Lösung Blatt 1

Prof. Dr. Annika Hoyer, Marc Schneble

30.04.2020

Contents

Vorbereitungen

Pakete laden

```
library(ggplot2)
```

Datenaufbereitung

```
wdi <- readRDS("R/wdi_daten.rds")
wdi_info <- readRDS("R/wdi_info.rds")
wdi13 <- wdi[wdi$Jahr == 2013,]
wdi_westeuropa <- wdi[wdi$Region == "Western Europe", ]
```

Aufgabe 1: ggplot2

Allgemeines

Quellen:

- Offizielles Buch: 'ggplot2: Elegant Graphics for Data Analysis' (Wickham, 2016).
- ggplot2 Cheat Sheet

Wieso ggplot2?

- Verwendung einer konsistenten *Grammatik*
- Plotspezifikation auf hohem Abstraktionslevel → komplexe Graphiken einfacher als mit base R Funktionen
- hohe Flexibilität
- Aussehen der Plots einfach per Themes anpassbar

Vergleich zu base R Graphiken:

- ggplot2 folgt einer *Grammatik* → Befehle folgen einer logischeren Syntax
- Befehle sind meist etwas länger für einfache Graphiken
- Befehle sind teilweise deutlich kürzer für komplexe Graphiken
- Befehle werden immer auf einen data.frame angewandt ⇒ Es gibt keine klassenspezifischen Methoden (plot.lm, plot.gam, . . .)

1a)

Die Grammatik von ggplot2

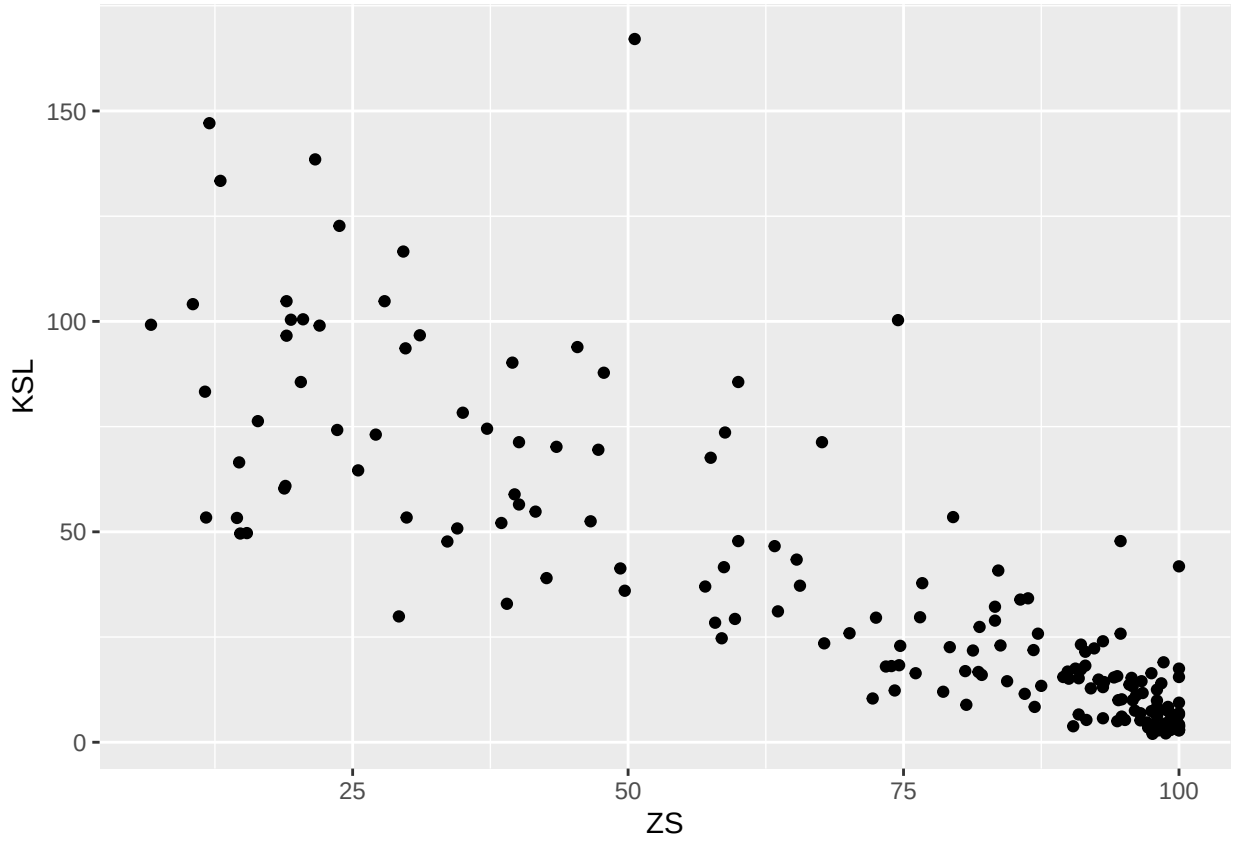
ggplot2-Graphiken bestehen aus einzelnen Ebenen (*layers*). Jede Ebene wird durch 5 Dimensionen definiert:

- *data*: Daten in Form eines data.frame
- *aesthetic mappings*: Zuordnungsregeln, auf welche Art die Variablen den Dimensionen des Plots zugeordnet werden
- *geom*: Plottyp (Scatterplot, Boxplot, ...). geom_-Funktionen sind Shortcuts für Erstellung einer Ebene
- *stat*: Eventuelle Transformation der Daten vor dem Plotten
- *position adjustment*: Räumliche Anordnung der Grafikelemente

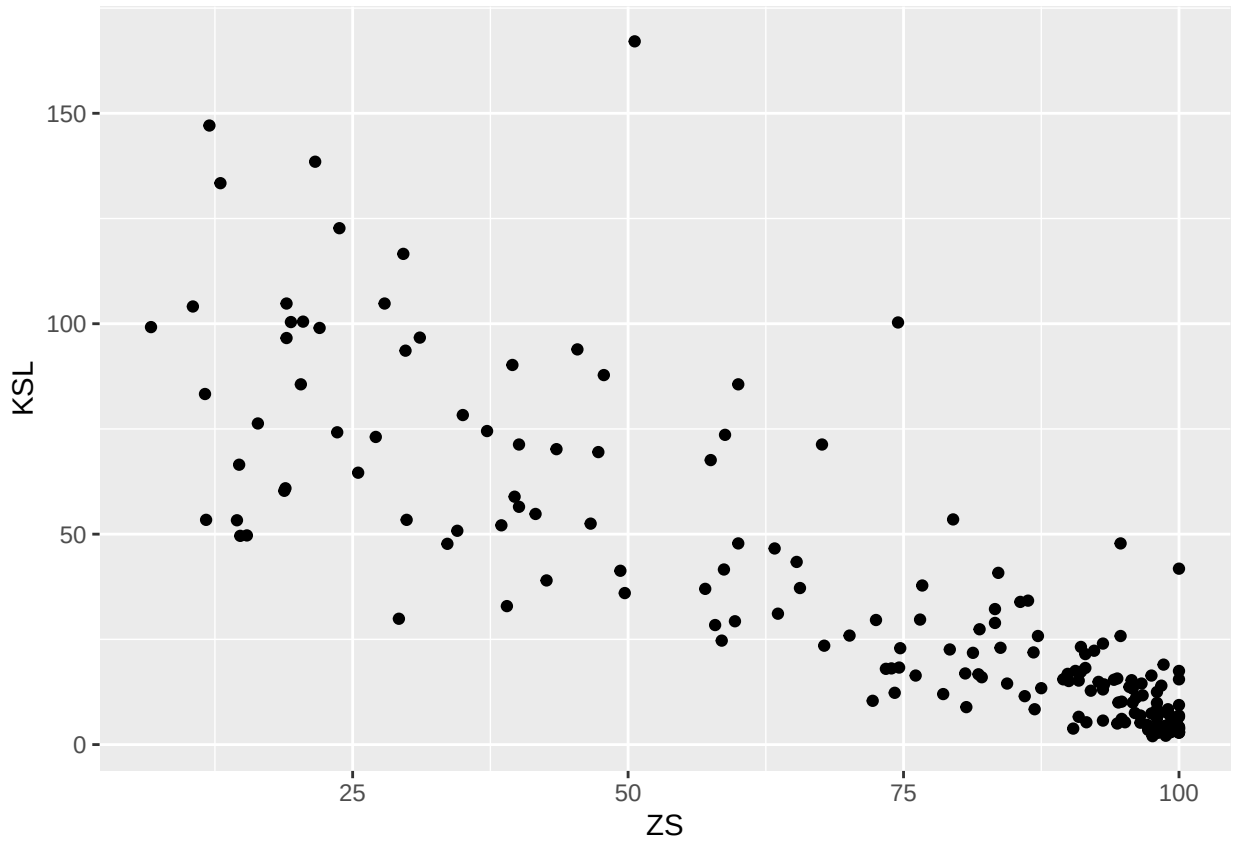
Beispiele für Definition einer Ebene:

Beispiel 1: Scatterplot

```
g <- ggplot(wdi13, aes(x = ZS, y = KSL))
# Shortcut-Definition der Ebene durch geom_point()
g + geom_point()
```

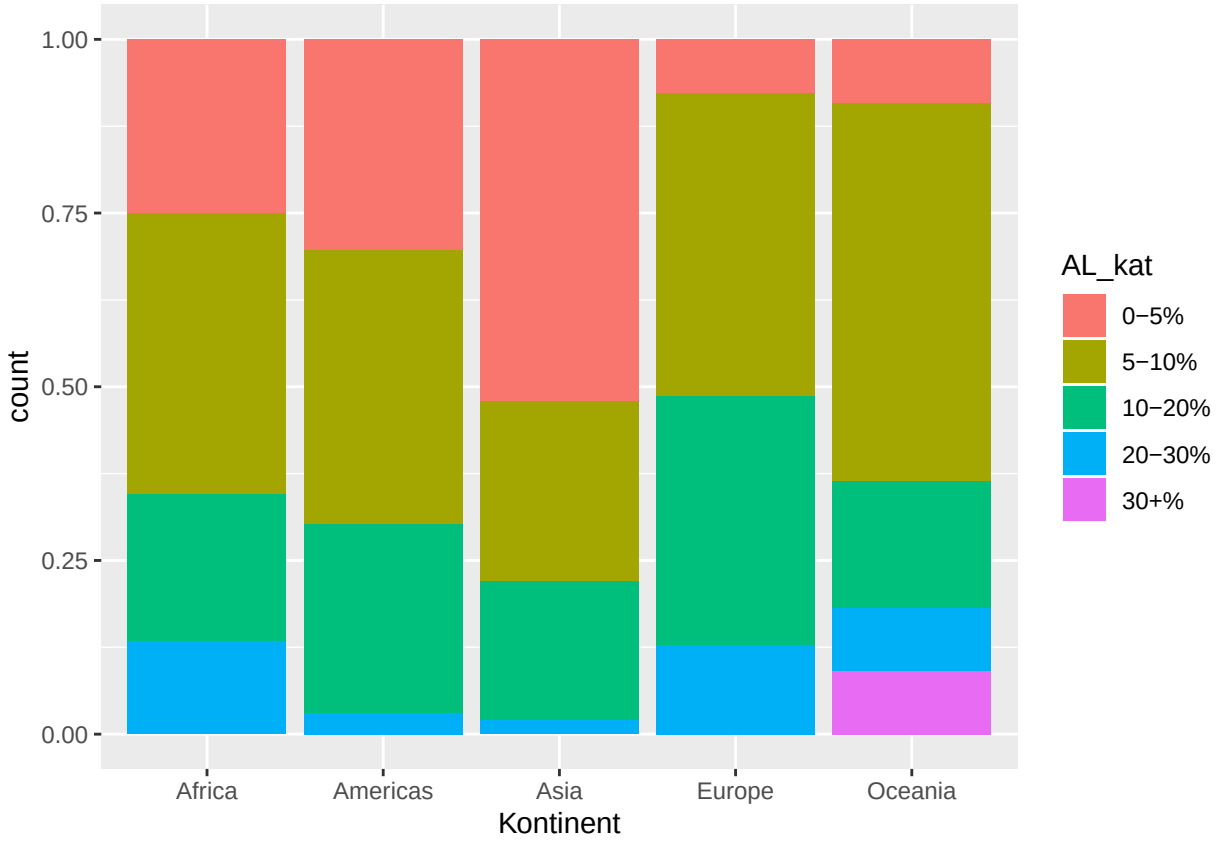



```
# Alternative: Selbe Ebene per Hand defineren
g + layer(geom = "point", stat = "identity", position = "identity")
```

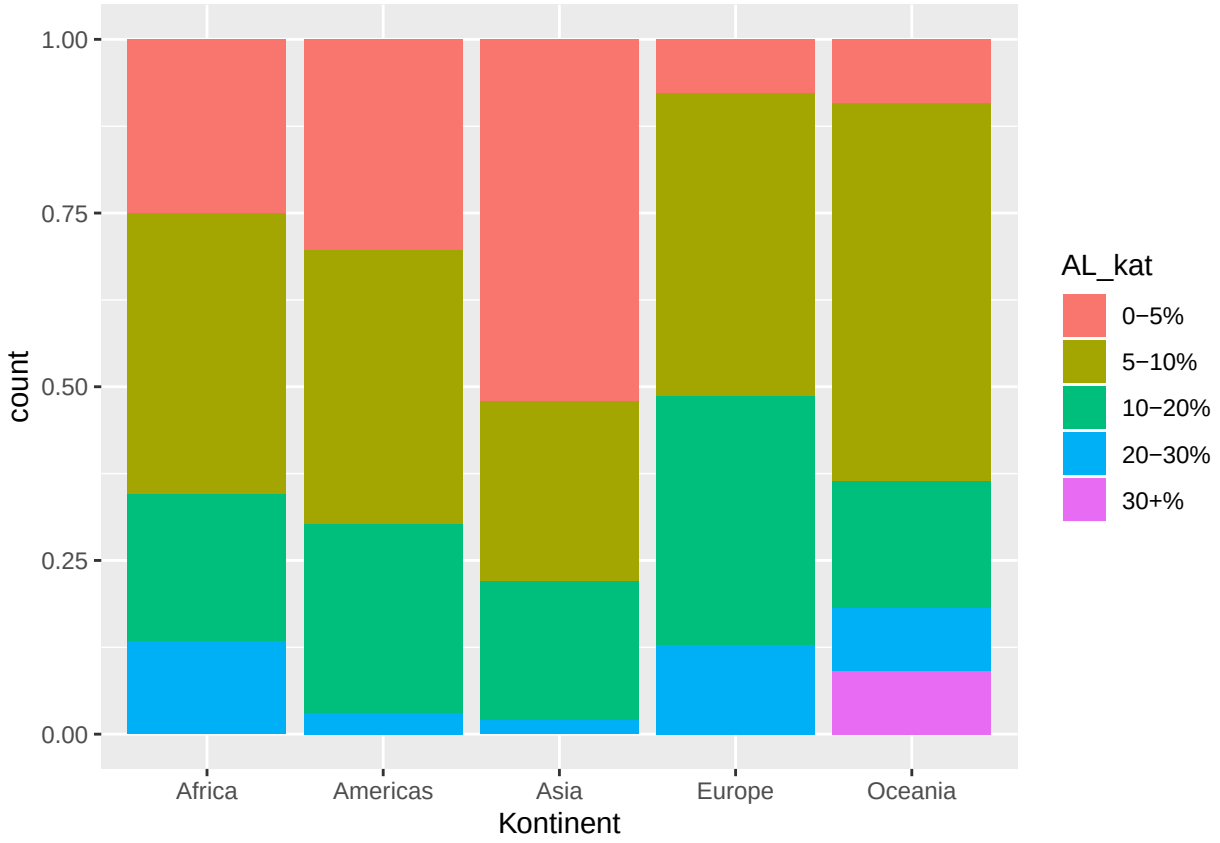


Beispiel 2: Balkendiagramm

```
g <- ggplot(wdi13[!is.na(wdi13$AL_kat),],
            aes(x = Kontinent, fill = AL_kat))
# Shortcut-Defintion der Ebene durch geom_bar()
g + geom_bar(position = "fill")
```



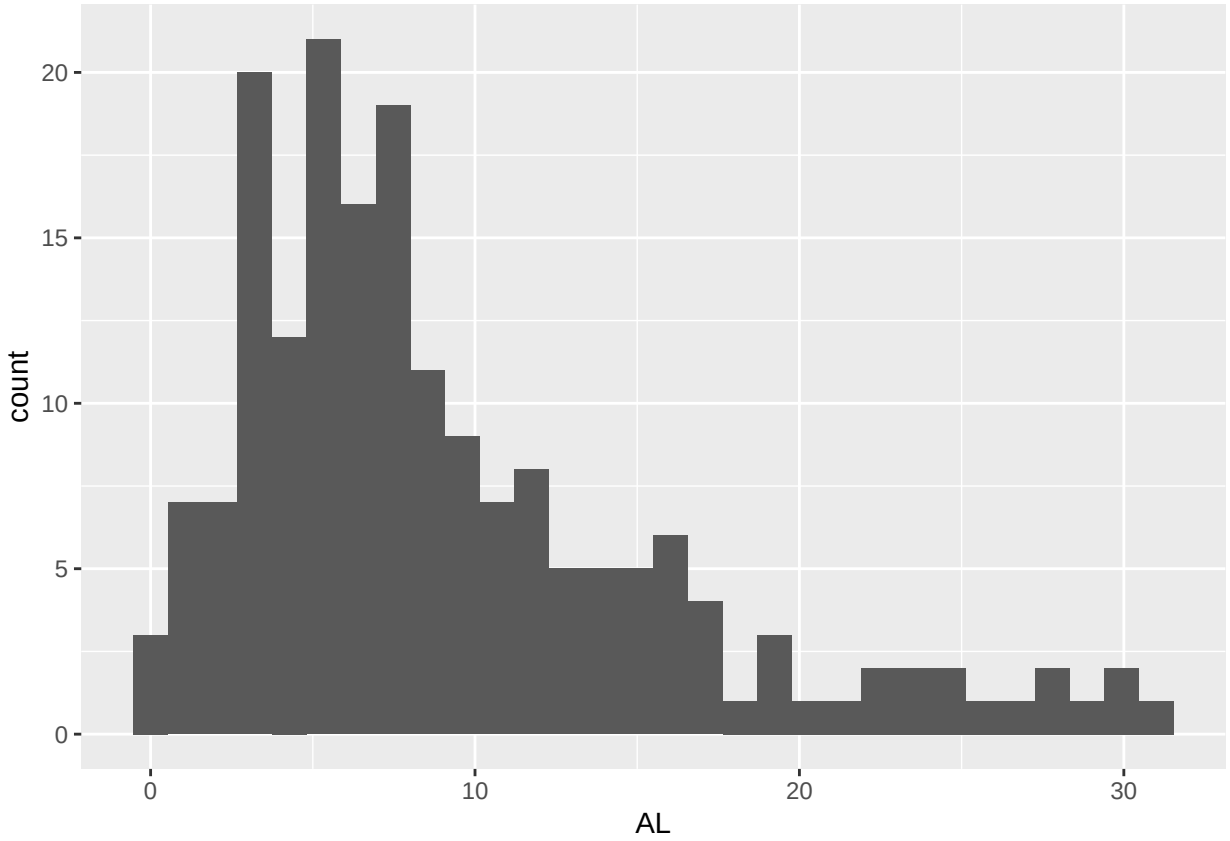

```
# Alternative: Selbe Ebene per Hand defineren
g + layer(geom = "bar", stat = "count", position = "fill")
```



Beispiel 3: Histogramm

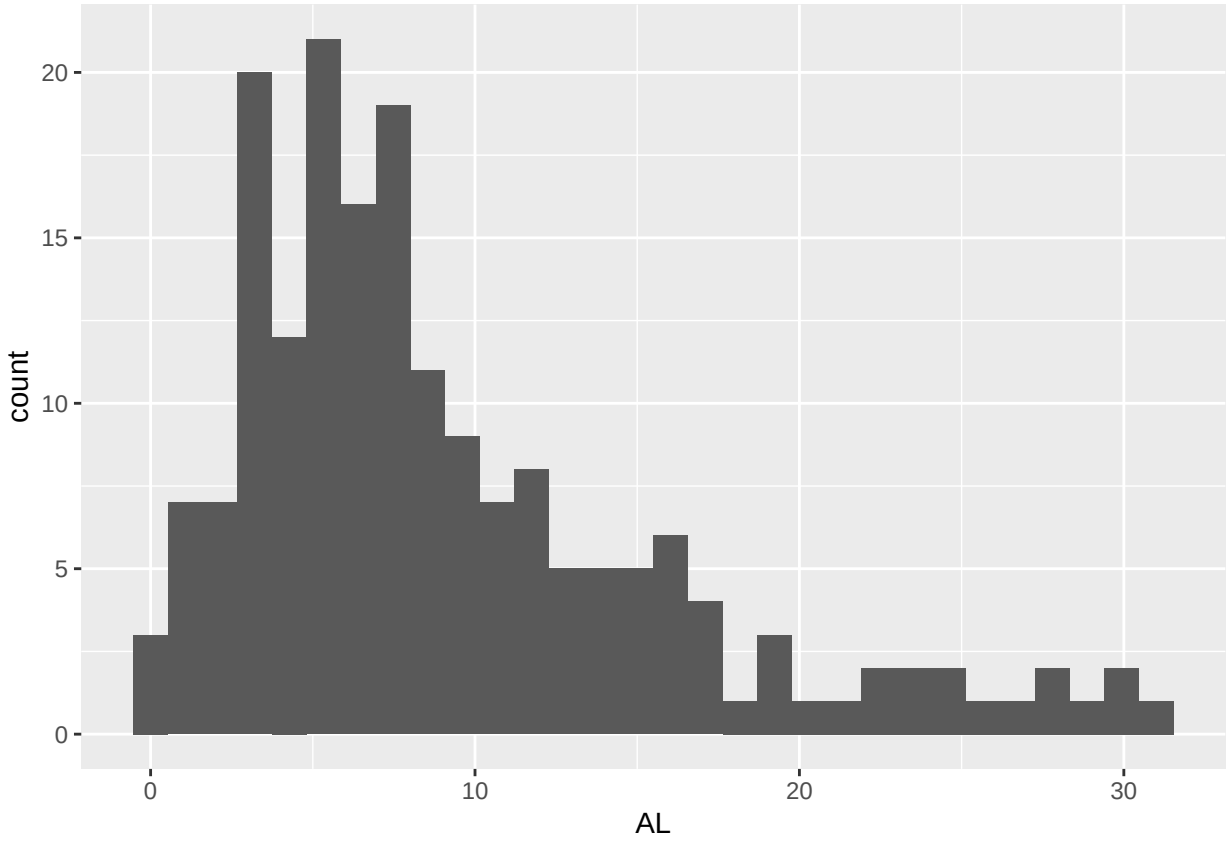
```
g <- ggplot(wdi13, aes(x = AL))
# Shortcut-Definton der Ebene durch geom_histogram()
g + geom_histogram()
```

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




```
# Alternative: Selbe Ebene per Hand defineren
g + layer(geom = "bar", stat = "bin", position = "stack")
```

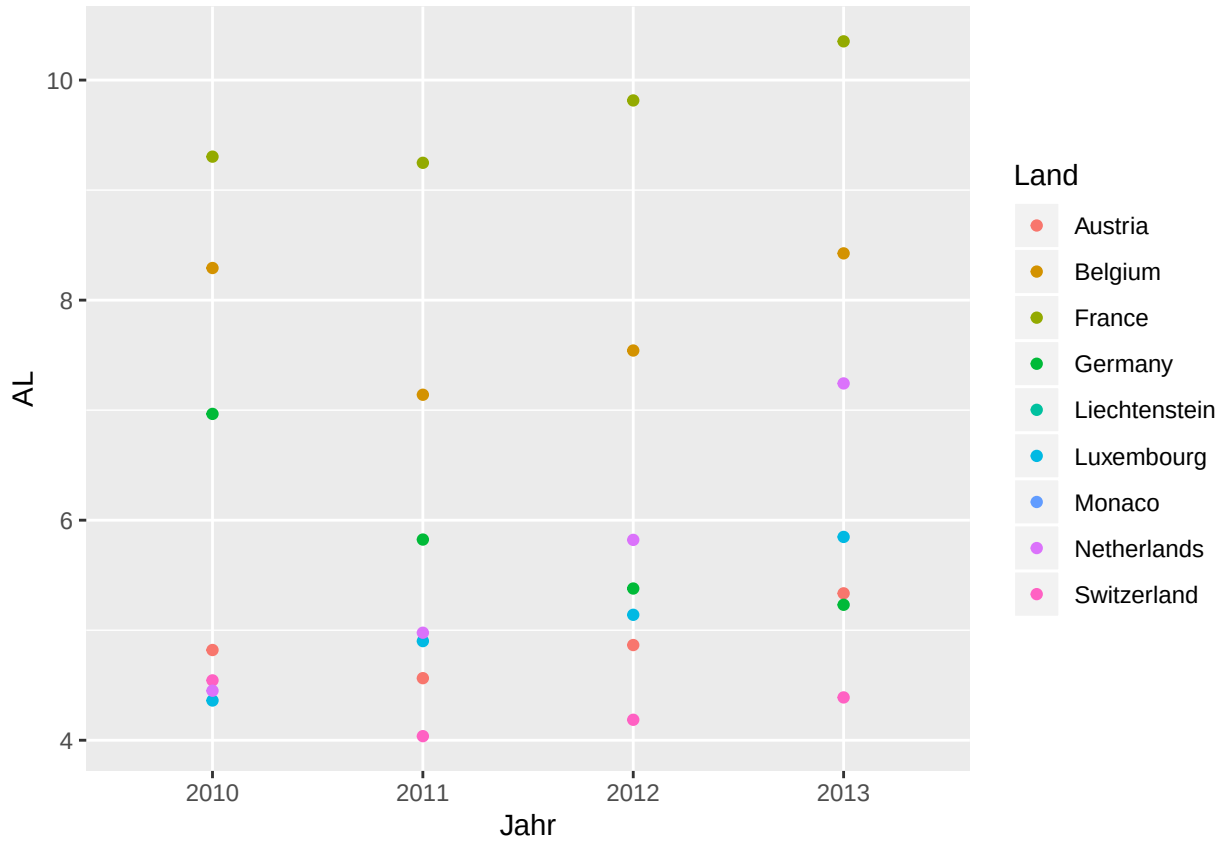
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



