5. Fortgeschrittene Plots

Gruppierte Balkendiagramme, Dumbbell Plots, Datenlabels und schrittweiser Aufbau eines Custom Themes

Dr. Paul Schmidt

```
for (pkg in c("tidyverse", "gapminder", "showtext", "ggtext")) {
  if (!require(pkg, character.only = TRUE)) install.packages(pkg)
}

showtext::showtext_opts(dpi = 300)
```

In den vorherigen Kapiteln haben wir die einzelnen Bausteine von ggplot2 kennengelernt: Achsen, Farben, Themes und Export. Jetzt ist es an der Zeit, sie zu vollstaendigen, publikationsreifen Grafiken zu kombinieren. Dafuer wechseln wir vom kleinen `PlantGrowth`-Datensatz zum {gapminder}-Datensatz - einem deutlich reichhaltigeren Datensatz, der Lebenserwartung, Bevoelkerung und BIP pro Kopf fuer 142 Laender von 1952 bis 2007 verfolgt.

Dabei werden uns mehrere praktische Herausforderungen begegnen: Wie man die Reihenfolge von Kategorien steuert, wie man Datenlabels korrekt an gruppierten Balken ausrichtet, und wie man Schritt fuer Schritt eine eigene Theme-Funktion aufbaut - inspiriert vom sauberen Design der Nature-Grafiken.

Datenvorbereitung

Wir filtern die gapminder-Daten, um 1952 und 2007 fuer sieben ausgewaehlte Laender zu vergleichen. Die `year`-Spalte muss in einen Faktor umgewandelt werden, damit die Gruppierung in ggplot2 korrekt funktioniert.

```
dat <- gapminder::gapminder %>%
  filter(year == 1952 | year == 2007) %>%
  filter(country %in% c(
    "Canada", "Germany", "Japan",
    "Netherlands", "Nigeria", "Vietnam", "Zimbabwe"
  )) %>%
  mutate(year = as.factor(year)) %>%
  droplevels()

dat
```

```
# A tibble: 14 × 6
   country     continent year  lifeExp      pop gdpPercap
   <fct>       <fct>     <fct>   <dbl>    <int>     <dbl>
 1 Canada      Americas  1952     68.8  14785584    11367.
 2 Canada      Americas  2007     80.7  33390141    36319.
 3 Germany     Europe    1952     67.5  69145952     7144.
 4 Germany     Europe    2007     79.4  82400996    32170.
 5 Japan       Asia      1952     63.0  86459025     3217.
 6 Japan       Asia      2007     82.6 127467972    31656.
 7 Netherlands Europe    1952     72.1  10381988     8942.
 8 Netherlands Europe    2007     79.8  16570613    36798.
 9 Nigeria     Africa    1952     36.3  33119096     1077.
10 Nigeria     Africa    2007     46.9 135031164     2014.
11 Vietnam     Asia      1952     40.4  26246839      605.
```

```
12 Vietnam     Asia      2007     74.2  85262356     2442.
13 Zimbabwe    Africa    1952     48.5   3080907      407.
14 Zimbabwe    Africa    2007     43.5  12311143      470.
```

Faktor-Level-Reihenfolge

Standardmaessig sortiert ggplot2 Faktor-Levels alphabetisch und wendet diese Reihenfolge auf der y-Achse von unten nach oben an (und auf der x-Achse von links nach rechts). In unserem Plot wuerde das bedeuten: Kanada unten und Simbabwe oben - eine rein willkuerliche Reihenfolge, die es dem Leser erschwert, Muster zu erkennen. Es ist deutlich sinnvoller, Laender nach einer aussagekraeftigen Variable zu sortieren, etwa ihrer Lebenserwartung im Jahr 2007.

Das {forcats}-Paket (wird mit tidyverse geladen) bietet praktische Funktionen fuer die Neuordnung von Faktor-Levels. Hier verwenden wir `fct_relevel()` mit einem vorsortierten Vektor von Laendernamen:

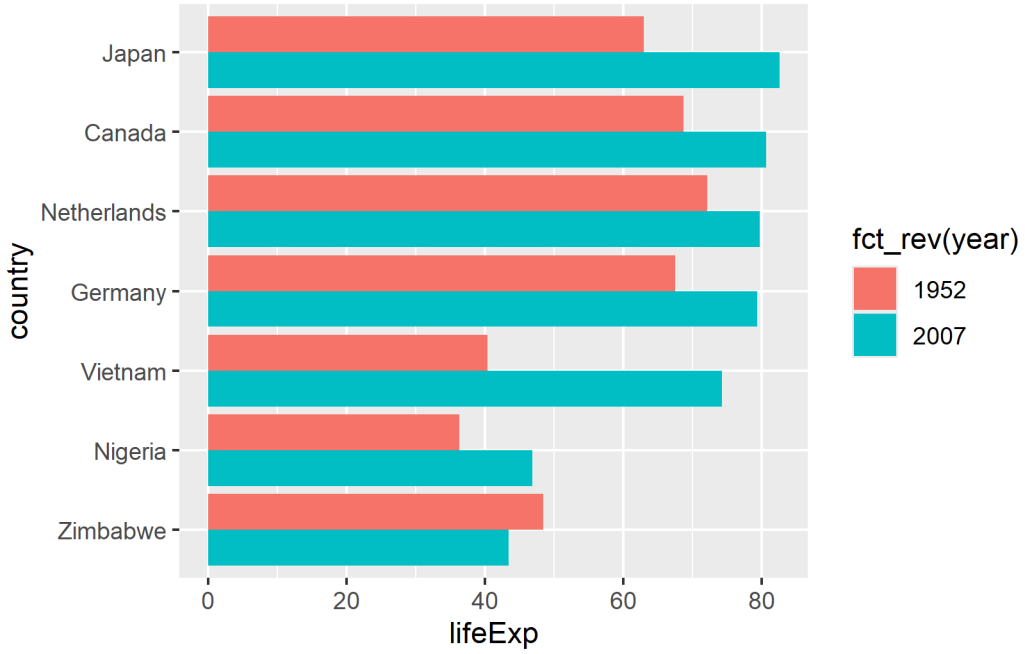
```
sorted_countries <- dat %>%
  filter(year == "2007") %>%
  arrange(lifeExp) %>%
  pull(country) %>%
  as.character()

dat <- dat %>%
  mutate(country = fct_relevel(country, sorted_countries))
```

