FH JOANNEUM
University of Applied Sciences

Von Zahlen zu Bildern

Interaktive Dashboards in Python

Ergänzende Folien

LAND KÄRNTEN

FH JOANNEUM
University of Applied Sciences

DIGITAL INNOVATION HUB SÜD

Digital Innovation Hub Süd

- Der *Digital Innovation Hub Süd (DIH SÜD)* ist ein **Kompetenznetzwerk**, das **Klein- und Mittelbetriebe (KMU)** bei der **digitalen Transformation** mit Expertise, Vernetzung und Infrastruktur unterstützt.
- Services:
  - Durchführung von Informationsveranstaltungen
  - Aktivitäten der Innovations- und Technologieberatung
  - **Durchführung von Qualifizierungsmaßnahmen (Schulungen)**
  - Begleitung bei der Entwicklung von Innovationen

Regionen und Netzwerk

- Den KMUs steht dabei das regionale Forschungs- und Innovationssystem folgender Bundesländer zur Verfügung: Steiermark, Kärnten, Burgenland, Osttirol
- Das Netzwerk des DIH Süd umfasst **Hochschulen**, **Forschungszentren** und **Inkubatoren** aus den genannten Regionen.

Finanzierung

Themenbereiche von DIH Süd

- Produktions- und Fertigungstechnologien
- Digitale Sicherheit
- **Daten & Künstliche Intelligenz**
- Digitale Geschäftsmodelle & -prozesse
- Nachhaltigkeit & Kreislaufwirtschaft
- Arbeit der Zukunft & Humanressourcen

VORSTELLUNG

FH JOANNEUM
University of Applied Sciences

DI Dr. Hannah Wimmer BSc

Ausbildung

- Diplomstudium Techn. Physik (TU Graz)
- Doktoratsstudium Biomedical Eng. (TU Graz / NTU Singapur)

Berufliche Aktivität

- Hochschullektorin an der FH JOANNEUM

Fokus in Forschung und Lehre

Signalverarbeitung, Computer Vision, Large Language Models, Graphentheorie

FH JOANNEUM
University of Applied Sciences

Mag. Raphaele Raab MSc

Ausbildung

- Lehramtstudium Mathematik
  (KFU Graz)
- Masterstudium
  "Data and Information Science"
  (FH JOANNEUM Graz)

Berufliche Aktivität

- Hochschullektor an der FH JOANNEUM

Fokus in Forschung und Lehre

Scripting, Machine Learning, Data Preprocessing, Mathematik

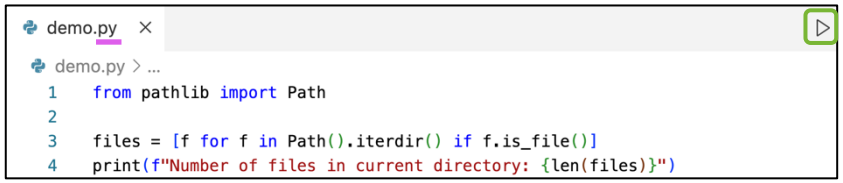
PYTHON SETUP – WICHTIGE BEGRIFFE

Programme, Skripte und Notebooks

- Ein **Programm** ist eine **Sammlung von Anweisungen** in einer bestimmten Programmiersprache, die der Computer ausführt, um eine bestimmte Aufgabe zu erledigen (z. B. Datenanalyse, Automatisierung).
- Ein Programm **besteht aus einem oder mehreren Skripten** (*.py*-Dateien) und weiteren Dateien (z.B. Konfigurationsdateien).
- Ein Skript ist ein einfaches Programm in einer einzelnen Datei – wird vereinfacht gesagt von oben nach unten ausgeführt.
- Ein **Jupyter Notebook** (*.ipynb*-Dateien) ist ein **interaktives Skript**, das Code, Ergebnisse und erklärenden Text kombiniert.

Skript

- Ein Skript-Datei in Python hat die Endung ***.py***.



- Um den darin enthaltenen Code auszuführen, klicken wir den **Run-Button** oder führen den Code mit direkt im Terminal mit folgendem Befehl aus:

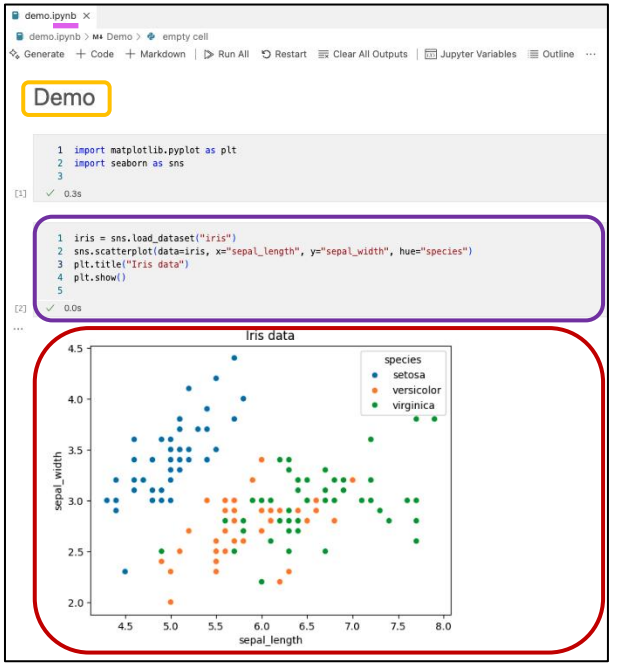
```
uv run <filename>
```

```
❯ uv run demo.py
```
← Terminal Input

```
Number of files in current directory: 4
```
← Terminal Output

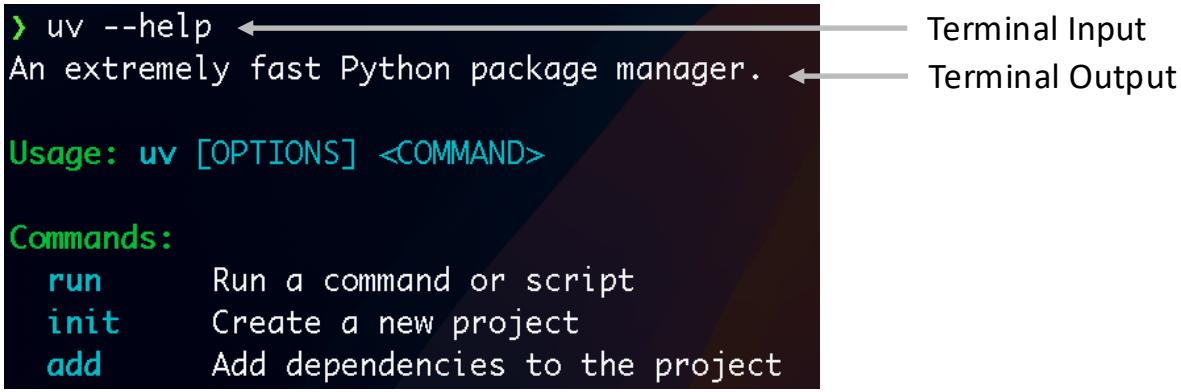
Juypter Notebook

- Ein Notebook hat die Endung ***.ipynb***.
- Es besteht aus mehreren Code-Zellen (**Input**).
- Der **Output** jeder Zelle wird direkt im Notebook angezeigt.
- **Text** wird in der Markup-Sprache *Markdown* geschrieben.



Terminal

- Terminal = Programm, das eine **textbasierte Schnittstelle** bereitstellt, über **die** man **Befehle an** den **Computer sendet**
- Shell = Software, die diese Befehle interpretiert und ausführt (z.B. *PowerShell* (*Windows*), *Bash*, …)



Wozu benötigt man ein Terminal?

- **Installation** von *Python* und *Python*-Paketen
- **Aktivierung von** sogenannten Virtual **Environments**
- **Ausführen von** *Python*-**Programmen**

Integrated Development Environment (IDE)

- Eine IDE ist ein Programm, das das Schreiben, Testen und Verwalten von Code erleichtert.
- Sie **kombiniert Funktionen** wie Terminalzugriff, Fehlermeldungen, Autovervollständigung und Debugger in einer Oberfläche.
- Wir können dafür **Visual Studio Code** (VS Code) als IDE **mit zusätzlichen Erweiterungen** (Extensions), insbesondere *Python* und *Jupyter*, verwenden.

Programmiersprache und Interpreter

- Jede Programmiersprache (z.B. Python) besitzt ein eigenes **Vokabular** (z.B. Schlüsselwörter wie `if`, `for`, `def`) und eine eigene **Grammatik** (Syntax).
- Damit der **Computer Code** in einer solchen Sprache **verstehen** und ausführen kann, braucht es einen **Interpreter** – ein Programm, das den Code Zeile für Zeile übersetzt und ausführt.
- Bei Python übernimmt das z.B. die Anwendung *python.exe* (unter *Windows*).

PYTHON INSTALLATION UND VIRTUAL ENVIRONMENTS

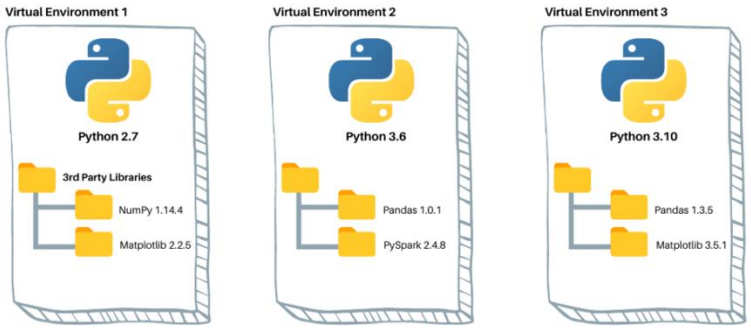
Herausforderungen bei der Installation

- Es existieren **verschiedene Python Versionen.**
- Es ist sinnvoll, sogenannte **Virtual Environments** zu **erstellen**.
- **Externe Python-Pakete** haben ebenfalls unterschiedliche Versionen und diese weisen **Abhängigkeiten** untereinander auf.

- Mit der Software **uv** lassen sich diese Herausforderungen effizient meistern.

