INF 354
Python
Data Packages

Lecture 11
9 May 2019

INFORMATIKA • DEPARTEMENT • INFORMATICS • DEPARTMENT
Universiteit van Pretoria
University of Pretoria

The contents of this lecture are as follows:

- Python packages
- A brief introduction to Data Science
- Numpy
- Matplotlib
- Pandas

Packages

- Functions and Methods available in the Python ecosystem can be very useful
- Since there are thousands of functions and methods available, you cannot keep and maintain all those in your project.
- That is where **packages** come in
- Package = Directory of Python Scripts
- Each script in a package is called a module
- These modules specify functions, methods and new Python types (i.e. classes) aimed at solving particular problems
- These packages aren’t installed with Python by default – you have to import them
- You can use pip – a package manager for Python
- pip would have been installed when you installed Python

Installing and using packages

- Suppose we want to install the Numpy package. (We'll discuss Numpy in a bit)
- Use pip:

```
SET HTTPS_PROXY=http://[username:password@]proxyserver:portnumber
```

Configure Proxy

```
pip3 install numpy
```

- Now that Numpy is installed, you can use it in your Python script
  1. Import whole package or specific module from package
  2. Use functions, methods or types provided by package

```python
import numpy

numpy.array([1, 2, 3])
```

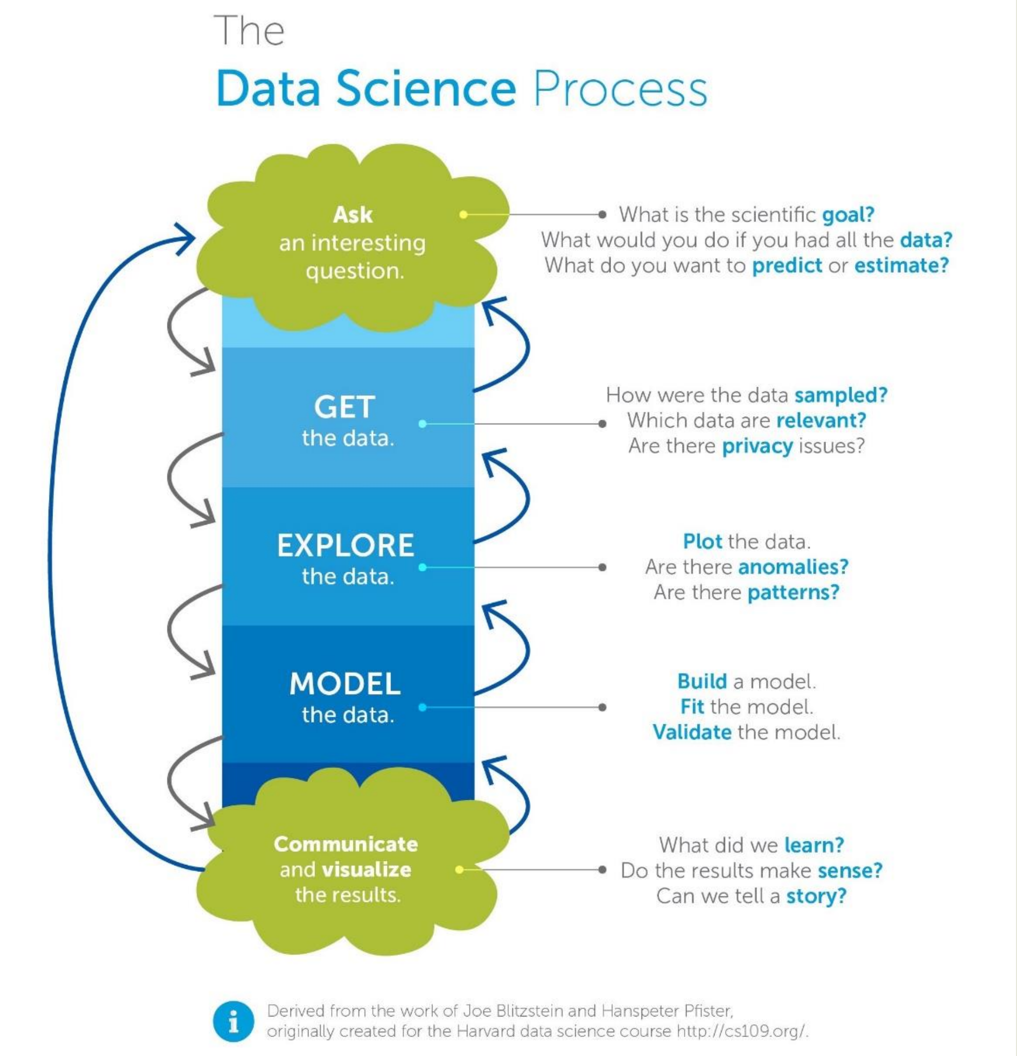
```python
import numpy as np

np.array([1, 2, 3])
```

```python
from numpy import array

array([1, 2, 3])
```

