Scientific Python Course
-
TU Braunschweig

Felix Köhler

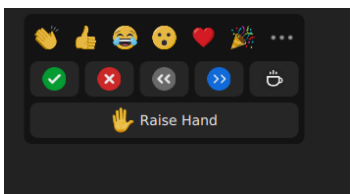
October 2021

How do you feel on Monday 9am?

1. Yes
2. Perfect Mood for learning to code
3. Tired, but excited
4. No

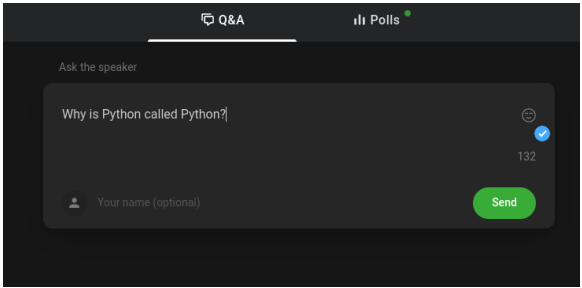
How to participate in the workshop?

- Use Slido to participate in interactive surveys
- Use Slido to ask questions, upvote or downvote questions of others
- The Zoom chat is your forum, ask for quick technical help, discuss etc.
  - The chat is monitored by my colleagues, but I will not actively look at it
- Use Zoom emotions to show if you agree, disagree, enjoy etc. (also use it to indicate whether I am too fast or too slow)



- Speak up and let us have a nice discussion (you will be cut out off the recording)

Now try to ask a silly question on Slido!



What is your favorite programming language?

1. MATLAB
2. C
3. C++
4. Fortran
5. Julia
6. other

How is your level of proficiency in Python?

1. Absolutely Nothing: What is Python?
2. Close to Nothing: I am here for the jump-start
3. Elementary: I did some simple tutorials
4. Intermediary: I used it for some stuff, but I mostly rely on Stackoverflow
5. Advanced: I used it for something more advanced
6. Expert: I answer nooby questions on Stackoverflow

The layout of the course

- 10 to 15 Minute breaks every 1 to 1.5 hours
- Lunch break from 13 : 00 to 14 : 00
- Course days end roughly at 17 : 00

Was the Python and VS Code installation successfull?

1. Yes, everything is fine - I also played around with it a bit
2. Yes, everything is fine - But I haven't done anything since
3. No, Python is not working
4. No, VS-Code is not working
5. No, None of the two is working

Overview I

Basics

- Why Python?
- Interacting with Python
- Elementary Syntax & Datastructures
- Module System

NumPy

- Arrays
- Vectorization
- Broadcasting & Reshaping
- Linear Algebra
- Outlook

Matplotlib

- Recap: Plotting on a computer
- Basic Plots

Overview II

Outlook

Pandas, Seaborn & Plotly

Pandas for dataintensive tasks
Seaborn & Plotly for visualizing dataframes easily
Other Plotly concepts

SciPy: Optimize & Statistics

The SciPy Library
Numerical Optimization in SciPy
Statistics and Distributions in SciPy
More on SciPy

Scikit-Learn

Machine Learning & Scikit API
Linear Regression
Logistic Regression

Overview III

Clustering
More with Scikit-Learn

TensorFlow Keras

A brief history of Deep Learning
The Basics of Deep Learning
A Neural Network Learning the Sine Function
Multilayer-Perceptron for hand-written digit recognition
Using a GPU on Google Colab
There is so much more

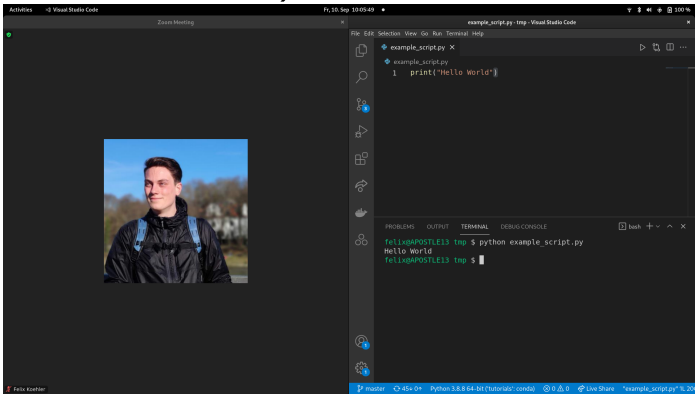
Outlook

Which topics are you most interested in?

1. NumPy & Matplotlib: I want to use Python like MATLAB or Octave
2. Pandas, Seaborn & Plotly: I want to do Data Analytics
3. SciPy: I like a library of well-maintained numerical algorithms
4. Scikit-Learn: I want to do some classical Machine Learning
5. TensorFlow & Keras: I like the hype

How to follow along the workshop?

Code along ;)

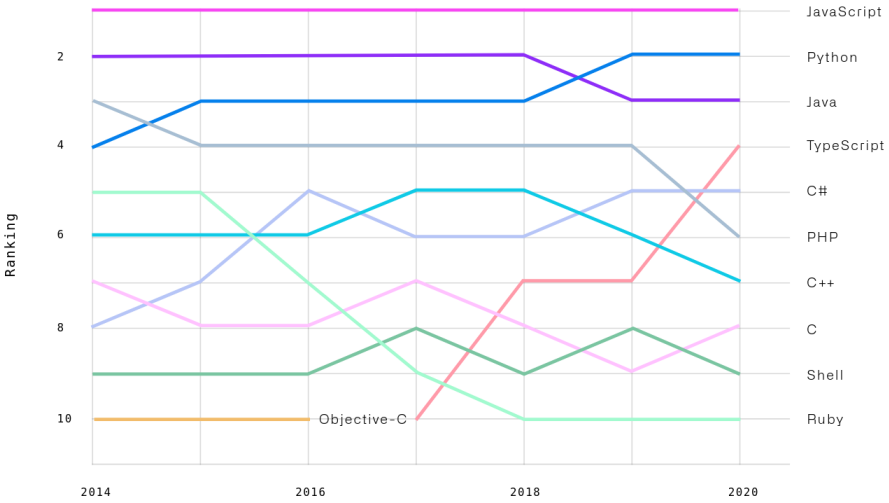


My general Tips for Learning Python

1. Use a top-down approach: You do not have to be an expert to write usable Python Code
   1.1 Make it
   1.2 Make it work
   1.3 Make it correct
   1.4 Make it fast
2. Use the internet and use different sources of exposure to Code
   2.1 Stack-Overflow is your friend
   2.2 Read from other learners on `medium.com`, `towardsdatascience.com` etc.
   2.3 Document your learning journey, write a blog post, chat with your peers etc.
   2.4 Try to read code of other Repos on GitHub
3. Use the Python Ecosystem of amazing packages
4. Learn by needing it (Do not learn everything all at once, rather have a broad overview and then learn that one Pandas function when you need it.)

Basics

Python is ubiquitous



1

[1] https://octoverse.github.com/

What is Python used for?

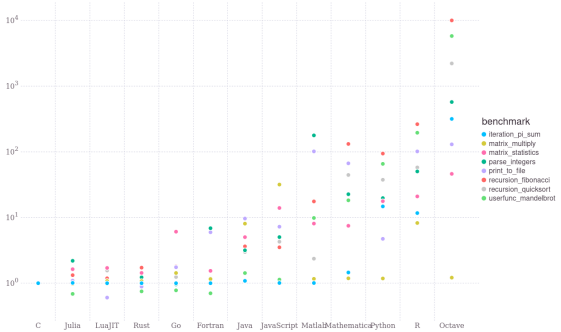
- Web Servers
- Scripting
- Machine Learning
- Scientific Computing
- . . . and many more

Soooo simple

Python is easy to use, freely available and has a rich ecosystem of packages.

Python is slow

- Interpreted instead of compiled
- Does not make use of multiple cores (at least not natively [2])



3

[2] NumPy of course does if one uses an accelerated BLAS library.

[3] `https://julialang.org/benchmarks/`

Python is fast

- Fast Prototyping
- Solve a problem in a fraction of the number of code lines compared to other languages (especially because of the many great libraries and frameworks)
- Installing libraries is a bless with Pip

Why do you want to learn/improve Python skills?

1. It is my first proper programming language
2. My peers are using it in the lab
3. I want to work with libraries that are only available in Python
4. I am curious
5. I want to become a Data Scientist at FAANG and earn 300k a year
6. Other

The Classical Python Shell

```
felix@APOSTLE13 tubs-python-course $ python
Python 3.9.6 (default, Jun 30 2021, 10:22:16)
[GCC 11.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

- Given the python executable is in the current path of the terminal.
- Windows User: Potentially first activate (ana-)conda

IPython, an enhanced Python Shell

```
felix@APOSTLE13 tubs-python-course $ ipython
Python 3.9.6 (default, Jun 30 2021, 10:22:16)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.26.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print("Hello World")
Hello World

In [2]:
```

- Simple Syntax highlighting
- Enhanced Auto-Complete (use ‘tab’)
- Auto-Complete also for filenames

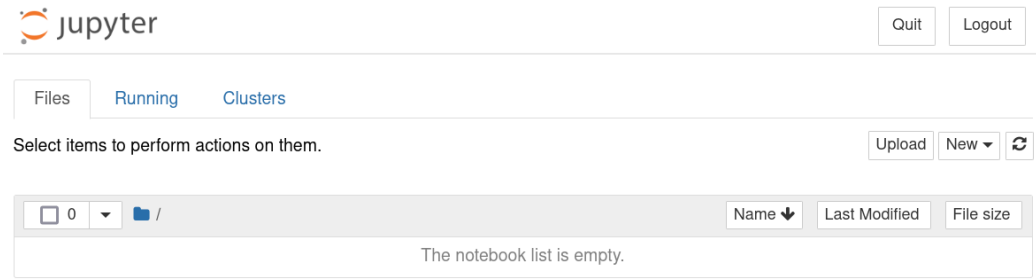
Executing a script

- Also just use the 'Play-Button' in VS Code

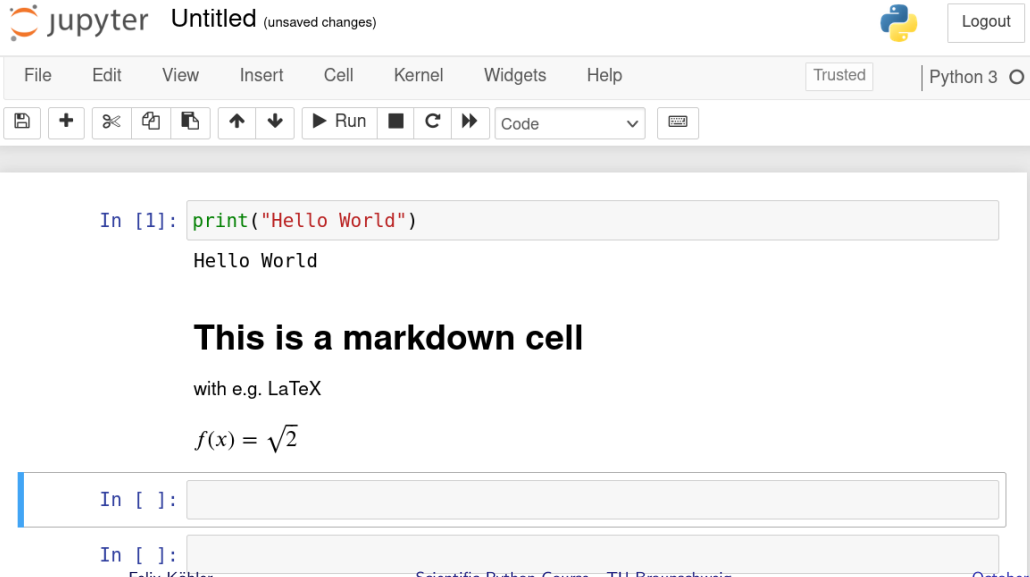
Jupyter Notebooks I

```
felix@APOSTLE13 empty $ jupyter-notebook
[I 10:12:48.165 NotebookApp] Serving notebooks from local directory: /home/felix/Documents/courses/tubs-python-course/empty
[I 10:12:48.165 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 10:12:48.165 NotebookApp] http://localhost:8888/?token=0e8bcc2dd4cf5b3e4b2f95eb5ef7e30e87a0cbe4b1b44dd2
[I 10:12:48.165 NotebookApp]  or http://127.0.0.1:8888/?token=0e8bcc2dd4cf5b3e4b2f95eb5ef7e30e87a0cbe4b1b44dd2
[I 10:12:48.165 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:12:48.196 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/felix/.local/share/jupyter/runtime/nbserver-7179-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=0e8bcc2dd4cf5b3e4b2f95eb5ef7e30e87a0cbe4b1b44dd2
     or http://127.0.0.1:8888/?token=0e8bcc2dd4cf5b3e4b2f95eb5ef7e30e87a0cbe4b1b44dd2
```

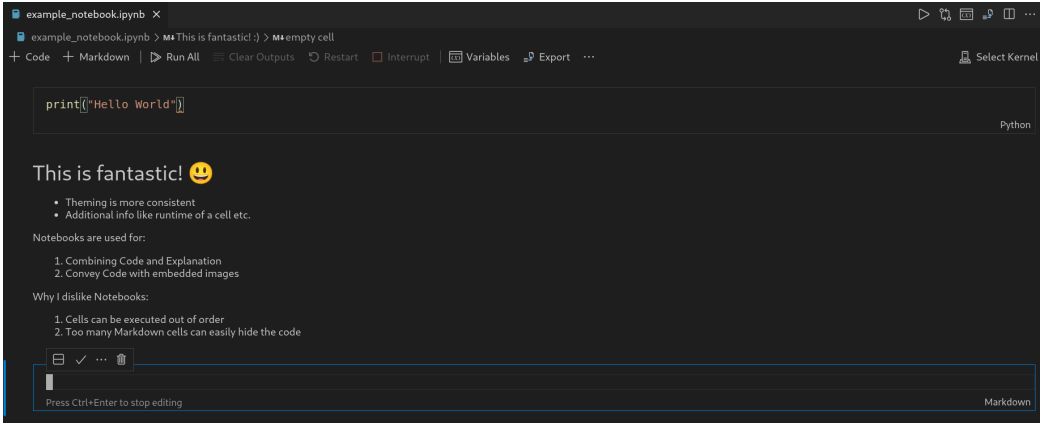
Jupyter Notebooks II



Jupyter Notebooks III



Jupyter Notebooks within VS Code



- Requires VS Code Extension

Other options

- *Jupyter-Lab*: An enhanced Jupyter-Notebook
- Notebooks on some online platforms
  - Google Colab & Kaggle (*free GPUs!*)
  - Binder
  - Notebook Preview on GitHub
- Debugging (handy using VS Code)
- Calling Python from another language
- Creating a binary with your entire environment

Which way would you use for simple and quick calculations?

1. The classical Python shell
2. IPython: The enhanced shell
3. Write a script and execute it
4. Use Jupyter-Notebooks
5. Something else

Which way would you use for data analytics?

1. The classical Python shell
2. IPython: The enhanced shell
3. Write a script (or multiple) and execute it
4. Use Jupyter-Notebooks
5. Something else

Which way would you use for writing scientific software?

1. The classical Python shell
2. IPython: The enhanced shell
3. Write a script (or multiple) and execute it
4. Use Jupyter-Notebooks
5. Something else

Python is dynamically typed