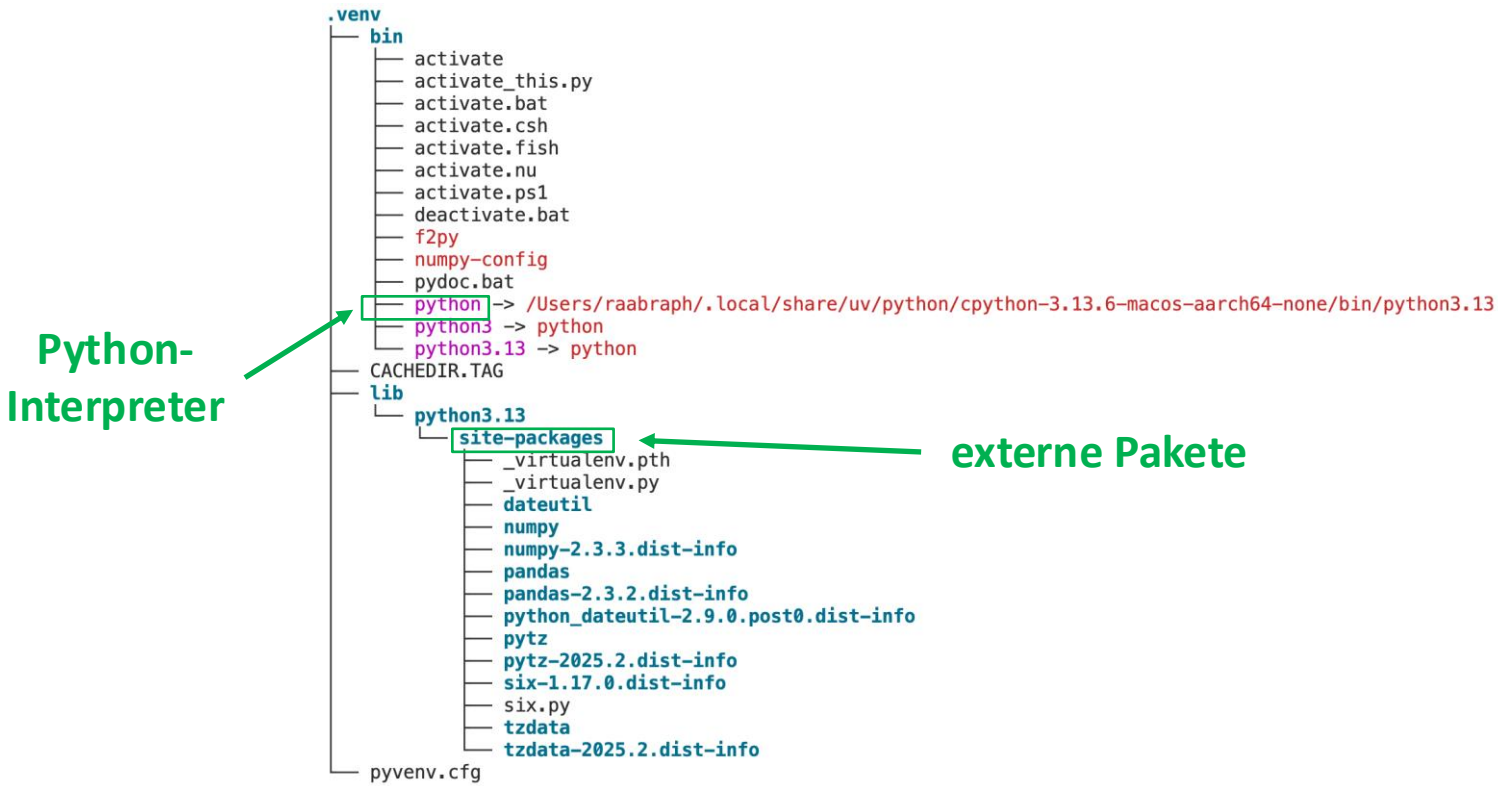
Virtual Environment

- = Ordnerstruktur (*.venv* Ordner), die alles Relevante beinhält um Python-Code ausführen zu können:
  Python-Interpreter (Windows: *python.exe*) und installierte **externe Python-Pakete** (z.B. *pandas*)



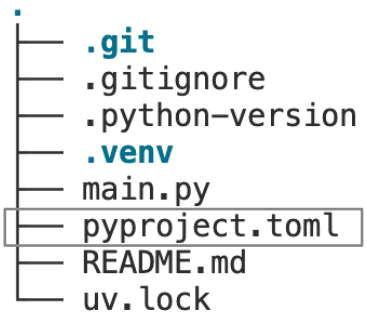
verschiedene Virtual Environments

Beispiel (MacOS)



Konfigurationsdateien

- Konfigurationsdateien (*pyproject.toml*, *uv.lock*) halten den Zustand eines solchen *Virtual Environments* fest.




```
[project]
name = "demo-uv"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.13"
dependencies = [
    "pandas>=2.3.2",
]
```

Python-Version

externe Python-Pakete

beispielhafte *pyproject.toml* Datei

Weshalb benötige ich ein *Virtual Environment*?

- unterschiedliche *Python*-Projekte erfordern unterschiedliche externe *Python*-Pakete
- komplexe Abhängigkeiten zwischen unterschiedlichen externen *Python*-Pakete
- *Linux* and *macOS* verfügen über eine vorinstallierte *Python*-Version, die bereits für interne Prozesse verwendet wird

→ Empfehlung:
Für jedes Projekt sollte ein eigenes Virtual Environment erstellt werden!

Installation uv

- Die Befehle zur Installation von **uv** finden unter https://docs.astral.sh/uv/.
- Kopiere den entsprechenden Befehl in die *PowerShell* (*Windows*) oder das *Terminal* (*macOS*).
- Um zu überprüfen, ob die Installation erfolgreich war, starte die *PowerShell* bzw. das *Terminal* neu und teste mit folgendem Befehl:

```
uv --help
```

Installation von Python mittels uv

- Mit folgendem Befehl (in *PowerShell* bzw. im *Terminal*) können wir eine bestimmte Python-Version installieren:

```
uv python install 3.13
```

Installation des Virtual Environments mittels uv

- Zur Installation eines Virtual Environments führen wir die folgenden beiden Befehle aus:

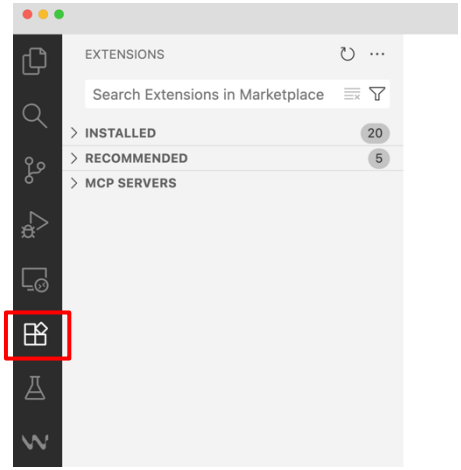
```
uv init --bare
```

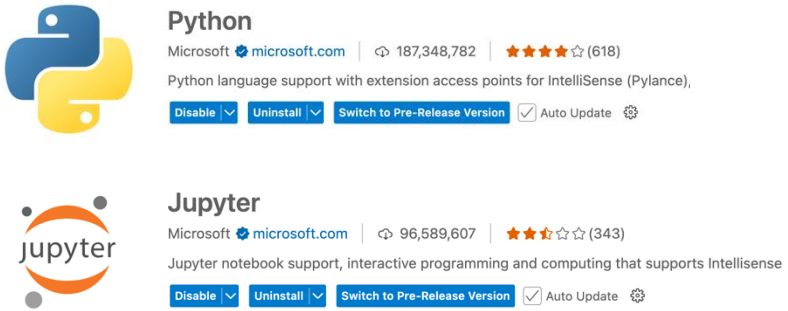
```
uv add <package-name>
```

- Damit erstellen wir die zuvor erwähnten Konfigurationsdateien, unser Virtual Environment (*.venv* Ordner) und fügen externe *Python*-Pakete hinzu. `<package-name>` muss durch die gewünschten Pakete (z.B. `ipykernel`, `pandas`) ersetzt werden.

VS Code Extensions

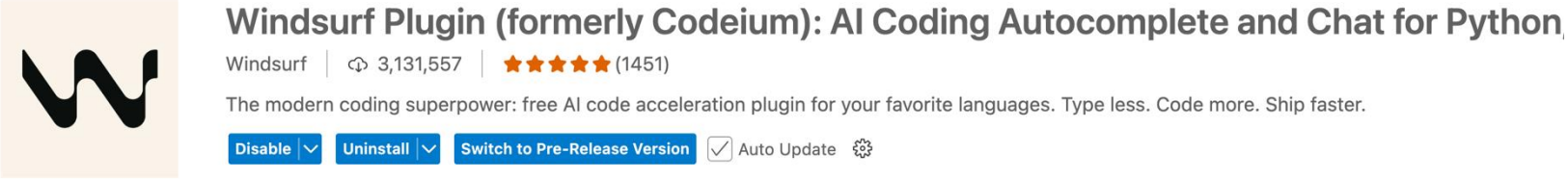
- In VS Code sollten auf jeden Fall die folgenden zwei **Extensions** installiert werden:





AI Coding Assistants

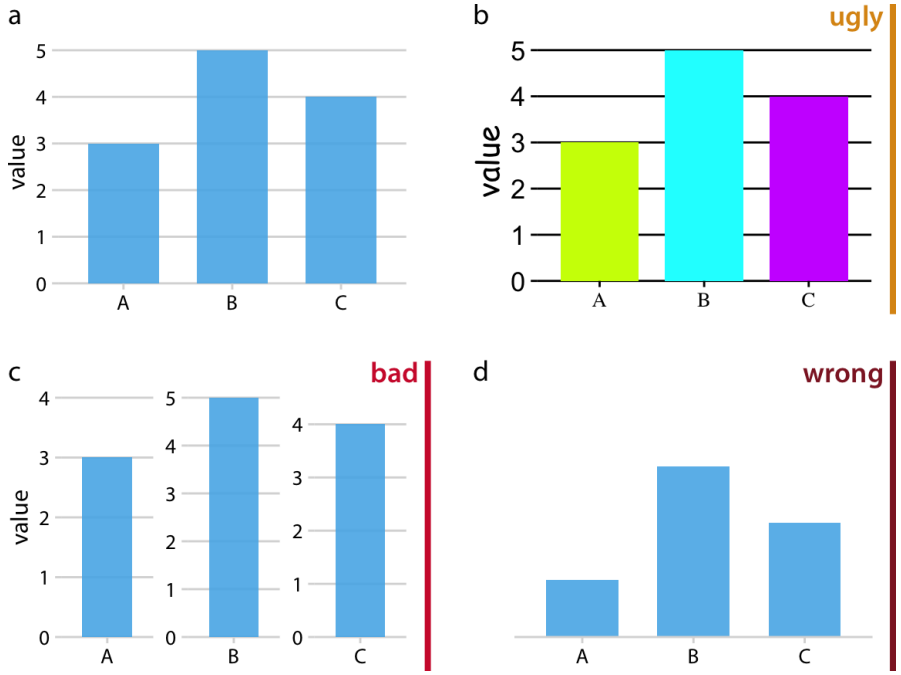
- Für die Verwendung von *GitHub Copilot* kann man folgender Anleitung folgen: https://code.visualstudio.com/docs/copilot/setup
- Sollte *Copilot* nicht funktionieren, kann man auch die Extension *Windsurf* verwenden.



