

Data & Cloud infrastructure

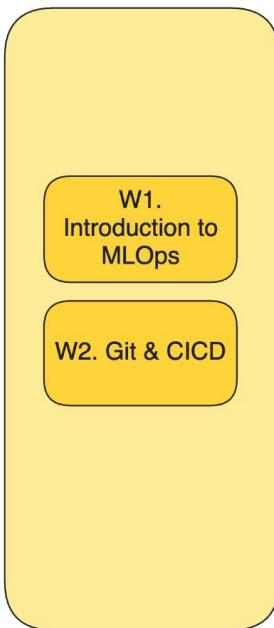
Sprint 2 | Week 3

INFO9023 - ML Systems Design

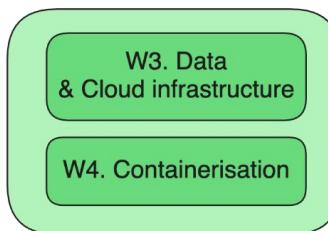
Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap

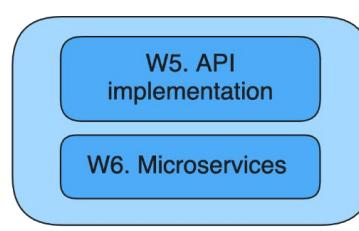
Sprint 1:
Project organisation



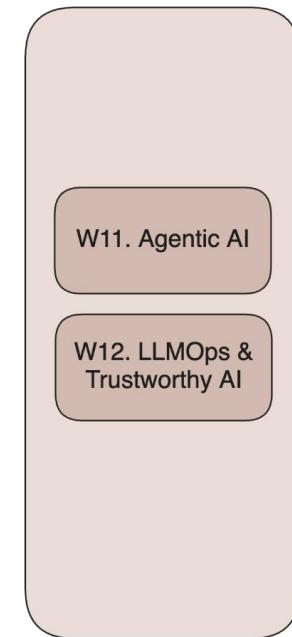
Sprint 2:
Cloud & containerisation



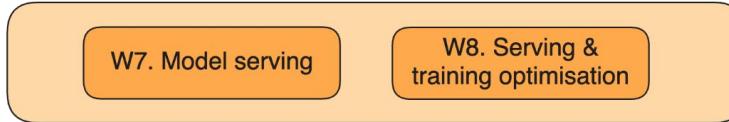
Sprint 3:
API implementation



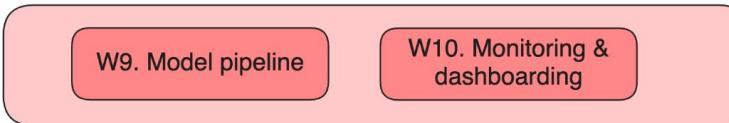
Sprint 6:
LLMOps



Sprint 4: Model serving & optimisation



Sprint 5: Pipeline & monitoring





Always has been.

It's all about data...





Agenda

What will we talk about today

Lecture

1. Data formats
2. Data models
3. Workload types
4. Extract, Transform and Load (ETL)

Guest lecture

5. Cloud infrastructure

Directed work

6. Cloud & data

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 milestones

- **MS1:** You will pick a spot between the **2nd and the 5th of March**. The exact spot available will be announced later.
- **MS2:** You will pick a spot between the **30th of March and the 2nd of April**. The exact spot available will be announced later.
- **MS3:** Each group will present in the classroom, other groups can attend the presentation. It will be on the **11th of May**.



Data formats

Data formats

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

Types:

1. **Text format:** Data is encoded as readable characters (ASCII/UTF-8).
 - *Properties:* Humane readable, easy to debug/interpret, larger size, slower to parse...
 - *Examples:* CSV, JSON, ...
2. **Binary format:** Data is encoded directly as raw bytes, not as readable characters.
 - *Properties:* More compact, less storage space, not readable, often optimise for specific workloads...
 - *Examples:* Parquet, pickle, ...
3. **Binary primary format:** *Protocols* designed from the ground up for binary serialisation and data exchange. They are not “text formats converted to binary”
 - *Properties:* Schema based, strong typing, used for streaming or service-to-service systems...
 - *Examples:* Protobuf, avro, ...

