

Datenanalyse und -visualisierung

mit der Programmiersprache R

Datenanalyse und -visualisierung

mit der Programmiersprache R

Pekka Sagner M.Sc.

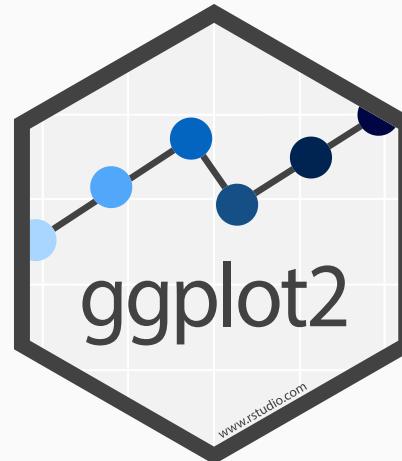
-  sagner@iwkoeln.de
-  iwkoeln.de/.../pekka-sagner
-  [@pekkasagner](https://twitter.com/pekkasagner)

Wintersemester 2022-23

Grammar of graphics (ggplot)

Inhalte und Ziele der Sitzung

- Einführung in die Visualisierung mit ggplot2
- Verständnis der verschiedenen Abbildungsoptionen
- Erkennen des Werts von Abbildungen für die explorative Datenanalyse



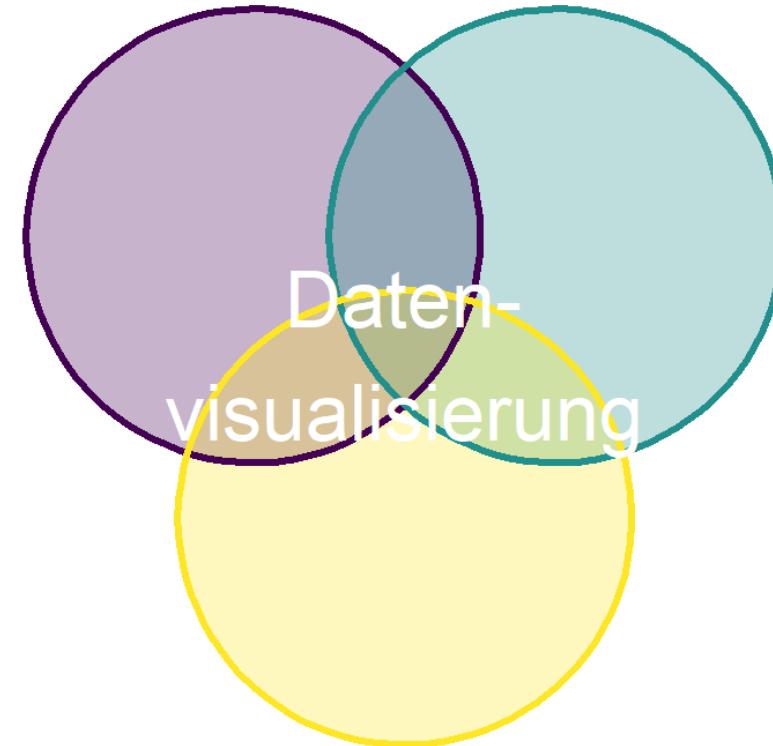
Quelle: github/rstudio.com

- Leseempfehlung:
 - R4DS, Kapitel 3 (Wickham/Grolemund, 2021)
 - A Layered Grammar of Graphics (Wickham, 2010)
 - ggplot2: Elegant Graphics for Data Analysis (Wickham, 2021)
 - R Graphics Coobook (Chang, 2021)

Warum Datenvisualisierung?

Warum Datenvisualisierung?

Programmieren Statistik



Kommunikation

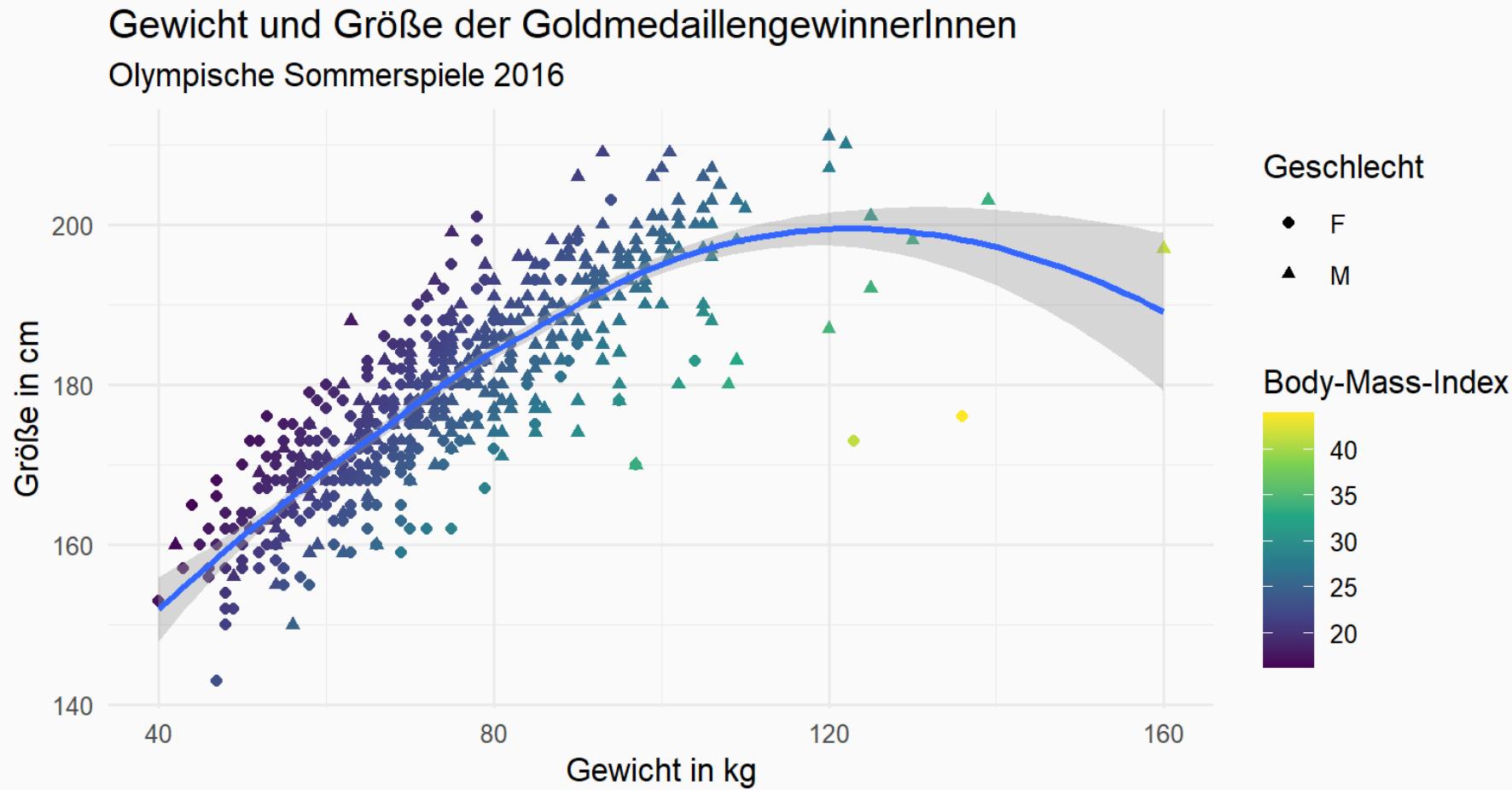
Warum Datenvisualisierung?

- Visualisierungen sind auf dem Weg zum besseren Verständnis von Daten ein unerlässliches Instrument.
- Eine gute Visualisierung zeigt möglicherweise unerwartete Dinge.
- Oft zeigen uns Abbildungen, dass **irgendetwas in den Daten nicht stimmt**.
- Sie sind deshalb ein mächtiges Instrument und sollten **immer Teil der explorativen Datenanalyse** sein.
- R ist wegen seiner guten Visualisierungsmöglichkeiten, die auf einer **Grammar of Graphics** basieren, sehr beliebt.

Aufbau eines Plots

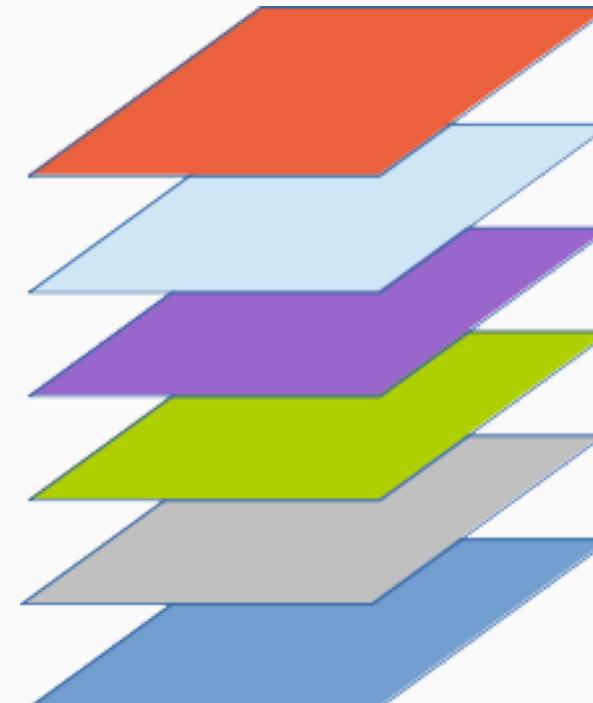
Der Aufbau eines Plots

Das Ziel



Elemente eines ggplot: Schicht für Schicht

1. **data**: Der Datensatz, den wir visualisieren wollen.
2. **mapping**: Grundstruktur des Plots
 - Ergibt sich aus den Daten.
 - Welche Variable wird welcher Achse zugewiesen?
 - Sollen Farben, Formen oder Größe von Variablenausprägungen abhängen?
3. **geoms**: Definiert den Plottyp, bzw. die Elemente im Plot.
4. **labs**: Titel, Untertitel, Quellenverweise und Co.
5. **themes**: Allgemeiner »Look« des Plots.
6. **scales**: Anpassung der Aesthetics.



Quelle: datasciencentral.com

1. Data

- Vor der Datenvisualisierung kommt die -analyse, kommt die -visualisierung, kommt die...
- Bevor wir uns einen Plot erstellen können, müssen wir uns zunächst im Klaren werden, was wir darstellen möchten.
- Im Folgenden Beispiel ist dies der Zusammenhang von **Gewicht** und **Körpergröße** von **AthletInnen**, die bei den Olympischen **Sommerspielen 2016** eine **Goldmedaille** gewonnen haben. Wir möchten außerdem den **Body-Mass-Index** visualisieren.

```
olympics_daten_plot <- olympics %>%
  filter(medal == "Gold") %>%
  filter(games == "2016 Summer") %>%
  mutate(bmi = weight / (height/100)^2)
```

1. Data: ggplot ()

```
ggplot(data = olympics_daten_plot)
```

- Plots werden mit `ggplot()` eingeleitet.
- In der Regel erhält dieser Befehl zwei Inputs:
 - `data`: Das Objekt (tibble), in dem die Daten stecken.
 - `mapping`: Die Struktur des Plots (Variablen auf Achsen, Farben, Größe), die auf Variablen im Datensatz zurückzuführen ist.

