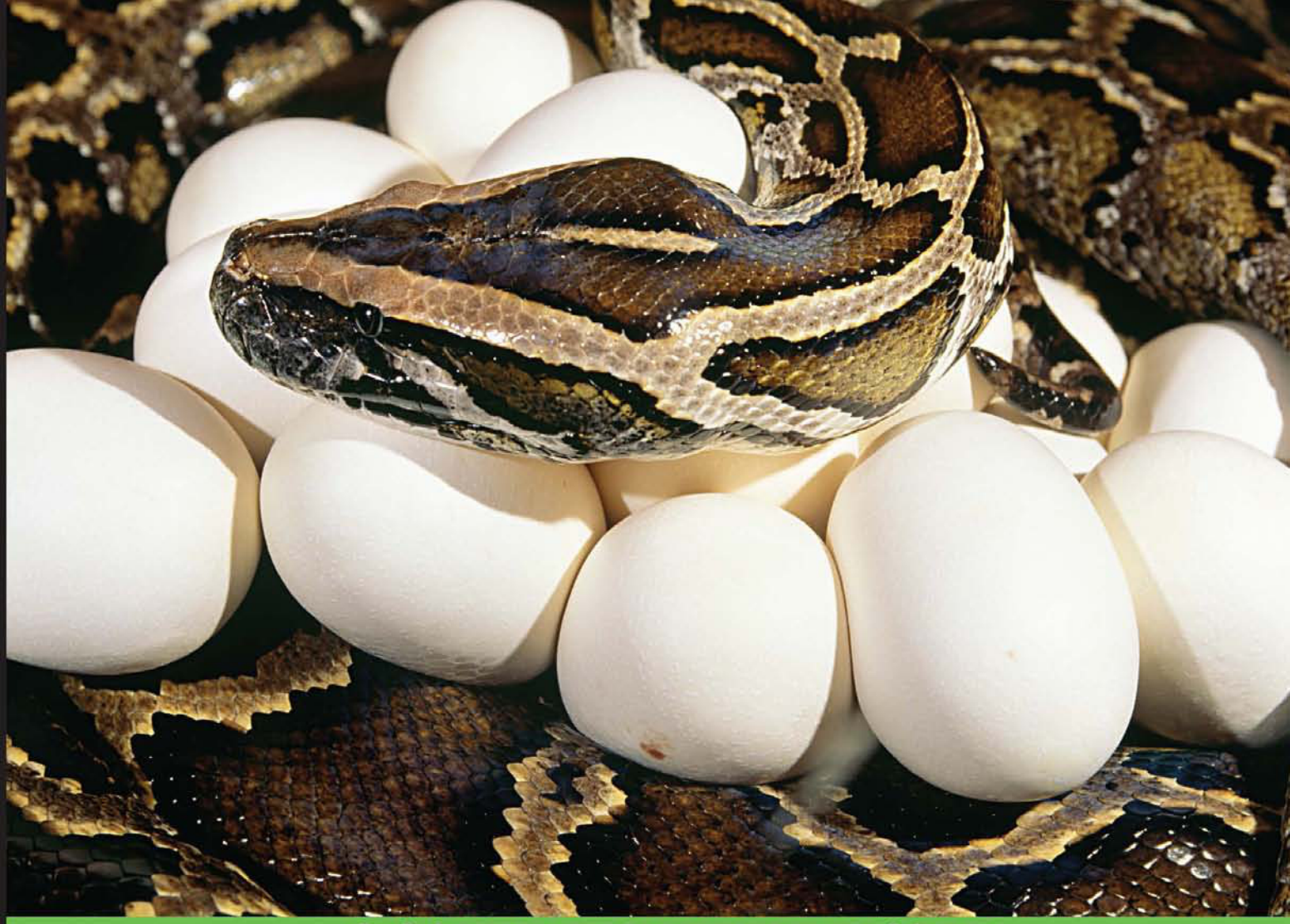


Quick answers to common problems

NumPy Cookbook

Second Edition

Over 90 fascinating recipes to learn and perform mathematical, scientific, and engineering Python computations with NumPy

Ivan Idris

[PACKT] PUBLISHING open source community experience distilled

NumPy Cookbook

Second Edition

Over 90 fascinating recipes to learn and perform

Ivan Idris

[PACKT] PUBLISHING
open source
community experience distilled

BIRMINGHAM - MUMBAI

NumPy Cookbook
Second Edition

www.packtpub.com

Credits

Author
Ivan Idris

Reviewers

Commissioning Editor

Acquisition Editors

Content Development Editor

Technical Editors

Copy Editor

Project Coordinator

Proofreaders

Indexer

Graphics

Production Coordinator
Shantanu N. Zagade

Cover Work
Shantanu N. Zagade

About the Author

Ivan Idris

code and interesting technical articles. He is the author of *NumPy Beginner's Guide*, *NumPy Cookbook*, *Python Data Analysis*, and *Learning NumPy*

`http://ivanidris.net/wordpress/`.

---

friends for their support.

About the Reviewers

Lev E. Givon is a doctoral candidate and neurocomputing researcher at the department of

(`http://neurokernel.github.io`

Mark Livingstone
computer companies (which no longer exist) in engineering, support, programming, and

peace at the local district courthouse. He is also a credit union director, and has completed

Lijun Xue

`http://royxue.me/`.

www.PacktPub.com

www.PacktPub.com.

www.PacktPub.com and as a print

service@packtpub.com for more details.

At www.PacktPub.com

https://www2.packtpub.com/books/subscription/packtlib

Why Subscribe?

www.PacktPub.com

immediate access.

these environments.

What this book covers

Chapter 1, *Winding Along with IPython*

Chapter 2, *Advanced Indexing and Array Concepts*

Chapter 3, *Getting to Grips with Commonly Used Functions*

Chapter 4, *Connecting NumPy with the Rest of the World*

Chapter 5, *Audio and Image Processing*

Chapter 6, *Special Arrays and Universal Functions*

heterogeneous data.

Chapter 7,

Chapter 8, *Quality Assurance*

Chapter 9, *Speeding Up Code with Cython*

Chapter 10, *Fun with Scikits*

Chapter 11, *Latest and Greatest NumPy*

Chapter 12, *Exploratory and Predictive Data Analysis with NumPy*

Conventions

include other contexts through the use of the `include`

```
from __future__ import print_function
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np
import matplotlib.pyplot as plt

def get_indices(high, size):
   #2. Generate random indices
   return np.random.randint(0, high, size)
```

```
from sklearn.datasets import load_sample_images
import matplotlib.pyplot as plt
import skimage.feature

dataset = load_sample_images()
img = dataset.images[0]
edges = skimage.feature.canny(img[..., 0])
plt.axis('off')
plt.imshow(edges)
plt.show()
```

```
$ sudo easy_install patsy
```

New terms and **important words**

Print

[]

[]

feedback@packtpub.com,

www.packtpub.com/authors.

Customer support

http://www.packtpub.com

http://www.packtpub.com/support

https://www.packtpub.com/sites/default/files/downloads/0945OS.pdf.

Errata

http://www.packtpub.com/submit-errata,
Errata Submission Form

of that title.

https://www.packtpub.com/books/content/support
information will appear under the **Errata** section.

Piracy

copyright@packtpub.com
pirated material.

Questions

questions@packtpub.com

1

IPython

Reading manual pages

Introduction

`http://ipython.org/`

media and plotting

http://www.
pythonanywhere.com/try-ipython/

the

readline
tornado and zmq.

setuptools
easy_install easy_install command is a popular package
pip easy_install pip
command is similar to easy_install and adds options such as uninstalling.

How to do it...

easy_install and
pip

Installing IPython and setuptools on Windows

http://ipython.org/
ipython-doc/stable/install/install.html#windows.

Install setuptools with an installer from http://pypi.python.org/pypi/
setuptools#files pip

```
cd C:\Python27\scripts
python .\easy_install-27-script.py pip
```

Installing IPython on Mac OS X

at https://developer.apple.com/xcode/.
Follow the easy_install/pip instructions or the instructions for installation from source provided later in this section.

Installing IPython on Linux

```
$ su - aptitude install ipython python-setuptools
```

```
$ su – yum install ipython python-setuptools-devel
```

following command will

```
$ su – emerge ipython
```

```
$ sudo apt-get install ipython python-setuptools
```

Installing IPython with easy_install or pip

dependencies recipes in this chapter with `easy_install`

```
$ sudo easy_install ipython pyzmq tornado readline
```

`easy_install` command in

```
$ sudo easy_install pip
```

`pip`

```
$ sudo pip install ipython pyzmq tornado readline
```

Installing from source

then

`https://github.com/ipython/ipython/archive/master.zip`.

```
$ tar xzf ipython-<version>.tar.gz
```

3.

```
$ git clone https://github.com/ipython/ipython.git
```

```
$ cd ipython
```

`sudo`

```
$ sudo python setup.py install
```

How it works...

Instructions `http://ipython.org/install.html`

Scientists and

Inline editing

`%run`

`pylab` switch

How to do it...

`pylab` `pylab`

```
$ ipython --pylab
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
```

```
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
details.

Welcome to pylab, a matplotlib-based Python environment [backend:
MacOSX].
For more information, type 'help(pylab)'.

In [1]: quit()
quit() or Ctrl + D quits the IPython shell.
```

Saving a session our experiments.

```
In [1]: %logstart
Activating auto-logging. Current session state plus future input
saved.
Filename       : ipython_log.py
Mode           : rotate
Output logging : False
Raw input log  : False
Timestamping   : False
State          : active

In [9]: %logoff
Switching logging OFF
```

Executing a system shell command

!

```
In [1]: !date
```

!

```
In [2]: thedate = !date
In [3]: thedate
```

Displaying history the `%hist`

```
In [1]: a = 2 + 2

In [2]: a
Out[2]: 4

In [3]: %hist
a = 2 + 2
a
%hist
```

Command-line Interface (**CLI**) environments.
`-g`

```
In [5]: %hist -g a = 2
    1: a = 2 + 2
```

Downloading the example code

`http://www.packtpub.com`
`http://www.packtpub.com/support`

How it works...

`%`

`%`

IPython as a system shell `http://ipython.org/ipython-doc/dev/interactive/shell.html`

We can open the `help` command. It is

`arange()` function.

How to do it...

Calling the help function `help`
function and then press the *Tab*

```
In [1]: help ar
arange          arctan          argpartition  array2string  array_str
arccos          arctan2         argsort       array_equal   arrow
arccosh         arctanh         argwhere      array_equiv
arcsin          argmax          around        array_repr
arcsinh         argmin          array         array_split
```

Querying with a question mark

`help`

```
In [3]: arange?
```

How it works...

`readline`
`docstrings`.

need it

How to do it...

Installing matplotlib on Windows
`http://www.enthought.com/products/epd.php`).

It `msvcp71.dll` `C:\Windows\system32` `http://www.dll-files.com/dllindex/dll-files.shtml?msvcp71`.

Installing matplotlib on Linux

Here is the

```
$ sudo apt-get install python-matplotlib
```

```
$ su - yum install python-matplotlib
```

Installing from source `tar.gz` release at Sourceforge (`http://sourceforge.net/projects/matplotlib/files/`),

```
$ git clone git://github.com/matplotlib/matplotlib.git
```

Once it has

```
$ cd matplotlib
$ sudo python setup.py install
```

Installing matplotlib on Mac OS X `http://sourceforge.net/projects/matplotlib/files/matplotlib/` and

Instructions `http://matplotlib.org/users/installing.html`

Installing `http://www.scipy.org/install.html`

notebook server can serve

Markdown `https://en.wikipedia.org/wiki/Markdown`) in text cells

`tornado` and `zmq`. See the *Installing IPython* recipe in this chapter for more information.

How to do it...

Running a notebook

```
$ ipython notebook
```

```
[NotebookApp] Using existing profile dir: u'/Users/ivanidris/.
ipython/profile_default'
[NotebookApp] The IPython Notebook is running at:
http://127.0.0.1:8888
[NotebookApp] Use Control-C to stop this server and shut down all
kernels.
```

this chapter.

Ctrl *C*.

Running a notebook in the pylab mode

mode with

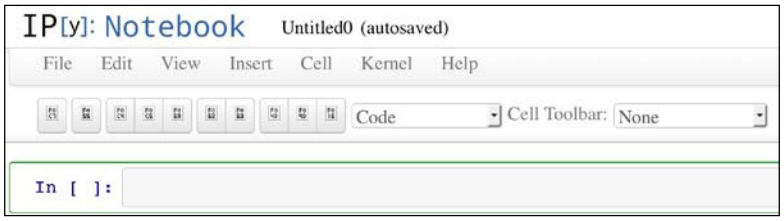
```
$ ipython notebook --pylab
```

`SciPy`, `NumPy`, and `matplotlib` modules.

plots with the `inline`

```
$ ipython notebook --pylab inline
```

New Notebook



`arange()`

Cell Run

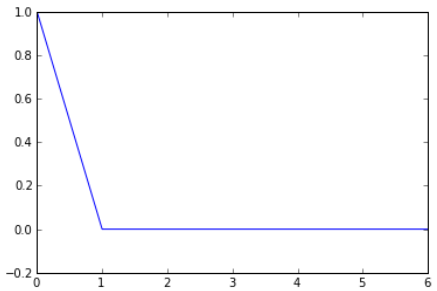
```
In [1]: a = arange(7)
```

3. Next enter the following command and press *Enter* in **Out [2]**

```
In [2]: a
Out[2]: array([0, 1, 2, 3, 4, 5, 6])
```

`sinc()`

```
In [3]: plot(sinc(a))
Out[3]: [<matplotlib.lines.Line2D at 0x103d9c690>]
```



How it works...

`pylab`

Installing IPython recipe found in this chapter

Example `http://nbviewer.ipython.org/github/ipython/ipython/blob/2.x/examples/Notebook/Index.ipynb`

for the `sinc()` function at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.sinc.html`

for the `plot()` function at `http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot`

How to do it...

The Print option **Print**

Downloading the notebook using the **Download** `.py` as a `.ipynb`

```
{
 "metadata": {
  "name": "Untitled1"
 },
 "nbformat": 2,
 "worksheets": [
  {
    "cells": [
    {
      "cell_type": "code",
      "collapsed": false,
      "input": [
        "plot(sinc(a))"
      ],
      "language": "python",
```

```
            "outputs": [
            {
              "output_type": "pyout",
              "prompt_number": 3,
              "text": [
                "[&lt;matplotlib.lines.Line2D at
                  0x103d9c690&gt;]"
              ]
            },
            {
              "output_type": "display_data",
              "png": "iVBORw0KGgoAAAANSUhEUgAAAXk
                AAAD9CAYAAABZVQdHAAAABHNCSVQICAgIf...
                mgkAAAAASUVORK5CYII=\n"
            }
            ],
            "prompt_number": 3
          }
          ]
        }
        ]
    }
```

https://en.wikipedia.org/wiki/JSON.

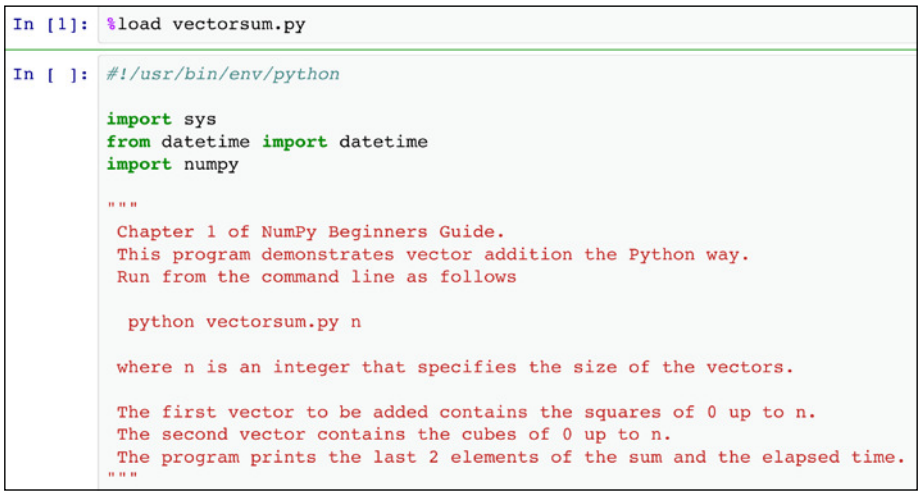
Saving the notebook

Save format, `.ipynb`

How to do it...

```
% load vectorsum.py
```

screenshot shows an example of what we see after loading `vectorsum.py` from *NumPy Beginner's Guide*



[redacted]

(for more information, see `https://en.wikipedia.org/wiki/Transport_Layer_Security`

authority

How to do it...

```
In [1]: from IPython.lib import passwd

In [2]: passwd()
Enter password:
Verify password:
Out[2]: 'sha1:0e422dfccef2:84cfbcb
  b3ef95872fb8e23be3999c123f862d856'
```

later on.

`openssl`

Setting up the `openssl`

`mycert.pem`

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
mycert.pem -out mycert.pem
Generating a 1024 bit RSA private key
......++++++
.........................++++++
writing new private key to 'mycert.pem'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:
```

`openssl`

```
$ man openssl
```

3.

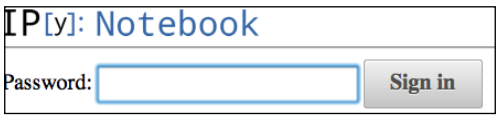
```
$ ipython profile create nbserver
```

~/.ipython/profile_nbserver/ipython_notebook_config.py.

```
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate'
c.NotebookApp.password = u'sha1:b...your password'
c.NotebookApp.port = 9999
```

and changed the port to 9999.

```
$ ipython notebook --profile=nbserver
[NotebookApp] Using existing profile dir: u'/Users/ivanidris/.
ipython/profile_nbserver'
[NotebookApp] The IPython Notebook is running at:
https://127.0.0.1:9999
[NotebookApp] Use Control-C to stop this server and shut down all
kernels.
```



How it works...

profile_<profilename> folder to the .ipython

--profile=<profile_name>

```
$ ipython profile list

Available profiles in IPython:
```

```
    cluster
    math
    pysh
    python3

    The first request for a bundled profile will copy it
    into your IPython directory (/Users/ivanidris/.ipython),
    where you can customize it.

Available profiles in /Users/ivanidris/.ipython:
    default
    nbserver
    sh
```

documentation for the `passwd()` function at `http://ipython.org/ipython-doc/2/api/generated/IPython.lib.security.html`

documentation at `https://www.openssl.org/docs/apps/openssl.html`

`easy_install` or `pip`

```
$ sudo easy_install sympy
$ sudo pip install sympy
```

How to do it...

~/.ipython/profile_sympy/ipython_config.py

```
c = get_config()
app = c.InteractiveShellApp

# This can be used at any point in a config file to load a sub config
# and merge it into the current one.
load_subconfig('ipython_config.py', profile='default')

lines = """
from __future__ import division
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
"""

# You have to make sure that attributes that are containers already
# exist before using them.  Simple assigning a new list will override
# all previous values.

if hasattr(app, 'exec_lines'):
    app.exec_lines.append(lines)
else:
    app.exec_lines = [lines]

# Load the sympy_printing extension to enable nice printing of sympy expr's.
```

```
if hasattr(app, 'extensions'):
    app.extensions.append('sympyprinting')
else:
    app.extensions = ['sympyprinting']
```

```
$ ipython --profile=sympy
```

3.

```
In [1]: expand((x+y)**7)
Out[1]:

 7      6          5  2       4  3       3  4       2  5        6    7
x  + 7⋅x ⋅y + 21⋅x ⋅y  + 35⋅x ⋅y  + 35⋅x ⋅y  + 21⋅x ⋅y  + 7⋅x⋅y  + y
```

http://sympy.org/en/index.html

2

Array Concepts

Resizing images
Comparing views and copies

Indexing with a list of locations
Indexing with Booleans

Introduction

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*.

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

`print()`

`print()`

Some of the examples in this chapter involve manipulating images. In order to do that,
Python Image Library (**PIL**

In *Chapter 1*, *Winding Along with IPython*, we discussed how to install `setuptools` and `pip`.

How to do it...

Installing from source

```
$ git clone https://github.com/scipy/scipy.git

$ python setup.py build
$ python setup.py install --user
```

`BLAS` and `LAPACK`

Installing SciPy on Linux

```
$ yum install python-scipy
```

```
$ urpmi python-scipy
```

```
$ sudo emerge scipy
```

```
$ sudo apt-get install python-scipy
```

Installing SciPy on Mac OS X

contains the `BLAS` and `LAPACK`

`https://developer.apple.com/xcode/`

`gfortran` `http://r.research.att.com/tools/`.

Installing SciPy using `easy_install` or `pip`

of these two commands (the need for `sudo`

```
$ [sudo] pip install scipy
```

```
$ [sudo] easy_install scipy
```

Installing on Windows

is to

Check your installation

```
import scipy
print(scipy.__version__)
print(scipy.__file__)
```

How it works...

`#scipy` IRC channel of `freenode`

mailing lists at `http://www.scipy.org/scipylib/mailing-lists.html`

How to do it...

Installing PIL on Windows

`http://www.pythonware.com/products/pil/`.

Installing on Debian or Ubuntu

the

```
$ sudo apt-get install python-imaging
```

Installing with `easy_install` or `pip`

appears

Install

```
$ easy_install PIL
$ sudo pip install PIL
```


Instructions

`http://pillow.readthedocs.org/en/latest/installation.html`

In this recipe, we

image data as an input.

We will resize the image using the `repeat()`

corresponding recipes in this chapter and *Chapter 1*, *Winding Along with IPython*.

How to do it...

First, import `SciPy` `lena()` function. It is used to load the image

```
lena = scipy.misc.lena()
```

```
lena = scipy.lena()
```

`assert_equal()` function from the `numpy.testing`

```
np.testing.assert_equal((LENA_X, LENA_Y), lena.shape)
```

3. `repeat()` function. We give this function a resize factor in the `x` and `y`

```
resized = lena.repeat(yfactor, axis=0).repeat(xfactor, axis=1)
```

We will plot the

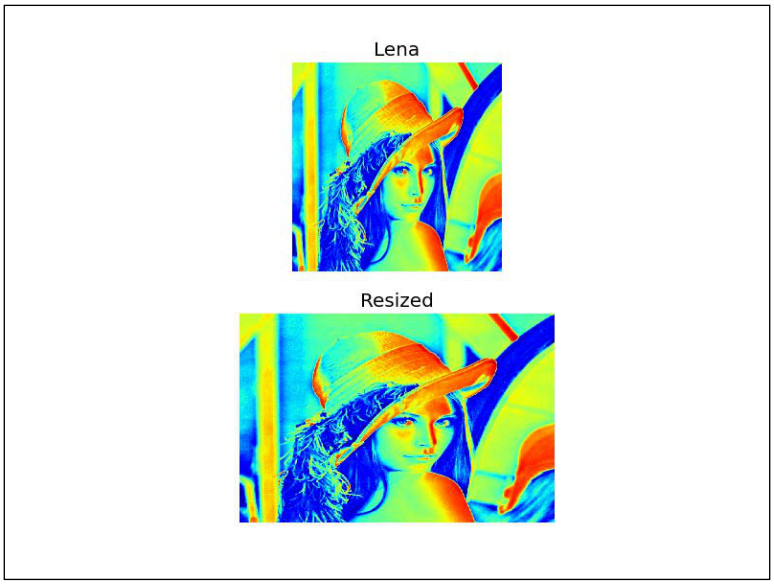
```
plt.subplot(211)
plt.title("Lena")
plt.axis("off")
plt.imshow(lena)
```

`subplot()`

`imshow()` `show()`

the end result.

```
plt.subplot(212)
plt.title("Resized")
plt.axis("off")
plt.imshow(resized)
plt.show()
```



complete code for this recipe from the `resize_lena.py`

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

# This script resizes the Lena image from Scipy.

# Loads the Lena image into an array
lena = scipy.misc.lena()

#Lena's dimensions
LENA_X = 512
LENA_Y = 512

#Check the shape of the Lena array
np.testing.assert_equal((LENA_X, LENA_Y), lena.shape)

# Set the resize factors
yfactor = 2
xfactor = 3

# Resize the Lena array
resized = lena.repeat(yfactor, axis=0).repeat(xfactor, axis=1)

#Check the shape of the resized array
np.testing.assert_equal((yfactor * LENA_Y, xfactor * LENA_Y),
resized.shape)

# Plot the Lena array
plt.subplot(211)
plt.title("Lena")
plt.axis("off")
plt.imshow(lena)

#Plot the resized array
plt.subplot(212)
plt.title("Resized")
plt.axis("off")
plt.imshow(resized)
plt.show()
```

How it works...

`repeat()` function

`subplot()`

`show()`

`imshow()` function

Installing matplotlib in *Chapter 1*, *Winding Along with IPython*

Installing SciPy in this chapter

Installing PIL in this chapter

`repeat()` `http://docs.scipy.org/doc/numpy/reference/generated/numpy.repeat.html`

It is important to have a

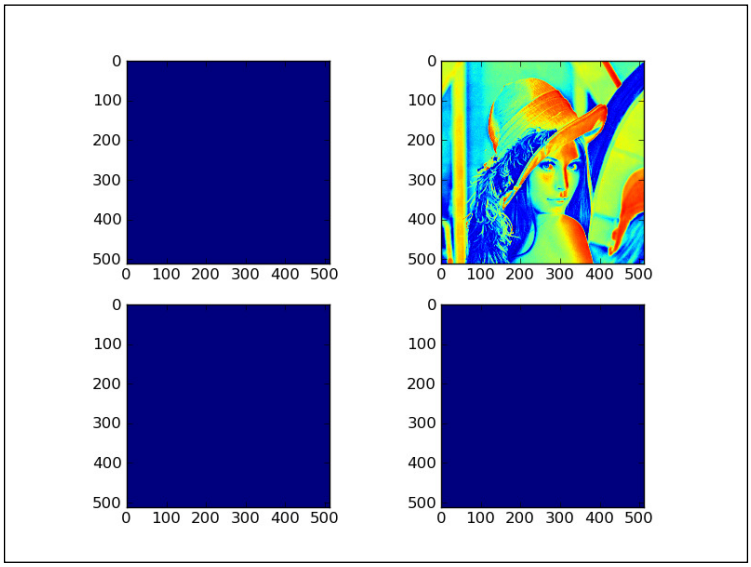
How to do it...

```
acopy = lena.copy()
```

```
aview = lena.view()
```

3. Set all the values of the view to `0` with a `flat`

```
aview.flat = 0
```



is the code of this tutorial, showing the copies from the `copy_view.py`

```
import scipy.misc
import matplotlib.pyplot as plt

lena = scipy.misc.lena()
acopy = lena.copy()
aview = lena.view()

# Plot the Lena array
plt.subplot(221)
plt.imshow(lena)

#Plot the copy
plt.subplot(222)
plt.imshow(acopy)

#Plot the view
plt.subplot(223)
plt.imshow(aview)

# Plot the view after changes
```

```
aview.flat = 0
plt.subplot(224)
plt.imshow(aview)

plt.show()
```

How it works...

changing the view at the end of the program, we changed the original

`view()` function is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.view.html`

How to do it...

```
plt.imshow(lena[:,::-1])
```

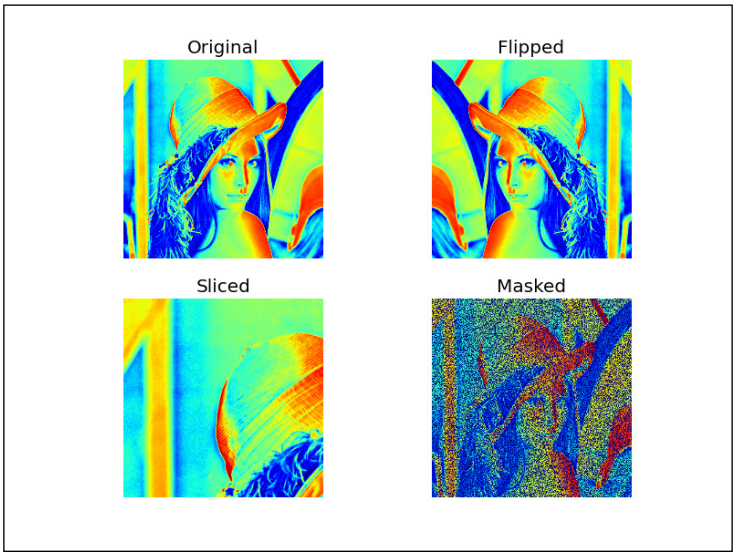
```
plt.imshow(lena[:lena.shape[0]/2,:lena.shape[1]/2])
```

3.

to `0`

```
mask = lena % 2 == 0
masked_lena = lena.copy()
masked_lena[mask] = 0
```

All of these



Here is the complete code for this recipe from the `flip_lena.py`

```
import scipy.misc
import matplotlib.pyplot as plt

# Load the Lena array
lena = scipy.misc.lena()

# Plot the Lena array
plt.subplot(221)
plt.title('Original')
plt.axis('off')
plt.imshow(lena)

#Plot the flipped array
plt.subplot(222)
plt.title('Flipped')
plt.axis('off')
plt.imshow(lena[:,::-1])

#Plot a slice array
plt.subplot(223)
```

```
plt.title('Sliced')
plt.axis('off')
plt.imshow(lena[:lena.shape[0]/2,:lena.shape[1]/2])

# Apply a mask
mask = lena % 2 == 0
masked_lena = lena.copy()
masked_lena[mask] = 0
plt.subplot(224)
plt.title('Masked')
plt.axis('off')
plt.imshow(masked_lena)

plt.show()
```

Installing matplotlib in *Chapter 1*, *Winding Along with IPython*

Installing SciPy in this chapter

Installing PIL in this chapter

How to do it...

