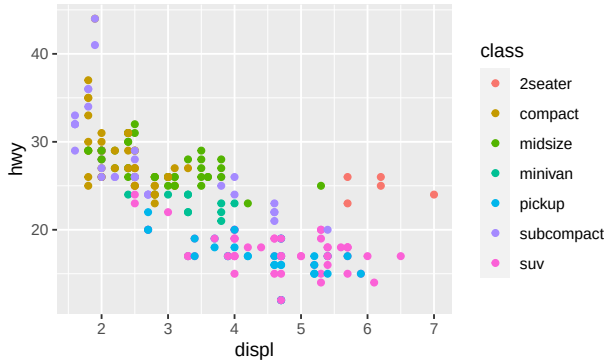



Wie oben bei `color` gesehen, werden **aesthetics** als Argumente der `geom_`-Funktion oder in `aes()` als `mapping` angegeben. Die Werte sind Vektoren der Länge 1 oder `nrow(data)`. In `mapping` können auch Spaltennamen aus `data` stehen. Sowohl für `data` als auch für `mapping` können Default-Werte in `ggplot()` angegeben werden, welche in einzelnen Layern ggf überschrieben werden.

Wir können Plot-Aufrufe etwas kürzer schreiben, wenn wir die Positionen der Argumente der eingesetzten Funktionen kennen.

- Die ersten beiden Argumente der Funktion `ggplot()` sind `data` und `mapping`.
- Die ersten beiden Argumente der Funktion `aes()` sind `x` und `y`.
- Das erste Argument einer `geom_`-Funktion ist `mapping`.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class))
```




**Aufgabe:**

