

Häufig verwendete Bibliotheken

```
import matplotlib.pyplot as plt
import scipy.stats as st
import seaborn as sns
import pandas as pd
import numpy as np
```

Hilfe anzeigen

```
help(np.random)
Hilfetext eines Packages anzeigen
```

```
help(np.random.choice)
Hilfetext einer Funktion anzeigen
```

Numpy Basics

`np.e`: Konstante mit Wert von e
`np.pi`: Konstante mit Wert von π
`np.sqrt(2)`: Wurzel von Zahl berechnen
`np.square(2)`: Quadrat von Zahl berechnen
`np.abs(-45)`: Absolutwert einer Zahl berechnen
`np.round(2.35, x)`: Zahl auf x Nachkomastellen runden
`np.log(100)`: Natürlicher Logarithmus berechnen
`np.log10(100)`: Logarithmus mit Basis 10 berechnen
`np.prod(arr)`: Produkt der Zahlen von *arr*

Numpy Arrays

```
arr = np.array([2, 1, 4, 5, -8, 10])
Numpy-Array erzeugen
```

```
np.linspace(start=1, stop=2, num=4)
num Zahlen zwischen start und stop
In diesem Beispiel: [1.0, 1.333, 1.666, 2.0]
```

```
np.arange(start=1, stop=4, step=.6)
Zahlen von start bis stop mit inkrement step
In diesem Beispiel: [1.0, 1.6, 2.2, 2.8, 3.4]
```

```
new_arr = arr.reshape((n, m))
1-dimensionales Array in 2-dimensionales umwandeln
Neues Array hat  $n$  Zeilen mit je  $m$  Elementen
```

```
np.percentile(arr, q=[2.5, 97.5])
Werte der Quantile eines Datensatz anzeigen
In diesem Beispiel die 2.5%- und 97.5%-Quantile
```

```
np.sum(arr > x)
Werte grösser als  $x$  in einem Array zählen
```

```
arr = np.cumsum(arr)
Kumulative Summe des Array arr berechnen
```

```
np.nanmean(arr)
Mittelwert berechnen und NaN ignorieren
```

```
np.tile(arr, x)
Array  $x$ -Mal wiederholen und aneinander hängen
```

```
np.repeat(["M1", "M2", "M3"], [x1, x2, x3])
Jeder Werte des Array mit Index  $i$ ,  $x_i$ -mal wiederholen
```

```
np.corrcoef(arr_x, arr_y)
Korrelationsmatrix berechnen
```

Numpy Zufallszahlen

```
arr = np.random.choice(arr, size=1000)
Neues Array mit der Grösse size erstellen und zufällig
mit Werten aus dem übergebenen Array befüllen
```

```
arr = np.arange(1, 25)
np.random.choice(arr, 24, replace=False)
Zahlen zufällig sortieren
In diesem Beispiel die Zahlen 1 bis 24
```

```
np.random.seed(35)
Zufallszahlengenerator mit einem Wert initialisieren
```

```
np.random.seed()
Wert des Zufallszahlengenerator wieder löschen
```

```
np.random.normal(size=1000)
size Standard-Normalverteilte Zufallszahlen generieren
```

Pandas Series

```
series = pd.Series([79.98, 80.04, 80.02])
series = pd.Series(
    [1, 5, 9, 15, 20],
    index=["mo", "di", "mi", "do", "fr"]
)
```

`series.sum()`: Die Summe der Elemente von *series*
`series.prod()`: Das Produkt der Elemente von *series*
`series.mean()`: Der Durchschnitt der Elemente von *series*
`series.median()`: Der Median der Elemente von *series*
`series.var()`: Die Varianz der Elemente von *series*
`series.std()`: Standardabweichung von *series*
`series.count()`: Anzahl Elemente der *series*
`series.round(x)`: Werte auf x Nachkomastellen runden
`series.index`: Zeilenbeschrift der Elemente von *series*
`series.size`: Die Anzahl der Elemente von *series*
`series[1]`: Zugriff auf ein Elemente via Index
`series["mi"]`: Zugriff via Zeilenbeschrift

Quantile und Quartilsdifferenz

```
series.quantile(q=0.25, interpolation="midpoint")
Quantile (z.B. 25%, 75%, ...) von series berechnen
```

```
q75, q25 = series.quantile(q = [.75, .25],
    interpolation="midpoint")
iqr = q75 - q25
Quartilsdifferenz von series berechnen
```

