R Notebook

Contents

Einführung

- ggplot2 basiert auf der Idee der "grammar of graphics"
- Demnach lässt sich jede Grafik aufbauen als Kombination von
  - Daten
  - Koordinatensystem
  - "geoms"
- "geoms" sind dabei visuelle Repräsentationen der Daten (Punkte, Kreise, Flächen, Linien, etc)
- "geoms" haben Eigenschaften (sog. "aesthetics"), z.B. größe, farbe, x- und y-Position etc.
- die Daten werden auf die "aesthetics" gemapped

Datenpräsentation: geom

- Art der visuellen Präsentation der Daten
  - Balken

    - Kreis
    - Linien
    - Punkte
    - Flächen
    - etc.
- kann grob unterteilt werden in:
    - Univariat
        * diskret (faktor)
        * stetig
    - Bivariat
        * x und y diskret
        * x oder y diskret
        * x und y stetig
- Über `aes()` werden hier die Werte für x bzw. y zugewiesen

```
library(ggplot2)
library(car)
```

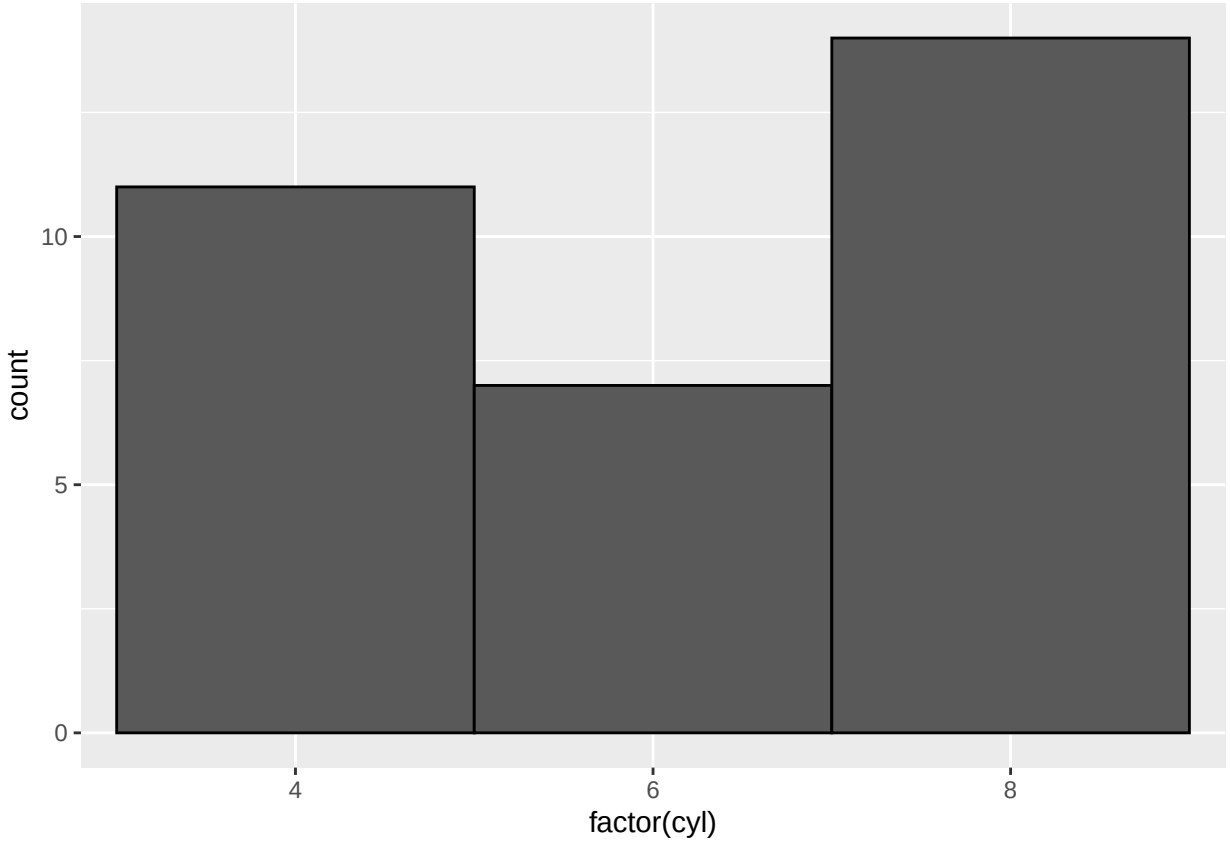
```
## Loading required package: carData
```

```
library(magrittr)
data("mtcars")
data("economics")
```

Univariat:

Bar-Plot

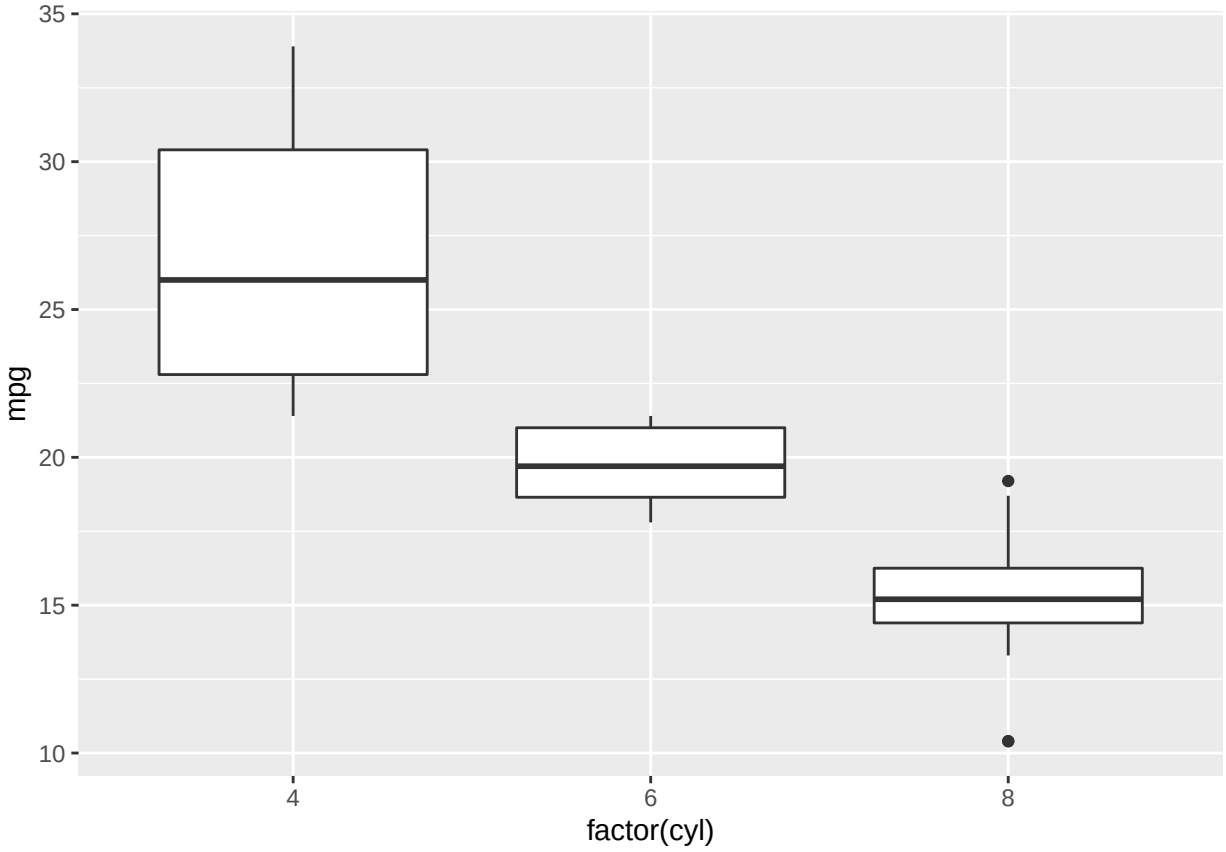
```
plot <- ggplot(mtcars, aes(x = factor(cyl))) +
  geom_bar(width = 1, color = "black")
plot
```



Bivariat:

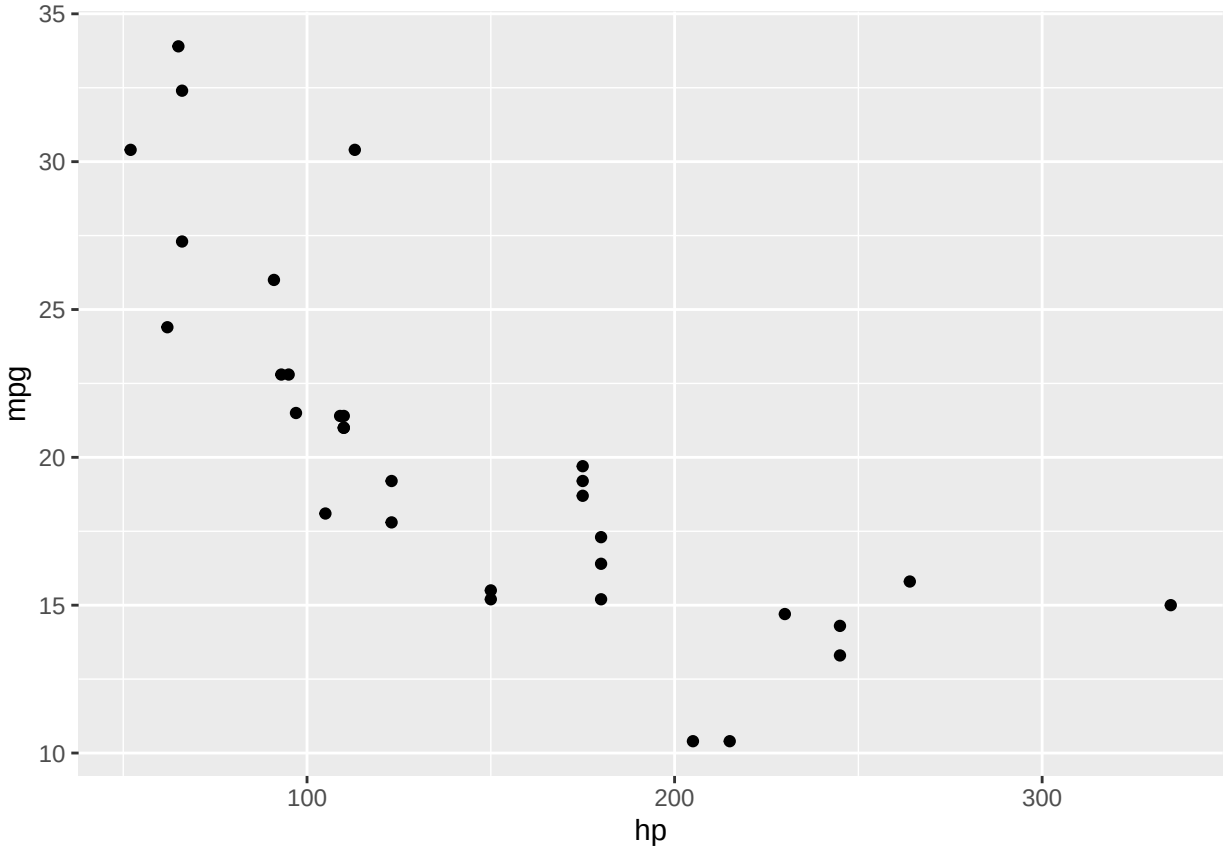
Box-Plot

```
ggplot(data = mtcars) +
  geom_boxplot(aes(x = factor(cyl), y = mpg))
```

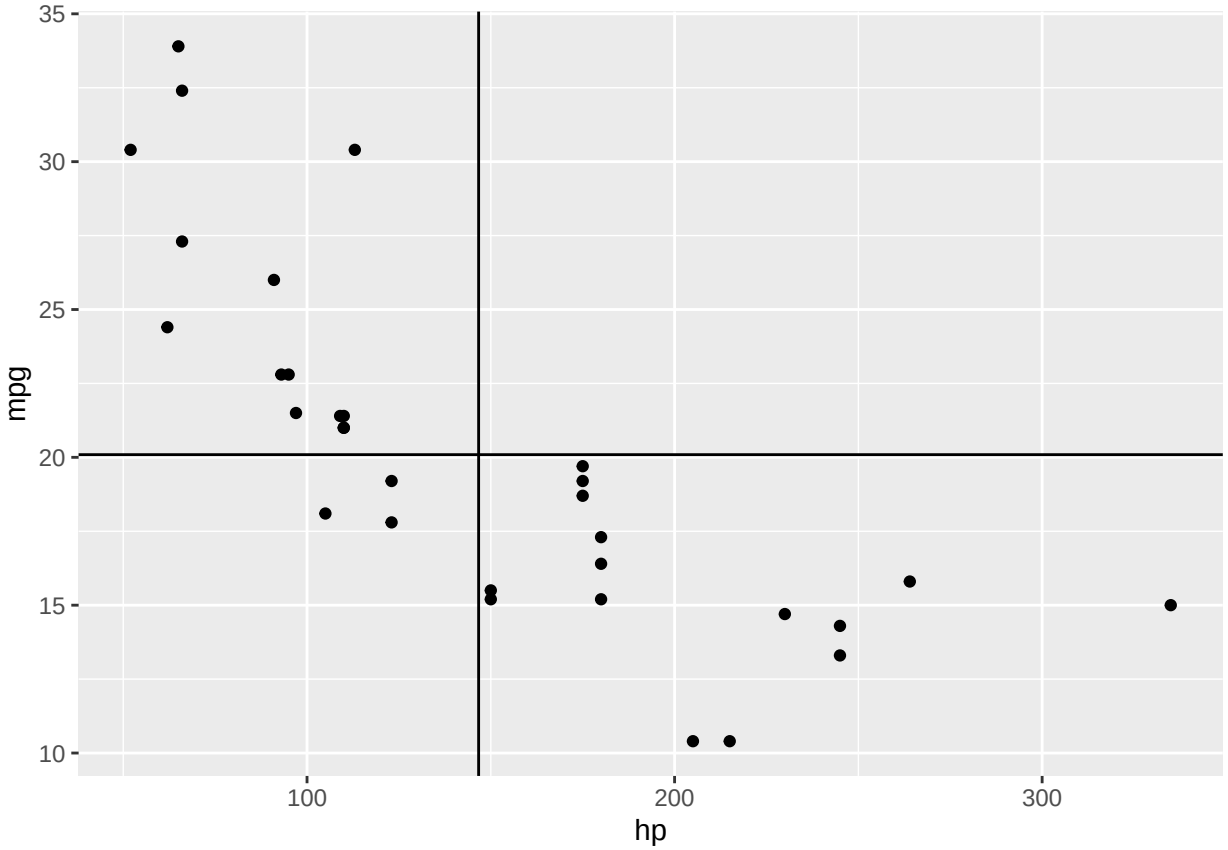


Scatter-Plot

```
plot <- ggplot(data = mtcars) +
  geom_point(aes(x = hp, y = mpg))
plot
```



