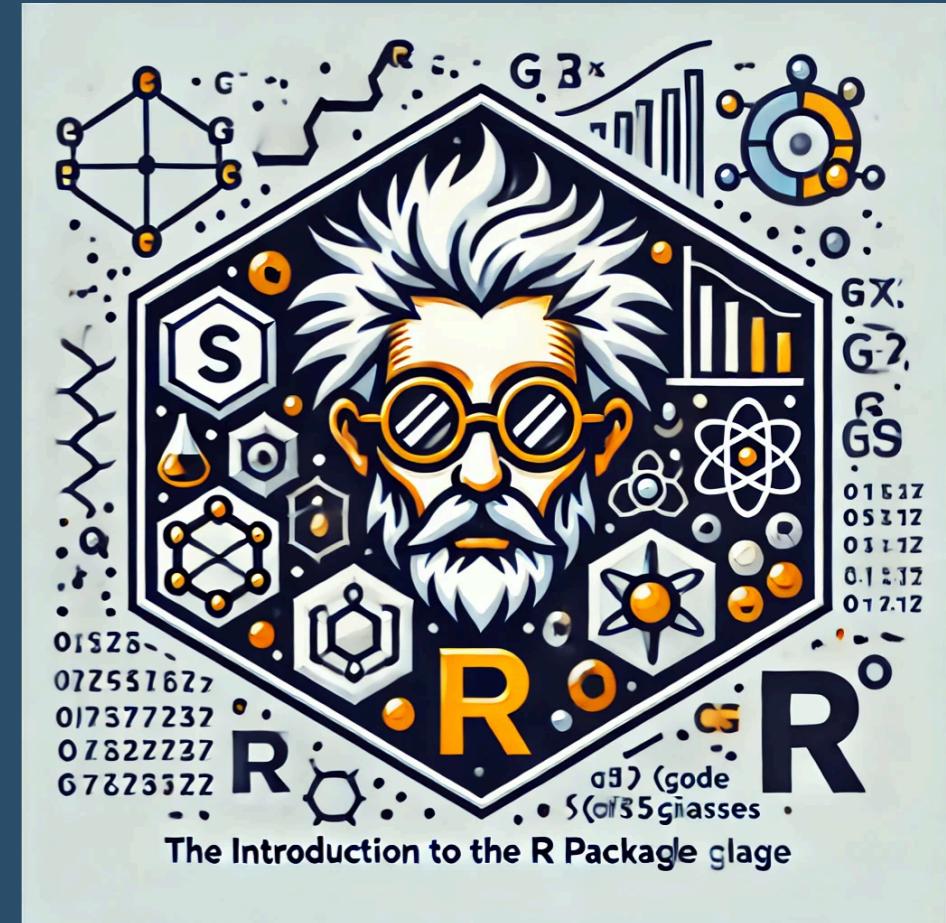


# Das Tidyverse und erste Schritte in ggplot2

Grundlagen der Datenanalyse und Statistik mit R

Prof. Dr. Daniel Schnitzlein

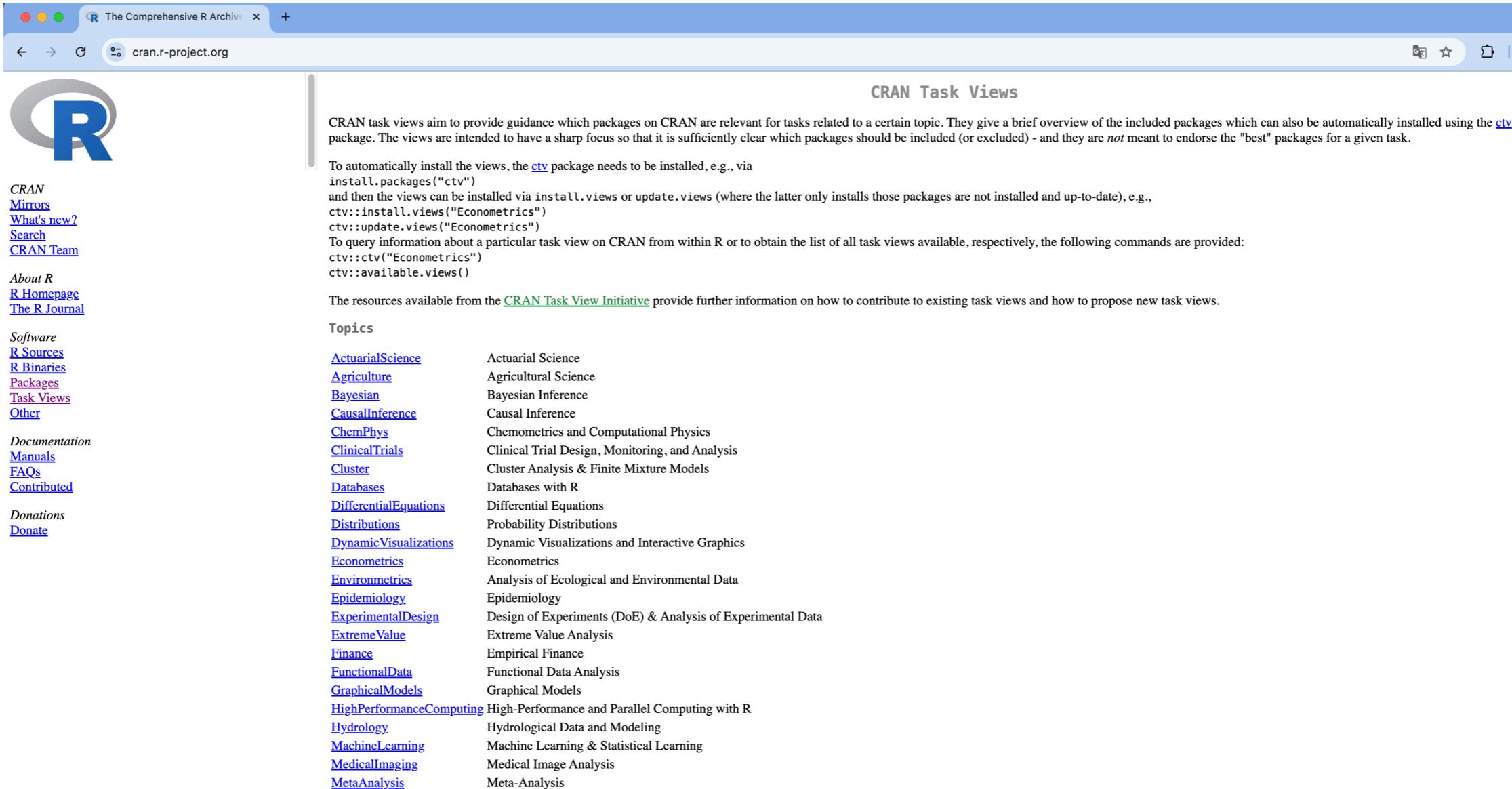
GitHub-Repo: [https://github.com/dschnitzlein/GDSR\\_2/](https://github.com/dschnitzlein/GDSR_2/)



