DSF4-NB-1, Statistics with Python, 2020-06-11, S. Haug, University of Bern.

Statistics with Python

This one day course introduces basic statistical concepts used in Data Science with Python. It is more "how do I use this concept in Python" than "what is this concept". Some familiarity with statistical concepts are assumed.

Learning Objectives

Participants can/do

- understand probability and know typical distributions
- understand the meaning of common characteristics and uncertainties (standard deviations, sigmas, p-values, confidence levels, ...)
- apply linear regression for parameter estimation
- pply statistical tests
- Interpret and present results from statistical data analysis

Methods

The teaching languages are Python and English. The teaching methods include short theoretical introdcutions and example intersected with hands-on exercises.

Prerequisites

- Equivalent to Data Science Fundamentals 1-3
- Experience with Jupyter notebooks
- Working in Jupyter notebook environment
- Statistics fundamentals preferred
- Basic programming skills are necessary, we don't reserve time for basic programming concepts.

Python References

- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html)
- https://docs.scipy.org/doc/scipy/reference/ (https://docs.scipy.org/doc/scipy/reference/)
- https://matplotlib.org/users/index.html (https://matplotlib.org/users/index.html)

Schedule

09:00 - 11:30 Course
10:30 - 11:00 Break
11:00 - 12:30 Course
12:30 - 13:30 Lunch
13:30 - 15:00 Course
15:00 - 15:30 Break
15:30 - 17:00 Course

Content

0. Our Playground - the Iris Dataset and SciPy

When doing (Data) Science with Python, we use data and Python modules typically from the SciPy ecosystem (www.scipy.org). In this training we use the Iris dataset and the follwing SciPy modules.

- NumPy (for efficient/fast processing of large datasets)
- Pandas (for convenient data handling)
- SciPy (including the statistical module)
- Mathplotlib (and possibly Seaborn) for visualisation

Other packages like TensorFlow may not be part of SciPy, however, also much used.

Load the needed modules

In [5]:
```
# Load the needed python libraries by executing this python code (press
ctrl enter)
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
print('Congrats, you just loaded numpy, scipy.stats and mathplot.pyplot
loaded !')
```

```
Congrats, you just loaded numpy, scipy.stats and mathplot.pyplot loaded !
```

Load the Iris dataset (5 min)

```
In [7]: # Load the Iris dataset into a dataframe an study the dataframe content
        import pandas as pd
        url = 'https://www.openml.org/data/get_csv/61/dataset_61_iris.arff'
        dataframe = pd.read_csv(url) #,names=['slength','swidth','plength','pwid
        th','name'])
        dataframe.head()
        dataframe[48:52]
        dataframe[98:102]
```

Out[7]:

	sepallength	sepalwidth	petallength	petalwidth	class
98	5.1	2.5	3.0	1.1	Iris-versicolor
99	5.7	2.8	4.1	1.3	Iris-versicolor
100	6.3	3.3	6.0	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica

```
In [8]: dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepallength    150 non-null float64
sepalwidth     150 non-null float64
petallength    150 non-null float64
petalwidth     150 non-null float64
class          150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

What are the numbers in the table, which unit do they have?

1.0 Random variables and probability density functions (pdf)

In practice the measurement process, the data taking, is a **random, or stochastic, process**. The outcome varies from measurement to measurement. There are three (at least) reasons:

- Measurements are normally on a sample, not the full population. Samples fluctuate.
- Sensors have limited resolution, measurements on the same sample vary within the resolution
- According to quantum mechanics, i.e. at smallest distances, measurements are by nature stochastic

Our Iris dataset is a sample from 50 flowers in each class. So in each class there are 50 varying measurements for each of the four observables, sepal and petal length and width. This is due to the first and maybe the second reason (quantum mechanics can be neclected at scales larger than molecules). In descriptive statistics the observables are therefore called **random** variables. Let us call one x for examplification.

If x can take on any value from a continuous range, we write $f(x;\theta)dx$ as the probability that the measurement's outcome lies between x and $x + dx$. The function $f(x;\theta)$ is called the **probability density function (p.d.f.)**, which may depend on one or more parameters θ (for example the Iris class).

A random variable can be discrete or continuous. If discrete, then we use $f(x;\theta)$ to denote the probability to find the value x (in python the term probability mass fundtion, pmf, is then used). In the following the term p.d.f. is often taken to cover both the continuous and discrete cases, although technically the term density should only be used in the continuous case.

The p.d.f. is always normalized to **unity** (the number 1), i.e. the integral, i.e. the surface under the curve equals one. Both x and θ may have multiple components and are then often written as vectors. If θ is unknown, we may wish to estimate its value from a given set of measurements of x; this is a central topic of statistics (see next notebook on parameter estimation and regression).

The p.d.f. should be chosen to describe the fluctuation of the random variable in a best possible way. In other words, we should always choose an approprate p.d.f to describe our data. Some very useful and much used p.d.f. follow.

1.1 The normal pdf.

The normal (or Gaussian) probability density function is probably the most used one (informally the "bell curve"). It derives its importance in large part from the *central limit theorem*: "In most situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a "bell curve") even if the original variables themselves are not normally distributed." https://en.wikipedia.org/wiki/Central_limit_theorem (https://en.wikipedia.org/wiki/Central_limit_theorem)

Example: If one flips a coin many times the probability of getting a given number of heads in a series of flips will approach a normal curve, with mean equal to half the total number of flips in each series. (In the limit of an infinite number of flips, it will equal a normal curve.)

This means that in many or most cases it is sufficient to know the characteristics of the normal p.d.f. Others can be looked up if needed. Also often unspecified statements like the *error*, or better, the *uncertainty* refer to their meaning on the normal p.d.f.

As a formula the normal distribution function looks like (in one dimension)

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\exp(\frac{-(x-\mu)^2}{2\sigma^2})$$

It reads, given the distribution parameters mean μ and standard deviation σ, x follows this function.

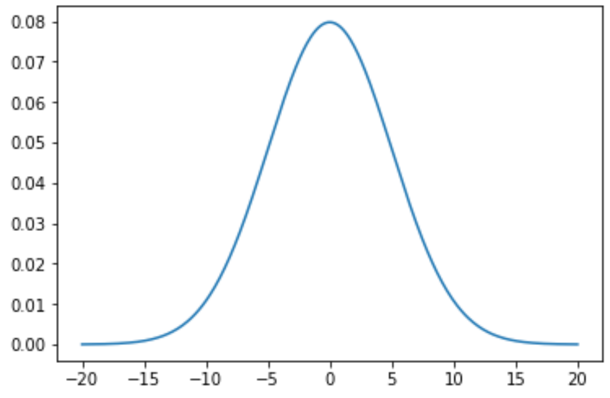
Exercise 1.1 Plot the "analytical" Normal p.d.f (5 min)

Plot the normal distribution with mean 0 and standard deviation 5 for 400 x values between -20 to 20. Repeat this for two other means and standard deviations. How big is the surface under the curves ?

(See also scipy.stat.norm https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.stats.norm.html (https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.stats.norm.html))

```
In [10]: scipy.stats.norm.pdf(-5.,0,5)
         scipy.stats.norm.cdf(0,0,5)
         # Part of the solution:
         x = np.linspace(-20,20,400) # 400 bins from -20 to 20
         plt.plot(x, scipy.stats.norm.pdf(x,0,5))
```

```
Out[10]: [<matplotlib.lines.Line2D at 0x7fcf4e96f2b0>]
```



1.2 The Poisson pdf

The Poisson distribution is popular for modelling the number of times an event occurs in an interval of time or space. For example:

- The number of meteorites greater than 1 meter diameter that strike Earth in a year
- The number of patients arriving in an emergency room between 10 and 11 pm

The probability mass function is

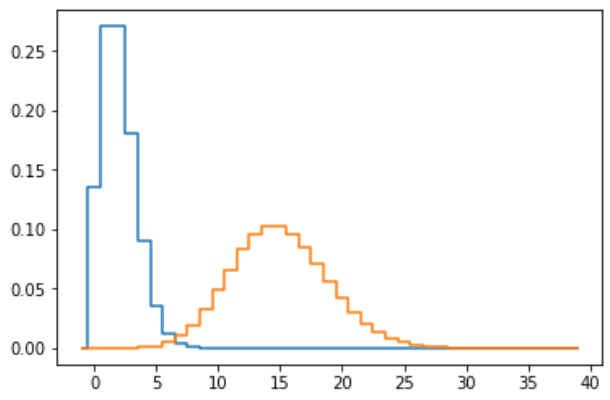
$$f(k;\lambda) = \frac{\lambda^k exp(-\lambda)}{k!}$$

For large k the normal distribution is an excellent approximation of the poisson p.d.f. For k below 20 one should be careful using statements based on the normal distribution.

The standard deviation of the Poisson pdf is simply $\sqrt{\lambda}$. This is very convenient, in particular for higher λ where the normal pdf is a good approximation and the probabilistic interpretation is easy (see below).

Plot the Poisson