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

Data formats

Row-major

Column-major

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

Data formats

Row-major
Column-major

| Format | Binary/Text | Human readable | Use cases | Example |
|----------|----------------|----------------|-------------------------------|---|
| JSON | Text | Yes | Everywhere | { 'foo': [1,2,3, ...]} |
| CSV | Text | Yes | Everywhere | name,age\nAlice,30 |
| Parquet | Binary | No | Amazon Redshift | (Binary column chunks – not readable in editor) |
| Avro | Binary primary | No | Hadoop | Schema: {"type":"record","fields":[{"name":"age","type":"int"}]} |
| Protobuf | Binary primary | No | TensorFlow (TFRecord) | proto\nclass User { int32 age = 1; }\n |
| Pickle | Binary | No | Python, PyTorch serialization | b'\x80\x04\x95...' |

Row-major vs column-major

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 $\Rightarrow [1, 2, 3, 4, 5, 6]$

Column-major

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

$A = [[1, 2, 3], [4, 5, 6]]$
Stored $\Rightarrow [1, 4, 2, 5, 3, 6]$

| 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

```
# 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)
```

Data models

Data models

Data model: Describes the *representation* of data. Defines the structure, organisation and relationship of the data. It ensures it is retrieved efficiently.

Data models we will cover:

1. Relational
2. Document
3. Key-value
4. Graph
5. Wide-column

1. Relational model

- Persistent idea in computer science, introduced by Edgar F. Codd in 1970.
- Data is organized into **relations** (tables) where each relation is a set of **tuples** (rows)
- Rigid **schema**

Table = Relation

Row = Tuple

| Column 1 | Column 2 | Column 3 | ... |
|----------|----------|----------|-----|
| Sample 1 | | | |
| Sample 2 | | | |
| Sample 3 | | | |
| ... | | | |

Normalisation

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 |



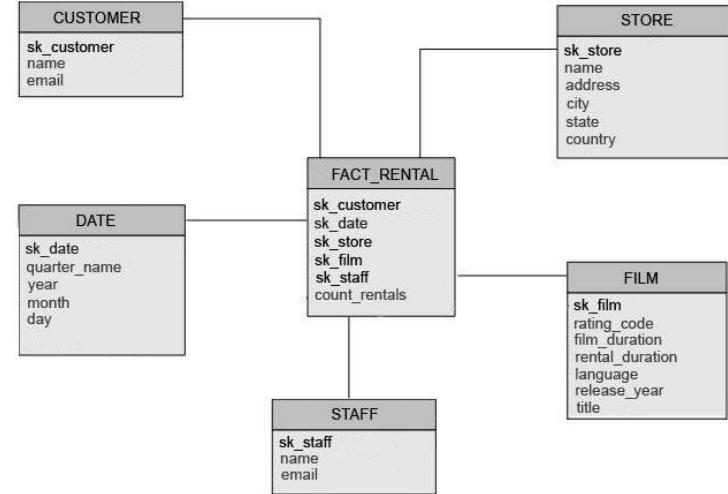
| Title | Author | Format | Publisher ID | Price |
|-----------------|----------------|-----------|--------------|-------|
| Harry Potter | J.K. Rowling | Paperback | 1 | \$20 |
| Harry Potter | J.K. Rowling | E-book | 1 | \$10 |
| Sherlock Holmes | Conan Doyle | Paperback | 2 | \$30 |
| The Hobbit |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

1. **Fact tables** | The “center”
 - Contains quantitative data, the “business transactions or events”(e.g., sales amount, revenue)
 - And foreign keys linking to dimension tables
2. **Dimension tables** | The “leafs”
 - Contains attribute and descriptive text for the facts (e.g., product category, customer demographics)

Splitting your data between fact and dimension tables is called **normalisation**. It reduces redundancy for tables with high cardinality.



Forms of normalisation

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)

Forms of normalisation

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)



Covered in this course

"[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."

Forms of normalisation

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 |

Forms of normalisation

First normal form (1NF)

First normal form: each field contains a single value. A field may not contain a set of values or a nested record.

Book

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

Title - Subject

| <u>Title</u> | <u>Subject name</u> |
|--|---------------------|
| Beginning MySQL Database Design and Optimization | MySQL |
| Beginning MySQL Database Design and Optimization | Database |
| Beginning MySQL Database Design and Optimization | Design |

Forms of normalisation

Second normal form (2NF)

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 | |

Forms of normalisation

Second normal form (2NF)

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:

- 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 |

Forms of normalisation

Third normal form (3NF)

Third normal form: no transitive functional dependencies between attributes.

Transitive functional dependency: 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 |

Forms of normalisation

Third normal form (3NF)

Satisfies 3NF ! 

