Data pipelines & Cloud infrastructure

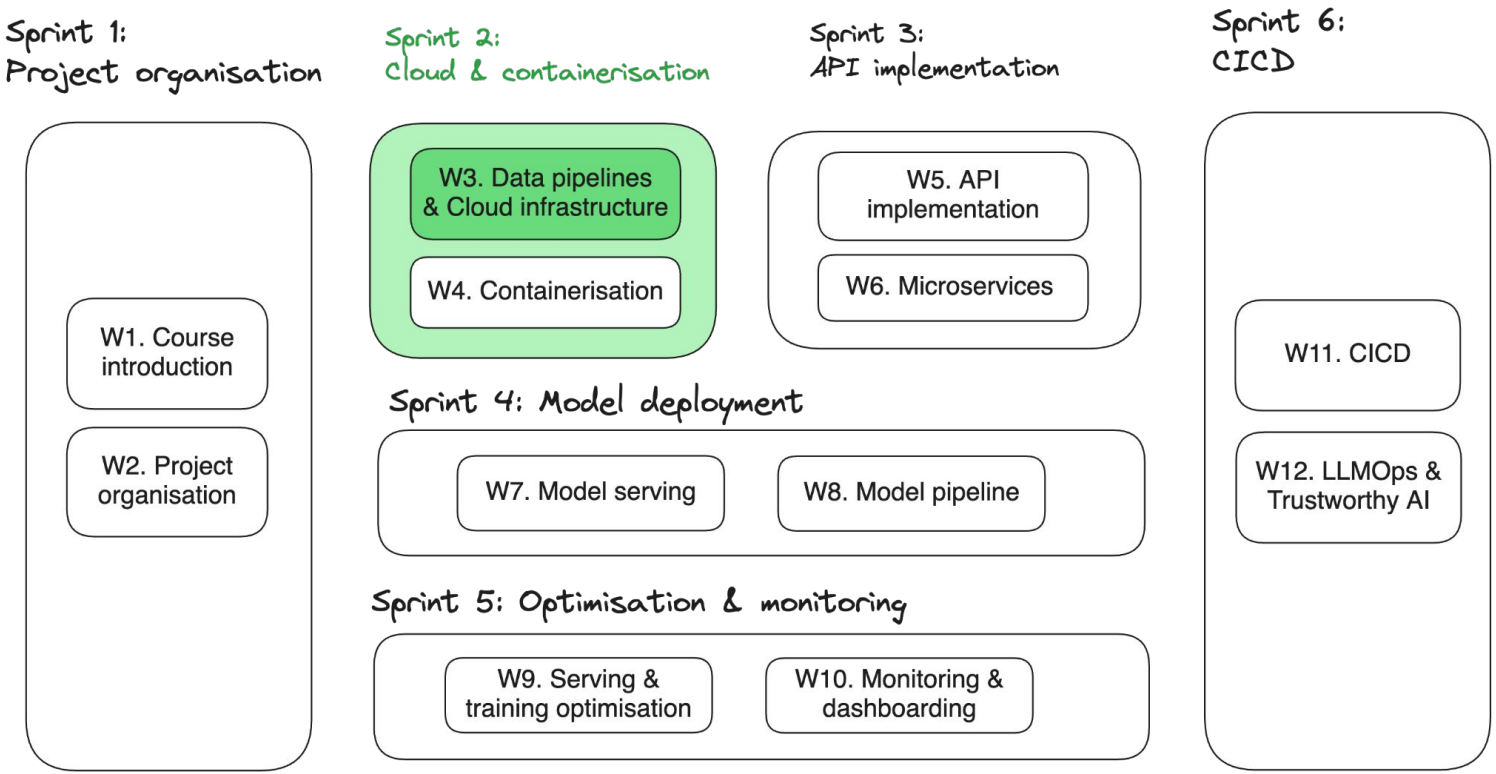
Sprint 2 - Week 3

INFO 9023 - Machine Learning Systems Design

Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

2025 Spring

Status on our overall course roadmap



Agenda

What will we talk about today

Lecture

1. Data formats
2. Data models
3. Databases
4. Exploratory Data Analysis (EDA)
5. Data cleaning & feature engineering

Guest lecture

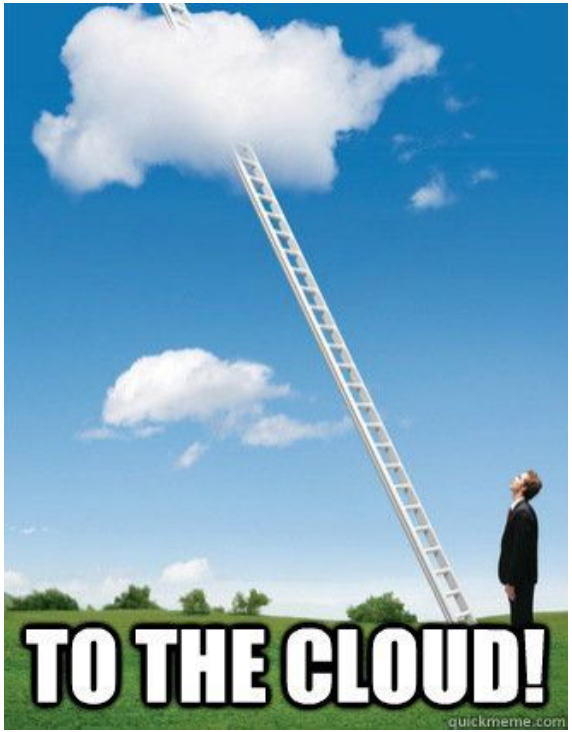
6. Cloud infrastructure


Always has been.
It’s all about data…
imgflip.com

Overall data process

Last week


THERE IS NO CLOUD
IT'S JUST
SOMEONE ELSE'S COMPUTER
imgflip.com


TO THE CLOUD!
quickmeme.com

Today’s objective and how to learn more about it

Objective: Overview of basic concepts as they relate to ML applications. Going in depth over database management is out of scope for this course.

Want to learn more about these topics? See:

- INFO0009: **Database**
  - Realization of computer systems centered on a database
  - Relational database, access, storage and more
- INFO9016: **Advanced Databases**
  - Distributed databases and their applications
  - Spatial databases, temporal databases, deductive databases, data warehousing, and NoSQL databases.
- INFO9014: **Knowledge representation and reasoning**
  - Ontologies, Knowledge Graphs, and Open Information Systems
  - Declarative approaches to reasoning and data integration
- INFO8002: **Topics in Distributed Systems**
  - Distributed File Systems and Distributed Programming
  - Hadoop, MapReduce, Spark

Project objective for sprint 2

This course focuses on what comes **around** your ML model.
Do **not** spend significant time optimising your data or model.
No grading on the **accuracy** of the model itself.

#	Week	Work package	Requirement
2.1	W03	Prepare your data and run an Exploratory Data Analysis.	Required
2.2	W03	Prepare your Cloud environment. That means creating a Cloud project, granting correct access rights to all members of your group and setting up a billing account. **Attention**: You can have free credits for the Cloud, as explained during the course.	Required
2.3	W04	Train your ML model	Required
2.4	W04	Evaluate your ML model	Required
2.5	W03 & W04	Document your data analysis and model performance	Required

Project milestones

- The main way to handover the results of your projects will be during the **3 milestones**.
  - The first Milestone will be used to present the work you did during sprint 1 & 2
  - The second Milestone for the work you did in sprint 3 & 4
  - The final milestone will be used to present the overall project outcome and work from sprint 5 & 6.
- The main purpose of the Milestones is to **provide feedback and guidance on the project**
- **Make it your own**! Focus on what is relevant and interesting. You are free to decide which material (if any) you will use to present your results (short slide deck, demo, show codes, ...).
- You will also need to do a pull request on Github and **email** the teaching staff for review
  - We will focus on the README and pull request description
  - Make sure to document any relevant part!

Project milestones

Practicals

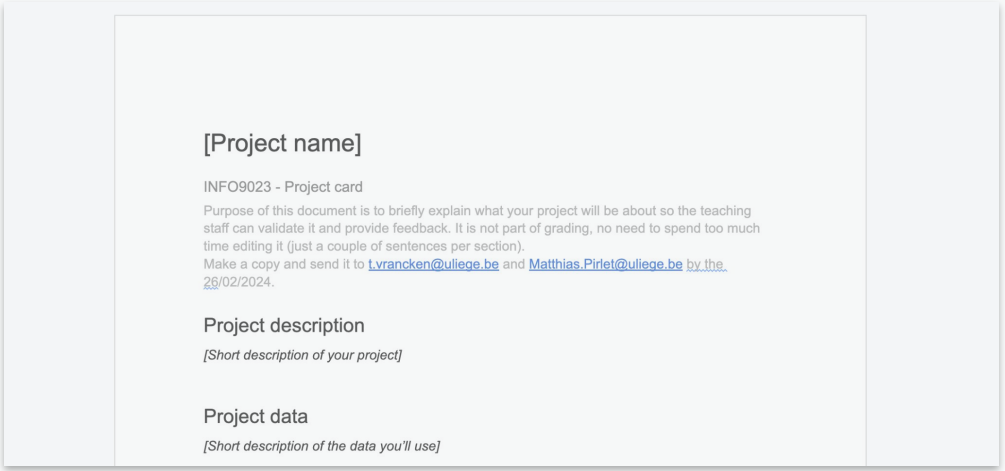
- The milestone presentation will take 15min (10min presentation + 5min QA)
- MS 1 will be the week of the 3rd of March. You have the choice to do it
  - In presence: Monday 3rd of march between 13:30 - 16:30
  - Online: Google Meets, on Tuesday 4th or Thursday 6th of March, always 13:30 - 16:30
- We will send a mail with a link to book a Google Meets slot

Project deliverables

Reminder from last week!
This should already have been sent!

⚠️ Submit your **team** and **project** by filling in this project card template (make a copy) and sending it to the teaching staff.

- Purpose is for the teaching staff to validate the project ideas and potentially provide feedback.
- Fill it in before next lesson.



[Project name]

INFO9023 - Project card

Purpose of this document is to briefly explain what your project will be about so the teaching staff can validate it and provide feedback. It is not part of grading, no need to spend too much time editing it (just a couple of sentences per section).
Make a copy and send it to t.vrancken@uliege.be and Matthias.Pirlet@uliege.be by the 26/02/2024.

