Matplotlib Crashkurs

August 9, 2021

1 Matplotlib

1.1 Crashkurs

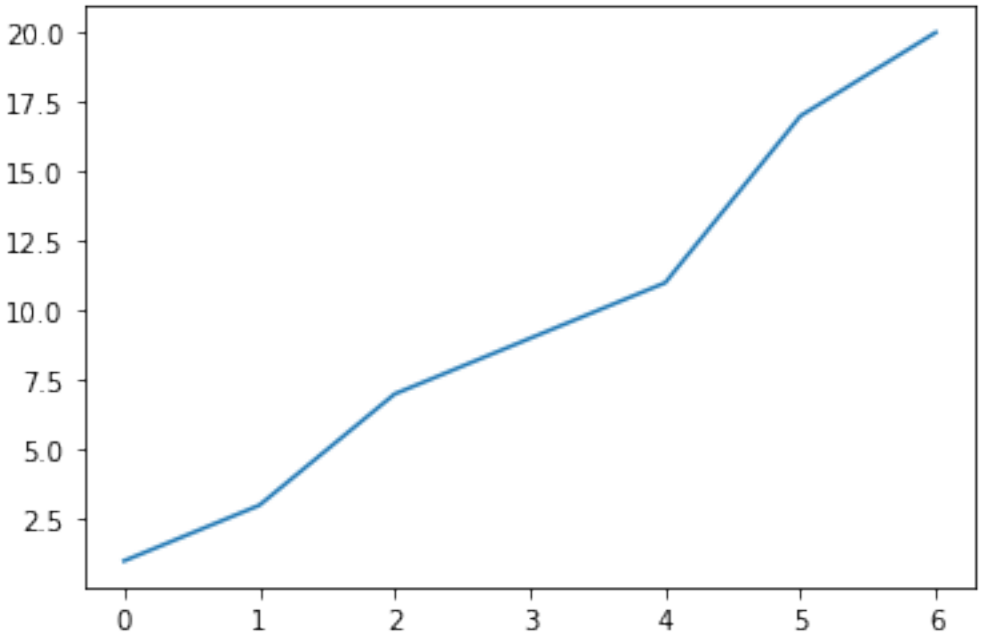
In diesem Tutorial werden grundlegende Möglichkeiten der Bibliothek *Matplotlib* behandelt. Mit *Matplotlib* können auf relativ einfache Weise auch ausgefallene und sehr individuell gestaltete Diagramme erstellt werden. *Matplotlib* setzt die Installation von *NumPy* voraus.

Wir werden auch noch kurz auf *Seaborn* eingehen, sozusagen eine Erweiterung von Matplotlib (*Seaborn* setzt auf Matplotlib und Pandas voraus).

Da wir hier Juypter Notbook verwenden, werden wir die "Magig Function" *%matplotlib inline*. Dadurch werden die erstellen Diagramme in das Jupyter Notebook statisch eingebunden. Möchte man interaktive Diagramme anzeigen, dann verwendet man die Funktion *%matplotlib notebook*.

Auch hier gilt wieder, dass wir die Möglichkeiten dieser Bibliothek nur oberflächlich behandeln können. Beginnen wir mit einem einfachen Beispiel und zeigen grundsätzichliche Möglicheiten, ein Diagramm nach eigenen Wünschen zu formatieren.

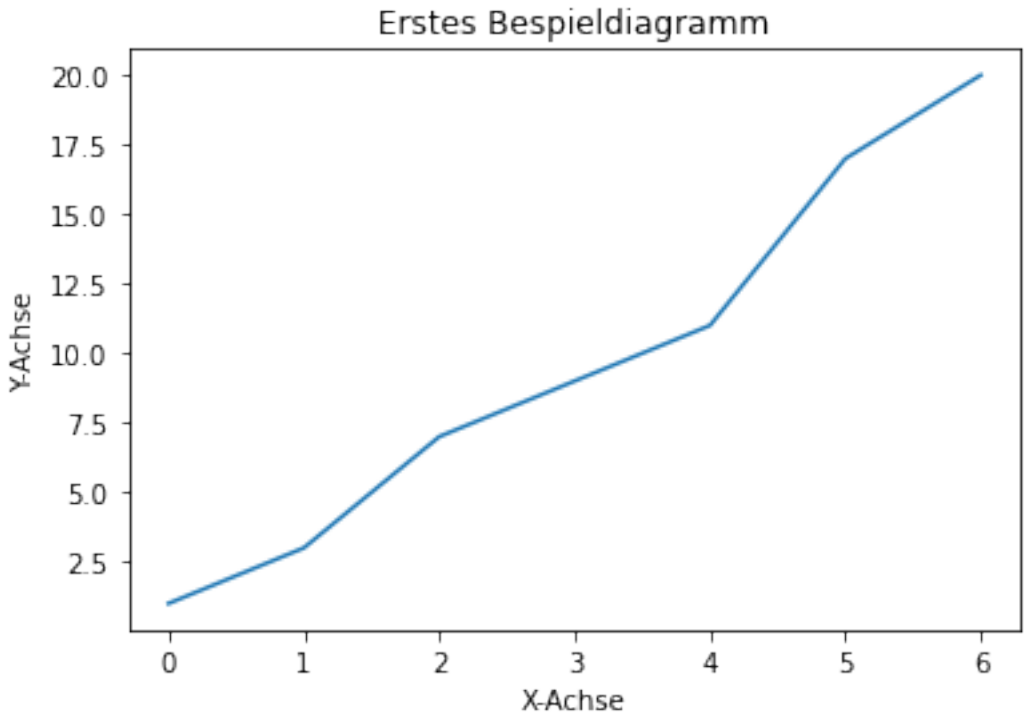
[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
# Einen einfachen Beispieldatensatz generieren: 1-dimensionales Array

x = np.array([1,3,7,9,11,17,20])
plt.plot(x)
plt.show()
```



Im obigen Diagramm werden die Zahlenwerte im Array x in der Reihenfolge, wie sie im Array gespeichert werden, als Liniendiagramm mit *plot* dargestellt.

Es ist üblich, die importierte Bibliothek *matplotlib.pyplot* mit dem Alias *plt* zu benennen. Nach dem Aufruf der Methode *plot* rufen wir noch *plt.show()* auf. Dadurch können auch nach dem Aufruf von *plot* (oder anderer Methoden) noch weitere Eigenschaften des Diagramms definiert werden. Zum Beispiel können wir Achsenbeschriftungen vornehmen oder auch noch einen Diagramm-Titel einfügen:

[2]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
# Einen einfachen Beispieldatensatz generieren: 1-dimensionales Array

x = np.array([1,3,7,9,11,17,20])
plt.plot(x)
plt.title("Erstes Bespieldiagramm")
plt.xlabel("X-Achse")
plt.ylabel("Y-Achse")
plt.show()
```



Die Darstellung der Daten können wir beeinflussen. Dafür gibt es zwei Möglichkeiten:

- Durch Angabe eines Formatstrings “Markersymbol + Farbe + Linientyp”
- Durch dedizierte Angabe der Argumente *color*, *marker* und *linestyle*

Die zweite Möglichkeit sollte vorgezogen werden, da sie lesbareren Quellcode darstellt.

Die Basisfarben werden dabei wie folgt abekürzt:

- b : blau
- g : grün
- r : rot
- c : cyan
- m : magenta
- y : gelb
- k : schwarz
- w : weiß

Die geläufigsten Linientypen (Auswahl):

- “-”: Durchgezogene Linie
- “–”: Gestrichelte Linie
- “:”: gepunktete Linie
- “-.”: Abwechselnd Punkte und Strich

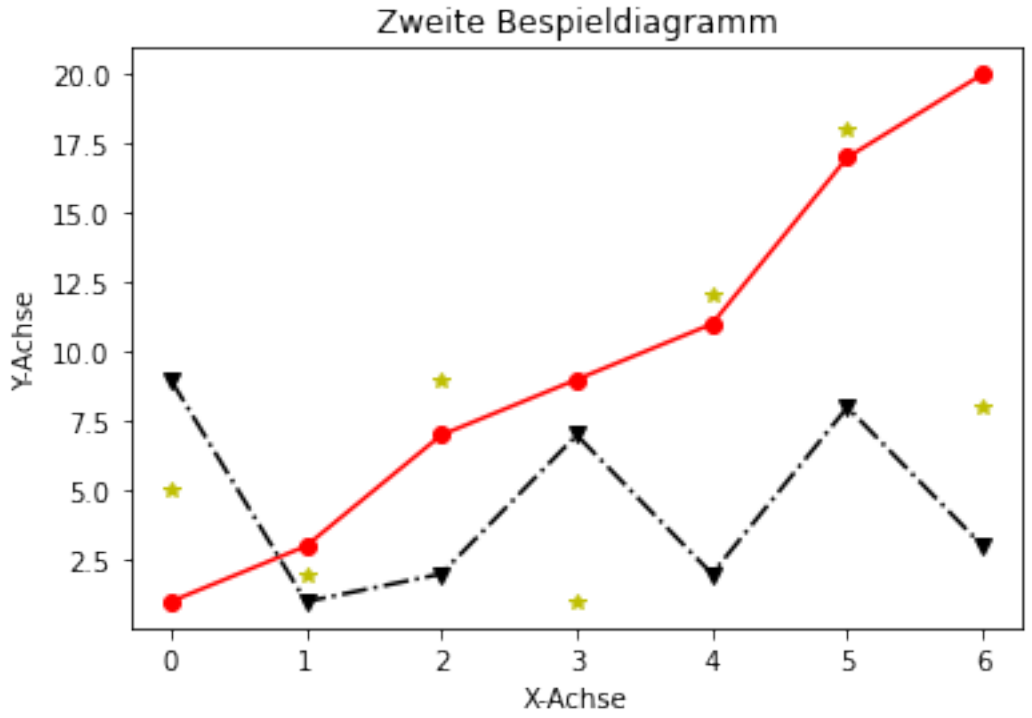
Markertypen (Auswahl):

- “o”: Kreise

- “v”: Dreiecke
- “x”: Kreuze
- “d”: Raute
- “*”: Sterne

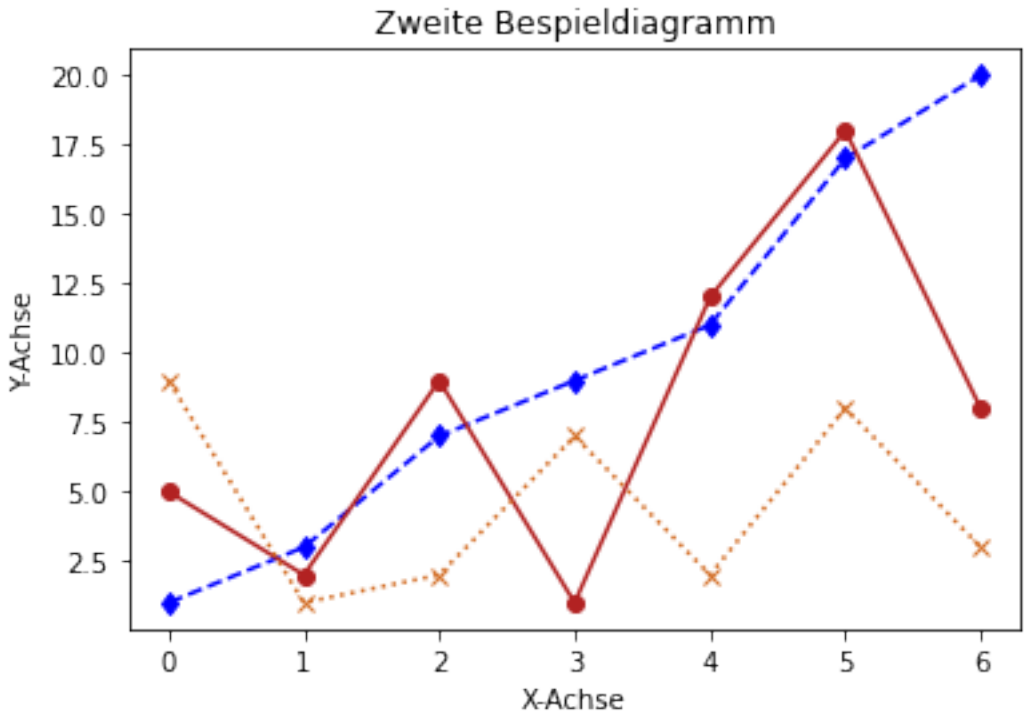
Im folgenden Beispiel erstellen wir noch einige weitere Datensätze und zeigen diese im gleichen Diagramm an, jeweils unterschiedlich formatiert.

[3]:
```
%matplotlib inline

# Variante 1
x1 = np.array([1,3,7,9,11,17,20])
x2 = np.array([5,2,9,1,12,18,8])
x3 = np.array([9,1,2,7,2,8,3])

plt.plot(x1, "or-")
plt.plot(x2, "*y")
plt.plot(x3, "vk-.")
plt.title("Zweite Bespieldiagramm")
plt.xlabel("X-Achse")
plt.ylabel("Y-Achse")
plt.show()
```



Im folgenden Beispiel formatieren wir durch Angabe der Attributwerte für Farbe (color), Symbol (marker) und Linientyp (linestyle):

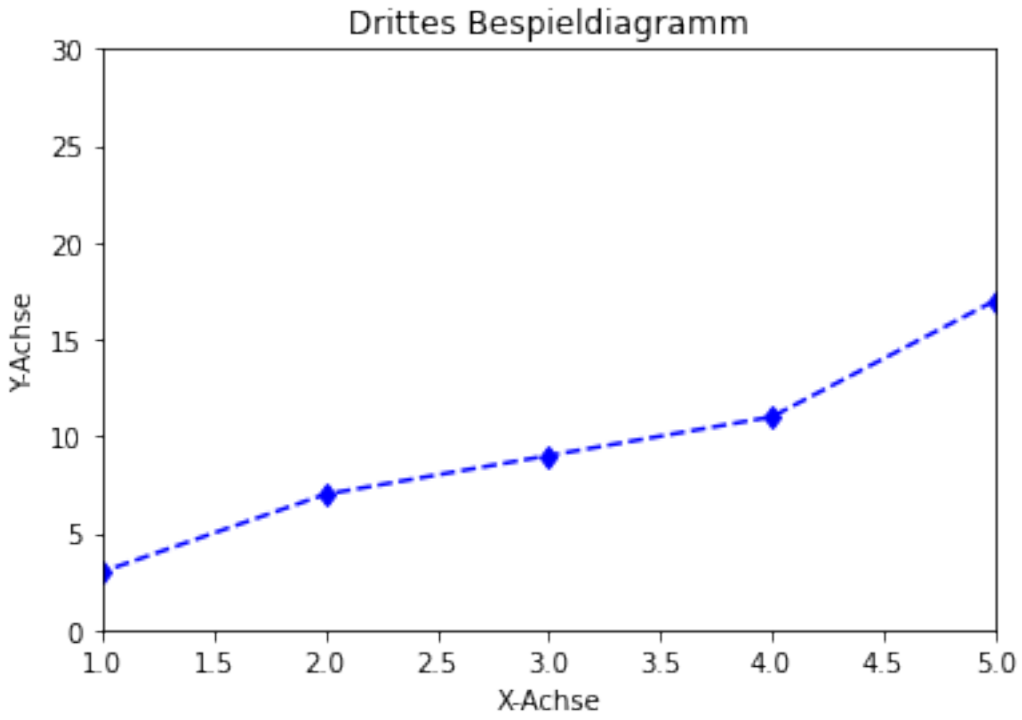
[4]:
```
%matplotlib inline

# Variante 2
x1 = np.array([1,3,7,9,11,17,20])
x2 = np.array([5,2,9,1,12,18,8])
x3 = np.array([9,1,2,7,2,8,3])

plt.plot(x1, color="blue", marker="d", linestyle="--")
plt.plot(x2, "*y", color="firebrick", marker="o", linestyle="-")
plt.plot(x3, "vk-.", color="chocolate", marker="x", linestyle=":")
plt.title("Zweite Bespieldiagramm")
plt.xlabel("X-Achse")
plt.ylabel("Y-Achse")
plt.show()
```



Wir können die Wertebereiche der Achsen definieren:

[5]:
```
%matplotlib inline

# Wertebereiche der Achsen
x1 = np.array([1,3,7,9,11,17,20])

xmin, xmax, ymin, ymax = 1,5,0,30
```

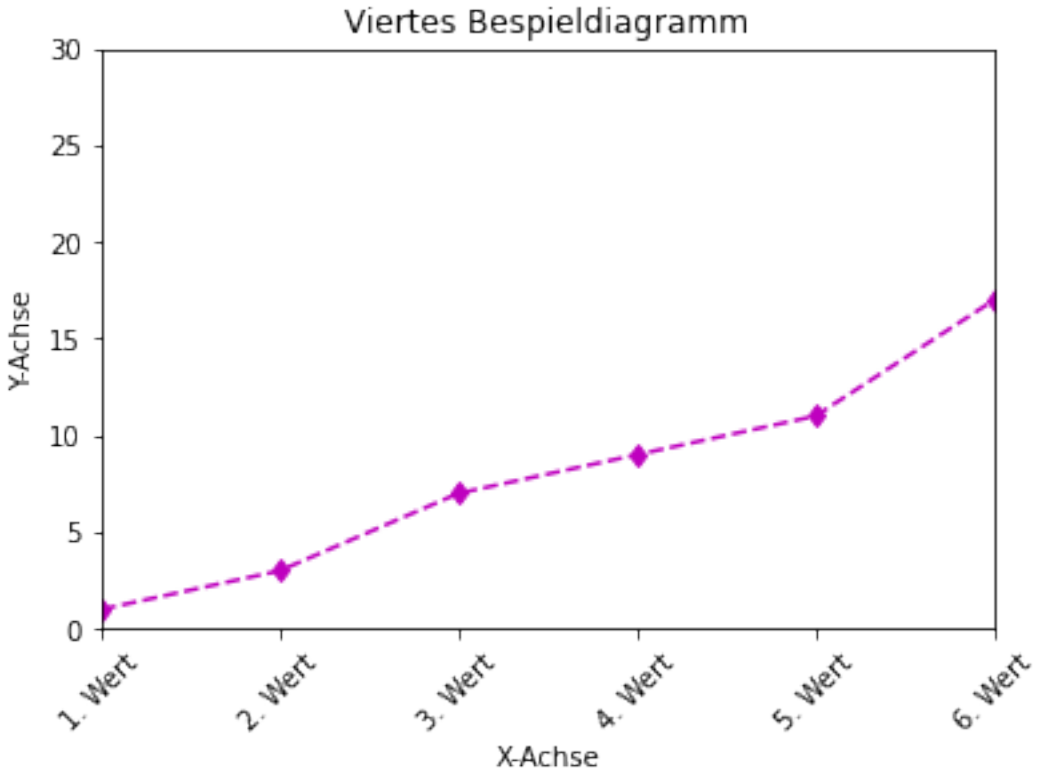
```
plt.plot(x1, color="blue", marker="d", linestyle="--")
plt.title("Drittes Bespieldiagramm")
plt.xlabel("X-Achse")
plt.ylabel("Y-Achse")

plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```



Mit *xticks* können wir auch die Bezeichnungen auf der X-Achse ändern.

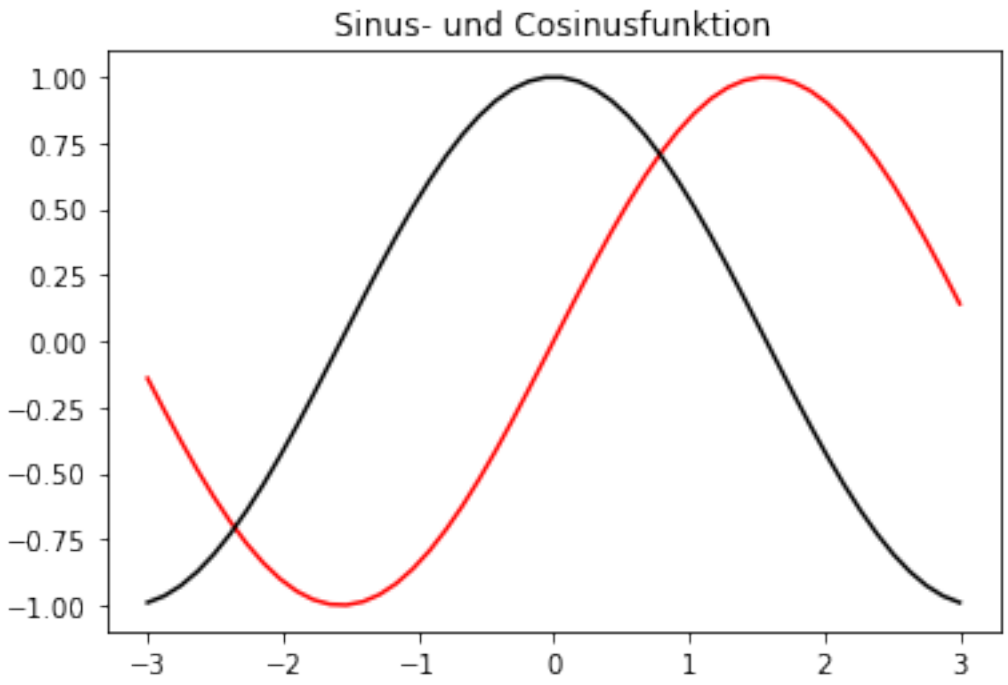
[6]:
```
%matplotlib inline

# Werte auf der X-Achse definieren
x1 = np.array([1,3,7,9,11,17,20])

xmin, xmax, ymin, ymax = 1,5,0,30

plt.plot(x1, color="m", marker="d", linestyle="--")
plt.title("Viertes Bespieldiagramm")
plt.xlabel("X-Achse")
plt.ylabel("Y-Achse")

plt.axis([xmin, xmax, ymin, ymax])
```

```
plt.xticks(np.arange(6), ("1. Wert", "2. Wert", "3. Wert", "4. Wert", "5.␣
 ↪Wert", "6. Wert"),
           rotation=45)
plt.show()
```



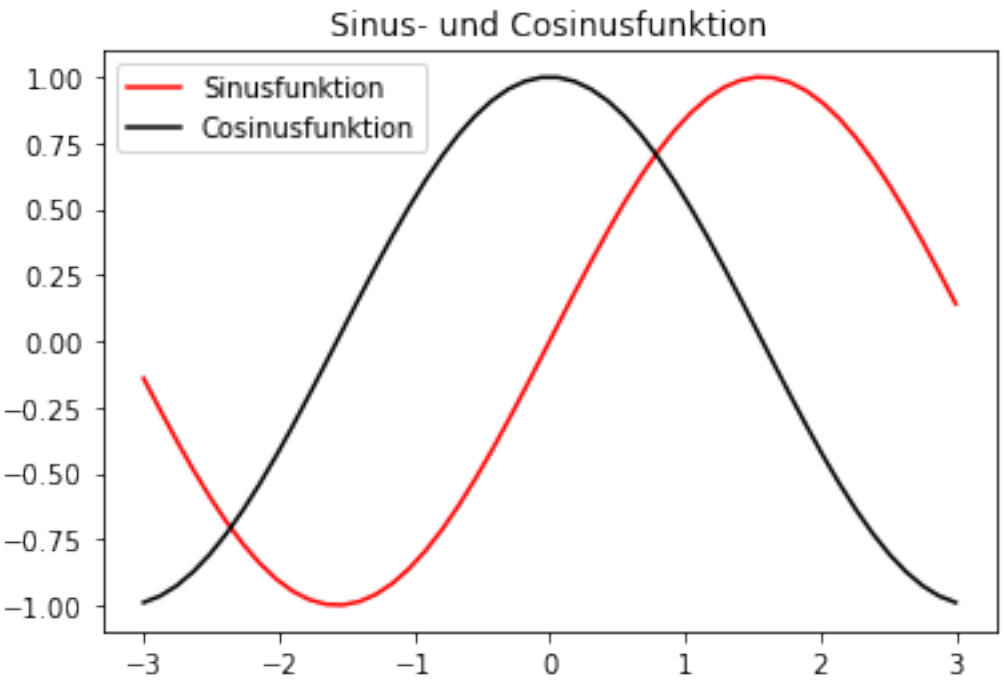
Im folgenden verwenden wir Werte für die x- wie auch für die y-Achse. Angenommen wir wollen die Sinus- und die Cosinus-Funktion in einem Diagramm darstellen:

[7]:
```
%matplotlib inline

# Sinus- und Cosinusfunktion

x = np.linspace(-3, 3, 50)
s = np.sin(x)
c = np.cos(x)

plt.plot(x,s, c="r")
plt.plot(x,c, c="k")
plt.title("Sinus- und Cosinusfunktion")
plt.show()
```



Damit wir die beiden Funktionen auseinanderhalten können fügen wir eine Legende hinzu:

[8]:
```
%matplotlib inline

# Sinus- und Cosinusfunktion

x = np.linspace(-3, 3, 50)
s = np.sin(x)
c = np.cos(x)

plt.plot(x,s, c="r", label="Sinusfunktion")
plt.plot(x,c, c="k", label="Cosinusfunktion")
plt.title("Sinus- und Cosinusfunktion")
plt.legend()
plt.show()
```

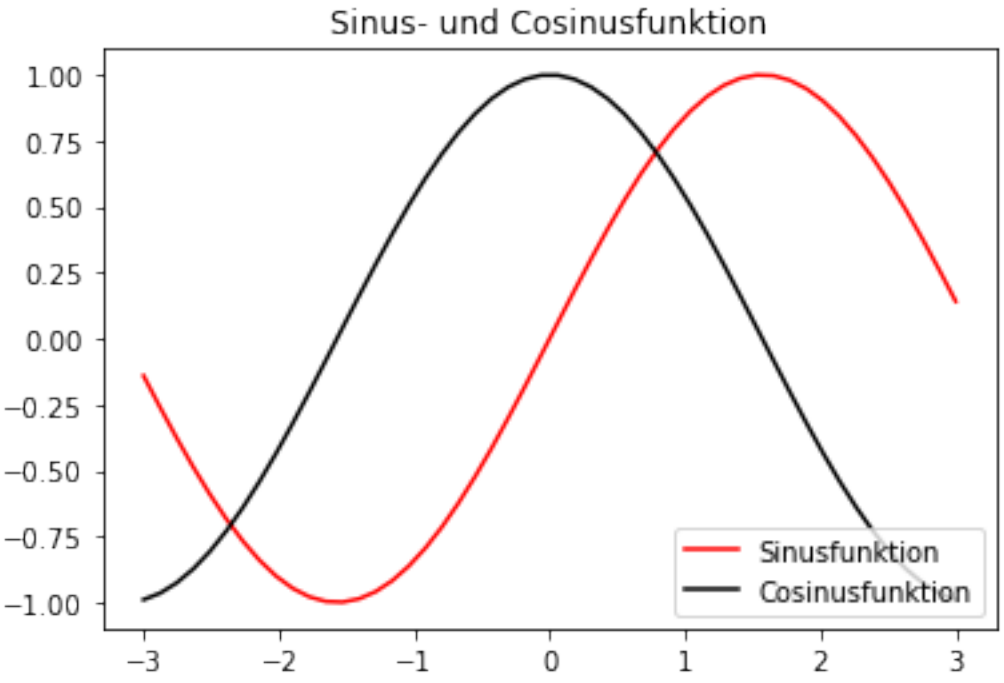


Matplotlib sucht sich hier selbst aus, wo die Legende eingefügt wird (wo die Legende möglichst nichts überdeckt). Die Position kann aber auch definiert werden, in man die Location angibt. Diese kann entweder über eine textliche Beschreibung oder über einen Zahlencode definert werden. Die am häufigsten verwendeten Positionen:

Positions-String	Zahlencode
best	0
upper right	1
upper left	2
lower left	3
lower right	4

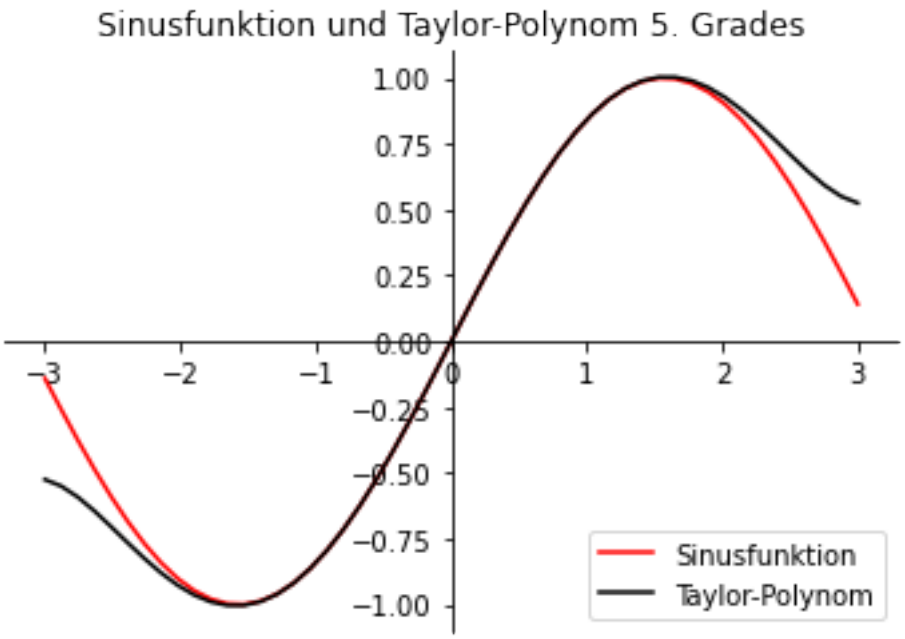
[9]:
```
%matplotlib inline

# Sinus- und Cosinusfunktion

x = np.linspace(-3, 3, 50)
s = np.sin(x)
c = np.cos(x)

plt.plot(x,s, c="r", label="Sinusfunktion")
plt.plot(x,c, c="k", label="Cosinusfunktion")
plt.title("Sinus- und Cosinusfunktion")
plt.legend(loc="lower right")
```

```
plt.show()
```



Falls man noch die üblichen Achsen einzeichnen möchte, verwendet man am besten die *spines*-Methode. Damit kann man die standardmäßigen Achsen am oberen und rechten Rand (top und right) ausblenden, indem man die Position mit *set_position* auf *none* setzt und stattdessen die untere und linke Achse mit *zero* auf 0 setzt.

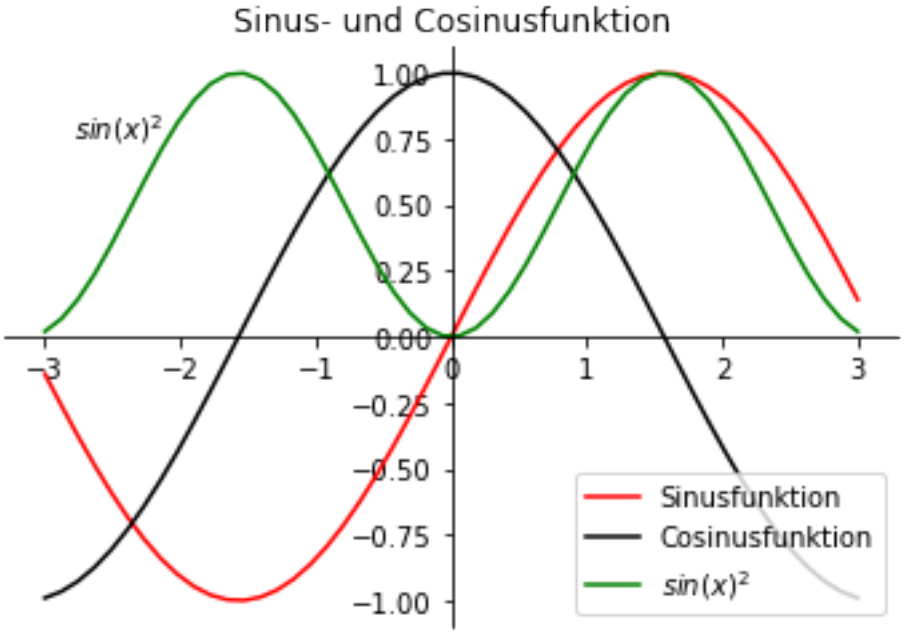
[10]:
```
%matplotlib inline

# Sinus- und Cosinusfunktion

x = np.linspace(-3, 3, 50)
s = np.sin(x)
t = x-1/6*x**3 + 1/120*x**5

plt.plot(x,s, c="r", label="Sinusfunktion")
plt.plot(x,t, c="k", label="Taylor-Polynom")
plt.title("Sinusfunktion und Taylor-Polynom 5. Grades")
plt.legend(loc="lower right")
ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
plt.show()
```



Mit *text* kann man Beschriftungen in ein Diagramm einfügen. Dazu gibt man die X- und Y-Koordinate an. Wenn man einen Python-Raw-String (mir vorangestelltem r) übergibt, sind auch $LaTeX$-Formatierung möglich!

[11]:
```
%matplotlib inline

# Sinus- und Cosinusfunktion

x = np.linspace(-3, 3, 50)
s = np.sin(x)
c = np.cos(x)
f = np.sin(x)**2



plt.plot(x,s, c="r", label="Sinusfunktion")
plt.plot(x,c, c="k", label="Cosinusfunktion")
plt.plot(x,f, c="g", label=r"$sin(x)^2$")

plt.title("Sinus- und Cosinusfunktion")
plt.legend(loc="lower right")
ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
```

```
plt.text(-2.8, 0.75, r"$sin(x)^2$")
plt.show()
```



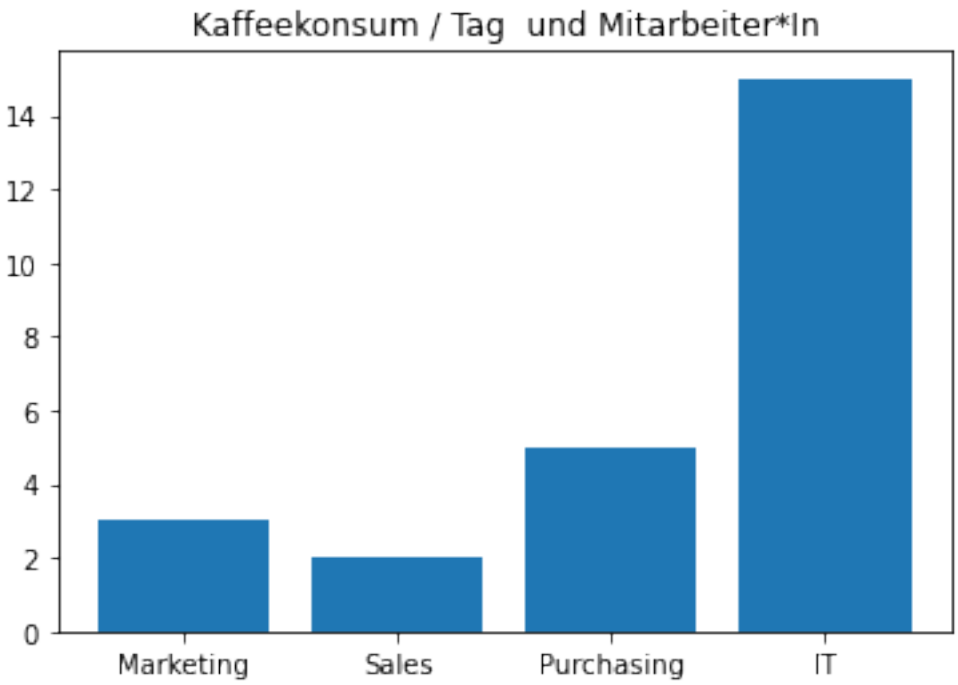
1.2 Weitere Diagrammarten

Die folgenden Beispiele zeigen weitere Diagrammtypen:

- Balkendiagramm
- Punktediagramm (Scatterplot)
- Histogramm
- Tortendiagramm
- Boxplot

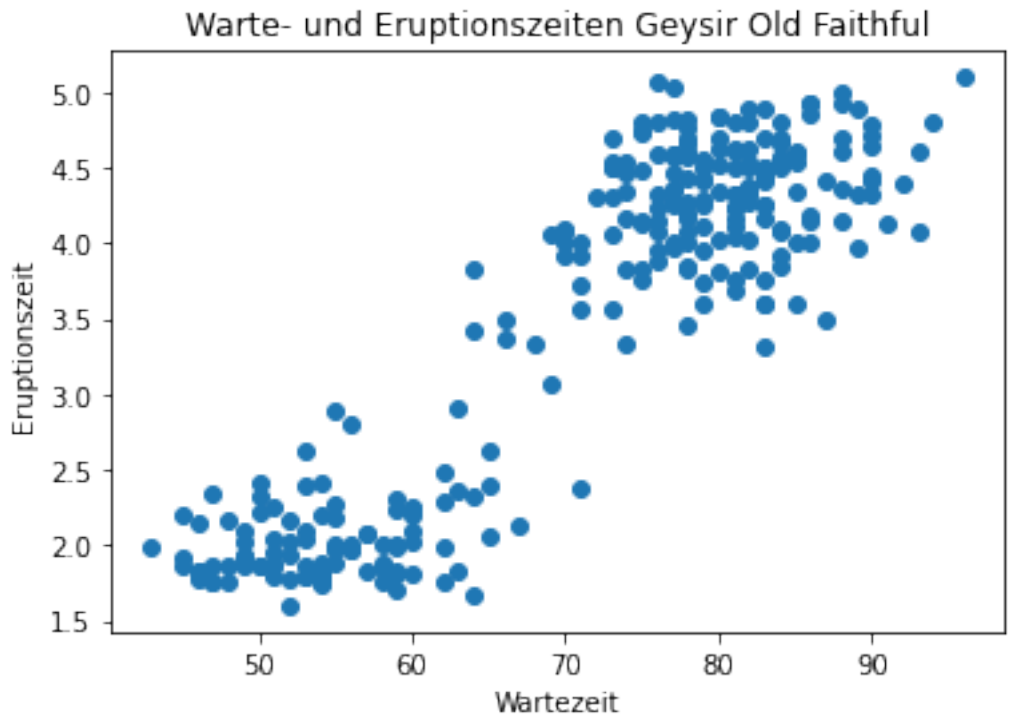
1.2.1 Balkendiagramm

[12]:
```
%matplotlib inline

# Balkendiagramm
y = np.array([3, 2, 5, 15])
lables = ["Marketing", "Sales", "Purchasing", "IT"]

plt.bar(lables, y)
plt.title("Kaffeekonsum / Tag  und Mitarbeiter*In")
plt.show()
```



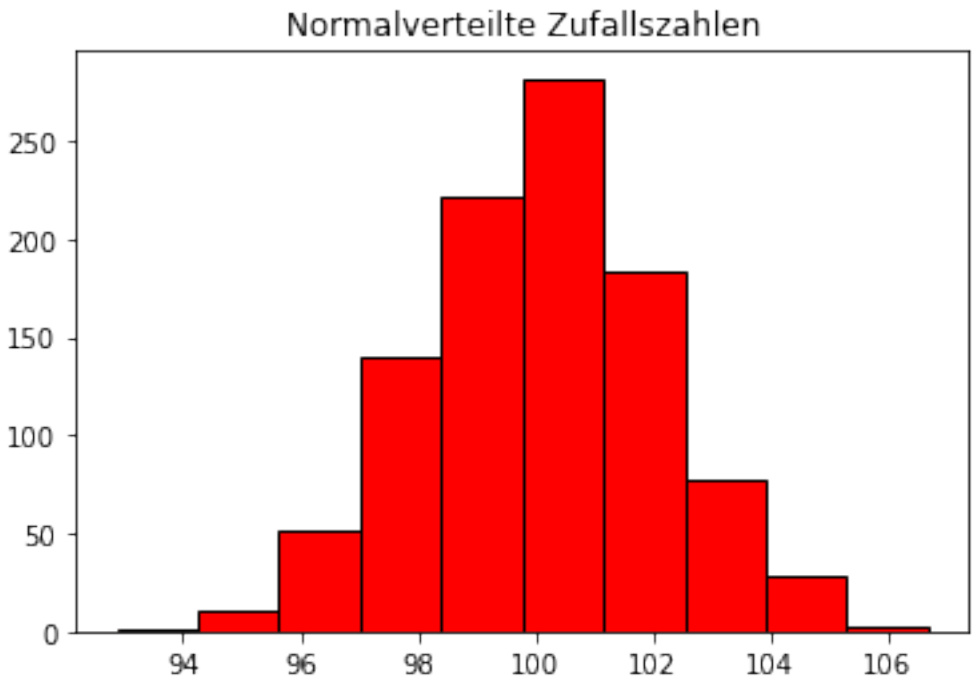
1.2.2 Punktediagramm (Scatterplot)

[13]:
```
%matplotlib inline
import pandas as pd

# Punktediagramm

url = "https://raw.githubusercontent.com/troescherw/datasets/master/faithful.
 ↪csv"
df = pd.read_csv(url, delimiter=";")
plt.scatter(df.waiting, df.eruptions)
plt.title("Warte- und Eruptionszeiten Geysir Old Faithful")
plt.xlabel("Wartezeit")
plt.ylabel("Eruptionszeit")
plt.show()
```



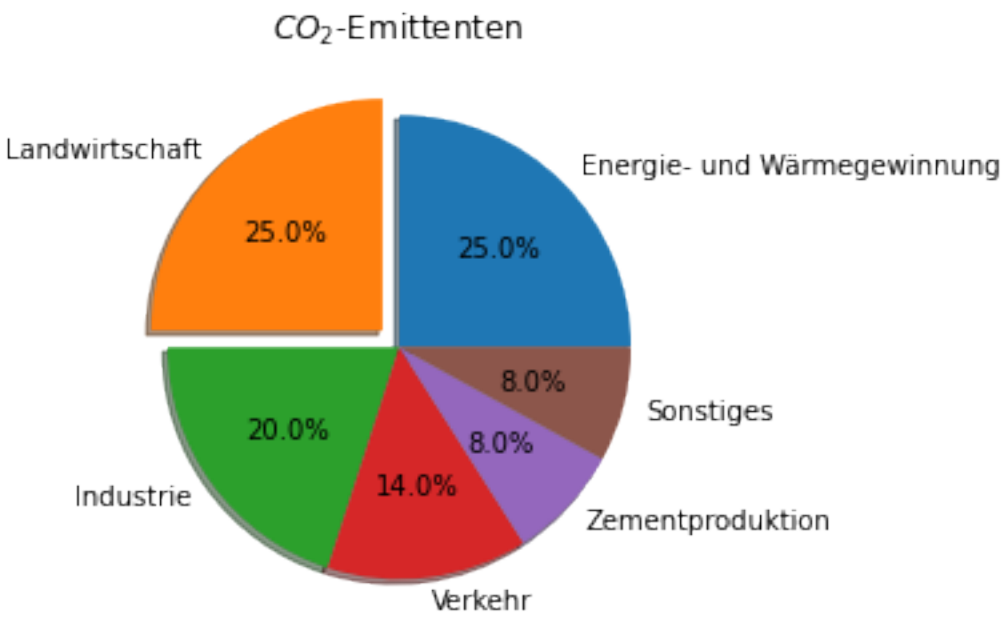
1.2.3 Histogramm

[14]:
```
%matplotlib inline
import numpy as np

# Histogramm

normalverteilt = np.random.normal(100,2,1000)
plt.hist(normalverteilt, edgecolor="k", color="red")
plt.title("Normalverteilte Zufallszahlen")
plt.show()
```



1.2.4 Tortendiagramm

[15]:
```
%matplotlib inline
import numpy as np

# Tortendiagramm

co2_emission = [25, 25, 20, 14, 8,8]

emittenten = ["Energie- und Wärmegewinnung", "Landwirtschaft", "Industrie",
              "Verkehr", "Zementproduktion", "Sonstiges"]

explodes = [0,0.1,0,0,0,0] #Bruchteil des Radius, mit dem ein Keil versetzt wird

plt.pie(co2_emission,  labels=emittenten, explode=explodes, autopct="%1.1f%%",
        shadow=True)
plt.title(f"$CO_2$-Emittenten")
plt.text(-1.5,-1.5, "Quelle: Umweltbundesamt")
plt.show()
```

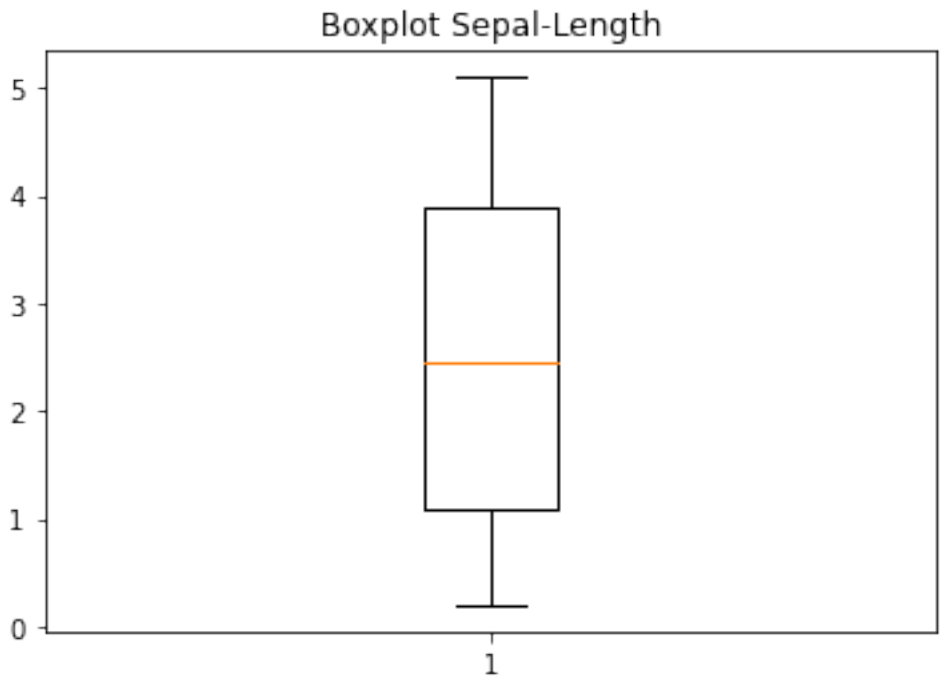


Quelle: Umweltbundesamt

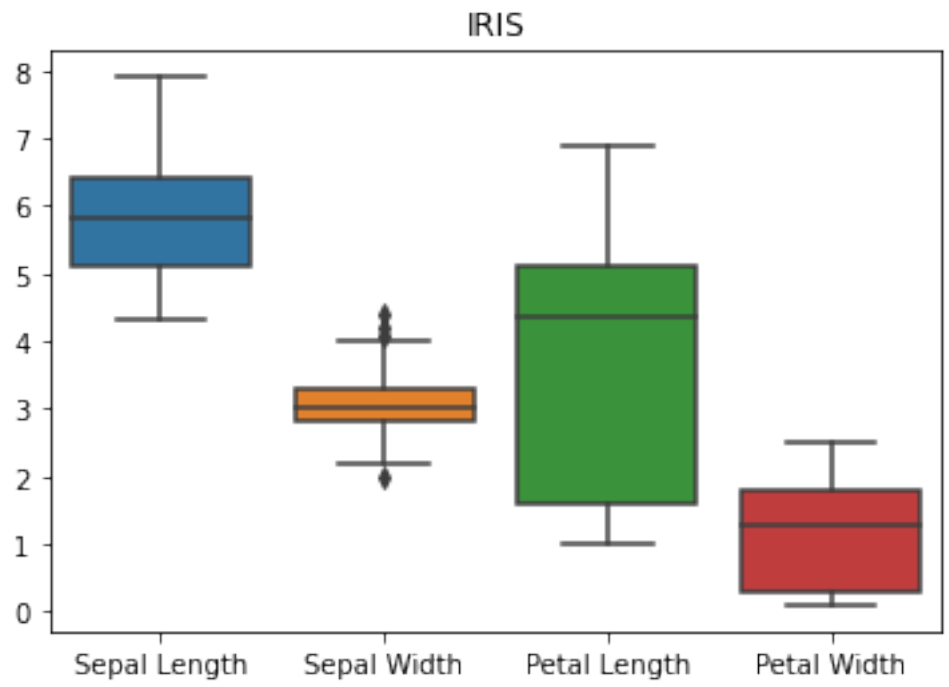
1.2.5 Boxplot

Ein einzelnes Boxplot-Diagramm kann mit Matplotlib recht einfach erstellt werden. Schwieriger wird es, wenn man mehrere Boxplots in einem Diagramm gegenüberstellen möchte. Das geht, ist aber (unnötig) kompliziert. Das geht viel einfacher mit *seaborn*!

[16]:
```
%matplotlib inline
from sklearn.datasets import load_iris

iris = load_iris().data
plt.boxplot(iris[0])
plt.title("Boxplot Sepal-Length")
plt.show()
```



[17]:
```
%matplotlib inline
import seaborn as sns
import pandas as pd
from sklearn.datasets import load_iris

iris = pd.DataFrame(load_iris().data)
iris.columns=["Sepal Length", "Sepal Width", "Petal Length", "Petal Width"]
sns.boxplot(data=iris)
plt.title("IRIS")
plt.show()
```



1.3 Mehrere Diagramme darstellen

Es können mehrere Diagramme in einer Grafik dargestellt werden. Hierfür gibt es mehrere Möglichkeiten - zwei davon möchte ich aufzeigen. In beiden bedient man sich der Methode *subplots*.

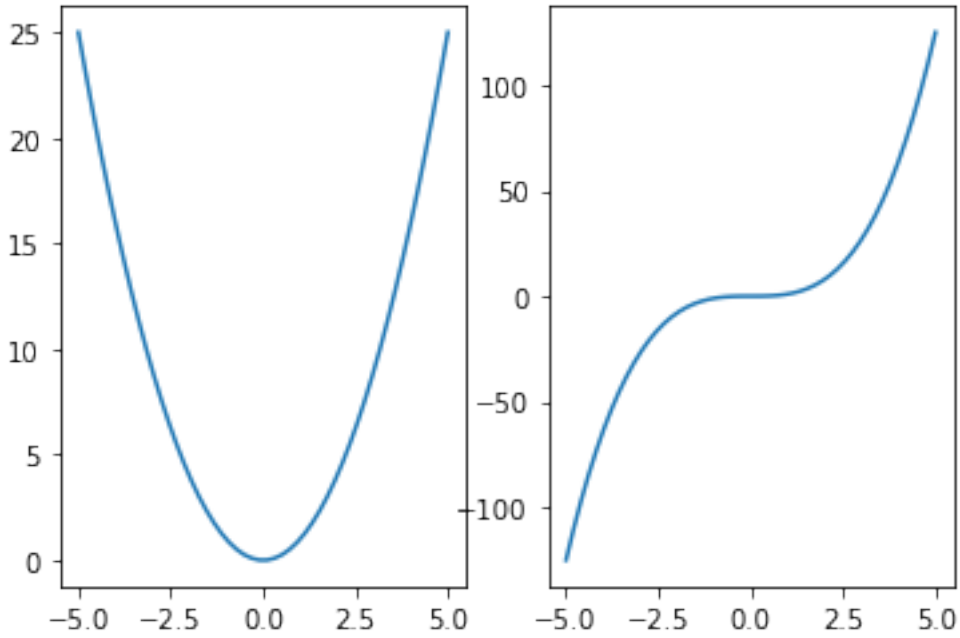
[18]:
```
# Daten für 4 Diagramme

x = np.linspace(-5,5,50)
f1 = x**2
f2 = x**3
f3 = x**2 * np.cos(x)
f4 = x**3 * np.sin(x)
```

Mit der ersten Methode wird mit *subplots* angegeben, wie viele Diagramme man darstellen möchte. Auch hierfür gibt es wieder zwei Möglichkeiten: Mit *nrows* und *ncols* oder man gibt die jeweiligen Zahlen (sofern sie <10 sind) direkt hintereinandgeschrieben als Argument an. Die dritte Zahl gibt an, in welches "Feld" man plotten möchte, dabei wird von oben links bis unten rechts bei 1 beginnend gezählt. Das Feld oben links hat also den Wert 1, bei einem Diagramm mit z.B. 2 Zeilen und 2 Spalten würde mit der Zahl 4 das Diagramm im Feld unten rechts eingetragen werden.

Angenommen wir wollen 2 Diagramme darstellen, und zwar in einer Zeile und 2 Spalten:

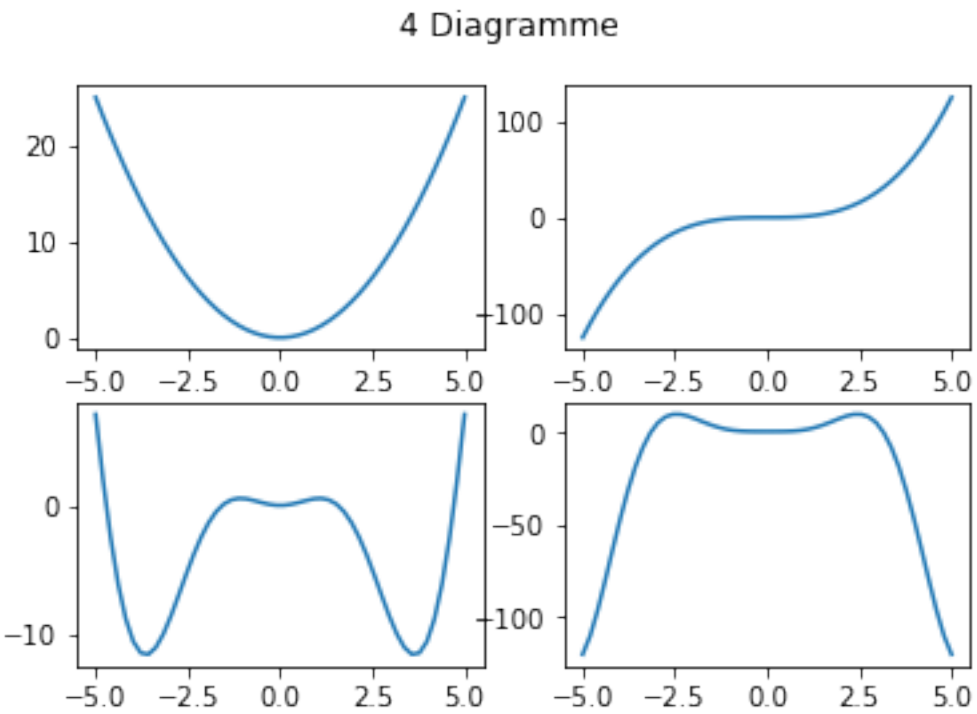
[19]:
```
%matplotlib inline

plt.subplot(1,2,1) #In Klammern: 1 Zeile, 2 Spalten, Diagramm 1
plt.plot(x,f1)
```

```
plt.subplot(1,2,2) #In Klammern: 1 Zeile, 2 Spalten, Diagramm 2
plt.plot(x,f2)
plt.show()
```



Oder alle 4 Diagramme in zwei Zeilen und zwei Spalten:

[20]:
```
%matplotlib inline

plt.subplot(2,2,1) #In Klammern: 2 Zeilen, 2 Spalten, Diagramm 1
plt.plot(x,f1)
plt.subplot(2,2,2) #In Klammern: 2 Zeilen, 2 Spalten, Diagramm 2
plt.plot(x,f2)
plt.subplot(2,2,3) #In Klammern: 2 Zeilen, 2 Spalten, Diagramm 3
plt.plot(x,f3)
plt.subplot(2,2,4) #In Klammern: 2 Zeilen, 2 Spalten, Diagramm 4
plt.plot(x,f4)
plt.suptitle("4 Diagramme")
plt.show()
```



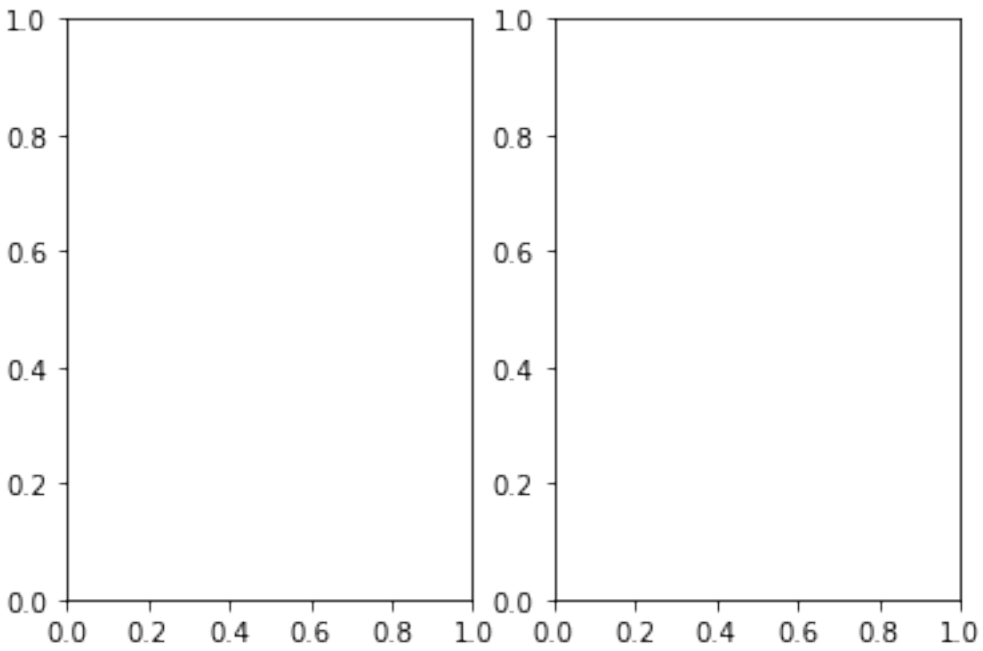
Die zweite Möglichkeit entspricht der Objetorientierten Denkweise und wird häufig bevorzugt, weil sie flexibler ist. Wenn wir ein Diagramm mit *plt* erstellen, so werden 2 Objekte erstellt: Ein Objekt, das als "Rahmen" für unsere Diagramme dient, genannt *figure*, sowie der "Platzzahlter" für das eigentliche Diagramm, genannt *axes* (nicht zu verwechseln mit den Achsen im Diagramm!).

Mit *subplots* (im Gegensatz zum ersten Beispiel im Plural, also mit einem "s"!) können nun (immer) ein *figure*-Objekt und - je nach Bedarf - mehrere *axes*-Objekte erstellt werden.

Im Beispiel erstellen wir ein *figure*-Objekt und zwei *axes*-Objekte. Die *axes*-Objekte werden in einem Numpy-Array zurückgegeben:

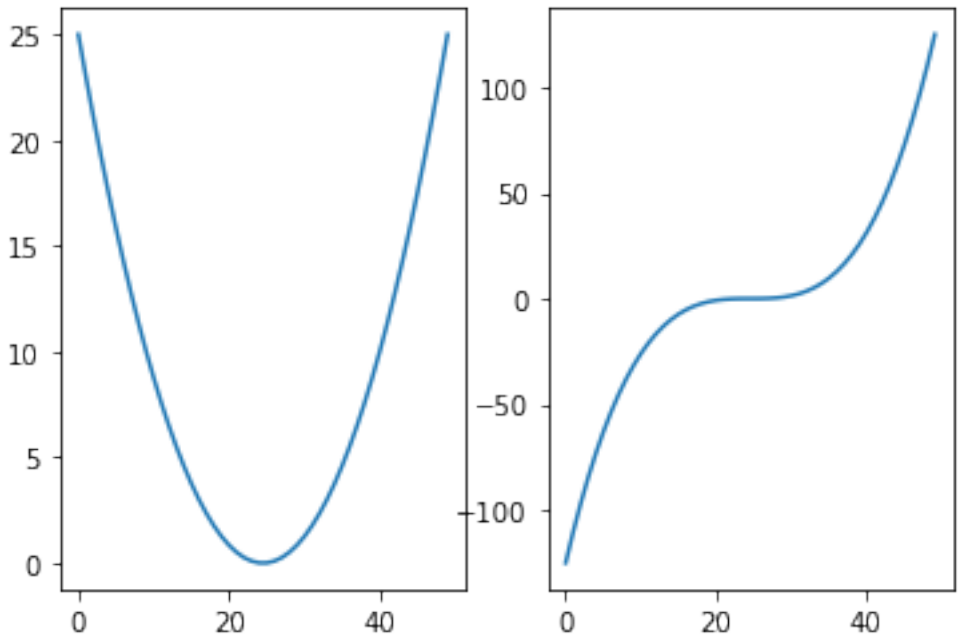
[21]:
```
%matplotlib inline

fig, ax = plt.subplots(nrows=1, ncols=2)
print(type(ax))
print(ax.shape)
```

```
<class 'numpy.ndarray'>
(2,)
```



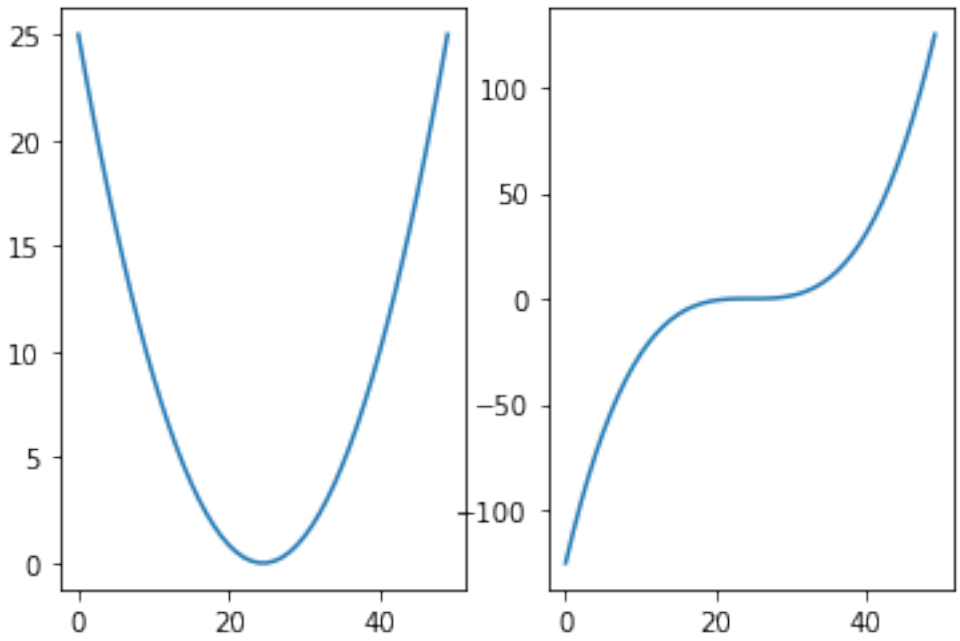
In diese beiden Objekte können wir nun unsere Inhalte, also die Diagramme, einfügen:

[22]:
```
%matplotlib inline

fig, ax = plt.subplots(nrows=1, ncols=2)
ax[0].plot(f1)
ax[1].plot(f2)
plt.show()
```



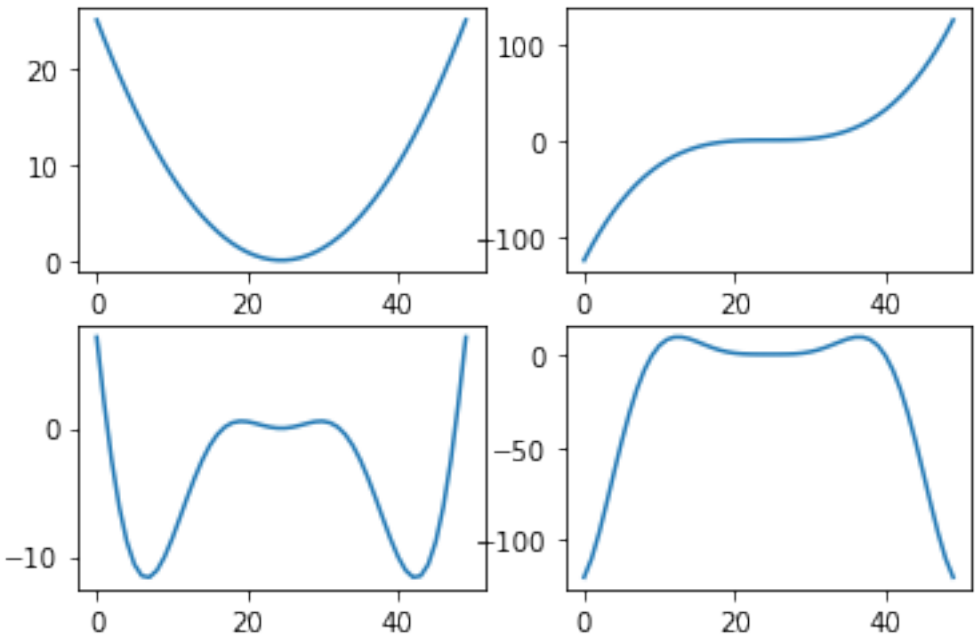
Alterativ können wir den *axes*-Objekten auch gleich Variablennamen zuweisen, indem wir *unpacking* verwenden:

[23]:
```
%matplotlib inline

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
ax1.plot(f1)
ax2.plot(f2)
plt.show()
```



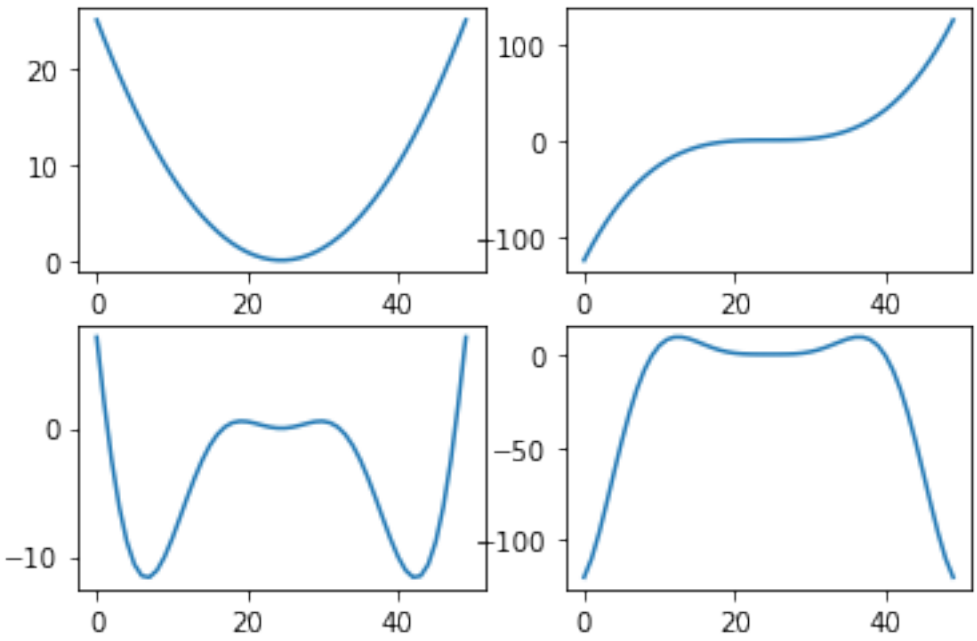
Nun wollen wir wieder 4 Diagramme einfügen. Bei der Angabe von 2 Zeilen und 2 Spalten erhalten wir also 2 x 2 *axes*-Objekte zurück, die wir in einem Tupel speichern können:

[24]:
```
%matplotlib inline

fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(nrows=2, ncols=2)
ax1.plot(f1)
ax2.plot(f2)
ax3.plot(f3)
ax4.plot(f4)
plt.show()
```



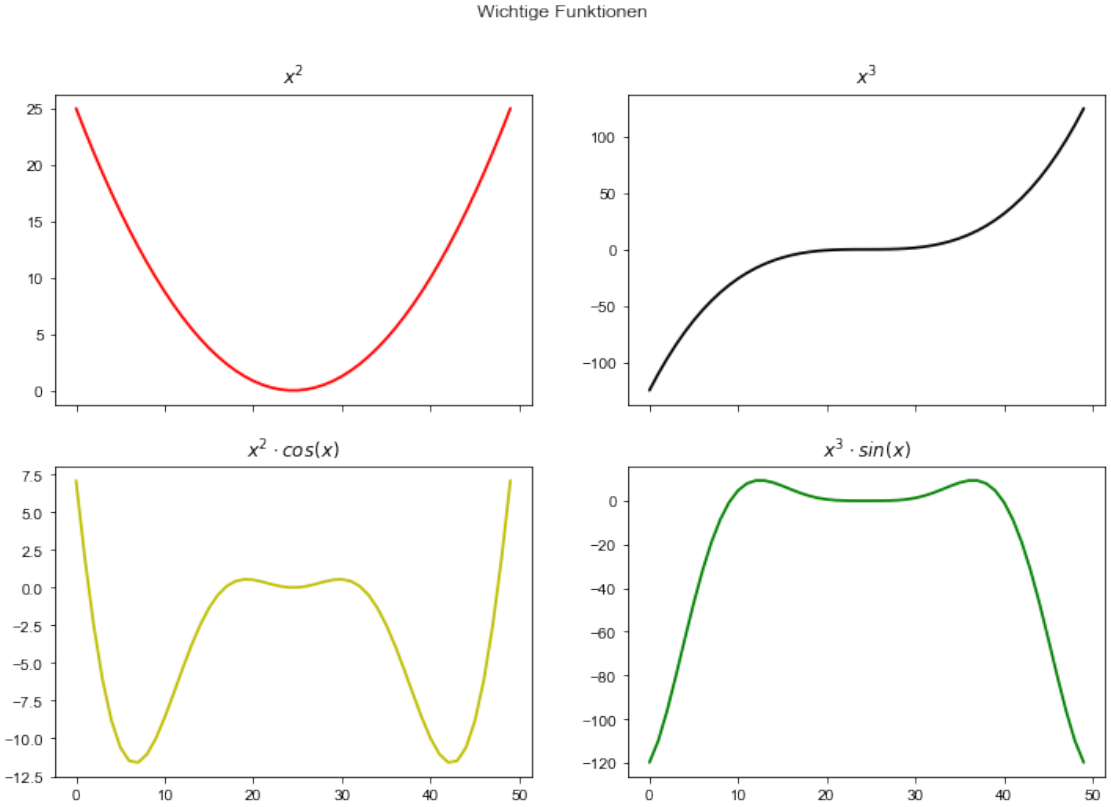
Alternativ hätten wir natütürlich auch wieder die Indizes verwenden können, nun natürlich in einem 2-dimensionalen Array:

[25]:
```
%matplotlib inline

fig, ax = plt.subplots(nrows=2, ncols=2)
ax[0,0].plot(f1)
ax[0,1].plot(f2)
ax[1,0].plot(f3)
ax[1,1].plot(f4)
plt.show()
```



Da die X-Achse bei allen Diagramm gleich ist, können wir die x-Achse bei den oberen Diagrammen auch ausblenden lassen, indem wir *sharex* auf *True* setzen. Wir können für jedes Diagramm auch individuelle Titel einfügen. Aber Achtung: Die Methode hierfür heißt **nicht** *title()*, sondern *set_title()*! Einen gemeinsamen Titel gibt man mit *suptitle* an. Mit *plt.rcparams* kann man in einem Notbook auch noch die Größe der Grafik definieren.

[26]:
```
%matplotlib inline

plt.rcParams['figure.figsize'] = [12, 8]

fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(nrows=2, ncols=2, sharex=True)
plt.style.use("seaborn")
ax1.plot(f1, color="r")
ax2.plot(f2, color="k")
ax3.plot(f3, color="y")
ax4.plot(f4, color="g")

ax1.set_title(r"$x^2$")
ax2.set_title(r"$x^3$")
ax3.set_title(r"$x^2 \cdot cos(x)$")
ax4.set_title(r"$x^3 \cdot sin(x)$")
plt.suptitle("Wichtige Funktionen")
plt.show()
```


Wichtige Funktionen
x^2
x^3
$x^2 \cdot cos(x)$
$x^3 \cdot sin(x)$