Python Libraries for Machine Learning(extras)

Francesco Pugliese, PhD

neural1977@gmail.com

We will discuss following….

- ✓ Flask
- ✓ Beautifulsoup
- ✓ PyMongo
- ✓ Plotly
- ✓ Dash

Flask

- ✓ Flask is a micro-framework.
- ✓ A framework "is a code library that makes a developer's life easier when building reliable, scalable, and maintainable web applications" by providing reusable code or extensions for common operations.
- ✓ It is lightweight and its modular design makes it easily adaptable to developer’s needs.
- ✓ It has a number of out of the box features listed below:
  - ✓ Built-in development server
  - ✓ A fast debugger
  - ✓ Integrated support for unit testing
  - ✓ RESTful request dispatching
  - ✓ Jinja2 templating
  - ✓ Secure cookies support
  - ✓ Unicode-based
  - ✓ WSGI compliance
  - ✓ Ability to plug any ORM
  - ✓ HTTP request handling

Flask vs. Django

vectors by @pch-vector @iconicbestiary

Flask vs Django

- ✓ Both are web frameworks written in Python.
- ✓ Django is a full-stack web framework for Python, whereas Flask is a lightweight and extensible Python web framework.
- ✓ Flask is like snorkelling whereas Django is scuba diving. Flask is light and easy whereas Django is capable of preparing you to enter deeper waters.
- ✓ Django’s tag is, The web framework for perfectionist with a deadline. Because everything is built in, you don’t need to bother yourself with creating the files and thinking about how you should structure your app. It’s all ready for you and you can immediately get started with building your app (Prax).
- ✓ It really depends on what you’re making, but if you’re building a simple app, consider Flask

Flask structure

- ✓ Instead of cramming all your code into one place, Flask helps you organize
  - ✓ Logic
  - ✓ Design
  - ✓ Database into separate files.

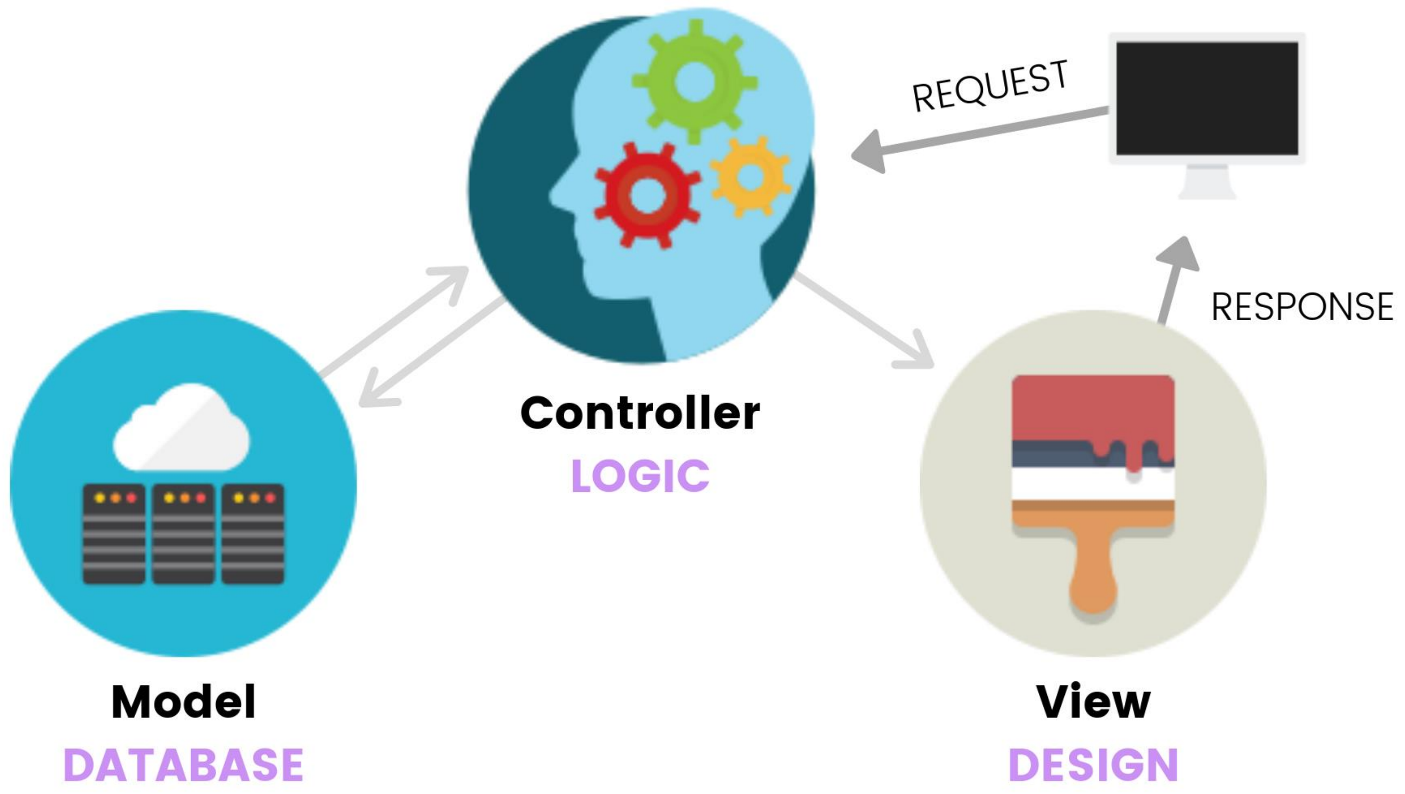
- ✓ **LOGIC**: "main.py" imports the Flask module, creates a web server, creates an instance of the Flask class — your new web application! This is where you write all your "logical" code structuring your web app.

