Einführung in die Statistik mit Python

RS-eco

Biodiversity & Global Change Lab
Technische Universität München

rs-eco@posteo.de

15. November 2019

Unstatistik des Monats

Klimapaket macht Flugtickets wesentlich teurer

Steuer um 76 Prozent rauf - GroKo macht Flugtickets noch mal teurer!

http://www.rwi-essen.de/unstatistik/96/

Klimapaket macht Flugtickets wesentlich teurer

- Erhöhung der Steuer auf Inlandsflüge von derzeit 7,50 auf 13,03 € (+76%), bei Auslandsflügen von 42,18 auf 59,43 € (+41%)
- Dabei fällt unter den Tisch, dass sich die absoluten Erhöhungen auf gerade einmal 5,53 € beziehungsweise 17,25 € belaufen – und sich am Ende in Form weitaus geringerer relativer Erhöhungen auf die Ticketpreise auswirken werden.
- Die billigsten Tickets für eine kurzfristigen Hin- und Rückflug von München nach Berlin kostet rund 135 €, für einen langfristig geplanten Flug rund 65 €. Diese Flüge würden sich um rund 4,1 % bis rund 8,5 % verteuern.
- Auch wenn die Berechnung der prozentualen Steuererhöhungen mathematisch nicht zu kritisieren ist, so ist es doch die implizite, dramatisierende Botschaft an den Leser, die aus einer Mücke einen Elefanten macht

http://www.rwi-essen.de/unstatistik/96/

Datenverarbeitung mit Python

Apply Funktion

```
# Import library
import seaborn as sns

# Read dataset
titanic = sns.load_dataset("titanic")

# Create a new function:
def num_missing(x): return sum(x.isnull())

# Applying per column (axis=0)
print(titanic.apply(num_missing, axis=0))

# Applying per row (axis=1)
print(titanic.apply(num_missing, axis=1).head())
```

Imputation von fehlenden Werten

```
# Import function to determine the mode
from scipy.stats import mode

mode(titanic['deck']) # This returns both mode and count.
# Mode can return multiple values with high frequency.

# Take the first mode value
mode(titanic['deck']).mode[0]

# Impute the values:
titanic['deck'].fillna(mode(titanic['deck']).mode[0],
                       inplace=True)

# Now check the missing values again to confirm:
print(titanic.apply(num_missing, axis=0))
```

Crosstab

```
# Import library
import pandas as pd

# Create crosstab
pd.crosstab(titanic["survived"],titanic["sex"], margins=True)

# Define Function for percentage conversion
def percConvert(ser): return ser/float(ser[-1])

# Apply function to crosstab output
pd.crosstab(titanic["survived"],titanic["sex"],
            margins=True).apply(percConvert, axis=1)
```

Zusammenführen von zwei DataFrames

```
# First we create some dummy data
band = pd.DataFrame({"name":["Mick","John","Paul"],
                     "band":["Stones","Beatles","Beatles"]})
print(band)

instruments = pd.DataFrame({"name":["John","Paul","Keith"],
                            "instrument":["guitar","bass",
                                          "guitar"]})
print(instruments)

# "Mutating" joins combine variables
pd.merge(band, instruments, on="name", how="inner")
pd.merge(band, instruments, how="left")
pd.merge(band, instruments, how="right")
pd.merge(band, instruments, how="outer")
```

Pivot-Tabelle mit Pandas erstellen

```
# Import library
import seaborn as sns

# Datensatz laden
titanic = sns.load_dataset("titanic")

# Pivot-Tabelle generieren
titanic.pivot_table(columns="class", index="sex",
                    values="survived", aggfunc="mean")

# Oder so
params = {"columns": "class", "index": "sex",
          "values": "survived", "aggfunc": "mean"}
titanic_pivot = titanic.pivot_table(**params)

print(titanic_pivot) # Show pivot-table
```

Unpivot - melt()

- melt() ist die Umkehrung von pivot()/pivot_table() und macht den DataFrame wieder länger

```
import pandas as pd

df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two', 'two'],
                   'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'baz': [1, 2, 3, 4, 5, 6]})
print(df)

df_pivot = df.pivot(index='foo', columns='bar', values='baz')
print(df_pivot)

# Show column names
df_pivot.columns.values

# Move row Index to column
df_pivot.reset_index(inplace=True)

# Unpivot DataFrame
df_pivot.melt(id_vars=['foo'], value_vars=['A', 'B', 'C'],
              var_name='bar', value_name='baz')
```

stack() und unstack()

- stack() macht den DataFrame länger
- unstack() ist die Umkehrung von stack() und macht den DataFrame wieder breiter

		c0		c1
		c00	c01	c10
r0	r00	(0,0)	(0,1)	(0.2)
	r01	(1,0)	(1,1)	(1,2)

d.stack()

			c0	c1
r0	r00	c00	(0,0)	NaN
		c01	(0,1)	NaN
		c10	NaN	(0,2)
	r01	c00	(1,0)	NaN
		c01	(1,1)	NaN
		c10	NaN	(1,2)

Stacked (i.e. taller)

d.unstack()

	c0				c1	
	c00		c01		c10	
	r00	r01	r00	r01	r00	r01
r0	(0,0)	(1,0)	(0,1)	(1,1)	(0,2)	(1,2)

Unstacked (i.e.broader)

Cheatsheets

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

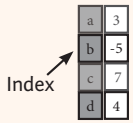
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns

Index	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-5
```
Get one element

```
>>> df[1:]
   Country     Capital  Population
1    India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat([0],[0])
'Belgium'
```
Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at([0], ['Country'])
'Belgium'
```
Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasília
Population  207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasília
```
Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```
Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
```
Series `s` where value is not >1 `s` where value is <-1 or >2

```
>>> df[df['Population']>1200000000]
```
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```
Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```
Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```
Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```
(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```
Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```
Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a     10.0
b      NaN
c      5.0
d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c    5.0
d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

https://assets.datacamp.com/blog_assets/PandasPythonForDataScience.pdf

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

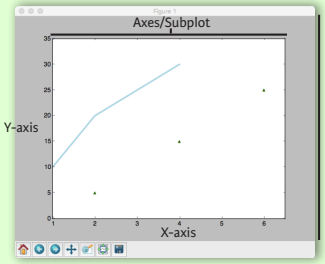
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                 Add padding to a plot
>>> ax.axis('equal')                        Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])   Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                     Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',         Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                   No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),          Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,        Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                      Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)    Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                    Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                         Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])          Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])     Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                 Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                 Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')               Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')     Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)            Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                   Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)           Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                             Plot a histogram
>>> ax3.boxplot(y)                          Make a box and whisker plot
>>> ax3.violinplot(z)                       Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                     Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                  Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)               Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                 Plot contours
>>> axes2[2].contourf(data1)                Plot filled contours
>>> axes2[2]= ax.clabel(CS)                 Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                               Clear an axis
>>> plt.clf()                               Clear the entire figure
>>> plt.close()                             Close a window
```

DataCamp
Learn Python for Data Science Interactively

https://assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))
>>> plt.title("title")                                 Step 4
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	Create a figure and one subplot

Seaborn styles

`>>> sns.set()`	(Re)set the seaborn default
`>>> sns.set_style("whitegrid")`	Set the matplotlib parameters
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	Set the matplotlib parameters
`>>> sns.axes_style("whitegrid")`	Return a dict of params or use with `with` to temporarily set the style

Context Functions

`>>> sns.set_context("talk")`	Set context to `"talk"`
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

`>>> sns.set_palette("husl",3)`	Define the color palette
`>>> sns.color_palette("husl")`	Use with `with` to temporarily set palette
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	Set your own color palette

3 Plotting With Seaborn

Axis Grids

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	Draw a categorical plot onto a Facetgrid
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	Plot data and regression model fits across a FacetGrid

`>>> h = sns.PairGrid(iris)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	
`>>> sns.pairplot(iris)`	Plot pairwise bivariate distributions
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	Grid for bivariate plot with marginal univariate plots
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	Plot bivariate distribution

Categorical Plots

Scatterplot

`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	Scatterplot with one categorical variable
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	Categorical scatterplot with non-overlapping points

Bar Chart

`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	Show count of observations

Point Plot

`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	Show point estimates and confidence intervals as rectangular bars

Boxplot

`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	Boxplot
`>>> sns.boxplot(data=iris,orient="h")`	Boxplot with wide-form data

Violinplot

`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	Violin plot

Regression Plots

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	Plot data and a linear regression model fit

Distribution Plots

`>>> plot = sns.distplot(data.y, kde=False, color="b")`	Plot univariate distribution

Matrix Plots

`>>> sns.heatmap(uniform_data,vmin=0,vmax=1)`	Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

`>>> g.despine(left=True)`	Remove left spine
`>>> g.set_ylabels("Survived")`	Set the labels of the y-axis
`>>> g.set_xticklabels(rotation=45)`	Set the tick labels for x
`>>> g.set_axis_labels("Survived", "Sex")`	Set the axis labels
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	Set the limit and ticks of the x-and y-axis

Plot

`>>> plt.title("A Title")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear an entire figure
`>>> plt.close()`	Close a window

DataCamp
Learn Python for Data Science Interactively

https://assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.pdf

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score — Estimator score method

```
>>> knn.score(X_test, y_test)
```

Metric scoring functions

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Classification Report — Precision, recall, f1-score and support

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

https://assets.datacamp.com/blog_assets/Scikit_Learn_Cheat_Sheet_Python.pdf

Nützliche Ressourcen

Nützliche Ressourcen

- Statistik allgemein
  - http://www.statsoft.com/Textbook/Elementary-Statistics-Concepts
  - http://www.reiter1.com/Glossar/Glossar.htm
- Python allgemein
  - T.R. Padmanabhan - Programming with Python (Springer Verlag)
  - Heiko Kalista - Python3 (Hanser Verlag)
- Statistik mit Python
  - http://gael-varoquaux.info/stats_in_python_tutorial/
  - Thomas Haslwanter - An Introduction to Statistics with Python (Springer Verlag)
  - Markus Feiks - Empirische Sozialforschung mit Python (Springer Verlag)
  - Scipy Lecture Notes (http://scipy-lectures.org/)

Nachtrag

Namensräume verbinden - join() & path.join()

```
list1 = ['1','2','3','4']; s = "-"

# joins elements of list1 by '-' and stores in string s
s.join(list1)

# Create filename from basename and format
base_filename='my_figure'
format = 'pdf'
filename = ".".join([base_filename, format])

# Specify full path of file directory, please change accordingly
dir_name='/home/matt/Documents/Github/pyStats/'

# Create filename with full path
"".join([dir_name, filename])

# Alternative approach using the os package
import os
os.path.join(dir_name, base_filename + "." + format)
```

Speichern von bestimmten Abbildungen

```
import matplotlib.pyplot as plt

f1 = plt.figure()
plt.plot(range(10), range(10), "o")
plt.show()

# Import library and dataset
iris = sns.load_dataset('iris')

# Hist only
f2 = plt.figure()
sns.distplot(a=iris["sepal_length"], hist=True, kde=False, rug=False )
plt.show()

# Save with combined dir_name and filename
file = "".join([dir_name, filename])
file # Make sure the path is correct, before saving!!!

f1.savefig(file, bbox_inches='tight')