```
x_plt <- seq(0, 1, len=7)
tb <- tibble(
  x_val = rep(x_plt, times=3),
```

```
  y_val = c(sin(2*pi*x_plt)/2+0.5, x_plt^2, sqrt(x_plt)),
  fun = rep(c("sin", "sqr", "sqrt"), each=length(x_plt)))
tb$z_val = abs(0.5-tb$x_val)+0.1
tb
## # A tibble: 21 x 4
##    x_val  y_val fun   z_val
##    <dbl>  <dbl> <chr> <dbl>
##  1 0     0.5    sin   0.6
##  2 0.167 0.933  sin   0.433
##  3 0.333 0.933  sin   0.267
##  4 0.5   0.5    sin   0.1
##  5 0.667 0.0670 sin   0.267
##  6 0.833 0.0670 sin   0.433
##  7 1     0.5    sin   0.6
##  8 0     0      sqr   0.6
##  9 0.167 0.0278 sqr   0.433
## 10 0.333 0.111  sqr   0.267
## # ... with 11 more rows
```

1. Plotte die drei Funktionen in `tb` (als Linien und Punkte im selben Plot) in unterschiedlicher Farbe durch ein entsprechendes `mapping`. Das `ggplot`-Objekt soll nur zwei Layer haben.
2. Färbe nun alle Punkte schwarz, ohne die Linienfarbe zu ändern. Außerdem soll die Größe der Punkte durch `tb$z_val` bestimmt sein.

`geom_line` hat unter anderen die aesthetics

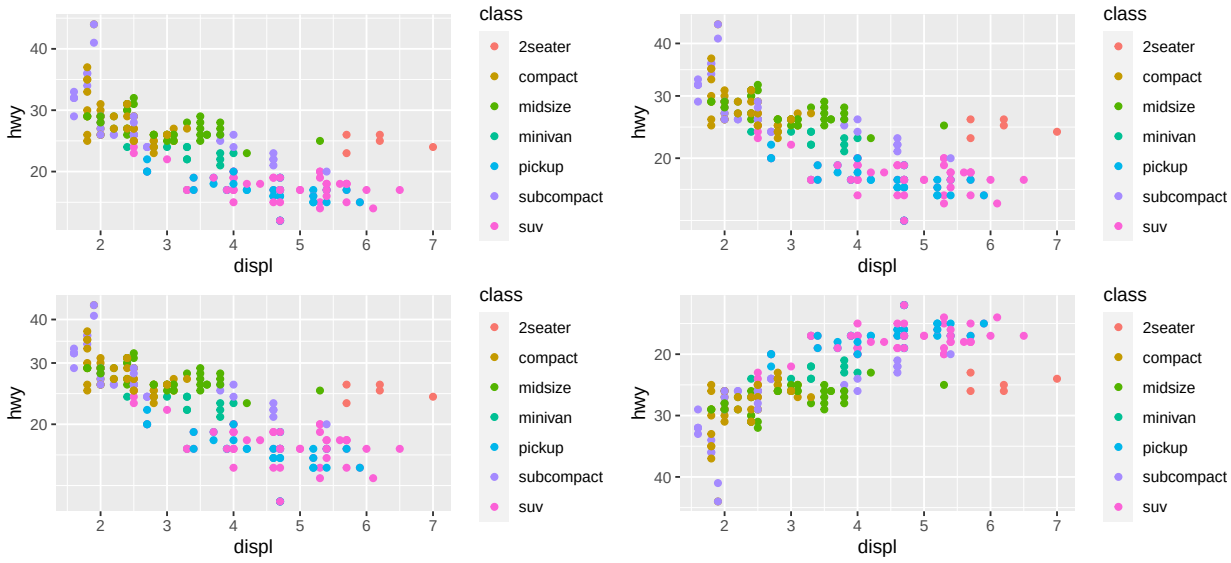
- x
- y
- alpha
- color
- linetype
- size

3. Verbreitere die Dicke der Linien in vorigem Plot und lass jede Funktion mit einem anderen Linientyp anzeigen.

# 3 Nützliches für Skalen

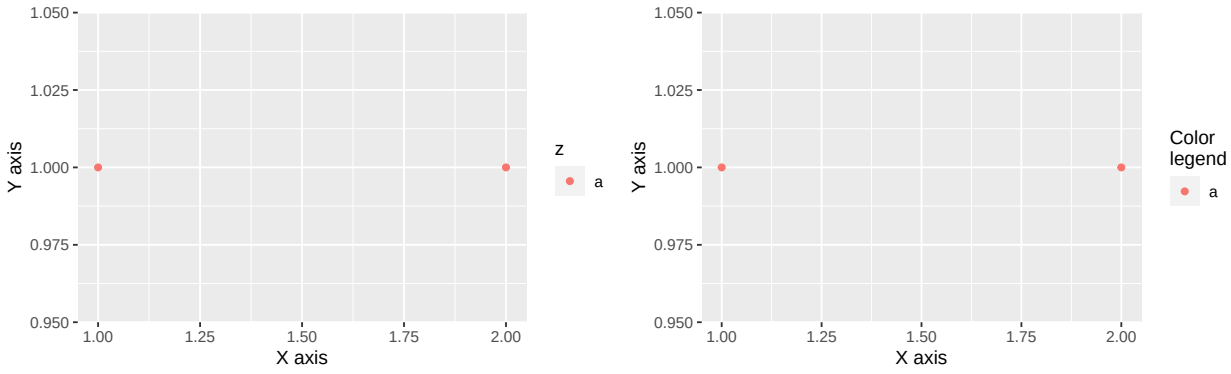
## 3.1 Andere Skalen der Achsen

```
plt <- ggplot(mpg, aes(displ, hwy)) + geom_point(aes(color = class))
p0 <- plt
p1 <- plt +
  scale_y_sqrt()
p2 <- plt +
  scale_y_log10()
p3 <- plt +
  scale_y_reverse()
gridExtra::grid.arrange(p0, p1, p2, p3, nrow=2)
```

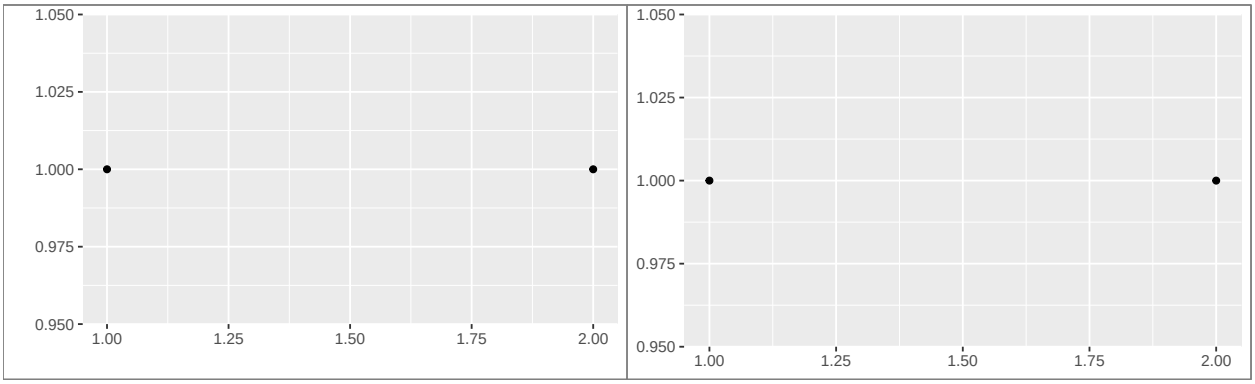



## 3.2 Achsenbeschriftung

