01_slides_jupyter

December 9, 2024

1 AICE1006 - Data Analytics

1.1 Lecture 1 - Introduction

Zhiwu Huang Lecturer (Assistant Professor) Vision, Learning and Control (VLC) Research Group School of Electronics and Computer Science (ECS) University of Southampton

Office Hour: Wed 2PM-3PM, Please book in advance. `Zhiwu.Huang@soton.ac.uk`

Credit: Marco Forgione, Researcher, SUPSI-USI

2 Overview

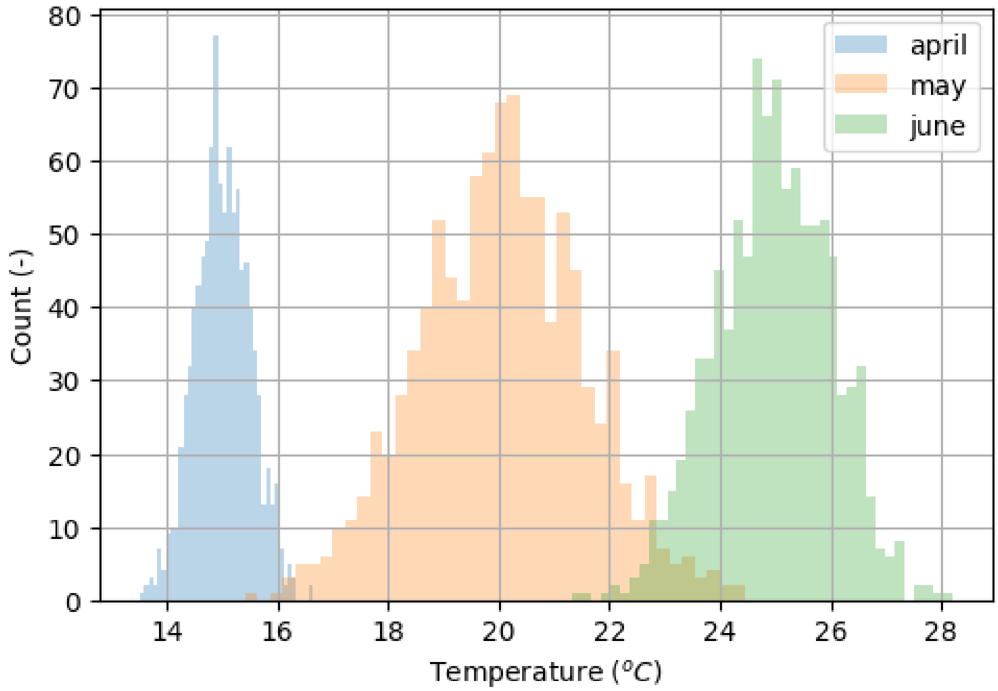
Objective: * To improve your programming skills with focus on data science applications * To get acquainted with the most common data manipulation techniques * To develop an intuitive understanding of real-world datasets

Main Topics: * Data ingestion (how to load data) * Data wrangling (concatenate, join, pivot,…) * Data visualization (scatter plots, histograms, box plots,…) * Basic data analysis (mean, standard deviation, correlation,…) * Advanced data analysis (pricipal component analysis, manifold learning, clustering,…) * More advanced (advanced machine/deep learning,…): NOT INCLUDED IN AICE1006

You may see similar concepts in the other modules like AICE1008: Mathematics. The focus of the Data Analytics module is on applications rather than theory!

2.1 Data Visualization Examples

[1]:
```
import matplotlib.pyplot as plt; import numpy as np
temp_april = np.random.normal(15, 0.5, 1000) # mean, std, n_sample
temp_may = np.random.normal(20, 1.5, 1000); temp_june = np.random.normal(25, 1,␣
↪1000)
kwargs = dict(histtype='stepfilled', alpha=0.3,  bins=40)
plt.figure(figsize=(6, 4)); plt.hist(temp_april, **kwargs, label='april')
plt.hist(temp_may, **kwargs, label='may'); plt.hist(temp_june, **kwargs,␣
↪label='june')
plt.xlabel('Temperature $(^oC)$', fontsize=10); plt.ylabel('Count (-)',␣
↪fontsize=10);
plt.grid(); plt.legend(fontsize=10);
```



