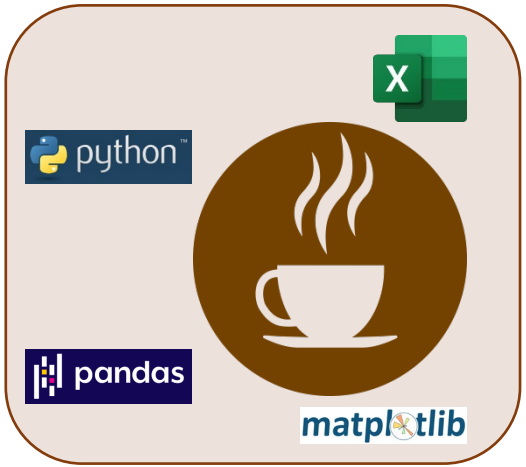
UNIVERSITÄT BERN

Data Visualization: Excel versus Python

When to use which approach ?



University Library Bern, Science Library

Dr. Michael Horn, Coffee & Bit(e)s, Spring 2021

This Lecture

Content

- Excel and Python tools in a nutshell
- data visualization: data size and data complexity
- data visualization: chart/ plot types
- conclusion

Excel and Python tools in a nutshell

Excel / Python tools (jupyter, pandas, matplotlib)

- Excel:
  - Microsoft product
  - desktop application
  - intuitive handling
  - multifunctional:
    - spreadsheets for data handling
    - allows complex data analysis
    - allows data visualization
    - …
  - has some relevant limitations in comparison to python tools

- Python tools:
  - open source: non-proprietary, transparent
  - Python:
    - programming (scripting) language
    - easy to learn, very powerful and flexible
  - jupyter notebook:
    - browser-based application
    - integrates code, code output and documentation
  - pandas and matplotlib
    - python libraries (pandas: data processing, matplotlib: data visualization)
    - fast, powerful, flexible, easy to use, very robust

Data size and data complexity

Small and simple data sets

Typical scenario:

- manual data acquisition
- rapid data visualization

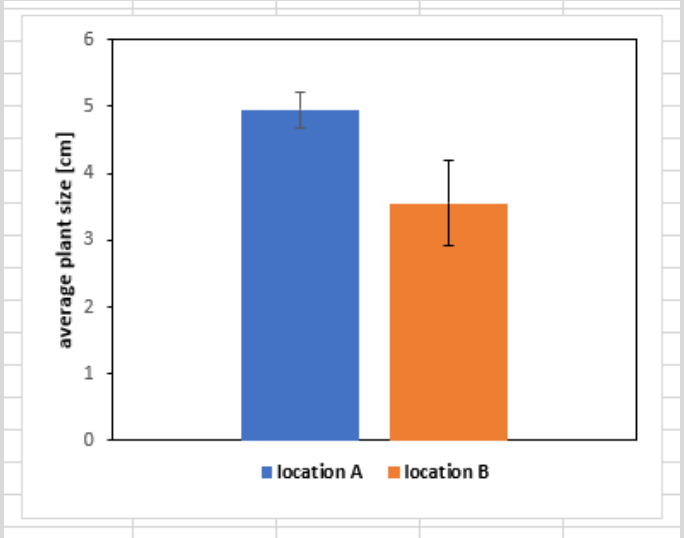
Some pros and cons:

(+) very intuitive

(+) quickly ready for use

(-) requires intermediate steps (average, stddev)

	A	B	C
1		plant size [cm]	
2		location A	location B
3		5.1	3.5
4		4.9	3
5		4.7	3.2
6		4.6	3.1
7		5	3.6
8		5.4	4.9
9			
10			
11	average	4.95	3.55
12	stddev	0.2629956	0.6396614
13			



Data size and data complexity

Small and simple data sets

Some pros and cons:

(+) requires no intermediate steps

(+) faster post processing (axes labels, etc.)