Werte einlesen

```
np.loadtxt(r"./data.txt")
Daten für ein Array aus einem Textfile laden
```

```
frame = pd.read_csv(r"./data.csv",
    sep=",", index_col=0)
Werte für eine Frame aus einem CSV auslesen
```

```
pd.read_table(r"./gamma.txt",
    delim_whitespace=True)
leerzeichengetrennte Daten einlesen mit Pandas
```

Pandas DataFrame

```
frame = pd.DataFrame({
    "Luzern": ([1, 5, 9, 15, 20]),
    "Basel": ([3, 4, 12, 16, 18]),
    "Zuerich": ([8, 6, 10, 17, 23])
}, index=["jan", "feb", "mar", "apr", "mai"]
)
```

frame.columns: Spaltenname auslesen
frame.shape: Anzahl Zeilen und Spalten des Frames
frame.T: Zeilen und Spalten vertauschen
frame.describe(): Kennzahlen jeder Spalte anzeigen
frame.mean(axis=0): Durchschnitt pro Spalte berechnen
frame.mean(axis=1): Durchschnitt pro Zeile berechnen
frame.head(n): Erste n Zeilen des Frames anzeigen
frame.tail(n): Letzte n Zeilen des Frames anzeigen
frame.index: Zeilenbeschriftung des Frames anzeigen
frame.drop(x, 0): Zeile mit dem Index x löschen
frame.drop(x, 1): Spalte x löschen
frame.corr(): Korrelationsmatrix berechnen

Umgang mit DataFrame

```
copy = frame.copy()
```

Kopie eines DataFrame erstellen

```
frame.loc["mar": "mai", "Luzern"]
```

Auf einen Bereiche in einem DataFrame zugreifen

```
frame.loc[["mar", "mai"], ["Basel", "Zuerich"]]
```

Auf ausgewählt Elemente in einem DataFrame zugreifen

```
frame.sort_values(by='Luzern', ascending=False)
```

Daten im Frame nach einer Spalte sortieren

```
frame.nsmallest(n, 'Luzern')
```

```
frame.nlargest(n, 'Luzern')
```

Daten im Frame nach einer Spalte sortieren und dann n Zeilen mit grösstem oder kleinstem Werte zurückgeben

```
filtered = frame[frame['Luzern'] == 0]
```

Daten anhand des Wertes einer Spalte filtern

```
mean = frame.mean()['Luzern']
frame.loc[frame['Luzern'] < mean, 2:5]
```

Daten anhand des Wertes einer Spalte filtern und die Zeilen einschränken (in diesem Beispiel 2 bis 5)

DataFrame für die nächsten Befehle

```
df=DataFrame({
    "Behandlung": np.repeat(
        ["A", "B", "C", "D"],
        [4, 6, 6, 8]
    ),
    "Koagulationszeit":
        [62, 60, 63, 59, 63, 67, 71, ...]
})
```

```
df_group = df.groupby('Behandlung')
```

DataFrame nach Spaltenwerte gruppieren

```
df_group.get_group('A')
```

Auf gleiche Gruppe zugreifen

```
df_group.get_group('A')['Koagulationszeit']
```

Auf die Werte der Gruppe zugreifen

Matplotlib PyPlot

```
plt.title("..."): Titel des Plots festlegen
plt.xlabel("..."): X-Achsenbeschriftung festlegen
plt.ylabel("..."): Y-Achsenbeschriftung festlegen
plt.show(): Plot anzeigen
plt.tight_layout(): vermeiden, dass Beschriftungen in Diagrammen enden
```

```
plt.subplot(nrows=2, ncols=3, index=4)
plt.subplot(234)
```

Sub-Plot mit 2 Zeilen und 3 Spalten erstellen und den nächsten Plot an der Position 4 einfügen. Die Position wird von links nach rechts und dann von oben nach unten gezählt

Plots erstellen

```
series.hist(bins=[0, 1, 10, 11, 12])
```

Histogramm mit angegebenen Klassengrenzen plotten

```
series.plot(kind="hist", edgecolor="black")
```

Histogramm erstellen und Balken mit Farbe umrahmen

```
series.plot(kind="hist", density=True, ...)
```

Normiertes Histogramm erstellen, bzw. Fläche des Histogramms auf eins normieren

```
series.plot(kind="hist", bins=20, ...)
```

Anzahl Klassen des Histogramm manuell festlegen

```
series.plot(kind='hist', cumulative=True,
            histtype='step', density=True)
```

Empirische kumulative Verteilungsfunktion plotten

```
series.plot(kind='box', title='Methode A')
```

