INFI-IS

5. Jahrgang

...wird laufend erweitert

Albert Greinöcker

HTL Anichstraße

1. Dezember 2024

CCA – COMPETENCE CENTRE
HTL Anichstraße

Überblick

Ein Einblick in Data Science:

- Erste Datenanalyse und Visualisierung mit `NumPy`, `Pandas`, `Matplotlib`, `Plotly`
- Statistische Auswertung von Datensätzen inkl. Signifikanztests
- Reporting bzw. interaktive Bereitstellung der Auswertungen
- Einführung in Machine Learning mit `KERAS`
- Erweiterte Konzepte von Datenbanken, Umgang mit großen Datenbeständen und un- bzw. semistrukturierten Daten

Vorteile von `PyCharm` für Data Science in Python:

- Scientific Mode: Man wird gefragt sobald z.B: `NumPy` verwendet wird
  - Es werden die Plots (Visualisierung der Daten) angezeigt
  - Es werden die Daten als Tabelle angezeigt
  - Es wird zur aktuell markierten Funktion die Doku mit Parameter usw. angezeigt
- Es besteht die Möglichkeit für "Execute Selection in Python Console", um nicht immer das komplette File ausführen zu müssen.
  - Entweder über das Kontextmenü oder `CTRL+ALT+E`
  - Alle bereits eingegebenen Befehle werden gemerkt (also auch imports)

IDE-Alternativen:

- `Visual Studio Code` (mit Data Science-Extensions): `https://code.visualstudio.com/docs/datascience/data-science-tutorial`)
- JetBrains `DataSpell`: `https://www.jetbrains.com/de-de/dataspell/`

Was ist NumPy?

- Abkürzung für `Numerical Python`
- Bildet die Basis für viele Python-Projekte (Vor allem zum Erzeugen von Arrays)
- Verwaltet multi-dimensionale Arrays und stellt Funktionen dafür bereit
- Kann mit großen Datenmengen umgehen
- Sehr schnell, da wesentliche Teile davon in `C` ausprogrammiert sind
- Geht von den Berechnungsmöglichkeiten her relativ weit (werden wir nicht ausschöpfen)
- Basis-Datenstruktur ist ein `ndarray` (n-dimensionales Array) - oder auch NumPy Array

```
1 # Hat sich so eingebürgert dass als np importiert wird
2 import numpy as np
3 a = np.array([1,2,3]) # So wird ein 1-dimensionales Array angelegt
4 print(a) # Ausgabe des Arrays
5 print(a.shape) # Wie viele Werte auf den einzelnen Dimensionen
6 print(len(a.shape)) # Wie viele Dimensionen
7 print(a.dtype) # Um welchen Datentyp handelt es sich?
```

- Ausführlicheres Tutorial: `https://www.tutorialspoint.com/numpy/index.htm`

Hilfreiche Funktionen für die Erzeugung von (mehrdimensionalen) Arrays

Eindimensionale Arrays:

```
a1 = np.arange(10) # [0 1 2 3 4 5 6 7 8 9]
a11 = np.arange(10, 20, 0.5) # (start, stop, step)
# 5 Werte zwischen 1 und 2 mit gleichem Abstand:  [1. 1.25 1.5 1.75 2.]
a2 = np.linspace(1, 2, 5)
a3 = np.ones(10) # lauter 1en
a4 = np.zeros(10) # lauter 0en
a5 = np.random.rand(10) # Zufallszahlen zwischen [0,1[]
# 5 Ganzahlen zufällig gezogen zwischen [10 und 20[]
a6 = np.random.randint(10,20,5) # Gleichverteilt (alle Zahlen haben die gleiche
   Chance, gezogen zu werden)
```

Mehrdimensionale Arrays: Hier werden einfach weitere Dimensionen angegeben

```
a5 = np.random.rand(3,2,4)
```

Erzeugt...

```
[[[0.9502729  0.47978706 0.80420501 0.69607102]
[0.12001849 0.07467159 0.74332051 0.90530865]]

[[0.7824477  0.29988021 0.39840717 0.02550705]
[0.10465624 0.37901605 0.01341696 0.2704336 ]]

[[0.69887726 0.79655702 0.77091248 0.6599289 ]
[0.97888692 0.11476978 0.86355308 0.44119605]]]
```

Umgang mit Datentypen

Der Datentyp kann bei der Erzeugung gesetzt werden:

```
d1 = np.array([1,2,3], dtype=complex) # [1.+0.j 2.+0.j 3.+0.j]
```

```
d1 = np.array([1,2,3], dtype=str) # ['1' '2' '3']
```

Auch bei bereits erzeugten Arrays kann der Typ geändert werden:

```
d2 = d2.astype(int) #[1 2 3]
# copy=false : in das gleiche Array
# casting='safe' : Es gibt einen Fehler wenn casting nicht funktioniert
ad1 = ad1.astype(str, copy=False, casting='safe')
# die nicht built-in Datentypen:
db = np.array(['1995-10-28 23:55', '2020-01-18 23:01'])
db = db.astype('M') # Datetime
```

Wichtigste verfügbare Datentypen:

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string

Weitere Hilfreiche Funktionen

Ändern der Dimension:

```
dr1 = np.zeros(10)
dr1 = dr1.reshape(2,5)
print(dr1)
```

Erzeugt...

```
[[0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]]
```

Einfache Rechenoperationen: Operatoren können auf ganze Arrays angewandt werden

```
 r1 = np.arange(10) * 10
```

Erzeugt...

```
[0 10 20 30 40 50 60 70 80 90]
```

Zugriff auf einzelne Elemente: So wie in Python gewohnt

```
a1 = np.arange(10) #[0 1 2 3 4 5 6 7 8 9]
print(a1[1]) # 1
print(a1[1:4]) # [1 2 3]
print(a1[-1]) #9 Beginnt von hinten
print(a1[6:]) # [6 7 8 9]
```

Mehrdimensionaler Zugriff auf einzelne Elemente

Ändern der Dimension:

```
1  am = np.array([[1,2,3],[4,5,6],[7,8,9]])
2  print("Gesamte Matrix")
3  print(am)
4  print("Spalte 2")
5  print(am[:,1]) #Spalte 2
6  print("Zeile 2")
7  print(am[1,:]) #Zeile 2
8
9  print("Einzelne Werte")
10 print(am[[1,2],[0,1]]) #Positionen (1,0), (2,1)), also 4 und 8
```

Erzeugt...

```
Gesamte Matrix
[[1 2 3]
[4 5 6]
[7 8 9]]
Spalte 2
[2 5 8]
Zeile 2
[4 5 6]
Einzelne Werte
[4 8]
```

Abfragen von Array-Werten

Mittels Bedingungen: Diese Variante geht mit "normalen" Python Arrays nicht!

```
1          ab = np.arange(10) #[0 1 2 3 4 5 6 7 8 9]
2          print("ab < 5:")
3          print(ab[ab < 5])
4          print("~ab < 5:") #~ist 'not'
5          print(ab[~(ab < 5)])
```

Erzeugt...

```
ab < 5:
[0 1 2 3 4]
~ab < 5:
[5 6 7 8 9]
```

Fehlende Werte: Die Konstante `np.nan`

```
1          a = np.array([np.nan, 1,2,np.nan,3,4,5])
2          print a[~np.isnan(af)] # Filtert alle nicht fehlenden Werte
```

Komplexere Operationen auf Array-Werte

```
1 am = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

Wenn man nicht die Werte, sondern die Positionen haben möchte (`where`):

```
1 i = np.where(am > 5)
```

Ergibt...

```
(array([1, 2, 2, 2]), array([2, 0, 1, 2]))
```

Das kann wiederum als Index verwendet werden:

```
1 print(am[i]) # [6 7 8 9]
```

Sogar eine Zuweisung an alle diese Positionen ist dann möglich:

```
1 am[i] = 20
```

Ergibt:

```
[[ 1  2  3]
[ 4  5 20]
[20 20 20]]
```

Manipulation von Array-Werten mittels Funktionen

Diese Funktionen werden mit np (dem numpy import alias), nicht dem Objekt, aufgerufen.

Daten für die Beispiele unten:

```
1 am = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
```

Hier die wichtigsten Funktionen:

```
1 # Zeilenweise angehängt. axis=1 Spaltenweise angehängt
2 am = np.append(am,[[13,14,15,16]], axis=0)
3 # Ähnlich dazu insert, hier muss der Index wo eingefügt werden soll
    angegeben werden
```

```
1 #Zerlege die Matrix Spaltenweise in sub-Arrays der Größe 2
2 af2 = np.split(af,2 , 1)
```

```
1 #Lösche die 3. Zeile in af. Die neue Matrix wird zurückgegeben
2 af3 = np.delete(af, 2,0)
```

```
1 #Schmeisse die doppelten hinaus und gib Ergebnis als 1-dim Array zurück
2 af4 = np.unique(af)
```

```
1 #Mache aus einem mehrdimensionalen Array ein Eindimensionales
2 af5 = np.ravel(af4)
```

Weitere Funktionen und Details unter:
https://www.tutorialspoint.com/numpy/numpy_array_manipulation.htm

Kennwerte von Daten

```
1        a1 = np.random.normal(170,10, 1000) #Wird für die Beispiele
            verwendet
```

Lagemaße: Geben das Zentrum von Verteilungen an und dienen der Zusammenfassung von Daten

- `Mittelwert`: Summe / Anzahl
  `a1.mean()`
- `Median`: Wenn man die Werte sortiert ist es der Wert in der Mitte
  `np.median(a1)`
- `10% - Quantil`: links von diesem Wert sind die unteren 10 %
  `np.quantile(a1,0.1)`
- `unteres Quartil`: links von diesem Wert ist das untere Viertel
  `np.quantile(a1,0.25)`
- `Modus`: Der Wert der am öftesten Vorkommt

```
1                  from scipy import stats #dazu braucht man eine
                      weitere Bibliothek
2                  a2 = np.random.randint(10,20,500)
3                  m = stats.mode(a2)
4                  print("Modus", m.mode[0], m.count[0])
```

Streuungsmaße (Dispersionsmaße)

- `Standardabweichung`: Wie stark streuen die einzelnen Werte um den Mittelwert
  `np.std(a1)` 1x Standardabweichung: 68% der Werte 2x sind 95% 3x 99,7%, also z.B. mittleren 95%: `np.mean(a1) - np.std(a1)*2, np.mean(a1) + np.std(a1)*2`
- Spannweite (Range): max - min
  `a1.max() - a1.min()`

Umgang mit fehlenden Werten

- Wenn einzelne Zellen den Wert nan beinhalten, dann wird aus den Berechnungen auch `nan`.
- Möchte man, dass diese Werte ignoriert werden, dann gibt es spezielle Funktionen die mit `nan` beginnen, z.B.: `np.nanmean()`

```
1 print(np.nanmean(col1))
```

- Weitere Methoden, die mit `nan` umgehen können:
  - `np.nanmedian()`
  - `np.nanstd()`
  - `np.nanquantile()`
  - `np.nansum()`
- Es ist allerdings empfohlen, diese Methoden nicht unüberlegt zu verwenden und immer die `nan`'s zu ignorieren, denn oft ist es nicht beabsichtigt dass diese enthalten sind.
- Der Befehl, der für jeden Wert checkt es ein fehlender Wert ist: `np.isnan(col1)`
- So checkt man, ob irgendein fehlender Wert drin ist: `np.isnan(col1).any()`
- Eine Strategie könnte sein, die fehlenden Werte durch einen default-Wert zu ersetzen (das kann z.B. der Mittelwert sein, das kommt auf die Daten an).