Project description

[Short description of your project]

Project data

[Short description of the data you'll use]

LIÈGE université

Data formats

Data formats

	Format	Binary/Text	Human-readable	Example use cases
	JSON	Text	Yes	Everywhere
Row-major	**CSV**	**Text**	**Yes**	**Everywhere**
Column-major	**Parquet**	**Binary**	**No**	**Amazon Redshift**
	Avro	Binary primary	No	Hadoop
	Protobuf	Binary primary	No	TensorFlow (TFRecord)
	Pickle	Binary	No	Python, PyTorch serialization

Data formats

Serialisation: process of converting data structures (like objects, tables, or messages) into a format that can be stored or transmitted and then later reconstructed (deserialized)

Binary Primary Formats: Designed for binary serialization and structured data exchange.

- Compact & Efficient: Uses less space compared to generic binary formats.
- Schema-based: Ensures structured and predictable serialization.
- Cross-platform: Can be used across different languages and systems.
- Fast Serialization & Deserialization: Optimized for rapid encoding/decoding.

Binary Formats: Specific purposes

- Parquet: Designed for efficient storage and querying, not necessarily for real-time data transmission.
- Pickle: Python-specific, not portable to other languages, and not optimized for speed in large-scale distributed systems.

Row-major vs column-major

Column-major

- Stored and retrieved column by column
- Good for accessing features or computing statistics

A = [[1, 2, 3],
[4, 5, 6]]
Stored ⇒ [1, 4, 2, 5, 3, 6]

Row-major

Stored and retrieved row by row
Good for accessing samples efficiently (uniquely or by rule)

A = [[1, 2, 3],
[4, 5, 6]]
Stored ⇒ [1, 2, 3, 4, 5, 6]

Column 1	Column 2	Column 3	...
Sample 1			
Sample 2			
Sample 3			
Sample 4			
…			

Row-major vs. column-major

Often leads to misuse of pandas…

- Pandas Dataframe is **column-major** while Numpy is **row-major**.
- Accessing a **column in pandas** is significantly **faster** than accessing a row
- Numpy is often faster than pandas
- Numpy can be specified to be column-based

```
# Get the column `date`, 1000 loops
%timeit -n1000 df["Date"]

# Get the first row, 1000 loops
%timeit -n1000 df.iloc[0]
```

```
1.78 µs ± 167 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
145 µs ± 9.41 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
df_np = df.to_numpy()
%timeit -n1000 df_np[0]
%timeit -n1000 df_np[:,0]
```

```
147 ns ± 1.54 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
204 ns ± 0.678 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Text vs binary format

	Text files	**Binary files**
Examples	CSV, JSON	Parquet
Pros	Human readable	Compact
Store the number *1000000*?	7 characters -> 7 bytes	If stored as int32, only 4 bytes

You can unload the result of an Amazon Redshift query to your Amazon S3 data lake in Apache Parquet, an efficient open columnar storage format for analytics. Parquet format is up to 2x faster to unload and consumes up to 6x less storage in Amazon S3, compared with text formats. This enables you to save data transformation and enrichment you have done in

Text vs binary format

	Text files	Binary files
Examples	CSV, JSON	Parquet
Pros	Human readable	Compact
Store the number *1000000*?	7 characters -> 7 bytes	If store

You can unload the result of an Amazon Redshift query to your Amazon S3 data lake in Apache Parquet, an efficient open columnar storage format for analytics. Parquet format is up to 2x faster to unload and consumes up to 6x less storage in Amazon S3, compared with text formats. This enables you to save data transformation and enrichment you have done in

Data models

Data models

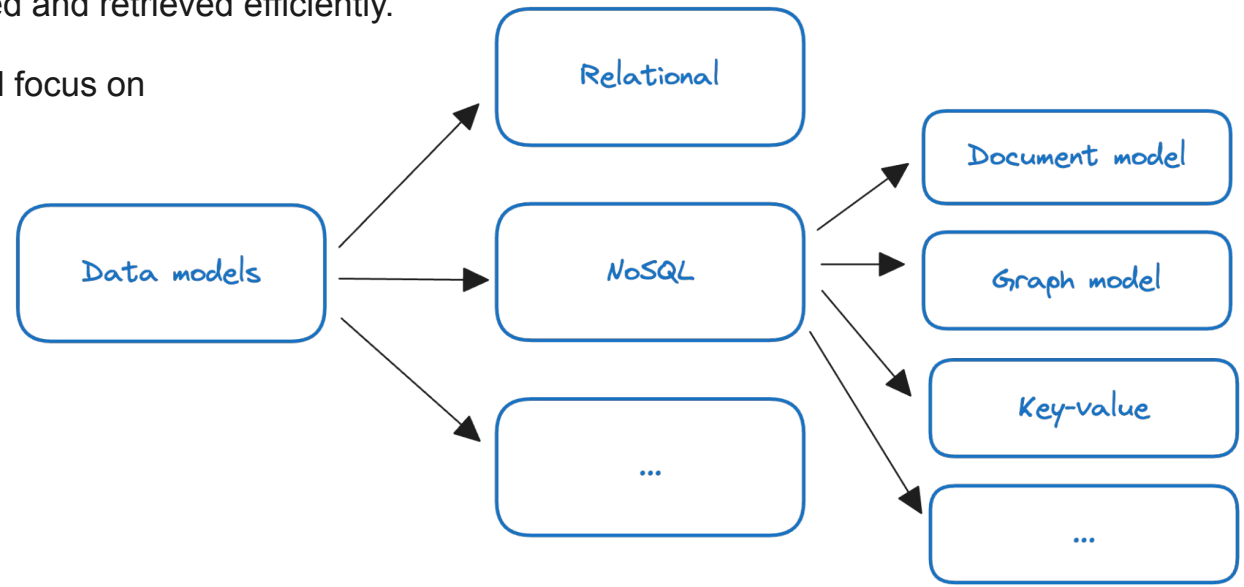
A **data model** describes how your data is represented.

It defines the structure, organisation and relationship of your data.

It ensures that your data is managed and retrieved efficiently.

There are many models, but we will focus on

relational and **NoSQL**.



Relational data model (RDBMS)

- Persistent idea in computer science, introduced by Edgar F. Codd in 1970.
- Data is organised in **relation** and each relation is a set of **tuples** (e.g. a row from a table)

Row (tuple)

Column 1	Column 2	Column 3	...
Sample 1			
Sample 2			
Sample 3			
...			

Relational data model

… the dawn of all civilised computer science…

Normalisation of Relational data model

Normalisation is the act of reducing repetition of data by separating a table into multiple connected tables.

Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	US	$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	$15

Title	Author	Format	Publisher ID	Price
Harry Potter	J.K. Rowling	Paperback	1	$20
Harry Potter	J.K. Rowling	E-book	1	$10
Sherlock Holmes	Conan Doyle	Paperback	2	$30
The Hobbit	J.R.R. Tolkien	Paperback	1	$30
Sherlock Holmes	Conan Doyle	Paperback	2	$15

Publisher ID	Publisher	Country
1	Banana Press	UK
2	Guava Press	US

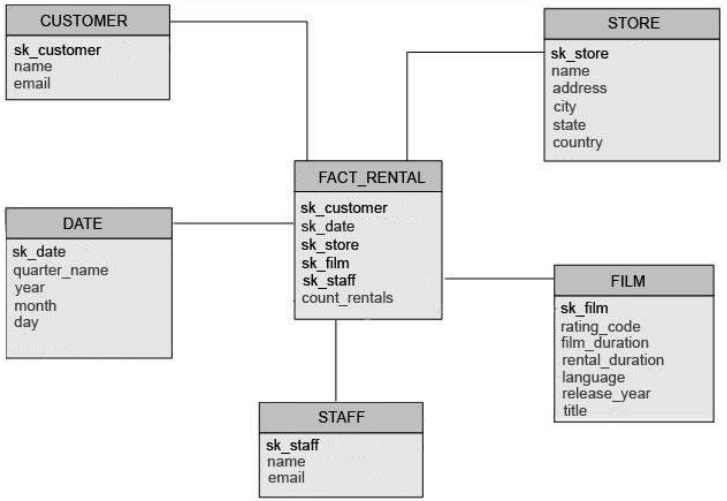
Normalisation of Relational data model

A star schema is a type of relational database schema that is composed of a single, central fact table that is surrounded by dimension tables

1. **Fact tables** store quantitative data corresponding to business transactions or events, containing measures (e.g., sales amount, revenue) and foreign keys linking to dimension tables
2. **Dimension tables** provide descriptive context for facts, containing attributes (e.g., product category, customer demographics) that enable filtering, grouping, and analysis.

Splitting your data between fact and dimension tables is called **normalisation**.

Fact tables have a high cardinality and are normalized to avoid redundancy, whereas dimension tables may be partially denormalized to enhance query performance.



Borges, Clemente & Macías, José. (2014). Facilitating the Interaction with Data Warehouse Schemas through a Visual Web-Based Approach. Computer Science and Information Systems. 11. 481-501. 10.2298/CSIS131130032B.

Different forms for normalisation of relational data models

Normalisation of relational data models exists in different forms

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)
6. Fifth Normal Form (5NF) / Project-Join Normal Form (PJNF)
7. Sixth Normal Form (6NF)

https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Different forms for normalisation of relational data models

Normalisation of relational data models exists in different forms

1. **First Normal Form (1NF)**
2. **Second Normal Form (2NF)**
3. **Third Normal Form (3NF)**
4. Boyce-Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)
6. Fifth Normal Form (5NF) / Project-Join Normal Form (PJNF)
7. Sixth Normal Form (6NF)

We will only the first 3 forms of normalisation as those cover most real-world implementations.

"[Higher form of normalisation] do exist but are rarely considered in practical design. Disregarding these rules may result in less-than-perfect database design but shouldn't affect functionality."

https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Different forms for normalisation of relational data models

Starting with a base example

Title	Author	Author Nationality	Format	Price	Subject	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Chad Russell	American	Hardcover	49.99	MySQL Database Design	520	Thick	Apress	USA	1	Tutorial

https://en.wikipedia.org/wiki/Database_normalization

Different forms for normalisation of relational data models

First normal form (1NF)

In the **first normal form** each field contains a single value. A field may not contain a set of values or a nested record.

Book

Title	Author	Author Nationality	Format	Price	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Chad Russell	American	Hardcover	49.99	520	Thick	Apress	USA	1	Tutorial

Title - Subject

Title	Subject name
Beginning MySQL Database Design and Optimization	MySQL
Beginning MySQL Database Design and Optimization	Database
Beginning MySQL Database Design and Optimization	Design

https://en.wikipedia.org/wiki/Database_normalization

Different forms for normalisation of relational data models

Second normal form (2NF)

In the **second normal form** all key attribute must be fully determinant to all non-key attributes. You must not be able to reduce key information. Mostly relevant for composite-keys that can be simplified.

Does not satisfy 2NF.

Composite key {title, format} is not **determinant**.
Some subset of facts can be determined only on part of the composite key (only “price” is dependent on “format”)

Book

Title	Format	Author	Author Nationality	Price	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Hardcover	Chad Russell	American	49.99	520	Thick	Apress	USA	1	Tutorial
Beginning MySQL Database Design and Optimization	E-book	Chad Russell	American	22.34	520	Thick	Apress	USA	1	Tutorial
The Relational Model for Database Management: Version 2	E-book	E.F.Codd	British	13.88	538	Thick	Addison-Wesley	USA	2	Popular science
The Relational Model for Database Management: Version 2	Paperback	E.F.Codd	British	39.99	538	Thick	Addison-Wesley	USA	2	Popular science

https://en.wikipedia.org/wiki/Database_normalization

Different forms for normalisation of relational data models

Second normal form (2NF)

In the **second normal form** all key attribute must be fully determinant to all non-key attributes. You must not be able to reduce key information. Mostly relevant for composite-keys that can be simplified.

To conform to 2NF, split the table so that

- all attributes that are determined based on {title} are in the *book* table
- all the attributes dependent on {title, format} are in the *price* table

Book

Title	Author	Author Nationality	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	Apress	USA	1	Tutorial
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	Addison-Wesley	USA	2	Popular science

Price

Title	Format	Price
Beginning MySQL Database Design and Optimization	Hardcover	49.99
Beginning MySQL Database Design and Optimization	E-book	22.34
The Relational Model for Database Management: Version 2	E-book	13.88
The Relational Model for Database Management: Version 2	Paperback	39.99

https://en.wikipedia.org/wiki/Database_normalization

Different forms for normalisation of relational data models

Third normal form (3NF)

In the **third normal form** there are no transitive functional dependencies between attributes.
A *transitive functional dependency* occurs when a non-key attribute in a database table is functionally dependent on another non-key attribute, rather than directly on the primary key.

Does not satisfy 3NF.

Many transitive dependencies:

- Title → Author → Author nationality
- Title → Publisher → Publisher country
- Title → Genre ID → Genre name

Transitive functional dependency

Book

Title	Author	Author Nationality	Pages	Thickness	Publisher	Publisher Country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	Apress	USA	1	Tutorial
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	Addison-Wesley	USA	2	Popular science

Price

Title	Format	Price
Beginning MySQL Database Design and Optimization	Hardcover	49.99
Beginning MySQL Database Design and Optimization	E-book	22.34
The Relational Model for Database Management: Version 2	E-book	13.88
The Relational Model for Database Management: Version 2	Paperback	39.99

https://en.wikipedia.org/wiki/Database_normalization

Different forms for normalisation of relational data models

Third normal form (3NF)

In the **third normal form** there are no transitive functional dependencies between attributes.
A *transitive functional dependency* occurs when a non-key attribute in a database table is functionally dependent on another non-key attribute, rather than directly on the primary key.

Satisfies 3NF ! ✅

Book

Title	Author	Pages	Thickness	Publisher	Genre ID
Beginning MySQL Database Design and Optimization	Chad Russell	520	Thick	Apress	1
The Relational Model for Database Management: Version 2	E.F.Codd	538	Thick	Addison-Wesley	2

Price

Title	Format	Price
Beginning MySQL Database Design and Optimization	Hardcover	49.99
Beginning MySQL Database Design and Optimization	E-book	22.34
The Relational Model for Database Management: Version 2	E-book	13.88
The Relational Model for Database Management: Version 2	Paperback	39.99

Author

Author	Author Nationality
Chad Russell	American
E.F.Codd	British

Publisher

Publisher	Country
Apress	USA
Addison-Wesley	USA

Genre

Genre ID	Name
1	Tutorial
2	Popular science

https://en.wikipedia.org/wiki/Database_normalization

Normalisation of Relational data model

Pros & Cons

Pros	Cons
• Removes redundancy - memory efficient • Consistency - standardised values • Easy to update a value throughout all occurrences • More security flexibility by restricting access to certain information	• Data spread across multiple relations • Query performance - Joining can be an expensive/slow operation • More complexity • More maintenance

SQL query language

- SQL is a **query language**, it allows you to query most relational data models (and more!)
- SQL is a **declarative language**
  - You describe the outputs you want and the computer figures out the steps needed to retrieve it and how to optimize them.
  - By opposition languages such as Python are **imperative languages**
    - You declare the exact steps that the computer should execute.

