Statistics and Machine Learning in Python

Release 0.3 beta

Edouard Duchesnay, Tommy Löfstedt, Feki Younes

Nov 13, 2019

CONTENTS

1 **Introduction** **1**
- 1.1 Python ecosystem for data-science . . . 1
- 1.2 Introduction to Machine Learning . . . 5
- 1.3 Data analysis methodology . . . 6

2 **Python language** **9**
- 2.1 Import libraries . . . 9
- 2.2 Basic operations . . . 9
- 2.3 Data types . . . 10
- 2.4 Execution control statements . . . 16
- 2.5 Functions . . . 19
- 2.6 List comprehensions, iterators, etc. . . . 20
- 2.7 Regular expression . . . 21
- 2.8 System programming . . . 22
- 2.9 Scripts and argument parsing . . . 28
- 2.10 Networking . . . 29
- 2.11 Modules and packages . . . 30
- 2.12 Object Oriented Programming (OOP) . . . 30
- 2.13 Exercises . . . 32

3 **Scientific Python** **33**
- 3.1 Numpy: arrays and matrices . . . 33
- 3.2 Pandas: data manipulation . . . 42
- 3.3 Matplotlib: data visualization . . . 54

4 **Statistics** **69**
- 4.1 Univariate statistics . . . 69
- 4.2 Lab 1: Brain volumes study . . . 107
- 4.3 Multivariate statistics . . . 117
- 4.4 Time Series in python . . . 130

5 **Machine Learning** **147**
- 5.1 Dimension reduction and feature extraction . . . 147
- 5.2 Clustering . . . 162
- 5.3 Linear methods for regression . . . 171
- 5.4 Linear classification . . . 184
- 5.5 Non linear learning algorithms . . . 201
- 5.6 Resampling Methods . . . 205
- 5.7 Ensemble learning: bagging, boosting and stacking . . . 218

5.8 Gradient descent 233

6 Deep Learning **243**

6.1 Backpropagation 243

6.2 Multilayer Perceptron (MLP) 257

6.3 Convolutional neural network 276

6.4 Transfer Learning Tutorial 304

7 Indices and tables **315**

CHAPTER

ONE

INTRODUCTION

1.1 Python ecosystem for data-science

1.1.1 Python language

- Interpreted
- Garbage collector (do not prevent from memory leak)
- Dynamically-typed language (Java is statically typed)

1.1.2 Anaconda

Anaconda is a python distribution that ships most of python tools and libraries

Installation

1. Download anaconda (Python 3.x) http://continuum.io/downloads

2. Install it, on Linux

```
bash Anaconda3-2.4.1-Linux-x86_64.sh
```

3. Add anaconda path in your PATH variable in your `.bashrc` file:

```
export PATH="${HOME}/anaconda3/bin:$PATH"
```

Managing with "conda"

Update conda package and environment manager to current version

```
conda update conda
```

Install additional packages. Those commands install qt back-end (Fix a temporary issue to run spyder)

```
conda install pyqt
conda install PyOpenGL
conda update --all
```

Install seaborn for graphics

```
conda install seaborn
# install a specific version from anaconda chanel
conda install -c anaconda pyqt=4.11.4
```

List installed packages

```
conda list
```

Search available packages

```
conda search pyqt
conda search scikit-learn
```

Environments

- A conda environment is a directory that contains a specific collection of conda packages that you have installed.
- Control packages environment for a specific purpose: collaborating with someone else, delivering an application to your client,
- Switch between environments

List of all environments

:: conda info –envs

1. Create new environment
2. Activate
3. Install new package

```
conda create --name test
# Or
conda env create -f environment.yml
source activate test
conda info --envs
conda list
conda search -f numpy
conda install numpy
```

Miniconda

Anaconda without the collection of (>700) packages. With Miniconda you download only the packages you want with the conda command: `conda install PACKAGENAME`

1. Download anaconda (Python 3.x) https://conda.io/miniconda.html
2. Install it, on Linux

```
bash Miniconda3-latest-Linux-x86_64.sh
```

3. Add anaconda path in your PATH variable in your `.bashrc` file:

```
export PATH=${HOME}/miniconda3/bin:$PATH
```

4. Install required packages

```
conda install -y scipy
conda install -y pandas
conda install -y matplotlib
conda install -y statsmodels
conda install -y scikit-learn
conda install -y sqlite
conda install -y spyder
conda install -y jupyter
```

1.1.3 Commands

python: python interpreter. On the dos/unix command line execute wholes file:

```
python file.py
```

Interactive mode:

```
python
```

Quite with `CTL-D`

ipython: advanced interactive python interpreter:

```
ipython
```

Quite with `CTL-D`

pip alternative for packages management (update `-U` in user directory `--user`):

```
pip install -U --user seaborn
```

For neuroimaging:

```
pip install -U --user nibabel
pip install -U --user nilearn
```

spyder: IDE (integrated development environment):

- Syntax highlighting.
- Code introspection for code completion (use `TAB`).
- Support for multiple Python consoles (including IPython).
- Explore and edit variables from a GUI.
- Debugging.
- Navigate in code (go to function definition) `CTL`.

3 or 4 panels:

text editor	help/variable explorer
	ipython interpreter

Shortcuts: - `F9` run line/selection

1.1.4 Libraries

scipy.org: https://www.scipy.org/docs.html

Numpy: Basic numerical operation. Matrix operation plus some basic solvers.:

```
import numpy as np
X = np.array([[1, 2], [3, 4]])
#v = np.array([1, 2]).reshape((2, 1))
v = np.array([1, 2])
np.dot(X, v) # no broadcasting
X * v # broadcasting
np.dot(v, X)
X - X.mean(axis=0)
```

Scipy: general scientific libraries with advanced solver:

```
import scipy
import scipy.linalg
scipy.linalg.svd(X, full_matrices=False)
```

Matplotlib: visualization:

```
import numpy as np
import matplotlib.pyplot as plt
#%matplotlib qt
x = np.linspace(0, 10, 50)
sinus = np.sin(x)
plt.plot(x, sinus)
plt.show()
```

Pandas: Manipulation of structured data (tables). input/output excel files, etc.

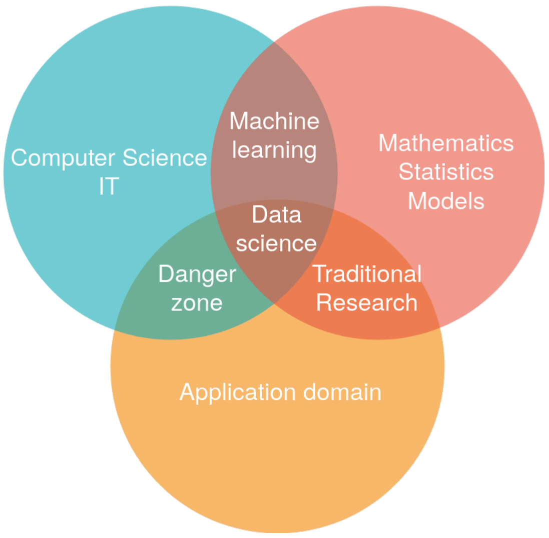
Statsmodel: Advanced statistics

Scikit-learn: Machine learning

library	Arrays data, Num. comp, I/O	Structured data, I/O	Solvers: basic	Solvers: advanced	Stats: basic	Stats: advanced	Machine learning
Numpy	X		X				
Scipy			X	X	X		
Pandas		X					
Statmodels					X	X	
Scikit-learn							X

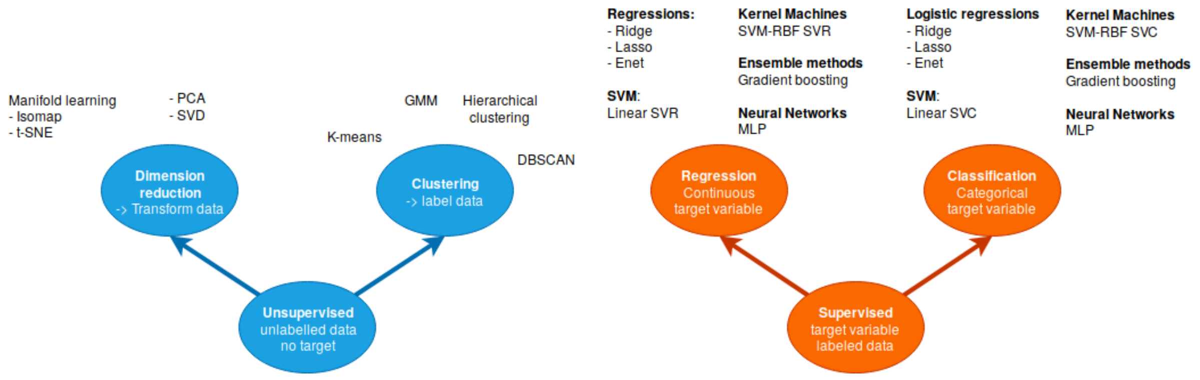
1.2 Introduction to Machine Learning

1.2.1 Machine learning within data science



Machine learning covers two main types of data analysis:

1. Exploratory analysis: **Unsupervised learning**. Discover the structure within the data. E.g.: Experience (in years in a company) and salary are correlated.
2. Predictive analysis: **Supervised learning**. This is sometimes described as **"learn from the past to predict the future"**. Scenario: a company wants to detect potential future clients among a base of prospects. Retrospective data analysis: we go through the data constituted of previous prospected companies, with their characteristics (size, domain, localization, etc…). Some of these companies became clients, others did not. The question is, can we possibly predict which of the new companies are more likely to become clients, based on their characteristics based on previous observations? In this example, the training data consists of a set of *n* training samples. Each sample, x_i, is a vector of *p* input features (company characteristics) and a target feature ($y_i \in \{Yes, No\}$ (whether they became a client or not).



1.2.2 IT/computing science tools

- High Performance Computing (HPC)
- Data flow, data base, file I/O, etc.

- Python: the programming language.
- Numpy: python library particularly useful for handling of raw numerical data (matrices, mathematical operations).
- Pandas: input/output, manipulation structured data (tables).

1.2.3 Statistics and applied mathematics

- Linear model.
- Non parametric statistics.
- Linear algebra: matrix operations, inversion, eigenvalues.

1.3 Data analysis methodology

1. **Formalize customer's needs into a learning problem:**
    - **A target variable: supervised problem.**
        - Target is qualitative: classification.
        - Target is quantitative: regression.
    - **No target variable: unsupervised problem**
        - Vizualisation of high-dimensional samples: PCA, manifolds learning, etc.
        - Finding groups of samples (hidden structure): clustering.
2. **Ask question about the datasets**
    - Number of samples
    - Number of variables, types of each variable.
3. **Define the sample**
    - For prospective study formalize the experimental design: inclusion/exlusion criteria. The conditions that define the acquisition of the dataset.
    - For retrospective study formalize the experimental design: inclusion/exlusion criteria. The conditions that define the selection of the dataset.
4. In a document formalize (i) the project objectives; (ii) the required learning dataset (more specifically the input data and the target variables); (iii) The conditions that define the acquisition of the dataset. In this document, warn the customer that the learned algorithms may not work on new data acquired under different condition.
5. Read the learning dataset.
6. (i) Sanity check (basic descriptive statistics); (ii) data cleaning (impute missing data, recoding); Final Quality Control (QC) perform descriptive statistics and think ! (remove possible confounding variable, etc.).
7. Explore data (visualization, PCA) and perform basic univariate statistics for association between the target an input variables.
8. Perform more complex multivariate-machine learning.

9. Model validation using a left-out-sample strategy (cross-validation, etc.).
10. Apply on new data.

CHAPTER

TWO

PYTHON LANGUAGE

Note: Click *here* to download the full example code

Source Kevin Markham https://github.com/justmarkham/python-reference

2.1 Import libraries

```
# 'generic import' of math module
import math
math.sqrt(25)

# import a function
from math import sqrt
sqrt(25)    # no longer have to reference the module

# import multiple functions at once
from math import cos, floor

# import all functions in a module (generally discouraged)
# from os import *

# define an alias
import numpy as np

# show all functions in math module
content = dir(math)
```

2.2 Basic operations

```
# Numbers
10 + 4          # add (returns 14)
10 - 4          # subtract (returns 6)
10 * 4          # multiply (returns 40)
10 ** 4         # exponent (returns 10000)
10 / 4          # divide (returns 2 because both types are 'int')
10 / float(4)   # divide (returns 2.5)
5 % 4           # modulo (returns 1) - also known as the remainder
```

(continues on next page)

(continued from previous page)

```
10 / 4          # true division (returns 2.5)
10 // 4         # floor division (returns 2)


# Boolean operations
# comparisons (these return True)
5 > 3
5 >= 3
5 != 3
5 == 5

# boolean operations (these return True)
5 > 3 and 6 > 3
5 > 3 or 5 < 3
not False
False or not False and True     # evaluation order: not, and, or
```

2.3 Data types

```
# determine the type of an object
type(2)         # returns 'int'
type(2.0)       # returns 'float'
type('two')     # returns 'str'
type(True)      # returns 'bool'
type(None)      # returns 'NoneType'

# check if an object is of a given type
isinstance(2.0, int)            # returns False
isinstance(2.0, (int, float))   # returns True

# convert an object to a given type
float(2)
int(2.9)
str(2.9)

# zero, None, and empty containers are converted to False
bool(0)
bool(None)
bool('')    # empty string
bool([])    # empty list
bool({})    # empty dictionary

# non-empty containers and non-zeros are converted to True
bool(2)
bool('two')
bool([2])
```

2.3.1 Lists

Different objects categorized along a certain ordered sequence, lists are ordered, iterable, mutable (adding or removing objects changes the list size), can contain multiple data types .. chunk-chap13-001

```
# create an empty list (two ways)
empty_list = []
empty_list = list()

# create a list
simpsons = ['homer', 'marge', 'bart']

# examine a list
simpsons[0]     # print element 0 ('homer')
len(simpsons)   # returns the length (3)

# modify a list (does not return the list)
simpsons.append('lisa')                 # append element to end
simpsons.extend(['itchy', 'scratchy'])  # append multiple elements to end
simpsons.insert(0, 'maggie')            # insert element at index 0 (shifts everything␣
↪right)
simpsons.remove('bart')                 # searches for first instance and removes it
simpsons.pop(0)                         # removes element 0 and returns it
del simpsons[0]                         # removes element 0 (does not return it)
simpsons[0] = 'krusty'                  # replace element 0

# concatenate lists (slower than 'extend' method)
neighbors = simpsons + ['ned','rod','todd']

# find elements in a list
simpsons.count('lisa')      # counts the number of instances
simpsons.index('itchy')     # returns index of first instance

# list slicing [start:end:stride]
weekdays = ['mon','tues','wed','thurs','fri']
weekdays[0]         # element 0
weekdays[0:3]       # elements 0, 1, 2
weekdays[:3]        # elements 0, 1, 2
weekdays[3:]        # elements 3, 4
weekdays[-1]        # last element (element 4)
weekdays[::2]       # every 2nd element (0, 2, 4)
weekdays[::-1]      # backwards (4, 3, 2, 1, 0)

# alternative method for returning the list backwards
list(reversed(weekdays))

# sort a list in place (modifies but does not return the list)
simpsons.sort()
simpsons.sort(reverse=True)     # sort in reverse
simpsons.sort(key=len)          # sort by a key

# return a sorted list (but does not modify the original list)
sorted(simpsons)
sorted(simpsons, reverse=True)
sorted(simpsons, key=len)

# create a second reference to the same list
num = [1, 2, 3]
same_num = num
same_num[0] = 0         # modifies both 'num' and 'same_num'

# copy a list (three ways)
```

(continues on next page)

(continued from previous page)

```
new_num = num.copy()
new_num = num[:]
new_num = list(num)

# examine objects
id(num) == id(same_num) # returns True
id(num) == id(new_num)  # returns False
num is same_num         # returns True
num is new_num          # returns False
num == same_num         # returns True
num == new_num          # returns True (their contents are equivalent)

# conatenate +, replicate *
[1, 2, 3] + [4, 5, 6]
["a"] * 2 + ["b"] * 3
```

2.3.2 Tuples

Like lists, but their size cannot change: ordered, iterable, immutable, can contain multiple data types

```
# create a tuple
digits = (0, 1, 'two')          # create a tuple directly
digits = tuple([0, 1, 'two'])   # create a tuple from a list
zero = (0,)                     # trailing comma is required to indicate it's a tuple

# examine a tuple
digits[2]           # returns 'two'
len(digits)         # returns 3
digits.count(0)     # counts the number of instances of that value (1)
digits.index(1)     # returns the index of the first instance of that value (1)

# elements of a tuple cannot be modified
# digits[2] = 2       # throws an error

# concatenate tuples
digits = digits + (3, 4)

# create a single tuple with elements repeated (also works with lists)
(3, 4) * 2          # returns (3, 4, 3, 4)

# tuple unpacking
bart = ('male', 10, 'simpson')  # create a tuple
```

2.3.3 Strings

A sequence of characters, they are iterable, immutable

```
# create a string
s = str(42)         # convert another data type into a string
s = 'I like you'
```

(continues on next page)

(continued from previous page)

```
# examine a string
s[0]                # returns 'I'
len(s)              # returns 10

# string slicing like lists
s[:6]               # returns 'I like'
s[7:]               # returns 'you'
s[-1]               # returns 'u'

# basic string methods (does not modify the original string)
s.lower()           # returns 'i like you'
s.upper()           # returns 'I LIKE YOU'
s.startswith('I')   # returns True
s.endswith('you')   # returns True
s.isdigit()         # returns False (returns True if every character in the string is a␣
↪digit)
s.find('like')      # returns index of first occurrence (2), but doesn't support regex
s.find('hate')      # returns -1 since not found
s.replace('like','love')    # replaces all instances of 'like' with 'love'

# split a string into a list of substrings separated by a delimiter
s.split(' ')        # returns ['I','like','you']
s.split()           # same thing
s2 = 'a, an, the'
s2.split(',')       # returns ['a',' an',' the']

# join a list of strings into one string using a delimiter
stooges = ['larry','curly','moe']
' '.join(stooges)   # returns 'larry curly moe'

# concatenate strings
s3 = 'The meaning of life is'
s4 = '42'
s3 + ' ' + s4       # returns 'The meaning of life is 42'
s3 + ' ' + str(42)  # same thing

# remove whitespace from start and end of a string
s5 = '  ham and cheese  '
s5.strip()          # returns 'ham and cheese'

# string substitutions: all of these return 'raining cats and dogs'
'raining %s and %s' % ('cats','dogs')                       # old way
'raining {} and {}'.format('cats','dogs')                   # new way
'raining {arg1} and {arg2}'.format(arg1='cats',arg2='dogs') # named arguments

# string formatting
# more examples: http://mkaz.com/2012/10/10/python-string-format/
'pi is {:.2f}'.format(3.14159)      # returns 'pi is 3.14'
```

2.3.4 Strings 2/2

Normal strings allow for escaped characters

```
print('first line\nsecond line')
```

Out:

```
first line
second line
```

raw strings treat backslashes as literal characters

```
print(r'first line\nfirst line')
```

Out:

```
first line\nfirst line
```

sequece of bytes are not strings, should be decoded before some operations

```
s = b'first line\nsecond line'
print(s)

print(s.decode('utf-8').split())
```

Out:

```
b'first line\nsecond line'
['first', 'line', 'second', 'line']
```

2.3.5 Dictionaries

Dictionaries are structures which can contain multiple data types, and is ordered with key-value pairs: for each (unique) key, the dictionary outputs one value. Keys can be strings, numbers, or tuples, while the corresponding values can be any Python object. Dictionaries are: unordered, iterable, mutable

```
# create an empty dictionary (two ways)
empty_dict = {}
empty_dict = dict()

# create a dictionary (two ways)
family = {'dad':'homer', 'mom':'marge', 'size':6}
family = dict(dad='homer', mom='marge', size=6)

# convert a list of tuples into a dictionary
list_of_tuples = [('dad','homer'), ('mom','marge'), ('size', 6)]
family = dict(list_of_tuples)

# examine a dictionary
family['dad']       # returns 'homer'
len(family)         # returns 3
family.keys()       # returns list: ['dad', 'mom', 'size']
family.values()     # returns list: ['homer', 'marge', 6]
family.items()      # returns list of tuples:
                    #   [('dad', 'homer'), ('mom', 'marge'), ('size', 6)]
'mom' in family     # returns True
'marge' in family   # returns False (only checks keys)

# modify a dictionary (does not return the dictionary)
```

(continues on next page)

(continued from previous page)

```
family['cat'] = 'snowball'              # add a new entry
family['cat'] = 'snowball ii'           # edit an existing entry
del family['cat']                       # delete an entry
family['kids'] = ['bart', 'lisa']       # value can be a list
family.pop('dad')                       # removes an entry and returns the value ('homer')
family.update({'baby':'maggie', 'grandpa':'abe'})   # add multiple entries

# accessing values more safely with 'get'
family['mom']                       # returns 'marge'
family.get('mom')                   # same thing
try:
    family['grandma']               # throws an error
except  KeyError as e:
    print("Error", e)

family.get('grandma')               # returns None
family.get('grandma', 'not found')  # returns 'not found' (the default)

# accessing a list element within a dictionary
family['kids'][0]                   # returns 'bart'
family['kids'].remove('lisa')       # removes 'lisa'

# string substitution using a dictionary
'youngest child is %(baby)s' % family   # returns 'youngest child is maggie'
```

Out:

```
Error 'grandma'
```

2.3.6 Sets

Like dictionaries, but with unique keys only (no corresponding values). They are: unordered, iterable, mutable, can contain multiple data types made up of unique elements (strings, numbers, or tuples)

```
# create an empty set
empty_set = set()

# create a set
languages = {'python', 'r', 'java'}         # create a set directly
snakes = set(['cobra', 'viper', 'python'])  # create a set from a list

# examine a set
len(languages)              # returns 3
'python' in languages       # returns True

# set operations
languages & snakes          # returns intersection: {'python'}
languages | snakes          # returns union: {'cobra', 'r', 'java', 'viper', 'python'}
languages - snakes          # returns set difference: {'r', 'java'}
snakes - languages          # returns set difference: {'cobra', 'viper'}

# modify a set (does not return the set)
languages.add('sql')        # add a new element
```

(continues on next page)

(continued from previous page)

```
languages.add('r')           # try to add an existing element (ignored, no error)
languages.remove('java')     # remove an element
try:
    languages.remove('c')        # try to remove a non-existing element (throws an error)
except  KeyError as e:
    print("Error", e)
languages.discard('c')       # removes an element if present, but ignored otherwise
languages.pop()              # removes and returns an arbitrary element
languages.clear()            # removes all elements
languages.update('go', 'spark') # add multiple elements (can also pass a list or set)

# get a sorted list of unique elements from a list
sorted(set([9, 0, 2, 1, 0]))    # returns [0, 1, 2, 9]
```

Out:

```
Error 'c'
```

2.4 Execution control statements

2.4.1 Conditional statements

```
x = 3
# if statement
if x > 0:
    print('positive')

# if/else statement
if x > 0:
    print('positive')
else:
    print('zero or negative')

# if/elif/else statement
if x > 0:
    print('positive')
elif x == 0:
    print('zero')
else:
    print('negative')

# single-line if statement (sometimes discouraged)
if x > 0: print('positive')

# single-line if/else statement (sometimes discouraged)
# known as a 'ternary operator'
'positive' if x > 0 else 'zero or negative'

'positive' if x > 0 else 'zero or negative'
```

Out:

```
positive
positive
positive
positive
```

2.4.2 Loops

Loops are a set of instructions which repeat until termination conditions are met. This can include iterating through all values in an object, go through a range of values, etc

```
# range returns a list of integers
range(0, 3)     # returns [0, 1, 2]: includes first value but excludes second value
range(3)        # same thing: starting at zero is the default
range(0, 5, 2)  # returns [0, 2, 4]: third argument specifies the 'stride'

# for loop
fruits = ['apple', 'banana', 'cherry']
for i in range(len(fruits)):
    print(fruits[i].upper())

# alternative for loop (recommended style)
for fruit in fruits:
    print(fruit.upper())

# use range when iterating over a large sequence to avoid actually creating the integer␣
↪list in memory
v = 0
for i in range(10 ** 6):
    v += 1


quote = """
our incomes are like our shoes; if too small they gall and pinch us
but if too large they cause us to stumble and to trip
"""

count = {k:0 for k in set(quote.split())}
for word in quote.split():
    count[word] += 1


# iterate through two things at once (using tuple unpacking)
family = {'dad':'homer', 'mom':'marge', 'size':6}
for key, value in family.items():
    print(key, value)

# use enumerate if you need to access the index value within the loop
for index, fruit in enumerate(fruits):
    print(index, fruit)

# for/else loop
for fruit in fruits:
    if fruit == 'banana':
        print("Found the banana!")
        break   # exit the loop and skip the 'else' block
```

(continues on next page)

(continued from previous page)

```
    else:
        # this block executes ONLY if the for loop completes without hitting 'break'
        print("Can't find the banana")

# while loop
count = 0
while count < 5:
    print("This will print 5 times")
    count += 1      # equivalent to 'count = count + 1'
```

Out:

```
APPLE
BANANA
CHERRY
APPLE
BANANA
CHERRY
dad homer
mom marge
size 6
0 apple
1 banana
2 cherry
Can't find the banana
Found the banana!
This will print 5 times
This will print 5 times
This will print 5 times
This will print 5 times
This will print 5 times
```

2.4.3 Exceptions handling

```
dct = dict(a=[1, 2], b=[4, 5])

key = 'c'
try:
    dct[key]
except:
    print("Key %s is missing. Add it with empty value" % key)
    dct['c'] = []

print(dct)
```

Out:

```
Key c is missing. Add it with empty value
{'a': [1, 2], 'b': [4, 5], 'c': []}
```

2.5 Functions

Functions are sets of instructions launched when called upon, they can have multiple input values and a return value

```
# define a function with no arguments and no return values
def print_text():
    print('this is text')

# call the function
print_text()

# define a function with one argument and no return values
def print_this(x):
    print(x)

# call the function
print_this(3)       # prints 3
n = print_this(3)   # prints 3, but doesn't assign 3 to n
                    #   because the function has no return statement

#
def add(a, b):
    return a + b

add(2, 3)

add("deux", "trois")

add(["deux", "trois"], [2, 3])

# define a function with one argument and one return value
def square_this(x):
    return x ** 2

# include an optional docstring to describe the effect of a function
def square_this(x):
    """Return the square of a number."""
    return x ** 2

# call the function
square_this(3)          # prints 9
var = square_this(3)    # assigns 9 to var, but does not print 9

# default arguments
def power_this(x, power=2):
    return x ** power

power_this(2)    # 4
power_this(2, 3) # 8

# use 'pass' as a placeholder if you haven't written the function body
def stub():
    pass

# return two values from a single function
```

(continues on next page)

(continued from previous page)

```
def min_max(nums):
    return min(nums), max(nums)

# return values can be assigned to a single variable as a tuple
nums = [1, 2, 3]
min_max_num = min_max(nums)         # min_max_num = (1, 3)

# return values can be assigned into multiple variables using tuple unpacking
min_num, max_num = min_max(nums)    # min_num = 1, max_num = 3
```

Out:

```
this is text
3
3
```

2.6 List comprehensions, iterators, etc.

2.6.1 List comprehensions

Process which affects whole lists without iterating through loops. For more: http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html

```
# for loop to create a list of cubes
nums = [1, 2, 3, 4, 5]
cubes = []
for num in nums:
    cubes.append(num**3)

# equivalent list comprehension
cubes = [num**3 for num in nums]    # [1, 8, 27, 64, 125]

# for loop to create a list of cubes of even numbers
cubes_of_even = []
for num in nums:
    if num % 2 == 0:
        cubes_of_even.append(num**3)

# equivalent list comprehension
# syntax: [expression for variable in iterable if condition]
cubes_of_even = [num**3 for num in nums if num % 2 == 0]    # [8, 64]

# for loop to cube even numbers and square odd numbers
cubes_and_squares = []
for num in nums:
    if num % 2 == 0:
        cubes_and_squares.append(num**3)
    else:
        cubes_and_squares.append(num**2)

# equivalent list comprehension (using a ternary expression)
# syntax: [true_condition if condition else false_condition for variable in iterable]
cubes_and_squares = [num**3 if num % 2 == 0 else num**2 for num in nums]    # [1, 8, 9,␣
↪64, 25]
```

(continues on next page)

(continued from previous page)

```
# for loop to flatten a 2d-matrix
matrix = [[1, 2], [3, 4]]
items = []
for row in matrix:
    for item in row:
        items.append(item)

# equivalent list comprehension
items = [item for row in matrix
              for item in row]      # [1, 2, 3, 4]

# set comprehension
fruits = ['apple', 'banana', 'cherry']
unique_lengths = {len(fruit) for fruit in fruits}   # {5, 6}

# dictionary comprehension
fruit_lengths = {fruit:len(fruit) for fruit in fruits}              # {'apple': 5, 'banana
↪': 6, 'cherry': 6}
```

2.7 Regular expression

1. Compile Regular expression with a patetrn

```
import re

# 1. compile Regular expression with a patetrn
regex = re.compile("^.+(sub-.+)_(ses-.+)_(mod-.+)")
```

2. Match compiled RE on string

Capture the pattern `anyprefixsub-<subj id>_ses-<session id>_<modality>`

```
strings = ["abcsub-033_ses-01_mod-mri", "defsub-044_ses-01_mod-mri", "ghisub-055_ses-02_
↪mod-ctscan" ]
print([regex.findall(s)[0] for s in strings])
```

Out:

```
[('sub-033', 'ses-01', 'mod-mri'), ('sub-044', 'ses-01', 'mod-mri'), ('sub-055', 'ses-02',
↪ 'mod-ctscan')]
```

Match methods on compiled regular expression

Method/Attribute	Purpose
match(string)	Determine if the RE matches at the beginning of the string.
search(string)	Scan through a string, looking for any location where this RE matches.
findall(string)	Find all substrings where the RE matches, and returns them as a list.
finditer(string)	Find all substrings where the RE matches, and returns them as an iterator.

2. Replace compiled RE on string

```
regex = re.compile("(sub-[^_]+)") # match (sub-...)_
print([regex.sub("SUB-", s) for s in strings])

regex.sub("SUB-", "toto")
```

Out:

```
['abcSUB-_ses-01_mod-mri', 'defSUB-_ses-01_mod-mri', 'ghiSUB-_ses-02_mod-ctscan']
```

Replace all non-alphanumeric characters in a string

```
re.sub('[^0-9a-zA-Z]+', '', 'h^&ell`.,|o w]{+orld')
```

2.8 System programming

2.8.1 Operating system interfaces (os)

```
import os
```

Current working directory

```
# Get the current working directory
cwd = os.getcwd()
print(cwd)

# Set the current working directory
os.chdir(cwd)
```

Out:

```
/home/edouard/git/pystatsml/python_lang
```

Temporary directory

```
import tempfile

tmpdir = tempfile.gettempdir()
```

Join paths

```
mytmpdir = os.path.join(tmpdir, "foobar")

# list containing the names of the entries in the directory given by path.
os.listdir(tmpdir)
```

Create a directory

```
if not os.path.exists(mytmpdir):
    os.mkdir(mytmpdir)

os.makedirs(os.path.join(tmpdir, "foobar", "plop", "toto"), exist_ok=True)
```

2.8.2 File input/output

```
filename = os.path.join(mytmpdir, "myfile.txt")
print(filename)

# Write
lines = ["Dans python tout est bon", "Enfin, presque"]

## write line by line
fd = open(filename, "w")
fd.write(lines[0] + "\n")
fd.write(lines[1]+ "\n")
fd.close()

## use a context manager to automatically close your file
with open(filename, 'w') as f:
    for line in lines:
        f.write(line + '\n')

# Read
## read one line at a time (entire file does not have to fit into memory)
f = open(filename, "r")
f.readline()    # one string per line (including newlines)
f.readline()    # next line
f.close()

## read one line at a time (entire file does not have to fit into memory)
f = open(filename, 'r')
f.readline()    # one string per line (including newlines)
f.readline()    # next line
f.close()

## read the whole file at once, return a list of lines
f = open(filename, 'r')
f.readlines()   # one list, each line is one string
f.close()

## use list comprehension to duplicate readlines without reading entire file at once
f = open(filename, 'r')
[line for line in f]
f.close()

## use a context manager to automatically close your file
with open(filename, 'r') as f:
    lines = [line for line in f]
```

Out:

```
/tmp/foobar/myfile.txt
```

2.8.3 Explore, list directories

Walk

```python
import os

WD = os.path.join(tmpdir, "foobar")

for dirpath, dirnames, filenames in os.walk(WD):
    print(dirpath, dirnames, filenames)
```

Out:

```
/tmp/foobar ['plop'] ['myfile.txt']
/tmp/foobar/plop ['toto'] []
/tmp/foobar/plop/toto [] []
```

glob, basename and file extension TODO FIXME

```python
import tempfile
import glob

tmpdir = tempfile.gettempdir()

filenames = glob.glob(os.path.join(tmpdir, "*", "*.txt"))
print(filenames)

# take basename then remove extension
basenames = [os.path.splitext(os.path.basename(f))[0] for f in filenames]
print(basenames)
```

Out:

```
['/tmp/foobar/myfile.txt']
['myfile']
```

shutil - High-level file operations

```python
import shutil

src = os.path.join(tmpdir, "foobar",  "myfile.txt")
dst = os.path.join(tmpdir, "foobar",  "plop", "myfile.txt")
print("copy %s to %s" % (src, dst))

shutil.copy(src, dst)

print("File %s exists ?" % dst, os.path.exists(dst))

src = os.path.join(tmpdir, "foobar",  "plop")
dst = os.path.join(tmpdir, "plop2")
print("copy tree %s under %s" % (src, dst))

try:
    shutil.copytree(src, dst)

    shutil.rmtree(dst)

    shutil.move(src, dst)
except (FileExistsError, FileNotFoundError) as e:
    pass
```

Out:

```
copy /tmp/foobar/myfile.txt to /tmp/foobar/plop/myfile.txt
File /tmp/foobar/plop/myfile.txt exists ? True
copy tree /tmp/foobar/plop under /tmp/plop2
```

2.8.4 Command execution with subprocess

- For more advanced use cases, the underlying Popen interface can be used directly.
- Run the command described by args.
- Wait for command to complete
- return a CompletedProcess instance.
- Does not capture stdout or stderr by default. To do so, pass PIPE for the stdout and/or stderr arguments.

```
import subprocess

# doesn't capture output
p = subprocess.run(["ls", "-l"])
print(p.returncode)

# Run through the shell.
subprocess.run("ls -l", shell=True)

# Capture output
out = subprocess.run(["ls", "-a", "/"], stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
# out.stdout is a sequence of bytes that should be decoded into a utf-8 string
print(out.stdout.decode('utf-8').split("\n")[:5])
```

Out:

```
0
['.', '..', 'bin', 'boot', 'cdrom']
```

2.8.5 Multiprocessing and multithreading

Process

A process is a name given to a program instance that has been loaded into memory and managed by the operating system.

Process = address space + execution context (thread of control)

Process address space (segments):

- Code.
- Data (static/global).
- Heap (dynamic memory allocation).
- Stack.

Execution context:

- Data registers.
- Stack pointer (SP).
- Program counter (PC).
- Working Registers.

OS Scheduling of processes: context switching (ie. save/load Execution context)

Pros/cons

- Context switching expensive.
- (potentially) complex data sharing (not necessary true).
- Cooperating processes - no need for memory protection (separate address spaces).
- Relevant for parrallel computation with memory allocation.

Threads

- Threads share the same address space (Data registers): access to code, heap and (global) data.
- Separate execution stack, PC and Working Registers.

Pros/cons

- Faster context switching only SP, PC and Working Registers.
- Can exploit fine-grain concurrency
- Simple data sharing through the shared address space.
- Precautions have to be taken or two threads will write to the same memory at the same time. This is what the **global interpreter lock (GIL)** is for.
- Relevant for GUI, I/O (Network, disk) concurrent operation

In Python

- The `threading` module uses threads.
- The `multiprocessing` module uses processes.

Multithreading

```python
import time
import threading

def list_append(count, sign=1, out_list=None):
    if out_list is None:
        out_list = list()
    for i in range(count):
        out_list.append(sign * i)
        sum(out_list) # do some computation
    return out_list

size = 10000   # Number of numbers to add

out_list = list() # result is a simple list
```

(continues on next page)

(continued from previous page)

```
thread1 = threading.Thread(target=list_append, args=(size, 1, out_list, ))
thread2 = threading.Thread(target=list_append, args=(size, -1, out_list, ))

startime = time.time()
# Will execute both in parallel
thread1.start()
thread2.start()
# Joins threads back to the parent process
thread1.join()
thread2.join()
print("Threading ellapsed time ", time.time() - startime)

print(out_list[:10])
```

Out:

```
Threading ellapsed time  1.7868659496307373
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Multiprocessing

```
import multiprocessing

# Sharing requires specific mecanism
out_list1 = multiprocessing.Manager().list()
p1 = multiprocessing.Process(target=list_append, args=(size, 1, None))
out_list2 = multiprocessing.Manager().list()
p2 = multiprocessing.Process(target=list_append, args=(size, -1, None))

startime = time.time()
p1.start()
p2.start()
p1.join()
p2.join()
print("Multiprocessing ellapsed time ", time.time() - startime)

# print(out_list[:10]) is not availlable
```

Out:

```
Multiprocessing ellapsed time  0.3927607536315918
```

Sharing object between process with Managers

Managers provide a way to create data which can be shared between different processes, including sharing over a network between processes running on different machines. A manager object controls a server process which manages shared objects.

```
import multiprocessing
import time

size = int(size / 100)   # Number of numbers to add

# Sharing requires specific mecanism
out_list = multiprocessing.Manager().list()
```

(continues on next page)

(continued from previous page)

```
p1 = multiprocessing.Process(target=list_append, args=(size, 1, out_list))
p2 = multiprocessing.Process(target=list_append, args=(size, -1, out_list))

startime = time.time()

p1.start()
p2.start()

p1.join()
p2.join()

print(out_list[:10])

print("Multiprocessing with shared object ellapsed time ", time.time() - startime)
```

Out:

```
[0, 1, 2, 0, 3, -1, 4, -2, 5, -3]
Multiprocessing with shared object ellapsed time  0.7650048732757568
```

2.9 Scripts and argument parsing

Example, the word count script

```
import os
import os.path
import argparse
import re
import pandas as pd

if __name__ == "__main__":
    # parse command line options
    output = "word_count.csv"
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--input',
                        help='list of input files.',
                        nargs='+', type=str)
    parser.add_argument('-o', '--output',
                        help='output csv file (default %s)' % output,
                        type=str, default=output)
    options = parser.parse_args()

    if options.input is None :
        parser.print_help()
        raise SystemExit("Error: input files are missing")
    else:
        filenames = [f for f in options.input if os.path.isfile(f)]

    # Match words
    regex = re.compile("[a-zA-Z]+")

    count = dict()
    for filename in filenames:
```

(continues on next page)

(continued from previous page)

```
        fd = open(filename, "r")
        for line in fd:
            for word in regex.findall(line.lower()):
                if not word in count:
                    count[word] = 1
                else:
                    count[word] += 1

    fd = open(options.output, "w")

    # Pandas
    df = pd.DataFrame([[k, count[k]] for k in count], columns=["word", "count"])
    df.to_csv(options.output, index=False)
```

2.10 Networking

```
# TODO
```

2.10.1 FTP

```
# Full FTP features with ftplib
import ftplib
ftp = ftplib.FTP("ftp.cea.fr")
ftp.login()
ftp.cwd('/pub/unati/people/educhesnay/pystatml')
ftp.retrlines('LIST')

fd = open(os.path.join(tmpdir, "README.md"), "wb")
ftp.retrbinary('RETR README.md', fd.write)
fd.close()
ftp.quit()

# File download urllib
import urllib.request
ftp_url = 'ftp://ftp.cea.fr/pub/unati/people/educhesnay/pystatml/README.md'
urllib.request.urlretrieve(ftp_url, os.path.join(tmpdir, "README2.md"))
```

Out:

```
-rw-r--r--    1 ftp      ftp          3019 Oct 16 00:30 README.md
-rw-r--r--    1 ftp      ftp       9588437 Oct 28 19:58␣
↪StatisticsMachineLearningPythonDraft.pdf
```

2.10.2 HTTP

```
# TODO
```

2.10.3 Sockets

```
# TODO
```

2.10.4 xmlrpc

```
# TODO
```

2.11 Modules and packages

A module is a Python file. A package is a directory which MUST contain a special file called `__init__.py`

To import, extend variable *PYTHONPATH*:

```
export PYTHONPATH=path_to_parent_python_module:${PYTHONPATH}
```

Or

```
import sys
sys.path.append("path_to_parent_python_module")
```

The `__init__.py` file can be empty. But you can set which modules the package exports as the API, while keeping other modules internal, by overriding the __all__ variable, like so:

`parentmodule/__init__.py` file:

```
from . import submodule1
from . import submodule2

from .submodule3 import function1
from .submodule3 import function2

__all__ = ["submodule1", "submodule2",
           "function1", "function2"]
```

User can import:

```
import parentmodule.submodule1
import parentmodule.function1
```

Python Unit Testing

2.12 Object Oriented Programming (OOP)

Sources

- http://python-textbok.readthedocs.org/en/latest/Object_Oriented_Programming.html

Principles

- **Encapsulate** data (attributes) and code (methods) into objects.

- **Class** = template or blueprint that can be used to create objects.
- An **object** is a specific instance of a class.
- **Inheritance**: OOP allows classes to inherit commonly used state and behaviour from other classes. Reduce code duplication
- **Polymorphism**: (usually obtained through polymorphism) calling code is agnostic as to whether an object belongs to a parent class or one of its descendants (abstraction, modularity). The same method called on 2 objects of 2 different classes will behave differently.

```python
import math

class Shape2D:
    def area(self):
        raise NotImplementedError()

# __init__ is a special method called the constructor

# Inheritance + Encapsulation
class Square(Shape2D):
    def __init__(self, width):
        self.width = width

    def area(self):
        return self.width ** 2

class Disk(Shape2D):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

shapes = [Square(2), Disk(3)]

# Polymorphism
print([s.area() for s in shapes])

s = Shape2D()
try:
    s.area()
except NotImplementedError as e:
    print("NotImplementedError")
```

Out:

```
[4, 28.274333882308138]
NotImplementedError
```

2.13 Exercises

2.13.1 Exercise 1: functions

Create a function that acts as a simple calulator If the operation is not specified, default to addition If the operation is misspecified, return an prompt message Ex: `calc(4,5,"multiply")` returns 20 Ex: `calc(3,5)` returns 8 Ex: `calc(1, 2, "something")` returns error message

2.13.2 Exercise 2: functions + list + loop

Given a list of numbers, return a list where all adjacent duplicate elements have been reduced to a single element. Ex: `[1, 2, 2, 3, 2]` returns `[1, 2, 3, 2]`. You may create a new list or modify the passed in list.

Remove all duplicate values (adjacent or not) Ex: `[1, 2, 2, 3, 2]` returns `[1, 2, 3]`

2.13.3 Exercise 3: File I/O

1. Copy/paste the BSD 4 clause license (https://en.wikipedia.org/wiki/BSD_licenses) into a text file. Read, the file and count the occurrences of each word within the file. Store the words' occurrence number in a dictionary.

2. Write an executable python command `count_words.py` that parse a list of input files provided after `--input` parameter. The dictionary of occurrence is save in a csv file provides by `--output`. with default value word_count.csv. Use: - open - regular expression - argparse (https://docs.python.org/3/howto/argparse.html)

2.13.4 Exercise 4: OOP

1. Create a class `Employee` with 2 attributes provided in the constructor: `name`, `years_of_service`. With one method `salary` with is obtained by `1500 + 100 * years_of_service`.
2. Create a subclass `Manager` which redefine `salary` method `2500 + 120 * years_of_service`.
3. Create a small dictionary-nosed database where the key is the employee's name. Populate the database with: samples = Employee('lucy', 3), Employee('john', 1), Manager('julie', 10), Manager('paul', 3)
4. Return a table of made name, salary rows, i.e. a list of list [[name, salary]]
5. Compute the average salary

Total running time of the script: (0 minutes 3.188 seconds)

CHAPTER

THREE

SCIENTIFIC PYTHON

Note: Click *here* to download the full example code

3.1 Numpy: arrays and matrices

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional (numerical) arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

Sources:

- Kevin Markham: https://github.com/justmarkham

```
import numpy as np
```

3.1.1 Create arrays

Create ndarrays from lists. note: every element must be the same type (will be converted if possible)

```
data1 = [1, 2, 3, 4, 5]             # list
arr1 = np.array(data1)              # 1d array
data2 = [range(1, 5), range(5, 9)]  # list of lists
arr2 = np.array(data2)              # 2d array
arr2.tolist()                       # convert array back to list
```

create special arrays

```
np.zeros(10)
np.zeros((3, 6))
np.ones(10)
np.linspace(0, 1, 5)            # 0 to 1 (inclusive) with 5 points
np.logspace(0, 3, 4)            # 10^0 to 10^3 (inclusive) with 4 points
```

arange is like range, except it returns an array (not a list)

```
int_array = np.arange(5)
float_array = int_array.astype(float)
```

3.1.2 Examining arrays

```
arr1.dtype       # float64
arr2.dtype       # int32
arr2.ndim        # 2
arr2.shape       # (2, 4) - axis 0 is rows, axis 1 is columns
arr2.size        # 8 - total number of elements
len(arr2)        # 2 - size of first dimension (aka axis)
```

3.1.3 Reshaping

```
arr = np.arange(10, dtype=float).reshape((2, 5))
print(arr.shape)
print(arr.reshape(5, 2))
```

Out:

```
(2, 5)
[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
```

Add an axis

```
a = np.array([0, 1])
a_col = a[:, np.newaxis]
print(a_col)
#or
a_col = a[:, None]
```

Out:

```
[[0]
 [1]]
```

Transpose

```
print(a_col.T)
```

Out:

```
[[0 1]]
```

Flatten: always returns a flat copy of the orriginal array

```
arr_flt = arr.flatten()
arr_flt[0] = 33
print(arr_flt)
print(arr)
```

Out:

```
[33.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[0. 1. 2. 3. 4.]
 [5. 6. 7. 8. 9.]]
```

Ravel: returns a view of the original array whenever possible.

