Artificial Intelligence | Basics of Algorithms & Application

Exercise 2: "Introduction to Python"

Timo Sturm & Dr. Dominik Jung
ki@is.tu-darmstadt.de
Prof. Dr. Peter Buxmann | Information Systems | Software & Digital Business
School of Business, Economics & Law
TU Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

WIRTSCHAFTS INFORMATIK

python™

Outline

1 Introduction to Python

- 1.1 Overview
- 1.2 Data Import & Management
- 1.3 Data Visualization & Exploration
- 1.4 Development Environments
- 1.5 Resources & Hands-On Exercises

Outline

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Introduction to Python

1.1 Overview

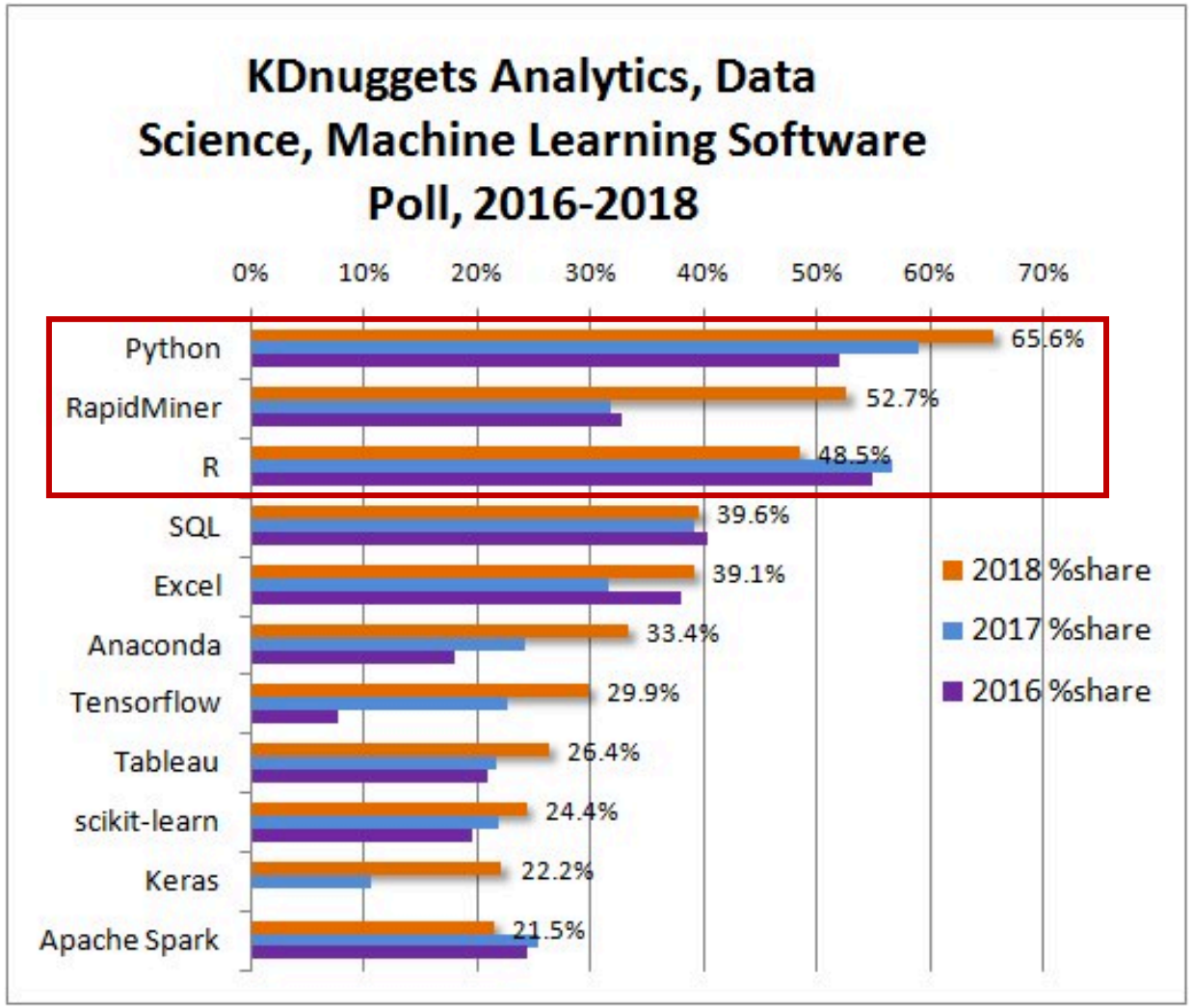
1.2 Data Import & Management

1.3 Data Visualization & Exploration

1.4 Development Environments

1.5 Resources & Hands-On Exercises

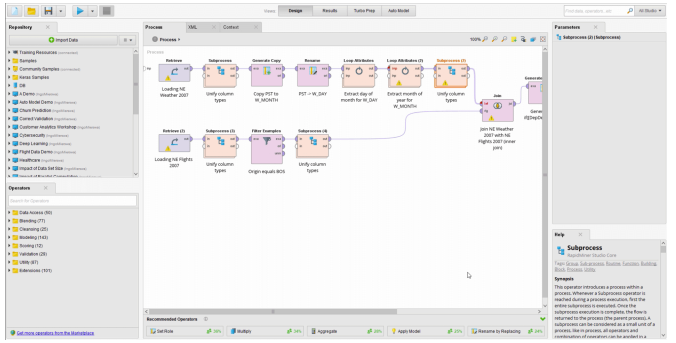
Rising Popularity of Python



Source: https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

- **Python** & **R** currently represent the **most popular programming languages** in the data science area

- **RapidMiner** has become the **leading GUI-based tool** for conducting data science projects and its popularity is increasing even further



RapidMiner Overview

- Interpreted, high-level, general-purpose programming language

- **Origin**: Developed by *Guido van Rossum* in Amsterdam at the Dutch National Research Institute for Mathematics and Computer Science
  - **Goal**: Promote simplicity = a programming language should be *easy-to-use* & *easy-to-read*
  - **Multi-paradigm language**: Supports *object-oriented*, *structured*, *functional*, *aspect-oriented,* and more programming paradigms
  - **Integration of other programming languages**: If speed is important, time-critical functions can be moved to extension modules written in other programming languages, e.g., C

- Currently, **two Python versions** are used side by side by developers
  - **Latest Version**: Python 3
  - **Still widely used**: Python 2 *(official support will end after 2020!)*

  → With Python 3, many fundamental changes were introduced

  → Many libraries were not transferred to Python 3, so developers have to keep using Python 2 when they want to use such libraries in their Python code

  → As a result, developers have to switch between both versions depending on their use of libraries

Programming Philosophy: The “Zen of Python”

- **Goal**: Promote simplicity = a programming language should be *easy-to-use* & *easy-to-read*

Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python in a Nutshell: Indentations, Variables, Comments

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

I. Indentations

