Chair of Innovation Economics

Technische Universität Berlin

Introduction to Python for data scientists

workshop, 20.08.2019

Slides and files

https://drive.google.com/drive/folders/1xCaSxCaxO4hig8uh47prJpGpgCgn1eT-?usp=sharing

Goals

- Have a working Python environment set up
- Be able to run Python code
- Know basic syntax / know where to look for help
- Be able to install and use new packages
- Have Jupyter set up
- Load, save and manipulate tabular data in Python
- Calculate using data, plot results

Agenda

10 min **Introduction**
80 min **Basic Python syntax with exercises**

10 min **Break**

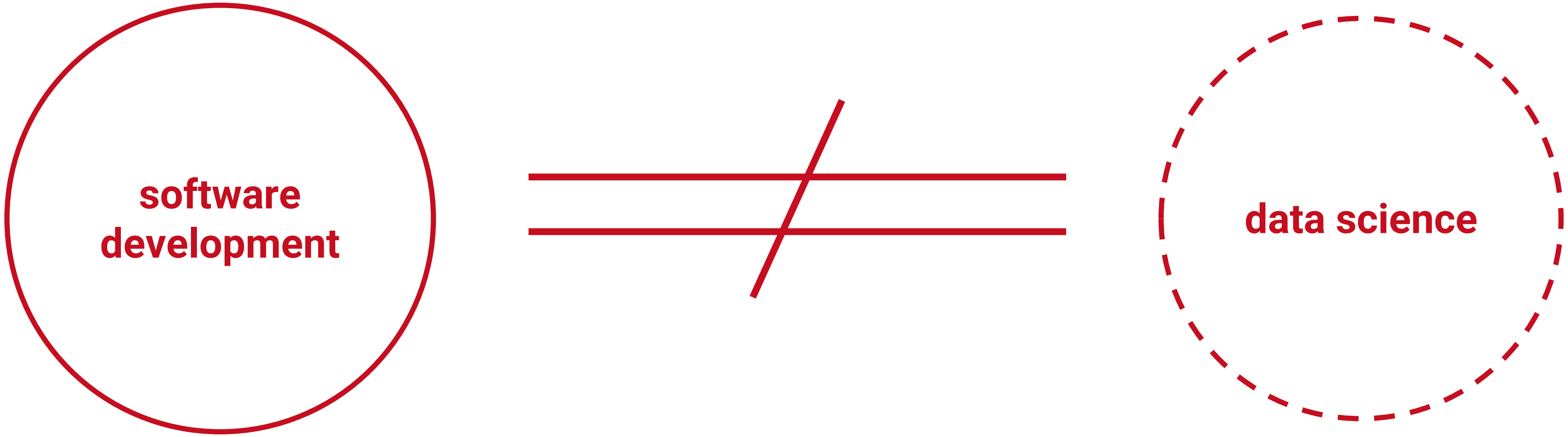
10 min **Modules & packages**
20 min **Jupyter**
60 min **Introduction to data analysis with Pandas**

10 min **Break**

Remaining time: **Final exercise, questions**

Introduction

Data science?



mobile apps, websites, business software, robots, autonomous cars, crypto currencies, satellites, nuclear plants
teams of professional developers
thousands of lines of code
project management, release dates, leanagilescrum
requirements
software design, architecture, patterns, styles
clients, users
result: tested, working, re-usable, safe code

data analysis, statistics, simulations, physical, mathematical computations
scientist(s), co-authors
as few lines of code as possible
project management !?
problems, ideas, data
exploration
scientific community
result: (reproducible) findings

Data science

- easy to learn
- quick results
- reliable results
- easy to share
- reproducible

data science

data analysis, statistics, simulations, physical, mathematical computations
scientist(s), co-authors
as few lines of code as possible
project management !?
problems, ideas, data
exploration
scientific community
result: (reproducible) findings

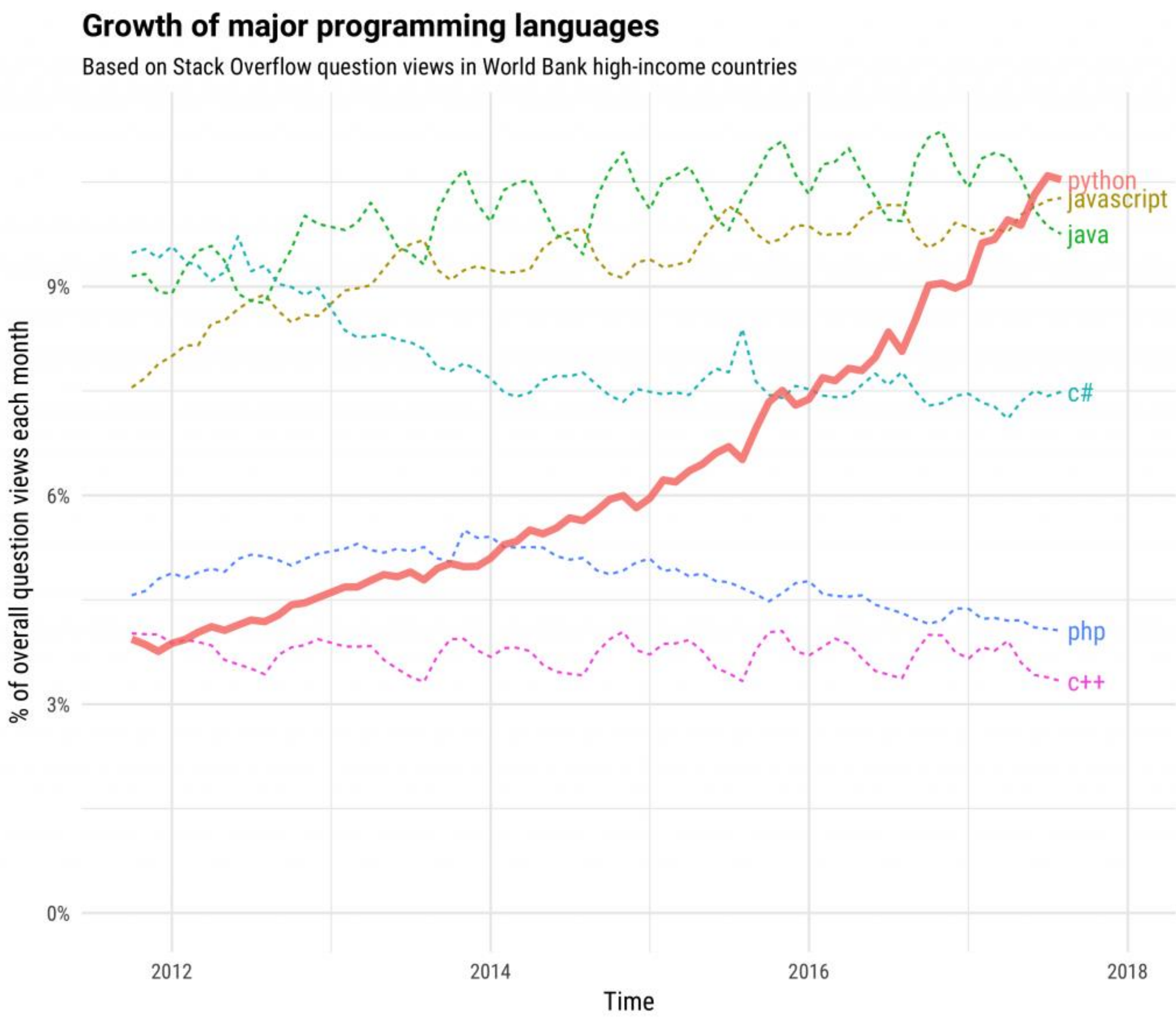
What is Python?