```
1 col1[np.isnan(col1)] = -10
```

- Das ersetzen / behandeln der fehlenden Werte (bzw. die Auswahl der Strategie) ist keine triviale Angelegenheit, da gibt es teilweise Diskussionen in der Wissenschaft.

Daten aus Datei in NumPy importieren

- Das ist prinzipiell möglich, sollte aber eigentlich über Pandas gemacht werden. Mehr dazu später.
- Die Funktion dazu ist `genfromtxt`
- Es werden jede Menge Parameter angeboten, die bezgl. Festlegen der Datentypen nicht so funktionieren wie erwartet, deshalb folgende Vorgehensweise:
  1. Importieren des kompletten Datensatzes, mit den automatisch erkannten Datentypen
  2. Konvertieren in die gewünschten Datentypen nach dem Import (umständlich, macht aber weniger Kopfweh)
  3. Jede Spalte für sich aufbereiten und für die Analyse verwenden

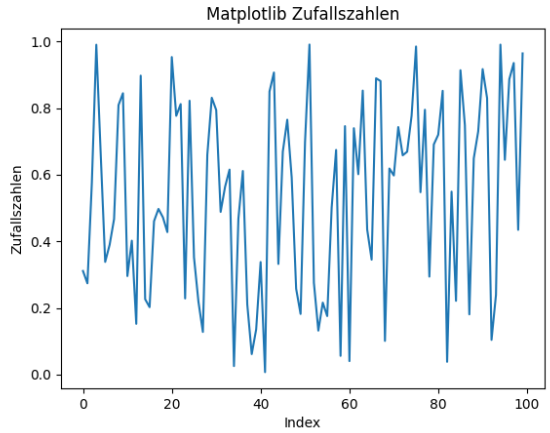
```
1 # Lade den Datensatz mit einem bestimmten Trennzeichen zwischen den
    Spalten, die erste Spalte wird nicht verwendet, da sie
    Spaltenüberschriften beinhaltet
2 d = np.genfromtxt('dataset.csv', delimiter=",", skip_header=1)
3 col1 = d[:,0] # Holen der ersten Spalte. Das ist dann ein ndarray
4 col1 = col1.astype('int') # Übertragen der Spalte in einen bestimmten
    Datentyp
```

Matplotlib - Allgemeines

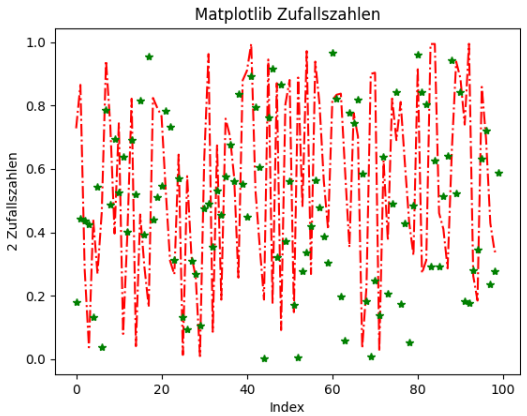
- Ist eine Bibliothek füt die Datenvisualisierung in Python
- Ist eher für die einfache Erstellung von Grafiken gedacht...
- `Plotly` besprechen wir später: kann schönere Grafiken, ist aber komplexer
- installation: `pip install matplotlib`
- import: `from matplotlib import pyplot as plt`
- Ausführliche Doku: `https://matplotlib.org/`

Grundaufbau:

```
1  x = np.arange(100)
2  y = np.random.rand(100)
3  plt.title("Matplotlib Zufallszahlen")
4  plt.xlabel("Index")
5  plt.ylabel("Zufallszahlen")
6  plt.plot(x,y)
7  # Speichern der Datei im aktuellen Arbeitsverzeichnis
8  plt.savefig("m1.png")
9  # Es geht ein viewer auf und zeigt den Plot. Nicht vor savefig
       aufrufen!
10 plt.show()
```



Matplotlib - mehrere Plots



Es besteht die Möglichkeit, mehrere Plots in eine Grafik zu geben:

```
y2 = np.random.rand(100)
plt.title("Matplotlib Zufallszahlen")
plt.xlabel("Index")
plt.ylabel("2 Zufallszahlen")
plt.plot(x,y, "r-.") # durchgägngige Linie in rot
plt.plot(x,y2,"g*") # Sterne in grün
plt.savefig("m2.png")
plt.show()
```

Kürzel für Farben:

- `'b'` Blue
- `'g'` Green
- `'r'` Red
- `'c'` Cyan
- `'m'` Magenta
- `'y'` Yellow
- `'k'` Black
- `'w'` White

Kürzel für Punkte bzw. Linien:

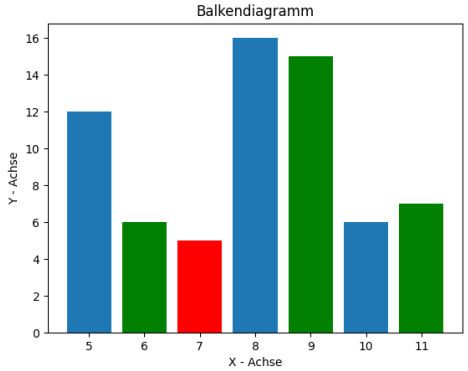
- `'-'` Durchgängige Linie
- `'--'` Strichlierte Linie
- `'-.'` Punt-Strich-Linie
- `':'` Punktierte Linie
- `'o'` Kreis (als Punkt)
- `'*'` Stern (als Punkt)
- `'+'` Plus-Zeichen (als Punkt)
- `'d'` bzw. `'D'` Kleiner bzw. großer Diamant (als Punkt)

Weitere Möglichkeiten unter
`https://www.tutorialspoint.com/numpy/numpy_matplotlib.htm`

Matplotlib - Arten von Plots

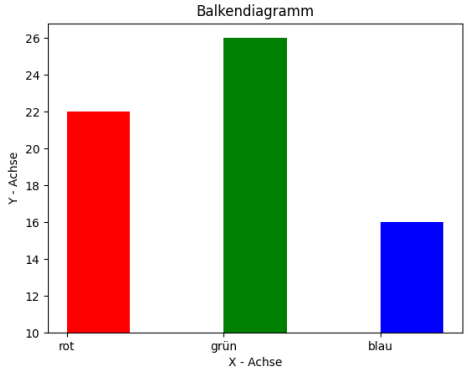
Das sind die wichtigsten zusätzlich angebotenen Plot-Varianten:
Balkendiagramm:

```
x = [5,8,10] # Position der Balken
y = [12,16,6] # Höhe der Balken

# Werden für die grünen Balken verwendet
x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center')
plt.bar(7, 5, color = 'r', align = 'center') # Geht auch ohne Array
plt.title('Balkendiagramm')
plt.xlabel('X - Achse')
plt.ylabel('Y - Achse')
plt.show()
```



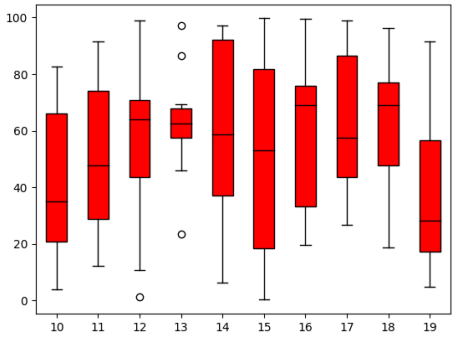
Weitere Parameter:

```
x = [1,2,3] # Position der Balken
y = [12,16,6] # Höhe der Balken

plt.bar(x,y, align = 'edge', width=0.4, bottom=10, color=['r','g','b'],
   tick_label=['rot','grün','blau'])
```



Matplotlib - Arten von Plots

Boxplot:




```
d = np.random.rand(100) * 100
d = d.reshape(10,10)

plt.boxplot(d, vert=True, # Horizontal oder Vertikal
        notch=False,  # Einbiegung beim Median
        patch_artist=True,  # Mit Farbe angefüllt
        labels=np.arange(10,20), # Beschriftung X-Achse
        boxprops=dict(facecolor='r', color='black'),  # Füllfarbe und Rahmenfarbe
        medianprops=dict(color='black')) # Farbe des Medianstrichs

plt.show()
```

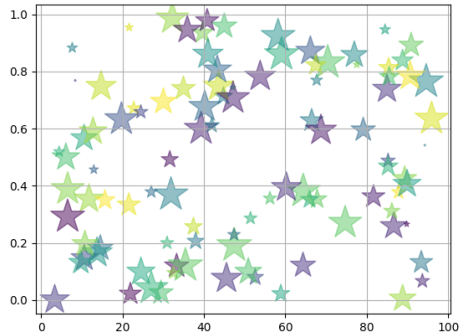
Erklärungen zu den Boxplots:

- Zur Darstellung von Verteilungen von Zahlen
- Mittlere Strich: Median
- Rote Box: Die mittleren 50%
- Die beiden Striche an den Enden: Bereich +- 2* Standardabweichung
- Punkte ausserhalb: Sogenannte Ausreisser

Matplotlib - Arten von Plots

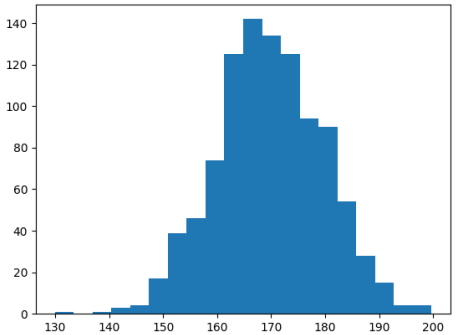
Scatterplot:

```
x = np.random.rand(100) * 100
y = np.random.rand(100)
# Diese Werte werden, wenn gewünscht, als Farben abgebildet
col = np.random.rand(100) * 100
size = np.random.rand(100) * 1000
plt.scatter(x,y,
c=col, # Farbwerte werden hier abgebildet
s=size, # Größe wird hier abgebildet
marker="*",
alpha=0.5) # Transparenz der Punkte
plt.grid(True)
plt.show()
```



Histogramm:

```
#Normalverteilung (Mittelwert, Standardabweichung, Anzahl)
x = np.random.normal(170, 10, 1000)
plt.hist(x, bins=20) #bins: Wie viele Balken
plt.savefig("out/m7.png")
plt.show()
```



Weitere Möglichkeiten unter: `https://matplotlib.org/stable/gallery/index.html`

Kleine Beispielauswertung - Wetterdaten London

- Das Beispiel befindet sich unter: `ex_02_matplotlib_sample_session.py`
- `NumPy` soll eigentlich nur eine Basisbibliothek sein und ist nicht direkt dafür gedacht, Datensätze auszuwerten
- Zu diesem Zweck wird später `Pandas` verwendet werden

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  d = np.genfromtxt('data/london_weather.csv', delimiter=",", skip_header=1 )
5
6  dt =  d[:,0] #Datum mit folgendem Aufbau: 19790103 (3.Jänner 1979)
7  # Aufteilen in Tag, Monat, Jahr
8  day = (dt % 100).astype('i')
9  month = (dt % 10000 / 100).astype('i')
10 year = (dt % 100000000 / 10000).astype('i')
11
12 # Check ob es funktioniert hat
13 print("Jahr:", np.unique(year, return_counts=True))
14 print("Monat", np.unique(month, return_counts=True))
15 print("Tag:", np.unique(day, return_counts=True))
16 print("Jahr MIN MAX" , np.min(year), np.max(year))
17
18 sun = d[:,2] # Sonnenstunden
19 print (sun)
20
21 #PLausibilitätscheck
22 print("Sun MIN MAX" , np.min(sun), np.max(sun))
23 plt.boxplot(sun)
24 plt.show()
25
26 sun1979 = sun[year == 1979] #Holen der Sonnenstunden im Jahr 1979
27 sun2020 = sun[year == 2020]
28 plt.boxplot([sun1979, sun2020]) #Gegenüberstellung der Sonnenstunden
29 plt.show()
```