A brief introduction to Data Science

- Data science is the field of study that combines domain expertise, programming skills, and knowledge of math and statistics to extract meaningful insights from data.
- Analysts and business users can then translate these insights into tangible business value.
- It may involve the application of machine learning algorithms to numbers, text, images, video, audio, and more to perform tasks which ordinarily require human intelligence, such as
  - Automated data entry
  - Spam detection
  - Recommendation engines
  - Medical diagnoses
  - Intelligent marketing (customer segmentation, churn prediction, customer value management)
  - Financial analysis (e.g. algorithmic trading, portfolio management, fraud detection, etc.)
  - Predictive maintenance
  - Image recognition

Typical Data Science Process



Limits of built-in Python lists

- Python lists are quite powerful:
  - It can hold collections of different types (string, number, object, list)
  - It can be manipulated with ease (change, add, or remove elements)
- However, data scientists often need to carry out mathematical/statistical operations over entire collections of values – and they want it done **fast**.
- Suppose you have two lists, one holding the heights and one holding the weights of each person in a group. Now, what happens when we try to calculate each person's BMI:

```
In [1]: height = [1.73, 1.68, 1.71, 1.89, 1.79]

In [2]: height
Out[2]: [1.73, 1.68, 1.71, 1.89, 1.79]

In [3]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]

In [4]: weight
Out[4]: [65.4, 59.2, 63.6, 88.4, 68.7]

In [5]: weight / height ** 2
TypeError: unsupported operand type(s) for **: 'list' and 'int'
```

Python does not support math on entire lists

Looping through the list will be slow

Numpy

- Numpy (Python package) is short for Numeric Python
- Alternative to the regular Python list: **Numpy Array**
- Numpy array is similar to Python list
- Calculations over entire arrays
- Easy and Fast

```
import numpy as np

np_height = np.array(height)

np_height
array([ 1.73,  1.68,  1.71,  1.89,  1.79])

np_weight = np.array(weight)

 np_weight
 array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

```
bmi = np_weight / np_height ** 2

bmi
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

Unique Numpy Behaviour

- Numpy arrays contain only one type.
- Some operations behave differently to standard Python lists:

```
python_list = [1, 2, 3]

numpy_array = np.array([1, 2, 3])

python_list + python_list
[1, 2, 3, 1, 2, 3]

numpy_array + numpy_array
array([2, 4, 6])
```

+ operator

Python Lists: Appends two lists

Numpy Arrays: Adds elements of same index

Quick Numpy Array Queries

Getting value at index of array (same as Python list)

Testing all values for some specific condition

Getting a subset of array containing only values that meet some condition

```
bmi
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

bmi[1]
20.975

bmi > 23
array([False, False, False,  True, False], dtype=bool)

bmi[bmi > 23]
array([ 24.747])
```

2D Numpy Arrays

```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, 68.7]])
```

```
np_2d.shape
(2, 5)
```

2 rows, 5 columns

```
                0       1       2       3       4
array([[  1.73,   1.68,   1.71,   1.89,   1.79],   0
       [  65.4,   59.2,   63.6,   88.4,   68.7]])   1
```

```
np_2d[0][2]
1.71

np_2d[0,2]
1.71
```

```
np_2d[:,1:3]
array([[  1.68,   1.71],
       [ 59.2 ,  63.6 ]])
```

```
np_2d[1,:]
array([ 65.4,  59.2,  63.6,  88.4,  68.7])
```

Basic Statistics with Numpy

- First step in analysing data is exploration (getting to know your data)
- The list and array examples we've looked at so far have been easy to explore simply by looking at it.
- Data Science usually involves processing thousands, millions, or even billions of numbers

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

In [2]: np_city = ... # Implementation left out

In [3]: np_city
Out[3]:
array([[  1.64,  71.78],
       [  1.37,  63.35],
       [  1.6 ,  55.09],
       ...,
       [  2.04,  74.85],
       [  2.04,  68.72],
       [  2.01,  73.57]])
```

- This is a 2D Numpy array with 5000 rows and 2 columns
- Each row represents a person record
- The first column represents that person's height
- The second column represents that person's weight

- Numpy arrays offer basic statistical methods to explore large data sets such as this one.

Basic Statistics with Numpy

```
In [4]: np.mean(np_city[:,0])
Out[4]: 1.7472

In [5]: np.median(np_city[:,0])
Out[5]: 1.75

In [6]: np.corrcoef(np_city[:,0], np_city[:,1])
Out[6]:
array([[ 1.      , -0.01802],
       [-0.01803,  1.      ]])

In [7]: np.std(np_city[:,0])
Out[7]: 0.1992
```

- Mean = arithmetic average of a collection's values
- Median = value lying at the midpoint of a sorted collection.
- Correlation coefficients = measure of the strength of the relationship between two variables. (e.g. height and weight)
- Standard Deviation = measure of how spread out numbers are.
- Numpy arrays also have faster implementations of sum() and sort()

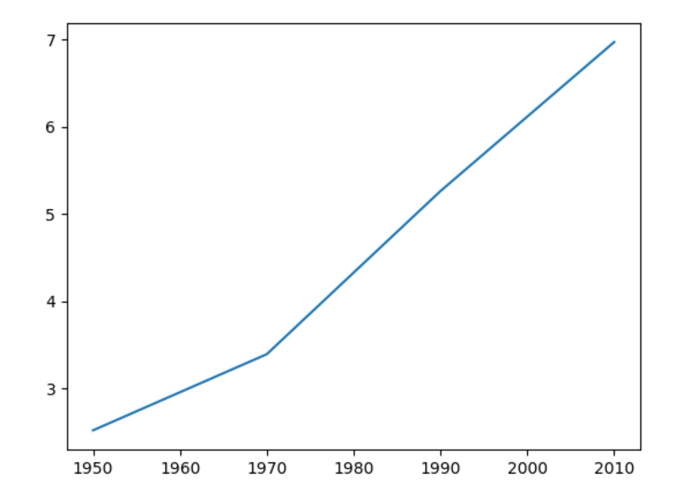
Matplotlib

- Data visualisation helps with the data analysis process in 2 ways:
    1. Exploring data
    2. Reporting insights
- Matplotlib is a Python package for data visualisation.

pip3 install matplotlib

- It has set of charting functions, like plot which generates a line chart

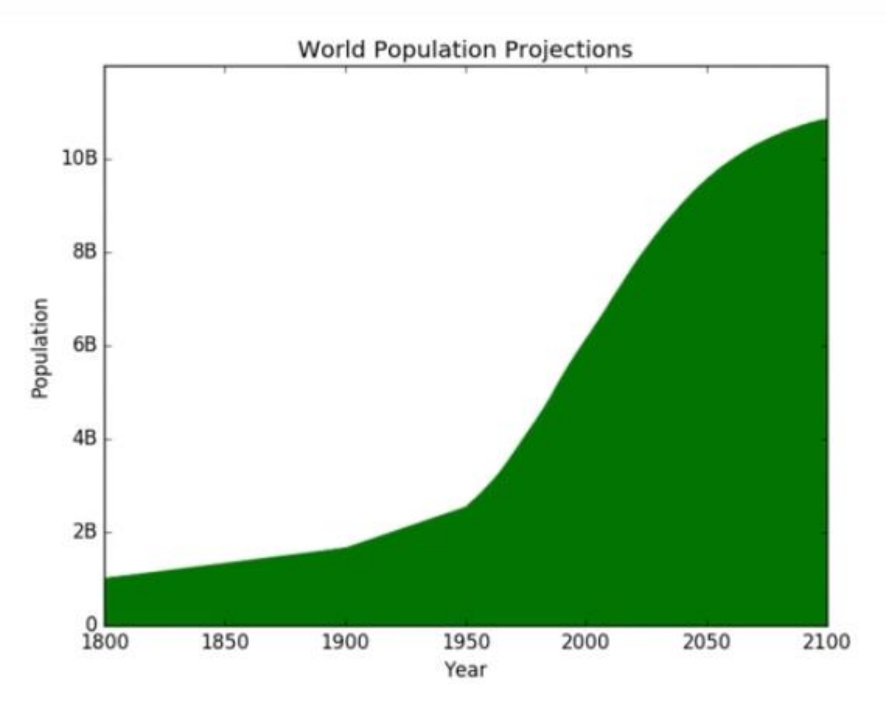
```python
import matplotlib.pyplot as plt

year = [1950, 1970, 1990, 2010]
pop = [2.519, 3.392, 5.263, 6.972]

plt.plot(year, pop)

plt.show()
```



Matplotlib - Customisation

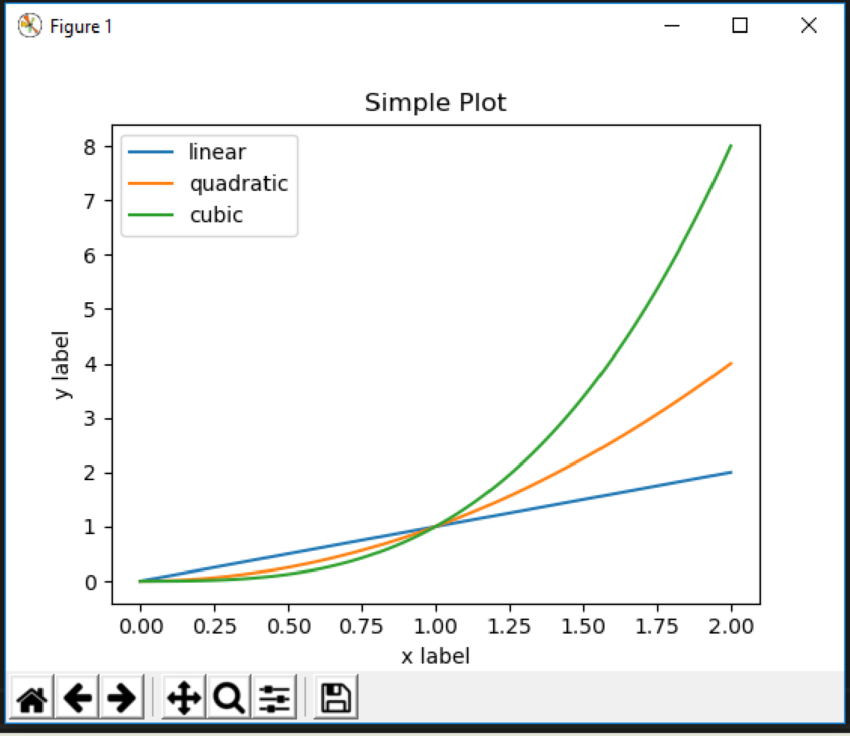
- You can also customise your visualisation by setting certain properties.

```
plt.fill_between(year,population,0,color='green')

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')

plt.yticks([0,2,4,6,8,10],
           ['0','2B','4B','6B','8B','10B'])

plt.show()
```



Matplotlib

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

x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```



For more on matplotlib: https://matplotlib.org/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py

Pandas

- Numpy arrays are useful for data manipulation and fast calculations **BUT** they only hold values of a single type
- Big Data is however known for is volume and **variety**, so you cannot expect all data to be of the same type.
- Pandas is a high-level data manipulation package that extends Numpy – it has most of the same easy and fast methods of Numpy but has added methods and works with different data types
- Pandas stores data in a **dataframe** (essentially 2D tables):

```
In [1]: brics = ... # declaration left out

In [2]: brics
Out[2]:

         country  population      area    capital
BR        Brazil         200   8515767   Brasilia
RU        Russia         144  17098242     Moscow
IN         India        1252   3287590  New Delhi
CH         China        1357   9596961    Beijing
SA  South Africa          55   1221037   Pretoria
```

Pandas

- Dataframes are typically not built manually. Instead, you import data from an external file (such as CSV file)

brics.csv

```
,country,population,area,capital
BR,Brazil,200,8515767,Brasilia
RU,Russia,144,17098242,Moscow
IN,India,1252,3287590,New Delhi
CH,China,1357,9596961,Beijing
SA,South Africa,55,1221037,Pretoria
```

```python
brics = pd.read_csv("path/to/brics.csv", index_col = 0)
```

	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221037	Pretoria

Pandas

- Data can be accessed in various ways

Column Access

```
In [8]: brics["country"]
Out[8]:

BR           Brazil
RU           Russia
IN            India
CH            China
SA     South Africa
Name: country, dtype: object
```

```
In [9]: brics.country
Out[9]:

BR           Brazil
RU           Russia
IN            India
CH            China
SA     South Africa
Name: country, dtype: object
```

```
In [10]: brics["on_earth"] = [True, True, True, True, True]

In [11]: brics
Out[11]:
         country  population      area    capital on_earth
BR        Brazil         200   8515767   Brasilia     True
RU        Russia         144  17098242     Moscow     True
IN         India        1252   3287590  New Delhi     True
CH         China        1357   9596961    Beijing     True
SA  South Africa          55   1221037   Pretoria     True
```

Add New Column

```
In [12]: brics["density"] = brics["population"] / brics["area"] * 1000000

In [13]: brics
Out[13]:

         country  population      area    capital on_earth     density
BR        Brazil         200   8515767   Brasilia     True   23.485847
RU        Russia         144  17098242     Moscow     True    8.421918
IN         India        1252   3287590  New Delhi     True  380.826076
CH         China        1357   9596961    Beijing     True  141.398928
SA  South Africa          55   1221037   Pretoria     True   45.043680
```

Add New Calculated Column

Row Access

```
In [14]: brics.loc["BR"]
Out[14]:

country           Brazil
population           200
area             8515767
capital         Brasilia
density         23.48585
on earth            True
Name: BR, dtype: object
```

Single Cell Access

```
brics.loc["CH","capital"]
Beijing

brics["capital"].loc["CH"]
Beijing

brics.loc["CH"]["capital"]
Beijing
```

Video

- https://www.youtube.com/watch?v=a9UrKTVEeZA&t=835s

*Note the producer of this video works in a different IDE (namely Jupyter). However, he uses Python so the code will work just as well in VS Code.