0.

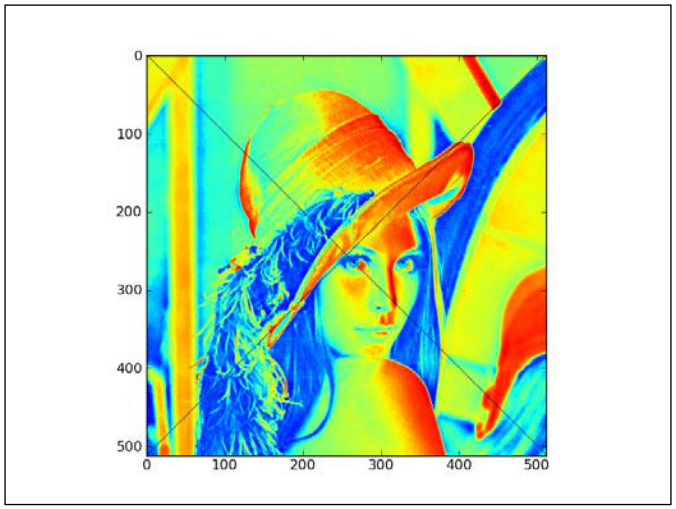
0

x and y

```
lena[range(xmax), range(ymax)] = 0
```

Set the values of the other diagonal to 0.

```
lena[range(xmax-1,-1,-1), range(ymax)] = 0
```



is the complete code for this recipe from the `fancy.py`

```
import scipy.misc
import matplotlib.pyplot as plt

# This script demonstrates fancy indexing by setting values
# on the diagonals to 0.

# Load the Lena array
lena = scipy.misc.lena()
xmax = lena.shape[0]
ymax = lena.shape[1]

# Fancy indexing
# Set values on diagonal to 0
# x 0-xmax
# y 0-ymax
lena[range(xmax), range(ymax)] = 0

# Set values on other diagonal to 0
# x xmax-0
# y 0-ymax
```

```
lena[range(xmax-1,-1,-1), range(ymax)] = 0

# Plot Lena with diagonal lines set to 0
plt.imshow(lena)
plt.show()
```

How it works...

`x` values and `y`

3.

implementation is documented at `http://docs.scipy.org/doc/numpy-dev/reference/internals.code-explanations.html#fancy-indexing-check`

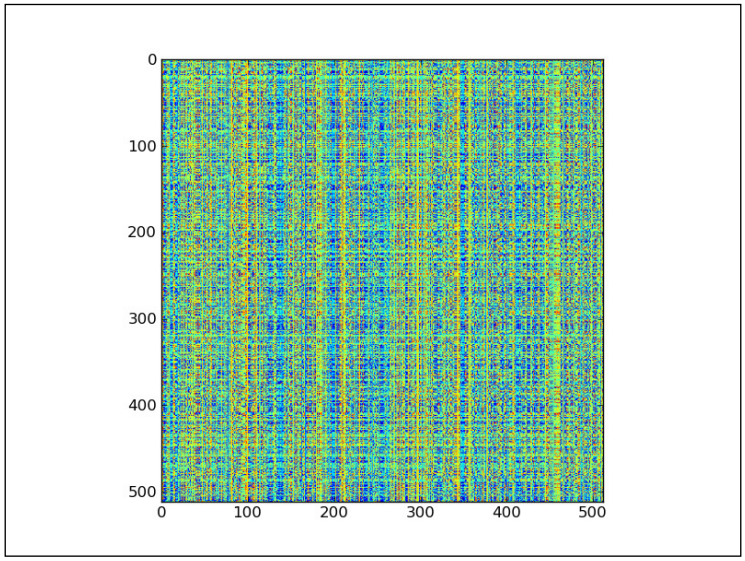
use the `ix_()`

How to do it...

`shuffle()` function of the `numpy.random`

```
def shuffle_indices(size):
   arr = np.arange(size)
   np.random.shuffle(arr)

   return arr
```

```
plt.imshow(lena[np.ix_(xindices, yindices)])
```



Here is the complete code for the recipe from the `ix.py`

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

# Load the Lena array
lena = scipy.misc.lena()
xmax = lena.shape[0]
ymax = lena.shape[1]

def shuffle_indices(size):
   '''
   Shuffles an array with values 0 - size
   '''
```

```
    arr = np.arange(size)
    np.random.shuffle(arr)

    return arr

xindices = shuffle_indices(xmax)
np.testing.assert_equal(len(xindices), xmax)
yindices = shuffle_indices(ymax)
np.testing.assert_equal(len(yindices), ymax)

# Plot Lena
plt.imshow(lena[np.ix_(xindices, yindices)])
plt.show()
```

ix_() documentation page at http://docs.scipy.org/doc/numpy/reference/generated/numpy.ix_.html

Boolean indexing is indexing

How to do it...

Image with dots on the diagonal.

Fancy indexing

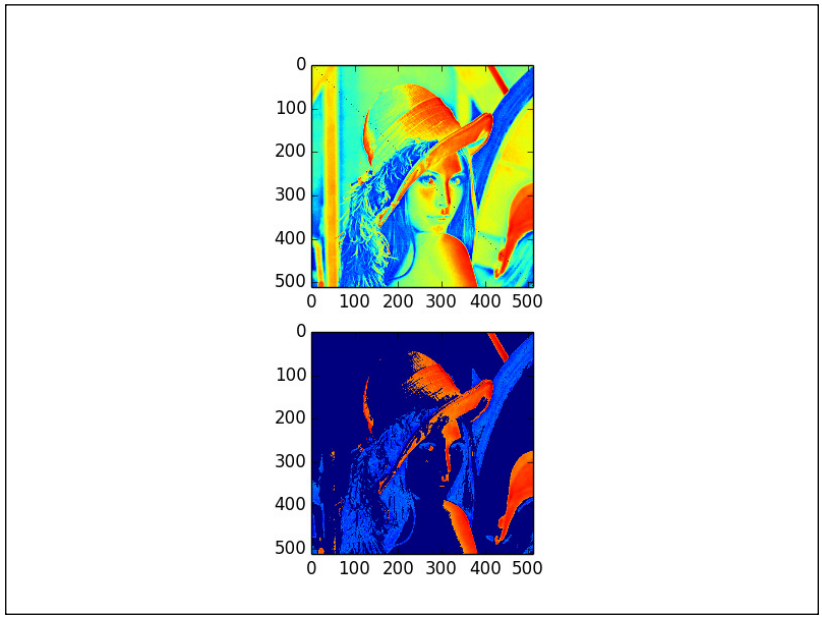
we select modulo 4

```
def get_indices(size):
    arr = np.arange(size)
    return arr % 4 == 0
```

```
lena1 = lena.copy()
xindices = get_indices(lena.shape[0])
yindices = get_indices(lena.shape[1])
lena1[xindices, yindices] = 0
plt.subplot(211)
plt.imshow(lena1)
```

and set them to `0`

```
lena2[(lena > lena.max()/4) &
  (lena < 3 * lena.max()/4)] = 0
```



Here is the complete code for this recipe from the `boolean_indexing.py`

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

# Load the Lena array
```

```
lena = scipy.misc.lena()

def get_indices(size):
   arr = np.arange(size)
   return arr % 4 == 0

# Plot Lena
lena1 = lena.copy()
xindices = get_indices(lena.shape[0])
yindices = get_indices(lena.shape[1])
lena1[xindices, yindices] = 0
plt.subplot(211)
plt.imshow(lena1)

lena2 = lena.copy()
# Between quarter and 3 quarters of the max value
lena2[(lena > lena.max()/4) & (lena < 3 * lena.max()/4)] = 0
plt.subplot(212)
plt.imshow(lena2)


plt.show()
```

How it works...

Since indexing with

Fancy Indexing

`ndarray` class has a `strides`

Explaining the

`http://en.wikipedia.org/wiki/Sudoku`.

How to do it...

sudoku

```
sudoku = np.array([
    [2, 8, 7, 1, 6, 5, 9, 4, 3],
    [9, 5, 4, 7, 3, 2, 1, 6, 8],
    [6, 1, 3, 8, 4, 9, 7, 5, 2],
    [8, 7, 9, 6, 5, 1, 2, 3, 4],
    [4, 2, 1, 3, 9, 8, 6, 7, 5],
    [3, 6, 5, 4, 2, 7, 8, 9, 1],
    [1, 9, 8, 5, 7, 3, 4, 2, 6],
    [5, 4, 2, 9, 1, 6, 3, 8, 7],
    [7, 3, 6, 2, 8, 4, 5, 1, 9]
    ])
```

itemsize ndarray

```
itemsize
strides = sudoku.itemsize * np.array([27, 3, 9, 1])
```

3. Now we can as_strided() function of the np.lib.stride_tricks

```
squares = np.lib.stride_tricks.as_strided
  (sudoku, shape=shape, strides=strides)
print(squares)
```

```
[[[[2 8 7]
    [9 5 4]
    [6 1 3]]

  [[1 6 5]
    [7 3 2]
    [8 4 9]]

  [[9 4 3]
    [1 6 8]
    [7 5 2]]]


 [[[8 7 9]
    [4 2 1]
```

```
    [3 6 5]]

  [[6 5 1]
    [3 9 8]
    [4 2 7]]

  [[2 3 4]
    [6 7 5]
    [8 9 1]]]


 [[[1 9 8]
    [5 4 2]
    [7 3 6]]

  [[5 7 3]
    [9 1 6]
    [2 8 4]]

  [[4 2 6]
    [3 8 7]
    [5 1 9]]]]
```

strides.py

```
import numpy as np

sudoku = np.array([
   [2, 8, 7, 1, 6, 5, 9, 4, 3],
   [9, 5, 4, 7, 3, 2, 1, 6, 8],
   [6, 1, 3, 8, 4, 9, 7, 5, 2],
   [8, 7, 9, 6, 5, 1, 2, 3, 4],
   [4, 2, 1, 3, 9, 8, 6, 7, 5],
   [3, 6, 5, 4, 2, 7, 8, 9, 1],
   [1, 9, 8, 5, 7, 3, 4, 2, 6],
   [5, 4, 2, 9, 1, 6, 3, 8, 7],
   [7, 3, 6, 2, 8, 4, 5, 1, 9]
   ])

shape = (3, 3, 3, 3)

strides = sudoku.itemsize * np.array([27, 3, 9, 1])

squares = np.lib.stride_tricks.as_strided(sudoku, shape=shape,
strides=strides)
print(squares)
```

How it works...

We applied stride

strides http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.strides.html

Without

How to do it...

read()

function returns a `data`

```
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
```

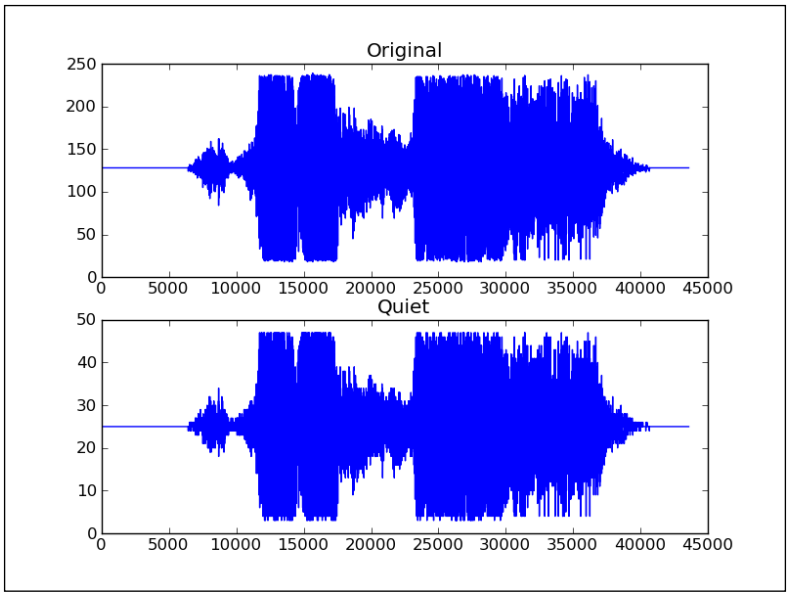
Original

```
plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)
```

3.

```
newdata = data * 0.2
newdata = newdata.astype(np.uint8)

scipy.io.wavfile.write("quiet.wav",
    sample_rate, newdata)
```

```
plt.subplot(2, 1, 2)
plt.title("Quiet")
plt.plot(newdata)

plt.show()
```



Here is the complete code for this recipe from the `broadcasting.py`

```
import scipy.io.wavfile
import matplotlib.pyplot as plt
import urllib2
import numpy as np

# Download audio file
response = urllib2.urlopen('http://www.thesoundarchive.com/
austinpowers/smashingbaby.wav')
print(response.info())
```

```
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print("Data type", data.dtype, "Shape", data.shape)

# Plot values original audio
plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)

# Create quieter audio
newdata = data * 0.2
newdata = newdata.astype(np.uint8)
print("Data type", newdata.dtype, "Shape", newdata.shape)

# Save quieter audio file
scipy.io.wavfile.write("quiet.wav",
    sample_rate, newdata)

# Plot values quieter file
plt.subplot(2, 1, 2)
plt.title("Quiet")
plt.plot(newdata)

plt.show()
```

`scipy.io.read()` function page at `http://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.read.html`

`scipy.io.write()` function page at `http://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.write.html`

concept is explained at `http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html`

3

Functions

`sqrt()`, `log()`, `arange()`, `astype()`, and `sum()`

`ceil()`, `modf()`, `where()`, `ravel()`, and `take()`

`sort()` and `outer()`

`diff()`, `sign()`, and `eig()`

`histogram()` and `polyfit()`

`compress()` and `randint()`

Finding prime factors

Simulating trading at random

Sieving integers with the Sieve of Eratosthenes

Introduction

In this recipe, we

Fibonacci series integers starting with zero, where

For more
at `http://en.wikipedia.org/wiki/Fibonacci_number`.

golden ratio

$$\varphi = \frac{1+\sqrt{5}}{2}$$

We will use the `sqrt()`, `log()`, `arange()`, `astype()`, and `sum()`

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

How to do it...

`sum_fibonacci.py`

```
import numpy as np
```

```
#Each new term in the Fibonacci sequence is generated by adding the
previous two terms.
#By starting with 1 and 2, the first 10 terms will be:

#1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

#By considering the terms in the Fibonacci sequence whose values do
not exceed four million,
#find the sum of the even-valued terms.

#1. Calculate phi
phi = (1 + np.sqrt(5))/2
print("Phi", phi)

#2. Find the index below 4 million
n = np.log(4 * 10 ** 6 * np.sqrt(5) + 0.5)/np.log(phi)
print(n)

#3. Create an array of 1-n
n = np.arange(1, n)
print(n)

#4. Compute Fibonacci numbers
fib = (phi**n - (-1/phi)**n)/np.sqrt(5)
print("First 9 Fibonacci Numbers", fib[:9])

#5. Convert to integers
# optional
fib = fib.astype(int)
print("Integers", fib)

#6. Select even-valued terms
eventerms = fib[fib % 2 == 0]
print(eventerms)

#7. Sum the selected terms
print(eventerms.sum())
```

is calculate the golden ratio (see `http://en.wikipedia.org/wiki/Golden_ratio`), also called the **golden section** or golden mean.

`sqrt()` `5`

```
phi = (1 + np.sqrt(5))/2
print("Phi", phi)
```

```
Phi 1.61803398875
```

`log()`

```
n = np.log(4 * 10 ** 6 * np.sqrt(5)
    + 0.5)/np.log(phi)
print(n)
```

`n`

```
33.2629480359
```

3. `arange()`

```
n = np.arange(1, n)
```

```
fib = (phi**n - (-1/phi)**n)/np.sqrt(5)
print("First 9 Fibonacci Numbers", fib[:9])
```

I could have made a unit test instead of a print statement. A unit test is

Chapter 8, *Quality Assurance*, for pointers on how to write a unit test.

```
First 9 Fibonacci Numbers [ 1.  1.  2.  3.  5.  8. 13. 21.
34.]
```

Convert to integers.

`astype()`

```
fib = fib.astype(int)
print("Integers", fib)
```

```
Integers [      1       1       2       3       5       8      13
21      34
  ... snip ... snip ...
  317811  514229  832040 1346269 2178309 3524578]
```

Select the

Indexing with Booleans recipe in *Chapter 2*, *Advanced Indexing and Array Concepts*

```
eventerms = fib[fib % 2 == 0]
print(eventerms)
```

```
[      2       8      34     144     610    2584   10946   46368
196418  832040 3524578]
```

How it works...

In this recipe, we used the `sqrt()`, `log()`, `arange()`, `astype()`, and `sum()` functions.

Function	Description
`sqrt()`	`http://docs.scipy.org/doc/numpy/reference/generated/numpy.sqrt.html)`
`log()`	`http://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html#numpy.log)`
`arange()`	`http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html)`
`astype()`	`http://docs.scipy.org/doc/numpy/reference/generated/numpy.chararray.astype.html)`
`sum()`	`http://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html)`

Indexing with Booleans recipe in *Chapter 2*, *Advanced Indexing and Array Concepts*

Prime factors (`http://en.wikipedia.org/wiki/Prime_factor`

method (`http://en.wikipedia.org/wiki/Fermat%27s_factorization_method`

N *c* and *d*,

$$N = cd = (a+b)(a-b) = a^2 - b^2$$

How to do it...

fermatfactor.py

```
from __future__ import print_function
import numpy as np

#The prime factors of 13195 are 5, 7, 13 and 29.

#What is the largest prime factor of the number 600851475143 ?


N = 600851475143
LIM = 10 ** 6

def factor(n):
   #1. Create array of trial values
   a = np.ceil(np.sqrt(n))
   lim = min(n, LIM)
   a = np.arange(a, a + lim)
   b2 = a ** 2 - n

   #2. Check whether b is a square
```

```
    fractions = np.modf(np.sqrt(b2))[0]

    #3. Find 0 fractions
    indices = np.where(fractions == 0)

    #4. Find the first occurrence of a 0 fraction
    a = np.ravel(np.take(a, indices))[0]
                # Or a = a[indices][0]

    a = int(a)
    b = np.sqrt(a ** 2 - n)
    b = int(b)
    c = a + b
    d = a - b

    if c == 1 or d == 1:
       return

    print(c, d)
    factor(c)
    factor(d)

factor(N)
```

a

```
    a = np.ceil(np.sqrt(n))
    lim = min(n, LIM)
    a = np.arange(a, a + lim)
    b2 = a ** 2 - n
```

We used the `ceil()`

b

b modf()

function to get the fractional part of the `b`

```
    fractions = np.modf(np.sqrt(b2))[0]
```

3. Find `0` fractions.

   Call the `where()`
   fractional part is `0`

   ```
   indices = np.where(fractions == 0)
   ```

   occurrence of a zero fraction.

   First, call the `take()` `indices`
   `ravel()`

   ```
   a = np.ravel(np.take(a, indices))[0]
   ```

`a = a[indices][0].`

```
1234169 486847
1471 839
6857 71
```

How it works...

`ceil()`, `modf()`, `where()`, `ravel()`, and `take()`

Function	Description
`ceil()`	Calculates the `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ceil.html`)
`modf()`	Returns `http://docs.scipy.org/doc/numpy/reference/generated/numpy.modf.html`)
`where()`	Returns `http://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html`)
`ravel()`	Returns a see `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ravel.html`)
`take()`	element `http://docs.scipy.org/doc/numpy/reference/generated/numpy.take.html`)

A palindromic

How to do it...

palindromic.py

```
import numpy as np

#A palindromic number reads the same both ways.
#The largest palindrome made from the product of two 2-digit numbers
is 9009 = 91 x 99.

#Find the largest palindrome made from the product of two 3-digit
numbers.


#1. Create  3-digits numbers array
a = np.arange(100, 1000)
np.testing.assert_equal(100, a[0])
np.testing.assert_equal(999, a[-1])

#2. Create products array
numbers = np.outer(a, a)
numbers = np.ravel(numbers)
numbers.sort()
np.testing.assert_equal(810000, len(numbers))
np.testing.assert_equal(10000, numbers[0])
np.testing.assert_equal(998001, numbers[-1])

#3. Find largest palindromic number
for number in numbers[::-1]:
   s = str(numbers[i])

   if s == s[::-1]:
      print(s)
      break
```

We will create an
function, `arange()`.

`assert_equal()`

function from the `numpy.testing`

```
a = np.arange(100, 1000)
np.testing.assert_equal(100, a[0])
np.testing.assert_equal(999, a[-1])
```

`outer()`

`ravel()`

Call the `sort()`

```
numbers = np.outer(a, a)
numbers = np.ravel(numbers)
numbers.sort()
np.testing.assert_equal(810000, len(numbers))
np.testing.assert_equal(10000, numbers[0])
np.testing.assert_equal(998001, numbers[-1])
```

How it works...

We saw the `outer()` function
(`http://en.wikipedia.org/wiki/Outer_product`

`sort()`

There's more...

The steady state vector

A **Markov chain** is
chains, please refer to `http://en.wikipedia.org/wiki/Markov_chain`
time `t` depends on the state at time `t-1` `t-1`

, **up**,
and **down**

Far into the distant
`http://`
`en.wikipedia.org/wiki/Steady_state`

stochastic matrix (see `http://en.wikipedia.org/wiki/Stochastic_matrix`) *A*
`x`

$$Ax = x$$

Another `http://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors`

learning, and other sciences.

How to do it...

`steady_state_vector.py`

```
from __future__ import print_function
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np


today = date.today()
```

```
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('AAPL', start, today)
close =  [q[4] for q in quotes]

states = np.sign(np.diff(close))

NDIM = 3
SM = np.zeros((NDIM, NDIM))

signs = [-1, 0, 1]
k = 1

for i, signi in enumerate(signs):
   #we start the transition from the state with the specified sign
   start_indices = np.where(states[:-1] == signi)[0]

   N = len(start_indices) + k * NDIM

   # skip since there are no transitions possible
   if N == 0:
      continue

   #find the values of states at the end positions
   end_values = states[start_indices + 1]

   for j, signj in enumerate(signs):
      # number of occurrences of this transition
      occurrences = len(end_values[end_values == signj])
      SM[i][j] = (occurrences + k)/float(N)

print(SM)
eig_out = np.linalg.eig(SM)
print(eig_out)

idx_vec = np.where(np.abs(eig_out[0] - 1) < 0.1)
print("Index eigenvalue 1", idx_vec)

x = eig_out[1][:,idx_vec].flatten()
print("Steady state vector", x)
print("Check", np.dot(SM, x))
```

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

```
today = date.today()
start = (today.year - 1, today.month, today.day)
quotes = quotes_historical_yahoo('AAPL', start, today)
```

Select the close price.

```
close =  [q[4] for q in quotes]
```

3.

```
diff()
sign()                    -1                    1
and 0
```

```
states = np.sign(np.diff(close))
```

Initialize the stochastic matrix to `0` values.

Flat

Initialize the stochastic matrix with the `zeros()`

```
NDIM = 3
SM = np.zeros((NDIM, NDIM))
```

For each sign, select the corresponding start state indices.

Now the code

sign. Select the indices with the `where()` `k` is a smoothing

```
signs = [-1, 0, 1]
k = 1

for i, signi in enumerate(signs):
   #we start the transition from the state with the specified sign
    start_indices = np.where(states[:-1] == signi)[0]
```

Smoothing and the stochastic matrix.

additive smoothing (`http://en.wikipedia.org/wiki/Additive_smoothing`).

```
N = len(start_indices) + k * NDIM

# skip since there are no transitions possible
if N == 0:
    continue

#find the values of states at the end positions
end_values = states[start_indices + 1]

for j, signj in enumerate(signs):
    # number of occurrences of this transition
    occurrences = len
      (end_values[end_values == signj])
    SM[i][j] = (occurrences + k)/float(N)

print(SM)
```

```
[[ 0.5047619   0.00952381  0.48571429]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33774834  0.00662252  0.65562914]]
```

Eigenvalues and eigenvectors.

linalg

the eig()

```
eig_out = numpy.linalg.eig(SM)
print(eig_out)
```

eig()

```
(array([ 1.        ,  0.16709381,  0.32663057]), array([[
5.77350269e-01,   7.31108409e-01,   7.90138877e-04],
       [  5.77350269e-01,  -4.65117036e-01,  -9.99813147e-01],
       [  5.77350269e-01,  -4.99145907e-01,   1.93144030e-02]]))
```

1

1

0.9 and 1.1

```
idx_vec = np.where
  (np.abs(eig_out[0] - 1) < 0.1)
print("Index eigenvalue 1", idx_vec)

x = eig_out[1][:,idx_vec].flatten()
```

```
Index eigenvalue 1 (array([0]),)
Steady state vector [ 0.57735027  0.57735027  0.57735027]
Check [ 0.57735027  0.57735027  0.57735027]
```

How it works...

`diff()`, `sign()`, and `eig()` functions were

Function	Description
`diff()`	Calculates the `http://docs.scipy.org/doc/numpy/reference/generated/numpy.diff.html`).
`sign()`	Returns `http://docs.scipy.org/doc/numpy/reference/generated/numpy.sign.html`).
`eig()`	Returns the `http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eig.html`).