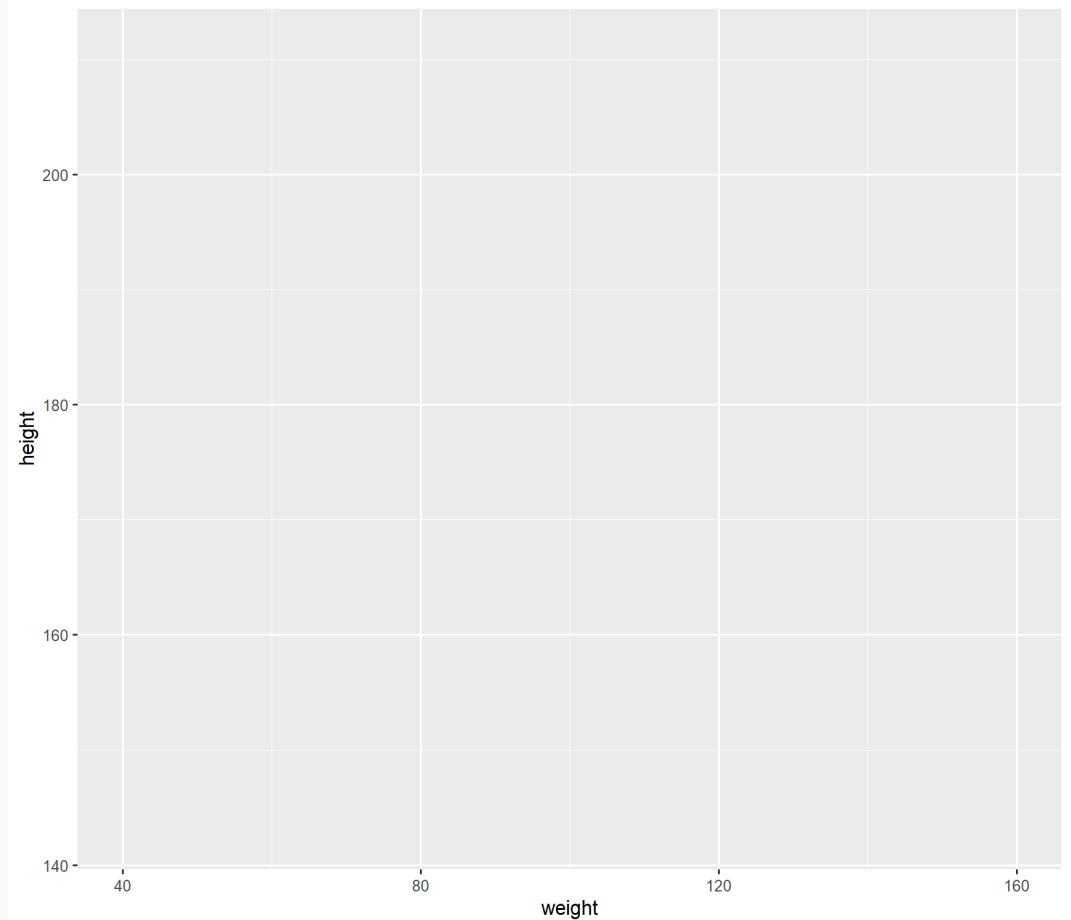
2. Mapping: aes ()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height))
```

- Es steht eine ganze Reihe an Aesthetics zur Verfügung, z. B.:
- `aes (`

- `x`, `y` → Achsen
- `color`, `fill` → Farben
- `alpha` → Transparenz
- `size` → Größe
- `shape` → Form
- `linetype` → Linientyp

)



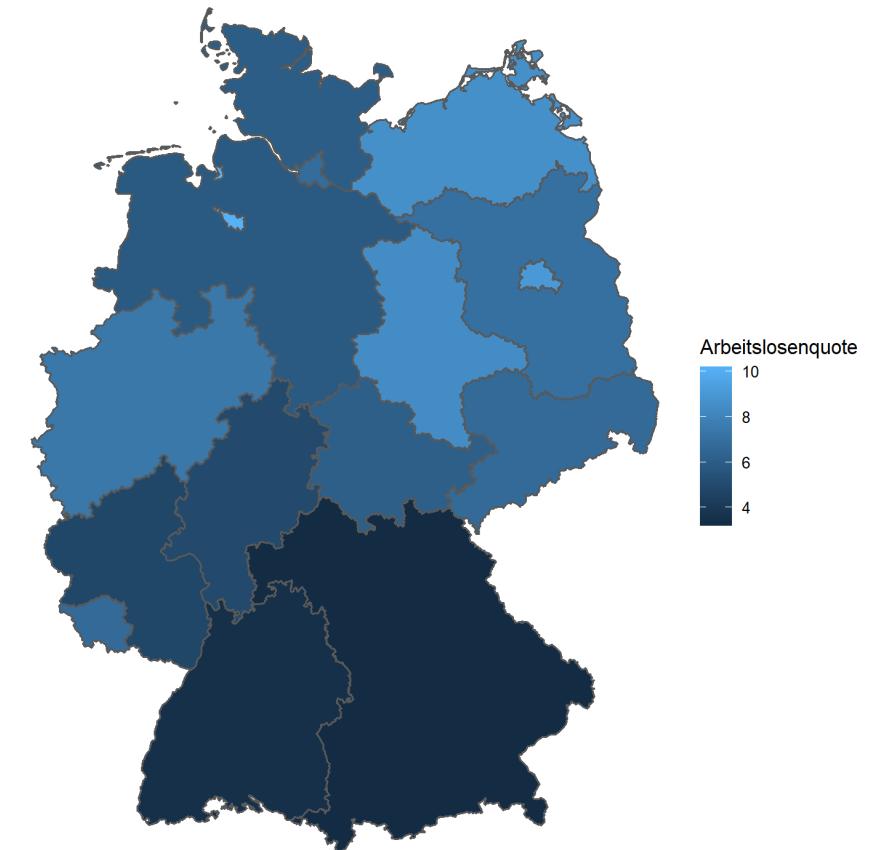
2. Mapping: aes ()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height))
```

- Äquivalent zum obigen Code ist:

```
ggplot(olympics_daten_plot,  
       aes(x = weight, y = height))
```

- Bisher sehen wir nur ein leeres Panel - dank `x` und `y` in `aes()` mit einer x-Achsen- und y-Achsen-Beschriftung.
- Dies liegt daran, dass ggplot bisher nur weiß, **was** es plotten soll, aber noch nicht **wie**.



Elemente eines Plots verbinden

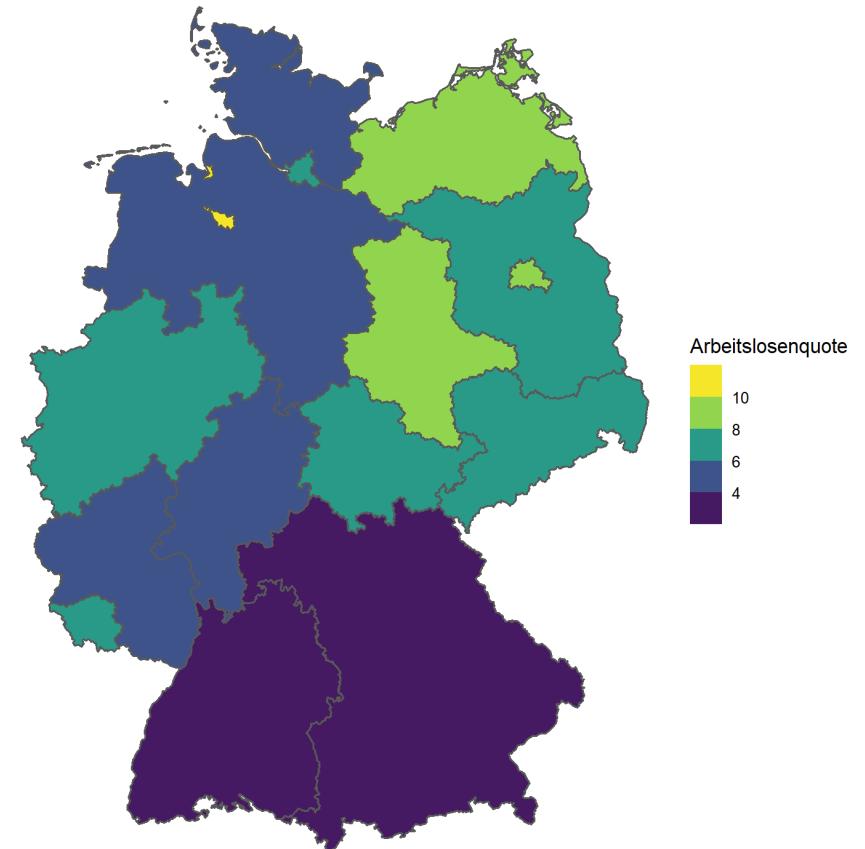
- Die Logik, verschiedene Elemente eines Plots zu gestalten und miteinander zu verbinden, ist ähnlich der Pipe-Struktur.
- Aber wir benutzen ein Pluszeichen anstelle der Pipe.

+ statt %>%

3. Geoms: geom_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point()
```

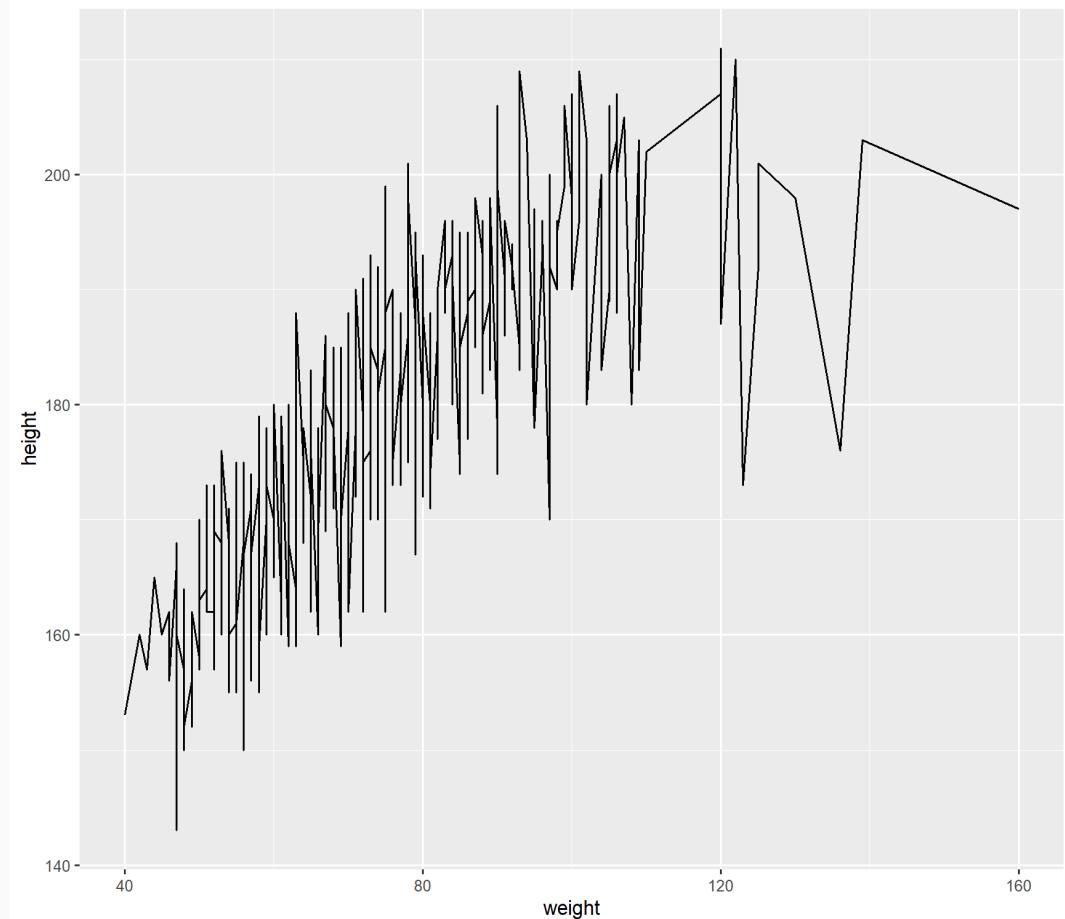
- Die Liste an `geom_`-Optionen ist lang, z. B.:
 - `geom_col()` → Säulendiagramme
 - `geom_line()` → Liniendiagramme
 - `geom_boxplot()` → Boxplots
 - `geom_smooth()` → Trends



3. Geoms: geom_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_line()
```

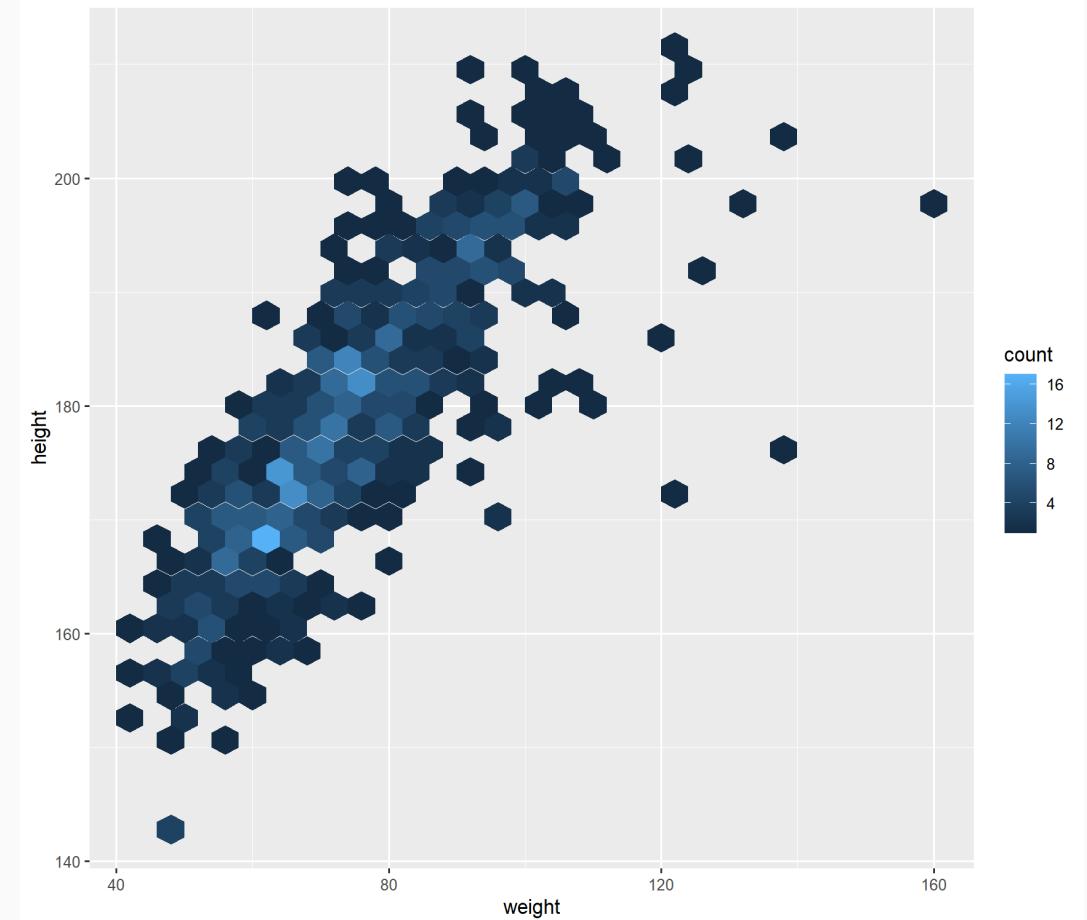
- Die Wahl des »richtigen« **geoms** hängt von
 - den **Daten**
 - und der Message, die wir transportieren möchten, ab.
- Ein Liniendiagramm ist für diese mappings auf die x- und y-Achse nicht sinnvoll.