```
In [11]: x = np.arange(-1, 40)
         plt.plot(x,scipy.stats.poisson.pmf(x,2),drawstyle='steps-mid')
         plt.plot(x,scipy.stats.poisson.pmf(x,15),drawstyle='steps-mid')

Out[11]: [<matplotlib.lines.Line2D at 0x7fcf5142dac8>]
```



1.3 The Binomial PDF

The binomial pdf models the probability of getting k positives when drawing n times, each time asking positive or false. For example,

$$f(k;n,p) = \binom{n}{k} \cdot p^k q^{n-k}$$

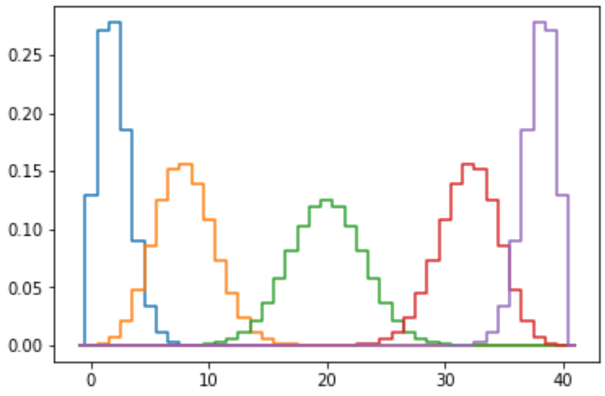
The binomial distribution converges towards the Poisson distribution as the number of trials goes to infinity while the product np remains fixed or at least p tends to zero. Therefore, the Poisson distribution with parameter λ = np can be used as an approximation to B(n, p) of the binomial distribution if n is sufficiently large and p is sufficiently small. According to two rules of thumb, this approximation is good if n ≥ 20 and p ≤ 0.05, or if n ≥ 100 and np ≤ 10.

For n>20 and p not too close to 1 or 0, the normal distribution is also here a good approximation.

Plot the binomial pdf for various parameters

```
In [13]: x = np.arange(-1,42)
         plt.plot(x,scipy.stats.binom.pmf(x,40,0.05),drawstyle='steps-mid')
         plt.plot(x,scipy.stats.binom.pmf(x,40,0.2),drawstyle='steps-mid')
         plt.plot(x,scipy.stats.binom.pmf(x,40,0.5),drawstyle='steps-mid')
         plt.plot(x,scipy.stats.binom.pmf(x,40,0.8),drawstyle='steps-mid')
         plt.plot(x,scipy.stats.binom.pmf(x,40,0.95),drawstyle='steps-mid')
```

```
Out[13]: [<matplotlib.lines.Line2D at 0x7fcf4e8b8940>]
```

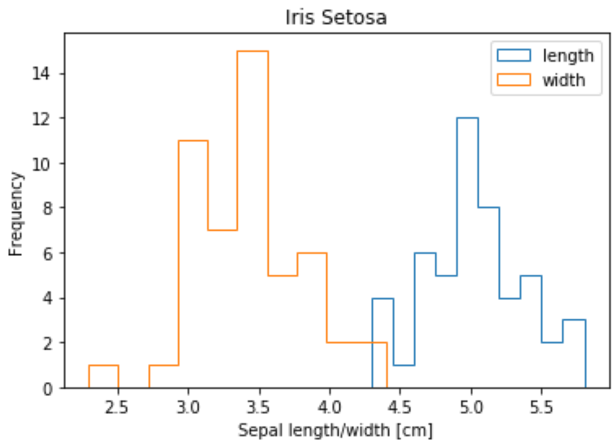


1.4 The Iris PDF

We can now plot the columns of the Iris dataset and ask ourself if the pdf is Normal, Poisson or Binomial, i.e. what is a good pdf model for our data?

```
In [14]: df_setosa = dataframe[dataframe['class']=='Iris-setosa']
         df_setosa['sepallength'].plot(kind="hist",fill=False,histtype='step',tit
         le='Iris Setosa', label="length")
         ax = df_setosa['sepalwidth'].plot(kind="hist",fill=False,histtype='step
         ', label="width")
         ax.set_xlabel('Sepal length/width [cm]')
         ax.set_ylabel('Frequency')
         plt.legend()
```

```
Out[14]: <matplotlib.legend.Legend at 0x7fcf4e890e80>
```



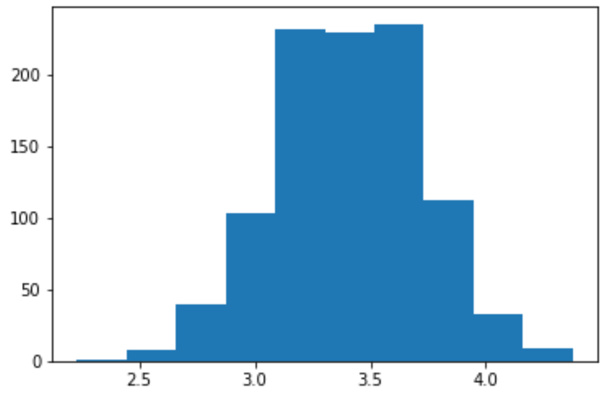
1.5 Generating PDF with Python

One can simulate data sets by generating them from probability density functions. The computer does this with a so called Monte Carlo (MC) algorithm. It draws x values (pseudo) randomly from the given distribution. The the actual draws of the random variable are called random variates. Simulations can be very useful when planning an experiment and developing the analysis method. Instead of real data one can use the simulated data.

Generate a normal dataset with 100 values, following a normal distribution.

```
In [15]: n = scipy.stats.norm.rvs(3.418,0.318,1000) # 1000 random values from a n
         ormal distribution with mean 3.418 and SD 0.318
         plt.hist(n)
```

```
Out[15]: (array([  1.,   8.,  39., 103., 231., 229., 235., 112.,  33.,   9.]),
          array([2.22992886, 2.44455027, 2.65917168, 2.87379309, 3.0884145 ,
                 3.30303591, 3.51765732, 3.73227873, 3.94690014, 4.16152155,
                 4.37614296]),
          <a list of 10 Patch objects>)
```



1.6 p-Value, Significance, Confidence Level and Interval

The p-value of a (measured) value is the surface (integral) under the pdf above that value, i.e. the probability of getting such a value or higher. Thus, the p-value is a measure for significance. Most implementations of statistical tests return the p-value and the value of the test statistic.

```
In [16]: ## The p-value of 3 on a normal pdf with mean of 2 and SD of 5
         print('p-value = %1.2f'%scipy.stats.norm.cdf(3,2,5))
```

```
p-value = 0.58
```

```
In [19]: ## The p-value of measuring a setosa slength longer than 5.2
         df_above=df_setosa[df_setosa['sepallength']>5.2]
         p = df_above['sepallength'].sum()/df_setosa.sum()[0]
         print('p-value = %1.2f'% p)
```