# Was sind R-Pakete?

- R besteht aus einer Basiskomponente, die durch sogenannte R-Pakete erweitert werden kann um bestimmte Aufgaben zu lösen.
- Das wichtigste Repository für R-Pakete, das Comprehensive R Archive Network (CRAN), listet zur Zeit **23.039** (Dezember 2025) verfügbare Pakete auf (zuzüglich einer Unmenge von Paketen auf Github).
- Für alle (die meisten) verfügbaren Pakete ist eine umfangreiche Dokumentation verfügbar.
- Möglichkeiten den Überblick zu behalten:
  - CRAN Task Views (siehe nächste Seite)
  - Social Media Posts
  - R-User Konferenzen (meist auch via YouTube verfügbar) oder
  - Diskussionen in Foren und/oder GitHub.

# CRAN Task Views



The screenshot shows a web browser window displaying the CRAN Task Views page. The URL in the address bar is `cran.r-project.org`. The page features the R logo and navigation links for CRAN Mirrors, What's new?, Search, and CRAN Team. The main content area is titled "CRAN Task Views" and describes the purpose of task views. It includes code snippets for installing the `ctv` package and using it to install task views. It also provides commands for querying information about task views. A section titled "Topics" lists various task views with their descriptions:

<a href="#">ActuarialScience</a>	Actuarial Science
<a href="#">Agriculture</a>	Agricultural Science
<a href="#">Bayesian</a>	Bayesian Inference
<a href="#">CausalInference</a>	Causal Inference
<a href="#">ChemPhys</a>	Chemometrics and Computational Physics
<a href="#">ClinicalTrials</a>	Clinical Trial Design, Monitoring, and Analysis
<a href="#">Cluster</a>	Cluster Analysis & Finite Mixture Models
<a href="#">Databases</a>	Databases with R
<a href="#">DifferentialEquations</a>	Differential Equations
<a href="#">Distributions</a>	Probability Distributions
<a href="#">DynamicVisualizations</a>	Dynamic Visualizations and Interactive Graphics
<a href="#">Econometrics</a>	Econometrics
<a href="#">Environmetrics</a>	Analysis of Ecological and Environmental Data
<a href="#">Epidemiology</a>	Epidemiology
<a href="#">ExperimentalDesign</a>	Design of Experiments (DoE) & Analysis of Experimental Data
<a href="#">ExtremeValue</a>	Extreme Value Analysis
<a href="#">Finance</a>	Empirical Finance
<a href="#">FunctionalData</a>	Functional Data Analysis
<a href="#">GraphicalModels</a>	Graphical Models
<a href="#">HighPerformanceComputing</a>	High-Performance and Parallel Computing with R
<a href="#">Hydrology</a>	Hydrological Data and Modeling
<a href="#">MachineLearning</a>	Machine Learning & Statistical Learning
<a href="#">MedicalImaging</a>	Medical Image Analysis
<a href="#">MetaAnalysis</a>	Meta-Analysis

