

```
In [17]:  
1 def rufe(name, loud=False):  
2     if loud:  
3         print(f"HALLO, {name.upper()}!")  
4     else:  
5         print(f"Hello, {name}!")  
6  
7 rufe("Bob") # Verwendung des Default-Wertes  
8 rufe("Fred", True)
```

```
Hello, Bob!  
HALLO, FRED!
```

Kann man eine Liste mit Argumenten ("positional") übergeben?

```
In [117]:  
1 argumente = ["Bob", True]  
2 rufe ( argumente )      # falsch  
3  
Hallo, ['Bob', True]!
```

```
In [118]:  
1 rufe ( *argumente )  
2 # richtig: Die spezielle  
3 # "Asterisk"-Syntax * "entpackt"  
4 # eine Liste  
5  
HALLO, BOB!
```



## Entpacken von Dictionaries mit dem Asterisk-Operator:

```
In [19]: 1 # Wie war die Funktion "rufe" nochmal deklariert?  
2 # Alternativ: ?rufe (für docstring) und ??rufe (für den kompletten Quelltext)  
3 help(rufe)
```

```
Help on function rufe in module __main__:  
  
rufe(name, loud=False)
```

```
In [189]: 1 # Entpacken mit dem **-Operator  
2 d = {"loud": True, "name": "Fred"}  
3 rufe(**d)  
4 # äquivalent zu: rufe(loud=True,  
5 # name="Fred")
```

```
HALLO, FRED!
```

```
In [340]: 1 # Kann auch mit dem *-Operator  
2 # kombiniert werden:  
3 rufe(*["Fred"], **{"loud":True})
```

```
HALLO, FRED!
```

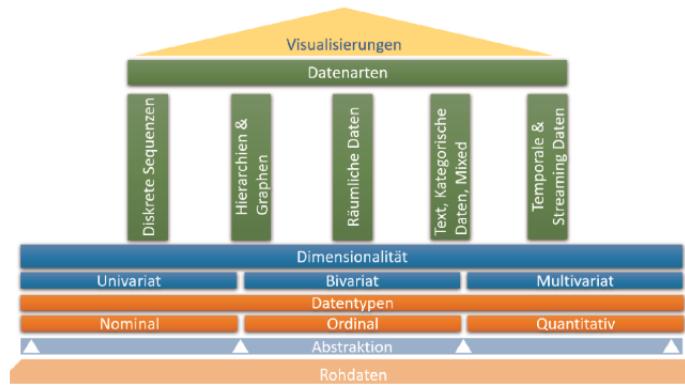


Abb. 3 aus: Nazemi, Kawa & Kaupp, Lukas & Burkhardt, Dirk & Below, Nicola. (2021). Datenvisualisierung. 10.1515/9783110657807-026.

## 1. Visualisierung und Encodings

## 2. DecisionTreeClassifier

## 1.1 Datentyp (=Skala), Datendimensionalität und Datenarten

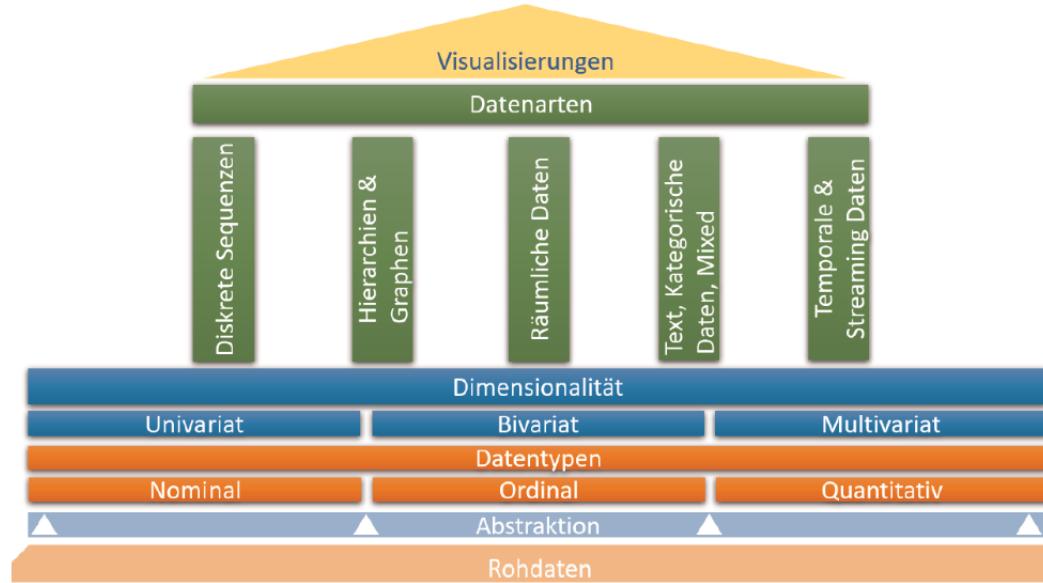


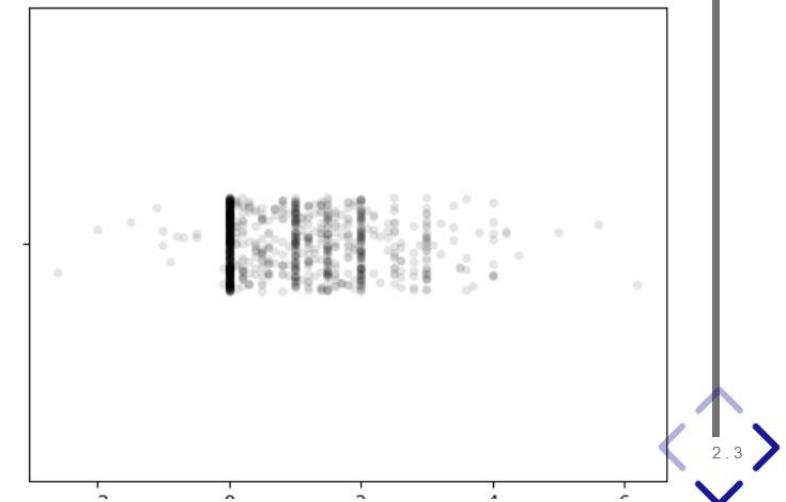
Abb. 3 aus: Nazemi, Kawa & Kaupp, Lukas & Burkhardt, Dirk & Below, Nicola. (2021). Datenvizualisierung. 10.1515/9783110657807-026.

## 1.2 Wiederholung: Visualisierung von numerischen Daten

```
In [44]: 1 # Äquivalent:  
2 # import pandas as pd  
3 # data_oldpeak = pd.read_csv(  
4 #     "VL02_Material/heart.csv")["Oldpeak"].tolist()  
5  
6 with open("VL02_Material/heart_oldpeak.txt","r") as f:  
7     data_oldpeak = f.read() \  
8         .replace("[","") \  
9         .replace("]","",  
10        .split(",")  
11    data_oldpeak = [float(x) for x in data_oldpeak]  
12  
13 data_oldpeak[:5]  
14
```