GUIDELINES FOR DATA VISUALIZATION

Goal of data visualization

- Data visualization is part science and part art.
- A visualization should therefore be both accurate and aesthetically pleasing.



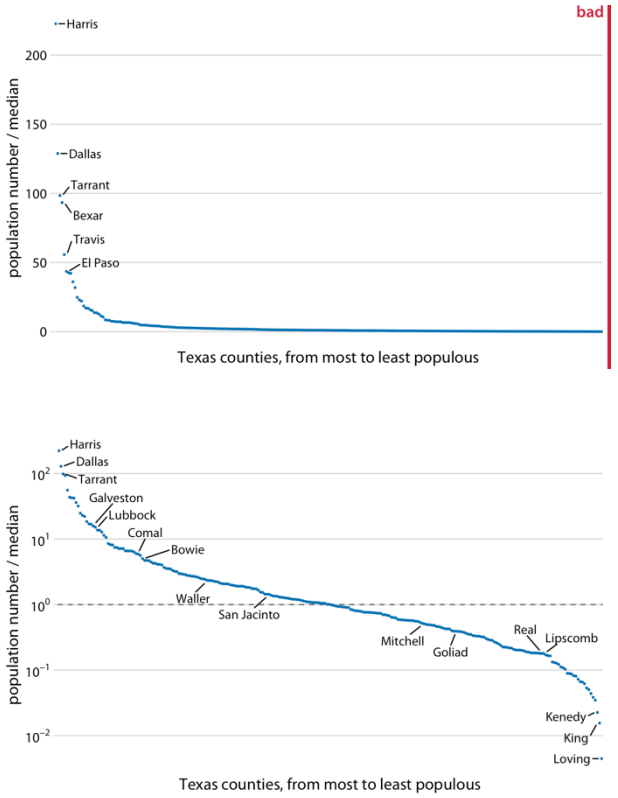
Q: https://clauswilke.com/dataviz/

Coordinate systems and axes

- Cartesian coordinate system
- Non-linear axes
- Polar coordinate system

Linear vs. non-linear axes

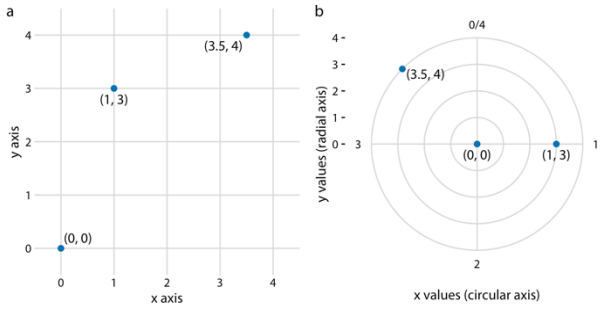
- linear: evenly spaced
- non-linear
  (e.g. logarithmic):
  not evenly spaced, can be used to better plot data, that varies across different orders of magnitude
  (here: proportions)

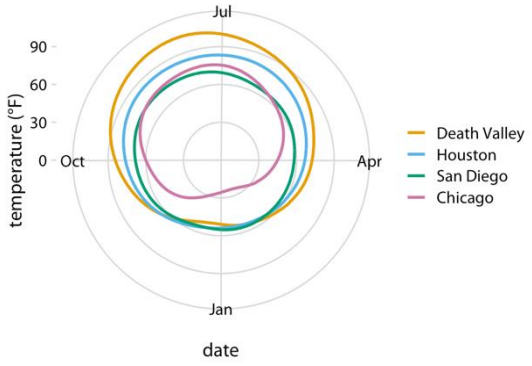


Q: https://clauswilke.com/dataviz/

Polar coordinate system

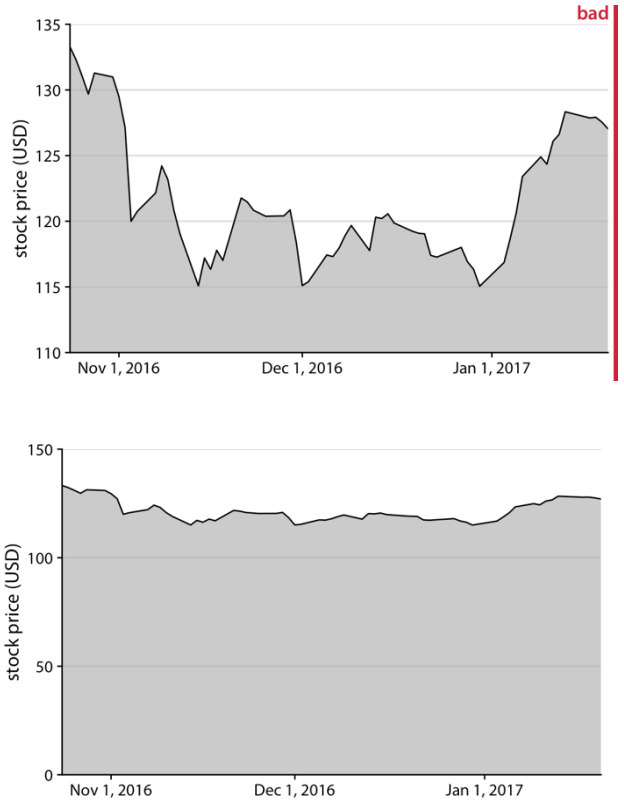
- In a polar coordinate system positions are specified via angle and distance from the origin.
- This display of temperatures throughout the year better illustrates the similarity between temperatures in December and January.





Coordinates axes and origin

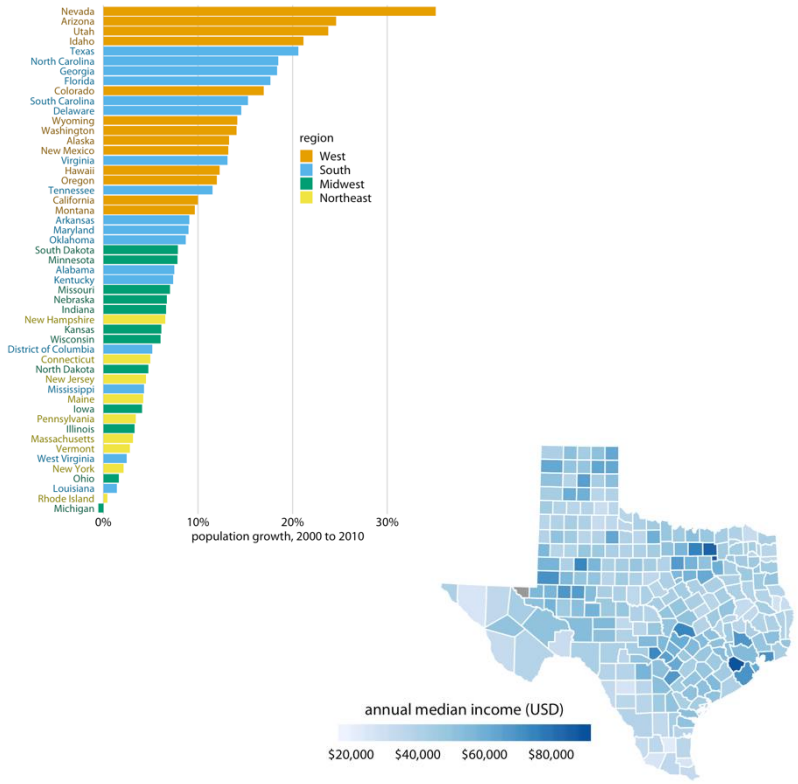
- Visualizations should start at zero.
- In the chart above, the fluctuations appear larger than they actually are.



Q: https://clauswilke.com/dataviz/

Color scales

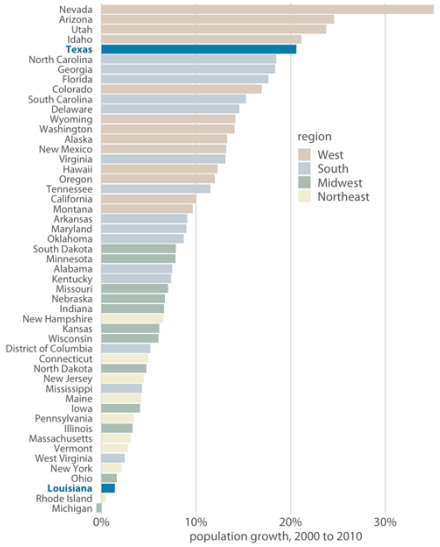
Qualitative features (here: region) should be displayed using qualitative color scales, while quantitative features (here: income) should be displayed using sequential or diverging color scales.

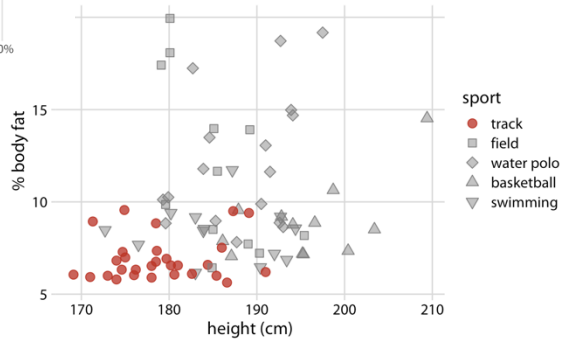


Q: https://clauswilke.com/dataviz/

Colors to highlight

Color can be used effectively to highlight important aspects of the data.

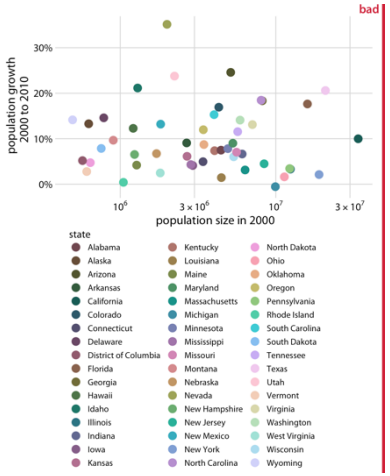


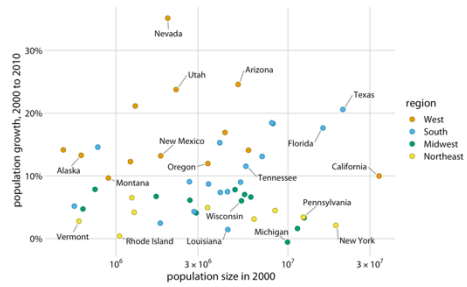


Q: https://clauswilke.com/dataviz/

Using to many colors

Instead of displaying all categories of a high-cardinality categorical variable, it is better to group them and highlight the most important ones using labels.