# Was bedeuten Pakete für R?

- Pakete erweitern den Funktionsumfang von R erheblich. Pakete werden von der Community und/oder Unternehmen erstellt und gepflegt.
- **Vorteil**: Durch die aktive und breite Community stellt R auch für hochspezialisierte Fragestellungen Lösungen bereit, die proprietäre Software u.U. nicht enthalten würde.
- **Vorteil**: Deutlich höhere Entwicklungsgeschwindigkeit neuer Methoden.
- **Nachteil**: Nicht alle Pakete sind zwingend miteinander kompatibel. Fallen die Akteure, die ein Paket betreuen aus, kann es sein, dass die Funktionalität verloren geht, wenn sich R weiterentwickelt.
- **Nachteil**: Ebenso umfasst Versionierung nicht nur R sondern auch alle Pakete.
- **Realität**: Viele populäre Pakete werden in Gruppen zusammengefasst entwickelt und gepflegt. Oft sind zentrale Figuren entweder bei Unternehmen angestellt, die einen Bezug zu R haben, oder die R extensiv nutzen.

# Es gibt Pakete die R erweitern, z.B. die **easystats** Paket-Gruppe

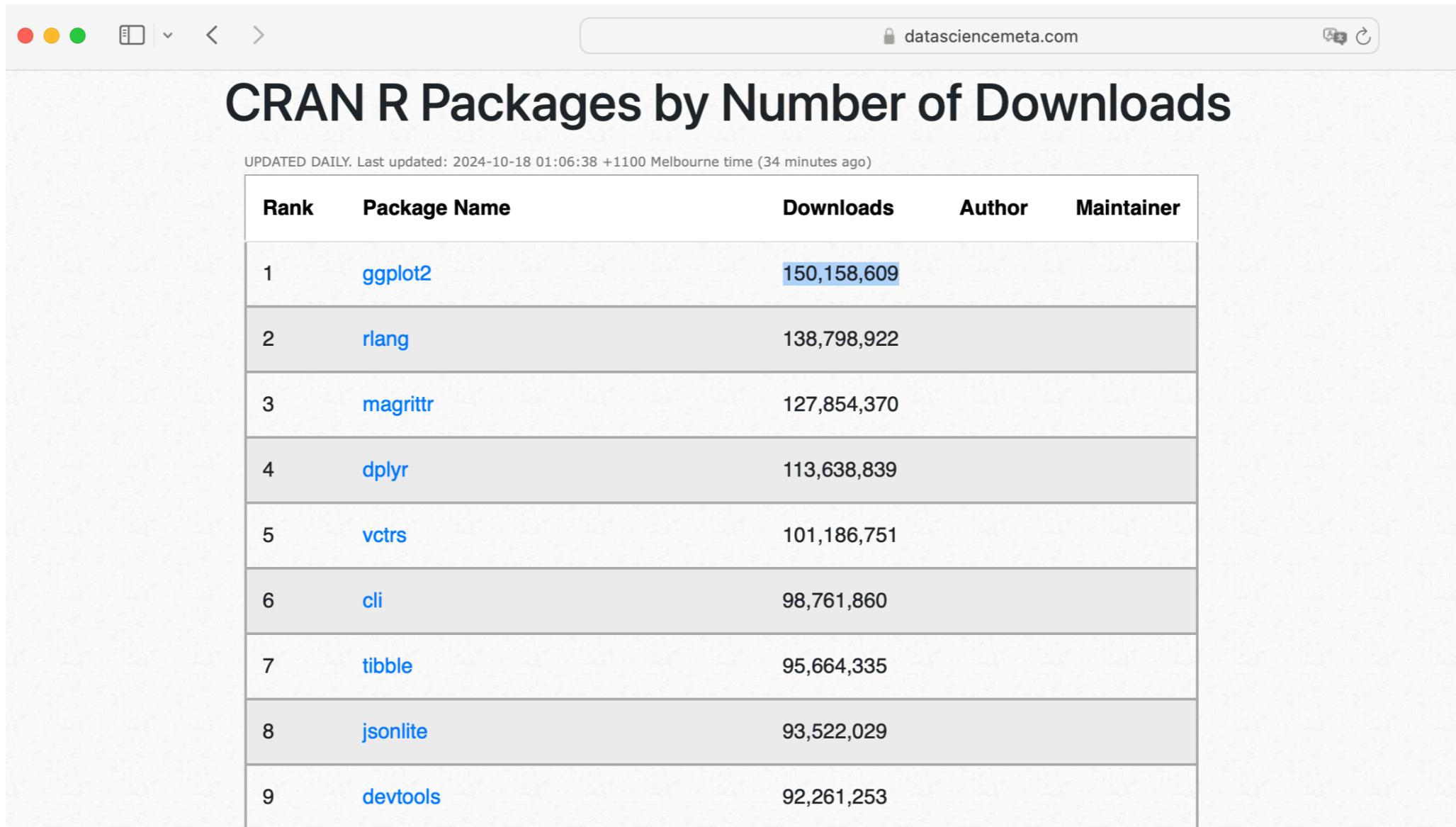


Es gibt Pakete die auch die Art und Weise verändern wie mit R gearbeitet wird, z.B. die Pakete aus dem **tidyverse**