- “Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.”
- Invented in the 90s by Guido van Rossum (NL)
- Free / Open Source
- Has become de-facto standard for data science (along with other languages and tools…)

Python is simple

- Easy to learn, quick to get results with
- Intuitive syntax, readable code
- Hard to break
- Runs on every major platform (windows, macOS, linux etc.)
- Huge offer of packages (“add-ons”) to choose from (most problems are already solved)
- Well-tried in professional software development
- Large active community, many tutorials etc.

Python is trendy



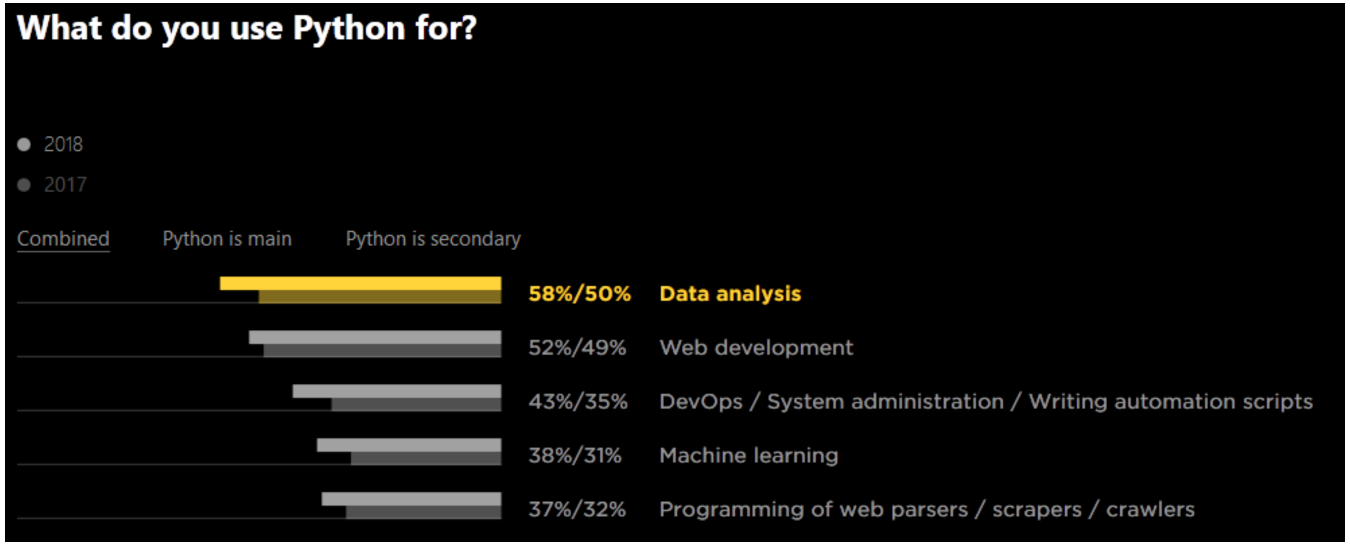
Everybody is using Python

Python: de-facto standard for data analysis

KDnuggets Software Poll (n>1,800)

Platform	2019 % share	2018 % share	% change
Python	65.8%	65.6%	0.2%
R Language	46.6%	48.5%	-4.0%
SQL Language	32.8%	39.6%	-17.2%
Java	12.4%	15.1%	-17.7%
Unix shell/awk	7.9%	9.2%	-13.4%
C/C++	7.1%	6.8%	3.7%
Javascript	6.8%	na	na
Other programming and data languages	5.7%	6.9%	-17.1%
Scala	3.5%	5.9%	-41.0%
Julia	1.7%	0.7%	150.4%
Perl	1.3%	1.0%	25.2%

https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html



https://www.jetbrains.com/research/python-developers-survey-2018/

Running python

1. **Interactive mode:** Running code in the console. On Windows: ⊞ win + R → "cmd" → `ipython`
2. **Scripts:** Executing a python-script (filenames with .py extension) → console: `python filename.py`
3. **Notebooks:** Running code in a "notebook", e.g. Jupyter

Exercise (2 mins):

Start Python in interactive mode

Python syntax

Python syntax

- Variables: `x = 1 + 2`
- Basic blocks: `=, ==, >, <, >=, <=, !=, not, and, or, (, )`
- Each line is a command
- *`# comments`* will not be executed
- `x = 1 + 2 # anything after “#” is a comment`
- Python3 supports Unicode: `𠕎 = 1 + 2`
- use `print(…)` to output expressions

Python syntax: Basic types & operations

- `None` type
- Booleans: `True`, `False`
- Numbers: `int` (`-1`, `0`, `1`, …) `float` (`-1.1283`, …) , complex (…)
- Text: `str` (`“this is a text”`, `‘this is a text’`)
- Sequences / sets:

```
list: [0, 1, 2], [‘a’, ‘b’, 1], [True, False], []
tuple: (0, 1, 2), (‘a’, ‘b’, 1)
set: {0, 1, 2}, {True, ‘a’, 1}, {‘a’, 1, ‘a’}
```