```
arr_flt = arr.ravel()
arr_flt[0] = 33
print(arr_flt)
print(arr)
```

Out:

```
[33.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[33.  1.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]]
```

3.1.4 Summary on axis, reshaping/flattening and selection

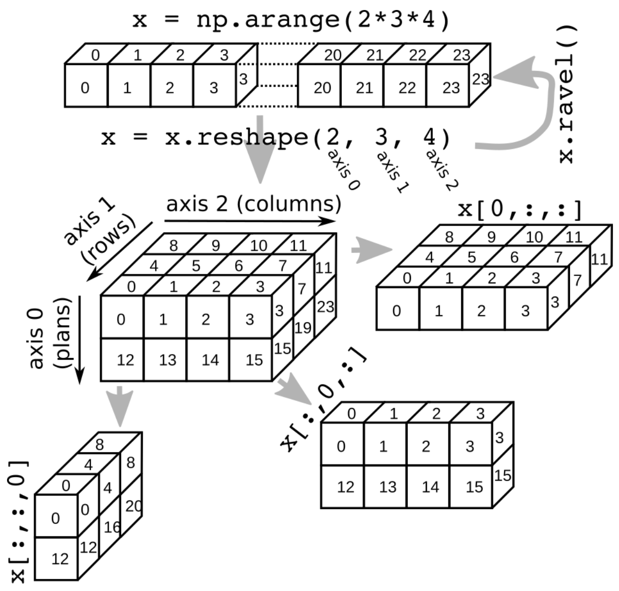
Numpy internals: By default Numpy use C convention, ie, Row-major language: The matrix is stored by rows. In C, the last index changes most rapidly as one moves through the array as stored in memory.

For 2D arrays, sequential move in the memory will:

- **iterate over rows (axis 0)**
    - iterate over columns (axis 1)

For 3D arrays, sequential move in the memory will:

- **iterate over plans (axis 0)**
    - **iterate over rows (axis 1)**
        * iterate over columns (axis 2)



```
x = np.arange(2 * 3 * 4)
print(x)
```

Out:

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

Reshape into 3D (axis 0, axis 1, axis 2)

```
x = x.reshape(2, 3, 4)
print(x)
```

Out:

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Selection get first plan

```
print(x[0, :, :])
```

Out:

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

Selection get first rows

```
print(x[:, 0, :])
```

Out:

```
[[ 0  1  2  3]
 [12 13 14 15]]
```

Selection get first columns

```
print(x[:, :, 0])
```

Out:

```
[[ 0  4  8]
 [12 16 20]]
```

Ravel

```
print(x.ravel())
```

Out:

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

3.1.5 Stack arrays

Stack flat arrays in columns

```
a = np.array([0, 1])
b = np.array([2, 3])

ab = np.stack((a, b)).T
print(ab)

# or
np.hstack((a[:, None], b[:, None]))
```

Out:

```
[[0 2]
 [1 3]]
```

3.1.6 Selection

Single item

```
arr = np.arange(10, dtype=float).reshape((2, 5))

arr[0]         # 0th element (slices like a list)
arr[0, 3]      # row 0, column 3: returns 4
arr[0][3]      # alternative syntax
```

Slicing

Syntax: `start:stop:step` with `start` *(default 0)* `stop` *(default last)* `step` *(default 1)*

```
arr[0, :]      # row 0: returns 1d array ([1, 2, 3, 4])
arr[:, 0]      # column 0: returns 1d array ([1, 5])
arr[:, :2]     # columns strictly before index 2 (2 first columns)
arr[:, 2:]     # columns after index 2 included
arr2 = arr[:, 1:4]    # columns between index 1 (included) and 4 (excluded)
print(arr2)
```

Out:

```
[[1. 2. 3.]
 [6. 7. 8.]]
```

Slicing returns a view (not a copy)

```
arr2[0, 0] = 33
print(arr2)
print(arr)
```

Out:

```
[[33.  2.  3.]
 [ 6.  7.  8.]]
[[ 0. 33.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]]
```

Row 0: reverse order

```
print(arr[0, ::-1])

# The rule of thumb here can be: in the context of lvalue indexing (i.e. the indices are␣
↪placed in the left hand side value of an assignment), no view or copy of the array is␣
↪created (because there is no need to). However, with regular values, the above rules␣
↪for creating views does apply.
```

Out:

```
[ 4.  3.  2. 33.  0.]
```

Fancy indexing: Integer or boolean array indexing

Fancy indexing returns a copy not a view.

Integer array indexing

```
arr2 = arr[:, [1,2,3]]  # return a copy
print(arr2)
arr2[0, 0] = 44
print(arr2)
print(arr)
```

Out:

```
[[33.  2.  3.]
 [ 6.  7.  8.]]
[[44.  2.  3.]
 [ 6.  7.  8.]]
[[ 0. 33.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]]
```

Boolean arrays indexing

```
arr2 = arr[arr > 5]  # return a copy

print(arr2)
arr2[0] = 44
print(arr2)
print(arr)
```

Out:

```
[33.  6.  7.  8.  9.]
[44.  6.  7.  8.  9.]
[[ 0. 33.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]]
```

However, In the context of lvalue indexing (left hand side value of an assignment) Fancy authorizes the modification of the original array

```
arr[arr > 5] = 0
print(arr)
```

Out:

```
[[0. 0. 2. 3. 4.]
 [5. 0. 0. 0. 0.]]
```

Boolean arrays indexing continues

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob'])
names == 'Bob'                          # returns a boolean array
names[names != 'Bob']                   # logical selection
(names == 'Bob') | (names == 'Will')    # keywords "and/or" don't work with boolean arrays
names[names != 'Bob'] = 'Joe'           # assign based on a logical selection
np.unique(names)                        # set function
```

3.1.7 Vectorized operations

```
nums = np.arange(5)
nums * 10                           # multiply each element by 10
nums = np.sqrt(nums)                # square root of each element
np.ceil(nums)                       # also floor, rint (round to nearest int)
np.isnan(nums)                      # checks for NaN
nums + np.arange(5)                 # add element-wise
np.maximum(nums, np.array([1, -2, 3, -4, 5]))  # compare element-wise

# Compute Euclidean distance between 2 vectors
vec1 = np.random.randn(10)
vec2 = np.random.randn(10)
dist = np.sqrt(np.sum((vec1 - vec2) ** 2))

# math and stats
rnd = np.random.randn(4, 2) # random normals in 4x2 array
rnd.mean()
rnd.std()
rnd.argmin()                # index of minimum element
rnd.sum()
rnd.sum(axis=0)             # sum of columns
rnd.sum(axis=1)             # sum of rows

# methods for boolean arrays
(rnd > 0).sum()             # counts number of positive values
(rnd > 0).any()             # checks if any value is True
(rnd > 0).all()             # checks if all values are True

# random numbers
np.random.seed(12234)       # Set the seed
np.random.rand(2, 3)        # 2 x 3 matrix in [0, 1]
np.random.randn(10)         # random normals (mean 0, sd 1)
np.random.randint(0, 2, 10) # 10 randomly picked 0 or 1
```

3.1.8 Broadcasting

Sources: https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html Implicit conversion to allow operations on arrays of different sizes. - The smaller array is stretched or "broadcasted" across the larger array so that they have compatible shapes. - Fast vectorized operation in C instead of Python. - No needless copies.

Rules

Starting with the trailing axis and working backward, Numpy compares arrays dimensions.

- If two dimensions are equal then continues
- If one of the operand has dimension 1 stretches it to match the largest one
- When one of the shapes runs out of dimensions (because it has less dimensions than the other shape), Numpy will use 1 in the comparison process until the other shape's dimensions run out as well.

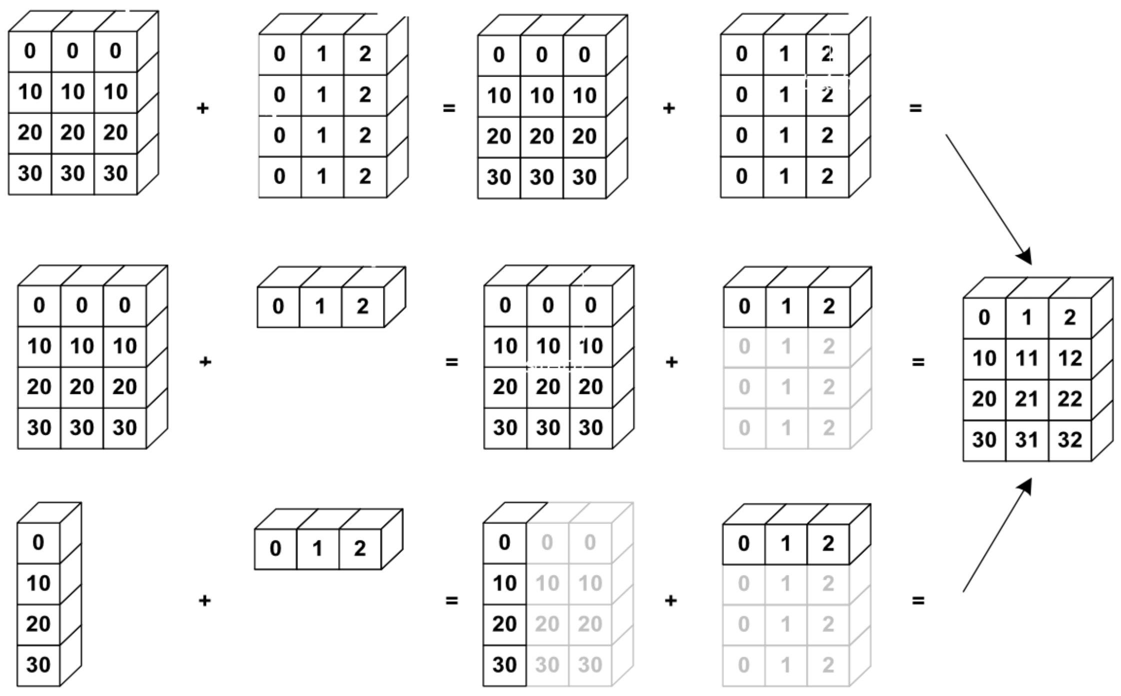


Fig. 1: Source: http://www.scipy-lectures.org

```
a = np.array([[ 0,  0,  0],
              [10, 10, 10],
              [20, 20, 20],
              [30, 30, 30]])

b = np.array([0, 1, 2])
```

(continues on next page)

(continued from previous page)

```
print(a + b)
```

Out:

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

Examples

Shapes of operands A, B and result:

```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 1
Result (3d array):  15 x 3 x 5
```

3.1.9 Exercises

Given the array:

```
X = np.random.randn(4, 2) # random normals in 4x2 array
```

- For each column find the row index of the minimum value.
- Write a function `standardize(X)` that return an array whose columns are centered and scaled (by std-dev).

Total running time of the script: (0 minutes 0.039 seconds)

Note: Click *here* to download the full example code

3.2 Pandas: data manipulation

It is often said that 80% of data analysis is spent on the cleaning and small, but important, aspect of data manipulation and cleaning with Pandas.

Sources:

- Kevin Markham: https://github.com/justmarkham
- Pandas doc: http://pandas.pydata.org/pandas-docs/stable/index.html

Data structures

- **Series** is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call *pd.Series([1,3,5,np.nan,6,8])*
- **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It stems from the *R data.frame()* object.

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

3.2.1 Create DataFrame

```
columns = ['name', 'age', 'gender', 'job']

user1 = pd.DataFrame([['alice', 19, "F", "student"],
                      ['john', 26, "M", "student"]],
    columns=columns)

user2 = pd.DataFrame([['eric', 22, "M", "student"],
                      ['paul', 58, "F", "manager"]],
    columns=columns)

user3 = pd.DataFrame(dict(name=['peter', 'julie'],
                          age=[33, 44], gender=['M', 'F'],
                          job=['engineer', 'scientist']))

print(user3)
```

Out:

```
    name  age gender        job
0  peter   33      M   engineer
1  julie   44      F  scientist
```

3.2.2 Combining DataFrames

Concatenate DataFrame

```
user1.append(user2)
users = pd.concat([user1, user2, user3])
print(users)
```

Out:

```
    name  age gender        job
0  alice   19      F    student
1   john   26      M    student
0   eric   22      M    student
1   paul   58      F    manager
0  peter   33      M   engineer
1  julie   44      F  scientist
```

Join DataFrame

```
user4 = pd.DataFrame(dict(name=['alice', 'john', 'eric', 'julie'],
                          height=[165, 180, 175, 171]))
print(user4)
```

Out:

```
    name  height
0  alice     165
1   john     180
2   eric     175
3  julie     171
```

Use intersection of keys from both frames

```
merge_inter = pd.merge(users, user4, on="name")

print(merge_inter)
```

Out:

```
    name  age gender        job  height
0  alice   19      F    student     165
1   john   26      M    student     180
2   eric   22      M    student     175
3  julie   44      F  scientist     171
```

Use union of keys from both frames

```
users = pd.merge(users, user4, on="name", how='outer')
print(users)
```

Out:

```
    name  age gender        job  height
0  alice   19      F    student   165.0
1   john   26      M    student   180.0
```

(continues on next page)

(continued from previous page)

```
2   eric   22      M    student   175.0
3   paul   58      F    manager     NaN
4  peter   33      M   engineer     NaN
5  julie   44      F  scientist   171.0
```

Reshaping by pivoting

“Unpivots” a DataFrame from wide format to long (stacked) format,

```
staked = pd.melt(users, id_vars="name", var_name="variable", value_name="value")
print(staked)
```

Out:

```
     name variable      value
0   alice      age         19
1    john      age         26
2    eric      age         22
3    paul      age         58
4   peter      age         33
5   julie      age         44
6   alice   gender          F
7    john   gender          M
8    eric   gender          M
9    paul   gender          F
10  peter   gender          M
11  julie   gender          F
12  alice      job    student
13   john      job    student
14   eric      job    student
15   paul      job    manager
16  peter      job   engineer
17  julie      job  scientist
18  alice   height        165
19   john   height        180
20   eric   height        175
21   paul   height        NaN
22  peter   height        NaN
23  julie   height        171
```

“pivots” a DataFrame from long (stacked) format to wide format,

```
print(staked.pivot(index='name', columns='variable', values='value'))
```

Out:

```
variable age gender height        job
name
alice     19      F    165    student
eric      22      M    175    student
john      26      M    180    student
julie     44      F    171  scientist
paul      58      F    NaN    manager
peter     33      M    NaN   engineer
```

3.2.3 Summarizing

```
# examine the users data

users                   # print the first 30 and last 30 rows
type(users)             # DataFrame
users.head()            # print the first 5 rows
users.tail()            # print the last 5 rows


users.index             # "the index" (aka "the labels")
users.columns           # column names (which is "an index")
users.dtypes            # data types of each column
users.shape             # number of rows and columns
users.values            # underlying numpy array
users.info()            # concise summary (includes memory usage as of pandas 0.15.0)
```

Out:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 5
Data columns (total 5 columns):
name      6 non-null object
age       6 non-null int64
gender    6 non-null object
job       6 non-null object
height    4 non-null float64
dtypes: float64(1), int64(1), object(3)
memory usage: 288.0+ bytes
```

3.2.4 Columns selection

```
users['gender']         # select one column
type(users['gender'])   # Series
users.gender            # select one column using the DataFrame

# select multiple columns
users[['age', 'gender']]        # select two columns
my_cols = ['age', 'gender']     # or, create a list...
users[my_cols]                  # ...and use that list to select columns
type(users[my_cols])            # DataFrame
```

3.2.5 Rows selection (basic)

iloc is strictly integer position based

```
df = users.copy()
df.iloc[0]     # first row
df.iloc[0, 0]  # first item of first row
df.iloc[0, 0] = 55

for i in range(users.shape[0]):
    row = df.iloc[i]
```

(continues on next page)

(continued from previous page)

```
    row.age *= 100 # setting a copy, and not the original frame data.

print(df)  # df is not modified
```

Out:

```
/home/edouard/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py:5096:␣
↪SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/
↪indexing.html#indexing-view-versus-copy
  self[name] = value
    name  age gender        job  height
0     55   19      F    student   165.0
1   john   26      M    student   180.0
2   eric   22      M    student   175.0
3   paul   58      F    manager     NaN
4  peter   33      M   engineer     NaN
5  julie   44      F  scientist   171.0
```

ix supports mixed integer and label based access.

```
df = users.copy()
df.loc[0]         # first row
df.loc[0, "age"]  # first item of first row
df.loc[0, "age"] = 55

for i in range(df.shape[0]):
    df.loc[i, "age"] *= 10

print(df)  # df is modified
```

Out:

```
    name  age gender        job  height
0  alice  550      F    student   165.0
1   john  260      M    student   180.0
2   eric  220      M    student   175.0
3   paul  580      F    manager     NaN
4  peter  330      M   engineer     NaN
5  julie  440      F  scientist   171.0
```

3.2.6 Rows selection (filtering)

simple logical filtering

```
users[users.age < 20]        # only show users with age < 20
young_bool = users.age < 20  # or, create a Series of booleans...
young = users[young_bool]            # ...and use that Series to filter rows
users[users.age < 20].job    # select one column from the filtered results
print(young)
```

Out:

```
    name  age gender      job  height
0  alice   19      F  student   165.0
```

Advanced logical filtering

```
users[users.age < 20][['age', 'job']]           # select multiple columns
users[(users.age > 20) & (users.gender == 'M')]   # use multiple conditions
users[users.job.isin(['student', 'engineer'])]  # filter specific values
```

3.2.7 Sorting

```
df = users.copy()

df.age.sort_values()                          # only works for a Series
df.sort_values(by='age')                      # sort rows by a specific column
df.sort_values(by='age', ascending=False) # use descending order instead
df.sort_values(by=['job', 'age'])         # sort by multiple columns
df.sort_values(by=['job', 'age'], inplace=True) # modify df

print(df)
```

Out:

```
    name  age gender        job  height
4  peter   33      M   engineer     NaN
3   paul   58      F    manager     NaN
5  julie   44      F  scientist   171.0
0  alice   19      F    student   165.0
2   eric   22      M    student   175.0
1   john   26      M    student   180.0
```

3.2.8 Descriptive statistics

Summarize all numeric columns

```
print(df.describe())
```

Out:

```
             age      height
count   6.000000    4.000000
mean   33.666667  172.750000
std    14.895189    6.344289
min    19.000000  165.000000
25%    23.000000  169.500000
50%    29.500000  173.000000
75%    41.250000  176.250000
max    58.000000  180.000000
```

Summarize all columns

```
print(df.describe(include='all'))
print(df.describe(include=['object']))  # limit to one (or more) types
```

Out:

```
        name        age gender      job      height
count      6   6.000000      6        6    4.000000
unique     6        NaN      2        4         NaN
top     eric        NaN      M  student         NaN
freq       1        NaN      3        3         NaN
mean     NaN  33.666667    NaN      NaN  172.750000
std      NaN  14.895189    NaN      NaN    6.344289
min      NaN  19.000000    NaN      NaN  165.000000
25%      NaN  23.000000    NaN      NaN  169.500000
50%      NaN  29.500000    NaN      NaN  173.000000
75%      NaN  41.250000    NaN      NaN  176.250000
max      NaN  58.000000    NaN      NaN  180.000000
        name gender      job
count      6      6        6
unique     6      2        4
top     eric      M  student
freq       1      3        3
```

Statistics per group (groupby)

```python
print(df.groupby("job").mean())

print(df.groupby("job")["age"].mean())

print(df.groupby("job").describe(include='all'))
```

Out:

```
                 age      height
job
engineer   33.000000         NaN
manager    58.000000         NaN
scientist  44.000000  171.000000
student    22.333333  173.333333
job
engineer     33.000000
manager      58.000000
scientist    44.000000
student      22.333333
Name: age, dtype: float64
           name                        ... height
          count unique    top freq mean ...    min    25%    50%    75%    max
job                                     ...
engineer      1      1  peter    1  NaN ...    NaN    NaN    NaN    NaN    NaN
manager       1      1   paul    1  NaN ...    NaN    NaN    NaN    NaN    NaN
scientist     1      1  julie    1  NaN ...  171.0  171.0  171.0  171.0  171.0
student       3      3   eric    1  NaN ...  165.0  170.0  175.0  177.5  180.0

[4 rows x 44 columns]
```

Groupby in a loop

```python
for grp, data in df.groupby("job"):
    print(grp, data)
```

Out:

```
engineer     name  age gender       job  height
4  peter   33      M  engineer     NaN
manager    name  age gender      job  height
3  paul   58      F  manager     NaN
scientist     name  age gender        job  height
5  julie   44      F  scientist   171.0
student      name  age gender      job  height
0  alice   19      F  student   165.0
2   eric   22      M  student   175.0
1   john   26      M  student   180.0
```

3.2.9 Quality check

Remove duplicate data

```
df = users.append(df.iloc[0], ignore_index=True)

print(df.duplicated())                 # Series of booleans
# (True if a row is identical to a previous row)
df.duplicated().sum()                  # count of duplicates
df[df.duplicated()]                    # only show duplicates
df.age.duplicated()                    # check a single column for duplicates
df.duplicated(['age', 'gender']).sum() # specify columns for finding duplicates
df = df.drop_duplicates()              # drop duplicate rows
```

Out:

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

Missing data

```
# Missing values are often just excluded
df = users.copy()

df.describe(include='all')              # excludes missing values

# find missing values in a Series
df.height.isnull()           # True if NaN, False otherwise
df.height.notnull()          # False if NaN, True otherwise
df[df.height.notnull()]      # only show rows where age is not NaN
df.height.isnull().sum()     # count the missing values

# find missing values in a DataFrame
df.isnull()             # DataFrame of booleans
df.isnull().sum()       # calculate the sum of each column
```

Strategy 1: drop missing values

```
df.dropna()             # drop a row if ANY values are missing
df.dropna(how='all')    # drop a row only if ALL values are missing
```

Strategy 2: fill in missing values

```
df.height.mean()
df = users.copy()
df.loc[df.height.isnull(), "height"] = df["height"].mean()

print(df)
```

Out:

```
    name  age gender        job  height
0  alice   19      F    student  165.00
1   john   26      M    student  180.00
2   eric   22      M    student  175.00
3   paul   58      F    manager  172.75
4  peter   33      M   engineer  172.75
5  julie   44      F  scientist  171.00
```

3.2.10 Rename values

```
df = users.copy()
print(df.columns)
df.columns = ['age', 'genre', 'travail', 'nom', 'taille']

df.travail = df.travail.map({ 'student':'etudiant',  'manager':'manager',
                'engineer':'ingenieur', 'scientist':'scientific'})
# assert df.travail.isnull().sum() == 0



df['travail'].str.contains("etu|inge")
```

Out:

```
Index(['name', 'age', 'gender', 'job', 'height'], dtype='object')
```

3.2.11 Dealing with outliers

```
size = pd.Series(np.random.normal(loc=175, size=20, scale=10))
# Corrupt the first 3 measures
size[:3] += 500
```

Based on parametric statistics: use the mean

Assume random variable follows the normal distribution Exclude data outside 3 standard-deviations: - Probability that a sample lies within 1 sd: 68.27% - Probability that a sample lies within 3 sd: 99.73% (68.27 + 2 * 15.73)

```
size_outlr_mean = size.copy()
size_outlr_mean[((size - size.mean()).abs() > 3 * size.std())] = size.mean()
print(size_outlr_mean.mean())
```

Out:

```
248.48963819938044
```

Based on non-parametric statistics: use the median

Median absolute deviation (MAD), based on the median, is a robust non-parametric statistics. https://en.wikipedia.org/wiki/Median_absolute_deviation

```
mad = 1.4826 * np.median(np.abs(size - size.median()))
size_outlr_mad = size.copy()

size_outlr_mad[((size - size.median()).abs() > 3 * mad)] = size.median()
print(size_outlr_mad.mean(), size_outlr_mad.median())
```

Out:

```
173.80000467192673 178.7023568870694
```

3.2.12 File I/O

csv

```
import tempfile, os.path
tmpdir = tempfile.gettempdir()
csv_filename = os.path.join(tmpdir, "users.csv")
users.to_csv(csv_filename, index=False)
other = pd.read_csv(csv_filename)
```

Read csv from url

```
url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
salary = pd.read_csv(url)
```

Excel

```
xls_filename = os.path.join(tmpdir, "users.xlsx")
users.to_excel(xls_filename, sheet_name='users', index=False)

pd.read_excel(xls_filename, sheet_name='users')

# Multiple sheets
with pd.ExcelWriter(xls_filename) as writer:
    users.to_excel(writer, sheet_name='users', index=False)
```

(continues on next page)

(continued from previous page)

```
    df.to_excel(writer, sheet_name='salary', index=False)

pd.read_excel(xls_filename, sheet_name='users')
pd.read_excel(xls_filename, sheet_name='salary')
```

SQL (SQLite)

```
import pandas as pd
import sqlite3

db_filename = os.path.join(tmpdir, "users.db")
```

Connect

```
conn = sqlite3.connect(db_filename)
```

Creating tables with pandas

```
url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
salary = pd.read_csv(url)

salary.to_sql("salary", conn, if_exists="replace")
```

Push modifications

```
cur = conn.cursor()
values = (100, 14000, 5,  'Bachelor', 'N')
cur.execute("insert into salary values (?, ?, ?, ?, ?)", values)
conn.commit()
```

Reading results into a pandas DataFrame

```
salary_sql = pd.read_sql_query("select * from salary;", conn)
print(salary_sql.head())

pd.read_sql_query("select * from salary;", conn).tail()
pd.read_sql_query('select * from salary where salary>25000;', conn)
pd.read_sql_query('select * from salary where experience=16;', conn)
pd.read_sql_query('select * from salary where education="Master";', conn)
```

Out:

```
   index  salary  experience education management
0      0   13876           1  Bachelor          Y
1      1   11608           1      Ph.D          N
2      2   18701           1      Ph.D          Y
3      3   11283           1    Master          N
4      4   11767           1      Ph.D          N
```

3.2.13 Exercises

Data Frame

1. Read the iris dataset at 'https://github.com/neurospin/pystatsml/tree/master/datasets/iris.csv'
2. Print column names
3. Get numerical columns
4. For each species compute the mean of numerical columns and store it in a `stats` table like:

```
      species  sepal_length  sepal_width  petal_length  petal_width
0      setosa         5.006        3.428         1.462        0.246
1  versicolor         5.936        2.770         4.260        1.326
2   virginica         6.588        2.974         5.552        2.026
```

Missing data

Add some missing data to the previous table `users`:

```
df = users.copy()
df.ix[[0, 2], "age"] = None
df.ix[[1, 3], "gender"] = None
```

Out:

```
/home/edouard/git/pystatsml/scientific_python/scipy_pandas.py:440: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  df.ix[[0, 2], "age"] = None
/home/edouard/git/pystatsml/scientific_python/scipy_pandas.py:441: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  df.ix[[1, 3], "gender"] = None
```

1. Write a function `fillmissing_with_mean(df)` that fill all missing value of numerical column with the mean of the current columns.
2. Save the original users and "imputed" frame in a single excel file "users.xlsx" with 2 sheets: original, imputed.

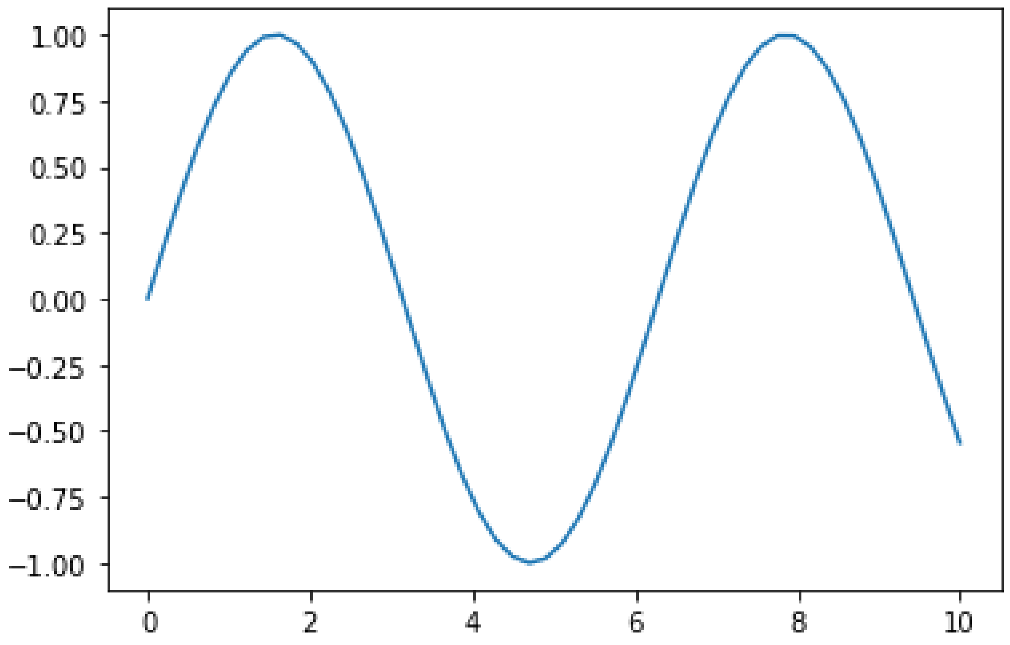
Total running time of the script: (0 minutes 1.488 seconds)

3.3 Matplotlib: data visualization

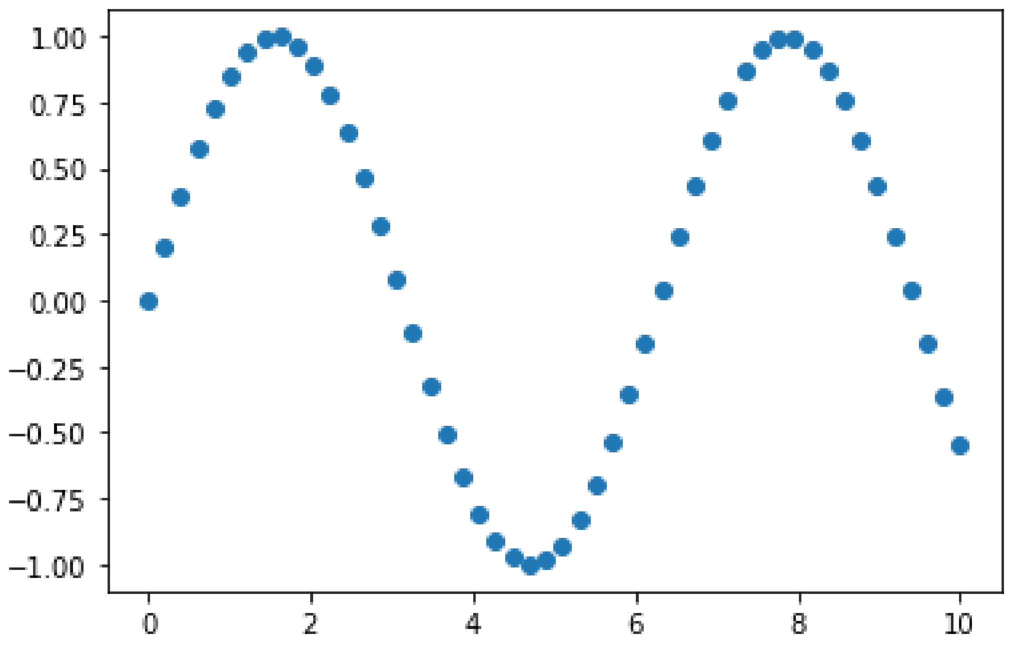
Sources - Nicolas P. Rougier: http://www.labri.fr/perso/nrougier/teaching/matplotlib - https://www.kaggle.com/benhamner/d/uciml/iris/python-data-visualizations

3.3.1 Basic plots

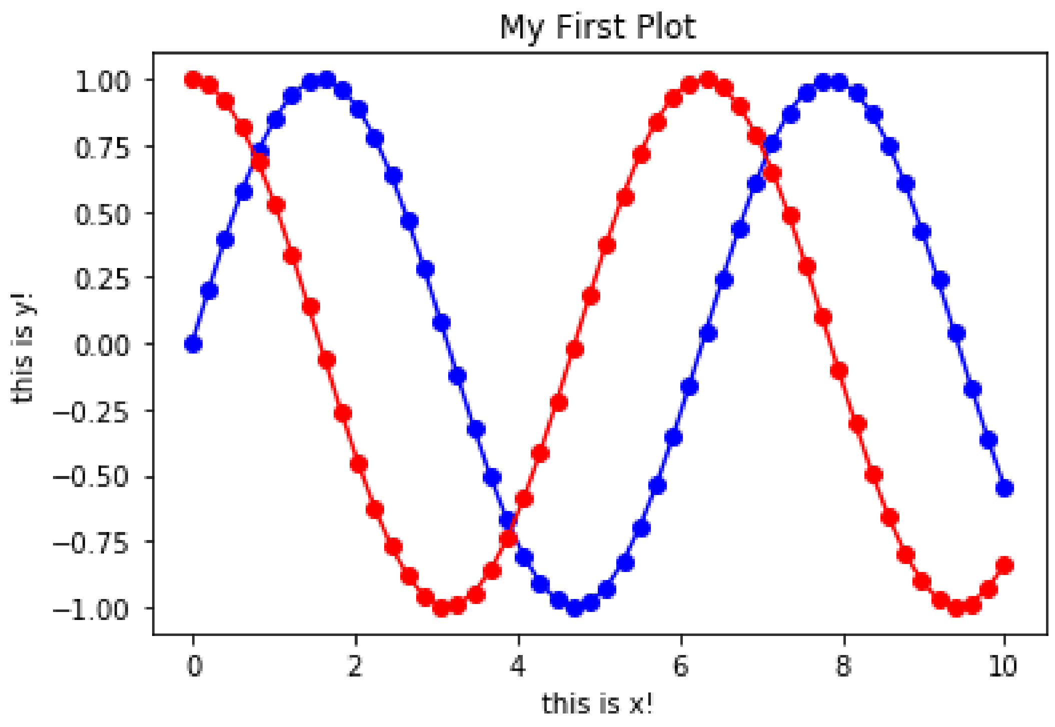
```
import numpy as np
import matplotlib.pyplot as plt

# inline plot (for jupyter)
%matplotlib inline

x = np.linspace(0, 10, 50)
sinus = np.sin(x)

plt.plot(x, sinus)
plt.show()
```



```
plt.plot(x, sinus, "o")
plt.show()
# use plt.plot to get color / marker abbreviations
```



```
# Rapid multiplot

cosinus = np.cos(x)
plt.plot(x, sinus, "-b", x, sinus, "ob", x, cosinus, "-r", x, cosinus, "or")
plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot')
plt.show()
```

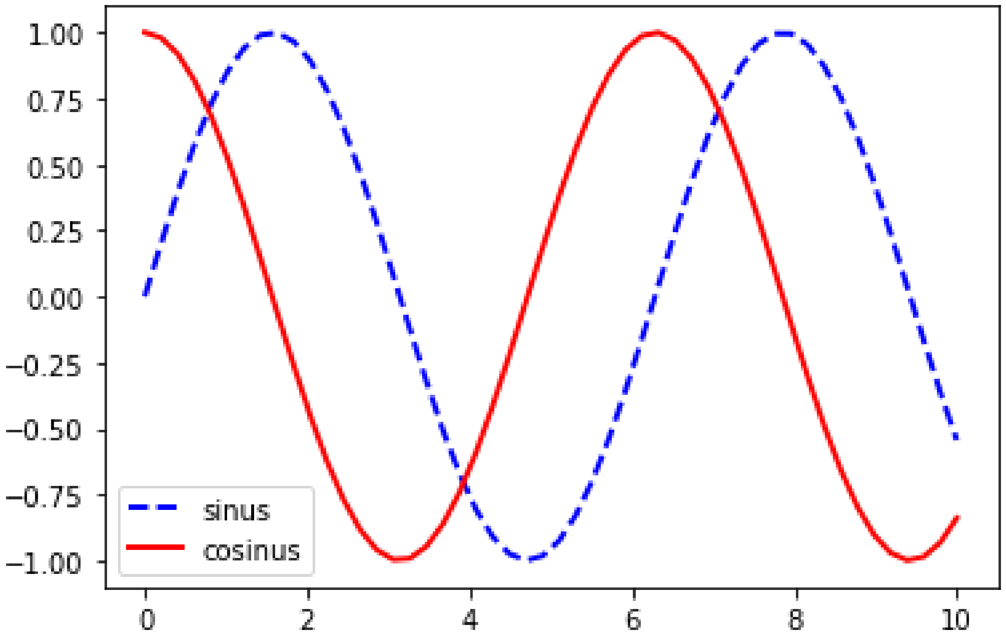


```
# Step by step
plt.plot(x, sinus, label='sinus', color='blue', linestyle='--', linewidth=2)
```

(continues on next page)

(continued from previous page)

```
plt.plot(x, cosinus, label='cosinus', color='red', linestyle='-', linewidth=2)
plt.legend()
plt.show()
```



3.3.2 Scatter (2D) plots

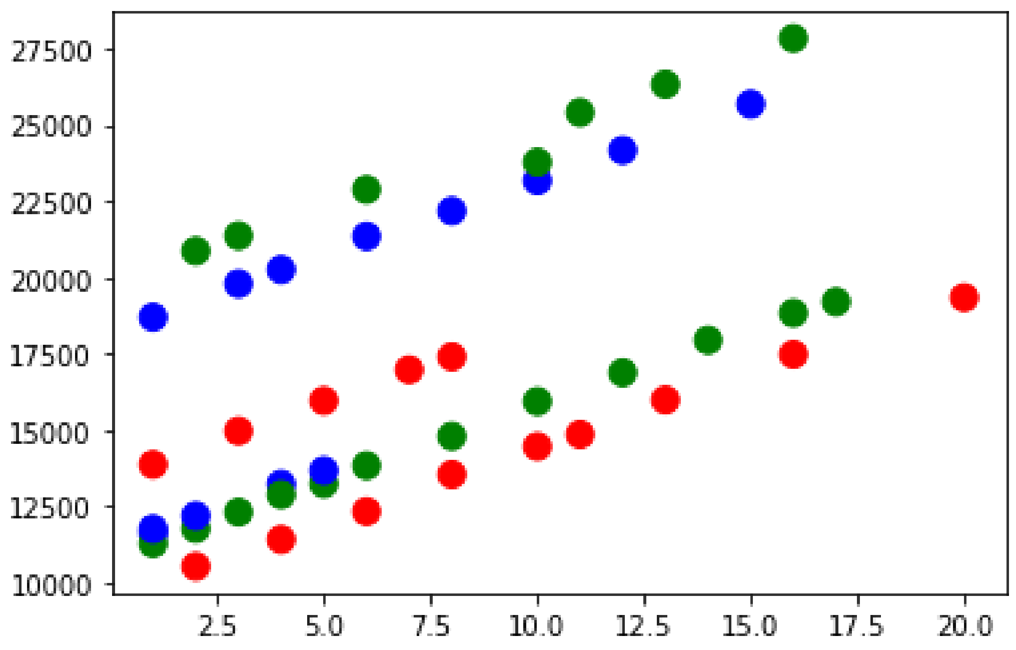
Load dataset

```
import pandas as pd
try:
    salary = pd.read_csv("../datasets/salary_table.csv")
except:
    url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
    salary = pd.read_csv(url)

df = salary
```

Simple scatter with colors

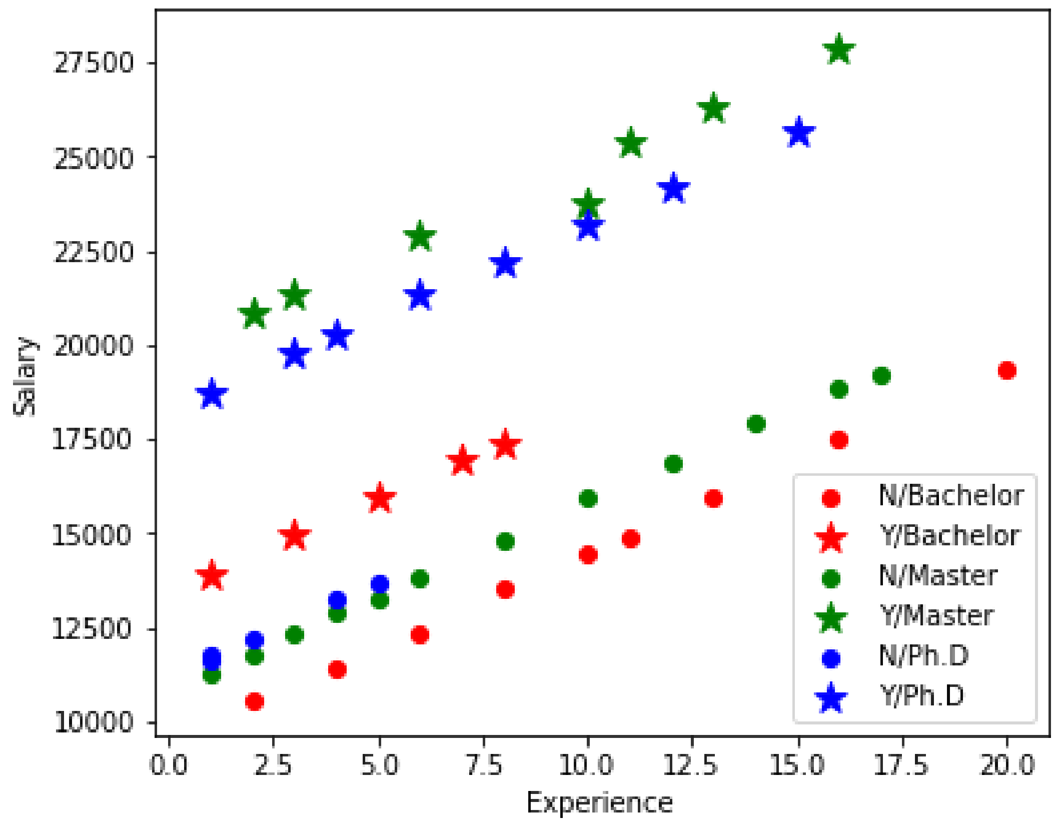
```
colors = colors_edu = {'Bachelor':'r', 'Master':'g', 'Ph.D':'blue'}
plt.scatter(df['experience'], df['salary'], c=df['education'].apply(lambda x: colors[x]),␣
↪s=100)
```

```
<matplotlib.collections.PathCollection at 0x7f39efac6358>
```



Scatter plot with colors and symbols

