The Python Ecosystem for Data Analysis (Introduction)

Konstantin KNYAZKOV
Konstantin.knyazkov@globms.com

GMS
GLOBAL MARKET SOLUTIONS

Agenda

- **Python**, basic syntax;
- **Pandas**, **numpy**, DataFrame API;
- **IPython** environment, notebooks, architecture;
- **Visualization** and data analysis libraries (matplotlib, seaborn,);
- **Python + .Net**

Goals

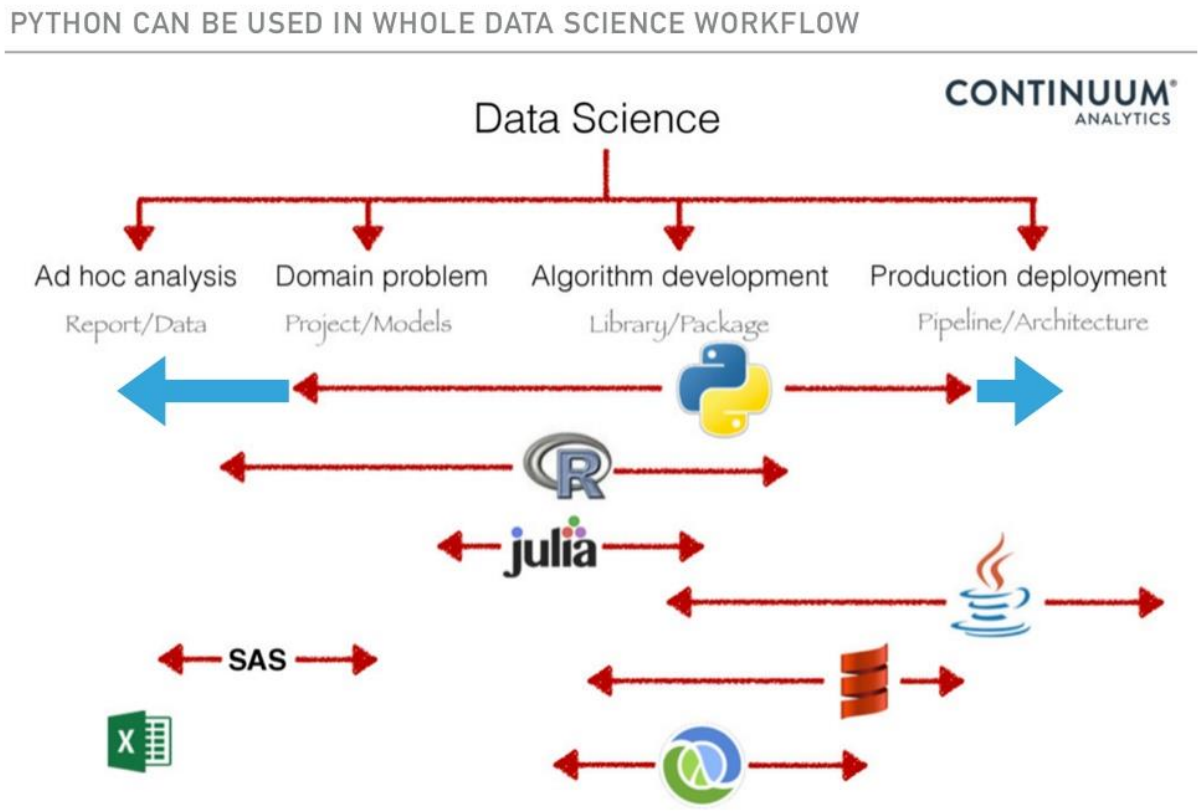
- Technical overview – to give a view of the technology;
- Help to not get confused with many tools and libraries;

PYTHON

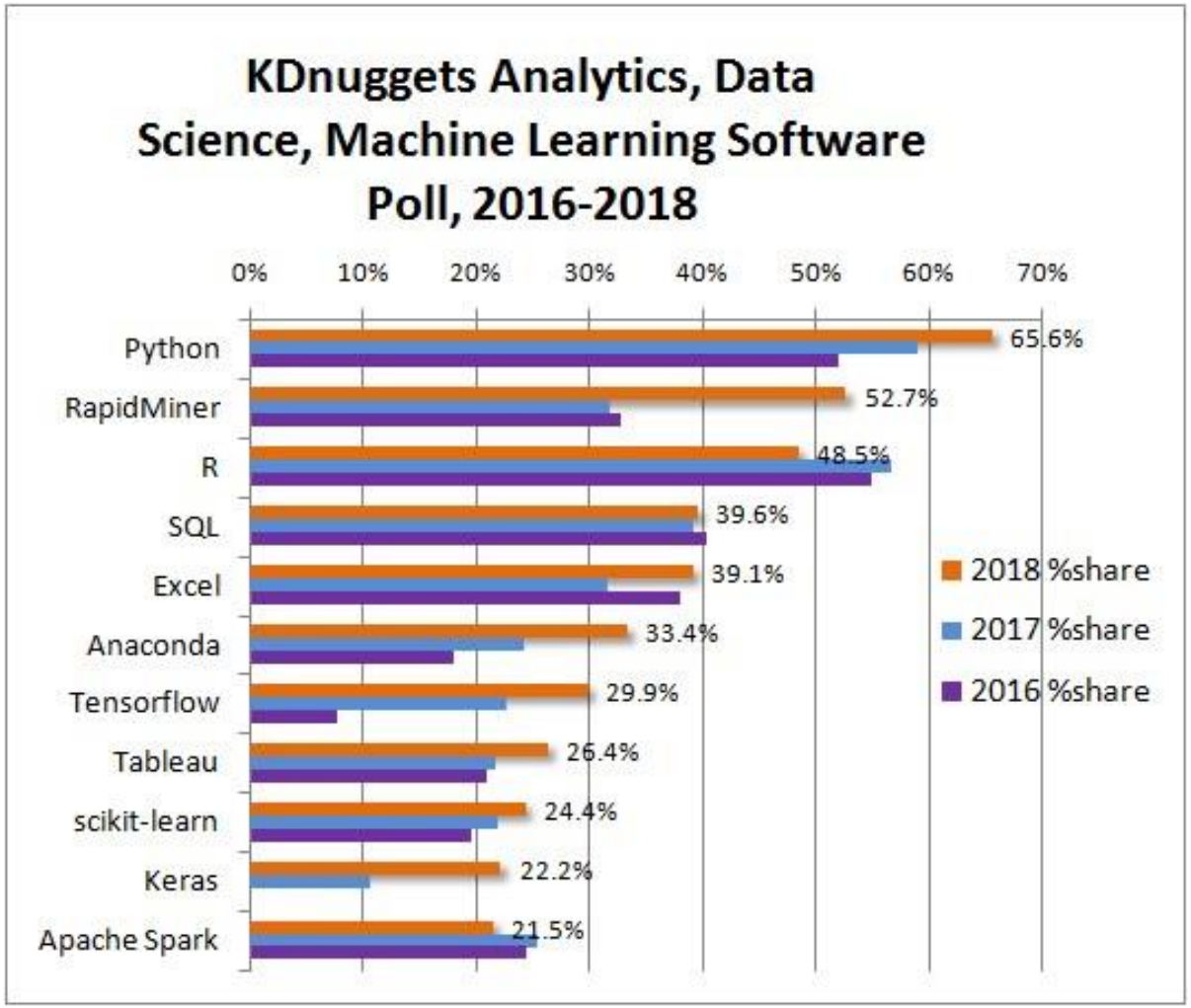
Motivation 1: Who uses Python?

Image Source: http://digiguyzz.com/technology/python-programming/python-an-introduction/

Motivation 2: World of Data Science



https://speakerdeck.com/chdoig/the-state-of-python-for-data-science-pyss-2015?slide=22



Motivation 3: Personal Perception (~)

Experience

Applied python in several commercial or academia projects of different types (GUI, data analysis, web applications, data acquisition, as configuration system, machine learning).

Pros

- Useful as a default scripting language;
- Never forget syntax;
- Good for prototyping, fast enough;
- Code is always aligned;
- Interactive features;
- Large amount of libraries, huge community;
- Glue language – good for integration of different components;
- Good integration with C/Fortran;
- Python is data-handling friendly.

Cons

- Hard to create and support large enterprise-level systems (GIL, dynamic typing);
- Poor multithreading;
- Interpretable – may be slower traditional compiled code;
- Some not cute peculiarities (one-line lambdas, __init__);

History

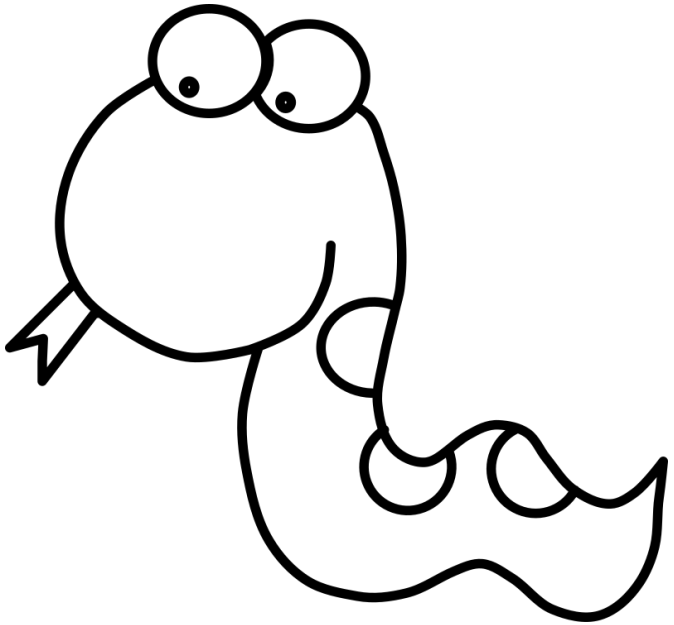
- Created by Guido van Rossum as a successor to the ABC language;
- Released in 1991 (4-years before Java);
- Interesting fact: the famous MIT course «Structure and Interpretation of Computer Programs» (SICP) has changed the language from Scheme to Python;

Zen of Python

- Beautiful is better than ugly.
- **Explicit is better than implicit.**
- **Simple is better than complex.**
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- **Readability counts.**
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- **There should be one-- and preferably only one --obvious way to do it.**
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Dossier

- Multiple Programming Paradigms;
  - Object Oriented Programming
  - Structured Programming
  - Functional Programming
  - Aspect Oriented Programming
- Type system;
  - Dynamic typing
  - Duck typing
  - Strongly typed
  - ** There are type annotations, see* https://docs.python.org/3/library/typing.html
- Compiled to bytecode*;
- Exception-based error handling.
- Automatic memory management;



PYTHON SYNTAX

```python
#
# Monte Carlo valuation of European call option
# in Black-Scholes-Merton model
# bsm_mcs_euro.py
#
import numpy as np

# Parameter Values
S0 = 100.  # initial index level
K = 105.  # strike price
T = 1.0  # time-to-maturity
r = 0.05  # riskless short rate
sigma = 0.2  # volatility

I = 100000  # number of simulations

# Valuation Algorithm
z = np.random.standard_normal(I)  # pseudorandom numbers
ST = S0 * np.exp((r - 0.5 * sigma ** 2) * T + sigma * np.sqrt(T) * z)
  # index values at maturity
hT = np.maximum(ST - K, 0)  # inner values at maturity
C0 = np.exp(-r * T) * np.sum(hT) / I  # Monte Carlo estimator

# Result Output
print "Value of the European Call Option %5.3f" % C0
```