GitHub-Repo: [https://github.com/dschnitzlein/GDSR\\_2/](https://github.com/dschnitzlein/GDSR_2/)

# Die wichtigsten Pakete nach Downloads



The screenshot shows a web browser window with the URL [datasciencemeta.com](https://datasciencemeta.com) in the address bar. The page title is "CRAN R Packages by Number of Downloads". A note at the top says "UPDATED DAILY. Last updated: 2024-10-18 01:06:38 +1100 Melbourne time (34 minutes ago)". Below is a table with the following data:

Rank	Package Name	Downloads	Author	Maintainer
1	<a href="#">ggplot2</a>	150,158,609		
2	<a href="#">rlang</a>	138,798,922		
3	<a href="#">magrittr</a>	127,854,370		
4	<a href="#">dplyr</a>	113,638,839		
5	<a href="#">vctrs</a>	101,186,751		
6	<a href="#">cli</a>	98,761,860		
7	<a href="#">tibble</a>	95,664,335		
8	<a href="#">jsonlite</a>	93,522,029		
9	<a href="#">devtools</a>	92,261,253		

# Die Rolle von Posit (früher RStudio)

- Die Entwicklung von R zu der Verbreitung die R heute hat, wäre ohne die grafische IDE RStudio die 2011 von der damals gleichnamigen Firma auf den Markt gebracht wurde nicht denkbar gewesen.
- RStudio hat aber auch massiv die Entwicklung von R von einem “Spielplatz für Akademiker” hin zu einer zuverlässigen Grundlage für Produktivsysteme vorangetrieben.
- RStudio verdient dabei ihr Geld mit der Bereitstellung von R Entwicklungsumgebungen (u.a. RStudio) für Unternehmen.
- Inzwischen hat sich RStudio in Posit umbenannt und hat neben dem Fokus auf R auch Python im Blick.
- Viele hoch innovative Pakete in R und Python werden direkt oder indirekt von Posit entwickelt.
- Dadurch ergibt sich natürlich eine gewisse Macht aber auch eine gewisse Notwendigkeit Standards zu setzen (siehe tidyverse).
- Nicht unbedingt in der ganzen R Community beliebt.

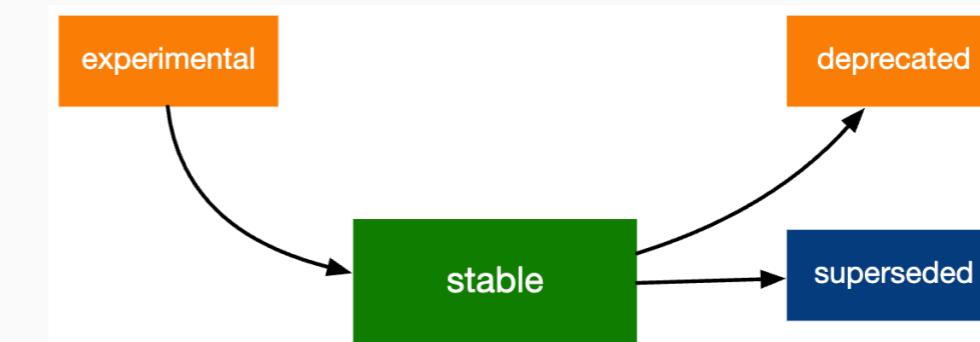
# Beispiel: Lebenszyklus für Funktionen

- Ein weit verbreitetes Problem mit Funktionen in R Paketen war lange Zeit (und ist es stellenweise immer noch), dass diese teilweise sehr kurzfristig durch neue Versionen ersetzt werden.
- Insbesondere in Produktivumgebungen ist das sehr ärgerlich, wenn z.B. Dashboards nach Updates neu angepasst werden müssen.
- Idealerweise kündigen die jeweiligen Administratoren der Pakete solche Änderungen an. Jedoch ist das nicht immer so.
- Posit/RStudio hat um diesem Problem zu beheben einen Lebenszyklus für Funktionen/Pakete etc. eingeführt.

## Lifecycle stages

Source: [vignettes/stages.Rmd](#)

This vignette describes the four primary stages of the tidyverse lifecycle: stable, deprecated, superseded, and experimental.



A diagram showing the transitions between the four main stages: experimental can become stable and stable can become deprecated or superseded.

The lifecycle stages can apply to packages, functions, function arguments, and even specific values of a function argument. However, you'll mostly see them used to label individual functions, so that's the language we use below.

# Beispiel: Positron

The screenshot shows the RStudio interface with the following details:

- Title Bar:** Aufgabenblatt\_1.R — GDSR\_1
- File Menu:** New, Open, Quarto, etc.
- Version:** R 4.4.1
- Project:** GDSR\_1
- Explorer:** Shows files: .Rhistory, Abbildung1.pdf, Aufgabenblatt\_1.R, WestRoxbury.csv
- Code Editor:** Displays the R script "Aufgabenblatt\_1.R". The code performs data cleaning, preparation, and analysis on the "WestRoxbury.csv" dataset.
- Session View:** Shows the R environment with no variables created.
- Console View:** Shows the R startup message and the command "Show Traceback".
- Variables View:** Shows the R environment with no variables created.
- Status Bar:** Zeile 9, Spalte 1 (34 ausgewählt), Leerzeichen: 5, UFT-8, LF, R

# Das tidyverse besteht aktuell aus acht Kernpakten



*At a high level, the tidyverse is a language for solving data science challenges with R code. Its primary goal is to facilitate a conversation between a human and a computer about data.*

*Less abstractly, the tidyverse is a collection of R packages that share a high-level design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next.*

(Wickham et al. 2019)

# In der aktuellen Version 2.0.0 sind das folgende Pakete

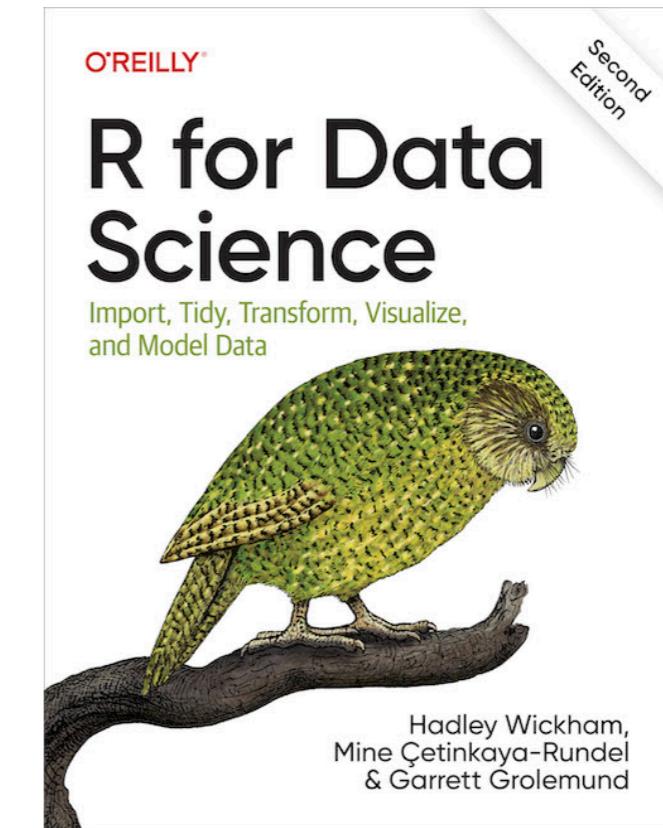
- `ggplot2` ist ein System zur deklarativen Erstellung von Abbildungen, das auf der `Grammar of Graphics` basiert.
- `dplyr` bietet eine Grammatik der Datenmanipulation und stellt einen einheitlichen Satz von "Verben" bereit, mit denen sich die häufigsten Aufgaben bei der Datenmanipulation lösen lassen.
- `tidyverse` bietet eine Reihe von Funktionen um Daten in `tidy data` zu transformieren. `Tidy data` sind Daten in einer einheitlichen Form.
- `readr` bietet eine schnelle und benutzerfreundliche Möglichkeit, rechteckige Daten (wie z.B. csv) einzulesen. Es ist so konzipiert, dass es viele verschiedene Arten von Daten flexibel handhaben kann.
- `purrr` erweitert das Toolkit für funktionale Programmierung (FP) von R um einen vollständigen und einheitlichen Satz von Werkzeugen für die Arbeit mit Funktionen und Vektoren.
- `tibble` ist eine moderne Neuinterpretation des Data Frame mit einigen Zusatzfeatures.
- `stringr` bietet eine Reihe Funktionen, die die Arbeit mit Strings (Zeichenketten) so einfach wie möglich machen sollen.
- `forcats` bietet eine Reihe nützlicher Tools, die gängige Probleme mit Faktoren lösen.
- `lubridate` bietet eine Reihe nützlicher Funktionen zur Arbeit mit Datumsangaben und Uhrzeiten.

# Die Zielsetzung des tidyverse

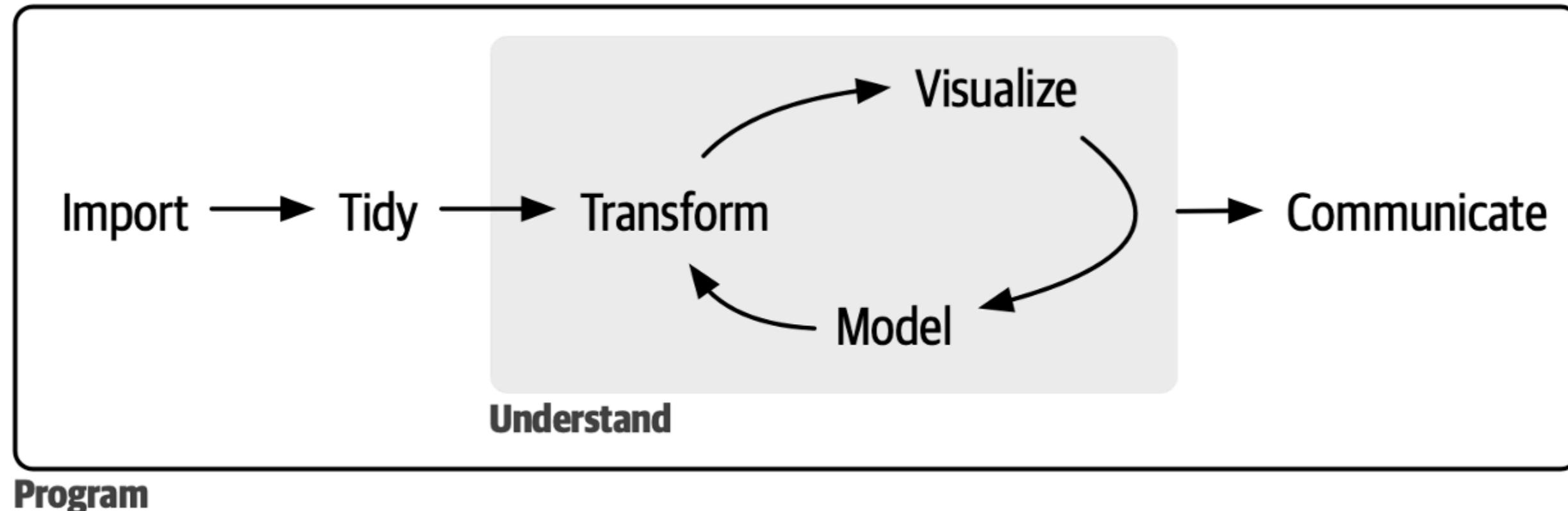
- Liegen Daten in der Form vor, wie das tidyverse die Daten gerne hätte (tidy data), dann sollten alle Funktionen aller tidyverse-Pakete und der sich auf das tidyverse berufenden Pakete **out-of-the-box** funktionieren.
- Alle Funktionen des tidyverse sollen einer gemeinsamen Logik im Funktionsaufbau folgen.
- Die tidyverse Pakete orientieren sich dabei an einem genormten Ablauf einer Datenanalyse.
- Dieser hohe Standardisierungsgrad hat dazu beigetragen z.B. ggplot2 zum wichtigsten R-Paket zu machen mit aktuell (Dezember 2025) 174 Mio. Downloads (über CRAN).
- Posit wendet dabei die Standards sowohl auf R als auch auf Python an. Mittelfristig ist das Ziel einen nahtlosen Austausch zwischen Python und R Paketen zu ermöglichen.

# Datenanalyse mit dem tidyverse

- Im Internet gibt es zahlreiche Tutorials, die erklären, wie man mit den Paketen des tidyverse arbeitet.
- Eine sehr gute und umfassende Referenz ist Hadley Wickhams Buch **R for Data Science**.
- Die 2. Auflage des Buches ist auch als (Open-Source-)Online-Version unter <https://r4ds.hadley.nz/> verfügbar.
- Die Ausführungen in dem Buch sind Grundlage dieses Abschnitts.

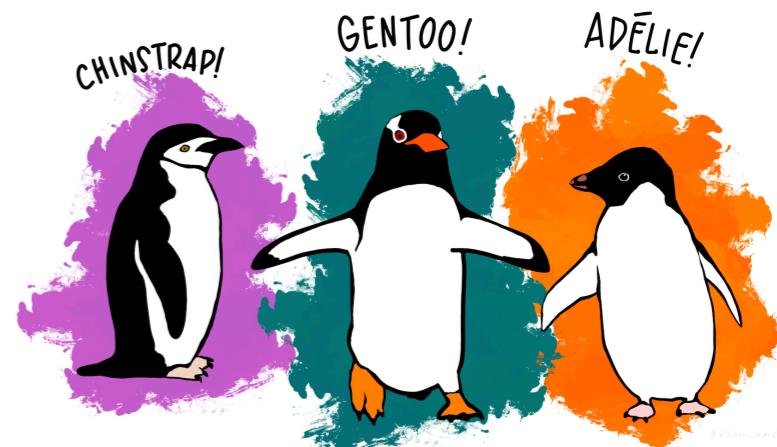


# Der Datenanalyse Prozess



# Wir starten heute mit Datenvisualisierung

- Wir müssten eigentlich zuerst damit anfangen zu lernen Daten zu *importieren* und in die *richtige Form* (tidy data) zu bringen.
- Allerdings ist das zu 80% öde und langweilig und zu 20% nervend und frustrierend (Zitat aus R for Data Science).
- Daher starten wir erstmal mit Visualisieren und Transformieren von Daten die bereits in optimaler Form vorliegen.
- Lassen Sie uns aufs Eis gehen.



# Unser Datensatz palmerpenguins

- Der Datensatz den wir hier nutzen ist der Palmer Penguins Datensatz (344 Pinguine):
- Die Informationen wurden von 2007 bis 2009 von Kristen Gorman im Rahmen des Palmer Station Long Term Ecological Research Program gesammelt.
- Der Datensatz enthält drei qualitative Variablen und vier quantitative Variablen und ermöglicht dadurch viele verschiedene Visualisierungen.
- Sie können die Daten via `install.packages('palmerpenguins')` installieren und wie jedes andere R-Paket auch via `library(palmerpenguins)` laden. Der Datensatz heißt **penguins**.



R Code ⟳ Start Over

▷ Run Code

```
1 install.packages('palmerpenguins')
2 library(palmerpenguins)
```

Details zum Datensatz finden Sie hier: <https://allisonhorst.github.io/palmerpenguins/index.html>

# Die Hauptdarsteller



Adeliepinguin

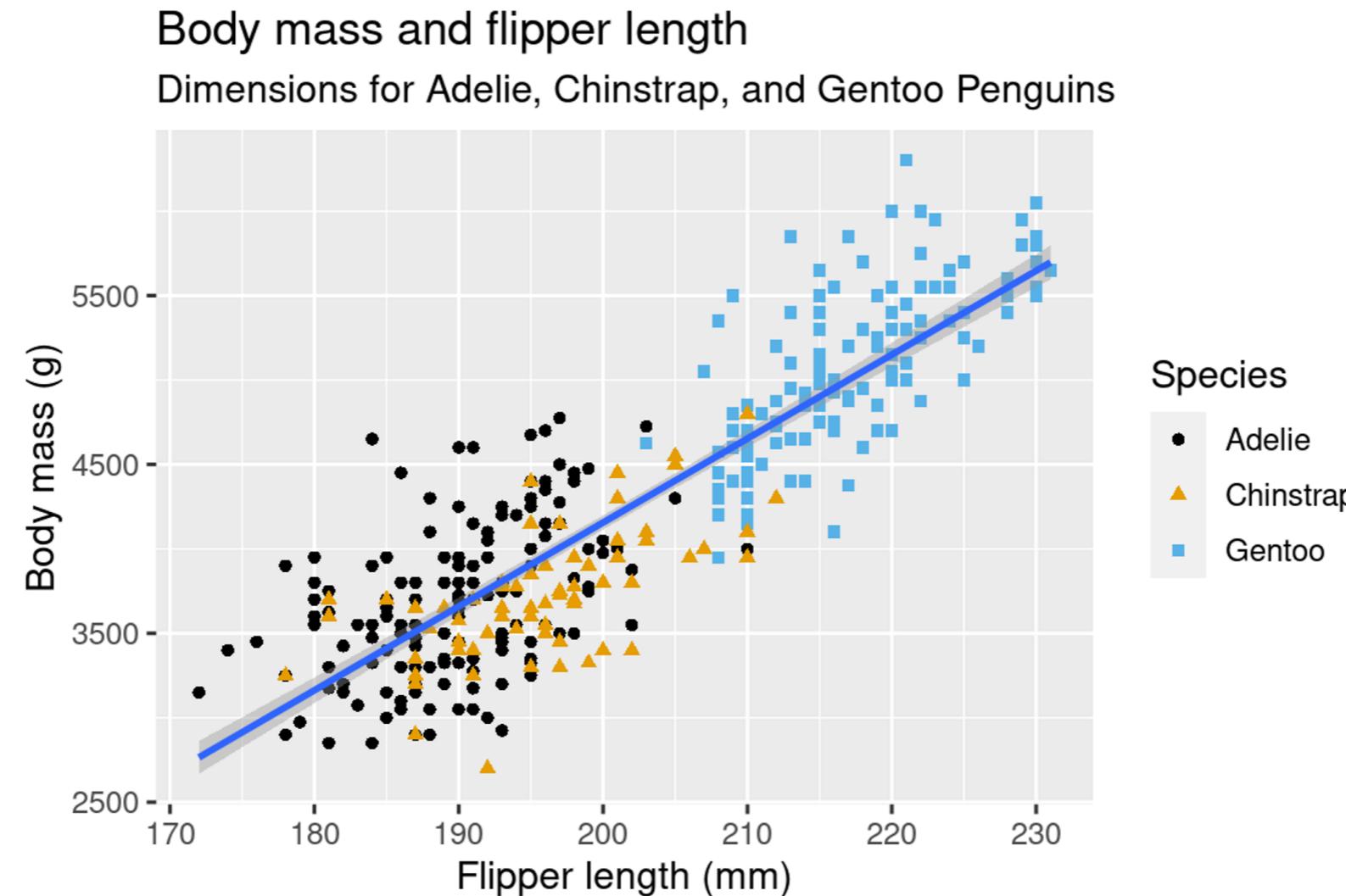


Chinstrap Pinguin / Zügelpinguin



Gentoo Pinguin / Eselspinguin

# Haben Pinguine mit längeren Flossen ein höheres oder ein niedrigeres Gewicht im Vergleich zu Pinguinen mit kürzeren Flossen?



# Data Visualization mit ggplot2

- **ggplot2** ist Teil des **tidyverse** und ein (das wichtigste) R-Paket zur Erstellung von Datenvisualisierungen.
- **ggplot2** basiert auf einer zugrundeliegende Grammatik, der **Grammar of Graphics** (Wilkinson 1999), die es ermöglicht, beliebige Abbildungen durch die Kombination unabhängiger Komponenten zusammenzustellen.
- Das macht **ggplot2** so leistungsfähig. Anstatt auf eine Reihe von vordefinierten Abbildungen beschränkt zu sein, können Sie neue Visualisierungen erstellen, die auf Ihr spezifisches Problem zugeschnitten sind.
- Es gibt viele gute Online Tutorials und Open Access Bücher zu **ggplot2**:
  - z.B. das Buch von Hadley Wickham: <https://ggplot2-book.org/> oder das
  - **ggplot2** Cheat Sheet von posit <https://posit.co/wp-content/uploads/2022/10/data-visualization-1.pdf>



# Schlüsselkomponenten

Jede ggplot2 Abbildung hat drei zentrale Komponenten: Daten, aesthetic mappings zwischen Variablen und Elementen der Abbildung sowie mindestens einen Layer (Schicht), die beschreibt wie die Beobachtungen dargestellt werden sollen. Layer können mit einer geom Funktion erstellt werden.

R Code ⟳ Start Over

▷ Run Code

```
1 library(ggplot2) # Alternativ könnten wir auch das Meta Paket tidyverse einbinden.
```

Ein erstes einfaches Beispiel: Ein Scatterplot (Streudiagramm) beschrieben durch data: penguins, aesthetic mapping: flipper\_length\_mm auf der X-Achse und body\_mass\_g auf der Y-Achse, layer: Punkte.

R Code ⟳ Start Over

▷ Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) + geom_point()
```

# Struktur des Funktionsaufrufs

data und aesthetic mappings sind in ggplot() enthalten und die layer werden mit + hinzugefügt. Die Eigenschaften in ggplot() werden an die layer vererbt.

R Code  Start Over

 Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) + geom_point()
```

Hinweis: Die Zusätze data=, x=, und y= in der ggplot() Funktion sind nicht unbedingt notwendig aber machen den Code lesbarer. Probieren Sie es einfach aus!

# Colour, size, shape und andere aesthetic attributes

- Die **aesthetic attributes** `colour`, `size` und `shape` können genutzt werden um zusätzliche Variablen zu einer Abbildung hinzuzufügen. Sie werden in `aes()` mit eingebunden und können auch kombiniert werden.

R Code 

 Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g, colour=species)) + geom_point()
```

- `ggplot2` kümmert sich um die Details der Erstellung einer passenden Skala und Legende. Später werden wir diese auch ausblenden oder bearbeiten.
- Probieren Sie was oben passiert wenn Sie statt `species` die Variable `bill_length_mm` nutzen.
- Wenn ein Element auf eine `fixes` Attribut gesetzt werden soll, können Sie dieses direkt in der `geom_` Funktion angeben.

R Code 

 Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) + geom_point(colour='blue')
```

# Faceting

Faceting bietet die Möglichkeit die Daten nach Kategorien auszuwerten.

R Code

⟳ Start Over

▷ Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) + geom_point() + facet_wrap(~species)
```

# Code Konventionen

Um den Code lesbarer zu machen, ist es Konvention jede Schicht oder zusätzliches Element auf eine eigene Zeile zu schreiben.

R Code ↻ Start Over ▷ Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) +  
2   geom_point() +  
3   facet_wrap(~species)
```

# Oft genutzte geoms

`geom_point( )`

Streudiagramm (Punkte)

`geom_smooth( )`

u.a. Trendlinien inkl. Konfidenzintervalle

`geom_boxplot( )`

Boxplot

`geom_histogram( )` und `geom_freqpoly( )`

Histogramme bzw. Häufigkeitsverteilung

`geom_bar( )`

Balken-/Säulendiagramm

`geom_path( )` und `geom_line( )`

Verbindungen zwischen Datenpunkten

<https://r-graph-gallery.com/>

Viele weitere Beispiele

Probieren Sie es aus:

R Code ↻ Start Over ▷ Run Code

```
1 ggplot(data=penguins, aes(x=flipper_length_mm, y=body_mass_g)) +  
2   geom_point()
```

# Zurück zu unserer wichtigen Fragestellung

Der vollständige Befehl für die Abbildung weiter vorne im Skript lautet:

R Code ↻ Start Over ▷ Run Code

```
1 library(ggthemes)
2
3 ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
4   geom_point(aes(color = species, shape = species)) +
5   geom_smooth(method = "lm") +
6   labs(title = "Body mass and flipper length", subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", x
7     scale_color_colorblind()
```

# Aufbau der Abbildung Stück für Stück

R Code

⟳ Start Over

▷ Run Code

```
1 ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g))
```

# Themes (später mehr dazu)

ggplot2 ermöglicht es anhand von Themes die Formatierung von Abbildungen weitgehend zu automatisieren (z.B. Anwendung von einheitlichem Corporate Design). Wir werden uns mit Themes später im Seminar im Detail beschäftigen.

R Code 

 Run Code

```
1 ggplot(data = penguins, aes(x = flipper_length_mm, y = body_mass_g)) +  
2   geom_point(aes(color = species, shape = species)) +  
3   geom_smooth(method = "lm") +  
4   labs(title = "Body mass and flipper length", subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", x =  
5     theme_economist())
```

# Man kann Themes zu allem (!) im Internet finden

