Data Analysis & Programming for Operations Management (DAPOM)

Wout van Wezel (Coordinator)

w.m.c.van.wezel@rug.nl

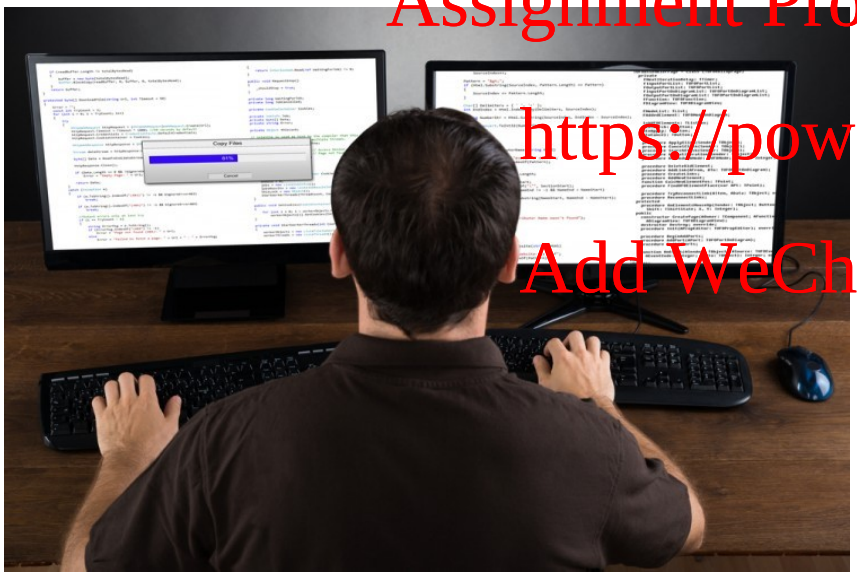
050-3637181

DUI-621

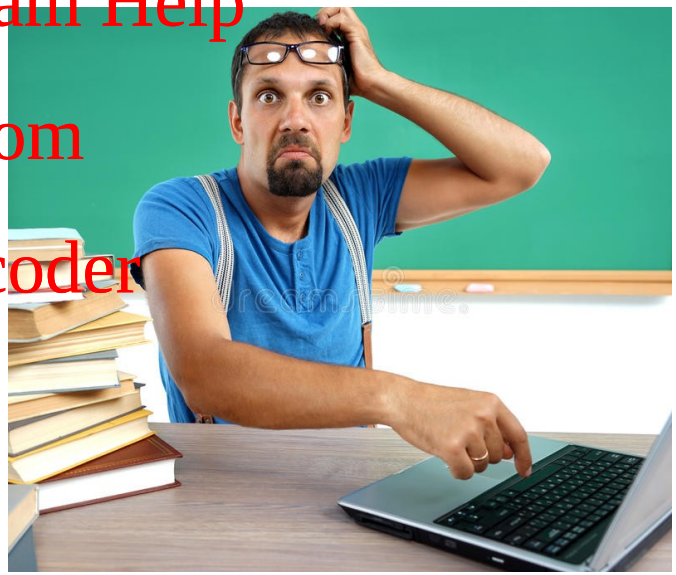
Databases

- No Data Science without Big Data!
- Topic today:
  - Databases
  - Big Data
  - Json
  - Elasticsearch

But first, practicals



VS



Programming

- Programming is more a mindset than a skill.
- Essentially, programming is really easy. We have:
  - Variables

    ```
    a=3
    b=10
    ```
  - Branching based on conditions

    ```
    if a > 4: a=a+2
    else: a=a+3
    ```
  - Iteration (for, while)

    ```
    while a<b do: a=a*2
    ```
  - Encapsulation (procedures, objects, libraries): combine multiple programming commands to logical units that you can reuse.

Programming

- If you don't understand something, take your time to read it


```
A=[[1,2,3],[7,8,9],[10,11,12]]
B=[1,2,3,4,5,6,7]
B[A[0][1]]
```


- Dissect this. Always start at the deepest level:
  - `A` is a list with lists of numbers
  - `B` is a list of numbers
  - `A[0]` is the list `[1,2,3]`
  - So `A[0][1]` is `[1,2,3][1]` is `2`
  - And `B[2]` is 3.

Practicals

- Practicals will get more complex. Proposed approximate time division:
  - Before the B-practical: understand the A-Practical, install the required components, and already start with the assignment for the B-practical (4 a 5 hours)
  - At the B-practical: confront us with the problems you encountered, and what you did to solve it. What did you google? Did you look at the documents of the libraries? Did you check Stackoverflow? What of the offered solutions did not work?
  - After the B-practical: finish the assignments if you hadn't yet (3 a 4 hours)

Question

- If you are on Marktplaats, what kind of response time do you expect when searching?
- Especially look at the 'facets'; counters how many hits exists with your query for each subcategory.
- How can we make such a website?

Databases

- This week, you will work with a file of all restaurants in Groningen.
- You read the file, and select a few restaurants based on some keyword in the name: it must contain 'pizz', because you only want to look at pizzerias. Works fine, and very fast.
- But what if you need to filter all restaurants in The Netherlands. Approx 15.000. Still fine, feels instantaneous.
- What if you need to select all people with the name 'rutte' from all 17.000.000 Dutch people. Any guess how long this takes in Python?

Databases

- Let's try it; file with 15.000.000 Wikipedia titles

- Find titles with 'star' in them
  - Python: 190 seconds
  - Microsoft Access (Database): 66 seconds
  - Elasticsearch: 1.6 seconds

- Titles that start with 'star' (typeahead functionality on websites):
  - Python: 21 seconds
  - Access: 14 seconds
  - Wikipedia (online): 0.5 seconds
  - Elasticsearch: 0.005 seconds

Advantages of a database

- Essentially, all systems you interact with use a database
- Much faster than text files
- Entries (records) can be updated/deleted without reading and writing the complete file
- Can work in a client/server model; thousands of simultaneous users updating and querying
- Which database can we use for Data Science?

'Traditional' databases

- In relational databases, you determine the structure of your data beforehand
- Databases are tables, and each table consists of columns (fields) and rows (records)
- Most databases talk to the outside world with Sql; a simple but powerful standardized language to query and update databases
- For example: `select name from Persons where age<25` will give me a new table with the corresponding rows.
- Most important trick to speed up queries: sorting (called an index)

Persons		
p_number	**name**	**age**
1	John	16
2	William	28
3	Mary	24
…	…	…

Example algorithm: Binary Search (like a phonebook)

- Step 1: compare the median (the number at 50% of the list) with the number you are searching.
- If found: finished. If it is higher, go to step 1 with the upper half. If it is lower, go to step 1 with the lower half.
- Search for 93: 44<93, so look at median of 50..99. 90<93, so look at median of 93..99. 98>93 so look at 93.

Unsorted
50
11
14
90
67
44
98
20
93
99
17

sorted
11
14
17
20
44
50
67
90
93
98
99

n	searches
100	7
1,000	10
10,000	13
100,000	17
1,000,000	20
1,000,000,000	30
10,000,000,000	33

'Traditional' databases

- Let's look again: `select name from Persons where age<25` will give me a new table with the corresponding rows
- If I store the age unsorted, I have to check all rows individually.
- If the age is stored sorted, I must search for the first element first element >25, and then all record below that are in my set

Unsorted
50
11
14
90
67
44
98
20
93
99
17

Sorted
11
14
17
20
44
50
67
90
93
98
99

n	searches
100	7
1,000	10
10,000	13
100,000	17
1,000,000	20
1,000,000,000	30
10,000,000,000	33

'Traditional' databases

- Impressive performance, but in this course, we are more interested in Data Science, which involves:
  - Making selections on really big data
  - Making aggregations and calculations
- This should be fast. A single user can wait for a few seconds, but:
  - If you have many users in a client/server setting, requests are queued and waiting time becomes very long
  - If you want to train a neural network or run an optimization model, you may need to do thousands or millions of queries in a short time

‘column oriented’ databases

- Relational databases store data as rows in memory:

Persons		
p_number	**name**	**age**
1	John	16
2	William	28
3	Mary	24
…	…	…

1	John	16	2	William	28	3	Mary	24

- Column oriented databases store the columns.

1	2	3	John	William	Mary	16	28	24

- This has several advantages, which is why relational databases are not often used for Data Science.

‘column oriented’ databases

1	2	3	John	William	Mary	16	28	24

- Many libraries exists for calculations, statistics, and machine learning (e.g., numpy, scikit). These are highly optimized, and you can simply feed them an array of numbers.
- In a column oriented database, the data is already stored as an array of numbers.
- So, first advantage of column oriented database: to calculate the average of all ages, I only need to process (in this case) 3 bytes, instead of 21 bytes.

'column oriented' databases

- Second advantage:

  - Your computer has a processor (e.g., Intel I7) which does all the calculations.

  - Your computer has memory (e.g., 8GB) where it stores all the data.

  - To do a calculation, the processor needs to get data from the main memory, do the calculation, and store the data back in main memory.
  - A processor itself has some memory as well. Give it a small block of data (for example: 32768 bytes) and do all calculations on that on one go, before writing it back.
  - This processor memory is easily 100 times faster than the normal computer memory.

‘column oriented’ databases

- This is where the second advantage comes from.
- By storing numbers contiguously in main memory, larger blocks are copied in the processor memory, and less transferring from processor memory to main memory is needed.

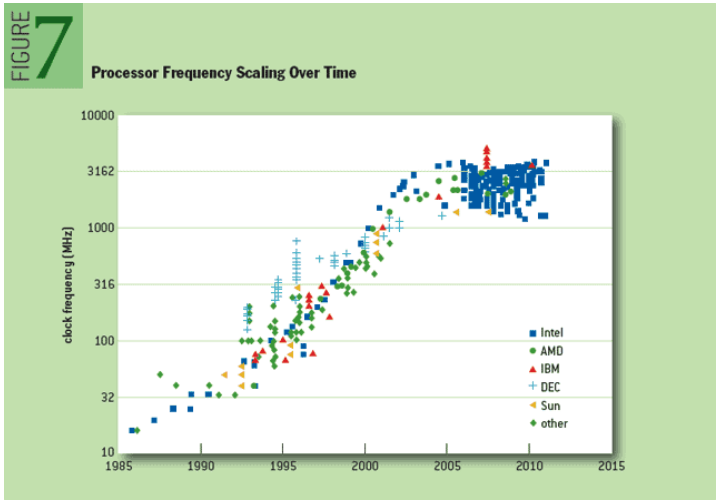
1	John	16	2	William	28	3	Mary	24

versus

1	2	3	John	William	Mary	16	28	24

'column oriented' databases

- For a long time, the speed of the CPU (e.g., 2.8Ghz) increased every year.
- A computationally intensive program doubled in speed each few years without having to change the program!
- However, both putting more components on a chip and increasing the speed increases heat. Now, number of instructions per second is not really increasing anymore.

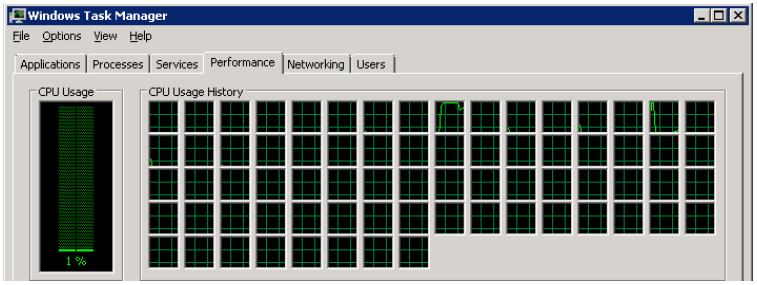


‘column oriented’ databases

- Instead, to improve performance, manufacturers put more processors in parallel (dual core, quad core, etc.)
- A third advantage of column oriented databases can split a column in multiple smaller columns, and give data to each core individually

1	2	3	John	William	Mary	16	28	24

- These are three reasons why (in this query), Access (14 seconds) is 2800 times slower than Elasticsearch (5ms)



Big Data

- What is big data?

- Various definitions, but a common ground is:
  - High volume (too big to fit on one computer)
  - High velocity
  - High variety

- Traditional databases have trouble distributing data over multiple computers. Therefore, sometimes big data is defined as data that does not fit on one computer.

- Various column oriented databases are developed from the start with this ability. If you can spread the load over multiple cores in one processor, you can also spread it over multiple computers.

Big Data

- So, column oriented databases are more suitable for big data, because they:

  - Are very fast
  - Are equipped to do numerical analysis, statistics, and machine learning
  - Can scale very well over powerful processors with many cores, and over multiple computers in a network (sometimes even hundreds or thousands of computers)

Big Data

- Philosophical question: what is big data?

- Too big for Excel? (1 million rows)

- Too big for one computer? What does too big mean?
  - It does not fit in memory (typically 8 to 256 GB)

  - It does not fit on harddrive (typically 1 to 32 TB)
- Calculate the approximate data size you need
  - Example: each second measure machine temperature to predict breakdown
  - 31.536.000 data points per year per machine
  - 100 machines, is 3 billion points, is approx. 3GB

Big Data

- Typically, in big data we measure many dimensions (temperature, vibrations, power usage, humidity, machine efficiency, etc.)
- We then get many data points for many features, and we can:
  - Aggregate (average power usage per day, per machine, or per department, or per factory, etc.)
  - Correlate (relation between temperature and machine efficiency)
  - Time series analysis (power usage at time *t* versus breakdown at time *t-1*)
- This is where column oriented databases are indispensable

Elasticsearch

- Lucene is a well known full text search engine (started in 1999)
- Created as an alternative for commercial databases that were lousy at full text search (Microsoft Sql server, Oracle)
- Elasticsearch was created as a program that uses the Lucene core, but adds, for example:
  - Communication with databases
  - Big Data (parallelize the database over multiple computers)
  - Aggregations (calculations, for example the number of taxi trips per weekday and the average price)
  - Machine learning (relation between weather (rain, wind, snow, etc.), day, time, and number of taxi trips) (Paid version only)

Elasticsearch

- Elasticsearch is a server. It starts and waits until it gets commands from another program.
- There are Python libraries that you can use, which makes it easy to:
  - Create a database (=index)
  - Insert data into the index
  - Query data from the index
  - Delete data
- Elastic (like many other systems) talks Json.

What is a server?

- A server often refers to a computer that runs server programs.
- Standardization of communication with server programs allows that developers can focus:
  - One developer creates a database server
  - Another developer creates a desktop program with which the database can be filled.
  - Yet another developer creates a website that can show the data.

- There are two kinds of standards:
  - Communication
  - Representation of the data

Server communication

- Elasticsearch uses Http for communication. It is the same communication protocol used by web browsers to communicate with the web server. Hence, you can talk to Elasticsearch with your web browser.
- http://localhost:9200/wikititles/_search?q=Title:star
- Elasticsearch uses Json to represent data.

```
[
  {
    "PersonId": 1,
    "Name": "Wout",
    "Age": 49
  },
  {
    "PersonId": 2,
    "Name": "Anna",
    "Age": 16
  }
]
```

- Json (Javascript Object Notation)
- Advantages
  - You can have a hierarchy
  - Your records don't all have to be the same

- Json Format, in itself really simple:
  - "key":"value"
  - [] to denote an array of values of the same field
  - {} to denote an object. Can be nested.


```
{
    "name": "wout",
    "age": 49,
    "data": {
        "cars": ["Volvo",
        "Porsche",
        "Lexus"],
        "hobbies": ["programming",
        "18thcenturypoetry"]
    }
}
```

Elasticsearch

- Elastic 'talks' Json. You specify commands in Json, documents are stored as Json, and it responds with Json.
- Note that all numeric data is also stored in columns for fast querying and calculations.
- Most important commands are:
  - Create an index
  - Insert a record in the index
  - Delete a record from the index
  - Delete the whole index
  - Query the index
    - Search for records
    - Make calculations
- The Python layers for Elastic also use Json

```
from datetime import datetime
from elasticsearch import Elasticsearch

es = Elasticsearch([{'host': '127.0.0.1', 'port': 9200}])

es.indices.create(index='persons', ignore=400)

es.index(index="persons", id=1, body={"name": "wout",
"hobby": "programming", "age": 49, "timestamp":
datetime.now()})

es.index(index="persons", id=2, body={"name": "anna",
"hobby": "netflix", "age": 16, "timestamp": datetime.now()})
```

```
import json
from elasticsearch import Elasticsearch

es = Elasticsearch([{'host':'127.0.0.1','port':9200}])

search_body = {
    "query": {
        "bool": {
            "must": {
                "term":{
                    "hobby": "netflix"
                }
            }
        }
    }
}

result = es.search(index="persons", body=search_body)
print (json.dumps(result, indent=2))
```

```
{
   "took": 2,
   "timed_out": false,
   "_shards": {
      "total": 1,
      "successful": 1,
      "skipped": 0,
      "failed": 0
   },
   "hits": {
      "total": {
         "value": 1,
         "relation": "eq"
      },
      "max_score": 0.6931472,
      "hits": [
         {
            "_index": "persons",
            "_type": "_doc",
            "_id": "2",
            "_score": 0.6931472,
            "_source": {
               "name": "anna",
               "hobby": "netflix",
               "age": 16,
               "timestamp": "2019-09-22T22:47:46.869209"
            }
         }
      ]
   }
}
anna
```

Info on the shards that responded

General info about the query

Array with records that conform to the query

Meta information for the record

Source record

Dictionary

- How do I get to this data in Python?
- Dictionary: similar to a list (or array), but you don't give numbers as index. It is more generic, you can use any key as index:


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)

x = thisdict["model"]
```

Dictionary

- In Elasticsearch, the result of a query can be retrieved as a dictionary:
- `Result["data"]` gives back a dictionary
- `Result["data"]["cars"]` gives back an array
- `Result["data"]["cars"][0]` gives back the first element of the array.


```
{
    "name": "wout",
    "age": 49,
    "data": {
        "cars": ["Volvo",
        "Porsche",
        "Lexus"],
        "hobbies": ["programming",
        "18thcenturypoetry"]
    }
}
```

- `print(result["hits"]["hits"][0]["_source"]["name"])`

Assignment

- Next week (week 3): we work with gps-data. You will create your own (really simple version of) Strava.
- In week 4/5: Elasticsearch will be used in the weekly assignments.
- In the end assignment: everything comes together:
  - Meal delivery service
  - You get much data which you will import in Elasticsearch
  - Query and parse the data using Json
  - Work with Gps data
  - Optimization problem based on aggregate patterns in the data (e.g., determine number of couriers needed using the average number of orders per hour)

Any questions?