3. Geoms: geom_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_hex()
```

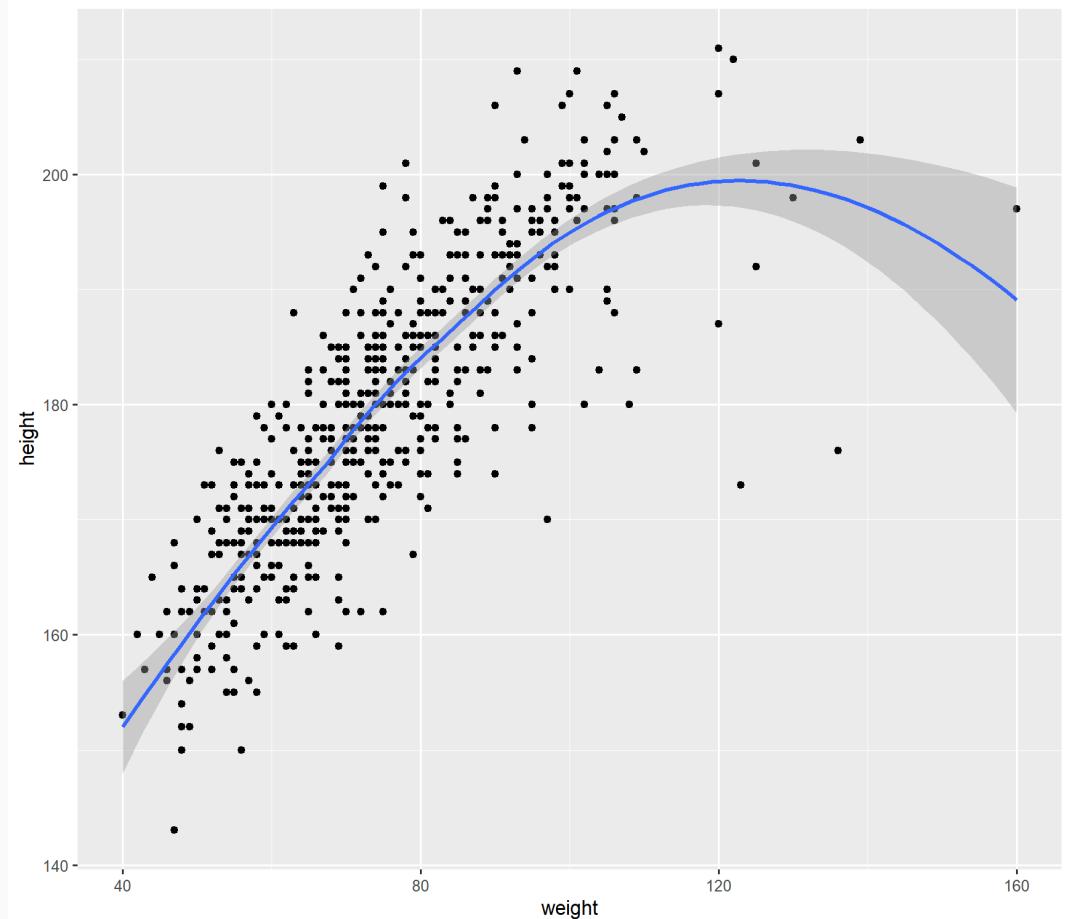
- Die Wahl des »richtigen« **geoms** hängt von
 - den Daten
 - und der **Message**, die wir transportieren möchten, ab.
- Der etwas außergewöhnliche Hexplot betont die Dichte (Häufigkeit) von Gewichts-Größen-Kombinationen in einem gewissen Bereich.
- Steht eben genau die Häufigkeit der Gewichts-Größen-Kombinationen im Fokus **kann** diese Plotart sinnvoll sein.



3. Geoms: geom_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point() +  
  geom_smooth()
```

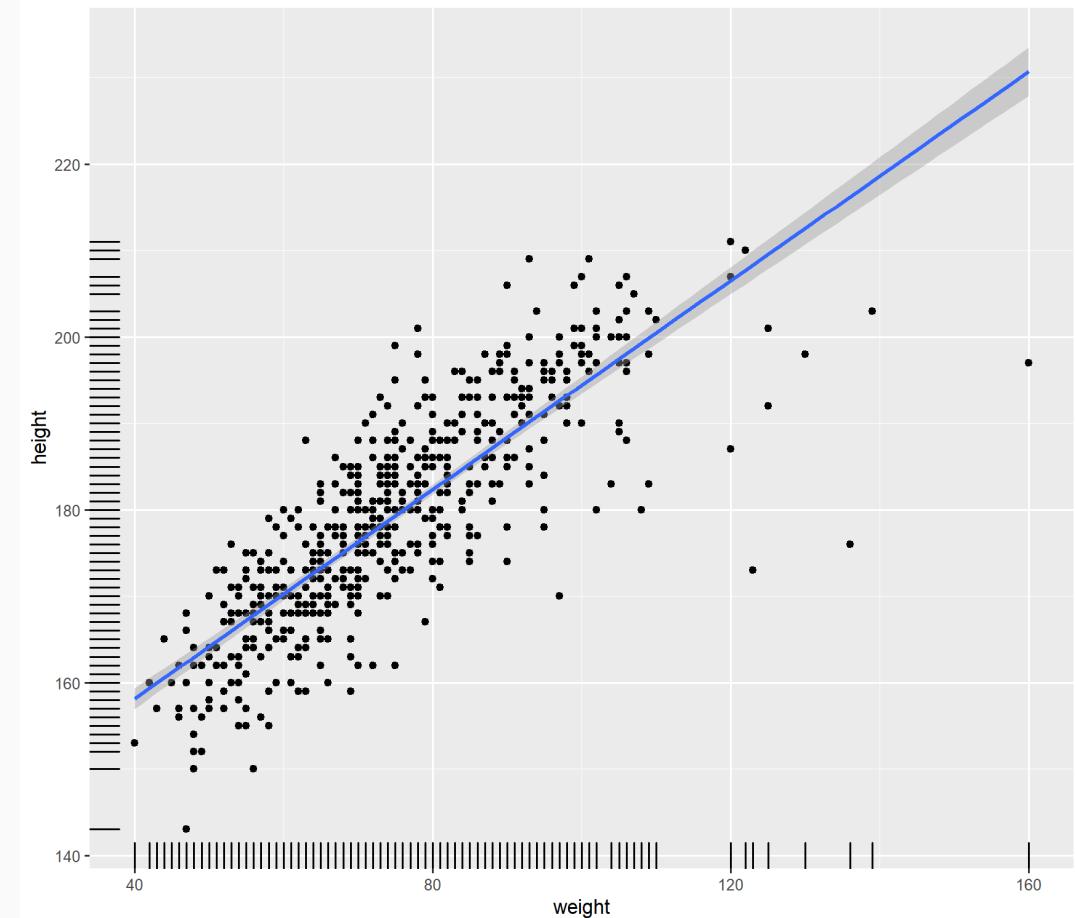
- Geoms können miteinander kombiniert werden.
- `geom_smooth()` fügt eine Trendlinie ein.
Standardmäßig mit der **LOESS**-Methode (locally weighted scatterplot smoothing) oder mit der **GAM**-Methode (generalized additive model) (GAM, falls ≥ 1.000 Beobachtungen).
- `geom_smooth(method = "lm")` fügt eine lineare Trendlinie ein (linear model).



3. Geoms: geom_...()

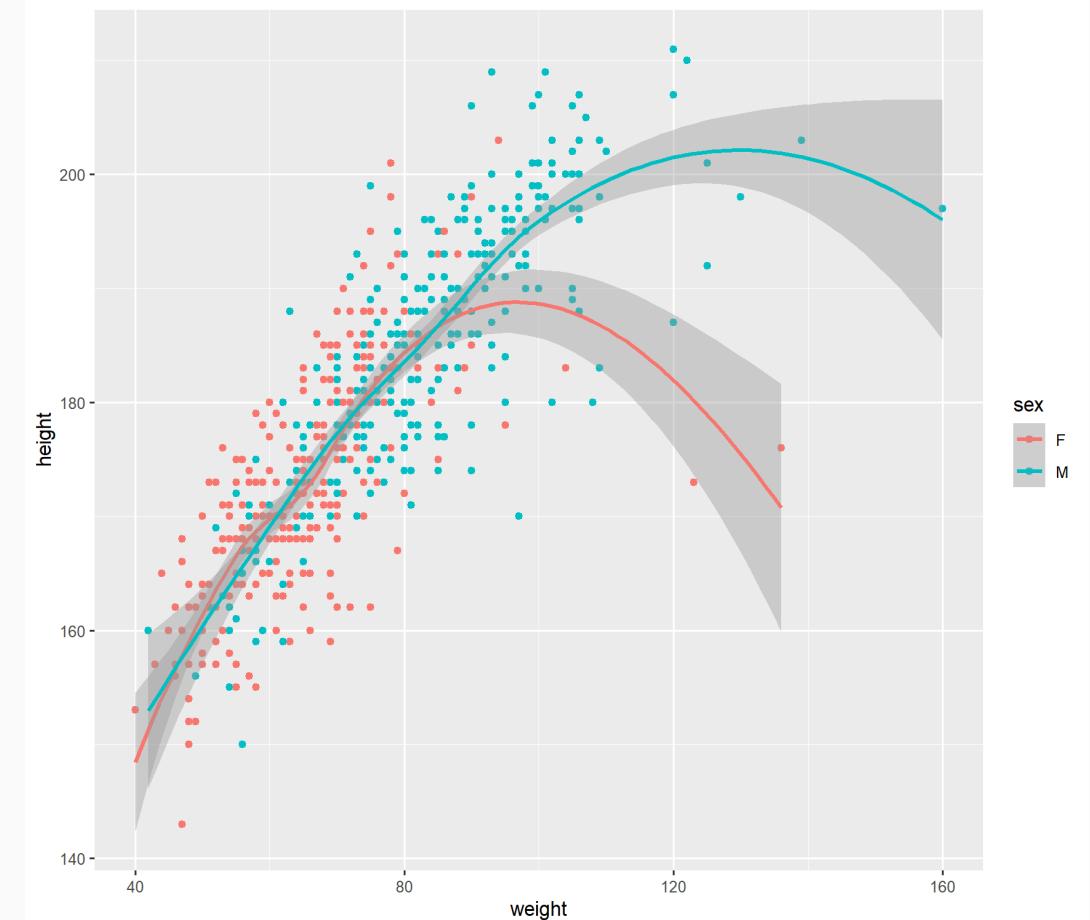
```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point() +  
  geom_smooth(method = "lm") +  
  geom_rug()
```

- Im Prinzip lassen sich beliebig viele Geoms miteinander kombinieren.
- Man sollte bei der Wahl und der Menge der Geoms jedoch auch immer seine **Zielgruppe bedenken**.
- Niemandem (außer vielleicht dem Autor selbst) ist gedient, wenn ein Plot so komplex wird, dass er von der Zielgruppe nicht verstanden wird.



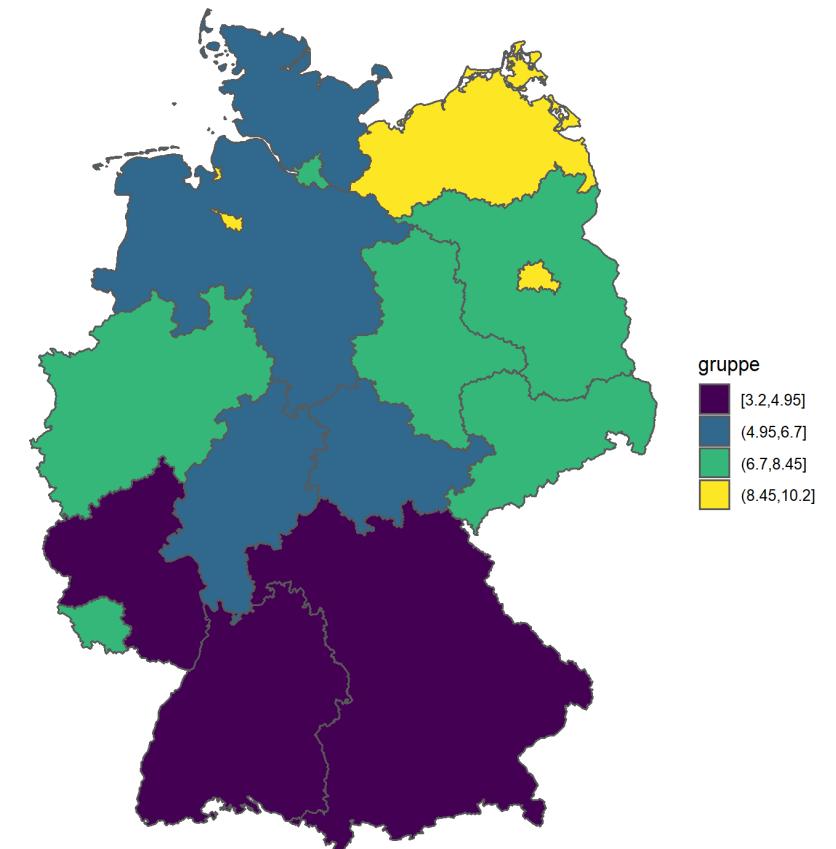
```
ggplot(data = olympics_daten_plot,
       mapping = aes(x = weight, y = height,
                     col = sex)) +
  geom_point() +
  geom_smooth()
```

- Geoms **»erben«** ihre Eigenschaften von `aes()` in `ggplot()`.
- Das kann erwünscht sein.
- Falls nicht, so sollten die jeweiligen Aesthetics direkt in die Geoms eingefügt werden.