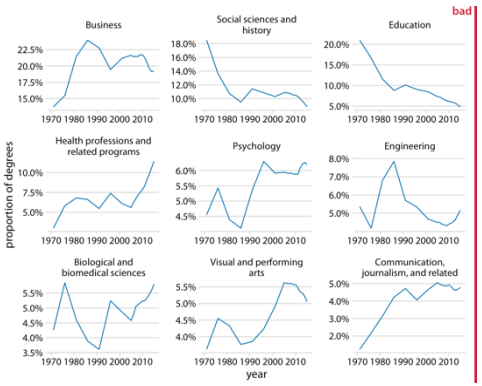


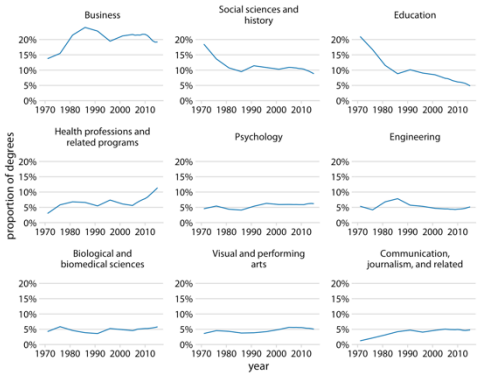


Q: https://clauswilke.com/dataviz/

Multi-panel data

- For multi-panel plots with a lot of information, clarity is especially important.
- Therefore, all panels should use the same axis ranges and scales.

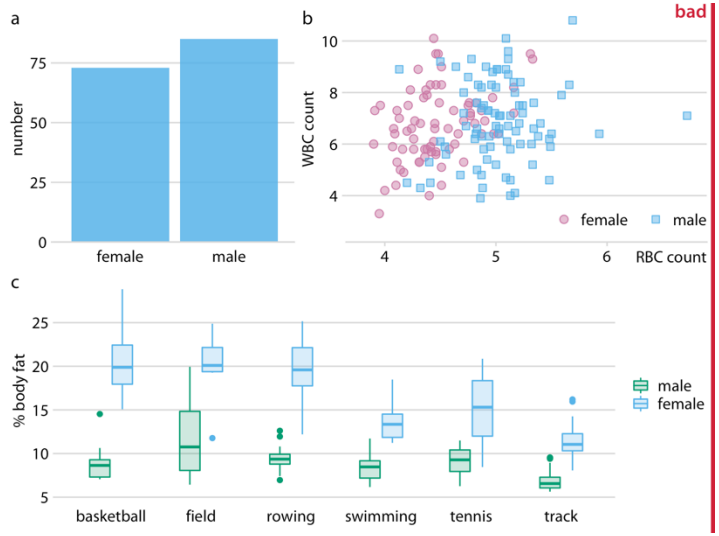


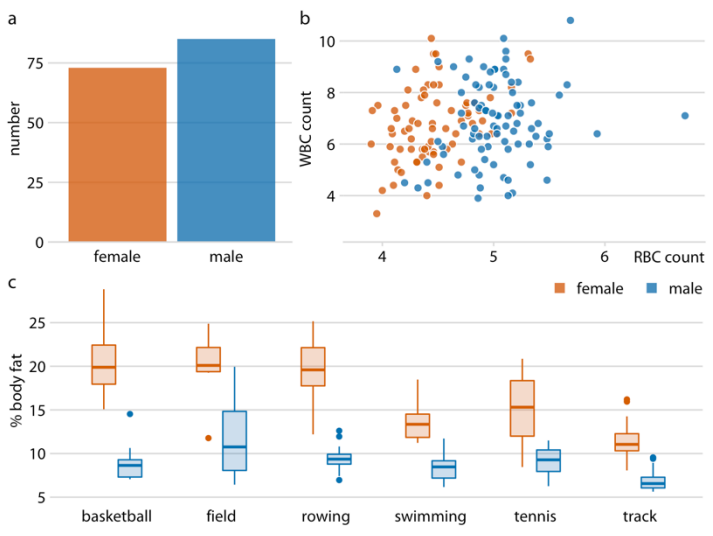


Q: https://clauswilke.com/dataviz/

Consistency with colors

Both plots below show the same data, but the right one is clearer because color is used consistently.





Q: https://clauswilke.com/dataviz/

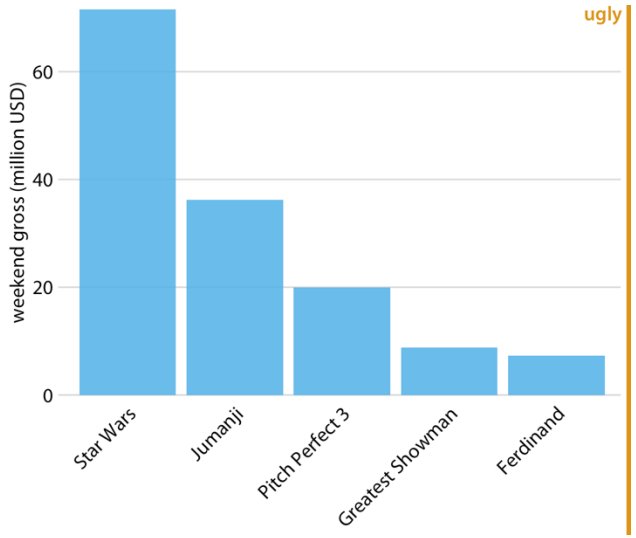
Visualizing amounts

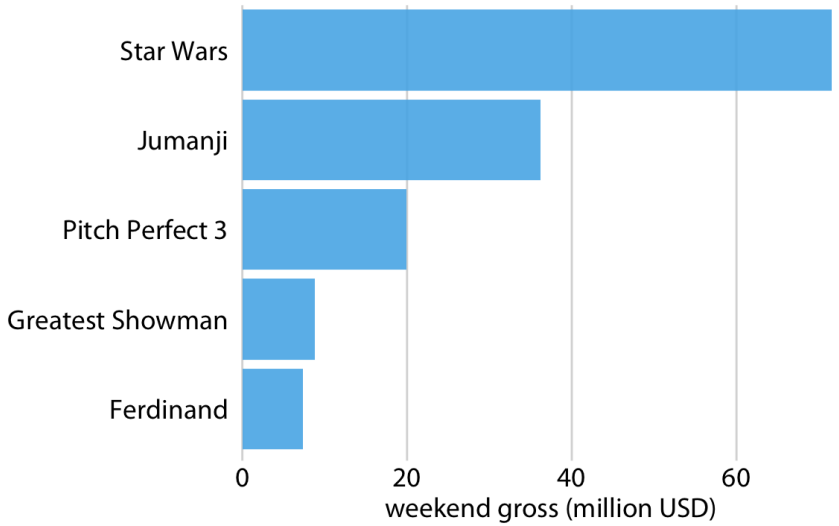
- Visualizing amounts compares quantitative values across categories (e.g., sales by brand, population by city, age by sport).
- Bar plots are well suited for comparing amounts.

Q: https://clauswilke.com/dataviz/

Swapping x- and y-axis

If bar labels take up too much space, it is advisable to swap the x- and y-axes.

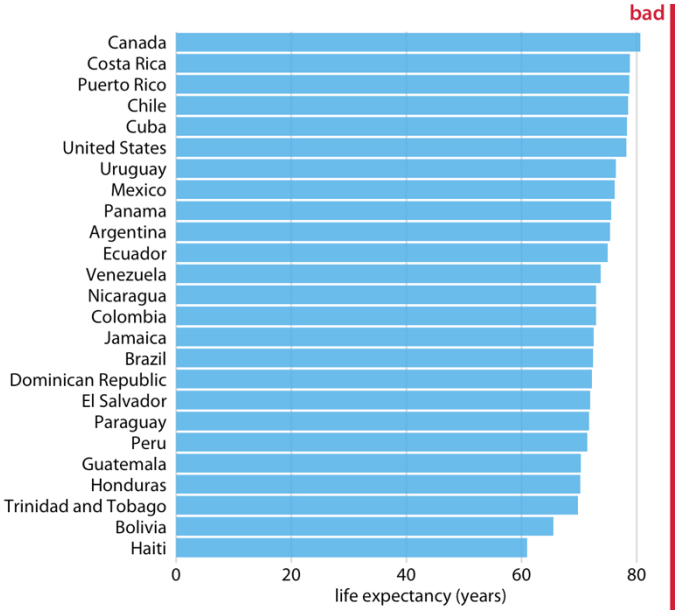


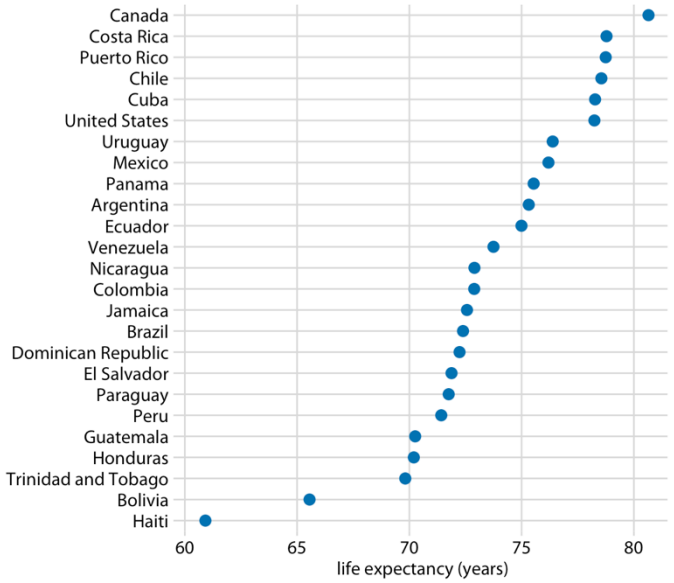


Q: https://clauswilke.com/dataviz/

Dots plot instead of bar chart

If the exact values matter more than the magnitude, a dot plot can be a good alternative.

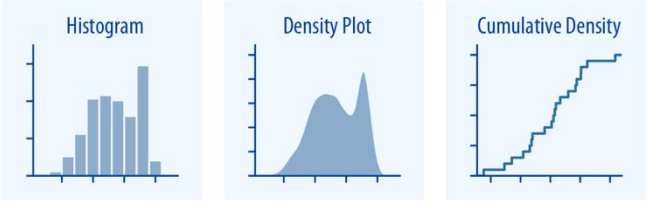




Q: https://clauswilke.com/dataviz/

Visualizing distributions

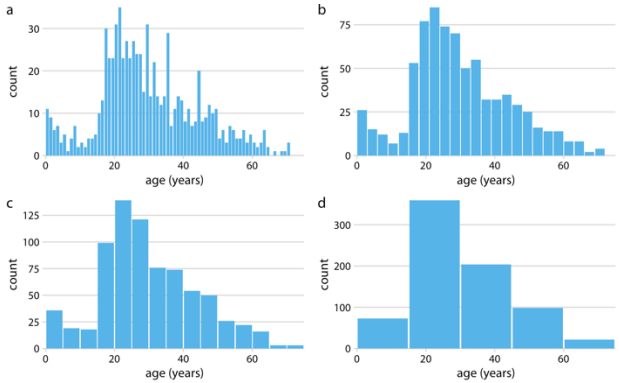
- To understand how (quantitative) data is distributed, use a histogram or a density plot.
- For example, the age distribution of individuals in a dataset can be visualized this way.