```
In [1]: text = "Hello World"

In [2]: type(text)
Out[2]: str

In [3]: integer = 2

In [4]: type(integer)
Out[4]: int

In [5]: floating_number = 42.0

In [6]: type(floating_number)
Out[6]: float

In [7]: text = floating_number

In [8]: type(floating_number)
Out[8]: float

In [9]:
```

Conditional (If) Statements

```
In [1]: if 5 < 3:
   ...:     print("This is True")
   ...: else:
   ...:     print("This is not True")
   ...:
This is not True

In [2]: something = "Hello World"

In [3]: if something is None:
   ...:     print("It is None")
   ...: elif type(something) is not str:
   ...:     print("It must be something else")
   ...: else:
   ...:     print("It is a string")
   ...:
It is a string
```

For Loops (C-Style)

```
In [1]: for i in range(5):
   ...:     print("Hello World")
   ...:
Hello World
Hello World
Hello World
Hello World
Hello World

In [2]: for i in range(5):
   ...:     print(f"Hello World {i:}")
   ...:
Hello World 0
Hello World 1
Hello World 2
Hello World 3
Hello World 4
```

- Variables declared inside a for-loop are available outside of it

While Loops

```
In [1]: b = 5

In [2]: while b > 2:
   ...:     print(b)
   ...:     b -= 1
   ...:
5
4
3

In [3]: while True:
   ...:     pass
   ...:
^C---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-3-414c137564b4> in <module>
      1 while True:
----> 2     pass
      3

KeyboardInterrupt:
```

- Variables declared inside a while-loop are available outside of it

Python does not have switch statements. Have you ever used them in a different language?

1. Yes
2. No

Functions I

```
In [1]: def say_hi():
   ...:     print("Hello")
   ...:

In [2]: say_hi()
Hello

In [3]: def greeting():
   ...:     name = input("What is your name?")
   ...:     print(f"Hello {name:}")
   ...:     return name
   ...:

In [4]: returned_name = greeting()
What is your name? Felix
Hello  Felix

In [5]: returned_name
Out[5]: ' Felix'

In [6]: name
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-9bc0cb2ed6de> in <module>
----> 1 name

NameError: name 'name' is not defined
```

- Variables declared inside a function are **not** available outside of it

Functions II

```
In [1]: b = 2

In [2]: def add_two(variable):
   ...:     variable += 2
   ...:     return variable
   ...:

In [3]: b_add_two = add_two(b)

In [4]: b
Out[4]: 2

In [5]: b_add_two
Out[5]: 4

In [6]: a = 1; c = 3

In [7]: def add_up_variables(var_1, var_2, var_3):
   ...:     return var_1 + var_2 + var_3
   ...:

In [8]: print(add_up_variables(a, b, c))
6
```

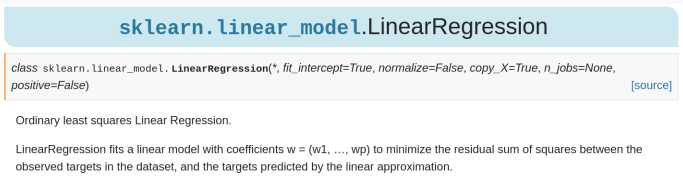
- Primitive datatypes are passed by value, others by reference

Functions III

```
In [1]: def print_strings(first, second):
   ...:     print(f"{first:} {second:}")
   ...:

In [2]: print_strings("Hello", "World")
Hello World

In [3]: print_strings(second="World", first="Hello")
Hello World

In [4]: def print_additional_args(regular, *args, **kwargs):
   ...:     print(regular)
   ...:     print(args)
   ...:     print(kwargs)
   ...:

In [5]: print_additional_args("Python Course", 2, 42, [5.0, 6.0], something=True, something_else=False)
Python Course
(2, 42, [5.0, 6.0])
{'something': True, 'something_else': False}
```

- Keyword arguments ignore order
- Keyword arguments have to come **after** positional arguments
- `*args` and `**kwargs` allow for more, not previously specified (Keyword) arguments

Functions IV



sklearn.linear_model.LinearRegression

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

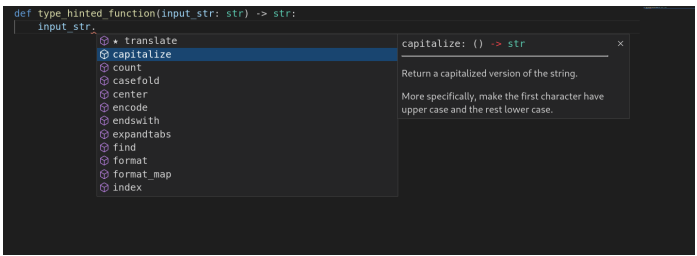
4

- Many libraries have functions with many pre-defined keyword arguments
- Allows to encode default options to certain algorithms

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

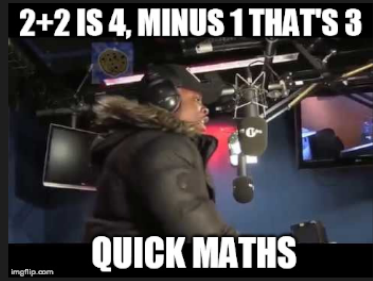
Type Hinting

- Helps the intellisense of an IDE
- Can be checked for by a Linter
- However, types are **not** enforced by the compiler
- Python uses Ducktyping: If the datatype has the required methods, then everything is fine



Simple Math

```
In [1]: 5 / 3   # Correct floating point division by default
Out[1]: 1.6666666666666667

In [2]: 5**3
Out[2]: 125

In [3]: from math import sqrt

In [4]: sqrt(25)
Out[4]: 5.0

In [5]: 3 + 2 - 1
Out[5]: 4

In [6]: (5 + 3j) * 2.5
Out[6]: (12.5+7.5j)
```



[5]

[5] https://www.youtube.com/watch?v=NOmwdg-VKnk

List (=Resizable Array)

```
In [1]: my_favorite_fruits = ["apple", "banana", "kiwi"]

In [2]: my_favorite_fruits[0]
Out[2]: 'apple'

In [3]: my_favorite_vegetables = ["tomato", "potato"]

In [4]: my_favorites = my_favorite_fruits + my_favorite_vegetables

In [5]: my_favorites
Out[5]: ['apple', 'banana', 'kiwi', 'tomato', 'potato']

In [6]: my_favorite_vegetables.append("cucumber")

In [7]: my_favorite_vegetables
Out[7]: ['tomato', 'potato', 'cucumber']

In [8]: some_variables_I_like = ["Hello World", 42, 3.141, None, True]

In [9]: some_variables_I_like
Out[9]: ['Hello World', 42, 3.141, None, True]
```

- Indexing starts from zero (which is also better ;))
- They can hold any type and can be flexibly changed in size

Dictionary (=Associative Map)

```
In [1]: opinions_on_fruits = {
   ...:     "apple": "Sour and Crunchy",
   ...:     "banana": "Sweet and Yellow",
   ...:     "kiwi": "Intense and Tasty",
   ...:     }

In [2]: opinions_on_fruits["banana"]
Out[2]: 'Sweet and Yellow'

In [3]: opinions_on_variables = {
   ...:     42: "The full truth",
   ...:     "Hello World": "Something basic",
   ...:     True: "A Boolean expression",
   ...:     }

In [4]: opinions_on_variables[42]
Out[4]: 'The full truth'

In [5]: opinions_on_variables[True]
Out[5]: 'A Boolean expression'

In [6]: options = {
   ...:     "solver": {
   ...:         "type": "rk45",
   ...:         "rtol": 1e-4,
   ...:         "atol": 1e-4,
   ...:     },
   ...:     "saveat": [0.0, 0.1, 0.2, 0.3]
   ...: }

In [7]: options
Out[7]:
{'solver': {'type': 'rk45', 'rtol': 0.0001, 'atol': 0.0001},
 'saveat': [0.0, 0.1, 0.2, 0.3]}
```

- Commonly used to specify options
- Relation to the JSON file format

Tuples (=Fixed Size Array)

```
In [1]: a_list = [1, 2, 3]

In [2]: a_tuple = (1, 2, 3)

In [3]: a_list.append(4)

In [4]: a_tuple.append(4)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-0a3b3fd15ee9> in <module>
----> 1 a_tuple.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```

- In arguments to functions, one commonly sees Tuples and Lists interchangably

Other Datastructures

- Sets `a = {"Python", 3, [21, 42]}`
- Ordered Dicts, Ordered Sets etc.
- Different types of strings
  - Format strings `mia = 13; print(f"Mia is {mia:03d} years old")`
  - Raw Strings `r"Can contain control \ symbols"`
- ... more from packages (e.g. NumPy Arrays)

Iterating over datastructures

```
In [1]: favorite_fruits = ["apple", "banana", "kiwi"]

In [2]: for element in favorite_fruits:
   ...:     print("Felix likes {:s}".format(element))
   ...:
Felix likes apple
Felix likes banana
Felix likes kiwi

In [3]: for position, element in enumerate(favorite_fruits):
   ...:     rank = position + 1
   ...:     print(f"Rank {rank:}: {element:}")
   ...:
Rank 1: apple
Rank 2: banana
Rank 3: kiwi

In [4]: opinions_on_fruits = {
   ...:     "apple": "Sour and Crunchy",
   ...:     "banana": "Sweet and Yellow",
   ...:     "kiwi": "Intense and Tasty",
   ...:     }

In [5]: for key, value in opinions_on_fruits.items():
   ...:     print(f"About {key:} Felix thinks: {value:}")
   ...:
About apple Felix thinks: Sour and Crunchy
About banana Felix thinks: Sweet and Yellow
About kiwi Felix thinks: Intense and Tasty
```

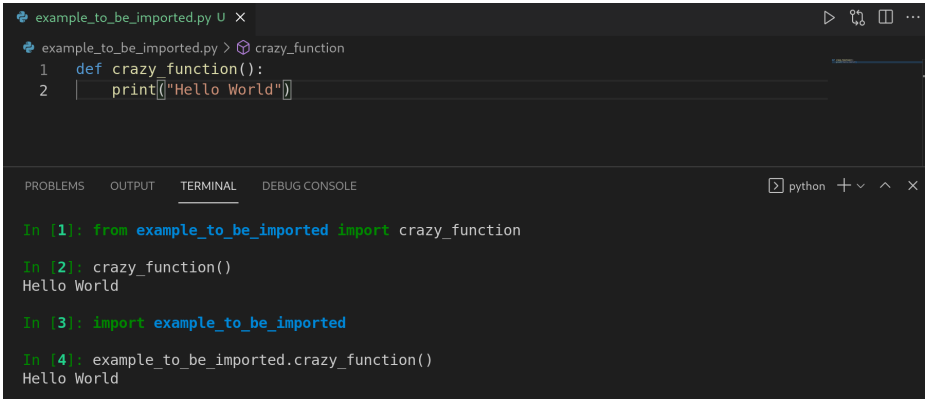
- `range(3)` is also just creating an iterable object

Anonymous Functions (=Lambdas)

```
In [1]: f = lambda x: x**2

In [2]: f(2)
Out[2]: 4

In [3]: f(6.0)
Out[3]: 36.0

In [4]: wrong_capitalization = ["bANanA", "APPLE"]

In [5]: correct_capitalization = list(map(lambda x: x.capitalize(), wrong_capitalization))

In [6]: correct_capitalization
Out[6]: ['Banana', 'Apple']
```

- Commonly used when handing over functions to over functions

Using Code from another file



Paths are relevant

- The Path from which the interpreter is started [6] is the root folder (the 'present working directory')
- The pwd can be changed during runtime `import os; os.chdir("SOMEWHERE ELSE")`
- Paths can be added to `import sys; sys.path.append("SOMEWHERE ELSE")` to search for modules inside of it

```
In [4]: import os

In [5]: os.getcwd()
Out[5]: '/home/felix/Documents/courses/tubs-python-course'

In [6]: os.chdir("src")

In [7]: os.getcwd()
Out[7]: '/home/felix/Documents/courses/tubs-python-course/src'

In [8]: import sys

In [9]: sys.path
Out[9]:
['/usr/bin',
 '/usr/lib/python39.zip',
 '/usr/lib/python3.9',
 '/usr/lib/python3.9/lib-dynload',
 '',
 '/usr/lib/python3.9/site-packages',
 '/usr/lib/python3.9/site-packages/IPython/extensions',
 '/home/felix/.ipython']
```

[6] the pwd of the terminal

Using Packages

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

In [2]: np.linspace(0, 3, 10)
Out[2]:
array([0.        , 0.33333333, 0.66666667, 1.        , 1.33333333,
       1.66666667, 2.        , 2.33333333, 2.66666667, 3.        ])

In [3]: import tensorflow as tf

In [4]: tf.linspace(0, 3, 10)
2021-09-16 11:44:19.149009: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Out[4]:
<tf.Tensor: shape=(10,), dtype=float64, numpy=
array([0.        , 0.33333333, 0.66666667, 1.        , 1.33333333,
       1.66666667, 2.        , 2.33333333, 2.66666667, 3.        ])>
```

- Do not give local files the same name as packages (e.g. do not name a file ‘numpy.py’)
- Use common package abbreviations when importing modules
  - numpy $\rightarrow$ np
  - pandas $\rightarrow$ pd
  - tensorflow $\rightarrow$ tf

Summary

- Python is dynamically typed
- For loops `for sth in iterable`
- While Loops `while true_condition`
- Conditionals `if true_condition: ... elif other_condition: ... else: ...`
- Functions:
  - explicit: `def some_function(arguments): body`
  - anonymous: `anon_function = lambda arguments: body`
- Python uses indentation for scoping
- Built-In Datastructures:
  - Lists `["Hello World", 42]`
  - Dictionaries `{"solver": "rk45", "rtol": 1e-4}`
  - Tuples `("I am", "fixed", "size")`
- Use code from other files / packages: `import numpy as np`

What we did not cover

- More Built-In Functionality for Datastructures (sorting, reversing, list comprehensions . . .)
- File I/O (partially covered later)
- Classes and Object-Oriented Programming (will be covered on day 3)
- The Object-Oriented Nature of Python
- Exceptions
- Python's relation to C/C++/Fortran (but we will exploit this with NumPy and TensorFlow)

NumPy

What are NumPy Arrays? Why should you care?

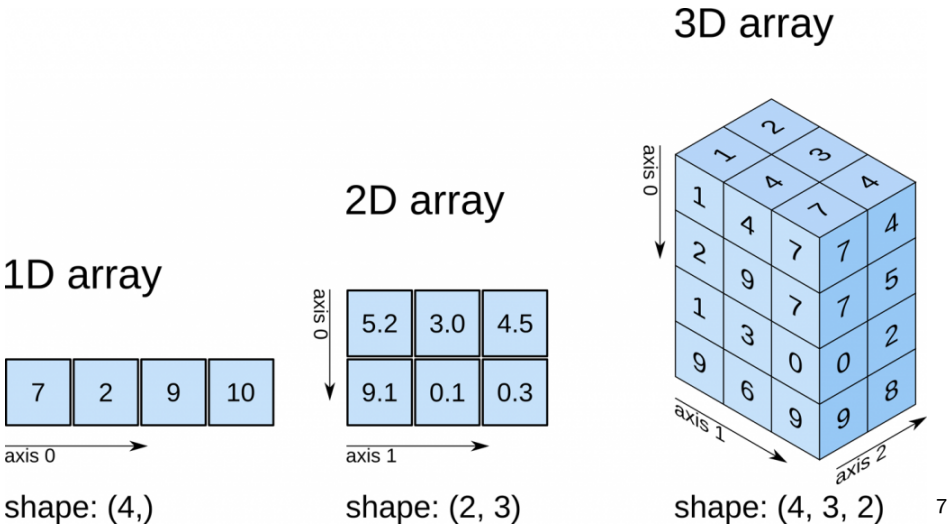
- Software-Components written in C
- Can only contain elements of one datatype (like C/C++ arrays)
- Allow for highly efficient broadcasting and vectorization
- Bind many useful libraries e.g. for linear algebra, Fast Fourier Transform etc.
- $>$ 1000 times more efficient than Python lists
- Cornerstone of almost all scientific computing in Python
- inspired many other packages like *TensorFlow*, *PyTorch*, *Theano* etc.

Ways of creating a NumPy Array

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

In [2]: first_array = np.array([1, 2, 3, 4])

In [3]: first_array.dtype
Out[3]: dtype('int64')

In [4]: float_array = np.array([1.2, 4.8, 9.7])

In [5]: float_array
Out[5]: array([1.2, 4.8, 9.7])

In [6]: float_array.dtype
Out[6]: dtype('float64')

In [7]: float_array[1]
Out[7]: 4.8

In [8]: float_array[2] = 42.0

In [9]: float_array
Out[9]: array([ 1.2,  4.8, 42. ])
```

- NumPy Arrays behave similar to list **but they only allow for one datatype** (=dtype)
- Creation routines similar to Matlab:
  `np.linspace()`, `np.logspace()`, `np.ones()`, `np.zeros()` ...