```
tb <- tibble(x = 1:2, y = 1, z = "a")
p <- ggplot(tb, aes(x, y)) + geom_point(aes(color = z))
p1 <- p +
  xlab("X axis") +
  ylab("Y axis")
p2 <- p + labs(x = "X axis", y = "Y axis", color = "Color\nlegend")
gridExtra::grid.arrange(p1, p2, nrow=1)
```

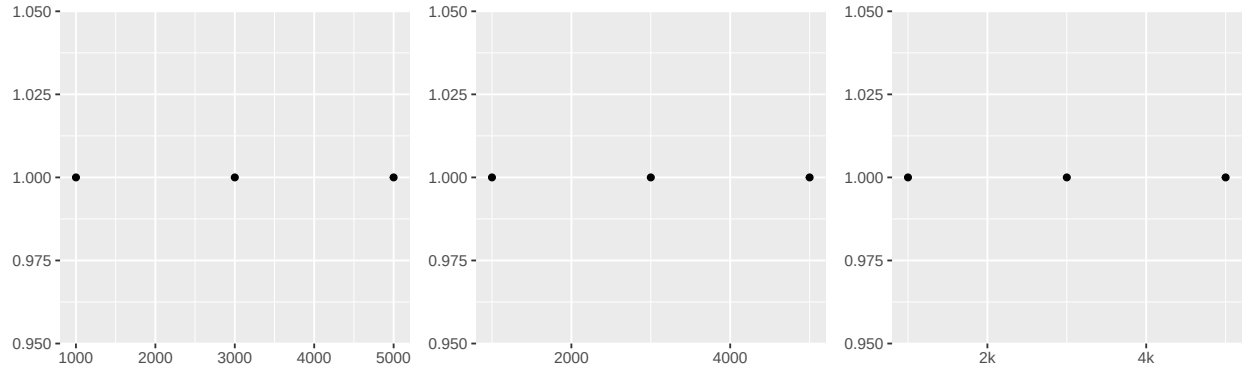



```
p <- ggplot(tb, aes(x, y)) +
  geom_point() +
  theme(plot.background = element_rect(color = "grey50"))
p1 <- p + labs(x = "", y = "")
p2 <- p + labs(x = NULL, y = NULL)
gridExtra::grid.arrange(p1, p2, nrow=1)
```

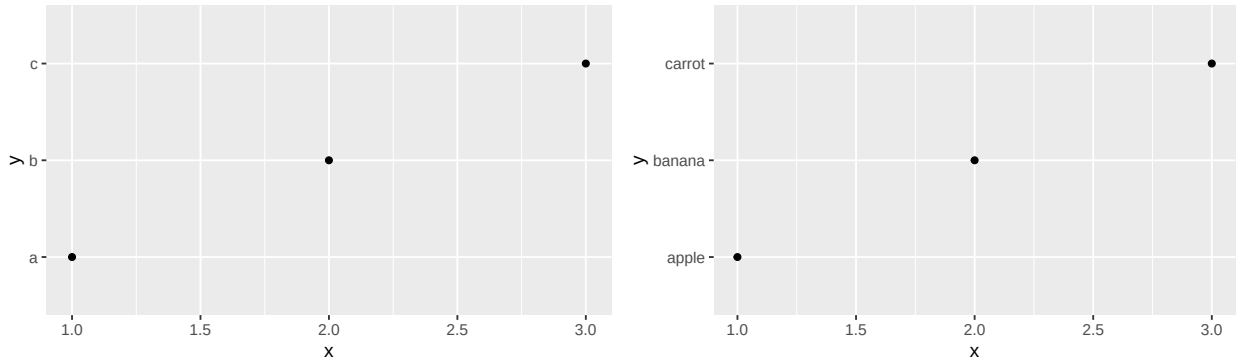



## 3.3 Breaks and Labels

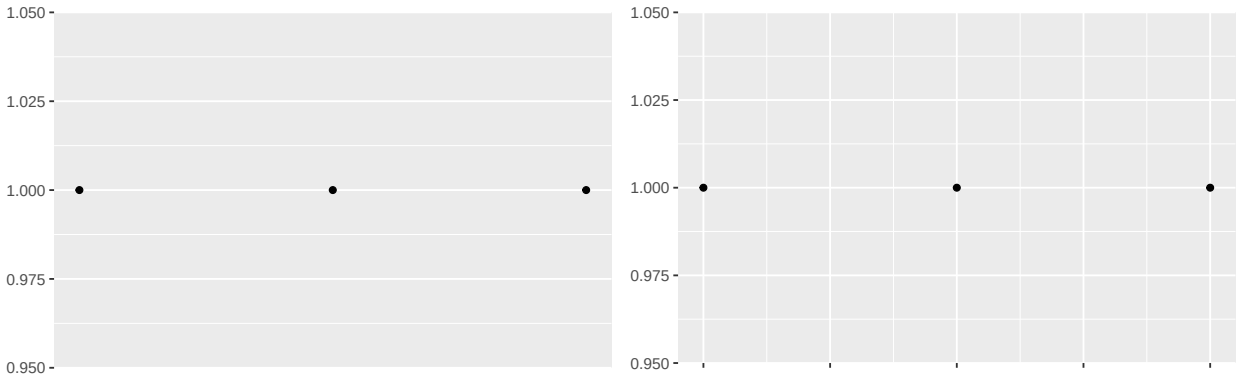
```
tb <- tibble(x = c(1, 3, 5) * 1000, y = 1)
axs <- ggplot(tb, aes(x, y)) +
  geom_point() +
  labs(x = NULL, y = NULL)
p1 <- axs
p2 <- axs + scale_x_continuous(breaks = c(2000, 4000))
p3 <- axs + scale_x_continuous(breaks = c(2000, 4000), labels = c("2k", "4k"))
gridExtra::grid.arrange(p1, p2, p3, nrow=1)
```