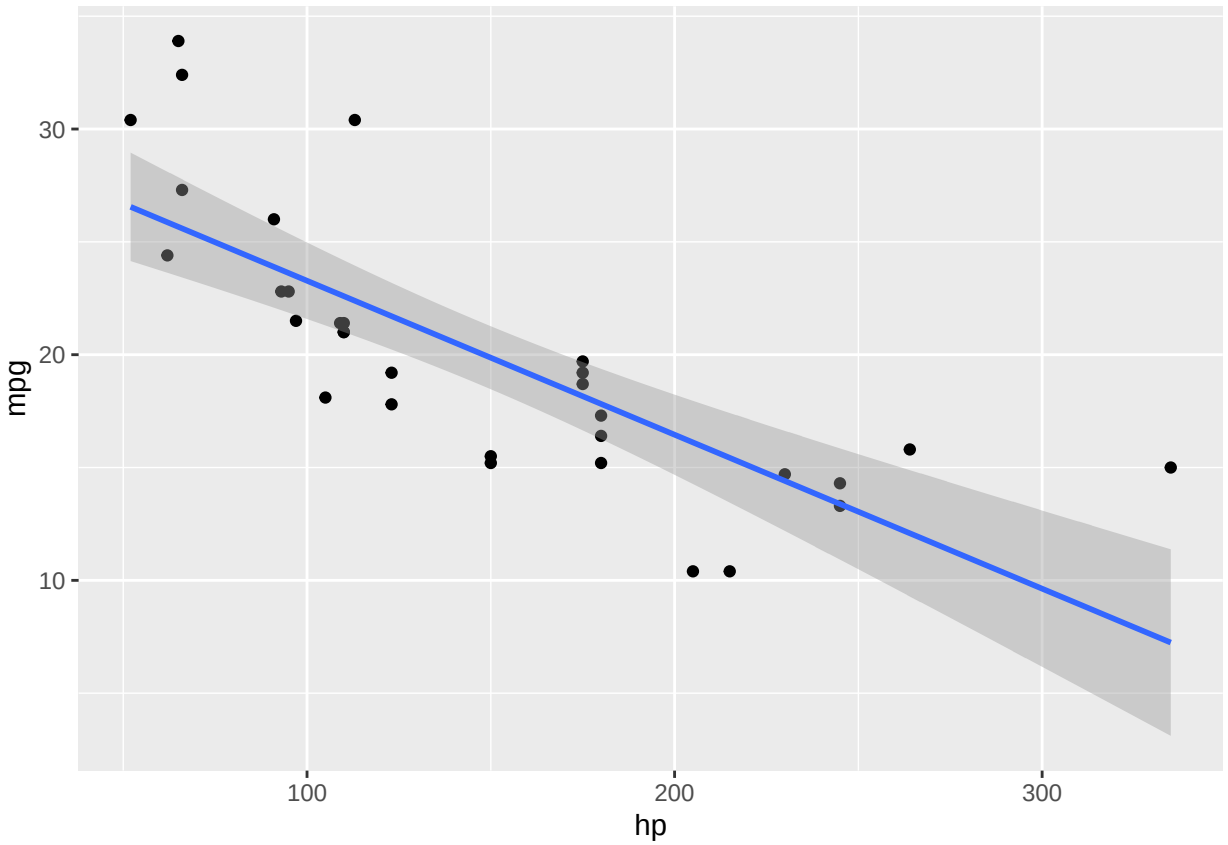

```
plot +
  geom_hline(aes(yintercept = mean(mpg))) +
  geom_vline(aes(xintercept = mean(hp)))
```



Regressions-Plot

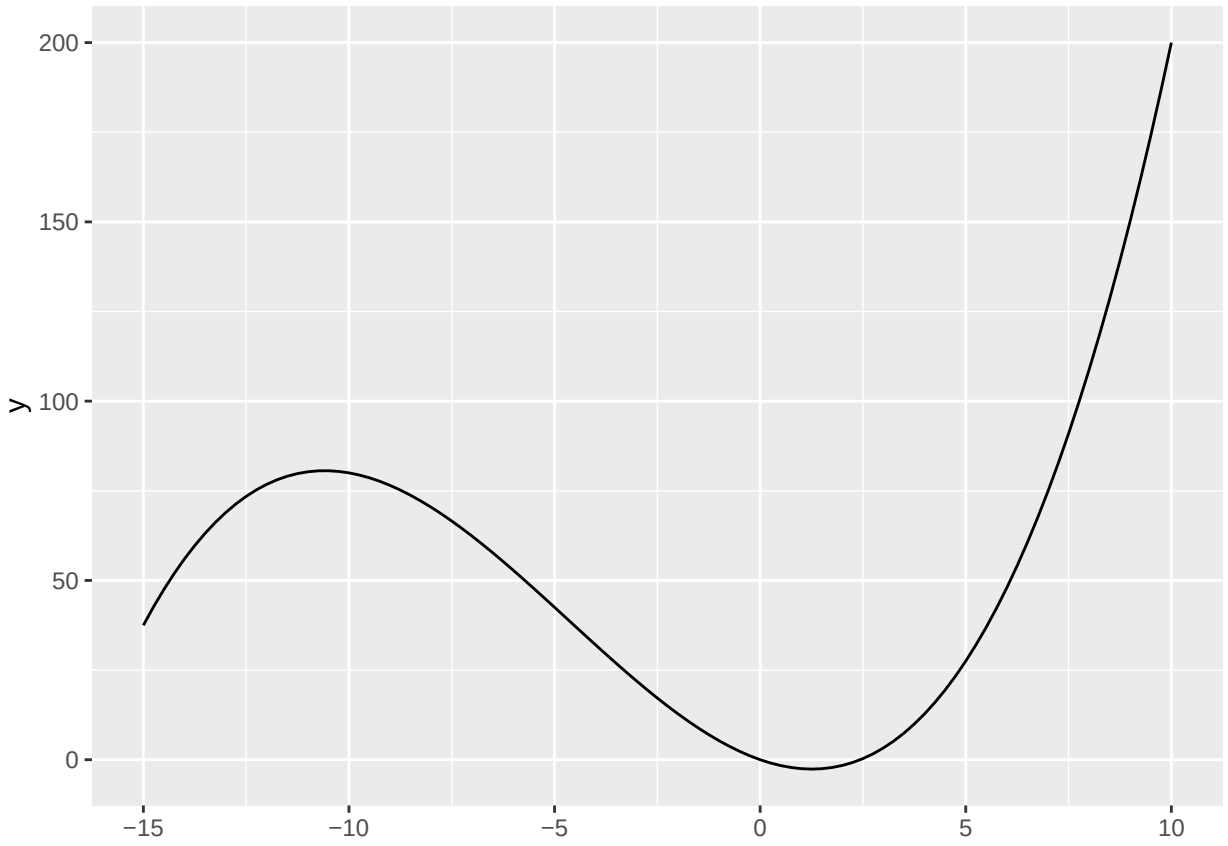
```
plot +
  geom_smooth(method = "lm", aes(x = hp, y = mpg))
```

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



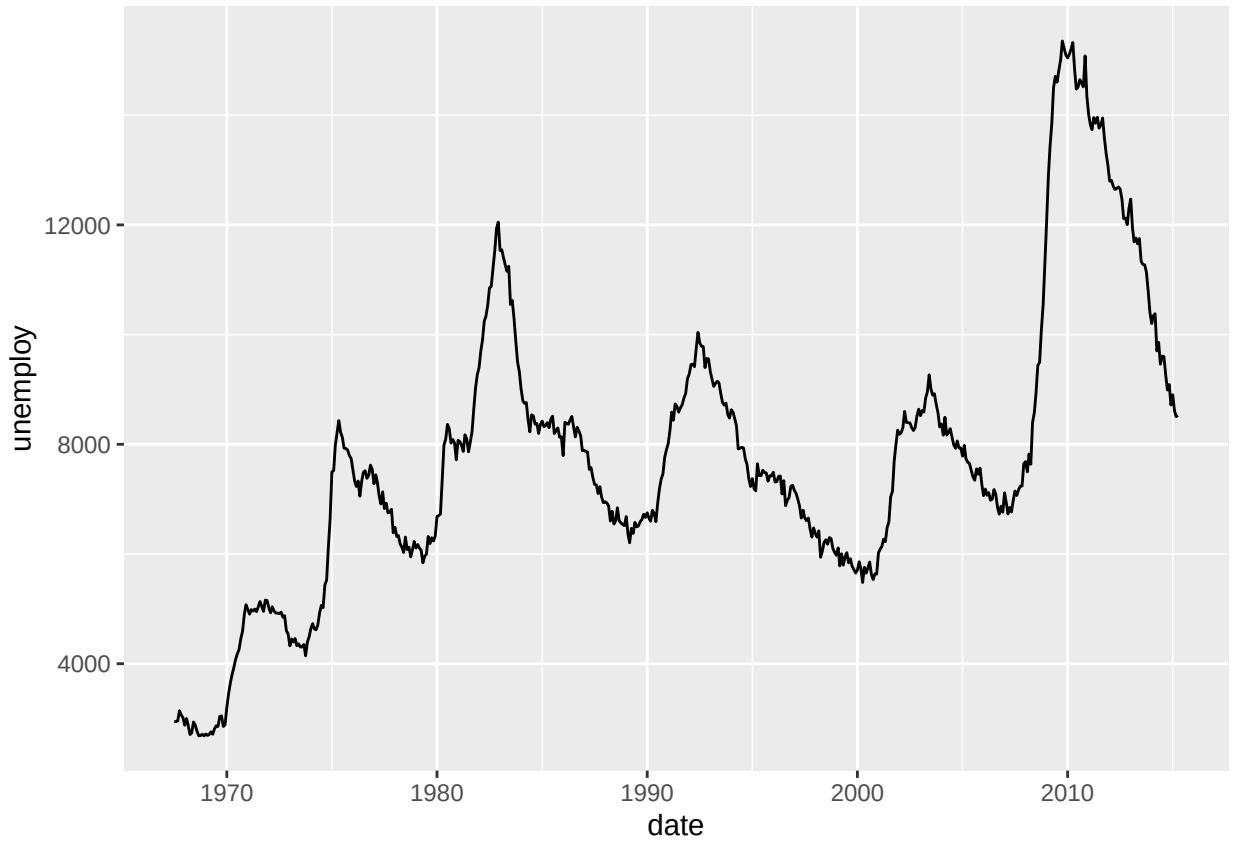
Funktion

```
fun <- function(x) {0.1 * x^3 + 1.4 * x^2 - 4 * x}

ggplot() +
  stat_function(fun = fun, xlim = range(-15, 10))
```

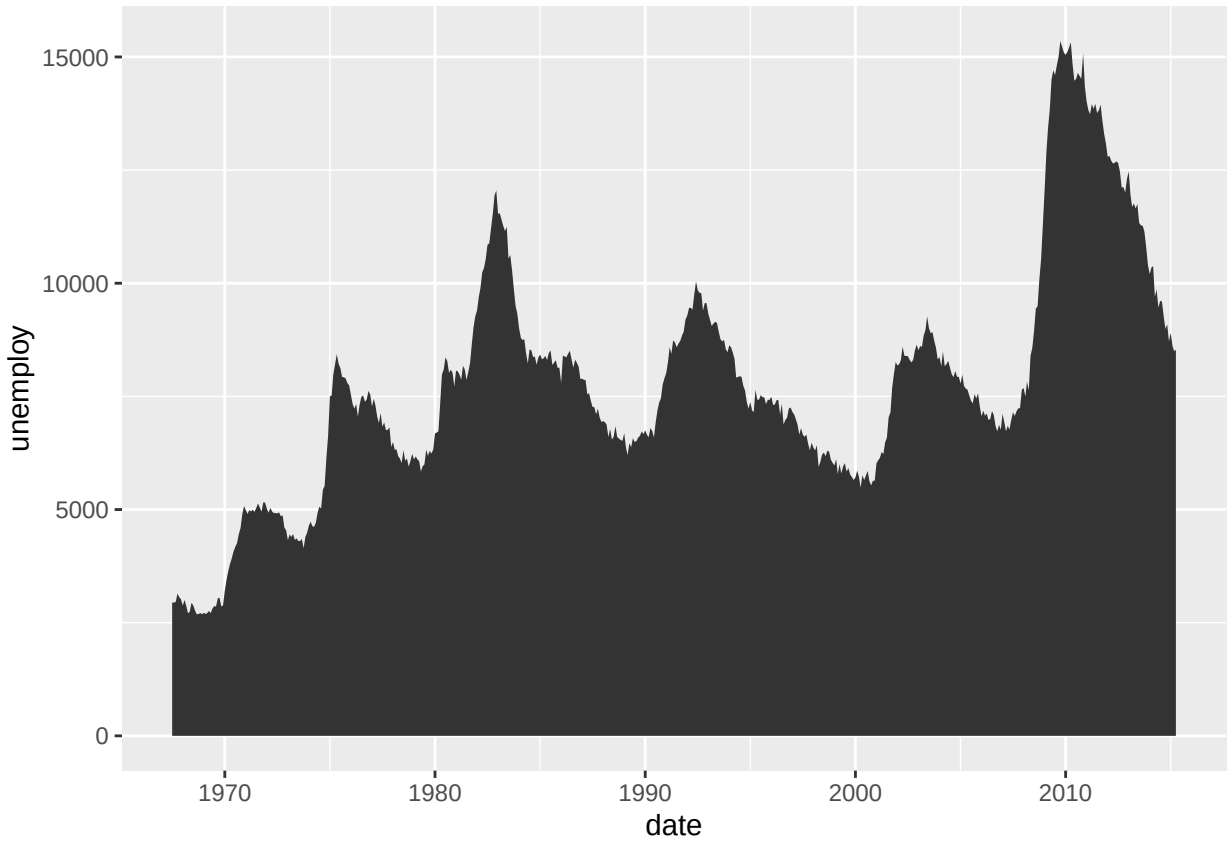


Zeitreihenplots

```
data("economics", package = "ggplot2")

plot <- ggplot(economics, aes(x = date, y = unemploy))
plot +
  geom_line()
```



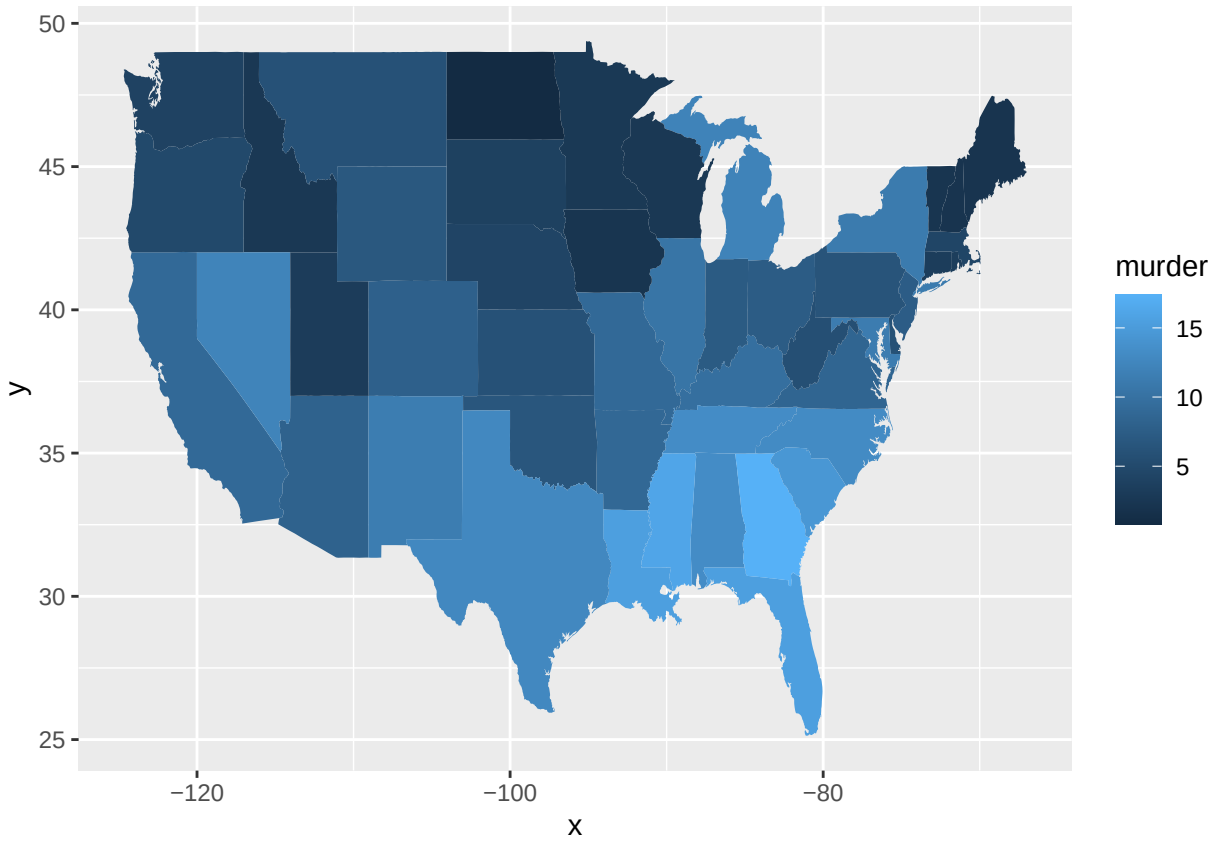

```
plot +
  geom_area()
```



Sonstiges

Maps

```
library(maps)
data("USArrests")
data <- data.frame(murder = USArrests$Murder,
                   state = tolower(rownames(USArrests)) )
map <- map_data("state")
ggplot(data, aes(fill = murder)) +
  geom_map(aes(map_id = state), map = map) +
  expand_limits(x = map$long, y = map$lat)
```

