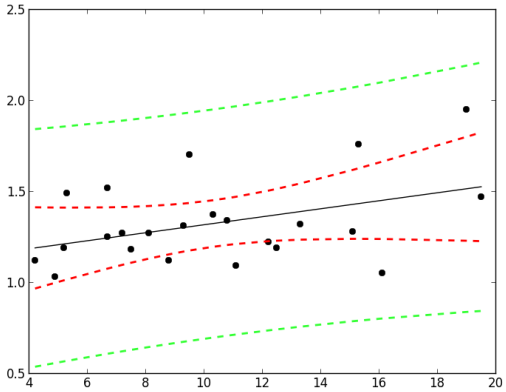


University of Applied Sciences

An Introduction to

Statistics

Author:
Thomas Haslwanter

email:
thomas.haslwanter@fh-linz.at

Version: 1.4
March 10, 2013

Contents

Chapter 1

Introduction

"Statistics ist the explanation of variance in the light of what remains unexplained."

Statistics was originally invented - as so many other things - by the famous mathematician C.F. Gauss, who said about his own work *"Ich habe fleissig sein müssen; wer es gleichfalls ist, wird eben so weit kommen"*. Even if your aspirations are not that high, you can get a lot out of statistics. In fact, if your work with real data, you probably won't be able to avoid it. Statistics can

- Describe variation.
- Make quantitative statements about populations.
- Make predictions.

Books: There are a number of good books about statistics. My favorite is [Altman(1999)]: it does not talk a lot about computers and modeling, but gives you a terrific introduction into the field. Many formulations and examples in this manuscript have been taken from that book. A more modern book, which is more voluminous and in my opionion a bit harder to read, is [Riffenburgh(2012)]. If you are interested in a simple introduction to modern regression modeling, check out [Kaplan(2009)].

WWW: On the web, you find good very extensive statistics information in English under *http://www.statsref.com/*. A good German webpage on statistics and regulatory issues is *http://www.reiter1.com/*.

1.1 Why Statistics?

Statistics will help you to

- Clarify the question.
- Identify the variable and the measure of that variable that will answer that question.
- Determine the required sample size.
- Find the correct analysis for your data.
- Make predictions based on your data.

1.2 Population and samples

While the whole *population* of a group has certain characteristics, we can typically never measure all of them. Instead, we have to confine ourselves to investigate a representative *sample* of this group, and estimate the properties of the population. Great care should be used to make the sample representative for the population you study.

1.3 Projects

For this course, you will choose a partner, and analyze one of the five projects described below. You will have to

1. Read up on the problem.
2. Design the study:
   (a) Determine the parameter to analyze.
   (b) Decide on the requirements of the sample population.
   (c) Plan the randomization.
   (d) Decide which test you want to use for the analysis.
3. Present your study design at the *Intermediate Presentation*
4. Analyze dummy data provided by me.
5. Generate the appropriate graphs.
6. Present the results at the *Final Presentation*.

Two groups will independently work on each project. You can choose from one of the following projects:

1. Surgical Trainer
2. Dry Eyes and Lasik
3. Recovery after Stroke
4. Active Office: Activity and Attention
5. Pathological Heart Muscles

1.3.1 Surgical Trainer

Researcher	David Fuerst
Topic	Development of a surgical trainer for surgeries on the spine.
Task	Develop a study design for a test if training on a model spine has the same educational benefits as training surgeries on human cadavers.

1.3.2 Dry Eyes and Lasik

Researcher	Michael Ring
Topic	Benefits of iodine rinsing on dry eyes after Lasik surgery.
Task	Corneal reshaping with a laser, or "Lasik"-surgery, often causes severe dry eye problems. Treatment at the "Therme Bad Hall" promises relieve for dry eye patients. Design a study that uses the device developed by Michael Ring to investigate if the benefits of such a treatment are significant.

1.3.3 Recovery after Stroke

Researcher	Thomas Minarik
Topic	Improvements due to home-training after stroke
Task	The typical treatment after stroke consists of intensive physiotherapy during the weeks in hospital, followed by intermittent treatment at the physiotherapist when the patients return home. We want to improve the recovery by introducing interactive home-based therapy. With this study we want to investigate if interactive home-based therapy leads to an improvement, compared to classical therapy.

1.3.4 Active Office: Activity and Attention

Researcher	Bernhard Schwartz
Topic	Improvements of attention due to increased activity in an office environment.
Task	Your ability to focus on your work depends on a lot of factors. One of them is your physical activity. We want to investigate how different working environments (e.g. working sitting vs. working standing) can affect your concentration at work.

1.3.5 Pathological Heart Muscles

Researcher	Sandra Mayr
Topic	The effects of *medication* and other factors on the structure of the cardiac muscle, as investigated by atomic force microscopy (AFM) on histological samples.
Task	To distinguish between healthy and hypertrophic hearts, the lengths of the sarcomeres of the heart muscles are measured using an AFM. Samples from hearts of healthy subjects and from patients with hypertrophic hearts are supplied by the Wagner-Jauregg Hospital in Linz.

1.4 Programming Matters

1.4.1 Python

There are three reasons why I have decided to use Python for this lecture.

1. It is the most elegant programming language that I know.
2. It is free.
3. It is powerful.

I have not seen many books on Python that I really liked. My favorite introductory book is [Harms and McDonald(2010)].

In general, I suggest that you start out by installing a Python distribution which includes the most important libraries. My favorites here are [Python(xy)(2013)] and [WinPython(2013)], which are very good starting points when you are using Windows. The former one has the advantage that most available documentation and help files also get installed locally. Mac and Unix users should check out the installations tips from Johansson (see Table 1.1).

There are also many tutorials available on the internet (Table 1.1). Personally, most of the time I just google; thereby I stick primarily a) to the official pages, and b) to *http://stackoverflow.com/*. Also, I have found user groups surprisingly active and helpful!

http://jrjohansson.github.com/	*Lectures on scientific computing with Python.* Great ipython notebooks!
http://docs.python.org/2/tutorial/	*The Python tutorial.* The official introduction.
http://scipy-lectures.github.com/	*Python Scientific Lecture Notes.* Pretty comprehensive.
http://www.greenteapress.com/thinkpython/	*ThinkPython* A free book on Python.
http://www.scipy.org/NumPy_for_Matlab_Users	*NumPy for Matlab Users* Start here if you have Matlab experience.

Table 1.1: Python on the WWW

If you decide to install things manually, you need the following modules in addition to the Python standard library:

- *numpy* ... For working with vectors and arrays.
- *scipy* ... All the essential scientific algorithms, including those for statistics.
- *matplotlib* ... The de-facto standard module for plotting and visualization.
- *pandas* ... Adds *DataFrames* (imagine powerful spreadsheets) to Python.
- *statsmodels* ... This one is only required if you want to look more into statistical modeling.

Also, make sure that you have a good programming environment! Currently, my favorite way of programming is similar to my old Matlab style: I first get the individual steps worked out interactively in *ipython*. And to write a program, I then go to either *Spyder* (which is free) or *Wing* (which is very good, but commercial).

Here an example, to get you started with Python (you find a corresponding ipython notebook under *http://nbviewer.ipython.org/url/work.thaslwanter.at/CSS/Code/getting_started.ipynb*):

Example-Session

Listing 1.1: gettingStarted.py

```
'''Short demonstration of Python for scientific data analysis

This script covers the following points:
* Plotting a sine wave
* Generating a column matrix of data
* Writing data to a text-file, and reading data from a text-file
* Waiting for a button-press to continue the program exectution
* Using a dictionary, which is similar to MATLAB structures
* Extracting data which fulfill a certain condition
* Calculating the best-fit-line to noisy data
* Formatting text-output
* Waiting for a keyboard-press
* Calculating confidence intervals for line-fits
* Saving figures

For such a short program, the definition of a "main" function, and calling
it by default when the module is imported by the main program, is a bit
superfluous. But it shows good Python coding style.

'''

'''
Author: Thomas Haslwanter
Version:  1.2
Date:  March-2013
'''
```

```
# In contrast to MATLAB, you explicitly have to load the modules that you need.
# And don't worry here about not knowing the right modules: numpy, scipy, and
# matplotlib is almost everything you will need most of the time, and you
# will quickly get used to those.
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

def main():
    '''Define the main function. '''
    # Create a sine-wave
    t = np.arange(0,10,0.1)
    x = np.sin(t)

    # Save the data in a text-file, in column form
    # The formatting is a bit clumsy: data are by default row variables; so to
    # get a matrix, you stack the two rows above each other, and then transpose
    # the matrix
    outFile = 'test.txt'
    np.savetxt(outFile, np.vstack([t,x]).T)

    # Read the data into a different variable
    inData = np.loadtxt(outFile)
    t2 = inData[:,0] # Note that Python starts at "0"!
    x2 = inData[:,1]

    # Plot the data, and wait for the user to click
    plt.show()
    plt.plot(t2,x2)
    plt.waitforbuttonpress()

    # Generate a noisy line
    t = np.arange(-100,100)
    # use a Python "dictionary" for named variables
    par = {'offset':100, 'slope':0.5, 'noiseAmp':4}
    x = par['offset'] + par['slope']*t + par['noiseAmp']*sp.randn(len(t))

    # Select "late" values, i.e. with t>10
    xHigh = x[t>10]
    tHigh = t[t>10]

    # Plot the "late" data
    plt.close()
    plt.plot(tHigh, xHigh)

    # Determine the best-fit line
    # To do so, you have to generate a matrix with "time" in the first
    # column, and a column of "1" in the second column:
    xMat = np.vstack((tHigh, np.ones(len(tHigh)))).T
    slope, intercept = np.linalg.lstsq(xMat, xHigh)[0]

    # Show and plot the fit, and save it to a PNG-file with a medium resolution.
    # The "modern" way of Python-formatting is used
    plt.hold(True)
    plt.plot(tHigh, intercept + slope*tHigh, 'r')
    plt.savefig('linefit.png', dpi=200)
    plt.waitforbuttonpress()
    plt.close()
    print 'Fit line: intercept = {0:5.3f}, and slope = {1:5.3f}'.format(
        intercept, slope)
    raw_input('Thanks for using programs by Thomas!')
```

```
    # If you want to know confidence intervals, best switch to "pandas"
    # Note that this is an advanced topic, and requires new data structures
    # such ad "DataFrames" and "ordinary-least-squares" or "ols-models".
    import pandas
    myDict = {'x':tHigh, 'y':xHigh}
    df = pandas.DataFrame(myDict)
    model = pandas.ols(y=df['y'], x=df['x'])
    print model
    raw_input('These are the summary results from Pandas - Hit any key to
        continue')


if __name__=='__main__':
    main()     # Execute the main function
```

1.4.2 Pandas

Pandas is a Python module which provides suitable data structures for statistical analysis. The following piece of code shows you how pandas can be used for data analysis:

Listing 1.2: pandasIntro.py

```
'''Introductions into using "Pandas": Simple work with two distributions
Pandas is a Python framework for statistical analysis. It allows you to label
and group data, etc.

'''

'''
Author : Thomas Haslwanter
Date : March 2013
Ver : 1.1
'''

import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import numpy as np
from os.path import join
from getdata import getData

def simple_analysis():
    '''Simple statistical analysis of dummy data'''

    # Generate dummy data
    # Since there are more rendeer than elephants, I cannot use a Python
    # dictionary. Instead I use "Series" from pandas. The other common pandas
    # datastructure is "DataFrame".
    data = {'rendeer' : pd.Series(stats.norm.rvs(size=70, loc=300, scale=50)),
        'elephants' : pd.Series(stats.norm.rvs(size=50, loc=500, scale=100))}
    df = pd.DataFrame(data)

    # Check some simple stats
    print 'Numbers'
    print df.count()

    print '\nMeans'
    print df.mean()

    print '\nMins'
    print df.min()

    print '\nMax'
```

```
    print df.max()

    # Confidence intervals for the mean elephant
    se =df['elephants'].std()/np.sqrt(df['elephants'].count())
    level = 0.975
    tval = stats.t.ppf(level, df['elephants'].count()-1)
    cis = df['elephants'].mean() + tval*se*np.array([-1., 1])
    print '95% CI for the weight of elephants: {0} - {1}'.format(cis[0], cis[1])

    # Is the weight of elephants different from 500 kg?
    testWeight = 500
    (tv, pv) = stats.ttest_1samp(df['elephants'].dropna(), testWeight)

    if pv < 0.05:
        print 'Elephants don''t weigh {0}kg'.format(testWeight)
    else:
        print 'Elephants weigh approximately {0}kg'.format(testWeight)

    # Are elephants heavier than rendeer?

    (tv, pv) = stats.ttest_ind(df['elephants'].dropna(), df['rendeer'])
    if pv < 0.05:
        if df['elephants'].mean() > df['rendeer'].mean():
            print 'Elephants are heavier than rendeer'
        else:
            print 'Rendeer are heavier than elephants'
    else:
        'Elephants and rendeer weigh the same'

    return df


def simple_plots(df):
    ''' Make some plots '''
    df.plot()
    plt.show()

    df.hist()
    plt.show()

    df.boxplot()
    plt.show()

    df.plot(style=['x','o'])
    plt.show()


def example_altman():
    '''Example from Altman "Practical statistics for medical research'''

    data = getData('altman_94.txt')

    lean = pd.Series(data[data[:,1]==1,0])
    obese = pd.Series(data[data[:,1]==0,0])

    df = pd.DataFrame({'lean':lean, 'obese':obese})

    print(df.mean())
    plt.show()

    df.boxplot()
    plt.show()
```

```
    stats.ttest_ind(lean, obese)

if __name__ == '__main__':
    df = simple_analysis()
    simple_plots(df)
    example_altman()
```

Here is also a good place to introduce the short function that we will use a number of times to simplify the reading in of data:

Listing 1.3: getdata.py

```
'''Get data for the Python programs for statistics, FH OOe.
Most are from the tables in the Altman book "Practical Statistics for Medical
    Research.
I use these data quite often, so I have put those by default in a subdirectory
"data_altman". This function reads them from there.'''

'''
Author:  Thomas Haslwanter
Date:    March-2013
Version: 1.3
'''

from os.path import join
from numpy import genfromtxt
import os

def getData(inFile, subDir='data_altman'):
    '''Data are taken from examples in D. Altman, "Practical Statistics for
       Medical Research" '''
    dataDir = os.path.join('..','Data', subDir)
    inFile = join(dataDir, inFile)
    try:
        data = genfromtxt(inFile, delimiter=',')
    except IOError:
        print inFile + ' does not exist!'
        data = ()
    return data

if __name__ == '__main__':
    data = getData('altman_93.txt')
    print data
```

Chapter 2

Basic Principles

2.1 Datatypes

The type of your data is essential for the choice of test that you have to use for your data analysis. Your data can have one of the following datatypes:

2.1.1 Categorical

boolean

Some data can only have two values. For example,

1. male/female
2. smoker/non-smoker

nominal

Many classifications require more than two categories, e.g. *married / single / divorced*

ordinal

These are ordered categorical data, e.g. *very few / few / some / many / very many*

2.1.2 Numerical

Numerical discrete

For example *Number of children: 0 1 2 3 4 5*

2.1.3 Continuous

Whenever possible, it is best to record the data in their original continuous format, and only with a sensible number of decimal places. For example, it does not make sense to record the body size with more than 1 mm accuracy, as there are larger changes in body height between the size in the morning and the size in the evening, due to compression of the intervertebral disks.

2.2 Data Display

When working with a statistical data set, you should *always* first look at the raw-data. Our visual system is incredibly good at recognizing patterns in visually represented data.

2.2.1 Scatter Plots

This is the simplest way of representing your data: just plot each individual data point. (In cases where many data points are superposed, you may want to add a little bit of jitter to show each data point.)

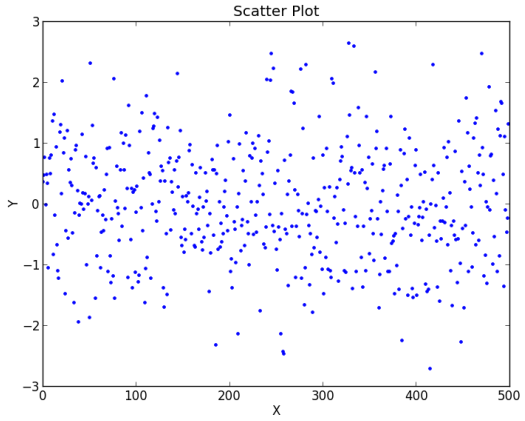


Figure 2.1: Scatter plot

2.2.2 Histograms

Histograms provide a first good overview of the distribution of your data. If you divide by the overall number of data points, you get a *relative frequency histogram*; and if you just connect the top center points of each bin, you obtain a *relative frequency polygon*.

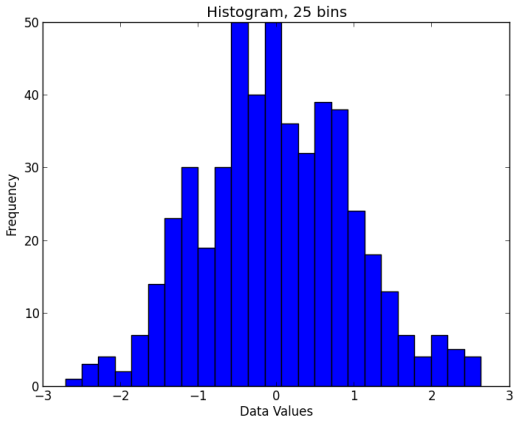


Figure 2.2: Histogram

2.2.3 Cumulative Frequencies

Cumulative frequency curves indicate the number (or percent) of data with less than a given value. This is important for the statistical analysis (e.g. when we want to know the data range containing 95% of all the values). Cumulative frequencies are also useful for comparing the distribution of values in two or more different groups of individuals.

When you use percentage points, the cumulative frequency presentation has the additional advantage that it is bounded:

$$0 \leq x \leq 1$$

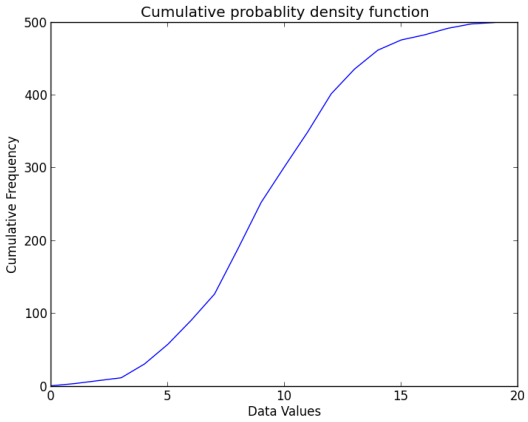


Figure 2.3: Cumulative frequency function

2.2.4 Box Plots

Box plots are frequently used in scientific publications to indicate values in two or more groups. The error bars typically indicate the *range*. However, outliers are often excluded, and plotted separately. There are a number of tests to check for outliers. One of them is to check for data which lie more than 1.5 * *inter-quartile-range* (IQR) above or below the first/third quartile (see Section 3.1.2).