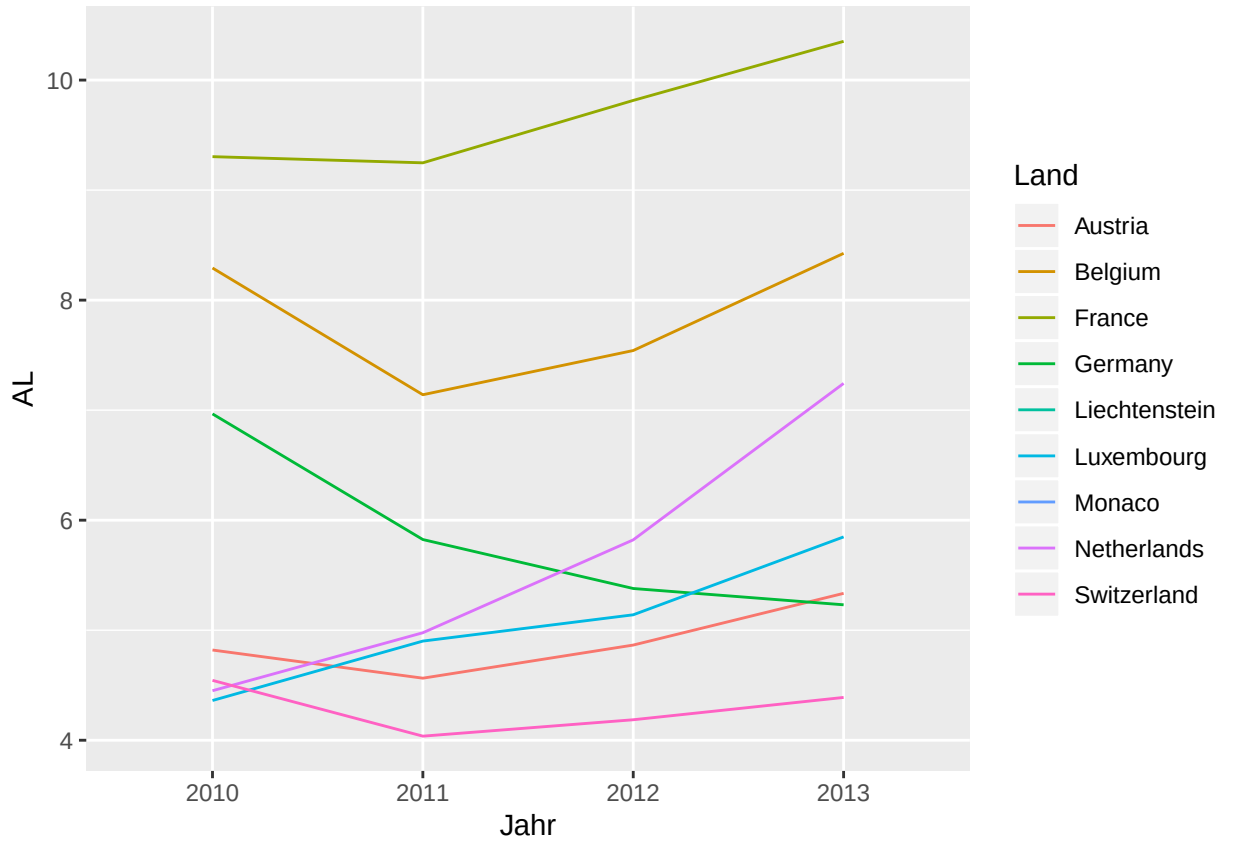
1b)

Elemente einer Ebene:

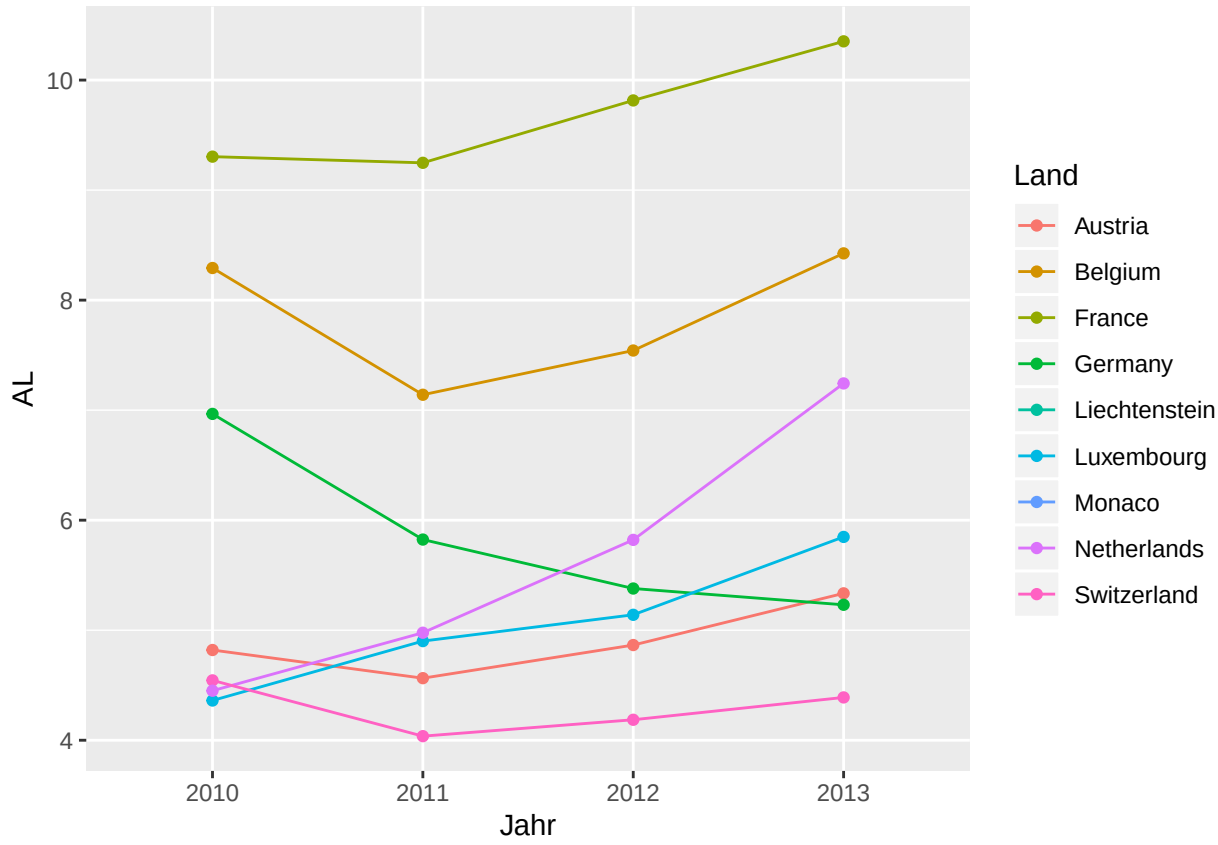
```
# data: selbsterklärend

# i) aesthetic mapping
g <- ggplot(wdi_westeuropa, aes(x = Jahr, y = AL, color = Land))
g + geom_point()
```



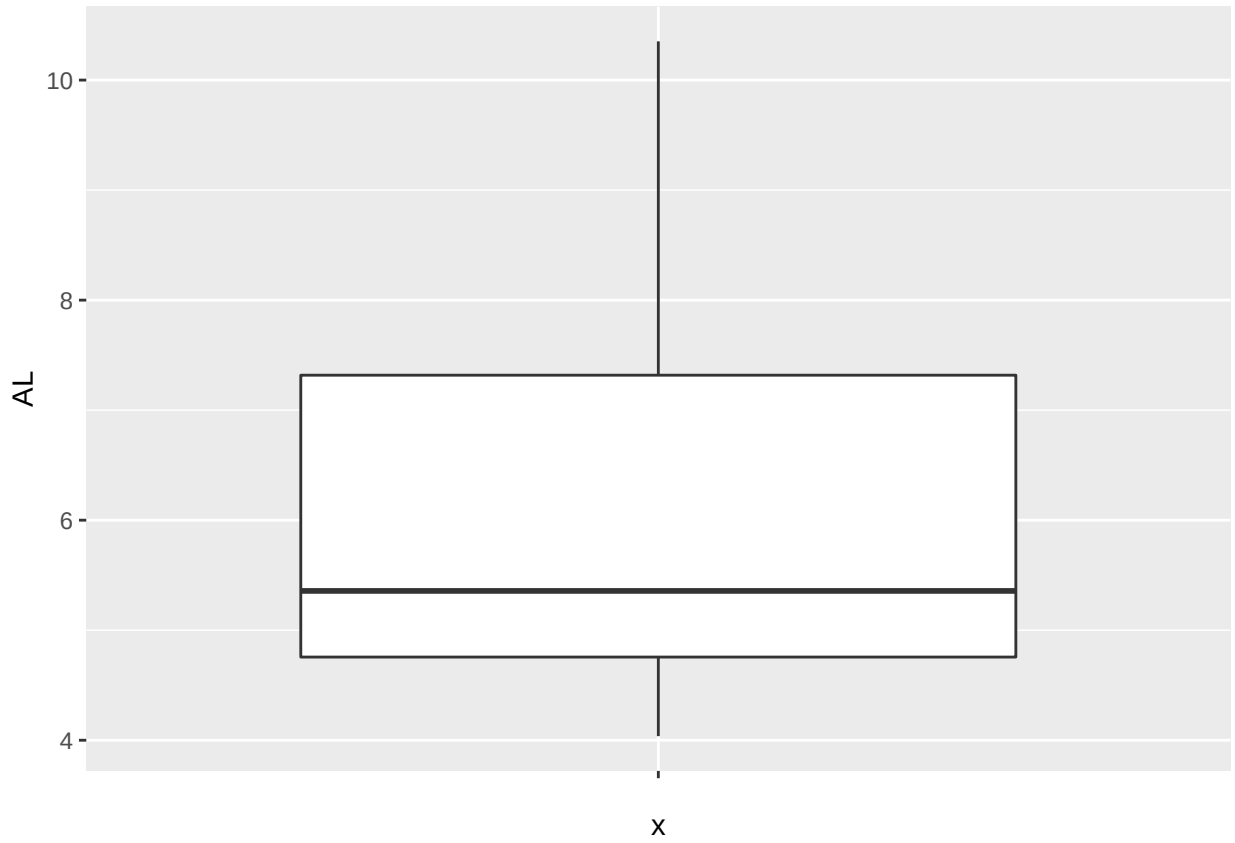

```
# ii) geom
g <- ggplot(wdi_westeuropa, aes(x = Jahr, y = AL, color = Land))
g + geom_line(aes(group = Land))
```



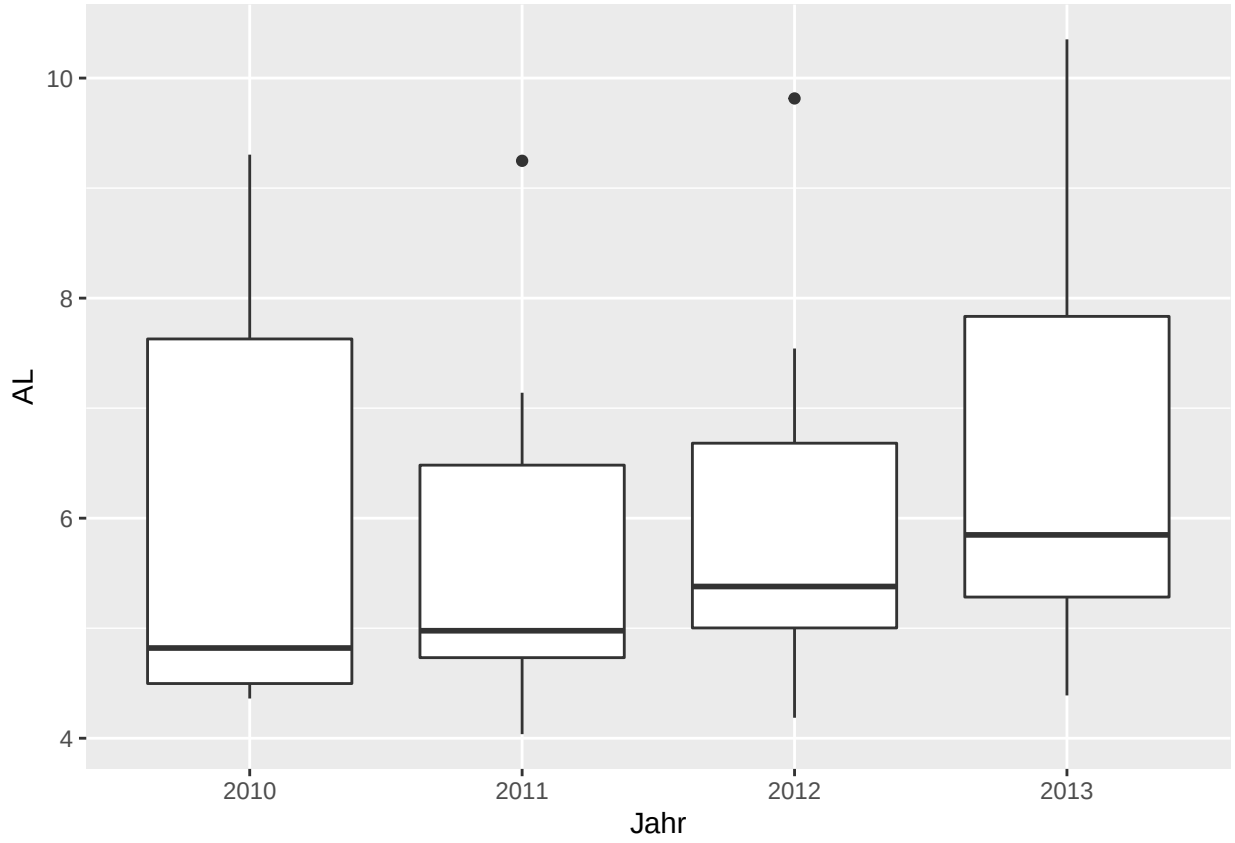

```
g + geom_point() + geom_line(aes(group = Land))
```



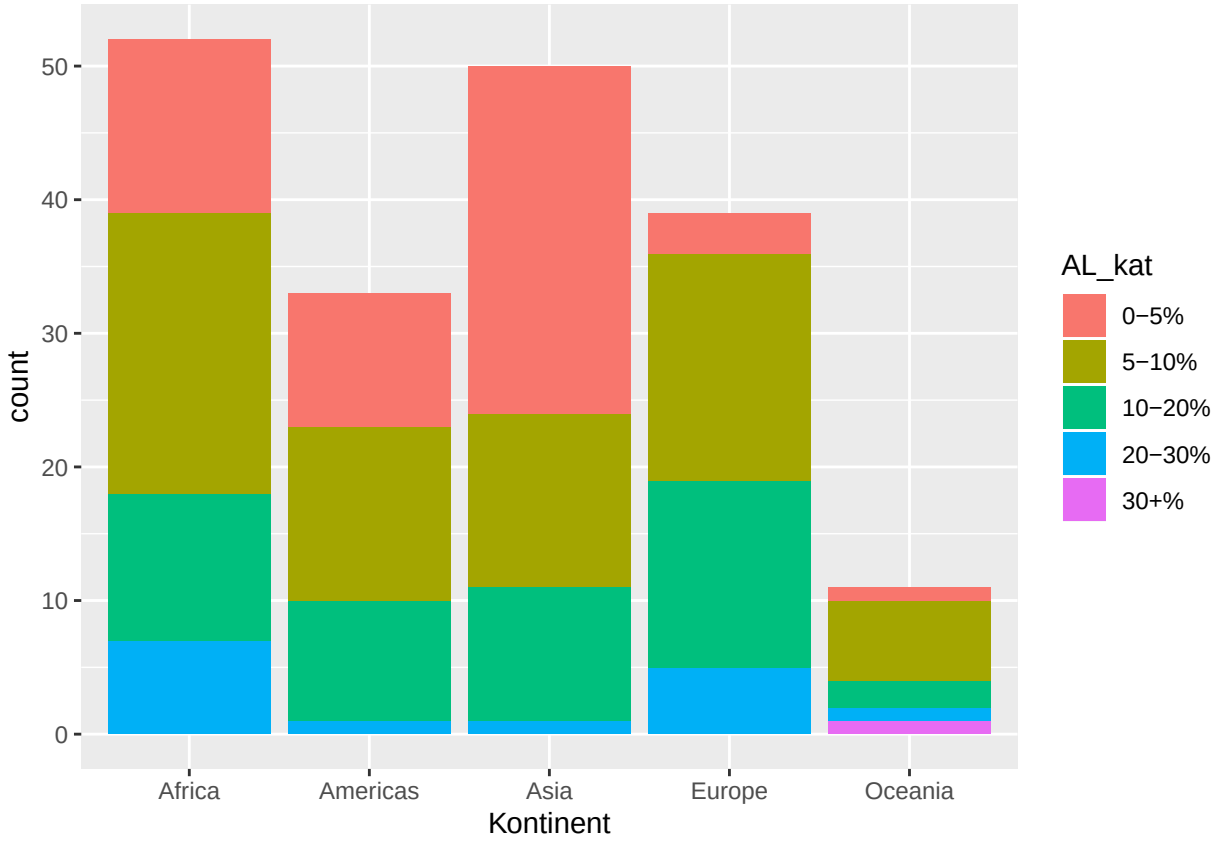

```
ggplot(wdi_westeuropa, aes(x = "", y = AL)) + geom_boxplot()
```



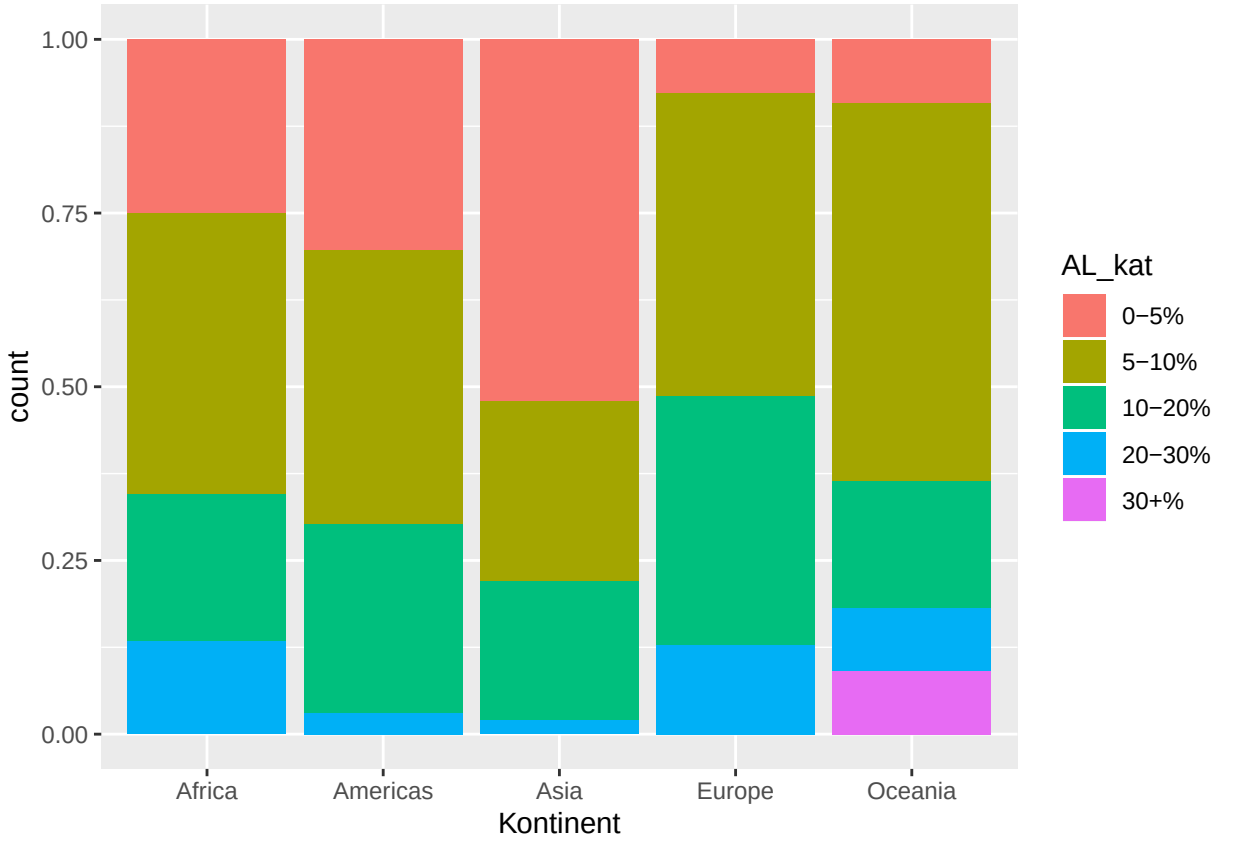

```
ggplot(wdi_westeuropa, aes(x = Jahr, y = AL)) + geom_boxplot()
```



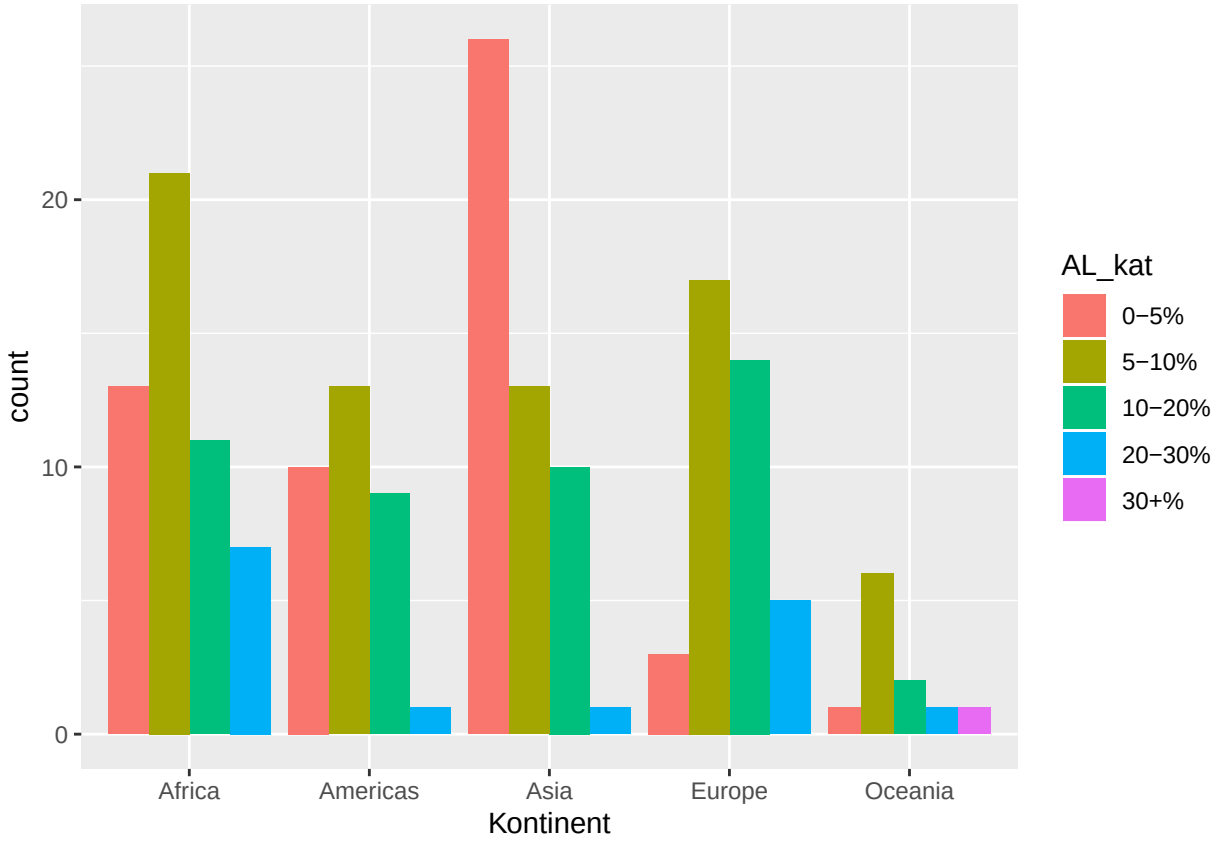

```
# iii) stat
# Scatterplot:
# ?geom_point -> stat = "identity"
# Balkendiagramm:
# ?geom_count -> stat = "sum" (stat_sum())
# Histogramm:
# ?geom_histogram -> stat = "bin" (stat_bin())

# iv) position
g <- ggplot(wdi13[!is.na(wdi13$AL_kat),],
            aes(x = Kontinent, fill = AL_kat))
g + geom_bar(position = "stack")
```



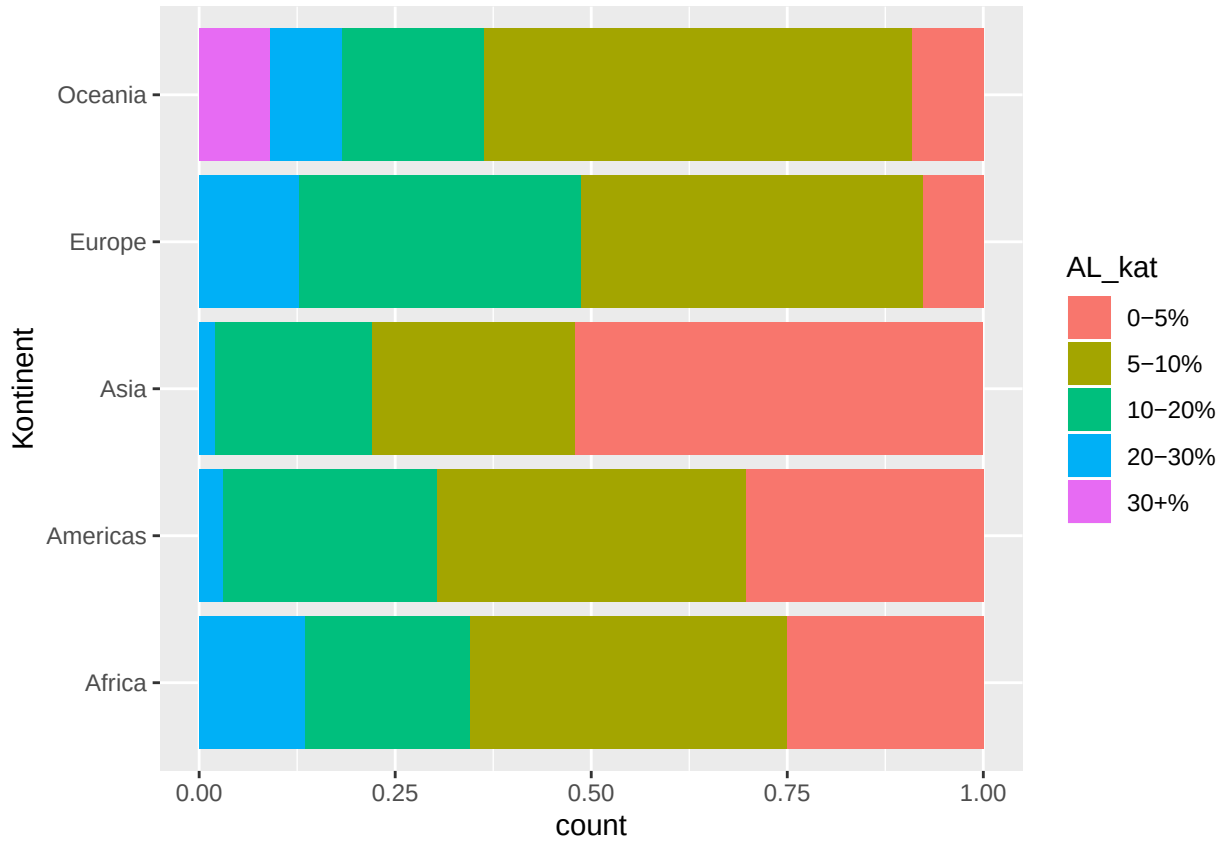

```
g + geom_bar(position = "fill")
```




```
g + geom_bar(position = "dodge")
```




```
# Horizontal
g + geom_bar(position = "fill") + coord_flip()
```