```
SELECT
  EXTRACT(YEAR FROM starttime) AS year,
  EXTRACT(MONTH FROM starttime) AS month,
  COUNT(starttime) AS number_one_way
FROM
  mydb.return_transactions
WHERE
  start_station_name != end_station_name
GROUP BY year, month
ORDER BY year ASC, month ASC
```

Dividing SQL into sublanguages

SQL operations are divided into multiple **sub-languages** such as:

- **Data Query Language (DQL)** is responsible for reading, or querying, data from a database. In SQL, this corresponds to the `SELECT`
- **Data Manipulation Language (DML)** is responsible for adding, editing or deleting data from a database. In SQL, this corresponds to the `INSERT`, `UPDATE`, and `DELETE`
- **Data Definition Language (DDL)** is responsible for defining the way data is structured in a database. In SQL, this corresponds to manipulating tables through the `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE`
- **Data Control Language (DCL)** is responsible for the administrative tasks of controlling the database itself, most notably granting and revoking database permissions for users. In SQL, this corresponds to the `GRANT`, `REVOKE`, and `DENY` commands (among other things)

SQL is so popular that DQL and DML influence many NoSQL databases.

SQL is prevalent in data management!
Important to learn 🧠

NoSQL

Called “Not Only SQL” to cover data models that do not fit into relational data model (SQL)

- **Document model**: targets use cases where data comes in self-contained documents and relationships between one document and another are rare.
- **Graph model**: goes in the opposite direction, targeting use cases where relationships between data items are common and important.
- **Key-value**: a simple model where data is stored as a collection of key-value pairs, making it ideal for applications that require fast lookups and simple retrieval patterns, such as caching or session management.

Surprise #2: Hating on RDBMS. There was ample room for free-form response in the survey, and a lot of people took the opportunity to bash on relational database technology. There were some colorful RDBMS-related responses to the question what's your biggest hope for NoSQL in 2012? And many of them were downright angry. While NoSQL was for a time the “new shiny thing” that many wanted to play with, it's pretty clear that most of the respondents are looking to NoSQL technology not because it is what the cool kids are doing, but because they are trying to eliminate real pain. What pain you might ask?

https://www.couchbase.com/press-releases/couchbase-survey-shows-accelerated-adoption-of-nosql-in-2012/

NoSQL

Document model

A document store organizes data in a semi-structured format (e.g., JSON, BSON, XML)

- Unlike relational tables, documents can have **flexible schemas**.
- Documents are self-contained, meaning they include both *data* and *metadata*
  - Can be queried on different fields
- Often documents are represented in **text format** (JSON, XML, …)
- Documents databases use **unique key identifiers**
  - Faster to retrieve item on it
- Note that you can transform a relational model (e.g. table) into a document model
  - You then gain the flexibility of adding new fields while losing the relational aspect


