2. Farben, Formen und Paletten

Statische Aesthetics, aes()-Mapping und Farbpaletten fuer jeden Anlass

Dr. Paul Schmidt

```
for (pkg in c("ggplot2", "scales", "RColorBrewer", "MetBrewer", "viridis")) {
  if (!require(pkg, character.only = TRUE)) install.packages(pkg)
}
```

Farbe ist eines der maechtigsten Werkzeuge in der Datenvisualisierung. Ein gut gewaehltes Farbschema kann Gruppenunterschiede hervorheben, die Aufmerksamkeit des Lesers lenken und einen Plot sofort zugaenglich machen. Ein schlecht gewaehltes kann Muster verschleiern, Leser verwirren oder Menschen mit Farbfehlsichtigkeit ausschliessen.

In ggplot2 kann man visuelle Eigenschaften wie Farbe, Form, Groesse und Transparenz auf zwei grundsaetzlich verschiedene Arten festlegen: als **statische Werte** (gleich fuer alle Datenpunkte) oder **dynamisch gemappt** auf Variablen in den Daten ueber `aes()`. Diese Unterscheidung zu verstehen ist wesentlich - sie bestimmt, ob eine visuelle Eigenschaft rein kosmetisch ist oder tatsaechlich Informationen kodiert.

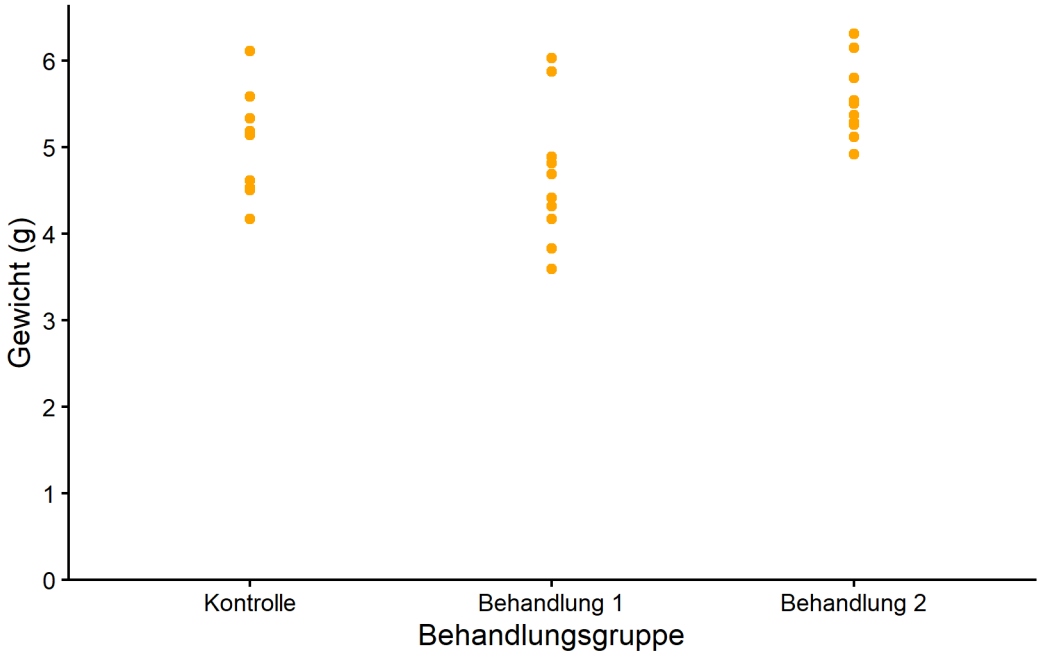
Wir arbeiten weiter mit dem verfeinerten Plot aus dem vorherigen Kapitel:

```
myplot <- ggplot(data = PlantGrowth) +
  aes(y = weight, x = group) +
  scale_y_continuous(
    name = "Gewicht (g)",
    limits = c(0, NA),
    breaks = seq(0, 6),
    expand = expansion(mult = c(0, 0.05))
  ) +
  scale_x_discrete(
    name = "Behandlungsgruppe",
    labels = c(
      ctrl = "Kontrolle",
      trt1 = "Behandlung 1",
      trt2 = "Behandlung 2"
    )
  ) +
  theme_classic()
```