High-Dimensional Arrays and the shape I



High-Dimensional Arrays and the shape II

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

In [2]: some_matrix = np.array([[1, 2, 3], [4, 5, 6]])

In [3]: some_matrix
Out[3]:
array([[1, 2, 3],
       [4, 5, 6]])

In [4]: some_matrix.shape
Out[4]: (2, 3)

In [5]: some_matrix[1, 2]
Out[5]: 6
```

- Note that the Shape is a tuple

[7] https://predictivehacks.com/tips-about-numpy-arrays/

What is the shape of a 500×500 pixel RGBA image?

1. `(500, 500)`
2. `(500, 500, 3)`
3. `(500, 500, 4)`
4. `(100_000,)`

Slicing

= Creating sub-arrays from arrays

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

In [2]: some_matrix = np.random.rand(3, 4)

In [3]: some_matrix
Out[3]:
array([[0.70327974, 0.23334445, 0.30214029, 0.57711425],
       [0.42243323, 0.84738962, 0.17862322, 0.58919821],
       [0.09049139, 0.52865358, 0.56864201, 0.6612896 ]])

In [4]: extract_from_some_matrix = some_matrix[1, 1:2]

In [5]: extract_from_some_matrix
Out[5]: array([0.84738962])

In [6]: extract_from_some_matrix = some_matrix[1, 1:3]

In [7]: extract_from_some_matrix
Out[7]: array([0.84738962, 0.17862322])
```

A bit more on indexing in Python

- Index starts at 0
- Ranges are always **exclusive the last element**
- Negative indices address elements revers from the end of an iterable object (i.e. -1 is the last element, `[1, 2, 3][-1] = 3`)

Element-wise Multiplication by default

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

In [2]: a_range_of_numbers = np.arange(0, 10, 1)

In [3]: a_range_of_numbers
Out[3]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [4]: a_range_of_numbers * 2.4
Out[4]: array([ 0. ,  2.4,  4.8,  7.2,  9.6, 12. , 14.4, 16.8, 19.2, 21.6])

In [5]: a_range_of_numbers / 2
Out[5]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])

In [6]: np.sqrt(a_range_of_numbers)
Out[6]:
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])

In [7]: a_range_of_numbers**np.pi
Out[7]:
array([  0.        ,   1.        ,   8.82497783,  31.5442807 ,
        77.88023365, 156.99254531, 278.37757775, 451.80787259,
       687.29133511, 995.04164489])
```

Better than in Matlab ;)

Matrix Multiplication I

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 8 \\ 18 \end{bmatrix} \tag{1}$$

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

In [2]: matrix = np.array([[1, 2], [3, 4]])

In [3]: vector = np.array([2, 3])

In [4]: matrix @ vector
Out[4]: array([ 8, 18])
```

Matrix Multiplication II

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix}^T \begin{bmatrix} 1 \\ 5 \end{bmatrix} = 17 \tag{2}$$

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

In [2]: vector_1 = np.array([2, 3])

In [3]: vector_2 = np.array([1, 5])

In [4]: vector_1.T @ vector_2
Out[4]: 17
```

Matrix Multiplication III

Example of a rotation matrix

$$\mathbf{R} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3}$$

$$\tilde{\vec{x}} = \mathbf{R}\vec{x} \tag{4}$$

Matrix Multiplication IV

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

In [2]: def rotation_matrix(angle: float) -> np.ndarray:
   ...:     return np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
   ...:

In [3]: R = rotation_matrix(np.pi/4)

In [4]: R
Out[4]:
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])

In [5]: x = np.array([1, 0])

In [6]: R @ x
Out[6]: array([0.70710678, 0.70710678])
```

Type Hinting for Arrays

Indicate as `def someting(argument: np.ndarray) -> np.ndarray:`
More information is then found in the docstring (i.e. the str below the function head)

```
def fit(self, X, y, sample_weight=None):
    """
    Fit linear model.

    Parameters
    ----------
    X : {array-like, sparse matrix} of shape (n_samples, n_features)
        Training data

    y : array-like of shape (n_samples,) or (n_samples, n_targets)
        Target values. Will be cast to X's dtype if necessary

    sample_weight : array-like of shape (n_samples,), default=None
        Individual weights for each sample
```

[8]

[8] Example from the sourcecode of Scikit-Learn for linear regression

Applying functions element-wise vs. looping

How to apply sin to all elements in an array?

```
my_array = np.array([1, 2, 3, 4])
for i in range(len(my_array)):
    my_array[i] = np.sin(my_array[i])
```

Bad, because of ugly `range(len(my_array))` construct and **slow**. Looping in Python has extreme overhead.

NumPy's core feature is vectorization, applying a function to all elements at the same time. Therefore, better:

```
my_array = np.array([1, 2, 3, 4])
my_array = np.sin(my_array)
```

SIMD vs. Vectorization

Levels of Parallelism in a Computer

1. Process-level
2. Thread-Level
3. Instruction-Level

Reduction Operations

=

Operations, that remove at least one axis from the shape

```
In [2]: a_2d_array = np.array([[1, 2, 3], [4, 5, 6]])

In [3]: a_2d_array
Out[3]:
array([[1, 2, 3],
       [4, 5, 6]])

In [4]: a_2d_array.mean()
Out[4]: 3.5

In [5]: a_2d_array.mean(axis=0)
Out[5]: array([2.5, 3.5, 4.5])

In [6]: np.mean(a_2d_array, axis=0)
Out[6]: array([2.5, 3.5, 4.5])

In [7]: np.mean(a_2d_array, axis=0).shape
Out[7]: (3,)

In [8]: np.mean(a_2d_array, axis=0, keepdims=True).shape
Out[8]: (1, 3)

In [9]: np.squeeze(np.mean(a_2d_array, axis=0, keepdims=True)).shape
Out[9]: (3,)
```

- An axis with 1 element is proxy and can be ignored (e.g. `np.squeeze`)

Exploiting Modern Computer Architecture

- Vector Instruction Units in each CPU core
- Multiple Cores per CPU
- (With NumPy extensions): Use of accelerators (components specifically designed for certain computations)

How many cores does your current computer have?

1. 1
2. 2
3. 4
4. 8
5. 16
6. More

Efficient Element-Wise Multiplication in high dimensions I

Given the vector and the Matrix

$$\vec{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{5}$$

How to efficiently do row-wise multiplication.
Idea blow-up $\vec{x}$ to $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix} \tag{6}$$

And then element-wise multiplication since the shape match.
But this is **memory-intensive**.

Efficient Element-Wise Multiplication in high dimensions II

Create a proxy-dimension and use NumPy's auto-broadcasting.

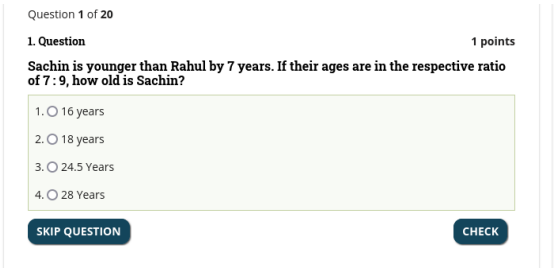
```
In [2]: vector = np.array([1, 2])

In [3]: matrix = np.array([[1, 2, 3], [4, 5, 6]])

In [4]: vector.shape
Out[4]: (2,)

In [5]: vector_with_proxy_axis = vector.reshape((-1, 1))

In [6]: vector_with_proxy_axis.shape
Out[6]: (2, 1)

In [7]: matrix.shape
Out[7]: (2, 3)

In [8]: vector_with_proxy_axis * matrix
Out[8]:
array([[ 1,  2,  3],
       [ 8, 10, 12]])
```

That is the great power of NumPy, making numerical computations worthwhile in such a slow language as Python.

Sachin is younger than Rahul by 7 years. If there ages are in the respective ratio of 7 : 9, how old is Sachin?

1. 10 years
2. 18 years
3. 24.5 years
4. 28 years

Solving Linear Systems I



9

$$\begin{bmatrix} -1 & 1 \\ -9 & 7 \end{bmatrix} \begin{bmatrix} S \\ R \end{bmatrix} = \begin{bmatrix} 7 \\ 0 \end{bmatrix} \tag{7}$$

Solving Linear Systems II

```
In [2]: system_matrix = np.array([[-1, 1], [-9, 7]])

In [3]: right_hand_side = np.array([7, 0])

In [4]: np.linalg.solve(system_matrix, right_hand_side)
Out[4]: array([24.5, 31.5])
```

`np.linalg.solve` works like the backward slash operator in MATLAB
Also solves over-determined (least-squares) systems

[9] https://examtimequiz.com/maths-quiz-on-problem-on-ages/

Random Numbers

- Generate uniformly distributed random numbers in range $[0, 1]$

  `np.random.rand(SHAPE)`

- Generate normally distributed random numbers with $\mu = 0$ and $\sigma = 1$

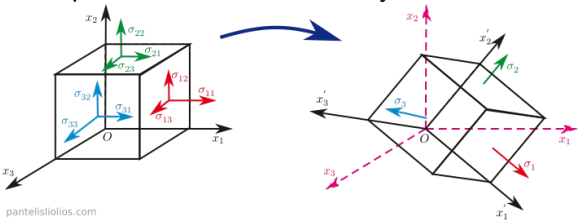
  `np.random.randn(SHAPE)`

Note that for compatibility with MATLAB the shape is entered directly and not inside a tuple

```
In [2]: np.mean(np.random.rand(34, 30))
Out[2]: 0.5045344106456807

In [3]: np.mean(np.random.randn(45, 11, 101))
Out[3]: -0.004122787376187225
```

Eigenvalues I

Principal Stresses of the Cauchy stress tensor.



10

$$\sigma = \begin{bmatrix} 5 & 3 & 1 \\ 3 & 9 & 2 \\ 1 & 2 & 6 \end{bmatrix} = \begin{bmatrix} 0.43 & 0.87 & -0.21 \\ 0.82 & -0.48 & -0.32 \\ 0.38 & 0.03 & 0.93 \end{bmatrix} \begin{bmatrix} 11.51 & 0 & 0 \\ 0 & 3.39 & 0 \\ 0 & 0 & 5.10 \end{bmatrix} \begin{bmatrix} 0.43 & 0.87 & -0.21 \\ 0.82 & -0.48 & -0.32 \\ 0.38 & 0.03 & 0.93 \end{bmatrix}^T \tag{8}$$

Eigenvalues II

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

In [2]: sigma = np.array([[5, 3, 1], [3, 9, 2], [1, 2, 6]])

In [3]: eigenvalues, eigenvectors = np.linalg.eig(sigma)

In [4]: eigenvalues
Out[4]: array([11.51203667,  3.39243865,  5.09552468])

In [5]: eigenvectors
Out[5]:
array([[ 0.4347332 ,  0.8764332 , -0.20705527],
       [ 0.81839365, -0.4804335 , -0.31530222],
       [ 0.37581762,  0.03238037,  0.92612776]])

In [6]: eigenvalue_matrix = np.diag(eigenvalues)

In [7]: eigenvalue_matrix
Out[7]:
array([[11.51203667,  0.        ,  0.        ],
       [ 0.        ,  3.39243865,  0.        ],
       [ 0.        ,  0.        ,  5.09552468]])

In [8]: eigenvectors @ eigenvalue_matrix @ eigenvectors.T
Out[8]:
array([[5., 3., 1.],
       [3., 9., 2.],
       [1., 2., 6.]])
```

[10] https://www.pantelisliolios.com/principal-stresses-and-invariants/

Documentation is key

numpy.linalg.eig

`linalg.eig(a)` [source]

Compute the eigenvalues and right eigenvectors of a square array.

Parameters: `a` : *(..., M, M) array*
Matrices for which the eigenvalues and right eigenvectors will be computed

Returns: `w` : *(..., M) array*
The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When *a* is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs

`v` : *(..., M, M) array*
The normalized (unit "length") eigenvectors, such that the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`.

Raises: LinAlgError
If the eigenvalue computation does not converge.

11

[11] https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html

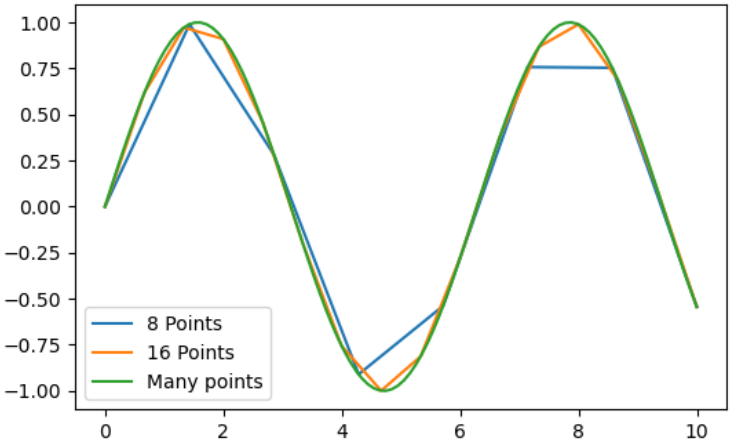
What we did not cover

- Fast-Fourier Transform
- Efficient (Tensor/) Matrix Multplication in high dimensions using Einstein Notation `np.einsum`
- (Arg-) Sorting `np.argsort` (also allows for the selection of an axis it is applied on)
- Saving and loading NumPy arrays
- The concept of ufunc
- The rules of auto-broadcasting (you will learn them intuitively when using NumPy for long enough)

Matplotlib

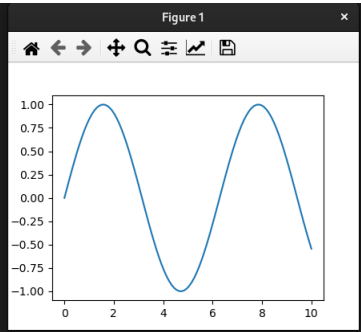
Refresher: How drawing works on a Computer

Approximate Curves by finite number of points (and lines)



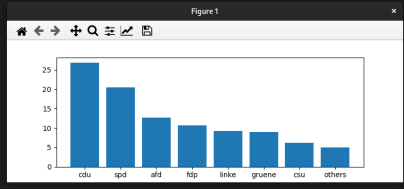
Line Plot with classical interface

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

In [2]: import matplotlib.pyplot as plt

In [3]: x = np.linspace(0, 10, 100)

In [4]: y = np.sin(x)

In [5]: plt.plot(x, y)
Out[5]: [<matplotlib.lines.Line2D at 0x7f8ae5f46f10>]

In [6]: plt.show()
```