```
[
    {
        "id": 1,
        "title": "The Great Gatsby",
        "author": "F. Scott Fitzgerald",
        "publication_year": 1925
    },
    {
        "id": 2,
        "title": "1984",
        "author": "George Orwell",
        "publication_year": 1949,
        "genre": "Dystopian",
        "page_count": 328
    },
    {
        "id": 3,
        "title": "To Kill a Mockingbird",
        "author": "Harper Lee",
        "chapters": ["Chapter 1", "Chapter 2", "Chapter 3"],
        "publisher": "J. B. Lippincott & Co."
    }
]
```

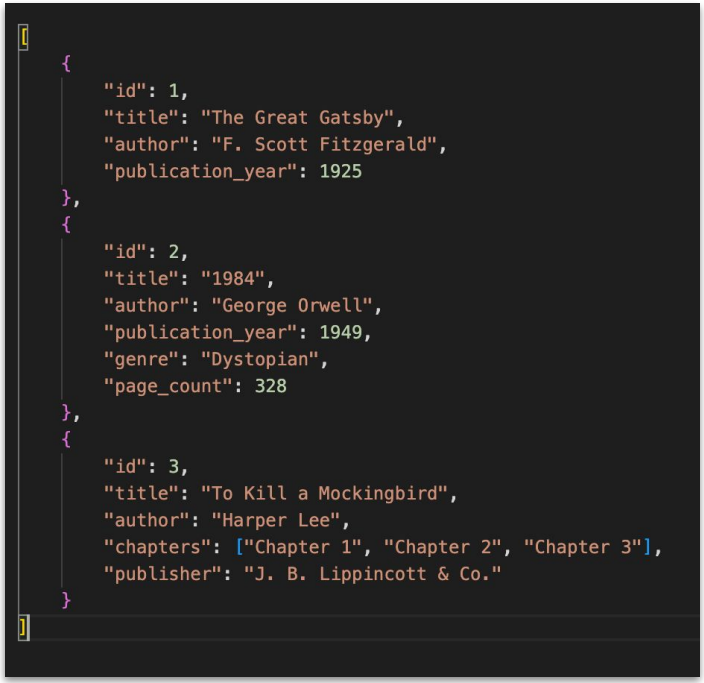
NoSQL

Document model

Standard Examples:

- MongoDB (most popular, JSON-like storage)
- CouchDB (focuses on replication and offline sync)
- Firestore (Firebase) (real-time cloud-based NoSQL)

Pros	Cons
Flexible schema allows easy modifications	Not optimized for complex relationships (joins)
Supports hierarchical and nested data structures	Indexing large documents can be challenging
Good for applications requiring varied data formats (e.g., content management, catalogs)	Query performance can degrade if documents grow too large



NoSQL

Graph model

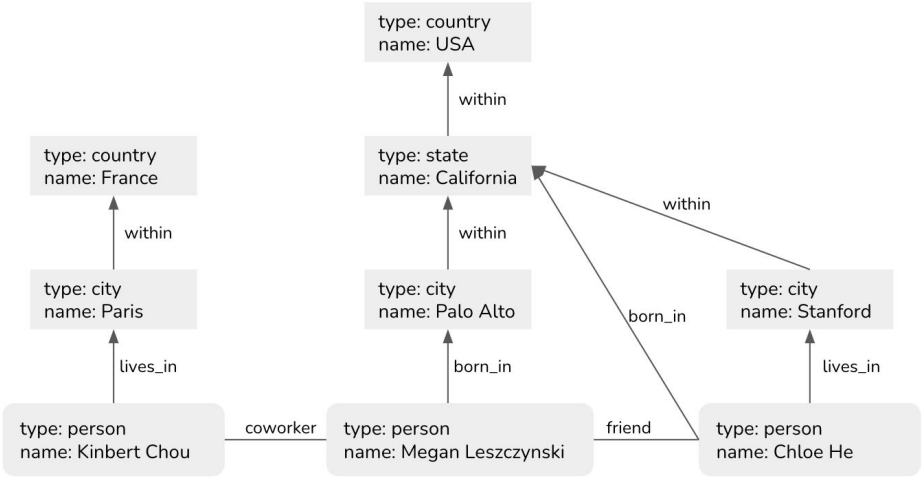
A **graph** consists of **nodes** and **edges.**

The edges describe the relationship between the nodes.

Fast to query based on relationships

- Find all people born in the US
  - Follow the edges linked to US
  - In relational database we would have to do a full search on the “born_in” field
  - Document database would be even worst…

Neo4j is a popular database for Graph models.



NoSQL

Key-Value

A key-value store is a simple data model where data is stored as a collection of key-value pairs. The key is a unique identifier, and the value can be a simple data type (string, integer) or a more complex object (JSON, BLOB).

The content of the “value” of each data point is opaque to the database. You cannot query against it.

Pros	**Cons**
Extremely fast read/write operations (O(1) complexity)	Limited querying capabilities (no structured search)
Simple to scale horizontally	Cannot perform complex filtering, sorting, or aggregations efficiently
Highly efficient for caching, session storage, and configuration management	Keys must be known in advance for efficient lookups

Databases

Data base

Data models define how you represent and store your data.

Databases are the implementation of how data is stored and retrieve on **machines**.

Important to know as all ML application will be built on top of databases.

We look at two types of *relational* databases:

- **Transactional** databases
- **Analytical** databases

OnLine Transaction Processing (OLTP)

OLTP databases are built to perform high amounts of **transactions**, which can be a sequence of operations such as insert, update, delete, or retrieve. High volume of low complexity transactions.

OLTP databases aim at bringing **low latency** and **high availability**. Often used to store and access **current** data.

A core concept of OLTP is **ACID transactions** to the database:

- **Atomicity**: A transaction is all-or-nothing, meaning either all operations succeed, or none are applied.
- **Consistency**: All the transactions coming through must follow predefined rules. For example, a transaction must be made by a valid user.
- **Isolation**: Concurrent transactions execute as if they were sequential, preventing interference.
- **Durability**: Once a transaction is committed, its changes persist even in the event of a system failure.

https://www.ibm.com/blog/olap-vs-oltp/

OnLine Analytical Processing (OLAP)

OLAP databases are built to perform complex analytical operations over a large amount of rows. Compute a lower volume of heavier jobs that require to perform calculations over data.

Used for **analytical** jobs.

E.g.

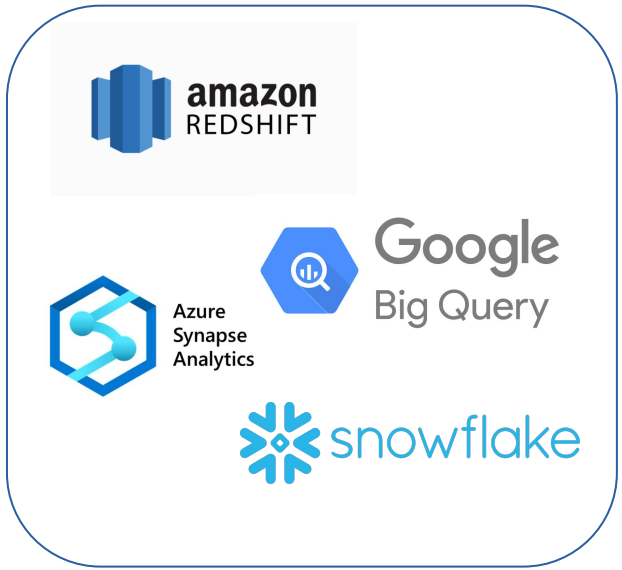
- *What is the average purchase amount for users of a specific age?*