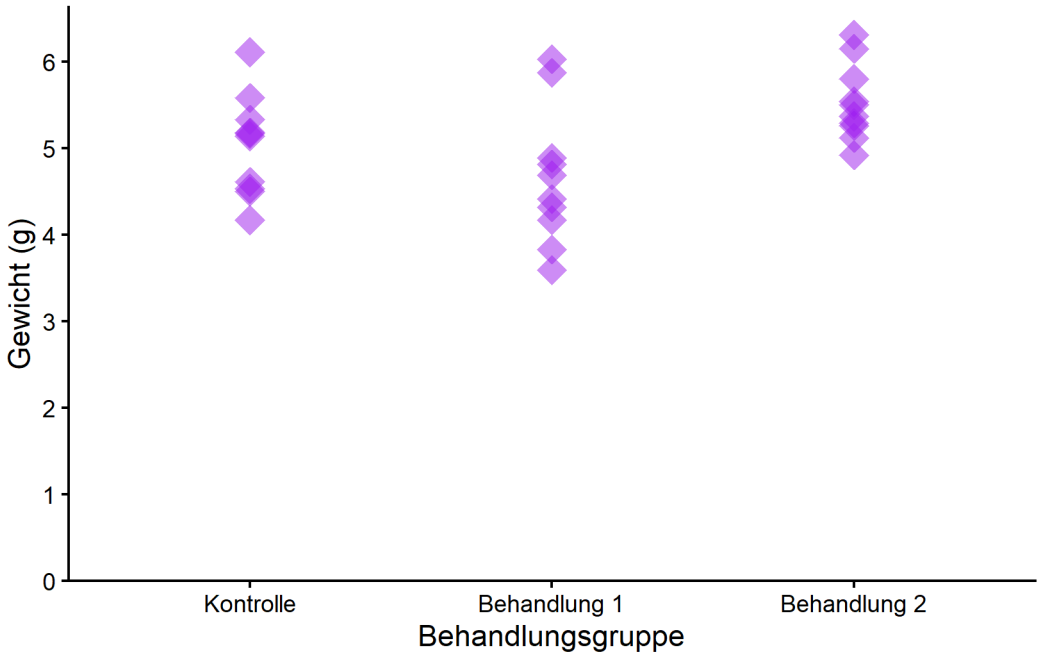
Statische Aesthetics

Wenn man aesthetische Eigenschaften direkt in einer `geom_*()`-Funktion setzt - aber *ausserhalb* von `aes()` - gelten sie einheitlich fuer alle Datenpunkte. Das ist der einfachere der beiden Ansaetze: Man waehlt eine feste Farbe, Form oder Groesse, und jeder Punkt sieht gleich aus. Das ist nuetzlich, wenn die Gruppierung bereits ueber die Achse vermittelt wird (wie in unserem Plot, wo die Gruppen entlang der x-Achse getrennt sind) und zusaetzliche Farbkodierung redundant waere.

```
myplot +
  geom_point(color = "orange")
```




```
myplot +
  geom_point(
    color = "purple",
    shape = 18,
    size = 5,
    alpha = 0.5
  )
```



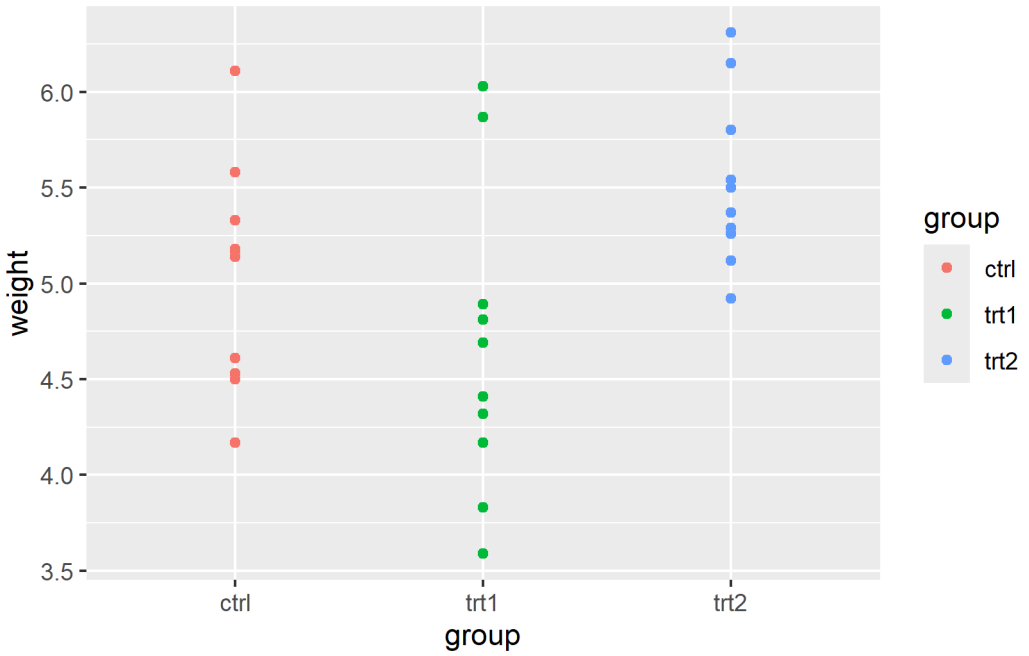
Im ersten Plot sind alle Punkte orange. Im zweiten werden mehrere Eigenschaften gleichzeitig modifiziert: lila Farbe, Diamantform (18), groessere Punkte und 50% Transparenz. Man beachte, dass diese Aenderungen rein kosmetisch sind - sie kodieren keine Informationen aus den Daten.