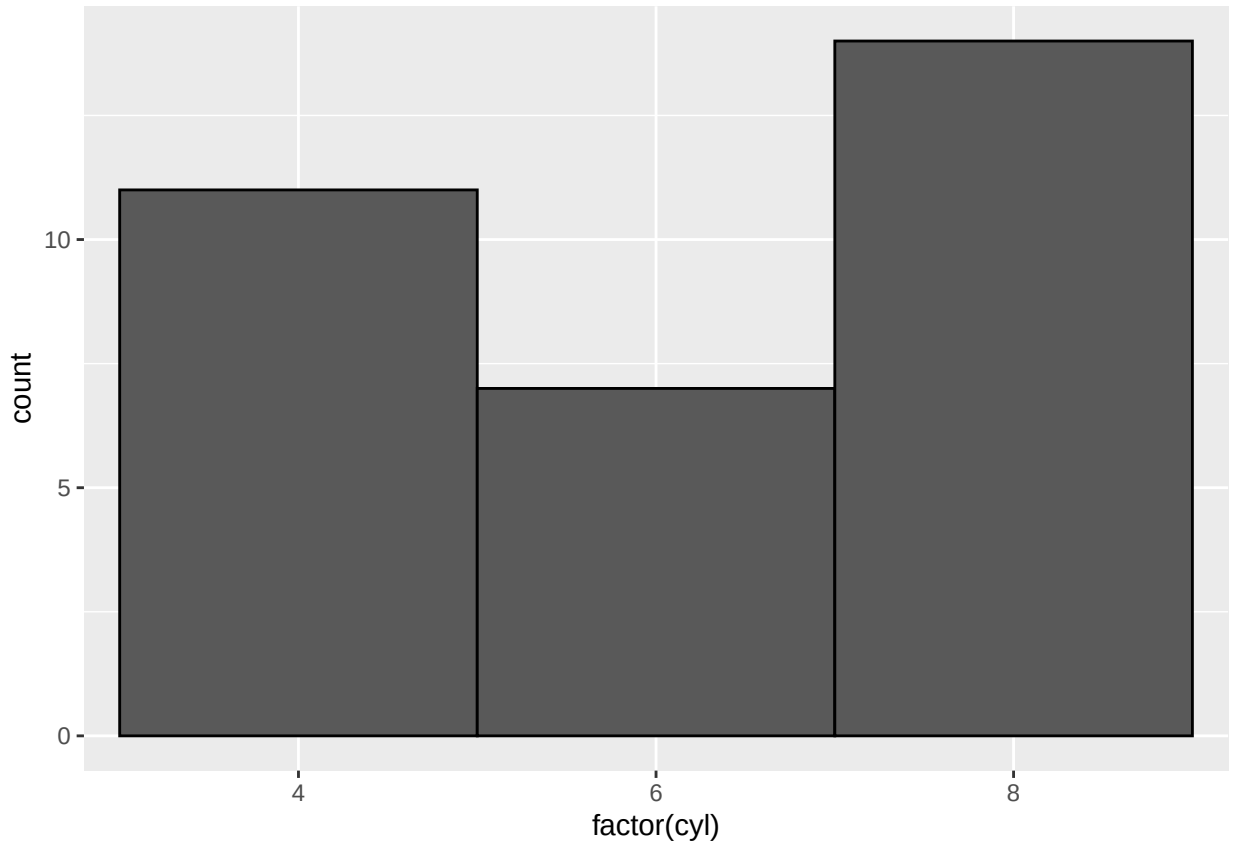


Koordinatensystem: coords und scale

coords

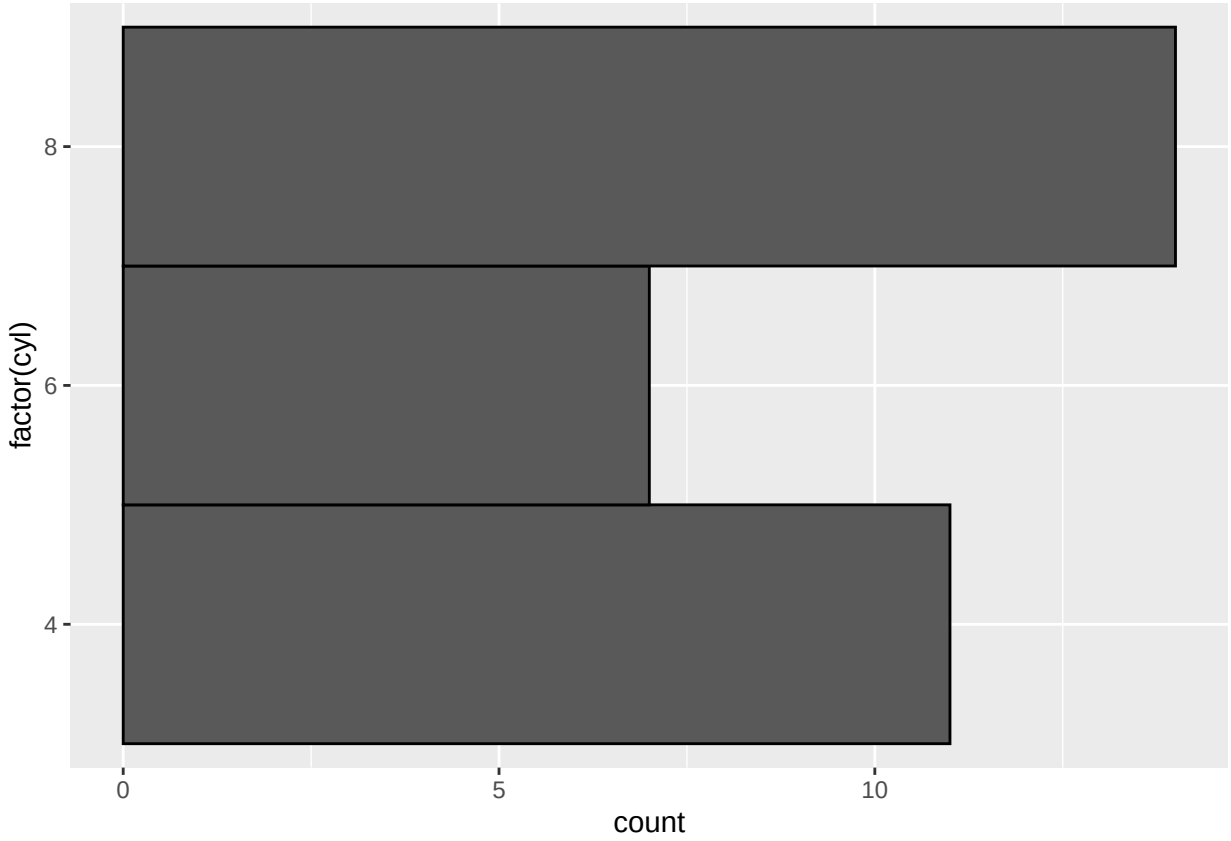
Normal

```
plot <- ggplot(mtcars, aes( x = factor(cyl))) +
  geom_bar(width = 1, color = "black")
plot
```



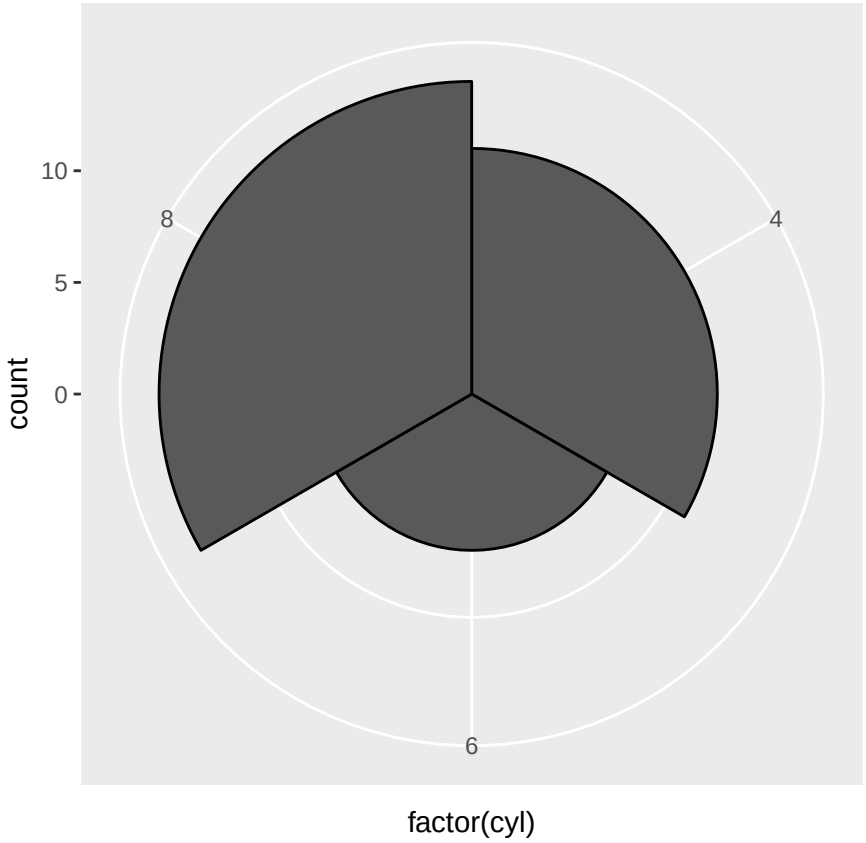
Flip

```
plot +
  coord_flip()
```

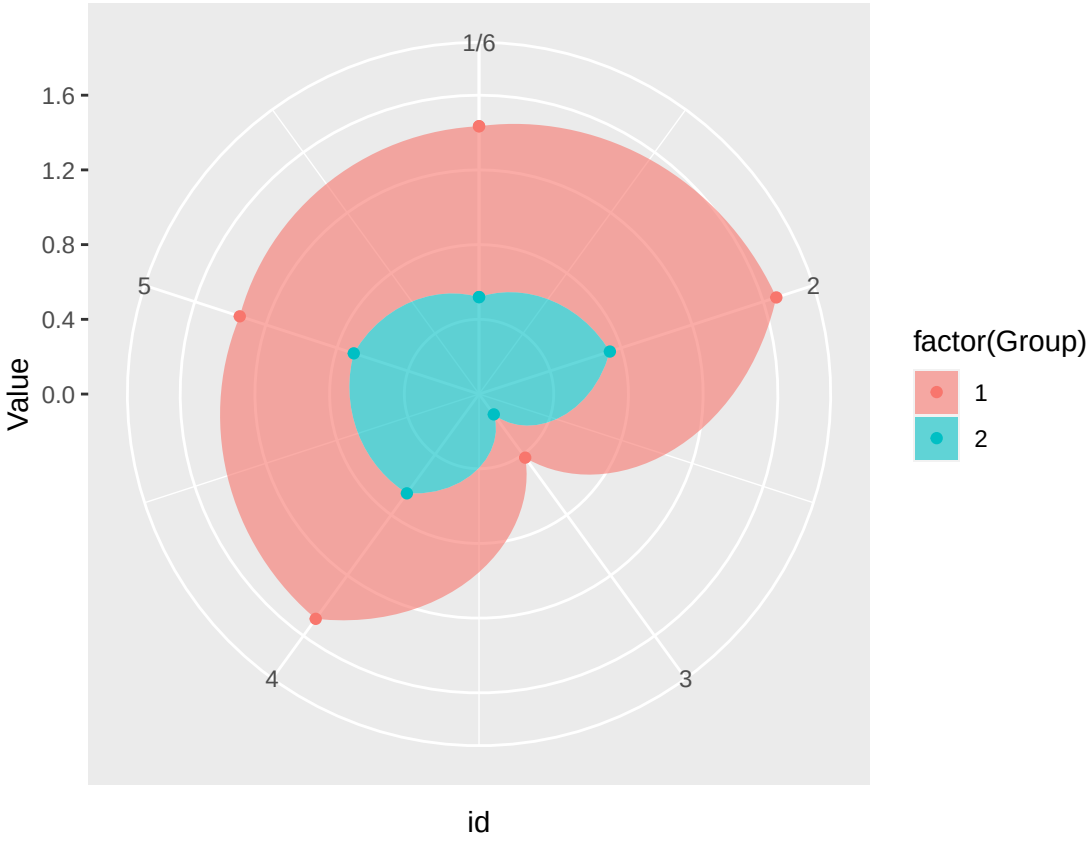


Polar

```
plot +
  coord_polar()
```




```
set.seed(42)
data <- data.frame(Obs1 = runif(5, 0, 1),
                   Obs2 = runif(5, 0, 1))

data <- rbind(data, data[1, ])

data.long <- reshape(data,
                     varying = c("Obs1", "Obs2"),
                     v.names = c("Value"),
                     #idvar = "id",
                     direction = "long",
                     timevar = "Group")


plot <- ggplot(data.long) +
  geom_area(aes(x = id, y = Value, fill = factor(Group)), alpha = 0.6) +
  geom_point(aes(x = id, y = Value, color = factor(Group)), position = "stack") +
  coord_polar(theta = "x")
plot
```