- *What is the maximum house price in a specific location for a specific date range?*
- *…*

https://www.jedox.com/en

Concretely… OLTP and OLAP options

OLTP databases

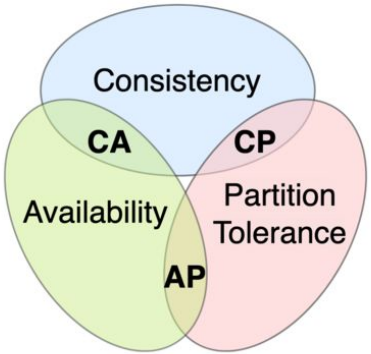
OLAP databases



CAP theorem

In database theory, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that any distributed data store can provide only two of the following three guarantees:

- Consistency: Every read receives the most recent write or an error.
- Availability: Every request received by a non-failing node in the system must result in a response.
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.



Typically, you need to tradeoff between consistency and latency.

https://en.wikipedia.org/wiki/CAP_theorem

Optimising a relational database.

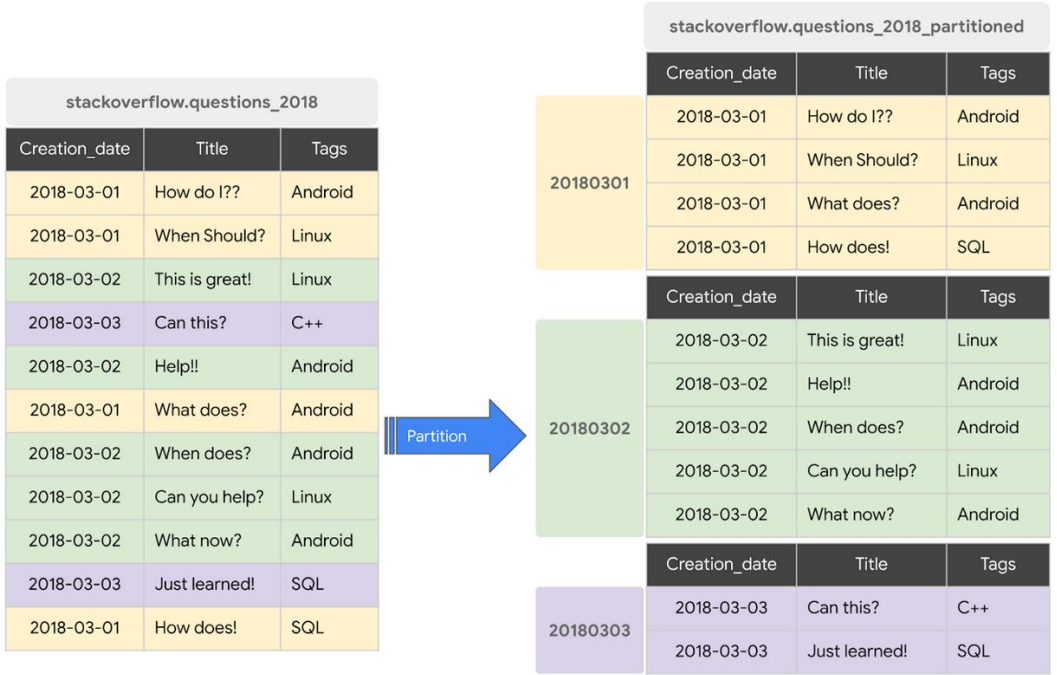
- **Indexing**: Data structure applied to one (or multiple) key columns to organise and speed the data retrieval operations applied on this column. Avoids searching the entire column for specific values. A downside of it is a higher memory consumption.
- **Query optimisation**: Selecting only required columns, using `WHERE` operations effectively. Know how the query is constructed and operated.
- **Caching**: Store frequently accessed results in RAM. Important for more complex systems (e.g. data/ML pipeline).
- **Hardware optimisation**: Obviously some hardware upgrade can significantly improve your database performance (e.g. faster CPU or SSDs). Sometimes abstracted away when using managed Cloud database.
- **Partitioning**: … ⬇️

Efficiency example: Partitioning your data

Divide table into segments, called **partitions**, that make it easier to manage and query your data.

You can typically split large tables into many smaller partitions using data ingestion time or TIMESTAMP/DATE column or an INTEGER column (e.g. if you will often apply operations on specific time frames).

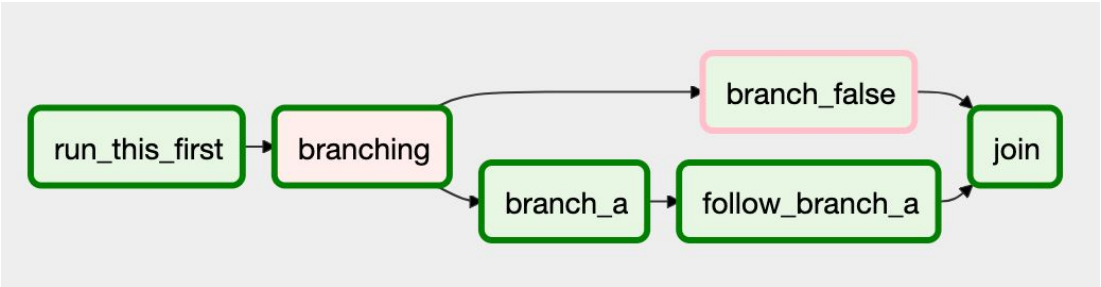
Decoupled storage and compute architecture leverages column-based partitioning simply to minimize the amount of data that slot workers read from disk.



https://cloud.google.com/blog/topics/developers-practitioners/bigquery-explained-storage-overview

Airflow

- Workflow orchestration tool often used for ETL pipelines.
- Easily scheduled or triggered
- Allows defining ETL tasks as Directed Acyclic Graphs (DAGs).
- Integrates well with cloud services and modern data stacks.



Not a database!

Spark

An open-source, distributed computing engine designed for large-scale data processing.

Built for speed and ease of use, leveraging in-memory computation.

Key features

- In-memory computing → Faster execution than disk-based systems.
- Lazy evaluation → Optimized execution plans for efficiency.
- Resilient Distributed Datasets (RDDs) → Fault-tolerant data abstraction.
- Scalability → Runs on clusters using YARN, Mesos, or Kubernetes.

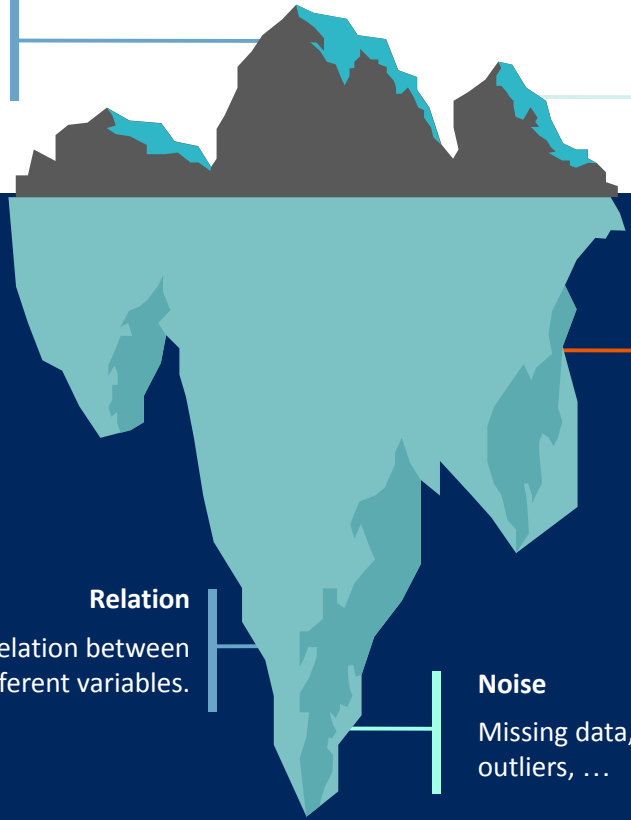
Not a database!

Exploratory Data Analysis

Assuming you understand your data is a common pitfall

Looking at a few observations

Reading a few rows and assuming it's representative



Looking at a few columns

Assuming from a column's title that you understand what it represents

Distribution

Skewed or unexpected distribution.

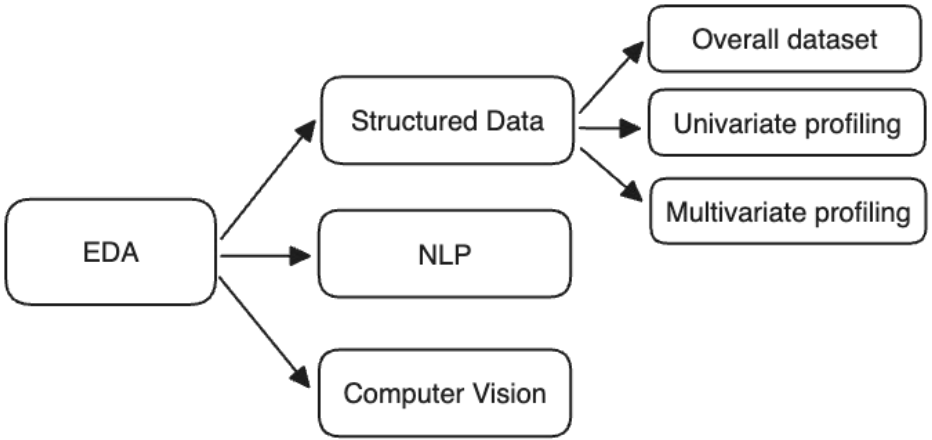
Relation

Unknown relation between different variables.

Noise

Missing data, formatting issue, outliers, …

Different types of data to explore and steps to take



Exploratory Data Analysis (EDA) on structured data

First, know that there are different tools to use for the EDA

- Doing it yourself: **Pandas** (or else)
  - More freedom / customisation
  - More efforts
  - Risk to overlook important part
- Useful tool: **YData Profiling**
  - More managed
  - Risk to not dig deep enough in specific cases

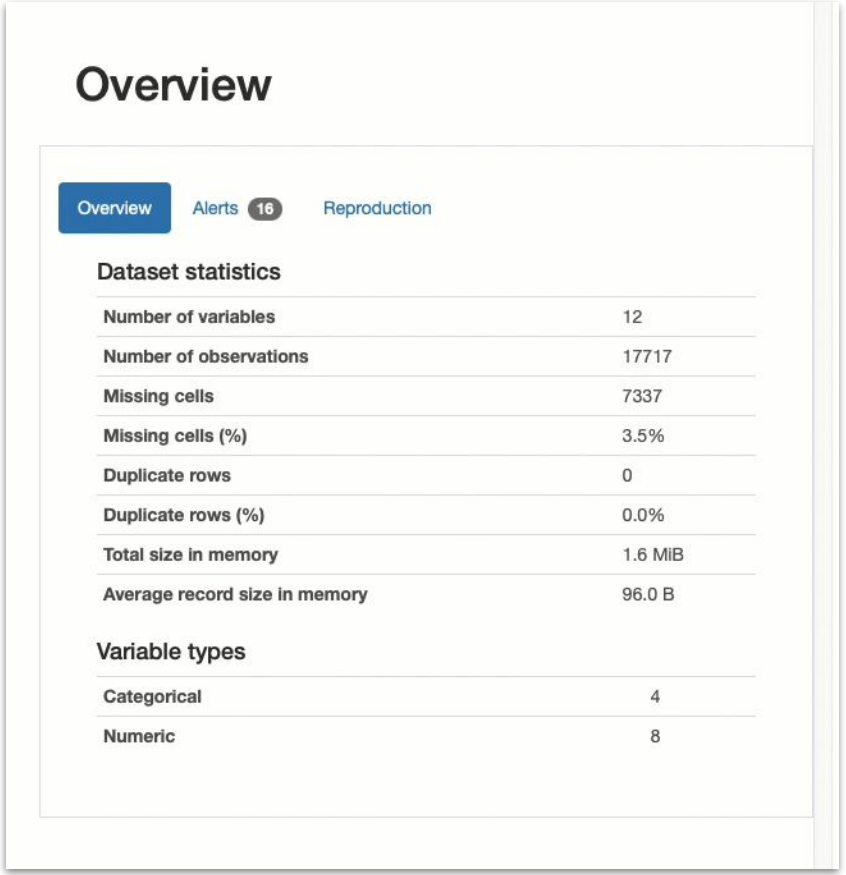
```
import pandas as pd
from pandas_profiling import ProfileReport