Book

| <u>Title</u> | <u>Author</u> | <u>Pages</u> | <u>Thickness</u> | <u>Publisher</u> | <u>Genre ID</u> |
|---|---------------|--------------|------------------|------------------|-----------------|
| 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

| <u>Title</u> | <u>Format</u> | <u>Price</u> |
|---|---------------|--------------|
| 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

| <u>Author</u> | <u>Author Nationality</u> |
|---------------|---------------------------|
| Chad Russell | American |
| E.F.Codd | British |

Publisher

| <u>Publisher</u> | <u>Country</u> |
|------------------|----------------|
| Apress | USA |
| Addison-Wesley | USA |

Genre

| <u>Genre ID</u> | <u>Name</u> |
|-----------------|-----------------|
| 1 | Tutorial |
| 2 | Popular science |

Forms of normalisation

Pros & Cons

| Pros | Cons |
|---|---|
| <ul style="list-style-type: none">• Removes redundancy - <u>memory</u> efficient• Consistency - standardised values• Easy to update a value throughout all occurrences• More security flexibility by restricting access to certain information | <ul style="list-style-type: none">• 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 

2. Document model

Document model: Organizes data in a *semi-structured* format like JSON or BSON.

- Unlike relational models, there is no fixed schema. It is more flexible!
- Each observation (“document”) includes both the data and the metadata (keys)
- Often documents are represented in text format (JSON, XML, ...)
- Observations can be retrieved through declarative queries on non-key values
 - Often with “SQL like” languages
- Complex operations (e.g. joins) are less efficient than for relational databases

```
● ● ●

SELECT
    id, date_time, user_type, prediction
FROM
    `my_project.my_dataset.my_documents`
WHERE
    publication_year BETWEEN '1960' AND '1980'
    AND publisher != 'lorem ipsum';
```

```
[{"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."}]
```

2. Document model

| 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., user information, chatbot messages, ...) | Query performance can degrade if documents grow too large |

3. Graph model



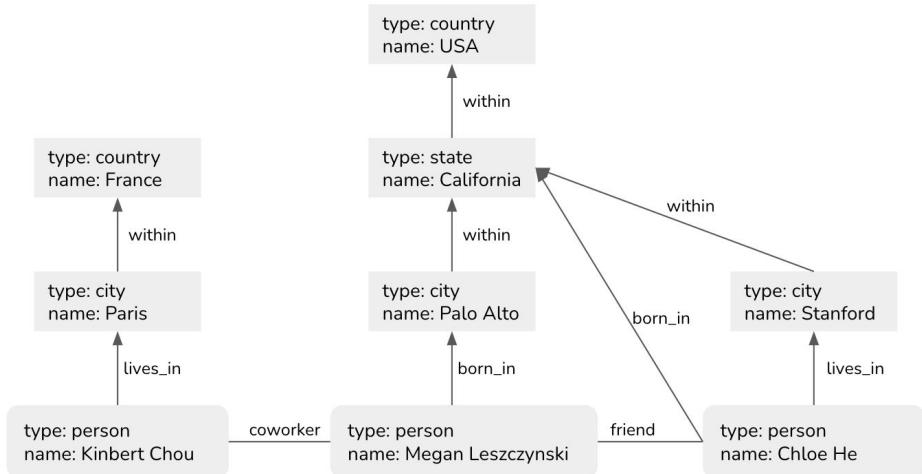
A graph consists of **nodes** and **edges**.

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.



4. Key-value model

Key-value: 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 “value” of each data point is opaque to the database. You cannot query against it.

It is extremely fast at querying against keys.

| Pros | Cons |
|---|--|
| Extremely fast read/write operations when querying on keys ($O(1)$ 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 |

5. Wide-column model

Wide-column: It stores data in column families rather than rows. Often described as a two-dimensional key-value map.

Designed to handle petabytes of data across thousands of machines with high write throughput.

Often used for Internet of Things (IoT) or large time series data.

| Pros | Cons |
|---|--|
| Massive scalability, can scale horizontally across thousands of commodity servers | No complex querying - typically doesn't support complex "joins" or multi-table relationships |
| High write throughput, ideal for high velocity streaming devices | Mostly efficient on <i>primary</i> keys. The columns used for querying (look-ups) should be set at the beginning |
| Performance for aggregation due to the data being stored by column | Cost and maintenance complexity |

Workload types

Workload types

Data model: Describes the *representation* of data. Defines the structure, organisation and relationship of the data. It ensures it is retrieved efficiently.

Workload types: Describes the *operational patterns* of a system. Defines whether a database is optimised for high-volume individual *transactions* or complex *analytical* processing.