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

For the purpose of

`http://en.wikipedia.org/wiki/Power_law`

$$y \quad cx^{k}$$

principle (see `http://en.wikipedia.org/wiki/Pareto_principle`) for

want to initiate a

How to do it...

powerlaw.py

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np
import matplotlib.pyplot as plt

#1. Get close prices.
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('IBM', start, today)
close =  np.array([q[4] for q in quotes])

#2. Get positive log returns.
logreturns = np.diff(np.log(close))
pos = logreturns[logreturns > 0]

#3. Get frequencies of returns.
counts, rets = np.histogram(pos)
# 0 counts indices
indices0 = np.where(counts != 0)
rets = rets[:-1] + (rets[1] - rets[0])/2
# Could generate divide by 0 warning
freqs = 1.0/counts
freqs = np.take(freqs, indices0)[0]
rets = np.take(rets, indices0)[0]
freqs =  np.log(freqs)

#4. Fit the frequencies and returns to a line.
p = np.polyfit(rets,freqs, 1)

#5. Plot the results.
plt.title('Power Law')
plt.plot(rets, freqs, 'o', label='Data')
plt.plot(rets, p[0] * rets + p[1], label='Fit')
plt.xlabel('Log Returns')
plt.ylabel('Log Frequencies')
plt.legend()
plt.grid()
plt.show()
```

Now calculate the log returns for the close prices. For more information on log returns, refer to `http://en.wikipedia.org/wiki/Rate_of_return`.

of these values with the `diff()`

```
logreturns = np.diff(np.log(close))
pos = logreturns[logreturns > 0]
```

`histogram()` function.

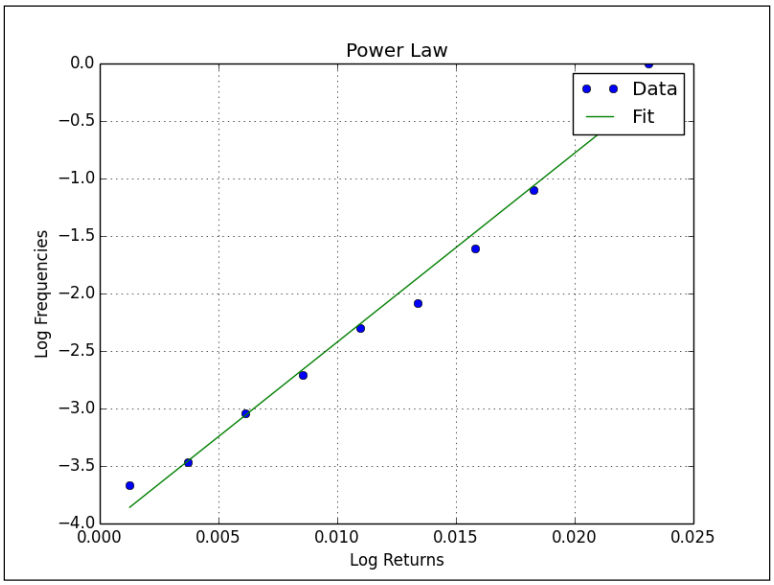
```
counts, rets = np.histogram(pos)
# 0 counts indices
indices0 = np.where(counts != 0)
rets = rets[:-1] + (rets[1] - rets[0])/2
# Could generate divide by 0 warning
freqs = 1.0/counts
freqs = np.take(freqs, indices0)[0]
rets = np.take(rets, indices0)[0]
freqs =  np.log(freqs)
```

3.

`polyfit()`

```
p = np.polyfit(rets,freqs, 1)
```

```
plt.title('Power Law')
plt.plot(rets, freqs, 'o', label='Data')
plt.plot(rets, p[0] * rets + p[1], label='Fit')
plt.xlabel('Log Returns')
plt.ylabel('Log Frequencies')
plt.legend()
plt.grid()
plt.show()
```



How it works...

`histogram()` function calculates the histogram of a dataset. It returns the histogram
`polyfit()`
this case, we

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

documentation page for the `histogram()` function at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html`

documentation page for the `polyfit()` function at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html`

[redacted]

http://en.wikipedia.org/wiki/Rate_of_return) of

[redacted]

See also section for the corresponding recipes.

How to do it...

periodic.py

```
from __future__ import print_function
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt

#1. Get close prices.
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('AAPL', start, today)
close =  np.array([q[4] for q in quotes])

#2. Get log returns.
logreturns = np.diff(np.log(close))

#3. Calculate breakout and pullback
```

```
freq = 0.02
breakout = scipy.stats.scoreatpercentile(logreturns, 100 * (1 - freq)
)
pullback = scipy.stats.scoreatpercentile(logreturns, 100 * freq)

#4. Generate buys and sells
buys = np.compress(logreturns < pullback, close)
sells = np.compress(logreturns > breakout, close)
print(buys)
print(sells)
print(len(buys), len(sells))
print(sells.sum() - buys.sum())

#5. Plot a histogram of the log returns
plt.title('Periodic Trading')
plt.hist(logreturns)
plt.grid()
plt.xlabel('Log Returns')
plt.ylabel('Counts')
plt.show()
```

`scoreatpercentile()` function,

```
freq = 0.02
breakout = scipy.stats.scoreatpercentile
  (logreturns, 100 * (1 - freq) )
pullback = scipy.stats.scoreatpercentile
  (logreturns, 100 * freq)
```

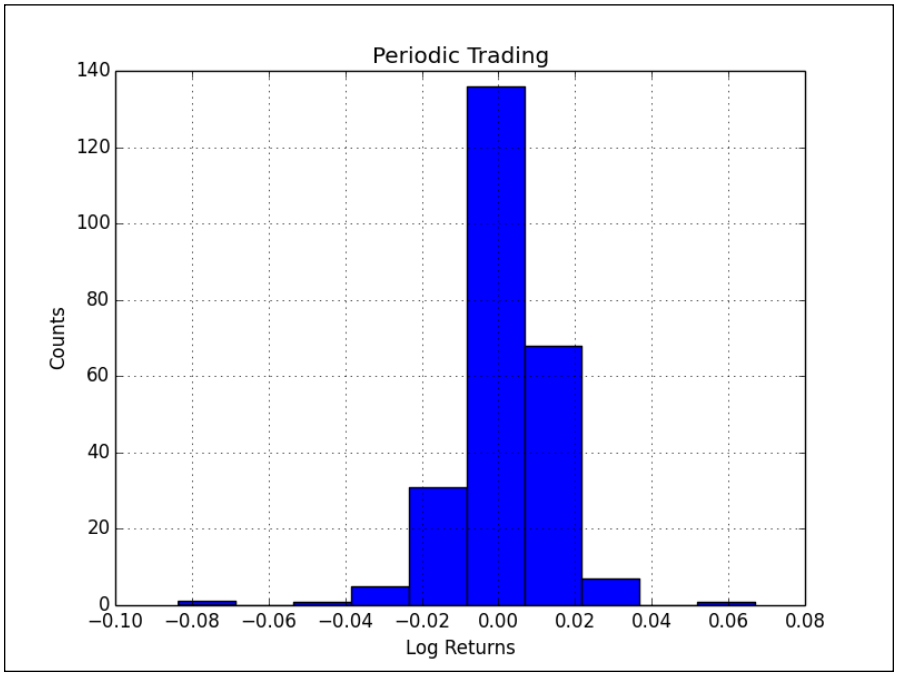
`compress()`

```
buys = np.compress(logreturns < pullback, close)
sells = np.compress(logreturns > breakout, close)
print(buys)
print(sells)
print(len(buys), len(sells))
print(sells.sum() - buys.sum())
```

```
[  77.76375466   76.69249773  102.72        101.2          98.57
]
[ 74.95502967  76.55980292  74.13759123  80.93512599  98.22      ]
5 5
-52.1387025726
```

3. of the log returns.

```
plt.title('Periodic Trading')
plt.hist(logreturns)
plt.grid()
plt.xlabel('Log Returns')
plt.ylabel('Counts')
plt.show()
```



How it works...

We encountered the `compress()`
elements of the input that

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

Chapter 2, *Advanced Indexing and Array Concepts*

documentation page for the `compress()` function at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.compress.html`

In the previous

See also section of the corresponding recipe.

How to do it...

`random_periodic.py`

```
from __future__ import print_function
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np
import matplotlib.pyplot as plt

def get_indices(high, size):
   #2. Generate random indices
   return np.random.randint(0, high, size)

#1. Get close prices.
```

```
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('AAPL', start, today)
close =  np.array([q[4] for q in quotes])

nbuys = 5
N = 2000
profits = np.zeros(N)

for i in xrange(N):
   #3. Simulate trades
   buys = np.take(close, get_indices(len(close), nbuys))
   sells = np.take(close, get_indices(len(close), nbuys))
   profits[i] = sells.sum() - buys.sum()

print("Mean", profits.mean())
print("Std", profits.std())

#4. Plot a histogram of the profits
plt.title('Simulation')
plt.hist(profits)
plt.xlabel('Profits')
plt.ylabel('Counts')
plt.grid()
plt.show()
```

First we need an

```
                                    randint()


   return np.random.randint(0, high, size)
```

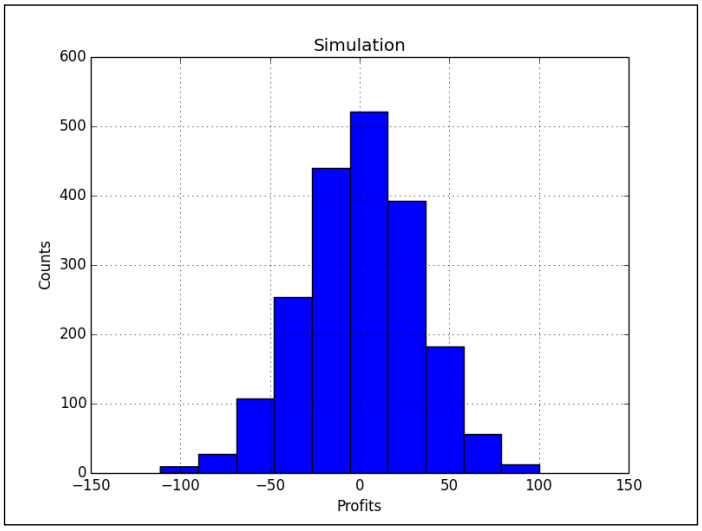
Simulate trades.

```
   take()

   buys = np.take(close, get_indices(len(close), nbuys))
   sells = np.take(close, get_indices(len(close), nbuys))
   profits[i] = sells.sum() - buys.sum()
```

3.

```
   plt.title('Simulation')
```

```
plt.hist(profits)
plt.xlabel('Profits')
plt.ylabel('Counts')
plt.grid()
plt.show()
```



How it works...

We used the `randint()` `numpy.random` module contains

Function	Description
`rand()`	Creates on dimension parameters. If no dimensions are specified, a single float is returned (see `http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.rand.html`).
`randn()`	Sample `0` and variance `1`. `rand()` (see `http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html`).
`randint()`	Returns an optional output shape (see `http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randint.html`).

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

Eratosthenes

Sieve of Eratosthenes (see `http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`

How to do it...

`arange()`

```
a = np.arange(i, i + LIM, 2)
```

Sieve out the multiples of `p`.

`p`

```
a = a[a % p != 0]
```

```
from __future__ import print_function
import numpy as np

LIM = 10 ** 6
N = 10 ** 9
P = 10001
primes = []
p = 2

#By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13,
we can see that the 6th prime is 13.

#What is the 10 001st prime number?

def sieve_primes(a, p):
```

```
    #2. Sieve out multiples of p
    a = a[a % p != 0]

    return a

for i in xrange(3, N, LIM):
    #1. Create a list of consecutive integers
    a = np.arange(i, i + LIM, 2)

    while len(primes) < P:
        a = sieve_primes(a, p)
        primes.append(p)

        p = a[0]

print(len(primes), primes[P-1])
```

4

Interfacing with R

Introduction

We will go into the details of exchanging data with these environments.

buffer interface can expose

Python Imaging Library (**PIL**).

See also section of this recipe for instructions.

How to do it...

buffer.py

```
import numpy as np
import Image #from PIL import Image (Python 3)
import scipy.misc

lena = scipy.misc.lena()
data = np.zeros((lena.shape[0], lena.shape[1], 4), dtype=np.int8)
data[:,:,3] = lena.copy()
img = Image.frombuffer("RGBA", lena.shape, data, 'raw', "RGBA", 0, 1)
img.save('lena_frombuffer.png')

data[:,:,3] = 255
data[:,:,0] = 222
img.save('lena_modified.png')
```

```
lena = scipy.misc.lena()
data = np.zeros((lena.shape[0], lena.shape[1], 4), dtype=numpy.
int8)
data[:,:,3] = lena.copy()
```

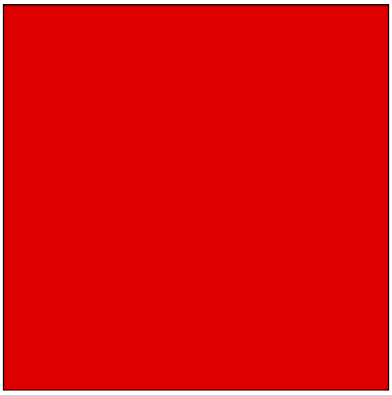
```
img = Image.frombuffer("RGBA", lena.shape, data, 'raw', "RGBA", 0,
1)
img.save('lena_frombuffer.png')
```

3.

```
data[:,:,3] = 255
data[:,:,0] = 222
img.save('lena_modified.png')
```

In computer graphics, the position of the origin is different than in

or image, and the *y* axis goes down (see `http://en.wikipedia.org/wiki/2D_computer_graphics#Non-standard_orientation_of_the_coordinate_system`).



How it works...

How to do it...

array_interface.py

```
from __future__ import print_function
import numpy as np
import Image
import scipy.misc

lena = scipy.misc.lena()
data = np.zeros((lena.shape[0], lena.shape[1], 4), dtype=np.int8)
data[:,:,3] = lena.copy()
img = Image.frombuffer("RGBA", lena.shape, data, 'raw', "RGBA", 0, 1)
array_interface = img.__array_interface__
print("Keys", array_interface.keys())
```

```
print("Shape", array_interface['shape'])
print("Typestr", array_interface['typestr'])

numpy_array = np.asarray(img)
print("Shape", numpy_array.shape)
print("Data type", numpy_array.dtype)
```

__array_interface__

```
array_interface = img.__array_interface__
print("Keys", array_interface.keys())
print("Shape", array_interface['shape'])
print("Typestr", array_interface['typestr'])
```

```
Keys ['shape', 'data', 'typestr']
Shape (512, 512, 4)
Typestr |u1
```

ndarray __array_interface__ asarray()

```
numpy_array = np.asarray(img)
print("Shape", numpy_array.shape)
print("Data type", numpy_array.dtype)
```

```
Shape (512, 512, 4)
Data type uint8
```

How it works...

Using the buffer protocol in this chapter

http://docs.scipy.org/doc/numpy/reference/arrays.interface.html

and its open source alternative, Octave, are
`scipy.io` `savemat()`
a `.mat`

Installing
pointers for installing at `http://www.gnu.org/software/octave/download.html`.
See also section of this recipe, for instructions on installing

How to do it...

`octave.py`

```
import numpy as np
import scipy.io

a = np.arange(7)

scipy.io.savemat("a.mat", {"array": a})
```

`savemat()` `.mat`

```
a = np.arange(7)
scipy.io.savemat("a.mat", {"array": a})
```

```
octave-3.4.0:2> load a.mat
octave-3.4.0:3> array
array =

  0
  1
```

```
2
3
4
5
6
```

Installing SciPy in *Chapter 2*, *Advanced Indexing and Array Concepts*

documentation for the `savemat()` function at `http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.io.savemat.html`

R is a popular **RPy2** is an interface

How to do it...

Installing with `pip` or `easy_install`

```
$ easy_install rpy2
```

```
$ sudo pip install rpy2
$ pip freeze|grep rpy2
rpy2==2.4.2
```

Installing from source `tar.gz`

```
$ tar -xzf <rpy2_package>.tar.gz
$ cd <rpy2_package>
$ python setup.py build install
```

programming language homepage at `http://www.r-project.org/`

`http://rpy.sourceforge.net/`

some sample R datasets and plot the data of one of them.

How to do it...

rdatasets.py

```
from rpy2.robjects.packages import importr
import numpy as np
import matplotlib.pyplot as plt

datasets = importr('datasets')
mtcars = datasets.__rdata__.fetch('mtcars')['mtcars']

plt.title('R mtcars dataset')
plt.xlabel('wt')
plt.ylabel('mpg')
plt.plot(mtcars)
plt.grid(True)
plt.show()
```

motorcars https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html

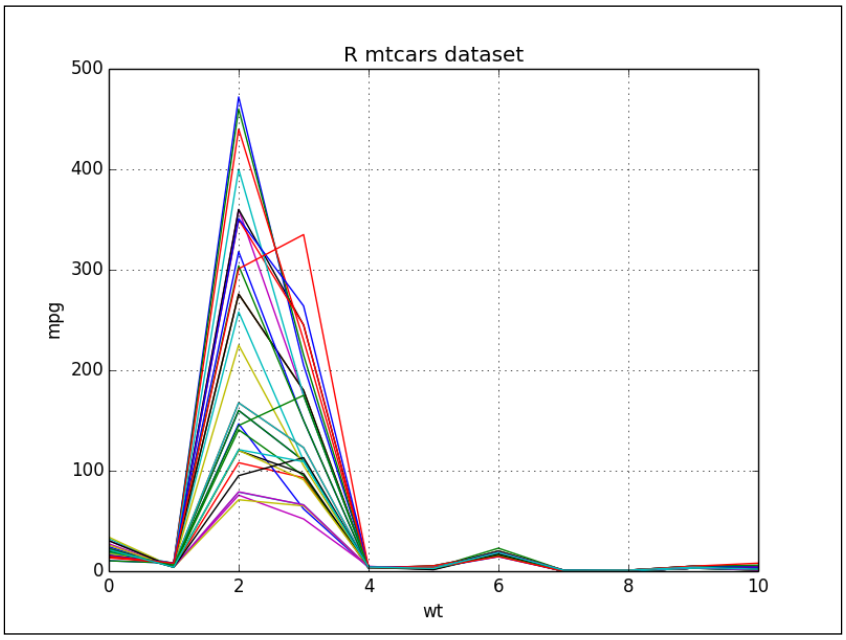
importr()

import R

mtcars

```
datasets = importr('datasets')
mtcars = np.array(datasets.mtcars)


plt.plot(mtcars)
plt.show()
```

miles per gallon (**mpg**) and **weight** (**wt**



Installing matplotlib in *Chapter 1*, *Winding Along with IPython*

Jython is the
the **Java virtual machine** (**JVM**
JPype
modules, which are

How to do it...

`http://sourceforge.net/projects/jpype/files/`.

```
$ python setup.py install
```

In this

How to do it...

hellojpype.py

```
import jpype
import numpy as np

#1. Start the JVM
jpype.startJVM(jpype.getDefaultJVMPath())

#2. Print hello world
jpype.java.lang.System.out.println("hello world")

#3. Send a NumPy array
values = np.arange(7)
java_array = jpype.JArray(jpype.JDouble, 1)(values.tolist())

for item in java_array:
   jpype.java.lang.System.out.println(item)

#4. Shutdown the JVM
jpype.shutdownJVM()
```

```
jpype.startJVM(jpype.getDefaultJVMPath())
```

"hello world"

```
jpype.java.lang.System.out.println("hello world")
```

3.

```
values = np.arange(7)
java_array = jpype.JArray
  (jpype.JDouble, 1)(values.tolist())

for item in java_array:
```

```
    jpype.java.lang.System.out.
      println(item)
```

After

```
jpype.shutdownJVM()
```

```
hello world
0.0
1.0
2.0
3.0
4.0
5.0
6.0
JVM activity report     :
  classes loaded       : 31
JVM has been shutdown
```

How it works...

Java Native Interface (**JNI**

of that fact.

Installing JPype in this chapter

homepage at `http://jpype.sourceforge.net/`

Google App Engine (**GAE**

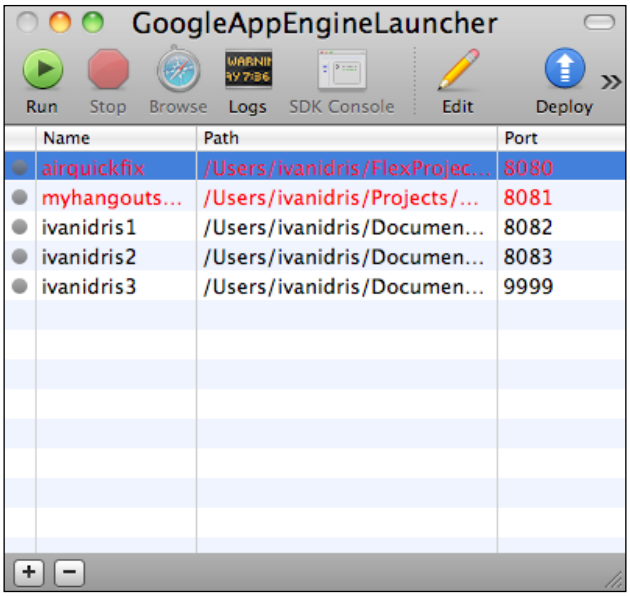
How to do it...

`https://developers.google.com/appengine/downloads`.

in the **Preferences** dialog of the launcher application.

```
dev_appserver.py
appcfg.py
```

Run and **Deploy**



How to do it...

Create a new application with the launcher (**File** | **New Application**). Name it `numpycloud`

```
app.yaml
favicon.ico
index.yaml
main.py
```

Add

First we
to the `app.yaml`

```
- name: NumPy
  version: "1.6.1"
```

```
application: numpycloud
version: 1
runtime: python27
api_version: 1
threadsafe: yes

handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: .*
  script: main.app

libraries:
```

```
- name: webapp2
  version: "2.5.1"
- name: numpy
  version: "1.6.1"
```

3. main.py

a MainHandler

```
def get(self):
    self.response.out.write
      ('Hello world!<br/>')
    self.response.out.write
      ('NumPy sum = ' + str
      (numpy.arange(7).sum()))
```

```
import webapp2
import numpy

class MainHandler(webapp2.RequestHandler):
    def get(self):
      self.response.out.write('Hello world!<br/>')
      self.response.out.write('NumPy sum = ' +
        str(numpy.arange(7).sum()))

app = webapp2.WSGIApplication([('/', MainHandler)],
                              debug=True)
```

Browse dev_appserver.py

```
Hello world!
NumPy sum = 21
```

How it works...

[redacted]

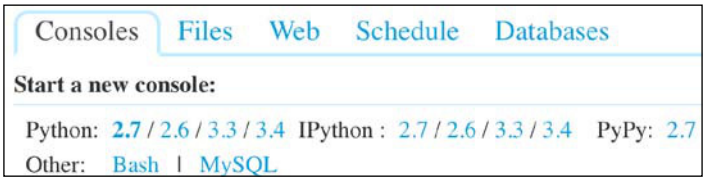
In *Chapter 1*, *Winding Along with IPython*

free, at least

see `https://www.pythonanywhere.com/batteries_included/`.

How to do it...

Once we **PythonAnywhere**



Write

```
from __future__ import print_function
import urllib2
import re
import time
import numpy as np

prices = np.array([])

for i in xrange(3):
   req = urllib2.Request('http://finance.google.com/finance/
info?client=ig&q=AAPL')
   req.add_header('User-agent', 'Mozilla/5.0')
   response = urllib2.urlopen(req)
   page = response.read()
   m = re.search('l_cur" : "(.*)"', page)
   prices = np.append(prices, float(m.group(1)))
   avg = prices.mean()
```

```
stddev = prices.std()

devFactor = 1
bottom = avg - devFactor * stddev
top = avg + devFactor * stddev
timestr = time.strftime("%H:%M:%S", time.gmtime())

print(timestr, "Average", avg, "-Std", bottom, "+Std", top)
time.sleep(60)
```

containing prices and calculate the mean and standard deviation of the prices.

the code.

After we are done with the code on our local machine, we can upload the script

Files

3. **Consoles** **Bash**

```
18:32 ~/cookbook $ python avg_price.py AAPL 1
15:31:29 Average 667.24 -Std 667.24 +Std 667.24
15:32:29 Average 667.365 -Std 667.24 +Std 667.49
15:33:29 Average 667.376666667 -Std 667.273279584 +Std 667.480053749
```

How it works...

5

Adding images

Blurring images

Repeating audio fragments

Introduction

for it and concentrate on having fun. In *Chapter 10*, *Fun with Scikits*
more image processing recipes that use `scikits-image`

appreciate the recipes.

[redacted]

It is

[redacted]

See also section has a reference to the corresponding recipe.

How to do it...

We will

An

```
img = np.zeros((N, N), np.uint8)
NSQUARES = 30
centers = np.random.random_integers(0, N, size=(NSQUARES, 2))
radii = np.random.randint(0, N/9, size=NSQUARES)
colors = np.random.randint(100, 255, size=NSQUARES)
```

initialized with functions from the `numpy.random` random integers.

in the previous step. With the `clip()`

`meshgrid()` function gives us the

`N` and `M`, it

along the *x* *y* axis.

```
In: x = linspace(1, 3, 3)

In: x
Out: array([ 1.,  2.,  3.])
In: y = linspace(1, 2, 2)

In: y
Out: array([ 1.,  2.])

In: meshgrid(x, y)
Out:
[array([[ 1.,  2.,  3.],
        [ 1.,  2.,  3.]]),
 array([[ 1.,  1.,  1.],
        [ 2.,  2.,  2.]])]
```

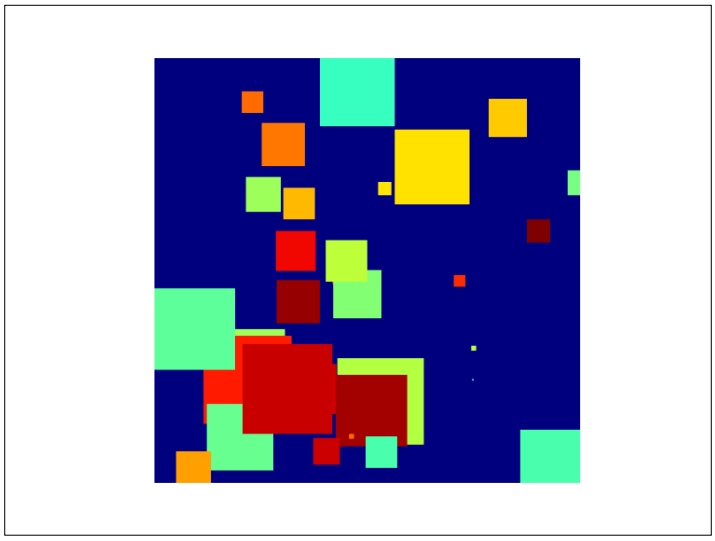
3.

```
for i in xrange(NSQUARES):
   xindices = range(centers[i][0] - radii[i], centers[i][0]
+ radii[i])
   xindices = np.clip(xindices, 0, N - 1)
   yindices = range(centers[i][1] - radii[i], centers[i][1]
+ radii[i])
   yindices = np.clip(yindices, 0, N - 1)

   if len(xindices) == 0 or len(yindices) == 0:
      continue
   coordinates = np.meshgrid(xindices, yindices)
   img[coordinates] = colors[i]
```

Before we
with the `tofile()`
map with the `memmap()`

```
img.tofile('random_squares.raw')
img_memmap = np.memmap('random_squares.raw', shape=img.shape)
```

```
plt.imshow(img_memmap)
plt.axis('off')
plt.show()
```



Here is the complete source code for this recipe from the `memmap.py`

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

N = 512
NSQUARES = 30

# Initialize
img = np.zeros((N, N), np.uint8)
centers = np.random.random_integers(0, N, size=(NSQUARES, 2))
radii = np.random.randint(0, N/9, size=NSQUARES)
colors = np.random.randint(100, 255, size=NSQUARES)

# Generate squares
for i in xrange(NSQUARES):
```

```
    xindices = range(centers[i][0] - radii[i], centers[i][0]
+ radii[i])
    xindices = np.clip(xindices, 0, N - 1)
    yindices = range(centers[i][1] - radii[i], centers[i][1]
+ radii[i])
    yindices = np.clip(yindices, 0, N - 1)

    if len(xindices) == 0 or len(yindices) == 0:
       continue

    coordinates = np.meshgrid(xindices, yindices)
    img[coordinates] = colors[i]

# Load into memory map
img.tofile('random_squares.raw')
img_memmap = np.memmap('random_squares.raw', shape=img.shape)

# Display image
plt.imshow(img_memmap)
plt.axis('off')
plt.show()
```

How it works...

We

Function	Description
`zeros()`	
`random_integers()`	
`randint()`	`random_` `integers()` a closed interval.
`clip()`	maximum.
`meshgrid()`	*x* *y* coordinates.
`tofile()`	
`memmap()`	
`axis()`	axes. For instance, we can turn them off.

Installing matplotlib in *Chapter 1*, *Winding Along with IPython*

at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.memmap.html`

In this recipe, we **Mandelbrot fractal** (see `http://en.wikipedia.org/wiki/Mandelbrot_set`

covered in this recipe.

See also section has a reference to the related recipe.

How to do it...