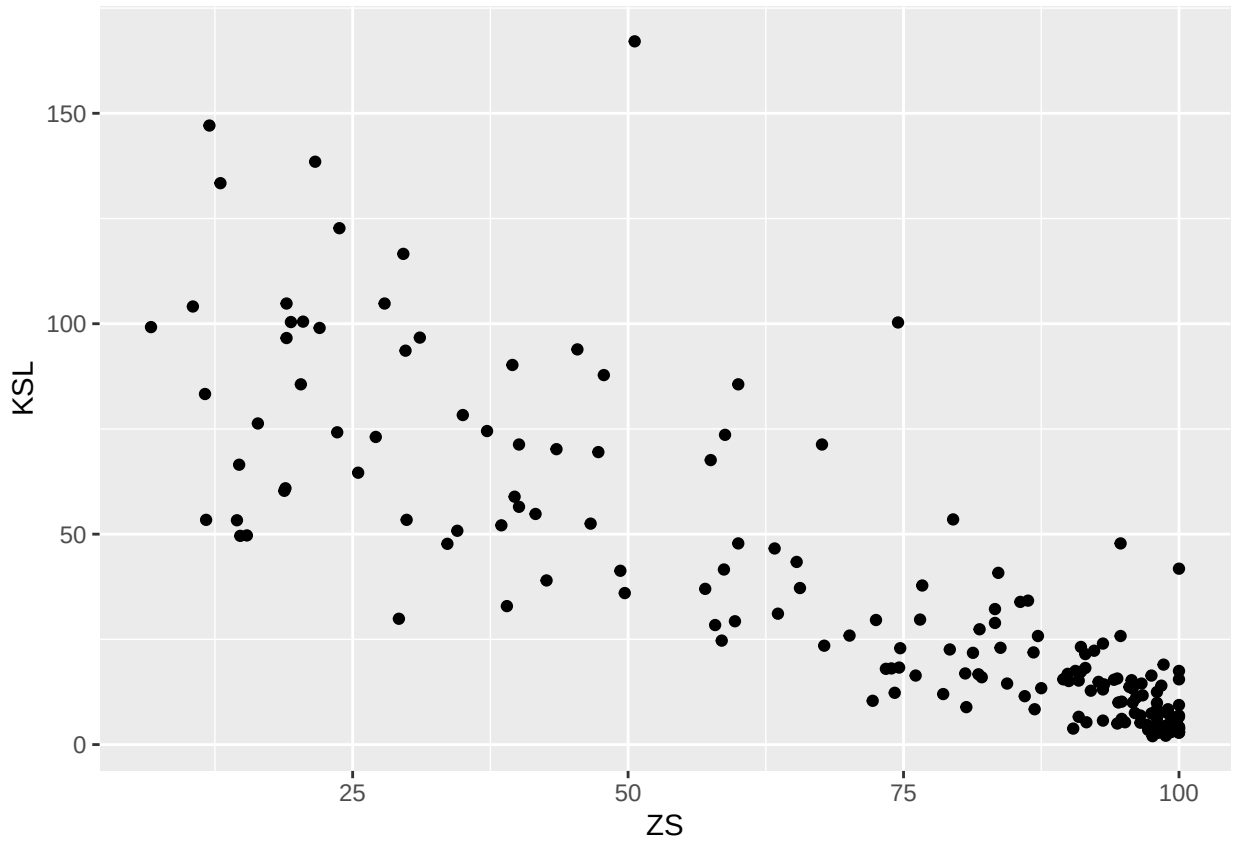


1c)

Anmerkungen zu Themes:

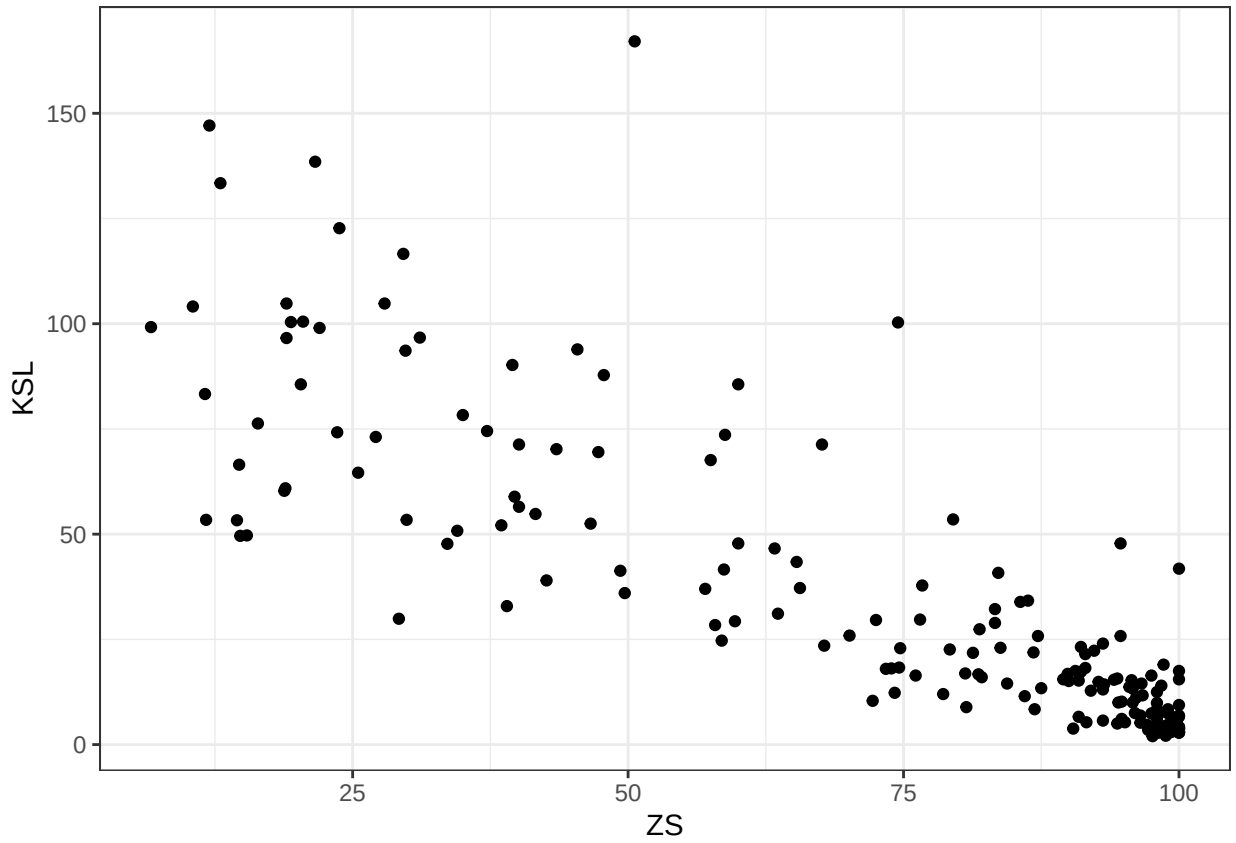
Im ggplot2-Paket verfügbare Themes finden sich in `?theme_classic`. Daneben gibt es auch verschiedene Pakete wie ggthemes oder ggthemr.

```
g <- ggplot(wdi13, aes(x = ZS, y = KSL)) +
  geom_point()

# i) Themes
# ?theme
# -> umfassende Einstellungsmöglichkeiten
# ?theme_minimal
# -> Einige vordefinierte Themes zur einfachen Benutzung
# ?theme_update
g
```

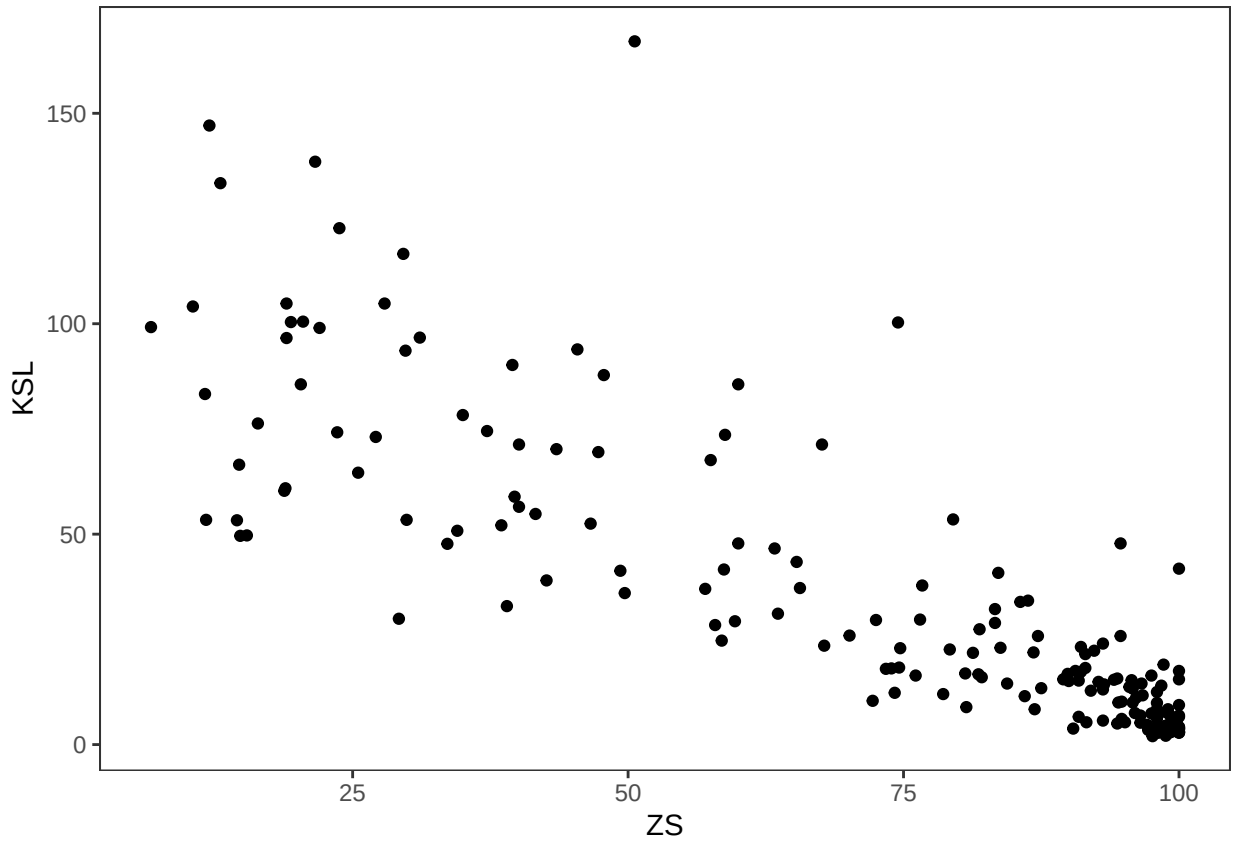



```
g + theme_void()
```

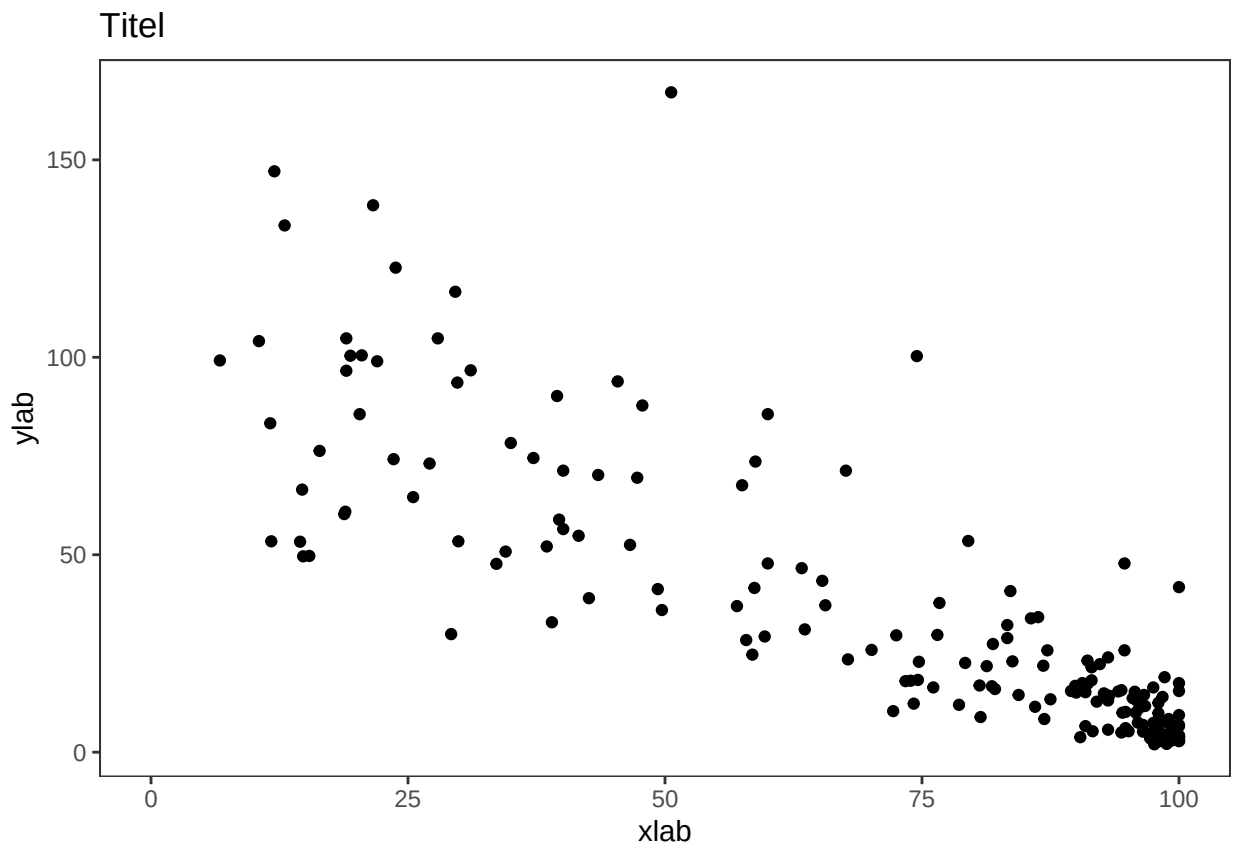
```
g + theme_bw()
```



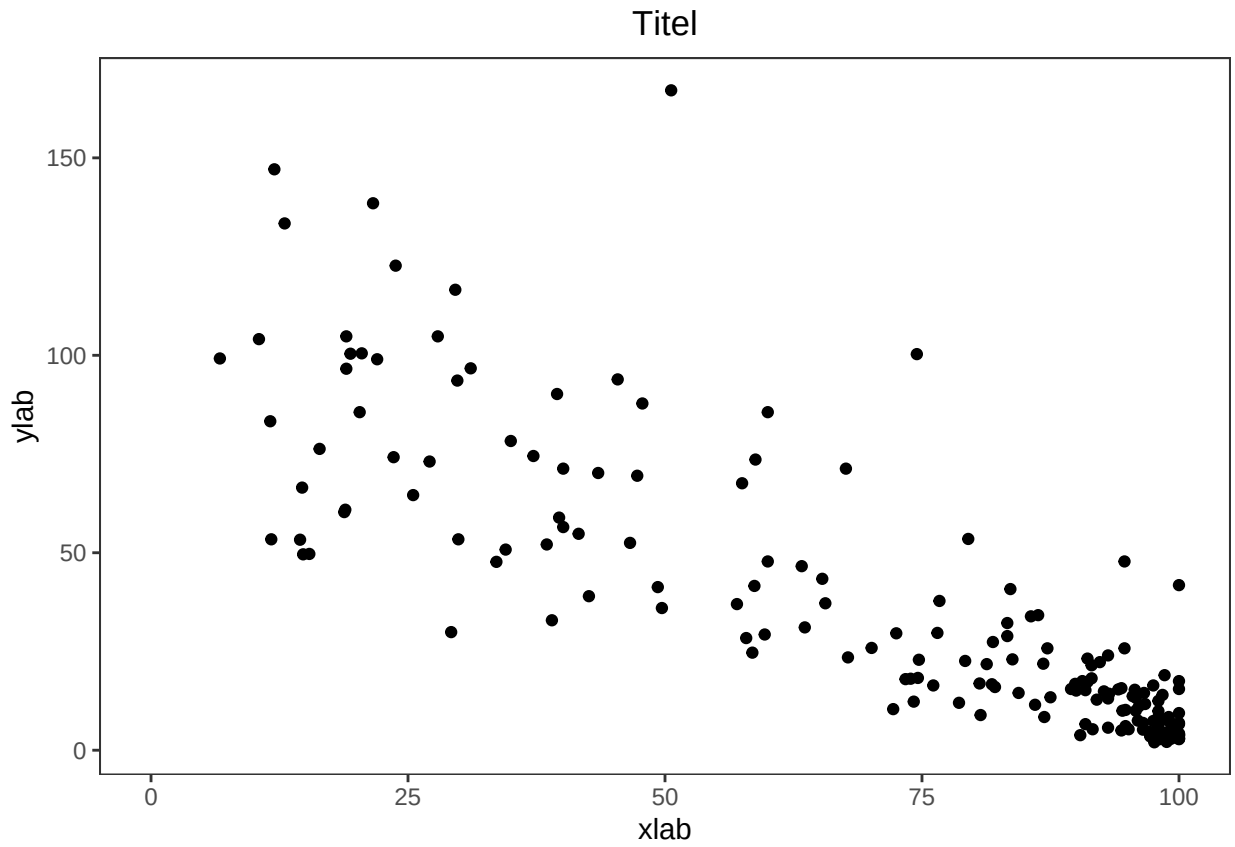

```
g + theme_bw() + theme(panel.grid = element_blank())
```



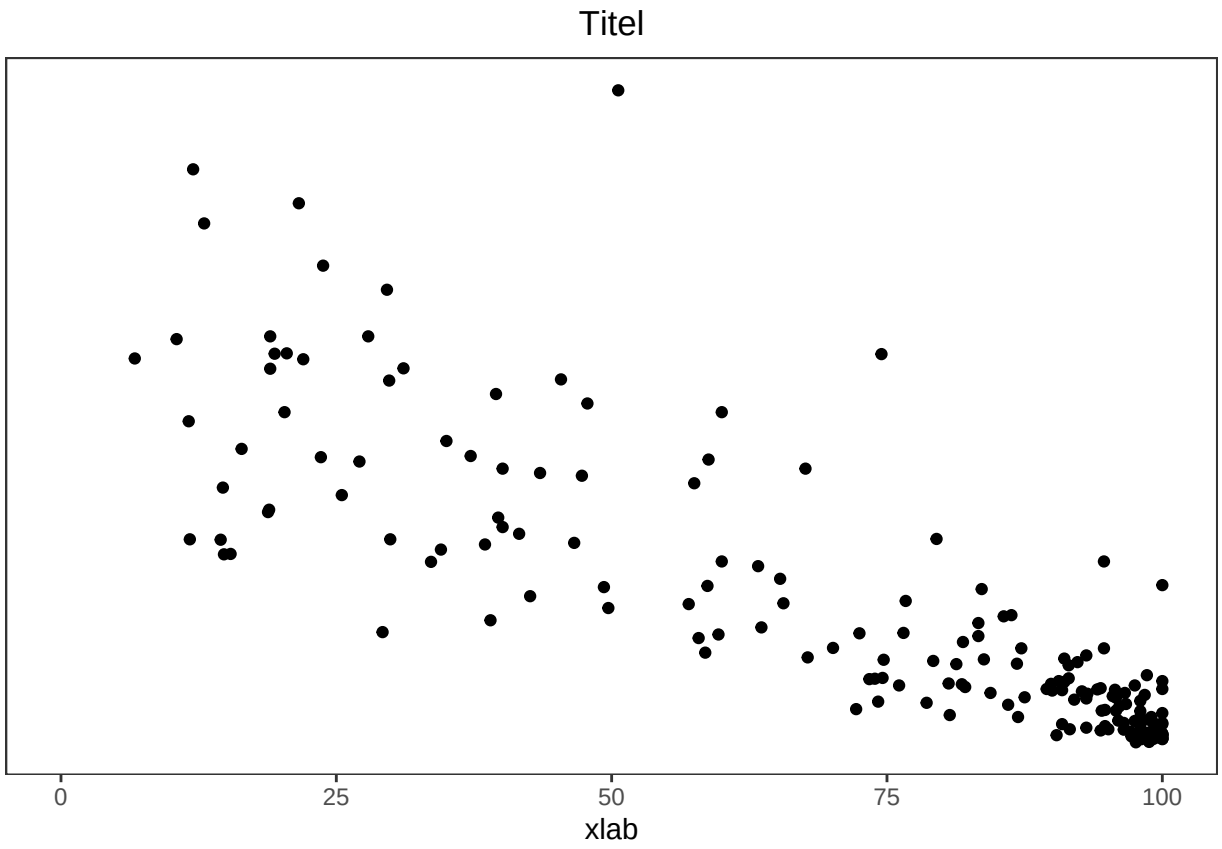

```
# Festlegen des globalen Themes
theme_set(theme_bw() + theme(panel.grid = element_blank()))


# ii) Plot- & Achsenbeschriftungen, Panel
# Beschriftungen
g2 <- g + ggtitle("Titel") + xlab("xlab") + ylab("ylab") + xlim(c(0,100))
g2
```



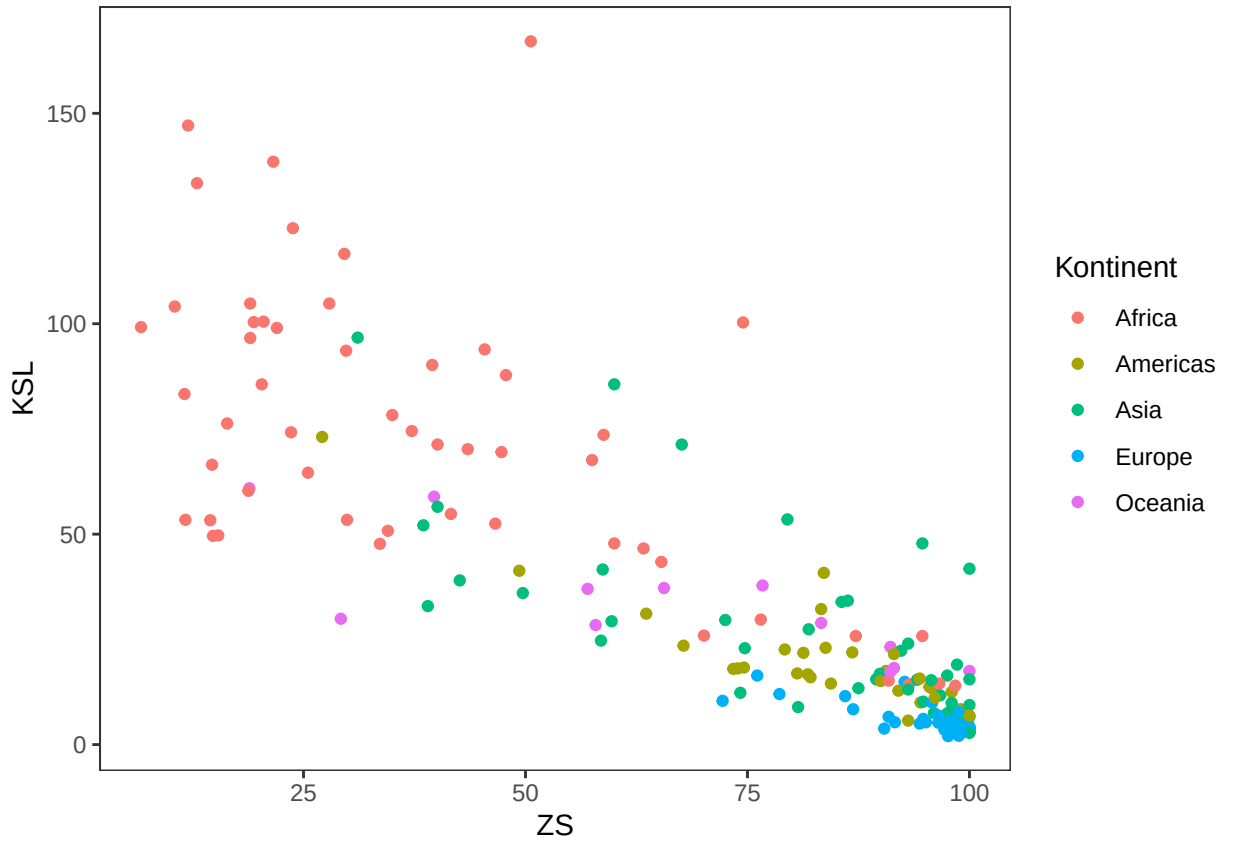

```
g2 <- g2 + theme(plot.title = element_text(hjust = 0.5))
g2
```



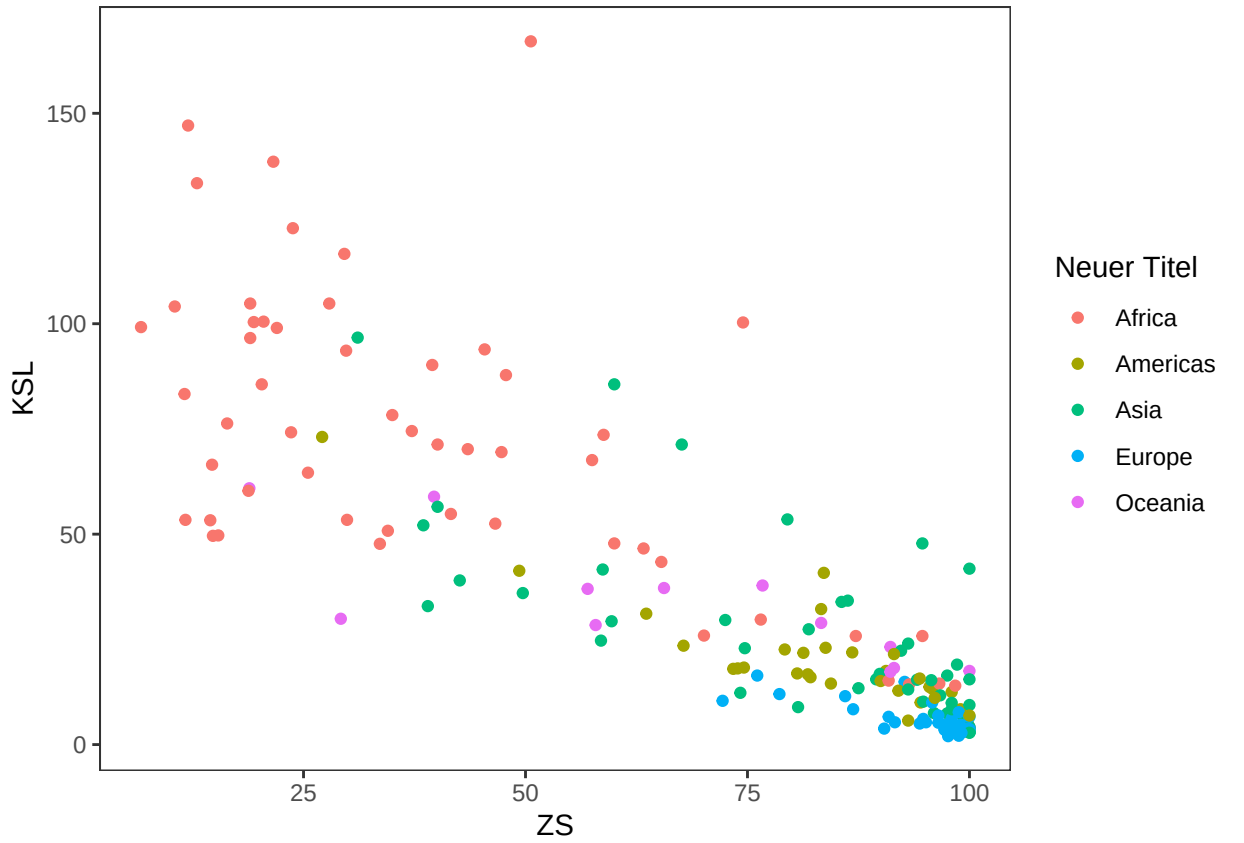

```
g2 + theme(axis.text.y = element_blank(),
           axis.ticks.y = element_blank(),
           axis.title.y = element_blank())
```