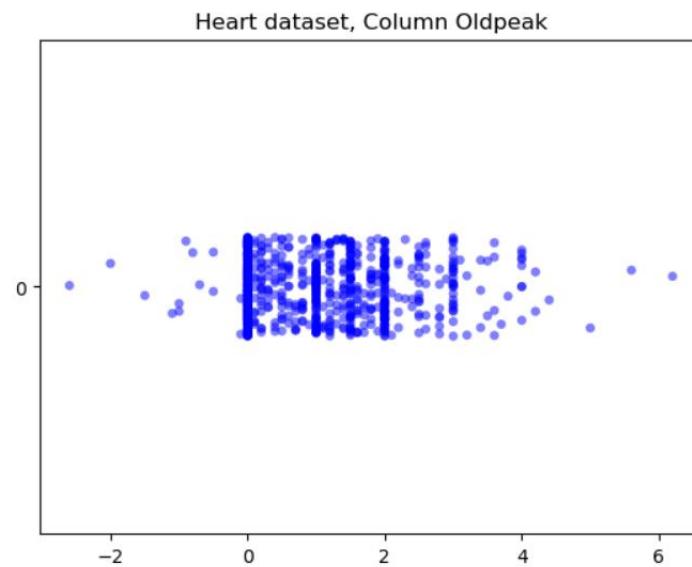
Out[44]: [0.0, 1.0, 0.0, 1.5, 0.0]

```
In [47]: 1 # Strip-Plot mit sinnvollen Default-Argumenten  
2  
3 import seaborn as sns  
4  
5 def mystripplot(data, color="black",  
6                  alpha=1.0, title=""):  
7     ax = sns.stripplot ( data=data, orient="h",  
8                          color=color,  
9                          alpha=alpha )  
10    ax.set( title=title )  
11  
12 mystripplot ( data_oldpeak, alpha=0.1 )  
13
```



In [420]:

```
 1 # Unpacking mit "Asterisk" *
 2
 3 styledict = {"color": "blue",
 4             "alpha": 0.5}      # Dictionary, also eine Sammlung von key/value-pairs
 5
 6 mystripplot ( data_oldpeak, title="Heart dataset, Column Oldpeak", **styledict )
 7 # mit ** wird das Dictionary entpackt und die Inhalte als Keyword-Argumente verwendet
 8
 9 # ... ist das gleiche wie:
10 # mystripplot ( data_oldpeak, title="Heart dataset, Column Oldpeak", color="blue", alpha=0.5 )
11
12 # Frage: Was ist ein Positional Argument? Was ist ein Keyword Argument?
13
```



Der \*-Operator kann auch in der Funktionsdeklaration verwendet werden:

- \*args für beliebig viele Positional-Argumente
- \*\*kwargs für beliebig viele Keyword-Argumente

Achtung: Nach \*args können Argumente nur noch per Keyword angesprochen werden (positional unerreichbar)

```
In [421]: 1 def mystripplot(data, *args, title, **kwargs ):
2     kwargs["s"] = 50 # innerhalb der Funktion ist kwargs ein Dictionary, das die keyword-Argumente enthält
3     ax = sns.stripplot ( data=data, orient="h", **kwargs )
4     ax.set( title=title )
5
6 mystripplot ( data_oldpeak, 123, 456, 789, "Heart dataset, Column Oldpeak", color="blue", alpha=0.5, marker="^" )
7
```

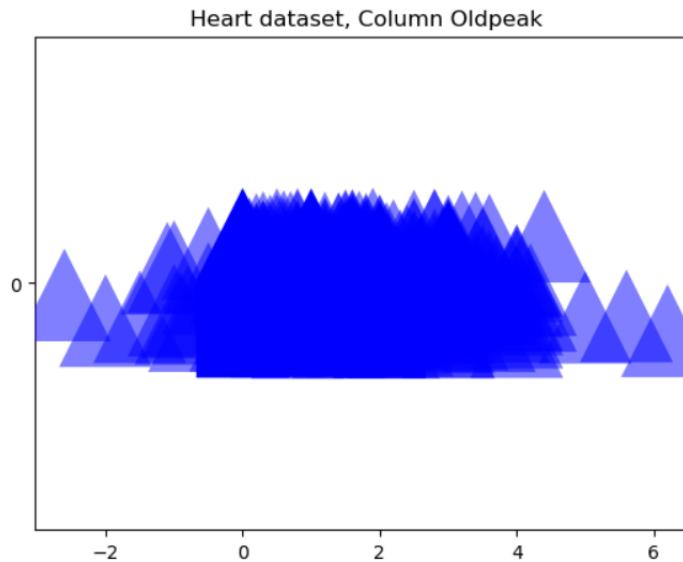
---

```
-----  
TypeError                                     Traceback (most recent call last)
Cell In[421], line 6
      3     ax = sns.stripplot ( data=data, orient="h", **kwargs )
      4     ax.set( title=title )
----> 6 mystripplot ( data_oldpeak, 123, 456, 789, "Heart dataset, Column Oldpeak", color="blue", alpha=0.
      5, marker="^" )

TypeError: mystripplot() missing 1 required keyword-only argument: 'title'
```

In [90]:

```
1 # Nun sprechen wir das title-Argument Argument korrekt via Keyword an:  
2  
3 def mystriplot(data, *args, title, **kwargs):  
4     kwargs["s"] = 50 # innerhalb der Funktion ist kwargs ein Dictionary, das die keyword-Argumente enthält  
5     ax = sns.stripplot ( data=data, orient="h", **kwargs )  
6     ax.set( title=title )  
7  
8 mystriplot ( data_oldpeak, 123, 456, 789, title="Heart dataset, Column Oldpeak", color="blue", alpha=0.5, marker="^" )  
9
```



In `seaborn.stripplot`: Eine Benennung der Argumente wird mit \* erzwungen. Weitere mögliche Argumente für die intern verwendeten low-level-Plotmethoden werden mit \*\*kwargs "durchgereicht".

```
In [ ]: 1 def stripplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None,
2                         jitter=True, dodge=False, orient=None, color=None, palette=None,
3                         size=5, edgecolor='gray', linewidth=0, hue_norm=None,
4                         native_scale=False, formatter=None, legend='auto', ax=None, **kwargs)
5 ...
6     size = kwargs.get("s", size)
7
8     kwargs.update(
9         s=size ** 2,
10        edgecolor=edgecolor,
11        linewidth=linewidth,
12    )
13 ...
14     points = ax.scatter(sub_data["x"], sub_data["y"], color=color, **kwargs)
15 ...
```

<https://github.com/mwaskom/seaborn/blob/f9827a3ae2a998aeeb97d1931ae03f8a3f7109c2/seaborn/categorical.py> (Z. 416, 2618, 2658)

**Frage:** Was passiert in Zeile 6?

Zur Angabe der Punktgröße verwendet `seaborn.stripplot` den Parameter `size`. Daneben gibt es noch den low-level-Parameter `s` der unterliegenden Matplotlib-API. In Zeile 6 wird die Variable `size` durch `s` überschrieben, falls letztere angegeben wurde (d.h. vorhanden ist im Dictionary `kwargs`).

## 1.3 Plotting mit Matplotlib

(Grundlage für Pandas und Seaborn)

- Konvention: `import matplotlib.pyplot as plt`
- Basisobjekte `Figure` und `Axes`, z.B. via  
`fig, axes = plt.subplots (`  
 `figsize=(5,2), nrows=2, ncols=2 )`
- "implicit" interface: Einfache Plot-Funktionalität via `plt`
- "explicit" interface: Feinere Steuermöglichkeit via `ax`

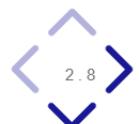
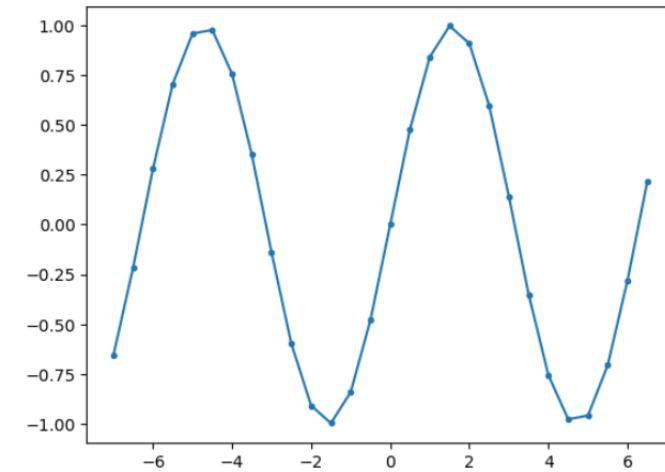
### Healthcare Data Analytics