Aesthetic Mapping mit aes()

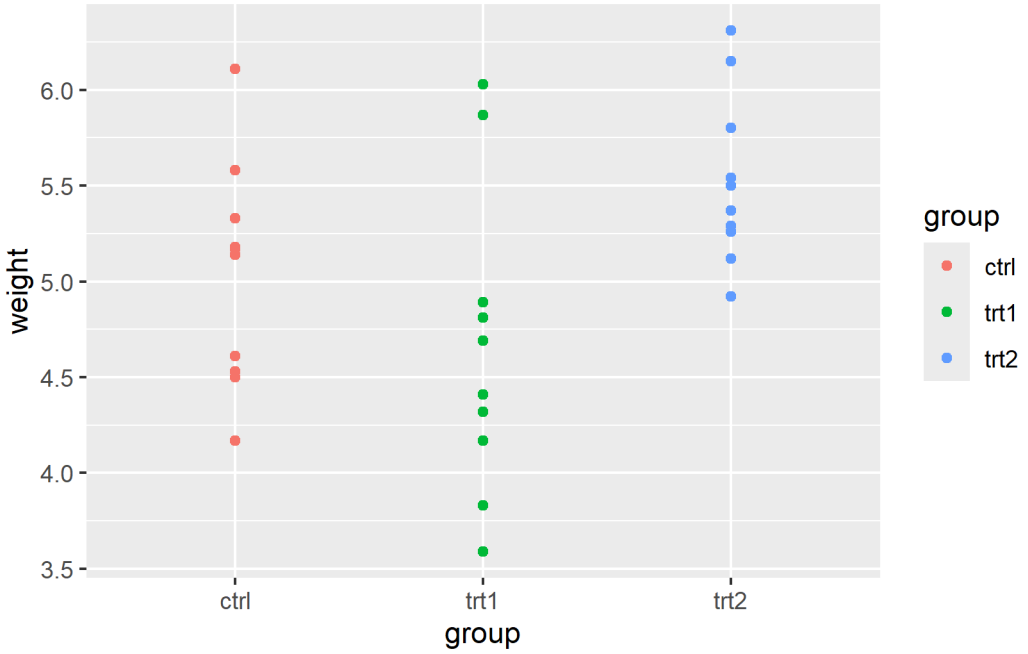
Die eigentliche Staerke von ggplot2 liegt im **dynamischen Mapping** ueber `aes()`. Anstatt eine feste Farbe fuer alle Punkte zu setzen, kann man eine Variable auf die Farb-Aesthetic mappen, sodass jede Gruppe automatisch ihre eigene Farbe erhaelt. So sagt man ggplot2: „Verwende Farbe, um die Gruppierung in meinen Daten darzustellen."