(-) more intricate

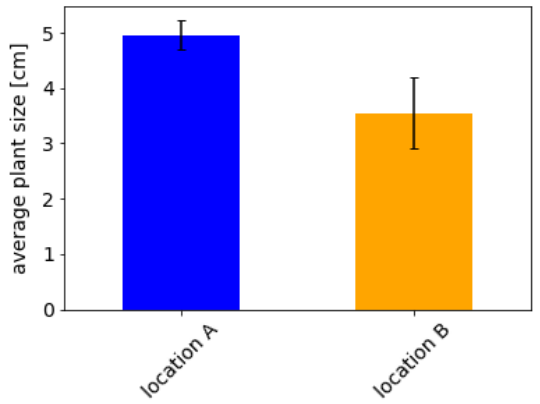
(-) slower general handling

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

```
data = pd.read_excel("simple_small_ds.xlsx",
                     engine="openpyxl")
data
```

	location A	location B
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
5	5.4	4.9

```
plot = data.mean().plot(kind="bar", color=["blue", "orange"],
                        yerr=[data.std(ddof=0)[0], data.std(ddof=0)[1]],
                        capsize=3)
plot.set_ylabel("average plant size [cm]", fontsize=14, labelpad=8)
plot.tick_params(axis="both", labelsize=14)
plot.tick_params(axis="x", labelsize=14, rotation=45)
plt.show()
```



Data size and data complexity

Larger and more complex data sets

Typical scenario:

- manual or automated data acquisition
- more elaborate data visualization

Example data set:

- Iris data set in csv-format
- 4 columns with numeric data
- 1 column with categorical data
- 150 data points (rows)

iris_dataset_shuffled.csv

```
6.3,3.3,4.7,1.6,Iris-versicolor
7.4,2.8,6.1,1.9,Iris-virginica
6.1,2.6,5.6,1.4,Iris-virginica
4.8,3.4,1.9,0.2,Iris-setosa
5.9,3.2,4.8,1.8,Iris-versicolor
5.4,3.7,1.5,0.2,Iris-setosa
6.0,2.9,4.5,1.5,Iris-versicolor
7.9,3.8,6.4,2.0,Iris-virginica
4.7,3.2,1.3,0.2,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
6.4,3.1,5.5,1.8,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.0,2.7,5.1,1.6,Iris-versicolor
6.8,3.2,5.9,2.3,Iris-virginica
5.5,2.3,4.0,1.3,Iris-versicolor
```

...

	A	B	C	D	E
1	sepal length [cm]	sepal width [cm]	petal length [cm]	petal width [cm]	species
2	6.3	3.3	4.7	1.6	Iris-versicolor
3	7.4	2.8	6.1	1.9	Iris-virginica
4	6.1	2.6	5.6	1.4	Iris-virginica
5	4.8	3.4	1.9	0.2	Iris-setosa
6	5.9	3.2	4.8	1.8	Iris-versicolor

...

python

	sepal_length (cm)	sepal_width (cm)	petal_length (cm)	petal_width (cm)	species
0	6.3	3.3	4.7	1.6	Iris-versicolor
1	7.4	2.8	6.1	1.9	Iris-virginica
2	6.1	2.6	5.6	1.4	Iris-virginica
3	4.8	3.4	1.9	0.2	Iris-setosa
4	5.9	3.2	4.8	1.8	Iris-versicolor

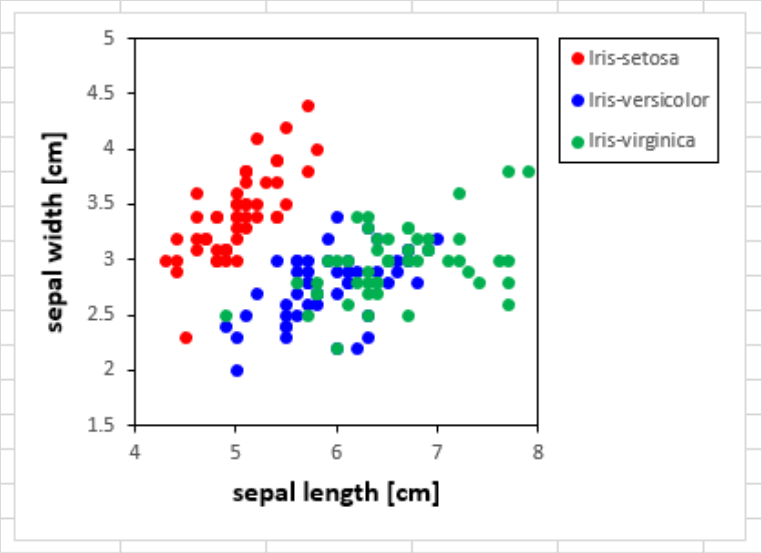
...