2.2 Data Visualization Examples

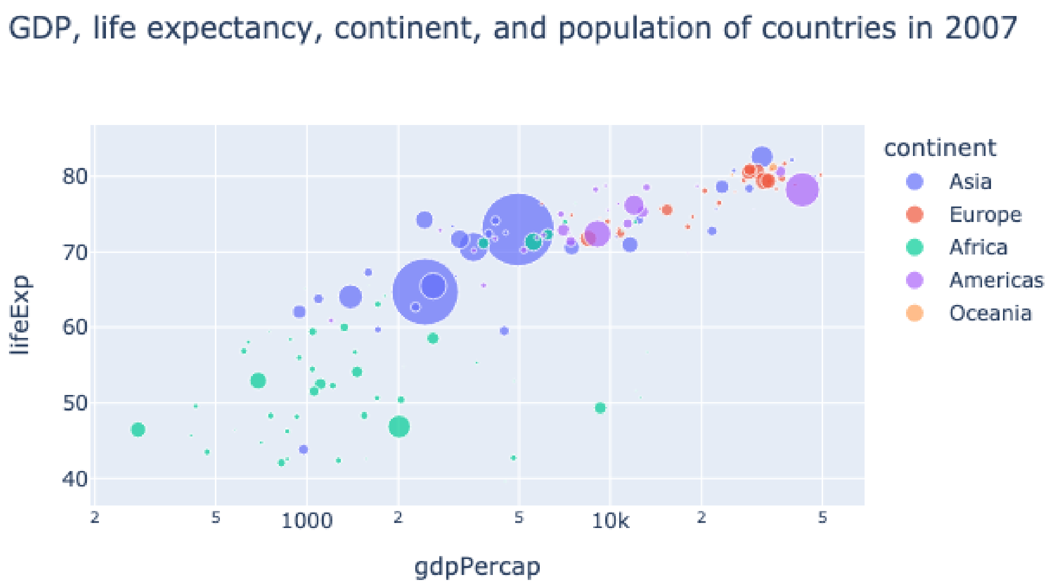
[4]:
```
### Scatter plot
import plotly.express as px; from IPython.display import Image
data_canada_it = px.data.gapminder().query("country == 'Canada' or country ==␣
↪'Italy'")
fig = px.bar(data_canada_it, x='year', y='pop', color="country",␣
↪barmode="group", width= 800, height=400)
fig.show();
# fig.write_image('figure.png');Image(filename='figure.png')
```

The signal $y(t)$ follows a **sinuisoidal pattern** over time.

2.3 Data Visualization Examples

[ ]:
```
import plotly.express as px; from IPython.display import Image
df = px.data.gapminder()
# A richer scatterplot
fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp",␣
↪size="pop", color="continent", hover_name="country", log_x=True,
           title="GDP, life expectancy, continent, and population of countries␣
↪in 2007", size_max=30, width=650, height=400)
fig.show(); #fig.write_image('figure.png');Image(filename='figure.png')
```

[ ]:



3 Software tools

- Python (programming language)
- Jupyter Notebook (web-based programming interface)
- Python libraries:
  - Numpy (numerical mathematics)
  - **Pandas** (tabular data handling)
  - Matplotlib, Pandas, and Seaborn (data visualization)
  - …

3.1 Resources

Required: * The course's slides * Official documentation of python, pandas, matplotlib, … * Google, stack overflow, …

Optional: * The book "Python Data Science" by Jake Vanderplas (available on-line for free) * The book "Python for Data Analysis" by Wes McKinney (link on the publisher's website)

4 Getting started

- Install all the required packages. Type in a terminal:

```
$ pip install numpy scipy pandas scikit-learn matplotlib jupyter
$ pip install rise
$ pip install plotly
```

- Start Jupyter notebook. Type in a terminal:

```
$ jupyter notebook
```

A page should appear in your default browser. Otherwise, copy-paste the link from your terminal in the browser.

On Windows, Chrome seems to work better.

5 Introduction to Jupyter notebook

A web-based Python environment very good for: * Rapid prototyping * Small/medium data science projects * Presenting results (these slides made with Jupyter!)

Jupyter allows mixing **formatted text** and python code. The cell below contains Python code!