Initialize the `x`, `y`, and `z` `meshgrid()`, `zeros()`, and `linspace()`

```
x, y = np.meshgrid(np.linspace(x_min, x_max, SIZE),
                   np.linspace(y_min, y_max, SIZE))
c = x + 1j * y
z = c.copy()
fractal = np.zeros(z.shape, dtype=np.uint8) + MAX_COLOR
```

If `z`

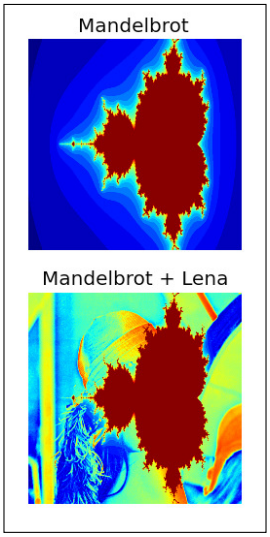
$$z_{n+1} = z_n^2 + c$$

Here, `c`

escape time algorithm

of these points is used as the `c`

```
for n in range(ITERATIONS):
    print(n)
    mask = numpy.abs(z) <= 4
    z[mask] = z[mask] ** 2 +  c[mask]
    fractal[(fractal == MAX_COLOR) & (-mask)] = (MAX_COLOR - 1) *
n / ITERATIONS
Plot the fractal with matplotlib:
plt.subplot(211)
plt.imshow(fractal)
plt.title('Mandelbrot')
plt.axis('off')
Use the choose() function to pick a value from the fractal or Lena
array:
plt.subplot(212)
plt.imshow(numpy.choose(fractal < lena, [fractal, lena]))
plt.axis('off')
plt.title('Mandelbrot + Lena')
```



following is the complete code for this recipe from the `mandelbrot.py`

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import lena

ITERATIONS = 10
lena = lena()
SIZE = lena.shape[0]
MAX_COLOR = 255.
x_min, x_max = -2.5, 1
y_min, y_max = -1, 1

# Initialize arrays
x, y = np.meshgrid(np.linspace(x_min, x_max, SIZE),
                   np.linspace(y_min, y_max, SIZE))
c = x + 1j * y
z = c.copy()
fractal = np.zeros(z.shape, dtype=np.uint8) + MAX_COLOR
# Generate fractal
for n in range(ITERATIONS):
    mask = np.abs(z) <= 4
    z[mask] = z[mask] ** 2 +  c[mask]
    fractal[(fractal == MAX_COLOR) & (-mask)] = (MAX_COLOR - 1) *
n / ITERATIONS

# Display the fractal
plt.subplot(211)
plt.imshow(fractal)
plt.title('Mandelbrot')
plt.axis('off')

# Combine with lena
plt.subplot(212)
plt.imshow(np.choose(fractal < lena, [fractal, lena]))
plt.axis('off')
plt.title('Mandelbrot + Lena')

plt.show()
```

How it works...

Function	Description
`linspace()`	
`choose()`	on a condition
`meshgrid()`	*x* *y* coordinates

Installing matplotlib recipe in *Chapter 1*, *Winding Along with IPython*

Installing SciPy recipe in *Chapter 2*, *Advanced Indexing and Arrays*

`http://en.wikipedia.org/wiki/Gaussian_filter`

and a spiral `http://en.wikipedia.org/wiki/Polar_coordinate_system`

How to do it...

```
NFIGURES = 5
k = np.random.random_integers(1, 5, NFIGURES)
a = np.random.random_integers(1, 5, NFIGURES)
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
```

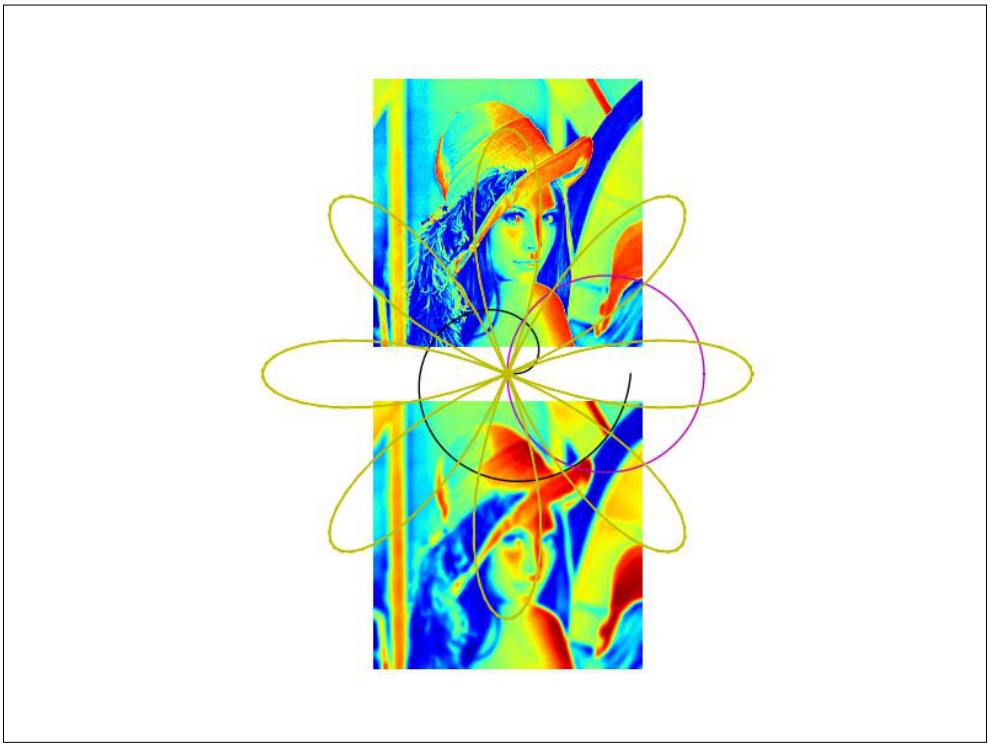
4

```
plt.subplot(212)
blurred = scipy.ndimage.gaussian_filter(lena, sigma=4)
```

```
plt.imshow(blurred)
plt.axis('off')
```

3. `polar()`

```
theta = np.linspace(0, k[0] * np.pi, 200)
plt.polar(theta, np.sqrt(theta), choice(colors))

for i in xrange(1, NFIGURES):
   theta = np.linspace(0, k[i] * np.pi, 200)
   plt.polar(theta, a[i] * np.cos(k[i] * theta), choice(colors))
```



Here is the complete code for this recipe from the `spiral.py`

```
import numpy as np
import matplotlib.pyplot as plt
from random import choice
import scipy
import scipy.ndimage
```

```
# Initialization
NFIGURES = 5
k = np.random.random_integers(1, 5, NFIGURES)
a = np.random.random_integers(1, 5, NFIGURES)

colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

lena = scipy.misc.lena()
plt.subplot(211)
plt.imshow(lena)
plt.axis('off')

# Blur Lena
plt.subplot(212)
blurred = scipy.ndimage.gaussian_filter(lena, sigma=4)

plt.imshow(blurred)
plt.axis('off')

# Plot in polar coordinates
theta = np.linspace(0, k[0] * np.pi, 200)
plt.polar(theta, np.sqrt(theta), choice(colors))

for i in xrange(1, NFIGURES):
   theta = np.linspace(0, k[i] * np.pi, 200)
   plt.polar(theta, a[i] * np.cos(k[i] * theta), choice(colors))

plt.axis('off')

plt.show()
```

How it works...

We made use of the

Function	Description
`gaussian_filter()`	
`random_integers()`	
`polar()`	

scipy.ndimage http://docs.scipy.org/doc/scipy-0.14.0/reference/ndimage.html

As we saw in *Chapter 2*, *Advanced Indexing and Array Concepts*, we can do neat things with WAV urllib2

Chapter 2, *Advanced Indexing and Array Concepts*.

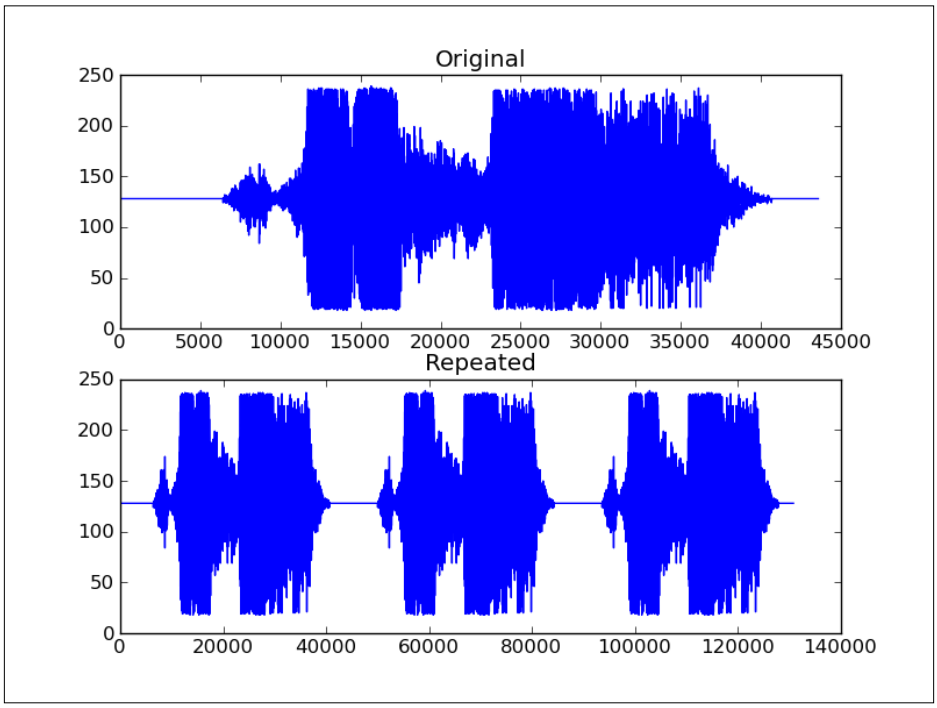
How to do it...

repeat() function, in this case, it is more appropriate to use the tile() repeat() function would have the effect of enlarging

```
In: x = array([1, 2])

In: x
Out: array([1, 2])

In: repeat(x, 3)
Out: array([1, 1, 1, 2, 2, 2])

In: tile(x, 3)
Out: array([1, 2, 1, 2, 1, 2])
```

tile()

```
repeated = np.tile(data, 3)


plt.title("Repeated")
plt.plot(repeated)
```



Here is the complete code for this recipe from the `repeat_audio.py`

```
import scipy.io.wavfile
import matplotlib.pyplot as plt
import urllib2
import numpy as np

response = urllib2.urlopen('http://www.thesoundarchive.com/
austinpowers/smashingbaby.wav')
print(response.info())
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print("Data type", data.dtype, "Shape", data.shape)
```

```
plt.subplot(2, 1, 1)
plt.title("Original")
plt.plot(data)

plt.subplot(2, 1, 2)

# Repeat the audio fragment
repeated = np.tile(data, 3)

# Plot the audio data
plt.title("Repeated")
plt.plot(repeated)
scipy.io.wavfile.write("repeated_yababy.wav",
    sample_rate, repeated)

plt.show()
```

How it works...

Function	Description
`scipy.io.wavfile.read()`	
`numpy.tile()`	
`scipy.io.wavfile.write()`	specified sample rate

`scipy.io` `http://docs.scipy.org/doc/scipy-0.14.0/reference/io.html`

http://en.wikipedia.org/wiki/Piano_key_frequencies

$$440 \cdot 2^{\frac{n-49}{12}}$$

Here, n
the amplitude, duration, and phase at random.

How to do it...

Initialize to

200 2000

0.01 0.2

0 and 2 pi

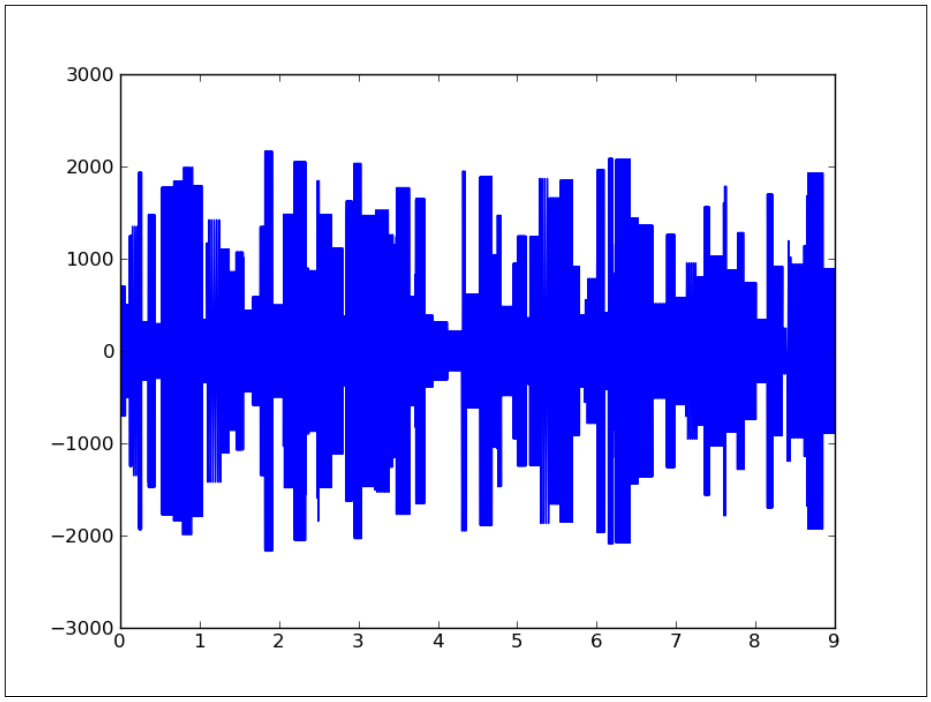
```
NTONES = 89
amps = 2000. * np.random.random((NTONES,)) + 200.
durations = 0.19 * np.random.random((NTONES,)) + 0.01
keys = np.random.random_integers(1, 88, NTONES)
freqs = 440.0 * 2 ** ((keys - 49.)/12.)
phi = 2 * np.pi * np.random.random((NTONES,))
```

Write a generate()

```
def generate(freq, amp, duration, phi):
 t = np.linspace(0, duration, duration * RATE)
 data = np.sin(2 * np.pi * freq * t + phi) * amp

 return data.astype(DTYPE)
```

3.

```
for i in xrange(NTONES):
   newtone = generate(freqs[i], amp=amps[i],
duration=durations[i], phi=phi[i])
   tone = np.concatenate((tone, newtone))


plt.plot(np.linspace(0, len(tone)/RATE, len(tone)), tone)
plt.show()
```



tone_generation.py

```
import scipy.io.wavfile
import numpy as np
```

```
import matplotlib.pyplot as plt

RATE = 44100
DTYPE = np.int16

# Generate sine wave
def generate(freq, amp, duration, phi):
 t = np.linspace(0, duration, duration * RATE)
 data = np.sin(2 * np.pi * freq * t + phi) * amp

 return data.astype(DTYPE)

# Initialization
NTONES = 89
amps = 2000. * np.random.random((NTONES,)) + 200.
durations = 0.19 * np.random.random((NTONES,)) + 0.01
keys = np.random.random_integers(1, 88, NTONES)
freqs = 440.0 * 2 ** ((keys - 49.)/12.)
phi = 2 * np.pi * np.random.random((NTONES,))

tone = np.array([], dtype=DTYPE)

# Compose
for i in xrange(NTONES):
   newtone = generate(freqs[i], amp=amps[i],
duration=durations[i], phi=phi[i])
   tone = np.concatenate((tone, newtone))

scipy.io.wavfile.write('generated_tone.wav', RATE, tone)

# Plot audio data
plt.plot(np.linspace(0, len(tone)/RATE, len(tone)), tone)
plt.show()
```

How it works...

`concatenate()` function was used to concatenate sine waves.

`concatenate()` function is documented at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.concatenate.html`

in software than in hardware.

How to do it...

`iirdesign()`

`scipy.signal`

`iirdesign()` function of the `scipy.signal`
module. IIR
`http://en.wikipedia.org/wiki/Infinite_impulse_response`.
We are not going to go into all the details of the `iirdesign()`
at the documentation at `http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirdesign.html`

Maximum loss

Minimum attenuation

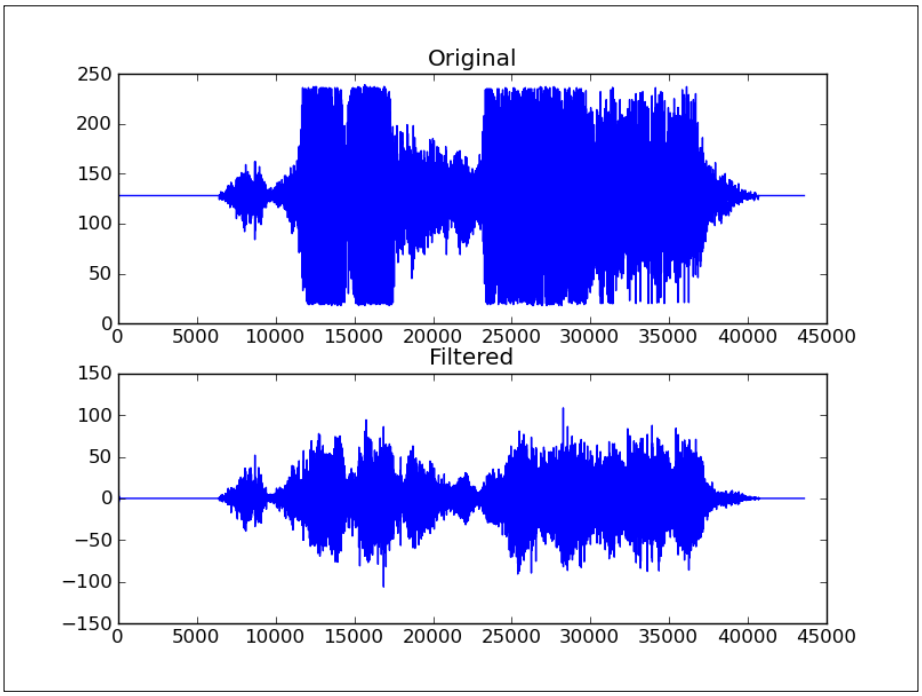
```
b,a = scipy.signal.iirdesign(wp=0.2, ws=0.1, gstop=60,
 gpass=1, ftype='butter')
```

(`http://en.wikipedia.org/wiki/Butterworth_filter` **Butterworth**

`scipy.signal.lfilter()` function. It accepts as

```
filtered = scipy.signal.lfilter(b, a, data)
```

```
scipy.io.wavfile.write('filtered.wav', sample_rate, filtered.
astype(data.dtype))
```




```
import scipy.io.wavfile
import matplotlib.pyplot as plt
import urllib2
import scipy.signal

response =urllib2.urlopen('http://www.thesoundarchive.com/
austinpowers/smashingbaby.wav')
print response.info()
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print("Data type", data.dtype, "Shape", data.shape)

plt.subplot(2, 1, 1)
plt.title("Original")
```

```
plt.plot(data)

# Design the filter
b,a = scipy.signal.iirdesign(wp=0.2, ws=0.1, gstop=60, gpass=1,
ftype='butter')

# Filter
filtered = scipy.signal.lfilter(b, a, data)

# Plot filtered data
plt.subplot(2, 1, 2)
plt.title("Filtered")
plt.plot(filtered)

scipy.io.wavfile.write('filtered.wav', sample_rate, filtered.
astype(data.dtype))

plt.show()
```

How it works...

We created and

Function	Description
`scipy.signal.iirdesign()`	Creates an parameter list, which is documented at `http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirdesign.html`.
`scipy.signal.lfilter()`	

Sobel operator (`http://en.wikipedia.org/wiki/Sobel_operator`

How to do it...

x direction, set the axis parameter to `0`

```
sobelx = scipy.ndimage.sobel(lena, axis=0, mode='constant')
```

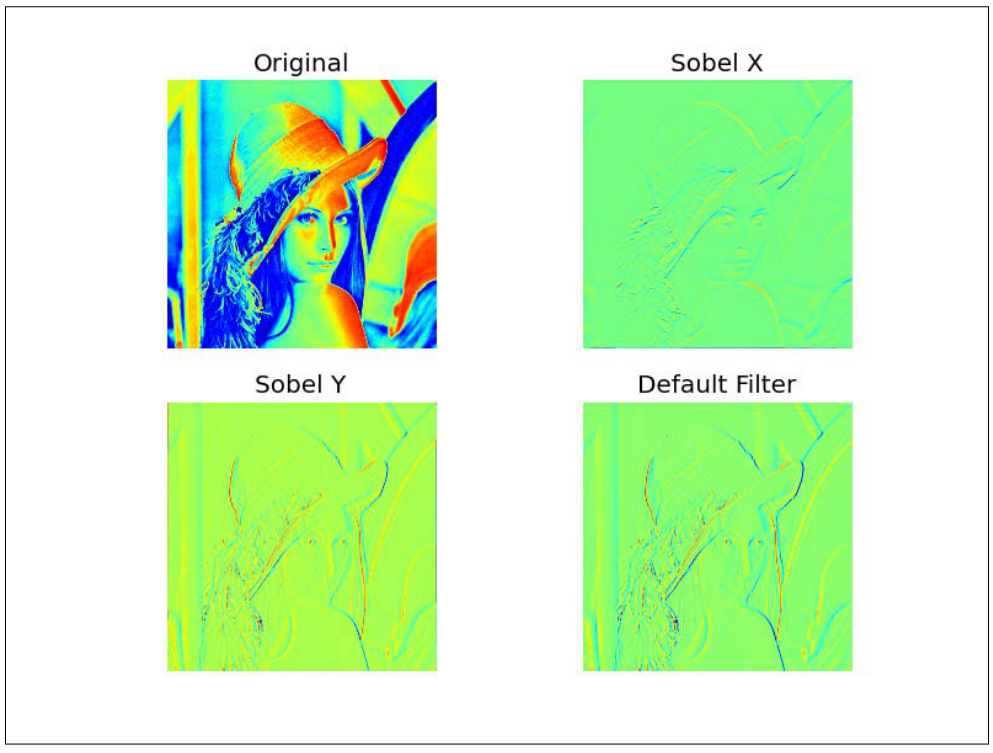
y direction, set the axis parameter to `1`

```
sobely = scipy.ndimage.sobel(lena, axis=1, mode='constant')
```

3.

```
default = scipy.ndimage.sobel(lena)
```

Here are the original and resulting image plots, showing edge detection with the



```
import scipy
import scipy.ndimage
import matplotlib.pyplot as plt

lena = scipy.misc.lena()

plt.subplot(221)
plt.imshow(lena)
plt.title('Original')
plt.axis('off')

# Sobel X filter
sobelx = scipy.ndimage.sobel(lena, axis=0, mode='constant')

plt.subplot(222)
plt.imshow(sobelx)
plt.title('Sobel X')
plt.axis('off')

# Sobel Y filter
sobely = scipy.ndimage.sobel(lena, axis=1, mode='constant')

plt.subplot(223)
plt.imshow(sobely)
plt.title('Sobel Y')
plt.axis('off')

# Default Sobel filter
default = scipy.ndimage.sobel(lena)

plt.subplot(224)
plt.imshow(default)
plt.title('Default Filter')
plt.axis('off')

plt.show()
```

How it works...

We applied the

axis independent.

6

Creating a universal function

`chararray`

Ignoring negative and extreme values

`recarray` function

Introduction

Universal functions (Ufuncs)

We can create a function.

`frompyfunc()`

How to do it...

```
def double(a):
    return 2 * a
```

Create the universal function with `frompyfunc()`

1

```
from __future__ import print_function
import numpy as np

def double(a):
   return 2 * a

ufunc = np.frompyfunc(double, 1, 1)
print("Result", ufunc(np.arange(4)))
```

```
Result [0 2 4 6]
```

How it works...

`frompyfunc()`

were written in C.

documentation of the `frompyfunc()` `http://docs.scipy.org/doc/numpy/reference/generated/numpy.frompyfunc.html`

Pythagorean triple

(`http://en.wikipedia.org/wiki/Pythagorean_triple`

$a^2 + b^2 = c^2$.

$3^2 + 4^2 = 5^2$.

Pythagorean Theorem

$$a = m^2 - n^2, b = 2mn, c = m^2 + n^2$$

In this example, we will see universal functions in action.

How to do it...

`m` and `n` indices.

```
m = np.arange(33)
n = np.arange(33)
```

`a`, `b`, and `c`
`outer()` function to get the Cartesian products,

```
a = np.subtract.outer(m ** 2, n ** 2)
b = 2 * np.multiply.outer(m, n)
c = np.add.outer(m ** 2, n ** 2)
```

3. `a`, `b`, and `c` values. However, we still

those values with the `where()`

```
idx =  np.where((a + b + c) == 1000)
```

`numpy.testing`

```
np.testing.assert_equal
  (a[idx]**2 + b[idx]**2, c[idx]**2)
```

`triplets.py`

```
from __future__ import print_function
import numpy as np

#A Pythagorean triplet is a set of three natural numbers, a < b < c,
for which,
```

```
#a ** 2 + b ** 2 = c ** 2
#
#For example, 3 ** 2 + 4 ** 2 = 9 + 16 = 25 = 5 ** 2.
#
#There exists exactly one Pythagorean triplet for which a + b + c =
1000.
#Find the product abc.

#1. Create m and n arrays
m = np.arange(33)
n = np.arange(33)

#2. Calculate a, b and c
a = np.subtract.outer(m ** 2, n ** 2)
b = 2 * np.multiply.outer(m, n)
c = np.add.outer(m ** 2, n ** 2)

#3. Find the index
idx =  np.where((a + b + c) == 1000)

#4. Check solution
np.testing.assert_equal(a[idx]**2 + b[idx]**2, c[idx]**2)
print(a[idx], b[idx], c[idx])
      # [375] [200] [425]
```

How it works...

the `outer()`
are

documentation for the `outer()` universal function at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.ufunc.outer.html`

specialized `chararray` `ndarray`

`chararray`

How to do it...

```
carray = np.array(html).view(np.chararray)
```

Expand `expandtabs()`

```
8
carray = carray.expandtabs(1)
```

3. Split lines with the `splitlines()`

```
carray = carray.splitlines()
```

```
import urllib2
import numpy as np
import re

response = urllib2.urlopen('http://python.org/')
html = response.read()
html = re.sub(r'<.*?>', '', html)
carray = np.array(html).view(np.chararray)
carray = carray.expandtabs(1)
carray = carray.splitlines()
print(carray)
```

How it works...

We saw the specialized `chararray` class in action. It offers several vectorized string

for the specialized `chararray` class is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.chararray.html`

Masked arrays `MaskedArray` class
from the `numpy.ma` `ndarray`

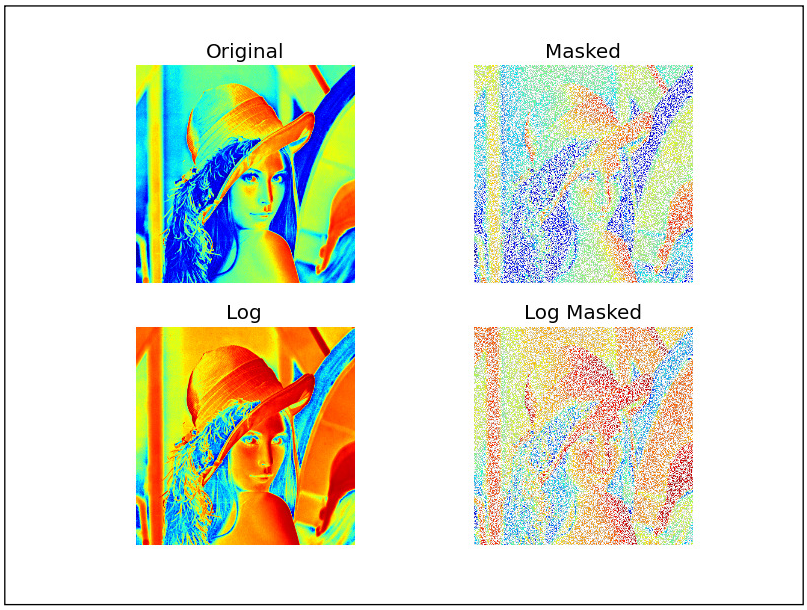
values thereof.

How to do it...

values that are either `0` or `1`

```
random_mask = np.random.randint
  (0, 2, size=lena.shape)


masked_array = np.ma.array
  (lena, mask=random_mask)


from __future__ import print_function
import numpy as np
from scipy.misc import lena
import matplotlib.pyplot as plt


lena = lena()
random_mask = np.random.randint(0, 2, size=lena.shape)

plt.subplot(221)
plt.title("Original")
plt.imshow(lena)
plt.axis('off')

masked_array = np.ma.array(lena, mask=random_mask)
print(masked_array)
plt.subplot(222)
plt.title("Masked")
plt.imshow(masked_array)
plt.axis('off')
```

```
plt.subplot(223)
plt.title("Log")
plt.imshow(np.log(lena))
plt.axis('off')

plt.subplot(224)
plt.title("Log Masked")
plt.imshow(np.log(masked_array))
plt.axis('off')

plt.show()
```



How it works...

We applied a random

in the `numpy.ma`

documentation for the `numpy.ma` module is at `http://docs.scipy.org/doc/numpy/reference/maskedarray.html`

How to do it...