Einführung in

In [53]:  
1 # Erzeuge Daten (später mehr zu numpy)  
2 import numpy as np  
3 x = np.arange (-7, 7, 0.5)  
4 y = np.sin(x) # vektorielles Rechnen, Ergebnis ist  
5 # ein Vektor von Funktionswerten  
6 x[:5]  
7

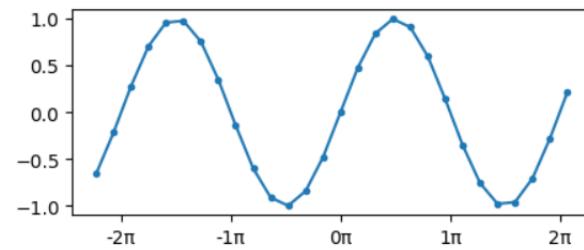
Out[53]: `array([-7. , -6.5, -6. , -5.5, -5. ])`

In [56]:  
1 # implizites Interface  
2 import matplotlib.pyplot as plt  
3 plt.plot ( x, y, marker="." );

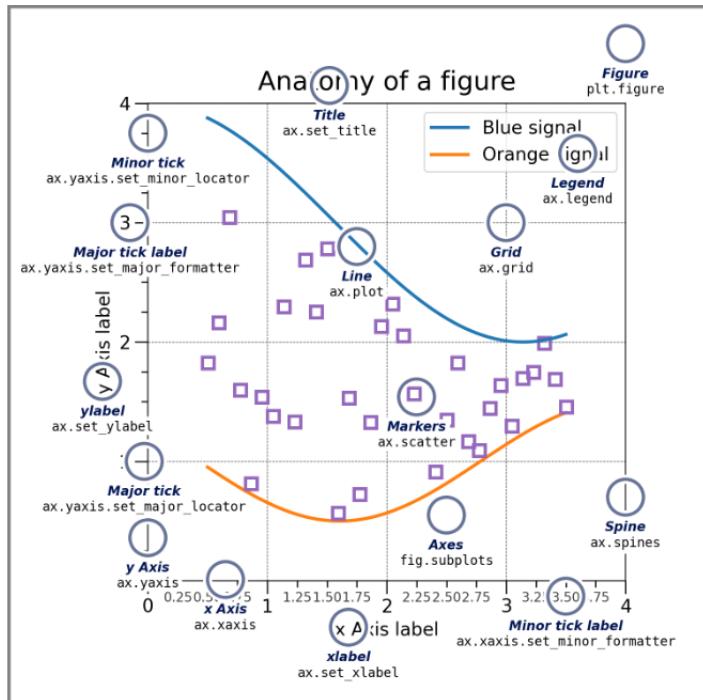


In [55]:

```
1 # explizites Interface: nicht möglich im Interface plt.plot ( x,y )
2 fig, ax = plt.subplots ( figsize=(5,2) )
3 ax.plot ( x, y, marker="." );
4 ax.set_xticks ( [i*np.pi for i in range(-2,3)] )
5 ax.set_xticklabels ( [f"{i}\u03c0" for i in range(-2,3)] );
```



## Vokabular:



[https://matplotlib.org/stable/tutorials/introductory/quick\\_start.html#sphx-glr-tutorials-introductory-quick-start-py](https://matplotlib.org/stable/tutorials/introductory/quick_start.html#sphx-glr-tutorials-introductory-quick-start-py)

## Grundlegende Plotbefehle:

- plt.scatter / ax.scatter: Punkte
- plt.plot / ax.plot: Linien

## Beschriftungen:

- plt.xlabel / ax.set\_xlabel (ebenso für y)
- plt.xticks / ax.set\_xticks (ebenso für y)
- - / ax.set\_xticklabels (ebenso für y)
- plt.title / ax.set\_title: Titel je Plot
- plt.suptitle / -: Titel über Subplots

## Skalierung und Formatierung:

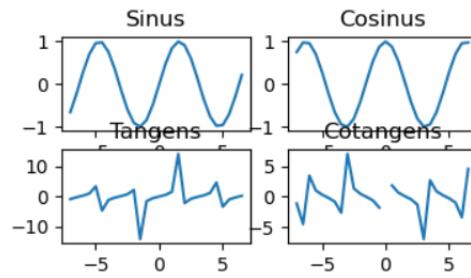
- plt.xlim / ax.set\_xlim (ebenso für y)
- plt.tight\_layout / -: Subplot-Abstände

Ein Beispiel für einen Plot mit Subplots (weiterhin `x=np.arange(-7,7,0.1)`):

```
In [58]: 1 fig, axes = plt.subplots ( figsize=(4,2), nrows=2, ncols=2 ) # Rückgabewert axes ist ein (Numpy) Array
2 # Plot Links oben
3 axes[0,0].plot ( x, np.sin(x) ); axes[0,0].set_title ( "Sinus" )
4
5 # Plot rechts oben
6 axes[0,1].plot ( x, np.cos(x) ); axes[0,1].set_title ( "Cosinus" )
7
8 # Plot Links unten
9 axes[1,0].plot ( x, np.tan(x) ); axes[1,0].set_title ( "Tangens" )
10
11 # Plot rechts unten
12 axes[1,1].plot ( x, 1/np.tan(x) ); axes[1,1].set_title ( "Cotangens" )
13
```

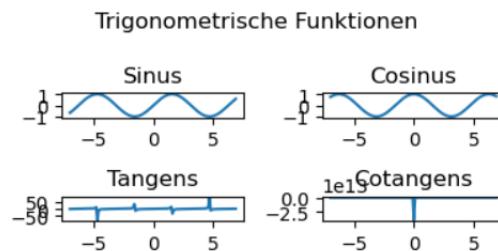
```
C:\Users\schirmef\AppData\Local\Temp\ipykernel_12608\4138917434.py:12: RuntimeWarning: divide by zero encou
ntered in true_divide
    axes[1,1].plot ( x, 1/np.tan(x) ); axes[1,1].set_title ( "Cotangens" )
```

Out[58]: Text(0.5, 1.0, 'Cotangens')



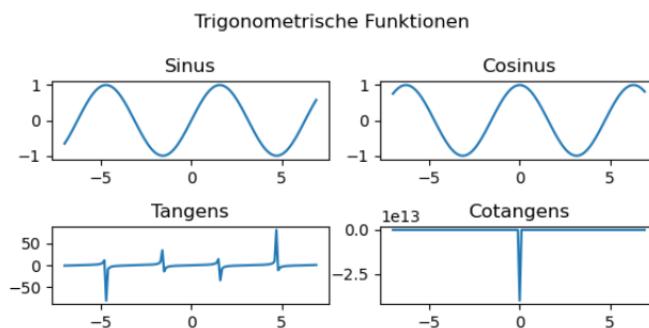
Korrigiere die Abstände zwischen den Subplots mit `plt.tight_layout()` und füge eine Überschrift hinzu:

```
In [17]: 1 fig, axes = plt.subplots ( figsize=(4,2), nrows=2, ncols=2 ) # Rückgabewert axes ist ein (Numpy) Array
2 # Plot Links oben
3 axes[0,0].plot ( x, np.sin(x) ); axes[0,0].set_title ( "Sinus" )
4
5 # Plot rechts oben
6 axes[0,1].plot ( x, np.cos(x) ); axes[0,1].set_title ( "Cosinus" )
7
8 # Plot Links unten
9 axes[1,0].plot ( x, np.tan(x) ); axes[1,0].set_title ( "Tangens" )
10
11 # Plot rechts unten
12 axes[1,1].plot ( x, 1/np.tan(x) ); axes[1,1].set_title ( "Cotangens" )
13
14 plt.suptitle("Trigonometrische Funktionen")
15 plt.tight_layout( );
16
```