Two workload types considered:

- OnLine Transaction Processing (OLTP)
- OnLine Analytical Processing (OLAP)

OnLine Transaction Processing (OLTP)

OnLine Transaction Processing (OLTP): Perform high volumes of low complexity *transactions*, which can be a sequence of operations such as insert, update, delete, or retrieve.

Brings *low latency* and *high availability*.

OLTP database perform **ACID transactions**:

- **Atomicity:** Every single operation in a transaction must succeed, otherwise none are applied (“all-or-nothing” rule).
- **Consistency:** All the transactions coming through must follow predefined rules. For example, a transaction must be made by a valid user or follow specific data types.
- **Isolation:** Concurrent transactions execute as if they were sequential, preventing interference.
- **Durability:** Once a transaction is “committed” (saved), its changes persist even in the event of a system failure or power outage.

OnLine Analytical Processing (OLAP)

OnLine Analytical Processing (OLAP): Perform complex *analytical* operations over a large amount of rows. Compute a lower volume of heavier jobs that require to perform calculations over data.

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?*
- ...

Optimised for *complex aggregation queries*. *High throughput*, designed to scan millions or billions of row to generate insights. Usually *column major*.

In ML, primarily used for *offline training* and *batch feature engineering* where historical data is aggregated.

Workload types

| Workload type | OnLine Transaction Processing (OLTP) | OnLine Analytical Processing (OLAP) |
|----------------|---|---|
| Goal | Fast updates & lookups. ACID transactions | Deep analysis & aggregation |
| ML Use | Real-time <i>serving</i> (fetch 1 user) | Offline <i>training</i> (scan all users, produce features through aggregations) |
| Storage | Usually Row-major | Often Column-major |
| Models | Relational, Document, Key-Value | Relational, Wide-Column |

Optimising a relational database.

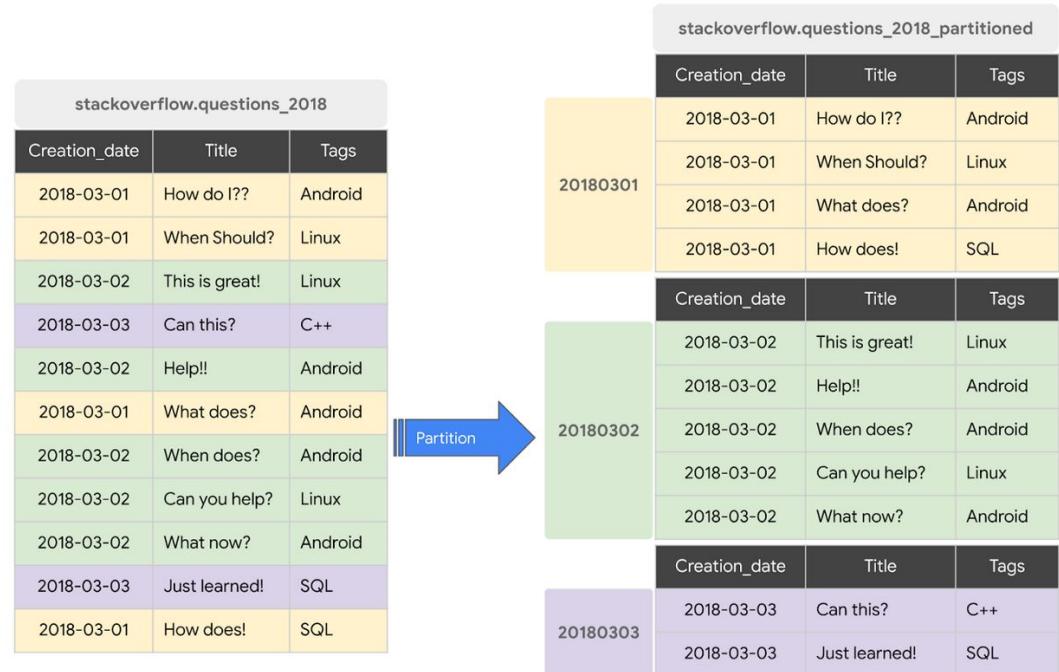
- **Indexing:** Data structure applied to one (or multiple) key columns to organise and speed the data retrieval operations applied on this column. Avoids searching the entire column for specific values. A downside of it is a higher memory consumption.
- **Query optimisation:** Selecting only required 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.

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

Minimize the amount of data that slot workers read from disk, typically for column majors.



