Data Analysis and Visualization for Chemists and Material Scientists

A Practical Guide with Python

Table of contents

1 Welcome to this Course!

This course is designed for better understanding how Python can be used for data analysis and visualization in the fields of chemistry and materials science. Before diving into the course content, please take a moment to review the following important informations.

Disclaimer

This website is under construction and continuous development. The content provided is for educational purposes only and may be subject to change. The authors and contributors are not responsible for any errors or omissions, or for any outcomes related to the use of the information contained herein. Please read the license, privacy policy, and disclaimers of this website.

Prequisits

- Basic knowledge of Python is useful but not required.
- Fundamental understanding of statistics is mandatory.
- Basic knowledge of chemistry and materials science is recommended.
- You do not need to download or install any software to complete this course. All exercises can be done in the cloud using Google Colab.
- But if you want to run the notebooks on your local machine, there is a Installation Guide available.

1.1 Who is the course for?

- For everyone who is interested in data analysis and visualization in the field of chemistry and materials science.

What can you expect from this course

- This course wants to show you the advantage of using a programming language for data analysis and visualization in comparison to GUI-based softwares.
- The course will give you a very brief introduction to Python and its libraries.
- The main focus will be on the libraries `numpy`, `pandas`, `scipy`, `scikit-learn`, `matplotlib`.
- The course is organized interactivly. You will get the chance to practice with exercises.
- Upon successful completion of this course, you will have acquired a comprehensive understanding of the fundamental components of Python and the key packages necessary for the analysis and presentation of your own research data.
- At the end of this course you will be able to find enough resources to dive deeper into Python and you can test your knowledge by an example exam.

What can you NOT expect from this course

- You will **not get a deep explanation of the Python language**. Please consider **full Python tutorials** to get a deeper overview of Python.
- You will not learn object-oriented programming or functional programming in Python.
- This is **not a statistics course**.

How this course is structured

The plan is to create different levels of content, so that you can choose the level of difficulty that suits you best. The course is designed to be flexible and adaptable to your needs, allowing you to focus on the areas that are most relevant to your work or interests.

! Important

Due to the fact that this course is **still under construction**, the content may change over time. The current version of the course is a work in progress and may not reflect the final structure or content.
If someone is interested in contributing to this course, please feel free to contribute. The course is hosted via GitHub. Contributions, suggestions, and feedback are highly appreciated. Please refer in the README to the Contribution Guidelines for more

details.

This course is divided into three possible paths to accomplish the learning objectives:

- **Beginner Path**: Focuses on first contact with Python, covering basic concepts and introductory modules.
- **Advanced Path**: Provides a deeper dive into different libraries and specialized visualization plots. [work in progress]
- **Challenging Path**: Explores advanced topics and complex data visualization techniques, with more difficult exercises and in-depth analysis. [TODO]

The different difficulty levels are marked with the following icons:

- Introduction Parts:
- Beginner Path:
- Advanced Path:
- Challenging Path:

The icons are located under the title of each lecture. The more stars you see, the more difficult the content is.

Each lecture includes examples related to chemistry and material science. At the end of each part, there are exercises to test your understanding and reinforce the concepts learned. These exercises are designed to be practical and relevant to real-world scenarios in chemistry and materials science.

Comprehensive Exam [TODO] At the very end of the course, there is a comprehensive exam which covers a complete data analysis and visualization example.

❗ Important

It is recommended to try out every lecture and exercise to get the most out of this course. If the lecture is created as interactive notebook, you can run the notebook in the cloud using Google Colab via this icon or download it and run it on your local machine via this icon . If any data files are used you can download them via this icon and .

Part I

Essentials

2 Introduction

Difficulty level:

What do you need for data analysis and visualization?

- Research normally contains handling a lot of data.
- Clear presentation of data is essential to the understanding of data.
- It helps to improve your scientific communication and make your results more accessible to others.
- There are many tools available for data analysis and visualization.

A very simple approach would be the use of basic spreadsheet programs with graphical user interfaces *e.g.* Excel, LibreOffice Calc …

But for more advanced analysis you need probably specific plotting and analysis tools:

- You can either use programs with GUIs *e.g.* LabPlot, QtiPlot, Scilab, SciDAVis, Origin …
- or a GUI based program with command line interface *e.g.* Xmgrace, GNU Octave …
- or command line based tools *e.g.* Gnuplot, Matlab, Mathematica …
- or programming languages *e.g.* Python, R, Julia …

This is a small election of tools, that can be used for data analysis and plotting.

For a larger list see these Wikipedia lists:

- List of Numerical Analysis Software
- List of Graphical Software
- List of Statistical Software
- List of Computer Algebra Systems

Why you should learn Python?

- open-source
- well documented with a lot of online resources:
  - tutorials,
  - documentations,
  - stackoverflow ect.
- easy to learn
- intuitive
- highly flexible
- wide range of applications
- huge number of different python packages are available
- very easy to adapt to other programming languages such as `R` and `Julia`
- it is not limited to a specific operating system
- you can automate repetitive tasks
- you can create templates for your analysis and visualization tasks
- one of the most popular languages in machine learning and data science

3 Python?

Difficulty level:

Short History of Python

(see Wikipedia for more details)

- 1989: the beginning of python
- 1991: Guido van Rossum, a Dutch programmer and father of Python, implemented and published the first version of Python.
- Fun fact: The name Python came from the show Monty Python's Flying Circus.
- 1994: Python 1.0 was released
- 2000: Python 2.0 was released
- 2008: Python 3.0 was released with new syntax and features
- Important: **python3 to python2 is backward incompatible** *e.g.* `print("Hello World")` python3 and `print "Hello World"` python2.
- `python2` is nowadays outdated. It is not recommended using it for new projects.

Van Rossum designed Python as a "Computer Programming Language for Everybody".

What is Python?

- Python is an **interpreted** language compared to compiled languages *e.g.*, C or C++.
- Python is **often slower** compared to compiled languages.
- Python has **dynamic type checking** and **garbage collection**.
- Python supports both **object-oriented** and **functional programming**.
- Python is a **high-level language**, meaning it is closer to human language than the machine language understood by computers.

Simplified Scheme how Python works internally:

If you execute a python program, the python interpreter will convert the code into byte code and then the byte code will be executed by the python virtual machine (PVM).

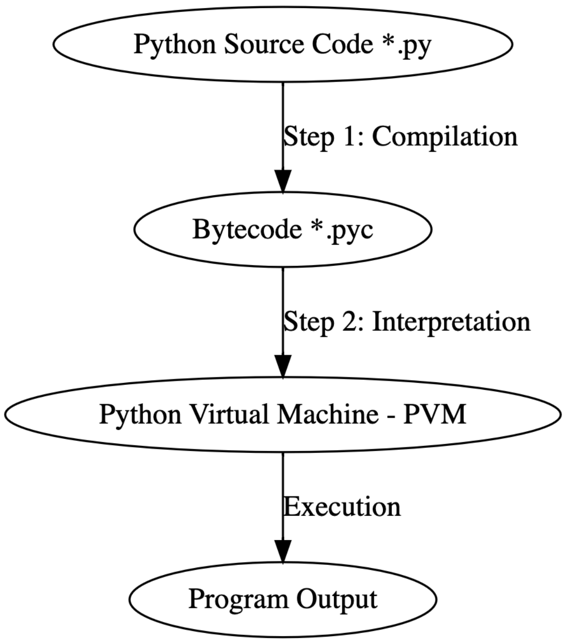


Figure 3.1: Interpreter Scheme - Simplified

How I can learn Python?

There are a lot of resources to dive deeper into Python *e.g.*:

- https://www.py4e.com/ (Python for Everybody)
- https://py-tutorial-de.readthedocs.io/de/python-3.3/ (German Tutorial)
- https://www.w3schools.com/python/default.asp
- https://jakevdp.github.io/PythonDataScienceHandbook/
- https://exercism.org/ (Coding Exercises)
- https://www.freecodecamp.org/ (Coding Exercises)
- https://realpython.com/tutorials/data-viz/
- https://www.youtube.com/watch?v=LHBE6Q9XlzI

Use AI to learn Python

❗ Important

- Language Models like GPT-3 can help you to learn Python.
- Today it is not always necessary to learn the syntax of a programming languages by heart.
- You can use AI tools to help you with the syntax and concentrate on the problem-solving part.
- **Learn how to prompt** your problem to the AI and get a proper solution.
- But **be aware** that the AI is **not perfect** and can give you **wrong solutions**.
- So it is **important** that you can **understand the code** and can debug it.
- AI is only a tool not a replacement for a human.

Practice, Practice, Practice

❗ Important

As it is with languages, you can only learn it if you **practice** it. So, start you own projects and have fun with it!

4 Installation Guide

Difficulty level:

Here is a short instruction how to install Python on your local PC or you can use Google Colab to solve all the exercises of this course.

The use of Google Colab needs a Google Account. Please read the Terms of Service and Privacy Policy

How to install Python locally?

Fundamental Python websites:

- Python
- Python documentation

1. Install Python Interpreter (you do not needed if you use an environment manager)

Your preferred searching engine is your friend to find the best way to install Python on your system. Please choose that method which is suitable for you.

Python can be installed by several ways:

- directly by Python official site
  - the installation guide can be found under python wiki
- or via package manager of your os:
  - *e.g.:* `sudo apt install python` (linux debian) or `brew install python` (macOS)
- or via docker, wsl ect.
- or via a conda or mamba python package and environment managers which have a python interpreter on board and are available for Windows, Linux and macOS [my recommendation]

2. Python Package and Environment manager

The advantage of using a python package and environment manager is that you have a python interpreter directly on board, but you can also directly create different python enviroments and install and remove python packages.

Conda

There are differerent `conda` installer: (Please pay attention which one is suitable for you (https://docs.anaconda.com/distro-or-miniconda/).

> ⚠ Warning
>
> Please read the Anaconda Terms of Service FAQs and Terms of Service) **not** every case is **free** of charge of use.

- Anaconda Distribution is a comprehensive distribution which includes conda and hundreds of preinstalled packages and tools.
- miniconda is the light version of it which contains only conda, python interpreter and few fundamental packages
- miniforge minimal installer for conda and using only the community conda-forge channel

Mamba

Another python package and environment manager is mamba.

`mamba` is a reimplementation of `conda`:

- micromamba is a statically linked version of `mamba`
- `mamba`and it is currently faster than `conda`

> Tip
>
> **Recommendation:** ***micromamba***
> Install it like it is explained under the micromamba documentation: - https://mamba.readthedocs.io/en/latest/installation/micromamba-installation.html

Please install in one of the above explained ways Python and use your preferred searching engine to get more information.

3. Set up an environment

It is often very useful to have different python environments for different python projects because of the need of different python package versions.

You can use `conda` or `micromamba` to create different environments. There exists also other virtual environment manager.

In this course the explanation is restricted to `micromamba` as an example. If you want to use something else there exists tons of information online how to use other programs.

Micromamba: Most important commands are:

Read for more detail: https://mamba.readthedocs.io/en/latest/user_guide/micromamba.html

Creating a new virtual environment:

```
micromamba create --name <myenvname>
```

Install new packages:

```
micromamba install <packagename>
```

List all environments:

```
micromamba env list
```

Activate an environment:

```
micromamba activate <myenvname>
```

List all packages of this environment:

```
micromamba list
```

4. Usefull packages for Data Analysis and Visualization:

- pip - package installer for python instead of conda
- jupyter-notebook/jupyterlab - interactive computing environment
- ipykernel - IPython Kernel for Jupyter
- matplotlib - data visualization library
- numpy - numerical library
- scipy - scientific library
- pandas - data manipulation library
- seaborn - data visualization library
- scikit-learn - machine learning library
- statsmodels - statistical library

special packages for data visualization:

- bokeh - interactive data visualization library
- plotly - interactive data visualization library
- networkx - network analysis library
- python-ternary - ternary plot library
- altair - declarative statistical visualization library
- umap-learn - dimensionality reduction library

special packages for chemistry and material science:

- ase - atomic simulation environment
- pymatgen - Python Materials Genomics
- RdKit - cheminformatics library
- openbabel - cheminformatics library
- pymol-opensource - molecular visualization library

For more chemistry and material science packages please check the Awesome Python Chemistry repository.

Short Cut

Tip

Recommendation Use *yml-file* with all needed packages and configurations:

Save your needed packages in an `environment.yml` file *e.g.*:

```
name: myenv
channels:
 - conda-forge
dependencies:
 - python=3.12
 - pip
 - ipykernel
 - jupyterlab
 - pandas
 - numpy
 - matplotlib
 - scikit-learn
 - scipy
 - seaborn
 - statsmodels
```

and create an environment with this specific packages:

```
micromamba env create -f environment.yml
```

Then you can activate it via:

```
micromamba activate myenv
```

5. Test your installation

Test your installation by opening the interactive python mode by typing in your terminal (Linux, macOS) / command prompt (Windows):

```
python
```

then something like this should be opened in your terminal (Linux, macOS) / command prompt (Windows)

```
Python 3.12.7 | packaged by conda-forge | (main, Oct  4 2024, 15:57:01) [Clang 17.0.6 ] on da
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

then type:

```
print("Hello World!")
```

If this works your installation was successful!

5 Editors

Difficulty level:

Choose an editor

After you installed Python successfully you need an editor for writing your Python programs. A python script is a text file with the ending `.py`. Technicallly you could use everything where you can write text, but it is not really purposeful.

An editor with **syntax highlighting**, **code completion** and **debugging** is very useful.

Editors:

- Spyder https://www.spyder-ide.org/
- PyCharm (JetBrains) https://www.jetbrains.com/products/compare/?product=pycharm&product=pycharm-ce
- Jupyter Notebook https://jupyter.org/ https://code.visualstudio.com/docs/datascience/jupyter-notebooks
- Jupyter Lab https://jupyter.org/ https://code.visualstudio.com/docs/datascience/jupyter-notebooks
- Google Google Colab https://colab.research.google.com/ etc.

Tip

Recommendation: *Visual Studio Code*

Python extension for VS Code:

- Python from Microsoft,
- Pylance from Microsoft
- and Jupyter from Microsoft (optional it is only needed if you want to use Jupyter Notebooks in VS Code)

Alternatively you can use a `Jupyter Notebook` (`*.ipynb`) to execute code. Jupyter Notebooks are interactive documents that allow you to write and execute Python code line by line. In comparison to a Python script, which is a plain text file with the `.py` extension, Jupyter Notebooks provide a more user-friendly interface for data analysis and visualization.

Write the code in a cell and execute it by pressing the run button.

Advantage of using Jupyter Notebooks:

- The code can run cell by cell and the output is directly shown below the cell.
- Further you can write text and equations in markdown cells.
- Jupyter Notebooks can also handle different programming languages like `R`or `Julia`.
- Also `Markdown` and `LaTeX` can be used to write text and equations.
- The output of the code can be visualized directly in the notebook.
- Therefore, it is very popular in the data science community.

Jupyter Notebooks can be used in `VS Code`, `Jupyter Lab`, `Jupyter Notebook` or `Google Colab`.

In this course all exercises are provided as Jupyter Notebooks. You can download it or run it directly in Google Colab.

Using Jupyter Notebooks in VS Code

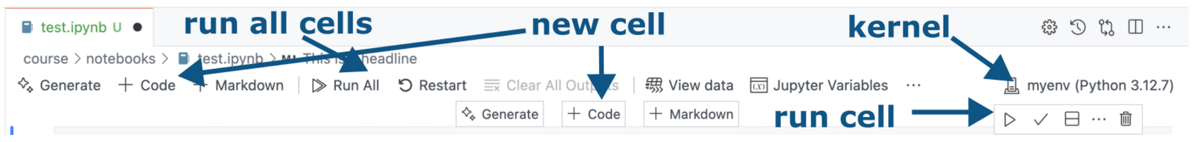
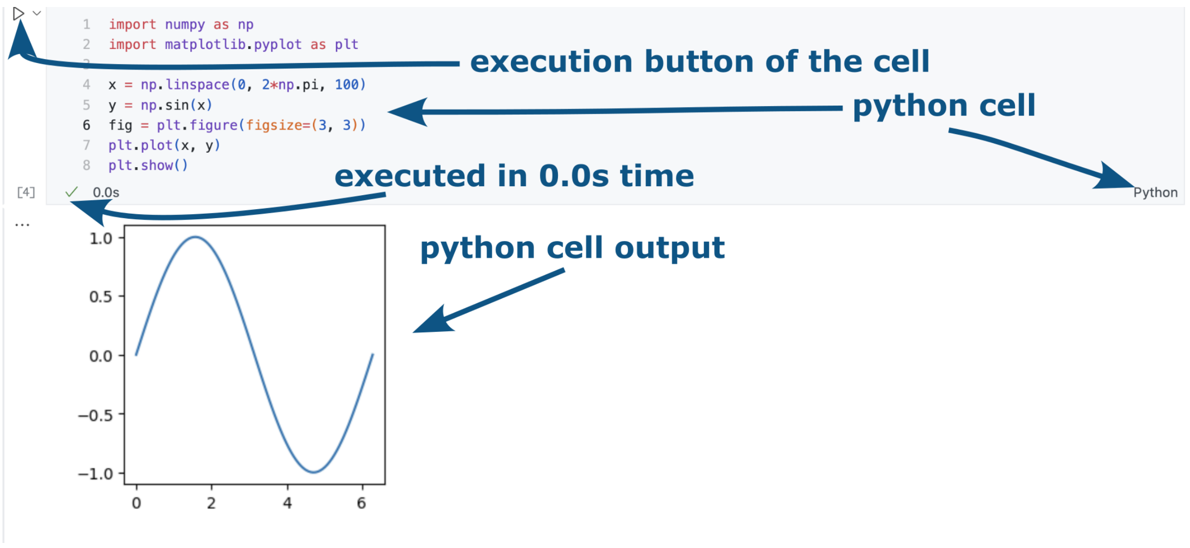


Figure 5.1: VS Code



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

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
fig = plt.figure(figsize=(3, 3))
plt.plot(x, y)
plt.show()
```

Figure 5.2: Python Cell

5.0.1 Difference Between Jupyter Notebooks and Python Scripts

- **Interactivity**: Jupyter Notebooks allow you to run code in chunks (cells) and see the output immediately, making them ideal for experimentation and visualization. Python scripts (`.py` files) are typically executed all at once.
- **Documentation**: Notebooks support Markdown cells for adding rich text, equations, and images alongside your code. Python scripts are plain text files and require comments for documentation.
- **Use Case**: Notebooks are great for exploratory data analysis and teaching, while Python scripts are better suited for production code and automation.

5.1 Google Colab Tutorial

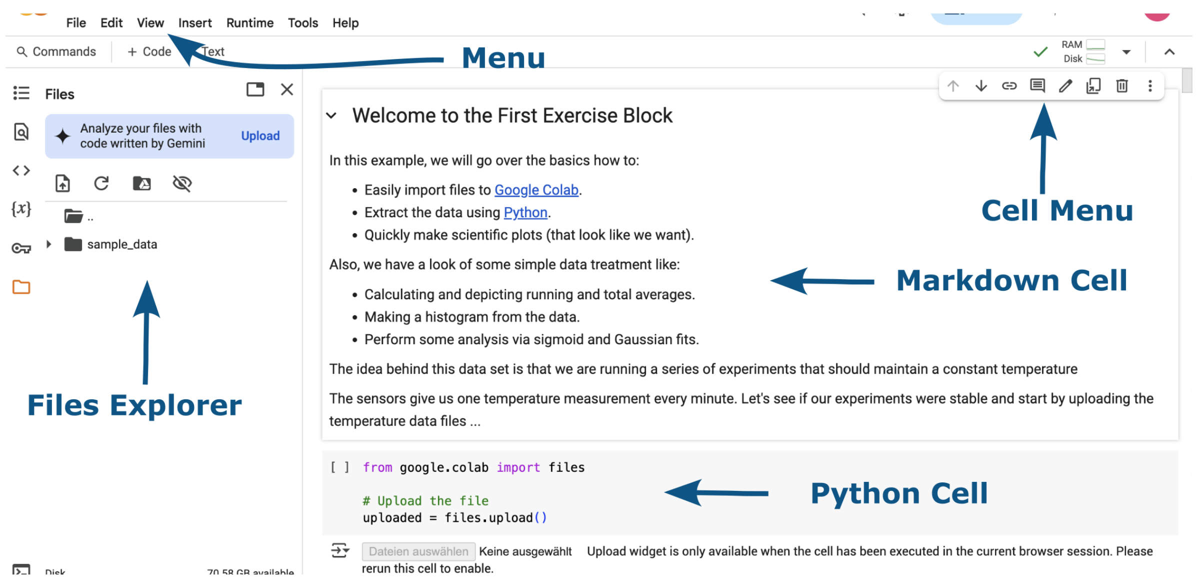
Google Colab is a free, cloud-based platform that allows you to run Jupyter Notebooks without any setup. To get started with Google Colab:

The use of Google Colab needs a Google Account. Please read the Terms of Service and Privacy Policy

1. Visit Google Colab and sign in with your Google account.
2. Create a new notebook or upload an existing `.ipynb` file.
3. Write and execute code in cells, just like in Jupyter Notebooks.
4. Save your work to Google Drive or download it as a `.ipynb` file.

5.1.1 Key Features of Google Colab

- **Free Access to GPUs/TPUs**: Accelerate your computations by enabling GPU or TPU support from the "Runtime" menu.
- **Collaboration**: Share notebooks with others and work on them simultaneously, similar to Google Docs.
- **Pre-installed Libraries**: Many popular Python libraries are pre-installed, saving you setup time.
- **Integration with Google Drive**: Easily access and save files to your Google Drive.



Welcome to the First Exercise Block

In this example, we will go over the basics how to:

- Easily import files to Google Colab.
- Extract the data using Python.
- Quickly make scientific plots (that look like we want).

Also, we have a look of some simple data treatment like:

- Calculating and depicting running and total averages.
- Making a histogram from the data.
- Perform some analysis via sigmoid and Gaussian fits.

The idea behind this data set is that we are running a series of experiments that should maintain a constant temperature

The sensors give us one temperature measurement every minute. Let's see if our experiments were stable and start by uploading the temperature data files ...

```
[ ] from google.colab import files

    # Upload the file
    uploaded = files.upload()
```

```
Dateien auswählen Keine ausgewählt   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```

Figure 5.3: Google Colab Interface

Upload data on Google Colab:

```
from google.colab import files
```

5.1.2 Upload a file

```
uploaded = files.upload()
```

5.1.3 Access the uploaded file

```
for filename in uploaded.keys():
    print(f'User uploaded file "{filename}" with size {len(uploaded[filename])} bytes.')#
```

5.1.4 Important Short Cuts in Google Colab

- Run cell and select next cell: `Shift + Enter`
- Run focused celll: `/Ctrl + Enter`
- Insert cell below: `/Ctrl + M B`
- Insert cell above: `/Ctrl + M A`

- **Delete cell**: `/Ctrl + M D`
- **Undo last action**: `/Ctrl + M Z`
- **Comment/Uncomment line**: `/Ctrl + Alt + M`
- **Find and replace**: `/Ctrl + H`
- **Show keyboard shortcuts**: `/Ctrl + K H`

Test your chosen setup

- open your editor
- open a new file and save this file as helloworld.py
- write your first test program:

```python
print("Hello World!")
```

- execute your program by using your IDE/Editor
- or using command line in the terminal (Linux/MacOS) / command prompt under Windows.

```
cd /path/to/file
python helloworld.py
```

The ouput should be: “Hello World!”.

If you are using Jupyter Notebook you can also write this code in a cell and execute it by pressing the run button.

Tip

Congratulations **You have successfully set up python!!**

Part II

Python Crash Course

6 Python Start

Difficulty level:

This will be a very fast and basic start into coding with Python.

Let‘s make an initial quiz.

You will see coding with Python is very easy and intuitive!
Please remember you can always open this file in `Google Colab` or download it and run it on your local machine .

Quiz

Elementary Syntax

Commands

Each line of code is a command. The computer reads the code from top to bottom and executes each command in succession order.

First some terminology:

Term	Definition
Command	*A line of code that tells the computer to do something.*
Syntax	*The rules that govern how commands are written.*
Execute	*To run a command.*
Comment	*A note in the code that is not executed.*
Indentation	*The space at the beginning of a line of code that tells the computer that the line is part of a block of code.*
Error	*A message that tells you that something is wrong with your code.*
Stack trace	*A list of error messages that Python prints when an error occurs.*
Variable	*A name that stores a value.*
Data type	*The type of value that a variable stores.*
Declaration	*The process of assigning memory to a variable.*
Integer	*A whole number.*
Float	*A number with a decimal point.*
Boolean	*A value that is either True or False.*
List	*A collection of items.*
String	*A sequence of characters.*
Function	*A block of code that performs a specific task.*
Argument	*A value that is passed to a function.*
Parameter	*A variable that is used in a function.*
Return	*The value that a function returns.*
Keyword Arguments	*Arguments that are passed to a function by name.*
Positional Arguments	*Arguments that are passed to a function by position.*
Default Argument	*An argument that has a default value.*
Operator	*A symbol that performs a specific operation.*
Expression	*A combination of values and operators that evaluates to a value.*
Statement	*A line of code that performs an action.*

Term	Definition
Block	*A group of statements that are executed together.*
Method	*A function that is associated with an object.*
Class	*A blueprint for creating objects.*
Object	*An instance of a class.*
Instance	*A specific object created from a class.*
Mutable	*A type of object that can be changed.*
Immutable	*A type of object that cannot be changed.*
Module	*A collection of functions and variables.*
Library	*A collection of modules.*
Package	*A collection of related modules.*
Script	*A file that contains code that can be run independently.*

Example - Hello World

`print` is a command that tells the computer to display the text or variable that follows it.

Try it out:

- Copy the code below and paste it into a code cell.
- Or open this file in `Google Colab` and run the code.
- Or download this file and run it on your local machine.

```python
print("Hello World")
```

```
Hello World
```

If you want to span a command over multiple lines you can use \ or if you are using parenthesis in your command you can directly use a new line.

```python
result = 1 + 2 + 3 + \
         7 + 8 + 9
numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
```

Indentation

In Python code blocks are not structured by brackets or semicolons like `C/C++` or `Java` but by indentation. This means that the code inside a loop or a function is indented by a tab or spaces.

Warning

Indentations are crucial in Python. If you don't indent your code correctly, you will get a typical beginner error.
Pay attention **do not mix tabs and spaces** in your code.

Example - Wrong Indentation

Try it out:

```python
print("correct indentation")
    print("wrong indentation")
```

```
IndentationError: unexpected indent (1842851760.py, line 2)
  Cell In[3], line 2
    print("wrong indentation")
    ^
IndentationError: unexpected indent
```

All the lines in the block must have the same indentation:

```python
    print("correct indentation")
    print("correct indentation")
    print("correct indentation")
```

```
correct indentation
correct indentation
correct indentation
```

Error messages

When you run a command that has an error, Python will print an error message.

The so-called stack trace. It is a list of error messages that Python prints when an error occurs.

It gives you information about the error and the location of the error in your code.

The stack track is read from bottom to top.

The last line contains the error message and the line number where the error occurred.

Tip

Often the error messages are not very clear. You can search for the error message in the internet. Stackoverflow has a lot of answers to common errors. Or you can ask some AI *e.g.* ChatGPT.

Example - Cryptic Error Message

What does the error message tell you? If you are not sure ask the internet or your favorite AI.

```python
a = [1,2,3]
print(a[3])
```

```
IndexError: list index out of range
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[5], line 2
      1 a = [1,2,3]
----> 2 print(a[3])
IndexError: list index out of range
```

Solution

The error message tells you that you are trying to access an index that is out of range.

Python counts from 0.

The **first** element is at **index 0**, the second element is at index 1, and so on.
e.g. `weekdays=["Monday", "Tuesday", "Wednesday"]`

elements	"Monday"	"Tuesday"	"Wednesday"
index	0	1	2

element `Tuesday` is at index `1`. The length of the list is `N`. The last element is at index `N-1`.

Comments

Comments are important. They help you and others to understand your code. You can use the # symbol to write comments.

Docstrings are used to document the code for example with pydoc. They are using triple quotes “‘ “ ” “ “ “ “ ‘.

```
# This is a comment

"""
This is a documentation.
You can document your code for example by pydoc
"""
```

```
'\nThis is a documentation.\nYou can document your code for example by pydoc\n'
```

Modules

There exist a lot of libraries and modules in Python. Libraries is a term to describe a collection of modules. Packages are a way to collect related modules together within a single tree-like hierarchy. Modules are a collection of files. A script is a file that can be run independently. You can use the `import` statement to import the whole module. You can use the `from` statement to import a specific part of the module. You can use the `dir()` function to list the names in a module. You can use the `help()` function to get help on a module.

```python
import numpy as np
# the library numpy is imported
from matplotlib import pyplot as plt
# the library pyplot is imported from matplotlib module
```

Variables

In Python, you don't need to declare the type of a variable. You can assign a value to a variable using the `=` operator. Python is managing the memory allocation for you.

```python
a = 1  #a is a variable
b = "String" #b is a string
print(1, " is an", type(a))
print(b, " is a", type(b))
```

```
1  is an <class 'int'>
String  is a <class 'str'>
```

Data Types

Python has several data types. The most common are:

Data Type	Description
`int`	Integers
`float`	Floating point numbers
`str`	Strings
`bool`	Booleans
`list`	Lists
`tuple`	Tuples
`dict`	Dictionaries
`set`	Sets

You can use the `type()` function to get the type of a variable. You can use the `isinstance()` function to check if a variable is an instance of a class. Type casting is the process of converting one data type to another. You can use the `int()`, `float()`, `str()`, `bool()`, `list()`, `tuple()`, `dict()`, `set()` functions to cast a variable to a different type.

```python
x = 5 #int
print(x,type(x))# print the type of x
```

```
5 <class 'int'>
```

```python
y = 5.12 #float
print(y,type(y))
```

```
5.12 <class 'float'>
```

```python
c = 2.8j #complex
print(c,type(c))
```

```
2.8j <class 'complex'>
```

```
s = "Hello World" #string
print(s,type(s))
```

```
Hello World <class 'str'>
```

```
print("length of word: ", len(s)) # length of string
print("character on position 2: ", s[2])
print("last 3 characters: ", s[-3:])
s2 = s + "!"
print(s2)
s3 = "\"Hello world\"!"
print(s3)
```

```
length of word:  11
character on position 2:  l
last 3 characters:  rld
Hello World!
"Hello world"!
```

```
d = dict(name="Max",lastname="Musterman",height=1.89) # dictionary
print(d,type(d))
```

```
{'name': 'Max', 'lastname': 'Musterman', 'height': 1.89} <class 'dict'>
```

```
b = True # boolean
print(b,type(b))
```

```
True <class 'bool'>
```

```
dataset = {1,12,3} # set
print(dataset,type(dataset))
```

```
{1, 3, 12} <class 'set'>
```

```
dataset2 = set((1.2,2,2)) # set
print(dataset2,type(dataset2))
```

```
{1.2, 2} <class 'set'>
```

```
r = range(0,10,2) # range
print(r,type(r))
```

```
range(0, 10, 2) <class 'range'>
```

```
l = [1,2,2,3] # list
print(l,type(l))
print("length of list",len(l))
```

```
[1, 2, 2, 3] <class 'list'>
length of list 4
```

```
t = (1,2)# tuple
print(t,type(t))
```

```
(1, 2) <class 'tuple'>
```

```
#type conversion
x = 5 #int
f = float(x)
print(f,type(f))
```

```
5.0 <class 'float'>
```

Mutable and Immutable Objects

Immutable objects cannot be changed. Mutable objects can be changed. Immutable objects are: `int`, `float`, `bool`, `str`, `tuple`, `frozenset`. Mutable objects are: `list`, `dict`, `set`.

That means if you change an immutable object, a new object is created. If you change a mutable object, the object is changed.

```python
a = 1
b = a

print("a:",a)
print("b:",b)

a = 2
print("------")
print("a:",a)
print("b:",b)

b = 3
print("------")
print("a:",a)
print("b:",b)

b = a
print("------")
print("a:",a)
print("b:",b)
```

```
a: 1
b: 1
------
a: 2
b: 1
------
a: 2
b: 3
```

```
------
a: 2
b: 2
```

mutable objects

```python
a = [1,2,3]
b = a

print("a:",a)
print("b:",b)

a[0] = 4
print("------")
print("a:",a)
print("b:",b)

b[1] = 5
print("------")
print("a:",a)
print("b:",b)
```

```
a: [1, 2, 3]
b: [1, 2, 3]
------
a: [4, 2, 3]
b: [4, 2, 3]
------
a: [4, 5, 3]
b: [4, 5, 3]
```

This happens because a and b are pointing to the same memory location. So if you change a, b will also change. If you want to avoid this, you can use the copy() method.

```python
b = a.copy()
print("------")
print("a:",a)
print("b:",b)

a[2] = 6
print("------")
print("a:",a)
print("b:",b)

b[2] = 7
print("------")
print("a:",a)
print("b:",b)
```

```
------
a: [4, 5, 3]
b: [4, 5, 3]
------
a: [4, 5, 6]
b: [4, 5, 3]
------
a: [4, 5, 6]
b: [4, 5, 7]
```

String Formatting

You can use the following escape characters:

Escape Character	Description
`\n`	New line
`\t`	Tab
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\b`	Backspace
`\r`	Carriage return
`\f`	Form feed
`\ooo`	Octal value
`\xhh`	Hex value

You can use the `+` operator to concatenate strings.

```
a = "This "
b = "is a string"
print(a + b)
```

```
This is a string
```

For print formatting you can use the `format()` method. You can use the `f-string` method. See for more information here

Other methods are the `%` operator and the `str.format()` method.