[5]:
```python
print("Welcome to the data challenge module!")
#a = 1
#b = a + 1
#print(b)
```

```
Welcome to the data challenge module!
```

Don't use Jupyter for a large software development project! Go for an IDE like PyCharm or VSCode instead.

5.1 Cell

The cell is the basic input unit in a Jupyter notebook. Common operations:

- Add a cell
  - Select an existing cell by clicking in the white area at its left
  - Click on Insert -> Insert Cell Above/Below (or click a/b)
- Remove a cell
  - Select a cell
  - Click on Edit -> Delete Cells (or click x)
- Move a cell
  - Select the cell (hold shift to select multiple cells)
  - Use the arrows to move the cell up and down

Cell type

- A cell may contain either Python code or Markdown (= formatted text).
- This slide is made up of Markdown cells!
- To change the cell type, select Python or Markdown in the drop-down menu

6 Python for data science

Python is an *interpreted*, *dynamically typed* programming language. * No need to compile the code. A virtual machine interprets it on the fly. * No need to declare variable types. They are inferred at run-time (dynamic typing).

[6]:
```python
# Sum numbers from 0 to 9
result = 0
for i in range(10):
    result += i
print(result) # result = 0 + 1 + 2 + .. + 9 = 45
```

```
45
```

[7]:
```python
result = "foo" # now the type of the variable result changes dynamically to str
```

Conversely, C is a compiled, statically typed language.

```c
// file main.c
// C language: sum numbers from 0 to 9
int result=0; // need to declare result as an integer variable
for(int i=0; i<10; i++){
    result += i;
}
printf("%d", result);
//result = 'foo' // this would result in a compile error!
```

In C, we need to first compile main.c, and then execute the generated program.

7 Python for data science

- Python is user-friendly, but it can be slower than a compiled language when used without care
- To get full speed, specialized libraries for the most intensive tasks must be used

In this course, we use Python for data science in conjunction with packages such as: * numpy for number crunching * pandas for handling tabular data * matplotlib and plotly for data visualization * scikit-learn for machine learning * …

Using these packages, we can tackle different data science tasks in Python without compromising performance.

But first, let us review some Python basics!

8 Data types

Python is a *dynamically typed* language. No need to define variable type. It is inferred at run-time.

[8]:
```python
counter = 1 # an integer
val = 3.14 # a float (floating point number)
message = "Hello, world" # a string. Also message = 'Hello, world' works
flag = False # a Boolean variable. Possible values: {True, False}
#val_c = 0.5 + 0.4j # a complex number (real + imaginary part)

type(message)
```

[8]: str

The variable type can change at run-time:

[9]:
```
message = 42 # message is now an int!
type(message)
```

[9]: int

The variable message was first a string, then it was changed to int. This is OK for Python.

9 Basic operations

Basic arithmetic operations `+`, `-`, `/`, `*` behave as expected on numeric types:

[10]:
```
val1 = 3.14
val2 = 42
val3 = val1 + val2
val3
```

[10]: 45.14

[11]:
```
type(val1), type(val2), type(val3)
```

[11]: (float, int, float)

[12]:
```
val2/val1
```

[12]: 13.375796178343949

Basic operations `+` and `*` are also defined on strings:

[13]:
```
message1 = "Hello, "
message2 = "World"
message = message1  + message2 # string "sum"
message
```

[13]: 'Hello, World'

[14]:
```
message1 * 3 # string "multiplication"
```

[14]: 'Hello, Hello, Hello, '

10 Data structures

10.1 Lists

A **list** is an ordered collection of python objects:

[15]:
```
object_lst = [3.14, "pear", 42] # a float, a string, an integer
object_lst
```

[15]:
```
[3.14, 'pear', 42]
```

You can access its elements with square bracket notation and integer indexes 0,1,2...,`len(object_lst)-1`

[16]:
```
object_lst[2] # note: starting index is 0!
```

[16]:
```
42
```

You can modify the elements of a list:

[17]:
```
object_lst[2] = 44
object_lst
```

[17]:
```
[3.14, 'pear', 44]
```

You can append to a list using the append method:

[18]:
```
object_lst.append(-5)
object_lst
```

[18]:
```
[3.14, 'pear', 44, -5]
```

11 Data structures

11.1 Lists

You can access more elements at once and get a sub-list:

[19]:
```
object_lst
```

[19]:
```
[3.14, 'pear', 44, -5]
```

[20]:
```
object_lst[1:3] # returns the elements from index 1 to 3-1
```

[20]:
```
['pear', 44]
```

The ''sum" of two lists is the concatenation of its elements

[21]:
```
list1 = ["A", "B", "C"]
list2 = ["D", "E", "F"]
list1 + list2
```

[21]:
```
['A', 'B', 'C', 'D', 'E', 'F']
```

The ''multiplication" of a list times an integer is also defined:

[22]:
```
list1 * 3
```

[22]: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C']

12 Data structures

12.1 Tuples

A **tuple** is pretty much like a list, but it is immutable:

[23]:
```
object_tup = ("A", "B", False)
object_tup
```

[23]: ('A', 'B', False)

[24]:
```
object_tup[0]
```

[24]: 'A'

[25]:
```
# Don't do this!
#object_tup[0] = True
```

13 Data structures

13.1 Dictionaries

A **dictionary** is a collection of key-value pairs:

[26]:
```
# a dictionary is a collection of key-value pairs
fruit_cost_dict = {"apple": 1.0, "pear": 1.5, "strawberry": 2.0} # fruit cost␣
    ↪(CHF)
fruit_cost_dict
```

[26]: {'apple': 1.0, 'pear': 1.5, 'strawberry': 2.0}

“apple”, “pear”, “strawberry” are the keys, 1.0, 1.5, 2.0 are the values

You can access its elements using *square bracket* notation:

[27]:
```
fruit_cost_dict["strawberry"] # dictionary[key] returns the corresponding value
```

[27]: 2.0

You can modify the value associated to a key:

[28]:
```
fruit_cost_dict["strawberry"] = 2.5
```

You can even add a new key-value pair:

[29]:
```
fruit_cost_dict["mango"] = 5.0
```

14 Control Flow

14.1 If statement

Basic usage of the **if** statement: check a condition. If `True`, execute the code in the following block

[30]:
```
x = 1
if x > 0:
    print("x is greater than 0")
```

```
x is greater than 0
```

The optional **else** block, it is executed when the condition is `False`:

[31]:
```
x = 0
if x > 0:
    print("x is greater than 0")
else:
    print("x is smaller than (or equal to) 0")
```

```
x is smaller than (or equal to) 0
```

multiple **elif** blocks may be used to handle multiple cases:

[32]:
```
x = 0
if x == 0:
    print("x is equal 0")
elif x > 0:
    print("x is greater than 0")
else:
    print("x is smaller than 0")
```

```
x is equal 0
```

15 Control Flow

15.1 While loop

A **while** loop executes a block of code until a certain condition becomes `False` or a **break** statement is encountered:

[33]:
```
result = 0
idx = 0
while idx < 5:
    result = result + idx # or result += idx
    idx = idx + 1
result # = 0 + 1 + 2 + 3 + 4 = 10
```

[33]:
```
10
```

[34]:
```
result = 0
idx = 0
while idx < 5:

    result = result + idx
    idx = idx + 1

    if result > 4:
        break

result # = 0 + 1 + 2 + 3 + 4 = 10
```

[34]: 6

16 Control Flow

16.1 For loop

A **for** loop iterates over a sequence (a collection like a list or a tuple, or an iterator).

[35]:
```
name_lst = ["Marco", "Alice", "Dario", "Anna"]

for name in name_lst:
    print(name)
```

```
Marco
Alice
Dario
Anna
```

The **for** loop is generally more readable than the **while** equivalent:

[36]:
```
idx = 0
while(idx < len(name_lst)):
    print(name_lst[idx])
    idx = idx + 1
```

```
Marco
Alice
Dario
Anna
```

16.2 The range iterator

It is very common to iterate over a numerical range. The `range` iterator comes in handy:

[37]:
```
# Print the square of the numbers from 0 to 4
for idx in range(5): # no need to declare a list [0, 1, 2, 3, 4]
    print(idx, idx**2) # print idx, idx squared
```

```
0 0
1 1
2 4
3 9
4 16
```

The code above is way more practical than explicitly constructing the sequence of values 0, 1, 2, 3, 4

In Python, `range(5)` is an **iterator** that behaves like the list [0, 1, 2, 3, 4]. It is more convenient to use the range iterator instead of explicitly constructing the equivalent list.

[38]:
```
list(range(5)) # now the range is explicitly converted to an equivalent list
```

[38]:
```
[0, 1, 2, 3, 4]
```

We can also specify a starting index:

[39]:
```
list(range(1, 11)) # values from 1 to 10
```

[39]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

17 Control Flow

17.1 List comprehension

A very common task: **filter** the elements of a sequence according to a **logic condition**.

Example: take all the strings starting with A from a list

[40]:
```
name_lst = ["Marco", "Alice", "Dario", "Anna"]
```

[41]:
```
names_A = [] # result list
for name in name_lst:
    if name.startswith("A"):
        names_A.append(name)
names_A
```

[41]:
```
['Alice', 'Anna']
```

List comprehension offers a shorthand alternative:

[42]:
```
names_A = [name for name in name_lst if name.startswith("A")]
names_A
```

[42]:
```
['Alice', 'Anna']
```

18 Control Flow

18.1 List comprehension cont'd

Sometimes we want to **filter and transform** the elements of the sequence.

Example: take the **square** of all the **odd numbers** from 1 to 10

[43]:
```
squared_odds = [] # result_list
for num in range(1, 11):
    if num %2 == 1: # if the remainder of the division by 2 is zero...
        squared_odds.append(num**2)
squared_odds
```

[43]:
```
[1, 9, 25, 49, 81]
```

This can also be done with list comprehension:

[44]:
```
squared_odds = [el**2 for el in range(1, 11) if el%2 == 1]
squared_odds
```

[44]:
```
[1, 9, 25, 49, 81]
```

The general notation of **List comprehension** is thus: `[expr for el in iterable if condition]`.

19 Modules

19.1 The math module

In python, a **module** is just a .py python file (or a folder with more .py files). To use a module, you must first `import` it!

Example: the `math` module, with is part of the Python Standard Library.

[45]:
```
import math # import the math module
```

[46]:
```
math.pi, math.sin(math.pi/2), math.cos(math.pi/2), math.exp(0), math.log(1)
```

[46]:
```
(3.141592653589793, 1.0, 6.123233995736766e-17, 1.0, 0.0)
```

Sometimes we want to import specific objects from a module:

[47]:
```
from math import pi, sin, cos, exp, log
```

[48]:
```
sin(pi/2), cos(pi/2), exp(0), log(1)
```

[48]:
```
(1.0, 6.123233995736766e-17, 1.0, 0.0)
```

20 Modules

20.1 The os module

Another useful module is `os`. It exposes functionalities from the operative system.

[49]:
```
import os
```

[50]:
```
os.listdir() # list of files in a folder (with no arguments, the working folder)
```

[50]:
```
['01_exercises.ipynb',
 '01_slides_jupyter.ipynb',
 '.ipynb_checkpoints',
 'example_notebook.ipynb',
 '01_exercises_solution.ipynb',
 'img']
```

Example: find all the jupyter notebooks (.ipynbs) in a folder:

[51]:
```
# find all the jupyter notebooks (.ipynb) in the current folder
ipynb_lst = []
for file in os.listdir():
    if file.endswith(".ipynb"):
        ipynb_lst.append(file)
ipynb_lst
```

[51]:
```
['01_exercises.ipynb',
 '01_slides_jupyter.ipynb',
 'example_notebook.ipynb',
 '01_exercises_solution.ipynb']
```

[52]:
```
ipynb_lst = [file for file in os.listdir() if file.endswith(".ipynb")] # list␣
↪of all .ipynb files
```

21 Today's challenge

- Set up jupyter with all the required packages (Slide 9, getting started)
  - `pip install` all the required packages
  - Start jupyter
- Familiarize with Jupyter notebook:
  - Access jupyter from your favorite web browser
  - Create a new notebook
  - Create/remove/reorder/execute cells
  - Change cells type from code to markdown
  - Save notebook with name
- Familiarize with Python types and basic control flow
  - Strings, integers, floats
  - List, dictionaries
  - For loop, if statement