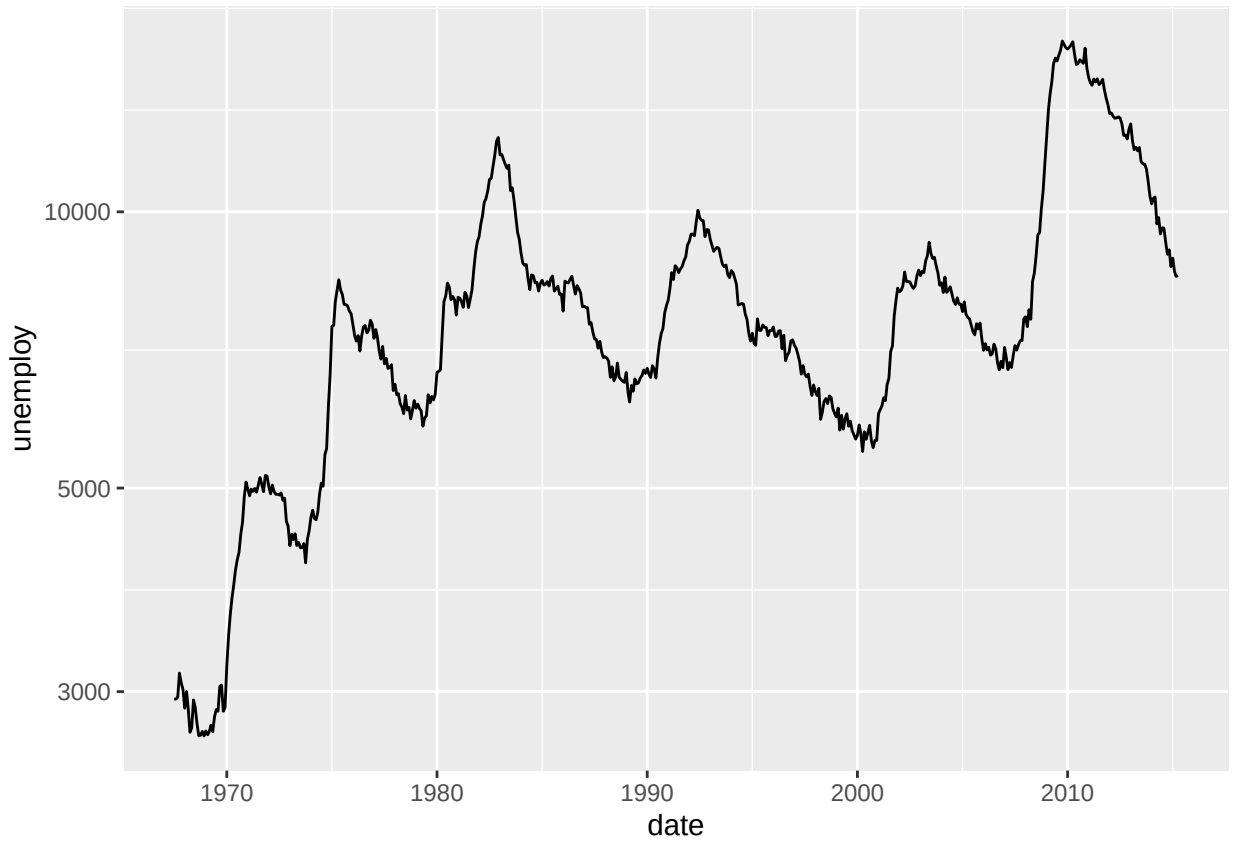


scale

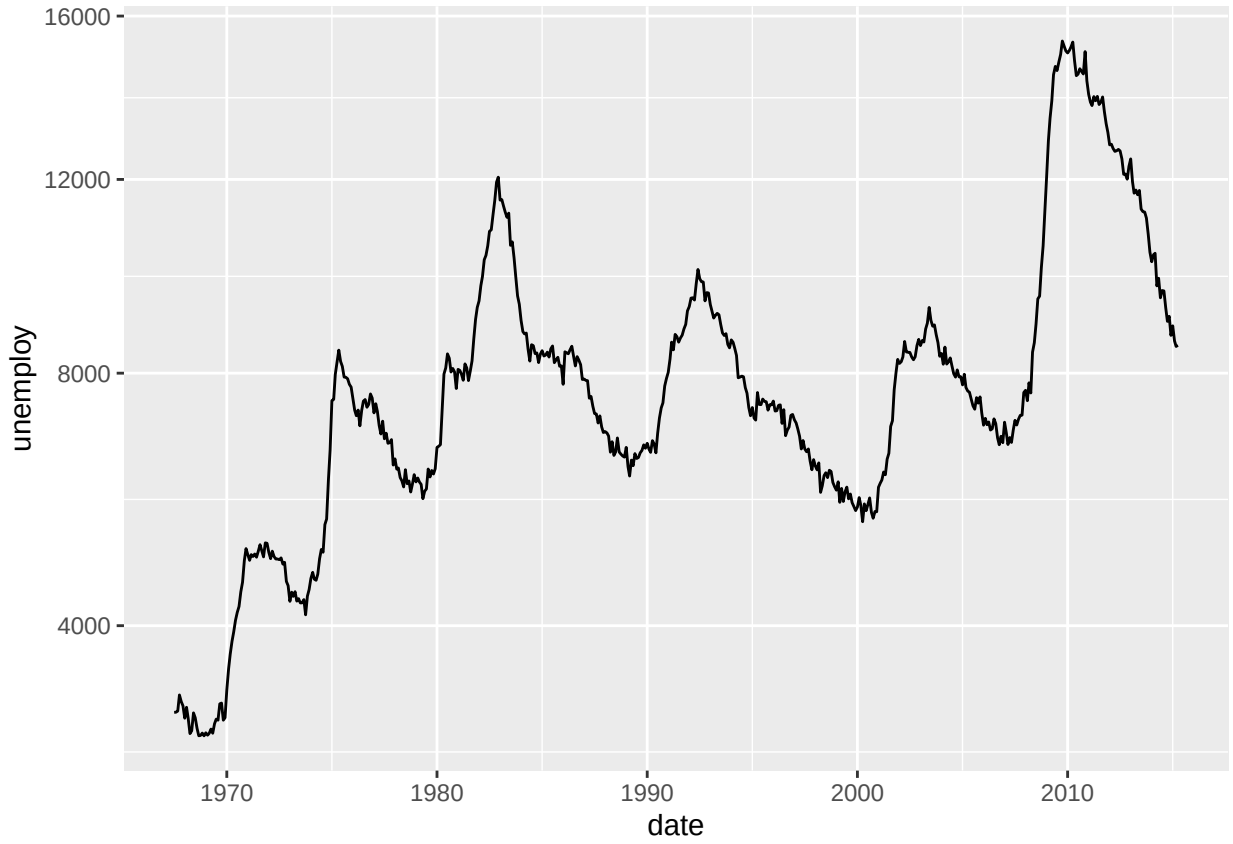
- ggplot unterscheidet i.d.R. selbständig zwischen diskreter und stetiger Skalierung, dies kann dennoch auch vorgegeben werden
- interessanter an dieser Stelle: Skalierung erlaubt Beeinflussung von Darstellungsbereich, "Breaks", Achsen-Skalierung, etc.

Achsenskalierung

```
data("economics", package = "ggplot2")

plot <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line()

plot +
  scale_y_log10()
```

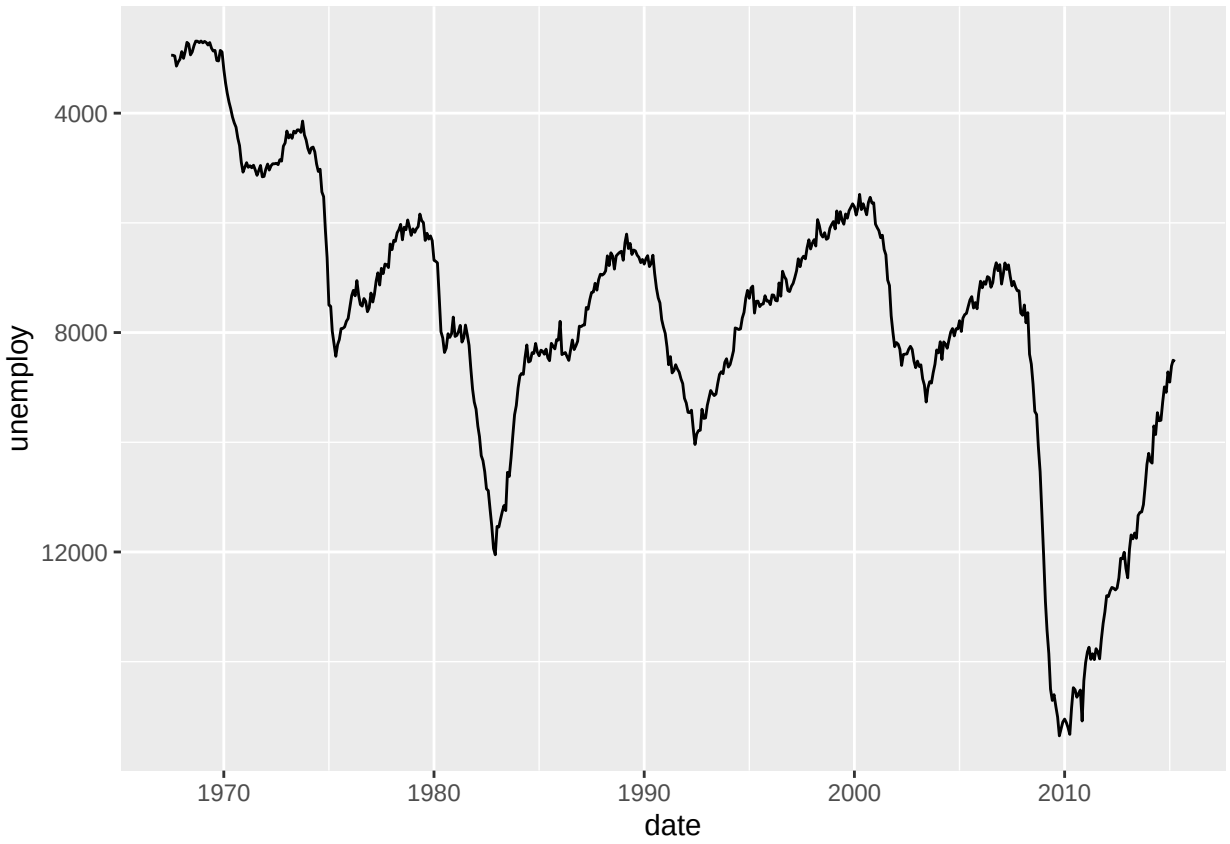



```
plot +
  scale_y_sqrt()
```



Hier sieht man wie sich einfach durch addition der entsprechenden `scale_*` Funktion ein z.B. logarythmisches oder Wurzel-Skalenniveau auf der Y-Achse umsätzen lässt.

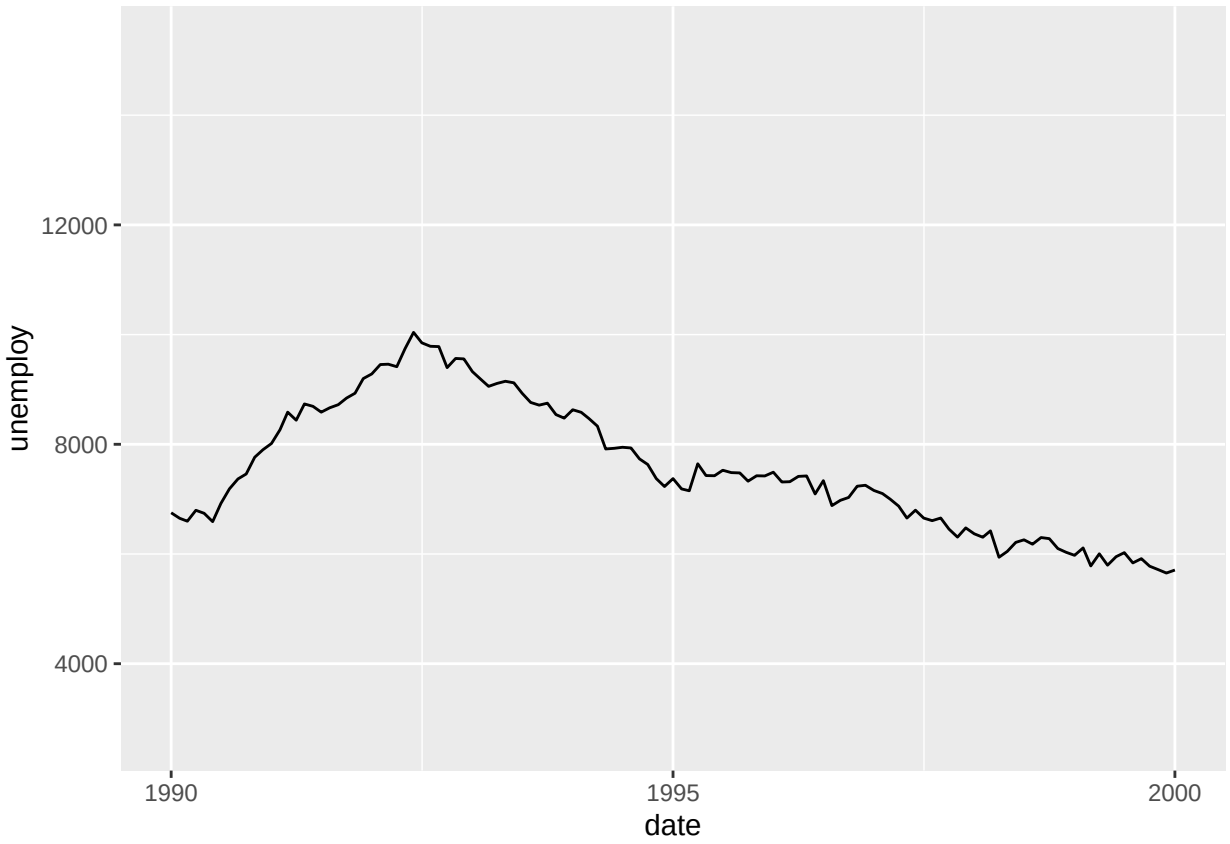
```
plot +
  scale_y_reverse()
```



Darstellungsbereich

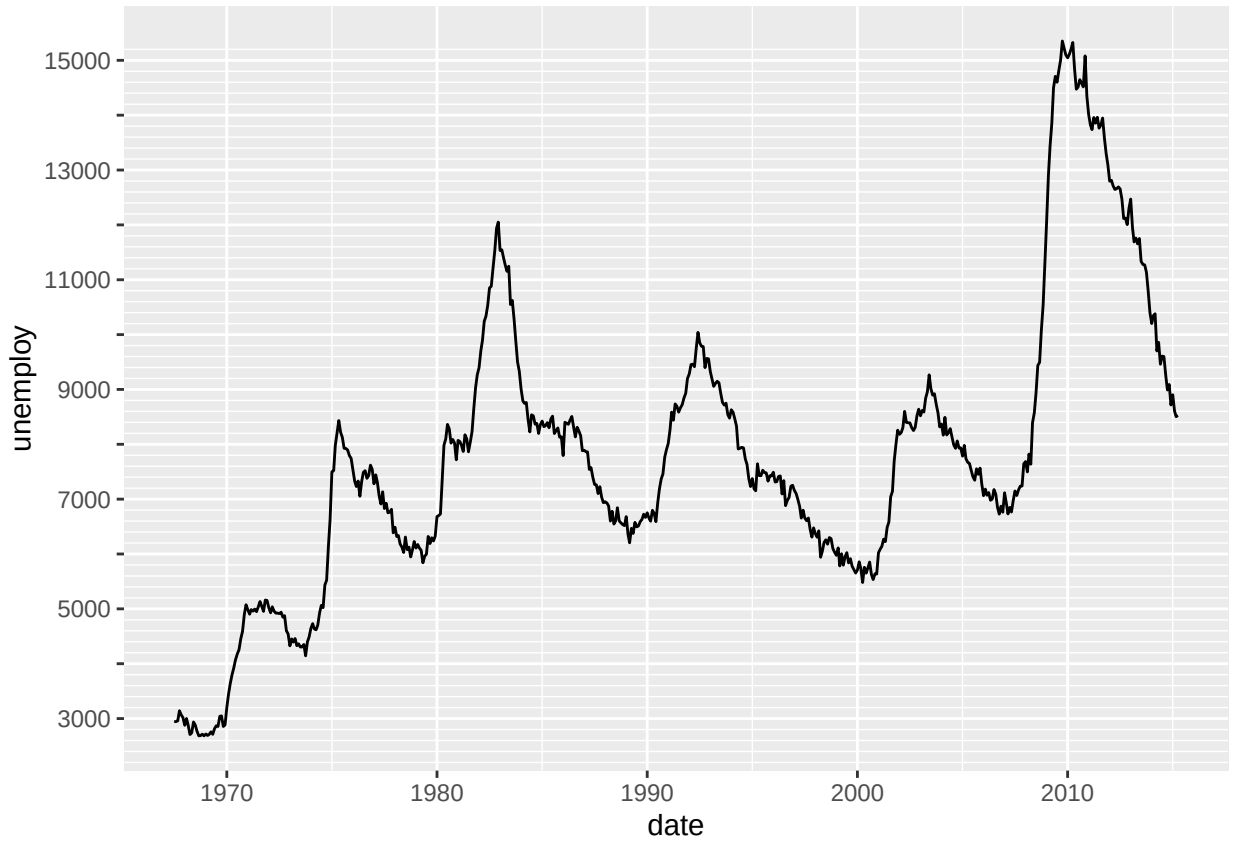
```
data("economics", package = "ggplot2")

plot <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line()

plot +
  scale_x_date(limits = as.Date(c("1990-01-01", "2000-01-01")))
```

```
## Warning: Removed 453 row(s) containing missing values (geom_path).
```



Breaks

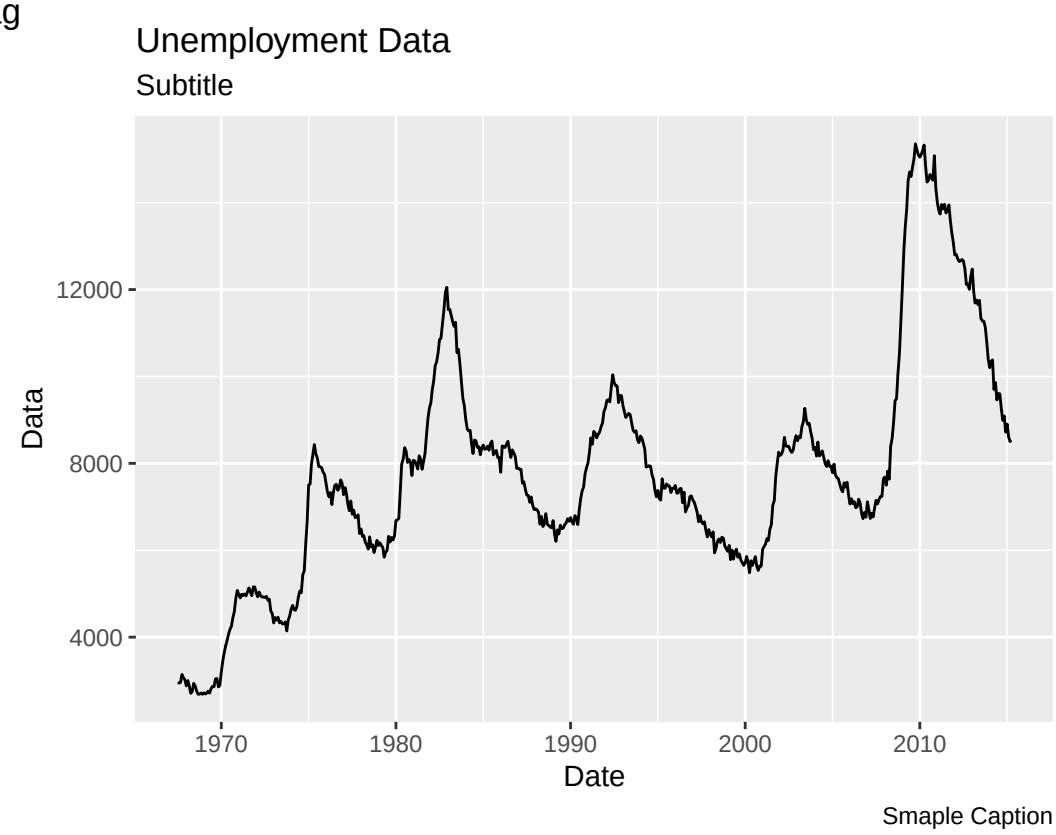
```
plot +
  scale_y_continuous(
    breaks = seq(from = 0, to = max(economics$unemploy), by = 1000),
    minor_breaks = seq(from = 0, to = max(economics$unemploy), by = 200),
    labels = function(x) ifelse(x %% 2000 == 1000, as.character(x), ""))
```



labels

```
data("economics", package = "ggplot2")

ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  labs(title = "Unemployment Data", subtitle = "Subtitle", caption = "Smaple Caption", tag = "This is a
```