> Alternative Ansaetze zur Neuordnung
>
> Anstatt einen sortierten Vektor zu erstellen und `fct_relevel()` zu verwenden, gibt es weitere Optionen:
>
> - **`fct_rev()` direkt in `aes()`**: `aes(y = fct_rev(country))` kehrt die aktuelle Reihenfolge um, ohne die Daten zu veraendern. Schnell, aber nur Umkehrung - keine Sortierung nach einer Variable.
> - **`limits` in der Scale**: `scale_y_discrete(limits = c("Zimbabwe", "Nigeria", ...))` legt die Reihenfolge manuell fest. Funktioniert, ist aber fragil - wenn ein Laendername sich aendert oder ein Level fehlt, wird es stillschweigend weggelassen.
> - **`fct_reorder()`**: `mutate(country = fct_reorder(country, lifeExp))` sortiert nach einer anderen Variable. Der kompakteste Ansatz fuer einfache Faelle, erfordert aber Sorgfalt, wenn die Sortiervariable fuer mehrere Jahre existiert.

Gruppiertes Balkendiagramm

Ein klassischer Weg, paarweise Vergleiche zu visualisieren, ist ein gruppiertes Balkendiagramm. `geom_col()` erstellt Balken aus den Daten, und `position_dodge()` platziert Balken verschiedener Gruppen nebeneinander statt uebereinander:

```
ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_discrete(limits = c("1952", "2007"))
```



Hier fallen zwei Details auf. Erstens: `fct_rev(year)` kehrt den Year-Faktor um, sodass die 2007-Balken ueber den 1952-Balken innerhalb jeder Laendergruppe erscheinen - passend zur intuitiven Erwartung „Neuer oben". Zweitens: `limits = c("1952", "2007")` in der Scale stellt sicher, dass die Legende 1952 vor 2007 zeigt (chronologische Reihenfolge), unabhaengig von der internen Faktor-Level-Ordnung.

> Alternative fuer Legendenreihenfolge: guides()
>
> Anstatt `limits` in der Scale zu verwenden, kann man die Legendenreihenfolge auch mit `guides(fill = guide_legend(reverse = TRUE))` umkehren. Das ist manchmal praktischer, wenn die Scale bereits andere Einstellungen hat.

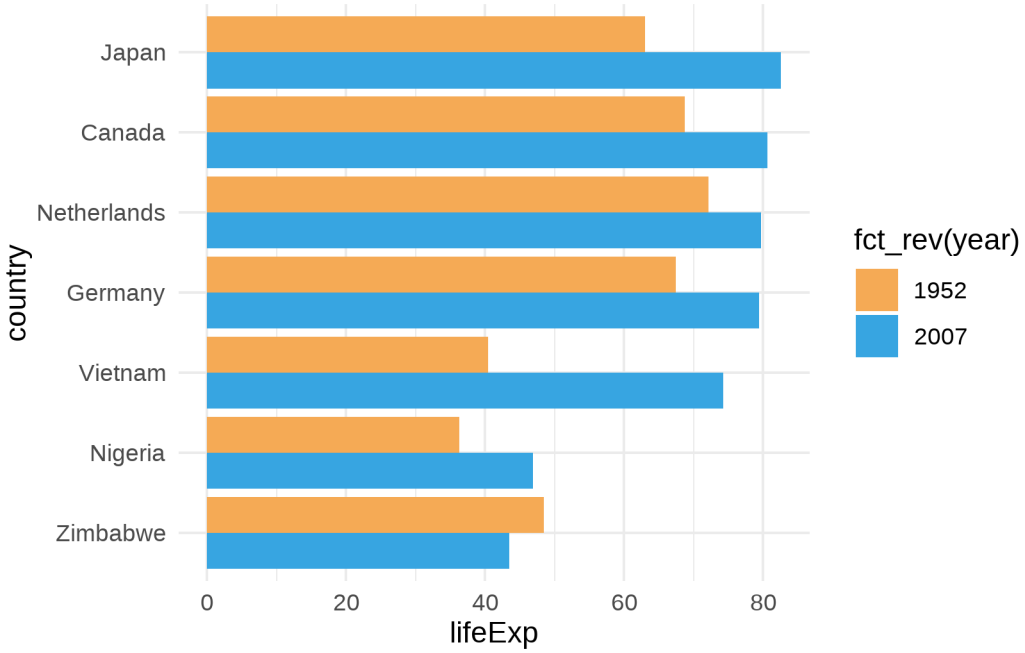
Schrittweiser Aufbau eines Custom Themes

Anstatt ein fertiges Theme zu verwenden, bauen wir eines von Grund auf auf - inspiriert vom sauberen, minimalistischen Stil der Nature-Grafiken. Dieser schrittweise Ansatz macht es einfach zu verstehen, was jedes Theme-Element bewirkt, und das Ergebnis ist eine wiederverwendbare `theme_nature()`-Funktion.