- In most programming languages, the indentation of code is only used to improve readability without affecting any semantics
  - ➢ In Python, the indentation matters as it is used to define code blocks!
    - **Example**: Only the left code snippet is correct:

```
if(x < y):
  print(y)
```

```
if(x < y):
print(y)
```

      - ➢ **Why?**

        → In the **right code snippet:** The print function is on the same level as the if statement, so the consequent, which is aimed to be executed if the if-condition is true, is missing

        → In the **left code snippet:** The consequent is defined through the indentation of the print function

II. Variables

- When declaring variables, no data type needs to be indicated explicitly
  - ➢ The data type is automatically determined based on the declared value
    - ```
      a = 5
      b = -0.537
      c = "hello!"
      ```

III. Comments

- Single-line comments are indicated with a hash sign: #
- Multi-line comments are indicated by surrounding the text with three apostrophes: ''' '''

```
# this is a single-line comment
print("hello!")
'''
this multi-line comment
can cover more than just
one line
'''
```
```

Python in a Nutshell: Basic Built-in Data Types

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

Value Type	Description	Examples
boolean	• Represent either true or false	*True, False*

Numeric ***(int, long, float, complex)***	• **int**: a whole number
	• **long**: an integer of unlimited size
	• **float**: a real number written with a decimal point
	• **complex**: a number of the form a + bJ, with:
	  ○ a & b: floats
	  ○ J: square root of -1 (= imaginary number)
		• **int**: *1, -5, -200, 5432, ...*
		• **long**: *1, -5, -200, 5432, 43243944949494994, ...*
		• **float**: *0.54, -0.63678, 0.0005, ...*
		• **complex**: *2+3j, 5j, 100+6j, ...*

String	• Representation of text	*“hello“, “a“, “xyz43[4]“, ...*
Collections (Arrays) ***(list, tuple, set, dictionary)***	• **list**: a collection which is *ordered* and *changeable*; *allows* duplicates • **tuple**: a collection which is *ordered* and *unchangeable*; *allows* duplicates • **set**: a collection which is *ordered* and *unindexed*; *does not allow* duplicates • **dictionary**: a collection which is *unordered*, *changeable*, and *indexed*; *does not allow* duplicates	• ***list/set****: [“banana“, “apple“, “peach“], [1.1, 0.583, 0.001, 54.43], ...* • ***tuple****: (“banana“, “apple“, “peach“), (1.1, 0.583, 0.001, 54.43), ...* • **dictionary***:* *{* *“firstName“ : “Ronald“,* *“lastName“ : “McDonald“,* *“yearOfBirth“ : 1981* *}*

Python in a Nutshell: Basic Statements

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

- **def:** *Define a function*
  → functions can return multiple values!