- ✓ **DESIGN**: The Flask Framework looks for HTML files in a folder called templates. Put all your HTML files in there. In static folder, store CSS, JavaScript, images, and other files.

- ✓ **DATABASE**: Flask does not support databases natively, but there are a number of Flask extensions such as **Flask-SQLAlchemy**.

- ✓ Check this colab notebook for a sample application using Flask

Flask structure



from Web Programming with Flask — Intro to Computer Science — Harvard's CS50 (2018)

Web scraping

- ✓ Internet as a data source

- ✓ Web Scraping is the process of downloading data from websites and extracting valuable information from that data. It is a great tool to have in your tool kit because it allows you to get rich varieties of data.

- ✓ Each year, more and more businesses adopt web scraping tools as part of their business intelligence and advertising initiatives.

- ✓ Extract information from websites. This is an ideal scenario when no API or datasets are available for a particular subject or use case under consideration.

- ✓ A popular use of web scraping is to search for online deals like airline tickets, concerts etc. For example, a python script could scrape a website when ticket sales go online, and use a bot to purchase the best tickets. A script would be able to do this much more quickly and efficiently than a human, as it can generate multiple requests per minute.

Web scraping

- ✓ BeautifulSoup is a web scraping library which is best used for small projects. For larger projects libraries like Scrapy and Selenium start to shine

- ✓ Approach for web scrapping **–**
  - ✓ **Step 1**: Find the URL you want to scrape
  - ✓ **Step 2**: Identify the structure of the sites HTML - to quickly identify which elements you need to target.
  - ✓ **Step 3**: Install Beautiful Soup and Requests **-** pip install requests and pip install beautifulsoup4
  - ✓ **Step 4**: Web Scraping Code - refer to next slide
  - ✓ **Step 5**: Isolating the results –

    Example: paragraphs = page_content.find_all("p")[i].text - This basically finds all of the <p> elements in the HTML. the .text allows us to select only the text from inside all the <p> elements

- ✓ **Note**: Sample example for webscrapping using BeautifulSoup4 can be found in this colab notebook

Web scraping using BeautifulSoup

```
from bs4 import BeautifulSoup
import requests
# Here, we're just importing both Beautiful Soup and the Requests
library


page_link = 'the_url_you_want_to_scrape.scrape_it_real_good.com'
# this is the url that we've already determined is safe and legal to
scrape from.


page_response = requests.get(page_link, timeout=5)
# here, we fetch the content from the url, using the requests
library


page_content = BeautifulSoup(page_response.content, "html.parser")
#we use the html parser to parse the url content and store it in a
variable.


textContent = []
for i in range(0, 20):
    paragraphs = page_content.find_all("p")[i].text
    textContent.append(paragraphs)
# In my use case, I want to store the speech data I mentioned
earlier.  so in this example, I loop through the paragraphs, and
push them into an array so that I can manipulate and do fun stuff
with the data.
```