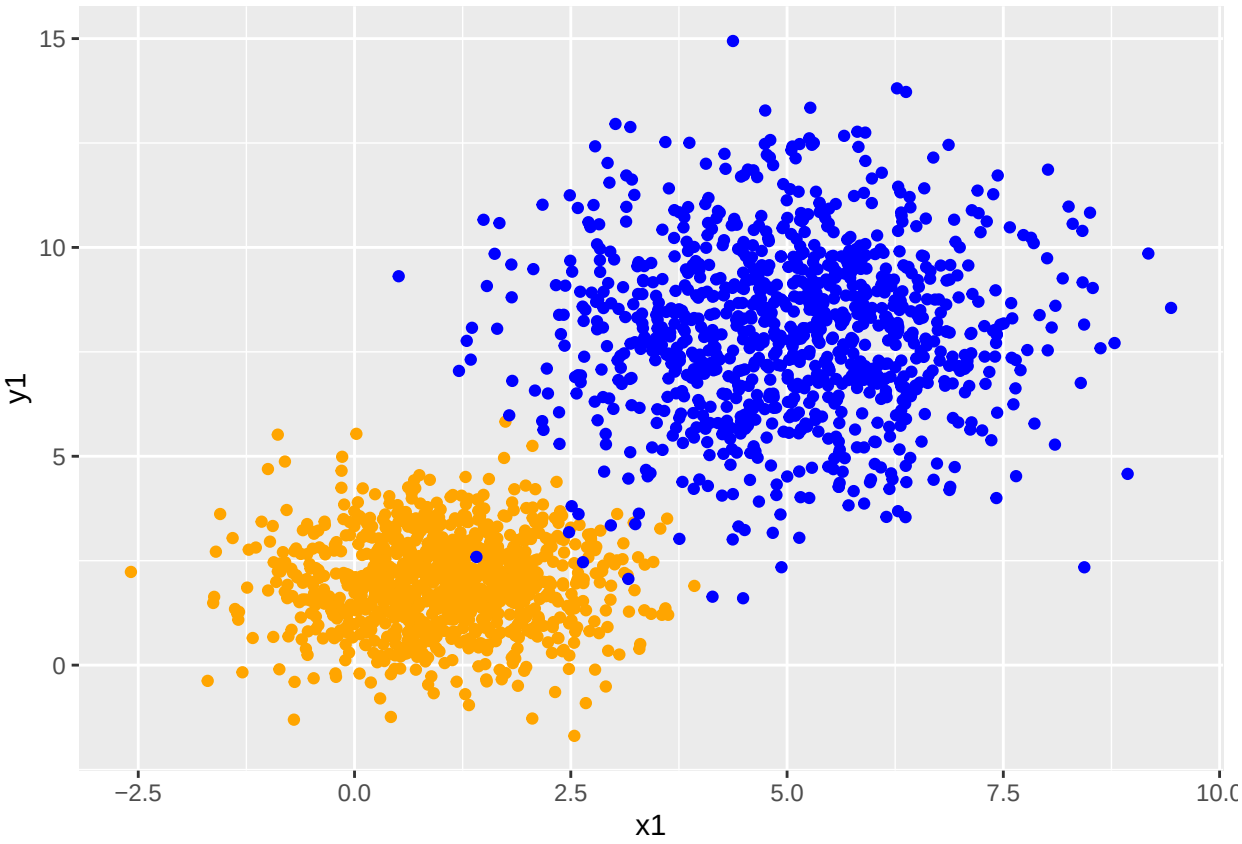
This is a tag



Farben: color und groups

Einfache Farben

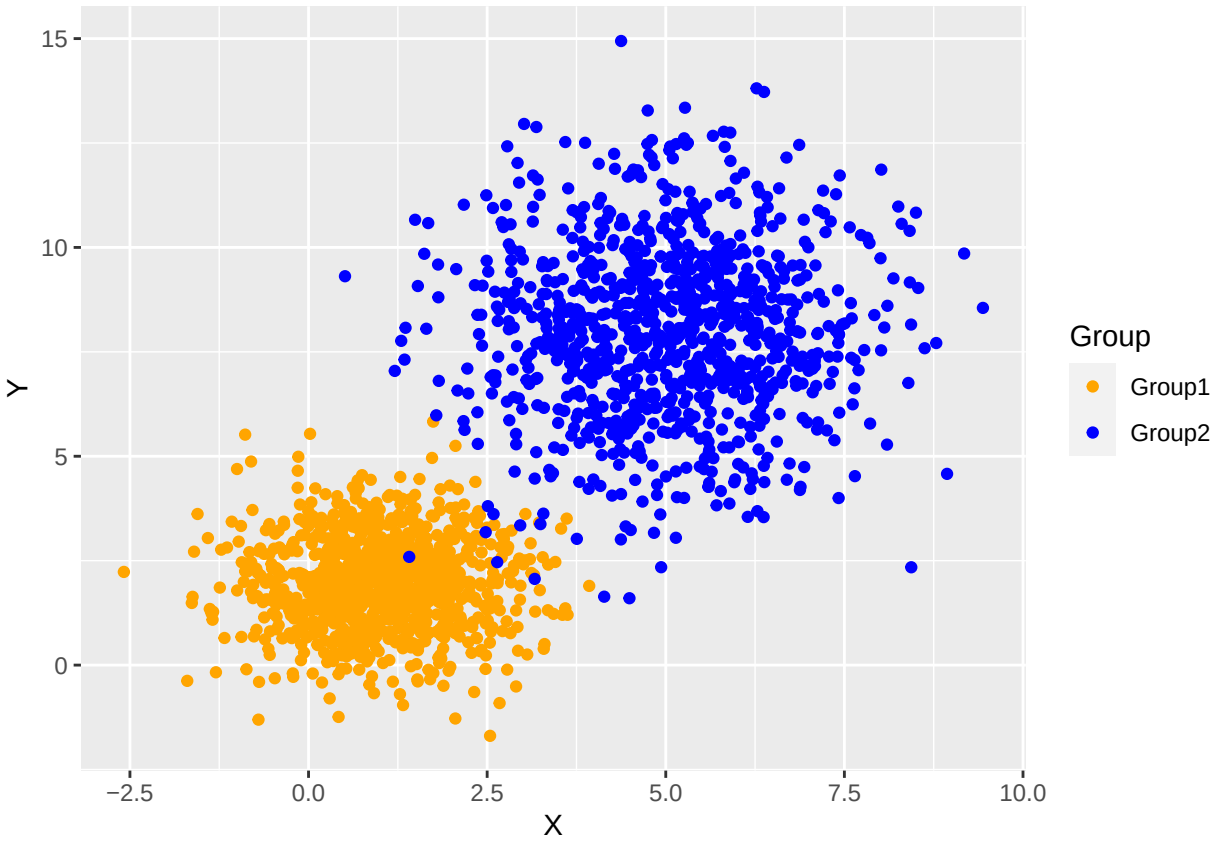
```
library(MASS)
n <- 1000
sigma1 <- matrix(c(1, 0, 0, 1.2), byrow = TRUE, nrow = 2)
sigma2 <- matrix(c(2, 0, 0, 4), byrow = TRUE, nrow = 2)
u1 <- mvrnorm(n = n, mu <- c(1, 2), Sigma = sigma1)
u2 <-  mvrnorm(n = n, mu <- c(5, 8), Sigma = sigma2)


data <- data.frame(x1 = u1[, 1], y1 = u1[, 2], x2 = u2[, 1], y2 = u2[, 2])

plot <- ggplot(data = data) +
  geom_point(aes(x = x1, y = y1), color = "orange") +
  geom_point(aes(x = x2, y = y2), color = "blue")

plot
```



Einfache Umsetzung von Farben in einer Grafik. Hier müssen für jedes Dataset das Attribut “color” konstant auf einen Wert gesetzt werden. Für aufwändige Grafiken empfiehlt es sich das Attribut “color” datengetrieben zu befühlen.

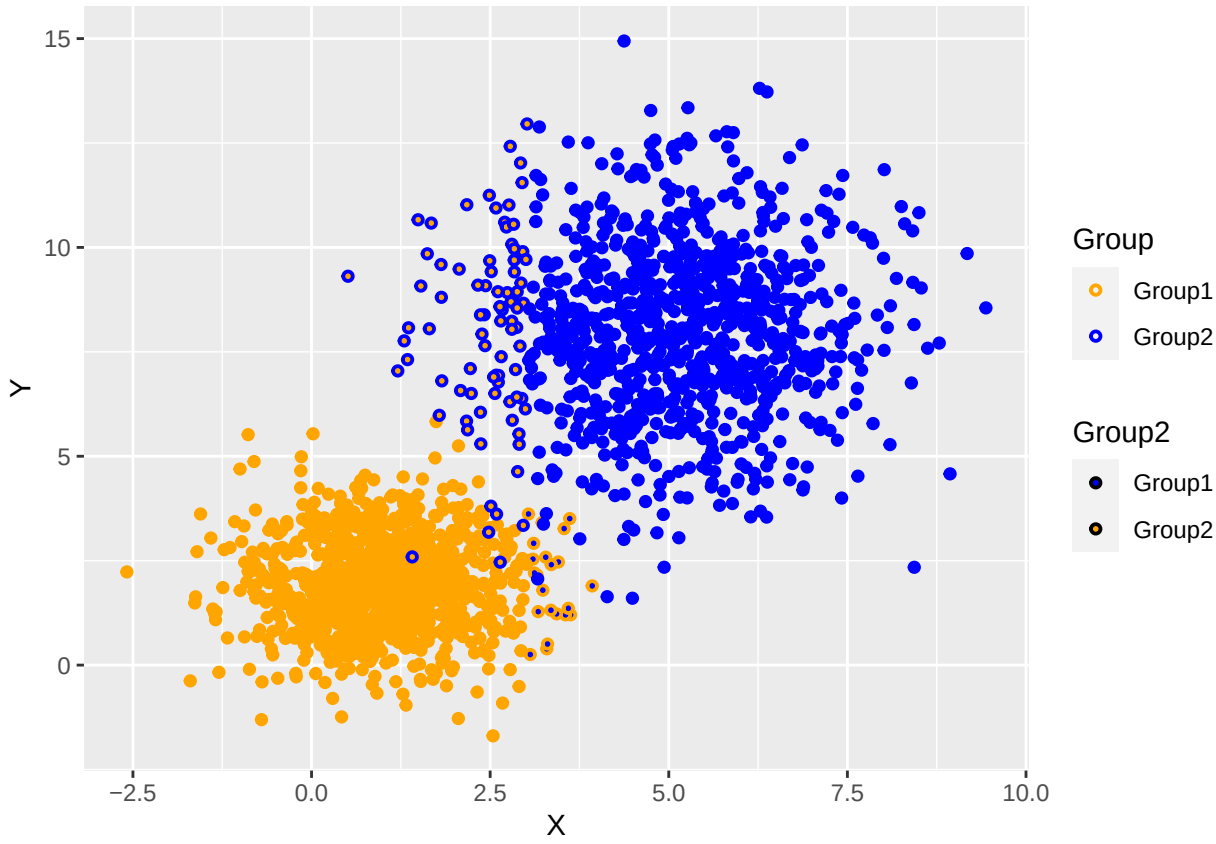
Benutzung von fill, color und group

```
data <- data.frame(rbind(cbind("Group1", u1), cbind("Group2", u2)))
names(data) <- c("Group", "X", "Y")
data$Group %<>% factor
data$X %<>% as.numeric
data$Y %<>% as.numeric

plot <- ggplot(data = data) +
  geom_point(aes(x = X, y = Y, colour = Group)) +
  scale_color_manual(values = c("Group1" = "orange", "Group2" = "blue"))
plot
```



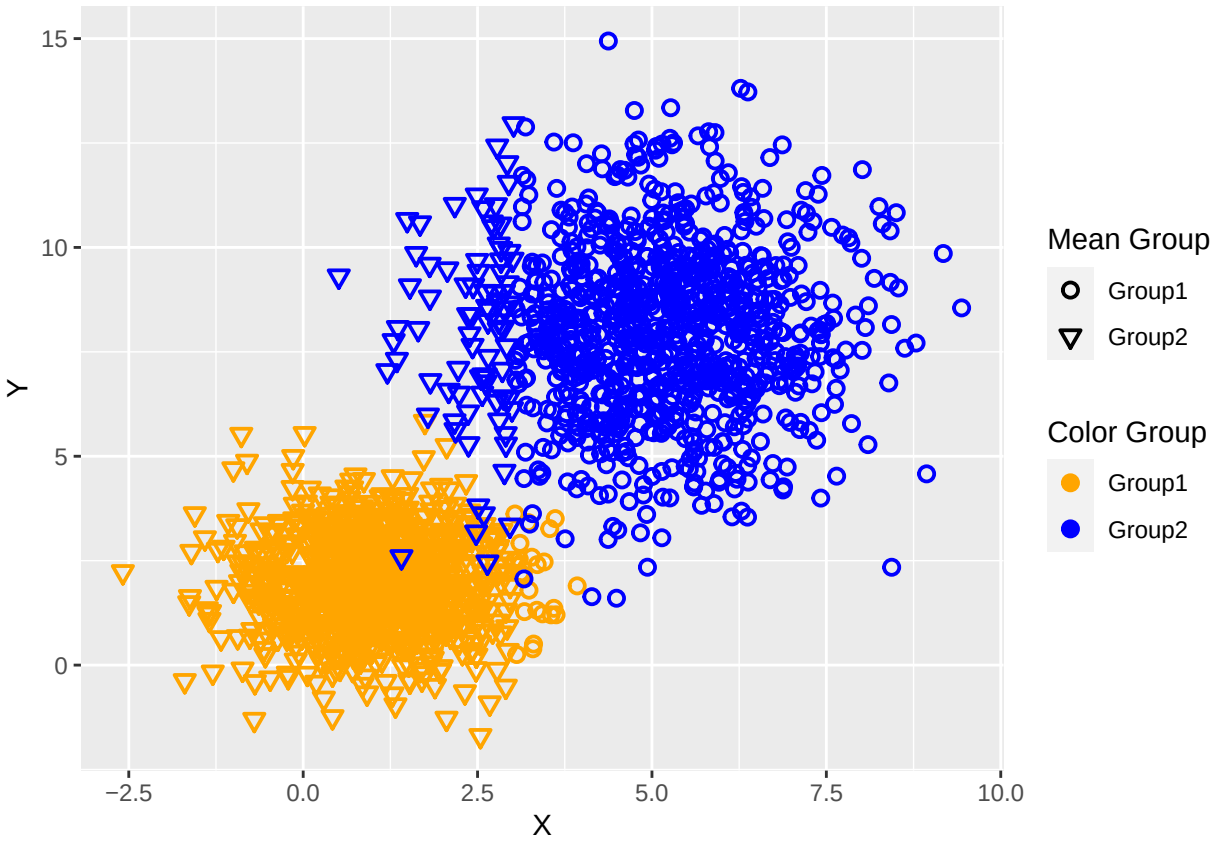
Hier wurde die Datentabelle um eine Variable `Gruppe` ergänzt, welche den entsprechenen Datensatz einer Gruppe zuordnet. Das `color` Attribut wird hier nun innerhalb der `aes` Funktion zugewiesen. Anstatt die Farbe konstant für einen Datensatz zu setzen, kann ggplot nun Automatische die Gruppenzugehörigkeit bestimmen. Den Gruppen werden dann automatisch Farben zugewiesen (i.d.R. rot und blau). Die Farbzuweisung kann nun über eine anpassung der "Farbskala" umgesetzt werden, dabei wird jeder spezifische Gruppenwert auf einen Farbwert gemappt.

```
data$Group2 <- factor(ifelse(data$X > mean(data$X), "Group1", "Group2"))

plot <- ggplot(data = data) +
  geom_point(aes(x = X, y = Y, fill = Group2, color = Group), shape = 21, stroke = 1, size = 1) +
  scale_color_manual(values = c("Group1" = "orange", "Group2" = "blue")) +
  scale_fill_manual(values = c("Group1" = "blue", "Group2" = "orange"))
plot
```