```
def myAwesomeFunction():
  x = 1
  y = 2
  return x,y

def main():
  a, b = myAwesomeFunction()
  print(b)
```

- **class:** *Define a class*

```
class student():
  def __init__(self, firstName, lastName, age):
     self.firstName = firstName
     self.lastName = lastName
     self.age = age
```

- **if/else:** *Define a conditional control flow*

```
if x == 42:
  print("I also like the Hitchhikers guide to the galaxy")
elif x < 35:
  x = x + 1
else:
  x = x * 2
```

- **for & while loop:** *Define a loop with some ending condition*

```
z = 0

for x in range(4,10):
  z = z + x
  print(z)

while z < 1000:
  print(z)
  z = z + 1
```

Important Python Libraries for Data Science

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

I. Data Structures & Statistics

- *NumPy*
- *Pandas*
- *...*

II. Visualization

- *Matplotlib*
- *Plotly*
- *...*

III. Machine Learning

- *Scikit-learn*
- *Tensorflow*
- *Keras*
- *XGBoost*
- *Pytorch*
- *...*

→ **Many more useful libraries for data science exist!**

→ See, e.g., these (non-scientific) overview articles:

- Medium (2018): *"Top 20 Python libraries for data science in 2018"*
- Kdnuggets (2018): *"Top 10 Python Data Science Libraries"*
- *Big Data Made Simple (2019): "Top 20 Python libraries for Data Science"*
- *...*

Outline

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Introduction to Python

1.1 Overview

1.2 Data Import & Management

1.3 Data Visualization & Exploration

1.4 Development Environments

1.5 Resources & Hands-On Exercises

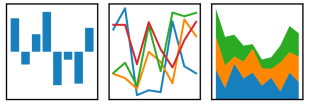
Pandas: Data Import & Export (1/2)

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



- The Python library *pandas* provides a rich set of data **import** and **export** capabilities
- Both **flat-file-based** (e.g., CSV, JSON, Excel, ...) & **server-based** (e.g., SQL, Google Big Query, ...) imports and exports are supported:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

Source & more information: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Pandas: Data Import & Export (2/2)

- Each import operator returns a pandas **dataFrame** (see next slide)

- **Example**: *read_csv()*

```
import pandas as pd

data = pd.read_csv('URL/To/Some/CSV/file.csv', delimiter=',')
print(data)
```

  - **Result output example**:

	last_name	first_name	age	student?
0	Skywalker	Luke	22	True
1	Targaryen	Daenerys	22	False
2	Weasley	Ronald	13	True
3	White	Walter	50	False

  - The import functions automatically identify fitting variable types for the values contained in the indicated file, e.g., in the example above:
    - **all values** of ***last_name & first_name***: *str*
    - **all values** of ***age***: *integer*
    - **all values** of ***student?***: *boolean*

Pandas: DataFrame – The primary data structure

- *"**Two-dimensional size-mutable**, **potentially heterogeneous** tabular data **structure** with **labeled axes** (rows and columns). [...]. The primary pandas data structure."*

  – Pandas dataFrame API Reference

  → i.e.: a modifiable table-like data structure with namable columns and rows that can hold different data types (e.g., one column may hold integers while another one holds booleans)

DataFrame Example:

	last_name	first_name	age	student?
0	Skywalker	Luke	22	True
1	Targaryen	Daenerys	22	False
2	Weasley	Ronald	13	True
3	White	Walter	50	False

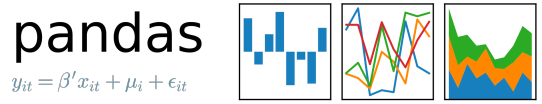
- DataFrames are heavily used in data science projects as they allow for an ***easy*** and ***efficient* data management**
  - Enable efficient column-wise data manipulations which are especially useful for data preparation
  - Other popular libraries (e.g., scikit-learn) are generally used in combination with pandas dataFrames

Pandas: Data Selection (with dataFrames)

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

- Based on some pandas *dataFrame* df, multiple possibilities exist to select a required data subset. Some popular approaches for selecting data subsets are:

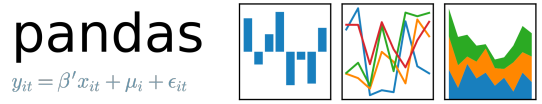


Approach	Focus	Example Code	Example Code Explanation
(1) By column name	**Single column**	• `df["last_name"]`	• Select column named "last_name"
	Multiple columns	• `df[["last_name", "age"]]`	• Select the columns named "last_name" and "age" (in this order)
(2) By position	**Single row**	• `df.iloc[1]`	• Select second row
	Range of rows	• `df.iloc[:3]` • `df.iloc[1:4]` • `df.iloc[[1,3,2]]`	• Select row number 0 to 2 • Select row number 1 to 3 • Select row number 1, 3, and 2 (in this order)
	Single column	• `df.iloc[:,2]`	• Select third column
	Range of columns	• `df.iloc[:,:4]` • `df.iloc[:,1:3]` • `df.iloc[:,[1,0,2]]`	• Select column number 0 to 3 • Select column number 1 to 2 • Select column number 1, 0, and 2 in this order
	Single cell	• `df.iloc[1,2]`	• Select cell in row 1 and column 2
	Range of cells	• `df.iloc[1:3,[1,0,2]]`	• Select row 1 to 2 and column 1, 0, and 2 (in this order)
(3) Condition-based	**Condition: minimum age**	• `df[df["age"] > 15]`	• Select every row where age > 15
	Condition: specific first name	• `df[df["first_name"] == "Luke"]`	• Select every row where first_name equals "Luke"
(4) Query method	**Query focus: age range**	• `df.query('(age < 30) & (age > 15)')`	• Select every row where 30 > age > 15
	Query focus: specific first name & minimum age	• `df.query('(first_name != "Luke") & (age > 15)')`	• Select every row where first_name is not "Luke" and age > 15

For the example code snippets, imagine you have the following dataFrame stored in a variable called df:

	last_name	first_name	age	student?
0	Skywalker	Luke	22	True
1	Targaryen	Daenerys	22	False
2	Weasley	Ronald	13	True
3	White	Walter	50	False

Pandas: Data Manipulation (with dataFrames)



- Pandas and other Python libraries offer a variety of possibilities for data manipulation that can be used to perform an efficient data preparation
- As the required functionality varies from project to project, we recommend that you explore possible capabilities by your own based on some specific use case

  → See, e.g., pandas documentation

- However, to provide you with some basic insights, capabilities for handling missing data are discussed in the following slides

Pandas: Handling missing data (with dataFrames) (1/2)

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

NumPy

1) Identify missing data

a. In Python, two value types may indicate missing values:

- **None**: official built-in Python object used to indicate a missing value
  - → clearly indicates a missing value ☺
  - → produces unnecessary processing overhead & must be treated as a special value ☹
- **NaN**: value type contained in the *numpy* library and is used to indicate that a variable is *"Not a Number"* (= NaN)
  - → very efficient & can be easily treated like a number (i.e., can be used in arithmetic) ☺
  - → does only represent missing numerical data & always requires the *numpy* library ☹

b. Luckily, pandas treats both value types interchangeable & provides two helpful functions to identify them:

- **isnull():** generates a table filled with booleans indicating missing values (= values which are either *None* or *NaN*)
- **notnull():** generates the opposite of *isnull()*

last_name	first_name	age	student?
Skywalker	Luke	*NaN*	True
Targaryen	Daenerys	22	False
Weasly	*None*	13	True
White	Walter	50	False

isnull()

last_name	first_name	age	student?
False	False	***True***	False
False	False	False	False
False	***True***	False	False
False	False	False	False

Pandas: Handling missing data (with dataFrames) (2/2)

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

NumPy

2) Handle missing data

- Pandas provides two helpful functions to handle missing values (= *None* & *NaN)*:
  - **dropna():** Returns a filtered version in which rows containing any missing values are discarded
  - **fillna():** Returns a copy of the provided data set in which missing values are replaced with an indicated value or based on a defined replacement method* (i.e.: *'backfill', 'bfill', 'pad', 'ffill'*)

last_name	first_name	age	student?
Skywalker	Luke	*NaN*	True
Targaryen	Daenerys	22	False
Weasly	*None*	13	True
White	Walter	50	False

dropna()

last_name	first_name	age	student?
Targaryen	Daenerys	22	False
White	Walter	50	False

last_name	first_name	age	student?
Skywalker	Luke	*NaN*	True
Targaryen	Daenerys	22	False
Weasly	*None*	13	True
White	Walter	50	False

fillna(value="missing")

last_name	first_name	age	student?
Skywalker	Luke	***missing***	True
Targaryen	Daenerys	22	False
Weasly	***missing***	13	True
White	Walter	50	False

last_name	first_name	age	student?
Skywalker	Luke	22	True
Targaryen	Daenerys	22	False
Weasly	*None*	13	True
White	Walter	*NaN*	False

fillna(method="ffill")

last_name	first_name	age	student?
Skywalker	Luke	22	True
Targaryen	Daenerys	22	False
Weasly	***Daenerys***	13	True
White	Walter	***13***	False

*See: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

Outline

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Introduction to Python

1.1 Overview

1.2 Data Import & Management

1.3 Data Visualization & Exploration

1.4 Development Environments

1.5 Resources & Hands-On Exercises

Data Visualization with Python

- Python offers a big variety of visualization libraries
- Popular libraries are (inter alia): matplotlib & plotly
- But many more exist that offer varying foci!
  → Please feel free to explore any other libraries to identify the ones that fit your needs ☺

- In the following slides, some basic visualizations are discussed to provide you some insights on how to use the libraries for visualizing your data
  → **However**: This is just the tiny tip of the iceberg of visualizations you can generate! (Again: It's worth to explore the libraries!)

Plotting Data with *matplotlib*

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

matplotlib

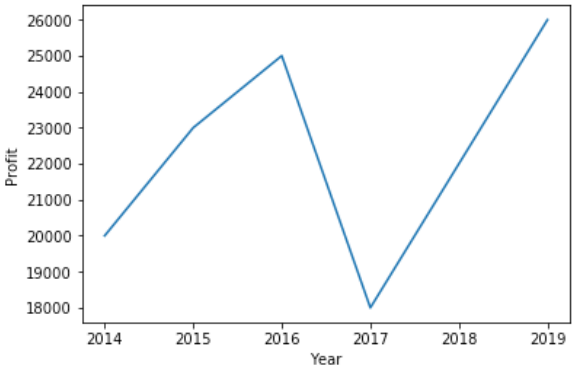
a) Use matplotlib *explicitly* to plot data

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

# import data from a csv file into a pandas dataFrame:
data = pd.read_csv('profitPerYear.csv', delimiter=',')

# plot the two dimensions year and profit and name the x and y axis accordingly:
plt.plot(data['year'], data['profit'])
plt.xlabel('Year')
plt.ylabel('Profit')
```

output



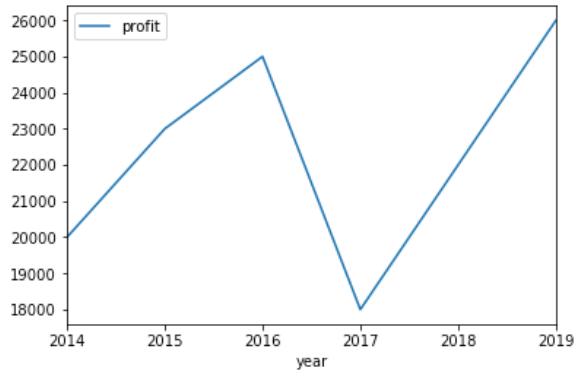
a) Use matplotlib *implicitly* to plot data (with a pandas dataFrame)

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

# import data from a csv file into a pandas dataFrame:
data = pd.read_csv('profitPerYear.csv', delimiter=',')

# plot the dataframe based on the indicated values for x and y axis:
data.plot(x='year', y='profit')
```

output



Example dataFrame:

year	profit
2014	20000
2015	23000
2016	25000
2017	18000
2018	22000
2019	26000

Create a Bar Chart with *matplotlib*

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

matplotlib

a) Use matplotlib *explicitly* to create a bar chart

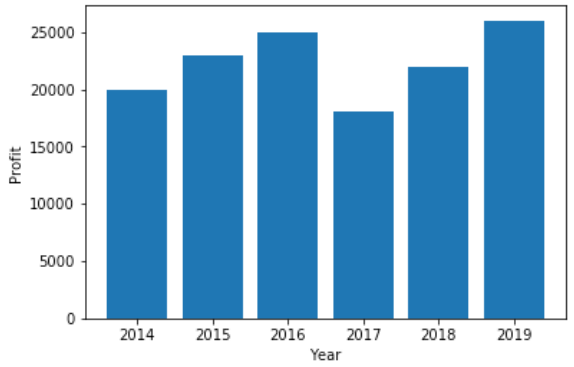
- Simply replace the function *plot()* with *bar()*

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

# import data from csv into a pandas dataFrame:
data = pd.read_csv('profitPerYear.csv', delimiter=',')

# plot the two dimensions year and profit and name the x and y axis accordingly:
plt.bar(data['year'], data['profit'])
plt.xlabel('Year')
plt.ylabel('Profit')
```

output



Example dataFrame:

year	profit
2014	20000
2015	23000
2016	25000
2017	18000
2018	22000
2019	26000

b) Use matplotlib *implicitly* to create a bar chart (with a pandas dataFrame)

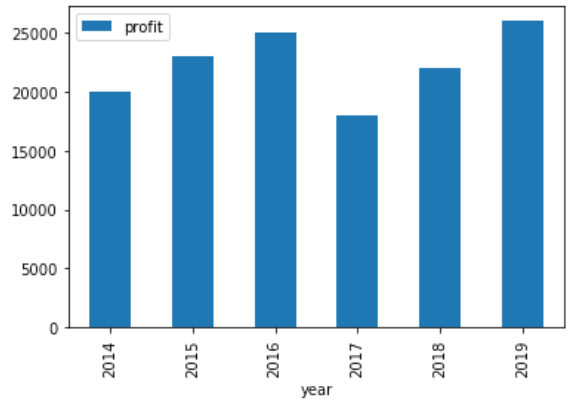
- Simply indicate *'bar'* as chart type with the additionally introduced *kind* parameter

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

# import data from csv into a pandas dataFrame:
data = pd.read_csv('profitPerYear.csv', delimiter=',')

# plot the dataframe based on the indicated values for x and y axis:
data.plot(x='year', y='profit', kind='bar')
```

output



Outline

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Introduction to Python

1.1 Overview

1.2 Data Import & Management

1.3 Data Visualization & Exploration

1.4 Development Environments

1.5 Resources & Hands-On Exercises

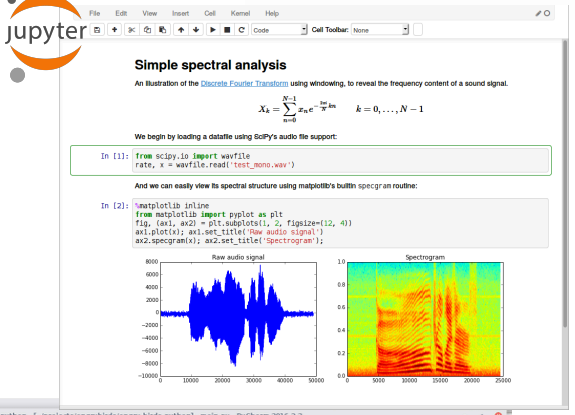
Overview of Python Development Environments

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

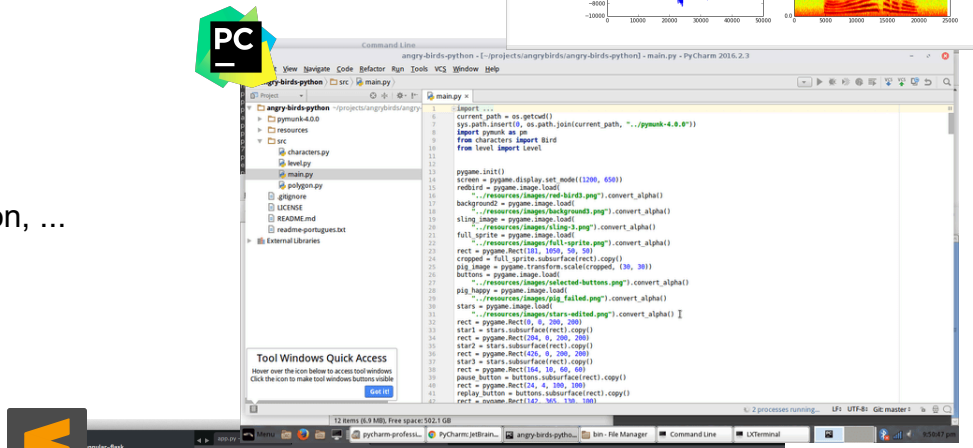
a) Interactive Notebooks: Jupyter Notebooks

- Document-like coding environment: allows to structure & execute your code along text snippets
- Allows a straightforward documented script-like programming
- Currently very popular in the data science community as it allows an easy documented exploration of data & AI algorithms! (but not really suited to implement huge applications)



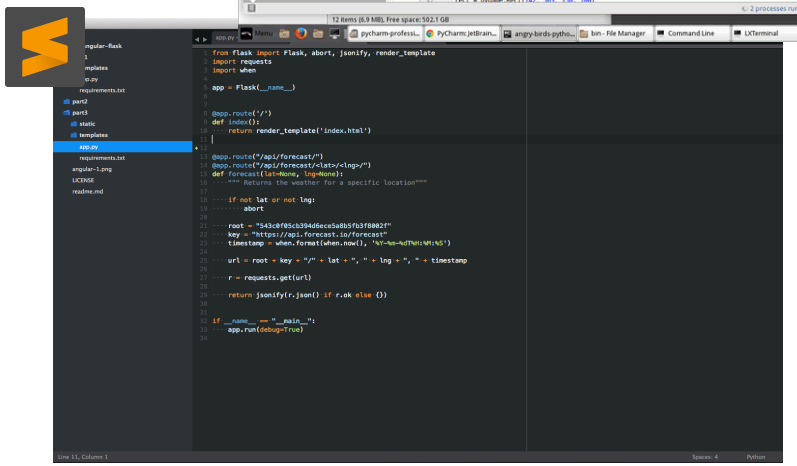
b) Python-specific IDE: PyCharm

- Integrated development environment (IDE) for Python developed by JetBrains
- Includes: code analysis, a graphical debugger, an unit tester, version control integration, ...
- A commercial and a (free) community version exists
- Especially useful for managing big complex programming projects
- **Alternatives**: Spyder, Thonny, ...



c) General Code/Text Editor: Sublime Text

- Lightweight editors that provide helpful capabilities for text editing, such as, e.g., text highlighting based on the chosen programming language
- Especially useful for quickly creating scripts
- **Alternatives**: Atom, Visual Studio Code, ...



Anaconda

- Open-source-distribution of Python (commercial version exists as well)
- Very popular among data scientists!

  → As it comes with many tools and libraries used in data science, including:
  - A package manager (called *"conda"*) for installing and maintaining Python libraries
  - Environment manager for managing different installations of Python (e.g., to switch between a Python 2 and 3 installation)
  - Development tools like Jupyter Notebook
  - ...and much more!
- See: https://www.anaconda.com/

Outline

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

1 Introduction to Python

1.1 Overview

1.2 Data Import & Management

1.3 Data Visualization & Exploration

1.4 Development Environments

1.5 Resources & Hands-On Exercises

Helpful Resources

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Official Documentation:**
  - Python 3
  - Python 2
- **Free eBook:**
  - Swaroop C H (2013): *„A Byte of Python“*
    *(**PDF version**: scroll „introduction“ page down for a download link)*
- **Free Online Courses:**
  - Codecademy’s Python 2 & 3 courses
  - Google’s Python Class
  - Tutsplus Python Tutorial
    (combines many further resources)

Try it yourself!

WIRTSCHAFTS INFORMATIK

TECHNISCHE UNIVERSITÄT DARMSTADT

- Install the most recent version of the Anaconda Python distribution (Python 2 or 3 version)
  - **URL**: https://www.anaconda.com/distribution/#download-section
- Explore the data management and exploration capabilities of Python

  by completing the tasks of the exercise 2 sheet