Exkurs: Bildverarbeitung mit PIL

Bibliothek Pillow (PIL - Python Imaging Library):

- ist die am meisten verwendete Image Bibliothek in Python
- Ziel wird es sein, die Daten in ein numPy-Array einzulesen und zu verändern
- Die Bilder können dann mittels Matplotlib dargestellt werden
- Installation: pip install Pillow
- Einbinden mit import PIL

Laden und anzeigen eines Bildes ist sehr einfach:

```
1 from PIL import Image
2
3 img = Image.open('img/htl-logo.png')
4 print(img.format) # Speicherformat
5 print(img.size) # Dimensionen
6 print(img.mode) # Farbmodus
7 img.show() # Anzeigen des Bildes im default-Bildeditor
```

Laden des Bildes mit Matplotlib:

```
1 from matplotlib import image
2 from matplotlib import pyplot
3 image = image.imread('img/htl-logo.png')
4 print(image.dtype) # Werte als float32 gespeichert
5 print(image.shape) # (x,y, farbebenen) farb-ebenen = [r,g,b,opacity]
6 print(image[0][0]) # Pixel links oben
7 pyplot.imshow(image) # Bild auf die Zeichenfläche geben
8 pyplot.show()
```

Grundsätzliche Veränderungen von Bildern

Als Graustufenbild speichern:

```
#Weitere Modi: "1", 'L', 'P', 'RGB', 'RGBA', 'CMYK', 'YCbCr', 'HSV', "I", 'F'
im = np.array(Image.open('img/htl-logo.png').convert('L'))
#ist jetzt ein numpy-Array
print(im.shape) #(x,y)
print(im[0][0])
gr_im= Image.fromarray(im).save('img/htl-logo-gray.png')
```

Größe Ändern

```
img_small = np.array(Image.open('img/htl-logo.png').resize((200,200)))
Image.fromarray(img_small).save('img/htl-logo-small.png')
```

Trimming (Bildausschnitt)

```
im = np.array(Image.open('img/htl-logo.png'))
im_trim = im[150:300, 330:1350]
Image.fromarray(im_trim).save('img/htl-logo-trim.png')
```

PIL und NumPy - Grudlegendes

Das Einlesen der Bildinformationen mit NumPy ermöglicht eine leichte Manipulation des Bildes:

```
1  from PIL import Image
2  from numpy import asarray
3  image = Image.open('img/htl-logo.png')
4  # Übertragen des Bildes in ein numpy-Array
5  data = asarray(image)
6  print(data.shape) #Dimensionen des Bildes (x,y, farb-ebenen): (268, 1345, 4)
7  print(data[0][0]) #Pixel links oben: [255 255 255   0]
8  print(data[100,100,0]) # Rot-Wert des ersten Pixel
9  # Hier könnte die Manipulation stattfinden
10 # So kann man aus dem numpy-array wieder ein PIL-Bild machen
11 image2 = Image.fromarray(data)
```

Weitere Möglichkeiten der Abfrage:

```
1  # Alle Farbwerte eines Pixels
2  print(data[100,100,0:3]) # [r,g,b]
3  # Die rot-Werte aller Pixel
4  red = np.ravel(data[:,:,0 ]) #ravel: mache 1-dim array
5  # Die Positionen der grün-Werte, wo Bedingung erfüllt ist
6  lt_100 = np.where(data[:,:,1] < 100)
```

Zeile 6 Ergibt folgendes Ergebnis...

```
(array([ 82,  82,  82, ..., 267, 267, 267]), array([ 355,  356,  357, ..., 1319, 1320, 132
```

PIL und NumPy - Manipulation und erste Statistik

Gleich wie das Abfragen kann man den Colon (:) - Operator auch für die Zuweisung verwenden

```
data[:,:,0] = 100 # Alle rot-Werte auf 100 setzen
data[1:10,100:200,0] = 100 # Nur in einem bestimmten Bereich den Wert setzen
```

Zuweisen von Werten, wo eine bestimmte Bedingung erfüllt ist:

```
lt_100 = np.where(data[:,:,1] < 100) # Positionen, wo Bedingung erfüllt ist
data[lt_100] = 100 # Diese können als Index für die Manipulation verwendet werden
```

Mittelwert der rot-Werte

```
red = np.ravel(data[:,:,0 ]) #Alle rot-Werte
mean = np.mean(red)
```

Verwendung von `Matplotlib` in diesem Kontext:

```
from matplotlib import pyplot as plt
plt.boxplot(red)
plt.show()
```

Was ist Scipy?

Eine Bibliothek für wissenschaftliches Rechnen:

- Verwendet intern `NumPy`
- Kann numerische Integration
- `FFT` (schnelle Fourier Transformation)
  Zerlegung eines Audiosignals über die Zeit in die Frequenzanteile
- lineare Algebra
- Regression

Wir werden nur einfache, ausgewählte Funktionen von dieser Bibliothek verwenden.

Was ist Pandas?

- Eine Bibliothek für die Datenanalyse
- Baut auf `Numpy` und `Matplotlib` auf
- Hat die Datenstruktur `DataFrame`, die einer tabellarischen Ansicht entspricht
- Hält die Daten direkt im Hauptspeicher (deshalb sehr schnell)
- Kann mit vielen unterschiedlichen Datenformaten umgehen (`CSV`, `XLS`, `JSON`, ...)
- Unterstützung im Umgang mit Datensätzen: `Merging`, `Slicing`, `Subsetting`, `Grouping`
- Guter Umgang mit Zeitreihen
- `import pandas as pd`

Wichtige Datentypen:

- `Series`:
  - 1-dimensionales Array mit homogenen Daten
  - z.B. eine Sammlung aus Integer-Werten
  - Größe nicht veränderbar, aber die Inhalte
  - Ein Index kann vergeben werden, der den Zugriff stark beschleunigt
- `DataFrame`:
  - 2-dimensionaler Datensatz mit Zeilen und Spalten
  - Labels für die Spalten möglich
  - Die Größe ist veränderbar (`mutable`)
  - Spalten können unterschiedliche Typen beinhalten
  - Arithmetische Operationen auf Spalten möglich
  - Daten können sehr einfach aus unterschiedlichen Dateiformaten eingelesen werden
- `Panel`:
  - 3-dimensionaler Datensatz mit Zeilen und Spalten und Zeit
  - Wird z.B. für Befragungen angewandt die mehrmals stattfinden

Series (1/2)

- Am einfachsten wird eine Variable vom Typ `Series` direkt oder über `NumPy` erzeugt:

```
1 import pandas as pd
2 import numpy as np
3
4 s = pd.Series([1,2,3])
5 print (s)
6 data = np.array(['a', 'b', 'c'])
7 s = pd.Series(data)
8 print(s)
```

Ergibt...

```
0    1
1    2
2    3
dtype: int64
```

```
0    a
1    b
2    c
dtype: object
```

- Es kann auch ein eigener Index vergeben werden:

```
1 data = np.array(['a','b','c','d'])
2 index = [100,101,102,103]
3 s = pd.Series(data,index=index)
4 print (s)
```

Ergibt...

```
100    a
101    b
102    c
103    d
dtype: object
```

Series (2/2)

- Erzeugung aus einem `Dictionary`:

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
```

- Erzeugung aus einem skalaren Wert (5 Wird so oft gespeichert wie Index-Werte verfügbar sind):

```
s = pd.Series(5, index=[0, 1, 2, 3])
```

Zugriff auf die Werte so wie auf Arrays:

```
print(s[1])    #1.
print(s[1:3])    #1. 2.
```

Auch der Zugriff über die label-Indices möglich (also 'a', 'b', 'c')

```
print(s['c']) #2.
print(s[['a', 'c']]) # 0. 2.
```

Trotz der label-Indices (also 'a', 'b', 'c') ist ein Zugriff über die Position möglich:

```
print(s[0]) #2.
```

Bei Zugriff auf einen nicht vorhandenen Index kommt ein `KeyError`:

DataFrame (1/3)

Ein einfaches Anlegen eines `DataFrame` mit 2 Spalten:

```
1 data = [[1, 'a'], [2, 'b'] , [ 3, 'c']]
2 df = pd.DataFrame(data, columns = ['zahlen', 'buchstaben'])
3 print (df)
```

Ergibt...

```
   zahlen buchstaben
0       1          a
1       2          b
2       3          c
```

Es kann natürlich wieder ein Index vergeben werden:

```
1 df = pd.DataFrame(data, columns = ['zahlen', 'buchstaben'],
2         index = ['z1', 'z2', 'z3'])
```

Ergibt...

```
    zahlen buchstaben
z1       1          a
z2       2          b
z3       3          c
```

Holen der Zeilenindices:

```
1 df.index
```

DataFrame (2/3)

Erzeugen eines `DataFrame` aus einem Dictionary:

```
data = {'Name':['Hubert', 'Herbert', 'Franz'],
        'Age':[28,42, 12]}
df= pd.DataFrame(data)
```

Erzeugen aus `Series` von Daten (mit Verknüpfung über den Index):

```
d = {'spalte1' : pd.Series([1, 2, 3], index=['x', 'y', 'z']),
        'spalte2' : pd.Series([1, 2, 3, 4], index=['x', 'y', 'z', 'a'])}
df = pd.DataFrame(d)
```

Ergibt...

	spalte1	spalte2
a	NaN	4
x	1.0	1
y	2.0	2
z	3.0	3

Zugriff auf die Spalte über den Namen (und über Zeilenposition):

```
print (df['spalte1']) # Nan 1.0 2.0 3.0
print (df['spalte1'][1]) # 1.0
```

Hinzufügen einer Neuen Spalte geht auch über den Namen

```
df['spalte3'] = pd.Series([100,200,300],index=['a','x','y'])
```

Löschen einer Spalte mit `del`:

```
del df['spalte1']
```

DataFrame (3/3)

Auswahl der Zeilen über den Indexnamen mit `loc`:

```
print(df.loc['a'])
print(df.loc[['a','x']]) # Beide Zeilen werden ausgewählt
```

Auswahl der Zeilen über die Position (en) mit `iloc`:

```
print(df.iloc[0]) # Auswahl der ersten Zeile
print(df.iloc[0:2]) # Auswahl der Zeilen 1 und 2 (: Operator wie gewohnt verwenden)
```

Zeilen hinzufügen mit `append`:

```
df2 = pd.DataFrame([[70., 80.], [71., 81.]] , columns=['spalte2', 'spalte3'])
df = df.append(df2) # Die Spalten werden gematched
```

Löschen von Spalten über den Index mit `drop`:

```
df = df.drop(df.index[0]) # Zuerst den Zeilennamen an der Stelle 0 holen
df = df.drop(df.index[1:3]) # Auch hier kann der : - Operator angewandt werden
df = df = df.drop(columns='Gemnr') # Geht auch mit Spaltennamen
```

Anzahl aller Zellen:

```
print('size:', df.size)
```

Übertragung in ein normales Array:

```
print('values:', df.values)
```

Anzeigen der ersten und letzten Zeilen (optional mit Angabe der Anzahl):