```
tb2 <- tibble(x = 1:3, y = c("a", "b", "c"))
p1 <- ggplot(tb2, aes(x, y)) +
  geom_point()
p2 <- ggplot(tb2, aes(x, y)) +
  geom_point() +
  scale_y_discrete(labels = c(a = "apple", b = "banana", c = "carrot"))
gridExtra::grid.arrange(p1, p2, nrow=1)
```

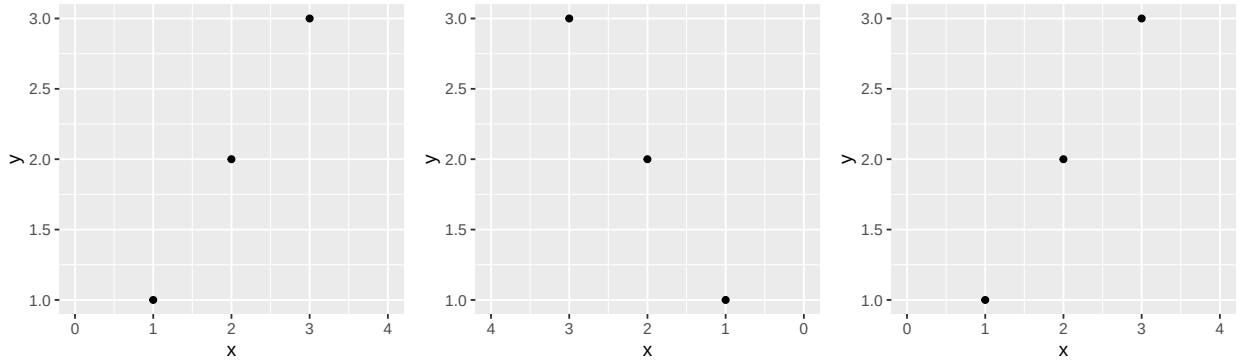



```
p1 <- axs + scale_x_continuous(breaks = NULL)
p2 <- axs + scale_x_continuous(labels = NULL)
gridExtra::grid.arrange(p1, p2, nrow=1)
```



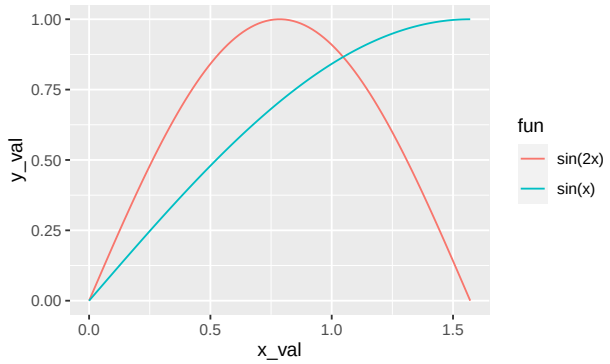

## 3.4 Limits