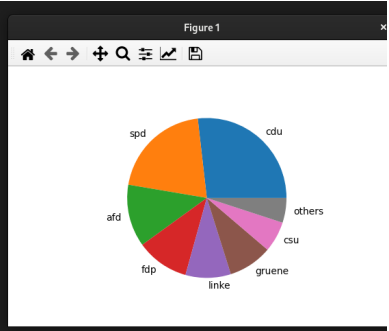
Bar Plot with classical interface

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

In [2]: import matplotlib.pyplot as plt

In [3]: results = np.genfromtxt("https://github.com/Ceyron/machine-learning-and-simulation/files/7183521/election_2017_results.csv")

In [4]: parties = np.genfromtxt("https://github.com/Ceyron/machine-learning-and-simulation/files/7183522/election_2017_parties.csv", dtype=str)

In [5]: plt.bar(parties, results)
Out[5]: <BarContainer object of 8 artists>

In [6]: plt.show()
```



Pie Plot with classical interface

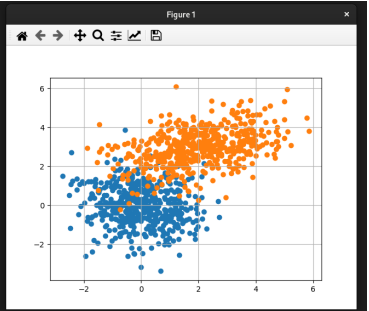
```
In [8]: plt.pie(x=results, labels=parties)
Out[8]:
([<matplotlib.patches.Wedge at 0x7f246a6f95e0>,
  <matplotlib.patches.Wedge at 0x7f246a7142b0>,
  <matplotlib.patches.Wedge at 0x7f2403be83d0>,
  <matplotlib.patches.Wedge at 0x7f2403bd9df0>,
  <matplotlib.patches.Wedge at 0x7f240380f850>,
  <matplotlib.patches.Wedge at 0x7f2403bd7700>,
  <matplotlib.patches.Wedge at 0x7f241004cf70>,
  <matplotlib.patches.Wedge at 0x7f241031a400>],
 [Text(0.7319212066816395, 0.8211524506507258, 'cdu'),
  Text(-0.7573783283031301, 0.7977330805581252, 'spd'),
  Text(-1.071141753279434, -0.2503104959474541, 'afd'),
  Text(-0.6288068482072894, -0.9025530165301178, 'fdp'),
  Text(0.019025061701251567, -1.0998354636159282, 'linke'),
  Text(0.6087886295921922, -0.9161748765815731, 'gruene'),
  Text(0.9603156999338709, -0.5364641241131787, 'csu'),
  Text(1.0864301627089432, -0.17224837170847082, 'others')])

In [9]: plt.show()
```



Scatter Plot with classical interface

$$\vec{X} \propto \mathcal{N}(\vec{0}, \mathbf{I}) \qquad \vec{Y} = \vec{\mu} + \mathbf{L}\vec{X} \quad \rightarrow \quad \vec{Y} \propto \mathcal{N}(\vec{\mu}, \mathbf{\Sigma}) \quad \text{if} \quad \mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T \tag{9}$$

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

In [2]: import matplotlib.pyplot as plt

In [3]: standard_mv_normal_data = np.random.randn(500, 2)

In [4]: plt.scatter(standard_mv_normal_data[:, 0], standard_mv_normal_data[:, 1])
Out[4]: <matplotlib.collections.PathCollection at 0x7f750332a430>

In [5]: mu = np.array([2, 3])

In [6]: cov = np.array([[2, 0.7], [0.7, 1]])

In [7]: L = np.linalg.cholesky(cov)

In [8]: custom_mv_normal_data = mu + standard_mv_normal_data @ L.T

In [9]: plt.scatter(custom_mv_normal_data[:, 0], custom_mv_normal_data[:, 1])
Out[9]: <matplotlib.collections.PathCollection at 0x7f75612b8910>

In [10]: plt.grid()

In [11]: plt.show()
```

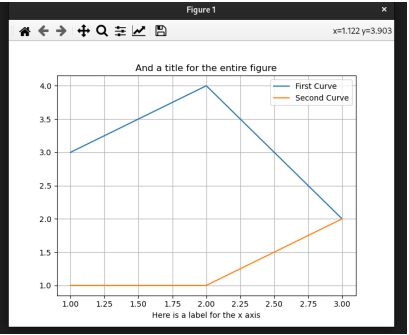


- Have you seen the auto-broadcasting when adding mu?

The interactive mode

No need to do plt.show() anymore. Instantly updates!

```
In [1]: import matplotlib.pyplot as plt

In [2]: plt.ion()
Out[2]: <matplotlib.pyplot._IonContext at 0x7f5af1830430>

In [3]: plt.figure()
Out[3]: <Figure size 640x480 with 0 Axes>

In [4]: plt.plot([1, 2, 3], [3, 4, 3], label="First Curve")
Out[4]: [<matplotlib.lines.Line2D at 0x7f5a7cb7fd60>]

In [5]: plt.plot([1, 2, 3], [1, 1, 3], label="Second Curve")
Out[5]: [<matplotlib.lines.Line2D at 0x7f5a981caac0>]

In [6]: plt.legend()
Out[6]: <matplotlib.legend.Legend at 0x7f5aa0abd970>

In [7]: plt.grid()

In [8]: plt.xlabel("Here is a label for the x axis")
Out[8]: Text(0.5, 23.52222222222222, 'Here is a label for the x axis')

In [9]: plt.title("And a title for the entire figure")
Out[9]: Text(0.5, 1.0, 'And a title for the entire figure')
```



Using Matplotlib in Jupyter-Notebooks

```
import numpy as np
import matplotlib.pyplot as plt
```
[1] ✓ 0.4s Python

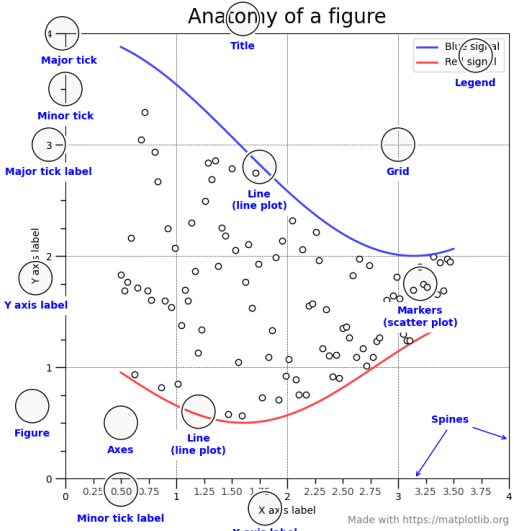
```
data = np.genfromtxt(
        "https://www.marketwatch.com/investing/stock/aapl"
        "/downloaddatapartial?startdate=10/15/2020%"
        "2000:00:00&enddate=09/16/2021%2000:00:00&"
        "daterange=d30&frequency=p1d&csvdownload="
        "true&downloadpartial=false&newdates=false",
        skip_header=1,
        delimiter=",",
        dtype=str
)
```
[18] ✓ ✓ 0.2s Python

```
days = data[:, 0]
opening_price = np.char.replace(data[:, 1], '"', '').astype(np.float32)
```
[19] ✓ 0.2s Python

```
plt.plot(days, opening_price)
```
[20] ✓ 0.2s Python

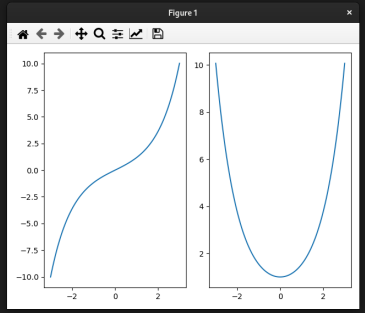
```
[<matplotlib.lines.Line2D at 0x7fa36eb0d310>]
```

Anatomy of a Figure



Subplots - the easy way

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

In [2]: import matplotlib.pyplot as plt

In [3]: x = np.linspace(-3, 3, 100)

In [4]: y_1 = np.sinh(x)

In [5]: y_2 = np.cosh(x)

In [6]: plt.ion()
Out[6]: <matplotlib.pyplot._IonContext at 0x7f8245639eb0>

In [7]: plt.figure()
Out[7]: <Figure size 640x480 with 0 Axes>

In [8]: plt.subplot(121)
Out[8]: <AxesSubplot:>

In [9]: plt.plot(x, y_1)
Out[9]: [<matplotlib.lines.Line2D at 0x7f81ec8ff1f0>]

In [10]: plt.subplot(122)
Out[10]: <AxesSubplot:>

In [11]: plt.plot(x, y_2)
Out[11]: [<matplotlib.lines.Line2D at 0x7f81d0f77730>]

In [12]: plt.tight_layout()
```



Documentation is key

matplotlib.pyplot.plot

`matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)` [source]

Plot y versus x as lines and/or markers.

Call signatures:

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

The coordinates of the points or line nodes are given by *x, y*.

The optional parameter *fmt* is a convenient way for defining basic formatting like color, marker and linestyle. It's a shortcut string notation described in the *Notes* section below.

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

You can use `Line2D` properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
...      linewidth=2, markersize=12)
```

When conflicting with *fmt*, keyword arguments take precedence.

13

[13] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

What we did not cover

- Plotting in 3D (we will do this with Plotly ;))
- Image Plots
- Heatmaps
- Histogram
- Streamplots
- Interactivity
- Other APIs for Matplotlib (although the presented will likely cover 90% of what you want to do with it.)

You can find examples scripts here:
`https://matplotlib.org/stable/tutorials/introductory/sample_plots.html`

Pandas, Seaborn & Plotly

Datascience = Python?

- Working with big data becomes more and more prevalent
- In Science and Engineering: The Future will be hybrid models of Data Science/Machine Learning + Physical Models (sometimes called Scientific Machine Learning)
- Python provides easy access to data handling tools, Machine Learning and Scientific Computing
- A 'Glue Language', that allows you to control complex existing libraries

Intro to DataFrames

A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types

	A	B	C	D	E	F	G
1	party	result	seats	chairman	budget_2020_pro	budget_2020_con	budget_2020_neutral
2	cdu/csu	33.1	245	Ralph Brinkhaus	236	0	10
3	spd	20.5	152	Rolf Mützenich	135	0	17
4	afd	12.6	87	Alexander Gaulano	0	83	8
5	fdp	10.7	80	Christian Lindner	0	69	10
6	linke	9.2	69	Dietmar Bartsch	0	57	12
7	gruene	8.9	67	Anton Hofreiter	0	60	7
8	others	5	9	ohne	0	1	3

Creating a DataFrame

```
In [1]: import pandas as pd

In [2]: an_example_df = pd.DataFrame()

In [3]: an_example_df
Out[3]:
Empty DataFrame
Columns: []
Index: []

In [4]: an_example_df["party"] = ["cdu/csu", "spd"]

In [5]: an_example_df["result"] = [33.1, 20.5]

In [6]: an_example_df
Out[6]:
     party  result
0  cdu/csu    33.1
1      spd    20.5

In [7]: an_example_df.dtypes
Out[7]:
party      object
result    float64
dtype: object
```

Reading in a database

```
In [1]: import pandas as pd

In [2]: df_from_excel = pd.read_excel("https://github.com/Ceyron/machine-learning-and-simulation/files/7235037/19th_bundestag_example.ods")

In [3]: df_from_excel
Out[3]:
     party  result  seats          chairman  budget_2020_pro  budget_2020_con  budget_2020_neutral
0   cdu/csu    33.1    245    Ralph Brinkhaus              236                0                   10
1      spd    20.5    152      Rolf Mützenich              135                0                   17
2      afd    12.6     87  Alexander Gauland                0               83                    8
3      fdp    10.7     80  Christian Lindner                0               69                   10
4    linke     9.2     69    Dietmar Bartsch                0               57                   12
5   gruene     8.9     67    Anton Hofreiter                0               60                    7
6   others     5.0      9              ohne                 0                1                    3

In [4]: df_from_csv = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235038/19th_bundestag_example.csv")

In [5]: df_from_csv
Out[5]:
     party  result  seats          chairman  budget_2020_pro  budget_2020_con  budget_2020_neutral
0   cdu/csu    33.1    245    Ralph Brinkhaus              236                0                   10
1      spd    20.5    152      Rolf Mützenich              135                0                   17
2      afd    12.6     87  Alexander Gauland                0               83                    8
3      fdp    10.7     80  Christian Lindner                0               69                   10
4    linke     9.2     69    Dietmar Bartsch                0               57                   12
5   gruene     8.9     67    Anton Hofreiter                0               60                    7
6   others     5.0      9              ohne                 0                1                    3
```

Basic Usage of DataFrames

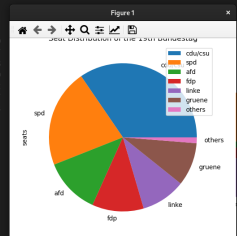
```
In [3]: bundestag_19["party"]
Out[3]:
0    cdu/csu
1        spd
2        afd
3        fdp
4      linke
5     gruene
6     others
Name: party, dtype: object

In [4]: bundestag_19[["party", "result", "seats"]]
Out[4]:
     party  result  seats
0  cdu/csu    33.1    245
1      spd    20.5    152
2      afd    12.6     87
3      fdp    10.7     80
4    linke     9.2     69
5   gruene     8.9     67
6   others     5.0      9

In [5]: bundestag_19.iloc[3, :]
Out[5]:
party                              fdp
result                            10.7
seats                               80
chairman             Christian Lindner
budget_2020_pro                      0
budget_2020_con                     69
budget_2020_neutral                 10
Name: 3, dtype: object
```