# Save to working directory
f2.savefig("hist_sepal_length.pdf", bbox_inches='tight')
```

Übung

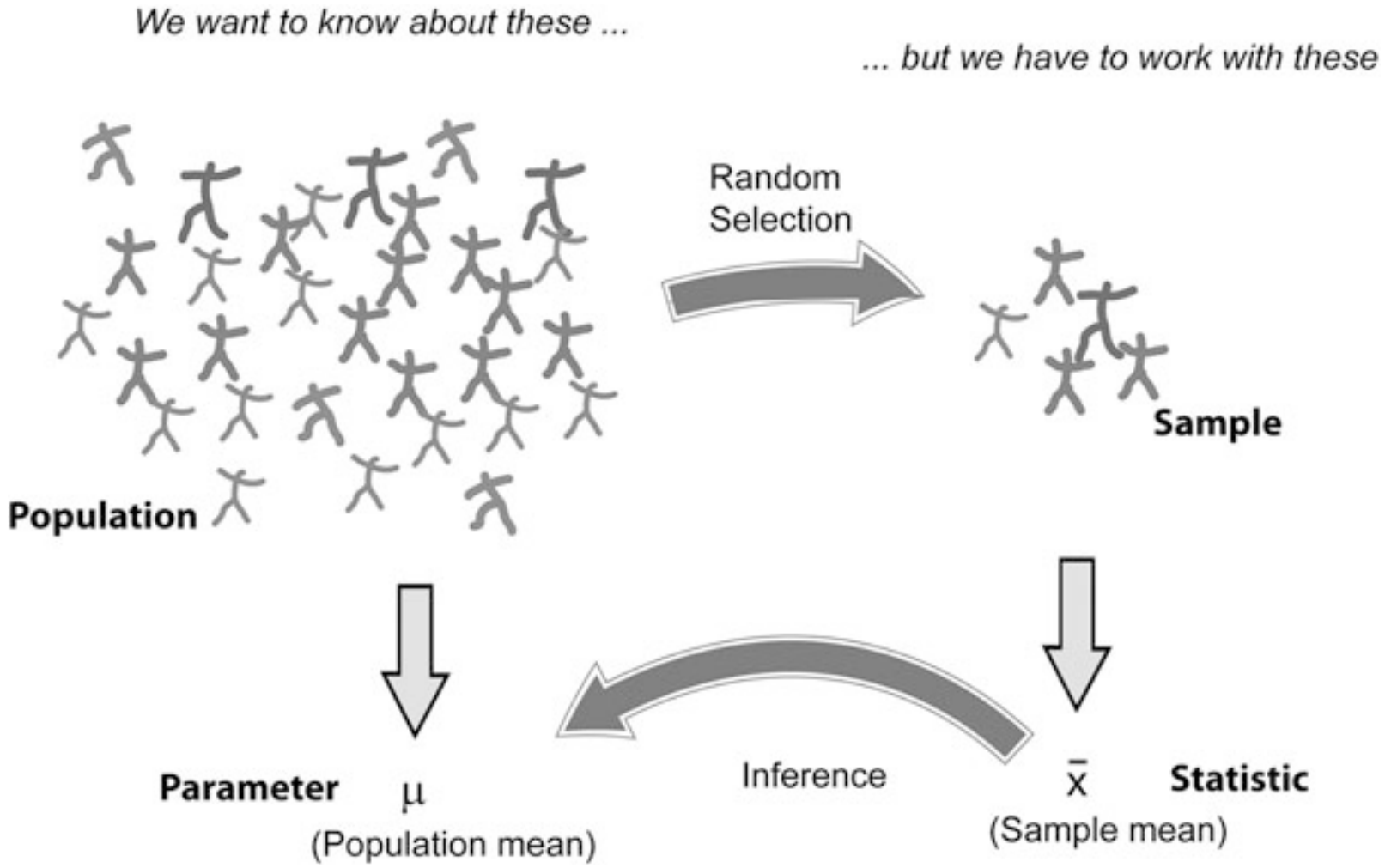
Übung

- Ihr findet sämtliche Kursunterlagen unter: http://github.com/RS-eco/pyStats
- Öffnet das Jupyter Notebook für Tag 3 und:
  - Ladet den Datensatz **schoko.csv** in Python
  - Verschafft euch einen Überblick über den Datensatz
  - Findet sämtliche Bio-Schokolade mit einem Preis nicht teurer als 1,6 €
  - Sortiert den Datensatz nach Bio und Preis
  - Erstellt eine Pivot-Tabelle mit Marke, Kategorie und Durchschnitts-Preis (pivot_table())
  - Erstellt eine Heatmap von der Pivot-Tabelle (sns.heatmap())
  - Erstellt eine Grafik (sns.catplot()) mit Bio, Preis & Kategorie
  - Und nun nochmal mit sns.violinplot()
  - Erstellt nun noch einen Scatterplot (sns.scatterplot()) mit Kakaogehalt, Preis, Kategorie & Anzahl der Inhaltsstoffe

Verteilungs- & Dichtefunktionen

Wahrscheinlichkeitsverteilungen

- Die mathematischen Werkzeuge zur Beschreibung der Verteilung numerischer Daten in Populationen und Stichproben.

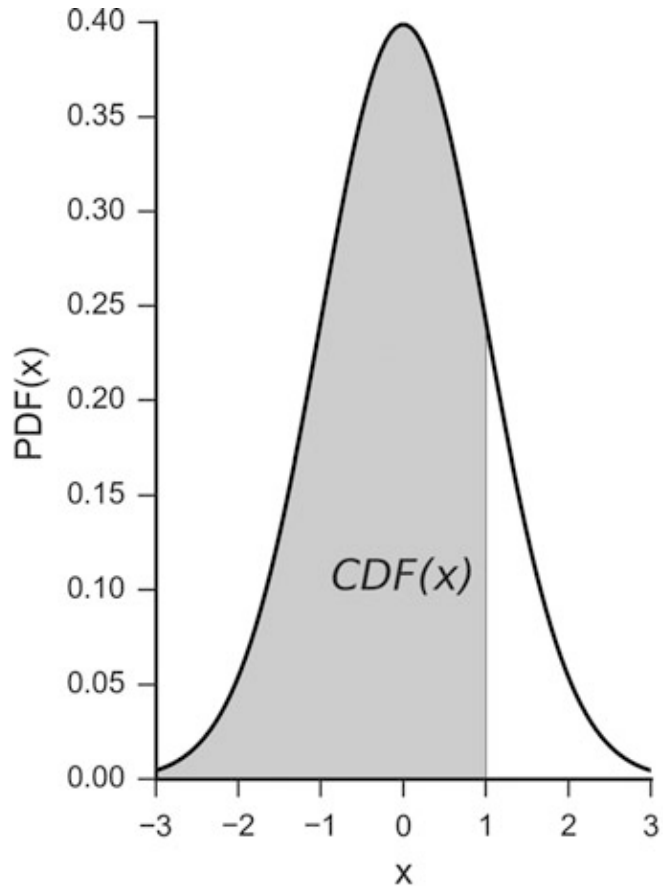


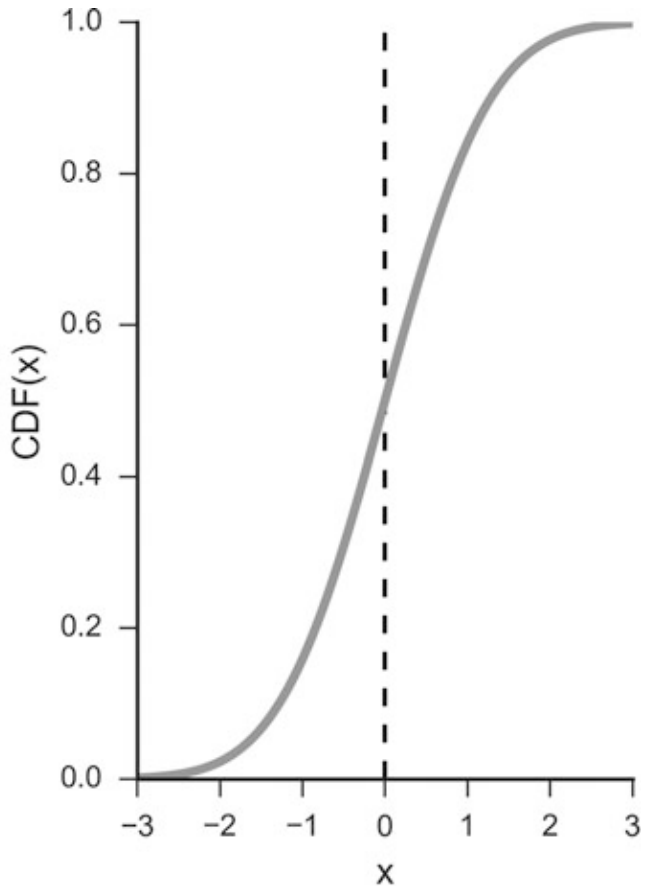
Wahrscheinlichkeitsverteilungen

- Die mathematischen Werkzeuge zur Beschreibung der Verteilung numerischer Daten in Populationen und Stichproben.
- **Diskrete Verteilungen**
  - Für eine gegebene diskrete Verteilung wird der Satz aller Wahrscheinlichkeiten als Wahrscheinlichkeitsmassenfunktion (Probability Mass Function = PMF) dieser Verteilung bezeichnet.
- **Kontinuierliche Verteilungen**
  - Für eine gegebene kontinuierliche Verteilung ist die Kurve, die die Wahrscheinlichkeit für jeden Wert beschreibt, d.h. die Wahrscheinlichkeitsverteilung, eine kontinuierliche Funktion, die Wahrscheinlichkeitsdichtefunktion (Probability Density Function = PDF).

Verteilungs- & Dichtefunktion

- **Verteilungsfunktion** = kumulierte relative Häufigkeit von Ereignissen, S-förmig
- **Dichtefunktion** = Ableitung der Verteilungsfunktion, glockenförmig

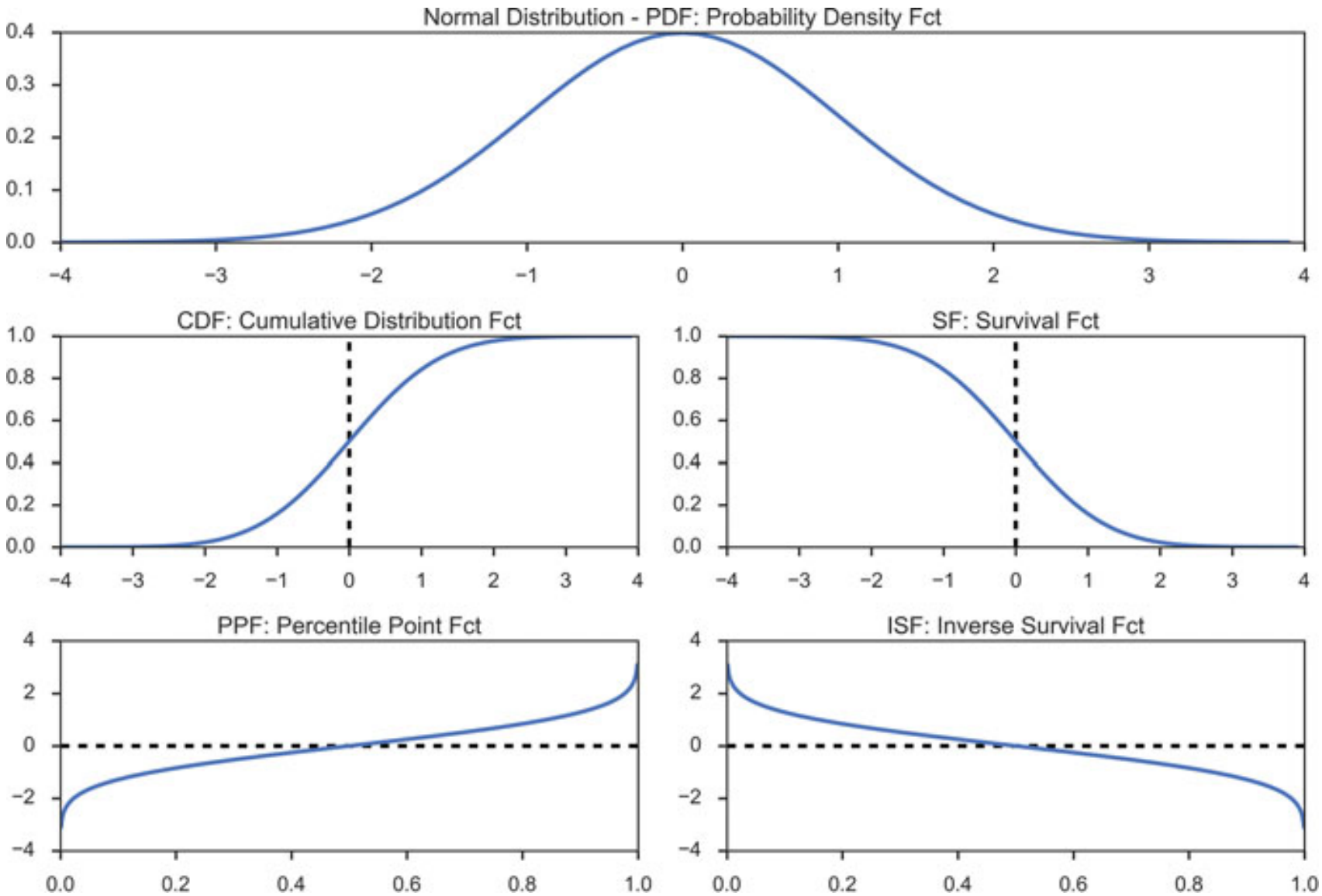




Verteilungs- & Dichtefunktion

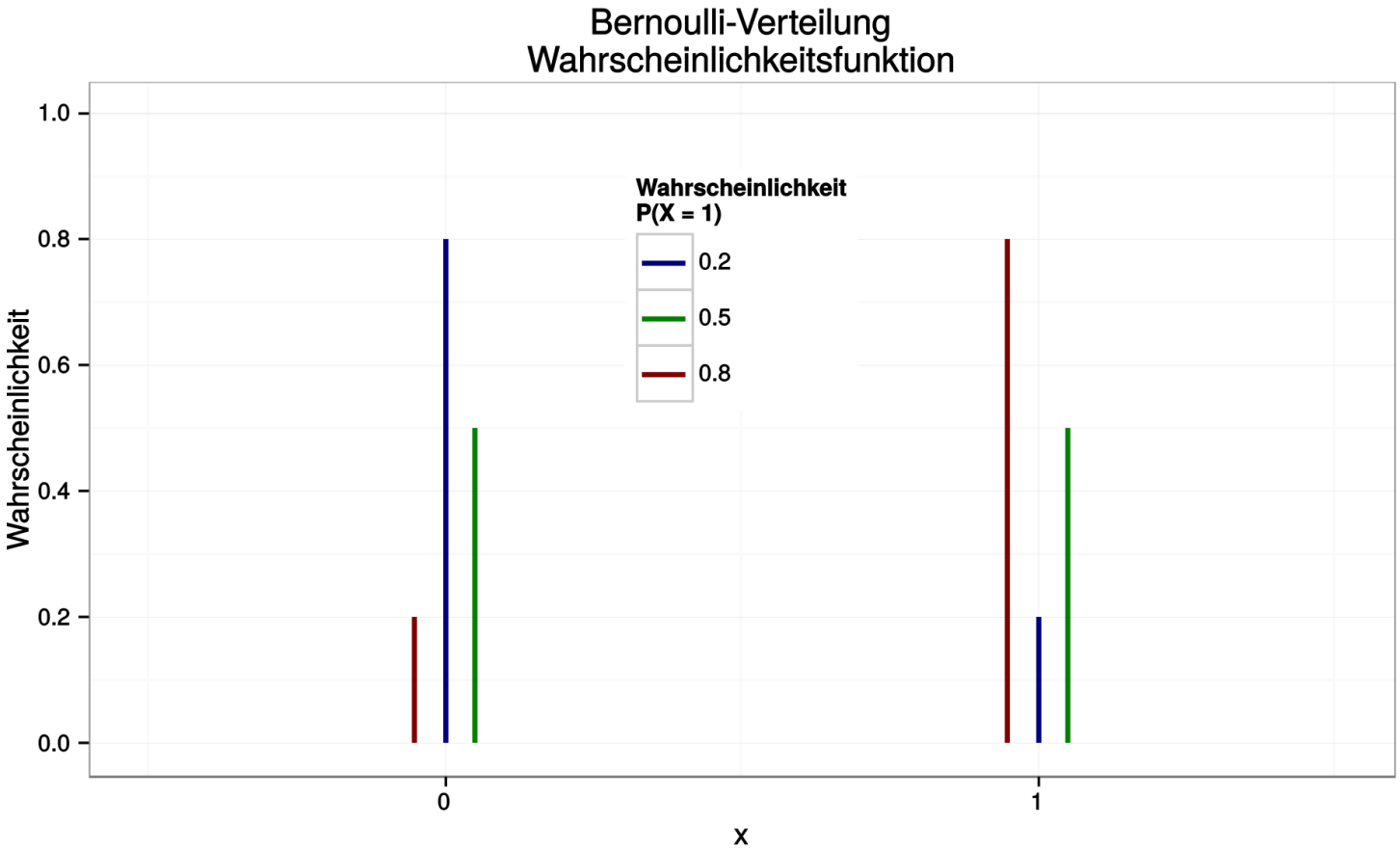
- **Verteilungsfunktion** = kumulierte relative Häufigkeit von Ereignissen, S-förmig
- **Dichtefunktion** = Ableitung der Verteilungsfunktion, glockenförmig
  - Dichtefunktion wird oft fälschlicherweise "Verteilungsfunktion" genannt
  - Dichtefunktionen sind etwas Abstraktes, aus dem sich nichts ohne weiteres ablesen lässt.
- Aus den S-förmigen Verteilungsfunktionen dagegen kann man Häufigkeiten direkt ablesen, oder wenigstens durch einfache Differenzbildung berechnen.

Darstellungen von Wahrscheinlichkeitsdichten



Diskrete Verteilungen für endliche Mengen

- **Bernoulli-Verteilung** = Beschreibung von zufälligen Ereignissen, bei denen es nur zwei mögliche Versuchsausgänge und eine vorgegebene Obergrenze gibt



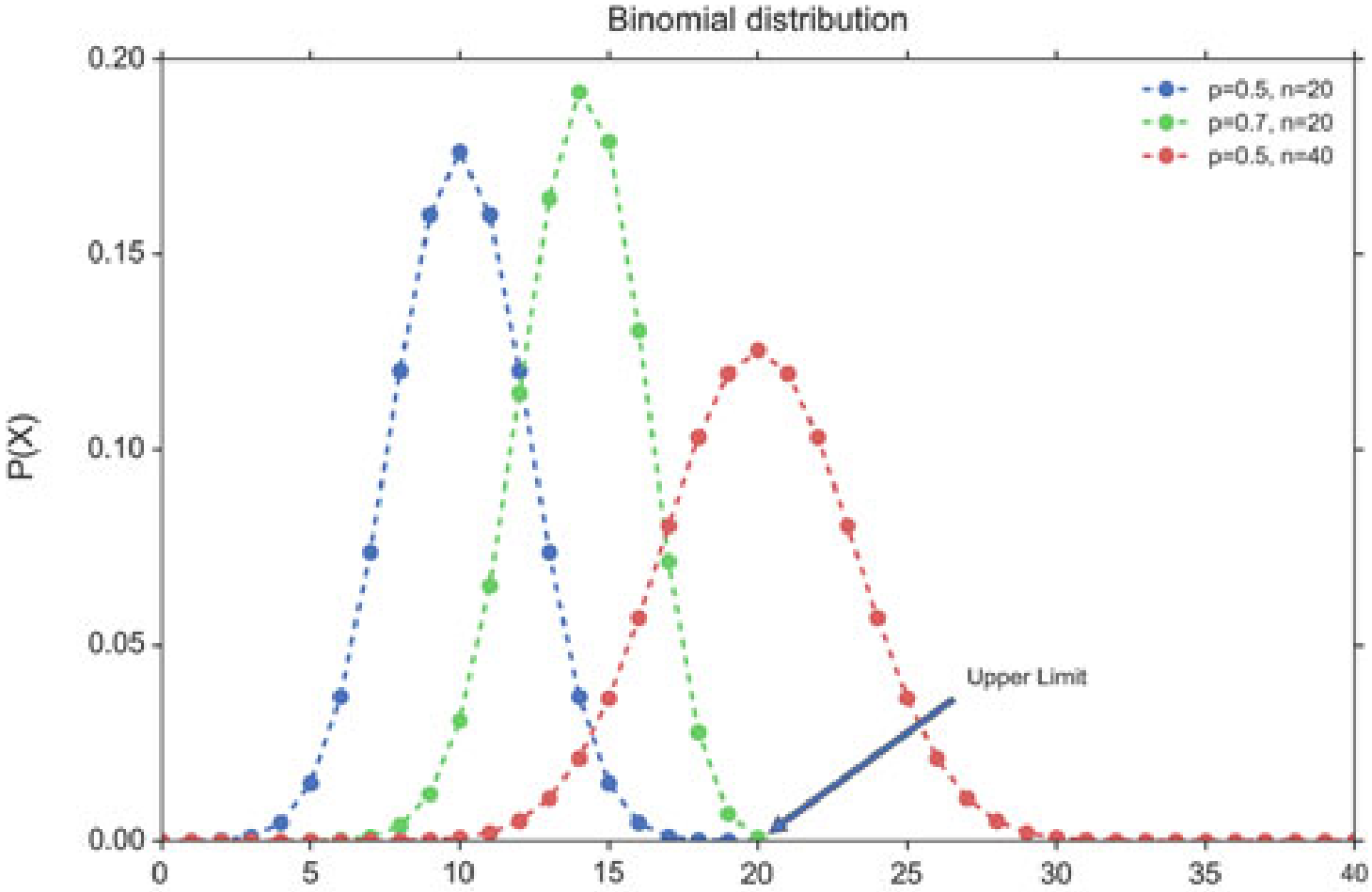
Diskrete Verteilungen für endliche Mengen

- **Bernoulli-Verteilung** = Beschreibung von zufälligen Ereignissen, bei denen es nur zwei mögliche Versuchsausgänge und eine vorgegebene Obergrenze gibt

```
from scipy import stats

p = 0.5
bernoulliDist = stats.bernoulli(p)

# Probability mass function
p_tails = bernoulliDist.pmf(0)
p_heads = bernoulliDist.pmf(1)

# And we can simulate 10 Bernoulli trials with
trials = bernoulliDist.rvs(10) # rvs = random variates
trials
```

Diskrete Verteilungen für endliche Mengen

- **Binominalverteilung** = beschreibt die Anzahl der Erfolge in einer Serie von gleichartigen und unabhängigen Versuchen, die jeweils genau zwei mögliche Ergebnisse haben („Erfolg“ oder „Misserfolg“).



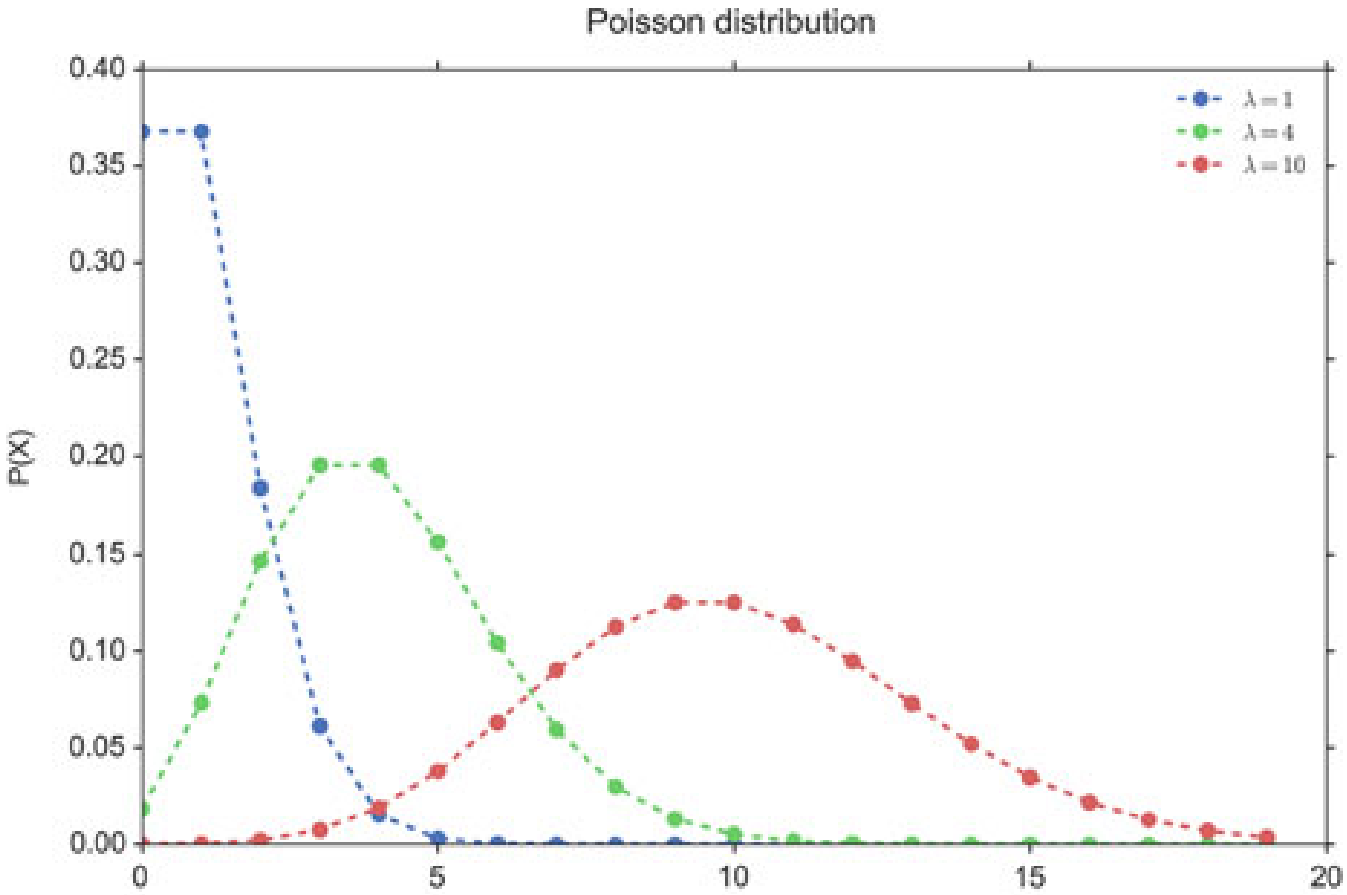
Diskrete Verteilungen für endliche Mengen

- **Binominalverteilung** = beschreibt die Anzahl der Erfolge in einer Serie von gleichartigen und unabhängigen Versuchen, die jeweils genau zwei mögliche Ergebnisse haben („Erfolg“ oder „Misserfolg“).

```
from scipy import stats
import numpy as np

# Frozen distribution function
(p, num) = (0.5, 4)
binomDist = stats.binom(num, p)

# calculate the probabilities how often heads come up
# during four tosses, given by the PMF
binomDist.pmf(np.arange(5))
```

Diskrete Verteilungen für unendliche Mengen

- **Poisson-Verteilung** = eine Wahrscheinlichkeitsverteilung, mit der die Anzahl von Ereignissen modelliert werden kann, die bei konstanter mittlerer Rate unabhängig voneinander in einem festen Zeitintervall oder räumlichen Gebiet eintreten.



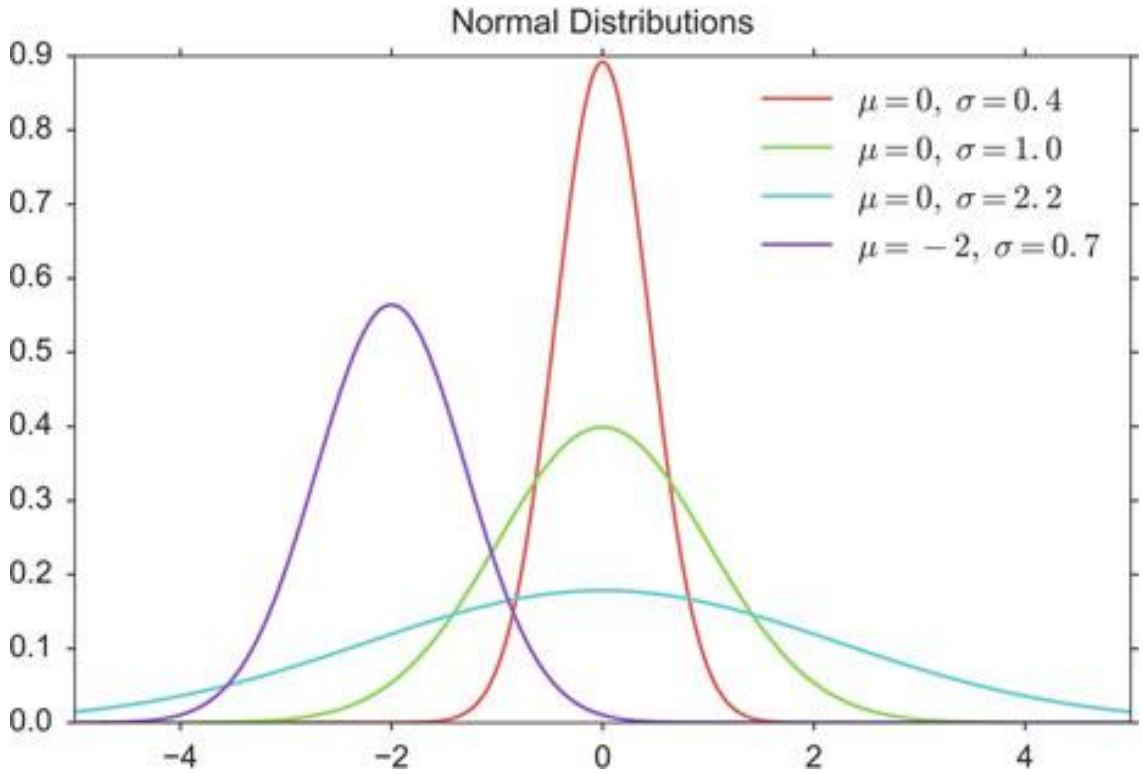
Diskrete Verteilungen für unendliche Mengen

- **Poisson-Verteilung** = eine Wahrscheinlichkeitsverteilung, mit der die Anzahl von Ereignissen modelliert werden kann, die bei konstanter mittlerer Rate unabhängig voneinander in einem festen Zeitintervall oder räumlichen Gebiet eintreten.

```
# Generate the distribution.
# Watch out NOT to divide integers,
# as "3/4" gives "0" in Python 2.x!
prob = 62./(365./7)
pd = stats.poisson(prob)

# Select the interesting numbers,
# calculate the PMF, and print the results
x = [0,2,5]
y = pd.pmf(x)*100
for num, solution in zip(x,y):
  print('''The chance of having {0} fatal accidents in one
        week is {1:4.1f}%.'''.format(num,solution))
```

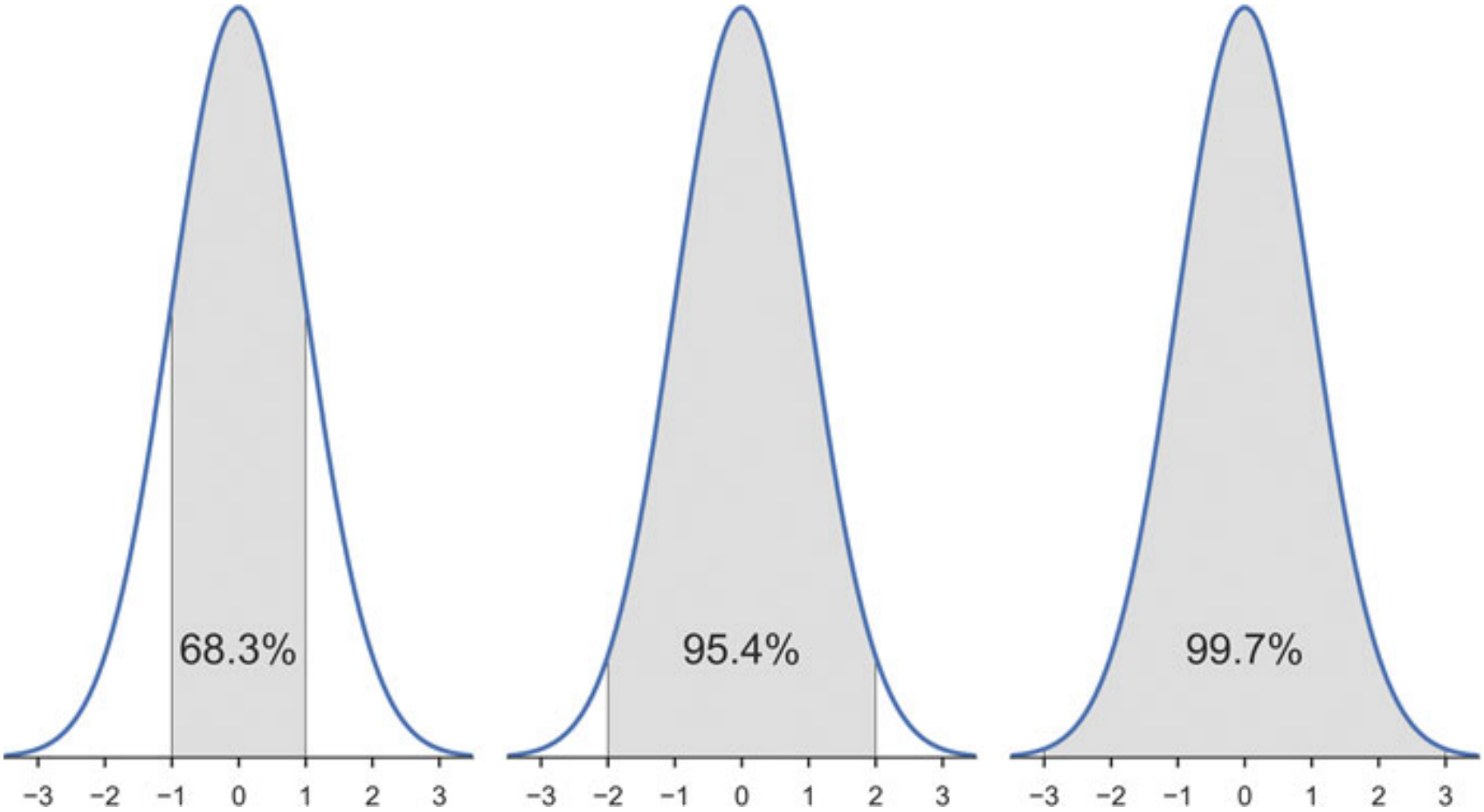
Kontinuerliche Verteilungen mit unbeschränktem Intervall

- **Normal- oder Gaußverteilung**
  - wichtigste Verteilungsfunktion in der Statistik
  - besitzt 2 Parameter (Mittelwert und Standardabweichung)
  - Die Summe oder Differenz von mehreren Normalverteilung ergibt wieder eine Normalverteilung



Kontinuerliche Verteilungen mit unbeschränktem Intervall

- **Normal- oder Gaußverteilung**



Fläche unter $\pm$ 1, 2, und 3 Standardabweichungen einer Normalverteilung

Kontinuerliche Verteilungen mit unbeschränktem Intervall

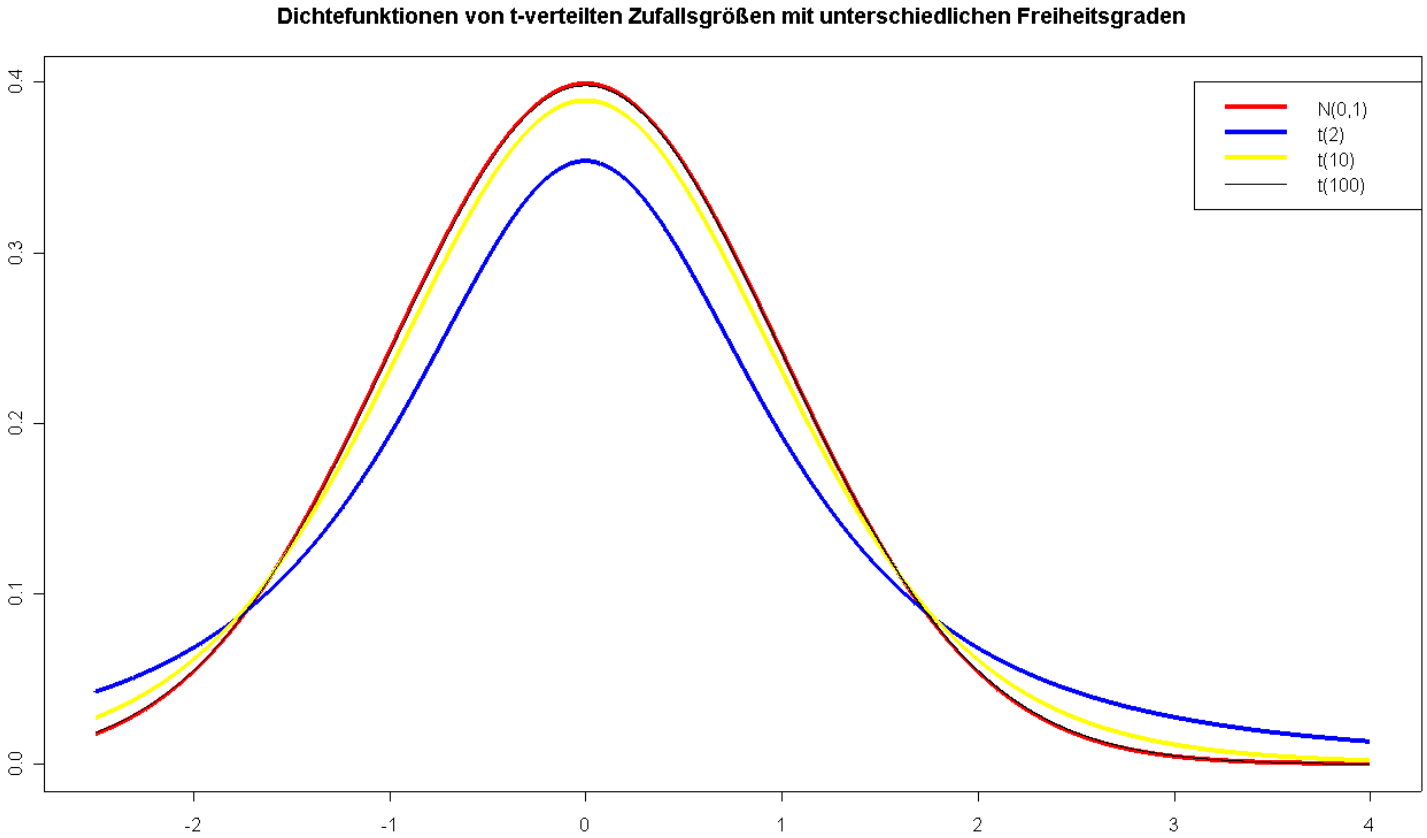
Normal- oder Gaußverteilung

Beispiel zur Berechnung des Intervals der PDF für Mittelwert = -2 und SD = 0.7 welche 95% der Daten beinhaltet.

```
import numpy as np
from scipy import stats

mu = -2
sigma = 0.7
myDistribution = stats.norm(mu, sigma)
significanceLevel = 0.05

myDistribution.ppf(
  [significanceLevel/2, 1-significanceLevel/2])
```

Verteilungen abgeleitet von der Normalverteilung

- **t-Verteilung** wird mit wachsendem n schmaler und geht für $n \to \infty$ in die Normalverteilung über
  - Wird bei niedrigen Stichprobengrößen verwendet, wenn der wahre Mittelwert und die wahre Standardabweichung unbekannt sind.



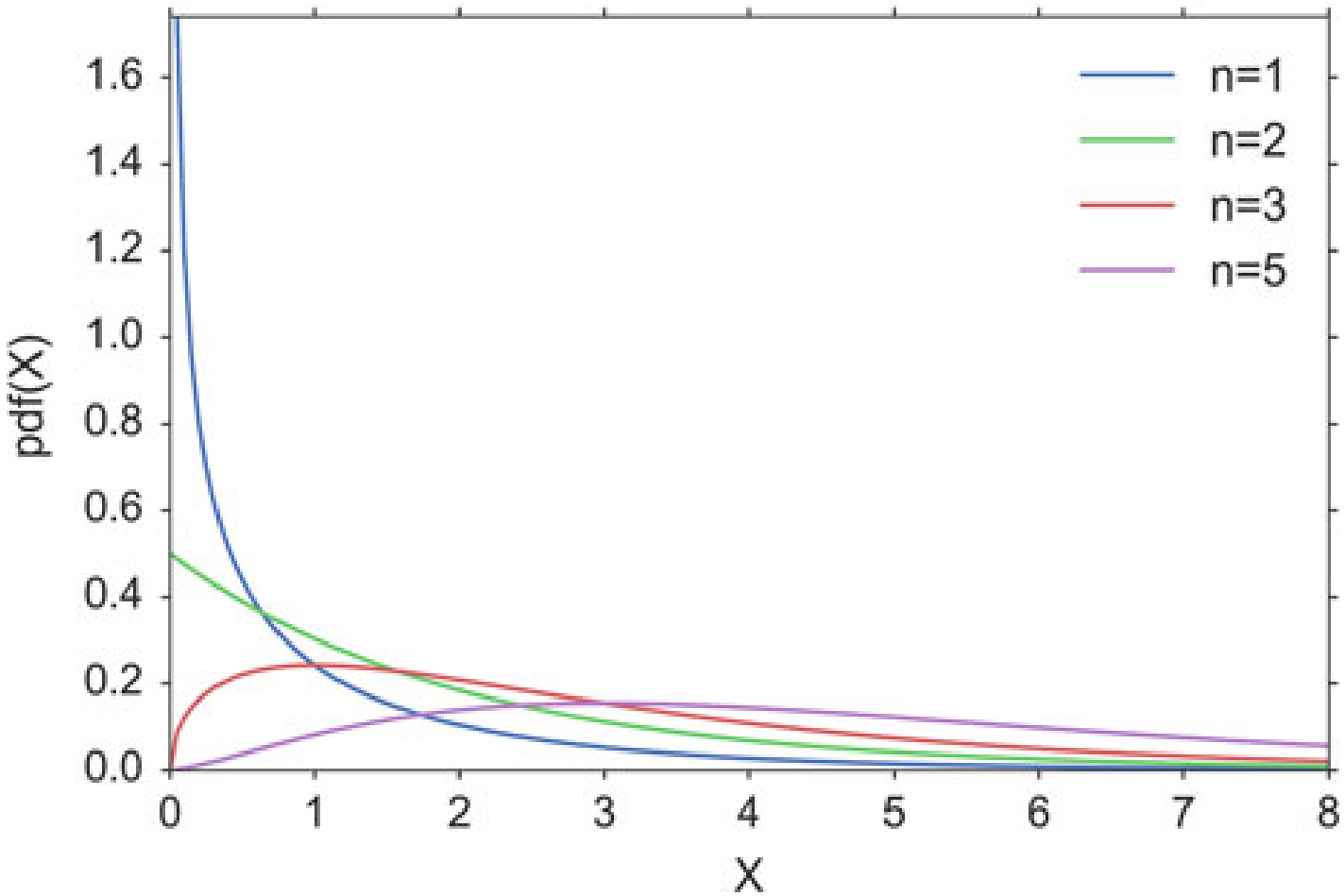
Verteilungen abgeleitet von der Normalverteilung

t-Verteilung

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

x = [52, 70, 65, 85, 62, 83, 59] # Enter the data

# Generate the t-distribution: DOF = length data minus 1.
td = stats.t(len(x)-1); alpha = 0.01

# From the t-distribution, you use the "PPF" function and
# multiply it with the standard error
tval = abs( td.ppf(alpha/2)*stats.sem(x) )
print('mean +/- 99%CI = {0:3.1f} +/- {1:3.1f}'.
      format(np.mean(x),tval))
```

Verteilungen abgeleitet von der Normalverteilung

- **Chi-Quadratverteilung** = eine stetige Wahrscheinlichkeitsverteilung über der Menge der nichtnegativen reellen Zahlen. Sie hat einen einzigen Parameter, nämlich die Anzahl der Freiheitsgrade n.



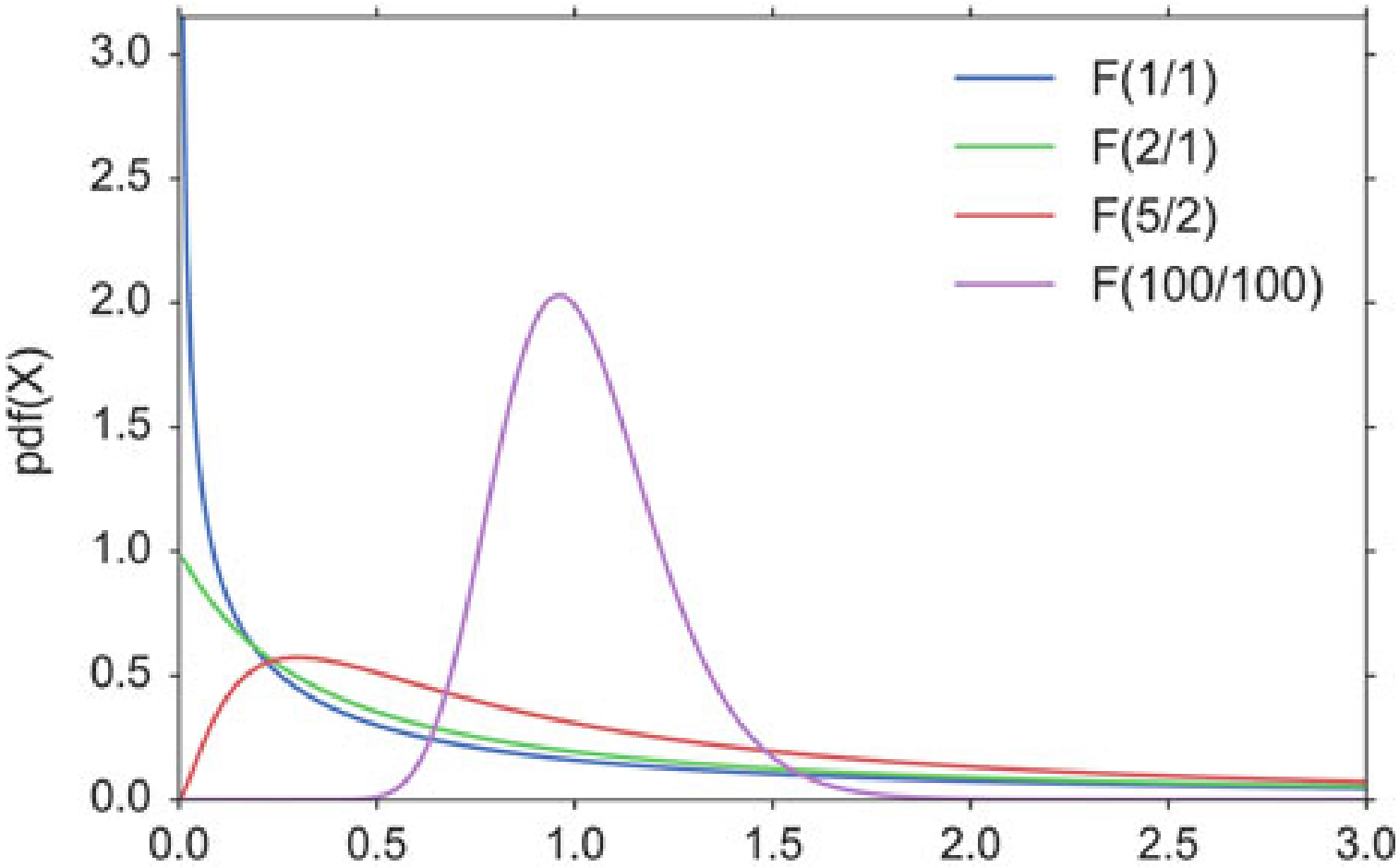
Verteilungen abgeleitet von der Normalverteilung

- **Chi-Quadratverteilung** mit 3 DOF

```
# Define the normal distribution
nd = stats.norm()

# Generate three sets of random variates
# from this distribution
numData = 1000
data1 = nd.rvs(numData)
data2 = nd.rvs(numData)
data3 = nd.rvs(numData)

# Show a histogram of the sum of the squares of
# these random data
plt.hist(data1**2+data2**2 +data3**2, 100)
plt.show()
```

Verteilungen abgeleitet von der Normalverteilung

- **F-Verteilung** = Quotient zweier jeweils durch die zugehörige Anzahl der Freiheitsgrade geteilter Chi-Quadrat-verteilter Zufallsvariablen. Die F-Verteilung besitzt zwei unabhängige Freiheitsgrade als Parameter.



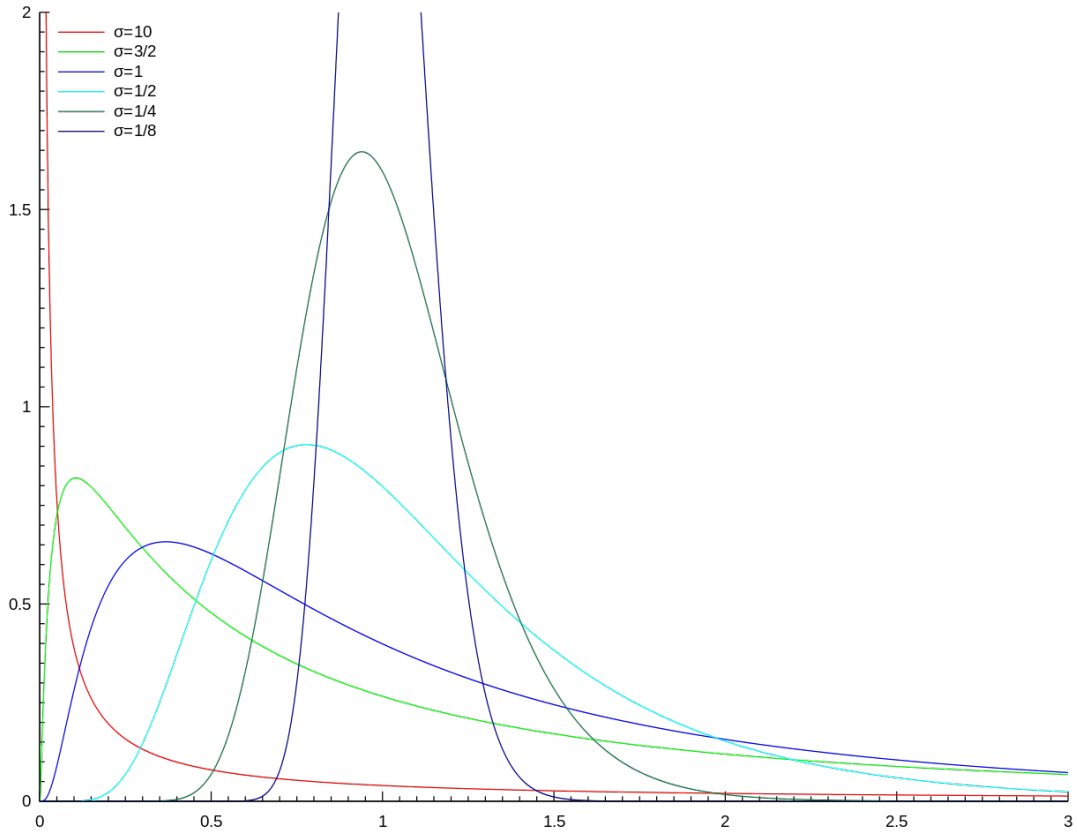
Verteilungen abgeleitet von der Normalverteilung

F-Verteilung

```
apples1 = [110, 121, 143]
apples2 = [88, 93, 105, 124]

fval = np.std(apples1, ddof=1)/np.std(apples2, ddof=1)
fd = stats.distributions.f(len(apples1),len(apples2))
pval = fd.cdf(fval)

print('The p-value of the F-distribution = {0}.'.format(pval))
if pval>0.025 and pval<0.975:
  print('The variances are equal.')
```

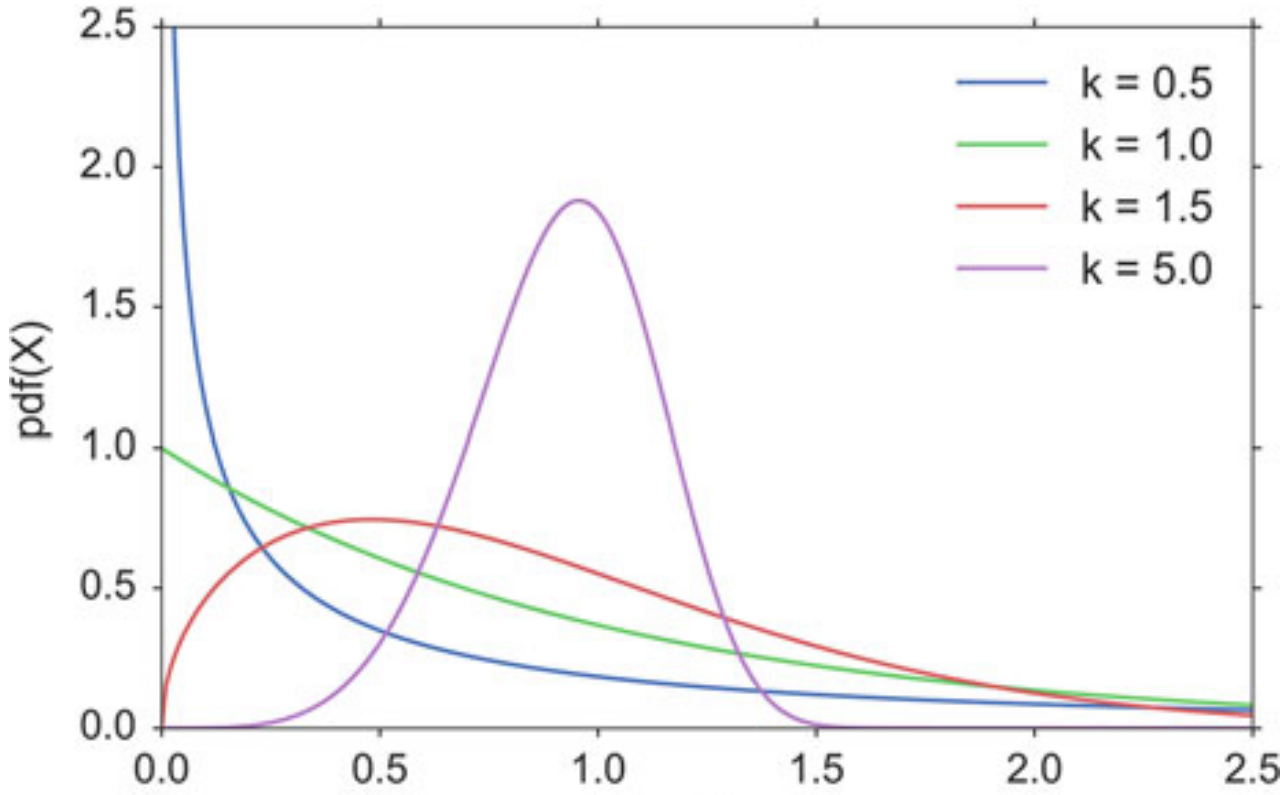
weitere kontinuierliche Verteilungsfunktionen

- **Log-Normalverteilung** = kontinuierliche Wahrscheinlichkeitsverteilung für eine Variable, die nur positive Werte annehmen kann. Sie beschreibt die Verteilung einer Zufallsvariablen die mit dem Logarithmus transformiert normalverteilt ist.



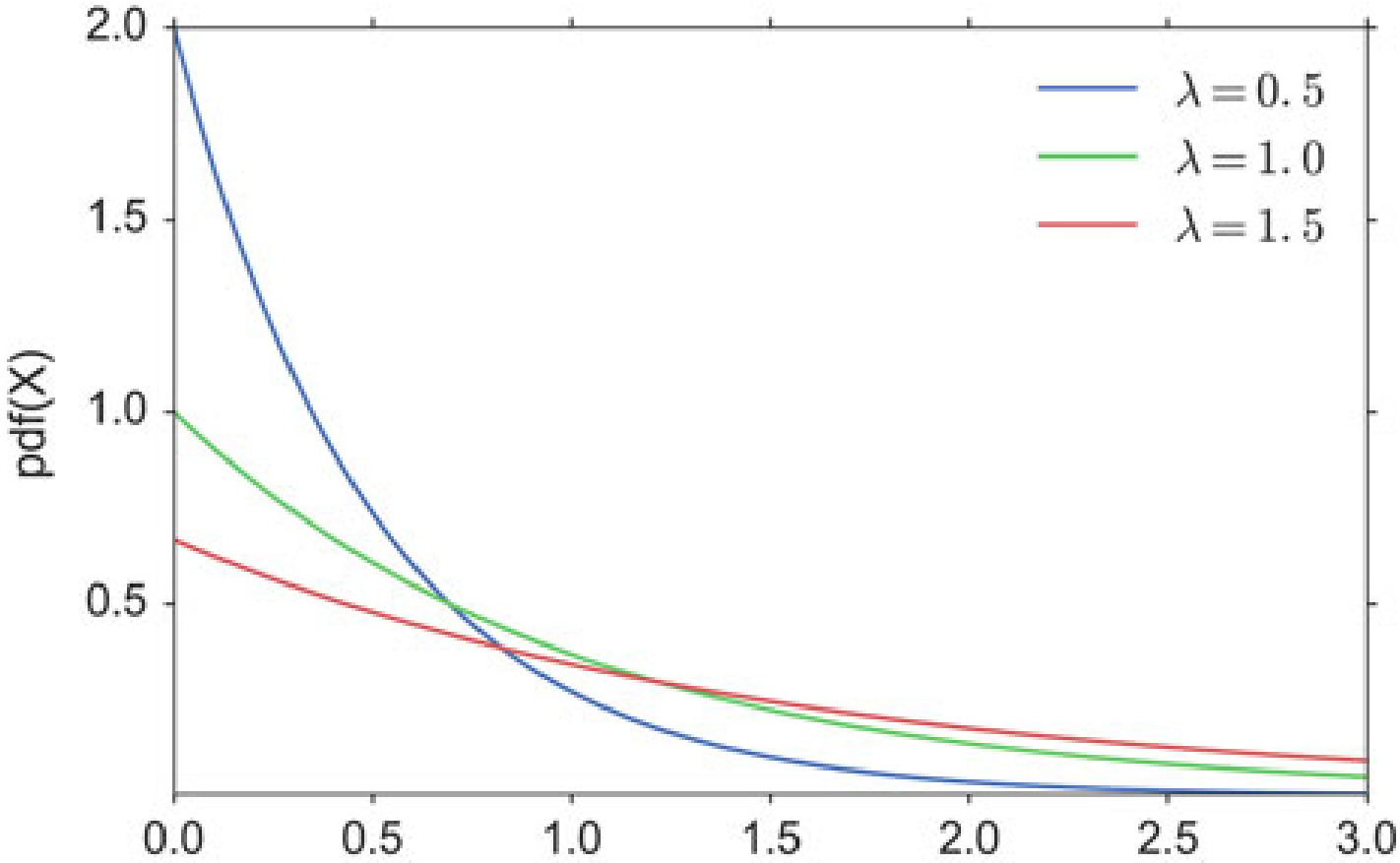
weitere kontinuierliche Verteilungsfunktionen

- **Weibull-Verteilung** = zweiparametrige stetige Wahrscheinlichkeitsverteilung über der Menge der positiven reellen Zahlen. Sie wird unter anderem zur statistischen Modellierung von Windgeschwindigkeiten oder zur Beschreibung der Lebensdauer und Ausfallhäufigkeit von elektronischen Bauelementen herangezogen.



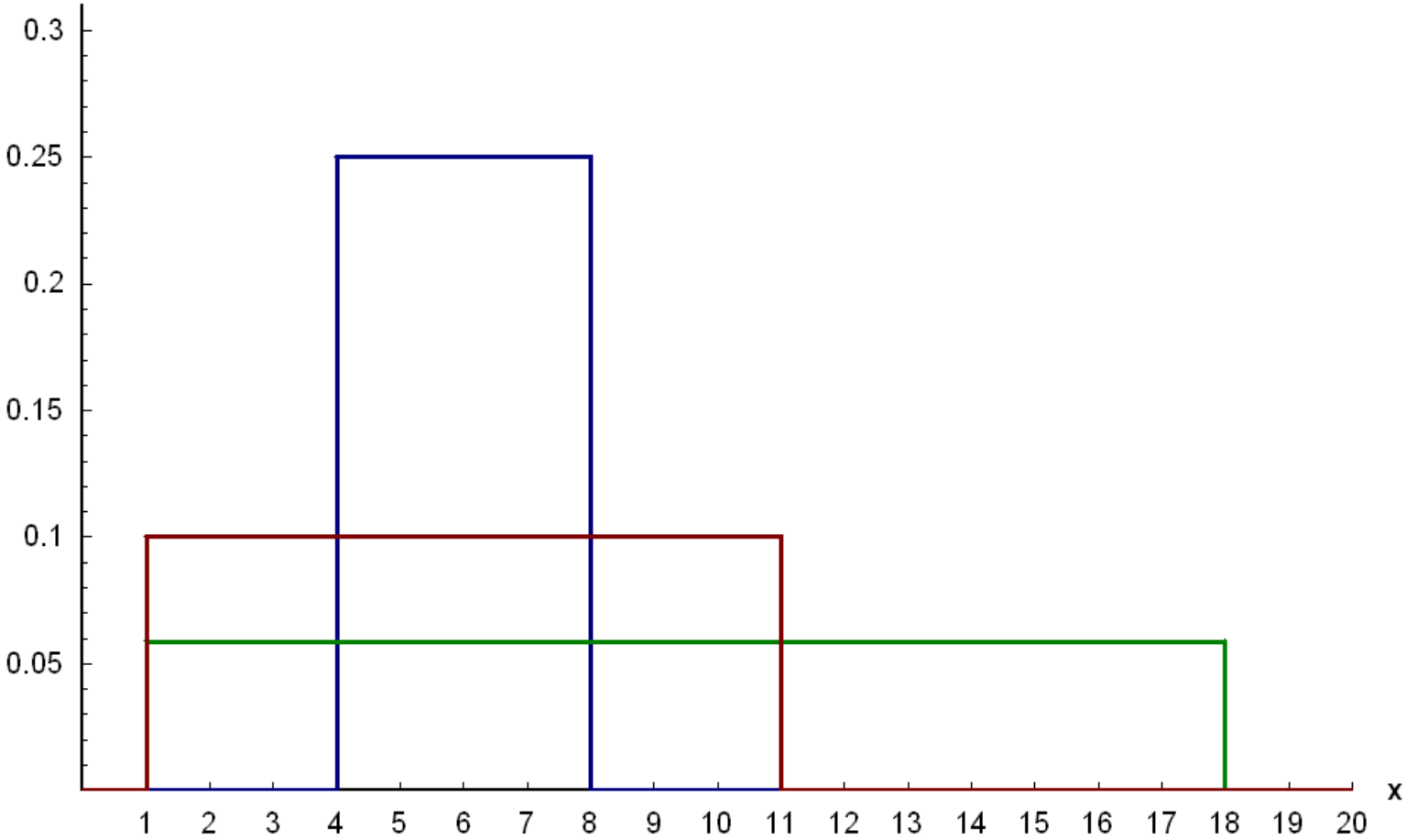
weitere kontinuierliche Verteilungsfunktionen

- **Exponentialverteilung** = stetige Wahrscheinlichkeitsverteilung über der Menge der nicht-negativen reellen Zahlen, die durch eine Exponentialfunktion gegeben ist, z.B. Dauer von zufälligen Zeitintervallen



weitere kontinuierliche Verteilungsfunktionen

- **Stetige Gleichverteilung** = alle Teilintervalle gleicher Länge besitzen dieselbe Wahrscheinlichkeit



Normalitätsprüfung

Wahrscheinlichkeitsdiagramme

- **QQ-Plots** - Die Quantile eines bestimmten Datensatzes werden gegen die Quantile einer Referenzverteilung, typischerweise der Standardnormalverteilung, aufgetragen.
- **PP-Plots** - Die CDF (cumulative-distribution-function) eines gegebenen Datensatzes wird gegen die CDF einer Referenzverteilung aufgetragen.
- **Probability Plots** - Die geordneten Werte eines gegebenen Datensatzes werden gegen die Quantile einer Referenzverteilung aufgetragen.

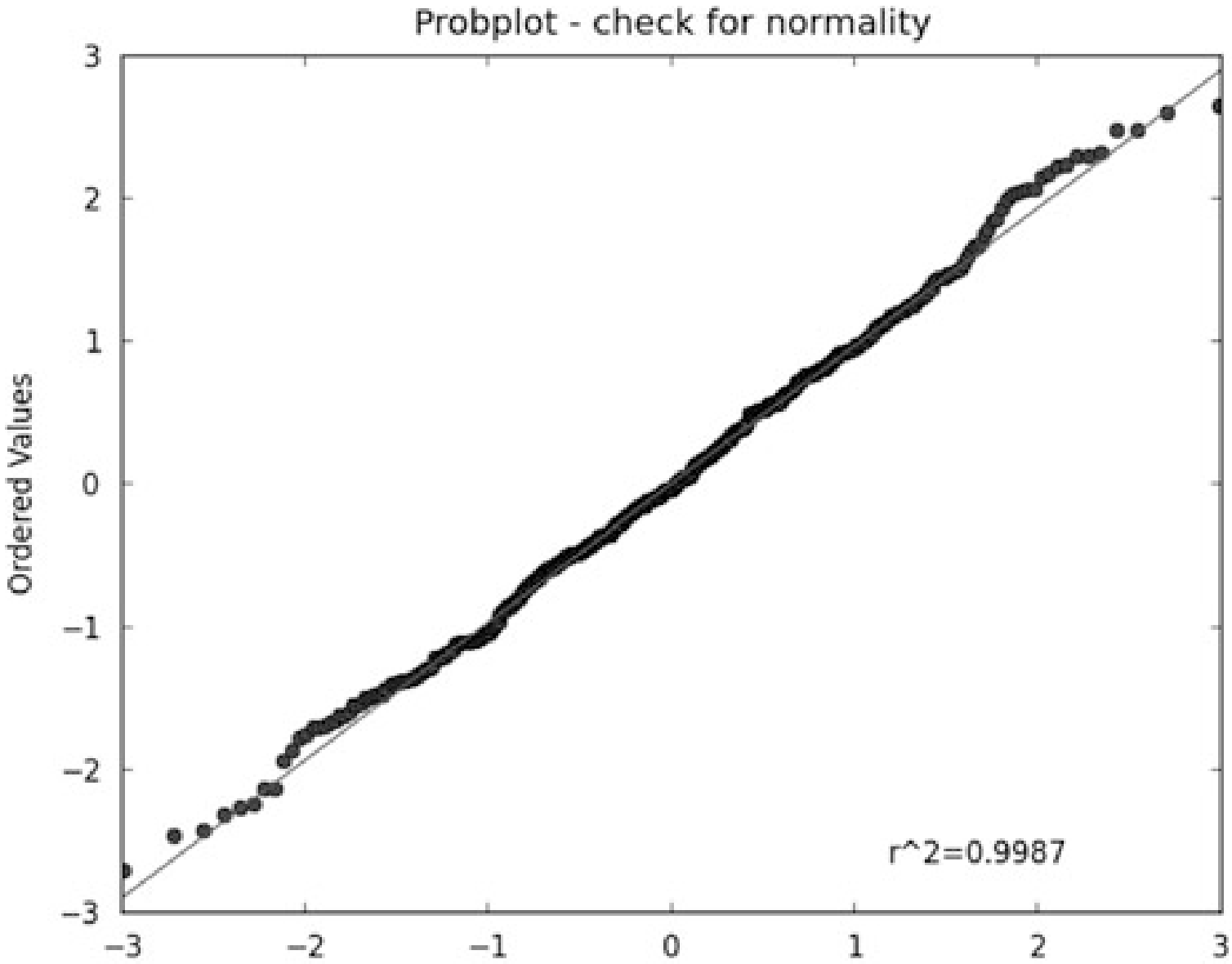
- In allen drei Fällen sind die Ergebnisse ähnlich:
  - Wenn die beiden zu vergleichenden Verteilungen ähnlich sind, liegen die Punkte etwa auf der Linie $y = x$.
  - Wenn die Verteilungen linear verbunden sind, liegen die Punkte etwa auf einer Linie, aber nicht unbedingt auf der Linie $y = x$.

Wahrscheinlichkeitsdiagramme

- In Python kann mit dem Befehl **probplot()** ein Wahrscheinlichkeitsdiagramm erstellt werden:

```
from scipy import stats
import matplotlib.pyplot as plt
nsample = 100
np.random.seed(7654321)

# A t distribution with small degrees of freedom:
x = stats.t.rvs(3, size=nsample)
res = stats.probplot(x, plot=plt)
plt.show()
```

Wahrscheinlichkeitsdiagramme



Normalitätstests

- Bei Normalitätstests können unterschiedliche Herausforderungen auftreten:
  - Manchmal sind nur wenige Stichproben verfügbar
  - Manchmal liegen viele Daten, aber einige extrem abweichende Werte vor
- Tests zur Beurteilung der Normalität (oder der Ähnlichkeit mit einer bestimmten Verteilung) lassen sich grob in zwei Kategorien einteilen:
  - Tests, die auf einem Vergleich ("best fit") mit einer bestimmten Verteilung basieren, die oft in Bezug auf ihre CDF spezifiziert ist: **Kolmogorov-Smirnov-Test**, Lilliefors-Test, Anderson-Darling-Test, Cramer-von-Mises-Kriterium, **Shapiro-Wilk Test** und Shapiro-Francia Test.
  - Tests, die auf einer deskriptiven Statistik der Stichprobe basieren: Schräglauftest, der Kurtosetest, der **D'Agostino-Pearson Omnibus-Test** oder der Jarque-Bera-Test.

Normalitäts-Tests

- Der Python-Befehl:

```
stats.normaltest(x)
```

  verwendet den D'Agostino-Pearson Omnibus-Test. Dieser Test kombiniert einen Schiefe- und Kurtosis-Test zu einer einzigen, globalen "Omnibus"-Statistik.

- Mehr gebräuchlich ist bei kleinen Stichproben (weniger als 50 Datenpunkte) der **Shapiro-Wilk-Test**:

```
stats.shapiro(x)
```

- Und wenn die Probe mehr als 50 Datenpunkte hat, der **Kolmogorov-Smirnov-Test**:

```
stats.kstest(x)
```

Daten-Transformation

- Wenn die Daten signifikant von einer Normalverteilung abweichen, ist es manchmal ratsam, die Daten zu transformieren um sie einer Normalverteilung anzunähern.
- Oft haben Daten Werte, die nur positiv sein können (z.B. die Größe von Personen): Solche Daten können oft durch eine **Log-Transformation** normalisiert werden:

```
# example dataframe
df = pd.DataFrame({'a': [0, 1, 2, 3],
                   'b': [4, 5, np.nan, 7],
                   'c': [8, 9, 10, 11]})


# apply log(x+1) element-wise to a subset of columns
df_log = df[['a', 'b']].applymap(lambda x: np.log(x+1))
```

- **Weitere Transformationen:** Wurzel-Transformation, Power-Transformation, Box-Cox Transformation, Logit Transformation

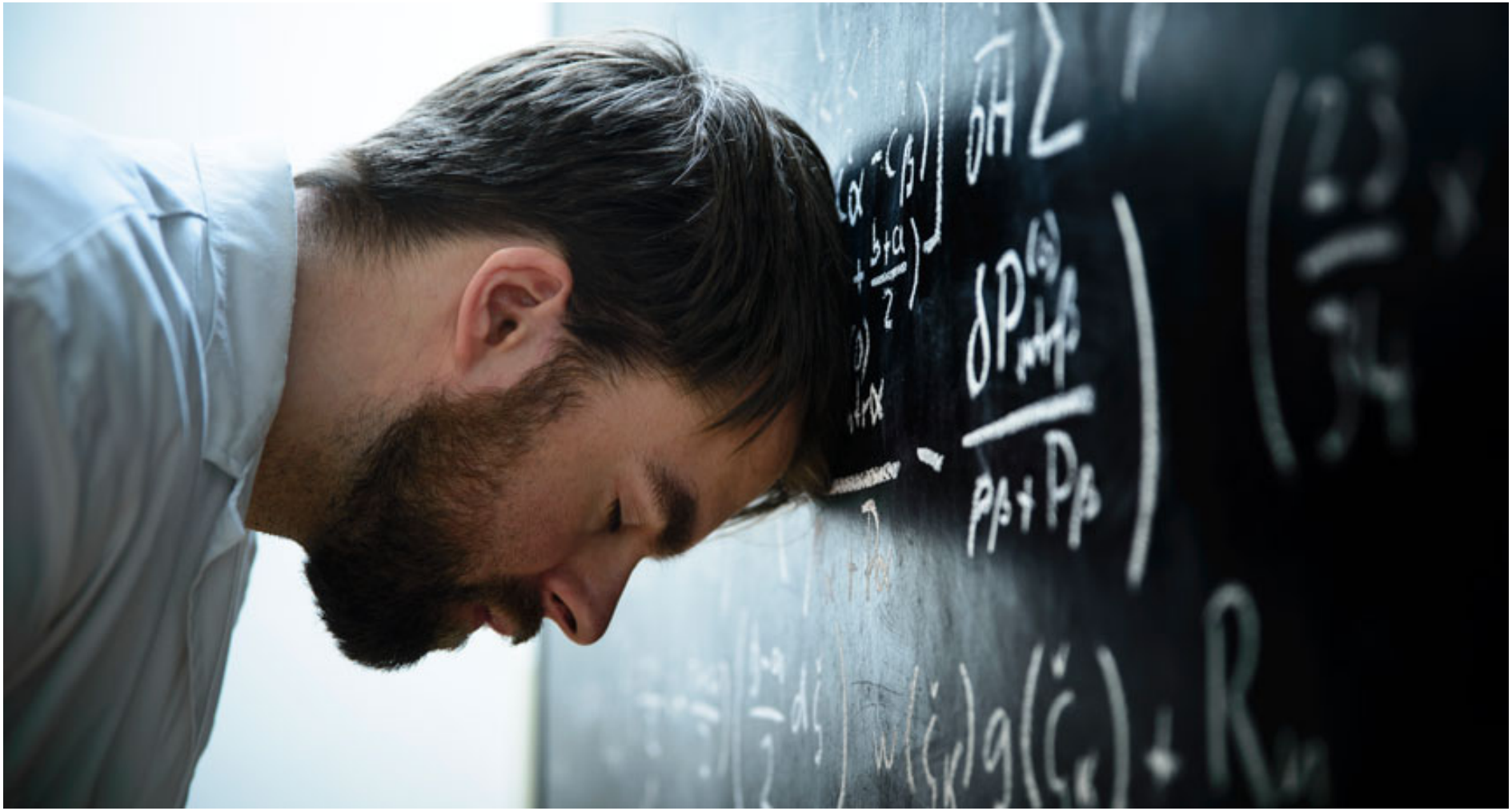
Fehlerwerte, p-Wert und Stichprobenumfang

Die Interpretation des p-Wertes

- Ein Wert von $p < 0,05$ für die Nullhypothese ist wie folgt zu interpretieren: Wenn die Nullhypothese wahr ist, ist die Chance, eine Teststatistik zu finden, die so extrem wie oder extremer als die beobachtete Statistik ist, weniger als 5 %. Das ist nicht dasselbe wie zu sagen, dass die Nullhypothese falsch ist, und noch weniger, dass eine alternative Hypothese wahr ist!
- Die Angabe eines p-Wertes allein ist für die statistische Analyse von Daten heutzutage nicht mehr ausreichend. Darüber hinaus sollten auch die Konfidenzintervalle für die zu untersuchenden Parameter angegeben werden.

Statistische Signifikanz

- Statistiker wollen auf das Standardmaß der Wissenschaft verzichten



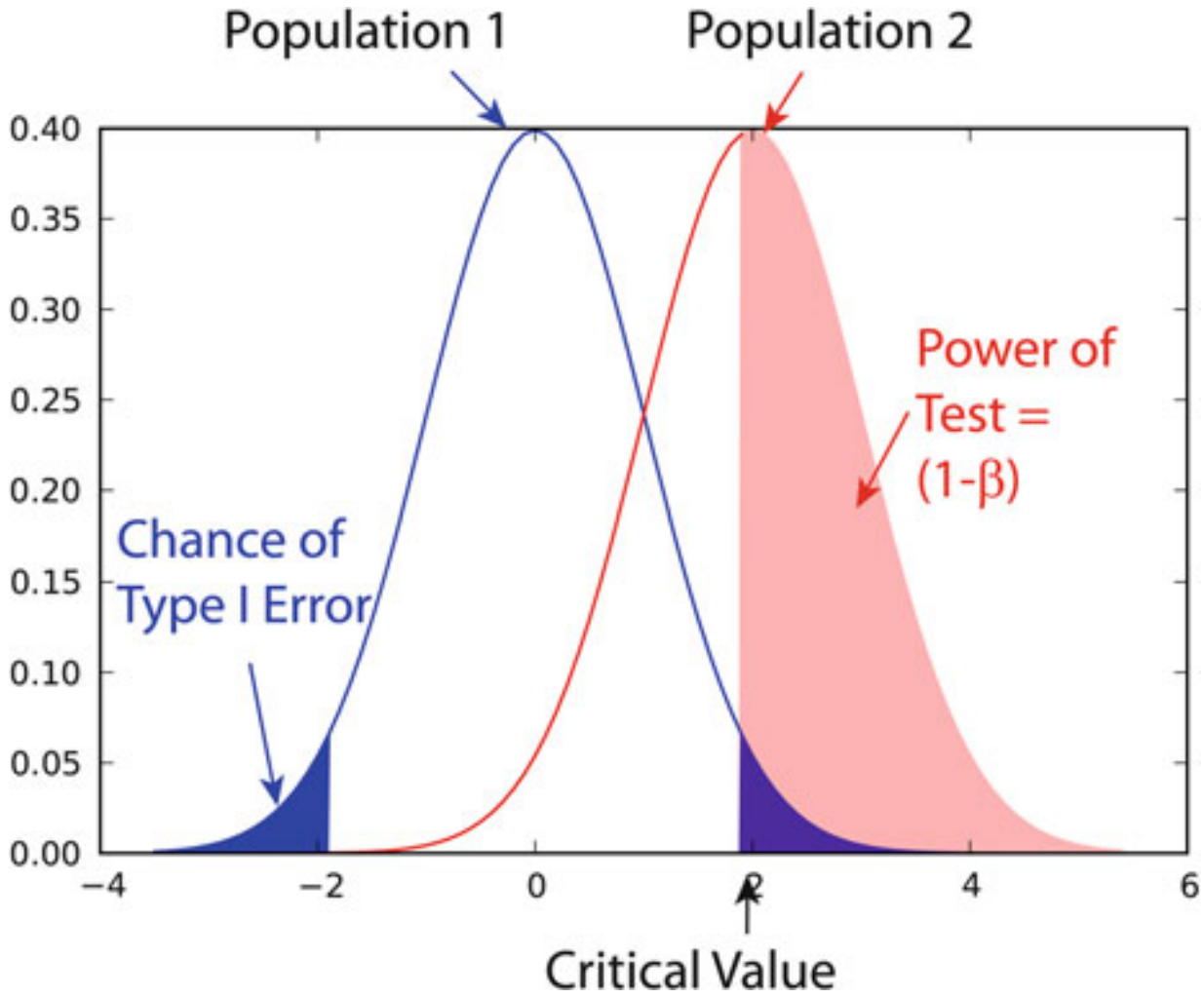
https://www.sciencenews.org/article/statisticians-standard-measure-significance-p-values?tgt=nr

Fehlerarten

- Beim Hypothesentest können zwei Arten von Fehlern auftreten.
- **Typ-I-Fehler** = Fehler, bei denen das Ergebnis signifikant ist, obwohl die Null Hypothese wahr ist.
  - Die Wahrscheinlichkeit eines Typ-I-Fehlers wird üblicherweise mit α angegeben und vor Beginn der Datenanalyse festgelegt.
  - Ein Typ-I-Fehler wäre die Diagnose von Krebs ("positives" Testergebnis), obwohl die Testperson gesund ist.
- **Typ-II-Fehler** = Fehler, bei denen das Ergebnis nicht signifikant ist, trotz der Tatsache, dass die Null-Hypothese falsch ist.
  - Ein Typ-II-Fehler wäre eine "gesunde" Diagnose ("negatives" Testergebnis), auch wenn der Betroffene Krebs hat.
  - Die Wahrscheinlichkeit für diese Art von Fehler wird üblicherweise mit β angegeben.
  - Die "Stärke" eines statistischen Tests ist definiert als $(1-\beta) * 100$ und ist die Chance, die alternative Hypothese richtig zu akzeptieren.

Stärke eines statistischen Tests

- Um die Stärke eines Tests zu finden, benötigt man eine alternative Hypothese.



Stichprobengröße

- Die Stärke oder Empfindlichkeit eines binären Hypothesentests ist die Wahrscheinlichkeit, dass der Test die Nullhypothese korrekt ablehnt, wenn die alternative Hypothese wahr ist.
- Die Bestimmung der Stärke eines statistischen Tests und die Berechnung der Mindeststichprobengröße, die erforderlich ist, um einen Effekt einer bestimmten Größe zu erkennen, wird als Poweranalyse bezeichnet.
- Sie beinhaltet 4 Faktoren:
  1. Wahrscheinlichkeit für Typ-I-Fehler
  2. Wahrscheinlichkeit für Typ-II-Fehler
  3. Effektgröße, d.h. die Größe des untersuchten Effekts im Verhältnis zur Standardabweichung der Probe.
  4. Stichprobenumfang

Berechnung der Stichprobengröße

- Möchte man im Rahmen einer Studie eine Befragung durchführen, ist es interessant zu wissen, wie viele Personen befragt werden müssen.
- Der Stichprobenumfang bzw. die Stichprobengröße kann berechnet werden und hängt davon ab, wie sicher oder überzeugt man von seinen Ergebnissen sein möchte.
- Der Umfang einer Stichprobe erhöht sich, je sicherer man sich sein möchte.
- Zur Berechnung der Stichprobengröße benötigen wir unterschiedliche Parameter, unter anderem den Z-Wert, der sich aus der z-Verteilung ergibt.
- Der Stichprobenumfang lässt sich auch berechnen, wenn die Grundgesamtheit unbekannt ist.

Stichprobengröße bei bekannter Population

```
from scipy import stats
import math

def sample_size_pop(N, e=0.05, c=0.95, p=0.5, extra=None):
  z = stats.norm.ppf((1 + c) / 2)
  frac_n = (z**2 * p*(1-p)) / e**2
  frac_d = 1 + ((z**2 * p*(1-p)) / (e**2 * N))
  n = frac_n / frac_d
  if extra:
    n = n + n * extra
  return math.ceil(n)   # Werte aufrunden

# Beispiel
n = 4000
sample_size_pop(n, c=0.99, e=0.03, p=0.5, extra=0.05)
```

Stichprobengröße bei unbekannter Population

```
from scipy import stats
import math
def sample_size(e=0.05, c=0.95, p=0.5, extra=None):
  z = stats.norm.ppf((1 + c) / 2)
  n = (z**2 * p * (1-p)) / e**2
  if extra:
   n = n + n * extra
  return math.ceil(n)

sample_size(p=0.09, e=0.01, c=0.95)
```

Sensitivität und Spezifität

- Sensitivität = Anteil der korrekt als positiv klassifizierten Objekte an der Gesamtheit der tatsächlich positiven Objekte
- Spezifität = Anteil der korrekt als negativ klassifizierten Objekte an der Gesamtheit der in Wirklichkeit negativen Objekte

		Condition		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	False Positive (Type I error)	**Positive predictive value=** $\frac{\Sigma \text{True Positive}}{\Sigma \text{Test Outcome Positive}}$
	Test Outcome Negative	False Negative (Type II error)	**True Negative**	**Negative predictive value=** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{True Positive}}{\Sigma \text{Condition Positive}}$	**Specificitivity =** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Condition Negative}}$	

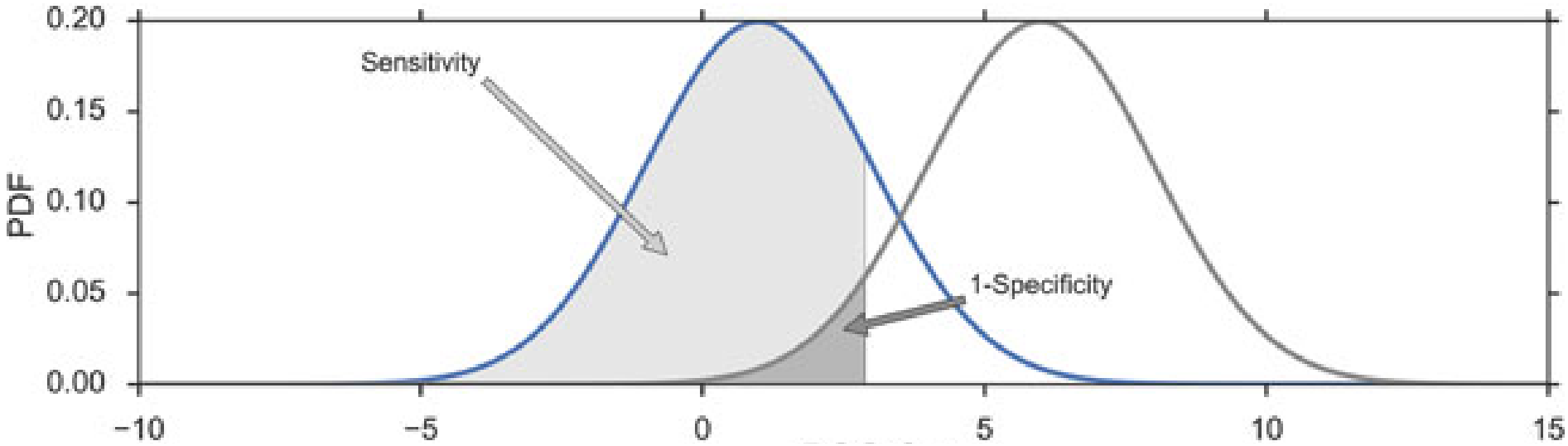
Sensitivität und Spezifität

- Positiver Vorhersagewert (PPV) = Anteil der korrekt als positiv klassifizierten Objekte an der Gesamtheit der als positiv klassifizierten Ergebnisse
- Negativer Vorhersagewert (NPV) = Anteil der korrekt als negativ klassifizierten Objekte an der Gesamtheit der als negativ klassifizierten Ergebnisse

		Condition		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	False Positive (Type I error)	**Positive predictive value=** $\frac{\Sigma \text{True Positive}}{\Sigma \text{Test Outcome Positive}}$
	Test Outcome Negative	False Negative (Type II error)	**True Negative**	**Negative predictive value=** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Test Outcome Negative}}$
		Sensitivity = $\frac{\Sigma \text{True Positive}}{\Sigma \text{Condition Positive}}$	**Specificitivity =** $\frac{\Sigma \text{True Negative}}{\Sigma \text{Condition Negative}}$	

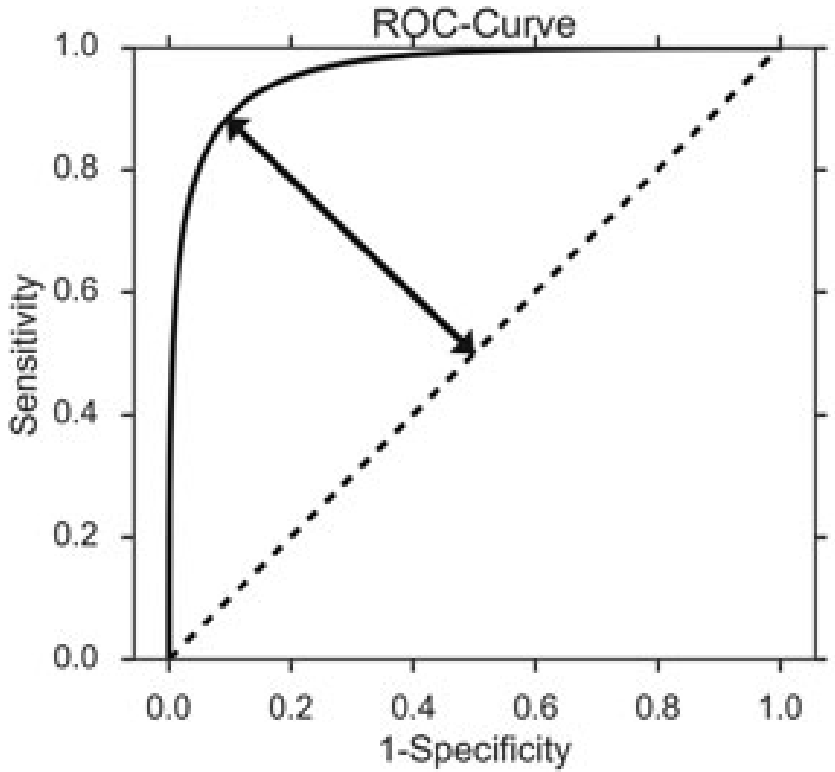
ROC-Kurve

- Eng verbunden mit Sensitivität und Spezifität ist die Receiver-Operating-Characteristic (ROC)-Kurve.
- Dies ist ein Diagramm, das die Beziehung zwischen der wahren positiven Rate (auf der vertikalen Achse) und der falsch positiven Rate (auf der horizontalen Achse) darstellt.



ROC-Kurve

- Eng verbunden mit Sensitivität und Spezifität ist die Receiver-Operating-Characteristic (ROC)-Kurve.
- Dies ist ein Diagramm, das die Beziehung zwischen der wahren positiven Rate (auf der vertikalen Achse) und der falsch positiven Rate (auf der horizontalen Achse) darstellt.



Induktive Statistik

Tests der Mittelwerte numerischer Daten

- Hypothesentests für die Mittelwerte von Gruppen
  - Vergleich einer Gruppe mit einem Festwert.
  - Vergleich von zwei Gruppen im Verhältnis zueinander.
  - Vergleich von drei oder mehr Gruppen miteinander.
- In jedem Fall unterscheiden wir zwischen zwei Fällen.
- Wenn die Daten annähernd normalverteilt sind, können die sogenannten **parametrischen Tests** verwendet werden.
- Diese Tests sind empfindlicher als nicht-parametrische Tests, erfordern aber, dass bestimmte Annahmen erfüllt sind.
- Wenn die Daten nicht normalverteilt sind oder nur in geordneter Form vorliegen, sollten die entsprechenden **nonparametrischen Tests** verwendet werden.

Einstichproben-t-Test

- Um den Mittelwert normalverteilter Daten mit einem Referenzwert zu vergleichen, verwenden wir typischerweise den Einstichproben-t-Test, der auf der t-Verteilung basiert.
- In Python können Teststatistik und p-Wert für den Einstichproben-t-Test wie folgt berechnet werden:

```
# One-sample t-Test
from scipy import stats

np.random.seed(7654567)  # fix seed to get same result
rvs = stats.norm.rvs(loc=5, scale=10, size=(50,2))

# Test if mean of random sample is equal to true mean
t, pVal = stats.ttest_1samp(rvs,5.0)
print(t); print(pVal)
```

Wilcoxon-Vorzeichen-Rang-Test

- Wenn die Daten nicht normalverteilt sind, sollte der Einstichproben-t-Test nicht verwendet werden (obwohl dieser Test ziemlich robust gegenüber Abweichungen von der Normalverteilung ist).
- Stattdessen müssen wir einen nichtparametrischen Test auf den Mittelwert anwenden. Wir können dies tun, indem wir den Wilcoxon Rangsummentest durchführen.
- Beachtet, dass dieser Test im Gegensatz zum Einstichproben-t-Test nach einer Differenz von Null sucht.

Wilcoxon-Vorzeichen-Rang-Test

- In Python kann der Wilcoxon-Vorzeichen-Rang-Test wie folgt berechnet werden:

```
# Wilcoxon Signed Rank Sum test
d = [6, 8, 14, 16, 23, 24, 28, 29, 41, -48, 49, 56, -60]
rank, pVal = stats.wilcoxon(d)
print(rank); print(pVal)
```

- Diese Methode besteht aus drei Schritten:
  1. Berechnen Sie die Differenz zwischen jeder Beobachtung und dem Wert des Interesses
  2. Die Zeichen der Unterschiede ignorierend, ordnen Sie sie in der Größenordnung.
  3. Berechnen Sie die Summe der Ränge aller negativen (oder positiven) Ränge, entsprechend den Beobachtungen unterhalb (oder oberhalb) des gewählten hypothetischen Wertes.

Gepaarter t-Test

- Beim Vergleich von zwei Gruppen miteinander sind zwei Fälle zu unterscheiden.
- Im ersten Fall werden zwei Werte, die zu unterschiedlichen Zeiten vom gleichen Subjekt aufgenommen wurden, miteinander verglichen.
- Zum Beispiel die Größe der Schüler beim Eintritt in die Grundschule und nach dem ersten Jahr, um zu überprüfen, ob sie gewachsen sind.
- Da uns nur der Unterschied in jedem Probanden zwischen der ersten und der zweiten Messung interessiert, wird dieser Test als gepaarter t-Test bezeichnet und ist im Wesentlichen gleichbedeutend mit einem Ein-Stichproben-Test für die mittlere Differenz.

Gepaarter t-Test

- Daher liefern die beide Tests **stats.ttest_1samp** und **stats.ttest_rel** das gleiche Ergebnis (bis auf winzige numerische Unterschiede).

```
# Load libraries
import numpy as np
from scipy import stats

# Create random data for two time periods
np.random.seed(1234)
data = np.random.randn(10)+0.1
data1 = np.random.randn(10)*5 # dummy data
data2 = data1 + data

# paired t-test
print(stats.ttest_rel(data2, data1))
# same group-difference as "data"

# one-sample t-test on data
print(stats.ttest_1samp(data, 0))
```

Zweistichproben-t-Test

- Ein ungepaarter t-Test vergleicht zwei von einander unabhängige Gruppen.
- Ein Beispiel wäre der Vergleich der Wirkung von zwei Medikamenten, die an zwei verschiedene Patientengruppen abgegeben werden.
- Die Grundidee ist die gleiche wie beim one-sample t-Test. Aber statt der Varianz des Mittelwerts, benötigen wir jetzt die Varianz der Differenz zwischen den Mittelwerten der beiden Gruppen.

```
# two-sample t-Test
rvs1 = stats.norm.rvs(loc=5,scale=10,size=500)
rvs2 = stats.norm.rvs(loc=5,scale=10,size=500)

t_stat, pVal = stats.ttest_ind(rvs1,rvs2)
print(pVal)
```

Mann-Whitney test

- Wenn die Messwerte von zwei Gruppen nicht normalverteilt sind, müssen wir auf einen nichtparametrischen Test zurückgreifen.
- Der häufigste nichtparametrische Test für den Vergleich zweier unabhängiger Gruppen ist der Mann-Whitney-Test.
- Die Teststatistik für diesen Test wird üblicherweise mit u angegeben:

```
# Create two groups of data
group1 = [1, 5 ,7 ,3 ,5 ,8 ,34 ,1 ,3 ,5 ,200, 3]
group2 = [10, 18, 11, 12, 15, 19, 9, 17, 1, 22, 9, 8]

# Calculate u and probability of a difference
u_statistic, pVal = stats.mannwhitneyu(group1, group2)

# Print p-Value
print (pVal)
```

Varianz analyse (ANOVA)

- Varianzanalyse (ANOVA) = Unterteilung der Varianz in die Varianz zwischen Gruppen und innerhalb von Gruppen um zu sehen, ob diese Verteilungen der Nullhypothese entsprechen, dass alle Gruppen aus der gleichen Verteilung stammen
- Die Variablen, die die verschiedenen Gruppen unterscheiden, werden oft als Faktoren oder Behandlungen bezeichnet.
- Wenn wir zum Beispiel eine Gruppe mit "No Treatment", eine andere mit "Treatment A" und eine dritte mit "Treatment B" vergleichen, dann führen wir eine Ein-Wege-ANOVA durch, wobei "Treatment" den einen Analysefaktor darstellt.

Varianz analyse (ANOVA)

- Da die Nullhypothese darin besteht, dass es keinen Unterschied zwischen den Gruppen gibt, basiert der Test auf einem Vergleich der beobachteten Variation zwischen den Gruppen (d.h. zwischen ihren Mittelwerten) und der erwarteten Variabilität innerhalb der Gruppen (d.h. zwischen den Probanden).
- Der Vergleich erfolgt in allgemeiner Form als F-Test zum Vergleich von Varianzen, aber für zwei Gruppen führt der t-Test zu genau dem gleichen Ergebnis.
- Die einseitige ANOVA geht davon aus, dass alle Proben aus normalverteilten Populationen mit gleicher Varianz entnommen werden.
- Die Annahme der gleichen Varianz kann mit dem **Leventest** überprüft werden.

Summe der Quadrate

- Die grundlegende Technik der ANOVA ist eine Aufteilung der Summe der Quadrate (SS) in 3 Komponenten, die sich auf die im Modell verwendeten Effekte beziehen:
  - eine Gesamtabweichung basierend auf allen Beobachtungsabweichungen vom großen Mittelwert (SS_{Total}),
  - eine Behandlungsabweichung ($SS_{Treatment}$) und
  - eine Fehlervarianz basierend auf allen Beobachtungsabweichungen von ihren entsprechenden Behandlungsmitteln (SS_{Error}).
- Die Behandlungsvarianz basiert auf den Abweichungen der Behandlungsmittel vom großen Mittelwert, wobei das Ergebnis mit der Anzahl der Beobachtungen in jeder Behandlung multipliziert wird
- Die drei Summen der Quadrate stehen folgendermaßen zueinander: $SS_{Total} = SS_{Error} + SS_{Treatment}$
- Wenn die Nullhypothese wahr ist, sind alle drei Varianzschätzungen gleich (innerhalb des Stichprobenfehlers).

Freiheitsgrade

- In Statistik hat eine Gruppe von n Werten n Freiheitsgrade (DF).
- Wenn wir nur die Form der Verteilung der Werte betrachten, können wir von jedem Wert den Mittelwert der Stichprobe abziehen. Dann haben die restlichen Daten nur n - 1 DF.
- Die Anzahl der Freiheitsgrade DF kann auf ähnliche Weise wie die Quadratsummen aufgeteilt werden: $DF_{Total} = DF_{Error} + DF_{Treatment}$
- Wenn wir z.B. 22 Patienten haben, die in 3 Gruppen unterteilt sind, werden die DFs bei der ANOVA wie folgt unterteilt:
  - 1 DF für den Gesamtmittelwert.
  - 2 DF für den Mittelwert jeder der drei Gruppen
  - 19 DF (= 22 - 1 - 2) bleiben für die verbleibenden Abweichungen von den Gruppenmitteln übrig.

Varianz analyse (ANOVA)

- Die Nullhypothese von ANOVAs ist, dass alle Gruppen aus der gleichen Population stammen.
- Unter der Nullhypothese, dass zwei normalverteilte Populationen gleiche Varianzen aufweisen, erwarten wir, dass das Verhältnis der beiden Stichprobenvarianten eine F-Verteilung aufweist.
- Der F-Wert ist der größere mittlere Quadratwert geteilt durch den kleineren Wert. (Wenn wir nur zwei Gruppen haben, ist der F-Wert das Quadrat des entsprechenden t-Wertes).
- Aus dem F-Wert können wir den entsprechenden p-Wert ablesen.

Varianz analyse (ANOVA)

- Dies lässt sich wie folgt in Python testen:

```
# Load stats library
import scipy.stats as stats

# Create data
tillamook = [0.0571,0.0813,0.0831,0.0976,0.0817]
newport = [0.0873,0.0662,0.0672,0.0819,0.0749,0.0649]
petersburg = [0.0974,0.1352,0.0817,0.1016,0.0968]
magadan = [0.1033,0.0915,0.0781,0.0685,0.0677, 0.0697]

# One-way ANOVA
stats.f_oneway(tillamook, newport, petersburg, magadan)
```

Varianz analyse (ANOVA)

- Eine detailliertere Form der ANOVA wird durch das statsmodels Paket ermöglicht:

```
# Import libraries
import pandas as pd
import seaborn as sns
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Load data
iris = sns.load_dataset("iris")

# Perform ANOVA
model = ols('sepal_length ~ C(species)', iris).fit()
anovaResults = anova_lm(model)
print(anovaResults)
```

Multiple Vergleiche

- Nullhypothese einer ANOVA = die Mittel aller Proben sind gleich sind
- Wenn also eine einseitige ANOVA ein signifikantes Ergebnis liefert, wissen wir nur, dass sie nicht identisch sind.
- Aber wir möchten auch wissen, für welche Probenpaare die Hypothese der gleichen Werte abgelehnt wird.
- In diesem Fall führen wir mehrere Tests gleichzeitig durch, einen Test für jedes Probenpaar (Post-Hoc-Analyse).
- Post-Hoc-Analyse = Untersuchung der Daten, nachdem das Experiment abgeschlossen ist, nach Mustern die nicht vorher spezifiziert wurden.
- Dies führt zu einem Mehrfachtest-Problem: Da wir mehrere Vergleichstests durchführen, sollten wir das Risiko, ein signifikantes Ergebnis zu erhalten, durch eine Korrektur der p-Werte kompensieren, auch wenn unsere Nullhypothese wahr ist.
- Wir haben dazu eine Reihe von Möglichkeiten: **Tukey HSD, Bonferroni-Korrektur, Holm-Korrektur, ...**

Tukey's Test

- Tukey's Test, manchmal auch als Tukey Honest Significant Difference Test (HSD)-Methode bezeichnet, ist ein Test, der spezifisch für den Vergleich aller Paare von k unabhängigen Proben ist.
- Tukey's HSD kontrolliert die Fehlerrate Typ I über mehrere Vergleiche hinweg und wird allgemein als akzeptable Technik angesehen.
- Kann als Post-Hoc-Analyse verwendet werden, um zu testen, zwischen welchen beiden Gruppenmitteln ein signifikanter Unterschied besteht:

```
# Load function
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Tukey HSD test
res2 = pairwise_tukeyhsd(iris['sepal_length'],
                         iris['species'])

print(res2)
```

Bonferroni- & Holm-Korrektur

- Wir können aber auch t-Tests für alle Paare durchführen, die p-Werte berechnen und für das Mehrfachtest-Problem korrigieren.
- **Bonferroni-Korrektur** = p-Wert / Anzahl der Tests
- **Holm-Korrektur** = Vergleicht nacheinander den niedrigsten p-Wert mit einer Typ-I-Fehlerrate, die nach jedem Test reduziert wird.

```
# Multi-comparison of groups
mod = MultiComparison(iris['sepal_length'], iris['species'])

# Bonferroni Correction
mod.allpairtest(stats.ttest_rel, method='b')rtp[0]

# Holm correction
mod.allpairtest(stats.ttest_rel, method='Holm')[0]
```

Kruskal-Wallis test

- Wenn wir drei oder mehr Gruppen miteinander vergleichen und Daten nicht normalverteilt sind, verwendent man den **Kruskal-Wallis-Test**.
- Wenn die Nullhypothese wahr ist, folgt die Teststatistik für den Kruskal-Wallis-Test der Chi-Quadrat-Verteilung.

```
# Load stats library
from scipy import stats

# Create dummy data
x = [1, 1, 1]
y = [2, 2, 2]
z = [2, 2]

# Perform Kruskal-Wallis test
stats.kruskal(x, y, z)
```

Two-Way ANOVA

- Zwei-Wege-ANOVAs haben statt einem Faktor zwei Faktoren.
- Wir können nicht nur sehen, ob jeder der Faktoren signifikant ist; wir können auch überprüfen, ob das Zusammenspiel der Faktoren einen signifikanten Einfluss auf die Verteilung der Daten hat.

```
import pandas as pd
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Load dataset
titanic = sns.load_dataset("titanic")

# ANOVA with interaction
formula = 'age ~ C(sex) + C(pclass) + C(pclass):C(sex)'
lm = ols(formula, titanic).fit()
anova_lm(lm)
```

Three-Way ANOVA

- Bei mehr als zwei Faktoren ist es empfehlenswert, die statistische Modellierung zu verwenden.
- Wie immer bei der Analyse von statistischen Daten sollte man zunächst die Daten visuell überprüfen.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
df = sns.load_dataset("exercise")

sns.factorplot("time", "pulse", hue="kind",
               col="diet", data=df,
               hue_order=["rest", "walking", "running"],
               palette="YlGnBu_d",
               aspect=.75).despine(left=True)
plt.show()
```

Tests an kategorischen Daten

- In einer Datenprobe wird die Anzahl der Daten, die in eine bestimmte Gruppe fallen, als Frequenz bezeichnet, so dass die Analyse kategorischer Daten die Analyse von Frequenzen ist.
- Beim Vergleich von zwei oder mehr Gruppen werden die Daten oft in Form einer Häufigkeitstabelle, manchmal auch Kontingenztabelle genannt, dargestellt.
- Für die Analyse von Häufigkeitstabellen gibt es eine Reihe von statistischen Tests: **Chi-Quadrat-Test, Fisher's Exact Test, McNemar's Test, Cochran's Q Test**

One-Way Chi-Square Test

- Chi-Quadrat-Test = Dies ist der am häufigsten verwendete Typ.
- Es handelt sich um einen Hypothesentest, der überprüft, ob die Einträge in den einzelnen Zellen einer Häufigkeitstabelle alle aus der gleichen Verteilung stammen.
- Mit anderen Worten, es überprüft die Nullhypothese H_0, dass die Ergebnisse unabhängig von der Zeile oder Spalte sind, in der sie erscheinen.
- Die alternative Hypothese H_a spezifiziert nicht die Art der Zuordnung, so dass eine hohe Aufmerksamkeit auf die Daten erforderlich ist, um die vom Test gelieferten Informationen zu interpretieren.

One-Way Chi-Square Test

- Nehmen wir an, dass du mit deinen Freunden wandern bist. Jeden Abend wird ein Los gezogen, wer Geschirr spülen muss. Am Ende der Reise scheinst du den größten Teil der Arbeit erledigt zu haben:

```
import pandas as pd
df = pd.DataFrame({"name":["You","Peter","Laura","Paul"],
                   "dishes":[10,7,6,5]})
```

- Wie wahrscheinlich es ist, dass diese Verteilung durch Zufall entstanden ist?
- Die erwartete Frequenz $D = n_{total}/n_{Personen} = 7$
- Die Wahrscheinlichkeit, dass diese Verteilung durch Zufall zustande kam, ist:

```
from scipy import stats
V, p = stats.chisquare(df["dishes"])
print(p) # 0.572406
```

Chi-Quadrat-Kontingenztest

- Wenn Daten in Zeilen und Spalten angeordnet werden können, können wir überprüfen, ob die Zahlen in den einzelnen Spalten vom Zeilenwert abhängig sind. Aus diesem Grund wird dieser Test auch als Kontingenztest bezeichnet.
- Der Chi-Quadrat-Kontingenztest basiert auf einer Teststatistik, die die Abweichung der beobachteten Daten von den zu erwarteten Werten misst, wenn laut Nullhypothese kein Zusammenhang bestehen würde.
- Der Chi-Quadrat-Test ist ein reiner Hypothesentest. Es sagt aus, ob die beobachtete Häufigkeit aus einer einzelnen Population durch Zufall zurückzuführen ist.
- Der Python-Befehl **stats.chi2_contingency** gibt die folgende Liste aus (Chi-Quadrat-Wert, p-Wert, Freiheitsgrad, erwartete Werte).

```
data = np.array([[43,9], [44,4]])
V, p, dof, expected = stats.chi2_contingency(data)
print(p) # 0.300384770391
```

Fisher's exact Test

- Während der Chi-Quadrat-Test ungefähr ist, ist der Fisher's Exact Test ein genauer Test.
- Er ist rechnerisch teurer und aufwendiger als der Chi-Quadrat-Test und wurde ursprünglich nur für kleine Stichprobenzahlen verwendet. Im Allgemeinen ist es jedoch mittlerweile der bessere Test.
- Die Implementierung in Python ist recht trivial:

```
data = np.array([[43,9], [44,4]])
oddsratio, p = stats.fisher_exact(obs)
print(p) # 0.23915695682
```

Auswahl des richtigen Tests für den Vergleich von Gruppen

No. of groups compared	Independent samples	Paired samples
Groups of nominal data		
1	One-sample t-test or Wilcoxon signed rank sum test	-
2 or more	Fisher's exact test or chi-square test	McNemar's test
Groups of ordinal data		
2	Mann–Whitney U test	Wilcoxon signed rank test
3 or more	Kruskal–Wallis test	Friedman test
Groups of continuous data		
2	Student's t-test or Mann–Whitney test	Paired t-test or Wilcoxon signed-rank sum test
3 or more	ANOVA or Kruskal–Wallis test	Repeated measures ANOVA or Friedman test

Hinweis: Tests zum Vergleich einer Gruppe mit einem festen Wert sind die gleichen wie für den Vergleich zweier Gruppen mit gepaarten Stichproben.

Übung

Übung

- Ladet den Datensatz **schoko.csv** in Python
- Prüft ob der Preis von Schokolade normalverteilt ist (visuell und statistisch)
- Prüft ob Bio-Schokolade signifikant teurer/billiger als normale Schokolade ist (visuell und statistisch)
- Prüft ob der Schokoladen-Preis von der Kategorie abhängig ist (visuell und statistisch)
- Erstellt eine Kontingenztabelle mit Bio & Fair und prüft ob es möglich ist, dass die Verteilung durch Zufall zustande kam
- Erstellt eine Kontingenztabelle mit Bio & Kategorie und prüft ob es möglich ist, dass die Verteilung durch Zufall zustande kam

Vielen Dank für eure Aufmerksamkeit!