```
tb <- tibble(x = 1:3, y = 1:3)
base <- ggplot(tb, aes(x, y)) + geom_point()
p1 <- base + xlim(0, 4)
p2 <- base + xlim(4, 0)
p3 <- base + lims(x = c(0, 4))
gridExtra::grid.arrange(p1, p2, p3, nrow=1)
```




**Aufgabe:**

```
x_plt <- seq(0, pi/2, len=100)
tb <- tibble(
  x_val = rep(x_plt, 2),
  y_val = c(sin(x_plt), sin(2*x_plt)),
  fun = rep(c("sin(x)", "sin(2x)"), each=length(x_plt)))
plt <- ggplot(tb, aes(x_val, y_val, color=fun)) +
  geom_line()
plt
```




1. Änder die Achsenbeschriftung auf `x` und `y` ab, und den Titel der Legende auf `function`.
2. Sorge dafür, dass die dickeren Grid-Linien der x-Achse bei $(0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{\pi}{2})$ erscheinen und die dickeren Grid-Linien der y-Achse bei $(\sqrt{\frac{0}{4}}, \sqrt{\frac{1}{4}}, \sqrt{\frac{2}{4}}, \sqrt{\frac{3}{4}}, \sqrt{\frac{4}{4}})$
3. Ändere die Beschriftung der Gitterpositionen in ansprechendere Werte ab, zB `pi/6, pi/4, ...` und `1/2, 1/sqrt(2), ...`.

# 4 Bonus: GeomBar und GeomHistogramm

Es folgen Code-Beispiele zur Erzeugung von Bar-Plots und Histogramme mit `geom_bar()` und `geom_histogramm()`.

Ein Bar-Plot stellt diskrete / kategorische Daten dar. Ein Histogramm stellt kontinuierliche Variablen dar.

Zur Illustration verwenden wir den Datensatz `ggplot2::diamonds`. Dieser enthält verschiedene Eigenschaften von knapp 54000 Diamanten.

Die Eigenschaften sind unter anderen:

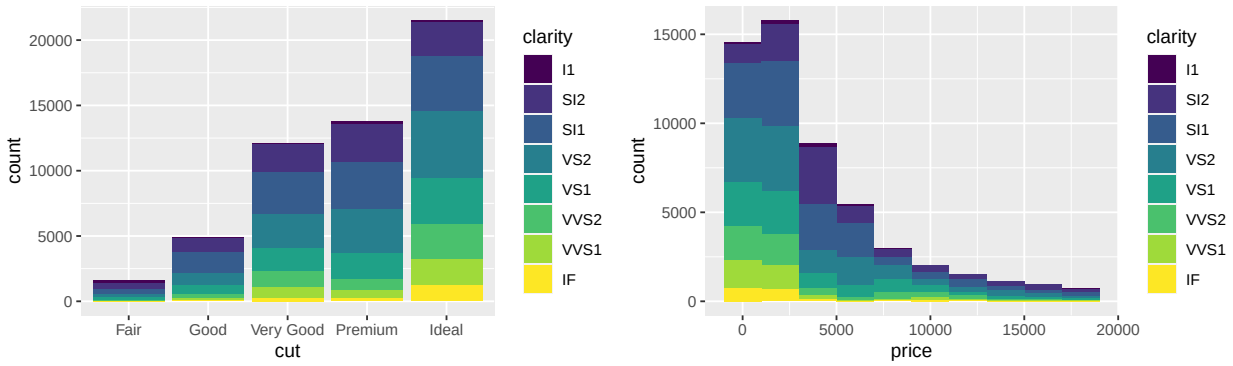
- price (`integer`): Preis in US-Dollar zwischen 326 und 18823
- cut (ordered `factor`): Qualität des Schliffs (Fair, Good, Very Good, Premium, Ideal)
- clarity (ordered `factor`): Reinheit (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