Working with DataFrames

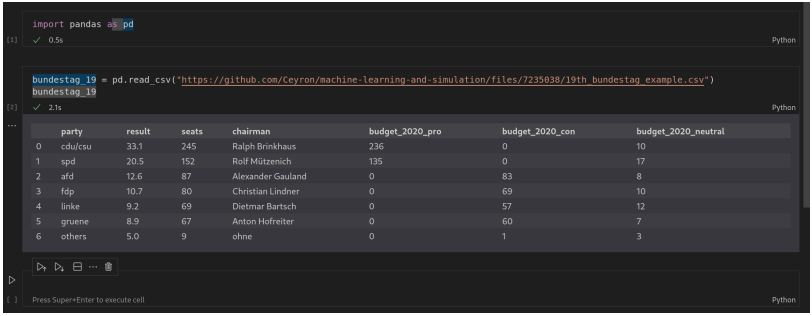
```
In [12]: bundestag_19[["result", "seats"]].sum()
Out[12]:
result    100.0
seats     709.0
dtype: float64

In [13]: bundestag_19[["result", "seats"]].mean()
Out[13]:
result     14.285714
seats     101.285714
dtype: float64

In [14]: bundestag_19[["result", "seats"]].std()
Out[14]:
result     9.568948
seats     75.957695
dtype: float64

In [15]: bundestag_19[bundestag_19["result"] > 10.0]
Out[15]:
     party  result  seats          chairman  budget_2020_pro  budget_2020_con  budget_2020_neutral
0  cdu/csu    33.1    245    Ralph Brinkhaus              236                0                   10
1      spd    20.5    152     Rolf Mützenich              135                0                   17
2      afd    12.6     87  Alexander Gauland                0               83                    8
3      fdp    10.7     80  Christian Lindner                0               69                   10
```

Quick Plotting from the DataFrame

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt

In [3]: bundestag_19 = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235038/19th_bundestag_example.csv", index_col="p
   ...: arty")

In [4]: bundestag_19
Out[4]:
        result  seats           chairman  budget_2020_pro  budget_2020_con  budget_2020_neutral
party
cdu/csu    33.1    245     Ralph Brinkhaus              236                0                   10
spd        20.5    152       Rolf Mützenich             135                0                   17
afd        12.6     87   Alexander Gauland                0               83                    8
fdp        10.7     80   Christian Lindner                0               69                   10
linke       9.2     69     Dietmar Bartsch                0               57                   12
gruene      8.9     67     Anton Hofreiter                0               60                    7
others      5.0      9                ohne                0                1                    3

In [5]: bundestag_19.plot.pie(y="seats")
Out[5]: <AxesSubplot:ylabel='seats'>

In [6]: plt.tight_layout()

In [7]: plt.title("Seat Distribution of the 19th Bundestag")
Out[7]: Text(0.5, 1.0, 'Seat Distribution of the 19th Bundestag')

In [8]: plt.show()
```



Pandas in Jupyter Notebooks



```
import pandas as pd
```

[1] ✓ 0.5s Python

```
bundestag_19 = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235038/19th_bundestag_example.csv")
bundestag_19
```

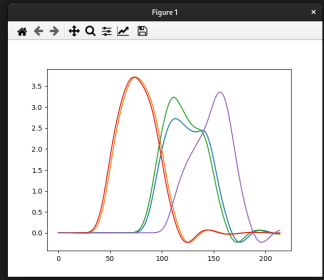
[2] ✓ 2.1s Python

	party	result	seats	chairman	budget_2020_pro	budget_2020_con	budget_2020_neutral
0	cdu/csu	33.1	245	Ralph Brinkhaus	236	0	10
1	spd	20.5	152	Rolf Mützenich	135	0	17
2	afd	12.6	87	Alexander Gauland	0	83	8
3	fdp	10.7	80	Christian Lindner	0	69	10
4	linke	9.2	69	Dietmar Bartsch	0	57	12
5	gruene	8.9	67	Anton Hofreiter	0	60	7
6	others	5.0	9	ohne	0	1	3

[ ] Press Super+Enter to execute cell Python

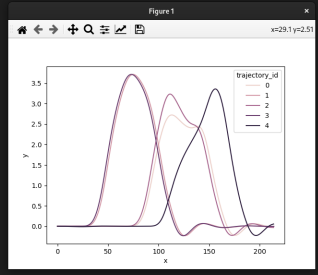
Visualizing Trajectories - The naive way

```
In [1]: import pandas as pd

In [2]: trajectories = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235349/takeover_trajectories.csv.g
   ...: z")

In [3]: trajectories
Out[3]:
      trajectory_id   time           x           y
0                 0   0.00    0.000000  0.000000e+00
1                 0   0.01    0.000000  0.000000e+00
2                 0   0.02    0.221986 -1.087682e-08
3                 0   0.03    0.443752 -2.506980e-08
4                 0   0.04    0.665335 -4.079909e-08
...             ...    ...         ...          ...
5000              4   9.96  212.747779  4.973857e-02
5001              4   9.97  212.967307  5.130742e-02
5002              4   9.98  213.186842  5.276027e-02
5003              4   9.99  213.406384  5.409817e-02
5004              4  10.00  213.625932  5.532225e-02

[5005 rows x 4 columns]

In [4]: import matplotlib.pyplot as plt

In [5]: plt.ion()
Out[5]: <matplotlib.pyplot._IonContext at 0x7fcedb1b59a0>

In [6]: plt.figure()
Out[6]: <Figure size 640x480 with 0 Axes>

In [7]: for i in range(5):
   ...:     plt.plot(trajectories[trajectories["trajectory_id"] == i]["x"], trajectories[trajectories["trajectory_id"] == i]["y"])
   ...:
```



Visualizing Trajectories with seaborn

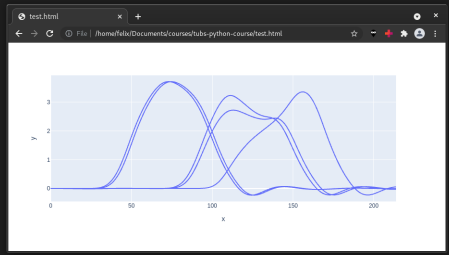
```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt

In [3]: import seaborn as sns

In [4]: trajectories = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235349/takeover_trajectories.csv.gz")

In [5]: sns.lineplot(data=trajectories, x="x", y="y", hue="trajectory_id")
Out[5]: <AxesSubplot:xlabel='x', ylabel='y'>

In [6]: plt.show()
```



- Seaborn is built on-top of Matplotlib
- Many visually pleasing presets

Visualizing Trajectories with Plotly

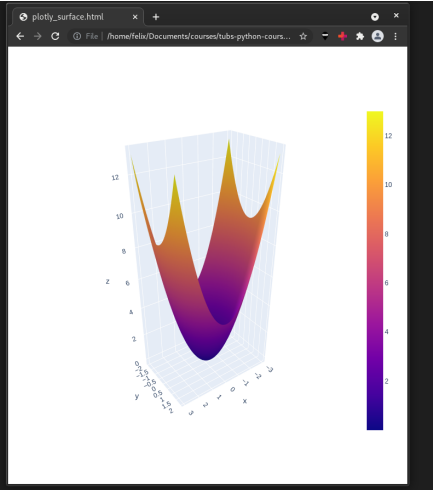
```
In [1]: import pandas as pd
In [2]: import plotly.express as px
In [3]: trajectories = pd.read_csv("https://github.com/Ceyron/machine-learning-and-simulation/files/7235349/takeover_trajectories.csv.gz")
In [4]: fig = px.line(data_frame=trajectories, x="x", y="y", line_group="trajectory_id")
In [5]: fig.show()
In [6]:
```



- Plotly is completely different from Matplotlib
- It relies on rendering in the web browser instead of stand-alone windows
- Plotly Express is a high-level convenience interface

3D Plots with plotly I

```
In [1]: import numpy as np
In [2]: import plotly.graph_objects as go
In [3]: x = np.linspace(-3, 3, 100)
In [4]: y = np.linspace(-2, 2, 100)
In [5]: X, Y = np.meshgrid(x, y)
In [6]: Z = X**2 + Y**2
In [7]: fig = go.Figure(go.Surface(x=x, y=y, z=Z))
In [8]: fig.write_html("plotly_surface.html")
In [9]:
```



3D Plots with plotly II

- 3D Plots in general can be difficult
  - Whether x & y have to be two-dimensional or one-dimensional
- Even within Plotly the interface is inconsistent

Which presented Plotting Library did you like the most?

1. Matplotlib: The classic
2. Seaborn: The re-imagined classic
3. Plotly: The new-kid

What we did not cover

- Pandas
  - Dataframe Joins
  - Data cleaning (batched type conversion ...)
  - Grouping (or did we?)
- Seaborn
  - Plot Aesthetics
  - Statistical Plotting
- Plotly
  - Plotly Express
  - Plotly Figure Factory (templates for many commonly used advanced visualizations)
  - Hosting applications with dash (usually requires paying)
- Data Science in general
  - Accessing databases with SQL

What was the most interesting yesterday?

1. The cool questions
2. Hands-on coding
3. Learning to create fancy visualizations
4. Learning about high-dimensional linear algebra
5. Python in general

SciPy: Optimize & Statistics

A bit of history

- Python was first released by Guido van Rossum in 1991
- Python was **not** initially designed for array computing (=scientific computing)
- First Array Computing Package 'Numeric'
- Later, another Array Computing Package 'Numarray'
- 2005, packages unified into NumPy
- Implementation of many numerical algorithms based on NumPy and by binding existing Fortran/C/C++ Libraries (like MATLAB does it)

The components of SciPy

- **cluster**: hierarchical clustering, vector quantization, K-means
- **constants**: physical constants and conversion factors
- **fft**: Discrete Fourier Transform algorithms
- **fftpack**: Legacy interface for Discrete Fourier Transforms
- **integrate**: numerical integration routines
- **interpolate**: interpolation tools
- **io**: data input and output
- **linalg**: linear algebra routines
- **misc**: miscellaneous utilities (e.g. example images)
- **ndimage**: various functions for multi-dimensional image processing
- **ODR:** orthogonal distance regression classes and algorithms
- **optimize**: optimization algorithms including linear programming
- **signal**: signal processing tools
- **sparse**: sparse matrices and related algorithms
- **spatial**: algorithms for spatial structures such as k-d trees, nearest neighbors, Convex hulls, etc.
- **special**: special functions
- **stats**: statistical functions
- **weave**: tool for writing C/C++ code as Python multiline strings (now deprecated in favor of Cython[6])

A simple example of optimization

$$\min_x f(x) = x^2 \tag{10}$$

```
In [1]: from scipy import optimize

In [2]: f = lambda x: x**2

In [3]: optimize.minimize_scalar(f)
Out[3]:
     fun: 0.0
    nfev: 8
     nit: 4
 success: True
       x: 0.0

In [4]: result_object = optimize.minimize_scalar(f)
```

A quick classification of optimization problems/algorithms

- Usage of derivative information
  - Gradient-Free (mostly heuristics and probabilistic based)
  - First-Order Methods (e.g. gradient descent)
  - Second-Order Methods (e.g. Newton Method)
- Convexity
  - Convex
    - Linear Programming
    - Quadratic Programming / Least Squares
  - Nonconvex
- Constraints
  - Unconstrained
  - Equality-Constrained
  - Inequality-Constrained
- Smoothness / Continuity

Further resources:

`https://scipy-lectures.org/advanced/mathematical_optimization/`

Documentation is key

scipy.optimize.minimize

scipy.optimize.minimize(*fun*, *x0*, *args=()*, *method=None*, *jac=None*, *hess=None*, *hessp=None*, *bounds=None*, *constraints=()*, *tol=None*, *callback=None*, *options=None*) [source]

Minimization of scalar function of one or more variables.

Parameters: **fun : *callable***

The objective function to be minimized.

```
fun(x, *args) -> float
```

where `x` is an 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where 'n' is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

Simple Multivariate Example with Bounds

```
In [1]: from scipy import optimize

In [2]: f = lambda x: x[0]**2 + x[1]**2

In [3]: optimize.minimize(f, [1.5, 1.5], bounds=[(1, 2), (1, 2)])
Out[3]:
      fun: 2.0
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([2., 2.])
  message: 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     nfev: 6
      nit: 1
     njev: 2
   status: 0
  success: True
        x: array([1., 1.])
```

One could also provide the Jacobian (=Gradient) and Hessian information explicitly. In this case, Scipy is approximating them by Finite Differences.

Typical Pitfalls in Numerical Optimization

- Local vs. Global Optima (convexity: Local optima are Global ones) - However, often a 'good' local optimum is sufficient
- Method being unable to converge
- Usage of a bad initial point
- Running out of memory to store large objects (like Hessian matrices)

Optimization in Machine Learning

Important Optimization in Machine Learning is related to optimizing the parameters of a function, not the inputs to the function.

The World of Distributions

```
In [1]: from scipy import stats

In [2]: stats.
```

alexandergovern()	burr	exponweib
alpha	burr12	f
anderson()	cauchy	f_oneway
anderson_ksamp()	chi	F_onewayBadInputSizesWarning
anglit	chi2	F_onewayConstantInputWarning
ansari()	chi2_contingency()	fatiguelife
arcsine	chisquare()	find_repeats
argus	circmean	fisher_exact
barnard_exact()	circstd	fisk
bartlett()	circvar	fligner
bayes_mvs()	combine_pvalues	foldcauchy
bernoulli	contingency	foldnorm
beta	cosine	friedmanchisquare
betabinom	cramervonmises	gamma
betaprime	cramervonmises_2samp	gausshyper
biasedurn	crystalball	gaussian_kde
binned_statistic()	cumfreq	genexpon
binned_statistic_2d()	describe	genextreme
binned_statistic_dd()	dgamma	gengamma
binom	differential_entropy	genhalflogistic
binom_test()	dirichlet	genhyperbolic
binomtest()	distributions	geninvgauss
boltzmann	dlaplace	genlogistic
bootstrap()	dweibull	gennorm
boschloo_exact()	energy_distance	genpareto
boxcox()	entropy	geom
boxcox_llf()	epps_singleton_2samp	gilbrat
boxcox_normmax()	erlang	gmean
boxcox_normplot()	expon	gompertz
bradford	exponnorm	gstd
brunnermunzel()	exponpow	gumbel_l

Example of how to use Scipy Stats

```
In [1]: from scipy import stats

In [2]: grade = stats.norm(2.2, 0.6)

In [3]: grade.rvs(10)
Out[3]:
array([2.02721178, 2.02025534, 2.06439575, 2.26921928, 2.26747895,
       2.36716452, 3.0259937 , 3.16200956, 1.42972126, 2.53666024])

In [4]: grade.pdf(2.0)
Out[4]: 0.6289720461549887

In [5]: grade.cdf(2.5)
Out[5]: 0.691462461274013

In [6]: grade.stats()
Out[6]: (array(2.2), array(0.36))

In [7]: grade.ppf(0.8)
Out[7]: 2.7049727401437487
```

Statistical Tests with SciPy

Statistical Tests consist of the following

- Assumptions on the data
- A test statistic
- A p-value
- A Null Hypothesis

The Null Hypothesis is usually that everything is 'as usual', e.g. a drug has no effect, a physical phenomenon does not exist etc.
If the p-value is below a threshold α (usually 0.05%) the Null Hypothesis is rejected in favor of the alternative Hypothesis (commonly then that there is significant evidence that a drug has an effect, a physical phenomenon exists etc.)
Usually, the test statistic is associated with a particular distribution which one or two-sided tail probabilities then result in the p-value.

Have you used statistical tests before?

1. Yes
2. No

Testing whether samples are from a Normal distribution

Null Hypothesis: The samples are from a Normal distribution (meaning they are from one Normal distribution)