```
## Figure size
plt.figure(figsize=(6,5))

## Define colors / symbols manually
symbols_manag = dict(Y='*', N='.')
colors_edu = {'Bachelor':'r', 'Master':'g', 'Ph.D':'b'}

## group by education x management => 6 groups
for values, d in salary.groupby(['education','management']):
    edu, manager = values
    plt.scatter(d['experience'], d['salary'], marker=symbols_manag[manager], color=colors_
↪edu[edu],
               s=150, label=manager+"/"+edu)

## Set labels
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.legend(loc=4)  # lower right
plt.show()
```



3.3.3 Saving Figures

```
### bitmap format
plt.plot(x, sinus)
plt.savefig("sinus.png")
plt.close()

# Prefer vectorial format (SVG: Scalable Vector Graphics) can be edited with
# Inkscape, Adobe Illustrator, Blender, etc.
plt.plot(x, sinus)
plt.savefig("sinus.svg")
plt.close()

# Or pdf
plt.plot(x, sinus)
plt.savefig("sinus.pdf")
plt.close()
```

3.3.4 Seaborn

Sources: - http://stanford.edu/~mwaskom/software/seaborn - https://elitedatascience.com/python-seaborn-tutorial

If needed, install using: `pip install -U --user seaborn`

Boxplot

Box plots are non-parametric: they display variation in samples of a statistical population without making any assumptions of the underlying statistical distribution.

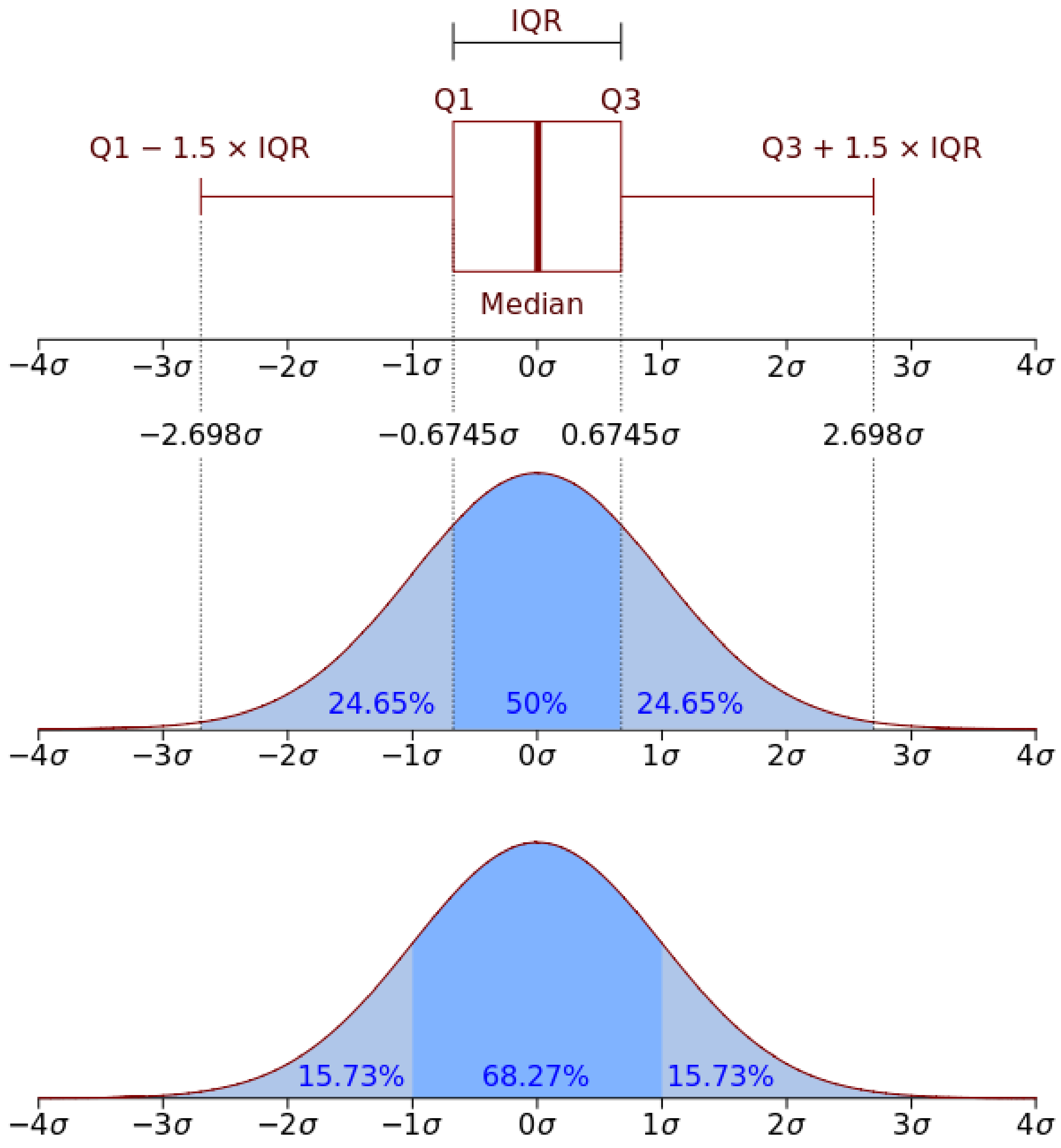
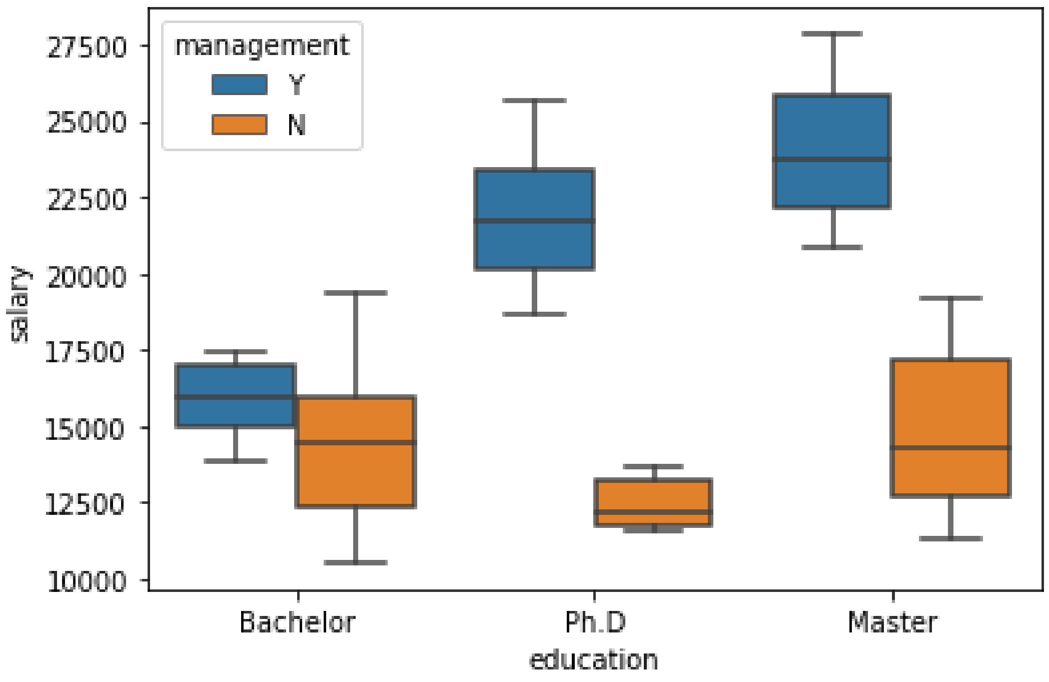


Fig. 2: title

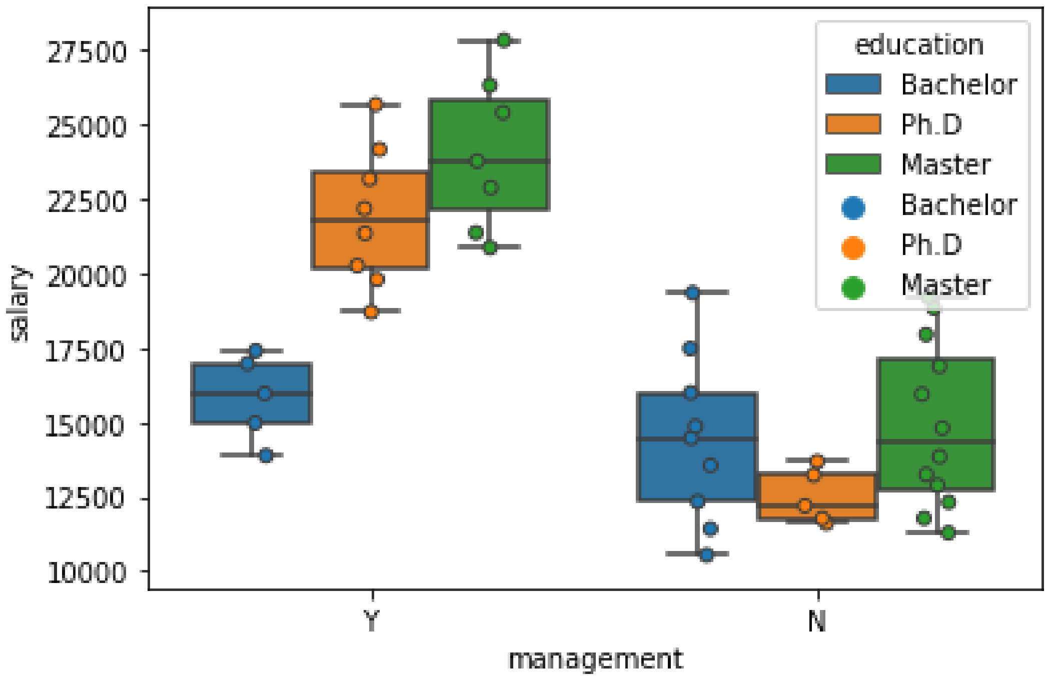
```python
import seaborn as sns

sns.boxplot(x="education", y="salary", hue="management", data=salary)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39ed42ff28>
```



```
sns.boxplot(x="management", y="salary", hue="education", data=salary)
sns.stripplot(x="management", y="salary", hue="education", data=salary, jitter=True,␣
↪dodge=True, linewidth=1)# Jitter and split options separate datapoints according to␣
↪group"
```

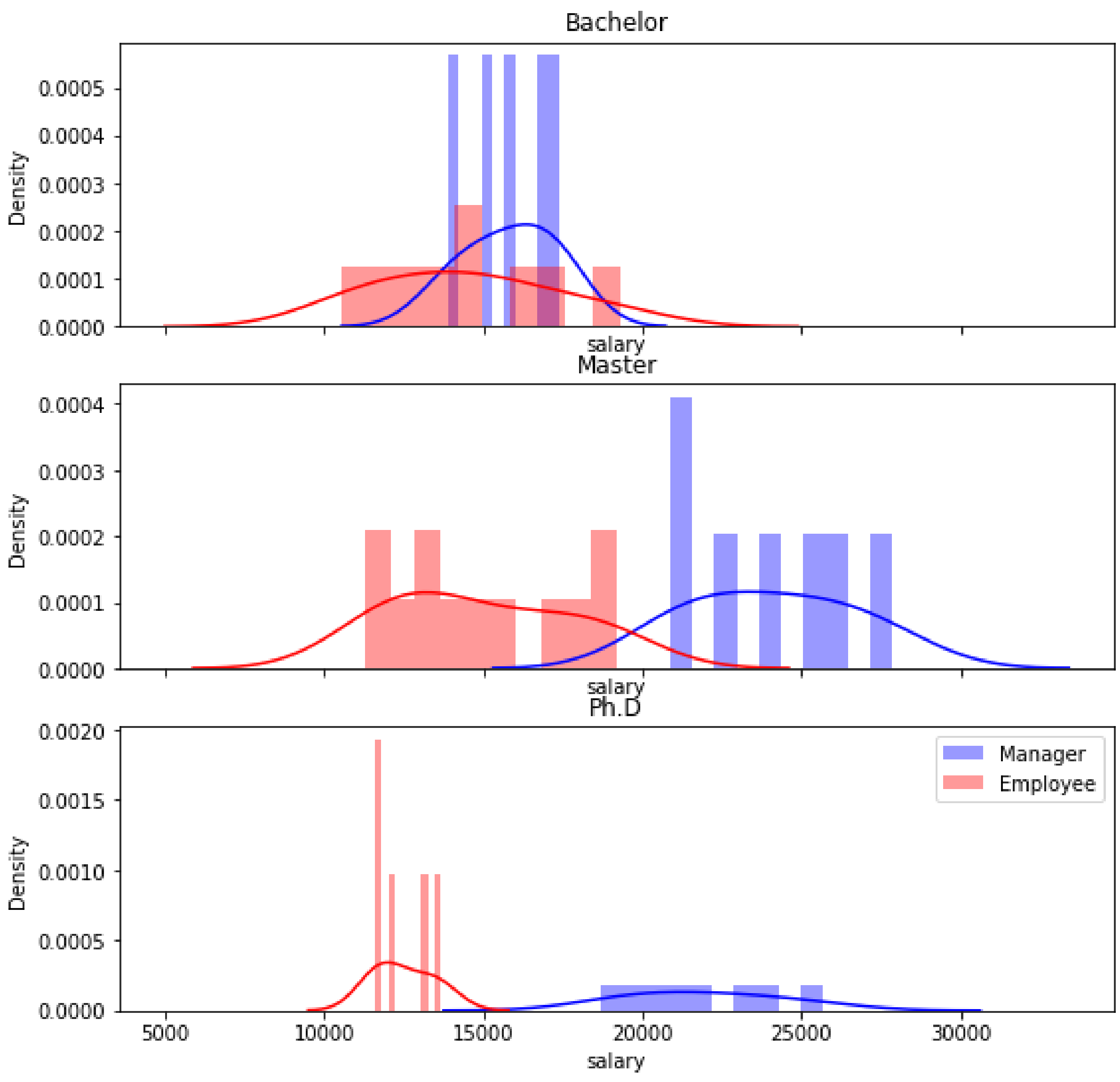
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39eb61d780>
```



### Density plot with one figure containing multiple axis

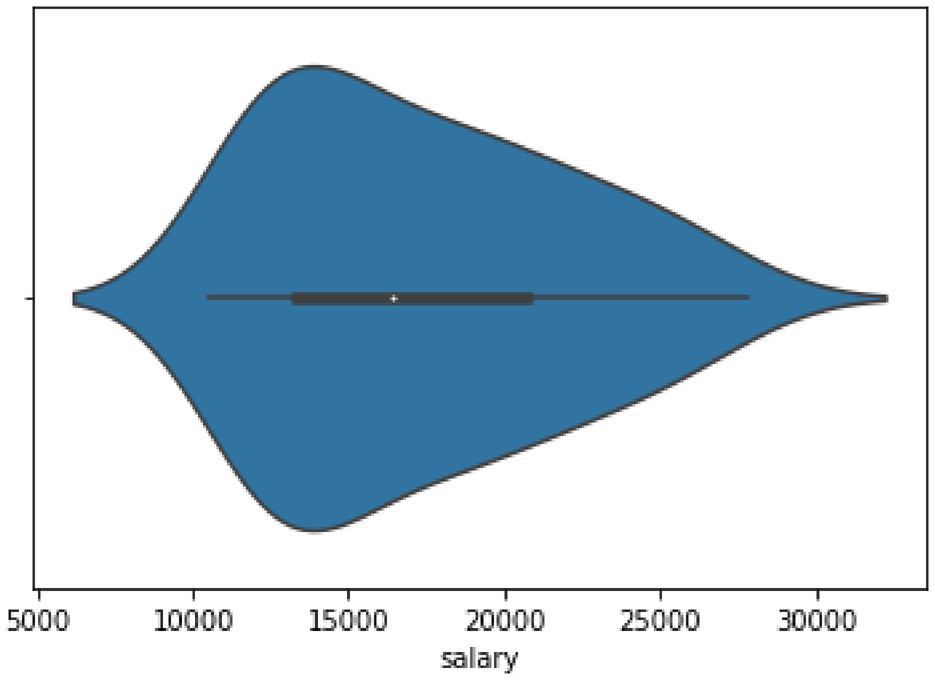
One figure can contain several axis, whose contain the graphic elements

```
# Set up the matplotlib figure: 3 x 1 axis

f, axes = plt.subplots(3, 1, figsize=(9, 9), sharex=True)

i = 0
for edu, d in salary.groupby(['education']):
    sns.distplot(d.salary[d.management == "Y"], color="b", bins=10, label="Manager",␣
↪ax=axes[i])
    sns.distplot(d.salary[d.management == "N"], color="r", bins=10, label="Employee",␣
↪ax=axes[i])
    axes[i].set_title(edu)
    axes[i].set_ylabel('Density')
    i += 1
ax = plt.legend()
```



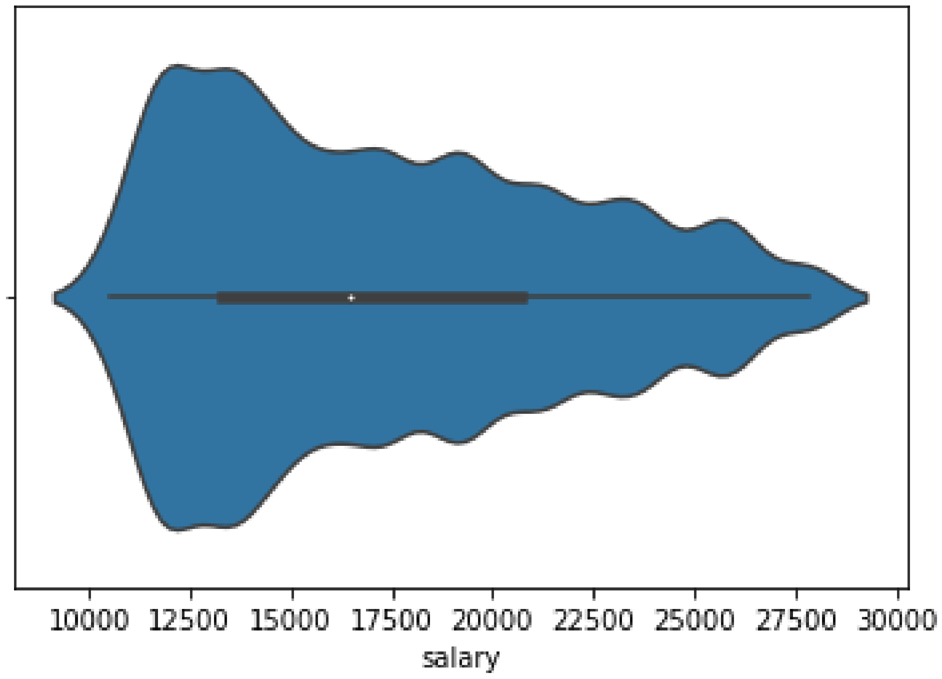
Violin plot (distribution)

```
ax = sns.violinplot(x="salary", data=salary)
```

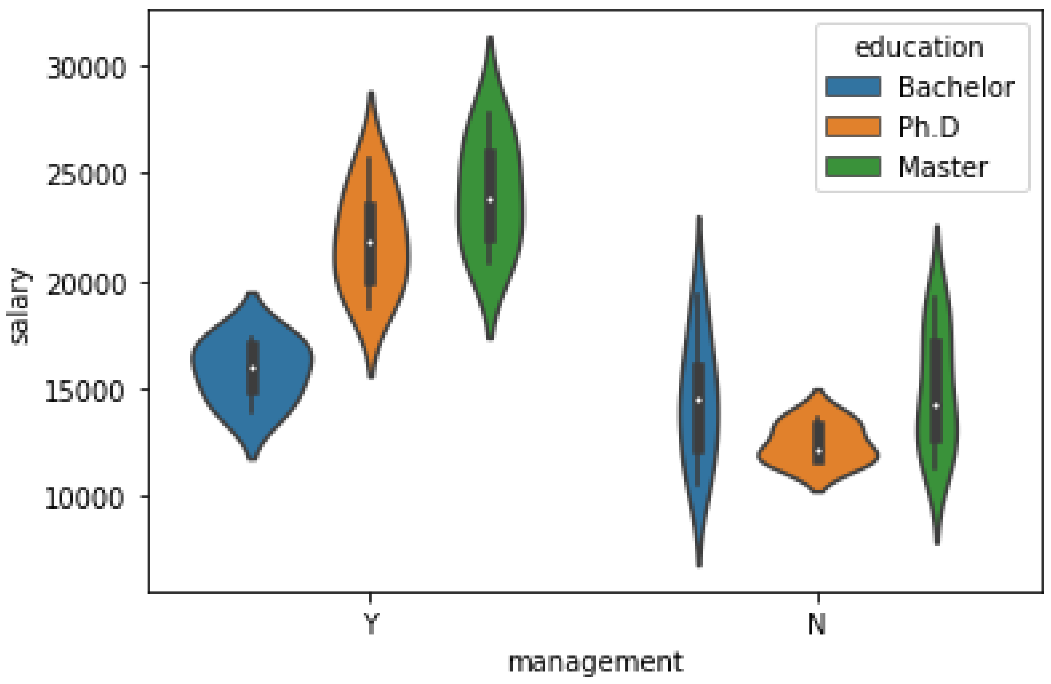


Tune bandwidth

```
ax = sns.violinplot(x="salary", data=salary, bw=.15)
```



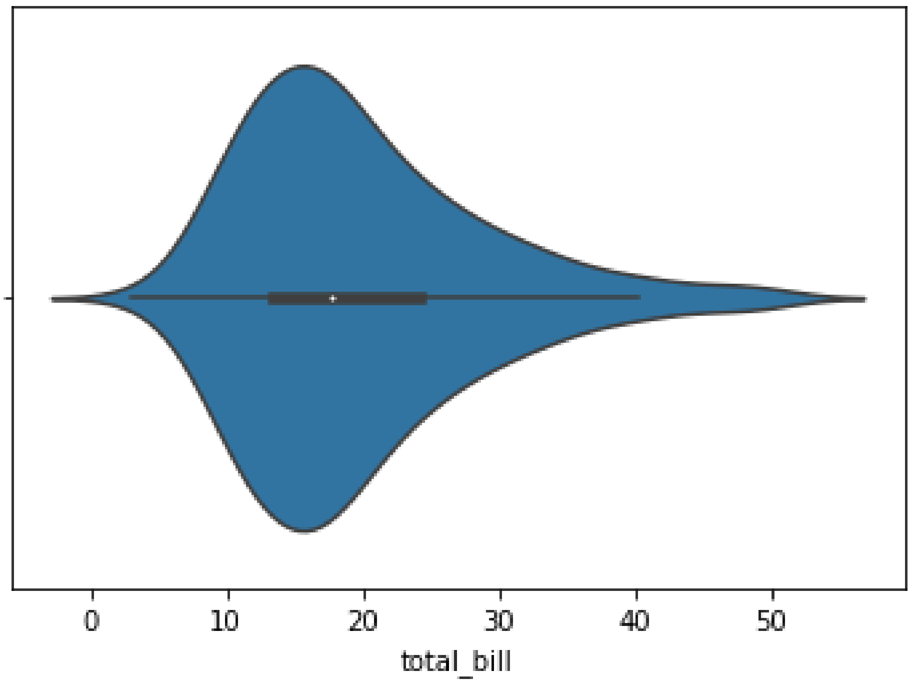
```
ax = sns.violinplot(x="management", y="salary", hue="education", data=salary)
```



Tips dataset One waiter recorded information about each tip he received over a period of a few months working in one restaurant. He collected several variables:

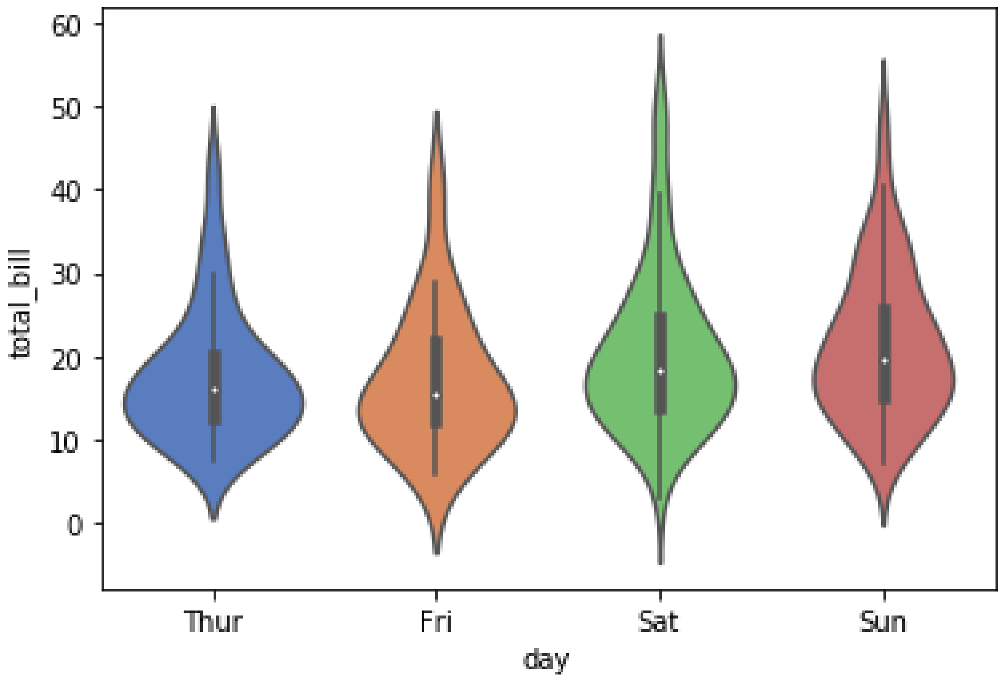
```
import seaborn as sns
#sns.set(style="whitegrid")
tips = sns.load_dataset("tips")
print(tips.head())

ax = sns.violinplot(x=tips["total_bill"])
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```



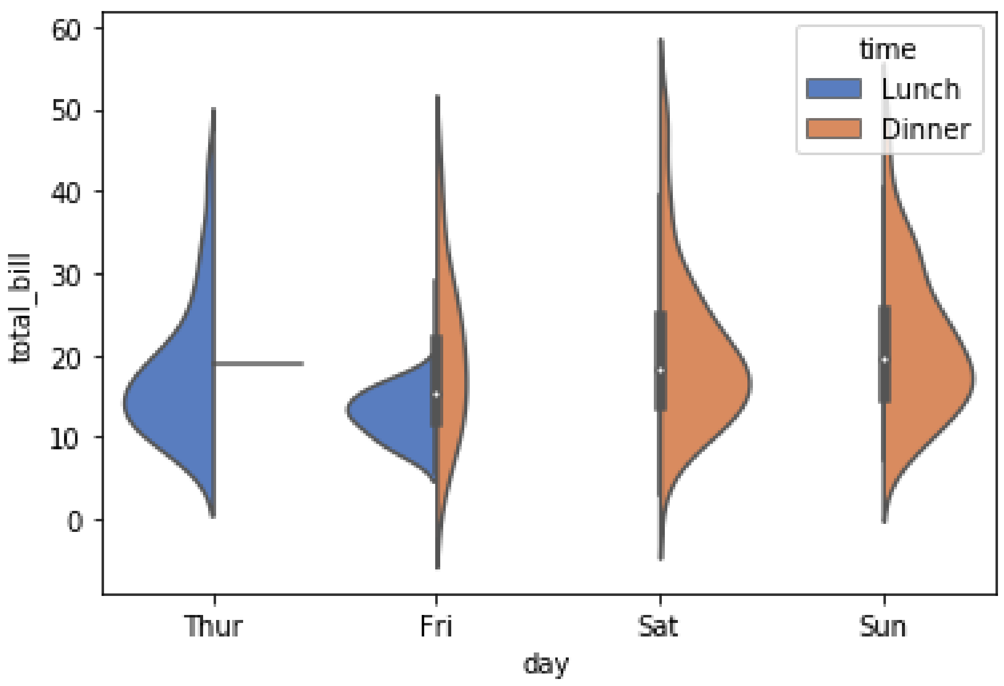
Group by day

```
ax = sns.violinplot(x="day", y="total_bill", data=tips, palette="muted")
```



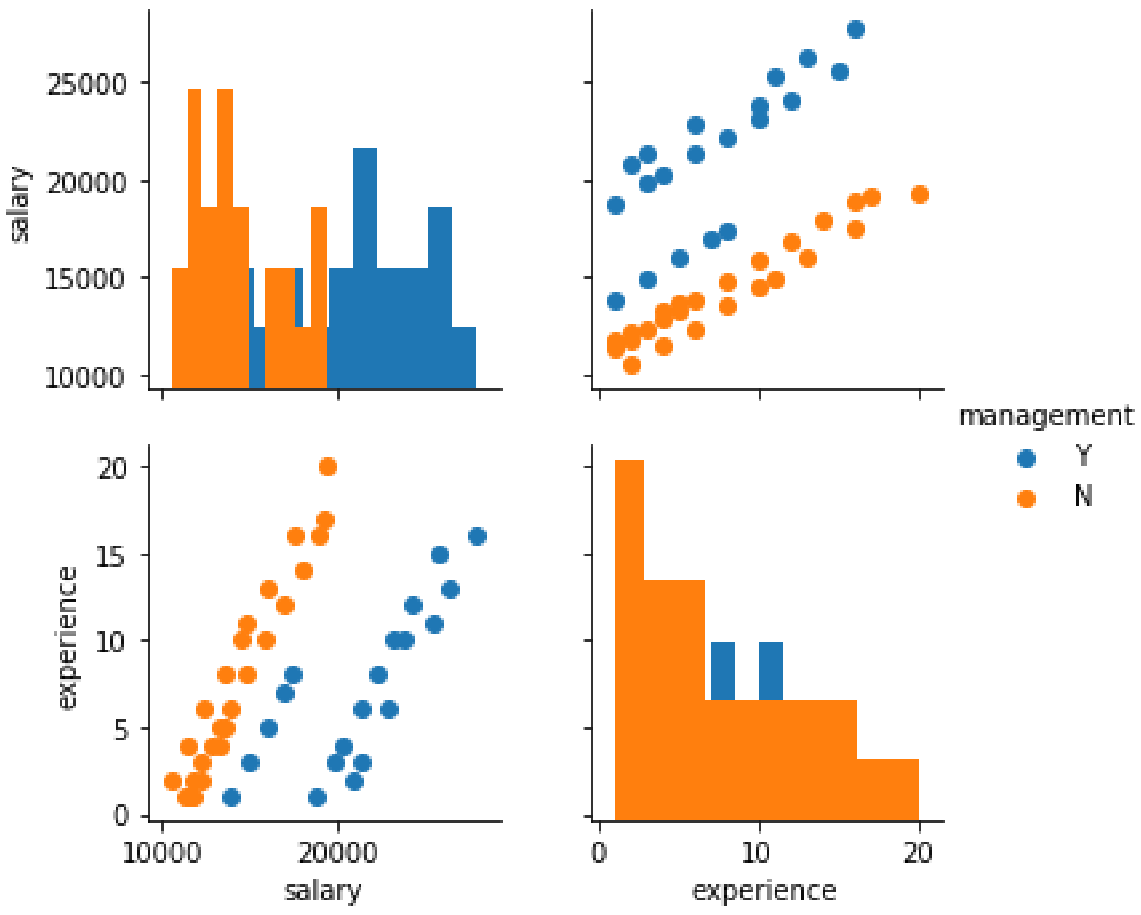
Group by day and color by time (lunch vs dinner)

```
ax = sns.violinplot(x="day", y="total_bill", hue="time", data=tips, palette="muted",␣
↪split=True)
```



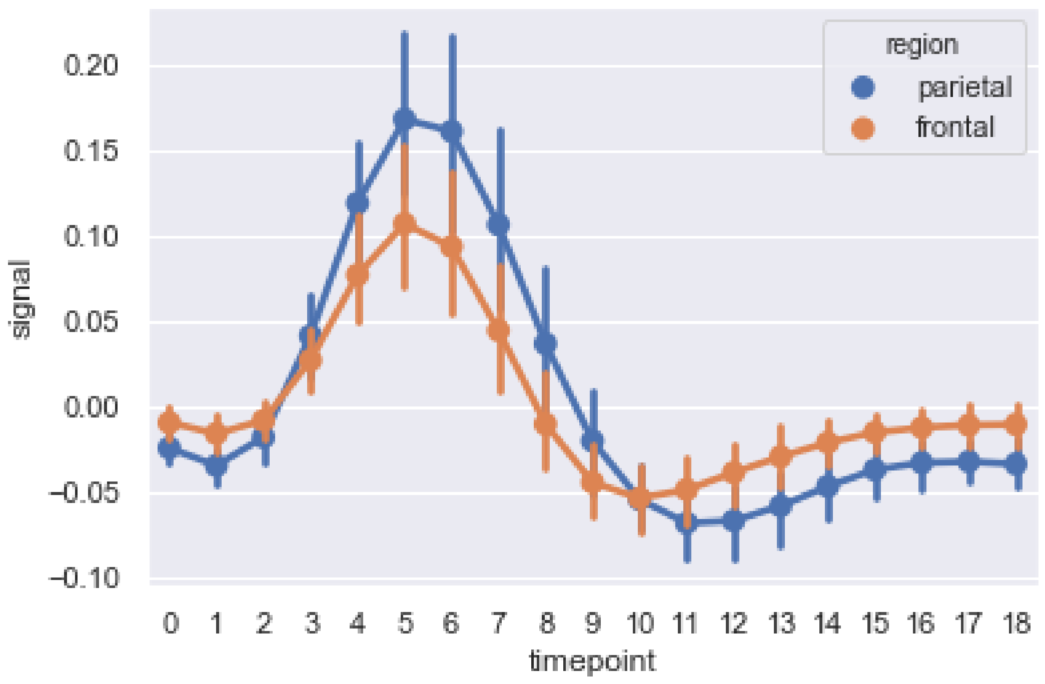
Pairwise scatter plots

```
g = sns.PairGrid(salary, hue="management")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
ax = g.add_legend()
```



3.3.5 Time series

```
import seaborn as sns
sns.set(style="darkgrid")

# Load an example dataset with long-form data
fmri = sns.load_dataset("fmri")

# Plot the responses for different events and regions


ax = sns.pointplot(x="timepoint", y="signal",
             hue="region", style="event",
             data=fmri)
# version 0.9
# sns.lineplot(x="timepoint", y="signal",
#              hue="region", style="event",
#              data=fmri)
```


region
parietal
frontal
signal
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
timepoint

CHAPTER

FOUR

STATISTICS

4.1 Univariate statistics

Basics univariate statistics are required to explore dataset:

- Discover associations between a variable of interest and potential predictors. It is strongly recommended to start with simple univariate methods before moving to complex multivariate predictors.
- Assess the prediction performances of machine learning predictors.
- Most of the univariate statistics are based on the linear model which is one of the main model in machine learning.

4.1.1 Estimators of the main statistical measures

Mean

Properties of the expected value operator $\mathrm{E}(\cdot)$ of a random variable X

$$E(X+c) = E(X) + c \tag{4.1}$$
$$E(X+Y) = E(X) + E(Y) \tag{4.2}$$
$$E(aX) = aE(X) \tag{4.3}$$

The estimator $\bar{x}$ on a sample of size n: $x = x_1, ..., x_n$ is given by

$$\bar{x} = \frac{1}{n}\sum_i x_i$$

$\bar{x}$ is itself a random variable with properties:

- $E(\bar{x}) = \bar{x}$,
- $Var(\bar{x}) = \frac{Var(X)}{n}$.

Variance

$$Var(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$$

The estimator is

$$\sigma_x^2 = \frac{1}{n-1}\sum_i (x_i - \bar{x})^2$$

Note here the subtracted 1 degree of freedom (df) in the divisor. In standard statistical practice, $df = 1$ provides an unbiased estimator of the variance of a hypothetical infinite population. With $df = 0$ it instead provides a maximum likelihood estimate of the variance for normally distributed variables.

Standard deviation

$$Std(X) = \sqrt{Var(X)}$$

The estimator is simply $\sigma_x = \sqrt{\sigma_x^2}$.

Covariance

$$Cov(X, Y) = E((X - E(X))(Y - E(Y))) = E(XY) - E(X)E(Y).$$

Properties:

$$\begin{aligned} \operatorname{Cov}(X, X) &= \operatorname{Var}(X) \\ \operatorname{Cov}(X, Y) &= \operatorname{Cov}(Y, X) \\ \operatorname{Cov}(cX, Y) &= c\operatorname{Cov}(X, Y) \\ \operatorname{Cov}(X + c, Y) &= \operatorname{Cov}(X, Y) \end{aligned}$$

The estimator with $df = 1$ is

$$\sigma_{xy} = \frac{1}{n-1}\sum_i (x_i - \bar{x})(y_i - \bar{y}).$$

Correlation

$$Cor(X, Y) = \frac{Cov(X, Y)}{Std(X)Std(Y)}$$

The estimator is

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}.$$

Standard Error (SE)

The standard error (SE) is the standard deviation (of the sampling distribution) of a statistic:

$$SE(X) = \frac{Std(X)}{\sqrt{n}}.$$

It is most commonly considered for the mean with the estimator

$$\begin{aligned} SE(x) &= Std(X) = \sigma_{\bar{x}} && (4.4) \\ &= \frac{\sigma_x}{\sqrt{n}}. && (4.5) \end{aligned}$$

Exercises

- Generate 2 random samples: $x \sim N(1.78, 0.1)$ and $y \sim N(1.66, 0.1)$, both of size 10.
- Compute $\bar{x}, \sigma_x, \sigma_{xy}$ (xbar, xvar, xycov) using only the np.sum() operation. Explore the np. module to find out which numpy functions performs the same computations and compare them (using assert) with your previous results.

4.1.2 Main distributions

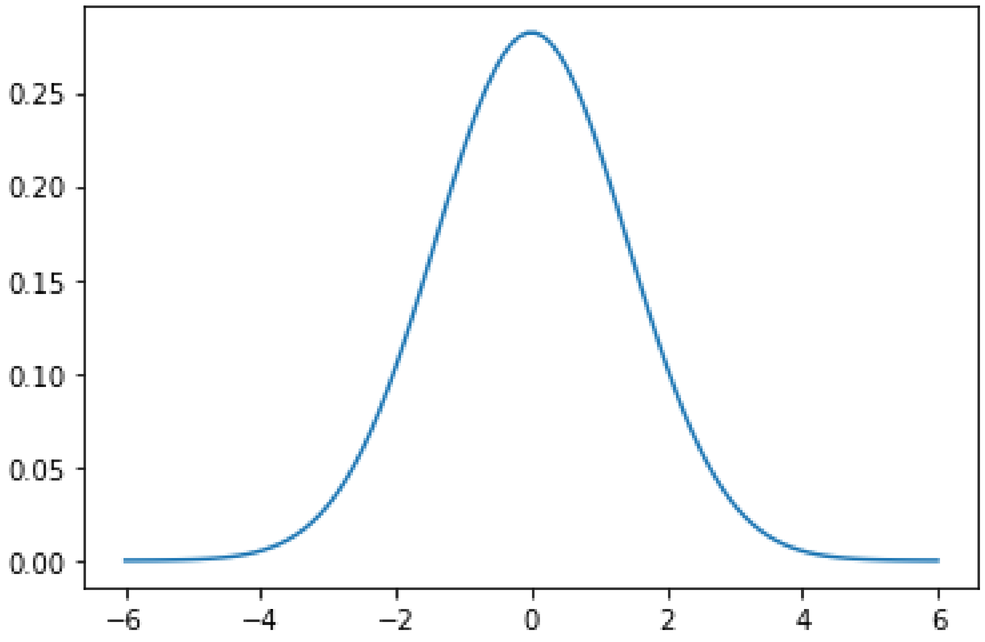
Normal distribution

The normal distribution, noted $\mathcal{N}(\mu, \sigma)$ with parameters: μ mean (location) and $\sigma > 0$ std-dev. Estimators: $\bar{x}$ and σ_x.

The normal distribution, noted $\mathcal{N}$, is useful because of the central limit theorem (CLT) which states that: given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined expected value and well-defined variance, will be approximately normally distributed, regardless of the underlying distribution.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
%matplotlib inline

mu = 0 # mean
variance = 2 #variance
sigma = np.sqrt(variance) #standard deviation",
x = np.linspace(mu-3*variance,mu+3*variance, 100)
plt.plot(x, norm.pdf(x, mu, sigma))
```

```
[<matplotlib.lines.Line2D at 0x7f5cd6d3afd0>]
```



The Chi-Square distribution

The chi-square or χ^2_n distribution with n degrees of freedom (df) is the distribution of a sum of the squares of n independent standard normal random variables $\mathcal{N}(0, 1)$. Let $X \sim \mathcal{N}(\mu, \sigma^2)$, then, $Z = (X - \mu)/\sigma \sim \mathcal{N}(0, 1)$, then:

- The squared standard $Z^2 \sim \chi^2_1$ (one df).
- **The distribution of sum of squares** of n normal random variables: $\sum_i^n Z_i^2 \sim \chi^2_n$

The sum of two χ^2 RV with p and q df is a χ^2 RV with $p + q$ df. This is useful when summing/subtracting sum of squares.

The χ^2-distribution is used to model **errors** measured as **sum of squares** or the distribution of the sample **variance**.

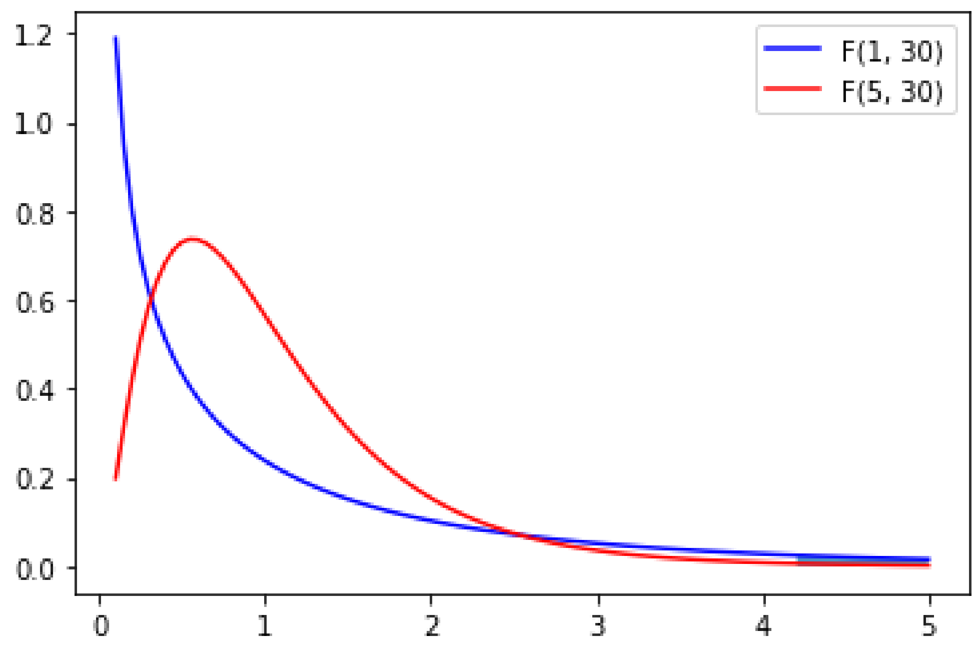
The Fisher's F-distribution

The F-distribution, $F_{n,p}$, with n and p degrees of freedom is the ratio of two independent χ^2 variables. Let $X \sim \chi^2_n$ and $Y \sim \chi^2_p$ then:

$$F_{n,p} = \frac{X/n}{Y/p}$$

The F-distribution plays a central role in hypothesis testing answering the question: **Are two variances equals?, is the ratio or two errors significantly large ?**.

```
import numpy as np
from scipy.stats import f
import matplotlib.pyplot as plt
%matplotlib inline

fvalues = np.linspace(.1, 5, 100)

# pdf(x, df1, df2): Probability density function at x of F.
plt.plot(fvalues, f.pdf(fvalues, 1, 30), 'b-', label="F(1, 30)")
plt.plot(fvalues, f.pdf(fvalues, 5, 30), 'r-', label="F(5, 30)")
plt.legend()

# cdf(x, df1, df2): Cumulative distribution function of F.
# ie.
proba_at_f_inf_3 = f.cdf(3, 1, 30) # P(F(1,30) < 3)

# ppf(q, df1, df2): Percent point function (inverse of cdf) at q of F.
f_at_proba_inf_95 = f.ppf(.95, 1, 30) # q such P(F(1,30) < .95)
assert f.cdf(f_at_proba_inf_95, 1, 30) == .95

# sf(x, df1, df2): Survival function (1 - cdf) at x of F.
proba_at_f_sup_3 = f.sf(3, 1, 30) # P(F(1,30) > 3)
assert  proba_at_f_inf_3 + proba_at_f_sup_3 == 1

# p-value: P(F(1, 30)) < 0.05
low_proba_fvalues = fvalues[fvalues > f_at_proba_inf_95]
plt.fill_between(low_proba_fvalues, 0, f.pdf(low_proba_fvalues, 1, 30),
                 alpha=.8, label="P < 0.05")
plt.show()
```



The Student's t-distribution

Let $M \sim \mathcal{N}(0, 1)$ and $V \sim \chi^2_n$. The t-distribution, T_n, with n degrees of freedom is the ratio:

$$T_n = \frac{M}{\sqrt{V/n}}$$

The distribution of the difference between an estimated parameter and its true (or assumed) value divided by the standard deviation of the estimated parameter (standard error) follow a t-distribution. **Is this parameters different from a given value?**

4.1.3 Hypothesis Testing

Examples

- Test a proportion: Biased coin ? 200 heads have been found over 300 flips, is it coins biased ?
- Test the association between two variables.
  - Exemple height and sex: In a sample of 25 individuals (15 females, 10 males), is female height is different from male height ?
  - Exemple age and arterial hypertension: In a sample of 25 individuals is age height correlated with arterial hypertension ?

Steps

1. Model the data
2. Fit: estimate the model parameters (frequency, mean, correlation, regression coeficient)
3. Compute a test statistic from model the parameters.
4. Formulate the null hypothesis: What would be the (distribution of the) test statistic if the observations are the result of pure chance.

5. Compute the probability (p-value) to obtain a larger value for the test statistic by chance (under the null hypothesis).

Flip coin: Simplified example

Biased coin ? 2 heads have been found over 3 flips, is it coins biased ?

1. Model the data: number of heads follow a Binomial disctribution.
2. Compute model parameters: N=3, P = the frequency of number of heads over the number of flip: 2/3.
3. Compute a test statistic, same as frequency.
4. Under the null hypothesis the distribution of the number of tail is:

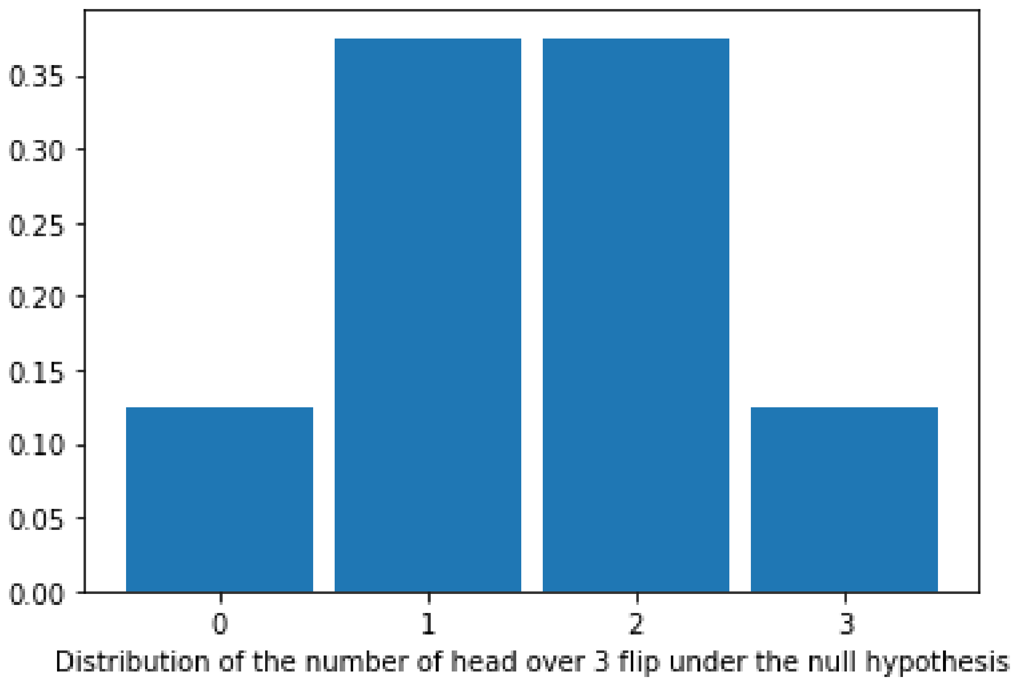
1	2	3	count #heads
			0
H			1
	H		1
		H	1
H	H		2
H		H	2
	H	H	2
H	H	H	3

8 possibles configurations, probabilities of differents values for p are: x measure the number of success.

- $P(x = 0) = 1/8$
- $P(x = 1) = 3/8$
- $P(x = 2) = 3/8$
- $P(x = 3) = 1/8$

```
plt.bar([0, 1, 2, 3], [1/8, 3/8, 3/8, 1/8], width=0.9)
_ = plt.xticks([0, 1, 2, 3], [0, 1, 2, 3])
plt.xlabel("Distribution of the number of head over 3 flip under the null hypothesis")
```

```
Text(0.5, 0, 'Distribution of the number of head over 3 flip under the null hypothesis')
```



3. Compute the probability (p-value) to observe a value larger or equal that 2 under the null hypothesis ? This probability is the p-value:

$$P(x \geq 2|H_0) = P(x = 2) + P(x = 3) = 3/8 + 1/8 = 4/8 = 1/2$$

Flip coin: Real Example

Biased coin ? 60 heads have been found over 100 flips, is it coins biased ?

1. Model the data: number of heads follow a Binomial disctribution.
2. Compute model parameters: N=100, P=60/100.
3. Compute a test statistic, same as frequency.
4. Compute a test statistic: 60/100.
5. Under the null hypothesis the distribution of the number of tail (k) follow the **binomial distribution** of parameters N=100, **P=0.5**:

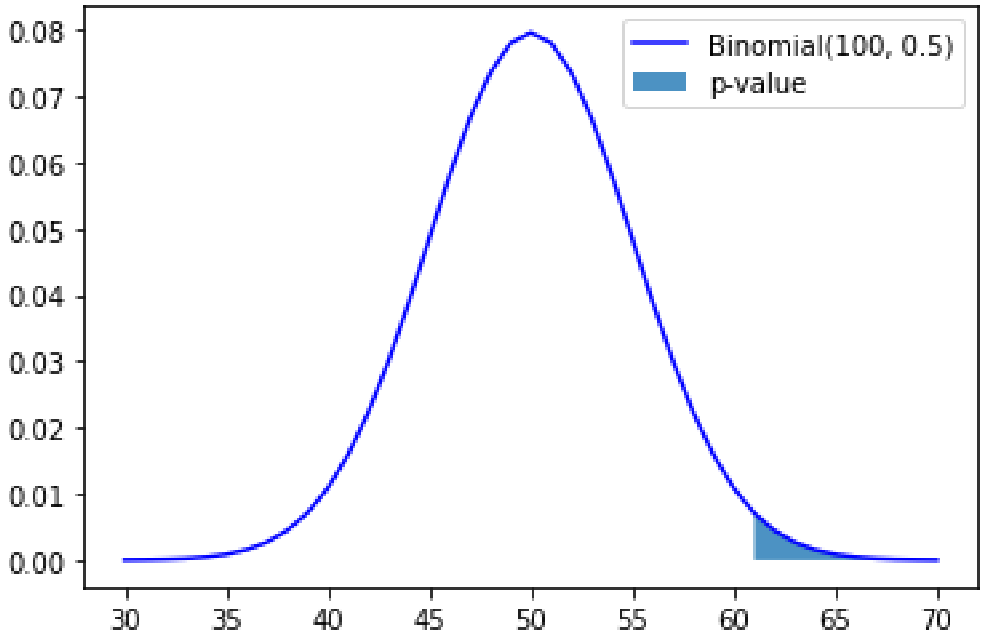
$$Pr(X = k|H_0) = Pr(X = k|n = 100, p = 0.5) = \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)}.$$

$$\begin{aligned} P(X = k \geq 60|H_0) &= \sum_{k=60}^{100} \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)} \\ &= 1 - \sum_{k=1}^{60} \binom{100}{k} 0.5^k (1 - 0.5)^{(100-k)}, \text{the cumulative distribution function.} \end{aligned}$$

Use tabulated binomial distribution

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

#tobs = 2.39687663116 # assume the t-value
succes = np.linspace(30, 70, 41)
plt.plot(succes, scipy.stats.binom.pmf(succes, 100, 0.5), 'b-', label="Binomial(100, 0.5)
↪")
upper_succes_tvalues = succes[succes > 60]
plt.fill_between(upper_succes_tvalues, 0, scipy.stats.binom.pmf(upper_succes_tvalues, 100,
↪ 0.5), alpha=.8, label="p-value")
_ = plt.legend()


pval = 1 - scipy.stats.binom.cdf(60, 100, 0.5)
print(pval)
```

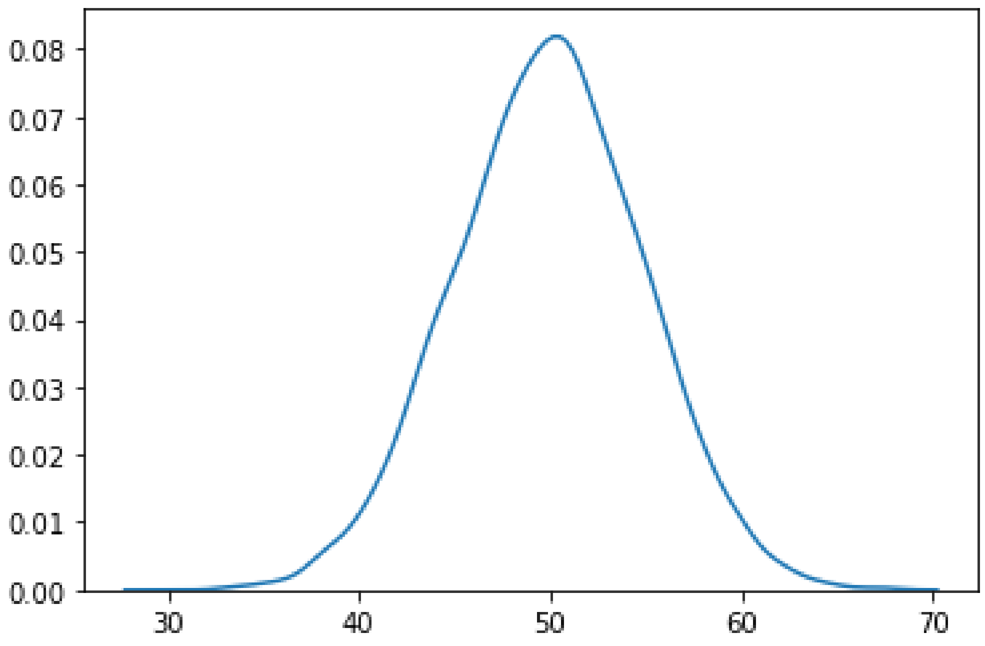
```
0.01760010010885238
```



Random sampling of the Binomial distribution under the null hypothesis

```
sccess_h0 = scipy.stats.binom.rvs(100, 0.5, size=10000, random_state=4)
print(sccess_h0)

#sccess_h0 = np.array([) for i in range(5000)])
import seaborn as sns
_ = sns.distplot(sccess_h0, hist=False)

pval_rnd = np.sum(sccess_h0 >= 60) / (len(sccess_h0) + 1)
print("P-value using monte-carlo sampling of the Binomial distribution under H0=", pval_
↪rnd)
```

```
[60 52 51 ... 45 51 44]
P-value using monte-carlo sampling of the Binomial distribution under H0= 0.
↪025897410258974102
```



One sample t-test

The one-sample t-test is used to determine whether a sample comes from a population with a specific mean. For example you want to test if the average height of a population is $1.75\ m$.

1 Model the data

Assume that height is normally distributed: $X \sim \mathcal{N}(\mu, \sigma)$, ie:

$$\text{height}_i = \text{average height over the population} + \text{error}_i \tag{4.6}$$

$$x_i = \bar{x} + \varepsilon_i \tag{4.7}$$

The ε_i are called the residuals

2 Fit: estimate the model parameters

$\bar{x}, s_x$ are the estimators of μ, σ.

3 Compute a test statistic

In testing the null hypothesis that the population mean is equal to a specified value $\mu_0 = 1.75$, one uses the statistic:

$$t = \frac{\text{difference of means}}{\text{std-dev of noise}}\sqrt{n} \tag{4.8}$$

$$t = \text{effect size}\sqrt{n} \tag{4.9}$$

$$t = \frac{\bar{x} - \mu_0}{s_x}\sqrt{n} \tag{4.10}$$

Remarks: Although the parent population does not need to be normally distributed, the distribution of the population of sample means, $\overline{x}$, is assumed to be normal. By the central limit theorem, if the sampling of the parent population is independent then the sample means will be approximately normal.

4 Compute the probability of the test statistic under the null hypotheis. This require to have the distribution of the t statistic under H_0.

Example

Given the following samples, we will test whether its true mean is 1.75.

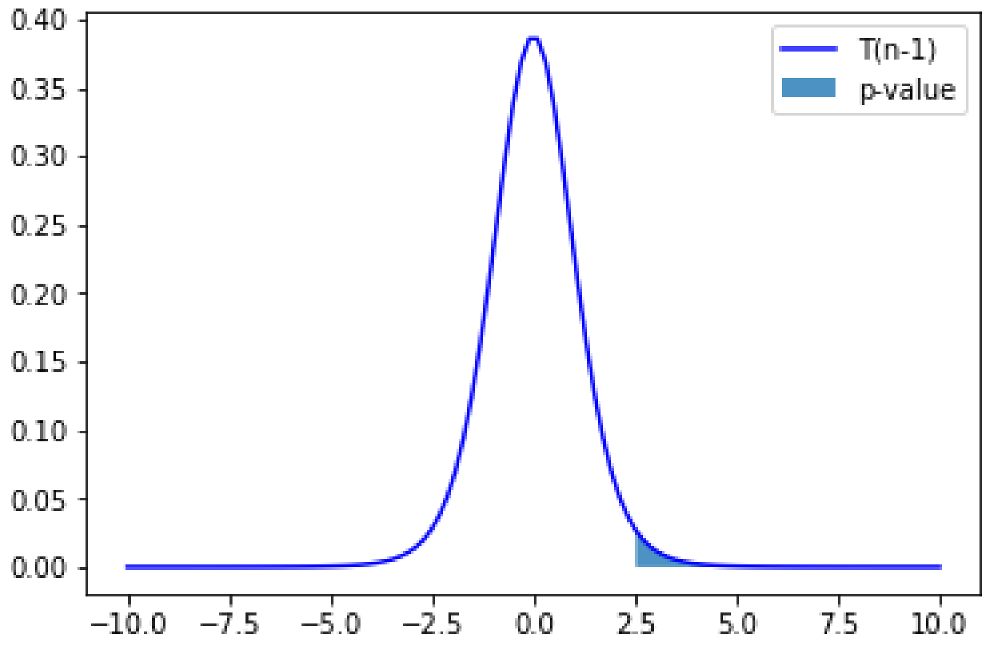
Warning, when computing the std or the variance, set `ddof=1`. The default value, `ddof=0`, leads to the biased estimator of the variance.

```
import numpy as np

x = [1.83,  1.83,  1.73,  1.82,  1.83,  1.73,  1.99,  1.85,  1.68,  1.87]

xbar = np.mean(x) # sample mean
mu0 = 1.75 # hypothesized value
s = np.std(x, ddof=1) # sample standard deviation
n = len(x) # sample size

print(xbar)

tobs = (xbar - mu0) / (s / np.sqrt(n))
print(tobs)
```

```
1.816
2.3968766311585883
```

The **:math:'p'-value** is the probability to observe a value t more extreme than the observed one t_{obs} under the null hypothesis H_0: $P(t > t_{obs}|H_0)$

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

#tobs = 2.39687663116 # assume the t-value
tvalues = np.linspace(-10, 10, 100)
plt.plot(tvalues, stats.t.pdf(tvalues, n-1), 'b-', label="T(n-1)")
upper_tval_tvalues = tvalues[tvalues > tobs]
plt.fill_between(upper_tval_tvalues, 0, stats.t.pdf(upper_tval_tvalues, n-1), alpha=.8,␣
↪label="p-value")
_ = plt.legend()
```



4.1.4 Testing pairwise associations

Univariate statistical analysis: explore association betweens pairs of variables.

- In statistics, a **categorical variable** or **factor** is a variable that can take on one of a limited, and usually fixed, number of possible values, thus assigning each individual to a particular group or "category". The levels are the possibles values of the variable. Number of levels = 2: binomial; Number of levels > 2: multinomial. There is no intrinsic ordering to the categories. For example, gender is a categorical variable having two categories (male and female) and there is no intrinsic ordering to the categories. For example, Sex (Female, Male), Hair color (blonde, brown, etc.).
- An **ordinal variable** is a categorical variable with a clear ordering of the levels. For example: drinks per day (none, small, medium and high).
- A **continuous** or **quantitative variable** $x \in \mathbb{R}$ is one that can take any value in a range of possible values, possibly infinite. E.g.: salary, experience in years, weight.

What statistical test should I use?

See: http://www.ats.ucla.edu/stat/mult_pkg/whatstat/

### Pearson correlation test: test association between two quantitative variables

Test the correlation coefficient of two quantitative variables. The test calculates a Pearson correlation coefficient and the p-value for testing non-correlation.

Let x and y two quantitative variables, where n samples were obeserved. The linear correlation coeficient is defined as :

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

Under H_0, the test statistic $t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}}$ follow Student distribution with $n - 2$ degrees of freedom.

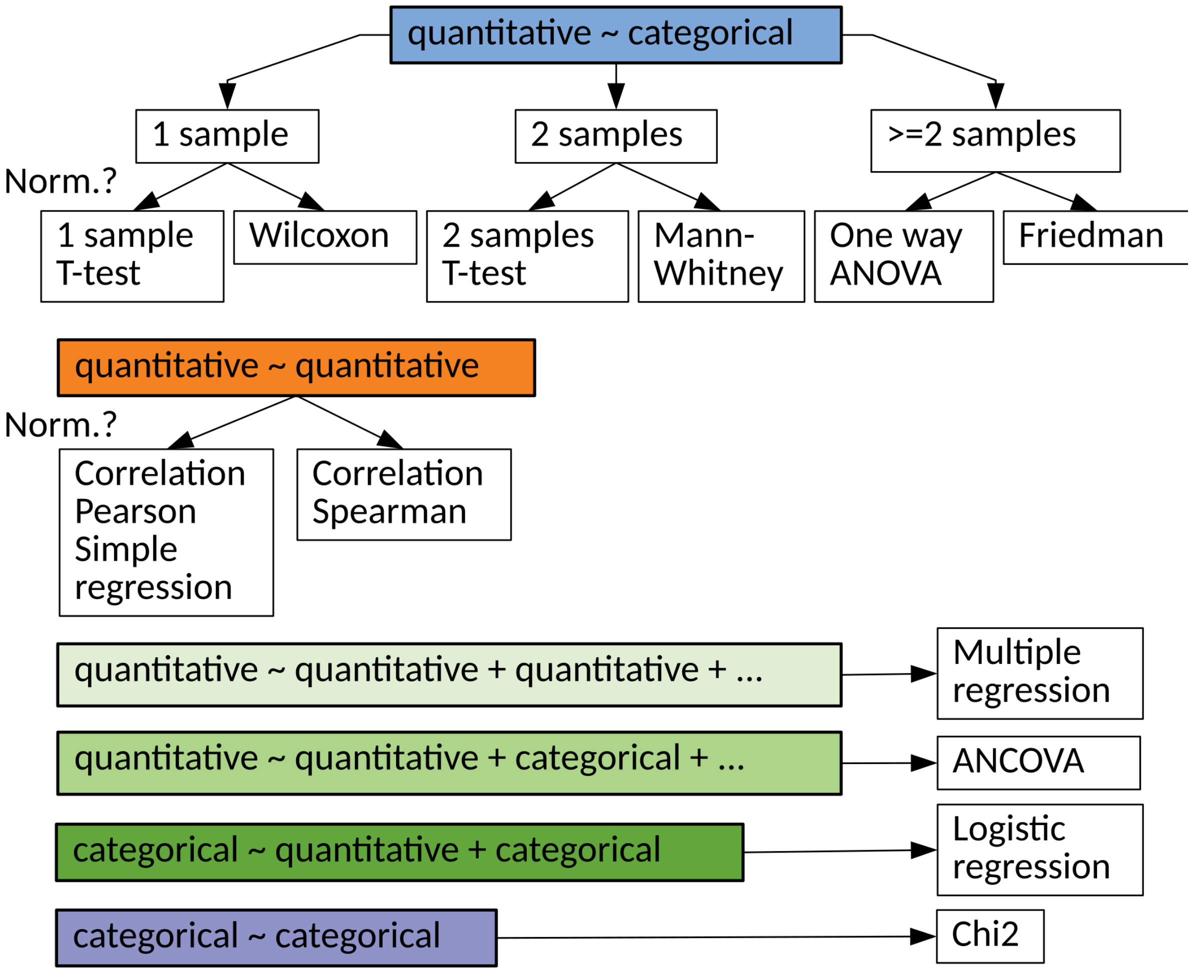

quantitative ~ categorical
1 sample
2 samples
>=2 samples
Norm.?
1 sample T-test
Wilcoxon
2 samples T-test
Mann-Whitney
One way ANOVA
Friedman
quantitative ~ quantitative
Norm.?
Correlation Pearson Simple regression
Correlation Spearman
quantitative ~ quantitative + quantitative + ...
Multiple regression
quantitative ~ quantitative + categorical + ...
ANCOVA
categorical ~ quantitative + categorical
Logistic regression
categorical ~ categorical
Chi2

Fig. 1: Statistical tests

```
import numpy as np
import scipy.stats as stats
n = 50
x = np.random.normal(size=n)
y = 2 * x + np.random.normal(size=n)

# Compute with scipy
cor, pval = stats.pearsonr(x, y)
print(cor, pval)
```

```
0.904453622242007 2.189729365511301e-19
```

Two sample (Student) t-test: compare two means

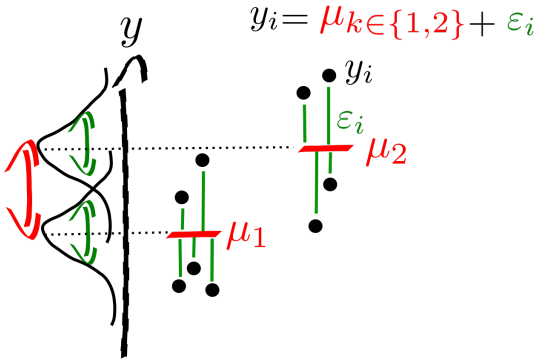


Fig. 2: Two-sample model

The two-sample t-test (Snedecor and Cochran, 1989) is used to determine if two population means are equal. There are several variations on this test. If data are paired (e.g. 2 measures, before and after treatment for each individual) use the one-sample t-test of the difference. The variances of the two samples may be assumed to be equal (a.k.a. homoscedasticity) or unequal (a.k.a. heteroscedasticity).

1. Model the data

Assume that the two random variables are normally distributed: $y_1 \sim \mathcal{N}(\mu_1, \sigma_1), y_2 \sim \mathcal{N}(\mu_2, \sigma_2)$.

2. Fit: estimate the model parameters

Estimate means and variances: $\bar{y_1}, s^2_{y_1}, \bar{y_2}, s^2_{y_2}$.

3. t-test

The general principle is

$$
\begin{aligned}
t &= \frac{\text{difference of means}}{\text{standard dev of error}} && (4.11)\\
&= \frac{\text{difference of means}}{\text{its standard error}} && (4.12)\\
&= \frac{\bar{y_1} - \bar{y_2}}{\sqrt{\sum \varepsilon^2}} \sqrt{n-2} && (4.13)\\
&= \frac{\bar{y_1} - \bar{y_2}}{s_{\bar{y_1} - \bar{y_2}}} && (4.14)
\end{aligned}
$$

Since y_1 and y_2 are independant:

$$
\begin{aligned}
s^2_{\bar{y_1} - \bar{y_2}} &= s^2_{\bar{y_1}} + s^2_{\bar{y_2}} = \frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2} && (4.15)\\
\text{thus} & && (4.16)\\
s_{\bar{y_1} - \bar{y_2}} &= \sqrt{\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}} && (4.17)
\end{aligned}
$$

Equal or unequal sample sizes, unequal variances (Welch's t-test)

Welch's t-test defines the t statistic as

$$t = \frac{\bar{y_1} - \bar{y_2}}{\sqrt{\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}}}.$$

To compute the p-value one needs the degrees of freedom associated with this variance estimate. It is approximated using the Welch–Satterthwaite equation:

$$\nu \approx \frac{\left(\frac{s^2_{y_1}}{n_1} + \frac{s^2_{y_2}}{n_2}\right)^2}{\frac{s^4_{y_1}}{n_1^2(n_1-1)} + \frac{s^4_{y_2}}{n_2^2(n_2-1)}}.$$

Equal or unequal sample sizes, equal variances

If we assume equal variance (ie, $s^2_{y_1} = s^2_{y_1} = s^2$), where s^2 is an estimator of the common variance of the two samples:

$$
\begin{aligned}
s^2 &= \frac{s_{y_1}^2(n_1-1) + s_{y_2}^2(n_2-1)}{n_1+n_2-2} && (4.18)\\
&= \frac{\sum_i^{n_1}(y_{1i} - \bar{y_1})^2 + \sum_j^{n_2}(y_{2j} - \bar{y_2})^2}{(n_1-1) + (n_2-1)} && (4.19)
\end{aligned}
$$

then

$$s_{\bar{y_1} - \bar{y_2}} = \sqrt{\frac{s^2}{n_1} + \frac{s^2}{n_2}} = s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Therefore, the t statistic, that is used to test whether the means are different is:

$$t = \frac{\bar{y_1} - \bar{y_2}}{s \cdot \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}},$$

Equal sample sizes, equal variances

If we simplify the problem assuming equal samples of size $n_1 = n_2 = n$ we get

$$\begin{aligned} t &= \frac{\bar{y_1} - \bar{y_2}}{s\sqrt{2}} \cdot \sqrt{n} && (4.20)\\ &\approx \text{effect size} \cdot \sqrt{n} && (4.21)\\ &\approx \frac{\text{difference of means}}{\text{standard deviation of the noise}} \cdot \sqrt{n} && (4.22) \end{aligned}$$

Example

Given the following two samples, test whether their means are equal using the **standard t-test, assuming equal variance**.

```
import scipy.stats as stats

height = np.array([ 1.83,  1.83,  1.73,  1.82,  1.83,  1.73,  1.99,  1.85,  1.68,  1.87,
                    1.66,  1.71,  1.73,  1.64,  1.70,  1.60,  1.79,  1.73,  1.62,  1.77])

grp = np.array(["M"] * 10 + ["F"] * 10)

# Compute with scipy
print(stats.ttest_ind(height[grp == "M"], height[grp == "F"], equal_var=True))
```

```
Ttest_indResult(statistic=3.5511519888466885, pvalue=0.00228208937112721)
```

ANOVA F-test (quantitative ~ categorial (>=2 levels))

Analysis of variance (ANOVA) provides a statistical test of whether or not the means of several groups are equal, and therefore generalizes the t-test to more than two groups. ANOVAs are useful for comparing (testing) three or more means (groups or variables) for statistical significance. It is conceptually similar to multiple two-sample t-tests, but is less conservative.

Here we will consider one-way ANOVA with one independent variable, ie one-way anova.

Wikipedia:

- Test if any group is on average superior, or inferior, to the others versus the null hypothesis that all four strategies yield the same mean response
- Detect any of several possible differences.
- The advantage of the ANOVA F-test is that we do not need to pre-specify which strategies are to be compared, and we do not need to adjust for making multiple comparisons.

- The disadvantage of the ANOVA F-test is that if we reject the null hypothesis, we do not know which strategies can be said to be significantly different from the others.

1. Model the data

A company has applied three marketing strategies to three samples of customers in order increase their business volume. The marketing is asking whether the strategies led to different increases of business volume. Let y_1, y_2 and y_3 be the three samples of business volume increase.

Here we assume that the three populations were sampled from three random variables that are normally distributed. I.e., $Y_1 \sim N(\mu_1, \sigma_1), Y_2 \sim N(\mu_2, \sigma_2)$ and $Y_3 \sim N(\mu_3, \sigma_3)$.

2. Fit: estimate the model parameters

Estimate means and variances: $\bar{y}_i, \sigma_i, \quad \forall i \in \{1, 2, 3\}$.

3. F-test

The formula for the one-way ANOVA F-test statistic is

$$F = \frac{\text{Explained variance}}{\text{Unexplained variance}} \tag{4.23}$$

$$= \frac{\text{Between-group variability}}{\text{Within-group variability}} = \frac{s_B^2}{s_W^2}. \tag{4.24}$$

The "explained variance", or "between-group variability" is

$$s_B^2 = \sum_i n_i(\bar{y}_{i\cdot} - \bar{y})^2/(K-1),$$

where $\bar{y}_{i\cdot}$ denotes the sample mean in the ith group, n_i is the number of observations in the ith group, $\bar{y}$ denotes the overall mean of the data, and K denotes the number of groups.

The "unexplained variance", or "within-group variability" is

$$s_W^2 = \sum_{ij}(y_{ij} - \bar{y}_{i\cdot})^2/(N-K),$$

where y_{ij} is the jth observation in the ith out of K groups and N is the overall sample size. This F-statistic follows the F-distribution with $K-1$ and $N-K$ degrees of freedom under the null hypothesis. The statistic will be large if the between-group variability is large relative to the within-group variability, which is unlikely to happen if the population means of the groups all have the same value.

Note that when there are only two groups for the one-way ANOVA F-test, $F = t^2$ where t is the Student's t statistic.

Chi-square, χ^2 (categorial ~ categorial)

Computes the chi-square, χ^2, statistic and p-value for the hypothesis test of independence of frequencies in the observed contingency table (cross-table). The observed frequencies are tested against an expected contingency table obtained by computing expected frequencies based on the marginal sums under the assumption of independence.

Example: 20 participants: 10 exposed to some chemical product and 10 non exposed (exposed = 1 or 0). Among the 20 participants 10 had cancer 10 not (cancer = 1 or 0). χ^2 tests the association between those two variables.

```
import numpy as np
import pandas as pd
import scipy.stats as stats

# Dataset:
# 15 samples:
# 10 first exposed
exposed = np.array([1] * 10 + [0] * 10)
# 8 first with cancer, 10 without, the last two with.
cancer = np.array([1] * 8 + [0] * 10 + [1] * 2)

crosstab = pd.crosstab(exposed, cancer, rownames=['exposed'],
                       colnames=['cancer'])
print("Observed table:")
print("---------------")
print(crosstab)

chi2, pval, dof, expected = stats.chi2_contingency(crosstab)
print("Statistics:")
print("-----------")
print("Chi2 = %f, pval = %f" % (chi2, pval))
print("Expected table:")
print("---------------")
print(expected)
```

```
Observed table:
---------------
cancer   0  1
exposed
0        8  2
1        2  8
Statistics:
-----------
Chi2 = 5.000000, pval = 0.025347
Expected table:
---------------
[[5. 5.]
 [5. 5.]]
```

Computing expected cross-table

```
# Compute expected cross-table based on proportion
exposed_marg = crosstab.sum(axis=0)
exposed_freq = exposed_marg / exposed_marg.sum()
```

(continues on next page)

(continued from previous page)

```
cancer_marg = crosstab.sum(axis=1)
cancer_freq = cancer_marg / cancer_marg.sum()

print('Exposed frequency? Yes: %.2f' % exposed_freq[0],
      'No: %.2f' % exposed_freq[1])
print('Cancer frequency? Yes: %.2f' % cancer_freq[0],
      'No: %.2f' % cancer_freq[1])

print('Expected frequencies:')
print(np.outer(exposed_freq, cancer_freq))

print('Expected cross-table (frequencies * N): ')
print(np.outer(exposed_freq, cancer_freq) * len(exposed))
```

```
Exposed frequency? Yes: 0.50 No: 0.50
Cancer frequency? Yes: 0.50 No: 0.50
Expected frequencies:
[[0.25 0.25]
 [0.25 0.25]]
Expected cross-table (frequencies * N):
[[5. 5.]
 [5. 5.]]
```

4.1.5 Non-parametric test of pairwise associations

Spearman rank-order correlation (quantitative ~ quantitative)

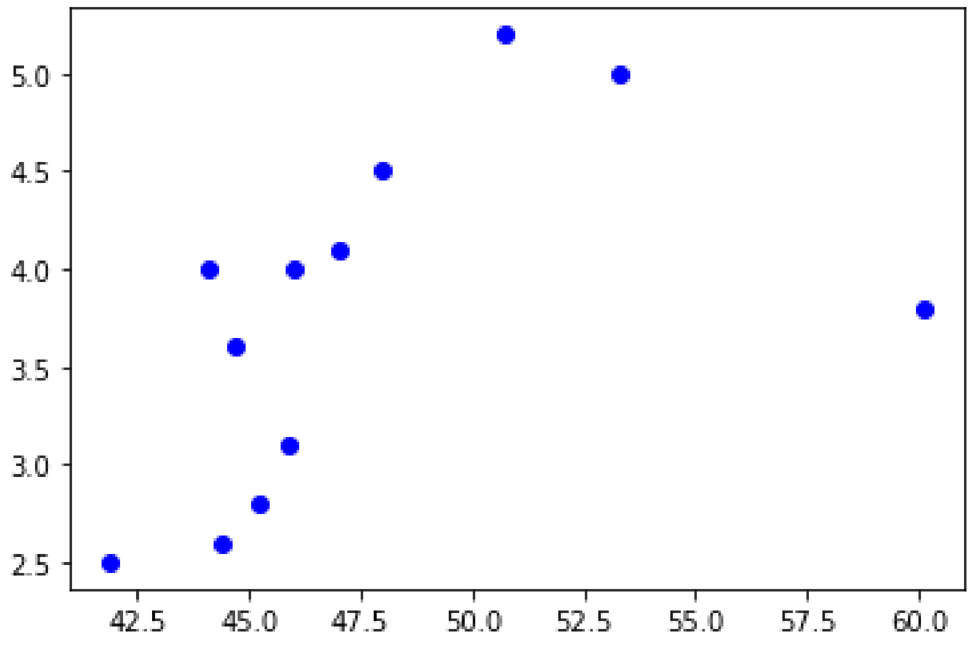
The Spearman correlation is a non-parametric measure of the monotonicity of the relationship between two datasets.

When to use it? Observe the data distribution: - presence of **outliers** - the distribution of the residuals is not Gaussian.

Like other correlation coefficients, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.

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

x = np.array([44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 46, 47, 48, 60.1])
y = np.array([2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8, 4, 4.1, 4.5, 3.8])

plt.plot(x, y, "bo")

# Non-Parametric Spearman
cor, pval = stats.spearmanr(x, y)
print("Non-Parametric Spearman cor test, cor: %.4f, pval: %.4f" % (cor, pval))

# "Parametric Pearson cor test
cor, pval = stats.pearsonr(x, y)
print("Parametric Pearson cor test: cor: %.4f, pval: %.4f" % (cor, pval))
```

```
Non-Parametric Spearman cor test, cor: 0.7110, pval: 0.0095
Parametric Pearson cor test: cor: 0.5263, pval: 0.0788
```



Wilcoxon signed-rank test (quantitative ~ cte)

Source: https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ (i.e. it is a paired difference test). It is equivalent to one-sample test of the difference of paired samples.

It can be used as an alternative to the paired Student's t-test, t-test for matched pairs, or the t-test for dependent samples when the population cannot be assumed to be normally distributed.

When to use it? Observe the data distribution: - presence of outliers - the distribution of the residuals is not Gaussian

It has a lower sensitivity compared to t-test. May be problematic to use when the sample size is small.

Null hypothesis H_0: difference between the pairs follows a symmetric distribution around zero.

```python
import scipy.stats as stats
n = 20
# Buisness Volume time 0
bv0 = np.random.normal(loc=3, scale=.1, size=n)
# Buisness Volume time 1
bv1 = bv0 + 0.1 + np.random.normal(loc=0, scale=.1, size=n)

# create an outlier
bv1[0] -= 10

# Paired t-test
```

(continues on next page)

(continued from previous page)

```
print(stats.ttest_rel(bv0, bv1))

# Wilcoxon
print(stats.wilcoxon(bv0, bv1))
```

```
Ttest_relResult(statistic=0.8167367438079456, pvalue=0.4242016933514212)
WilcoxonResult(statistic=40.0, pvalue=0.015240061183200121)
```

Mann–Whitney U test (quantitative ~ categorial (2 levels))

In statistics, the Mann–Whitney U test (also called the Mann–Whitney–Wilcoxon, Wilcoxon rank-sum test or Wilcoxon–Mann–Whitney test) is a nonparametric test of the null hypothesis that two samples come from the same population against an alternative hypothesis, especially that a particular population tends to have larger values than the other.

It can be applied on unknown distributions contrary to e.g. a t-test that has to be applied only on normal distributions, and it is nearly as efficient as the t-test on normal distributions.

```
import scipy.stats as stats
n = 20
# Buismess Volume group 0
bv0 = np.random.normal(loc=1, scale=.1, size=n)

# Buismess Volume group 1
bv1 = np.random.normal(loc=1.2, scale=.1, size=n)

# create an outlier
bv1[0] -= 10

# Two-samples t-test
print(stats.ttest_ind(bv0, bv1))

# Wilcoxon
print(stats.mannwhitneyu(bv0, bv1))
```

```
Ttest_indResult(statistic=0.6227075213159515, pvalue=0.5371960369300763)
MannwhitneyuResult(statistic=43.0, pvalue=1.1512354940556314e-05)
```

4.1.6 Linear model

Given n random samples $(y_i, x_{1i}, \ldots, x_{pi})$, $i = 1, \ldots, n$, the linear regression models the relation between the observations y_i and the independent variables x_i^p is formulated as

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} + \varepsilon_i \qquad i = 1, \ldots, n$$

- The β's are the model parameters, ie, the regression coeficients.
- β_0 is the intercept or the bias.
- ε_i are the **residuals**.
- **An independent variable (IV).** It is a variable that stands alone and isn't changed by the other variables you are trying to measure. For example, someone's age might be an

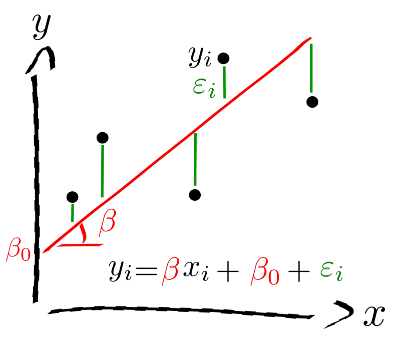


Fig. 3: Linear model

independent variable. Other factors (such as what they eat, how much they go to school, how much television they watch) aren't going to change a person's age. In fact, when you are looking for some kind of relationship between variables you are trying to see if the independent variable causes some kind of change in the other variables, or dependent variables. In Machine Learning, these variables are also called the **predictors**.

- A **dependent variable**. It is something that depends on other factors. For example, a test score could be a dependent variable because it could change depending on several factors such as how much you studied, how much sleep you got the night before you took the test, or even how hungry you were when you took it. Usually when you are looking for a relationship between two things you are trying to find out what makes the dependent variable change the way it does. In Machine Learning this variable is called a **target variable**.

Simple regression: test association between two quantitative variables

Using the dataset "salary", explore the association between the dependant variable (e.g. Salary) and the independent variable (e.g.: Experience is quantitative).

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
salary = pd.read_csv(url)
```

1. Model the data

Model the data on some **hypothesis** e.g.: salary is a linear function of the experience.

$$\text{salary}_i = \beta \ \text{experience}_i + \beta_0 + \epsilon_i,$$

more generally

$$y_i = \beta \ x_i + \beta_0 + \epsilon_i$$

- β: the slope or coefficient or parameter of the model,
- β_0: the **intercept** or **bias** is the second parameter of the model,

- ϵ_i: is the ith error, or residual with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

The simple regression is equivalent to the Pearson correlation.

2. Fit: estimate the model parameters

The goal it so estimate β, β_0 and σ^2.

Minimizes the **mean squared error (MSE)** or the **Sum squared error (SSE)**. The so-called **Ordinary Least Squares (OLS)** finds β, β_0 that minimizes the $SSE = \sum_i \epsilon_i^2$

$$SSE = \sum_i (y_i - \beta\ x_i - \beta_0)^2$$

Recall from calculus that an extreme point can be found by computing where the derivative is zero, i.e. to find the intercept, we perform the steps:

$$\begin{aligned}
\frac{\partial SSE}{\partial \beta_0} = \sum_i (y_i - \beta\ x_i - \beta_0) = 0\\
\sum_i y_i = \beta\ \sum_i x_i + n\ \beta_0\\
n\ \bar{y} = n\ \beta\ \bar{x} + n\ \beta_0\\
\beta_0 = \bar{y} - \beta\ \bar{x}
\end{aligned}$$

To find the regression coefficient, we perform the steps:

$$\frac{\partial SSE}{\partial \beta} = \sum_i x_i (y_i - \beta\ x_i - \beta_0) = 0$$

Plug in β_0:

$$\begin{aligned}
\sum_i x_i (y_i - \beta\ x_i - \bar{y} + \beta \bar{x}) = 0\\
\sum_i x_i y_i - \bar{y} \sum_i x_i = \beta \sum_i (x_i - \bar{x})
\end{aligned}$$

Divide both sides by n:

$$\begin{aligned}
\frac{1}{n} \sum_i x_i y_i - \bar{y}\bar{x} = \frac{1}{n} \beta \sum_i (x_i - \bar{x})\\
\beta = \frac{\frac{1}{n} \sum_i x_i y_i - \bar{y}\bar{x}}{\frac{1}{n} \sum_i (x_i - \bar{x})} = \frac{Cov(x, y)}{Var(x)}.
\end{aligned}$$

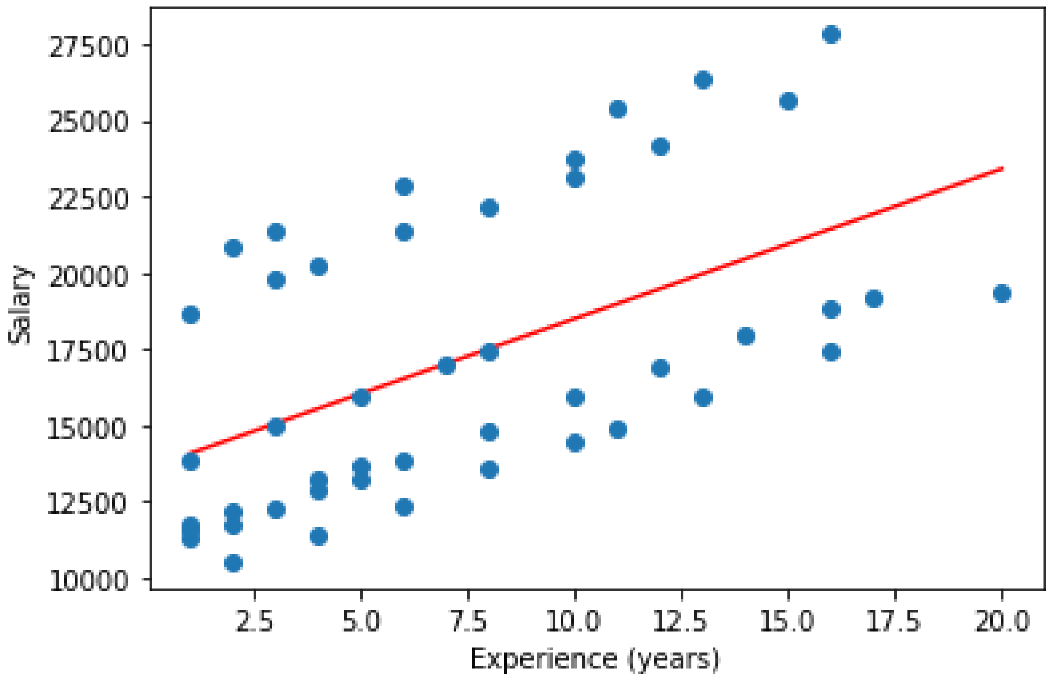
```python
from scipy import stats
import numpy as np
y, x = salary.salary, salary.experience
beta, beta0, r_value, p_value, std_err = stats.linregress(x,y)
print("y = %f x + %f,  r: %f, r-squared: %f,\np-value: %f, std_err: %f"
      % (beta, beta0, r_value, r_value**2, p_value, std_err))

print("Regression line with the scatterplot")
yhat = beta * x  +  beta0 # regression line
```

(continues on next page)

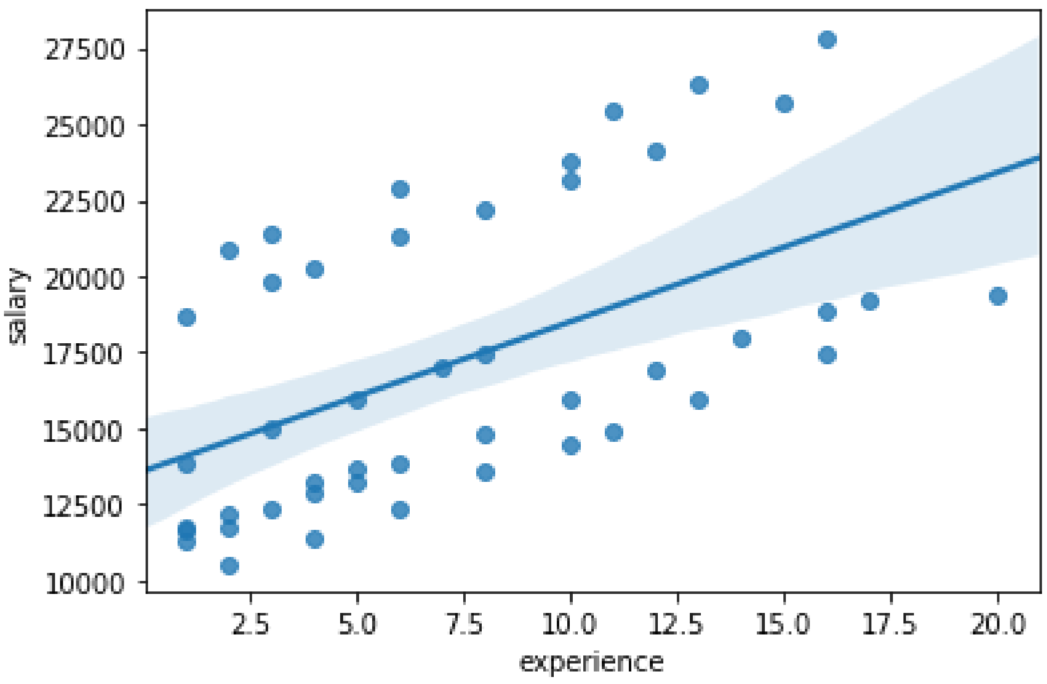
(continued from previous page)

```
plt.plot(x, yhat, 'r-', x, y,'o')
plt.xlabel('Experience (years)')
plt.ylabel('Salary')
plt.show()

print("Using seaborn")
import seaborn as sns
sns.regplot(x="experience", y="salary", data=salary);
```

```
y = 491.486913 x + 13584.043803,  r: 0.538886, r-squared: 0.290398,
p-value: 0.000112, std_err: 115.823381
Regression line with the scatterplot
```



```
Using seaborn
```



3. F-Test

3.1 Goodness of fit

The goodness of fit of a statistical model describes how well it fits a set of observations. Measures of goodness of fit typically summarize the discrepancy between observed values and the values expected under the model in question. We will consider the **explained variance** also known as the coefficient of determination, denoted R^2 pronounced **R-squared**.

The total sum of squares, SS_{tot} is the sum of the sum of squares explained by the regression, SS_{reg}, plus the sum of squares of residuals unexplained by the regression, SS_{res}, also called the SSE, i.e. such that

$$SS_{\text{tot}} = SS_{\text{reg}} + SS_{\text{res}}$$

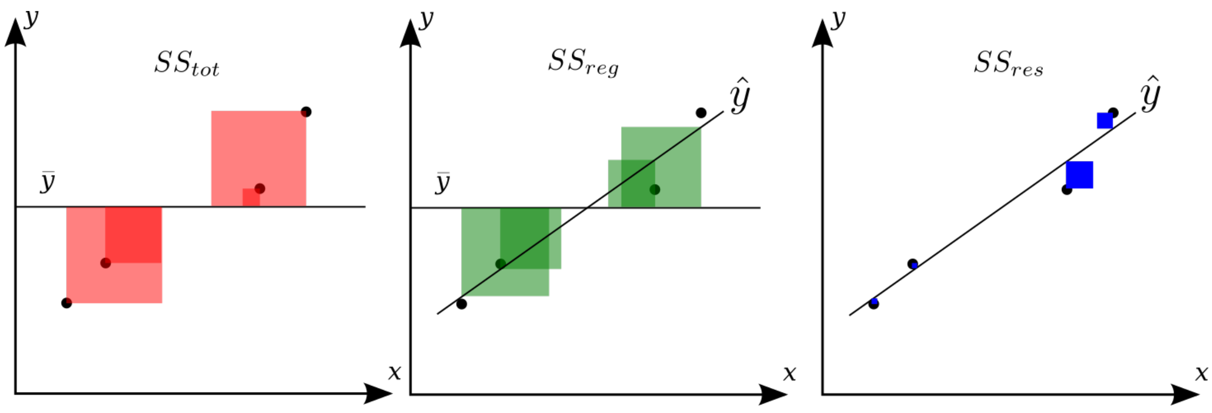


Fig. 4: title

The mean of y is

$$\bar{y} = \frac{1}{n} \sum_i y_i.$$

The total sum of squares is the total squared sum of deviations from the mean of y, i.e.

$$SS_\text{tot} = \sum_i (y_i - \bar{y})^2$$

The regression sum of squares, also called the explained sum of squares:

$$SS_\text{reg} = \sum_i (\hat{y}_i - \bar{y})^2,$$

where $\hat{y}_i = \beta x_i + \beta_0$ is the estimated value of salary $\hat{y}_i$ given a value of experience x_i.

The sum of squares of the residuals, also called the residual sum of squares (RSS) is:

$$SS_\text{res} = \sum_i (y_i - \hat{y}_i)^2.$$

R^2 is the explained sum of squares of errors. It is the variance explain by the regression divided by the total variance, i.e.

$$R^2 = \frac{\text{explained SS}}{\text{total SS}} = \frac{SS_\text{reg}}{SS_{tot}} = 1 - \frac{SS_{res}}{SS_{tot}}.$$

3.2 Test

Let $\hat{\sigma}^2 = SS_\text{res}/(n-2)$ be an estimator of the variance of ϵ. The 2 in the denominator stems from the 2 estimated parameters: intercept and coefficient.

- **Unexplained variance:** $\frac{SS_\text{res}}{\hat{\sigma}^2} \sim \chi^2_{n-2}$
- **Explained variance:** $\frac{SS_\text{reg}}{\hat{\sigma}^2} \sim \chi^2_1$. The single degree of freedom comes from the difference between $\frac{SS_\text{tot}}{\hat{\sigma}^2} (\sim \chi^2_{n-1})$ and $\frac{SS_\text{res}}{\hat{\sigma}^2} (\sim \chi^2_{n-2})$, i.e. $(n-1)-(n-2)$ degree of freedom.

The Fisher statistics of the ratio of two variances:

$$F = \frac{\text{Explained variance}}{\text{Unexplained variance}} = \frac{SS_\text{reg}/1}{SS_\text{res}/(n-2)} \sim F(1, n-2)$$

Using the F-distribution, compute the probability of observing a value greater than F under H_0, i.e.: $P(x > F|H_0)$, i.e. the survival function $(1 -$ Cumulative Distribution Function$)$ at x of the given F-distribution.

Multiple regression

Theory

Muliple Linear Regression is the most basic supervised learning algorithm.

Given: a set of training data $\{x_1, ..., x_N\}$ with corresponding targets $\{y_1, ..., y_N\}$.

In linear regression, we assume that the model that generates the data involves only a linear combination of the input variables, i.e.

$$y(x_i, \beta) = \beta^0 + \beta^1 x_i^1 + ... + \beta^P x_i^P,$$

or, simplified

$$y(x_i, \beta) = \beta_0 + \sum_{j=1}^{P-1} \beta_j x_i^j.$$

Extending each sample with an intercept, $x_i := [1, x_i] \in R^{P+1}$ allows us to use a more general notation based on linear algebra and write it as a simple dot product:

$$y(x_i, \beta) = x_i^T \beta,$$

where $\beta \in R^{P+1}$ is a vector of weights that define the $P+1$ parameters of the model. From now we have P regressors + the intercept.

Minimize the Mean Squared Error MSE loss:

$$MSE(\beta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - y(x_i, \beta))^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i^T \beta)^2$$

Let $X = [x_0^T, ..., x_N^T]$ be a $N \times P+1$ matrix of N samples of P input features with one column of one and let be $y = [y_1, ..., y_N]$ be a vector of the N targets. Then, using linear algebra, the **mean squared error (MSE) loss can be rewritten**:

$$MSE(\beta) = \frac{1}{N} ||y - X\beta||_2^2.$$

The β that minimises the MSE can be found by:

$$\nabla_\beta \left(\frac{1}{N} ||y - X\beta||_2^2 \right) = 0 \tag{4.25}$$

$$\frac{1}{N} \nabla_\beta (y - X\beta)^T (y - X\beta) = 0 \tag{4.26}$$

$$\frac{1}{N} \nabla_\beta (y^T y - 2\beta^T X^T y + \beta^T X^T X\beta) = 0 \tag{4.27}$$

$$-2X^T y + 2X^T X\beta = 0 \tag{4.28}$$

$$X^T X\beta = X^T y \tag{4.29}$$

$$\beta = (X^T X)^{-1} X^T y, \tag{4.30}$$

where $(X^T X)^{-1} X^T$ is a pseudo inverse of X.

Fit with numpy

```
import numpy as np
from scipy import linalg
np.random.seed(seed=42)  # make the example reproducible
```

(continues on next page)

(continued from previous page)

```
# Dataset
N, P = 50, 4
X = np.random.normal(size= N * P).reshape((N, P))
## Our model needs an intercept so we add a column of 1s:
X[:, 0] = 1
print(X[:5, :])

betastar = np.array([10, 1., .5, 0.1])
e = np.random.normal(size=N)
y = np.dot(X, betastar) + e

# Estimate the parameters
Xpinv = linalg.pinv2(X)
betahat = np.dot(Xpinv, y)
print("Estimated beta:\n", betahat)
```

```
[[ 1.         -0.1382643   0.64768854  1.52302986]
 [ 1.         -0.23413696  1.57921282  0.76743473]
 [ 1.          0.54256004 -0.46341769 -0.46572975]
 [ 1.         -1.91328024 -1.72491783 -0.56228753]
 [ 1.          0.31424733 -0.90802408 -1.4123037 ]]
Estimated beta:
 [10.14742501  0.57938106  0.51654653  0.17862194]
```

4.1.7 Linear model with statsmodels

Sources: http://statsmodels.sourceforge.net/devel/examples/

Multiple regression

Interface with statsmodels

```
import statsmodels.api as sm

## Fit and summary:
model = sm.OLS(y, X).fit()
print(model.summary())

# prediction of new values
ypred = model.predict(X)

# residuals + prediction == true values
assert np.all(ypred + model.resid == y)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.363
Model:                            OLS   Adj. R-squared:                  0.322
Method:                 Least Squares   F-statistic:                     8.748
Date:                Wed, 06 Nov 2019   Prob (F-statistic):           0.000106
```

(continues on next page)

(continued from previous page)

```
Time:                        18:03:24   Log-Likelihood:                -71.271
No. Observations:                  50   AIC:                             150.5
Df Residuals:                      46   BIC:                             158.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         10.1474      0.150     67.520      0.000       9.845      10.450
x1             0.5794      0.160      3.623      0.001       0.258       0.901
x2             0.5165      0.151      3.425      0.001       0.213       0.820
x3             0.1786      0.144      1.240      0.221      -0.111       0.469
==============================================================================
Omnibus:                        2.493   Durbin-Watson:                   2.369
Prob(Omnibus):                  0.288   Jarque-Bera (JB):                1.544
Skew:                           0.330   Prob(JB):                        0.462
Kurtosis:                       3.554   Cond. No.                         1.27
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
```

Interface with Pandas

Use R language syntax for data.frame. For an additive model: $y_i = \beta^0 + x_i^1 \beta^1 + x_i^2 \beta^2 + \epsilon_i \equiv$ `y ~ x1 + x2`.

```
import statsmodels.formula.api as smfrmla

df = pd.DataFrame(np.column_stack([X, y]), columns=['inter', 'x1','x2', 'x3', 'y'])
print(df.columns, df.shape)
# Build a model excluding the intercept, it is implicit
model = smfrmla.ols("y~x1 + x2 + x3", df).fit()
print(model.summary())
```

```
Index(['inter', 'x1', 'x2', 'x3', 'y'], dtype='object') (50, 5)
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.363
Model:                            OLS   Adj. R-squared:                  0.322
Method:                 Least Squares   F-statistic:                     8.748
Date:                Wed, 06 Nov 2019   Prob (F-statistic):           0.000106
Time:                        18:03:24   Log-Likelihood:                -71.271
No. Observations:                  50   AIC:                             150.5
Df Residuals:                      46   BIC:                             158.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     10.1474      0.150     67.520      0.000       9.845      10.450
x1             0.5794      0.160      3.623      0.001       0.258       0.901
```

(continues on next page)

(continued from previous page)

```
x2             0.5165      0.151      3.425      0.001       0.213       0.820
x3             0.1786      0.144      1.240      0.221      -0.111       0.469
==============================================================================
Omnibus:                        2.493   Durbin-Watson:                   2.369
Prob(Omnibus):                  0.288   Jarque-Bera (JB):                1.544
Skew:                           0.330   Prob(JB):                        0.462
Kurtosis:                       3.554   Cond. No.                         1.27
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
```

Multiple regression with categorical independent variables or factors: Analysis of covariance (ANCOVA)

Analysis of covariance (ANCOVA) is a linear model that blends ANOVA and linear regression. ANCOVA evaluates whether population means of a dependent variable (DV) are equal across levels of a categorical independent variable (IV) often called a treatment, while statistically controlling for the effects of other quantitative or continuous variables that are not of primary interest, known as covariates (CV).

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

try:
    df = pd.read_csv("../datasets/salary_table.csv")
except:
    url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
    df = pd.read_csv(url)
```

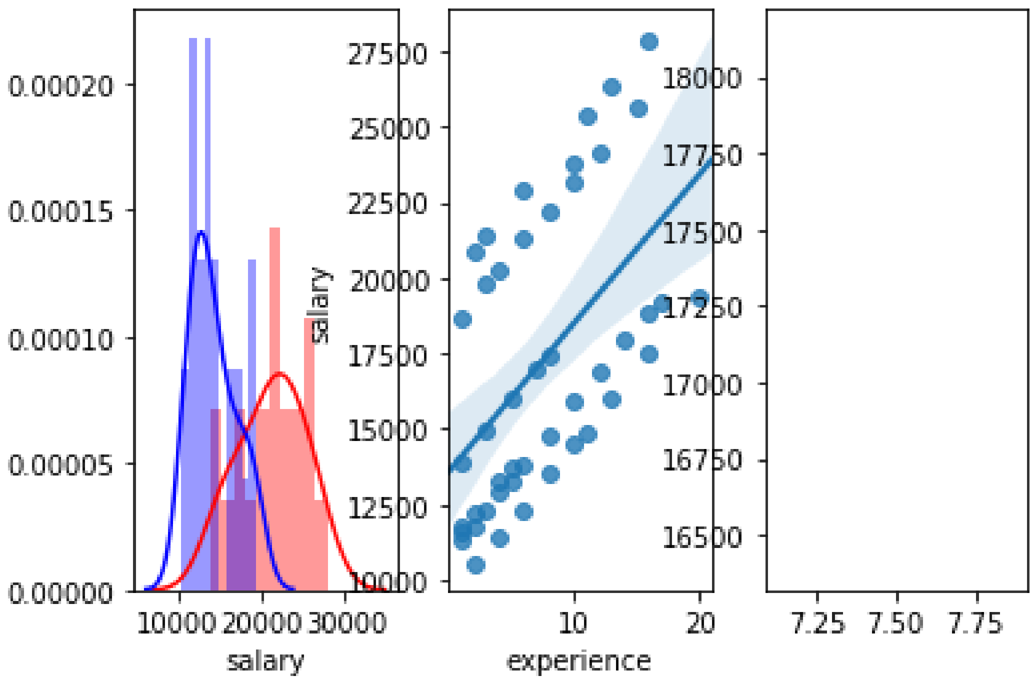
```
import seaborn as sns
fig, axes = plt.subplots(1, 3)

sns.distplot(df.salary[df.management == "Y"], color="r", bins=10, label="Manager:Y",␣
↪ax=axes[0])
sns.distplot(df.salary[df.management == "N"], color="b", bins=10, label="Manager:Y",␣
↪ax=axes[0])

sns.regplot("experience", "salary", data=df, ax=axes[1])

sns.regplot("experience", "salary", color=df.management,
            data=df, ax=axes[2])

#sns.stripplot("experience", "salary", hue="management", data=df, ax=axes[2])
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-28-c2d69cab90c3> in <module>
      8
```

(continues on next page)

(continued from previous page)

```
      9 sns.regplot("experience", "salary", hue=df.management,
---> 10             data=df, ax=axes[2])
     11 
     12 #sns.stripplot("experience", "salary", hue="management", data=df, ax=axes[2])


TypeError: regplot() got an unexpected keyword argument 'hue'
```



One-way AN(C)OVA

- ANOVA: one categorical independent variable, i.e. one factor.
- ANCOVA: ANOVA with some covariates.

```python
import statsmodels.formula.api as smfrmla

oneway = smfrmla.ols('salary ~ management + experience', df).fit()
print(oneway.summary())
aov = sm.stats.anova_lm(oneway, typ=2) # Type 2 ANOVA DataFrame
print(aov)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 salary   R-squared:                       0.865
Model:                            OLS   Adj. R-squared:                  0.859
Method:                 Least Squares   F-statistic:                     138.2
Date:                Thu, 07 Nov 2019   Prob (F-statistic):           1.90e-19
Time:                        12:16:50   Log-Likelihood:                -407.76
No. Observations:                  46   AIC:                             821.5
Df Residuals:                      43   BIC:                             827.0
Df Model:                           2
```

(continues on next page)

(continued from previous page)

```
Covariance Type:            nonrobust
==================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept        1.021e+04    525.999     19.411      0.000    9149.578    1.13e+04
management[T.Y]  7145.0151    527.320     13.550      0.000    6081.572    8208.458
experience        527.1081     51.106     10.314      0.000     424.042     630.174
==============================================================================
Omnibus:                       11.437   Durbin-Watson:                   2.193
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               11.260
Skew:                          -1.131   Prob(JB):                      0.00359
Kurtosis:                       3.872   Cond. No.                         22.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
                  sum_sq    df           F        PR(>F)
management  5.755739e+08   1.0  183.593466  4.054116e-17
experience  3.334992e+08   1.0  106.377768  3.349662e-13
Residual    1.348070e+08  43.0         NaN           NaN
```

Two-way AN(C)OVA

Ancova with two categorical independent variables, i.e. two factors.

```python
import statsmodels.formula.api as smfrmla

twoway = smfrmla.ols('salary ~ education + management + experience', df).fit()
print(twoway.summary())
aov = sm.stats.anova_lm(twoway, typ=2) # Type 2 ANOVA DataFrame
print(aov)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 salary   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     226.8
Date:                Thu, 07 Nov 2019   Prob (F-statistic):           2.23e-27
Time:                        12:16:52   Log-Likelihood:                -381.63
No. Observations:                  46   AIC:                             773.3
Df Residuals:                      41   BIC:                             782.4
Df Model:                           4
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
Intercept            8035.5976    386.689     20.781      0.000    7254.663    8816.532
education[T.Master]  3144.0352    361.968      8.686      0.000    2413.025    3875.045
education[T.Ph.D]    2996.2103    411.753      7.277      0.000    2164.659    3827.762
management[T.Y]      6883.5310    313.919     21.928      0.000    6249.559    7517.503
experience            546.1840     30.519     17.896      0.000     484.549     607.819
==============================================================================
```

(continues on next page)

(continued from previous page)

```
Omnibus:                        2.293   Durbin-Watson:                   2.237
Prob(Omnibus):                  0.318   Jarque-Bera (JB):                1.362
Skew:                          -0.077   Prob(JB):                        0.506
Kurtosis:                       2.171   Cond. No.                         33.5
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
                  sum_sq    df           F        PR(>F)
education   9.152624e+07   2.0   43.351589  7.672450e-11
management  5.075724e+08   1.0  480.825394  2.901444e-24
experience  3.380979e+08   1.0  320.281524  5.546313e-21
Residual    4.328072e+07  41.0         NaN           NaN
```

Comparing two nested models

oneway is nested within twoway. Comparing two nested models tells us if the additional predictors (i.e. education) of the full model significantly decrease the residuals. Such comparison can be done using an F-test on residuals:

```
print(twoway.compare_f_test(oneway))  # return F, pval, df
```

```
(43.35158945918107, 7.672449570495418e-11, 2.0)
```

Factor coding

See http://statsmodels.sourceforge.net/devel/contrasts.html

By default Pandas use "dummy coding". Explore:

```
print(twoway.model.data.param_names)
print(twoway.model.data.exog[:10, :])
```

```
['Intercept', 'education[T.Master]', 'education[T.Ph.D]', 'management[T.Y]', 'experience']
[[1. 0. 0. 1. 1.]
 [1. 0. 1. 0. 1.]
 [1. 0. 1. 1. 1.]
 [1. 1. 0. 0. 1.]
 [1. 0. 1. 0. 1.]
 [1. 1. 0. 1. 2.]
 [1. 1. 0. 0. 2.]
 [1. 0. 0. 0. 2.]
 [1. 0. 1. 0. 2.]
 [1. 1. 0. 0. 3.]]
```

Contrasts and post-hoc tests

```
# t-test of the specific contribution of experience:
ttest_exp = twoway.t_test([0, 0, 0, 0, 1])
ttest_exp.pvalue, ttest_exp.tvalue
print(ttest_exp)

# Alternatively, you can specify the hypothesis tests using a string
twoway.t_test('experience')

# Post-hoc is salary of Master different salary of Ph.D?
# ie. t-test salary of Master = salary of Ph.D.
print(twoway.t_test('education[T.Master] = education[T.Ph.D]'))
```

```
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0           546.1840     30.519     17.896      0.000     484.549     607.819
==============================================================================
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0           147.8249    387.659      0.381      0.705    -635.069     930.719
==============================================================================
```

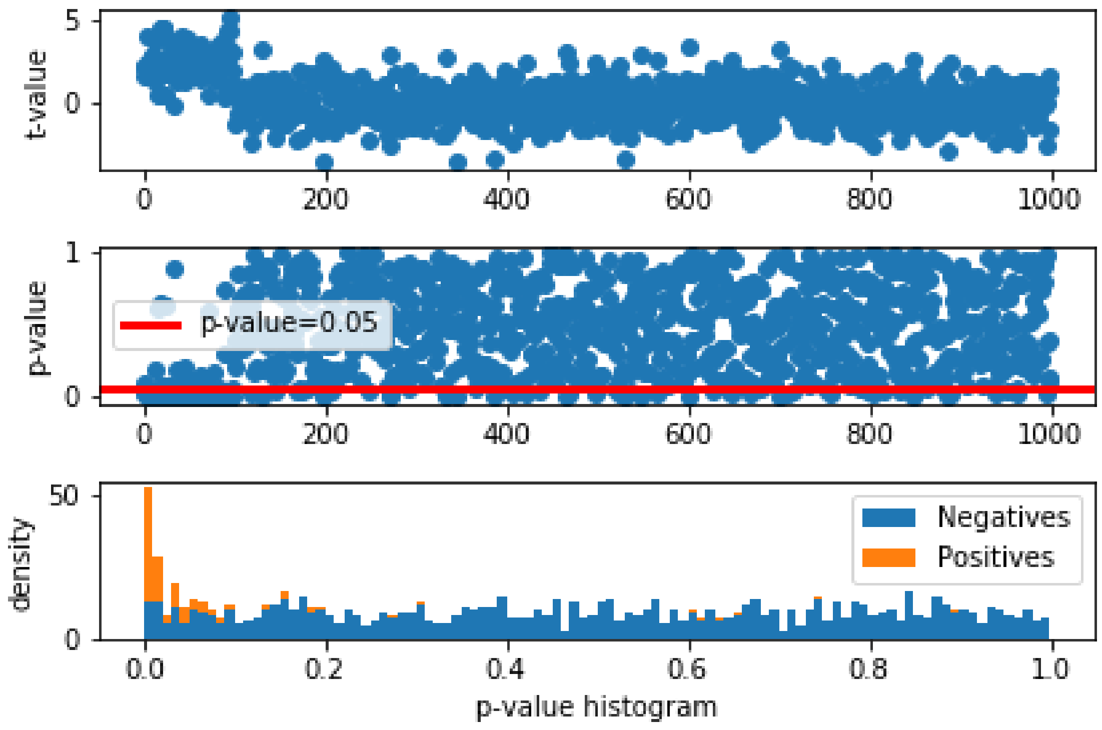
4.1.8 Multiple comparisons

```
import numpy as np
np.random.seed(seed=42)  # make example reproducible

# Dataset
n_samples, n_features = 100, 1000
n_info = int(n_features/10)  # number of features with information
n1, n2 = int(n_samples/2), n_samples - int(n_samples/2)
snr = .5
Y = np.random.randn(n_samples, n_features)
grp = np.array(["g1"] * n1 + ["g2"] * n2)

# Add some group effect for Pinfo features
Y[grp=="g1", :n_info] += snr

#
import scipy.stats as stats
import matplotlib.pyplot as plt
tvals, pvals = np.full(n_features, np.NAN), np.full(n_features, np.NAN)
for j in range(n_features):
    tvals[j], pvals[j] = stats.ttest_ind(Y[grp=="g1", j], Y[grp=="g2", j],
                                          equal_var=True)

fig, axis = plt.subplots(3, 1)#, sharex='col')

axis[0].plot(range(n_features), tvals, 'o')
axis[0].set_ylabel("t-value")
```

(continues on next page)

(continued from previous page)

```
axis[1].plot(range(n_features), pvals, 'o')
axis[1].axhline(y=0.05, color='red', linewidth=3, label="p-value=0.05")
#axis[1].axhline(y=0.05, label="toto", color='red')
axis[1].set_ylabel("p-value")
axis[1].legend()

axis[2].hist([pvals[n_info:], pvals[:n_info]],
    stacked=True, bins=100, label=["Negatives", "Positives"])
axis[2].set_xlabel("p-value histogram")
axis[2].set_ylabel("density")
axis[2].legend()

plt.tight_layout()
```



Note that under the null hypothesis the distribution of the *p*-values is uniform.

Statistical measures:

- **True Positive (TP)** equivalent to a hit. The test correctly concludes the presence of an effect.
- True Negative (TN). The test correctly concludes the absence of an effect.
- **False Positive (FP)** equivalent to a false alarm, **Type I error**. The test improperly concludes the presence of an effect. Thresholding at p-value < 0.05 leads to 47 FP.
- False Negative (FN) equivalent to a miss, Type II error. The test improperly concludes the absence of an effect.

```
P, N = n_info, n_features - n_info  # Positives, Negatives
TP = np.sum(pvals[:n_info ] < 0.05)  # True Positives
FP = np.sum(pvals[n_info: ] < 0.05)  # False Positives
print("No correction, FP: %i (expected: %.2f), TP: %i" % (FP, N * 0.05, TP))
```

Bonferroni correction for multiple comparisons

The Bonferroni correction is based on the idea that if an experimenter is testing P hypotheses, then one way of maintaining the familywise error rate (FWER) is to test each individual hypothesis at a statistical significance level of $1/P$ times the desired maximum overall level.

So, if the desired significance level for the whole family of tests is α (usually 0.05), then the Bonferroni correction would test each individual hypothesis at a significance level of α/P. For example, if a trial is testing $P = 8$ hypotheses with a desired $\alpha = 0.05$, then the Bonferroni correction would test each individual hypothesis at $\alpha = 0.05/8 = 0.00625$.

```
import statsmodels.sandbox.stats.multicomp as multicomp
_, pvals_fwer, _, _  = multicomp.multipletests(pvals, alpha=0.05,
                                               method='bonferroni')
TP = np.sum(pvals_fwer[:n_info ] < 0.05)  # True Positives
FP = np.sum(pvals_fwer[n_info: ] < 0.05)  # False Positives
print("FWER correction, FP: %i, TP: %i" % (FP, TP))
```

The False discovery rate (FDR) correction for multiple comparisons

FDR-controlling procedures are designed to control the expected proportion of rejected null hypotheses that were incorrect rejections (“false discoveries”). FDR-controlling procedures provide less stringent control of Type I errors compared to the familywise error rate (FWER) controlling procedures (such as the Bonferroni correction), which control the probability of at least one Type I error. Thus, FDR-controlling procedures have greater power, at the cost of increased rates of Type I errors.

```
import statsmodels.sandbox.stats.multicomp as multicomp
_, pvals_fdr, _, _  = multicomp.multipletests(pvals, alpha=0.05,
                                               method='fdr_bh')
TP = np.sum(pvals_fdr[:n_info ] < 0.05)  # True Positives
FP = np.sum(pvals_fdr[n_info: ] < 0.05)  # False Positives

print("FDR correction, FP: %i, TP: %i" % (FP, TP))
```

4.1.9 Exercises

Simple linear regression and correlation (application)

Load the dataset: birthwt Risk Factors Associated with Low Infant Birth Weight at https://raw.github.com/neurospin/pystatsml/master/datasets/birthwt.csv

1. Test the association of mother’s (`bwt`) age and birth weight using the correlation test and linear regeression.
2. Test the association of mother’s weight (`lwt`) and birth weight using the correlation test and linear regeression.
3. Produce two scatter plot of: (i) age by birth weight; (ii) mother’s weight by birth weight.

Conclusion ?

Simple linear regression (maths)

Considering the salary and the experience of the salary table. https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv

Compute:

- Estimate the model paramters β, β_0 using scipy stats.linregress(x,y)
- Compute the predicted values $\hat{y}$

Compute:

- $\bar{y}$: y_mu
- SS_tot: ss_tot
- SS_reg: ss_reg
- SS_res: ss_res
- Check partition of variance formula based on sum of squares by using assert np.allclose(val1, val2, atol=1e-05)
- Compute R^2 and compare it with the r_value above
- Compute the F score
- Compute the p-value:
- Plot the $F(1, n)$ distribution for 100 f values within $[10, 25]$. Draw $P(F(1, n) > F)$, i.e. color the surface defined by the x values larger than F below the $F(1, n)$.
- $P(F(1, n) > F)$ is the p-value, compute it.

Multiple regression

Considering the simulated data used below:

1. What are the dimensions of $\text{pinv}(X)$?
2. Compute the MSE between the predicted values and the true values.

```python
import numpy as np
from scipy import linalg
np.random.seed(seed=42)  # make the example reproducible

# Dataset
N, P = 50, 4
X = np.random.normal(size= N * P).reshape((N, P))
## Our model needs an intercept so we add a column of 1s:
X[:, 0] = 1
print(X[:5, :])

betastar = np.array([10, 1., .5, 0.1])
e = np.random.normal(size=N)
y = np.dot(X, betastar) + e

# Estimate the parameters
Xpinv = linalg.pinv2(X)
```

(continues on next page)

(continued from previous page)

```
betahat = np.dot(Xpinv, y)
print("Estimated beta:\n", betahat)
```

Two sample t-test (maths)

Given the following two sample, test whether their means are equals.

```
height = np.array([ 1.83,  1.83,  1.73,  1.82,  1.83,
                    1.73,1.99,  1.85,  1.68,  1.87,
                    1.66,  1.71,  1.73,  1.64,  1.70,
                    1.60,  1.79,  1.73,  1.62,  1.77])
grp = np.array(["M"] * 10 + ["F"] * 10)
```

- Compute the means/std-dev per groups.
- Compute the t-value (standard two sample t-test with equal variances).
- Compute the p-value.
- The p-value is one-sided: a two-sided test would test `P(T > tval)` and `P(T < -tval)`. What would the two sided p-value be?
- Compare the two-sided p-value with the one obtained by `stats.ttest_ind` using `assert np.allclose(arr1, arr2)`.

Two sample t-test (application)

Risk Factors Associated with Low Infant Birth Weight: https://raw.github.com/neurospin/pystatsml/master/datasets/birthwt.csv

1. Explore the data
2. Recode smoke factor
3. Compute the means/std-dev per groups.
4. Plot birth weight by smoking (box plot, violin plot or histogram)
5. Test the effect of smoking on birth weight

Two sample t-test and random permutations

Generate 100 samples following the model:

$$y = g + \varepsilon$$

Where the noise $\varepsilon \sim N(1, 1)$ and $g \in \{0, 1\}$ is a group indicator variable with 50 ones and 50 zeros.

- Write a function `tstat(y, g)` that compute the two samples t-test of y splited in two groups defined by g.
- Sample the t-statistic distribution under the null hypothesis using random permutations.
- Assess the p-value.

Univariate associations (developpement)

Write a function `univar_stat(df, target, variables)` that computes the parametric statistics and p-values between the `target` variable (provided as as string) and all `variables` (provided as a list of string) of the pandas DataFrame `df`. The target is a quantitative variable but variables may be quantitative or qualitative. The function returns a DataFrame with four columns: `variable`, `test`, `value`, `p_value`.

Apply it to the salary dataset available at https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv, with target being `S`: salaries for IT staff in a corporation.

Multiple comparisons

This exercise has 2 goals: apply you knowledge of statistics using vectorized numpy operations. Given the dataset provided for multiple comparisons, compute the two-sample t-test (assuming equal variance) for each (column) feature of the `Y` array given the two groups defined by `grp` variable. You should return two vectors of size `n_features`: one for the t-values and one for the p-values.

ANOVA

Perform an ANOVA dataset described bellow

- Compute between and within variances
- Compute F-value: `fval`
- Compare the p-value with the one obtained by `stats.f_oneway` using `assert np.allclose(arr1, arr2)`

```
# dataset
mu_k = np.array([1, 2, 3])    # means of 3 samples
sd_k = np.array([1, 1, 1])    # sd of 3 samples
n_k = np.array([10, 20, 30])  # sizes of 3 samples
grp = [0, 1, 2]               # group labels
n = np.sum(n_k)
label = np.hstack([[k] * n_k[k] for k in [0, 1, 2]])

y = np.zeros(n)
for k in grp:
    y[label == k] = np.random.normal(mu_k[k], sd_k[k], n_k[k])

# Compute with scipy
fval, pval = stats.f_oneway(y[label == 0], y[label == 1], y[label == 2])
```

Note: Click *here* to download the full example code

4.2 Lab 1: Brain volumes study

The study provides the brain volumes of grey matter (gm), white matter (wm) and cerebrospinal fluid) (csf) of 808 anatomical MRI scans.

4.2.1 Manipulate data

Set the working directory within a directory called "brainvol"

Create 2 subdirectories: *data* that will contain downloaded data and *reports* for results of the analysis.

```
import os
import os.path
import pandas as pd
import tempfile
import urllib.request

WD = os.path.join(tempfile.gettempdir(), "brainvol")
os.makedirs(WD, exist_ok=True)
#os.chdir(WD)

# use cookiecutter file organization
# https://drivendata.github.io/cookiecutter-data-science/
os.makedirs(os.path.join(WD, "data"), exist_ok=True)
#os.makedirs("reports", exist_ok=True)
```

Fetch data

- Demographic data *demo.csv* (columns: *participant_id*, *site*, *group*, *age*, *sex*) and tissue volume data: *group* is Control or Patient. *site* is the recruiting site.
- Gray matter volume *gm.csv* (columns: *participant_id*, *session*, *gm_vol*)
- White matter volume *wm.csv* (columns: *participant_id*, *session*, *wm_vol*)
- Cerebrospinal Fluid *csf.csv* (columns: *participant_id*, *session*, *csf_vol*)

```
base_url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/brain_volumes/%s'
data = dict()
for file in ["demo.csv", "gm.csv", "wm.csv", "csf.csv"]:
    urllib.request.urlretrieve(base_url % file, os.path.join(WD, "data", file))

demo = pd.read_csv(os.path.join(WD, "data", "demo.csv"))
gm = pd.read_csv(os.path.join(WD, "data", "gm.csv"))
wm = pd.read_csv(os.path.join(WD, "data", "wm.csv"))
csf = pd.read_csv(os.path.join(WD, "data", "csf.csv"))

print("tables can be merge using shared columns")
print(gm.head())
```

Out:

```
tables can be merge using shared columns
  participant_id session    gm_vol
0    sub-S1-0002  ses-01  0.672506
```

(continues on next page)

(continued from previous page)

```
1    sub-S1-0002  ses-02  0.678772
2    sub-S1-0002  ses-03  0.665592
3    sub-S1-0004  ses-01  0.890714
4    sub-S1-0004  ses-02  0.881127
```

Merge tables according to *participant_id*

```
brain_vol = pd.merge(pd.merge(pd.merge(demo, gm), wm), csf)
assert brain_vol.shape == (808, 9)
```

Drop rows with missing values

```
brain_vol = brain_vol.dropna()
assert brain_vol.shape == (766, 9)
```

Compute Total Intra-cranial volume $tiv\_vol = gm\_vol + csf\_vol + wm\_vol$.

```
brain_vol["tiv_vol"] = brain_vol["gm_vol"] + brain_vol["wm_vol"] + brain_vol["csf_vol"]
```

Compute tissue fractions $gm\_f = gm\_vol / tiv\_vol$, $wm\_f = wm\_vol / tiv\_vol$.

```
brain_vol["gm_f"] = brain_vol["gm_vol"] / brain_vol["tiv_vol"]
brain_vol["wm_f"] = brain_vol["wm_vol"] / brain_vol["tiv_vol"]
```

Save in a excel file *brain_vol.xlsx*

```
brain_vol.to_excel(os.path.join(WD, "data", "brain_vol.xlsx"),
                   sheet_name='data', index=False)
```

4.2.2 Descriptive Statistics

Load excel file *brain_vol.xlsx*

```
import os
import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smfrmla
import statsmodels.api as sm

brain_vol = pd.read_excel(os.path.join(WD, "data", "brain_vol.xlsx"),
                          sheet_name='data')
# Round float at 2 decimals when printing
pd.options.display.float_format = '{:,.2f}'.format
```

Descriptive statistics Most of participants have several MRI sessions (column *session*) Select on rows from session one "ses-01"

```
brain_vol1 = brain_vol[brain_vol.session == "ses-01"]
# Check that there are no duplicates
assert len(brain_vol1.participant_id.unique()) == len(brain_vol1.participant_id)
```

Global descriptives statistics of numerical variables

```
desc_glob_num = brain_vol1.describe()
print(desc_glob_num)
```

Out:

```
          age  gm_vol  wm_vol  csf_vol  tiv_vol   gm_f   wm_f
count 244.00  244.00  244.00   244.00   244.00 244.00 244.00
mean   34.54    0.71    0.44     0.31     1.46   0.49   0.30
std    12.09    0.08    0.07     0.08     0.17   0.04   0.03
min    18.00    0.48    0.05     0.12     0.83   0.37   0.06
25%    25.00    0.66    0.40     0.25     1.34   0.46   0.28
50%    31.00    0.70    0.43     0.30     1.45   0.49   0.30
75%    44.00    0.77    0.48     0.37     1.57   0.52   0.31
max    61.00    1.03    0.62     0.63     2.06   0.60   0.36
```

Global Descriptive statistics of categorical variable

```
desc_glob_cat = brain_vol1[["site", "group", "sex"]].describe(include='all')
print(desc_glob_cat)

print("Get count by level")
desc_glob_cat = pd.DataFrame({col: brain_vol1[col].value_counts().to_dict()
                             for col in ["site", "group", "sex"]})
print(desc_glob_cat)
```

Out:

```
       site    group  sex
count   244      244  244
unique    7        2    2
top      S7  Patient    M
freq     65      157  155
Get count by level
          site  group    sex
Control    nan  87.00    nan
F          nan    nan  89.00
M          nan    nan 155.00
Patient    nan 157.00    nan
S1       13.00    nan    nan
S3       29.00    nan    nan
S4       15.00    nan    nan
S5       62.00    nan    nan
S6        1.00    nan    nan
S7       65.00    nan    nan
S8       59.00    nan    nan
```

Remove the single participant from site 6

```
brain_vol = brain_vol[brain_vol.site != "S6"]
brain_vol1 = brain_vol[brain_vol.session == "ses-01"]
desc_glob_cat = pd.DataFrame({col: brain_vol1[col].value_counts().to_dict()
                             for col in ["site", "group", "sex"]})
print(desc_glob_cat)
```

Out:

```
         site  group    sex
Control   nan  86.00    nan
F         nan    nan  88.00
M         nan    nan 155.00
Patient   nan 157.00    nan
S1      13.00    nan    nan
S3      29.00    nan    nan
S4      15.00    nan    nan
S5      62.00    nan    nan
S7      65.00    nan    nan
S8      59.00    nan    nan
```

Descriptives statistics of numerical variables per clinical status

```
desc_group_num = brain_vol1[["group", 'gm_vol']].groupby("group").describe()
print(desc_group_num)
```

Out:

```
        gm_vol
         count mean  std  min  25%  50%  75%  max
group
Control  86.00 0.72 0.09 0.48 0.66 0.71 0.78 1.03
Patient 157.00 0.70 0.08 0.53 0.65 0.70 0.76 0.90
```

4.2.3 Statistics

Objectives:

1. Site effect of gray matter atrophy
2. Test the association between the age and gray matter atrophy in the control and patient population independently.
3. Test for differences of atrophy between the patients and the controls
4. Test for interaction between age and clinical status, ie: is the brain atrophy process in patient population faster than in the control population.
5. The effect of the medication in the patient population.

```
import statsmodels.api as sm
import statsmodels.formula.api as smfrmla
import scipy.stats
import seaborn as sns
```

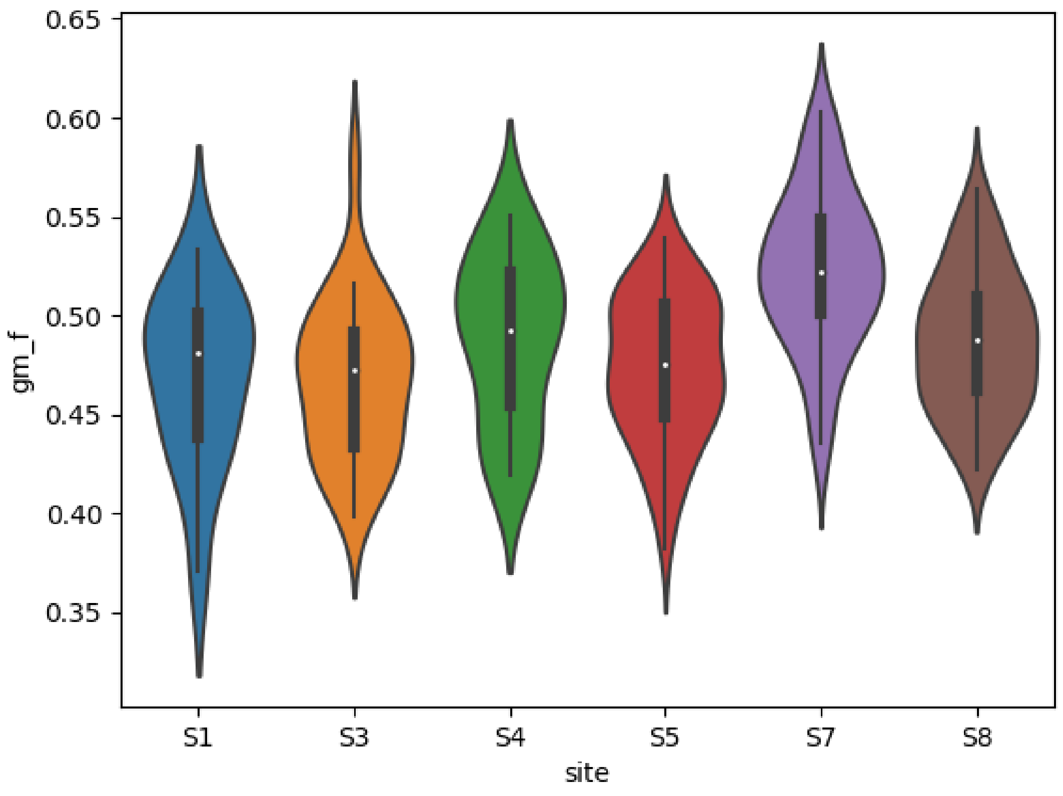
1 Site effect on Grey Matter atrophy

The model is Oneway Anova gm_f ~ site The ANOVA test has important assumptions that must be satisfied in order for the associated p-value to be valid.

- The samples are independent.
- Each sample is from a normally distributed population.
- The population standard deviations of the groups are all equal. This property is known as homoscedasticity.

Plot

```
sns.violinplot("site", "gm_f", data=brain_vol1)
```



Stats with scipy

```
fstat, pval = scipy.stats.f_oneway(*[brain_vol1.gm_f[brain_vol1.site == s]
                                   for s in brain_vol1.site.unique()])
print("Oneway Anova gm_f ~ site F=%.2f, p-value=%E" % (fstat, pval))
```

Out:

```
Oneway Anova gm_f ~ site F=14.82, p-value=1.188136E-12
```

Stats with statsmodels

```
anova = smfrmla.ols("gm_f ~ site", data=brain_vol1).fit()
# print(anova.summary())
print("Site explains %.2f%% of the grey matter fraction variance" %
      (anova.rsquared * 100))

print(sm.stats.anova_lm(anova, typ=2))
```

Out:

```
Site explains 23.82% of the grey matter fraction variance
          sum_sq     df      F   PR(>F)
```

(continues on next page)

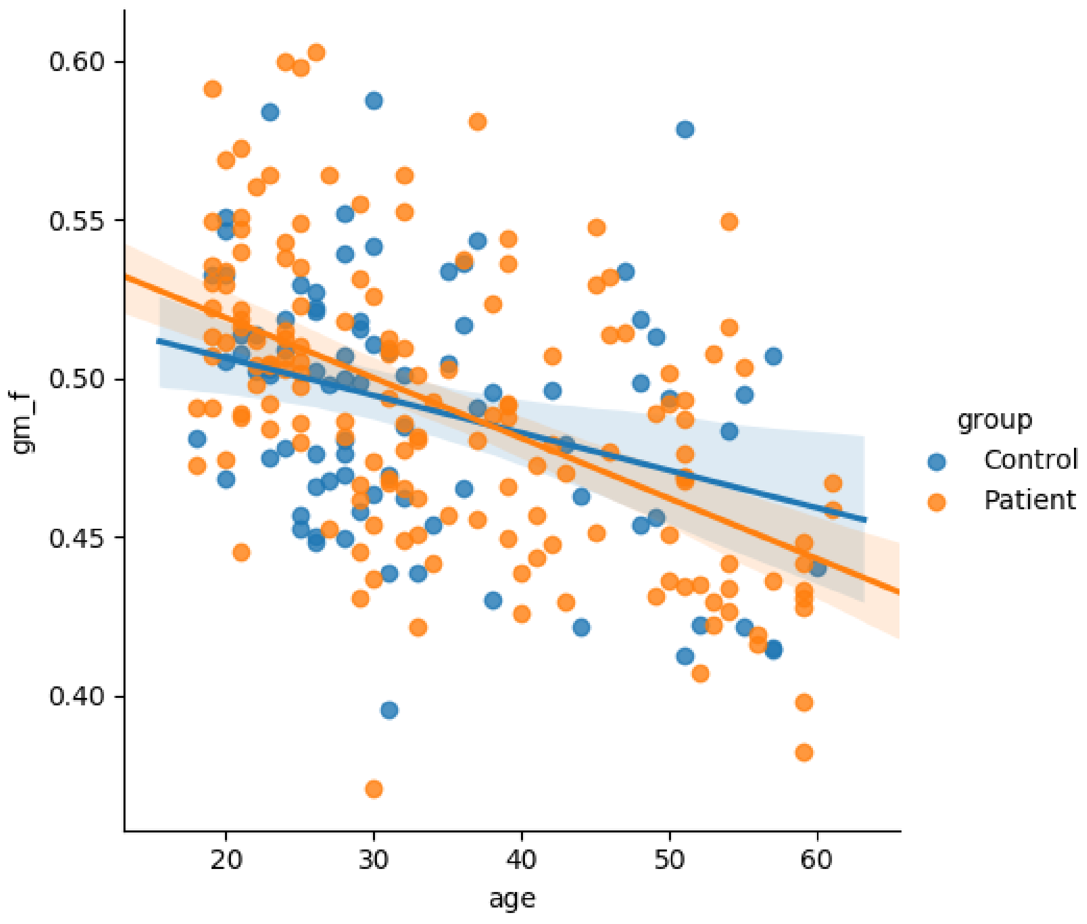
(continued from previous page)

```
site        0.11   5.00 14.82    0.00
Residual    0.35 237.00    nan     nan
```

2. Test the association between the age and gray matter atrophy in the control and patient population independently.

Plot

```
sns.lmplot("age", "gm_f", hue="group", data=brain_vol1)

brain_vol1_ctl = brain_vol1[brain_vol1.group == "Control"]
brain_vol1_pat = brain_vol1[brain_vol1.group == "Patient"]
```



Stats with scipy

```
print("--- In control population ---")
beta, beta0, r_value, p_value, std_err = \
    scipy.stats.linregress(x=brain_vol1_ctl.age, y=brain_vol1_ctl.gm_f)

print("gm_f = %f * age + %f" % (beta, beta0))
print("Corr: %f, r-squared: %f, p-value: %f, std_err: %f"\
      % (r_value, r_value**2, p_value, std_err))

print("--- In patient population ---")
beta, beta0, r_value, p_value, std_err = \
```

(continues on next page)

(continued from previous page)

```
    scipy.stats.linregress(x=brain_vol1_pat.age, y=brain_vol1_pat.gm_f)

print("gm_f = %f * age + %f" % (beta, beta0))
print("Corr: %f, r-squared: %f, p-value: %f, std_err: %f"\
      % (r_value, r_value**2, p_value, std_err))

print("Decrease seems faster in patient than in control population")
```

Out:

```
--- In control population ---
gm_f = -0.001181 * age + 0.529829
Corr: -0.325122, r-squared: 0.105704, p-value: 0.002255, std_err: 0.000375
--- In patient population ---
gm_f = -0.001899 * age + 0.556886
Corr: -0.528765, r-squared: 0.279592, p-value: 0.000000, std_err: 0.000245
Decrease seems faster in patient than in control population
```

Stats with statsmodels

```
print("--- In control population ---")
lr = smfrmla.ols("gm_f ~ age", data=brain_vol1_ctl).fit()
print(lr.summary())
print("Age explains %.2f%% of the grey matter fraction variance" %
      (lr.rsquared * 100))

print("--- In patient population ---")
lr = smfrmla.ols("gm_f ~ age", data=brain_vol1_pat).fit()
print(lr.summary())
print("Age explains %.2f%% of the grey matter fraction variance" %
      (lr.rsquared * 100))
```

Out:

```
--- In control population ---
                            OLS Regression Results
==============================================================================
Dep. Variable:                   gm_f   R-squared:                       0.106
Model:                            OLS   Adj. R-squared:                  0.095
Method:                 Least Squares   F-statistic:                     9.929
Date:              jeu., 31 oct. 2019   Prob (F-statistic):            0.00226
Time:                        16:09:40   Log-Likelihood:                 159.34
No. Observations:                  86   AIC:                            -314.7
Df Residuals:                      84   BIC:                            -309.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5298      0.013     40.350      0.000       0.504       0.556
age           -0.0012      0.000     -3.151      0.002      -0.002      -0.000
==============================================================================
Omnibus:                        0.946   Durbin-Watson:                   1.628
Prob(Omnibus):                  0.623   Jarque-Bera (JB):                0.782
Skew:                           0.233   Prob(JB):                        0.676
```

(continues on next page)

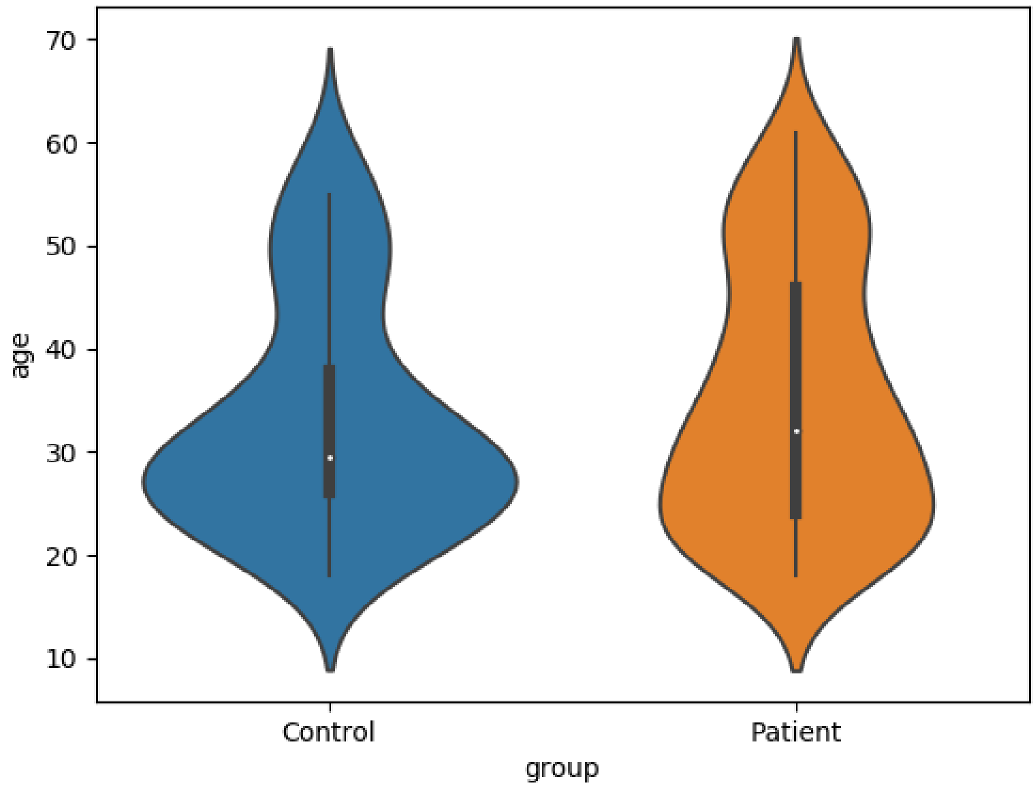
(continued from previous page)

```
Kurtosis:                       2.962   Cond. No.                         111.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
Age explains 10.57% of the grey matter fraction variance
--- In patient population ---
                            OLS Regression Results
==============================================================================
Dep. Variable:                   gm_f   R-squared:                       0.280
Model:                            OLS   Adj. R-squared:                  0.275
Method:                 Least Squares   F-statistic:                     60.16
Date:                jeu., 31 oct. 2019   Prob (F-statistic):           1.09e-12
Time:                        16:09:40   Log-Likelihood:                 289.38
No. Observations:                 157   AIC:                            -574.8
Df Residuals:                     155   BIC:                            -568.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5569      0.009     60.817      0.000       0.539       0.575
age           -0.0019      0.000     -7.756      0.000      -0.002      -0.001
==============================================================================
Omnibus:                        2.310   Durbin-Watson:                   1.325
Prob(Omnibus):                  0.315   Jarque-Bera (JB):                1.854
Skew:                           0.230   Prob(JB):                        0.396
Kurtosis:                       3.268   Cond. No.                         111.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
Age explains 27.96% of the grey matter fraction variance
```

Before testing for differences of atrophy between the patients ans the controls **Preliminary tests for age x group effect** (patients would be older or younger than Controls)

Plot

```
sns.violinplot("group", "age", data=brain_vol1)
```



Stats with scipy

```
print(scipy.stats.ttest_ind(brain_vol1_ctl.age, brain_vol1_pat.age))
```

Out:

```
Ttest_indResult(statistic=-1.2155557697674162, pvalue=0.225343592508479)
```

Stats with statsmodels

```
print(smfrmla.ols("age ~ group", data=brain_vol1).fit().summary())
print("No significant difference in age between patients and controls")
```

Out:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    age   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     1.478
Date:                jeu., 31 oct. 2019   Prob (F-statistic):              0.225
Time:                        16:09:40   Log-Likelihood:                -949.69
No. Observations:                 243   AIC:                             1903.
Df Residuals:                     241   BIC:                             1910.
Df Model:                           1
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------------------
Intercept           33.2558      1.305     25.484      0.000      30.685      35.826
group[T.Patient]     1.9735      1.624      1.216      0.225      -1.225       5.172
==============================================================================
Omnibus:                       35.711   Durbin-Watson:                   2.096
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.726
Skew:                           0.569   Prob(JB):                     3.16e-05
Kurtosis:                       2.133   Cond. No.                         3.12
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly␣
↪specified.
No significant difference in age between patients and controls
```

Preliminary tests for sex x group (more/less males in patients than in Controls)

```
crosstab = pd.crosstab(brain_vol1.sex, brain_vol1.group)
print("Obeserved contingency table")
print(crosstab)

chi2, pval, dof, expected = scipy.stats.chi2_contingency(crosstab)

print("Chi2 = %f, pval = %f" % (chi2, pval))
print("Expected contingency table under the null hypothesis")
print(expected)
print("No significant difference in sex between patients and controls")
```

Out:

```
Obeserved contingency table
group  Control  Patient
sex
F            33       55
M            53      102
Chi2 = 0.143253, pval = 0.705068
Expected contingency table under the null hypothesis
[[ 31.14403292  56.85596708]
 [ 54.85596708 100.14403292]]
No significant difference in sex between patients and controls
```

3. Test for differences of atrophy between the patients and the controls

```
print(sm.stats.anova_lm(smfrmla.ols("gm_f ~ group", data=brain_vol1).fit(), typ=2))
print("No significant difference in age between patients and controls")
```

Out:

```
          sum_sq     df    F  PR(>F)
group       0.00   1.00 0.01    0.92
Residual    0.46 241.00  nan     nan
No significant difference in age between patients and controls
```

This model is simplistic we should adjust for age and site

```
print(sm.stats.anova_lm(smfrmla.ols(
        "gm_f ~ group + age + site", data=brain_vol1).fit(), typ=2))
print("No significant difference in age between patients and controls")
```

Out:

```
          sum_sq     df     F  PR(>F)
group       0.00   1.00  1.82    0.18
site        0.11   5.00 19.79    0.00
age         0.09   1.00 86.86    0.00
Residual    0.25 235.00   nan     nan
No significant difference in age between patients and controls
```

4. **Test for interaction between age and clinical status, ie: is the brain** atrophy process in patient population faster than in the control population.

```
ancova = smfrmla.ols("gm_f ~ group:age + age + site", data=brain_vol1).fit()
print(sm.stats.anova_lm(ancova, typ=2))

print("= Parameters =")
print(ancova.params)

print("%.3f%% of grey matter loss per year (almost %.1f%% per decade)" %\
      (ancova.params.age * 100, ancova.params.age * 100 * 10))

print("grey matter loss in patients is accelerated by %.3f%% per decade" %
      (ancova.params['group[T.Patient]:age'] * 100 * 10))
```

Out:

```
           sum_sq     df     F  PR(>F)
site         0.11   5.00 20.28    0.00
age          0.10   1.00 89.37    0.00
group:age    0.00   1.00  3.28    0.07
Residual     0.25 235.00   nan     nan
= Parameters =
Intercept              0.52
site[T.S3]             0.01
site[T.S4]             0.03
site[T.S5]             0.01
site[T.S7]             0.06
site[T.S8]             0.02
age                   -0.00
group[T.Patient]:age  -0.00
dtype: float64
-0.148% of grey matter loss per year (almost -1.5% per decade)
grey matter loss in patients is accelerated by -0.232% per decade
```

Total running time of the script: (0 minutes 4.267 seconds)

4.3 Multivariate statistics

Multivariate statistics includes all statistical techniques for analyzing samples made of two or more variables. The data set (a $N \times P$ matrix $\mathbf{X}$) is a collection of N independent samples column **vectors** $[\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_N]$ of length P

$$\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1^T- \\ \vdots \\ -\mathbf{x}_i^T- \\ \vdots \\ -\mathbf{x}_P^T- \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1P} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{iP} \\ \vdots & & \vdots & & \vdots \\ x_{N1} & \cdots & x_{Nj} & \cdots & x_{NP} \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1P} \\ \vdots & & \vdots \\ & \mathbf{X} & \\ \vdots & & \vdots \\ x_{N1} & \dots & x_{NP} \end{bmatrix}_{N \times P}.$$

4.3.1 Linear Algebra

Euclidean norm and distance

The Euclidean norm of a vector $\mathbf{a} \in \mathbb{R}^P$ is denoted

$$\|\mathbf{a}\|_2 = \sqrt{\sum_i^P {a_i}^2}$$

The Euclidean distance between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^P$ is

$$\|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_i^P (a_i - b_i)^2}$$

Dot product and projection

Source: Wikipedia

Algebraic definition

The dot product, denoted "·" of two P-dimensional vectors $\mathbf{a} = [a_1, a_2, ..., a_P]$ and $\mathbf{a} = [b_1, b_2, ..., b_P]$ is defined as

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \sum_i a_i b_i = \begin{bmatrix} a_1 & \dots & \mathbf{a}^T & \dots & a_P \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ \mathbf{b} \\ \vdots \\ b_P \end{bmatrix}.$$

The Euclidean norm of a vector can be computed using the dot product, as

$$\left\|\mathbf{a}\right\|_2 = \sqrt{\mathbf{a} \cdot \mathbf{a}}.$$

Geometric definition: projection

In Euclidean space, a Euclidean vector is a geometrical object that possesses both a magnitude and a direction. A vector can be pictured as an arrow. Its magnitude is its length, and its direction is the direction that the arrow points. The magnitude of a vector $\mathbf{a}$ is denoted by $\|\mathbf{a}\|_2$. The dot product of two Euclidean vectors $\mathbf{a}$ and $\mathbf{b}$ is defined by

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \ \|\mathbf{b}\|_2 \cos\theta,$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$.

In particular, if $\mathbf{a}$ and $\mathbf{b}$ are orthogonal, then the angle between them is 90° and

$$\mathbf{a} \cdot \mathbf{b} = 0.$$

At the other extreme, if they are codirectional, then the angle between them is 0° and

$$\mathbf{a} \cdot \mathbf{b} = \left\|\mathbf{a}\right\|_2 \left\|\mathbf{b}\right\|_2$$

This implies that the dot product of a vector $\mathbf{a}$ by itself is

$$\mathbf{a} \cdot \mathbf{a} = \left\|\mathbf{a}\right\|_2^2.$$

The scalar projection (or scalar component) of a Euclidean vector $\mathbf{a}$ in the direction of a Euclidean vector $\mathbf{b}$ is given by

$$a_b = \left\|\mathbf{a}\right\|_2 \cos\theta,$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$.

In terms of the geometric definition of the dot product, this can be rewritten

$$a_b = \frac{\mathbf{a} \cdot \mathbf{b}}{\left\|\mathbf{b}\right\|_2},$$

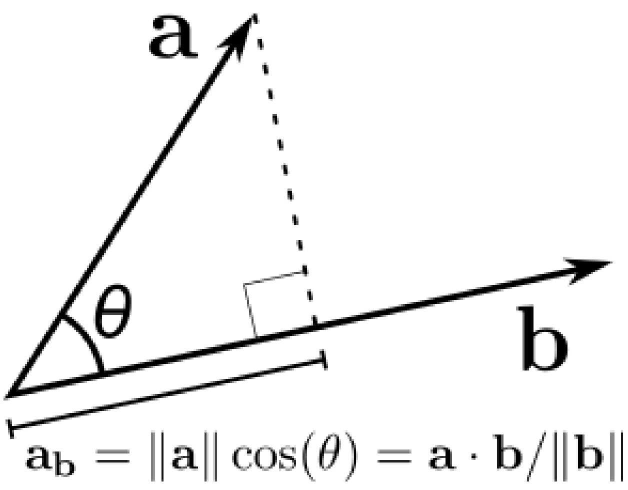

a
θ
b
$\mathbf{a_b} = \|\mathbf{a}\| \cos(\theta) = \mathbf{a} \cdot \mathbf{b}/\|\mathbf{b}\|$

Fig. 5: Projection.

```
import numpy as np
np.random.seed(42)

a = np.random.randn(10)
b = np.random.randn(10)

np.dot(a, b)
```

```
-4.085788532659924
```

4.3.2 Mean vector

The mean $(P \times 1)$ column-vector μ whose estimator is

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x_i} = \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} x_{i1} \\ \vdots \\ x_{ij} \\ \vdots \\ x_{iP} \end{bmatrix} = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_j \\ \vdots \\ \bar{x}_P \end{bmatrix}.$$

4.3.3 Covariance matrix

- The covariance matrix $\mathbf{\Sigma_{XX}}$ is a **symmetric** positive semi-definite matrix whose element in the j, k position is the covariance between the j^{th} and k^{th} elements of a random vector i.e. the j^{th} and k^{th} columns of $\mathbf{X}$.
- The covariance matrix generalizes the notion of covariance to multiple dimensions.
- The covariance matrix describe the shape of the sample distribution around the mean assuming an elliptical distribution:

$$\mathbf{\Sigma_{XX}} = E(\mathbf{X} - E(\mathbf{X}))^T E(\mathbf{X} - E(\mathbf{X})),$$

whose estimator $\mathbf{S_{XX}}$ is a $P \times P$ matrix given by

$$\mathbf{S_{XX}} = \frac{1}{N-1}(\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T)^T(\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T).$$

If we assume that $\mathbf{X}$ is centered, i.e. $\mathbf{X}$ is replaced by $\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T$ then the estimator is

$$\mathbf{S_{XX}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X} = \frac{1}{N-1}\begin{bmatrix} x_{11} & \cdots & x_{N1} \\ x_{1j} & \cdots & x_{Nj} \\ \vdots & & \vdots \\ x_{1P} & \cdots & x_{NP} \end{bmatrix}\begin{bmatrix} x_{11} & \cdots & x_{1k} & x_{1P} \\ \vdots & & \vdots & \vdots \\ x_{N1} & \cdots & x_{Nk} & x_{NP} \end{bmatrix} = \begin{bmatrix} s_1 & \ldots & s_{1k} & s_{1P} \\ & \ddots & s_{jk} & \vdots \\ & & s_k & s_{kP} \\ & & & s_P \end{bmatrix},$$

where

$$s_{jk} = s_{kj} = \frac{1}{N-1}\mathbf{x_j}^T\mathbf{x_k} = \frac{1}{N-1}\sum_{i=1}^{N} x_{ij}x_{ik}$$

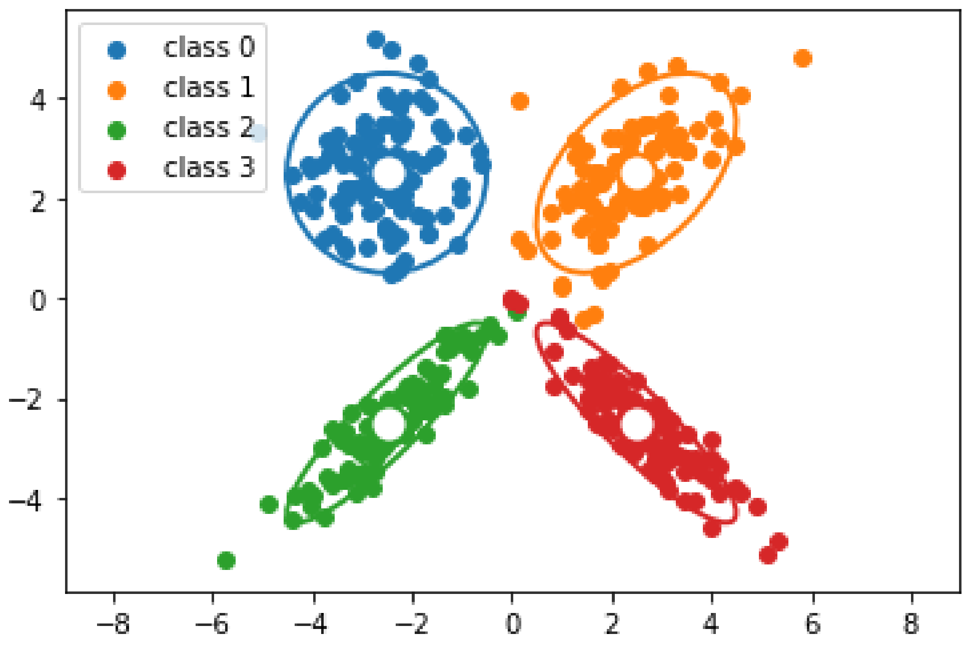
is an estimator of the covariance between the j^{th} and k^{th} variables.

```
## Avoid warnings and force inline plot
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
##
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import pystatsml.plot_utils
import seaborn as sns  # nice color
```

(continues on next page)

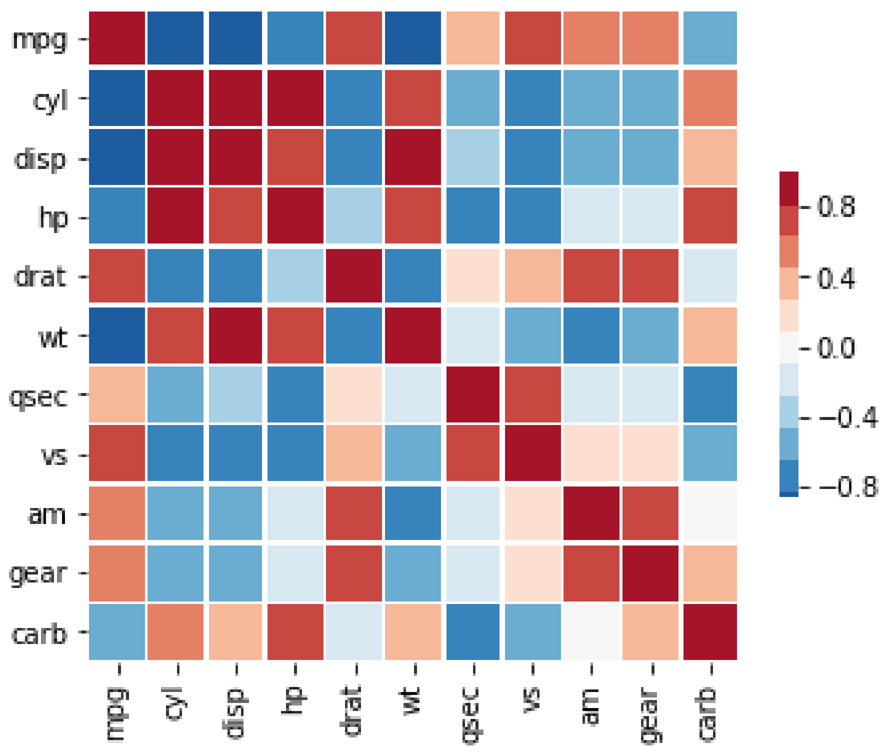
(continued from previous page)

```
np.random.seed(42)
colors = sns.color_palette()

n_samples, n_features = 100, 2

mean, Cov, X = [None] * 4, [None] * 4, [None] * 4
mean[0] = np.array([-2.5, 2.5])
Cov[0] = np.array([[1, 0],
                   [0, 1]])

mean[1] = np.array([2.5, 2.5])
Cov[1] = np.array([[1, .5],
                   [.5, 1]])

mean[2] = np.array([-2.5, -2.5])
Cov[2] = np.array([[1, .9],
                   [.9, 1]])

mean[3] = np.array([2.5, -2.5])
Cov[3] = np.array([[1, -.9],
                   [-.9, 1]])

# Generate dataset
for i in range(len(mean)):
    X[i] = np.random.multivariate_normal(mean[i], Cov[i], n_samples)

# Plot
for i in range(len(mean)):
    # Points
    plt.scatter(X[i][:, 0], X[i][:, 1], color=colors[i], label="class %i" % i)
    # Means
    plt.scatter(mean[i][0], mean[i][1], marker="o", s=200, facecolors='w',
                edgecolors=colors[i], linewidth=2)
    # Ellipses representing the covariance matrices
    pystatsml.plot_utils.plot_cov_ellipse(Cov[i], pos=mean[i], facecolor='none',
                                          linewidth=2, edgecolor=colors[i])

plt.axis('equal')
_ = plt.legend(loc='upper left')
```



4.3.4 Correlation matrix

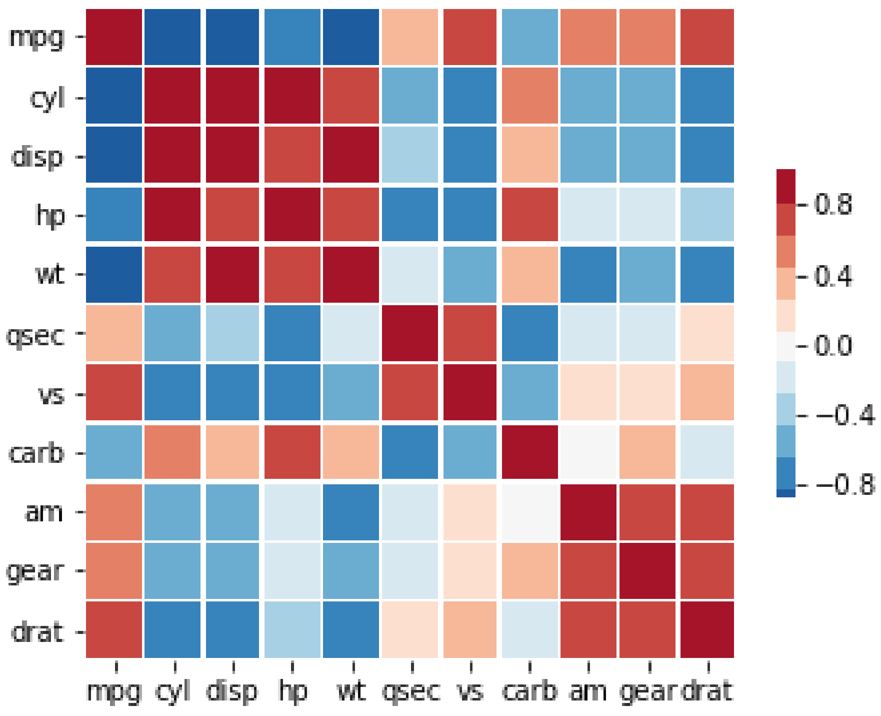
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

url = 'https://python-graph-gallery.com/wp-content/uploads/mtcars.csv'
df = pd.read_csv(url)

# Compute the correlation matrix
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(5.5, 4.5))
cmap = sns.color_palette("RdBu_r", 11)
# Draw the heatmap with the mask and correct aspect ratio
_ = sns.heatmap(corr, mask=None, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



Re-order correlation matrix using AgglomerativeClustering

```
# convert correlation to distances
d = 2 * (1 - np.abs(corr))

from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=3, linkage='single', affinity="precomputed
↪").fit(d)
lab=0

clusters = [list(corr.columns[clustering.labels_==lab]) for lab in set(clustering.labels_
↪)]
print(clusters)

reordered = np.concatenate(clusters)

R = corr.loc[reordered, reordered]

f, ax = plt.subplots(figsize=(5.5, 4.5))
# Draw the heatmap with the mask and correct aspect ratio
_ = sns.heatmap(R, mask=None, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
[['mpg', 'cyl', 'disp', 'hp', 'wt', 'qsec', 'vs', 'carb'], ['am', 'gear'], ['drat']]
```



4.3.5 Precision matrix

In statistics, precision is the reciprocal of the variance, and the precision matrix is the matrix inverse of the covariance matrix.

It is related to **partial correlations** that measures the degree of association between two variables, while controlling the effect of other variables.

```
import numpy as np

Cov = np.array([[1.0, 0.9, 0.9, 0.0, 0.0, 0.0],
                [0.9, 1.0, 0.9, 0.0, 0.0, 0.0],
                [0.9, 0.9, 1.0, 0.0, 0.0, 0.0],
                [0.0, 0.0, 0.0, 1.0, 0.9, 0.0],
                [0.0, 0.0, 0.0, 0.9, 1.0, 0.0],
                [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]])

print("# Precision matrix:")
Prec = np.linalg.inv(Cov)
print(Prec.round(2))

print("# Partial correlations:")
Pcor = np.zeros(Prec.shape)
Pcor[::] = np.NaN

for i, j in zip(*np.triu_indices_from(Prec, 1)):
    Pcor[i, j] = - Prec[i, j] / np.sqrt(Prec[i, i] * Prec[j, j])

print(Pcor.round(2))
```

```
# Precision matrix:
[[ 6.79 -3.21 -3.21  0.    0.    0.  ]
 [-3.21  6.79 -3.21  0.    0.    0.  ]
 [-3.21 -3.21  6.79  0.    0.    0.  ]
 [ 0.   -0.   -0.    5.26 -4.74 -0.  ]
 [ 0.    0.    0.   -4.74  5.26  0.  ]
 [ 0.    0.    0.    0.    0.    1.  ]]
# Partial correlations:
[[  nan  0.47  0.47 -0.   -0.   -0.  ]
 [  nan   nan  0.47 -0.   -0.   -0.  ]
 [  nan   nan   nan -0.   -0.   -0.  ]
 [  nan   nan   nan   nan  0.9   0.  ]
 [  nan   nan   nan   nan   nan -0.  ]
 [  nan   nan   nan   nan   nan   nan]]
```

4.3.6 Mahalanobis distance

- The Mahalanobis distance is a measure of the distance between two points $\mathbf{x}$ and μ where the dispersion (i.e. the covariance structure) of the samples is taken into account.
- The dispersion is considered through covariance matrix.

This is formally expressed as

$$D_M(\mathbf{x}, \mu) = \sqrt{(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu)}.$$

Intuitions

- Distances along the principal directions of dispersion are contracted since they correspond to likely dispersion of points.
- Distances othogonal to the principal directions of dispersion are dilated since they correspond to unlikely dispersion of points.

For example

$$D_M(\mathbf{1}) = \sqrt{\mathbf{1}^T \mathbf{\Sigma}^{-1} \mathbf{1}}.$$

```
ones  = np.ones(Cov.shape[0])
d_euc = np.sqrt(np.dot(ones, ones))
d_mah = np.sqrt(np.dot(np.dot(ones, Prec), ones))

print("Euclidean norm of ones=%.2f. Mahalanobis norm of ones=%.2f" % (d_euc, d_mah))
```

```
Euclidean norm of ones=2.45. Mahalanobis norm of ones=1.77
```

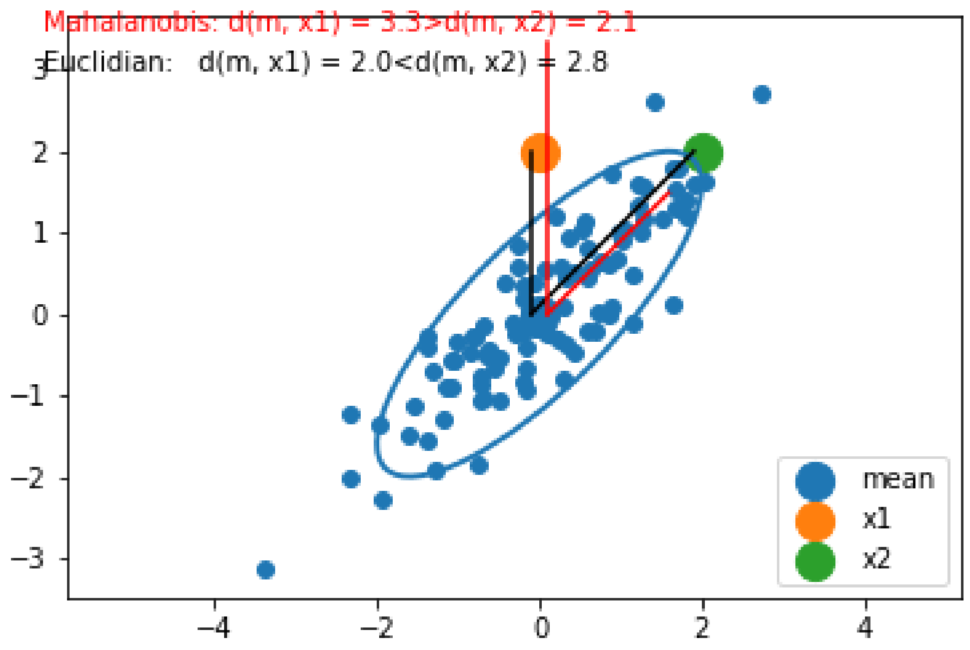
The first dot product that distances along the principal directions of dispersion are contracted:

```
print(np.dot(ones, Prec))
```

```
[0.35714286 0.35714286 0.35714286 0.52631579 0.52631579 1.        ]
```

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import pystatsml.plot_utils
%matplotlib inline
np.random.seed(40)
colors = sns.color_palette()

mean = np.array([0, 0])
Cov = np.array([[1, .8],
                [.8, 1]])
samples = np.random.multivariate_normal(mean, Cov, 100)
x1 = np.array([0, 2])
x2 = np.array([2, 2])

plt.scatter(samples[:, 0], samples[:, 1], color=colors[0])
plt.scatter(mean[0], mean[1], color=colors[0], s=200, label="mean")
plt.scatter(x1[0], x1[1], color=colors[1], s=200, label="x1")
plt.scatter(x2[0], x2[1], color=colors[2], s=200, label="x2")

# plot covariance ellipsis
pystatsml.plot_utils.plot_cov_ellipse(Cov, pos=mean, facecolor='none',
                                      linewidth=2, edgecolor=colors[0])
# Compute distances
d2_m_x1 = scipy.spatial.distance.euclidean(mean, x1)
d2_m_x2 = scipy.spatial.distance.euclidean(mean, x2)

Covi = scipy.linalg.inv(Cov)
dm_m_x1 = scipy.spatial.distance.mahalanobis(mean, x1, Covi)
dm_m_x2 = scipy.spatial.distance.mahalanobis(mean, x2, Covi)

# Plot distances
vm_x1 = (x1 - mean) / d2_m_x1
vm_x2 = (x2 - mean) / d2_m_x2
jitter = .1
plt.plot([mean[0] - jitter, d2_m_x1 * vm_x1[0] - jitter],
         [mean[1], d2_m_x1 * vm_x1[1]], color='k')
plt.plot([mean[0] - jitter, d2_m_x2 * vm_x2[0] - jitter],
         [mean[1], d2_m_x2 * vm_x2[1]], color='k')

plt.plot([mean[0] + jitter, dm_m_x1 * vm_x1[0] + jitter],
         [mean[1], dm_m_x1 * vm_x1[1]], color='r')
plt.plot([mean[0] + jitter, dm_m_x2 * vm_x2[0] + jitter],
         [mean[1], dm_m_x2 * vm_x2[1]], color='r')

plt.legend(loc='lower right')
plt.text(-6.1, 3,
         'Euclidian:   d(m, x1) = %.1f<d(m, x2) = %.1f' % (d2_m_x1, d2_m_x2), color='k')
plt.text(-6.1, 3.5,
         'Mahalanobis: d(m, x1) = %.1f>d(m, x2) = %.1f' % (dm_m_x1, dm_m_x2), color='r')

plt.axis('equal')
print('Euclidian   d(m, x1) = %.2f < d(m, x2) = %.2f' % (d2_m_x1, d2_m_x2))
print('Mahalanobis d(m, x1) = %.2f > d(m, x2) = %.2f' % (dm_m_x1, dm_m_x2))
```

```
Euclidian   d(m, x1) = 2.00 < d(m, x2) = 2.83
Mahalanobis d(m, x1) = 3.33 > d(m, x2) = 2.11
```



If the covariance matrix is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance. If the covariance matrix is diagonal, then the resulting distance measure is called a normalized Euclidean distance.

More generally, the Mahalanobis distance is a measure of the distance between a point x and a distribution $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$. It is a multi-dimensional generalization of the idea of measuring how many standard deviations away x is from the mean. This distance is zero if x is at the mean, and grows as x moves away from the mean: along each principal component axis, it measures the number of standard deviations from x to the mean of the distribution.

4.3.7 Multivariate normal distribution

The distribution, or probability density function (PDF) (sometimes just density), of a continuous random variable is a function that describes the relative likelihood for this random variable to take on a given value.

The multivariate normal distribution, or multivariate Gaussian distribution, of a P-dimensional random vector $\mathbf{x} = [x_1, x_2, \ldots, x_P]^T$ is

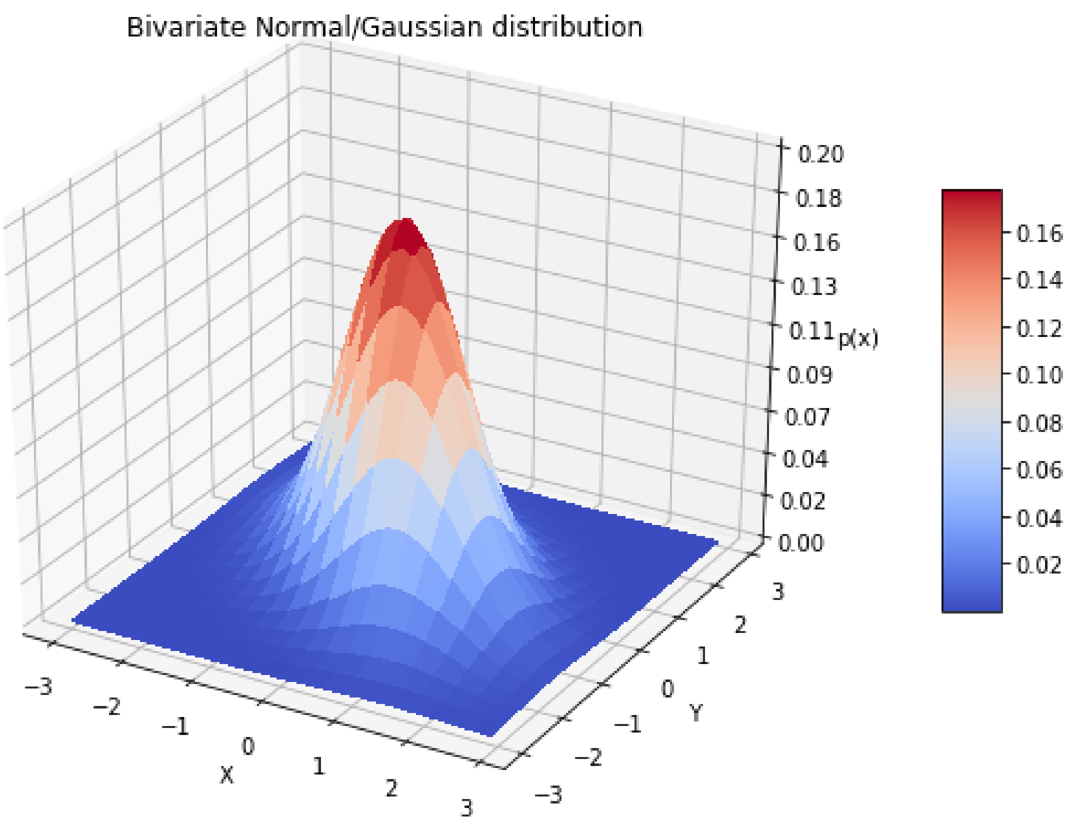
$$\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{P/2}|\mathbf{\Sigma}|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\}.$$

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
```

(continues on next page)

(continued from previous page)

```
def multivariate_normal_pdf(X, mean, sigma):
    """Multivariate normal probability density function over X (n_samples x n_features)"""
    P = X.shape[1]
    det = np.linalg.det(sigma)
    norm_const = 1.0 / (((2*np.pi) ** (P/2)) * np.sqrt(det))
    X_mu = X - mu
    inv = np.linalg.inv(sigma)
    d2 = np.sum(np.dot(X_mu, inv) * X_mu, axis=1)
    return norm_const * np.exp(-0.5 * d2)

# mean and covariance
mu = np.array([0, 0])
sigma = np.array([[1, -.5],
                  [-.5, 1]])

# x, y grid
x, y = np.mgrid[-3:3:.1, -3:3:.1]
X = np.stack((x.ravel(), y.ravel())).T
norm = multivariate_normal_pdf(X, mean, sigma).reshape(x.shape)

# Do it with scipy
norm_scpy = multivariate_normal(mu, sigma).pdf(np.stack((x, y), axis=2))
assert np.allclose(norm, norm_scpy)

# Plot
fig = plt.figure(figsize=(10, 7))
ax = fig.gca(projection='3d')
surf = ax.plot_surface(x, y, norm, rstride=3,
        cstride=3, cmap=plt.cm.coolwarm,
        linewidth=1, antialiased=False
    )

ax.set_zlim(0, 0.2)
ax.zaxis.set_major_locator(plt.LinearLocator(10))
ax.zaxis.set_major_formatter(plt.FormatStrFormatter('%.02f'))

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('p(x)')

plt.title('Bivariate Normal/Gaussian distribution')
fig.colorbar(surf, shrink=0.5, aspect=7, cmap=plt.cm.coolwarm)
plt.show()
```



4.3.8 Exercises

Dot product and Euclidean norm

Given $\mathbf{a} = [2, 1]^T$ and $\mathbf{b} = [1, 1]^T$

1. Write a function `euclidean(x)` that computes the Euclidean norm of vector, $\mathbf{x}$.
2. Compute the Euclidean norm of $\mathbf{a}$.
3. Compute the Euclidean distance of $\|\mathbf{a} - \mathbf{b}\|_2$.
4. Compute the projection of $\mathbf{b}$ in the direction of vector $\mathbf{a}$: b_a.
5. Simulate a dataset $\mathbf{X}$ of $N = 100$ samples of 2-dimensional vectors.
6. Project all samples in the direction of the vector $\mathbf{a}$.

Covariance matrix and Mahalanobis norm

1. Sample a dataset $\mathbf{X}$ of $N = 100$ samples of 2-dimensional vectors from the bivariate normal distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$ where $\mu = [1, 1]^T$ and $\mathbf{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8, 1 \end{bmatrix}$.
2. Compute the mean vector $\bar{\mathbf{x}}$ and center $\mathbf{X}$. Compare the estimated mean $\bar{\mathbf{x}}$ to the true mean, μ.
3. Compute the empirical covariance matrix $\mathbf{S}$. Compare the estimated covariance matrix $\mathbf{S}$ to the true covariance matrix, $\mathbf{\Sigma}$.

4. Compute $\mathbf{S}^{-1}$ (`Sinv`) the inverse of the covariance matrix by using `scipy.linalg.inv(S)`.
5. Write a function `mahalanobis(x, xbar, Sinv)` that computes the Mahalanobis distance of a vector $\mathbf{x}$ to the mean, $\bar{\mathbf{x}}$.
6. Compute the Mahalanobis and Euclidean distances of each sample $\mathbf{x}_i$ to the mean $\bar{\mathbf{x}}$. Store the results in a 100×2 dataframe.

4.4 Time Series in python

Two libraries:

- Pandas: https://pandas.pydata.org/pandas-docs/stable/timeseries.html
- scipy http://www.statsmodels.org/devel/tsa.html

4.4.1 Stationarity

A TS is said to be stationary if its statistical properties such as mean, variance remain constant over time.

- constant mean
- constant variance
- an autocovariance that does not depend on time.

what is making a TS non-stationary. There are 2 major reasons behind non-stationaruty of a TS:

1. Trend – varying mean over time. For eg, in this case we saw that on average, the number of passengers was growing over time.
2. Seasonality – variations at specific time-frames. eg people might have a tendency to buy cars in a particular month because of pay increment or festivals.

4.4.2 Pandas Time Series Data Structure

A Series is similar to a list or an array in Python. It represents a series of values (numeric or otherwise) such as a column of data. It provides additional functionality, methods, and operators, which make it a more powerful version of a list.

```python
import pandas as pd
import numpy as np

# Create a Series from a list
ser = pd.Series([1, 3])
print(ser)

# String as index
prices = {'apple': 4.99,
         'banana': 1.99,
         'orange': 3.99}
ser = pd.Series(prices)
```

(continues on next page)

(continued from previous page)

```
print(ser)

x = pd.Series(np.arange(1,3), index=[x for x in 'ab'])
print(x)
print(x['b'])
```

```
0    1
1    3
dtype: int64
apple     4.99
banana    1.99
orange    3.99
dtype: float64
a    1
b    2
dtype: int64
2
```

4.4.3 Time Series Analysis of Google Trends

source: https://www.datacamp.com/community/tutorials/time-series-analysis-tutorial

Get Google Trends data of keywords such as 'diet' and 'gym' and see how they vary over time while learning about trends and seasonality in time series data.

In the Facebook Live code along session on the 4th of January, we checked out Google trends data of keywords 'diet', 'gym' and 'finance' to see how they vary over time. We asked ourselves if there could be more searches for these terms in January when we're all trying to turn over a new leaf?

In this tutorial, you'll go through the code that we put together during the session step by step. You're not going to do much mathematics but you are going to do the following:

- Read data
- Recode data
- Exploratory Data Analysis

4.4.4 Read data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Plot appears on its own windows
%matplotlib inline
# Tools / Preferences / Ipython Console  / Graphics  / Graphics Backend / Backend:␣
↪"automatic"
# Interactive Matplotlib Jupyter Notebook
# %matplotlib inline

```

(continues on next page)

(continued from previous page)

```
try:
    url = "https://raw.githubusercontent.com/datacamp/datacamp_facebook_live_ny_
↪resolution/master/datasets/multiTimeline.csv"
    df = pd.read_csv(url, skiprows=2)
except:
    df = pd.read_csv("../datasets/multiTimeline.csv", skiprows=2)

print(df.head())

# Rename columns
df.columns = ['month', 'diet', 'gym', 'finance']

# Describe
print(df.describe())
```

```
     Month  diet: (Worldwide)  gym: (Worldwide)  finance: (Worldwide)
0  2004-01                100                31                    48
1  2004-02                 75                26                    49
2  2004-03                 67                24                    47
3  2004-04                 70                22                    48
4  2004-05                 72                22                    43
             diet         gym     finance
count  168.000000  168.000000  168.000000
mean    49.642857   34.690476   47.148810
std      8.033080    8.134316    4.972547
min     34.000000   22.000000   38.000000
25%     44.000000   28.000000   44.000000
50%     48.500000   32.500000   46.000000
75%     53.000000   41.000000   50.000000
max    100.000000   58.000000   73.000000
```

4.4.5 Recode data

Next, you'll turn the 'month' column into a DateTime data type and make it the index of the DataFrame.

Note that you do this because you saw in the result of the .info() method that the 'Month' column was actually an of data type object. Now, that generic data type encapsulates everything from strings to integers, etc. That's not exactly what you want when you want to be looking at time series data. That's why you'll use .to_datetime() to convert the 'month' column in your DataFrame to a DateTime.

Be careful! Make sure to include the inplace argument when you're setting the index of the DataFrame df so that you actually alter the original index and set it to the 'month' column.

```
df.month = pd.to_datetime(df.month)
df.set_index('month', inplace=True)

print(df.head())
```

```
            diet  gym  finance
month
2004-01-01   100   31       48
```

(continues on next page)

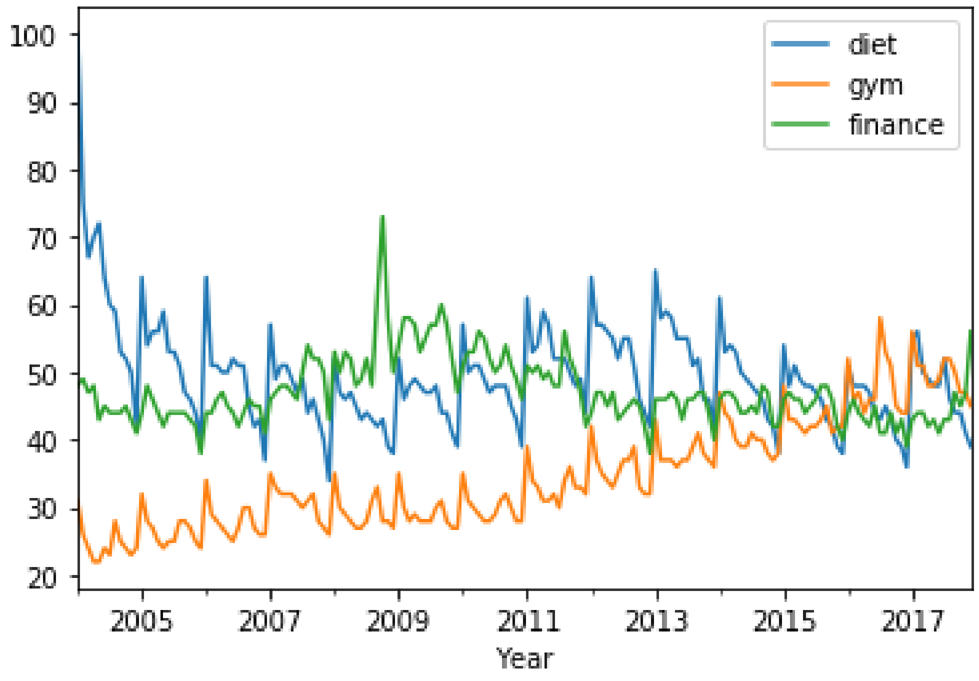
(continued from previous page)

```
2004-02-01    75   26        49
2004-03-01    67   24        47
2004-04-01    70   22        48
2004-05-01    72   22        43
```

4.4.6 Exploratory Data Analysis

You can use a built-in pandas visualization method .plot() to plot your data as 3 line plots on a single figure (one for each column, namely, 'diet', 'gym', and 'finance').

```
df.plot()
plt.xlabel('Year');

# change figure parameters
# df.plot(figsize=(20,10), linewidth=5, fontsize=20)

# Plot single column
# df[['diet']].plot(figsize=(20,10), linewidth=5, fontsize=20)
# plt.xlabel('Year', fontsize=20);
```



Note that this data is relative. As you can read on Google trends:

Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0 means the term was less than 1% as popular as the peak.

4.4.7 Resampling, Smoothing, Windowing, Rolling average: Trends

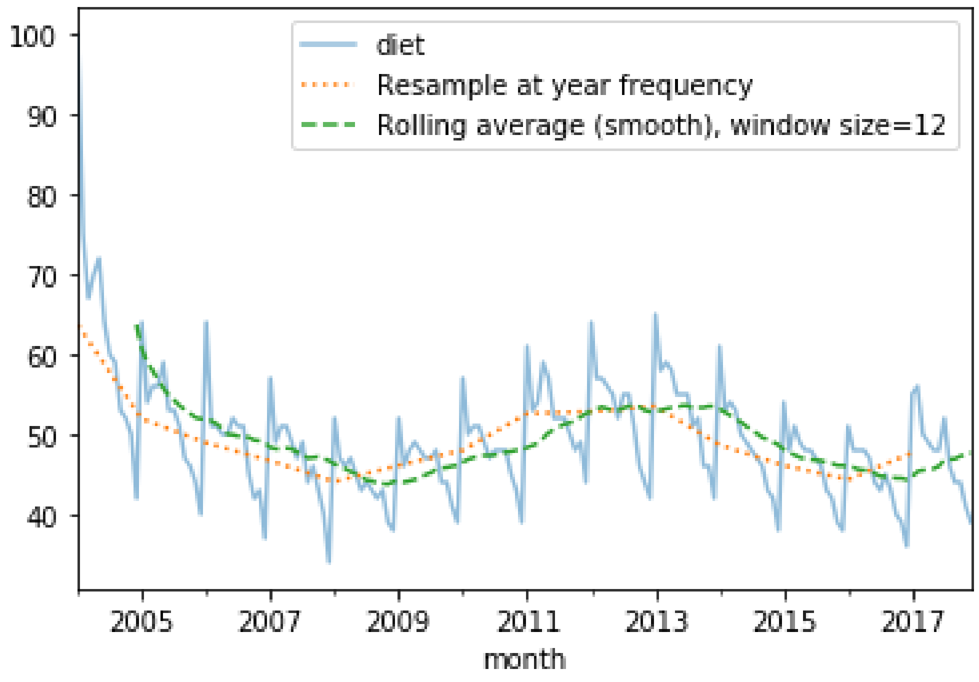
Rolling average, for each time point, take the average of the points on either side of it. Note that the number of points is specified by a window size.

Remove Seasonality with pandas Series.

See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html A: 'year end frequency' year frequency

```
diet = df['diet']

diet_resamp_yr = diet.resample('A').mean()
diet_roll_yr = diet.rolling(12).mean()

ax = diet.plot(alpha=0.5, style='-') # store axis (ax) for latter plots
diet_resamp_yr.plot(style=':', label='Resample at year frequency', ax=ax)
diet_roll_yr.plot(style='--', label='Rolling average (smooth), window size=12', ax=ax)
ax.legend()
```

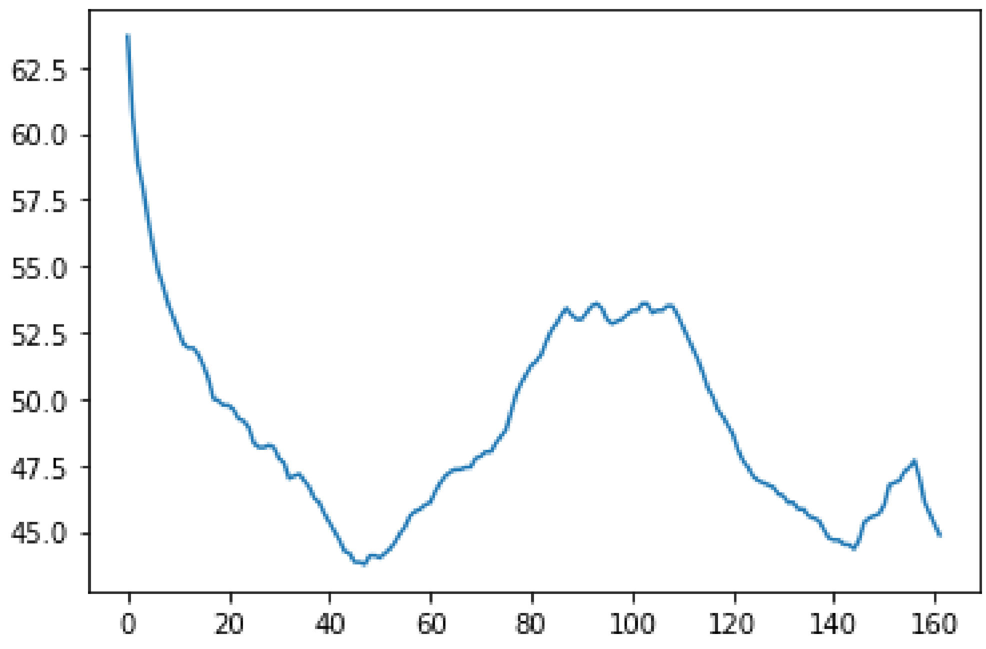
```
<matplotlib.legend.Legend at 0x7f0db4e0a2b0>
```



Rolling average (smoothing) with Numpy

```
x = np.asarray(df[['diet']])
win = 12
win_half = int(win / 2)
# print([((idx-win_half), (idx+win_half)) for idx in np.arange(win_half, len(x))])

diet_smooth = np.array([x[(idx-win_half):(idx+win_half)].mean() for idx in np.arange(win_
↪half, len(x))])
plt.plot(diet_smooth)
```

```
[<matplotlib.lines.Line2D at 0x7f0db4cfea90>]
```

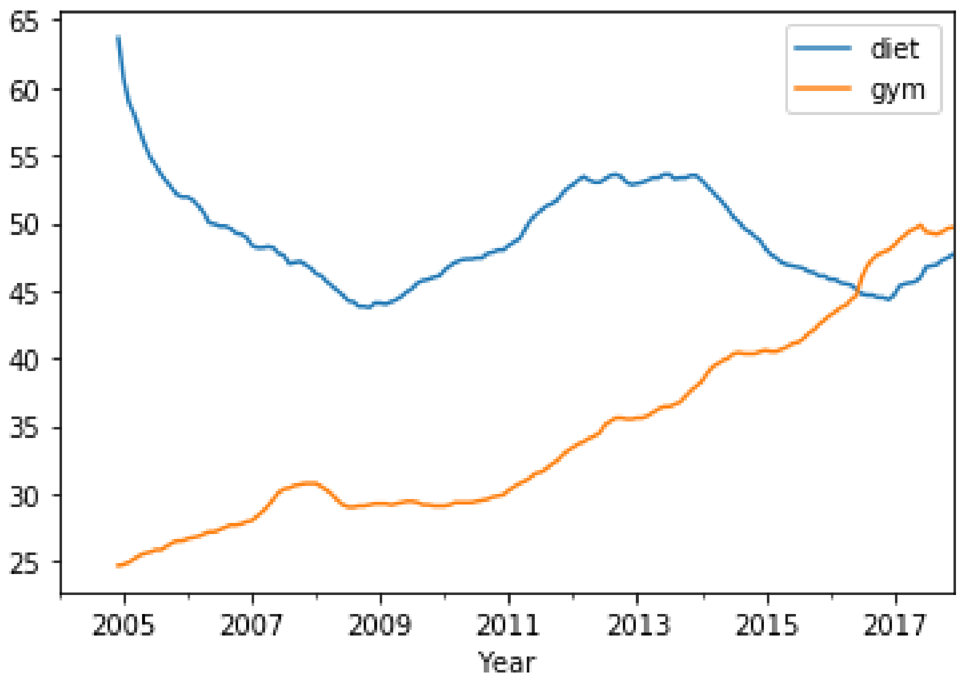


Trends Plot Diet and Gym

Build a new DataFrame which is the concatenation diet and gym smoothed data

```
gym = df['gym']

df_avg = pd.concat([diet.rolling(12).mean(), gym.rolling(12).mean()], axis=1)
df_avg.plot()
plt.xlabel('Year')
```

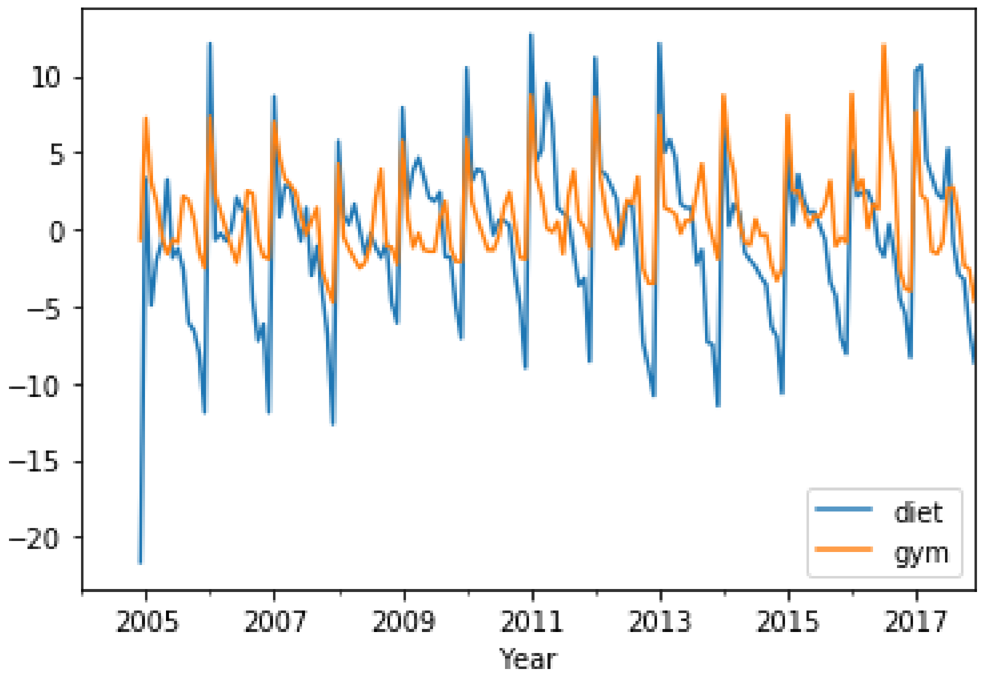
```
Text(0.5, 0, 'Year')
```



Detrending

```
df_dtrend = df[["diet", "gym"]] - df_avg
df_dtrend.plot()
plt.xlabel('Year')
```

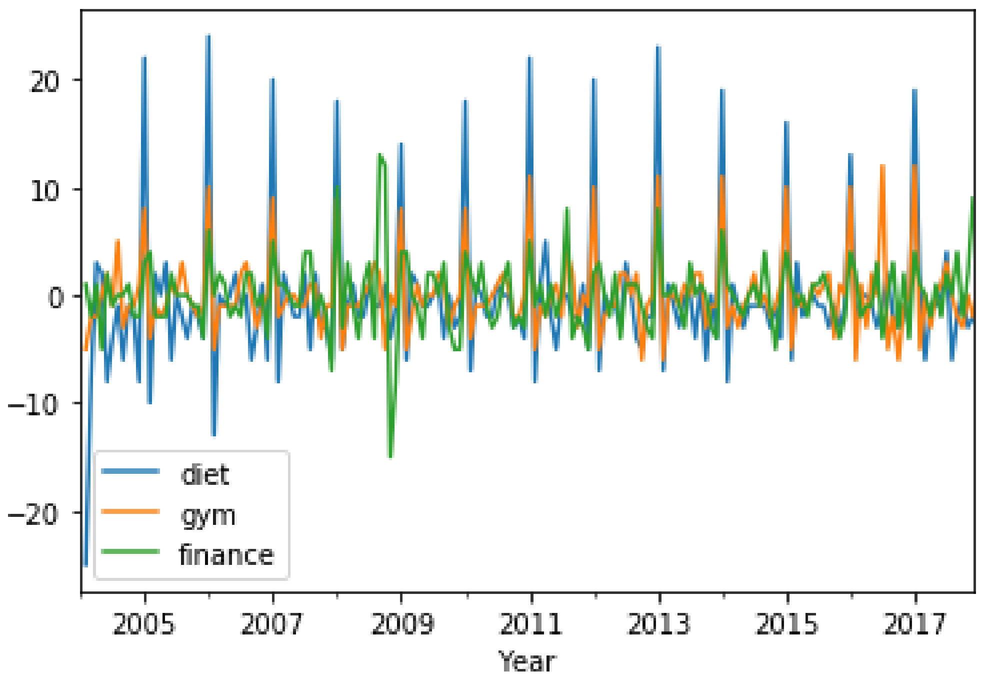
```
Text(0.5, 0, 'Year')
```



4.4.8 First-order differencing: Seasonal Patterns

```
# diff = original - shiftted data
# (exclude first term for some implementation details)
assert np.all((diet.diff() == diet - diet.shift())[1:])

df.diff().plot()
plt.xlabel('Year')
```

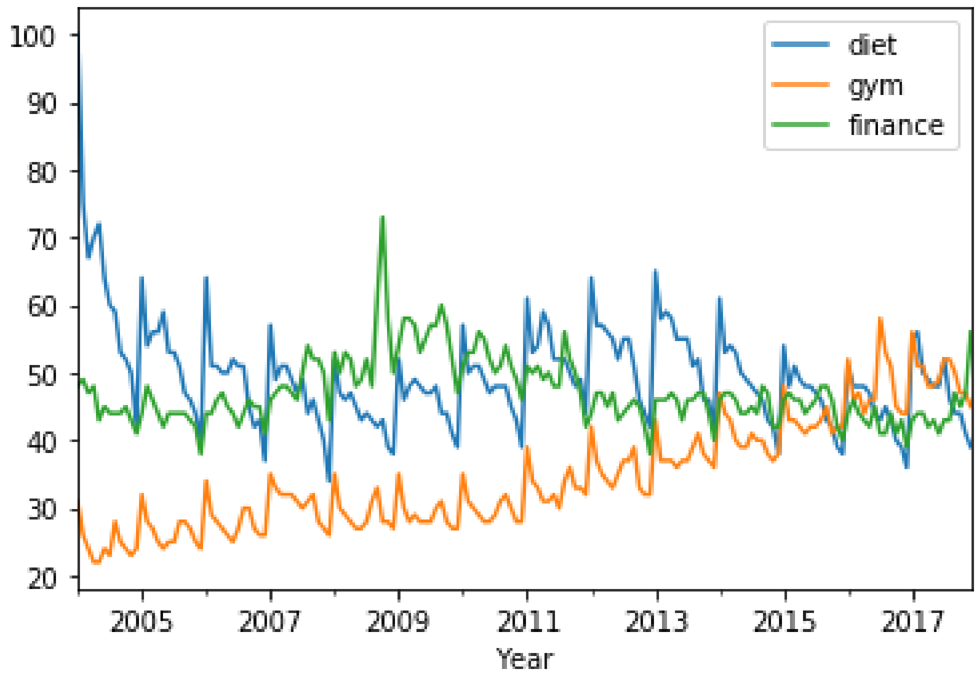
```
Text(0.5, 0, 'Year')
```



4.4.9 Periodicity and Correlation

```
df.plot()
plt.xlabel('Year');
print(df.corr())
```

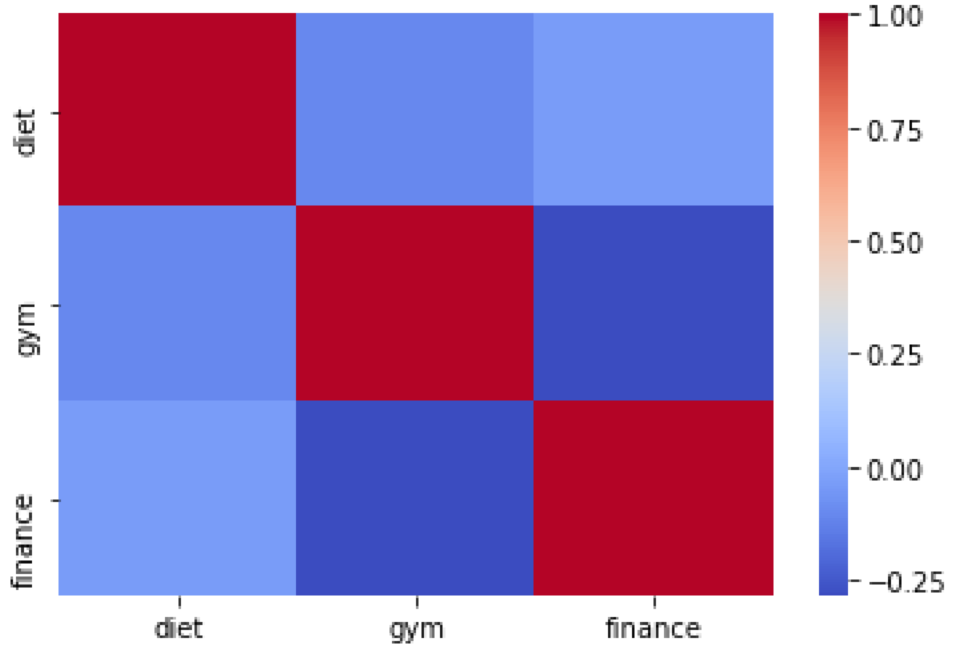
```
             diet       gym   finance
diet     1.000000 -0.100764 -0.034639
gym     -0.100764  1.000000 -0.284279
finance -0.034639 -0.284279  1.000000
```



Plot correlation matrix

```
sns.heatmap(df.corr(), cmap="coolwarm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0db29f3ba8>
```



'diet' and 'gym' are negatively correlated! Remember that you have a seasonal and a trend component. From the correlation coefficient, 'diet' and 'gym' are negatively correlated:

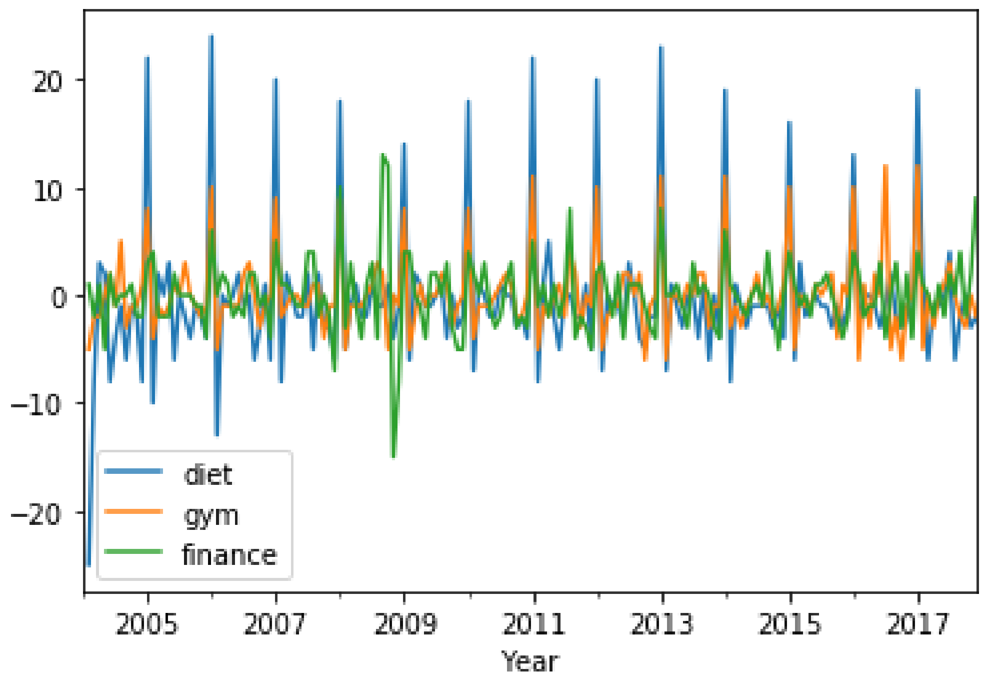
- trends components are negatively correlated.
- seasonal components would positively correlated and their

The actual correlation coefficient is actually capturing both of those.

Seasonal correlation: correlation of the first-order differences of these time series

```
df.diff().plot()
plt.xlabel('Year');

print(df.diff().corr())
```

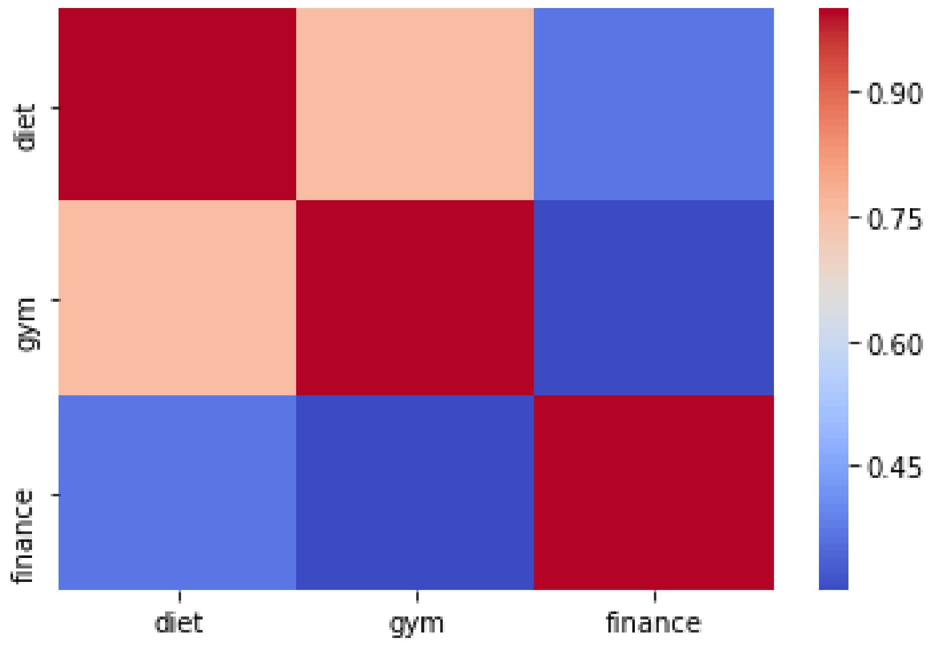
```
             diet       gym   finance
diet     1.000000  0.758707  0.373828
gym      0.758707  1.000000  0.301111
finance  0.373828  0.301111  1.000000
```



Plot correlation matrix

```
sns.heatmap(df.diff().corr(), cmap="coolwarm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0db28aeb70>
```



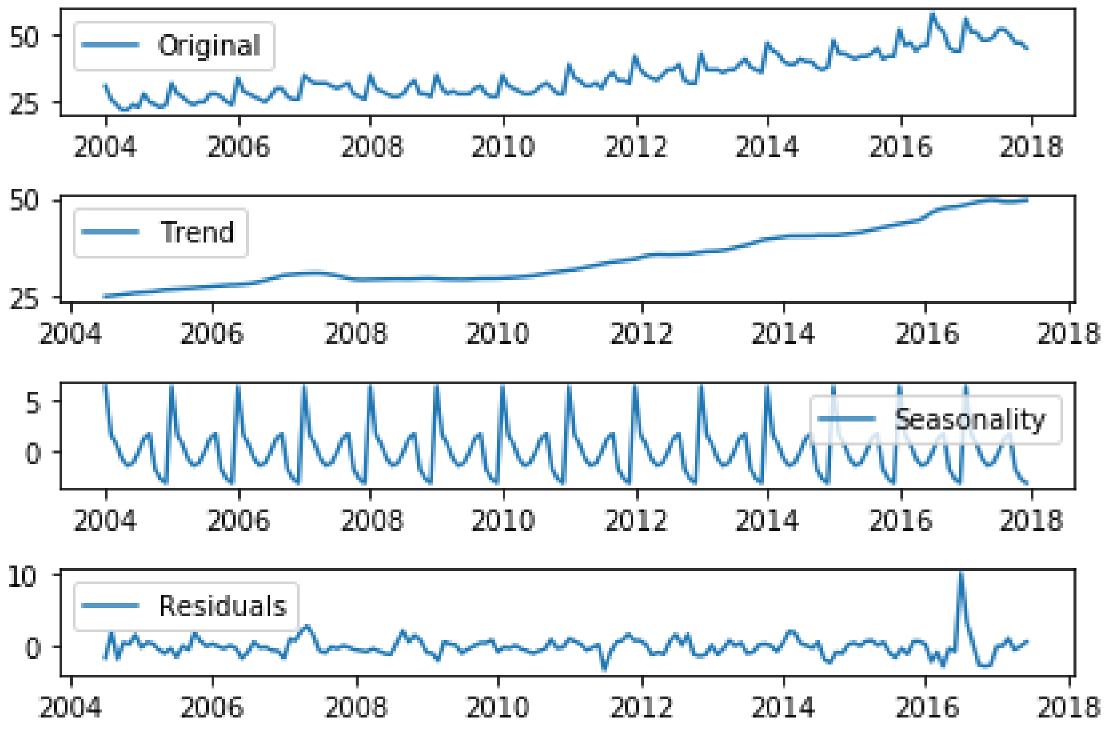
Decomposing time serie in trend, seasonality and residuals

```
from statsmodels.tsa.seasonal import seasonal_decompose

x = gym

x = x.astype(float) # force float
```

(continues on next page)

(continued from previous page)

```
decomposition = seasonal_decompose(x)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(x, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



4.4.10 Autocorrelation

A time series is periodic if it repeats itself at equally spaced intervals, say, every 12 months. Autocorrelation Function (ACF): It is a measure of the correlation between the TS with a lagged version of itself. For instance at lag 5, ACF would compare series at time instant t1. . . t2 with series at instant t1-5. . . t2-5 (t1-5 and t2 being end points).

Plot

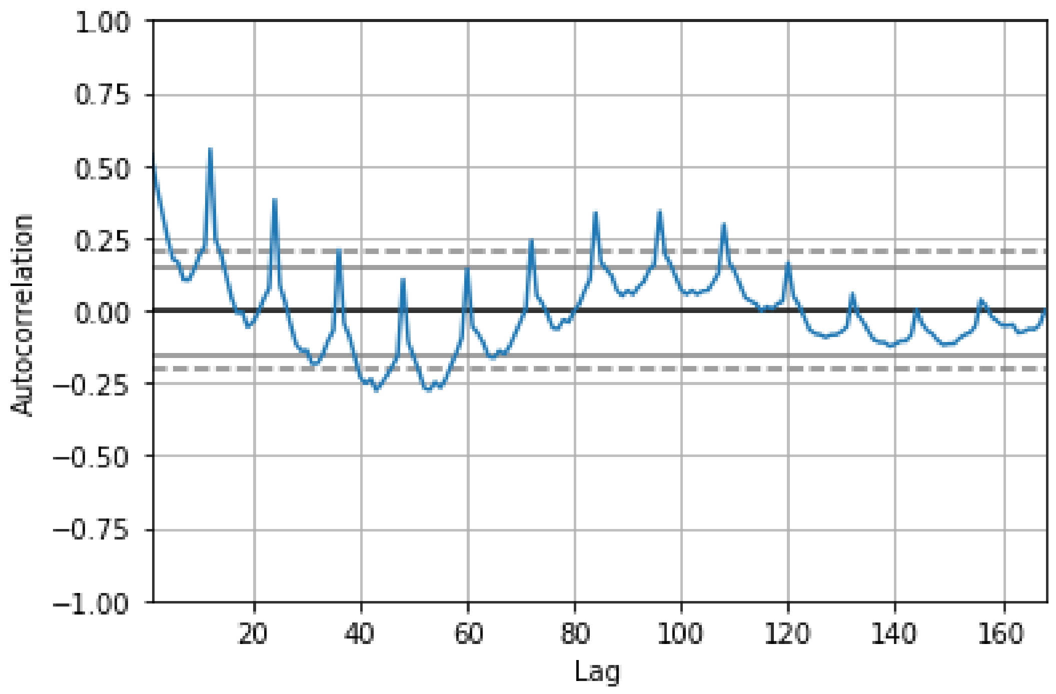
```
# from pandas.plotting import autocorrelation_plot
from pandas.plotting import autocorrelation_plot
```

(continues on next page)

(continued from previous page)

```
x = df["diet"].astype(float)
autocorrelation_plot(x)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0db25b2dd8>
```

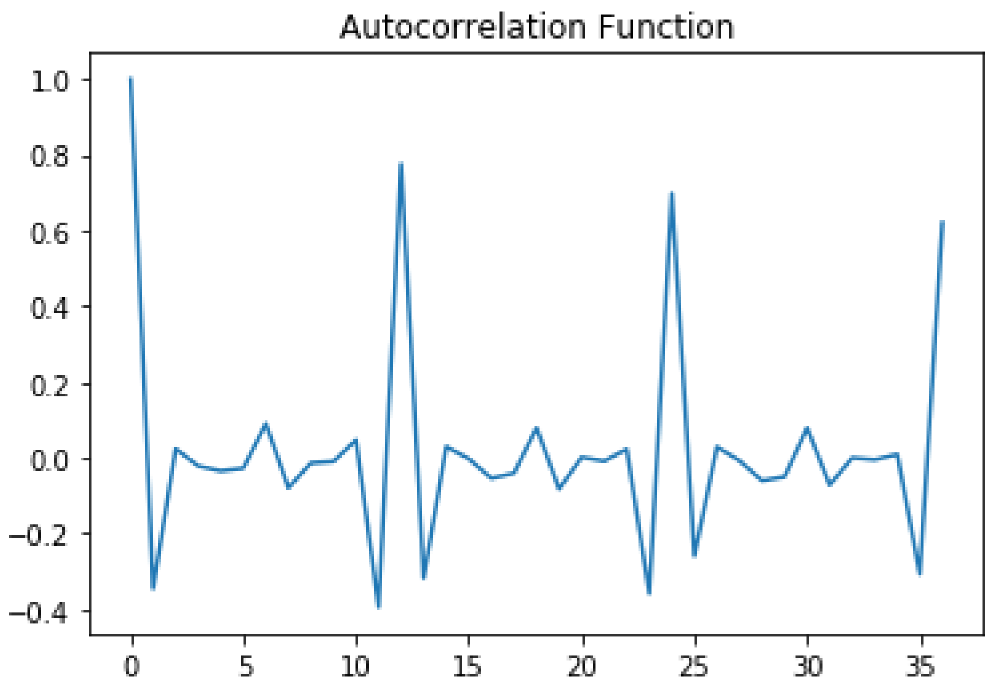


Compute Autocorrelation Function (ACF)

```
from statsmodels.tsa.stattools import acf

x_diff = x.diff().dropna() # first item is NA
lag_acf = acf(x_diff, nlags=36)
plt.plot(lag_acf)
plt.title('Autocorrelation Function')
```

```
/home/edouard/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/stattools.py:541:␣
↪FutureWarning: fft=True will become the default in a future version of statsmodels. To␣
↪suppress this warning, explicitly set fft=False.
  warnings.warn(msg, FutureWarning)
```

```
Text(0.5, 1.0, 'Autocorrelation Function')
```



ACF peaks every 12 months: Time series is correlated with itself shifted by 12 months.

4.4.11 Time Series Forecasting with Python using Autoregressive Moving Average (ARMA) models

Source:

- https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781783553358/7/ch07lvl1sec77/arma-models
- http://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model
- ARIMA: https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

ARMA models are often used to forecast a time series. These models combine autoregressive and moving average models. In moving average models, we assume that a variable is the sum of the mean of the time series and a linear combination of noise components.

The autoregressive and moving average models can have different orders. In general, we can define an ARMA model with p autoregressive terms and q moving average terms as follows:

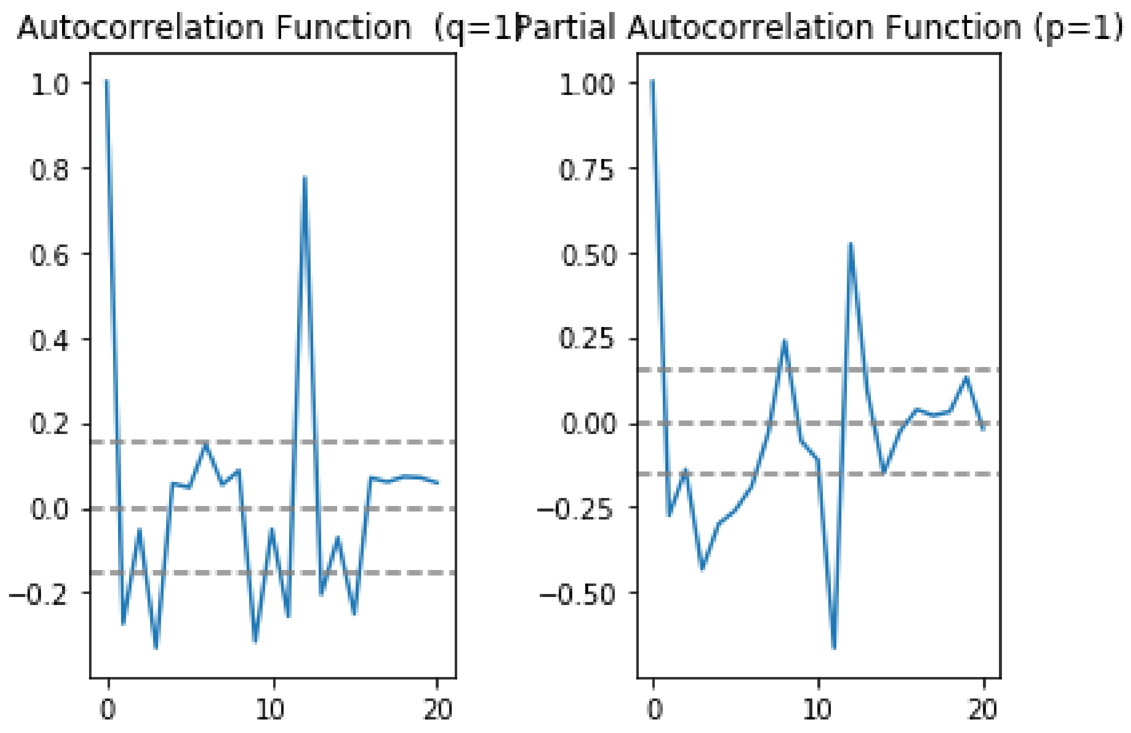
$$x_t = \sum_i^p a_i x_{t-i} + \sum_i^q b_i \varepsilon_{t-i} + \varepsilon_t$$

Choosing p and q

Plot the partial autocorrelation functions for an estimate of p, and likewise using the autocorrelation functions for an estimate of q.