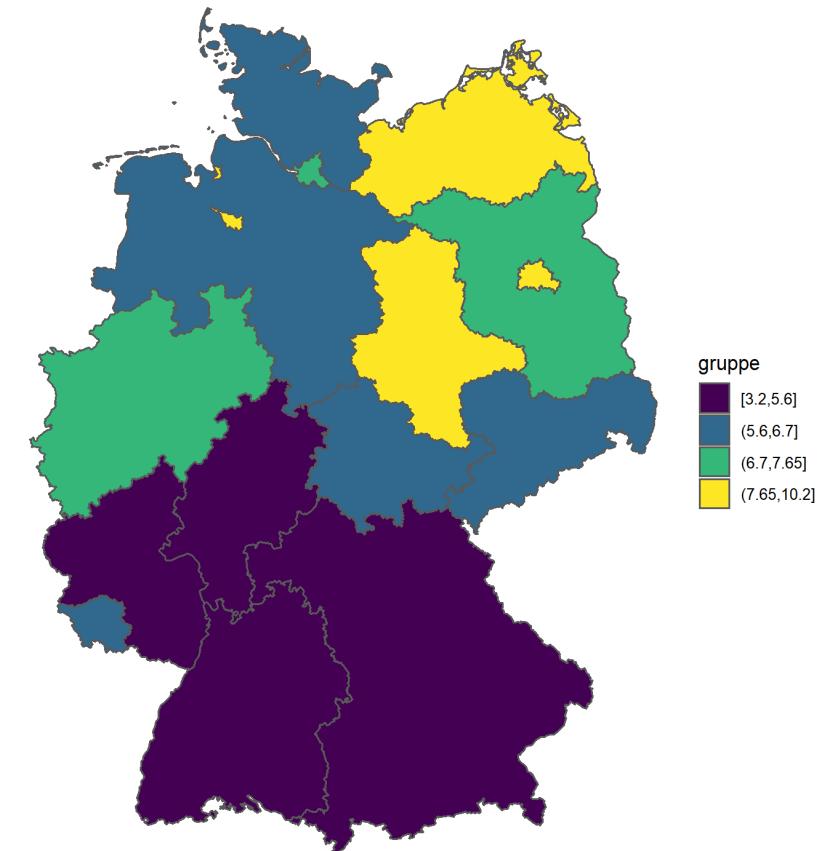
```
ggplot(data = olympics_daten_plot,
       mapping = aes(x = weight, y = height)) +
  geom_point(aes(col = sex)) +
  geom_smooth()
```

- Geoms »erben« ihre Eigenschaften von `aes()` in `ggplot()`.
- Das kann erwünscht sein.
- Falls nicht, so sollten die jeweiligen Aesthetics direkt in die Geoms eingefügt werden.
- Hier »erbt« `geom_smooth()` lediglich die Aesthetics `x` und `y`.
- Die Aesthetic `col = sex` ist alleinig dem `geom_point()` zugewiesen.



```
ggplot(data = olympics_daten_plot,
       mapping = aes(x = weight, y = height)) +
  geom_point(aes(col = bmi, shape = sex)) +
  geom_smooth()
```

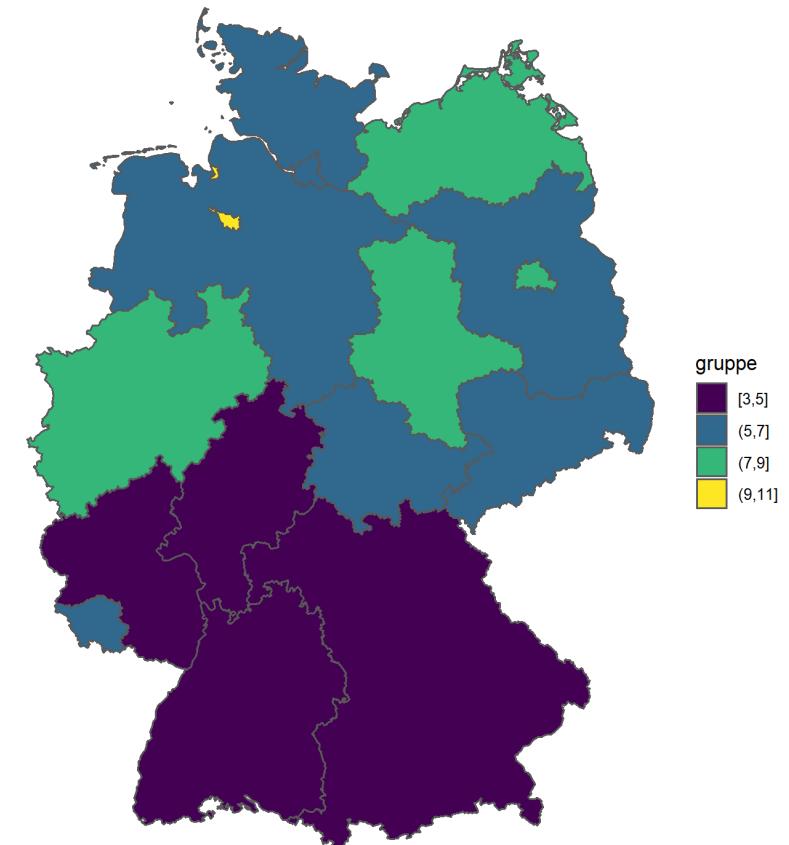
- Auch hier können wir Aesthetics wieder kombinieren.
- So lassen sich einfach zusätzliche Dimensionen in einen Plot einbringen.
- `col = bmi` weist dem BMI eine kontinuierliche Farbskala zu, da es sich um eine numerische Variable handelt.
- Zuvor, bei `col = sex`, war es eine Character-Variable und der Variable wurde eine diskrete Farbskala zugewiesen.
- Die `shape`-Aesthetic bietet sich beispielsweise für diskrete Variablen mit wenigen Ausprägungen an.



4. Beschriftungen: labs (...)

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  labs(  
    title = "Gewicht und Größe der GoldmedaillengewinnerInnen"  
    subtitle = "Olympische Sommerspiele 2016",  
    x = "Gewicht in kg",  
    y = "Größe in cm",  
    caption = "Quelle: TidyTuesday",  
    shape = "Geschlecht",  
    color = "Body-Mass-Index")
```

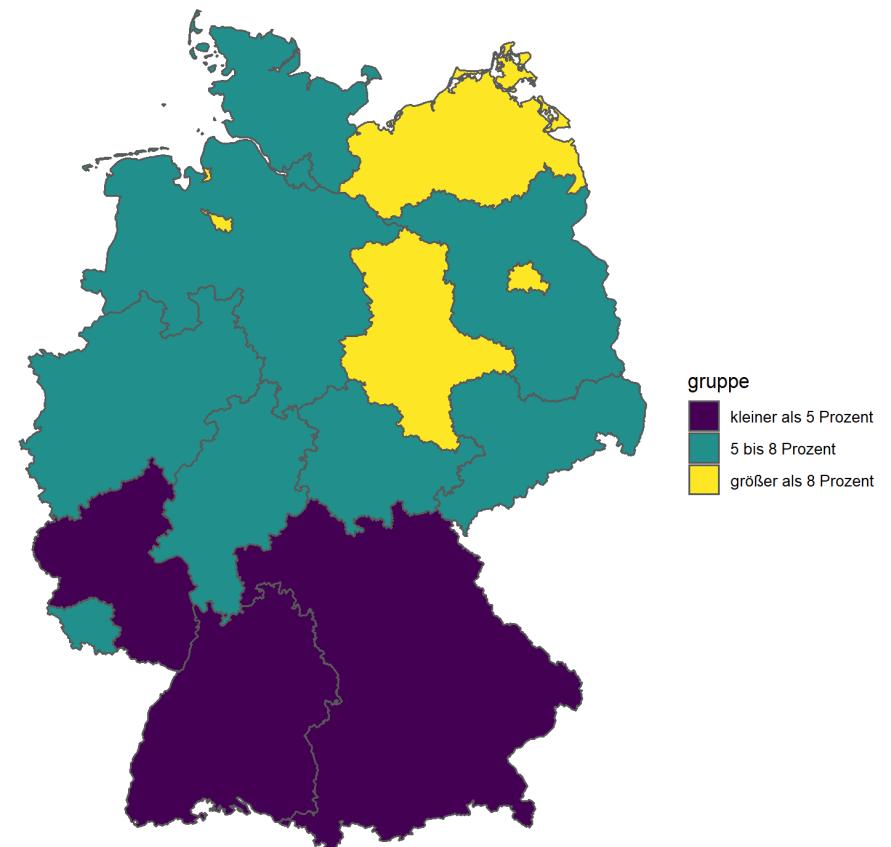
- Es gibt verschiedene Möglichkeiten, die Achsenbeschriftungen, Titel und Legenden anzupassen.
- Eine der einfachsten ist über die `labs()`-Funktion.



5. theme_...(...)

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  theme_bw()
```

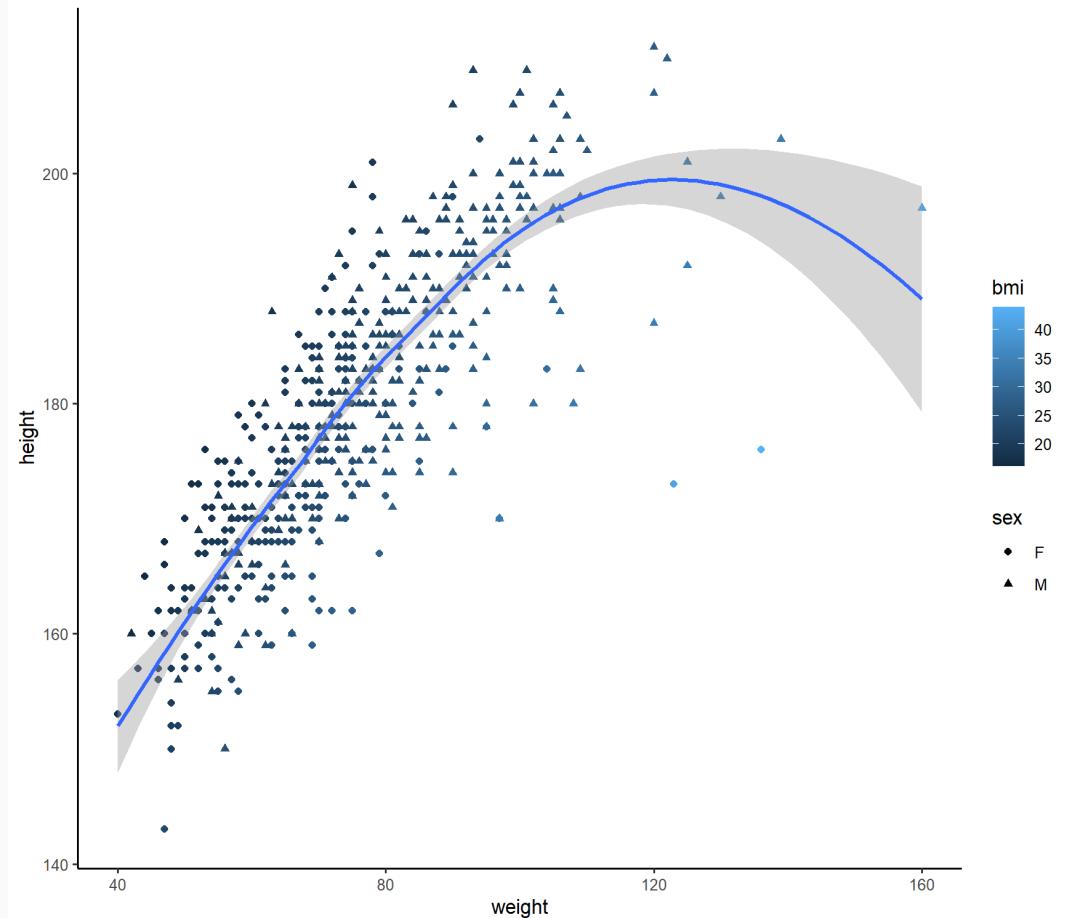
- `theme_...(...)` bietet vordefinierte Einstellungen, die das Aussehen des Plots bestimmen.
- Beispiele für vordefinierte Themes:
 - `theme_bw()`
 - `theme_classic()`
 - `theme_void()`
 - `theme_gray()`
- Verschiedene Pakete bieten zahlreiche weitere Themes, z. B.:
 - `ggthemes::theme_economist()`
 - `ggthemes::theme_excel_new()`



5. theme_...(...)

