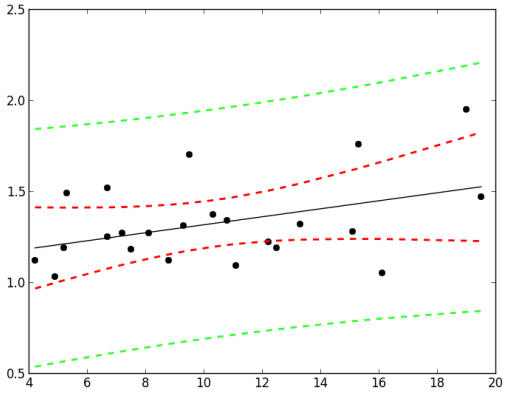


Upper Austrian University of Applied Sciences

An Introduction to

Statistics

Author:
Thomas Haslwanter

email:
thomas.haslwanter@fh-linz.at

Version: 7.0
May 28, 2015

Contents

Preface

In doing the data analysis in my own research work, I was often slowed down by two things: 1) I did not know enough statistics, and 2) the books available would provide a theoretical background, but no real practical help. The book you are holding in your hands (or on your tablet or laptop) is intended to be the book that would have solved just this problem. It should provide enough basic understanding so you know what you are doing; and it should provide you with the tools to do so. In providing statistical solutions for the most basic statistical problem, I believe that I cover at least 90% of the problems that most physicists, biologists, and medical doctors encounter in their work. So if you are the typical graduate student working on your degree, chances are that you will find the solution - explanation and source-code - here.

In contrast, for serious statistical analyses statistical modeling is state-of-the-art. But for most medical research and life science applications, hypothesis tests form the common ground for the statistical analysis. This is the reason I have focussed on statistical basics and hypothesis tests, and give only a brief outlook at other statistical approaches. I am well aware that most of the tests presented in this book can also be done with the the methods of statistical modeling. But in many cases, this is not the jargon used in typical life science journals. Advandes statistical analysis goes beyond the scope of this book, and - to be frank - beyond my statistical knowledge.

My motivation to provide the solutions in Python are based on two considerations. One is that I would like them to be available to everyone. While commercial solutions like Matlab, SPSS, Minitab etc. are available, most of us can only use them legally as long as they are in academia. In contrast, Python is completely free, as in "free beer". The second reason is that Python is the most beautiful coding language that I have yet encountered; and around 2010 Python and its documentation had matured to the point where one could use it without being a hacker. All together, this book and Python provides a free, beautiful package that covers all the statistics that most researchers need to do in their lifetime. OK, for really serious statistical modeling :math:'R' still sets the standard. But most will be more than happy with the tools that the Python ecosystem offers today.

For whom this book is

This book assumes that

- you have some basic programming experience (If you have zero prior programming experience, you may want to start out with getting going with Python, using some of the great links given in the text. Starting programming *and* starting statistics may be a bit much at a time.)
- you have some data that you want to analyze (For almost all cases, a working Python program is provided. All you have to do is select the right program, adjust it so that it reads in your data, and interpret the results.)
- you are familiar with the basic ideas of statistics, but are not a statistics expert (If you are already a statistics expert, the online help in Python will be sufficient to allow you to do most of your data anlysis right away.)

The idea of this book is to give you all (or at least most of) the tools that you will need for your statistical data analysis. Thereby I try to provide all the background required to understand what you are doing. I will not proof any theorems, and won't indulge in mathematics where it is unnecessary. This approach explains why so much code is included: in principle, you have to define our problem, select the corresponding program, and adapt it to your needs. This should allow you to get going quickly, even if you have little Python experience. This is also the reason why I have not provided the software as a Python module, since I expect that you have to tailor each program to your specific setup (data format, etc).

How to use this book

- If you just want to look something up, simply go to the HTML-version of the book.
- If you want to go through it systematically, or if you prefer to read printed material, you may want to download the PDF-version of the book.
- If you want to get the whole package, and/or if you want to contribute to the book, clone the github repository of the book, which includes all the Python programs, the sample data used in the book, the TEX-files, RST-files, and all the images.

  If you have never used github, you might want to check out this introduction to github. But don't be scared off, you can download individual files easily from your web-browser.

Chapter 1 gives an introduction to the book, especially to the Python programming environment that we are going to use.

Chapter 2 proceeds with an introduction to statistical analysis.

Chapter 3 provides the basis on which statistic rests: continuous and discrete distribution functions.

Chapters 4-11 form the heart of the introduction: they introduce the different statistical tests, and give examples (including the Python code) on how to use them.

Chapters 12-14 provide an outlook to advanced statistical analysis procedures, with an introduction to statistical modeling in Chapter 13, and a presentation of the basic ideas of Bayesian Statistics in Chapter 14.

Code samples are marked as follows

Python code samples, listed in the Appendix.

Contributor List

If you have a suggestion or correction, please send email to thomas.haslwanter@fh-linz.at. If I make a change based on your feedback, I will add you to the contributor list (unless you ask to be omitted).

If you include at least part of the sentence the error appears in, that makes it easy for me to search. Page and section numbers are fine, too, but not as easy to work with. Thanks!

- Connor Johnson wrote a very nice blog explaining the results of statsmodels OLS command, which formed the basis of a large part of the section on *Statistical Models*.
- To demonstrate Bayesian statistics and MCMC-models, I took the example of the Challenger disaster from the excellent open source e-book Probabilistic-Programming-and-Bayesian-Methods-for-Hackers by Cam Davidson Pilon.

- Fabian Pedregosa's blog on ordinal logistic regression allowed me to include a topic that is admittedly beyond my own skills.

Chapter 1

Introduction

"Statistics is 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"* ("I have had to be diligent; if you are diligent too, you will get as far as I have."). 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 estimated parameters.
- 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 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)]. [Kaplan(2009)] provides a simple introduction to modern regression modeling. A very good introduction to Generalized Linear Models is [Dobson and Barnett(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://www.biostathandbook.com/
- http://onlinestatbook.com/2/index.html
- http://www.itl.nist.gov/div898/handbook/index.htm

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

Exercises: The solutions to a number of examples are provided in the Appendix. 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 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, the interpretation of data can quickly become 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.

Chapter 2

Python

2.1 Getting Started

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)].

There are also many tutorials available on the internet (see links below). 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!

2.1.1 Python Links

- Python Scientific Lecture Notes. If you don't read anything else, read this!
- NumPy for Matlab Users Start here if you have Matlab experience.
- Lectures on scientific computing with Python. Great IPython notebooks, from JR Johansson!
- The Python tutorial. The official introduction.

2.1.2 Free Python Books

- A Byte of Python. Free book, very good at the introductory level.
- Learn Python the Hard Way, 3rd Ed A popular, free book that you can work through.
- ThinkPython. Free book, for advanced programmers.
- Introduction to Python for Econometrics, Statistics and Data Analysis (pdf) by Kevin Sheppard: A good free book, which introduces Python with a focus on statistics.

2.1.3 Installation and Updates

In general, I suggest that you start out by installing a Python distribution which includes the most important libraries. Unless you have a specific requirement for 64-bit versions, I recommend that you install a 32-bit version of Python: it facilitates many activities that require compilation of module parts, e.g. for Bayesian statistics (PyMC), or when you want to speed up your programs with Cython. Since all the Python packages required for this course are now available for Python 3.x, I will use Python 3 for this book. However, all the scripts included should also work for Python 2.7. Make sure that you use a current vrsion of IPython (3.x), since the IPython notebooks provided with this book won't run on IPython 2.x. My favorite Python distributions are

1. WinPython Recommended for Windows users.
2. anaconda From Continuum. For Windows, Mac, and Linux.

Neither of these two distributions required administrator rights. Personally I am using WinPython, which is free and customizabla. *anaconda* is a more recent distribution, and is free for educational purposes.

Mac and Unix users should check out the installations tips from Johansson.

The programs included in this book have been tested under Windows and Linux, using the following package versions:

- *ipython 3.1.0* ... For interactive work.
- *numpy 1.9.2* ... For working with vectors and arrays.
- *scipy 0.15.1* ... All the essential scientific algorithms, including those for statistics.
- *matplotlib 1.4.3* ... The de-facto standard module for plotting and visualization.
- *pandas 0.16.0* ... Adds *DataFrames* (imagine powerful spreadsheets) to Python.
- *patsy 0.3.0* ... For working with statistical formulas.
- *statsmodels 0.6.1* ... For statistical modeling and advanced analysis.
- *seaborn 0.5.1* ... For visualization of statistical data.
- *PyMC 2.3.4* ... For Bayesian statistics, including Markov chain Monte Carlo simulations.
- *scikits.bootstrap 0.3.2* ... Provides bootstrap confidence interval algorithms for scipy.

2.1.4 PyPI - the Python Package Index

The Python Package Index (*PyPI*) is a repository of software for the Python programming language. It currently contains about 60'000 packages!

Packages in this index can be installed easily, from the Windows *command shell ("cmd")* or the Linux *Terminal*, with

```
pip install [package]
```

and updated with

```
pip install [package] -U
```

To get a list of all the Python packages installed on your computer, type

```
pip list
```

2.2 IPython

Make sure that you have a good programming environment! For me, the most efficient way to write new code is as follows: I first get the individual steps worked out interactively in an ipython *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.

2.2.1 First Session with the IPython Qt Console

An important aspect of statistical data analysis is the interactive, visual inspection of the data. Therefore I strongly recommend to start the data analysis in the *IPython Qt Console*. To get you started with python, I will go step-by-step through a short IPython session (Fig. 2.2).

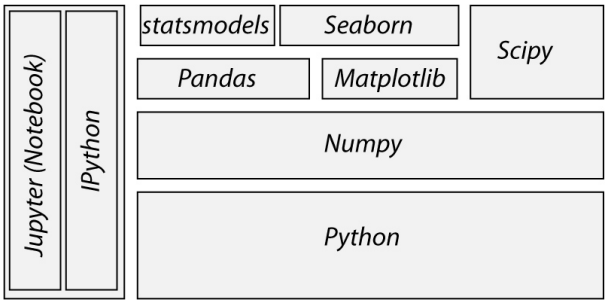


Figure 2.1: The structure of the most important Python packages for statistical applications.

For maximum flexibility, I started my session from the *WinPython Command Prompt*, with the command `ipython qtconsole`. The *WinPython Command Prompt* is nothing else but a command terminal, with the environment variables set such that Python is readily found.

First IPython starts out telling you about the version of IPython and Python you are using, and listing the most important help calls.

- **In [1]:** The first command `%pylab inline` loads numpy and matplotlib into the current workspace, and directs matplotlib to show plots *inline*. (This is automatically done if IPython is started with the commandline option "–pylab=inline".)

  To understand what is happening here requires a short detour into the structure of scientific Python. Fig.2.1 shows the connection of the most important Python packages that are used in this book.

  Python itself is an interpretative programming language, with no optimization for working with vectors or matrices or for producing plots. *Packages* which extend the abilities of Python must be loaded explicitly. The most important package for scientific applications is *numpy*, which makes working with vectors and matrices fast and efficient, and *matplotlib*, which is the most common package used for producing graphical output. *scipy* contains important scientific algorithms. For the statistical data analysis, `scipy.stats` contains the majority of the algorithms that will be used in this book. *pandas* is a more recent addition, which has become widely adopted for statistical data analysis. It provides *DataFrames*,

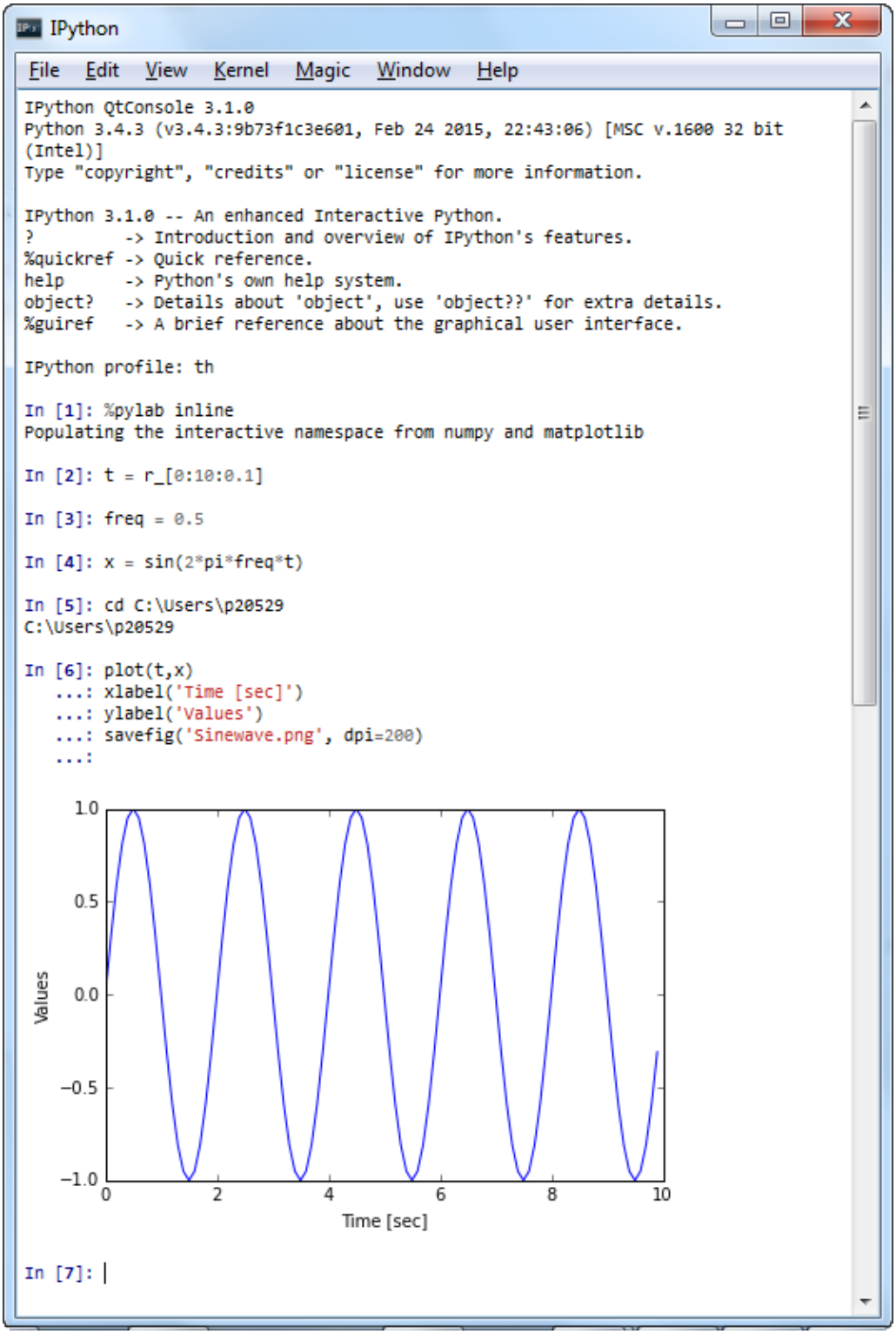


Figure 2.2: Sample session in the IPython Qt-console.

which are labelled, 2-dimensional data structures, making work with data more intuitive and clearer. *seaborn* extends the plotting abilities of *matplotlib*, with a focus on statistical graphs. And *statsmodels* contains many modules for statistical modeling, and for advanced statistical analysis. Both *seaborn* and *statsmodels* make use of the *pandas'* DataFrames.

IPython provides the tools for interactive data analysis. It lets you quickly display graphs and change directories, explore the workspace, provides a command history etc. The ideas and base structure of IPython have been so successful that one component, the *IPython Notebook* has been turned into a project of its own, *Jupyter*, which is now also used by other languages like Julia, R, and Ruby.

- **In [2]:** The command `t = r_[0:10:0.1]` is a shorthand version for `t = arange(0, 10, 0.1)`, and generates a vector from 0 to 10, with a step size of 0.1. `_r` (and `arange`) is a command in the numpy package. However, since `numpy` has already been imported into the current workspace by IPython, we can use these commands right away.

- **In [4]:** Since t is a vector, and *sin* is a function from numpy, the sine-value is calculated automatically for each value of t.

- **In [5]:** In Python scripts, changes of the current folder have to be performed with `os.chdir()`. However, common task with interactive computing, like directory changes (`%cd`), bookmarks for directories (`%bookmark`), inspection of the workspace (`%who` and `%who`), etc, are implemented as *IPython magic functions*.

- **In [6]:** Since we started out with the command `%pylab inline`, IPython generates plots in the Qt-console, as shown in Fig. 2.2. I have mentioned above that *matplotlib* handles the graphics output. In the IPython Qt-console, you can switch between inline graphs and output into a separate graphics-window with `%matplotlib inline` and `%matplotlib qt4` (see Fig.2.3). This allows you to zoom in and pan, get the cursor position (which can be handy in helping to find outliers), and get interactive input with the command `ginput`. matplotlib's plotting commands closely follow the MATLAB conventions. Note that also generating graphics files is very simple (here I generate the PNG-file "Sinewave.png", with a resolution of 200 dots-per-inch.)

2.2.2 Personalizing IPython

When working on a new problem, I always start out with IPython. Once I have the individual steps working, I use the IPython command *%history* to get the commands I have used, and switch to an integrated development environment (typically *Wing* or *Spyder*).

To start up IPython quickly in the location and with the configuration I like, I use the following tricks:

To personalize IPython, generate your own profile as shown below. Thereby, *myname* has replaced with your login name, and *mydir* with your home-directory (i.e. the directory that opens up when you run `cmd` in Windows, or `terminal` in Linux):

In Windows

- Type Win+R, and start a command shell with `cmd`

- In the newly created command shell, execute the following commands:

```
ipython profile create myname
```

(This generates a folder *.ipython\profile_myname\startup*")

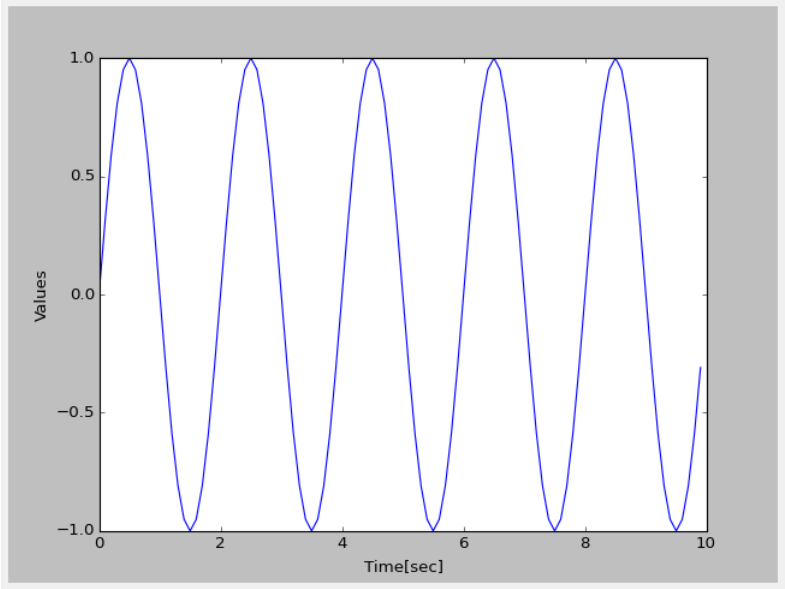


Figure 2.3: Graphical output window, using the Qt4-framework. This allows you to pan, zoom, and get interactive input.

- Into this folder, place a file with e.g. the name 00_*myname.py*, containing

```
import pandas as pd
import os
os.chdir(r'C:\mydir')
```

  Note: since Windows uses "̈to separate directories, but "̈is also the escape character in strings, directory paths have to be preceded by "r", indicating *raw strings*.

- Generate a file "ipy.bat" in *mydir*, containing

```
[Python-directory]\Scripts\ipython3 qtconsole --profile myname --
    pylab=inline
```

To see all ipython notebooks for the course, do the following:

- Type Win+R, and start a command shell with `cmd`
- Run the commands

```
cd [ipynb-directory]
[Python-directory]\Scripts\ipython3 notebook --pylab=inline
```

- Again, if you want, you can put this command sequence into a batch-file.

In Linux

- Start a Linux terminal with the command `terminal`
- In the newly created command shell, execute the following command

```
ipython profile create myname
```

  (This generates a folder *.ipython/profile_myname/startup*")

- Into this folder, place a file with e.g. the name 00_*myname.py*, containing

```
import pandas as pd
import os
os.chdir(mydir)
```

- In your .bashrc file, enter the line

```
alias ipy='ipython3 qtconsole --profile myname --pylab=inline'
```

- To see all IPython notebooks, do the following:
  - Go to *mydir*
  - Create the file *ipynb.sh*, containing the lines

```
#!/bin/bash
cd [wherever_you_have_the_ipynb_files]
ipython3 notebook
```

  - Make the file executable, with `chmod 755 ipynb.sh`

Now you can start "your" ipython by just typing `ipy`, and the notebooks by typing "ipynb.sh"

In OSX

Here the procedure should be the same as in Linux [CHECK!!!]

2.2.3 IPython Tips

1. Use IPython in the qtconsole, and start it in *pylab*-mode: pylab: `ipython qtconsole ---pylab=inline`
2. For help on e.g. `plot`, use `plot?` or `help(plot)`.
3. Check out the help tips when you start ipython.
4. Customize ipython on your computer, as described above: it will save you time in the long run!
5. TAB-completion, for file- and directory names, variable names, AND for commands.
6. To switch between inline and external graphs, use `%matplotlib inline` and `%matplotlib qt4`.

2.3 Developing Python Programs

2.3.1 First Python Script

IPython is very helpful in working out the command syntax and sequence. The next step is to turn this in to a Python program that can be run from the command-line. This section introduces a fairly large number of Python conventions and syntax.

An efficient way to to turn IPython commands into a function is to

- first obtain the command history with the command `%hist`.
- copy the history into a good IDE (e.g. *Wing*, see below)
- turn it into a working Python program by adding the relevant package information, etc.

In this case, this leads to

Listing 2.1: pythonScript.py

```
'''
Short demonstration of a Python script.

author: Thomas Haslwanter
date:   May-2015
ver:    1.0

'''

# Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import os

# Generate the time-values
t = np.r_[0:10:0.1]

# Set the frequency, and calculate the sine-value
freq = 0.5
x = np.sin(2*np.pi*freq*t)

# Plot the data
plt.plot(t,x)

# Format the plot
plt.xlabel('Time[sec]')
plt.ylabel('Values')

# Change the directory, and generate a figure
os.chdir(r'C:\Users\p20529')
plt.savefig('Sinewave.png', dpi=200)

# Put it on the screen
plt.show()
```

The following modifications were made from the IPython history:

- Files with the extension *.py* are called *Modules*.
- **1-8:** It is comment to precede a Python module with a header block. Multiline comments are given between " "' ". The first comment block describing the module should also contain information about author, date, and version number.
- **10:** Single-line comments use " # ".
- **11-13:** The required Python packages have to be imported explicitly. (In IPython, this is done by the command `%pylab`.) It is customary to import *numpy as np*, and *matplotlib.pyplot*, the matplotlib module which contains all the plotting commands, as *plt*.
- **16 etc:** The numpy command *r_* has to be addressed through the corresponding package name, i.e. `np.r_`. (In IPython, *%pylab* took care of that.)
- **20:** Note that also *pi* is in numpy, so `np.pi` is needed!
- **23 etc:** All the plotting commands have to be preceded by their corresponding module, *plt.*.
- **30:** Changes of the current directory require the `os` package. In Windows, you have to deal with the additional problem that directories in path-names are separated by "", which is also used as the escape-character in strings. To take "" literally, a string has to be preceded by "r" (for "r"aw string).

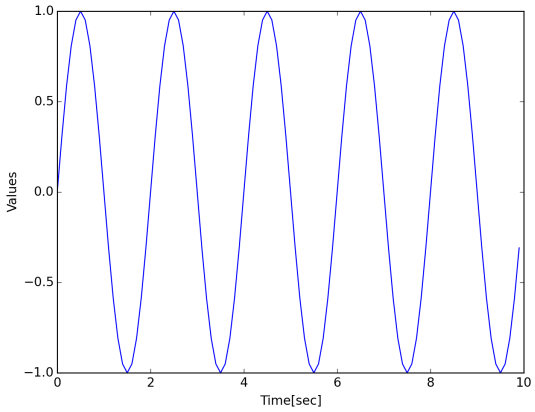


Figure 2.4: Output file from "pythonScript.py".

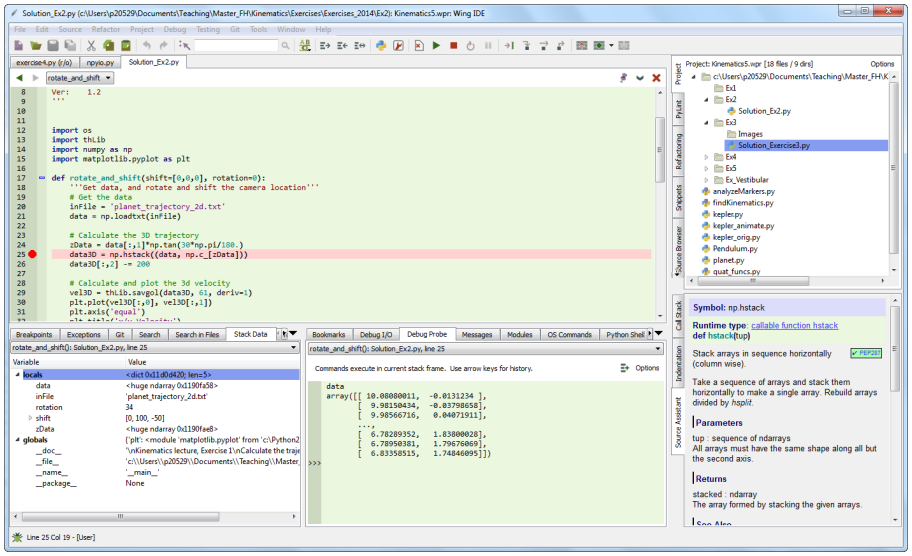


Figure 2.5: *Wing* is my favorite development environment, with probably the best existing debugger for Python.

- **34:** While IPython automatically shows graphical output, Python programs don't show the output until this is explicitly requested by `plt.show()`. The idea behind this is to optimize the program speed, only showing the graphical output when it is all finished.

To write a program, I typically take the commands I have worked out in ipython with `% history`, and take them to an IDE (integrated development environment): I either use *Wing* (my clear favorite Python IDE, although it is commercial) or *Spyder* (which is good and free). *PyCharm* is another IDE with a good debugger, and has very good vim-emulation.

2.3.2 Functions, Modules, and Packages

Python has different levels of modularization:

Function Are defined by the keyword `def`, and can be defined anywhere in Python. They return the object in the `return` statement, typically at the end of the function.

Modules Python *Modules* are files with the extension *.py*. They can contain function definitions, as well as valid Python statements.

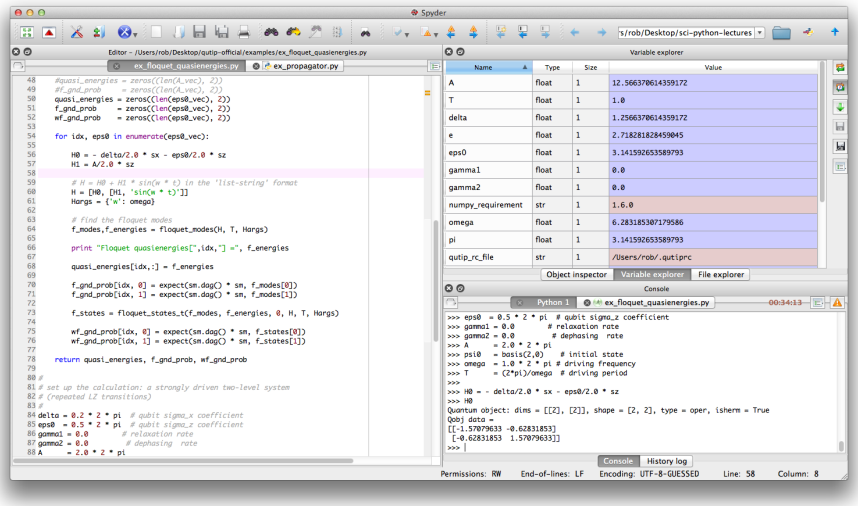


Figure 2.6: *Spyder* is a very good, free IDE.

Packages Python *packages* are folders containing multiple Python modules, and must have a file named *__init__.py*. For example, *numpy* is a Python package.

Functions

The following example shows how functions can be defined and used.

Listing 2.2: pythonFunction.py

```
1  '''Demonstration of a Python Function
2
3  author: thomas haslwanter, date: May-2015
4  '''
5
6  import numpy as np
7
8  def incomeAndExpenses(data):
9      '''Find the sum of the positive numbers, and the sum of the negative ones.
           '''
10     income = np.sum(data[data>0])
11     expenses = np.sum(data[data<0])
12
13     return (income, expenses)
14
15 if __name__=='__main__':
16     testData = np.array([-5, 12, 3, -6, -4, 8])
17
18     # If only real banks would be so nice ;)
19     if testData[0] < 0:
20         print('Your first transaction was a loss, and will be dropped.')
21         testData = np.delete(testData, 0)
22     else:
23         print('Congratulations: Your first transaction was a gain!')
24
25     (myIncome, myExpenses) = incomeAndExpenses(testData)
26     print('You have earned {0:5.2f} EUR, and spent {1:5.2f} EUR.'.format(
           myIncome, -myExpenses))
```

- **1-4:** Comment header.

- **6:** Since numpy will be required in that module, it has to be imported. To reduce the writing to a minimum, it is conventionally called *np*.
- **8/9:** Function definition, and a comment describing the function. Note that the function block is defined by the indentation, not by any brackets of *end* statements! This is a feature that irritates many Python novices, but that really helps to keep code clear and nicely formatted. Important: Python makes a difference between a tab and the equivalent amount of spaced. This can lead to errors which are really hard to detect, so you should use a good IDE that automatically converts tabs to spaces!
- **10:**
  - The `sum` command is taken from *numpy*, so it has to be preceded by `.np`.
  - In Python, function arguments are indicated by round brackets (*(...)*, whereas elements of lists, tuples, vectors, and arrays are indicated by square brackets (*[...]*).
  - In *numpy* you can select elements of an array either with an in index (see line **19**), or with a boolean array as here (line **10**).
- **13:** Python also uses round brackets to form groups of elements, so-called *tuples*. And the `return` statement does the obvious things: it returns elements from a function.
- **15:** Here quite a few new aspects of Python come together:
  - Just like function definitions, *if*-loops or *for*-loops use indentation to define their context.
  - Python conventionally uses underscores (_-) to indicate private variables, which are not used for typical programming tasks.
  - Here we check the name `\_\_name\_\_`, which is denoting the context of a module evaluation. If the module is run as a Python script, `\_\_name\_\_` is set to *__main__*. But if a module is imported, it is set to the name of the importing module. This way it is possible to add code to a function that is only used when the module is executed, but not when the functions in this module are `import`ed by other modules (see below).
- **16:** Definition of a *numpy* array.
- **25:** When two elements are returned from a function *(incomeAndExpenses*, they can be immediately assigned to two different Python objects (*myIncome, myExpenses*.
- **26:** While there are different ways to produce formatted strings, this is probably the most elegant one: curly brackets (*"..."*) indicate values that will be inserted, and can also contain formatting statements. The corresponding values are then passed into the string by the method `format`.

Modules

To *execute* a module from the commandline, type `python pythonFunction.py`. In Windows, if the extension *.py* is associated with the Python program, it suffices to double-click the module, or to type `pythonFunction.py` on the commandline. In WinPython the association of the extension *.py* with the Python function is set by the *WinPython Control Panel.exe*, by the command *"Register Distribution ..."* in the menu *"Advanced"*.

To run a module in IPython, use the magic function *%run*:

```
In [56]: %run pythonFunction
Your first transaction was a loss, and will be dropped.
You have earned 23.00 EUR, and spent 10.00 EUR.
```

Note that you either have to be in the directory where the function is defined, or you have to give the full pathname.

If you import a module multiple times, Python recognizes that the module is already known, and skips later imports. If you want to override this, and explicitly want to re-import a module that has changed, you have to use the command *reload* from the package *importlib*:

```
from importlib import reload
reload(pythonFunction)
```

The next example shows you how to import functions from one module into another one:

Listing 2.3: pythonImport.py

```
'''Demonstration of importing a Python module

author: ThH, date: May-2015'''

import numpy as np
import pythonFunction

# Generate test-data
testData = np.arange(-5, 10)

# Use a function from the imported module
out = pythonFunction.incomeAndExpenses(testData)

# Show some results
print('You have earned {0:5.2f} EUR, and spent {1:5.2f} EUR.'.format(out[0], -
    out[1]))
```

- textbf6: Here the module `pythonFunction` (that we have just discussed above) is imported. Note that the code in the section `if \_\_name\_\_ == '\_\_main\_\_'` is NOT executed when the module is imported!
- textbf12: To access the function *incomeAndExpenses* from the module *pythonFunction*, I have to use `incomeAndExpenses.pythonFunction(...)`

2.3.3 Python Tips

1. Stick to the standard conventions.
   - Every function should have documentation at the top.
   - ```
     import matplotlib.pyplot as plt
     import numpy as np
     import scipy as sp
     import pandas as pd
     import seaborn as sns
     ```
2. To get the current directory, use `os.path.abspath(os.curdir)`.
3. Everything in Python is an object: to find out about "obj", use `type(obj)` and `dir(obj)`.
4. Learn to use the debugger.
5. Know *lists*, *tuples*, and *dictionaries*; also, know about *numpy arrays* and *pandas DataFrames*.
6. Use functions a lot, and understand the `if __name__=='__main__':` construct.
7. If you have all your personal functions in the directory *mydir*, you can add this directory to your PYTHONPATH with the command
     ```

```
import sys
sys.path.append('mydir')
```

2.4 Python Data Structures

Tuple () A collection of different things. Tuples are *immutable*, i.e. they cannot be modified after creation.

List [ ] Lists are *mutable*, i.e. their elements can be modified. Therefore lists are typically used to collect items of the same type (numbers, strings, ...).

Array [ ] Vectors and matrices, for numerical data manipulation. Defined in *numpy*.

Dictionary {} Dictionaries are unordered *(key/value)* collections of content, where the content is addressed as *dict['key']* (see example below).

DataFrame Data structure optimized for working with named, statistical data. Defined in *pandas*. (See next chapter.)