Hier sieht man ein Beispiel, wie sich mehrere Gruppen dazu nutzen lassen mehrere Attribute (hier Füllfarbe und Linienfarbe) datengetrieben zu beeinflussen.

```
plot <- ggplot(data = data, aes(x = X, y = Y, shape = Group2, color = Group)) +
  geom_point(stroke = 1, size = 2) +
  scale_color_manual(name = "Color Group",
                     values = c("Group1" = "orange", "Group2" = "blue")) +
  scale_shape_manual(name = "Mean Group",
                     values = c("Group1" = 1, "Group2" = 6))
plot
```



Einschub: Long VS Wide Datenstrucktur

```
library(MASS)
library(magrittr)
n <- 1000
sigma1 <- matrix(c(1, 0, 0, 1.2), byrow = TRUE, nrow = 2)
sigma2 <- matrix(c(2, 0, 0, 4), byrow = TRUE, nrow = 2)
u1 <- mvrnorm(n = n, mu <- c(1, 2), Sigma = sigma1)
u2 <-  mvrnorm(n = n, mu <- c(5, 8), Sigma = sigma2)


data.wide <- data.frame(x1 = u1[, 1], y1 = u1[, 2], x2 = u2[, 1], y2 = u2[, 2])

data.long <- data.frame(rbind(cbind("Group1", u1), cbind("Group2", u2)))
names(data.long) <- c("Group", "X", "Y")
data.long$Group %<>% factor
data.long$X %<>% as.numeric
data.long$Y %<>% as.numeric

head(data.wide)
```

```
##          x1        y1       x2        y2
## 1 1.1188095 0.9052596 5.091501  7.487357
## 2 0.1064411 2.3109119 5.359812  6.562535
## 3 0.7888855 0.7937024 6.226056  3.959726
## 4 1.4887798 3.0772026 8.121957  8.779226
## 5 1.2203719 0.8863427 6.294620  3.216016
```

```
## 6 3.2962458 1.0536368 5.346431 10.125132
```

```
head(data.long)
```

```
##    Group         X         Y
## 1 Group1 1.1188095 0.9052596
## 2 Group1 0.1064411 2.3109119
## 3 Group1 0.7888855 0.7937024
## 4 Group1 1.4887798 3.0772026
## 5 Group1 1.2203719 0.8863427
## 6 Group1 3.2962458 1.0536368
```

- ggplot arbeitet besser mit long-Datenstrukturen
- arbeiten mit der intuitiven wide Struktur ebenfalls möglich, jedoch kann dies häufig wesentlich aufwändiger sein, da bei dem fill-Attribut dann für jede Gruppe eine eigene Zeile verwendet werden muss
- dank einigen automatisierten Funktionen lassen sich wide-Struckturen relativ unaufwändig in long-Struckturen überführen

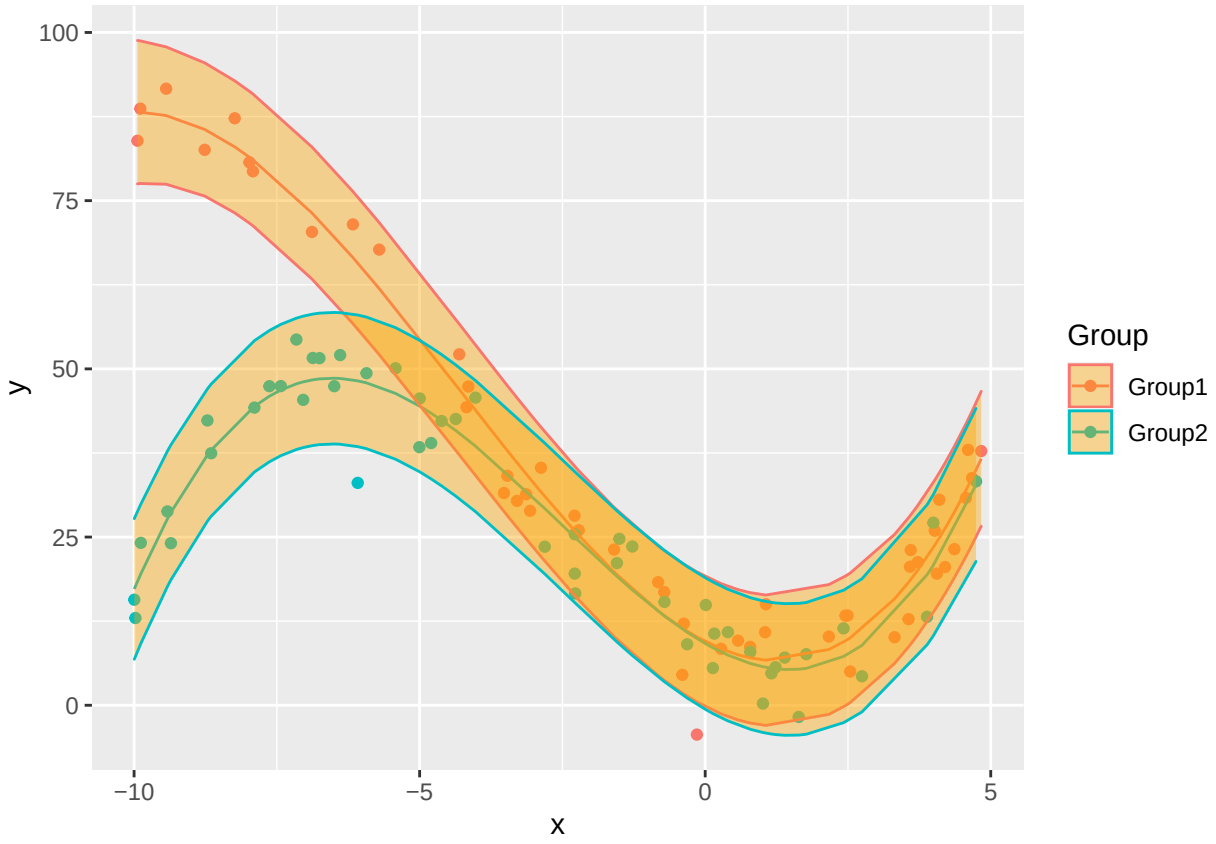
```
n <- 100
sigma1 <- matrix(c(1, 0, 0, 1.2), byrow = TRUE, nrow = 2)
sigma2 <- matrix(c(2, 0, 0, 4), byrow = TRUE, nrow = 2)
u1 <- mvrnorm(n = n, mu <- c(1, 2), Sigma = sigma1)
u2 <-  mvrnorm(n = n, mu <- c(5, 8), Sigma = sigma2)


data.wide <- data.frame(x1 = u1[, 1], y1 = u1[, 2], x2 = u2[, 1], y2 = u2[, 2])

library(reshape2)

data.wide$id <- 1:nrow(data.wide)

data.long <- reshape(data.wide,
                     varying = c("x1", "y1", "x2", "y2"),
                     v.names = c("x", "y"),
                     #idvar = "id",
                     direction = "long",
                     timevar = "Group")
```

Eyecandy: Themes

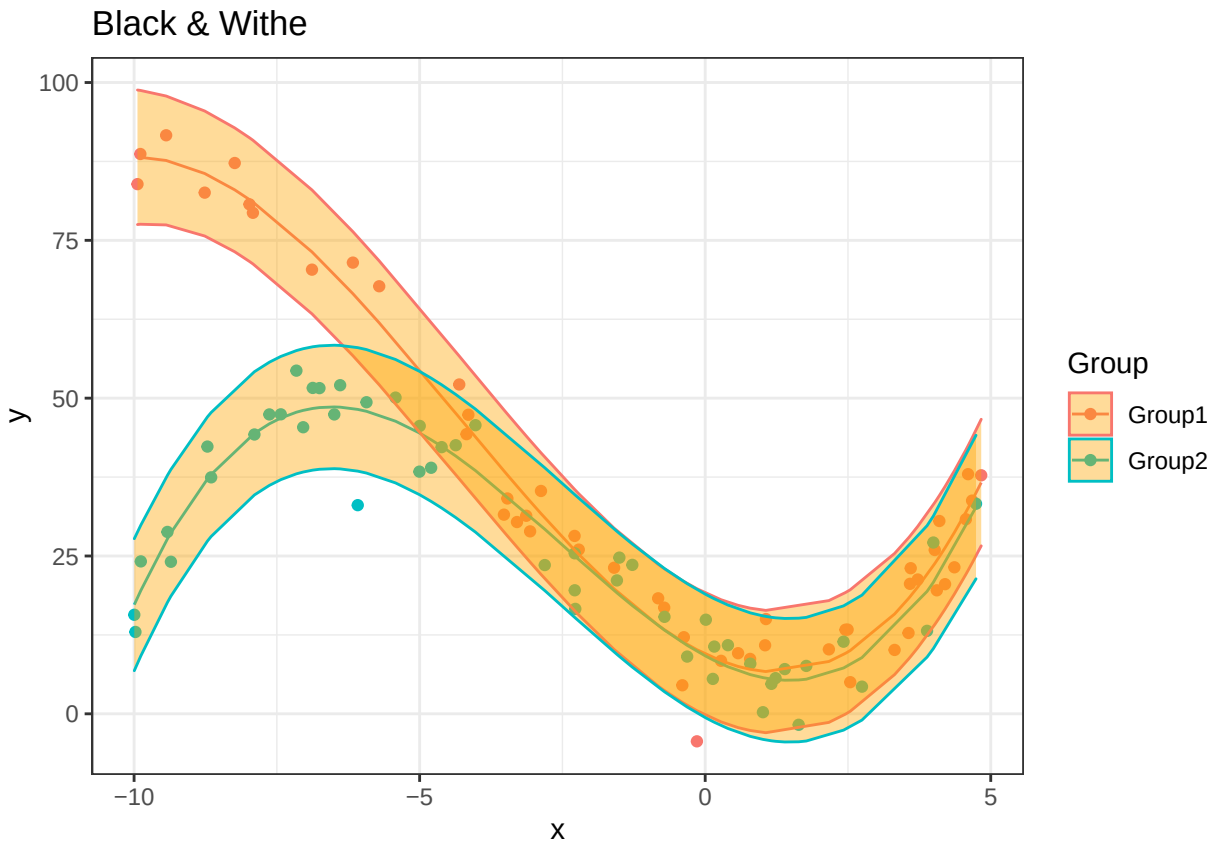
Allgemeine Darstellungsweise

```
set.seed(42)
x1 <- runif(n = 50, -10, 5)
x2 <- runif(n = 50, -10, 5)

u <- rnorm(n = 50, mean = 0, sd = 5)

y1 <- 10 + 0.1 * x1^3 + 1.4 * x1^2 - 4 * x1 + u
y2 <- 8 + 0.17 * x2^3 + 1.31 * x2^2 - 4.9 * x2 + u

data1 <- data.frame(Group = "Group1", y = y1, x = x1)
data2 <- data.frame(Group = "Group2", y = y2, x = x2)
```

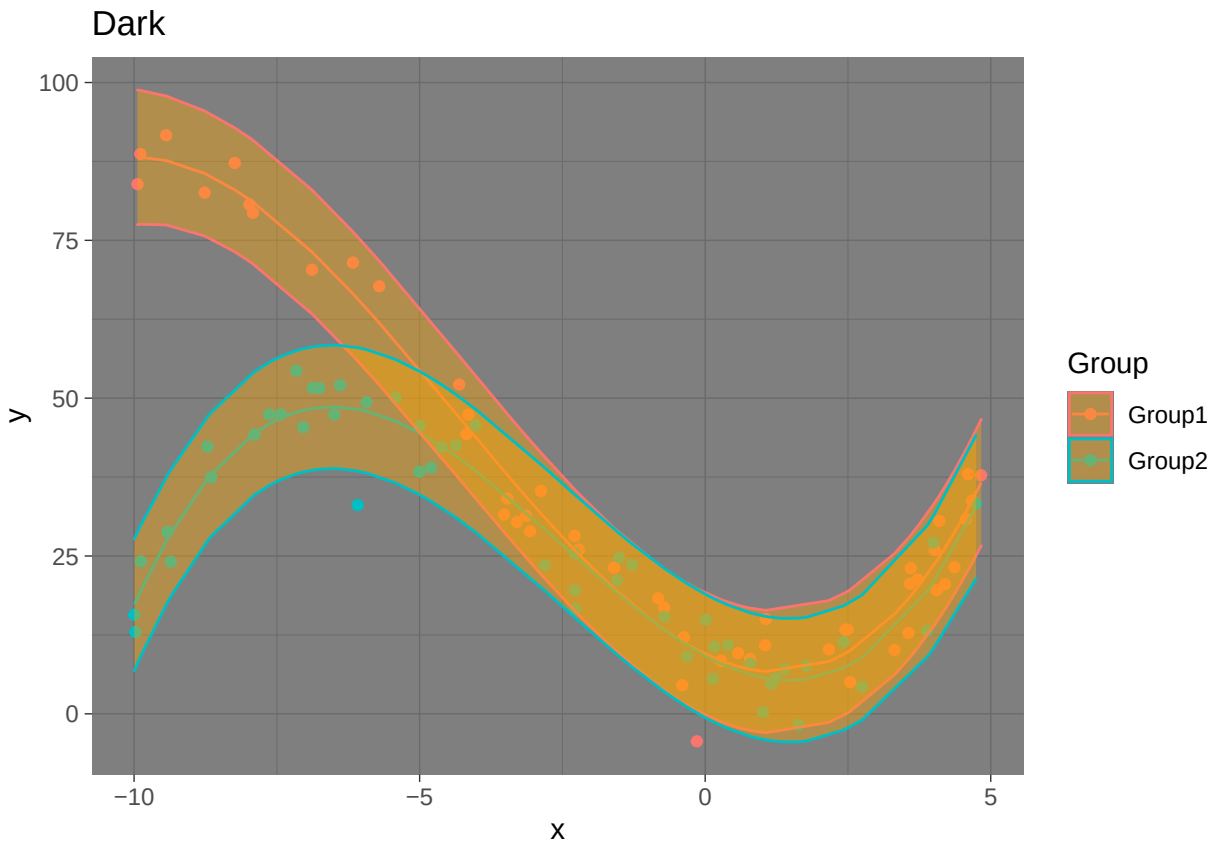
```
lm1 <- lm(y ~ poly(x, degree = 3), data = data1)
lm2 <- lm(y ~ poly(x, degree = 3), data = data2)
data <- rbind(cbind(data1, predict(lm1, data1, interval="predict")),
              cbind(data2, predict(lm2, data2, interval="predict")))

plot <- ggplot(data, aes(color = Group)) +
  geom_point(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = fit)) +
  geom_ribbon(aes(x = x, ymin = lwr, ymax = upr), fill = "orange", alpha = 0.4)
plot
```



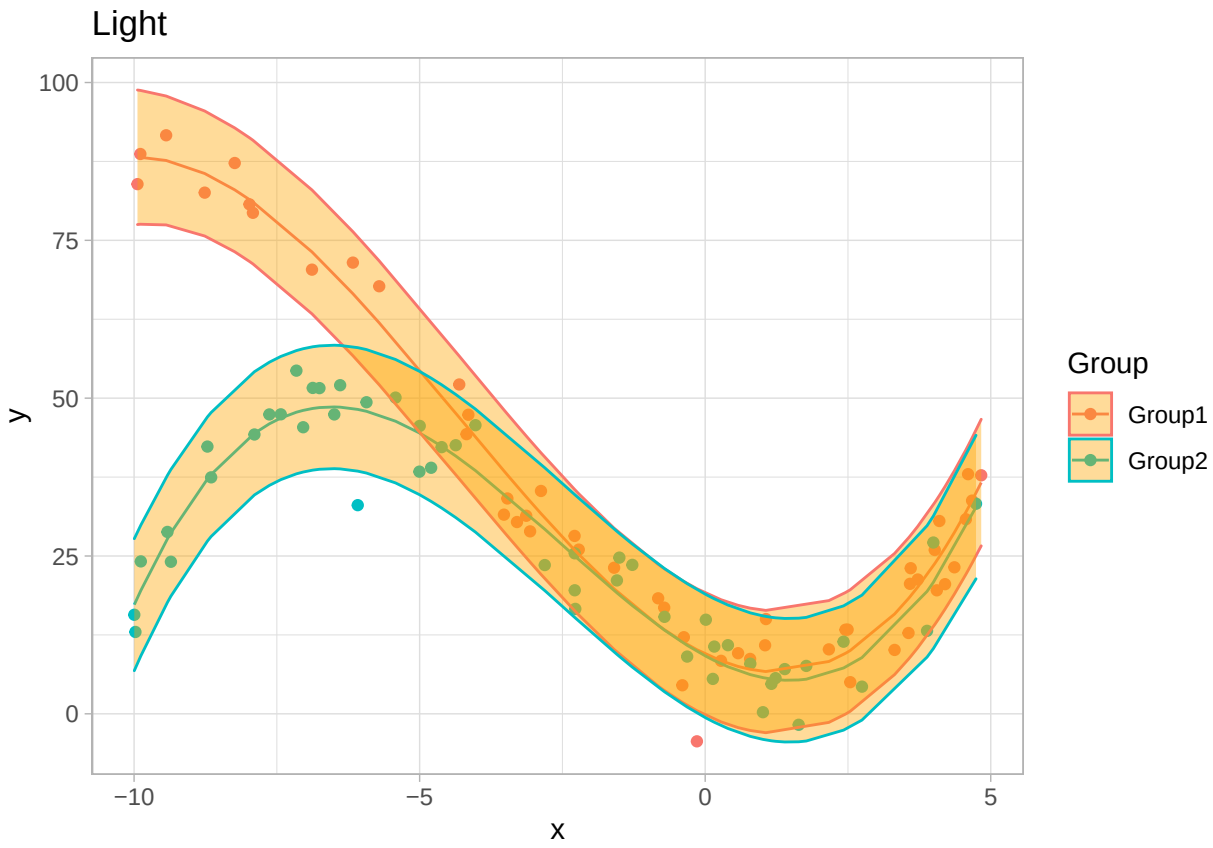

```
plot +
  labs(title = "Black & Withe") +
  theme_bw()
```



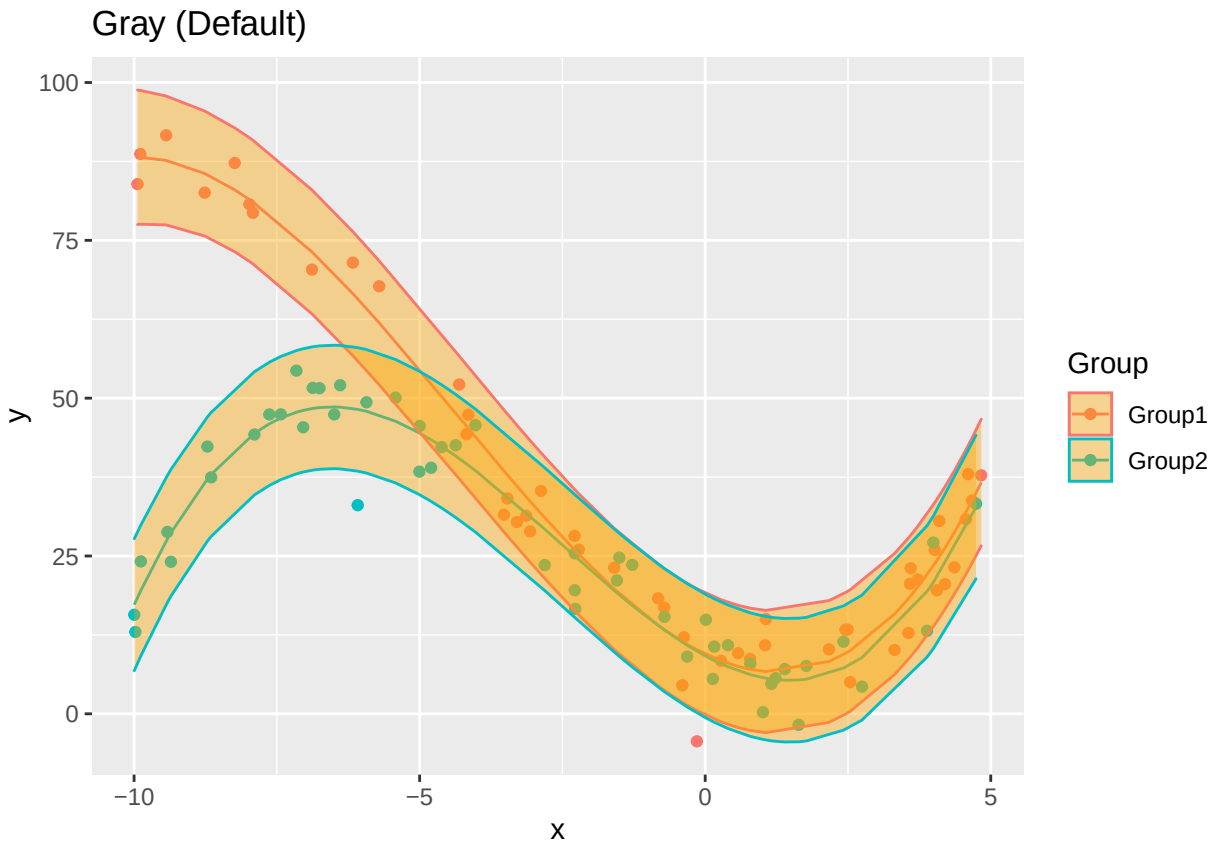

```
plot +
  labs(title = "Dark") +
  theme_dark()
```



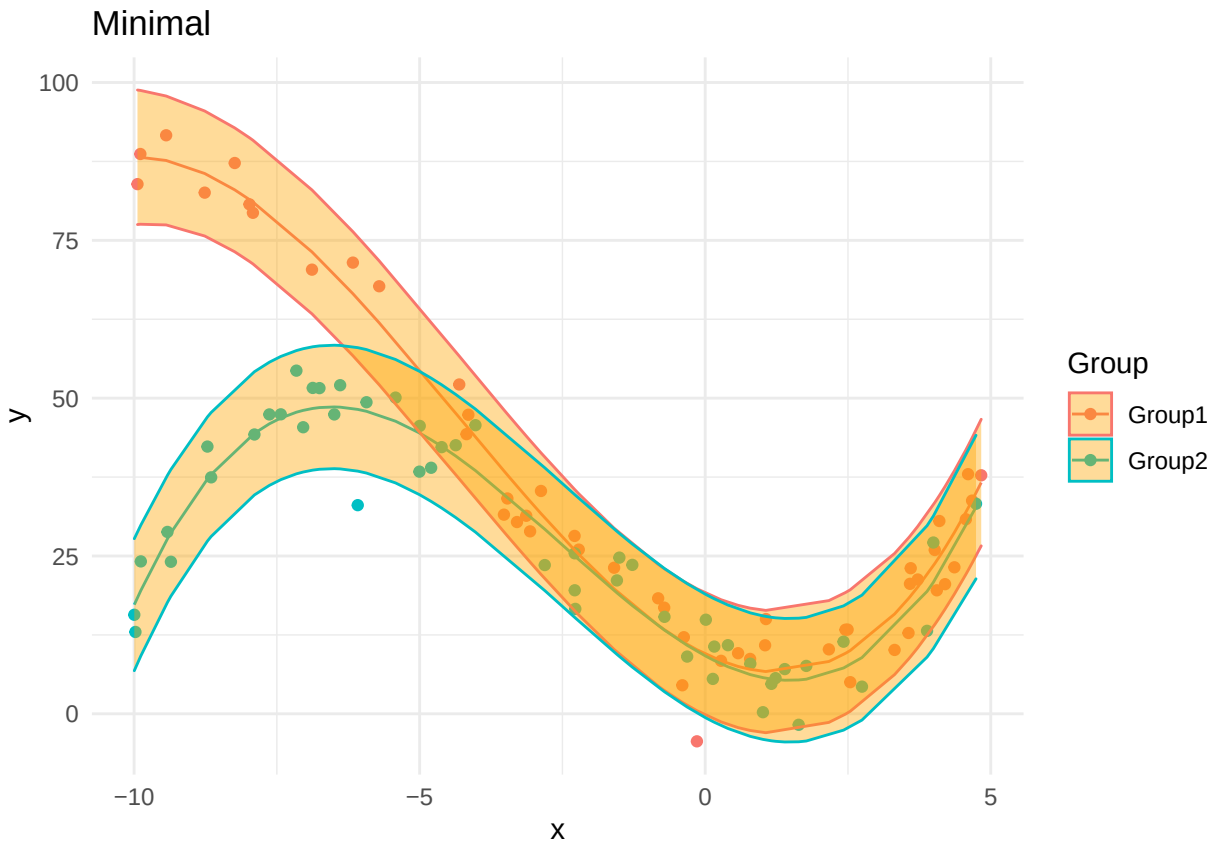

```
plot +
  labs(title = "Light") +
  theme_light()
```



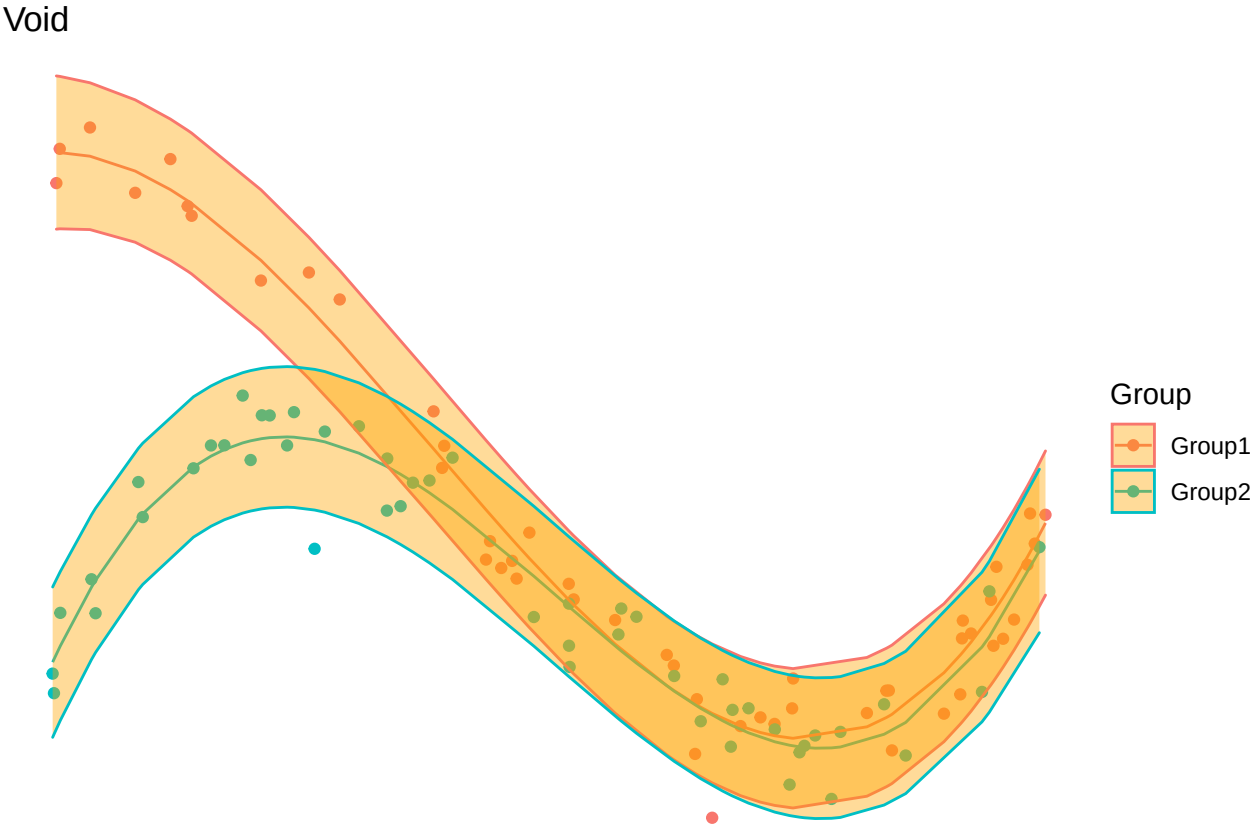

```
plot +
  labs(title = "Gray (Default)") +
  theme_gray()
```




```
plot +
  labs(title = "Minimal") +
  theme_minimal()
```




```
plot +
  labs(title = "Void") +
  theme_void()
```

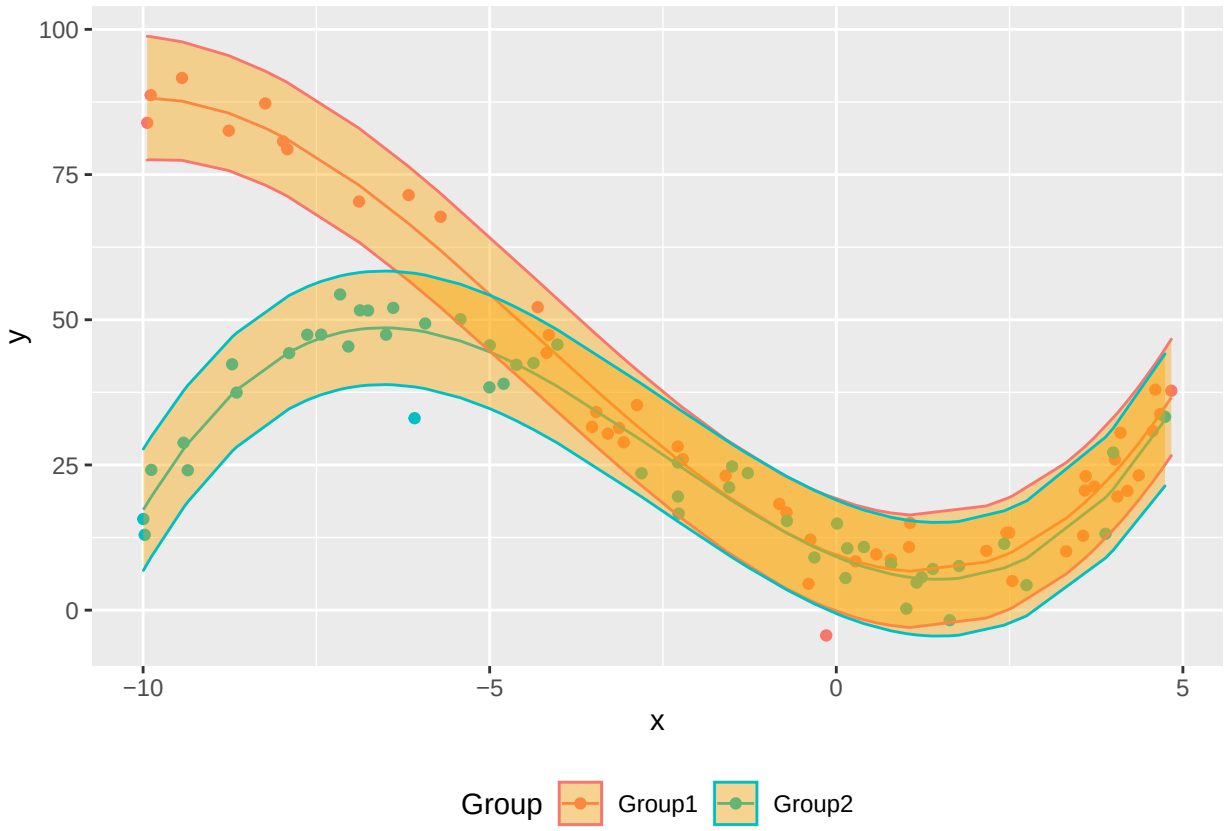


Mit den vorgefertigten Themes lässt sich schnell das Aussehend er Grafik an die eigenen Wünsche anpassen. Es lassen sich aber auch noch viele weitere, sehr viel detailliertere änderungen an dem Aussehend er Grafik tätigen.

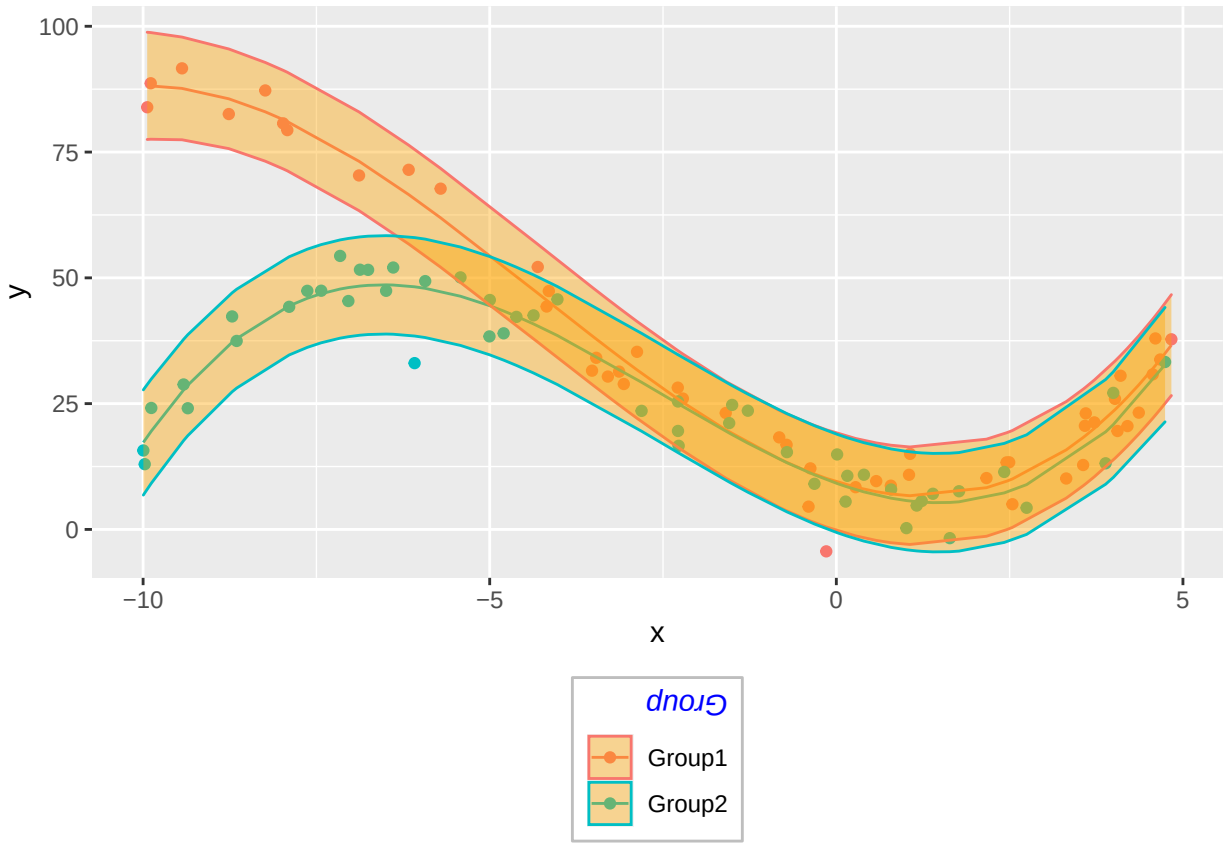
Weitere Optionen

Legende

```
plot +
  theme(legend.position = "bottom")
```




```
plot +
  theme(legend.position = "bottom",
        legend.title = element_text(color = "blue", angle = 180, face = "italic"),
        legend.direction = "vertical",
        legend.background = element_rect(colour = "grey"))
```



Dur die Anpassung verschiedener Attribute der `theme` Layer, können alle dargestellten Elemente verändert werden, dazu gehört auch die Legende, hier als Beispiel eine Veränderung der Position sowie verschiedene Formatierungen von Elementen der Legende.

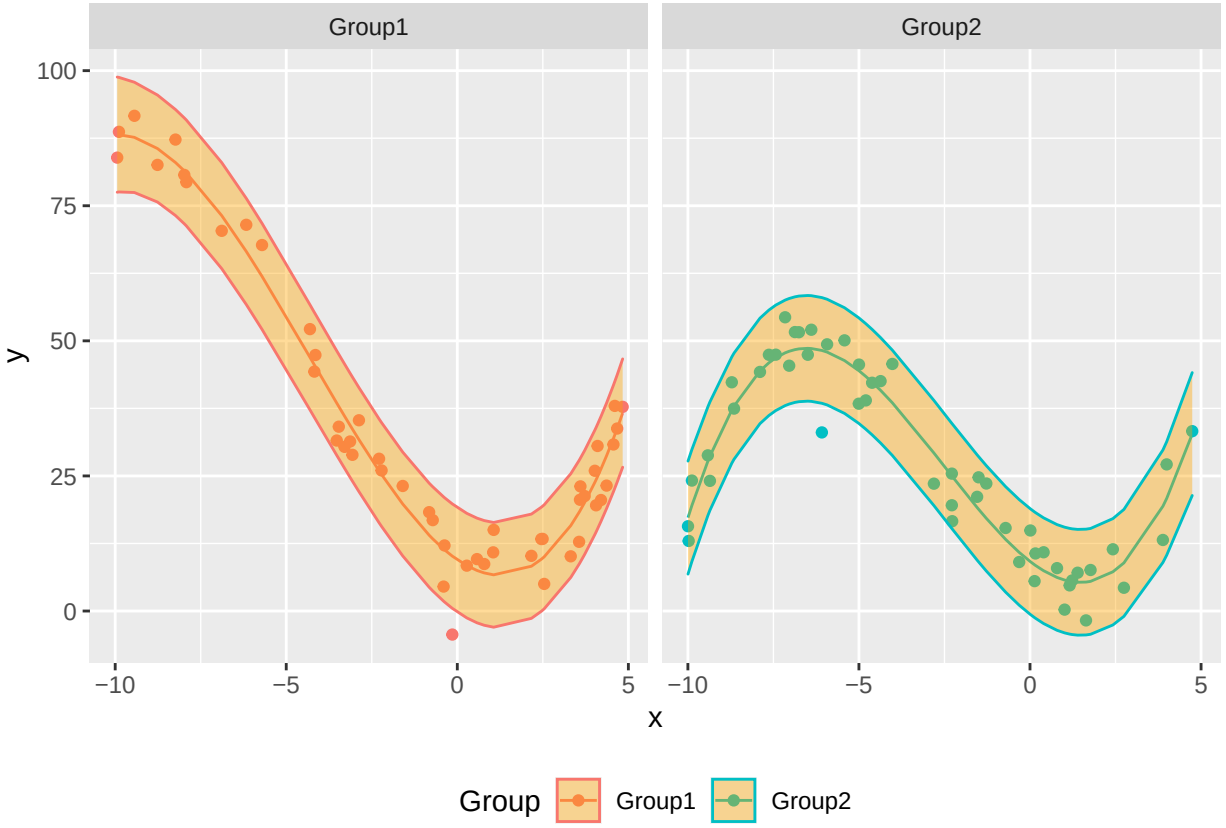
Fortgeschritten

Faceting

Feceting erlaubt das Automatische Plotten mehrerer, getrennter Grafiken, differenziert bei einer Gruppen Variable.

```
set.seed(42)
x1 <- runif(n = 50, -10, 5)
x2 <- runif(n = 50, -10, 5)

u <- rnorm(n = 50, mean = 0, sd = 5)

y1 <- 10 + 0.1 * x1^3 + 1.4 * x1^2 - 4 * x1 + u
y2 <- 8 + 0.17 * x2^3 + 1.31 * x2^2 - 4.9 * x2 + u

data1 <- data.frame(Group = "Group1", y = y1, x = x1)
data2 <- data.frame(Group = "Group2", y = y2, x = x2)

lm1 <- lm(y ~ poly(x, degree = 3), data = data1)
lm2 <- lm(y ~ poly(x, degree = 3), data = data2)
data <- rbind(cbind(data1, predict(lm1, data1, interval="predict")),
              cbind(data2, predict(lm2, data2, interval="predict")))
```

```
plot <- ggplot(data, aes(color = Group)) +
  geom_point(aes(x = x, y = y)) +
  geom_line(aes(x = x, y = fit)) +
  geom_ribbon(aes(x = x, ymin = lwr, ymax = upr), fill = "orange", alpha = 0.4)


plot +
  facet_grid(cols = vars(Group)) +
  theme(legend.position = "bottom")
```

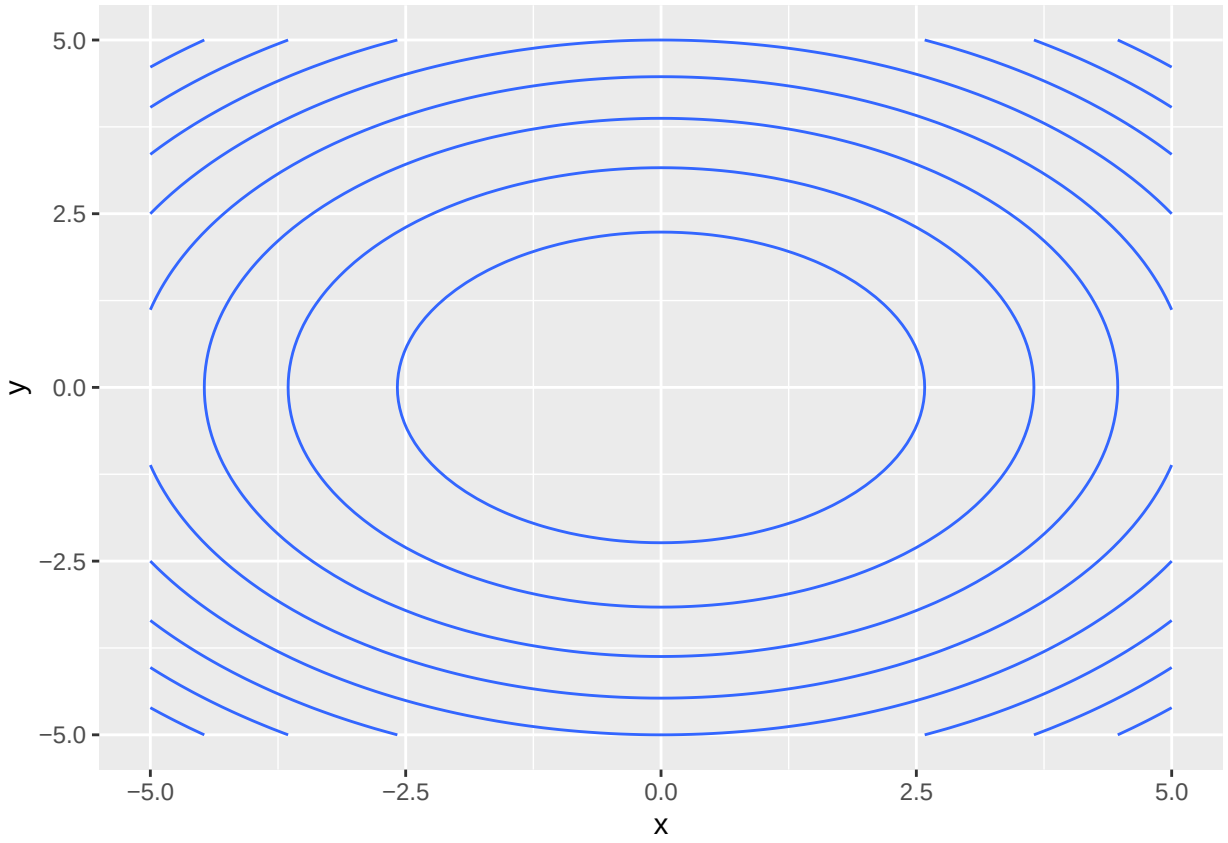


Hier wurden zwei Regressionen geschätzt, die Variable `Group` dient zur Unterscheidung der jeweiligen Daten der Stichprobe, der Prognose (aka. Fitting), den Prognoseintervallen etc.
Der Befehl `facet_grid` dient nun dazu die Grafiken für die einzelnen Gruppen Tabellenförmig anzuordnen (hier nur zwei Spalten, Zeilenweise oder Tabellen-Darstellung ebenfalls möglich)

Trivariat

```
fun <- function(x, z){-3 * x^2 - 4 * z^2}

i <- seq(-5, 5, by = .1)
ii <- sort(rep(i, length(i)))
data <- data.frame(x = ii, y = i)
data$z <- fun(data$x, data$y)

plot <- ggplot(data, aes(x = x, y = y, z = z)) +
  geom_contour()
```

```
plot
```

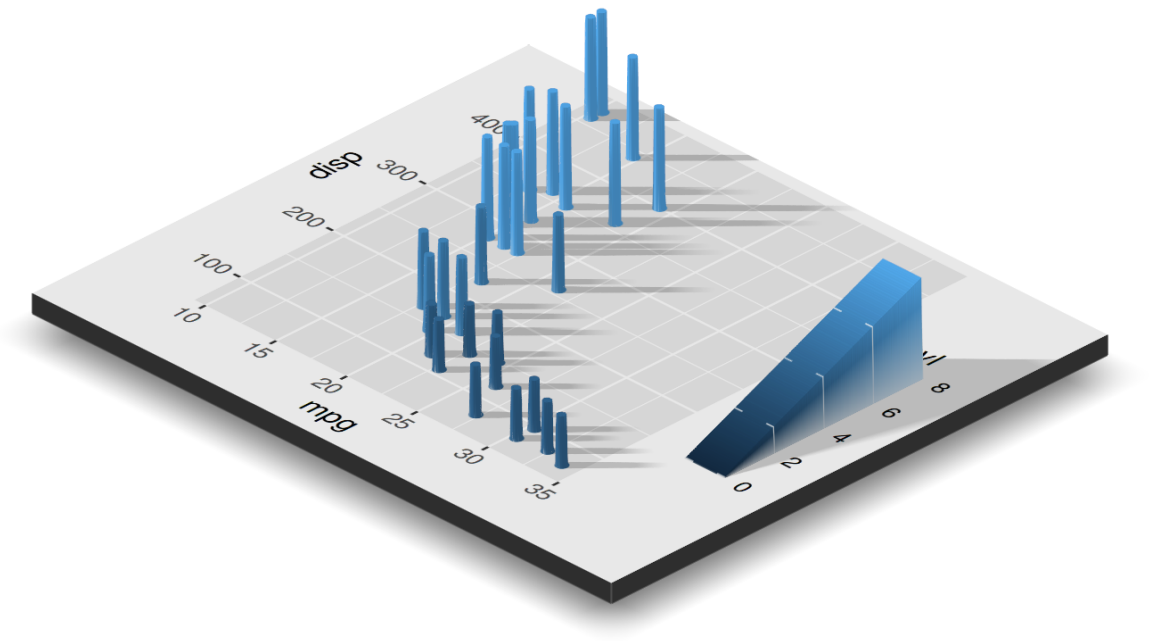


Es wird eine 3D Funktion definiert, anschließend wird eine Tabelle aus (x, z)-Positionen generiert und dann für jedes Tupel ein z-Wert berechnet. Der Befehl `geom_contour` führt nun zu einer Darstellung der Höhenlinien.

```
library(viridis)
```

```
## Loading required package: viridisLite
```

```
library(rayshader)
mtplot = ggplot(mtcars) +
 geom_point(aes(x=mpg,y=disp,color=cyl)) +
 scale_color_continuous(limits=c(0,8))
plot_gg(mtplot, width=3.5, multicore = FALSE, windowsize = c(1400,866), sunangle=225,
       zoom = 0.60, phi = 30, theta = 45)
render_snapshot(clear = TRUE)
```



stats

Die stats-Befehle erlauben das automatische Berechnen von einfachen Statistiken während dem Plotforgang.