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  theme_classic()
```

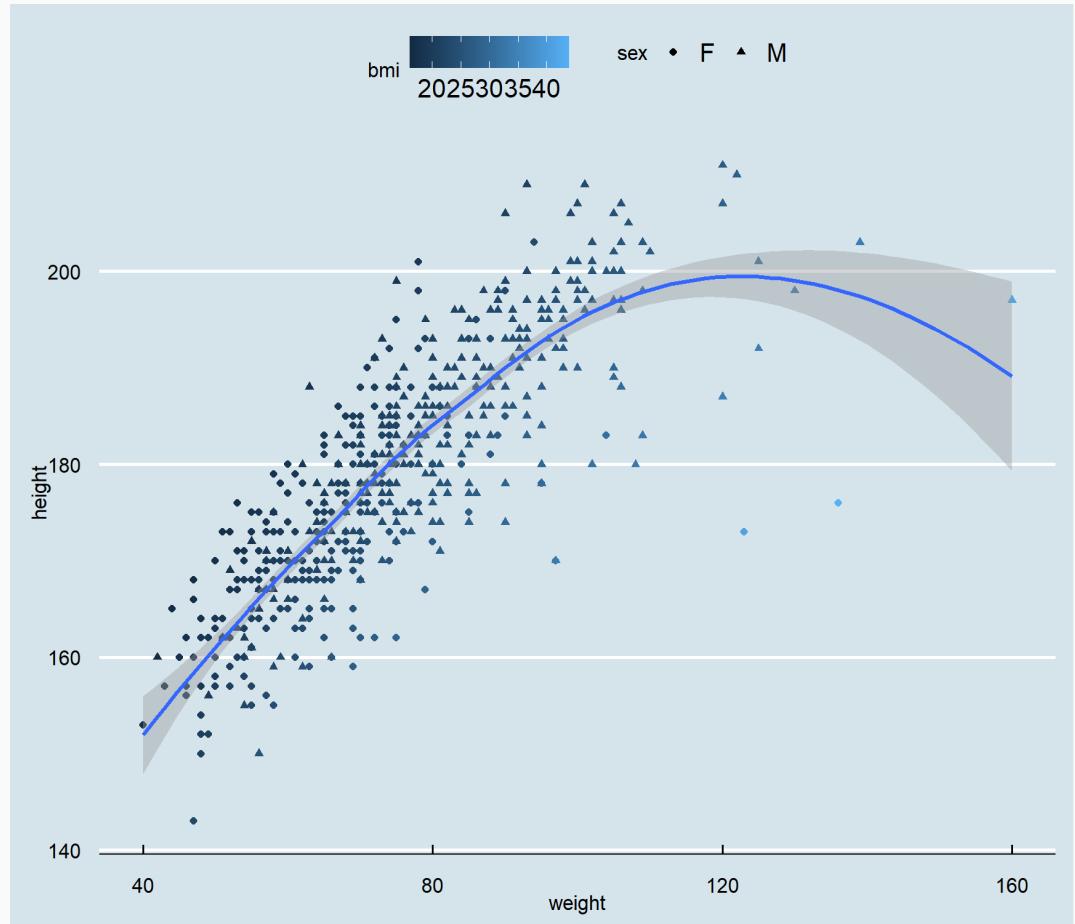
- `theme_...(...)` bietet vordefinierte Einstellungen, die das Aussehen des Plots bestimmen.
- Beispiele für vordefinierte Themes:
 - `theme_bw()`
 - **`theme_classic()`**
 - `theme_void()`
 - `theme_gray()`
- Verschiedene Pakete bieten zahlreiche weitere Themes, z. B.:
 - `ggthemes::theme_economist()`
 - `ggthemes::theme_excel_new()`



5. theme_...(...)

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  ggthemes::theme_economist()
```

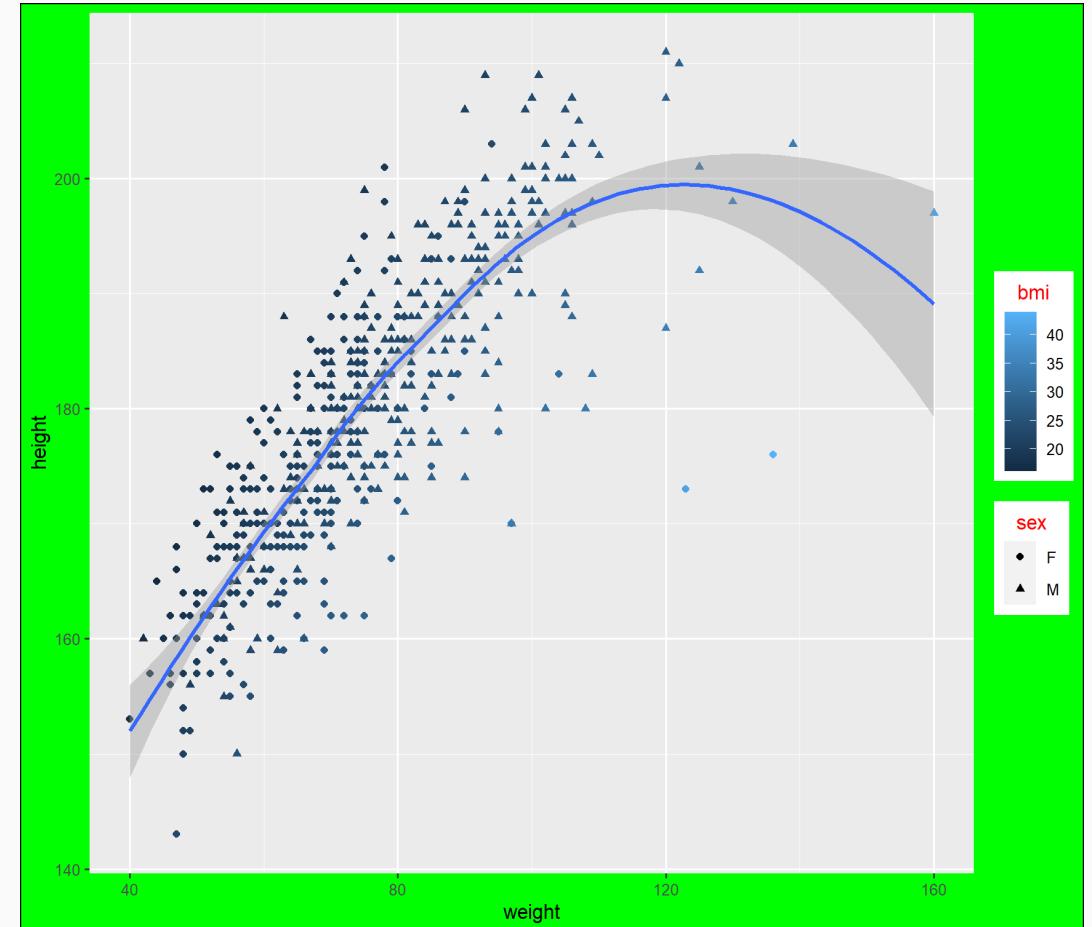
- `theme_...(...)` bietet vordefinierte Einstellungen, die das Aussehen des Plots bestimmen.
- Beispiele für vordefinierte Themes:
 - `theme_bw()`
 - `theme_classic()`
 - `theme_void()`
 - `theme_gray()`
- Verschiedene Pakete bieten zahlreiche weitere Themes, z. B.:
 - `ggthemes::theme_economist()`
 - `ggthemes::theme_excel_new()`



5. theme_... (...) und theme (...)

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  theme(plot.background =  
        element_rect(fill = "green"),  
        legend.title =  
        element_text(color = "red"))
```

- `theme(...)` bietet Anpassungsmöglichkeiten für **alle** Teile des Plots.
- Die [Liste an Optionen](#) ist entsprechend lang.
- Falls keine konkreten Layoutvorgaben für Plots gemacht werden, genügt in der Regel eines der Standard-Themes.



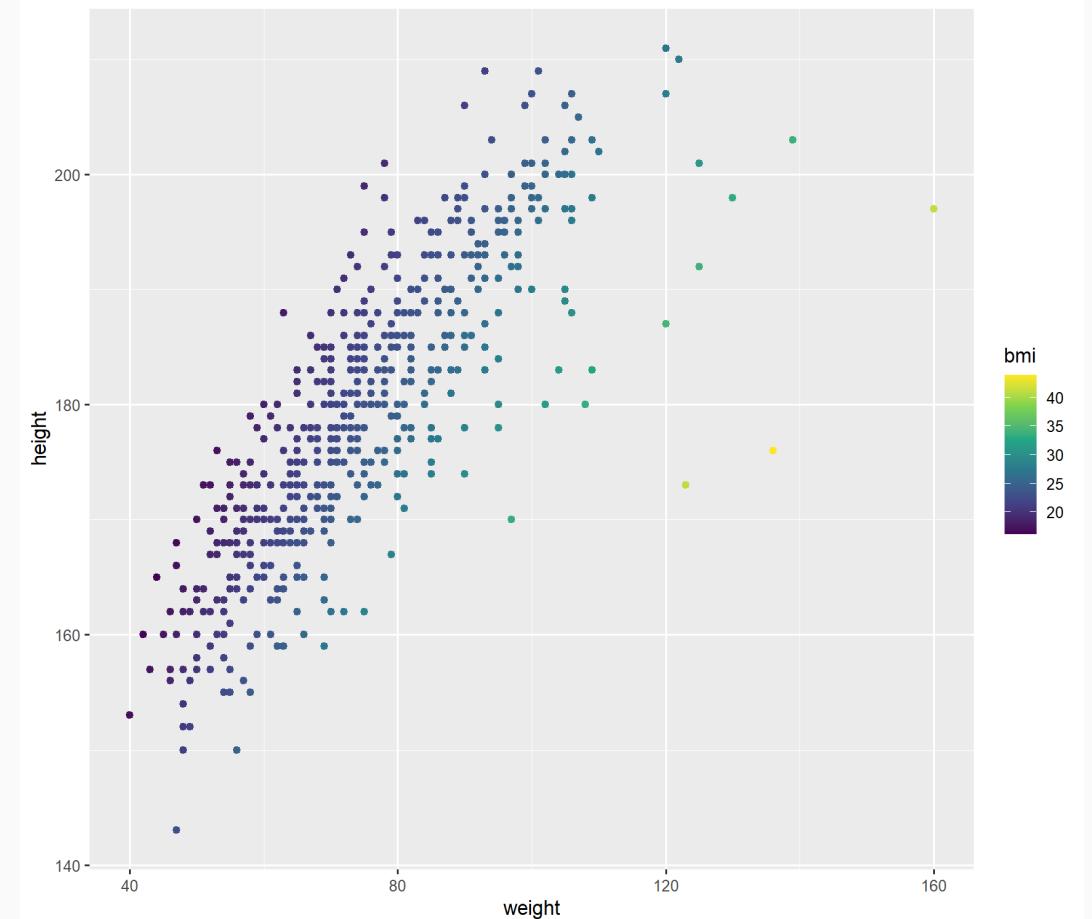
6. scale_...()

- Die Gruppe der `scale_...()`-Funktionen bietet Möglichkeiten, alle Aesthetics anzupassen.
- Das heißt, es gibt `scale_...()` Funktionen für alle Aesthetics:
 - `scale_x_...()`, `scale_y_...()` → Achsen
 - `scale_color_...()`, `scale_fill_...()` → Farben
 - `scale_alpha_...()` → Transparenz
 - `scale_size_...()` → Größe
 - `scale_shape_...()` → Form
 - `scale_linetype_...()` → Linientyp

6. scale_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi)) +  
  scale_color_viridis_c()
```