Data size and data complexity

Larger and more complex data sets

Intuitive workflow:

- open csv-file in Excel
- generate three data series corresponding to the categories in the “species”-column (manual filtering, copy-pasting, selecting of data)
- visualize data



Some pros and cons:

(+) workflow more intricate, but still intuitive

(-) requires manual re-organization of the data set

(-) requires repetitive execution of work steps

Data size and data complexity

Larger and more complex data sets

Some pros and cons:

(+) no repetitive execution of manual work steps (automated procedure)

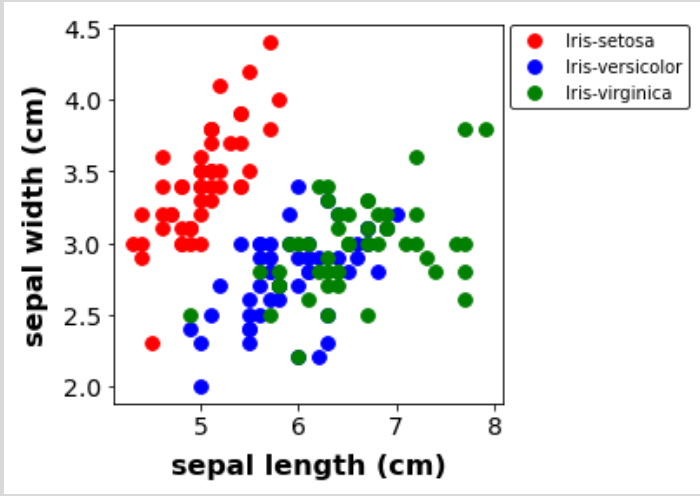
(+) reusable code: code can be gradually adapted to new requirements

(-) initial effort is higher

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

```
df = pd.read_csv("iris_dataset_shuffled.csv",
                 names=["sepal_length (cm)",
                        "sepal_width (cm)",
                        "petal_length (cm)",
                        "petal_width (cm)",
                        "species"])
```

```
fig, ax = plt.subplots(figsize=(4,4), nrows=1,
                       ncols=1)
colors = {"Iris-setosa":"red",
          "Iris-versicolor":"blue",
          "Iris-virginica":"green"}
grouped_dataframe = df.groupby("species")
for key, data in grouped_dataframe:
    sl_sw = data.plot(ax=ax, x=0, y=1, kind="scatter",
                      label=key, color=colors[key], s=60)
    sl_sw.set_xlabel("sepal length (cm)", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_sw.set_ylabel("sepal width (cm)", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_sw.tick_params(axis="both", labelsize=14)
    sl_sw.legend(bbox_to_anchor=
    (1.505, 1.025)).get_frame().set_edgecolor("black")
plt.show()
```



Data size and data complexity

Very large data sets

Typical scenario:

- automated data acquisition
- very large file containing the data set

Example data set:

- random value data set in csv-format
- 5 columns with numeric data
- 2`000`000 data points (rows)

very_large_dataset.csv

```
1.764052345967664,0.4001572083672233,0.9787379841057392,2.240893199201458,1.8675579901499675
-0.977277879876411,0.9500884175255894,-0.1513572082976979,-0.10321885179355784,0.41059850193837233
0.144043571160878,1.454273506962975,0.7610377251469934,0.12167501649282841,0.44386323274542566
0.33367432737426683,1.4940790731576061,-0.20515826376580087,0.31306770165090136,-0.8540957393017248
-2.5529898158340787,0.6536185954403606,0.8644361988595057,-0.7421650204064419,2.2697546239876076
-1.4543656745987648,0.04575851730144607,-0.1871838500258336,1.5327792143584575,1.469358769900285
0.1549474256969163,0.37816251960217356,-0.8877857476301128,-1.980796468223927,-0.3479121493261526
0.1563489691039800 5,1.2302906807277207,1.2023798487844113,-0.3873268174079523,-0.30230275057533557
-1.0485529650670926,-1.4200179371789752,-1.7062701906250126,1.9507753952317897,-0.5096521817516535
-0.4380743016111864,-1.2527953600499262,0.7774903558319101,-1.6138978475579515,-0.2127402802139687
-0.8954665611936756,0.386902497859262,-0.510805137568873,-1.180632184122412,-0.028182228338654868
0.42833187053041766,0.06651722238316789,0.3024718977397814,-0.6343220936809636,-0.3627411659871381
-0.672460447775951,-0.3595531615405413,-0.813146282044454,-1.7262826023316769,0.1774261422537528 3
-0.4017809362082619,-1.6301983469660446,0.4627822555257742,-0.9072983643832422,0.05194539579613895
0.7290905621775369,0.12898291075741067,1.1394006845433007,-1.2348258203536526,0.402341641177549
```

...

Data size and data complexity

Very large data sets

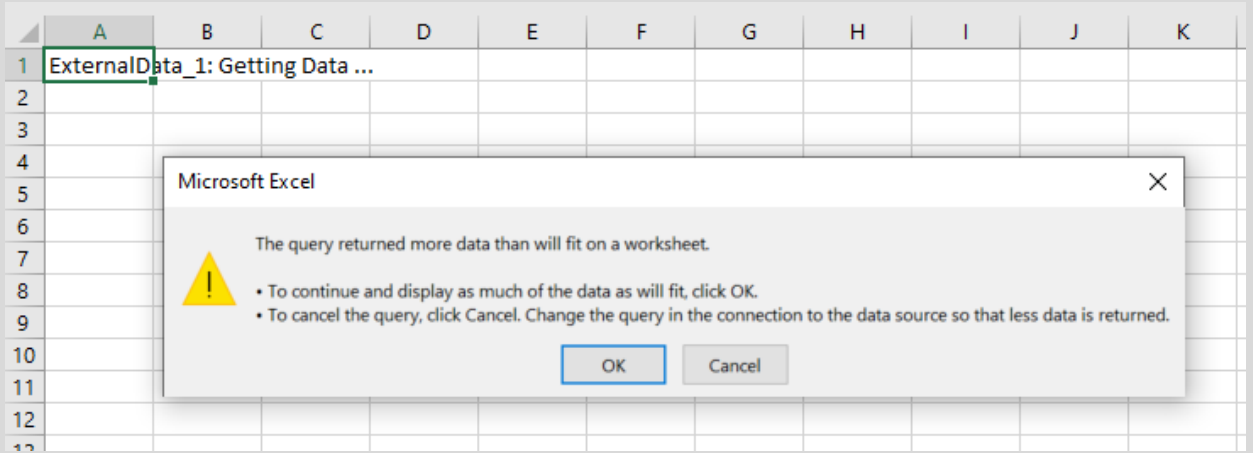
Intuitive workflow:

- open csv-file as Excel worksheet
- visualize data as shown before

Some pros and cons:

(-) data file cannot be properly opened

(-) data visualization cannot be performed as expected



Data size and data complexity

Very large data sets

Some pros and cons:

(+) data file can be opened without problems

(+) data visualization can be performed as expected

(-) procedure can require some computational time (depending on size of data set)

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

```
df_vl = pd.read_csv('very_large_dataset.csv',
                    header=None)
df_vl
```

	0	1	2	3	4
0	1.764052	0.400157	0.978738	2.240893	1.867558
1	-0.977278	0.950088	-0.151357	-0.103219	0.410599
2	0.144044	1.454274	0.761038	0.121675	0.443863
3	0.333674	1.494079	-0.205158	0.313068	-0.854096
4	-2.552990	0.653619	0.864436	-0.742165	2.269755
...	...	...	...	...	...
1999995	-1.802549	1.129603	-0.480182	0.374793	-0.188933
1999996	-0.145537	-0.260605	0.886023	1.078775	-0.236004
1999997	-1.513279	1.294779	0.267679	1.263666	-0.360712
1999998	-1.401144	1.129229	1.161400	0.484200	1.359349
1999999	0.590211	0.467616	-1.944715	-0.343985	-0.740190

2000000 rows × 5 columns

```
fig, ax = plt.subplots(figsize=(4,4), nrows=1, ncols=1)
scatter_plot = df_vl.plot(ax=ax, x=0, y=1, kind="scatter")
scatter_plot.set_xlabel("first column", fontsize=16,
                        fontweight="bold", labelpad=8)
scatter_plot.set_ylabel("second column", fontsize=16,
                        fontweight="bold", labelpad=8)
plt.show()
```

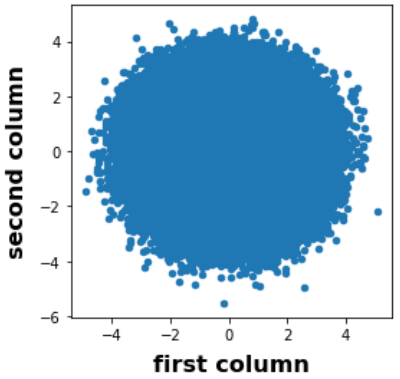


Chart / Plot types

Choices

https://support.microsoft.com/en-us/office/available-chart-types-in-office-a6187218-807e-4103-9e0a-27cdb19afb90#OfficeVersion=Windows

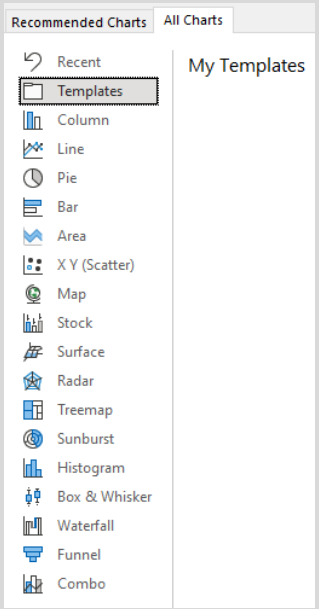
Some pros and cons:

(+) some relevant chart types are available

(+) chart recommendation is available

(-) choices are very restricted

(-) chart recommendation can be little constructive



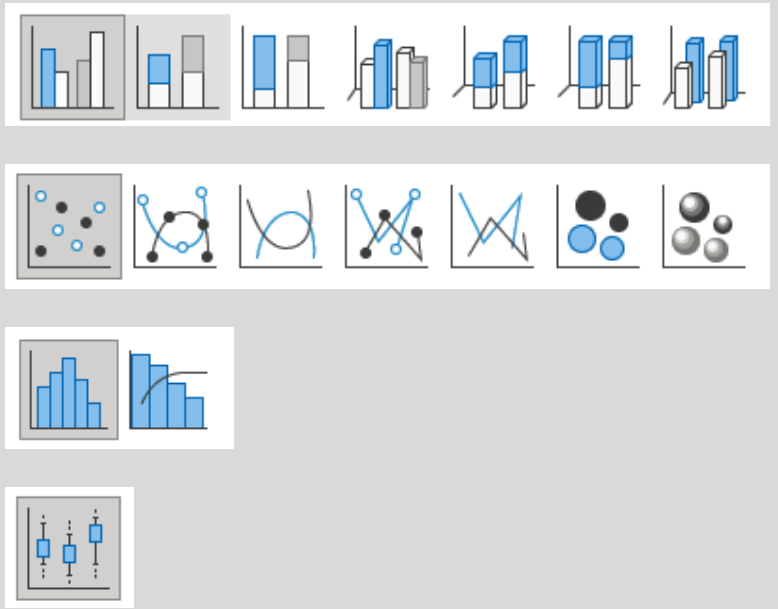


Chart / Plot types

Choices

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

Some pros and cons:

(+) large amount of chart types available

(+) good documentation

(+) good tutorials/ examples

(-) initial effort is higher (e.g. searching for code)

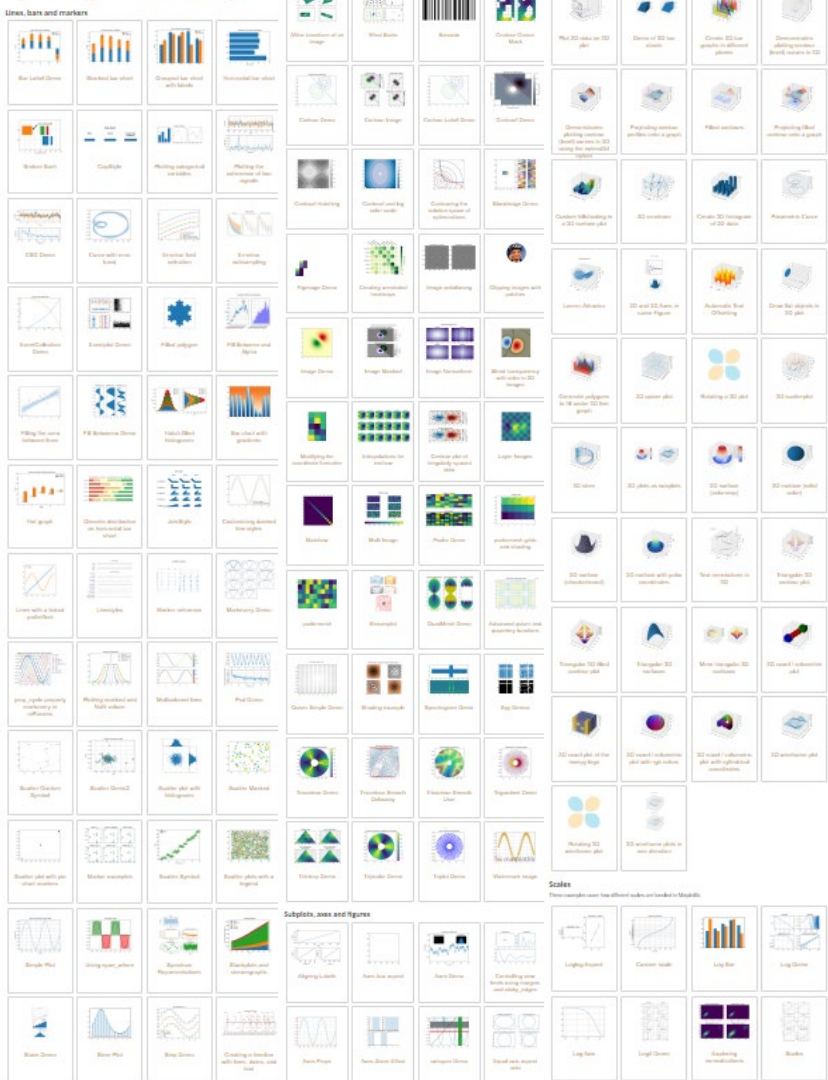
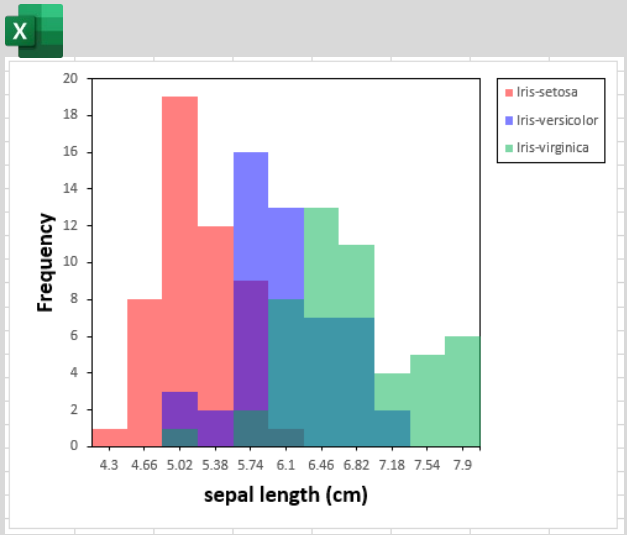


Chart / Plot types

Usability

Use case:

- histogram of the Iris sepal length data
- separated by species
- integrated in one single chart