```
print(df.head()) # Die ersten Zeilen
print(df.tail(4)) # Die letzten 4 Zeilen
```

Panel

Werden wir nicht verwenden, deshalb hier nur Grundlegendes
Ist ab Python 3.7 auch nicht mehr im Sprachumfang von `Pandas` zu finden, man verwendet `xarray`

Importieren und Exportieren von Daten

Es gibt hier für alle gängigen Datentypen Importfunktionen. hier die wichtigsten:

- `pd.read_table()`: Die allgemeinste Form für das Einlesen von Zeichengetrennten Datensätzen
- `pd.read_csv()`: Diese ist fast identlisch zur obigen Methode, ist also nur ein Wrapper, der `pd.read_table()` aufruft
- d = pd.read_excel('data/bev_meld.xlsx'): Excel Daten können auch direkt eingelesen werden. Per default wird das erste Worksheet eingelesen, möchte man ein anderes setzt man den Parameter `sheet_name=1` für das 2. Sheet.

Der Export ist ähnlich einfach:

- `df.to_csv('bar.txt', sep='')`
  Weitere Parameter unter:
  `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html`
- `df.to_excel('bar.xls', sheet_name='Sheet1')`.
  Weitere Parameter unter:
  `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_excel.html`
- `df.to_json`
- `df.to_html`
- `df.to_latex`
- `df.to_sql`: Schreibt in eine Datenbank unter Verwendung von `SQLAlchemy`

Sehr viele Formate für Import/Export werden unterstütutzt, hier ein Überblick:
`https://pandas.pydata.org/docs/reference/io.html`

Erstes Abfragen eines Dataframes

Grundsätzlicher Überblick über die Daten:

- `df.describe()`: Ein Grobüberblick über den Datensatz (inkl. einfacher Statistiken)
- `df.head(n)`: Zeige die ersten n-Zeilen (default: n=5)
- `df.tail(n)`: Zeige die letzten n-Zeilen.

Auswahl von Zeilen und Spalten:

- `print(df[3:]`: Auswahl von Zeilen wie gewohnt (hier: Zeile 3 bis zum Ende)
- `df.Spaltenname`: Zugriff Über den Spaltennamen möglich
- `df[['Spalte1', 'Spalte2']]`: Es können auch mehrere Spalten ausgewählt werden
- `df.iloc[:,3:]`: Über den Index kann man gleich mehrere Spalten auswählen (Alle Spalten ab dem Index 3)
- `df.iloc[:,[3,5,7]]`: Man kann auch eine Liste von Spalten angeben. Hier: 3, 5, 7
- `df.loc[:, df.columns != 'Bezirk']`: Alle Spalten ausser z.B. `Bezirk`.

Auswahl von Zeilen anhand von Kriterium:

- `df.loc[df.Bezirk == 'IL']`: Auswahl einzelner Zeilen (hier nur die wo Bezirk IL ist)
- `df.loc[df['Bezirk'].isin(['IL','IM'])]`: So kann man mehrere Bezirke auswählen

Zusammenfassen von Daten mit `groupby`

Die Methode groupby kann man als Vorbereitung für die Gruppierung verstehen. Mit dem erstellten können dann unterschiedliche Funktionen angewandt werden (wie z.B. `sum`):

```
1 bew_sum = df.groupby('Bezirk').sum() # es werden alle verbleibenenden
    Spalten in den Gruppen aufsummiert.
```

Möchte man nur bestimmte Spalten aufsummieren, muss man den Datensatz vorher entsprechend herrichten

Möglichkeiten der Datenvisualisierung

- **Matplotlib** (siehe Folien davor). Mit Pandas kann man aus dem Dataframe mit .values einfach ein ndarray erhalten, dann kann man Matplotlib wie gewohnt verwenden.

```
n = df.values[:, 3:]
plt.plot(n)
plt.show()
```

- **Pandas** hat stark vereinfachte plot-Funktionen über die von Matplotlib gelegt. Der Aufruf erfolgt über die DataFrame-Objekte, z.B.: das einfache Erstellen mehrerer Boxplots:

```
df.boxplot(column="spaltenname", by="Bezirk")
plt.show()
```

  weitere Infos: `https://pandas.pydata.org/docs/getting_started/intro_tutorials/04_plotting.html`
- **Seaborn**: Vereinfachte Befehle, ansprechende Grafiken, baut auch auf Matplotlib auf. Muss mit `pip` installiert werden. z.B.: das einfache Erstellen mehrerer Boxplots:

```
import seaborn as sns
sns.boxplot(x=w['Bezirk'], y=w['spaltenname'], data=w)
plt.show()
```

  weitere Infos: `https://seaborn.pydata.org/`
- **Plotly Express**: Haben wir voriges Jahr schon kurz kennengelernt, lässt sich auch mit Pandas DataFrames verwenden, z.B.:

```
import plotly.express as px
df = pd.DataFrame(dict(a=[1,3,2], b=[3,2,1]))
fig2 = px.bar(df)
fig2.show()
```

  weitere Infos: `https://plotly.com/python/pandas-backend/`

Skalenniveaus

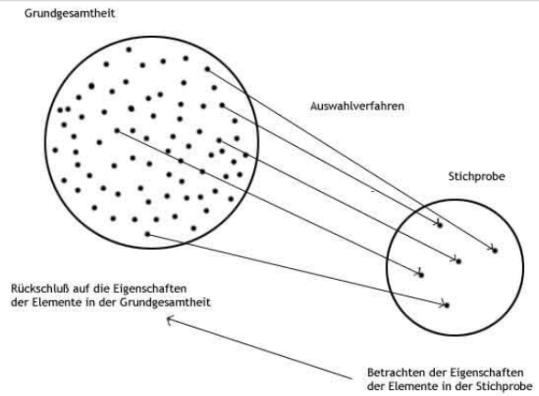
Das Skalenniveau oder Messniveau bestimmt, welche Eigenschaften die gemessenen Variablen haben. Dies bestimmt auch, welche statistische Verfahren auf sie angewandt werden dürfen.

- *Nominalskala*
  - Einfache Klasseneinteilung, wobei sich die einzelnen Antwortalternativen ausschliessen
  - Es besteht zwischen den einzelnen Klassen **keine Ordnung**
  - Beispiele: Religion, Wohnort, Familienstand
  - Statistisch sind hier nur **Häufigkeitszählungen** möglich
- *Ordinalskala*
  - Hier kann eine Rangordnung der einzelnen Alternativen erstellt werden
  - Beispiele: Einkommensgruppen, Altersgruppen, Schulnoten
  - Statistisch kann hier z.B. der **Median** berechnet werden (und alle statisischen Verfahren, die darauf aufbauen)
- *Intervallskala*
  - Ist bereits metrisches Skalenniveau
  - Aussagen wie: "Der Unterschied zwischen 2 gemessenen Werten ist doppelt so groß" ist trotzdem nicht zulässig
  - Beispiele: Grad Celsius (hier wird der Nullpunkt beliebig gewählt), Jahreszahlen mit Sterbejahr oder Geburtsjahr
  - Statistisch kann hier z.B. der **arithmetische Mittelwert** berechnet werden (und alle Verfahren, die darauf aufbauen). Sehr viele Verfahren werden dadurch schon anwendbar.
- *Ratioskala*
  - hat zusätzlich noch einen absoluten Nullpunkt
  - Beispiele: Länge, Gewicht, Alter, ...
  - Statistisch kann hier z.B. der **geometische Mittelwert** berechnet werden
  - Unterschied zu Intervallskala:
    - Alter (*ratio*): P1 10 Jahre, P2 20 Jahre, d.h. 20 / 10 = 2, d.h. P2 ist doppelt so alt wie P1
    - Alter über Geburtsjahr (*intervall*): P1 ist 2000 geboren, P2 ist 1990 geboren, aber 2000 / 1990 <> 2

Die Grundgesamtheit

- Dazu gehören alle Personen, über die man (als Gesamtheit) Aussagen machen möchte: `räumlich`, `zeitlich`, `sachlich` eingegrenzt.
- Diese soll genau festgelegt werden. Idealerweise (wenn auch selten) ist eine vollständige Liste der Personen vorhanden.
- Beispiele:
  - Alle für die nächste Nationalratswahlen Wahlberechtigten in Österreich
  - Alle Absolventen der HTL Anichstraße ab einem bestimmten Jahrgang
- Ist es möglich und sinnvoll alle Personen der Grundgesamtheit zu befragen, spricht man von einer **Vollerhebung**
  - Meist nur bei kleinen Populationen möglich (Ausnahme: Befragungen im Rahmen von Volkszählungen)
  - Vorteil: Daten können beliebig weit aufgesplittet werden und über alle Teile können Aussagen gemacht werden. Alle statistischen Aussagen, die darauf basieren, können ohne Schwankungsbreite interpretiert werden
  - Nachteil: Meist sehr teuer bzw. unmöglich
- Wenn es nicht möglich oder unwirtschaftlich ist, alle Personen zu befragen: Dann verwendet man eine **Stichprobe**.

Die Stichprobe



- Ist eine Auswahl von Personen der Grundgesamtheit
- die Auswahl muss **repräsentativ** sein:
  - Jede Person muss die gleiche Chance haben, in die Stichprobe mitaufgenommen zu werden
  - Keine Person darf mehrmals teilnehmen/darin vorkommen
- Eine hohe Anzahl an Teilnehmern allein garantiert nicht die Repräsentativität:
  - Beispiel: Eine Onlinebefragung, wo jeder mehrmals teilnehmen kann ist **nicht** repräsentativ und läßt somit keine statistischen Aussagen über die Grundgesamtheit zu, auch wenn 5000 Teilnahmen verzeichnet sind.
- Diese Auswahl kann getroffen werden durch:
  - **Zufallsauswahl**: z.B. durch Auflistung aller Personen der Grundgesamtheit und ziehen von Zufallszahlen zur Ermittlung der Personen. Man erhält daraus eine Zufallsstichprobe.
  - **Quota-Verfahren**: Auswahl entsprechend der Verteilung in der Gesamtbevölkerung (z.B. Alter, Geschlecht, Wohnbezirk)
- Jeder Stichprobe ist mit einem Fehler behaftet:
  - **Standardfehler** : Das Verwenden der Stichprobe allein führt schon zu fehlerhaften Schätzungen der wahren Werte
  - **Systematische** Fehler: Wenn die Stichprobe falsch gezogen wurde, wird das Ergebnis verzerrt. Bsp.: Schüler der 4. und 5. Klassen sind bei einer Schülerbefragung in der Stichprobe stark überrepräsentiert.

Stichprobenberechnung

Bei einer sehr großen Grundgesamtheit (größer 100.000) kann man die vereinfachte Formel verwenden:

$$n = \frac{(t^2 * p * q)}{e^2}$$

wobei....

- **n** = Stichprobengröße
- **t** = Konfidenzstufe (t = 1 = 68,3% Sicherheit, t = 2 = 95,5% Sicherheit und t = 3 = 99,7% Sicherheit bzw. t= 1.96 für 95%)
  Sie gibt an mit welcher Wahrscheinlichkeit die Ergebnisse der Stichprobe die wahren Werte der Grundgesamtheit widerspiegeln.
- **p** = Erwartete Variabilität: Sie bezieht sich auf die Wahrscheinlichkeit eines bestimmten Merkmals in der Grundgesamtheit. Bei maximaler Unsicherheit (z. B. ein Ja/Nein-Merkmal mit einer Wahrscheinlichkeit von 50 % für beide Antworten) ist die benötigte Stichprobengröße am größten
- **q** = 1 - p
- **e** = Gewünschte Fehlermarge (zB +/- 5 %)

Bei kleinerer Grundgesamtheit wird diese (**N**) in der Formel berücksichtigt:

$$n = \frac{(t^2 * p * q * N)}{(t^2 * p * q + e^2 * (N-1))}$$

Was ist deskriptive Statistik?

- Erste Beschreibungen und einen Überblick über Daten werden gegeben
- Maßzahlen wie Mittelwert, Median, Streuung, ... werden ermittelt
- Grafische Darstellung der Daten wie Histogramme, Boxplot oder Streudiagramme
- Wichtig sind Häufigkeitsverteilungen (`Kontingenztabellen`): Tabellen die zeigen wie oft bestimmte Werte z.B. in einer Spalte bzw. Kategorie vorkommen
- Auch interessant um Ausreißer zu entdecken
- So können erste Plausibilität-Checks (Überprüfung ob die Daten Sinn machen) durchgeführt werden und ev. Fehler in den Daten erkannt werden.
- Als Vorstufe für tiefer gehende statistische Verfahren (inferenzielle Statistik):
  - Erste Trends werden erkannt, die dann statistisch untermauert werden können. diese Trends werden als Hypothesen formuliert.
  - Bestimmt auch ob bestimmte statistische Verfahren zulässig sind.

Beispiele:

- Darstellung der Notenverteilung in einer Klasse mittels Histogramm und Berechnung des Klassendurchschnitts.
- Analyse der monatlichen Umsätze eines Unternehmens, um den Durchschnittsumsatz zu ermitteln und Schwankungen aufzuzeigen.

Was ist inferenzielle Statistik?

- Hier wird versucht Schlussfolgerungen auf die Grundgesamtheit auf Basis der Stichprobe zu machen. Es werden die oben formulierten Hypothesen statistisch untermauert (oder auch nicht bei gegenteiligen Ergebnis).
- Es wird also gezeigt ob ein Zusammenhang oder Unterschied in den Variablen oder Gruppen des Datensatzes statistisch **signifikant** ist.
- Es werden auch bestimmte Punkte oder Intervalle in der Grundgesamtheit geschätzt (s. Regression).

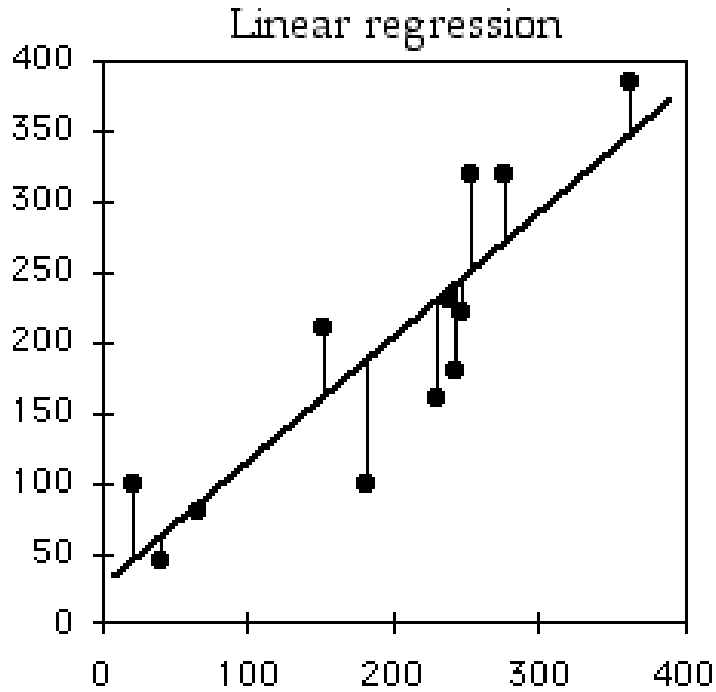
Beispiele:

- Ein neues Medikament wird an 500 Patienten getestet, um Rückschlüsse auf die gesamte Bevölkerung zu ziehen.
  Hypothese: "Das Medikament senkt den Blutdruck signifikant."
- Prüfung, ob Schülergruppen in einer neuen Unterrichtsmethode bessere Noten erzielen als mit der alten Methode

Auf den folgenden Folien werden einzelne inferenzstatistische Verfahren besprochen. Die Messniveaus bestimmen welche Verfahren überhaupt zulässig sind.

(lineare) Regression

- Hier wird versucht, eine abhängige Variable mit einer (oder mehreren) unabhängigen Variablen in Beziehung zu bringen
  - Beispiel: Unabhängige Variable: Jahreszahl. Abhängige Variable: Bevölkerung z.B. von Innsbruck.
  - Es wird nun versucht, die Bevölkerungszahl für bestimmte Jahre vorauszusagen
- Es wird hier versucht, eine gerade (lineare Funktion) durch die Punkte zu legen, so dass die quadrierten Abstände der Punkte zur geraden minimal wird.
- Mittels entwickeln einer linearen Funktion ist es dann möglich, Prognosen für die Zukunft zu erstellen (unter der Bedingung das sich die Entwicklung linear verhält (also z.B. daß das Bevölkerungswachstum linear ansteigt.))



(lineare) Regression in Python (1 /2)

- Es gibt mehrere Bibliotheken, die die lineare Regression unterstützen, `statsmodels` ist eigentlich der standard dazu (einfach mit `pip` installieren).
- Bauen es Modells:

```
1  import statsmodels.api as sm
2
3  # Als Eingang dient ein DataFrame. Falls dieser noch nicht in der für die
       Analyse notwendigen Form vorhanden ist, muss man einen erstellen
4  df_reg = pd.DataFrame({"years" : years, "mean_temp" : mean_temp})
5
6  # so wird das Modell gebaut. Links die abhängige, rechts die unabhängige
       Variable
7  model = sm.OLS.from_formula('mean_temp ~ years ', df_reg).fit()
8
9  a = model.params[1] # Die Werte für y = ax+b
10 b = model.params[0]
11
12 # R-quadrat (Wert) von 0-1: Zeigt wie gut die Gerade die Daten repräsentiert
13 rs = model.rsquared
14 fitted = model.fittedvalues # Die Werte auf der Geraden
15 # Abweichungen von Punkten auf der geraden zu den Datenpunkten
16 resid = model.resid
```

(lineare) Regression in Python (2 /2)

- Wie kann man die Vorhersagen machen?

```
# Auch hier bildet ein DataFrame die Basis
df_pred = pd.DataFrame({"years" : np.arange(2020,2040)})
# Wichtig ist dass die Spaltennamen der unabhängigen Variablen denen im Modell
    entsprechen
predictions = model.predict(df_pred)
# Die Predictions sind dann eine einfache Liste, die die Vorhersagen (hief für
    die einzelnen Jahre) beinhalten.
```

- Visualisierung der Ergebnisse.
  - Man kann mehrere Plots auf eine Zeichnung geben, indem man erst am Ende `plt.show()` angibt.
  - In diesem Fall am Besten die Werte und dann die Regressiongserade
  - Für den Blick in die Zukunft muss man den Zeichenbereich erweiteren mit z.B.: `plt.xlim([1980, 2040])`

```
plt.plot(val.years, predictions) # Die Vorhersage
plt.plot(df_reg.years, df_reg.mean_temp) # Die Werte selbst
plt.xlim([1980, 2040])
plt.show() # Erst jetzt show aufrufen, dann wird alles auf eine Grafik
    gezeichnet
```

Kontingenztabelle

- Entsteht durch die Verknüpung von 2 (oder mehr) Merkmalen in einer Tabelle
- Es werden die Häufigkeiten der kombinierten Merkmale angegeben

Beispiel: Kreuzung der Merkmale

- Haben sich während Ihrer Ausbildung Firmen mit Jobangeboten bei Ihnen gemeldet? (Eigentlich sehr unpräzise gefragt)
- Haben Sie studiert/studieren Sie?

	Jobangeboten gemeldet		
Studiert	Ja	Nein	Row Total
FH	12	16	28
	7.389	20.611	
UNI	2	22	24
	6.333	17.667	
Nicht	5	15	20
	5.278	14.722	
Column Total	19	53	72

wobei...

- ...in der ersten Zeile der Zelle der beobachtete Werte
- ...in der zweiten Zeile der Zelle der erwartete Wert steht: $\frac{RowTotal * ColumnTotal}{SumTotal}$, z.B. $\frac{28*19}{72} = 7.389$
- ...Pro Zeile und Spalte die Randsummen stehen
- ...Rechts unten und die Summe der Randsummen stehen (`Sum Total`)

X^2 - Unabhängigkeitstest

- Der Unabhängigkeitstest ist ein Signifikanztest auf Unabhängigkeit in der Kontingenztafel
- Man betrachtet zwei statistische Merkmale X und Y, die beliebig skaliert sein können. Man interessiert sich dafür, ob die Merkmale stochastisch unabhängig sind.
- Es wird die Nullhypothese H_0: Die Merkmale X und Y sind stochastisch unabhängig aufgestellt.
- Berechnung: $X^2 = \sum_{j=1}^{m} \sum_{k=1}^{r} \frac{(n_{jk} - n*_{jk})^2}{n*_{jk}}$ mit (m-1)(r-1) Freiheitsgraden (d.f.)
  wobei...
  - ...n_{jk} die beobachteten und
  - ...$n*_{jk}$ die erwarteten Werte sind.
- Der ermittelte X^2 - Wert wird in einer Tabelle nachgeschlagen, um zu sehen ob das Ergebnis signifikant ist (d.h. es besteht ein Zusammenhang zwischen den Merkmalen). Wenn $\mathbf{p < 0.05}$ dann signifikant, wenn $\mathbf{p < 0.01}$ dann hochsignifikant

Beispiel:

	Jobangeboten gemeldet		
Studiert	Ja	Nein	Row Total
FH	12	16	28
	7.389	20.611	
UNI	2	22	24
	6.333	17.667	
Nicht	5	15	20
	5.278	14.722	
Column Total	19	53	72

```
Statistics for All Table Factors
Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  7.956873     d.f. =  2     p =  0.01871487
```

X^2 -Umsetzung in Python/Pandas

```
# Wie bei den meisten Verfahren wird scipy verwendet
from scipy.stats import chi2_contingency

df = pd.read_csv('data/student-mat.csv', sep=";")

# als Basis für den chi^2 werden Kontingenztabellen verwendet:
ct_internet_higher = pd.crosstab(df['internet'], df['higher'])

chi, p, dof, expected = chi2_contingency(ct_internet_higher)
print ("Chi:",chi)
print ("p: %.5f" % p) # < 0.05 : Signifikanter Unterschied in den Gruppen
print ("dof" ,dof)
print ("expected",expected)


#Zum kopieren ist das Beispiel auch im github-Projekt:
# Es sollen die beobachteten als auch die erwarteten Werte (expected)
   dargestellt werden
sns.heatmap(ct_internet_higher, annot=False, cmap="YlGnBu")
sns.heatmap(ct_internet_higher, annot=ct_internet_higher,
   annot_kws={'va':'bottom'}, fmt="", cbar=False , cmap="YlGnBu")
sns.heatmap(ct_internet_higher, annot=expected, annot_kws={'va':'top'},
   fmt=".2f", cbar=False, cmap="YlGnBu")
plt.show()
```

Korrelation

- Eine Korrelation misst die Stärke einer statistischen Beziehung von zwei Variablen **A** und **B** zueinander.
- Der Korrelationskoeffizient bewegt sich zwischen
  - 1: Ja mehr **A**, desto mehr **B** und umgekehrt (direkter Zusammenhang)
  - -1 Ja mehr **A**, desto weniger **B** und umgekehrt (negativer Zusammenhang)
  - 0: Es besteht kein Zusammenhang zwischen den Variablen
- Wann welcher korrelationskoeffizient? (Parameter `method`):
  - `pearson`: Mindestvoraussetzung: Intervallskala
  - `spearman`: Mindestvoraussetzung: Ordinalskala.
    - Ist einfacher zu berechnen
  - `kendall`: Mindestvoraussetzung: Ordinalskala
    - Für kleinere Stichproben robuster
- Befehl dazu:

```
1 # corr ist sogar auf's Dataframe definiert
2 c = df_corr.corr(method ='spearman') #oder pearson
3 print(c) # beinhaltet nur den Korrelationskoeffizienten
```

- Möchte man auch die p-Werte haben muss man `scipy` verwenden:

```
1 from scipy.stats import spearmanr, pearsonr
2 # jetzt kommt auch der p-Wert und ein Korrelationskoeffizient allgemein
3 corr, p = spearmanr(df_corr['G3'], df_corr['Walc'])
4 print("corr: %.6f" % corr)
5 print("p-value: %.6f" % p)
```

- Visualisierung der Werte am Besten wieder mit einem Heatplot:

```
1 sns.heatmap(df_corr, annot=True, cmap="YlGnBu")
```

Mann-Whitney-U-Test

Für diesen Test hier (wie auch schon bei denen davor) gilt für den Wert **p**:

- `p < 0.05` ...signifikant
- `p < 0.01` ...hochsignifikant

Mann-Whitney-U-Test:

- Verwendung: Bei 2 unabhängigen Stichproben (z.B. 2 Gruppen aufgeteilt nach Geschlecht)
- Verfahren: Sind die `Mediane` zweier Stichproben signifikant unterschiedlich? Es werden nur die Ränge und nicht die tatsächlichen Werte bei der Berechnung verwendet.
- Messniveau: zumindest `ordinal` (die Werte müssen sortierbar sein)

```
1 from scipy.stats import mannwhitneyu
2 a = df['Walc']
3 a_t = a.loc[df['famsup'] == 'yes']
4 a_f = a.loc[df['famsup'] == 'no']
5
6 s, p = mannwhitneyu(a_m, a_f)
7
8 print("test statistics:", s)
9 print("p-value", p)
```

Übersicht über die Tests

So sollten die unterschiedlichen Tests ausgewählt werden:

- 2 Variablen gegenübergestellt mit Messniveau
  - Nominalskala: X^2
  - Ordinalskala: `Korrelation (Spearman)`
  - Intervall/Ratioskala: `Korrelation (Pearson)`
- 2 unabhängige Gruppen (z.B. Männlich/Weiblich) gegenübergestellt Variablen mit Messniveau
  - Ordinalskala: `Mann-Whitney-U-Test`
  - Intervall/Ratioskala: `t-Test` (haben wir nicht besprochen)

Was ist Künstliche Intelligenz (KI)

- zielt darauf ab, Maschinen zu entwickeln die intelligentes Verhalten simulieren und somit
- Tätigkeiten so wie der Mensch (oder sogar besser) ausführen können
- Klassische Aufgaben bzw. Problemstellungen:
  - Erkennung von Mustern: Handschriften, Fingerabdrücke
  - Verstehen von Sprache
  - Computer Vision (CV): Extraktion von Informationen aus Bildern und Videos
  - Entscheidungsunterstützung z.B. in Expertensystemen (Abbildung von Fachwissen auf deren Basis Entscheidungen getroffen werden). Folgendes ist dafür notwendig:
    - Wissensbasis: Spezialisiertes Wissen wird in Regeln und Heuristiken abgebildet.
    - Inferenzmaschine: Hier werden logische Schlussfolgerungen getroffen durch Anwendung der Regeln auf die konkreten Eingaben bzw. Fakten
- Klassische Techniken:
  - Regelbasierte Systeme
  - Genetische Algorithmen (Python-Bibliothek DEAP (Distributed Evolutionary Algorithms in Python):)
  - Machine Learning, z.B. Neuronale Netze

Was ist Machine Learning?

- Ein Teilbereich der KI (künstliche Intelligenz)
- Algorithmen und statistische Modelle, die Probleme lösen können ohne dass sie explizit dafür programmiert wurden
- Sondern es wird aus Erfahrung (Daten) gelernt, in dem z.B. Muster extrahiert werden
- Die Basis sind meistens sehr große Datenmengen, die verarbeitet werden müssen, deshalb hohe Rechenleistung notwendig

Was bedeutet Lernen in diesem Kontext?

Es gibt unterschiedliche Arten von Lernen (überlappen sich teilweise):

- **Supervised** Learning (Überwachtes Lernen):
  - Zu den Daten ist schon die Klassifikation bekannt (z.B. bei den MNIST-Zahlen-Images lt. Aufgabe)
  - Eingangsdaten werden `Features` bezeichnet, die Ausgaben / Klassifikation dazu die `Labels`.
  - Daraus werden Möglichkeiten erzeugt neue Daten zu klassifizieren.
  - Verfahren:
    - Lineare und logistische Regression
    - Entscheidungsbäume (decision trees) und Random Forests
    - Machine Learning
- **Unsupervised** Learning:
  - Es werden Muster in den Daten gesucht wobei nicht bekannt ist welchen Output bzw. Klassifikation (`Labels`) die Daten haben.
  - **Self-Supervised Learning**: Hier sind auch keine Labels vorhanden, nur wird hier versucht, fehlende Labels zu ersetzen, basierend auf bestimmte Lernaufgaben. Die Ergebnisse können dann eine Basis für supervised Learning sein.
- **Semi-Supervised Learning**: eine Zwischenstufe, bei der teilweise ge-labelte Daten zur Verfügung stehen
- **Reinforcement** Learning (bestärkendes Lernen):
  - Modelle (sogenannte Agenten) lernen durch Belohnungen für die Ausführung bestimmter Aktionen in einer Umgebung (kann z.B. über Sensoren wahrgenommen werden oder z.B. auch die Gegner/Hindernisse in einem Computerspiel)
  - Python-Package dazu: `Gymnasium` bzw. `Gym`: `https://gymnasium.farama.org/`

Bei komplexen Aufgaben werden mehrere dieser Strategien hintereinander angewandt

K-Means - Algorithmus

- Eine Form der Clusteranalyse, bei der eine (unüberwachte) Gruppierung der Daten anhand von bestimmten Features vorgenommen wird
- ES werden Datenpunkte in K Gruppen (Cluster) geteilt, indem die Distanz zwischen den Datenpunkten und den Clusterzentren minimiert wird.
- `K`, also die Anzahl der Gruppen werden am Anfang festgelegt

- Vorgangsweise (passiert in der `fit`-Methode):
  1. Zu Beginn werden die Clusterzentren (z.B. zufällig) festgelegt
  2. Zuordnung der Datenpunkte zu den nächstgelegenen Clustern (üblicherweise die euklidische Distanz)
  3. Aktualisierung der Clusterzentren: Nachdem alle Datenpunkte einem Cluster zugeordnet wurden, werden die Positionen der Clusterzentren neu berechnet
  4. Wiederholung: Die Schritte 2 und 3 werden iterativ wiederholt, bis ein Abbruchkriterium erfüllt ist. Typische Abbruchkriterien sind eine maximale Anzahl von Iterationen oder eine minimale Veränderung in den Positionen der Clusterzentren zwischen zwei aufeinanderfolgenden Iterationen
- Wenn dieses Prozess abgeschlossen ist kann darauf basierend eine Zuordnung für neue Punkte gemacht werden (`predict`)

- Ein Beispiel dazu in Python gibt es in den Machine-Learninf Beispielen unter: `ex_10_kmeans`

s

Was sind neuronale Netze?

Ein gutes Video dazu gibt es unter: https://www.youtube.com/watch?v=aircAruvnKk
Bestandteile:

- Neuron:
- Schichten (Layers);
- Zustandsüberführungsfunktion:

Ein neuronales Netz hat 3 Arten von Schichten:

- Eingabeschicht (Input Layer): Hier werden die Eingangsdaten eingespeist. Jedes Neuron entspricht hier einer Eingangsvariable (im Fall von MNIST ist es ein Pixel des Bildes)
- Versteckte Schichten (Hidden Layers):
  - Zwischen Ein- und Ausgabeschicht können ein oder mehrere weitere Schichten bestehen
  - In allen diesen Schichten werden Muster erkannt, anfangs sehr grob, je weiter die Schichten gehen desto komplexer werden die Muster
  - Durch diese Ebenen entsteht die Komplexität eines NN
- Ausgabeschicht (Output Layer):
  - Hier kommt dann das Ergebnis der Klassifikation heraus
  - In der Regel liegt bei jedem Ergebnis-Neuron in dieser Schicht eine Wahrscheinlichkeit an, wie sicher die Klassifikation in eine Kategorie fällt

Meist ist aber die Aufbereitung der Daten dass sie überhaupt sinnvoll in das NN einfliessen können die größte Herausforderung

KERAS

Beispiele unter: `https://github.com/albertgreinoecker/machine_learning_examples`

Was sind die wesentlichen Eigenschaften von KERAS:

- Open-Source-Softwarebibliothek
- KERAS ist eine sehr bekannte Bibliothek für Deep Learning.
- Es werden schon die bekanntesten Testdatensätze mitgeliefert
- Hohe Ebene der Abstraktion, somit kann man mit nur wenigen Zeilen Code komplexe neuronale Netzwerkarchitekturen erstellen.
- Unterstützt viele unterschiedliche Arten von Neuronalen Netzen, auch deren Kombination
- Lauffähig auf `CPU` und `GPU` (Graphics Processing Units)

Warum ist es besser, das Lernen auf der GPU laufen zu lassen:

- Eine GPU hat viele kleine Kerne, die für die **Parallelverarbeitung** geeignet sind
- Schnellere Matrix- und Vektoroperationen durch Tensor Cores (Tensoren sind kleine mathematische Recheneinheiten, Skalar, Vektor, Matrix, ...)

Die bekanntesten Alternativen:

- **TensorFlow**: ist eigentlich die Basisbibliothek, auf der Keras aufbaut. Von `Google` entwickelt.
- **PyTorch**: ist eine von `Facebook` entwickelte Open-Source-Bibliothek. Sie ist besonders bekannt für ihre Flexibilität und Benutzerfreundlichkeit bei der Entwicklung von Deep Learning Modellen.

KERAS - Erste Anwendung mit NMIST - Handschriften

Vollständiges Beispiel unter: `ex_02_first_learning.py`

Laden und Aufbereiten der Daten:

```
1  # Holen der Testdaten (x...Bilder, y...Labels)
2  (x_train, y_train), (x_test, y_test) = mnist.load_data()
3
4  # Standardisieren der Daten auf Werten zwichen 0 und 1
5  x_train = x_train.astype("float32") / 255
6  x_test = x_test.astype("float32") / 255
7
8  # Mache eine one-hot-Kodierung aus den Labels:
9  y_train = keras.utils.to_categorical(y_train, num_classes)
10 y_test = keras.utils.to_categorical(y_test, num_classes)
11
12 # Jedes Pixel soll ein Eingang in das NN werden, aber nicht als Matrix
13 x_train = x_train.reshape(60000, 784)
14 x_test = x_test.reshape(10000, 784)
```

- Normalerweise nimmt die Aufbereitung der Daten viel Zeit in Anspruch
- Wie man sieht hat hier KERAS schon viele Hilfsmethoden zur Verfügung
- Sonst kann hier Pandas und damit verbundene Packages nützlich sein

KERAS - Erste Anwendung mit NMIST - Handschriften

Festlegen der Struktur des Netzes (Model):