```
diamonds
```

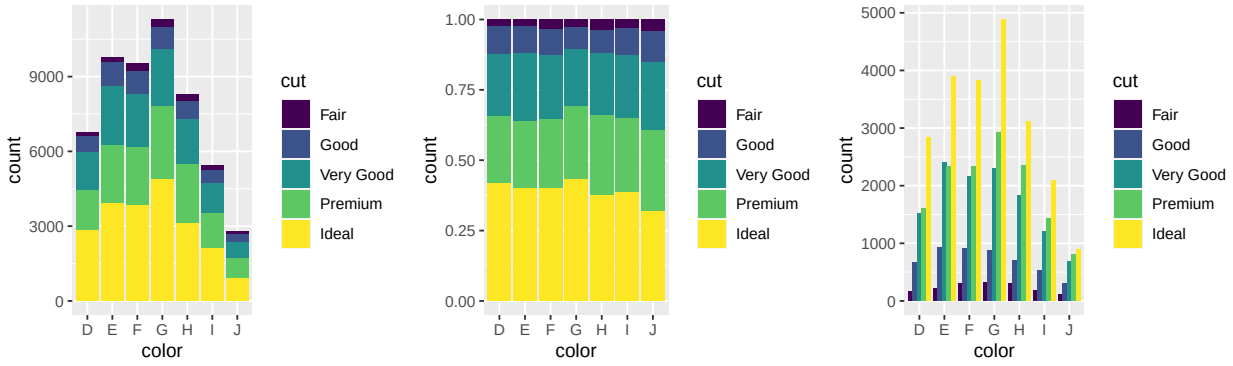
```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
```

```
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```
plt <- ggplot(diamonds, aes(fill=clarity))
l1 <- geom_bar(aes(x=cut))
l2 <- geom_histogram(aes(x=price), binwidth = 2e3)
gridExtra::grid.arrange(plt + l1, plt + l2, nrow=1)
```




```
plt <- ggplot(diamonds, aes(color, fill = cut))
p1 <- plt + geom_bar()
p2 <- plt + geom_bar(position = "fill")
p3 <- plt + geom_bar(position = "dodge")
gridExtra::grid.arrange(p1, p2, p3, nrow=1)
```




**Aufgabe:**

```
# Bundeswahlleiter, Wiesbaden 2017
# Endgültig gewählte Bewerberinnen und Bewerber bei der Wahl zum 19. Deutschen Bundestag (24. September
data_raw <- readr::read_delim("mdb.csv", delim=";")
mdb <- data_raw[,c("Name", "Vorname", "Geschlecht", "Geburtsjahr", "Partei_KurzBez")]
names(mdb) <- c("last_name", "first_name", "gender", "birth_year", "party")
party_colors <- c(
  SPD = "#DF0B25",
  CSU = "#87bbe6",
  CDU = "#000000",
  AfD = "#1A9FDD",
  "DIE LINKE" = "#BC3475",
```

```
  "GRÜNE" = "#4A932B",
  FDP = "#FEEB34"
  )
```

```
mdb
## # A tibble: 709 x 5
##    last_name first_name    gender birth_year party
##    <chr>     <chr>         <chr>       <dbl> <chr>
##  1 Abercron  Michael       m            1952 CDU
##  2 Achelwilm Doris Maria   w            1976 DIE LINKE
##  3 Aggelidis Grigorios     m            1965 FDP
##  4 Akbulut   Gökay         w            1982 DIE LINKE
##  5 Albani    Stephan       m            1968 CDU
##  6 Alt       Renata        w            1965 FDP
##  7 Altenkamp Norbert Maria m            1972 CDU
##  8 Altmaier  Peter         m            1958 CDU
##  9 Amthor    Philipp       m            1992 CDU
## 10 Amtsberg  Luise         w            1984 GRÜNE
## # ... with 699 more rows
```

Das Tibble `mdb` enthält Daten der Mitglieder des Bundestages (Stand 2017, direkt nach der Wahl).

**Hinweis:** Werden Code-Dateien mit Umlauten oder anderen Sonderzeichen nicht richtig angezeigt, kann `File` → `Reopen with Encoding` → `UTF-8` helfen.

1. Erzeuge einen Bar-Plot, der die Geschlechterverhältnisse für jede Partei veranschaulicht.
2. Erzeuge ein Histogramm, das anzeigt in welchem Jahrzehnt die Mitglieder des Bundestags jeweils wurden. Nutze dabei die Farben `party_colors`, um die Parteizugehörigkeit anzuzeigen.

**Hinweis:** Um die `fill`-Skale zu ändern, nutze `scale_fill_manual()` analog zu `scale_color_manual()`.