Data Analysis with Python

Informatica Feminale, 2018-08-18

Christine Koppelt

Introduction

Agenda

- Saturday
  - Getting used to Jupyter
  - Quick python repetition
  - Getting started with pandas
  - Descriptive statistics
  - Combining data, cleaning data
- Sunday
  - Plotting & visualization
  - Time series
  - Linear Regression

Course Materials

https://github.com/cko/if2018-data (https://github.com/cko/if2018-data)

About me

- Senior Consultant at INNOQ since 2011
- Software development since 2007
- Diplom Mathematikerin (FH)
- Current Focus: Microservices, Devops, Data Engineering

What/How/Why data analysis?

Jupyter Overview

Project Jupyter

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

https://jupyter.org/

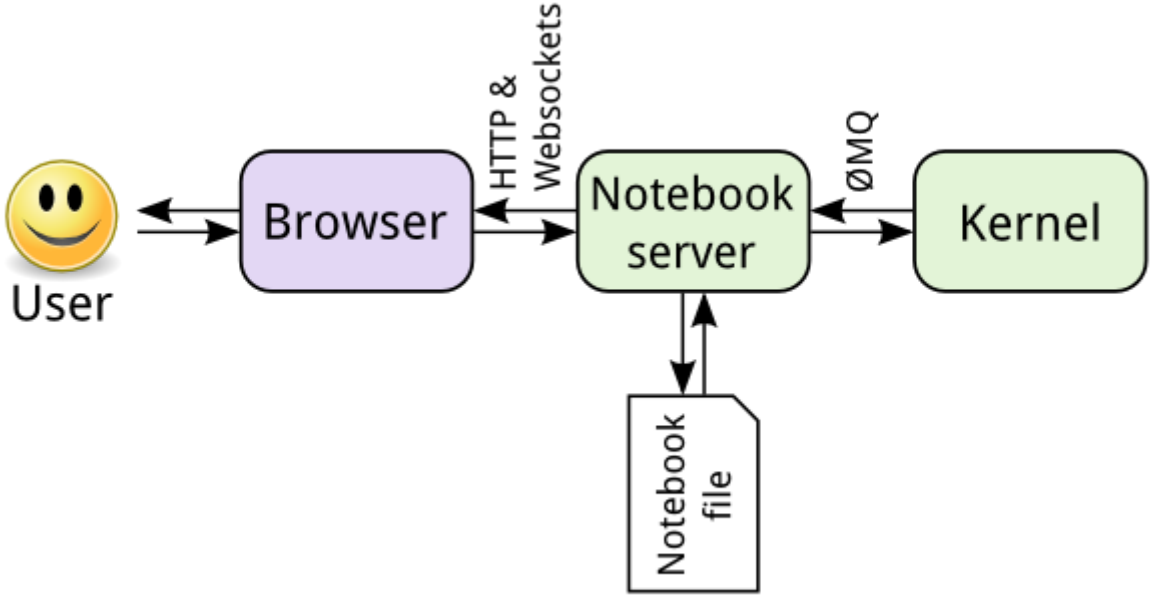
- Origin: iPython, iPython Notebook
- Open source, BSD license
- Started in 2014 by Fernando Pérez, assistant professor in the Department of Statistics at UC Berkeley
- Supported by Microsoft, Google and several foundations
- Very popular in the data analysis / data science / machine learning space

Jupyter Ecosystem

- Supports ~50 languages: Python, R, Julia, Scala, ...
- Similar software: MATLAB, Mathematica, R Studio, Tableau, PowerBI, Excel
- ipywidgets, interactive
- nbviewer
- nbconvert
- RISE, nbpresent
- latex, rst export
- Hub

Demo

Architecture



Use Cases

- Data analysis, data exploration, machine learning
- Data query tool (for debugging or for support)
- Python in the browser
- Publishing and sharing
- Presentations
- Not: software development

Run Cells

- Run and stay at current cell: `Ctrl+Enter`
- Run and advance to next cell: `Shift+Enter`
- Run all cells in a notebook -> Menu

Manage Cells

- Switch between command and edit mode: `Enter`, `ESC`/`Ctrl+M`
- In command mode:
  - Delete cell: `dd`
  - Add cell before `a` or after `b` current cell
  - Copy cell: `c` + `v`
  - Change cell type: markdown `m`, code `y`, raw `r`

Exercise 1 - Jupyter

Goals:

- Have a working Jupyter environment ready
- Getting familiar with Jupyter

Tasks:

- Create a notebook file, create some code cells, write some Python code, like `print('Hello world')`, and execute it
- Create a markdown cell
- Try some shortcuts:
  - Execute a cell: `Ctrl+Enter` and `Shift+Enter`
  - Create a cell before `a` or after `b`
  - Copy `c` and paste `v` a cell
- Print your current working directory

Python

Why Python?

- Easy to use and general-purpose language
- Many scientific libraries for data analysis
- Many libraries for accessing data
- Free & open source
- Your company might already use it for sth else

Variables in Python

- Untyped variable
- Can be re-assigned (no final / val)
- Check current type with `type(variable)`

Data types

- Simple data types: strings, integers, floating point numbers, boolean

In [83]:
```
string1 = 'A string in single quotes allows embedded "double" quotes.'
string2 = "A string in double qoutes allows embedded 'single' quotes."
string3 = """A string in tripled quotes allows multi line string
with "double" and 'single' quotes."""
int1 = 99999999999999999999999999999999999999999999999 # unlimited precision in
 Python 3
float1 = 0.123456789
bool1 = True
string4 = str(int1) # type conversion
type(int1)
```

Out[83]:
```
int
```

List

In [65]:
```
shopping_list = [ 'milk', 'cheese', 'bread' ]
shopping_list.append(0) # add an element
shopping_list[0] # get first element
shopping_list[-1] # get last element
shopping_list[0:2] # get slice, left including, right excluding
len(shopping_list) # get lengh of a list
```

Out[65]:
```
4
```

Dict

In [66]:
```
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]} # variant 1
d2 = dict(a='some value', b=[1, 2, 3, 4]) # variant2
d1['c'] = False # add an item
d1['a'] # get a value, KeyError if key does not exist
d1.get('x', 'default value') # avoid KeyError, get default value if key does not
 exist
```

Out[66]:
```
'default value'
```

Control Structures and Indentation

- Blocks are structured by colon and indentation

In [67]:
```
shopping_list = [ 'milk', 'cheese', 'bread' ]
if not shopping_list:
    print('Nothing to buy today')
for item in shopping_list:
    print(f'Buy {item}')
```

```
Buy milk
Buy cheese
Buy bread
```

Functions & Methods

- Positional arguments, keyword arguments, default values
- Multiple return values (tulple)

In [1]:
```
def add_and_multiply(a, b, c=0):
  """
  Add and multiply some numbers together
  """
  return a + b + c, a * b * c


result1 = add_and_multiply(1, 2) # use positional arguments
result1 # is a tuple
sum1, product1 = result1 # unpack the tuple

sum2, product2 = add_and_multiply(c=4, a=1, b=2) # use keyword arguments and unp
ack the result tuple
product2
```

Out[1]:
```
8
```

Classes, Objects, Constructor

In [6]:
```
class C(object):
    def __init__(self, a=0, b=0):
        self.a = a
        self.b = b

    def get_sum(self):
        return self.a + self.b

c1 = C()
c2 = C(3,5)
c2.get_sum()
```

Out[6]:
```
8
```

Imports

- Non built-in modules must be imported
- Either module or single function/class or all

In [77]:
```
import math # import math module
from random import random # import only the random function from the random modu
le
from datetime import * # import all classes from datetime module (avoid)

math.pi
timedelta(seconds=5)
random()
```

Out[77]:
```
0.6991655016949093
```

Python Code Completion and Help in Jupyter

- Code completion: `Tab`
- Python docstring: `Shift+Tab` (repeated)
- Help
  - `?object`: docstring of the class or function
  - `??object`: source code of module or class or function
- Doesn't work so well for built-ins
- h overview of Jupyter short cuts

Exercise 2 - Python

- Goals:
  - Remember your Python skills
  - Getting used to write Python code in Jupyter
- Tasks:
  - Try code completion and help
    - `Tab`, `Shift+Tab`, ?python module or class or function
  - Strings
    - Concatenate two strings,
    - Concatenate a string and a number
  - list
    - Concatenate two lists
    - Remove the second element from the list
  - dict
    - Change a value in a existing dict
    - Get a list of all keys and a list of all values of a dict

Libraries for Data Analysis

Pandas

- Python library (can be used independent of Jupyter)
- Data structues and tools for data analysis (in-memory)
- Tabluar data and time series
- Homepage: https://pandas.pydata.org/ (https://pandas.pydata.org/)
- Documentation: https://pandas.pydata.org/pandas-docs/stable/ (https://pandas.pydata.org/pandas-docs/stable/)

Numpy

- Fast and efficient N-dimensional array object ndarray
- Functions for working with large arrays and matrices, linear algebra operations
- Used as container for pasing data betweend algorithms and libraries
- Homepage:http://www.numpy.org (http://www.numpy.org)

SciPy

- Collection of packages for different mathematical standard problems
  - stats: Probability distributions, various statistical tests, descriptive statistics
  - signal: Signal processing tools
  - linalg: Linear algebra routines and matrix decompositions
  - integrate: Numerical integration routines and differential equation solvers
- Homepage: https://www.scipy.org (https://www.scipy.org)

Visualization Libraries

- matplotlib (https://matplotlib.org (https://matplotlib.org))
  - Most popular visualization library in Python
  - Integrated in Pandas
- seaborn (https://seaborn.pydata.org (https://seaborn.pydata.org))
  - Based on matplotlib
  - Goal: Making prettier graphs easier
- bokeh (https://bokeh.pydata.org (https://bokeh.pydata.org))
  - Independent of matplotlib
  - Interactive graphics

Default imports

- Aliases `np`, `pd`, `plt` are very common.
- `%matplotlib inline` tells Jupyter to render plots inline
- `plt.rcParams["figure.figsize"] = [10,4]` makes the plots a bit larger

In [72]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import bokeh as bk
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10,4]
```

System check

Do you have a working installation of all required packages?