Ausgangspunkt

Wir beginnen mit `theme_minimal()` und einer eigenen Farbpalette fuer die beiden Jahre:

```
sysfonts::font_add_google("Kanit", "kanit")
showtext::showtext_auto()

year_colors <- c("1952" = "#F7AA59", "2007" = "#37A9E1")

ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_manual(limits = names(year_colors), values = year_colors) +
  theme_minimal()
```

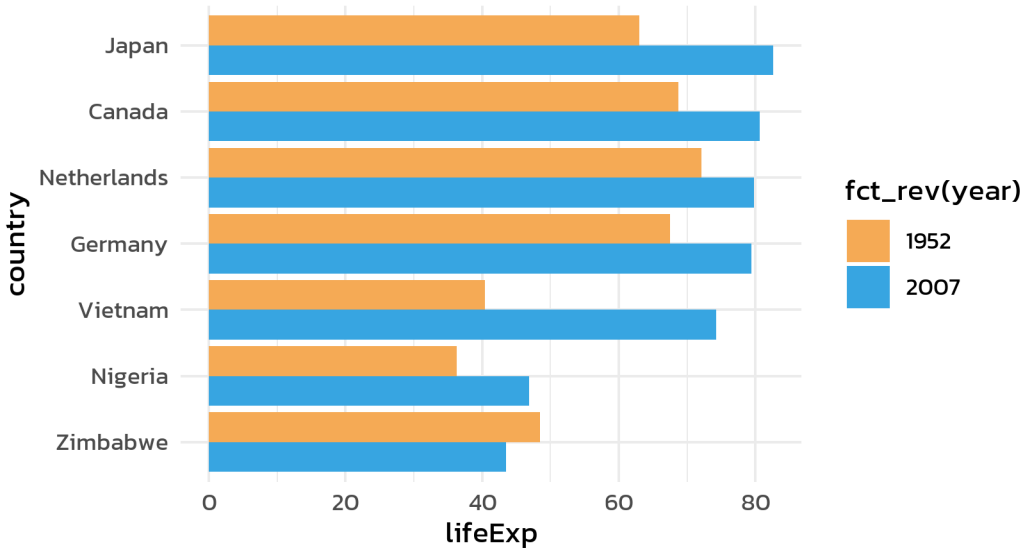


Das sieht bereits sauberer aus als das Standard-Theme, aber die Schrift ist noch generisch, der Titel fehlt, und mehrere Elemente muessen verfeinert werden.

Schrift, Titel und Untertitel

Als naechstes wenden wir die Kanit-Schrift an, fuegen einen fetten Titel hinzu und verwenden `element_textbox_simple()` aus {ggtext} fuer einen Untertitel, der HTML-Formatierung unterstuetzt - das wird spaeter fuer farbkodierten Text nuetzlich:

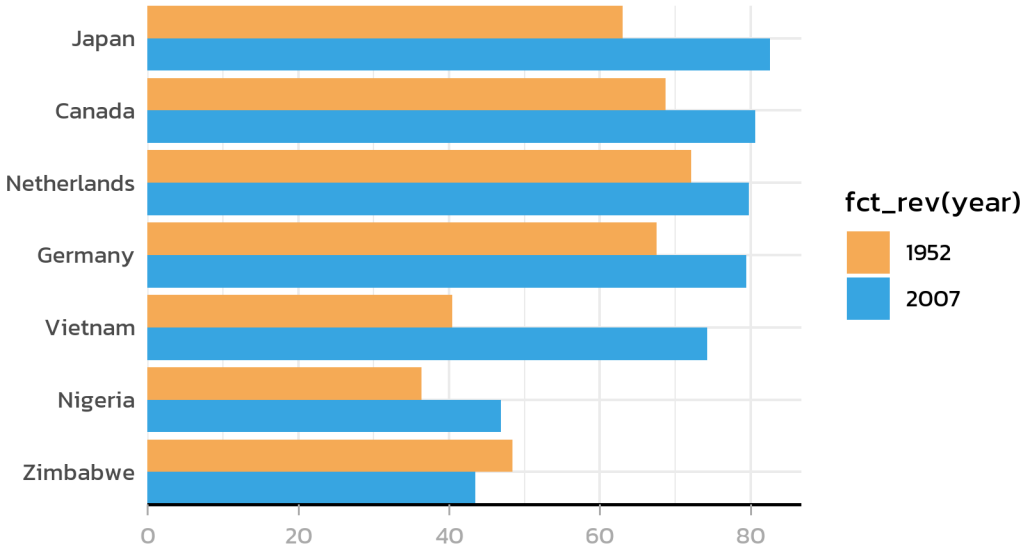
```
ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_manual(limits = names(year_colors), values = year_colors) +
  labs(title = "LEBENSERWARTUNG", subtitle = "Ein Vergleich von 1952 und 2007") +
  theme_minimal() +
  theme(
    text = element_text(family = "kanit"),
    plot.title.position = "plot",
    plot.title = element_text(size = 15, face = "bold"),
    plot.subtitle = ggtext::element_textbox_simple(
      size = 10, margin = margin(0, 0, 10, 0)
    )
  )
```



Achsenanpassung

Die y-Achsenlinie ist bei einem horizontalen Balkendiagramm unnoetig (die Balken selbst definieren die Kategorien). Die x-Achse bekommt eine saubere schwarze Linie mit dezenten grauen Tick-Marks:

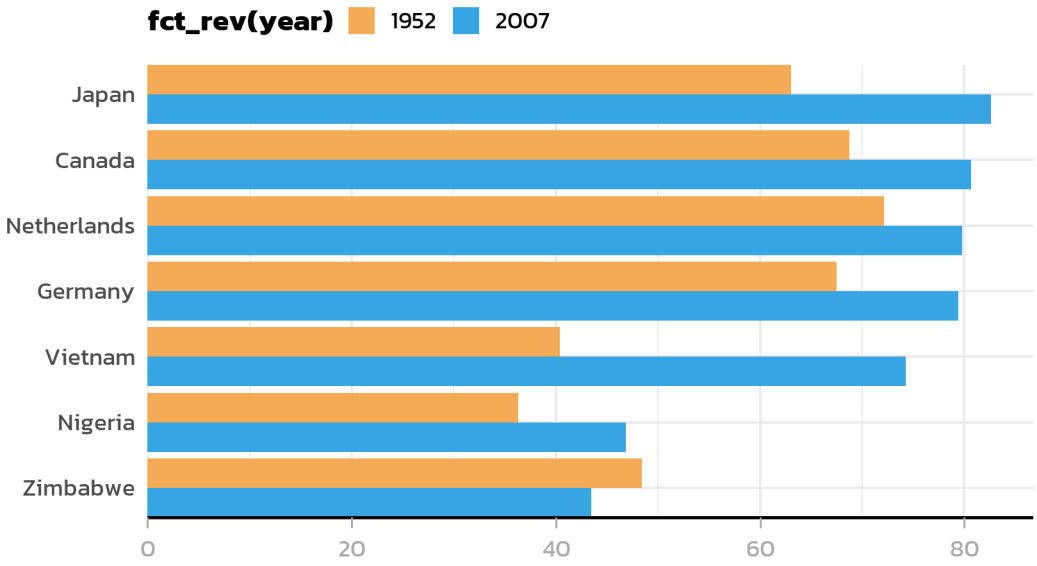
```
ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_manual(limits = names(year_colors), values = year_colors) +
  scale_y_discrete(name = NULL, expand = c(0, 0)) +
  scale_x_continuous(name = NULL, expand = expansion(mult = c(0, 0.05))) +
  labs(title = "LEBENSERWARTUNG", subtitle = "Ein Vergleich von 1952 und 2007") +
  theme_minimal() +
  theme(
    text = element_text(family = "kanit"),
    plot.title.position = "plot",
    plot.title = element_text(size = 15, face = "bold"),
    plot.subtitle = ggtext::element_textbox_simple(
      size = 10, margin = margin(0, 0, 10, 0)
    ),
    axis.line.y = element_blank(),
    axis.text.x = element_text(color = "#AAAAAA"),
    axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
    axis.ticks.length.x = unit(4, "pt"),
    axis.line.x = element_line(color = "black", linewidth = 0.6)
  )
```



Legendenpositionierung

Die Legende wandert in die obere linke Ecke. Ein subtiles aber wichtiges Detail: `legend.margin` mit einem negativen oberen Rand zieht die Legende naeher an den Untertitel und reduziert verschwendeten Weissraum:

```
ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_manual(limits = names(year_colors), values = year_colors) +
  scale_y_discrete(name = NULL, expand = c(0, 0)) +
  scale_x_continuous(name = NULL, expand = expansion(mult = c(0, 0.05))) +
  labs(title = "LEBENSERWARTUNG", subtitle = "Ein Vergleich von 1952 und 2007") +
  theme_minimal() +
  theme(
    text = element_text(family = "kanit"),
    plot.title.position = "plot",
    plot.title = element_text(size = 15, face = "bold"),
    plot.subtitle = ggtext::element_textbox_simple(
      size = 10, margin = margin(0, 0, 10, 0)
    ),
    axis.line.y = element_blank(),
    axis.text.x = element_text(color = "#AAAAAA"),
    axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
    axis.ticks.length.x = unit(4, "pt"),
    axis.line.x = element_line(color = "black", linewidth = 0.6),
    legend.position = "top",
    legend.box.just = "left",
    legend.justification = "left",
    legend.title = element_text(face = "bold"),
    legend.key.size = unit(0.4, "cm"),
    legend.margin = margin(-5, 0, 0, 0)
  )
```