```
model = keras.Sequential(
[
        keras.Input(shape=(784,)),
        layers.Dense(16, activation="relu"),
        layers.Dense(num_classes, activation="softmax"),
]
)
```

1. `layer_dense`: Eine Ebene wo alle Knoten mit allen Vorgängerknoten verbunden sind
2. `units`: Wie viele Neuronen hat die Ebene im Neuronalen Netz
3. `activation`: Die Aktivierungsfunktion, also wie die summierten und gewichteten Eingangswerte an den Ausgang weitergegeben werden. folgende wichtige Funktionen gibt es:
   1. **relu**: `f(x) = max(0,x)`: <0 wird zu 0, >0 wird einfach weitergegeben
   2. **sigmoid**: "Quetscht" beliebige Werte in ein Intervall von 0-1
   3. **softmax**: Gibt Wahrscheinlichkeiten bei den entsprechenden Ausgängen aus
   4. Eine Liste weiterer Aktivierungsfunktionen gibt es unter: `https://keras.io/api/layers/activations/`

Einen Überblick über das erzeugte Model bekommt man mit `model.summary()`

KERAS - Erste Anwendung mit NMIST - Handschriften

Compilieren des Models: Hier werden Metriken über das Lernen festgelegt

```
model.compile(loss="categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"])
```

1. **loss**: Verlustfunktion, die die Unterschiede zwischen Vorhersagen und tatsächlichem Wert bestimmt und einen Verlustscore vergibt. Mögliche Werte:
   1. **categorical_crossentropy**: Bei Kategorien am Ausgang (wie in unserem Fall)
   2. **binary_crossentropy**: Bei 2 Kategorien
2. **optimizer**: Welcher Backpropagation-Algorithmus wird angewandt. Nimmt basierend auf dem Verlustscore eine Aktualisierung der Gewichte vor. Mögliche Einstellungen:
   1. **adam**: Stochastischer Gradientenabstieg. Sucht schrittweise das Minumum
   2. **rmsprop**: Bewegt sich schneller auf das Minimum zu mit der Gefahr es zu überschreiten
3. **metrics**: Der Wert, der die Klassifikations-Qualität wiedergibt: Am Besten `'accuracy'` verwenden, das ist die Korrektklassifikationsrate.

Trainieren des Models (Lernen):

```
history = model.fit(x_train, y_train, batch_size=batch_size,
    epochs=epochs, validation_split=0.1)
```

1. **history**: Beinhaltet die Ergebnisse des Lernens
2. **epochs**: Wie oft soll der komplette Datensatz durchlaufen werden zum Lernen
3. **batch_size**: Nach wie vielen Durchläufen sollen die Gewichte upgedated werden
4. **validation_split**: Welcher Anteil wird für die Validierung verwendet

Interpretation der Ergebnisse des Lernens

Während des Lernens werden 4 Parameter in einer Kurve angezeigt:

- **accuracy**: Prozentsatz wie viele samples richtig klassifiziert wurden: z.B. 1000 Versuche, 950 richtig klassifiziert ergibt 95 %.
- **loss**: Wir verwenden z.B. crossentropy (Wird für die Kategorisierung von mehr als 2 Gruppen verwendet). Diese Berechnet den Abstand der Prozentwerte der Klassifikation mit den wahren Werten, loss ist kein Prozentwert!
- **val accuracy**: Macht das gleiche aber mit den Ergebnissen eines vom Lernprozess getrennten Testdatensatz
- **val loss**: Macht das gleiche aber mit den Ergebnissen eines vom Lernprozess getrennten Testdatensatz

Natürlich soll das Modell anhand der Test- und nicht an Hand der Trainingsdaten evaluiert werden (man erhält wieder die beiden bekannten Metriken `loss` und `accuracy`):

```
1 score = model.evaluate(x_test, y_test, verbose=2)
2 print("Test loss:", score[0])
3 print("Test accuracy:", score[1])
```

Die gewünschten Vorhersagen bekommt man dann mit:

```
1 pred = model.predict(x_test)
2 print(pred[1]) # Wahrscheinlichkeiten für Image 2
3 print(y_labels[1]) # Das richtige Label dazu
4 pred_i = np.argmax(pred[i]) # Position des höchsten Wertes
```

Laden und Speichern

- Das Trainieren des Modells soll ja nicht jedes Mal wenn Prognosen zu machen sind erfolgen, sondern das Modell und die Gewichte werden gespeichert und wieder geladen
- So ist es möglich dass komplexe Netze auch auf schwächeren Rechnern für Prognosen verwendet werden können
- Das Laden und Speichern ist nicht auf Python beschränkt, sondern ist ein technologieübergreifendes Format für `Tensorflow` (also nicht einmal auf `KERAS` beschränkt)

So kann man Model und Gewichte speichern:

```
model.save('datei.mdl')
model.save_weights('datei.h5')
```

So können sie in Python wieder geladen werden

```
model = keras.models.load_model('datei.mdl')
model.load_weights('datei.h5')
```

Die einzelnen Gewichte pro Neuron kann man auch abfragen und in gewünschten Formaten speichern (und laden):

```
weights = model.get_weights()
weights[0] # Hole die Gewichte des ersten Layer
# Übertragung der Gewichte in JSON
j =json.dumps(pd.Series(weights).to_json(orient='values'), indent=3)
# Übertragung des Models in JSON
model_json = model.to_json()
```

KERAS - Inkludierte Testdaten

Diese Daten werden bei Bedarf einfach heruntergeladen und sind dann in sehr gut aufbereiteter Form verfügbar.
Imports: `from keras.datasets import mnist, fashion_mnist, imdb, ....`
Holen der Daten: `(x_train, y_train), (x_test, y_test) = imdb.load_data()`

Hier ein Auszug der verfügbaren Datensätze:

- `mnist`: Die uns bereits bekannten Handschriften
- `fashion_mnist`: Die uns bereits bekannten Modeartikel
- `CIFAR-10` und `CIFAR-100`: 60.000 32x32 Farbbildern, die in 10 bzw. 100 Klassen unterteilt sind. Die Klassen repräsentieren alltägliche Objekte wie Flugzeuge, Autos, Vögel, Katzen, Hirsche, Hunde, Frösche, Pferde, Schiffe und Lastwagen.
- `boston_housing`: Verschiedene Aspekte von Wohnhäusern in der Umgebung von Boston, z.B. Kriminalitätsrate, Anteil nicht gewerblicher Geschäftsflächen, Lehrer-Schüler-Verhältnis (Zahlenmässigen Verhältnis)
- `IMDB` (Internet Movie Database):Online-Datenbank, die Informationen über Filme, Fernsehserien, Schauspieler, Filmproduktionen, Filmbewertungen, Trivia und vieles mehr sammelt und zur Verfügung stellt

KERAS - CNN (Convolutional Neural Networks)

- Hier werden herkömmlichen Neuronalen Netzen noch Vorbereitungsebenen vorangestellt
- Convolutional Layer (`Conv2D`):
  - Hier werden bestimmte Filter (auch Kernels genannt) über die Datenstrukturen bewegt mit dem Ziel, einfache Muster wie Kanten oder Texturen zu erkennen
  - Es entstehen dann Werte wie gut die Werte im Pixelfenster dem Filter entsprechen
  - `layers.Conv2D(32, kernel_size=(3, 3), activation="relu")`
- Pooling-Operationen wie `MaxPooling2D`, `AveragePooling2D`:
  - Es werden aus einem bestimmten Fenster nur z.B. die Maximalwerte gespeichert
  - Dient der Dimensionsreduktion und hilft für die geringere Empfindlichkeit bezüglich der genauen Positionen
  - `layers.MaxPooling2D(pool_size=(2, 2))`
- Flatten:
  - Nachdem die Eingabedaten durch mehrere Faltungs- und Pooling-Schichten gegangen sind, liegen die Merkmale in einem hochdimensionalen Format vor.
  - Die Flatten-Operation wandelt diese mehrdimensionale Struktur der Merkmale in einen eindimensionalen Vektor um.
  - `layers.Flatten()`
- Dropout:
  - Hier werden zufällig Neuronen deaktiviert (meist 50%)
  - Es soll verhindert werden dass das NN zu genau lernt (Gefahr von `Overfitting`))
  - layers.Dropout(0.5)
- Nach der Kombination dieser Schritte (können auch mehrmals wiederholt werden) beginnt das uns bekannte NN zu arbeiten
- Verwendung Klassifikation von Bildern, Videos und Audiodaten

KERAS - CNN (Convolutional Neural Networks)

- Grundsätzlich ist der Aufbau sehr ähnlich zu den herkömmlichen Neuronalen Netzen
- Die CNN-Layer werden noch vorne angestellt, das Ergebnis dieser ist dann der Eingang in das herkömmliche NN (s. Zeile 10)
- Die Form der Ausgabe ist auch gleich (s. Zeile 11)

```
1  model = keras.Sequential(
2          [
3                  keras.Input(shape=input_shape),
4                  layers.Conv2D(32, kernel_size=(3, 3),
                      activation="relu"),
5                  layers.MaxPooling2D(pool_size=(2, 2)),
6                  layers.Conv2D(64, kernel_size=(3, 3),
                      activation="relu"),
7                  layers.MaxPooling2D(pool_size=(2, 2)),
8                  layers.Flatten(),
9                  layers.Dropout(0.5),
10                 layers.Dense(160, activation="relu"),
11                 layers.Dense(num_classes, activation="softmax"),
12         ]
13 )
```

Beziehen von Daten für Machine Learning

Hier eine Sammlung der wichtigsten Quellen für das Beziehen von Daten:

- `kaggle.com`
  - Hier sind die Daten oft schon in für ML geeigneter Form abgelegt
  - Alle möglichen echten Daten (Text, Bilder, Audio, ...)
- `https://www.tensorflow.org/datasets`
  - Datensätze die schon für Machine Learning aufbereitet
  - Es gibt ein Package für die direkte Integration in Python (KERAS):
    `pip install tensorflow tensorflow-datasets`
- `https://www.google.com/publicdata/directory`
- `https://datasetsearch.research.google.com`
  - Hier wird eine Metasuche über mehrere Datenquellen angeboten
- UCI Machine Learning Dataset: `https://archive.ics.uci.edu/datasets`
- Einfache Suche über github (dort ist nicht nur Quellcode sondern auch Daten)
- Österreich-spezifische Daten:`https://www.data.gv.at/suche/` (nur bedingt für ML geeignet)

Vorbereiten von Information (hier: Bilder)

- Natürlich möchte man nicht mit vorgefertigten Datensammlung arbeiten sondern mit eigenen Bildern
- Die Aufbereitung ist meist der aufwändigste Teil
- Deshalb bietet hier KERAS einige Hilfsmethoden an:

Lesen von Files aus einem Verzeichnis:

```
dataset =
   tf.keras.preprocessing.image_dataset_from_directory('/path/to/dataset',
        image_size=(224, 224),  # Resize images to this size
        batch_size=32,  # Number of images to load at each iteration
        label_mode='categorical'  # 'categorical', 'binary', 'int', or
           None
)
```

Der Aufbau muss so sein:

```
        /path/to/dataset/
                cats/
                        cat001.jpg
                        cat002.jpg
                ...
                dogs/
                        dog001.jpg
                        dog002.jpg
                ...
```

Ein komplettes Beispiel (auch mit predictions) befindet sich in `ex_08_prepare_images.py`

Alternative bzw. weiterführende Technologien zu CV (Computer Vision)

- **Teachablemachine**: `https://teachablemachine.withgoogle.com/train`
  - von Google
  - Hier kann man Modelle basierend auf Fotos und Audio. Man kann Klassifikationen machen und Gestures und Posen erkennen
  - Es ist kein Code notwendig
  - das trainierte Modell is exportierbar (als `TensorFlow`- bzw. `TensorFlow-Lite` (optimiert für embedded und mobile Geräte) Modell) und kann somit z.B. in KERAS importiert werden
- **MediaPipe**: `https://mediapipe-studio.webapps.google.com/home` bzw. `https://developers.google.com/mediapipe`
  - auch von Google...
  - beinhaltet fertige Modelle für: Face and Object Detection, Image Classification,
  - Beispiele dazu (in Verbindung mit der Kamera) in den Machine Learning Beispielen auf github (`ex_06_...`)
  - Wird oft in Verbindung mit `OpenCV` verwendet
- **OpenCV** (Open Source Computer Vision Library): Hat folgende Funktionalitäten:
  - Gute Unterstützung für einfache Bildmanipulation
  - Einfaches Ansprechen der Kamera
  - Konturerkennung und Formanalyse
  - Bewegungsverfolgung und Objekterkennung: Verfolgen von Bewegungen in Videos, Erkennen von Gesichtern, Autos
  - Maschinelles Learning und Deep Learning für fortgeschrittene Bildverarbeitungsaufgaben
- **Hugging Face**: `https://huggingface.co/`
  - Große Sammlung von vortrainierten Modellen und Datensätzen (von Community erstellt)
  - Enthält vor allem Sprachmodelle

LLM (Large Language Models)

- KIs, die speziell darauf trainiert sind, Texte zu verstehen, zu generieren und mit ihnen zu interagieren.
- Werden mit sehr großen Mengen an Daten (Texten) trainiert um viele Sprachen, Dialekte und Schreibstile abdecken zu können
- Texte haben die spezielle Eigenschaft dass sie sequentiell aufgebaut sind (d.h. das "davor" und "danach" ist relevant)
- Basis für die Verarbeitung der Daten bildet ein sogenannter **Transformer**, der in klar vordefinierten Stufen vorgeht (s. später)
- verwendet für NLP (Natural Language Processing)

Verwendungsgebiete von `LLM`:

- **Textgenerierung**: Erstellung von Artikeln, Geschichten, Gedichten und anderen Arten von schriftlichem Content.
- Konversation: Führung natürlicher Dialoge mit Benutzern in Chatbots oder virtuellen Assistenten.
- **Übersetzung**: Übersetzung von Texten zwischen verschiedenen Sprachen mit Berücksichtigung des Kontexts.
- **Textzusammenfassung**: Erstellung prägnanter Zusammenfassungen längerer Texte.
- **Frage-Antwort-Systeme**: Beantwortung spezifischer Fragen basierend auf einem gegebenen Text oder allgemeinem Wissen.
- **Sentiment-Analyse**: Einschätzung der Stimmung oder Meinung in Texten.

RNN (Rekurrente Neural Networks)

- Alle Netze die wir bis jetzt kennengelernt haben sind Feed-Forward-Netzwerke, die ihre Informationen nur an die Folge-Layer weitergeben
- Rekurrente NN geben Information auch an davorliegende Layer weiter
- Somit hat das RNN eine Art Gedächtnis
- Frühere Generationen von ChatGPT (`Generative Pretrained Transformer`) (`GPT-1` und `GPT-2`) basieren darauf
- Kann sowohl supervised als auch unsupervised sein:
  - **supervised**: Vor allem bei der Textklassifizierung
  - **unsupervised**: Bei allen generativen Ansätzen wie GPT (dazu später)
- Nachteile dieses Ansatzes:
  - Durch den sequentiellen Ansatz können diese sehr langsam sein
  - Vanishing Gradient Problem (Verschwindendes Gradientenproblem): Bei langen Sequenzen werden die Gewichte der ersten Schichten des Netzes wenig angepasst, so ist es schwierig komplexe Muster zu lernen weil eine Anfangsklassifikation nicht gut stattfindet
- Lösungsansätze dafür:
  - `Long Short-Term Memory` (LSTM):
    - 1997 vorgestellt von Sepp Hochreiter (JKU Linz!) und Jürgen Schmidhuber
    - Forget Gate (Vergessensgate): Entscheidung welche Informationen pro Zelle gelöscht werden sollen
  - `Gated Recurrent Units` (GRU):
    - Weiterentwickung von LSTM (aus dem Jahr 2014)
    - Vereinfachung

Transformer-Modell

- Wurde von `Hugging Face` entwickelt
- Der Transformer bildet die Basis für die Verarbeitung von sequentiellen Daten und gliedert sich in folgende Schritte:
  - `Tokenisierung`: Der Eingabetext wird in kleinere Einheiten (Tokens) zerlegt
  - `Einbettung`: Jeder Token wird in einen Vektor umgewandelt, der in einem hochdimensionalen Raum repräsentiert wird. Diese Vektoren enthalten Informationen über die Bedeutung der Wörter sowie ihre Beziehung zueinander.
  - GPT-3 und GPT-4 basieren darauf. Hier wird auch RNN durch Self-Attention ersetzt
- Kann sowohl supervised als auch unsupervised sein:
  - **supervised**: bei Textklassifizierung, Sentiment-Analyse und maschinelle Übersetzung
  - **unsupervised**: Bei allen generativen Ansätzen wie ChatGPT (dazu später)

Hugging Face

- "Das github" der Open Source KI-Modelle
- Hugging Face ist ein Unternehmen, das sich auf künstliche Intelligenz (KI) und natürliche Sprachverarbeitung (NLP) spezialisiert hat.
- Es ist vor allem bekannt für seine Arbeit an Transformer-Modellen wie BERT, GPT (Generative Pre-trained Transformer), und vielen anderen, die in einer Vielzahl von Sprachverarbeitungsaufgaben führend sind.
- bietet eine umfangreiche Sammlung von vorab trainierten Modellen und Werkzeugen für NLP und KI bietet, sodass Entwickler und Forscher leicht auf hochmoderne Sprachmodelle zugreifen und sie anpassen können.
- Läuft mit `PyTorch` im Hintergrund, funktioniert aber auch mit `Tensorflow`
- Modelle und Daten werden bei der Verwendung automatisch heruntergeladen und sind dann leicht verwendbar

Welche Kategorien von Modellen gibt es dort?
Wichtige Teile der Website:

- `Spaces`:
  - Hier werden maschinellen Lernmodelle, Anwendungen oder Demos in einer benutzerfreundlichen Umgebung in einer lauffähigen Version angeboten.
  - Perfekt zum Ausprobieren bevor man das Modell auf den Rechner holt
- `Models`:

Angebotene Kategorien:

- `Comic`:
- `Soundgenerierung`:
- Text to Image
- Feature Extraction

Bekannte Modelle bzw. Anwendungen:

- `EZ-Audio`:
  - Sehr natürlich klingende Audioaussprache mit Hilfe von NLP
- Voice Clone:
  - Lässt die Stimme wie eine bestimmte Person klingen
  - Benötigt eine kurze Trainingsphase mit der Zielstimme
- `llama2`:
- LayoutLM: Es werden aus gescannten Seiten Text, Layout, Bilder, Formulare, Tabellen, ... extraierht, also eine komplette Layoutanalyse durchgeführt

GPU vs. CPU

Die Verwendung einer GPU (`Graphics Processing Unit`) für künstliche Intelligenz (KI) ist oft effizienter als die Verwendung einer CPU (`Central Processing Unit`) aus mehreren Gründen:

- Optimiert für die Parallelverarbeitung
  - GPU besteht aus tausenden von Kernen, die Parallel angesprochen werden können
  - Bei der KI müssen viele Operationen parallel ausgeführt werden (z.B. Matrixmultiplikationen)
- Hohe Durchsatzrate für numerische Operationen
- KI-Frameworks sind für die Verwendung der GPU optimiert.
  Es gibt eine Schnittstelle genannt CUDA (Compute Unified Device Architecture), die es ermöglicht, die Rechenleistung von NVIDIA-GPUs für allgemeine, nicht-grafische Berechnungen über eine API (C, C++, Python) zu nutzen.

Rechtliche Bestimmungen bei KI - AI Act

Der AI Act (Künstliche Intelligenz Gesetz) ist ein Gesetzesvorschlag der Europäischen Kommission, der den Einsatz von künstlicher Intelligenz (KI) innerhalb der Europäischen Union regulieren soll.

- Ist seit 2. August 2024 in Kraft, es gibt allerdings Übergangsfristen
- Bestimmte Anwendungen von KI, die als Bedrohung für die Sicherheit, Grundrechte oder die Werte der EU angesehen werden, sind verboten (z.B. Social Scoring, Manipulative Systeme oder biometrische Überwachung im öffentlichen Raum)
- KI-Systeme, die in sicherheitskritischen oder grundrechtlich sensiblen Bereichen eingesetzt werden (z. B. im Gesundheitswesen, in der Strafverfolgung oder bei der Arbeitssuche), unterliegen strengen Anforderungen bezüglich Sicherheit und Vertrauenswürdigkeit.
- Benutzer müssen informiert werden, dass sie mit einer KI interagieren (z. B. bei Chatbots) - wenn moderates Riskio besteht - sonst keine Kennzeichnungspflicht. Sie haben das Recht auf Alternativen (z.B. in Callcentern oder bei virtuellen Assistenten)
- KI-Systeme, die in Bereichen mit hohem Risiko eingesetzt werden (wie kritische Infrastrukturen, Bildung, Beschäftigung oder Strafverfolgung), müssen folgende Anforderungen erfüllen:
  - Risikobewertungen und Tests vor der Markteinführung (Obliegt den Firmen selbst)
  - Dokumentationspflichten, die sicherstellen, dass der Betrieb der KI nachvollziehbar ist
  - Transparenzanforderungen, damit Benutzer das System im Bezug auf KI verstehen
  - Überwachungsmechanismen zur Sicherstellung der kontinuierlichen Einhaltung
- Die EU plant eine Aufsichtsbehörde für KI, die die Einhaltung der Vorschriften überwachen soll.
- Es soll schwere Strafen bei Missachtung geben (bis zu 6 % des weltweiten Jahresumsatzes)

Rechtliche Bestimmungen bei KI - Datenschutz

Folgende Punkte der Datenschutz-Grundverordnung (DSGVO) sind für KI relevant:

- Rechtsgrundlage für Datenverarbeitung: Personenbezogene Daten dürfen nur auf Basis einer klaren Rechtsgrundlage verarbeitet werden, z. B. Einwilligung oder vertragliche Notwendigkeit.
- Transparenz: Betroffene müssen darüber informiert werden, wenn ihre Daten von einem KI-System verarbeitet werden, und in verständlicher Weise darüber aufgeklärt werden, wie und warum dies geschieht.
- Recht auf Auskunft und Löschung: Betroffene haben das Recht, Informationen darüber zu erhalten, welche Daten von ihnen verarbeitet werden und können die Löschung dieser Daten verlangen.
- Automatisierte Entscheidungen: Menschen haben das Recht, nicht ausschließlich einer automatisierten Entscheidung unterworfen zu sein, die rechtliche oder ähnliche erhebliche Auswirkungen auf sie hat. Dies betrifft insbesondere KI-Systeme, die Entscheidungen über Kredite, Versicherungen oder Einstellungen treffen.

Ethik-Richtlinien der EU:

- Rechenschaftspflicht: KI-Entwickler und -Betreiber müssen für die Auswirkungen ihrer Systeme Verantwortung übernehmen.
- Datensouveränität: Nutzer sollten die Kontrolle darüber haben, wie ihre Daten gesammelt, gespeichert und verwendet werden.
- Nichtdiskriminierung: KI-Systeme sollten fair und frei von Vorurteilen sein, um keine Personengruppe zu diskriminieren.