

UNIVERSITY OF APPLIED SCIENCES

AN INTRODUCTION TO

Statistics

Author:
Thomas HASLWANTER

email:
thomas.haslwanger@fh-linz.at



Version: 3.1.1
May 12, 2014



This work is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License.

Contents

1	Introduction	7
1.1	Why Statistics?	8
1.2	What you should already know	8
1.3	Projects	8
1.4	Programming Matters	9
1.4.1	Python	9
1.4.2	Pandas	12
1.4.3	Statsmodels	14
1.4.4	Seaborn	14
1.4.5	General Routines	14
1.5	Exercises	14
2	Basic Principles	17
2.1	Datatypes	17
2.1.1	Categorical	17
2.1.2	Numerical	17
2.2	Data Display	18
2.2.1	Scatter Plots	18
2.2.2	Histograms	18
2.2.3	KDE-plots	18
2.2.4	Cumulative Frequencies	20
2.2.5	Box Plots	21
2.2.6	Programs: Data Display	21
2.3	Study Design	21
2.3.1	Types of Studies	21
2.3.2	Design of Experiments	22
2.3.3	Structure of Experiments	24
2.3.4	Data Management	24
2.3.5	Clinical Investigation Plan	25
2.4	Exercises	25
3	Distributions of one Variable	27
3.1	Characterizing a Distribution	27
3.1.1	Population and samples	27
3.1.2	Continuous Distribution Functions	27
3.1.3	Distribution Center	29
3.1.4	Quantifying Variability	30
3.1.5	Parameters Describing the Form of a Distribution	31
3.2	Distribution Functions	32
3.2.1	Normal Distribution	32
3.2.2	Central Limit Theorem	35

3.2.3	Application Example	35
3.2.4	Other Continuous Distributions	36
3.2.5	Discrete Distributions	43
3.3	Data Analysis	45
3.3.1	Data Screening	45
3.3.2	Normality Check	45
3.3.3	Transformation	46
3.3.4	Confidence Intervals	46
3.4	Exercises	47
3.4.1	Python	47
3.4.2	Distributions	47
3.4.3	Analysis	47
3.4.4	Continuous Distributions	47
3.4.5	Discrete Distributions	48
4	Statistical Tests	49
4.1	Hypothesis tests	49
4.1.1	Types of Error	50
4.1.2	Sample Size	50
4.1.3	The "p-value fallacy"	52
4.2	Sensitivity and Specificity	52
4.3	ROC Curve	54
4.4	Common Statistical Tests for Comparing Groups	54
5	Test of Means of Continuous Data	57
5.1	Distribution of a Sample Mean	57
5.1.1	One sample t-test for a mean value	57
5.1.2	Wilcoxon signed rank sum test	57
5.2	Comparison of Two Groups	58
5.2.1	Non-parametric Comparison of Two Groups: Mann-Whitney Test	58
5.2.2	Statistical Tests vs Statistical Modeling	59
5.3	Comparison of More Groups	59
5.3.1	Analysis of Variance - ANOVA	59
5.3.2	Multiple Comparisons	61
5.3.3	Kruskal-Wallis test	62
5.4	Exercises	62
5.4.1	One or Two Groups	62
5.4.2	Multiple Groups	63
6	Tests on Discrete Data	65
6.1	Tests on Ordinal Data	65
6.2	Binomial Test	65
7	Tests on Categorical Data	67
7.1	One Proportion	67
7.2	Frequency Tables	68
7.2.1	Chi-square Test	68
7.2.2	Fisher's Exact Test	69
7.2.3	McNemar's Test	70
7.2.4	Cochran's Q Test	71
7.3	Analysis Programs	71
7.4	Exercises	72

7.4.1	Fisher's Exact Test - The Tea Experiment	72
8	Relation Between Two Continuous Variables	73
8.1	Correlation	73
8.1.1	Correlation Coefficient	73
8.1.2	Coefficient of determination	73
8.1.3	Rank Correlation	75
8.2	Regression	75
8.2.1	Introduction	76
8.2.2	Assumptions	78
8.3	Exercises	79
9	Relation Between Several Variables	81
9.1	Two-way ANOVA	81
9.2	Multilinear Regression	82
10	Statistical Models	83
10.1	Model language	83
10.1.1	Design Matrix	83
10.1.2	Example: Program Effectiveness	85
10.2	Linear Regression Analysis with Python	85
10.2.1	Model Results	87
10.2.2	The <i>adjusted</i> R^2 Value	88
10.2.3	Model Coefficients and Their Interpretation	90
10.2.4	Analysis of Residuals	93
10.2.5	Comparison	95
10.2.6	Regression Using Sklearn	95
10.2.7	Conclusion	96
10.3	Assumptions	96
10.3.1	Interpretation	99
10.4	Bootstrapping	100
11	Analysis of Survival Times	101
11.1	Survival Probabilities	101
11.1.1	Kaplan-Meier survival curve	101
11.2	Comparing Survival Curves in Two Groups	102
A	Appendix	103
A.1	Python Programs	103
A.2	Lecture Schedule	151
A.3	To Do	151
	Index Topics	154
	Python Programs	156

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 opinion a bit harder to read, is [Riffenburgh(2012)]. If you are interested in a simple introduction to modern regression modeling, check out [Kaplan(2009)]. A very good introduction to Generalized Linear Models is [AJ and AG(2008)]. If you know your basic statistics, this is a good, advanced starter into statistical modeling.

WWW: On the web, you find good very extensive statistics information in English under

- <http://www.statsref.com/>
- <http://www.vassarstats.net/>
- <http://udel.edu/~mcdonald/statintro.html>
- <http://onlinestatbook.com/2/index.html>

A good German webpage on statistics and regulatory issues is <http://www.reiter1.com/>.

Exercises: Many examples are already solved in the text. For the use in lectures (or for self-test), additional exercises are provided at the end of most chapters. For lecturers, solutions to these exercises can be provided on demand. Please contact me directly for that via email.

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.

Without statistics, your interpretation of your data can be massively flawed. Take for example the estimated number of German tanks during World War II, also known as the *German tank problem* (http://en.wikipedia.org/wiki/German_tank_problem): from standard intelligence data, the estimate for the number of German tanks produced per month was 1550; in contrast, the statistical estimate from the tanks observed led to a number of 327, which was very close to the actual production number of 342.

1.2 What you should already know

From previous courses on math, quality control, signal analysis, etc., you are probably already familiar with a number of statistical concepts. While they will be dealt with in detail later on, let me list them here to make sure we start at the same level:

- mean
- median
- mode
- standard deviation
- variance
- confidence intervals
- t-test
- boxplot
- normal distribution
- regression coefficient
- coefficient of determination

1.3 Projects

The biggest problems in statistics do *not* arise from a faulty analysis, but from a faulty experimental design. If you have a suitable topic of interest, you can select to do a project instead of the final exam. For this 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. Analyze some data.
4. Generate the appropriate graphs.
5. Write up a summary of your project.

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)]. A good free book, which introduces Python with a focus on statistics, is "Introduction to Python for Econometrics, Statistics and Data Analysis", by Kevin Sheppard, Oxford University.

In general, I suggest that you start out by installing a Python distribution which includes the most important libraries. Since I suggest that you use Python > 3.3 for this course, [Python(xy)(2013)] (which comes complete with help) is currently not an option, since it is only available for Python 2.7. All the Python packages required for this course are now available for Python 3, so I don't see a good reason to stay with Python 2.7. My favorites Python 3.3 distributions are

1. [WinPython(2013)] No admin-rights required. Recommended for Windows users.
2. [Anaconda(2014)] From Continuum. For Windows, Mac, and Linux. By default installs to Python 2.7.x, but can upgrade to Python 3.x.

which are very good starting points when you are using Windows. *winpython* does not require administrator rights, and *anaconda* is a more recent distribution, which is free for educational purposes.

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!

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

- *ipython* ... For interactive work.
- *numpy* ... For working with vectors and arrays.

http://scipy-lectures.github.com http://www.scipy.org/NumPy_for_Matlab_Users https://github.com/jrjohansson/scientific-python-lectures http://docs.python.org/2/tutorial http://swaroopch.com/notes/python http://learnpythonthehardway.org/book/ http://www.greenteapress.com/thinkpython	<i>Python Scientific Lecture Notes</i> . Pretty comprehensive. <i>NumPy for Matlab Users</i> Start here if you have Matlab experience. <i>Lectures on scientific computing with Python</i> . Great IPython notebooks! <i>The Python tutorial</i> . The official introduction. <i>A Byte of Python</i> . Free book, very good at the introductory level. <i>Learn Python the Hard Way, 3rd Ed</i> A popular, free book that you can work through. <i>ThinkPython</i> . Free book, for advanced programmers.
---	--

Table 1.1: Python on the WWW

- *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.
- *patsy* ... For working with statistical formulas.
- *statsmodels* ... For statistical modeling and advanced analysis.
- *seaborn* ... For visualization of statistical data.

Ipython

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 an [IPython(2013)] *qtconsole*. IPython provides interactive computing with Python, similar to the commandline in Matlab. It comes with a command history, interactive data visualization, command completion, and a lot of features that make it quick and easy to try out code. When ipython is started in *pylab mode* (which is the typical configuration), it automatically loads numpy and matplotlib.pyplot into the active workspace, and provides a very convenient, Matlab-like programming environment. A very helpful new addition is the browser-based *ipython notebook*, with support for code, text, mathematical expressions, inline plots and other rich media. Please check out the links to the ipython notebooks in this statistics introduction. I believe that it will help you to get up to speed with python much more quickly.

And to write a program, I then go to either *Spyder* (which is free) or *Wing* (which is very good, but commercial). *PyCharm* is another IDE with a good debugger, and has very good vim-emulation.

The flexibility of Python has the "disadvantage" that it can come in different flavors or coding styles. When you know the different approaches, they are great to use. But when you get started, it can be a bit confusing. The following section from the Matplotlib documentation may help to clarify these things:

Matplotlib, pylab, and pyplot: how are they related?

Matplotlib is the whole package; *pylab* is a Matlab-like module in matplotlib that gets installed alongside matplotlib; and *matplotlib.pyplot* is a module in matplotlib.

Pyplot provides the state-machine interface to the underlying plotting library in matplotlib. This means that figures and axes are implicitly and automatically created to achieve the desired plot. For example, calling *plot* from pyplot will automatically create the necessary figure and axes to achieve the desired plot. Setting a *title* will then automatically set that title to the current axes object:

```
import matplotlib.pyplot as plt

plt.plot(np.arange(10))
plt.title("Simple Plot")
plt.show()
```

Pylab combines the pyplot functionality (for plotting) with the numpy functionality (for mathematics and for working with arrays) in a single namespace, making that namespace (or environment) even more MATLAB-like. For example, one can call the sin and cos functions just like you could in MATLAB, as well as having all the features of pyplot.

The pyplot interface is generally preferred for non-interactive plotting (i.e., scripting). The pylab interface is convenient for interactive calculations and plotting, as it minimizes typing. Note that this is what you get if you use the ipython shell with the -pylab option, which imports everything from pylab and makes plotting fully interactive.

Coding Styles in Python

In Python you will find different coding styles and usage patterns. These styles are all perfectly valid, and each have their pros and cons. Just about all of the examples can be converted into another style and achieve the same results. The only caveat is to avoid mixing the coding styles for your own code.

Of the different styles, there are two that are officially supported. Therefore, these are the preferred ways to use matplotlib.

For the preferred pyplot style, the imports at the top of your scripts will typically be:

```
import matplotlib.pyplot as plt
import numpy as np
```

Then one calls, for example, np.arange, np.zeros, np.pi, plt.figure, plt.plot, plt.show, etc. So, a simple example in this style would be:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 10, 0.2)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

Note that this example used pyplot's state-machine to automatically and implicitly create a figure and an axes. For full control of your plots and more advanced usage, use the pyplot interface for creating figures, and then use the object methods for the rest:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 10, 0.2)
y = np.sin(x)
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y)
plt.show()
```

Next, the same example using a pure MATLAB-style:

```
from pylab import *
x = arange(0, 10, 0.2)
y = sin(x)
plot(x, y)
```

So, why all the extra typing as one moves away from the pure MATLAB-style? For very simple things like this example, the only advantage is academic: the wordier styles are more explicit, more clear as to where things come from and what is going on. For more complicated

applications, this explicitness and clarity becomes increasingly valuable, and the richer and more complete object-oriented interface will likely make the program easier to write and maintain.

Here an example, to get you started with Python. For interactive work, it is simplest to use the *pylab mode*, as shown in the example below. The corresponding ipython notebook http://nbviewer.ipython.org/github/thomas-haslwanter/statsintro/blob/master/ipynb/getting_started.ipynb) shows how numpy and matplotlib can be addressed directly. (This is common practice when writing functions.)

Example-Session



Code: "gettingStarted.py" (p 103) gives a short demonstration of Python for scientific data analysis.

1.4.2 Pandas

[Pandas(2013)] is a Python module which provides suitable data structures for statistical analysis. It significantly enhances the abilities of Python for data input, data organization, and data manipulation. In the following, I assume that pandas has been imported with

```
import pandas as pd
```

A good introduction to pandas can be found under <http://www.randalolson.com/2012/08/06/statistical-analysis-made-easy-in-python/>


Data Input

Pandas offers tools for reading and writing data between in-memory data structures and different formats, e.g. CSV and text files, Microsoft Excel, and SQL databases. For example, if you have data in your clipboard, you can import them directly with

```
data = pd.read_clipboard()
```

Or data from "Sheet1" in an Excel-file "data.xls" can be read in easily with

```
xls = pd.io.parsers.ExcelFile('data.xls')
data = xls.parse('Sheet1')
```

Example:  **Code:** "readZip.py" (p 106) This advanced script shows you how you can directly import data from an MS-Excel file from a zipped archive on the web.

Data Handling and Manipulation

To handle labeled data, pandas introduces *DataFrame* objects. A *DataFrame* is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table. It is generally the most commonly used pandas object. At first, handling data with Pandas feels a bit unusual. To get you started, let me give you a specific example:

```
import numpy as np
import pandas as pd

t = np.arange(0,10,0.1)
x = np.sin(t)
y = np.cos(t)

df = pd.DataFrame({'Time':t, 'x':x, 'y':y})
```

In Pandas, rows are addressed through "indices", and columns through their "column" name. To address the first column only, you have two options:

```
df.Time
df['Time']
```

If you want to extract two columns at the same time, you have to use a Python-list:

```
data = df[['Time', 'y']]
```

To display the first or last rows, use

```
data.head()
data.tail()
```

For e.g. rows 5-10 (note that this are 6 numbers), use

```
data[4:10]
```

as $10 - 4 = 6$. (I know, the array indexing takes some time to get used to. Just keep in mind that Python addresses the *locations between* entries, not the entries, and that it starts at 0!!) To do this in one go, use

```
df[['Time', 'y']][4:10]
```

You can also apply the standard row/column notation, by using the method "ix":

```
df.ix[[0,2],4:10]
```

Finally, sometimes you want to have direct access to the data, not to the DataFrame. You can do this with

```
data.values
```

Pandas offers powerful functions to handle missing data and "nans", and other kinds of data manipulation like pivoting. For example, you can use data-frames to efficiently group objects, and do a statistical evaluation of each group. The following data are simulated (but realistic) data of a survey on how many hours a day people watch on the TV, grouped into "m"ale and "f"emale responses:

```
data = pd.DataFrame({
    'Gender': ['f', 'f', 'm', 'f', 'm', 'm', 'f', 'm', 'f', 'm'],
    'TV': [3.4, 3.5, 2.6, 4.7, 4.1, 4.0, 5.1, 4.0, 3.7, 2.1]
})
```

```
# Group the data
grouped = data.groupby('Gender')
```

```
# Get the groups as DataFrames
df_female = grouped.get_group('f')
```

```
# Get the corresponding numpy-array
values_female = grouped.get_group('f').values
```

```
# or equivalently
groups = grouped.groups
values_female = groups['f']
```

```
# Do some overview statistics
print(grouped.describe())
```

produces

		TV
Gender		
f	count	5.000000
	mean	4.080000
	std	0.769415
	min	3.400000
	25%	3.500000

	50%	3.700000
	75%	4.700000
	max	5.100000
m	count	5.000000
	mean	3.360000
	std	0.939681
	min	2.100000
	25%	2.600000
	50%	4.000000
	75%	4.000000
	max	4.100000

For statistical analysis, pandas becomes really powerful if you combine it with *statsmodels* (see below).

1.4.3 Statsmodels

[statsmodels(2013)] is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. In its latest release (version 0.5), statsmodels also allows the formulation of models with the popular formula language also used by *R*, the leading statistics package. For example, data on the connection between academic "success", "intelligence" and "diligence" can be described with the model $success \sim intelligence * diligence$, which would capture the direct effect of "intelligence" and "diligence", as well as the interaction. You find more information on that topic in the section "Statistical Models".

While for complex statistical models *R* still has an edge, python has a much clearer and more readable syntax, and is arguably more powerful for the data manipulation often required for statistical analysis.



Code: "statsmodelsIntro.py" (p 107) shows you how the combination of pandas and statsmodels can be used for data analysis.

1.4.4 Seaborn

[Seaborn(2014)] is a Python visualization library based on matplotlib. Its primary goal is to provide a concise, high-level interface for drawing statistical graphics that are both informative and attractive. Seaborn can be installed easily by opening up a *WinPython Command Prompt*, and then typing `pip install seaborn`.

```
x = linspace(1, 7, 50)
y = 3 + 2*x + 1.5*randn(len(x))
sns.regplot(x,y)
```

already produces a nice and informative regression plot

1.4.5 General Routines

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:



Code: "getData.py" (p 108) Gets the input data for many Python programs in this script.

1.5 Exercises

- Read in data from different sources:

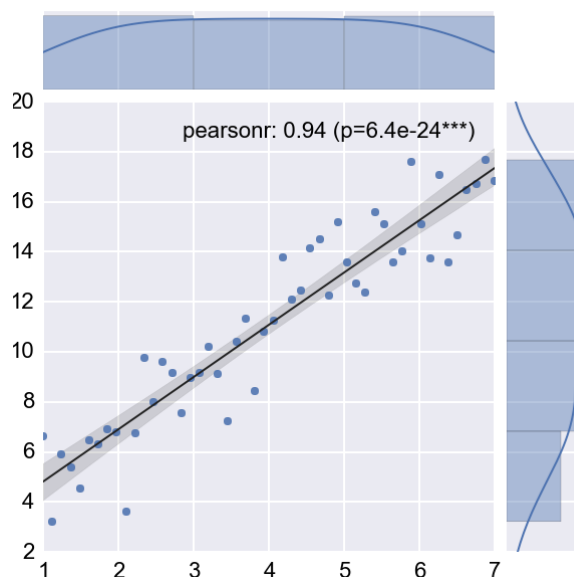


Figure 1.1: Regression plot, from *seaborn*, with a substantial amount of added information.

- A CVS-file with a header ('Data\Swimming\swimming_100m.csv')
- An MS-Excel file ('Data\data.dobson\GLM_data\Table 2.8 Waist loss.xls')
- Data from the WWW (see "readZip.py" A.2)
- – Generate a pandas dataframe, with the x-column time stamps from 0 to 10 sec, at a rate of 10 Hz, the y-column data values with a sine with 1.5 Hz, and the z-column the corresponding cosine values. Label the x-column "Xvals", and the y-column "YVals", and the z-column "ZVals".
- Show the head of this dataframe
- Extract the data in lines 10-15 from "YVals" and "ZVals", and write them to the file "out.txt".

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*

Numerical 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. The programs used for making the plots presented here can be found in "figsBasicPrinciples.py" (p 109). For data visualization, check out the Python package *seaborn*, which aims to provide a concise, high-level interface for drawing statistical graphics that are both informative and attractive.

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.)

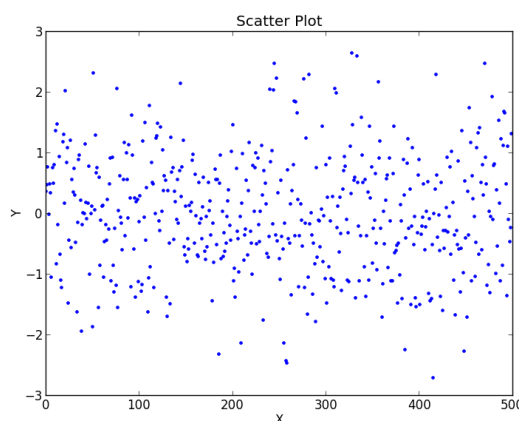


Figure 2.1: Scatter plot

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*.

You can also smooth histograms with *kernel-density-estimations (kde-plots)*. Those are nicely implemented and described in *seaborn*.

2.2.3 KDE-plots

¹ Kernel density estimates are closely related to histograms, but can be endowed with properties such as smoothness or continuity by using a suitable kernel. To see this, we compare the construction of histogram and kernel density estimators, using these 6 data points: $x_1 = 2.1$, $x_2 = 1.3$, $x_3 = 0.4$, $x_4 = 1.9$, $x_5 = 5.1$, $x_6 = 6.2$. For the histogram, first the horizontal axis is divided into sub-intervals or bins which cover the range of the data. In this case, we have 6 bins each of width 2. Whenever a data point falls inside this interval, we place a box of height $1/12$. If more than one data point falls inside the same bin, we stack the boxes on top of each other.

For the kernel density estimate, we place a normal kernel with variance 2.25 (indicated by the red dashed lines) on each of the data points x_i . The kernels are summed to make the kernel

¹This paragraph is taken from Wikipedia, "Kernel density estimation".

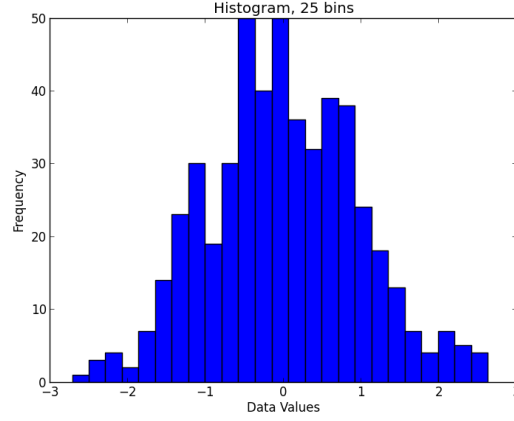


Figure 2.2: Histogram

density estimate (solid blue curve). The smoothness of the kernel density estimate is evident compared to the discreteness of the histogram, as kernel density estimates converge faster to the true underlying density for continuous random variables.

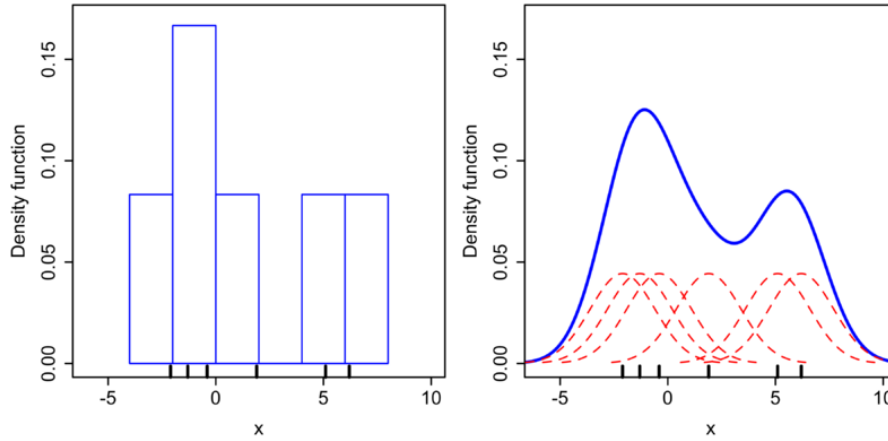


Figure 2.3: Comparison of the histogram (left) and kernel density estimate (right) constructed using the same data. The 6 individual kernels are the red dashed curves, the kernel density estimate the blue curves. The data points are the rug plot on the horizontal axis. (from Wikipedia)

The bandwidth of the kernel is a free parameter which exhibits a strong influence on the resulting estimate. To illustrate its effect, we take a simulated random sample from the standard normal distribution (plotted at the blue spikes in the rug plot on the horizontal axis). The grey curve is the true density (a normal density with mean 0 and variance 1). In comparison, the red curve is undersmoothed since it contains too many spurious data artifacts arising from using a bandwidth $h = 0.05$ which is too small. The green curve is oversmoothed since using the bandwidth $h = 2$ obscures much of the underlying structure. The black curve with a bandwidth of $h = 0.337$ is considered to be optimally smoothed since its density estimate is close to the true density.

It can be shown that under certain conditions the optimal choice for h is

$$h = \left(\frac{4\hat{\sigma}^5}{3n} \right)^{\frac{1}{5}} \approx 1.06\hat{\sigma}n^{-1/5}, \quad (2.1)$$

where $\hat{\sigma}$ is the standard deviation of the samples.

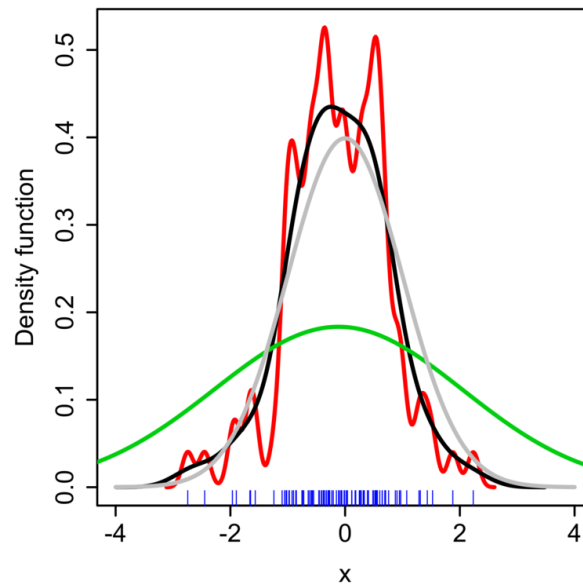


Figure 2.4: Kernel density estimate (KDE) with different bandwidths of a random sample of 100 points from a standard normal distribution. Grey: true density (standard normal). Red: KDE with $h=0.05$. Green: KDE with $h=2$. Black: KDE with $h=0.337$. (from Wikipedia)

2.2.4 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:

$$0 \leq x \leq 1$$

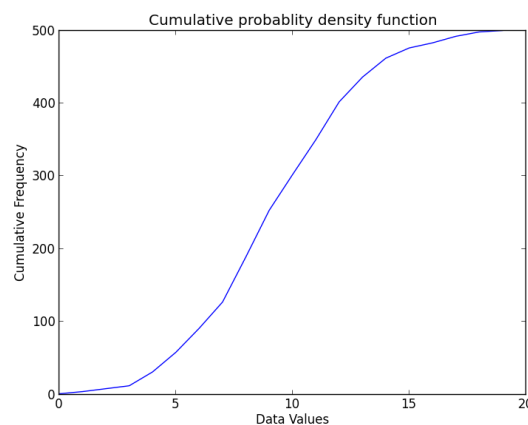


Figure 2.5: Cumulative frequency function

2.2.5 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 $1.5 \times \text{inter-quartile-range}$ (IQR) above or below the first/third quartile (see Section 3.1.4).

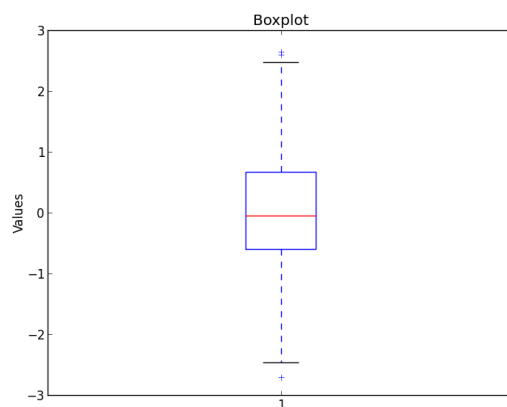



Figure 2.6: Box plot

Boxplots are often combined with KDE-plots to produce so-called *violin-plots*, as shown in Figure 2.7.

2.2.6 Programs: Data Display

 **python**™ **Code:** "figsBasicPrinciples.py" (p 109) shows how the plots in this section have been generated.

2.3 Study Design

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 ob-



Figure 2.7: Violinplot, produced with *seaborn*.

served 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 assignment.

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 biased 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*. If also the person who does the analysis does not know which group the subject has been allocated to, we speak about *triple 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 observe 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

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

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.
17. The designation of a *monitor* for the investigation.
18. The designation of a *clinical investigator*.
19. Specification the data handling.

2.4 Exercises

1. (a) Read in the data from 'Data\amst\babyboom.dat.txt'.
- (b) Inspect them visually, and give a numerical description of the data.
- (c) Are the data normally distributed?
- (d) How would you design the corresponding study?

Chapter 3

Distributions of one Variable

3.1 Characterizing a Distribution

3.1.1 Population and samples

While the whole *population* of a group has certain characteristics, we can typically never measure all of them. In many cases, the population distribution is described by an idealized, continuous distribution function.

In the analysis of measured data, in contrast, we have to confine ourselves to investigate a (hopefully representative) *sample* of this group, and estimate the properties of the population from this sample.

3.1.2 Continuous Distribution Functions

A continuous distribution function describes the distribution of a population, and can be represented in several equivalent ways:

Probability Density Function (PDF)

The PDF, or density of a continuous random variable, is a function that describes the relative likelihood for a random variable X to take on a given value x . In the mathematical fields of probability and statistics, a *random variate* x is a particular outcome of a *random variable* X : the random variates which are other outcomes of the same random variable might have different values.

Since the likelihood to find any given value cannot be less than zero, and since the variable has to have some value, the PDF has the following properties:

- $PDF(x) \geq 0 \forall x \in \mathbb{R}$
- $\int_{-\infty}^{\infty} PDF(x)dx = 1$

Cumulative Density Function (CDF)

The probability to find a value between a and b is given by the integral over the PDF in that range (see Fig. 3.1), and the *Cumulative Density Function* tells you for each value which percentage of the data has a lower value (Figure 3.2). Together, this gives us

$$\mathbb{P}(a \leq X \leq b) = \int_a^b PDF(x)dx = CDF(b) - CDF(a) \quad (3.1)$$

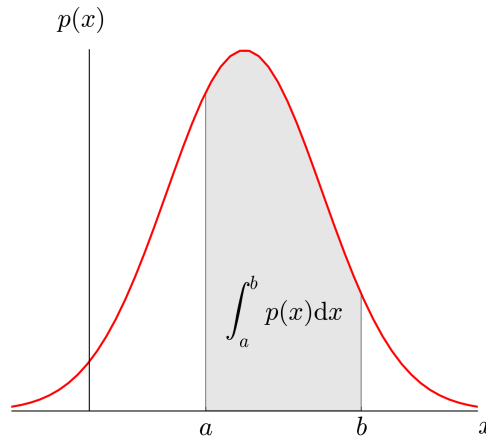


Figure 3.1: Probability Density Function (PDF) of a value x . The integral over the PDF between a and b gives the likelihood of finding the value of x in that range.

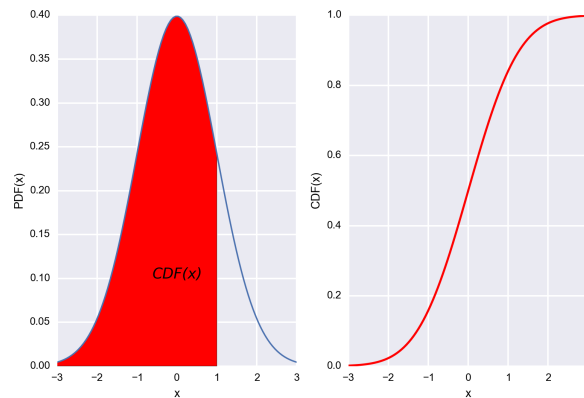


Figure 3.2: *Probability density function* (left) and *Cumulative density function* (right) of a normal distribution.

Other important presentations of Probability Densities

Figure 3.3 shows a number of functions that are equivalent to the PDF, but each represent a different aspect of the probability distribution.

Note: It is very important that you understand this figure!

- *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: gives the probability of obtaining a value larger than the given value. It can also be interpreted as the proportion of data "surviving" above a certain value.
- *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.

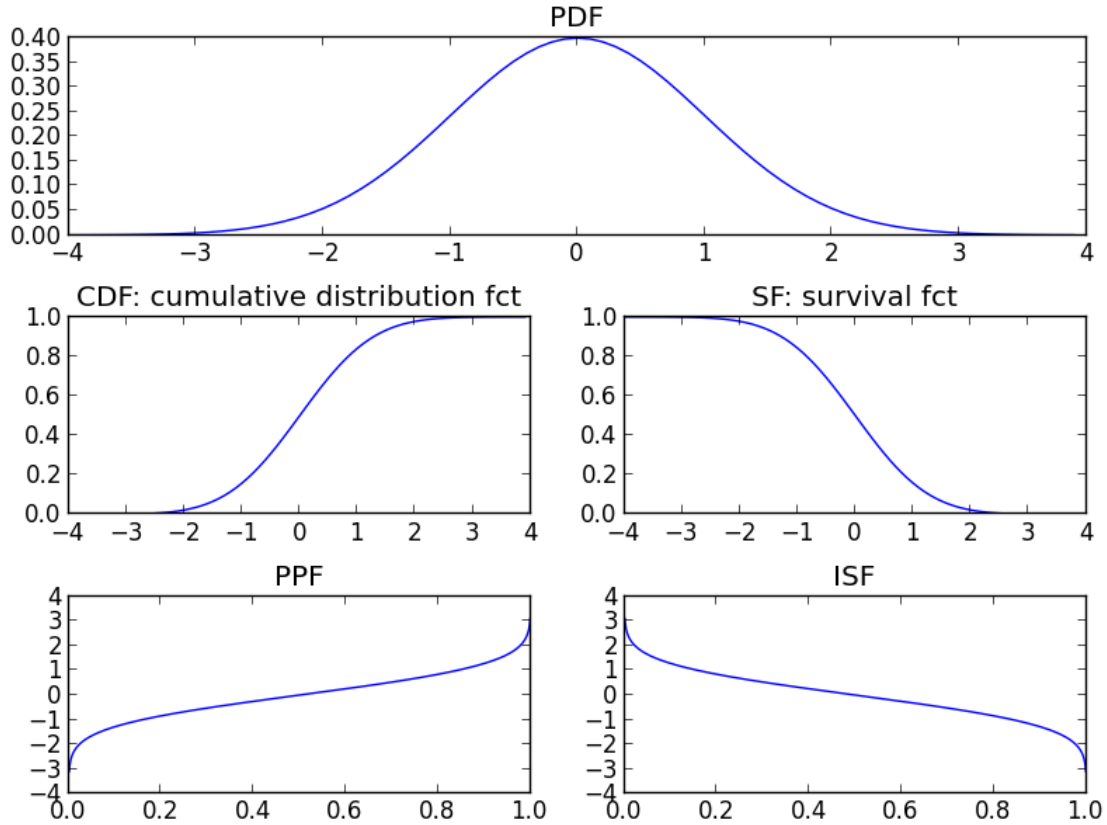


Figure 3.3: Utility functions for continuous distributions, here for the normal distribution.

3.1.3 Distribution Center

When we have a datasample from a distribution, we can characterize the center of the distribution with different parameters:

Mean

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

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} \quad (3.2)$$

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.

3.1.4 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 $1.5 \times \text{inter-quartile-range}$ (IQR) above or below the first/third quartile (see below).

Percentiles

Centiles, also called *percentile*, give the value below which a given percentage of the values occur. It corresponds to a value with a specified cumulative frequency. While you won't often hear the expression *centiles*, you will frequently encounter specific centiles:

- When you look for the data range which includes 95% of the data, you have to find the 2.5th and the 97.5th 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.2.6).

Standard Deviation and Variance

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

$$var = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1} \quad (3.3)$$

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.

Figure 3.4 indicates why the sample standard deviation underestimates the standard deviation of the underlying distribution.

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

$$s = \sqrt{var} \quad (3.4)$$

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
```

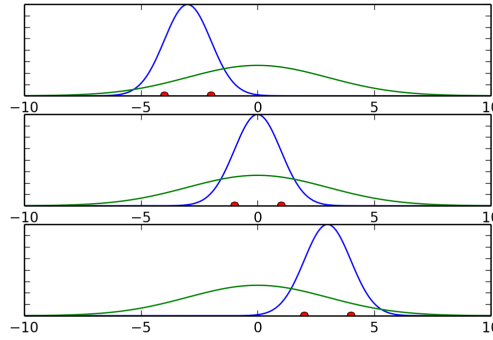


Figure 3.4: Gaussian distributions fitted to selections of data from the underlying distribution: While the average mean of a number of samples converges to the real mean, the sample standard deviation underestimates the standard deviation from the distribution.

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 σ/\sqrt{n} .

For the *sample standard error of the mean*, which is the one you will be working with most of the time, we have

$$SE = \frac{s}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}} \cdot \frac{1}{\sqrt{n}} \quad (3.5)$$

3.1.5 Parameters Describing the Form of a Distribution

Location

A *location parameter* x_0 determines the "location" or shift of a distribution.

$$f_{x_0}(x) = f(x - x_0)$$

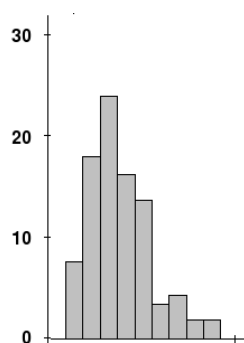
Examples of location parameters include the mean, the median, and the mode.

Scale

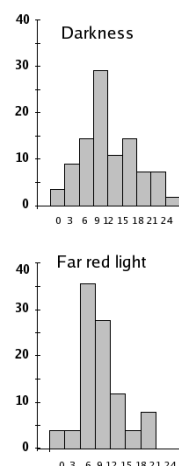
The *scale parameter* describes the width of a probability distribution. If s is large, then the distribution will be more spread out; if s is small then it will be more concentrated. If the probability density exists for all values of s , then the density (as a function of the scale parameter only) satisfies

$$f_s(x) = f(x/s)/s,$$

where f is the density of a standardized version of the density.



(a) Example of experimental data with non-zero (positive) skewness.



(b) The "Darkness" data is platykurtic (0.194), while "Far Red Light" shows leptokurtosis (0.055).

Figure 3.5: Skew and Kurtosis of distributions (from Wikipedia).

Shape Parameters

A shape parameter is any parameter of a probability distribution that is neither a location parameter nor a scale parameter. If *location* and *shape* already completely determine the distribution (as is the case for e.g. the normal distribution or the exponential distribution, etc.), then these distributions don't have any *shape parameter*. It follows that the *skewness* and *kurtosis* of these distribution are constants.

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.

Kurtosis

Kurtosis is any measure of the "peakedness" of the probability distribution. Distributions with negative or positive excess kurtosis are called platykurtic distributions or leptokurtic distributions respectively.

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 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 :

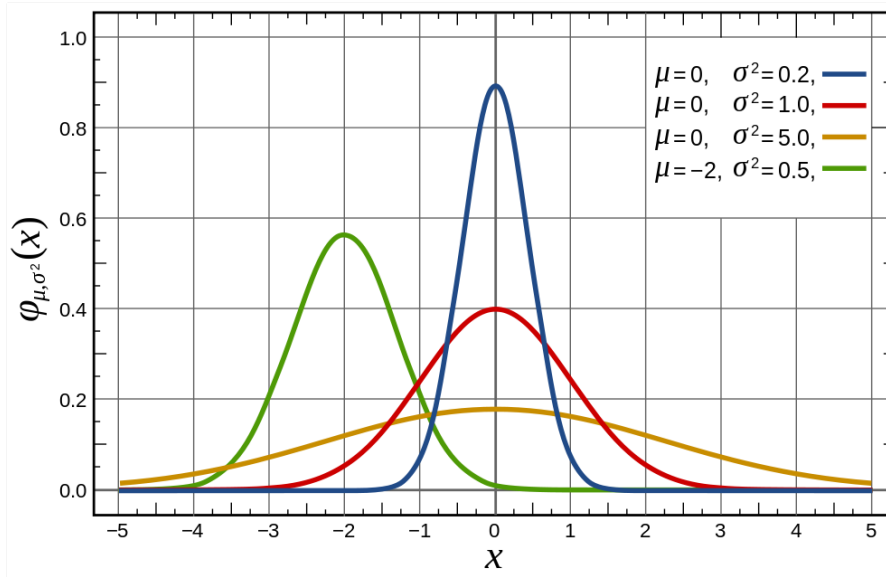


Figure 3.6: Normal Distribution

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad (3.6)$$

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. 3.7)

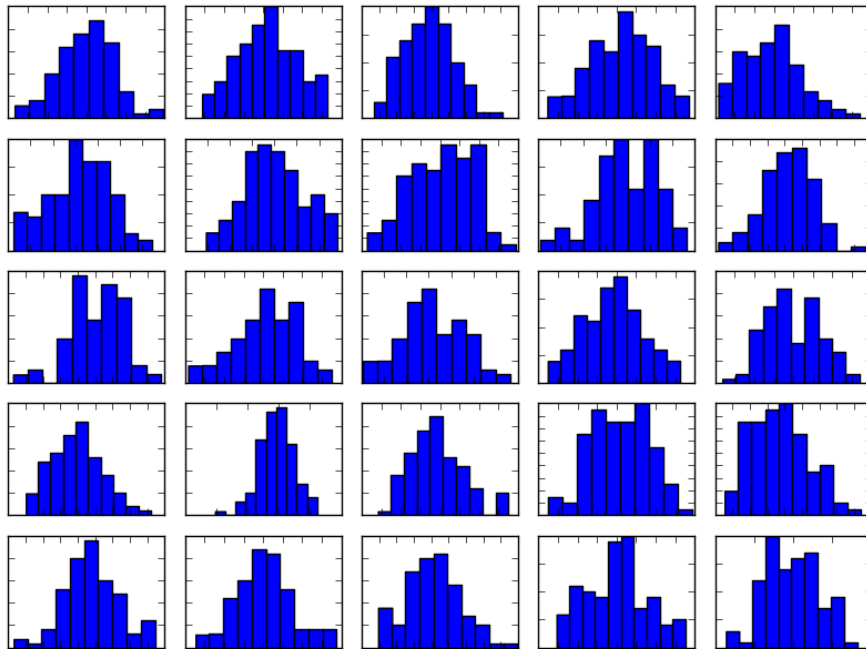


Figure 3.7: 25 randomly generated normal distributions of 100 points.

Some examples of applications are:

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

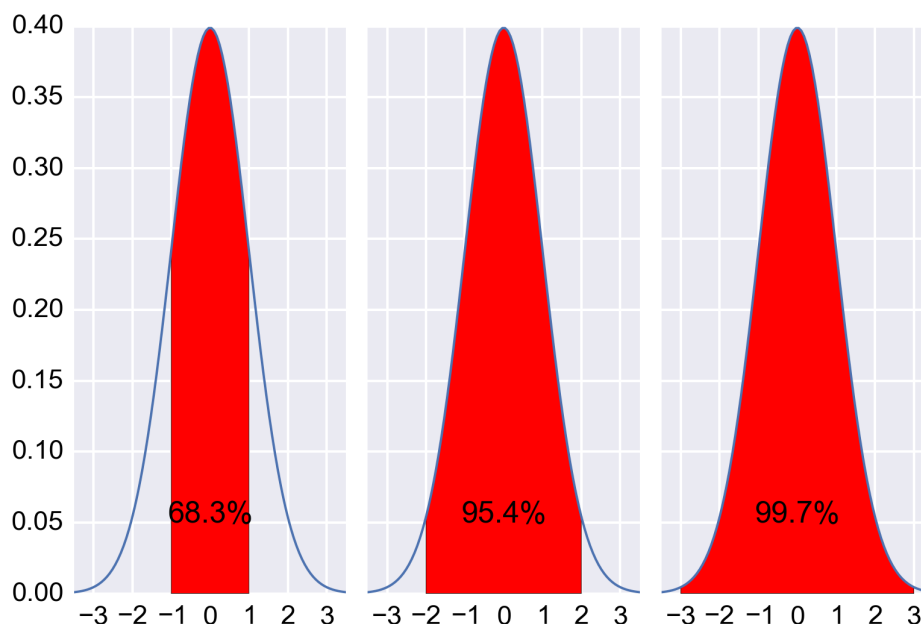


Figure 3.8: Area under ± 1 , 2, and 3 standard deviations of a normal distribution.

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

Table 3.1: Tails of a normal distribution.

- If the average man is 175 cm tall with a standard deviation of 6 cm and the average woman is 168 cm tall with a standard deviation 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 standard deviation 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 $X \in N(\mu, \sigma)$.

Note: In Python, the most elegant way of working with distribution function is a two-step procedure:

- In the first step, you create your distribution (e.g. `nd = stats.norm()`). Note that is a *distribution* (in Python parlance a "frozen distribution"), not a function yet!
- In the second step, you decide which function you want to use from this distribution, and calculate the function value for the desired x-input (e.g. `y = nd.cdf(x)`)



Code: "distributionNormal.py" (p 112) shows simple manipulations of

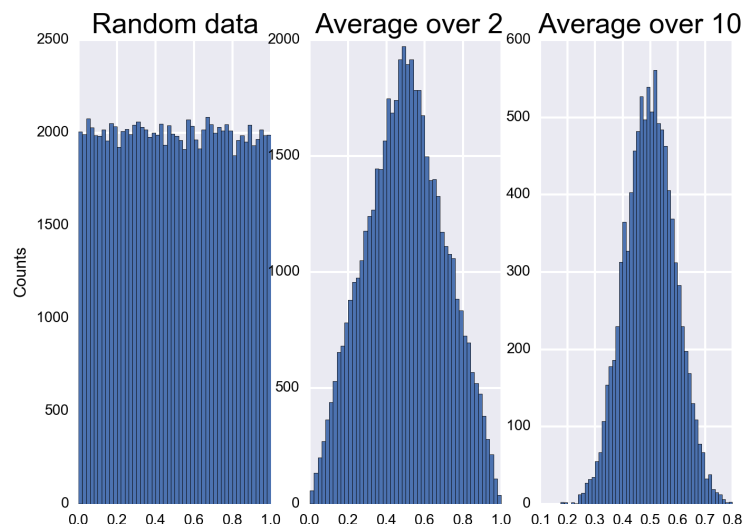


Figure 3.9: Demonstration of the "Central Limit Theorem": Left) Histogram of random data between 0 and 1. Center) Histogram of average over two datapoints.) Right) Histogram of average over 10 datapoints.

normal distribution functions.

```
In [33]: from scipy import stats
In [34]: mu = -2
In [35]: sigma = sqrt(0.5)
In [36]: myDistribution = stats.norm(mu, sigma)
In [37]: significanceLevel = 0.05
In [38]: myDistribution.ppf([significanceLevel/2, 1-significanceLevel/2])
Out[38]: array([-3.38590382, -0.61409618])
```

Example of how to calculate the interval of the PDF containing 95% of the data, for the green curve in Figure 3.6

3.2.2 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. 3.9 shows that averaging over 10 uniformly distributed data already produces a smooth, almost Gaussian distribution.



python Code: "centralLimitTheorem.py" (p 111) demonstrates that already averaging over 10 uniformly distributed datapoints produces an almost Gaussian distribution.

3.2.3 Application Example

To illustrate the ideas behind the use of distribution functions, let us go step-by-step through the analysis of the following problem:

The average weight of a newborn child in the US is 3.5 kg, with a standard deviation of 0.76 kg. If we want to check all children that are *significantly different* from the typical baby, what should we do with a child that is born with a weight of 2.6 kg?

- Find the distribution that characterizes healthy babies.

- Calculate the CDF at the interesting value ($CDF(2.6 \text{ kg}) = 0.118$).
- Interpret the result (*"If the baby is healthy, the chance that its weight deviates by at least the observed value from the mean is $2 \times 11.8\% = 23.6\%$ - This is not significant"*).

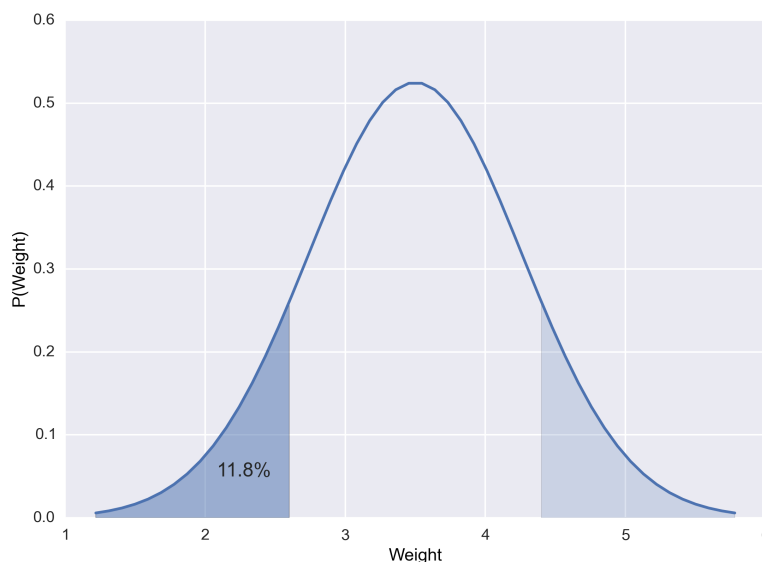


Figure 3.10: The chance that a healthy baby weighs 2.6 kg or less is 11.8% (darker blue area). The chance that the difference from the mean is that much is twice that much, as the lighter blue area must be added.

3.2.4 Other Continuous Distributions

The distributions you will encounter most frequently are:

- **Normal distribution** - the "ideal" continuous probability distribution
- **t-distribution** - for sample distributions (What you will probably use most often.)
- **χ -square distribution** - for describing variability
- **F-distribution** - for comparing variability

In the following, we will describe these distributions in more detail.

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. 3.11).

If \bar{x} is the sample mean, and s the sample standard deviation, then

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} \quad (3.7)$$

A very frequent application of the t-distribution is in the calculation of confidence intervals (Eq. 3.3.4):

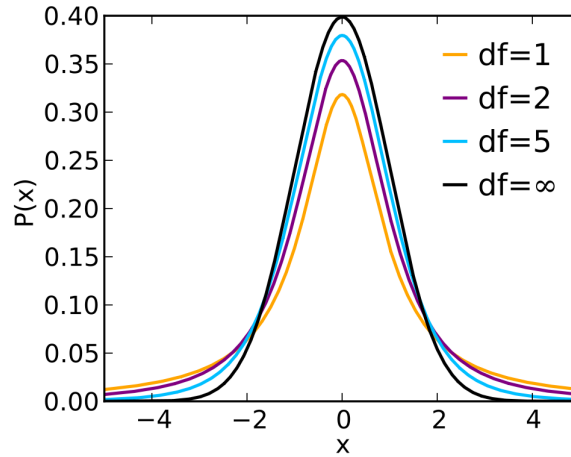


Figure 3.11: t Distribution

```

In [28]: n = 20
In [29]: alpha = 0.05
In [30]: stats.t(20).ppf(1-alpha/2)
Out[30]: 2.0859634472658364

In [31]: stats.norm.ppf(1-alpha/2)
Out[31]: 1.959963984540054

```

Calculating the t -values for confidence intervals, for $n = 20$ and $\alpha = 0.05$. For comparison, I also calculate the corresponding value from the normal distribution.

Since the t -distribution has longer tails than the normal distribution, it is much less sensitive to outliers (see Figure 3.12).

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:

$$\sum_{i=1}^n X_i^2 \in \chi_n^2 \quad (3.8)$$

Application Example

A pill producer is ordered to deliver pills with a standard deviation of $\sigma = 0.05$. From the next batch of pills you pick $n = 13$ random samples. These samples x_1, x_2, \dots, x_n have a weight of 3.04, 2.94, 3.01, 3.00, 2.94, 2.91, 3.02, 3.04, 3.09, 2.95, 2.99, 3.10, 3.02 g.

Question: is the standard deviation larger than allowed?

Answer:

Since the Chi-square distribution describes the distribution of the summed squares of random variates from a *standard normal distribution*, we have to normalize our data before we calculate the corresponding CDF-value:

$$1 - CDF_{\chi^2_{(n-1)}} \left(\sum \left(\frac{x - \bar{x}}{\sigma} \right)^2 \right) = 0.1929 \quad (3.9)$$

Interpretation: if the batch of pills is from a distribution with a standard deviation of $\sigma = 0.05$, the likelihood of obtaining a chi-square value as large or larger than the one observed

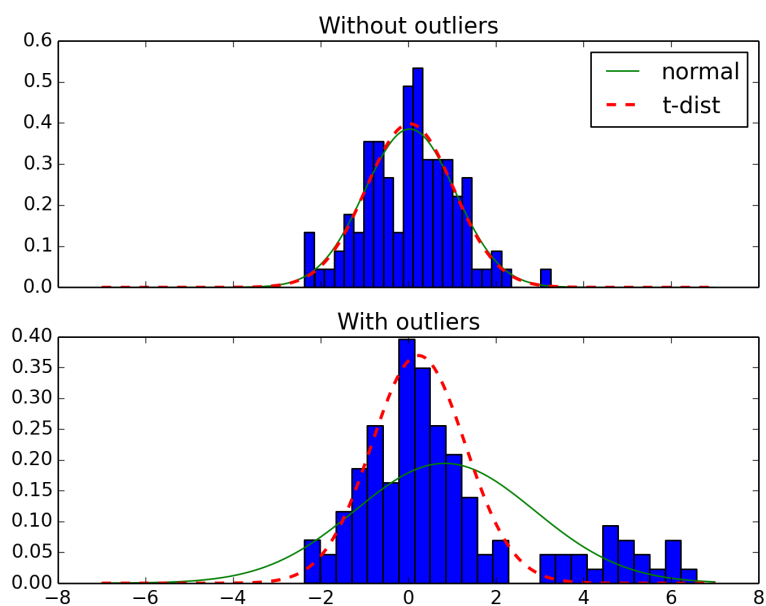


Figure 3.12: The t-distribution is much more robust against outliers than the normal distribution.

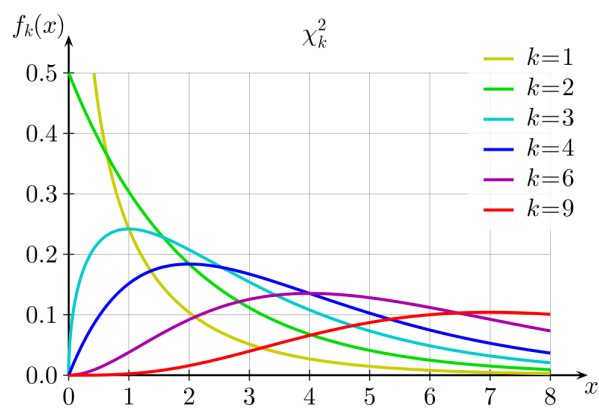


Figure 3.13: Chi-square Distribution

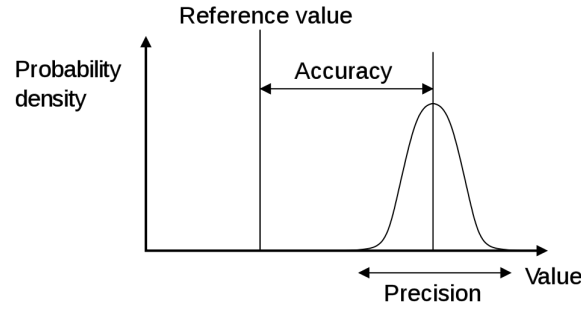


Figure 3.14: Accuracy and precision of a measurement are two different characteristics!

is about 19%, so it is not atypical. In other words, the batch matches the expected standard deviation.

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

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:

$$F(r_1, r_2) = \frac{\chi_{r_1}^2 / r_1}{\chi_{r_2}^2 / r_2} \quad (3.10)$$

where $\chi_{r_1}^2$ and $\chi_{r_2}^2$ 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 If you want to investigate whether two groups have the same variance, you have to calculate the ratio of the sample standard deviations squared:

$$F = \frac{S_x^2}{S_y^2} \quad (3.11)$$

where S_x is the sample standard deviation of the first sample, and S_y the sample standard deviation for the second sample.

Application Example

Take for example the case that you want to compare two methods to measure eye movements. The two methods can have different accuracy and different precision. With your test you want to determine if the precision of the two methods is equivalent, or if one method is more precise than the other.

When you look 20 deg to the right, you get the following results: Method 1: [20.7, 20.3, 20.3, 20.3, 20.7, 19.9, 19.9, 19.9, 20.3, 20.3, 19.7, 20.3] Method 2: [19.7, 19.4, 20.1, 18.6, 18.8, 20.2, 18.7, 19.]

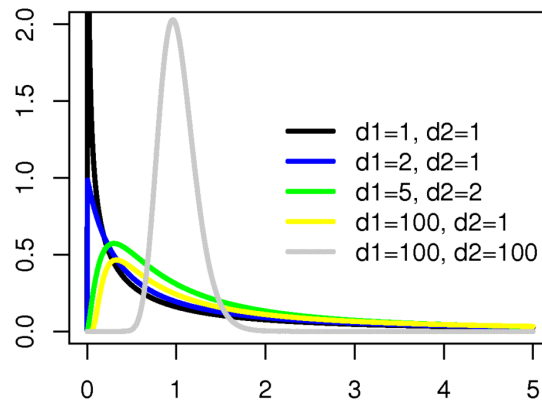


Figure 3.15: F Distribution

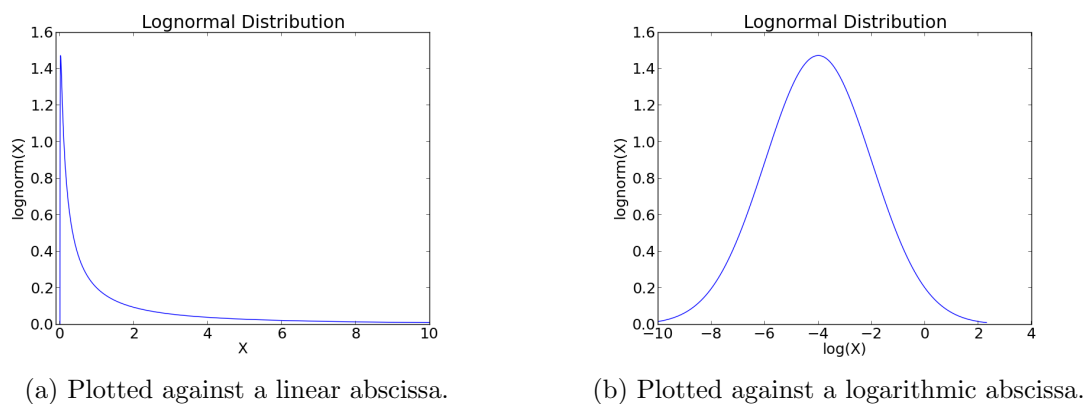


Figure 3.16: Lognormal distribution

The F statistic is $F = 0.494$, and has $n - 1$ and $m - 1$ degrees of freedom, where n and m are the number of recordings with each method. The code sample below shows that the F statistic is close to the center of the distribution, so we cannot reject the hypothesis that the two methods have the same precision.

```
In [1]: method1 = array([20.7, 20.3, 20.3, 20.3, 20.7, 19.9, 19.9,
19.9, 20.3,
20.3, 19.7, 20.3])
In [2]: method2 = array([19.7, 19.4, 20.1, 18.6, 18.8, 20.2, 18.7,
19. ])
In [3]: fval = var(method1, ddof=1)/var(method2, ddof=1)
In [4]: fd = stats.f(len(method2)-1,len(method2)-1)
In [5]: p = fd.cdf(fval)
In [6]: print p
Out[6]: 0.041
In [7]: if (p<0.025) or (p>0.975):
        print 'There is a significant difference between the two
        distributions.'
```

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 3.16).

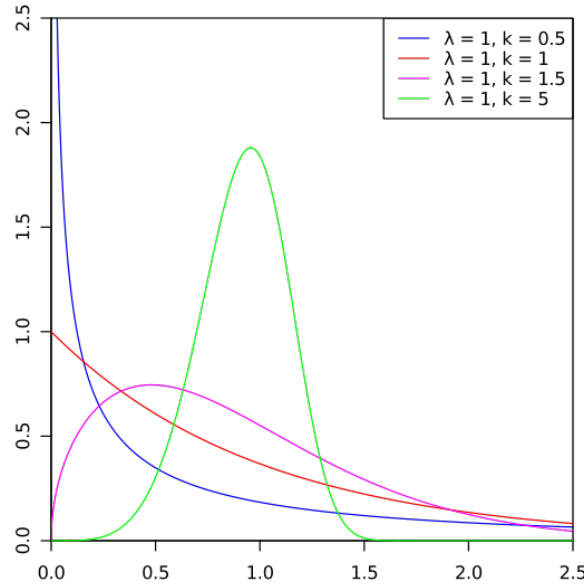


Figure 3.17: Weibull Distribution

Weibull Distribution

The Weibull distribution is the most commonly used distribution for modeling reliability data or "survival" data. It has two parameters, which allow it to handle increasing, decreasing or constant failure-rates (see Figure 3.17). It is defined as

$$f_x(x) = \begin{cases} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases} \quad (3.12)$$

where $k > 0$ is the *shape parameter* and $\lambda > 0$ is the *scale parameter* of the distribution. Its complementary cumulative distribution function is a stretched exponential function.

If the quantity x is a "time-to-failure", the Weibull distribution gives a distribution for which the failure rate is proportional to a power of time. The shape parameter, k , is that power plus one, and so this parameter can be interpreted directly as follows:

- A value of $k < 1$ indicates that the failure rate decreases over time. This happens if there is significant "infant mortality", or defective items failing early and the failure rate decreasing over time as the defective items are weeded out of the population.
- A value of $k = 1$ indicates that the failure rate is constant over time. This might suggest random external events are causing mortality, or failure.
- A value of $k > 1$ indicates that the failure rate increases with time. This happens if there is an "aging" process, or parts that are more likely to fail as time goes on.

In the field of materials science, the shape parameter k of a distribution of strengths is known as the Weibull modulus.

Exponential Distribution

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

$$f_x(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \quad (3.13)$$

The exponential PDF is shown in Figure 3.18

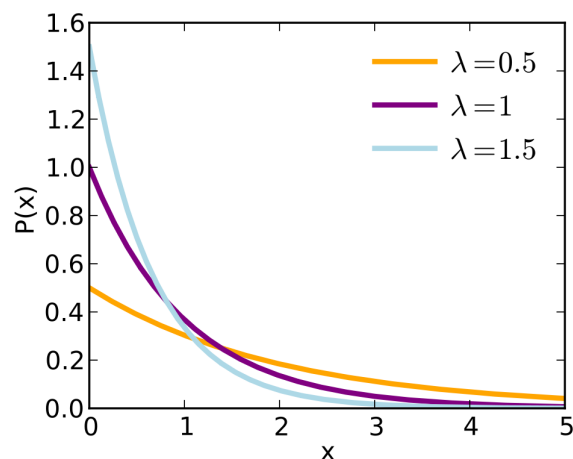


Figure 3.18: Exponential Distribution

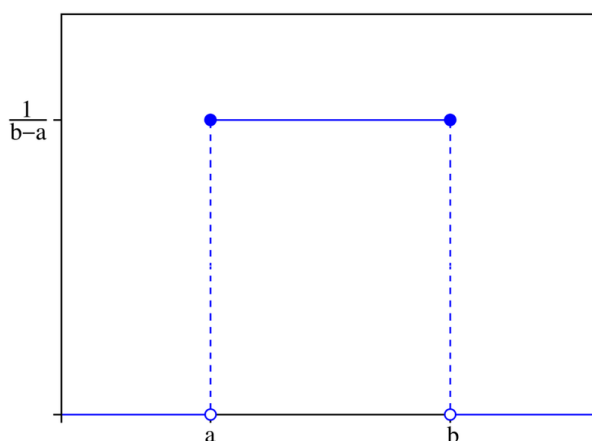


Figure 3.19: Uniform Distribution

Uniform Distribution

This is a simple one: an even probability for all data values (Figure 3.19). Not very common for real data.

Programs: Continuous Distribution Functions

Working with distribution functions in Python takes a bit to get used to. But once you get the concept, it is marvellously easy. In my opinion, the most logical way is first to define the function, with all the parameters that it requires; and then, to use the methods of this function, e.g. *pdf*, or *cdf*:

```
In [1]: from scipy import stats
In [2]: myDF = stats.norm(5,3)
In [3]: x = linspace(-5, 15, 101)
In [4]: y = myDF.pdf(x)
```



Code: "distributionContinuous.py" (p 115) shows different continuous distribution functions.

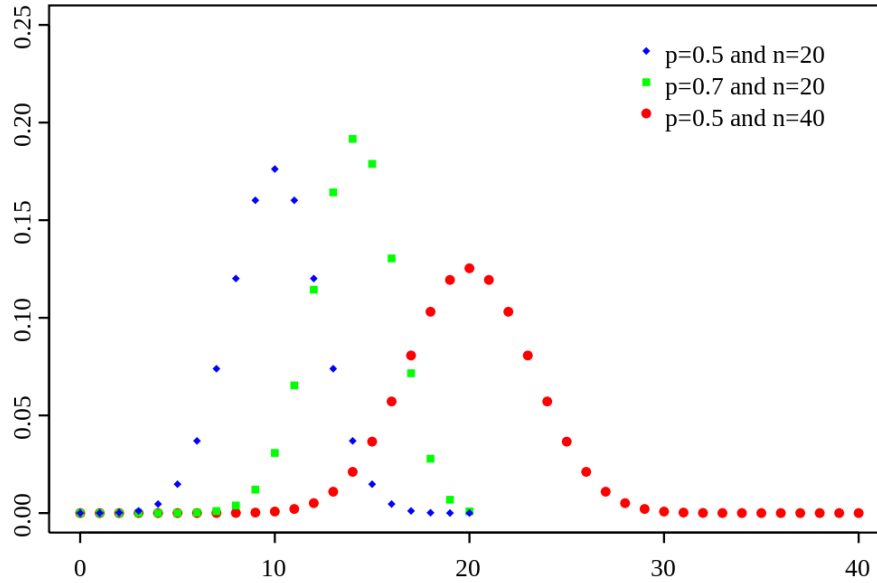


Figure 3.20: Binomial Distribution

3.2.5 Discrete Distributions

While the functions describing continuous distributions are referred to as *probability density 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 $X \sim \text{Bin}(n, p)$ or $X \sim B(n, p)$ and the probability mass function is given at $X = k$ by the equation:

$$P[X = k] = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases} \quad 0 \leq p \leq 1, \quad n \in \mathbb{N} \quad (3.14)$$

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

The binomial distribution for $n = 1$ is sometimes referred to as *Bernoulli 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).

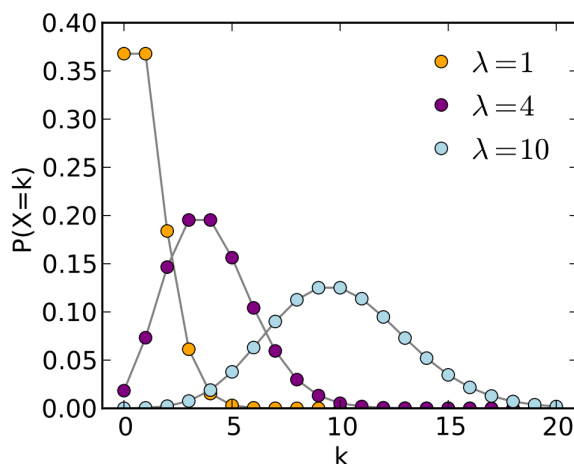


Figure 3.21: Poisson Distribution

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

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!} \quad (3.15)$$

Programs: Discrete Distribution Functions

 **python**™ **Code:** "distributionDiscrete.py" (p 117) shows different continuous distribution functions.

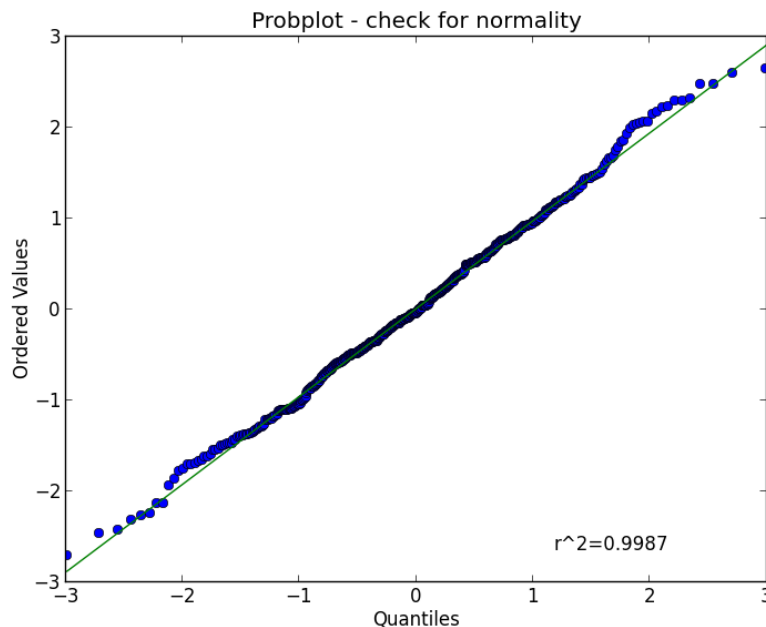


Figure 3.22: QQ-Plot, to check for normality of distribution.

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

Many statistical tests already assume that your data are normally distributed, i.e. that they are linearly related to the standard normal distribution. If you use any of these tests, you first have to check this assumption.

QQ-Plots

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 3.22).

Hypothesis Tests for Normality

To test the null hypothesis that data come from a normally distributed population, when the null hypothesis does not specify which normal distribution i.e., it does not specify the expected value and variance of the distribution, you can use the *Lilliefors test*. It is a normality test based

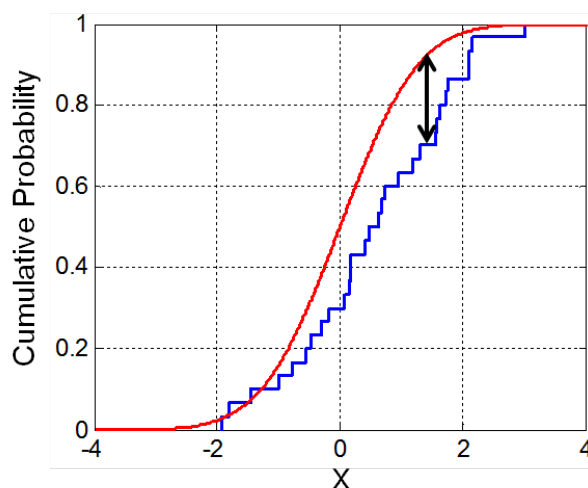


Figure 3.23: Illustration of the Kolmogorov-Smirnov statistic. Red line is CDF, blue line is an ECDF, and the black arrow is the K-S statistic (from Wikipedia).

on the *Kolmogorov–Smirnov test*, which quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples. Note that for the original Kolmogorov-Smirnov test this implies that in a test for normality, you have to standardize your sample distribution (i.e. subtract the mean, and divide by the standard deviation), and specify your reference distribution (i.e. if you want to check for normality, the normal distribution).

The Python command `stats.normaltest(x)` uses a method by D’Agostino and Pearson, which compares skew and kurtosis to that of a normal distribution; Altman mainly uses the *Shapiro-Wilk W test* [Altman(1999)]; and a number of other tests are also available.



python **Code:** "checkNormality.py" (p 119) shows how to check graphically and quantitatively if a given distribution is normal.

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 3.16.

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

$$ci = mean \pm se * t_{n,\alpha} \quad (3.16)$$

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 $\alpha = 0.975$.

3.4 Exercises

3.4.1 Python

1. Create an numpy-array, containing the data 1,2,3,...,10. Calculate mean and sample(!)-standard deviation. (Correct answer: 3.03)

3.4.2 Distributions

1. Generate and plot the Probability Density Function (PDF) of a normal distribution, with a mean of 5 and a standard deviation of 3.
2. Generate 1000 random data from this distribution.
3. Calculate the standard error of the mean of these data. (Correct answer: ca. 0.096)
4. Plot the histogram of these data.
5. From the PDF, calculate the interval containing 95% of these data. (Correct answer: [-0.88, 10.88])

3.4.3 Analysis

1. (a) Read in the data from 'Data\amstat\calcium.dat.txt'.
(b) Check for erroneous entries.
(c) Check the Alkaline Phosphatase levels for normality. Use a log-transform on the data, and re-check.

3.4.4 Continuous Distributions

1. **Normal Distribution:** Your doctor tells you that he can use hip implants for surgery even if they are 1 mm bigger or smaller than the specified size. And your financial officer tells you that you can discard 1 out of 1000 hip implants, and still make a profit.
What is the required standard deviation for the producer of the hip implants, to simultaneously satisfy both requirements? (Correct answer: $\sigma = 0.304mm$)
2. **T-Distribution:** Measuring the weight of your colleagues, you have obtained the following weights: 52, 70, 65, 85, 62, 83, 59 kg. Calculate the corresponding mean, and the 99% confidence interval for the mean. Note: with n values you have n-1 DOF for the t-distribution. (Correct answer: 68.0 +/- 17.2 kg)
3. **Chi-square Distribution:** Create 3 normally distributed datasets (mean = 1, SD = 1), with 1000 samples each. Then square them, sum them (so that you have 1000 data-points), and create a histogram with 100 bins. This should be similar to the curve for the Chi-square distribution, with 3 DOF (i.e. it should come down at the left, see figure below).
4. **F Distribution:** You have two apple trees. 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. Are the variances from the two trees different? Note: calculate the corresponding F-value, and check if the CDF for the corresponding F-distribution is < 0.025 . (Correct answer: no)

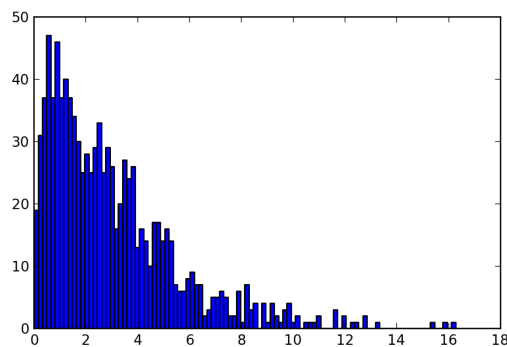


Figure 3.24: chi2 distribution with 3 degrees of freedom.

3.4.5 Discrete Distributions

1. **Binomial Distribution** "According to research, pure blue eyes in Europe approach greatest frequency in Finland, Sweden and Norway(at 72%), followed by Estonia, Denmark(69%); Latvia, Ireland(66%); Scotland(63%); Lithuania(61%); The Netherlands(58%); Belarus, England(55%); Germany(53%); Poland, Wales(50%); Russia, The Czech Republic(48%); Slovakia(46%); Belgium(43%); Austria, Switzerland, Ukraine(37%); France, Slovenia(34%); Hungary(28%); Croatia(26%); Bosnia and Herzegovina(24%); Romania(20%); Italy(18%); Serbia, Bulgaria(17%); Spain(15%); Georgia, Portugal(13%); Albania(11%); Turkey and Greece(10%). Further analysis shows that the average occurrence of blue eyes in Europe is 34%, with 50% in Northern Europe and 18% in Southern Europe."

If we have 15 Austrian students in the class-room, what ist the chance of finding 3, 6, or 10 students with blue eyes? (Correct answer: 9%, 20.1%, and 1.4%)

2. **Poisson Distribution** On the streets of Austria there were 62 fatal accidents in 2012. Assuming that those are evenly distributed, we have on average $62 / (365/7) = 1.19$ fatal accidents per week. How big is the chance that in a given week there are no, 2, or 5 accidents? (Correct answer: 30.5%, 21.5%, 0.6%)

Chapter 4

Statistical Tests

4.1 Hypothesis tests

A statistical hypothesis test is a method of statistical inference using data from a scientific study. In statistics, a result is called *statistically significant* if it has been predicted as unlikely to have occurred by chance alone, according to a pre-determined threshold probability, the *significance level*. These tests are used in determining what outcomes of a study would lead to a rejection of the *null hypothesis* for a pre-specified *level of significance*. The critical region of a hypothesis test is the set of all outcomes which cause the null hypothesis to be rejected in favor of the *alternative hypothesis*.

A typical approach is as follows: you

- state your hypothesis.
- decide which value you want to test your hypothesis on, which is called the *test statistic* T .
- compute from the observations the observed value t_{obs} of the test statistic.
- Calculate the *p-value*. This is the probability, under the null hypothesis, of sampling a test statistic at least as extreme as that which was observed.
- Reject the null hypothesis, in favor of the alternative hypothesis, if and only if the p-value is less than the significance level (the selected probability) threshold. 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.

Keep in mind that unlikely events *do* happen: In 1980, for example, a woman named Maureen Wilcox played the Rhode Island and the Massachusetts lotteries at the same time. And she hit the correct numbers for both. Unfortunately, she picked all the correct numbers for Massachusetts on her Rhode Island ticket, and all the right Rhode Island numbers on her Massachusetts ticket. Such events are unlikely - but they do happen!

Example 1: Let us compare the weight of two groups of subject. Then the *null hypothesis* is that there is *null* difference in the weight between the two groups. If a statistical comparison of the weight produces a p-value of 0.03, this means that *the probability that the null hypothesis is correct is 0.03, or 3%*. Since this probability is quite low, we say that *there is a significant difference between the weight of the two groups*.

Example 2: If we want to check the assumption that the mean value of a group is 7, then the null hypothesis would be: *"We assume that there is null difference between the mean value in our population and the value 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*. In quality control, a Type I error is called *producer risk*, because you keep a produced item despite the fact that it meets the regulatory requirements.

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. In quality control, a Type II error is called a *consumer risk*, because the consumer obtains an item that does not meet the regulatory requirements.

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 *effect size*, i.e. 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 4th is then automatically fixed.

The absolute 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*.

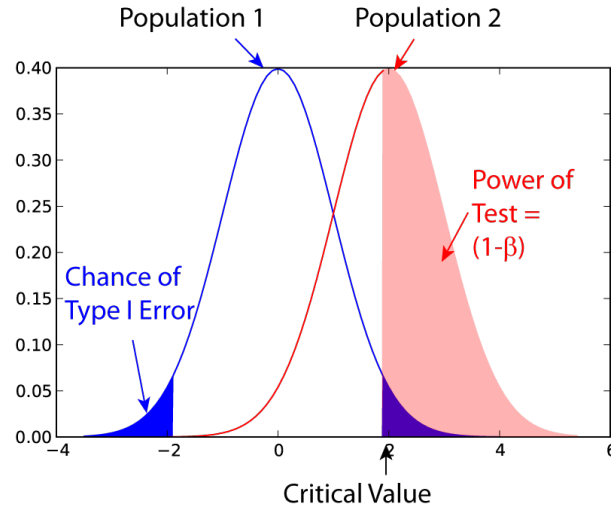


Figure 4.1: *Power* of a statistical test, for comparing the mean value of two given distributions.

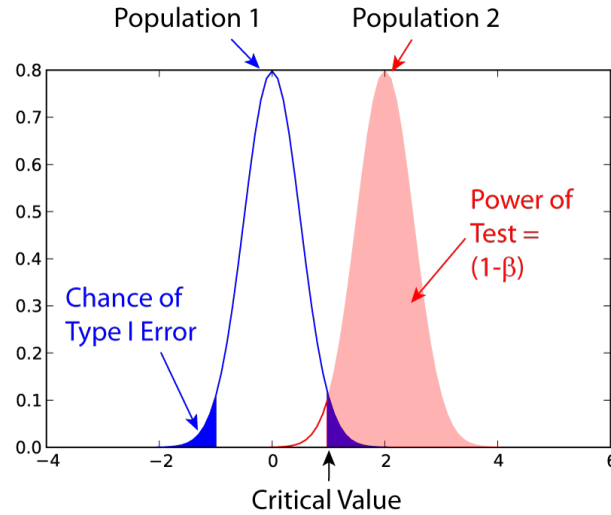


Figure 4.2: Effect of an increase in sampling size on the power of a test.

Examples for some special cases

Test on one mean: if we have the hypothesis that the data population has a mean value of x_1 and a standard deviation of σ , and the actual population has a mean value of $x_1 + D$ and the same standard deviation, we can find such a difference with a *minimum sample number* of

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

Here z is the standardized normal variable (see also chapter 3.2.1)

$$z = \frac{x - \mu}{\sigma}. \quad (4.2)$$

and $d = \frac{D}{\sigma}$ the effect size.

In words, if the real mean has a value of x_1 , we want to detect this correctly in at least $1 - \alpha\%$ of all tests; and if the real mean is shifted by D or more, we want to detect this with a likelihood of at least $1 - \beta\%$.

Test between two different populations: For finding a difference between two normally distributed means, with standard deviations of σ_1 and σ_2 , the minimum number of samples we need in each group to detect an absolute difference D is

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

Programs: SampleSize



Code: "sampleSize.py" (p 120): sample size calculation for normally distributed groups.

4.1.3 The "p-value fallacy"

p values are often used to measure evidence against a hypothesis. Unfortunately, they are often incorrectly viewed as an error probability for rejection of the hypothesis, or, even worse, as the posterior probability (i.e. after the data have been collected) that the hypothesis is true. As an example, take the case where the alternative hypothesis is that the mean is just a fraction of one standard deviation larger than the mean under the null hypothesis: in that case, a sample that produces a p-value of 0.05 may just as likely be produced if the alternative hypothesis is true as if the null hypothesis is true!

[Sellke(2001)] have investigated this question in detail, and recommend to use a "calibrated p-value" to estimate the probability of making a mistake when rejecting the null hypothesis, when the data produce a p-value p :

$$\alpha(p) = \frac{1}{1 + \frac{1}{-e p \log(p)}} \quad (4.4)$$

with $e = \exp(1)$, and \log the natural logarithm. For example, $p = 0.05$ leads to $\alpha = 0.29$, and $p = 0.01$ to $\alpha = 0.11$.

Remember, p only indicates the likelihood of obtaining a certain value for the test statistic if the null hypothesis is true - nothing else!

And keep in mind that improbable events do happen, even if not very frequently. For example, back in 1980 a woman named Maureen Wilcox bought tickets for both the Rhode Island lottery and the Massachusetts lottery. And she got the correct numbers for both lotteries. Unfortunately for her, she picked all the correct numbers for Massachusetts on her Rhode Island ticket, and all the right numbers for Rhode island on her Massachusetts ticket :(Seen statistically, the p-value for such an event would be extremely small - but it did happen anyway.

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:

- **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** = proportion of patients with positive test results who are correctly diagnosed.

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

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

- **Negative predictive value** = 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.

High prevalence		Low prevalence	
T ₊ P ₊	T ₊ P ₋	T ₊ P ₊	T ₊ P ₋
	T ₋ P ₋		T ₋ P ₋
T ₋ P ₊		T ₋ P ₊	

Figure 4.4: Effect of prevalence on PPV and NPV. "T" stands for "test", and "P" for "patient". (For comparison with below: T+P+ = TP, T-P- = TN, T+P- = FP, and T-P+ = FN)

Figure 4.5 gives a worked example:

Related calculations

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

		Patients with <u>bowel cancer</u> (as confirmed on <u>endoscopy</u>)		
		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	<u>Negative predictive value</u> = TN / (FN + TN) = 1820 / (10 + 1820) ≈ 99.5%
		<u>Sensitivity</u> = TP / (TP + FN) = 20 / (20 + 10) ≈ 67%	<u>Specificity</u> = TN / (FP + TN) = 1820 / (180 + 1820) = 91%	

Figure 4.5: Worked example. (From: Wikipedia)

- Likelihood ratio negative = $\frac{1 - \text{sensitivity}}{\text{specificity}} = \frac{1 - 0.6667}{0.91} = 0.37$

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 ROC Curve

Closely related to *Sensitivity* and *Specificity* is the *receiver operating characteristic (ROC)* curve. This is a graph displaying the relationship between the true positive rate (on the vertical axis) and the false positive rate (on the horizontal axis). The technique comes from the field of engineering, where it was developed to find the predictor which best discriminates between two given distributions. In the ROC-curve (Figure 4.6) this point is given by the value with the largest distance to the diagonal.

4.4 Common Statistical Tests for Comparing Groups

Table 4.1 gives an overview of the most common statistical tests for different combinations of data.

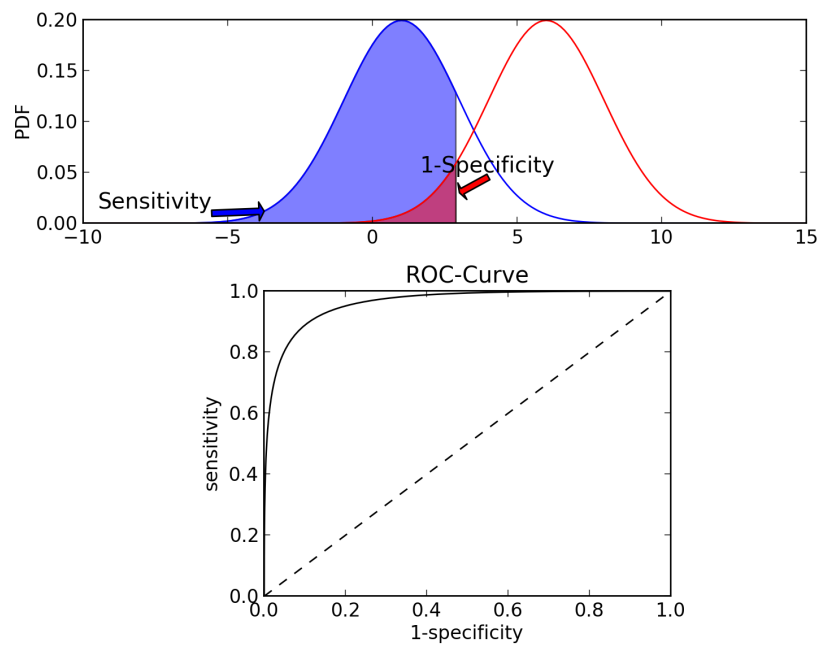


Figure 4.6: Top: Probability density functions for two distributions. Bottom: corresponding *ROC-curve*.

No. of Groups Compared	Independent Samples	Paired Samples
Groups of Nominal Data		
2 or more	Fisher's exact test or Chi-Square test	McNemar's test
Groups of Ordinal Data		
2	Mann-Whitney U test	Wilcoxon signed rank test
3 or more	Kruskal-Wallis test	Friedman test
Groups of Continuous Data		
2	Student's t-test or Mann-Whitney test	Paired t-test or Wilcoxon signed-rank test
3 or more	ANOVA or Kruskal-Wallis test	Repeated Measures ANOVA or Friedman test

Table 4.1: Typical tests for statistical problems.

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?

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



Code: "oneSample.py" (p 121): Sample analysis for one group of continuous data.

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).

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

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

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

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*.

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*.

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(-Wilcoxon) test*. Watch out, because this test is sometimes also referred to as *Wilcoxon rank-sum test*. This is different than the *Wilcoxon signed rank sum test*!



Code: "twoSample.py" (p 123): Comparison of two groups, paired and unpaired.

5.2.2 Statistical Tests vs Statistical Modeling

With the advent of cheap computing power, statistical modeling has been a booming field. This has also affected classical statistical analysis, as most problems can be viewed from two perspectives: you can either make a statistical hypothesis, and verify or falsify that hypothesis; or you can make a statistical model, and analyse the significance of the model parameters.

Let me use a classical t-test as an example.

Classical t-test

Let us take performance measurements from a racing team, on two different occasions. During Race_1, the members of the team achieve a score of [79., 100., 93., 75., 84., 107., 66., 86., 103., 81., 83., 89., 105., 84., 86., 86., 112., 112., 100., 94.], and during Race_2 [92., 100., 76., 97., 72., 79., 94., 71., 84., 76., 82., 57., 67., 78., 94., 83., 85., 92., 76., 88.].

These numbers can be generated, and a t-test comparing the two groups can be done, with the following Python commands:

```
from scipy import stats
random.seed(123)
race_1 = np.round(randn(20)*10+90)
race_2 = np.round(randn(20)*10+85)
stats.ttest_rel(race_1, race_2)
```

Statistical Modeling

```
import pandas as pd
import statsmodels.formula.api as sm
df = pd.DataFrame({'Race1': race_1, 'Race2': race_2})
result = sm.ols(formula='I(Race2-Race1) ~ 1', data=df).fit()
print result.summary()
```

5.3 Comparison of More Groups

5.3.1 Analysis of Variance - ANOVA

The idea behind the *ANalysis Of VAriance* (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!

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.

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*.

ANOVA uses traditional standardized terminology. The definitional equation of sample variance is $s^2 = \frac{1}{n-1} \sum (y_i - \bar{y})^2$, where the divisor is called the degrees of freedom (DF), the summation is called the sum of squares (SS), the result is called the mean square (MS) and the squared terms are deviations from the sample mean. ANOVA estimates 3 sample variances: a total variance based on all the observation deviations from the grand mean, an error variance based on all the observation deviations from their appropriate treatment means and a treatment variance. The treatment variance is based on the deviations of treatment means from the grand mean, the result being multiplied by the number of observations in each treatment to account for the difference between the variance of observations and the variance of means. If the null hypothesis is true, all three variance estimates are equal (within sampling error).

The fundamental technique is a partitioning of the total sum of squares SS into components related to the effects used in the model. For example, the model for a simplified ANOVA with one type of treatment at different levels.

$$SS_{\text{Total}} = SS_{\text{Error}} + SS_{\text{Treatments}} \quad (5.1)$$

The number of degrees of freedom DF can be partitioned in a similar way: one of these components (that for error) specifies a chi-squared distribution which describes the associated sum of squares, while the same is true for "treatments" if there is no treatment effect.

$$DF_{\text{Total}} = DF_{\text{Error}} + DF_{\text{Treatments}} \quad (5.2)$$

Example: one-way ANOVA

As an example, let us take the red cell folate levels ($\mu\text{g/l}$) in three groups of cardiac bypass patients given different levels of nitrous oxide ventilation (Amess et al, 1978), described in the Python code example below. I first show the result of this ANOVA test, and then explain the steps to get there.

	DF	SS	MS	F	p (>F)
C(treatment)	2	15515.76	7757.88	3.71	0.043
Residual	19	39716.09	2090.32	NaN	NaN

- 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 (MS) is the SS divided by the corresponding degrees of freedom (DF).
- The *F-test* or *variance ratio test* is used for comparing the factors of the total deviation. 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 A.14).

$$F = \frac{\text{variance between treatments}}{\text{variance within treatments}} \quad (5.3)$$

$$F = \frac{MS_{\text{Treatments}}}{MS_{\text{Error}}} = \frac{SS_{\text{Treatments}}/(I-1)}{SS_{\text{Error}}/(n_T - I)} \quad (5.4)$$

- 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 3.2.4). From the F-value, we can look up the corresponding p-value.



Code: "anovaOneway.py" (p 125): different aspects of one-way ANOVAs.

5.3.2 Multiple Comparisons

The Null hypothesis in a one-way ANOVA is that the means of all the samples are the same. So if a one-way ANOVA yields a significant result, we only know that they are *not* the same.

However, often we are not just interested in the joint hypothesis if all samples are the same, but we would also like to know for which pairs of samples the hypothesis of equal values is rejected. In this case we conduct several tests at the same time, one test for each pair of samples. (Typically, this is done with *t - tests*.)

This results, as a consequence, in a *multiple testing problem*: since we perform multiple comparison tests, we should compensate for the risk of getting a significant result, even if our null hypothesis is true. This can be done by correcting the p-values to account for this. We have a number of options to do so:

- Tukey HSD
- Bonferroni correction
- Holms correction
- ... and others ...

Tukey's Test

Tukey's test, sometimes also referred to as the *Tukey Honest Significant Difference (HSD) method*, controls for the Type I error rate across multiple comparisons and is generally considered an acceptable technique. It is based on a formula very similar to that of the t-test. In fact, Tukey's test is essentially a t-test, except that it corrects for multiple comparisons.

The formula for Tukey's test is:

$$q_s = \frac{Y_A - Y_B}{SE} \quad (5.5)$$

where Y_A is the larger of the two means being compared, Y_B is the smaller of the two means being compared, and SE is the standard error of the data in question. This q_s value can then be compared to a q value from the *studentized range distribution*, which takes into account the multiple comparisons. If the q value is larger than the critical value obtained from the distribution, the two means are said to be significantly different. Note that the studentized range statistic is the same as the t-statistic except for a scaling factor ($\text{np.sqrt}(2)$).



python™ Code: "multipleTesting.py" (p 128): this script provides an example where three treatments are compared.

Bonferroni correction

Tukey's studentized range test (HSD) is a test specific to the comparison of all pairs of k independent samples. Instead we can run t-tests on all pairs, calculate the p-values and apply one of the p-value corrections for multiple testing problems. 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),
```

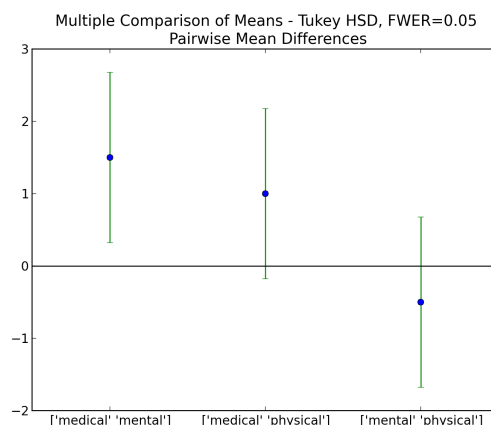


Figure 5.1: Comparing the means of multiple groups - here three different treatment options.

```
array([ 0.15,  0.9 ,  0.03]),
0.016952427508441503,
0.016666666666666666)
```

Holms correction

The Holm adjustment sequentially compares the lowest p-value with a Type I error rate that is reduced for each consecutive test. For example, if you have three groups (and thus three comparisons), this means that the first p-value is tested at the $.05/3$ level (.017), the second at the $.05/2$ level (.025), and third at the $.05/1$ level (.05). This method is generally considered superior to the Bonferroni adjustment.

5.3.3 Kruskal-Wallis test

When we compare two groups to each other, we use the *t-test* when the data are normally distributed and the non-parametric *Mann-Whitney test* otherwise. For three or more groups, the test for normally distributed data is the *ANOVA-test*; for not-normally distributed data, the corresponding test is the *Kruskal-Wallis test*. When the null hypothesis is true the test statistic for the Kruskal-Wallis test follows the *Chi squared distribution*.



python™ Code: "KruskalWallis.py" (p 131): Example of a Kruskal-Wallis test (for not normally distributed data).

5.4 Exercises

5.4.1 One or Two Groups

1. Wilcoxon signed rank sum test

The daily energy intake from 11 healthy women is [5260., 5470., 5640., 6180., 6390., 6515., 6805., 7515., 7515., 8230., 8770.] kJ. The data are clearly not normally distributed.

Is this value significantly different from the recommended value of 7725? (Correct answer: yes, $p=0.018$)

2. t-test of independent samples

In a clinic, 15 lacy patients weight [76., 101., 66., 72., 88., 82., 79., 73., 76., 85., 75., 64., 76., 81., 86.] kg, and 15 sporty patients weigh [64., 65., 56., 62., 59., 76., 66., 82., 91., 57., 92., 80., 82., 67., 54.] kg.

Are the lacy patients significantly heavier? (Correct answer: yes, $p=0.045$)

3. Kolmogorov-Smirnov test

Are the two datasets normally distributed? (Correct answer: yes, they are)

4. Mann-Whitney test

Are the lacy patients still heavier, if you check with the Mann-Whitney test? (Correct answer: yes, $p=0.039$)

5.4.2 Multiple Groups

(The following example is taken from the really good, but somewhat advanced book by AJ Dobson: "An Introduction to Generalized Linear Models")

1. Get the data

The file [https://github.com/thomas-haslwanter/statsintro/blob/master/Data/data_others/Table 6.6 Plant experiment.xls](https://github.com/thomas-haslwanter/statsintro/blob/master/Data/data_others/Table%206.6%20Plant%20experiment.xls) contains data from an experiment with plants in three different growing conditions. Get the data into Python. Hint: use the module `xlrd`

2. Perform an ANOVA

Are the three groups different? (Correct answer: yes, they are.)

3. Multiple Comparisons

Using the Tukey test, which of the pairs are different? (Correct answer: only TreatmentA and TreatmentB differ)

4. Kruskal-Wallis

Would a non-parametric comparison lead to a different result? (Correct answer: no)

Chapter 6

Tests on Discrete Data

Data can be discrete for different reasons. One is that you acquired them in a discrete way (e.g. levels in a questionnaire.) Another one is that your paradigm only gives discrete results (e.g. rolling a dice). For the analysis of such data, we can build on the tools that we have already covered in the previous chapters.

6.1 Tests on Ordinal Data

Ordinal data have clear rankings, e.g. "none - little - some - much - very much". However they are not continuous. For the analysis of such *rank ordered data* we can use rank order methods for the analysis:

Two groups When comparing two rank ordered groups, we can use the *Mann-Whitney test*
5.2.1


Three or more groups When comparing two rank ordered groups, we can use the *Kruskal-Wallis test* 5.3.3

6.2 Binomial Test

Suppose we have a board game that depends on the roll of a die and attaches special importance to rolling a 6. In a particular game, the die is rolled 235 times, and 6 comes up 51 times. If the die is fair, we would expect 6 to come up $235/6 = 39.17$ times. Is the proportion of 6's significantly higher than would be expected by chance, on the null hypothesis of a fair die?

To find an answer to this question using the *Binomial Test*, we consult the binomial distribution with $n=235$ and $p=1/6$, to determine the probability of finding exactly 51 sixes in a sample of 235 if the true probability of rolling a 6 on each trial is $1/6$. We then find the probability of finding exactly 52, exactly 53, and so on up to 235, and add all these probabilities together. In this way, we calculate the probability of obtaining the observed result (51 6s) or a more extreme result (≥ 51 6's) assuming that the die is fair. In this example, the result is 0.0265443, which indicates that observing 51 6's is unlikely (significant at the 5% level) to come from a die that is not loaded to give many 6's (one-tailed test).

Clearly a die could roll too few sixes as easily as too many and we would be just as suspicious, so we should use the two-tailed test which (for example) splits the 5% probability across the two tails.

 **python**™ **Code:** "binomial.py" (p 132): Example of a one-and two-sided binomial test.

Chapter 7

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*.

If you have only one sample group of data, the analysis options are somewhat limited. In contrast, a number of statistical tests exist for the analysis of frequency tables.

Chi-square test This is the most common type. It is a hypothesis test, which checks if the entries in the individual cells all come from the same distribution. In other words, it checks the null hypothesis H_0 that the results are independent of the row or column in which they appear. The alternative hypothesis H_a does not specify the type of association, so close attention to the data is required to interpret the information provided by the test.

Fisher's Exact Test While the chi-square test is approximate, the *Fisher's Exact Test* is an exact test. As it is computationally much more expensive and intricate than the chi-square test, it was originally used only for small sample numbers. However, in general it is now the more advisable test to use.

McNemar's Test This is a *matched pair test* for 2x2 tables.

Cochran's Q Test Cochran's Q test is an extension to the McNemar's test for related samples that provides a method for testing for differences between three or more *matched/paired* sets of frequencies or proportions. For example, if you have exactly the same samples analyzed by 3 different laboratories, and you want to check if the results are statistically equivalent, you would use this test.

7.1 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 in a population with a characteristic with probability p , the standard error of samples with this characteristic is given by

	<i>Right Handed</i>	<i>Left Handed</i>	<i>Total</i>
<i>Males</i>	43	9	52
<i>Females</i>	44	4	48
<i>Total</i>	87	13	100

Table 7.1: Example of a frequency table

	<i>Right Handed</i>	<i>Left Handed</i>	<i>Total</i>
<i>Males</i>	45.2	6.8	52
<i>Females</i>	41.8	6.2	48
<i>Total</i>	87	13	100

Table 7.2: Corresponding expected values for Table 7.1

$$se(p) = \sqrt{p(1-p)/n} \quad (7.1)$$

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.

7.2 Frequency Tables

7.2.1 Chi-square Test

The chi-square test is based on a test statistic that measures the divergence of the observed data from the values that would be expected under the null hypothesis of no association. This requires calculation of the expected values based on the data. When n is the total number of observations included in the table, the expected value for each cell in a two-way table is

$$expectedFrequency = \frac{RowTotal * ColumnTotal}{n} \quad (7.2)$$

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

$$V = \sum_i \frac{(o_i - e_i)^2}{e_i} \approx \chi_f^2 \quad (7.3)$$

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

Assumptions

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

- 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 $e_i \geq 5$.

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

Degrees of Freedom

The degrees of freedom can be computed by the numbers of absolute observed frequencies which can be chosen freely. For example, only one cell of a 2x2 table with the sums at the side and bottom needs to be filled, and the others can be found by subtraction. In general, an $r \times c$ table has $df = (r - 1) \times (c - 1)$ degrees of freedom. We know that the sum of absolute expected frequencies is

$$\sum_i o_i = n \quad (7.4)$$

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 1

The Python command `stats.chi2.contingency` returns, the χ^2 -value, the p-value, the degrees of freedom, and the expected values. For the example data in Table 7.1, the results are $\chi^2 = 1.1, p = 0.3, df = 1$. In other words, there is no indication that there is a difference in left-handed people vs right-handed people between males and females.

Example 2

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

$$V = \frac{(n_- - n/2)^2}{n/2} + \frac{(n_+ - n/2)^2}{n/2} \approx \chi_1^2 \quad (7.5)$$

.

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.

7.2.2 Fisher's Exact Test

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. It is

	B		<i>Totals</i>
	1	0	
A 1	a	b	a+b
0	c	d	c+d
<i>Totals</i>	a+c	b+d	N=a+b+c+d

Table 7.3: General Structure of 2x2 Frequency Tables

called "exact" because the significance of the deviation from a null hypothesis can be calculated exactly, rather than relying on an approximation that becomes exact in the limit as the sample size grows to infinity, as with many statistical tests.

Fisher is said to have devised the test following a comment from Dr Muriel Bristol, who claimed to be able to detect whether the tea or the milk was added first to her cup. The test is useful for categorical data that result from classifying objects in two different ways; it is used to examine the significance of the association (contingency) between the two kinds of classification. So in Fisher's original example, one criterion of classification could be whether milk or tea was put in the cup first; the other could be whether Dr Bristol thinks that the milk or tea was put in first. We want to know whether these two classifications are associated - that is, whether Dr Bristol really can tell whether milk or tea was poured in first. Most uses of the Fisher test involve, like this example, a 2 x 2 contingency table. The p-value from the test is computed as if the margins of the table are fixed, i.e. as if, in the tea-tasting example, Dr Bristol knows the number of cups with each treatment (milk or tea first) and will therefore provide guesses with the correct number in each category. As pointed out by Fisher, this leads under a null hypothesis of independence to a hypergeometric distribution of the numbers in the cells of the table.

In using the test, you have to decide if you want to use a one-tailed test or a two-tailed test. The former one looks for the probability to find a distribution as extreme or more extreme as the observed one. The latter one (which is the default in python) also considers tables as extreme in the opposite direction.

7.2.3 McNemar's Test

Although the McNemar test bears a superficial resemblance to a test of categorical association, as might be performed by a 2x2 chi-square test or a 2x2 Fisher exact probability test, it is doing something quite different. The test of association examines the relationship that exists among the cells of the table. The McNemar test examines the difference between the proportions that derive from the marginal sums of the table (see Table 7.3): $p_A = (a + b)/N$ and $p_B = (a + c)/N$. The question in the McNemar test is: do these two proportions, p_A and p_B , significantly differ? And the answer it receives must take into account the fact that the two proportions are not independent. The correlation of p_A and p_B is occasioned by the fact that both include the quantity a in the upper left cell of the table.

McNemar's test can be used for example in studies in which patients serve as their own control, or in studies with "before and after" design.

Example

In the following example, a researcher attempts to determine if a drug has an effect on a particular disease. Counts of individuals are given in the table, with the diagnosis (disease: present or absent) before treatment given in the rows, and the diagnosis after treatment in the columns. The test requires the same subjects to be included in the before-and-after measurements (matched pairs).

	<i>After: present</i>	<i>After: absent</i>	<i>Row total</i>
<i>Before: present</i>	101	121	222
<i>Before: absent</i>	59	33	92
<i>Column total</i>	160	154	314

Table 7.4: McNemar's Test: example

In this example, the null hypothesis of "marginal homogeneity" would mean there was no effect of the treatment. From the above data, the McNemar test statistic with Yates's continuity correction is

The general solution for the McNemar's test is

$$\chi^2 = \frac{(|b - c| - \text{correctionFactor})^2}{b + c}. \quad (7.6)$$

For small number of sample numbers the *correctionFactor* should be 0.5 (*Yates's correction*) or 1.0 (*Edward's correction*). (For $b + c < 25$, the binomial calculation should be performed, and indeed, most software packages simply perform the binomial calculation in all cases, since the result then is an exact test in all cases.) Using Yates's correction, we get

$$\chi^2 = \frac{(|121 - 59| - 0.5)^2}{121 + 59} \quad (7.7)$$

has the value 21.01, which is extremely unlikely from the distribution implied by the null hypothesis. Thus the test provides strong evidence to reject the null hypothesis of no treatment effect.

7.2.4 Cochran's Q Test

Cochran's Q test is a hypothesis test where the response variable can take only two possible outcomes (coded as 0 and 1). It is a non-parametric statistical test to verify if k treatments have identical effects. Cochran's Q test should not be confused with *Cochran's C test*, which is a variance outlier test.

Example

12 subjects are asked to perform 3 tasks. The outcome of each task is *success* or *failure*. The results are coded 0 for *failure* and 1 for *success*. In the example, subject 1 was successful in task 2, but failed tasks 1 and 3 (see Table ??).

The null hypothesis for the Cochran's Q test is that there are no differences between the variables. If the calculated probability p is below the selected significance level, the null-hypothesis is rejected, and it can be concluded that the proportions in at least 2 of the variables are significantly different from each other. For our example (Table ??), the analysis of the data provides *Cochran's Q* = 8.6667 and a significance of $p = 0.013$. In other words, at least one of the three Tasks is easier or harder than the others.

7.3 Analysis Programs

With computers, the computational steps are trivial:



Code: "compGroups.py" (p 133): Analysis of categorical data.

<i>Subject</i>	<i>Task 1</i>	<i>Task 2</i>	<i>Task 3</i>
1	0	1	0
2	1	1	0
3	1	1	1
4	0	0	0
5	1	0	0
6	0	1	1
7	0	0	0
8	1	1	0
9	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	1	0

Table 7.5: Cochran's Q Test: Success or failure for 12 subjects on 3 tasks.

7.4 Exercises

7.4.1 Fisher's Exact Test - The Tea Experiment

At a party, a lady claimed to be able to tell whether the tea or the milk was added first to a cup. Fisher proposed to give her eight cups, four of each variety, in random order. One could then ask what the probability was for her getting the number she got correct, but just by chance.

The experiment provided the Lady with 8 randomly ordered cups of tea - 4 prepared by first adding milk, 4 prepared by first adding the tea. She was to select the 4 cups prepared by one method. (This offered the Lady the advantage of judging cups by comparison.)

The null hypothesis was that the Lady had no such ability.

Calculate if the claim of the lady is supported if she gets three out of the four pairs correct. (Correct answer: No. If she gets three correct, that chance that a selection of "three or greater" was random is 0.243. She needs to get all four correct, if we set the rejection threshold at 0.05)

Chapter 8

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)*.

8.1 Correlation

8.1.1 Correlation Coefficient

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

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (8.1)$$

With

$$s_{xy} = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{n - 1} \quad (8.2)$$

and s_x, s_y the sample standard deviations of the x and y values, respectively, the Equation 8.1 can also be written as

$$r = \frac{s_{xy}}{s_x \cdot s_y}. \quad (8.3)$$

Pearson's correlation coefficient, sometimes also referred to as *population correlation coefficient* or *sample correlation*, can take any value from -1 to +1. Examples are given in Figure 8.2. Note that the formula for the correlation coefficient is symmetrical between x and y .

8.1.2 Coefficient of determination

The *coefficient of determination* or R^2 is the square of the correlation. It is easier to interpret than the correlation coefficient r : Values of R^2 close to 1 are good, values close to 0 are poor. To explain the interpretation of R^2 , let us look at the math more formally:

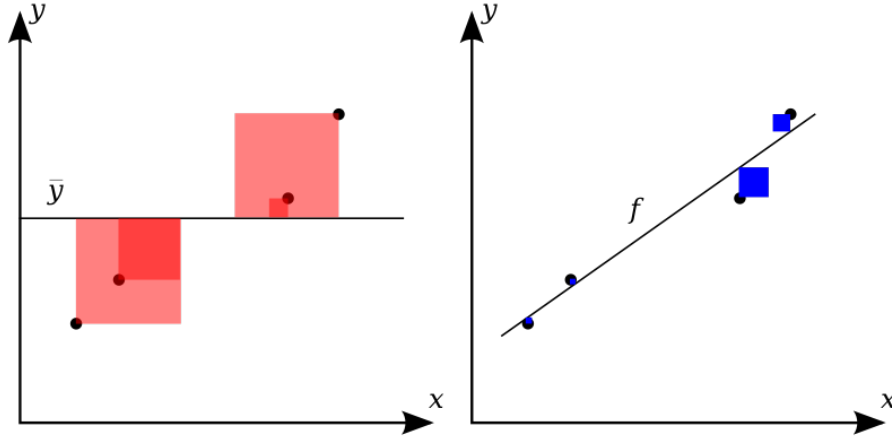


Figure 8.1: The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of R^2 is to one. The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value (from Wikipedia)

A data set has values y_i , each of which has an associated modelled value f_i (also sometimes referred to as \hat{y}_i). Here, the values y_i are called the *observed values** and the modelled values f_i are sometimes called the *predicted values*.

In the following \bar{y} is the mean of the observed data:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (8.4)$$

where n is the number of observations.

The "variability" of the data set is measured through different sums of squares:

$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$, the total sum of squares (proportional to the sample variance);

$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2$, the regression sum of squares, also called the explained sum of squares.

$SS_{\text{res}} = \sum_i (y_i - f_i)^2$, the sum of squares of residuals, also called the residual sum of squares.

The notations SS_R and SS_E should be avoided, since in some texts their meaning is reversed to "Residual sum of squares" and "Explained sum of squares", respectively.

The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}. \quad (8.5)$$

Relation to unexplained variance

In a general form, R^2 can be seen to be related to the unexplained variance, since the second term compares the unexplained variance (variance of the model's errors) with the total variance (of the data). See fraction of variance unexplained.

Adjusted R^2

For multiple regression, the *adjusted R^2* value (written as \bar{R}^2) is often used instead of R^2 :

$$\bar{R}^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1} \quad (8.6)$$

where n is the sample size and p is the number of independent variables.

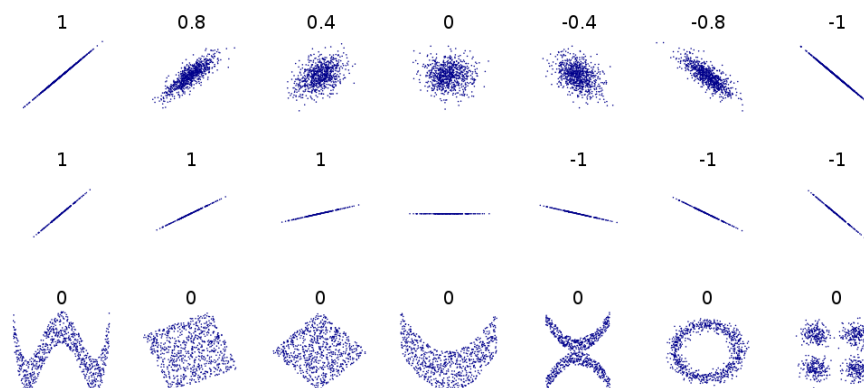


Figure 8.2: Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Note that the correlation reflects the non-linearity and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). 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)

Examples

How large R^2 or \bar{R}^2 must be to be considered good depends on the discipline. They are usually expected to be larger in the physical sciences than it is in biology or the social sciences. In finance or marketing, it also depends on what is being modeled.

Caution: the sample correlation and R^2 are misleading if there is a nonlinear relationship between the independent and dependent variables!

8.1.3 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* is also a rank correlation coefficient, measuring the association between two measured quantities. It is harder to calculate than Spearman's rho, but it has been argued that confidence intervals for Spearman's rho are less reliable and less interpretable than confidence intervals for Kendall's tau-parameters.

8.2 Regression

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

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

$$y_i = k * x_i + d + \epsilon_i \quad (8.7)$$

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

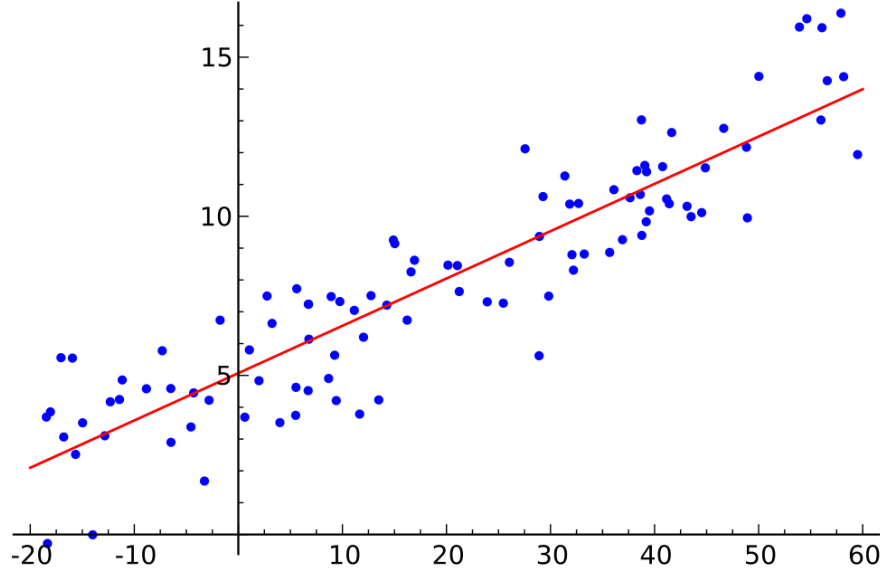


Figure 8.3: Linear regression. (From Wikipedia)

symmetrical: it is assumed that the x -values are known exactly, and that all the variability lies in the residuals.

8.2.1 Introduction

¹ Given a data set $\{y_i, x_{i1}, \dots, x_{ip}\}_{i=1}^n$ 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

$$y_i = \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \epsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \epsilon_i, \quad i = 1, \dots, n, \quad (8.8)$$

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

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \quad (8.9)$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}. \quad (8.10)$$

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

¹This section has been taken from Wikipedia

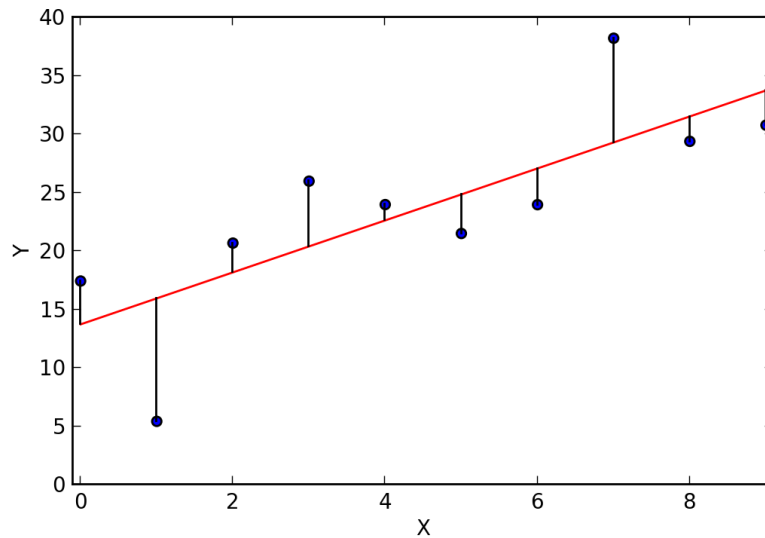


Figure 8.4: Best-fit linear regression line (red) and residuals (black).

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 $i = 1, \dots, n$. 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.
- β 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.8.8, we have a *simple linear regression*, corresponding to Eq.8.7. If $i > 1$ we talk about *multilinear regression* or *multiple linear regression*.

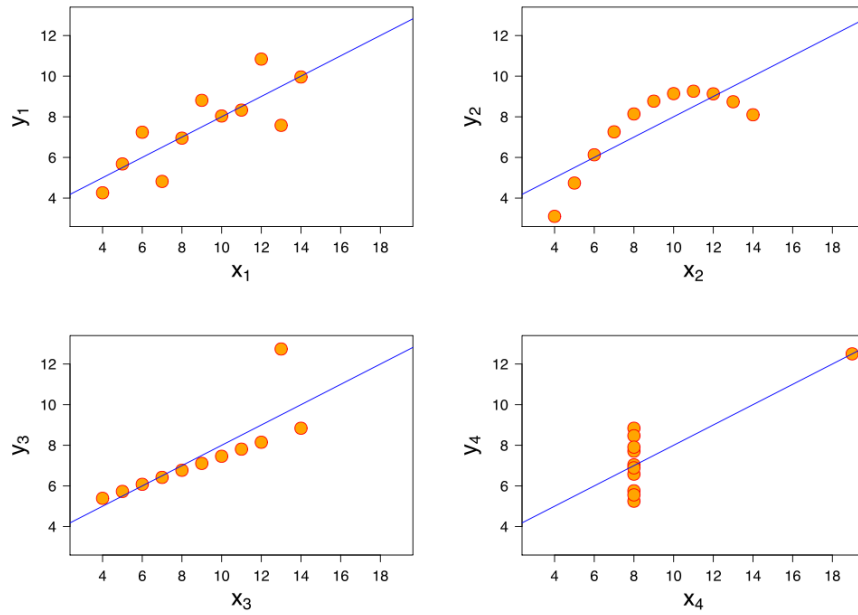


Figure 8.5: The sets in the *Anscombe's quartet* have the same linear regression line but are themselves very different.

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 :


$$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i, \quad (8.11)$$

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 : $h_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i$.

8.2.2 Assumptions

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

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

 **python**™ **Code:** "multivariate.py" (p 136): Analysis of multivariate data (regression, correlation).

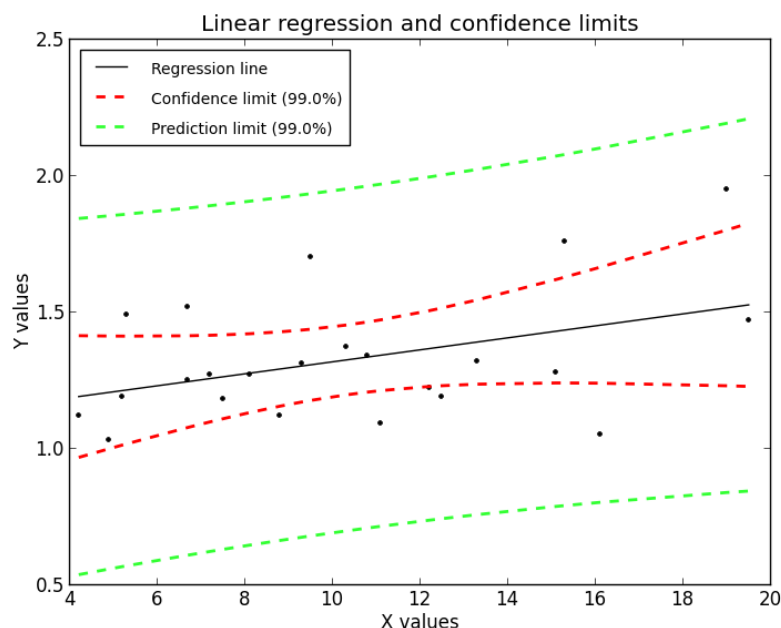


Figure 8.6: Regression, with confidence intervals for the mean, as well as for the predicted data. The red dotted line shows the confidence interval for the mean; and the green dotted line the confidence interval for predicted data. (This can be compared to the standard error and the standard deviation for a population.)

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* A.20. The output of this program is shown in Figure 8.6. This program also shows how Python programs intended for distribution should be documented.

 **python** Code: "fitLine.py" (p 138): Linear regression fit.

8.3 Exercises

1. Correlation

Read in the data for the average yearly temperature at the Sonnblick, from https://github.com/thomas-haslwanter/statsintro/blob/master/Data/data_others/AvgTemp.xls Calculate the Pearson and Spearman correlation, and Kendall's tau, for the temperature vs. year.

2. Regression

For the same data, calculate the yearly increase in temperature, assuming a linear increase with time. Is this increase significant?

3. Normality Check

For the data from the regression model, check if the model is ok by testing if the residuals are normally distributed (e.g. by using the Kolmogorov-Smirnov test)

Chapter 9

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 *Kruskal-Wallis Test* has to be used.

What should we do if we have paired data?


If we have matched pairs for two groups, and the differences are not normally distributed, we can use the *Wilcoxon signed rank sum test*. The rank test for more than two groups of matched data is the *Friedman test*.¹

An example for the application of the Friedman test: Ten professional piano players are blindfolded, and are asked to judge the quality of three different pianos. Each player rates each piano on a scale of 1 to 10 (1 being the lowest possible grade, and 10 the highest possible grade). The null hypothesis is that all three pianos rate equally. To test the null hypothesis, the Friedman test is used on the ratings of the ten piano players.

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*.

9.1 Two-way ANOVA

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".

 **python** Code: "anovaTwoway.py" (p 141): Two-way Analysis of Variance (ANOVA).


	df	sum_sq	mean_sq	F	PR(>F)
C(fetus)	2	324.00	162.00	2113.10	1.05e-27
C(observer)	3	1.19	0.39	5.21	6.497-03
C(fetus):C(observer)	6	0.56	0.09	1.22	3.29e-01

¹It may be worth mentioning that Thom Baguley suggested the following: Where one-way repeated measures ANOVA is not appropriate, rank transformation followed by ANOVA will provide a more robust test with greater statistical power than the Friedman test.

Residual	24	1.84	0.07	NaN	NaN
----------	----	------	------	-----	-----

9.2 Multilinear Regression

If you have truly independent variables, *multilinear regression* is a straightforward 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.

 **python**™ **Code:** "mult_regress.py" (p 142): Multiple regression example.

Chapter 10

Statistical Models

10.1 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

$$y \sim x + a + b + a : b \quad (10.1)$$

The symbols in Table 10.1 are used on the right hand side to denote different interactions. A complete set of the description is found under [patsy(2013)].

10.1.1 Design Matrix

Definition

A very general definition of a regression model is the following:

$$y = f(x, \epsilon) \quad (10.2)$$

In the case of a linear regression model, the model can be rewritten as:

$$y = X\beta + \epsilon, \quad (10.3)$$

Y is a vector of dimension $(n \times 1)$ and is called the *endogenous variable*, the *design matrix* X is a matrix of dimension $(n \times k)$ where each column is an explanatory variable, and ϵ is the error term. β is the vector of dimension $(k \times 1)$ and contains the parameters we want to estimate.

Operator	Meaning
\sim	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 10.1: Formula syntax

Examples

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

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad (10.4)$$

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

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \\ 1 & x_6 \\ 1 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \quad (10.5)$$

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

$$y_i = \beta_0 + \beta_1 w_i + \beta_2 x_i + \epsilon_i \quad (10.6)$$

This model can be written in matrix terms as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & w_1 & x_1 \\ 1 & w_2 & x_2 \\ 1 & w_3 & x_3 \\ 1 & w_4 & x_4 \\ 1 & w_5 & x_5 \\ 1 & w_6 & x_6 \\ 1 & w_7 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \quad (10.7)$$

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

$$y_{ij} = \mu_i + \epsilon_{ij} \quad (10.8)$$

which can be written

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \quad (10.9)$$

It should be emphasized that in this model μ_i represents the mean of the i th 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:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij} \quad (10.10)$$

with the constraint that τ_1 is zero.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu \\ \tau_2 \\ \tau_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \quad (10.11)$$

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.

10.1.2 Example: Program Effectiveness



Code: "regSpector.py" (p 144): Estimation of a linear regression model with statsmodels, using the Spector and Mazzeo (1980) data set.

10.2 Linear Regression Analysis with Python

The following is based on the blog of Connor Johnson.

We will use Python to explore measures of fit for linear regression: the coefficient of determination (R^2), hypothesis tests (F, t, Omnibus), AIC, BIC, and other measures.

First we will look at a small data set from DASL library, regarding the correlation between tobacco and alcohol purchases in different regions of the United Kingdom. The interesting feature of this data set is that Northern Ireland is reported as an outlier. Notwithstanding, we will use this data set to describe two tools for calculating a linear regression. We will alternatively use the *statsmodels* and *sklearn* modules for calculating the linear regression, while using *pandas* for data management, and *matplotlib* for plotting. To begin, we will import the modules, get the data into Python, and have a look at them:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
from sklearn.linear_model import LinearRegression
from scipy import stats

data_str = '''Region Alcohol Tobacco
North 6.47 4.03
Yorkshire 6.13 3.76
Northeast 6.19 3.77
East_Midlands 4.89 3.34
West_Midlands 5.63 3.47
East_Anglia 4.52 2.92
Southeast 5.89 3.20
Southwest 4.79 2.71
```

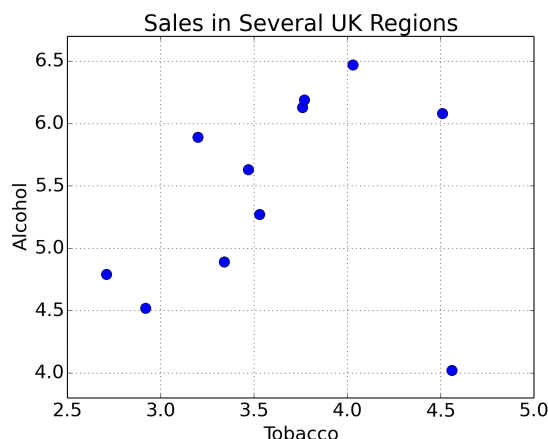


Figure 10.1: Sales of Alcohol vs Tobacco in the UK. We notice that there seems to be a linear trend, and one outlier, which corresponds to North Ireland.

```
Wales 5.27 3.53
Scotland 6.08 4.51
Northern_Ireland 4.02 4.56'''

# Read in the data. Note that for Python 2.x, you have to change the "import
" statement
from io import StringIO
df = pd.read_csv(StringIO(data_str), sep=r'\s+')

# Plot the data
df.plot('Tobacco', 'Alcohol', style='o')
plt.ylabel('Alcohol')
plt.title('Sales in Several UK Regions')
plt.show()
```

Fitting the model, leaving the outlier for the moment away is then very easy:

```
result = smf.ols('Alcohol ~ Tobacco', df[:-1]).fit()
print(result.summary())
```

Note that using the formula API from statsmodels, an intercept is automatically added. This gives us

OLS Regression Results						
=====						
Dep. Variable:	Alcohol	R-squared:	0.615			
Model:	OLS	Adj. R-squared:	0.567			
Method:	Least Squares	F-statistic:	12.78			
Date:	Sun, 27 Apr 2014	Prob (F-statistic):	0.00723			
Time:	13:19:51	Log-Likelihood:	-4.9998			
No. Observations:	10	AIC:	14.00			
Df Residuals:	8	BIC:	14.60			
Df Model:	1					
=====						
	coef	std err	t	P> t	[95.0% Conf. Int.]	

Intercept	2.0412	1.001	2.038	0.076	-0.268	4.350
Tobacco	1.0059	0.281	3.576	0.007	0.357	1.655
=====						
Omnibus:	2.542	Durbin-Watson:		1.975		
Prob(Omnibus):	0.281	Jarque-Bera (JB):		0.904		
Skew:	-0.014	Prob(JB):		0.636		
Kurtosis:	1.527	Cond. No.		27.2		

And now we have a very nice table of mostly meaningless numbers. I will go through and explain each one. The left column of the first table is mostly self explanatory. The degrees of freedom of the model are the number of predictor, or explanatory variables. The degrees of freedom of the residuals is the number of observations minus the degrees of freedom of the model, minus one (for the offset).

Most of the values listed in the summary are available via the `result` object. For instance, the R^2 value is obtained by `result.rsquared`. If you are using IPython, you may type `result.` and hit the TAB key, and a list of attributes for the `result` object will drop down.

10.2.1 Model Results

Definitions for Regression with Intercept

n is the number of observations, p is the number of regression parameters. For example, if you fit a straight line, $k = 2$. In the following \hat{y}_i will indicate the fitted model values, and \bar{y} will indicate the mean.

- $SSM = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$ is the *Sum of Square for Model*, or the sum of squares for the regression.
- $SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ is the *sum of Squares for Error*, or the sum of squares for the residuals.
- $SST = \sum_{i=1}^n (y_i - \bar{y})^2$ is the *Sum of Squares Total*, and is equivalent to the sample variance multiplied by $n - 1$.

For multiple regression models, $SSM + SSE = SST$

- $DFM = k - 1$ is the (*Corrected*) *Degrees of Freedom for Model*. (The "-1" comes from the fact that we are only interested in the correlation, not in the absolute offset of the data.)
- $DFE = n - k$ is the *Degrees of Freedom for Error*
- $DFT = n - 1$ is the (*Corrected*) *Degrees of Freedom Total*. The Horizontal line regression is the null hypothesis model.

For multiple regression models with intercept, $DFM + DFE = DFT$.

- $MSM = SSM/DFM$: *Mean of Squares for Model*
- $MSE = SSE/DFE$: *Mean of Squares for Error*. MSE is an unbiased estimate for σ^2 for multiple regression models.
- $MST = SST/DFT$: *Mean of Squares Total*, which is the sample variance of the y-variable.

The R^2 Value

The R^2 value indicates the proportion of variation in the y-variable that is due to variation in the x-variables. For simple linear regression, the R^2 value is the square of the sample correlation r_{xy} . For multiple linear regression with intercept (which includes simple linear regression), the R^2 value is defined as

$$R^2 = \frac{SSM}{SST} \quad (10.12)$$

10.2.2 The *adjusted* R^2 Value

Many researchers prefer the adjusted \bar{R}^2 value (Eq 8.6), which is penalized for having a large number of parameters in the model:

Here is the logic behind the definition of \bar{R}^2 : R^2 is defined as $R^2 = 1 - SSE/SST$ or $1 - R^2 = SSE/SST$. To take into account the number of regression parameters p , define the *adjusted R-squared* value as

$$1 - \bar{R}^2 = \frac{\text{Variance for Error}}{\text{Variance Total}} \quad (10.13)$$

where (*Sample*) *Variance for Error* is estimated by $SSE/DFE = SSE/(n - k)$, and (*Sample*) *Variance Total* is estimated by $SST/DFT = SST/(n - 1)$. Thus,

$$1 - \bar{R}^2 = \frac{SSE/(n - k)}{SST/(n - 1)} \quad (10.14)$$

$$= \frac{SSE}{SST} \frac{n - 1}{n - k} \quad (10.15)$$

so

$$\bar{R}^2 = 1 - \frac{SSE}{SST} \frac{n - 1}{n - k} \quad (10.16)$$

$$= 1 - (1 - R^2) \frac{n - 1}{n - k} \quad (10.17)$$

The F-test

If t_1, t_2, \dots, t_m are independent, $N(0, \sigma^2)$ random variables, then $\sum_{i=1}^m \frac{t_i^2}{\sigma^2}$ is a χ^2 (chi-squared) random variable with m degrees of freedom.

For a multiple regression model with intercept,

$$Y_j = \alpha + \beta_1 X_{1j} + \dots + \beta_n X_{nj} + \epsilon_j = \alpha + \sum_{i=1}^n \beta_i X_{ij} + \epsilon_j = E(Y_j|X) + \epsilon_j \quad (10.18)$$

we want to test the following null hypothesis and alternative hypothesis:

H_0 : $\beta_1 = \beta_2 = \dots = \beta_n = 0$

H_1 : $\beta_j \neq 0$, for at least one value of j

This test is known as the overall *F-test for regression*.

It can be shown that if H_0 is true and the residuals are unbiased, homoscedastic, independent, and normal (see section 10.3):

1. SSE/σ^2 has a χ^2 distribution with DFE degrees of freedom.
2. SSM/σ^2 has a χ^2 distribution with DFM degrees of freedom.
3. SSE and SSM are independent random variables.

If u is a χ^2 random variable with n degrees of freedom, v is a χ^2 random variable with m degrees of freedom, and u and v are independent, then if $F = \frac{u/n}{v/m}$ has an F distribution with (n, m) degrees of freedom.

If H_0 is true,

$$F = \frac{(SSM/\sigma^2)/DFM}{(SSE/\sigma^2)/DFE} = \frac{SSM/DFM}{SSE/DFE} = \frac{MSM}{MSE}, \quad (10.19)$$

has an F distribution with (DFM, DFE) degrees of freedom, and is independent of σ .

We can test this directly in Python with


```

N = result.nobs
k = result.df_model+1
dfm, dfe = k-1, N - k
F = result.mse_model / result.mse_resid
p = 1.0 - stats.f.cdf(F, dfm, dfe)
print('F-statistic: {:.3f}, p-value: {:.5f}'.format( F, p ))

```

which gives us

```
F-statistic: 12.785, p-value: 0.00723
```

Here, `stats.f.cdf(F, m, n)` returns the cumulative sum of the F-distribution with shape parameters $m = k-1 = 1$, and $n = N - k = 8$, up to the F-statistic F . Subtracting this quantity from one, we obtain the probability in the tail, which represents the probability of observing F-statistics more extreme than the one observed.

Log-Likelihood Function

A very common approach in statistics is the idea of *Maximum Likelihood* estimation. The basic idea is quite different from the *minimum square* approach: there, the model is constant, and the errors of the response are variable; in contrast, in the maximum likelihood approach, the data response values are regarded as constant, and the likelihood of the model is maximised.

For the Classical Linear Regression Model (with normal errors) we have

$$\epsilon = y_i - \sum_{k=1}^n \beta_k x_{ik} = y_i - \hat{y}_i \text{ in } N(0, \sigma^2) \quad (10.20)$$

so the probability density is given by

$$p(\epsilon_i) = \Phi\left(\frac{y_i - \hat{y}_i}{\sigma}\right) \quad (10.21)$$

where $\Phi(z)$ is the standard normal probability distribution function. The probability of independent samples is the product of the individual probabilities

$$\Pi_{total} = \prod_{i=1}^n p(\epsilon_i) \quad (10.22)$$

The *Log Likelihood function* is defined as

$$\begin{aligned}
 \ln(\mathfrak{L}) &= \ln(\Pi_{total}) \\
 &= \ln \left[\prod_{i=1}^n \frac{1}{\sigma\sqrt{2\pi}} \exp \left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma^2} \right) \right] \\
 &= \sum_{i=1}^n \left[\log \left(\frac{1}{\sigma\sqrt{2\pi}} \right) - \left(\frac{(y_i - \hat{y}_i)^2}{2\sigma^2} \right) \right]
 \end{aligned}$$

It can be shown that the maximum likelihood estimator of σ^2 is

$$E(\sigma^2) = \frac{SSE}{n} \quad (10.23)$$

We can calculate this in Python as follows:

```

N = result.nobs
SSR = result.ssr
s2 = SSR / N
L = ( 1.0/np.sqrt(2*np.pi*s2) ) ** N * np.exp( -SSR/(s2*2.0) )

```

```
print('ln(L) =', np.log( L ))

>>> ln(L) = -4.99975869739
```

Information Content of Statistical Models - AIC and BIC

To judge the quality of your model, you should first visually inspect the residuals. In addition, you can also use a number of numerical criteria to assess the quality of a statistical model. These criteria represent various approaches for balancing model accuracy with parsimony.

We have already encountered the *adjusted R^2* value (Eq 8.6), which - in contrast to the R^2 value - decreases if there are too many regressors in the model.

Other commonly encountered criteria are the *Akaike Information Criterion (AIC)* and the Schwartz or *Bayesian Information Criterion (BIC)*, which are based on the log-likelihood described in the previous section. Both measures introduce a penalty for model complexity, but the AIC penalizes complexity less severely than the BIC. The *Akaike Information Criterion AIC* is given by

$$AIC = 2 * k - 2 * \ln(\mathfrak{L}) \quad (10.24)$$

and the Schwartz or *Bayesian Information Criterion BIC* by

$$BIC = k * \ln(N) - 2 * \ln(\mathfrak{L}) \quad (10.25)$$

are other commonly encountered criteria. Here, N is the number of observations, k is the number of parameters, and \mathcal{L} is the likelihood. We have two parameters in this example, the slope and intercept. The AIC is a relative estimate of information loss between different models. The BIC was initially proposed using a Bayesian argument, and does not related to ideas of information. Both measures are only used when trying to decide between different models. So, if you have one regression for alcohol sales based on cigarette sales, and another model for alcohol consumption that incorporated cigarette sales and lighter sales, then you would be inclined to choose the model that had the lower AIC or BIC value.

10.2.3 Model Coefficients and Their Interpretation

Coefficients

The *coefficients* or weights of the linear regression are contained in `result.params`, and returned as a pandas Series object, since we used a pandas DataFrame as input. This is nice, because the coefficients are named for convenience.

```
result.params
>>> Intercept    2.041223
>>> Tobacco      1.005896
>>> dtype: float64
```

We can obtain this directly by computing

$$\beta = (X^T X)^{-1} X^T y. \quad (10.26)$$

Here, X is the matrix of predictor variables as columns, with an extra column of ones for the constant term, y is the column vector of the response variable, and β is the column vector of coefficients corresponding to the columns of X . In Python:

```
df['Eins'] = np.ones(( len(df), ))
Y = df.Alcohol[:-1]
X = df[['Tobacco', 'Eins']][:-1]
```

Standard Error

To obtain the *standard errors of the coefficients* we will calculate the covariance-variance matrix, also called the covariance matrix, for the estimated coefficients β of the predictor variables using

$$C = \text{cov}(\beta) = \sigma^2 (X X^T)^{-1}. \quad (10.27)$$

Here, σ^2 is the variance, or the MSE (mean squared error) of the residuals. The standard errors are the square roots of the elements on the main diagonal of this covariance matrix. We can perform the operation above, and calculate the element-wise square root using the following Python code,

```
X = df.Tobacco[:-1]

# add a column of ones for the constant intercept term
X = np.vstack(( np.ones(X.size), X ))

# convert the NumPy array to matrix
X = np.matrix( X )

# perform the matrix multiplication,
# and then take the inverse
C = np.linalg.inv( X * X.T )

# multiply by the MSE of the residual
C *= result.mse_resid

# take the square root
SE = np.sqrt(C)

print(SE)

>>> [[ 0.28132158          nan]
>>> [          nan  1.00136021]]
```

t-statistic

We use the t-test to test the null hypothesis that the coefficient of a given predictor variable is zero, implying that a given predictor has no appreciable effect on the response variable. The alternative hypothesis is that the predictor does contribute to the response. In testing we set some threshold, $\alpha = 0.05$, or 0.01 , and if $\Pr(T \geq |t|) < \alpha$, then we reject the null hypothesis at our threshold α , otherwise we fail to reject the null hypothesis. The t-test generally allows us to evaluate the importance of different predictors, *assuming that the residuals of the model are normally distributed about zero*. If the residuals do not behave in this manner, then that suggests that there is some non-linearity between the variables, and that their t-tests should not be used to assess the importance of individual predictors. Furthermore, it might be best to try to modify the model so that the residuals do tend the cluster normally about zero.

The t statistic is given by the ratio of the coefficient (or factor) of the predictor variable of interest, and its corresponding standard error. If β is the vector of coefficients or factors of our predictor variables, and SE is our standard error, then the t statistic is given by,

$$t_i = \beta_i / SE_{i,i} \quad (10.28)$$

So, for the first factor, corresponding to the slope in our example, we have the following code,

```
i = 1
beta = result.params[i]
se = SE[i,i]
```

```
t = beta / se
print('t =', t)

>>> t = 3.5756084542390316
```

Once we have a t statistic, we can (sort of) calculate the probability of observing a statistic at least as extreme as what we've already observed, given our assumptions about the normality of our errors by using the code,

```
N = result.nobs
k = result.df_model + 1
dof = N - k
p_onesided = 1.0 - stats.t(dof).cdf(t)
p = p_onesided * 2.0
print('p = {0:.3f}'.format(p))

>>> p = 0.007
```

Here, `dof` are the degrees of freedom, which should be eight, which is the number of observations, N , minus the number of parameters, which is two. The CDF is the cumulative sum of the PDF. We are interested in the area under the right hand tail, beyond our t statistic, t , so we subtract the cumulative sum up to that statistic from one in order to obtain the tail probability on the other side. We then multiply this tail probability by two to obtain a two-tailed probability.

Confidence Interval

The confidence interval is built using the standard error, the p -value from our T -test, and a critical value from a T -test having $N - k$ degrees of freedom, where k is the number of observations and P is the number of model parameters, i.e., the number of predictor variables. The confidence interval is the range of values we would expect to find the parameter of interest, based on what we have observed. You will note that we have a confidence interval for the predictor variable coefficient, and for the constant term. A smaller confidence interval suggests that we are confident about the value of the estimated coefficient, or constant term. A larger confidence interval suggests that there is more uncertainty or variance in the estimated term. Again, let me reiterate that hypothesis testing is only one perspective. Furthermore, it is a perspective that was developed in the late nineteenth and early twentieth centuries when data sets were generally smaller and more expensive to gather, and data scientists were using books of logarithm tables for arithmetic.

The confidence interval is given by,

$$CI = \beta_i \pm z \cdot SE_{i,i} \quad (10.29)$$

Here, β is one of the estimated coefficients, z is a *critical-value*, which is the t -statistic required to obtain a probability less than the alpha significance level, and $SE_{i,i}$ is the standard error. The critical value is calculated using the inverse of the cumulative distribution function. (The cumulative distribution function is the cumulative sum of the probability distribution.) In code, the confidence interval using a t -distribution looks like,

```
i = 0

# the estimated coefficient, and its variance
beta, c = result.params[i], SE[i,i]

# critical value of the t-statistic
N = result.nobs
P = result.df_model
dof = N - P - 1
z = stats.t(dof).ppf(0.975)
```

```
# the confidence interval
print(beta - z * c, beta + z * c)
```

10.2.4 Analysis of Residuals

The OLS command from `statsmodels.formula.api` provides some additional information about the residuals of the model: Omnibus, Skewness, Kurtosis, Durbin-Watson, Jarque-Bera, and the Condition number. In the following we will briefly describe these parameters.

Skewness and Kurtosis

Skew and kurtosis refer to the shape of a distribution. *Skewness* is a measure of the asymmetry of a distribution, and *kurtosis* is a measure of its curvature, specifically how pointed the curve is. (For normally distributed data approximately 3.) These values are calculated by hand as

$$S = \frac{\hat{\mu}_3}{\hat{\sigma}^3} = \frac{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^3}{\left(\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \right)^{3/2}}, \quad (10.30a)$$

$$K = \frac{\hat{\mu}_4}{\hat{\sigma}^4} = \frac{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^4}{\left(\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \right)^2} \quad (10.30b)$$

As you see, the $\hat{\mu}_3$ and $\hat{\mu}_4$ are the third and fourth central moments of a distribution. One possible Python implementation would be,

```
d = Y - result.fittedvalues

S = np.mean( d**3.0 ) / np.mean( d**2.0 )**(3.0/2.0)
# equivalent to:
# S = stats.skew(result.resid, bias=True)

K = np.mean( d**4.0 ) / np.mean( d**2.0 )**(4.0/2.0)
# equivalent to:
# K = stats.kurtosis(result.resid, fisher=False, bias=True)
print('Skewness: {:.3f}, Kurtosis: {:.3f}'.format( S, K ))

>>> Skewness: -0.014, Kurtosis: 1.527
```

Omnibus Test

The Omnibus test uses skewness and kurtosis to test the null hypothesis that a distribution is normal. In this case, we're looking at the distribution of the residual. If we obtain a very small value for $\Pr(\text{Omnibus})$, then the residuals are not normally distributed about zero, and we should maybe look at our model more closely. The `statsmodels` OLS function uses the `stats.normaltest()` function:

```
(K2, p) = stats.normaltest(result.resid)
print('Omnibus: {0}, p = {1}'.format(K2, p))

>>> Omnibus: 2.5418981690649174, p = 0.28056521527106976
```

Thus, if either the skewness or kurtosis suggests non-normality, this test should pick it up.

Durbin-Watson

The Durbin-Watson test is used to detect the presence of autocorrelation (a relationship between values separated from each other by a given time lag) in the residuals. Here the lag is one.

$$DW = \frac{\sum_{i=2}^N ((y_i - \hat{y}_i) - (y_{i-1} - \hat{y}_{i-1}))^2}{\sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (10.31)$$

```
DW = np.sum( np.diff( result.resid.values )**2.0 ) / result.ssr
print('Durbin-Watson: {:.5f}'.format( DW ))

>>> Durbin-Watson: 1.97535
```

Jarque-Bera Test

The Jarque-Bera test is another test that considers skewness (S), and kurtosis (K). The null hypothesis is that the distribution is normal, that both the skewness and excess kurtosis equal zero, or alternatively, that the skewness is zero and the regular run-of-the-mill kurtosis is three. Unfortunately, with small samples the Jarque-Bera test is prone rejecting the null hypothesis - that the distribution is normal when it is in fact true.

$$JB = \frac{N}{6} \left(S^2 + \frac{1}{4}(K - 3)^2 \right) \quad (10.32)$$

Calculating the JB-statistic using the χ^2 distribution with two degrees of freedom we have,

```
JB = (N/6.0) * ( S**2.0 + (1.0/4.0)*( K - 3.0 )**2.0 )
p = 1.0 - stats.chi2(2).cdf(JB)
print('JB-statistic: {:.5f}, p-value: {:.5f}'.format( JB, p ))

>>> JB-statistic: 0.90421, p-value: 0.63629
```

Condition Number

The *condition number* measures the sensitivity of a function's output to its input. When two predictor variables are highly correlated, which is called multicollinearity, the coefficients or factors of those predictor variables can fluctuate erratically for small changes in the data, or the model. Ideally, similar models should be similar, i.e., have approximately equal coefficients. Multicollinearity can cause numerical matrix inversion to crap out, or produce inaccurate results. One approach to this problem in regression is the technique of *ridge regression*, which is available in the `sklearn` Python module.

We calculate the condition number by taking the eigenvalues of the product of the predictor variables (including the constant vector of ones) and then taking the square root of the ratio of the largest eigenvalue to the least eigenvalue. If the condition number is greater than thirty, then the regression may have multicollinearity.

```
X = np.matrix( X )
EV = np.linalg.eig( X * X.T )
print(EV)

>>> (array([ 0.18412885, 136.51527115]), matrix([[ -0.96332746,
-0.26832855],
>>> [ 0.26832855, -0.96332746]]))
```

Note that $X.T * X$ should be $(P+1) \times (P+1)$, where P is the number of degrees of freedom of the model (the number of predictors) and the $+1$ represents the addition of the constant vector of ones for the intercept term. In our case, the product should be a 2×2 matrix, so we'll have two eigenvalues. Then our condition number is given by,

```
CN = np.sqrt( EV[0].max() / EV[0].min() )
print('Condition No.: {:.5f}'.format( CN ))

>>> Condition No.: 27.22887
```

Our condition number is juuust below 30, so we can sort of sleep okay.

10.2.5 Comparison

Now that we have seen an example of linear regression with a reasonable degree of linearity, compare that with an example of one with a significant outlier. In practice, outliers should be understood before they are discarded, because they might turn out to be very important. They might signify a new trend, or some possibly catastrophic event.

```
X = df[['Tobacco', 'Eins']]
Y = df.Alcohol
result = sm.OLS( Y, X ).fit()
result.summary()
```

```

OLS Regression Results
=====
Dep. Variable:          Alcohol    R-squared:                0.050
Model:                  OLS        Adj. R-squared:           -0.056
Method:                 Least Squares    F-statistic:             0.4735
Date:                  Sun, 27 Apr 2014    Prob (F-statistic):       0.509
Time:                  12:58:27    Log-Likelihood:          -12.317
No. Observations:       11    AIC:                     28.63
Df Residuals:           9    BIC:                     29.43
Df Model:                1
=====
               coef      std err          t      P>|t|      [95.0% Conf. Int.]
-----
Intercept      4.3512      1.607       2.708     0.024      0.717      7.986
Tobacco        0.3019      0.439       0.688     0.509     -0.691      1.295
=====
Omnibus:                 3.123    Durbin-Watson:           1.655
Prob(Omnibus):           0.210    Jarque-Bera (JB):         1.397
Skew:                   -0.873    Prob(JB):                 0.497
Kurtosis:                3.022    Cond. No.:                25.5
=====
```

10.2.6 Regression Using Sklearn

Scikit-learn (*sklearn*) is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.

In order to use *sklearn*, we need to input our data in the form of vertical vectors. Whenever one slices off a column from a NumPy array, NumPy stops worrying whether it is a vertical or horizontal vector. MATLAB works differently, as it is primarily concerned with matrix operations. NumPy, however has a matrix class for whenever the verticalness or horizontalness of an array is important. Therefore, in our case, we'll cast the DataFrame as Numpy matrix so that vertical arrays stay vertical once they are sliced off the data set.

```
data = np.matrix( df )
```

Next, we create the regression objects, and fit the data to them. In this case, we'll consider a clean set, which will fit a linear regression better, which consists of the data for all of the regions except Northern Ireland, and an original set consisting of the original data

```
cln = LinearRegression()
org = LinearRegression()

X, Y = data[:,2], data[:,1]
cln.fit( X[:-1], Y[:-1] )
org.fit( X, Y )

clean_score = '{0:.3f}'.format( cln.score( X[:-1], Y[:-1] ) )
original_score = '{0:.3f}'.format( org.score( X, Y ) )
```

The next piece of code produces a scatter plot of the regions, with all of the regions plotted as empty blue circles, except for Northern Ireland, which is depicted as a red star.

```
mpl.rcParams['font.size']=16

plt.plot( df.Tobacco[:-1], df.Alcohol[:-1], 'bo', markersize=10,
          label='All other regions, $R^2$ = '+clean_score )

plt.hold(True)
plt.plot( df.Tobacco[-1:], df.Alcohol[-1:], 'r*', ms=20, lw=10,
          label='N. Ireland, outlier, $R^2$ = '+original_score)
```

The next part generates a set of points from 2.5 to 4.85, and then predicts the response of those points using the linear regression object trained on the clean and original sets, respectively.

```
test = np.arange( 2.5, 4.85, 0.1 )
test = np.array( np.matrix( test ).T )

plot( test, cln.predict( test ), 'k' )
plot( test, org.predict( test ), 'k--' )
```

Finally, we limit and label the axes, add a title, overlay a grid, place the legend at the bottom, and then save the figure.

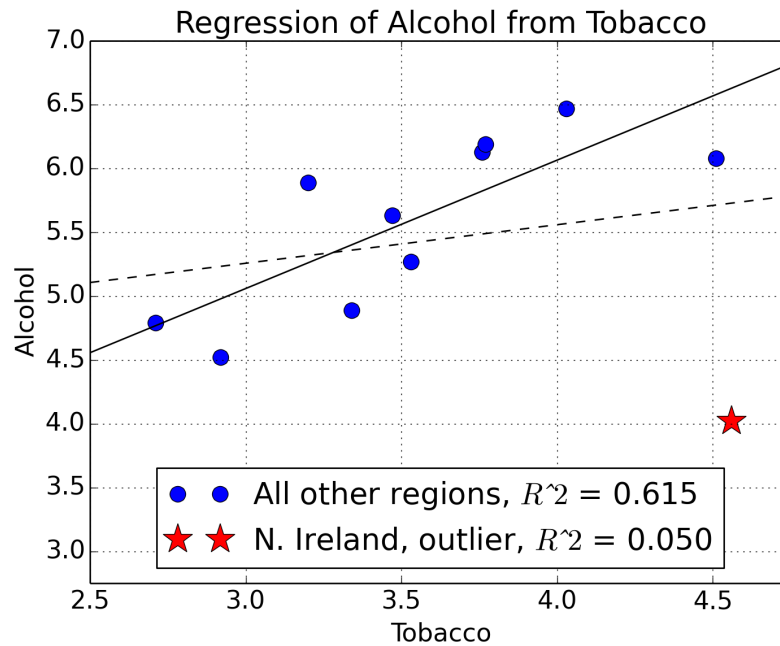
```
xlabel('Tobacco') ; xlim(2.5,4.75)
ylabel('Alcohol') ; ylim(2.75,7.0)
title('Regression of Alcohol from Tobacco')
grid()
legend(loc='lower center')
```

10.2.7 Conclusion

Before you do anything, visualize your data. If your data is highly dimensional, then at least examine a few slices using boxplots. At the end of the day, use your own judgement about a model based on your knowledge of your domain. Statistical tests should guide your reasoning, but they shouldn't dominate it. In most cases, your data will not align itself with the assumptions made by most of the available tests. Here is a very interesting article from Nature on classical hypothesis testing. A more intuitive approach to hypothesis testing is Bayesian analysis.

10.3 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 .

10.3.1 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 fixed—that 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.

 **python**™ **Code:** “modeling.py” (p 145) shows an example.

10.4 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.

 **python**™ **Code:** "bootstrapDemo.py" (p 147): Example of bootstrapping the confidence interval for the mean of a sample distribution.

Chapter 11

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.

11.1 Survival Probabilities

11.1.1 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

$$p_k = p_{k-1} * \frac{r_k - f_k}{r_k} \quad (11.1)$$

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 11.1).

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

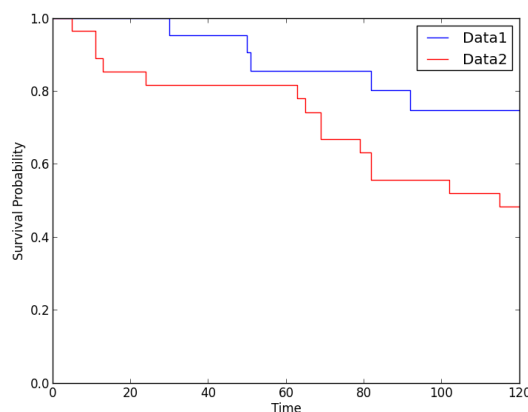


Figure 11.1: Survival curve corresponding to a motion sickness experiment, described in more detail in [Altman(1999)], chapter 13

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

11.2 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 A.26).

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.



Code: "survival.py" (p 148): Survival analysis (Kaplan-Meier curves).

Appendix A

Appendix

A.1 Python Programs

Listing A.1: gettingStarted_ipy.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 execution
  (Note: this does NOT work in ipython, if you run it with inline figures!)
* 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, date: May-2013

# In contrast to MATLAB, you explicitly have to load the modules that you need.
# for interactive work, there is a shortcut: "pylab", which loads numpy and
# matplotlib.pyplot into the current workspace
from pylab import *

# For Python programs, it is common to load the modules explicitly.
# 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.
# Check out "gettingStarted.py", how this script would look with that explicit
# use of the modules.

def main():
    '''Define the main function. '''
    # Create a sine-wave
    t = arange(0,10,0.1)
    x = 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'
savetxt(outFile, vstack([t,x]).T)

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

# Plot the data, and wait for the user to click
show()
plot(t2,x2)
title('Hit any key to continue')
waitforbuttonpress()

# Generate a noisy line
t = 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']*randn(len(t))

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

# Plot the "late" data
close()
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 = vstack((tHigh, ones(len(tHigh))))).T
slope, intercept = 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
hold(True)
plot(tHigh, intercept + slope*tHigh, 'r')
title('Hit any key to continue')
savefig('linefit.png', dpi=200)
waitforbuttonpress()
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 as "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
```

Listing A.2: readZip.py

```

'''Get data from MS-Excel files, which are stored zipped on the Web.
'''

# author: Thomas Haslwanter, date: Jan-2014

import urllib
import io
import zipfile
import pandas as pd

def getDataDobson(url, inFile):
    '''Extract data from a zipped-archive'''

    # get the zip-archive
    GLM_archive = urllib.request.urlopen(url).read()

    # make the archive available as a byte-stream
    zipdata = io.BytesIO()
    zipdata.write(GLM_archive)

    # extract the requested file from the archive, as a pandas XLS-file
    myzipfile = zipfile.ZipFile(zipdata)
    xlsfile = myzipfile.open(inFile)

    # read the xls-file into Python, using Pandas, and return the extracted data
    xls = pd.ExcelFile(xlsfile)
    df = xls.parse('Sheet1', skiprows=2)

    return df

if __name__ == '__main__':
    # Select archive (on the web) and the file in the archive
    url = 'http://cdn.crcpress.com/downloads/C9500/GLM_data.zip'
    inFile = r'GLM_data/Table 2.8 Waist loss.xls'

    df = getDataDobson(url, inFile)
    print(df)

    input('All done!')

```

Listing A.3: statsmodelsIntro.py

```

'''Introductions into using "statsmodels"

'''

# author: Thomas Haslwanter, date: April-2013

import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.formula.api as sm
import sys
import matplotlib.pyplot as plt
if sys.version_info[0] == 3:
    from urllib.request import urlopen
else:
    from urllib import urlopen

def simple_fit():
    ''' Example: Linear regression fit '''

    # To get reproducible values, I provide a seed value
    np.random.seed(987654321)

    # Generate a noisy line
    x = np.arange(100)
    y = 0.5*x - 20 + np.random.randn(len(x))
    df = pd.DataFrame({'x':x, 'y':y})

    # Fit a linear model ...
    model = sm.ols('y~x', data=df).fit()

    # ... and print the summary
    print((model.summary()))

    return model.params

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

    # Get the data
    inFile = 'altman_94.txt'
    url_base = 'https://raw.githubusercontent.com/thomas-haslwanter/statsintro/master/Data/
data_altman/'
    url = url_base + inFile
    data = np.genfromtxt(urlopen(url), delimiter=',')

    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())

    df.boxplot()
    plt.show()

    stats.ttest_ind(lean, obese)

    print(df.mean())
if __name__ == '__main__':
    simple_fit()
    pandas_boxplot()

```

Listing A.4: getdata.py

```

'''Get data for the Python programs for statistics, FH 00e.
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.

If the data are not found locally, they are retrieved from the WWW.
'''

#Author: Thomas Haslwanter, May-2014

from os.path import join
from numpy import genfromtxt
import os
if sys.version_info[0] == 3:
    from urllib.request import urlopen
    from urllib.parse import urlparse
else:
    from urlparse import urlparse
    from urllib import urlopen

def getData(inFile, subDir=r'.\Data'):
    '''Data are taken from examples in D. Altman, "Practical Statistics for
    Medical Research" '''
    dataDir = os.path.join(os.path.dirname(__file__), subDir)
    fullInFile = join(dataDir, inFile)
    try:
        data = genfromtxt(fullInFile, delimiter=',')
    except IOError:
        print((fullInFile + ' does not exist: Trying to read from WWW'))
        try:
            url_base = 'https://raw.githubusercontent.com/thomas-haslwanter/statsintro/
            master/Data/'
            url = os.path.join(url_base, inFile)
            print(url)
            fileHandle = urlopen(url)
            data = genfromtxt(fileHandle, delimiter=',')
        except:
            print((url + ' also does not exist!'))
            data = ()
    return data

if __name__ == '__main__':
    data = getData(r'data_altman\altman_93.txt')
    print(data)

```

Listing A.5: figs_BasicPrinciples.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
- violinplots
- probplots
- cumulative density functions
- regression fits

'''

# author: Thomas Haslwanter, date: May-2013

# 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
import seaborn as sns
import pandas as pd

def main():
    # 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 indicate 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()

# Violinplot
nd = stats.norm
data = nd.rvs(size=(100))
nd2 = stats.norm(loc = 0.5, scale = 1.2)
data2 = nd2.rvs(size=(100))

# Use the seaborn package for the violin plot, and set the context for "
# poster"
sns.set(context='poster')
df = pd.DataFrame({'Girls':data, 'Boys':data2})
sns.violinplot(df)
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()

if __name__ == '__main__':
    main()

```

Listing A.6: fig_centralLimitTheorem.py

```

'''
Practical demonstration of the central limit theorem
'''

# author: Thomas Haslwanter, date: March-2014

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

def main():
    '''Demonstrate central limit theorem.'''
    # Generate data
    ndata = 1e5
    nbins = 50
    data = np.random.random(ndata)

    # Show them
    fig, axs = plt.subplots(1,3)
    sns.set(context='talk')

    axs[0].hist(data,bins=nbins)
    axs[0].set_title('Random data')
    axs[0].set_ylabel('Counts')

    axs[1].hist( np.mean(data.reshape((ndata/2,2))), axis=1), bins=nbins)
    axs[1].set_title(' Average over 2')

    axs[2].hist( np.mean(data.reshape((ndata/10,10))),axis=1), bins=nbins)
    axs[2].set_title(' Average over 10')

    curDir = os.path.abspath(os.path.curdir)
    outFile = 'CentralLimitTheorem.png'
    plt.savefig(outFile, dpi=200)
    print('Data written to {0}'.format(os.path.join(curDir, outFile)))

    plt.show()

if __name__ == '__main__':
    main()

```

Listing A.7: distributionNormal.py

```

''' Simple manipulations of normal distribution functions.
- Different displays of normally distributed data
- Compare different samples from a normal distribution
- Work with the cumulative distribution function (CDF)

'''

# author: Thomas Haslwanter, date: Sept-2013

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

myMean = 0
mySD = 3
x = np.arange(-5,15,0.1)

def simple_normal():
    ''' Different aspects of a normal distribution'''
    # Generate the data
    x = np.arange(-4,4,0.1) # generate the desired x-values
    x2 = np.arange(0,1,0.001)

    nd = stats.norm() # First simply define the normal distribution;
                     # don't calculate any values yet

    # This is a more complex plot-layout: the first row
    # is taken up completely by the PDF
    ax = plt.subplot2grid((3,2),(0,0), colspan=2)

    plt.plot(x,nd.pdf(x))
    plt.xlim([-4,4])
    plt.title('Normal Distribution - PDF')

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

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

    plt.subplot(325)
    plt.plot(x2,nd.ppf(x2))
    plt.title('PPF')

    plt.subplot(326)
    plt.plot(x2,nd.isf(x2))
    plt.title('ISF')
    plt.tight_layout()
    plt.show()

def shifted_normal():
    '''PDF, scatter plot, and histogram.'''
    # Generate the data
    # Plot a normal distribution: "Probability density functions"
    myMean = 5
    mySD = 2
    y = stats.norm.pdf(x, myMean, mySD)

```



```

plt.plot(x,y)
plt.title('Shifted Normal Distribution')
plt.show()

# Generate random numbers with a normal distribution
numData = 500
data = stats.norm.rvs(myMean, mySD, size = numData)
plt.plot(data, '.')
plt.title('Normally distributed data')
plt.show()

plt.hist(data)
plt.title('Histogram of normally distributed data')
plt.show()
plt.close()

def many_normals():
    '''Show multiple samples from the same distribution, and compare means.'''
    # Do this 25 times, and show the histograms
    numRows = 5
    numData = 50
    numData = 100
    plt.figure()
    for ii in range(numRows):
        for jj in range(numRows):
            data = stats.norm.rvs(myMean, mySD, size=numData)
            plt.subplot(numRows,numRows,numRows*ii+jj+1)
            plt.hist(data)
            plt.gca().set_xticklabels(())
            plt.gca().set_yticklabels(())

    plt.tight_layout()
    plt.show()
    plt.close()

    # Check out the mean of 1000 normally distributed samples
    numTrials = 1000;
    numData = 100
    myMeans = np.ones(numTrials)*np.nan
    for ii in range(numTrials):
        data = stats.norm.rvs(myMean, mySD, size=numData)
        myMeans[ii] = np.mean(data)
    print(('The standard error of the mean, with {0} samples, is {1}'.format(
        numData, np.std(myMeans))))

def values_fromCDF():
    '''Calculate an empirical cumulative distribution function, compare it with
       the exact one, and
       find the exact point for a specific data value.'''

    # Generate normally distributed random data
    myMean = 5
    mySD = 2
    numData = 100
    data = stats.norm.rvs(myMean, mySD, size=numData)

    # Calculate the cumulative distribution function, CDF
    numbins = 20
    counts, bin_edges = np.histogram(data, bins=numbins, normed=True)
    cdf = np.cumsum(counts)
    cdf /= np.max(cdf)

    # compare with the exact CDF

```

```

plt.step(bin_edges[1:],cdf)
plt.hold(True)
plt.plot(x, stats.norm.cdf(x, myMean, mySD), 'r')

# Find out the value corresponding to the x-th percentile: the
# "cumulative distribution function"
value = 2
myMean = 5
mySD = 2
cdf = stats.norm.cdf(value, myMean, mySD)
print(('With a threshold of {0:4.2f}, you get {1}% of the data'.format(value
    , round(cdf*100))))

# For the percentile corresponding to a certain value:
# the "inverse cumulative distribution function"
value = 0.025
icdf = stats.norm.isf(value, myMean, mySD)
print(('To get {0}% of the data, you need a threshold of {1:4.2f}'.format
    ((1-value)*100, icdf)))
plt.show()

if __name__ == '__main__':
    simple_normal()
    shifted_normal()
    many_normals()
    values_fromCDF()

```

Listing A.8: Continuous

```

''' Different continuous distribution functions.
- Normal distribution
- Exponential distribution
- T-distribution
- F-distribution
- Logistic distribution
- Weibull distribution
- Lognormal distribution
- Uniform distribution

'''

# author: Thomas Haslwanter, date: Jun-2013

# 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})

#-----
def showDistribution(x, 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()
    close()

#-----
def show_continuous():
    """Show a variety of continuous distributions"""

    x = linspace(-10,10,201)

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

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

```

```

# Weibull distribution
# ... with the shape parameter set to 1 and 2
# Don't worry that in Python it is called "weibull_min": the "weibull_max"
  is
# simply mirrored about the origin.
showDistribution(arange(0,5,0.02), stats.weibull_min(1), stats.weibull_min
  (2),
                'Weibull Distribution', 'X', 'P(X)', ['k=1', 'k=2'], xmin=0,
                xmax=4)

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

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

# Lognormal distribution
x = logspace(-9,1,1001)+1e-9
showDistribution(x, 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()

#-----
if __name__ == '__main__':
    show_continuous()

```

Listing A.9: Discrete

```

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

'''

# author: Thomas Haslwanter, date: April-2013

# 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

#-----
def show_binomial():
    """Show an example of binomial distributions"""

    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 distribution')
    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()

#-----
def show_poisson():
    """Show different views of a Poisson distribution"""

    fig, ax = plt.subplots(3,1)

    k = np.arange(25)
    pd = stats.poisson(10)

    ax[0].plot(k, pd.pmf(k), 'x-')
    ax[0].set_title('Poisson distribution')
    ax[0].set_xticklabels([])
    ax[0].set_ylabel('PMF (X)')

    ax[1].plot(k, pd.cdf(k))
    ax[1].set_xlabel('X')
    ax[1].set_ylabel('CDF (X)')

    y = np.linspace(0,1,100)
    ax[2].plot(y, pd.ppf(y))
    ax[2].set_xlabel('X')
    ax[2].set_ylabel('PPF (X)')

    plt.tight_layout()
    plt.show()

#
#-----

if __name__ == '__main__':

```

```
show_binomial()  
show_poisson()
```

Listing A.10: checkNormality.py

```

'''
Graphical and quantitative check, if a given distribution is normal.
The Kolmogorov-Smirnov(KS) test has been replaced with the Lilliefors-test,
since the original KS-test is unreliable when mean and std of the distribution
are not known.
'''

# author: Thomas Haslwanter, date: April-2014

import numpy as np
import scipy.stats as stats
from statsmodels.stats.diagnostic import kstest_normal
import matplotlib.pyplot as plt

myMean = 0
mySD = 3
x = np.arange(-5,15,0.1)

def check_normality():
    '''Check if the distribution is normal.'''
    # Generate and show a distribution
    numData = 100

    # To get reproducible values, I provide a seed value
    np.random.seed(987654321)

    data = stats.norm.rvs(myMean, mySD, size=numData)
    plt.hist(data)
    plt.show()

    # --- >>> START stats <<< ---
    # Graphical test: if the data lie on a line, they are pretty much
    # normally distributed
    _ = stats.probplot(data, plot=plt)
    plt.show()

    # The scipy normaltest is based on D-Agostino and Pearsons test that
    # combines skew and kurtosis to produce an omnibus test of normality.
    stats.normaltest(data)

    # Or you can check for normality with Lilliefors-test
    ksStats, pVal = kstest_normal(data)

    # Alternatively with original Kolmogorov-Smirnov test
    #_,pVal = stats.kstest((data-np.mean(data))/np.std(data,ddof=1), 'norm')

    if pVal > 0.05:
        print('Data are normally distributed')
    # --- >>> STOP stats <<< ---

    return pVal

if __name__ == '__main__':
    p = check_normality()
    # raw_input('Done')

```

Listing A.11: sampleSize.py

```

'''Calculate the sample size for experiments, for normally distributed groups,
for:
- Experiments with one single group
- Comparing two groups

'''

# author: Thomas Haslwanter, May 2013

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))))

    return 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))))

    return n

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

```


Listing A.12: oneSample.py

```

'''Analysis of one sample of data

This script shows how to
- Use a t-test for a single mean
- Use a non-parametric test (Wilcoxon signed rank) to check a single mean
- Compare the values from the t-distribution with those of a normal distribution

'''

# author: Thomas Haslwanter, date: Jan-2014

import numpy as np
import scipy.stats as stats
from getdata import getData

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', subDir='../Data\data_altman')

    # 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
    # --- >>> START stats <<< ---
    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'
    # --- >>> STOP stats <<< ---

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

    return prob # should be 0.018137235176105802

def compareWithNormal():
    # 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)))

return normProb # should be 0.054201154690070759

if __name__ == '__main__':
    check_mean()
    compareWithNormal()
```

Listing A.13: twoSample.py

```

''' Comparison of two groups
- Analysis of paired data
- Analysis of unpaired data

'''

# author: Thomas Haslwanter, date: July-2013

from numpy import genfromtxt, mean, std
import scipy.stats as stats
import matplotlib.pyplot as plt
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', subDir=r'..\Data\data_altman')

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

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

    # --- >>> START stats <<< ---
    # 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
    rankSum, p_value = stats.wilcoxon(post - pre)
    # --- >>> STOP stats <<< ---
    print(("paired wilcoxon-test", p_value))

    return p_value # should be 0.0033300139117459797

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', subDir=r'..\Data\data_altman')

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

    mean(group1)
    mean(group2)

```

```

# --- >>> START stats <<< ---
# 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)
# --- >>> STOP stats <<< ---

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

return p_value # should be 0.0010608066929400244

if __name__ == '__main__':
    paired_data()
    unpaired_data()

```

Listing A.14: anovaOneway.py

```

''' Analysis of Variance (ANOVA)
- Levene test
- ANOVA - oneway
- Do a simple one-way ANOVA, using statsmodels
- Show how the ANOVA can be done by hand.
- For the comparison of two groups, a one-way ANOVA is equivalent to
  a T-test:  $t^2 = F$ 

'''

# author: Thomas Haslwanter, date: May-2013

import numpy as np
import scipy.stats as stats
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.

    Twenty-two patients undergoing cardiac bypass surgery were randomized to one
    of three ventilation groups:

    Group I: Patients received 50% nitrous oxide and 50% oxygen mixture
    continuously for 24 h.
    Group II: Patients received a 50% nitrous oxide and 50% oxygen mixture only
    during the operation.
    Group III: Patients received no nitrous oxide but received 35-50% oxygen for
    24 h.

    The data show red cell folate levels for the three groups after 24h'
    ventilation.

    '''

    # Get the data
    print('One-way ANOVA: -----')
    data = getData('altman_910.txt', subDir='../Data\data_altman')

    # 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]

    # --- >>> START stats <<< ---
    # 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)
    # --- >>> STOP stats <<< ---

    # Print the results
    print('Data from 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()
anovaResults = anova_lm(model)
print(anovaResults)

# Check if the two results are equal. If they are, there is no output
np.testing.assert_almost_equal(F_statistic, anovaResults['F'][0])

return (F_statistic, pVal) # should be (3.711335988266943,
    0.043589334959179327)

#-----
def show_teqf():
    """Shows the equivalence of t-test and f-test, for comparing two groups"""

    # Get the data
    data = pd.read_csv(r'..\Data\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('\nT^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)))

    # numeric test
    np.testing.assert_almost_equal(t_val**2, F_statistic)

    return F_statistic

# -----
def anova_statsmodels():
    ''' do the ANOVA with a function '''

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

    anova_results = anova_lm(ols('height ~ 1 + sex', data).fit())
    print('\nANOVA with "statsmodels" -----')
    print(anova_results)

    return anova_results['F'][0]

#-----
def anova_byHand():
    """ Calculate the ANOVA by hand. While you would normally not do that, this
        function shows
        how the underlying values can be calculated.
    """

    # Get the data
    data = getData('altman_910.txt', subDir='..\Data\data_altman')

    # Convert them to pandas-form and group them by their group value
    df = pd.DataFrame(data, columns=['values', 'group'])

```

```

groups = df.groupby('group')

# The "total sum-square" is the squared deviation from the mean
ss_total = np.sum((df['values']-df['values'].mean())**2)

# Calculate ss_treatment and ss_error
(ss_treatments, ss_error) = (0, 0)
for val, group in groups:
    ss_error += sum((group['values'] - group['values'].mean())**2)
    ss_treatments += len(group) * (group['values'].mean() - df['values'].
                                   mean())**2

df_groups = len(groups)-1
df_residuals = len(data)-len(groups)
F = (ss_treatments/df_groups) / (ss_error/df_residuals)
df = stats.f(df_groups,df_residuals)
p = df.sf(F)

print(('ANOVA-Results: F = {0}, and p<{1}'.format(F, p)))

return (F, p)

if __name__ == '__main__':
    anova_oneway()
    anova_byHand()
    show_teqf()
    anova_statsmodels()
    #raw_input('Done!')
```

Listing A.15: multipleTesting.py

```

'''Multiple testing
This script provides an example, where three treatments are compared. It first
performs a one-way ANOVA, to see if there is a difference between the groups.
Then it performs multiple comparisons, to check which of the groups are
different.

Taken from an example by Josef Perktold (http://jpktd.blogspot.co.at/)

'''

# author: Thomas Haslwanter, date: April-2013

import numpy as np
from scipy import stats
import pandas as pd

def main():
    # Note: the statsmodels module is required here.
    from statsmodels.stats.multicomp import (pairwise_tukeyhsd,
                                             MultiComparison)

    from statsmodels.formula.api import ols
    from statsmodels.stats.anova import anova_lm

    # Set up the data, as a structured array.
    # The first and last field are 32-bit intergers; the second field is an
    # 8-byte string. Note that here we can also give names to the individual
    # fields!
    dta2 = np.rec.array([
        ( 1, 'mental', 2 ),
        ( 2, 'mental', 2 ),
        ( 3, 'mental', 3 ),
        ( 4, 'mental', 4 ),
        ( 5, 'mental', 4 ),
        ( 6, 'mental', 5 ),
        ( 7, 'mental', 3 ),
        ( 8, 'mental', 4 ),
        ( 9, 'mental', 4 ),
        (10, 'mental', 4 ),
        (11, 'physical', 4 ),
        (12, 'physical', 4 ),
        (13, 'physical', 3 ),
        (14, 'physical', 5 ),
        (15, 'physical', 4 ),
        (16, 'physical', 1 ),
        (17, 'physical', 1 ),
        (18, 'physical', 2 ),
        (19, 'physical', 3 ),
        (20, 'physical', 3 ),
        (21, 'medical', 1 ),
        (22, 'medical', 2 ),
        (23, 'medical', 2 ),
        (24, 'medical', 2 ),
        (25, 'medical', 3 ),
        (26, 'medical', 2 ),
        (27, 'medical', 3 ),
        (28, 'medical', 1 ),
        (29, 'medical', 3 ),
        (30, 'medical', 1 )], dtype=[('idx', '<i4'),
                                     ('Treatment', '|S8'),
                                     ('StressReduction', '<i4')])

    # First, do an one-way ANOVA

```



```

df = pd.DataFrame(dta2)
model = ols('StressReduction ~ C(Treatment)', df).fit()

anovaResults = anova_lm(model)
print(anovaResults)
if anovaResults['PR(>F)'][0] < 0.05:
    print('One of the groups is different.')

#Then, do the multiple testing
mod = MultiComparison(dta2['StressReduction'], dta2['Treatment'])
print((mod.tukeyhsd().summary()))

# The following code produces the same printout
res2 = pairwise_tukeyhsd(dta2['StressReduction'], dta2['Treatment'])
#print res2[0]

# Show the group names
print((mod.groupsunique))

# Generate a print
import matplotlib.pyplot as plt
xvals = np.arange(3)
plt.plot(xvals, res2.meandiffs, 'o')
#plt.errorbar(xvals, res2.meandiffs, yerr=np.abs(res2[1][4].T-res2[1][2]),
#             ls='o')
errors = np.ravel(np.diff(res2.confint)/2)
plt.errorbar(xvals, res2.meandiffs, yerr=errors, ls='o')
xlim = -0.5, 2.5
plt.hlines(0, *xlim)
plt.xlim(*xlim)
pair_labels = mod.groupsunique[np.column_stack(res2._multicomp.pairindices)]
plt.xticks(xvals, pair_labels)
plt.title('Multiple Comparison of Means - Tukey HSD, FWER=0.05' +
          '\n Pairwise Mean Differences')

# Save to outfile
outFile = 'MultComp.png'
plt.savefig('MultComp.png', dpi=200)
print(('Figure written to {0}'.format(outFile)))

plt.show()

# Instead of the Tukey's test, we can do pairwise t-test
# First, with the "Holm" correction
rtp = mod.allpairtest(stats.ttest_rel, method='Holm')
print((rtp[0]))

# and then with the Bonferroni correction
print((mod.allpairtest(stats.ttest_rel, method='b')[0]))

# Done this way, the variance is calculated at each comparison.
# If you want the joint variance across all samples, you have to
# use a few tricks: (http://jpktd.blogspot.co.at/2013/03/multiple-comparison-and-tukey-hsd-or\_25.html)
res2 = pairwise_tukeyhsd(dta2['StressReduction'], dta2['Treatment'])
studentized_mean = res2.meandiffs
studentized_variance = res2.variance

t_stat = (studentized_mean / studentized_variance) / np.sqrt(2)
dof = len(dta2) - len(mod.groupsunique)
my_pvalues = stats.t.sf(np.abs(t_stat), dof) * 2 # two-sided

# Now with the Bonferroni correction

```

```
from statsmodels.stats.multitest import multipletests
res_b = multipletests(my_pvalues, method='b')

return res2.variance

if __name__ == '__main__':
    main()
```

Listing A.16: KruskalWallis.py

```
'''Example of a Kruskal-Wallis test (for not normally distributed data)
Taken from http://www.brightstat.com/index.php?option=com\_content&task=view&id=41&Itemid=1&limit=1&limitstart=2
'''

# author: Thomas Haslwanter, date: May-2013

from scipy.stats.mstats import kruskalwallis
from numpy import array

def main():
    '''These data could be a comparison of the smog levels in four different
    cities.'''

    # 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])

    # --- >>> START stats <<< ---
    # Perform the Kruskal-Wallis test
    h, p = kruskalwallis(city1, city2, city3, city4)
    # --- >>> STOP stats <<< ---

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

    return h

if __name__ == '__main__':
    main()
```

Listing A.17: binomialTest.py

```

'''Binomial Test
Example of a one-and two-sided binomial test. Taken from Wikipedia
http://en.wikipedia.org/wiki/Binomial_test
'''

# author: Thomas Haslwanter, date: July-2013

from scipy import stats

def binomial_test(checkVal):
    '''Binomial Test'''

    # Set the parameters
    n = 235
    p = 1/6.

    # --- >>> START stats <<< ---
    # Calculate the on-sided test, i.e. the likelihood that you get the same or
    # more times of "6"
    bd = stats.binom(n,p)
    p_oneTail = bd.sf(checkVal-1)    # how many values are "higher than" checkVal
    -1

    # Calculate the two-sided test, i.e. how many cases "as extreme or more"
    # than the given case are likely to occur by chance:
    p_twoTail = stats.binom_test(checkVal, n, p)
    # --- >>> STOP stats <<< ---

    return (p_oneTail, p_twoTail)

#-----
if __name__ == '__main__':

    checkVal = 51
    p1, p2 = binomial_test(checkVal)
    print(('The chance that you roll {0} or more "6" is {1}, and the chance of
    an event as extreme as {0} or more rolls is {2}'.format(checkVal, p1, p2)
    ))

```

Listing A.18: compGroups.py

```

''' Analysis of categorical data
- Analysis of one proportion
- Chi-square test
- Fisher exact test
- McNemar's test
- Cochran's Q test

'''

# author: Thomas Haslwanter, date: Mar-2014

import numpy as np
import scipy.stats as stats
import pandas as pd
from statsmodels.sandbox.stats.runs import cochrans_q, mcnemar

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.
    Suppose a general practitioner chooses a random sample of 215 women from
    the patient register for her general practice, and finds that 39 of them
    have a history of suffering from asthma. '''

    # Get the data
    numTotal = 215
    numPositive = 39

    # --- >>> START stats <<< ---
    # 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
    # --- >>> STOP stats <<< ---

    # 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 ci

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]])

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

```

```

# --- >>> STOP stats <<< ---

# Print the result
print('\nCHI 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 chi2_corrected

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

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

    # --- >>> START stats <<< ---
    # Calculate the Fisher Exact Test
    # Note that by default, the option "alternative='two-sided'" is set;
    # other options are 'less' or 'greater'.
    fisher_result = stats.fisher_exact(obs)
    # --- >>> STOP stats <<< ---

    # Print the result
    print('\nFISHER -----')
    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])))

    return fisher_result

def cochransQ():
    '''Cochran's Q test: 12 subjects are asked to perform 3 tasks. The outcome
        of each task is "success" or
    "failure". The results are coded 0 for failure and 1 for success. In the
        example, subject 1 was successful
    in task 2, but failed tasks 1 and 3.
    '''

    tasks = np.array([[0,1,1,0,1,0,0,1,0,0,0,0],
                      [1,1,1,0,0,1,0,1,1,1,1,1],
                      [0,0,1,0,0,1,0,0,0,0,0,0]])

    # I prefer a DataFrame here, as it indicates directly what the values mean
    df = pd.DataFrame(tasks.T, columns = ['Task1', 'Task2', 'Task3'])

    # --- >>> START stats <<< ---
    (Q, pVal) = cochrans_q(df)
    # --- >>> STOP stats <<< ---
    print('\nCOCHRAN'S Q -----')
    print('Q = {0:5.3f}, p = {1:5.3f}'.format(Q, pVal))
    if pVal < 0.05:
        print("There is a significant difference between the three tasks.")

def tryMcNemar():
    '''McNemars Test should be run in the "exact" version, even though

```

approximate formulas are typically given in the lecture scripts. Just ignore the statistic that is returned, because it is different for the two options.

In the following example, a researcher attempts to determine if a drug has an effect on a particular disease. Counts of individuals are given in the table, with the diagnosis (disease: present or absent) before treatment given in the rows, and the diagnosis after treatment in the columns. The test requires the same subjects to be included in the before-and-after measurements (matched pairs).

```
'''
f_obs = np.array([[101, 121], [59, 33]])
(statistic, pVal) = mcnemar(f_obs)
print('\nMCNEMAR\'S TEST
-----')
print('p = {0:5.3f}'.format(pVal))
if pVal < 0.05:
    print("There was a significant change in the disease by the treatment.")

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

Listing A.19: multivariate.py

```

''' Analysis of multivariate data
- Regression line
- Correlation (Pearson-rho, Spearman-rho, and Kendall-tau)

'''

# author: Thomas Haslwanter, date: Jun-2013

import pandas as pd
from getdata import getData
from scipy import stats
from numpy import testing

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

    Data from 24 type 1 diabetic patients, relating Fasting blood glucose (mmol/
    l) to mean circumferential shortening velocity (%/sec), derived from
    echocardiography .

    '''

    # Get the data
    data = getData('altman_11_6.txt', subDir=r'..\Data\data_altman')

    df = pd.DataFrame(data, columns=['glucose', 'Vcf'])
    # --- >>> START stats <<< ---
    model = pd.ols(y=df['Vcf'], x=df['glucose'])
    print(model.summary())
    # --- >>> STOP stats <<< ---

    return model.f_stat['f-stat'] # should be 4.4140184331462571

def correlation():
    '''Pearson correlation, and two types of rank correlation (Spearman, Kendall
    )
    comparing age and %fat (measured by dual-photon absorptiometry) for 18
    normal adults.'''

    # Get the data
    data = getData('altman_11_1.txt', subDir='..\Data\data_altman')
    x = data[:,0]
    y = data[:,1]

    # --- >>> START stats <<< ---
    # Calculate correlations
    corr = {}
    corr['pearson'], _ = stats.pearsonr(x,y)
    corr['spearman'], _ = stats.spearmanr(x,y)
    corr['kendall'], _ = stats.kendalltau(x,y)
    # --- >>> STOP stats <<< ---

    print(corr)

    # Assert that Spearman's rho is just the correlation of the ranksorted data
    testing.assert_almost_equal(corr['spearman'], stats.pearsonr(stats.rankdata(
        x), stats.rankdata(y)) [0])

    return corr['pearson'] # should be 0.79208623217849117

if __name__ == '__main__':
    regression_line()

```



```
correlation()
```

Listing A.20: fitLine.py

```

'''
Linear regression fit

Parameters
-----
x : ndarray
    Input / Predictor
y : ndarray
    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 = {3:6.5
        f}'\
        .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]))

```

Listing A.21: anovaTwoway.py

```

''' Two-way Analysis of Variance (ANOVA)
The model is formulated using the "patsy" formula description. This is very
similar to the way
models are expressed in R.

'''

# author: Thomas Haslwanter, date: Jan-2014

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

def anova_interaction():
    '''ANOVA with interaction: Measurement of fetal head circumference,
    by four observers in three fetuses, from a study investigating the
    reproducibility of ultrasonic fetal head circumference data.'''

    # Get the data
    data = getData('altman_12_6.txt', subDir='../Data\data_altman')

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

    # --- >>> START stats <<< ---
    # Determine the ANOVA with interaction
    formula = 'hs ~ C(fetus) + C(observer) + C(fetus):C(observer)'
    lm = ols(formula, df).fit()
    anovaResults = anova_lm(lm)
    # --- >>> STOP stats <<< ---
    print(anovaResults)

    return anovaResults['F'][0]

if __name__ == '__main__':
    anova_interaction()

```

Listing A.22: 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: May-2013

# 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)

    # To get reproducible values, I provide a seed value
    np.random.seed(987654321)

    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})

    # --- >>> START stats <<< ---
    # Fit the model
    model = ols("z ~ x + y", df).fit()

    # Print the summary
    print((model.summary()))
    # --- >>> STOP stats <<< ---

    return model._results.params # should be array([-4.99754526,  3.00250049,
    -0.50514907])

```

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

    # --- >>> START stats <<< ---
    M = np.vstack((np.ones(len(X)), X, Y)).T
    bestfit = np.linalg.lstsq(M,Z)
    # --- >>> STOP stats <<< ---

    print(('Best fit plane:', bestfit))

    return bestfit

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

Listing A.23: regSpector.py

```

'''Linear Regression
Estimation of a linear regression model using the Spector and Mazzeo (1980) data
set.

Documentation:
    data on 32 students TUCE scores
    5 columns with rows = students

    1) Grade ..... post grade
    2) constant .. term
    3) psi
    4) tuce ..... tuce (test of understanding of college economics) score
    5) gpa ..... grade point average

'''

'''
Author: Bruno Rodrigues
Date:   March-2013
Ver:   1.3 (Thomas Haslwanter)
'''

# For this we only need to import statsmodels
import statsmodels.api as sm

def main():
    # We load the spector dataset as a pandas dataframe
    # Of course, you can load your own datasets
    data = sm.datasets.spector.load_pandas()

    # We define y as the endogenous variable, and x as the
    # exogenous variable.
    # Note that if you load your own data, the methods endog
    # and exog will not be available and you will have to
    # explicitly define the endogenous and exogenous variables
    y, x = data.endog, data.exog

    # We do the regression
    reg = sm.OLS(y, x).fit()

    # And here we can see the results in a very nice looking table
    print('SUMMARY -----')
    print((reg.summary()))

    # We can only take a look at the parameter values though
    print('PARAMETERS -----')
    print((reg.params))

    # We can also extract the residuals
    print('RESIDUALS -----')
    print((reg.resid))

    # This line is just to prevent the output from vanishing when you
    # run the program by double-clicking
    input('Done - Hit any key to finish.')

if __name__ == '__main__':
    main()

```


Listing A.24: 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: May-2013

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 data
    data = read_csv(r'..\Data\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('-----')
    model3Results = anova_lm(model3)
    print(model3Results)

    # Just to check the correct run
    return model3Results['F'][0] # should be 156.1407931415788

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

    # Generate the data

    # To get reproducible values, I provide a seed value
    np.random.seed(987654321)

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

    # --- >>> START stats <<< ---
    # 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()

```

```
# --- >>> STOP stats <<< ---

# 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()))

return res.params # should be [ 4.74244177,  2.60675788,  2.03793634]

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

Listing A.25: bootstrap.py

```
''' Example of bootstrapping the confidence interval for the mean of a sample
distribution
Since no bootstrapping is implemented in Python, this function requires
"bootstrap.py", which is included.

Note: The original scikits-bootstrapping module, which works only under
Python 2, is available from http://github.org/cgevens/scikits-bootstrap

'''

# author: Thomas Haslwanter, date: Jan-2014

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

def generate_data():
    # To get reproducible values, I provide a seed value
    sp.random.seed(987654321)

    # 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):
    # --- >>> START stats <<< ---
    # Calculate the bootstrap
    CIs = bootstrap.ci(data=data, statfunction=sp.mean)
    # --- >>> STOP stats <<< ---

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

    return CIs

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

Listing A.26: 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: May-2013

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)
flme1 = 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]
    flme1[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(flme1)
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 significantly different.'.
        format(p)))
else:
    print(('p={0}, the two survival curves are not significantly different.'.
        format(p)))

return(p, chi2)

def main():
    '''The data in this example give the life talbe for motion sickness data
    from an experiment with vertical movement at a frequency of 0.167 Hz and
    acceleration 0.111 g, and of a second experiment with 0.333 Hz and
    acceleration
    of 0.222 g.
    '''

    # get the data
    data1 = getData('altman_13_2.txt', subDir='../Data\data_altman')
    data2 = getData('altman_13_3.txt', subDir='../Data\data_altman')

    # 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
# --- >>> START stats <<< ---
(p, X2) = logrank(data1, data2)
# --- >>> STOP stats <<< ---

return p    # supposed to be 0.073326322306832212

if __name__=='__main__':
    main()

```

A.2 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. Continuous Tests
9. **Discussion of Projects**
10. Categorical Tests
11. Correlation [T]
12. Regression [T]
13. ANOVA [T]
14. Statistical Models
15. **Presentation of Projects**

A.3 To Do

1. Outlier detection
2. Bayesian statistics
3. Improve *statistical modeling*

Bibliography

- [AJ and AG(2008)] Dobson AJ and Barnett AG. *An Introduction to Generalized Linear Models*. CRC Press, 3rd edition, 2008.
- [Altman(1999)] Douglas G. Altman. *Practical Statistics for Medical Research*. Chapman & Hall/CRC, 1999.
- [Anaconda(2014)] Anaconda, 2014. URL <https://store.continuum.io/cshop/anaconda/>.
- [Harms and McDonald(2010)] D. Harms and K. McDonald. *The Quick Python Book (2nd Ed)*. Manning Publications Co., 2010.
- [IPython(2013)] IPython, 2013. URL <http://ipython.org/>.
- [Kaplan(2009)] Daniel Kaplan. *Statistical Modeling: A Fresh Approach*. Macalester College, 2009.
- [Pandas(2013)] Pandas, 2013. URL <http://pandas.pydata.org/>.
- [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.
- [Seaborn(2014)] Seaborn, 2014. URL <http://www.stanford.edu/~mwaskom/software/seaborn/>.
- [Sellke(2001)] M.J.; Berger J.O. Sellke, T.; Bayarri. Calibration of p values for testing precise null hypotheses. *The American Statistician*, 55:62–71, 2001.
- [statsmodels(2013)] statsmodels, 2013. URL <http://statsmodels.sourceforge.net/>.
- [WinPython(2013)] WinPython, 2013. URL <http://winpython.sourceforge.net/>.

Index Topics

- Akaike Information Criterion AIC, 88
- ANOVA, 57
- ANOVA, two-way, 79
- Bayesian Information Criterion BIC, 88
- bias, 20
- blinding, 22
- Bonferroni correction, 59
- bootstrapping, 98
- centiles, 28
- central limit theorem, 33
- coefficient of determination, 71
 - adjusted, 72
- condition number, 92
- confidence interval, 44
- control group, 20
- correlation
 - Kendall's τ , 73
 - Pearson, 71
 - Spearman, 73
- correlation coefficient, 71
- covariate, 74
- Cox regression model, 100
- crossover studies, 21
- cumulative density function, 25, 26
- cumulative frequency, 18
- data
 - categorical, 15
 - numerical, 15
 - ordinal, 15
- design matrix, 81
- distributions
 - Bernoulli, 41
 - binomial, 41
 - chi square, 35
 - continuous, 34
 - discrete, 41, 42
 - exponential, 39
 - F distribution, 37
 - lognormal, 38
 - normal, 30
 - poisson, 41
 - t-distribution, 34
 - uniform, 40
 - weibull, 39
- documentation, 22
- endogenous variable, 74
- error
 - Type I, 48
 - Type II, 48
- exogenous variable, 74
- factor, 22
- frequency tables, 66
- geometric mean, 27
- Holms correction, 60
- hypotheses, 47
- IQR, 28
- Kaplan-Meier survival curve, 99
- kurtosis, 30
- log likelihood function, 87
- Maximum likelihood, 87
- mean, 27
- median, 27
- mode, 27
- Multiple Comparisons, 59
- negative predictive value, 50
- nominal, 15
- normality check, 43
- percentiles, 28
- plots
 - $Q - Q$ plot, 43
 - boxplot, 19
 - histogram, 16
 - kde, 16
 - scatter, 16
 - violinplot, 19
- positive predictive value, 50
- power, 48
- prevalence, 51
- probability density function, 25, 31
- Python
 - ipython, 8
 - Matplotlib, 8
 - pylab, 9
 - pyplot, 8
- random variate, 25
- randomization, 21
- randomized controlled trial, 20

- range, 28
- regressand, 74
- regression, 73
 - multilinear, 75, 79, 80
 - multiple, 82
- regressor, 74
- replication, 22
- ROC curve, 52

- sample selection, 22
- sample size, 48
- scikit-learn, 93
- sensitivity, 50
- shape parameters, 30
- skewness, 30
- specificity, 50
- standard deviation, 28
- standard error, 29
- statistic, 30
- statistical modeling, 81
- survival times, 99

- test
 - t-test, one sample, 55
 - t-test, paired, 56
 - ANOVA, 57, 79
 - binomial, 63
 - chi square, 66
 - Cochran's Q, 65, 69
 - F-test, 58
 - Fisher's exact, 67
 - Friedman, 79
 - Kolmogorov-Smirnov, 44
 - Kruskal-Wallis, 60
 - Levene, 57
 - Lilliefors, 43
 - logrank, 100
 - Mann-Whitney, 56
 - McNemar's, 68
 - Shapiro-Wilk, 44
 - Tukey's, 59
 - variance ratio, 58
 - Wilcoxon signed rank sum, 55
- transformation, 44

- variance, 28
- variate, 33

Python Programs

anovaOneway, 58

anovaTwoway, 79

binomialTest, 63

bootstrapDemo, 98

checkNormality, 44

compGroups, 69

distributionContinuous, 40

distributionDiscrete, 33, 42

distributionNormal, 33

figs.BasicPrinciples, 19

fitLine, 77

getData, 12

gettingStarted, 10

KruskalWallis, 60

linear regression model, 83

modeling, 97

multipleRegression, 80

multipleTesting, 59

multivariate, 76

oneSample, 55

readZip, 10

sampleSize, 50

statsmodelsIntro, 12

survival, 100

twoSample, 56