1. Draw I (pseudo)random numbers $z(i)$, $i \in \{1, 2, \ldots, I\}$, from the standard normal distribution.
2. Calculate all resulting index levels at maturity $S_T(i)$ for given $z(i)$ and Equation 1-1.
3. Calculate all inner values of the option at maturity as $h_T(i) = \max(S_T(i) - K, 0)$.
4. Estimate the option present value via the Monte Carlo estimator given in Equation 1-2.

$$S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}z\right)$$

$$C_0 \approx e^{-rT}\frac{1}{I}\sum_{I} h_T(i)$$

Source: https://www.oreilly.com/library/view/python-for-finance/9781491945360/ch01.html

Whitespaces

- Python is indentation-sensitive, it means that indentation level semantically defines the program structure;
- **Recommendation:** always configure an editor to **replace TAB with fixed number of SPACES**;

Syntax Examples

```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

DEMO: PYTHON SYNTAX (PYTHON.ipynb)

Basic Types

In [2]:
```
3979866786756756576879797987679683278235 89
```
Out[2]: 3979866786756756576879797987679683278235 89

In [52]:
```
4.5
```
Out[52]: 4.5

In [53]:
```
"aaa"
```
Out[53]: 'aaa'

In [54]:
```
[3, 4, 5]
```
Out[54]: [3, 4, 5]

In [55]:
```
{"a": 2, "f": "s"}
```
Out[55]: {'a': 2, 'f': 's'}

In [56]:
```
(3, len(d))
```
Out[56]: (3, 8)

Group Assignment

In [58]:
```
a, b, c = 1, 2, 3
a, b, c
```
Out[58]: (1, 2, 3)

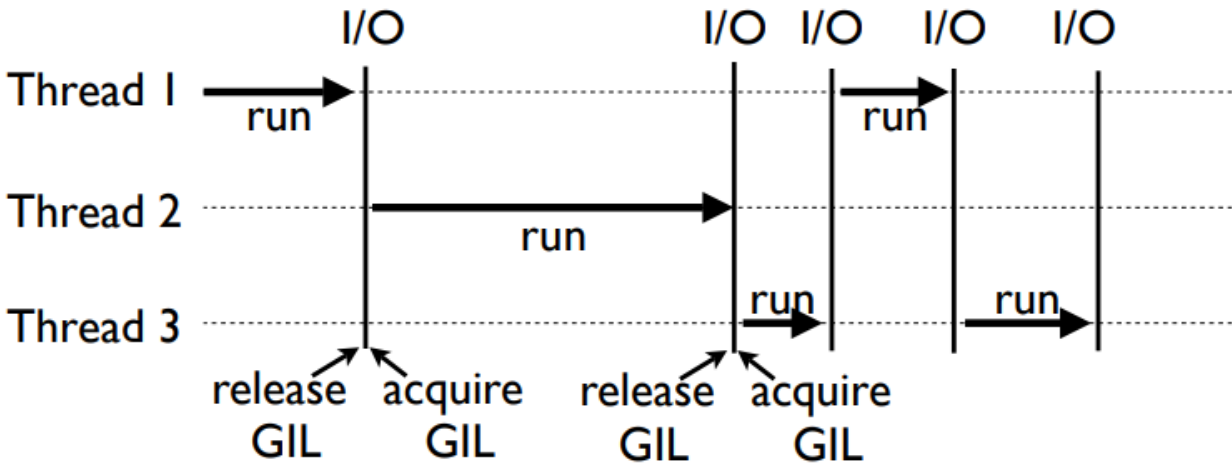
Implementations

- CPython is the reference implementation of Python, written in C. It compiles Python code to intermediate bytecode which is then interpreted by a virtual machine. CPython provides the highest level of compatibility with Python packages and C extension modules.
- PyPy is a Python interpreter implemented in a restricted statically-typed subset of the Python language called RPython. The interpreter features a just-in-time compiler and supports multiple back-ends (C, CLI, JVM).
- Jython is a Python implementation that compiles Python code to Java bytecode which is then executed by the JVM (Java Virtual Machine). Additionally, it is able to import and use any Java class like a Python module.
- Python for .NET is a package which provides near seamless integration of a natively installed Python installation with the .NET Common Language Runtime (CLR). This is the inverse approach to that taken by IronPython (see above), to which it is more complementary than competing with.
- IronPython is an implementation of Python for the .NET framework. It can use both Python and .NET framework libraries, and can also expose Python code to other languages in the .NET framework.

https://docs.python-guide.org/starting/which-python/

Multithreading and GIL

- Cooperative multitasking;
- A **global interpreter lock (GIL)** is a synchronization mechanism that only one native thread can execute at a time. Exactly one thread can be executed at a time.
  - One thread runs Python, while N others sleep or await I/O
- Reasons for using GIL include:
  - increased speed of single-threaded programs (no necessity to acquire or release locks on all data structures separately),
  - Simplifies many low-level details (memory management, callouts to C extensions, etc.)
  - easy integration of C libraries that usually are not thread-safe,
  - ease of implementation (having a single GIL is much simpler to implement than a lock-free interpreter or one using fine-grained locks).



See multiprocessing:
https://docs.python.org/2/library/multiprocessing.html

https://www.dabeaz.com/python/UnderstandingGIL.pdf

Python 2 vs Python 3

- Python 2.7 end of life 2020
- Reason – mainly because of encodings
- Backward incompatibilities:
  - Print and exec become functions
  - All classes are new-style
  - Massive usage of generators intead of lists
  - All text is Unicode, encoded text is bytes
  - asyncio
  - Exception chaining
  - and others..
- **Recommendation:** use Python 3.x (~)

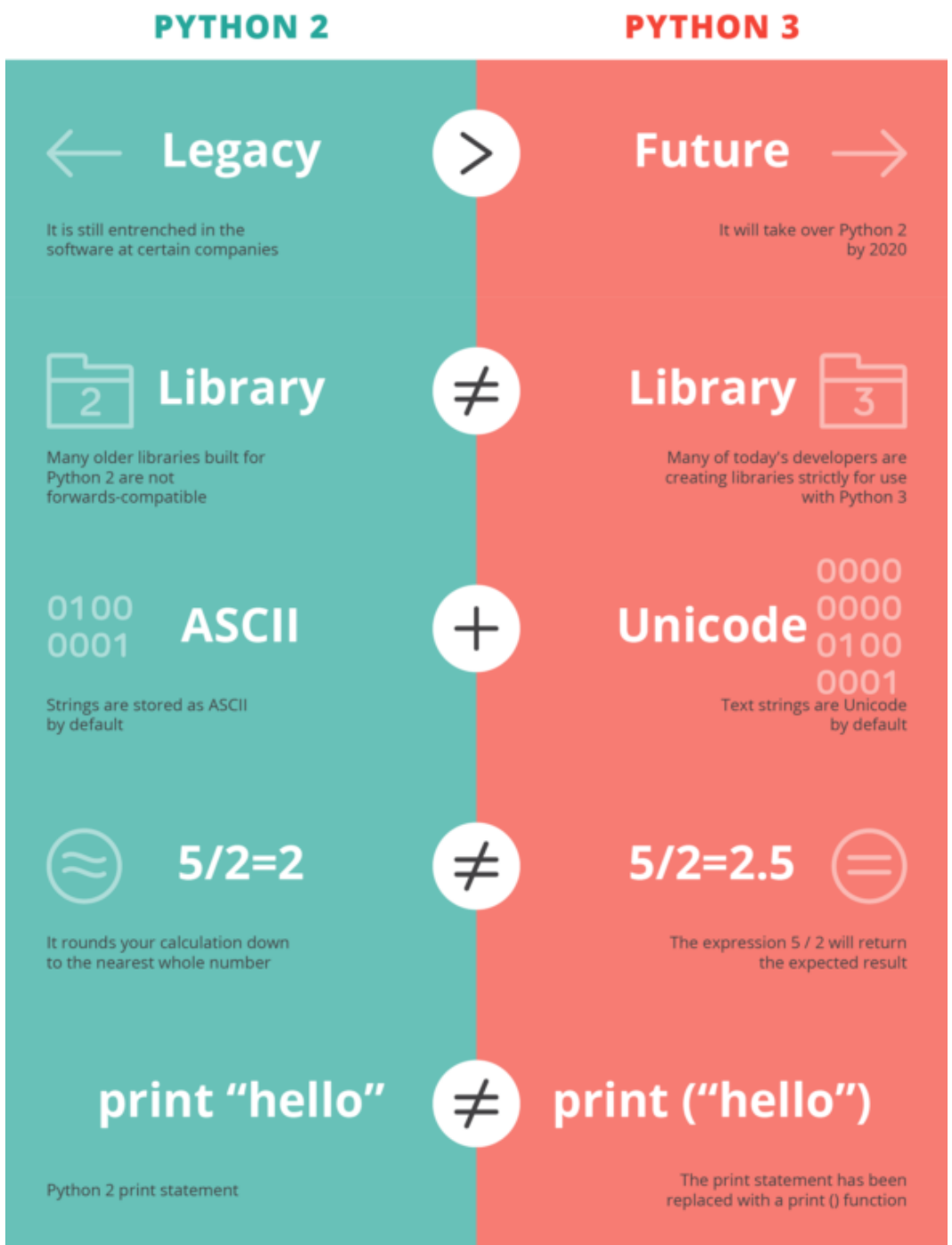
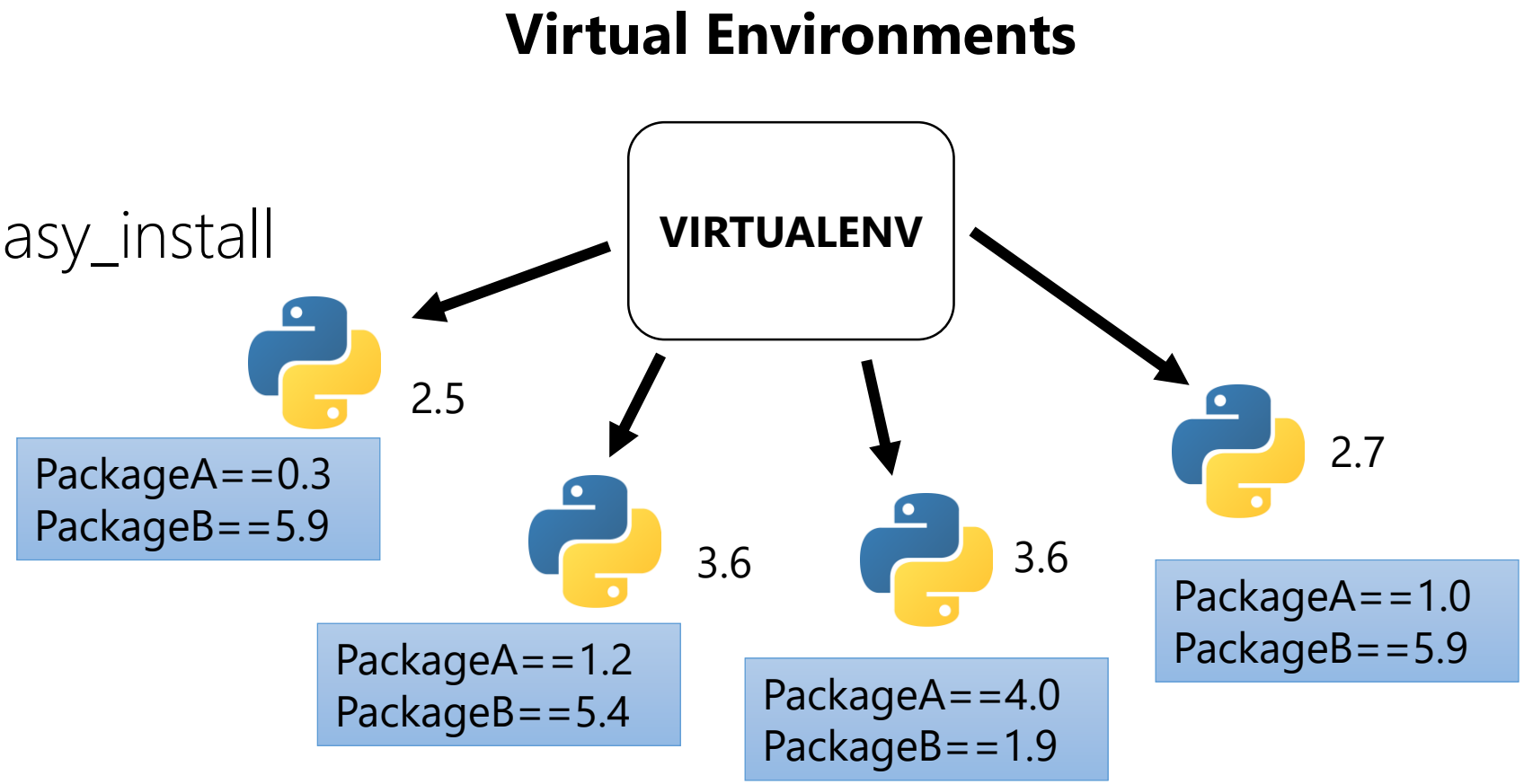


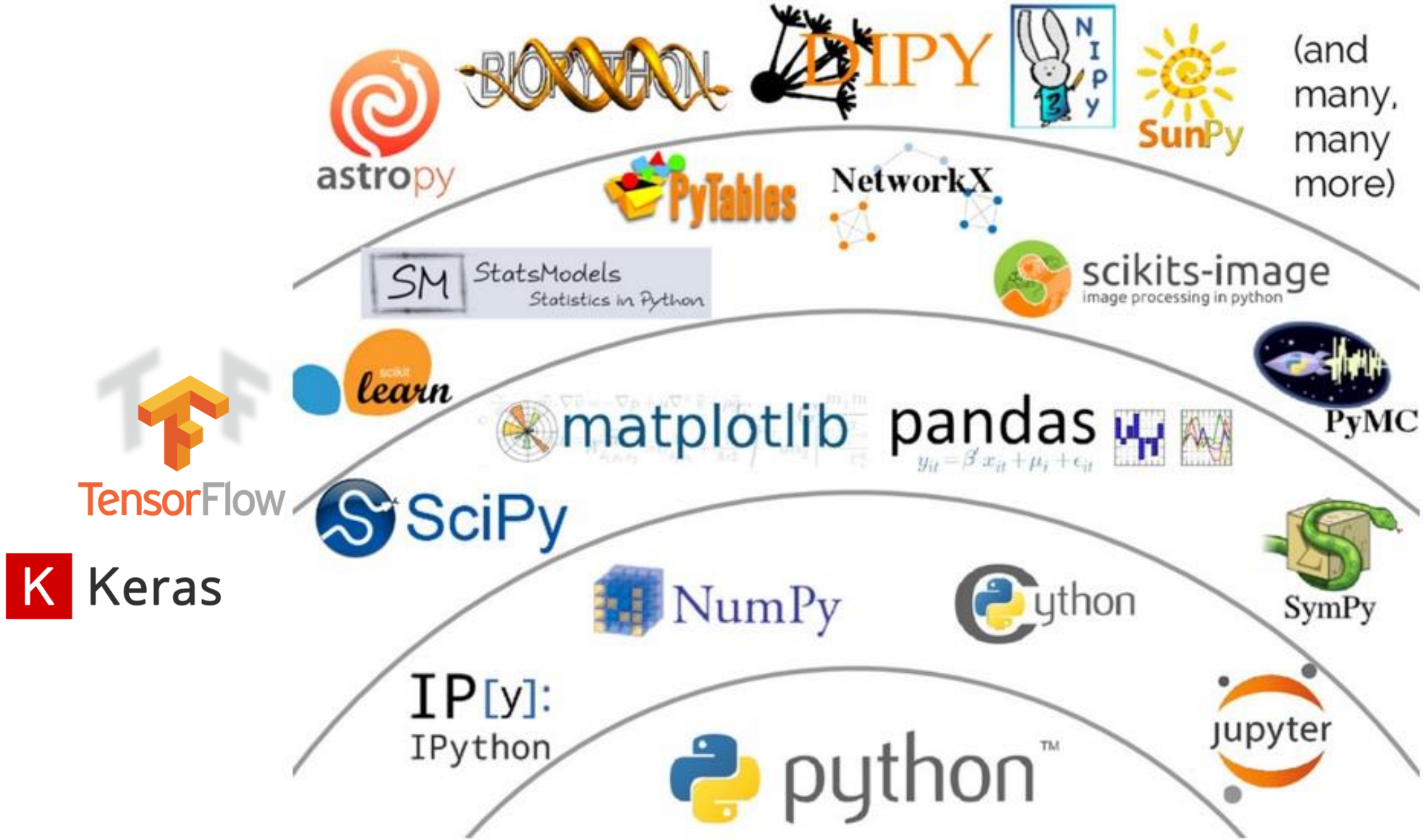
Image Source: https://learntocodewith.me/programming/python/python-2-vs-python-3/

Package management

- pip install
- pip uninstall
- pip freeze
- pip search
- Pip is better than easy_install (~)



DATA SCIENCE STACK



https://speakerdeck.com/jakevdp/the-state-of-the-stack-scipy-2015-keynote?slide=8

IPYTHON

jupyter REPORT-MACRO-COMPARE Last Checkpoint: 03/19/2018 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

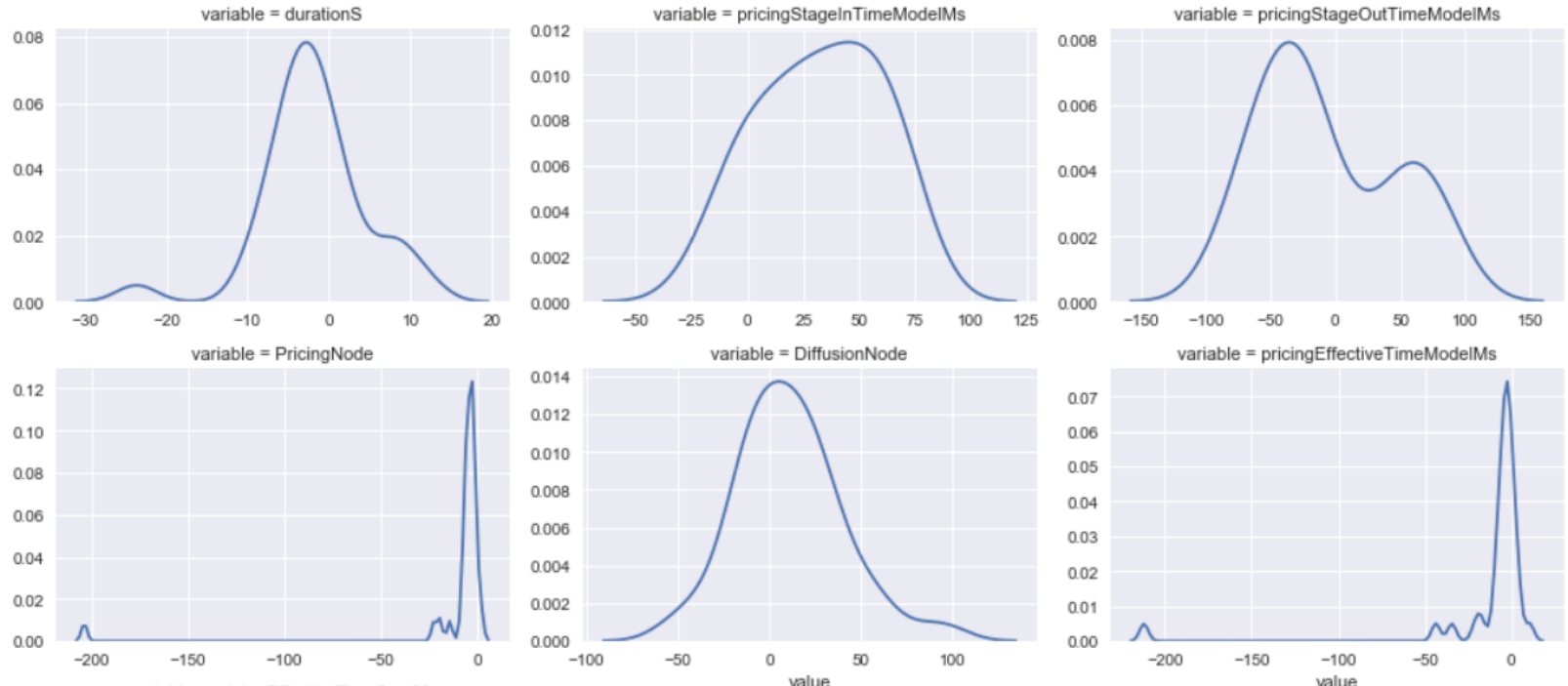
Code

101	1	100	PricingNode	1.288126
36	1000	6	pricingStageInTimeModelMs	-2.594400
173	100	200	pricingEffectiveTimeModelMs	-4.525926
5	1	100	durationS	12.178795
139	1	200	DiffusionNode	93.205145

In [10]:

```
figSize(7.0, 4)
g = sns.FacetGrid(M, col="variable", size=3, aspect=1.5, col_wrap=3, sharey=False, sharex=False)
fg = g.map(sns.kdeplot, "value")

#sns.kdeplot(D["pricingEffectiveTimeModelMs"].values)
```



IPython

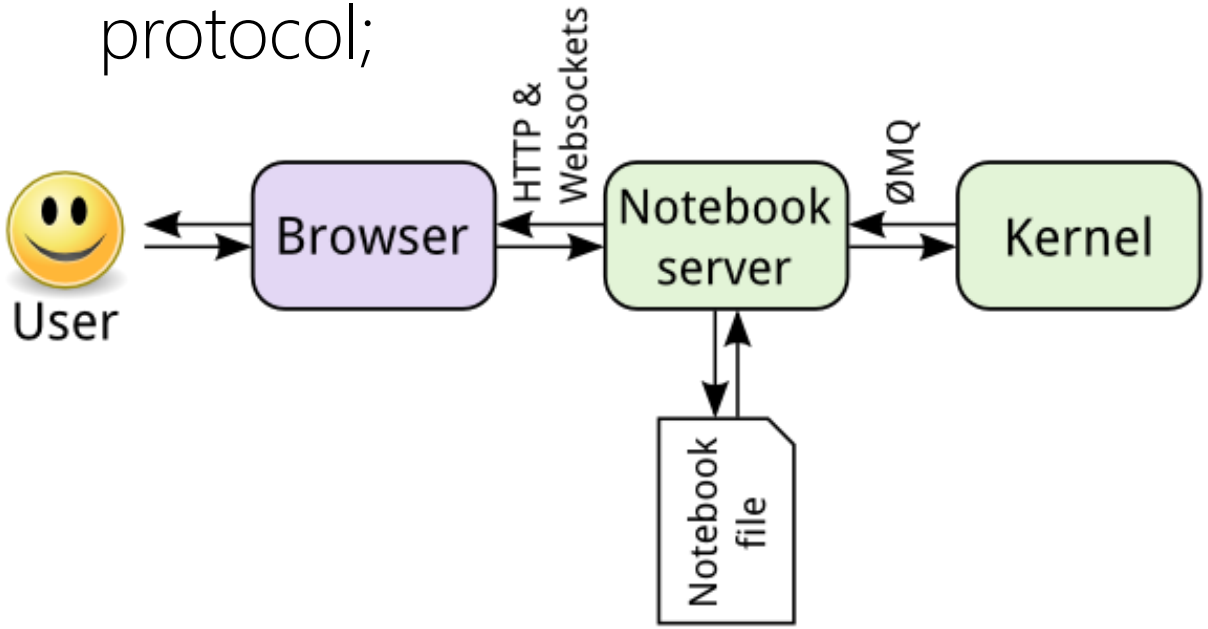
- Interactive Python: 2001; https://ipython.org/
- Interactive environment;
- Notebooks – executable books or reproducible publications. Example (Click Me): https://notebooks.azure.com/losc/projects/tutorials/html/LOSC_Event_tutorial.ipynb
- Examples: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks

Jupyter

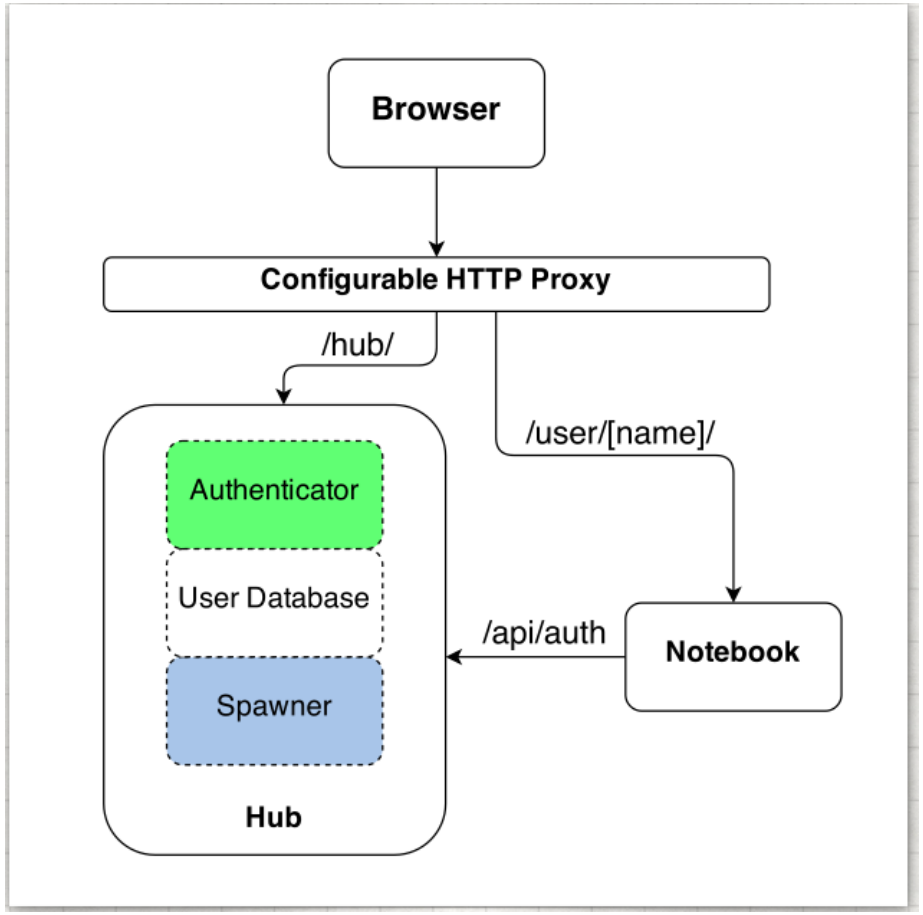
- Language-agnostic interactive environment;
- There are backends for many languages;
- Good example is PySpark;
- Jupyter; http://jupyter.org/
- JupyterHub; https://jupyterhub.readthedocs.io/
- JupyterLab: http://jupyterlab.readthedocs.io/

Jupyter notebook architecture

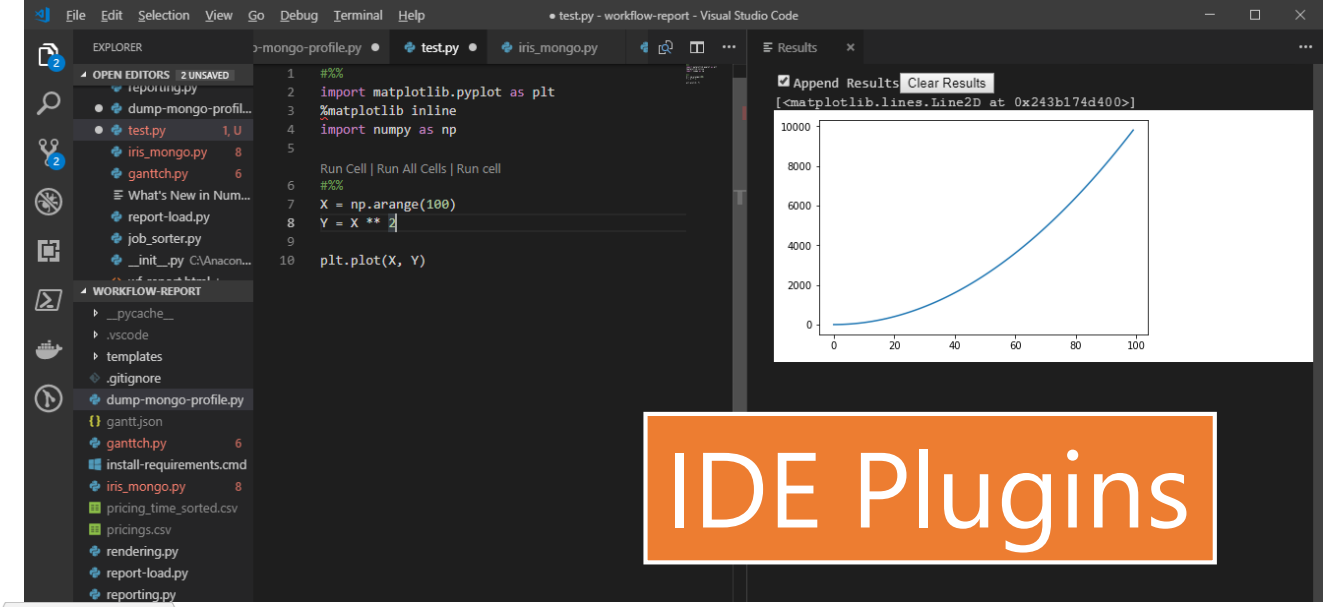
- A Python kernel as a backend – saves the state of interpreter;
- Many frontends;
- ZeroMQ as a messaging protocol;

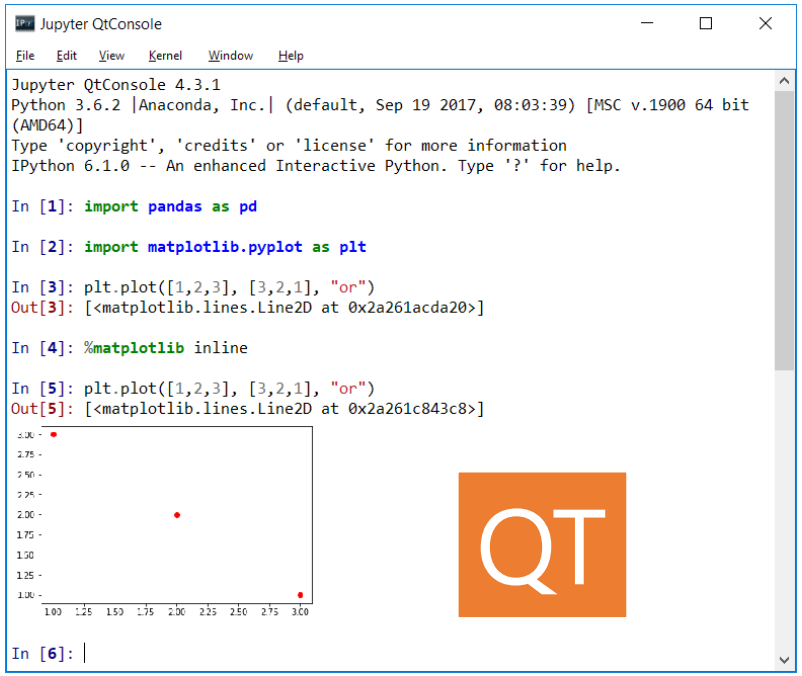


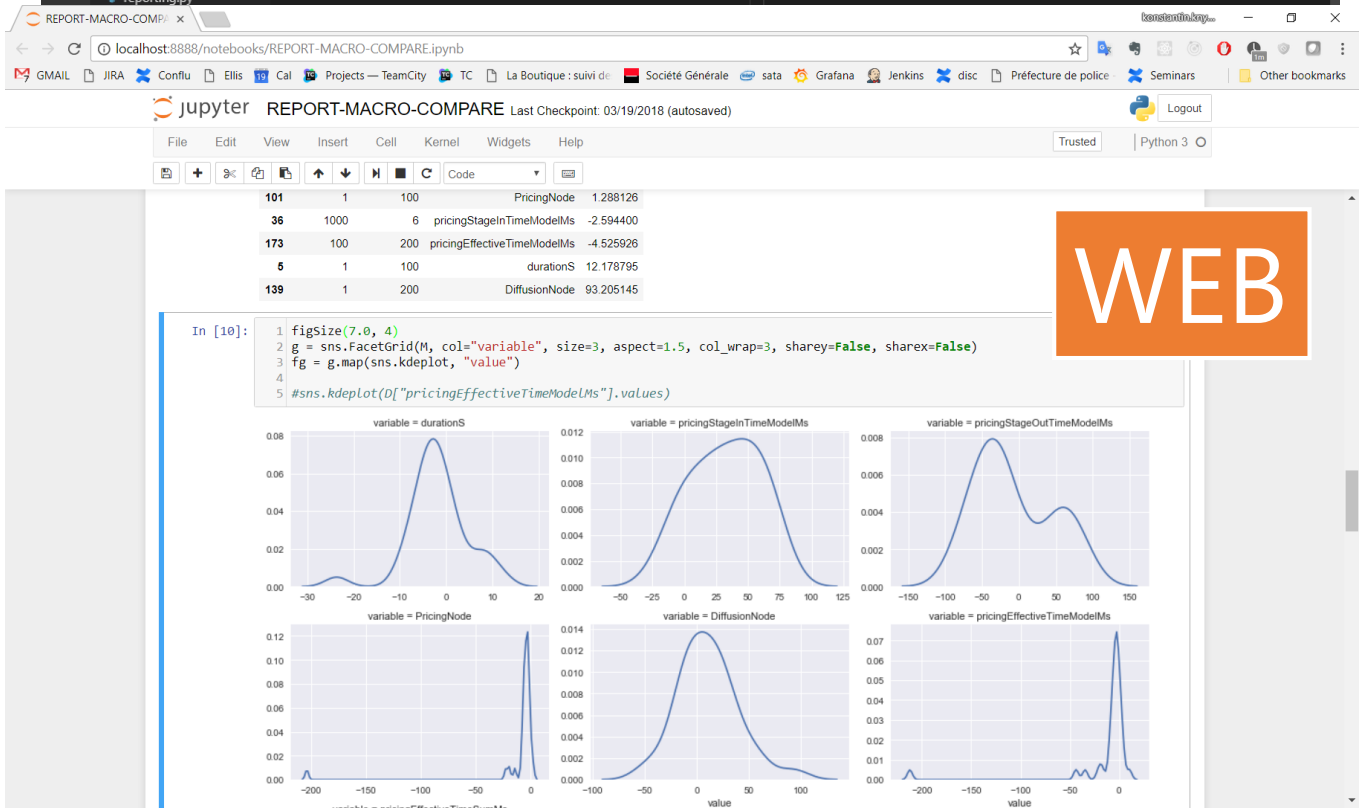
JupyterHub is more suitable for enterprise usage




IPython: C:kkn/Documents
In [7]: import numpy as np
In [8]: np.random.
np.random.absolute_import
np.random.bench
np.random.beta
np.random.binomial
np.random.bytes
np.random.chisquare
np.random.choice
np.random.dirichlet
np.random.division
np.random.exponential
np.random.f
np.random.gamma
np.random.geometric
np.random.get_state
np.random.gumbel
np.random.hypergeometric
np.random.info
np.random.laplace
np.random.Lock
np.random.logistic
np.random.lognormal
np.random.logseries
np.random.mtrand
np.random.multinomial
np.random.multivariate_normal
np.random.negative_binomial
np.random.noncentral_chisquare
np.random.noncentral_f
np.random.normal
np.random.np
np.random.operator
np.random.pareto
np.random.permutation
np.random.poisson
np.random.power
np.random.print_function
np.random.rand
np.random.randint
np.random.randn
np.random.random
np.random.random_integers
np.random.random_sample
np.random.RandomState
np.random.ranf
np.random.rayleigh
np.random.sample
np.random.seed
np.random.set_state
np.random.shuffle
np.random.standard_cauchy
np.random.standard_exponential
np.random.standard_gamma
np.random.standard_normal
np.random.standard_t
np.random.test
np.random.triangular
np.random.uniform
np.random.vonmises
np.random.wald
np.random.warnings
np.random.weibull
np.random.zipf
Console


File Edit Selection View Go Debug Terminal Help
test.py - workflow-report - Visual Studio Code
EXPLORER
OPEN EDITORS 2 UNSAVED
dump-mongo-profil...
test.py 1, U
iris_mongo.py 8
ganttch.py 6
What's New in Num...
report-load.py
job_sorter.py
__init__.py C:\Anacon...
WORKFLOW-REPORT
__pycache__
.vscode
templates
.gitignore
dump-mongo-profile.py
gantt.json
ganttch.py 6
install-requirements.cmd
iris_mongo.py 8
pricing_time_sorted.csv
pricings.csv
rendering.py
report-load.py
reporting.py
#%%
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
Run Cell | Run All Cells | Run cell
#%%
X = np.arange(100)
Y = X ** 2
plt.plot(X, Y)
Results
Append Results
Clear Results
[<matplotlib.lines.Line2D at 0x243b174d400>]
IDE Plugins


Jupyter QtConsole
File Edit View Kernel Window Help
Jupyter QtConsole 4.3.1
Python 3.6.2 |Anaconda, Inc.| (default, Sep 19 2017, 08:03:39) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: import pandas as pd
In [2]: import matplotlib.pyplot as plt
In [3]: plt.plot([1,2,3], [3,2,1], "or")
Out[3]: [<matplotlib.lines.Line2D at 0x2a261acda20>]
In [4]: %matplotlib inline
In [5]: plt.plot([1,2,3], [3,2,1], "or")
Out[5]: [<matplotlib.lines.Line2D at 0x2a261c843c8>]
In [6]:
QT


localhost:8888/notebooks/REPORT-MACRO-COMPARE.ipynb
jupyter REPORT-MACRO-COMPARE Last Checkpoint: 03/19/2018 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
101 1 100 PricingNode 1.288126
36 1000 6 pricingStageInTimeModelMs -2.594400
173 100 200 pricingEffectiveTimeModelMs -4.525926
5 1 100 durationS 12.178795
139 1 200 DiffusionNode 93.205145
In [10]:
figSize(7.0, 4)
g = sns.FacetGrid(M, col="variable", size=3, aspect=1.5, col_wrap=3, sharey=False, sharex=False)
fg = g.map(sns.kdeplot, "value")
#sns.kdeplot(D["pricingEffectiveTimeModelMs"].values)
variable = durationS
variable = pricingStageInTimeModelMs
variable = pricingStageOutTimeModelMs
variable = PricingNode
variable = DiffusionNode
variable = pricingEffectiveTimeModelMs
value
WEB

NUMPY

Numpy

NumPy is a Python package to efficiently deal with large datasets **in-memory**, providing containers for homogeneous data, heterogeneous data, and string arrays.

- In-memory
- Efficient
- Applications:
  - Image processing
  - Signal processing
  - Linear algebra

- All operations are vectorized;
- Slicing operations do not make copies – they return views on the original array;

Main data types

- Integer: int8, int16, int32, int64, uint8, …
- Float: float16m float32, float64,…
- Complex: complex64, complex128,…
- Boolean: bool8
- Date: datetime64, timedelte64,..
- Unicode string
- Default: float64

Trigonometric functions

sin, cos, tan, arcsin, arccos, arctan, hypot, arctan2, degrees, radians, unwrap, deg2rad, rad2deg

Hyperbolic functions

sinh, cosh, tanh, arcsinh, arccosh, arctanh

Rounding

around, round_, rint, fix, floor, ceil, trunc

Sums, products, differences

prod, sum, nansum, cumprod, cumsum, diff, ediff1d, gradient, cross, trapz

Exponents and logarithms

exp, expm1, exp2, log, log10, log2, log1p, logaddexp, logaddexp2

Other special functions

i0, sinc

Floating point routines

signbit, copysign, frexp, ldexp

Arithmetic operations

add, reciprocal, negative, multiply, divide, power, subtract, true_divide, floor_divide, fmod, mod, modf, remainder

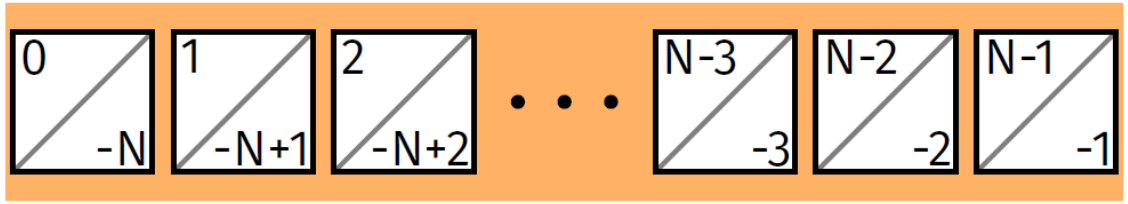
Handling complex numbers

angle, real, imag, conj

Miscellaneous

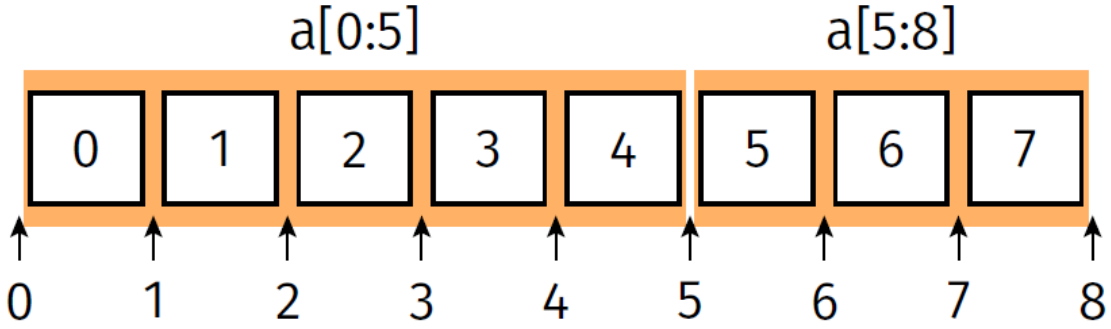
convolve, clip, sqrt, square, absolute, fabs, sign, maximum, minimum, fmax, fmin, nan_to_num, real_if_close, interp

Indexing



Slicing

basic syntax: `[start:stop:step]`



- if `step=1`
  - slice contains the elements `start` to `stop-1`
  - slice contains `stop-start` elements
- `start`, `stop`, and also `step` can be negative
- default values:
  - `start` 0, i.e. starting from the first element
  - `stop` N, i.e up to and including the last element
  - `step` 1

a[:3, :5]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

a[-3:, -3:]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

a[1, 3:6]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

a[1::2, ::3]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

a[a % 3 == 0]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

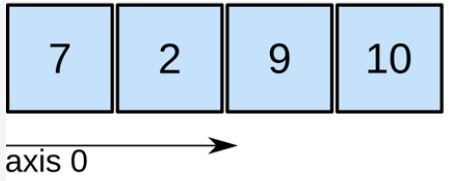
a[(1, 1, 2, 2, 3, 3), (3, 4, 2, 5, 3, 4)]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Vector shapes

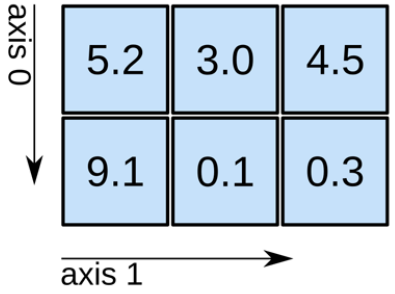
```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
```

1D array



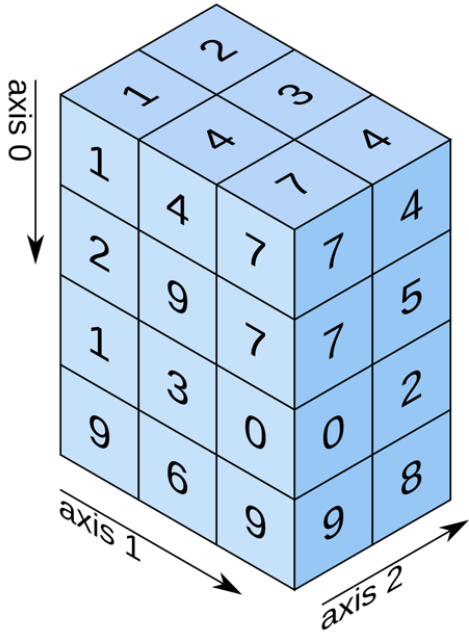
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

DEMO:
NUMPY
(NUMPY.ipynb)

In [1]:
```
import numpy as np
```

In [2]:
```
A = np.array([3, 3, 3, 4, 5])
B = np.array([1, 2, 3, 4, 5])
A, B
```

Out[2]:
```
(array([3, 3, 3, 4, 5]), array([1, 2, 3, 4, 5]))
```

In [7]:
```

```

Out[7]:
```
array([ 0.14112001,  0.14112001,  0.14112001, -0.7568025 , -0.95892427])
```

In [4]:
```
A.shape
```

Out[4]:
```
(5,)
```

In [5]:
```
A * 5
```

Out[5]:
```
array([15, 15, 15, 20, 25])
```

In [6]:
```
A * B
```

Out[6]:
```
array([ 3,  6,  9, 16, 25])
```

In [7]:
```
np.dot(A, B)
```

Out[7]:
```
59
```

In [8]:
```
np.sin(A)
```

Out[8]:
```
array([ 0.14112001,  0.14112001,  0.14112001, -0.7568025 , -0.95892427])
```

In [8]:
```
m = np.random.rand(3,4)
c = np.random.rand(3)
m, c
```

Out[8]:
```
(array([[ 0.4521938 ,  0.79356996,  0.38956654,  0.57762292],
        [ 0.6716598 ,  0.53219249,  0.31114092,  0.18599609],
        [ 0.76728194,  0.83662351,  0.6775626 ,  0.86567498]]),
 array([ 0.34750787,  0.35770428,  0.64900755]))
```

DEMO:
NUMPY PERF
(NUMPY-VS-CS.ipynb)

CSharp

In [1]:
```
%cd cs
```

```
C:\Workspace\playground\PythonPresentation\ipython\cs
```

In [2]:
```
!dotnet new console --force
```

```
Getting ready...
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on C:\Workspace\playground\PythonPresentation\ipython\cs\cs.csproj...
  Restoring packages for C:\Workspace\playground\PythonPresentation\ipython\cs\cs.csproj...
  Generating MSBuild file C:\Workspace\playground\PythonPresentation\ipython\cs\obj\cs.csproj.nuget.g.props.
  Restore completed in 604.7 ms for C:\Workspace\playground\PythonPresentation\ipython\cs\cs.csproj.

Restore succeeded.
```

In [3]:
```
!dotnet add package MathNet.Numerics -v 4.4.0 -s https://api.nuget.org/v3/index.json
```

```
  Writing C:\Users\kkn\AppData\Local\Temp\tmp22FC.tmp
info : Adding PackageReference for package 'MathNet.Numerics' into project 'C:\Workspace\playground\PythonPresentation\ipython
\cs\cs.csproj'.
log  : Restoring packages for C:\Workspace\playground\PythonPresentation\ipython\cs\cs.csproj...
info : Package 'MathNet.Numerics' is compatible with all the specified frameworks in project 'C:\Workspace\playground\PythonPre
sentation\ipython\cs\cs.csproj'.
info : PackageReference for package 'MathNet.Numerics' version '4.4.0' added to file 'C:\Workspace\playground\PythonPresentatio
n\ipython\cs\cs.csproj'.
info : Committing restore...
info : Writing lock file to disk. Path: C:\Workspace\playground\PythonPresentation\ipython\cs\obj\project.assets.json
log  : Restore completed in 511.57 ms for C:\Workspace\playground\PythonPresentation\ipython\cs\cs.csproj.
```

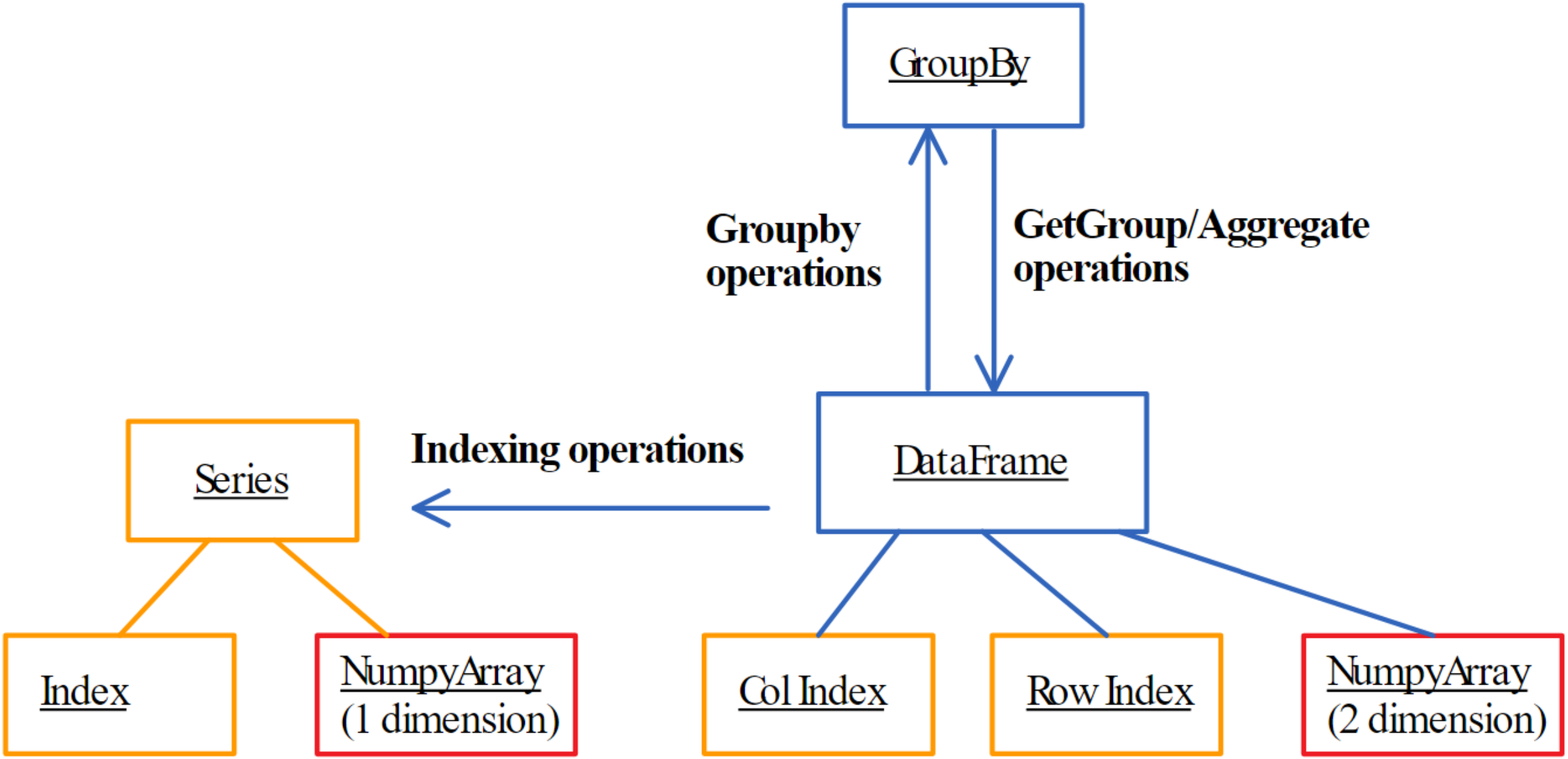
In [10]:
```
%%file Program.cs
using System;
using System.Diagnostics;
using System.Threading;
using MathNet.Numerics.LinearAlgebra;
using MathNet.Numerics.LinearAlgebra.Double;

public class Program {
    public static void Main(string[] args){
```

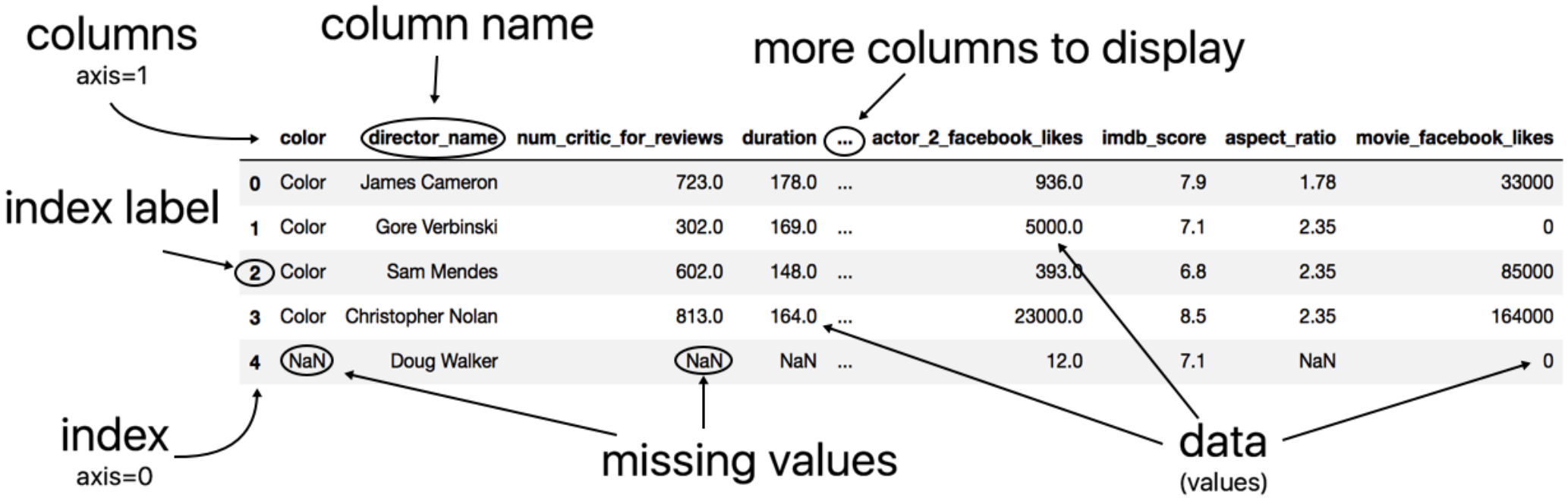
PANDAS

Pandas

- Defines tabular data types: database-like tables, with labelled rows and columns;
- API consisting of only two main entities (DataFrame and Series) is quite simple but powerful;
- Data consolidation and data integration: remove duplicates, clean data, manage missing values, indexing;
- Summarization: create pivot tables;
- In-memory SQL-like operations: join, group-by;
- Date handling;
- Easy visualization;



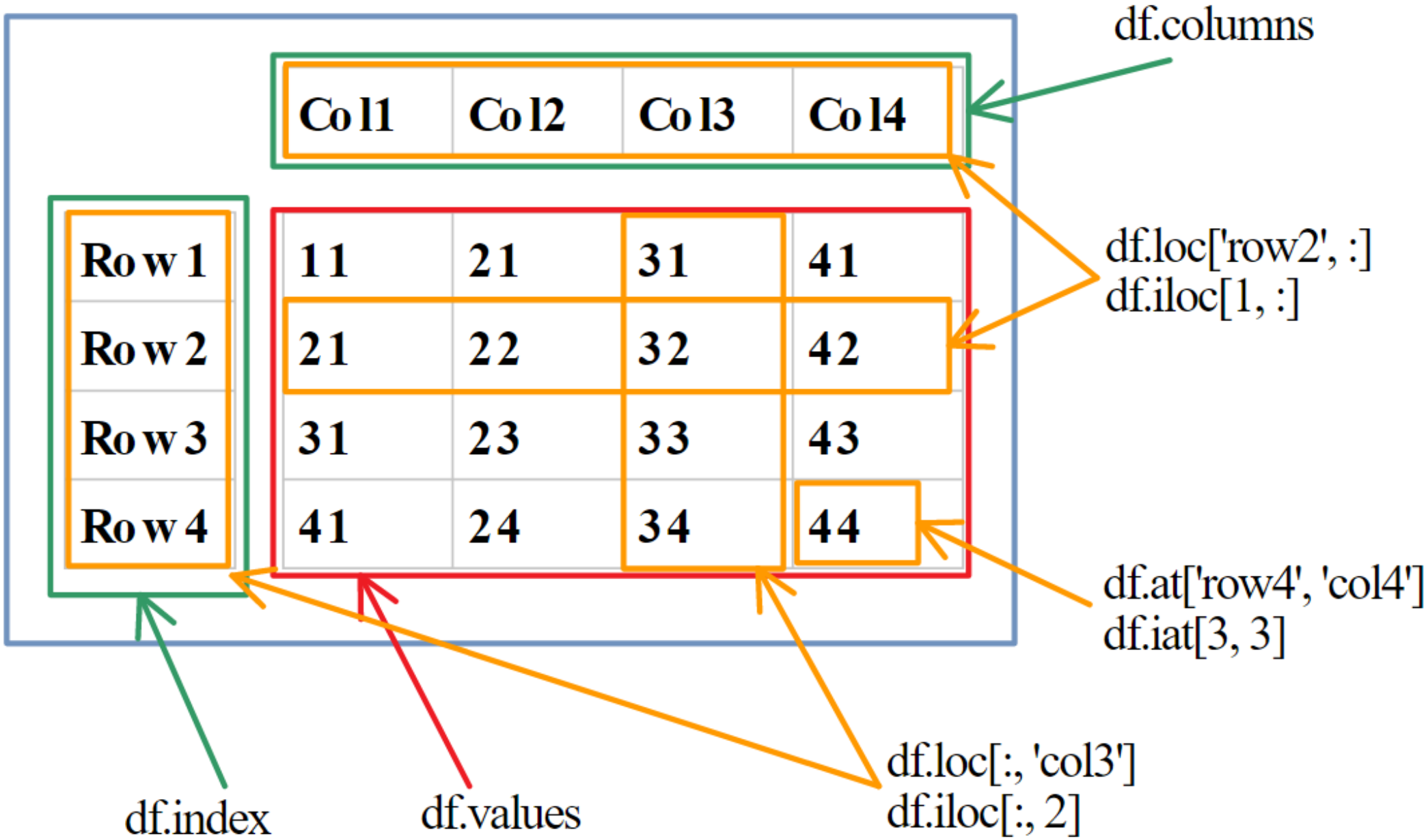
https://candide-guevara.github.io/diagram/python/2015/05/23/pandas-api.html



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

- DF is composed of: index, columns, data (values);
- Index = ordered labels;
- Each row and column has a label;

Pandas Dataframe

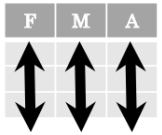


https://candide-guevara.github.io/diagram/python/2015/05/23/pandas-api.html

Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

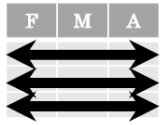
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



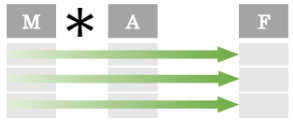
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```

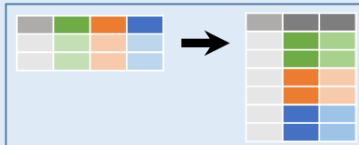
Create DataFrame with a MultiIndex

Method Chaining

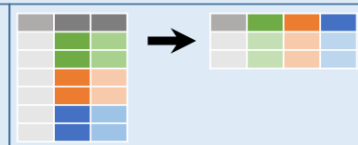
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
      .rename(columns={
              'variable' : 'var',
              'value' : 'val'})
      .query('val >= 200')
    )
```

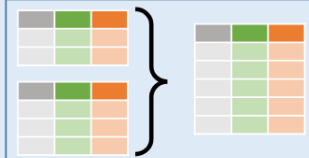
Reshaping Data – Change the layout of a data set



pd.melt(df)
Gather columns into rows.



df.pivot(columns='var', values='val')
Spread rows into columns.



pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

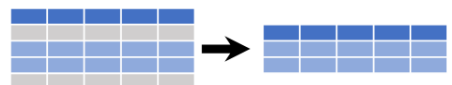
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

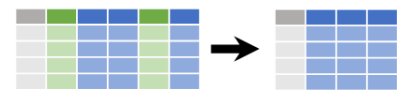
df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)



df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

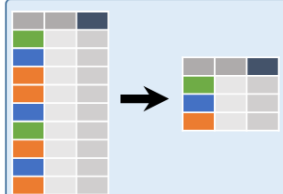
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

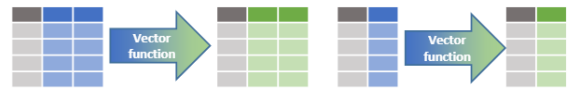
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

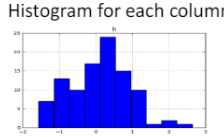
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

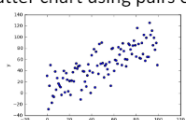
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

DEMO:
PANDAS
(PANDAS.ipynb)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
def figSize(wx,wy):
    matplotlib.rcParams['figure.figsize'] = (wx, wy)
import seaborn as sns
```

Simple Dataframes

In [2]:
```
WINE_DATA="winemag-data_first150k.csv"
```

In [8]:
```
df_wine = pd.read_csv(WINE_DATA, header=0, index_col=0)
```

In [11]:
```
df_wine
```

	Spain	Ripe aromas of fig, blackberry and cassis are ...	Carodorum Selección Especial Reserva	96	110.0	Northern Spain	Toro	NaN	Tinta de Toro	Carmen Rodríguez
2	US	Mac Watson honors the memory of a wine once ma...	Special Selected Late Harvest	96	90.0	California	Knights Valley	Sonoma	Sauvignon Blanc	Macauley
3	US	This spent 20 months in 30% new French oak, an...	Reserve	96	65.0	Oregon	Willamette Valley	Willamette Valley	Pinot Noir	Ponzi
4	France	This is the top wine from La Bégude, named aft...	La Brûlade	95	66.0	Provence	Bandol	NaN	Provence red blend	Domaine de la Bégude
5	Spain	Deep, dense and pure from the opening bell, th...	Numanthia	95	73.0	Northern Spain	Toro	NaN	Tinta de Toro	Numanthia
6	Spain	Slightly gritty black-fruit aromas include a s...	San Román	95	65.0	Northern Spain	Toro	NaN	Tinta de Toro	Maurodos
7	Spain	Lush cedary black-fruit aromas are luxe and of...	Carodorum Único Crianza	95	110.0	Northern Spain	Toro	NaN	Tinta de Toro	Bodega Carmen Rodríguez
8	US	This re-named vineyard was formerly bottled as...	Silice	95	65.0	Oregon	Chehalem Mountains	Willamette Valley	Pinot Noir	Bergström

In [5]:
```
#!pip install qgrid
import qgrid
```

VISUALIZATION

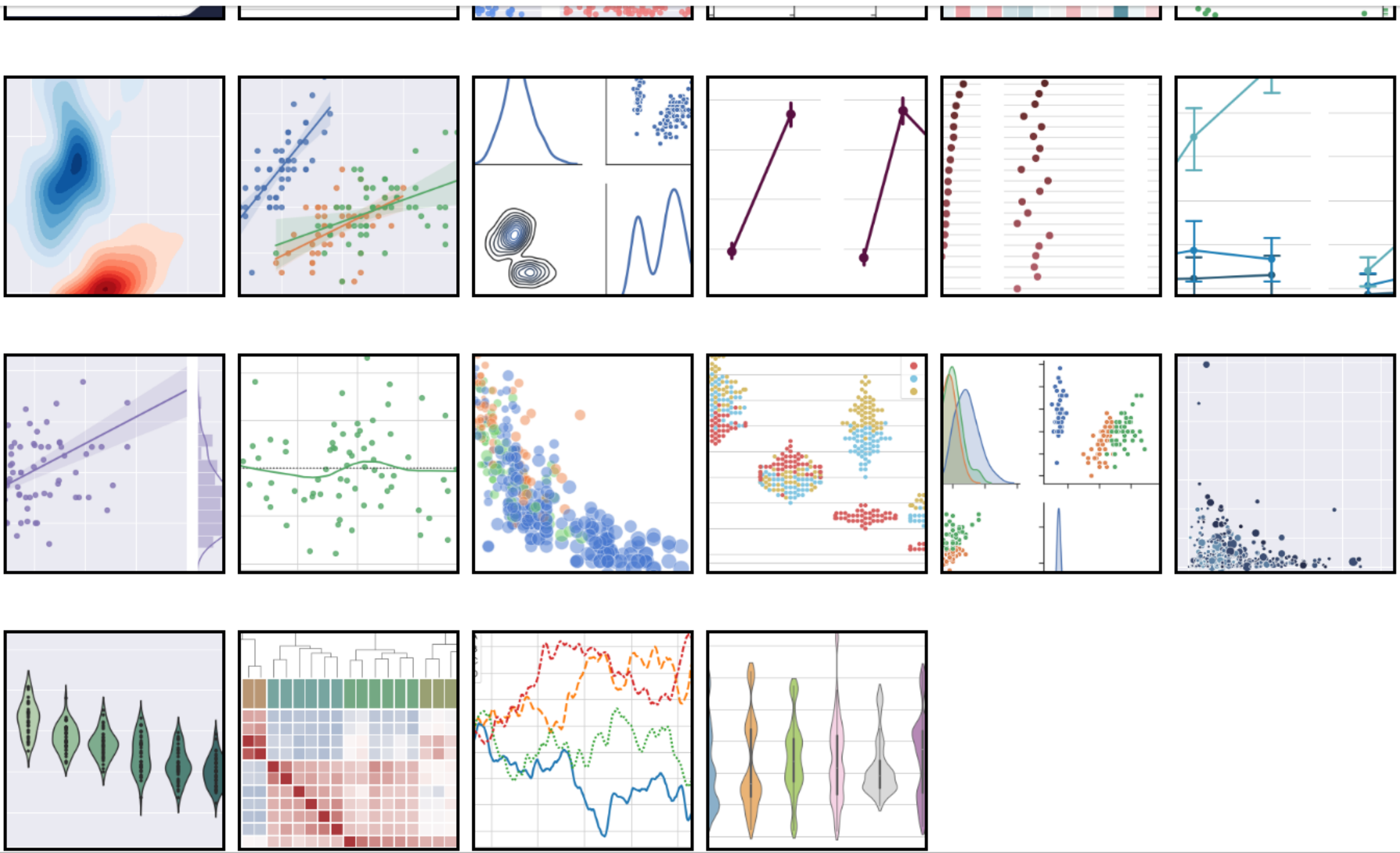
MATPLOTLIB

- Matplotlib is the base library for charting;
- MATLAB-like interface;
- Base for a large number of third party packages, like seaborn, holoviews, ggplot, …

DEMO:
MATPLOTLIB
(VISUALIZATIONS.ipynb)

seaborn 0.9.0 Gallery Tutorial API Site Page Search

- http://seaborn.pydata.org/examples/index.html



Bokeh

Bokeh

1.0.2 ▾ INSTALLATION USER GUIDE GALLERY TUTORIAL REFERENCE RELEASES DEVELOPER GUIDE

Welcome to Bokeh ¶

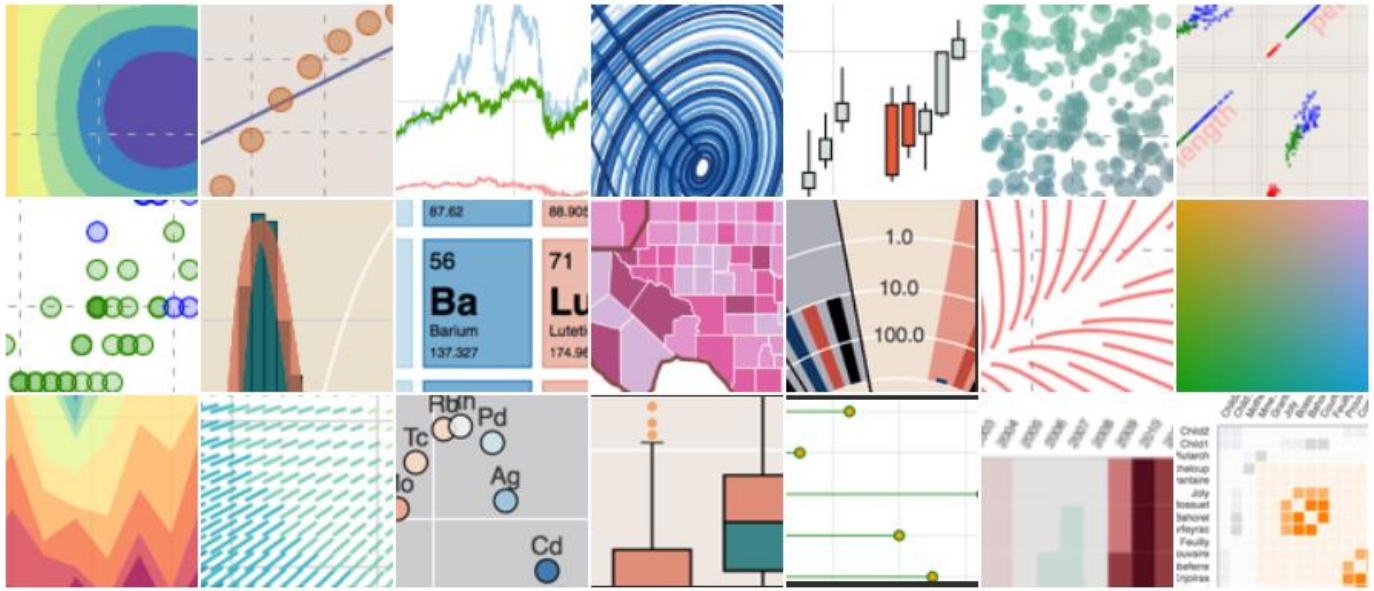
Bokeh is an interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise co graphics, and to extend this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone and easily create interactive plots, dashboards, and data applications.

To get started using Bokeh to make your visualizations, start with the User Guide.

For examples of how you might use Bokeh with your own data, check out the Gallery.

For detailed information about specific Bokeh components, consult the Reference Guide.

If you are interested in contributing to Bokeh, or extending the library, see the Developer Guide.



Plotly


(−0.4845348, 0.3549026)
plotly
Cobalt
Atomic Mass: 58.933195
Co
As
Rh Pd Ag Cd In Sn Sb
Ir Pt Au Hg Tl Pb Bi
Mt Ds Rg Cn Uut Fl Uup
Eu Gd Tb Dy Ho Er Tm
Am Cm Bk Cf Es Fm Md

DEMO: PLOTLY

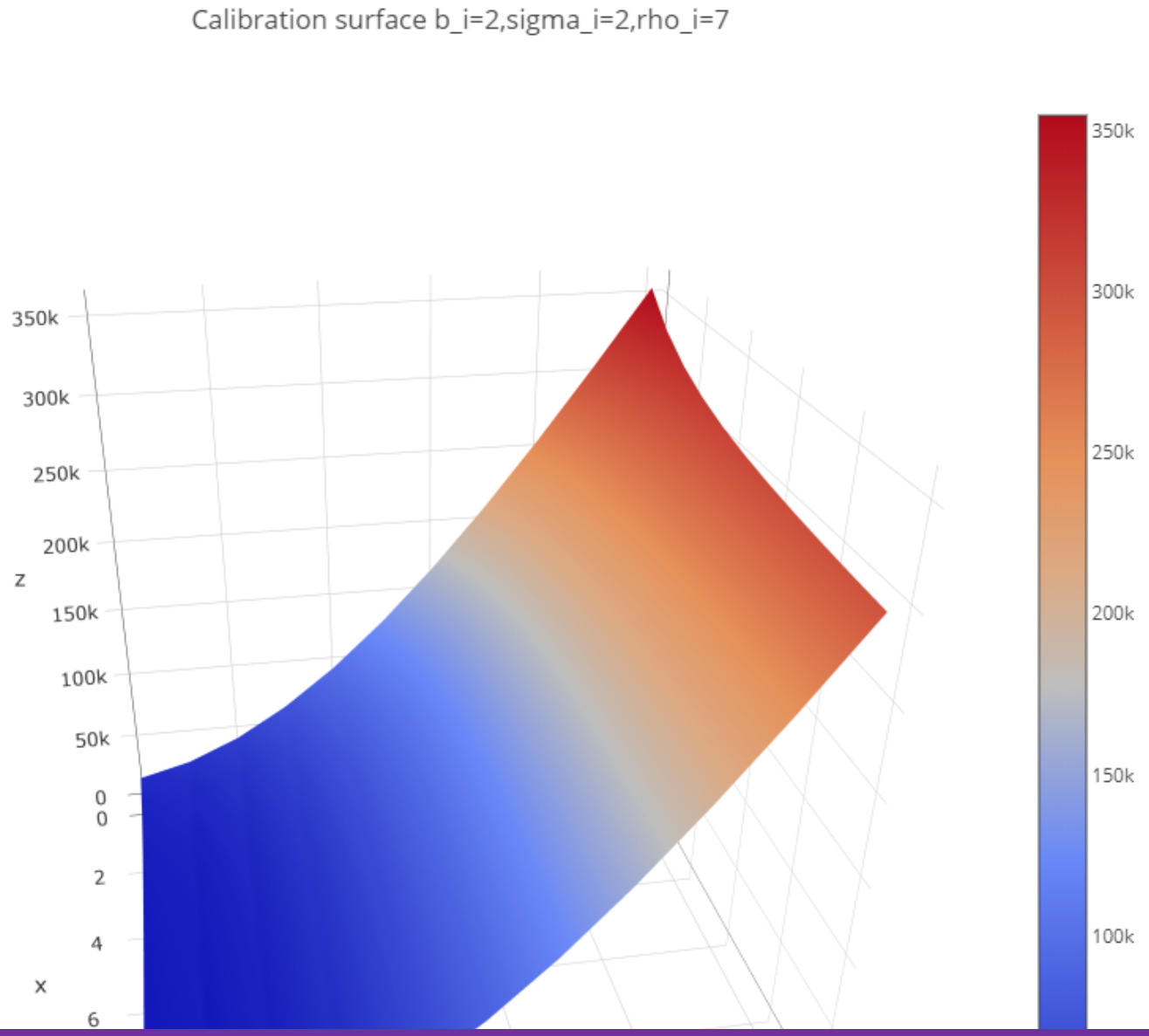
(VISUALIZATION-CALIBRATION.ipynb)