# Read the HCC Dataset
df = pd.read_csv("hcc.csv")

# Produce the data profiling report
original_report = ProfileReport(df, title='Original
Data')
original_report.to_file("original_report.html")
```


https://docs.profiling.ydata.ai/4.6/ – https://pandas.pydata.org/

Overall dataset sanity and noise

Look for

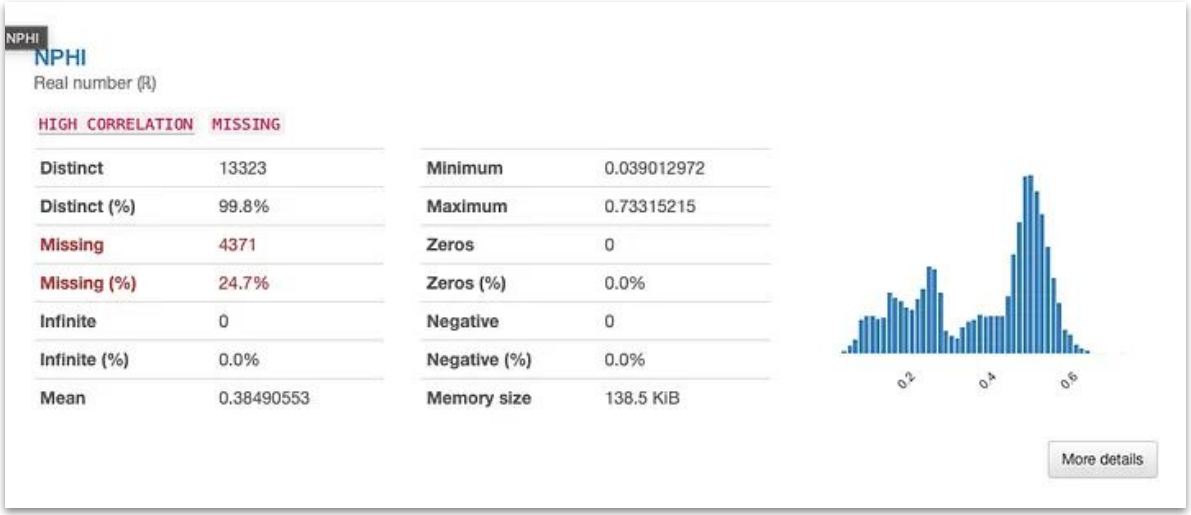
- General statistics
  - Memory size
  - Number of rows
  - Column types
  - …
- You can look for duplicated columns
  - `df.duplicated(subset=[columns])`
- Or rows with too many missing values



https://towardsdatascience.com/5-powerful-python-libraries-you-need-to-know-to-enhance-your-eda-process-f0100d563c16

Univariate profiling

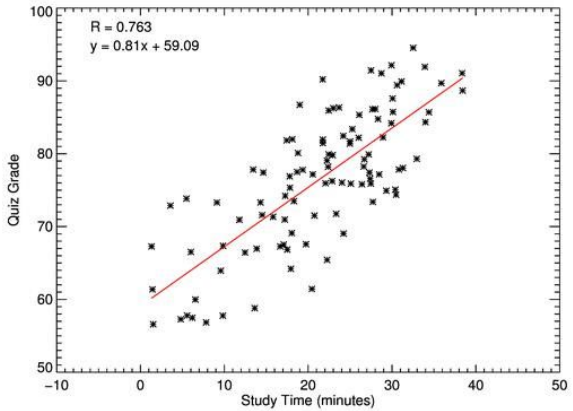
- Understanding **independent** variables
- You can use pandas **describe** and a **histogram** of value counts
- YData Profiling shows is in the **variable** tab

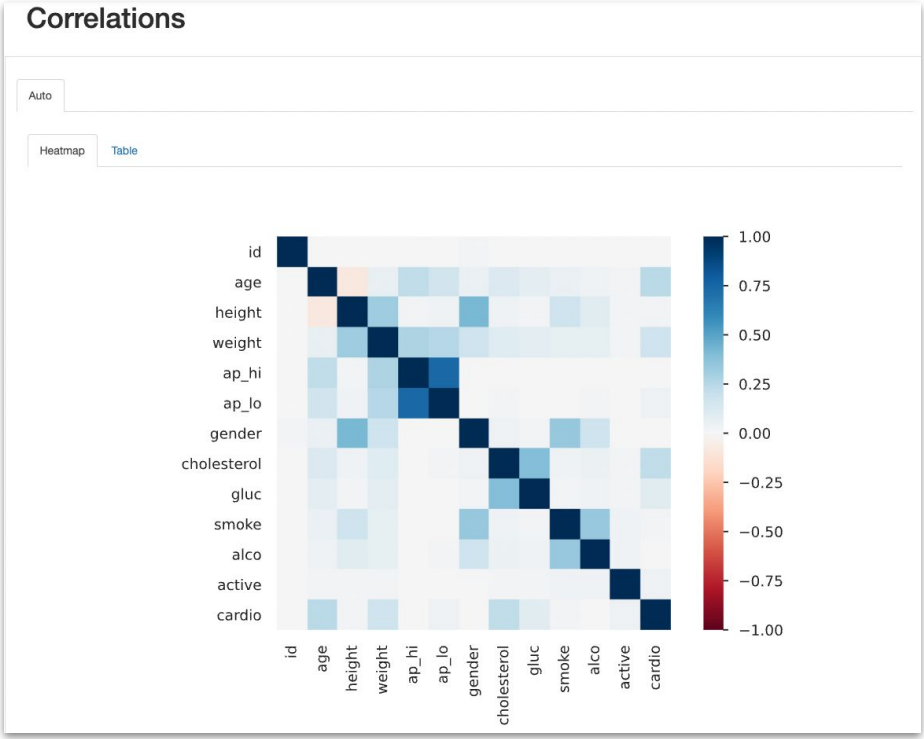


https://towardsdatascience.com/5-powerful-python-libraries-you-need-to-know-to-enhance-your-eda-process-f0100d563c16

Multivariate profiling

- Look at the relationship between different variables
- Plot **pair of variables** against each other
- Plot **correlation matrix**



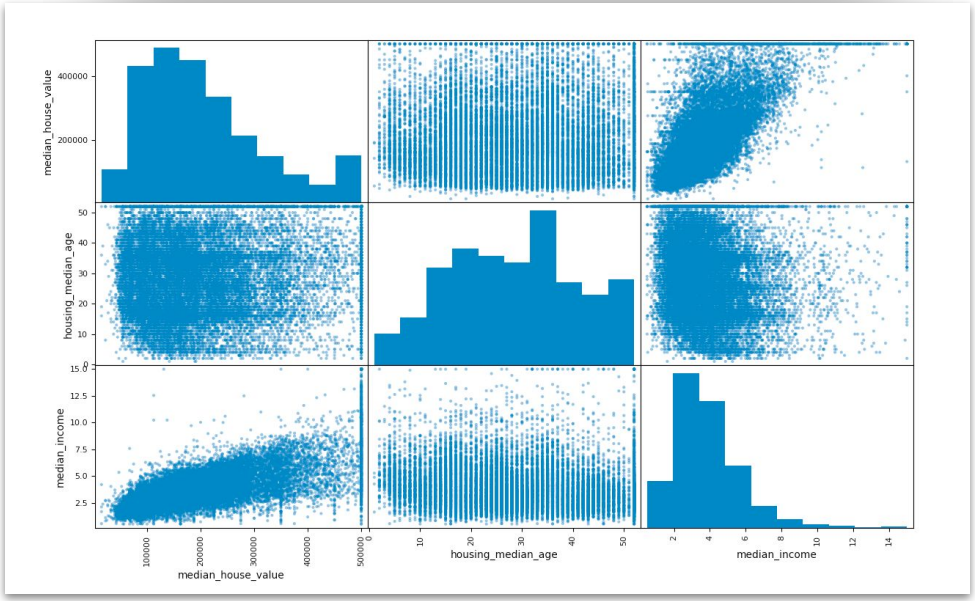


Ajibade, Samuel & Taiwo, Fele & Adedara,. (2016). A Systematic Review of Data Visualization Methods.

Multivariate profiling

- … Or with a **Matrix** of **scatter plots** for each pair of variables

```python
from pandas.plotting import scatter_matrix

df = pd.DataFrame(np.random.randn(1000, 4), columns=["a", "b", "c",
"d"])
scatter_matrix(df, alpha=0.2, figsize=(6, 6), diagonal="kde");
```



https://pandas.pydata.org/docs/user_guide/visualization.html#scatter-matrix-plot

EDA for Natural Language Processing

Context:

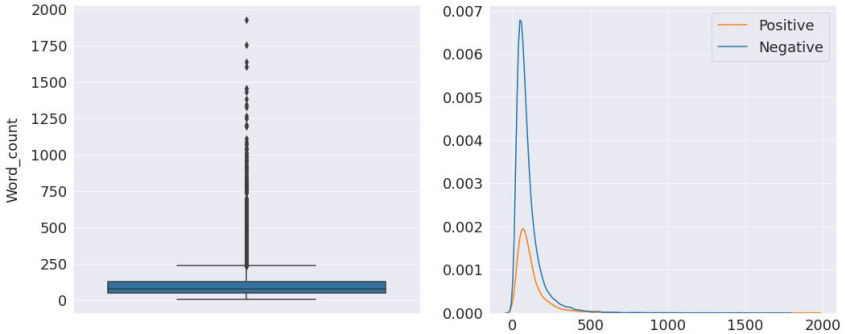
- Assume you work for a **publishing company** that has **300k news articles** from the last 30 years
- They want to build a **semantic search engine**
- The documents use domain specific language so you want to **fine-tune a text embedding model**
- For that, you need a labeled data set of **Question and Answer pairs**
    - We want it to specifically use the **language** and **domain** of the newspaper publisher
    - The publisher only has documents

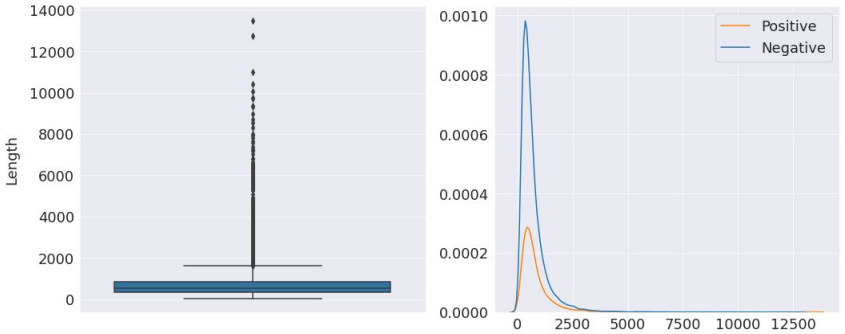
Open question:

- What do you need to check before using the data?

EDA for Natural Language Processing

- Statistical distribution of your texts
  - *Relevant to know token size if you want to use an API (Embed, generate, …)*





LIÈGE université

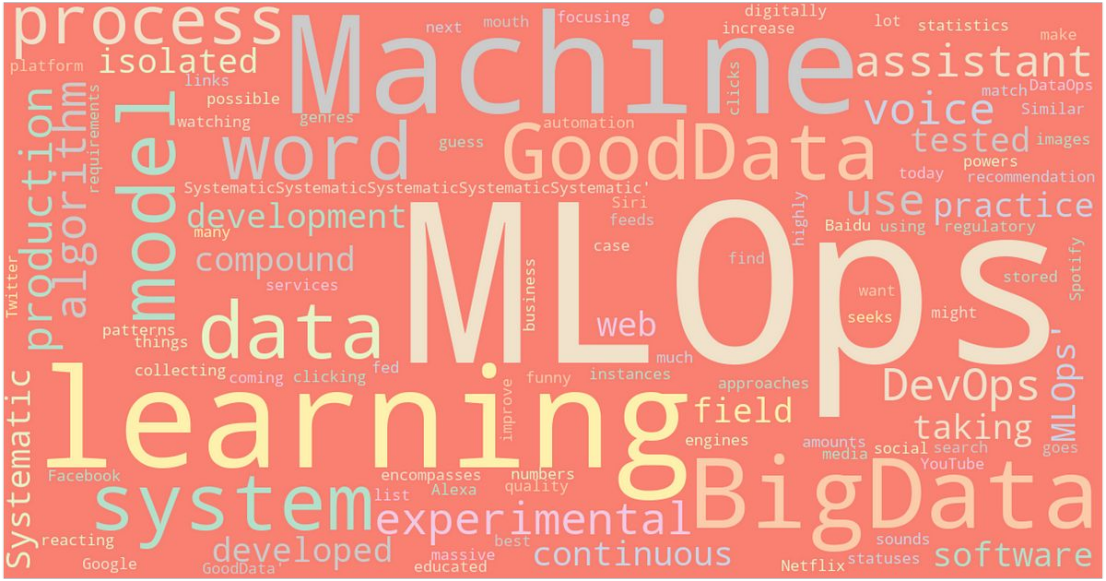
EDA for Natural Language Processing

Organically explore and **read through data samples** of texts

- General understanding
- Get a feeling of amount of **noise** or specific **cleaning steps require** (e.g. run into HTML formatted data)
- Easy to get biased understanding! Can't go through a significant sample

Wordclouds

- Interesting to get a sense of topics
- In reality not super useful…

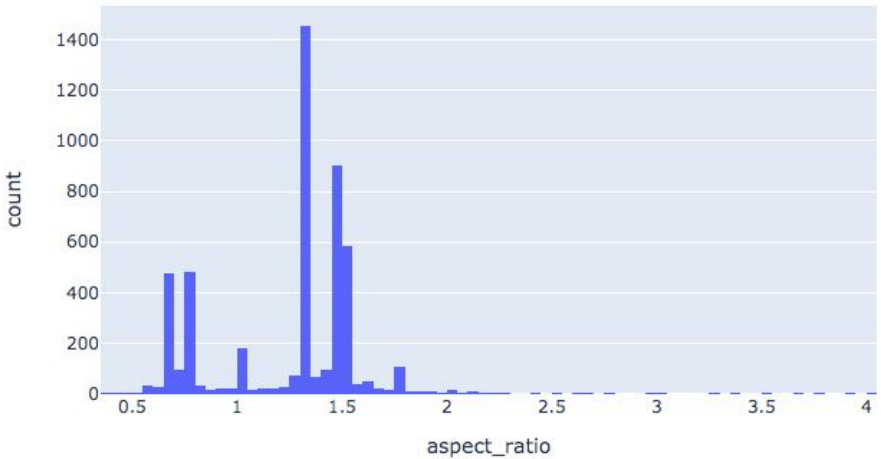


LIÈGE université

EDA for Computer Vision

Look at image **sizes** and **aspect ratios**.

Can help you decide on **type of resizing** (destructive or not, desired output resolution, padding, …).



https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

EDA for Computer Vision

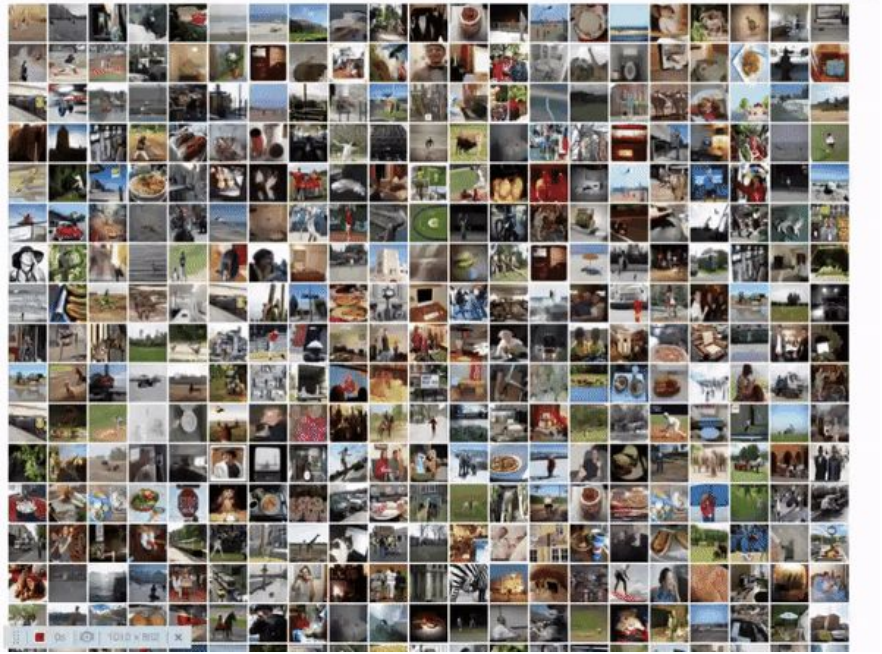
Organically explore and **look through data samples** of images

- General understanding
- Get a feeling of amount of **noise** or specific **cleaning steps require** (e.g. see specific ratio issues)
- Easy to get biased understanding! Can't go through a significant sample

Can be done with

- Matplotlib
- A dedicated tool like Google facet
- HTML rendering to visualize and explore in a notebook.

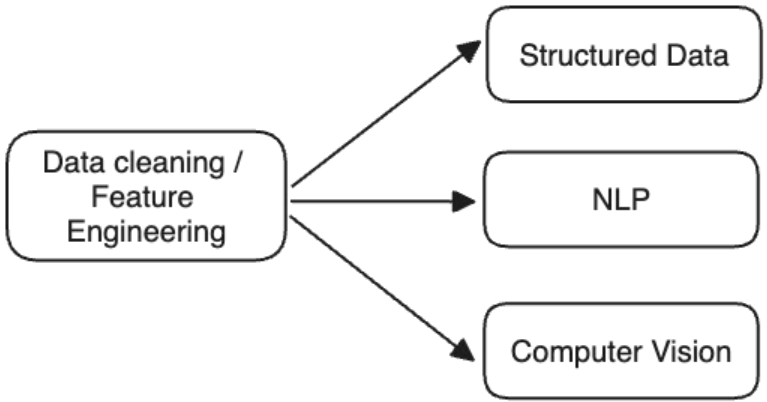
```
from easyimages import EasyImageList