Boxplot von *series* erstellen und Titel des Plots festlegen

```
frame.boxplot("T", by="Time")
```

Daten gruppieren und durch einen Boxplot anzeigen

```
frame.plot(kind='scatter', x='wine', y='mor')
```

Streudiagramm mit zwei ausgewählten Achsen erstellen

Parameter x und y müssen auf Indizes des *frame* verweisen

```
b, a = np.polyfit(x_series, y_series, deg=1)
x = np.linspace(x_series.min(), x_series.max())
plt.plot(x, a + b * x, c='orange')
```

Plotten einer Regressionsgerade, bei welcher die Daten einem Polynom ersten Grades angeglichen wurden

```
st.probplot(arr, plot=plt)
```

QQ-Plot anhand einer Normalverteilung

```
t = np.linspace(start, stop, amount_of_numbers)
plt.plot(t, f(t), linewidth=1.0)
```

Kurve der Funktion $f(t)$ im Intervall $[start, stop]$ plotten

Mehrere DataFrame Plots in einem Matplotlib

Struktur der Daten für die nächsten Befehle

```
"high" "medium" "low"
0.71 2.2 2.25
1.66 2.93 3.93
2.01 3.08 5.08
...
```

```
plt.figure(1)
plt.subplot(1,3,1)
iron.low.plot(kind='box')
plt.subplot(1,3,2)
iron.medium.plot(kind='box', ax=plt.gca())
plt.subplot(1,3,3)
iron.high.plot(kind='box', ax=plt.gca())
plt.tight_layout()
plt.show()
```

Verteilungen

cdf: Kumulative Verteilungsfunktion $P(X \leq x)$
 ppf: Quantile der Verteilung α_q
 pdf: Dichte an der Stelle x
 pmf: Punktw'keit an der Stelle $P(X = x)$
 rvs: *size* Zufallszahlen generieren

Uniformverteilung

```
st.uniform.cdf(x=1, loc=4, scale=5)
 $P(X \leq 1)$  falls  $X \sim Unif(4, 9)$ 
scale ist nicht Endwert, sondern Länge des Intervall
```

```
st.uniform.pdf(x=1, loc=0, scale=7)
Dichte an der Stelle  $x = 1$  falls  $X \sim Unif(0, 7)$ 
```

Binom-, Exponential- und Poissonverteilung

```
st.binom.cdf(k=5100, n=10000, p=0.5)
 $P(X \leq 5100)$  falls  $X \sim Bin(10000, 0.5)$ 
```

```
st.binom.pmf(k=1000, n=1000, p=0.5)
Wert der Wahrscheinlichkeitsdichtefunktion an der Stelle  $k$ 
```

```
st.expon.cdf(x=4, loc=0, scale=1/3)
 $P(X \leq 4)$  falls  $X \sim e^3$ 
scale muss mit  $\frac{1}{\lambda}$  angegeben werden
```

```
st.expon.pdf(x=1, loc=0, scale=1/3)
Dichte an der Stelle  $x = 1$  falls  $X \sim e^3$ 
```

```
st.poisson.pmf(mu=1.5, k=2)
 $P(X = 2)$  falls  $X \sim Pois(1.5)$ 
```

Normalverteilung

```
st.norm.cdf(x=130, loc=100, scale=15)
 $P(X \leq 130)$  falls  $X \sim \mathcal{N}(100, 15^2)$ 
st.norm.ppf(q=0.05, loc=100, scale=15)
5% Quantile falls  $X \sim \mathcal{N}(100, 15^2)$ 
```

```
st.norm.cdf(x=1.5)
 $P(X \leq 1.5)$  falls  $X \sim \mathcal{N}(0, 1^2)$ 
```

T Verteilung

```
st.t.cdf(x=168, df=149, loc=164,
scale=10/np.sqrt(150))
 $P[\bar{X}_{150} \leq 168]$  falls  $u = 164$ ,  $\hat{\sigma} = 10$  und  $T \sim t_{149}$  (150 Messungen - 1 geschätzter Parameter)
```

```
st.t.ppf(0.05, df=v)
Quantile einer T-Verteilung mit  $v$  Freiheitsgrade
```

F Verteilung

```
st.f.cdf(x=F, dfn=DF_G, dfd=DF_E)
Kumulative Wahrscheinlichkeitsverteilung mit  $F = \frac{MS_G}{MS_E}$ ,  $DF_G = g - 1$ ,  $DF_E = n - g$  und  $n = m \cdot g$ 
```

```
st.f.ppf(q=0.95, dfn=DF_G, dfd=DF_E)
95% Perzentil mit  $DF_G = g - 1$ ,  $DF_E = n - g$  und  $n = m \cdot g$ 
```

Generierung zufallsverteilter Zahlen

```
st.norm.rvs(size=n)
n normalverteilte Zahlen
```

```
st.uniform.rvs(size=3, loc=0, scale=7)
uniform verteilte Zufallszahlen,  $X_i \sim Unif(0, 7)$ 
```

```
st.t.rvs(size=n, df=v)
generiert n t-verteilte Zahlen mit v Freiheitsgrade
```

```
st.chi2.rvs(size=n, df=v)
generiert n chi-verteilte Zahlen mit v Freiheitsgrade
```

Integral berechnen

```
from scipy.integrate import quad
f = lambda x: x * (15 - x/4)
ans, _ = quad(f, 0, 60)
In diesem Fall das Integral:  $\int_0^{60} x * (15 - \frac{x}{4})$ 
```

Gleichung lösen

```
from sympy.solvers import solve
from sympy import Symbol
x = Symbol('x')
solve(x**2/9000 * (15/2 - x/12) - 0.9, x)
Löst die Gleichung:  $\frac{x^2}{9000}(\frac{15}{2} - \frac{x}{12}) - 0.9 = 0$ 
```

Vertrauensintervall

```
st.t.interval(alpha=0.95, df=12, loc=80.02,
scale=0.024/np.sqrt(13))
95% Vertrauensintervall einer t-Verteilung
wenn  $n = 13$ ,  $\sigma = 0.024$  und  $u = 80.02$ 
```

```
st.norm.interval(alpha=0.99, loc=31,
scale=6/np.sqrt(10))
99% Vertrauensintervall falls  $X \sim \mathcal{N}(31, 6/\sqrt{10})$ 
```

Statistische Tests

```
st.binom_test(x=3, n=5, p=0.5,
             alternative='two-sided')
```

Vorzeichentest mit x Erfolge bei n Versuchen und einer Erfolgswahrscheinlichkeit von 50%

`alternative`: {'two-sided', 'greater', 'less'}, **optional**
Angabe der Alternativhypothese. Der Standardwert ist 'two-sided'.

Wilcoxon-Test

```
st.wilcoxon(x_arr, y_arr, zero_method="wilcox",
           correction=True)
```

Berechnet den Wilcoxon Vorzeichenrangtest der Differenz zwischen den beiden Serien `x_arr` und `y_arr`

```
st.wilcoxon(arr, zero_method="wilcox",
           correction=True)
```

Berechnet den Wilcoxon Vorzeichenrangtest, erwartet das `arr` schon die Differenz zweier Serien beinhaltet

`zero_method`: {"pratt", "wilcox", "zsplit"}, **optional**
"wilcox": Wilcoxon treatment: discards all zero-differences

```
st.wilcoxon(arr, zero_method="wilcox",
           correction=True).pvalue
```

Auf den p-Value des Wilcoxon-Tests zugreifen

t-Test

```
st.ttest_rel(series1, series2)
```

Statistischer Test für gepaarte Stichproben

```
st.ttest_ind(series1, series2, equal_var=False)
```

Statistischer Test für ungepaarte Stichproben

```
st.ttest_1samp(series, 1).pvalue
```

P-Wert eines T-Tests für eine Serie berechnen mit der Nullhypothese $\mu = 1$
pvalue ist der P-Wert des zweiseitigen Tests

Mann-Whitney U-Test

```
st.mannwhitneyu(series1, series2,
                alternative='two-sided')
```

Mann-Whitney U-Test (aka Wilcoxon Rank-sum Test)

Varianzanalyse

```
from statsmodels.graphics.factorplots
import interaction_plot
from statsmodels.stats.anova import anova_lm
from statsmodels.formula.api import ols
from patsy.contrasts import Sum
```

Benötigte Bibliotheksfunktionen für Varianzanalysen
Varianz-Analyse mit Stripcharts zwischen x und y

```
sns.boxplot(x="...", y="...", data=frame)
```

Varianz-Analyse mit Boxplots zwischen x und y

```
sns.distplot(Fstat, kde=False, norm_hist=True,
             hist_kws=dict(edgecolor="black", linewidth=2))
```

F-Statistik plotten

```
sns.boxplot(series.index.weekday, series)
sns.boxplot(series.index.month, series)
sns.boxplot(series.index.quarter, series)
sns.boxplot(series.index.year, series)
```