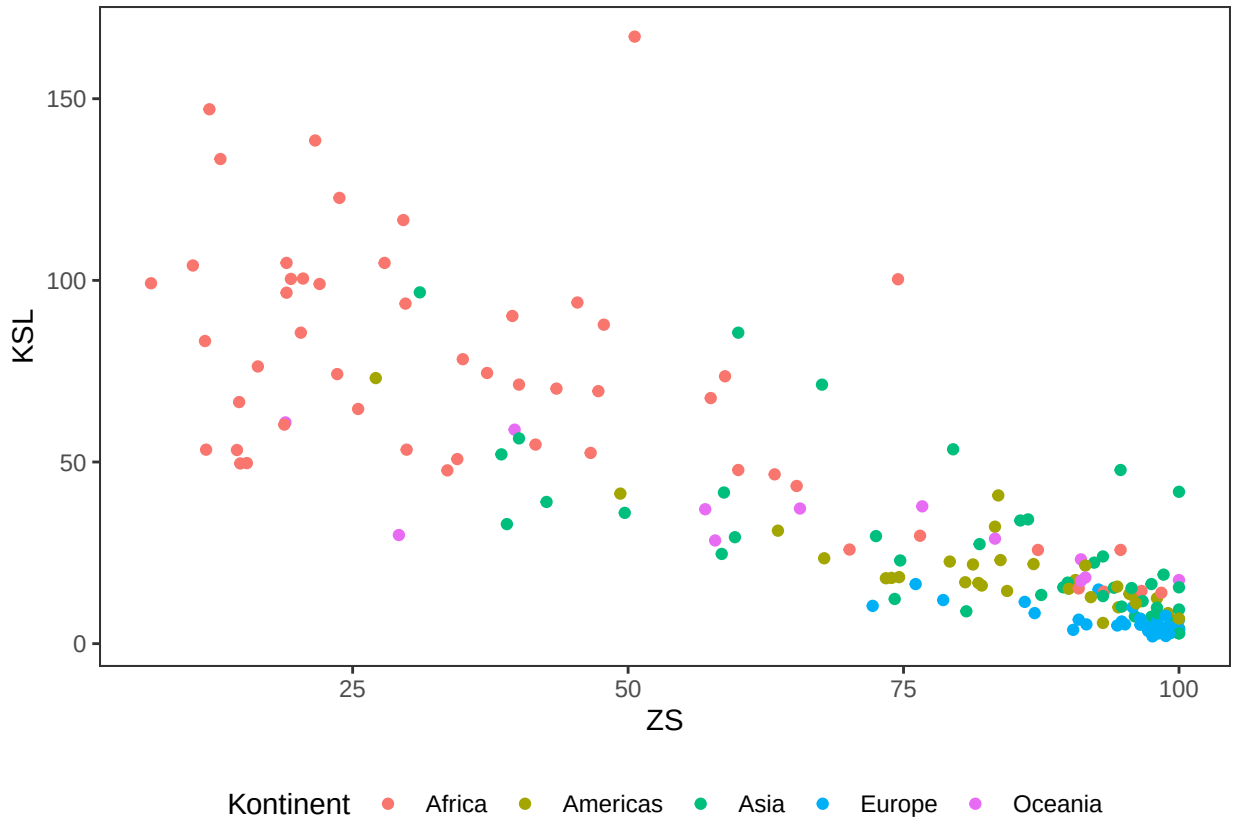

```
# Panel
# -> siehe Theme-Definton in (i)


# iii) Legende, Farben ändern
g <- ggplot(wdi13, aes(x = ZS, y = KSL, color = Kontinent)) +
  geom_point()
g
```



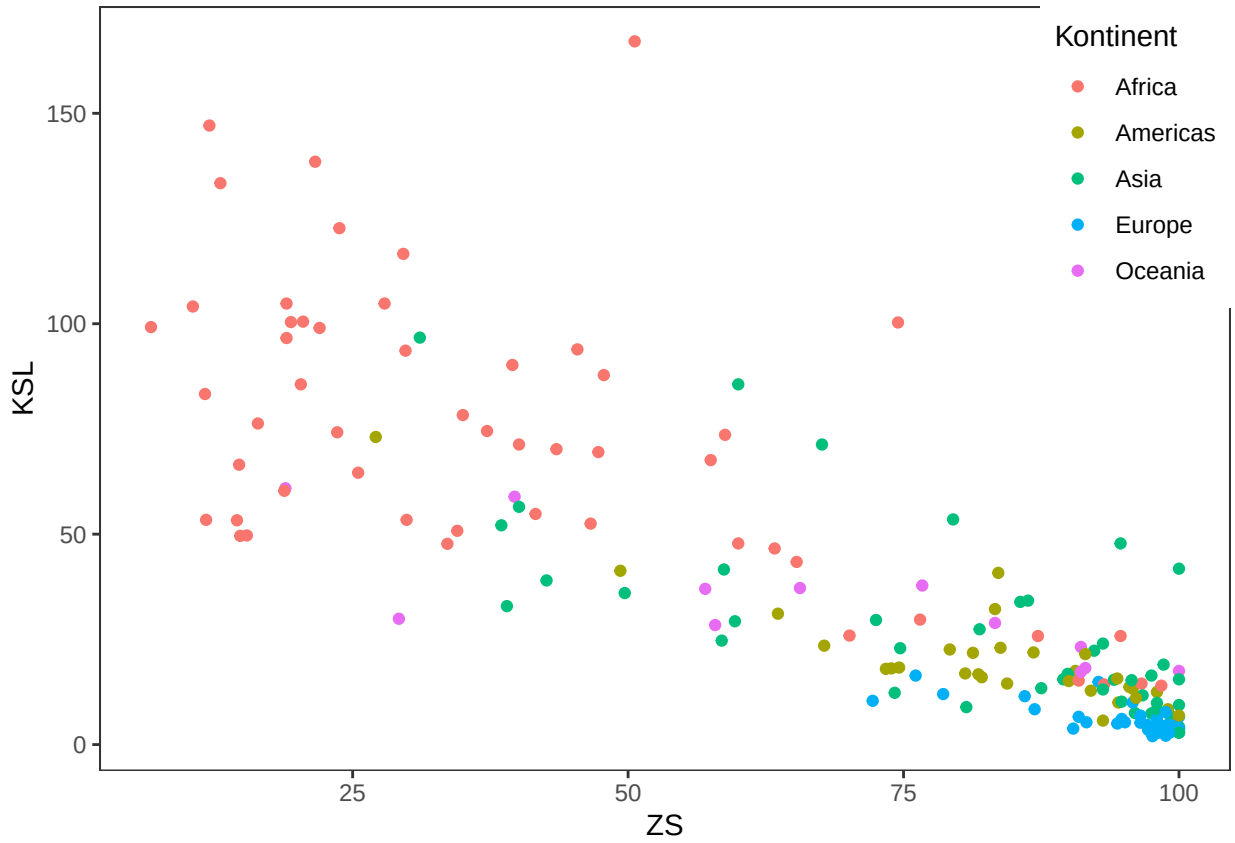

```
# Titel
g + guides(color = guide_legend(title = "Neuer Titel"))
```



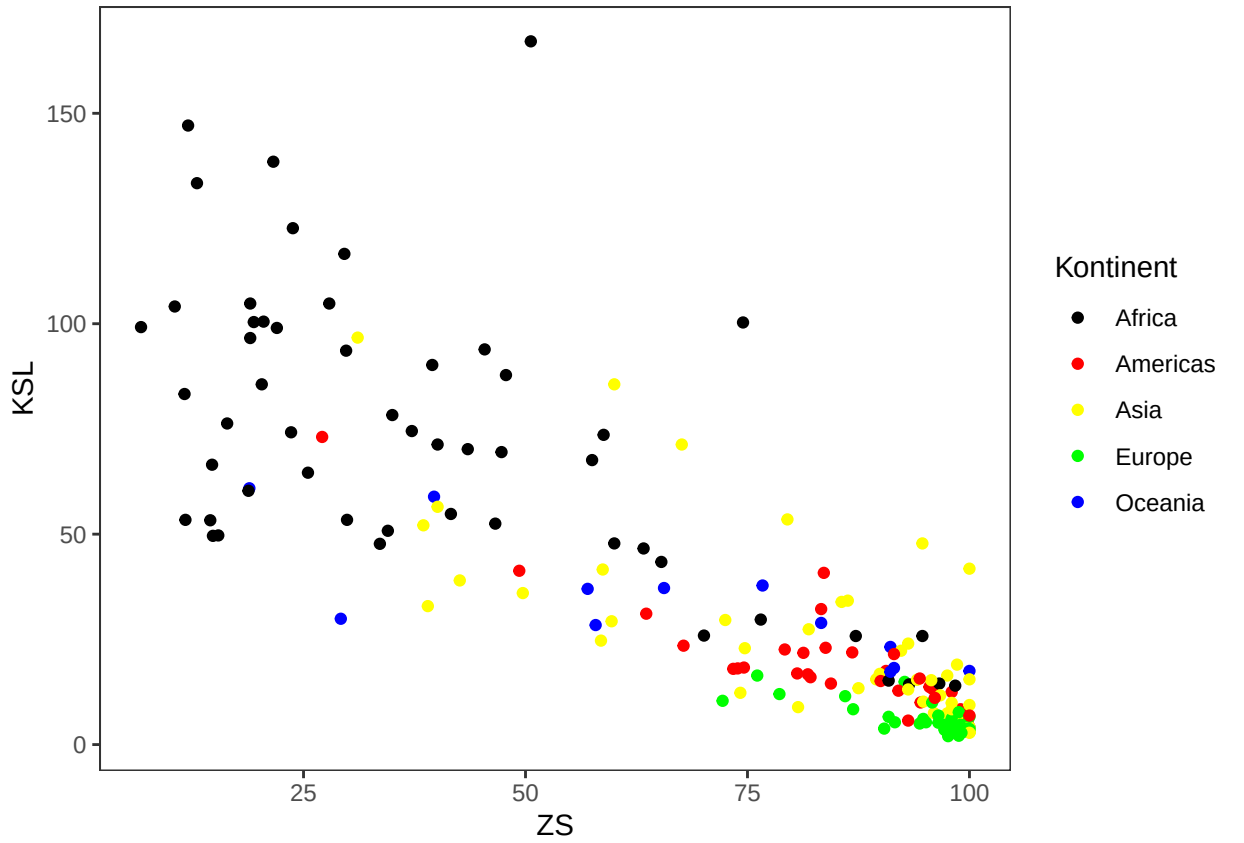

```
# Position
g + theme(legend.position = "bottom")
```



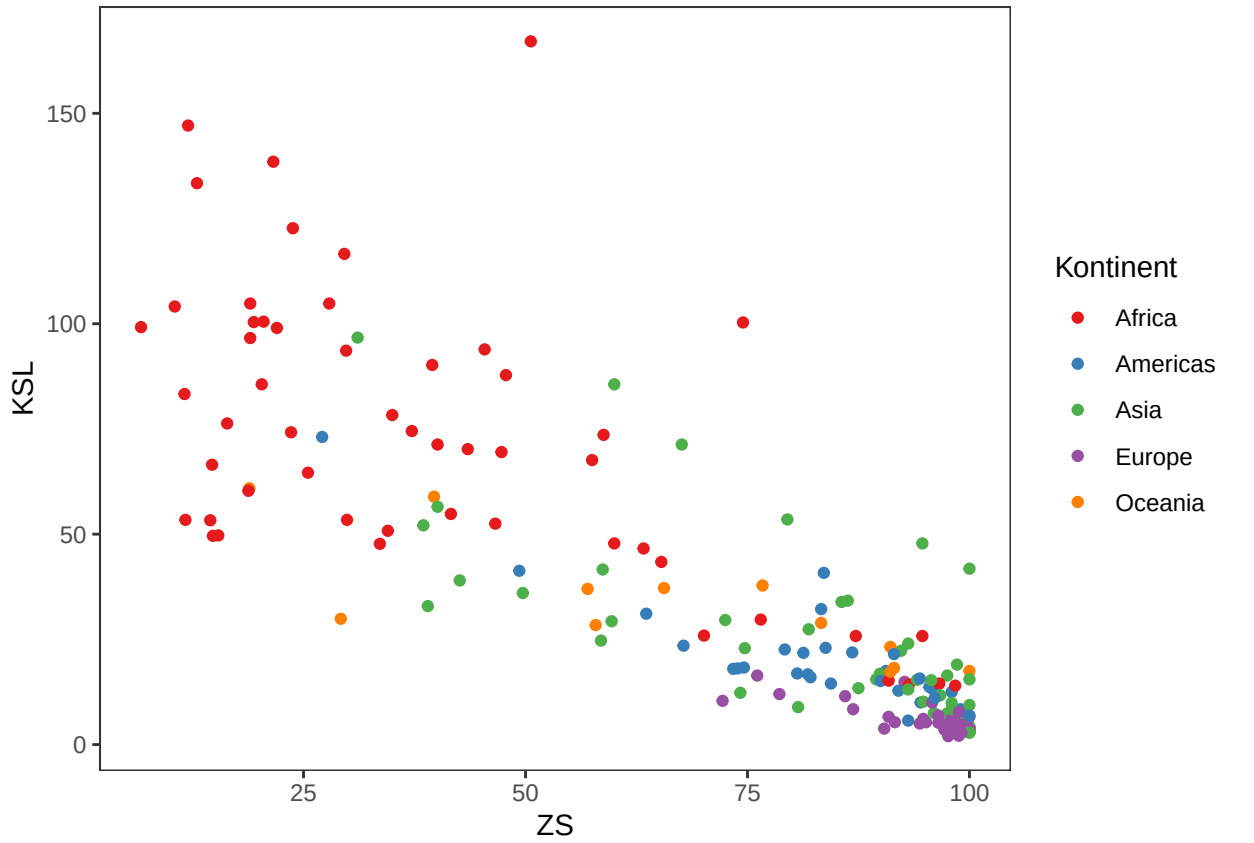

```
g + theme(legend.justification=c(1,1),
          legend.position=c(1,1))
```



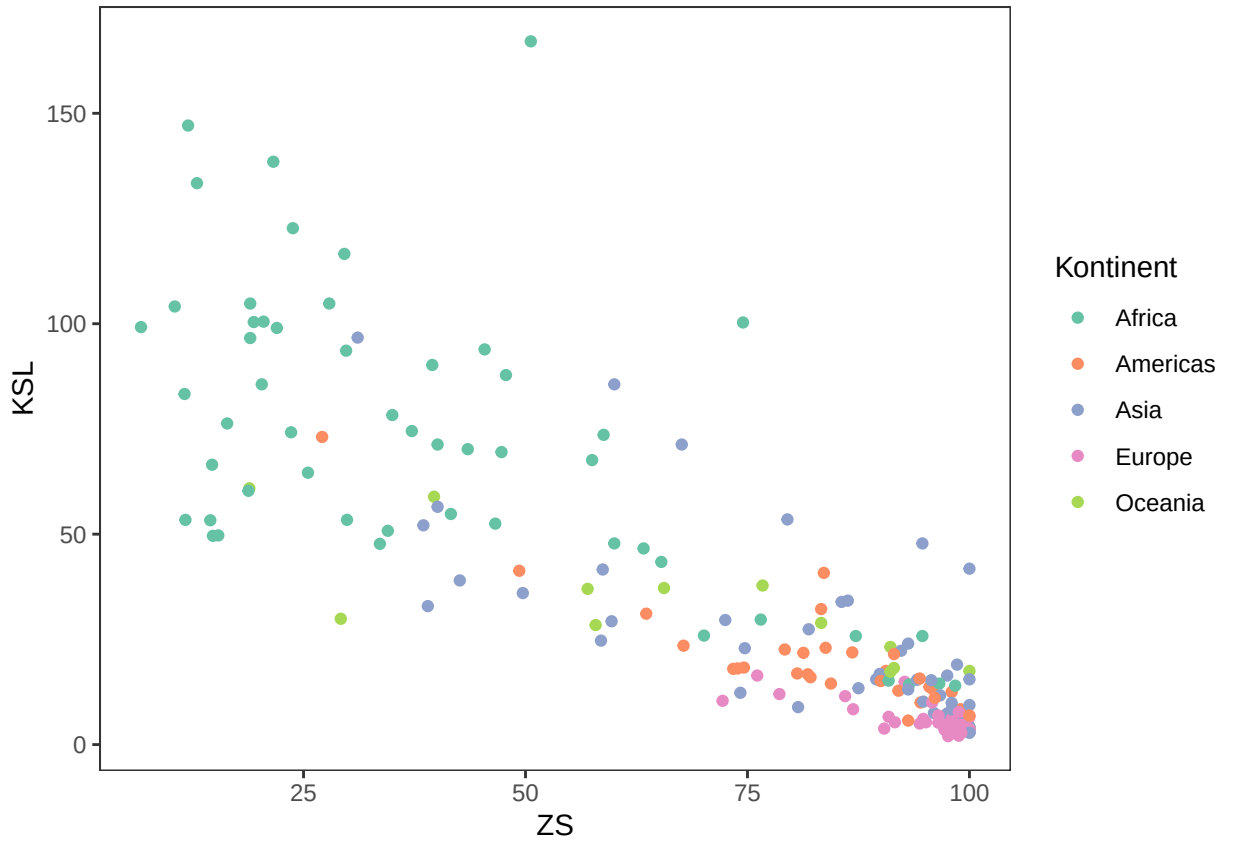

```
# Farben
g + scale_color_manual(values = c("black","red","yellow","green","blue"))
```



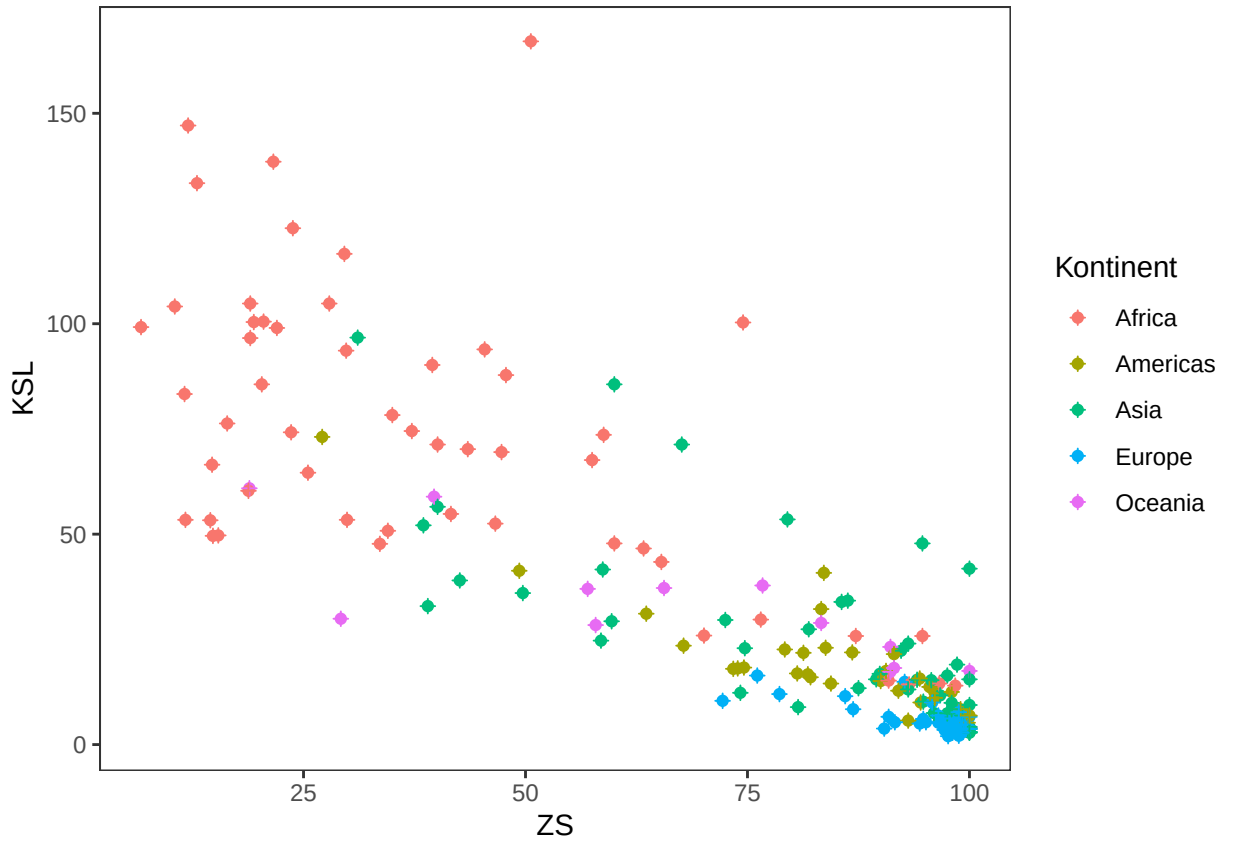

```
# RColorBrewer zur Verwendung vordefinierter Sets
g + scale_color_brewer(palette = "Set1")
```



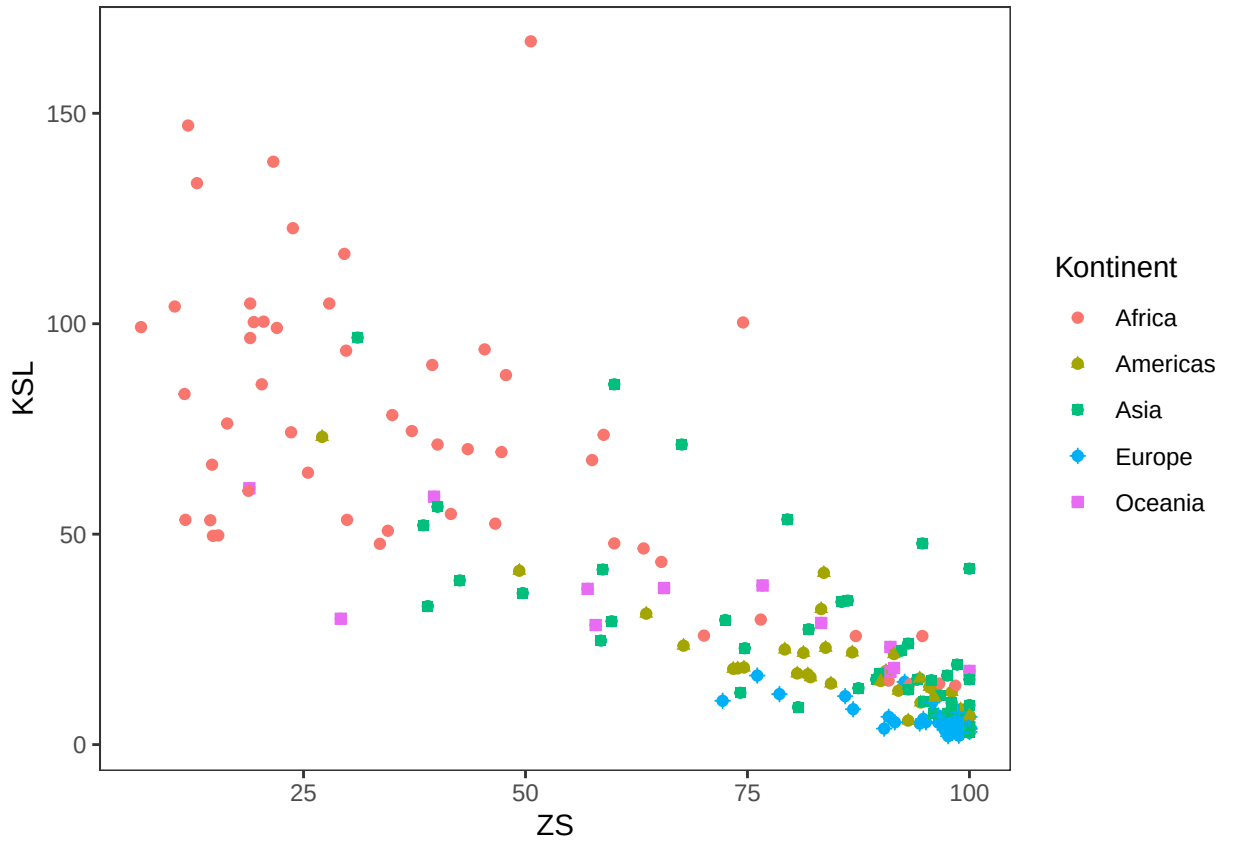

```
g + scale_color_brewer(palette = "Set2")
```



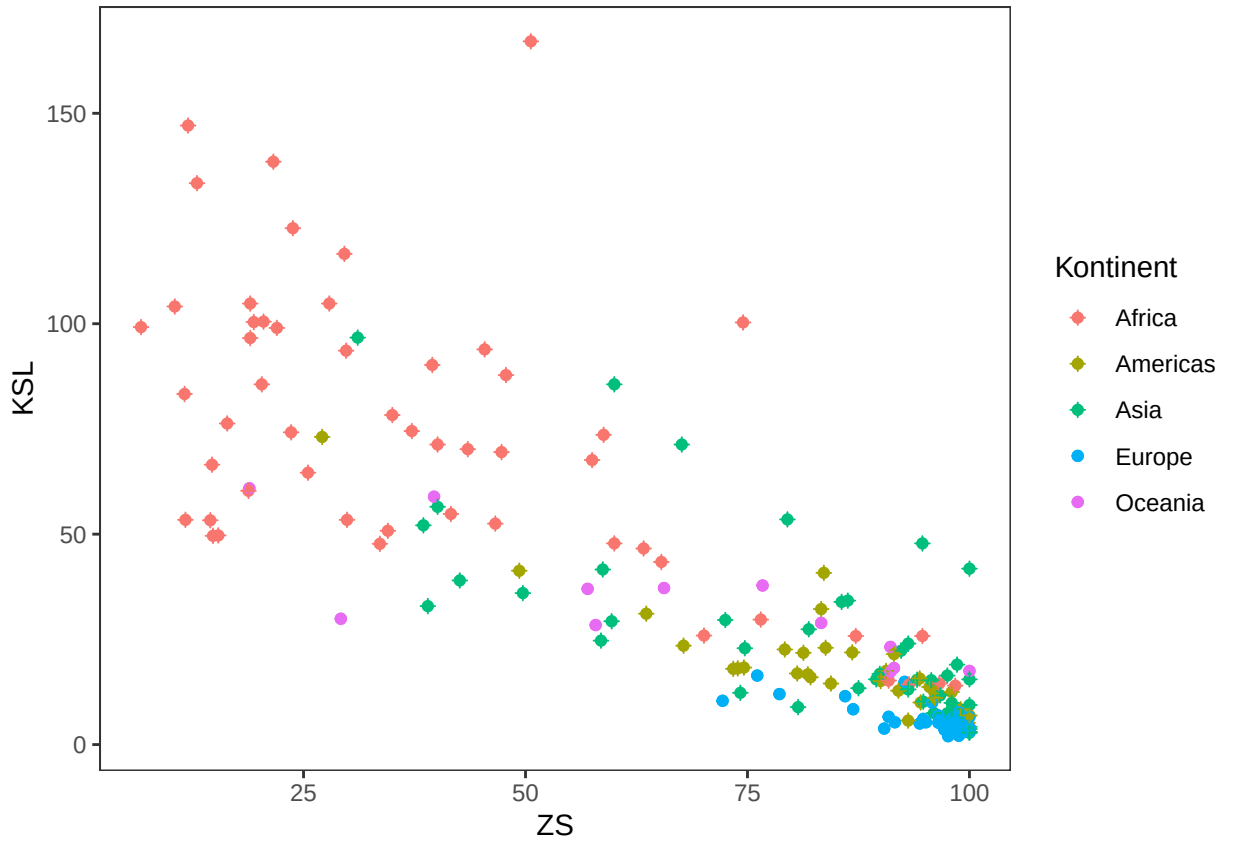

```
# Symbole
g + geom_point(shape = 3)
```