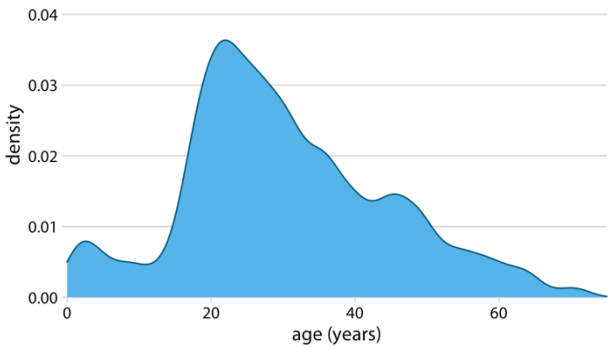


Q: https://clauswilke.com/dataviz/

Bin width and Density Plots

- When creating a histogram, we should try different bin widths to find one that accurately represents the data.
- Alternatively, use a density plot to approximate the underlying distribution.

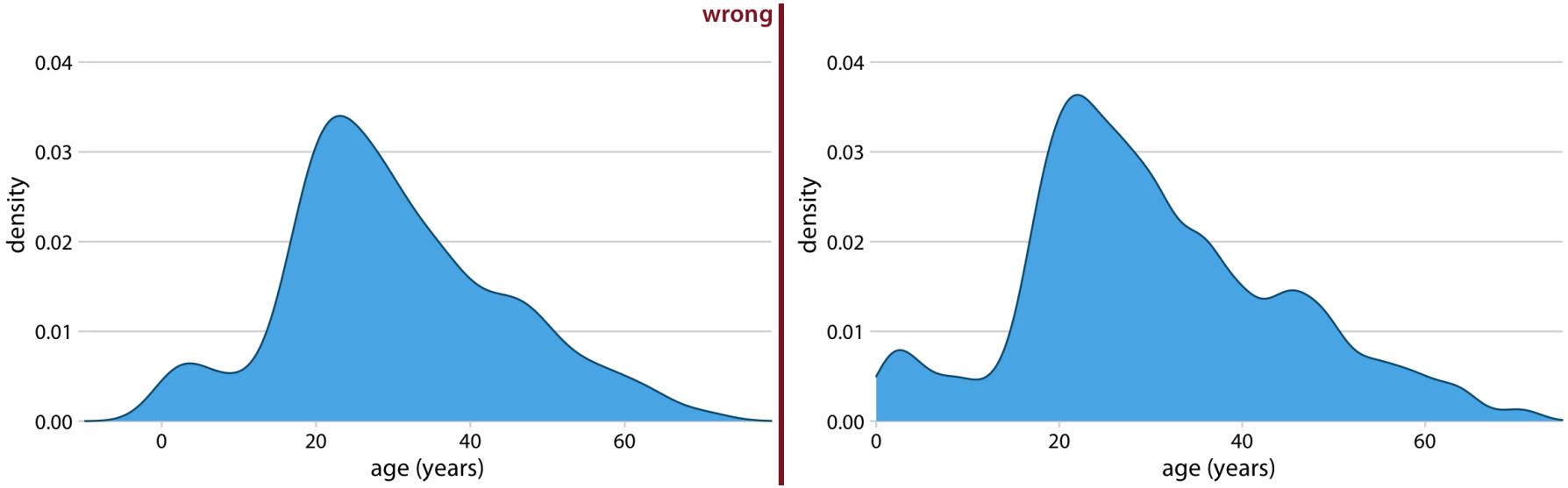




Q: https://clauswilke.com/dataviz/

Density plots

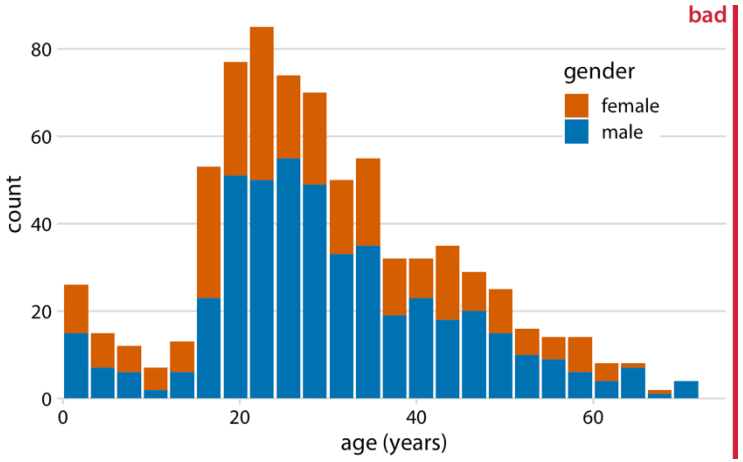
Be aware that density plots can produce data where none exists (here: negative numbers).

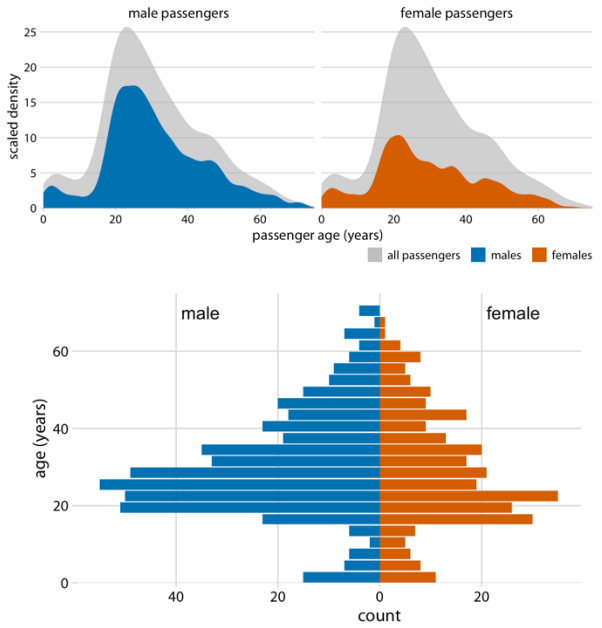


Q: https://clauswilke.com/dataviz/

Comparing distributions

Stacked histograms should be avoided because they obscure where values begin and make groups (here: female) hard to compare.

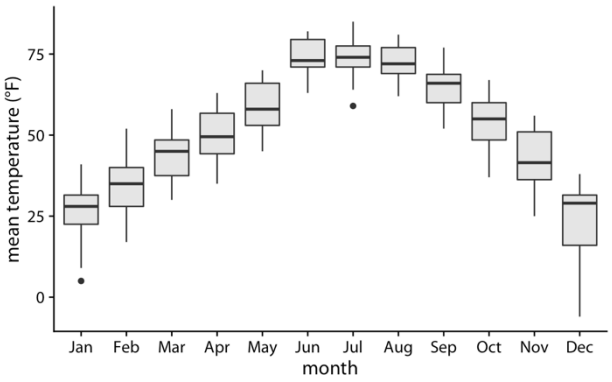


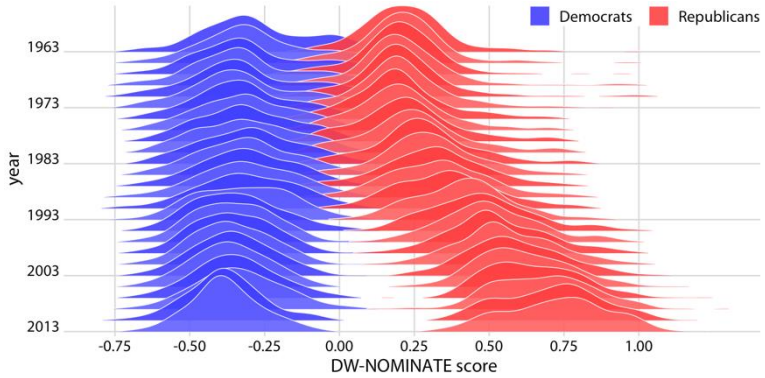


Q: https://clauswilke.com/dataviz/

Visualizing many distributions

Boxplots or violin plots (horizontal direction) and ridgeline plots (vertical direction) scale well for visualizing many distributions at once.

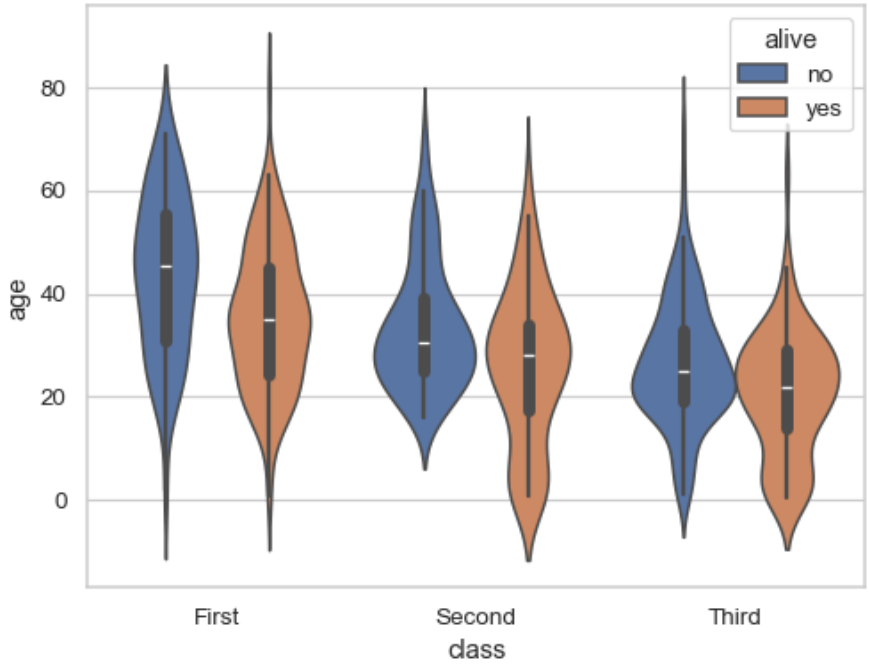




Q: https://clauswilke.com/dataviz/

Violin plot

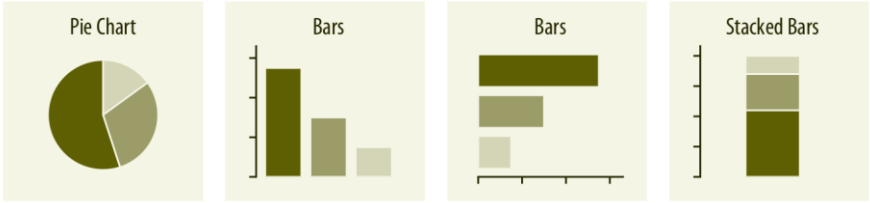
As an alternative to boxplots, we can use violin plots, which combine a boxplot with a density estimate (left and right).