Boxplot für gruppierte Daten nach Wochentag, Monat, Quartal, Jahr, ... anzeigen
series muss ein DatetimeIndex haben

DataFrame für die nächsten Befehle

```
frame = pd.DataFrame({
    "Treatment": np.repeat(["Vak", "CO2"], 3),
    "steak_id": [7.66, 6.98, 7.80, 5.26, ...]
})
```

```
fit = ols("steak_id ~ Treatment", data=frame).fit()
fit.summary()
```

Gruppenmittelmodell berechnen zwischen der Id des

Steaks und der ausgeführten Behandlung

```
fit_pred = fit.get_prediction()
fit_pred.conf_int()
```

Vertrauensintervalle für Gruppenmittelwerte

```
anova_lm(fit)
```

Anova Tabelle berechnen

DataFrame für die nächsten Befehle

```
frame = pd.DataFrame({
    "Batch": np.tile(["1", "2", "3"], 4),
    "Methode": np.repeat(["8500", "9100"], 6),
    "Y": np.array([90.3, 89.2, 98.2, ...])
})
```

```
interaction_plot(x=frame["Batch"],
                trace=frame["Methode"], response=frame["Y"])
```

Interaktionsplot erstellen
- entlang der y-Achse die Zielgröße (response)
- entlang der x-Achse der durch x festgelegte Faktor
- für jede Stufe in trace wird dann eine Linie gezogen

```
formula = "Y ~ C(Methode, Sum) + C(Batch, Sum)"
fit = ols(formula, data=frame).fit()
```

Zweiweg-Varianzanalyse mit Blöcken zwischen den Datenspalten *Methode* und *Batch*

DataFrame für die nächsten Befehle

```
frame = pd.DataFrame({
    "Konz": np.repeat(["A", "B", "C", "D"], 6),
    "Temp": np.tile(np.repeat(["1", "2"], 3), 4),
    "Y": np.array([82, 46, 16, 20, 13, ...])
})
```

```
formula = "Y ~ C(Konz, Sum) * C(Temp, Sum)"
fit = ols(formula, data=frame).fit()
```

Faktorielle Experimente mit den zwei Faktoren *Konzentration* und *Temperatur*

Zeitreihen

```
from statsmodels.tsa.seasonal
import seasonal_decompose
```

Benötigte Bibliotheksfunktionen für Zeitreihen

DataFrame für die nächsten Befehle

| | TravelDate | Passengers |
|---|------------|------------|
| 0 | 1/1/1949 | 112 |
| 1 | 2/1/1949 | 118 |
| 2 | 3/1/1949 | 132 |
| 3 | 4/1/1949 | 129 |
| 4 | 5/1/1949 | 121 |
| . | | ... |

```
def boxcox(frame, lambd):
    return np.log(frame) if (lambd == 0)\
        else (frame**lambd - 1) / lambd
```

BoxCox-Funktion definieren

```
series = frame["Passengers"].shift(-5)
```

Zeitverschiebung (shifting) mit $k = -5$

```
col = frame["TravelDate"]
frame["TravelDate"] = pd.DatetimeIndex(col)
frame.set_index("TravelDate", inplace=True)
```

Datums-Index einer Zeitreihe setzen

```
frame["Passengers"].rolling(window=12).mean()
```

Bewegendes Mittel (moving average) berechnen
bei einer Fenstergrösse von 12

```
seasonal_decompose(frame["Passengers"],
    model="additive", freq=12).plot()
```

Zerlegen einer Zeitreihe in die verschiedenen Faktoren
bei einer Fenstergrösse von 12

```
seasonal_decompose(np.log(frame["Passengers"]),
    model="add").resid.plot()
```

Residuen Plot von logarithmierten Daten anzeigen

```
seasonal_decompose(frame["Passengers"],
    model="mul").plot()
```

Zerlegen einer Zeitreihe mit dem multiplikativen Modell

```
frame.resample("A").mean()
```

Zeitreihe so umformen, dass jede Zeile den
Jahresdurchschnitt eines Jahres enthält

Allgemein

```
from scipy.special import comb
comb(N=5, k=3, exact=True)
```

Binomialkoeffizient berechnen, in diesem Beispiel $\binom{5}{3}$

```
import warnings
warnings.filterwarnings('ignore')
```

Python-Warnungen ausblenden

```
import matplotlib
matplotlib.rcParams['figure.dpi'] = 150
```

Plot grösser machen (für High-DPI-Screens)

```
%matplotlib inline
```

Plots in Jupyter-Notebook direkt anzeigen