Vereinfachung des Codes (damit wir den Plot in den Slides größer machen können) mittels `zip`:

```
In [20]: 1 fig,axes = plt.subplots(figsize=(6,3),nrows=2,ncols=2)
2
3 for ax, f, title in zip( axes.flatten(),
4                           [np.sin, np.cos, np.tan,
5                            lambda x: 1/np.tan(x)],
6                           ["Sinus", "Cosinus",
7                            "Tangens", "Cotangens"] ):
7
8     ax.plot ( x, f(x) )
9     ax.set_title ( title )
10
11 plt.suptitle ( "Trigonometrische Funktionen" )
12 plt.tight_layout ( );
13
```



```
In [21]: 1 # Rank 2-Array
2 a = np.array ( [[1,2],[3,4]] )
3 a
4
```

Out[21]: array([[1, 2],  
[3, 4]])

```
In [22]: 1 # Mit .flatten wird ein höherdimensionales Array
2 # in ein Array ohne zusätzliche Dimensionen umge-
3 # wandelt
4 a.flatten()
5
```

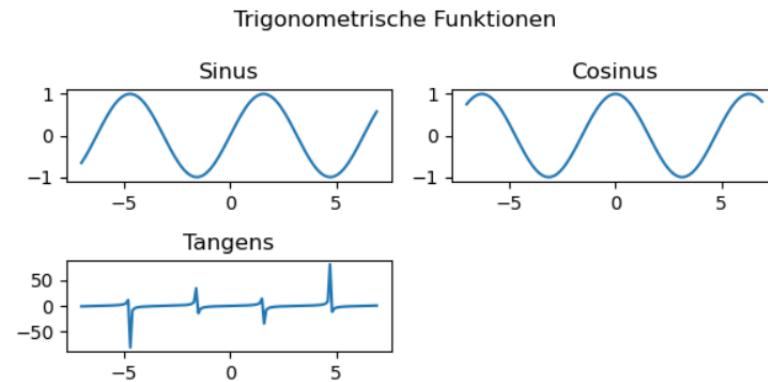
Out[22]: array([1, 2, 3, 4])

```
In [28]: 1 # zip zum Verknüpfen von Listen gleicher Länge
2 list ( zip([1,2,3], [4,5,6]) )
3
```

Out[28]: [(1, 4), (2, 5), (3, 6)]

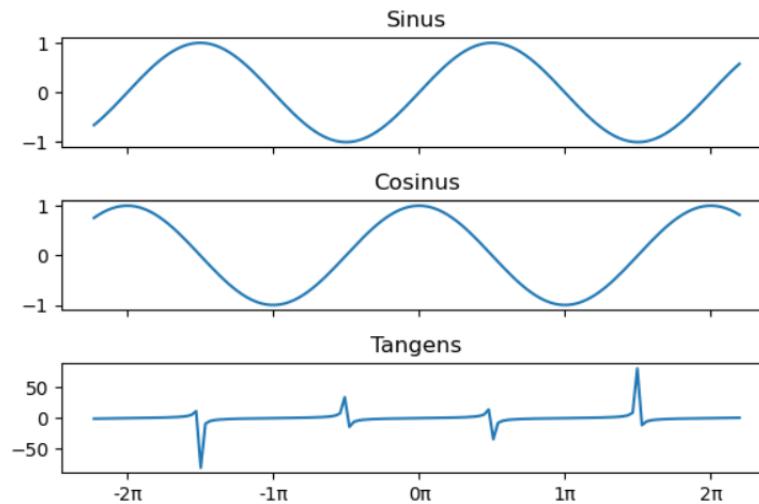
Entfernen eines Plots ("Cotangens ist ja nur 1/Tangens"):

```
In [40]: 1 fig,axes = plt.subplots ( figsize=(6,3), nrows=2, ncols=2)
2
3 for ax, f, title in zip ( axes.flatten()[:-1], [np.sin, np.cos, np.tan], ["Sinus", "Cosinus", "Tangens"] ):
4     ax.plot ( x, f(x) )
5     ax.set_title ( title )
6
7 axes[1,1].axis("off")
8
9 plt.suptitle ( "Trigonometrische Funktionen" )
10 plt.tight_layout ( );
11
```



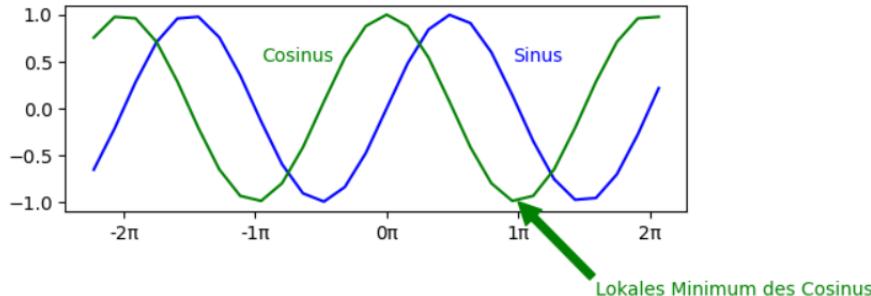
Anordnung in einer Zeile oder Spalte ist sinnvoller. Verwendung von `sharex` und `set_xticklabels`:

```
In [52]: 1 fig,axes = plt.subplots ( figsize=(6,4), nrows=3, ncols=1, sharex=True )
2
3 for ax, f, title in zip ( axes, [np.sin, np.cos, np.tan], ["Sinus", "Cosinus", "Tangens"] ):
4     ax.plot ( x, f(x) )
5     ax.set_title ( title )
6
7 axes[0].set_xticks ( [i*np.pi for i in range(-2,3)] )
8 axes[0].set_xticklabels ( [f"{i}\pi" for i in range(-2,3)] );
9
10 plt.tight_layout ( );
11
```



## Abschließend: Text und Annotations.

```
In [60]: 1 fig,ax = plt.subplots ( figsize=(6,2) )
2
3 ax.plot ( x, np.sin(x), color="blue" )
4 ax.plot ( x, np.cos(x), color="green" )
5 ax.set_xticks ( [i*np.pi for i in range(-2,3)] )
6 ax.set_xticklabels ( [f"{i}\u03c0" for i in range(-2,3)] );
7
8 ax.text ( s="Sinus", x=3, y=0.5, color="blue" );
9 ax.text ( s="Cosinus", x=-3, y=0.5, color="green" );
10 # Standardmäßig werden die Koordinaten im "Datenkoordinatensystem" angegeben.
11 # Mit transform=ax.transAxes können die Koordinaten auch relativ zum Plotfenster angegeben werden; (0,0) ist dann "Links unten".
12
13 ax.annotate ( "Lokales Minimum des Cosinus", xy=(np.pi,-1.0), xytext=(5,-2),
14                 arrowprops=dict(color="green"), color="green" );
15
```



## 1.4 Werteskalen

```
In [8]: 1 import pandas as pd
2 df = pd.read_csv("VL02_Material/heart.csv")
3 df
```

```
Out[8]:   Age  Sex ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG
0    40    M      ATA          140        289          0       Normal
1    49    F      NAP          160        180          0       Normal
2    37    M      ATA          130        283          0        ST
3    48    F      ASY          138        214          0       Normal
4    54    M      NAP          150        195          0       Normal
...
913  45    M      TA           110        264          0       Normal
914  68    M      ASY          144        193          1       Normal
915  57    M      ASY          130        131          0       Normal
916  57    F      ATA          130        236          0       LVH
917  38    M      NAP          138        175          0       Normal
```

918 rows × 12 columns

Daten können in verschiedenen Skalen vorliegen:

**Kategoriale Skalen** (qualitativ, "kategoriale Features")

- **Nominalskala:** keine Ordnung vorhanden,  
z.B. {True, False}, {rot,grün,blau}
- **Ordinalskala:** Eine Ordnung ist sinnvoll,  
z.B. {schlecht, mittel, gut}

**Kardinale Skalen** (quantitativ, "numerische Features")

(kann noch unterteilt werden in Intervallskala und Verhältnisskala)