Q: https://seaborn.pydata.org/generated/seaborn.violinplot.html

Visualizing proportions

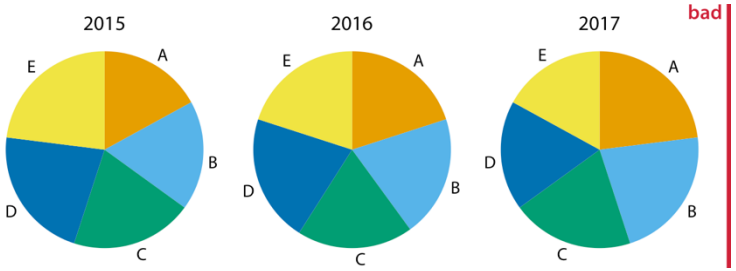
- Visualizing proportions, whether absolute or relative, is important.
- Pie charts are often used for this, although they are not always optimal.

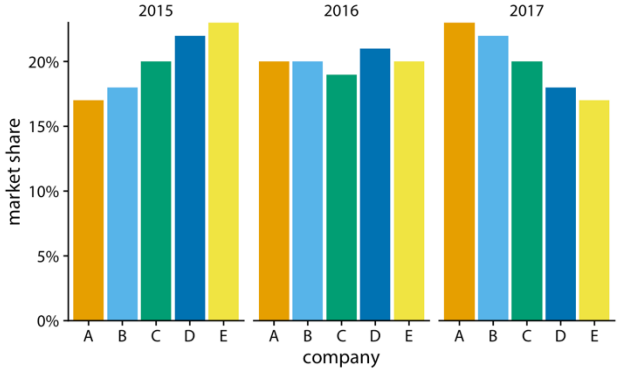


Q: https://clauswilke.com/dataviz/

Depicting trends in pie chars

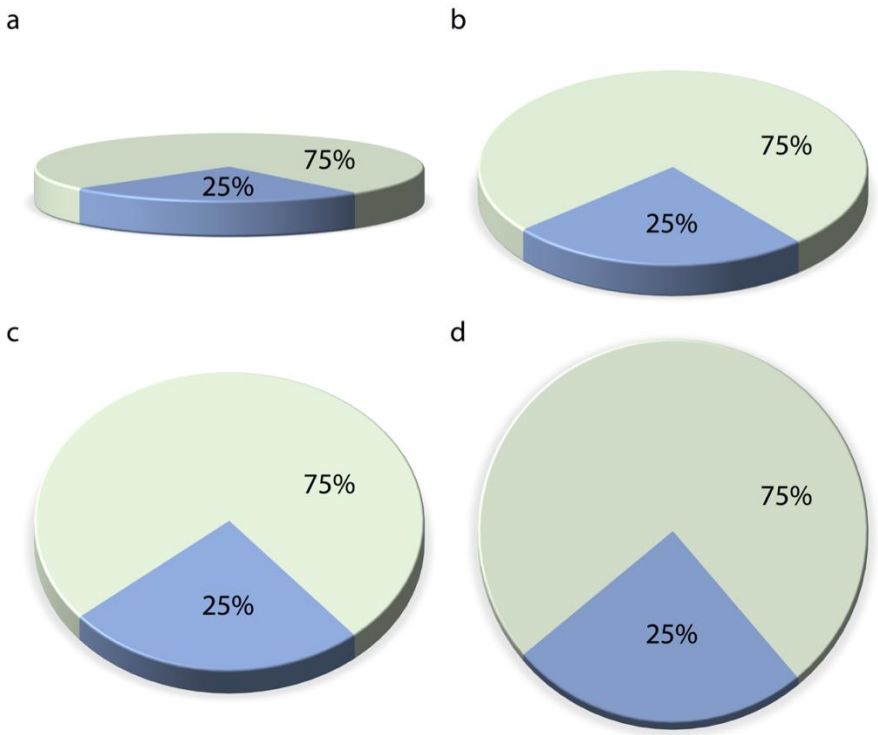
Trends are hard to see in pie charts because their visibility depends on the slice position.





3D plots

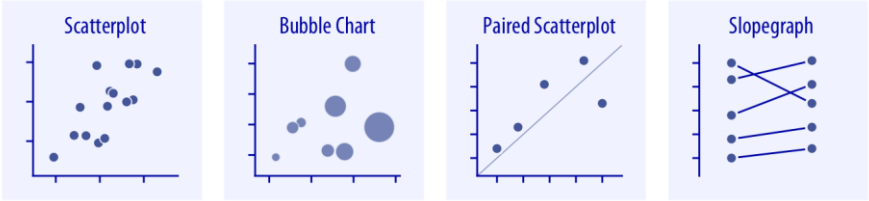
3D plots should mostly be avoided because perspective distorts perception and makes accurate comparison difficult.



Q: https://clauswilke.com/dataviz/

Visualizing associations

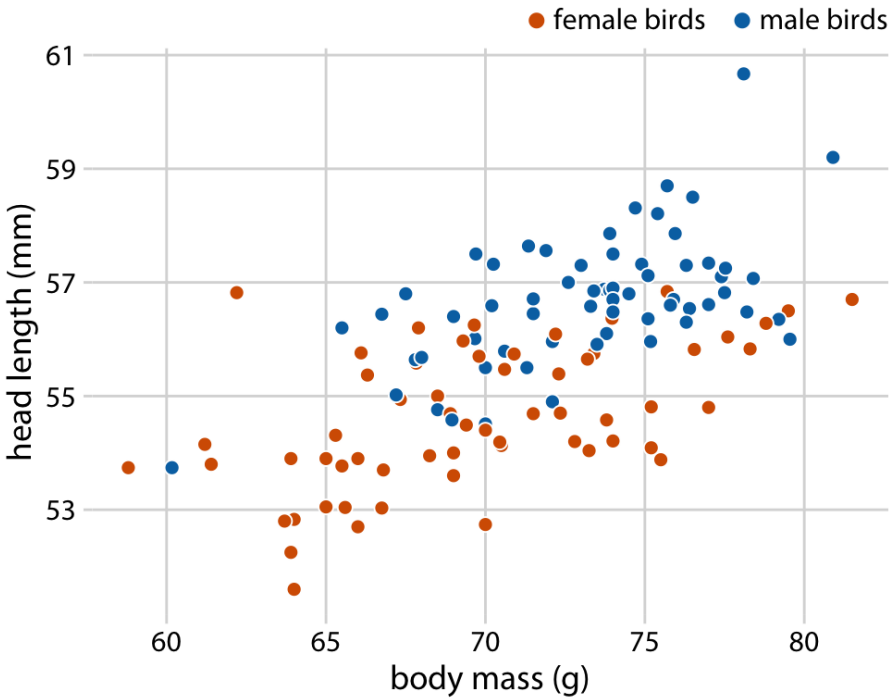
- Most datasets contain more than one quantitative feature.
- To visualize relationships between such feature, scatter plots are used most often.

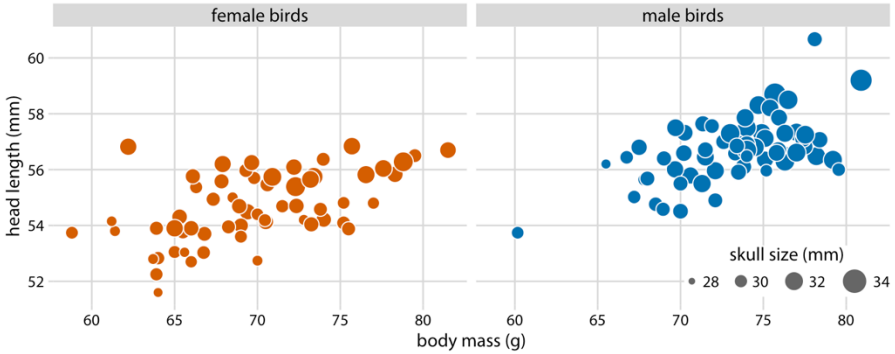


Q: https://clauswilke.com/dataviz/

Bubble Charts

Bubble charts add a third quantitative variable via size.

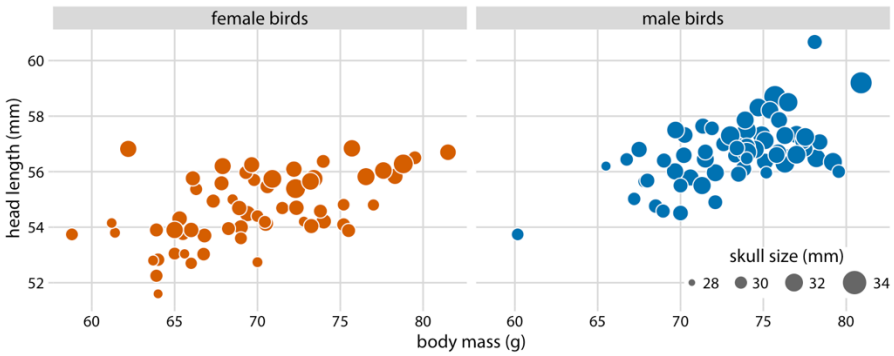


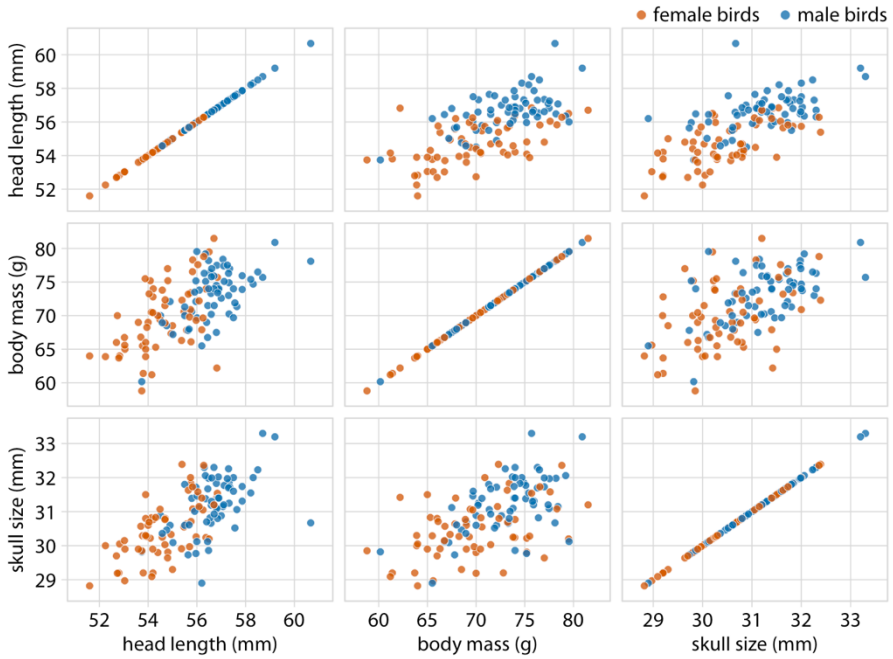


Q: https://clauswilke.com/dataviz/

Bubble Charts

Their disadvantage is that differences in size are hard to compare accurately.