```
a = 1.5434
b = "nm"
print("This is an integer %d %s" % (a, b))
print("This is a float formating with minimum \
1 number of character wide and 2 digits %1.2f %s" % (a, b))
print("This is scientific notation with \
2 digits %.2e %s" % (a, b))
print("This is a string %s %s" % (a, b))
print("This is an example of using \
format() method {0} {1}".format(a, b))
print("This is an example of using format() \
method with named arguments {a} {b}".format(a=a, b=b))
print(f"This is an example of using f-string {a} {b}")
```

```
This is an integer 1 nm
This is a float formating with minimum 1 number of character wide and 2 digits 1.54 nm
This is scientific notation with 2 digits 1.54e+00 nm
This is a string 1.5434 nm
This is an example of using format() method 1.5434 nm
This is an example of using format() method with named arguments 1.5434 nm
This is an example of using f-string 1.5434 nm
```

Operators

There are different types of operators in Python.

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulo
`**`	Exponentiation
`//`	Floor division
`==`	Equal
`!=`	Not equal
`<`	Less than
`>`	Greater than
`<=`	Less than or equal
`>=`	Greater than or equal
`and`	Logical AND
`or`	Logical OR
`not`	Logical NOT
`is`	Identity
`in`	Membership

```python
a = 5.3
b = 2
c = 3

print("division: ", a/b)
print("division: ", b/c, " type: ", type(b/c))
print("integer division: ", a//b)
print("modulo: ", a%b)
print("float multiplication: ", a*b, " type: ", type(a*b))
print("integer multiplication: ", b*c, " type: ", type(b*c))
print("exponentiation: ", a**2)
```

```
division:  2.65
division:  0.6666666666666666  type:  <class 'float'>
integer division:  2.0
modulo:  1.2999999999999998
float multiplication:  10.6  type:  <class 'float'>
integer multiplication:  6  type:  <class 'int'>
exponentiation:  28.09
```

7 Python Intermediate

Difficulty level:

Often we need with user input, files, system and paths. In this chapter we will cover these topics.

I/O (Input/Output)

You can use the `print()` function to **print a message** to the screen. You can use the `input()` function to get **input from the user**. You can use the `open()` function to **open a file**. You can use the `write()` function to **write to a file**. You can use the `read()` function to **read from a file**. You can use the `close()` function to **close a file**. You can use the `with` statement to **open a file** and **automatically close** it when you are done. You can use the `os` module to **work with files and directories**. You can use the `sys` module to **work with command line arguments**. You can use the `argparse` module to work also with **command line arguments**.

This should print "Hello World!" to the console

```python
print("Hello World!")
```

```
Hello World!
```

This should ask the user to enter a number and print it to the console

```python
print(input("Enter a number: "))
```

This should write "Hello World!" to the file "file.txt"

```python
open("file.txt", "w").write("Hello World!")
```

```
12
```

This should read the file "file.txt" and print the content to the console

```python
print(open("file.txt").read())
```

```
Hello World!
```

This should print "Hello World!" to the console without a newline

```
print("Hello World without newline.", end="")
print("Next print statement.")
```

```
Hello World without newline.Next print statement.
```

This should read the file “file.txt” and print the content to the console

```
with open("file.txt", "r") as file: print(file.read())
```

```
Hello World!
```

Write a file with the content “Hello World!” and close it

```
file = open("file.txt", "w")
file.write("Hello World!")
file.close()
```

System

There are a lot of modules in Python to work with the system. You can use the `os` module to **work with files and directories**. You can use the `sys` module to **work with command line arguments**. You can use the `argparse` module to work also with **command line arguments**.

Most important functions are: - `os.getcwd()` to get the current working directory. - `os.chdir()` to change the current working directory. - `os.listdir()` to list the files in a directory. - `os.mkdir()` to create a directory. - `os.rmdir()` to remove a directory. - `os.remove()` to remove a file. - `os.rename()` to rename a file. - `os.path.exists()` to check if a file or directory exists. - `os.path.isfile()` to check if a file exists. - `os.path.isdir()` to check if a directory exists. - `os.path.join()` to join two paths. - `os.path.basename()` to get the base name of a path. - `os.path.dirname()` to get the directory name of a path. - `os.path.abspath()` to get the absolute path of a path. - `os.path.split()` to split a path into a directory and a file. - `os.path.splitext()` to split a path into a base name and an extension. - `os.path.getsize()` to get the size of a file. - `os.path.getmtime()` to get the modification time of a file.

- `sys.argv` to get the command line arguments.
- `sys.exit()` to exit the program.
- `sys.stdin` to read from the standard input.
- `sys.stdout` to write to the standard output.
- `sys.stderr` to write to the standard error.
- `argparse.ArgumentParser()` to create a parser.
- `add_argument()` to add an argument to the parser.
- `parse_args()` to parse the command line arguments.

This should remove the file "file.txt"

```
import os
os.remove("file.txt")
```

For `sys` you can use the `sys.argv` to get the command line arguments. `sys.argv` is a list of the command line arguments. `sys.argv[0]` is the name of the script. `sys.argv[1]` is the first argument.

```python
import sys
first_argument = sys.argv[1]
```

For more information about the sys module you can visit the official documentation.

For `Argparse` you can use the following code: For python scripts: You can use argparse to parse command line arguments.

```python
from argparse import ArgumentParser

parser = ArgumentParser(description="This is a description.")
parser.add_argument("--arg1", help="This is the first argument.")
parser.add_argument("---arg2", help="This is the second argument.")
args = parser.parse_args()
```

For more information about `Argparse` you can visit the official documentation.

Paths

Pay attention to the paths in your code. They are different defined in Windows and Linux. In Windows and macOS, you use backslashes `/` and in Linux, you use forward slashes `\`.

To avoid this problem you can use the `os` or `pathlib` module to make your code platform independent.

```
import os

path = os.path.join('folder1', 'folder2', 'folder3', 'data.dat')
print(path)
```

```
folder1/folder2/folder3/data.dat
```

```
working_dir = os.getcwd()
print(working_dir)
```

```
/Users/stk/dev/PythonForChemists_public/course/chapters/Python
```

```
from pathlib import Path

path = Path('folder1') / 'folder2' / 'folder3' / 'data.dat'
print(path)
```

```
folder1/folder2/folder3/data.dat
```

```
working_dir = Path.cwd()
print(working_dir)
```

```
/Users/stk/dev/PythonForChemists_public/course/chapters/Python
```

Lists

Lists collect multiple items in a single variable.

You can use the `[]` operator to create a list. You can use the `append()` method to add an item to a list. You can use the `insert()` method to add an item at a specific position. You can use the `del` statement to delete an item from a list. You can use the `remove()` method to remove an item from a list. You can use the `pop()` method to remove an item at a specific position. You can use the `clear()` method to remove all items from a list. You can use the `copy()` method to copy a list. You can use the `count()` method to count the number of items in a list. You can use the `sort()` method to sort a list. You can use the `reverse()` method to reverse a list. You can use the `extend()` method to add items from another list. You can use the `index()` method to get the index of an item. You can use the `len()` function to get the length of a list. You can use the `list()` function to create a list.

```
a = ['a', 'b', 'c']
b = [1,3,'a', 1j]
len(a) #length of list
```

```
3
```

Warning

Index is starting with **0** in Python.

```
l = ['first', 'second', 'third']
l[0]
```

```
'first'
```

Last element is reached by index -1.

```
l[-1]
```

```
'third'
```

Control Structures

Control structures are used to control the flow of a program. The most common control structures are:

- You can use the `if` statement to execute a block of code if a condition is true.
- You can use the `elif` statement to execute a block of code if the first condition is - false and the second condition is true.
- You can use the `else` statement to execute a block of code if the condition is false.
- You can use the `for` loop to iterate over a sequence.
- You can use the `while` loop to execute a block of code as long as a condition is true.
- You can use the `break` statement to exit a loop.
- You can use the `continue` statement to skip the rest of the code in a loop.
- You can use the `pass` statement to do nothing.

if statement

```python
x = 0
if x < 0:
    print("x < 0")
elif x > 0:
    print("x > 0")
else:
    print("x = 0")
```

```
x = 0
```

break,continue,pass

```python
for i in range(10):
    if i == 5:
        break
    print(i)
```

```
0
1
2
3
4
```

```python
for i in range(10):
    if i == 5:
        continue
    print(i)
```

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

```python
for i in range(10):
    if i == 5:
        pass
    print(i)
```

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

For Loops

```
for i in range(5): #from 0 to 4
    print(i)
```

```
0
1
2
3
4
```

```
for i in range(1,10,2): # start 1, stop 10 excluded, step 2
    print(i)
```

```
1
3
5
7
9
```

```
l = list(range(0,10))
```

```
l
```

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

```
for i in l:  # using list
    print(i)
```

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

While Loops

```
x = 0
while x  < 4:
    print(l[x])
    x = x + 1
```

```
0
1
2
3
```

```
print("while loop with continue and break statement")
n = 0
while(n < 10):
    n+=1
    if n == 5:
        continue
    if n == 7:
        print("The loop reached 7 and will break now.")
        break
    print(n)
```

```
while loop with continue and break statement
1
2
3
4
6
The loop reached 7 and will break now.
```

Functions

Functions are defined using the `def` keyword. You can use the `return` keyword to return a value from a function. The parameters of a function are defined in the parentheses. Multiple parameters are separated by commas. You can use default values for the parameter *e.g.* `b=5`. Multiple return values are separated by commas. They are stored in a tuple.

```
def summation(a,b=5):
    return a+b, a-b
```

```
summation(4,2)
```

```
(6, 2)
```

```
sum, sub = summation(4)
print(sum)
print(sub)
```

```
9
-1
```

```
x = 3

def multiple_return_value(x,a,b):
    n = x+a
    m = x-b
    return [n,m]
print(multiple_return_value(x,5,10)[0],multiple_return_value(x,5,10)[1])
```

```
8 -7
```

Style guideline for writing python code

For writing a readable code, it is important to follow a style guideline. The most common style guideline for Python is PEP 8.

Exercises

Download it locally and try to solve the exercises.

Basic Python

Or open it in `Google Colab`:

Basic Python

8 Basic Modules

Difficulty level:

Numpy and Pandas

Numpy

- Numpy is the most important library for Python.
- The standard data types in Python are very slow and not very efficient for data analysis.
- Numpy is based mainly on `C` an `C++`.
- This allows Numpy to be faster than plain Python.
- With Numpy a new data type is introduced **numpy arrays**.
- Numpy arrays are multidimensional arrays that are much faster than Python lists.
- The libary also includes many mathematical functions and methods for linear algebra.

More information can be found at the Numpy website.

Load the required libraries

```
import numpy as np
```

Numpy Arrays

An array can be described as multidimensional lists. For example a matrix is a 2D array.

```
mat = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(mat)
```

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

The elements of an array can be accessed using the index of the element.

```
print(mat[0, 0])   # 1 element at row 0, column 0


print(mat[1, 2])  # 6 element at row 1, column 2


print(mat[:, 0])  # [1 4 7] all elements in column 0


print(mat[1, :])  # [4 5 6] all elements in row 1
```

```
1
6
[1 4 7]
[4 5 6]
```

```
empty_mat = np.empty((3,3),dtype=float)
print(empty_mat)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
ones_mat = np.ones((3, 3))
print(ones_mat)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
zeros_mat = np.zeros((3, 3))
print(zeros_mat)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
arr = np.arange(1, 10, 2) # an array from 1 to 10 with a step of 2
print(arr)
```

```
[1 3 5 7 9]
```

```
arr2 = np.linspace(0,100,5) # an array from 0 to 100 with 5 elements
print(arr2)
```

```
[  0.  25.  50.  75. 100.]
```

```
rand_mat = np.random.rand(3,3) # a 3x3 matrix with random numbers between 0 and 1
print(rand_mat)
```

```
[[0.26706679 0.17764155 0.09276725]
 [0.54301213 0.86887834 0.67742749]
 [0.9525382  0.0509319  0.79483746]]
```

Often you need to know the shape of an array. The shape of an array is a tuple that contains the number of elements in each dimension.

```
print(mat.shape) # (3, 3) 3 rows and 3 columns
print(mat.size)  # 9 total number of elements
print(mat.ndim)  # 2 number of dimensions
print(mat.dtype) # int64 data type of the elements
```

```
(3, 3)
9
2
int64
```

Pandas

- Pandas is a library for data manipulation and analysis.
- It is built on top of Numpy.
- With Pandas you are working with **dataframes** and not with arrays like in Numpy.
- Dataframes are two-dimensional labeled data structures with columns of potentially different types.
- It is like a table in a database or a spreadsheet. Pandas has a lot of methods to manipulate dataframes.
- You can select subsets of the data, filter, sort, group, merge, join, etc.
- You can statistically analyze the data, export the data to different file formats, but also plot the data with the help of `matplotlib`.

More information can be found under Pandas website.

```
import pandas as pd
```

Panda DataFrames

Panda DataFrames are two-dimensional labeled data structures with columns of potentially different types like a table.

```
data = {
    "Name": ["Water", "Oxygen", "Hydrogen", "Carbon Dioxide", "Methane", "Ammonia", "Nitroger
    "Formula": ["H2O", "O2", "H2", "CO2", "CH4", "NH3", "N2", "SO2"],
    "Molar Mass (g/mol)": [18.015, 32.00, 2.016, 44.01, 16.04, 17.03, 28.013, 64.07]
}

df = pd.DataFrame(data)
print(df)
```

```
             Name Formula  Molar Mass (g/mol)
0           Water     H2O              18.015
1          Oxygen      O2              32.000
2        Hydrogen      H2               2.016
3  Carbon Dioxide     CO2              44.010
4         Methane     CH4              16.040
5         Ammonia     NH3              17.030
6        Nitrogen      N2              28.013
7  Sulfur Dioxide     SO2              64.070
```

Panda DataSeries

Panda DataSeries are one-dimensional labeled arrays.

```
series_data = pd.Series([1, 2, 3, 4, 5], index=["a", "b", "c", "d", "e"])
print(series_data)
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Matplotlib

Matplotlib is the main **plotting** library in Python. It is mainly used for static 2D plots.

More information can be found under Matplotlib website.

Later in this course we will get a detailed tutorial on how to use Matplotlib.

9 Advanced Modules

Difficulty level:

Advanced Modules

Scipy

Scipy is a library that builds on Numpy. It is specialized in **scientific computing**. It includes modules for statistical calculations, optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and more.

More information can be found at the Scipy website.

Later on we will see how to use Scipy for statistical calculations and for analysis of spectra.

Scikit-learn

Scikit-learn is a library for **machine learning**. It includes modules for classification, regression, clustering, dimensionality reduction, model selection and preprocessing.

More information can be found at the Scikit-learn website.

Later on we will see how to use Scikit-learn for classification and regression.

Seaborn

Seaborn is a library for **data visualization**. It is based on Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics.

More information can be found at the Seaborn website.

Later on we will see how to use Seaborn for data visualization.

Part III

Data Handling and Preprocessing

10 Introduction to Data Science in Chemistry and Materials Science

Difficulty level:

More information

- Data Science for Beginner by Microsoft
- Research Data Managment (RDM) in Chemistry
- An introduction to Python for Chemistry
- Scientific Computing for Chemists with Python
- Python for Data Science in Chemistry
- Introduction to Data Science with Python
- Practical Data Science in Python

Data Collection and Storage

What is data?

Data is a collection of

- numbers
- words
- measurements
- observations

If you work with data, please have the FAIR principles in mind:

- **F**indable
- **A**ccessible
- **I**nteroperable
- **R**eusable

see also the UIBK FAIR Principles

How do you get data?

At the beginning of every data science project, you need to collect data.

Data can be collected from various sources, such as:

- experimental measurements
- calculations and simulations
- surveys
- existing databases
- web scraping
- APIs

etc.

How do you store the data?

This data can be stored in different locations, such as:

- local files
- databases
- cloud storages
- cluster storages
- web servers
- etc.

If you measure data yourself, either your measurement device will store the data in a specific format, or you will have to store it yourself. Sometimes you have to transform the data into a different format to be able to work with it.

If you use data from external sources, you have to make sure that you have legal rights to use the data. Check the data license and terms of use.

Further, make sure how the data is stored and how you can access it.

How do you access external data?

To access data from external sources, you can use different tools and libraries.

For example, if you want to access data from a chemical database e.g.

- Materials Project, you can use the `pymatgen` library. (Requires API key, which can be obtained by registering on the website.)
- RCSB PDB Protein Data Bank, you can use the `rcsb` library.
- PubChem, you can use the `pubchempy` library.

… and many more.

If you use APIs please read the documentation of the API to understand how to access the data. And make sure to respect the API usage policy and database terms of use.

Which data formats do you use?

Data can be stored in various data formats, such as:

- Plain Text files (e.g. CSV, DAT, TXT)
- Text files with structure (e.g. JSON, XML)
- Spreadsheet files
- Binary files (e.g. HDF5, Parquet, NetCDF,Feather, Pickle, npy, etc.)
- Databases
- chemical and molecular data formats (e.g. XYZ, CIF, PDB, etc.) etc.

Plain Text Files

A text file often contains a header with the names of the columns and then the data in rows. Columns can be separated by different delimiters (`spaces`, `,`, `;`, `tabs`, …). For example, a file with data from an experiment could look like this:

```
Time/s  Temperature/°C
0         20
10        21
...     ...
```

JSON or XML

JSON stands for JavaScript Object Notation. Data is structured in a key-value format, so that both humans and machines can read it easily. For example, a JSON file with the same data as above could look like this:

```
{
  "data": [
    {"Time": 0, "Temperature": 20},
    {"Time": 10, "Temperature": 21},
    ...
  ]
}
```

XML stands for Extensible Markup Language. It is also designed to be both human-readable and machine-readable. For example, an XML file with the same data as above could look like this:

```
<data>
  <measurement>
    <Time>0</Time>
    <Temperature>20</Temperature>
  </measurement>
  <measurement>
    <Time>10</Time>
    <Temperature>21</Temperature>
  </measurement>
  ...
```

Binary Files

Data is stored in a binary format and can be read with specific libraries. It is often used for large datasets, as it is more efficient than plain text files. The computer is able to read and write binary files faster than text files. Some common binary file formats are:

- **HDF5**: Hierarchical Data Format, used for large datasets
- **Parquet**: Columnar storage format, used for big data
- **NetCDF**: Network Common Data Form, array-orriented, often for geoscience data
- **Feather**: a fast column-based serialization for data frames, initially designed for R and Python, helps to share data between languages
- **Pickle**: Python-specific format, used for serializing Python objects
- **npy**: Numpy-specific format, used for saving numpy arrays

Here is a list of comparison of binary file formats: Comparison of data serialization formats

Databases

Databases are designed for big data storage. The advantage of databases is that they can be queried and updated easily. There are different types of databases, such as:

- **SQL** databases (e.g. SQLite, MySQL, PostgreSQL)
- **NoSQL** databases (e.g. MongoDB)
- **Graph databases**
- **Time series databases**

Chemical and Molecular Data Formats

There are specific data formats for chemical and molecular data, such as:

- **XYZ**: Cartesian coordinates of atoms
- **CIF**: Crystallographic Information File, used for crystallographic data
- **PDB**: Protein Data Bank, used for protein structures
- **MOL**: Molecule file format, used for chemical structures
- **SDF**: Structure Data File, used for chemical structures
- **SMILES**: Simplified Molecular Input Line Entry System, used for chemical structures
- **InChI**: International Chemical Identifier, used for chemical structures
- **nmrML**: Nuclear Magnetic Resonance Markup Language, used for NMR data
- **NMReDATA**: Nuclear Magnetic Resonance Electronic Data Aggregation, used for NMR data
- **JCAMP-DX**: Joint Committee on Atomic and Molecular Physical Data, used for spectroscopy data
- **mzML**: Mass Spectrometry Markup Language, used for mass spectrometry data
- **animl**: Analytical Information Markup Language, used for analytical data
- **FASTA**: used for DNA and protein sequences

etc.

Other Data Formats

- **Images**: Images can be stored in different formats, such as JPEG, PNG, TIFF, BMP, GIF, etc.
- **Audio**: Audio files can be stored in different formats, such as MP3, WAV, FLAC, etc.
- **Video**: Video files can be stored in different formats, such as MP4, AVI, MOV, etc.

What type of data do you have?

Next what do you have to consider is the types of your data.

Depending on the type of data analysis could be different.

Numerical Data

Numerical data is data that is expressed with numbers. It can be further divided into two types:

- **Discrete**: Data that can only take certain values (e.g. integers)
- **Continuous**: Data that can take any value within a certain range (e.g. real numbers)

For example,

- measured temperature over time: **continuous**
- number of chemical substances, which are measured: **discrete**

Categorical Data

Categorial means that data is divided into categories. It can be further divided into two types:

- **Ordinal**: Data that has a specific order or ranking
- **Nominal**: Data that has no specific order or ranking

For example,

- blood type: **nominal** (A, B, AB, O)
- chemical function group: **nominal** (alcohol, ketone, aldehyde, carboxylic acid, etc.)
- purity of a substance: **ordinal** (low, medium, high)
- hardness of a material: **ordinal** (soft, medium, hard)

Quiz

11 Simple Data Import

Difficulty level:

Data Reading

How do you read the data?

Depending on the data format, you can use different libraries to read the data.

Reading Plain Text Files

You can use the `pandas` or `numpy` library to read CSV files.

Pandas

Pandas is read function is quite fast and can read large files. The advantage is that different data types can be read in the same file. The reading functions return a DataFrame object.

Pandas has different functions to read different file formats.

- `pandas.read_csv()` function is can read CSV files.
- `pandas.read_table()` function is can read general delimiter files.
- `pandas.read_fwf()` function is can read fixed-width files.

Mostly used function is `pandas.read_csv()` because you can specify the delimiter, header, and other options.

```python
import pandas as pd

data = pd.read_csv(temperature_data)
data
```

	time;temperature
0	1;303.073024218
1	2;302.951624807
2	3;302.831229733

	time;temperature
3	4;302.73615227
4	5;302.708880354
...	...
44635	44636;296.102947663
44636	44637;296.173110138
44637	44638;296.140000813
44638	44639;296.169289777
44639	44640;296.287904152

The data has a different delimiter than the default `comma`. You can specify the delimiter using the `sep` parameter.

```python
data = pd.read_csv(temperature_data, sep=';')
data
```

	time	temperature
0	1	303.073024
1	2	302.951625
2	3	302.831230
3	4	302.736152
4	5	302.708880
...	...	...
44635	44636	296.102948
44636	44637	296.173110
44637	44638	296.140001
44638	44639	296.169290
44639	44640	296.287904

Now the data is read correctly. The header is already taken from the first row. If you want to specify the header, you can use the `header` parameter.

```python
import pandas as pd
df = pd.read_csv('file.csv', header=None) # No header
df = pd.read_csv('file.csv', header=0) # Header is in the first row
df = pd.read_csv('file.csv', header=1) # Header is in the second row
```

```
data = pd.read_csv(temperature_data, sep=';', header=0)
data
```

	time	temperature
0	1	303.073024
1	2	302.951625
2	3	302.831230
3	4	302.736152
4	5	302.708880
...	...	...
44635	44636	296.102948
44636	44637	296.173110
44637	44638	296.140001
44638	44639	296.169290
44639	44640	296.287904

If your data contains whitespace, you can use the `skipinitialspace` parameter to remove initial whitespaces.

```
data = pd.read_csv(temperature_dat, skipinitialspace=True, sep=" ")
data
```

	1	303.073024218
0	2	302.951625
1	3	302.831230
2	4	302.736152
3	5	302.708880
4	6	302.647462
...	...	...
44634	44636	296.102948
44635	44637	296.173110
44636	44638	296.140001
44637	44639	296.169290
44638	44640	296.287904

Now the data has no header. You can specify the header using the `names` parameter.

```
data = pd.read_csv(temperature_dat, sep=' ', skipinitialspace=True,header=1,names=['t', 'T']
# important to set header=None, otherwise the first line is used as header
data
```

	t	T
0	3	302.831230
1	4	302.736152
2	5	302.708880
3	6	302.647462
4	7	302.513749
...	...	...
44633	44636	296.102948
44634	44637	296.173110
44635	44638	296.140001
44636	44639	296.169290
44637	44640	296.287904

You see that also not `.csv` files can be read with the `read_csv()` function.

The `read_csv()` function has a lot of parameters. Look in the documentation. You can see which parameters you can set https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

For example,

- `delimiter` parameter can be used to specify the delimiter instead of `sep`. Both are the same. Default is `,`.
- `header` parameter can be used to specify the header row. Default is inferred from the file.
- `skipinitialspace` parameter can be used to remove initial whitespaces. Default is `False`.
- `names` parameter can be used to specify the column names.
- `skiprows` parameter can be used to skip rows at the beginning of the file.
- `skipfooter` parameter can be used to skip rows at the end of the file.
- `nrows` parameter can be used to read only a specific number of rows.
- `usecols` parameter can be used to read only specific columns.
- `dtype` parameter can be used to specify the data type of the columns.

- `na_values` parameter can be used to specify the missing values.
- `keep_default_na` parameter can be used to specify if the default missing values should be kept. Default is `True`.
- `na_filter` parameter can be used to recognize missing values without `NA` or `NaN` values. Default is `True`.
- `true_values` parameter can be used to specify the values that should be recognized as `True`.
- `false_values` parameter can be used to specify the values that should be recognized as `False`.
- `parse_dates` parameter can be used to parse dates. Default is `False`.

Some examples are:

```
data = pd.read_csv(temperature_data, sep=';', header=0, names=['t', 'T'], skiprows=1)
```

Now the first row is skipped, only 44638 rows are read instead of 44639.

Missing value examples:

```
data = pd.read_csv(temperature_nan_dat, sep=' ', skipinitialspace=True,header=None,names=['t
print(data.loc[20:26]) #print some rows to see the NaN values
```

```
     t            T
20  21  302.020507467
21  22
22  23  301.845408096
23  24  301.833550446
24  25  301.785933229
25  26  301.846169501
26  27  301.779994697
```

If the `na_filter`is set to `False`, the missing values are not recognized. But if it set on `True`, the missing values are recognized.

```
data = pd.read_csv(temperature_nan_dat, sep=' ', skipinitialspace=True,header=None,names=['t
print(data.loc[20:26]) #print some rows to see the NaN values
```

```
     t          T
20  21  302.020507
21  22         NaN
22  23  301.845408
23  24  301.833550
24  25  301.785933
25  26  301.846170
26  27  301.779995
```

Numpy

Numpy has two main functions to read text files. - `numpy.loadtxt()` function is used to read text files. - `numpy.genfromtxt()` function is used to read text files with missing values.

In comparison to the `pandas` library, the `numpy` library is slower and can not read different data types in the same file. So you can not read a file with strings and numbers in the same file.

If you try to read a file with a header row, you will get an **error**.

```
import numpy as np
data = np.loadtxt(temperature_data, delimiter=';')
```

```
ValueError: could not convert string 'time' to float64 at row 0, column 1.
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
ValueError: could not convert string to float: 'time'
The above exception was the direct cause of the following exception:
ValueError                                Traceback (most recent call last)
Cell In[10], line 2
      1 import numpy as np
----> 2 data = np.loadtxt(temperature_data, delimiter=';')
File ~/y/envs/myenv/lib/python3.12/site-packages/numpy/lib/_npyio_impl.py:1380, in loadtxt(f
   1377 if isinstance(delimiter, bytes):
   1378     delimiter = delimiter.decode('latin1')
-> 1380 arr = _read(fname, dtype=dtype, comment=comment, delimiter=delimiter,
   1381             converters=converters, skiplines=skiprows, usecols=usecols,
   1382             unpack=unpack, ndmin=ndmin, encoding=encoding,
   1383             max_rows=max_rows, quote=quotechar)
   1385 return arr
```

```
File ~/y/envs/myenv/lib/python3.12/site-packages/numpy/lib/_npyio_impl.py:1021, in _read(fnam
   1018     data = _preprocess_comments(data, comments, encoding)
   1020 if read_dtype_via_object_chunks is None:
-> 1021     arr = _load_from_filelike(
   1022         data, delimiter=delimiter, comment=comment, quote=quote,
   1023         imaginary_unit=imaginary_unit,
   1024         usecols=usecols, skiplines=skiplines, max_rows=max_rows,
   1025         converters=converters, dtype=dtype,
   1026         encoding=encoding, filelike=filelike,
   1027         byte_converters=byte_converters)
   1029 else:
   1030     # This branch reads the file into chunks of object arrays and then
   1031     # casts them to the desired actual dtype.  This ensures correct
   1032     # string-length and datetime-unit discovery (like `arr.astype()`).
   1033     # Due to chunking, certain error reports are less clear, currently.
   1034     if filelike:
ValueError: could not convert string 'time' to float64 at row 0, column 1.
```

You can specify the header row using the `skiprows` parameter. If you want to skip one row, the `skiprows=1` parameter is set at 1.

```python
data = np.loadtxt(temperature_data, delimiter=';', skiprows=1)
data
```

```
array([[1.00000000e+00, 3.03073024e+02],
       [2.00000000e+00, 3.02951625e+02],
       [3.00000000e+00, 3.02831230e+02],
       ...,
       [4.46380000e+04, 2.96140001e+02],
       [4.46390000e+04, 2.96169290e+02],
       [4.46400000e+04, 2.96287904e+02]])
```

Now the data is read correctly. You can see that the data is read as a numpy array and not as a DataFrame. This can be a disadvantage if you want to use the data as a DataFrame but an advantage if you want to use numpy functions to process the data.

If you have data with whitespace, you do not need to specify the `delimiter`paramter because the default is whitespace.

```
data = np.loadtxt(temperature_dat)
data
```

```
array([[1.00000000e+00, 3.03073024e+02],
       [2.00000000e+00, 3.02951625e+02],
       [3.00000000e+00, 3.02831230e+02],
       ...,
       [4.46380000e+04, 2.96140001e+02],
       [4.46390000e+04, 2.96169290e+02],
       [4.46400000e+04, 2.96287904e+02]])
```

`genfromtxt()` gives you more flexibility to read files with missing values.

First using the `loadtxt()` function, you get an **error** because of the missing values.

```
data = np.loadtxt(temperature_nan_dat)
data
```

```
ValueError: the number of columns changed from 2 to 1 at row 22; use `usecols` to select a su
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 data = np.loadtxt(temperature_nan_dat)
      2 data
File ~/y/envs/myenv/lib/python3.12/site-packages/numpy/lib/_npyio_impl.py:1380, in loadtxt(fr
   1377 if isinstance(delimiter, bytes):
   1378     delimiter = delimiter.decode('latin1')
-> 1380 arr = _read(fname, dtype=dtype, comment=comment, delimiter=delimiter,
   1381             converters=converters, skiplines=skiprows, usecols=usecols,
   1382             unpack=unpack, ndmin=ndmin, encoding=encoding,
   1383             max_rows=max_rows, quote=quotechar)
   1385 return arr
File ~/y/envs/myenv/lib/python3.12/site-packages/numpy/lib/_npyio_impl.py:1021, in _read(fnam
   1018     data = _preprocess_comments(data, comments, encoding)
   1020 if read_dtype_via_object_chunks is None:
-> 1021     arr = _load_from_filelike(
   1022         data, delimiter=delimiter, comment=comment, quote=quote,
   1023         imaginary_unit=imaginary_unit,
   1024         usecols=usecols, skiplines=skiplines, max_rows=max_rows,
   1025         converters=converters, dtype=dtype,
   1026         encoding=encoding, filelike=filelike,
   1027         byte_converters=byte_converters)
   1029 else:
   1030     # This branch reads the file into chunks of object arrays and then
   1031     # casts them to the desired actual dtype.  This ensures correct
   1032     # string-length and datetime-unit discovery (like `arr.astype()`).
   1033     # Due to chunking, certain error reports are less clear, currently.
   1034     if filelike:
ValueError: the number of columns changed from 2 to 1 at row 22; use `usecols` to select a su
```

If you try to read the file with an empty entrance at row 22, you wil get still an **error** with the `genfromtxt()` function.

```
data = np.genfromtxt(temperature_nan_dat)
data
```

```
ValueError: Some errors were detected !
    Line #22 (got 1 columns instead of 2)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[14], line 1
----> 1 data = np.genfromtxt(temperature_nan_dat)
      2 data
File ~/y/envs/myenv/lib/python3.12/site-packages/numpy/lib/_npyio_impl.py:2331, in genfromtx
   2329 # Raise an exception ?
   2330 if invalid_raise:
-> 2331     raise ValueError(errmsg)
   2332 # Issue a warning ?
   2333 else:
   2334     warnings.warn(errmsg, ConversionWarning, stacklevel=2)
ValueError: Some errors were detected !
    Line #22 (got 1 columns instead of 2)
```

But why? What do you think is the reason for the error?

Caution

The reason is that the `genfromtxt()` function expects the same number of columns in each row. The delimiter is set default to `whitespace`. But if you have a missing value, the function expects a value. An error is raised because at row 22 the function is detecting only one column due to the missing value.

How can you solve this problem?

Caution

You can **NOT** solve this problem with the `genfromtxt()` function if you have missing values and delimiter is whitespace. Either you have to fill the missing value with a value or you have to use the `pandas` library.

If you have not `whitespace` as delimiter, you can use the `genfromtxt()` function with missing values.

```
data = np.genfromtxt(temperature_nan_data,delimiter=';')
data[20:26] # print some rows to see the NaN values
```

```
array([[ 21.        , 302.02050747],
       [ 22.        ,          nan],
       [ 23.        , 301.8454081 ],
       [ 24.        , 301.83355045],
       [ 25.        , 301.78593323],
       [ 26.        , 301.8461695 ]])
```

The different parameters that can be set are for `loadtxt()` function:

(see documentation https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.htm l

- `delimiter` parameter can be used to specify the delimiter. Default is whitespace.
- `skiprows` parameter can be used to skip rows at the beginning of the file.
- `usecols` parameter can be used to read only specific columns.
- `dtype` parameter can be used to specify the data type of the columns.
- `comments` parameter can be used to specify the comment character. Default is `#`.
- `max_rows` parameter can be used to read only a specific number of rows after skipping rows.
- `unpack` parameter can be used to unpack the columns, so each column is returned as a separate array.

and for `genfromtxt()` function

(see documentation https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html#numpy.genfromtxt)

- `delimiter` parameter can be used to specify the delimiter. Default is whitespace.