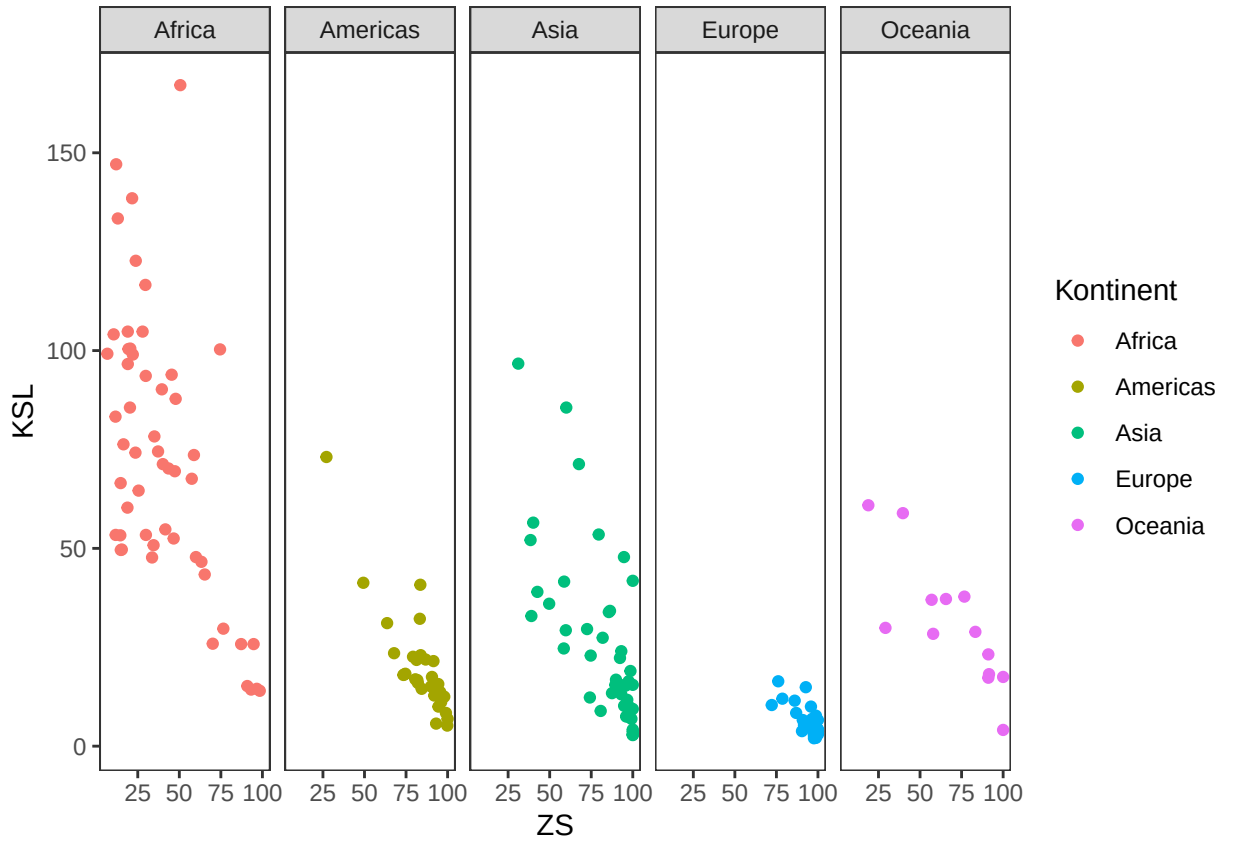

```
g2 <- g + geom_point(aes(shape = Kontinent))
g2
```



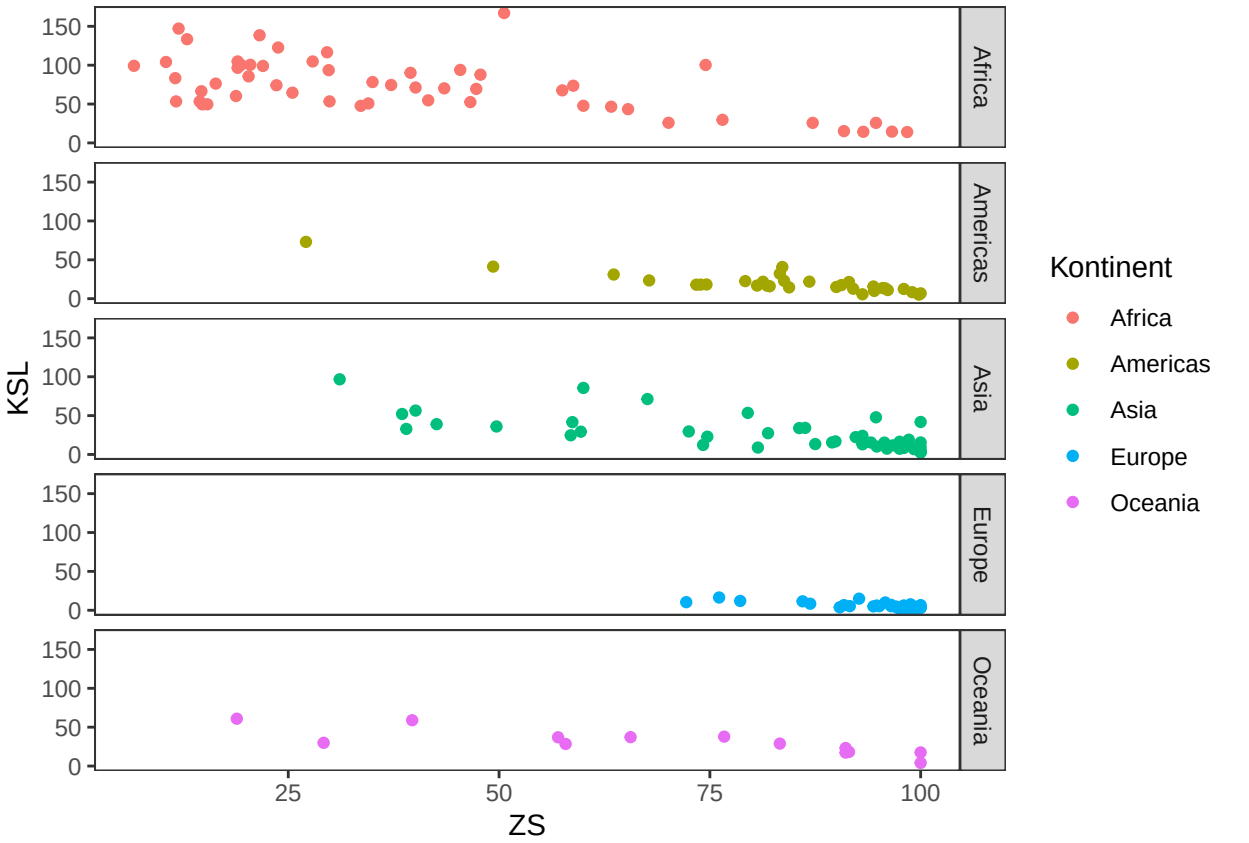

```
g2 + scale_shape_manual(values = c(3,3,3,1,1))
```



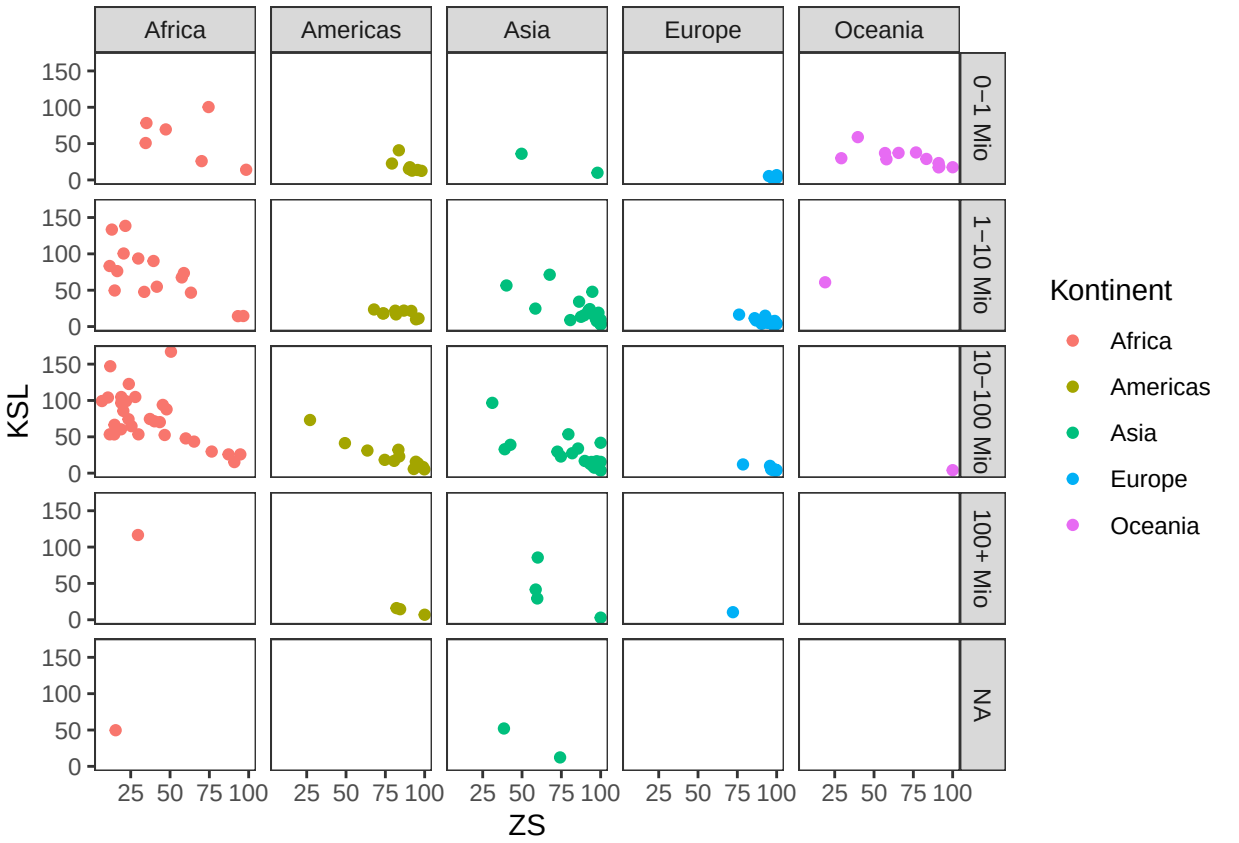

```
# iv) Faceting
g + facet_grid(. ~ Kontinent)
```




```
g + facet_grid(Kontinent ~ .)
```



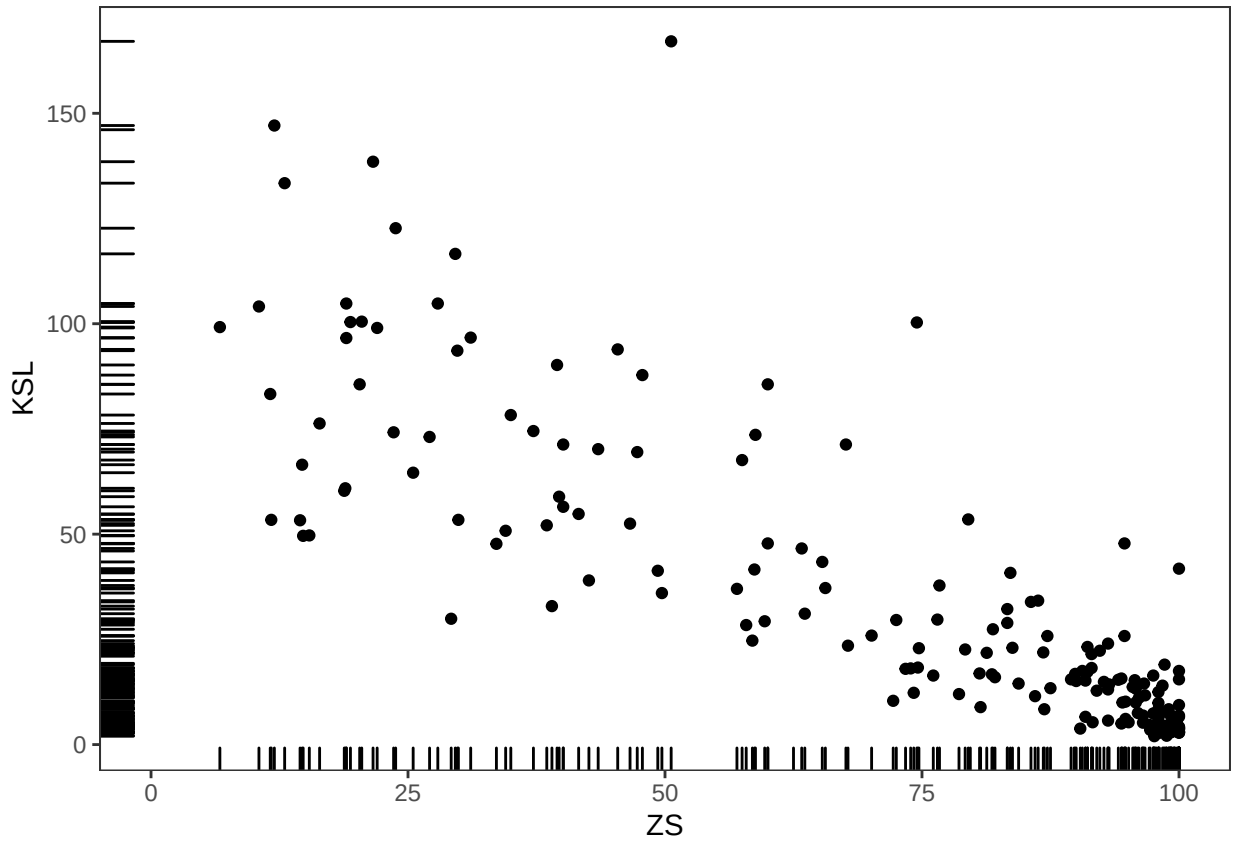

```
g + facet_grid(Bev_kat ~ Kontinent)
```



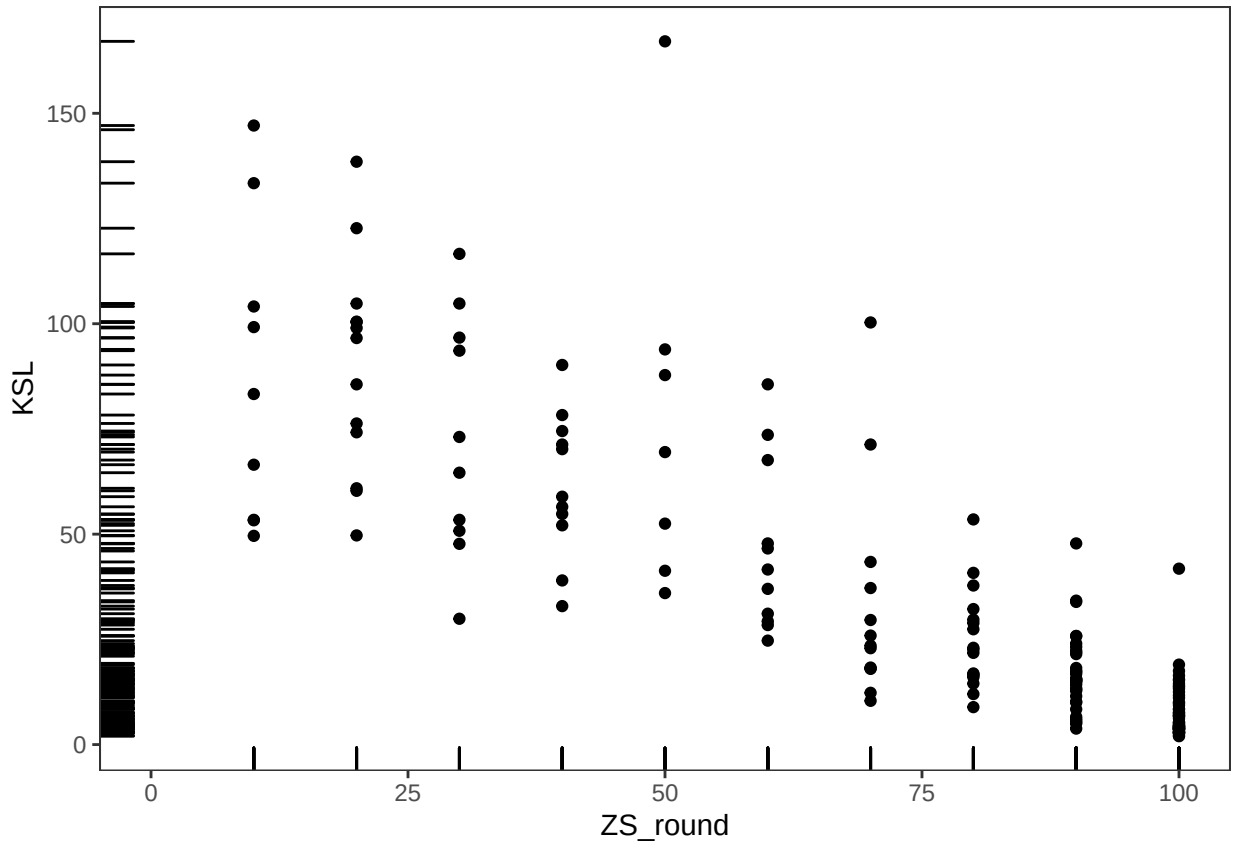
Aufgabe 2: Scatterplots

2a)

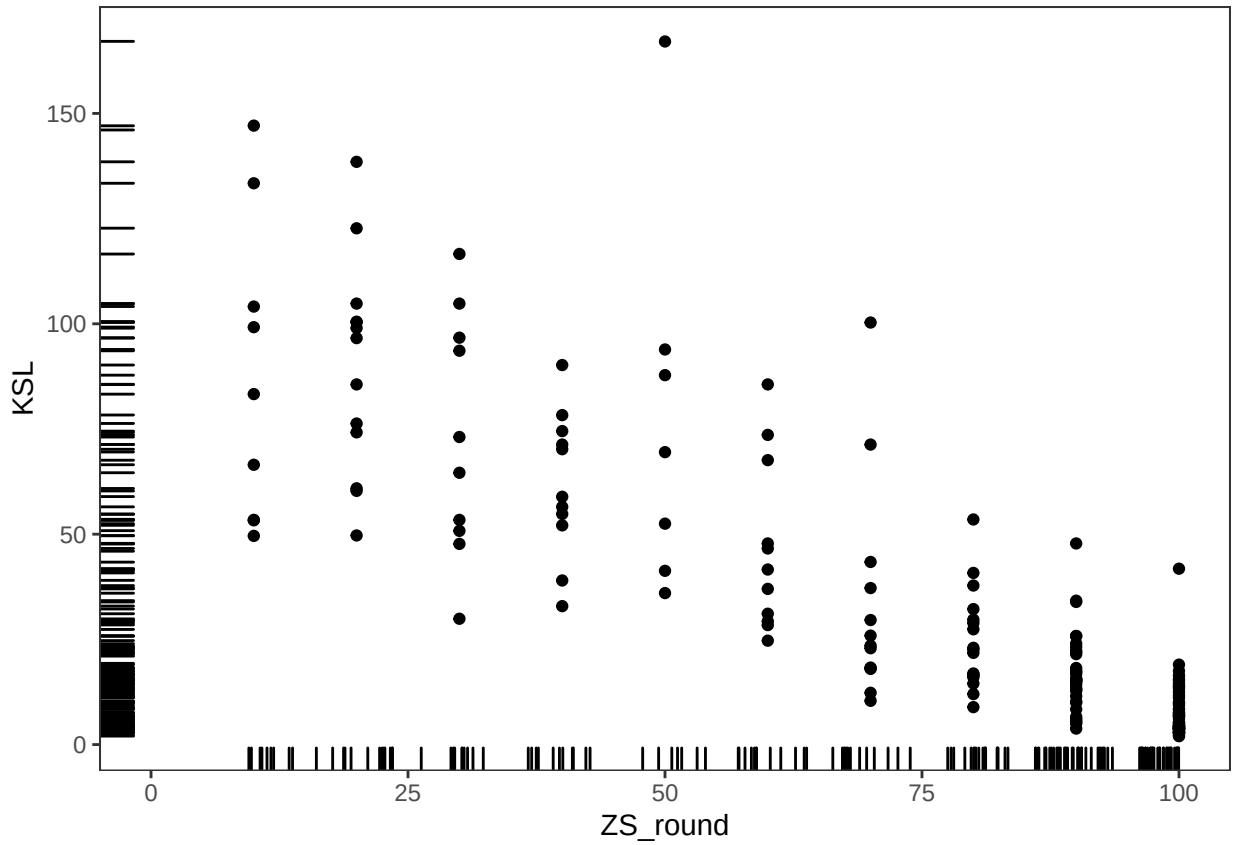
```
# Scatterplot mit empirischen Randverteilungen
g <- ggplot(wdi13, aes(x = ZS, y = KSL))
g2 <- g + geom_point() + xlim(c(0,100))
g2 + geom_rug(sides = "bl")
```




```
# Obige Darstellungsart ist problematisch, falls nur wenige unterschiedliche
# Werte vorkommen
# -> Lösung: Jittering
wdi13$ZS_round <- round(wdi13$ZS, digits = -1)
gr <- ggplot(wdi13, aes(x = ZS_round, y = KSL)) + geom_point() + xlim(c(0,100))
gr + geom_rug(sides = "bl")
```

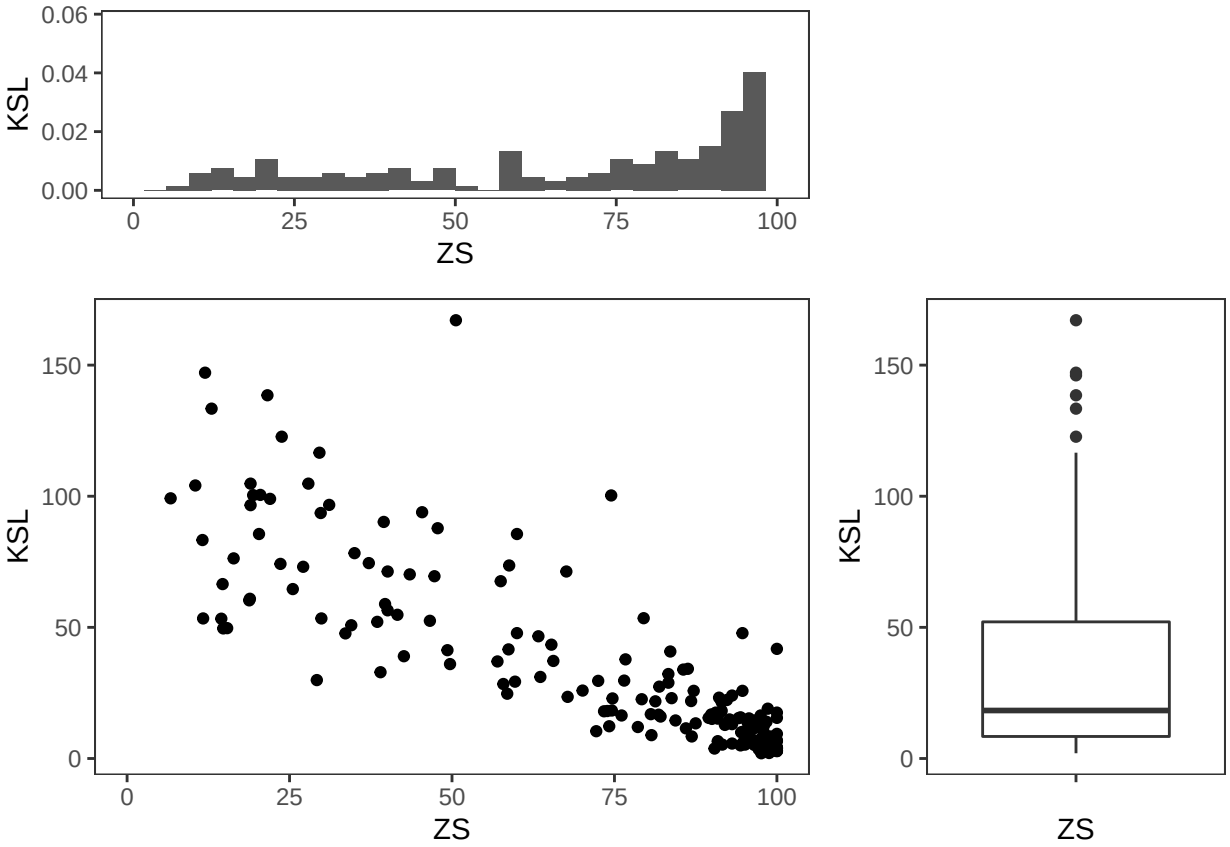



```
gr + geom_rug(sides = "bl", position = "jitter")
```



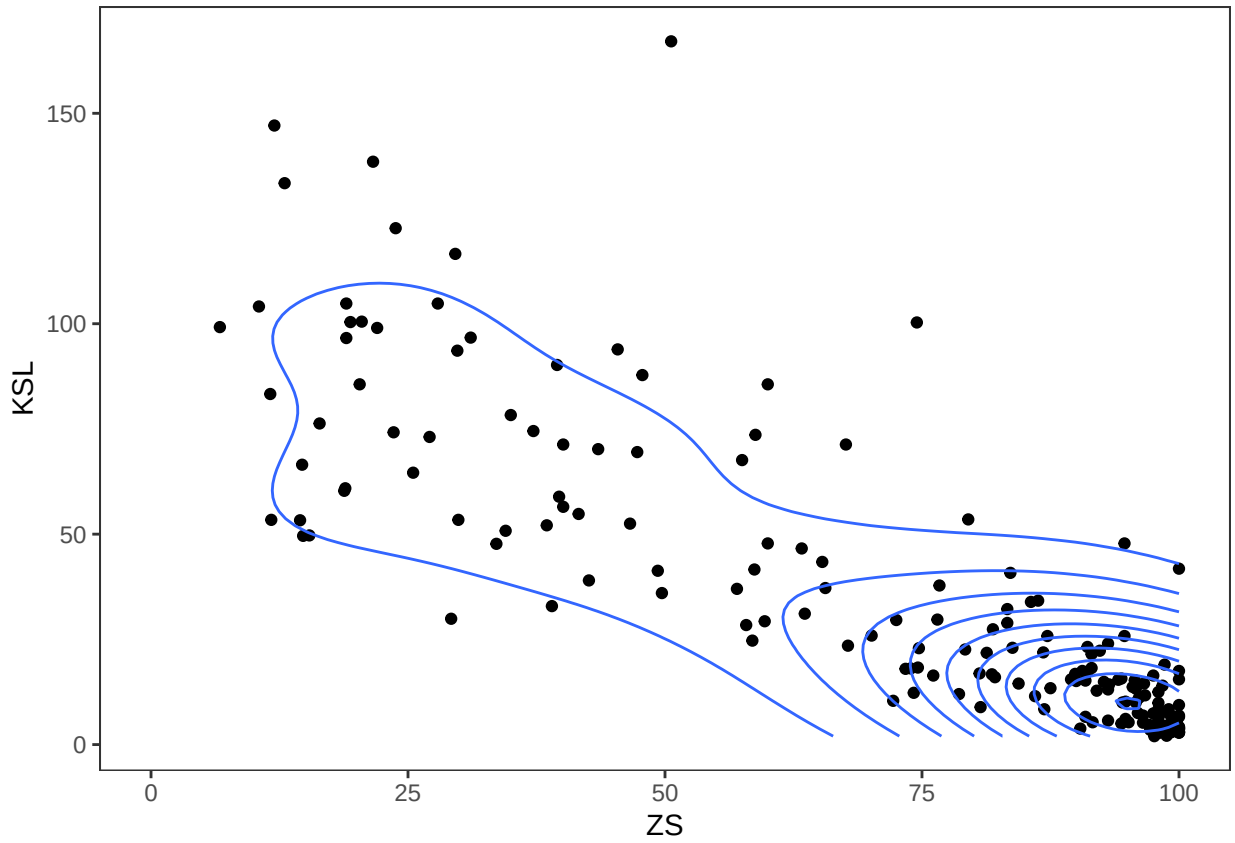

```
# Scatterplot mit empirischen Randverteilungen
library(gridExtra)
grid.arrange(g + geom_histogram(aes(y=..density..)) + xlim(c(0,100)),
             ggplot() + theme_minimal(),
             g2,
             g + geom_boxplot(aes(x="")),
             nrow = 2, ncol = 2, widths = c(2,1), heights = c(1,2))
```

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

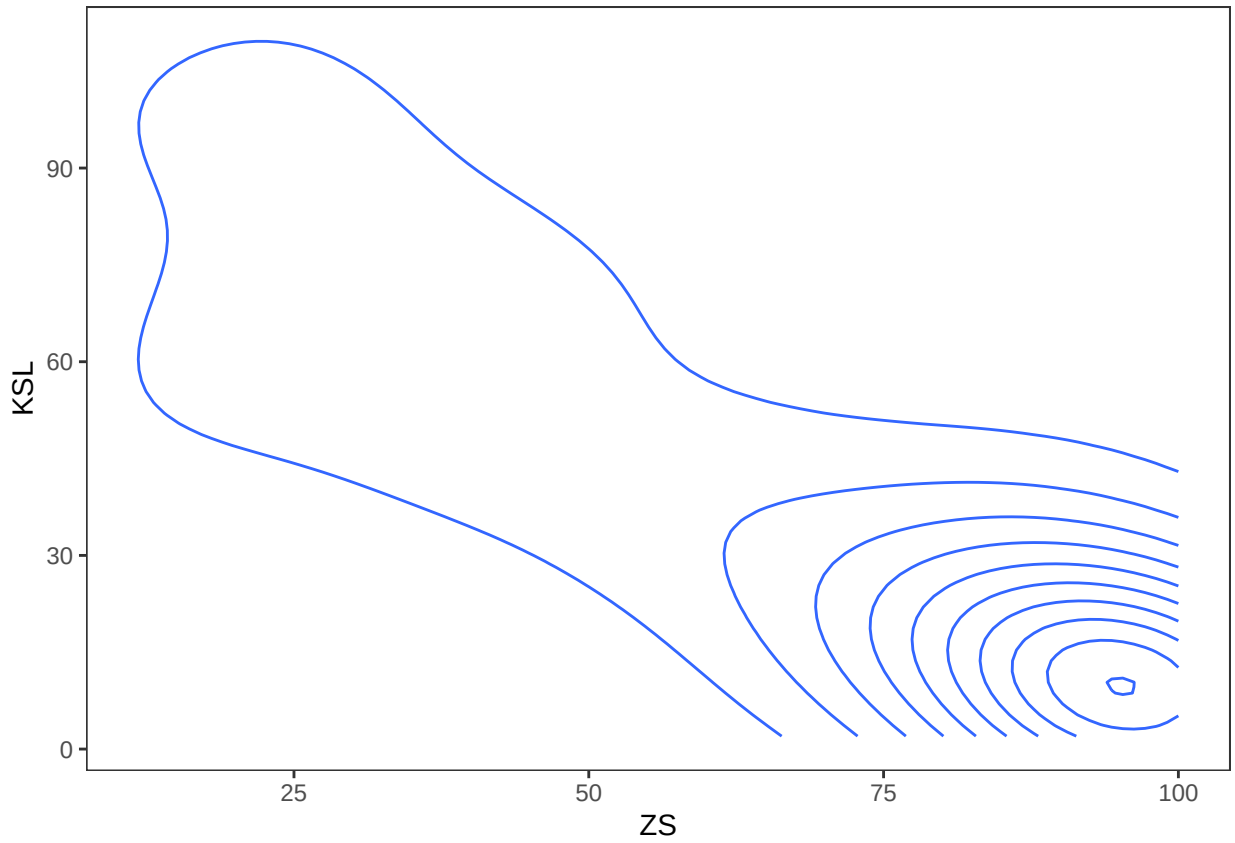


2b)

```
g2 + geom_density2d()
```




```
g + geom_density2d()
```

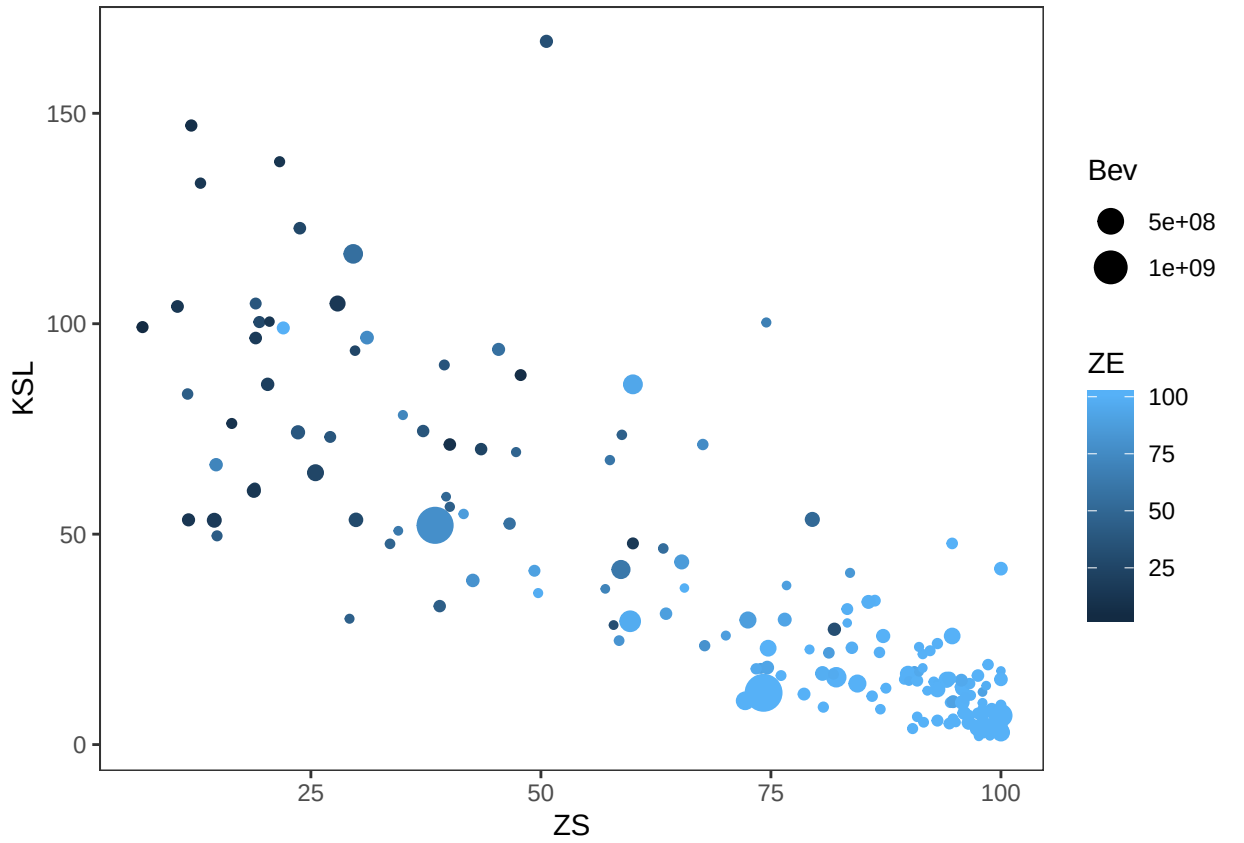


2c)

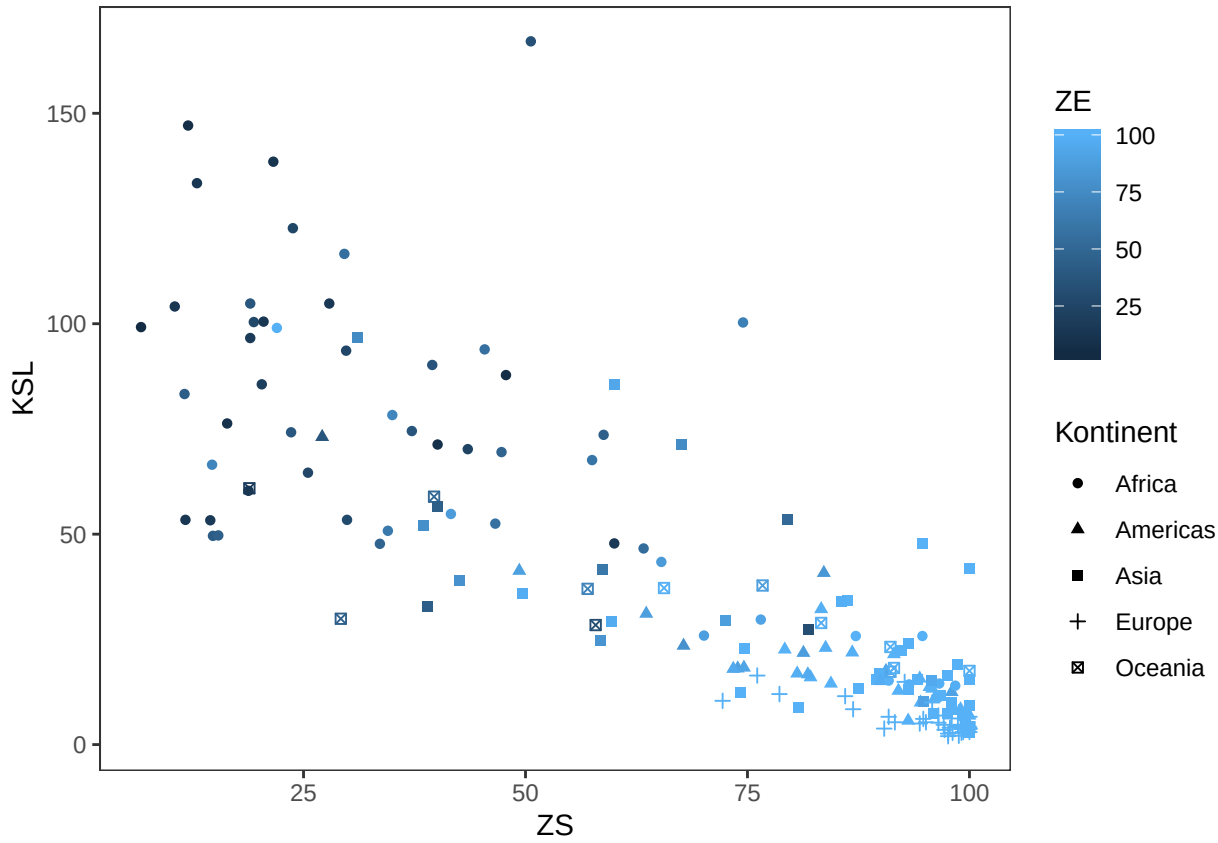
Mögliche Dimensionen zur Darstellung weiterer Variablen:

- Farbe
- Punktgröße
- Punktform

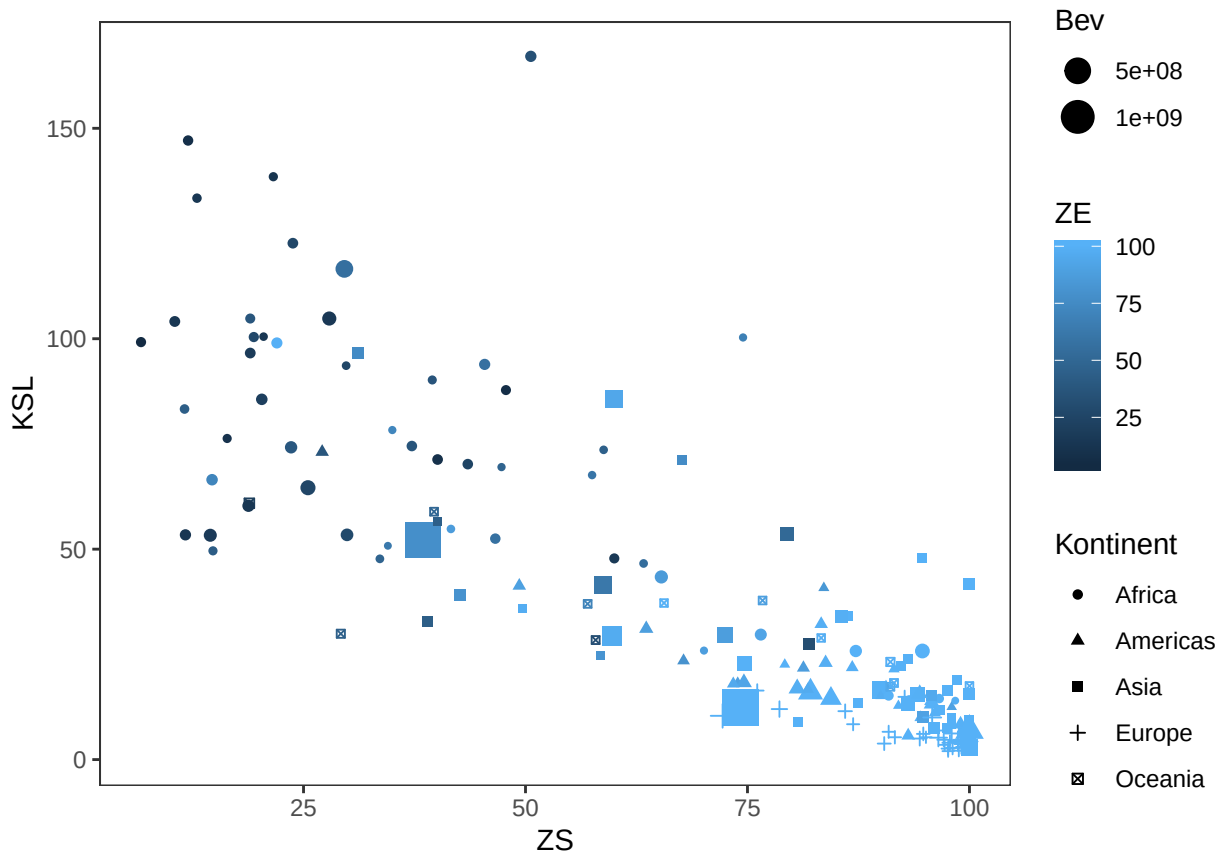
```
# Scatterplot mit weiteren Variablen
g + geom_point(aes(col=ZE, size=Bev))
```




```
# auch möglich: Symbole
g + geom_point(aes(col=ZE, shape=Kontinent))
```




```
# Aber nicht übertreiben: Alles zusammen wäre hier etwas unübersichtlich!
g + geom_point(aes(col=ZE, size=Bev, shape=Kontinent))
```



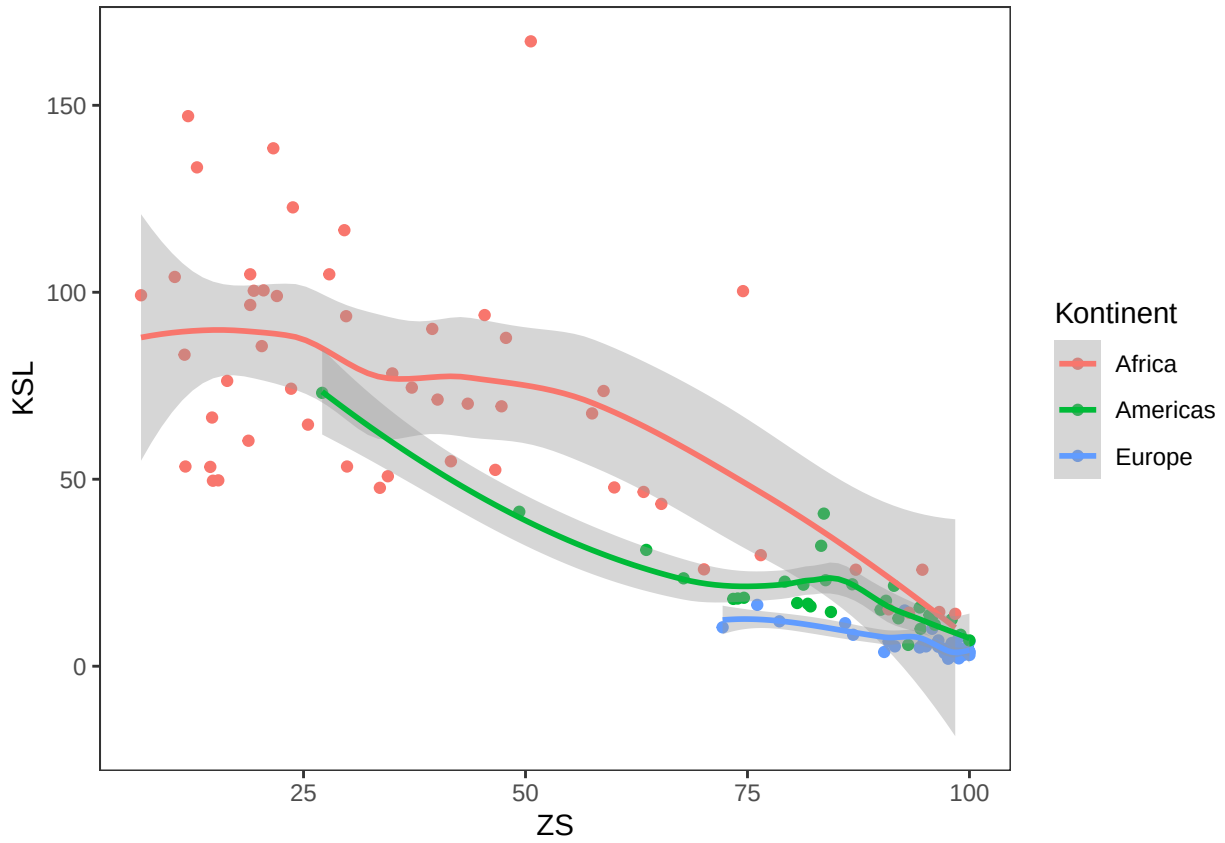
Kurzer Exkurs: Vereinzelt kann es auch hilfreich sein Animationen bzw. GIFs zu erstellen:

- gganimate Paket (siehe auch die Beispiele hier)
- rgl Paket

2d)

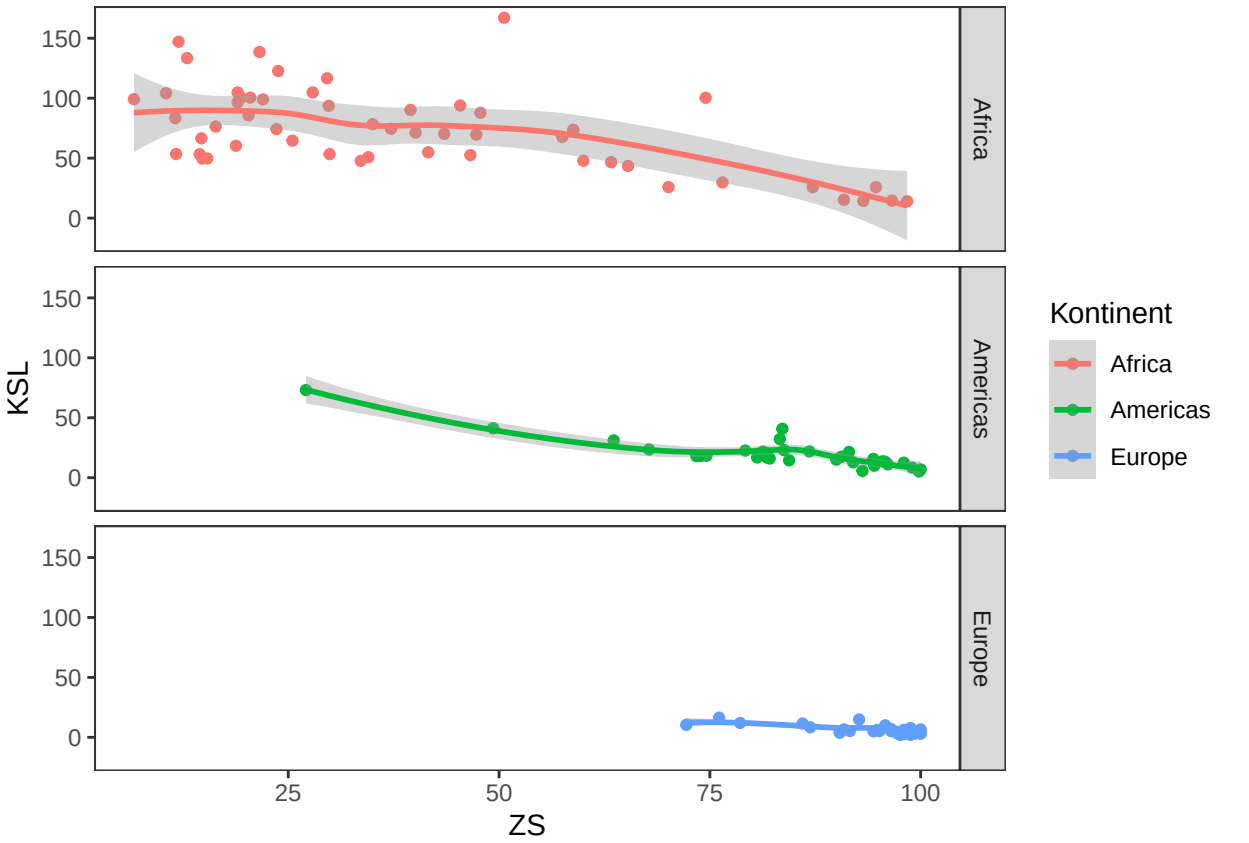
```
dat <- wdi13[wdi13$Kontinent %in% c("Africa","Americas","Europe"),]
gc <- g %+% dat
# Scatter Plot mit LOESS-Glättungskurven unterteilt nach dritter Variable
gc + geom_point(aes(col = Kontinent)) +
  geom_smooth(aes(group = Kontinent, col = Kontinent))
```

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




```
gc + geom_point(aes(col = Kontinent)) +
  geom_smooth(aes(group = Kontinent, col = Kontinent)) +
  facet_grid(Kontinent ~ .)
```

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



2e)

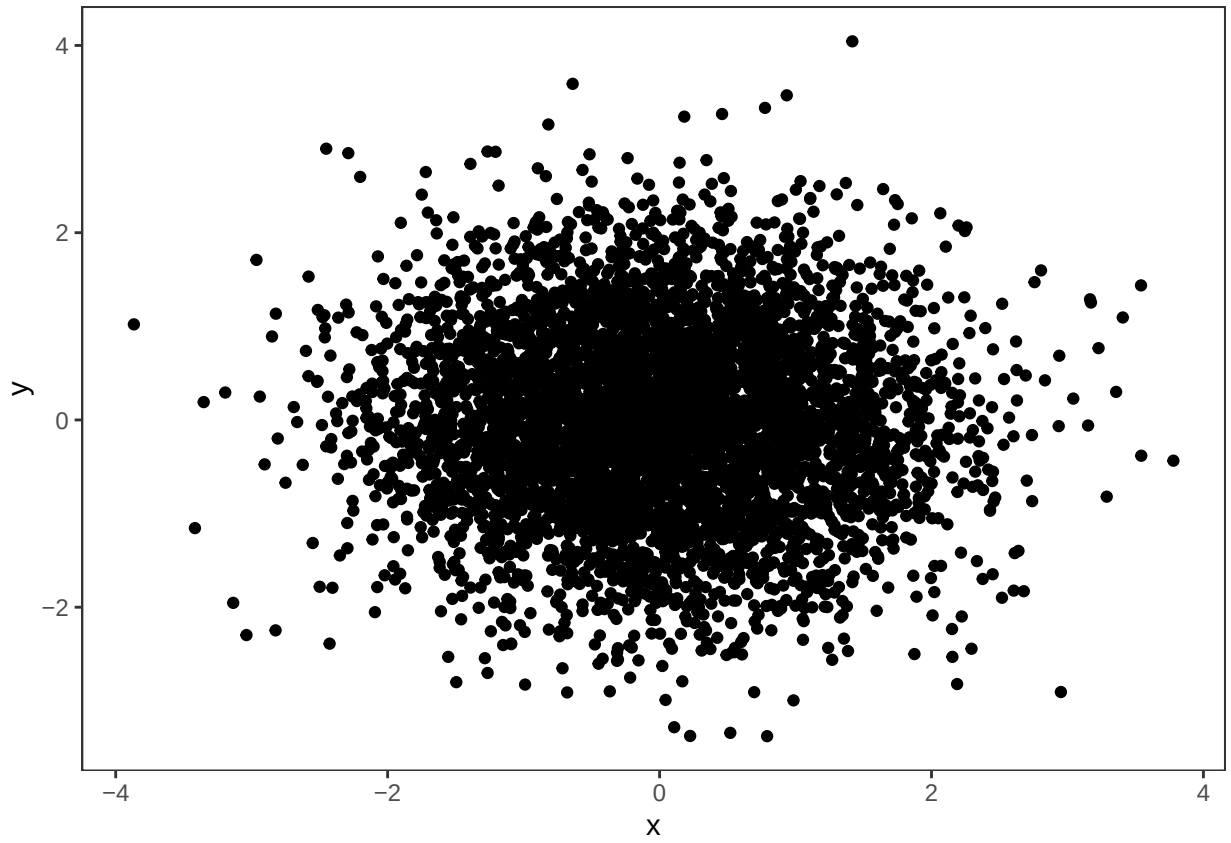
Probleme bei Scatterplots für hochdimensionale Daten:

- Overplotting
- Vektorgraphiken (z.B. pdf) werden sehr groß

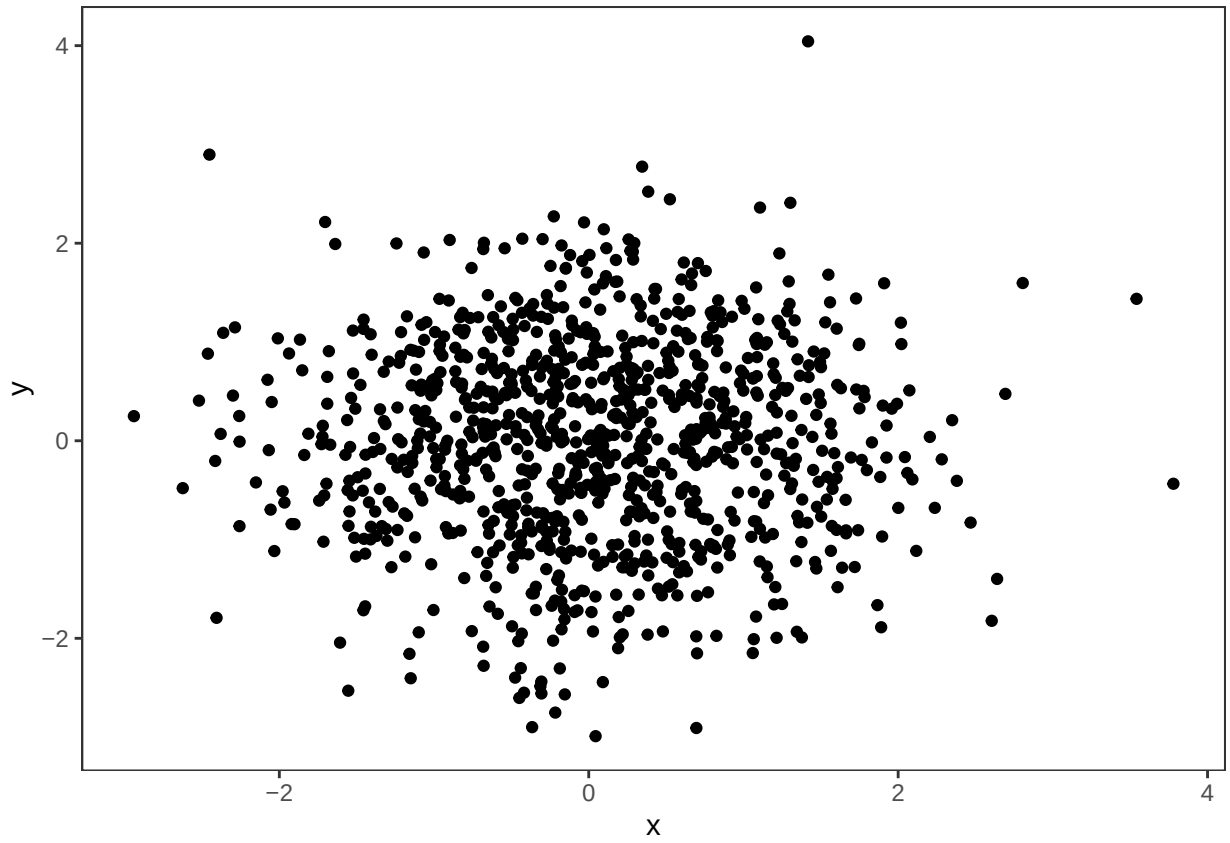
Alternativen:

- Sampling: Darstellung nur von x zufällig gezogenen Beobachtungen aus den Ursprungsdaten
- Transparenz: Punkte mit transparenter Farbe plotten, um Overplotting sichtbar zu machen
- Dichte: Kerndichteschätzung schätzen und plotten
- Binning: Scatterplot in Raster unterteilen, pro Kästchen die Anzahl an Punkten darin darstellen, wie bei Histogramm

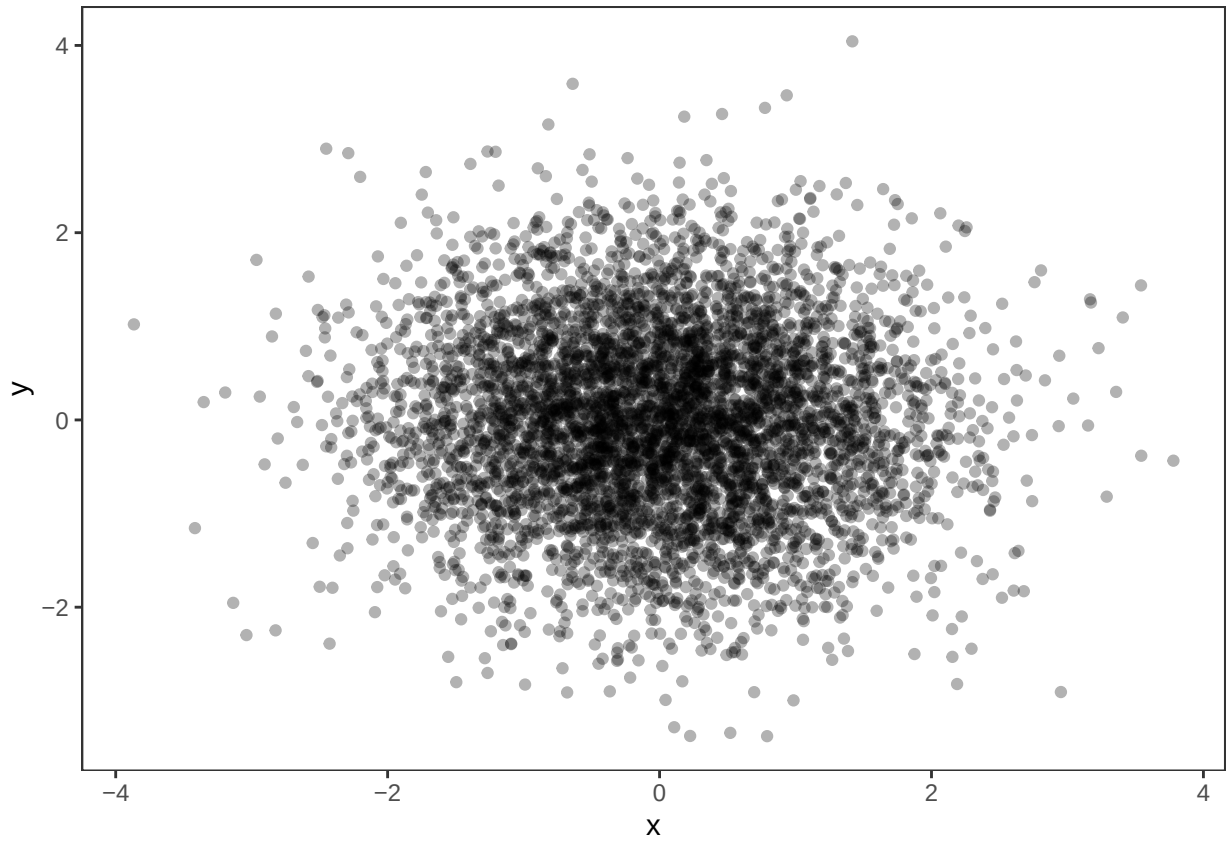
```
dat <- data.frame(x = rnorm(5000),y=rnorm(5000))
g <- ggplot(dat,aes(x=x,y=y))
# Problem: Overplotting
g + geom_point()
```



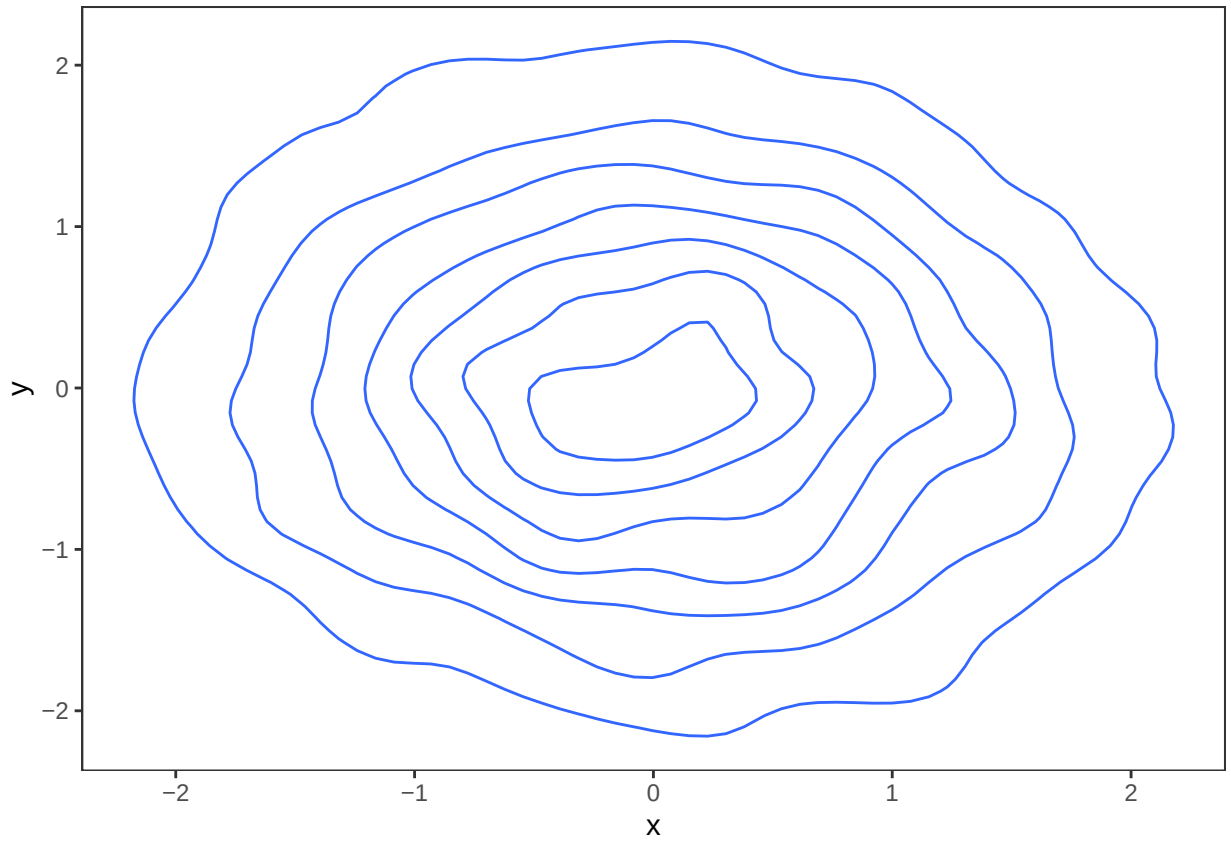

```
# Sampling
dat_kurz <- dat[sample(1:nrow(dat), size = 1000),]
ggplot(dat_kurz,aes(x=x,y=y)) + geom_point()
```



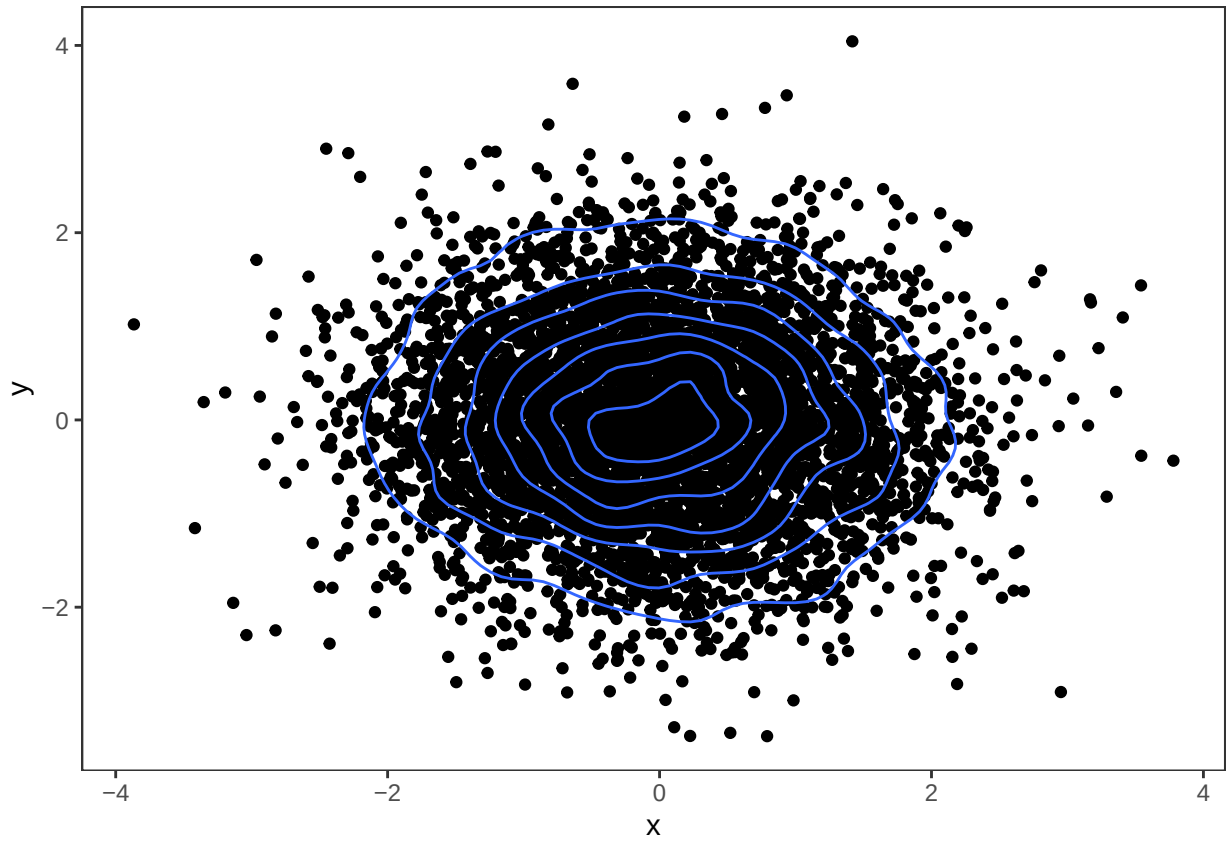

```
# Transparenz
g + geom_point(alpha = 0.3)
```



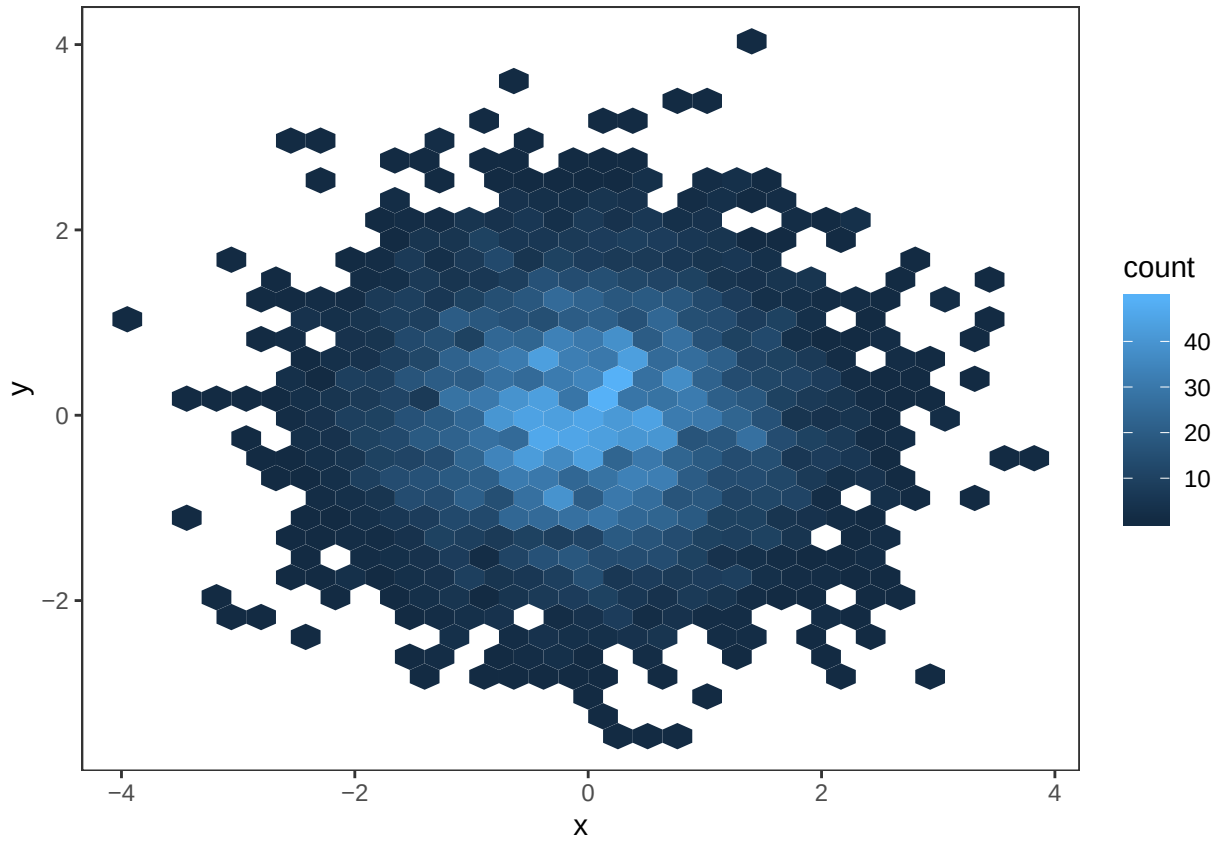

```
# Dichte
g + geom_density2d()
```




```
g + geom_point() + geom_density2d()
```



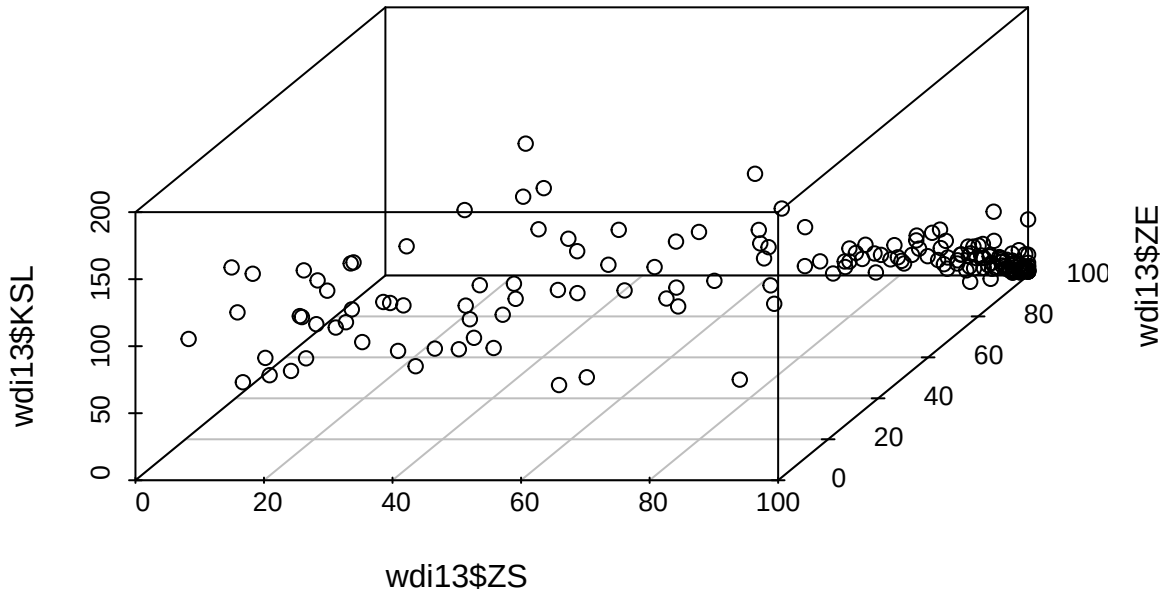

```
# Binning (hexagonal)
library(hexbin)
g + stat_binhex()
```



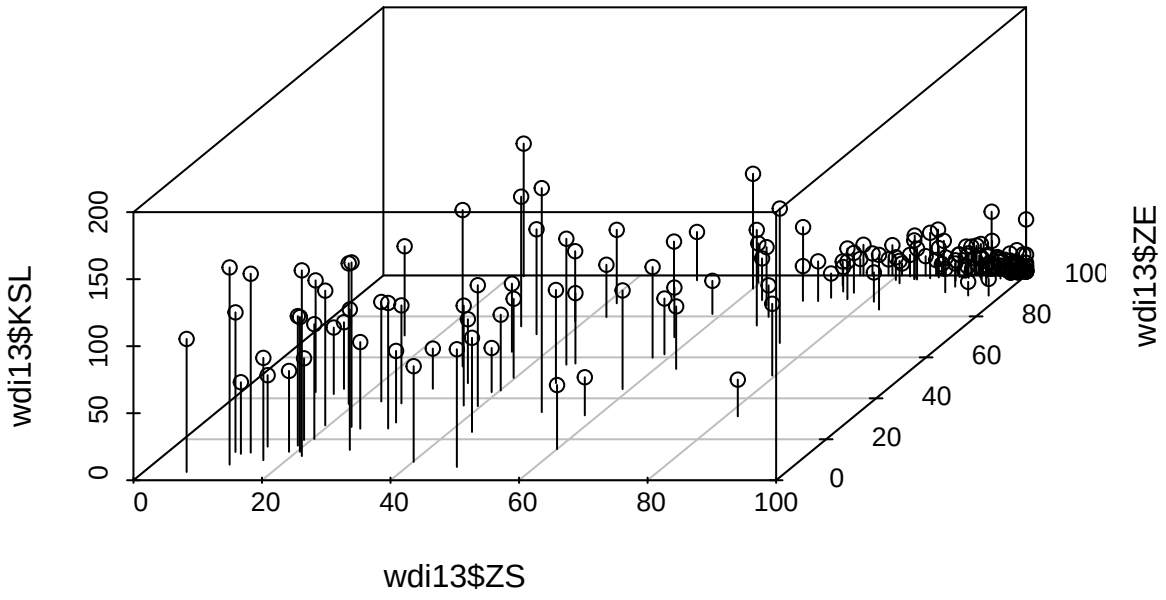
2f)

3D-Scatterplot

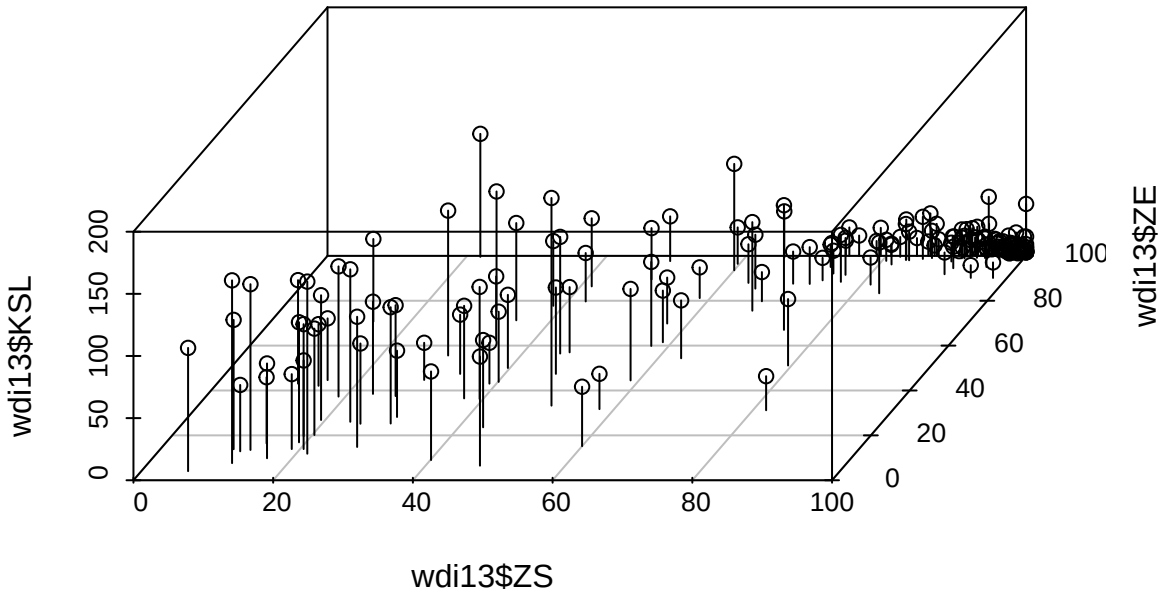
```
library(scatterplot3d)
scatterplot3d(wdi13$ZS, wdi13$ZE, wdi13$KSL, type = "p", angle = 55)
```



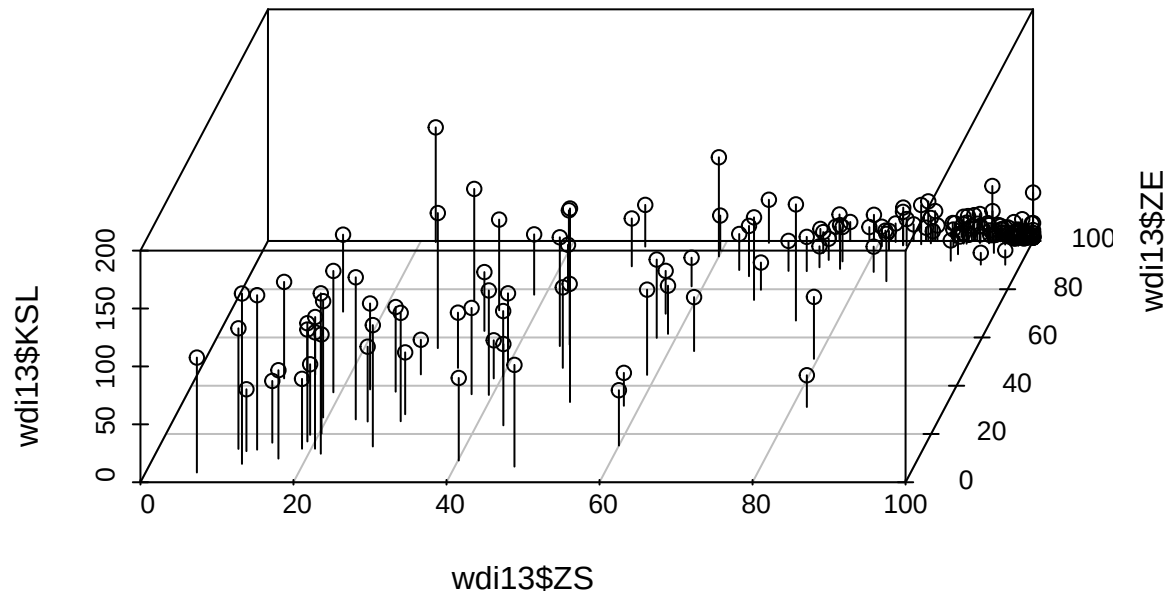

```
scatterplot3d(wdi13$ZS, wdi13$ZE, wdi13$KSL, type = "h", angle = 55)
```




```
scatterplot3d(wdi13$ZS, wdi13$ZE, wdi13$KSL, type = "h", angle = 65)
```




```
scatterplot3d(wdi13$ZS, wdi13$ZE, wdi13$KSL, type = "h", angle = 75)
```



Übersichtlichkeit: Bei so wenigen Punkten noch einigermaßen ok, allgemein jedoch schwierig → besser: 2D-Scatterplot mit Farbdimension oder interaktive Graphiken

Aufgabe 3: Spezielle Graphiken

3a)

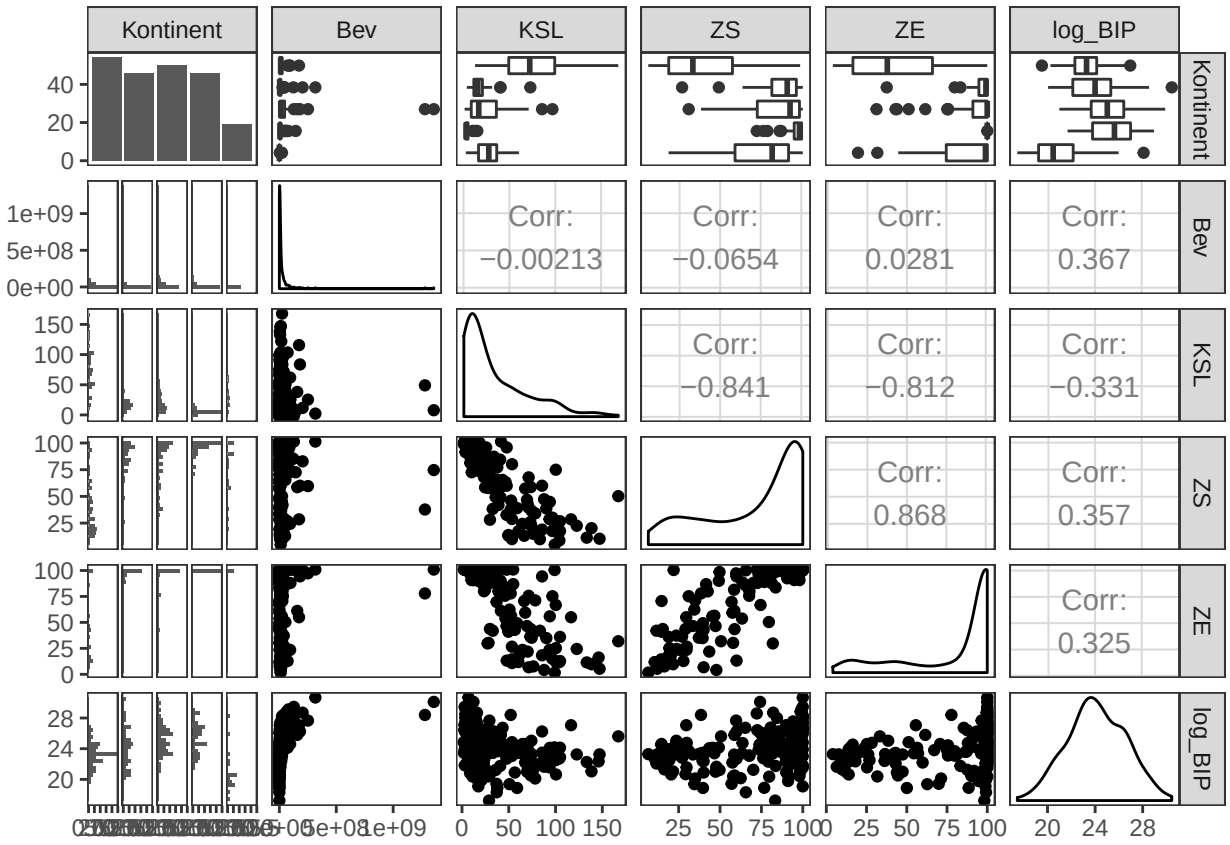
```
# Scatterplot Matrix mit Regressionsgeraden
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
dat <- wdi13[,c("Kontinent","Bev","KSL","ZS","ZE","log_BIP")]
ggpairs(dat)
```

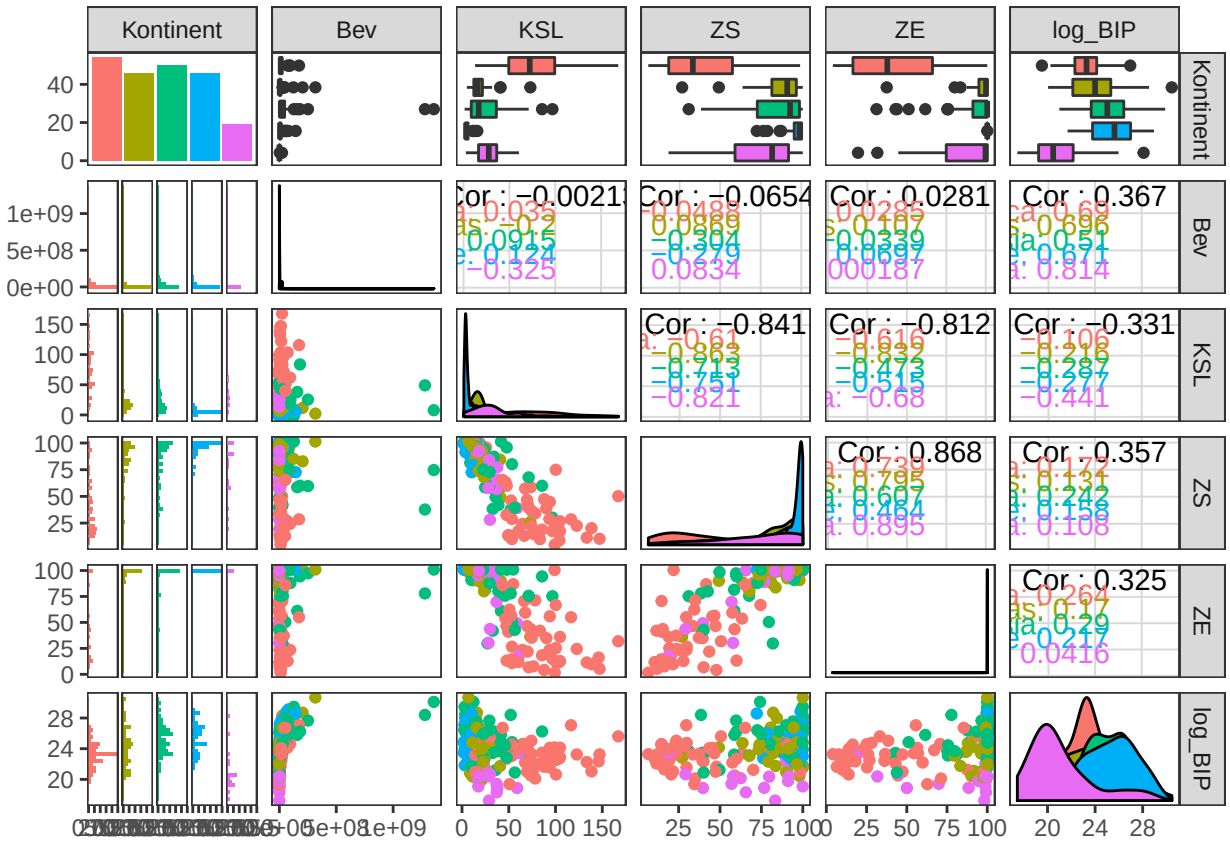
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

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




```
ggpairs(dat, aes(color=Kontinent))
```

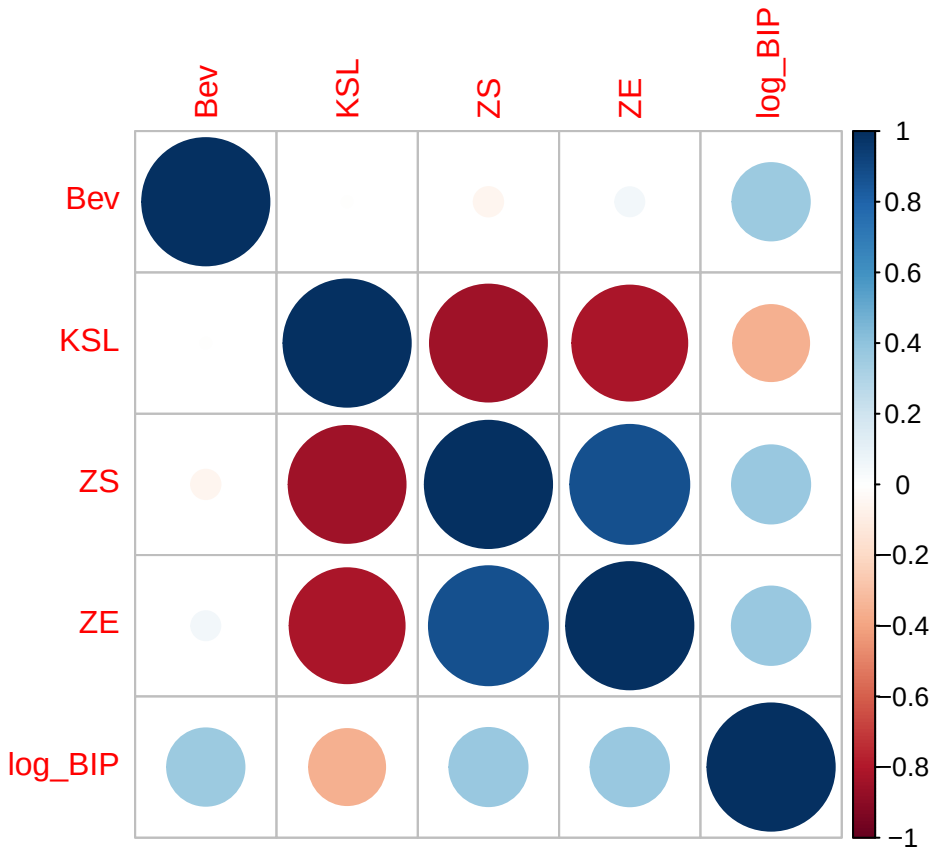
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using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



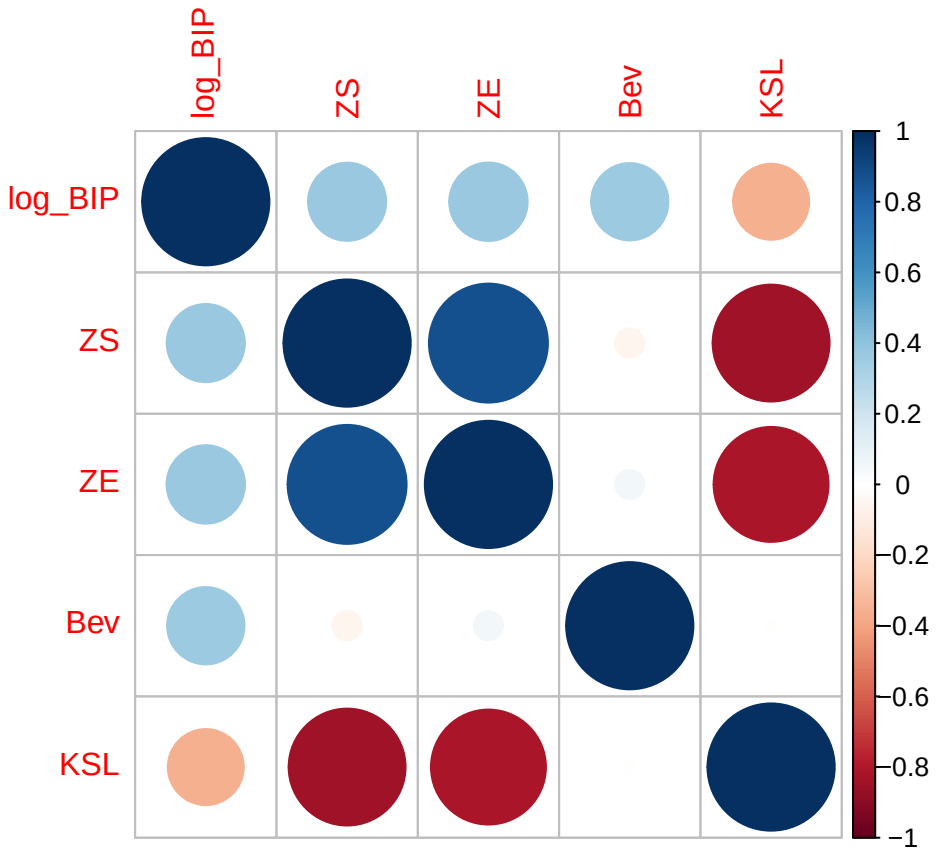

```
# Korrelationsmatrix
library(corrplot)
```

```
## corrplot 0.84 loaded
```

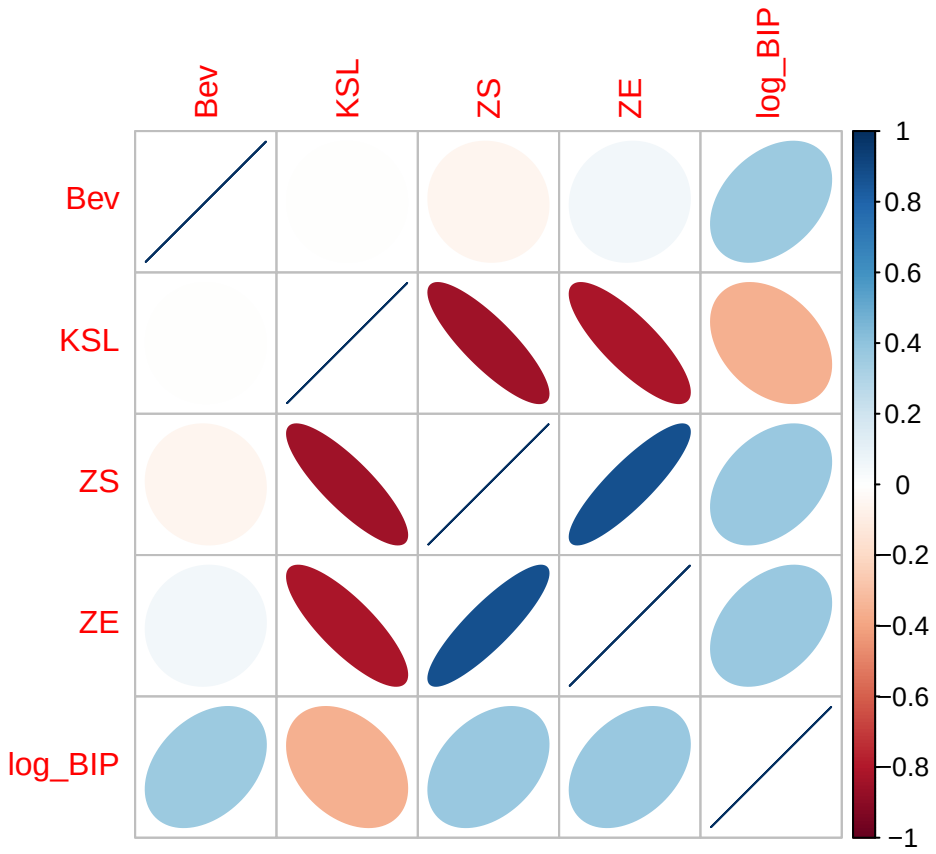
```
dat$Kontinent <- NULL
mat <- cor(dat, use = "complete.obs")
corrplot(mat, method="circle")
```



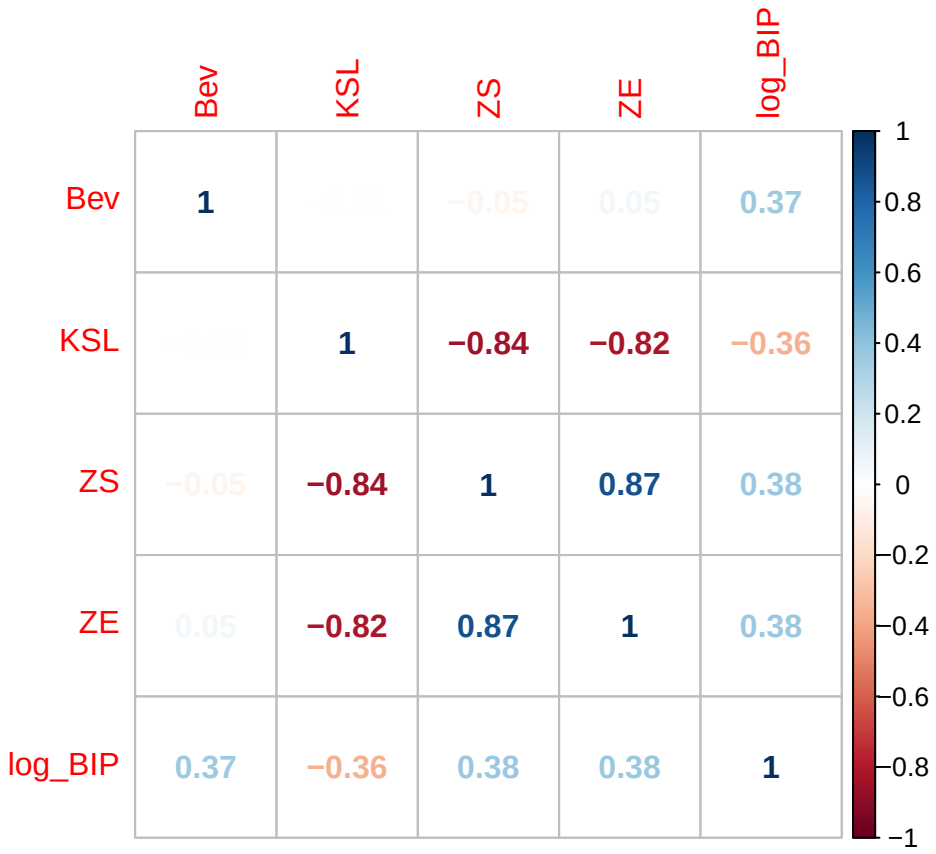

```
corrplot(mat, method="circle", order="hclust")
```




```
corrplot(mat, method="ellipse")
```

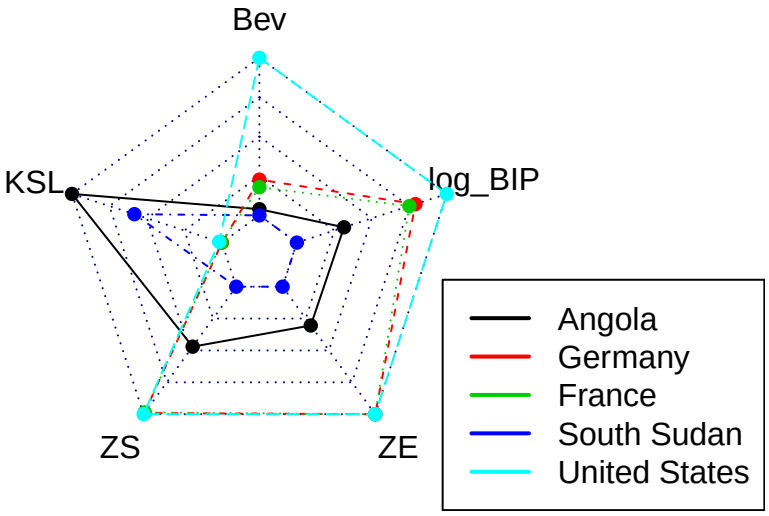



```
corrplot(mat, method="number")
```



3b)

```
library(fmsb)
dat <- wdi13[wdi13$Land %in% c("Germany","France","United States",
                               "Angola","South Sudan"),]
row.names(dat) <- dat$Land
dat <- dat[,c("Bev","KSL","ZS","ZE","log_BIP")]

radarchart(dat, maxmin = FALSE)
legend("bottomright", col = 1:nrow(dat), lwd = 2, legend = row.names(dat))
```



Chernoff-Gesichter

Quelle: https://gramener.com/faces/

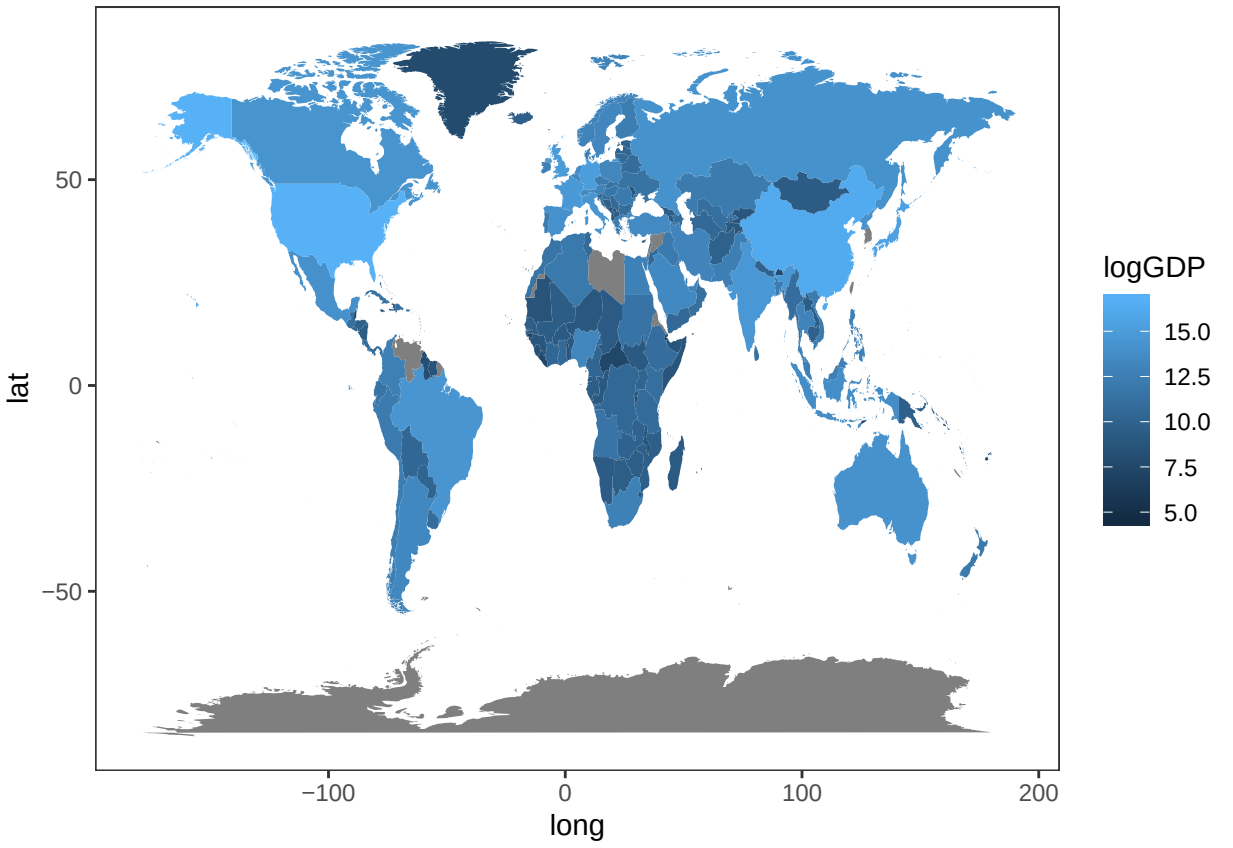
Wann sind Chernhoff-Gesichter prinzipiell sinnvoll?

- Vergleich von einzelnen Individuen / Beobachtungseinheiten
- Vergleich nur anhand weniger Variablen
- Variablen müssen klare Richtung aufweisen, da z.B. die Darstellung einer Variable auf der Skalar 'lächelnder Mund' ↔ 'trauriger Mund' eine implizite Wertung enthält

3c)

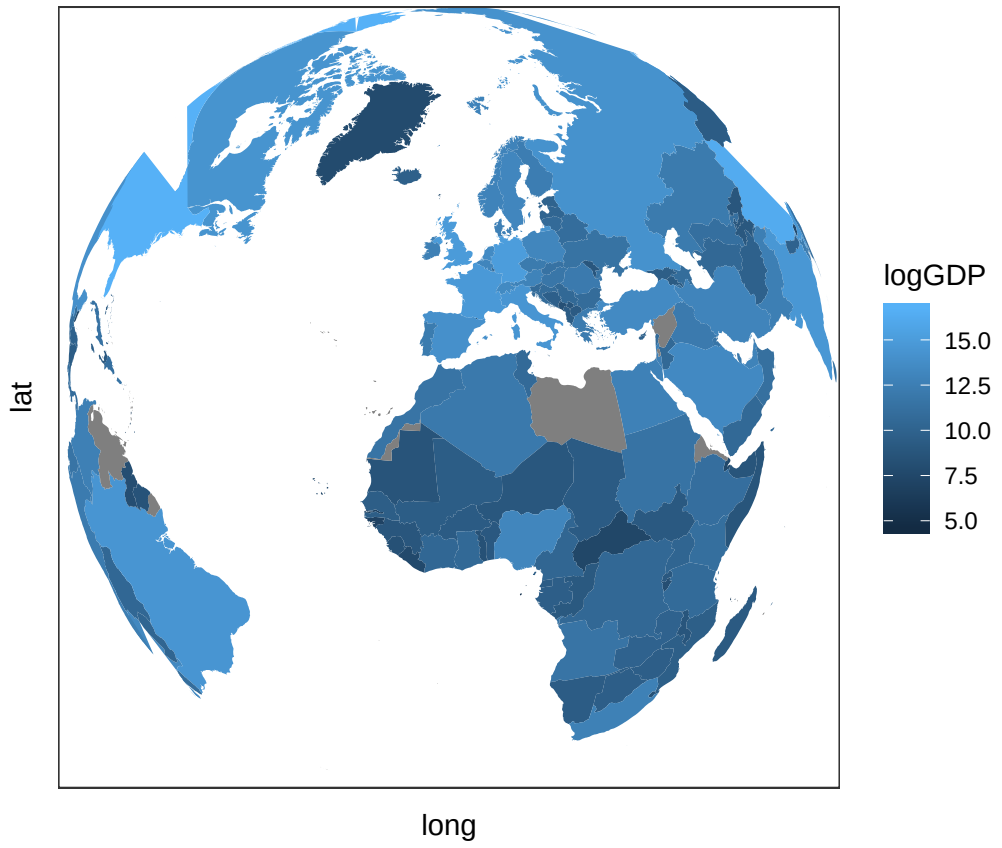
Weltkarte

```
world <- map_data("world")
gdp <- read.csv2("R/GDP.csv")
world$logGDP <- log(gdp$GDP[match(world$region, gdp$Country)])
g <- ggplot(world, aes(x = long, y = lat, group = group, fill = logGDP)) +
  geom_polygon()
g
```



Ändern des Koordinatensystems

```
g + coord_map("ortho", orientation=c(31, -4, 0))
```



Aufgabe 4: Interaktiv

Anmerkung: Mit manipulate und shiny behandeln wir hier zwei sehr flexible, aber bei Weitem nicht alle R-basierten Möglichkeiten für interaktive Grafiken. Weitere nützliche Pakete sind z.B. das Paket iplots oder das Paket plotly. Letzteres bietet insbesondere die Möglichkeit, ggplot2-Grafiken auf einfache Weise um Interaktivität zu erweitern und bietet auch eine einfache Integrationsmöglichkeit in shiny-Dashboards.

Darüber hinaus gibt es auch eine Reihe kommerzieller Produkte wie etwa Tableau oder QlikView, welche wie shiny die Erstellung interaktiver Dashboards ermöglichen.

4a)

```
library(manipulate)

# Einfacher Scatterplot
manipulate(
  ggplot(wdi[wdi$Jahr == jahr,], aes(x = ZS, y = KSL)) + geom_point() +
    xlim(c(0,100)),
  jahr = picker(2010,2011,2012,2013))

# Scatterplot mit KDE
plot_gg <- function(jahr, bw.x, bw.y) {
  ggplot(wdi[wdi$Jahr == jahr,], aes(x = ZS, y = KSL)) +
    geom_point() + xlim(c(0,100)) +
    geom_density2d(h = c(bw.x, bw.y))
}
```

```
manipulate(
  plot_gg(jahr, bw.x, bw.y),
  jahr = picker(2010,2011,2012,2013),
  bw.x = slider(10, 100, step = 10),
  bw.y = slider(10, 100, step = 10))
```

4b)

Grundlagen zu shiny

- shiny als Paket erzeugt HTML-Oberfläche, welche in jedem Browser betrachte werden kann
- Jedes shiny-Programm besteht aus einem UI- und einem Server-Codeteil.
- ui: Definition der UI-Elemente (Aufbau der Seite, wo werden Plots angezeigt (ohne Plots zu definieren!), wo werden Schieberegler etc. angezeigt)
- server: Enthält wirklichen R-Code. Plot-Funktionen werden UI-Elementen zugeordnet, in welchen sie angezeigt werden
- Wodurch wird shiny interaktiv? → Prinzip der Reaktivität:
- Jeder Auswahlbutton, Schieberregler etc. ordnet den aktuell ausgewählten Wert einer *reaktiven* Variable zu.
- Hängt eine R-Funktion (z.B. mit einem Plot als output) im server-Code von einer reaktiven Variable als Argument ab, so wird die Funktion von shiny automatisch jedes Mal neu ausgeführt, wenn sich die reaktive Variable ändert (d.h. wenn der Benutzer einen Eingabewert verändert).

Weiterführendes:

- Beispiele für shiny-Apps sowie gute einführende Tutorials finden sich auf shiny.rstudio.com.
- Interaktive shiny-Elemente lassen sich auch innerhalb von html-Markdown-Dokumenten anzeigen: Quelle (siehe Punkt 7)
- Eine vom StaBLab programmierte shiny-App zu Wahlanalysen findet sich unter koala.stat.uni-muenchen.de.

Aufruf einer shiny-App

```
library(shiny)
runApp("R/4b_shiny.R")
```

Inhalt der 4b_shiny.R-Datei:

```
shinyApp(
  ui = shinyUI(fluidPage(

    titlePanel("Shiny - Beispiel"),

    sidebarLayout(
      sidebarPanel(
        selectInput("jahr", label = "Wählen Sie das Jahr aus:",
                    choices = 2010:2013, selected = 2013),
        sliderInput("bw.x", label = "Bandweite in x-Richtung:",
                    min = 10, max = 100, step = 10, value = 50),
        sliderInput("bw.y", label = "Bandweite in y-Richtung:",
                    min = 10, max = 100, step = 10, value = 50)
      ),

      mainPanel(
        plotOutput("scatterplot")
      )
```

```
    )
  )),
  server = shinyServer(function(input, output) {

    output$scatterplot <- renderPlot({
      ggplot(wdi[wdi$Jahr == input$jahr,], aes(x = ZS, y = KSL)) +
        geom_point() + xlim(c(0,100)) +
        geom_density2d(h = c(input$bw.x, input$bw.y))
    })
  })
)
```

Appendix

Weitere Beispiele dynamischer Graphiken mit Zeitkomponente:

- Hans Rosling's 200 Countries, 200 Years, 4 Minutes
- Gapminder.org