```
triples = np.arange(0, len(close), 3)
print("Triples", triples[:10], "...")


signs = np.ones(len(close))
print("Signs", signs[:10], "...")
```

in *Chapter 2*, *Advanced Indexing and Array Concepts*.

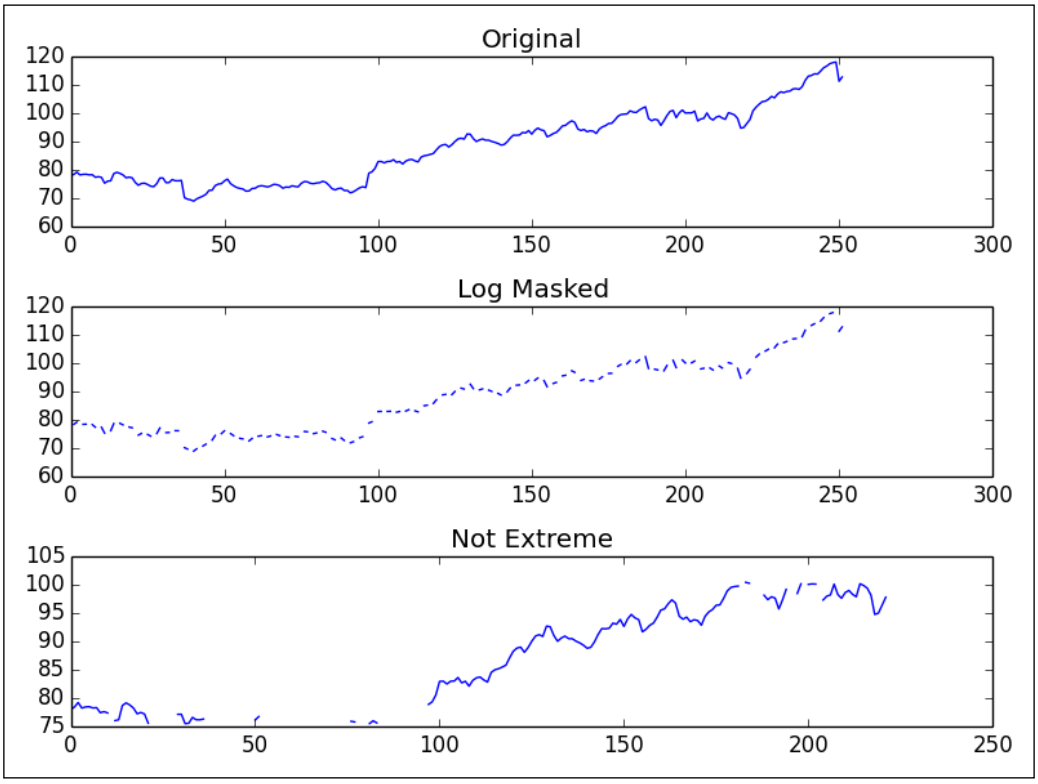
```
signs[triples] = -1
print("Signs", signs[:10], "...")


ma_log = np.ma.log(close * signs)
print("Masked logs", ma_log[:10], "...")
```

```
Triples [ 0  3  6  9 12 15 18 21 24 27] ...
Signs [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.] ...
Signs [-1.  1.  1. -1.  1.  1. -1.  1.  1. -1.] ...
Masked logs [-- 5.93655586575 5.95094223368 -- 5.97468290742
5.97510711452 --
 6.01674381162 5.97889061623 --] ...
```

```
dev = close.std()
avg = close.mean()
```

```
inside = numpy.ma.masked_outside
  (close, avg - dev, avg + dev)
print("Inside", inside[:10], "...")
```

```
Inside [-- -- -- -- -- -- 409.429675172
  410.240597855 -- --] ...
```



complete program for this tutorial

```
from __future__ import print_function
import numpy as np
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import matplotlib.pyplot as plt
```

```
def get_close(ticker):
   today = date.today()
   start = (today.year - 1, today.month, today.day)

   quotes = quotes_historical_yahoo(ticker, start, today)

   return np.array([q[4] for q in quotes])


close = get_close('AAPL')

triples = np.arange(0, len(close), 3)
print("Triples", triples[:10], "...")

signs = np.ones(len(close))
print("Signs", signs[:10], "...")

signs[triples] = -1
print("Signs", signs[:10], "...")

ma_log = np.ma.log(close * signs)
print("Masked logs", ma_log[:10], "...")

dev = close.std()
avg = close.mean()
inside = np.ma.masked_outside(close, avg - dev, avg + dev)
print("Inside", inside[:10], "...")

plt.subplot(311)
plt.title("Original")
plt.plot(close)

plt.subplot(312)
plt.title("Log Masked")
plt.plot(np.exp(ma_log))

plt.subplot(313)
plt.title("Not Extreme")
plt.plot(inside)

plt.tight_layout()
plt.show()
```

How it works...

Functions in the `numpy.ma` we regard as illegal. For instance, negative values are not allowed for the `log()` and `sqrt()` `NULL` or `None`

documentation for the `numpy.ma` module is at `http://docs.scipy.org/doc/numpy/reference/maskedarray.html`

`recarray` `ndarray`

is **standard deviation of log returns** (see `http://en.wikipedia.org/wiki/Rate_of_return#Arithmetic_and_logarithmic_return`). Reward, on the other

`recarray()` function.

How to do it...

```
weights = np.recarray((len(tickers),),
  dtype=[('symbol', np.str_, 16),
    ('stdscore', float), ('mean', float),
      ('score', float)])


for i, ticker in enumerate(tickers):
   close = get_close(ticker)
   logrets = np.diff(np.log(close))
   weights[i]['symbol'] = ticker
```

```
   weights[i]['mean'] = logrets.mean()
   weights[i]['stdscore'] = 1/logrets.std()
   weights[i]['score'] = 0
```

previous step.

3. We now

```
for key in ['mean', 'stdscore']:
    wsum = weights[key].sum()
    weights[key] = weights[key]/wsum



weights['score'] = (weights
  ['stdscore'] + weights['mean'])/2
weights['score'].sort()


from __future__ import print_function
import numpy as np
from matplotlib.finance import quotes_historical_yahoo
from datetime import date

tickers = ['MRK', 'T', 'VZ']

def get_close(ticker):
   today = date.today()
   start = (today.year - 1, today.month, today.day)

   quotes = quotes_historical_yahoo(ticker, start, today)

   return np.array([q[4] for q in quotes])


weights = np.recarray((len(tickers),), dtype=[('symbol', np.str_,
16),
   ('stdscore', float), ('mean', float), ('score', float)])

for i, ticker in enumerate(tickers):
   close = get_close(ticker)
   logrets = np.diff(np.log(close))
   weights[i]['symbol'] = ticker
   weights[i]['mean'] = logrets.mean()
   weights[i]['stdscore'] = 1/logrets.std()
```

```
        weights[i]['score'] = 0

    for key in ['mean', 'stdscore']:
        wsum = weights[key].sum()
        weights[key] = weights[key]/wsum

    weights['score'] = (weights['stdscore'] + weights['mean'])/2
    weights['score'].sort()

    for record in weights:
        print("%s,mean=%.4f,stdscore=%.4f,score=%.4f" %
    (record['symbol'], record['mean'], record['stdscore'],
    record['score']))
```

```
MRK,mean=0.8185,stdscore=0.2938,score=0.2177
T,mean=0.0927,stdscore=0.3427,score=0.2262
VZ,mean=0.0888,stdscore=0.3636,score=0.5561
```

`0` and `1`
from the start of the recipe. According to the output, `VZ`

How it works...

`recarray`

`arr.field`.

functions in the `numpy.recarray` module.

documentation for the `numpy.recarray` module is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.recarray.html`

7

`timeit`

Installing `line_profiler`

`line_profiler`

`cProfile` extension

`PuDB`

Introduction

`timeit` is a

`sort()`

quicksort and **merge sort** algorithms have an average running time of O(N log N), so

How to do it...

```
times = np.array([])

for size in sizes:
    integers = np.random.random_integers
      (1, 10 ** 6, size)



def measure():
    timer = timeit.Timer('dosort()',
      'from __main__ import dosort')
    return timer.timeit(10 ** 2)
```

3.

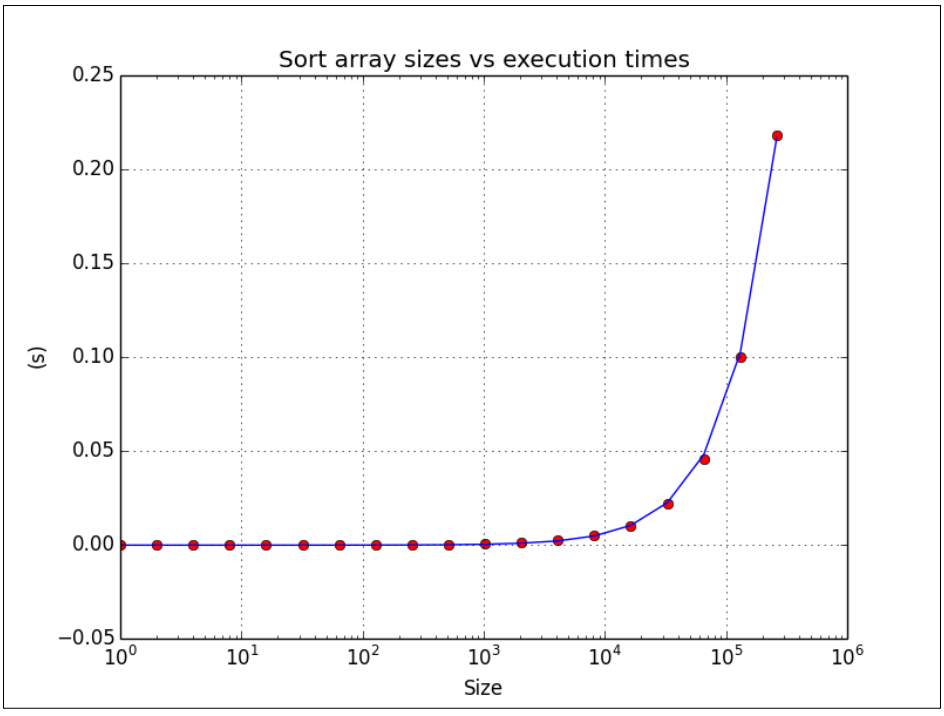
```
times = np.append(times, measure())
```

Fit the times into the theoretical model of `n log n`. Since we

```
fit = np.polyfit(sizes * powersOf2, times, 1)


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


# This program measures the performance of the NumPy sort function
# and plots time vs array size.
integers = []

def dosort():
   integers.sort()

def measure():
   timer = timeit.Timer('dosort()', 'from __main__ import dosort')

   return timer.timeit(10 ** 2)

powersOf2 = np.arange(0, 19)
```

```
sizes = 2 ** powersOf2

times = np.array([])

for size in sizes:
   integers = np.random.random_integers(1, 10 ** 6, size)
   times = np.append(times, measure())

fit = np.polyfit(sizes * powersOf2, times, 1)
print(fit)
plt.title("Sort array sizes vs execution times")
plt.xlabel("Size")
plt.ylabel("(s)")
plt.semilogx(sizes, times, 'ro')
plt.semilogx(sizes, np.polyval(fit, sizes * powersOf2))
plt.grid()
plt.show()
```

screenshot shows the resulting plot for the running time versus



How it works...

We measured the average running time of the `sort()` following

Function	Description
`random_integers()`	
`append()`	
`polyfit()`	
`polyval()`	corresponding value for a given value of `x`
`semilogx()`	plots data using a logarithmic scale on the *X* axis

documentation for `timeit` is at `http://docs.python.org/2/library/timeit.html`

`timeit`. We can

How to do it...

`pylab`

```
$ ipython --pylab
```

```
In [1]: a = arange(1000)
```

read `http://en.wikipedia.org/wiki/42_%28number%29`

```
In [2]: %timeit searchsorted(a, 42)
100000 loops, best of 3: 7.58 us per loop
```

```
                .I

import numpy as np

def invert(n):
  a = np.matrix(np.random.rand(n, n))

  return a.I

sizes = 2 ** np.arange(0, 12)

for n in sizes:
  invert(n)
```

```
In [1]: %run -t invert_matrix.py

IPython CPU timings (estimated):
  User   :       6.08 s.
  System :       0.52 s.
Wall time:      19.26 s.
```

the script with the `p`

```
In [2]: %run -p invert_matrix.py

852 function calls in 6.597 CPU seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
       12    3.228    0.269    3.228    0.269 {numpy.linalg.
lapack_lite.dgesv}
       24    2.967    0.124    2.967    0.124 {numpy.core.
multiarray._fastCopyAndTranspose}
       12    0.156    0.013    0.156    0.013 {method 'rand' of
'mtrand.RandomState' objects}
       12    0.087    0.007    0.087    0.007 {method 'copy' of
'numpy.ndarray' objects}
       12    0.069    0.006    0.069    0.006 {method 'astype' of
'numpy.ndarray' objects}
       12    0.025    0.002    6.304    0.525 linalg.py:404(inv)
```

```
       12    0.024    0.002    6.328    0.527 defmatrix.
py:808(getI)
        1    0.017    0.017    6.596    6.596 invert_matrix.
py:1(<module>)
       24    0.014    0.001    0.014    0.001 {numpy.core.
multiarray.zeros}
       12    0.009    0.001    6.580    0.548 invert_matrix.
py:3(invert)
       12    0.000    0.000    6.264    0.522 linalg.py:244(solve)
       12    0.000    0.000    0.014    0.001 numeric.
py:1875(identity)
        1    0.000    0.000    6.597    6.597 {execfile}
       36    0.000    0.000    0.000    0.000 defmatrix.py:279(__
array_finalize__)
       12    0.000    0.000    2.967    0.247 linalg.py:139(_
fastCopyAndTranspose)
       24    0.000    0.000    0.087    0.004 defmatrix.py:233(__
new__)
       12    0.000    0.000    0.000    0.000 linalg.py:99(_
commonType)
       24    0.000    0.000    0.000    0.000 {method '__array_
prepare__' of 'numpy.ndarray' objects}
       36    0.000    0.000    0.000    0.000 linalg.py:66(_
makearray)
       36    0.000    0.000    0.000    0.000 {numpy.core.
multiarray.array}
       12    0.000    0.000    0.000    0.000 {method 'view' of
'numpy.ndarray' objects}
       12    0.000    0.000    0.000    0.000 linalg.py:127(_to_
native_byte_order)
        1    0.000    0.000    6.597    6.597 interactiveshell.
py:2270(safe_execfile)
```

How it works...

We ran the aforementioned

Column	Description
ncalls	
tottime	total time spent in a function
percall	
cumtime	cumulative time spent in function and functions function, including recursive calls

http://ipython.org/ipython-doc/dev/interactive/magics.html

line_profiler

steps in this recipe.

setuptools

See also

How to do it...

Install `line_profiler` with `easy_install`

```
$ easy_install line_profiler
$ pip install line_profiler
```

Install the development version.

```
$ git clone https://github.com/rkern/line_profiler
```

```
$ python setup.py install
```

Installing IPython in *Chapter 1*, *Winding Along with IPython*

installed `line_profiler`, we can

How to do it...

`@profile`

```
import numpy as np
import time

@profile
def multiply(n):
  A = np.random.rand(n, n)
  time.sleep(np.random.randint(0, 2))
  return np.matrix(A) ** 2

for n in 2 ** np.arange(0, 10):
  multiply(n)
```

```
$ kernprof.py -l -v mat_mult.py
Wrote profile results to mat_mult.py.lprof
Timer unit: 1e-06 s

File: mat_mult.py
Function: multiply at line 4
Total time: 3.19654 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def multiply(n):
     6        10        13461   1346.1      0.4     A = numpy.
random.rand(n, n)
     7        10      3000689 300068.9     93.9     time.
sleep(numpy.random.randint(0, 2))
     8        10       182386  18238.6      5.7     return numpy.
matrix(A) ** 2
```

How it works...

`@profile` decorator tells `line_profiler`
explains the

Column	Description
`Line #`	
`Hits`	times the line was executed
`Time`	time spent executing the line
`Per Hit`	time spent executing the line
`% Time`	time spent executing the line relative to the time spent executing all the lines
`Line Contents`	content of the line

`line_profiler` `https://github.com/rkern/line_profiler`

`cProfile` is a `C`

`cProfile`, which transposes

How to do it...

Write the following `transpose()`

```
def transpose(n):
  random_values = np.random.random((n, n))
  return random_values.T
```

```
cProfile.run('transpose (1000)')
```

```
import numpy as np
import cProfile

def transpose(n):
   random_values = np.random.random((n, n))
   return random_values.T

cProfile.run('transpose (1000)')
```

```
4 function calls in 0.029 CPU seconds
  Ordered by: standard name
  ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
       1    0.001    0.001    0.029    0.029 <string>:1(<module>)
       1    0.000    0.000    0.028    0.028 cprofile_transpose.
py:5(transpose)
       1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
       1    0.028    0.028    0.028    0.028 {method 'random_
sample' of 'mtrand.RandomState' objects}
```

columns in the output

http://docs.python.org/2/library/profile.html

with pstats tutorial at http://pymotw.com/2/profile/#module-pstats

"If debugging is the process of removing software bugs, then programming must be the process of putting them in." Edsger Dijkstra, Dutch computer scientist, winner of the 1972 Turing Award

```
import numpy as np

a = np.arange(7)
print(a[8])
```

pdb

How to do it...

```
In [1]: %run buggy.py
---------------------------------------------------------------------
---------
IndexError                                Traceback (most recent
call last)
.../site-packages/IPython/utils/py3compat.pyc in execfile(fname,
*where)
    173             else:
    174                 filename = fname
--> 175             __builtin__.execfile(filename, *where)
```

```
.../buggy.py in <module>()
      2
      3 a = numpy.arange(7)
----> 4 print a[8]

IndexError: index out of bounds



In [2]: %debug
> .../buggy.py(4)<module>()
      2
      3 a = numpy.arange(7)
----> 4 print a[8]
```

3. `list` command, or use the shorthand `l`

```
ipdb> list
      1 import numpy as np
      2
      3 a = np.arange(7)
----> 4 print(a[8])


ipdb> len(a)
7


ipdb> print(a)
[0 1 2 3 4 5 6]
```

call stack

`bt`

```
ipdb> bt
 .../py3compat.py(175)execfile()
    171             if isinstance(fname, unicode):
    172                 filename = fname.encode(sys.
getfilesystemencoding())
    173             else:
```

```
    174                 filename = fname
--> 175             __builtin__.execfile(filename, *where)

> .../buggy.py(4)<module>()
      0 print a[8]


ipdb> u
> .../site-packages/IPython/utils/py3compat.py(175)execfile()
    173             else:
    174                 filename = fname
--> 175             __builtin__.execfile(filename, *where)


ipdb> d
> .../buggy.py(4)<module>()
      2
      3 a = np.arange(7)
----> 4 print(a[8])
```

How it works...

Command	Description
`list` or `l`	
`bt`	
`u`	
`d`	

http://docs.python.org/2/library/pdb.html

ipdb https://pypi.python.org/pypi/ipdb

PuDB

How to do it...

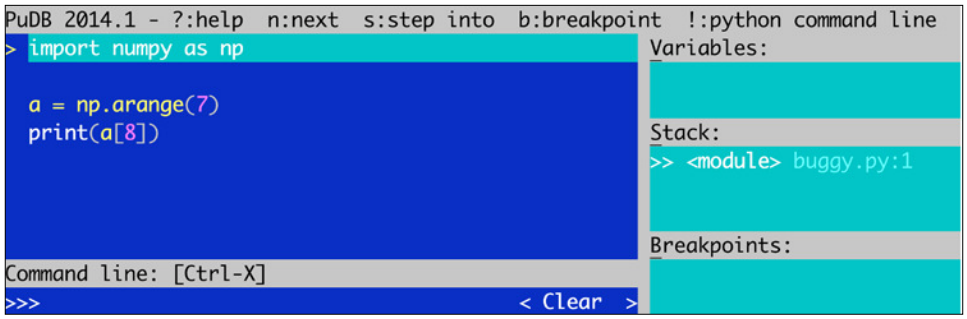
pudb

pudb

pip

```
$ sudo easy_install pudb
$ pip install pudb
$ pip freeze|grep pudb
pudb==2014.1
```

```
$ python -m pudb buggy.py
```



q exits

n

j and k b.

https://pypi.python.org/pypi/pudb

8

"If you lie to the computer, it will get you."

- Perry Farrar, Communications of the ACM, Volume 28

`docstrings`

Writing unit tests

Introduction

Behavior-Driven Development (**BDD**).

Install `pip` or `easy_install`

How to do it...

Choose one of the following options to install `pyflakes`

`pip`

```
$ sudo pip install pyflakes
```

`easy_install`

```
$ sudo easy_install pyflakes
```

`pyflakes` as well. For instance, on Red Hat do

```
$ sudo yum install pyflakes
```

```
$ sudo apt-get install pyflakes
```

`https://launchpad.net/pyflakes`

We will

pyflakes.

How to do it...

(see http://git-scm.com/book/en/v2/Getting-Started-Installing-Git

```
$ git clone git://github.com/numpy/numpy.git numpy
```

https://github.com/numpy/numpy.

numpy

```
$ pyflakes *.py
pavement.py:71: redefinition of unused 'md5' from line 69
pavement.py:88: redefinition of unused 'GIT_REVISION' from line 86
pavement.py:314: 'virtualenv' imported but unused
pavement.py:315: local variable 'e' is assigned to but never used
pavement.py:380: local variable 'sdir' is assigned to but never used
pavement.py:381: local variable 'bdir' is assigned to but never used
pavement.py:536: local variable 'st' is assigned to but never used
setup.py:21: 're' imported but unused
setup.py:27: redefinition of unused 'builtins' from line 25
setup.py:124: redefinition of unused 'GIT_REVISION' from line 118
setupegg.py:17: 'setup' imported but unused
setupscons.py:61: 'numpy' imported but unused
setupscons.py:64: 'numscons' imported but unused
setupsconsegg.py:6: 'setup' imported but unused
```

How it works...

reports is limited.

Pylint is

than `http://www.logilab.org/card/pylint_manual`.

`easy_install` or `pip`

```
$ easy_install pylint
$ sudo pip install pylint
```

How to do it...

```
$ pylint *.py
No config file found, using default configuration
************* Module pavement
C: 60: Line too long (81/80)
C:139: Line too long (81/80)
...
W: 50: TODO
W:168: XXX: find out which env variable is necessary to avoid the pb with
python
```

```
W: 71: Reimport 'md5' (imported line 143)
F: 73: Unable to import 'paver'
F: 74: Unable to import 'paver.easy'
C: 79: Invalid name "setup_py" (should match (([A-Z_][A-Z0-
9_]*)|(__.*__))$)
F: 86: Unable to import 'numpy.version'
E: 86: No name 'version' in module 'numpy'
C:149: Operator not followed by a space
if sys.platform =="darwin":
                ^^
C:202:prepare_nsis_script: Missing docstring
W:228:bdist_superpack: Redefining name 'options' from outer scope (line
74)
C:231:bdist_superpack.copy_bdist: Missing docstring
W:275:bdist_wininst_nosse: Redefining name 'options' from outer scope
(line 74)
```

How it works...

```
MESSAGE_TYPE: LINE_NUM:[OBJECT:] MESSAGE
```

[R]

[C]

[W]

[E]

[F]

Pychecker is

How to do it...

the `tar.gz` from `Sourceforge` (`http://pychecker.sourceforge.net/`

```
$ python setup.py install
```

`pip`

```
$ sudo pip install http://sourceforge.net/projects/pychecker/
files/pychecker/0.8.19/pychecker-0.8.19.tar.gz/download
```

```
$ pychecker *.py
...
Warnings...

...

setup.py:21: Imported module (re) not used
setup.py:27: Module (builtins) re-imported

...
```

Doctests are

use the `doctest` module to run these tests.

How to do it...

Write the `docstring`

```
docstring
"""
Test for the factorial of 3 that should pass.
>>> factorial(3)
6

Test for the factorial of 0 that should fail.
>>> factorial(0)
1
"""

return np.arange(1, n+1).cumprod()[-1]
```

3. `doctest`

```
doctest.testmod()
```

is the complete test example code from the `docstringtest.py`

```
import numpy as np
import doctest

def factorial(n):
   """
```

```
    Test for the factorial of 3 that should pass.
    >>> factorial(3)
    6

    Test for the factorial of 0 that should fail.
    >>> factorial(0)
    1
    """
    return np.arange(1, n+1).cumprod()[-1]

doctest.testmod()
```

```
                              -v
```

```
$ python docstringtest.py -v
Trying:
    factorial(3)
Expecting:
    6
ok
Trying:
    factorial(0)
Expecting:
    1
**********************************************************************
****
File "docstringtest.py", line 11, in __main__.factorial
Failed example:
    factorial(0)
Exception raised:
    Traceback (most recent call last):
      File ".../doctest.py", line 1253, in __run
        compileflags, 1) in test.globs
      File "<doctest __main__.factorial[1]>", line 1, in <module>
        factorial(0)
      File "docstringtest.py", line 14, in factorial
        return numpy.arange(1, n+1).cumprod()[-1]
    IndexError: index out of bounds
1 items had no tests:
```

```
    __main__
**********************************************************************
****
1 items had failures:
   1 of   2 in __main__.factorial
2 tests in 2 items.
1 passed and 1 failed.
***Test Failed*** 1 failures.
```

How it works...

`index out of bounds`
will do in the next tutorial.

`doctest` documentation at `http://docs.python.org/2/library/doctest.html`

Test-driven development (**TDD**
development almost
manic focus on unit testing.

, where we

convenience functions in the `numpy.testing`
suggests, is dedicated to testing.

How to do it...

factorial()

```
def factorial(n):
  if n == 0:
    return 1

  if n < 0:
    raise ValueError, "Don't be so negative"

  return np.arange(1, n+1).cumprod()
```

TestCase

class from the `unittest`
calling the `factorial()`

0

```
class FactorialTest(unittest.TestCase):
    def test_factorial(self):
      #Test for the factorial of 3 that should pass.
      self.assertEqual(6, factorial(3)[-1])
      np.testing.assert_equal(np.array([1, 2, 6]),
      factorial(3))

    def test_zero(self):
      #Test for the factorial of 0 that should pass.
      self.assertEqual(1, factorial(0))

    def test_negative(self):
      #Test for the factorial of negative numbers that
       should fail.
      # It should throw a ValueError, but we expect
       IndexError
      self.assertRaises(IndexError, factorial(-10))
```

```
            factorial()
import numpy as np
import unittest

def factorial(n):
   if n == 0:
      return 1

   if n < 0:
      raise ValueError, "Don't be so negative"

   return np.arange(1, n+1).cumprod()

class FactorialTest(unittest.TestCase):
   def test_factorial(self):
      #Test for the factorial of 3 that should pass.
      self.assertEqual(6, factorial(3)[-1])
      np.testing.assert_equal(np.array([1, 2, 6]), factorial(3))

   def test_zero(self):
      #Test for the factorial of 0 that should pass.
      self.assertEqual(1, factorial(0))

   def test_negative(self):
      #Test for the factorial of negative numbers that should
fail.
      # It should throw a ValueError, but we expect IndexError
      self.assertRaises(IndexError, factorial(-10))

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

```
.E.
======================================================================
ERROR: test_negative (__main__.FactorialTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unit_test.py", line 26, in test_negative
    self.assertRaises(IndexError, factorial(-10))
```

```
  File "unit_test.py", line 9, in factorial
    raise ValueError, "Don't be so negative"
ValueError: Don't be so negative

----------------------------------------------------------------------
----
Ran 3 tests in 0.001s

FAILED (errors=1)
```

How it works...

We saw how to implement simple unit tests using the standard `unittest`
We wrote a test class that extends the `TestCase` class from the `unittest`

Function	Description
`numpy.testing.assert_equal()`	
`unittest.assertEqual()`	
`unittest.assertRaises()`	whether an exception is thrown

`testing`

Function	Description
`assert_almost_equal()`	raises an exception if two precision
`assert_approx_equal()`	raises an exception if two significance
`assert_array_almost_equal()`	precision
`assert_array_equal()`	
`assert_array_less()`	do not have the same shape, and the elements

Function	Description
`assert_raises()`	fails if a specified exception is defined arguments
`assert_warns()`	fails if a specified warning is not thrown
`assert_string_equal()`	

Mocks are

using the `mock`

How to do it...

First, we will install the `mock`

`mock`

```
$ sudo easy_install mock
```

`do_work()` method, which calls a dangerous `factorial()`

```
reactor.factorial = MagicMock(return_value=6)
```

6.