```
In [1]: myTuple = ('abc', np.arange(0,3,0.2), 2.5)

In [2]: myTuple[2]
Out[2]: 2.5

In [3]: myList = ['abc', 'def', 'ghij']

In [4]: myList.append('klm')

In [5]: myList2 = [1,2,3]

In [6]: myList3 = [4,5,6]

In [7]: myList2 + myList3
Out[7]: [1, 2, 3, 4, 5, 6]

In [8]: myArray = np.array(myList2)

In [9]: myArray2 = np.array(myList3)

In [10]: myArray + myArray2
Out[10]: array([5, 7, 9])

In [11]: myDict = dict(one=1, two=2, info='some information')

In [12]: myDict2 = {'ten':1, 'twenty':20, 'info':'more information'}

In [13]: myDict['info']
Out[13]: 'some information'

In [14]: myDict.keys()
Out[14]: dict_keys(['one', 'info', 'two'])
```

2.4.1 Indexing and Slicing

For addressing individual elements in Python lists or tuples, or in numpy arrays, the following conventions are used (Fig. 2.7):

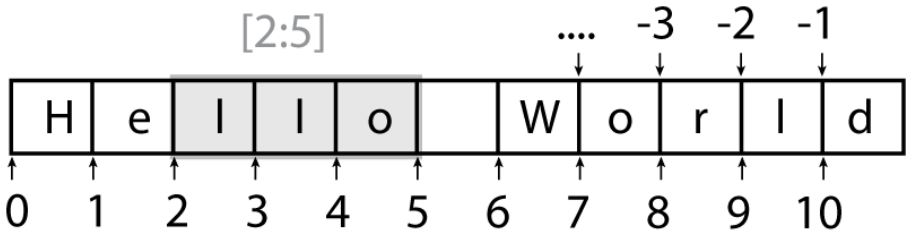


Figure 2.7: Conventions for Python Indexing and Slicing.

- Indexing starts at 0, NOT at 1.
- The index refers to the *pointer*, not to the variable.
- When one element is addressed, the element indicated by the pointer is selected. For example, with `data = 'Hello World', data[1] gives the letter \emph{"e"`.
- Addressing a *slice of data* (gray shaded area in Fig. 2.7) returns the elements between the selected pointers. E.g. `data[2:5]` returns *"el"*. The number of elements returned (here *3*) equals the difference between the selected pointers (here *5-2*).
- When slicing, the third parameter indicates the step size. E.g. `data[1::3]` produces *"eood"*.
- You can also use negative indices, corresponding stepping backwards from the end of the object.

Vectors and Arrays

numpy is the Python module that makes working with numbers efficient. By default, it produces vectors. The commands most frequently used to generate numbers are

np.zeros Generates zeros. Note that it takes only *one* input. If you want to generate a matrix of zeroes, this input has to be a tuple containing the number of rows/columns!

np.ones Generates ones.

np.random.randn Generates normally distributed numbers, with a mean of 0 and a standard deviation of 1.

np.arange Generates a range of numbers. Parameters can be *start, end, steppingInterval*. Note that the end-value is *excluded*!

np.linspace Generates linearly spaced numbers.

np.array Generates a numpy array from the given data.

```
import numpy as np

np.zeros(3)
>>>  array([ 0.,  0.,  0.])

np.zeros( (2,3) )
>>> array([[ 0.,  0.,  0.],
           [ 0.,  0.,  0.]])

np.arange(3)
>>> array([0, 1, 2])
```

```
np.linspace(0,10,6)
>>> array([  0.,   2.,   4.,   6.,   8.,  10.])

np.arange(1,3,0.5)
>>> array([ 1. ,  1.5,  2. ,  2.5])

array([[1,2], [3,4]])
>>> array([ [1, 2],
            [3, 4] ])
```

There are a few points that are peculiar to Python, and that are worth noting:

- Matrices are simply *lists of lists*. Therefore the first element of a matrix gives you the first row:

```
import numpy as np
Amat = np.array([ [1,2], [3,4] ])
Amat[0]
>>> array([1, 2])
```

- A vector is not the same as a 1-dimensional matrix! This is one of the few un-intuitive features of Python, and can lead to mistakes that are hard to find. For example, vectors cannot be transposed, but matrices can.

```
import numpy as np

x = np.arange(3)
Amat = np.array([ [1,2], [3,4] ])

x.T == x
>>> array([ True,  True,  True], dtype=bool)

Amat.T == A
>>> array([[ True, False],
           [False,  True]], dtype=bool)
```

2.5 Plots in Python

The Python core does not include any possibilities to generate plots. This is added by other packages. The by far most common package used for plotting is Matplotlib. Matplotlib is not part of the core of Python. But if you installed Python with a scientific package like *WinPython* or *anaconda*, it will be included. Matplotlib is intended to mimic the style of Matlab. As such, users can either generate plots in the MATLAB style, or in the traditional Python style (see below).

Matplotlib contains a number of different modules and features:

matplotlib.pyplot This is the module that is commonly used to generate plots, and is by convention imported in Python functions and scripts with `import matplotlib.pyplot as plt`. *pyplot* provides the interface to the plotting library in matplotlib. Pyplot handles a lot of little details, such as creating figures and axes for the plot, so that the user can concentrate on the data analysis.

matplotlib.mlab Contains a number of functions that are commonly used in MATLAB, such as `find`, `griddata`, etc.

"backends" Matplotlib can produce output in many different formats, which are referred to as *backends*:

- In an *Ipython Notebook*, or in an *IPython Qt-console*, the command `%matplotlib inline` directs output into the current browser window. (`%pylab inline` is a combination of loading pylab, and directing plot-output inline.
- In the same environment, `%matplotlib qt4` directs the output into a separate graphics window (Fig. 2.3). This allows panning and zooming the plot, and interactive selection of points on the plot by the user.
- In addition, output can be directed to external files, e.g. in PDF, PNG, or JPG format.

pylab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and numpy (for mathematics and working with arrays) in a single name space. Although many examples use pylab, it is no longer recommended, and should only be used in IPython, to facilitate interactive development of code.

2.5.1 Interactive plots

Matplotlib provided different ways how to interact with the user. Unfortunatly, this interaction is less intuitive than in Matlab. To bypass most of these problems, you can base your code on the examples below:


Code: "interactivePlots.py" (p 168) gives a short demonstration of Python for scientific data analysis.

2.5.2 Graphical Output in Python: Functional and Object-oriented Approach

Python plots can be generated in a MATLAB-like style, or in an object oriented, more pythonic way. These styles are all perfectly valid, and each have their pros and cons. The only caveat is to avoid mixing the coding styles for your own code.

First, consider the frequently used pyplot style:

```
# Import the required packages, with their conventional names
import matplotlib.pyplot as plt
import numpy as np

# Generate the plot
x = np.arange(0, 10, 0.2)
y = np.sin(x)
plt.plot(x, y)

# Display it on the screen
plt.show()
```

Note that the creation of the required figure and an axes is done automatically by pyplot.

Second, a more pythonic, object oriented style, which may be clearer when working with multiple figures and axes:

```
# Import the required packages
import matplotlib.pyplot as plt
import numpy as np

# Generate the data
x = np.arange(0, 10, 0.2)
y = np.sin(x)

# Generate the (first) plot
fig = plt.figure()
```

```
# On that plot, add one axis
ax = fig.add_subplot(111)

# On that axis, plot the x/y-data
ax.plot(x, y)

# Display the resulting plot
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.

Example-Session

 **Code:** "gettingStarted.py" (p 165) gives a short demonstration of Python for scientific data analysis.

2.6 Pandas

pandas is a widely used Python package which has been contributed by Wes McKinney. It provides data structures suitable for statistical analysis, and adds functions that facilitate data input, data organization, and data manipulation. It is common to `import pandas as pd`, which reduces the typing a bit.

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

2.6.1 Data Handling

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, ask for several variables in a 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 there 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 extract rows and columns in one go, use

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

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

```
df.iloc[[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
```

Note: Data selection

While pandas' DataFrames are similar to numpy arrays, their philosophy is different, and I have wasted a lot of nerves addressing data correctly. Therefore I want to explicitly point out the differences here:

numpy handels *rows* first. E.g. data[0] is the first row of an array

pandas starts with the columns. E.g. df['values'][0] is the first element of the column 'values'.

If a DataFrame has labelled rows, you can extract for example the row "rowlabel" with with `df.loc['rowlabel']`. If you want to address a row by its number, e.g. row number "15", use `df.iloc[15]`. You can also use *iloc* to address *rows/columns*, e.g. `df.iloc[2:4,3]`.

Slicing of rows also works, e.g. `df[0:5]` for the first 5 rows. A sometimes confusing convention is that if you want to slice out a single row, e.g. row "5", you have to use `df[5:6]`. If you use `df[5]` alone, you get an error!

2.6.2 Grouping

Pandas offers powerful functions to handle missing data and "nans". It also allows more complex types 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')

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

# Plot the data:
grouped.boxplot()

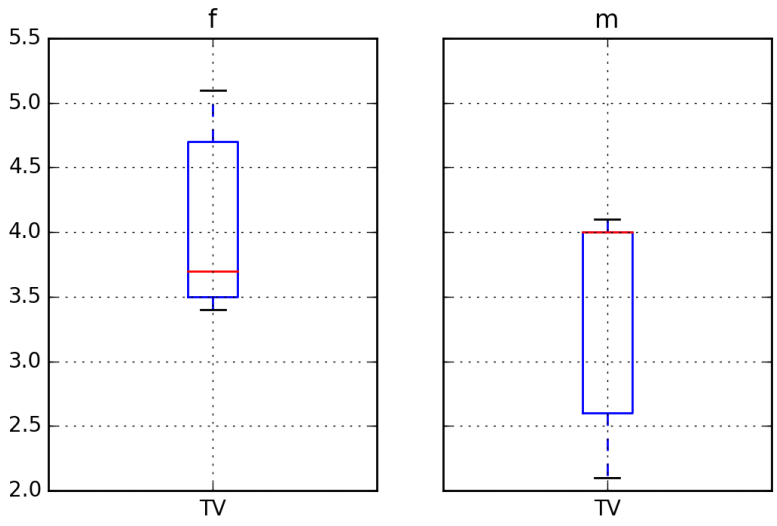
#----------------------------------------
# Get the groups as DataFrames
df_female = grouped.get_group('f')

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

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

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).

2.7 Statsmodels

statsmodels is a Python package contributed to the community by the statsmodels development team. It has a very active user community, and has in the last five years massively improved the suitability of Python for statistical data analysis. *statsmodels* 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. *statsmodels* also allows the formulation of models with the popular formula language based on the notation introduced by Wilkinson and Rogsers ([Wilkinson and Rogers(1973)], and also used by *S* and *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 172) shows you how the combination of pandas and statsmodels can be used for data analysis.

2.8 Seaborn

seaborn 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.

```
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 (Fig. 2.8).

2.8.1 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 173) Gets the input data for many Python programs in this script.

2.9 Exercises

- Read in data from different sources:
  - A CVS-file with a header ('Data\data_kaplan\swim100m.csv')
  - An MS-Excel file ('Data\data_others\GLM_data\Table 6.6 Plant experiment.xls')
  - Data from the WWW (see "readZip.py" A.3)

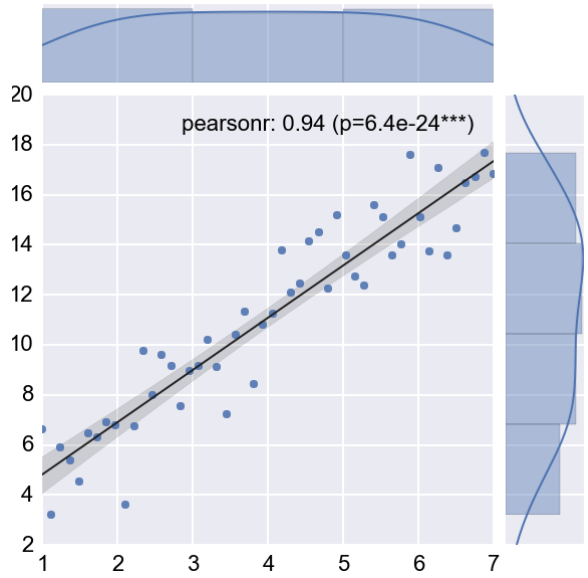


Figure 2.8: Regression plot, from *seaborn*. The main axis shows the data, the best-fit line, and the confidence intervals for the fit. It also provides the Pearson Correlation Coefficient, and the p-value for the inclination. The additional axes on top and on the right show histogram and Kernel-Density-Plots for the x- and y-data, respectively.

- – 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 3

Data Input

This chapter deals with how to get read in data for statistical analysis (in Python), and thus forms the link between the chapter on *Python* and the first chapter on data analysis. And while it may be hard to believe, reading data into the system in the correct format is often one of the most time consuming parts of the data analysis.

Data input can be complicated by a number of problems, like different separators between data entries (such as spaces and/or tabulators), or empty lines at the end of the file. In addition, data may have been saved in different formats, such as MS Excel, Matlab, HDF5 (which also contains the Matlab-format), or in databases. Understandably, we cannot cover all possible combinations here. But I will try to give an overview of where and how to start with data input.

3.1 Data from Textfiles

3.1.1 Visual Inspection

When your data are available as ASCII-files, you should *always* start out with a *visual inspection* of the data. In particular, you should check

- Do the data have a header and/or a footer?
- Are there empty lines at the end of the file?
- Are there white-spaces before the first number, or at the end of each line? (The latter is a lot harder to see.)
- Are the data separated by tabulators, and/or by spaces? (Tip: you should use a text-editor which can visualize tabs, spaces, and end-of-line (EOL) characters.)

3.1.2 Reading ASCII-data into Python

In Python, I *strongly* suggest that you start out reading in and inspecting your data in the *Ipython qtconsole*. It allows you to move around much more easily, try things out, and quickly get feedback on how successful your commands were. When you have your command syntax worked out, you can obtain the command history with `%history`, copy it into your IDE, and turn it into a program.

While the a numpy command `loadtxt` allows you to read in simply formatted text data, most of the time, I go straight to *pandas*, as it provides significantly more powerful tools for data-entry. A typical workflow can contain the following lines:

```
import pandas as pd

cd 'C:\Data\storage'
pwd     # Check if you were successful: this is not always the case!
ls      # List the files in that directory
inFile = 'data.txt'
df = pd.read_csv(inFile)
df.head()   # Check if the first line was read in properly, and compare it to
    the values in the ASCII-file

'''
Here I typically have to play around with the options of pd.read_csv, to get
    things right.
Make sure you check that the number of column headers is equal to the number of
    columns that you expect. It can happen that everything gets red in - but into
     one large single column!
'''

df.tail()   # Check the last line, and compare it to the values in the ASCII-
    file
```

3.1.3 Regular Expressions

Working with text data often requires the use of simple *regular expressions*. *Regular expressions* are a very powerful way of finding and/or manipulating text strings. Many books have been written about them.

- https://www.debuggex.com/cheatsheet/regex/python provides a convenient cheat sheet for regular expressions in Python
- http://www.regular-expressions.info gives and a comprehensive description

Let me give you two useful examples:

1. `df = pd.read_csv(inFile, sep='[ ;,]*'`

   would read in data that are separated by a *combination* ("[...]") of *one or more* ("*") whitespaces, semicolons, and/or commas.

2. `vel = df.filter(regex='Vel*')`

   would extract all the columns that start with *'Vel'*, if your DataFrame *df* contains for example the columns *['Time', 'PosX', 'PosY', 'PosZ', 'VelX', 'VelY', 'VelZ']*.

3.2 Input from MS Excel

Pandas also offers tools for reading and writing data for MS Excel, in-memory data structures and SQL databases.

[1]There are two approaches to reading an excel file. The `read_excel` function and the `ExcelFile` class. `read_excel` is for reading one file with file-specific arguments (ie. identical data formats across sheets). `ExcelFile` is for reading one file with sheet-specific arguments (ie. various data formats across sheets). Choosing the approach is largely a question of code readability and execution speed.

Equivalent class and function approaches to read a single sheet:

[1]The following section has been taken from the *pandas* documentation.

```
# using the ExcelFile class
xls = pd.ExcelFile('path_to_file.xls')
data = xls.parse('Sheet1', index_col=None, na_values=['NA'])