Decision flow

4. Visualizing the Decision Flow

You might consider adding a slide with a simplified "Decision Tree" to help students choose a database for their ML System:

1. **Is your data highly relational with complex links?** \rightarrow **Graph or Relational**.
2. **Do you need to store massive features for real-time ML serving?** \rightarrow **Key-Value** (Memorystore) or **Wide-Column** (Bigtable).
3. **Are you performing "Offline" training on billions of rows?** \rightarrow **OLAP** (BigQuery).
+1
4. **Is your data format changing rapidly (unstructured)?** \rightarrow **Document** (Firestore).
+1

Example database services

Relational model

OLTP databases



OLAP databases



Example database services

Other models

Document model



Wide-column model



Cloud Bigtable
(Google Cloud)



Apache
Cassandra
(Self-managed)

Graph model



Key-value model



MongoDB
(Self-managed)



Memorystore
(Google Cloud)



DynamoDB
(AWS)



Cache for Redis
(Azure)

Extract, Transform and Load (ETL)

Extract, Transform and Load (ETL)

Extract, Transform and Load (ETL): Process of extracting data from a source, transforming it into a desired format, and loading it into a destination system. How to “move” and “transform” data between two systems.

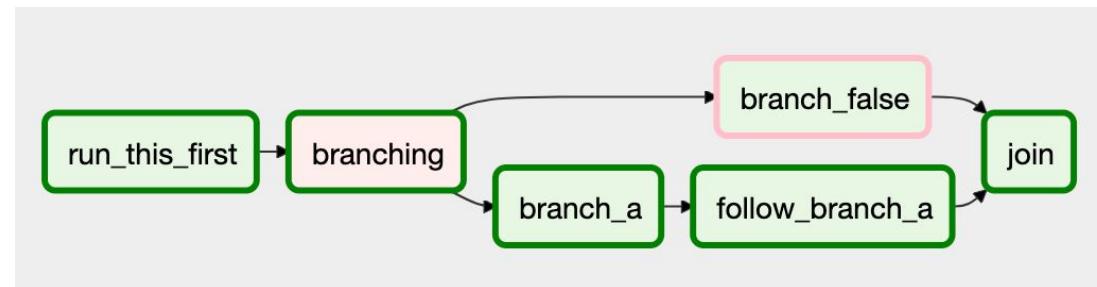
| Tool | Category | Description |
|----------------|------------------|--|
| Apache Airflow | Orchestration | An open-source tool used to schedule and monitor workflows as Directed Acyclic Graphs (DAGs). |
| Apache Spark | Compute (Batch) | A distributed computing engine designed for large-scale data processing using fast, in-memory computation. |
| Apache Beam | Compute (Stream) | A unified programming model for defining both Batch and Streaming data processing pipelines. |

Airflow



An open-source platform used to schedule, monitor, and manage complex workflows using *Directed Acyclic Graphs (DAGs)*

- Workflows can be defined in *Python code*, allowing for dynamic pipeline generation and version control
- Easily scheduled or triggered
- Integrates well with cloud services and modern data stacks



Spark



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

Mostly used for *batch* heavy jobs.

Key features

- In-memory computing → Data is kept in the system's active memory (RAM). Faster execution than disk-based systems.
- Lazy evaluation → Delays the execution of an operation until its result is explicitly required by an "Action" (like `.count()` or `.show()`). Optimized execution plans for efficiency.

ML Use Case: Processing a years' worth of raw transaction logs in Cloud Storage to create a unified feature table for offline training.



Google Cloud
Dataproc

Beam



Unified programming model for defining both Batch and Streaming data processing pipelines

Flexible data pipeline, that allows the same code to handle historical data (Batch) and real-time events (Streaming).

- Abstracts away the underlying execution engine, allowing you to focus on the data logic rather than the infrastructure.

ML Use Case: Real-time feature engineering, such as calculating a user's "average clicks in the last 10 minutes" as they browse your app.



Google Cloud
Dataflow

Cloud infrastructure

Wrap-up

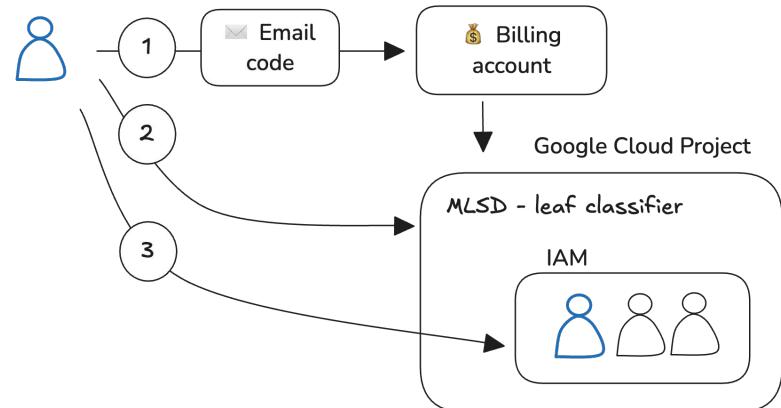
Cloud & credits

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](#)