Ein wichtiges Detail: `aes()` und `data` kann man entweder global in `ggplot()` (gilt fuer alle Layer) oder lokal innerhalb einer bestimmten `geom_*()`-Funktion (gilt nur fuer diesen Layer) definieren. Fuer einfache Plots mit einem einzelnen Geom erzeugen beide dasselbe Ergebnis. Die Unterscheidung wird wichtig, wenn man mehrere Geoms mit unterschiedlichen Datenquellen kombiniert - eine Technik, die wir in spaeteren Kapiteln verwenden werden.

```
ggplot(data = PlantGrowth) +
  aes(y = weight, x = group, color = group) +
  geom_point()
```

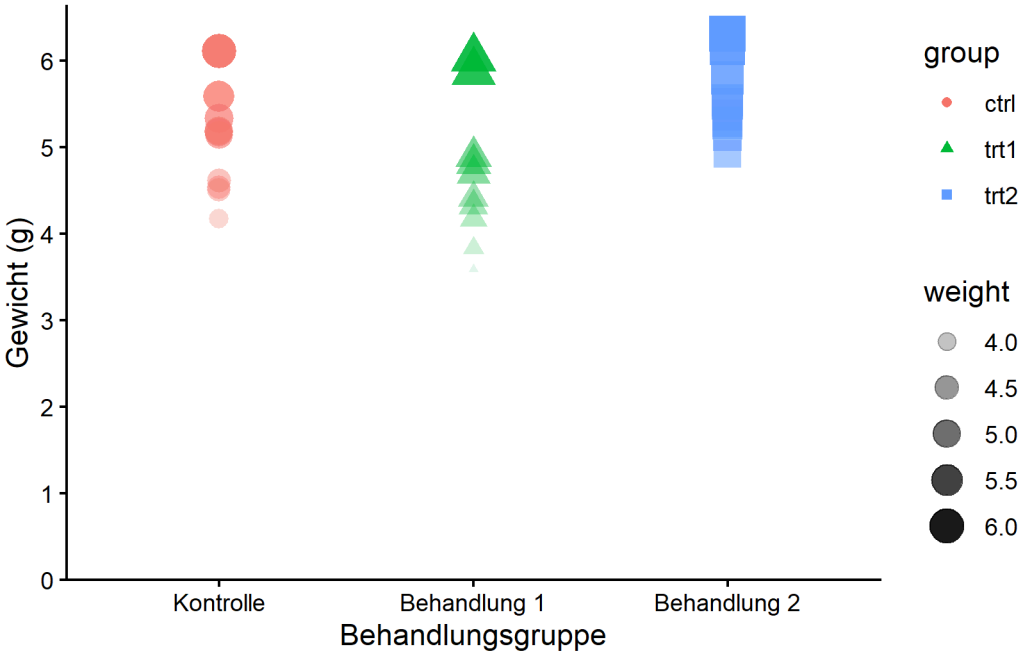



```
ggplot() +
  geom_point(
    data = PlantGrowth,
    aes(y = weight, x = group, color = group)
  )
```



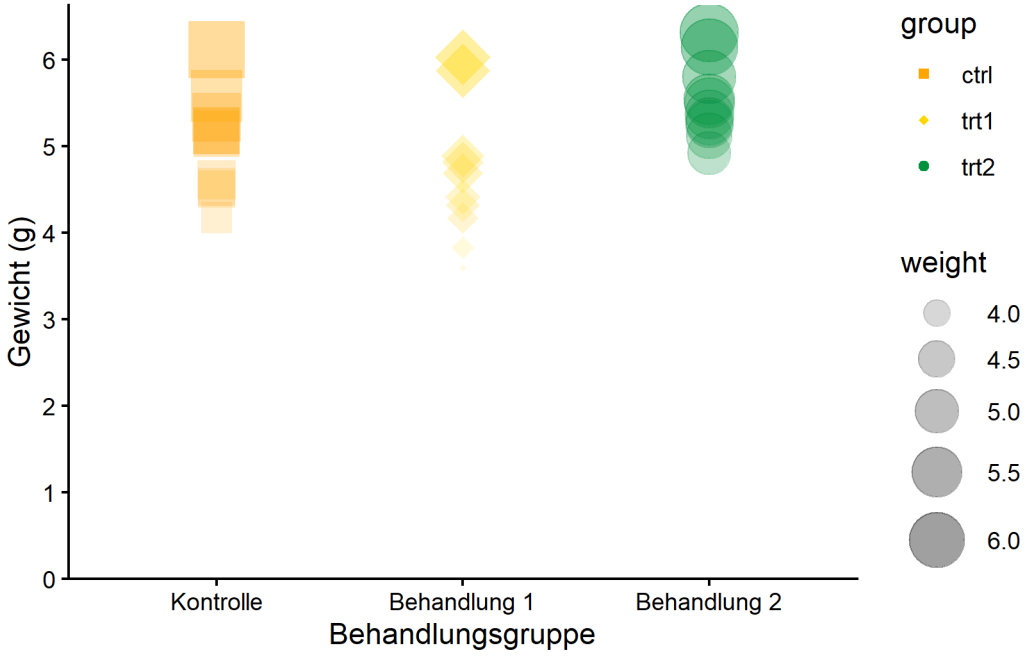
Mehrere Aesthetics koennen gleichzeitig gemappt werden. Das ist nuetzlich, wenn ein einzelner visueller Kanal nicht ausreicht - beispielsweise liefert das gleichzeitige Mappen von Farbe *und* Form auf dieselbe Variable eine redundante Kodierung, die farbenblinden Lesern hilft. Hier haengen Farbe und Form von `group` ab, waehrend Groesse und Transparenz von `weight` abhaengen:

```
myplot +
  geom_point(aes(
    color = group,
    shape = group,
    size = weight,
    alpha = weight
  ))
```