```
p-value = 0.24
```

Some times one refers to "double sided" p-values, i.e. what is the probability of obtaining a value outside $x \pm \Delta x$. If so, this needs to be taken into account when calculating the probability.

Which significance is considered as **significant** is a matter of convention. Some fields consider a p-value below 0.05 as significant. 0.05 is then the **significance level** α.

Very often only the number of standard deviations and not the actual p-value is reported. If so, one has to assume a normal pdf where standard deviations have commonly known p-values, i.e. probabilities.

Screenshot%202020-06-05%20at%2013.29.38.png

Example: People showing up in the emergancy room of a hospital on Fridays is reported as

$$N_{Friday} = 81 \pm \Delta 9$$

Our interpretation is then, assuming that 81 is the true mean, 31.8% of all Fridays will have more than 90 or less than 72 people showing up. If the actual pdf is not normal, this interpretation is wrong. However, maybe good enough.

In this case $\Delta 9$ is a **confidence interval** with a confidence level of 68.2%.

2.7 Uncertainties

All data have uncertainties. These should always be communicated when showing scientific numbers or plots. We distinguish between two types.

- Statistical uncertainties
  - Fluctuations, can be made smaller by taking more data, i.e. get more statistics
- Systematic uncertainties
  - Shift of data in one direction due to some "mistake" in the measurement, e.g. wrongly calibrated instrument showing all measured values systematically higher as they really are. Or for instance uncertainty due to the choice of methods and tools

The statistics tools can mostly handle the statistical uncertainties. There is no mathematical recipe for dealing with systematical uncertainties. You have to think through your experiment and try to estimate the influence of everything that can go wrong.

When uncertainties are stated on numbers or in graphs as error bars or error bands they generally show one standard deviation. If the data are well described by a normal p.d.f, the interpretation of one standard deviation is clear: if the measurement is repeated many times, 32% (or about 1/3 of the measurements) **should** be outside the error bars.

1.8 Recapitulation

- Collected data almost always contain random variables. Why?
- Random variables can be modelled with PDFs.
- There are many PDFs (infinite). What is the most important PDF?
- Can you give by heart p-values for different standard deviations on the normal PDF?

A table with some other common PDFs is attached to this notebook.

2.0 Describing the data with descriptive statistics

Statistics is a branch of mathematics dealing with the collection, analysis, interpretation, presentation, and organization of data. Two main statistical methods are used in data analysis: descriptive statistics, which summarize data from a sample using indexes such as the mean or standard deviation (see moments), and inferential statistics, which draw conclusions from data that are subject to random variation (e.g. observational errors, sampling variation).

2.1 Moments of the p.d.f. (mean, variance, standard deviation)

The n^{th} moment of a random variable x with p.d.f. $f(x)$ is

$$\alpha_n \equiv E[x^n] = \int_{-\infty}^{\infty} x^n f(x) dx$$

In the discrete case for $n = 1$ this integral becomes the sum known as the arithemtic mean:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The most commonly used moments are the **mean μ (or expectation value) and variance σ^2**:

$$\mu \equiv \alpha_1$$
$$\sigma^2 \equiv V[x] \equiv \int_{-\infty}^{\infty} (x-\mu)^2 f(x) dx = \ldots = \alpha_2 - \mu^2$$

The mean is the location or the “center of mass” of the p.d.f., and the variance is a measure of the square of its width. It is often convenient to use the **standard deviation (SD)** of x, σ, defined as the square root of the variance. In the discrete case the variance becomes

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

For the normal p.d.f the standard deviation is its width.

Based on higher moments other distribution descriptors are formed. **Skewness** and **Kurtosis** you may encounter. The skewness is a number indicating the deviation from a symmetric form. Kurtosis is a number indicating if the tails of the distribution is larger or smaller then the tails of the normal distribution.

```
In [20]: m,v,s,k = scipy.stats.norm.stats(10,5,moments='mvsk')
         print(m,v**0,5,s,k)
```

```
10.0 1.0 5 0.0 0.0
```

```
In [23]: df_setosa.describe()
         print('%1.2f %1.2f %1.2f %1.2f' % (df_setosa['sepallength'].mean(),df_se
         tosa['sepallength'].std(),df_setosa['sepallength'].skew(),df_setosa['sep
         allength'].kurt()))
```

```
5.01 0.35 0.12 -0.25
```

2.2 Quantiles, median and mode

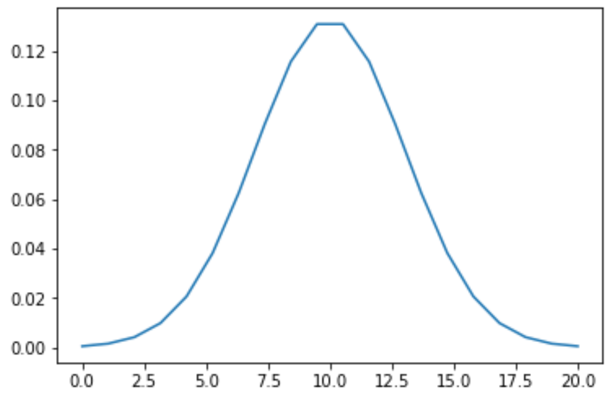
The **quantile** x_α is the value of the random variable x at which α% of the area is below x. An important special case is the **median,** $x_{med} \equiv x_{50}$. At the median half the area lies above and half lies below. For the normal p.d.f. the median equals the mean. The most probable value of a distribution is called **mode**.

Special quantiles are the quartiles and percentiles. The first quartile is the x_{25}, the second the x_{50} etc. Percentiles are for example x_{13} etc.

In [24]:
```
x = np.linspace(0,20,20)
print(scipy.stats.norm.ppf(0.5,10,3)) # Median = second quantile = x_{5
0} percentile
print(scipy.stats.norm.ppf(0.25,10,3)) #
plt.plot(x,scipy.stats.norm.pdf(x,10,3))
```

```
10.0
7.976530749411754
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x7fcf4e65c208>]
```



2.3 Standard Error

The standard error or standard error mean is defined:

$$SE = \frac{\sigma}{\sqrt{n}}$$

where σ is the sample mean and n is the sample size. If you report SE and not SD, be sure you know the difference and be clear on what you report.

In [25]:
```
print('Setosa sepal length mean %1.2f +- %1.2f'  % (df_setosa['sepalleng
th'].mean(), df_setosa['sepallength'].std()))
print('Setosa sepal length mean %1.2f +- %1.2f'  % (df_setosa['sepalleng
th'].mean(), df_setosa['sepallength'].sem()))
print(df_setosa['sepallength'].std()/df_setosa['sepallength'].size**0.5)
```

```
Setosa sepal length mean 5.01 +- 0.35
Setosa sepal length mean 5.01 +- 0.05
0.049849569625391305
```

Get descriptive statistics from the normal pdf and plot (5 min)

https://docs.scipy.org/doc/scipy/reference/stats.html (https://docs.scipy.org/doc/scipy/reference/stats.html)

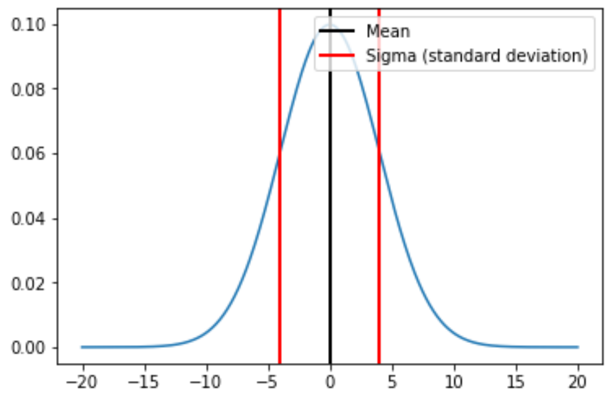
Get some desciptive statistics from a normal (continous) p.d.f. with mean 0 and SD 4.

In [26]:
```
mean, variance, skewness, kurtosis = scipy.stats.norm.stats(0,4,moments=
'mvsk')
print(mean, variance, skewness, kurtosis)
```

```
0.0 16.0 0.0 0.0
```

Plot the p.d.f and some moments

```
In [27]: x = np.linspace(-20,20,400)
         sigma=variance**0.5
         plt.plot(x,scipy.stats.norm.pdf(x,mean,sigma))
         plt.axvline(x=mean, linewidth=2, color = 'k',label="Mean") # Plot the me
         an as a vertical line
         plt.axvline(x=mean-sigma, linewidth=2, color = 'r', label="Sigma (standa
         rd deviation)")
         plt.axvline(x=mean+sigma, linewidth=2, color = 'r')
         plt.legend(loc='upper right')

Out[27]: <matplotlib.legend.Legend at 0x7fcf4e616cc0>
```



Get descriptive statistics from the Iris dataset and plot (5 min)

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html)

```
In [28]: df_setosa.describe()

Out[28]:
```

	sepallength	sepalwidth	petallength	petalwidth
count	50.00000	50.000000	50.000000	50.00000
mean	5.00600	3.418000	1.464000	0.24400
std	0.35249	0.381024	0.173511	0.10721
min	4.30000	2.300000	1.000000	0.10000
25%	4.80000	3.125000	1.400000	0.20000
50%	5.00000	3.400000	1.500000	0.20000
75%	5.20000	3.675000	1.575000	0.30000
max	5.80000	4.400000	1.900000	0.60000

```
In [29]: df_setosa.sem()

Out[29]: sepallength    0.049850
         sepalwidth     0.053885
         petallength    0.024538
         petalwidth     0.015162
         dtype: float64
```

```
In [30]: 0.35249/(50)**0.5
         df_setosa.skew()
```

```
Out[30]: sepallength    0.120087
         sepalwidth     0.107053
         petallength    0.071846
         petalwidth     1.197243
         dtype: float64
```

2.4 Descriptive Graphs

Python offers many ways for visually describing data. We'll look at some common ones.

Histograms and binning

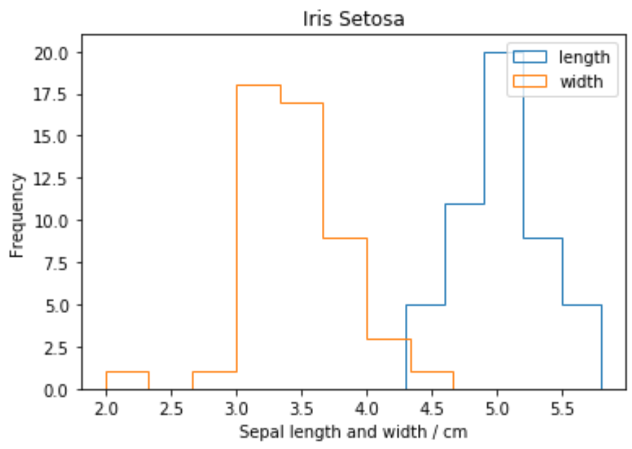
Histograms are the common way to visually display the pdf of a data column. It consists of bins on the x axis containing the frequency/number of entries in each bin. The plotted result depends on the binning choice. It is important to be able to judge

- overbinning
- underbinning

Let's plot Iris Setosa length with various binnings.