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 |
|-----|------|--|
| 2.1 | W03 | Prepare your data and run an Exploratory Data Analysis (EDA). |
| 2.2 | W03 | Train and evaluate your ML model. |
| 2.3 | W03 | Make sure to <u>document</u> the result of your EDA and model evaluation (e.g. through slides or a markdown file in your “docs/” folder) |
| 2.4 | 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. |
| 2.5 | W03 | Store your data to the cloud and modify your training script so that it can automatically run with the data you stored on the cloud. Carefully select the database service you will use on the cloud, and document this choice. Different options might be more appropriate: <ul style="list-style-type: none">• Relational SQL database (BigQuery)• NoSQL database (firestore)• Blob storage (cloud storage) If data storage or the choice of database is not trivial for your use case, make sure to discuss it with the teaching staff. |

Lecture summary

| Topic | Concepts | To know for... | |
|-----------------------------------|--|----------------|------|
| | | Project | Exam |
| Data formats | <ul style="list-style-type: none">• Data format (row-major vs column-major; text vs binary) | | Yes |
| Data models | <ul style="list-style-type: none">• Data models (relational, document, key-value, graph, wide-column) | | Yes |
| Workload types | <ul style="list-style-type: none">• OnLine Transactional Processing (OLTP) and OnLine Analytical Processing (OLAP) | | Yes |
| Extract, Transform and Load (ETL) | <ul style="list-style-type: none">• Airflow, Spark and Beam | | Yes |
| Cloud infrastructure | <ul style="list-style-type: none">• Introduction to the Cloud | Yes | |

That's it for today!



Data is the key to a performant ML model



Garbage in
Garbage **OUT**

THIS IS YOUR MACHINE LEARNING SYSTEM?

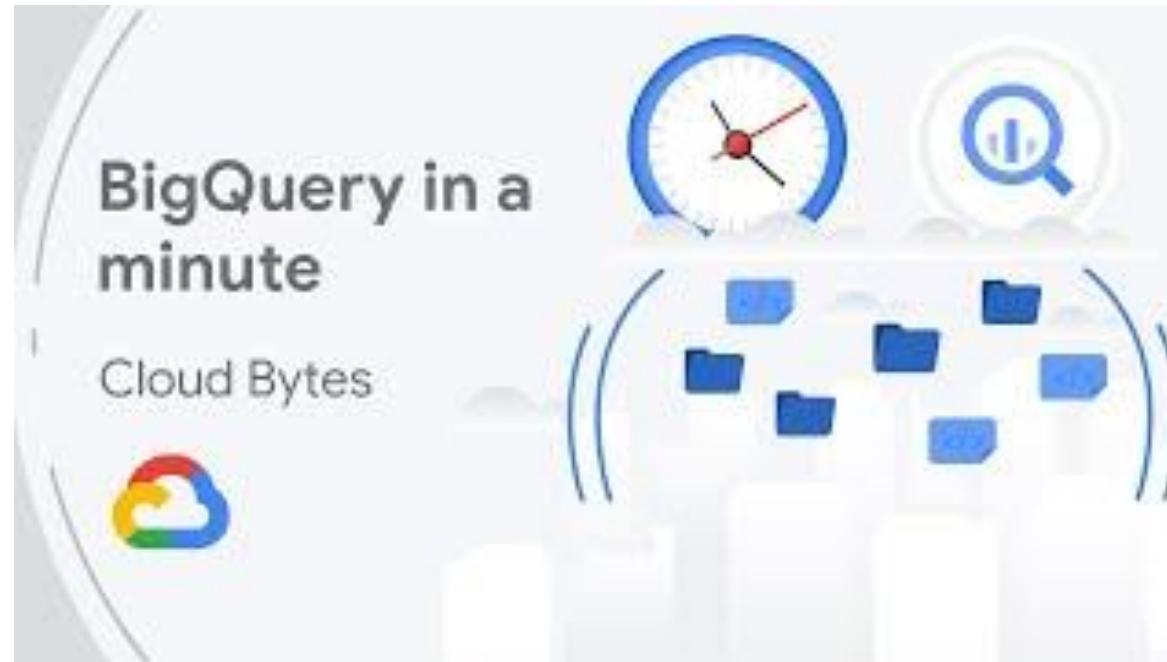
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.