Genau wie `scale_x_*` und `scale_y_*` das Achsenaussehen steuern, gibt es Scale-Funktionen fuer jede Aesthetic. Diese bieten die vollstaendige Kontrolle ueber das Mapping:

```
myplot +
  geom_point(aes(
    color = group,
    shape = group,
    size = weight,
    alpha = weight
  )) +
  scale_color_manual(
    values = c("orange", "gold", "#00923f")
  ) +
  scale_shape_manual(
    values = c(15, 18, 19)
  ) +
  scale_size_continuous(
    range = c(1, 10)
  ) +
  scale_alpha_continuous(
    range = c(0.1, 0.4)
  )
```



Farbpaletten

Einzelne Farben manuell aus langen Listen auszuwaehlen ist muehsam und fuehrt oft zu Kombinationen, die visuell kollidieren oder schwer zu unterscheiden sind. Ein besserer Ansatz sind kuratierte Farbpaletten - vordefinierte Farbsets, die so gestaltet sind, dass sie harmonisch, unterscheidbar und oft auch fuer Menschen mit Farbfehlsichtigkeit zugaenglich sind.

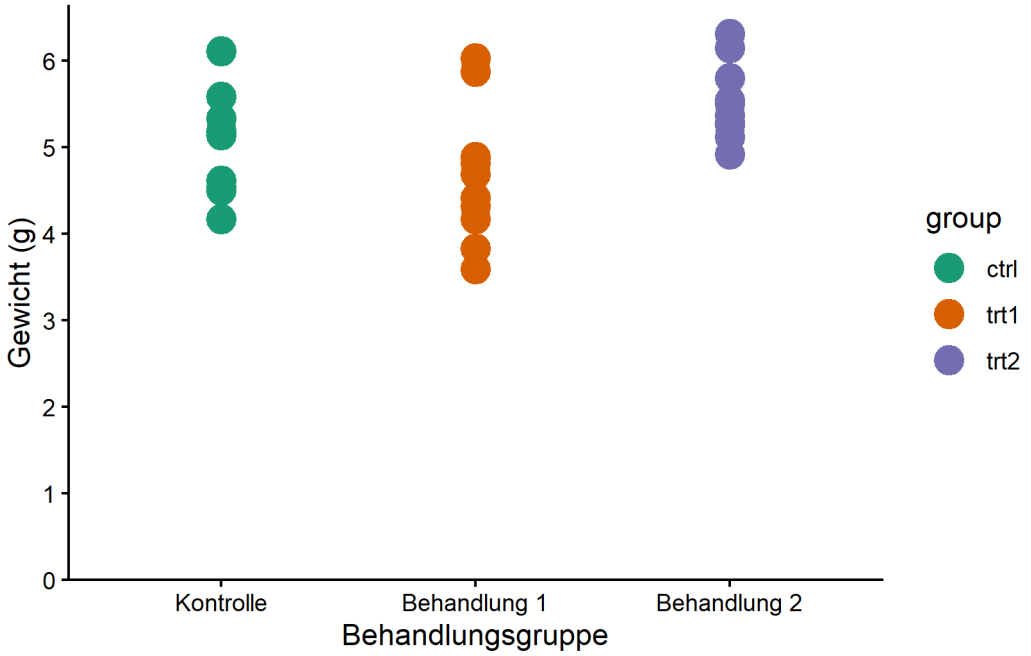
Es gibt zwei generelle Ansaetze: Man kann einen Farbvektor aus einer Palette erzeugen und an `scale_color_manual()` uebergeben, oder man verwendet eine dedizierte `scale_color_*()`-Funktion, die die Palette direkt anwendet. Beide Varianten werden unten gezeigt.

RColorBrewer

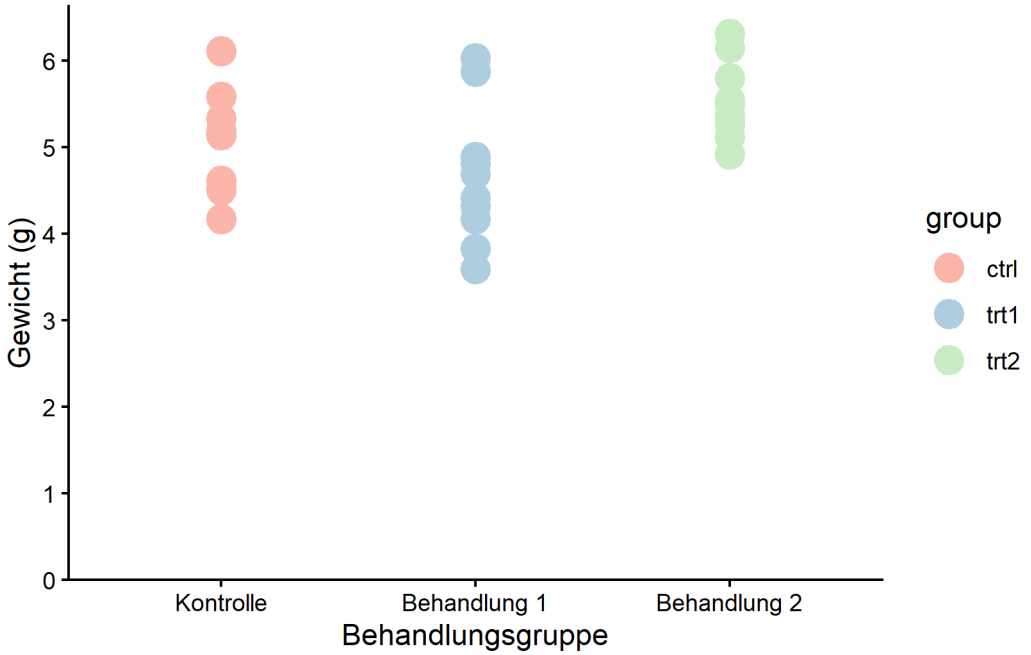
Das {RColorBrewer}-Paket bietet eine bewaehrte Sammlung von Paletten, die urspruenglich fuer Kartografie entworfen, aber in allen Fachbereichen weit verbreitet sind. Es stellt drei Palettentypen bereit: *sequentiell* (fuer geordnete Daten), *divergierend* (fuer Daten mit einem aussagekraeftigen Mittelpunkt) und *qualitativ* (fuer kategorische Gruppen wie unsere).

```
color_vector <- RColorBrewer::brewer.pal(3, "Dark2")

myplot +
  geom_point(aes(color = group), size = 5) +
  scale_color_manual(values = color_vector)
```




```
myplot +
  geom_point(aes(color = group), size = 5) +
  scale_color_brewer(palette = "Pastel1")
```

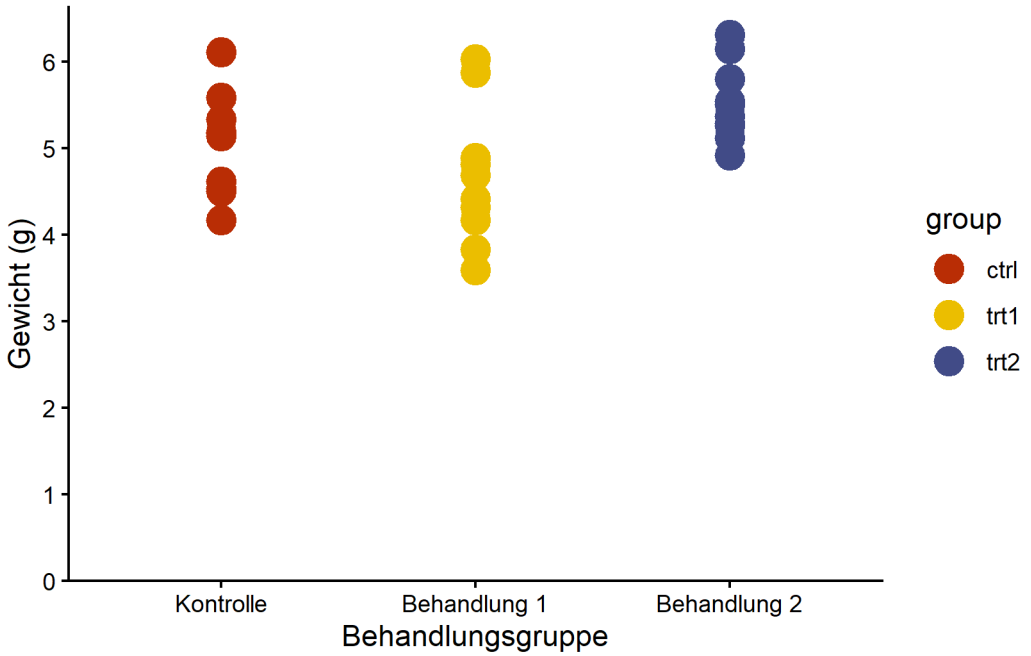


MetBrewer

Das {MetBrewer}-Paket bietet Paletten, die von Kunstwerken des Metropolitan Museum of Art inspiriert sind.

```
color_vector <- MetBrewer::met.brewer("VanGogh2", 3)

myplot +
  geom_point(aes(color = group), size = 5) +
  scale_color_manual(values = color_vector)
```



Farbenblind-freundlich: viridis

Wenn man Plots fuer Publikation oder Praesentation erstellt, sollte Barrierefreiheit bei Farben kein Nachgedanke sein. Etwa 8% der Maenner und 0,5% der Frauen haben eine Form von Farbfehlsichtigkeit. Die viridis-Palettenfamilie ist speziell so gestaltet, dass sie perzeptuell

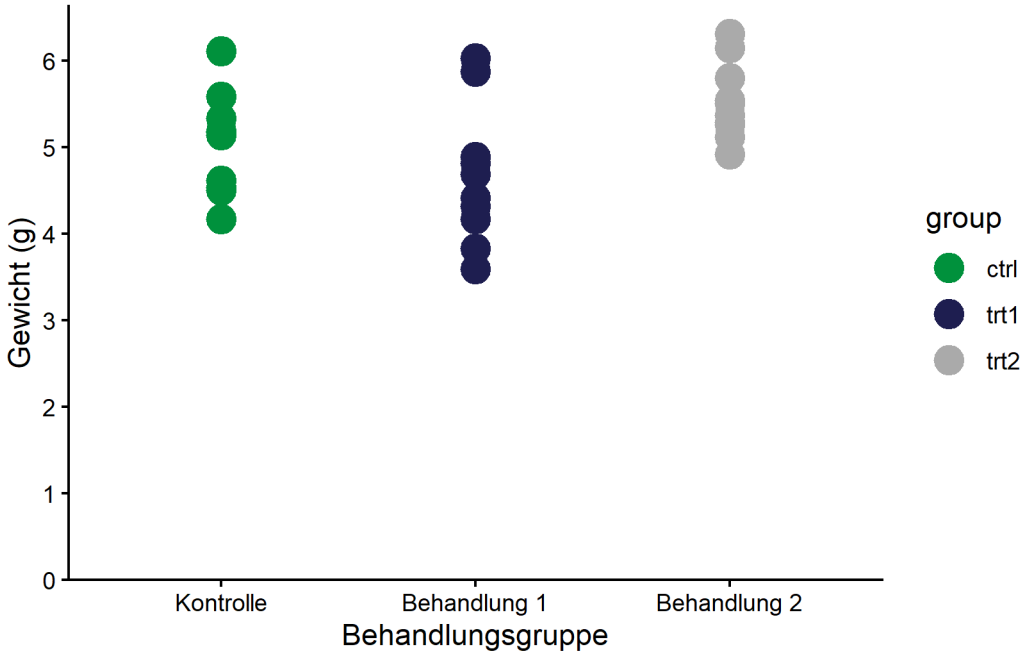
gleichmaessig (gleiche Schritte in den Daten erzeugen gleiche visuelle Schritte) und fuer die haeufigsten Formen von Farbenblindheit zugaenglich ist.

```
myplot +
  geom_point(aes(color = group), size = 5) +
  scale_color_viridis_d()
```