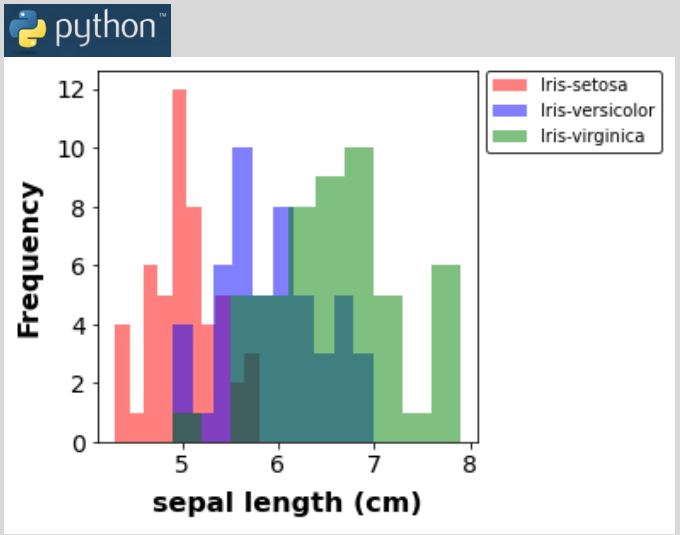


Chart / Plot types

Usability

Workflow in Excel:

- open csv-file in Excel
- generate three data subsets corresponding to the categories in the “species”-column (manual filtering, copy-pasting, selecting of data)
- generate a set of appropriate bins (manual)
- load and activate the Analysis ToolPak in Excel
- from the ToolPak use the analysis tool *Histogram* to calculate the bin frequencies for each data subset
- visualize the frequencies as individual data sets in a bar plot

Some problems with this workflow:

- very laborious and inefficient
- does not actually generate a histogram, but a bar plot

See: https://superuser.com/questions/1310668/how-do-i-overlay-two-histograms-in-excel

Chart / Plot types

Usability

Workflow in Python: minor changes to the existing code for the scatter plot → code for the histogram.

Code for scatter plot

```
fig, ax = plt.subplots(figsize=(4,4), nrows=1,
                       ncols=1)
colors = {"Iris-setosa":"red",
          "Iris-versicolor":"blue",
          "Iris-virginica":"green"}
grouped_dataframe = df.groupby("species")
for key, data in grouped_dataframe:
    sl_sw = data.plot(ax=ax, x=0, y=1, kind="scatter",
                      label=key, color=colors[key], s=60)
    sl_sw.set_xlabel("sepal length (cm)", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_sw.set_ylabel("sepal width (cm)", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_sw.tick_params(axis="both", labelsize=14)
    sl_sw.legend(bbox_to_anchor=
    (1.505, 1.025)).get_frame().set_edgecolor("black")
plt.show()
```

Code for histogram

```
fig, ax = plt.subplots(figsize=(4,4), nrows=1,
                       ncols=1)
colors = {"Iris-setosa":"red",
          "Iris-versicolor":"blue",
          "Iris-virginica":"green"}
grouped_dataframe = df.groupby("species")
for key, data in grouped_dataframe:
    sl_10 = data["sepal_length (cm)"].plot(ax=ax, kind="hist",
                      label=key, color=colors[key],
                      alpha=0.5, bins=10)
    sl_10.set_xlabel("sepal length (cm)", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_10.set_ylabel("Frequency", fontsize=16,
                     fontweight="bold", labelpad=8)
    sl_10.tick_params(axis="both", labelsize=14)
    sl_10.legend(bbox_to_anchor=
    (1.505, 1.025)).get_frame().set_edgecolor("black")
plt.show()
```

Conclusion: DV Excel versus Python tools

When to use which approach ?

- Excel:
  - small and simple data sets
  - basic chart types
  - little complex use cases
  - for preliminary estimations
  - for informal presentations/ meetings
  - if use of code should be avoided

- Python tools:
  - larger and more complex data sets
  - more intricate chart types
  - complex use cases with more elaborate data visualization
  - for data projects reaching a more mature level
  - for more formal presentations/ meetings
  - for publications

Thank you

for your attention

UNIVERSITÄT BERN

University Library Bern, Science Library

Dr. Michael Horn, Coffee & Bit(e)s, Spring 2021

@ www.unibe.ch/ub/sciencelibrary

➔ see «Coffee & Bit(e)s» for lecture notes

@ https://github.com/ubnpl/Coffee_and_Bites/tree/main/2021_DV_Excel_vs_Python

➔ see «Notebook_DV_Excel_vs_Python_2021.ipynb» for the Python code used