```
    )
    fig = go.Figure(data=data, layout=layout)
    return py.iplot(fig, filename='elevations-3d-surface')
```

In [9]:

```
draw3D(df, ["a_i", "nu_i", "Perf"], {"b_i": 2, "sigma_i":2, "rho_i" : 7})
```

Out[9]:



Visualization Examples

- https://python-graph-gallery.com/

PYTHON FINANCE

Financial packages in Python

- PyQL (QuantLib)
  - http://gouthamanbalaraman.com/blog/quantlib-python-tutorials-with-examples.html
  - https://www.youtube.com/playlist?list=PLu_PrO8j6XAvOAlZND9WUPwTHY_GYhJVr
- List of projects
  - https://github.com/wilsonfreitas/awesome-quant#python

PYTHON WITH .NET

Python with .Net

- **IronPython** https://ironpython.net/
  not a complete implementation, cannot use numpy and other optimized libraries;
- **PythonNet** https://pythonnet.github.io/
  truly seamless integration, convenient to use

- Pythonnet works surprisingly well;
- It is a real CPython with all the libraries;
- The only disadvantage is that we cannot use static methods and classes;
- Tested in PBB project for validation of calibration;

DEMO: REPORT GENERATOR

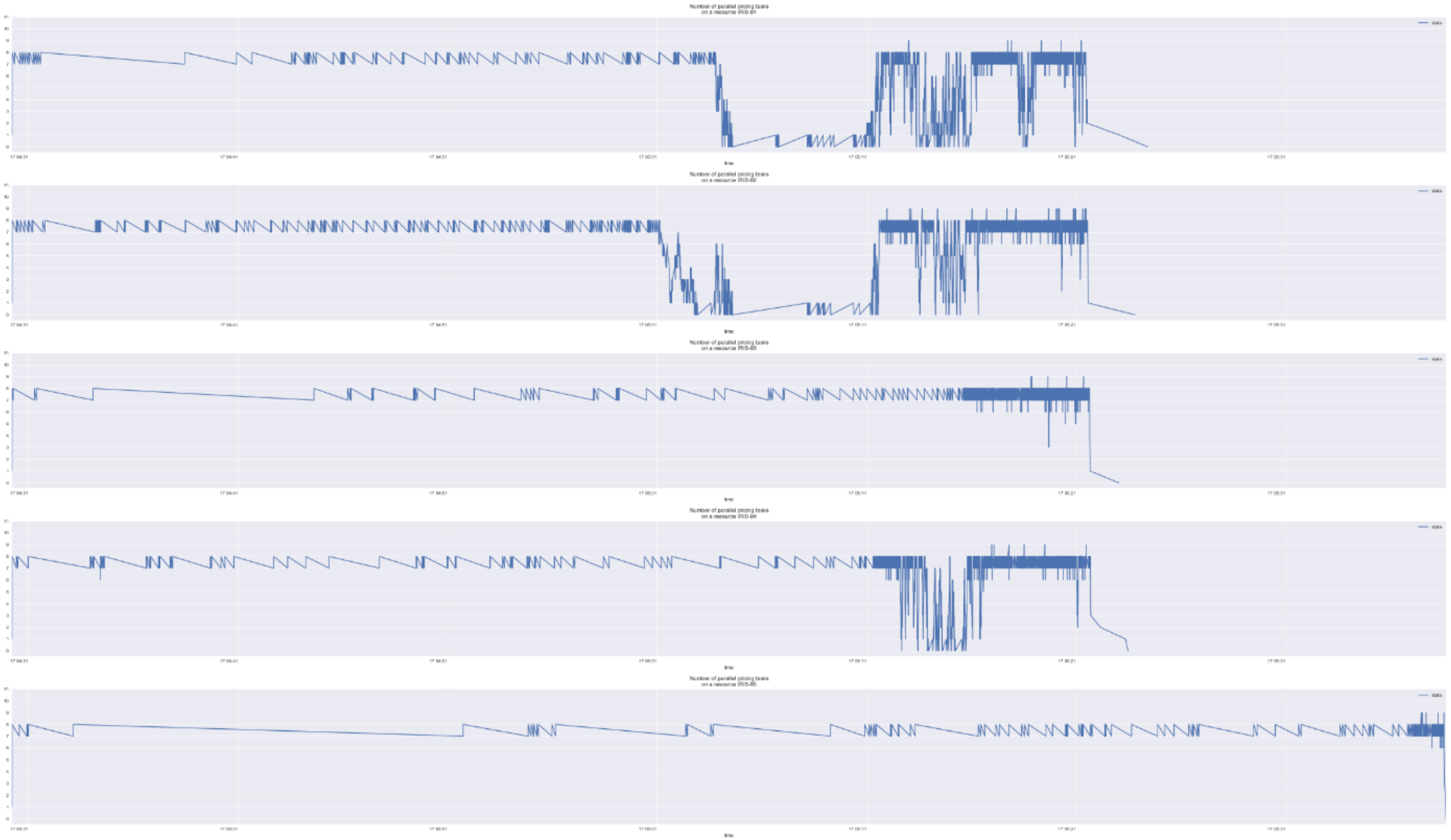
Workflow Stats Report Generator

- Render PDF describing;
- Load data from tracing;
- Render different visuals (tables, figures, charts);
- Technological stack
  - **Pandas** for data processing
  - **Flask** web server — http://flask.pocoo.org/
  - **Jinja2** templating engine — http://jinja.pocoo.org/docs/2.10/
  - **D3.js** –based Gantt chart — https://d3js.org/

Workflow Monitor

- Reuse all reporting code to present workflow stats on a web-site;
- A web UI for the task of listing and showing the reports for workflow runs. Now there are only two data analysis views available - soon I will add pricers stats.
- Technological stack
  - **Pandas** for data processing
  - **Flask** web server — http://flask.pocoo.org/
  - **Jinja2** templating engine — http://jinja.pocoo.org/docs/2.10/
  - **D3.js** –based Gantt chart — https://d3js.org/

Number of parallel tasks



IRIS — FloMon Server : DEV-GMS / Workflow : 20181217-051559-1570cd44-7082-4af6-bb55-833422b8aa0a / Report : Parallel Tasks

Q&A