Corporate Design

In der Praxis ist die Farbwahl oft gar nicht frei - viele Organisationen haben einen Corporate-Design-Leitfaden, der exakte Hex-Farben fuer alle Grafiken vorschreibt. In solchen Faellen definiert man einfach die vorgegebenen Farben und uebergibt sie ueber `scale_color_manual()`. Das Corporate-Gruen von BioMath ist beispielsweise `#00923f`:

```
myplot +
  geom_point(aes(color = group), size = 5) +
  scale_color_manual(values = c("#00923f", "#201E50", "#AAAAAA"))
```

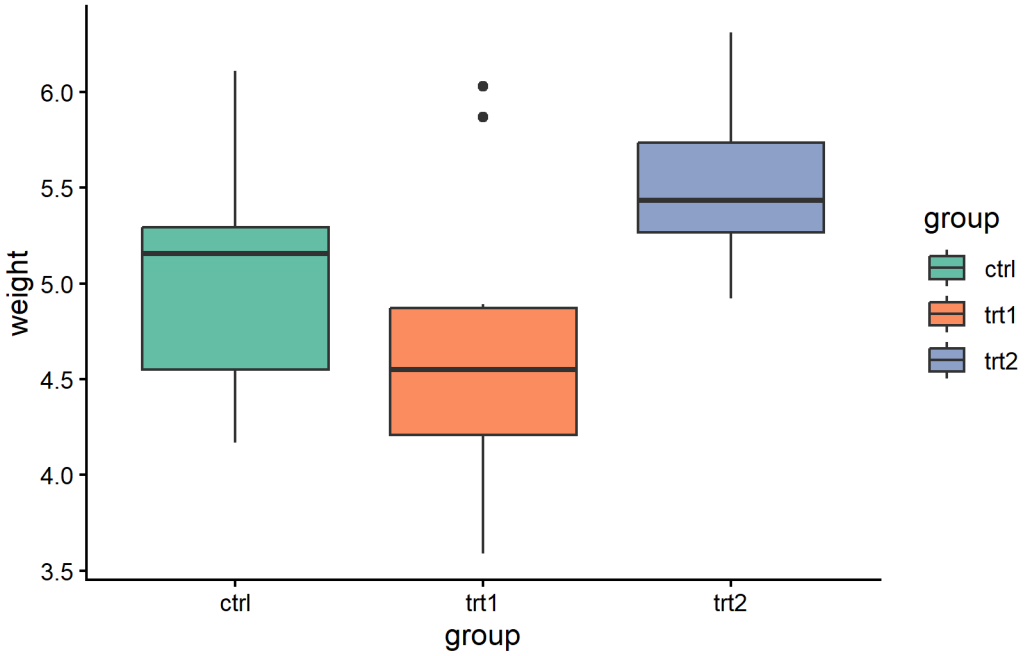


Color vs. fill

Ein haeufiger Stolperstein - besonders fuer Einsteiger - ist der Unterschied zwischen `color` und `fill`. Die Regel ist einfach: `color` steuert die **Umrandung** eines Geoms (Punkte, Linien, Raender von Balken), waehrend `fill` das **Innere** steuert (die gefuellte Flaeche von Balken, Boxplots, Violin Plots und bandartigen Geoms). Die entsprechenden Scale-Funktionen folgen demselben Muster: Alles was wir oben mit `scale_color_*()` behandelt haben, funktioniert identisch mit `scale_fill_*()`.

Die Unterscheidung ist besonders wichtig bei Geoms, die sowohl eine Umrandung als auch ein Inneres haben, wie `geom_boxplot()` oder `geom_col()`. Dort mappt man typischerweise `fill` auf eine Gruppierungsvariable (um den Koerper jedes Balkens oder Boxplots einzufaerben), waehrend man `color` beim Standard belaesst.

```
ggplot(data = PlantGrowth) +
  aes(y = weight, x = group, fill = group) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set2") +
  theme_classic()
```



Bibliography