**Frage:** Finden Sie im heart-Datensatz Beispiele für die verschiedenen Skalen?

```
In [86]: 1 df.info()  
2  
3  
4
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 918 entries, 0 to 917  
Data columns (total 12 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          -----          -----  
 0   Age          918 non-null    int64    
 1   Sex          918 non-null    object    
 2   ChestPainType 918 non-null  object    
 3   RestingBP     918 non-null  int64    
 4   Cholesterol   918 non-null  int64    
 5   FastingBS     918 non-null  int64    
 6   RestingECG    918 non-null  object    
 7   MaxHR         918 non-null  int64    
 8   ExerciseAngina 918 non-null object    
 9   Oldpeak       918 non-null  float64  
 10  ST_Slope       918 non-null  object    
 11  HeartDisease  918 non-null  int64    
dtypes: float64(1), int64(6), object(5)  
memory usage: 86.2+ KB
```

**Kardinale Skala / numerische Features:**  
z.B. Age, Cholesterol, MaxHR, Oldpeak

**Kategoriale Skala / kategorische Features:**

- Nominalskala: z.B. Sex, ChestPainType, HeartDisease
- Ordinalskala: z.B. ST\_Slope (?)

```
In [89]: 1 df["ST_Slope"].unique()  
2
```

Out[89]: array(['Up', 'Flat', 'Down'], dtype=object)

**Problem:** Machine Learning Verfahren können i.A. nur mit numerischen Repräsentationen der Daten umgehen.

Sei  $M$  eine diskrete Menge von kategorischen Werten,  
dann ist das **Label Encoding** eine injektive Funktion  
 $f : M \rightarrow \mathbb{N}$ .

Beispiel:  $M = \{\text{schlecht, mittel, gut}\}$

Label	Label Encoding
schlecht	1
mittel	2
gut	3

Label Encoding kann für **Ordinale Daten** verwendet werden,  
da diese eine natürliche Reihenfolge aufweisen.

injektiv bedeutet, dass nie zwei unterschiedliche  $x, y$  auf das gleiche  $z$  abgebildet werden.

Sei  $M$  eine diskrete Menge mit  $n$  kategorischen Werten,  
dann ist das **One-Hot Encoding** eine injektive Funktion  
 $f : M \rightarrow \{0, 1\}^n$ .

Beispiel:  $M = \{\text{Apfel, Birne, Zitrone}\}$

Apfel	Birne	Zitrone
1	0	0
0	1	0
0	0	1

One-Hot Encoding sollte für **Nominale Daten** verwendet werden, da diese *keine* natürliche Reihenfolge aufweisen.  
Auch für Ordinale Daten verwendbar.

## Encoding mit Pandas:

```
In [92]: 1 # Standard-Konversion zu Kategorischen Werten;  
2 # Nutzen: sparsame Darstellung der Daten.  
3 # Problem: Werte sind nicht korrekt angeordnet  
4  
5 df["ST_Slope"].astype("category")  
6
```

```
Out[92]: 0      Up  
1      Flat  
2      Up  
3      Flat  
4      Up  
...  
913    Flat  
914    Flat  
915    Flat  
916    Flat  
917    Up  
Name: ST_Slope, Length: 918, dtype: category  
Categories (3, object): ['Down', 'Flat', 'Up']
```

```
In [93]: 1 # Die Ordnung der Werte kann spezifiziert werden  
2  
3 from pandas.api.types import CategoricalDtype  
4 cat_type = CategoricalDtype(  
5     categories=["Down", "Flat", "Up"], ordered=True)  
6  
7 df["ST_Slope"].astype(cat_type)  
8
```

```
Out[93]: 0      Up  
1      Flat  
2      Up  
3      Flat  
4      Up  
...  
913    Flat  
914    Flat  
915    Flat  
916    Flat  
917    Up  
Name: ST_Slope, Length: 918, dtype: category  
Categories (3, object): ['Down' < 'Flat' < 'Up']
```

## Encoding mit Pandas:

```
In [98]: 1 # Label Encoding
2 # (Alternative: sklearn.preprocessing.OrdinalEncoder)
3
4 df[ "ST_Slope" ].astype( cat_type ).cat.codes
5
```

```
Out[98]: 0      2
1      1
2      2
3      1
4      2
..
913    1
914    1
915    1
916    1
917    2
Length: 918, dtype: int8
```

```
In [62]: 1 # One-Hot-Encoding
2 # (Alternative: sklearn.preprocessing.OneHotEncoder)
3
4 pd.get_dummies ( df[ "ChestPainType" ] )
5
```

```
Out[62]:   ASY  ATA  NAP  TA
0      0   1   0   0
1      0   0   1   0
2      0   1   0   0
3      1   0   0   0
4      0   0   1   0
...
913    0   0   0   1
914    1   0   0   0
915    1   0   0   0
916    0   1   0   0
917    0   0   1   0
```

918 rows × 4 columns

**One-Hot-Encoding mit pandas** (Achtung: Die Originaldaten bleiben unverändert; der Rückgabewert muss händisch aufgefangen und in die Daten integriert werden):

```
In [11]: 1 pd.get_dummies ( df[ "Sex" ] )  
2
```

Out[11]:

	F	M
0	0	1
1	1	0
2	0	1
3	1	0
4	0	1
...	...	...
913	0	1
914	0	1
915	0	1
916	1	0
917	0	1

918 rows × 2 columns

```
In [12]: 1 pd.get_dummies ( df[ "Sex" ], drop_first=True )  
2 # Parameter drop_first=True,  
3 # um n-1 statt n Spalten zu erhalten  
4 # (eliminiert Lineare Abhängigkeit der Spalten)  
5
```

Out[12]:

	M
0	1
1	0
2	1
3	0
4	1
...	...
913	1
914	1
915	1
916	0
917	1

918 rows × 1 columns

## Label Encoding mit sklearn:

```
In [66]: 1 # Der Vorteil der sklearn-Funktionen liegt darin, dass diese eine getrennte
2 # .fit und .transform sowie eine .inverse_transform-Funktionalität anbieten.
3 #   .fit / .fit_transform: Wird auf den Trainingsdaten angewendet
4 #   .transform: Wird auf den Testdaten angewendet
5
6 from sklearn.preprocessing import OrdinalEncoder
7 enc = OrdinalEncoder ( categories=[["Apfel","Birne","Zitrone"]], handle_unknown="use_encoded_value", unknown_value=-1 )
8 enc.fit_transform ( [["Zitrone"],["Apfel"],["Apfel"],["Birne"]] )
9
```

```
Out[66]: array([[2.],
   [0.],
   [0.],
   [1.]])
```

```
In [67]: 1 # Anwendung auf neue Daten ("Pflaume" wurde während des Trainings nicht gesehen)
2 enc.transform ( [["Zitrone"],["Pflaume"]] )
3
```

```
Out[67]: array([[ 2.],
   [-1.]])
```

```
In [121]: 1 enc.inverse_transform ( [[2],[1],[0],[-1]] )
2
```

```
Out[121]: array([('Zitrone'],
 ['Birne'],
 ['Apfel'],
 [None]), dtype=object)
```

## One-Hot-Encoding mit sklearn:

```
In [20]: 1 from sklearn.preprocessing import OneHotEncoder  
2 enc = OneHotEncoder()  
3 enc.fit_transform([[ "Zitrone"],[ "Apfel"],[ "Apfel"],[ "Birne"]])  
4
```

```
Out[20]: <4x3 sparse matrix of type '<class 'numpy.float64'>'  
with 4 stored elements in Compressed Sparse Row format>
```

```
In [22]: 1 enc = OneHotEncoder(sparse_output=False)  
2 enc.fit_transform([[ "Zitrone"],  
3                   [ "Apfel"],  
4                   [ "Apfel"],  
5                   [ "Birne"]]) # doppelte Klammer, weil man mehrere Spalten gleichzeitig one-hot-encoden könnte
```

```
Out[22]: array([[0., 0., 1.],  
                 [1., 0., 0.],  
                 [1., 0., 0.],  
                 [0., 1., 0.]])
```

```
In [25]: 1 enc.categories_
```

```
Out[25]: [array(['Apfel', 'Birne', 'Zitrone'], dtype=object)]
```

```
In [71]: 1 # Anwendung auf neue Daten ("Pflaume" wurde während des Trainings nicht gesehen)  
2 # => Fehlermeldung; verwende stattdessen handle_unknown="ignore" im Konstruktor  
3 enc.transform([[ "Zitrone"],[ "Pflaume"]])
```