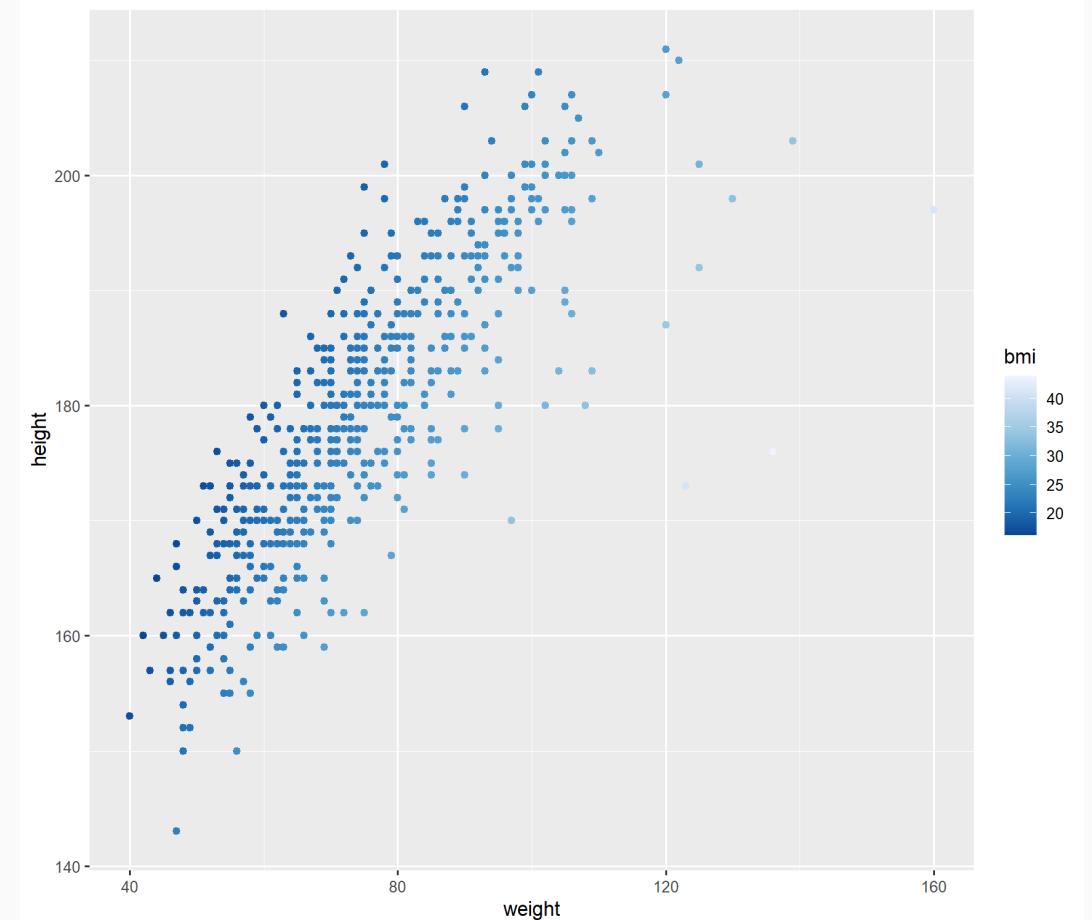
- Bezuglich der Farb-Aesthetics gibt es zahlreiche vorgefertigte Farbschemen, z. B.:
 - `scale_color_viridis_c()` /
`scale_color_viridis_d()`
 - `scale_color_distiller()`
- Und Hilfsfunktionen, um die Farben selbst festzulegen, z. B.:
 - `scale_color_gradient(...)`



6. scale_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi)) +  
  scale_color_distiller()
```

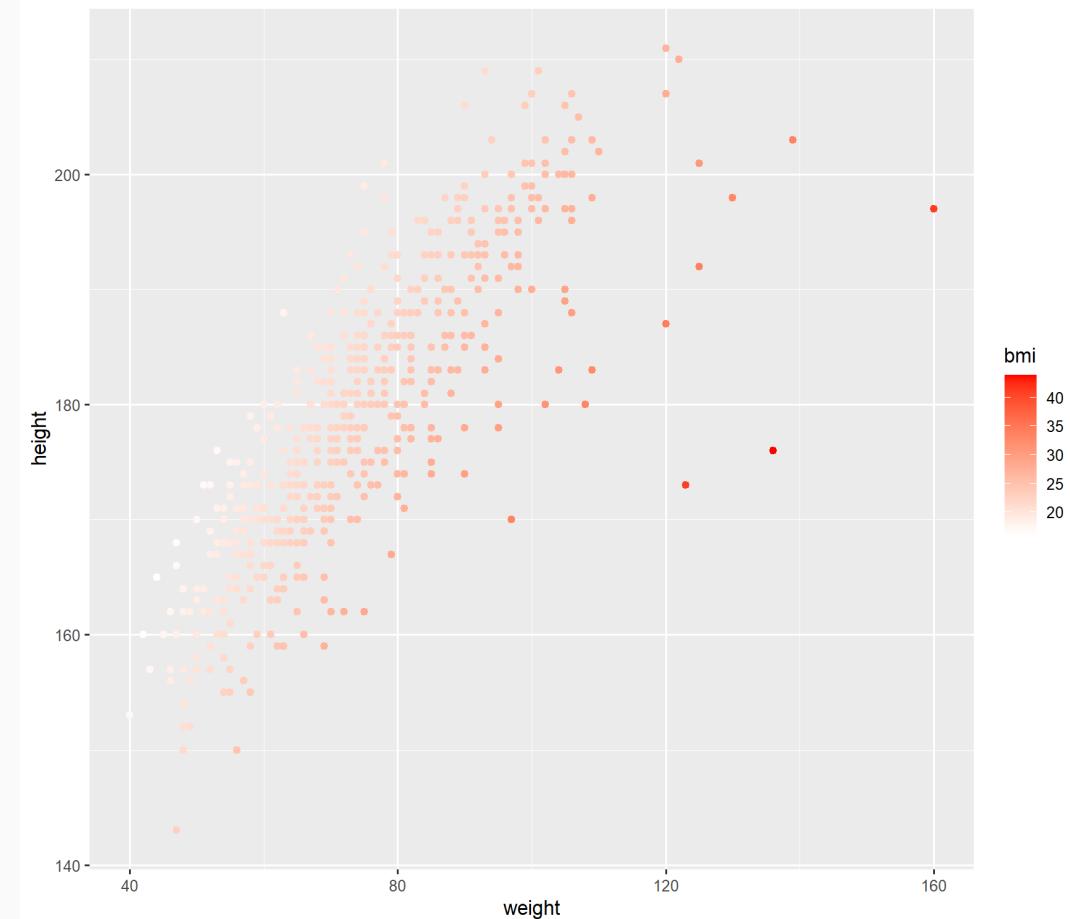
- Bezuglich der Farb-Aesthetics gibt es zahlreiche vorgefertigte Farbschemen, z. B.:
 - `scale_color_viridis_c()` /
`scale_color_viridis_d()`
 - **`scale_color_distiller()`**
- Und Hilfsfunktionen, um die Farben selbst festzulegen, z. B.:
 - `scale_color_gradient(...)`



6. scale_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi)) +  
  scale_color_gradient(low = "white",  
                       high = "red")
```

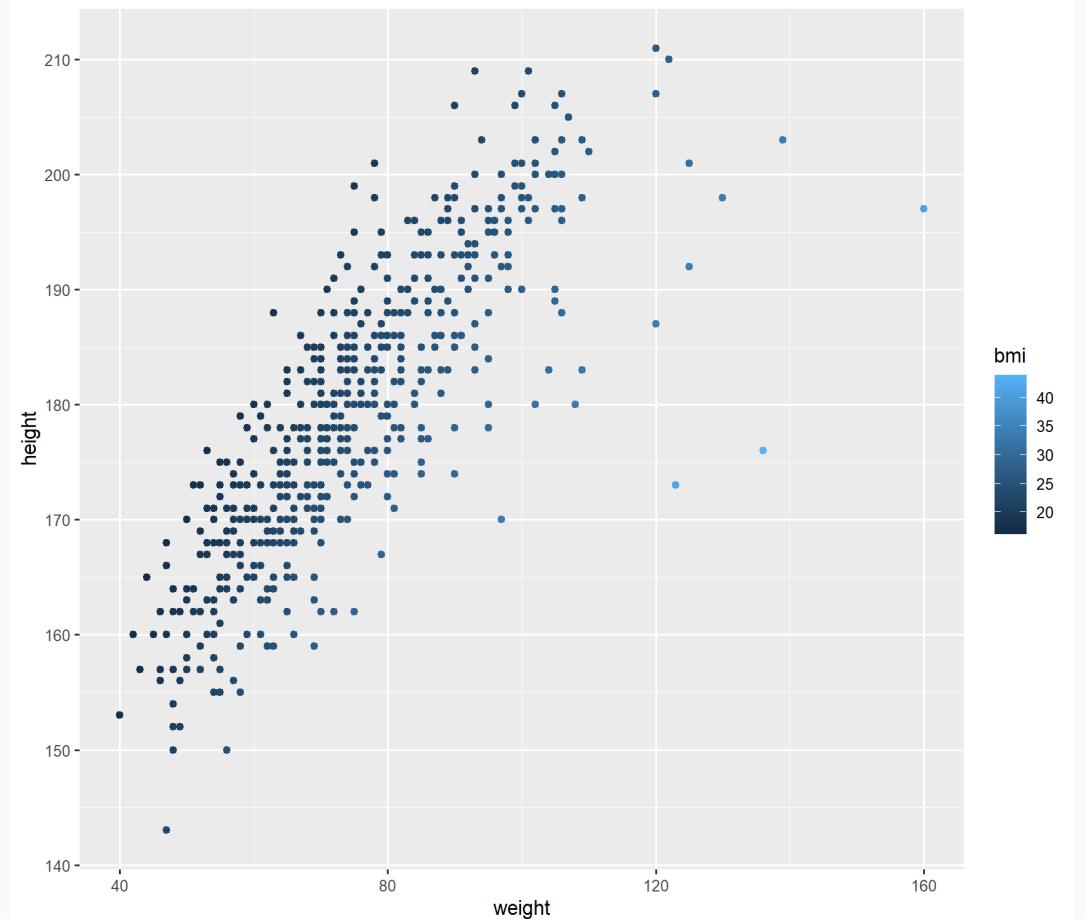
- Bezuglich der Farb-Aesthetics gibt es zahlreiche vorgefertigte Farbschemen, z. B.:
 - `scale_color_viridis_c()` / `scale_color_viridis_d()`
 - `scale_color_distiller()`
- Und Hilfsfunktionen, um die Farben selbst festzulegen, z. B.:
 - `scale_color_gradient(...)`



6. scale_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi)) +  
  scale_y_continuous(breaks =  
                     c(seq(from = 140,  
                         to = 220,  
                         by = 10)))
```

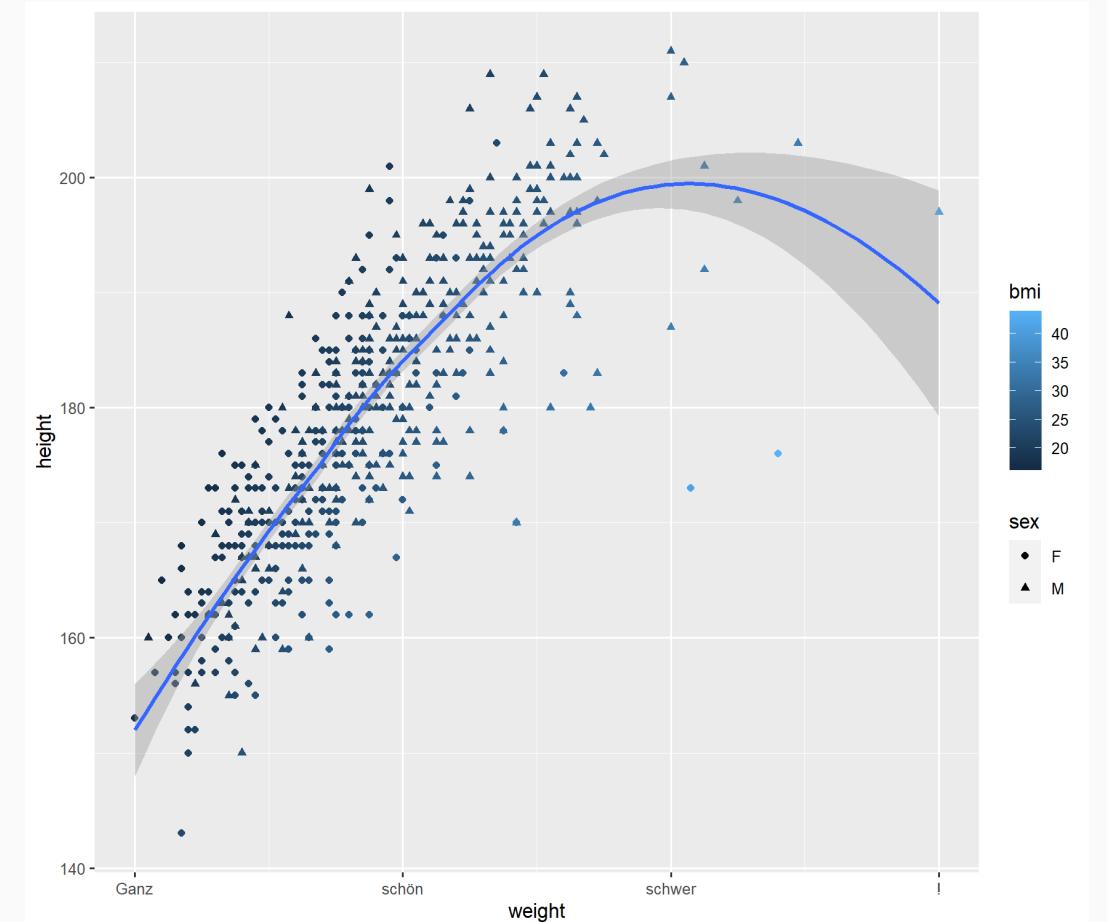
- Mit der `break`-Option in der `scale`-Achsen-Funktion lassen sich beispielsweise die Beschriftungen anpassen.



6. scale_...()