```
In [1]: from scipy import stats

In [2]: import numpy as np

In [3]: samples_a = stats.norm(0.0, 1.0).rvs(100)

In [4]: samples_b = stats.norm(2.0, 1.0).rvs(100)

In [5]: samples_together = np.concatenate((samples_a, samples_b))

In [6]: statistic, p = stats.normaltest(samples_together)

In [7]: p
Out[7]: 0.10568033831442886

In [8]: samples_a = stats.norm(0.0, 1.0).rvs(100)

In [9]: samples_b = stats.norm(4.0, 5.0).rvs(100)

In [10]: samples_together = np.concatenate((samples_a, samples_b))

In [11]: statistic, p = stats.normaltest(samples_together)

In [12]: p
Out[12]: 5.927638131470592e-09
```

Binomial Test

You flip a coin 100 times. You observe head 57 times. Is that significant enough to tell the coin is biased?

Null Hypothesis: The coin is **not biased**. (Everything is as usual)

```
In [1]: from scipy import stats

In [2]: n_total = 100

In [3]: n_head = 57

In [4]: head_prob_expected = 0.5

In [5]: binomtest_result = stats.binomtest(n_head, n_total, head_prob_expected)

In [6]: p = binomtest_result.pvalue

In [7]: p
Out[7]: 0.19334790449564243
```

Scikit-Learn

What is Machine Learning?

Data + Statistics/Probability Theory + Optimization
The idea of optimizing the adjustable values of a parameterized model to data.

Tasks in Machine Learning

Supervised vs. Unsupervised
Regression vs. Classification
Convex vs. Nonconvex optimization
Closed-Form Solution vs. Iterative approximations

Practical Considerations of Machine Learning

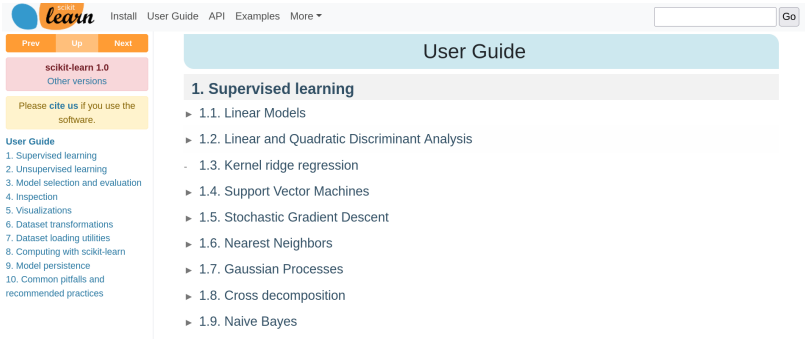
- Know your data, get as much prior knowledge as possible
- Know your task, what is the insight or the model you aspire
- Clean and preprocess the data (this can easily take up to 80% of the time)
- Divide a dataset into train, validate and test
- Prefer simple solutions over complicated ones (Occam's Razor) - A Neural Network is fancy, but often Linear Regression already does the job and is interpretable

Check out the list of common pitfalls compiled by the Scikit-Learn authors:
`https://scikit-learn.org/stable/common_pitfalls.html`

Have you used Machine Learning before?

1. Yes
2. No

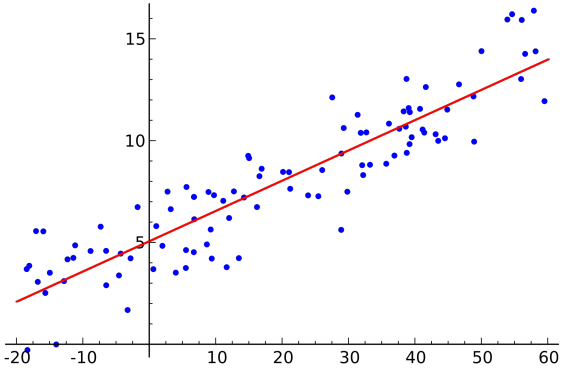
Sklearn has an amazing documentation



Theory on Linear Regression

Linear Regression is

- Supervised
- Regression
- with a closed-form solution of a convex optimization problem



Univariate Linear Regression (with intersection) I

```
# Code source: Jaques Grobler
# License: BSD 3 clause


import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

Univariate Linear Regression (with intersection) II

```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

Univariate Linear Regression (with intersection) III

```
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
```

Univariate Linear Regression (with intersection) IV

```
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

- Diabetes dataset has ten feature dimensions, but this example only considers one

Linear Regression for Nonlinear Problems

Exponential Growth

$$y(t) = y_0 \theta^t \tag{11}$$

with θ being the growth rate, and t being the time passed, usually days, months or years.
That's a nonlinear problem!
Make it linear by applying the logarithm.

$$\log y(t) = \log y_0 + t \log \theta \tag{12}$$

Now we have a linear regression problem for $\log \theta$ and the intercept $\log y_0$

Preprocessing before applying linear regression I

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression

msci_world_monthly = pd.read_csv("https://github.com/Ceyron/machine-learning-a
print(msci_world_monthly)

months_passed = np.array(list(msci_world_monthly.index)).reshape((-1, 1))
y = msci_world_monthly["value"]

y_log = np.log(y)
```

Preprocessing before applying linear regression II

```
lin_regr = LinearRegression()
lin_regr.fit(months_passed, y_log)

# Once fitted, the lin_regr instance contains attributes corresponding to the
# fitted weights and intercept. Only the slope is interest for the exponentia
# growth
log_theta = lin_regr.coef_[0]
theta = np.exp(log_theta)
print(f"Theta: {theta:1.4f}")

# Since we used monthly data, theta is corresponding to the monthly growth, t
# yearly growth is just 12 consecutive applications of theta
theta_yearly = theta**12
print(f"Theta yearly: {theta_yearly:1.4f}")
```

Preprocessing before applying linear regression III

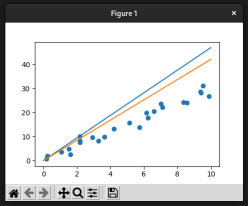
```
# The yearly percentage growth
percentage_growth = (theta_yearly - 1.0) * 100.0
print(f"Yearly growth in percent: {percentage_growth:1.4f} %")

y_pred_log = lin_regr.predict(months_passed)
y_pred = np.exp(y_pred_log)

# Add the prediction to the DataFrame as a series to simply use pandas
# integrated plotting routines (this prettily formats the apparent dates)
msci_world_monthly["value_predicted"] = y_pred
msci_world_monthly.plot.line(x="date", y=["value", "value_predicted"])

plt.title("End of Month Value for MSCI World, truth and predicted")
```

Preprocessing before applying linear regression IV

```
plt.grid()
plt.show()
```

Using Regularization to prevent overfitting

```
In [1]: import numpy as np; import matplotlib.pyplot as plt

In [2]: from sklearn.linear_model import LinearRegression, RidgeCV

In [3]: x = np.random.uniform(low=0.0, high=10.0, size=25)

In [4]: y_clean = 3.0 * x + 2.0 * np.random.randn(25)

In [5]: plt.ion(); plt.scatter(x, y_clean)
Out[5]: <matplotlib.collections.PathCollection at 0x7f93cd6bef40>

In [6]: y_perturbed = y_clean.copy(); y_perturbed[0] += 500

In [7]: normal_regr = LinearRegression(fit_intercept=False)

In [8]: ridge_cv_regr = RidgeCV(alphas=10.0**np.arange(-2, 3, 1), fit_intercept=False)

In [9]: normal_regr.fit(x.reshape(-1, 1), y_perturbed); ridge_cv_regr.fit(x.reshape(-1, 1), y_perturbed)
Out[9]:
RidgeCV(alphas=array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04]),
        fit_intercept=False)

In [10]: x_set = np.linspace(0, 10.0, 100)

In [11]: y_normal_regr = normal_regr.predict(x_set.reshape(-1, 1)); y_ridge_cv_regr = ridge_cv_regr.predict(x_set.reshape(-1, 1))

In [12]: plt.plot(x_set, y_normal_regr); plt.plot(x_set, y_ridge_cv_regr)
Out[12]: [<matplotlib.lines.Line2D at 0x7f93cd6be040>]

In [13]: ridge_cv_regr.alpha_
Out[13]: 100.0
```



General Points on Regularization I

Classical forms of Regularizations are

- L2-Regularization / Ridge - penalty term in the optimization target $\alpha||\vec{\theta}||_2^2$
- L1-Regularization / Lasso - penalty term in the optimization target $\beta||\vec{\theta}||_1$
- L1-L2-Regularization / Elastic Net - penalty term in the optimization target $\alpha||\vec{\theta}||_2^2 + \beta||\vec{\theta}||_1$

Sometimes a grid search over the strength of the Regularization is necessary.
Other forms of regularization

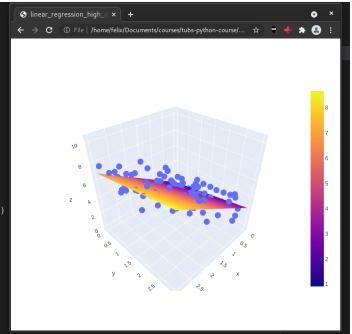
- 'Human-Based Regularization':
  - Using the right model complexity for the problem (favor simple models)
  - Remove unncessary features
  - Data Cleaning (removing outliers, limit ranges of inputs/outputs etc.)
- Physics-based constraints (physics-informed Machine Learning)
- Early-Stopping of training

General Points on Regularization II

- Restarting with different starting points
- Neural-Network related methods:
  - Dropout
  - Batch-Normalization

Linear Regression in higher dimensions

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

In [2]: import plotly.graph_objects as go

In [3]: from sklearn.linear_model import LinearRegression

In [4]: X = np.random.uniform(low=0.0, high=3.0, size=(100, 2))

In [5]: y = 2.0 * X[:, 0] + 0.5 * X[:, 1] + 1.0 + np.random.randn(100)

In [6]: fig = go.Figure(go.Scatter3d(x=X[:, 0], y=X[:, 1], z=y, mode="markers"));

In [7]: regr = LinearRegression()

In [8]: regr.fit(X, y)
Out[8]: LinearRegression()

In [9]: x_0_mesh, x_1_mesh = np.meshgrid(np.linspace(0, 3, 10), np.linspace(0, 3, 10))

In [10]: points_2d = np.vstack((x_0_mesh.flatten(), x_1_mesh.flatten())).T

In [11]: y_pred = regr.predict(points_2d)

In [12]: regr.coef_
Out[12]: array([1.99665886, 0.58927919])

In [13]: regr.intercept_
Out[13]: 0.9162740878228224

In [14]: fig.add_trace(go.Surface(x=np.linspace(0, 3, 10), y=np.linspace(0, 3, 10), z=y_pred.reshape(10, 10)));

In [15]: fig.update_layout(scene_aspectmode="cube");

In [16]: fig.show()
```

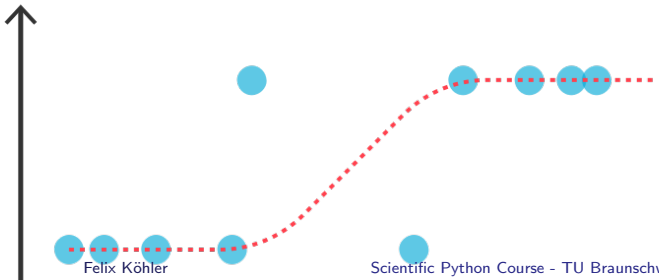


Theory on Logistic Regression

Logistic Regression is

- Supervised
- Classification
- (in most cases) convex optimization
- with NO closed-form solution

Logistic Regression is soft classification - to each prediction there is a probability associated.
Regression, because we model the continuous probability of an event.



Is someone already receiving pension based on his age? I

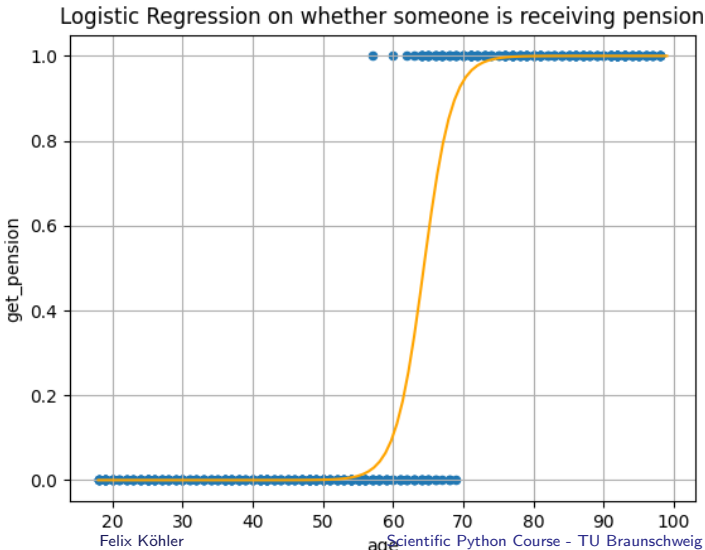
This dataset is artificial and is not accounting for the demographic differences in Germany (the distribution of people of different ages).

