Datenvisualisierung mit Pyplot

Repetitorium der Computerlinguistik

Marina Sedinkina
Folien von Benjamin Roth

CIS LMU München

Pyplot

- Daten und Funktionen plotten in Python.
- Paket der `matplotlib` Bibliothek.
- Verwendet Datenstrukturen der `numpy` Bibliothek
- Verwendung an die plotting-Befehle von `matlab` angelehnt
- Alle folgenden Beispiele verwenden die Imports:

```
import numpy as np
import matplotlib.pyplot as plt
```

Beispiel: X und Y-Werte Erzeugen

- `numpy` Array, das 256 X-Werte zwischen $-\pi$ und π enthält:

  ```
  X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
  ```

- `C` und `S` enthalten die dazugehörigen Y-Werte:

  ```
  C,S = np.cos(X), np.sin(X)
  ```

X und Y Werte mit Standardeinstellungen Darstellen

- Plot in Default-Viewer öffnen:

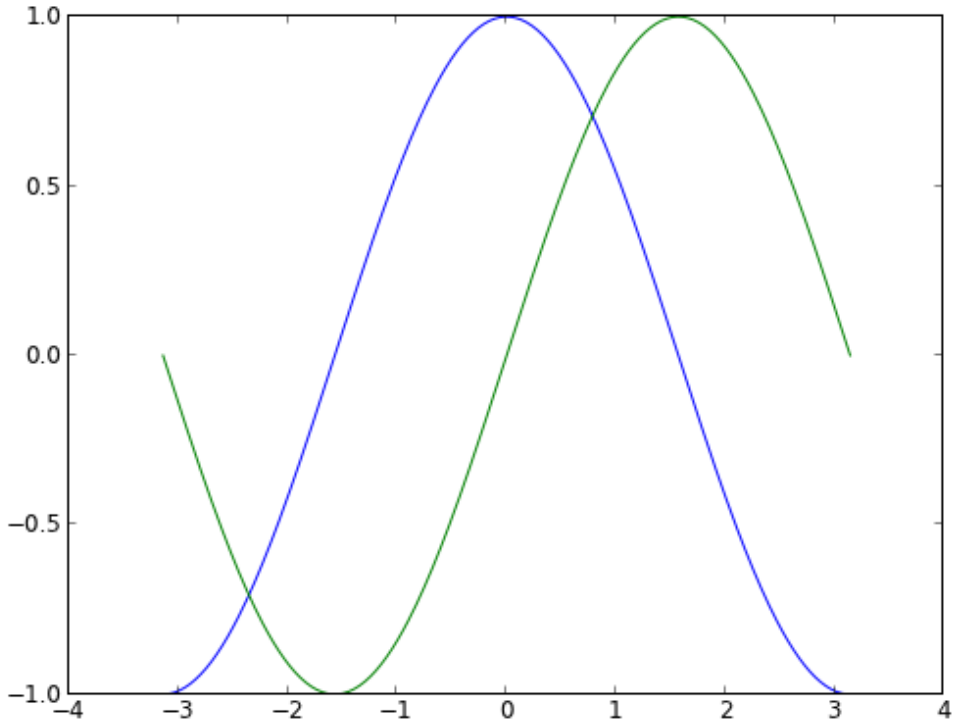
```
plt.plot(X,C)
plt.plot(X,S)
plt.show()
```

- Plot in Datei speichern:

```
plt.savefig("/path/to/example.png")
```

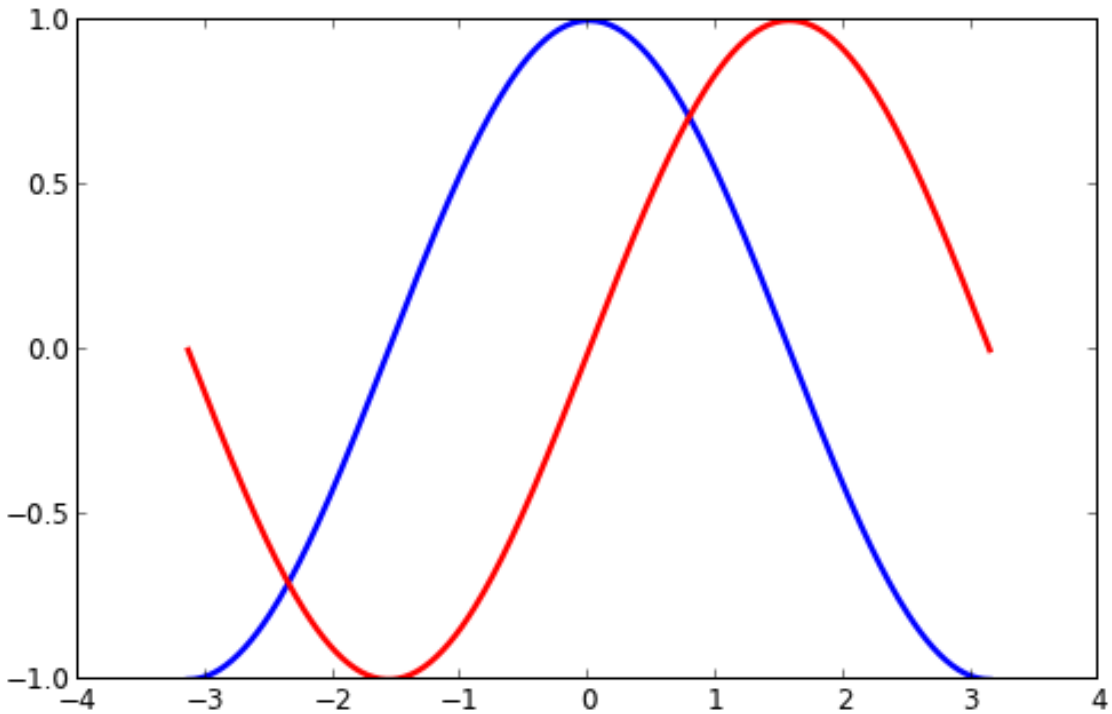
- Viele Formate möglich: pdf, png, ps, eps, svg
- Format wird anhand der Datei-Endung gewählt

X und Y Werte mit Standardeinstellungen Darstellen



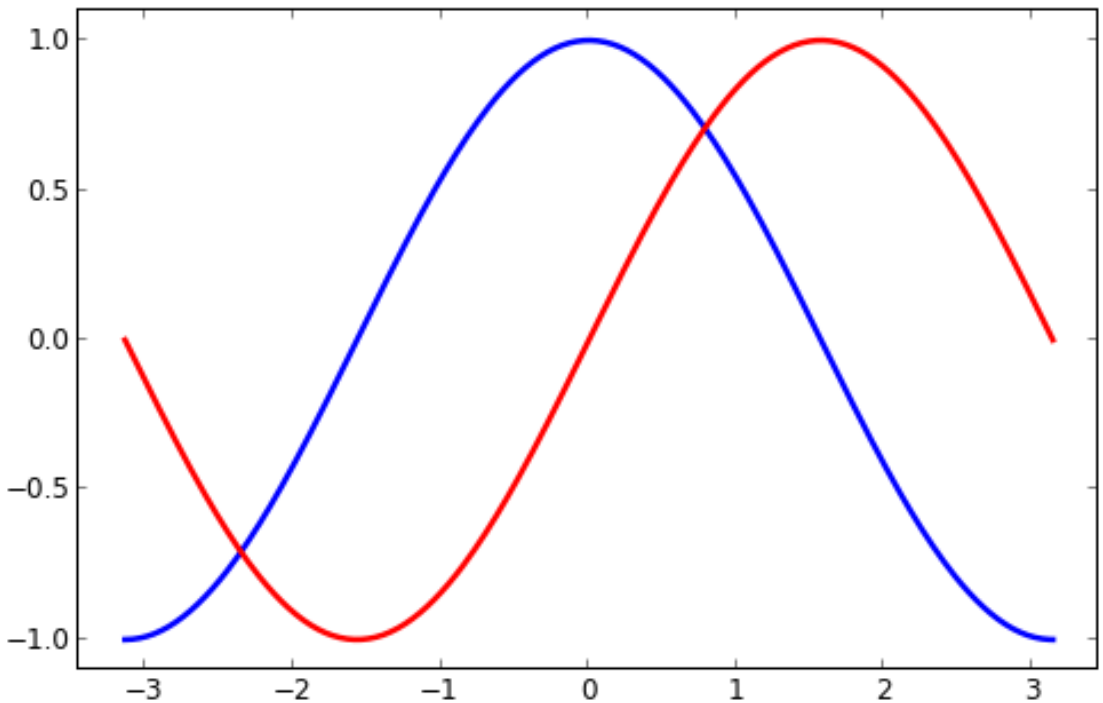
- Zeigt die wesentliche Information, aber verbesserungswürdig
- → kräftigere Linien, Farben
- → Hervorhebung besonderer Punkte
- → schönere Gestaltung der X- und Y-Achsen

Liniendicke, Farbe, Abmessungen



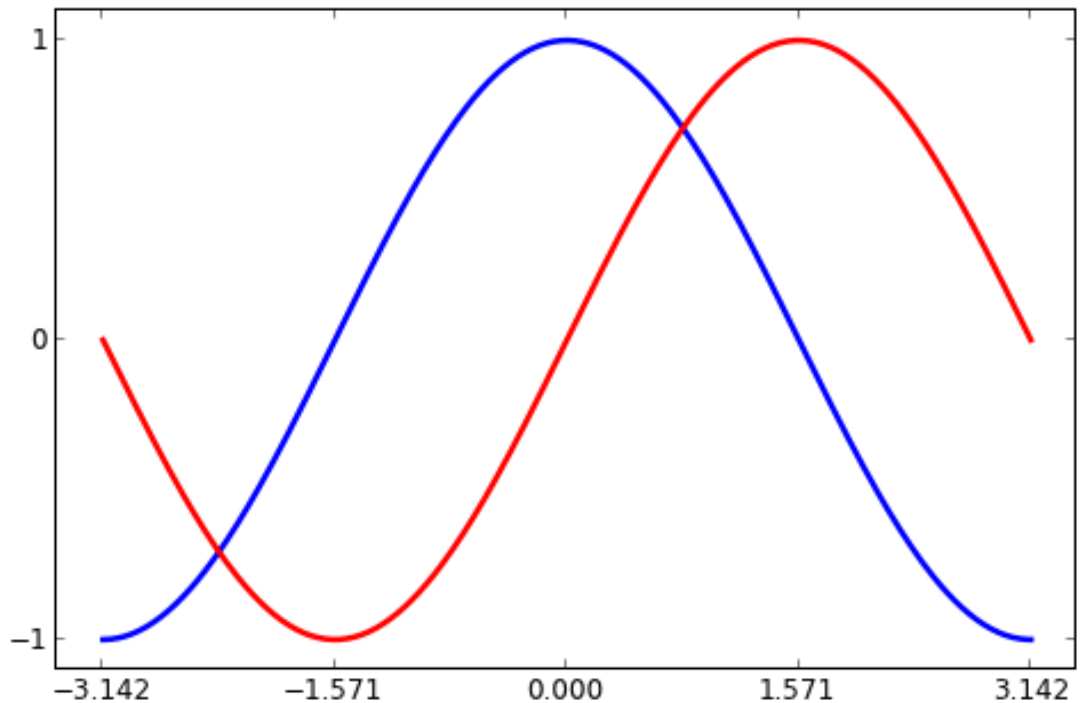

```
plt.figure(figsize=(10,6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
```

Zwischenraum von Außenachsen zu Plot



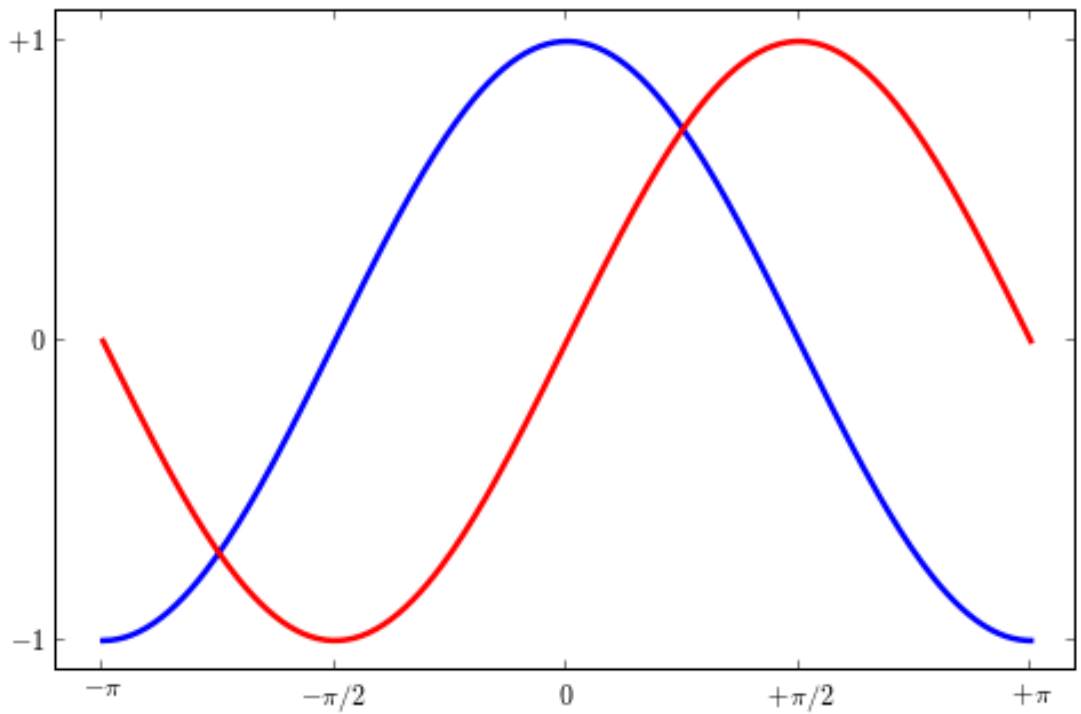

```
plt.xlim(X.min()*1.1, X.max()*1.1)
plt.ylim(C.min()*1.1, C.max()*1.1)
```

Achsenbeschriftung



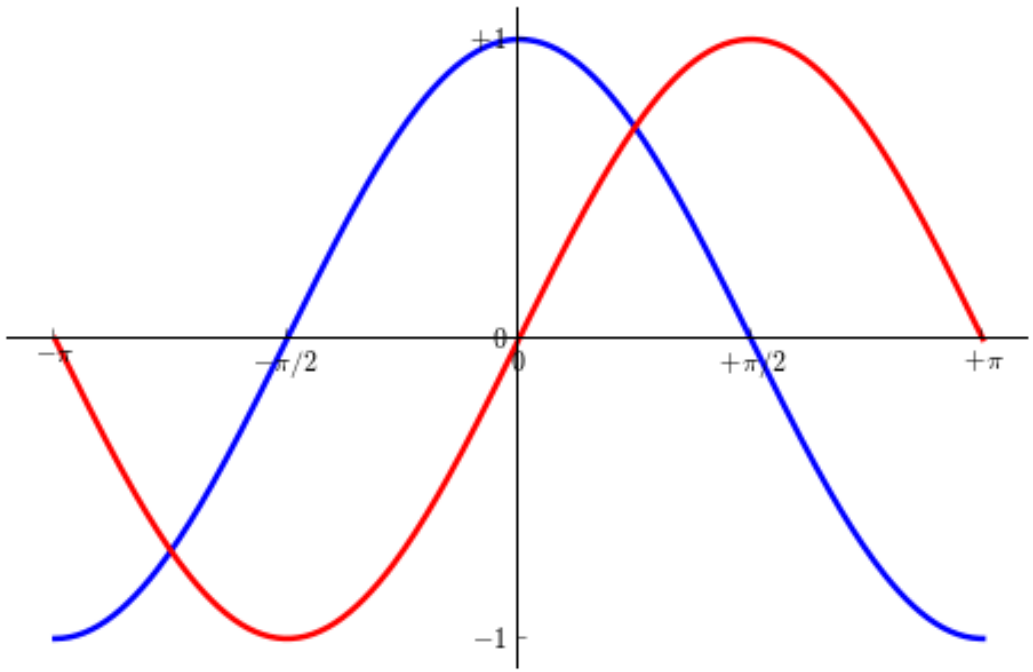

```
plt.xticks( [-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
```

Achsenbeschriftung mit LaTeX-Formeln



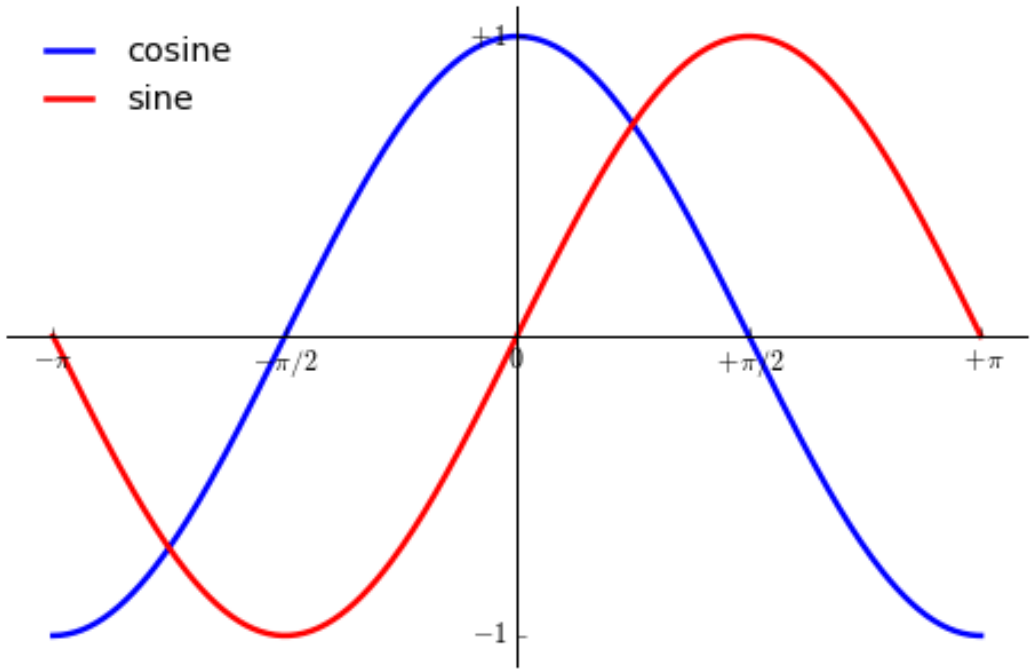

```
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
    [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1],
    [r'$-1$', r'$0$', r'$+1$'])
```

Achsen Verschieben



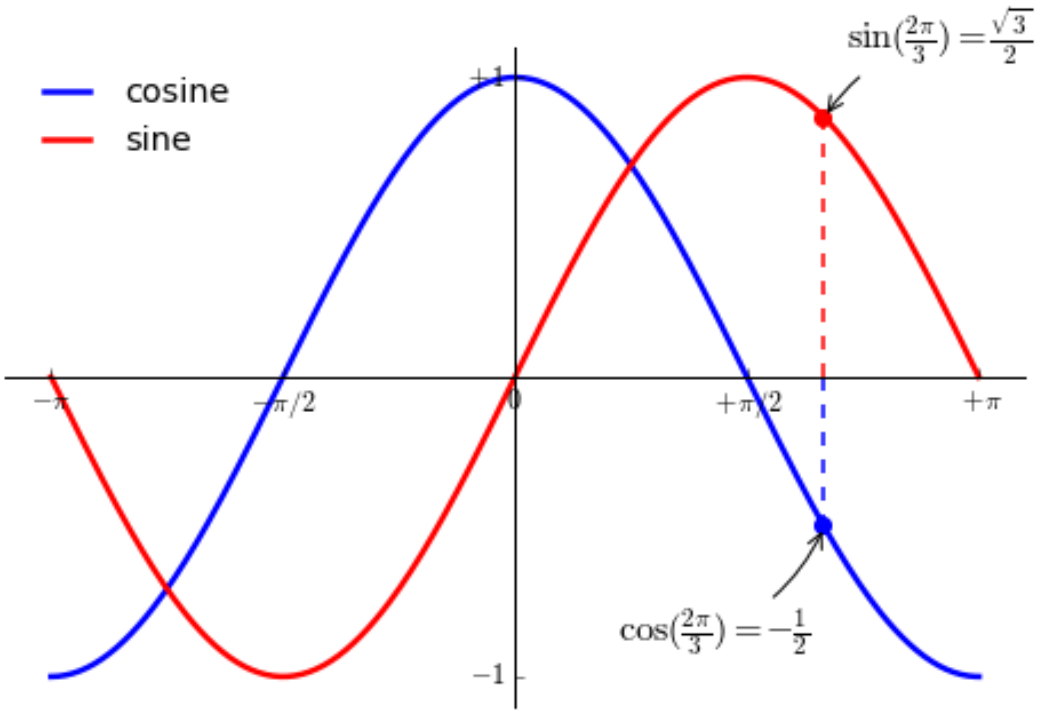

```
ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position(('data',0))
# ... do same for 'left' and 'right'
```

Legende Hinzufügen

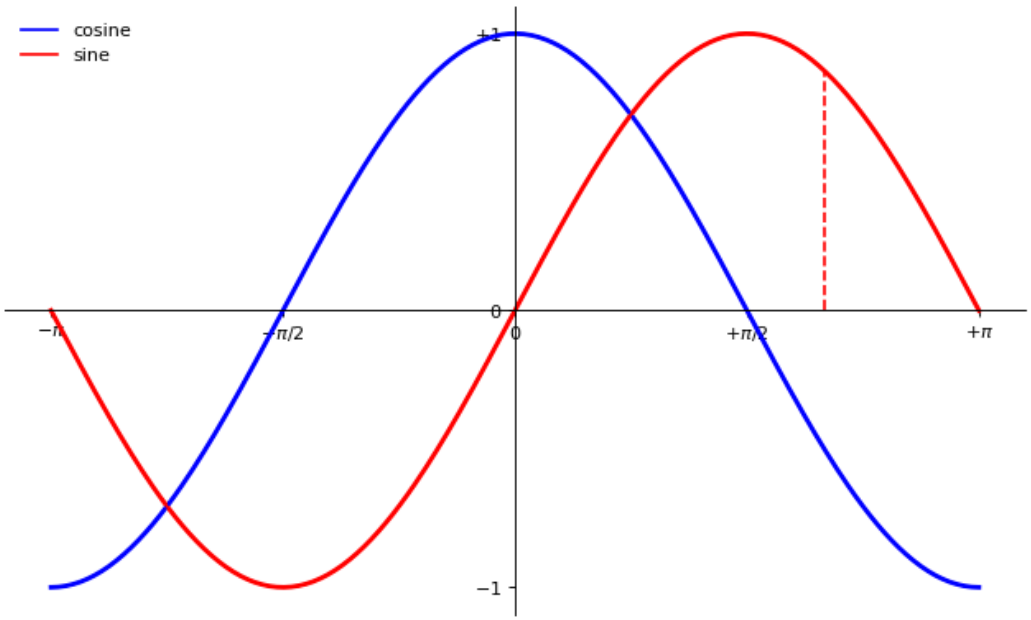



```
plt.plot(X, C, color="blue", ..., label="cosine")
plt.plot(X, S, color="red",  ..., label="sine")
plt.legend(loc='upper left', frameon=False)
```

Annotieren eines Punktes

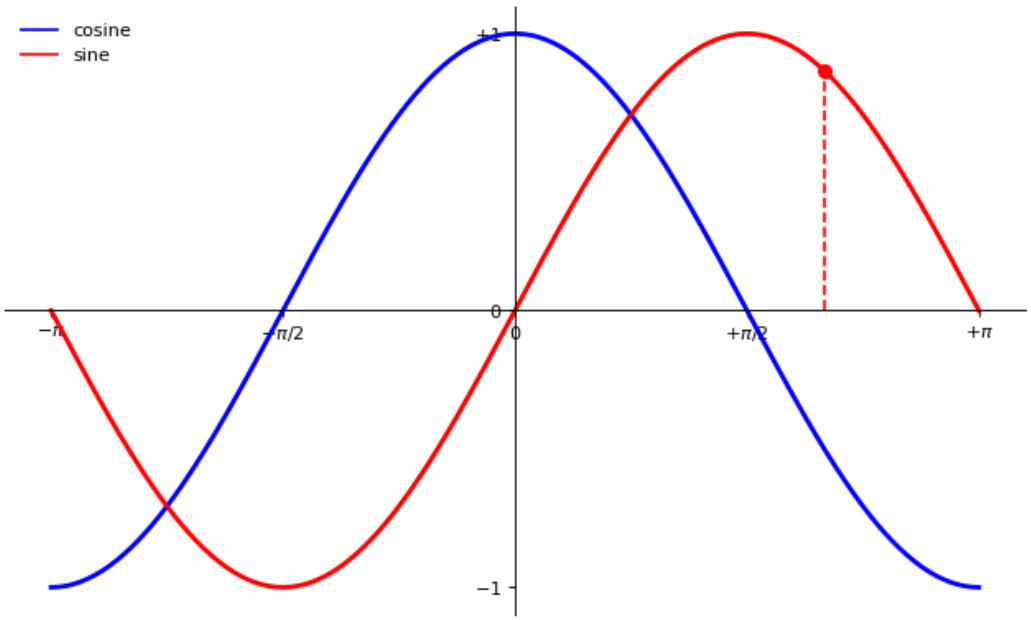


Annotieren eines Punktes



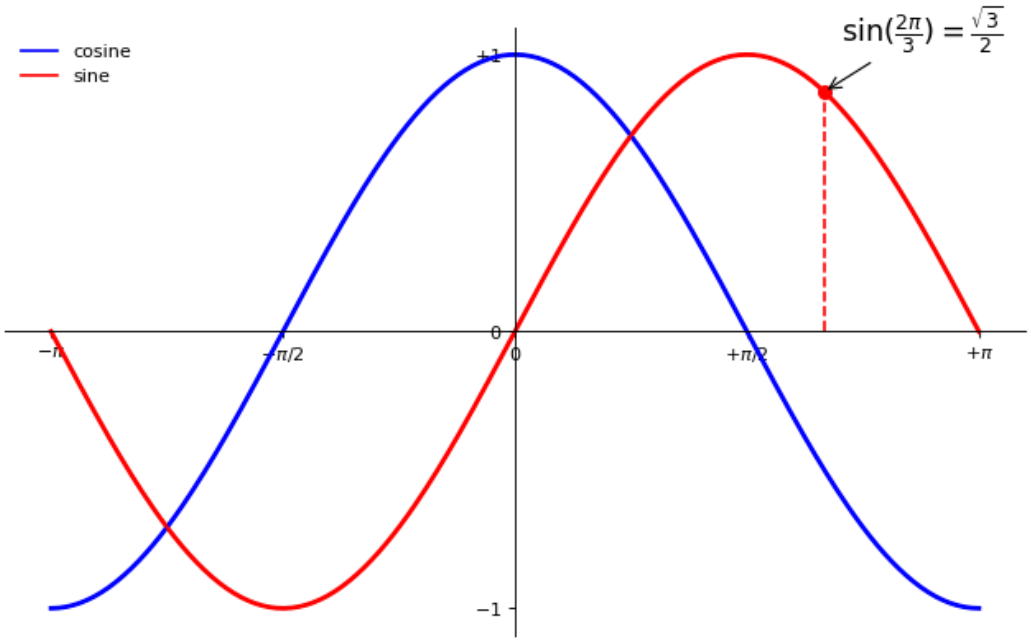

```
t = 2*np.pi/3
plt.plot([t,t],[0,np.sin(t)], color ='red',
         linewidth=1.5, linestyle="--")
```

Annotieren eines Punktes



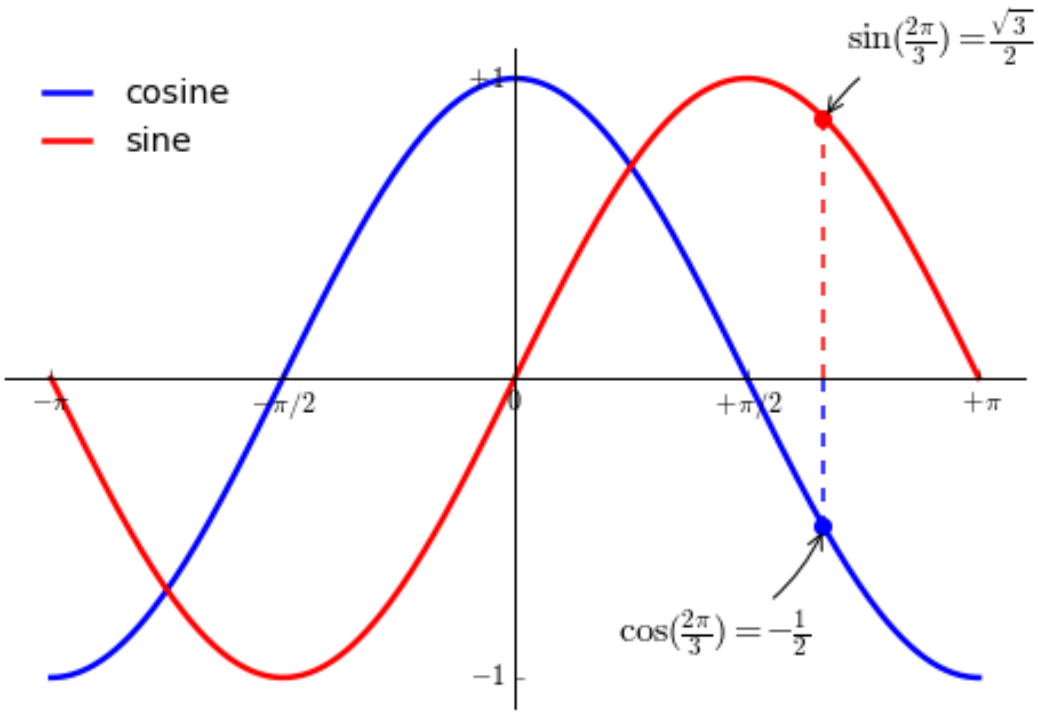

```
t = 2*np.pi/3
plt.plot([t,t],[0,np.sin(t)], color ='red',
         linewidth=1.5, linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='red')
```

Annotieren eines Punktes



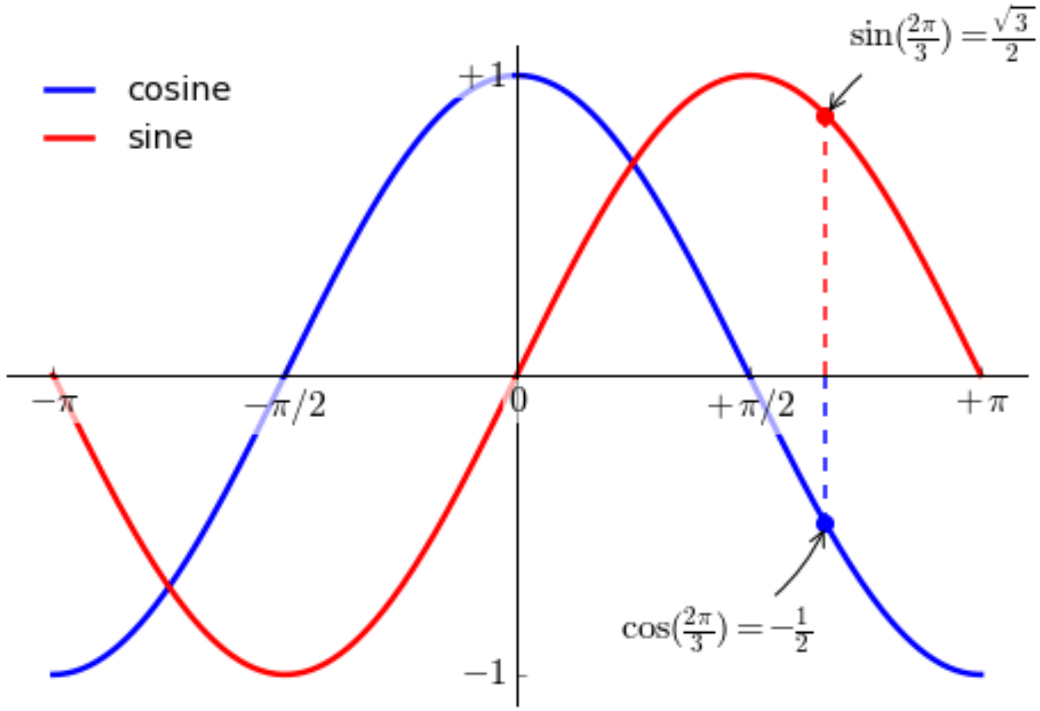

```
t = 2*np.pi/3
plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
  xy=(t, np.sin(t)), xycoords='data',
  xytext=(+10, +30), textcoords='offset pixels', fontsize=16,
  arrowprops=dict(arrowstyle="->"))
```

Annotieren eines Punktes



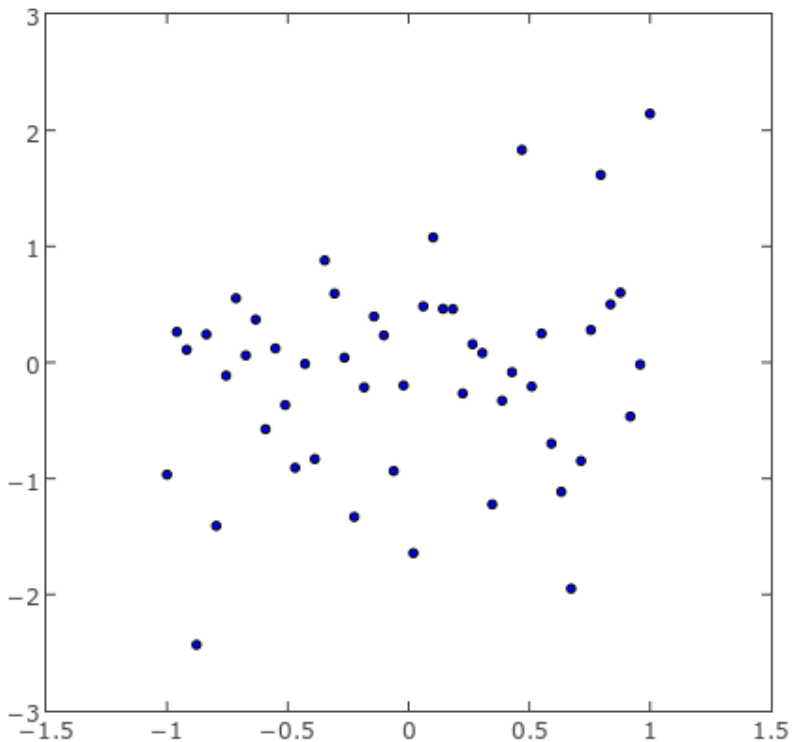

```
# ... do same for cos
```

Annotieren eines Punktes




```
ax = plt.gca()
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_bbox(dict(facecolor='white', edgecolor='white',
                        alpha=0.65 ))
```

Scatter Plots




```
plt.scatter(np.linspace(-1, 1, 50), np.random.randn(50))
```

Scatter Plots: Vieldimensionale Daten

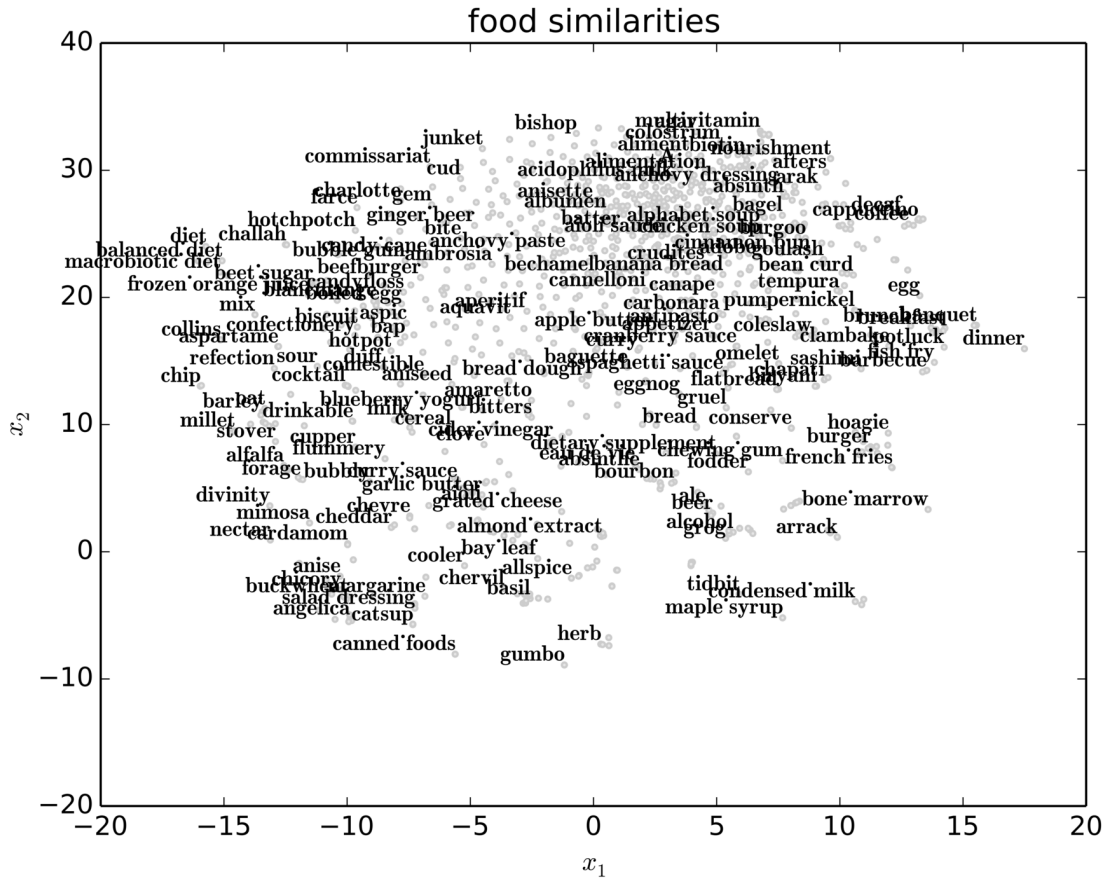
- Daten mit mehr als 2 Dimensionen nicht darstellbar
- $\Rightarrow$ Dimensionsreduktion (PCA ...)
- Wichtig für die *visuelle* Darstellung von Daten:
  - Lokale Distanzen bleiben erhalten
  - Gruppen/Cluster von Punkten bleiben erhalten
- $\Rightarrow$ Es gibt spezielle Algorithmen, die die obigen Kriterien erfüllen (*t-SNE*, van der Maaten und Hinton 2008)

Dimensionsreduktion mit t-SNE

- Daten mit mehr als 2 Dimensionen nicht darstellbar
- $\Rightarrow$ Dimensionsreduktion (PCA ...)
- Wichtig für die *visuelle* Darstellung von Daten:
  - Lokale Distanzen bleiben erhalten
  - Gruppen/Cluster von Punkten bleiben erhalten
- $\Rightarrow$ Es gibt spezielle Algorithmen, die die obigen Kriterien erfüllen (*t-SNE*, van der Maaten und Hinton 2008)

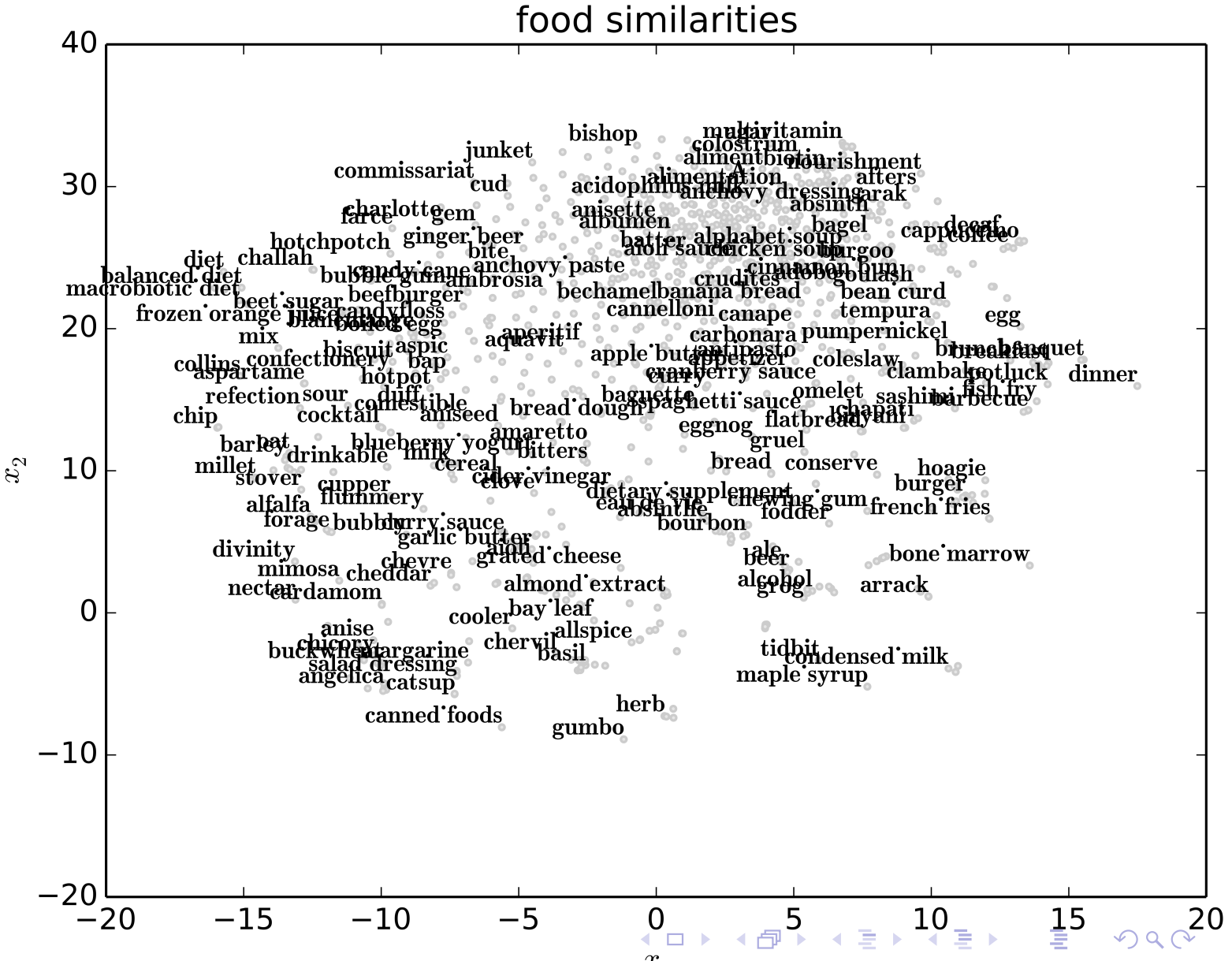
Dimensionsreduktion mit t-SNE: Essensvokabular

1. Stelle jedes Word als 100-dimensionalen Vektor dar (word2vec).
2. Erstelle 2-D Repräsentation mit TSNE



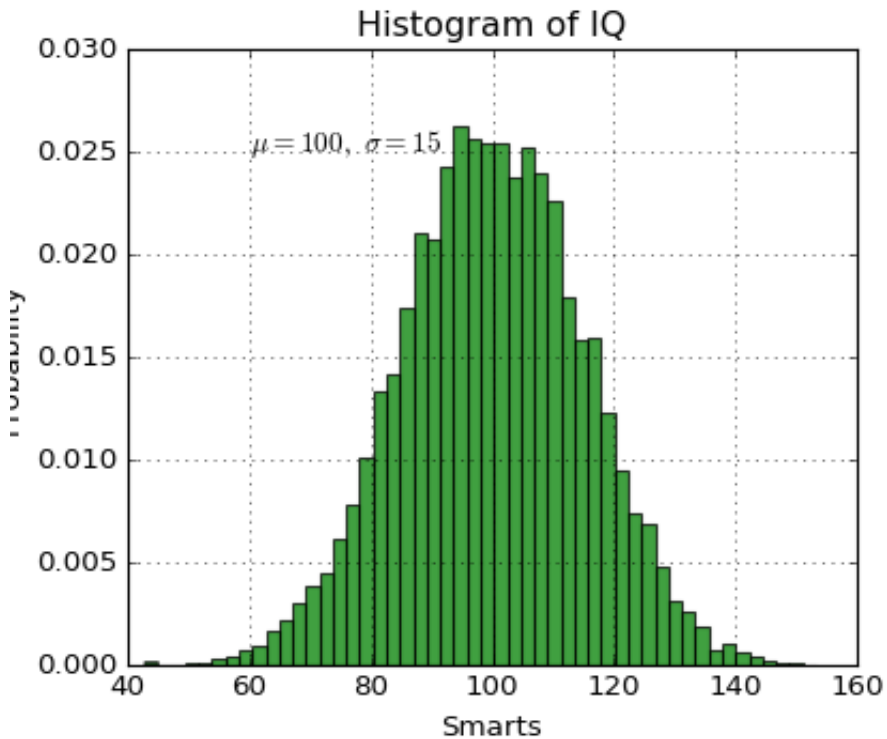

```
from sklearn.manifold import TSNE
```

Dimensionsreduktion mit t-SNE: Essensvokabular



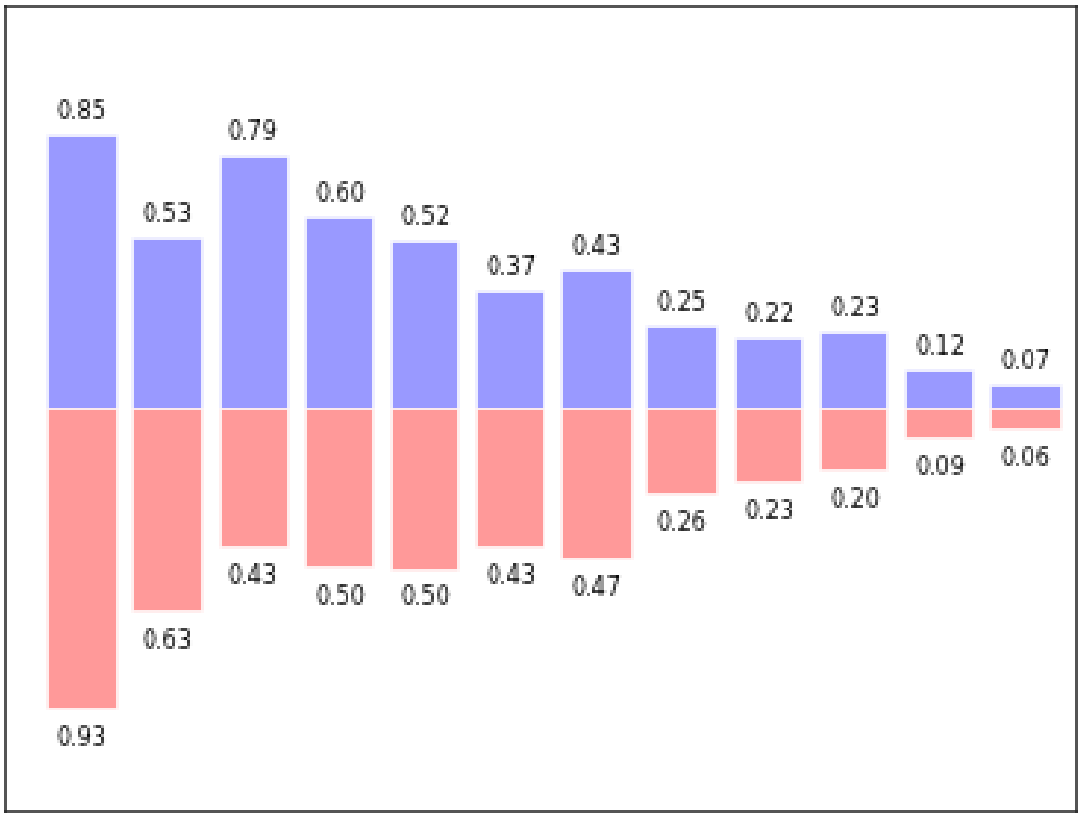
Histogramme

- Darstellung der Verteilung einer Größe.
- Frequenz von Werten innerhalb gleich großer Wertebereiche.




```
x = 100 + 15 * np.random.randn(10000)
plt.hist(x, 50)
```

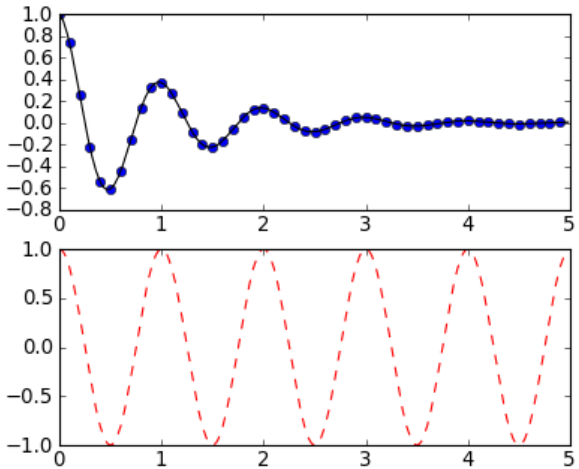
Balkendiagramme




```
plt.bar(X, +Y1)
plt.bar(X, -Y2)
for x,y in zip(X,Y1):
  plt.text(x+0.4, y+0.05, '%.2f' % y)
```

Subplots

- Pro Abbildungen können mehrere Plots angezeigt werden: Übereinander, Nebeneinander, etc



- Die jeweilige Region wird durch `subplot` ausgewählt

```
plt.subplot(2,1,1)
```

- 3 Argumente:
  1. Anzahl Zeilen
  2. Anzahl Spalten
  3. Ausgewähltes Feld (Nummer wird durchgezählt)

Subplots

Hilfe und Ressourcen

- Diese Folien basieren auf:
  - Matplotlib tutorial, Nicolas P. Rougier
    `https://www.labri.fr/perso/nrougier/teaching/matplotlib/`
  - `https://matplotlib.org/index.html`