Concrete database example: BigQuery



Popular databases

- **PostgreSQL** – Open-source, SQL-compliant, great for complex queries and scalability.
- **MySQL** – Popular relational database, widely used for web applications.
- **MongoDB** – NoSQL document database, ideal for flexible, schema-less data storage.
- **Redis** – In-memory key-value store, great for caching and real-time analytics.
- **Elasticsearch** – Search and analytics engine, optimized for full-text search.
- **Snowflake** – Cloud-based data warehouse, built for scalable analytics and BI.

**THE EASIEST WAY TO
IMPROVE MODEL ACCURACY...**

IS TO FOCUS ON THE DATA

imgflip.com

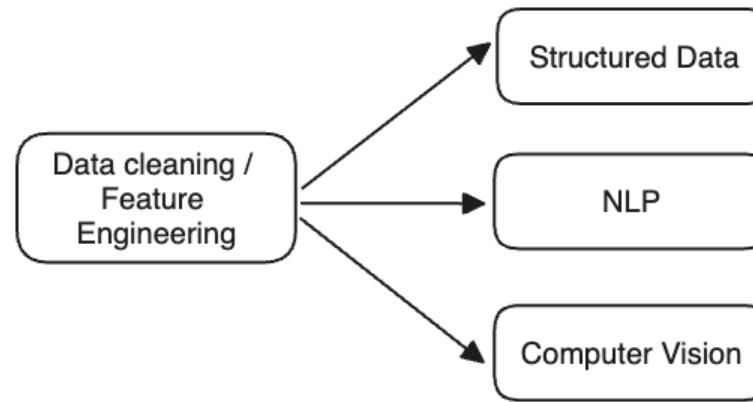
Relational data model

... the dawn of all civilised computer science...



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)

Considerations:

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

- 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



Scaling

1. Min/max scaling

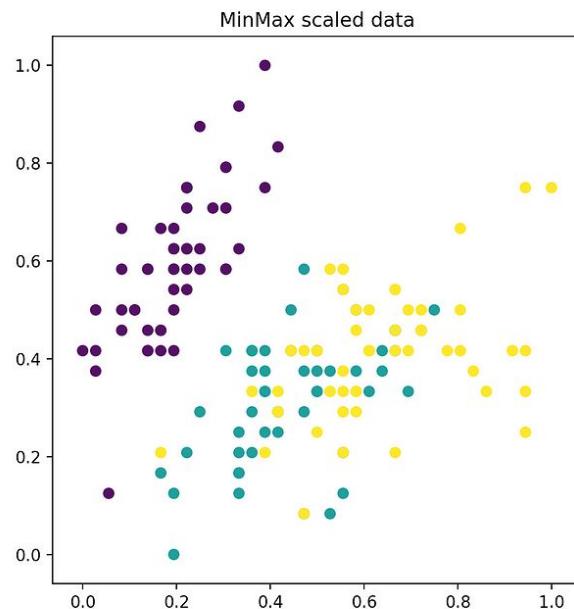
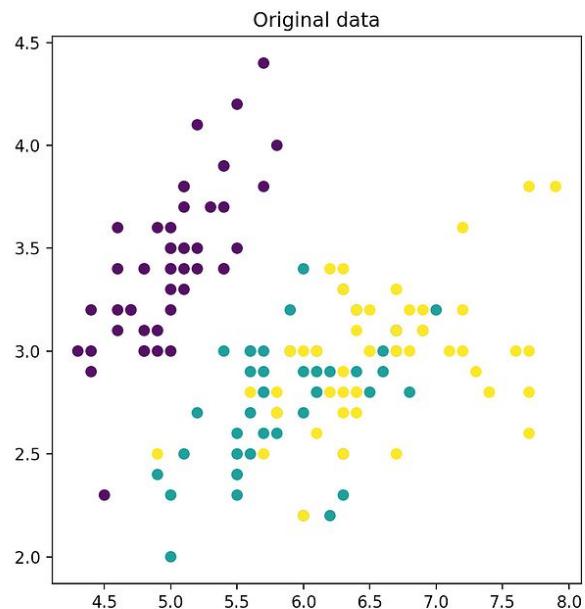
- You scale all values for variable X between 0 and 1
- Can be applied to **any kind of variable**
- Using the following formula:

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

- Typically helpful for variables that need to maintain 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