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

age_to_getting_pension = pd.read_csv("https://github.com/Ceyron/machine-learni

classifier = LogisticRegression()
classifier.fit(age_to_getting_pension[["age", ]], age_to_getting_pension["get

age_set = np.linspace(18, 99, 100)
prob = classifier.predict_proba(age_set.reshape((-1, 1)))[:, 1]
```

Is someone already receiving pension based on his age? II

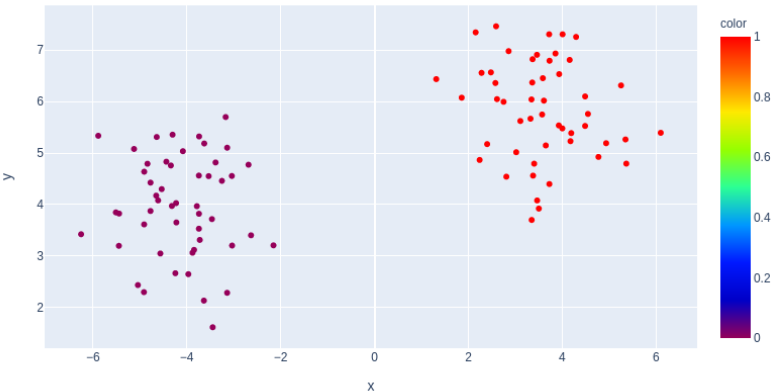
```
age_to_getting_pension.plot.scatter(x="age", y="get_pension")
plt.plot(age_set, prob, color="orange")
plt.grid()
plt.title("Logistic Regression on whether someone is receiving pension")
plt.show()
```

Is someone already receiving pension based on his age? III

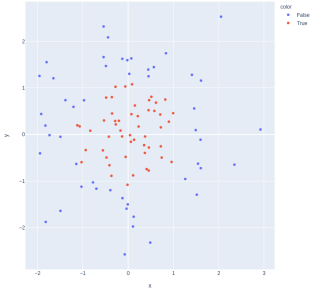


Classification in Higher dimensions I

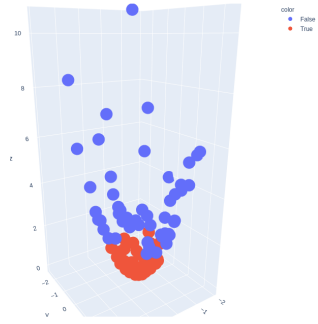
Usually means finding a high-dimensional separation surface.



Classification in Higher dimensions II



$$\vec{\phi}(\vec{x}) = \begin{bmatrix} x_0 \\ x_1 \\ x_0^2 + x_1^2 \end{bmatrix}$$



Using transformations into higher dimensions makes data points linearly separable.
Kernels = Computationally Efficient Way of performing inner products in high-dimensional spaces
Example: Supper Vector Machines

Classification in Higher dimensions III

sklearn.svm.SVC

class sklearn.svm.SVC(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : ***float, default=1.0***

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

Theory on K-Means Clustering

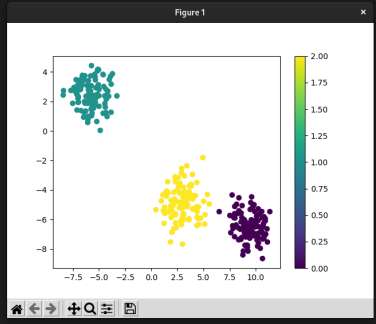
K-Means is

- Unsupervised
- Clustering
- non-convex optimization
- with NO closed-form solution

Clustering assigns a class/label to data points based on their structure. K-Means is a hard clustering, i.e. one data point only belongs to one class at a time and there are no probabilities.

Creating Gaussian Blobs

```
In [1]: from sklearn.datasets import make_blobs

In [2]: X, y = make_blobs(n_samples=300, n_features=2, centers=3)

In [3]: import matplotlib.pyplot as plt

In [4]: plt.scatter(X[:, 0], X[:, 1], c=y)
Out[4]: <matplotlib.collections.PathCollection at 0x7f44451dad00>

In [5]: plt.colorbar()
Out[5]: <matplotlib.colorbar.Colorbar at 0x7f444556a1c0>

In [6]: plt.show()
```



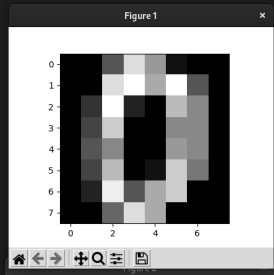
Notice, that we do not use the cluster assignments anymore after the data was created - we are doing **unsupervised Machine Learning**

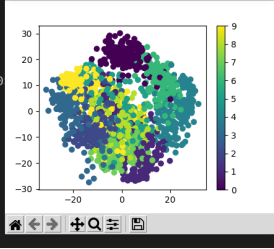
Clustering the artificial Blobs with K-Means

```
In [13]: from sklearn.cluster import KMeans

In [14]: clusterer = KMeans(n_clusters=3)

In [15]: clusterer.fit(X)
Out[15]: KMeans(n_clusters=3)

In [16]: y_pred = clusterer.predict(X)

In [17]: from sklearn.metrics import confusion_matrix

In [18]: cm = confusion_matrix(y, y_pred)

In [19]: cm
Out[19]:
array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```

Dimensionality Reduction on a digit dataset

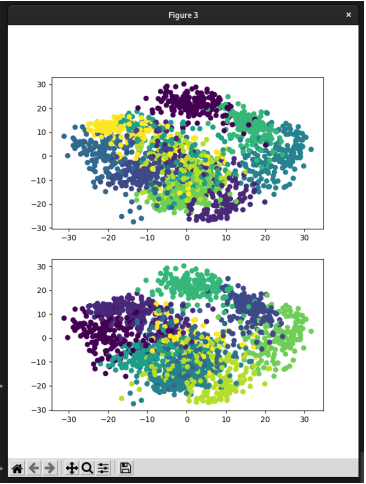
```
In [1]: from sklearn.datasets import load_digits

In [2]: X, y = load_digits(return_X_y=True)

In [3]: import matplotlib.pyplot as plt

In [4]: plt.ion()
Out[4]: <matplotlib.pyplot._IonContext at 0x7fc1d9831f10>

In [5]: plt.figure()
Out[5]: <Figure size 640x480 with 0 Axes>

In [6]: plt.imshow(X[0, :].reshape(8, 8), cmap="gray")
Out[6]: <matplotlib.image.AxesImage at 0x7fc1ce50bd00>

In [7]: from sklearn.decomposition import PCA

In [8]: X_trafo = PCA(n_components=2).fit_transform(X)

In [9]: plt.figure()
Out[9]: <Figure size 640x480 with 0 Axes>

In [10]: plt.scatter(X_trafo[:, 0], X_trafo[:, 1], c=y)
Out[10]: <matplotlib.collections.PathCollection at 0x7fc1cc13f670

In [11]: plt.colorbar()
Out[11]: <matplotlib.colorbar.Colorbar at 0x7fc1cc0baee0>

In [12]:
```





K-Means on dimensionality reduced datasets

```
In [23]: from sklearn.cluster import KMeans

In [24]: clusterer = KMeans(n_clusters=10)

In [25]: y_pred = clusterer.fit_predict(X)

In [26]: from sklearn.metrics import confusion_matrix

In [27]: confusion_matrix(y, y_pred)
Out[27]:
array([[  0,   0,   0,   0,   0,   0, 177,   1,   0,   0],
       [  1,   0,   2,   1,   0,  24,   0,   0,  99,  55],
       [ 13,   2,   0,   0,   4, 147,   1,   0,   8,   2],
       [155,  12,   0,   2,   7,   0,   0,   0,   7,   0],
       [  0,   0,   0,   0,  11,   0,   0, 163,   2,   5],
       [  1,  42,   1, 136,   0,   0,   0,   2,   0,   0],
       [  0,   0, 177,   0,   0,   0,   1,   0,   2,   1],
       [  0,   0,   0,   0, 175,   0,   0,   0,   2,   2],
       [  2,  52,   2,   4,   5,   3,   0,   0, 100,   6],
       [  6, 139,   0,   6,   8,   0,   0,   0,   1,  20]])

In [28]: import matplotlib.pyplot as plt

In [29]: plt.subplot(211)
Out[29]: <AxesSubplot:>

In [30]: plt.scatter(X_trafo[:, 0], X_trafo[:, 1], c=y)
Out[30]: <matplotlib.collections.PathCollection at 0x7fc1c8aa8100>

In [31]: plt.subplot(212)
Out[31]: <AxesSubplot:>

In [32]: plt.scatter(X_trafo[:, 0], X_trafo[:, 1], c=y_pred)
Out[32]: <matplotlib.collections.PathCollection at 0x7fc1c179b1f0>
```

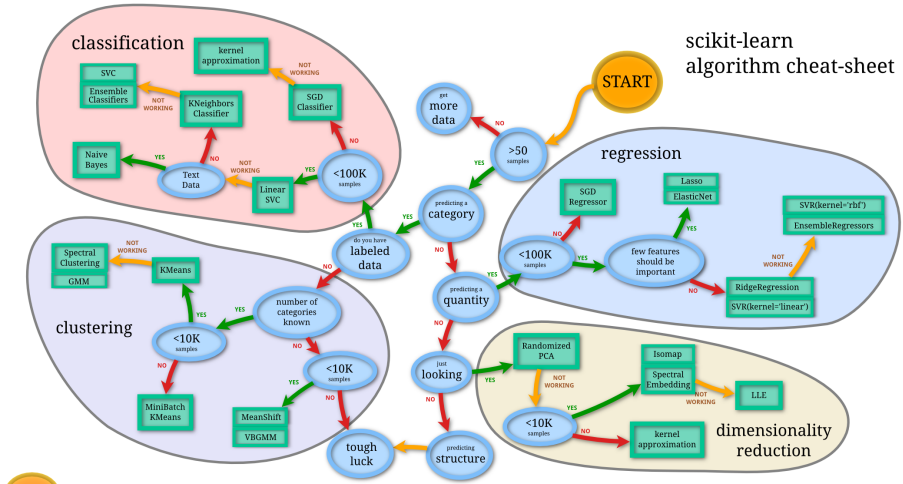


Effect of restarting K-Means

Shows that the optimization target of K-Means is non-convex

```
In [37]: clusterer = KMeans(n_clusters=10, n_init=1, random_state=21)

In [38]: y_pred = clusterer.fit_predict(X)

In [39]: confusion_matrix(y, y_pred)
Out[39]:
array([[  0,   0,   0,   0,   0, 176,   0,   2,   0,   0],
       [  2,   1,  54,  24,   1,   0,   0,   0, 100,   0],
       [  0,  15,   3, 148,   0,   1,   2,   0,   8,   0],
       [  0, 158,   0,   1,   2,   0,   6,   0,   7,   9],
       [  0,   0,   2,   0,   0,   0,  11, 166,   2,   0],
       [  2,   2,   0,   0, 125,   0,   0,   2,   0,  51],
       [177,   0,   0,   0,   0,   1,   0,   0,   3,   0],
       [  0,   0,  10,   0,   0,   0, 167,   0,   2,   0],
       [  2,   2,  10,   3,   4,   0,   4,   0, 101,  48],
       [  0,   9,  21,   0,   4,   0,   8,   0,   1, 137]])

In [40]: clusterer = KMeans(n_clusters=10, n_init=1, random_state=42)

In [41]: y_pred = clusterer.fit_predict(X)

In [42]: confusion_matrix(y, y_pred)
Out[42]:
array([[  0,   0,   0,   1,   0,   0, 177,   0,   0,   0],
       [128,   0,   0,   0,   0,  27,   0,   1,   0,  26],
       [ 10,   0,   8,   0, 148,   2,   1,   0,   3,   5],
       [  7,   0, 159,   0,   0,   0,   0,   2,   7,   8],
       [  2,   0,   0, 169,   0,   0,   0,   0,  10,   0],
       [  0,   2,   1,   2,   0,   0,   0, 137,   0,  40],
```

Choosing the right algorithm



The Design Philosophy of Scikit-Learn

- All algorithms are outfitted with sensible default values for the hyper-parameters
- If hyper-parameters greatly depend on the size of the dataset, a clever heuristic titelled 'auto' is available
- Hyper-Parameters of an Algorithm are set during instantiation of the object
- Using an algorithm usually follows the pattern: Instantiation, Fitting and Predicting/Transforming (i.e. defining hyper-parameters, training and using)
- The main type of algorithm objects are either predictors or data transformers

Further interesting techniques

- Auto-Machine Learning searches over different algorithms/regularizations/pipelines to find the perfect fit for the dataset
- Gradient-Boosting
- (More probabilistic Machine Learning)

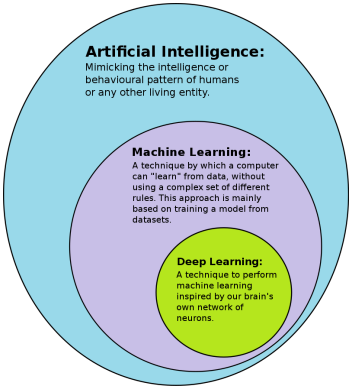
What we did not cover

- Parallelization using Joblib and Dask
- Dimensionality Reduction by Manifold Learning (=Nonlinear dimensionality reduction)
- Data Preprocessing
- Model Selection
- Model Compositing and Pipelines
- Many of the great algorithms implemented, for example
  - Support Vector Machines (SVM), for regression and classification
  - Decision Trees and Random Forests
  - Spectral Clustering

TensorFlow Keras

What makes learning deep?

- Many tunable parameters
- (Potentially) highly nested computations
- (Potentially) a complicated training process
- Most of the time: The Lack of interpretability, but remarkable performance



The deep learning revolution

- Huge, openly available datasets
- Increasing available computing power, especially due to accelerators (like GPUs)
- Incredible breakthroughs with high publicity
  - Immense research funding by big tech companies
  - Image Segmentation (AlexNet, ResNet)
  - Natural Language Processing (GPT3)
  - Computational Biology (Alphafold)
  - Scientific Machine Learning (Physics-Informed Neural Networks)

⋮

-

Ingredients of Deep Learning

- A computational graph (=Neural Network architecture)
- A loss function
- (Revers-Mode) Automatic Differentiation
- Gradient Descent based optimization
- Highly parallelized and accelerated computation

Amazing animation

https://www.3blue1brown.com/lessons/neural-networks

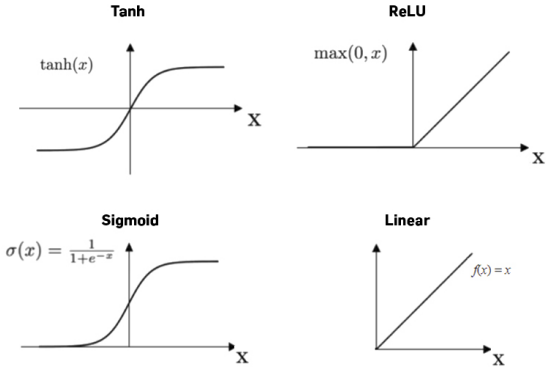
Deep Learning Landscape (in Python)

- Theano
- **TensorFlow**
- PyTorch
- (Julia Flux & Zygote)

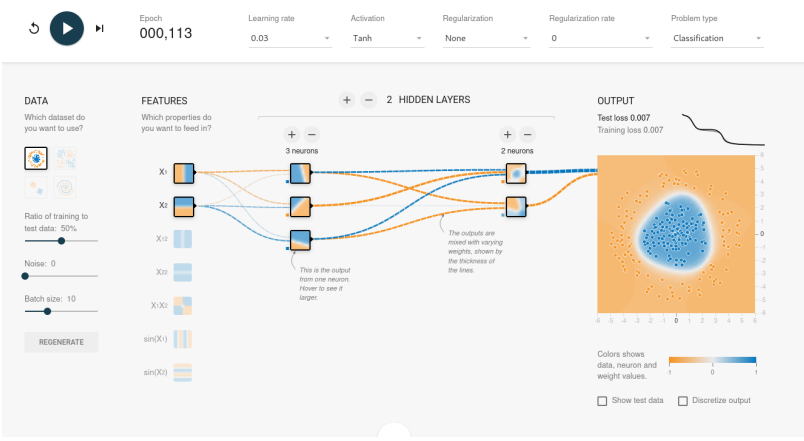
Simple Neural Networks (Multilayer Perceptrons/Fully Connected NN) can also be implemented by oneself - but could lack some of the features these frameworks provide. For any fancier architecture, please always resort to one of these **free** frameworks.

Computational Graphs & Neural Architectures I

Classical Feed-forward neural networks are a combination of linear affine mappings with a subsequent non-linear activation function



Computational Graphs & Neural Architectures II



Computational Graphs & Neural Architectures III

The universal approximation theorem
Under some loose assumptions, Neural Networks (with only one hidden layer) can approximate any function.
And functions are what we are interested in:

- Image Segmentation: Mapping from images to one of ten digit classes
- Control Theory: Mapping from senor measurements to control signals
- Natural Language Processing: Mapping from a document to a summary of it

Computational Graphs & Neural Architectures IV

$$
\begin{aligned}
n_1(x) &= Relu(-5x - 7.7) \\
n_2(x) &= Relu(-1.2x - 1.3) \\
n_3(x) &= Relu(1.2x + 1) \\
n_4(x) &= Relu(1.2x - .2) \\
n_5(x) &= Relu(2x - 1.1) \\
n_6(x) &= Relu(5x - 5)
\end{aligned}
$$

$$
\begin{aligned}
Z(x) = &-n_1(x) - n_2(x) - n_3(x) \\
&+ n_4(x) + n_5(x) + n_6(x)
\end{aligned}
$$

Functions (also Mathematically) are a huge concept, but essentially this is what we are doing here.

Loss Functions

Measuring how good our solution is.

The most common metrics are

- For Regression: Least Squares loss

$$\mathcal{L}(\theta) = \sum_i \left(f(\vec{x}_i, \theta) - y_i\right)^2 \tag{13}$$

- For Classification: (Sparse) Categorical Cross Entropy

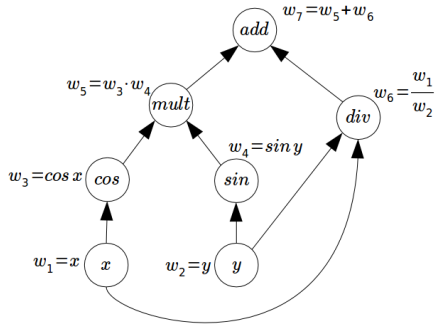
$$\mathcal{L}(\vec{\theta}) = \sum_i - \sum_d y_d \log f(\vec{x}_i, \vec{\theta})_d \tag{14}$$

The Classification loss works, because Neural Networks for Classification are Soft Classifiers - meaning that they assign probabilities to each class, instead of just selecting a class. A subsequent argmax can be used to extract the winning class.

Side Note: They stem from a probabilistic perspective on Learning and arise from the Maximum Likelihood Estimate for the assumptions of underlying distributions. They are not arbitrary!

Automatic Differentiation I

Obtaining gradients to high precision at identical cost to evaluating a function.
This is not Finite Differencing or Symbolic Differentiation. It is a clever application of the chain rule of differentiation.



The computational graph of $f(x, y) = \cos(x)\sin(y) + \frac{x}{y}$

Automatic Differentiation II

$$\nabla f(\vec{x}) = \begin{bmatrix} -\sin(x_1)\sin(x_2) + \frac{1}{x_2} \\ \cos(x_1)\cos(x_2) - \frac{x_1}{x_2^2} \end{bmatrix} \tag{15}$$

```
In [24]: import tensorflow as tf

In [25]: def crazy_function(x):
    ...:     return tf.cos(x[0]) * tf.sin(x[1]) + x[0]/x[1]
    ...:

In [26]: def crazy_function_gradient(x):
    ...:     return tf.convert_to_tensor([-tf.sin(x[0])*tf.sin(x[1]) + 1/x[1], tf.cos(x[0])*tf.cos(x[1]) - x[0]/x[1]**2])
    ...:

In [27]: position = tf.constant([2.0, 3.0])

In [28]: crazy_function(position)
Out[28]: <tf.Tensor: shape=(), dtype=float32, numpy=0.60794>

In [29]: crazy_function_gradient(position)
Out[29]: <tf.Tensor: shape=(2,), dtype=float32, numpy=array([0.20501329, 0.18976004], dtype=float32)>

In [30]: with tf.GradientTape() as tape:
    ...:     tape.watch(position)
    ...:     crazy_function_evaluated = crazy_function(position)
    ...:

In [31]: tape.gradient(crazy_function_evaluated, position)
Out[31]: <tf.Tensor: shape=(2,), dtype=float32, numpy=array([0.20501329, 0.18976004], dtype=float32)>
```

What is the coolest way of obtaining (exact) gradients of highly complicated functions?

1. Deriving the function by hand: I am a Math Sorcerer
2. Using Computer Algebra System: I want to see a closed-form solution, but do not like the pain
3. Finite Differencing: One tiny step ahead and take the difference
4. Automatic Differentiation: I like the chain rule

Gradient-Based Optimization (=Gradient Descent) I

$$\vec{\theta}^{[k+1]} = \vec{\theta}^{[k]} - \alpha \nabla_{\vec{\theta}} f(\vec{x}, \vec{\theta}) \tag{16}$$

with α being the learning rate. (It's selection and change over the iterations is a science for itself.)

```
In [1]: import tensorflow as tf

In [2]: f = lambda x: x[0]**2 + x[1]**2

In [3]: def f_grad(x):
   ...:     with tf.GradientTape() as tape:
   ...:         tape.watch(x)
   ...:         f_eval = f(x)
   ...:     return tape.gradient(f_eval, x)

In [4]: x = tf.convert_to_tensor([1.5, 1.5])
2021-09-30 10:56:34.489987: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep N
eural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

In [5]: for i in range(10):
   ...:     x = x - 0.1 * f_grad(x)
   ...:     print(x)
   ...:
tf.Tensor([1.2 1.2], shape=(2,), dtype=float32)
tf.Tensor([0.96000004 0.96000004], shape=(2,), dtype=float32)
tf.Tensor([0.768 0.768], shape=(2,), dtype=float32)
tf.Tensor([0.6144 0.6144], shape=(2,), dtype=float32)
tf.Tensor([0.49152002 0.49152002], shape=(2,), dtype=float32)
tf.Tensor([0.393216 0.393216], shape=(2,), dtype=float32)
tf.Tensor([0.3145728 0.3145728], shape=(2,), dtype=float32)
tf.Tensor([0.25165826 0.25165826], shape=(2,), dtype=float32)
tf.Tensor([0.20132661 0.20132661], shape=(2,), dtype=float32)
tf.Tensor([0.16106129 0.16106129], shape=(2,), dtype=float32)
```

Gradient-Based Optimization (=Gradient Descent) II

Be aware, this example is convex. Optimizing the parameters of a Neural Net is non-convex! Nowerdays, training Neural Networks by Gradient descent differs:

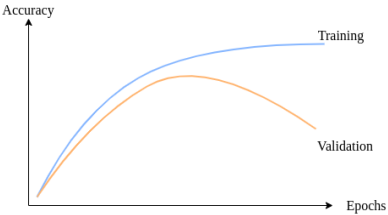
- Usage of *ADAM* instead of classical gradient descent: uses additional moment to overcome bad local minima
- Adaptive Learning rates
- Mini-Batching instead of working on the full dataset all at once.

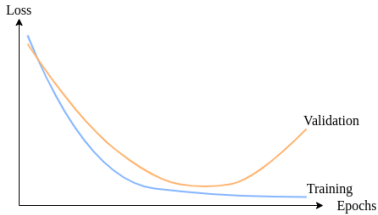
Optimizing a proper loss

Here: Not just look at minimizing a function, but a function depending on both an input and parameters.

Judging convergence

- Since the optimization is non-convex we could reach local minima
- Oscillations are common due to stochastic optimization
- Common practice: The model is well fitted when train and validation loss start diverging





Batching and the batch shape I

Applying operations in parallel is the key to making Neural Network training feasible.
If you have a vector of length 50, then its **data shape** is `(10, )`.
Now you have 50 of these vectors, then they can be combined in a matrix of shape `(50, 10)` where the zeroth axis is the **batch shape** and the first axis is the **data shape**.
Imagine you now have an image with 28×28 pixels each having a corresponding RBG value. Then one image is a 3D tensor of shape `(28, 28, 3)`. This is the **data shape** consisting of the resolution of the image and its color depth. If you now have 100 images, the total shape of the dataset is `(100, 28, 28, 3)` (a 4D tensor).
TensorFlow assumes that there is always a batch shape, even if it is just a proxy dimension. Therefore,

- if you have one sample with 100 feature dimensions, the dataset has the shape `(1, 100)`
- if you have 100 samples with only one feature dimension each, the dataset has the shape `(100, 1)`

Batching and the batch shape II

Just having a dataset of shape `(100, )` is arbitrary!
Note: The order for instance for images of batch - pixels - channel depth is valid for TensorFlow, it might differ for other Deep Learning Frameworks.

Batching and the batch shape III

Applying TensorFlow operations to datasets with a leading batch shape, will apply the operation to each element individually and in the best case with high parallelization.

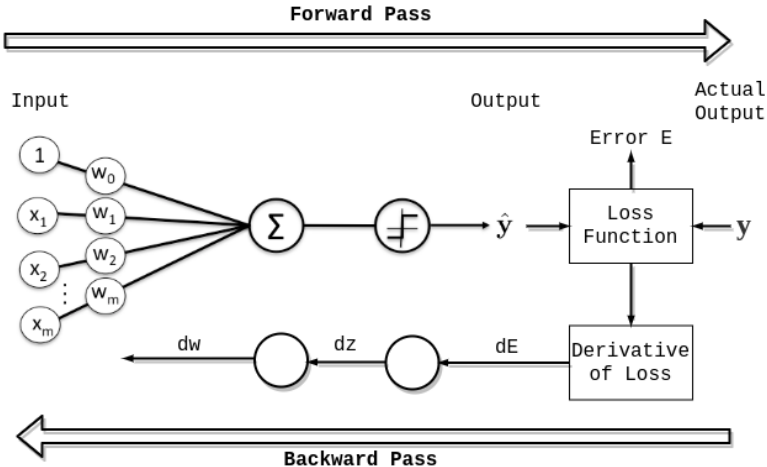
What is the shape of the dataset containing 420 Cauchy stress tensors?

1. `(420,)`
2. `(420, 3)`
3. `(420, 3, 3)`
4. `(420, 6)`

Mini-Batching and the Epoch

Use the size of the Minibatch as high as $80 - 90\%$ of the RAM (or GPU RAM) that you are dedicating towards Python session.

Neural Networks in Summary



What is **not** a main step in one epoch of training a Neural Network?

1. update parameters
2. sampling data points
3. forward pass
4. initialize the parameters
5. obtain gradients (backward pass)
6. evaluate loss

Let's learn the sin function I

```
import tensorflow as tf
import matplotlib.pyplot as plt

# For reproducibility
tf.random.set_seed(42)

x = tf.random.uniform((50,), minval=0.0, maxval=2*3.141)
y = tf.sin(x) + 0.2 * tf.random.normal((50, ))
x = tf.reshape(x, (-1, 1))
y = tf.reshape(y, (-1, 1))

plt.scatter(x.numpy(), y.numpy(), color="black")

model = tf.keras.Sequential([
```

Let's learn the sin function II

```
    # Play around with the model complexity
    tf.keras.layers.Dense(
        1000,
        activation="relu",
        kernel_regularizer="l2"
    ),
    tf.keras.layers.Dense(
        1000,
        activation="relu",
        kernel_regularizer="l2"
    ),
    tf.keras.layers.Dense(1, activation=None)
])
```

Let's learn the sin function III

```
model.compile(
    optimizer="adam",
    loss=tf.keras.losses.MeanSquaredError(),
)

x_set = tf.linspace(0.0, 2.0*3.141, 100)
x_set = tf.reshape(x_set, (-1, 1))
y_pred = model(x_set)
plt.plot(x_set, y_pred)


for i in range(10):
    model.fit(x, y, epochs=50, validation_split=0.2)
    y_pred = model(x_set)
```

Let's learn the sin function IV

```
    plt.plot(x_set, y_pred)

plt.show()
```

Scikit-Learn can also learn the Sine Function I

```
from scipy.sparse.construct import rand, random
from sklearn.neural_network import MLPRegressor
import numpy as np
import matplotlib.pyplot as plt

# For reproducibility
np.random.seed(42)

x = np.random.uniform(size=(50,), low=0.0, high=2*3.141)
y = np.sin(x) + 0.2 * np.random.normal(size=(50, ))
x = np.reshape(x, (-1, 1))
# y = np.reshape(y, (-1, 1))

plt.scatter(x.flatten(), y, color="black")
```

Scikit-Learn can also learn the Sine Function II

```
model = MLPRegressor(max_iter=1, random_state=42)
model.fit(x, y)

x_set = np.linspace(0.0, 2.0*3.141, 100)
x_set = np.reshape(x_set, (-1, 1))
y_pred = model.predict(x_set)
plt.plot(x_set, y_pred)


for i in range(10):
    model = MLPRegressor(hidden_layer_sizes=(1000, ), max_iter=100*(i+1), ran
    model.fit(x, y)
    y_pred = model.predict(x_set)
```

Scikit-Learn can also learn the Sine Function III

```
    plt.plot(x_set, y_pred)


plt.show()
```

The disadvantages of sklearn:

- Limited to feedforward MLP architectures only
- Training only on the CPU

But on the other side its even easier than with TensorFlow Keras.

Which Neural Network Interface do you like more?

1. Yes: TensorFlow Keras
2. No: Scikit-Learn

An easy network for the MNIST dataset I

```
import tensorflow as tf
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

# plt.imshow(x_train[0], cmap="gray")
# plt.show()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10),
```

An easy network for the MNIST dataset II

```
    tf.keras.layers.Softmax()
])

# print(tf.keras.losses.SparseCategoricalCrossentropy()(y_train[:1], model(x_

model.compile(optimizer="adam", loss=tf.keras.losses.SparseCategoricalCrossent
model.fit(x_train, y_train, epochs=5)

print(model(x_test[:5]))
print(tf.argmax(model(x_test[:5]), axis=1))
print(y_test[:5])
```

Google Colab in a nutshell

colab.research.google.com

```
[1] import tensorflow as tf
    import matplotlib.pyplot as plt

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
    x_train = x_train / 255.0
    x_test = x_test / 255.0

    # plt.imshow(x_train[0], cmap="gray")
    # plt.show()

    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dense(10),
        tf.keras.layers.Softmax()
    ])

    # print(tf.keras.losses.SparseCategoricalCrossentropy()(y_train[:1], model(x_train[:1])))

    model.compile(optimizer="adam", loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=["accuracy"])
    with tf.device("gpu"):
      model.fit(x_train, y_train, epochs=5)

    print(model(x_test[:5]))
    print(tf.argmax(model(x_test[:5]), axis=1))
    print(y_test[:5])

    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
    11493376/11490434 [==============================] - 0s 0us/step
    11501568/11490434 [==============================] - 0s 0us/step
    Epoch 1/5
```

Other Neural Architectures

- Convolutional Neural Networks
- Recurrent Neural Networks
- Residual Neural Networks
- Graph Neural Networks
- Neural ODEs
- ...

Take a look at geometric deep learning `https://youtu.be/w6Pw4MOzMuo`

Outlook

Python Libraries worth learning

- SymPy: A Computer-Algebra System
- Streamlit: Turn your scripts into web-applications in 5 minutes
- Jax: The future of Deep Learning in Python (by Google)
- OpenCV: Image Analysis
- Dask: Job based High-Performance computing with plugins to common schedulers like SLURM
- (Flask: Writing simple Web APIs and Web servers - out of the scope for this course)

What to do next?

- Fill out the feedback form from the Pad
- Attend tomorrow for basics in research software engineering
- Learn Numerical Linear Algebra in Python, the top-down approach:
  `https://github.com/fastai/numerical-linear-algebra`
- Read the amazing user guide of scikit-learn and do their tutorials
- Work through the TensorFlow Keras examples:
  `https://www.tensorflow.org/tutorials`