Li = EasyImageList.from_folder('/Users/jcieslik/Downloads/val2017/')
Li.symlink_images()
Li.html(sample=500, size=44)
```



https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

Data cleaning & feature engineering

Different types of data on which feature engineering can be applied



Handling missing values

There are ***always*** missing values…

Question:

“I am selling shoes and I want to create a demand forecasting model (daily basis). I know that the weather will impact my in-shop sales.

I am buying weather data but sometimes we have missing values for a few hours. How would you handle it in my ML model? Assume it’s a model that cannot handle missing values.”

Handling missing values

1. **Deletion**: Remove data with missing values
   a. Column deletion - remove column with too many missing values
      - Drawback: Can remove useful information (sometimes missing point is info by itself)
   b. Row deletion - Remove rows with too many missing values
      - Can remove noisy observations
      - Drawback: Can lead to bias (is there a reason for this value consistently missing?)

ID	Age	Gender	Annual income	Marital status	Number of children	Job	Buy?
1		A	150,000		1	Engineer	No
2	27	B	50,000			Teacher	No
3		A	100,000	Married	2		Yes
4		B			2	Engineer	Yes
5	35	B		Single	0	Doctor	Yes
6		A	50,000		0	Teacher	No

Handling missing values

1. **Deletion**: Remove data with missing values
2. **Imputation**: Automatically fill in the value
   a. Default value (0, “Unknown”, empty string, …)
      i. No data does not necessarily always means no information!
   b. Statistical measure
      i. Mean, median, mode
      ii. Fill-forward (time series)
      iii. Same last period (week, year, …)

Detecting outliers

Detection options:

- **Heuristics** (e.g. the average temperature must be lower than 100 degrees)
- **Statistical rules** (e.g. feature value must be within 2 standard deviations)
- **Models** (can use information from other features! e.g. Isolation Forest or One-class SVM)