```
In [31]: df_setosa = dataframe[dataframe['class']=='Iris-setosa']
         df_setosa['sepallength'].plot(kind="hist",bins=5,fill=False,histtype='st
         ep',title='Iris Setosa', label="length")
         # Test various number of bins in the next line
         mybins=np.linspace(2,5,10)
         ax = df_setosa['sepalwidth'].plot.hist(bins=mybins,fill=False,histtype='
         step', label="width")
         ax.set_xlabel('Sepal length and width / cm')
         ax.set_ylabel('Frequency')
         plt.legend()
```

```
Out[31]: <matplotlib.legend.Legend at 0x7fcf4e5c2780>
```

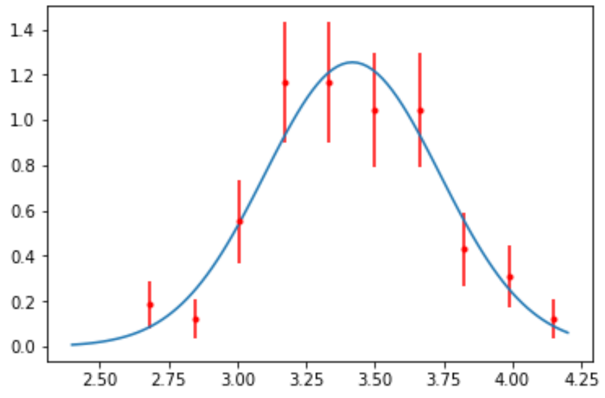


How do you choose the right binning?

Histograms with error/uncertainty bars

Here an example how to plot histograms with uncertainty bars

In [32]:
```
n = scipy.stats.norm.rvs(3.418,0.318,100)
## Draw a histogram which is not normalised
entries1, edges, patches = plt.hist(n, bins=10, histtype='step')
## Draw a histogram which IS normed
entries2, edges, patches = plt.hist(n, bins=10, histtype='step',density=
True)
## Close plt so that the previous histograms are not shown
plt.close()
## Calculate the poisson standard deviation and scale down to second his
togram
errors = np.sqrt(entries1) * entries2/entries1
## calculate bin centers
bin_centers = 0.5 * (edges[:-1] + edges[1:])
## draw errobars, use the sqrt error.
plt.errorbar(bin_centers, entries2, yerr=errors, fmt='r.')
## Draw a normal distribution
x = np.linspace(2.4,4.2,100)
plt.plot(x,scipy.stats.norm.pdf(x,3.418,0.318))
plt.show()
```

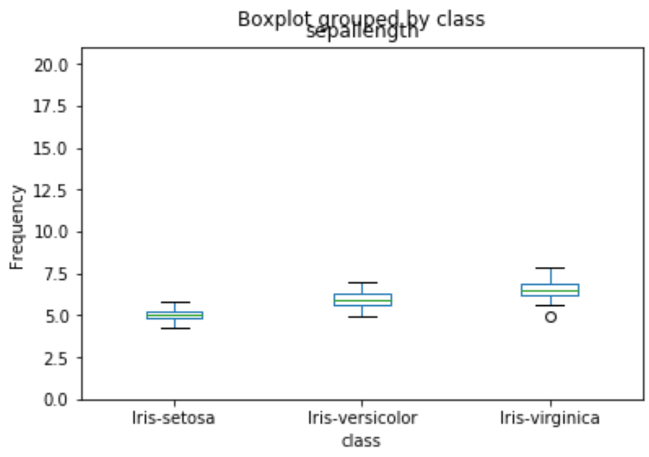


We see that 3 out of 10 data points are more than one standard deviation off the "theory" curve. This is how it should be.

Box Plots

Often box plots are used to visually inspect differences between categories. Let's display the setal length of the three Iris categories with bx plots (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html)):

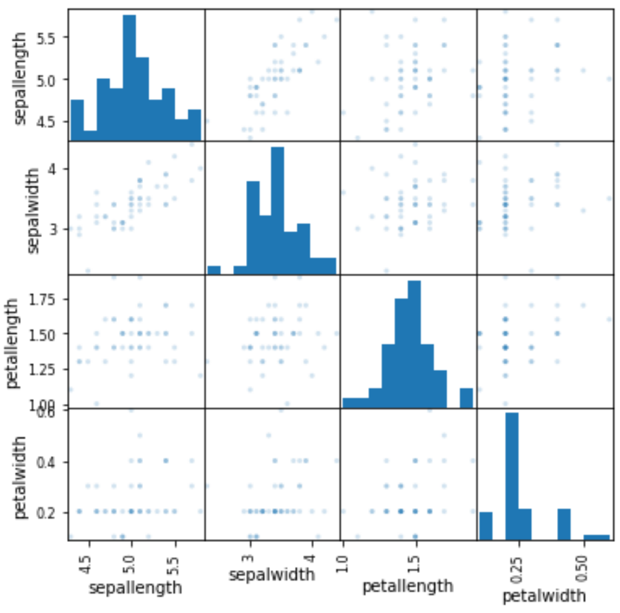
```
In [34]: dataframe.boxplot(column=['sepallength'],by='class')
         df_setosa['sepallength'].plot(kind="hist",bins=5,fill=False)

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcf4e77d4e0>
```



Scatter plots (correlations)

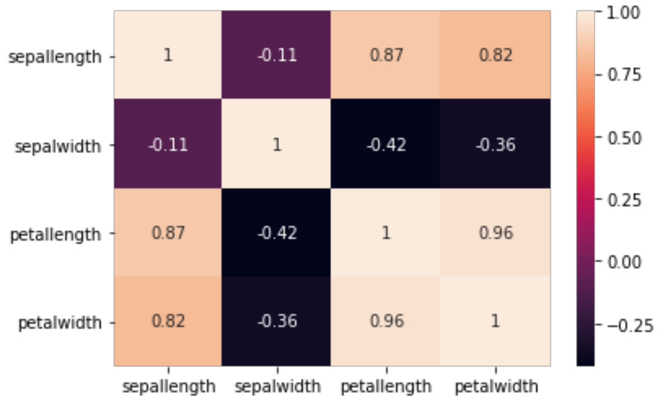
A visual inspection of correlations between columns is often useful.

```
In [35]: ## https://pandas.pydata.org/pandas-docs/stable/generated/pandas.plottin
         g.scatter_matrix.html
         from pandas.plotting import scatter_matrix
         df = dataframe
         scatter_matrix(df[df['class']=='Iris-setosa'], alpha=0.2, figsize=(6,
         6), diagonal='hist')
         plt.show()
```



```
In [36]: import seaborn as sn
         sn.heatmap(df.corr(), annot=True)
         plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/pandas/core/computation/check.py:1
9: UserWarning: The installed version of numexpr 2.4.3 is not supported i
n pandas and will be not be used
The minimum supported version is 2.6.1