ValueError

Cell In[71], line 3

```
1 # Anwendung auf neue Daten ("Pflaume" wurde während des Trainings nicht gesehen)
```

Traceback (most recent call last)



## 1.5 Zusammenfassung

### Visualisierung:

1. Für Visualisierungen in Python kann `matplotlib`, `pandas`, `seaborn` verwendet werden (und weitere Paketen z.B. zur interaktiven Datenexploration)
2. Da unser Ausgangspunkt oft `pandas`-Daten sind, ist die `pandas`-Funktionalität bequem
3. `pandas` bietet einen Wrapper um `matplotlib`-Funktionen, Befehle können kombiniert werden

### Skalen und Encodings:

1. Skalen nominal / ordinal (qualitativ, "kategorisch") und kardinal (quantitativ)
2. Label Encoding und One-Hot Encoding für kategorische Werte (Label Encoding nur für Ordinal-Skala)
3. **Zusammengefasst: One-Hot-Encoding als Standard-Verfahren zum Kodieren kategorischer Werte**

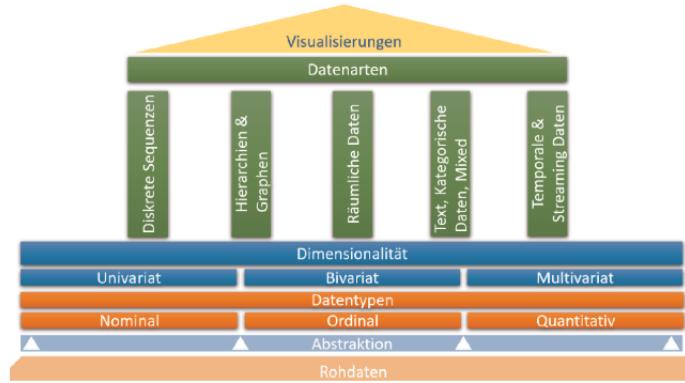
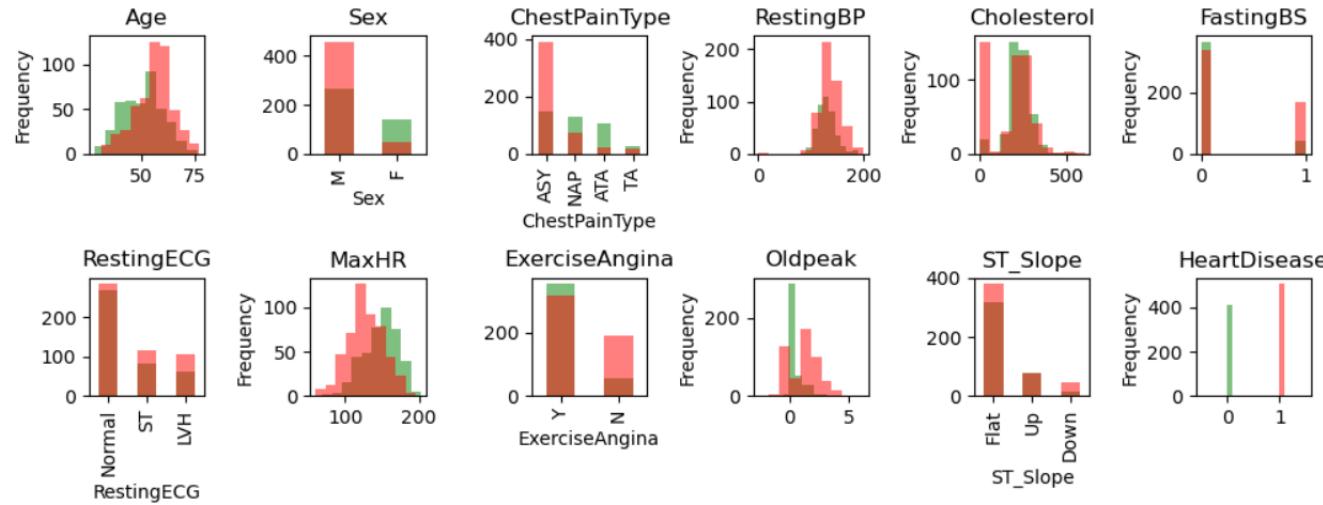


Abb. 3 aus: Nazemi, Kawa & Kaupp, Lukas & Burkhardt, Dirk & Below, Nicola. (2021). Datenvisualisierung. 10.1515/9783110657807-026.

## 1. Visualisierung und Encodings

## 2. DecisionTreeClassifier

```
In [3]: 1 def show_heartdata ( df ): # Alternative z.B. seaborn.pairplot
2     fig, axes = plt.subplots ( figsize=(10,4), nrows=2, ncols=len(df.columns)//2 )
3     for c,ax in zip ( df.columns, axes.flatten() ):
4         ax.set_title(c)
5         for hd in [0,1]:
6             if df[c].dtype == "object":
7                 df[c][df["HeartDisease"]==hd].value_counts().plot.bar( ax=ax, alpha=0.5, color="red" if hd==1 else "green" )
8             else:
9                 df[c][df["HeartDisease"]==hd].plot.hist ( ax=ax, alpha=0.5, color="red" if hd==1 else "green" )
10    plt.tight_layout();
11
12 show_heartdata ( df )
```



## 3.1 Daten-Preprocessing

Konvention: Die Daten sollen in einer Datenmatrix  $X$  und einem Zielvektor  $y$  vorliegen.

```
In [4]: 1 X = df.iloc[:600,:-1].copy()      # wir nehmen nur die ersten 600 Zeilen zum Trainieren.  
2 y = df.iloc[:600,-1].copy()        # Die letzte Spalte "HeartDisease" wird in eine Variable y absepariert.  
3                                         # (.loc/.iloc als Notation zum Zugriff auf Datenzeilen in pandas -- später dazu mehr)  
4
```

Umwandlung kategorischer in numerische Features (schlecht, aber schnell: alles via Label Encoding -> Hausaufgabe):

```
In [7]: 1 from sklearn.preprocessing import OrdinalEncoder  
2 enc = OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value=-1)  
3 X = pd.DataFrame ( enc.fit_transform(X.values), columns = X.columns ) # Spaltennamen gehen leider sonst verloren  
4 X.head() # Die Spalte "HeartDisease" fehlt, diese wird separat in der Variable y gespeichert.  
5
```

Out[7]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	12.0	1.0	1.0	35.0	134.0	0.0	1.0	93.0	0.0	10.0	2.0
1	21.0	0.0	2.0	48.0	33.0	0.0	1.0	79.0	0.0	19.0	1.0
2	9.0	1.0	1.0	25.0	129.0	0.0	2.0	24.0	0.0	10.0	2.0
3	20.0	0.0	0.0	33.0	63.0	0.0	1.0	33.0	1.0	24.0	1.0
4	26.0	1.0	2.0	42.0	45.0	0.0	1.0	47.0	0.0	10.0	2.0



