Applied Data Science with Python

Module 9 - Data

Overview

Data

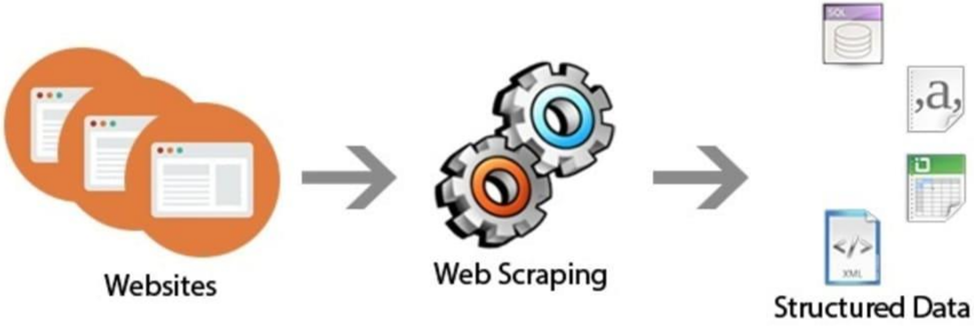
Web Scraping

Databases

Feature Engineering

Data

- Data is an essential component for **Data Scientist**.
- How we obtain this data and interpret this data is essential for getting quality results.
- Data is generally in different mediums for example:
  - Websites and database, as examples
- and parsed in differing formats.
  - JSON, XML, CSV, RAW Unformatted



THE EXPERIENCE YOU NEED
& THE SUPPORT TO SUCCEED

Data Mediums

With the introduction of the Internet a wealth of data can be obtained from websites, through a processes call **Web Scraping**.

Web Scraping is the process of taking the the unstructured **HTML** data and converting this into a structured format.

There are websites like Facebook, Twitter, Google,etc that provide **APIs** to their data which are parsed in a more usable state such as **JSON, XML, CSV.**

Python Libraries for Web Scraping

Some of the libraries that we can use for Web Scraping include:

Urllib2: It is a Python module which can be used for fetching URLs. It defines functions and classes to help in opening URLs (mostly HTTP) in a complex world. Also urllib and urllib3 are also availble

Requests: 'Requests' is a simple, easy-to-use HTTP library written in Python, easy to use and the one we will use.

BeautifulSoup: It is an excellent tool for pulling out information from a webpage. You can use it to parse the site's HTML and XML to extract meaningful data. *Please refer to this site for more:*

information: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Knowledge of HTML

Know what you're working:

A basic URL:

https://www.monster.com/jobs/search/?q=Software-Developer&where=Australia

You can deconstruct the above URL into two main parts:

- The base URL represents the path to the search functionality of the website. In the example above, the base URL is https://www.monster.com/jobs/search/.
- The query parameters represent additional values that can be declared on the page. In the example above, the query parameters are **?q=Software-Developer&where=Australia**.

Decipher Information in URL

Any job you'll search for on this website will use the same **base URL**. However, the query parameters will change depending on what you're looking for. You can think of them as query strings that get sent to the database to retrieve specific records.

Query parameters generally consist of three things:

- **Start**: The beginning of the query parameters is denoted by a question mark (**?**).
- **Information**: The pieces of information constituting one query parameter are encoded in **key-value pairs**, where related keys and values are joined together by an equals sign (**key=value**).
- **Separator**: Every URL can have multiple query parameters, which are separated from each other by an **ampersand (&)**.

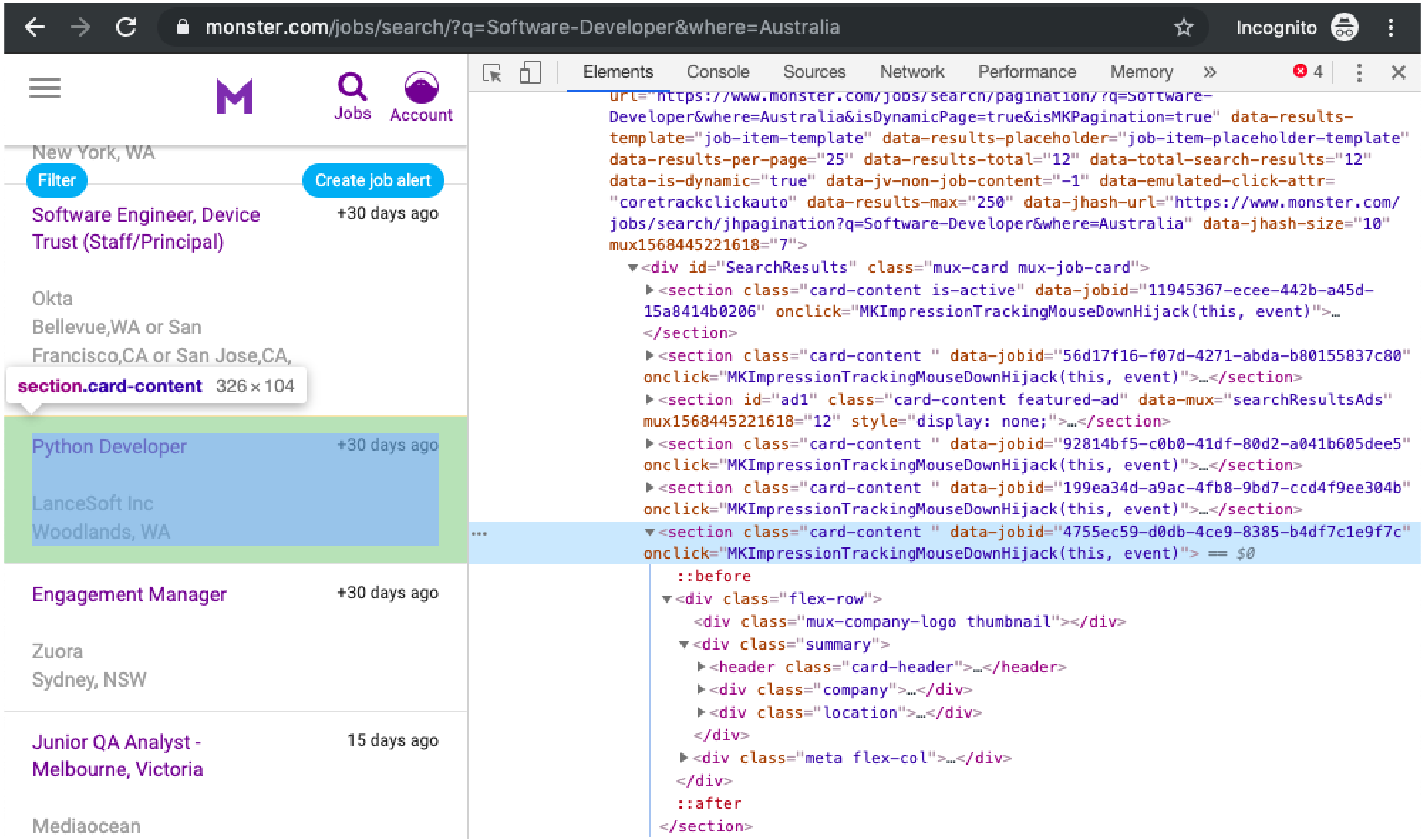
Equipped with this information, you can pick apart the URL's query parameters into two key-value pairs:

- **q=Software-Developer** selects the type of job you're looking for.
- **where=Australia** selects the **location,** you're looking for.

Inspect the Site Using Developer Tools

- Next, you'll want to learn more about how the data is structured for display. You'll need to understand the page structure to pick what you want from the HTML response that you'll collect in one of the upcoming steps.
- Developer tools can help you understand the structure of a website. All modern browsers come with developer tools installed. In this lecture, you'll see how to work with the developer tools in Chrome. The process will be very similar to other modern browsers.
- In Chrome, you can open up the developer tools through the menu ***View*** **→** ***Developer*** **→** ***Developer Tools***. You can also access them by right-clicking on the page and selecting the *Inspect* option, or by using a keyboard shortcut.
- Developer tools allow you to interactively explore the site's DOM(Document Object Model) to better understand the source that you're working with. To dig into your page's DOM, select the ***Elements*** **tab** in developer tools. You'll see a structure with clickable HTML elements. You can expand, collapse, and even edit elements right in your browser:

Inspect the Site Using Developer Tools



monster.com/jobs/search/?q=Software-Developer&where=Australia
Incognito
Jobs
Account
Filter
Create job alert
section.card-content 326 × 104
Elements
Console
Sources
Network
Performance
Memory
»
4

New York, WA

Software Engineer, Device Trust (Staff/Principal) +30 days ago

Okta
Bellevue,WA or San Francisco,CA or San Jose,CA,

Python Developer +30 days ago

LanceSoft Inc
Woodlands, WA

Engagement Manager +30 days ago

Zuora
Sydney, NSW

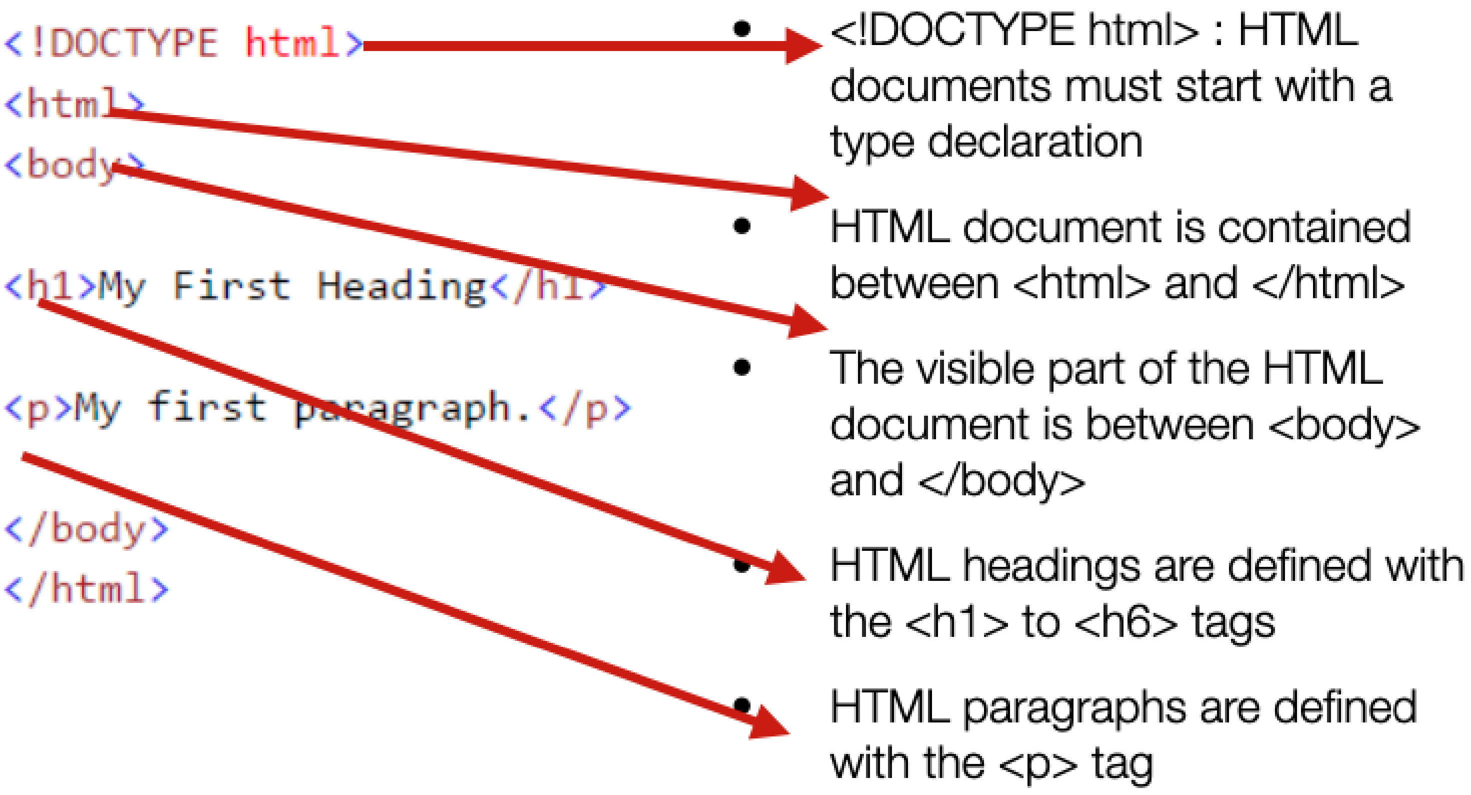
Junior QA Analyst - Melbourne, Victoria 15 days ago

Mediaocean

```
url="https://www.monster.com/jobs/search/pagination/?q=Software-
Developer&where=Australia&isDynamicPage=true&isMKPagination=true" data-results-
template="job-item-template" data-results-placeholder="job-item-placeholder-template"
data-results-per-page="25" data-results-total="12" data-total-search-results="12"
data-is-dynamic="true" data-jv-non-job-content="-1" data-emulated-click-attr=
"coretrackclickauto" data-results-max="250" data-jhash-url="https://www.monster.com/
jobs/search/jhpagination?q=Software-Developer&where=Australia" data-jhash-size="10"
mux1568445221618="7">
  ▼<div id="SearchResults" class="mux-card mux-job-card">
    ▶<section class="card-content is-active" data-jobid="11945367-ecee-442b-a45d-
    15a8414b0206" onclick="MKImpressionTrackingMouseDownHijack(this, event)">…
    </section>
    ▶<section class="card-content " data-jobid="56d17f16-f07d-4271-abda-b80155837c80"
    onclick="MKImpressionTrackingMouseDownHijack(this, event)">…</section>
    ▶<section id="ad1" class="card-content featured-ad" data-mux="searchResultsAds"
    mux1568445221618="12" style="display: none;">…</section>
    ▶<section class="card-content " data-jobid="92814bf5-c0b0-41df-80d2-a041b605dee5"
    onclick="MKImpressionTrackingMouseDownHijack(this, event)">…</section>
    ▶<section class="card-content " data-jobid="199ea34d-a9ac-4fb8-9bd7-ccd4f9ee304b"
    onclick="MKImpressionTrackingMouseDownHijack(this, event)">…</section>
…   ▼<section class="card-content " data-jobid="4755ec59-d0db-4ce9-8385-b4df7c1e9f7c"
    onclick="MKImpressionTrackingMouseDownHijack(this, event)"> == $0
        ::before
      ▼<div class="flex-row">
          <div class="mux-company-logo thumbnail"></div>
        ▼<div class="summary">
          ▶<header class="card-header">…</header>
          ▶<div class="company">…</div>
          ▶<div class="location">…</div>
          </div>
        ▶<div class="meta flex-col">…</div>
        </div>
        ::after
    </section>
```

THE EXPERIENCE YOU NEED
& THE SUPPORT TO SUCCEED

Basic HTML Site



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

- <!DOCTYPE html> : HTML documents must start with a type declaration
- HTML document is contained between <html> and </html>
- The visible part of the HTML document is between <body> and </body>
- HTML headings are defined with the <h1> to <h6> tags
- HTML paragraphs are defined with the <p> tag

HTML tags

- HTML links are defined with the <a> tag

- HTML tables are defined with<Table>, row as <tr> and rows are divided into data as <td>

- HTML list starts with <ul> (unordered) and <ol> (ordered). Each item of list starts with <li>

- More Info on HTML : http://www.w3schools.com/html/

Accessing HTML Content From a Page

This task requires a pip installation of the **requests library**, this can be completed from the command promp from Windows or from a terminsal window in Linux:

```
pip3 install requests
```

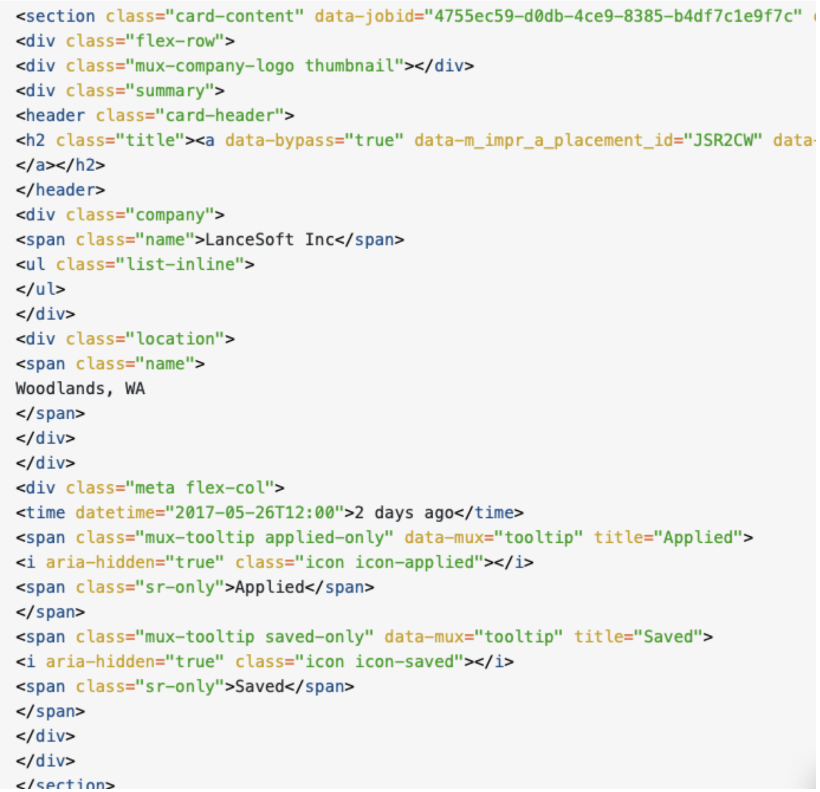
And from a python IDE type the following:

```
import requests
URL = 'https://www.monster.com/jobs/search/?q=Software-
     Developer&where=Australia'
page = requests.get(URL)
```

This code performs an HTTP request to the given URL. It retrieves the HTML data that the server sends back and stores that data in a Python object.

Static Websites

- The website we are scraping in this tutorial serves static HTML content. In this scenario, the server that hosts the site sends back HTML documents that already contain all the data you'll get to see as a user.
- When you inspect the page with developer tools, you may have discovered that a job posting consists of the following long and messy-looking HTML:



```
<section class="card-content" data-jobid="4755ec59-d0db-4ce9-8385-b4df7c1e9f7c"
<div class="flex-row">
<div class="mux-company-logo thumbnail"></div>
<div class="summary">
<header class="card-header">
<h2 class="title"><a data-bypass="true" data-m_impr_a_placement_id="JSR2CW" data-
</a></h2>
</header>
<div class="company">
<span class="name">LanceSoft Inc</span>
<ul class="list-inline">
</ul>
</div>
<div class="location">
<span class="name">
Woodlands, WA
</span>
</div>
</div>
<div class="meta flex-col">
<time datetime="2017-05-26T12:00">2 days ago</time>
<span class="mux-tooltip applied-only" data-mux="tooltip" title="Applied">
<i aria-hidden="true" class="icon icon-applied"></i>
<span class="sr-only">Applied</span>
</span>
<span class="mux-tooltip saved-only" data-mux="tooltip" title="Saved">
<i aria-hidden="true" class="icon icon-saved"></i>
<span class="sr-only">Saved</span>
</span>
</div>
</div>
</section>
```

THE EXPERIENCE YOU NEED
& THE SUPPORT TO SUCCEED

EIT

Lets Scrap a Website

The Requests library should be installed, we will also need to install the beautifulsoup library with the following command from either a terminal or command prompt, depending on operating system:

```
pip install beautifulsoup4
```

Next let's import out libraries:

```
import requests
from bs4 import BeautifulSoup
```

```
University_url="https://www.university-list.net/New-Zealand/universities-1000.htm"
```

Lets Scrap a Website

Next we will get the HTML code by quering the website:

```
page = requests.get(university_url)
print(page)
```

This should give us a status of 200. Next let's display our data.

```
print(page.text)
```

This may not be in the most readable state so we will parse this through BeautifulSoup().

```
soup = BeautifulSoup(page.text, 'html.parser')
print(type(soup)) # outputs the object type from Beautiful Soup
print(soup.pretty()) # pretty's the data into a structured readable format.
```

THE EXPERIENCE YOU NEED
& THE SUPPORT TO SUCCEED

Lets Scrap a Website continued

Next we will look for all **'a' tags** with attributes of **{'rel':'nofollow'}** and then display this data:

```
all_links = soup.findAll('a', attrs={'rel':'nofollow'})
all_links
```

Lastly, we will display only the URL of the University and the University Name:

```
for eachuniversity in all_links:
    print(eachuniversity['href']+","+eachuniversity.string)
```

Lets Scrap another Website

The next site we will scrap will be a wikipidia page, the URL is:

```
https://en.wikipedia.org/wiki/States_of_Germany
```

I will start you off the the two first parts to this activity and you will finish the other two tasks.

```
import requests
from bs4 import BeautifulSoup
import time

wiki = "https://en.wikipedia.org/wiki/States_of_Germany"

page = requests.get(wiki)
soup = BeautifulSoup(page.text,'html.parser')
print(soup.title.string)
```

Lets Scrap another Website continued

The next step will be to findall() **'a'** tags. We will then print out all links with ['href'].

```
all_links = soup.findAll('a', href=True)

# print all links from the webpage

for link in all_links:
    print(link['href'])
```

Lets Scrap another Website continued

Next we will get all the German States and display them as an output, by adding the following lines of code:

```
right_table  = soup.find('table', class_="sortable
wikitable").findAll('tr')


#find all tags with td
states=[]
for row in right_table:
      state_data_entry=row.findAll('td')
      If (len(state_data_entry)!=0):
            states.append(state_data_entry[2].find(text=True))
print(states)
```

Lets Scrap another Website continued

We will finish the next last task through our Jupyter Notbook. This involves creating a dataframe for the following columns.

```
‘STATE','SINCE', 'CAPITAL','LEGISLATION', 'HEAD_OF_GOVERNMENT','GOVERNMENT_COALITION',‘BUNDES_RAT_VOTES','AREA_KM_SQAURE','POPULATION','POP_PER_KM_SQ','CAPITAL','HUMAN_DEVELOPMENT_INDEX','GDP_PER_CAPITA'
```

A bit of House Keeping

- You should check a site's terms and conditions before you scrape them. It's their data and they likely have some rules to govern it.
- Be nice - A computer will send web requests much quicker than a user can. Make sure you space out your requests a bit so that you don't hammer the site's server.
- Scrapers break - Sites change their layout all the time. If that happens, be prepared to rewrite your code.
- Web pages are inconsistent - There's sometimes some manual clean up that has to happen even after you've gotten your data.

JSON

- JSON JavaScript Object Notation is a popular way to format data as a single human-readable string.
- Most of the APIs send response in JSON format.
- Example:

```
{"name": "Zophie", "isCat": true, "miceCaught": 0, "napsTaken": 37.5, "felineIQ": null}
```

JSON module

- The ***JSON library*** can parse JSON from strings or files.
- The library parses JSON into a Python dictionary or list.
- It can also convert Python dictionaries or lists into JSON strings.

Reading JSON

```
import json
json_data= {"name": "Zophie", "isCat": true,
    "miceCaught": 0, "napsTaken": 37.5,
    "felineIQ": null}
print(json.loads(json_data))
```

This will return that data as a Python dictionary

Writing JSON

```
import json
pythonValue = {'isCat': True, 'miceCaught': 0, 'name':
     'Zophie','felineIQ': None}
json_data = json.dumps(pythonValue)
json_data


Output:
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

XML

- Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- To use xml in our code we need to import the **xml.etree.ElementTree** library
- Example xml format:

```
<employees>
  <employee>
      <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
      <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
      <firstName>Peter</firstName> <lastName>Jones</lastName>
   </employee>
</employees>
```

Reading XML

```
import xml.etree.ElementTree as ET
tree =ET.parse("data/test.xml")
root = tree.getroot()
print(root.tag)

Output:
company
```

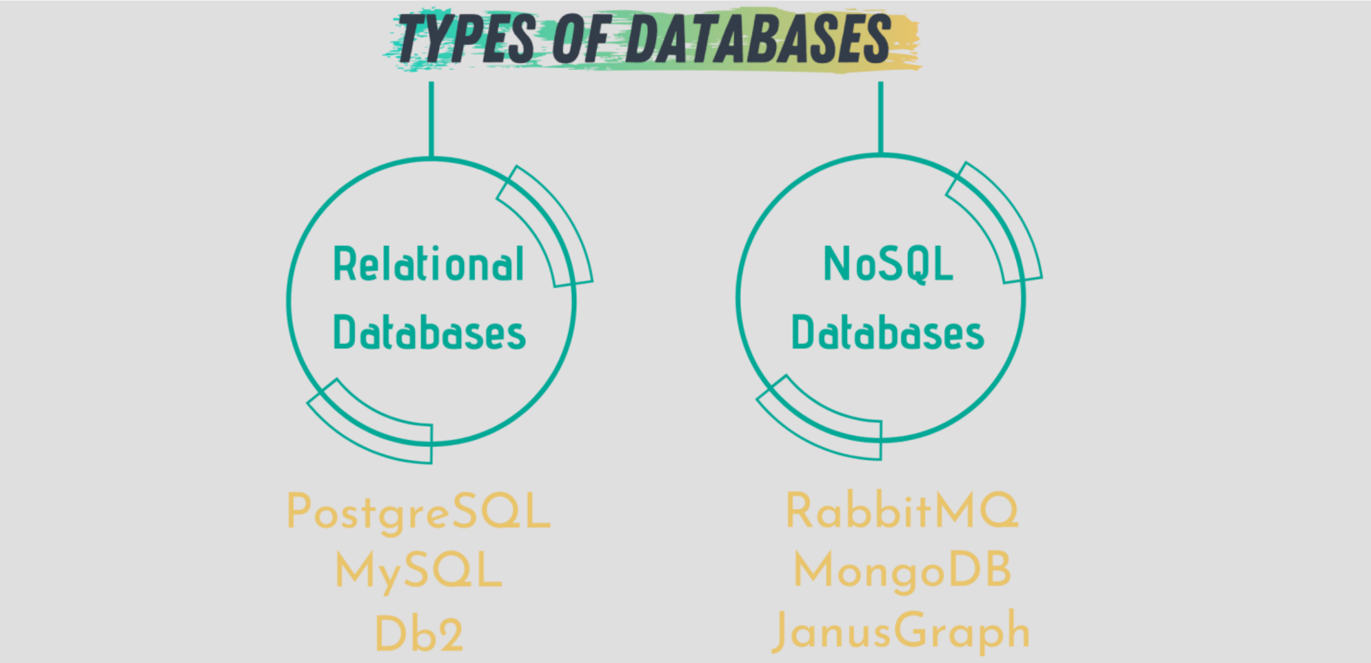
Why use a database

- In Data Science we work with **data**, lots of **data.**
- This data needs to be stored and structured somewhere that is easy to access, provide fast communication, and is secure.
- **Databases** make structured storage secure, efficient, and fast.

Databases

- A **database** is defined as a structured set of **data** held in a computer's memory or on the cloud that is accessible in various ways.
- In Data Science we need to **design**, **create, and interact** with databases when working on projects
- Sometimes we need to create everything from scratch and somtimes we just need to know how to communicate with an already existing database.

Types of databases



TYPES OF DATABASES

Relational Databases

PostgreSQL
MySQL
Db2

NoSQL Databases

RabbitMQ
MongoDB
JanusGraph

Relational Databases

- Data is organized and stored into tables that can be linked to each other, using some relation.
- As an example, an airline company can have a table of passengers for all flights, and another for passengers on a specific flight. A flight code can connect these two tables.
- Understanding the relationship can give us hints and insight that will make the process of analyzing and visualizing the data an easier task.
- The way to communicate and interact with relational databases is through using the SQL language.

Non-Relational Databases(NoSQL)

- These databases are those that connect the information stored in them by categories rather than relations.
- Types of NoSQL
  - Key-Value
    - Key–value (KV) stores use the associative array (also called a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key–value pairs, such that each possible key appears at most once in the collection
  - Document
    - The central concept of a document store is that of a "document". While the details of this definition differ among document-oriented databases, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Example encodings include XML, YAML, JSON and BSON
  - Graph
    - Graph databases are designed for data whose relations are well represented as a graph consisting of elements connected by a finite number of relations. Examples of data include social relations, public transport links, road maps, network topologies, etc
- NoSQL databases are increasingly used in big data and real-time web applications.

SQL vs NoSQL

- Determining what to use on a project, can come down to the CAP theorem.
- The CAP theorem describes the relationship between three aspects of your database: availability, consistency, and partition tolerance.
  - **Consistency:** This means that every inquiry to the database should return the most recent value.
  - **Availability:** This means anyone can make a request for data and get a response, even if one or more items of the database are down.
  - **Partition tolerance:** A *partition* is a communications break within a system. Tolerance means the database should function adequately even if the communications between aspects of it are broken.
- To select a database type, you need to prioritize two of the three aspects of the CAP theorem.
- If you care more about consistency and availabilty, then you should choose a r**elational database**.
- However, if you care more about availability and partition tolerance, or consistency and partition tolerance, then a **NoSQL database** will work better for your project.

NoSQLProducts to try

- MonoDB
  - Is a document-based database.
  - https://www.mongodb.com
  - https://university.mongodb.com/courses/M220P/about
- Neo4J
  - Is a graph-based database
  - https://neo4j.com/developer/graph-visualization/
  - https://neo4j.com/developer/graph-data-science/
- PickleDB
  - Is a key-value database
  - https://pythonhosted.org/pickleDB/
- Redis
  - Is a key-value database
  - https://redis.io/