- Maps:

```
map { ‘a’: 0.284, ‘b’: True, 1: “xy” }
```

Python syntax: Booleans

- `True, False # case sensitive!`
- `x = True`
- `y = False`
- `x or y # True`
- `x and y # False`
- `not x # False`
- `not y # True`
- `not ( x and y ) # True`

https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not

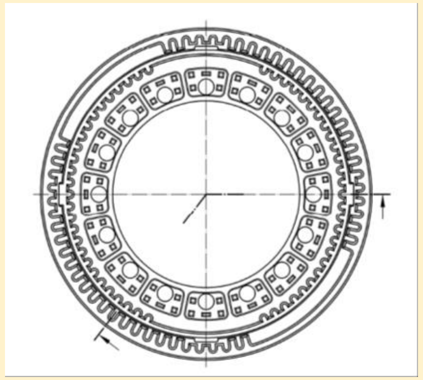
Python syntax: Numbers

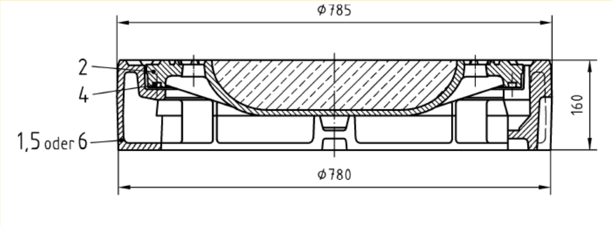
Operation	**Result**
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x % y`	remainder of x / y
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`float(x)`	*x* converted to floating point
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Python syntax: Numbers

Exercise (5 mins):





DIN 19584-1:2012-10

What is the area (in m²) of this DIN 19584 – A – D 400 manhole cover?

Hints: $\pi \approx 3.14$, $r = 0.5 * 785$ mm

Python syntax: strings

- x = “FG INNO”

- len(x) *# 7*
- “I” in x *# True*
- x.count(“N”) *# 2*
- x[1] *# G*
- x = x + “ is great” *# x = “FG INNO is great”*
- y = “!!!”
- x + y *# “FG INNO is great!!!”*
- “You can insert new lines \n and tabs \t etc.”

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Python syntax: lists

- `x = [1, 5, 7, 3, 5, 9]`

- `len(x) # 6`
- `sum(x) # 30`
- `max(x) # 9`
- `3 in x # True`
- `x.count(5) # 2`
- `x.sort() # x = [1, 3, 5, 5, 7, 9]`
- `x + [10, 11] # [1, 3, 5, 5, 7, 9, 10, 11]`

Python syntax: tuples & sets

Quiz

What is

(a) `len( set( [1,1,2,2,3,3] ) )`
(b) `(2,1,3).sort()` ?

Python syntax: list slicing

- `list[start:stop:step]`

- `x = [“DIN”, “ISO”, “DIN”, “ETSI”, “IETF”]`

- `x[1:3] # [“ISO”, “DIN”, “ETSI”]`
- `x[3:] # [“ETSI”, “IETF”]`
- `x[:1] # [“DIN”, “ISO”]`
- `x[:-3] # [“DIN”, “ISO”]`
- `x[1:4:2] # [“ISO”,”ETSI”]`

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Python syntax: list comprehension

- Create a list by giving a functional definition of the elements
- `x = [i for i in range(1,5)] # [1,2,3,4]`
- `y = [a/2 for a in range(0,3)] # [0, 0.5, 1]`
- `z = [(n, 2*n) for n in y] # [(0,0), (0.5, 1), (1,2)]`
- `z_1 = [(n, 2*n) for n in y if n<1] # [(0,0), (0.5, 1)]`
- `z_2 = [(n, 2*n) for n in y][::2] # [(0,0), (1, 2)]`

Python syntax: lists

Exercise (10 mins):

What is the sum of every second power of two:
2^n for $n = 0..10$?

Python syntax: lists

Solutions:

What is the sum of every second power of two:
2^n for $n = 0..10$? `1365`

```
2**0 + 2**1 + 2**2 + …
sum([2**n for n in [0,2,4,6,8,10]])
sum([2**n for n in range(0,11,2)])
sum([2**n for n in range(0,11) if n % 2 == 0])
sum([2**n for n in range(0,11)][::2])
```

Python syntax: maps

- `map = { key1: value1, key2: value2} # keys are unique`
- `map[key] # value`

- `levels = { “DIN”: “DE”, “DKE”: “DE”, “CEN”: “EU”, “CENELEC”: “EU”}`
- `level[“DIN”] # DE`
- `level[“ETSI”] = “EU”`

- `level.keys() # DIN, DKE, CEN, CENELEC`
- `Level.values() # DE, DE, EU, EU`

- `x = {i: i**2 for i in range(2,6)} # {2:4, 3:9, 4:16, 5:25}`

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Python syntax: control flows

- The usual suspects: `if`, `else`, `while`, `for`
- Inside these constructs, statements are intended (e.g. with a tab):

```
if x > 5:
    print("x is too big")
    x = 5
else:
    print("x is ok")
```

Python syntax: if, else, elif

```
if x > 5:
    print("x is too big")
elif x < 2:
    print("x is too small")
else:
    print("x is ok")

s = "negative" if x < 0 else "positive"
```

Python syntax: while

- Execute code as long as an expression is `True`

```
x = 0
while x < 10:
    print(x)
    x = x + 1

# 0 1 2 3 4 5 6 7 8 9
```

Python syntax: for

- Same as in list comprehensions, used to iterate over a sequence of items

```
for x in range(10):
     print(x)

for x in [i for i in range(10) if i % 2 == 0]:
     print(x)

for w in “an example sentence”.split(“ “):
     print(w)
```

Python syntax: control flow

Exercise (15 mins):

How many prime numbers are < 100?