3. We can

`factorial()` method was called with

```
reactor.factorial.assert_called_with(3, "mocked")
```

```
from __future__ import print_function
from mock import MagicMock
```

```
import numpy as np
import unittest

class NuclearReactor():
   def __init__(self, n):
      self.n = n

   def do_work(self, msg):
      print("Working")

      return self.factorial(self.n, msg)

   def factorial(self, n, msg):
      print(msg)

      if n == 0:
         return 1

      if n < 0:
         raise ValueError, "Core meltdown"

      return np.arange(1, n+1).cumprod()

class NuclearReactorTest(unittest.TestCase):
   def test_called(self):
      reactor = NuclearReactor(3)
      reactor.factorial = MagicMock(return_value=6)
      result = reactor.do_work("mocked")
      self.assertEqual(6, result)
      reactor.factorial.assert_called_with(3, "mocked")

   def test_unmocked(self):
      reactor = NuclearReactor(3)
      reactor.factorial(3, "unmocked")
      np.testing.assert_raises(ValueError)

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

We pass a string to the `factorial`

demonstrate what happens if we exercise the real code without

```
Working
.unmocked
.
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

How it works...

is replacing. We need to set them up to respond in an appropriate manner. For instance, the

```
6
```

`http://pypi.python.org/pypi/mock`

BDD (**Behavior-driven Development**

test according to certain conventions and rules. In this recipe, we will see an example of those conventions.

sentence consisting of several steps. Each step is more or less a unit test that we can write,

Lettuce to test the factorial function.

How to do it...

```
$ pip install lettuce
$ sudo easy_install lettuce
```

tests
features containing the factorial.feature
along with the functional descriptions and test code in the steps.py

```
./tests:
features

./tests/features:
factorial.feature   steps.py
```

3.

output. We
have different scenarios with the Given, When, and Then sections, which correspond

```
Feature: Compute factorial

    Scenario: Factorial of 0
      Given I have the number 0
      When I compute its factorial
      Then I see the number 1

    Scenario: Factorial of 1
      Given I have the number 1
      When I compute its factorial
      Then I see the number 1

    Scenario: Factorial of 3
      Given I have the number 3
      When I compute its factorial
      Then I see the number 1, 2, 6
```

attention to the text used to annotate the methods. It matches the text in the

```
   fromstring()
```

```
from lettuce import *
import numpy as np

@step('I have the number (\d+)')
def have_the_number(step, number):
    world.number = int(number)

@step('I compute its factorial')
def compute_its_factorial(step):
    world.number = factorial(world.number)

@step('I see the number (.*)')
def check_number(step, expected):
    expected = np.fromstring(expected, dtype=int, sep=',')
    np.testing.assert_equal(world.number, expected, \
        "Got %s" % world.number)

def factorial(n):
   if n == 0:
      return 1

   if n < 0:
      raise ValueError, "Core meltdown"

   return np.arange(1, n+1).cumprod()
```

to the `tests`

```
$ lettuce

Feature: Compute factorial        # features/factorial.feature:1

  Scenario: Factorial of 0        # features/factorial.feature:3
    Given I have the number 0     # features/steps.py:5
```

```
    When I compute its factorial  # features/steps.py:9
    Then I see the number 1       # features/steps.py:13

  Scenario: Factorial of 1        # features/factorial.feature:8
    Given I have the number 1     # features/steps.py:5
    When I compute its factorial  # features/steps.py:9
    Then I see the number 1       # features/steps.py:13

  Scenario: Factorial of 3        # features/factorial.feature:13
    Given I have the number 3     # features/steps.py:5
    When I compute its factorial  # features/steps.py:9
    Then I see the number 1, 2, 6 # features/steps.py:13

1 feature (1 passed)
3 scenarios (3 passed)
9 steps (9 passed)
```

How it works...

testing functions to test the different steps and the `fromstring()` function to create a

documentation at `http://lettuce.it/`

9

with Cython

Building a Hello World program

Calling C functions

Introduction

Cython

In order to use Sage
more information, see `https://www.enthought.com/products/canopy/`, `https://store.continuum.io/cshop/anaconda/`, and `http://sagemath.org/`. We will

How to do it...

We can install

`http://cython.org/#download`.

`cd` command.

```
$ python setup.py install
```

```
$ easy_install cython
$ sudo pip install cython
```

Install
`http://www.lfd.uci.edu/~gohlke/pythonlibs/#cython`.

`http://docs.cython.org/src/quickstart/install.html`

As is the tradition with programming languages, we will start with a Hello World example.
`.pyx`
`.c`

How to do it...

Hello World

pyx

```
def say_hello():
  print "Hello World!"
```

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("hello", ["hello.pyx"])]

setup(
        name = 'Hello world app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
     )
```

a name.

3. Build using

```
$ python setup.py build_ext --inplace
```

```
running build_ext
cythoning hello.pyx to hello.c
building 'hello' extension
creating build
```

```
from hello import say_hello
```

How it works...

we need to compile our code. We wrote a `.pyx` `Hello World` code and a `setup.py`

http://docs.cython.org/src/quickstart/build.html

We can

proportion http://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval

$$\sqrt{\frac{p(1-p)}{n}}$$

In the formula, **p** **n**

How to do it...

Write a .pyx

```
import numpy as np

def pos_confidence(numbers):
   diffs = np.diff(numbers)
   n = float(len(diffs))
   p = len(diffs[diffs > 0])/n
   confidence = np.sqrt(p * (1 - p)/ n)

   return (p, confidence)
```

setup.py
things, such as the name of the .pyx

```
from distutils.core import setup
from distutils.extension import Extension
```

```
from Cython.Distutils import build_ext

ext_modules = [Extension("binomial_proportion", ["binomial_
proportion.pyx"])]

setup(
        name = 'Binomial proportion app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
    )
```

3. write a matplotlib confidence()

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
from binomial_proportion import pos_confidence

#1. Get close prices.
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  numpy.array([q[4] for q in quotes])
print pos_confidence(close)
```

```
(0.56746031746031744, 0.031209043355655924)
```

How it works...

.pyx

See also

http://docs.cython.org/src/tutorial/numpy.html

We can call C `log()` function.
`log()` function

How to do it...

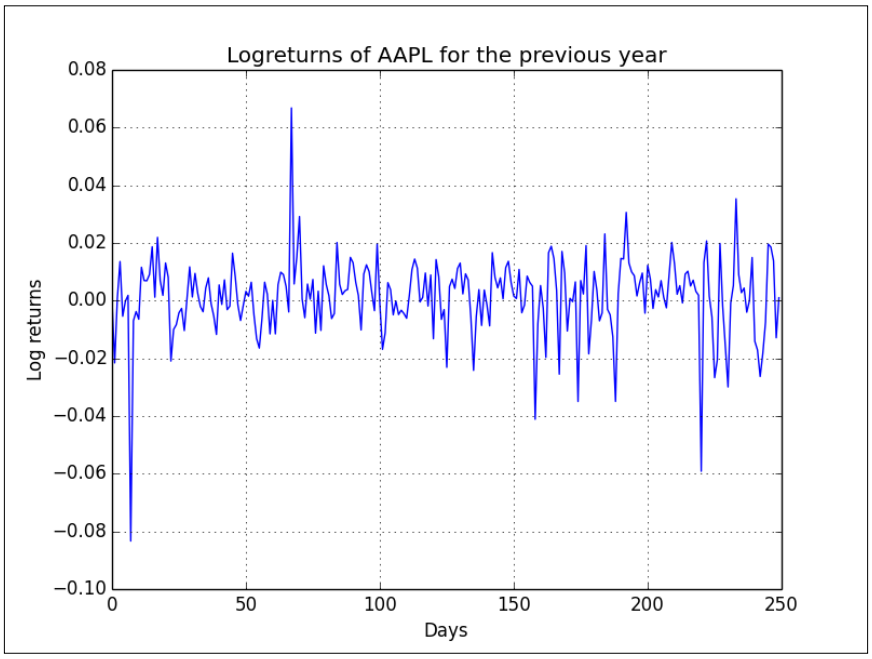
First, import the C `log()` function from the `libc`
`for` `diff()` function to

```
from libc.math cimport log
import numpy as np

def logrets(numbers):
   logs = [log(x) for x in numbers]
   return np.diff(logs)
```

values in the `setup.py`

`matplotlib`
`logrets()`

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy as np
from log_returns import logrets
import matplotlib.pyplot as plt

today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('AAPL', start, today)
close =  np.array([q[4] for q in quotes])
plt.plot(logrets(close))
plt.title('Logreturns of AAPL for the previous year')
plt.xlabel('Days')
plt.ylabel('Log returns')
plt.grid()
plt.show()
```



How it works...

We called the C `log()` functions, was used to

```
http://docs.cython.org/src/tutorial/external.html
```

$$\sum_{n=0}^{\infty} \frac{1}{n!}$$

See `http://en.wikipedia.org/wiki/E_%28mathematical_constant%29` for more information.

How to do it...

`1` to `n` (`n` is `40` in our example).

```
from __future__ import print_function
import numpy as np
import cProfile
import pstats

def approx_e(n=40, display=False):
   # array of [1, 2, ... n-1]
   arr = np.arange(1, n)

   # calculate the factorials and convert to floats
   arr = arr.cumprod().astype(float)

   # reciprocal 1/n
   arr = np.reciprocal(arr)

   if display:
    print(1 + arr.sum())

# Repeat multiple times because NumPy is so fast
def run(repeat=2000):
```

```
        for i in range(repeat):
            approx_e()


    cProfile.runctx("run()", globals(), locals(),
    "Profile.prof")

    s = pstats.Stats("Profile.prof")
    s.strip_dirs().sort_stats("time").print_stats()

    approx_e(display=True)
```

e approximation are shown in the following snippet. Refer to *Chapter 7*,

```
         8004 function calls in 0.016 seconds


   Ordered by: internal time


   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
     2000    0.007    0.000    0.015    0.000 numpy_approxe.
py:6(approx_e)
     2000    0.004    0.000    0.004    0.000 {method 'cumprod' of
'numpy.ndarray' objects}
     2000    0.002    0.000    0.002    0.000 {numpy.core.
multiarray.arange}
     2000    0.002    0.000    0.002    0.000 {method 'astype' of
'numpy.ndarray' objects}
        1    0.001    0.001    0.016    0.016 numpy_approxe.
py:20(run)
        1    0.000    0.000    0.000    0.000 {range}
        1    0.000    0.000    0.016    0.016 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}



2.71828182846
```

implementation
need a `for`
code for the `.pyx`

```
def approx_e(int n=40, display=False):
    cdef double sum = 0.
    cdef double factorial = 1.
    cdef int k

    for k in xrange(1,n+1):
        factorial *= k
        sum += 1/factorial

    if display:
        print(1 + sum)
```

```
import pstats
import cProfile
import pyximport
pyximport.install()

import approxe

# Repeat multiple times because Cython is so fast
def run(repeat=2000):
    for i in range(repeat):
        approxe.approx_e()

cProfile.runctx("run()", globals(), locals(), "Profile.prof")

s = pstats.Stats("Profile.prof")
s.strip_dirs().sort_stats("time").print_stats()

approxe.approx_e(display=True)
```

```
         2004 function calls in 0.001 seconds

   Ordered by: internal time
```

```
   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
     2000    0.001    0.000    0.001    0.000 {approxe.approx_e}
        1    0.000    0.000    0.001    0.001 cython_profile.
py:9(run)
        1    0.000    0.000    0.000    0.000 {range}
        1    0.000    0.000    0.001    0.001 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}



2.71828182846
```

How it works...

http://docs.cython.org/src/tutorial/profiling_tutorial.html

approximation
(see http://en.wikipedia.org/wiki/Stirling%27s_approximation for

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\sqrt{\pi}(\frac{n}{e})^n \sqrt[6]{8n^3 + 4n^2 + n + \frac{1}{30}}$$

How to do it...

ndarray

cimport

cdef

```
import numpy
cimport numpy

def ramanujan_factorial(numpy.ndarray n):
   sqrt_pi = numpy.sqrt(numpy.pi, dtype=numpy.float64)
   cdef numpy.ndarray root = (8 * n + 4) * n + 1
   root = root * n + 1/30.
   root = root ** (1/6.)
   return sqrt_pi * calc_eton(n) * root

def stirling_factorial(numpy.ndarray n):
    return numpy.sqrt(2 * numpy.pi * n) * calc_eton(n)

def calc_eton(numpy.ndarray n):
    return (n/numpy.e) ** n
```

setup.py

get_include() function. With this amendment, the setup.py

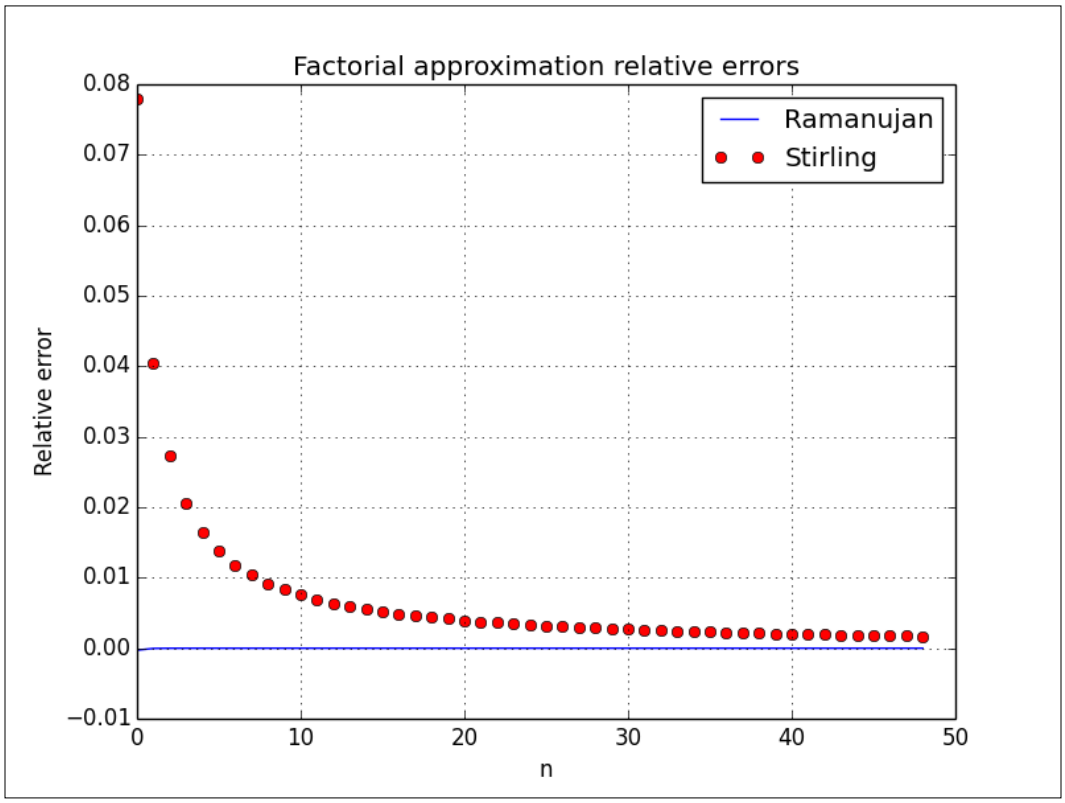
```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
```

```
import numpy

ext_modules = [Extension("factorial", ["factorial.pyx"], include_
dirs = [numpy.get_include()])]

setup(
        name = 'Factorial app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
     )
```

3.

to the factorial values that we will `cumprod()` function,

```
from factorial import ramanujan_factorial
from factorial import stirling_factorial
import numpy as np
import matplotlib.pyplot as plt

N = 50
numbers = np.arange(1, N)
factorials = np.cumprod(numbers, dtype=float)

def error(approximations):
   return (factorials - approximations)/factorials

plt.plot(error(ramanujan_factorial(numbers)), 'b-',
label='Ramanujan')
plt.plot(error(stirling_factorial(numbers)), 'ro',
label='Stirling')
plt.title('Factorial approximation relative errors')
plt.xlabel('n')
plt.ylabel('Relative error')
plt.grid()
plt.legend(loc='best')
plt.show()
```



How it works...

In this example, we saw a

`cimport`, which imports C declarations

Including directories with the `get_include()` function

`cdef`

`http://docs.cython.org/src/quickstart/cythonize.html`

10
Fun with Scikits

Installing statsmodels

Installing pandas

Resampling time series data

Introduction

Scikits

`statsmodels`

[redacted]

easy_install

[redacted]

Chapter 1, *Winding Along with IPython*

How to do it...

Installing with `easy_install`

```
$ pip install -U scikit-learn
$ easy_install -U scikit-learn
```

sudo

in front of the commands, or log in as admin.

Installing from source the source from `http://pypi.python.org/pypi/scikit-learn/`

```
cd
```

following

```
$ python setup.py install
```

[redacted]

to the data.

How to do it...

http://
scikit-learn.org/dev/modules/classes.html#module-sklearn.datasets.

following code from `sample_data.py`

```
from __future__ import print_function
from sklearn import datasets

boston_prices = datasets.load_boston()
print("Data shape", boston_prices.data.shape)
print("Data max=%s min=%s" % (boston_prices.data.max(), boston_prices.
data.min()))
print("Target shape", boston_prices.target.shape)
print("Target max=%s min=%s" % (boston_prices.target.max(), boston_
prices.target.min()))
```

```
Data shape (506, 13)
Data max=711.0 min=0.0
Target shape (506,)
Target max=50.0 min=5.0
```

Clustering

Dow Jones Industrial

Average (**DJI** or **DJIA**)
review in previous chapters.

How to do it...

`AffinityPropagation`

```
# 2011 to 2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)

#Dow Jones symbols
symbols = ["AA", "AXP", "BA", "BAC", "CAT",
   "CSCO", "CVX", "DD", "DIS", "GE", "HD",
   "HPQ", "IBM", "INTC", "JNJ", "JPM",
   "KO", "MCD", "MMM", "MRK", "MSFT", "PFE",
   "PG", "T", "TRV", "UTX", "VZ", "WMT", "XOM"]

quotes = []

for symbol in symbols:
    try:
        quotes.append(finance.quotes_historical_yahoo(symbol,
start, end, asobject=True))
    except urllib2.HTTPError as e:
        print(symbol, e)

close = np.array([q.close for q in quotes]).astype(np.float)
print(close.shape)
```

What we are

```
logreturns = np.diff(np.log(close))
print(logreturns.shape)

logreturns_norms = np.sum(logreturns ** 2, axis=1)
S = - logreturns_norms[:, np.newaxis] - logreturns_norms[np.
newaxis, :] + 2 * np.dot(logreturns, logreturns.T)
```

3. AffinityPropagation

```
aff_pro = sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

for symbol, label in zip(symbols, labels):
    print('%s in Cluster %d' % (symbol, label))


from __future__ import print_function
import datetime
import numpy as np
import sklearn.cluster
from matplotlib import finance
import urllib2

#1. Download price data

# 2011 to 2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)

#Dow Jones symbols
symbols = ["AA", "AXP", "BA", "BAC", "CAT",
   "CSCO", "CVX", "DD", "DIS", "GE", "HD",
   "HPQ", "IBM", "INTC", "JNJ", "JPM",
   "KO", "MCD", "MMM", "MRK", "MSFT", "PFE",
   "PG", "T", "TRV", "UTX", "VZ", "WMT", "XOM"]

quotes = []

for symbol in symbols:
    try:
        quotes.append(finance.quotes_historical_yahoo(symbol,
start, end, asobject=True))
    except urllib2.HTTPError as e:
        print(symbol, e)

close = np.array([q.close for q in quotes]).astype(np.float)
print(close.shape)
```

```
#2. Calculate affinity matrix
logreturns = np.diff(np.log(close))
print(logreturns.shape)

logreturns_norms = np.sum(logreturns ** 2, axis=1)
S = - logreturns_norms[:, np.newaxis] - logreturns_norms[np.
newaxis, :] + 2 * np.dot(logreturns, logreturns.T)

#3. Cluster using affinity propagation
aff_pro = sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

for symbol, label in zip(symbols, labels):
    print('%s in Cluster %d' % (symbol, label))
```

```
(29, 252)
(29, 251)
AA in Cluster 0
AXP in Cluster 6
BA in Cluster 6
BAC in Cluster 1
CAT in Cluster 6
CSCO in Cluster 2
CVX in Cluster 7
DD in Cluster 6
DIS in Cluster 6
GE in Cluster 6
HD in Cluster 5
HPQ in Cluster 3
IBM in Cluster 5
INTC in Cluster 6
JNJ in Cluster 5
JPM in Cluster 4
KO in Cluster 5
MCD in Cluster 5
MMM in Cluster 6
```

```
MRK in Cluster 5
MSFT in Cluster 5
PFE in Cluster 7
PG in Cluster 5
T in Cluster 5
TRV in Cluster 5
UTX in Cluster 6
VZ in Cluster 5
WMT in Cluster 5
XOM in Cluster 7
```

How it works...

overview of the functions we used in this

Function	Description
`sklearn.cluster.AffinityPropagation()`	Creates an `AffinityPropagation`
`sklearn.cluster.AffinityPropagation.fit()`	
`diff()`	are computed.
`log()`	
`sum()`	
`dot()`	

relevant documentation is at `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html`

[redacted]

statsmodels

How to do it...

Source and http://statsmodels.sourceforge.net/install.html

```
$ python setup.py install
```

setuptools

```
$ easy_install statsmodels
```

[redacted]

statsmodels

Anderson-Darling test for http://en.wikipedia.org/wiki/Anderson%E2%80%93Darling_test).

How to do it...

We will

p-value

```
from __future__ import print_function
import datetime
import numpy as np
from matplotlib import finance
from statsmodels.stats.adnorm import normal_ad

#1. Download price data

# 2011 to 2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)
```

```
quotes = finance.quotes_historical_yahoo('AAPL', start, end,
asobject=True)

close = np.array(quotes.close).astype(np.float)
print(close.shape)

print(normal_ad(np.diff(np.log(close))))

#Retrieving data for AAPL
#(252,)
#(0.57103805516803163, 0.13725944999430437)
```

p-value of 0.13

```
Retrieving data for AAPL
(252,)
(0.57103805516803163, 0.13725944999430437)
```

How it works...

statsmodels
input. For the 0.13. Since

How to do it...

scikit-image is a

x, *y*

How to do it...

```
$ pip install -U scikit-image
$ easy_install -U scikit-image
```

```
$ python setup.py install
```

Corner detection (`http://en.wikipedia.org/wiki/Corner_detection`) is a standard
Harris corner detector, which is great,

`jpeglib`

```
$ ./configure
$  make
$  sudo make install
```

How to do it...

```
dataset = load_sample_images()
img = dataset.images[0]
```

```
gray_img = rgb2gray(img)
```

3. Call the `corner_harris()`

```
harris_coords = corner_peaks(corner_harris(gray_img))
y, x = np.transpose(harris_coords)
```

```
from sklearn.datasets import load_sample_images
import matplotlib.pyplot as plt
import numpy as np
from skimage.feature import corner_harris
```

```
from skimage.feature import corner_peaks
from skimage.color import rgb2gray

dataset = load_sample_images()
img = dataset.images[0]
gray_img = rgb2gray(img)
harris_coords = corner_peaks(corner_harris(gray_img))
y, x = np.transpose(harris_coords)
plt.axis('off')
plt.imshow(img)
plt.plot(x, y, 'ro')
plt.show()
```

We get an image with dots, where the script detects corners, as shown in the

How it works...

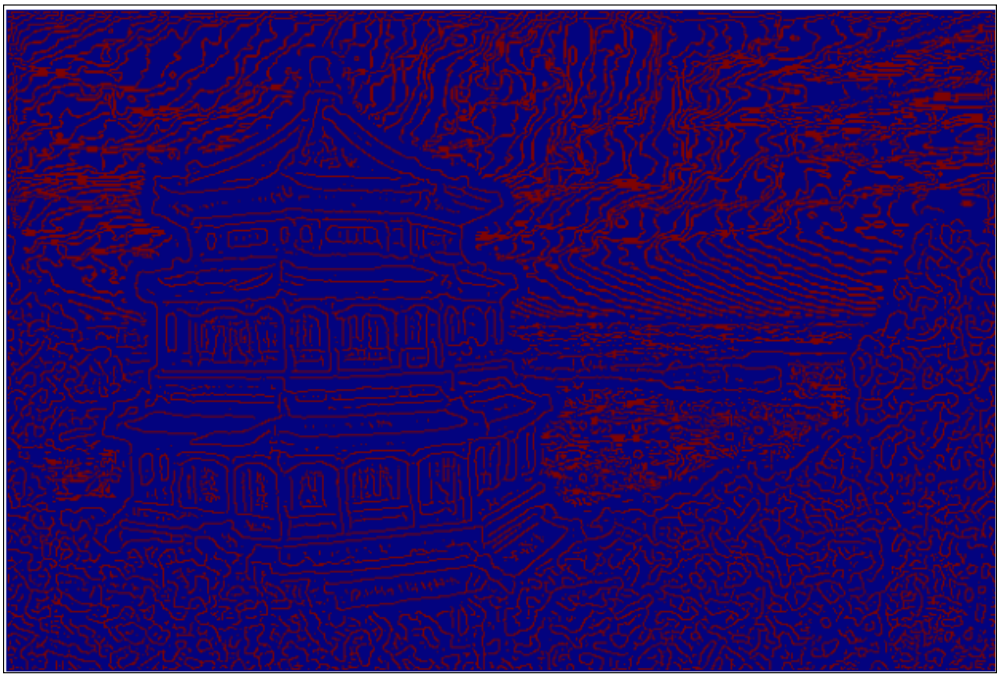
http://scikit-image.org/docs/dev/auto_examples/plot_corner.html

Edge detection is http://en.wikipedia.org/wiki/Edge_detection implementation

How to do it...

We will use the same edge_detection.py

```
from sklearn.datasets import load_sample_images
import matplotlib.pyplot as plt
import skimage.feature

dataset = load_sample_images()
img = dataset.images[0]
edges = skimage.feature.canny(img[..., 0])
plt.axis('off')
plt.imshow(edges)
plt.show()
```



documentation is at `http://scikit-image.org/docs/dev/auto_examples/plot_canny.html`

pandas is a
programming language, which are not coincidental. R is a specialized programming language popular with data scientists. For instance, R inspired the core `DataFrame`

How to do it...

pandas

```
$ sudo easy_install -U pandas
$ pip install pandas
```

python-pandas

```
$ sudo apt-get install python-pandas
```

```
$ git clone git://github.com/pydata/pandas.git
$ cd pandas
$ python setup.py install
```

documentation is at `http://pandas.pydata.org/pandas-docs/stable/install.html`

pandas

A pandas `DataFrame` is a matrix

How to do it...

First, we will create the `DataFrame`

```
data = {}

for i, symbol in enumerate(symbols):
```

```
   data[symbol] = np.diff(np.log(close[i]))

# Convention: import pandas as pd
df = pd.DataFrame(data,
  index=dates[0][:-1], columns=symbols)
```

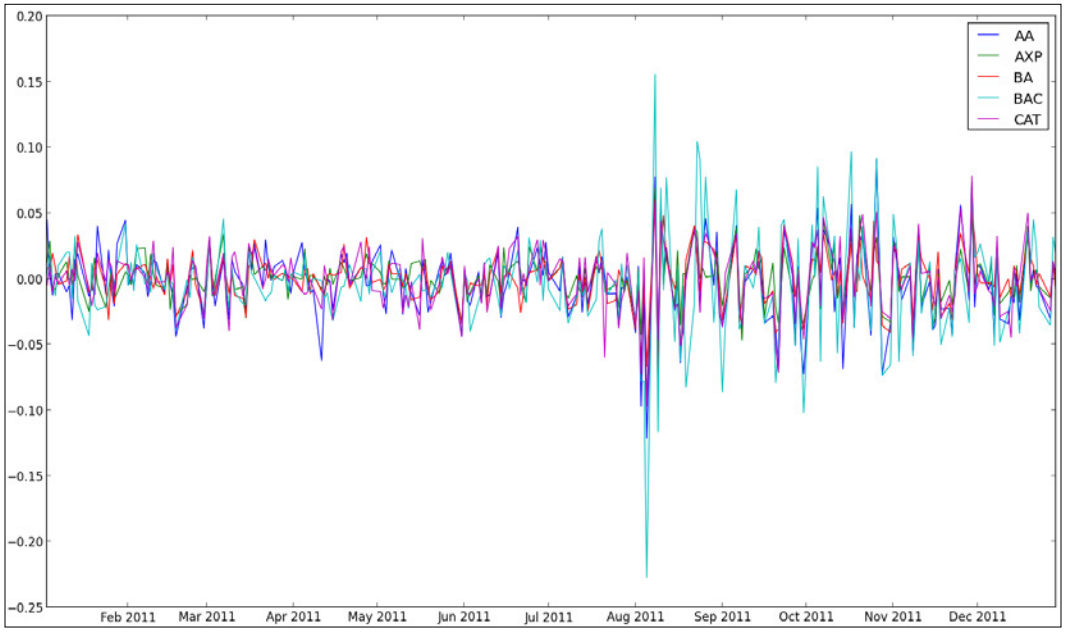
We can now perform operations such as calculating a correlation matrix or plotting on

```
print(df.corr())
df.plot()


from __future__ import print_function
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib import finance
import numpy as np

# 2011 to 2012
start = datetime(2011, 01, 01)
end = datetime(2012, 01, 01)

symbols = ["AA", "AXP", "BA", "BAC", "CAT"]

quotes = [finance.quotes_historical_yahoo(symbol, start, end,
asobject=True)
          for symbol in symbols]

close = np.array([q.close for q in quotes]).astype(np.float)
dates = np.array([q.date for q in quotes])

data = {}

for i, symbol in enumerate(symbols):
   data[symbol] = np.diff(np.log(close[i]))

df = pd.DataFrame(data, index=dates[0][:-1], columns=symbols)


print(df.corr())
df.plot()
plt.legend(symbols)
plt.show()
```

```
#            AA        AXP        BA       BAC       CAT
#AA    1.000000   0.768484   0.758264   0.737625   0.837643
#AXP   0.768484   1.000000   0.746898   0.760043   0.736337
#BA    0.758264   0.746898   1.000000   0.657075   0.770696
#BAC   0.737625   0.760043   0.657075   1.000000   0.657113
#CAT   0.837643   0.736337   0.770696   0.657113   1.000000
```

	AA	AXP	BA	BAC	CAT
AA	**1.000000**	**0.768484**	**0.758264**	**0.737625**	**0.837643**
AXP	**0.768484**	**1.000000**	**0.746898**	**0.760043**	**0.736337**
BA	**0.758264**	**0.746898**	**1.000000**	**0.657075**	**0.770696**
BAC	**0.737625**	**0.760043**	**0.657075**	**1.000000**	**0.657113**
CAT	**0.837643**	**0.736337**	**0.770696**	**0.657113**	**1.000000**

following



How it works...

We used the following `DataFrame`

Method	Description
`pandas.DataFrame()`	`DataFrame` with specified data,
`pandas.DataFrame.corr()`	correlation is used.
`pandas.DataFrame.plot()`	`matplotlib`.

related documentation is at `http://pandas.pydata.org/pandas-docs/dev/generated/pandas.DataFrame.html`

Chapter 4, *pandas Primer* *Python Data Analysis*, *Packt Publishing*

statsmodels `https://github.com/statsmodels/statsmodels/tree/master/statsmodels/datasets`.

copper prices, world consumption, and other parameters.

Before we start, we might need to install **patsy**

```
$ sudo easy_install patsy
$ pip install --upgrade patsy
```

How to do it...

In this section, we will load a dataset from statsmodels as a pandas `DataFrame` or `Series`

`load_pandas()`

```
data = statsmodels.api.datasets.copper.load_pandas()
```

`DataSet`

`Dataset` `exog`, which when loaded as a `pandas` `DataFrame` `endog`

`OLS`

its `fit()`

```
x, y = data.exog, data.endog

fit = statsmodels.api.OLS(y, x).fit()
print("Fit params", fit.params)
```

```
Fit params COPPERPRICE         14.222028
INCOMEINDEX       1693.166242
ALUMPRICE          -60.638117
INVENTORYINDEX    2515.374903
TIME               183.193035
```

3. `summary()`

```
print(fit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       WORLDCONSUMPTION   R-squared:                       0.978
Model:                           OLS   Adj. R-squared:                  0.974
Method:                Least Squares   F-statistic:                     224.9
Date:               Fri, 28 Sep 2012   Prob (F-statistic):           2.55e-16
Time:                       20:25:26   Log-Likelihood:                -172.62
No. Observations:                  25   AIC:                             355.2
Df Residuals:                      20   BIC:                             361.3
Df Model:                           4
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
COPPERPRICE       14.2220     12.090      1.176      0.253       -10.998    39.442
INCOMEINDEX     1693.1662   1970.555      0.859      0.400     -2417.339  5803.671
ALUMPRICE        -60.6381     32.023     -1.894      0.073      -127.437     6.161
INVENTORYINDEX  2515.3749   1670.948      1.505      0.148      -970.162  6000.912
TIME             183.1930     36.879      4.967      0.000       106.264   260.122
==============================================================================
Omnibus:                        8.007   Durbin-Watson:                   1.316
Prob(Omnibus):                  0.018   Jarque-Bera (JB):                6.014
Skew:                          -0.936   Prob(JB):                       0.0494
Kurtosis:                       4.506   Cond. No.                     2.20e+03
==============================================================================

The condition number is large, 2.2e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
from __future__ import print_function
import statsmodels.api

# See https://github.com/statsmodels/statsmodels/tree/master/
statsmodels/datasets
data = statsmodels.api.datasets.copper.load_pandas()

x, y = data.exog, data.endog

fit = statsmodels.api.OLS(y, x).fit()
print("Fit params", fit.params)
print()
print("Summary")
print()
print(fit.summary())
```

How it works...

`Dataset` class of statsmodels follows a special format. Among others, this class has the `endog` and `exog` `load()` function, which loads `load_pandas()` method, which loads data as `pandas` and consumption.

documentation is at `http://statsmodels.sourceforge.net/stable/datasets/index.html`

In this

How to do it...

`DataFrame` and calling its `resample()`

Before we can create a pandas `DataFrame`, we need to create a `DatetimeIndex` `DataFrame` constructor. Create the index from the downloaded

```
dt_idx = pandas.DatetimeIndex(quotes.date)
```

```
df = pandas.DataFrame (quotes.close, index=dt_idx,
columns=[symbol])
```

3.

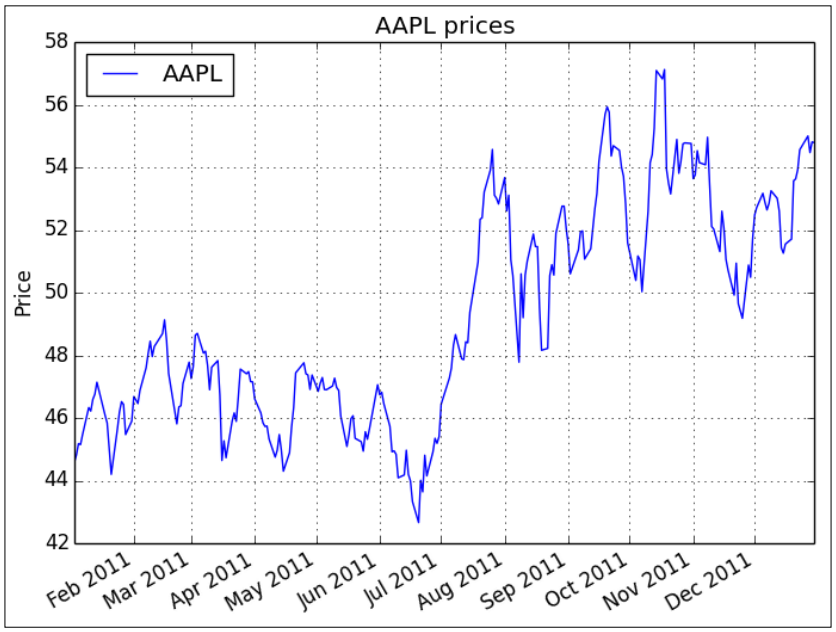
```
resampled = df.resample('M', how=numpy.mean)
print(resampled)
```

```
                AAPL
2011-01-31  336.932500
2011-02-28  349.680526
2011-03-31  346.005652
```

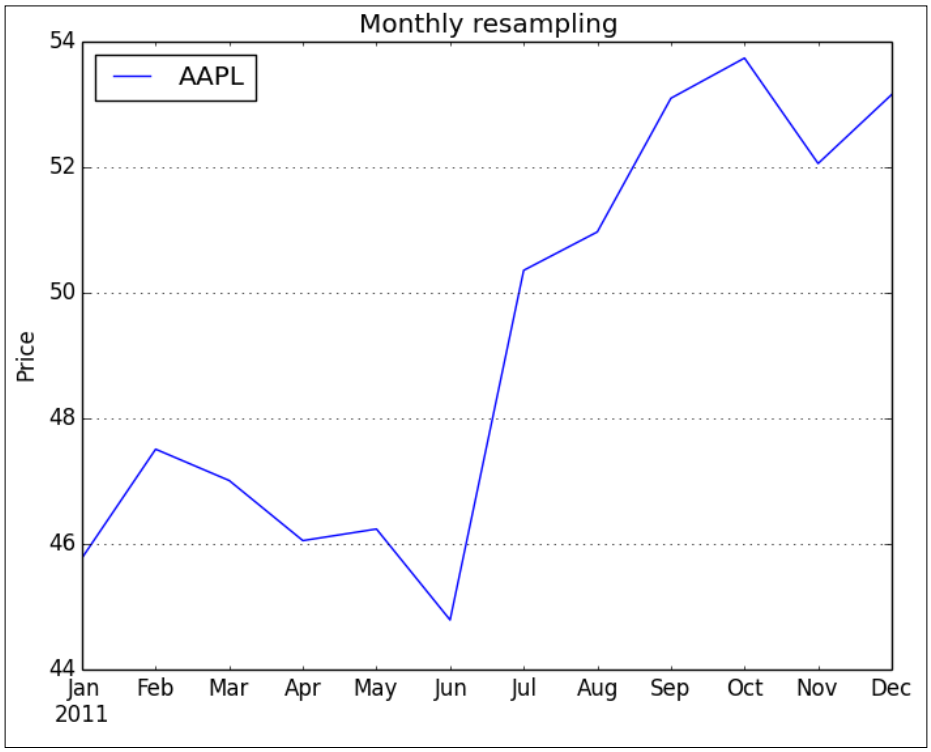
```
2011-04-30  338.960000
2011-05-31  340.324286
2011-06-30  329.664545
2011-07-31  370.647000
2011-08-31  375.151304
2011-09-30  390.816190
2011-10-31  395.532381
2011-11-30  383.170476
2011-12-31  391.251429
```

DataFrame plot()

```
df.plot()
resampled.plot()
plt.show()
```



resampled data has less data points, and therefore, the resulting plot is choppier,




```
from __future__ import print_function
import pandas
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib import finance
import numpy as np

# Download AAPL data for 2011 to 2012
start = datetime(2011, 01, 01)
end = datetime(2012, 01, 01)

symbol = "AAPL"
quotes = finance.quotes_historical_yahoo(symbol, start, end,
asobject=True)
```

```
# Create date time index
dt_idx = pandas.DatetimeIndex(quotes.date)

#Create data frame
df = pandas.DataFrame(quotes.close, index=dt_idx,
columns=[symbol])

# Resample with monthly frequency
resampled = df.resample('M', how=np.mean)
print(resampled)

# Plot
df.plot()
plt.title('AAPL prices')
plt.ylabel('Price')

resampled.plot()
plt.title('Monthly resampling')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```

How it works...

create a pandas `DataFrame`. We then resampled our time series data. A single character gives the resampling

`D`

`M`

`A` for annual

`how` parameter of the `resample()` method indicates how the data is sampled.

pandas documentation is at `http://pandas.pydata.org/pandas-docs/dev/generated/pandas.DataFrame.resample.html`

11

NumPy

`at()` method

`partition()` function

`nanmean()`, `nanvar()`, and `nanstd()` functions

`full()` and `full_like()` functions

Random sampling with `numpy.random.choice()`

`datetime64`

Introduction

NumPy Cookbook

at() method

`at()`
method indexing is indexing that does not involve

`at()` method is `ufunc.at(a, indices[, b])`
`b`
functions with two operands.

How to do it...

steps demonstrate how the `at()`

Create an `7` random integers from `-4` to `4` with a seed of `44`:

```
np.random.seed(44)
a = np.random.random_integers(-4, 4, 7)
print(a)
```

```
[ 0 -1 -3 -1 -4  0 -1]
```

`at()` method of the `sign`

```
np.sign.at(a, [2, 4])
print(a)
```

```
[ 0 -1 -1 -1 -1  0 -1]
```


universal function documentation is at `http://docs.scipy.org/doc/numpy/reference/ufuncs.html`

`partition()`
normal sorting.

Ref orting for more
information. A

order within the set of the top elements.

`partition()`

How to do it...

```
np.random.seed(20)
a = np.random.random_integers(0, 7, 9)
print(a)
```

```
[3 2 7 7 4 2 1 4 3]
```

```
print(np.partition(a, 4))
```

```
[2 3 1 2 3 7 7 4 4]
```

How it works...

We
the middle, at index `4`,

[redacted]

`http://docs.scipy.org/doc/numpy/reference/generated/numpy.partition.html`

[redacted]

It is the arithmetic mean, variance, and standard deviation of a set of data are.

A **jackknife resampling** (refer to `http://en.wikipedia.org/wiki/Jackknife_resampling`
resampling is to create datasets from the
In essence, we are attempting to estimate what will occur if at least one of the values is

How to do it...

`nanmean()`, `nanvar()`, and `nanstd()`

```
estimates = np.zeros((len(a), 3))
```

```
for i in xrange(len(a)):
   b = a.copy()
   b[i] = np.nan
   estimates[i,] = [np.nanmean(b), np.nanvar(b),
     np.nanstd(b)]
```

3.

```
print("Estimator variance", estimates.var(axis=0))
```

```
Estimator variance [ 0.00079905  0.00090129  0.00034604]
```

How it works...

We estimated the variances of the arithmetic mean, variance, and standard deviation of a

code for this recipe is in the `jackknife.py`

```
from __future__ import print_function
import numpy as np

np.random.seed(46)
a = np.random.randn(30)
estimates = np.zeros((len(a), 3))

for i in xrange(len(a)):
   b = a.copy()
   b[i] = np.nan

   estimates[i,] = [np.nanmean(b), np.nanvar(b), np.nanstd(b)]

print("Estimator variance", estimates.var(axis=0))
```


documentation page for `nanmean()` is at `http://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.nanmean.html`

documentation page for `nanvar()` is at `http://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.nanvar.html`

documentation page for `nanstd()` is at `http://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.nanstd.html`

[redacted]

`full()` and `full_like()` functions are new additions

```
>>> help(np.full)
Return a new array of given shape and type, filled with `fill_value`.
>>> help(np.full_like)
Return a full array with the same shape and type as a given array.
```

How to do it...

see how `full()` and `full_like()`

Create a `1 2` `full()` `7`

```
print(np.full((1, 2), 7))
```

```
array([[ 7.,  7.]])
```

```
print(np.full((1, 2), 7, dtype=np.int))
```

```
array([[7, 7]])
```

3. `full_like()`
`linspace()`

template for the `full_like()`

```
a = np.linspace(0, 1, 5)
print(a)
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
print(np.full_like(a, 7))
array([ 7.,  7.,  7.,  7.,  7.])
```

Again, we `7`
use

```
print(np.full_like(a, 7, dtype=np.int))
array([7, 7, 7, 7, 7])
```

How it works...

We produced `full()` and `full_like()` `full()`
`7` `full_like()`

choice()

Bootstrapping is a procedure similar to

samples from the original data of size *N*. Visualize the original data sample

For each generated sample, we compute the statistical estimator of interest (for example, the arithmetic mean).

How to do it...

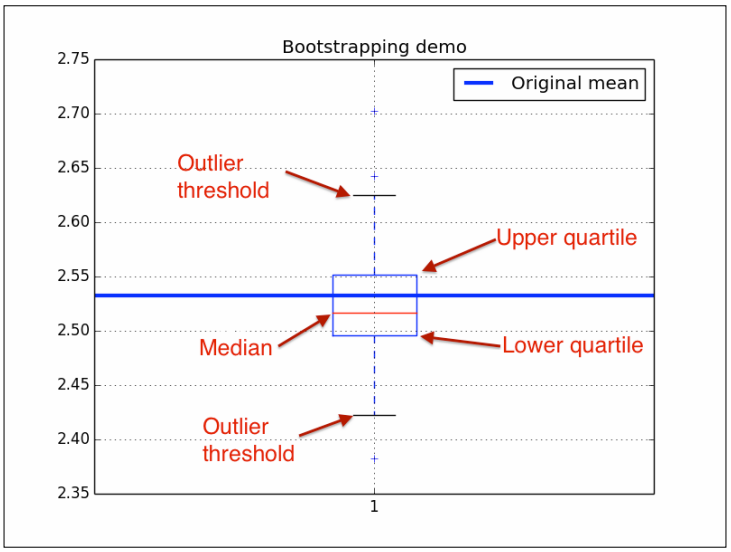
We will `numpy.random.choice()`

```
N = 400
np.random.seed(28)
data = np.random.binomial(5, .5, size=N)
```

```
bootstrapped = np.random.choice(data, size=(N, 30))
means = bootstrapped.mean(axis=0)
```

3. `matplotlib`

```
plt.title('Bootstrapping demo')
plt.grid()
plt.boxplot(means)
plt.plot(3 * [data.mean()], lw=3, label='Original mean')
plt.legend(loc='best')
plt.show()
```



How it works...

`numpy.random.choice()` `matplotlib`

*1.5 * (Q3 – Q1)* **interquartile range**.

`numpy.random.choice()` documentation is at `http://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.random.choice.html`

`matplotlib boxplot()` function documentation is at `http://matplotlib.org/api/pyplot_api.html`

`http://en.wikipedia.org/wiki/Box_plot`

datetime64

How to do it...

datetime64

Create a datetime64

```
print(np.datetime64('2015-05-21'))
```

```
numpy.datetime64('2015-05-21')
```

We created a datetime64 YYYY-MM-DD format, where Y M corresponds to the month, and D corresponds to

standard for representing dates and times.

YYYY-MM-DD, YYYY-MM, and YYYYMMDD

```
print(np.datetime64('20150521'))
print(np.datetime64('2015-05'))
```

```
numpy.datetime64('20150521')
numpy.datetime64('2015-05')
```

3.

T[hh:mm:ss]

```
local = np.datetime64('1578-01-01T21:18')
print(local)
```

```
numpy.datetime64('1578-01-01T21:18Z')
```

A string in the `-[hh:mm]`
can create a `datetime64`

```
with_offset = np.datetime64('1578-01-01T21:18-0800')
print(with_offset)
```

```
numpy.datetime64('1578-01-02T05:18Z')
```

```
   Z

            datetime64
```

```
print(local - with_offset)
```

```
numpy.timedelta64(-480,'m')
```

```
                timedelta64
```

How it works...

```
            datetime64

                                        datetime_demo.py
```

```
import numpy as np

print(np.datetime64('2015-05-21'))
#numpy.datetime64('2015-05-21')

print(np.datetime64('20150521'))
print(np.datetime64('2015-05'))
#numpy.datetime64('20150521')
#numpy.datetime64('2015-05')

local = np.datetime64('1578-01-01T21:18')
print(local)
#numpy.datetime64('1578-01-01T21:18Z')

with_offset = np.datetime64('1578-01-01T21:18-0800')
print(with_offset)
#numpy.datetime64('1578-01-02T05:18Z')

print(local - with_offset)
```

http://docs.scipy.org/doc/numpy/reference/arrays.datetime.html

http://en.wikipedia.org/wiki/ISO_8601

12

Predictive Data

Exploring atmospheric pressure

Introduction

exploratory and **predictive data analysis**

weather station at

http://www.knmi.nl/climatology/daily_data/download.html).

.npy format,

YYYYMMDD format

In this recipe, we

Chapter 10, *Fun with Scikits*

exploring.py

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.stats.adnorm import normal_ad

data = np.load('cbk12.npy')

# Multiply to get hPa values
meanp = .1 * data[:,1]

# Filter out 0 values
meanp = meanp[ meanp > 0]

# Get descriptive statistics
print("Max", meanp.max())
print("Min", meanp.min())
mean = meanp.mean()
print("Mean", mean)
print("Median", np.median(meanp))
std = meanp.std()
```

```
print("Std dev", std)

# Check for normality
print("Normality", normal_ad(meanp))

#histogram with Gaussian PDF
plt.subplot(211)
plt.title('Histogram of average atmospheric pressure')
_, bins, _ = plt.hist(meanp, np.sqrt(len(meanp)), normed=True)
plt.plot(bins, 1/(std * np.sqrt(2 * np.pi)) * np.exp(- (bins -
mean)**2/(2 * std**2)), 'r-', label="Gaussian PDF")
plt.grid()
plt.legend(loc='best')
plt.xlabel('Average atmospheric pressure (hPa)')
plt.ylabel('Frequency')

# boxplot
plt.subplot(212)
plt.boxplot(meanp)
plt.title('Boxplot of average atmospheric pressure')
plt.ylabel('Average atmospheric pressure (hPa)')
plt.grid()

# Improves spacing of subplots
plt.tight_layout()
plt.show()
```

Install `statsmodels`, if *Installing scikits-statsmodels* recipe *Chapter 10*, *Fun with Scikits*).

How to do it...

load()

```
data = np.load('cbk12.npy')
```

values to get values in `hPa` and remove `0`

```
# Multiply to get hPa values
meanp = .1 * data[:,1]

# Filter out 0 values
meanp = meanp[ meanp > 0]
```

3.

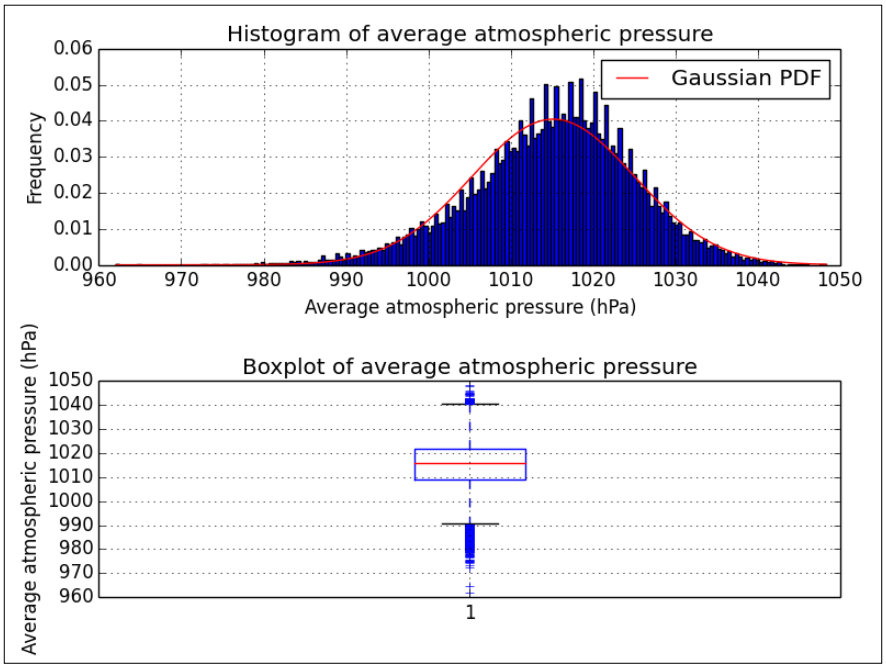
```
print("Max", meanp.max())
print("Min", meanp.min())
mean = meanp.mean()
print("Mean", mean)
print("Median", np.median(meanp))
std = meanp.std()
print("Std dev", std)
```

```
Max 1048.3
Min 962.1
Mean 1015.14058231
Median 1015.8
Std dev 9.85889134337
```

Chapter 10, Fun with Scikits

```
print("Normality", normal_ad(meanp))
```

```
Normality (72.685781095773564, 0.0)
```



Performing a normality test with statsmodels recipe from *Chapter 10*, *Fun with Scikits*

For an explanation of *Random sampling with numpy.random.choice()* recipe from *Chapter 11*, *Latest and Greatest NumPy*

documentation of the `load()` function is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.load.html`

`NaN` values. Also, we will

corresponding code is in the `day_range.py`

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
import calendar as cal

data = np.load('cbk12.npy')

# Multiply to get hPa values
highs = .1 * data[:,2]
lows = .1 * data[:,3]

# Filter out 0 values
highs[highs == 0] = np.nan
lows[lows == 0] = np.nan

# Calculate range and stats
ranges = highs - lows
print("Minimum daily range", np.nanmin(ranges))
print("Maximum daily range", np.nanmax(ranges))

print("Average daily range", np.nanmean(ranges))
print("Standard deviation", np.nanstd(ranges))
```

```
# Get months
dates = data[:,0]
months = (dates % 10000)/100
months = months[~np.isnan(ranges)]

monthly = []
month_range = np.arange(1, 13)

for month in month_range:
   indices = np.where(month == months)
   monthly.append(np.nanmean(ranges[indices]))

plt.bar(month_range, monthly)
plt.title('Monthly average of daily pressure ranges')
plt.xticks(month_range, cal.month_abbr[1:13])
plt.ylabel('Monthly Average (hPa)')
plt.grid()
plt.show()
```

How to do it...

We could leave missing values at their current `0`
set them to `NaN` to avoid confusion. Set the missing values to `NaN`

```
highs[highs == 0] = np.nan
lows[lows == 0] = np.nan
```

Compute the ranges, minima, maxima, mean, and standard deviations with the `nanmin()`, `nanmax()`, `nanmean()`, and `nanstd()`

```
ranges = highs - lows
print("Minimum daily range", np.nanmin(ranges))
print("Maximum daily range", np.nanmax(ranges))

print("Average daily range", np.nanmean(ranges))
print("Standard deviation", np.nanstd(ranges))
```

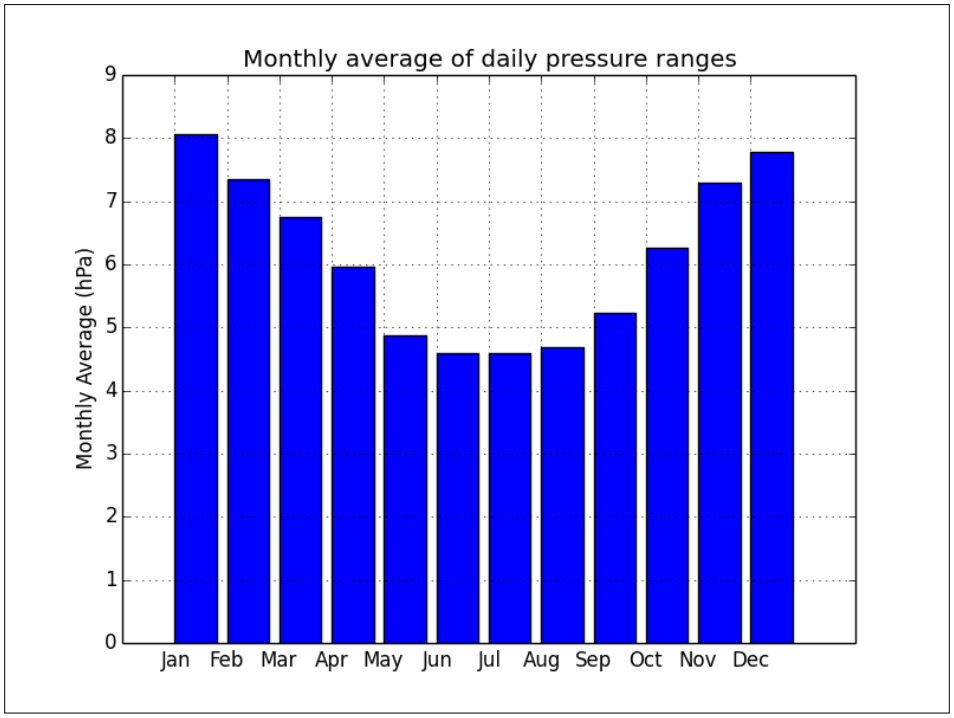
```
Minimum daily range 0.4
Maximum daily range 41.7
Average daily range 6.11945360571
Standard deviation 4.42162136692
```

3. YYYYMMDD format. With a

corresponding to the NaN

```
dates = data[:,0]
months = (dates % 10000)/100
months = months[~np.isnan(ranges)]


monthly = []
month_range = np.arange(1, 13)

for month in month_range:
   indices = np.where(month == months)
   monthly.append(np.nanmean(ranges[indices]))
```

In the last step, we draw a matplotlib



How it works...