# using the read_excel function
data = read_excel('path_to_file.xls', 'Sheet1', index_col=None, na_values=['
    NA'])
```

If this fails, give it a try with the Python package *xlrd*.

Example:  **Code:** "readZip.py" (p 171) This advanced script shows you how you can directly import data from an MS-Excel file from a zipped archive on the web.

3.3 Input from Matlab and other formats

Matlab Matlab support is built into *scipy*, with the command `scipy.io.loadmat`.

Clipboard If you have data in your clipboard, you can import them directly with `pd.read_clipboard()`

Other Fileformats pandas supports input from a number of additional formats. The simplest way to access them is typing `pd.read_ + TAB`, which shows you the currently available options.

Chapter 4

Basic Principles

4.1 Datatypes

The choice of appropriate statistical procedure depends on the data. If the variables are numeric, we are led to a certain statistical strategy. In contrast, if the variables represent qualitative categorizations, then we follow a different path.

The data can have one of the following datatypes:

4.1.1 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*

4.1.2 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.

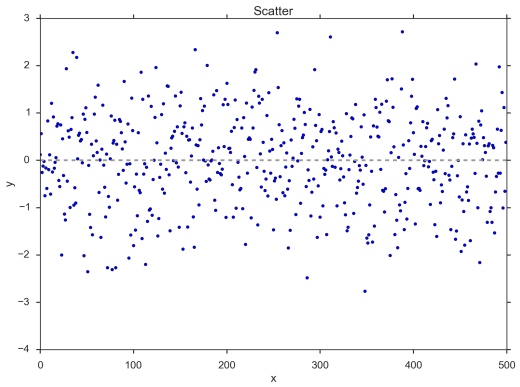


Figure 4.1: Scatter plot

4.2 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 174). 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.

4.2.1 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.)

4.2.2 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 Estimation (KDE) (kde-plots). Those are nicely implemented and described in *seaborn*.

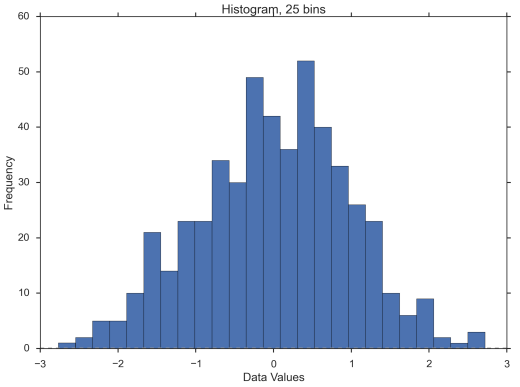


Figure 4.2: Histogram

4.2.3 KDE-plots

Histograms have the disadvantage that they are discontinuous, and that their shape critically depends on the chosen bin-width. In order to obtain smooth *probability densities*, i.e. curves describing the likelihood of finding an event in any given interval, the technique of *Kernel Density Estimation (KDE)* can be used. Thereby a normal distribution is typically used for the kernel. The width of this kernel function determines the amount of smoothing. To see how this works, we compare the construction of histogram and kernel density estimators, using these 6 data points: x = [2.1, 1.3, 0.4, 1.9, 5.1, 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 xi. The kernels are summed to make the kernel density estimate (solid blue curve). The smoothness of the kernel density estimate is evident. Compared to the discreteness of the histogram, the kernel density estimates converge faster to the true underlying density for continuous random variables.

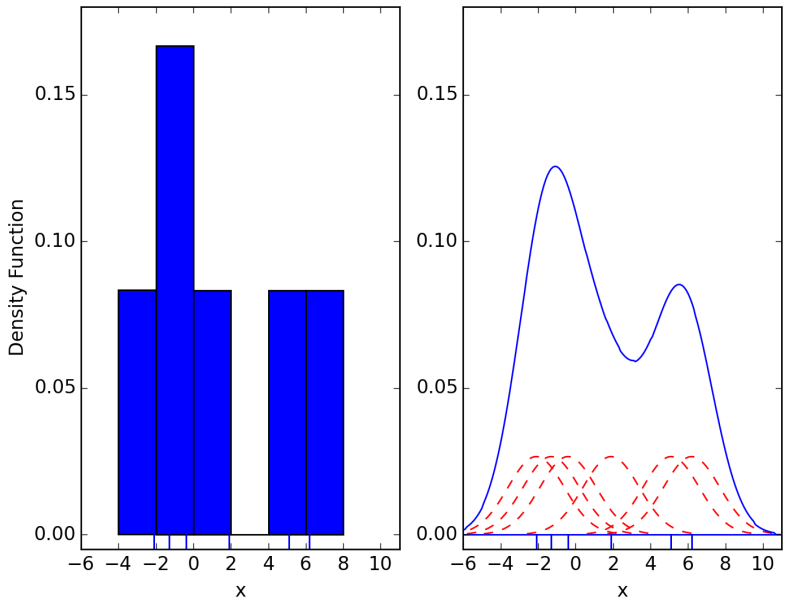


Figure 4.3: Comparison of the histogram (left) and kernel density estimate (right) constructed using the same data. The 6 individual kernels are the red dashed curves, the kernel density estimate the blue curves. The data points are the rug plot on the horizontal axis.

The bandwidth of the kernel is the parameter which determines how much we smooth out the contribution from each event. To illustrate its effect, we take a simulated random sample from the standard normal distribution, plotted as the blue spikes in the rug plot on the horizontal axis in Fig. 4.4, left. The right plot shows the true density (blue. a normal density with mean 0 and variance 1). In comparison, the gray dashed curve is undersmoothed since it contains too many spurious data artifacts arising from using a bandwidth h = 0.1 which is too small. The green dashed curve is oversmoothed since using the bandwidth h = 1 obscures much of the underlying structure. The red curve with a bandwidth of h = 0.42 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

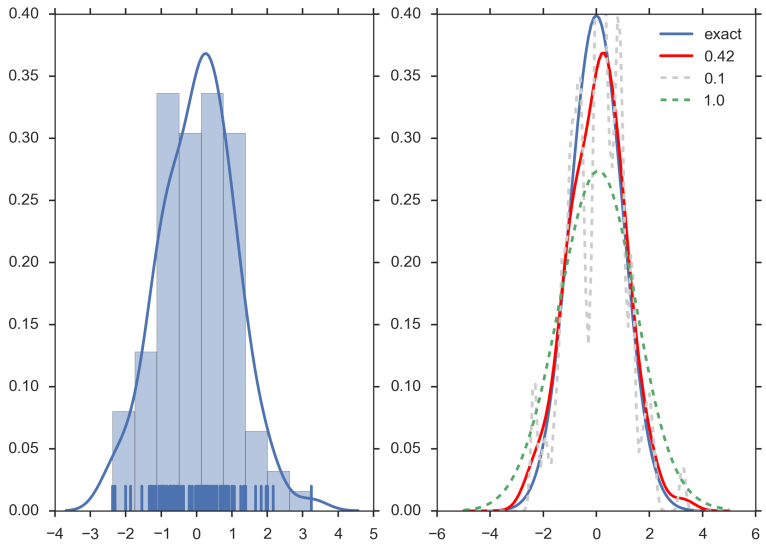


Figure 4.4: Left: Rug plot, histogram, and Kernel density estimate (KDE) of a random sample of 100 points from a standard normal distribution. Right: True density distribution (blue), and KDE with different bandwidths. Gray dashed: KDE with h=0.1; red: KDE with h=0.42 green dashed: KDE with h=1.0.

$$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \approx 1.06\hat{\sigma}n^{-1/5}, \tag{4.1}$$

where $\hat{\sigma}$ is the standard deviation of the samples ("Silverman's rule of thumb").

4.2.4 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:

$$0 \leq x \leq 1$$

4.2.5 Errorbars

Errorbars are a common way to show mean value and variability when comparing a few measurement values. Note that you have to state explicitly if your errorbars correspond to the *standard devation* or to the *standard error* of the data. Using *standard errors* has a nice feature: When error bars for the *standard error* for two groups overlap, you can be sure the difference between the two means is not statistically significant (P¿0.05). Watch out, though, since the opposite is not always true!

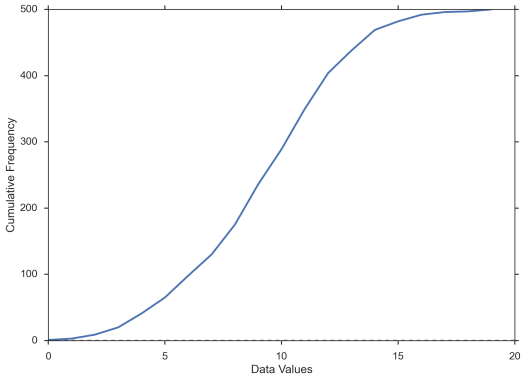


Figure 4.5: Cumulative frequency function for a normal distribution.

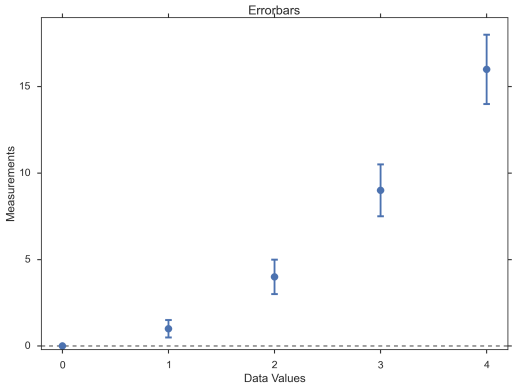


Figure 4.6: Errorbars

4.2.6 Box Plots

Box plots are frequently used in scientific publications to indicate values in two or more groups. The bottom and top of the box indicate the first and third quartiles, and the line inside the box shows the median. Care has to be taken with the whiskers, as different conventions exist for them. The most common form is that the lower whisker indicates the lowest value still within 1.5 *inter-quartile-range* (IQR) of the lower quartile, and the upper whisker the highest value still within 1.5 IQR of the upper quartile. Outliers (outside the whiskers) are plotted separately. Another convention is to have the whiskers indicate the full data range.

There are a number of tests to check for outliers. The method suggested by Tukey is to check for data which lie more than 1.5 * IQR above or below the first/third quartile (see Section 5.1.3).

Boxplots are often combined with KDE-plots to produce so-called *violin-plots* , as shown in Figure 4.8.

4.2.7 Programs: Data Display

Code: "figsBasicPrinciples.py" (p 174) shows how the plots in this section have been generated.

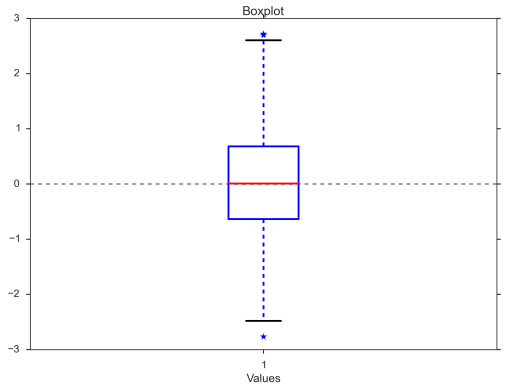


Figure 4.7: Box plot

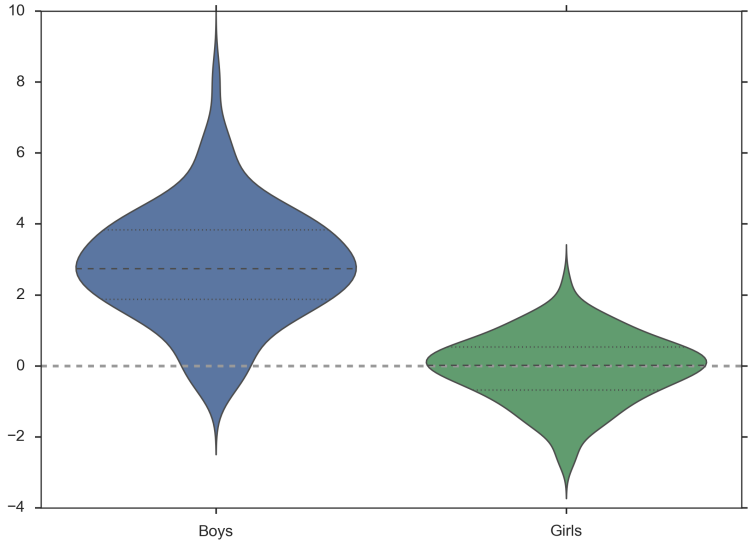


Figure 4.8: Violinplot, produced with *seaborn*.

4.3 Populations and Samples

The main difference between a *population* and a *sample* has to do with how observations are assigned to the data set (see Fig. 4.9).

Population includes all of the elements from a set of data.

Sample consists of one or more observations from the population.

More than one sample can be derived from the same population.

When estimating a *parameter* of a *population*, e.g. the weight of male Europeans, we can typically never measure all of them. We have to confine ourselves to investigate a (hopefully representative) random *sample* of this group. Based on the *sample statistic*, we use *statistical inference* to find out what we know about the *parameter*.

Parameter characteristic of a population, such as a mean or standard deviation. Often denoted with Greek symbols.

Statistic a measurable characteristic of a sample.

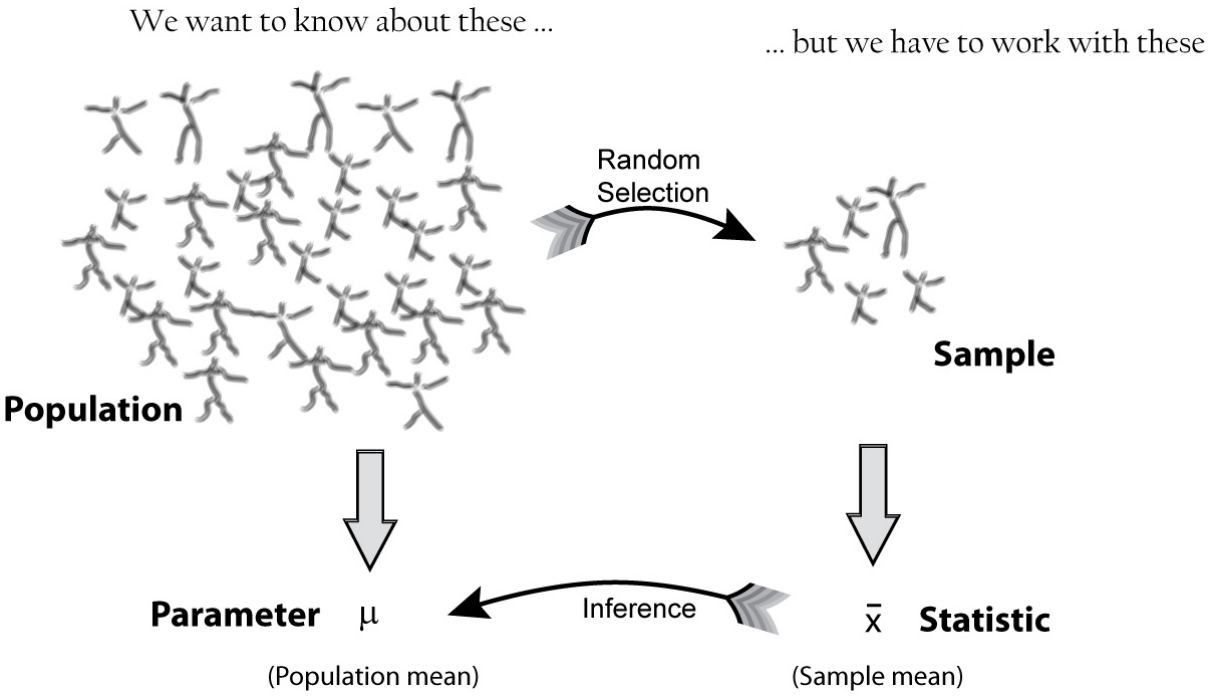


Figure 4.9: With statistical inference information from Samples is used to estimate Parameters from Populations.

Sampling distribution is the probability distribution of a given statistic based on a random sample.

Statistical inference enables you to make an educated guess about a population parameter based on a statistic computed from a sample randomly drawn from that population.

An example for parameters and statistics is given in Table 4.1.

	Sample Statistic	Population Parameter
Mean	$\bar{x}$	μ
Standard Deviation	s	σ

Table 4.1: Comparison of Sample Statistics and Population Parameters.

4.4 Degrees of Freedom

The concept of Degrees of Freedom (DOF) (DOF), which in mechanics seems to be crystal clear, is harder to grasp for statistical applications.

In mechanics, a particle which moves in a plane has *2 DOF*: at each point in time, two parameters (the x/y-coordinates) define the location of the particle. If the particle moves about in space, it is said to have *3 DOF*: the x/y/z-coordinates.

In statistics, a group of n values has n DOF. If we only look at the shape of the distribution of the values, we can subtract from each value the sample mean. Then, the remaining data only have *n-1* DF. (This is clearest for $n = 2$: if you know the mean value and the value of sample 1, then you can calculate the value of sample 2 by $val_2 = 2 * mean - val_1$.

The case becomes more complex if we have many groups. For example, in chapter 7.3.1, we describe an example with 22 patients, divided into 3 groups. In the *analysis of variance (ANOVA)*, these are divided as follows:

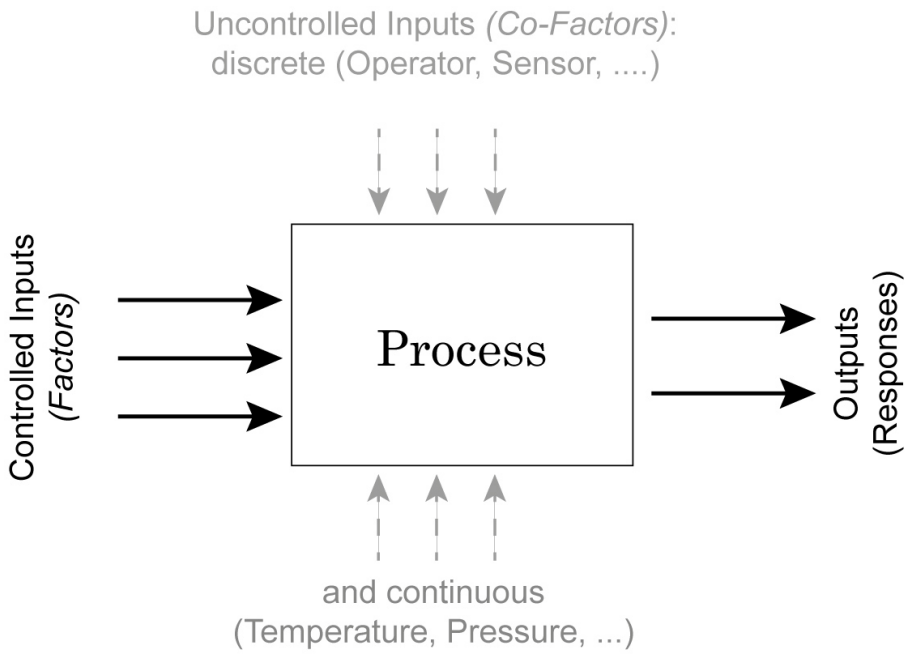


Figure 4.10: Process Schematic

- 1 DOF for the total mean value.
- 2 DOF for the mean value of each of the three groups (remember, if we know the mean values of 2 groups *and* the total mean, we can calculate the mean value of the third group.
- 19 DOF are left for the residuals.

4.5 Study Design

4.5.1 Terminology

In the context of study design, different terminologies can be found (Fig. 4.10):

- The controlled inputs are often called *factors* or *treatments*.
- The uncontrolled inputs are called *co-factors* or *nuisance factors*.

When we try to model a process with two inputs and one outputs, we can formulate a mathematical model for example as

$$X = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2 + \epsilon \tag{4.2}$$

The terms with the single "X's" are called *main effects*, and the terms with multiple "X's" *interaction terms*. And since the β parameters enter the equation only linearly, this is referred to as a *general linear model*. The ϵs are called *residuals*, and are expected to be approximately normally distributed around zero.

4.5.2 Overview

The first question you have to ask yourself is what you want to do. Do you want to

- Compare two or more groups, or one group to a fixed value?

- Screen the observed responses to identify factors/effects that are important?
- Maximize or minimize a response (variability, distanct to target, robustness)?, or
- Develop a regression model to quantify the dependence of a response variable on the process input?

The first question leads to an hypothesis test. The second one is a screening investigation, where you have to watch out for *multiple testing artefacts*. The third task is an optimization problem. And the last one brings you into the realm of *statistical modeling*.

Once you have determined *what* you want to do, you have to decide *how* you want to do this. You can either do *controlled experiments*. Or you can use *observations* to obtain the data that you are looking for. In a controlled experiment you typically try to vary only a single parameter, and investigate the effects of that parameter on the output.

4.5.3 Personal Tips

1. Be realistic about your task.
2. Plan in sufficient control/calibration experiments.
3. Take notes.
4. Store your data in a well structured way.

Preliminary Investigations and Murphy's Law

Most investigations require more than one round of experiments and analyses. The theory goes that you first state your hypothesis, then do the experiments, and accept or reject the hypothesis. Done. Most of my investigations have been less straightforward. Typically, you start out with an idea. After making sure that nobody else has solved your question before, you sit down and do the first rounds of measurements, and write the analysis programs required to analyze your data. Thereby you find most of the things that can go wrong (they typically do, as indicated by *Murphy's Law*), and what you should have done differently in the first place. Not only does that first round of investigation provide a *proof of principle* that your question is tractable, but you also have obtained some data on the variability of typical responses. This allows you to obtain a reasonable estimate of the number of subjects/samples you need in order to accept or reject your hypothesis. By this time you also know if your experimental setup is sufficient, or if you need a higher quality measurement equipment. The second round of investigations is in most cases the real stuff, and (hopefully) provides you with enough data to publish your findings.

Calibration Runs

If anyhow possible, you should start out and end your recordings with something that you know. For example during movement recordings I try to start out with recording a stationary point, and then move it 10 cm forward, left, and up. Having a recording where you know exactly what happens not only helps to detect drift in the sensors, and problems in the experimental setup. These recordings also help to verify the accuracy of your analysis programs.

Documentation

Make sure that you document all the factors that may influence your results, and everything that happens during the experiment:

- 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.

Be as brief as possible, but take down everything note-worthy that happens during your experiment. Be especially clear about the names of the recorded data-files, as this will be the first thing you need when you later analyze the data. Often you won't need all the details. But when you have outliers, weird data, etc., these notes can be invaluable for your later data analysis.

Data Storage

Try to have clear, intuitive, and practical naming conventions. For example, when you perform experiments with patients and with normals on different days, you could name these recordings "[p/n][yyyy/mm/dd]_[x].dat", e.g. *n20150329_a*. With this convention you have a natural grouping of normals and patients, and your data are automatically sorted logically by their date.

Always immediately store the rawdata, preferably in a separate directory. I prefer to make this directory read-only, so that I don't inadvertently delete valuable raw-data. You can in most cases easily redo an analysis run. But you often won't be able to repeat an experiment.

4.5.4 Types of Studies

Observational or experimental

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

Prospective or retrospective

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

Longitudinal or cross-sectional

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

Case control and Cohort studies

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

4.5.5 Design of Experiments

"Block whatever you can; and randomize the rest!"

I have mentioned above that you have factors (which you control) and nuisance factors, which influence your results, but which you cannot control and/or manipulate. Assume, for example, that you have an experiment where the result depends on the person who performs the experiment (e.g. the nurse who tests the subject), and on the time of the day. In that case you can *block* the factor *nurse*, by having all tests performed by the same nurse. But it won't be possible to test all subjects at the same time. So you try to average out time effects, by *randomly* mixing the timing of your subjects. If, in contrast, you measure your patients in the morning and your healthy subjects in the afternoon, you will invariably bring some *bias* into your data.

Bias

To explain the effects of selection bias on a statistical analysis, consider the 1936 presidential elections in the USA. The Republican A. Landon challenged the incumbent president, F.D. Roosevelt. The *Literary Digest*, at the time one of the most respected magazines, asked 10 million Americans about whom they would vote for. 2.4 million responded, and the Literary Digest predicted Landon would win 57

What went wrong?

First, the sample was badly chosen, and not representative for the American voter: the the mailing list for the survey were taken from telephone directories, club membership lists, and lists of magazine subscribers. Thus, they were strongly biased towards the American middle- and upper-class. And second, only about one fourth of the people asked responded. And people who respond to surveys are different from people who don't, the so-called *nonresponse bias*.

This example shows that a large sample size alone does not guarantee a representative response. You have to watch out for selection bias and nonresponse bias.

In general, when selecting our subject you try to make them representative of the group 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*. In a designed experiment, there may be several conditions, called factors, that are controlled by the experimenter. By having the groups differ in only one aspect, 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. (The subject "crosses over" from one treatment to the next.) To avoid causal effects, the sequence of the treatment allocation should be randomized.

Blinding

Consciously or not, the experimenter can significantly influence the outcome of an experiment. For example, a young researcher with a new "brilliant" idea for a new treatment will be bias in the execution of the experiment, as well in the analysis of the data, to see his hypothesis confirmed. To avoid such a subjective influence, ideally the experimenter as well as the subject should be blinded to the therapy. This is referred to as *double blinding*. If also the person who does the analysis does not know which group the subject has been allocated to, we speak about *triple blinding*.

Sample selection

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

1. Make sure that the samples are representative of the group you want to study.
2. In comparative studies, care is needed in making groups similar with respect to known sources of variation.
3. **Important:** Make sure that your selection of samples/subject sufficiently covers all parameters that you need!

Ad 1) For example, if you select your subjects randomly from patients at a hospital, you automatically bias your sample towards subjects with health problems.

Ad 3) For example, if you test the efficacy of a new rehabilitation therapy for stroke patients, do *not* just select patients who have had a stroke: make sure that your patient selection includes even numbers of patients with mild, medium, and severe symptoms. Otherwise you may end up with data who primarily include patients with little or no aftereffects of the stroke. (I hate to admit that this type of mistake has repeatedly happened to me, and cost me many months of work!)

Many surveys and studies fall short on these criteria (see section on *Bias* above). The field of "matching by propensity scores" ([Rosenbaum and Rubin(1983)]) attempts to correct these problems.

Sample size

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

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

This is known as power analysis. it is especially important in behavioral research, where research plans are not approved without careful sample size calculations.

4.5.6 Structure of Experiments

If each combination of factors is tested, we talk about a *full factorial design* of the experiment.

In planning the analysis, you have to keep the important distinction between *within subject* comparisons, and *between subjects* comparisons. The former one, *within subject comparisons*, allows you to detect smaller differences with the same number of subjects than *between subject comparisons*.

4.5.7 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.

4.6 Exercises

1. (a) Read in the data from 'Data\amstat\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 5

Distributions of one Variable

5.1 Characterizing a Distribution

5.1.1 Continuous Distribution Functions

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

Probability Density Function (PDF)

The Probability Density Function (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:

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

The PDF also defines the *expected value E[x]* of a continuous distribution of X:

$$\mathrm{E}[X] = \int_{-\infty}^{\infty} x f(x) \, \mathrm{d}x. \tag{5.1}$$

Cumulative Distribution 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. 5.1), and the Cumulative Distribution Function (CDF) specifies which percentage of the data below any given value (Figure 5.2). Together, this gives us

$$\mathbb{P}(a \leq X \leq b) = \int_{a}^{b} PDF(x)dx = CDF(b) - CDF(a) \tag{5.2}$$

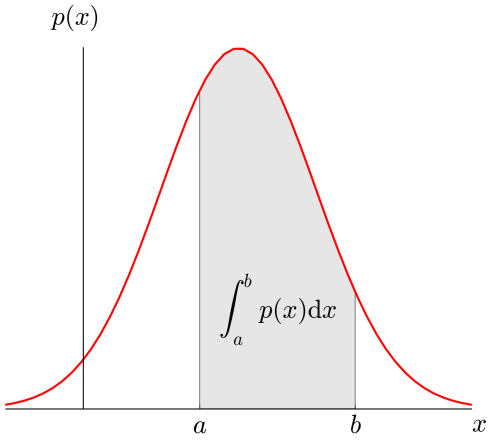


Figure 5.1: Probability Density Function (PDF) of a value x. The integral over the PDF between a and b gives the likelihood of finding the value of x in that range.

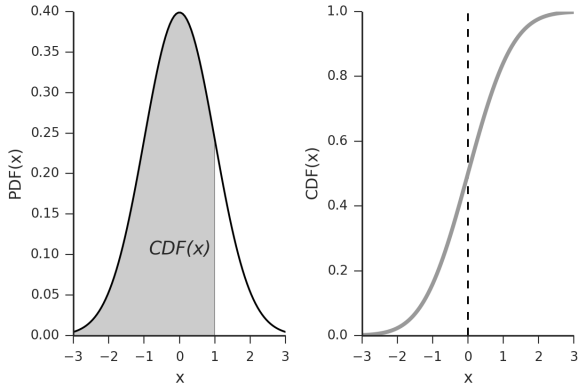


Figure 5.2: *Probability density function* (left) and *Cumulative distribution function* (right) of a normal distribution.

Other important presentations of Probability Densities

Figure 5.3 shows a number of functions that are equivalent to the PDF, but each represent a different aspect of the probability distribution.

I will give examples which demonstrate each aspect for a normal distribution describing the size of male subjects. To obtain the respective values, one has to multiply the PDF-function etc. with the observed standard distribution, and add the mean value:

- *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.

  Example: What is the chance that a man is between 160 and 165 cm tall?

- *Cumulative distribution function (CDF)*: gives the probability of obtaining a value smaller than the given value.

  Example: What is the chance that a man is less than 165 cm tall?

- *Survival Function (SF) (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.

  Example: What is the chance that a man is larger than 165 cm?

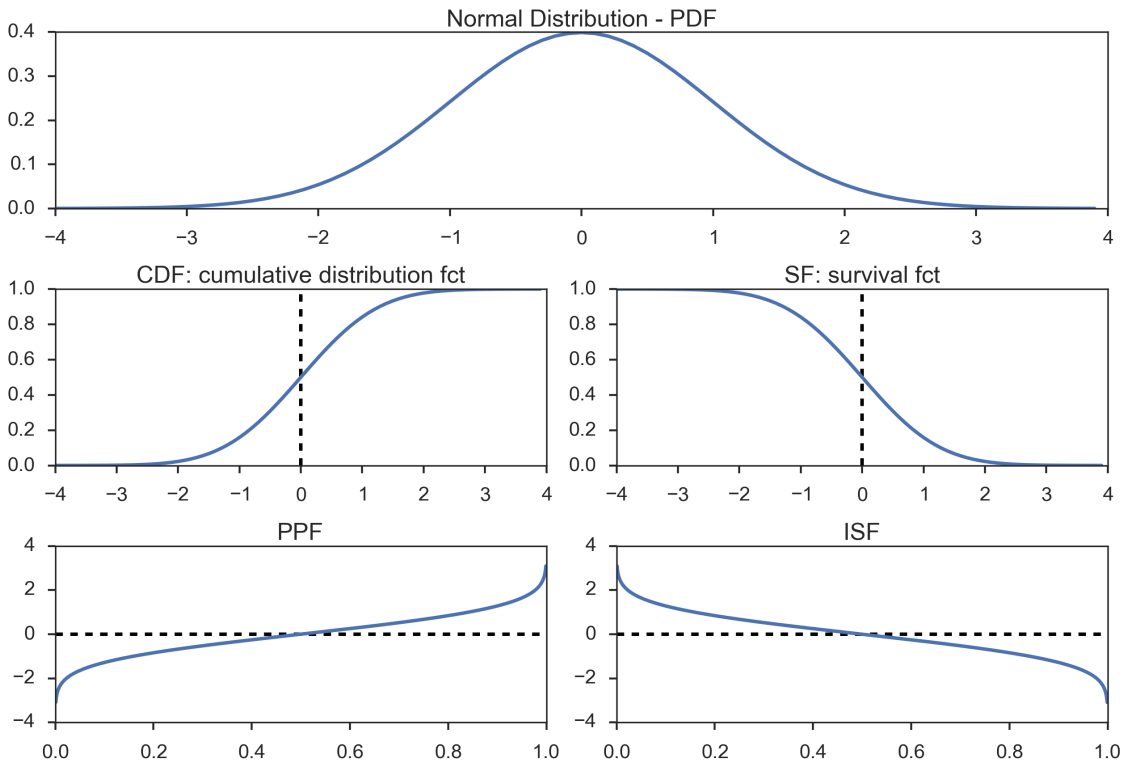


Figure 5.3: Utility functions for continuous distributions, here for the normal distribution.

- *Percentile Point Function (PPF) (PPF)*: the inverse of the CDF. Answers the question "Given a certain probability, what is the corresponding value for the CDF?"

  Example: Given that I am looking for a man who is smaller than 95% of all other men, what size does the subject have to be?

- *Inverse Survival Function (ISF) (ISF)*: the name says it all.

  Example: Given that I am looking for a man who is larger than 95% of all other men, what size does the subject have to be?

- Random Variate Sample (RVS) *(RVS)*: random variates from a given distribution. (A *variable* is the general type, a *variate* is a specific number.)

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)`)

5.1.2 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* $\bar{x}$:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.3}$$

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.

5.1.3 Quantifying Variability

Range

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

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 2.5^{th} and the 97.5^{th} percentile of your sample distribution.
- The 50th percentile is the *median*.
- Also important are the *quartiles*, i.e. the 25th and the 75th percentile. The difference between them is sometimes referred to as *inter-quartile range (IQR)*.

Median, upper and lower quartile are used for the data display in box plots (Fig.4.7).

Standard Deviation and Variance

In Fig. 4.9 I sketched out how we have to use the *sample statistic* to learn about the corresponding *population parameter*. The *maximum likelihood estimator of the sample variance* is given by

$$var = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} \tag{5.4}$$

The best *estimate of the population variance* is given by However, this systematically underestimates the *population variance*, and is therefore referred to as a *biased estimator*.

Figure 5.4 indicates that the sample variance underestimates the population variance, because the sample mean is chosen such as to minimize the variance of the data.

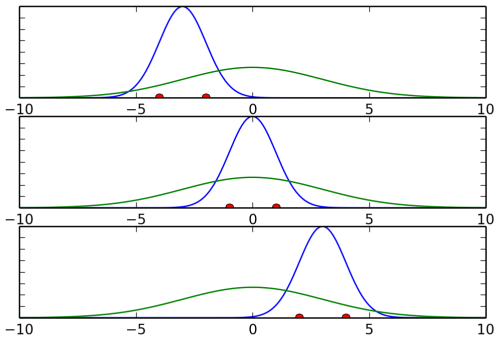


Figure 5.4: Gaussian distributions fitted to selections of data from the underlying distribution: While the average mean of a number of samples converges to the real mean, the sample standard deviation underestimates the standard deviation from the distribution.

Division by $n - 1$ corrects for this bias, and provides the best possible estimate of the population variance. The variance calculated with $n - 1$ is called the *sample variance*.

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

$$s = \sqrt{var} \tag{5.5}$$

As mentioned in Table 4.1, in statistics it is often common to denote the population standard deviation with σ, and the sample standard deviation with s.

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

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

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

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

Standard Error

The *standard error* is the estimate of the *standard deviation of a coefficient*. For example, in Fig. 5.5, we have used 100 datapoint from a normal distribution about *5*. The more datapoints we have to estimate the mean value, the better our estimate. With 100 datapoints the standard deviation of our estimate, or *standard error of the mean*, is 10 times smaller than the sample standard deviation.

For normally distributed data, the *sample standard error of the mean* (SE or SEM) is

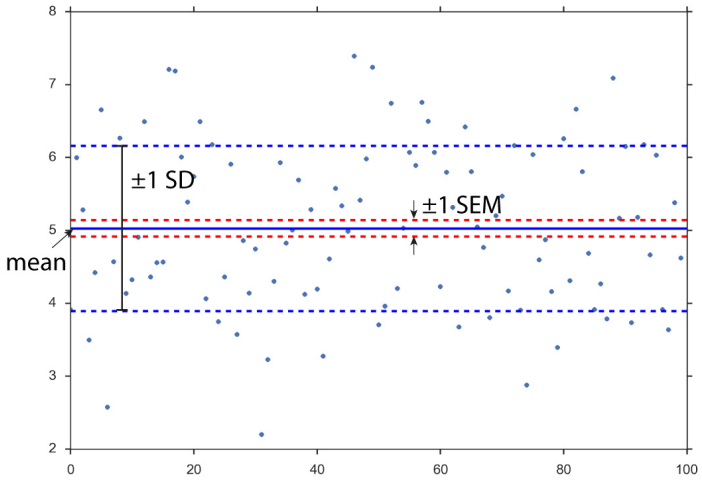


Figure 5.5: 100 random data points, from a normal distribution about 5. The sample mean is very close to the real mean. The standard deviation of the mean, or *standard error* is 10 times smaller than the standard deviation of the samples.

$$SE = \frac{s}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \cdot \frac{1}{\sqrt{n}} \tag{5.6}$$

Confidence Intervals

In the statistical analysis of data it has become common to state the Confidence Interval (CI) of an estimated parameter. The confidence interval reports the range that contains the true value for your parameter with a likelihood of $\alpha\%$.

If the sampling distribution is symmetrical and unimodal, it will often be possible to approximate the confidence interval by

$$ci = mean \pm std * N_{ISF}(\alpha) \tag{5.7}$$

where *std* is the standard deviation, and $N_{ISF}(\alpha)$ the inverse survival function for the normal distribution. For the 95% two-sided confidence intervals, for example, you have to set $\alpha = 0.025$ and $\alpha = 0.975$.

Note: If you want to know the confidence interval for the mean value, you have to replace the *standard deviation* by the *standard error*!

5.1.4 Parameters Describing the Form of a Distribution

In `scipy.stats`, continuous distribution functions are characterized by their *location* and their *scale*. Should the definition of a distribution require more than two parameters, the following parameters are called *shape parameters*. The exact meaning of each parameter can be found in the function definition.

For example, for the *normal distribution*, *(location/shape)* are given by *(mean/standard deviation)* of the distribution. In contrast, for the *uniform distribution*, *location/shape* are given by the *(start/end)* of the range where the distribution is different from zero.

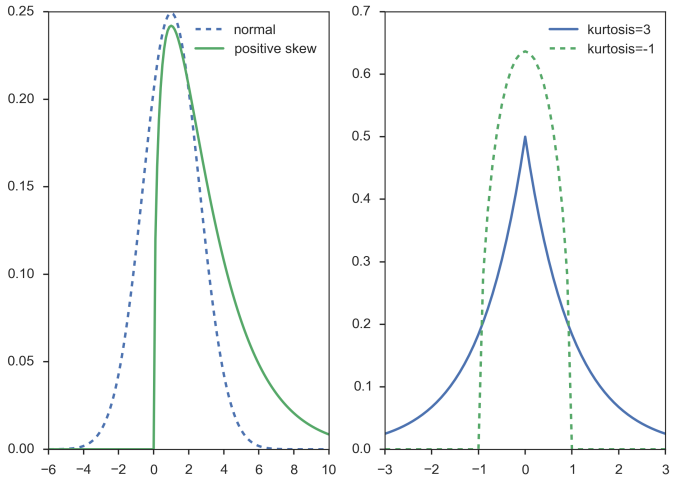


Figure 5.6: Left) Normal distribution, and distribution with positive skewness. Right) The (leptokurtic) Laplace distribution has an excess kurtosis of 3, and the (platykurtic) Wigner semicircle distribution an excess kurtosis of -1.

Location

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

$$f_{x0}(x) = f(x - x_0)$$

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.

Shape Parameters

It is customary to refer to all of the parameters beyond *location* and *scale* as *shape paraemters*. Thankfully, almost all of the distributions that we use in statistics have only one or two parameters. It follows that the *skewness* and *kurtosis* of these distribution are constants.

Skewness

Distributions are *skewed* if they depart from symmetry (Fig. 5.6, left). 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.

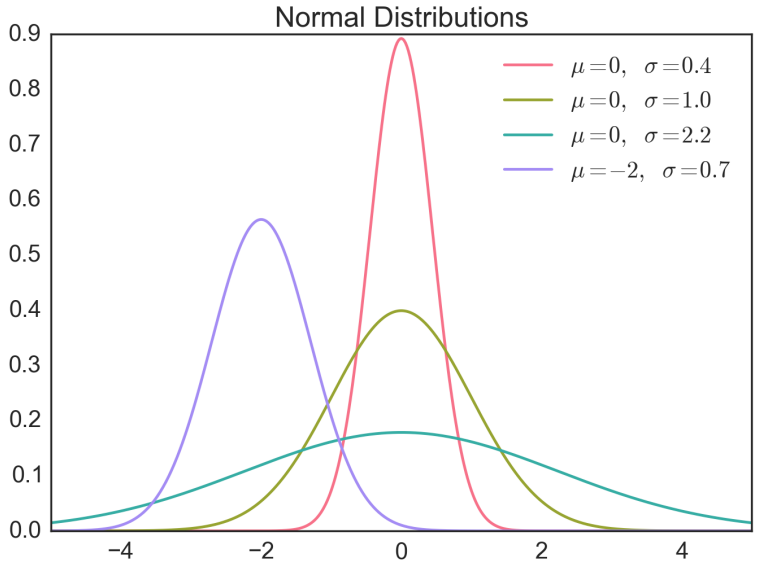


Figure 5.7: Normal Distribution

Kurtosis

Kurtosis is any measure of the "peakedness" of the probability distribution (Fig. 5.6, right), where the normal distribution is taken as the standard refernence. Distributions with negative or positive excess kurtosis are called *platykurtic* distributions or *leptokurtic* distributions, respectively.

5.2 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).

5.2.1 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 (see chapter 5.2.2). Mathematically, the normal distribution is characterized by a mean value μ, and a standard deviation σ:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{5.8}$$

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. 5.8)

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

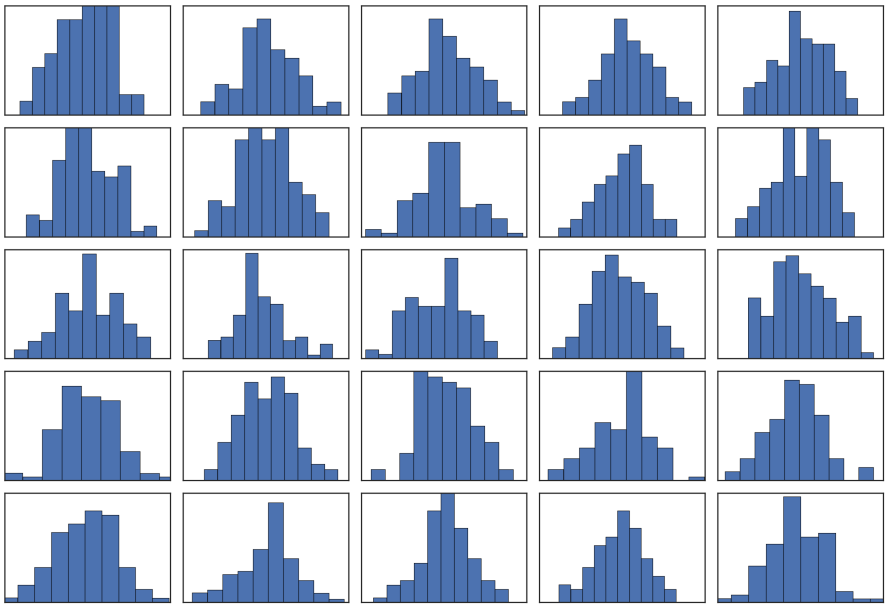


Figure 5.8: 25 randomly generated normal distributions of 100 points.

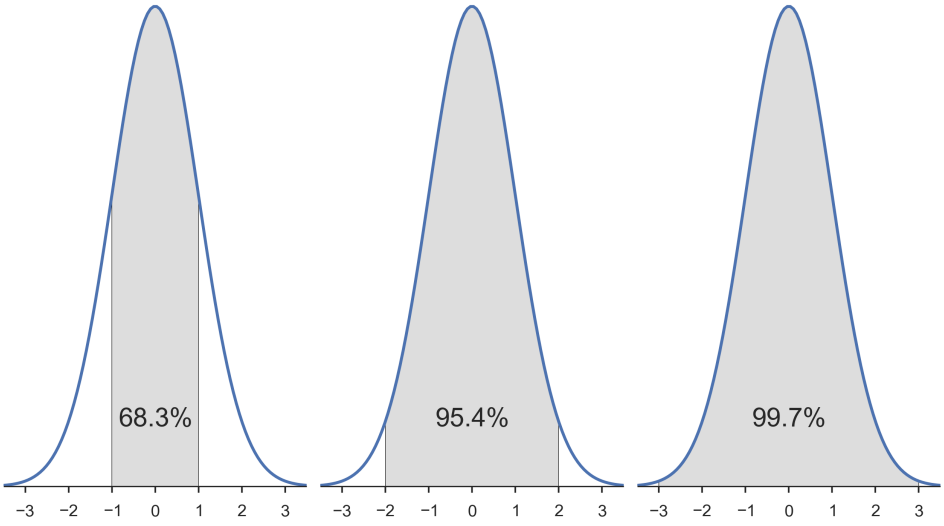


Figure 5.9: Area under +/1 1, 2, and 3 standard deviations of a normal distribution.

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

Table 5.1: Tails of a normal distribution.

 **Code:** "distributionNormal.py" (p 177) shows simple manipulations of normal distribution functions.

```
In [33]:  from scipy import stats
In [34]:  mu = -2
In [35]:  sigma = 0.7
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 blue curve in Figure 5.7

Sum of Normal Distributions

An important property of normal distributions is that the sum (or difference) of two normal distributions is also normally distributed. i.e., if

$$\begin{aligned} X &\sim N(\mu_X, \sigma_X^2) \\ Y &\sim N(\mu_Y, \sigma_Y^2) \\ Z &= X + Y, \end{aligned}$$

then

$$Z \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2). \tag{5.9}$$

Examples of Normal Distributions

- 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?
- 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?
- 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 a randomly selected man will be shorter than the randomly selected woman?

5.2.2 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. Or in other words, the sampling distribution of the mean tends toward normality, regardless of the distribution. 5.10 shows that averaging over 10 uniformly distributed data already produces a smooth, almost Gaussian distribution.

 **Code:** "centralLimitTheorem.py" (p 176) demonstrates that already averaging over 10 uniformly distributed datapoints produces an almost Gaussian distribution.

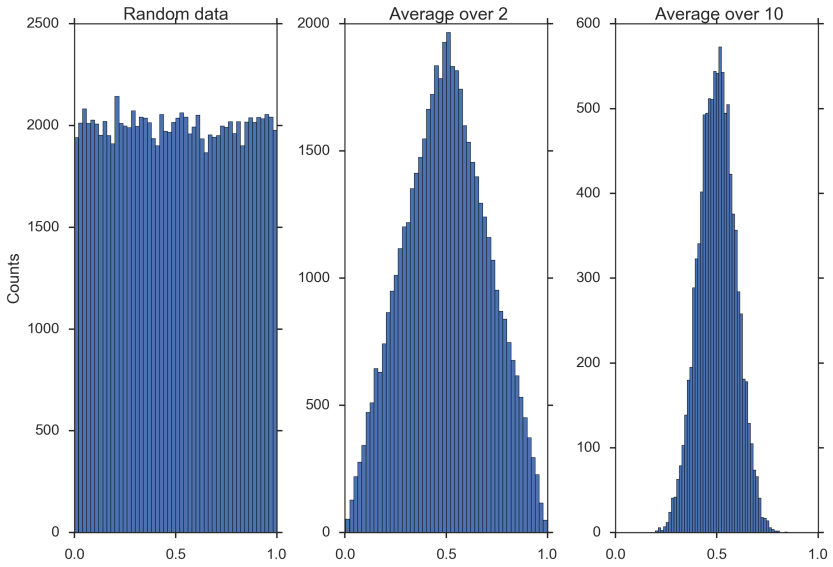


Figure 5.10: Demonstration of the "Central Limit Theorem": Left) Histogram of random data between 0 and 1. Center) Histogram of average over two datapoints.) Right) Histogram of average over 10 datapoints.

5.2.3 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 ($\mu = 3.5, \sigma = 0.76$).
- Calculate the CDF at the interesting value (*CDF(2.6 kg) = 0.118*).
- Interpret the result -see Fig. 5.11 (*"If the baby is healthy, the chance that its weight deviates by at least the observed value from the mean is 2*11.8% = 23.6% - This is not significant"*).

5.2.4 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. Other distributions you should have heard about will be mentioned briefly:

- **Lognormal distribution** - a normal distribution, plotted on an exponential scale. Often used to convert a strongly skewed distribution into a normal one.

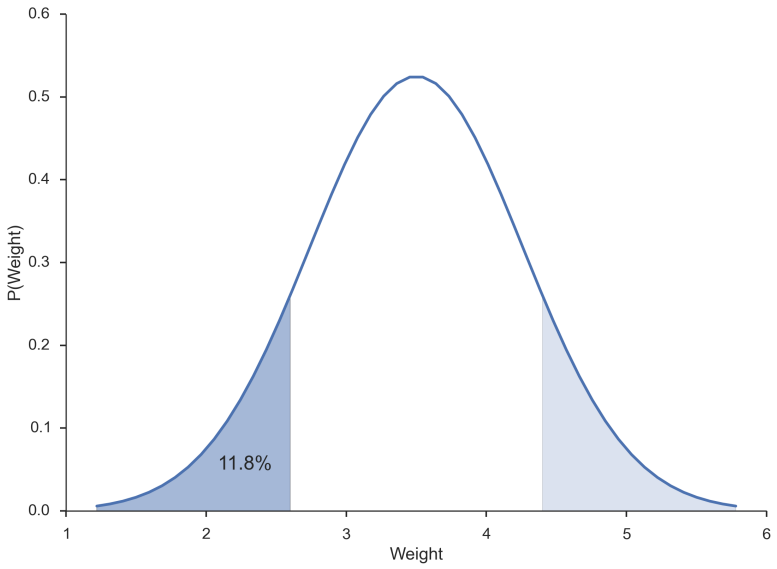


Figure 5.11: The chance that a healthy baby weighs 2.6 kg or less is 11.8% (darker blue area). The chance that the difference from the mean is that much is twice that much, as the lighter blue area must be added.

- **Weibull distribution** - mainly used for reliability or survival data.
- **Exponential distribution** - exponential curves
- **uniform distribution** - when everything is equally likely.

t Distribution

In 1908 W.S. Gosset, who worked for the Guinnes brewery in Dublin, was interested in the problems of small samples, for example the chemical properties of barley where sample sizes might be as low as 3. Since in these measurements the true variance of the mean was unknown, it must be approximated by the sample standard error of the mean. And the ratio between the sample mean and the standard error had a distribution that was unknown till Gosset, under the pseudonym "Student", solved that problem.

The corresponding distribution is the *t-distribution*, and converges for larger values towards the normal distribution (Fig. 5.12).

Since in most cases the population mean and its variance are unknown, one typically works with the t-distribution when analyzing sample data.

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

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} \tag{5.10}$$

A very frequent application of the t-distribution is in the calculation of confidence intervals for the mean. The *width* of the 95

$$ci = mean \pm std * t_{df,\alpha} \tag{5.11}$$

```
In [27]: n = 20
In [28]: df = n-1
```

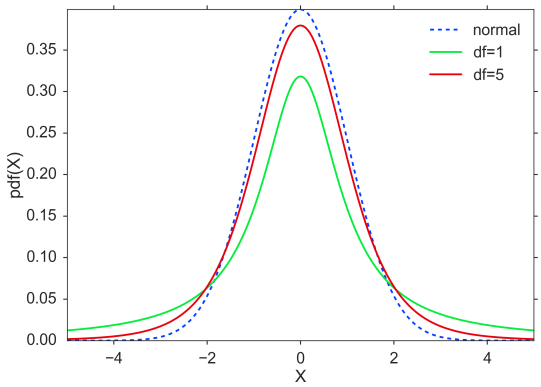


Figure 5.12: t Distribution

```
In [29]: alpha = 0.05
In [30]: stats.t(df).ppf(1-alpha/2)
Out[30]: 2.093

In [31]: stats.norm.ppf(1-alpha/2)
Out[31]: 1.960
```

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.

In Python, the 95% confidence interval for the mean can be obtained with a one-liner:

```
alpha = 0.95
df = len(data)-1
ci = stats.t.interval(alpha, df, loc=mean(data), scale=stats.sem(data))
```

Since the t-distribution has longer tails than the normal distribution, it is much less affected by extreme cases (see Figure 5.13).

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 ($X^2 \in \chi^2_1$). The *sum squares* of n independent and standard normal random variables has a chi-square distribution with n degrees of freedom (Fig. 5.14):

$$\sum_{i=1}^{n} X_i^2 \in \chi^2_n \tag{5.12}$$

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 $x_1, x_2, ..., x_n$ 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:

$$1 - CDF_{\chi^2{}_{(n-1)}} \left(\sum (\frac{x - \bar{x}}{\sigma})^2 \right) = 0.1929 \tag{5.13}$$

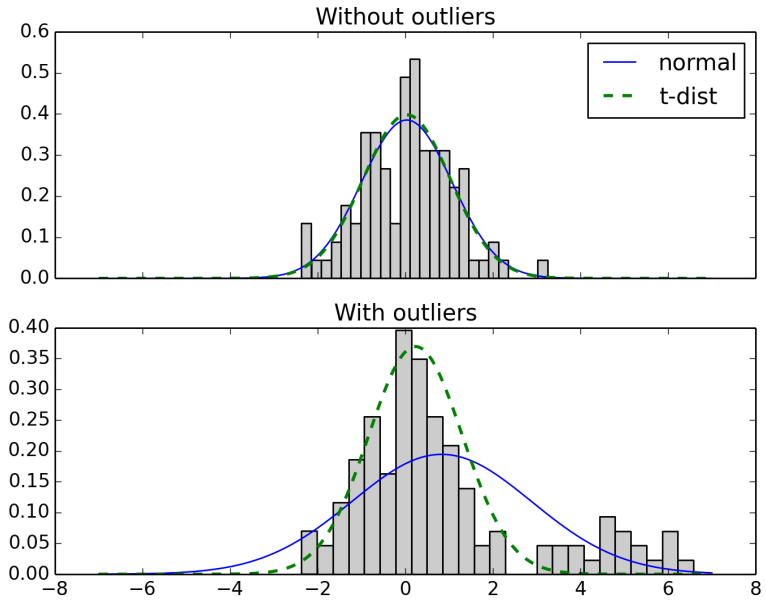


Figure 5.13: The t-distribution is much more robust against outliers than the normal distribution.

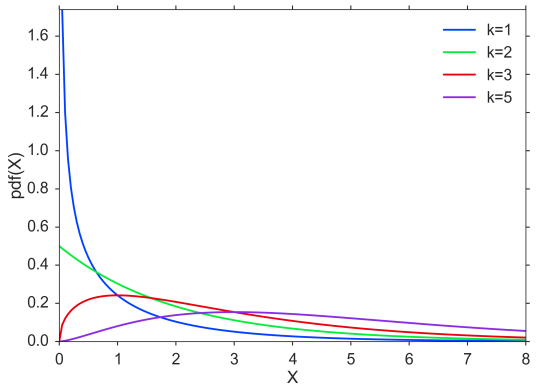


Figure 5.14: Chi-square Distribution

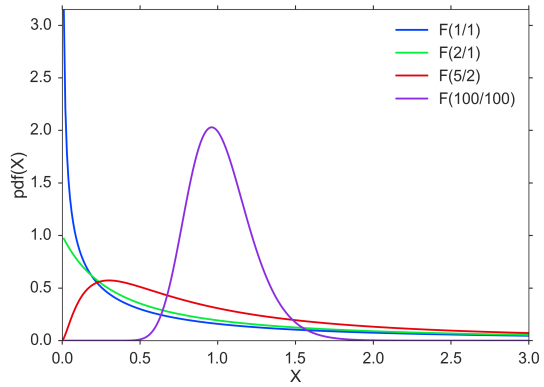


Figure 5.15: F Distribution

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 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*). Suppose we want to calculate the variance of a variable in two separate subsets. "Is one more diverse than the other?" It is tempting to take the ratio of the variances, S_1^2/S_2^2, but we don't know for sure how big or small that number must be before we conclude that the variances are different. This problem is solved by the F distribution. 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 (Fig. 5.15):

$$F(r_1, r_2) = \frac{\chi_{r1}^2/r_1}{\chi_{r2}^2/r_2} \tag{5.14}$$

where χ_{r1}^2 and χ_{r2}^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:

$$F = \frac{S_x^2}{S_y^2} \tag{5.15}$$

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

Application Example

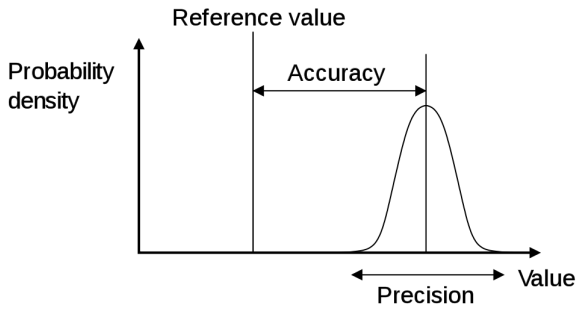


Figure 5.16: Accuracy and precision of a measurement are two different characteristics!

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 (Fig. 5.16). 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, 20.3, 19.7, 20.3] Method 2: [19.7, 19.4, 20.1, 18.6, 18.8, 20.2, 18.7, 19.]

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.'
```

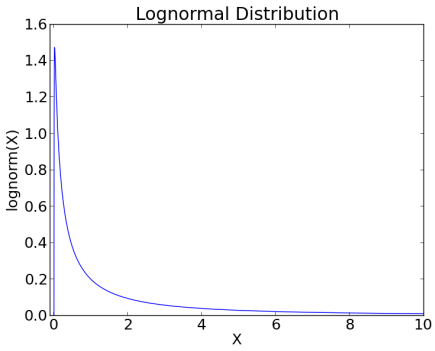
Lognormal Distribution

Normal distributions are the easiest ones to work with. In some circumstances a set of data with a positively skewed distribution can be transformed into a symmetric, normal distribution by taking logarithms. Taking logs of data with a skewed distribution will often give a distribution that is near to normal (see Figure 5.17).

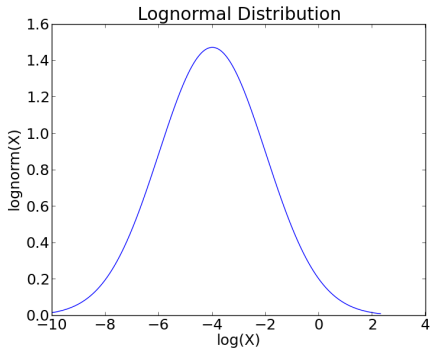
Weibull Distribution

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 5.18). It is defined as

$$f_x(x) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases} \tag{5.16}$$



(a) Plotted against a linear abscissa.



(b) Plotted against a logarithmic abscissa.

Figure 5.17: Lognormal distribution

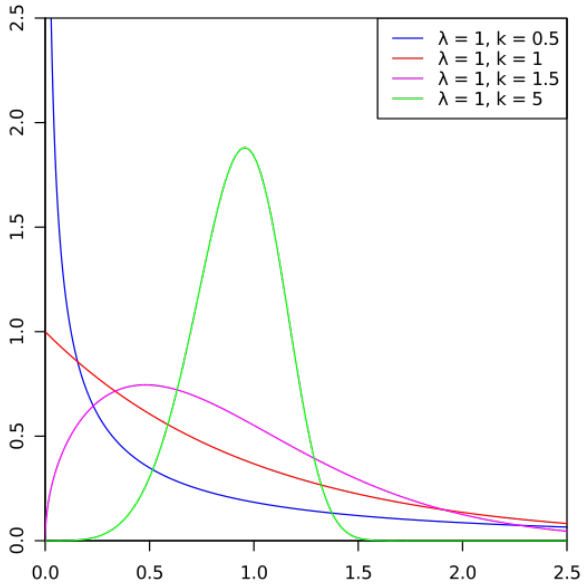


Figure 5.18: Weibull Distribution

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.

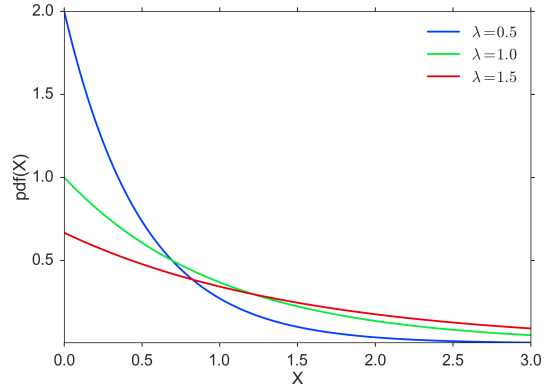


Figure 5.19: Exponential Distribution

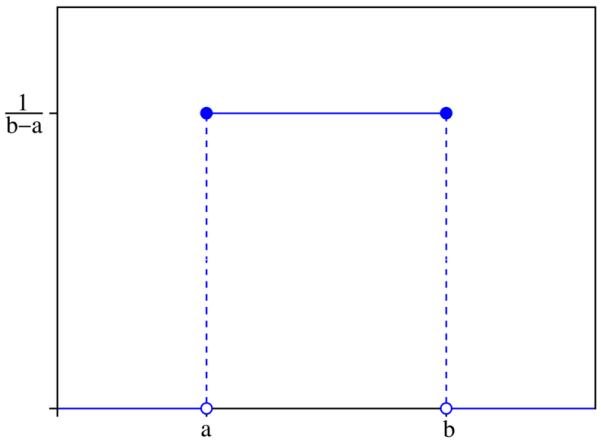


Figure 5.20: Uniform Distribution

Exponential Distribution

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

$$f_x(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{5.17}$$

The exponential PDF is shown in Figure 5.19

Uniform Distribution

This is a simple one: an even probability for all data values (Figure 5.20). 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 181) shows different continuous distribution functions.

5.2.5 Discrete Distributions

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

Two discretely distributions are commonly encountered: the *binomial distribution*, and the *Poisson distribution*. These two have the following properties:

	Mean	Variance
Binomial	$n \cdot p$	$np(1-p)$
Poisson	λ	λ

Table 5.2: Properties of discrete distributions.

The big difference between those two functions that you have to keep in mind: applications of the Binomial function have an inherent upper limit (e.g. when you throw dice five times, each side can come up a maximum of five times); in contrast, the Poisson distribution does not have an inherent upper limit (e.g. how many people you know).

Binomial Distribution

The Binomial is associated with the question "Out of a given (fixed) 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 $X \sim B(n,p)$ and the probability mass function is given at $X = k$ by the equation:

$$P[X=k] = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k} & 0 \le k \le n \\ 0 & \text{otherwise} \end{cases} \quad 0 \le p \le 1, \quad n \in \mathbb{N} \tag{5.18}$$

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

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

For n trials, we have the following properties:

- mean: np
- variance: $np(1-p)$

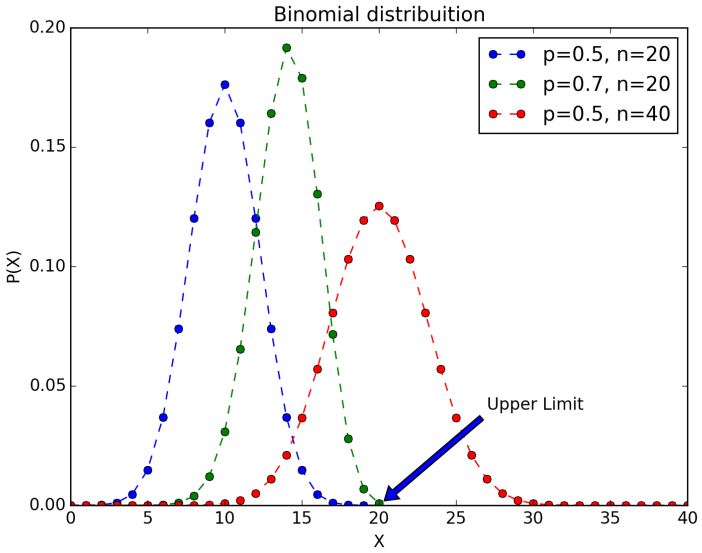


Figure 5.21: Binomial Distribution. Note that legal values exist only for integer x. The dotted lines in between only facilitate the grouping of the values to individual distribution parameters.

Example: 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 ($> 516's$) assuming that the die is fair. In this example, the result is 0.0265, 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.

Code: "binomial.py" (p 199): Example of a one-and two-sided binomial test.

Poisson Distribution

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

The Poisson Distribution is "very similar" to the Binomial Distribution. We are examining the number of times an event happens. The difference is subtle. Whereas the Binomial Distribution looks at how many times we register a success over a fixed total number of trials, the Poisson Distribution measures how many times a discrete event occurs, over a period of continuous space or time. There is no "total" value n. As with the previous sections, let's examine a couple of experiments or questions that might have an underlying Poisson nature.

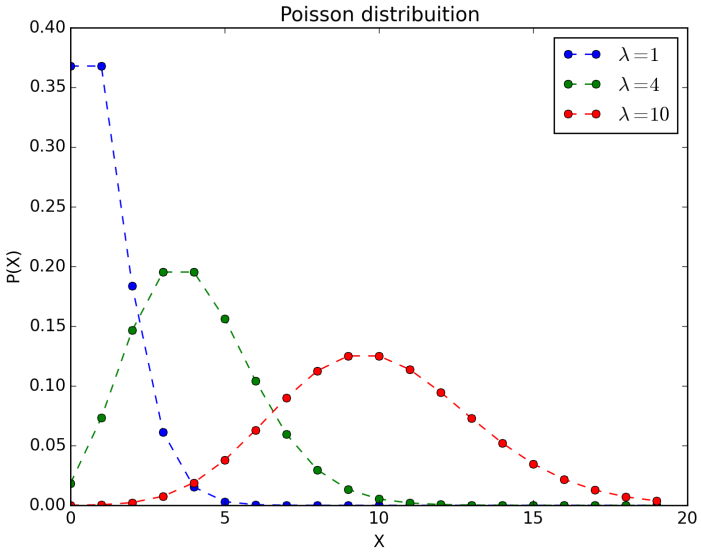


Figure 5.22: Poisson Distribution. Again note that legal values exist only for integer x. The dotted lines in between only facilitate the grouping of the values to individual distribution parameters.

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

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

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

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{5.19}$$

.

The Poisson distribution has the following properties:

- mean: λ
- variance: λ

Programs: Discrete Distribution Functions

Code: "distributionDiscrete.py" (p 183) shows different continuous distribution functions.

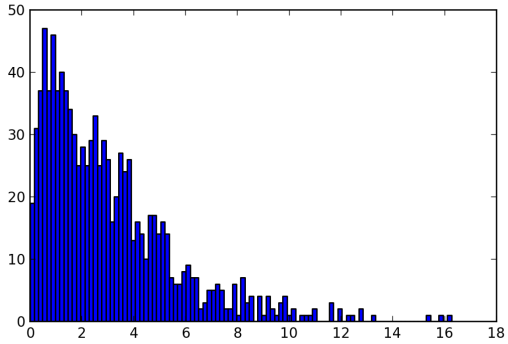


Figure 5.23: chi2 distribution with 3 degrees of freedom.

5.3 Exercises

Python

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

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])

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 = 0, SD = 1), with 1000 samples each. Then square them, sum them (so that you have 1000 datapoints), 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)

Discrete Distributions

1. Under which conditions do you use the *binomial distribution* to evaluate the likelihood of a discrete number of events? And under which do you use the *Poisson distribution*?

2. **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%)

3. **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 6

Statistical Data Analysis

6.1 Typical Analysis Procedure

In "the old days" (before computers with almost unlimited computational power were available), the statistical analysis of data was typically restricted to hypothesis tests: you formulate a hypothesis, collect your data, and then accept or reject the hypothesis. The resulting hypothesis tests form the basic framework for by far most analyses in medicine and life sciences, and the most important hypotheses tests will be described in the following chapters.

The advent of powerful computers has changed the game. Nowadays, the analysis of statistical data is (or at least should be) a highly interactive process: you look at the data, and generate models which may explain your data. Then you determine the best fit parameters for these models, and check these models, typically by looking at the residuals. If you are not happy with the results, you modify the model to improve the correspondence between models and data; when you are happy, you calculate the confidence interval for your model parameters, and form your interpretation based on these values. An introduction into this type of statistical analysis is provided in chapter 13.

In either case, you should start off with the following steps:

- Visually inspect your data.
- Find extreme samples, and check them carefully.
- Determine the data-type of your values.
- If you have continuous data, check whether or not they are normally distributed.
- Select and apply the appropriate test, or start with the model-based analysis of your data.

6.1.1 Data Screening

The first thing you have to do in your data analysis is simply *inspect your data visually.* Our visual system is enormously powerful, and if your data are properly displayed, you will often be able to see trends that may characterize the data. You should check for *missing data* in your data set, and *outliers* which can significantly influence the result of your analysis.

Outliers

There is no unique definition of *outliers.* However, for normally distributed samples they are often defined as data that lie either more than 1.5*IQR (inter-quartile range), or more than 2 standard deviations, from the sample mean. Outliers often fall in one of two groups: they are either caused by mistakes in the recording, in which case they should be excluded; or they constitute very important and valuable data points, in which case they have to be included in

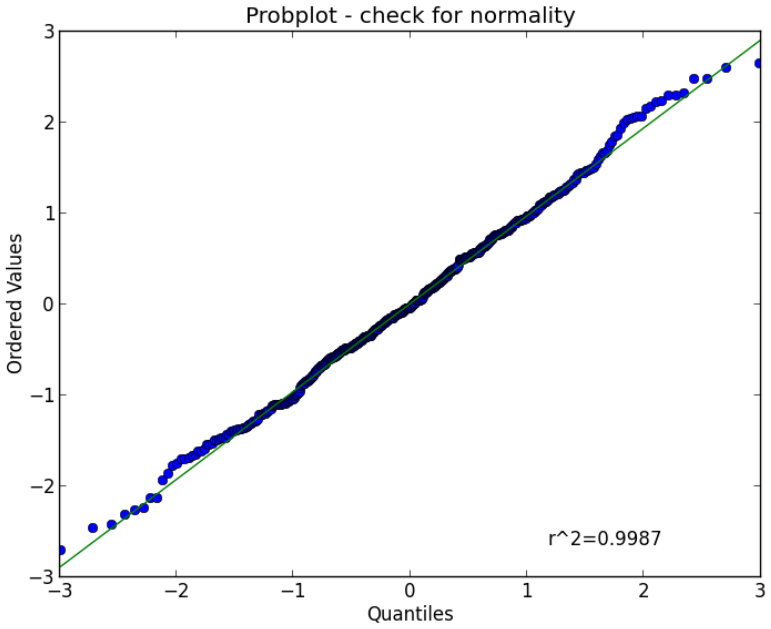


Figure 6.1: Probability-Plot, to check for normality of distribution.

the data analysis. To decide which of the two is the case, you have to check the underlying raw data (for saturation or invalid data values), and the protocols from your experiments (for mistakes that may have occurred during the recording). If you find an underlying problem, then - and only then - may you eliminate the outliers from the analysis. In every other case, you have to keep them!

6.1.2 Normality Check

Statistical hypothesis tests can be grouped into *parametric tests* and *non-parametric tests*. Parametric tests assume that the data can be well described by a distribution that is defined by one or more parameters, in most cases by a normal distribution. For the given data set, the best-fit parameters for this distribution are then determined, together with their confidence intervals, and interpreted.

However, this approach only works if the given data set is in fact well approximated by the chosen distribution. If not, the results of the parametric test can be completely wrong. In that case non-parametric tests have to be used which are less sensitive, but therefore do not depend on the data following a specific distribution.

Here we will focus on tests for normality. Again, you should start out with a visual inspection of the data, here with a *QQ-plot*, sometimes also referred to as *probplot*. For a quantitative evaluation one of the many existing normality tests should then be applied.

Probability-Plots

In statistics, different tools are available for the visual assessments of distributions. They are graphical methods for comparing two probability distributions by plotting their quantiles, or closely related parameters against each other:

QQ-Plots The "Q" in Quantile-Quantile Plot (QQ-Plot) stands for *quantile*. The quantiles of a given data set are plotted against the quantiles of a reference distribution, typically the standard normal distribution.

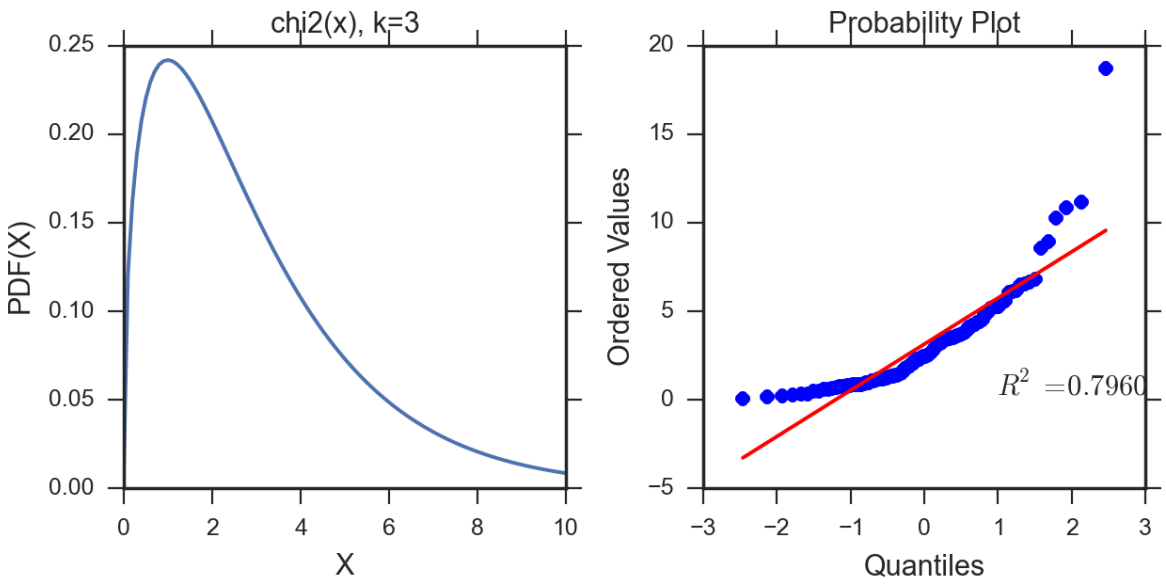


Figure 6.2: Left) Probility-density-function for a Chi2-distribution (k=3), which is clearly non-normal. Right) Corresponding Probability-plot

PP-Plots plot the *CDF (cumulative-distribution-function* of a given data set against the CDF of a reference distribution.

Probability Plots plot the ordered values of a given data set against the quantiles of a reference distribution.

In all three cases the results are similar: if the two distributions being compared are similar, the points will approximately lie on the line $y = x$. If the distributions are linearly related, the points will approximately lie on a line, but not necessarily on the line $y = x$ (Figure 6.2).

In Python, the plot can be generated with the command

```
stats.probplot(data, plot=plt)
```

To understand the principle behind those plots, look at the right plot in Fig. 6.2. Here we have *100* random datapoints from a Chi2-distribution. The x-value of the first data point is (approximately) the 1/100-quantile of a standard normal distribution (`stats.norm().ppf(0.01)`), which corresponds to *-2.33* (The exact value is slightly shifted, because of *Filliben's estimate*). The y-value is the smallest value of our data-set. Similarly, the second x-value corresponds approximately to `stats.norm().ppf(0.01)`, and the second y-value is the second-lowest value of the data set, etc.

Hypothesis Tests for Normality

In tests for normality, different challenges can arise: sometimes only few samples may be available, while other times one may have many data, but some extremely outlying values. To cope with the different situations different tests for normality have been developed. These tests to

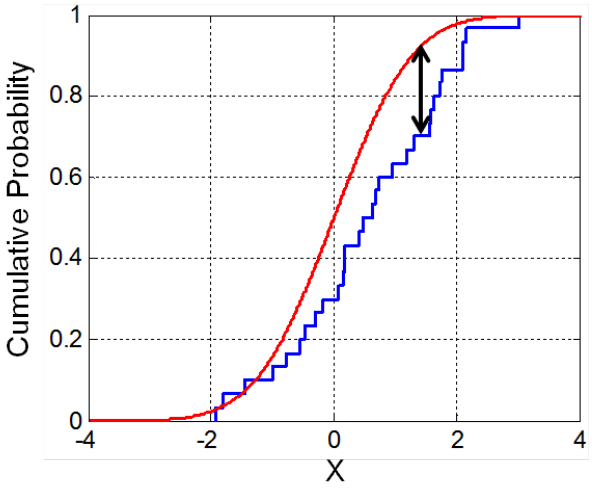


Figure 6.3: Illustration of the Kolmogorov-Smirnov statistic. Red line is CDF, blue line is an ECDF, and the black arrow is the K-S statistic (from Wikipedia).

evaluate normality (or similarity to some specific distribution) can be broadly divided into two categories:

1. Tests based on comparison (best fit) with a given distribution, often specified in terms of its CDF. Examples are the Kolmogorov-Smirnov test, the Lilliefors test, the Anderson-Darling test, the Cramr-von Mises criterion, as well as the Shapiro-Wilk and Shapiro-Francia tests.

2. Tests based on descriptive statistics of the sample. Examples are the skewness test, the kurtosis test, the DAgostino-Pearson omnibus test, or the Jarque-Bera test.

For example, the *Lilliefors test*, which is based on the *Kolmogorov–Smirnov test* , 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. (The original Kolmogorov-Smirnov test should be used carefully, especially if the number of samples is ca. ≤ 300.)

The *Shapiro-Wilk W test*, which depends on the covariance matrix between the order statistics of the observations, can also be used with ≤ 50 samples, and has been recommended by [Altman(1999)] and by [**?**].

The Python command `stats.normaltest(x)` uses the DAgostino-Pearson *omnibus test* . This test combines a skewness and kurtosis test to produce a single, global, "omnibus" statistic.

Code: "checkNormality.py" (p 185) shows how to check graphically and quantitatively if a given distribution is normal.

6.1.3 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 5.17.

6.2 Hypothesis tests

6.2.1 An Example

Assume that you are running a private educational institution. Your contract says that if your students score 110 in the final exam, where the national average is 100, you get a bonus. When the results are significantly lower, you loose your bonus (because the students are not good enough), and you have to hire more teachers; and when the results are significantly higher, you also loose your bonus (because you have spent too much money on teachers), and you have to cut back on the number of teachers.

The final exam of your 10 students produce the following scores (Fig. 6.4):

```
scores = array([ 109.4, 76.2, 128.7, 93.7, 85.6, 117.7, 117.2, 87.3, 100.3,
    55.1])
```

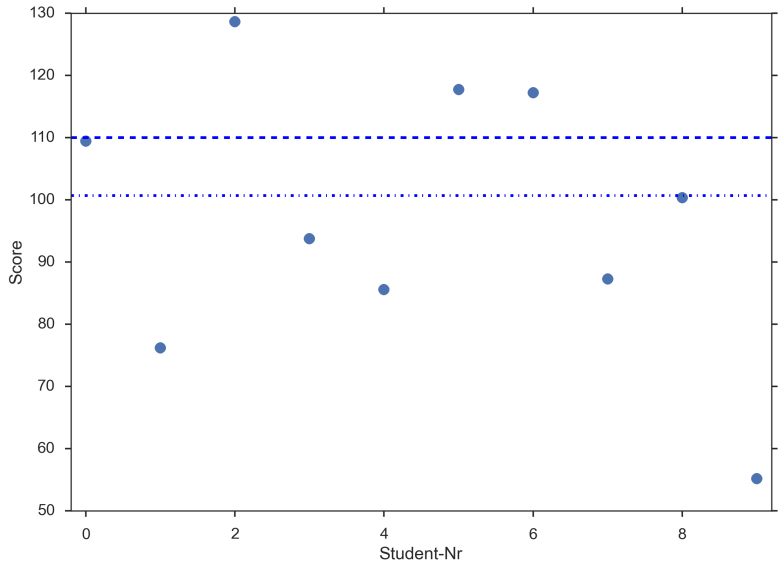


Figure 6.4: The question we ask: based on our sample mean (dashed-dotted line) and the obeserved variance of the data (the sample variance), do we believe that the population mean is different from 110 (dashed line)? Or in other words: our Null-hypothesis is that the difference between the population mean and 110 is zero.

The question we want to answer: Is the mean value of the scores (*97.1*) significantly different from 110?

A *normality test* (`stats.normaltest(scores)`) indicates that the data are probably taken from a normal distribution. Since we don't know the *population variance* of the results of students tested, we have to take our best guess, the *sample variance* (see also Fig. 4.9). And we know that the normalized difference between sample and the population mean, the *t-statistic*, follows the t-distribution (Eq. 5.2.4).

The difference between our sample mean and the value we want to compare it to (`np.mean(scores)- 110`) is *12.9*. Normalized by the sample standard error (Eq. 5.2.4), this gives a value of *t=-1.84*. Since the t-distribution is a known curve that depends only on the number of samples, and we can calculate the likelihood that we obtain a t-statistic of $|t| > 1.84$:

```
tval = (110-np.mean(scores))/stats.sem(scores)  # 1.84
td = stats.t(len(scores)-1)                     # "frozen" t-distribution
p = 2*td.sf(tval)                               # 0.0995
```

(The factor *2* in the last line of the code is required, since we have to combine the probability of $t < -1.84$ and $t > 1.84$.) Expressed in words, given our sample data, we can state that the likelihood that the population mean is *110* is 9.95%. But since a *statistical difference* is only given by convnetion if the likelihood is less than 5%, we conclude that the observed value of 97.1 is not significantly different from 110, and the bonus has to be paid out.

6.2.2 Generalization

Based on the previous example, the general procedure for *hypothesis tests* can be described as follows (the sketch in Fig. 4.9 indicates the meaning of many upcoming terms):

- A random sample is drawn from a population. (In our example, the random sample is our *scores*).
- A *Null hypothesis* is formulated. ("There is Null difference between the population mean and the value of 110.")
- A *test-statistic* is calculated, from which we know the probability distribution. (Here the *sample mean*, since we know that the mean value of samples from a normal distribution follows the t-distribution.)
- Comparing the observed value of the statistic (here the obtained t-value) with the corresponding distribution (the t-distribution), we can find the likelihood that a value as extreme as or more extreme than s the observed one is found by chance. This is the *p-value*.
- If the p-value is $p < 0.05$, we reject the Null-hypothesis, and speak of a *statistically significant difference*. If a value of $p < 0.001$ is obtained, the result is typically called *highly significant*. 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.

In other words, the p-value states how likely it is to obtain a value as extreme or more extreme by chance alone, *if the Null hypothesis is true*.

The value against which the p-value is compared is the *significance level*, and is often indicated with the letter α. The significance level is chosen, and typically set to 0.05.

This way of proceeding to test a hypothesis is called *statistical inference*.

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.

Additional Examples

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."*

6.2.3 The interpretation of the p-value, and the "p-value fallacy"

Note: A value of *p¡0.05* for the null hypothesis has to be interpreted as follows: *If the null hypothesis is true, the chance that we find a test statistic as extreme or more extreme than the one observed is less than 5%.* This is *not* the same as saying that the null hypothesis is false, and even less so, that an alternative hypothesis is true! Stating a p-value alone is no longer state-of-the-art for the statistical analysis of data. You should also state the the confidence intervals for the parameter that you investigate.

Therefore research is sometimes divided into *exploratory research* and *confirmatory research*. Take for example the case of Matt Motyl, a Psychology PhD student at the University of Virginia. In 2010, data from his study of nearly 2000 people indicated that political moderates saw shades of grey more accurately than people with more extreme political opinions, with a p-value of 0.01. However, when he tried to reproduce the data, the p-value dropped down to 0.59. So while the *exploratory research* showed that a certain hypothesis may be likely, the *confirmatory research* showed that the hypothesis did not hold ([Nuzzo(2014)]).

[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:

$$\alpha(p) = \frac{1}{1 + \frac{1}{-e\ p\ log(p)}} \tag{6.1}$$

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

6.2.4 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 6.5 shows the meaning of the *power* of a statistical test. Note that for finding the power of a test, you need an alternative hypothesis.

Type II Errors and P-value

In other words, 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!

6.2.5 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 4^{th} 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*.

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

$$n = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2}{d^2} \tag{6.2}$$

Here z is the standardized normal variable (see also chapter 5.2.1)

$$z = \frac{x-\mu}{\sigma}. \tag{6.3}$$

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

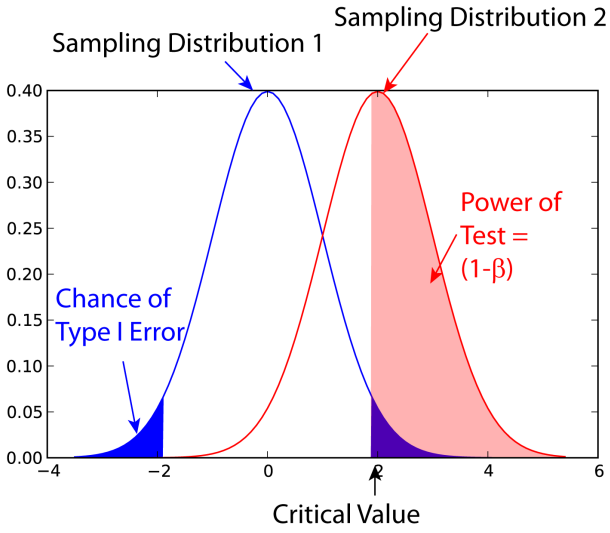


Figure 6.5: *Power* of a statistical test, for comparing the mean value of two sampling distributions.

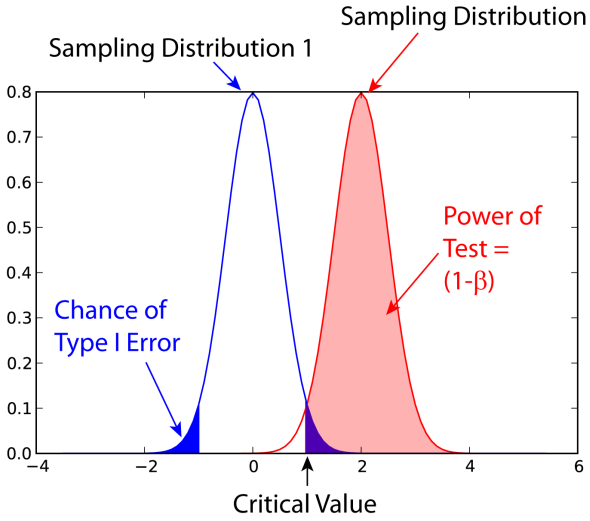


Figure 6.6: Effect of an increase in sampling size on the power of a test.

$$n_1 = n_2 = \frac{(z_{1-\alpha/2} + z_{1-\beta})^2(\sigma_1^2 + \sigma_2^2)}{D^2}. \tag{6.4}$$

Python Solution

statsmodels makes clever use of the fact that 3 of the 4 factors mentioned above are independent, and combines it with the Python feature of *named parameters* to provide a program that takes 3 of those parameters as input, and calculates the remaining 4^{th} parameter.

For example,

```
from statsmodels.stats import power
print(power.tt_ind_solve_power(effect_size = 0.5, alpha =0.05, power=0.8))
# Result: 63.77
```

tells us that if we compare two groups with the same number of subjects and the same standard deviation, require an $\alpha = 0.05$ a test power of 80%, and we want to detect a difference between the groups that is half the standard deviation, we need to test 64 subjects.

Similarly,