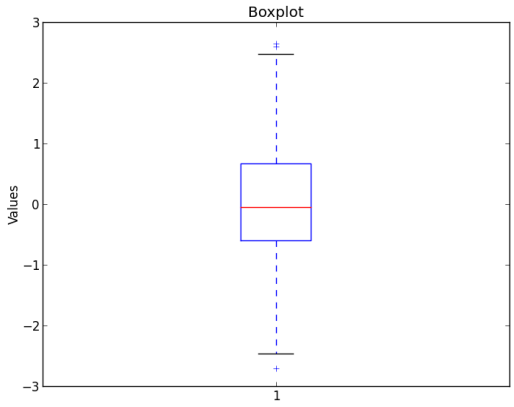


Figure 2.4: Box plot

2.2.5 Programs: Data Display

Listing 2.1: showStats.py

```
''' Show different ways to present statistical data
This script is written in "MATLAB" or "ipython" style, to show how
best to use Python interactively.
Note than in ipython, the "show()" commands are automatically generated.
The examples contain:
- scatter plots
```

```
- histograms
- boxplots
- probplots
- cumulative density functions
- regression fits

'''

'''
Author: thomas haslwanter
Date:   Dec-2012
Version: 1.1
'''

# First, import the libraries that you are going to need. You could also do
# that later, but it is better style to do that at the beginning.

# pylab imports the numpy, scipy, and matplotlib.pyplot libraries into the
# current environment
from pylab import *

import scipy.stats as stats

# Univariate data -------------------------
# Generate data that are normally distributed
x = randn(500)

# Scatter plot
plot(x,'.')
title('Scatter Plot')
xlabel('X')
ylabel('Y')
draw()
show()

# Histogram
hist(x)
xlabel('Data Values')
ylabel('Frequency')
title('Histogram, default settings')
show()

hist(x,25)
xlabel('Data Values')
ylabel('Frequency')
title('Histogram, 25 bins')
show()

# Cumulative probability density
numbins = 20
cdf = stats.cumfreq(x,numbins)
plot(cdf[0])
xlabel('Data Values')
ylabel('Cumulative Frequency')
title('Cumulative probablity density function')
show()

# Boxplot
# The error bars indiacte the range, and the box consists of the
# first, second (middle) and third quartile
boxplot(x)
title('Boxplot')
ylabel('Values')
```

```
show()

boxplot(x, vert=False)
title('Boxplot, horizontal')
xlabel('Values')
show()

# Check for normality
_ = stats.probplot(x, plot=plt)
title('Probplot - check for normality')
show()

# Bivariate data -------------------------

# Generate data
x = randn(200)
y = 10+0.5*x+randn(len(x))

# Scatter plot
scatter(x,y)
# This one is quite similar to "plot(x,y,'.')"
title('Scatter plot of data')
xlabel('X')
ylabel('Y')
show()

# LineFit
M = vstack((ones(len(x)), x)).T
pars = linalg.lstsq(M,y)[0]
intercept = pars[0]
slope = pars[1]
scatter(x,y)
hold(True)
plot(x, intercept + slope*x, 'r')
show()
```

2.3 Study Design

To design a medical study properly is not only advisable - it is even required by ISO 14155-1:2003, for *Clinical investigations of medical devices for human subjects*. This norm specifies many aspects of your clinical study. It enforces the preparation of a *Clinical Investigation Plan (CIP)*, specifying

- The designation of a *monitor* for the investigation.
- The designation of a *clinical investigator*.
- Specification the data handling.
- Specification of the inclusion/exclusion criteria for the subjects.
- Specification of the paradigm.
- Specification and justification of the chosen sample numbers.
- Description of the data analysis.

2.3.1 Types of Studies

Observational or experimental

With *observational* studies the researcher only collects information, but does not interact with the study population. In contrast, in *experimental* studies the researcher deliberately influences events (e.g. treats the patient with a new type of treatment) and investigates the effects of these interventions.

Prospective or retrospective

In *prospective* studies the data are collected, starting with the beginning of the study. In contrast, a *retrospective* study takes data acquired from previous events, e.g. routine tests taken at a hospital.

Longitudinal or cross-sectional

In *longitudinal* investigations, the researcher collects information over a period of time, maybe multiple times from each patient. In contrast, in *cross-sectional* studies individuals are observed only once. For example, most surveys are cross-sectional, but experiments are usually longitudinal.

Case control and Cohort studies

In *case control* studies, first the patients are treated, and then they are selected for inclusion in the study, based on some characteristic (e.g. if they responded to a certain medication). In contrast, in *cohort studies*, first subjects of interest are selected, and then these subjects are studied over time, e.g. for their response to a treatment.

2.3.2 Design of Experiments

Bias

In general, when selecting our subject you try to make them representative of the population that you want to study; and you try to conduct your experiments in a way representative of investigations by other researchers. However, it is *very* easy to get a *bias* into your data. Bias can arise from a number of sources:

- The selection of subjects.
- The structure of the experiment.
- The measurement device.
- The analysis of the data.

Care should be taken to avoid bias as much as possible.

Randomized controlled trial

The gold standard for experimental scientific clinical trials is the *randomized controlled trial*. Thereby bias is avoided by splitting the subjects to be tested into an *intervention group* and a *control group*. The group allocation is made *random*. By having the groups differ in only one aspect, i.e. is the factor *treatment*, we should be able to detect the effect of the treatment on the patients. Factors that can affect the outcome of the experiment are called *covariates* or *confoundings*. Through *randomization*, covariates should be balanced across the groups.

Randomization

This may be one of the most important aspects of experimental planning. Randomization is used to avoid bias as much as possible, and there are different ways to randomize an experiment. For the randomization, *random number generators*, which are available with most computer languages, can be used. To minimize the chance of bias, the randomly allocated numbers should be presented to the experimenter as late as possible.

Depending on the experiment, there are different ways to randomize the group assingment.

Simple randomization This procedure is robust against selection and accidental bias. The disadvantage is that the resulting groupsize can differ significantly.

For many types of data analysis it is important to have the same sample number in each group. To achieve this, other options are possible:

Block randomization This is used to keep the number of subjects in the different groups closely balanced at all times. For example, if you have two types of treatment, A and B, you can allocate them to two subjects in the following blocks:

1. AABB
2. ABAB
3. ABBA
4. BBAA
5. BABA
6. BAAB

Based on this, you can use a random number generator to generate random integers between 1 and 6, and use the corresponding blocks to allocate the respective treatments. This will keep the number of subjects in each group always almost equal.

Minimization A closely related, but not completely random way to allocate a treatment is *minimization*. Thereby you take whichever treatment has the smallest number of subjects, and allocate this treatment with a probability greater than 0.5 to the next patient.

Stratified randomization Sometimes you may want to include a wider variety of subjects, with different characteristics. For example, you may choose to have younger as well as older subjects. In that case you should try to keep the number of subjects within each *stratum* balanced. For this you will have to keep different lists of random numbers for each group of subjects.

Crossover studies

An alternative to randomization is the *crossover* design of studies. A crossover study is a longitudinal study in which subjects receive a sequence of different treatments. Every subject receives every treatment. To avoid causal effects, the sequence of the treatment allocation should be randomized.

Blinding

Consciously or not, the experimenter can significantly influence the outcome of an experiment. For example, a young researcher with a new "brilliant" idea for a new treatment will be bias in the execution of the experiment, as well in the analysis of the data, to see his hypothesis confirmed. To avoid such a subjective influence, ideally the experimenter as well as the subject should be blinded to the therapy. This is referred to as *double blinding*.

Replication

For variable measurements it is helpful to have a number of independent repetitions of each measurement.

Sample selection

When selecting your subjects, you should take care of two points:

1. Make sure that the samples are representative of the population.
2. In comparative studies, care is needed in making groups similar with respect to known sources of variation.

For example, if you select your subjects randomly from patients at a hospital, you automatically bias your sample towards subjects with health problems.

Sample size

Many studies fail, because the sample size is too small to observed an effect of the desired magnitude. To plan your sample size, you have to know

- What is the variance of the parameter in the population you are investigating.
- What is the magnitude of the effect you are interested in, relative to the standard deviation of the parameter.

2.3.3 Structure of Experiments

In a designed experiment, there may be several conditions, called *factors*, that are controlled by the experimenter. If each combination of factors is tested, we talk about a *factorial design* of the experiment.

In planning the analysis, you have to keep the important distinction between *within subject* comparisons, and *between subjects* comparisons.

2.3.4 Data Management

Documentation

Make sure that you document all the factors that may influence your results:

- The date and time of the experiment.
- Information about the experimenters and the subjects.
- The exact paradigm that you have decided on.
- Anything noteworthy that happens during the experiment.

Data Handling

You can already significantly facilitate the data handling by storing your data with telltale names. For example, if you execute your experiments in Vienna and in Linz, you can store your rawdata with the format "[town][year][month][day].dat". For example, an experiment in Vienna on April 1, 2013 would be stored as "vi20130401.dat".

When you have finished recording the data, back up your data right away. Best do that into a directory that is separate from the one where you do your data analysis afterwards.

2.3.5 Clinical Investigation Plan

The ISO norm 14155 specifies in detail the requirements for the *clinical investigation plan (CIP)*:

1. Type of study (e.g. double-blind, with or without control group etc.).
2. Discussion of the control group and the allocation procedure.
3. Description of the paradigm.
4. Description and justification of primary endpoint of study.
5. Description and justification of chosen measurement variable.
6. Measurement devices and their calibration.
7. Inclusion criteria for subjects.
8. Exclusion criteria for subjects.
9. Point of inclusion ("When is a subject part of the study?")
10. Description of the measurement procedure.
11. Criteria and procedures for the dropout of a subject.
12. Chosen sample number and level of significance, and their justification.
13. Procedure for documentation of negative effects or side-effects.
14. List of factors that can influence the measurement results or their interpretation.
15. Procedure for documentation, also for missing data.
16. Statistical analysis procedure.

Chapter 3

Distributions of one Variable

3.1 Characterizing a Distribution

3.1.1 Distribution Center

Mean

By default, when we talk about the *mean value* we mean the *arithmetic mean* $\bar{x}$:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.1}$$

Median

The *median* is that value that comes half-way when the data are ranked in order. In contrast to the mean, it is not affected by outlying data points.

Mode

The *mode* value is the most frequently occurring value in a distribution.

Geometric Mean

In some situations the *geometric mean* can be useful to describe the location of a distribution. It is usually close to the median, and can be calculated via the arithmetic mean of the log of the values.

3.1.2 Quantifying Variability

Range

This one is fairly easy: it is the difference between the highest and the lowest data value. The only thing that you have to watch out for: after you have acquired your data, you have to check for *outliers*, i.e. data points with a value much higher or lower than the rest of the data. Often, such points are caused by errors in the selection of the sample or in the measurement procedure. There are a number of tests to check for outliers. One of them is to check for data which lie more than 1.5**inter-quartile-range* (IQR) above or below the first/third quartile (see below).

Centiles

The *Cumulative distribution function (CDF)* tells you for each value which percentage of the data has a lower value (Figure 3.1). The value below which a given percentage of the values occur is called *centile* or *percentile*, and corresponds to a value with a specified cumulative frequency.

For example, when you look for the data range which includes 95% of the data, you have to find the 2.5^{th} and the 97.5^{th} percentile of your sample distribution.

The 50th percentile is the *median*.

Also important are the *quartiles*, i.e. the 25th and the 75th percentile. The difference between them is sometimes referred to as *inter-quartile range (IQR)*.

Median, upper and lower quartile are used for the data display in box plots (Fig.2.4).

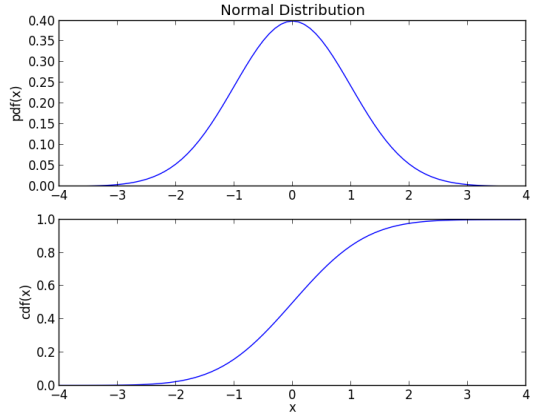


Figure 3.1: *Probability distribution function* (top) and *Cumulative distribution function* (bottom) of a normal distribution.

Standard Deviation and Variance

The *variance* (SD) of a distribution is defined as

$$var = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1} \tag{3.2}$$

Note that we divide by *n-1* rather than the more obvious n: dividing by n gives the variance of the observations around the sample mean, but we virtually always consider our data as a sample from some larger population and wish to use the sample data to estimate the variability in the population. Dividing by $n-1$ gives us a better estimate of the population variance.

The *standard deviation* is simply given by the square root of the variance:

$$s = \sqrt{var} \tag{3.3}$$

In statistics it is often common to denote the population standard deviation with σ, and the sample standard deviation with s.

Watch out: in Python, by default the variance is calculated for "n". You have to set "ddof=1" to obtain the variance for "n-1":

```
In[19]: data = arange(7,14)

In[20]: std(data, ddof=0)
Out[20]: 2.0
```

```
In[21]: std(data, ddof=1)
Out[21]: 2.1602468994692865
```

Standard Error

While the standard deviation is a good measure for the distribution of your values, often you are more interested in the distribution of the mean value. For example, when you measure the response to a new medication, you might be interested in how well you can characterize this response, i.e. is how well you know the mean value. This measure is called the *standard error of the mean*, or sometimes just the *standard error*. In a single sample from a population with a standard deviation of σ the variance of the sampling distribution of the mean is σ^2/n, and so the standard error of the mean is $\sigma/\sqrt{n}$.

Skewness

Distributions are *skewed* if they depart from symmetry. For example, if you have a measurement that cannot be negative, which is usually the case, then we can infer that the data have a skewed distribution if the standard deviation is more than half the mean. Such an asymmetry is referred to as *positive skewness*. The opposite, negative skewness, is rare.

Central Limit Theorem

The central limit theorem states that for identically distributed independent random variables (also referred to as *random variates*), the mean of a sufficiently large number of these variables will be approximately normally distributed.

3.2 Distribution Functions

The variable for a standardized distribution function is often called *statistic*. So you often find expressions like "the z-statistic" (for the normal distribution function), the "t-statistic" (for the t-distribution) or the "F-statistic" (for the F-distribution).

3.2.1 Probability and Samples

3.2.2 Normal Distribution

The *Normal distribution* or *Gaussian distribution* is by far the most important of all the distribution functions. This is due to the fact that the mean values of *all* distribution functions approximate a normal distribution for large enough sample numbers. Mathematically, the normal distribution is characterized by a mean value μ, and a standard deviation σ:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3.4}$$

where $-\infty < x < \infty$, and $f_{\mu,\sigma}$ is the *probability density function (PDF)* .

For smaller sample numbers, the sample distribution can show quite a bit of variability. For example, look at 25 distributions generated by sampling 100 numbers from a normal distribution (Fig. 3.3)

Some examples of applications are:

- If the average man is 175 cm tall with a variance of 6 cm, what is the probability that a man found at random will be 183 cm tall?

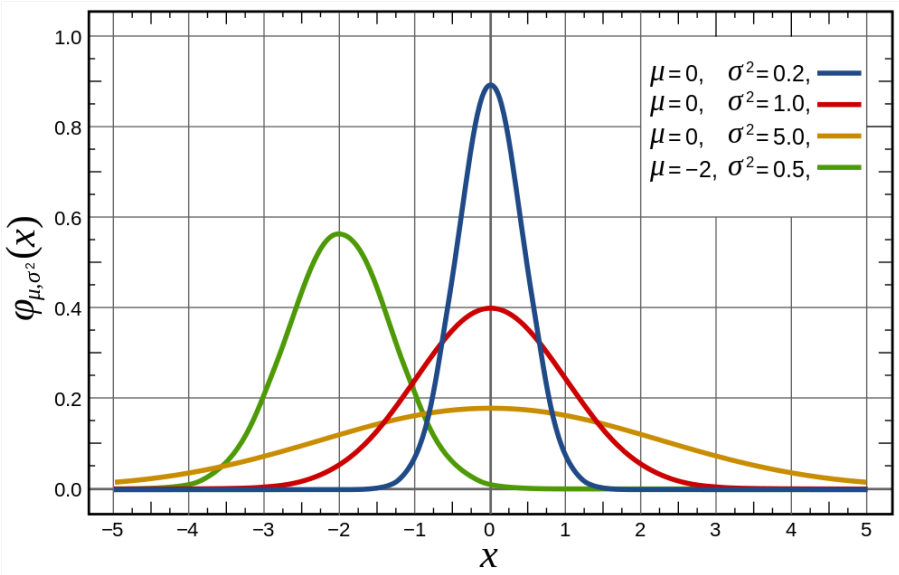


Figure 3.2: Normal Distribution

Range	Probability of being within range	outside range
mean ± 1SD	0.683	.317
mean ± 2SD	0.954	0.046
mean ± 3SD	0.9973	0.0027

Table 3.1: Tails of a normal distribution.

- If the average man is 175 cm tall with a variance of 6 cm and the average woman is 168 cm tall with a variance of 3 cm, what is the probability that the average man from a given sample will be shorter than the average woman from a given sample?
- If cans are assumed to have a variance of 4 grams, what does the average weight need to be in order to ensure that the 99% of all cans have a weight of at least 250 grams?

The normal distribution with parameters μ and σ is denoted as $N(\mu, \sigma)$. If the *random variate (rv)* X is normally distributed with expectation μ and standard deviation σ, one denotes: $X \sim N(\mu, \sigma)$ or $X \in N(\mu, \sigma)$.

Figure 3.4 shows a number of functions are commonly used to select appropriate points from the normal distribution function:

- *Probability density function (PDF)*: note that to obtain the probability for the variable appearing in a certain interval, you have to *integrate* the PDF over that range.
- *Cumulative distribution function (CDF)*: gives the probability of obtaining a value smaller than the given value
- *Survival function (SF)*: 1-CDF
- *Percentile point function (PPF)*: the inverse of the CDF. Answers the question "Given a certain probability, what is the corresponding value for the CDF?"
- *Inverse survival function (ISF)*: the name says it all.

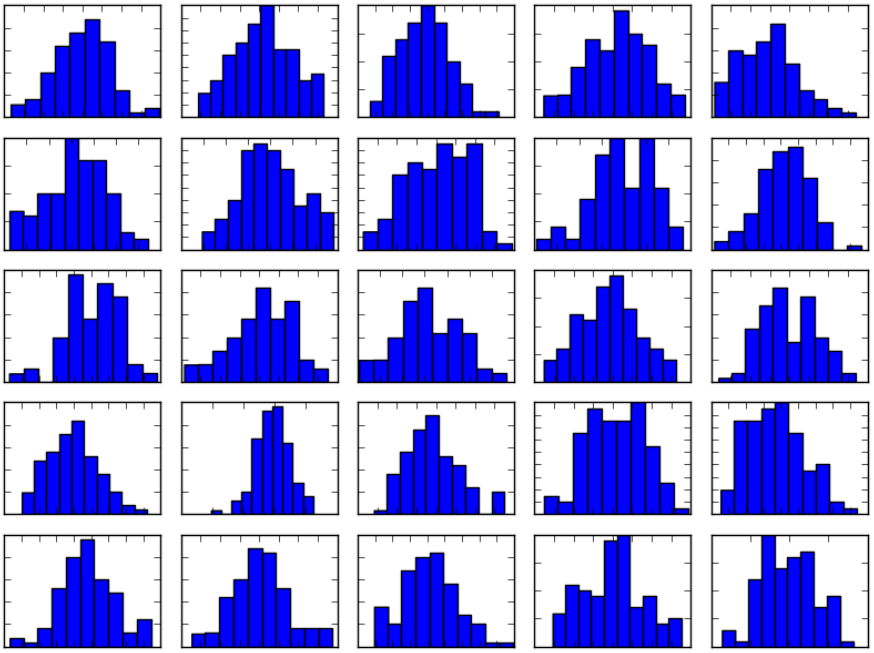


Figure 3.3: 25 randomly generated normal distributions of 100 points.

3.2.3 Other Continuous Distributions

t Distribution

For a small number of samples (ca.$<$ 10) from a normal distribution, the distribution of the mean deviates slightly from the normal distribution. The reason is that the sample mean does not coincide exactly with the population mean. This modified distribution is the *t-distribution*, and converges for larger values towards the normal distribution (Fig. 3.5).

Chi-square Distribution

The *Chi-square distribution* is related to normal distribution in a simple way: If a random variable X has a normal distribution ($X \in N(0,1)$), then X^2 has a chi-square distribution, with one degree of freedom ($X^2 \in \chi_1^2$). The sum squares of n independent and standard normal random variables has a chi-square distribution with n degrees of freedom:

$$\sum_{i=1}^{n} X_i^2 \in \chi_n^2 \tag{3.5}$$

F Distribution

Named after Sir Ronald Fisher, who developed the F distribution for use in determining critical values in ANOVAs (*ANalysis Of VAriance*). The cutoff values in an F table are found using three variables:

- ANOVA numerator degrees of freedom
- ANOVA denominator degrees of freedom
- significance level

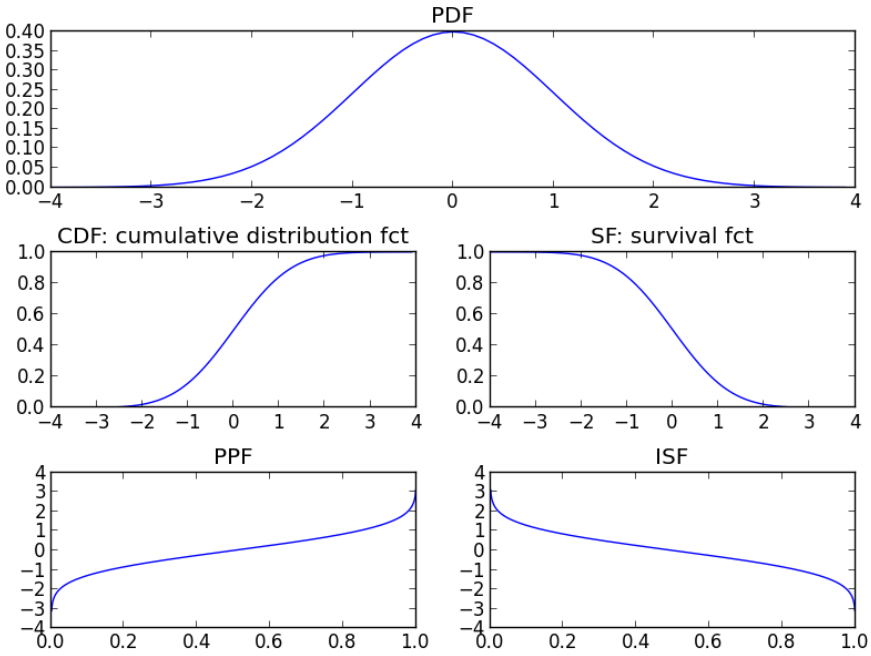


Figure 3.4: Utility functions for continuous distributions, here for the normal distribution.

ANOVA compares the size of the variance between two different samples. This is done by dividing the larger variance over the smaller variance. The formula for the resulting *F statistic* is:

$$F(r_1, r_2) = \frac{\chi^2_{r1}/r_1}{\chi^2_{r2}/r_2} \tag{3.6}$$

where χ^2_{r1} and χ^2_{r2} are the chi-square statistics of sample one and two respectively, and r_1 and r_2 are their degrees of freedom, i.e. the number of observations.

F-Test of Equality of Variances One example could be if you want to compare apples that look alike but are from different trees and have different sizes. If you want to investigate whether they have the same variance of the weight on average, you have to calculate

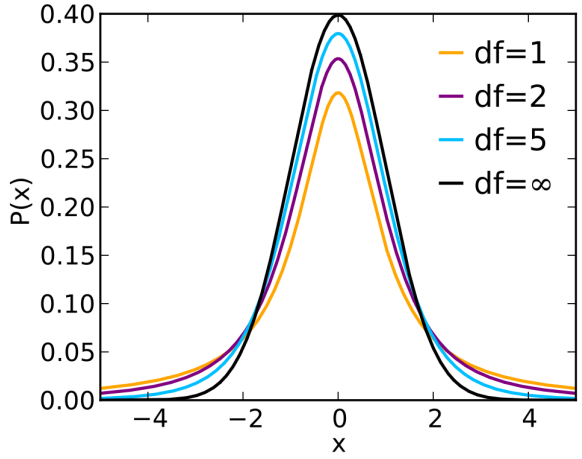


Figure 3.5: t Distribution

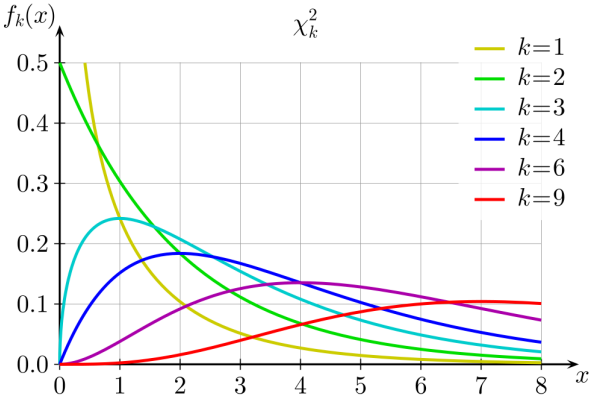


Figure 3.6: Chi-square Distribution

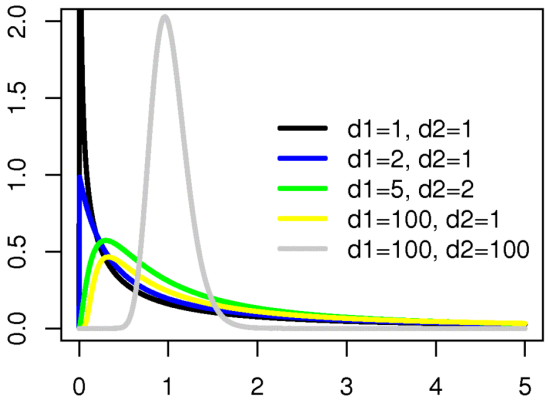


Figure 3.7: F Distribution

$$F = \frac{S_x^2}{S_y^2} \tag{3.7}$$

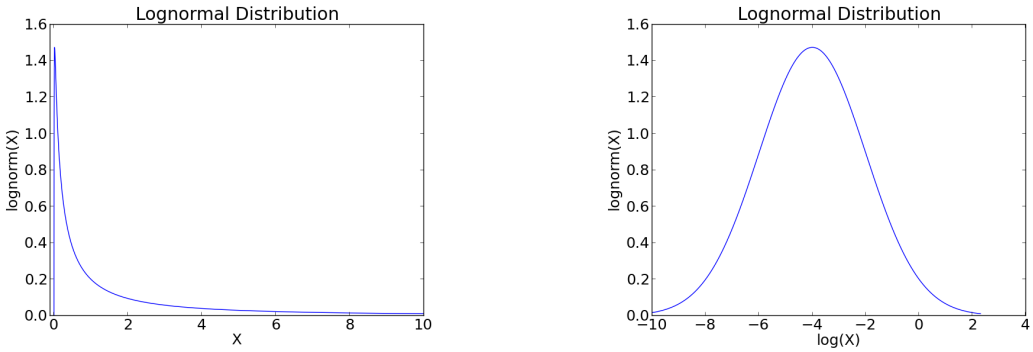
where S_x ist he sample standard deviation of the first batch of apples, and S_y the sample standard deviation for the second batch of apples.

There are three apples from the first tree that weigh 110, 121 and 143 grams respectively, and four from the other which weigh 88, 93, 105 and 124 grams respectively. The F statistic is $F1.05$, and has $n-1$ and $m-1$ degrees of freedom, where n and m are the number of apples in each batch. The code sample below shots what the F statistic is close to the center of the distribution, so we cannot reject the hypthesis that the two batches have the same variance.

```
In [1]:  apples1 = array([110, 121, 143])
In [2]:  apples2 = array([88, 93, 105, 124])
In [3]:  fval = std(apples1, ddof=1)/std(apples2, ddof=1)
In [4]:  fd = stats.distributions.f(3,4)
In [5]:fd.cdf(fval)
Out[27]: 0.537640478466751
```

Lognormal Distribution

In some circumstances a set of data with a positively skewed distribution can be transformed into a symmetric distribution by taking logarithms. Taking logs of data with a skewed distribution will often give a distribution that is near to normal (see Figure 3.8).



(a) Plotted against a linear abscissa.

(b) Plotted against a logarithmic abscissa.

Figure 3.8: Lognormal distribution

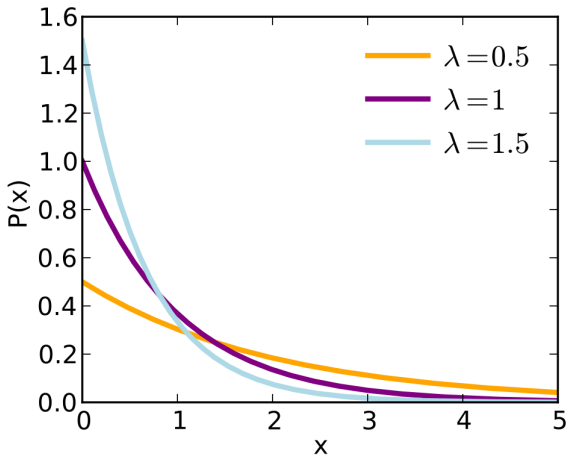


Figure 3.9: Exponential Distribution

Exponential Distribution

For a stochastic variable X with an *exponential distribution*, the probability distribution function is:

$$f_x(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.8}$$

The exponential PDF is shown in Figure 3.9

Uniform Distribution

This is a simple one: an even probability for all data values (Figure 3.10). Not very common for real data.

Programs: Continuous Distribution Functions

Listing 3.1: Continuous

```
''' Different continuous distribution functions.
- Normal distribution
- Exponential distribution
- T-distribution
- F-distribution
```

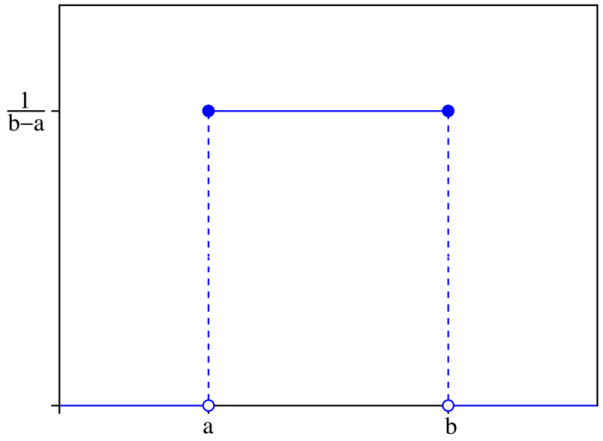


Figure 3.10: Uniform Distribution

```
- Logistic distribution
- Lognormal distribution
- Uniform distribution

'''

'''
Author:  Thomas Haslwanter
Date:    March-2013
Version: 1.3
'''

# Note: here I use the iPython approach, which is best suited for interactive
    work
from pylab import *
from scipy import stats
matplotlib.rcParams.update({'font.size': 18})

x = linspace(-10,10,201)
def showDistribution(d1, d2, tTxt, xTxt, yTxt, legendTxt, xmin=-10, xmax=10):
    '''Utility function to show the distributions, and add labels and title.'''
    plot(x, d1.pdf(x))
    if d2 != '':
        hold(True)
        plot(x, d2.pdf(x), 'r')
        legend(legendTxt)
    xlim(xmin, xmax)
    title(tTxt)
    xlabel(xTxt)
    ylabel(yTxt)
    show()


x = linspace(-10,10,201)
# Normal distribution
showDistribution(stats.norm, stats.norm(loc=2, scale=4),
                 'Normal Distribution', 'Z', 'P(Z)','')

# Exponential distribution
showDistribution(stats.expon, stats.expon(loc=-2, scale=4),
                 'Exponential Distribution', 'X', 'P(X)','')

# Students' T-distribution
# ... with 4, and with 10 degrees of freedom (DOF)
```

```
plot(x, stats.norm.pdf(x), 'g')
hold(True)
showDistribution(stats.t(4), stats.t(10),
                 'T-Distribution', 'X', 'P(X)',['normal', 't=4', 't=10'])

# F-distribution
# ... with (3,4) and (10,15) DOF
showDistribution(stats.f(3,4), stats.f(10,15),
                 'F-Distribution', 'F', 'P(F)',['(3,4) DOF', '(10,15) DOF'])

# Uniform distribution
showDistribution(stats.uniform,'' ,
                 'Uniform Distribution', 'X', 'P(X)','')

# Logistic distribution
showDistribution(stats.norm, stats.logistic,
                 'Logistic Distribution', 'X', 'P(X)',['Normal', 'Logistic'])

# Lognormal distribution
x = logspace(-9,1,1001)+1e-9
showDistribution(stats.lognorm(2), '',
                 'Lognormal Distribution', 'X', 'lognorm(X)','', xmin=-0.1)

# The log-lin plot has to be done by hand:
plot(log(x), stats.lognorm.pdf(x,2))
xlim(-10, 4)
title('Lognormal Distribution')
xlabel('log(X)')
ylabel('lognorm(X)')
show()
```

3.2.4 Discrete Distributions

While the functions describing continuous distributions are referred to as *probability distribution functions*, discrete distributions are described by *probability mass functions*.

Binomial Distribution

The Binomial is associated with the question "Out of a given number of trials, how many will succeed?" Some example questions that are modeled with a Binomial distribution are:

- Out of ten tosses, how many times will this coin land "heads"?
- From the children born in a given hospital on a given day, how many of them will be girls?
- How many students in a given classroom will have green eyes?
- How many mosquitos, out of a swarm, will die when sprayed with insecticide?

We conduct n repeated experiments where the probability of success is given by the parameter p and add up the number of successes. This number of successes is represented by the random variable X. The value of X is then between 0 and n.

When a random variable X has a Binomial Distribution with parameters p and n we write it as $X \sim Bin(n,p)$ or $X \sim B(n,p)$ and the probability mass function is given at $X = k$ by the equation:

$$P[X=k] = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & 0 \le k \le n \\ 0 & \text{otherwise} \end{cases} \quad 0 \le p \le 1, \quad n \in \mathbb{N} \tag{3.9}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

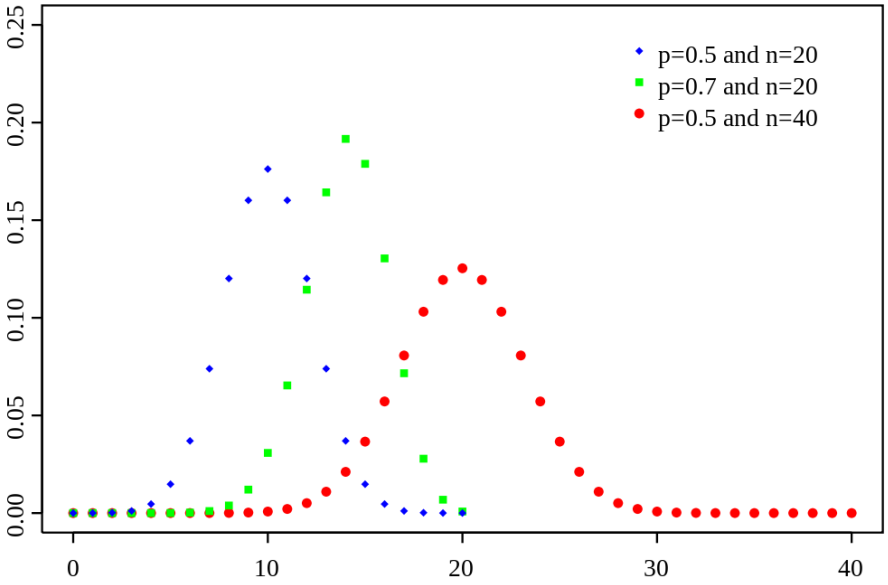


Figure 3.11: Binomial Distribution

Poisson Distribution

Any French speaker will notice that "Poisson" means "fish", but really there's nothing fishy about this distribution. It's actually pretty straightforward. The name comes from the mathematician Simon-Denis Poisson (1781-1840).

The Poisson Distribution is "very similar" to the Binomial Distribution. We are examining the number of times an event happens. The difference is subtle. Whereas the Binomial Distribution looks at how many times we register a success over a fixed total number of trials, the Poisson Distribution measures how many times a discrete event occurs, over a period of continuous space or time. There isn't a "total" value n. As with the previous sections, let's examine a couple of experiments or questions that might have an underlying Poisson nature.

- How many pennies will I encounter on my walk home?
- How many children will be delivered at the hospital today?
- How many products will I sell after airing a new television commercial?
- How many mosquito bites did you get today after having sprayed with insecticide?
- How many defects will there be per 100 metres of rope sold?

What's a little different about this distribution is that the random variable X which counts the number of events can take on *any non-negative integer* value. In other words, I could walk home and find no pennies on the street. I could also find one penny. It's also possible (although unlikely, short of an armored-car exploding nearby) that I would find 10 or 100 or 10,000 pennies.

Instead of having a parameter p that represents a component probability like in the Binomial distribution, this time we have the parameter "lambda" or λ which represents the "average or expected" number of events to happen within our experiment. The probability mass function of the Poisson is given by

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.10}$$

.

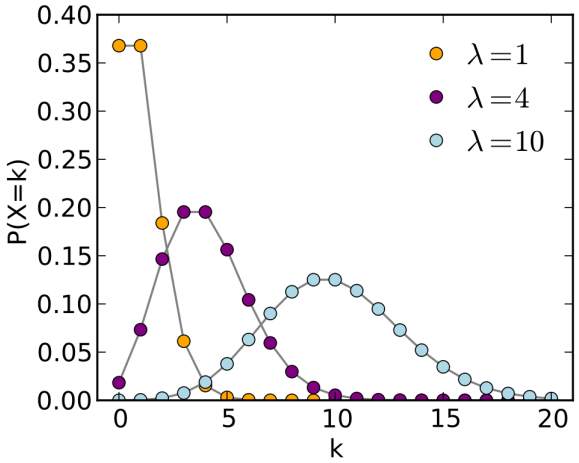


Figure 3.12: Poisson Distribution

Programs: Discrete Distribution Functions

Listing 3.2: Discrete

```
''' Different discrete distribution functions.
- Binomial distribution
- Poisson distribution (PMF, CDF, and PPF)

'''

'''
Author:  Thomas Haslwanter
Date:    Jan-2013
Version: 1.1
'''

# Note: here I use the modular approach, which is more appropriate for scripts
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np

bd1 = stats.binom(20, 0.5)
bd2 = stats.binom(20, 0.7)
bd3 = stats.binom(40, 0.5)
k = np.arange(40)
plt.plot(k, bd1.pmf(k), 'o-b')
plt.hold(True)
plt.plot(k, bd2.pmf(k), 'd-r')
plt.plot(k, bd3.pmf(k), 's-g')
plt.title('Binomial distribition')
plt.legend(['p=0.5 and n=20', 'p=0.7 and n=20', 'p=0.5 and n=40'])
plt.xlabel('X')
plt.ylabel('P(X)')
plt.show()

pd = stats.poisson(10)
plt.plot(k, pd.pmf(k),'x-')
plt.title('Poisson distribition - PMF')
plt.xlabel('X')
plt.ylabel('P(X)')
plt.show()

k = np.arange(30)
```

```
plt.plot(k, pd.cdf(k))
plt.title('Poisson distribition - CDF')
plt.xlabel('X')
plt.ylabel('P(X)')
plt.show()

y = np.linspace(0,1,100)
plt.plot(y, pd.ppf(y))
plt.title('Poisson distribition - PPF')
plt.xlabel('X')
plt.ylabel('P(X)')
plt.show()
```

3.3 Data Analysis

3.3.1 Data Screening

The first thing you have to do for your data analysis is simply *look at your data*. You should check for *missing data* in your data set, and *outliers* which can significantly influence the result of your analysis.

3.3.2 Normality Check

The first way to check if your data are normally distributed, i.e. that they are linearly related to the standard normal distribution. In statistics, *QQ plots* ("Q" stands for quantile) are used for visual assessments of distributions. They are a graphical method for comparing two probability distributions by plotting their quantiles against each other. First, the set of intervals for the quantiles are chosen. A point (x, y) on the plot corresponds to one of the quantiles of the second distribution (y-coordinate) plotted against the same quantile of the first distribution (x-coordinate). Thus the line is a parametric curve with the parameter which is the (number of the) interval for the quantile.

If the two distributions being compared are similar, the points in the $Q - Q$ plot will approximately lie on the line $y = x$. If the distributions are linearly related, the points in the $Q - Q$ plot will approximately lie on a line, but not necessarily on the line $y = x$ (Figure 3.13).

In addition, there are quantitative tests for normality. The test that I have encountered most frequently in recent literature is the *Kolmogorov-Smirnov test*. [1] Altman mainly uses the *Shapiro-Wilk W test* [Altman(1999)], and a number of other tests are also available.

3.3.3 Transformation

If your data deviate significantly from a normal distribution, it is sometimes possible to make the distribution approximately normal by transforming your data. For example, data often have values that can only be positive (e.g. the size of persons), and that have long positive tail: such data can often be made normal by applying a *log transform*. This is demonstrated in Figure 3.8.

3.3.4 Confidence Intervals

Although it is common to concentrate the analysis on the p-values, it is often much more informative to report the *confidence intervals* for your data. The confidence intervals are given by

$$ci = mean \pm se * t_{n,\alpha} \tag{3.11}$$

[1] see scipy.stats.kstest, example given in univariate.py

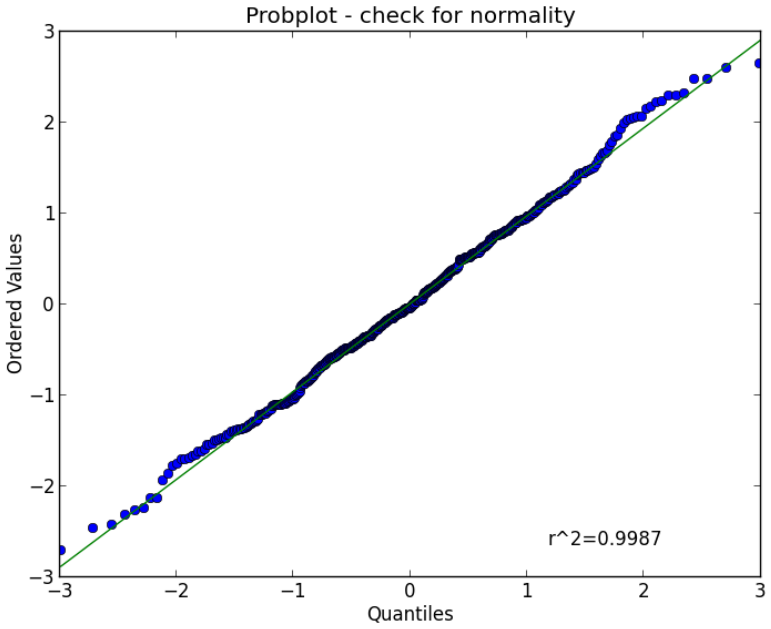


Figure 3.13: QQ-Plot, to check for normality of distribution.

where se is the standard error, and $t_{n,\alpha}$ the t statistic for n degrees of freedom. For the 95% two-sided confidence intervals, for example, you have to set $\alpha = 0.025$ and $\alpha = 0.975$.

Chapter 4

Statistical Tests

4.1 Hypothesis tests

Statistical evaluations are based on the initially often counterintuitive procedure of *hypothesis tests*. A hypothesis test is a standard format for assessing statistical evidence. It is ubiquitous in scientific literature, most often appearing in the form of statements of *statistical significance* and quotations like "$p < 0.01$" that pepper scientific journals. Thereby you proceed as follows: you

- state your hypothesis.
- decide which value you want to test your hypothesis on.
- calculate the *probability* p that you find the given value, assuming that your hypothesis is true

The first hypothesis is referred to as *null-hypothesis*, since we assume that there is *null* difference between the hypothesis and the result. The found probability for a specific target value is the *p-value* that you typically find in the literature. If $p < 0.05$, the difference between your sample and the value that you check is *significant*. If $p < 0.001$, we speak of a *highly significant* difference.

An example for a *null hypothesis*: "We assume that our population has a mean value of 7."

4.1.1 Types of Error

In hypothesis testing, two types of errors can occur:

Type I errors

These are errors, where you get a significant result despite the fact that the hypothesis is true. The likelihood of a Type I error is commonly indicated with α, and *is set before you start the data analysis*.

For example, assume that the population of young Austrian adults has a mean IQ of 105 (i.e. we are smarter than the rest) and a standard deviation of 15. We now want to check if the average FH student in Linz has the same IQ as the average Austrian, and we select 20 students. We set $\alpha = 0.05$, i.e. we set our significance level to 95%. Let us now assume that the average student has in fact the same IQ as the average Austrian. If we repeat our study 20 times, we will find one of those 20 times that our sample mean is significantly different from the Austrian average IQ. Such a finding would be a false result, despite the fact that our assumption is correct, and would constitute a *type I error*.

Type II errors and Test Power

If we want to answer the question "How much chance do we have to reject the null hypothesis when the alternative is in fact true?" Or in other words, "Whats the probability of detecting a real effect?" we are faced with a different problem. To answer these questions, we need an *alternative hypothesis*.

For the example given above, an *alternative hypothesis* could be: "We assume that our population has a mean value of 6."

A *Type II error* is an error, where you do *not* get a significant result, despite the fact that the null-hypothesis is false. The probability for this type of error is commonly indicated with β. The *power* of a statistical test is defined as $(1 - \beta) * 100$, and is the chance of correctly accepting the alternate hypothesis. Figure 4.1 shows the meaning of the *power* of a statistical test. Note that for finding the power of a test, you need an alternative hypothesis.

4.1.2 Sample Size

The power of a statistical test depends on four factors:

1. α, the probability for Type I errors
2. β, the probability for Type II errors ($\Rightarrow$ power of the test)
3. d, the magnitude of the investigated effect relative to σ, the standard deviation of the sample
4. n, the sample size

Only 3 of these 4 parameters can be chosen, the 4^{th} is then automatically fixed.

The size of the difference, d, between mean treatment outcomes that will answer the clinical question being posed is often called *clinical significance* or *clinical relevance*.

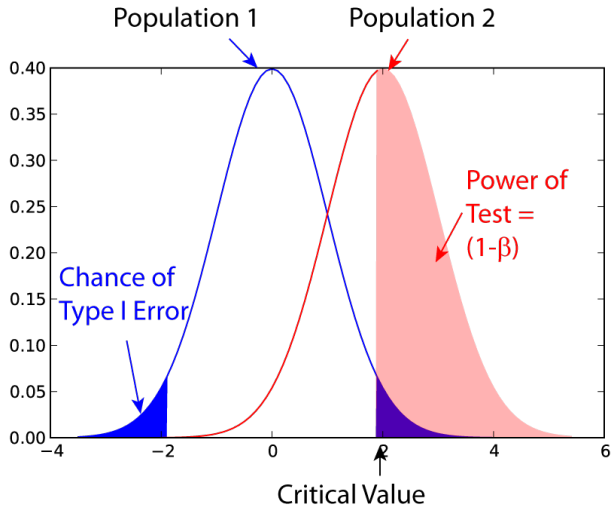


Figure 4.1: *Power* of a statistical test, for comparing the mean value of two given distributions.

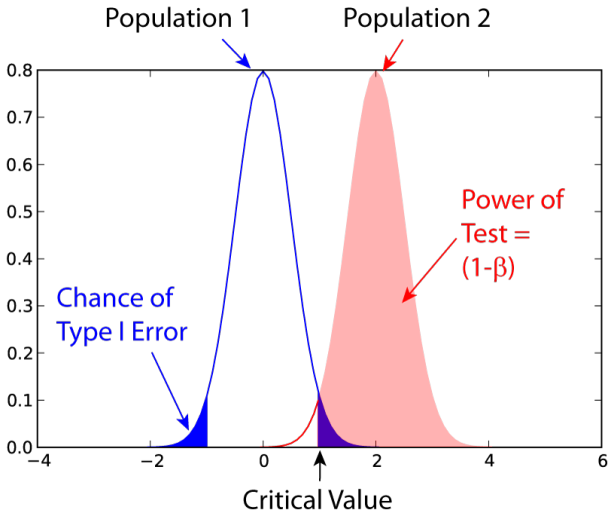


Figure 4.2: Effect of an increase in sampling size on the power of a test.

Examples for some special cases

For a test on one mean, this leads to a *minimum sample number* of

$$n = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2 \sigma^2}{d^2} \tag{4.1}$$

Here z is the standardized normal variable (see also chapter 3.2.2)

$$z = \frac{x - \mu}{\sigma}. \tag{4.2}$$

For finding a difference between two normally distributed means, the minimum number of samples we need in each group is

$$n_1 = n_2 = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2 (\sigma_1^2 + \sigma_2^2)}{d^2}. \tag{4.3}$$

Programs: SampleSize

Listing 4.1: sampleSize.py

```
'''Calculate the sample size for experiments, for normally distributed groups.

'''

'''
Author : Thomas Haslwanter
Date : Feb 2013
Ver : 1.0
'''

from scipy.stats import norm
from numpy import round
import numpy as np

def sampleSize_oneGroup(d, alpha=0.05, beta=0.2, sigma=1):
    '''Sample size for a single group.'''

    n = round((norm.ppf(1-alpha/2.) + norm.ppf(1-beta))**2 * sigma**2 / d**2)
```

```
        print('In order to detect a change of {0} in a group with an SD of {1},'.
            format(d, sigma))
        print('with significance {0} and test-power {1}, you need at least {2:d}
            subjects.'.format(alpha, 100*(1-beta), int(n)))

def sampleSize_twoGroups(d, alpha=0.05, beta=0.2, sigma1=1, sigma2=1):
    '''Sample size for two groups.'''

    n = round((norm.ppf(1-alpha/2.) + norm.ppf(1-beta))**2 * (sigma1**2 + sigma2
        **2) / d**2)

    print('In order to detect a change of {0} between groups with an SD of {1}
        and {2},'.format(d, sigma1, sigma2))
    print('with significance {0} and test-power {1}, you need in each group at
        least {2:d} subjects.'.format(alpha, 100*(1-beta), int(n)))

if __name__ == '__main__':
    sampleSize_oneGroup(0.5)
    print '\n'
    sampleSize_twoGroups(0.4, sigma1=0.6, sigma2=0.6)
```

4.2 Sensitivity and Specificity

Some of the more confusing terms in statistical analysis are *sensitivity* and *specificity* . A related topic are *positive predictive value (PPV)* and *negative predictive value (NPV)* . The following diagram shows how the four are related:

		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	**True Positive**	**False Positive** (Type I error)	**Positive predictive value** = Σ True Positive / Σ Test Outcome Positive
	Test Outcome Negative	**False Negative** (Type II error)	**True Negative**	Negative predictive value = Σ True Negative / Σ Test Outcome Negative
		Sensitivity = Σ True Positive / Σ Condition Positive	Specificity = Σ True Negative / Σ Condition Negative	

Figure 4.3: Relationship between sensitivity, specificity, positive predictive value and negative predictive value. (From: Wikipedia)

- **Sensitivity** = proportion of positives that are correctly identified by a test = probability of a positive test, given the patient is ill.
- **Specificity** = proportion of negatives that are correctly identified by a test = probability of a negative test, given that patient is well.
- **Positive predictive value** is the proportion of patients with positive test results who are correctly diagnosed.
- **Negative predictive value** is the proportion of patients with negative test results who are correctly diagnosed.

While sensitivity and specificity are independent of prevalence, they do not tell us what portion of patients with abnormal test results are truly abnormal. This information is provided by the positive/negative predictive value. However, as Fig. 4.4 indicates, these values are affected by the *prevalence* of the disease. In other words, we need to know the prevalence of the disease as well as the PPV/NPV of a test to provide a sensible interpretation of the test results.

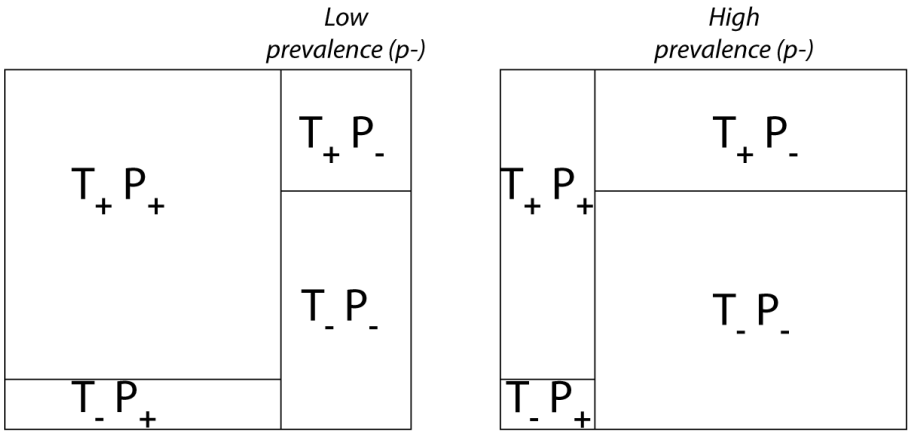


Figure 4.4: Effect of prevalence on PPV and NPV. "T" stands for "test", and "P" for "patient".

Figure 4.5 gives a worked example:

		Patients with bowel cancer (as confirmed on endoscopy)		
		Condition Positive	Condition Negative	
Fecal Occult Blood Screen Test Outcome	Test Outcome Positive	True Positive (TP) = 20	False Positive (FP) = 180	Positive predictive value = TP / (TP + FP) = 20 / (20 + 180) = **10%**
	Test Outcome Negative	False Negative (FN) = 10	True Negative (TN) = 1820	Negative predictive value = TN / (FN + TN) = 1820 / (10 + 1820) ≈ **99.5%**
		Sensitivity = TP / (TP + FN) = 20 / (20 + 10) ≈ **67%**	Specificity = TN / (FP + TN) = 1820 / (180 + 1820) = **91%**	

Figure 4.5: Worked example. (From: Wikipedia)

Related calculations

- False positive rate (α) = type I error = $1 - specificity = \frac{FP}{FP+TN} = \frac{180}{180+1820} = 9\%$
- False negative rate (β) = type II error = $1sensitivity = \frac{FN}{TP+FN} = \frac{10}{20+10} = 33\%$
- Power = sensitivity = 1β
- Likelihood ratio positive = $\frac{sensitivity}{1specificity} = \frac{66.67\%}{191\%} = 7.4$

Independent Variable		Dependent Variable			
		2 Categories	> 2 Categories	Ranked	Continuous
1 or 2 Categories	Not Matched (2 Samples)	Fisher's exact test	Fisher's exact test or χ^2 test	Mann-Whitney test	2-Sample t test
	Matched (Repeated) or Single Sample	Binomial(Poisson) test	χ^2 Goodness-of-fit test against uniform distribution	Wilcoxon signed-rank test	Paired t test
> 2 Categories	Not Matched	Fisher's exact test	Fisher's exact test or χ^2 test	Kruskal-Wallis test	One-way ANOVA
	Matched	Cochran's Q test	χ^2 Goodness-of-fit test against uniform distribution	Friedman test	One-way ANOVA
Ranked	Not Matched	Royston Ptrend test	Kruskal-Wallis test	Cusick Nptrend test	ANOVA trend test
	Matched	Cochran's Q test	χ^2 Goodness-of-fit test against uniform distribution	Page's L test	ANOVA trend test
	Continuous	Logistic regression	Multinomial regression	Ordinal regression	Common regression or correlation

Table 4.1: Typical tests for statistical problems.

- Likelihood ratio negative $= \frac{1 sensitivity}{specificity} = \frac{166.67\%}{91\%} = 0.37$

Hence with large numbers of false positives and few false negatives, a positive FOB screen test is in itself poor at confirming cancer (PPV = 10%) and further investigations must be undertaken; it did, however, correctly identify 66.7% of all cancers (the sensitivity). However as a screening test, a negative result is very good at reassuring that a patient does not have cancer (NPV = 99.5%) and at this initial screen correctly identifies 91% of those who do not have cancer (the specificity).

4.3 Large Sample Tests

Here I give an overview of the most common statistical tests for different combinations of data. This overview is taken from [Riffenburgh(2012)].

Chapter 5

Test of Means of Continuous Data

5.1 Distribution of a Sample Mean

5.1.1 One sample t-test for a mean value

If we knew the mean and the standard deviation of a normally distributed population, we would know exactly the standard error, and use values from the normal distribution to determine how likely it is to find a certain mean value, given the population mean and standard deviation. However, in practice we have to *estimate* the mean and standard deviation from the sample, and the resulting distribution for the mean value deviates slightly from a normal distribution. Such distributions are called *t-distributions*, and were first described by a researcher working under the pseudonym of "Student".

Let us look at a specific example: we take 100 normally distributed data, with a mean of 7 and with a standard deviation of 3. What is the chance of finding a mean value at a distance of 0.5 or more from the mean:?

Listing 5.1: ttest_1samp_notebook.py

```
# -*- coding: utf-8 -*-
# <nbformat>3.0</nbformat>

# <codecell>

import scipy.stats as stats
import numpy as np

# generate the data
np.random.seed(12345)
normDist = stats.norm(loc=7, scale=3)
data = normDist.rvs(100)
checkVal = 6.5

# t-test
t, tProb = stats.ttest_1samp(data, checkVal)

# Comparison with corresponding normal distribution
mmean = np.mean(data)
mstd = np.std(data, ddof=1)
normProb = stats.norm.cdf(6.5, loc=mmean, scale=mstd/np.sqrt(len(data)))*2

# compare
print('The probability from the t-test is ' + \
'{0:4.3f}, and from the normal distribution {1:4.3f}'.format(tProb, normProb))
```

>>> The probability from the t-test is 0.057, and from the normal distribution 0.054

Subject	Daily energy intake (kJ)	Difference from 7725 kJ	Ranks of differences
1	5260	2465	11
2	5470	2255	10
3	5640	2085	9
4	6180	1545	8
5	6390	1335	7
6	6515	1210	6
7	6805	920	4
8	7515	210	1.5
9	7515	210	1.5
10	8230	-505	3
11	8770	-1045	5

Table 5.1: Eaily energy intake of 11 healthy women with rank order of differences (ignoring their signs) from the recommended intake of 7725 kJ.

5.1.2 Wilcoxon signed rank sum test

If our data are not normally distributed, we cannot use the t-test (although this test is fairly robust against deviations from normality). Instead, we must use a *non-parametric* test on the mean value. We can do this by performing a *Wilcoxon signed rank sum test.* [1] [2] This method has three steps:

1. Calculate the difference between each observation and the value of interest.
2. Ignoring the signs of the differences, rank them in order of magnitude.
3. Calculate the sum of the ranks of all the negative (or positive) ranks, corresponding to the observations below (or above) the chosen hypothetical value.

In Table 5.1 you see an example, where the significance to a deviation from the value of 7725 is tested. The rank sum of the negative values gives $3 + 5 = 8$, and can be looked up in the corresponding tables to be significant. In practice, your computer program will nowadays do this for you. This example also shows another feature of rank evaluations: tied values (here 7515) get accorded their mean rank (here 1.5).

5.2 Comparison of Two Groups

When you compare two groups with each other, we have to distinguish between two cases. In the first case, we compare two values recorded from the same subject at two specific times. For example, we measure the size of students when they enter primary school and after their first year, and check if they have been growing. Since we are only interested in the *difference* between the first and the second measurement, this test is called *paired t-test*, and is essentially equivalent to a one-sample t-test for the mean difference.

The second test is if we compare two independent groups. For example, we can compare the effect of a two medications given to two different groups of patients, and compare how the two groups respond. This is called an *unpaired t-test*, or *t-test for two independent groups.*

[1] Python Example: scipy.stats.wilcoxon, in "univariate.py"

[2] The following description and example has been taken from [Altman(1999)], Table 9.2

If we have two independent samples the variance of the difference between their means is the *sum* of the separate variances, so the standard error of the difference in means is the square root of the sum of the separate variances:

$$\begin{aligned} se(\bar{x}_1 - \bar{x}_2) &= \sqrt{\text{var}(\bar{x}_1) + \text{var}(\bar{x}_2)} \\ &= \sqrt{\{se(\bar{x}_1)\}^2 + \{se(\bar{x}_2)\}^2} \\ &= \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \end{aligned}$$

where $\bar{x}_i$ is the mean of the i-th sample, and *se* indicates the *standard error*.

Listing 5.2: univariate.py

```
'''Univariate data analysis

This script shows how to
- Use different methods of the normal distribution
- Generate random data and plot a histogram
- Check for normality (Kolmogorov-Smirnov)
- Use a t-test for a single mean
- Use a non-parametric test (Wilcoxon signed rank) to check a single mean

'''

'''
Author: Thomas Haslwanter
Date:   Dec-2012
Version: 1.0
'''

from scipy.stats import norm
import scipy.stats as stats
import matplotlib.pyplot as plt
import numpy as np
from getdata import getData

def show_normal():
    '''Different aspects of the normal, Gaussian distribution.'''
    rv = norm()

    x = np.r_[-4:4:0.1]
    y = np.r_[0:1:0.001]

    ax = plt.subplot2grid((3,2),(0,0), colspan=2)
    #ax = plt.subplot(321)
    plt.plot(x,rv.pdf(x))
    plt.xlim([-4,4])
    plt.title('PDF')

    plt.subplot(323)
    plt.plot(x,rv.cdf(x))
    plt.xlim([-4,4])
    plt.title('CDF: cumulative distribution fct')

    plt.subplot(324)
    plt.plot(x,rv.sf(x))
    plt.xlim([-4,4])
    plt.title('SF: survival fct')
```

```
    plt.subplot(325)
    plt.plot(y,rv.ppf(y))
    plt.title('PPF')

    plt.subplot(326)
    plt.plot(y,rv.isf(y))
    plt.title('ISF')
    plt.tight_layout()
    plt.show()

def check_normality():
    '''Generate normally distributed data, and check normality them.'''

    # Generate and show a distribution
    plt.figure()
    myMean, mySD, numData = 50,10,500
    data = myMean + mySD*np.random.randn(numData)
    plt.hist(data)

    # Check for normality with Kolmogorov-Smirnov test
    _,pVal = stats.kstest((data-np.mean(data))/np.std(data,ddof=1), 'norm')
    if pVal > 0.05:
        print 'Data are probably normally distributed'

def check_mean():
    '''Data from Altman, check for significance of mean value.
    Compare average daily energy intake (kJ) over 10 days of 11 healthy women,
        and
    compare it to the recommended level of 7725 kJ.'''

    # Get data from Altman
    data = getData('altman_91.txt')

    # Watch out: by default the SD is calculated with 1/N!
    myMean = np.mean(data)
    mySD = np.std(data, ddof=1)
    print 'Mean and SD: {0:4.2f} and {1:4.2f}'.format(myMean, mySD)

    # Confidence intervals
    tf = stats.t(len(data)-1)
    ci = np.mean(data) + stats.sem(data)*np.array([-1,1])*tf.isf(0.025)
    print 'The confidence intervals are {0:4.2f} to {1:4.2f}.'.format(ci[0], ci
        [1])

    # Check for significance
    checkValue = 7725
    t, prob = stats.ttest_1samp(data, checkValue)
    if prob < 0.05:
        print '{0:4.2f} is significantly different from the mean (p={1:5.3f}).'.
            format(checkValue, prob)

    # For not normally distributed data, use the Wilcoxon signed rank test
    (rank, pVal) = stats.wilcoxon(data-checkValue)
    if pVal < 0.05:
        issignificant = 'unlikely'
    else:
        issignificant = 'likely'

    print 'It is ' + issignificant + ' that the value is {0:d}'.format(
        checkValue)

if __name__ == '__main__':
```

```
show_normal()
check_normality()
check_mean()
```

5.2.1 Non-parametric Comparison of Two Groups: Mann-Whitney Test

If the measurement values from the two groups are not normally distributed we have to resort to a non-parametric test. The most common test for that is the *Mann-Whitney(-Wilkoxon) test.*

5.3 Comparison of More Groups

5.3.1 Analysis of Variance

If we want to compare three or more groups with each other, we need to use a *one way analysis of variance (ANOVA)*, sometimes also called a *one factor ANOVA*. Because the null hypothesis is that there is no difference between the groups, the test is based on a comparison of the observed variation between the groups (i.e. between their means) with that expected from the observed variability between subjects. The comparison takes the general form of an *F test* to compare variances, but for two groups the t test leads to exactly the same answer. We will discuss ANOVAs in more detail in chapter 8.1.

F Test

The F test or *variance ratio test* is very simple. Under the null hypothesis that two Normally distributed populations have equal variances we expect the ratio of the two sample variances to have an *F distribution* (see section 3.2.3).

Bonferroni correction

If an ANOVA yields a significant result, we have to test which of the groups are different. Typically, this is done with $t-tests$. Since we perform multiple t tests, we should compensate for the risk of getting a significant result, even if our null hypothesis is true. The simplest - and at the same time quite conservative - approach is to divide the required p-value by the number of tests that we do (*Bonferroni correction*). For example, if you perform 4 comparisons, you check for significance not at $p = 0.05$, but at $p = 0.0125$.

While multiple testing is not yet included in Python standardly, you can get a number of multiple-testing corrections done with the statsmodels package:

```
  In[7]: from statsmodels.sandbox.stats.multicomp import multipletests
  In[8]: multipletests([.05, 0.3, 0.01], method='bonferroni')
Out[8]:
  (array([False, False,  True], dtype=bool),
  array([ 0.15,  0.9 ,  0.03]),
  0.016952427508441503,
  0.016666666666666666)
```

5.3.2 Kruskal-Wallis test

Just as analysis of variance is a more general form of t test, so there is a more general form of the non-parametric Mann-whitney test: the *Kruskal-Wallis test.* When the null hypothesis is true the test statistic follows the *Chi squared distribution.*

Listing 5.3: anovat.py

```
''' Explicit demonstration of properties of ANOVA
- For the comparison of two groups, a one-way ANOVA is equivalent to
  a T-test

.. math::
    t^2 = F

- Show how the ANOVA can be done by hand.
- Example of a Kruskal-Wallis test (for not normally distributed data)

'''

'''
Author: Thomas Haslwanter
Date:   Feb-2013
Ver:    0.4
'''

import os
import scipy as sp
import scipy.stats as stats
from numpy import array
import pandas
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Get the data
data = pandas.read_csv(r'.\data_kaplan\galton.csv')

# First, calculate the F- and the T-values, ...
F_statistic, pVal = stats.f_oneway(data['father'], data['mother'])
t_val, pVal_t = stats.ttest_ind(data['father'], data['mother'])

# ... and show that t**2 = F
print('From the t-test we get t^2={0:5.3f}, and from the F-test F={1:5.3f}'.
    format(t_val**2, F_statistic))

# ------------------------------------------------------------
# Second, do the ANOVA with a function ...
anova_results = anova_lm(ols('height ~ 1 + sex', data).fit())
print anova_results

# ... and then by hand, using the formulas fom Altman, p. 218
grouped = data.groupby('sex')
mdf = pandas.DataFrame({'male': grouped.get_group('M')['height'],
                        'female': grouped.get_group('F')['height']})
M = mdf.mean()
n = mdf.count()
S = sum((mdf**2).sum())
T = sum(mdf.sum())
B = sum(n*M**2)-T**2/sum(n)
W = S - sum(n*M**2)
N = sum(n)
Total = B+W


meanSq_group = B / (mdf.ndim-1)
meanSq_res = W / (N-2)
F = meanSq_group/meanSq_res
print('The hand-calculated F-value is: {0:5.3f}'.format(F))

# ------------------------------------------------------------
```

```
# And finally, give an example of the Kruskal-Wallis test
# Taken from http://www.brightstat.com/index.php?option=com_content&task=view&id
    =41&Itemid=1&limit=1&limitstart=2

# Get the data
city1 = array([68, 93, 123, 83, 108, 122])
city2 = array([119, 116, 101, 103, 113, 84])
city3 = array([70, 68, 54, 73, 81, 68])
city4 = array([61, 54, 59, 67, 59, 70])

# Perform the Kruskal-Wallis test
h, p = stats.mstats.kruskalwallis(city1, city2, city3, city4)

# Print the results
if p<0.05:
    print('There is a significant difference between the cities.')
else:
    print('No significant difference between the cities.')
```

Chapter 6

Tests on Categorical Data

In a sample of individuals the number falling into a particular group is called the *frequency*, so the analysis of categorical data is the analysis of frequencies. When two or more groups are compared the data are often shown in the form of a *frequency table*, sometimes also called *contingency table*.

6.1 One Proportion

If you have one sample group of data, you can check if your sample is representative of the standard population. To do so, you have to know the proportion p of the characteristic in the standard population. It can be shown that a in population with a characteristic with probability p, the standard error of samples with this characteristic is given by

$$se(p) = \sqrt{p(1-p)/n} \tag{6.1}$$

and the corresponding 95% confidence interval is

$$ci = mean \pm se * t_{n,0.95}$$

If your data lie outside this confidence interval, they are *not* representative of the population.

6.2 Frequency Tables

6.2.1 Chi-square Test

Assume you have observed absolute frequencies o_i and expected absolute frequencies e_i under the Null hypothesis of your test then it holds

$$V = \sum_i \frac{(o_i - e_i)^2}{e_i} \approx \chi^2_f \tag{6.2}$$

.

where f are the degrees of freedom. i might denote a simple index running from $1, ..., I$ or even a multiindex $(i_1, ..., i_p)$ running from $(1, ..., 1)$ to $(I_1, ..., I_p)$.

The test statistic V is approximately χ^2 distributed, if

	Right Handed	*Left Handed*	*Total*
Males	43	9	52
Females	44	4	48
Total	87	13	100

Table 6.1: Example of a frequency table

- for all absolute expected frequencies e_i holds $e_i \geq 1$ and
- for at least 80% of the absolute expected frequencies e_i holds $e_i \geq 5$.

The degrees of freedom can be computed by the numbers of absolute observed frequencies which can be chosen freely. We know that the sum of absolute expected frequencies is

$$\sum_i o_i = n \tag{6.3}$$

which means that the maximum number of degrees of freedom is $I - 1$. We might have to subtract from the number of degrees of freedom the number of parameters we need to estimate from the sample, since this implies further relationships between the observed frequencies.

Example

The χ^2 test can be used to generate "quick and dirty" test, e.g.

H_0 : The random variable X is symmetrically distributed versus

H_1 : the random variable X is not symmetrically distributed.

We know that in case of a symmetrical distribution the arithmetic mean $\bar{x}$ and median should be nearly the same. So a simple way to test this hypothesis would be to count how many observations are less than the mean $(n_-)$and how many observations are larger than the arithmetic mean (n_+). If mean and median are the same than 50% of the observation should smaller than the mean and 50% should be larger than the mean. It holds

$$V = \frac{(n_- - n/2)^2}{n/2} + \frac{(n_+ - n/2)^2}{n/2} \approx \chi_1^2 \tag{6.4}$$

.

Comments

The Chi-square test is a pure hypothesis test. It tells you if your observed frequency can be due to a random sample selection from a single population. A number of different expressions have been used for chi-square tests, which are due to the original derivation of the formulas (from the time before computers were pervasive). Expression such as *2x2 tables*, *r-c tables*, or *Chi-square test of contingency* all refer to frequency tables and are typically analyzed with chi-square tests.

6.2.2 Fisher's Exact Test

For small sample numbers, corrections should be made for some bias that is caused by the use of the continuous chi-squared distribution. This correction is referred to as *Yates correction*.

If the requirement that 80% of cells should have expected values of at least 5 is not fulfilled, *Fisher's exact test* should be used. This test is based on the observed row and column totals. The method consists of evaluating the probability associated with all possible 2x2 tables which have the same row and column totals as the observed data, making the assumption that the null hypothesis (i.e. that the row and column variables are unrelated) is true. In most cases, Fisher's exact test is preferable to the chi-square test. But until the advent of powerful computers, it was not practical. You should use it up to approximately 10-15 cells in the frequency tables.

6.3 Analysis Programs

With computers, the computational steps are trivial:

Listing 6.1: compGroups.py

```
''' Analysis of categorical data
- Analysis of one proportion
- Chi-square test
- Fisher exact test

'''

'''
Author:  Thomas Haslwanter
Date:    Jan-2013
Version: 1.0
'''

import numpy as np
import scipy.stats as stats


def oneProportion():
    '''Calculate the confidence intervals of the population, based on a
    given data sample.
    The data are taken from Altman, chapter 10.2.1.'''

    # Get the data
    numTotal = 215
    numPositive = 39

    # Calculate the confidence intervals
    p = float(numPositive)/numTotal
    se = np.sqrt(p*(1-p)/numTotal)
    td = stats.t(numTotal-1)
    ci = p + np.array([-1,1])*td.isf(0.025)*se

    # Print them
    print('ONE PROPORTION')
    print('The confidence interval for the given sample is {0:5.3f} to {1:5.3f}'
        .format(
        ci[0], ci[1]))
    return

def chiSquare():
    ''' Application of a chi square test to a 2x2 table.
    The calculations are done with and without Yate's continuity
    correction.
    Data are taken from Altman, Table 10.10:
    Comparison of number of hours' swimming by swimmers with or without erosion
        of dental enamel.
    >= 6h: 32 yes, 118 no
    <  6h: 17 yes, 127 no'''

    # Enter the data
    obs = np.array([[32, 118], [17, 127]])

    # Calculate the chi-square test
    chi2_corrected = stats.chi2_contingency(obs, correction=True)
    chi2_uncorrected = stats.chi2_contingency(obs, correction=False)

    # Print the result
    print('CHI SQUARE')
    print('The corrected chi2 value is {0:5.3f}, with p={1:5.3f}'.format(
        chi2_corrected[0], chi2_corrected[1]))
    print('The uncorrected chi2 value is {0:5.3f}, with p={1:5.3f}'.format(
        chi2_uncorrected[0], chi2_uncorrected[1]))
```

```
        return

    def fisherExact():
        '''Fisher's Exact Test:
        Data are taken from Altman, Table 10.14
        Spectacle wearing among juvenile delinquensts and non-delinquents who failed
             a vision test
        Spectecle wearers: 1 delinquent, 5 non-delinquents
        non-spectacle wearers: 8 delinquents, 2 non-delinquents'''

        # Enter the data
        obs = np.array([[1,5], [8,2]])

        # Calculate the Fisher Exact Test
        fisher_result = stats.fisher_exact(obs)

        # Print the result
        print('FISHER')
        print('The probability of obtaining a distribution at least as extreme '
        + 'as the one that was actually observed, assuming that the null ' +
        'hypothesis is true, is: {0:5.3f}.'.format(fisher_result[1]))

    if __name__ == '__main__':
        oneProportion()
        chiSquare()
        fisherExact()
```

Chapter 7

Relation Between Two Continuous Variables

If we have two related variables, the *correlation* measures the association between the two variables. In contrast, a *linear regression* is used for the prediction of the value of one variable from another. If we want to compare more than two groups of variables, we have to use a technique known as *Analysis of Variance (ANOVA)*.

7.1 Correlation

7.1.1 Correlation Coefficient

If the two variables are normally distributed, the standard measure of determining the *correlation coefficient*, often ascribed to *Pearson* , is

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{7.1}$$

Pearson's correlation coefficient, sometimes also referred to as *population correlation coefficient*, can take any value from -1 to +1. Examples are given in Figure 7.1. Note that the formula for the correlation coefficient is symmetrical between x and y.

7.1.2 Rank correlation

If the data distribution is not normal, a different approach is necessary. In that case one can rank the set of subjects for each variable and compare the orderings. There are two commonly used methods of calculating the rank correlation.

- *Spearman's* ρ, which is exactly the same as the Pearson correlation coefficient r calculated on the ranks of the observations.
- *Kendall's* τ.

7.2 Regression

We can use the method of *regression* when we want to predict the value of one variable from the other.

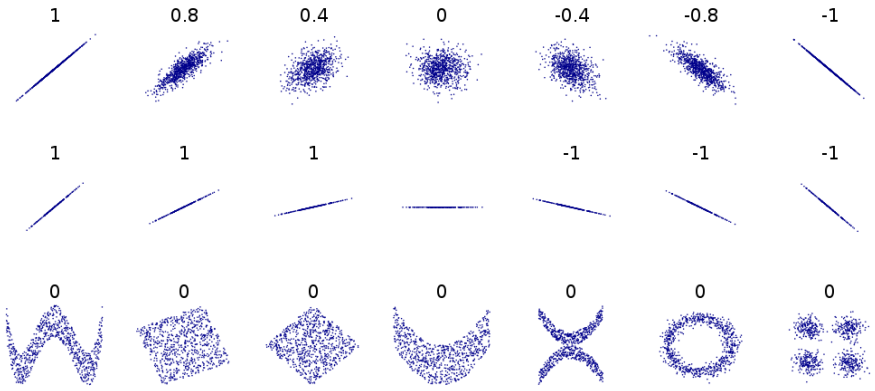


Figure 7.1: Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Note that the correlation reflects the non-linearity and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). N.B.: the figure in the center has a slope of 0 but in that case the correlation coefficient is undefined because the variance of Y is zero. (From: Wikipedia)

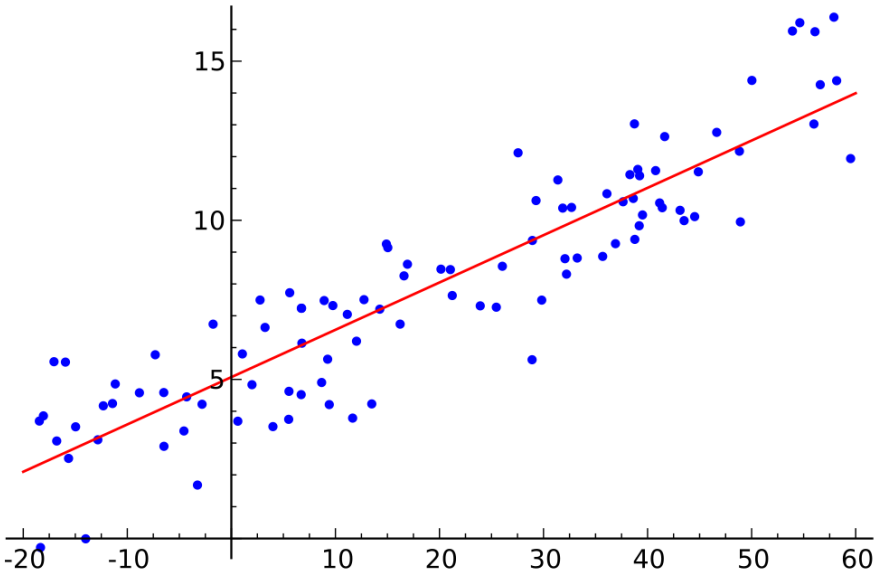


Figure 7.2: Linear regression. (From Wikipedia)

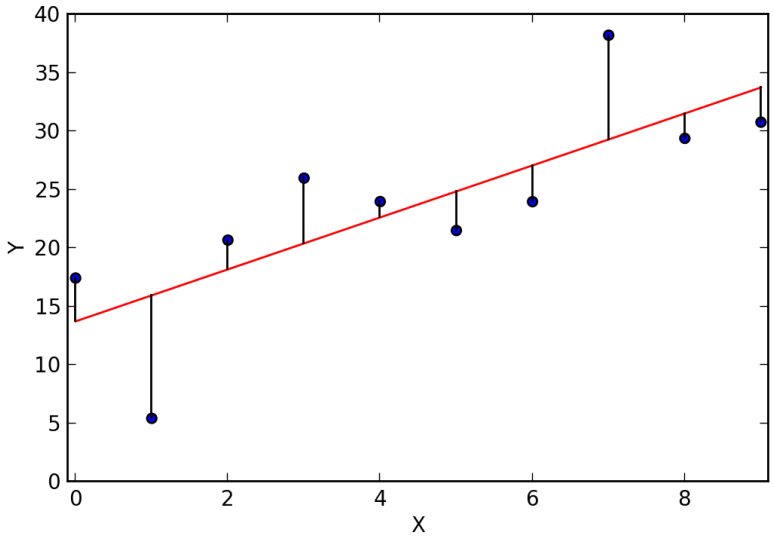


Figure 7.3: Best-fit linear regression line (red) and residuals (black).

When we search for the best-fit line to a given (x_i, y_i) dataset, we are looking for the parameters (k, d) which minimize the sum of the squared *residuals* ϵ_i in

$$y_i = k * x_i + d + \epsilon_i \tag{7.2}$$

where k is the *slope* or *inclination* of the line, and d the *intercept*. This is in fact just the one-dimensional example of the more general technique, which is described in the next section. Note that in contrast to the correlation, this relationship between x and y is no more symmetrical: it is assumed that the $x-$values are known exactly, and that all the variability lies in the residuals.

7.2.1 Introduction

[1] Given a data set $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^n$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y_i and the p-vector of regressors x_i is linear. This relationship is modelled through a *disturbance term* or *error variable* ϵ_i, an unobserved random variable that adds noise to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n, \tag{7.3}$$

where ^{T} denotes the transpose, so that $x_i^T \beta$ is the inner product between vectors x_i β.
Often these n equations are stacked together and written in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{7.4}$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}. \tag{7.5}$$

[1] This section has been taken from Wikipedia

Some remarks on terminology and general use:

- y_i is called the *regressand*, *endogenous variable*, *response variable*, *measured variable*, or *dependent variable*. The decision as to which variable in a data set is modeled as the dependent variable and which are modeled as the independent variables may be based on a presumption that the value of one of the variables is caused by, or directly influenced by the other variables. Alternatively, there may be an operational reason to model one of the variables in terms of the others, in which case there need be no presumption of causality.

- $\mathbf{x}_i$ are called *regressors*, *exogenous variables*, *explanatory variables*, *covariates*, *input variables*, *predictor variables*, or *independent variables*, but not to be confused with *independent random variables*. The matrix $\mathbf{X}$ is sometimes called the *design matrix*.

  - Usually a constant is included as one of the regressors. For example we can take $x_{i1} = 1$ for $i = 1, ..., n$. The corresponding element of β is called the *intercept*. Many statistical inference procedures for linear models require an intercept to be present, so it is often included even if theoretical considerations suggest that its value should be zero.
  - Sometimes one of the regressors can be a non-linear function of another regressor or of the data, as in polynomial regression and segmented regression. The model remains linear as long as it is linear in the parameter vector β.
  - The regressors x_{ij} may be viewed either as random variables, which we simply observe, or they can be considered as predetermined fixed values which we can choose. Both interpretations may be appropriate in different cases, and they generally lead to the same estimation procedures; however different approaches to asymptotic analysis are used in these two situations.

- $\boldsymbol{\beta}$ is a p-dimensional *parameter vector*. Its elements are also called *effects*, or *regression coefficients*. Statistical estimation and inference in linear regression focuses on β.

- ε_i is called the *error term*, *disturbance term*, or *noise*. This variable captures all other factors which influence the dependent variable y_i other than the regressors x_i. The relationship between the error term and the regressors, for example whether they are correlated, is a crucial step in formulating a linear regression model, as it will determine the method to use for estimation.

- If $i = 1$ and $p = 1$ in Eq.7.3, we have a *simple linear regression*, corresponding to Eq.7.2. If $i > 1$ we talk about *multilinear regression* or *multiple linear regression* .

Example. Consider a situation where a small ball is being tossed up in the air and then we measure its heights of ascent h_i at various moments in time t_i. Physics tells us that, ignoring the drag, the relationship can be modelled as :

$$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i, \tag{7.6}$$

where β_1 determines the initial velocity of the ball, β_2 is proportional to the standard gravity, and ϵ_i is due to measurement errors. Linear regression can be used to estimate the values of β_1 and β_2 from the measured data. This model is non-linear in the time variable, but it is linear in the parameters β_1 and β_2; if we take regressors $\mathbf{x}_i = (x_{i1}, x_{i2}) = (t_i, t_i^2)$, the model takes on the standard form : $h_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i$.

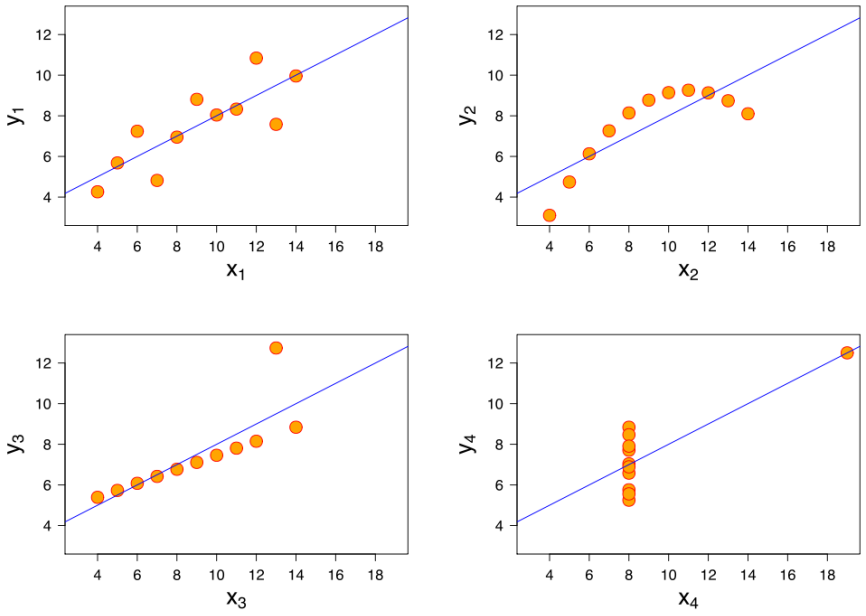


Figure 7.4: The sets in the *Anscombe's quartet* have the same linear regression line but are themselves very different.

7.2.2 Assumptions

To use the technique of linear regression, five assumptions should be fulfilled:

- The errors in the data values (i.e. the deviations from average) are independent from one another.
- The model must be appropriate. (A linear regression does not properly describe a quadratic curve.)
- The *independent variables* (i.e. x) are exactly known.
- The variance of the *dependent variable* (i.e. y) is the same for all values of x.
- The distribution of y is approximately normal for all values of x.

Listing 7.1: multivariate.py

```
''' Analysis of multivariate data
- Analysis of paired data
- Analysis of unpaired data
- Regression line
- Correlation

'''

'''
Author:  Thomas Haslwanter
Date:    Jan-2013
Version: 1.3
'''

from numpy import genfromtxt, mean, std
import scipy.stats as stats
```

```
import matplotlib.pyplot as plt
import pandas as pd
from getdata import getData

def paired_data():
    '''Analysis of paired data
    Compare mean daily intake over 10 pre-menstrual and 10 post-menstrual days (
        in kJ).'''

    # Get the data:  daily intake of energy in kJ for 11 women
    data = getData('altman_93.txt')

    mean(data, axis=0)
    std(data, axis=0, ddof=1)

    pre = data[:,0]
    post = data[:,1]

    # paired t-test: doing two measurments on the same experimental unit
    # e.g., before and after a treatment
    t_statistic, p_value = stats.ttest_1samp(post - pre, 0)

    # p < 0.05 => alternative hypothesis:
    # the difference in mean is not equal to 0
    print("paired t-test", p_value)

    # alternative to paired t-test when data has an ordinary scale or when not
    # normally distributed
    z_statistic, p_value = stats.wilcoxon(post - pre)
    print("paired wilcoxon-test", p_value)


def unpaired_data():
    ''' Then some unpaired comparison: 24 hour total energy expenditure (MJ/day)
        ,
    in groups of lean and obese women'''

    # Get the data: energy expenditure in mJ and stature (0=obese, 1=lean)
    energ = getData('altman_94.txt')

    # Group them
    group1 = energ[:, 1] == 0
    group1 = energ[group1][:, 0]
    group2 = energ[:, 1] == 1
    group2 = energ[group2][:, 0]

    mean(group1)
    mean(group2)

    # two-sample t-test
    # null hypothesis: the two groups have the same mean
    # this test assumes the two groups have the same variance...
    # (can be checked with tests for equal variance)
    # independent groups: e.g., how boys and girls fare at an exam
    # dependent groups: e.g., how the same class fare at 2 different exams
    t_statistic, p_value = stats.ttest_ind(group1, group2)

    # p_value < 0.05 => alternative hypothesis:
    # they don't have the same mean at the 5% significance level
    print("two-sample t-test", p_value)

    # For non-normally distributed data, perform the two-sample wilcoxon test
    # a.k.a Mann Whitney U
```

```
    u, p_value = stats.mannwhitneyu(group1, group2)
    print("two-sample wilcoxon-test", p_value)

    # Plot the data
    plt.plot(group1, 'bx', label='obese')
    plt.hold(True)
    plt.plot(group2, 'ro', label='lean')
    plt.legend(loc=0)
    plt.show()

def regression_line():
    '''Fit a line, using the powerful "ordinary least square" method of pandas
        '''

    # Get the data
    data = getData('altman_11_6.txt')

    df = pd.DataFrame(data, columns=['glucose', 'Vcf'])
    model = pd.ols(y=df['Vcf'], x=df['glucose'])
    print model.summary

def correlation():
    '''Pearson correlation, and two types of rank correlation (Spearman, Kendall
        )
    Data from 24 type 1 diabetic patients, relating Fasting blood glucose (mmol/
        l) to
    mean circumferential shortening velocity (%/sec).'''

    # Get the data
    data = getData('altman_11_1.txt')

    # Bring them into the dataframe-format
    df = pd.DataFrame(data, columns=['age', 'fat'])

    # Calculate correlations
    corr = {}
    corr['pearson'] = df['age'].corr(df['fat'], method = 'pearson')
    corr['spearman'] = df['age'].corr(df['fat'], method = 'spearman')
    corr['kendall'] = df['age'].corr(df['fat'], method = 'kendall')

    print(corr)

if __name__ == '__main__':
    paired_data()
    unpaired_data()
    regression_line()
    correlation()
```

Since to my knowledge there exists no program in the Python standard library (or numpy, scipy) to calculate the confidence intervals for a regression line, I include my corresponding program *lineFit.py* 7.2. The output of this program is shown in Figure 7.5. This program also shows how Python programs intended for distribution should be documented.

Listing 7.2: fitLine.py

```
'''
Linear regression fit

Parameters
----------
x : ndarray
    Input / Predictor
y : ndarray
```

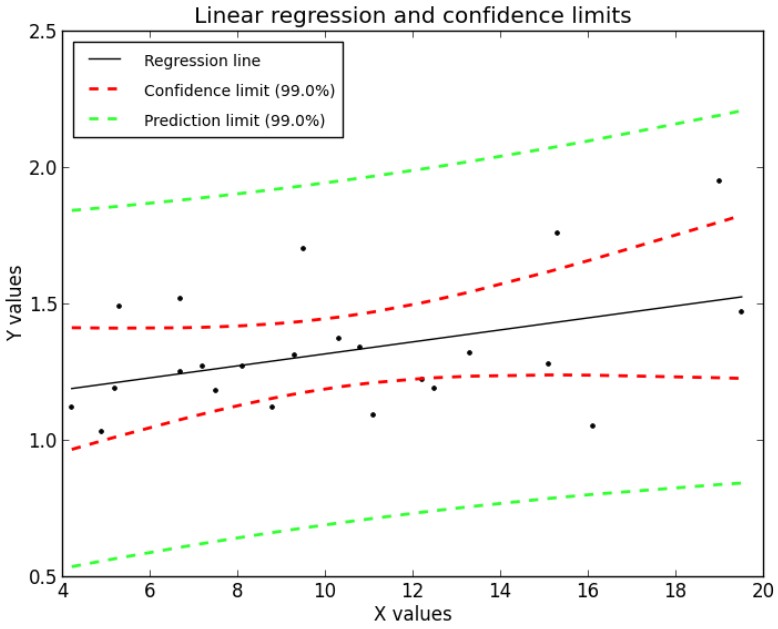


Figure 7.5: Regression, with confidence intervals for the mean, as well as for the predicted data.

```
    Input / Estimator
alpha : float
    Confidence limit [default=0.05]
newx : float or ndarray
    Values for which the fit and the prediction limits are calculated (optional)
plotFlag: int (optional)
    1 = plot, 0 = no_plot [default]

Returns
-------
a : float
    Intercept
b : float
    Slope
ci : ndarray
    Lower and upper confidence interval for the slope
info : dictionary, containing return information on
    - residuals
    - var_res
    - sd_res
    - alpha
    - tval
    - df
newy : list(ndarray)
    Predictions for (newx, newx-ciPrediction, newx+ciPrediction)

Examples
--------
>>> import numpy as np
>>> from fitLine import fitLine
>>> x = np.r_[0:10:11j]
>>> y = x**2
>>> (a,b,(ci_a, ci_b),_)=fitLine(x,y)

Notes
-----
```

```
    Example data and formulas are taken from
    D. Altman, "Practical Statistics for Medicine"
    '''

    '''
    author : thomas haslwanter
    date : 13.Dec.2012
    ver : 1.2
    '''

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

    def fitLine(x, y, alpha=0.05, newx=[], plotFlag=1):
        ''' Fit a curve to the data using a least squares 1st order polynomial fit
            '''

        # Summary data
        n = len(x)          # number of samples

        Sxx = np.sum(x**2) - np.sum(x)**2/n
    #    Syy = np.sum(y**2) - np.sum(y)**2/n    # not needed here
        Sxy = np.sum(x*y) - np.sum(x)*np.sum(y)/n
        mean_x = np.mean(x)
        mean_y = np.mean(y)

        # Linefit
        b = Sxy/Sxx
        a = mean_y - b*mean_x

        # Residuals
        fit = lambda xx: a + b*xx
        residuals = y - fit(x)

        var_res = np.sum(residuals**2)/(n-2)
        sd_res = np.sqrt(var_res)

        # Confidence intervals
        se_b = sd_res/np.sqrt(Sxx)
        se_a = sd_res*np.sqrt(np.sum(x**2)/(n*Sxx))

        df = n-2                            # degrees of freedom
        tval = stats.t.isf(alpha/2., df)  # appropriate t value

        ci_a = a + tval*se_a*np.array([-1,1])
        ci_b = b + tval*se_b*np.array([-1,1])

        # create series of new test x-values to predict for
        npts = 100
        px = np.linspace(np.min(x),np.max(x),num=npts)

        se_fit     = lambda x: sd_res * np.sqrt(  1./n + (x-mean_x)**2/Sxx)
        se_predict = lambda x: sd_res * np.sqrt(1+1./n + (x-mean_x)**2/Sxx)

        print 'Summary: a={0:5.4f}+/-{1:5.4f}, b={2:5.4f}+/-{3:5.4f}'.format(a,tval*
            se_a,b,tval*se_b)
        print 'Confidence intervals: ci_a=({0:5.4f} - {1:5.4f}), ci_b=({2:5.4f} -
            {3:5.4f})'.format(ci_a[0], ci_a[1], ci_b[0], ci_b[1])
        print 'Residuals: variance = {0:5.4f}, standard deviation = {1:5.4f}'.format
            (var_res, sd_res)
        print 'alpha = {0:.3f}, tval = {1:5.4f}, df={2:d}'.format(alpha, tval, df)
```

```
    # Return info
    ri = {'residuals': residuals,
        'var_res': var_res,
        'sd_res': sd_res,
        'alpha': alpha,
        'tval': tval,
        'df': df}

    if plotFlag == 1:
        # Plot the data
        plt.figure()

        plt.plot(px, fit(px),'k', label='Regression line')
        #plt.plot(x,y,'k.', label='Sample observations', ms=10)
        plt.plot(x,y,'k.')

        x.sort()
        limit = (1-alpha)*100
        plt.plot(x, fit(x)+tval*se_fit(x), 'r--', lw=2, label='Confidence limit
            ({0:.1f}%)'.format(limit))
        plt.plot(x, fit(x)-tval*se_fit(x), 'r--', lw=2 )

        plt.plot(x, fit(x)+tval*se_predict(x), '--', lw=2, color=(0.2,1,0.2),
            label='Prediction limit ({0:.1f}%)'.format(limit))
        plt.plot(x, fit(x)-tval*se_predict(x), '--', lw=2, color=(0.2,1,0.2))

        plt.xlabel('X values')
        plt.ylabel('Y values')
        plt.title('Linear regression and confidence limits')

        # configure legend
        plt.legend(loc=0)
        leg = plt.gca().get_legend()
        ltext = leg.get_texts()
        plt.setp(ltext, fontsize=10)

        # show the plot
        plt.show()

    if newx != []:
        try:
            newx.size
        except AttributeError:
            newx = np.array([newx])

        print 'Example: x = {0}+/-{1} => se_fit = {2:5.4f}, se_predict = {3:6.5f
            }'\
        .format(newx[0], tval*se_predict(newx[0]), se_fit(newx[0]), se_predict(
            newx[0]))

        newy = (fit(newx), fit(newx)-se_predict(newx), fit(newx)+se_predict(newx
            ))
        return (a,b,(ci_a, ci_b), ri, newy)
    else:
        return (a,b,(ci_a, ci_b), ri)

if __name__ == '__main__':
        # example data
        x = np.array([15.3, 10.8, 8.1, 19.5, 7.2, 5.3, 9.3, 11.1, 7.5, 12.2,
                      6.7, 5.2, 19.0, 15.1, 6.7, 8.6, 4.2, 10.3, 12.5, 16.1,
                      13.3, 4.9, 8.8, 9.5])
        y = np.array([1.76, 1.34, 1.27, 1.47, 1.27, 1.49, 1.31, 1.09, 1.18,
                      1.22, 1.25, 1.19, 1.95, 1.28, 1.52, np.nan, 1.12, 1.37,
```

```
                    1.19, 1.05, 1.32, 1.03, 1.12, 1.70])

goodIndex = np.invert(np.logical_or(np.isnan(x), np.isnan(y)))
(a,b,(ci_a, ci_b), ri,newy) = fitLine(x[goodIndex],y[goodIndex], alpha
    =0.01,newx=np.array([1,4.5]))
```

Chapter 8

Relation Between Several Variables

When we have two groups, we can ask the question: "Are they different?" The answer is provided by hypothesis tests: by a *t-test* if the data are normally distributed, or by a *Mann-Whitney test* otherwise. If we want to go one step further and predict the value of one variable from another, we have to use the technique of *linear regression*.

So what happens when we have more than two groups?

To answer the question "Are they different?" for more than two groups, we have to use the *Analysis of Variance (ANOVA)-test* for data where the residuals are normally distributed. If this condition is not fulfilled, the *Friedmann Test* has to be used. And if we want to and predict the value of one variable *many* other variables, linear regression has to be replaced by of *multilinear regression* , sometimes also referred to as *multiple linear regression*.

8.1 Variance Analysis

The idea behind ANOVA is to divide the variance into the variance *between* groups, and that *within* groups, and see if those distributions match the null hypothesis that all groups come from the same distribution. The variables that distinguish the different groups are often called *factors*.

(By comparison, t-tests look at the mean values of two groups, and check if those are consistent with the assumption that the two groups come from the same distribution.)

For example, if we compare a group with No treatment, another with treatment A, and a third with treatment B, then we perform a *one factor ANOVA*, sometimes also called *one-way ANOVA*, with "treatment" the one analysis factor. If we do the same test with men and with women, then we have a *two-factor* or *two-way ANOVA*, with "gender" and "treatment" as the two treatment factors. Note that with ANOVAs, it is quite important to have exactly the same number of samples in each analysis group!

The one-way ANOVA assumes all the samples are drawn from normally distributed populations with equal variance. To test this assumption, you can use the *Levene test*.

Compared to one-way ANOVAs, the analysis with two-way ANOVAs has a new element. We can look not only if each of the factors is significant; we can also check if the *interaction* of the factors has a significant influence on the distribution of the data. For sticking to the example above, if only women with treatment B get healthy, we have a significant interaction effect between "gender" and "treatment".

8.1.1 Example: one-way ANOVA

As an example, let us take the red cell folate levels ($\mu g/l$) in threee groups of cardiac bypass patients given different levels of nitrous oxide ventilation (Amess et al, 1978):

- First the "Sums of squares (SS)" are calculated. Here the SS between treatments is 15515.88, and the SS of the residuals is 39716.09 . The total SS is the sum of these two values.
- The mean squares is the SS divided by the corresponding degrees of freedom.
- The F-value is the larger mean squares value divided by the smaller value. (If we only have two groups, the F-value is the square of the corresponding t-value. See listing 7.1).
- From the F-value, we can looking up the corresponding p-value.

8.1.2 Example: two-way ANOVA

See the example in listing 9.1

Listing 8.1: anova.py

```
''' Analysis of Variance (ANOVA)
- Levene test
- ANOVA - oneway
- ANOVA - twoway

'''

'''
Author:  Thomas Haslwanter
Date:    March-2013
Version: 1.1
'''

import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
from getdata import getData
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

def anova_oneway():
    ''' One-way ANOVA: test if results from 3 groups are equal. '''

    # Get the data
    data = getData('altman_910.txt')

    # Sort them into groups, according to column 1
    group1 = data[data[:,1]==1,0]
    group2 = data[data[:,1]==2,0]
    group3 = data[data[:,1]==3,0]

    # First, check if the variances are equal, with the "Levene"-test
    (W,p) = stats.levene(group1, group2, group3)
    if p<0.05:
        print('Warning: the p-value of the Levene test is <0.05: p={0}'.format(p
            ))

    # Do the one-way ANOVA
    F_statistic, pVal = stats.f_oneway(group1, group2, group3)

    # Print the results
    print 'Altman 910:'
    print (F_statistic, pVal)
    if pVal < 0.05:
        print('One of the groups is significantly different.')
```

```
    # Elegant alternative implementation, with pandas & statsmodels
    df = pd.DataFrame(data, columns=['value', 'treatment'])
    model = ols('value ~ C(treatment)', df).fit()
    print anova_lm(model)

def anova_interaction():
    '''ANOVA with interaction: Measurement of fetal head circumference,
    by four observers in three fetuses.'''

    # Get the data
    data = getData('altman_12_6.txt')

    # Bring them in dataframe-format
    df = pd.DataFrame(data, columns=['hs', 'fetus', 'observer'])

    # Determine the ANOVA with interaction
    formula = 'hs ~ C(fetus) + C(observer) + C(fetus):C(observer)'
    lm = ols(formula, df).fit()
    print anova_lm(lm)

if __name__ == '__main__':
    anova_oneway()
    anova_interaction()
```

8.2 Multilinear Regression

If you have truly independent variables, *multilinear regression* is a straitforward extension of the simple linear regression. However, if your variables may be related to each other, you have to proceed much more carefully. For example, you may want to investigate how the prevalence of some disease correlates with age and with income: if you do so, you have to keep in mind that age and income are most likely correlated! For details, [Kaplan(2009)] gives a good introduction to that topic. Also, check out the chapter on Modeling.

Listing 8.2: mult_regress.py

```
'''Multiple Regression
Shows how to calculate just the best fit, or - using "statsmodels" - all the
corresponding statistical parameters.
Also shows how to make 3d plots.

'''

'''
Author: Thomas Haslwanter
Date:   March-2013
Ver:    1.0
'''

# The standard imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# For the 3d plot
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# For the statistic
from statsmodels.formula.api import ols
```

```
def generatedata():
    ''' Generate and show the data '''
    x = np.linspace(-5,5,101)
    (X,Y) = np.meshgrid(x,x)
    Z = -5 + 3*X-0.5*Y+np.random.randn(np.shape(X)[0], np.shape(X)[1])

    # Plot the figure
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X,Y,Z, cmap=cm.coolwarm)
    ax.view_init(20,-120)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    fig.colorbar(surf, shrink=0.6)
    plt.show()

    return (X.flatten(),Y.flatten(),Z.flatten())

def regressionmodel(X,Y,Z):
    '''Multilinear regression model, calculating fit, P-values, confidence
       intervals etc.'''

    # Convert the data into a Pandas DataFrame
    df = pd.DataFrame({'x':X, 'y':Y, 'z':Z})

    # Fit the model
    model = ols("z ~ x + y", df).fit()

    # Print the summary
    print model.summary()

def linearmodel(X,Y,Z):
    '''Just fit the plane'''

    M = np.vstack((np.ones(len(X)), X, Y)).T
    bestfit = np.linalg.lstsq(M,Z)
    print 'Best fit plane:', bestfit

if __name__ == '__main__':
    (X,Y,Z) = generatedata()
    regressionmodel(X,Y,Z)
    linearmodel(X,Y,Z)
```

Chapter 9

Statistical Models

9.1 Model language

The mini-language commonly used now in statistics to describe formulas was first used in the languages R and S, but is now also available in Python through the module *patsy*.

For instance, if we have some variable y, and we want to regress it against some other variables x, a, b, and the interaction of a and b, then we simply write

$$y \sim x + a + b + a : b \tag{9.1}$$

The symbols in Table 9.1 are used on the right hand side to denote different interactions. A complete set of the description is found under [patsy(2013)]

9.1.1 Design Matrix

Definition

In a regression model, written in matrix-vector form as

$$y = X\beta + \epsilon, \tag{9.2}$$

the matrix X is the *design matrix*.

Examples

Simple Regression Example of *simple linear regression* with 7 observations. Suppose there are 7 data points $\{y_i, x_i\}$, where $i = 1, 2, , 7$. The simple linear regression model is

Operator	Meaning
~	Separate the left-hand side from the right-hand side. If omitted, formula is assumed right-hand side only.
+	Combines terms on either side (set union).
-	Removes terms on the right from set of terms on the left (set difference).
*	a*b is shorthand for the expansion a + b + a:b.
/	a/b is shorthand for the expansion a + a:b. It is used when b is nested within a (e.g., states and counties)
:	Computes the interaction between terms on the left and right.
**	Takes a set of terms on the left and an integer n on the right and computes the * of that set of terms with itself n times.

Table 9.1: Formula sytax

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \tag{9.3}$$

where β_0 is the y-intercept and β_1 is the slope of the regression line. This model can be represented in matrix form as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \\ 1 & x_6 \\ 1 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{9.4}$$

where the first column of ones in the design matrix represents the y-intercept term while the second column is the x-values associated with the y-value.

Multiple Regression Example of *multiple regression* with covariates (i.e. independent variables) w_i and x_i. Again suppose that the data are 7 observations, and for each observed value to be predicted (y_i), there are two covariates that were also observed w_i and x_i. The model to be considered is

$$y_i = \beta_0 + \beta_1 w_i + \beta_2 x_i + \epsilon_i \tag{9.5}$$

This model can be written in matrix terms as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & w_1 & x_1 \\ 1 & w_2 & x_2 \\ 1 & w_3 & x_3 \\ 1 & w_4 & x_4 \\ 1 & w_5 & x_5 \\ 1 & w_6 & x_6 \\ 1 & w_7 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{9.6}$$

One-way ANOVA (Cell Means Model) Example with a one-way analysis of variance (ANOVA) with 3 groups and 7 observations. The given data set has the first three observations belonging to the first group, the following two observations belong to the second group and the final two observations are from the third group. If the model to be fit is just the mean of each group, then the model is

$$y_{ij} = \mu_i + \epsilon_{ij} \tag{9.7}$$

which can be written

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{9.8}$$

It should be emphasized that in this model μ_i represents the mean of the ith group.

One-way ANOVA (offset from reference group) The ANOVA model could be equivalently written as each group parameter τ_i being an offset from some overall reference. Typically this reference point is taken to be one of the groups under consideration. This makes sense in the context of comparing multiple treatment groups to a control group and the control group is considered the "reference". In this example, group 1 was chosen to be the reference group. As such the model to be fit is:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij} \tag{9.9}$$

with the constraint that τ_1 is zero.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu \\ \tau_2 \\ \tau_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{9.10}$$

In this model μ is the mean of the reference group and τ_i is the difference from group i to the reference group. τ_1 and is not included in the matrix because its difference from the reference group (itself) is necessarily zero.

9.2 Assumptions

Standard linear regression models with standard estimation techniques make a number of assumptions about the predictor variables, the response variables and their relationship. Numerous extensions have been developed that allow each of these assumptions to be relaxed (i.e. reduced to a weaker form), and in some cases eliminated entirely. Some methods are general enough that they can relax multiple assumptions at once, and in other cases this can be achieved by combining different extensions. Generally these extensions make the estimation procedure more complex and time-consuming, and may also require more data in order to get an accurate model.

The following are the major assumptions made by standard linear regression models with standard estimation techniques (e.g. ordinary least squares):

- **Weak exogeneity**. This essentially means that the predictor variables x can be treated as fixed values, rather than random variables. This means, for example, that the predictor variables are assumed to be error-free, that is they are not contaminated with measurement errors. Although not realistic in many settings, dropping this assumption leads to significantly more difficult errors-in-variables models.

- **Linearity**. This means that the mean of the response variable is a linear combination of the parameters (regression coefficients) and the predictor variables. Note that this assumption is much less restrictive than it may at first seem. Because the predictor variables are treated as fixed values (see above), linearity is really only a restriction on the parameters. The predictor variables themselves can be arbitrarily transformed, and in fact multiple copies of the same underlying predictor variable can be added, each one transformed differently. This trick is used, for example, in polynomial regression, which uses linear regression to fit the response variable as an arbitrary polynomial function (up to a given rank) of a predictor variable. This makes linear regression an extremely powerful inference method. In fact, models such as polynomial regression are often "too powerful", in that they tend to overfit the data. As a result, some kind of regularization must typically be used to prevent unreasonable solutions coming out of the estimation

process. Common examples are ridge regression and lasso regression. Bayesian linear regression can also be used, which by its nature is more or less immune to the problem of overfitting. (In fact, ridge regression and lasso regression can both be viewed as special cases of Bayesian linear regression, with particular types of prior distributions placed on the regression coefficients.)

- **Constant variance** (aka *homoscedasticity*). This means that different response variables have the same variance in their errors, regardless of the values of the predictor variables. In practice this assumption is invalid (i.e. the errors are heteroscedastic) if the response variables can vary over a wide scale. In order to determine for heterogeneous error variance, or when a pattern of residuals violates model assumptions of homoscedasticity (error is equally variable around the 'best-fitting line' for all points of x), it is prudent to look for a "fanning effect" between residual error and predicted values. This is to say there will be a systematic change in the absolute or squared residuals when plotted against the predicting outcome. Error will not be evenly distributed across the regression line. Heteroscedasticity will result in the averaging over of distinguishable variances around the points to get a single variance that is inaccurately representing all the variances of the line. In effect, residuals appear clustered and spread apart on their predicted plots for larger and smaller values for points along the linear regression line, and the mean squared error for the model will be wrong. Typically, for example, a response variable whose mean is large will have a greater variance than one whose mean is small. For example, a given person whose income is predicted to be $100,000 may easily have an actual income of $80,000 or $120,000 (a standard deviation]] of around $20,000), while another person with a predicted income of $10,000 is unlikely to have the same $20,000 standard deviation, which would imply their actual income would vary anywhere between -$10,000 and $30,000. (In fact, as this shows, in many cases often the same cases where the assumption of normally distributed errors fails the variance or standard deviation should be predicted to be proportional to the mean, rather than constant.) Simple linear regression estimation methods give less precise parameter estimates and misleading inferential quantities such as standard errors when substantial heteroscedasticity is present. However, various estimation techniques (e.g. weighted least squares and heteroscedasticity-consistent standard errors) can handle heteroscedasticity in a quite general way. Bayesian linear regression techniques can also be used when the variance is assumed to be a function of the mean. It is also possible in some cases to fix the problem by applying a transformation to the response variable (e.g. fit the logarithm of the response variable using a linear regression model, which implies that the response variable has a log-normal distribution rather than a normal distribution).

- **Independence of errors**. This assumes that the errors of the response variables are uncorrelated with each other. (Actual statistical independence is a stronger condition than mere lack of correlation and is often not needed, although it can be exploited if it is known to hold.) Some methods (e.g. generalized least squares) are capable of handling correlated errors, although they typically require significantly more data unless some sort of regularization is used to bias the model towards assuming uncorrelated errors. Bayesian linear regression is a general way of handling this issue.

- **Lack of multicollinearity in the predictors**. For standard least squares estimation methods, the design matrix X must have full column rank p; otherwise, we have a condition known as multicollinearity in the predictor variables. This can be triggered by having two or more perfectly correlated predictor variables (e.g. if the same predictor variable is mistakenly given twice, either without transforming one of the copies or by transforming one of the copies linearly). It can also happen if there is too little data available compared

to the number of parameters to be estimated (e.g. fewer data points than regression coefficients). In the case of multicollinearity, the parameter vector β will be non-identifiable, it has no unique solution. At most we will be able to identify some of the parameters, i.e. narrow down its value to some linear subspace of R^p. Methods for fitting linear models with multicollinearity have been developed. Note that the more computationally expensive iterated algorithms for parameter estimation, such as those used in generalized linear models, do not suffer from this problem and in fact it's quite normal to when handling categorical data—categorically-valued predictors to introduce a separate indicator variable predictor for each possible category, which inevitably introduces multicollinearity.

Beyond these assumptions, several other statistical properties of the data strongly influence the performance of different estimation methods:

- The statistical relationship between the error terms and the regressors plays an important role in determining whether an estimation procedure has desirable sampling properties such as being unbiased and consistent.
- The arrangement, or probability distribution of the predictor variables x has a major influence on the precision of estimates of β. Sampling and design of experiments are highly-developed subfields of statistics that provide guidance for collecting data in such a way to achieve a precise estimate of β.

9.2.1 Interpretation

A fitted linear regression model can be used to identify the relationship between a single predictor variable x_j and the response variable y when all the other predictor variables in the model are held fixed. Specifically, the interpretation of β_j is the expected change in y for a one-unit change in x_j when the other covariates are held fixedthat is, the expected value of the partial derivative of y with respect to x_j. This is sometimes called the "unique effect" of x_j on "y". In contrast, the "marginal effect" of x_j on y can be assessed using a correlation coefficient or simple linear regression model relating x_j to y; this effect is the total derivative of y with respect to x_j.

Care must be taken when interpreting regression results, as some of the regressors may not allow for marginal changes (such as dummy variables, or the intercept term), while others cannot be held fixed (recall the example from the introduction: it would be impossible to hold t_j fixed and at the same time change the value of t_i^2.

It is possible that the unique effect can be nearly zero even when the marginal effect is large. This may imply that some other covariate captures all the information in x_j, so that once that variable is in the model, there is no contribution of x_j to the variation in y. Conversely, the unique effect of x_j can be large while its marginal effect is nearly zero. This would happen if the other covariates explained a great deal of the variation of y, but they mainly explain variation in a way that is complementary to what is captured by x_j. In this case, including the other variables in the model reduces the part of the variability of y that is unrelated to x_j, thereby strengthening the apparent relationship with x_j.

The meaning of the expression held fixed may depend on how the values of the predictor variables arise. If the experimenter directly sets the values of the predictor variables according to a study design, the comparisons of interest may literally correspond to comparisons among units whose predictor variables have been held fixed by the experimenter. Alternatively, the expression held fixed can refer to a selection that takes place in the context of data analysis. In this case, we hold a variable fixed by restricting our attention to the subsets of the data that happen to have a common value for the given predictor variable. This is the only interpretation of held fixed that can be used in an observational study.

The notion of a unique effect is appealing when studying a complex system where multiple interrelated components influence the response variable. In some cases, it can literally be interpreted as the causal effect of an intervention that is linked to the value of a predictor variable. However, it has been argued that in many cases multiple regression analysis fails to clarify the relationships between the predictor variables and the response variable when the predictors are correlated with each other and are not assigned following a study design.

Listing 9.1: modeling.py

```
'''Simple linear models.
- "model_formulas" is based on examples in Kaplan "Statistical Modeling".
- "polynomial_regression" shows how to work with simple design matrices, like
    MATLAB's "regress" command.

'''

'''
Author: Thomas Haslwanter
Date:   Jan-2013
Ver:    2.0
'''

from pandas import read_csv
from statsmodels.formula.api import ols
import statsmodels.regression.linear_model as sm
from statsmodels.stats.anova import anova_lm
import numpy as np

def model_formulas():
    ''' Define models through formulas '''
    # Get the dta
    data = read_csv(r'.\data_kaplan\swim100m.csv')

    # Different models
    model1 = ols("time ~ sex", data).fit()  # one factor
    model2 = ols("time ~ sex + year", data).fit()    # two factors
    model3 = ols("time ~ sex * year", data).fit()    # two factors with
        interaction

    # Model information
    print model1.summary()
    print model2.summary()
    print model3.summary()

    # ANOVAs
    print '------------------------------------------------------------------------'
    print anova_lm(model1)
    print '------------------------------------------------------------------------'
    print anova_lm(model2)
    print '------------------------------------------------------------------------'
    print anova_lm(model3)

def polynomial_regression():
    ''' Define the model directly through the design matrix. Similar to MATLAB's
        "regress" command '''

    # Generate the data
    t = np.arange(0,10,0.1)
    y = 4 + 3*t + 2*t**2 + 5*np.random.randn(len(t))

    # Make the fit. Note that this is another "OLS" than the one in "
        model_formulas"!
    M = np.column_stack((np.ones(len(t)), t, t**2))
```

```
    res = sm.OLS(y, M).fit()

    # Display the results
    print 'Summary:'
    print res.summary()
    print 'The fit parameters are: {0}'.format(str(res.params))
    print 'The confidence intervals are:'
    print res.conf_int()

if __name__ == '__main__':
    model_formulas()
    polynomial_regression()
```

9.3 Bootstrapping

Another type of modelling is *bootstrapping*/. Sometimes you have data describing a distribution, but do not know what type of distribution it is. So what can you do if you want to find out e.g. confidence values for the mean?

The answer is bootstrapping. Bootstrapping is a scheme of *resampling*, i.e. taking additional samples repeatedly from the initial sample, to provide estimates of its variability. In a case where the distribution of the initial sample is unknown, bootstrapping is of especial help in that it provides information about the distribution.

Listing 9.2: bootstrap.py

```
''' Example of bootstrapping the confidence interval for the mean of a sample
    distribution
Since no bootstrapping is implemented in Python, you first have to install the
    the
scikits-bootstrapping module
   pip install -e git+http://github.org/cgevans/scikits-bootstrap.git#egg=
       Package
'''

'''
Author:  Thomas Haslwanter
Date:    March-2013
Version: 1.0
'''

import scipy as sp
import matplotlib.pyplot as plt
from scipy import stats
import scikits.bootstrap as bootstrap

def generate_data():
    # Generate a non-normally distributed datasample
    data = stats.poisson.rvs(2, size=1000)

    # Show the data
    plt.plot(data, '.')
    plt.title('Non-normally distributed dataset: Press any key to continue')
    # plt.show()
    plt.waitforbuttonpress()
    plt.close()

    return(data)

def calc_bootstrap(data):
    # Calculate the bootstrap
```

```
    CIs = bootstrap.ci(data=data, statfunction=sp.mean)

    # Print the data: the "*" turns the array CIs into a list
    print('The conficence intervals for the mean are: {0} - {1}'.format(*CIs))

if __name__ == '__main__':
    data = generate_data()
    calc_bootstrap(data)
```

Chapter 10

Analysis of Survival Times

When analyzing survival times, different problems come up than the ones discussed so far. One question is how do we deal with subjects dropping out of a study. For example, assume that we test a new cancer drug. While some subjects die, others may believe that the new drug is not effective, and decide to drop out of the study before the study is finished. A similar problem would be faced when we investigate how long a machine lasts before it breaks down.

10.1 Survival Probabilities

10.1.1 Kaplan-Meier survival curve

A clever way to deal with these problems is described in detail in [Altman(1999)]. First, the time is subdivided into small periods. Then the likelihood is calculated that a subject survives a given period. The survival probability is given by

$$p_k = p_{k-1} * \frac{r_k - f_k}{r_k} \tag{10.1}$$

where p_k is the probability to survive period k; r_k is the number of subjects still at risk (i.e. still being followed up) immediately before the k^{th} day, and f_k is the number of observed failures on the day k. The curve describing the resulting survival probability is called *life table*, *survival curve*, or *Kaplan-Meier curve* (see Figure 10.1).

Note that the survival curve changes only when a "failure" occurs, i.e. when a subject dies. *Censored* entries, describing either when a subject drops out of the study or when the study

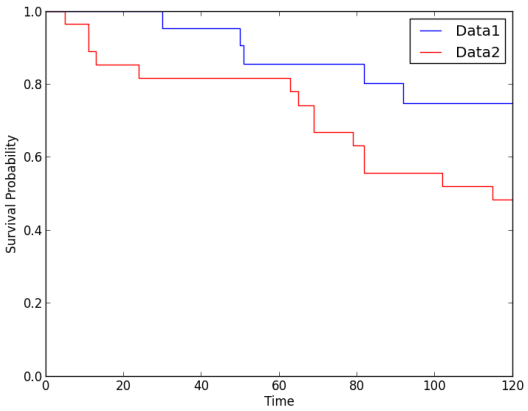


Figure 10.1: Survival curve corresponding to a motion sickness experiment, described in more detail in [Altman(1999)], chapter 13

finishes, are taken into consideration at the "failure" times, but otherwise do not affect the survival curve.

10.2 Comparing Survival Curves in Two Groups

The most common test for comparing independent groups of survival times is the *logrank test*. This test is a non-parametric hypothesis test, testing the probability that both groups come from the same underlying population. Since to my knowledge this test is not yet implemented in a Python library, I have included an implementation based on the equations given by [Altman(1999)] (see program 10.1).

To explore the effect of different variables on survival, more advanced methods are required. The *Cox regression model* introduced by Cox in 1972 is used widely when it is desired to investigate several variables at the same time. For details, check [Altman(1999)] or other statistic textbooks.

Listing 10.1: survival.py

```
'''Survival Analysis
The first function draws the Survival Curve (Kaplan-Meier curve).
The second function implements the logrank test, comparing two survival curves.
The formulas and the example are taken from Altman, Chapter 13

'''

'''
Author : Thomas Haslwanter
Date : March 2013
Ver : 1.0
'''

import numpy as np
from scipy import stats
from getdata import getData
import matplotlib.pyplot as plt

def kaplanmeier(data):
    '''Determine and the Kaplan-Meier curve for the given data.
    Censored times are indicated with "1" in the second column, uncensored with
        "0"'''
    times = data[:,0]
    censored = data[:,1]
    atRisk = np.arange(len(times),0,-1)

    failures = times[censored==0]
    num_failures = len(failures)
    p = np.ones(num_failures+1)
    r = np.zeros(num_failures+1)
    se = np.zeros(num_failures+1)

    # Calculate the numbers-at-risk, the survival probability, and the standard
        error
    for ii in range(num_failures):
        if failures[ii] == failures[ii-1]:
            r[ii+1] = r[ii]
            p[ii+1] = p[ii]
            se[ii+1] = se[ii]

        else:
            r[ii+1] = max(atRisk[times==failures[ii]])
            p[ii+1] = p[ii] * (r[ii+1] - sum(failures==failures[ii]))/r[ii+1]
```

```
            se[ii+1] = p[ii+1]*np.sqrt((1-p[ii+1])/r[ii+1])
            # confidence intervals could be calculated as ci = p +/- 1.96 se

    # Plot survival curve (Kaplan-Meier curve)
    # Always start at t=0 and p=1, and make a line until the last measurement
    t = np.hstack((0, failures, np.max(times)))
    sp = np.hstack((p, p[-1]))

    return(p,atRisk,t,sp,se)

def logrank(data_1, data_2):
    '''Logrank hypothesis test, comparing the survival times for two different
        datasets'''

    times_1 = data_1[:,0]
    censored_1 = data_1[:,1]
    atRisk_1 = np.arange(len(times_1),0,-1)
    failures_1 = times_1[censored_1==0]

    times_2 = data_2[:,0]
    censored_2 = data_2[:,1]
    atRisk_2 = np.arange(len(times_2),0,-1)
    failures_2 = times_2[censored_2==0]

    failures = np.unique(np.hstack((times_1[censored_1==0], times_2[censored_2
        ==0])))
    num_failures = len(failures)
    r1 = np.zeros(num_failures)
    r2 = np.zeros(num_failures)
    r  = np.zeros(num_failures)
    f1 = np.zeros(num_failures)
    f2 = np.zeros(num_failures)
    f  = np.zeros(num_failures)
    e1 = np.zeros(num_failures)
    f1me1 = np.zeros(num_failures)
    v = np.zeros(num_failures)

    for ii in range(num_failures):
        r1[ii] = np.sum(times_1 >= failures[ii])
        r2[ii] = np.sum(times_2 >= failures[ii])
        r[ii] = r1[ii] + r2[ii]

        f1[ii] = np.sum(failures_1==failures[ii])
        f2[ii] = np.sum(failures_2==failures[ii])
        f[ii] = f1[ii] + f2[ii]

        e1[ii] = r1[ii]*f[ii]/r[ii]
        f1me1[ii] = f1[ii] - e1[ii]
        v[ii] = r1[ii]*r2[ii]*f[ii]*(r[ii]-f[ii]) / ( r[ii]**2 *(r[ii]-1) )

    O1 = np.sum(f1)
    O2 = np.sum(f2)
    E1 = np.sum(e1)
    O1mE1 = np.sum(f1me1)
    V = sum(v)

    chi2 = (O1-E1)**2/V
    p = stats.chi2.sf(chi2, 1)

    print('X^2 = {0}'.format(chi2))
    if p < 0.05:
        print('p={0}, the two survival curves are signifcantly different.'.
            format(p))
```

```
        else:
            print('p={0}, the two survival curves are not signifcantly different.'.
                format(p))

        return(p, chi2)

    if __name__=='__main__':
        # get the data
        data1 = getData('altman_13_2.txt')
        data2 = getData('altman_13_3.txt')

        # Determine the Kaplan-Meier curves
        (p1, r1, t1, sp1,se1) = kaplanmeier(data1)
        (p2, r2, t2, sp2,se2) = kaplanmeier(data2)

        # Make a combined plot for both datasets
        plt.step(t1,sp1, where='post')
        plt.hold(True)
        plt.step(t2,sp2,'r', where='post')

        plt.legend(['Data1', 'Data2'])
        plt.ylim(0,1)
        plt.xlabel('Time')
        plt.ylabel('Survival Probability')
        plt.show()

        # Check the hypothesis that the two survival curves are the same
        logrank(data1, data2)
```

Appendix A

Appendix

A.1 Lecture Schedule

1. Introduction
2. Basics [T]
3. Study Design
4. Normal Distribution [T]
5. Other Continuous Distributions
6. Data Analysis [T]
7. Statistical Tests
8. Continous Tests
9. **Presentation Experimental Design**
10. Categorical Tests
11. Correlation [T]
12. Regression [T]
13. ANOVA [T]
14. Statistical Models
15. **Final Presentation**

Bibliography

[Altman(1999)] Douglas G. Altman. *Practical Statistics for Medical Research*. Chapman & Hall/CRC, 1999.

[Harms and McDonald(2010)] D. Harms and K. McDonald. *The Quick Python Book (2nd Ed)*. Manning Publications Co., 2010.

[Kaplan(2009)] Daniel Kaplan. *Statistical Modeling: A Fresh Approach*. Macalester College, 2009.

[patsy(2013)] patsy, 2013. URL `http://patsy.readthedocs.org/en/latest/overview.html`.

[Python(xy)(2013)] Python(xy), 2013. URL `http://code.google.com/p/pythonxy/`.

[Riffenburgh(2012)] R.H. Riffenburgh. *Statistics in Medicine*. Academic Press, 3rd edition, 2012.

[WinPython(2013)] WinPython, 2013. URL `http://code.google.com/p/winpython/`.

Index Topics

Python Programs