Python syntax: control flow

Exercise (15 mins): How many prime numbers are < 100?

```
count = 0
# look at all numbers n from 2 - 100:
for n in range(2,101):
        is_prime = True
        # if n can be divided by any number x<n, it’s not a prime number
        for x in range(2, n):
                if n % x == 0:
                        is_prime = False
                        break
        if is_prime:
                count += 1
print(count)
```

Python syntax: control flow

```
primes = []
# look at all numbers n from 2 – 100:
for n in range(2,101):
      is_prime = True
      # if n can be divided by any number 2>n>x, it’s not a prime number
      for x in range(2, n):
            if n % x == 0:
                  is_prime = False
                  break
      if is_prime:
            primes.add(n)
print(len(primes))
print(primes)
```

Python syntax: functions

- Re-usable code blocks with parameters

```
# a function that returns all prime numbers < limit
def primes(limit):
    p = [n for n in range(2, limit + 1)
           if not any([n % x == 0 for x in range(2, n))]

    return p

print(primes(1000))
print(373 in primes(400))
```

Python syntax: functions

```
def function_name(x, y, z):
    …
# parameters be optional and can have default values
def f(x, y, z=5):
    return x+y+z

f(1,2) # 8
f(1,2,3) # 6
f(y=2, x=1, z=3) # 6
```

Python syntax: (parameter) unpacking

```
a, b, c, d = [1, 2, 3, 4] # a=1, b=2, …

def f(x, y, z):
    return [x+y, z]

a, b = f(1,2,3) # a=3, b=3

f(*[1,2,3]) == f(1,2,3) # True, pass parameters as a list
f(*(1,2,3)) # … a tuple
f(**{'x': 1, 'z': 3, 'y': 2}) # … or named as a map
```

https://docs.python.org/3/tutorial/controlflow.html#unpacking-argument-lists

Python syntax: (parameter) unpacking

```python
import math

# example: transformation from cartesian to spherical coordinates
def spherical(x, y, z):
    r = math.sqrt(x**2 + y**2 + z**2)
    phi = math.arctan(y/x)
    theta = math.arccos(z/r)

    return (r, theta, phi)

# some_object.coordinates() = (x, y, z)
r, phi, theta = spherical(*some_object.coordinates())
```

https://docs.python.org/3/tutorial/controlflow.html#unpacking-argument-lists

Python syntax: functions

Exercise (10 mins):

Write a function that transforms spherical coordinates to cartesian coordinates: (r, theta, phi) → (x, y, z)

Hint:

$$x = r \sin\theta \cos\varphi$$
$$y = r \sin\theta \sin\varphi$$
$$z = r \cos\theta$$

Use `import math` *to load the math module, and the* `math.sin(x)` *and* `math.cos(x)` *functions.*

Python syntax: functions

```
import math

def cartesian(r, theta, phi):
      x = r * math.sin(theta) * math.cos(phi)
      y = r * math.sin(theta) * math.cos(phi)
      z = r * math.cos(theta)

      return (x, y, z)

coords = (1.5, 90, 180)
cartesian(*(coords))
```

https://docs.python.org/3/tutorial/controlflow.html - https://docs.python.org/3/library/math.html

Python syntax: functions

Quiz

How would you output just the resulting `y` of `cartesian(*(coords))` ?

Python syntax: lambda expressions

```
def plus(x, y):
     return x + y
plus(1,2) # 3

minus = lambda x,y: x - y
minus(3,1) # 2

map( lambda x: x**2 + (x/2), [1,2,3,4] )
# [1.5, 5.0, 10.5, 18.0]
```

https://docs.python.org/3/reference/expressions.html#lambda

10 min break

modules & packages

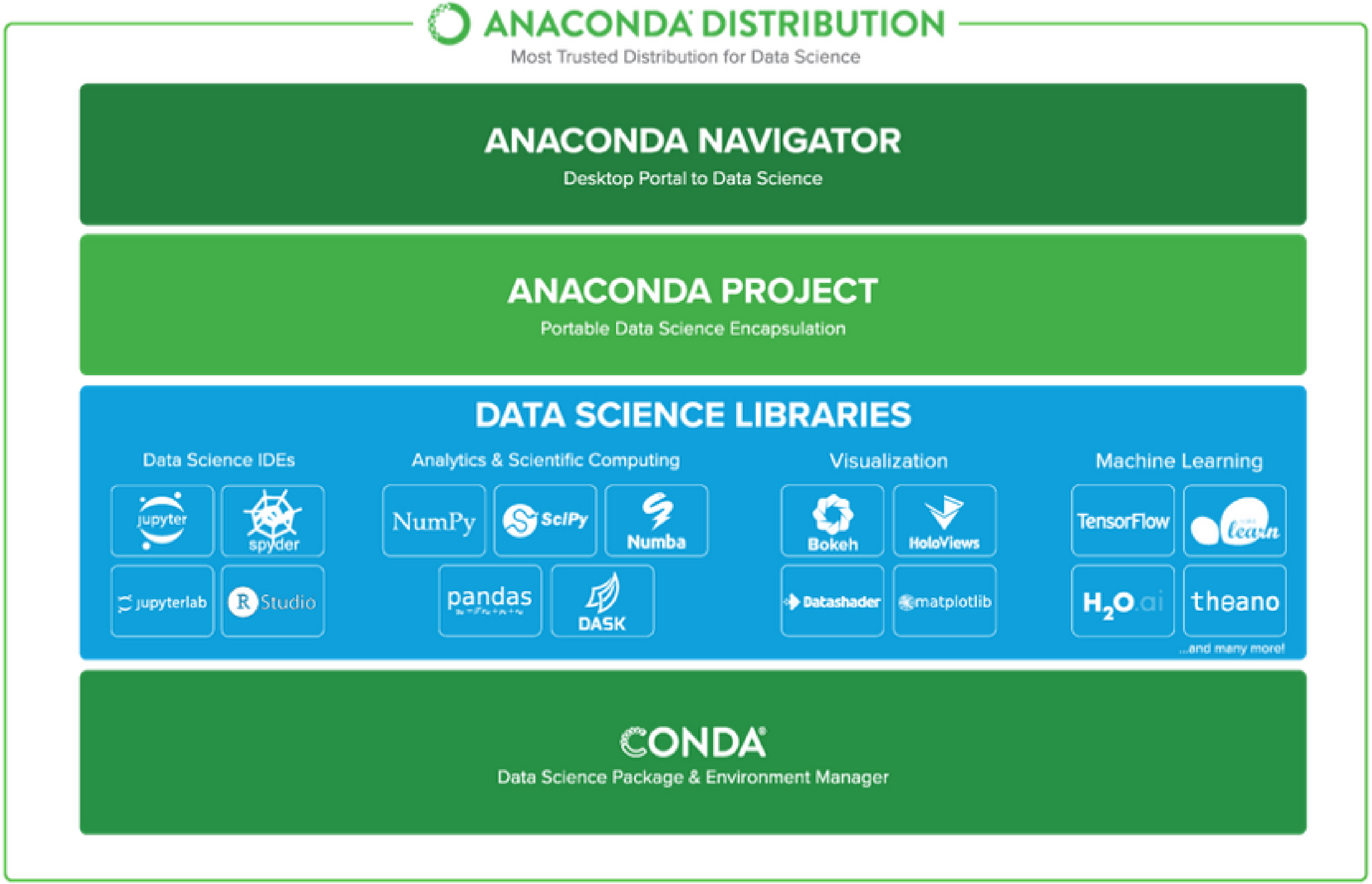
Python syntax: Modules

```python
import math
import math as m
from math import cos
from math import sin, pi
from math import pi as π

math.sin(0)
m.sin(0)
cos(0)
sin(0)
print(pi)
print(π)
```