```
effect_size = power.tt_ind_solve_power(alpha =0.05, power=0.8, nobs1=25)
# Result: 0.81
```

tells us that if we have an $\alpha = 0.05$, a test power of 80%, and 25 subjects in each group, then the smallest difference between the groups is 81% of the sample standard deviation.

The corresponding command for one sample t-tests is `tt_solve_power`.

Programs: SampleSize

 **Code:** "sampleSize.py" (p 187): sample size calculation for normally distributed groups with arbitrary standard deviations.

6.3 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:

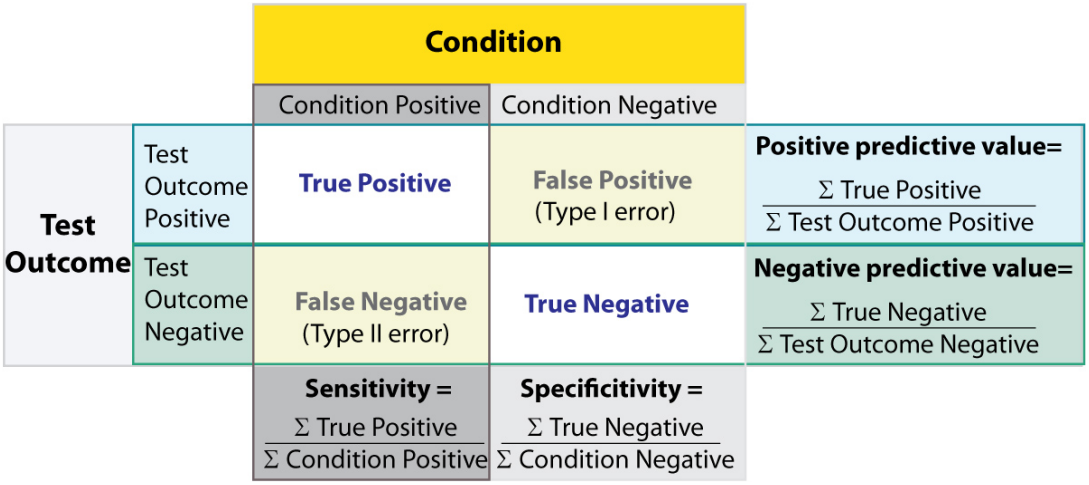


Figure 6.7: Relationship between sensitivity, specificity, positive predictive value and negative predictive value.

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 (PPV) Proportion of patients with positive test results who are correctly diagnosed.

Negative Predictive Value (NPV) Proportion of patients with negative test results who are correctly diagnosed.

For example, *pregnancy tests* have a high sensitivity: when a woman is pregnant, the probability that the test is positive is very high.

In contrast, an indicator for an attack with atomic weapons on the White House should have a very high specificity: if there is no attack, the probability that the test goes on should be very, very small.

Sensitivity and *specificity* characerize a test, while *PPV* and *NPV* are the parameters that tell the doctor how to interpret a test result.

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. 6.8 indicates, these values are affected by the *prevalence* of the disease. In other words, we need to know the prevalence of the disease as well as the PPV/NPV of a test to provide a sensible interpretation of the test results.

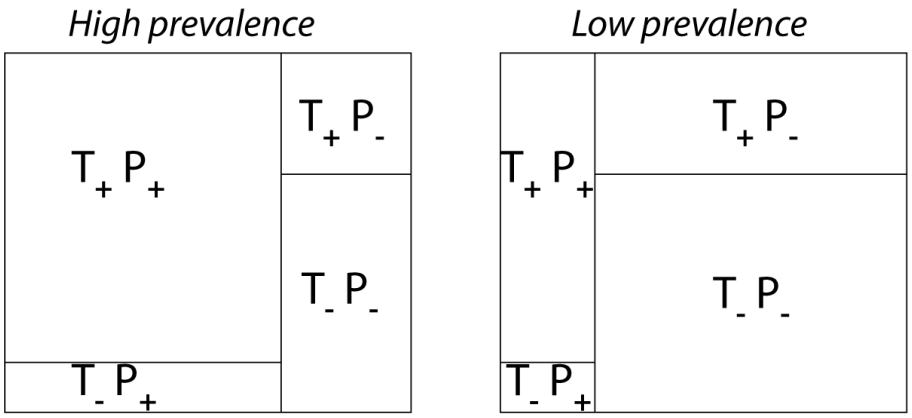


Figure 6.8: Effect of prevalence on PPV and NPV. "T" stands for "test", and "P" for "patient". (For comparison with below: T+P+ = TP, T-P- = TN, T+P- = FP, and T-P+ = FN)

Figure 6.9 gives a worked example:

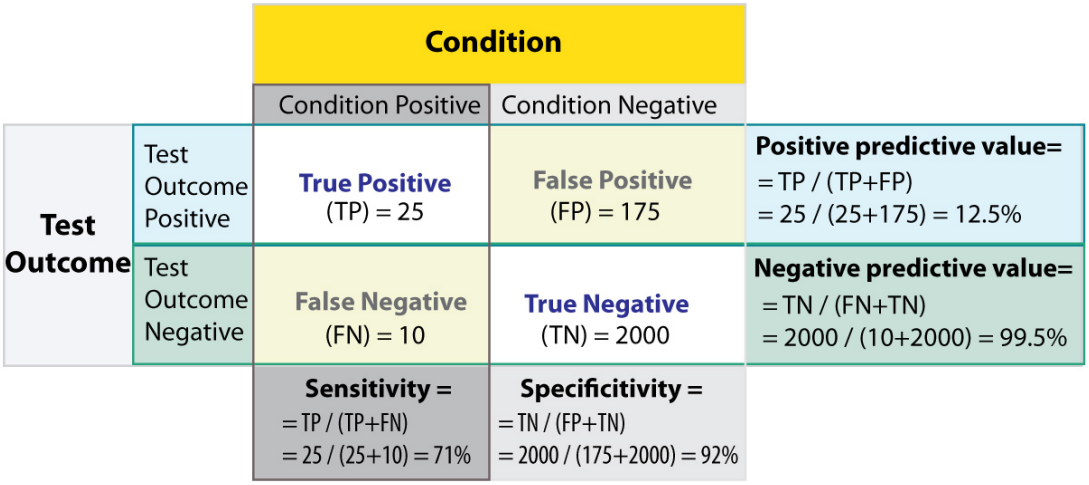


Figure 6.9: Worked example.

Related calculations

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

- Power = sensitivity = $1 - \beta$
- Likelihood ratio positive = $\frac{sensitivity}{1specificity} = \frac{66.67\%}{191\%} = 7.4$
- Likelihood ratio negative = $\frac{1sensitivity}{specificity} = \frac{166.67\%}{91\%} = 0.37$

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

6.4 ROC Curve

Closely related to *Sensitivity* and *Specificity* is the Reiciver Operating Characteristic (ROC) (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 6.10) this point is given by the value with the largest distance to the diagonal.

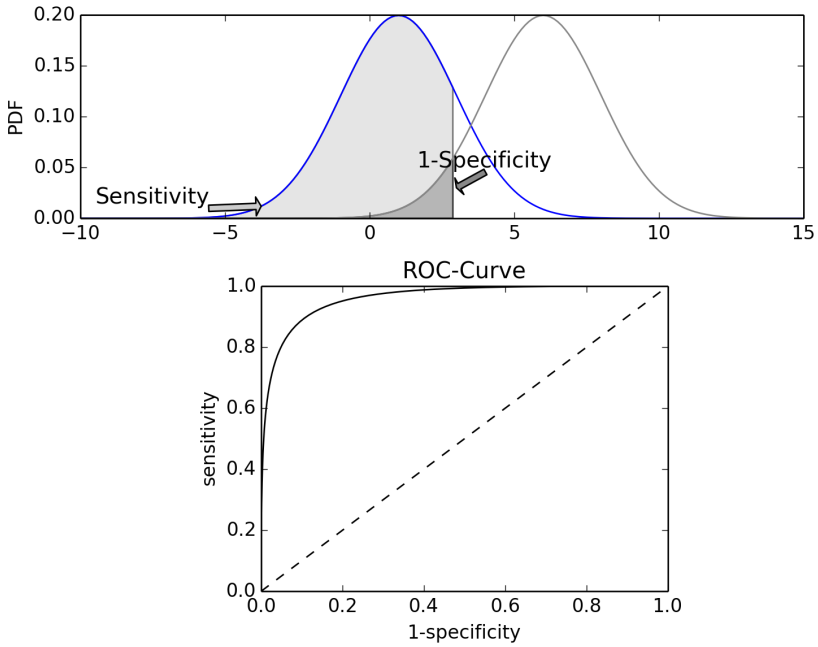


Figure 6.10: Top: Probability density functions for two distributions. Bottom: corresponding *ROC-curve*.

6.5 Common Statistical Tests for Comparing Groups

Table 6.1 gives an overview of the most common statistical tests for different combinations of data.

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 6.1: Typical tests for statistical problems.

6.5.1 Examples

2 groups, nominal male/female, blond-hair/black-hair. E.g. "Are females more blond than males?"

2 groups, nominal, paired 2 labs, analysis of blood samples. E.g. "Does the blood analysis from Lab1 indicate more infections than the analysis from Lab2?"

2 groups, ordinal black/white, ranking 100m sprint. E.g. "Are black sprinters more successful than white sprinters?"

2 groups, ordinal, paired sprinters, before/after diet. E.g. "Does a chocolate diet make sprinters more successful?"

3 groups, ordinal black/white/chinese, ranking 100m sprint. E.g. "Does ethnicity have an effect on the success of sprinters?"

3 groups, ordinal, paired sprinters, before/after diet. E.g. "Does a rice diet make Chinese sprinters more successful?"

2 groups, continuous male/female, IQ. E.g. "Are women more intelligent than men?"

2 groups, continuous, paired male/female, looking at diamonds. E.g. "Does looking at diamonds raise the female heart-beat more than the male?

3 groups, continuous Tyrolians, Viennese, Styrians; IQ. E.g. "Are Tyrolians smarter than people from other Austrian federal states?"

3 groups, continuous, paired Tyrolians, Viennese, Styrians; looking at mountains. E.g. "Does looking at mountains raise the heartbeat of Tyrolians more than those of other people?"

6.6 Exercises

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.

Chapter 7

Test of Means of Continuous Data

7.1 Distribution of a Sample Mean

7.1.1 One sample t-test for a mean value

To check the mean value of normally distributed data against a reference value, we typically use the *one sample t-test*, which is based on the *t-distribution*.

If we knew the mean and the standard deviation of a normally distributed population, we could calculate the corresponding 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 t-distribution for the mean slightly deviates from a normal distribution.

Example

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 in the example is 0.057, and from the normal distribution 0.054*

Since it is very important to understand the basic principles of how we arrive at the t-statistic and the corresponding p-value for this test, let me illustrate the underlying statistics by going step-by-step through the analysis:

- We have a population, with a mean value of 7 and a standard deviation of 3.
- From that population an observer takes 100 random samples. The sample mean is 7.10, close to but different from the real mean. The sample standard deviation is 3.12, and the standard error of the mean 0.312. This gives the observer an idea about the variability of the population.
- The observer knows that the distribution of the real mean follows a t-distribution, and that the *standard error of the mean* characterizes the width of that distribution.
- How likely it is that the real mean has a value of x_0 (e.g. 6.5, indicated by the red triangle in Fig. 7.1, left)? To find that out, this value has to be transformed, by subtracting the sample mean, and dividing by the standard error. (Fig. 7.1, right). This provides the *t-statistic* for this test (-1.93).
- The corresponding *p-value*, which tells us how likely it is that the real mean has a value of 6.5 or more extreme relative to the sample mean, is given by the red shaded area under the curve-wings: *2*CDF(t-statistic) = 0.057*, which means that the difference to 6.5 is

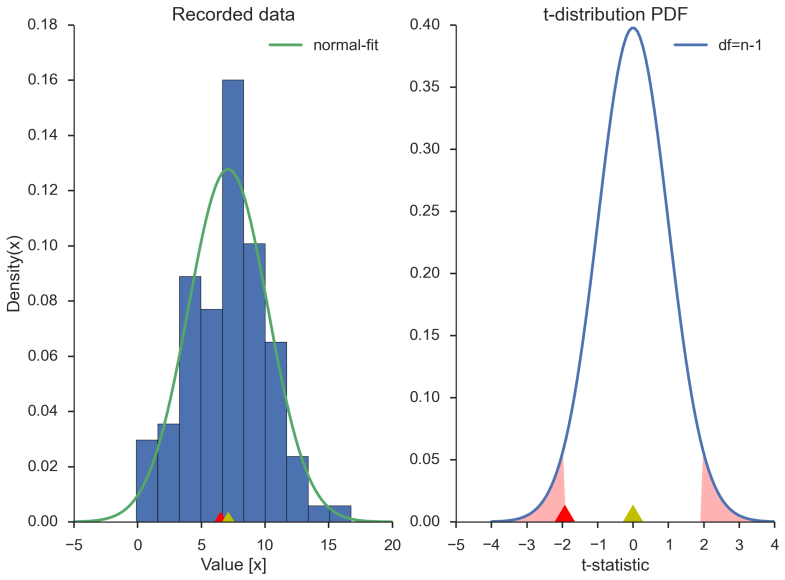


Figure 7.1: Left: Frequency histogram of the sample data, together with a normal fit. The sample mean, which is very close to the population mean, is indicated with a yellow triangle; the value to be checked with a red triangle. Right: sampling distribution of t (for n-1 degrees of freedom). At the bottom the normalized value of sample mean (yellow triangle) and value to be checked (red triangle). The red shaded area corresponds to the p-value.

just not significant. (The factor "2" comes from the fact that we have to check in both tails.)

Code: "oneSample.py" (p 188): Sample analysis for one group of continuous data.

7.1.2 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, see Fig. 5.13). 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 7.1 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).

[1] Python Example: scipy.stats.wilcoxon, in "univariate.py"
[2] The 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 7.1: Daily energy intake of 11 healthy women with rank order of differences (ignoring their signs) from the recommended intake of 7725 kJ.

7.2 Comparison of Two Groups

7.2.1 Paired T-Test

When we 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 grown. 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.

7.2.2 Unpaired T-Test

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:

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

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

Expressed in words, *The variance of a sum of independent random variables equals the sum of the variances.*

7.2.3 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 from the *Wilcoxon signed rank sum test*!

Code: "twoSample.py" (p 190): Comparison of two groups, paired and unpaired.

7.2.4 Statistical Hypothesis 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)
(t, pVal) = stats.ttest_rel (race_1, race_2)
print('The probability that the two distributions are equal is {0}'.format(
    pVal))
```

The command `random.seed(123)` initializes the random number generator with the number 123, which ensures that two consecutive runs of this code produce the same result, corresponding to the numbers given above.

Statistical Modeling

```
import pandas as pd
import statsmodels.formula.api as sm
np.random.seed(123)
df = pd.DataFrame({'Race1': race_1, 'Race2':race_2})
result = sm.ols(formula='I(Race2-Race1) ~ 1', data=df).fit()
print(result.summary())
```

The important line is the last but one, which produces the *results*. Thereby the *ordinary least square (ols)* function from *statsmodels* tests the model which describes the difference between the results of *Race1* and those of *Race2* with only an *offset* (also called *intercept* in the language of modeling). In other words, our model has only one paramter, the *intercept*. The results below show that the probability that this intercept is 0 is only 0.03: the difference is *significant*.

```
                            OLS Regression Results
==============================================================================
```

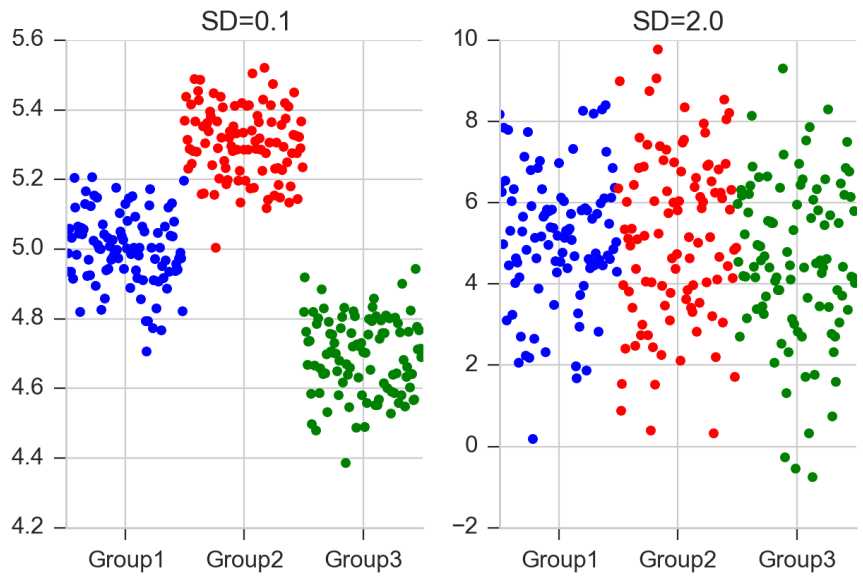


Figure 7.2: In both cases, the difference between the two groups is the same. But left, he difference within the groups is smaller than the differences between the groups, whereas right, the difference within the groups is larger than the difference between.

```
Dep. Variable:       I(Race2 - Race1)   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                       nan
Date:                Sun, 08 Feb 2015   Prob (F-statistic):               nan
Time:                        18:48:06   Log-Likelihood:                -85.296
No. Observations:                  20   AIC:                             172.6
Df Residuals:                      19   BIC:                             173.6
Df Model:                           0
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     -9.1000      3.950     -2.304      0.033       -17.367    -0.833
==============================================================================
Omnibus:                        0.894   Durbin-Watson:                   2.009
Prob(Omnibus):                  0.639   Jarque-Bera (JB):                0.793
Skew:                           0.428   Prob(JB):                        0.673
Kurtosis:                       2.532   Cond. No.                         1.00
==============================================================================
```

The output is explained in more model in the chapter 13. The important point here is that the t- and p-value for the *intercept* are the same as with the classical t-test above.

7.3 Comparison of More Groups

7.3.1 Analysis of Variance - ANOVA

The idea behind the ANalysis Of VAriance (ANOVA) (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! (This is called a *balanced ANOVA*: a balanced design has an equal number of observations for all possible combinations of factor levels.)

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 the equal variance assumption, one of the popular approaches is the *Levene test*.

ANOVA uses traditional terminology. DF indicates the degrees of freedom (DF) (see also section 4.4), the summation is called the Sum of Squares (SS) (SS), the result is called the mean square (MS) and the squared terms are deviations from the sample mean. In general, the *sample variance* is defined by

$$s^2 = \frac{1}{DF}\sum(y_i - \bar{y})^2 = \frac{SS}{DF} \tag{7.1}$$

The fundamental technique is a partitioning of the total sum of squares SS into components related to the effects used in the model (Fig. 7.3). Thereby ANOVA estimates 3 sample variances: a *total variance* based on all the observation deviations from the grand mean (calculated from SS_{Total}), a *treatment variance* (from $SS_{Treatments}$), and an *error variance* based on all the observation deviations from their appropriate treatment means (from SS_{Error}). 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).

$$SS_{\text{Total}} = SS_{\text{Error}} + SS_{\text{Treatments}} \tag{7.2}$$

where SS_{Total} is sum-squared devation from the overall mean, the SS_{Error} the sum-squared deviation from the mean within a group, and the $SS_{treatment}$ the sum-squared deviation between each group and the overall mean (Fig. 7.3).

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.

$$DF_{\text{Total}} = DF_{\text{Error}} + DF_{\text{Treatments}} \tag{7.3}$$

Example: one-way ANOVA

As an example, let us take the red cell folate levels ($\mu g/l$) 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
```

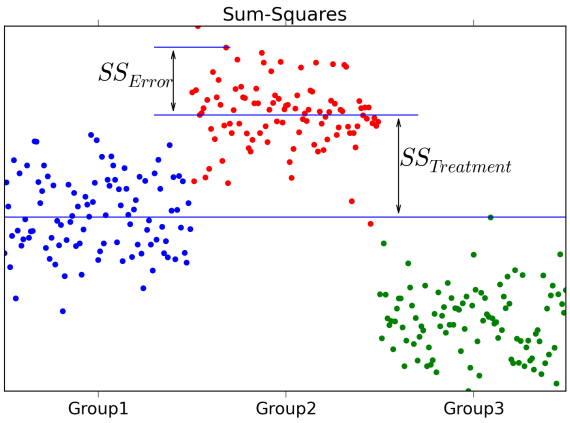


Figure 7.3: The long blue line indicates the grand mean over all data. The SS_{Error} describes the variability *within* the groups, and the $SS_{Treamtent}$ (summed over all respective points!) the variability *between* groups.

- 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.15).

$$F = \frac{\text{variance between treatments}}{\text{variance within treatments}} \tag{7.4}$$

$$F = \frac{MS_{\text{Treatments}}}{MS_{\text{Error}}} = \frac{SS_{\text{Treatments}}/(n_{groups}-1)}{SS_{\text{Error}}/(n_{total}-n_{groups})} \tag{7.5}$$

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

 **Code:** "anovaOneway.py" (p 192): different aspects of one-way ANOVAs: how to check the assumptions (with the Levene test), different ways to calculate a one-way ANOVA, and a demonstration that for the comparison between groups, a one-way ANOVA is equal to a T-test.

7.3.2 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

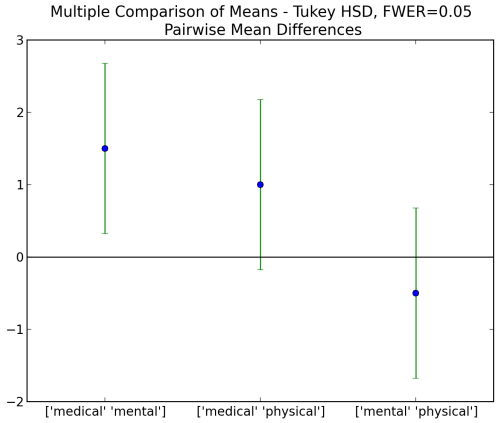


Figure 7.4: Comparing the means of multiple groups - here three different treatment options.

null hypothesis is true. This can be cone 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 test (Tukey HSD) (HSD) method, controls for the Type I error rate across multiple comparisons and is generally considered an acceptable technique. It is based on a statistic that we have not come across yet, the *studentized range*, which is commonly represented by the variable q. The *studentized range* computed from a list of numbers $x_1, ..., x_n$ is given by

$$q_n = \frac{\max\{ x_1, \ \dots \ x_n \} - \min\{ x_1, \ \dots \ x_n\}}{s} \tag{7.6}$$

where s is the sample standard deviation. In the Tukey HSD method the sample $x_1, ..., x_n$ is a sample of means and q is the basic test-statistic. It can be used as post-hoc analysis to test between which two groups means there is a significant difference (pairwise comparisons) after rejecting the null hypothesis that all groups are from the same population (i.e. all means are equal).

Code: "multipleTesting.py" (p 195): 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),
  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). As stated by Holm ([Holm(1979)])"Except in trivial non-interesting cases the sequentially rejective Bonferroni test has strictly larger probability of rejecting false hypotheses and thus it ought to replace the classical Bonferroni test at all instants where the latter usually is applied."

7.3.3 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*.

Code: "KruskalWallis.py" (p 198): Example of a Kruskal-Wallis test (for not normally distributed data).

7.4 Exercises

One or Two Groups

1. **Paired T-Test** and **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.

   Is this value significantly different from the recommended value of 7725? (Correct answer: yes, $p_{ttest} = 0.018$, and $p_{Wilcoxon} = 0.026$)

2. **t-test of independent samples**

   In a clinic, 15 lazy 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 lazy patients significantly heavier? (Correct answer: yes, p=0.045)

3. **Normality test**

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

4. **Mann-Whitney test**

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

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* 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 TreamtmentA and TreatmentB differ)

4. **Kruskal-Wallis**

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

Chapter 8

Tests on Categorical Data

In a data sample the number of data 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*. For example, Table 8.1 gives the number of right/left-handed subjects, *contingent* on the subject being male or female.

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.

8.1 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. The occurrence of a characteristic in a group of n people is described

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

Table 8.1: Example of a frequency table

by the binomial distribution, with $mean = p * n$. The standard error of samples with this characteristic is given by

$$se(p) = \sqrt{p(1-p)/n} \tag{8.1}$$

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.

8.1.1 Explanation

The innocent looking equation 8.1 is more involved than it seems at first:

If you have n independent samples from a binomial distribution $B(k,p)$, the variance of their sample mean is

$$\text{var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\text{var}(X_i) = \frac{n\ \text{var}(X_i)}{n^2} = \frac{\text{var}(X_i)}{n} = \frac{kpq}{n}$$

where $q = 1p$ and $\bar{X}$ is the same mean. This follows since

1. $var(cX) = c^2 var(X)$, for any random variable, X, and any constant c.
2. the variance of a sum of independent random variables equals the sum of the variances.

The standard error of $\bar{X}$ is the square root of the variance: $\sqrt{\frac{kpq}{n}}$. Therefore,

- When $k = n$, we get $se = \sqrt{pq}$.
- When $k = 1$, and the Binomial variables are just Bernoulli trials, the standard error is given by $se = \sqrt{\frac{pq}{n}}$.

8.1.2 Example

For example, let us look at incidence and mortality for breast cancer, and try to answer the following two questions: among the FH-students, how many occurrences of breast cancer should we expect per year? And how many of the female FH-students will probably die from breast cancer at the end of their life?

We know that:

- the FH OOe has about 5'000 students, about half of which are female.
- breast cancer hits predominantly women.
- the *incidence* of breast cancer in the age group 20-30 is about 10, where *incidence* is typically defined as the new occurrences of a disease per year per 100'000 people.
- 3.8% of all women die of cancer.

From these points of information, we can obtain the following parameters for our calculations

- n = 2'500
- $p_{incidence} = 10/100'000$
- $p_{mortality} = 3.8/100$.

	Right Handed	*Left Handed*	*Total*
Males	45.2	6.8	52
Females	41.8	6.2	48
Total	87	13	100

Table 8.2: Corresponding *expected values* for Table 8.1

The 95% confidence interval for the incidence is -0.7 - 1.2, and for the number of deaths 76 - 114. So we expect that every year most likely none or one of the FH-students will be diagnosed with breast cancer; but between 76 and 114 of the female students will eventually die from this disease.

8.2 Frequency Tables

If your data can be organized in a set of categories, and they are given as *frequencies*, i.e. the total number of samples in each category (not as percentages), the tests described in this section are appropriate for your data analysis.

Many of these tests analyze the *deviation from an expected value*. Since the chi-square distribution characterizes the variability of data (in other words, their deviation from a mean value), many of these tests refer to this distribution, and are accordingly termed *chi-square tests*.

Assume that you have observed absolute frequencies o_i and expected absolute frequencies e_i. Under the Null hypothesis all your data come from the same population, and the test statistic

$$V = \sum_i \frac{(o_i - e_i)^2}{e_i} \approx \chi^2_f \tag{8.2}$$

.

follows a chi square distribution with f degrees of freedom. i might denote a simple index running from $1, ..., I$ or a multiindex $(i_1, ..., i_p)$ running from $(1, ..., 1)$ to $(I_1, ..., I_p)$.

8.2.1 One-way Chi-square Test

For example, assume that you go hiking with your friends. Every evening, you draw lots who has to do the washing up. But at the end of the trip, you seem to have done most of the work:

You	Peter	Hans	Paul	Mary	Joe
10	6	5	4	5	3

You expect that there has been some foul play, and calculate how likely it is that this distribution came up by chance. The

$$expectedFrequency = \frac{n_{total}}{n_{people}} \tag{8.3}$$

is 5.5. The likelihood that this distribution came up by chance is

```
V, p = stats.chisquare(data)
print(p)
>>> 0.373130385949
```

In other words, you doing a lot of the washing up really could have been by chance!

8.2.2 Chi-square Contingency Test

If you can arrange your data in rows and columns, you can check if the numbers in the individual columns are contingent on the row value. For this reason, this test is sometimes called *contingency test.*

The chi-square contingency 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 (e.g. Table 8.2). When n is the total number of observations included in the table, the expected value for each cell in a two-way table is

$$expectedFrequency = \frac{RowTotal * ColumnTotal}{n} \tag{8.4}$$

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 (DOF) 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

$$\sum_i o_i = n \tag{8.5}$$

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 following list: (χ^2-value, p-value, degrees-of-freedom, expected values).

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

For the example data in Table 8.1, 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.

Note: These values assume the default setting, which uses the *Yates correction.* Without this correction, i.e. using Eq. 8.2, the results are $\chi^2 = 1.8, p = 0.18$.

		B	
	1	0	*Totals*
A 1	a	b	a+b
0	c	d	c+d
Totals	a+c	b+d	N=a+b+c+d

Table 8.3: General Structure of 2x2 Frequency Tables

Example 2

The Chi-square test can be used to generate a "quick and dirty" test of normality, e.g.

H_0 : The random variable X is symmetrically distributed versus

H_1 : the random variable X is not symmetrically distributed.

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

$$V = \frac{(n_- - n/2)^2}{n/2} + \frac{(n_+ - n/2)^2}{n/2} \approx \chi_1^2 \tag{8.6}$$

.

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.

8.2.3 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 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.

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 as or more extreme than the observed one. The latter one (which is the default in python) also considers tables as extreme in the opposite direction.

Note: The python command `stats.fisher_exact` returns by default the p-value for *finding a value as extreme or more extreme than the observed one.* According to Altman ([Altman(1999)]), this is a reasonable approach, although not all statisticians agree on that point.

Figure 8.1: First milk, then tea (top) - or first tea, then milk (bottom): Could you taste the difference?

Example: "A Lady Tasting Tea"

[1]R. A. Fisher was one of the founding fathers of modern statistics. One of his early, and perhaps the most famous, experiments was to test an English lady's claim that she could tell whether milk was poured before tea or not. Here is an account of the seemingly trivial event that had the most profound impact on the history of modern statistics, and hence, arguably, modern quantitative science [Box(1978)].

> Already, quite soon after he had come to Rothamstead, his presence had transformed one commonplace tea time to an historic event. It happened one afternoon when he drew a cup of tea from the urn and offered it to the lady beside him, Dr. B. Muriel Bristol, an algologist. She declined it, stating that she preferred a cup into which the milk had been poured first. "Nonsense," returned Fisher, smiling, "Surely it makes no difference." But she maintained, with emphasis, that of course it did. From just behind, a voice suggested, "Let's test her." It was William Roach who was not long afterward to marry Miss Bristol. Immediately, they embarked on the preliminaries of the experiment, Roach assisting with the cups and exulting that Miss Bristol divined correctly more than enough of those cups into which tea had been poured first to prove her case.
>
> Miss Bristol's personal triumph was never recorded, and perhaps Fisher was not satisfied at that moment with the extempore experimental procedure. One can be sure, however, that even as he conceived and carried out the experiment beside the trestle table, and the onlookers, no doubt, took sides as to its outcome, he was thinking through the questions it raised.

The real scientific significance of this experiment is in these questions. These are, allowing incidental particulars, the questions one has to consider before designing an experiment. We will look at these questions as pertaining to the "lady tasting tea", but you can imagine how these questions should be adapted to different situations.

[1]Adapted from Stat Labs: Mathematical statistics through applications by D. Nolan and T. Speed, Springer-Verlag, New York, 2000

- *What should be done about chance variations in the temperature, sweetness, and so on?* Ideally, one would like to make all cups of tea identical except for the order of pouring milk first or tea first. But it is never possible to control all of the ways in which the cups of tea can differ from each other. If we cannot control these variations, then the best we can do - we do mean the "best" - is by randomization.

- *How many cups should be used in the test? Should they be paired? In what order should the cups be presented?* The key idea here is that the number and ordering of the cups should allow a subject ample opportunity to prove his or her abilities and keep a fraud from easily succeeding at correctly discriminating the the order of pouring in all the cups of tea served.

- *What conclusion could be drawn from a perfect score or from one with one or more errors?* If the lady is unable to discriminate between the different orders of pouring, then by guessing alone, it should be highly unlikely for that person to determine correctly which cups are which for all of the cups tested. Similarly, if she indeed possesses some skill at differentiating between the orders of pouring, then it may be unreasonable to require her to make no mistakes so as to distinguish her ability from a pure guesser.

An actual scenario described by Fisher and told by many others as the "lady tasting tea" experiment is as follows.

- For each cup, we record the order of actual pouring and what the lady says the order is. We can summarize the result by a table like this:

		Order of actual pouring		
		Tea first	Milk first	
Lady says	Tea first	a	b	$a+b$
	Milk first	c	d	$c+d$
		$a+c$	$b+d$	n

  Here n is the total number of cups of tea made. The number of cups where tea is poured first is $a+c$ and the lady classifies $a+b$ of them as tea first. Ideally, if she can taste the difference, the counts b and c should be small. On the other hand, if she cannot really tell, we would expect a and c to be about the same.

- Suppose now that to test the lady, 8 cups of tea are prepared, 4 tea first, 4 milk first, and she is informed of the design (that there are 4 cups milk first and 4 cups tea first). Suppose also that the cups are presented to her in random order. Her task then is to identify the 4 cups milk first and 4 cups tea first.

  This design fixes the row and column totals in the table above to be 4 each. That is,

$$a+b=a+c=c+d=b+d=4.$$

  With these constraints, when any one of a, b, c, d is specified, the remaining three are uniquely determined:

$$b=4-a,\ c=4-a,\ \text{and}\ d=a$$

  In general, for this design, no matter how many cups (n) are served, the row total $a+b$ will equal $a+c$ because the subject knows how many of the cups are "tea first" (or one kind as supposed to the other). So once a is given, the other three counts are specified.

- We can test the discriminating skill of the lady, if any, by randomizing the order of the cups served. If we take the position that she has no discriminating skill, then the randomization

of the order makes the 4 cups chosen by her as tea first equally likely to be any 4 of the 8 cups served. There are $\binom{8}{4} = 70$ (in Python, choose `scipy.misc.comb(8,4,exact=True)`) possible ways to classify 4 of the 8 cups as "tea first". If the subject has no ability to discriminate between two preparations, then by the randomization, each of these 70 ways is equally likely. Only one of 70 ways leads to a completely correct classification. So someone with no discriminating skill has 1/70 chance of making no errors.

- It turns out that, if we assume that she has no discriminating skill, the number of correct classifications of tea first ("a" in the table) has a "hypergeometric" probability distribution (`hd=stats.hypergeom(8,4,4)` in Python). There are 5 possibilities: 0, 1, 2, 3, 4 for a and the corresponding probabilities (and Python commands for computing the probabilities) are tabulated below.

Number of correct calls	Python command	Probability
0	hd.pmf(0)	1/70
1	hd.pmf(1)	16/70
2	hd.pmf(2)	36/70
3	hd.pmf(3)	16/70
4	hd.pmf(4)	1/70

- With these probabilities, we can compute the p-value for the test of the hypothesis that the lady cannot tell between the two preparations. Recall that the p-value is the probability of observing a result as extreme or more extreme than the observed result assuming the null hypothesis. If she makes all correct calls,the p-value is 1/70 and if she makes one error (3 correct calls) then the p-value is $1/70 + 16/70 \sim 0.24$.

The test described above is known as "Fisher's exact test."

8.2.4 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 8.3): $p_A = (a+b)/N$ 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).

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

	After: present	*After: absent*	*Row total*
Before: present	101	121	222
Before: absent	59	33	92
Column total	160	154	314

Table 8.4: McNemar's Test: example

Subject	*Task 1*	*Task 2*	*Task 3*
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 8.5: Cochran's Q Test: Success or failure for 12 subjects on 3 tasks.

$$\chi^2 = \frac{(|b-c| - correctionFactor)^2}{b+c}. \tag{8.7}$$

where χ^2 has a chi-squared distribution with 1 degree of freedom. 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

$$\chi^2 = \frac{(|121-59| - 0.5)^2}{121+59} \tag{8.8}$$

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

8.2.5 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 8.5).

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 8.5), 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.

8.3 Analysis Programs

With computers, the computational steps are trivial:


Code: "compGroups.py" (p 200): Analysis of categorical data.

8.4 Exercises

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. (In the real, historical experiment, the lady got all eight cups correct.)

- 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)

Chi2 Contingency Test (1 DOF)

A test of the effect of a new drug on the heart rate has yielded the following results:

	Heart rate increased	*Heart rate NOT increased*	*Row total*
Treated	36	14	50
Not treated	30	25	55
Column total	66	39	105

- Does the drug affect the heart rate? (Correct answer: no)

- What would be the result if the response in one of the not-treated persons would have been different? Perform this test with and without the Yates-correction. (Correct anwer:
without Yates correction: yes, p=0.042
with Yates correction: no, p=0.067)

	Heart rate increased	*Heart rate NOT increased*	*Row total*
Treated	36	14	50
Not treated	29	26	55
Column total	65	40	105

One way Chi2-Test (> 1 DOF)

The city of Linz wants to know if people want to build a long beach along the Danube. They interview local people, and decide to collect 20 responses from each of the five age groups: (< 15, 15-30, 30-45, 45-60, > 60)

The questionnaire states: *"A beachside development will benefit Linz."*
and the possible answers are

1	2	3	4
Strongly agree	Agree	Disagree	Strongly Disagree

The city council wants to find out if the age of people influenced feelings about the development, particularly of those who felt negatively (i.e. "disagreed" or "strongly disagreed") about the planned development.

Age group (type)	Frequency of negative responses (Observed values)
< 15	4
15-30	6
30-45	14
45-60	10
> 60	16

The categories seem to show large differences of opinion between the groups.

- Are these differences significant? (Correct answer: yes, p=0.034)
- How many degrees of freedom does the resulting analysis have? (Correct answer: 4)

McNemar's Test

In a lawsuit regarding a murder the defense uses a questionnaire to show that the defendant is insane. As a result of the questionnaire, the accused claims "not guilty by reason of insanity".

In reply, the state attorney wants to show that the questionnaire does not work. He hires an experienced neurologist, and presents him with 40 patients, 20 of whom have completed the questionnaire with an "insane" result, and 20 with a "sane" result. When examined by the neurologist, the result is mixed: 19 of the "sane" people are found sane, but 6 of the 20 "insane" people are labelled as sane by the expert.

- Is this result significantly different from the questionnaire? (Correct answer: no)
- Would the result be significantly different, if the expert had diagnosed all "sane" people correctly? (Correct answer: yes)

	sane by expert	*insane by expert*	*Total*
sane	19	1	20
insane	6	14	20
Total	22	18	40

Chapter 9

Relation Between Two Continuous Variables

So far we have been dealing with problems where only one variable is measured. Expressions or functions which only depend on one variable are sometimes called *univariate*. If more than one variable is involved, we are dealing with *multivariate* problems. In the simplest case we have two varibles involved, and we need a *bivariate* data analysis.

For 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. For the correlation between many variables, you should look into "correlation tables" (nicely implemented in *seaborn*). And an extension of linear regression to more than two variables brings you into the realm of statistical modeling (13).

9.1 Correlation

9.1.1 Correlation Coefficient

The *correlation coefficient* between two variables answers the question: "Are the two variables related? I.e. if one variable changes, does the other also change?" If the two variables are normally distributed, the standard measure of determining the *correlation coefficient*, often ascribed to *Pearson* , is

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{9.1}$$

With the *sample covariance*

$$s_{xy} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1} \tag{9.2}$$

and s_x, s_y the sample standard deviations of the x and y values, respectively, Eq. 9.1 can also be written as

$$r = \frac{s_{xy}}{s_x \cdot s_y}. \tag{9.3}$$

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 9.4.

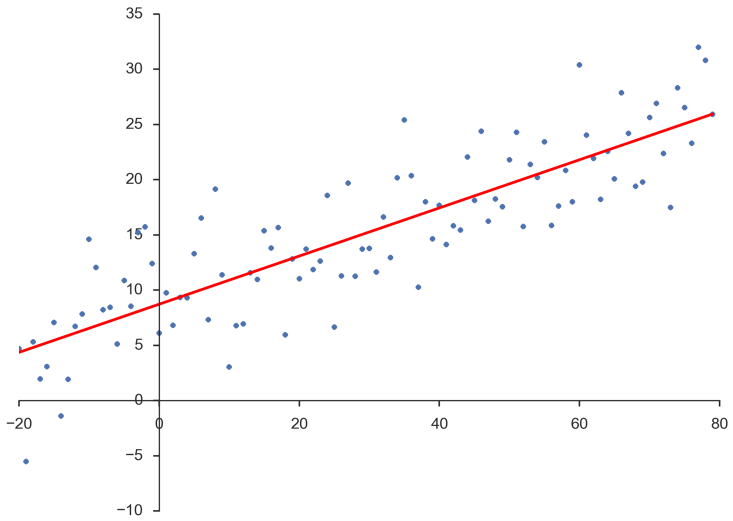


Figure 9.1: Linear regression.

Note that the formula for the correlation coefficient is symmetrical between x and y - which is not the case for linear regression!

9.1.2 Rank Correlation

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

- *Spearman's* ρ, which is exactly the same as the Pearson correlation coefficient r calculated on the ranks of the observations.
- *Kendall's* τ 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 Spearmans rho are less reliable and less interpretable than confidence intervals for Kendalls tau-parameters.

9.2 Regression

9.2.1 General linear regression model

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

$$y_i = k * x_i + d + \epsilon_i \tag{9.4}$$

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

9.2.2 Simple Regression

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

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

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

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

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

9.2.3 Design Matrix

Quadratic Fit

The equation for a quadratic fit to the given data is

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i, \tag{9.7}$$

This can be rewritten in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \\ 1 & x_6 & x_6^2 \\ 1 & x_7 & x_7^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{9.8}$$

General Formulation

In general, this can be rewritten in matrix form as:

$$\mathbf{y} = \mathbf{X} \cdot \beta + \varepsilon \tag{9.9}$$

$\mathbf{y}$ is a vector of dimension $(n \times 1)$ and is called the *endogenous variable*, the *design matrix* $\mathbf{X}$ is a matrix of dimension $(n \times k)$ where each column is an explanatory variable, and ε is the error term, the elements of which are assumed to be normally distributed about zero. β is the vector of dimension $(k \times 1)$ and contains the parameters we want to estimate.

9.2.4 Coefficient of determination

In order to interpret r, let me first define a few common terms.

Residuals Differences between observed values and predicted values.

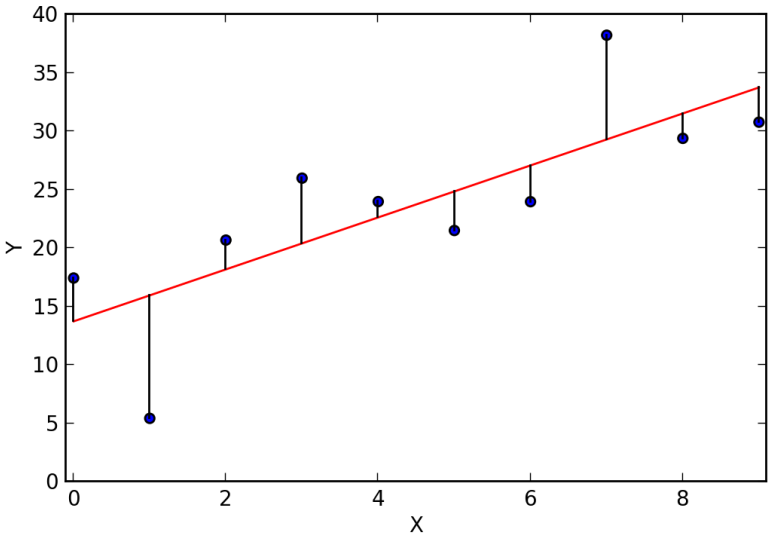


Figure 9.2: Best-fit linear regression line (red) and residuals (black).

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:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{9.10}$$

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);

$SS_{\text{mod}} = \sum_i (f_i - \bar{y})^2$, the sum of squares of the model values, 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.

With these expressions, the most general definition of the *coefficient of determination*, R^2 is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}. \tag{9.11}$$

Since

$$SS_{\text{tot}} = SS_{\text{mod}} + SS_{\text{res}} \tag{9.12}$$

Eq. 9.11 is equivalent to

$$R^2 = \frac{SS_{\text{mod}}}{SS_{\text{tot}}} \tag{9.13}$$

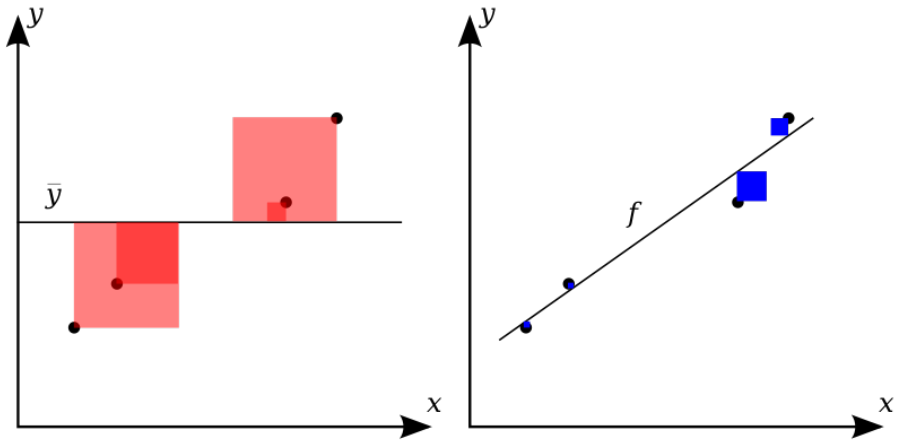


Figure 9.3: The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of R^2 is to one. The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value (from Wikipedia)

For simple linear regression (i.e. line-fits), the *coefficient of determination* or R^2 is the square of the correlation coefficient r. 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. Note that for general models it is common to write R^2, whereas for simple linear regression r^2 is used.

Relation to unexplained variance

In a general form, R^2 can be seen to be related to the unexplained variance, since the second term in Eq. 9.11 compares the unexplained variance (variance of the model's errors) with the total variance (of the data).

Examples

How large 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!

9.2.5 Coding

If you have vectors x, y containing your data, you can use `statsmodels` to create a design matrix that also includes the $1's$ for the offset:

```
import statsmodels.api as sm
Xmat = sm.add_constant(x)
```

The parameters are then easily found as

```
params = np.linalg.lstsq(Xmat, y)
```

However, you get a lot more information if you use the OLS-fit from `statmodels`:

```
import numpy as np
import statsmodels.api as sm

# Generate artificial data
nobs = 100
```

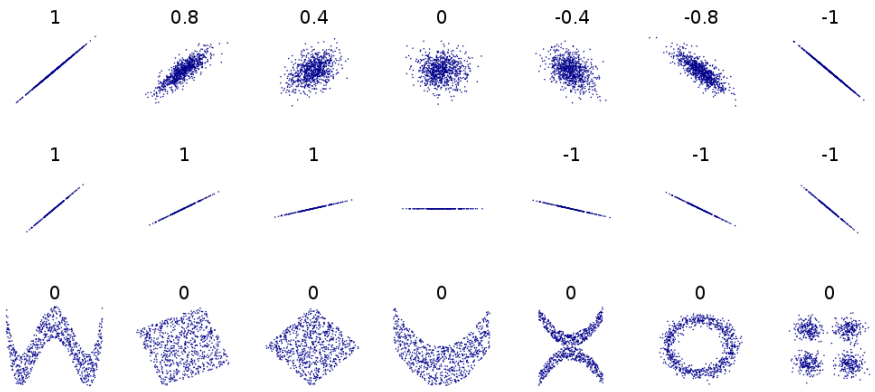


Figure 9.4: Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Note that the correlation reflects the non-linearity and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). 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. (In Wikipedia. Retrieved May 27, 2015, from http://en.wikipedia.org/wiki/Correlation_and_dependence)

```
X = np.random.random(nobs)
X = sm.add_constant(X)
beta = [5, 3.5]
e = np.random.random(nobs)
y = np.dot(X, beta) + e

# Fit regression model
results = sm.OLS(y, X).fit()

# Inspect the results
print(results.summary())
```

yields the following results:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.923
Model:                            OLS   Adj. R-squared:                  0.922
Method:                 Least Squares   F-statistic:                     1173.
Date:                Fri, 04 Jul 2014   Prob (F-statistic):           2.45e-56
Time:                        14:49:08   Log-Likelihood:                -15.390
No. Observations:                 100   AIC:                             34.78
Df Residuals:                      98   BIC:                             39.99
Df Model:                           1
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          5.4410      0.059     92.685      0.000         5.324      5.557
x1             3.5718      0.104     34.250      0.000         3.365      3.779
==============================================================================
Omnibus:                       21.620   Durbin-Watson:                   2.302
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                5.798
Skew:                           0.223   Prob(JB):                       0.0551
Kurtosis:                       1.908   Cond. No.                         4.60
==============================================================================
```

The meaning of many of these parameters is described in the chapter Statistical Models.

From the `results`, you can extract e.g. the model parameters, standard errors, confidence intervals, and residuals:

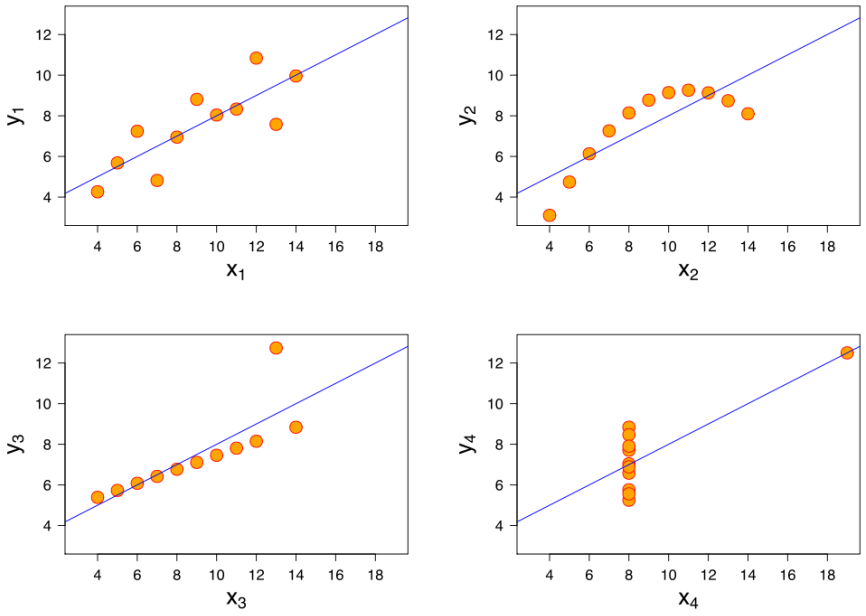


Figure 9.5: The sets in the *Anscombe's quartet* have the same linear regression line but are themselves very different.

```
params = results.params
std_err = results.bse
ConfInt = results.conf_int()
residuals = results.resid
```

9.2.6 Assumptions

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

1. The *independent variables* (i.e. x) are exactly known.
2. Validity. Most importantly, the data you are analyzing should map to the research question you are trying to answer. This sounds obvious but is often overlooked or ignored because it can be inconvenient. For example, a linear regression does not properly describe a quadratic curve.
3. Additivity and linearity. The most important mathematical assumption of the regression model is that its deterministic component is a linear function of the separate predictors.
4. Independence of errors from the values of the independent variables.
5. Equal variance of errors.
6. Normality of errors.

Code: "multivariate.py" (p 203): Analysis of multivariate data (regression, correlation).

Since to my knowledge there exists no program in the Python standard library (or numpy, scipy) to calculate the confidence intervals for a regression line, I include my corresponding program *lineFit.py* A.21. The output of this program is shown in Figure 9.6. This program also shows how Python programs intended for distribution should be documented.

Code: "fitLine.py" (p 205): Linear regression fit.

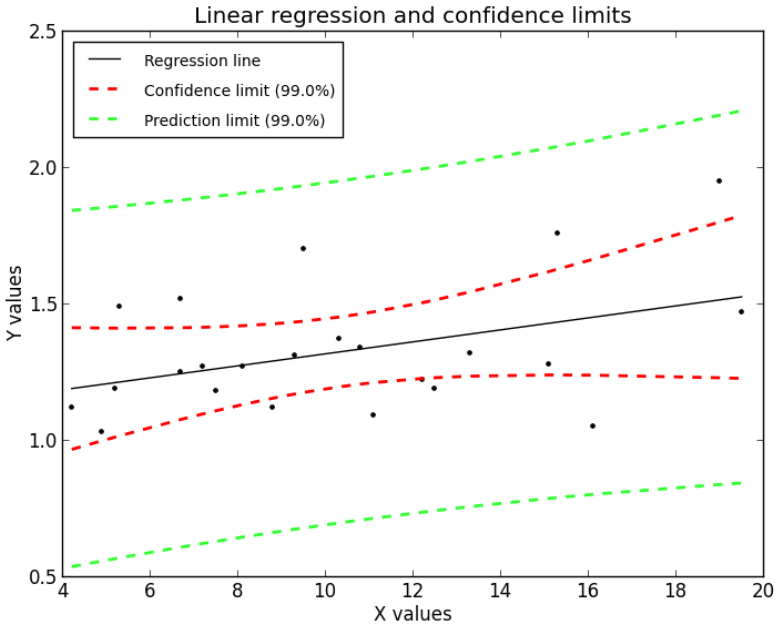


Figure 9.6: Regression, with confidence intervals for the mean, as well as for the predicted data. The red dotted line shows the confidence interval for the mean; and the green dotted line the confidence interval for predicted data. (This can be compared to the standard error and the standard deviation for a population.) The corresponding code can be found at p. 205

9.3 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 10

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.

When moving from two to many variables, the correlation coefficient gets replaced by the *correlation matrix*. And if we want to and predict the value of *many* other variables, linear regression has to be replaced by *multilinear regression*, sometimes also referred to as *multiple linear regression*.

However, watch out for the pitfalls that loom when you work with many variables! Take for example the following hypothetical case: you make a survey about the activity and life circumstances of a large range of people, covering all the numbers that you can get your hand on. In this survey you find out that a) rich people spend more time playing golf than poor people, and b) rich people tend to have fewer children than poor people. This leads to a strong negative correlation between playing golf and having children, and you may be tempted to (falsely) draw the conclusion that playing golf reduces your fertility, while in reality it is the higher income which causes both effects. [Kaplan(2009)] nicely describes where those problems come from, and how best to avoid them.

10.1 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

[1] It 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.

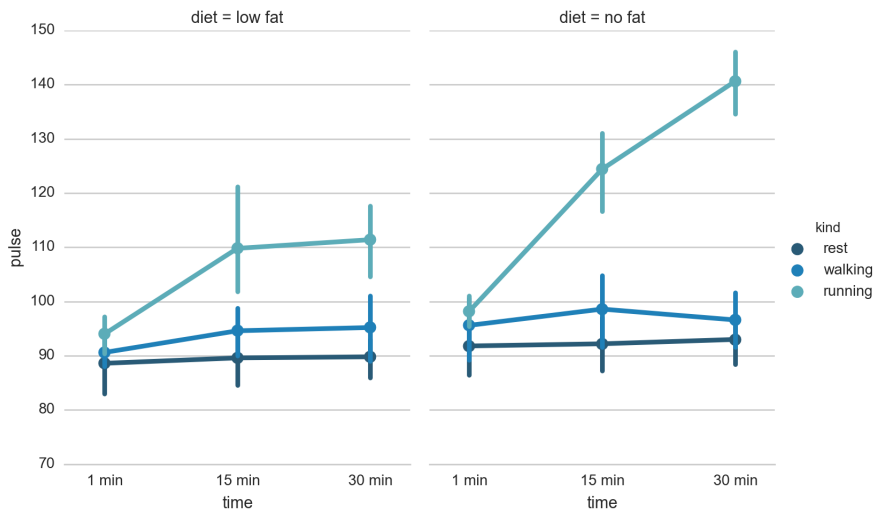


Figure 10.1: Three-way ANOVA.

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".

Code: "anovaTwoway.py" (p 208): 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
Residual            24    1.84    0.07      NaN       NaN
```

10.2 Three-way ANOVA

When you have more than two factors, it is recommendable to use *statistical modeling* for the data analysis (see Chapter 13). However, as always with the analysis of statistical data, you should first inspect the data visually. *seaborn* makes this quite simple:

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

df = sns.load_dataset("exercise")

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

10.3 Correlation Matrix

An elegant way to visualize the correlation between a large number of variables is the *correlation matrix*. Using *seaborn*, it can be implemented elegantly as follows:

```
import numpy as np
import seaborn as sns
```

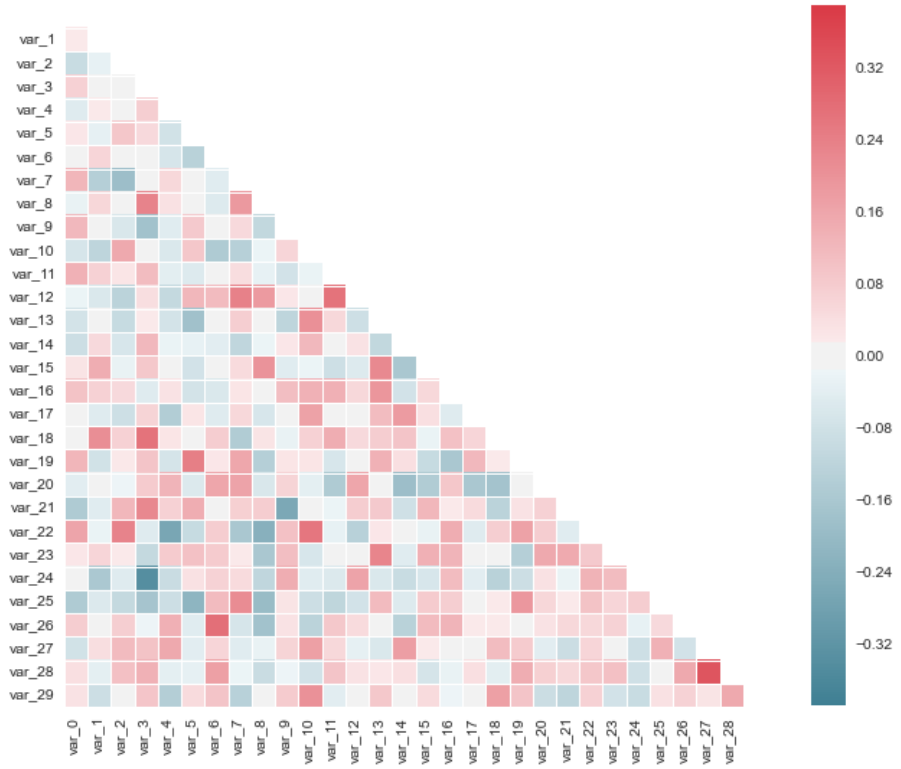


Figure 10.2: Visualization of the Correlation matrix.

```
import matplotlib.pyplot as plt
sns.set(style="darkgrid")

rs = np.random.RandomState(33)
d = rs.normal(size=(100, 30))

f, ax = plt.subplots(figsize=(9, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.corrplot(d, annot=False, sig_stars=False,
             diag_names=False, cmap=cmap, ax=ax)
f.tight_layout()
```

10.4 Multilinear Regression

If you have truly independent variables, *multilinear regression* is a straightforward extension of the simple linear regression.

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

$$y_i = \beta_0 + \beta_1 w_i + \beta_2 x_i + \epsilon_i \tag{10.1}$$

This model can be written in matrix terms as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & w_1 & x_1 \\ 1 & w_2 & x_2 \\ 1 & w_3 & x_3 \\ 1 & w_4 & x_4 \\ 1 & w_5 & x_5 \\ 1 & w_6 & x_6 \\ 1 & w_7 & x_7 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{10.2}$$

However, you have to watch out: 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.

Code: "mult_regress.py" (p 209): Multiple regression example.

Chapter 11

Analysis of Survival Times

When analyzing survival times, different problems come up than the ones discussed so far. One question is how to 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

$$p_k = p_{k-1} * \frac{r_k - f_k}{r_k} \tag{11.1}$$

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

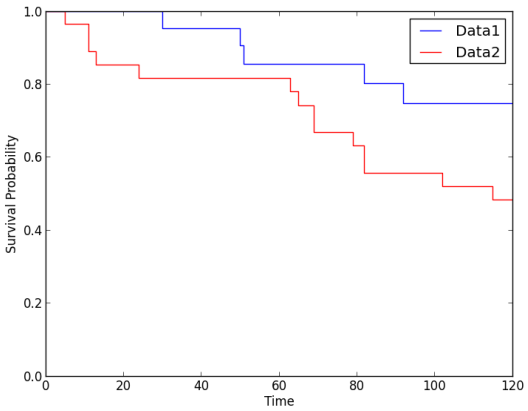


Figure 11.1: Survival curve corresponding to a motion sickness experiment, described in more detail in [Altman(1999)], chapter 13

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.29).

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 222): Survival analysis (Kaplan-Meier curves).

Chapter 12

Advanced Statistical Analysis

In this course we have presented the basic statistical data analysis with Python. However, Python has much more to offer: a number of Python packages allow you to significantly extend your statistical data analysis and modeling. In the following, I want to give a very brief overview of most interesting and powerful ones that I have found so far:

- statsmodels
- PyMC
- scikit-learn
- A.Dobson: "An Introduction to Generalized Linear Models"

12.1 statsmodels

Statsmodels is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator. Researchers across fields may find that statsmodels fully meets their needs for statistical computing and data analysis in Python. Features include:

- Linear Regression
- Generalized Linear Models
- Generalized Estimating Equations
- Robust Linear Models
- Linear Mixed Effects Models
- Regression with Discrete Dependent Variables
- ANOVA
- Time Series analysis
- Models for Survival and Duration Analysis
- Statistics (e.g. Multiple Tests, Sample Size Calculations etc.)
- Nonparametric Methods

- Generalized Method of Moments
- Empirical Likelihood
- Graphics functions
- A Datasets Package

A first introduction to statistical modeling, as well as some examples, are presented in chapter Statistical Models.

12.2 PyMC: Bayesian Statistics and Monte Carlo Markov Modeling

PyMC is a python module that implements Bayesian statistical models and fitting algorithms, including Markov chain Monte Carlo. Its flexibility and extensibility make it applicable to a large suite of problems. Along with core sampling functionality, PyMC includes methods for summarizing output, plotting, goodness-of-fit and convergence diagnostics.

PyMC provides functionalities to make Bayesian analysis as painless as possible. Here is a short list of some of its features:

- Fits Bayesian statistical models with Markov chain Monte Carlo and other algorithms.
- Includes a large suite of well-documented statistical distributions.
- Uses NumPy for numerics wherever possible.
- Includes a module for modeling Gaussian processes.
- Sampling loops can be paused and tuned manually, or saved and restarted later.
- Creates summaries including tables and plots.
- Traces can be saved to the disk as plain text, Python pickles, SQLite or MySQL database, or hdf5 archives.
- Several convergence diagnostics are available.
- Extensible: easily incorporates custom step methods and unusual probability distributions.
- MCMC loops can be embedded in larger programs, and results can be analyzed with the full power of Python.

A very recommendable, free ebook on Bayesian methods, which also provides a very good introduction to *PyMC*, is Probabilistic Programming & Bayesian Methods for Hackers. Warmly recommended!

An introduction to Bayesian Statistics, and an example from the "Bayesian Methods for Hackers" book, are presented in chapter Bayesian Statistics.

12.3 scikit-learn

scikit-learn is arguably the most advanced open source machine learning package available. It provides simple and efficient tools for data mining and data analysis, covering supervised as well as unsupervised learning.

It provides tools for

- **Classification** Identifying to which set of categories a new observation belongs to.
- **Regression** Predicting a continuous value for a new example.
- **Clustering** Automatic grouping of similar objects into sets.
- **Dimensionality reduction** Reducing the number of random variables to consider.
- **Model selection** Comparing, validating and choosing parameters and models.
- **Preprocessing** Feature extraction and normalization.

12.4 Generalized Linear Models

This is not really a Python package, but rather a book. However, this book that Annette Dobson has written has made *Generalized Linear Models (GLM)* [Dobson and Barnett(2008)] understandable and accessible for me. While the book presents solutions for the models for R and Stata, I have developed Python solutions for almost all examples in the book (`https://github.com/thomas-haslwanter/dobson`).

Chapter 13

Statistical Models

There is a substantial difference in approach between *hypothesis tests* and *statistical modeling*. In the former case, you typically start out with a *null hypothesis*. Based on your question and your data, you then select the appropriate statistical test as well as the desired significance level, and either accept or reject the null hypothesis.

In contrast, statistical modeling is much more an interactive analysis of the data. Based on a first look at the data, you typically start out with selecting a statistical model that may describe your data. In the previous chapters, we have for example described the hypythesized linear relationship between data with the model

$$y = k * x + d.$$

You then

- determine the model parameters (e.g. k and d),
- assess the quality of the model (e.g. through the R^2-value, or another suitable parameter),
- and inspect the residuals, to check if your proposed model has missed essential features in the data.

If you are not happy with the quality of the model, or if you find during the inspection of the residuals that you have either outliers or need another model, you modify your proposed model and repeat this procedure until you are happy with the results. You see that in comparison to hypothesis tests, statistical modeling involves much more an interactive playing with the data.

13.1 Model language

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

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

$$y \sim x + a + b + a : b \tag{13.1}$$

This formula language is based on the notation introduced by Wilkinson and Rogsers ([Wilkinson and Rogers(1973)], and also used by S and R. The symbols in Table 13.1 are used on the right hand side to denote different interactions.

A complete set of the description is found under `http:\\patsy.readthedocs.org`.

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

Table 13.1: Formula syntax

13.1.1 Design Matrix

Definition

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

$$y = f(x, \epsilon) \tag{13.2}$$

In the case of a linear regression model, the model can be rewritten as in Eq 9.9:

$$y = X\beta + \epsilon,$$

For a simple linear regression and multiple regression, the corresponding Design Matrices are given in 9.6 and 10.2, respectively.

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

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n, \tag{13.3}$$

where ^{T} denotes the transpose, so that $x_i^T\beta$ is the inner product between the vectors $\mathbf{x}_i$ and $\boldsymbol{\beta}$.

Often these n equations are stacked together and written in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{13.4}$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}. \tag{13.5}$$

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

[1] This section has been taken from Wikipedia

a presumption that the value of one of the variables is caused by, or directly influenced by the other variables. Alternatively, there may be an operational reason to model one of the variables in terms of the others, in which case there need be no presumption of causality.

- $\mathbf{x}_i$ are called *regressors*, *exogenous variables*, *explanatory variables*, *covariates*, *input variables*, *predictor variables*, or *independent variables*, but not to be confused with *independent random variables*. The matrix $\mathbf{X}$ is sometimes called the *design matrix*.
  - Usually a constant is included as one of the regressors. For example we can take $x_{i1} = 1$ for $i = 1, ..., n$. The corresponding element of β is called the *intercept*. Many statistical inference procedures for linear models require an intercept to be present, so it is often included even if theoretical considerations suggest that its value should be zero.
  - Sometimes one of the regressors can be a non-linear function of another regressor or of the data, as in polynomial regression and segmented regression. The model remains linear as long as it is linear in the parameter vector β (see Eq. 9.8).
- $\boldsymbol{\beta}$ is a p-dimensional *parameter vector*. Its elements are also called *effects*, or *regression coefficients*. Statistical estimation and inference in linear regression focuses on β.
- ε_i is called the *residuals*, *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.13.3, we have a *simple linear regression*, corresponding to Eq.9.6. If $i > 1$ we talk about *multilinear regression* or *multiple linear regression* (see Eq. 10.2).

Examples

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

$$h_i = \beta_1 t_i + \beta_2 t_i^2 + \varepsilon_i, \tag{13.6}$$

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

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

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

$$y_{ij} = \mu_i + \epsilon_{ij} \tag{13.7}$$

which can be written

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{13.8}$$

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

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

$$y_{ij} = \mu + \tau_i + \epsilon_{ij} \tag{13.9}$$

with the constraint that τ_1 is zero.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu \\ \tau_2 \\ \tau_3 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \\ \epsilon_7 \end{bmatrix} \tag{13.10}$$

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

13.1.2 Example: Program Effectiveness

Code: "regSpector.py" (p 211): Estimation of a linear regression model with statsmodels, using the Spector and Mazzeo (1980) data set.

13.2 Linear Regression Analysis with Python

[2] 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:

[2] The following is based on the blog of Connor Johnson.

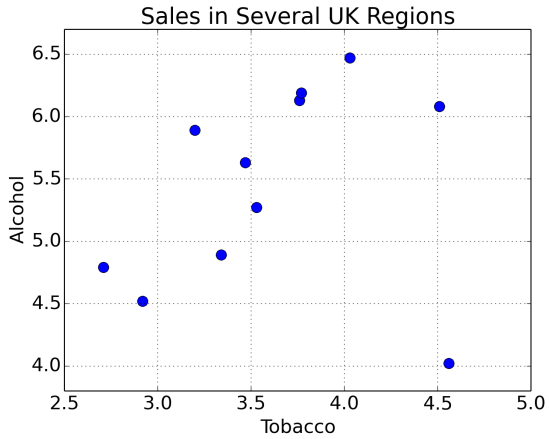


Figure 13.1: Sales of Alcohol vs Tobacco in the UK. We notice that there seems to be a linear trend, and one outlier, which corresponds to North Ireland.

```
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
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 (Df) 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.

13.2.1 Model Results

Definitions for Regression with Intercept

n is the number of observations, k 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.

- $SS_{\text{mod}} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$ is the *Model Sum of Squares*, or the sum of squares for the regression.
- $SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is the *Residuals Sum of Squares*, or the sum of squares for the errors.
- $SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$ is the *Total Sum of Squares*, and is equivalent to the sample variance multiplied by $n - 1$.

For multiple regression models, $SS_{\text{mod}} + SS_{\text{res}} = SS_{\text{tot}}$

- $DF_{\text{mod}} = k - 1$ is the *(Corrected) ModelDegrees of Freedom* . (The "-1" comes from the fact that we are only interested in the correlation, not in the absolute offset of the data.)
- $DF_{\text{res}} = n - k$ is the *Residuals Degrees of Freedom*
- $DF_{\text{tot}} = n - 1$ is the *(Corrected) Total Degrees of Freedom.* The Horizontal line regression is the null hypothesis model.

For multiple regression models with intercept, $DF_{\text{mod}} + DF_{\text{res}} = DF_{\text{tot}}$.

- $MS_{\text{mod}} = SS_{\text{mod}}/DF_{\text{mod}}$: *Model Mean of Squares*

- $MS_{\text{res}} = SS_{\text{res}}/DF_{\text{res}}$: *Residuals Mean of Squares.* MS_{res} is an unbiased estimate for σ^2 for multiple regression models.
- $MS_{\text{tot}} = SS_{\text{tot}}/DF_{\text{tot}}$: *Total Mean of Squares*, 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

$$R^2 = \frac{SS_{\text{mod}}}{SS_{\text{tot}}} \tag{13.11}$$

13.2.2 $\bar{R}^2$ - The *adjusted* R^2 Value

Many researchers prefer the *adjusted* $\bar{R}^2$ *value*, 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 $R^2 = 1 - SS_{\text{res}}/SS_{\text{tot}}$ or $1 - R^2 = SS_{\text{res}}/SS_{\text{tot}}$. To take into account the number of regression parameters p, define the *adjusted R-squared* value as

$$1 - \bar{R}^2 = \frac{ResidualVariance}{TotalVariance} \tag{13.12}$$

where *(Sample) Residual Variance* is estimated by $SS_{\text{res}}/DF_{\text{res}} = SS_{\text{res}}/(n-k)$, and *(Sample) Total Variance* is estimated by $SS_{\text{tot}}/DF_{\text{tot}} = SS_{\text{tot}}/(n-1)$. Thus,

$$\begin{aligned} 1 - \bar{R}^2 &= \frac{SS_{\text{res}}/(n-k)}{SS_{\text{tot}}/(n-1)} \\ &= \frac{SS_{\text{res}}}{SS_{\text{tot}}}\frac{n-1}{n-k} \end{aligned} \tag{13.13}$$

so

$$\begin{aligned} \bar{R}^2 &= 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}\frac{n-1}{n-k} \\ &= 1 - (1-R^2)\frac{n-1}{n-k} \end{aligned} \tag{13.14}$$

The F-test

If $t_1, t_2, ..., t_m$ 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,

$$Y_j = \alpha + \beta_1 X_{1j} + ... + \beta_n X_{nj} + \epsilon_i = \alpha + \sum_{i=1}^{n} \beta_i X_{ij} + \epsilon_j = E(Y_j|X) + \epsilon_j \tag{13.15}$$

we want to test the following null hypothesis and alternative hypothesis:
H_0: $\beta_1 = \beta_2 = ... = \beta_n = 0$
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 (i.e. all function values have the same variance), independent, and normal (see section 13.3):

1. SS_{res}/σ^2 has a χ^2 distribution with DF_{res} degrees of freedom.

2. SS_{mod}/σ^2 has a χ^2 distribution with DF_{mod} degrees of freedom.

3. SS_{res} and SS_{mod} 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 $F = \frac{u/n}{v/m}$ has an F distribution with (n, m) degrees of freedom.

If H0 is true,

$$F = \frac{(SS_{\text{mod}}/\sigma^2)/DF_{\text{mod}}}{(SS_{\text{res}}/\sigma^2)/DF_{\text{res}}} = \frac{SS_{\text{mod}}/DF_{\text{mod}}}{SS_{\text{res}}/DF_{\text{res}}} = \frac{MS_{\text{mod}}}{MS_{\text{res}}}, \tag{13.16}$$

has an F distribution with $(DF_{\text{mod}}, DF_{\text{res}})$ 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 *least 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

$$\epsilon = y_i - \sum_{k=1}^{n} \beta_k x_{ik} = y_i - \hat{y}_i \; in \; N(0, \sigma^2) \tag{13.17}$$

so the probability density is given by

$$p(\epsilon_i) = \Phi(\frac{y_i - \hat{y}_i}{\sigma}) \tag{13.18}$$

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

$$\Pi_{total} = \prod_{i=1}^{n} p(\epsilon_i) \tag{13.19}$$

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

$$E(\sigma^2) = \frac{SS_{\text{res}}}{n} \tag{13.20}$$

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 (section 13.2.2), 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

$$AIC = 2 * k - 2 * ln(\mathfrak{L}) \tag{13.21}$$

and the Schwartz or *Bayesian Information Criterion BIC* by

$$BIC = k * ln(N) - 2 * ln(\mathfrak{L}). \tag{13.22}$$

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 relate 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.

13.2.3 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

$$\beta = (X^T X)^{-1} X^T y. \tag{13.23}$$

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

$$C = cov(\beta) = \sigma^2 (X X^T)^{-1}. \tag{13.24}$$

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,

$$t_i = \beta_i / SE_{i,i} \tag{13.25}$$

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 weve 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,

$$CI = \beta_i \pm z \cdot SE_{i,i} \tag{13.26}$$

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)
```

13.2.4 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

$$S = \frac{\hat{\mu}_3}{\hat{\sigma}^3} = \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\right)^{3/2}}, \tag{13.27a}$$

$$K = \frac{\hat{\mu}_4}{\hat{\sigma}^4} = \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\right)^{2}} \tag{13.27b}$$

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, were looking at the distribution of the residual. If we obtain a very small value for Pr(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.

$$DW = \frac{\sum_{i=2}^{N}((y_i - \hat{y}_i) - (y_{i-1} - \hat{y}_{i-1}))^2}{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{13.28}$$

```
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 normalwhen it is in fact true.

$$JB = \frac{N}{6}\left(S^2 + \frac{1}{4}(K-3)^2\right) \tag{13.29}$$

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 well 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 (weak!), so we can sort of sleep okay.

13.2.5 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
==============================================================================
```

13.2.6 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. Therefore, in our case, well cast the DataFrame to 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 will 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 )

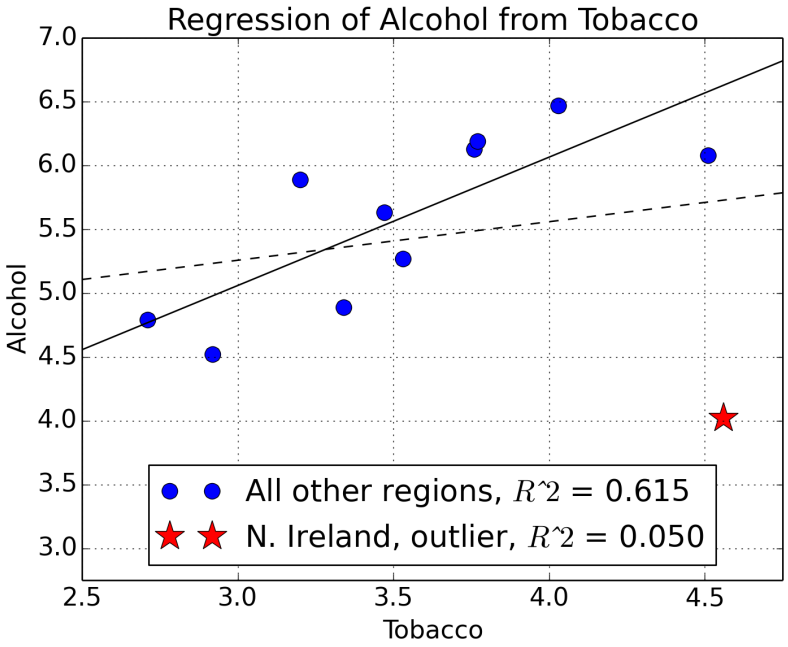
plt.hold(True)
plt.plot( df.Tobacco[-1:], df.Alcohol[-1:], 'r*', ms=20, lw=10,
    label='N. Ireland, outlier, $R 2$ = '+original_score)
```

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')
```

13.2.7 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 shouldnt dominate it. In most cases, your data will not align itself with the assumptions made by most of the available tests. Under http://www.nature.com/news/scientific-method-statistical-errors-1.14700 you can find a very interesting article from Nature on classical hypothesis testing. A more intuitive approach to hypothesis testing is Bayesian analysis.

13.3 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.

13.3.1 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 fixedthat is, the expected value of the partial derivative of y with respect to x_j. This is sometimes called the "unique effect" of x_j on "y". In contrast, the "marginal effect" of x_j on y can be assessed using a correlation coefficient or simple linear regression model relating x_j to y; this effect is the total derivative of y with respect to x_j.

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

It is possible that the unique effect can be nearly zero even when the marginal effect is large. This may imply that some other covariate captures all the information in x_j, so that once that

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

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

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

 **Code:** "modeling.py" (p 212) shows an example.

13.4 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.

 **Code:** "bootstrapDemo.py" (p 214): Example of bootstrapping the confidence interval for the mean of a sample distribution.

Chapter 14

Tests on Discrete Data

Data can be discrete for different reasons. On is that you acquired them in a discrete way (e.g. levels in a questionnaires.) 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.

14.1 Comparing Groups of Ranked 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* 7.2.3

Three or more groups When comparing two rank ordered groups, we can use the *Kruskal-Wallis test* 7.3.3

A more complex question arises when the requirement arises not only to compare two groups, but also to make quantitative predictions for ordinal data. For example, suppose you want to calculate the probability that a patient survives an operation, based on the amount of anesthetic he/she receives. The answer to this question involves statistical modelling, and the tool of *logistic regression*. If more than two ordinal (i.e. naturally ranked) levels are involved, the so-called *ordinal logistic regression* is used.

Hypothesis tests allow you to state quantitative probabilities on the likelihood of a hypothesis. *Linear Regression Modeling* allows you to make predictions and give confidence intervals for output variables that depend linearly on given inputs. But a large class of problems exceeds these requirements. For example, suppose you want to calculate the probability that a patient survives an operation, based on the amount of anesthetic he/she receives, and you want to find out how much anesthetic you can give the patient so that the chance of survival is at least 95%.

To cover such questions *Generalized Linear Models (GLMs)* have been introduced, which extend the technique of linear regression to a wide range of other problems. A general coverage of GLMs is beyond the goals of this book, and I would like to refer to the excellent book by Dobson [Dobson and Barnett(2008)]. While Dobson only gives solutions in *R* and *Stata*, I have made solutions in Python available for most of their problems (`https://github.com/thomas-haslwanter/dobson.git`).

In the following chapter I want to cover one commonly used case, *logistic regression*, and its extension to *ordinal logistic regression*. The Python solutions presented should allow the readers to solve similar problems on their own, and should give a brief insight in Generalized Linear Models.

14.2 Logistic Regression

So far we have been dealing with linear models, where a linear change on the input leads to a corresponding linear change on the output (Fig. 9.1):

$$y = k * x + d + \epsilon \tag{14.1}$$

However, for many applications this model is not suitable. Suppose we want to calculate the probability that a patient survives an operation, based on the amount of anesthetic he/she receives. This probability is bounded on both ends, since it has to be a value between 0 and 1.

We can achieve such a bounded relationship, though, if we don't use the output of Eq. 14.1 directly, but wrap it by another function:

$$p(x) = \frac{1}{1 + e^{\beta x + \alpha}} \tag{14.2}$$

In this model, the variable β that describes how quickly the function changes from 1 to 0, and α indicates the location of this change. This function is used frequently, and is called the *logistic function*.

14.2.1 Example: The Challenger Disaster

On January 28, 1986, the twenty-fifth flight of the U.S. space shuttle program ended in disaster when one of the rocket boosters of the Shuttle Challenger exploded shortly after lift-off, killing all seven crew members. The presidential commission on the accident concluded that it was caused by the failure of an O-ring in a field joint on the rocket booster, and that this failure was due to a faulty design that made the O-ring unacceptably sensitive to a number of factors including outside temperature. Of the previous 24 flights, data were available on failures of O-rings on 23, (one was lost at sea), and these data were discussed on the evening preceding the Challenger launch, but unfortunately only the data corresponding to the 7 flights on which there was a damage incident were considered important and these were thought to show no obvious trend. The data are shown below:

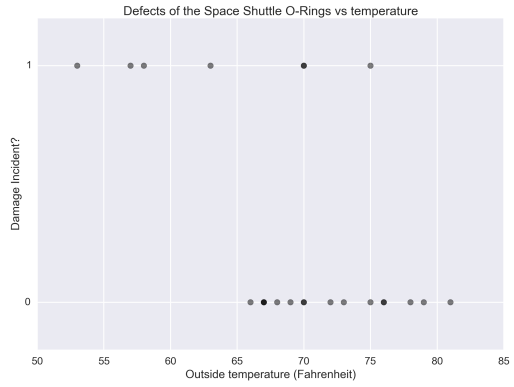


Figure 14.1: Failure of O-rings during space shuttle launches, as a function of temperature.

To simulate the probability of the O-rings failing, we can use the *logistic function*:

$$p(t) = \frac{1}{1 + e^{\beta t + \alpha}} \tag{14.3}$$

With a given p-value, the *binomial distribution* (section 5.2.5) determines the probability mass function for a given number of shuttle launches.

Listing 14.1: logitShort.py

```
import numpy as np
import os
import pandas as pd
from statsmodels.formula.api import glm
from statsmodels.genmod.families import Binomial

# Get the data
dataDir = r'..\Data\data_bayes'
fileName = 'challenger_data.csv'
inFile = os.path.join(dataDir, fileName)
challenger_data = np.genfromtxt(inFile, skip_header=1, usecols=[1, 2],
                                missing_values='NA', delimiter=',')
# Eliminate NaNs
challenger_data = challenger_data[~np.isnan(challenger_data[:, 1])]

# Create a dataframe, with suitable columns for the fit
df = pd.DataFrame()
df['temp'] = np.unique(challenger_data[:,0])
df['failed'] = 0
df['ok'] = 0
df['total'] = 0
df.index = df.temp.values

# Count the number of starts and failures
for ii in range(challenger_data.shape[0]):
    curTemp = challenger_data[ii,0]
    curVal  = challenger_data[ii,1]
    df.loc[curTemp,'total'] += 1
    if curVal == 1:
        df.loc[curTemp, 'failed'] += 1
    else:
        df.loc[curTemp, 'ok'] += 1

# fit the model

# --- >>> START stats <<< ---
model = glm('ok + failed ~ temp', data=df, family=Binomial()).fit()
# --- >>> STOP stats <<< ---

print(model.summary())
```

Code: "logit.py" (p 215) shows the full code for Fig. 14.2.

To summarize, we have three elements in our model

1. A probability distribution, which determines the probability of the outcome for a given trial (here the binomial distribution).
2. A linear model that relates the co-variates (here the temperature) to the variates (the failure/success of an O-ring).
3. A *link-function* that wraps the linear model to produce the parameter for the probability distribution.

14.3 Generalized Linear Models

The example above is an example of a *Generalized Linear Models (GLMS)*, a powerful tool for the analysis of wide range of statistical models. Here I will only describe the general principles. For details I refer to the excellent book by [Dobson and Barnett(2008)].

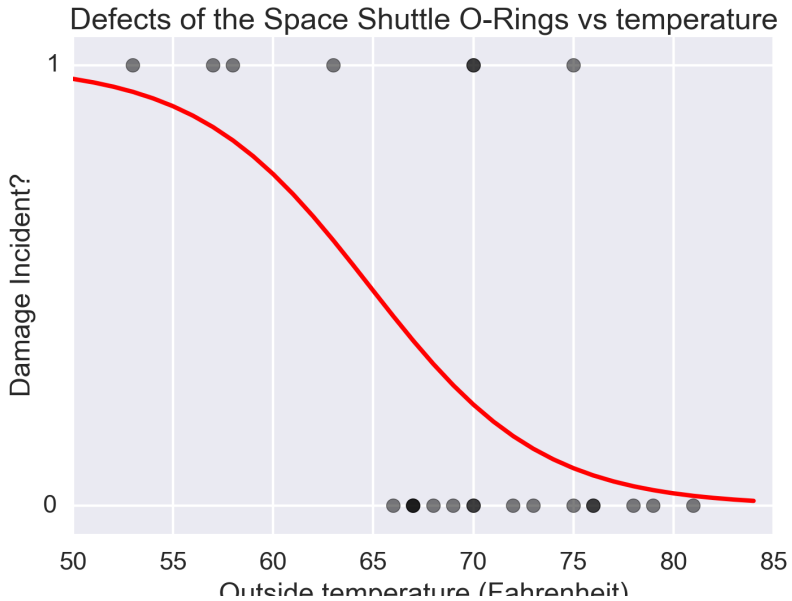


Figure 14.2: Probability for O-ring failure.

A GLM consists of three elements:

1. A probability distribution from the *exponential family*.
2. A linear predictor $\eta = X \cdot \beta$.
3. A link function g such that $E(Y) = \mu = g^{-1}(\eta)$.

14.3.1 Exponential Family of Distributions

The exponential family is a set of probability distributions of a certain form, specified below. This special form is chosen for mathematical convenience, on account of some useful algebraic properties, as well as for generality, as exponential families are in a sense very natural sets of distributions to consider. The exponential families include many of the most common distributions, including the normal, exponential, chi-squared, Bernoulli, Poisson distribution and many others. (A common distribution that is not from the exponential family is the T-distribution.)

In mathematical terms, an distribution from the exponential family has the general form

$$f_X(x|\theta) = h(x)g(\theta)\exp\left(\eta(\theta) \cdot T(x)\right) \tag{14.4}$$

where $T(x), h(x), g(\theta), \eta(\theta), and A(\theta)$ are known functions.

14.3.2 Linear Predictor and Link Function

The linear predictor for GLM is the same as the one used for *linear models*. The resulting terminology is unfortunately fairly confusing:

General Linear Models are models of the form $y = X\beta + \epsilon$, where ϵ is normally distributed (see Chapter **??**).

Generalized Linear Models encompass a much wider class of models, including all distributions from the exponential family *and* a link function.

The *link function* is an arbitrary function, with the only requirements that it is continuous and invertable.

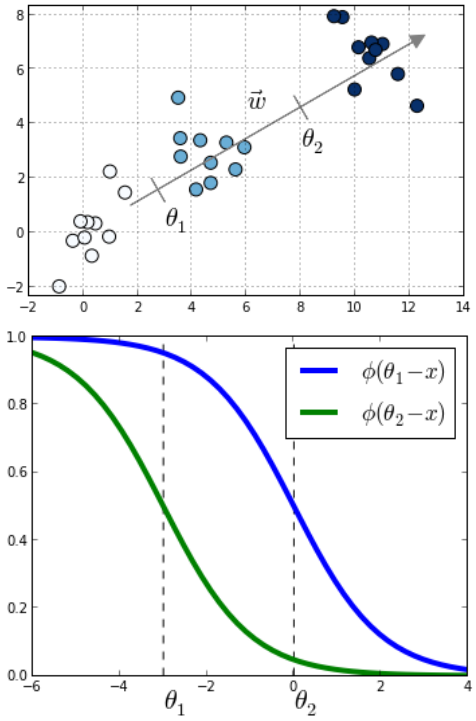


Figure 14.3: Toy example with three classes denoted in different colors. Also shown the vector of coefficients w and the thresholds θ_0 and θ_1. (Figure from Fabian Pedregosa, with permission)

14.4 Ordinal Logistic Regression

[1] The *logistic ordinal regression* model, also known as the proportional odds was introduced in the early 80s by McCullagh [McCullagh(1980), McCullagh and Nelder(1989)] and is a generalized linear model specially tailored for the case of predicting ordinal variables, that is, variables that are discrete (as in classification) but which can be ordered (as in regression). It can be seen as an extension of the logistic regression model to the ordinal setting.

Given $X \in \mathbb{R}^{n \times p}$ input data and $y \in \mathbb{N}^n$ target values. For simplicity we assume y is a non-decreasing vector, that is, $y_1 \leq y_2 \leq ...$ Just as the logistic regression models posterior probability $P(y = j|X_i)$ as the logistic function, in the logistic ordinal regression we model the *cumulative* probability as the logistic function. That is,

$$P(y \leq j|X_i) = \phi(\theta_j - w^T X_i) = \frac{1}{1 + \exp(w^T X_i - \theta_j)} \tag{14.5}$$

where w, θ are vectors to be estimated from the data and ϕ is the logistic function defined as $\phi(t) = \frac{1}{1+exp(t)}$.

Compared to multiclass logistic regression, we have added the constrain that the hyperplanes that separate the different classes are *parallel* for all classes, that is, the vector w is common across classes. To decide to which class will X_i be predicted we make use of the vector of thresholds θ. If there are K different classes, θ is a non-decreasing vector (that is, $\theta_1 \leq \theta_2 \leq ... \leq \theta_{K-1}$ of size $K1$. We will then assign the class j if the prediction $w^T X$ (recall that it's a linear model) lies in the interval $[\theta_{j1}, \theta_j[$. In order to keep the same definition for extremal classes, we define $\theta_0 = -\infty$ and $\theta_0 = +\infty$.

The intuition is that we are seeking a vector w such that Xw produces a set of values that are well separated into the different classes by the different thresholds θ. We choose a

[1] This section has been taken with permission from Fabian Pedregosa's blog on ordinal logistic regression, http://fa.bianp.net/blog/2013/logistic-ordinal-regression/

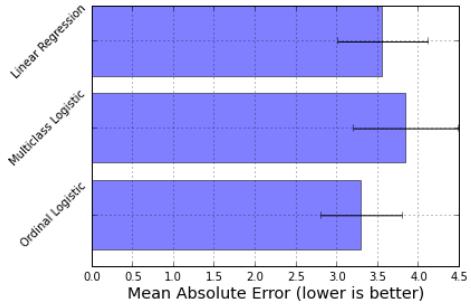


Figure 14.4: (Figure from Fabian Pedregosa, with permission)

logistic function to model the probability $P(y \leq j|X_i)$ but other choices are possible. In the proportional hazards model [McCullagh(1980)] the probability is modeled as $-\log(1 - P(y \leq j|X_i)) = \exp(\theta_j - w^T X_i)$. Other link functions are possible, where the link function satisfies $\text{link}(P(y \leq j|X_i)) = \theta_j - w^T X_i$. Under this framework, the logistic ordinal regression model has a logistic link function and the proportional hazards model has a log-log link function.

The logistic ordinal regression model is also known as the proportional odds model, because the ratio of corresponding odds for two different samples X_1 and X_2 is $\exp(w^T(X_1 - X_2))$ and so does not depend on the class j but only on the difference between the samples X_1 and X_2.

14.4.1 Optimization

Model estimation can be posed as an optimization problem. Here, we minimize the loss function for the model, defined as minus the log-likelihood:

$$\mathcal{L}(w, \theta) = -\sum_{i=1}^{n} \log(\phi(\theta_{y_i} - w^T X_i) - \phi(\theta_{y_i - 1} - w^T X_i)) \tag{14.6}$$

In this sum all terms are convex on w, thus the loss function is convex over w. It might be also jointly convex over w and θ, although I haven't checked. I use the function `fmin_slsqp` in `scipy.optimize` to optimize $\mathcal{L}$ under the constraint that θ is a non-decreasing vector. There might be better options, I don't know. If you do know, please leave a comment!.

Using the formula $\log(\phi(t))' = (1 - \phi(t))$, we can compute the gradient of the loss function as

$$\begin{aligned}
\nabla_w \mathcal{L}(w, \theta) &= \sum_{i=1}^{n} X_i(1 - \phi(\theta_{y_i} - w^T X_i) - \phi(\theta_{y_i - 1} - w^T X_i)) \\
\nabla_\theta \mathcal{L}(w, \theta) &= \sum_{i=1}^{n} e_{y_i} \left(1 - \phi(\theta_{y_i} - w^T X_i) - \frac{1}{1 - \exp(\theta_{y_i - 1} - \theta_{y_i})}\right) \\
&+ e_{y_i - 1} \left(1 - \phi(\theta_{y_i - 1} - w^T X_i) - \frac{1}{1 - \exp(-(\theta_{y_i - 1} - \theta_{y_i}))}\right)
\end{aligned}$$

where e_i is the ith canonical vector.

14.4.2 Code

I've implemented a Python version of this algorithm using Scipy's `optimize.fmin_slsqp` function. This takes as arguments the loss function, the gradient denoted before and a function that is ¿ 0 when the inequalities on are satisfied.

Code can be found here as part of the minirank package, which is my sandbox for code related to ranking and ordinal regression. At some point I would like to submit it to scikit-learn but right now the I don't know how the code will scale to medium-scale problems, but I suspect not great. On top of that I'm not sure if there is a real demand of these models for scikit-learn and I don't want to bloat the package with unused features. Performance

I compared the prediction accuracy of this model in the sense of mean absolute error (IPython notebook) on the boston house-prices dataset. To have an ordinal variable, I rounded the values to the closest integer, which gave me a problem of size 506 13 with 46 different target values. Although not a huge increase in accuracy, this model did give me better results on this particular dataset:

Here, ordinal logistic regression is the best-performing model, followed by a Linear Regression model and a One-versus-All Logistic regression model as implemented in scikit-learn.

 **Code:** "ologit.py" (p 217) corresponding Code by Fabian Pedregosa

Chapter 15

Bayesian Statistics

15.1 Bayesian vs. Frequentist Interpretation

Calculating probabilities is only one part of statistics. Another is the interpretation of them - and the consequences that come with different interpretations.

So far we have restricted ourselves to the *frequentist interpretation*, which interprets p as the *frequency of an occurrence*: if an outcome of an experiment has the probability p, it means that if that experiment is repeated N times (where N is a large number), then we observe this specific outcome $N * p$ times.

The *Bayesian interpretation* of p is quite different, and interprets p as our *believe of the likelihood* of a certain outcome. For some events, this makes a lot more sense. For example, in the upcoming semi-final of the soccer worldcup in Brazil, Argentine will play against the Netherlands, with Lionel Messi leading the Argentinian team. Since Messi is (for a soccer player) already fairly old, this may be a one-time event, and we will never have a large number N repetitions.

But in addition to this difference in interpretation, the Bayesian approach has another advantage: it lets us bring in *prior knowledge* into the calculation of the probability p, through the application of *Bayes' Theorem*:

In its most common form, it is:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}. \tag{15.1}$$

In the Bayesian interpretation, probability measures a degree of belief. Bayes's theorem then links the degree of belief in a proposition before and after accounting for evidence. For example, suppose it is believed with 50% certainty that a coin is twice as likely to land heads than tails. If the coin is flipped a number of times and the outcomes observed, that degree of belief may rise, fall or remain the same depending on the results.

John Maynard Keynes, a great economist and thinker, said "When the facts change, I change my mind. What do you do, sir?" This quote reflects the way a Bayesian updates his or her beliefs after seeing evidence.

For proposition A and evidence B,

- P(A), the *prior probabiliby*, is the initial degree of belief in A.
- $P(A|B)$, the *posterior probability*, is the degree of belief having accounted for B. It can be read as *"the probability of A, given that B is the case"*.
- the quotient $P(B|A)/P(B)$ represents the support B provides for A.

If the number of available data points is large, the difference in interpretation does typically not change the result significantly. If the number of data points is small, however, the possibility to bring in external knowledge may lead to a significantly improved estimation of p.

15.1.1 Bayesian Example

[1]Suppose a man told you he had a nice conversation with someone on the train. Not knowing anything about this conversation, the probability that he was speaking to a woman is 50% (assuming the speaker was as likely to strike up a conversation with a man as with a woman). Now suppose he also told you that his conversational partner had long hair. It is now more likely he was speaking to a woman, since women are more likely to have long hair than men. Bayes's theorem can be used to calculate the probability that the person was a woman.

To see how this is done, let W represent the event that the conversation was held with a woman, and L denote the event that the conversation was held with a long-haired person. It can be assumed that women constitute half the population for this example. So, not knowing anything else, the probability that W occurs is P(W) = 0.5.

Suppose it is also known that 75% of women have long hair, which we denote as $P(L|W) = 0.75$ (read: the probability of event L given event W is 0.75, meaning that the probability of a person having long hair (event "L"), given that we already know that the person is a woman ("event W") is 75%). Likewise, suppose it is known that 15% of men have long hair, or $P(L|M) = 0.15$, where M is the complementary event of W, i.e., the event that the conversation was held with a man (assuming that every human is either a man or a woman).

Our goal is to calculate the probability that the conversation was held with a woman, given the fact that the person had long hair, or, in our notation, $P(W|L)$. Using the formula for Bayes's theorem, we have:

$$P(W|L) = \frac{P(L|W)P(W)}{P(L)} = \frac{P(L|W)P(W)}{P(L|W)P(W) + P(L|M)P(M)}$$

where we have used the law of total probability to expand P(L). The numeric answer can be obtained by substituting the above values into this formula (the algebraic multiplication is annotated using "$\cdot$", the centered dot). This yields

$$P(W|L) = \frac{0.75 \cdot 0.50}{0.75 \cdot 0.50 + 0.15 \cdot 0.50} = \frac{5}{6} \approx 0.83,$$

i.e., the probability that the conversation was held with a woman, given that the person had long hair, is about 83%.

Another way to do this calculation is as follows. Initially, it is equally likely that the conversation is held with a woman as with a man, so the prior odds are 1:1. The respective chances that a man and a woman have long hair are 15% and 75%. It is 5 times more likely that a woman has long hair than that a man has long hair. We say that the likelihood ratio or Bayes factor is 5:1. Bayes's theorem in odds form, also known as Bayes's rule, tells us that the posterior odds that the person was a woman is also 5:1 (the prior odds, 1:1, times the likelihood ratio, 5:1). In a formula:

$$\frac{P(W|L)}{P(M|L)} = \frac{P(W)}{P(M)} \cdot \frac{P(L|W)}{P(L|M)}.$$

15.2 The Bayesian Approach in the Age of Computers

Bayes's theorem was named after the Reverend Thomas Bayes (1701 - 1761), who studied how to compute a distribution for the probability parameter of a binomial distribution. So it has

[1]This example, and the above definition of Bayes' Theorem, has been taken from Wikipedia

been around for a long time. The reason Bayes' Theorem has become so popular in statistics in recent years is the cheap availability of massive computational power. This allows the empirical calculation of posterior probabilities, one-by-one, for each new piece of evidence. This, combined with statistical approaches like *Markov Chain Monte Carlo (MCMC)* simulations, has allowed radically new statistical analysis procedures, and has led to what may be called "statistical trench warfare" between the followers of the different philosophies. If you don't believe me, check the corresponding discussions on the WWW.

For more information on that topic, check out (in order of rising complexity)

- Wikipedia, which has some nice explanations under *"Bayes ..."*
- Bayesian Methods for Hackers, a nice, free ebook, providing a practical introduction to the use of PyMC (see below).
- The PyMC User Guide: PyMC is a very powerful Python package which makes the application of MCMC techniques very simple.
- *Pattern Recognition and Machine Learning*, a comprehensive, but often quite technical book by Christopher M. Bishop [Bishop(2007)].

15.3 Example: The Challenger Disaster

This is an excerpt of the excellent 'Bayesian Methods for Hackers'. For the whole book, check out Bayesian Methods for Hackers.

On January 28, 1986, the twenty-fifth flight of the U.S. space shuttle program ended in disaster when one of the rocket boosters of the Shuttle Challenger exploded shortly after lift-off, killing all seven crew members. The presidential commission on the accident concluded that it was caused by the failure of an O-ring in a field joint on the rocket booster, and that this failure was due to a faulty design that made the O-ring unacceptably sensitive to a number of factors including outside temperature. Of the previous 24 flights, data were available on failures of O-rings on 23, (one was lost at sea), and these data were discussed on the evening preceding the Challenger launch, but unfortunately only the data corresponding to the 7 flights on which there was a damage incident were considered important and these were thought to show no obvious trend. The data are shown below:

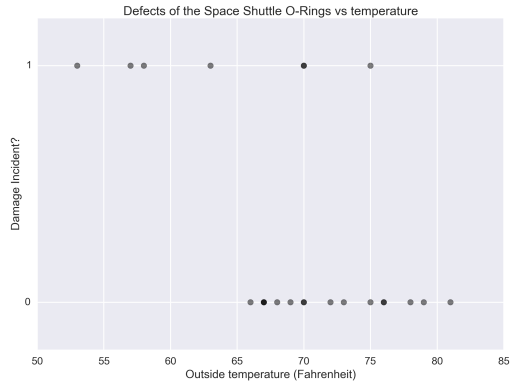


Figure 15.1: Failure of O-rings during space shuttle launches, as a function of temperature.

To simulate the probability of the O-rings failing, we need a function that goes from one to zero. One of the most frequently used functions for that is the *logistic function*:

$$p(t) = \frac{1}{1 + e^{\beta t + \alpha}} \tag{15.2}$$

In this model, the variable β that describes how quickly the function changes from 1 to 0, and α indicates the location of this change.

Using the Python package PyMC, a Monte-Carlo simulation of this model can be done remarkably easily:

```
# --- Perform the MCMC-simulations ---
temperature = challenger_data[:, 0]
D = challenger_data[:, 1]  # defect or not?

# Define the prior distributions for alpha and beta
# 'value' sets the start parameter for the simulation
# The second parameter for the normal distributions is the "precision",
# i.e. the inverse of the standard deviation
beta = pm.Normal("beta", 0, 0.001, value=0)
alpha = pm.Normal("alpha", 0, 0.001, value=0)

# Define the model-function for the temperature
@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta * t + alpha))

# connect the probabilities in `p` with our observations through a
# Bernoulli random variable.
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)

# Combine the values to a model
model = pm.Model([observed, beta, alpha])

# Perform the simulations
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)
```

From this simulation, we obtain not only our best estimate for α and β, but also information about our uncertainty about these values:

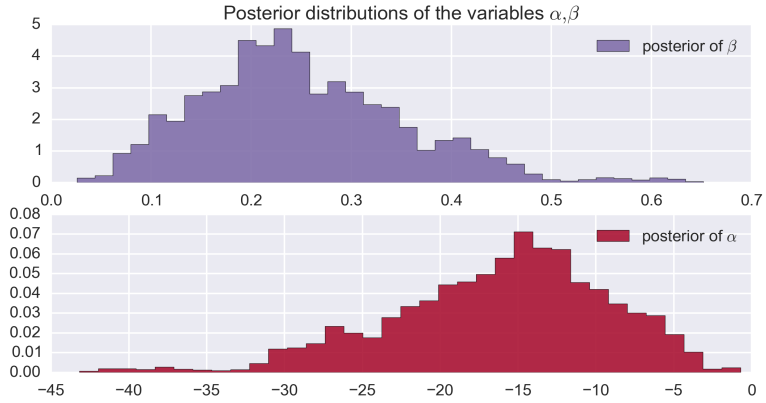


Figure 15.2: Probabilities for alpha and beta, from the MCMC simulations.

The probability curve for an O-ring failure thus looks as follows:

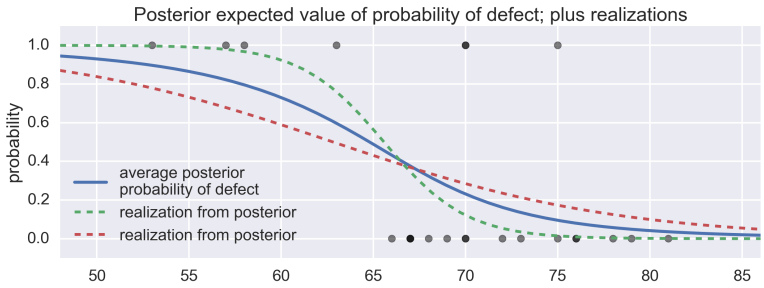


Figure 15.3: Probility for an O-ring failure, as a function of temperature.

One advantage of the MCMC simulation is, that it also provides confidence intervals for the probability:

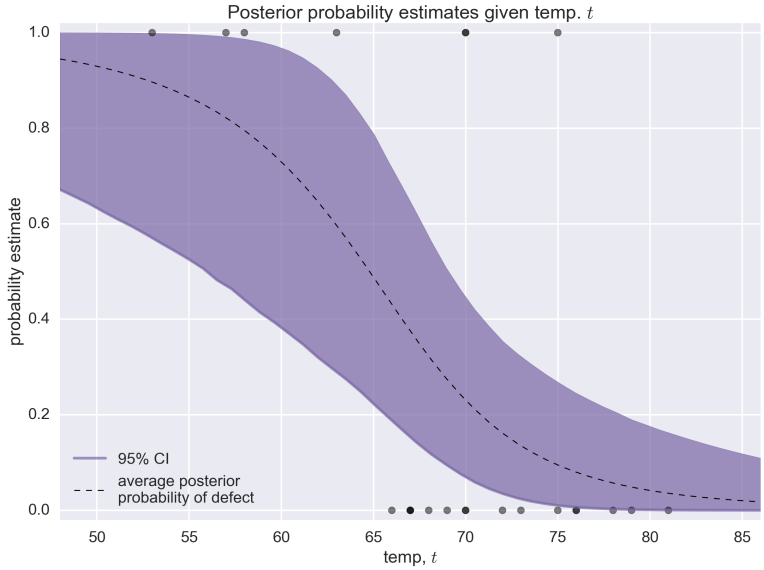


Figure 15.4: 95% Confidence Intervals for the Probility for an O-ring failure.

On the day of the Challenger disaster, the outside temperature was 31 degrees Fahrenheit. The posterior distribution of a defect occurring, given this temperature, almost guaranteed that the Challenger was going to be subject to defective O-rings.

Code: "challenger.py" (p 225): Full implementation of the MCMC simulation.

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 exectution
    (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 shotcut: "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 ad "DataFrames" and "ordinary-least-squares" or "ols-models".
    import pandas
    myDict = {'x':tHigh, 'y':xHigh}
    df = pandas.DataFrame(myDict)
    model = pandas.ols(y=df['y'], x=df['x'])
    print(model)
    #raw_input('These are the summary results from Pandas - Hit any key to
        continue')


if __name__=='__main__':
```

```
main()    # Execute the main function
```

Listing A.2: interactivePlots.py

```
# Source: http://scipy-central.org/item/84/1/simple-interactive-matplotlib-plots

'''Interactive graphs with Matplotlib have haunted me. So here I have collected
    a number of
tricks that should make interactive use of plots simpler. The functions below
    show how to

- Position figures on the screen (e.g. top left half of display)
- Pause and proceed automatically after a few sec
- Proceed on a click, or a keyboard hit
- Evaluate keyboard inputs

author: Thomas Haslwanter
date:   March-2015
ver:    1.0
license: Creative Commons Zero (almost public domain) http://scpyce.org/cc0

'''
import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk

t = np.arange(0,10,0.1)
c = np.cos(t)
s = np.sin(t)

def normalPlot():
    '''Just show a plot. The progam stops, and only continues when the plot is
        closed,
    either by hitting the "Window Close" button, or by typing "ALT+F4". '''

    plt.plot(t,s)
    plt.title('Normal plot: you have to close it to continue\nby clicking the "
        Window Close" button, or by hitting "ALT+F4"')
    plt.show()

def positionOnScreen():
    '''Position two plots on your screen. This uses the Tickle-backend, which I
        think is the default on all platforms.'''
    # Get the screen size
    root = tk.Tk()
    (screen_w, screen_h) = (root.winfo_screenwidth(), root.winfo_screenheight())
    root.destroy()

    # The program continues after the first plot
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(t,c)
    ax.set_title('Top Left: Close this one last')

    # Position the first graph in the top-left half of the screen
    position_topLeft = '{0}x{1}+{2}+{3}'.format(screen_w//2, screen_h//2,0,0)
    fig.canvas.manager.window.geometry(position_topLeft)

    # Put another graph in the top right half
    fig2 = plt.figure()
    ax2 = fig2.add_subplot(111)
    ax2.plot(t,s)
    # I don't completely understand why this one has to be closed first. But
        otherwise the program gets unstable.
    ax2.set_title('Top Right: Close this one first (e.g. with ALT+F4)')
```

```
    position_topRight = '{0}x{1}+{2}+{3}'.format(screen_w//2, screen_h//2,
        screen_w//2,0)
    fig2.canvas.manager.window.geometry(position_topRight)

    plt.show()

def showAndPause():
    '''Show a plot only for 2 seconds, and then proceed automatically'''
    plt.plot(t,s)
    plt.title('Don''t touch! I will proceed automatically.')

    plt.show(block=False)
    duration = 2    # [sec]
    plt.pause(duration)
    plt.close()

def waitForInput():
    ''' This time, proceed with a click or by hitting any key '''
    plt.plot(t,c)
    plt.title('Click in that window, or hit any key to continue')

    plt.waitforbuttonpress()
    plt.close()

def keySelection():
    '''Wait for user intput, and proceed depending on the key entered.
    This is a bit complex. But None of the versions I tried without
    key binding were completely stable.'''

    fig, ax = plt.subplots()
    fig.canvas.mpl_connect('key_press_event', on_key_event)

    # Disable default Matplotlib shortcut keys:
    keymaps = [param for param in plt.rcParams if param.find('keymap') >= 0]
    for key in keymaps:
        plt.rcParams[key] = ''

    ax.plot(t,c)
    ax.set_title('First, enter a vowel:')
    plt.show()

def on_key_event(event):
    '''Keyboard interaction'''

    #print('you pressed %s'%event.key)
    key = event.key

    # In Python 2.x, the key gets indicated as "alt+[key]"
    # Bypass this bug:
    if key.find('alt') == 0:
        key = key.split('+')[1]

    curAxis = plt.gca()
    if key in 'aeiou':
        curAxis.set_title('Well done!')
        plt.pause(1)
        plt.close()
    else:
        curAxis.set_title(key + ' is not a vowel: try again to find a vowel ....
            ')
        plt.draw()

if __name__ == '__main__':
```

```
normalPlot()
positionOnScreen()
showAndPause()
waitForInput()
keySelection()
```

Listing A.3: 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.4: 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 reproducable 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.github.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.5: getdata.py

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

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

#Author:  Thomas Haslwanter, June-2014

from os.path import join
from numpy import genfromtxt
import os
import sys
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.github.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.6: 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
- errorbars
- boxplots
- violinplots
- cumulative density functions

'''

# author: Thomas Haslwanter, date: Feb-2015

# 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

import mystyle   # custom module to set fontsize, etc

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

    # Set the fonts the way I like them
    sns.set_context('poster')
    sns.set_style('ticks')
    #mystyle.set()

    # Scatter plot
    scatter(arange(len(x)), x)
    xlim([0, len(x)])
    mystyle.printout('scatterPlot.png', xlabel='x', ylabel='y', title='Scatter')

    # Histogram
    hist(x)
    mystyle.printout('histogram_plain.png', xlabel='Data Values', ylabel='
        Frequency', title='Histogram, default settings')

    hist(x,25)
    mystyle.printout('histogram.png', xlabel='Data Values', ylabel='Frequency',
        title='Histogram, 25 bins')

    # Cumulative probability density
    numbins = 20
    plot(stats.cumfreq(x,numbins)[0])
    mystyle.printout('CumulativeFrequencyFunction.png', xlabel='Data Values',
        ylabel='Cumulative Frequency')

    # Boxplot
    # The ox consists of the first, second (middle) and third quartile
    boxplot(x, sym='*')
```

```
    mystyle.printout('boxplot.png', xlabel='Values', title='Boxplot')

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

    # Errorbars
    x = arange(5)
    y = x**2
    errorBar = x/2
    errorbar(x,y, yerr=errorBar, fmt='o', capsize=5, capthick=3)
    xlim([-0.2, 4.2])
    ylim([-0.2, 19])
    mystyle.printout('Errorbars.png', xlabel='Data Values', ylabel='Measurements
        ', title='Errorbars')

    # Violinplot
    nd = stats.norm
    data = nd.rvs(size=(100))

    nd2 = stats.norm(loc = 3, scale = 1.5)
    data2 = nd2.rvs(size=(100))

    # Use pandas and the seaborn package for the violin plot
    df = pd.DataFrame({'Girls':data, 'Boys':data2})
    #sns.violinplot(df, color = ["#999999", "#DDDDDD"])
    sns.violinplot(df)

    mystyle.printout('violinplot.png')

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

Listing A.7: fig_centralLimitTheorem.py

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

'''

# author: Thomas Haslwanter, date: July-2014

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

sns.set(context='poster', style='ticks')

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)
    #mystyle.set(14)
    #sns.set_context('paper')
    #sns.set_style('whitegrid')

    axs[0].hist(data,bins=nbins)
    axs[0].set_title('Random data')
    axs[0].set_xticks([0, 0.5, 1])
    axs[0].set_ylabel('Counts')

    axs[1].hist( np.mean(data.reshape((ndata/2,2)),  axis=1), bins=nbins)
    axs[1].set_xticks([0, 0.5, 1])
    axs[1].set_title(' Average over 2')

    axs[2].hist( np.mean(data.reshape((ndata/10,10)),axis=1), bins=nbins)
    axs[2].set_xticks([0, 0.5, 1])
    axs[2].set_title(' Average over 10')

    plt.tight_layout()
    mystyle.printout_plain('CentralLimitTheorem.png')

    plt.show()

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

Listing A.8: 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: July-2014

import numpy as np

import scipy.stats as stats
import matplotlib.pyplot as plt
from matplotlib.mlab import frange
import os
import seaborn as sns

myMean = 0
mySD = 3
x = np.arange(-5,15,0.1)
outDir = r'..\Images'

sns.set(context='poster', style='ticks', palette='deep')

def simple_normal():
    ''' Different aspects of a normal distribution'''
    # Generate the data
    x = np.arange(-4,4,0.1) # generate the desirded 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.gca().xaxis.set_ticks_position('bottom')
    plt.gca().yaxis.set_ticks_position('left')
    plt.yticks(np.linspace(0, 0.4, 5))
    plt.title('Normal Distribution - PDF')

    plt.subplot(323)
    plt.plot(x,nd.cdf(x))
    plt.gca().xaxis.set_ticks_position('bottom')
    plt.gca().yaxis.set_ticks_position('left')
    plt.xlim([-4,4])
    plt.ylim([0,1])
    plt.vlines(0, 0, 1, linestyles='--')
    plt.title('CDF: cumulative distribution fct')

    plt.subplot(324)
    plt.plot(x,nd.sf(x))
    plt.gca().xaxis.set_ticks_position('bottom')
    plt.gca().yaxis.set_ticks_position('left')
    plt.xlim([-4,4])
    plt.ylim([0,1])
    plt.vlines(0, 0, 1, linestyles='--')
    plt.title('SF: survival fct')
```

```
    plt.subplot(325)
    plt.plot(x2,nd.ppf(x2))
    plt.gca().xaxis.set_ticks_position('bottom')
    plt.gca().yaxis.set_ticks_position('left')
    plt.yticks(np.linspace(-4,4,5))
    plt.hlines(0, 0, 1, linestyles='--')
    plt.ylim([-4,4])
    plt.title('PPF')

    plt.subplot(326)
    plt.plot(x2,nd.isf(x2))
    plt.gca().xaxis.set_ticks_position('bottom')
    plt.gca().yaxis.set_ticks_position('left')
    plt.yticks(np.linspace(-4,4,5))
    plt.hlines(0, 0, 1, linestyles='--')
    plt.title('ISF')
    plt.ylim([-4,4])
    plt.tight_layout()

    outFile = 'DistributionFunctions.png'

    saveTo = os.path.join(outDir, outFile)
    plt.savefig(saveTo, dpi=200)

    print('OutDir: {0}'.format(outDir))
    print('Figure saved to {0}'.format(outFile))

    plt.show()

def shifted_normal():
    '''PDF, scatter plot, and histogram.'''
    # Generate the data
    # Plot a normal distribution: "Probability density functions"
    myMean = [0,0,0,-2]
    mySD2 = [0.2,1,5,0.5]
    t = frange(-5,5,0.02)
    sns.set_palette('husl', 4)
    for mu,sigma in zip(myMean, np.sqrt(mySD2)):
        y = stats.norm.pdf(t, mu, sigma)
        plt.plot(t,y, label='$\mu={0}, \; \t\sigma={1:3.1f}$'.format(mu,sigma))
    plt.legend()
    plt.xlim([-5,5])
    plt.title('Normal Distributions')
    outFile = 'Normal_Distribution_PDF.png'

    saveTo = os.path.join(outDir, outFile)
    plt.savefig(saveTo, dpi=200)

    print('OutDir: {0}'.format(outDir))
    print('Figure saved to {0}'.format(outFile))
    plt.show()

    # Generate random numbers with a normal distribution
    myMean = 0
    mySD = 3
    numData = 500
    data = stats.norm.rvs(myMean, mySD, size = numData)
    plt.scatter(np.arange(len(data)), data)
    plt.title('Normally distributed data')
    plt.xlim([0,500])
    plt.ylim([-10,10])
    plt.show()
```

```
        plt.close()

        #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
        myMean = 0
        mySD = 1
        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_xlim([-3, 3])
                plt.gca().set_xticks(())
                plt.gca().set_yticks(())
                plt.gca().set_xticklabels(())
                plt.gca().set_yticklabels(())

        plt.tight_layout()

        outFile = 'Normal_MultHist.png'

        saveTo = os.path.join(outDir, outFile)
        plt.savefig(saveTo, dpi=200)

        print('OutDir: {0}'.format(outDir))
        print('Figure saved to {0}'.format(outFile))
        plt.show()
        plt.close()

        # Check out the mean of 1000 normally distributded 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()
    many_normals()
    sns.set(font_scale=1.5, style='white')
    values_fromCDF()
    shifted_normal()
```

Listing A.9: 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: July-2014

# Note: here I use the iPython approach, which is best suited for interactive
    work
from pylab import *
from scipy import stats
import mystyle   # custom module to set fontsize, etc

#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__':
    mystyle.set()
    show_continuous()
```

Listing A.10: Discrete

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

'''

# author: Thomas Haslwanter, date: Feb-2015

# 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
import os
#import seaborn as sns
import mystyle

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

    ns = [20,20,40]
    ps = [0.5, 0.7, 0.5]

    #markersize = 8
    for (p,n) in zip(ps, ns):
        bd = stats.binom(n,p)
        x = np.arange(n+1)
        plt.plot(x, bd.pmf(x), 'o--', label='p={0:3.1f}, n={1}'.format(p,n))

    plt.legend()
    #sns.set_context('poster')
    #sns.set_style('ticks')
    #mystyle.set(14)

    plt.title('Binomial distribuition')
    plt.xlabel('X')
    plt.ylabel('P(X)')
    #sns.despine()
    plt.annotate('Upper Limit', xy=(20,0), xytext=(27,0.04),
                 arrowprops=dict(shrink=0.05))


    mystyle.printout_plain('Binomial_distribution_pmf.png')
    plt.show()


#-----------------------------------------------------------------------
def show_poisson():
    """Show an example of poisson distributions"""

    lambdas = [1,4,10]

    k = np.arange(20)
    markersize = 8
    for par in lambdas:
        plt.plot(k, stats.poisson.pmf(k, par), 'o--', label='$\lambda={0}$'.
            format(par))

    plt.legend()
    #sns.set_context('poster')
    #sns.set_style('ticks')
    #mystyle.set(14)
```

```
    plt.title('Poisson distribuition')
    plt.xlabel('X')
    plt.ylabel('P(X)')
    #sns.despine()

    mystyle.printout_plain('Poisson_distribution_pmf.png')

    plt.show()

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

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

    k = np.arange(25)
    pd = stats.poisson(10)
    mystyle.set(12)

    ax[0].plot(k, pd.pmf(k),'x-')
    ax[0].set_title('Poisson distribition')
    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()
    show_poisson_views()
```

Listing A.11: checkNormality.py

```
'''
Graphical and quantitative check, if a given distribution is normal.
- For small sample-numbers (<50), you should use the Shapiro-Wilk test or the "
    normaltest"
- for intermediate sample numbers, the Lilliefors-test is good since the
    original KS-test is unreliable when mean and std of the distribution
are not known.
- the Kolmogorov-Smirnov(KS) test should only be used for large sample numbers
    (>300)

'''

# author: Thomas Haslwanter, date: May-2014

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

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 reproducable 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()

    pVals = pd.Series()
    # The scipy normaltest is based on D-Agostino and Pearsons test that
    # combines skew and kurtosis to produce an omnibus test of normality.
    _, pVals['omnibus'] = stats.normaltest(data)

    # Shapiro-Wilk test
    _, pVals['Shapiro-Wilk'] = stats.shapiro(data)

    # Or you can check for normality with Lilliefors-test
    ksStats, pVals['Lilliefors'] = kstest_normal(data)

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

    print(pVals)
    if pVals['omnibus'] > 0.05:
        print('Data are normally distributed')
    # --- >>> STOP stats <<< ---
```

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

Listing A.12: 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.13: 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: Jun-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.ppf(0.975)
    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-tes
    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(checkVal, loc=mmean,
            scale=mstd/np.sqrt(len(data)))*2

    # compare
    print(('The probability from the t-test is ' + '{0:5.4f}, and from the
        normal distribution {1:5.4f}'.format(tProb, normProb)))

    return normProb # should be 0.054201154690070759

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

Listing A.14: 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(("Wilcoxon-Signed-Rank-Sum 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(("Mann-Whitney 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.15: 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
        dirng 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 form 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-forman 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.16: 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.

This dataset is taken from an R-tutorial, and contains a hypothetical sample
of 30 participants who are divided into three stress reduction treatment
groups (mental, physical, and medical). The values are represented on a
scale that ranges from 1 to 5. This dataset can be conceptualized as a
comparison between three stress treatment programs, one using mental
methods, one using physical training, and one using medication. The values
represent how effective the treatment programs were at reducing
participant's stress levels, with higher numbers indicating higher
effectiveness.

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

'''

# author: Thomas Haslwanter, date: June-2014

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.17: 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.18: 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.19: 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. What is the confidence interval
    for the prevalence of 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 delinquensts and non-delinquents who failed
        a vision test
    Spectecle wearers: 1 delinquent, 5 non-delinquents
    non-spectacle wearers: 8 delinquents, 2 non-delinquents'''

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

    # --- >>> 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 cochranQ():
    '''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.
    Is there a difference between the performance on the three tasks?
    '''

    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()
    cochranQ()
```

Listing A.20: 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 form
        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.21: 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.22: 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.23: 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 reproducable 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.24: 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.25: 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 dta
    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 reproducable 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.26: bootstrap.py

```
''' Example of bootstrapping the confidence interval for the mean of a sample
    distribution
This function requires "bootstrap.py", which is available from
https://github.com/cgevans/scikits-bootstrap

'''

# author: Thomas Haslwanter, date: Feb-2015

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

def generate_data():
    # To get reproducable 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 conficence intervals for the mean are: {0} - {1}'.format(*CIs)))

    return CIs

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

Listing A.27: logit.py

```
'''Logistic Regression
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-2015

import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import seaborn as sns
import mystyle
from statsmodels.formula.api import glm
from statsmodels.genmod.families import Binomial

def getData():
    '''Get the data '''

    dataDir = r'..\Data\data_bayes'
    fileName = 'challenger_data.csv'
    inFile = os.path.join(dataDir, fileName)

    data = np.genfromtxt(inFile, skip_header=1, usecols=[1, 2],
                                    missing_values='NA', delimiter=',')
    # Eliminate NaNs
    data = data[~np.isnan(data[:, 1])]

    return data

def prepareForFit(inData):
    ''' Make the temperature-values unique, and count the number of failures and
         successes
    Returns a DataFrame'''

    # Create a dataframe, with suitable columns for the fit
    df = pd.DataFrame()
    df['temp'] = np.unique(inData[:,0])
    df['failed'] = 0
    df['ok'] = 0
    df['total'] = 0
    df.index = df.temp.values

    # Count the number of starts and failures
    for ii in range(inData.shape[0]):
        curTemp = inData[ii,0]
        curVal  = inData[ii,1]
        df.loc[curTemp,'total'] += 1
        if curVal == 1:
            df.loc[curTemp, 'failed'] += 1
        else:
            df.loc[curTemp, 'ok'] += 1

    return df

def logistic(x, beta, alpha=0):
    ''' Logistic Function '''
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

def showResults(challenger_data, model):
    ''' Show the original data, and the resulting logit-fit'''
```

```
    # First plot the original data
    plt.figure()
    sns.set_context('poster')
    np.set_printoptions(precision=3, suppress=True)

    plt.scatter(challenger_data[:, 0], challenger_data[:, 1], s=75, color="k",
                alpha=0.5)
    plt.yticks([0, 1])
    plt.ylabel("Damage Incident?")
    plt.xlabel("Outside temperature (Fahrenheit)")
    plt.title("Defects of the Space Shuttle O-Rings vs temperature")
    plt.xlim(50, 85)

    # Plot the fit
    x = np.arange(50, 85)
    alpha = model.params[0]
    beta = model.params[1]
    y = logistic(x, beta, alpha)

    plt.hold(True)
    plt.plot(x,y,'r')
    outFile = 'ChallengerPlain.png'
    mystyle.printout_plain(outFile, outDir='..\Images')
    plt.show()


if __name__ == '__main__':
    inData = getData()
    dfFit = prepareForFit(inData)

    # fit the model

    # --- >>> START stats <<< ---
    model = glm('ok + failed ~ temp', data=dfFit, family=Binomial()).fit()
    # --- >>> STOP stats <<< ---

    print(model.summary())

    showResults(inData, model)
```

Listing A.28: ologit.py

```
"""
Implementation of logistic ordinal regression (aka proportional odds) model
"""

from sklearn import metrics
from scipy import linalg, optimize, sparse
import numpy as np
import warnings

BIG = 1e10
SMALL = 1e-12


def phi(t):
    """
    logistic function, returns 1 / (1 + exp(-t))
    """
    idx = t > 0
    out = np.empty(t.size, dtype=np.float)
    out[idx] = 1. / (1 + np.exp(-t[idx]))
    exp_t = np.exp(t[~idx])
    out[~idx] = exp_t / (1. + exp_t)
    return out

def log_logistic(t):
    """
    (minus) logistic loss function, returns log(1 / (1 + exp(-t)))
    """
    idx = t > 0
    out = np.zeros_like(t)
    out[idx] = np.log(1 + np.exp(-t[idx]))
    out[~idx] = (-t[~idx] + np.log(1 + np.exp(t[~idx])))
    return out


def ordinal_logistic_fit(X, y, alpha=0, l1_ratio=0, n_class=None, max_iter
    =10000,
                         verbose=False, solver='TNC', w0=None):
    """
    Ordinal logistic regression or proportional odds model.
    Uses scipy's optimize.fmin_slsqp solver.

    Parameters
    ----------
    X : {array, sparse matrix}, shape (n_samples, n_feaures)
        Input data
    y : array-like
        Target values
    max_iter : int
        Maximum number of iterations
    verbose: bool
        Print convergence information

    Returns
    -------
    w : array, shape (n_features,)
        coefficients of the linear model
    theta : array, shape (k,), where k is the different values of y
        vector of thresholds
    """

    X = np.asarray(X)
```

```
y = np.asarray(y)
w0 = None

if not X.shape[0] == y.shape[0]:
    raise ValueError('Wrong shape for X and y')

# .. order input ..
idx = np.argsort(y)
idx_inv = np.zeros_like(idx)
idx_inv[idx] = np.arange(idx.size)
X = X[idx]
y = y[idx].astype(np.int)
# make them continuous and start at zero
unique_y = np.unique(y)
for i, u in enumerate(unique_y):
    y[y == u] = i
unique_y = np.unique(y)

# .. utility arrays used in f_grad ..
alpha = 0.
k1 = np.sum(y == unique_y[0])
E0 = (y[:, np.newaxis] == np.unique(y)).astype(np.int)
E1 = np.roll(E0, -1, axis=-1)
E1[:, -1] = 0.
E0, E1 = map(sparse.csr_matrix, (E0.T, E1.T))

def f_obj(x0, X, y):
    """
    Objective function
    """
    w, theta_0 = np.split(x0, [X.shape[1]])
    theta_1 = np.roll(theta_0, 1)
    t0 = theta_0[y]
    z = np.diff(theta_0)

    Xw = X.dot(w)
    a = t0 - Xw
    b = t0[k1:] - X[k1:].dot(w)
    c = (theta_1 - theta_0)[y][k1:]

    if np.any(c > 0):
        return BIG

    #loss = -(c[idx] + np.log(np.exp(-c[idx]) - 1)).sum()
    loss = -np.log(1 - np.exp(c)).sum()

    loss += b.sum() + log_logistic(b).sum() \
        + log_logistic(a).sum() \
        + .5 * alpha * w.dot(w) - np.log(z).sum()  # penalty
    if np.isnan(loss):
        pass
        #import ipdb; ipdb.set_trace()
    return loss

def f_grad(x0, X, y):
    """
    Gradient of the objective function
    """
    w, theta_0 = np.split(x0, [X.shape[1]])
    theta_1 = np.roll(theta_0, 1)
    t0 = theta_0[y]
    t1 = theta_1[y]
    z = np.diff(theta_0)
```

```
        Xw = X.dot(w)
        a = t0 - Xw
        b = t0[k1:] - X[k1:].dot(w)
        c = (theta_1 - theta_0)[y][k1:]

        # gradient for w
        phi_a = phi(a)
        phi_b = phi(b)
        grad_w = -X[k1:].T.dot(phi_b) + X.T.dot(1 - phi_a) + alpha * w

        # gradient for theta
        idx = c > 0
        tmp = np.empty_like(c)
        tmp[idx] = 1. / (np.exp(-c[idx]) - 1)
        tmp[~idx] = np.exp(c[~idx]) / (1 - np.exp(c[~idx])) # should not need
        grad_theta = (E1 - E0)[:, k1:].dot(tmp) \
            + E0[:, k1:].dot(phi_b) - E0.dot(1 - phi_a)

        grad_theta[:-1] += 1. / np.diff(theta_0)
        grad_theta[1:] -= 1. / np.diff(theta_0)
        out = np.concatenate((grad_w, grad_theta))
        return out

    def f_hess(x0, s, X, y):
        x0 = np.asarray(x0)
        w, theta_0 = np.split(x0, [X.shape[1]])
        theta_1 = np.roll(theta_0, 1)
        t0 = theta_0[y]
        t1 = theta_1[y]
        z = np.diff(theta_0)

        Xw = X.dot(w)
        a = t0 - Xw
        b = t0[k1:] - X[k1:].dot(w)
        c = (theta_1 - theta_0)[y][k1:]

        D = np.diag(phi(a) * (1 - phi(a)))
        D_= np.diag(phi(b) * (1 - phi(b)))
        D1 = np.diag(np.exp(-c) / (np.exp(-c) - 1) ** 2)
        Ex = (E1 - E0)[:, k1:].toarray()
        Ex0 = E0.toarray()
        H_A = X[k1:].T.dot(D_).dot(X[k1:]) + X.T.dot(D).dot(X)
        H_C = - X[k1:].T.dot(D_).dot(E0[:, k1:].T.toarray()) \
            - X.T.dot(D).dot(E0.T.toarray())
        H_B = Ex.dot(D1).dot(Ex.T) + Ex0[:, k1:].dot(D_).dot(Ex0[:, k1:].T) \
            - Ex0.dot(D).dot(Ex0.T)

        p_w = H_A.shape[0]
        tmp0 = H_A.dot(s[:p_w]) + H_C.dot(s[p_w:])
        tmp1 = H_C.T.dot(s[:p_w]) + H_B.dot(s[p_w:])
        return np.concatenate((tmp0, tmp1))

        import ipdb; ipdb.set_trace()
        import pylab as pl
        pl.matshow(H_B)
        pl.colorbar()
        pl.title('True')
        import numdifftools as nd
        Hess = nd.Hessian(lambda x: f_obj(x, X, y))
        H = Hess(x0)
        pl.matshow(H[H_A.shape[0]:, H_A.shape[0]:])
        #pl.matshow()
```

```
        pl.title('estimated')
        pl.colorbar()
        pl.show()


    def grad_hess(x0, X, y):
        grad = f_grad(x0, X, y)
        hess = lambda x: f_hess(x0, x, X, y)
        return grad, hess

    x0 = np.random.randn(X.shape[1] + unique_y.size) / X.shape[1]
    if w0 is not None:
        x0[:X.shape[1]] = w0
    else:
        x0[:X.shape[1]] = 0.
    x0[X.shape[1]:] = np.sort(unique_y.size * np.random.rand(unique_y.size))

    #print('Check grad: %s' % optimize.check_grad(f_obj, f_grad, x0, X, y))
    #print(optimize.approx_fprime(x0, f_obj, 1e-6, X, y))
    #print(f_grad(x0, X, y))
    #print(optimize.approx_fprime(x0, f_obj, 1e-6, X, y) - f_grad(x0, X, y))
    #import ipdb; ipdb.set_trace()

    def callback(x0):
        x0 = np.asarray(x0)
        # print('Check grad: %s' % optimize.check_grad(f_obj, f_grad, x0, X, y))
        if verbose:
        # check that gradient is correctly computed
            print('OBJ: %s' % f_obj(x0, X, y))

    if solver == 'TRON':
        import pytron
        out = pytron.minimize(f_obj, grad_hess, x0, args=(X, y))
    else:
        options = {'maxiter' : max_iter, 'disp': 0, 'maxfun':10000}
        out = optimize.minimize(f_obj, x0, args=(X, y), method=solver,
            jac=f_grad, hessp=f_hess, options=options, callback=callback)

    if not out.success:
        warnings.warn(out.message)
    w, theta = np.split(out.x, [X.shape[1]])
    return w, theta


def ordinal_logistic_predict(w, theta, X):
    """
    Parameters
    ----------
    w : coefficients obtained by ordinal_logistic
    theta : thresholds
    """
    unique_theta = np.sort(np.unique(theta))
    out = X.dot(w)
    unique_theta[-1] = np.inf # p(y <= max_level) = 1
    tmp = out[:, None].repeat(unique_theta.size, axis=1)
    return np.argmax(tmp < unique_theta, axis=1)

if __name__ == '__main__':
    DOC = """
==============================================================================
    Compare the prediction accuracy of different models on the boston dataset
==============================================================================
    """
```

```
print(DOC)
from sklearn import cross_validation, datasets
boston = datasets.load_boston()
X, y = boston.data, np.round(boston.target)
#X -= X.mean()
y -= y.min()

idx = np.argsort(y)
X = X[idx]
y = y[idx]
cv = cross_validation.ShuffleSplit(y.size, n_iter=50, test_size=.1,
    random_state=0)
score_logistic = []
score_ordinal_logistic = []
score_ridge = []
for i, (train, test) in enumerate(cv):
    #test = train
    if not np.all(np.unique(y[train]) == np.unique(y)):
        # we need the train set to have all different classes
        continue
    assert np.all(np.unique(y[train]) == np.unique(y))
    train = np.sort(train)
    test = np.sort(test)
    w, theta = ordinal_logistic_fit(X[train], y[train], verbose=True,
                                    solver='TNC')
    pred = ordinal_logistic_predict(w, theta, X[test])
    s = metrics.mean_absolute_error(y[test], pred)
    print('ERROR (ORDINAL)  fold %s: %s' % (i+1, s))
    score_ordinal_logistic.append(s)

    from sklearn import linear_model
    clf = linear_model.LogisticRegression(C=1.)
    clf.fit(X[train], y[train])
    pred = clf.predict(X[test])
    s = metrics.mean_absolute_error(y[test], pred)
    print('ERROR (LOGISTIC) fold %s: %s' % (i+1, s))
    score_logistic.append(s)

    from sklearn import linear_model
    clf = linear_model.Ridge(alpha=1.)
    clf.fit(X[train], y[train])
    pred = np.round(clf.predict(X[test]))
    s = metrics.mean_absolute_error(y[test], pred)
    print('ERROR (RIDGE)    fold %s: %s' % (i+1, s))
    score_ridge.append(s)


print()
print('MEAN ABSOLUTE ERROR (ORDINAL LOGISTIC):    %s' % np.mean(
    score_ordinal_logistic))
print('MEAN ABSOLUTE ERROR (LOGISTIC REGRESSION): %s' % np.mean(
    score_logistic))
print('MEAN ABSOLUTE ERROR (RIDGE REGRESSION):    %s' % np.mean(score_ridge)
    )
# print('Chance level is at %s' % (1. / np.unique(y).size))
```

Listing A.29: 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)
    f1me1 = np.zeros(num_failures)
    v = np.zeros(num_failures)

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

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

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

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

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

    print(('X^2 = {0}'.format(chi2)))
    if p < 0.05:
        print(('p={0}, the two survival curves are signifcantly different.'.
            format(p)))
    else:
        print(('p={0}, the two survival curves are not signifcantly different.'.
            format(p)))

    return(p, chi2)

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()
```

Listing A.30: challenger.py

```
'''Example of PyMC - The Challenger Disaster
'''

# author: Thomas Haslwanter, date: Sept-2013

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
import seaborn as sns
import os
import pymc as pm

def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

sns.set_context('poster')

# --- Get and show the O-ring data ---
dataDir = r'..\Data\data_bayes'
inFile = os.path.join(dataDir, 'challenger_data.csv')

challenger_data = np.genfromtxt(inFile, skip_header=1, usecols=[1, 2],
                                missing_values='NA', delimiter=',')

# drop the NA values
challenger_data = challenger_data[~np.isnan(challenger_data[:, 1])]

# plot it, as a function of tempature (the first column)
print("Temp (F), O-Ring failure?")
print(challenger_data)

# First plot
plt.figure()
np.set_printoptions(precision=3, suppress=True)

plt.scatter(challenger_data[:, 0], challenger_data[:, 1], s=75, color="k",
            alpha=0.5)
plt.yticks([0, 1])
plt.ylabel("Damage Incident?")
plt.xlabel("Outside temperature (Fahrenheit)")
plt.title("Defects of the Space Shuttle O-Rings vs temperature")

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

plt.show()

# --- Perform the MCMC-simulations ---
temperature = challenger_data[:, 0]
D = challenger_data[:, 1]  # defect or not?

# Define the prior distributions for alpha and beta
# 'value' sets the start parameter for the simulation
# The second parameter for the normal distributions is the "precision",
# i.e. the inverse of the standard deviation
beta = pm.Normal("beta", 0, 0.001, value=0)
alpha = pm.Normal("alpha", 0, 0.001, value=0)
```

```
# Define the model-function for the temperature
@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta * t + alpha))

# connect the probabilities in `p` with our observations through a
# Bernoulli random variable.
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)

# Combine the values to a model
model = pm.Model([observed, beta, alpha])

# Perform the simulations
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)

# --- Show the resulting posterior distributions ---
alpha_samples = mcmc.trace('alpha')[:, None]  # best to make them 1d
beta_samples = mcmc.trace('beta')[:, None]

plt.figure(figsize=(12.5, 6))

# histogram of the samples:
plt.subplot(211)
plt.title(r"Posterior distributions of the variables $\alpha, \beta$")
plt.hist(beta_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\beta$", color="#7A68A6", normed=True)
plt.legend()

plt.subplot(212)
plt.hist(alpha_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\alpha$", color="#A60628", normed=True)
plt.legend()

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

plt.show()

# --- Show the probability curve ----
# Draw the probability as a function of time
t = np.linspace(temperature.min() - 5, temperature.max() + 5, 50)[:, None]
p_t = logistic(t.T, beta_samples, alpha_samples)

mean_prob_t = p_t.mean(axis=0)

plt.figure(figsize=(12.5, 4))

plt.plot(t, mean_prob_t, lw=3, label="average posterior \nprobability \
of defect")
plt.plot(t, p_t[0, :], ls="--", label="realization from posterior")
plt.plot(t, p_t[-2, :], ls="--", label="realization from posterior")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.title("Posterior expected value of probability of defect; \
plus realizations")
plt.legend(loc="lower left")
plt.ylim(-0.1, 1.1)
plt.xlim(t.min(), t.max())
plt.ylabel("probability")
```

```
plt.xlabel("temperature")

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

plt.show()

# --- Draw CIs ---
from scipy.stats.mstats import mquantiles

# vectorized bottom and top 2.5% quantiles for "confidence interval"
qs = mquantiles(p_t, [0.025, 0.975], axis=0)
plt.fill_between(t[:, 0], *qs, alpha=0.7,
                 color="#7A68A6")

plt.plot(t[:, 0], qs[0], label="95% CI", color="#7A68A6", alpha=0.7)

plt.plot(t, mean_prob_t, lw=1, ls="--", color="k",
         label="average posterior \nprobability of defect")

plt.xlim(t.min(), t.max())
plt.ylim(-0.02, 1.02)
plt.legend(loc="lower left")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.xlabel("temp, $t$")

plt.ylabel("probability estimate")
plt.title("Posterior probability estimates given temp. $t$")

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

plt.show()
```

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. Continous Tests
9. Categorical Tests
10. Correlation [T]
11. Regression [T]
12. ANOVA [T]
13. Statistical Models

Bibliography

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

[Bishop(2007)] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[Box(1978)] J. F. Box. *R. A. Fisher: The Life of a Scientist.* John Wiley & Sons, Inc., New York., 1978.

[Dobson and Barnett(2008)] A.J. Dobson and A. Barnett. *An Introduction to Generalized Linear Models*. CRC Press, 3rd edition, 2008.

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

[Holm(1979)] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

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

[McCullagh(1980)] P. McCullagh. Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)*, 42(2):109–142, 1980.

[McCullagh and Nelder(1989)] P. McCullagh and J.A. Nelder. *Generalized Linear Models*. Springer, 2nd edition, 1989.

[Nuzzo(2014)] Regina Nuzzo. Scientific method: statistical errors. *Nature*, 506(7487):150–152, Feb 2014. doi: 10.1038/506150a. URL http://dx.doi.org/10.1038/506150a.

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

[Rosenbaum and Rubin(1983)] P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

[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.

[Wilkinson and Rogers(1973)] G.N. Wilkinson and C.E. Rogers. Symbolic description of factorial models for analysis of variance. *Applied Statistics*, 22:392–399, 1973.

Glossary

Matplotlib A Python package which provides the ability to generate 2D and 3D plots. This includes the plotting commands, as well as the functionality of different output media, also called *backends*: an output can for example be into the IPython notebook; or into a PDF-file; or into a separate graphics window.. 27

power analysis Calculation of the minimum sample size required so that one can be reasonably likely to detect an effect of a given size.. 51

Probability Distribution A function which assigns a probability to each measurable subset of the possible outcomes of a random experiment.. 53

Acronyms

ANOVA ANalysis Of VAriance. 95

CDF Cumulative Distribution Function. 53

CI Confidence Interval. 58

DOF Degrees of Freedom. 45

ISF Inverse Survival Function. 55

KDE Kernel Density Estimation. 40

PDF Probability Density Function. 53

PPF Percentile Point Function. 55

QQ-Plot Quantile-Quantile Plot. 78

ROC Reiciver Operating Characteristic. 88

RVS Random Variate Sample. 55

SF Survival Function. 54

SS Sum of Squares. 96

Tukey HSD Tukey Honest Significant Difference test. 98

Index Topics

Python Programs