  ver=ver, min_ver=_MIN_NUMEXPR_VERSION), UserWarning)
```



	sepallength	sepalwidth	petallength	petalwidth
sepallength	1	-0.11	0.87	0.82
sepalwidth	-0.11	1	-0.42	-0.36
petallength	0.87	-0.42	1	0.96
petalwidth	0.82	-0.36	0.96	1

Annex 1 Probability

An abstract definition of probability can be given by considering a set S, called the sample space, and possible subsets $A, B, \ldots$ the interpretation of which is left open. The probability P is a real-valued function defined by the following axioms due to Kolmogorov (1933) [9]:

- For every subset A in S, $P(A) \geq 0$;
- For disjoint subsets (i.e., $A \cap B = \emptyset$), $P(A \cup B) = P(A) + P(B)$;
- $P(S) = 1$.

From this further properties can be derives, e.g.

- $P(\bar{A}) = 1 - P(A)$
- $P(A \cup \bar{A}) = 1$
- $P(\emptyset) = 0$
- if A in B, then $P(A) \leq P(B)$
- $P(A \cup \bar{A}) = P(A) + P(B) - P(A \cap B)$

Conditional probability

In addition, one defines the conditional probability $P(A|B)$ (read as P of A given B) as

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

As an example, when throwing the dice, consider obtaining more than 3 eyes given only trows with even number of eyes outcomes. We calculate the (conditional) probability:

$$P(n > 3 | n\ even) = \frac{P(n > 3 \cap n\ even)}{P(even)} = \frac{2/6}{3/6} = \frac{2}{3}$$

Independence

If A and B are independent, then

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

Some words on probability and bayesian versus frequentist statistics

Since data taking is data taking of random variables, we need to define and talk about probability. In mathematichs probability is defined in a rather abstract manner (see Annex below). For our purposes we go directly to the interpreation as either **relative frequency** or **subjective probability**. If A is a possible outcome of an experiment repeated n times, then the probability of A is the realtive frequency

$$P(A) = \lim_{n\to\infty} \frac{times\ outcome\ is\ A}{n}$$

The subjective probability is

$$P(A) = degree\ of\ belief\ that\ A\ is\ true$$

Both concepts are consistent with the abstract mathematical definition .

Bayes' theorem

From this definition and using the fact that $A \cap B$ and $B \cap A$ (intersection) are the same, one obtains Bayes' theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

first published by the Reverend Thomas Bayes (1702-1761). Statistics based on the relative frequency interpretation of probability is called frequentist statistics, on bayesian theorem bayesian statistics.

Other probability density functions (5 min)

This is just for your reference, no need to understand all these distributions. A quite good free and comprehensive book is here: http://staff.fysik.su.se/~walck/suf9601.pdf (http://staff.fysik.su.se/~walck/suf9601.pdf).

Table 1.4.1 Some common probability density functions, with corresponding characteristic functions and means and variances. In the Table, $\Gamma(k)$ is the gamma function, equal to $(k-1)!$ when k is an integer; ${}_1F_1$ is the confluent hypergeometric function of the 1st kind [11].

download.png

Distributions are either discrete or continuous. To study the distributions we use the statistical functions of scipy.stats (https://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html (https://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html)). There you also find more distributions than listed here.

DSF4-NB-2, Statistics with Python, 2020-06-11, S. Haug, University of Bern.

Parameter estimation / regression

Average expected study time : 3x45 min (depending on your background)

Learning outcomes :

- Know what is meant with parameter estimation and regression
- Perform linear regression with Python by example
- Perform non-linear regression with Python by example
- Know what non-parametric regression is

Main python module used

- the Scipy.stat module https://docs.scipy.org/doc/scipy/reference/stats.html (https://docs.scipy.org/doc/scipy/reference/stats.html)

Content :

What you should do for your uncertainties

When you have a data analysis project, you need to define the final numbers and plots you want to produce. In order to control your uncertaines, you should maintain a list/table with the largest uncertainties and their effect on the final number(s) as percentages.

Just a nice table

As a data scientist you should roughly know what 1, 2, 3 standard deviations ("sigmas") means in terms of probability (or area in the normal distribution).

Screen%20Shot%202018-05-30%20at%2015.52.21.png

3.0 Regression - Situation

We have data and want to extract model paramters from that data. An example would be to estimate the mean and the standard deviation, assuming a normal distribution. Another one would be to fit a straight line. For historical reasons this kind of analysis is often called regression. Some scientists just say fitting (model parameters to the data).

We distinguish between parametric and non-parametric models. A line and the normal distribution are both parametric.

3.1 About linear Regression

Linear regression means fitting linear parameters to a set of data points (x,y). x and y may be vectors. You may consider this as the simplest case of Machine Learning (see Module 3). Example, a line is described by

$$y = ax + b$$

Thus two parameters a (slope) and b (intersection with y axis) can be fitted to (x,y).

There are different fitting methods, mostly least squares or maximum likelihood are used.

Linear regression in Python

Import the Python libraries we need.

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import scipy.stats as stats
```

Read the data from file and do a linear regression for a line in the plength-pwidth space of the setosa sample. We use https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html), using least squares.

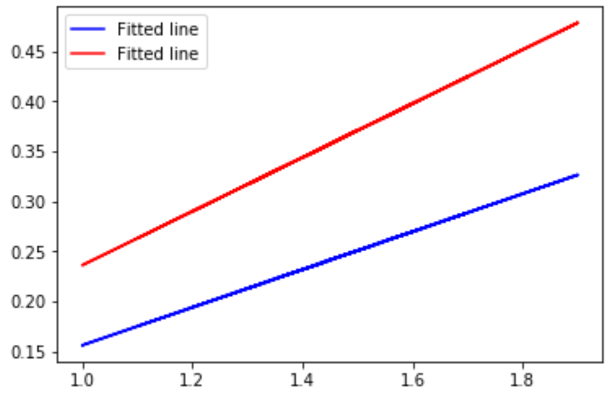
```
In [3]: url = 'https://www.openml.org/data/get_csv/61/dataset_61_iris.arff'
        df = pd.read_csv(url)
        #df_set = df[df['species']=='Iris-versicolor']
        df_set = df[df['class']=='Iris-setosa']
        plengths = df_set['petallength']
        pwidths  = df_set['petalwidth']
        slope, intercept, r_value, p_value, std_err = stats.linregress(plengths,
        pwidths)
        print (slope, intercept, std_err)

        0.18926247288503262 -0.03308026030368777 0.08489680724058374
```

The number of digits is ridiculous. Let's print it better.

```
In [4]: print ('Gradient = %1.2f +- %1.2f' % (slope,std_err))

        Gradient = 0.19 +- 0.08
```

let's look at the scatter plot to see if this makes sense.

In [6]:
```
#ax = df_set.plot(x='petallength',y='petalwidth',kind="scatter",c='c')
plt.plot(plengths, intercept + slope*plengths, 'b', label='Fitted line')
plt.plot(plengths, intercept + (slope+0.08)*plengths, 'r', label='Fitted
line')
plt.legend()
plt.show()
```



By eye it is hard to say how good this fit is. Try the same regression with versicolor. The result may be a bit clearer.

We now have a model, a straight line, whose shape we have chosen, but whose parameters (slope and intersection) have been estimated/fitted from data with the least squares method. It tells us that pwidth of a leaf is plength x slope (f(plength) = a x slope). So we can do interpolation and extrapolation, i.e. get the pwidth at any plength.

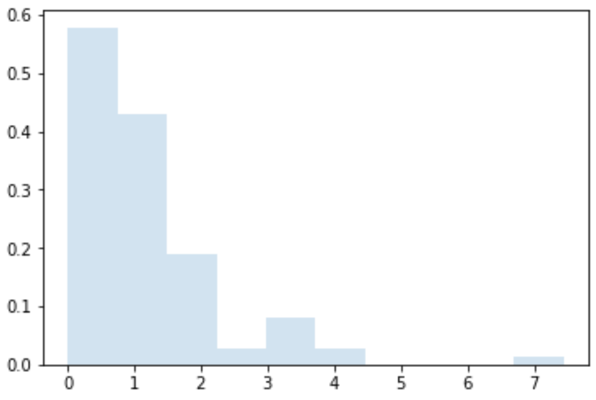
Example Exponential p.d.f.

With scale β and location μ

$$f(x) = \frac{1}{\beta}e^{-(x-\mu)/\beta}, x \geq \mu; \beta > 0$$

In [7]:
```
## Let us fit data to an exponential distribution
fig, ax = plt.subplots(1, 1)
## First generate a data set from an exponential distribution
x = stats.expon.rvs(size=100) #  scale = 0.0, location = 0.00, 1000 vari
ates
ax.hist(x, density=True, histtype='stepfilled', alpha=0.2)
## Fit scale and location to the histogram/data
loc, scale = stats.expon.fit(x) # ML estimator scale, lambda * exp(-lamb
da * x), scale =1/lambda
print(' Location = %1.2f , Scale = %1.2f' % (loc,scale))
plt.show()
```

```
 Location = 0.01 , Scale = 1.15
```



This fit method is poor in the sense that it doesn't return uncertainties on the fitted values. This we normally want to know. The curve_fit method below also returns the uncertainties.

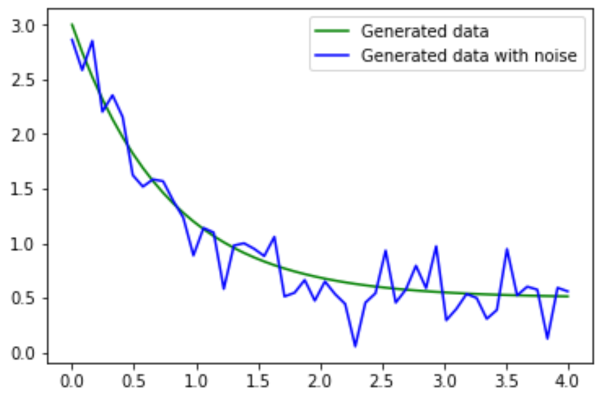
3.2 Non-linear regression

If a line is not streight it is curved. There are many mathematical functions whose parameters we can try to fit to experimental data points. Some examples: Polynominals (first order is linear regression, second order is a parabola etc), exponential functions, normal function, sindoial wave function etc. You need to choose an approriate shape/function to obtain a good result.