## 3.2 Der DecisionTreeClassifier aus Scikit-Learn

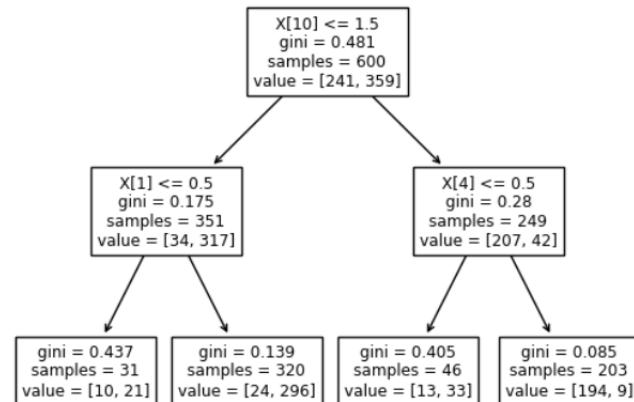
Die Machine Learning Modelle (und weitere Klassen z.B. zum Preprocessing) haben eine einheitliche Struktur, insb.:

- `obj = <sklearn-Klasse>(...)` (Initialisierung mit sog. Hyperparametern)
- `obj.fit ( X [ , y] )` (Anpassen des Objekts auf gegebene Trainingsdaten)
- `obj.predict ( X )` (Anwenden des trainierten Objekts zur Vorhersage der Zielgröße)
- `obj.transform ( X ), obj.fit_transform ( X )` (Anwendung von Preprocessing-Klassen)

**Ein einfacher Baum.** "gini" ist ein Maß für die "Impurity". 0 bedeutet: eine Menge enthält nur 0en oder nur 1en. 1/2 bedeutet: die Menge ist gut durchmischt. Der Splitting-Algorithmus sucht nach optimalen Feature/Threshold-Kombinationen, um die Impurity-Maße nach dem Split (Mittelung der Impurity-Maße der beiden neu erzeugten Teilmengen) zu minimieren.

In [371]:

```
1 from sklearn import tree
2 clf = tree.DecisionTreeClassifier ( max_depth=2 )
3 clf.fit ( X, y )
4 tree.plot_tree ( clf );
5
```



In [ ]:

```
1 # In der Baum-Visualisierung:
2 # Erste Zeile: Split-Kriterium
3 # Zweite Zeile: Impurity vor dem Split
4 # Dritte Zeile: Anzahl Zeilen vor dem Split
5 # Vierte Zeile: Anzahl y=0 vs y=1 vor dem Split
6
```

In [370]:

```
1 # Vorhersage der Klassen 0=gesund, 1=krank
2 # (kann dann mit den wahren Werten
3 # y verglichen werden)
4 clf.predict ( X )
5
```

Out[370]:

```
array([0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
```

## Zusammenfassung der Klassifikationsergebnisse in einer **Konfusionsmatrix**:

Wahre Klasse:	positiv (krank)	negativ (gesund)
Vorhergesagte Klasse:	positiv (krank)	? (TP)
	negativ (gesund)	? (FN)

### Vier Felder:

- **Richtig positiv (TP):** Der Patient ist krank, und der Test hat dies richtig angezeigt.
- **Falsch positiv (FP):** Der Patient ist gesund, aber der Test hat ihn fälschlicherweise als krank eingestuft.
- **Richtig negativ (TN):** Der Patient ist gesund, und der Test hat dies richtig angezeigt.
- **Falsch negativ (FN):** Der Patient ist krank, aber der Test hat ihn fälschlicherweise als gesund eingestuft.

Konvention: "Positiv" ist das "unnormale"/"das seltenere", das detektiert werden soll.

```
In [11]: 1 # Vergleiche den Vektor der predictions mit dem Vektor der Ground Truth (y)
2 predictions = clf.predict ( X )
3
4 TP =
5 FP =
6 TN =
7 FN =
```



```
In [8]: 1 TP = (y==1) & (predictions == 1)
2 FP = (y==0) & (predictions == 1)
3 TN = (y==0) & (predictions == 0)
4 FN = (y==1) & (predictions == 0)
5
6 TP = TP.sum()
7 FP = FP.sum()
8 TN = TN.sum()
9 FN = FN.sum()
10
11 TP, FP, TN, FN
```

Out[8]: (350, 47, 194, 9)

Zusammenfassung in einer **Konfusionsmatrix**:

Wahre Klasse:	positiv (krank)	negativ (gesund)
Vorhergesagte Klasse:	positiv (krank)	negativ (gesund)
	350 (TP) 47 (FP)	9 (FN) 194 (TN)

Von 600 Personen wurden also  $544=350+194$  Personen der korrekten Klasse zugeordnet.

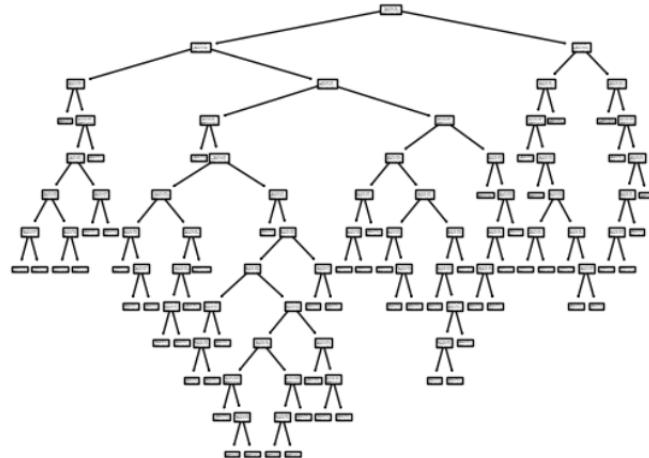
**Treffergenauigkeit (Accuracy):**  $(TP+TN)/(TP+TN+FP+FN) = 544 / 600 = 90,6\%$ .

Wir werden noch diskutieren, dass es in der Praxis bessere Metriken als die Accuracy gibt.

Für den Rest der Vorlesung soll ein langer Baum (**links**) mit einem kurzen Baum (**rechts**) verglichen werden.

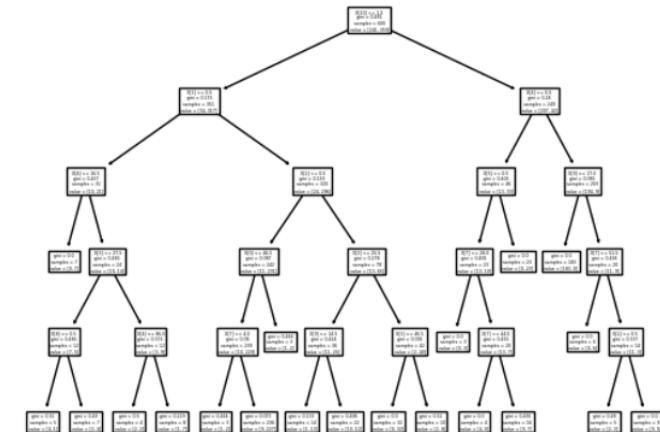
In [366]:

```
1 clf_long = tree.DecisionTreeClassifier ( )
2 clf_long.fit ( X, y )
3 tree.plot_tree ( clf_long );
4
```



In [360]:

```
1 clf_short = tree.DecisionTreeClassifier(max_depth=5,
2                                         min_samples_leaf=3)
3 clf_short.fit ( X, y )
4 tree.plot_tree ( clf_short );
5
```