Exploring atmospheric pressure recipe

annual.py

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


data = np.load('cbk12.npy')

# Multiply to get hPa values
avgs = .1 * data[:,1]
highs = .1 * data[:,2]
lows = .1 * data[:,3]

# Filter out 0 values
avgs = np.ma.array(avgs, mask = avgs == 0)
lows = np.ma.array(lows, mask = lows == 0)
highs = np.ma.array(highs, mask = highs == 0)

# Get years
years = data[:,0]/10000

# Initialize annual stats arrays
y_range = np.arange(1901, 2014)
nyears = len(y_range)
y_avgs = np.zeros(nyears)
```

```
y_highs = np.zeros(nyears)
y_lows = np.zeros(nyears)

# Compute stats
for year in y_range:
   indices = np.where(year == years)
   y_avgs[year - 1901] = np.mean(avgs[indices])
   y_highs[year - 1901] = np.max(highs[indices])
   y_lows[year - 1901] = np.min(lows[indices])

plt.title('Annual atmospheric pressure for De Bilt(NL)')
plt.ticklabel_format(useOffset=900, axis='y')

plt.plot(y_range, y_avgs, label='Averages')

# Plot ignoring NaNs
h_mask = np.isfinite(y_highs)
plt.plot(y_range[h_mask], y_highs[h_mask], '^', label='Highs')

l_mask = np.isfinite(y_lows)
plt.plot(y_range[l_mask], y_lows[l_mask], 'v', label='Lows')

plt.xlabel('Year')
plt.ylabel('Atmospheric pressure (hPa)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

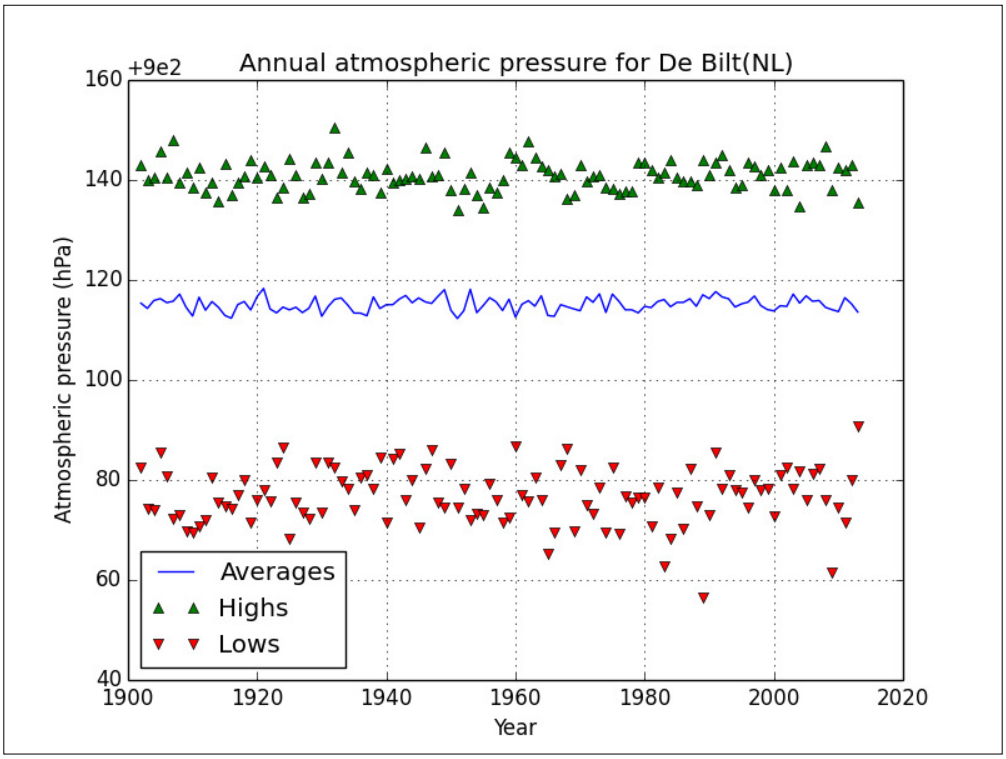
How to do it...

```
y_range = np.arange(1901, 2014)
nyears = len(y_range)
y_avgs = np.zeros(nyears)
y_highs = np.zeros(nyears)
y_lows = np.zeros(nyears)
```

```
for year in y_range:
   indices = np.where(year == years)
   y_avgs[year - 1901] = np.mean(avgs[indices])
   y_highs[year - 1901] = np.max(highs[indices])
   y_lows[year - 1901] = np.min(lows[indices])
```

3. NaN

```
h_mask = np.isfinite(y_highs)
plt.plot(y_range[h_mask], y_highs[h_mask], '^', label='Highs')

l_mask = np.isfinite(y_lows)
plt.plot(y_range[l_mask], y_lows[l_mask], 'v', label='Lows')
```



How it works...

used the `isfinite()` function to ignore the `NaN`
`NaN` values.

Exploring atmospheric pressure recipe

`isfinite()` function documentation is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.isfinite.html`

"Maximum visibility; 0: <100 m, 1:100-200 m, 2:200-300 m,..., 49:4900-5000 m, 50:5-6 km, 56:6-7 km, 57:7-8 km,..., 79:29-30 km, 80:30-35 km, 81:35-40 km,..., 89: >70 km)"

that we have a lot of `0`

`10` and `20` in

to `20` and `50`

`79`

`visibility.py`

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

data = np.load('cbk12.npy')

# Get minimum visibility
visibility = data[:,4]
```

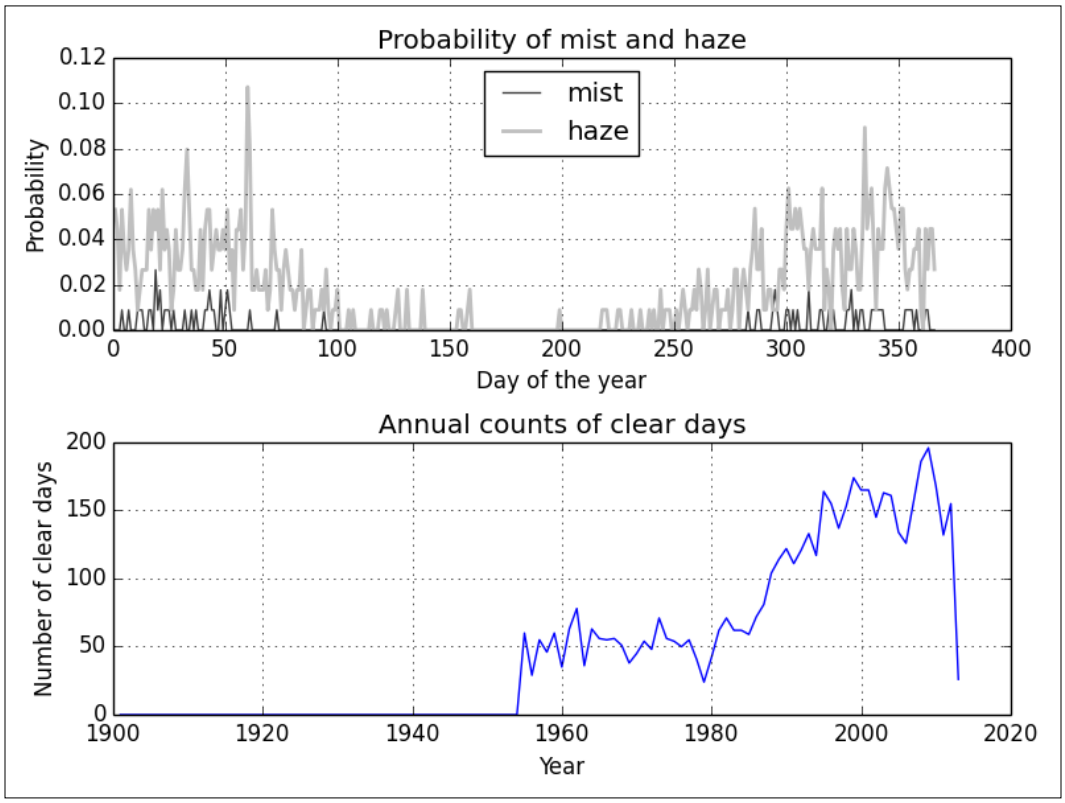
```
# doy
doy = data[:,0] % 10000

doy_range = np.unique(doy)

# Initialize arrays
ndoy = len(doy_range)
mist = np.zeros(ndoy)
haze = np.zeros(ndoy)


# Compute frequencies
for i, d in enumerate(doy_range):
   indices = np.where(d == doy)
   selection = visibility[indices]

   mist_truth = (10 < selection) & (selection < 20)
   mist[i] = len(selection[mist_truth])/(1. * len(selection))

   haze_truth = (20 < selection) & (selection < 50)
   haze[i] = len(selection[haze_truth])/(1. * len(selection))

# Get years
years = data[:,0]/10000

# Initialize annual stats arrays
y_range = np.arange(1901, 2014)
nyears = len(y_range)
y_counts = np.zeros(nyears)

# Get annual counts
for year in y_range:
   indices = np.where(year == years)
   selection = visibility[indices]
   y_counts[year - 1901] = len(selection[selection > 79])

plt.subplot(211)
plt.plot(np.arange(1, 367), mist, color='.25', label='mist')
plt.plot(np.arange(1, 367), haze, color='0.75', lw=2, label='haze')
plt.title('Probability of mist and haze')
plt.xlabel('Day of the year')
```

```
plt.ylabel('Probability')
plt.grid()
plt.legend(loc='best')

plt.subplot(212)
plt.plot(y_range, y_counts)
plt.xlabel('Year')
plt.ylabel('Number of clear days')
plt.title('Annual counts of clear days')
plt.grid()
plt.tight_layout()
plt.show()
```

How to do it...

Follow these steps to plot

```
for i, d in enumerate(doy_range):
   indices = np.where(d == doy)
   selection = visibility[indices]

   mist_truth = (10 < selection) & (selection < 20)
   mist[i] = len(selection[mist_truth])/(1. * len(selection))

   haze_truth = (20 < selection) & (selection < 50)
   haze[i] = len(selection[haze_truth])/(1. * len(selection))


for year in y_range:
   indices = np.where(year == years)
   selection = visibility[indices]
   y_counts[year - 1901] = len(selection[selection > 79])
```



How it works...

indicates the existence of a trend.

ignore `0` values as I mentioned earlier.

Exploring atmospheric pressure recipe

Studying annual atmospheric pressure averages recipe

http://en.wikipedia.org/wiki/Visibility

models are therefore called **autoregressive**.

. **Cross-validation** is a common approach to split the data into **train** and **test**

autoregressive.py

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt


data = np.load('cbk12.npy')

# Load average pressure
meanp = .1 * data[:,1]

# Split point for test and train data
cutoff = 0.9 * len(meanp)

for degree, marker in zip(xrange(1, 4), ['o', 'x','.']):
   poly = np.polyfit(meanp[:cutoff - 1], meanp[1:cutoff], degree)
   print('Polynomial coefficients', poly)

   fit = np.polyval(poly, meanp[cutoff:-1])
   error = np.abs(meanp[cutoff + 1:] - fit)/fit
   plt.plot(error, marker, color=str(.25* degree), label='Degree ' +
str(degree))
   plt.plot(np.full(len(error), error.mean()), lw=degree, label='Mean
for degree ' + str(degree))
```

```
    print("Absolute mean relative error", error.mean(), 'for polynomial
of degree', degree)
    print()

plt.title('Relative test errors for polynomial fits')
plt.ylabel('Relative error')
plt.grid()
plt.legend(loc='best')
plt.show()
```

How to do it...

With the

```
cutoff = 0.9 * len(meanp)
```

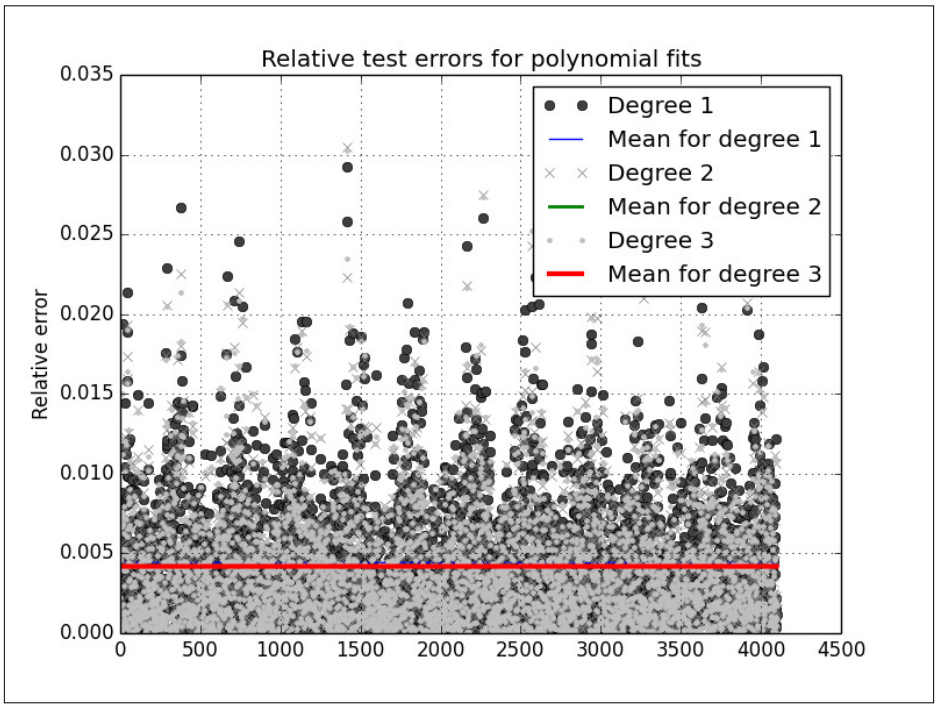
Fit the data with the `polyfit()` and `polyval()`

```
poly = np.polyfit(meanp[:cutoff - 1], meanp[1:cutoff], degree)
print('Polynomial coefficients', poly)

fit = np.polyval(poly, meanp[cutoff:-1])
```

3.

```
error = np.abs(meanp[cutoff + 1:] - fit)/fit
```

```
Polynomial coefficients [ 0.995542    4.50866543]
Absolute mean relative error 0.00442472512506 for polynomial of
degree 1

Polynomial coefficients [ -1.79946321e-04   1.17995347e+00
2.77195814e+00]
Absolute mean relative error 0.00421276856088 for polynomial of
degree 2

Polynomial coefficients [  3.17914507e-06  -6.62444552e-03
4.44558056e+00   2.76520065e+00]
Absolute mean relative error 0.0041906802632 for polynomial of
degree 3
```



How it works...

for atmospheric pressure), which is smaller than a percent. We see some potential outliers, `polyfit()` and `polyval()`

Exploring atmospheric pressure recipe

`http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29`

documentation for `polyfit()` is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html`

documentation for `polyval()` is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyval.html`

A simple pressure is to assume that values dance around a mean **μ**. We then assume in the simplest case that **deviations of consecutive values ε** from the

$$P_t = \mu + \varepsilon_t + \theta\varepsilon_{t-1}$$

θ.

moving_average.py

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime as dt
from scipy.optimize import leastsq


data = np.load('cbk12.npy')

# Load average pressure
meanp = .1 * data[:,1]

cutoff = 0.9 * len(meanp)

def model(p, ma1):
   return p * ma1

def error(p, t, ma1):
   return t - model(p, ma1)

p0 = [.9]
mu = meanp[:cutoff].mean()
params = leastsq(error, p0, args=(meanp[1:cutoff] - mu,
meanp[:cutoff-1] - mu))[0]
print(params)

abs_error = np.abs(error(params, meanp[cutoff+1:] - mu,
meanp[cutoff:-1] - mu))
```

```
plt.plot(abs_error, label='Absolute error')
plt.plot(np.full_like(abs_error, abs_error.mean()), lw=2,
label='Absolute mean error')
plt.title('Absolute error for the moving average model')
plt.ylabel('Absolute error (hPa)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

Installing SciPy recipe of *Chapter 2*, *Advanced Indexing and Array Concepts*.

How to do it...

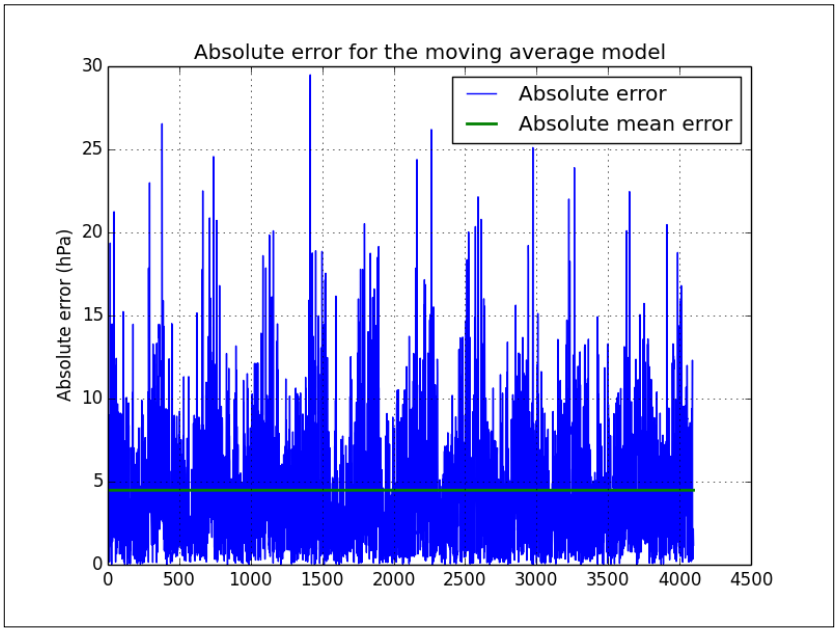
```
def model(p, ma1):
   return p * ma1

def error(p, t, ma1):
   return t - model(p, ma1)
```

`leastsq()` function and initial guess of `0.9`

```
p0 = [.9]
mu = meanp[:cutoff].mean()
params = leastsq(error, p0, args=(meanp[1:cutoff] - mu,
meanp[:cutoff-1] - mu))[0]
```

3.

```
abs_error = np.abs(error(params, meanp[cutoff+1:] - mu,
meanp[cutoff:-1] - mu))
```



How it works...

`leastsq()` function

Exploring atmospheric pressure recipe

`http://en.wikipedia.org/wiki/Moving-average_model`

documentation for `leastsq()` is at `http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.leastsq.html`

atmospheric pressure.

intrayear.py

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

data = np.load('cbk12.npy')

# Multiply to get hPa values
meanp = .1 * data[:,1]


# Get months
dates = data[:,0]
months = (dates % 10000)/100

monthly = []
vars = np.zeros(12)
month_range = np.arange(1, 13)

for month in month_range:
   indices = np.where(month == months)
   selection = meanp[indices]

   # Filter out 0 values
   selection = selection[selection > 0]

   monthly.append(selection)
   vars[month - 1] = np.var(selection)

def plot():
    plt.xticks(month_range, cal.month_abbr[1:13])
    plt.grid()
    plt.xlabel('Month')

plt.subplot(211)
plot()
plt.title('Atmospheric pressure box plots')
plt.boxplot(monthly)
plt.ylabel('Atmospheric pressure (hPa)')

plt.subplot(212)
```

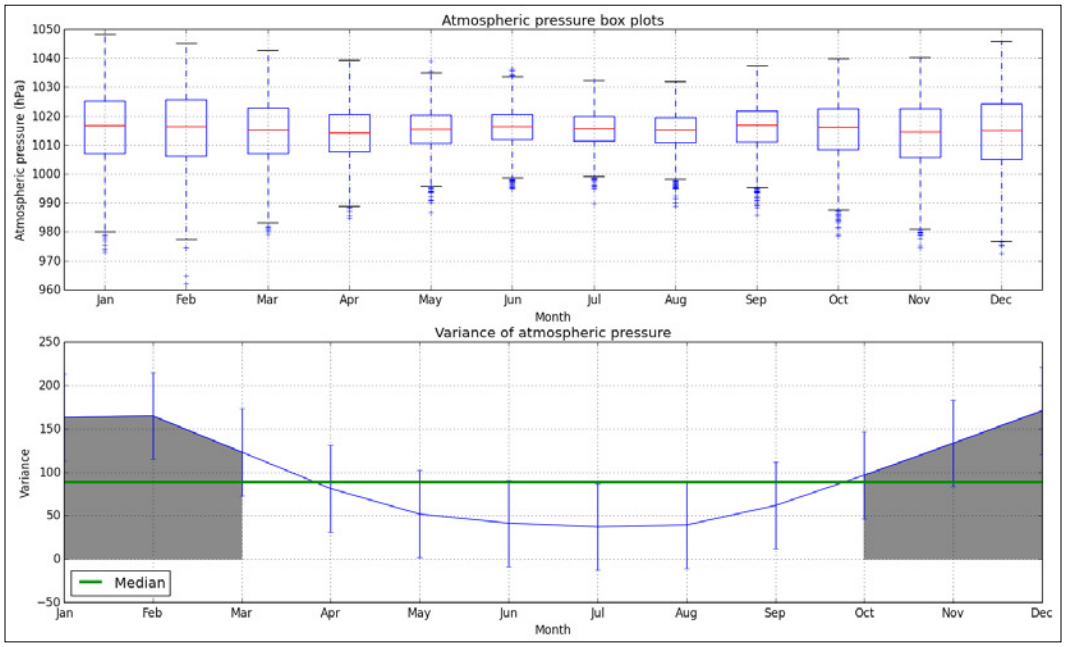
```
plot()

# Display error bars using standard deviation
plt.errorbar(month_range, vars, yerr=vars.std())
plt.plot(month_range, np.full_like(month_range, np.median(vars)),
lw=3, label='Median')

# Shades the region above the median
plt.fill_between(month_range, vars, where=vars>np.median(vars),
color='0.5')
plt.title('Variance of atmospheric pressure')
plt.ylabel('Variance')
plt.legend(loc='best')

plt.show()
```

How to do it...

While we are

```
plt.errorbar(month_range, vars, yerr=vars.std())
```

```
plt.fill_between(month_range, vars,
where=vars>np.median(vars), color='0.5')
```



How it works...

We matched months to measurements of atmospheric pressure. We used the matches

and

Exploring atmospheric pressure recipe

Studying annual atmospheric pressure averages recipe

documentation for `var()` is at `http://docs.scipy.org/doc/numpy/reference/generated/numpy.var.html`

pressure

Outliers

the **interquartile range**

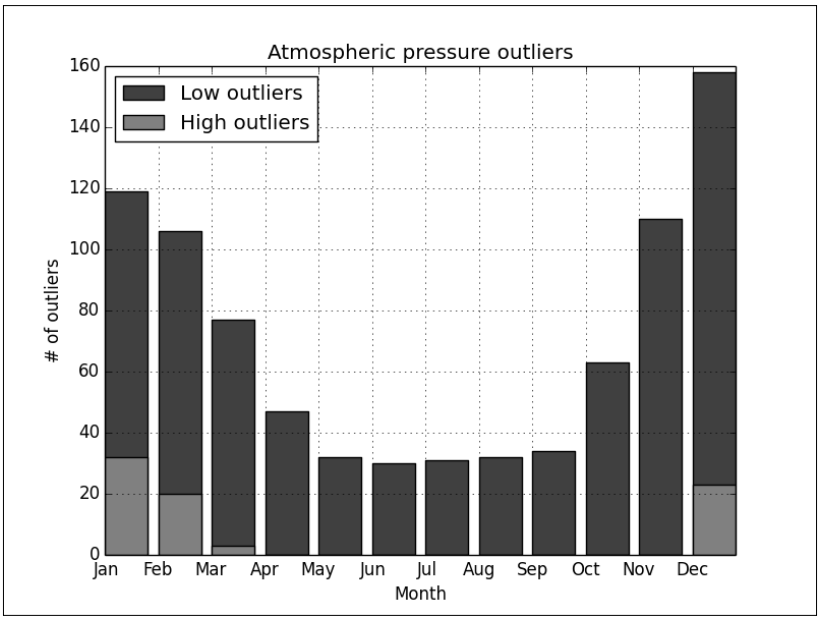
extreme.py

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

data = np.load('cbk12.npy')

# Multiply to get hPa values
meanp = .1 * data[:,1]

# Filter out 0 values
meanp = np.ma.array(meanp, mask = meanp == 0)

# Calculate quartiles and irq
q1 = np.percentile(meanp, 25)
median = np.percentile(meanp, 50)
q3 = np.percentile(meanp, 75)

irq = q3 - q1

# Get months
dates = data[:,0]
months = (dates % 10000)/100

m_low = np.zeros(12)
m_high = np.zeros(12)
month_range = np.arange(1, 13)

for month in month_range:
   indices = np.where(month == months)
   selection = meanp[indices]
   m_low[month - 1] = len(selection[selection < (q1 - 1.5 * irq)])
   m_high[month - 1] = len(selection[selection > (q3 + 1.5 * irq)])

plt.xticks(month_range, cal.month_abbr[1:13])
plt.bar(month_range, m_low, label='Low outliers', color='.25')
plt.bar(month_range, m_high, label='High outliers', color='0.5')
plt.title('Atmospheric pressure outliers')
plt.xlabel('Month')
```

```
plt.ylabel('# of outliers')
plt.grid()
plt.legend(loc='best')
plt.show()
```

How to do it...

```
                                                    percentile()
q1 = np.percentile(meanp, 25)
median = np.percentile(meanp, 50)
q3 = np.percentile(meanp, 75)

irq = q3 - q1


for month in month_range:
   indices = np.where(month == months)
   selection = meanp[indices]
   m_low[month - 1] = len(selection[selection < (q1 - 1.5 * irq)])
   m_high[month - 1] = len(selection[selection > (q3 + 1.5 *
irq)])
```



How it works...

`percentile()`

Exploring atmospheric pressure recipe

documentation for the `percentile()` function is at `http://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.percentile.html`

T

U

V

W

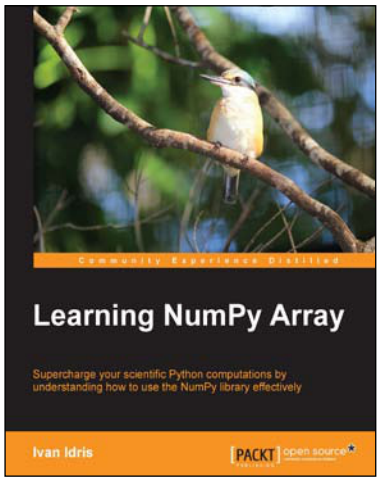
NumPy Cookbook
Second Edition

Mastering phpMyAdmin for Effective MySQL Management

www.packtpub.com.

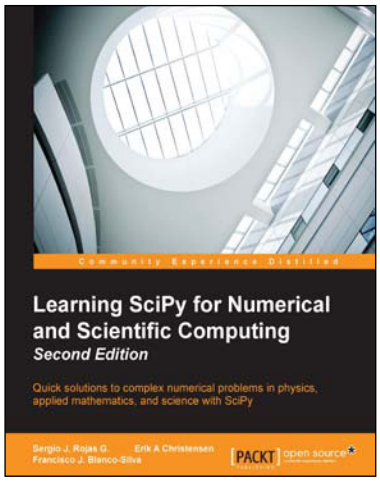
author@packtpub.com

[PACKT] open source
community experience distilled
PUBLISHING



mathematical computations.

simple manner.



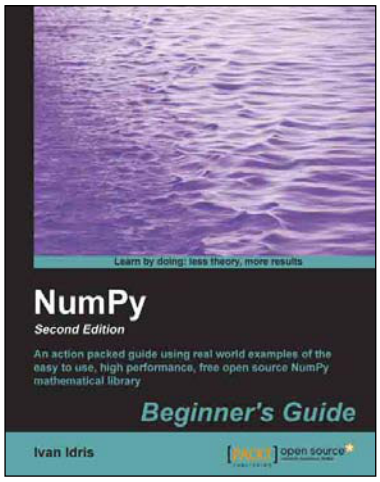
Second Edition

perform standard mathematical operations

functional form.

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
community experience distilled
PUBLISHING



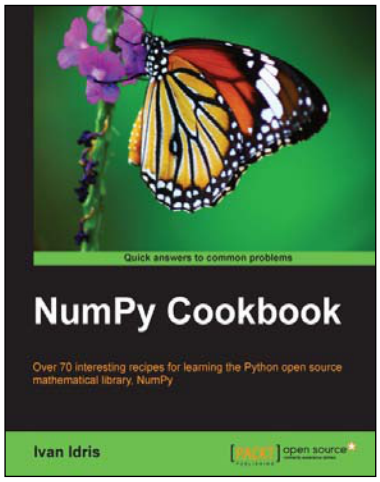
NumPy Beginner's Guide

Second Edition

ISBN: 9978-1-78216-608-5 Paperback: 310 pages

An action packed guide using real world examples of the easy to use, high performance, free open source NumPy mathematical library

1. Perform high performance calculations with clean and efficient NumPy code.
2. Analyze large data sets with statistical functions.
3. Execute complex linear algebra and mathematical computations.



NumPy Cookbook

ISBN: 978-1-84951-892-5 Paperback: 226 pages

Over 70 interesting recipes for learning the Python open source mathematical library, NumPy

1. Do high performance calculations with clean and efficient NumPy code.
2. Analyze large sets of data with statistical functions.
3. Execute complex linear algebra and mathematical computations.