- `skip_header` parameter can be used to skip rows at the beginning of the file.
- `skip_footer` parameter can be used to skip rows at the end of the file.
- `usecols` parameter can be used to read only specific columns.
- `dtype` parameter can be used to specify the data type of the columns.
- `comments` parameter can be used to specify the comment character. Default is `#`.
- `max_rows` parameter can be used to read only a specific number of rows after skipping rows.
- `unpack` parameter can be used to unpack the columns, so each column is returned as a separate array.
- `missing_values` parameter can be used to specify which values should be recognized as missing values.
- `filling_values` parameter can be used to specify the filling values for the missing values.
- `usemask` parameter can be used to return a masked array with missing values.
- `names` parameter can be used to specify the column names. If `names=True`, the column names are read from the first row.
- `replace_space` parameter can be used to replace spaces in the column names. Default is `_`.

etc.

! Important

The `pandas` library is faster and more flexible than the `numpy` library. Choose wisely which library you want to use. It depends on the data format, the data type and what kind of processing you want to do.

Data Writing

Numpy

Numpy has one function to write text files.

- `numpy.savetxt()` function is used to write text files.

```
np.savetxt('file.txt', data)
```

! Problems

If you have problems with importing data please check the following:

- Have first a look at the data file.
- Check the delimiter.
- Check the header.
- Check the missing values.
- Check the data type.
- Check the encoding.
- Check the file format.
- Check the file path.
- Check the file name.

You can use AI tool to get a code snippet for reading the data.

- *e.g.* give the AI tool the one or two line of the as an example and ask for the code snippet.
- *e.g.* beware do that only if you do not have sensitive data.

12 Simple Data Inspection

Difficulty level:

Inspection of Data

After you load your data you have to inspect it to:

- check if no data is consistent, no missing values
- check if the data is in the correct format
- check if the data is in the correct range
- check if the data is in the correct distribution
- get first insights into the data

Numpy

Overview of the Data

```
import numpy as np
data = np.loadtxt(temperature_data, delimiter=';', skiprows=1)
```

First of all you can use the `print()` function to get a quick overview of the data.

```
print(data)
```

```
[[1.00000000e+00 3.03073024e+02]
 [2.00000000e+00 3.02951625e+02]
 [3.00000000e+00 3.02831230e+02]
 ...
 [4.46380000e+04 2.96140001e+02]
 [4.46390000e+04 2.96169290e+02]
 [4.46400000e+04 2.96287904e+02]]
```

In numpy you can use the `shape` attribute to get the shape of the data.

```
print(data.shape)
```

```
(44640, 2)
```

The data has 44640 rows and 2 columns.

Data Type Conversion

You can use the `dtype` attribute to get the data type of the data.

```
print(data.dtype)
```

```
float64
```

The data type is `float64`.

You can transform the data to a `float64` data type if it is not already in this format.

```
data = data.astype('float64')
```

Missing Data and Corrupted Data

You can use the `isnan()` function to check if there are missing values in the data.

```
print(np.isnan(data).sum())
```

```
0
```

Pandas

Overview of the Data

You can use the `head()` function to get a quick overview of the first rows of the data.

```python
import pandas as pd

data = pd.read_csv(temperature_data,sep=';')
data.head()
```

	time	temperature
0	1	303.073024
1	2	302.951625
2	3	302.831230
3	4	302.736152
4	5	302.708880

The `describe()` function gives you a quick overview of the data distribution.

```python
data.describe()
```

	time	temperature
count	44640.000000	44640.000000
mean	22320.500000	297.937566
std	12886.602345	6.080222
min	1.000000	278.814670
25%	11160.750000	293.842755
50%	22320.500000	297.875930
75%	33480.250000	301.879560
max	44640.000000	316.006103

The `describe`function shows you the count, mean, standard deviation, minimum, 25%, 50%, 75% and maximum values of the data.

In this case, data has 44640 data points, The mean of the temperature is 298(6) K. The minimum temperature is 279 K and the maximum temperature is 316 K. Further the 25% quantile is 294 K, the 50% quantile is 298 K and the 75% quantile is 302 K. The measurement was taken from 1 to 44640 seconds which is 12 hours and 24 minutes. We suppose that is the correct time range which was to be expected.

This gives you a quick overview of the data distribution.

Missing Data and Corrupted Data

To check if there is missing data in the data set you can use the `isna()` function.

```
data.isna().sum()
```

```
time           0
temperature    0
dtype: int64
```

No missing data is found in this case.

You can check the data type using `dtypes` function to check if the data is in the correct format.

```
data.dtypes
```

```
time             int64
temperature    float64
dtype: object
```

```
data.dtypes
```

```
time             int64
temperature    float64
dtype: object
```

You see that `time` is an `int64` and `temperature` is a `float64`. For the analysis, you might want to convert the `time` to a `float64` as well.

```
data['time'] = data['time'].astype('float64')
```

```
data['time'] = data['time'].astype('float64')
data.dtypes
```

```
time           float64
temperature    float64
dtype: object
```

If we have missing data we can use the `fillna()` function to fill the missing data with a specific value.

```
data.fillna(0)
```

	time	temperature
0	1.0	303.073024
1	2.0	302.951625
2	3.0	302.831230
3	4.0	302.736152
4	5.0	302.708880
...	...	...
44635	44636.0	296.102948
44636	44637.0	296.173110
44637	44638.0	296.140001
44638	44639.0	296.169290
44639	44640.0	296.287904

```
data_missing = pd.read_csv(temperature_nan_data, header=None, skipinitialspace=True, sep=' '
data_missing.head()
```

	time	temperature
0	1	303.073024
1	2	302.951625
2	3	302.831230
3	4	302.736152
4	5	302.708880

One value is missing in the `temperature` column. We fill it with 0.

First let check where the data is missing.

```
data[data['temperature'].isna()]
```

time	temperature

At index 21 at time 22 s the temperature is missing.

Note

The handling of missing data is a complex topic. First of all you have to check why the data is missing. Is it a measurement error, a data processing error etc.
You have to decide if you want to fill the missing data with a specific value, drop the row or column or interpolate the missing data. The decision depends on the data and the analysis you want to perform. Dropping Data is always a delicate decision because you loose information. Sometimes it is not good scientific practice to drop data. For more information there a lot of research in this topic https://doi.org/10.1076/edre.7.4.353.8937

The time step can be estimated by the difference between the time steps of the previous and the next data point.

```python
data['time'].diff()
```

```
0        NaN
1        1.0
2        1.0
3        1.0
4        1.0
        ...
44635    1.0
44636    1.0
44637    1.0
44638    1.0
44639    1.0
Name: time, Length: 44640, dtype: float64
```

And we can summarize it via:

```python
data['time'].diff().value_counts()
```

```
time
1.0    44639
Name: count, dtype: int64
```

```python
data_missing['time'].diff().value_counts()
```

```
time
1.0    44639
Name: count, dtype: int64
```

get difference between temperature values

```
data_missing[10:30].diff()
```

	time	temperature
10	NaN	NaN
11	1.0	0.000728
12	1.0	-0.062319
13	1.0	-0.130198
14	1.0	-0.060203
15	1.0	0.003234
16	1.0	0.048911
17	1.0	-0.042025
18	1.0	0.021264
19	1.0	0.033401
20	1.0	0.054579
21	1.0	NaN
22	1.0	NaN
23	1.0	-0.011858
24	1.0	-0.047617
25	1.0	0.060236
26	1.0	-0.066175
27	1.0	-0.080531
28	1.0	-0.029080
29	1.0	0.026428

The time step is constantly 1 second. The difference between the temperature of the previous and the next data point is at 10^{-2} order. We can assume that in this case the data is consistent enough and we can fill the missing data with the mean of the previous and the next data point.

```
data['temperature'].fillna((data['temperature'].shift() + data['temperature'].shift(-1))/2, i
```

```
data_missing['temperature'].fillna((data_missing['temperature'].shift() + data_missing['tempe
data_missing[20:25]
```

	time	temperature
20	21	302.020507
21	22	301.932958

	time	temperature
22	23	301.845408
23	24	301.833550
24	25	301.785933

Now can analysis or plot the data.

Data Type Conversion

You get the data type of the data with the `dtypes` function.

```
data.dtypes
```

```
time           float64
temperature    float64
dtype: object
```

You can convert the data type with the `astype()` function.

```
data['time'] = data['time'].astype('float64')
```

Example: Data Import and Inspection

Exercises

Download it locally and try to solve the exercises.

Data Import and Inspection Example

Or open it in `Google Colab`:

Data Import and Inspection Example

Part IV

Data Visualization 1

13 Introduction into Data Visualization

Difficulty level:

Data Visualization

Visualization of data helps us to understand the data better and communicate the results. It is easier to understand a graph than a table of numbers.

Which technique should you use?

- How many variables do you have and how many do you want to compare at once?
- What type of data do you have?
- What type of relationship are you trying to represent?
- Do you want to analyze the distribution of the data?
- Do you want to analyze the correlation between the variables?
- Do you want to show ranking or ordering of the data?
- Do you want to a trend over time?
- Do you want to show the composition of the data?
- Do you want to predict the future values of the data?
- Do you want to show the relationship between the variables?

Additional resources

A good handbook for data visualization in Python is the book "Python Data Science Handbook" by Jake VanderPlas

---

How you should visualize data?

- A graphic should have a suitable font size.
- Choose colors that are easy to distinguish also for colorblind people (e.g. colorbrewer.
- The axis ticks and limits should be chosen in a way that the data is easy to read.
- Axis should be labeled and physical units should be not forgotten.

- The graphic should be not overloaded with information.
- Have always in mind who is the target audience of your graphic and where the graphic will be presented *e.g.* in a scientific paper, in a presentation or in a poster.

14 Basic Data Visualization Techniques

Difficulty level:

Basic Data Visualization Technique

The most popular data visualization libraries in Python is Matplotlib. Let‘s start with the basic data visualization techniques using Matplotlib.

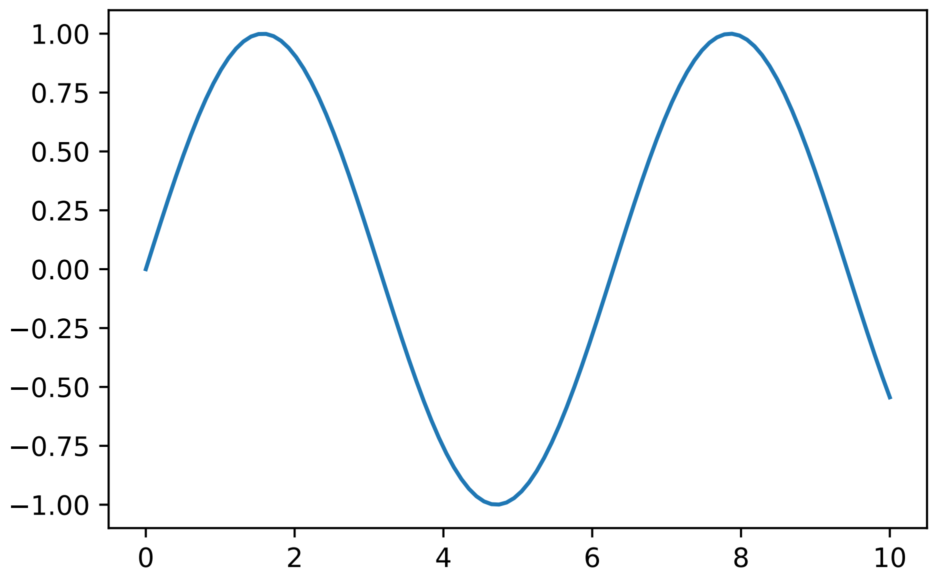
1. Generate some x-y data points.

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

x = np.linspace(0, 10, 100)
y = np.sin(x)
```

---

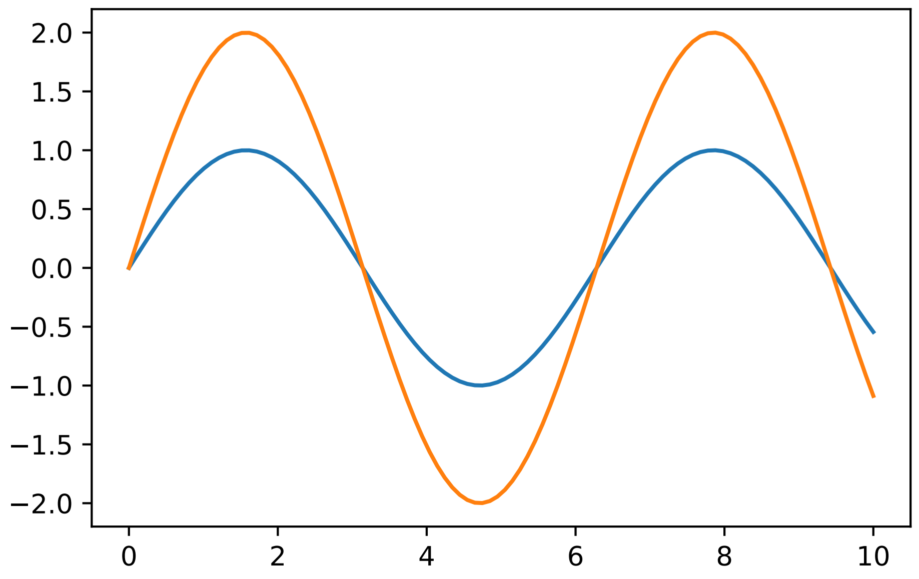
2. Plot the data points.

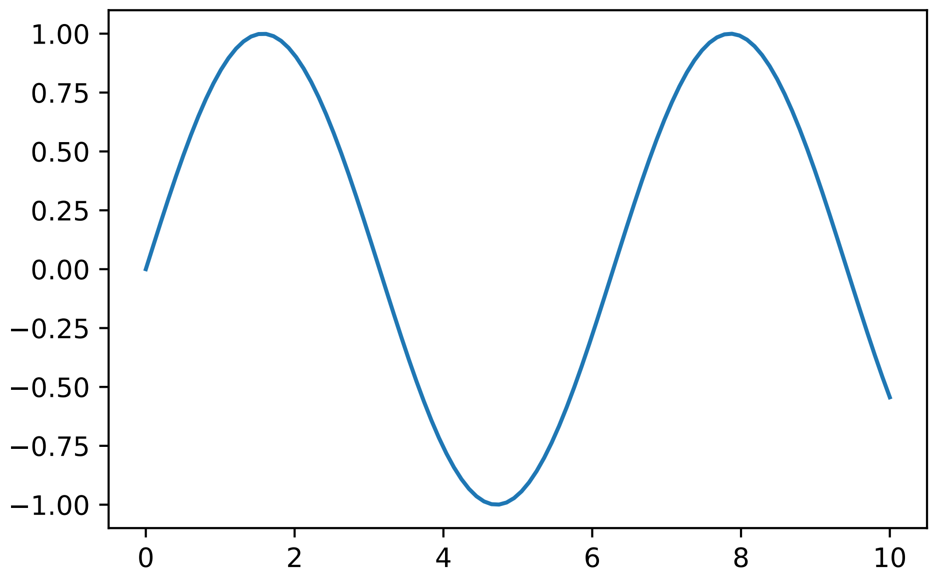
```
plt.plot(x, y)
plt.show()
```



To add more graphs to the same figure, use `plt.plot()` multiple times before `plt.show()`. If you want to create a new figure, use `plt.figure()` before `plt.plot()`.

```
plt.plot(x,y)
plt.plot(x,2*y)
plt.show()
plt.plot(x, y)
plt.show()
```





---

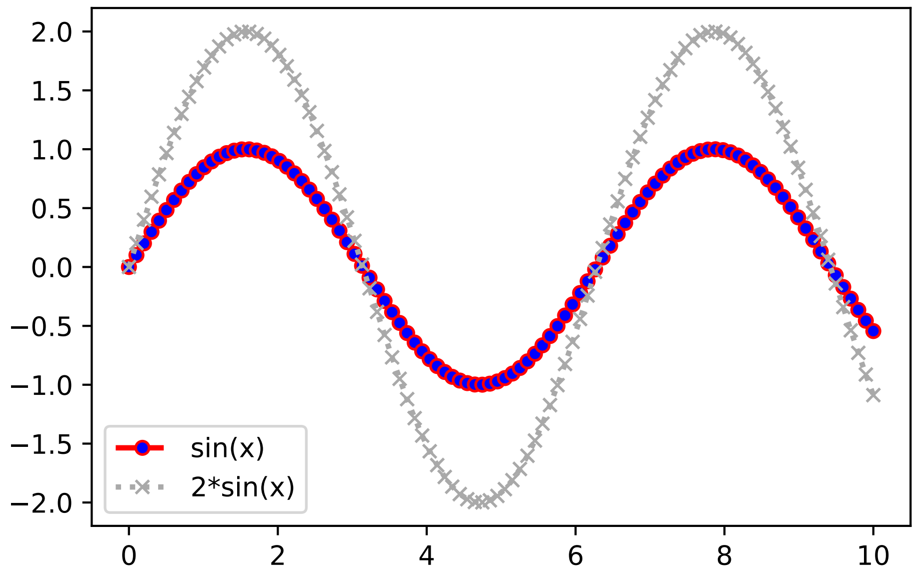
3. Adjust the plot.

The `plot()` function takes the following arguments:

- x-axis data points
- y-axis data points
- color: hex, or color name (e.g., 'red', 'blue','black'), abbreviated (e.g., 'r', 'b','k')
- linestyle: '-', '–', '-.', ':' or "solid", "dashed", "dashdot", "dotted"
- marker: 'o', 'x', '+', '*', 's', 'd', '^', 'v', '>', '<', 'p', 'h'
- linewidth - width of the line
- alpha - transparency of the line
- markerfacecolor - color of the marker face
- markersize - size of the marker
- label - label for the data points

You have to call `plt.legend()` to show the labels.

```
plt.figure()
plt.plot(x, y, color='red', linestyle='dashed', linewidth=2, marker='o',
            markerfacecolor='blue', markersize=5,
            label='sin(x)')
plt.plot(x, 2*y, color='darkgrey', linestyle='dotted', linewidth=2, marker='x',
             markerfacecolor='grey', markersize=5,
             label='2*sin(x)')
plt.legend()
plt.show()
```



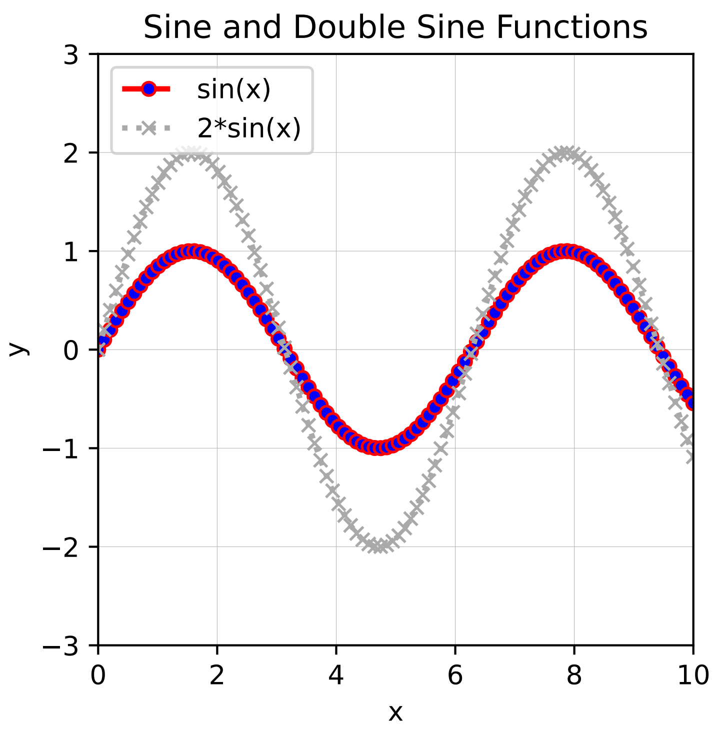
---

4. Adust the figure

This plot figure can be adjusted by changing the figure size, title, labels, and so on.

- `plt.xlabel()`: Set the x-axis label of the current axis.
- `plt.ylabel()`: Set the y-axis label of the current axis.
- `plt.title()`: Set a title for the axes.
- `plt.legend()`: Place a legend on the axes.
- `plt.grid()`: Configure the grid lines.
- `plt.xlim()`: Get or set the x-limits of the current axes.
- `plt.ylim()`: Get or set the y-limits of the current axes.
- `plt.xticks()`: Get or set the current tick locations and labels of the x-axis.
- `plt.yticks()`: Get or set the current tick locations and labels of the y-axis.
- `plt.figure()`: Create a new figure.
- `plt.show()`: Display a figure.

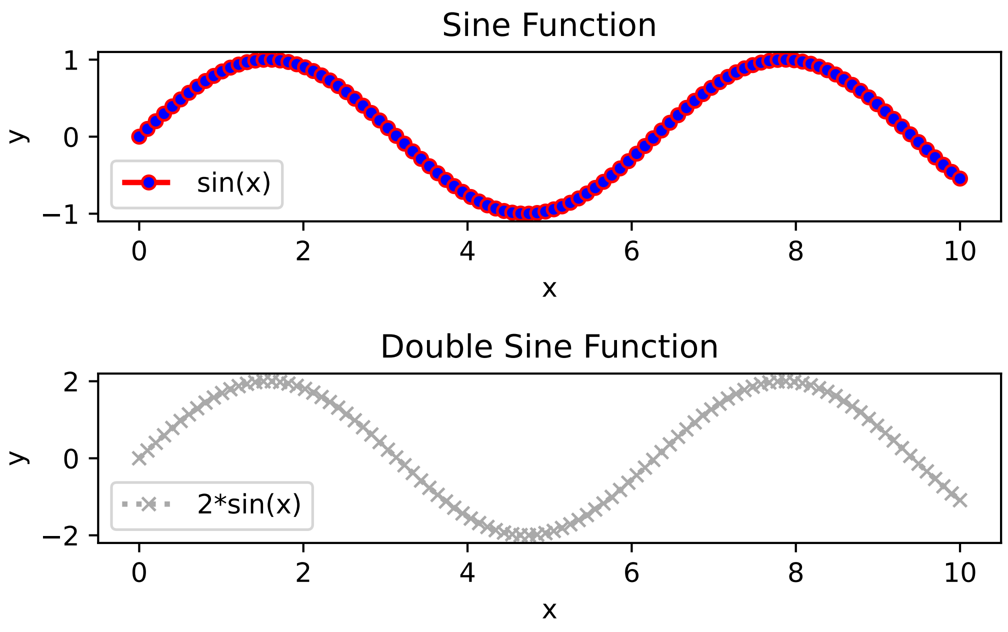
```
plt.figure(figsize=(4, 4), dpi=100)
# Create a figure with a specific size and resolution
plt.plot(x, y, color='red', linestyle='dashed', linewidth=2, marker='o',
            markerfacecolor='blue', markersize=5,
            label='sin(x)')
plt.plot(x, 2*y, color='darkgrey', linestyle='dotted', linewidth=2, marker='x',
            markerfacecolor='grey', markersize=5,
            label='2*sin(x)')
plt.xlim([0, 10]) # set the x-axis limits
plt.ylim([-3, 3]) # set the y-axis limits
plt.xticks(np.arange(0, 11, 2)) # set the x-axis ticks
plt.yticks(np.arange(-3, 4, 1)) # set the y-axis ticks
plt.xlabel('x') # set the x-axis label
plt.ylabel('y') # set the y-axis label
plt.title('Sine and Double Sine Functions') # set the title of the plot
plt.grid(linewidth=0.1)# set the grid linewidth
plt.legend(loc='upper left') # set the location of the legend: upper left, upper right, lowe
plt.show()
```



Creating multiple plots

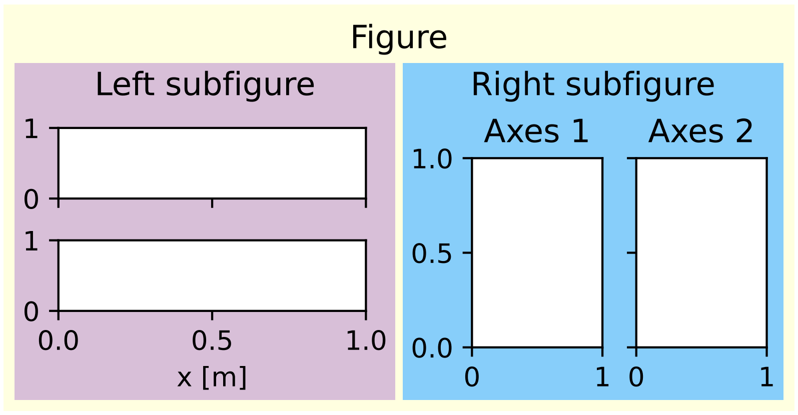
You can create multiple plots in the same figure by using the `subplot()` function.

```
plt.subplot(2, 1, 1)
plt.plot(x, y, color='red', linestyle='dashed', linewidth=2, marker='o',
            markerfacecolor='blue', markersize=5,
            label='sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Sine Function')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(x, 2*y, color='darkgrey', linestyle='dotted', linewidth=2, marker='x',
            markerfacecolor='grey', markersize=5,
            label='2*sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Double Sine Function')
plt.legend()
plt.tight_layout()
plt.show()
```



```
fig = plt.figure(figsize=(4,2),dpi=72,layout='constrained', facecolor='lightyellow')
# Create a figure with a specific layout and background color
# constrained layout automatically adjusts the subplots to fit the figure
fig.suptitle('Figure') # set the title of the figure object
subL, subR = fig.subfigures(1, 2) # create two subfigures
subL.set_facecolor('thistle') # set the background color of the left subfigure
sub_subL = subL.subplots(2, 1, sharex=True) # create two subplots in the left subfigure
sub_subL[1].set_xlabel('x [m]')
subL.suptitle('Left subfigure') # set the title of the left subfigure
subR.set_facecolor('lightskyblue') # set the background color of the right subfigure
sub_subR = subR.subplots(1, 2, sharey=True)
sub_subR[0].set_title('Axes 1') # set the title of the first subplot in the right subfigure
sub_subR[1].set_title('Axes 2') # set the title of the second subplot in the right subfigure
subR.suptitle('Right subfigure')
```

```
Text(0.5, 0.98, 'Right subfigure')
```

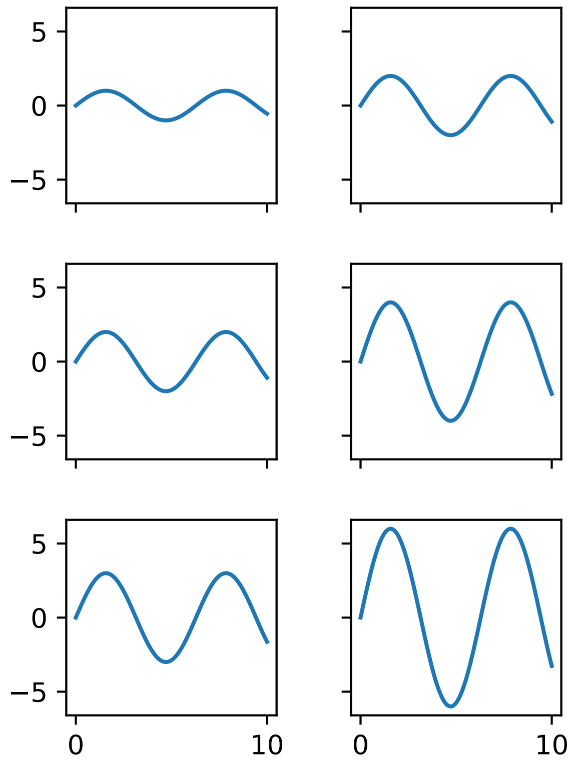


14.0.1 Subfigures and Gridspec

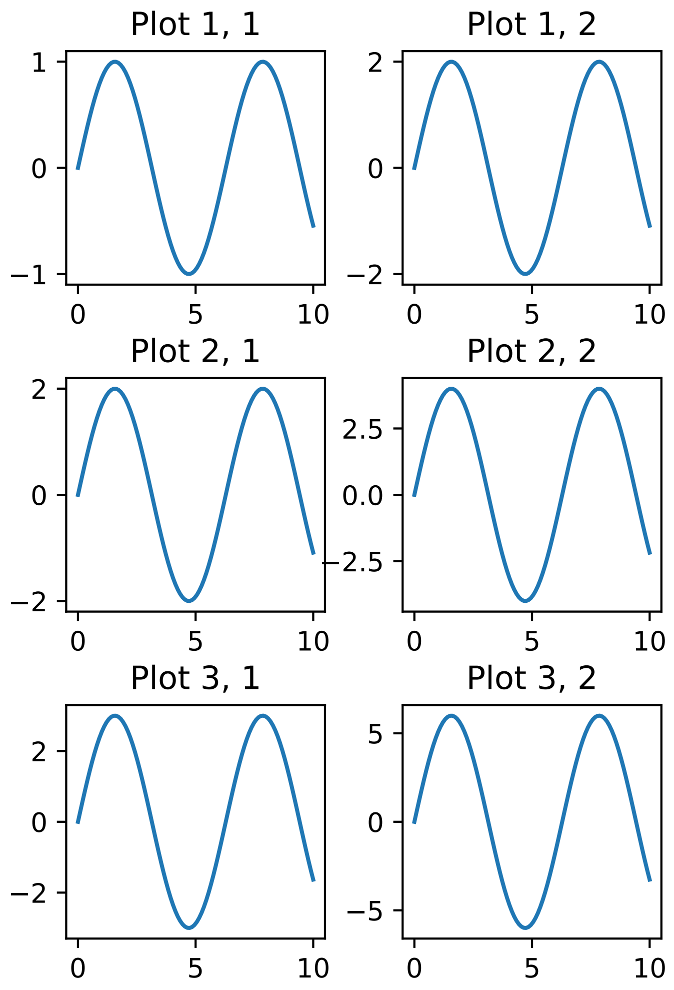
You can also create `subplots` and `gridspecs` to create more complex layouts.

```
from matplotlib.gridspec import GridSpec
from matplotlib import pyplot as plt
import numpy as np
```

```
x = np.linspace(0, 10, 100)
y = np.sin(x)

fig, ax = plt.subplots(3,2, figsize=(3,4), dpi=96, sharex=True, sharey=True, constrained_lay

for i in range(3):
    for j in range(2):
        ax[i,j].plot(x, y*(i+1)*(j+1))

plt.show()
```



```
x = np.linspace(0, 10, 100)
y = np.sin(x)

fig = plt.figure(figsize=(4, 6), dpi=100)
gs = GridSpec(3, 2, figure=fig, hspace=0.4, wspace=0.3)

for i in range(3):
    for j in range(2):
        ax = fig.add_subplot(gs[i, j])
        ax.plot(x, y*(i+1)*(j+1))
        ax.set_title(f'Plot {i+1}, {j+1}')

plt.show()
```



Exercises

Download it locally and try to solve the exercises.

Simple Plot Example

Or open it in `Google Colab`:

Simple Plot Example

15 Simple Plot Types

Difficulty level:

Types of Plots

There exists are lot of different visualization techniques.

How you should visualize the data depends on the data and the question you want to answer.

- **Distributions** can be visualized with a boxplot, a *violinplot*, a *histogram* or a *density plot*.
- **Relationships** between two variables can be visualized with a *scatterplot*, a *lineplot*, a *regplot* or a *jointplot*.
- **Descriptions** of the data can be visualized with a *barplot*, *network plot* or a *pie chart*.

15.1 Relationship plots

15.1.1 Scatter Plot

Scatter plot shows the relationship of observable over the abscissa *e.g.* time vs temperature as **discrete** function.

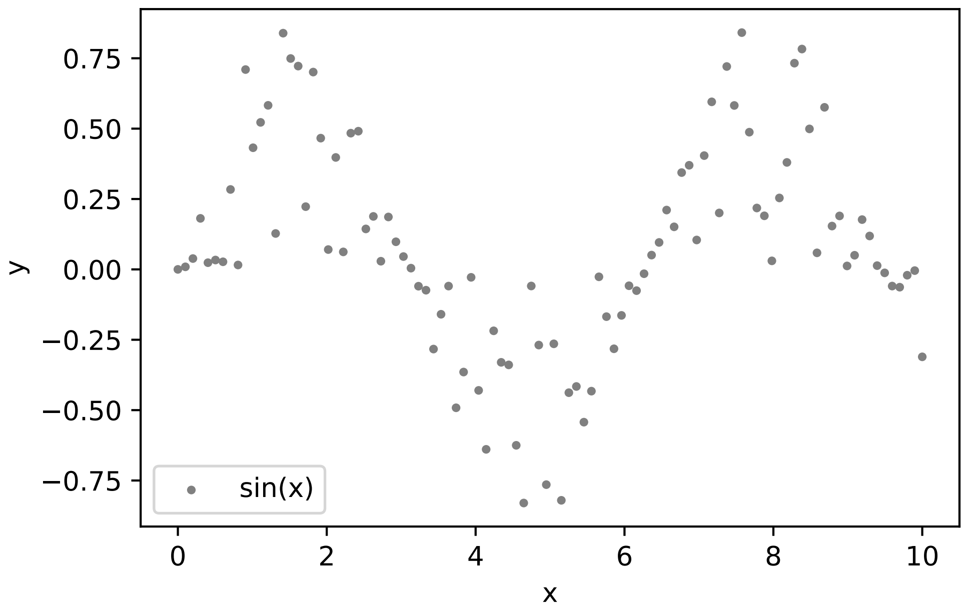
The `scatter()` function creates a scatter plot.

```
plt.scatter()
```

Note

The marker size can be adjusted with the `s` parameter.

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

x = np.linspace(0, 10, 100)
y = np.random.rand(100)*np.sin(x)

plt.scatter(x, y, color='grey', marker='o', label='sin(x)',s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



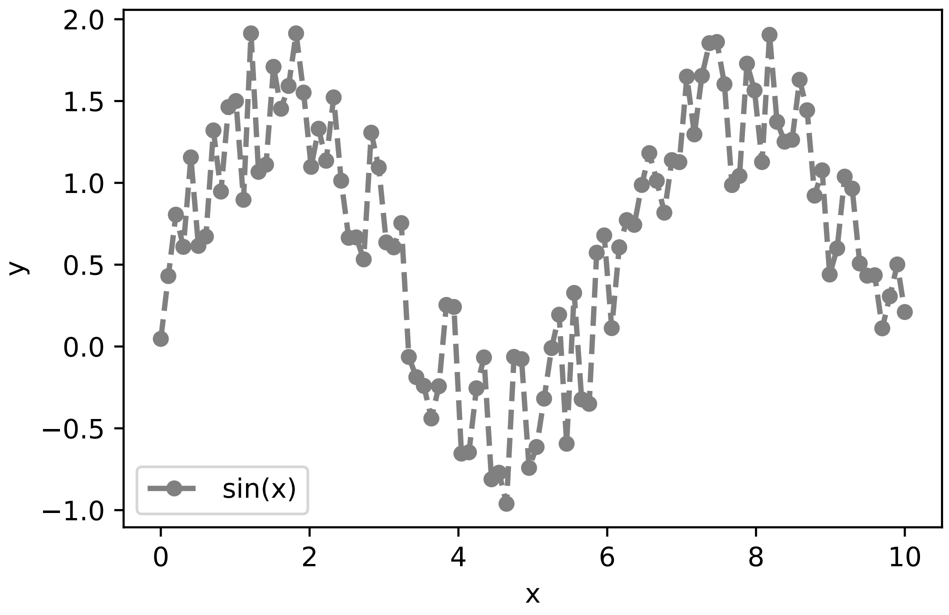
15.1.2 Line Plot

Line plot shows the relationship of observable over the abscissa *e.g.* time vs radioactivity decay as **continuous** function.

The `plot()` function creates a line plot but can also be used to create scatter plots.

> Note
>
> The marker size can be adjusted with the `markersize` parameter.

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

x = np.linspace(0, 10, 100)
y = np.random.rand(100)+np.sin(x)

plt.plot(x, y, color='grey', marker='o', label='sin(x)',markersize=5,linewidth=2,linestyle='-
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

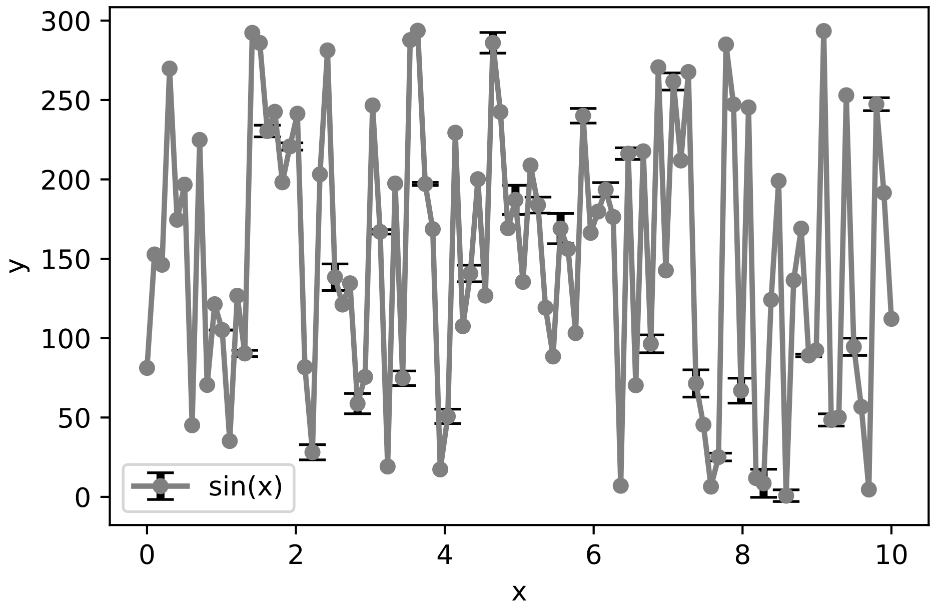


15.1.3 Errorbars

Errorbars can be added to the plot with the `errorbar()` function and the `yerr` parameter.

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

x = np.linspace(0, 10, 100)
y = np.random.rand(100)*298
yerr = np.random.rand(100)*10
```

```
plt.errorbar(x, y, yerr=yerr, color='grey', marker='o', label='sin(x)',markersize=5,linewidtl
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



- The `yerr` parameter can be used to set the error bars for the y-axis.
- The `xerr` parameter can be used to set the error bars for the x-axis.
- The parameter `errorevery` can be used to show only every 3th error bar - starting from the 10th error bar.
- The `capsize` parameter can be used to set the size of the error bar caps.
- The `elinewidth` parameter can be used to set the width of the error bar line.
- The `ecolor` parameter can be used to set the color of the error bar line.

15.2 Distribution Plots

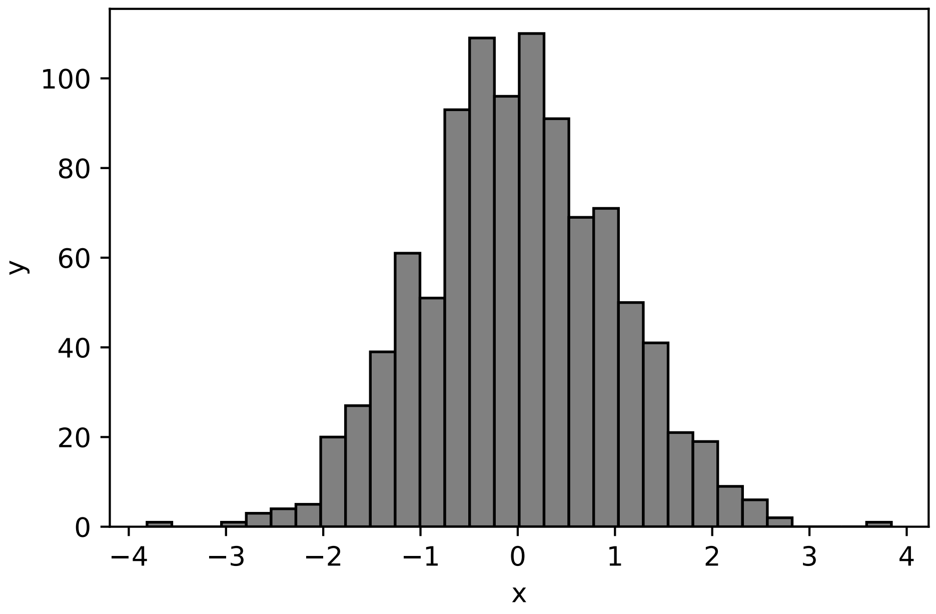
Histogram Plot

[Histogram] (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html) shows the distribution of a single variable *e.g.* count of a mass of individuals in a population. The data is divided into bins and the number of data points in each bin is plotted.

The histogram visualizes the skewness, kurtosis and outliers of the data.

The `hist()` function creates a histogram with `bins` number of bins.

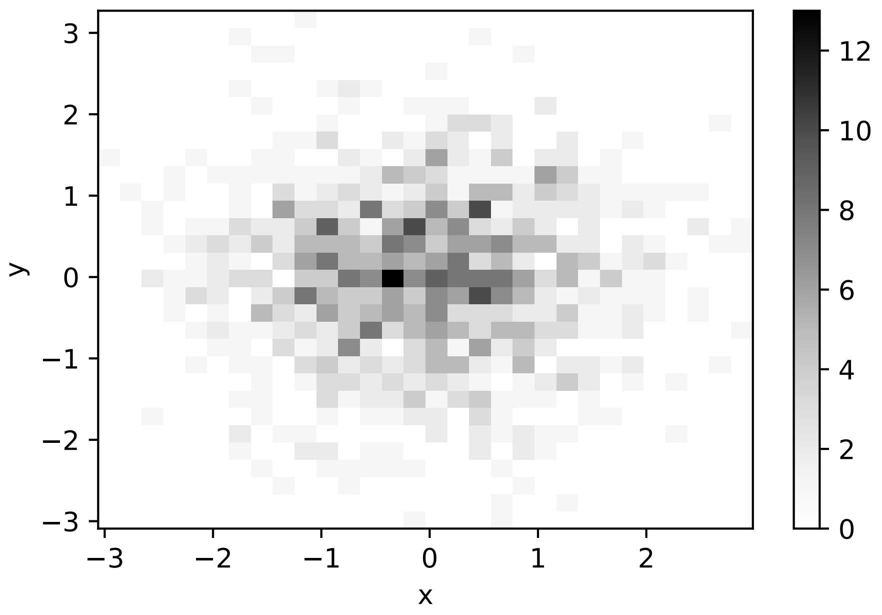
```python
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(1000)

plt.hist(x, bins=30, color='grey', edgecolor='black')

plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



2D histograms can be created with the `hist2d()` function.

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

x = np.random.randn(1000)
y = np.random.randn(1000)
```

```
plt.hist2d(x, y, bins=30, cmap='Greys')
plt.colorbar()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



- The `cmap` parameter is used to set the color map of the histogram.
- The `colorbar()` function adds a color bar to the plot.

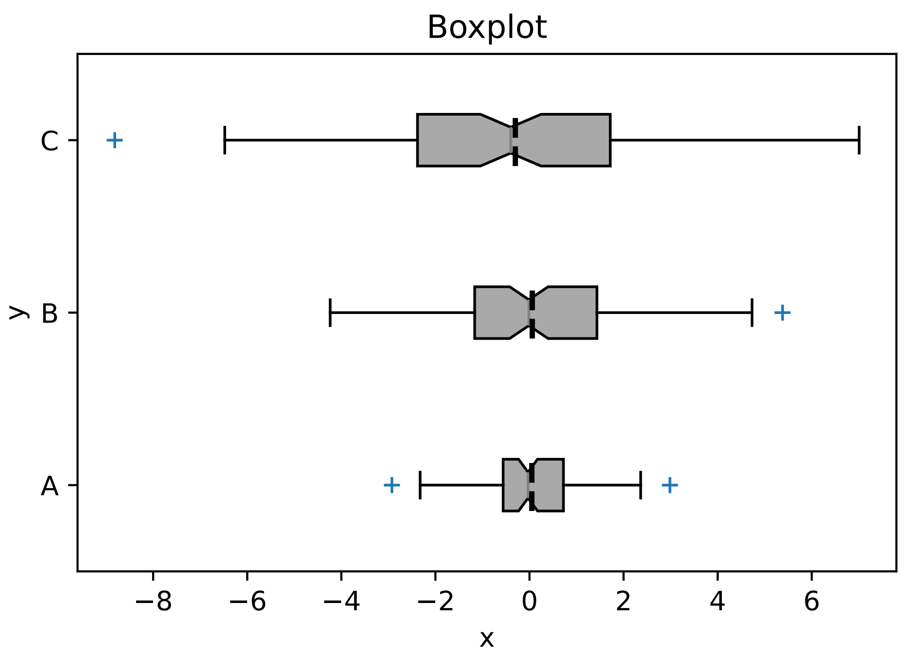
Boxplot

Box plot shows the distribution of a numerical variable for different categories. It shows the minimum, first quartile, median, third quartile and maximum of your data. Outliers can be identified. An example of this *e.g.* is the distribution of cancer cell survival time for different treatment groups.

The `boxplot()` function creates a boxplot.

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

std = 2
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
```

```
plt.boxplot(data, patch_artist=True, notch=True, showmeans=True, meanline=True, showfliers=T
            boxprops=dict(facecolor='darkgrey', color='black'),
            medianprops=dict(color='grey'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            meanprops=dict(color='black', linewidth=2))
plt.yticks([1, 2, 3], ['A', 'B', 'C'])
plt.title('Boxplot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



- The `patch_artist` parameter is used to fill the box with color.
- The `notch` parameter is used to create a notch in the box.
- The `showmeans` parameter is used to show the mean of the data.
- The `meanline` parameter is used to show the mean line.
- The `showfliers` parameter is used to show the outliers of the data.
- The `showbox` parameter is used to show the box of the data.
- The `showcaps` parameter is used to show the caps of the data.
- The `orientation` parameter is used to set the orientation of the boxplot. The default is `vertical`. If you want to create a horizontal boxplot, set it to `horizontal`.

- The `sym` parameter is used to set the symbol of the outliers. The default is `o`. If you want to use a different symbol, set it to `r+` or any other symbol you want to use.
- The `boxprops` parameter is used to set the properties of the box. The `facecolor` parameter is used to set the color of the box. The `color` parameter is used to set the color of the box outline.
- The `medianprops` parameter is used to set the properties of the median line. The `color` parameter is used to set the color of the median line.
- The `whiskerprops` parameter is used to set the properties of the whiskers. The `color` parameter is used to set the color of the whiskers.
- The `meanprops` parameter is used to set the properties of the mean line. The `color` parameter is used to set the color of the mean line.
- The `capprops` parameter is used to set the properties of the caps. The `color` parameter is used to set the color of the caps.

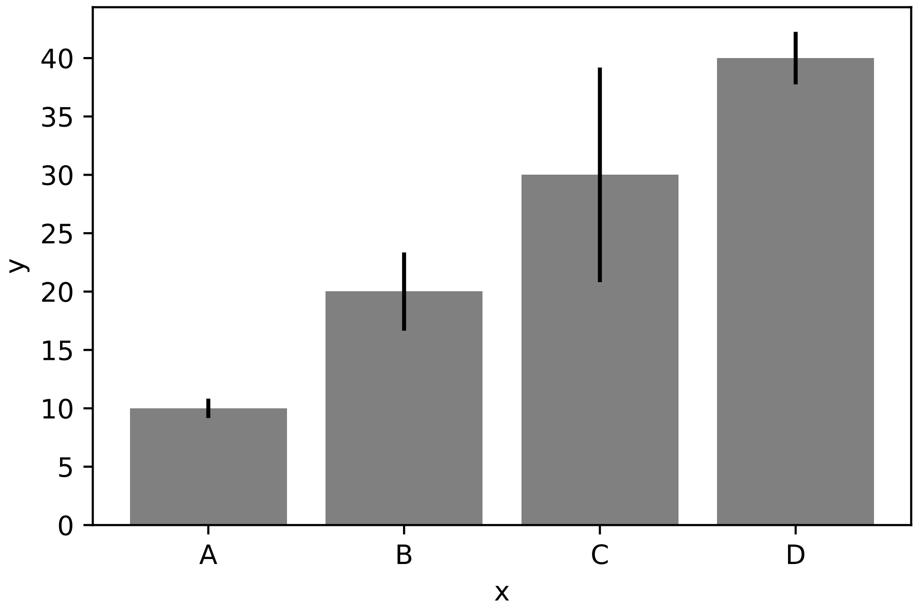
Proportion Plots

Bar Plot

Bar plot shows the relationship of a categorical variable with a numerical variable *e.g.* cancer cell survival time for different treatment groups. The height of the bar is proportional to the value of the investigated variable.

The `bar()` function creates a bar plot.

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

x = ['A', 'B', 'C', 'D']
y = [10, 20, 30, 40]
yerr = np.random.rand(4)*10

plt.bar(x, y,yerr=yerr, color='grey')

plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



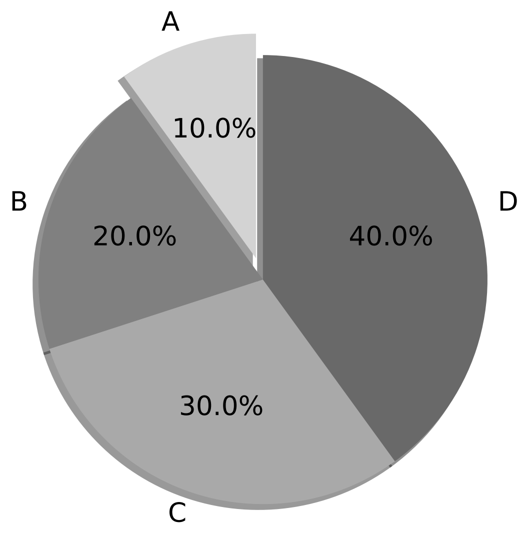
You can draw a star sign to indicate the significance of the data.

Pie Chart

Pie chart show the proportion of different categories in a single variable *e.g.* the content of different amino acids in a protein.

The `pie()` function creates a pie chart.

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

x = [10, 20, 30, 40]
labels = ['A', 'B', 'C', 'D']

plt.pie(x, labels=labels, autopct='%1.1f%%', startangle=90, colors=['lightgrey', 'grey', 'da
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```


A
10.0%
B
20.0%
D
40.0%
30.0%
C

16 Seaborn Plots

Difficulty level:

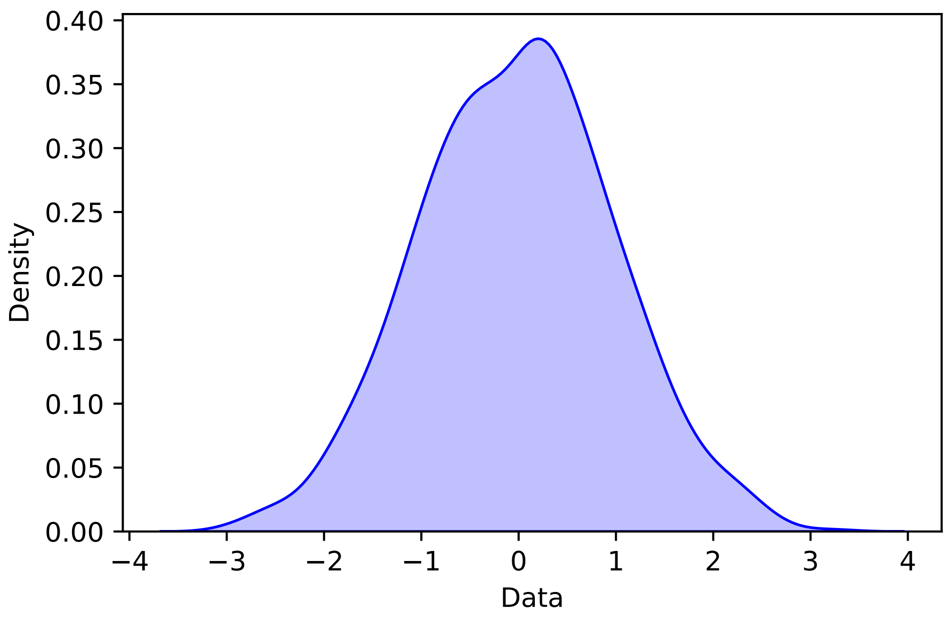
Plotting with Seaborn

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is built on top of Matplotlib and closely integrated with pandas data structures.

KDE Plot

Kernel density plot show the distribution of a single variable *e.g.* distribution of a mass of individuals in a population. The data is smoothed by a kernel density and the density of the data is plotted continuously.

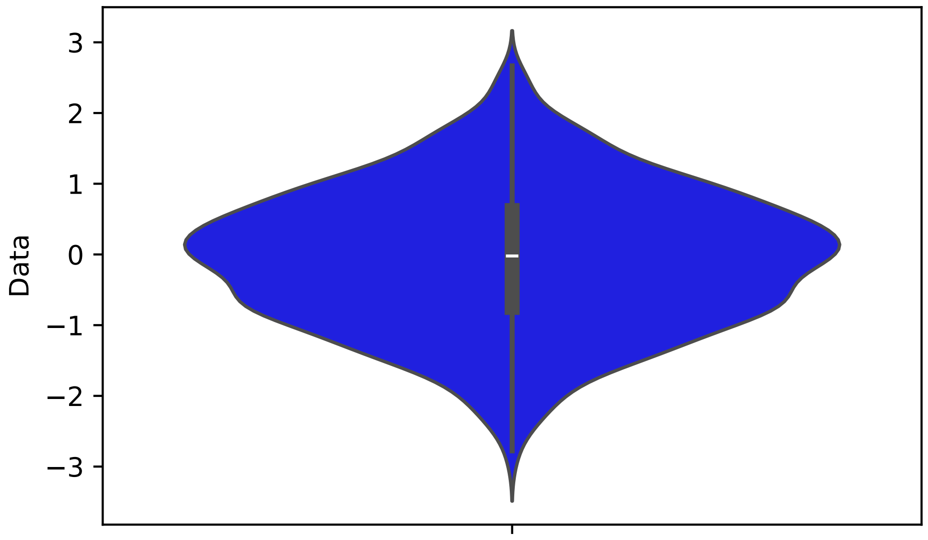
```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create a dataset
data = np.random.normal(loc=0, scale=1, size=1000)
df = pd.DataFrame(data, columns=["Data"])

# Create a KDE plot
sns.kdeplot(data=df["Data"], color="blue", shade=True)
plt.show()
```



Violin Plot

- Violin plot: Show the distribution of a numerical variable for different categories. It is similar to a box plot but it also shows the probability density of the data at different values. An example of this *e.g.* is the distribution of cancer cell survival time for different treatment groups.

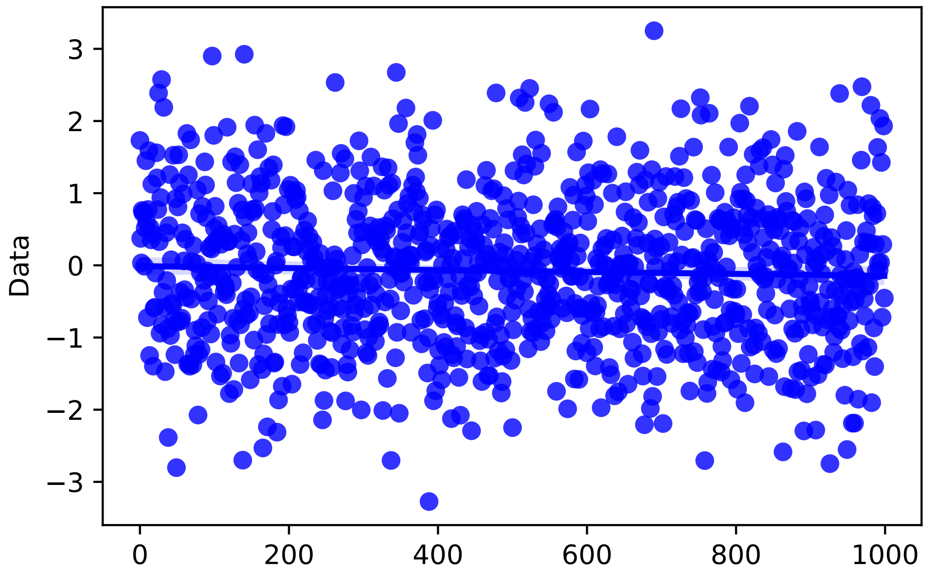
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create a dataset
data = np.random.normal(loc=0, scale=1, size=1000)
df = pd.DataFrame(data, columns=["Data"])

# Create a Violin plot
sns.violinplot(data=df["Data"], color="blue")
plt.show()
```



Correlation plots

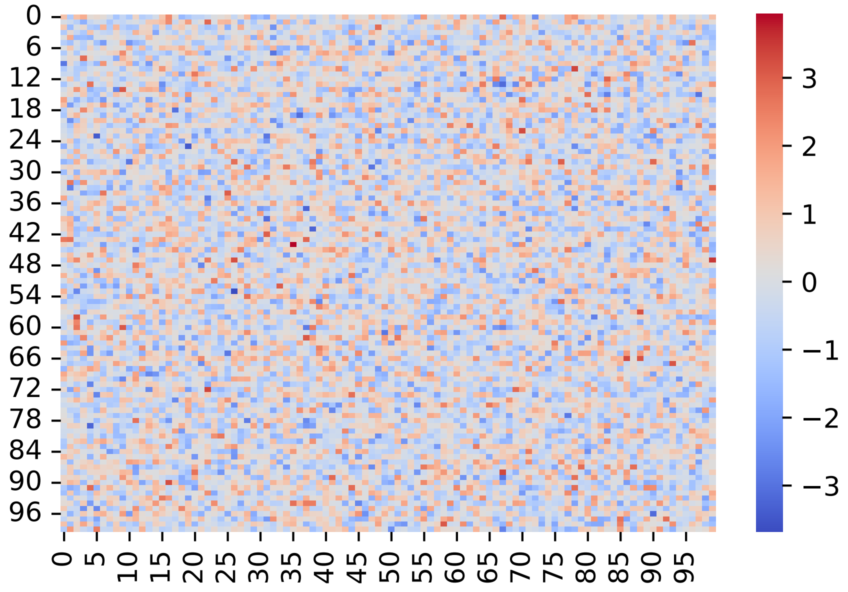
- Regression plot: Show a regression model between two variables *e.g.* the relationship between the concentration and the time.

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

# Create a dataset

data = np.random.normal(loc=0, scale=1, size=1000)
df = pd.DataFrame(data, columns=["Data"])

# Create a Regression plot
sns.regplot(x=np.arange(0, len(df)), y=df["Data"], color="blue")
plt.show()
```



- Heatmap shows the relationship of two categorical variables with a numerical variable *e.g.* cancer cell survival time for different treatment groups and different cancer types. The color of the cell is proportional to the value of the investigated variable.

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create a dataset
data = np.random.normal(loc=0, scale=1, size=(100, 100))
df = pd.DataFrame(data)

# Create a Heatmap
sns.heatmap(data=df, cmap="coolwarm")
plt.show()
```



Additional resources

- Seaborn Gallery
- Seaborn API
- Seaborn Tutorial
- Seaborn Documentation

Part V

Exercises

17 Exercise A 1

18 Exercise:

18.1 Temperature in Synthesis Reactor Part 1

In this exercise, we will repeat the first part of the course.

- We will learn how to read data from a file.
- We will learn how to inspect the data.
- We will learn how to make quickly some plots.

18.1.1 Data

The experiment:

- We have multiple synthesis reactors in which we are running a series of experiments.
- Each reactor has a temperature sensor which monitors the temperature inside of it every minute for one month.
- Our task is to check if the temperatures in the reactors are stable.
- The data for reactor 1 is stored in `Temperatures_1.dat`.
- The data for reactor 2 is stored in `Temperatures_2.dat`.
- The data for reactor 3 is stored in `Temperatures_3.dat`.

18.1.2 Data Path:

```
data_path = "https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exe
```

18.1.3 Task

0. Look at the plain data files. In which format are the data stored?
1. Read the data from the files.
2. Inspect the data. Are there any missing values? Are the data in the correct format? Is the data measured really over one month?
3. Transform the data into days.
4. Look at the distribution of the data. Is there some striking pattern?
5. Make a plot of the temperatures trend over time for reactor 1, 2 and 3.

18.1.4 Questions

- What do you think?
- Are the temperatures stable in the reactors?
- Which reactor has the most stable temperature?

Part VI

Statistical and Exploratory Data Analysis

19 Data Manipulation

Difficulty level:

Data Manipulation

Numpy

```
import numpy as np
```

Array manipulation

Often data has to be manipulated before it can be analyzed. Numpy has many methods for array manipulation.

For example,

- the shape of an array can be changed,
- multiple arrays can be concatenated,
- the elements of an array can be sorted
- non valid values NaN can be removed,
- linear algebra operations can be performed,

etc.

Sort arrays:

```
unsorted_arr = np.array([[3, 1, 5, 2, 4], [5, 2, 0, 8, 1], [3, 2, 9, 4 , 5]])
print("unsortd_arr: \n", unsorted_arr)
sorted_arr = np.sort(unsorted_arr,axis = 0) # sorted along the column
print("sorted_arr along columns: \n" , sorted_arr)
sorted_arr = np.sort(unsorted_arr,axis = 1) # sorted along the row
print("sorted_arr along rows: \n" , sorted_arr)
```

```
unsortd_arr:
 [[3 1 5 2 4]
 [5 2 0 8 1]
 [3 2 9 4 5]]
sorted_arr along columns:
 [[3 1 0 2 1]
 [3 2 5 4 4]
 [5 2 9 8 5]]
sorted_arr along rows:
 [[1 2 3 4 5]
 [0 1 2 5 8]
 [2 3 4 5 9]]
```

Concatenate arrays:

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.concatenate((a, b))
print(c)
```

```
[1 2 3 4 5 6]
```

Reshape arrays:

```
a = np.array([[1, 2],[3, 4],[5, 6]])
b = np.array([[7, 8],[9, 10],[11, 12]])
c = np.vstack((a, b)) # vertical stack
d = np.hstack((a, b)) # horizontal stack
print("vertical stack: \n", c)
print("horizontal stack: \n", d)
print("flatten array: \n", c.flatten())
```

```
vertical stack:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]]
horizontal stack:
 [[ 1  2  7  8]
 [ 3  4  9 10]
 [ 5  6 11 12]]
flatten array:
 [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Linear algebra operations *e.g.*:

- `np.log(a)` returns the **natural logarithm** of an array
- `np.log10(a)` returns the **base 10 logarithm** of an array
- `np.log2(a)` returns the **base 2 logarithm** of an array
- `np.log1p(a)` returns the **natural logarithm** of an array plus one
- `np.exp(a)` returns the **exponential** of an array
- `np.sqrt(a)` returns the **square root** of an array
- `np.abs(a)` returns the **absolute value** of an array
- `np.sum(a)` returns **element-wise sum** of two arrays
- `np.add(a)` returns **sum of all elements** arrays
- `np.subtract(a,b)` returns the **difference** of two arrays
- `np.multiply(a,b)` returns the **product** of two arrays
- `np.divide(a,b)` returns the **division** of two arrays
- `np.dot(a,b)` returns the **dot product** of two arrays
- `np.cross(a,b)` returns the **cross product** of two arrays
- `np.linalg.inv(a)` returns the **inverse of a matrix**
- `np.linalg.det(a)` returns the **determinant of a matrix**
- `np.linalg.eig(a)` returns the **eigenvalues and eigenvectors** of a matrix
- `np.linalg.solve(a,b)` returns the **solution of a linear system of equations**
- `np.convolv(a,b)` returns the **convolution** of two arrays

```python
a = np.array([[1, 2],[3, 4],[5, 6]])
b = np.array([[7, 8],[9, 10],[11, 12]])
c = np.sum(a) # sum of all elements
print("Summ of all elements: ", c)
c = np.add(a,b) # element wise addition
print("Element wise addition: ", c)
```

```
Summ of all elements:  21
Element wise addition:  [[ 8 10]
 [12 14]
 [16 18]]
```

19.0.0.1 Question:

What does the following code snippet do?

```
temperatures = np.array([20, 30, 40, 50, 60, 70, 80, 90, 100])
window = 3
np.convolve(temperatures,np.ones(window) / window,
                            mode='valid')
```

If you are not sure, look in the numpy documentation, ask AI, search in the internet or try it out in a code cell.

! Answer:

The code snippet calculates the moving average of the temperatures with a window size of 3.

Pandas

```
import pandas as pd
data = pd.DataFrame({
    "Name": ["Water", "Sulfuric Acid", "Ethanol", "Acetone", "Ammonia", "Benzene", "Methanol'
    "Formula": ["H2O", "H2SO4", "C2H5OH", "C3H6O", "NH3", "C6H6", "CH3OH", "C3H8O3"],
    "Molecular Weight (g/mol)": [18.015, 98.079, 46.07, 58.08, 17.03, 78.11, 32.04, 92.09],
    "Viscosity (mPa·s)": [1.002, 24.0, 1.2, 0.32, 0.26, 0.65, 0.544, 1412],
    "pH (Acidity)": [7, 1, 7.33, 7, 11.6, 7, 7.4, 5.5],
    "Chemical Type": ["Inorganic", "Acid", "Alcohol", "Ketone", "Base", "Aromatic Hydrocarbo
    "Concentration (M)": [55.5, 18.0, 1.0, 0.8, 0.5, 0.1, 1.5, 1.2]})
```

```
print("head of data: \n",data.head()) # print the first 5 rows
print("\n")
print("Number of data set: \n", data.shape[0]) # number of data set
print("\n")
print("Column Name: \n", data["Name"]) # print the column "Name"
print("\n")
print("2. Row: \n", data.iloc[1]) # print the second row
```

```
head of data:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone   C3H6O                    58.080              0.320
4         Ammonia     NH3                    17.030              0.260

   pH (Acidity) Chemical Type  Concentration (M)
0          7.00     Inorganic               55.5
1          1.00          Acid               18.0
2          7.33       Alcohol                1.0
3          7.00        Ketone                0.8
4         11.60          Base                0.5


Number of data set:
 8


Column Name:
 0            Water
1    Sulfuric Acid
2          Ethanol
3          Acetone
4          Ammonia
5          Benzene
6         Methanol
7         Glycerol
Name: Name, dtype: object


2. Row:
 Name                        Sulfuric Acid
Formula                            H2SO4
Molecular Weight (g/mol)          98.079
Viscosity (mPa·s)                   24.0
pH (Acidity)                         1.0
Chemical Type                       Acid
Concentration (M)                   18.0
Name: 1, dtype: object
```

Select subsets of the data:

Often you need only a specific part of your data with `pandas`it is quite easy to filter the data after specific properties.

```
# filter the rows where concentration is greater than 5
print("Filter the rows where concentration is greater than 5: \n", data[data["Concentration
print("\n")
# get the chemical type with a concentration higher than 5
print("Get the chemical type with a concentration higher than 5: \n",
      data.loc[data["Concentration (M)"]>5,"Chemical Type"])
print("\n")
```

```
Filter the rows where concentration is greater than 5:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1   Sulfuric Acid   H2SO4                    98.079             24.000

   pH (Acidity) Chemical Type  Concentration (M)
0           7.0     Inorganic               55.5
1           1.0          Acid               18.0


Get the chemical type with a concentration higher than 5:
 0    Inorganic
1         Acid
Name: Chemical Type, dtype: object
```

Manipulate data

You can

- `df.copy()`: copy the data
- `df.drop(columns=[...])`: drop columns
- `df.drop(index=[...])`: drop rows
- `df.drop_duplicates()`: drop duplicates
- `df.fillna(...)`: fill NaNs
- `df.dropna()`: drop NaNs
- `df.replace(...)`: replace values
- `pd.concat([df1, df2])`: concatenate data (concatenate along the rows)
- `pd.concat([df1, df2], axis=1)`: concatenate data (concatenate along the columns)

- `pd.merge(df1, df2)`: merge data in the sense of inner, outer, left, right join, see pandas documentation
- `df.agg(...)`: aggregate the data
- `df.transform(...)`: transform the data
- `df.value_counts()`: count the values
- `df.apply(...)`: apply a function to the data
- `df.groupby(...)`: group the data and perform operations on the groups
- `df.sort_values(...)`: sort the data
- `df.shift(...)`: shift the data
- `df.diff(...)`: get the difference between the data
- `df.ptc_shift(...)`: get the percentage change between the data
- `df.to_numpy()`: convert the dataframe to a numpy array
- `pd.DataFrame(arr)`: convert the numpy array to a dataframe etc.

Copy and concatenate data:

```
data2 = data.copy()
data2["Concentration 2 (M)"] = data2["Concentration (M)"] + 10 # increase the concentration 
print("data2: \n" , data2)


data3 = pd.concat([data, data2]) # concatenate the dataframes
print("concated data: \n", data3)
```

```
data2:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2        Ethanol  C2H5OH                    46.070              1.200
3        Acetone   C3H6O                    58.080              0.320
4        Ammonia     NH3                    17.030              0.260
5        Benzene    C6H6                    78.110              0.650
6       Methanol   CH3OH                    32.040              0.544
7       Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M)  Concentration 2 (M)
0          7.00             Inorganic               55.5                 65.5
1          1.00                  Acid               18.0                 28.0
2          7.33               Alcohol                1.0                 11.0
3          7.00                Ketone                0.8                 10.8
```

```
4         11.60                Base               0.5               10.5
5          7.00  Aromatic Hydrocarbon               0.1               10.1
6          7.40             Alcohol               1.5               11.5
7          5.50              Polyol               1.2               11.2
concated data:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone   C3H6O                    58.080              0.320
4         Ammonia     NH3                    17.030              0.260
5         Benzene    C6H6                    78.110              0.650
6        Methanol   CH3OH                    32.040              0.544
7        Glycerol  C3H8O3                    92.090           1412.000
0           Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone   C3H6O                    58.080              0.320
4         Ammonia     NH3                    17.030              0.260
5         Benzene    C6H6                    78.110              0.650
6        Methanol   CH3OH                    32.040              0.544
7        Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M)  Concentration 2 (M)
0          7.00             Inorganic               55.5                  NaN
1          1.00                  Acid               18.0                  NaN
2          7.33               Alcohol                1.0                  NaN
3          7.00                Ketone                0.8                  NaN
4         11.60                  Base                0.5                  NaN
5          7.00  Aromatic Hydrocarbon                0.1                  NaN
6          7.40               Alcohol                1.5                  NaN
7          5.50                Polyol                1.2                  NaN
0          7.00             Inorganic               55.5                 65.5
1          1.00                  Acid               18.0                 28.0
2          7.33               Alcohol                1.0                 11.0
3          7.00                Ketone                0.8                 10.8
4         11.60                  Base                0.5                 10.5
5          7.00  Aromatic Hydrocarbon                0.1                 10.1
6          7.40               Alcohol                1.5                 11.5
7          5.50                Polyol                1.2                 11.2
```

```
 # create a Panda Series
acidity = pd.Series(["basic", "acid", "basic", "neutral",
                    "acid", "acid", "basic", "basic"], name="Aciditiy Type")
print("acidity: \n",acidity)
# concatenate the dataframes
data4 = pd.concat([data3, acidity], axis=1)
print("concated data: \n", data4)
```

```
acidity:
 0      basic
1       acid
2      basic
3    neutral
4       acid
5       acid
6      basic
7      basic
Name: Aciditiy Type, dtype: object
concated data:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2        Ethanol  C2H5OH                    46.070              1.200
3        Acetone   C3H6O                    58.080              0.320
4        Ammonia     NH3                    17.030              0.260
5        Benzene    C6H6                    78.110              0.650
6       Methanol   CH3OH                    32.040              0.544
7       Glycerol  C3H8O3                    92.090           1412.000
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2        Ethanol  C2H5OH                    46.070              1.200
3        Acetone   C3H6O                    58.080              0.320
4        Ammonia     NH3                    17.030              0.260
5        Benzene    C6H6                    78.110              0.650
6       Methanol   CH3OH                    32.040              0.544
7       Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M)  Concentration 2 (M)  \
0          7.00             Inorganic               55.5                  NaN
1          1.00                  Acid               18.0                  NaN
2          7.33               Alcohol                1.0                  NaN
3          7.00                Ketone                0.8                  NaN
```

```
4         11.60                   Base             0.5              NaN
5          7.00  Aromatic Hydrocarbon             0.1              NaN
6          7.40                Alcohol             1.5              NaN
7          5.50                 Polyol             1.2              NaN
0          7.00              Inorganic            55.5             65.5
1          1.00                   Acid            18.0             28.0
2          7.33                Alcohol             1.0             11.0
3          7.00                 Ketone             0.8             10.8
4         11.60                   Base             0.5             10.5
5          7.00  Aromatic Hydrocarbon             0.1             10.1
6          7.40                Alcohol             1.5             11.5
7          5.50                 Polyol             1.2             11.2

  Aciditiy Type
0         basic
1          acid
2         basic
3       neutral
4          acid
5          acid
6         basic
7         basic
0         basic
1          acid
2         basic
3       neutral
4          acid
5          acid
6         basic
7         basic
```

Drop columns and rows:

```python
data4 = data4.drop(columns=["Concentration 2 (M)"]) # drop the column "Concentration 2 (M)"
print("data4: \n", data4)

data5 = data4.drop(index=[0, 1]) # drop the rows with index 0 and 1
print("data5: \n", data5)

data6 = data5.drop(columns=["Aciditiy Type"]) # drop the column "Aciditiy Type"
print("data6: \n", data6)
```

```
data4:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1  Sulfuric Acid    H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone    C3H6O                   58.080              0.320
4         Ammonia     NH3                    17.030              0.260
5         Benzene    C6H6                    78.110              0.650
6        Methanol   CH3OH                    32.040              0.544
7        Glycerol  C3H8O3                    92.090           1412.000
0           Water     H2O                    18.015              1.002
1  Sulfuric Acid    H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone    C3H6O                   58.080              0.320
4         Ammonia     NH3                    17.030              0.260
5         Benzene    C6H6                    78.110              0.650
6        Methanol   CH3OH                    32.040              0.544
7        Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
0          7.00             Inorganic               55.5         basic
1          1.00                  Acid               18.0          acid
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
0          7.00             Inorganic               55.5         basic
1          1.00                  Acid               18.0          acid
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
data5:
        Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
2   Ethanol  C2H5OH                     46.07              1.200
3   Acetone   C3H6O                     58.08              0.320
4   Ammonia     NH3                     17.03              0.260
5   Benzene    C6H6                     78.11              0.650
6  Methanol   CH3OH                     32.04              0.544
```

```
7  Glycerol  C3H8O3                        92.09          1412.000
2   Ethanol  C2H5OH                        46.07             1.200
3   Acetone   C3H6O                        58.08             0.320
4   Ammonia     NH3                        17.03             0.260
5   Benzene    C6H6                        78.11             0.650
6  Methanol   CH3OH                        32.04             0.544
7  Glycerol  C3H8O3                        92.09          1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
data6:
       Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
2   Ethanol  C2H5OH                        46.07             1.200
3   Acetone   C3H6O                        58.08             0.320
4   Ammonia     NH3                        17.03             0.260
5   Benzene    C6H6                        78.11             0.650
6  Methanol   CH3OH                        32.04             0.544
7  Glycerol  C3H8O3                        92.09          1412.000
2   Ethanol  C2H5OH                        46.07             1.200
3   Acetone   C3H6O                        58.08             0.320
4   Ammonia     NH3                        17.03             0.260
5   Benzene    C6H6                        78.11             0.650
6  Methanol   CH3OH                        32.04             0.544
7  Glycerol  C3H8O3                        92.09          1412.000

   pH (Acidity)         Chemical Type  Concentration (M)
2          7.33               Alcohol                1.0
3          7.00                Ketone                0.8
4         11.60                  Base                0.5
5          7.00  Aromatic Hydrocarbon                0.1
6          7.40               Alcohol                1.5
7          5.50                Polyol                1.2
```

```
2          7.33              Alcohol               1.0
3          7.00               Ketone               0.8
4         11.60                 Base               0.5
5          7.00  Aromatic Hydrocarbon               0.1
6          7.40              Alcohol               1.5
7          5.50               Polyol               1.2
```

Drop duplicates, fill NaNs and drop NaNs

```python
data5 = data4.drop_duplicates() # drop the duplicates
print("data5: \n", data5)

data6 = data5.fillna(0) # fill the NaNs with 0
print("data6: \n", data6)

data7 = data5.dropna() # drop the NaNs
print("data7: \n", data7)
```

```
data5:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2        Ethanol  C2H5OH                    46.070              1.200
3        Acetone   C3H6O                    58.080              0.320
4        Ammonia     NH3                    17.030              0.260
5        Benzene    C6H6                    78.110              0.650
6       Methanol   CH3OH                    32.040              0.544
7       Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
0          7.00             Inorganic               55.5         basic
1          1.00                  Acid               18.0          acid
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
data6:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                      18.015               1.002
1  Sulfuric Acid   H2SO4                      98.079              24.000
2        Ethanol  C2H5OH                      46.070               1.200
3        Acetone   C3H6O                      58.080               0.320
4        Ammonia     NH3                      17.030               0.260
5        Benzene    C6H6                      78.110               0.650
6       Methanol   CH3OH                      32.040               0.544
7       Glycerol  C3H8O3                      92.090            1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
0          7.00             Inorganic               55.5         basic
1          1.00                  Acid               18.0          acid
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
data7:
            Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                      18.015               1.002
1  Sulfuric Acid   H2SO4                      98.079              24.000
2        Ethanol  C2H5OH                      46.070               1.200
3        Acetone   C3H6O                      58.080               0.320
4        Ammonia     NH3                      17.030               0.260
5        Benzene    C6H6                      78.110               0.650
6       Methanol   CH3OH                      32.040               0.544
7       Glycerol  C3H8O3                      92.090            1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
0          7.00             Inorganic               55.5         basic
1          1.00                  Acid               18.0          acid
2          7.33               Alcohol                1.0         basic
3          7.00                Ketone                0.8       neutral
4         11.60                  Base                0.5          acid
5          7.00  Aromatic Hydrocarbon                0.1          acid
6          7.40               Alcohol                1.5         basic
7          5.50                Polyol                1.2         basic
```

Replace values:

```
# replace the values
data6["Acidity Type"].replace({"acid": "Acidic", "basic": "Basic", "neutral": "Neutral"}, i
print("data6: \n", data6)
```

```
data6:
              Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000
2        Ethanol  C2H5OH                    46.070              1.200
3        Acetone   C3H6O                    58.080              0.320
4        Ammonia     NH3                    17.030              0.260
5        Benzene    C6H6                    78.110              0.650
6       Methanol   CH3OH                    32.040              0.544
7       Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Acidity Type
0          7.00             Inorganic               55.5         Basic
1          1.00                  Acid               18.0        Acidic
2          7.33               Alcohol                1.0         Basic
3          7.00                Ketone                0.8       Neutral
4         11.60                  Base                0.5        Acidic
5          7.00  Aromatic Hydrocarbon                0.1        Acidic
6          7.40               Alcohol                1.5         Basic
7          5.50                Polyol                1.2         Basic
```

Count the values

```
# count the chemicals with different concentration
print("Count the chemicals with the different concentrations \n", data["Concentration (M)"].
```

```
Count the chemicals with the different concentrations
 Concentration (M)
55.5    1
18.0    1
1.0     1
0.8     1
0.5     1
```

```
0.1     1
1.5     1
1.2     1
Name: count, dtype: int64
```

Appply a function to the data

```
print("Apply a function to the data: \n", data["Concentration (M)"].apply(lambda x: x*2))
```

```
Apply a function to the data:
 0    111.0
1     36.0
2      2.0
3      1.6
4      1.0
5      0.2
6      3.0
7      2.4
Name: Concentration (M), dtype: float64
```

Transform data

```
# transform the data
data6["Concentration (M)"] = data6["Concentration (M)"].transform(lambda x: x*2)
print("data6: \n", data6)
```

```
data6:
              Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1   Sulfuric Acid   H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone   C3H6O                    58.080              0.320
4         Ammonia     NH3                    17.030              0.260
5         Benzene    C6H6                    78.110              0.650
6        Methanol   CH3OH                    32.040              0.544
7        Glycerol  C3H8O3                    92.090           1412.000

   pH (Acidity)         Chemical Type  Concentration (M) Aciditiy Type
```

```
0          7.00             Inorganic                 111.0        Basic
1          1.00                  Acid                  36.0       Acidic
2          7.33               Alcohol                   2.0        Basic
3          7.00                Ketone                   1.6      Neutral
4         11.60                  Base                   1.0       Acidic
5          7.00  Aromatic Hydrocarbon                   0.2       Acidic
6          7.40               Alcohol                   3.0        Basic
7          5.50                Polyol                   2.4        Basic
```

19.0.1 Sort data

```
print("sorted after concentration: ",data.sort_values(by="Concentration (M)", ascending=True
```

```
sorted after concentration:               Name Formula  Molecular Weight (g/mol)  Viscosity (
5        Benzene    C6H6                    78.110              0.650
4        Ammonia     NH3                    17.030              0.260
3        Acetone   C3H6O                    58.080              0.320
2        Ethanol  C2H5OH                    46.070              1.200
7       Glycerol  C3H8O3                    92.090           1412.000
6       Methanol   CH3OH                    32.040              0.544
1  Sulfuric Acid   H2SO4                    98.079             24.000
0          Water     H2O                    18.015              1.002

   pH (Acidity)         Chemical Type  Concentration (M)
5          7.00  Aromatic Hydrocarbon                0.1
4         11.60                  Base                0.5
3          7.00                Ketone                0.8
2          7.33               Alcohol                1.0
7          5.50                Polyol                1.2
6          7.40               Alcohol                1.5
1          1.00                  Acid               18.0
0          7.00             Inorganic               55.5
```

19.0.2 Group data

In pandas you can group data by a specific column and perform operations on the groups.

```
# sum of concentration by filter
print("sum of concentration: \n", data["Concentration (M)"].sum())
# sum of concentration group by type
print("Sum of concentration group by type: \n", data.groupby("Chemical Type")["Concentration
```

```
sum of concentration:
 78.6
Sum of concentration group by type:
 Chemical Type
Acid                    18.0
Alcohol                  2.5
Aromatic Hydrocarbon     0.1
Base                     0.5
Inorganic               55.5
Ketone                   0.8
Polyol                   1.2
Name: Concentration (M), dtype: float64
```

Transform Pandas DataFrame to Numpy Array and viceversa

```
print("Names: ", data5.columns)
df = data5.copy()
print(len(df))
# convert the dataframe to a numpy array
arr = df.to_numpy()
print("arr: \n", arr)
# convert the numpy array to a dataframe
df2 = pd.DataFrame(arr, columns=["Name", "Formula", "Molecular Weight (g/mol)","Viscosity (m
print("df2: \n", df2)
```

```
Names:  Index(['Name', 'Formula', 'Molecular Weight (g/mol)', 'Viscosity (mPa·s)',
       'pH (Acidity)', 'Chemical Type', 'Concentration (M)', 'Acidtiy Type'],
      dtype='object')
8
arr:
 [['Water' 'H2O' 18.015 1.002 7.0 'Inorganic' 55.5 'basic']
 ['Sulfuric Acid' 'H2SO4' 98.079 24.0 1.0 'Acid' 18.0 'acid']
 ['Ethanol' 'C2H5OH' 46.07 1.2 7.33 'Alcohol' 1.0 'basic']
 ['Acetone' 'C3H6O' 58.08 0.32 7.0 'Ketone' 0.8 'neutral']
 ['Ammonia' 'NH3' 17.03 0.26 11.6 'Base' 0.5 'acid']
 ['Benzene' 'C6H6' 78.11 0.65 7.0 'Aromatic Hydrocarbon' 0.1 'acid']
 ['Methanol' 'CH3OH' 32.04 0.544 7.4 'Alcohol' 1.5 'basic']
 ['Glycerol' 'C3H8O3' 92.09 1412.0 5.5 'Polyol' 1.2 'basic']]
df2:
             Name Formula Molecular Weight (g/mol) Viscosity (mPa·s)  \
0           Water     H2O                   18.015             1.002
1   Sulfuric Acid   H2SO4                   98.079              24.0
2         Ethanol  C2H5OH                    46.07               1.2
3         Acetone   C3H6O                    58.08              0.32
4         Ammonia     NH3                    17.03              0.26
5         Benzene    C6H6                    78.11              0.65
6        Methanol   CH3OH                    32.04             0.544
7        Glycerol  C3H8O3                    92.09            1412.0

  pH (Acidity)         Chemical Type Concentration (M) Aciditiy Type
0          7.0             Inorganic              55.5         basic
1          1.0                  Acid              18.0          acid
2         7.33               Alcohol               1.0         basic
3          7.0                Ketone               0.8       neutral
4         11.6                  Base               0.5          acid
5          7.0  Aromatic Hydrocarbon               0.1          acid
6          7.4               Alcohol               1.5         basic
7          5.5                Polyol               1.2         basic
```

20 Descriptive Statistics and Analysis

Difficulty level:

Descriptive Statistics and Analysis

Descriptive statistics summarize and analyze datasets by providing measures of central tendency, dispersion, and correlation. These statistics help in understanding the underlying patterns in chemical and materials science data.

- **Measures of Central Tendency:** Mean, median, and mode provide insights into the average or most common values in the dataset.
- **Variability:** Variance, standard deviation, and interquartile range (IQR) measure how spread out the data is.
- **Correlation & Covariance:** Determine relationships between two or more variables, useful for identifying dependencies in experimental data.
- **Summary Statistics:** Pandas provides built-in functions to calculate descriptive statistics for dataframes, including mean, median, standard deviation, and more. Further you can get a fast overview of the data using the `describe()` function.

Numpy

```python
import numpy as np
```

- `np.mean(a)` returns the **mean** of the elements of an array
- `np.average(a)` returns the **weighted average** of the elements of an array
- `np.median(a)` returns the **median** of the elements of an array
- `np.max(a)` returns the **maximum** of the elements of an array
- `np.min(a)` returns **the minimum** of the elements of an array
- `np.std(a)` returns **the standard** deviation of the elements of an array
- `np.var(a)` returns **the variance** of the elements of an array

- `np.covar(a)` returns the **covariance** of the elements of an array
- `np.percentile(a,p)` returns the **p-th percentile** of the elements of an array
- `np.quantile(a,q)` returns the **q-th quantile** of the elements of an array
- `np.corrcoef(a)` returns the **correlation coefficient** of the elements of an array
- `np.corrcoef(a,b)` returns the **correlation coefficient** of two arrays
- `np.histogram(a, [,bins,range,density,weights])` returns the **histogram** of the elements of an array
- `np.histogram2d(a,b, [,bins,range,density,weights])` returns the **2D histogram** of the elements of an array
- `np.histogramdd(a, [,bins,range,density,weights])` returns the **nD histogram** of the elements of an array

```
arr = np.array([1, 2, 3, 4, 5, 4, 7, 8, 9])
print(arr)
print("Mean: ", np.mean(arr))
print("Weighted average: ", np.average(arr, weights=[1, 2, 3, 4, 5, 4, 3, 2, 1]))
print("Median: ", np.median(arr))
print("Standard deviation: ", np.std(arr))
print("Variance: ", np.var(arr))
print("Correlation coefficient: ", np.corrcoef(arr))
print("Percentile: ", np.percentile(arr, 0.25))
print("Quantile: ", np.quantile(arr, 0.75))
print("IQR: ", np.percentile(arr, 0.75) - np.percentile(arr, 0.25))
```

```
[1 2 3 4 5 4 7 8 9]
Mean:  4.777777777777778
Weighted average:  4.68
Median:  4.0
Standard deviation:  2.57240820062005
Variance:  6.617283950617284
Correlation coefficient:  1.0
Percentile:  1.02
Quantile:  7.0
IQR:  0.040000000000000036
```

- `np.argmin(a)` returns the **index of the minimum** element of an array
- `np.argmax(a)` returns the **index of the maximum** element of an array
- `np.where(a)` returns the **indices of the elements** of an array that are non-zero
- `np.argwhere(a)` returns the **indices of the elements** of an array that are non-zero

- `np.nonzero(a)` returns the **indices of the elements** of an array that are non-zero
- `np.searchsorted(a,v)` returns the **index of the first element** of an array **that is greater than or equal to a value**
- `np.extract(condition,a)` returns the **elements of an array that satisfy a condition**

```
arr = np.array([1, 2, 3, 4, 5, 4, 7, 8, 9])
print(arr)
print("Index of the maximum element: ", np.argmax(arr))
print("Index of the minimum element: ", np.argmin(arr))
print("Maximum element: ", np.max(arr))
print("Minimum element: ", np.min(arr))
print("Find the index of element \"4\": ", np.where(arr == 4))
print("Find the index of element \">4\": ", np.argwhere(arr > 4),
       " and the elements are: ", arr[np.where(arr > 4)])
```

```
[1 2 3 4 5 4 7 8 9]
Index of the maximum element:  8
Index of the minimum element:  0
Maximum element:  9
Minimum element:  1
Find the index of element "4":  (array([3, 5]),)
Find the index of element ">4":  [[4]
 [6]
 [7]
 [8]]  and the elements are:  [5 7 8 9]
```

Pandas

```
import pandas as pd
data = pd.DataFrame({
    "Name": ["Water", "Sulfuric Acid", "Ethanol", "Acetone", "Ammonia", "Benzene", "Methanol"
    "Formula": ["H2O", "H2SO4", "C2H5OH", "C3H6O", "NH3", "C6H6", "CH3OH", "C3H8O3"],
    "Molecular Weight (g/mol)": [18.015, 98.079, 46.07, 58.08, 17.03, 78.11, 32.04, 92.09],
    "Viscosity (mPa·s)": [1.002, 24.0, 1.2, 0.32, 0.26, 0.65, 0.544, 1412],
    "pH (Acidity)": [7, 1, 7.33, 7, 11.6, 7, 7.4, 5.5],
    "Chemical Type": ["Inorganic", "Acid", "Alcohol", "Ketone", "Base", "Aromatic Hydrocarbo
    "Concentration (M)": [55.5, 18.0, 1.0, 0.8, 0.5, 0.1, 1.5, 1.2]})
```

Repeation how to get an overview of the data

- `data.head()`: Returns the first 5 rows of the dataframe

```
print("head of data: \n",data.head()) # print the first 5 rows
```

```
head of data:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0           Water     H2O                    18.015              1.002
1   Sulfuric Acid   H2SO4                    98.079             24.000
2         Ethanol  C2H5OH                    46.070              1.200
3         Acetone    C3H6O                    58.080              0.320
4         Ammonia     NH3                    17.030              0.260

   pH (Acidity) Chemical Type  Concentration (M)
0          7.00     Inorganic               55.5
1          1.00          Acid               18.0
2          7.33       Alcohol                1.0
3          7.00        Ketone                0.8
4         11.60          Base                0.5
```

- `data.shape[0]`: Returns the number of rows in the dataframe

```
print("Number of data set: \n", data.shape[0]) # number of data set
```

```
Number of data set:
 8
```

- `data.shape[1]`: Returns the number of columns in the dataframe

```
print("Number of data set: \n", data.shape[0]) # number of data set
```

```
Number of data set:
 8
```

- `data["Name"]`: Returns the column "Name" of the dataframe

```
print("Name of chemicals: \n", data["Name"]) # print the column "Name"
```

```
Name of chemicals:
 0            Water
1    Sulfuric Acid
2          Ethanol
3          Acetone
4          Ammonia
5          Benzene
6         Methanol
7         Glycerol
Name: Name, dtype: object
```

- `data.iloc[1]`: Returns the second row of the dataframe

```
print("2. Row: \n", data.iloc[1]) # print the second row
```

```
2. Row:
 Name                          Sulfuric Acid
Formula                               H2SO4
Molecular Weight (g/mol)             98.079
Viscosity (mPa·s)                      24.0
pH (Acidity)                            1.0
Chemical Type                          Acid
Concentration (M)                      18.0
Name: 1, dtype: object
```

- `data[data["Concentration (M)"] > 5]`: Returns the rows where the concentration is greater than 5
- `data.loc[data["Concentration (M)"]>5,"Chemical Type"]`: Returns the chemical type with a concentration higher than 5

```
print("Filter the rows where concentration is greater than 5: \n", data[data["Concentration
print("Get the chemical type with a concentration higher than 5: \n",
      data.loc[data["Concentration (M)"]>5,"Chemical Type"])
```

```
Filter the rows where concentration is greater than 5:
             Name Formula  Molecular Weight (g/mol)  Viscosity (mPa·s)  \
0          Water     H2O                    18.015              1.002
1  Sulfuric Acid   H2SO4                    98.079             24.000

   pH (Acidity) Chemical Type  Concentration (M)
0           7.0     Inorganic               55.5
1           1.0          Acid               18.0
```

```
Get the chemical type with a concentration higher than 5:
 0    Inorganic
1        Acid
Name: Chemical Type, dtype: object
```

Descriptive Statistics

- `data.info()`: Returns the information of the dataframe

```
print("Information of data: \n", data.info()) # information of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 7 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Name                       8 non-null      object
 1   Formula                    8 non-null      object
 2   Molecular Weight (g/mol)   8 non-null      float64
 3   Viscosity (mPa·s)          8 non-null      float64
 4   pH (Acidity)               8 non-null      float64
 5   Chemical Type              8 non-null      object
 6   Concentration (M)          8 non-null      float64
dtypes: float64(4), object(3)
memory usage: 580.0+ bytes
Information of data:
 None
```

Statistical Summary

- `data.describe()`: Returns the statistical summary of the dataframe

```
print("Statistical summary: \n", data.describe()) # statistical summary
```

```
Statistical summary:
        Molecular Weight (g/mol)  Viscosity (mPa·s)  pH (Acidity)  \
count                  8.000000           8.000000      8.000000
mean                  54.939250         179.997000      6.728750
std                   32.052446         497.871465      2.905421
min                   17.030000           0.260000      1.000000
25%                   28.533750           0.488000      6.625000
```

```
50%                     52.075000            0.826000      7.000000
75%                     81.605000            6.900000      7.347500
max                     98.079000         1412.000000     11.600000

       Concentration (M)
count           8.000000
mean            9.825000
std            19.411318
min             0.100000
25%             0.725000
50%             1.100000
75%             5.625000
max            55.500000
```

Statistical Measures

- `data["Concentration (M)"].mean()`: Returns the mean of the column "Concentration (M)"
- `data["Concentration (M)"].std()`: Returns the standard deviation of the column "Concentration (M)"
- `data["Concentration (M)"].median()`: Returns the median of the column "Concentration (M)"
- `data["Concentration (M)"].max()`: Returns the maximum of the column "Concentration (M)"
- `data["Concentration (M)"].min()`: Returns the minimum of the column "Concentration (M)"
- `data["Concentration (M)"].sum()`: Returns the sum of the column "Concentration (M)"
- `data["Concentration (M)"].mode()`: Returns the mode of the column "Concentration (M)"
- `data["Concentration (M)"].percentile(q)`: Returns the q-th percentile of the column "Concentration (M)"

```
print("mean of concentration: \n", data["Concentration (M)"].mean()) # mean of concentration
print("std of concentration: \n", data["Concentration (M)"].std()) # standard deviation of co
print("median of concentration: \n", data["Concentration (M)"].median()) # median of concentr
print("max of concentration: \n", data["Concentration (M)"].max()) # max of concentration
print("min of concentration: \n", data["Concentration (M)"].min()) # min of concentration
print("sum of concentration: \n", data["Concentration (M)"].sum()) # sum of concentration
print("mode of concentration: \n", data["Concentration (M)"].mode()) # mode of concentration
print("percentile of concentration: \n", data["Concentration (M)"].quantile(0.25)) # 25th pe
print("IQR of concentration: \n", data["Concentration (M)"].quantile(0.75) - data["Concentra
mean of concentration:
 9.825
std of concentration:
 19.41131849499888
median of concentration:
 1.1
max of concentration:
 55.5
min of concentration:
 0.1
sum of concentration:
 78.6
mode of concentration:
 0     0.1
1     0.5
2     0.8
3     1.0
4     1.2
5     1.5
6    18.0
7    55.5
Name: Concentration (M), dtype: float64
percentile of concentration:
 0.7250000000000001
IQR of concentration:
 4.9
```

- `data.groupby("Chemical Type")["Concentration (M)"].mean()`: Returns the mean of the column "Concentration (M)" grouped by "Chemical Type"

```
print("Mean of concentration group by type: \n", data.groupby("Chemical Type")["Concentratio
```

```
Mean of concentration group by type:
 Chemical Type
Acid                    18.00
Alcohol                  1.25
Aromatic Hydrocarbon     0.10
Base                     0.50
Inorganic               55.50
Ketone                   0.80
Polyol                   1.20
Name: Concentration (M), dtype: float64
```

- `data.agg({"Concentration (M)": ["mean", "std", "median", "max", "min", "sum"]})`: Returns the mean, standard deviation, median, maximum, minimum, and sum of the column "Concentration (M)"

```
print("Aggregate the data: \n", data.agg({"Concentration (M)": ["mean", "std", "median", "ma
```

```
Aggregate the data:
         Concentration (M)
mean              9.825000
std              19.411318
median            1.100000
max              55.500000
min               0.100000
sum              78.600000
```

Correlation and Covariance

- `data.corr()`: Returns the correlation matrix of the dataframe
- `data.cov()`: Returns the covariance matrix of the dataframe

```
print("Correlation matrix: \n", data["Concentration (M)"].corr(data["Molecular Weight (g/mol
print("Covariance matrix: \n", data["Concentration (M)"].cov(data["Molecular Weight (g/mol)"
```

```
Correlation matrix:
 -0.29516999429071955
Covariance matrix:
 -183.64893571428564
```

Pivot Tables Pivot tables are used to summarize and aggregate data inside a dataframe. - `data.pivot_table(index="Chemical Type", columns="Concentration (M)", values="Molecular Weight (g/mol)", aggfunc="mean")`: Returns a pivot table of the dataframe

```
data.pivot_table(index="Chemical Type", columns="Concentration (M)", values="Molecular Weigh
```

Concentration (M)	0.1	0.5	0.8	1.0	1.2	1.5	18.0	55.5
Chemical Type								
Acid	NaN	NaN	NaN	NaN	NaN	NaN	98.079	NaN
Alcohol	NaN	NaN	NaN	46.07	NaN	32.04	NaN	NaN
Aromatic Hydrocarbon	78.11	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Concentration (M)	0.1	0.5	0.8	1.0	1.2	1.5	18.0	55.5
Chemical Type								
Base	NaN	17.03	NaN	NaN	NaN	NaN	NaN	NaN
Inorganic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.015
Ketone	NaN	NaN	58.08	NaN	NaN	NaN	NaN	NaN
Polyol	NaN	NaN	NaN	NaN	92.09	NaN	NaN	NaN

Comparison of Data

- `data1 = data.compare(data2)`: Compares two dataframes and returns the differences

```python
data1 = data.copy()
data1["Concentration (M)"].iloc[0] = 0.5
print("Comparison of data: \n",
data.compare(data1))
```

```
Comparison of data:
   Concentration (M)
                self other
0               55.5   0.5
```

Window Functions

Window functions are used to perform calculations on a subset of the data. It is useful for calculating moving averages, cumulative sums, and other rolling calculations. It helps to smooth out the data and identify trends.

- `df.rolling(...)`: perform rolling window calculations
- `df.rolling(...).mean()`: calculate the rolling mean
- `df.rolling(...).mean()`: calculate the rolling mean
- `df.rolling(...).sum()`: calculate the rolling sum
- `df.rolling(...).std()`: calculate the rolling standard deviation
- `df.rolling(...).min()`: calculate the rolling minimum
- `df.rolling(...).max()`: calculate the rolling maximum
- `df.rolling(...).apply(...)`: apply a function to the rolling window
- `df.rolling(...).agg(...)`: aggregate the data in the rolling window
- `df.rolling(...).transform(...)`: transform the data in the rolling window
- `df.rolling(...).count()`: count the data in the rolling window

```python
# calculate the rolling mean
print("rolling mean: \n", data["Concentration (M)"].rolling(window=2).mean())
```

```
rolling mean:
 0      NaN
1    36.75
2     9.50
3     0.90
4     0.65
5     0.30
6     0.80
7     1.35
Name: Concentration (M), dtype: float64
```

Example: Descriptive Statistics

Exercises

Download it locally and try to solve the exercises.

Data Import and Inspection Example

Or open it in `Google Colab`:

Data Import and Inspection Example

21 Model Testing

Difficulty level:

Hypothesis Testing

Hypothesis testing is a statistical method for making decisions or inferences about a population based on a sample. It is widely used in chemistry and materials science to validate experimental results and compare datasets.

- **Null & Alternative Hypothesis:** The null hypothesis (H) assumes no effect, while the alternative hypothesis (H) suggests a significant difference.
- **t-tests:** Used to compare the means of two groups (one-sample, independent, paired t-tests).

```python
# Example: Null hypothesis - The mean is equal to 0
#H0:   = 0

# Alternative hypothesis - The mean is not equal to 0
#H1:    0

from scipy import stats
import numpy as np

data = [1, 2, 3, 4, 5]

# Perform a one-sample t-test
t_stat, p_value = stats.ttest_1samp(data, 0)

# Print results
print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value:.4f}")

# Interpretation
if p_value < 0.05:
    print("Reject the null hypothesis: The mean is significantly different from 0.")
```

```python
else:
    print("Fail to reject the null hypothesis: No significant difference from 0.")
```

```
T-Statistic: 4.2426, P-Value: 0.0132
Reject the null hypothesis: The mean is significantly different from 0.
```

```python
from scipy import stats
import numpy as np
# Generate two random samples (e.g., control and treatment groups)
np.random.seed(42)
control = np.random.normal(loc=50, scale=10, size=30)  # Mean=50, Std=10
treatment = np.random.normal(loc=55, scale=10, size=30)  # Mean=55, Std=10

# Perform an independent t-test
t_stat, p_value = stats.ttest_ind(control, treatment)

# Print results
print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value:.4f}")

# Interpretation
if p_value < 0.05:
    print("Reject the null hypothesis: The groups have significantly different means.")
else:
    print("Fail to reject the null hypothesis: No significant difference between groups.")
```

```
T-Statistic: -2.3981, P-Value: 0.0197
Reject the null hypothesis: The groups have significantly different means.
```

```python
from scipy import stats
import numpy as np
# Generate paired data (e.g., before and after measurements)
np.random.seed(42)
before = np.random.normal(loc=50, scale=10, size=30)  # Mean=50, Std=10
after = before + np.random.normal(loc=5, scale=2, size=30)  # Mean=55, Std=2

# Perform a paired t-test
t_stat, p_value = stats.ttest_rel(before, after)

# Print results
print(f"T-Statistic: {t_stat:.4f}, P-Value: {p_value:.4f}")
```

```
# Interpretation
if p_value < 0.05:
    print("Reject the null hypothesis: The means are significantly different.")
else:
    print("Fail to reject the null hypothesis: No significant difference between means.")
```

```
T-Statistic: -13.9936, P-Value: 0.0000
Reject the null hypothesis: The means are significantly different.
```

- **ANOVA (Analysis of Variance):** Compares means of multiple groups to check for significant differences.

```
from scipy import stats
import numpy as np

# Example: Comparing multiple groups
group1 = [10.2, 14.5, 13.3, 9.8, 12.7]
group2 = [9.5, 13.1, 12.9, 8.7, 11.3]
group3 = [11.2, 15.1, 14.3, 10.8, 13.7]

# Perform ANOVA test
f_stat, p_value = stats.f_oneway(group1, group2, group3)
print(f"F-statistic: {f_stat}, P-value: {p_value}")
```

```
F-statistic: 1.1840438281116243, P-value: 0.3393857967100532
```

- **Chi-square Test:** Used for categorical data to examine associations between different groups.

```
from scipy import stats
import numpy as nps
from scipy.stats import chi2_contingency

# Example: Comparing categorical data
observed = [[10, 20, 30], [6, 9, 17]]
chi2, p, dof, expected = chi2_contingency(observed)
print(f"Chi-square: {chi2}, P-value: {p}")
```

```
Chi-square: 0.27157465150403504, P-value: 0.873028283380073
```

Check the SciPy documentation for more statistical tests and functions.

And for more information on hypothesis testing, refer to the SciPy Hypothesis Tests.

Outlier Detection

Outliers are data points that significantly differ from other observations in a dataset. They have effect on specific statistical methods and can lead to incorrect conclusions.

- **Z-Score:** Measures how many standard deviations a data point is from the mean.

```python
import numpy as np
# Calculate Z-scores for a dataset
data = np.random.randn(100)
data[0] = 100  # Introduce an outlier
z_scores = stats.zscore(data)
print(z_scores)
print("Outliers: ", data[np.abs(z_scores) > 3])
```

```
[ 9.90505941e+00 -1.17555563e-01 -2.09660373e-01 -2.18651170e-01
 -1.76967409e-02  3.66966543e-02 -1.06186084e-01  1.41168632e-03
 -6.28039437e-02 -1.63520210e-01 -6.28279953e-02  5.48836555e-02
 -1.02566214e-01  5.75454398e-02 -3.61062551e-01 -1.67586938e-02
 -9.02739371e-02 -1.28894982e-01 -8.98023757e-02 -2.97819382e-01
 -1.20958229e-01 -6.32564718e-02  4.88669726e-02 -1.50830130e-01
 -1.79864198e-01 -1.49178145e-01 -7.40495357e-03 -6.60937643e-02
 -1.51979593e-01 -4.76346746e-02 -8.92704836e-02 -2.07851444e-03
 -1.69215845e-01 -1.31761619e-01 -1.38208825e-01 -2.45392806e-01
 -6.93581663e-02 -7.28660840e-02 -9.84706096e-02 -1.22450356e-01
 -2.40576439e-01 -1.41063695e-01 -1.33267465e-01 -1.79242313e-01
 -1.15117252e-01 -5.85607463e-02  8.97126597e-02 -8.15173250e-02
 -7.32167138e-02 -1.06429762e-01 -2.90936832e-01 -1.01634621e-01
 -9.29567067e-02  1.47441603e-01 -1.18226033e-01 -6.88152405e-02
 -1.02454742e-01 -2.15897199e-01  1.53463075e-02 -2.37584688e-02
 -1.98469971e-02 -1.89957661e-01  4.13539641e-02 -2.39223925e-01
 -4.02727343e-02  1.20151929e-01 -1.98075828e-01 -1.55634822e-01
 -8.90129980e-02 -1.49350076e-01 -2.54111176e-01 -9.21230935e-02
 -2.05255467e-01 -5.16037783e-02 -1.90961745e-01  5.60739202e-02
 -1.77339147e-01 -1.31201330e-01 -1.75975614e-02 -2.22118340e-01
 -7.62269755e-02  3.17849426e-02 -2.59795453e-01 -8.05113140e-02
 -7.29833792e-02 -2.07682769e-02 -2.22727225e-01 -2.31081190e-01
 -4.67669108e-02 -6.92716918e-02 -7.39227531e-02 -6.43233391e-02
 -1.67012118e-01 -7.57474055e-02 -6.96630699e-02 -1.70446175e-01
  8.76706957e-02 -5.15797196e-02 -2.18160659e-01 -3.33002660e-02]
Outliers:  [100.]
```

- **IQR (Interquartile Range):** Identifies outliers based on the spread of the data.

```python
import numpy as np
# Calculate IQR for a dataset
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1

# Define outlier thresholds
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Identify outliers
outliers = data[(data < lower_bound) | (data > upper_bound)]
print(outliers)
```

```
[100.          -2.6197451    2.46324211]
```

- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** Clustering algorithm that identifies outliers based on density.

```python
import numpy as np
from sklearn.cluster import DBSCAN

# Generate sample data
data = np.random.randn(100, 2)

# Fit DBSCAN model
dbscan = DBSCAN(eps=0.3, min_samples=5)
dbscan.fit(data)

# Identify outliers
outliers = data[dbscan.labels_ == -1]
print(outliers)
```

```
[[-0.97468167  0.7870846 ]
 [ 1.15859558 -0.82068232]
 [ 0.82206016  1.89679298]
 [-0.88951443 -0.81581028]
 [ 0.01300189  1.45353408]
 [-0.26465683  2.72016917]
 [ 0.62566735 -0.85715756]
 [-1.0708925   0.48247242]
 [ 0.47323762 -0.07282891]
 [-0.84679372 -1.51484722]
 [ 1.08305124  1.05380205]
 [-1.37766937 -0.93782504]
 [ 0.51504769  3.85273149]
 [ 2.31465857 -1.86726519]
 [ 0.68626019 -1.61271587]
 [ 2.14394409  0.63391902]
 [-2.02514259  0.18645431]
 [-1.20029641 -0.33450124]
 [-0.47494531 -0.65332923]
 [ 1.76545424  0.40498171]
 [-1.26088395  0.91786195]
 [ 2.1221562   1.03246526]
 [-1.51936997 -0.48423407]
 [ 1.26691115 -0.70766947]
 [-3.24126734 -1.02438764]
 [-0.25256815 -1.24778318]
 [ 1.6324113  -1.43014138]
 [-0.44004449  0.13074058]
 [ 1.44127329 -1.43586215]
 [ 1.16316375  0.01023306]
 [-0.98150865  0.46210347]
 [ 1.58601682 -1.2378155 ]
 [ 2.13303337 -1.9520878 ]
 [-0.02090159  0.11732738]
 [ 1.2776649  -0.59157139]
 [ 0.54709738 -0.20219265]
 [-0.2176812   1.09877685]
 [ 1.30547881  0.02100384]
 [ 0.68195297 -0.31026676]
 [ 0.32416635 -0.13014305]
 [-0.81822068  2.09238728]
 [-1.00601738 -1.21418861]
 [ 0.97511973 -0.14705738]
 [-1.44808434 -1.40746377]
 [ 0.31090757  1.47535622]
 [ 0.85765962 -0.15993853]]
```

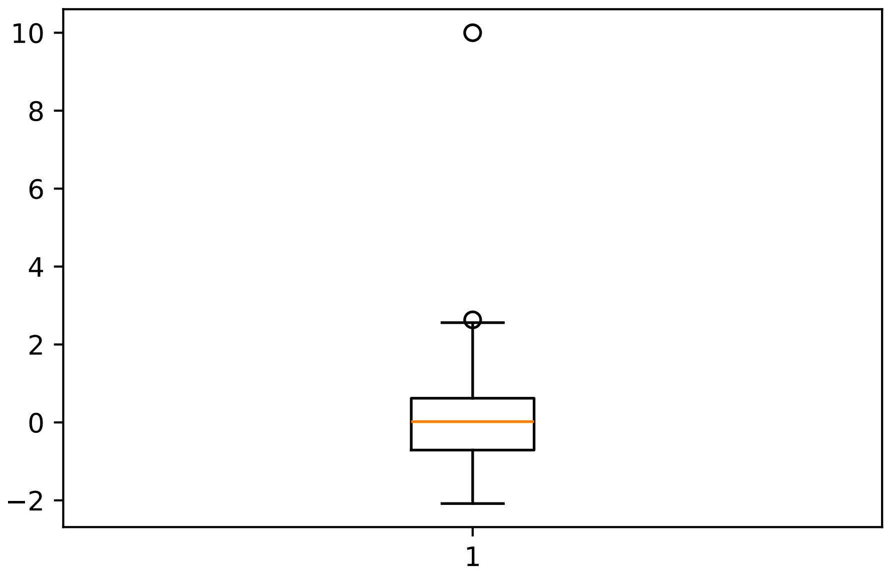
- **Isolation Forest:** Anomaly detection algorithm that isolates outliers in a dataset.

```
import numpy as np
from sklearn.ensemble import IsolationForest

# Generate sample data
data = np.random.randn(100, 2)
data[0] = [10, 10]  # Introduce an outlier

# Fit Isolation Forest model
isoforest = IsolationForest(contamination=0.1)
isoforest.fit(data)

# Identify outliers
outliers = data[isoforest.predict(data) == -1]
print(outliers)
```

```
[[10.         10.        ]
 [-2.12389572 -0.52575502]
 [-1.32023321  1.83145877]
 [ 2.06074792  1.75534084]
 [-1.18325851 -2.03923218]
 [-2.0674421  -0.08912004]
 [-0.2750517  -2.30192116]
 [ 3.07888081  1.11957491]
 [ 0.92617755  1.90941664]
 [-2.4716445  -0.79689526]]
```

- **Boxplot:** Visual representation of outliers in a dataset.

```
import numpy as np
from matplotlib import pyplot as plt
# Generate sample data'
data = np.random.randn(100)
data[0] = 10  # Introduce an outlier

# Create boxplot
plt.boxplot(data)
plt.show()
```



For more information on outlier detection, refer to the Scikit-learn documentation.

Example: Model Testing

Exercises

Download it locally and try to solve the exercises.

Model Testing Example

Or open it in `Google Colab`:

Model Testing Example

22 Inferential Statistics

Difficulty level:

Data Models

Linear Regression

Often we want to find a model which can explain the data. It is important to understand the data and the model to be able to interpret the results and make predictions.

The simplest model is the linear regression model. It assumes that the data has a linear relationship with the target variable. For example, if we have a single feature x and a target variable y, the linear regression model can be defined as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where y is the target variable, x is the feature, β_0 and β_1 are the coefficients, and ϵ is the error term.

For multiple features $x_1, x_2, \dots, x_n$, the linear regression model can be defined as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

where y is the target variable, $x_1, x_2, \dots, x_n$ are the features, $\beta_0, \beta_1, \dots, \beta_n$ are the coefficients, and ϵ is the error term.

In `python` there exists serveral libraries which can be used to fit a linear regression model.

22.0.1 Using `scipy`

A easy way is to use `scipy` library. The `scipy` library has a function called `linregress` which can be used to fit a linear regression model. The function returns the slope, intercept, r-value, p-value, and the standard error of the estimate. And with scipy version 1.15.2 also the intercept error.

```python
from scipy.stats import linregress
import numpy as np

x = [1, 2, 3, 4, 5]
y = x*np.random.normal(0, 1, 5)+np.random.normal(0, 1, 5)

results = linregress(x, y)

slope = results.slope
intercept = results.intercept
r_value = results.rvalue
p_value = results.pvalue
std_err = results.stderr
intercept_err = results.intercept_stderr
print("slope: %f    intercept: %f" % (slope, intercept))
print("R-squared: %f" % r_value**2)
print("p-value: %f" % p_value)
print("standard error: %f" % std_err)
print("Intercept error: %f" %intercept_err)
# Two-sided inverse Students t-distribution
# p - probability, df - degrees of freedom
from scipy.stats import t
tinv = lambda p, df: abs(t.ppf(p/2, df))
print("95% confidence interval: " + str(intercept - tinv(0.05, len(x)-2)*intercept_err) + " 
```

```
slope: 1.561849    intercept: -3.052131
R-squared: 0.745257
p-value: 0.059423
standard error: 0.527202
Intercept error: 1.748531
95% confidence interval: -8.61673825839533 to 2.5124765794392263
```

! Important

For older version `scipy`only returned 5 values with fields slope, intercept, rvalue, pvalue and stderr. For compatiblity reasons the return values are 5 elements tuple.

```python
from scipy.stats import linregress
slope, intercept, r, p, se = linregress(x, y)
print("slope: ", slope)
print("intercept: ", intercept)
print("r-value: ", r)
print("p-value: ", p)
print("standard error: ", se)
```

```
slope:  1.561849163253334
intercept:  -3.052130839478052
r-value:  0.8632824131244606
p-value:  0.059423389317344436
standard error:  0.527202065322916
```

And if you want to get the intercept error you can use the following return value as a object:

```python
from scipy.stats import linregress
results = linregress(x, y)
print("slope: ", results.slope)
print("intercept: ", results.intercept)
print("r-value: ", results.rvalue)
print("p-value: ", results.pvalue)
print("standard error: ", results.stderr)
print("intercept error: ", results.intercept_stderr)
```

```
slope:  1.561849163253334
intercept:  -3.052130839478052
r-value:  0.8632824131244606
p-value:  0.059423389317344436
standard error:  0.527202065322916
intercept error:  1.74853143937655
```

22.0.2 Using `statsmodels`

Another library is `statsmodels`. The `statsmodels` library provides more detailed information about the model, such as the coefficients, standard errors, t-values, p-values, and confidence intervals.

```python
import statsmodels.api as sm
import numpy as np

X = np.array([1, 2, 3, 4, 5])
y = X*np.random.normal(0, 1, 5)+np.random.normal(0, 1, 5)

X_with_const = sm.add_constant(X)  # Add intercept term
model = sm.OLS(y, X_with_const).fit()
print(model.summary())
```

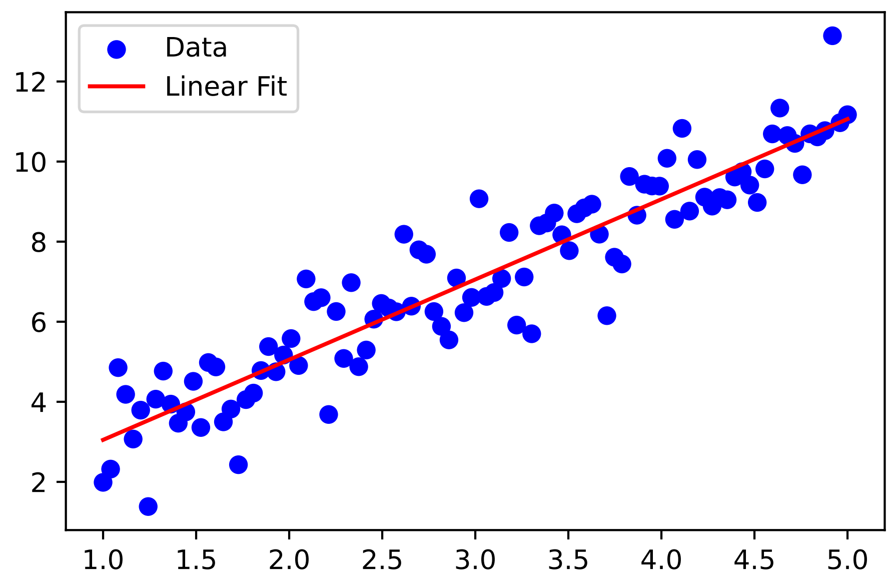
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.031
Model:                            OLS   Adj. R-squared:                 -0.292
Method:                 Least Squares   F-statistic:                   0.09623
Date:                Fri, 11 Apr 2025   Prob (F-statistic):              0.777
Time:                        10:41:31   Log-Likelihood:                -13.663
No. Observations:                   5   AIC:                             31.33
Df Residuals:                       3   BIC:                             30.54
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.4040      5.036      0.477      0.666     -13.624      18.431
x1            -0.4711      1.518     -0.310      0.777      -5.304       4.361
==============================================================================
Omnibus:                          nan   Durbin-Watson:                   1.941
Prob(Omnibus):                    nan   Jarque-Bera (JB):                0.408
Skew:                          -0.100   Prob(JB):                        0.815
Kurtosis:                       1.615   Cond. No.                         8.37
==============================================================================

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

Using `scikit-learn`

One of the most popular libraries is `scikit-learn`. The following code shows how to fit a linear regression model using `scikit-learn`:

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

# Sample data
X = np.linspace(1, 5, 100).reshape(-1, 1)
y = 2 * X + 1 + np.random.normal(0, 1, 100).reshape(-1, 1)

# Model fitting
model = LinearRegression()
model.fit(X, y)
print("Parameters:", model.get_params())
print("R-squared:", model.score(X, y))

# Predictions
y_pred = model.predict(X)

# Plot
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, y_pred, color='red', label='Linear Fit')
plt.legend()
plt.show()
```

```
Parameters: {'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
R-squared: 0.8698274418684563
```



22.1 Non-Linear Fits

Linear regression may not always be sufficient, especially for complex relationships. Non-linear models provide more flexibility.

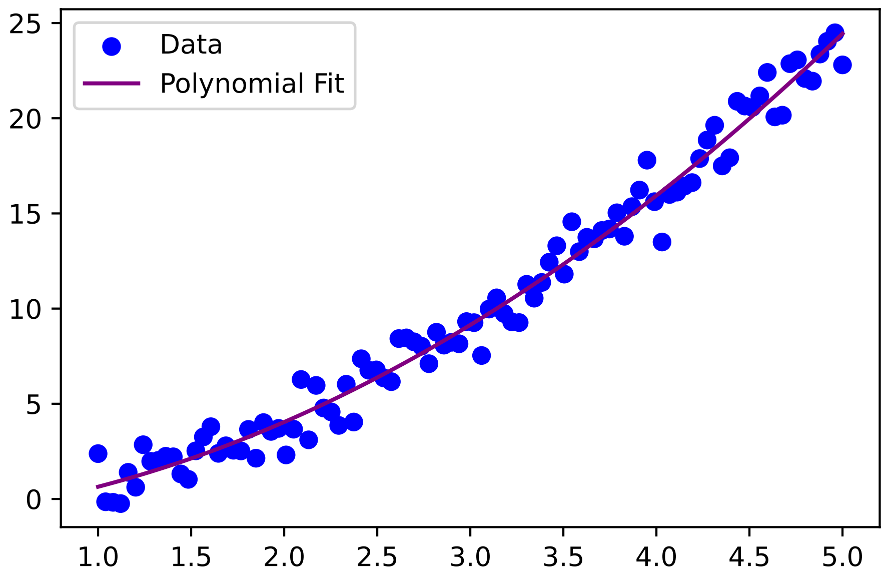
22.1.1 Polynomial Regression

Polynomial regression is a type of linear regression where the relationship between the independent variable x and the dependent variable y is modeled as an n-th degree polynomial.

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


# Sample data
X = np.linspace(1, 5, 100)
y = X**2 + np.random.normal(0, 1, 100)

model = np.polyfit(X, y, 2)
```

```
print("Coefficients:", model)

# Create a polynomial function

model = np.poly1d(model)

X_range = np.linspace(1, 5, 100)
y_fit = model(X_range)

print(model)

plt.scatter(X, y, color='blue', label='Data')
plt.plot(X_range, y_fit, color='purple', label='Polynomial Fit')
plt.legend()
```

```
Coefficients: [ 0.85020179  0.84927237 -1.05865633]
        2
0.8502 x + 0.8493 x - 1.059
```

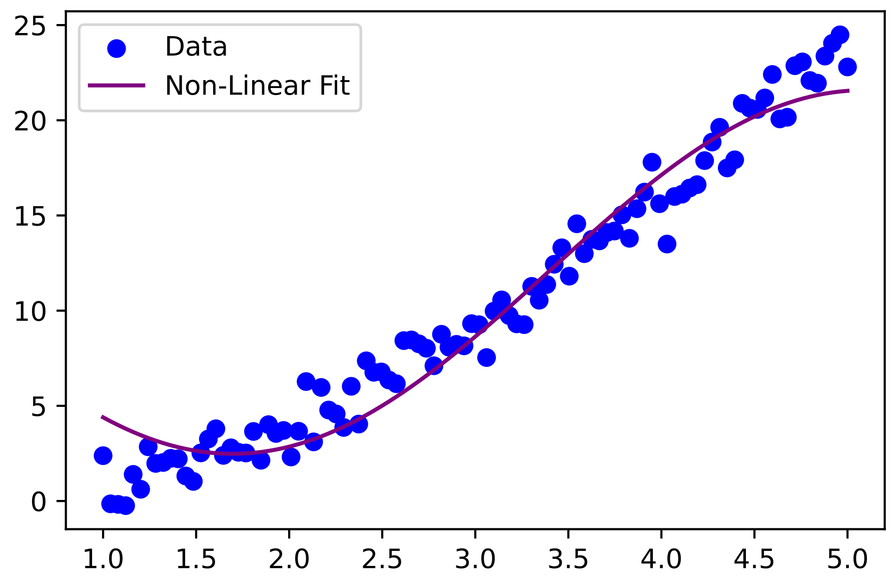


22.1.2 Curve Fitting with `scipy`

`scipy` provides the `curve_fit` function to fit a non-linear model to the data. The function requires the model function and the data as input.

```
from scipy.optimize import curve_fit

def nonlinear_func(x, a, b, c):
    return a * np.sin(b * x) + c

popt,pov  = curve_fit(nonlinear_func, X.flatten(), y)
perr = np.sqrt(np.diag(pov))

print("Fitted parameters:", popt)
print("Parameter errors:", perr)

X_range = np.linspace(1, 5, 100)
y_fit = nonlinear_func(X_range, *popt)

plt.scatter(X, y, color='blue', label='Data')
plt.plot(X_range, y_fit, color='purple', label='Non-Linear Fit')
plt.legend()
plt.show()
```

```
Fitted parameters: [-9.54970511  0.92619133 12.0288274 ]
Parameter errors: [0.27053402 0.01453711 0.31589986]
```



23 Advanced Statistical Analysis

Difficulty level:

Often data has multi dimensions and it is not easy to analyze it. Multivariate analysis is a set of statistical techniques used to analyze data with multiple variables. It helps in understanding the relationships between variables and identifying patterns in complex data.

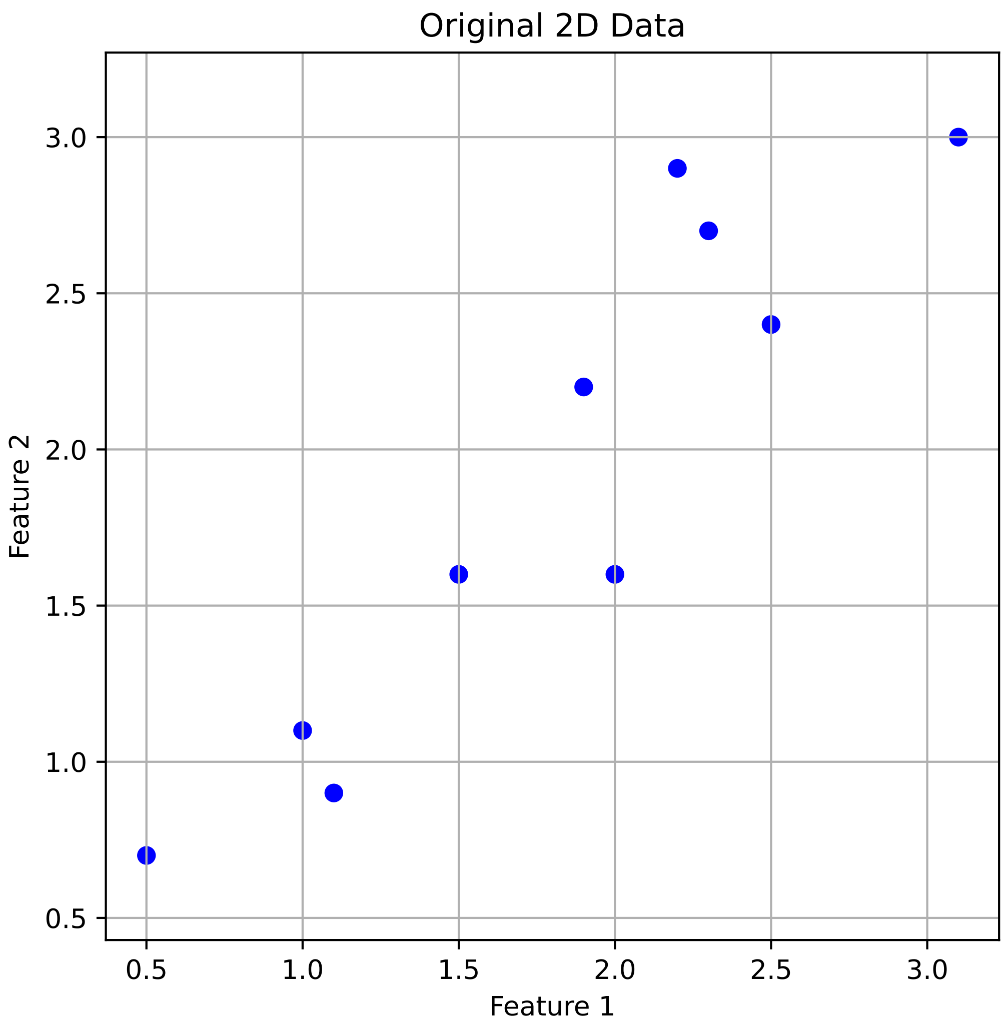
What can be done to analyze multi-dimensional data?

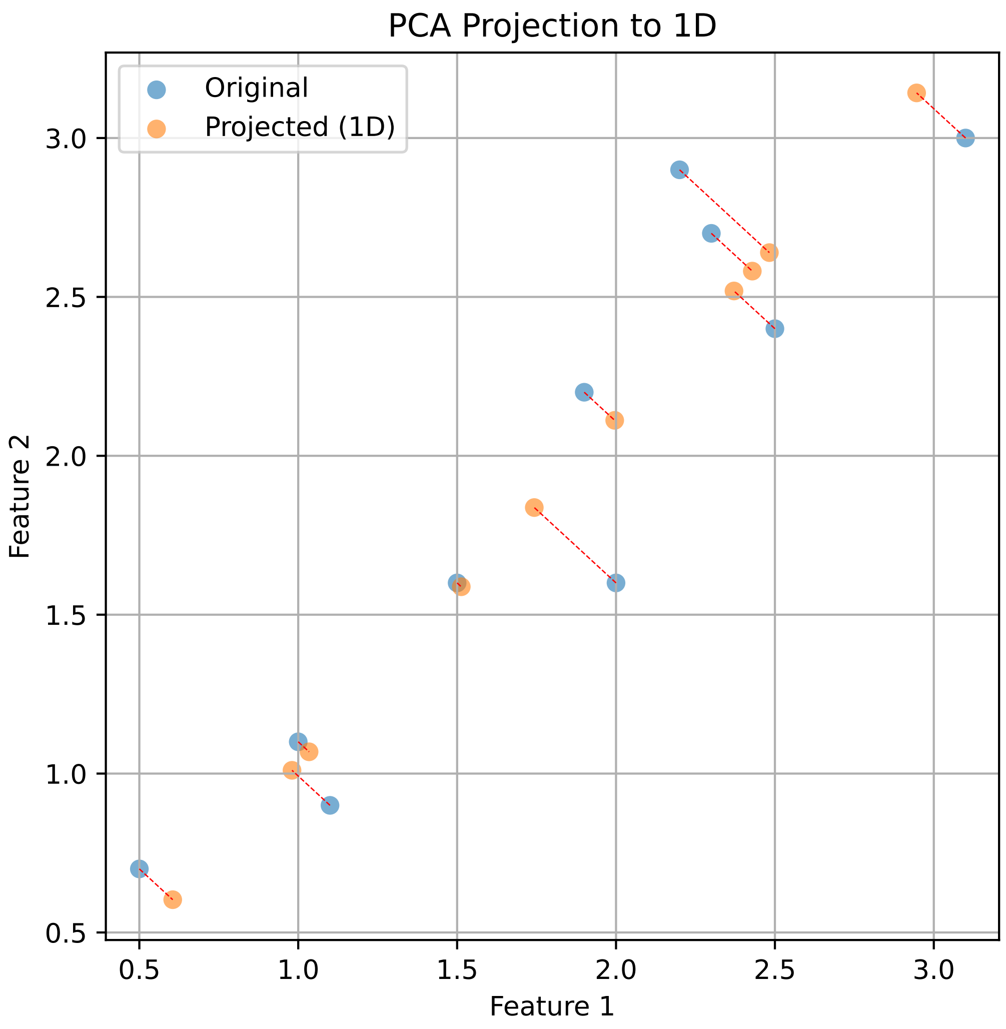
- **Dimensionality Reduction:** Simplifies complex datasets by reducing the number of variables.
- **Pattern Recognition:** Identifies underlying patterns and trends in data.
- **Feature Selection:** Selects relevant variables for modeling and prediction.

Dimensionality Reduction Techniques

Principal Component Analysis (PCA): reduces the dimensionality of data by transforming variables into uncorrelated components. It retains most of the variance in the data while reducing noise and redundancy. The PCA decomposition is based on the eigenvalues and eigenvectors of the covariance matrix of the data to identify the principal components with the highest variance.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Create a simple 2D dataset
X = np.array([
    [2.5, 2.4],
    [0.5, 0.7],
    [2.2, 2.9],
    [1.9, 2.2],
    [3.1, 3.0],
    [2.3, 2.7],
    [2, 1.6],
    [1, 1.1],
    [1.5, 1.6],
    [1.1, 0.9]
])

# Plot original data
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], color='blue')
plt.title("Original 2D Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.axis("equal")
plt.grid(True)
plt.show()

# Apply PCA to reduce to 1D
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X)
X_projected = pca.inverse_transform(X_pca)

# Plot the projected data
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], label='Original', alpha=0.6)
plt.scatter(X_projected[:, 0], X_projected[:, 1], label='Projected (1D)', alpha=0.6)
for orig, proj in zip(X, X_projected):
    plt.plot([orig[0], proj[0]], [orig[1], proj[1]], 'r--', linewidth=0.5)
plt.title("PCA Projection to 1D")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.axis("equal")
plt.grid(True)
plt.show()
```


Original 2D Data
Feature 1
Feature 2
0.5
1.0
1.5
2.0
2.5
3.0
0.5
1.0
1.5
2.0
2.5
3.0

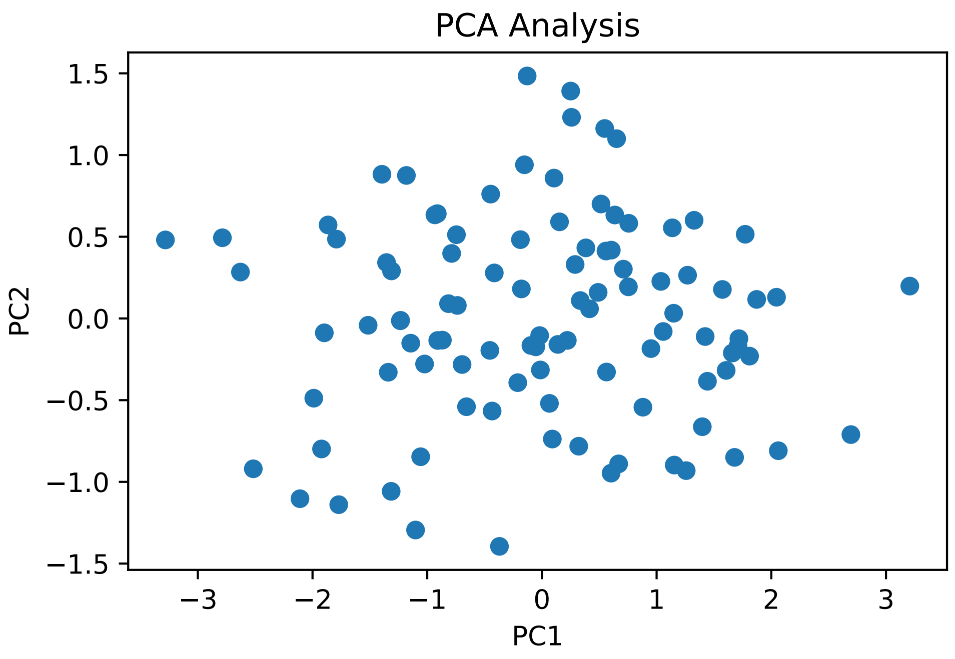


Sometimes the data needs to be scaled before applying PCA. The `StandardScaler` from `sklearn` can be used to scale the data.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Generate synthetic data
rng = np.random.RandomState(0)
n_samples = 100
cov = [[6, 1], [8, 4]]
X = rng.multivariate_normal(mean=[0, 0], cov=cov, size=n_samples)
```

```
# Standardizing Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Plot PCA results
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Analysis')
plt.show()
```

```
Explained Variance Ratio: [0.81292552 0.18707448]
```

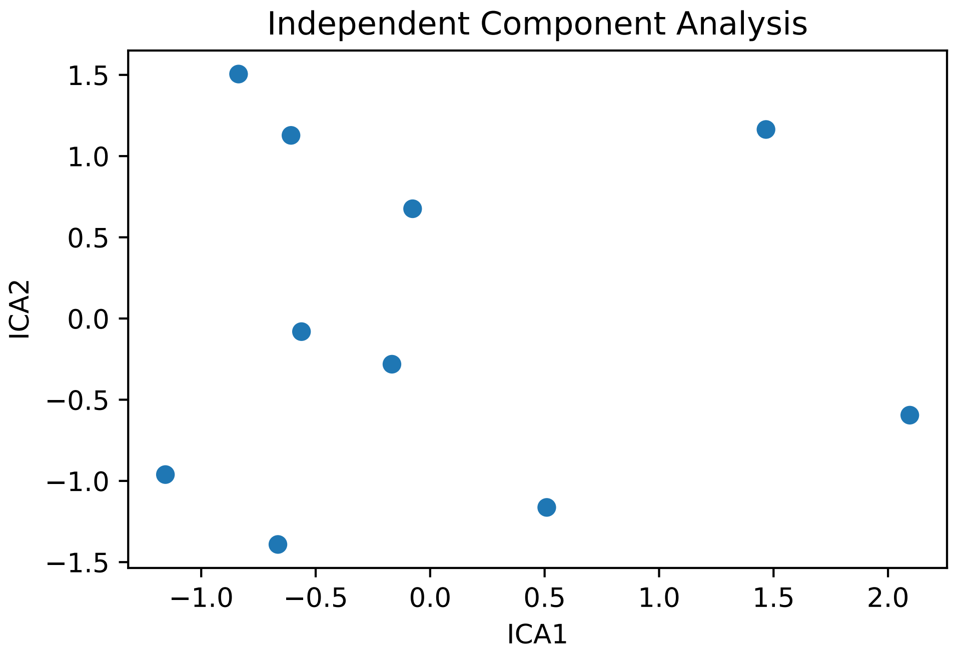


PCR (Principal Component Regression): Combines PCA and regression analysis. It first applies PCA to reduce dimensionality and then performs regression on the principal components. (see for more information https://en.wikipedia.org/wiki/Principal_component_regression and https://scikit-learn.org/stable/auto_examples/cross_decomposition/plot_pcr_vs_pls.html

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

from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
# Generate synthetic data
rng = np.random.RandomState(0)
n_samples = 100
cov = [[6, 1], [8, 4]]
X = rng.multivariate_normal(mean=[3, 5], cov=cov, size=n_samples)

# generate PCA components
pca = PCA(n_components=2).fit(X)
```

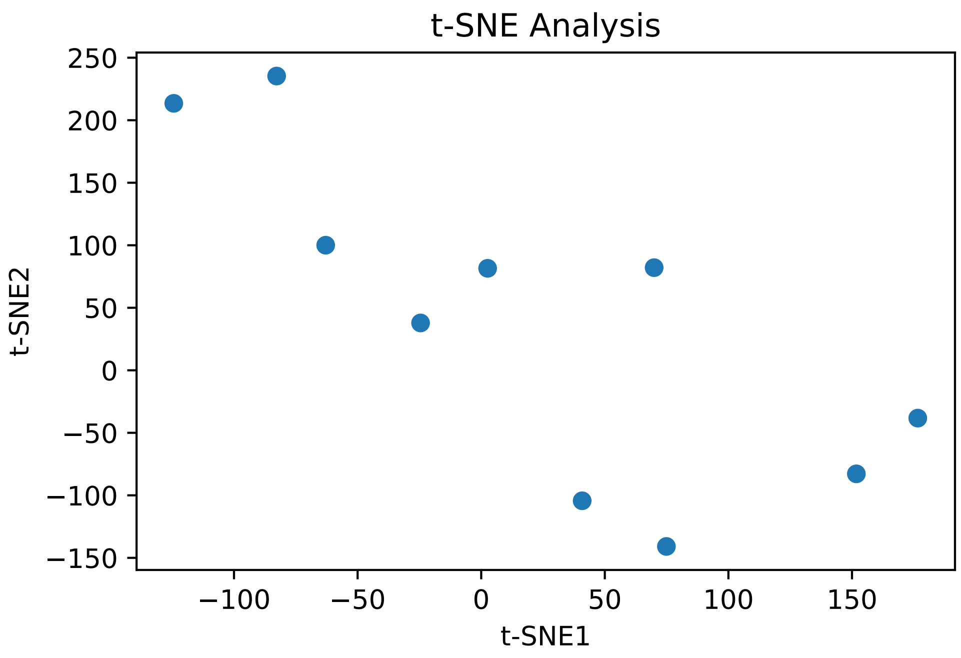
Independent Component Analysis (ICA): Separates a multivariate signal into additive subcomponents.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import FastICA

# Sample dataset
X = np.random.rand(10, 3)
ica = FastICA(n_components=2)
X_ica = ica.fit_transform(X)

# Plot ICA results
plt.scatter(X_ica[:, 0], X_ica[:, 1])
plt.xlabel('ICA1')
plt.ylabel('ICA2')
plt.title('Independent Component Analysis')
plt.show()
```



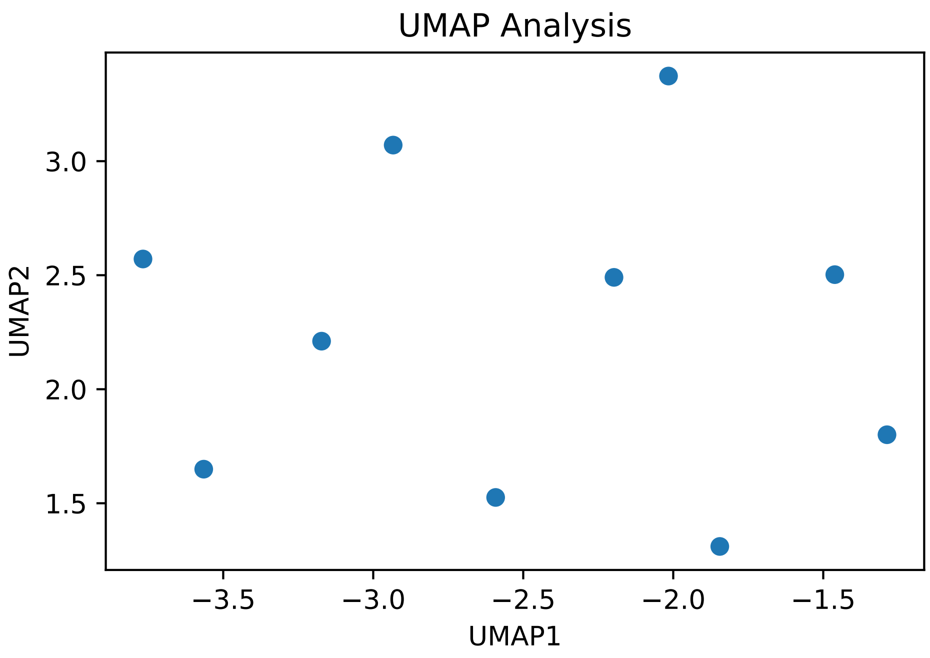
t-Distributed Stochastic Neighbor Embedding (t-SNE): Visualizes high-dimensional data in lower dimensions.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Sample dataset
X = np.random.rand(10, 3)
tsne = TSNE(n_components=2,perplexity=3,learning_rate='auto')
X_tsne = tsne.fit_transform(X)

# Plot t-SNE results
plt.scatter(X_tsne[:, 0], X_tsne[:, 1])
plt.xlabel('t-SNE1')
plt.ylabel('t-SNE2')
plt.title('t-SNE Analysis')
plt.show()
```



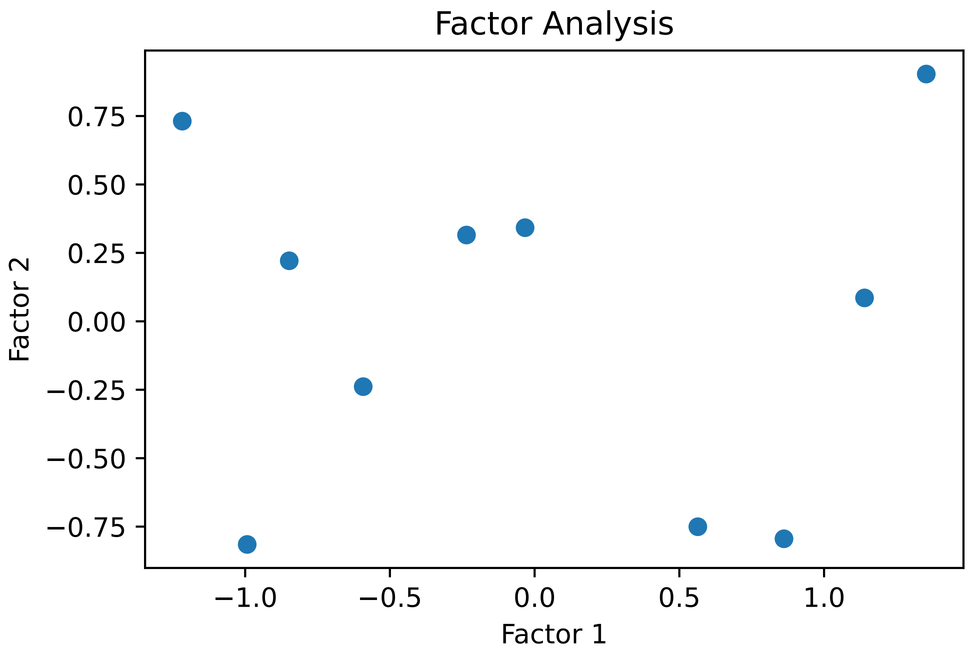
UMAP (Uniform Manifold Approximation and Projection): Reduces the dimensionality of data while preserving local and global structure.

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

# Sample dataset
X = np.random.rand(10, 3)
umap_model = umap.UMAP(n_components=2)
X_umap = umap_model.fit_transform(X)

# Plot UMAP results
plt.scatter(X_umap[:, 0], X_umap[:, 1])
plt.xlabel('UMAP1')
plt.ylabel('UMAP2')
plt.title('UMAP Analysis')
plt.show()
```



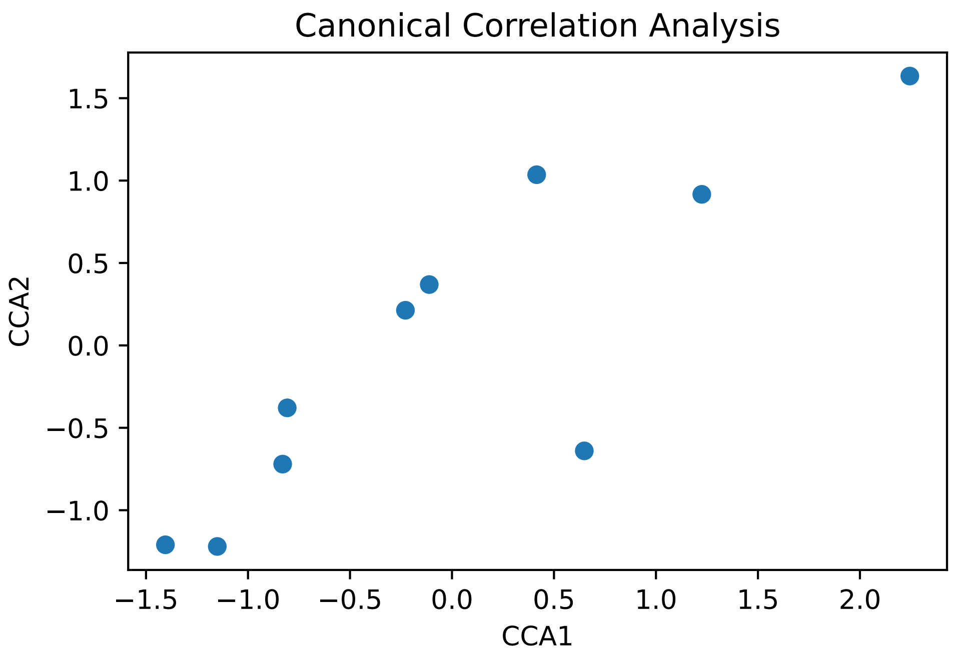
Factor Analysis: Identifies latent factors that explain the variance in observed variables.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import FactorAnalysis

# Sample dataset
X = np.random.rand(10, 3)
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X)

# Plot Factor Analysis results
plt.scatter(X_fa[:, 0], X_fa[:, 1])
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.title('Factor Analysis')
plt.show()
```



Canonical Correlation Analysis (CCA): Analyzes the relationship between two sets of variables.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cross_decomposition import CCA

# Sample dataset
X = np.random.rand(10, 3)
Y = np.random.rand(10, 3)
cca = CCA(n_components=2)
X_c, Y_c = cca.fit_transform(X, Y)

# Plot CCA results
plt.scatter(X_c[:, 0], Y_c[:, 0])
plt.xlabel('CCA1')
plt.ylabel('CCA2')

plt.title('Canonical Correlation Analysis')
plt.show()
```

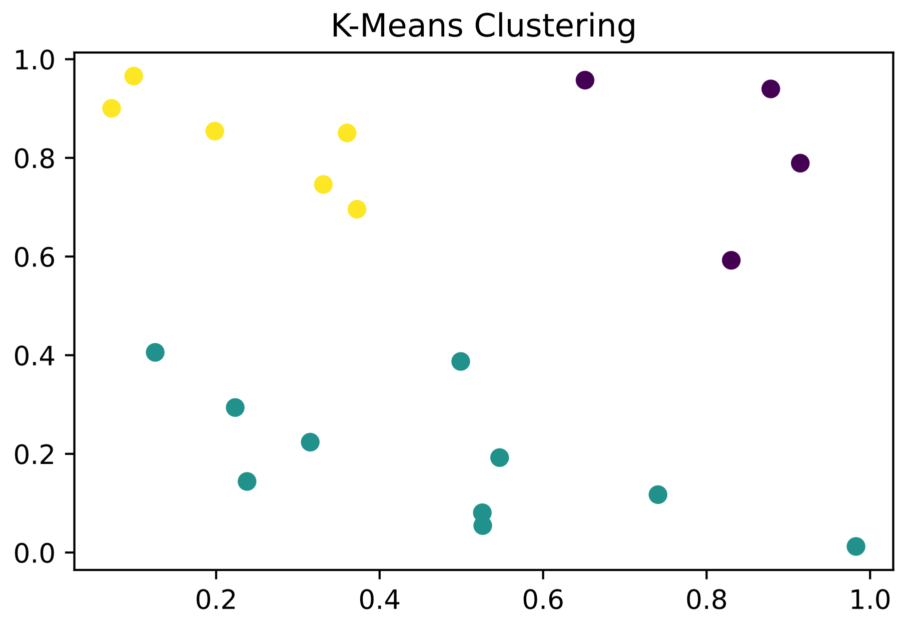


Clustering Techniques

Clustering is an unsupervised learning technique used to group similar data points based on patterns. In chemistry and materials science, clustering helps in categorizing material properties, identifying experimental trends, and classifying samples.

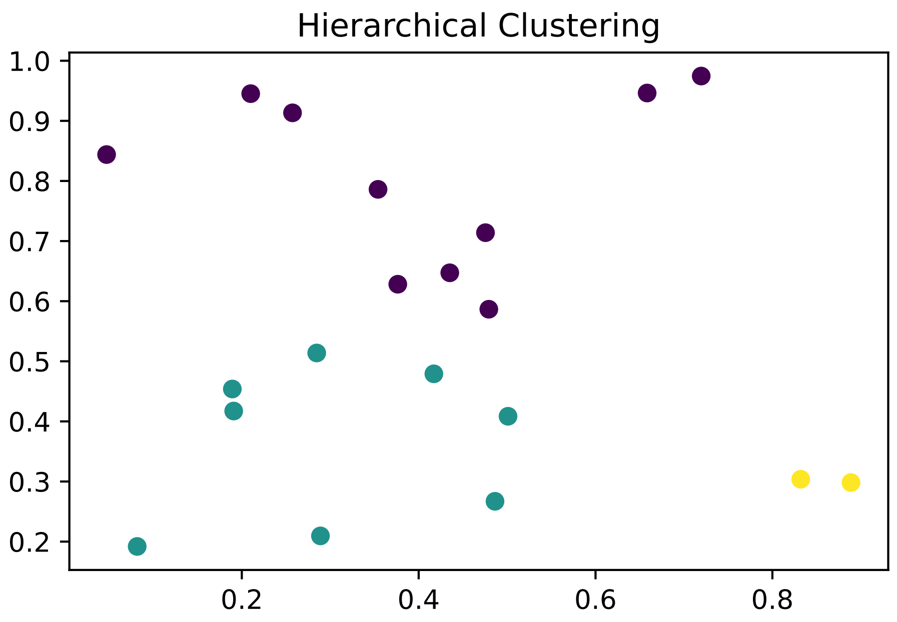
- **k-Means Clustering:** Partitions data into k clusters based on similarity.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Example dataset
X = np.random.rand(20, 2)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
labels = kmeans.labels_

# Plot Clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('K-Means Clustering')
plt.show()
```



- **Hierarchical Clustering:** Builds a tree of clusters using agglomerating or divisive methods.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

# Example dataset
X = np.random.rand(20, 2)
agg = AgglomerativeClustering(n_clusters=3)
labels = agg.fit_predict(X)

# Plot Clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('Hierarchical Clustering')
plt.show()
```


Hierarchical Clustering
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8

Part VII

Exercises

24 Exercise A 2

24.1 Temperature in Synthesis Reactor Part 2

In this exercise, we will repeat not-linear fits.

Let's analyze how bad the situation is in reactor 3 is. To do this, we will fit a sigmoid curve to the data set. Try to remember which kind of function we used to fit the non-linear data in the lecture.

In our case the sigmoid function is given by:

$$T(t) = T_0 + \Delta T \frac{1}{1 + e^{-a(t-t_0)}}$$

with the four fitting parameters being:

- temperature baseline $T_0 $,
- the temperature increase ΔT,
- the growth rate a,
- the time at the midpoint of the temperature increase t_0

Use your results from Part 1 of this exercise.

24.1.1 Task 1:

- Fit a sigmoid function to the data set of reactor 3.
- Plot the data and the fitted curve.

24.1.2 Questions:

- How much is the tempearture changed in reactor 3?
- How fast this change appears?
- When does the temperature reach the midpoint of the change?

24.1.3 Task2:

- Fit a suitable distribution function over the histograms plots of the temperature data

24.1.4 Questions:

- What can you say about the distribution of the temperature data?

25 Exercise B 1

26 Exercise:

26.1 Linear Regression: Calibration IR spectroscopy

In this exercise, we will repeat linear regressions.

- We will learn how to use `np.polyfit` to perform linear regression.
- We will learn how to use `scipy.stats.linregress` to perform linear regression.
- We will learn how to use `sklearn.linear_model.LinearRegression` to perform linear regression.
- We learn what are the differences between these three methods.
- We will learn how to use the linear regression to calculate the concentration of a unknown sample.

In quantitative spectroscopy we measure the (dimensionless) absorbance A of a beam of light at a selected frequency/wavelength to determine the concentration c of a sample. To do this, we first calibrate the device using solutions with known concentration.

26.1.1 Data

In our exercise we compare the calibrations of two groups of students available in the files `ir_spectroscopy1.dat` and `ir_spectroscopy2.dat`.

26.1.2 Data Path:

```
data_path = "https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exe
```

26.1.3 Task

- Load the data from the files `ir_spectroscopy1.dat` and `ir_spectroscopy2.dat`.
- Perform a linear regression on the data of each group.
- Compare the results of the three methods.
- Use the linear regression to predict the concentration of our unknown sample measured at the absorption of $A_{\text{sample}} = 0.753$.

26.1.4 Questions

- What is the concentration of the unknown sample?
- Are the differences between the three methods?
- Which method would you recommend for this type of data?

27 Exercise B 2

28 Exercise:

28.1 Linear Regression and Correlation: Experimental vs. Theoretical Data

In this exercise, we will repeat linear regressions and use it for demonstration of the correlation between experimental and theoretical data.

- We will learn how to use create symmetrical plots.
- We will learn to use z-scores outlier test.

In this exercise we compare experimentally measured electrochemical potentials of anthraquinone (AQ) derivatives with those calculated using theoretical methods. We want to find out how well the calculation can reproduce the experimental data. And check if there are any outliers in the data.

Anthraquinone is an aromatic organic molecule, with manifold applications. In nature, AQ and its derivatives are found in plants and microorganisms due to its key role in reversible redox reactions. For this reason AQ derivates are also key components in industrial processes and material development (such as battery research at the University of Innsbruck).

28.1.1 Data

The data for this exercise is taken from a recent collaboration between the Theoretical Chemistry Department and the Institute for Physical chemistry. (Phys.Chem.Chem.Phys. 2022, **24**, *16207 – 16219*).

The experimental data is stored in the file `experimental.dat` and the theoretical data is stored in the file `theory.dat`.

The data contains the following columns:

- `1-OH` : the name of the AQ derivative
- `-530.0` : the first redox potential in V
- `-1178.5`: the second redox potential in V

28.1.2 Data Path:

https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exercises/Anthraquinone/

28.1.3 Task

- Load the data from the files `experimental.dat` and `theory.dat`.
- Have fist a look at the data.
- Create a correlation plot of the experimental and theoretical data.
- Perform a linear regression of the data.
- Calculate the correlation coefficient.
- Check for outliers in the data using the z-scores method.

28.1.4 Questions

- How well do the theoretical data reproduce the experimental data?
- Are there any outliers in the data?

29 Exercise B 3

30 Exercise:

30.1 Arrhenious Plot

In this exercise, we will repeat linear regressions and how to use the logarithm of the

- We will learn how to use create `np.log`.
- We will learn how to select a specific range of data.
- We will learn how to use `scipy.linregress` to perform linear regression.

30.2 Exercise B.3 – Arrhenius Curve Fitting

In this exercise we have a look at diffusion data (either from experiment or theoretical calculations) and want to obtain the activation energy E_A.

Many dynamical properties in chemistry (such as rate constants, diffusion, etc.) follow an Arrhenian temperature dependency given as

$$y(T) = y_0 e^{-\frac{E_A}{RT}}$$

with y_0 being the pre-exponential factor, R and T are the molar gas constant (8.3145 J mol^{-1} K $^{-1}$) and temperature, respectively.

One potential option to obtain E_A is a non-linear exponential fit, but this is known to be less reliable than its linear counterpart!

Consequently, Mr. Svante Arrhenius used a linearization of the equation, which in case of the diffusion coefficient looks like this:

$$ln(D) = ln(D_0) - \frac{E_A}{R}\frac{1}{T}$$

This corresponds to a linear equation

$$y = a \cdot x + b$$

$$y = ln(D)$$

$$x = \frac{1}{T}$$

Applying standard linear regression we obtain

$$a = ln(D_0)$$
$$b = -\frac{E_A}{R}$$

From this we can directly access the activation energy E_A via:

$$E_A = -a \cdot R$$

Easy! :D

This time the files are in csv-format (= comma separated values), *i.e.* the different data columns are separated by comma symbols.

Luckily, we can again use the command *np.loadtext()*, but we have to indicate the comma by adding *delimiter = ","*.

Most programs such as Excel, Origin and scientific software can write data sets in this format. If you want to use python in your research, this is most likely the most common file format to input you data sets.

30.2.1 Data

We have three data sets with diffusion coefficients D in nm ^{2} /ps at different temperatures T in K.

- `D_vs_T_v1.csv` (Diffusion coefficient vs. temperature)
- `D_vs_T_v2.csv` (Diffusion coefficient vs. temperature)
- `D_vs_T_v3.csv` (Diffusion coefficient vs. temperature)

30.2.2 Data Path:

https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exercises/Arrhenius/

30.2.3 Task

- Load the data from the files `D_vs_T_v1.csv`, `D_vs_T_v2.csv` and `D_vs_T_v3.csv` into numpy arrays.
- Create a plot of the diffusion coefficient D vs. temperature T.
- Create a plot of the logarithm of the diffusion coefficient $ln(D)$ vs. $1/T$.
- Perform a linear regression of the data using `scipy.stats.linregress` only in the linear region of the data.
- Calculate the activation energy E_A from the slope of the linear regression.

30.2.4 Questions

- Which data set has the highest activation energy?

Part VIII

Data Visualization 2

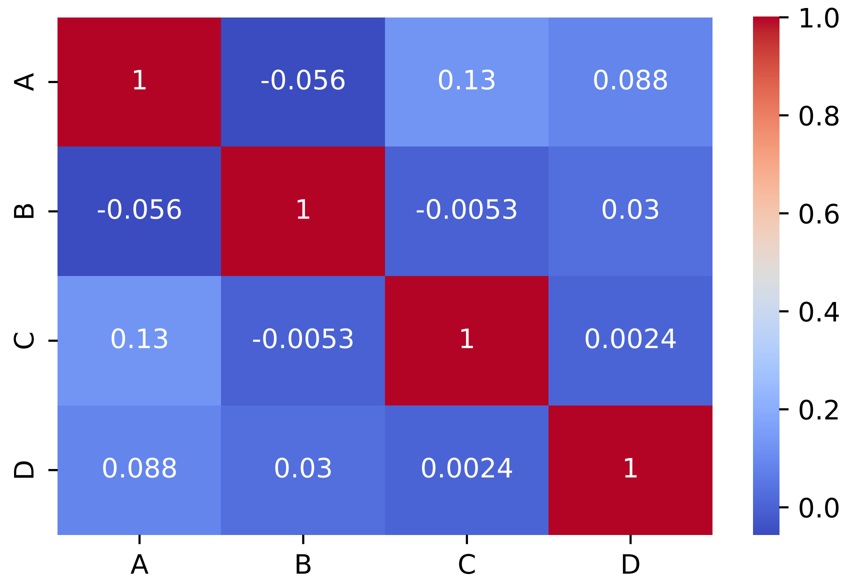
31 Advanced Plot Types

Difficulty level:

- Network plot: It is used to show the relationship between different nodes. It is used to show the relationship between different entities *e.g.* protein-protein interaction network PPI.

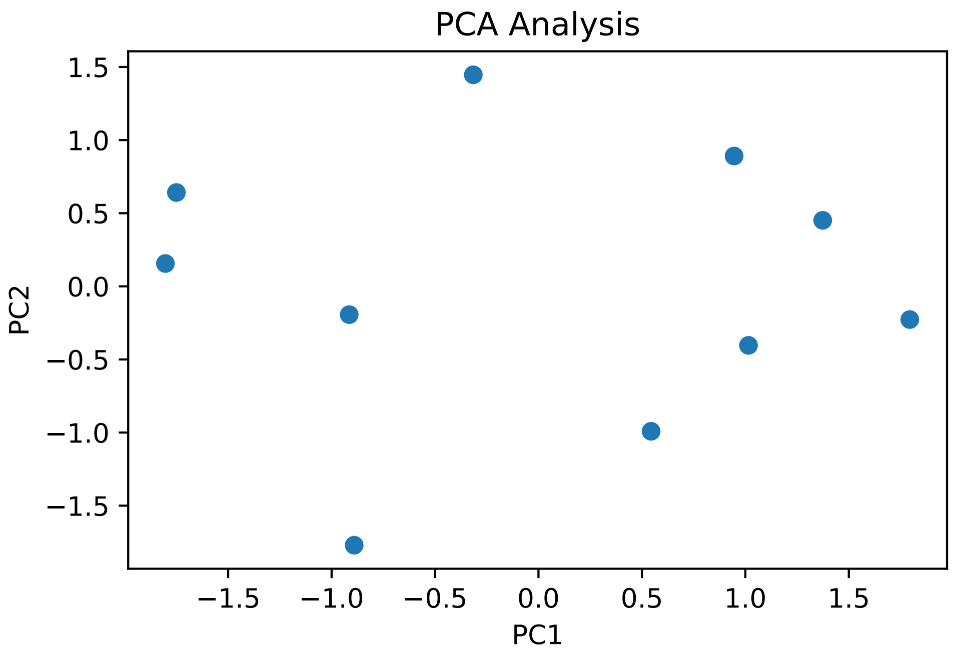
31.1 Multiple Correlation Analysis

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

rs = np.random.RandomState(33)
df = pd.DataFrame(data=rs.normal(size=(100, 4)), columns=['A', 'B', 'C', 'D'])


# Compute correlation matrix
corr_matrix = df.corr()

# Visualizing with a heatmap
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")

plt.show()
```



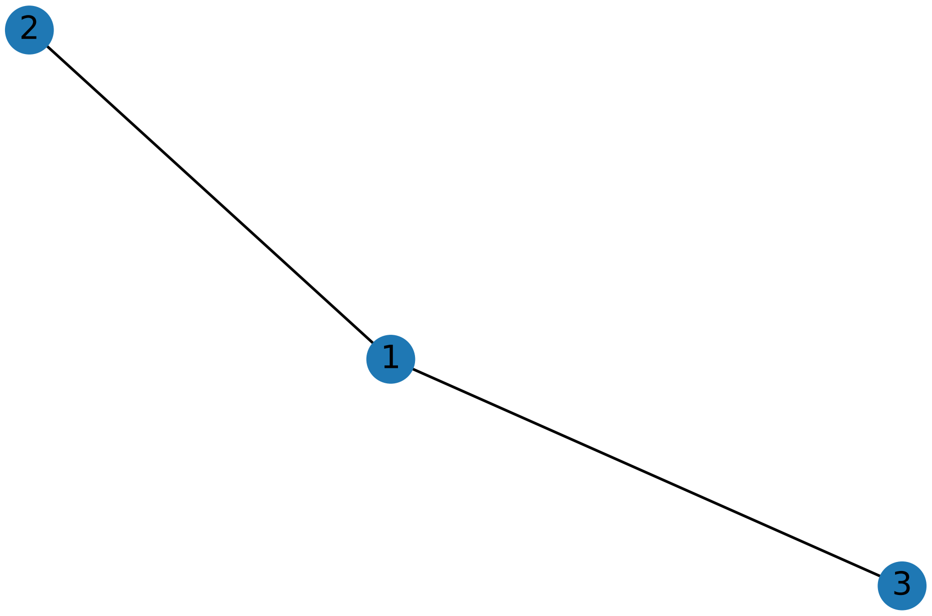
31.2 PCA Plot

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Sample dataset
X = np.random.rand(10, 3)

# Standardizing Data
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

# Applying PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot PCA results
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Analysis')
plt.show()
```



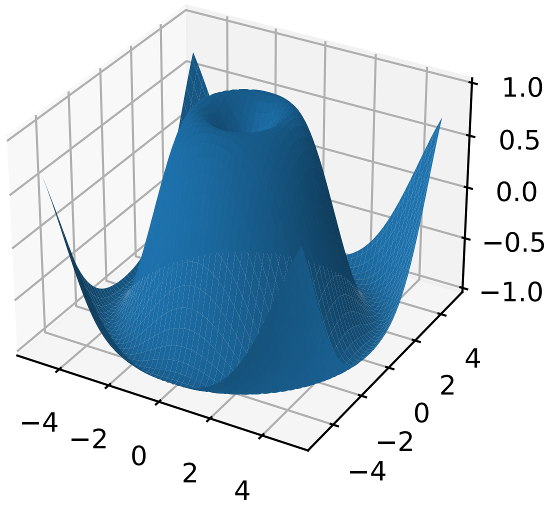
31.3 Network Plot

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node(1)
G.add_nodes_from([2, 3])
G.add_edge(1, 2)
G.add_edges_from([(1, 2), (1, 3)])
nx.draw(G, with_labels=True)
plt.show()
```



31.4 3D Plot

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z)
plt.show()
```


1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4
4
2
0
−2
−4

32 Interactive Plots

Difficulty level:

Interactive Plots

Matplotlib and ipywidget

```python
from matplotlib import pyplot as plt
import numpy as np
import ipywidgets as widgets
from IPython.display import display

x = np.linspace(0, 10, 100)
y = np.sin(x)

def plot_sine(frequency):
    plt.plot(x, np.sin(frequency*x))
    plt.show()

frequency_slider = widgets.FloatSlider(value=1, min=0.1, max=10, step=0.1)
widgets.interactive(plot_sine, frequency=frequency_slider)
```

```
interactive(children=(FloatSlider(value=1.0, description='frequency', max=10.0, min=0.1), Ou
```

Plotly

Ploty is a library that allows you to create interactive plots and dashboards, see the Plotly website.

```python
import plotly.express as px
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

fig = px.line(x=x, y=y, title='Sine function')
fig.show()
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html
```

Bokeh

Bokeh is a library that allows you to create interactive plots and dashboards, see the Bokeh website.

```python
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
import numpy as np

output_notebook()

x = np.linspace(0, 10, 100)
y = np.sin(x)

p = figure(title='Sine function')
p.line(x, y)
show(p)
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_lo

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

Altair

Altair is a library that allows you to create interactive plots and dashboards, see the Altair website.

```python
import altair as alt
import pandas as pd
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

df = pd.DataFrame({'x': x, 'y': y})

alt.Chart(df).mark_line().encode(
    x='x',
    y='y'
).properties(
    title='Sine function'
)
```

```
alt.Chart(...)
```

Part IX

Exercises

33 Exercise D 1

34 Exercise:

34.1 3D Plot

In this exercise, we will repeat to plot the 3D data.

34.1.1 Data

The 3D data is given in the file `grid_data.dat`. The data contains the following columns: - r_{OH_1} distance from O to H1 atom in Angstrom - r_{OH_2} distance from O to H2 atom in Angstrom - V potential energy in kcal/mol

34.1.2 Data Path:

```
data_path = "https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exe
```

34.1.3 Task

- Load the data from the files `grid_data.dat`.
- Create a 3D plot of the data using the `plot_surface` function.
- Use a meaningful color map.

35 Exercise D 2

36 Exercise:

36.1 Interactive Plot

In this exercise, we will repeat how to create an interactive plot using `ipywidgets`.

36.1.1 Task 1

- Create an interactive plot using `ipywidgets` and `matplotlib` for the van-Deemeter equation where the user can change the parameters a, b and c using sliders.

36.1.2 Task 1

- Create an interactive plot using `ipywidgets` and `matplotlib` for a rate equation together with an Arrhenius Plot where the user can change the parameters k_0,and E_a using sliders.

Part X

Special Analysis and Visualization Techniques

37 Analysis of Spectroscopy Data

Difficulty level:

Spectroscopy Data Analysis

Spectroscopy is one of the most used experimental techniques in chemistry *e.g.*:

- UV/Vis spectroscopy
- IR spectroscopy
- NMR spectroscopy
- Mass spectroscopy

The analysis of spectroscopy needs often some preprocessing steps *e.g.*:

- Baseline correction
- Smoothing
- Normalization
- Peak assignment
- Peak fitting
- Peak integration
- Peak deconvolution

Feature Detection

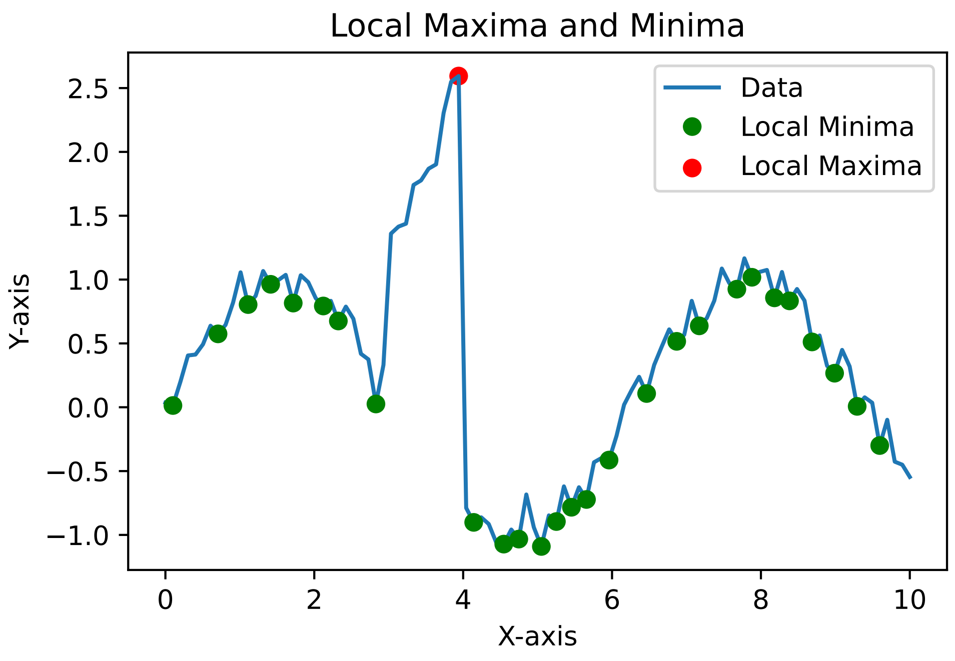
In spectroscopy data, peaks are often the most important features.

Maxima and minima are often used to identify the peaks in the data.

Global maxima and minima can be found using the `min` and `max`, `np.min` and `np.max` or `df['column_name'].min()` and `df['column_name'].max()` functions.

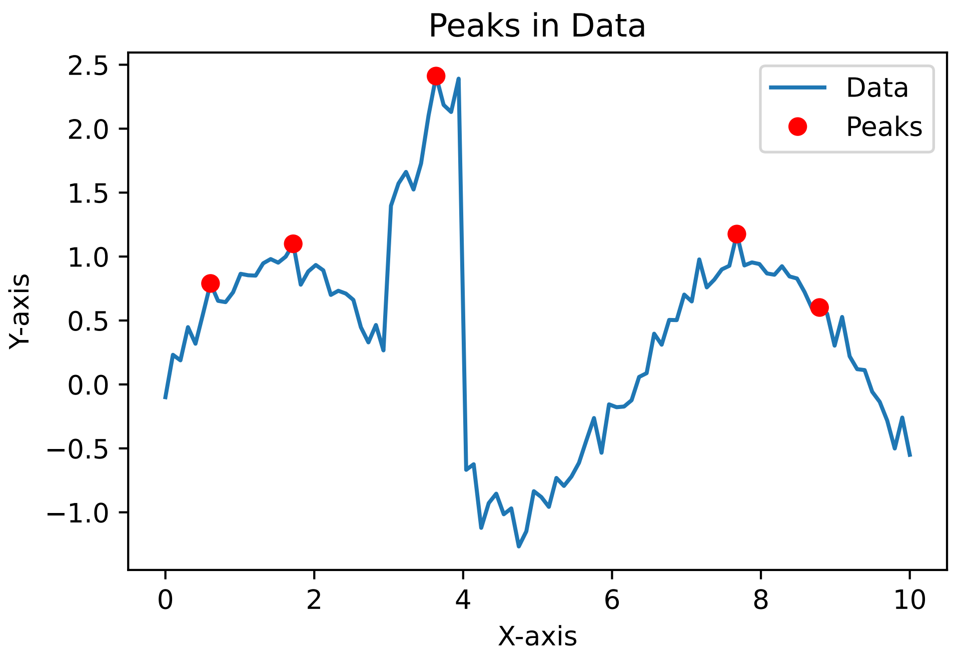
Local maxima and minima can be found also by using the `argrelextrema` function from the `scipy.signal` module.

```
import numpy as np
from scipy.signal import argrelextrema
import matplotlib.pyplot as plt

# Generate some data transmission data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
y[30:40] = np.exp(-y[30:40]) + 0.5

# Find local maxima and minima
local_max = argrelextrema(y, np.greater,order=200)[0]
local_min = argrelextrema(y, np.less)[0]

# Plot the data
plt.plot(x, y, label='Data')
plt.plot(x[local_min], y[local_min], 'o', color='green', label='Local Minima')
plt.scatter(x[local_max], y[local_max], color='red', label='Local Maxima')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Local Maxima and Minima')

plt.legend()

plt.show()
```



`find_peaks` function from the `scipy.signal` module can be used to find the peaks in the data. It finds the local maxima in the data and returns the indices of the peaks. If you want to find the minima, you can simply invert the data by multiplying it with -1.

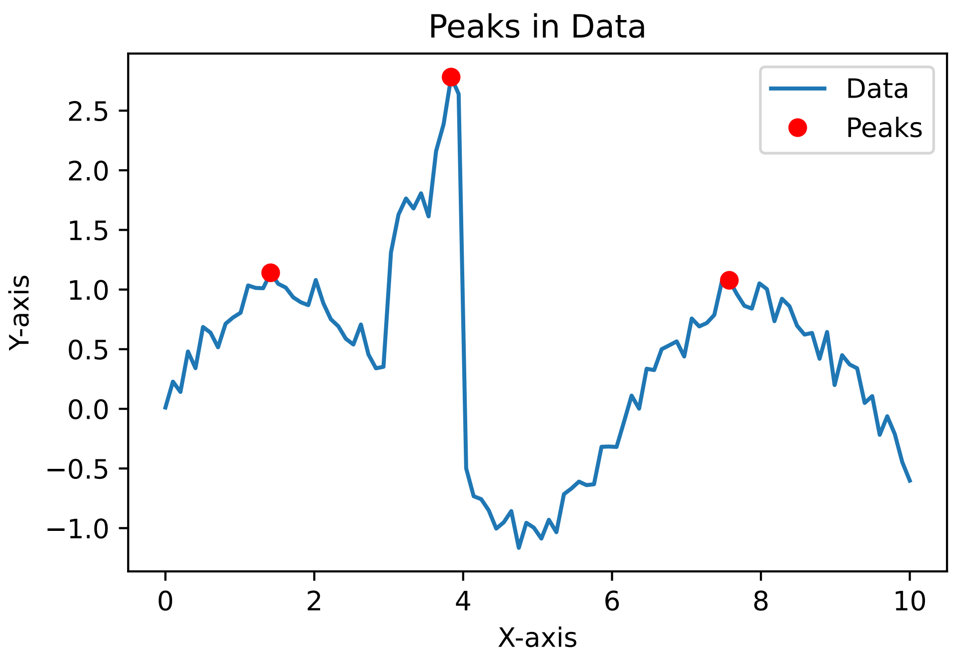
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
y[30:40] = np.exp(-y[30:40]) + 0.5
# Find peaks
peaks, _ = find_peaks(y, height=0.5, distance=10)
# Plot the data
plt.plot(x, y, label='Data')
plt.plot(x[peaks], y[peaks], 'o', color='red', label='Peaks')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Peaks in Data')
plt.legend()
plt.show()
```



You can also use the **height**, **threshold**, **distance**, **prominence** and **width** parameters to filter the peaks to enhance the peak detection.

- `height`: Minimum height of the peaks
- `threshold`: Minimum vertical distance to its neighboring samples
- `distance`: Minimum horizontal distance (in samples) between neighboring peaks
- `prominence`: Minimum prominence of the peaks

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
y[30:40] = np.exp(-y[30:40]) + 0.5
# Find peaks
peaks, _ = find_peaks(y, height=0.5, distance=10, prominence=0.5, width=1)
# Plot the data
plt.plot(x, y, label='Data')
plt.plot(x[peaks], y[peaks], 'o', color='red', label='Peaks')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Peaks in Data')
plt.legend()
plt.show()
```



Smoothing

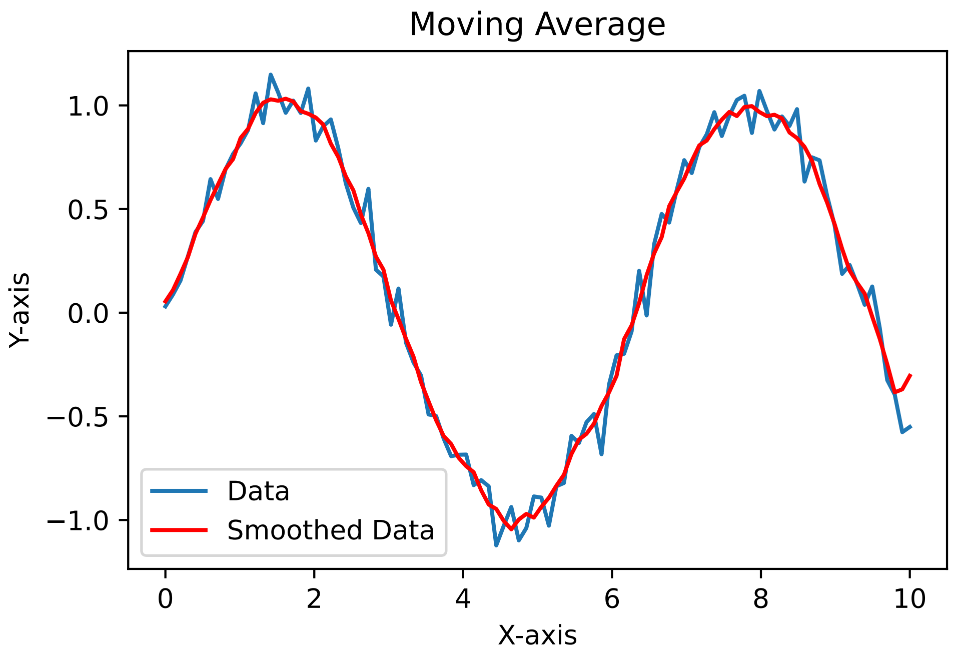
Often data is noisy and needs to be smoothed before further analysis.

- Moving average
- Weighted moving average
- Gaussian filter
- Savitzky-Golay filter

Moving Average

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


# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
```

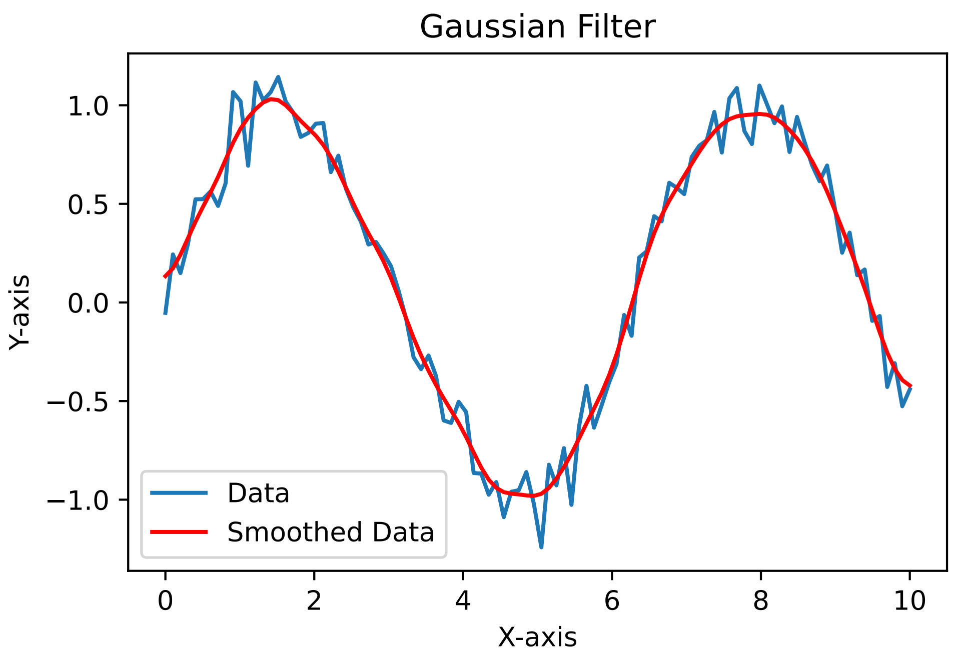
```
# Apply the moving average
window_size = 5

smoothed_data =np.convolve(y, np.ones(window_size)/window_size, mode='same')

# Plot the data
plt.plot(x, y, label='Data')
plt.plot(x, smoothed_data, label='Smoothed Data', color='red')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Moving Average')
plt.legend()
plt.show()
```



Gaussian Filter

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
```

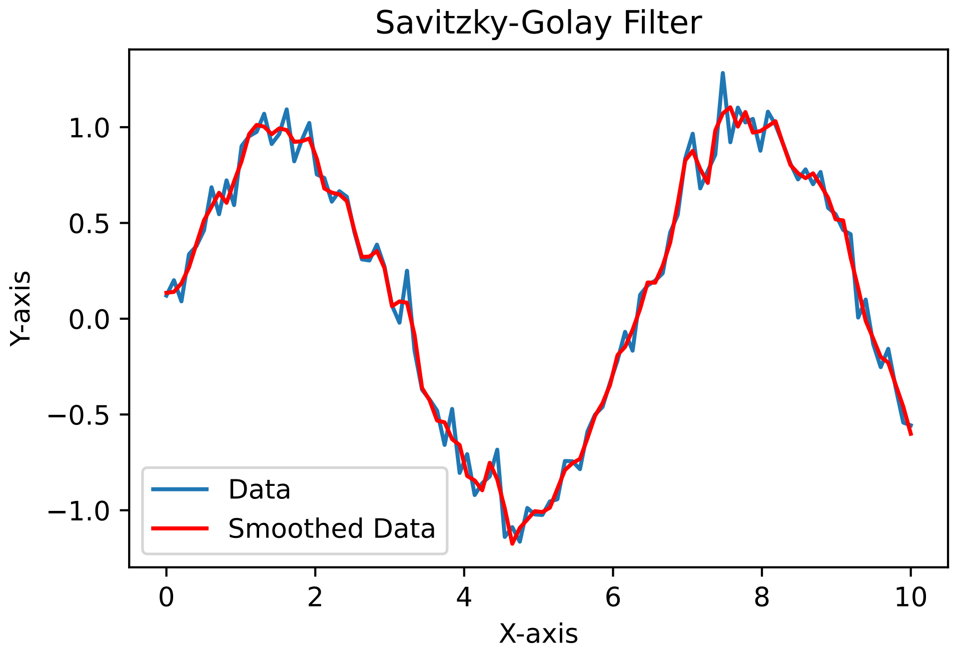
```
# Apply the Gaussian filter
sigma = 2
smoothed_data = gaussian_filter1d(y, sigma=sigma)
# Plot the data
plt.plot(x, y, label='Data')

plt.plot(x, smoothed_data, label='Smoothed Data', color='red')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Gaussian Filter')
plt.legend()
plt.show()
```



Savitzky-Golay Filter

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter
# Generate some data
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.1, x.size)
# Apply the Savitzky-Golay filter
```

```
window_size = 5
poly_order = 2
smoothed_data = savgol_filter(y, window_size, poly_order)
# Plot the data
plt.plot(x, y, label='Data')
plt.plot(x, smoothed_data, label='Smoothed Data', color='red')
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
plt.title('Savitzky-Golay Filter')
plt.legend()
plt.show()
```



Baseline Correction

A baseline correction is often needed to remove background noise from the signal and

The baseline correction can be done by different methods *e.g.*:

- Polynomial fitting
- Spline fitting
- Minimum value fitting
- Moving average etc.

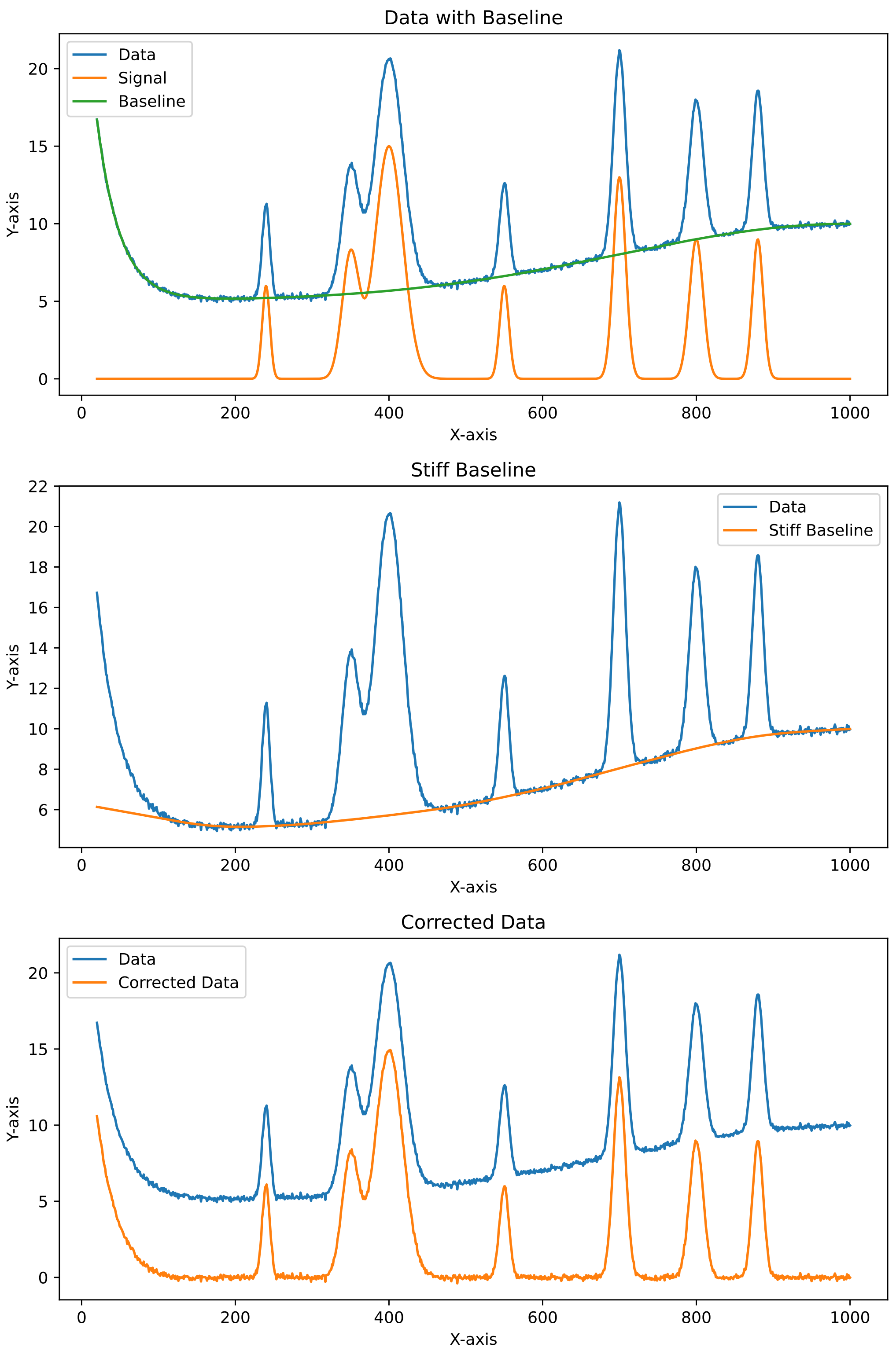
There exists also a package called `pybaselines` which can be used to correct the baseline of the data.

```
!pip install pybaselines
```

```
Requirement already satisfied: pybaselines in /Users/stk/y/envs/myenv/lib/python3.12/site-pa
Requirement already satisfied: numpy>=1.20 in /Users/stk/y/envs/myenv/lib/python3.12/site-pa
Requirement already satisfied: scipy>=1.6 in /Users/stk/y/envs/myenv/lib/python3.12/site-pac
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
from pybaselines import Baseline
from pybaselines.utils import gaussian


# Generate some data
x = np.linspace(20, 1000, 1000)
signal = (
    + gaussian(x, 6, 240, 5)
    + gaussian(x, 8, 350, 11)
    + gaussian(x, 15, 400, 18)
    + gaussian(x, 6, 550, 6)
    + gaussian(x, 13, 700, 8)
    + gaussian(x, 9, 800, 9)
    + gaussian(x, 9, 880, 7)
)
baseline = 5 + 6 * np.exp(-(x - 40) / 30) + gaussian(x, 5, 1000, 300)
noise = np.random.default_rng(0).normal(0, 0.1, len(x))
y = signal + baseline + noise

# Use the Baseline class to fit the baseline
baseline_fitter = Baseline(x_data=x)
# The baseline_fitter object can be used to fit the baseline with different methods in that
stiff_baseline = baseline_fitter.arpls(y, lam=5e5)[0]
# Correct the data by subtracting the baseline
data_corrected = y - stiff_baseline

# Create subplots
fig, axs = plt.subplots(3, 1, figsize=(8, 12))
axs[0].plot(x, y, label='Data')
axs[0].plot(x, signal, label='Signal')
```

```
axs[0].plot(x, baseline, label='Baseline')
axs[0].set_ylabel('Y-axis')
axs[0].set_xlabel('X-axis')
axs[0].set_title('Data with Baseline')
axs[0].legend()
axs[1].plot(x, y, label='Data')
axs[1].plot(x, stiff_baseline, label='Stiff Baseline')
axs[1].set_ylabel('Y-axis')
axs[1].set_xlabel('X-axis')
axs[1].set_title('Stiff Baseline')
axs[1].legend()
axs[2].plot(x, y, label='Data')
axs[2].plot(x, data_corrected, label='Corrected Data')
axs[2].set_ylabel('Y-axis')
axs[2].set_xlabel('X-axis')
axs[2].set_title('Corrected Data')
axs[2].legend()
plt.tight_layout()
plt.show()
```


Data with Baseline
Data
Signal
Baseline
Y-axis
X-axis
Stiff Baseline
Data
Stiff Baseline
Y-axis
X-axis
Corrected Data
Data
Corrected Data
Y-axis
X-axis

38 Analysis of Image Data

Difficulty level:

Image Data Analysis

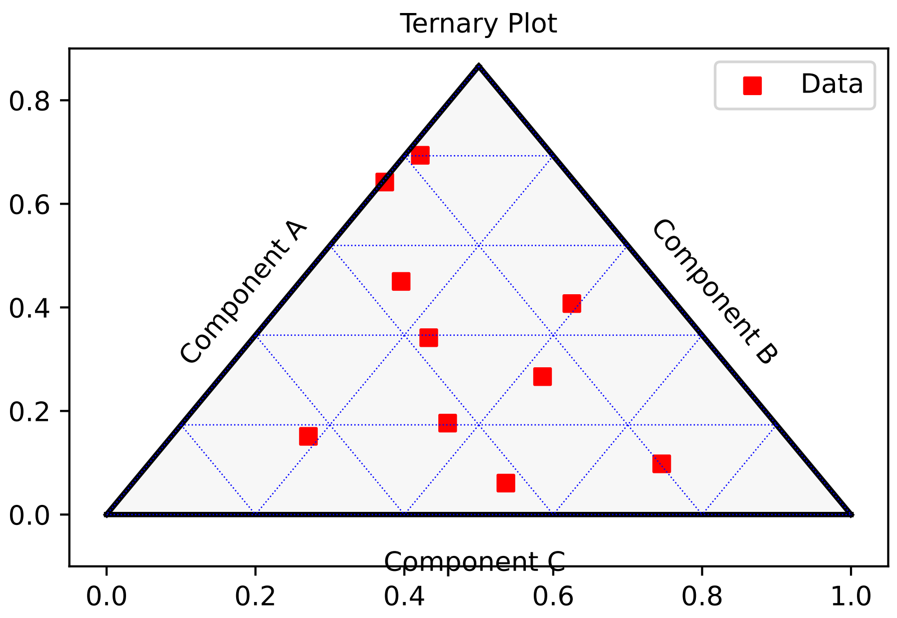
39 Ternary Plot

Difficulty level:

Ternary Plot

A ternary plot is a type of plot that is used to visualize the composition of three components.

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

# Create a figure
fig, tax = ternary.figure(scale=1.0)

# Draw Boundary and Gridlines
tax.boundary(linewidth=2.0)
tax.gridlines(multiple=0.2, color="blue")

# Set Axis labels and Title
fontsize = 10
tax.set_title("Ternary Plot", fontsize=fontsize)
tax.left_axis_label("Component A", fontsize=fontsize)
tax.right_axis_label("Component B", fontsize=fontsize)
tax.bottom_axis_label("Component C", fontsize=fontsize)

# Plot some data
points = np.random.dirichlet((1, 1, 1), 10)
tax.scatter(points, marker='s', color='red', label="Data")

# Legend
tax.legend()
plt.show()
```



Ternary Plot
Data
Component A
Component B
Component C
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
1.0

Part XI

Exercises

40 Exercise C 1

41 Exercise:

41.1 Spectroscopic data analysis

In this exercise, we will repeat how spectroscopic data can be analyzed effectively via Python `SciPy`library.

- We will learn denoising and smoothing of data (SM).
- We will learn how to correct the baseline of data (BC).
- We will learn how to detect peaks in data (PD).

These kind of analysis methods can be relevant when depicting X-ray diffraction patterns, spectrograms obtained from various techniques (e.g. IR/Raman, UV/Vis, X-ray, NMR, …), chromatograms and electropherograms and many more.

Often the experimental data is subject to noise or has a notable offset in the baseline.

`SciPy` offers a number of straightforward tools to quickly enhance the data to make it ready for plotting.

Let's start by reading in some X-ray diffraction data.

41.1.1 Data

The data is a simple X-ray diffraction pattern of a crystalline material. The experimental data is stored in `XRD_experimental.dat`and the theoretical data in `XRD_theory.dat`.

First column represents the 2-theta angle in degrees, the second column represents the intensity in arbitrary units.

41.1.2 Data Path:

```
data_path = 'https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exe
```

41.1.3 Task

- Load the data from the files `XRD_experimental.dat` and `XRD_theory.dat`.
- Have fist a look at the data.
- Normalize the data to the maximum intensity to 100.
- Smooth the experimental data using a Gaussian filter with a standard deviation of 2.0.
- Find the peaks in the experimental data using the `find_peaks` function from `scipy.signal`.
- Plot the smoothed experimental data.
- Add a baseline correction to the experimental data using the `minimum_fiter1d` function from `scipy.ndimage`.
- Plot the baseline corrected experimental data.
- Find the peaks in the baseline corrected data uing the `find_peaks` function from `scipy.signal`.
- Compare the smoothed experimental data with the smoothed and baseline corrected experimental data via a correlation plot and linear regression.
- Compare both experimental smoothed and baseline corrected data and theoretical data via a correlation plot and linear regression.

41.1.4 Questions

- What effect does the Gaussian filter have on the data?
- How does the baseline correction affect the data?
- How well does the smoothed data correlate with the theoretical data?

42 Exercise C 2

43 Exercise:

43.1 Chromotography and Spectroscopy Integration

In this exercise, we will repeat how spectroscopic data can be integrated via Python `SciPy`library.

- We will learn denoising and smoothing of data (SM).
- We will learn how to correct the baseline of data (BC).
- We will learn how to detect peaks in data (PD).
- We will learn how to integrate the area under the curve (AUC).

In this example we are using the featues of `finde-peaks()` to determine the left and right integration limits to integrate peaks of an example chromatogram.

To achieve this, we first have to reduce the noise and apply a baseline correction.

Let's start by loading the file!

43.1.1 Data

The chromatographic data is stored in a file called `Chromatogram.csv`.

First column represents the time in min and the second column represents the intensity in arbitrary units (a.u.).

43.1.2 Data Path:

```
data_path = "https://raw.githubusercontent.com/stkroe/PythonForChemists/main/course/data/exe
```

43.1.3 Task

- Load the data from the files `Chromatogram.csv`.
- Apply Gaussian smoothing and minimum filtered baseline correction.
- Detect the peaks and integrate the area under the curve.
- Plot the chromatogram with the detected peaks and the integrated area under the curve.

43.1.4 Questions

- What is the area under the curve for each peak?

Part XII

Playground

44 Exercise Playground

45 Playground for Python

You can use the `load_wine` dataset from the `sklearn` library to practice data manipulation and visualization. Below is a code snippet that demonstrates how to load the dataset, perform some basic data analysis, and visualize it using `matplotlib`.

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
```

The `load_wine` dataset contains information about different types of wines, including their chemical properties and quality ratings. You can use this dataset to practice various data manipulation techniques, such as filtering, grouping, and aggregating data.

```python
data = load_wine()
data
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2]),
 'frame': None,
 'target_names': array(['class_0', 'class_1', 'class_2'], dtype='<U7'),
 'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\n------------------------\n\n**Data
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'flavanoids',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity',
  'hue',
  'od280/od315_of_diluted_wines',
  'proline']}
```

```python
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df['targe_name'] = df['target'].map({0: 'Class 0', 1: 'Class 1', 2: 'Class 2'})
df.style \
  .format(precision=2) \
  .highlight_max(axis=0, color='lightgreen') \
  .highlight_min(axis=0, color='lightcoral') \
  .set_caption("Wine Dataset Features") \
  .set_table_styles(
    [{'selector': 'th.col0', 'props': [('color', 'blue'), ('font-weight', 'bold')]}]
  ) \
  .set_properties(**{'text-align': 'left'}) \
  .set_table_attributes('style="width: 100%; border: 1px solid black;"') \
  .set_table_styles(
    [{'selector': 'th', 'props': [('background-color', '#f2f2f2'), ('color', 'black')]}],
    axis=0
  )
display(df)
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_ph
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_ph
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39
...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56

What you can do with this dataset:

- Load the dataset and explore its structure
- Perform basic data analysis, such as calculating summary statistics
- Visualize the data using different types of plots (e.g., scatter plots, histograms)

Part XIII

Repetition

46 Exercise Recap Python

46.0.1 Quiz

Exercise Recap Python

Open it locally:

Exercise Recap Python

Open it in Colab: Exercise Recap Python

Part XIV

Exam

47 Final Exam

48 1. Temperature vs time plot

First data set: `ExamA_XX.dat`

Temperature in K | Time in seconds

- Draw a line plot of the temperature(y-axis) in K vs time(x-axis) in hours.
- Calculate the average temperature and the standard deviation. Plot both values as horizontal lines in the plot.
- Create a figure which is large enough for publication, label everything, and use a suitable font size for the labels, legend, x-ticks and y-ticks.

49 2. Correlation plot

Second data set: `ExamB_XX.csv`

Experiment in eV | Theory in eV

- Draw a scatter plot of the experimental(y-axis) in eV vs theoretical(x-axis) in eV.
- Calculate the correlation coefficient and the linear regression parameter.
- Draw the linear regression line in the plot.
- Give the parameters and R ^{2} of the linear regression line in the plot.
- Create a figure which is large enough for publication, label everything, and use a suitable font size for the labels, legend, x-ticks and y-ticks.

50 3. X-ray diffraction plot

Third data set: `ExamC_XX.csv`

2 θ in degrees | Intensity in arbitrary units

- Draw a line plot of the intensity(y-axis) in arbitrary units vs 2 theta(x-axis) in degrees.
- Get the peak positions and draw text boxes with the peak positions in the plot.
- Create a figure which is large enough for publication, label everything, and use a suitable font size for the labels, legend, x-ticks and y-ticks.

Part XV

Informations