With the Scipy.stat module we can look for preprogrammed functions (in principle you can program your own function whose parameters you want to fit too): https://docs.scipy.org/doc/scipy/reference/stats.html (https://docs.scipy.org/doc/scipy/reference/stats.html).

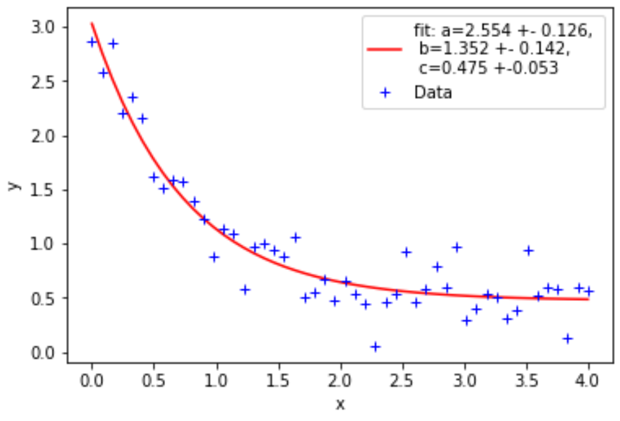
The scipy.optimize module provides a more general non-linear least squares fit. Look at and play with this example. It is complex and you will probably need some time testing, googling etc.

In [8]:
```
from scipy.optimize import curve_fit

def func(x, a, b, c):
     return a * np.exp(-b * x) + c

xdata = np.linspace(0, 4, 50) #
y = func(xdata, 2.5, 1.3, 0.5)
plt.plot(xdata, y, 'g-', label='Generated data')
np.random.seed(1729)
y_noise = 0.2 * np.random.normal(size=xdata.size)
ydata = y + y_noise
plt.plot(xdata, ydata, 'b-', label='Generated data with noise')
plt.legend()
plt.show()
```



In [9]:
```
popt, pcov = curve_fit(func, xdata, ydata)
print(popt)
perr = np.sqrt(np.diag(pcov)) # Standard deviation = square root of the
variance being on the diagonal of the covariance matrix
plt.plot(xdata, func(xdata, *popt), 'r-',label= \
         'fit: a=%5.3f +- %5.3f, \n b=%5.3f +- %5.3f, \n c=%5.3f +-%5.3f
' % \
         (popt[0],perr[0],popt[1],perr[1],popt[2],perr[2]))
plt.xlabel('x')
plt.ylabel('y')
plt.plot(xdata, ydata, 'b+', label='Data')
plt.legend()
plt.show()
perr = np.sqrt(np.diag(pcov)) # Standard deviation = square root of the
variance being on the diagonal of the covariance matrix
perr
```

```
[2.55423706 1.35190947 0.47450618]
```



Out[9]:
```
array([0.12605755, 0.14212384, 0.05315968])
```

3.3 Non-parametric regression

So far we have used functions (models) with some predefined shape/form. The parameters we fitted to data. If we have no clue about the form, we may try to fit with non-parametric methods. However, these require more data as also the shape needs to guessed or fitted from the data. So normally a non-parametric method gives poorer results.

There are several ways to do this in Python. You make look at this if you are interested:

https://pythonhosted.org/PyQt-Fit/NonParam_tut.html (https://pythonhosted.org/PyQt-Fit/NonParam_tut.html)

DSF4-NB-3, Statistics with Python, 2020-06-11, S. Haug, University of Bern.

4. Hypothesis Testing

Practice various tests with the Python scipy stats module.

Average expected study time : 3x45 min (depending on your background)

Learning outcomes :

- Know the hypothesis test vocabulary
- Can calculate p-values on different p.d.f's
- Can perform normality tests of samples
  - D'Agostino-Pearson test (Shapiro-Wilk, Kolmogorov-Smirnow)
- Can perform t-Test on a single sample (testing against a given mean, gaussian)
- Can perform a Wilcoxon on a single sample (testing against a given median, not gaussian, only symmetric assumption)
- Can comparing two groups of data, e.g. treated and control, know which tests are suited for which cases
- Can perform some ANOVA tests

Bonus skills

- More Python tricks
- Get to know the Python stats module better: https://docs.scipy.org/doc/scipy/reference/stats.html (https://docs.scipy.org/doc/scipy/reference/stats.html)

Literature examples

- https://en.wikipedia.org/wiki/Statistical_hypothesis_testing (https://en.wikipedia.org/wiki/Statistical_hypothesis_testing)
- http://greenteapress.com/wp/think-stats-2e/ (http://greenteapress.com/wp/think-stats-2e/) (online book covering most of the CAS Module 2)
- For life sciences (bioinformatics) for example https://cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf (https://cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf)
- (Particle) Physics http://pdg.lbl.gov/2018/reviews/rpp2018-rev-statistics.pdf (http://pdg.lbl.gov/2018/reviews/rpp2018-rev-statistics.pdf)
- Finance, Insurance: For instance Erik Boelvik, Computation and Modelling in Insurance and Finance

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
```

3.1 Vocabulary and an example

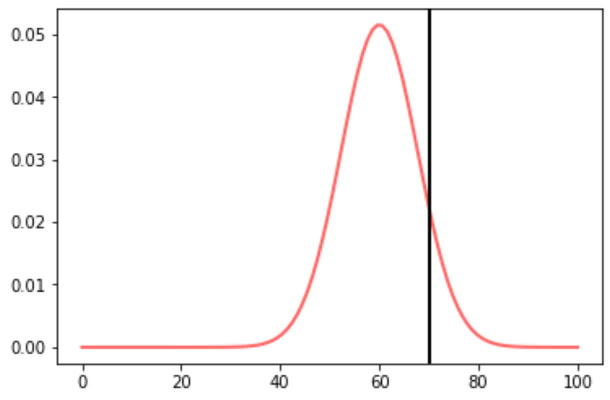
The hypothesis being tested is called the Null hypothesis. When doing a statistical test, the test statistic of a sample is calculated and the p-value of this, under the null hypothesis assumption is given.

This p-value is the probability of obtaining such a sample if the null hypothesis is true. A significance level of 5% is often used as the threshold where you say, this is so unlikely that I consider the null hypothesis as wrong. However, 5% is not very significant (5 times of 100 this should actually happen).

Example (not real)

- Based on samples from previous years, the female/(male+female) ratio in a class is 0.6 (60%). This is a mean calculated counting heads in classes of 100 people. On average we counted 60 female students. So we can safely assume that the ratio is normally distributed with a standard deviation equal to the square root of 60.
- Our (a bit stupid null hypotesis) is that the ratio stays constant for the next 10 years.
- This year we count 70 females in a class of 100. What is the p-value ?

In [2]:
```
def get_pvalue_on_gauss(data,mu,nbins):
  sigma = np.sqrt(mu) # Standard deviation
  #print('The standard deviation is',sigma)
  # Plot the null hypothesis
  x = np.linspace(0,nbins, nbins) # 100 bins
  plt.plot(x, stats.norm.pdf(x,mu,sigma),'r-', lw=2, alpha=0.6, label='n
orm pdf')
  plt.axvline(x=data, linewidth=2, color = 'k')
  plt.show()
  p_value = 1 - stats.norm.cdf(data,mu,sigma) # cdf = cumulated density
function, see below
  #print('The p-value = %1.3f'% p_value)
  return p_value

p_value = get_pvalue_on_gauss(70,60,100)
print('p-Value is %1.2f'%p_value)
```



```
p-Value is 0.10
```

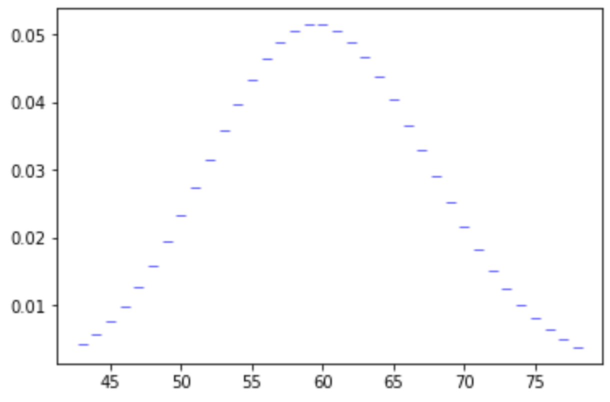
Do you still think our null hypothesis is valid ?

1 - CDF is the p-value. If you want to know more about CDF, read this: https://en.wikipedia.org/wiki/Cumulative_distribution_function (https://en.wikipedia.org/wiki/Cumulative_distribution_function)

The same calculation with a Poisson distribution

```
In [3]: def get_pvalue_on_poisson(data,mu,nbins):
           p_value = 1 - stats.poisson.cdf(data,mu)
           #print('The p-value = %1.3f'% p_value)
           xp = np.arange(stats.poisson.ppf(0.01, mu), stats.poisson.ppf(0.99, m
        u))
           plt.plot(xp, stats.poisson.pmf(xp,mu),'b_', lw=2, alpha=0.6, label='p
        oisson pdf')
           #plt.show()
           return p_value

        p_value = get_pvalue_on_poisson(70,60,100)
        print('p-Value %1.2f'%p_value)
```

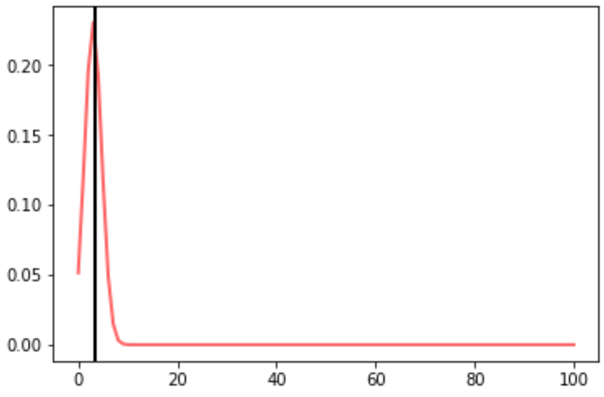
```
p-Value 0.09
```



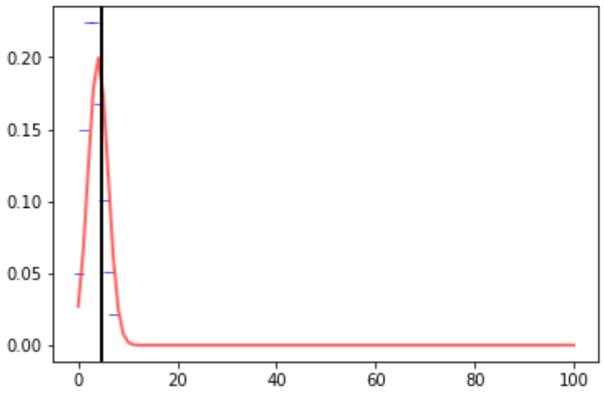
p-value differences between Normal and Poisson

How much does the p-value on the normal distribution (approximation) from the one on the Poisson distribution (supposed to be the correct):

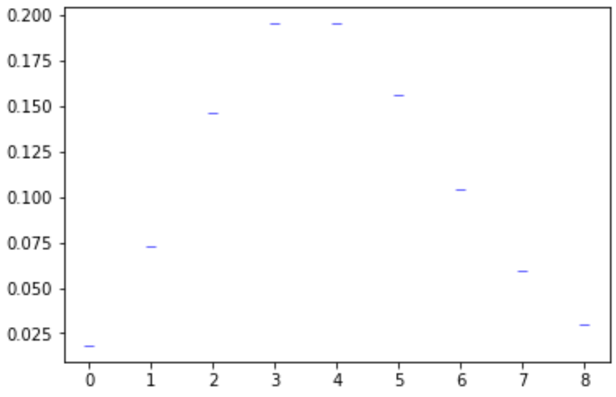
In [5]:
```
data_scale=70/60.
for mu in range(3,5):
   pv_gauss=get_pvalue_on_gauss(data_scale*mu,mu,100)
   pv_poisson=get_pvalue_on_poisson(data_scale*mu,mu,100)
   print('mu = %1.2f %1.2f %1.2f' % (mu,pv_gauss,pv_poisson))
```



```
mu = 3.00 0.39 0.35
```



```
mu = 4.00 0.37 0.37
```



So we see that in our case (mu = 60), the p-value on the normal distribution is only one percent higher than on the poisson distribution. Conclusion: We can safely use the normal distribution as an approximation.

Error types and statistical power

- Type I : Reject the null hypothesis due to a fluctuation (false positive)
- Type II : Keep the null hypothesis by interpreting a real effect as a fluctuation (false negative)
- (Type 0 : Make the right conclusion but asking the wrong question)

People may talk about the **statistical power** of their experiment. This is a number between 0 and 1 telling how often your experiment would lead to a type II error. So a high power (close to 1) means rare type II errors. The statistical power is infuenced by your significance criterion, the (physical) magnitude of your effect, the sample size ("statistics").

Many tests

There are many tests on the market and some or most of them are implemented in Python (or R). We don't look at them all. We just practise on a few useful cases. You would look up suiteable tests for your analysis. The important thing is to know what a test assumes (distribution etc) and how to interprete the result and if you can trust the result.

4.2 Normality tests of single samples

Normality tests can be performed for each column of data. Each normality test reports a p-value that answers this question:

If you randomly sample from a Gaussian population, what is the probability of obtaining a sample that deviates from a Gaussian distribution as much (or more so) as this sample does?

Normality tests are not good for small samples (<10-20 values). You can do it, but the interpration of the output is not straight forward.

There are several normality tests in the Python stats module:

- D’Agostino-Pearson
- Shapiro-Wilk
- Kolomogorv-Smirnov

The first one is considered the best one, so use that.

https://docs.scipy.org/doc/scipy/reference/stats.html (https://docs.scipy.org/doc/scipy/reference/stats.html)

Let us look at the p-values of some normality tests with generated data.

In [6]:
```
pts = 1000
np.random.seed(28041990) # By fixing the seed, you can make the generati
on reproducible
# Generate two data samples a and b, each with 1000 entries
a = np.random.normal(0, 1, size=pts) # mean = 0, width = 1 (std deviatio
n)
b = np.random.normal(2, 1, size=pts)
#x = np.concatenate((a, b))
x = a
k2, p = stats.normaltest(x) # D Agostino-Pearson. The method returns the
test statistic value and the p-value
alpha = 0.001 # Rejection criterion defined by you
print('Alpha = ',alpha)
print('p = ',p)
if p < alpha:  # null hypothesis: x comes from a normal distribution
    print("The null hypothesis can be rejected")
else:
 print("The null hypothesis cannot be rejected")
```

```
Alpha =  0.001
p =  0.12660904659553507
The null hypothesis cannot be rejected
```

This is not very surprising as we generated the data from a normal distribution.

Load the iris sepal length of species and get some descriptive values.

In [7]:
```
import pandas as pd
url='https://www.openml.org/data/get_csv/61/dataset_61_iris.arff'
dataframe = pd.read_csv(url)
df_setosa=dataframe[dataframe['class'] == 'Iris-setosa']
dfs = df_setosa['sepalwidth']
print(stats.describe(dfs)) # Print some descriptive statistics
dfs.describe() # the pandas describe prints similar statistics
```

```
DescribeResult(nobs=50, minmax=(2.3, 4.4), mean=3.418, variance=0.1451795
918367347, skewness=0.1038140820747848, kurtosis=0.6851340609499261)
```

Out[7]:
```
count    50.000000
mean      3.418000
std       0.381024
min       2.300000
25%       3.125000
50%       3.400000
75%       3.675000
max       4.400000
Name: sepalwidth, dtype: float64
```

Check the normality of the iris sepal length of species

In [8]:
```
k2, p = stats.normaltest(dfs) # D Agostino-Pearson test
print("p-value = %1.2f" % p)
```

```
p-value = 0.39
```

What does this p-value mean? If you want to know what the k2 is, look it up in the documentation.

We can also try the shapiro test that the stat module offers:

```
In [9]:  s,p = stats.shapiro(dfs)
         print('Test statistic = ',s,' p-value =',p)

         Test statistic =  0.968691885471344  p-value = 0.20465604960918427
```

In this case the shapiro test returns a much lower p-value. What can you do about this difference?

t-Test (with a given mean)

With the t-test you can test your data against a normal distribution with a given mean:

If the data were sampled from a Gaussian population with a mean equal to the hypothetical value you entered, what is the chance of randomly selecting N data points and finding a mean as far (or further) from the hypothetical value as observed here?

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html#scipy.stats.normaltest (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html#scipy.stats.normaltest)

```
In [11]:  print('The sample mean of the sepal length = ',dfs.mean())
          stats.ttest_1samp(dfs,3.3)

          The sample mean of the sepal length =  3.418

Out[11]:  Ttest_1sampResult(statistic=2.1898492754769694, pvalue=0.0333256445571966
          44)
```

Interpretation: If the real average of the sepal length of setosa leaves is 3.3 cm and if the length of the leaves is normally distributed, the probability of sampling this data or worse data from setosa leaves is 3.3%.

Our null hypothesis in this case was a normal distribution with mean equal to 3.3 cm.

4.3 Non-parametric Wilcoxon signed rank test

This test answers the same question as the t-test above, but is not testing against a normal distribution. If you like, you can make an example yourself.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon)

4.4 Tests for comparing two samples

In many experiments one uses two groups, a treated group and a control group which is not treated. The goal is then to ckeck the effect of the treatment.

One way to answer this, is to check how much the treated group differs from the control group. The Python stats module offers several tests to do this.

If the data is "paired", this can be used in the tests (they may get better because there is more "information or structure" in the data). So we need to know what paired data is in order to choose the right test.

Paired/related data

- Repeated measurements on the same object/individual, e.g. before and after treatment

Unpaired/independent data

- Independent, e.g. from separate individuals

We don't practise all tests (too much for one day).

WHICH TEST TO CHOOSE - THE DIAGRAM

Screen%20Shot%202018-05-31%20at%2022.50.37.png

4.4.1 Unpaired t test for normally distributed data with not assuming equal variance

Let us perform a test comparing the sepal width of the setosa and the virginica species. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html#scipy.stats.ttest_ind (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html#scipy.stats.ttest_ind)

In [12]:
```
df_virginica=dataframe[dataframe['class'] == 'Iris-virginica']
df_virginica.head() # Just cheking that we got the right data by looking
at the first five (head)
```

Out[12]:

	sepallength	sepalwidth	petallength	petalwidth	class
100	6.3	3.3	6.0	2.5	Iris-virginica
101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3.0	5.9	2.1	Iris-virginica
103	6.3	2.9	5.6	1.8	Iris-virginica
104	6.5	3.0	5.8	2.2	Iris-virginica

We did test that the setosa setal width is very normal. Let us also check the virginica setal width.

In [13]:
```
k2, p = stats.normaltest(df_virginica['sepalwidth']) # D Agostino-Pearso
n test
print("p-value = %1.2f" % p)
```

```
p-value = 0.28
```

This looks also very normal. How likely is it that these two samples come from a normal sample with the same mean?

```
In [14]: stats.ttest_ind(df_setosa['sepalwidth'],df_virginica['sepalwidth'], equa
         l_var = False) # False means we don't assume equal variances (Welsch's t
         -test)

Out[14]: Ttest_indResult(statistic=6.289384996672061, pvalue=9.58603917037916e-09)
```

Interpretation : It is extremly unlikely that these samples would come from a population with the same mean. The sample means were:

```
In [15]: print("%1.2f %1.2f" % (df_virginica.mean()['sepalwidth'], df_setosa.mean
         ()['sepalwidth']))

         2.97 3.42
```

4.4.2 Unpaired t test for normally distributed data assuming equal variance

```
In [16]: stats.ttest_ind(df_setosa['sepalwidth'],df_virginica['sepalwidth'], equa
         l_var = True) # True means we assume equal variances

Out[16]: Ttest_indResult(statistic=6.289384996672061, pvalue=8.916634067006443e-0
         9)
```

Interpretation : It is extremly unlikely that these samples would come from a population with the same mean and the same variance.

The p-value got slightly worse, but such small p-values have no meaning anyway. p-values below 0.01 are often not that precise. We just know they are below 0.01.

4.4.3 Unpaired t test for *not* normally distributed data

Compute the Mann-Whitney rank test on samples x and y. Only use with more than 20 data points. http://localhost:8888/notebooks/work/SDA-4-Hypothesis-Testing.ipynb#t-Test (http://localhost:8888/notebooks/work/SDA-4-Hypothesis-Testing.ipynb#t-Test)

```
In [17]: stats.mannwhitneyu(df_setosa['sepalwidth'],df_virginica['sepalwidth'])

Out[17]: MannwhitneyuResult(statistic=426.5, pvalue=5.904384107706829e-09)
```

Let us do this test with samples from the same population.

```
In [18]: print(stats.mannwhitneyu(df_setosa['sepalwidth'][:25],df_setosa['sepalwi
         dth'][25:50]))
         print(stats.mannwhitneyu(df_setosa['sepalwidth'][:10],df_setosa['sepalwi
         dth'][40:50]))
         print(stats.mannwhitneyu(df_setosa['sepalwidth'][:5],df_setosa['sepalwid
         th'][45:50]))

         MannwhitneyuResult(statistic=260.0, pvalue=0.1550563944967709)
         MannwhitneyuResult(statistic=42.5, pvalue=0.297517575814569)
         MannwhitneyuResult(statistic=9.0, pvalue=0.2641796636354743)
```

The p-value varies quite a lot with the size of the samples. Here we would need to study where it gets stable.

Compute the **Kolmogorov-Smirnov** statistic on 2 samples. This is a test for the null hypothesis that 2 independent samples are drawn from the same continuous distribution. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html#scipy.stats.ks_2samp (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html#scipy.stats.ks_2samp)

```
In [19]: stats.ks_2samp(df_setosa['sepalwidth'],df_virginica['sepalwidth'])
```

```
Out[19]: Ks_2sampResult(statistic=0.5, pvalue=3.6276162006545173e-06)
```

Interpretation: It is very unlikely that these sets come from the same population.

Let's do it on the same population.

```
In [20]: print(stats.ks_2samp(df_setosa['sepalwidth'][:25],df_setosa['sepalwidth
         '][25:50]))
         print(stats.ks_2samp(df_setosa['sepalwidth'][:10],df_setosa['sepalwidth
         '][40:50]))
         print(stats.ks_2samp(df_setosa['sepalwidth'][:5],df_setosa['sepalwidth
         '][45:50]))
```

```
         Ks_2sampResult(statistic=0.20000000000000007, pvalue=0.6485239915518617)
         Ks_2sampResult(statistic=0.20000000000000007, pvalue=0.9747892465409951)
         Ks_2sampResult(statistic=0.4, pvalue=0.6974048780205908)
```

For both tests we get very small p-values as expected, when the sets are from different species. When we split one of the datasets into two subsets, the p-values are higher.

4.4.4 Paired t test for normally distributed data

Calculate the T-test on TWO RELATED samples. This is a test for the null hypothesis that 2 related or repeated samples have identical average (expected) values.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html#scipy.stats.ttest_rel (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html#scipy.stats.ttest_rel)

Let's test if the swidth and the slength (related/paired) of the setosa sample have identical means.

```
In [21]: stats.ttest_rel(df_setosa['sepalwidth'],df_setosa['sepallength'])
```

```
Out[21]: Ttest_relResult(statistic=-42.799455674951105, pvalue=1.7724677938534975e
         -40)
```

As expected the p-value is very small (we know that swidth and slength have very different means). Let's test two subsamples from the same sample:

```
In [22]: stats.ttest_rel(df_setosa.iloc[0:25,1:2],df_setosa.iloc[25:50,1:2])
```

```
Out[22]: Ttest_relResult(statistic=array([1.08300729]), pvalue=array([0.2895664]))
```

As expected the p-value is high.

4.4.4 Paired test for *not* normally distributed data

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ (i.e. it is a paired difference test). It can be used as an alternative to the paired Student's t-test, t-test for matched pairs, or the t-test for dependent samples when the population cannot be assumed to be normally distributed. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon)

```
In [22]: # If you like, make your own example.
```

4.5 ANOVA tests (analysis of variance)

Python stats module provides a one-way Anova test. The one-way ANOVA tests the null hypothesis that two or more groups have the same population mean (a generalisation of the t-test to more than two samples). The test is applied to samples from two or more groups, possibly with differing sizes.

Please read the documentation about assumptions before using it: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html#scipy.stats.f_oneway (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html#scipy.stats.f_oneway)

```
In [24]: f, p_value = stats.f_oneway(df_setosa['sepalwidth'][0:10],df_setosa['sep
         alwidth'][10:20],df_setosa['sepalwidth'][20:30])
         p_value
Out[24]: 0.07117940263843624
```

```
In [25]: f, p_value = stats.f_oneway(df_setosa['sepalwidth'],df_virginica['sepalw
         idth'])
         p_value
Out[25]: 8.916634067006418e-09
```

In last example on assumption is not fulfilled.

Beyond one-way ANOVA

There are examples and code for two-way, three-way and four-way etc ANOVA out there. We don't have the time to discuss them to practise them today. You can do this example if you like (maybe with Iris data):

http://www.pybloggers.com/2016/03/three-ways-to-do-a-two-way-anova-with-python/ (http://www.pybloggers.com/2016/03/three-ways-to-do-a-two-way-anova-with-python/)

```
In [26]: from statsmodels.stats.anova import anova_lm
```

4.5 Final important words

If you are in doubt about your results because you think you have too little data or strangely distributed data you should/could:

- Check by using another tool and see if there are changes in the results
- Manipulate your (test) data (by adding/changing artifical datapoints etc) and study the effect on the result

In the end you should enable people to interprete your results in a correct way and come to the same conclusions as you. So your results should be reproducible within the uncertainties you state.