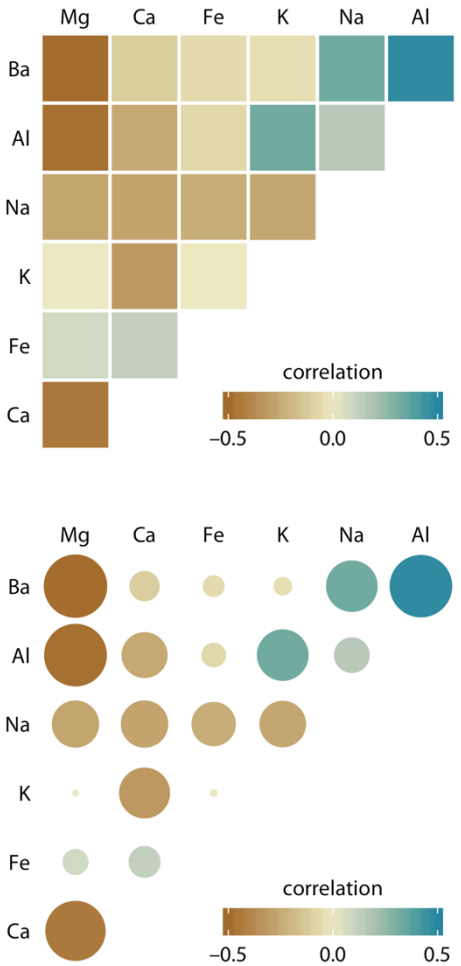




Q: https://clauswilke.com/dataviz/

Correlograms

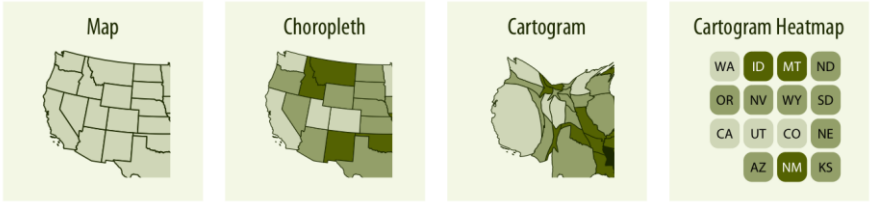
- Correlograms visualize correlation coefficients.
- By using circles and scaling the size accordingly, we can highlight weak or zero correlations.



Q: https://clauswilke.com/dataviz/

Visualizing geospatial data

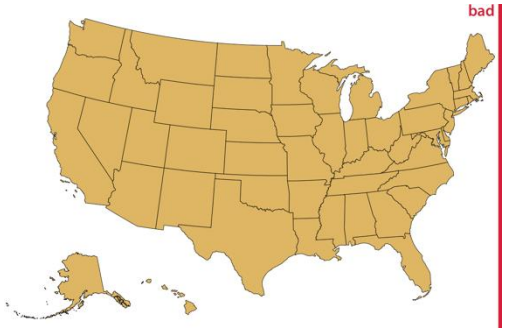
- Geospatial data can be visualized using maps, such as choropleth maps.
- These display data values over a realistic geographic layout.

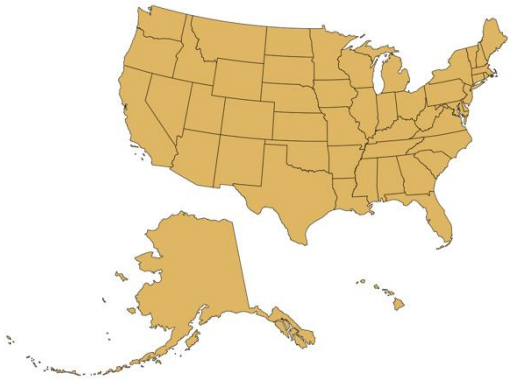


Q: https://clauswilke.com/dataviz/

Limitations with projections

- The Earth is a sphere, so 2D map projections inevitably distort information.
- This can lead to very misleading and inaccurate representations.

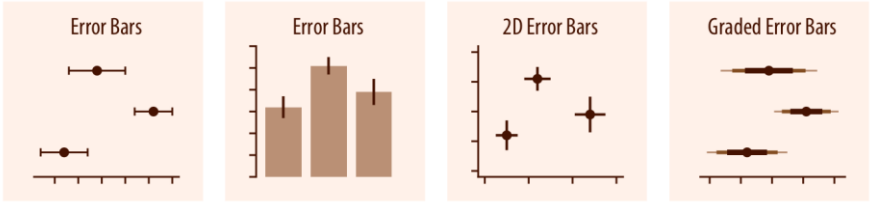




Q: https://clauswilke.com/dataviz/

Visualizing uncertainty

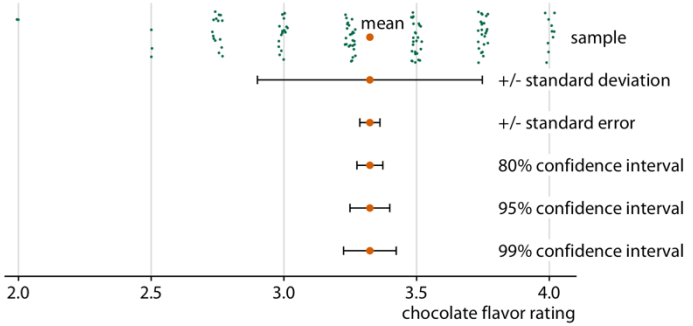
- In many cases, showing only the quantity (e.g., election results) is not sufficient; uncertainty must also be communicated.
- However, uncertainty can be represented in different ways.

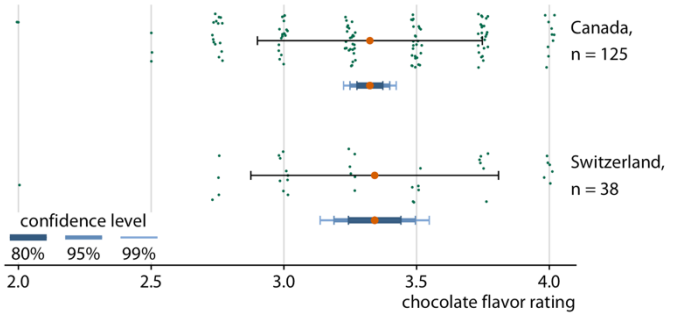


Q: https://clauswilke.com/dataviz/

Different Meaning of error bars

- Uncertainty can be visualized using errors bars.
- Since error bars can have different meanings, specify explicitly what the error bar represents.



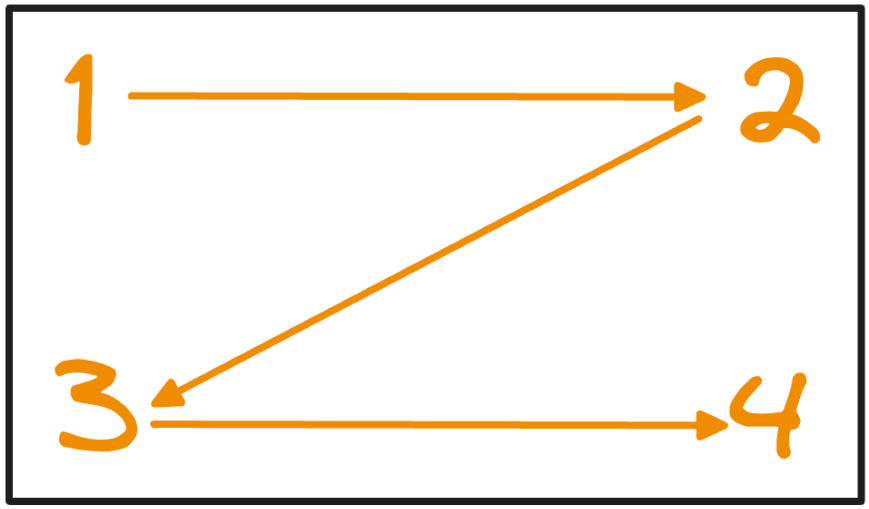


Q: https://clauswilke.com/dataviz/

GUIDELINES FÜR DASHBOARDS

Visual hierarchy

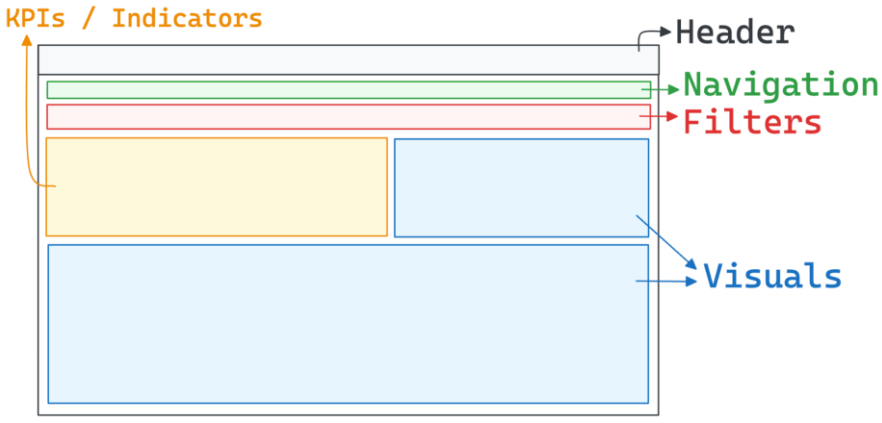
- In addition to general visualization principles, dashboards must consider visual hierarchy.
- This should follow the typical reading direction of a person.

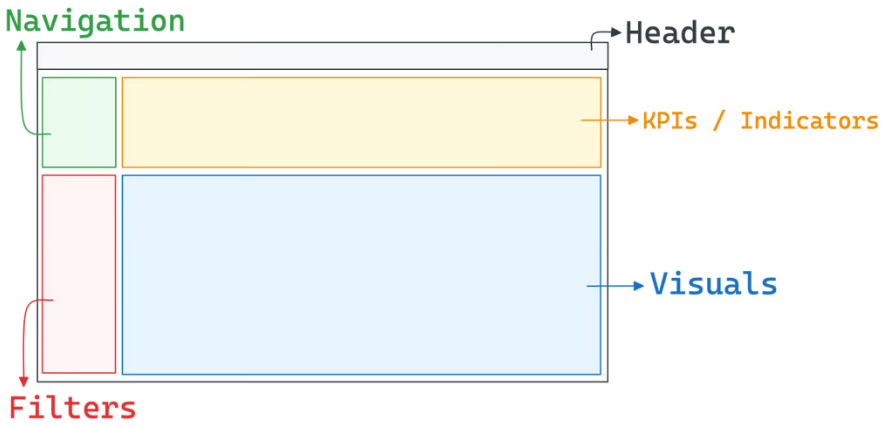


Q: https://www.datacamp.com/tutorial/dashboard-design-tutorial?dc_referrer=https%3A%2F%2Fwww.google.com%2F

Layout

Layout choices also matter, such as top-rail or left-rail designs.