```
# Ex. Feature value must be within 2 standard deviations
df[np.abs(df.A - df.A.mean()) <= (2 * df.A.std())]
```

Considerations**:**

- Be careful not to remove **valuable outliers** (e.g. fraud detection)
- Remove outliers before other transformations (e.g. normalisation)
- Anomalies can be **global** (point), **contextual** (conditional) or **collective** (individual points are not anomalous and the collective group is an outlier)
    - ⇒ Make your outlier detection system dependent on the type of anomalies you're tackling!

Scaling

- Apply a formula to all values of your feature to scale it within a specific range.
- Can help *scale* features with varying ranges in a more constrained one
- It is sometimes harder for ML models to compare features operating in a small or huge range
- Important to **understand your data** and apply the right scaling technique

LIÈGE université

Scaling

1. Min/max scaling

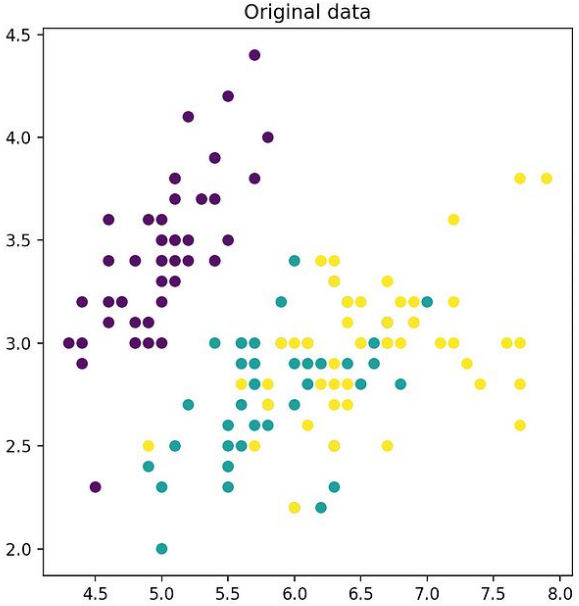
- You scale all values for variable X between 0 and 1
- Can be applied to **any kind of variable**
- Using the following formula:

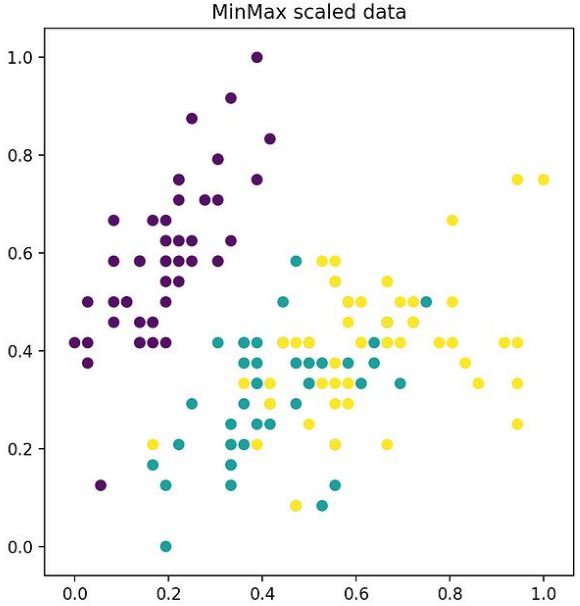
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Typically helpful for variables that need to maintain a the proportions but within a smaller range
- Example: Scaling *pixel values* in an image. Go from a [0-255] range to a [0-1] range while keeping the proportion, making computations more stable and facilitating learning in neural networks.

Scaling

1. Min/max scaling





LIÈGE université

Scaling

2. Z-score

- Scale feature so it has the properties of a normal distribution with a **mean of 0** and a **standard deviation of 1**.
- The feature itself should **already follow a Gaussian distribution**
- Using the following formula:

$$x_{scaled} = \frac{x - mean}{sd}$$

- Typically helpful when trying to compare scores or measurements between different units (test scores, heights, weights, …)
- Example: Several features are students scores for different tests. Some are out of 10, 20 or 100.

Scaling

3. Log

- **Non-linear** transformation using the **logarithm function**
- Mostly used to reduce the possible effect of **extreme values**. Long tailed distribution.
- By simply applying a logarithm you reduce the range without significantly impacting the relationship

$$X_{\log} = \log(X + c)$$

- Typically used for **skewed data** to reduce the disparity of extreme values
- Examples: Economics data (personal income) or demographics (population size)

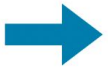
Encoding categorical features

Ordinal encoding

(aka Label encoding)

- Each finite category gets assigned an integer value
- There needs to be a ranking in the values, an ordinal relationship

Breakfast
Every day
Never
Rarely
Most days
Never



Breakfast
3
0
1
2
0

E.g.

- Education level: high school, bachelor, Master, PhD, …
- Income level: less than 50K, 50K-100K, over 100K, …
- Satisfaction rating: extremely dislike, dislike, neutral, like, extremely like

https://wandb.ai/wandb_fc/kaggle_tutorials/reports/Handling-Categorical-Features-With-Examples--VmlldzoyMTY4NDgz

Encoding categorical features

One-hot-encoding

Create new columns indicating the presence (or absence) of each possible value in the original data.

- No relation between categories needed
- Great for decision tree models

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

https://wandb.ai/wandb_fc/kaggle_tutorials/reports/Handling-Categorical-Features-With-Examples--VmlldzoyMTY4NDgz

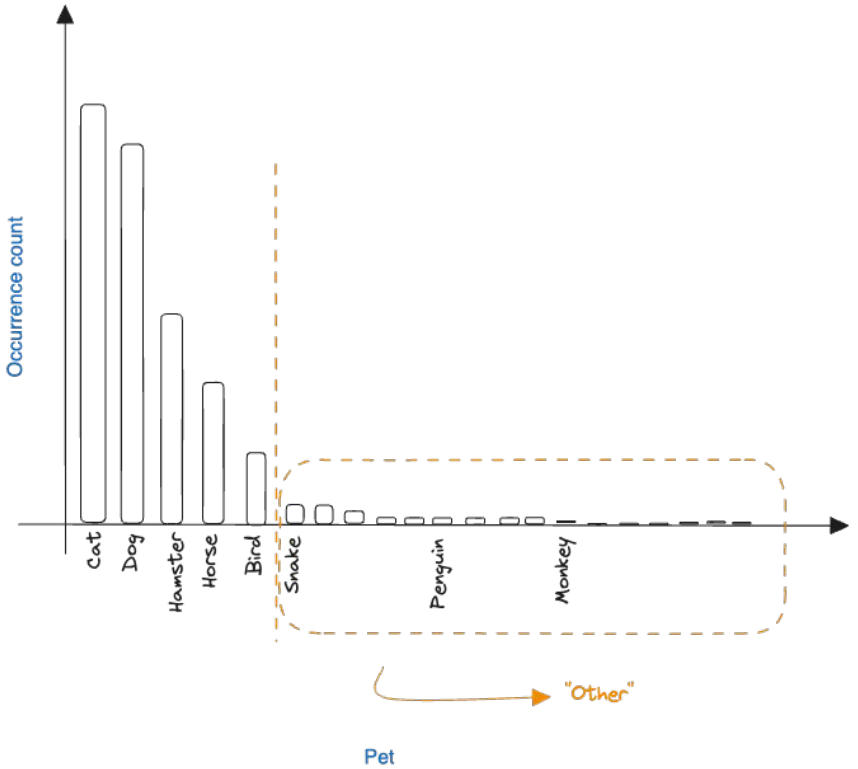
Encoding categorical features

One-hot-encoding

Struggles with **unseen values** and **long tail** distributions

⇒ Create an “Other” category

- Possible loss of information though…



Encoding categorical features

One-hot-encoding

Struggles with **u**

⇒ Create an “O

- Pos

Pet

Encoding categorical features

```python
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder


categorical_feature = np.asarray([['blue'], ['green'], ['blue'], ['red'],
['green']])
ordinal_encoder = OrdinalEncoder()
one_hot_encoder = OneHotEncoder(sparse=False)

ordinal_feature = ordinal_encoder.fit_transform(categorical_feature)
one_hot_feature = one_hot_encoder.fit_transform(categorical_feature)

# Result:
print(ordinal_feature)
# [[0.]
#  [1.]
#  [0.]
#  [2.]
#  [1.]]
print(one_hot_feature)
# [[1. 0. 0.]
#  [0. 1. 0.]
#  [1. 0. 0.]
#  [0. 0. 1.]
#  [0. 1. 0.]]
```


https://scikit-learn.org/stable/modules/preprocessing.html

Other best practices with feature engineering

- If you **oversample** your data, do it **after splitting**
- Only use **train** data statistics for: scaling, normalizing, handling missing values, …
  - Allows you to do objective testing.
- Keep track of **data lineage** (trace clearly how data flows from source to consumption).
- Understand the **feature importance** of your model.
- Measure **correlation** between features and labels.
- Use features that **generalize** well.

Natural Language Processing

For text, we can apply some **data cleaning**. These techniques are less useful in the era of transformers/LLMs !

- Removing Noise: Removing dataset specific noise (HTML tags, punctuation, special characters, …)
- Lowercasing: All text to lowercase
- Stop Words Removal: Removing common words that usually don't carry much meaning (e.g."and", "the", "a", …)
- Stemming and Lemmatization: Reducing words to their base or root form (e.g. "running" → "run")

Natural Language Processing

Here we can also apply **feature engineering**, though once again it is less relevant in the era of transformers/LLMs.

- Bag of Words (BoW): Represent a text as a vector of occurrences of each words in it.
- Term Frequency-Inverse Document Frequency (TF-IDF): Similar but occurrences are weighted per rarity of words.
- N-grams: Creating combinations of adjacent words of length 'n' to capture context and word order to some degree.
- Word Embeddings: Semantic representation of words based on the predicting likelihood of other words around them. Words close to each other in the vector space will have similar semantics. Popular models such as Word2Vec, GloVe, or FastText.
- Part-of-Speech (POS) Tagging: Identifying the part of speech (noun, verb, adjective, etc.) for each word. This can help in understanding the role of each word in a sentence and can be used as features.

Computer Vision

Standard computer vision data **cleaning steps**:

- Grayscale Conversion: Converting images to grayscale to reduce complexity for certain tasks, as color may not always be necessary.
- Normalization: Scaling pixel values to a certain range, e.g., [0,1] or [-1,1], for consistent model input.
- Noise Reduction: Applying filters (e.g., Gaussian blur) to smooth images and reduce noise.
- Contrast Adjustment: Enhancing the contrast of an image to make features more distinguishable.
- Image Resizing: Scaling images to a uniform size to fit the input requirements of a model or to reduce computational load.

Usually uses raw data. The feature creation is **embedded in the deep learning model**.

There are less examples of statistical models for feature representation.

Cloud infrastructure

Wrap-up

Cloud & credits

What & why

Throughout this course, you will get to use Google Cloud.

- Demos / Directed Work
- Group project
  - You will need to deploy different parts of your application. You are **free to select any Cloud or server provider**.
  - We will provide Google Cloud credits (50$ per student).

We will send out an email with a link to redeem those credits.

Next week we will do a **demo** of Google Cloud. We will focus on Google Cloud Storage and BigQuery. We recommend you to already login to the Google Cloud Console to be able to follow along. (follow the link coming by email)

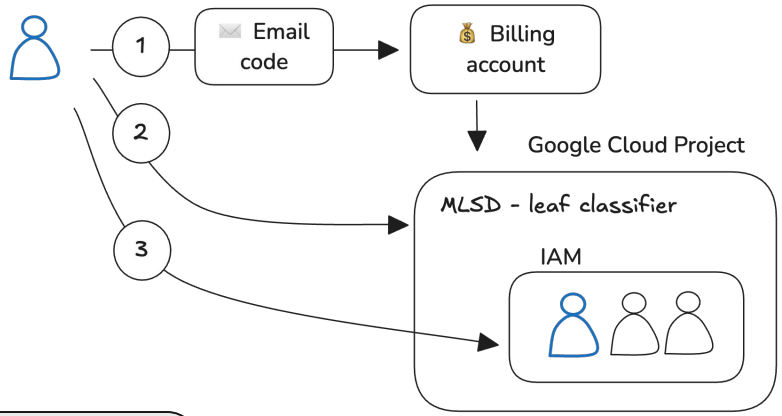
Cloud & credits

How to activate credits and manage projects

Only **one student** per group needs to create a project and activate the credits. If you run out of credits you can use the ones of someone else on your team.

Steps to setting up a project

1. Click on the link received by email to redeem credits and attach them to a billing account
2. Create a Google Cloud Project and link it to the billing account you created
3. Grant access to the other members of your team to this project through the IAM portal



⚠️ Next week we will do a tour of google cloud and we will demo how to setup a google cloud project!
No action needed from your side till then.

LIÈGE université

Project objective for sprint 2

This course focuses on what comes **around** your ML model.
Do **not** spend significant time optimising your data or model.
No grading on the **accuracy** of the model itself.

#	Week	Work package	Requirement
2.1	W03	Prepare your data and run an Exploratory Data Analysis.	Required
2.2	W03	Prepare your Cloud environment. That means creating a Cloud project, granting correct access rights to all members of your group and setting up a billing account. **Attention**: You can have free credits for the Cloud, as explained during the course.	Required
2.3	W04	Train your ML model	Required
2.4	W04	Evaluate your ML model	Required
2.5	W03 & W04	Document your data analysis and model performance	Required

Lecture summary

Topic	Concepts	To know for…	
		Project	Exam
Data formats	• Data format (row-major vs column-major; text vs binary)		Yes
Data models	• Data models (relational, NoSQL, document, graph)		Yes
Databases	• OnLine Transactional Processing • OnLine Analytical Processing		Yes
Exploratory Data Analysis	• Overall dataset sanity and noise • Univariate profiling • Multivariate profiling • EDA for Natural Language Processing and Computer Vision	Yes	
Data cleaning & feature engineering	• Data cleaning & feature engineering techniques for Structured data, Computer Vision and Natural Language Processing		Yes
Cloud infrastructure	• Introduction to the Cloud	Yes	

That’s it for today!

LIÈGE université