Partial Autocorrelation Function (PACF): This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Eg at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

```
from statsmodels.tsa.stattools import acf, pacf

x = df["gym"].astype(float)

x_diff = x.diff().dropna() # first item is NA
# ACF and PACF plots:

lag_acf = acf(x_diff, nlags=20)
lag_pacf = pacf(x_diff, nlags=20, method='ols')

#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function  (q=1)')

#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(x_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function (p=1)')
plt.tight_layout()
```



In this plot, the two dotted lines on either sides of 0 are the confidence interevals. These can be used to determine the p and q values as:

- p: The lag value where the PACF chart crosses the upper confidence interval for the first time, in this case p=1.

- q: The lag value where the ACF chart crosses the upper confidence interval for the first time, in this case q=1.

Fit ARMA model with statsmodels

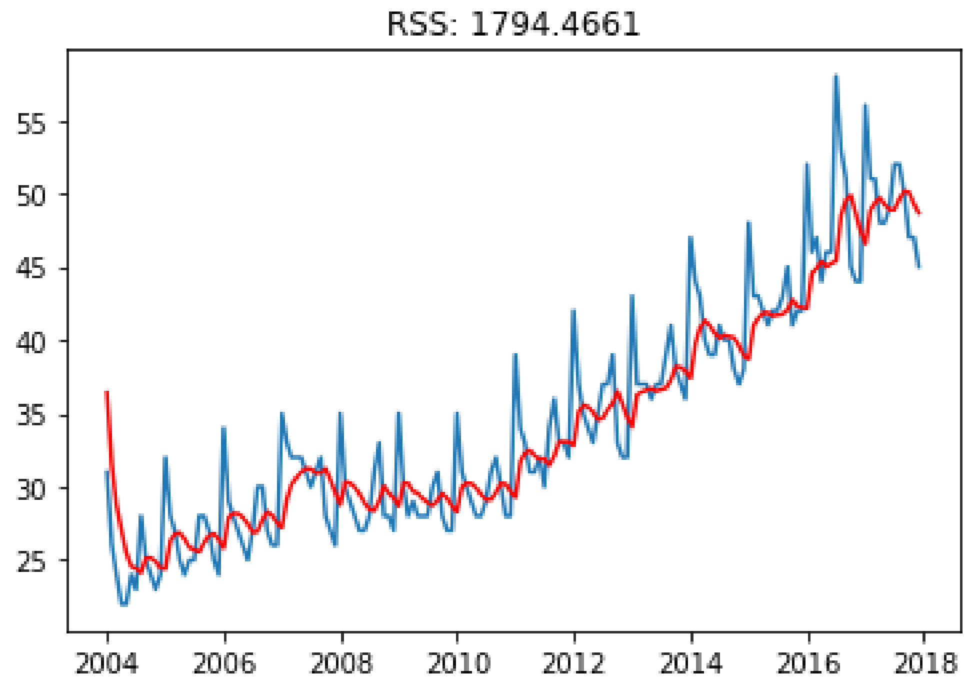
1. Define the model by calling `ARMA()` and passing in the p and q parameters.
2. The model is prepared on the training data by calling the `fit()` function.
3. Predictions can be made by calling the `predict()` function and specifying the index of the time or times to be predicted.

```
from statsmodels.tsa.arima_model import ARMA

model = ARMA(x, order=(1, 1)).fit() # fit model

print(model.summary())
plt.plot(x)
plt.plot(model.predict(), color='red')
plt.title('RSS: %.4f'% sum((model.fittedvalues-x)**2))
```

```
/home/edouard/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.
↪py:165: ValueWarning: No frequency information was provided, so inferred frequency MS␣
↪will be used.
  % freq, ValueWarning)
/home/edouard/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/kalmanf/kalmanfilter.
↪py:221: RuntimeWarning: divide by zero encountered in true_divide
  Z_mat, R_mat, T_mat)
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                    gym   No. Observations:                  168
Model:                     ARMA(1, 1)   Log Likelihood                -436.852
Method:                       css-mle   S.D. of innovations              3.229
Date:                Tue, 29 Oct 2019   AIC                            881.704
Time:                        11:47:14   BIC                            894.200
Sample:                    01-01-2004   HQIC                           886.776
                         - 12-01-2017
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         36.4316      8.827      4.127      0.000      19.131      53.732
ar.L1.gym      0.9967      0.005    220.566      0.000       0.988       1.006
ma.L1.gym     -0.7494      0.054    -13.931      0.000      -0.855      -0.644
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0033           +0.0000j            1.0033            0.0000
MA.1            1.3344           +0.0000j            1.3344            0.0000
-----------------------------------------------------------------------------
```

```
Text(0.5, 1.0, 'RSS: 1794.4661')
```


RSS: 1794.4661
55
50
45
40
35
30
25
2004
2006
2008
2010
2012
2014
2016
2018

CHAPTER

FIVE

MACHINE LEARNING

5.1 Dimension reduction and feature extraction

5.1.1 Introduction

In machine learning and statistics, dimensionality reduction or dimension reduction is the process of reducing the number of features under consideration, and can be divided into feature selection (not addressed here) and feature extraction.

Feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.

The input matrix $\mathbf{X}$, of dimension $N \times P$, is

$$\begin{bmatrix} x_{11} & \dots & x_{1P} \\ & & \\ \vdots & \mathbf{X} & \vdots \\ & & \\ x_{N1} & \dots & x_{NP} \end{bmatrix}$$

where the rows represent the samples and columns represent the variables.

The goal is to learn a transformation that extracts a few relevant features. This is generally done by exploiting the covariance $\mathbf{\Sigma_{XX}}$ between the input features.

5.1.2 Singular value decomposition and matrix factorization

Matrix factorization principles

Decompose the data matrix $\mathbf{X}_{N\times P}$ into a product of a mixing matrix $\mathbf{U}_{N\times K}$ and a dictionary matrix $\mathbf{V}_{P\times K}$.

$$\mathbf{X} = \mathbf{U}\mathbf{V}^T,$$

If we consider only a subset of components $K < rank(\mathbf{X}) < \min(P, N-1)$, $\mathbf{X}$ is approximated by a matrix $\hat{\mathbf{X}}$:

$$\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{U}\mathbf{V}^T,$$

Each line of $\mathbf{x_i}$ is a linear combination (mixing $\mathbf{u_i}$) of dictionary items $\mathbf{V}$.

N P-dimensional data points lie in a space whose dimension is less than $N-1$ (2 dots lie on a line, 3 on a plane, etc.).

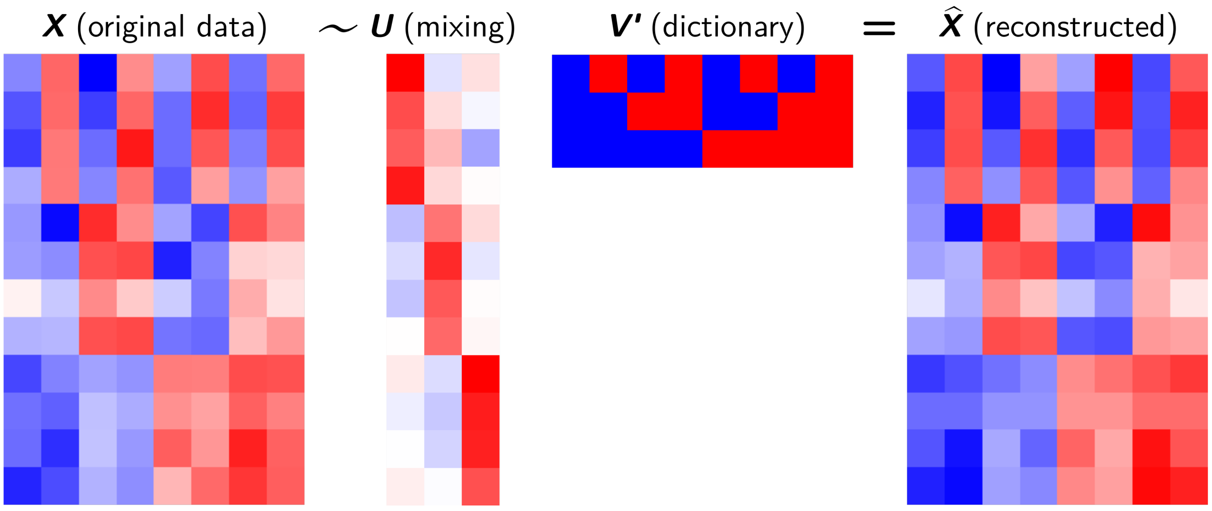


Fig. 1: Matrix factorization

Singular value decomposition (SVD) principles

Singular-value decomposition (SVD) factorises the data matrix $\mathbf{X}_{N \times P}$ into a product:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

where

$$\begin{bmatrix} x_{11} & & x_{1P} \\ & \mathbf{X} & \\ x_{N1} & & x_{NP} \end{bmatrix} = \begin{bmatrix} u_{11} & & u_{1K} \\ & \mathbf{U} & \\ u_{N1} & & u_{NK} \end{bmatrix} \begin{bmatrix} d_1 & & 0 \\ & \mathbf{D} & \\ 0 & & d_K \end{bmatrix} \begin{bmatrix} v_{11} & & v_{1P} \\ & \mathbf{V}^T & \\ v_{K1} & & v_{KP} \end{bmatrix}.$$

U: **right-singular**

- $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_K]$ is a $P \times K$ orthogonal matrix.
- It is a **dictionary** of patterns to be combined (according to the mixing coefficients) to reconstruct the original samples.
- $\mathbf{V}$ perfoms the initial **rotations** (**projection**) along the $K = \min(N, P)$ **principal component directions**, also called **loadings**.
- Each $\mathbf{v}_j$ performs the linear combination of the variables that has maximum sample variance, subject to being uncorrelated with the previous $\mathbf{v}_{j-1}$.

D: **singular values**

- $\mathbf{D}$ is a $K \times K$ diagonal matrix made of the singular values of $\mathbf{X}$ with $d_1 \geq d_2 \geq \cdots \geq d_K \geq 0$.
- $\mathbf{D}$ scale the projection along the coordinate axes by $d_1, d_2, \cdots, d_K$.

- Singular values are the square roots of the eigenvalues of $\mathbf{X}^T\mathbf{X}$.

$\mathbf{V}$**: left-singular vectors**

- $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_K]$ is an $N \times K$ orthogonal matrix.
- Each row $\mathbf{v_i}$ provides the **mixing coefficients** of dictionary items to reconstruct the sample $\mathbf{x_i}$
- It may be understood as the coordinates on the new orthogonal basis (obtained after the initial rotation) called **principal components** in the PCA.

SVD for variables transformation

$\mathbf{V}$ transforms correlated variables ($\mathbf{X}$) into a set of uncorrelated ones ($\mathbf{UD}$) that better expose the various relationships among the original data items.

$$\begin{aligned}
\mathbf{X} &= \mathbf{UDV}^T, && (5.1)\\
\mathbf{XV} &= \mathbf{UDV}^T\mathbf{V}, && (5.2)\\
\mathbf{XV} &= \mathbf{UDI}, && (5.3)\\
\mathbf{XV} &= \mathbf{UD} && (5.4)
\end{aligned}$$

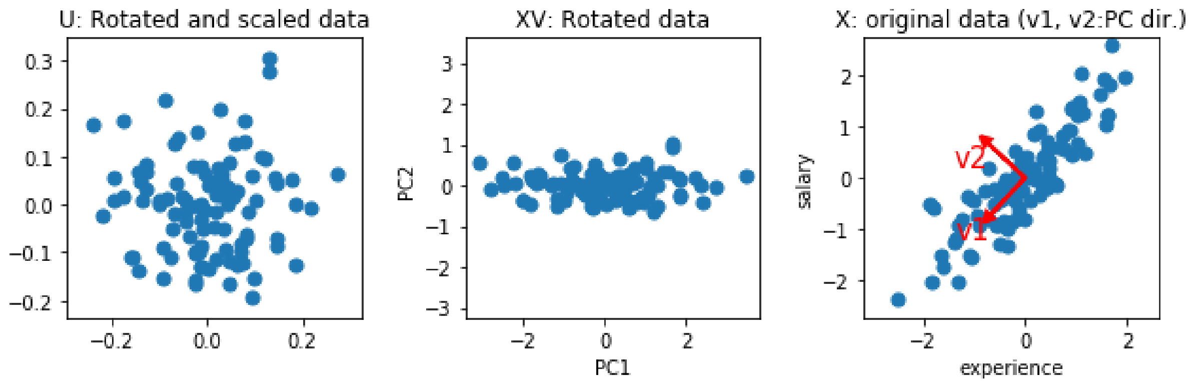
At the same time, SVD is a method for identifying and ordering the dimensions along which data points exhibit the most variation.

```python
import numpy as np
import scipy
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

np.random.seed(42)

# dataset
n_samples = 100
experience = np.random.normal(size=n_samples)
salary = 1500 + experience + np.random.normal(size=n_samples, scale=.5)
X = np.column_stack([experience, salary])

# PCA using SVD
X -= X.mean(axis=0)  # Centering is required
U, s, Vh = scipy.linalg.svd(X, full_matrices=False)
# U : Unitary matrix having left singular vectors as columns.
#     Of shape (n_samples,n_samples) or (n_samples,n_comps), depending on
#     full_matrices.
#
# s : The singular values, sorted in non-increasing order. Of shape (n_comps,),
#     with n_comps = min(n_samples, n_features).
#
# Vh: Unitary matrix having right singular vectors as rows.
#     Of shape (n_features, n_features) or (n_comps, n_features) depending
# on full_matrices.
```

(continues on next page)

(continued from previous page)

```
plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.scatter(U[:, 0], U[:, 1], s=50)
plt.axis('equal')
plt.title("U: Rotated and scaled data")

plt.subplot(132)

# Project data
PC = np.dot(X, Vh.T)
plt.scatter(PC[:, 0], PC[:, 1], s=50)
plt.axis('equal')
plt.title("XV: Rotated data")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.subplot(133)
plt.scatter(X[:, 0], X[:, 1], s=50)
for i in range(Vh.shape[0]):
    plt.arrow(x=0, y=0, dx=Vh[i, 0], dy=Vh[i, 1], head_width=0.2,
              head_length=0.2, linewidth=2, fc='r', ec='r')
    plt.text(Vh[i, 0], Vh[i, 1],'v%i' % (i+1), color="r", fontsize=15,
             horizontalalignment='right', verticalalignment='top')
plt.axis('equal')
plt.ylim(-4, 4)

plt.title("X: original data (v1, v2:PC dir.)")
plt.xlabel("experience")
plt.ylabel("salary")

plt.tight_layout()
```



5.1.3 Principal components analysis (PCA)

Sources:

- C. M. Bishop *Pattern Recognition and Machine Learning*, Springer, 2006
- Everything you did and didn't know about PCA
- Principal Component Analysis in 3 Simple Steps

Principles

- Principal components analysis is the main method used for linear dimension reduction.
- The idea of principal component analysis is to find the K **principal components directions** (called the **loadings**) $\mathbf{V}_{K \times P}$ that capture the variation in the data as much as possible.
- It converts a set of N P-dimensional observations $\mathbf{N}_{N \times P}$ of possibly correlated variables into a set of N K-dimensional samples $\mathbf{C}_{N \times K}$, where the $K < P$. The new variables are linearly uncorrelated. The columns of $\mathbf{C}_{N \times K}$ are called the **principal components**.
- The dimension reduction is obtained by using only $K < P$ components that exploit correlation (covariance) among the original variables.
- PCA is mathematically defined as an orthogonal linear transformation $\mathbf{V}_{K \times P}$ that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

$$\mathbf{C}_{N \times K} = \mathbf{X}_{N \times P} \mathbf{V}_{P \times K}$$

- PCA can be thought of as fitting a P-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipse is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a commensurately small amount of information.
- Finding the K largest axes of the ellipse will permit to project the data onto a space having dimensionality $K < P$ while maximizing the variance of the projected data.

Dataset preprocessing

Centering

Consider a data matrix, $\mathbf{X}$, with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), ie. $\mathbf{X}$ is replaced by $\mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^T$.

Standardizing

Optionally, standardize the columns, i.e., scale them by their standard-deviation. Without standardization, a variable with a high variance will capture most of the effect of the PCA. The principal direction will be aligned with this variable. Standardization will, however, raise noise variables to the save level as informative variables.

The covariance matrix of centered standardized data is the correlation matrix.

Eigendecomposition of the data covariance matrix

To begin with, consider the projection onto a one-dimensional space ($K = 1$). We can define the direction of this space using a P-dimensional vector $\mathbf{v}$, which for convenience (and without loss of generality) we shall choose to be a unit vector so that $\|\mathbf{v}\|_2 = 1$ (note that we are only interested in the direction defined by $\mathbf{v}$, not in the magnitude of $\mathbf{v}$ itself). PCA consists of two mains steps:

Projection in the directions that capture the greatest variance

Each P-dimensional data point $\mathbf{x}_i$ is then projected onto $\mathbf{v}$, where the coordinate (in the coordinate system of $\mathbf{v}$) is a scalar value, namely $\mathbf{x}_i^T\mathbf{v}$. I.e., we want to find the vector $\mathbf{v}$ that maximizes these coordinates along $\mathbf{v}$, which we will see corresponds to maximizing the variance of the projected data. This is equivalently expressed as

$$\mathbf{v} = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{N}\sum_i \left(\mathbf{x}_i^T\mathbf{v}\right)^2.$$

We can write this in matrix form as

$$\mathbf{v} = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{N}\|\mathbf{X}\mathbf{v}\|^2 = \frac{1}{N}\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} = \mathbf{v}^T\mathbf{S_{XX}}\mathbf{v},$$

where $\mathbf{S_{XX}}$ is a biased estiamte of the covariance matrix of the data, i.e.

$$\mathbf{S_{XX}} = \frac{1}{N}\mathbf{X}^T\mathbf{X}.$$

We now maximize the projected variance $\mathbf{v}^T\mathbf{S_{XX}}\mathbf{v}$ with respect to $\mathbf{v}$. Clearly, this has to be a constrained maximization to prevent $\|\mathbf{v}_2\| \to \infty$. The appropriate constraint comes from the normalization condition $\|\mathbf{v}\|_2 \equiv \|\mathbf{v}\|_2^2 = \mathbf{v}^T\mathbf{v} = 1$. To enforce this constraint, we introduce a Lagrange multiplier that we shall denote by λ, and then make an unconstrained maximization of

$$\mathbf{v}^T\mathbf{S_{XX}}\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1).$$

By setting the gradient with respect to $\mathbf{v}$ equal to zero, we see that this quantity has a stationary point when

$$\mathbf{S_{XX}}\mathbf{v} = \lambda\mathbf{v}.$$

We note that $\mathbf{v}$ is an eigenvector of $\mathbf{S_{XX}}$.

If we left-multiply the above equation by $\mathbf{v}^T$ and make use of $\mathbf{v}^T\mathbf{v} = 1$, we see that the variance is given by

$$\mathbf{v}^T\mathbf{S_{XX}}\mathbf{v} = \lambda,$$

and so the variance will be at a maximum when $\mathbf{v}$ is equal to the eigenvector corresponding to the largest eigenvalue, λ. This eigenvector is known as the first principal component.

We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance amongst all possible directions that are orthogonal to those already considered. If we consider the general case of a K-dimensional projection space, the optimal linear projection for which the variance of the projected data is maximized is now defined by the K eigenvectors, $\mathbf{v_1}, \ldots, \mathbf{v_K}$, of the data covariance matrix $\mathbf{S_{XX}}$ that corresponds to the K largest eigenvalues, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_K$.

Back to SVD

The sample covariance matrix of **centered data** $\mathbf{X}$ is given by

$$\mathbf{S_{XX}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X}.$$

We rewrite $\mathbf{X}^T\mathbf{X}$ using the SVD decomposition of $\mathbf{X}$ as

$$\begin{aligned}
\mathbf{X}^T\mathbf{X} &= (\mathbf{UDV}^T)^T(\mathbf{UDV}^T)\\
&= \mathbf{VD}^T\mathbf{U}^T\mathbf{UDV}^T\\
&= \mathbf{VD}^2\mathbf{V}^T\\
\mathbf{V}^T\mathbf{X}^T\mathbf{XV} &= \mathbf{D}^2\\
\frac{1}{N-1}\mathbf{V}^T\mathbf{X}^T\mathbf{XV} &= \frac{1}{N-1}\mathbf{D}^2\\
\mathbf{V}^T\mathbf{S_{XX}}\mathbf{V} &= \frac{1}{N-1}\mathbf{D}^2
\end{aligned}$$

.

Considering only the k^{th} right-singular vectors $\mathbf{v}_k$ associated to the singular value d_k

$$\mathbf{v_k}^T\mathbf{S_{XX}}\mathbf{v_k} = \frac{1}{N-1}d_k^2,$$

It turns out that if you have done the singular value decomposition then you already have the Eigenvalue decomposition for $\mathbf{X}^T\mathbf{X}$. Where - The eigenvectors of $\mathbf{S_{XX}}$ are equivalent to the right singular vectors, $\mathbf{V}$, of $\mathbf{X}$. - The eigenvalues, λ_k, of $\mathbf{S_{XX}}$, i.e. the variances of the components, are equal to $\frac{1}{N-1}$ times the squared singular values, d_k.

Moreover computing PCA with SVD do not require to form the matrix $\mathbf{X}^T\mathbf{X}$, so computing the SVD is now the standard way to calculate a principal components analysis from a data matrix, unless only a handful of components are required.

PCA outputs

The SVD or the eigendecomposition of the data covariance matrix provides three main quantities:

1. **Principal component directions** or **loadings** are the **eigenvectors** of $\mathbf{X}^T\mathbf{X}$. The $\mathbf{V}_{K\times P}$ or the **right-singular vectors** of an SVD of $\mathbf{X}$ are called principal component directions of $\mathbf{X}$. They are generally computed using the SVD of $\mathbf{X}$.
2. **Principal components** is the $N \times K$ matrix $\mathbf{C}$ which is obtained by projecting $\mathbf{X}$ onto the principal components directions, i.e.

$$\mathbf{C}_{N\times K} = \mathbf{X}_{N\times P}\mathbf{V}_{P\times K}.$$

Since $\mathbf{X} = \mathbf{UDV}^T$ and $\mathbf{V}$ is orthogonal ($\mathbf{V}^T\mathbf{V} = \mathbf{I}$):

$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}\mathbf{V}^T_{N\times P}\mathbf{V}_{P\times K} \tag{5.5}$$
$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}^T_{N\times K}\mathbf{I}_{K\times K} \tag{5.6}$$
$$\mathbf{C}_{N\times K} = \mathbf{U}\mathbf{D}^T_{N\times K} \tag{5.7}$$
$$\tag{5.8}$$

Thus $\mathbf{c}_j = \mathbf{X}\mathbf{v}_j = \mathbf{u}_j d_j$, for $j = 1, \ldots K$. Hence $\mathbf{u}_j$ is simply the projection of the row vectors of $\mathbf{X}$, i.e., the input predictor vectors, on the direction $\mathbf{v}_j$, scaled by d_j.

$$\mathbf{c}_1 = \begin{bmatrix} x_{1,1}v_{1,1} + \ldots + x_{1,P}v_{1,P} \\ x_{2,1}v_{1,1} + \ldots + x_{2,P}v_{1,P} \\ \vdots \\ x_{N,1}v_{1,1} + \ldots + x_{N,P}v_{1,P} \end{bmatrix}$$

3. The **variance** of each component is given by the eigen values $\lambda_k, k = 1, \ldots K$. It can be obtained from the singular values:

$$var(\mathbf{c}_k) = \frac{1}{N-1}(\mathbf{X}\mathbf{v}_k)^2 \tag{5.9}$$
$$= \frac{1}{N-1}(\mathbf{u}_k d_k)^2 \tag{5.10}$$
$$= \frac{1}{N-1}d_k^2 \tag{5.11}$$

Determining the number of PCs

We must choose $K^* \in [1, \ldots, K]$, the number of required components. This can be done by calculating the explained variance ratio of the K^* first components and by choosing K^* such that the **cumulative explained variance** ratio is greater than some given threshold (e.g., $\approx 90\%$). This is expressed as

$$\text{cumulative explained variance}(\mathbf{c}_k) = \frac{\sum_j^{K^*} var(\mathbf{c}_k)}{\sum_j^{K} var(\mathbf{c}_k)}.$$

Interpretation and visualization

PCs

Plot the samples projeted on first the principal components as e.g. PC1 against PC2.

PC directions

Exploring the loadings associated with a component provides the contribution of each original variable in the component.

Remark: The loadings (PC directions) are the coefficients of multiple regression of PC on original variables:

$$\mathbf{c} = \mathbf{X}\mathbf{v} \tag{5.12}$$

$$\mathbf{X}^T\mathbf{c} = \mathbf{X}^T\mathbf{X}\mathbf{v} \tag{5.13}$$

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{c} = \mathbf{v} \tag{5.14}$$

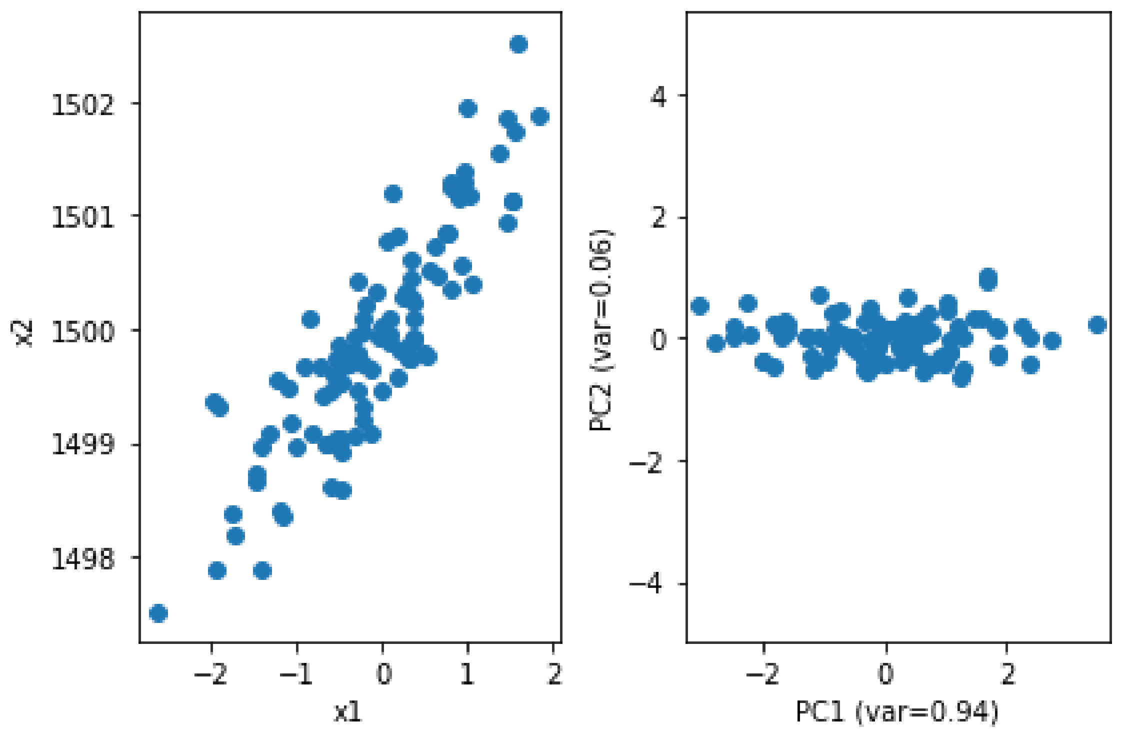
Another way to evaluate the contribution of the original variables in each PC can be obtained by computing the correlation between the PCs and the original variables, i.e. columns of $\mathbf{X}$, denoted $\mathbf{x}_j$, for $j = 1, \ldots, P$. For the k^{th} PC, compute and plot the correlations with all original variables

$$cor(\mathbf{c}_k, \mathbf{x}_j), j = 1 \ldots K, j = 1 \ldots K.$$

These quantities are sometimes called the *correlation loadings*.

```python
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

np.random.seed(42)

# dataset
n_samples = 100
experience = np.random.normal(size=n_samples)
salary = 1500 + experience + np.random.normal(size=n_samples, scale=.5)
X = np.column_stack([experience, salary])

# PCA with scikit-learn
pca = PCA(n_components=2)
pca.fit(X)
print(pca.explained_variance_ratio_)

PC = pca.transform(X)

plt.subplot(121)
plt.scatter(X[:, 0], X[:, 1])
plt.xlabel("x1"); plt.ylabel("x2")

plt.subplot(122)
plt.scatter(PC[:, 0], PC[:, 1])
plt.xlabel("PC1 (var=%.2f)" % pca.explained_variance_ratio_[0])
plt.ylabel("PC2 (var=%.2f)" % pca.explained_variance_ratio_[1])
plt.axis('equal')
plt.tight_layout()
```

```
[0.93646607 0.06353393]
```



5.1.4 Multi-dimensional Scaling (MDS)

Resources:

- http://www.stat.pitt.edu/sungkyu/course/2221Fall13/lec8_mds_combined.pdf
- https://en.wikipedia.org/wiki/Multidimensional_scaling
- Hastie, Tibshirani and Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer, Second Edition.

The purpose of MDS is to find a low-dimensional projection of the data in which the pairwise distances between data points is preserved, as closely as possible (in a least-squares sense).

- Let $\mathbf{D}$ be the $(N \times N)$ pairwise distance matrix where d_{ij} is *a distance* between points i and j.
- The MDS concept can be extended to a wide variety of data types specified in terms of a similarity matrix.

Given the dissimilarity (distance) matrix $\mathbf{D}_{N\times N} = [d_{ij}]$, MDS attempts to find K-dimensional projections of the N points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^K$, concatenated in an $\mathbf{X}_{N\times K}$ matrix, so that $d_{ij} \approx \|\mathbf{x}_i - \mathbf{x}_j\|$ are as close as possible. This can be obtained by the minimization of a loss function called the **stress function**

$$\mathrm{stress}(\mathbf{X}) = \sum_{i\neq j} \left(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|\right)^2.$$

This loss function is known as *least-squares* or *Kruskal-Shepard* scaling.

A modification of *least-squares* scaling is the *Sammon mapping*

$$\mathrm{stress}_{\mathrm{Sammon}}(\mathbf{X}) = \sum_{i\neq j} \frac{(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2}{d_{ij}}.$$

The Sammon mapping performs better at preserving small distances compared to the *least-squares* scaling.

Classical multidimensional scaling

Also known as *principal coordinates analysis*, PCoA.

- The distance matrix, $\mathbf{D}$, is transformed to a *similarity matrix*, $\mathbf{B}$, often using centered inner products.
- The loss function becomes

$$\text{stress}_{\text{classical}}(\mathbf{X}) = \sum_{i \neq j} \big(b_{ij} - \langle \mathbf{x}_i, \mathbf{x}_j \rangle \big)^2.$$

- The stress function in classical MDS is sometimes called *strain*.
- The solution for the classical MDS problems can be found from the eigenvectors of the similarity matrix.
- If the distances in $\mathbf{D}$ are Euclidean and double centered inner products are used, the results are equivalent to PCA.

Example

The `eurodist` datset provides the road distances (in kilometers) between 21 cities in Europe. Given this matrix of pairwise (non-Euclidean) distances $\mathbf{D} = [d_{ij}]$, MDS can be used to recover the coordinates of the cities in *some* Euclidean referential whose orientation is arbitrary.

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

# Pairwise distance between European cities
try:
    url = '../datasets/eurodist.csv'
    df = pd.read_csv(url)
except:
    url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/eurodist.csv'
    df = pd.read_csv(url)

print(df.iloc[:5, :5])

city = df["city"]
D = np.array(df.iloc[:, 1:])  # Distance matrix

# Arbitrary choice of K=2 components
from sklearn.manifold import MDS
mds = MDS(dissimilarity='precomputed', n_components=2, random_state=40, max_iter=3000,␣
↪eps=1e-9)
X = mds.fit_transform(D)
```

```
        city  Athens  Barcelona  Brussels  Calais
0     Athens       0       3313      2963    3175
1  Barcelona    3313          0      1318    1326
```

(continues on next page)

(continued from previous page)

```
2   Brussels    2963        1318         0      204
3     Calais    3175        1326       204        0
4  Cherbourg    3339        1294       583      460
```

Recover coordinates of the cities in Euclidean referential whose orientation is arbitrary:

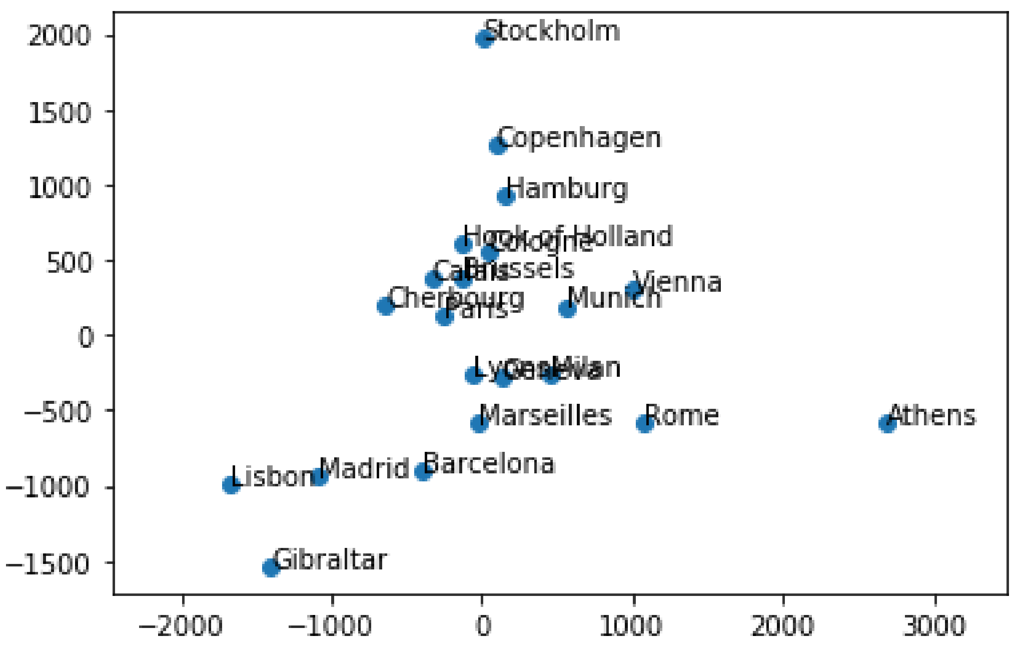
```
from sklearn import metrics
Deuclidean = metrics.pairwise.pairwise_distances(X, metric='euclidean')
print(np.round(Deuclidean[:5, :5]))
```

```
[[   0. 3116. 2994. 3181. 3428.]
 [3116.    0. 1317. 1289. 1128.]
 [2994. 1317.    0.  198.  538.]
 [3181. 1289.  198.    0.  358.]
 [3428. 1128.  538.  358.    0.]]
```

Plot the results:

```
# Plot: apply some rotation and flip
theta = 80 * np.pi / 180.
rot = np.array([[np.cos(theta), -np.sin(theta)],
                [np.sin(theta),  np.cos(theta)]])
Xr = np.dot(X, rot)
# flip x
Xr[:, 0] *= -1
plt.scatter(Xr[:, 0], Xr[:, 1])

for i in range(len(city)):
    plt.text(Xr[i, 0], Xr[i, 1], city[i])
plt.axis('equal')
```

```
(-1894.1017744377398,
 2914.3652937179477,
 -1712.9885463201906,
 2145.4522453884565)
```



Determining the number of components

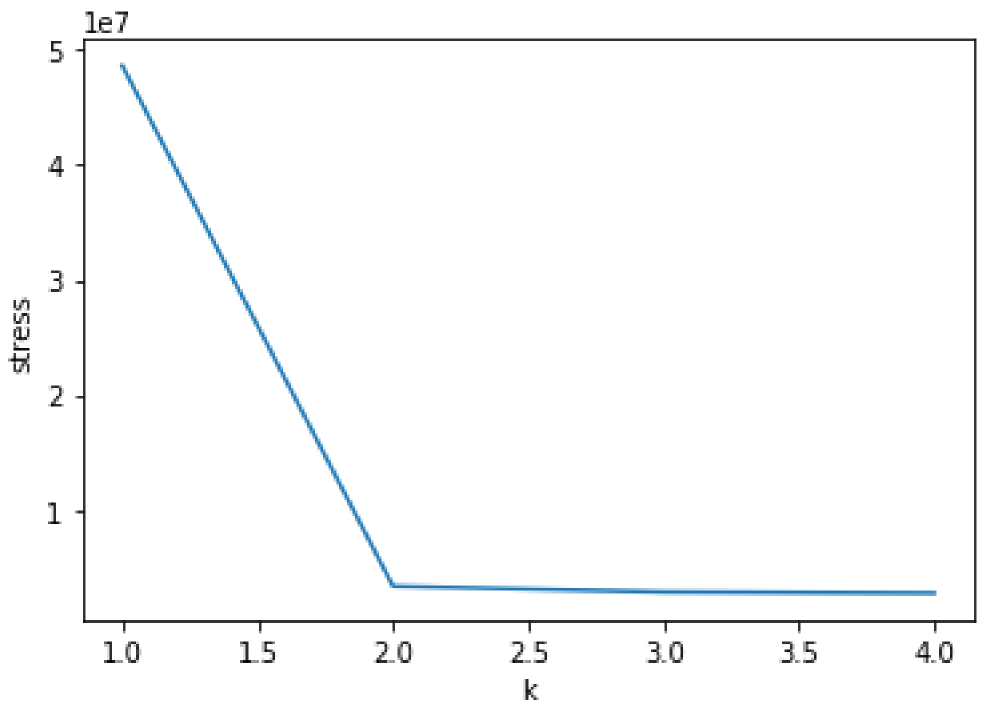
We must choose $K^* \in \{1, \ldots, K\}$ the number of required components. Plotting the values of the stress function, obtained using $k \leq N - 1$ components. In general, start with $1, \ldots K \leq 4$. Choose K^* where you can clearly distinguish an *elbow* in the stress curve.

Thus, in the plot below, we choose to retain information accounted for by the first *two* components, since this is where the *elbow* is in the stress curve.

```
k_range = range(1, min(5, D.shape[0]-1))
stress = [MDS(dissimilarity='precomputed', n_components=k,
           random_state=42, max_iter=300, eps=1e-9).fit(D).stress_ for k in k_range]

print(stress)
plt.plot(k_range, stress)
plt.xlabel("k")
plt.ylabel("stress")
```

```
[48644495.28571428, 3356497.365752386, 2858455.495887962, 2756310.637628011]
```

```
Text(0, 0.5, 'stress')
```



5.1.5 Nonlinear dimensionality reduction

Sources:

- Scikit-learn documentation
- Wikipedia

Nonlinear dimensionality reduction or **manifold learning** cover unsupervised methods that attempt to identify low-dimensional manifolds within the original P-dimensional space that represent high data density. Then those methods provide a mapping from the high-dimensional space to the low-dimensional embedding.

Isomap

Isomap is a nonlinear dimensionality reduction method that combines a procedure to compute the distance matrix with MDS. The distances calculation is based on geodesic distances evaluated on neighborhood graph:

1. Determine the neighbors of each point. All points in some fixed radius or K nearest neighbors.
2. Construct a neighborhood graph. Each point is connected to other if it is a K nearest neighbor. Edge length equal to Euclidean distance.
3. Compute shortest path between pairwise of points d_{ij} to build the distance matrix $\mathbf{D}$.
4. Apply MDS on $\mathbf{D}$.

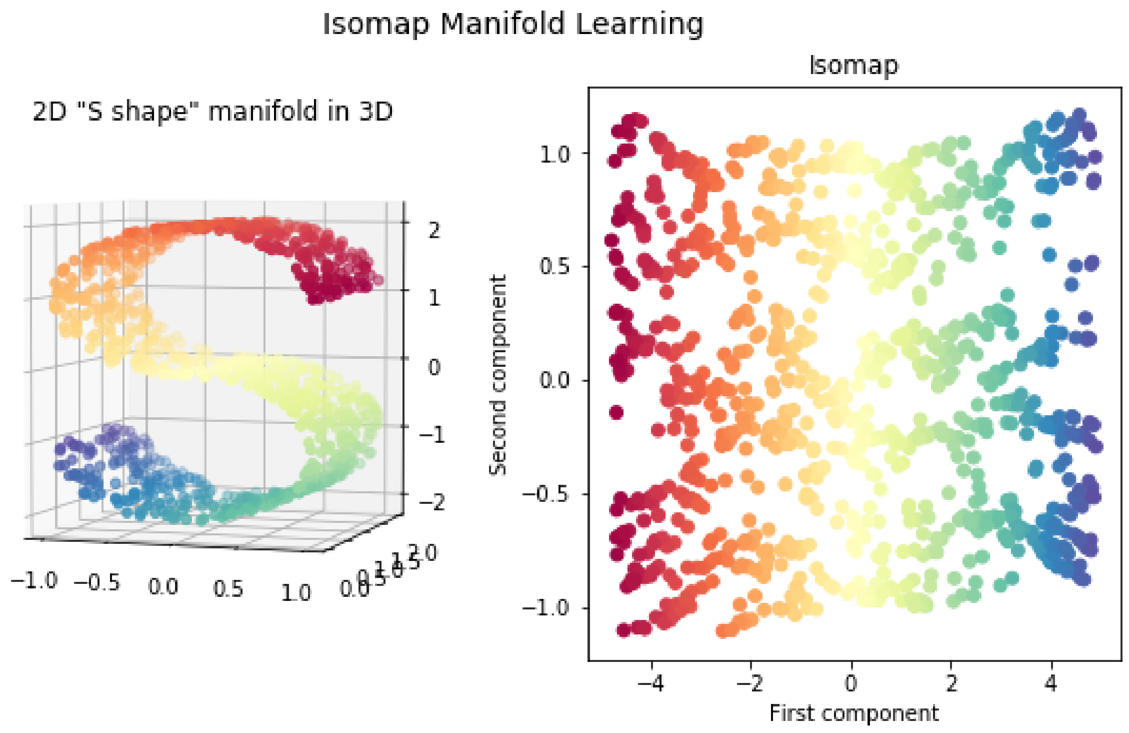
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import manifold, datasets
```

(continues on next page)

(continued from previous page)

```
X, color = datasets.samples_generator.make_s_curve(1000, random_state=42)

fig = plt.figure(figsize=(10, 5))
plt.suptitle("Isomap Manifold Learning", fontsize=14)

ax = fig.add_subplot(121, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax.view_init(4, -72)
plt.title('2D "S shape" manifold in 3D')

Y = manifold.Isomap(n_neighbors=10, n_components=2).fit_transform(X)
ax = fig.add_subplot(122)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("Isomap")
plt.xlabel("First component")
plt.ylabel("Second component")
plt.axis('tight')
```

```
(-5.276311544714793, 5.4164373180970316, -1.23497771017066, 1.2910940054965336)
```



5.1.6 Exercises

PCA

Write a basic PCA class

Write a class `BasicPCA` with two methods:

- `fit(X)` that estimates the data mean, principal components directions **V** and the explained variance of each component.
- `transform(X)` that projects the data onto the principal components.

Check that your `BasicPCA` gave similar results, compared to the results from `sklearn`.

Apply your Basic PCA on the `iris` dataset

The data set is available at: https://raw.github.com/neurospin/pystatsml/master/datasets/iris.csv

- Describe the data set. Should the dataset been standardized?
- Describe the structure of correlations among variables.
- Compute a PCA with the maximum number of components.
- Compute the cumulative explained variance ratio. Determine the number of components K by your computed values.
- Print the K principal components directions and correlations of the K principal components with the original variables. Interpret the contribution of the original variables into the PC.
- Plot the samples projected into the K first PCs.
- Color samples by their species.

MDS

Apply MDS from `sklearn` on the `iris` dataset available at:

https://raw.github.com/neurospin/pystatsml/master/datasets/iris.csv

- Center and scale the dataset.
- Compute Euclidean pairwise distances matrix.
- Select the number of components.
- Show that classical MDS on Euclidean pairwise distances matrix is equivalent to PCA.

5.2 Clustering

Wikipedia: Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). Clustering is one of the main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Sources: http://scikit-learn.org/stable/modules/clustering.html