Q: https://www.datacamp.com/tutorial/dashboard-design-tutorial?dc_referrer=https%3A%2F%2Fwww.google.com%2F

Types of Dashboards

Depending on the use case and the audience, there are several types of dashboards.

- Exploratory Dashboards: to give first insights
- Monitoring Dashboards: to visualize real time data
- Analytical Dashboards: to give an overview of completed analysis
- Strategic Dashboards: to offer decisions support

Q: https://www.datacamp.com/tutorial/dashboard-design-tutorial?dc_referrer=https%3A%2F%2Fwww.google.com%2F

STREAMLIT

Python library for building dashboards

streamlit

- Mithilfe von *streamlit* können wir **direkt aus Python-Skripten** (.py-Dateien) **Web-Applikationen** erstellen.
- Wir verwenden *streamlit* hier zur Erstellung von **Dashboards**.
- Dabei benötigen wir zusätzliche externe Bibliotheken wie *altair* und *plotly* für eine ästhetisch ansprechendere und interaktivere Visualisierung von Graphiken.

streamlit

- Wir können uns unser Dashboard nach dem **Baukastenprinzip** erstellen.
- Die verschiedenen Bausteine betrachten wir auf den nächsten Folien.
- Zunächst erstellen wir uns aber ein Python-Projekt.
- Mit folgendem Befehl wird dann das Dashboard aus der *.py*-Datei generiert.

```
streamlit run app.py
```

Text

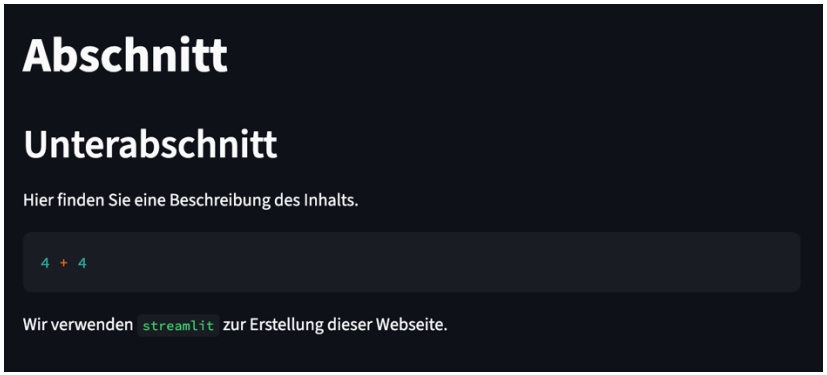
- Zum Schreiben von Text bietet streamlit unterschiedliche Optionen: *st.write*, *st.markdown*, *st.title*
- Im unteren Beispiel verwenden wir die Markup Sprache Markdown.

app.py 1, U

app.py

```python
import streamlit as st

st.markdown(
    """
    # Abschnitt
    ## Unterabschnitt
    Hier finden Sie eine Beschreibung des Inhalts.
    ```python
    4 + 4
    ```
    Wir verwenden `streamlit` zur Erstellung dieser Webseite.
    """
)
```



Tabellen

- Zur Ausgabe von Tabellen können wir folgende Befehle verwenden: *st.dataframe*, *st.table*

app.py 1, U

app.py

```
1  import plotly.express as px
2  import streamlit as st
3
4  df = px.data.gapminder()
5
6  st.markdown("# Gapminder dataset")
7  st.dataframe(df)
8
```


Gapminder dataset

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
0	Afghanistan	Asia	1952	28.801	8425333	779.4453	AFG	4
1	Afghanistan	Asia	1957	30.332	9240934	820.853	AFG	4
2	Afghanistan	Asia	1962	31.997	10267083	853.1007	AFG	4
3	Afghanistan	Asia	1967	34.02	11537966	836.1971	AFG	4
4	Afghanistan	Asia	1972	36.088	13079460	739.9811	AFG	4
5	Afghanistan	Asia	1977	38.438	14880372	786.1134	AFG	4
6	Afghanistan	Asia	1982	39.854	12881816	978.0114	AFG	4
7	Afghanistan	Asia	1987	40.822	13867957	852.3959	AFG	4
8	Afghanistan	Asia	1992	41.674	16317921	649.3414	AFG	4
9	Afghanistan	Asia	1997	41.763	22227415	635.3414	AFG	4

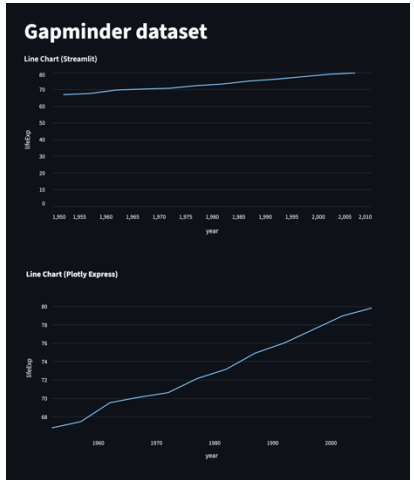
Diagramme

- Zur Erstellung von Diagrammen können wir streamlit Funktionalitäten nutzen oder auf zusätzliche externe Bibliotheken wie *plotly* oder *altair* zurückgreifen: *st.line_chart*, *st.plotly_chart*, *st.altair_chart*

app.py 2, U

app.py

```python
import altair as alt
import plotly.express as px
import streamlit as st

df = px.data.gapminder()
df_filtered = df[df["country"] == "Austria"]

st.markdown("# Gapminder dataset")

st.markdown("###### Line Chart (Streamlit)")
st.line_chart(df_filtered, x="year", y="lifeExp")

fig = px.line(df_filtered, x="year", y="lifeExp", title="Line Chart (Plotly Express)")
st.plotly_chart(fig)
```



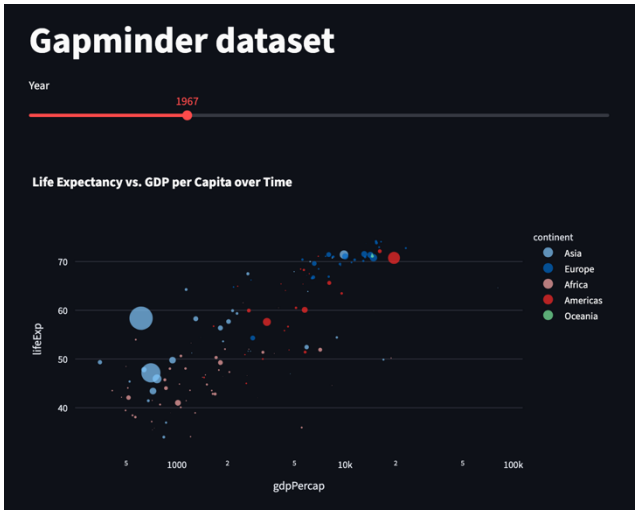
Widgets

- Mithilfe von Widgets lassen sich beispielsweise Diagramme interaktiver gestalten: *st.slider*, *st.selectbox*, *st.button*

app.py 2, U

app.py

```python
import altair as alt
import plotly.express as px
import streamlit as st

df = px.data.gapminder()

st.markdown("# Gapminder dataset")

year = st.slider("Year", min_value=1952, max_value=2007, value=1952, step=5)

fig = px.scatter(
    df.query("year == @year"),
    x="gdpPercap",
    y="lifeExp",
    size="pop",
    color="continent",
    log_x=True,
    title="Life Expectancy vs. GDP per Capita over Time",
)

st.plotly_chart(fig, use_container_width=True)
```



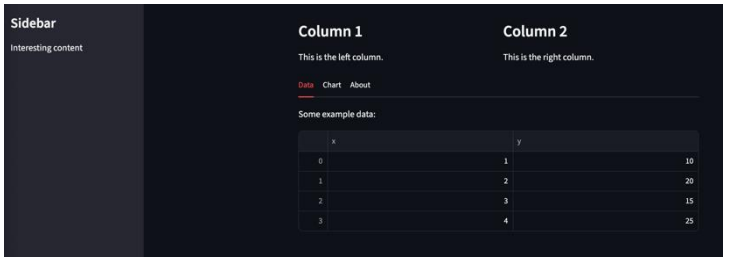
Layout Containers

- Wir können das Dashboard in verschiedene Teilbereiche strukturieren: *st.columns*, *st.sidebar*, *st.tabs*

app.py 1, U

app.py

```python
import pandas as pd
import streamlit as st

st.sidebar.title("Sidebar")
name = st.sidebar.text("Interesting content")

col1, col2 = st.columns(2)

with col1:
    st.subheader("Column 1")
    st.write("This is the left column.")

with col2:
    st.subheader("Column 2")
    st.write("This is the right column.")

tab1, tab2, tab3 = st.tabs(["Data", "Chart", "About"])

with tab1:
    st.write("Some example data:")
    df = pd.DataFrame({"x": [1, 2, 3, 4], "y": [10, 20, 15, 25]})
    st.dataframe(df)

with tab2:
    st.write("A simple line chart:")
    st.line_chart(df, x="x", y="y")

with tab3:
    st.write("This is a simple demo with **columns**, a **sidebar**, and **tabs**.")

```



Mehrere Seiten

- Platzieren wir einzelne *.py*-Dateien in einen Ordner mit dem Namen *pages*, erzeugt uns *streamlit* ein Website mit mehreren verschiedenen Seiten.

```
.
├── app.py
├── archive.txt
├── pages
│   ├── __init__.py
│   ├── 1_Seite 1.py
│   └── 2_Seite 2.py
├── pyproject.toml
├── README.md
└── uv.lock
```

Konfigurationsdatei zum automatischen Update

- Einstellung in `.streamlit/config.toml` zur automatischen Neuausführung der App bei Dateiänderungen.
- Script wird bei jeder Änderung neu geladen.

```
[server]
runOnSave = true
```

Weitere nützliche Features

st.fragment

- Decorator, um Funktionen als unabhängig neu ladbare Fragmente zu definieren.
- Interaktionen innerhalb des Fragments führen nur zu dessen Neuladen, nicht der gesamten App.
- Automatisches, periodisches Neuladen möglich.

Session State

- Ermöglicht das Speichern und Teilen von Variablen über mehrere Reruns hinweg .
- Zustand bleibt persistent, auch bei Interaktionen und Neuladen der App.
- Variablen können gezielt gelesen, gesetzt und verändert werden