BeautifulSoup methods

```
soup.title
# <title>Returns title tags and the content between the tags</title>


soup.title.string
# u'Returns the content inside a title tag as a string'


soup.p
# <p class="title"><b>This returns everything inside the paragraph
tag</b></p>


soup.p['class']
# u'className' (this returns the class name of the element)


soup.a
# <a class="link" href="http://example.com/example" id="link1">This
would return the first matching anchor tag</a>


// Or, we could use the find all, and return all the matching anchor
tags


soup.find_all('a')
# [<a class="link" href="http://example.com/example1"
id="link1">link2</a>,
#  <a class="link" href="http://example.com/example2"
id="link2">like3</a>,
#  <a class="link" href="http://example.com/example3"
id="link3">Link1</a>]


soup.find(id="link3")
# <a class="link" href="http://example.com/example3" id="link3">This
returns just the matching element by ID</a>
```

Should I scrape the web in the first place?

- ✓ An alternative to web scraping is using an API, if one is available. Obviously, in many cases, this isn't an option, but API's do provide faster and often more reliable data.

- ✓ Here are a few great APIs. Some APIs also provide more content than what would be available through web scraping.

- ✓ **Be Polite**: Web scraping can also overload a server, if you are making a large amount of requests, and scraping large amounts of data. As I mentioned earlier, it's a good idea, before you start, to check the robots.txt before scraping.

- ✓ Another good way to be polite when scraping is to be completely **transparent**, and even notify people to let them know you're going to crawl their site, why you are doing it, and what you are using the data for.

- ✓ One way to do this, and highly recommended, is to use a **user agent**. You can import a user agent library in python by pip installing the user_agent library.

Should I scrape the web in the first place?

- ✓ Your user agent can provide information like a link to more information about the scraper you’re using. Your page about the scraper can and should include the information about what you’re using it for, what IP address you are crawling from, and possibly a way to contact you, if your bot causes any problems.

- ✓ The point is that web scraping can cause problems, and we don’t want to cause problems.

- ✓ The gold rule is to just be open and honest in communicating to webmasters. If you respond to complaints quickly, you should be fine.

Mongo DB

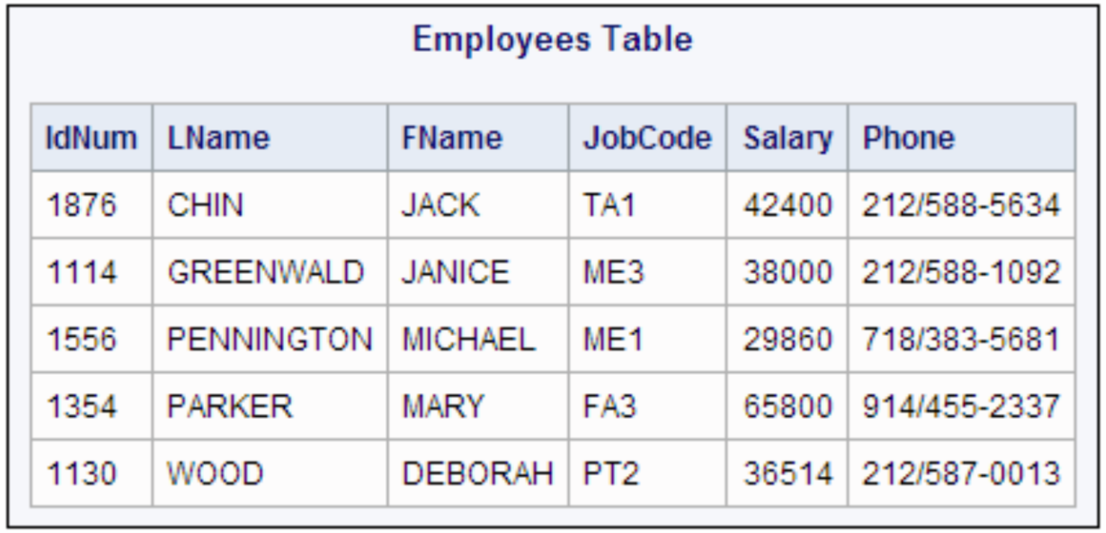
- ✓ MongoDB is a NoSQL database. Unlike traditional relational databases, NoSQL is not composed of tables and rows, but instead it stores data as documents in collections.

- ✓ Flexibility is one of the many benefits of a NoSQL database because it does not require a rigid schema.

- ✓ There are many ways to interact with a MongoDB –
  - ✓ **Mongo shell** – command line interface for MongoDB
  - ✓ **PyMongo** (if you are a Python developer) – More details in the slides ahead
  - ✓ **Mongoose** - Object Data Modeling (ODM) library for **MongoDB** and **Node.js**
  - ✓ **Mongo Atlas** – GUI platform for viewing the collections, data in your database (like PhpMyAdmin if you know some PHP)

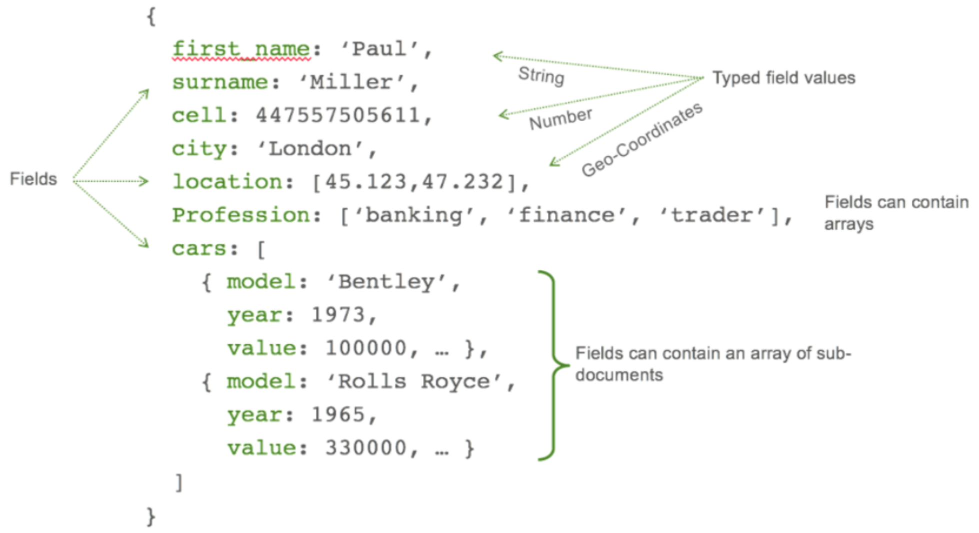
  Methods mostly remain the same for Mongo Shell and PyMongo with slight changes in syntax

MongoDB



Employees Table

IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

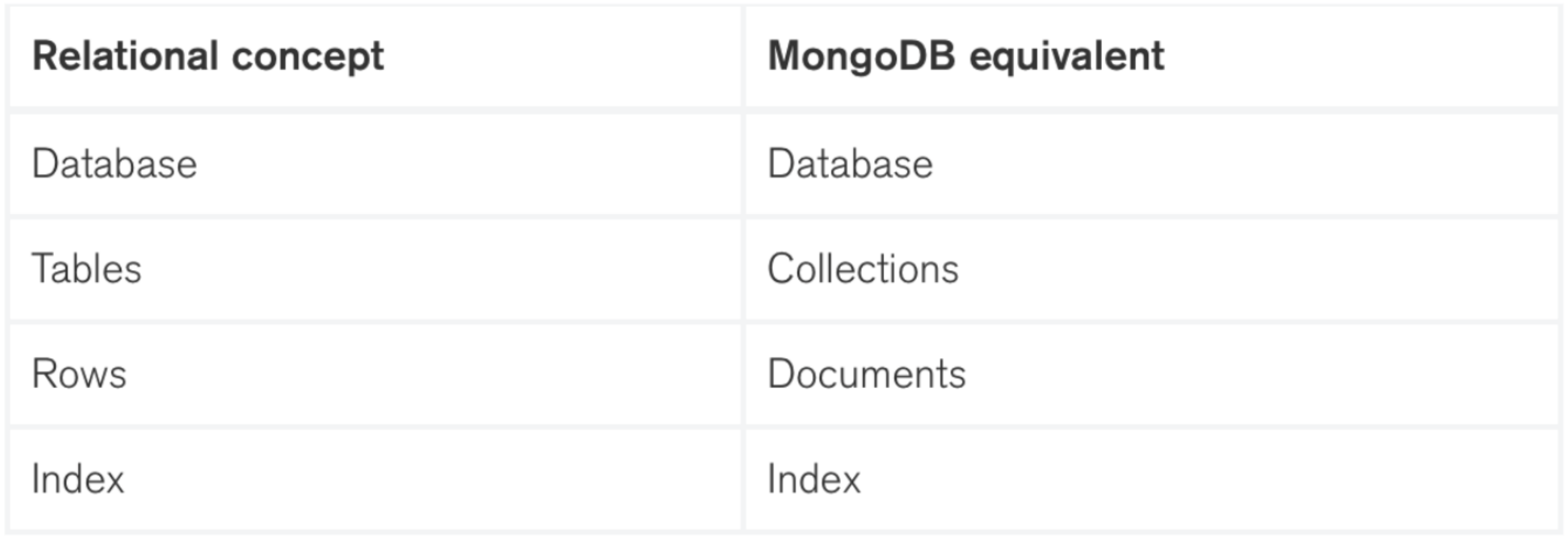


```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields
String
Number
Geo-Coordinates
Typed field values
Fields can contain arrays
Fields can contain an array of sub-documents

SQL Rows vs MongoDB (NoSQL) Document | Sources:
http://support.sas.com/documentation/cdl/en/sqlproc/63043/HTML/default/n1hz0uhw57yye2n16m5r103jjpjj.htm
and https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

MongoDB Terminologies



Relational concept	MongoDB equivalent
Database	Database
Tables	Collections
Rows	Documents
Index	Index

Relational VS MongoDB | Source: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

PyMongo

- ✓ Wrapper library for interacting with MongoDB database
- ✓ MongoDB is a NoSQL database. Unlike traditional relational databases, NoSQL is not composed of tables and rows, but instead it stores data as documents in collections.
- ✓ **Installing PyMongo**: pip install pymongo
- ✓ **Accessing the database:** First of all, you need to import PyMongo into your program. Then, use the MongoClient() object to connect to your database.
  - ✓ **Example**: if you want to connect to the database hosted at a client with IP address '192.168.4.12', to the database named 'example_db', and the collection named 'example_col', we can use the following code:

```
import pymongo
ex_col = pymongo.MongoClient("192.168.4.12").example_db.example_col
```

- ✓ More details about how to use PyMongo in this colab notebook

PyMongo vs MongoShell

For example, findOneAndUpdate in the Mongo Shell would look like:

```
db.collection.findOneAndUpdate(
   { "name" : "A. MacDyver" },
   { $inc : { "points" : 5 } },
   { sort : { "points" : 1 } }
)
```

And the PyMongo find_one_and_update syntax would look like:

```
db.collection.find_one_and_update(
   { "name" : "A. MacDyver" },
   { "$inc" : { "points" : 5 } },
   { "sort" : { "points" : 1 } }
)
```

PyMongo commands

- ✓ **Find** – You can use the command find() to query your database
  - ✓ **Example:** titanic_data.find({"sex": "male"}) – returns all the documents in data with surname "Doe"

- ✓ **Fine_one** - finds the first document that matches the filter criteria
  - ✓ **Example:** titanic_data.find_one({"sex": "male"}) – returns only one document

  Note: The command find() without any filter is equivalent to find all. You can then use .count() to see the total number of documents in the collection. Example: titanic_data.find().count()

- ✓ **Insert**: You can insert documents in to a collection using insert command
  - ✓ **Example**: mp_col = db['mongo_practice']
    mp_col.insert({"first_name": "Fabio", "surname": "Luciani", "age": 32})
    mp_col.find_one({"surname": "Luciani"}) – returns one document

  Note: The field "_id" of the document will be created automatically by MongoDB and is unique across documents.

PyMongo commands

- ✓ **Update:** Updates a document based a filter criteria

  Example: ex_col.update(<filter>, <update>)

  Note:By default, update() method performs update to one document. An alternative to this is the update_one() method. If you want to update many documents that satisfy the filter criteria, set the 'multi' option to true:

- ✓ **Delete:** Finally, the delete_one() and delete_many() commands have the syntax as follows

  Example: ex_col.delete_one(<filter>)

  ex_col.delete_many(<filter>)

  - ✓ The **delete_one()** command deletes a document that satisfies the filter criteria, while the **delete_all()** command deletes all matched documents.

- ✓ More details about how to use PyMongo commands in this colab notebook

Plotly

- ✓ Plotly is both a company and an open-source library
- ✓ Plotly the company focusses on data visualization for business intelligence: such as reporting, dashboards and hosting BI solutions
- ✓ Plotly the open source library is a general data visualization library focussed on interactive visualizations
- ✓ Plotly has libraries for Javascript, React, R and Python
- ✓ The most popular version is the Python library
- ✓ Using Plotly we can create interactive plots as .html files. While users can still interact with these plots (zoom in, select and hover), these plots cannot be connected to changing data sources
- ✓ Documentation: http://www.plot.ly/python

Dash

- ✓ Often users want plots to be able to interact with each other, interact with components, or have plots updated in real time
- ✓ To accomplish this level of tasks we need a dashboard
- ✓ Dash is an open source library from the Plotly company that allows you to create full dashboard with multiple components, interactivity and multiple plots
- ✓ Instead of creating a .html file, Dash will produce a dashboard web application at a local URL (**Example**: 127.0.0.1:8050)
- ✓ You can then visit and interact with this dashboard in the web application
- ✓ Since Dash renders a web application, you can then deploy your dashboards online

Dash

- ✓ Dash apps are composed of two parts –
  - ✓ Layout of the application and it describes what the application looks like
  - ✓ Interactivity of the application

- ✓ Good news is that you don’t need to know any HTML or CSS to use Dash

- ✓ Most HTML tags are given as Python classes

- ✓ **Example**: typing html.H1(children= "Hello Dash”) inside your Dash script results in the HTML element <h1>Hello Dash</h1>

- ✓ Dash offers 2 distinct component libraries –
  - ✓ **dash_html_components** library – which has a component for every HTML tag, like the H1 tag
  - ✓ **dash_core_components** – offers high-level interactive components that are generated with Javascript, HTML and CSS through the React.js library
    - ✓ A reference to all available dash core components can be found at –
      - ✓ https://dash.plot.ly/dash-core-components

Dash

- ✓ Dash allows you to leverage previous knowledge of HTML and CSS to create very customized dashboards
  - ✓ Pick a relevant html component
  - ✓ Insert parameters in to html component
  - ✓ Adjust CSS style dictionary that works for entire application

- ✓ Technically no knowledge of HTML or CSS is needed to create a Dash dashboard

- ✓ But to stylize and customize Dash dashboards, knowledge of HTML and CSS will help you a lot

- ✓ A HTML Div element is a division i.e.., a section or block of the web app(dashboard)

- ✓ CSS allows for styling HTML elements
  - ✓ Fonts, colours, Borders etc..,
  - ✓ Dash uses dictionaries to pass in CSS style calls

Dash callbacks

- ✓ Callbacks are very useful to add interactivity or events to our Dash applications so that when user performs an actions such as button click or text update etc.., respective dash components will re-render based on the action.

- ✓ Steps to create a callback for interactions –

    - ✓ Create a function to return some desired output

    - ✓ Decorate that function with an @app.callback decorator
        - ✓ Set an output to a component id
        - ✓ Set an input to a component id

    - ✓ Connect the desired properties

References:

- ✓ https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/ - **Flask**

- ✓ https://blog.datawow.io/introduction-to-mongodb-pymongo-e1948ec6b819 – **PyMongo**

- ✓ https://codeburst.io/web-scraping-101-with-python-beautiful-soup-bb617be1f486 **- BeautifulSoup**

- ✓ https://www.udemy.com/course/interactive-python-dashboards-with-plotly-and-dash/ - **Plotly and Dash**

Francesco **Pugliese**