Modules & packages

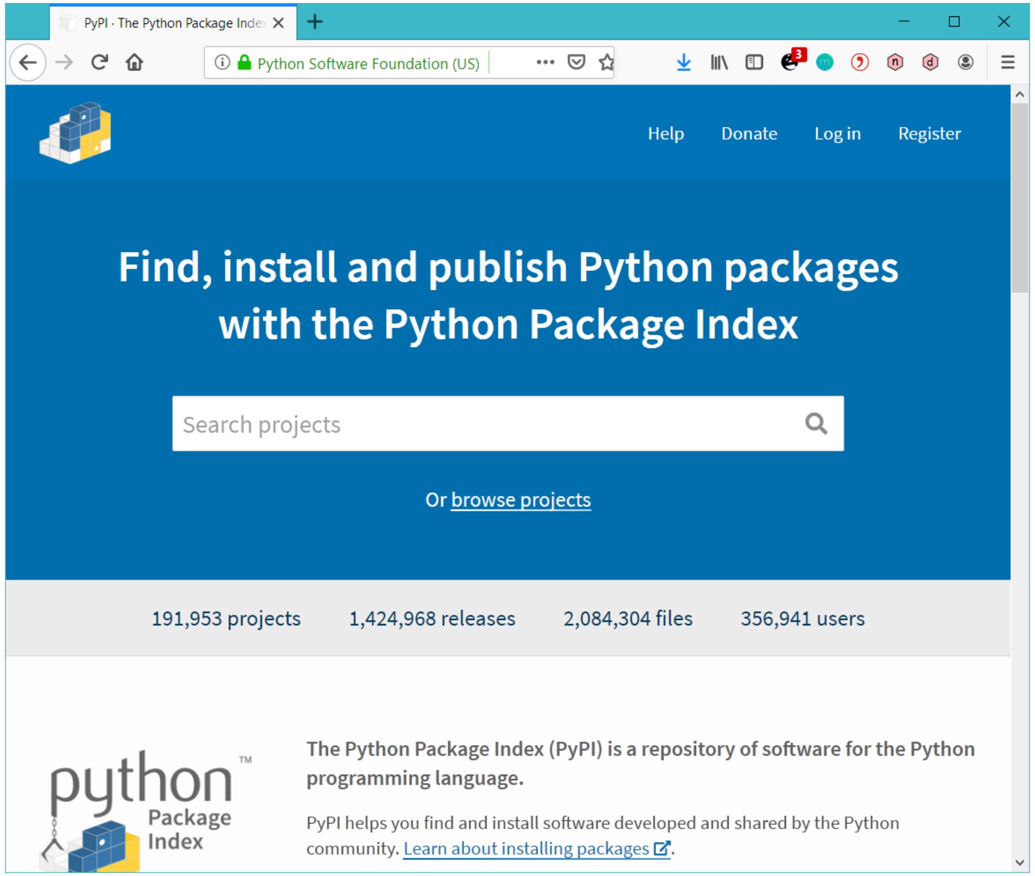
- Default **modules** that are always included in Python: https://docs.python.org/3/library/index.html. Examples:
  - math – mathematical functions
  - time — time access and conversions
  - random — generate pseudo-random numbers
- **Packages** (contain modules), open source, listed on https://pypi.org. Currently ~190,000 projects. Examples:
  - Numpy – (fast) scientific computing with Python
  - Pandas – data analysis library
  - TensorFlow – machine learning
  - Scikit-Learn – machine learning
- If you are using the Anaconda distribution, a lot of packages will already be installed!