### 3.3 Vorhersage der Klassen 0=gesund, 1=krank auf den Trainingsdaten

```
In [258]: 1 # Erstes Modell: Langer Entscheidungsbaum  
2 pred_long = clf_long.predict(X)  
3 pred_long  
4
```

```
Out[258]: array([0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,  
0, 1, 1, 0, 0,  
    0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,  
0, 0, 1, 0, 0,  
    1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,  
0, 0, 1, 0, 0,  
    0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,  
0, 0, 1, 0, 0,  
    0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,  
0, 1, 1, 1, 0,  
    1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,  
0, 0, 0, 0, 0,  
    0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,  
0, 0, 0, 0, 1,  
    1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,  
1, 0, 0, 0,  
    0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,  
0, 0, 0, 1, 1,  
    1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0,  
    1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,  
0, 1, 0, 0, 0,  
    1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,  
1, 1, 1, 0, 1,  
    1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 1, 1,  
    1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,  
1, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,  
1, 1, 1, 1, 0,  
    1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 0, 1, 1, 1,
```

```
In [361]: 1 # Alternativmodell: kurzer Entscheidungsbaum  
2 pred_short = clf_short.predict(X)  
3 pred_short  
4
```

```
Out[361]: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,  
0, 1, 1, 0, 1,  
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,  
0, 0, 1, 0, 0,  
    1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,  
0, 0, 1, 0, 0,  
    0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,  
0, 1, 1, 1, 0,  
    1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,  
0, 0, 1, 0, 0,  
    0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,  
0, 1, 1, 1, 0,  
    1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,  
0, 0, 0, 0, 0,  
    0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,  
1, 0, 1, 0, 1,  
    1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,  
1, 0, 0, 0, 0,  
    0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,  
0, 0, 0, 1, 0,  
    1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,  
0, 1, 0, 0, 0,  
    1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,  
1, 1, 1, 0, 1,  
    1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,  
0, 0, 0, 1, 1,  
    1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,  
1, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,  
1, 1, 1, 1, 1,  
    1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 0, 1, 1, 1,
```

## 3.4 Evaluation auf den Trainingsdaten

In [260]:

```
 1 # Langer Baum
 2
 3 TP = (y==1) & (pred_long == 1)
 4 FP = (y==0) & (pred_long == 1)
 5 TN = (y==0) & (pred_long == 0)
 6 FN = (y==1) & (pred_long == 0)
 7
 8 TP = TP.sum()
 9 FP = FP.sum()
10 TN = TN.sum()
11 FN = FN.sum()
12 acc = (TP+TN)/(TP+TN+FP+FN)
13
14 print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
15 print(f"Accuracy: {acc*100:.02f}%")
16
```

```
TP: 359, FP: 0, TN: 241, FN: 0
Accuracy: 100.00%
```

In [362]:

```
 1 # kurzer Baum
 2
 3 TP = (y==1) & (pred_short == 1)
 4 FP = (y==0) & (pred_short == 1)
 5 TN = (y==0) & (pred_short == 0)
 6 FN = (y==1) & (pred_short == 0)
 7
 8 TP = TP.sum()
 9 FP = FP.sum()
10 TN = TN.sum()
11 FN = FN.sum()
12 acc = (TP+TN)/(TP+TN+FP+FN)
13
14 print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
15 print(f"Accuracy: {acc*100:.02f}%")
16
```

```
TP: 349, FP: 30, TN: 211, FN: 10
Accuracy: 93.33%
```

## 3.6 Evaluation auf Daten, die nicht zum trainieren verwendet wurden ("Test/Validierungsdaten")

```
In [285]: 1 X_test = df.iloc[600:,:-1].copy()      # wir nehmen die verbleibenden Zeilen zum Testen (besser wäre: randomisiert)
2 y_test = df.iloc[600:,-1].copy()
3
```

**Frage:** Was könnte bei der Neuerstellung des Encoders auf die Testdaten schiefgehen?

```
In [237]: 1 # from sklearn.preprocessing import OrdinalEncoder
2 # enc = OrdinalEncoder()
3 # X_test = pd.DataFrame ( enc.fit_transform(X_test), columns = X_test.columns )
4
```

```
In [286]: 1 # Wiederverwendung des Label-Encoders, der für die Trainingsdaten verwendet wurde, damit die Zuordnung Wert->Zahl identisch ist
2 X_test = pd.DataFrame ( enc.transform(X_test.values), columns = X_test.columns )
3
```

## Evaluierung auf den Testdaten:

```
In [291]: 1 # Langer Baum
2 pred_long_test = clf_long.predict(X_test)
3
4 TP = (y_test==1) & (pred_long_test == 1)
5 FP = (y_test==0) & (pred_long_test == 1)
6 TN = (y_test==0) & (pred_long_test == 0)
7 FN = (y_test==1) & (pred_long_test == 0)
8
9 TP = TP.sum()
10 FP = FP.sum()
11 TN = TN.sum()
12 FN = FN.sum()
13 acc = (TP+TN)/(TP+TN+FP+FN)
14
15 print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
16 print(f"Accuracy: {acc*100:.02f}%")
17
```

TP: 101, FP: 39, TN: 130, FN: 48  
Accuracy: 72.64%

```
In [363]: 1 # kurzer Baum
2 pred_short_test = clf_short.predict(X_test)
3
4 TP = (y_test==1) & (pred_short_test == 1)
5 FP = (y_test==0) & (pred_short_test == 1)
6 TN = (y_test==0) & (pred_short_test == 0)
7 FN = (y_test==1) & (pred_short_test == 0)
8
9 TP = TP.sum()
10 FP = FP.sum()
11 TN = TN.sum()
12 FN = FN.sum()
13 acc = (TP+TN)/(TP+TN+FP+FN)
14
15 print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
16 print(f"Accuracy: {acc*100:.02f}%")
17
```

TP: 117, FP: 52, TN: 117, FN: 32  
Accuracy: 73.58%

**Zusammenfassung:** Nicht die Performance auf den Trainingsdaten, sondern die Performance auf den zuvor nicht gesehenen Testdaten gibt Auskunft über die Generalisierungsfähigkeit. Ein Unterschied zwischen Trainings- und Testperformance lässt auf einen **Overfit** schließen.

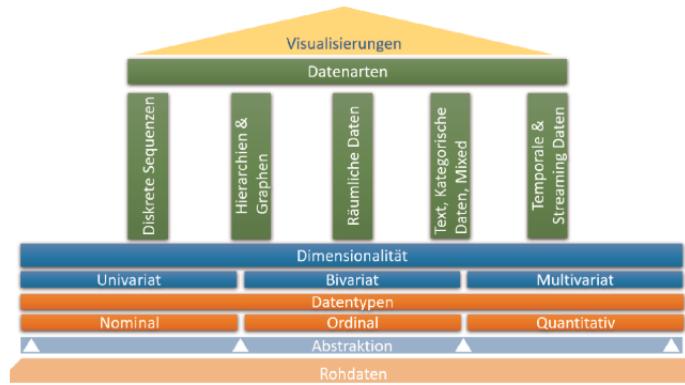


Abb. 3 aus: Nazemi, Kawa & Kaupp, Lukas & Burkhardt, Dirk & Below, Nicola. (2021). Datenvisualisierung. 10.1515/9783110657807-026.

## 1. Visualisierung und Encodings

## 2. DecisionTreeClassifier

**Vielen Dank für Ihre  
Aufmerksamkeit!**

# Anhang

## 1.10.7. Mathematical formulation

Given training vectors  $x_i \in R^n$ ,  $i=1, \dots, l$  and a label vector  $y \in R^l$ , a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node  $m$  be represented by  $Q_m$  with  $n_m$  samples. For each candidate split  $\theta = (j, t_m)$  consisting of a feature  $j$  and threshold  $t_m$ , partition the data into  $Q_m^{left}(\theta)$  and  $Q_m^{right}(\theta)$  subsets

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

The quality of a candidate split of node  $m$  is then computed using an impurity function or loss function  $H()$ , the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_{\theta} G(Q_m, \theta)$$

Recurse for subsets  $Q_m^{left}(\theta^*)$  and  $Q_m^{right}(\theta^*)$  until the maximum allowable depth is reached,  $n_m < \min_{samples}$  or  $n_m = 1$ .

<https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation>

### 1.10.7.1. Classification criteria

If a target is a classification outcome taking on values  $0, 1, \dots, K-1$ , for node  $m$ , let

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

be the proportion of class  $k$  observations in node  $m$ . If  $m$  is a terminal node, `predict_proba` for this region is set to  $p_{mk}$ . Common measures of impurity are the following.

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

Log Loss or Entropy:

$$H(Q_m) = - \sum_k p_{mk} \log(p_{mk})$$

<https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation>