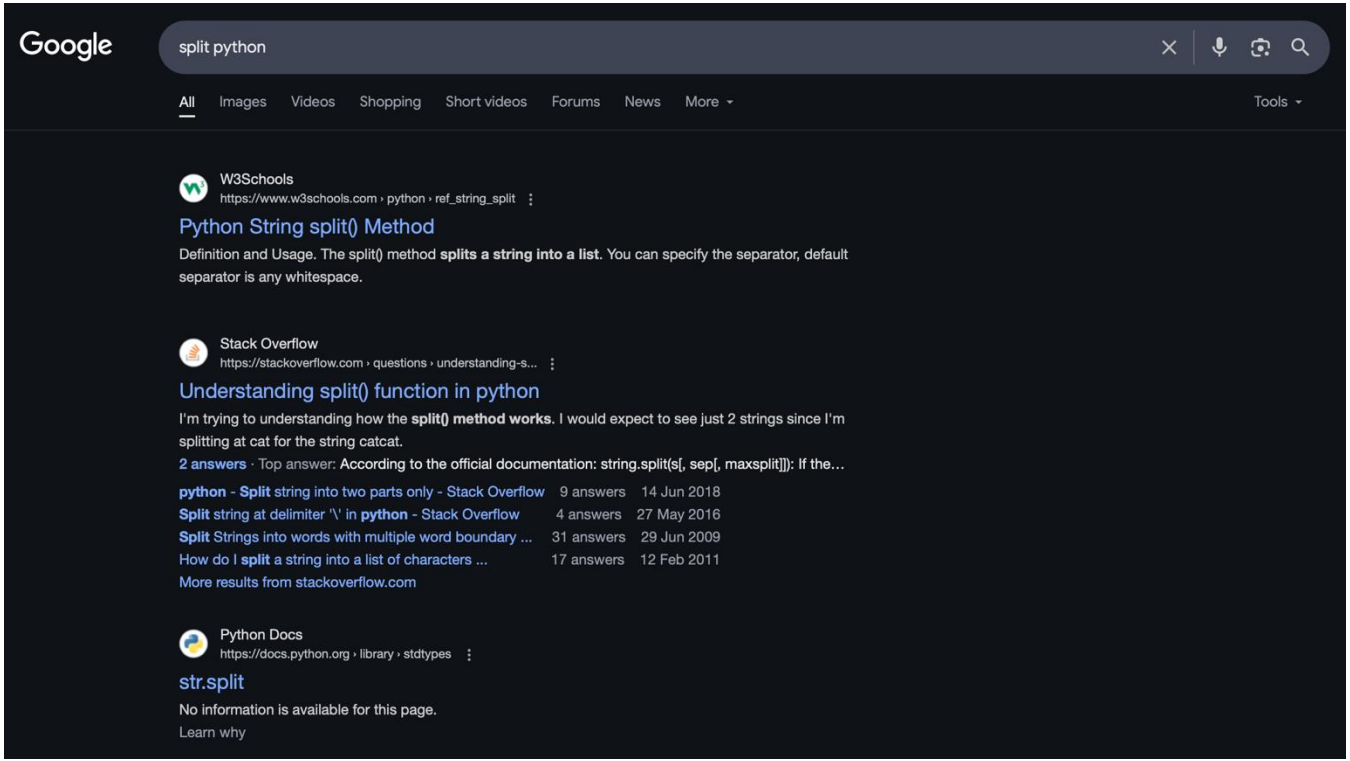
Pandas, Matplotlib, seaborn

Practical Programming in Chemistry

Prof. Philippe Schwaller

Last week

Python functions, classes

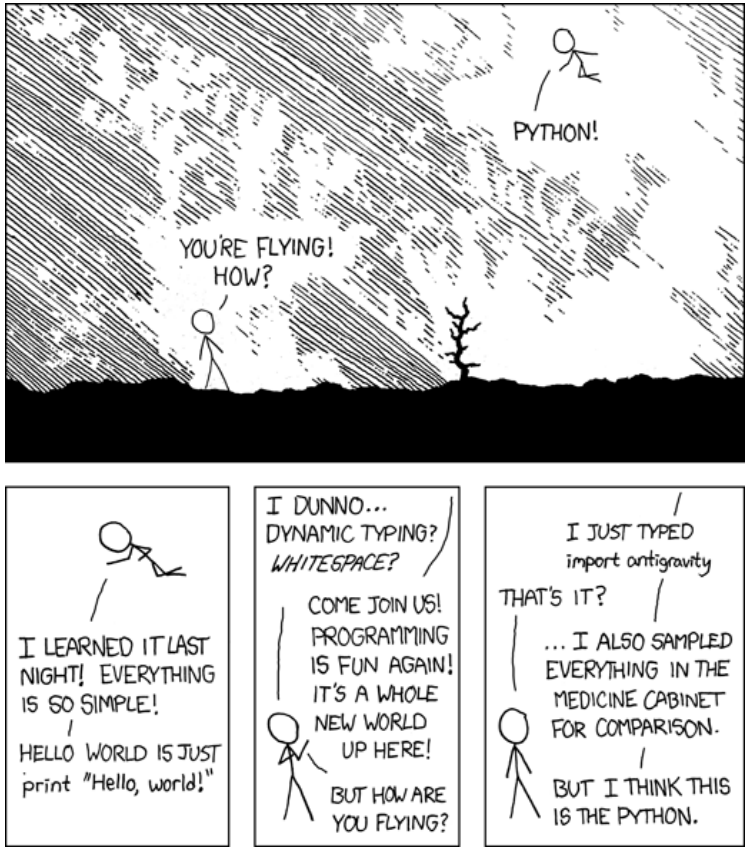


This lecture

Python packages (free functionality, a pip install away!)

- Numpy
- Pandas
- Matplotlib
- Seaborn (optional)
- (scipy, one exercise, optional)
- If you want to run the code of the examples shown in this lecture (pandas, matplotlib, seaborn):

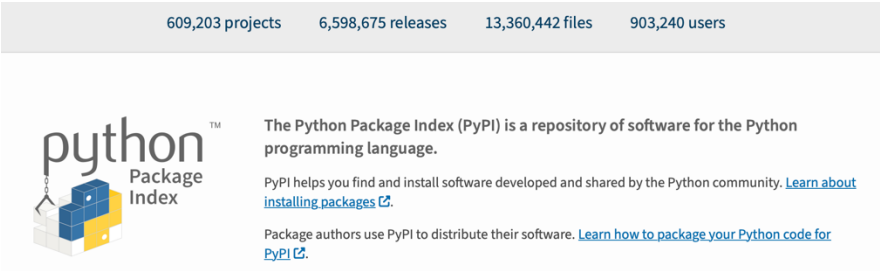
https://schwallergroup.github.io/practical-programming-in-chemistry/tutorials/lecture_04/01_pandas.html


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

XKCD

How to install packages in Python?

- Through package managers, the two most common are pypi and conda.



pip install numpy

to install the numpy package

ANACONDA®

conda install numpy

to install the numpy package

External packages, e.g. Numpy

- Numpy (Python's equivalent to Matlab)

```
import numpy as np
```

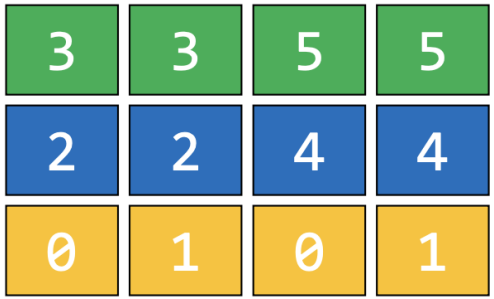
```
from numpy import *
```

This might overwrite some other functions, e.g. from math → sum.
Hard to debug, as you might not know, what functions is called.

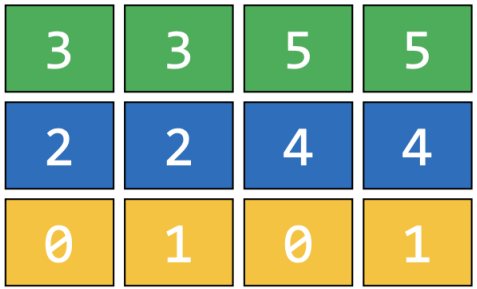
Creating arrays

```
arr1d = np.array([2,2,4,4])
print(arr1d)
```


```
arr2d = np.array([[3,3,5,5],
                  [2,2,4,4],
                  [0,1,0,1]])
print(arr2d)
```



arr2d




```
print(arr2d.size)  # 12 elements
print(arr2d.ndim)  # 2 dimensions
print(arr2d.shape) # (3,4)
```

np.ones(), np.zeros(), np.full()

array shape

```
arr1 = np.ones(3)
```

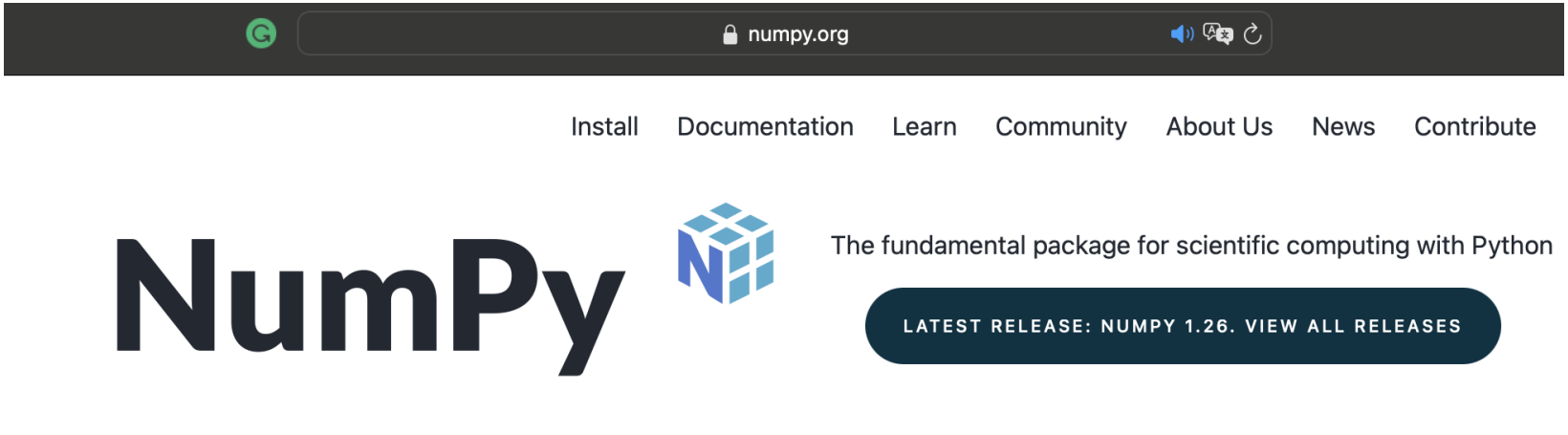

```
arr0 = np.zeros(3)
```


```
arrf = np.full(shape=(2,3), fill_value=9)
# or, in short
# arr3 = np.full((2,3), 9)
```

Numpy – manupulating arrays

(brief intro, you'll learn much more in the exercises)

How to get numpy?



- both package managers pip and conda coexist
- if available via pip, I tend to use it as it's the officially recommended one, and simplicity

Installing in a running notebook

- You can access the command line using “!” in front of your command

```
# in the command line
pip install numpy

# in a jupyter notebook, use ! to access the command line
!pip install numpy
```

When you install from the command line, make sure you are in the **correct conda environment**.

Functionality to manipulate arrays

- https://numpy.org/doc/stable/reference/routines.array-manipulation.html

Changing array shape

`reshape`(a, newshape[, order])	Gives a new shape to an array without changing its data.
`ravel`(a[, order])	Return a contiguous flattened array.
`ndarray.flat`	A 1-D iterator over the array.
`ndarray.flatten`([order])	Return a copy of the array collapsed into one dimension.

Joining arrays

`concatenate`([axis, out, dtype, casting])	Join a sequence of arrays along an existing axis.
`stack`(arrays[, axis, out, dtype, casting])	Join a sequence of arrays along a new axis.
`block`(arrays)	Assemble an nd-array from nested lists of blocks.
`vstack`(tup, *[, dtype, casting])	Stack arrays in sequence vertically (row wise).
`hstack`(tup, *[, dtype, casting])	Stack arrays in sequence horizontally (column wise).
`dstack`(tup)	Stack arrays in sequence depth wise (along third axis).
`column_stack`(tup)	Stack 1-D arrays as columns into a 2-D array.
`row_stack`(tup, *[, dtype, casting])	Stack arrays in sequence vertically (row wise).

Advanced slicing (start:stop:step)

```
>>> x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> x[1:7:2]
array([1, 3, 5])
```

Filtering arrays

Selecting all non-nan values (nan= Not a Number)

```
>>> x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])
>>> x[~np.isnan(x)]
array([1., 2., 3.])
```

Adding a constant to specific elements (here x < 0)

```
>>> x = np.array([1., -1., -2., 3])
>>> x[x < 0] += 20
>>> x
array([ 1., 19., 18., 3.])
```

https://numpy.org/doc/stable/user/basics.indexing.html#

Why can I just not use lists instead of arrays?

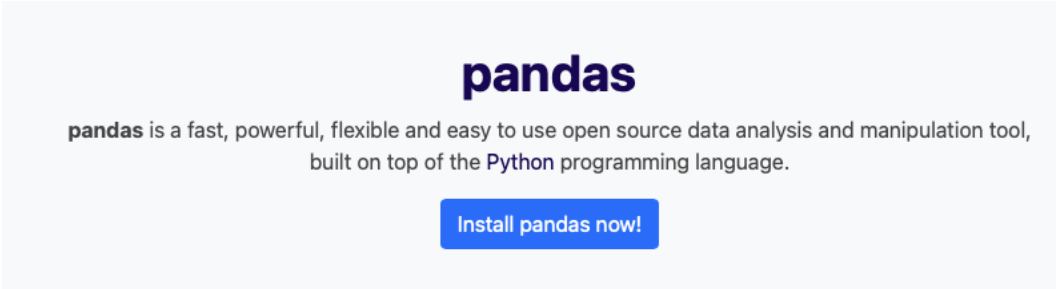
```
In [67]: masses = [5, 10, 20]  # grams
    ...: molar_masses = [58.44, 18.015, 46.07]  # Sodium Chloride, Water, Ethanol
    ...: volumes = [0.1, 0.2, 0.3]  # liters
    ...:
    ...: molarity = masses / molar_masses / volumes  # mol/L
```

```
TypeError                                 Traceback (most recent call last)
Cell In[67], line 5
      2 molar_masses = [58.44, 18.015, 46.07]  # Sodium Chloride, Water, Ethanol
      3 volumes = [0.1, 0.2, 0.3]  # liters
----> 5 molarity = masses / molar_masses / volumes  # mol/L

TypeError: unsupported operand type(s) for /: 'list' and 'list'
```

```
In [69]: import numpy as np

In [70]: masses = np.array([5, 10, 20])  # grams
    ...: molar_masses = np.array([58.44, 18.015, 46.07])  # Sodium Chloride, Water, Ethanol
    ...: volumes = np.array([0.1, 0.2, 0.3])  # liters
    ...:
    ...: molarity = masses / molar_masses / volumes  # mol/L

In [71]: molarity
Out[71]: array([0.85557837, 2.77546489, 1.44707329])
```

Pandas – handling tabular data with Python

Pandas – the way to handle tabular data in Python



How to install? → simply use “pip install pandas” in your conda environment.

How to use?

```
In [72]: import numpy as np
    ...: import pandas as pd
```

Series and dataframes

```
[2]: compounds = pd.Series(["Water", "Ethanol", "Glucose", "Sodium Chloride", "Methane"])
     compounds
```

```
[2]: 0              Water
     1            Ethanol
     2            Glucose
     3    Sodium Chloride
     4            Methane
     dtype: object
```

Series are one-dimensional array-like object,
→ Imagine single columns in an Excel spreadsheet

```
[2]: data = {
         "Compound": ["Water", "Ethanol", "Glucose", "Sodium Chloride", "Methane"],
         "Molecular Weight": [18.015, 46.07, 180.16, 58.44, 16.04],
         "Melting Point (°C)": [0, -114.1, 146, 801, -182.5]
     }
     df = pd.DataFrame(data)
```

Dataframes are two-dimensional, tabular data structures with labeled axes (rows and columns). → spreadsheet

Basic Data Inspection for DataFrames (df)

```
[2]: data = {
         "Compound": ["Water", "Ethanol", "Glucose", "Sodium Chloride", "Methane"],
         "Molecular Weight": [18.015, 46.07, 180.16, 58.44, 16.04],
         "Melting Point (°C)": [0, -114.1, 146, 801, -182.5]
     }
     df = pd.DataFrame(data)
```

```
df.head()
```

	Compound	Molecular Weight	Melting Point (°C)
0	Water	18.015	0.0
1	Ethanol	46.070	-114.1
2	Glucose	180.160	146.0
3	Sodium Chloride	58.440	801.0
4	Methane	16.040	-182.5

```
df.head(2)
```

	Compound	Molecular Weight	Melting Point (°C)
0	Water	18.015	0.0
1	Ethanol	46.070	-114.1

First few rows of the dataframe (default: 5)

Basic Data Inspection for DataFrames (df)

```
[2]: data = {
         "Compound": ["Water", "Ethanol", "Glucose", "Sodium Chloride", "Methane"],
         "Molecular Weight": [18.015, 46.07, 180.16, 58.44, 16.04],
         "Melting Point (°C)": [0, -114.1, 146, 801, -182.5]
     }
     df = pd.DataFrame(data)
```

```
df.describe()
```

	Molecular Weight	Melting Point (°C)
count	5.000000	5.000000
mean	63.745000	130.080000
std	67.564785	395.170225
min	16.040000	-182.500000
25%	18.015000	-114.100000
50%	46.070000	0.000000
75%	58.440000	146.000000
max	180.160000	801.000000

Summary statistics for numerical columns

```
df.dtypes
```

```
Compound              object
Molecular Weight     float64
Melting Point (°C)   float64
dtype: object
```

Types of the columns

Accessing the data in a df


```
mw = df["Molecular Weight"]
mw
```

```
0     18.015
1     46.070
2    180.160
3     58.440
4     16.040
Name: Molecular Weight, dtype: float64
```

Access the “Molecular Weight” column

```
solids = df[df["Melting Point (°C)"] > 25]
solids
```

	Compound	Molecular Weight	Melting Point (°C)
2	Glucose	180.16	146.0
3	Sodium Chloride	58.44	801.0

Filter by melting point > 25

```
df["Melting Point (°C)"] > 25
```

```
0    False
1    False
2     True
3     True
4    False
Name: Melting Point (°C), dtype: bool
```

Indexing – loc and iloc

```
df.index = ['A', 'B', 'C', 'D', 'E']
```

```
df
```

	Compound	Molecular Weight	Melting Point (°C)
A	Water	18.015	0.0
B	Ethanol	46.070	-114.1
C	Glucose	180.160	146.0
D	Sodium Chloride	58.440	801.0
E	Methane	16.040	-182.5

- **.loc** is **label-based**, meaning you use the labels of the rows and columns to select data.
- **.iloc** is **integer position-based**, so you use integer indices to select data.

```
df.loc['B']
```

```
Compound              Ethanol
Molecular Weight        46.07
Melting Point (°C)     -114.1
Name: B, dtype: object
```

```
df.iloc[1]
```

```
Compound              Ethanol
Molecular Weight        46.07
Melting Point (°C)     -114.1
Name: B, dtype: object
```

Indexing – loc and iloc


```
df.index = ['A', 'B', 'C', 'D', 'E']
```

```
df
```

	Compound	Molecular Weight	Melting Point (°C)
A	Water	18.015	0.0
B	Ethanol	46.070	-114.1
C	Glucose	180.160	146.0
D	Sodium Chloride	58.440	801.0
E	Methane	16.040	-182.5

```
# Select the Molecular Weight and Melting Point for Ethanol and Glucose
df.loc[['B', 'C'], ["Molecular Weight", "Melting Point (°C)"]]
```

	Molecular Weight	Melting Point (°C)
B	46.07	-114.1
C	180.16	146.0

Label-based

```
# Select the Molecular Weight and Melting Point for Ethanol and Glucose
df.iloc[[1, 2], [1, 2]]
```

	Molecular Weight	Melting Point (°C)
B	46.07	-114.1
C	180.16	146.0

Index-based

Operations on dfs


```
average_molecular_weight = df["Molecular Weight"].mean()
print(f"Average Molecular Weight: {average_molecular_weight}")
```

```
Average Molecular Weight: 63.745000000000005
```

```
min_melting_point = df["Melting Point (°C)"].min()
max_melting_point = df["Melting Point (°C)"].max()
print(f"Minimum Melting Point: {min_melting_point} °C")
print(f"Maximum Melting Point: {max_melting_point} °C")
```

```
Minimum Melting Point: -182.5 °C
Maximum Melting Point: 801.0 °C
```

Common string format specifiers (super useful!)

Specifier	Description	Example using f-strings	Output
d	Formats an integer as a decimal number.	f'{42:d}'	42
f	Formats a floating-point number as a fixed-point number.	f'{3.14159:.2f}'	3.14
e	Formats a floating-point number in scientific notation.	f'{0.0015:.2e}'	1.50e-03
%	Formats a number as a percentage.	f'{0.75:.1%}'	75.0%

```
average_molecular_weight = df["Molecular Weight"].mean()
print(f"Average Molecular Weight: {average_molecular_weight:.2f}")
```

```
Average Molecular Weight: 63.75
```

Sorting by value in column


```
sorted_by_weight_df = df.sort_values(by="Molecular Weight")
print("Sorted by Molecular Weight:")
sorted_by_weight_df
```

```
Sorted by Molecular Weight:
```

	Compound	Molecular Weight	Melting Point (°C)
E	Methane	16.040	-182.5
A	Water	18.015	0.0
B	Ethanol	46.070	-114.1
D	Sodium Chloride	58.440	801.0
C	Glucose	180.160	146.0

```
sorted_by_weight_df = df.sort_values(by="Molecular Weight", ascending=False)
print("Sorted by Molecular Weight:")
sorted_by_weight_df
```

```
Sorted by Molecular Weight:
```

	Compound	Molecular Weight	Melting Point (°C)
C	Glucose	180.160	146.0
D	Sodium Chloride	58.440	801.0
B	Ethanol	46.070	-114.1
A	Water	18.015	0.0
E	Methane	16.040	-182.5

Adding a new column


```
df["Moles (in 100g)"] = 100 / df["Molecular Weight"]
df
```

	Compound	Molecular Weight	Melting Point (°C)	Moles (in 100g)
0	Water	18.015	0.0	5.550930
1	Ethanol	46.070	-114.1	2.170610
2	Glucose	180.160	146.0	0.555062
3	Sodium Chloride	58.440	801.0	1.711157
4	Methane	16.040	-182.5	6.234414

More advanced filtering using `query()`


```
filtered_df = df.query('`Molecular Weight` > 30 and `Melting Point (°C)` < 100')
filtered_df
```

	Compound	Molecular Weight	Melting Point (°C)	Moles (in 100g)
1	Ethanol	46.07	-114.1	2.17061

```
filtered_df = df.query('`Molecular Weight` > 30 or `Melting Point (°C)` > 100')
filtered_df
```

	Compound	Molecular Weight	Melting Point (°C)	Moles (in 100g)
1	Ethanol	46.07	-114.1	2.170610
2	Glucose	180.16	146.0	0.555062
3	Sodium Chloride	58.44	801.0	1.711157

Combining DataFrames

- Often you will have data that comes from multiple sources (files), and you would like to combine them into a single DataFrame
- Pandas provides useful functionality for that:
  - **pd.merge(df1, df2)**: Merging two DataFrames representing different sets of properties for a selection of compounds.
  - **pd.concat(df1, df2)**: Concatenating DataFrames containing properties of different compounds to create a single, comprehensive DataFrame.
  - **df1.join([df2])**: Combining DataFrames with different information about compounds based on their index. This is useful when the indices represent a shared order or align datasets based on their order rather than a specific key.
- Let’s look at some examples.

pd.merge(df1, df2)

```python
# DataFrame of physical properties
df_physical = pd.DataFrame({
    "Compound": ["Water", "Ethanol", "Methane"],
    "Boiling Point (°C)": [100, 78.37, -161.5],
    "Density (g/mL)": [1.0, 0.789, 0.000656]
})

# DataFrame of chemical properties
df_chemical = pd.DataFrame({
    "Compound": ["Water", "Ethanol", "Methane"],
    "Flammability": ["No", "Yes", "Yes"]
})

# Merging on 'Compound'
df_merged = pd.merge(df_physical, df_chemical, on="Compound")
df_merged
```

	Compound	Boiling Point (°C)	Density (g/mL)	Flammability
0	Water	100.00	1.000000	No
1	Ethanol	78.37	0.789000	Yes
2	Methane	-161.50	0.000656	Yes

pd.concat(df1, df2)

```python
# DataFrame of compounds set 1
df_set1 = pd.DataFrame({
    "Compound": ["Water", "Ethanol"],
    "Molecular Weight": [18.015, 46.07],
    "Melting Point (°C)": [0, -114.1]
})

# DataFrame of compounds set 2
df_set2 = pd.DataFrame({
    "Compound": ["Glucose", "Sodium Chloride"],
    "Molecular Weight": [180.16, 58.44],
    "Melting Point (°C)": [146, 801]
})

# Concatenating vertically
df_concatenated = pd.concat([df_set1, df_set2], axis=0).reset_index(drop=True)
df_concatenated
```

	Compound	Molecular Weight	Melting Point (°C)
0	Water	18.015	0.0
1	Ethanol	46.070	-114.1
2	Glucose	180.160	146.0
3	Sodium Chloride	58.440	801.0

df1.join([df2])

```python
# DataFrame of compounds
df_compounds = pd.DataFrame({
    "Compound": ["Water", "Ethanol", "Methane"]
}).set_index("Compound")

# DataFrame of boiling points
df_bp = pd.DataFrame({
    "Boiling Point (°C)": [100, 78.37, -161.5]
}, index=["Water", "Ethanol", "Methane"])

# DataFrame of melting points
df_mp = pd.DataFrame({
    "Melting Point (°C)": [0, -114.1, -182.5]
}, index=["Water", "Ethanol", "Methane"])

# Combining using join
df_combined = df_compounds.join([df_bp, df_mp])
df_combined
```

Compound	Boiling Point (°C)	Melting Point (°C)
Water	100.00	0.0
Ethanol	78.37	-114.1
Methane	-161.50	-182.5

.apply() on single column

```python
import pandas as pd

data = {
    "Compound": ["Water", "Ethanol", "Glucose", "Sodium Chloride", "Methane"],
    "Molecular Weight (g/mol)": [18.015, 46.07, 180.16, 58.44, 16.04]
}
df = pd.DataFrame(data)

# Convert Molecular Weight from g/mol to kg/mol using a lambda function
df['Molecular Weight (kg/mol)'] = df['Molecular Weight (g/mol)'].apply(lambda x: x / 1000)
df
```

	Compound	Molecular Weight (g/mol)
0	Water	18.015
1	Ethanol	46.070
2	Glucose	180.160
3	Sodium Chloride	58.440
4	Methane	16.040

Molecular Weight (kg/mol)

```python
def convert_to_kg_per_mol(weight_g_per_mol):
    return weight_g_per_mol / 1000

# Apply the function to the 'Molecular Weight (g/mol)' column
df['Molecular Weight (kg/mol)'] = df['Molecular Weight (g/mol)'].apply(convert_to_kg_per_mol)
df
```

	Compound	Molecular Weight (g/mol)	Molecular Weight (kg/mol)
0	Water	18.015	0.018015
1	Ethanol	46.070	0.046070
2	Glucose	180.160	0.180160
3	Sodium Chloride	58.440	0.058440
4	Methane	16.040	0.016040

.apply() on single column


```python
def convert_to_kg_per_mol(weight_g_per_mol):
    return weight_g_per_mol / 1000

# Apply the function to the 'Molecular Weight (g/mol)' column
df['Molecular Weight (kg/mol)'] = df['Molecular Weight (g/mol)'].apply(convert_to_kg_per_mol)
df
```

	Compound	Molecular Weight (g/mol)	Molecular Weight (kg/mol)
0	Water	18.015	0.018015
1	Ethanol	46.070	0.046070
2	Glucose	180.160	0.180160
3	Sodium Chloride	58.440	0.058440
4	Methane	16.040	0.016040

.apply() on multiple columns

```python
# Adding mass of solute (in grams) and volume of solution (in liters) to the DataFrame
df['Mass (g)'] = [18, 46, 180, 58, 16]  # Example masses
df['Volume (L)'] = [1, 0.5, 1, 0.5, 1]  # Example volumes

# Calculate molarity using molecular weight, mass, and volume
df['Molarity (mol/L)'] = df.apply(lambda row: row['Mass (g)'] / row['Molecular Weight (g/mol)'] / row['Volume (L)'], axis=1)
df
```

	Compound	Molecular Weight (g/mol)	Molecular Weight (kg/mol)	Mass (g)	Volume (L)	Molarity (mol/L)
0	Water	18.015	0.018015	18	1.0	0.999167
1	Ethanol	46.070	0.046070	46	0.5	1.996961
2	Glucose	180.160	0.180160	180	1.0	0.999112
3	Sodium Chloride	58.440	0.058440	58	0.5	1.984942
4	Methane	16.040	0.016040	16	1.0	0.997506

```python
# Define a function to calculate molarity
def calculate_molarity(row):
    mass_g = row['Mass (g)']
    molecular_weight_g_per_mol = row['Molecular Weight (g/mol)']
    volume_L = row['Volume (L)']
    molarity = mass_g / molecular_weight_g_per_mol / volume_L
    return molarity

# Apply the function to the DataFrame to calculate molarity
df['Molarity (mol/L)'] = df.apply(calculate_molarity, axis=1)
df
```

	Compound	Molecular Weight (g/mol)	Molecular Weight (kg/mol)	Mass (g)	Volume (L)	Molarity (mol/L)
0	Water	18.015	0.018015	18	1.0	0.999167
1	Ethanol	46.070	0.046070	46	0.5	1.996961
2	Glucose	180.160	0.180160	180	1.0	0.999112
3	Sodium Chloride	58.440	0.058440	58	0.5	1.984942
4	Methane	16.040	0.016040	16	1.0	0.997506

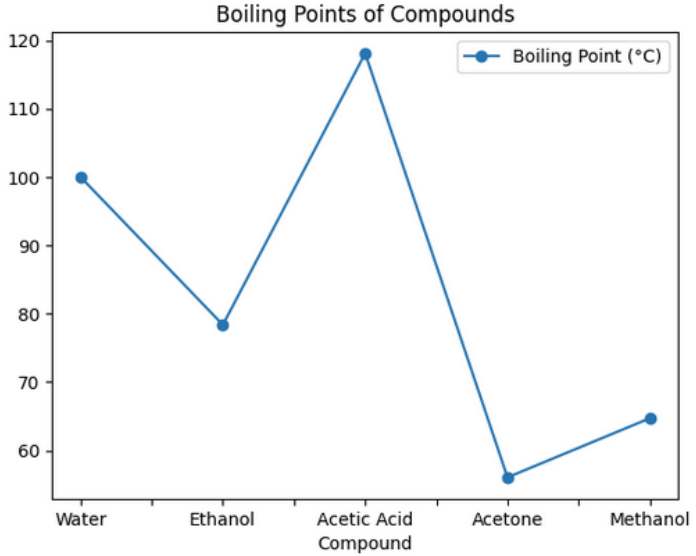
Note: we have to use the **axis** keyword.

In pandas DataFrames:
- **axis=0**: Perform the operation vertically (column by column).
- **axis=1**: Perform the operation horizontally (row by row).

Making basic plots in pandas

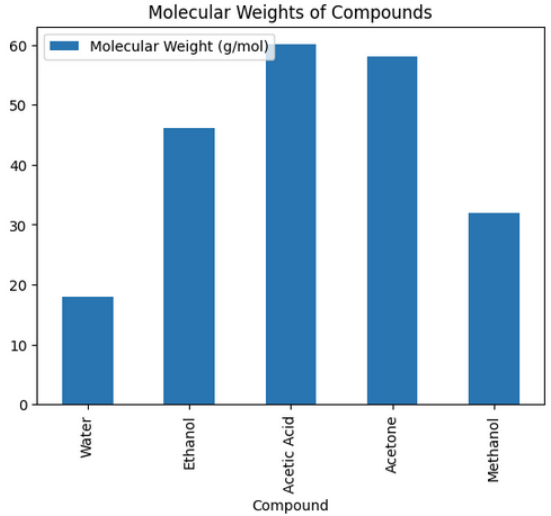
```
df.plot(x='Compound', y='Boiling Point (°C)', marker='o', linestyle='-', title='Boiling Points of Compounds')
```

```
<Axes: title={'center': 'Boiling Points of Compounds'}, xlabel='Compound'>
```




```
df.plot(kind='bar', x='Compound', y='Molecular Weight (g/mol)', title='Molecular Weights of Compounds')
```

```
<Axes: title={'center': 'Molecular Weights of Compounds'}, xlabel='Compound'>
```



While Pandas plotting is convenient for quick and straightforward plots, **Matplotlib** provides more control and flexibility for customizing plots.

Matplotlib – the default Python plotting library

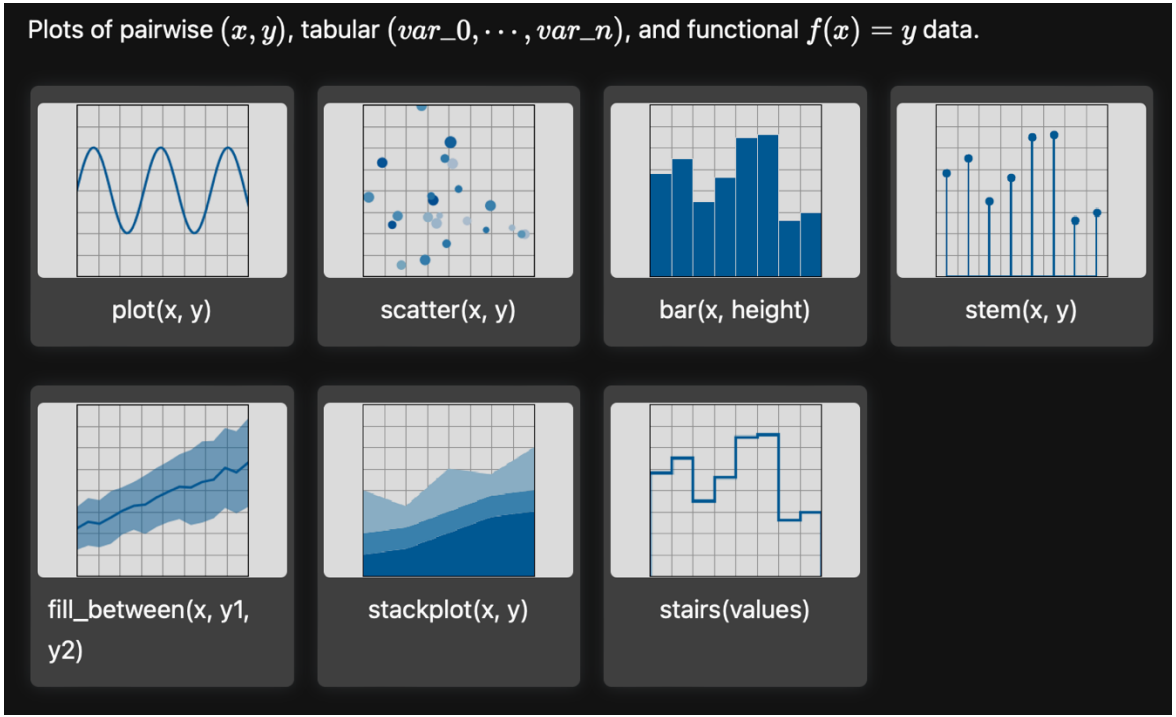
Matplotlib

How to get it? `pip install matplotlib`
How to import it? `import matplotlib.pyplot as plt`

https://matplotlib.org

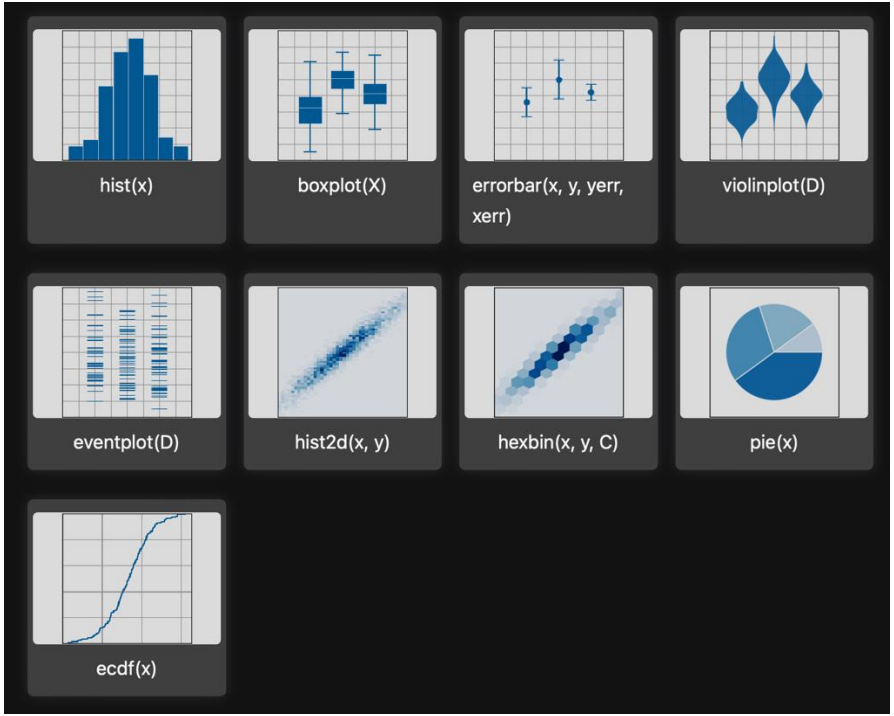
Pairwise data

https://matplotlib.org/stable/plot_types/index.html#pairwise-data



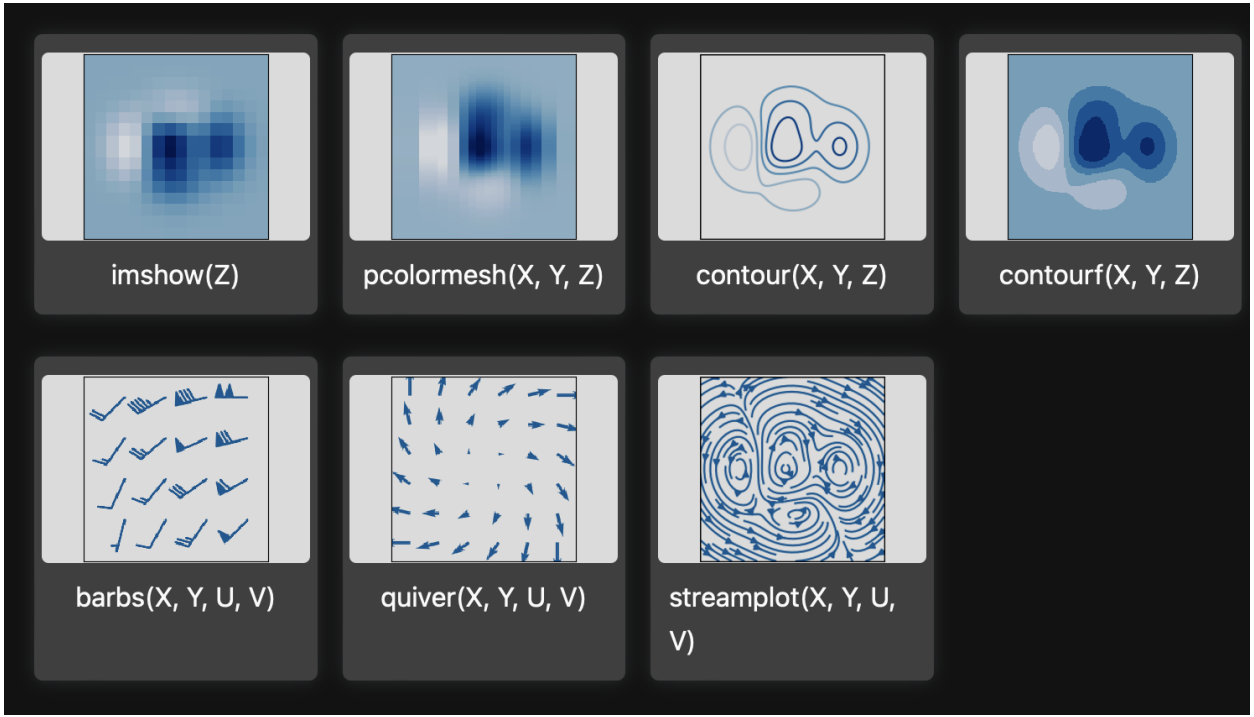
Statistical distributions

https://matplotlib.org/stable/plot_types/index.html#statistical-distributions

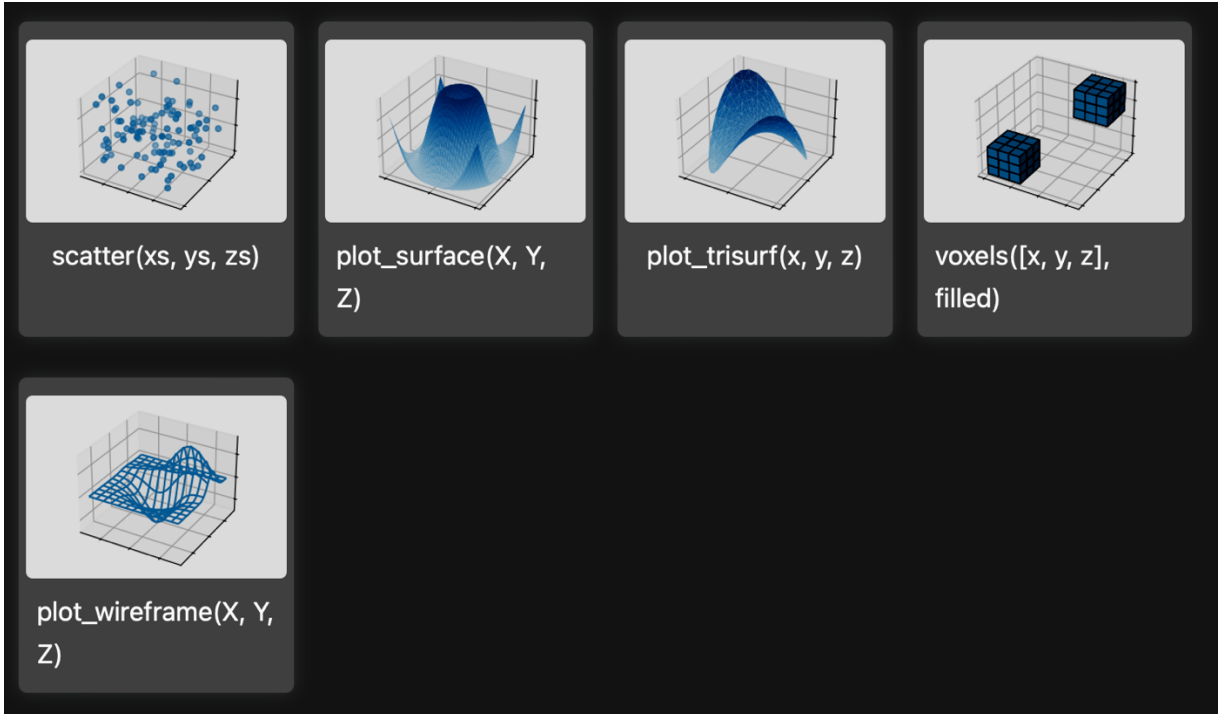


Gridded data

https://matplotlib.org/stable/plot_types/index.html#gridded-data



3D and volumetric data



Gallery with 100s of examples

- https://matplotlib.org/stable/gallery/index.html

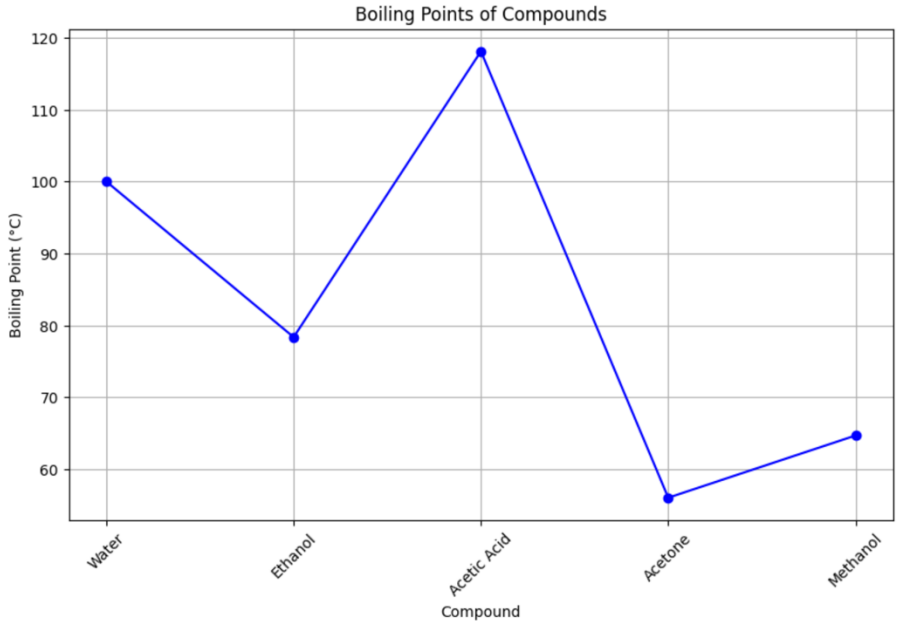
Line plot

```python
import pandas as pd
import matplotlib.pyplot as plt
```

```python
data = {
    'Compound': ['Water', 'Ethanol', 'Acetic Acid', 'Acetone', 'Methanol'],
    'Boiling Point (°C)': [100, 78.37, 118.1, 56.05, 64.7],
    'Molecular Weight (g/mol)': [18.015, 46.07, 60.052, 58.08, 32.04]
}
df = pd.DataFrame(data)
df
```

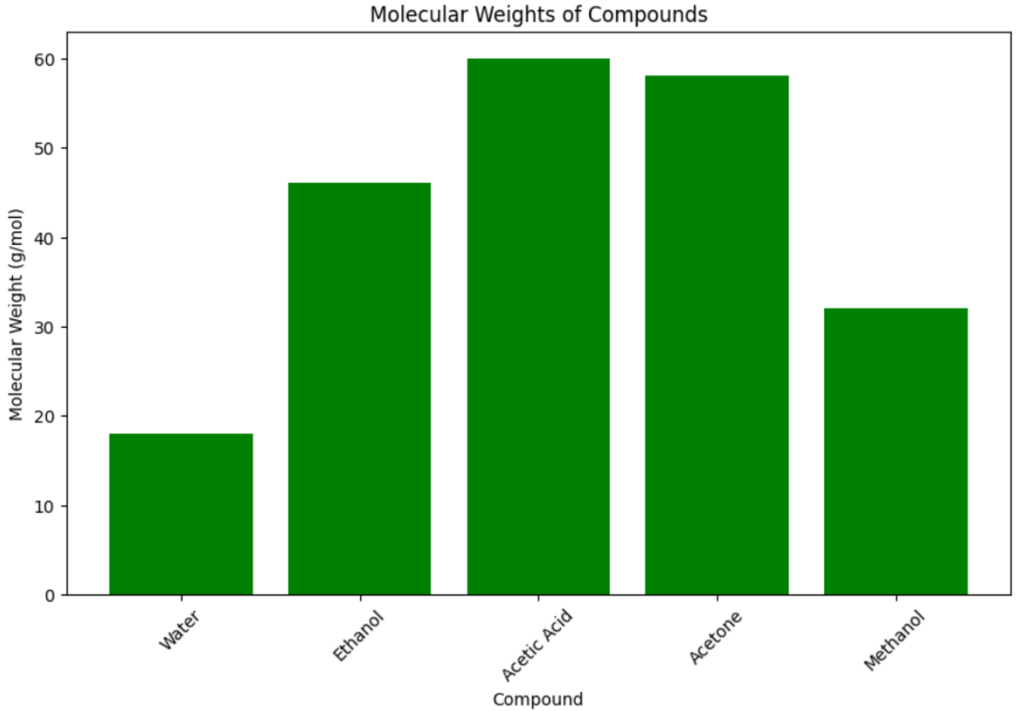
	Compound	Boiling Point (°C)	Molecular Weight (g/mol)
0	Water	100.00	18.015
1	Ethanol	78.37	46.070
2	Acetic Acid	118.10	60.052
3	Acetone	56.05	58.080
4	Methanol	64.70	32.040

```python
plt.figure(figsize=(10, 6))  # Set the figure size
plt.plot(df['Compound'], df['Boiling Point (°C)'], marker='o', linestyle='-', color='blue')
plt.title('Boiling Points of Compounds')
plt.xlabel('Compound')
plt.ylabel('Boiling Point (°C)')
plt.grid(True)
plt.xticks(rotation=45)  # Rotate the x-axis labels for better readability
```



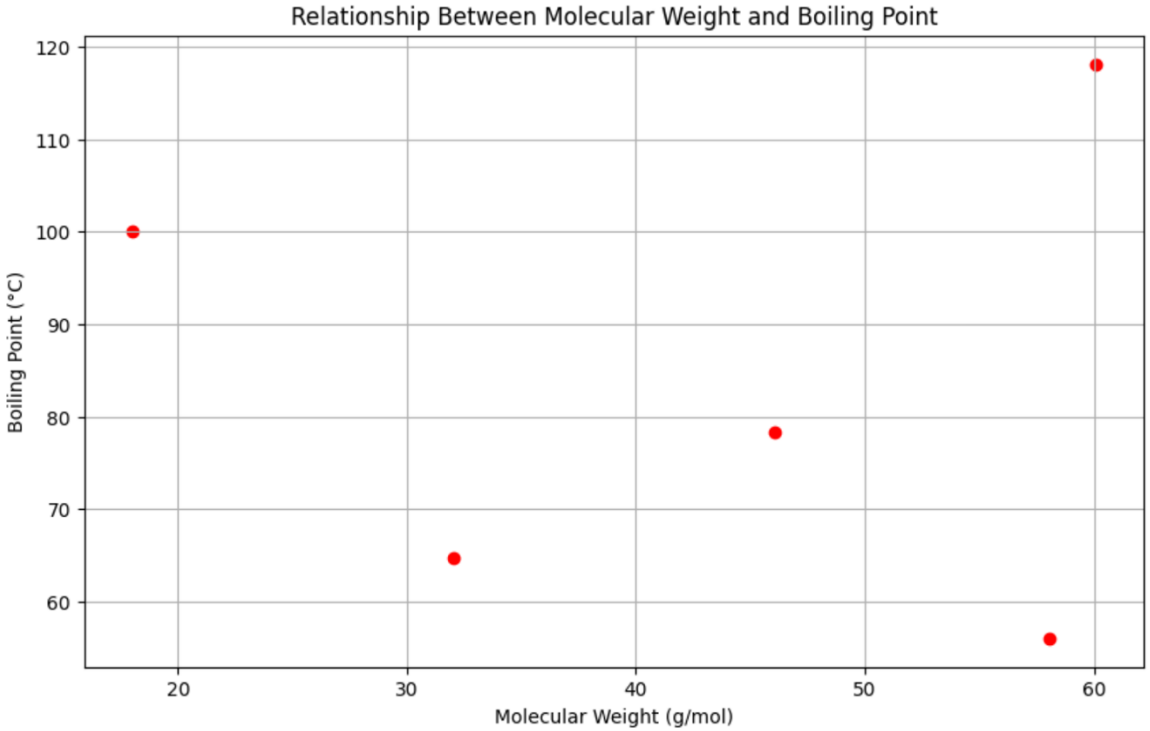
Bar Chart

```
plt.figure(figsize=(10, 6))
plt.bar(df['Compound'], df['Molecular Weight (g/mol)'], color='green')
plt.title('Molecular Weights of Compounds')
plt.xlabel('Compound')
plt.ylabel('Molecular Weight (g/mol)')
plt.xticks(rotation=45)
```



Scatter plot

```
plt.figure(figsize=(10, 6))
plt.scatter(df['Molecular Weight (g/mol)'], df['Boiling Point (°C)'], color='red')
plt.title('Relationship Between Molecular Weight and Boiling Point')
plt.xlabel('Molecular Weight (g/mol)')
plt.ylabel('Boiling Point (°C)')
plt.grid(True)
```



Saving figures

```
plt.savefig('plot.png')
plt.savefig('plot.svg')
plt.savefig('plot.pdf')
```

plot.pdf
plot.png
plot.svg

matplotlib.pyplot.savefig

matplotlib.pyplot.savefig(*args, **kwargs) [source]

Save the current figure.

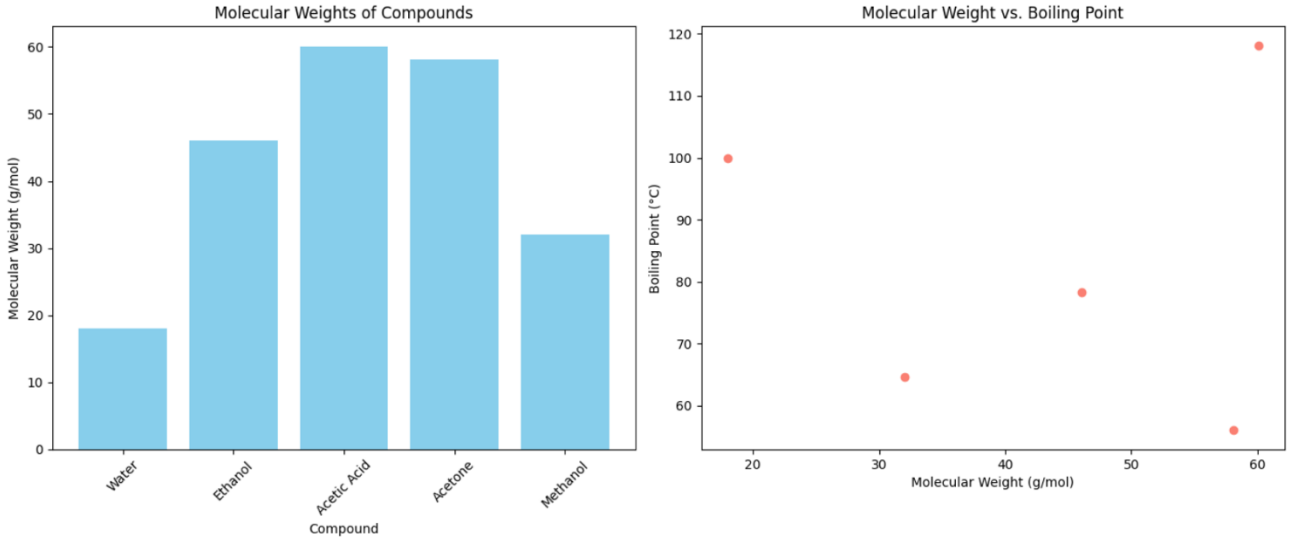
Call signature:

```
savefig(fname, *, transparent=None, dpi='figure', format=None,
        metadata=None, bbox_inches=None, pad_inches=0.1,
        facecolor='auto', edgecolor='auto', backend=None,
        **kwargs
       )
```

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html#matplotlib-pyplot-savefig

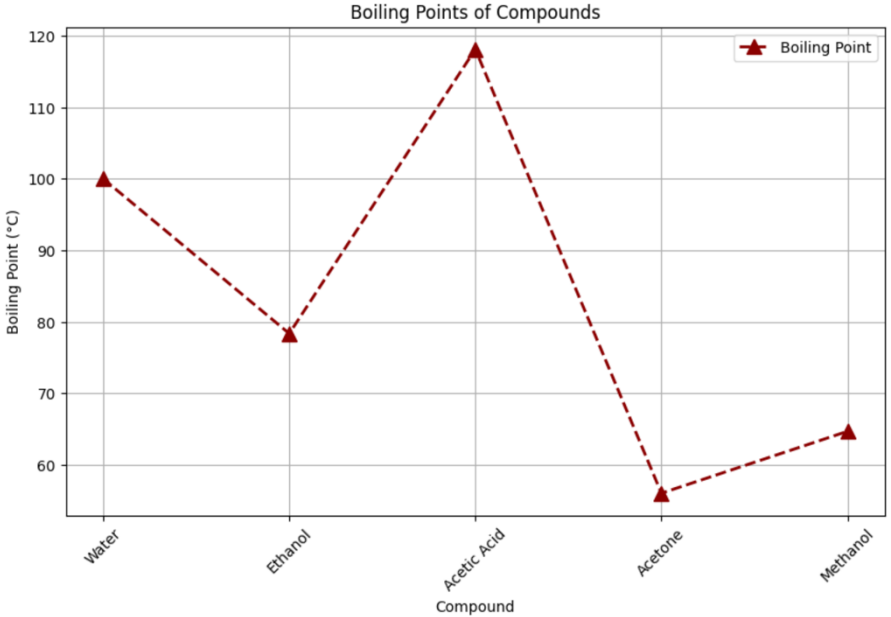
subplots

```python
fig, ax = plt.subplots(1, 2, figsize=(14, 6))  # 1 row, 2 columns

# First subplot
ax[0].bar(df['Compound'], df['Molecular Weight (g/mol)'], color='skyblue')
ax[0].set_title('Molecular Weights of Compounds')
ax[0].set_xlabel('Compound')
ax[0].set_ylabel('Molecular Weight (g/mol)')
ax[0].tick_params(axis='x', rotation=45)

# Second subplot
ax[1].scatter(df['Molecular Weight (g/mol)'], df['Boiling Point (°C)'], color='salmon')
ax[1].set_title('Molecular Weight vs. Boiling Point')
ax[1].set_xlabel('Molecular Weight (g/mol)')
ax[1].set_ylabel('Boiling Point (°C)')

plt.tight_layout()  # Adjust layout to not overlap
```



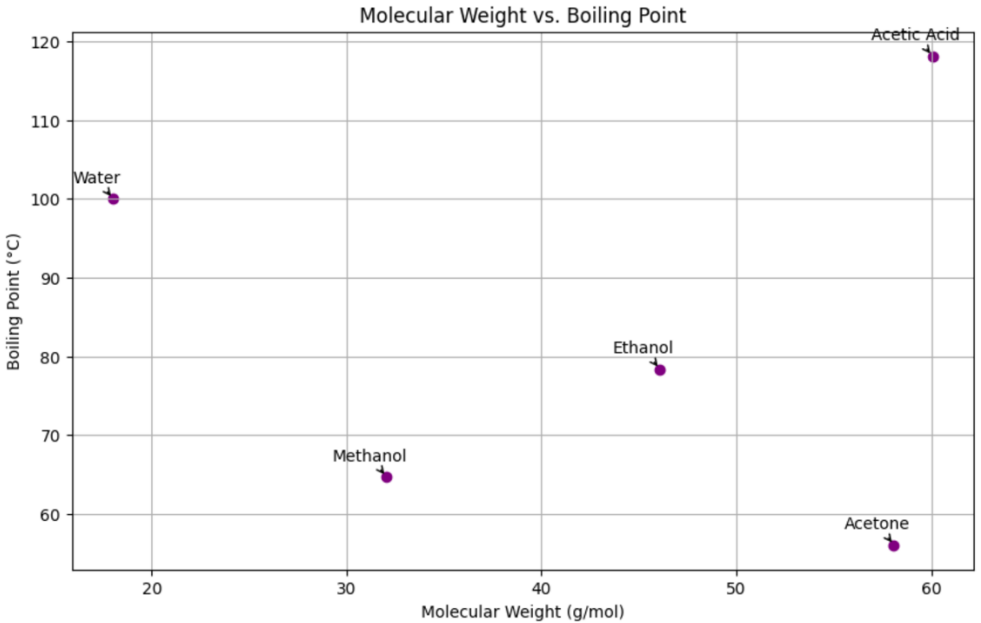
Customizing line styles and markers

```
plt.figure(figsize=(10, 6))
plt.plot(df['Compound'], df['Boiling Point (°C)'],
         color='darkred', linestyle='--', marker='^', markersize=10, linewidth=2,
         label='Boiling Point')
plt.title('Boiling Points of Compounds')
plt.xlabel('Compound')
plt.ylabel('Boiling Point (°C)')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
```



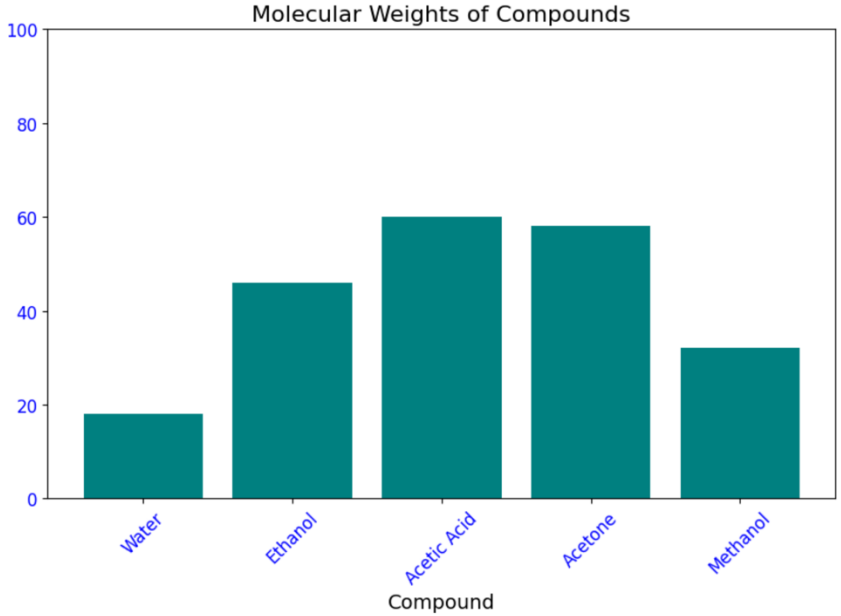
Annotating points

```python
plt.figure(figsize=(10, 6))
plt.scatter(df['Molecular Weight (g/mol)'], df['Boiling Point (°C)'], color='purple')

# Highlight the point for Acetic Acid
for i, row in df.iterrows():
    bp = row['Boiling Point (°C)']
    mw = row['Molecular Weight (g/mol)']
    plt.annotate(row['Compound'], (mw, bp),
                 textcoords="offset points", xytext=(-10,10), ha='center',
                 arrowprops=dict(arrowstyle='->', color='black'))

plt.title('Molecular Weight vs. Boiling Point')
plt.xlabel('Molecular Weight (g/mol)')
plt.ylabel('Boiling Point (°C)')
plt.grid(True)
```



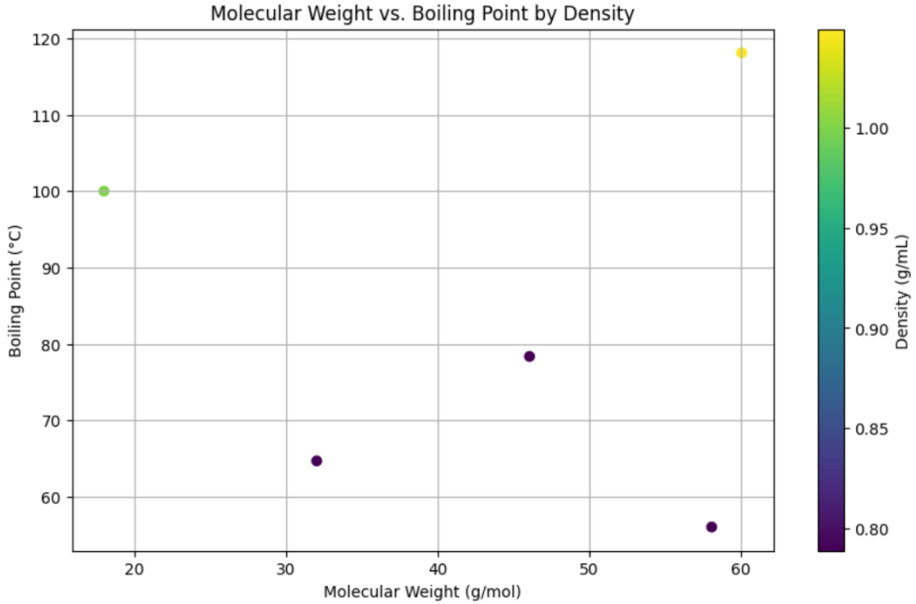
Customizing axes

```
plt.figure(figsize=(10, 6))
plt.bar(df['Compound'], df['Molecular Weight (g/mol)'], color='teal')

# Setting the range for the y-axis
plt.ylim(0, 100)

# Customizing tick labels
plt.xticks(rotation=45, fontsize=12, color='blue')
plt.yticks(fontsize=12, color='blue')

plt.title('Molecular Weights of Compounds', fontsize=16)
plt.xlabel('Compound', fontsize=14)
```



Colorbars

```python
# Assuming an additional 'Density (g/mL)' column in the DataFrame
df['Density (g/mL)'] = [1.0, 0.789, 1.049, 0.790, 0.791]  # Example densities

plt.figure(figsize=(10, 6))
sc = plt.scatter(df['Molecular Weight (g/mol)'], df['Boiling Point (°C)'],
                 c=df['Density (g/mL)'], cmap='viridis')
plt.colorbar(sc, label='Density (g/mL)')

plt.title('Molecular Weight vs. Boiling Point by Density')
plt.xlabel('Molecular Weight (g/mol)')
plt.ylabel('Boiling Point (°C)')
plt.grid(True)
```

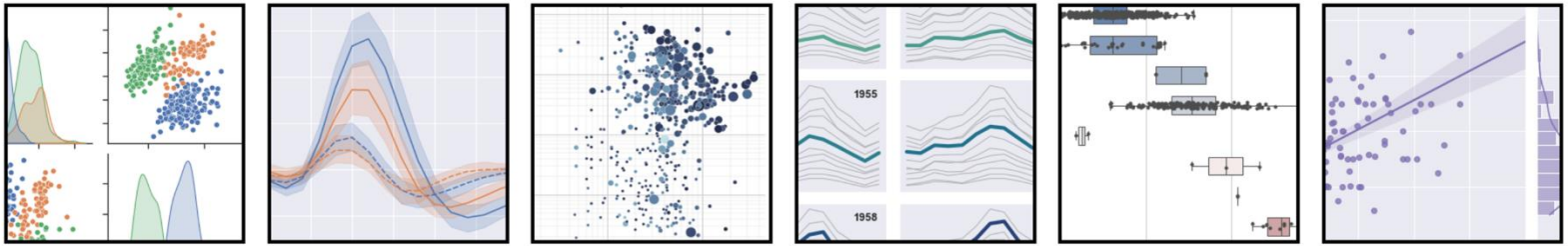


Seaborn – an extension of matplotlib to make more aesthetic plots

(I made most of my plots using seaborn)

How to make nicer plots → seaborn

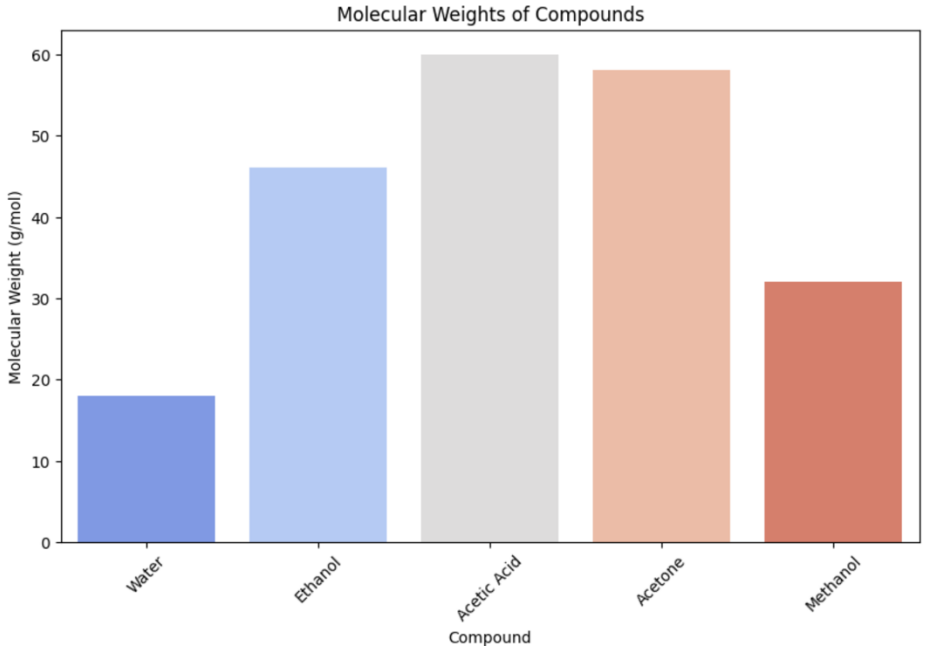
seaborn: statistical data visualization



- → pip install seaborn
- → import seaborn as sns
- → Documentation: https://seaborn.pydata.org

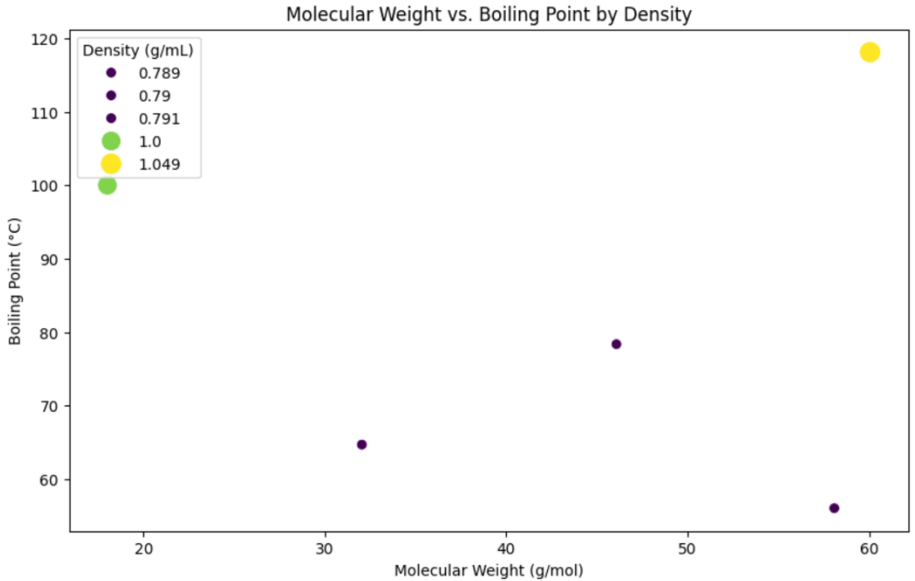
Bar plot

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Compound', y='Molecular Weight (g/mol)', data=df, palette='coolwarm', hue='Compound')
plt.title('Molecular Weights of Compounds')
plt.xticks(rotation=45)
```



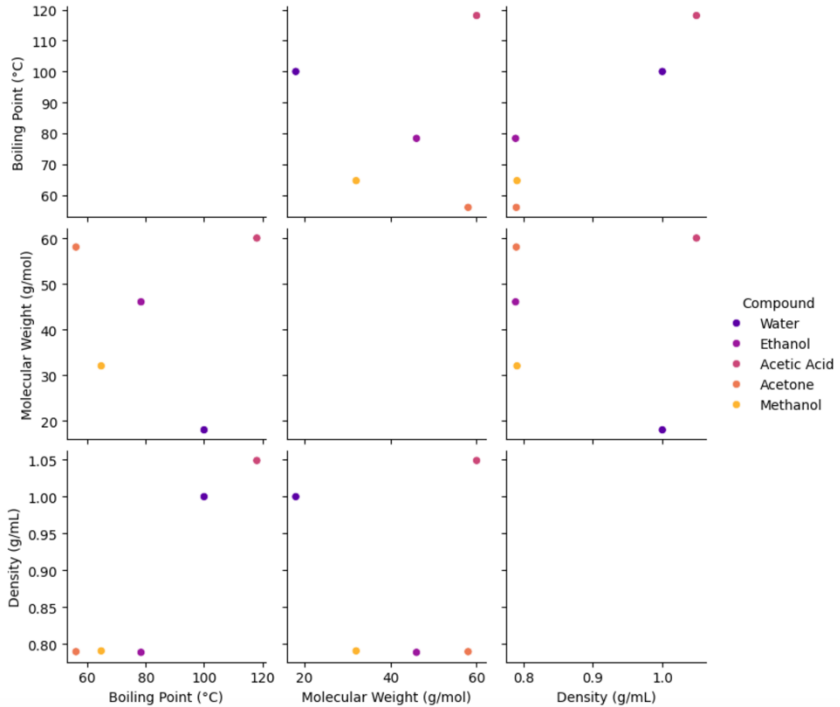
Scatter plot


```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Molecular Weight (g/mol)', y='Boiling Point (°C)', data=df,
                hue='Density (g/mL)', palette='viridis', size='Density (g/mL)',
                sizes=(50, 200))
plt.title('Molecular Weight vs. Boiling Point by Density')
```



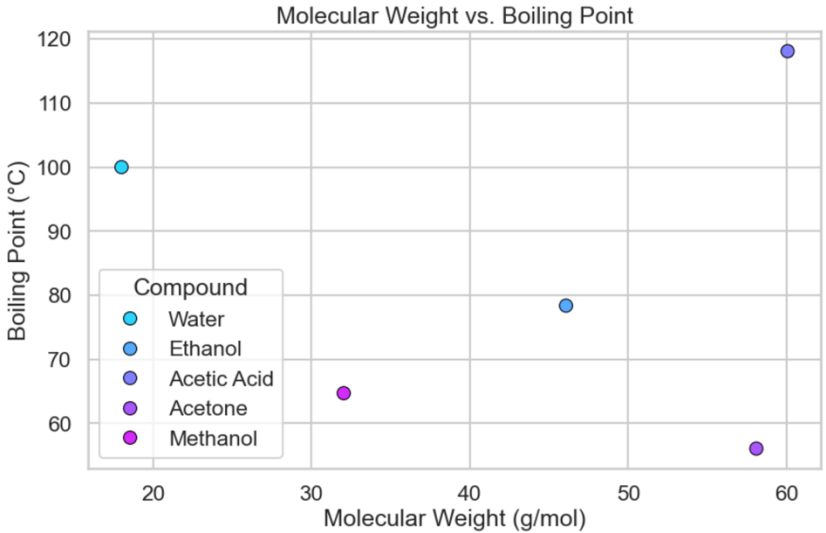
Pair plot


```
sns.pairplot(df, hue='Compound', palette='plasma')
```

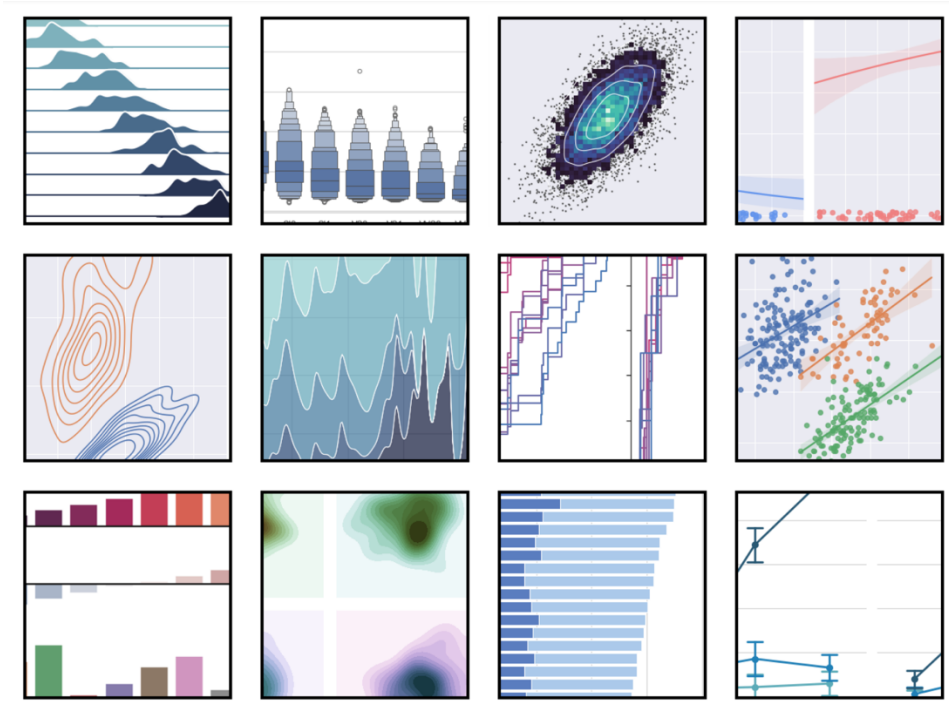


More customization

```
sns.set_theme(style='whitegrid')
sns.set_context('talk')
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Molecular Weight (g/mol)', y='Boiling Point (°C)',
                data=df, hue='Compound', palette='cool', s=100, edgecolor='black')
plt.title('Molecular Weight vs. Boiling Point')
```



There are many more seaborn examples



https://seaborn.pydata.org/examples/index.html

https://seaborn.pydata.org/tutorial/aesthetics.html