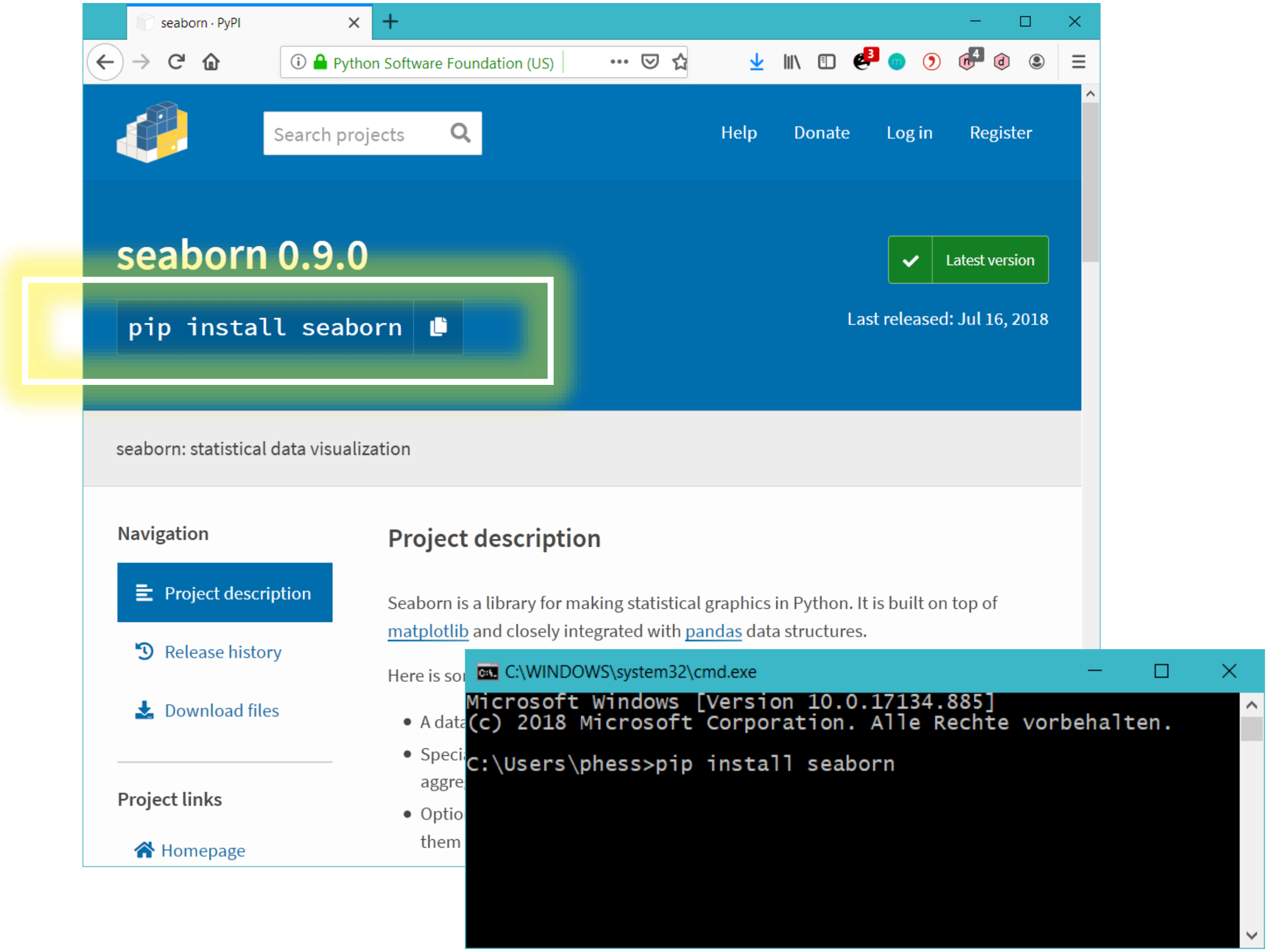
Modules & packages



https://docs.anaconda.com/anaconda/packages/py3.7_win-64/

Modules & packages





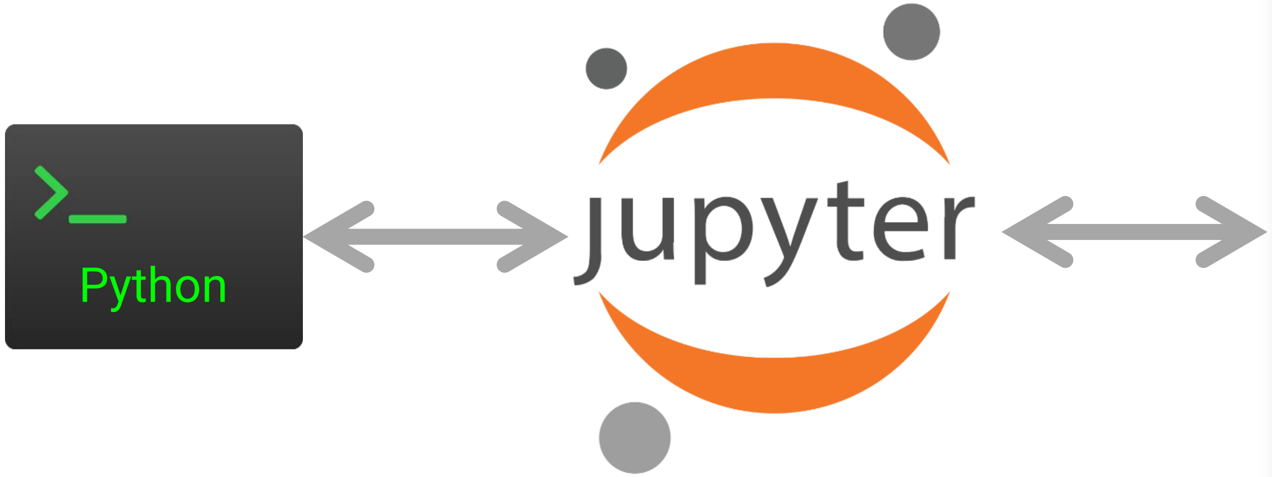
Modules & packages

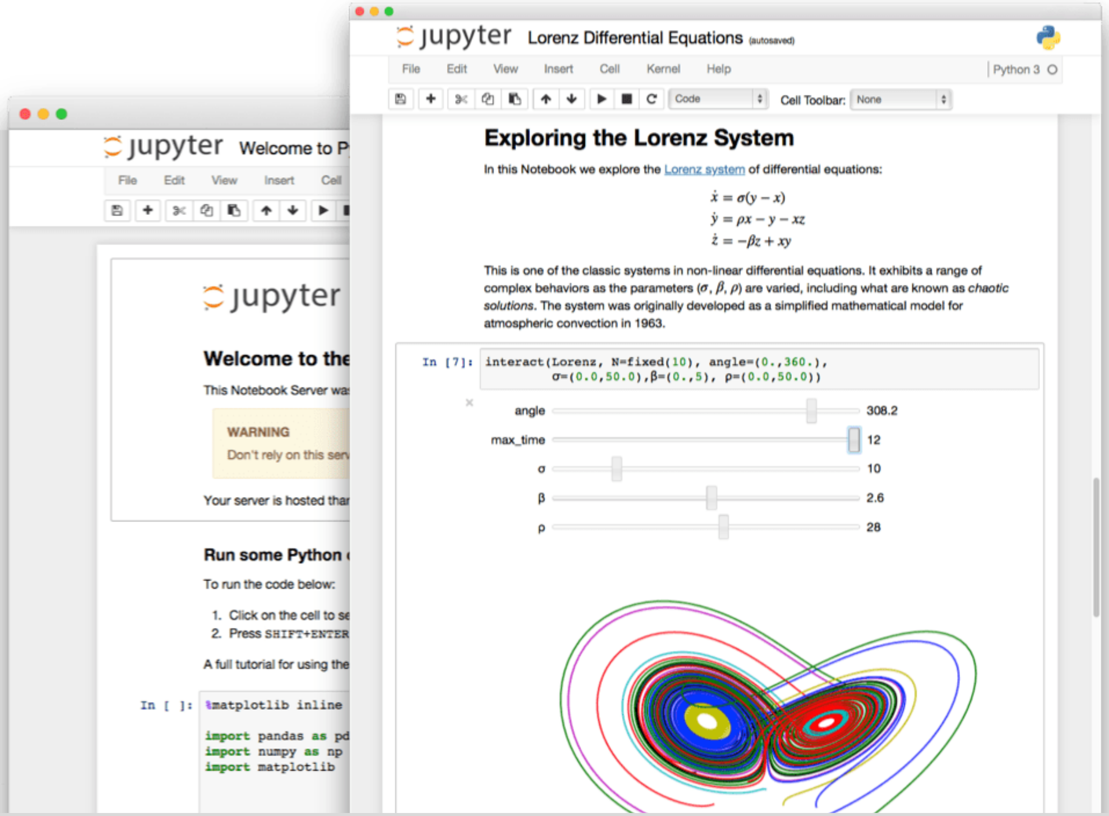
```
import seaborn as sns

# some magic so the plot will be shown when produced in the console
%matplotlib qt

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

# Plot
sns.lineplot(x="timepoint", y="signal",
             hue="region", style="event", data=d)
```