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

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

| Breakfast |
|-----------|
| Every day |
| Never |
| Rarely |
| Most days |
| Never |



| Breakfast |
|-----------|
| 3 |
| 0 |
| 1 |
| 2 |
| 0 |

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 | Yellow | Green |
|--------|-----|--------|-------|
| Red | 1 | 0 | 0 |
| Red | 1 | 0 | 0 |
| Yellow | 0 | 1 | 0 |
| Green | 0 | 0 | 1 |
| Yellow | 0 | 1 | 0 |

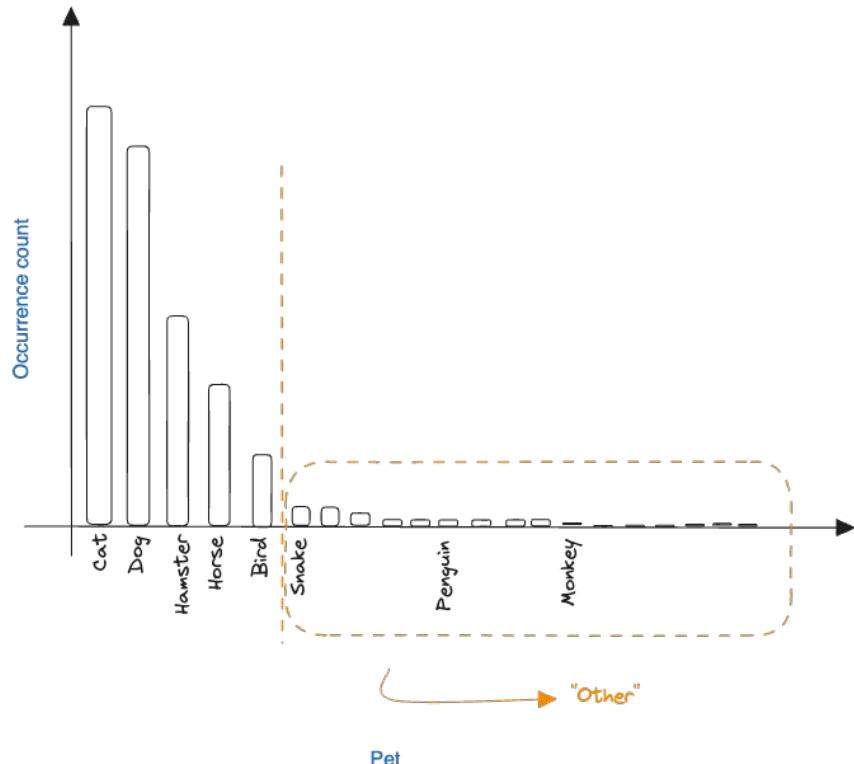
Encoding categorical features

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

```
● ● ●

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.]]
```

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.

2. Relational model | OLTP

Example database tools



Cloud SQL
(Google Cloud)



Amazon RDS
(AWS)



**Azure SQL
Database**
(Azure)

5. Wide-column model

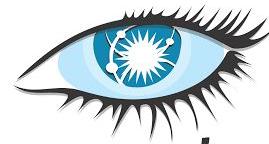
Example database services



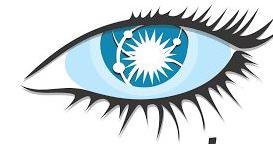
Cloud Bigtable
(Google Cloud)



Keyspaces
(AWS)



cassandra
CosmosDB
(Cassandra API)
(Azure)



Apache
Cassandra
(Self-managed)

4. Key-value model

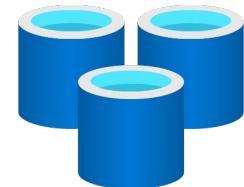
Example database services



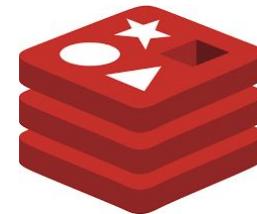
Memorystore
(Google Cloud)



DynamoDB
(AWS)



Cache for Redis
(Azure)



MongoDB
(Self-managed)

2. Document model

Example database services



Firestore
(Google Cloud)



DocumentDB
(AWS)



CosmosDB
(Azure)



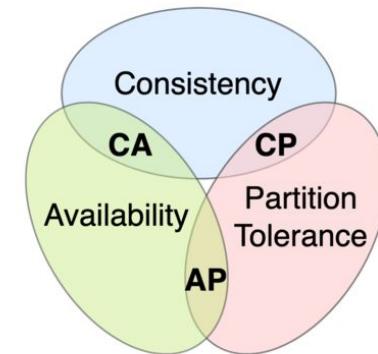
MongoDB
(Self-managed)

CAP theorem

CAP theorem (aka Brewer's theorem): 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.



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)



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?