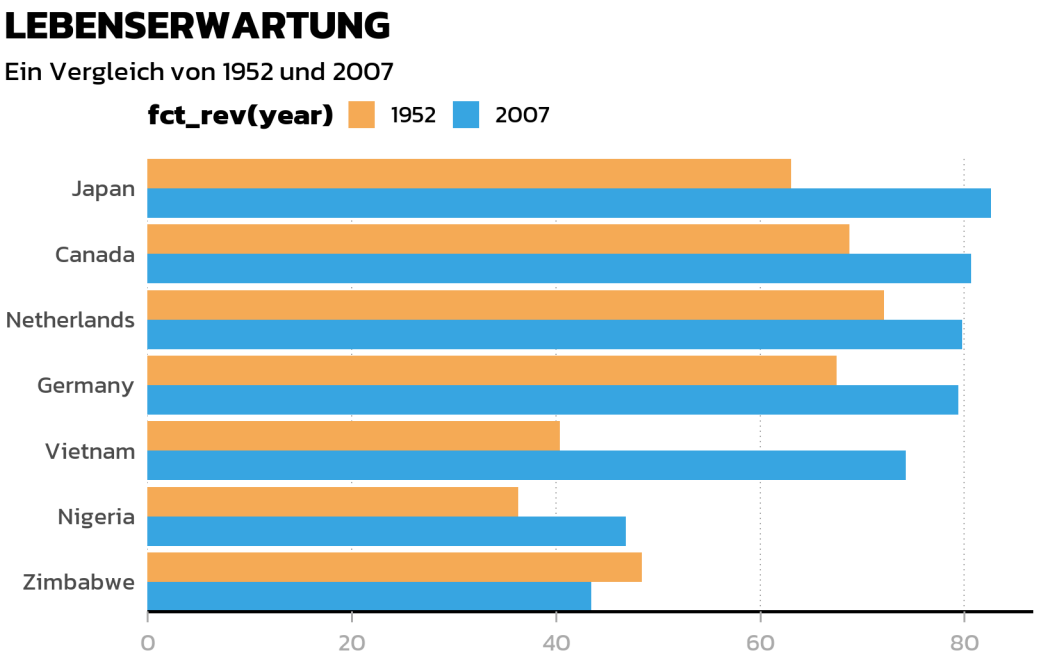
> **i** Der Trick mit dem negativen Margin
>
> `legend.margin = margin(-5, 0, 0, 0)` nutzt einen negativen oberen Rand, um die Legende nach oben zu ziehen, naeher an den Untertitel. Das ist ein gaengiger Trick, um das Layout zu straffen, wenn die Legende oben sitzt. Der Wert muss per Augenmass angepasst werden - ein zu grosser negativer Rand fuehrt dazu, dass die Legende den Untertitel ueberlappt.

Gitternetzlinien

Zum Schluss entfernen wir das Minor-Grid und die horizontalen Major-Gridlines (sie fuegen bei einem Balkendiagramm nur Unordnung hinzu) und behalten nur vertikale gepunktete Linien als dezente Referenzmarkierungen:

```
ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  scale_fill_manual(limits = names(year_colors), values = year_colors) +
  scale_y_discrete(name = NULL, expand = c(0, 0)) +
  scale_x_continuous(name = NULL, expand = expansion(mult = c(0, 0.05))) +
  labs(title = "LEBENSERWARTUNG", subtitle = "Ein Vergleich von 1952 und 2007") +
  theme_minimal() +
  theme(
    text = element_text(family = "kanit"),
    plot.title.position = "plot",
    plot.title = element_text(size = 15, face = "bold"),
    plot.subtitle = ggtext::element_textbox_simple(
      size = 10, margin = margin(0, 0, 10, 0)
    ),
    axis.line.y = element_blank(),
    axis.text.x = element_text(color = "#AAAAAA"),
    axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
    axis.ticks.length.x = unit(4, "pt"),
    axis.line.x = element_line(color = "black", linewidth = 0.6),
    legend.position = "top",
    legend.box.just = "left",
    legend.justification = "left",
```

```
    legend.title = element_text(face = "bold"),
    legend.key.size = unit(0.4, "cm"),
    legend.margin = margin(-5, 0, 0, 0),
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_blank(),
    panel.grid.major.x = element_line(
      linetype = "dotted", color = "#AAAAAA", linewidth = 0.3
    )
  )
```



Alles in eine Funktion verpacken

Nun, da wir mit jedem Element zufrieden sind, verpacken wir den gesamten `theme()`-Aufruf in eine wiederverwendbare Funktion. So ist das Anwenden des Themes auf jeden beliebigen Plot nur noch eine Zeile Code:

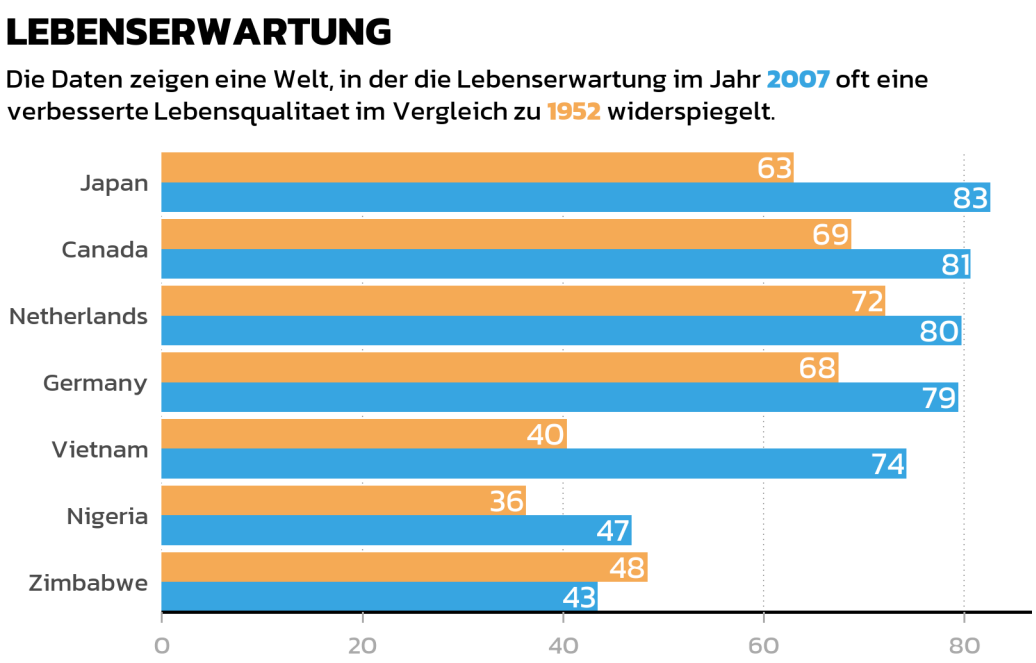
```
theme_nature <- function(base_size = 12) {
  theme_minimal(base_size = base_size) +
    theme(
      text = element_text(family = "kanit"),
      plot.title.position = "plot",
      plot.title = element_text(size = 15, face = "bold"),
      plot.subtitle = ggtext::element_textbox_simple(
        size = 10, margin = margin(0, 0, 10, 0)
      ),
      axis.line.y = element_blank(),
      axis.text.x = element_text(color = "#AAAAAA"),
      axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
      axis.ticks.length.x = unit(4, "pt"),
      axis.line.x = element_line(color = "black", linewidth = 0.6),
      legend.position = "top",
      legend.box.just = "left",
      legend.justification = "left",
      legend.title = element_text(face = "bold"),
      legend.key.size = unit(0.4, "cm"),
      legend.margin = margin(-5, 0, 0, 0),
      panel.grid.minor = element_blank(),
      panel.grid.major.y = element_blank(),
      panel.grid.major.x = element_line(
        linetype = "dotted", color = "#AAAAAA", linewidth = 0.3
```

```
      )
    )
}
```

Gestyltes Balkendiagramm mit Datenlabels

Jetzt kombinieren wir alles: das gruppierte Balkendiagramm, das Custom Theme, Datenlabels in den Balken und einen HTML-formatierten Untertitel, in dem die Farbkodierung direkt im Text erklaert wird - was eine separate Legende ersetzt:

```
long_subtitle <- "Die Daten zeigen eine Welt, in der die Lebenserwartung im Jahr <b
style='color:#37A9E1;'>2007</b> oft eine verbesserte Lebensqualitaet im Vergleich
zu <b style='color:#F7AA59;'>1952</b> widerspiegelt."

p <- ggplot(data = dat) +
  aes(x = lifeExp, y = country, fill = fct_rev(year)) +
  geom_col(position = position_dodge()) +
  geom_text(
    mapping = aes(label = round(lifeExp), group = fct_rev(year)),
    position = position_dodge(width = 0.9),
    hjust = 1.1,
    color = "white",
    family = "kanit"
  ) +
  scale_y_discrete(name = NULL, expand = c(0, 0)) +
  scale_x_continuous(
    name = NULL,
    expand = expansion(mult = c(0, 0.05))
  ) +
  scale_fill_manual(
    guide = "none",
    name = "Jahr",
    limits = names(year_colors),
    values = year_colors
  ) +
  labs(
    title = "LEBENSERWARTUNG",
    subtitle = long_subtitle
  ) +
  theme_nature()

p
```



Mehrere Details in diesem Code verdienen eine Erklaerung:

- `group = fct_rev(year)` **in** `geom_text()`: Ohne eine explizite `group`-Aesthetic weiss `geom_text()` nicht, zu welchem Balken jedes Label gehoert, und kann die Labels nicht korrekt dodgen. Das `group` muss zur Fill-Aesthetic passen (einschliesslich `fct_rev()`), damit Labels und Balken auf exakt die gleiche Weise gedodgt werden.
- `position_dodge(width = 0.9)`: Die Standard-Dodge-Breite von `geom_col()` ist 0.9. Wenn man Labels oder andere Layer auf gedodgte Balken legt, muss man diese Breite explizit angeben - sonst dodgen die Labels mit einer anderen Breite als die Balken, was zu Fehlausrichtung fuehrt.
- `family = "kanit"` **in** `geom_text()`: Anders als die meisten Theme-Elemente werden Texte in Geoms *nicht* von den `theme()`-Einstellungen beeinflusst. Wenn der Plot eine benutzerdefinierte Schrift verwendet, muss sie in jedem `geom_text()`- oder `geom_label()`-Aufruf explizit angegeben werden.
- `guide = "none"`: Entfernt die Legende, da der farbkodierte Untertitel bereits die Bedeutung jeder Farbe erklaert - ein saubereres Design, das visuelle Unordnung reduziert.

! Geoms erben nicht die Font-Family vom Theme

Das ist eine haeufige Fehlerquelle: `theme(text = element_text(family = "kanit"))` setzt die Schrift fuer *Theme-Elemente* (Titel, Achsenlabels, Legendentext), aber **nicht** fuer Text, der von Geoms wie `geom_text()` oder `geom_label()` gezeichnet wird. Diese Geoms verwenden die Standardschrift, es sei denn, `family` wird explizit gesetzt. Vergisst man das, entstehen Plots, bei denen der Titel eine Schrift verwendet und die Datenlabels eine andere.

Legenden-Alternative: Farbkodierter Untertitel

Anstelle einer traditionellen Legende nutzt der gestylte Untertitel HTML `<b style='color:...'>` Tags, um die Jahreslabels direkt im Text einzufaerben. Das funktioniert nur, wenn der Untertitel `element_textbox_simple()` aus ggtext verwendet (was unser `theme_nature()` tut). Der Vorteil ist ein saubereres Design - der Leser sieht die Farbkodierung im Kontext, anstatt Legendeneintraege mit Plot-Elementen abgleichen zu muessen.

Checkpoint: Gruppiertes Balkendiagramm

Dieser eigenstaendige Code-Block reproduziert das gestylte Balkendiagramm von Grund auf:

```
library(tidyverse)
library(gapminder)
library(showtext)
library(ggtext)

showtext_opts(dpi = 300)
font_add_google("Kanit", "kanit")
showtext_auto()

dat <- gapminder %>%
  filter(year %in% c(1952, 2007)) %>%
  filter(country %in% c(
    "Canada", "Germany", "Japan",
    "Netherlands", "Nigeria", "Vietnam", "Zimbabwe"
  )) %>%
  mutate(year = as.factor(year)) %>%
  droplevels()

sorted_countries <- dat %>%
  filter(year == "2007") %>%
  arrange(lifeExp) %>%
  pull(country) %>%
  as.character()

dat <- dat %>%
  mutate(country = fct_relevel(country, sorted_countries))

year_colors <- c("1952" = "#F7AA59", "2007" = "#37A9E1")

theme_nature <- function(base_size = 12) {
  theme_minimal(base_size = base_size) +
    theme(
      text = element_text(family = "kanit"),
      plot.title.position = "plot",
      plot.title = element_text(size = 15, face = "bold"),
      plot.subtitle = element_textbox_simple(
        size = 10, margin = margin(0, 0, 10, 0)
      ),
      axis.line.y = element_blank(),
      axis.text.x = element_text(color = "#AAAAAA"),
      axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
      axis.ticks.length.x = unit(4, "pt"),
      axis.line.x = element_line(color = "black", linewidth = 0.6),
      legend.position = "top",
      legend.box.just = "left",
      legend.justification = "left",
      legend.title = element_text(face = "bold"),
      legend.key.size = unit(0.4, "cm"),
      legend.margin = margin(-5, 0, 0, 0),
      panel.grid.minor = element_blank(),
      panel.grid.major.y = element_blank(),
      panel.grid.major.x = element_line(
        linetype = "dotted", color = "#AAAAAA", linewidth = 0.3
      )
    )
}

long_subtitle <- "Die Daten zeigen eine Welt, in der die Lebenserwartung im Jahr
<b style='color:#37A9E1;'>2007</b> oft eine verbesserte Lebensqualitaet im
Vergleich zu <b style='color:#F7AA59;'>1952</b> widerspiegelt."
```

LEBENSERWARTUNG

Die Daten zeigen eine Welt, in der die Lebenserwartung im Jahr **2007** oft eine verbesserte Lebensqualitaet im Vergleich zu **1952** widerspiegelt.

Dumbbell Plot

Ein gruppiertes Balkendiagramm eignet sich gut zum Vergleich absoluter Werte, aber wenn der Fokus auf der *Veraenderung* zwischen zwei Zeitpunkten liegt, erzaehlt ein Dumbbell Plot die Geschichte oft klarer. Er verbindet zwei Datenpunkte mit einem Liniensegment, sodass Richtung und Ausmass der Veraenderung sofort sichtbar werden - besonders wenn manche Laender sich dramatisch verbessert haben, waehrend andere sich kaum veraendert haben.

Wir benoetigen zunaechst eine Wide-Format-Version der Daten, mit einer Spalte pro Jahr, damit `geom_segment()` Linien vom 1952-Wert zum 2007-Wert zeichnen kann:

```
dat_wide <- dat %>%
  select(country, year, lifeExp) %>%
  pivot_wider(
    names_from = year,
    values_from = lifeExp,
    names_prefix = "year_"
  ) %>%
  mutate(
    max_x = pmax(year_2007, year_1952),
    diff = year_2007 - year_1952,
    diff_lab = sprintf("%+d", round(diff))
  )

dat_wide
```

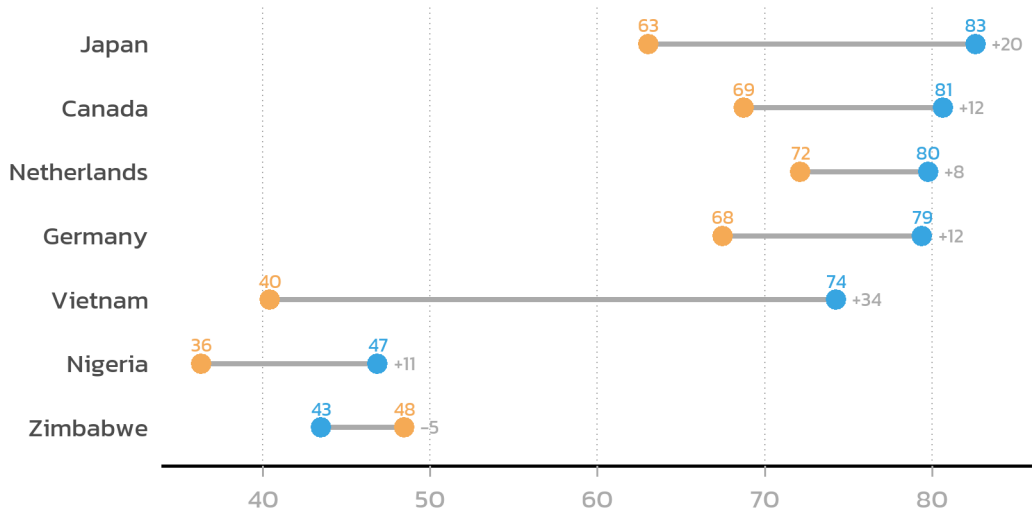
```
# A tibble: 7 × 6
  country     year_1952 year_2007 max_x  diff diff_lab
  <fct>           <dbl>     <dbl> <dbl> <dbl> <chr>
1 Canada           68.8      80.7  80.7 11.9  +12
2 Germany          67.5      79.4  79.4 11.9  +12
3 Japan            63.0      82.6  82.6 19.6  +20
4 Netherlands      72.1      79.8  79.8  7.63 +8
5 Nigeria          36.3      46.9  46.9 10.5  +11
6 Vietnam          40.4      74.2  74.2 33.8  +34
7 Zimbabwe         48.5      43.5  48.5 -4.96 -5
```

Die `pmax()`-Funktion findet den groesseren der beiden Jahreswerte fuer jedes Land - das brauchen wir spaeter, um die Differenzlabels rechts von jedem Dumbbell zu positionieren. Die `sprintf("%+d", ...)`-Formatierung stellt sicher, dass positive Veraenderungen ein `+`-Zeichen anzeigen (wie “+38”) fuer sofortige visuelle Klarheit.

Der Dumbbell Plot verwendet drei Layer: `geom_segment()` fuer die Verbindungslinien, `geom_point()` fuer die farbigen Endpunkte und zwei `geom_text()`-Layer fuer die Wertlabels und Differenzlabels:

```
p2 <- ggplot(data = dat) +
  aes(x = lifeExp, y = country, color = fct_rev(year)) +
  geom_segment(
    data = dat_wide,
    aes(x = year_1952, xend = year_2007, y = country, yend = country),
    color = "#AAAAAA",
    linewidth = 1
  ) +
  geom_point(size = 3) +
  geom_text(
    mapping = aes(label = round(lifeExp)),
    size = 2.5,
    vjust = -1,
    family = "kanit"
```

```
  ) +
  geom_text(
    data = dat_wide,
    mapping = aes(x = max_x, label = diff_lab),
    size = 2.5,
    hjust = 0,
    position = position_nudge(x = 1),
    color = "#AAAAAA",
    family = "kanit"
  ) +
  scale_color_manual(
    name = "Jahr",
    limits = c("1952", "2007"),
    values = year_colors,
    guide = "none"
  ) +
  scale_y_discrete(name = NULL) +
  scale_x_continuous(name = NULL) +
  labs(
    title = "LEBENSERWARTUNG",
    subtitle = long_subtitle
  ) +
  theme_nature()

p2
```



Der erste `geom_text()` zeigt die tatsaechlichen Werte ueber jedem farbigen Punkt. Der zweite `geom_text()` nutzt die Wide-Format-Daten, um Differenzlabels (wie “+38” oder “+3”) rechts von jedem Dumbbell zu platzieren - `position_nudge(x = 1)` verschiebt das Label leicht, damit es nicht mit dem aeussersten Punkt ueberlappt.

Man beachte, dass der Segment-Layer ein separates `data`-Argument (`data = dat_wide`) verwendet, waehrend der Point-Layer das originale Long-Format `dat` nutzt. Das ist ein haeufiges Muster in ggplot2: Verschiedene Layer koennen aus verschiedenen Datenquellen zeichnen, solange die Aesthetics uebereinstimmen.

Checkpoint: Dumbbell Plot

Dieser eigenstaendige Code-Block reproduziert den Dumbbell Plot von Grund auf:

```
library(tidyverse)
library(gapminder)
library(showtext)
library(ggtext)

showtext_opts(dpi = 300)
font_add_google("Kanit", "kanit")
showtext_auto()

dat <- gapminder %>%
  filter(year %in% c(1952, 2007)) %>%
  filter(country %in% c(
    "Canada", "Germany", "Japan",
    "Netherlands", "Nigeria", "Vietnam", "Zimbabwe"
  )) %>%
  mutate(year = as.factor(year)) %>%
  droplevels()

sorted_countries <- dat %>%
  filter(year == "2007") %>%
  arrange(lifeExp) %>%
  pull(country) %>%
  as.character()

dat <- dat %>%
  mutate(country = fct_relevel(country, sorted_countries))

year_colors <- c("1952" = "#F7AA59", "2007" = "#37A9E1")

theme_nature <- function(base_size = 12) {
  theme_minimal(base_size = base_size) +
    theme(
      text = element_text(family = "kanit"),
      plot.title.position = "plot",
      plot.title = element_text(size = 15, face = "bold"),
      plot.subtitle = element_textbox_simple(
        size = 10, margin = margin(0, 0, 10, 0)
      ),
      axis.line.y = element_blank(),
      axis.text.x = element_text(color = "#AAAAAA"),
      axis.ticks.x = element_line(color = "#AAAAAA", linewidth = 0.4),
      axis.ticks.length.x = unit(4, "pt"),
      axis.line.x = element_line(color = "black", linewidth = 0.6),
      legend.position = "top",
      legend.box.just = "left",
      legend.justification = "left",
      legend.title = element_text(face = "bold"),
      legend.key.size = unit(0.4, "cm"),
      legend.margin = margin(-5, 0, 0, 0),
      panel.grid.minor = element_blank(),
      panel.grid.major.y = element_blank(),
      panel.grid.major.x = element_line(
        linetype = "dotted", color = "#AAAAAA", linewidth = 0.3
      )
    )
}

long_subtitle <- "Die Daten zeigen eine Welt, in der die Lebenserwartung im Jahr
<b style='color:#37A9E1;'>2007</b> oft eine verbesserte Lebensqualitaet im
Vergleich zu <b style='color:#F7AA59;'>1952</b> widerspiegelt."

dat_wide <- dat %>%
  select(country, year, lifeExp) %>%
```

LEBENSERWARTUNG

Die Daten zeigen eine Welt, in der die Lebenserwartung im Jahr 2007 oft eine verbesserte Lebensqualitaet im Vergleich zu 1952 widerspiegelt.

Bibliography