Jupyter

Jupyter

- Write Python code in a “notebook” in your browser
- Output is included in the notebook
- Easy to share notebooks





Jupyter notebooks

jupyter_examples.ipynb

Running python

1. **Interactive mode:** Running code in the console. On Windows: ⊞ win + R → "cmd" → `ipython`
2. **Scripts:** Executing a python-script (filenames with .py extension) → console: `python filename.py`
3. **Notebooks:** Running code in a "notebook", e.g. Jupyter

Exercise (2 mins):

Start Jupyter: ⊞ win → "Jupyter" → enter

Jupyter notebooks

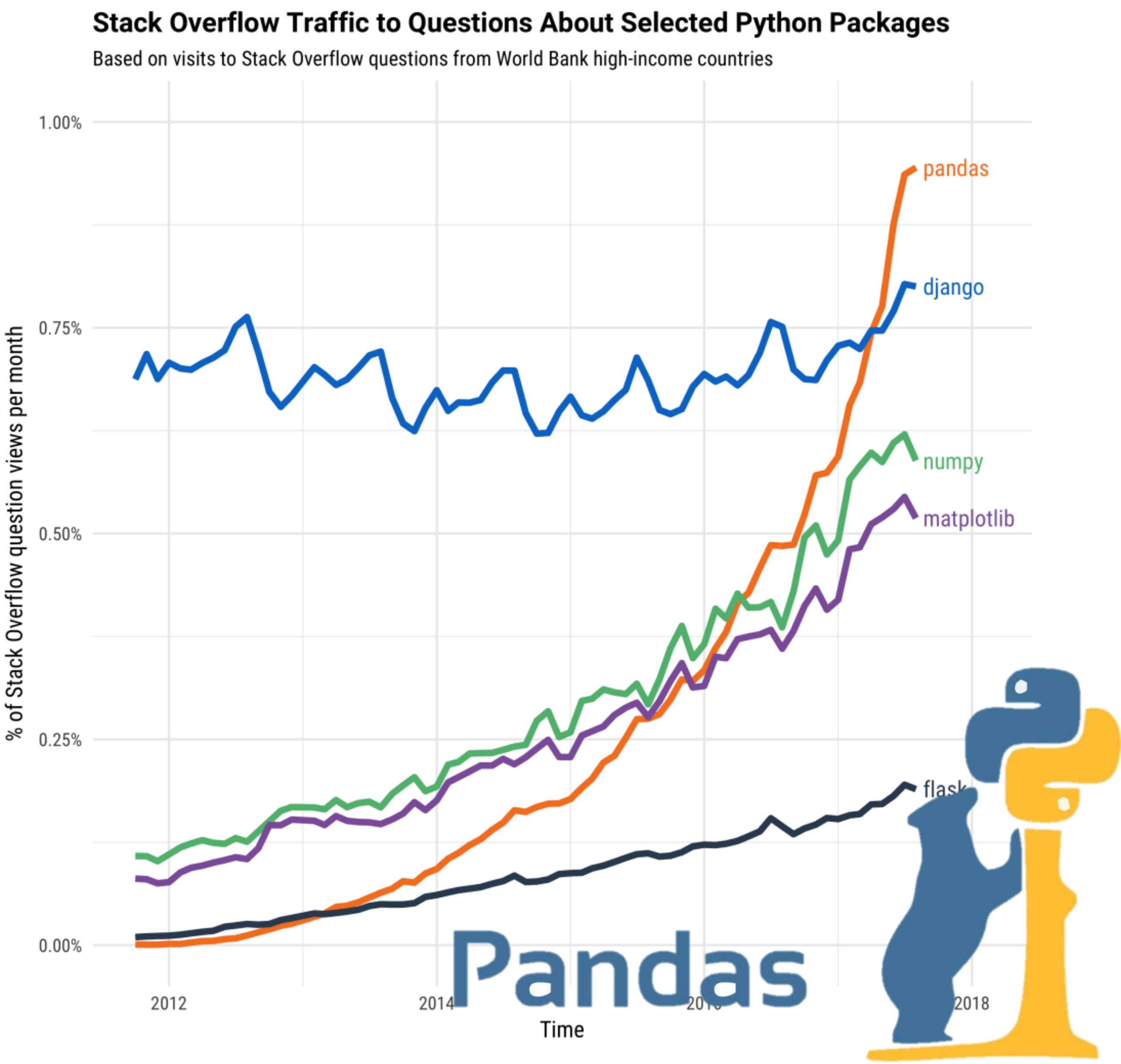
Exercise (20 mins):

- Create a new Jupyter notebook called „<your name>“ on your desktop.
- Copy and run one of the plotting examples from https://seaborn.pydata.org/examples/index.html
- Upload your notebook to the cloud at https://tubcloud.tu-berlin.de/s/yXken8P3toea5XR
- Look at and run some of the others' examples

Introduction to data analysis with Pandas

Pandas

- Part of the Anaconda distribution
- The “Excel”, “R data.tables” of Python
- Easy loading / saving / manipulation of tabular data
- Integrated easy plotting
- Querying data
- Support for time series
- …



Pandas: DataFrame, Series

- Core classes: `DataFrame` and Series
- `Series`: An ordered, indexed sequence of arbitrary values (numbers, strings, dates, ...)
- `DataFrame`: A two dimensional datatype with rows, columns and an index – a table. Or: a collection of Series (columns) with the same index

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Pandas

- **Creating and exploring a dataset**
- **Indexing:** Selecting a certain data item from a Series, or a data item or sequence of data items from a DataFrame
- **Aggregation:** Running functions on sequences of data items
- **Grouping**
- **Selection:** Selecting items/sequences using logic
- **Plotting**

https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html

Pandas

[pandas_intro.ipynb]

Pandas: Series

Exercise (10 mins):