```
ggplot(data = olympics_daten_plot,  
       mapping = aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi, shape = sex)) +  
  geom_smooth() +  
  scale_x_continuous(breaks =  
                     c(40, 80, 120, 160),  
                     labels =  
                     c("Ganz", "schön", "schwer", "!" ))
```

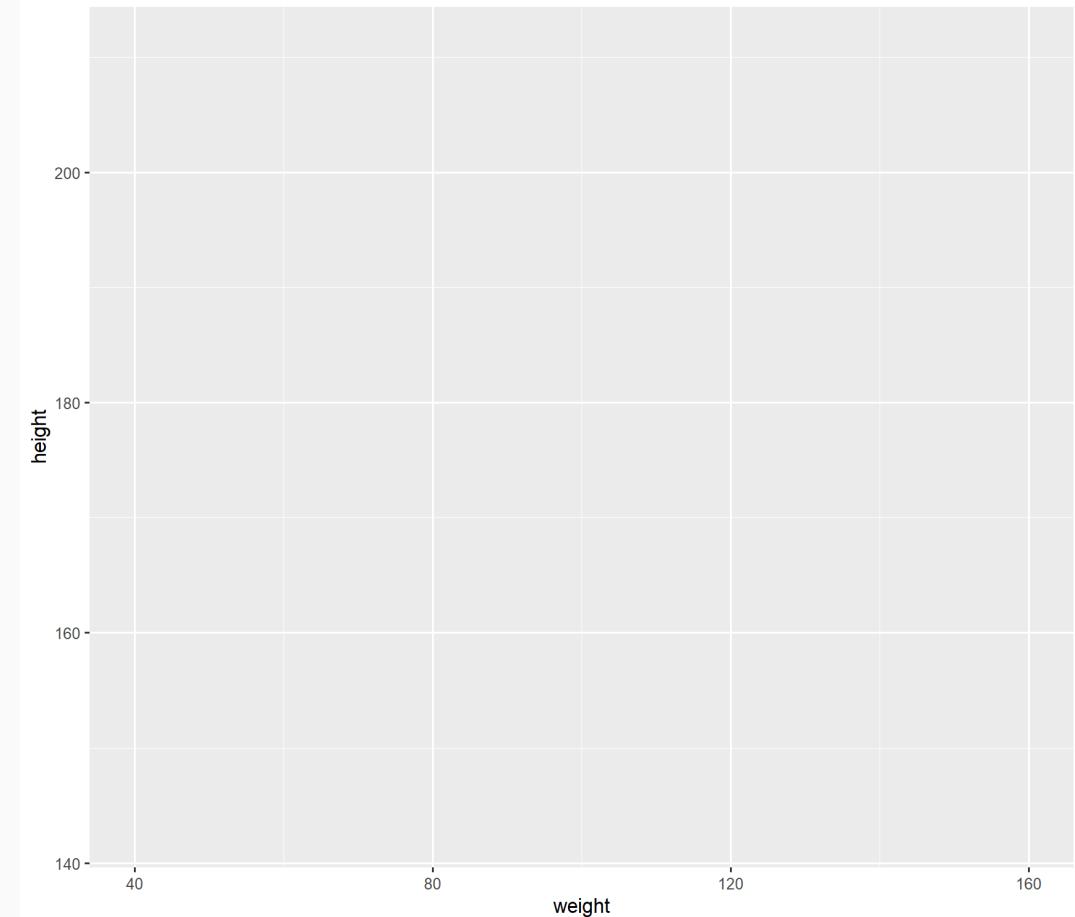
- Mit dem `break`-Argument in der `scale`-Achsen-Funktion lassen sich beispielsweise die Beschriftungen anpassen.
- In Kombination mit dem `labels`-Argument können Achsenlabels verändert werden, ohne die Daten zu verändern.



»Putting it all together«

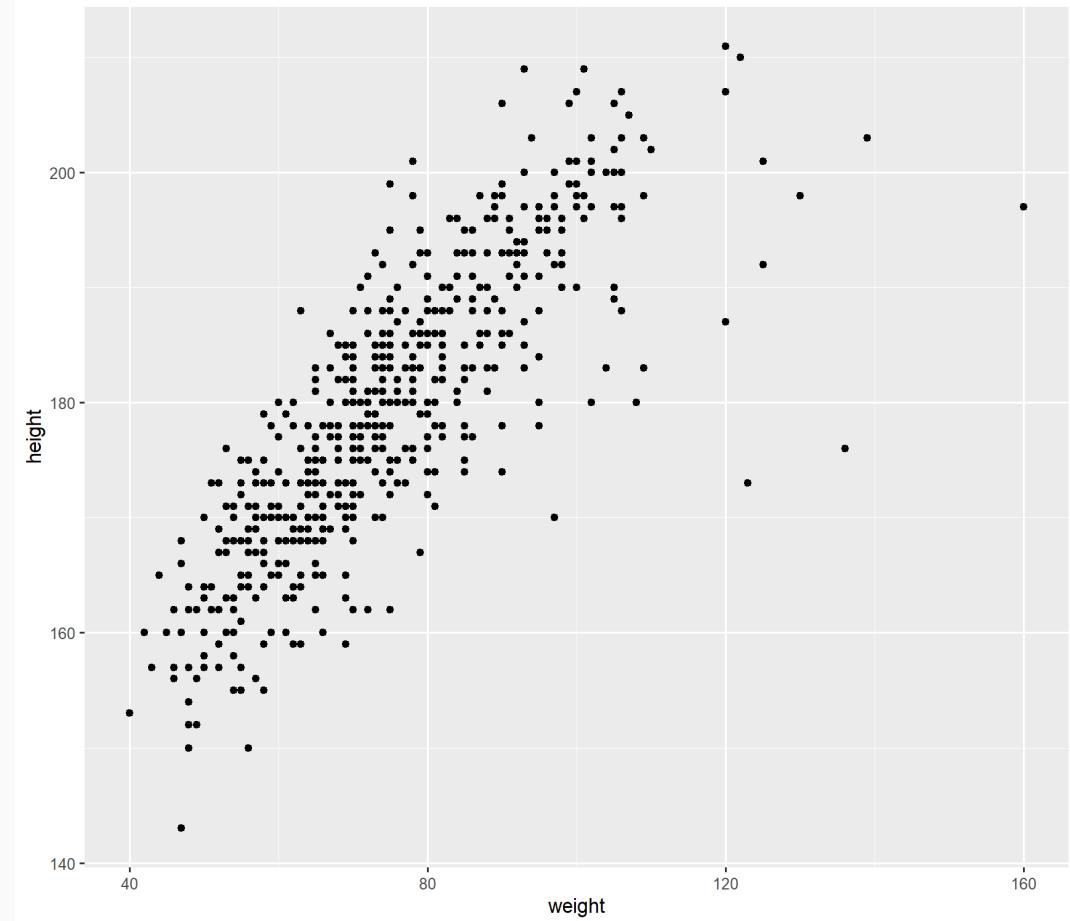
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height))
```



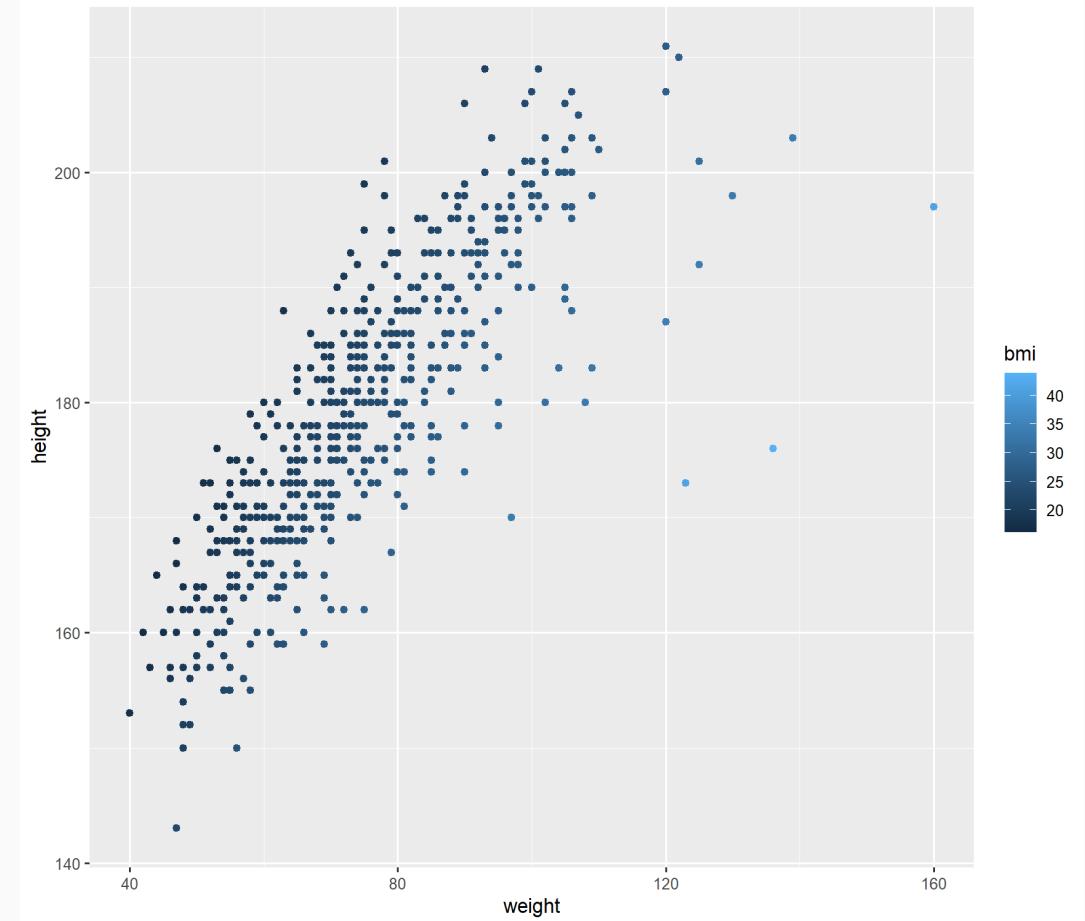
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point()
```



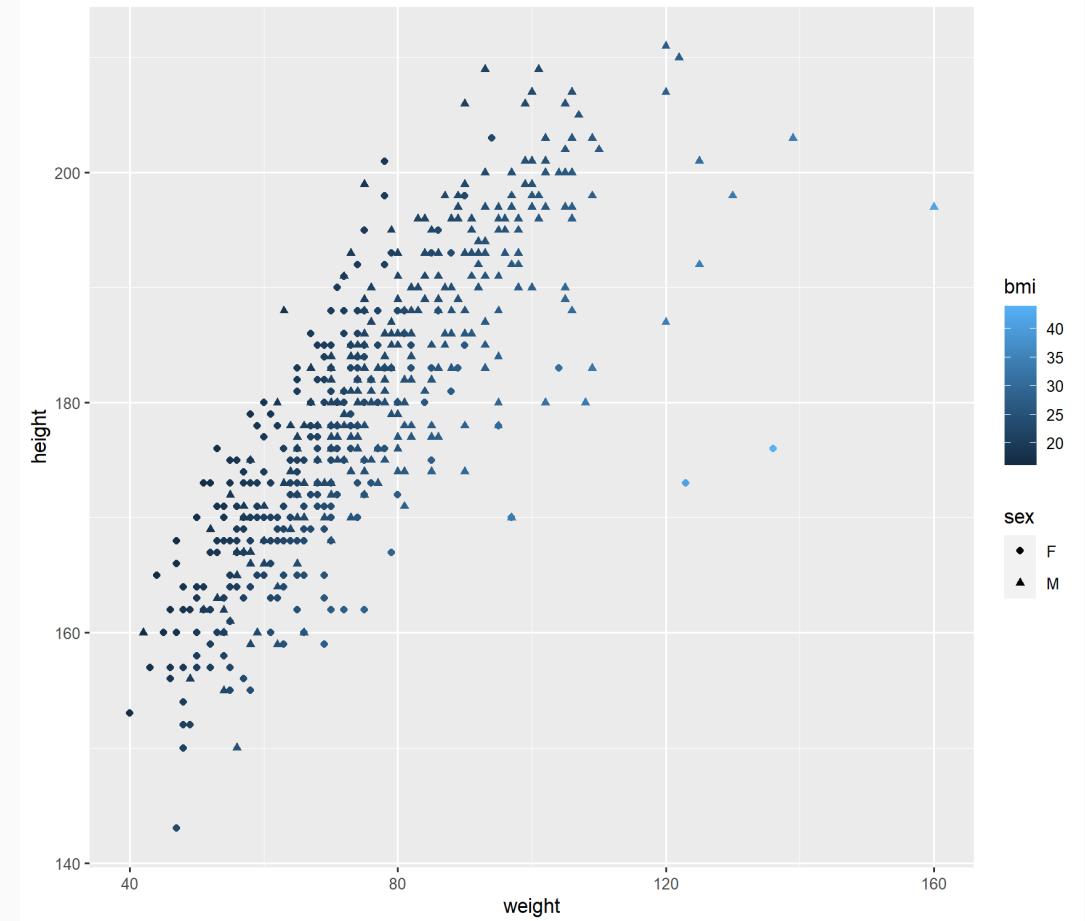
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi))
```



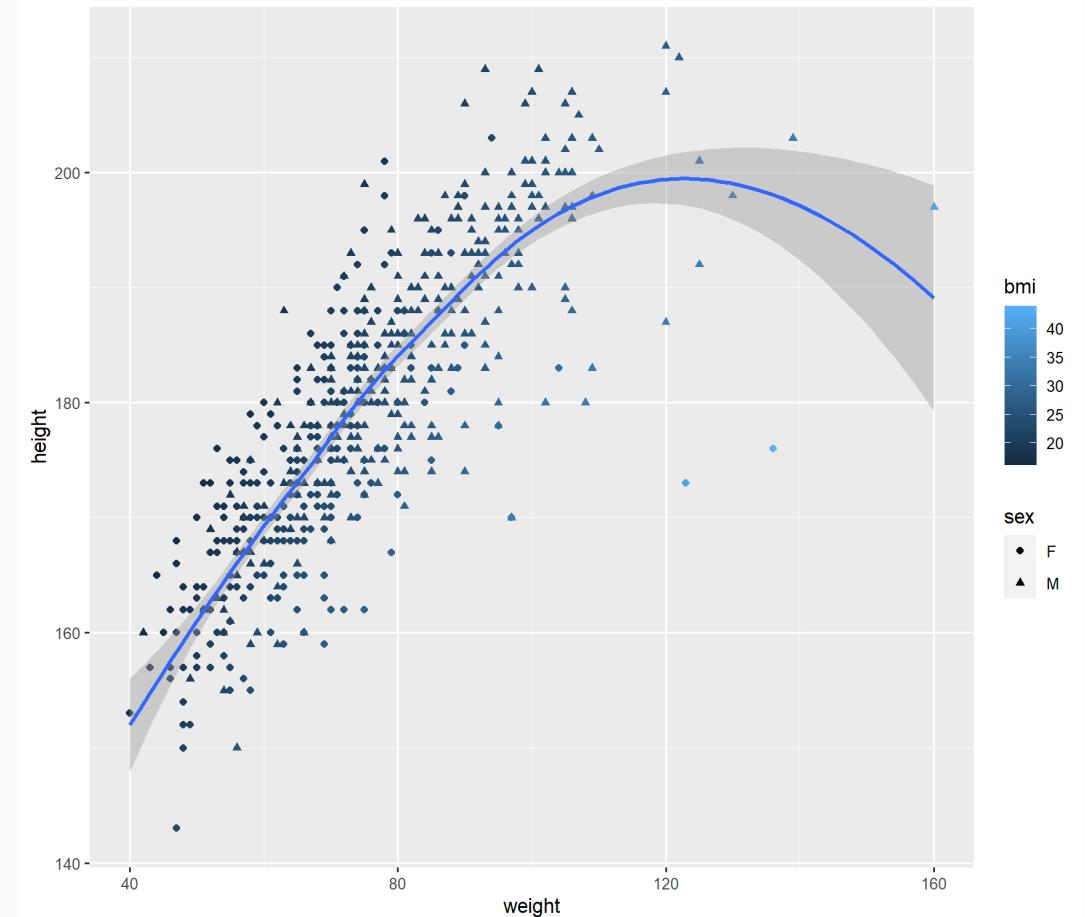
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi,  
                 shape = sex))
```



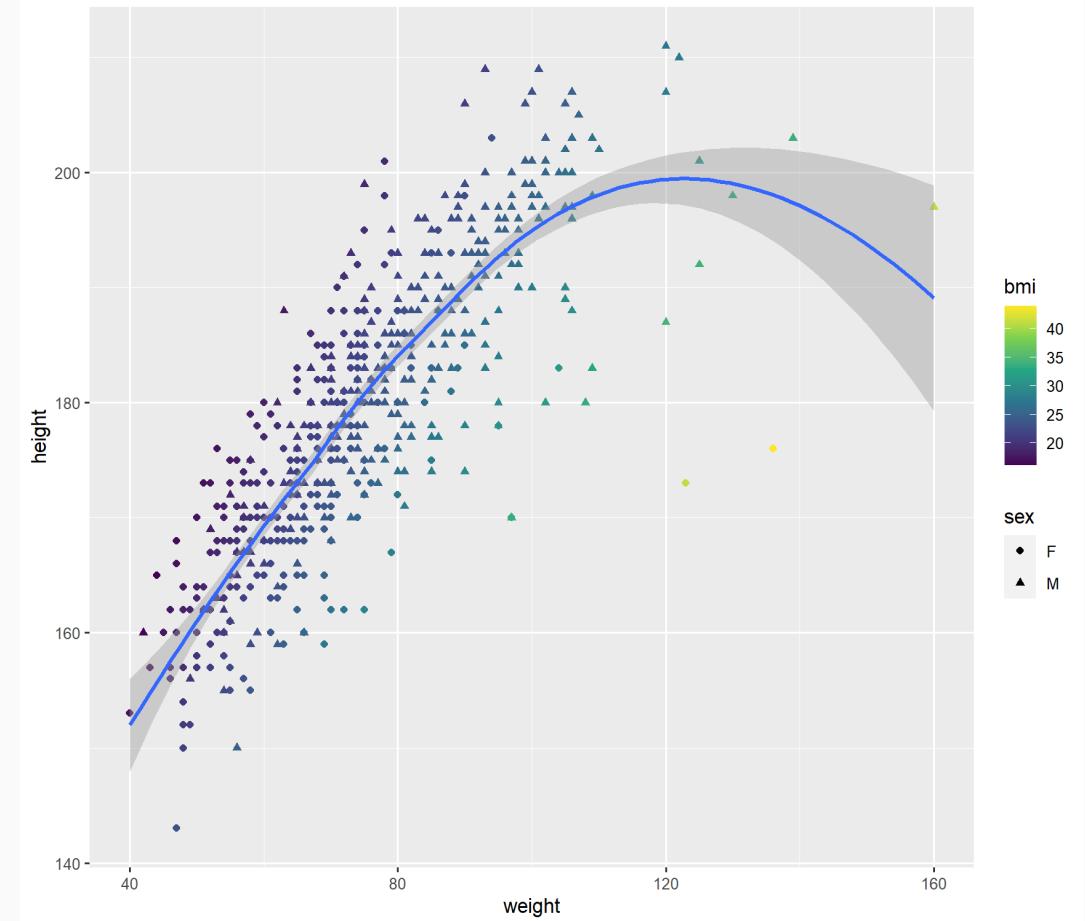
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi,  
                 shape = sex)) +  
  geom_smooth()
```



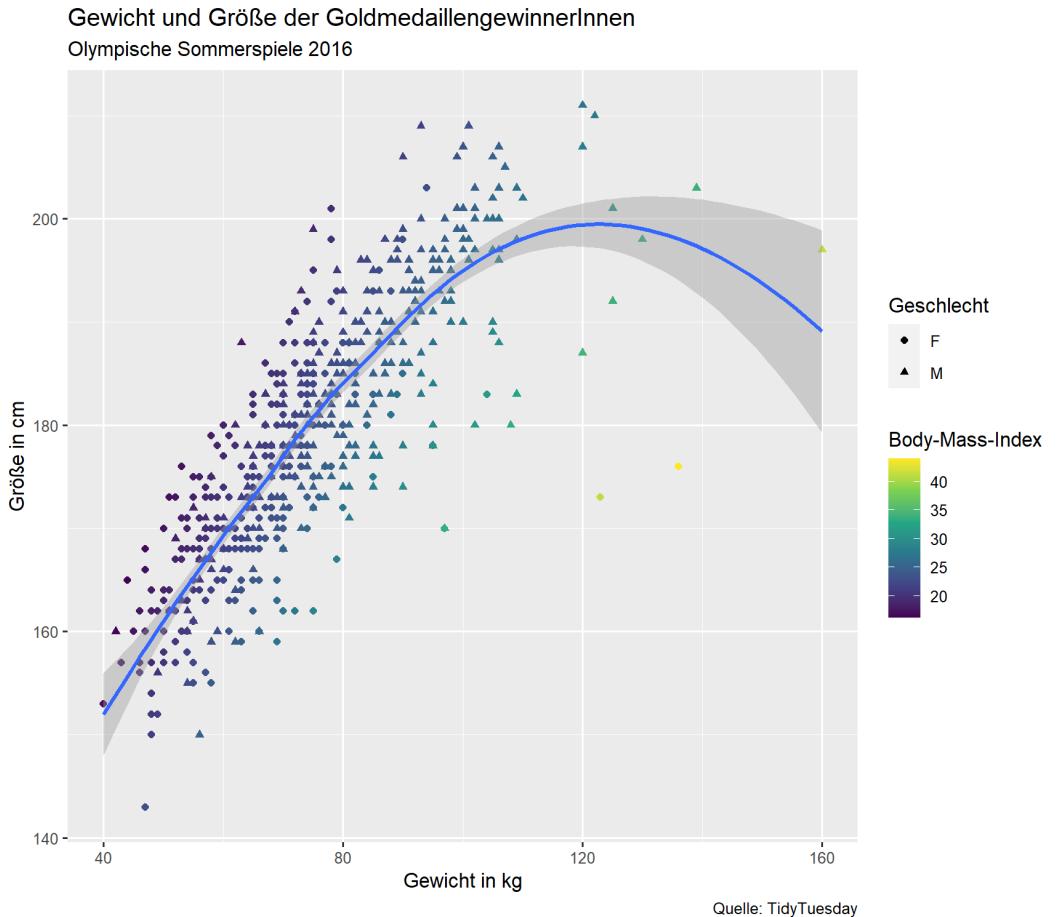
»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi,  
                 shape = sex)) +  
  geom_smooth() +  
  scale_color_viridis_c()
```



»Putting it all together«

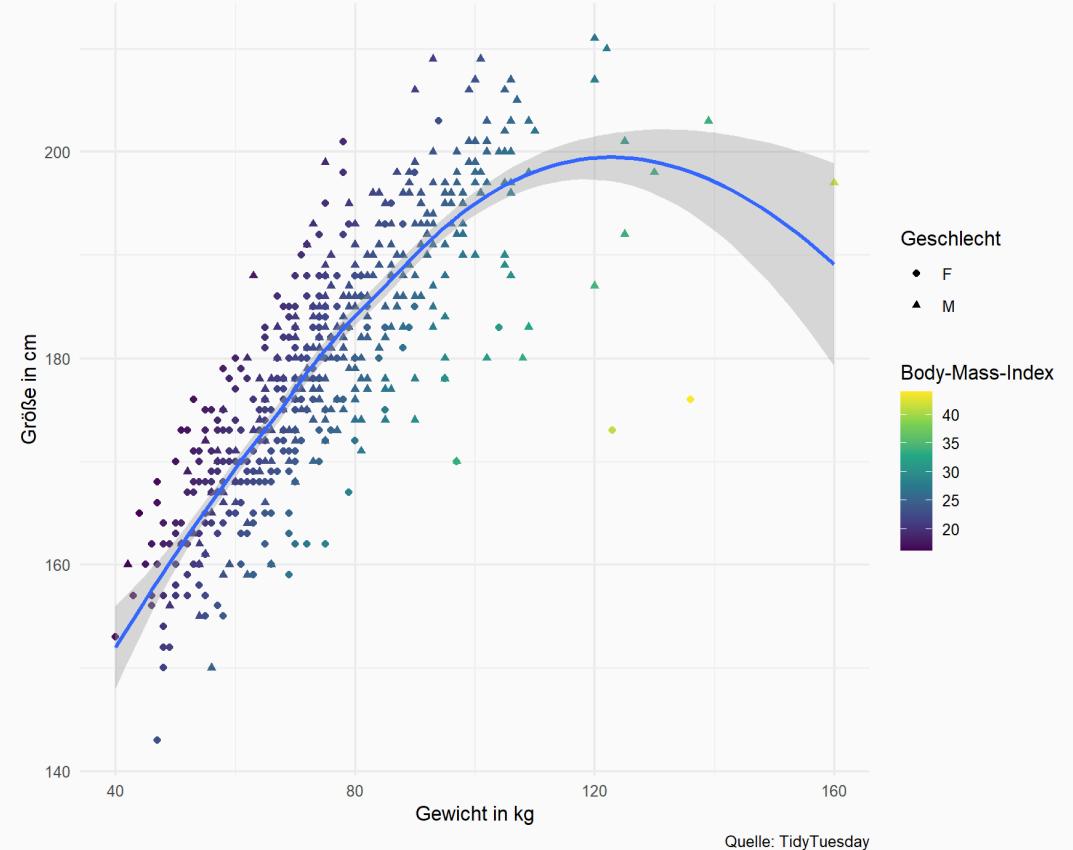
```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi,  
                 shape = sex)) +  
  geom_smooth() +  
  scale_color_viridis_c() +  
  labs(  
    title = "Gewicht und Größe der GoldmedaillengewinnerInnen",  
    subtitle = "Olympische Sommerspiele 2016",  
    x = "Gewicht in kg",  
    y = "Größe in cm",  
    color = "Body-Mass-Index",  
    shape = "Geschlecht",  
    caption = "Quelle: TidyTuesday")
```



»Putting it all together«

```
olympics_daten_plot %>%  
  ggplot(aes(x = weight, y = height)) +  
  geom_point(aes(col = bmi,  
                 shape = sex)) +  
  geom_smooth() +  
  scale_color_viridis_c() +  
  labs(  
    title = "Gewicht und Größe der GoldmedaillengewinnerInnen",  
    subtitle = "Olympische Sommerspiele 2016",  
    x = "Gewicht in kg",  
    y = "Größe in cm",  
    color = "Body-Mass-Index",  
    shape = "Geschlecht",  
    caption = "Quelle: TidyTuesday") +  
  theme_minimal()
```

Gewicht und Größe der GoldmedaillengewinnerInnen
Olympische Sommerspiele 2016



Genug der Theorie. Ab nach