Generate a random Series that is correlated (>= 0.5) to the prime numbers below 100

Pandas: Series

Exercise (10 mins): Generate a random Series that is correlated (>= 0.5) to the prime numbers below 100

```
corr = -1
r = None

while corr < 0.6:
    r = pd.Series(random.sample(range(1, 100), 25), index=primes.index)
    corr = primes.corr(r)

r.plot(title="correlation = " + str(corr))
primes.plot()
```

Pandas: indexing Series

- **Series** → Value (1d)

```
s1 = Series(range(5))
s1[2] # 2
```

index	
0	0
1	1
2	2
3	3
4	4

```
s2 = Series(
    range(5),
    index=['A', 'B', 'C', 'D', 'E']
)
s2['D'] # 3
```

index	
A	0
B	1
C	2
D	3
E	4

Pandas: indexing DataFrames

- **DataFrame** → Series (2d) / Value (1d)

```
df = DataFrame({
    'a': range(5),
    'b': range(2,7)
})

df['b'] # 2 3 4 5 6 (column)
df.iloc[0] # 0 2 (row)
df.iloc[1,1] # 3 (value)
df.iloc[::2] # every second row
```

index	a	b
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

index	a	b
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

index	a	b
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

index	a	b
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

Pandas: indexing DataFrames

- **names or numbers**

```
df1 = DataFrame([
    range(5),
    range(2,7)
])

df2 = DataFrame(
    { ‘a’: range(5),
      ‘b’: range(2,7),
      ‘C’: range(3,8) },
    index=[‘A’,’B’,’C’,’D’,’E’]
)
```

index	**0**	**1**
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

index	**a**	**b**	**c**
A	0	2	3
B	1	3	4
C	2	4	5
D	3	5	6
E	4	6	7

Pandas: indexing DataFrames

- **names or numbers**

```
df1[0] # first column
df1.iloc[0] # first row
df1.iloc[0,1] # row=0, col=1
df1.iloc[::2,1] # every second item of col=1
```

index	0	1
0	0	2
1	1	3
2	2	4
3	3	5
4	4	6

```
df2['a']
df2[['a','c']]
df2.loc['A'] == df2.iloc[0] # True
df2.loc[['C','E'],['a','b']] # 2 4
                             # 4 6
```

index	a	b	c
A	0	2	3
B	1	3	4
C	2	4	5
D	3	5	6
E	4	6	7

Pandas: aggregating DataFrames

Series → Value (1d): `Series(range(5)).sum() # 10`

DataFrame → Series (2d) / Value (1d)

```
df2[‘a’].sum() # 10
df2.sum() # 10 20 25
df2[‘A’].sum() # 5
df2.sum(axis=1) # 5 8 11 14 17
df2.sum().sum() # 55
```

axis=1

axis=0
default

index	**a**	**b**	**c**	*SUM*
A	0	2	3	**5**
B	1	3	4	**8**
C	2	4	5	**11**
D	3	5	6	**14**
E	4	6	7	**17**
SUM	**10**	**20**	**25**	**55**

Other common aggregations: `mean, median, mode, max, min, var, std`
https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats

Pandas: aggregating DataFrames

- **Compute any aggregation:**

- `df2.aggregate(numpy.sum) # == df2.sum()`
- `df2.aggregate(numpy.sum, axis=1) # == df2.sum(axis=1)`
- `df2.aggregate(numpy.log) # 5x3 df with log(item)`
- `df2.aggregate([numpy.sum, numpy.mean, numpy.std])`
- `df2.aggregate(lambda x: x**2 + x)`

Pandas: aggregating DataFrames

- **Custom aggregations on axes**

```
# aggregate each column with A - E
df2.apply(lambda rows: (rows[‘A’] – rows[‘E’]))

# aggregate each row with a / c
df2.apply(lambda cols: (cols[‘a’]/cols’[c’]), axis1)
```

Pandas: DataFrame transposition

- df2.T

index	a	b	c
A	0	2	3
B	1	3	4
C	2	4	5
D	3	5	6
E	4	6	7

index	A	B	C	D	E
a	0	1	2	3	4
b	2	3	4	5	6
c	3	4	5	6	7

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.transpose.html

Pandas: DataFrame indexing & aggregation

Exercise (10 mins, choose one):

- Get all 2019 innovation expenditures of sectors with numbers starting with “C“
- Get the innovation expenditures of “C 26” from 2010 to 2019
- Imagine that data was corrupted in every 2nd year. Use only the 1st, 3rd, … year to calculate mean expenditures.
- Calculate the average change in expenditures since 2006 for sectors “C ..”

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Pandas: DataFrame indexing & aggregation

- `inno_num.loc[`
  `"2019", [c for c in inno_num.columns if c[0]=="C"]`
  `]`
- `inno_num.loc[`
  `[str(y) for y in range(2010,2020)], “C 26”`
  `]`
- `inno_num.loc[::2, :].mean()`
- `inno_num[[c for c in inno_num.columns if c[0]=="C"]].apply(lambda years: (years[-1] - years[0])).mean()`

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Pandas: grouping

dfg =

index	gender	age	height
0	M	60	1.80
1	F	30	1.60
2	F	10	1.50
3	M	7	1.45
4	M	3	0.73

```
dfg.groupby(by=“gender)
```

index	gender	age	height
0	M	60	1.80
3	M	7	1.45
4	M	3	0.73

index	gender	age	height
1	F	30	1.60
2	F	10	1.50

```
dfg.groupby(by=“gender”).mean()
```

	age	height
gender		
F	20.000000	1.550000
M	23.333333	1.326667

```
dfg.groupby(by="gender").aggregate([np.mean, np.std])
```

	age		height	
	mean	std	mean	std
gender				
F	20.000000	14.142136	1.550000	0.070711
M	23.333333	31.817186	1.326667	0.545558

Pandas: grouping

Average expenditures per sector groups

Use index for grouping:

	2006	2007	2008	200
Nummer				
B 05-09	0.3	0.5	0.4	
C 10-12	2.5	2.7	2.5	
C 13-15	0.6	0.8	0.8	

`inno_num.T`

```
(inno_num.T
 .groupby(by=lambda index: index[0])
 .apply(np.mean))
```

	2006	2007	2008	200
B	0.300000	0.500000	0.400000	0.20
C	4.847059	4.988235	5.417647	4.75
D	1.900000	2.500000	2.300000	2.50

10 min break

Introduction to Python for data scientists

Final exercise: