Data preparation

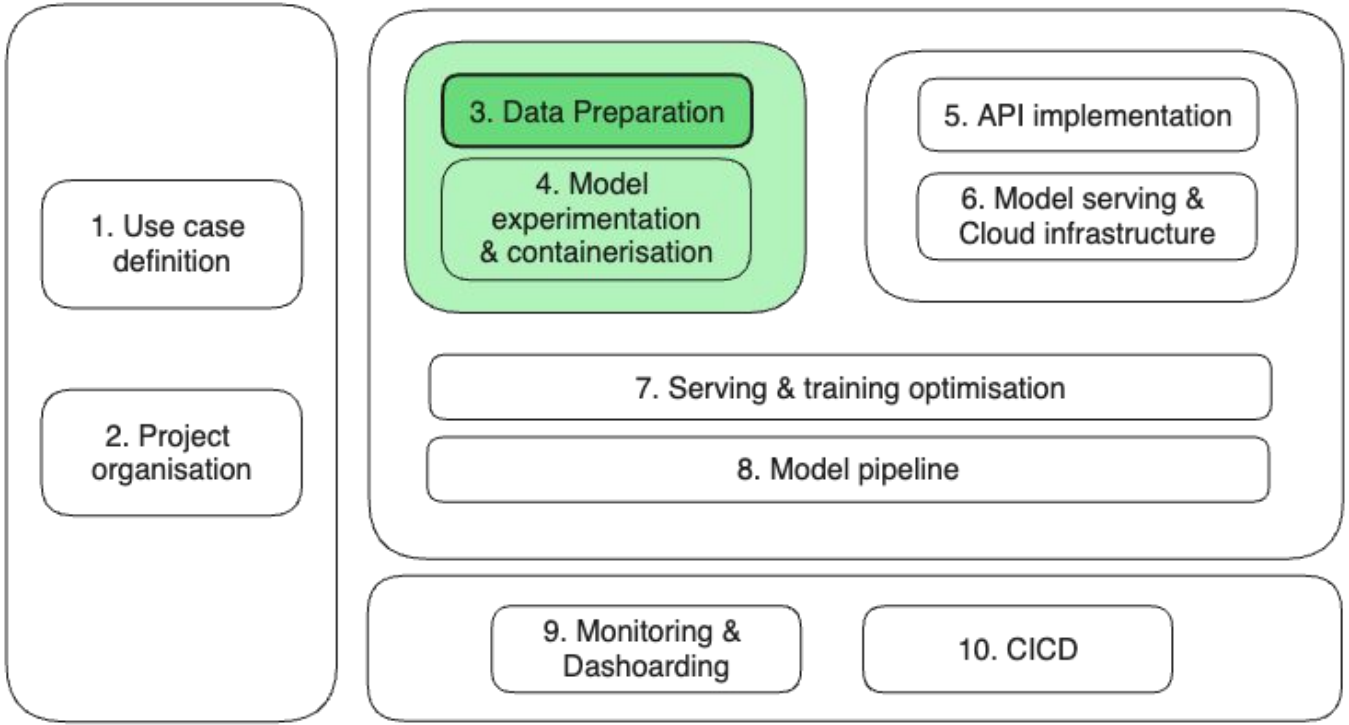
Sprint 2 - Week 3

INFO 9023 - Machine Learning Systems Design

2024 H1

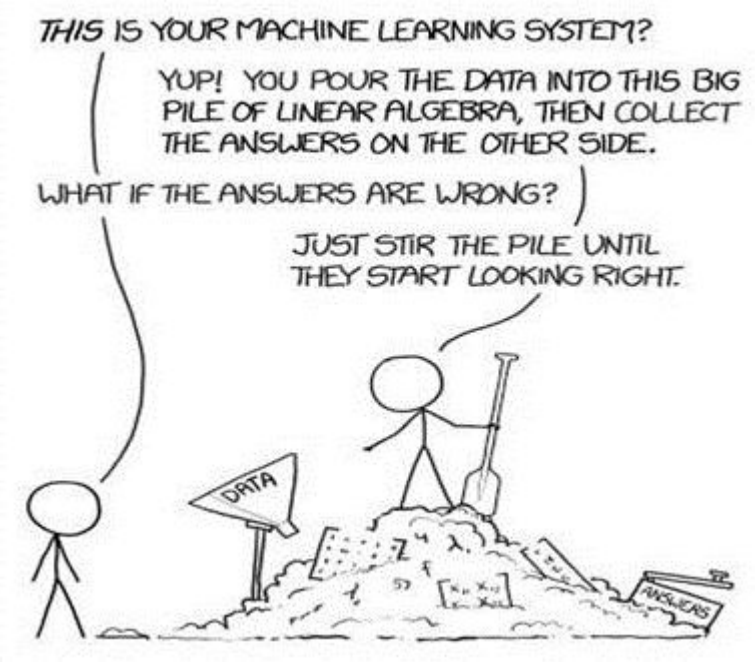
Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap




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

Data is the key to a performant ML model



Agenda

What will we talk about today

Lecture

1. Data formats & data models
2. Databases
3. Exploratory Data Analysis (EDA)

Lab

4. Ydata profiling

Lecture

5. Data cleaning & feature engineering

Overall data process

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

Data formats & data models

LIÈGE université

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	Hadoop, Amazon Redshift
	Avro	Binary primary	No	Hadoop
	Protobuf	Binary primary	No	Google, TensorFlow (TFRecord)
	Pickle	Binary	No	Python, PyTorch serialization

Row-major vs column-major

Column-major

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

Row-major

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

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 (but offers less abstraction - more manual coding work)
- 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

Data models

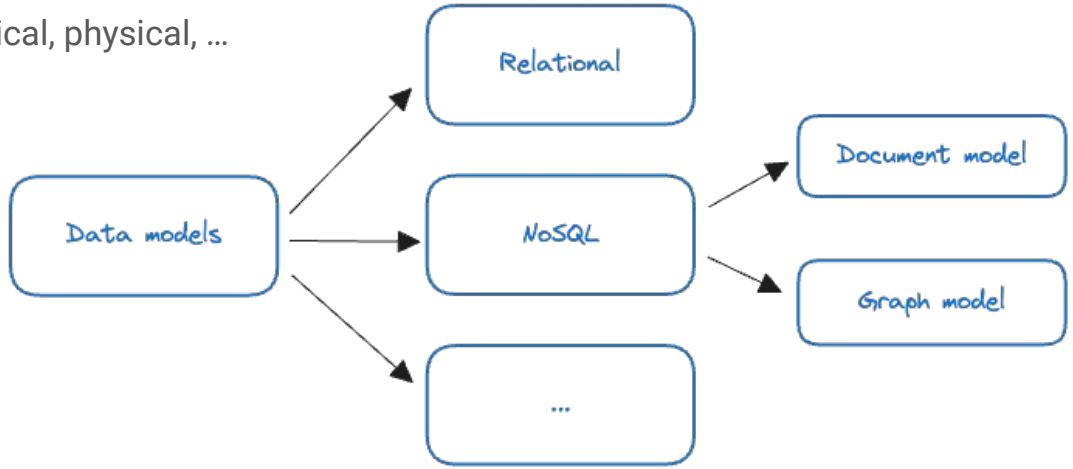
A **data model** describes how your data is represented.

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

It ensures that your data is managed and retrieved efficiently.

There are many models such as conceptual, logical, physical, …

But we will focus on **relational** and **NoSQL**.



Relational data model

- 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): unordered

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

Column: unordered

Relational data model

… the dawn of all civilised computer science…

Normalisation of Relational data model

Creating tables and relationships between those 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

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 operation • More complexity • More maintenance

SQL query language

- SQL is a **query language**, it allows you to query for example 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
```


More info in “INFO0009 Database” course

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 the query language of many databases! Important to learn 🧠

NoSQL

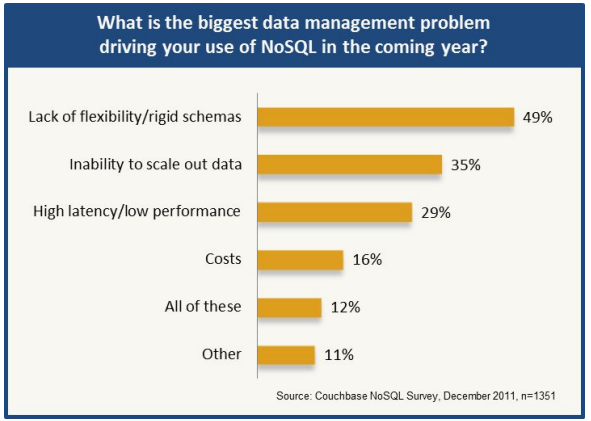
Not all data fit well to a relational database. E.g. It might not follow a strict schema.

We consider two examples of NoSQL data models:

1. **Document model**: targets use cases where data comes in self-contained documents and relationships between one document and another are rare.
2. **Graph model**: goes in the opposite direction, targeting use cases where relationships between data items are common and important.

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?

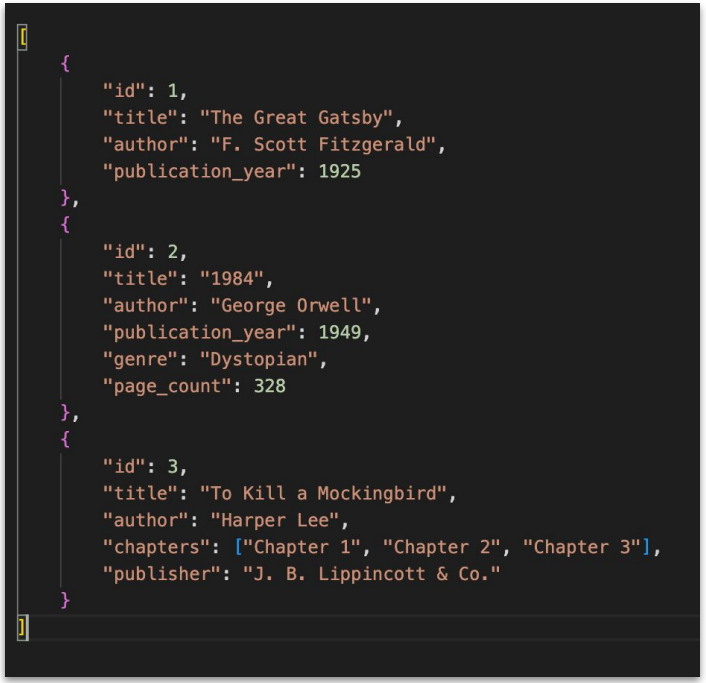
Surprise #3: Schema management is the #1 pain driving NoSQL adoption.



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

NoSQL: Document model

- Documents **don't have relationships**
- Often documents are represented as a **continuous string** (JSON, XML, …)
  - Counter example: MongoDB is a NoSQL that uses Binary-JSON (BSON) ⇒ binary-encoded serialization of JSON-like documents.
- Each document can be retrieved using their **unique key**
- Document models are typically **schema-less**
  - Sometimes the application feeding the database still follows a strict schema. MongoDB allows for a schema validation feature
- 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

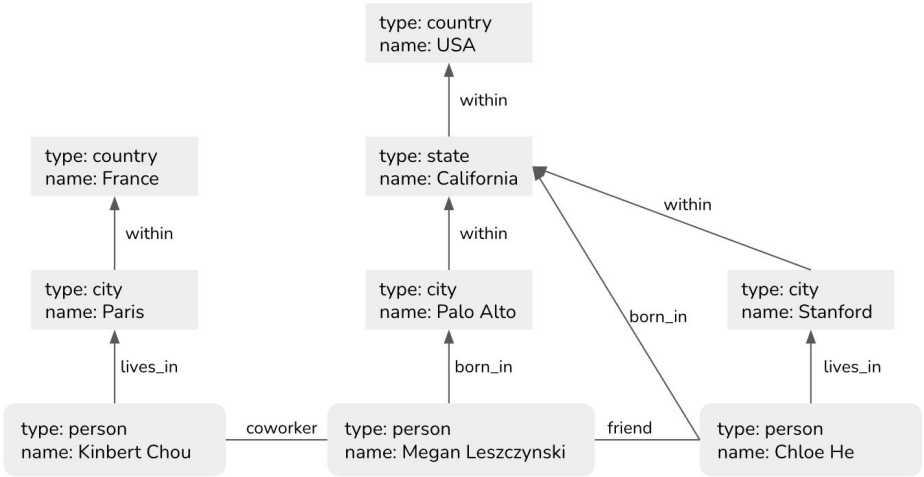


NoSQL: Graph model

A **graph** consists of **nodes** and **edges** describing the relationship between these nodes.

Fast to query based on relationships

- E.g. 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…



NoSQL

Pros & Cons

Pros	Cons
• **Scalability**: designed to scale out using distributed clusters • **Flexibility**: Not limited to structured (schematised) data • **Performance** for specific type of data retrieval such as graphs	• Lack of **standardisation** between different databases using NoSQL (different APIs and query languages) • Lack of **consistency**, often do not adhere to ACID • **Complexity** can sometimes occur by opposition to a leaner relational system

Databases

LIÈGE université

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 a bit about them as any ML application will be built on top of one.

We look at two types of databases (often relational but not a hard requirement):

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

Database: OnLine Transaction Processing (OLTP)

OLTP databases are meant to perform many **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**: All the steps in a transaction are completed successfully as a group. If any step between the transaction fails, all other steps must fail also. For example, if a user's payment fails, you don't want to still assign a driver to that user.
- **Consistency**: All the transactions coming through must follow predefined rules. For example, a transaction must be made by a valid user.
- **Isolation**: Two transactions happen at the same time as if they were isolated. Two users accessing the same data won't change it at the same time. For example, you don't want two users to book the same driver at the same time.
- **Durability**: Once a transaction has been committed, it will remain committed even in the case of a system failure. For example, after you've ordered a ride and your phone dies, you still want your ride to come.

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

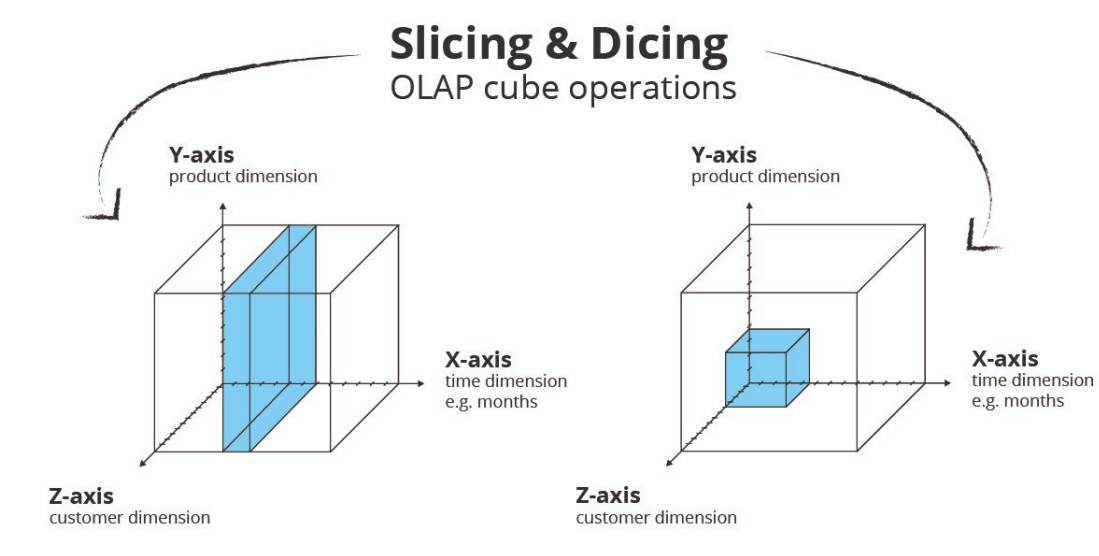
Database: OnLine Analytical Processing (OLAP)

Compute heavier jobs that require to aggregate data over multiple rows.

Used for **analytical** jobs. (E.g. What is the average purchase of the users of a specific age?)

Can be optimised when combining different dimensions or data sources.

Often used for storing and processing **historical** data.

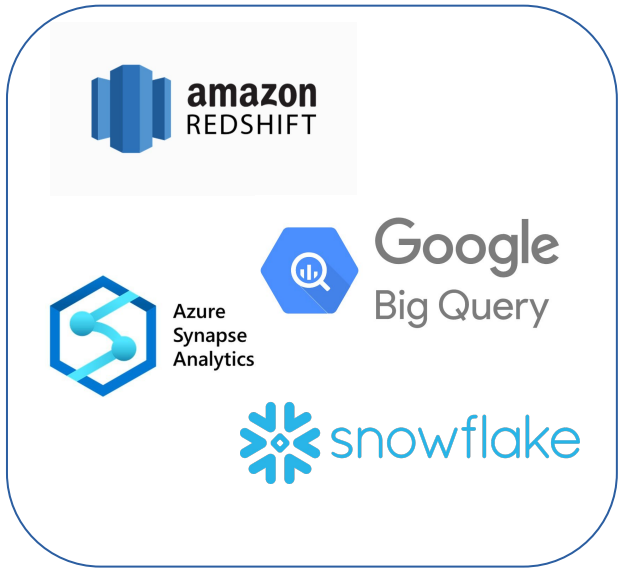


https://www.jedox.com/en

Concretely… OLTP and OLAP options

OLTP databases

OLAP databases



Concrete database example: BigQuery

Optimising a relational database.

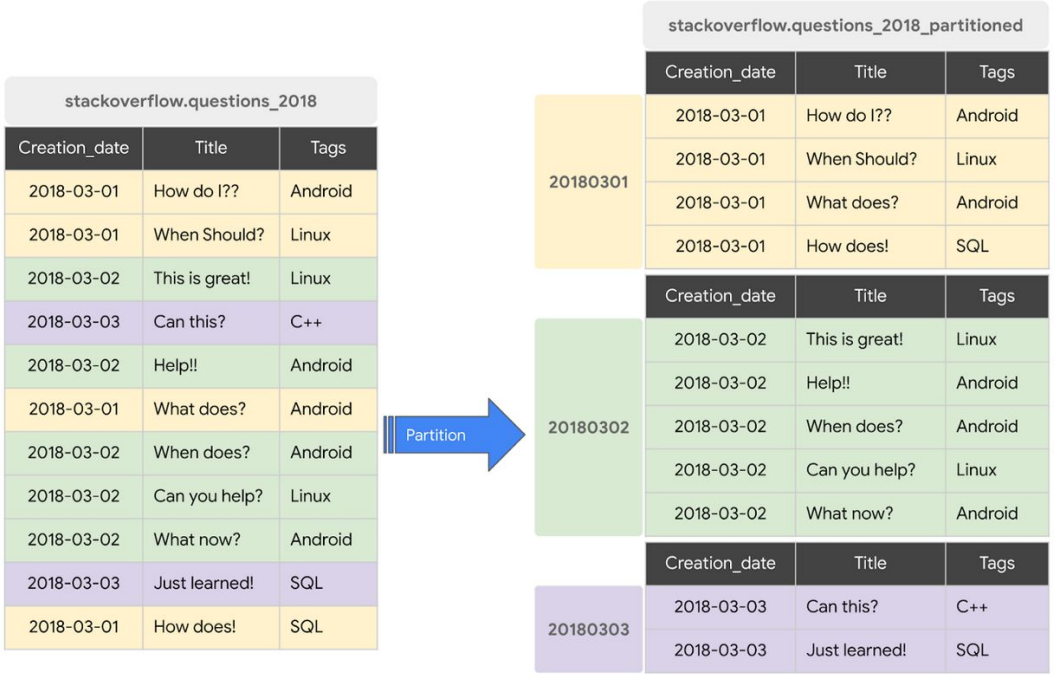
- **Indexing**: Data structure applied to one (or multiple) key columns to organise and speed the data retrieval operations applied on this column. Avoids searching the entire column for specific values. A downside of it is a higher memory consumption.
- **Query optimisation**: Selecting only required columns, using `WHERE` operations effectively. Know how the query is constructed and operated.
- **Caching**: Store frequently accessed 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).

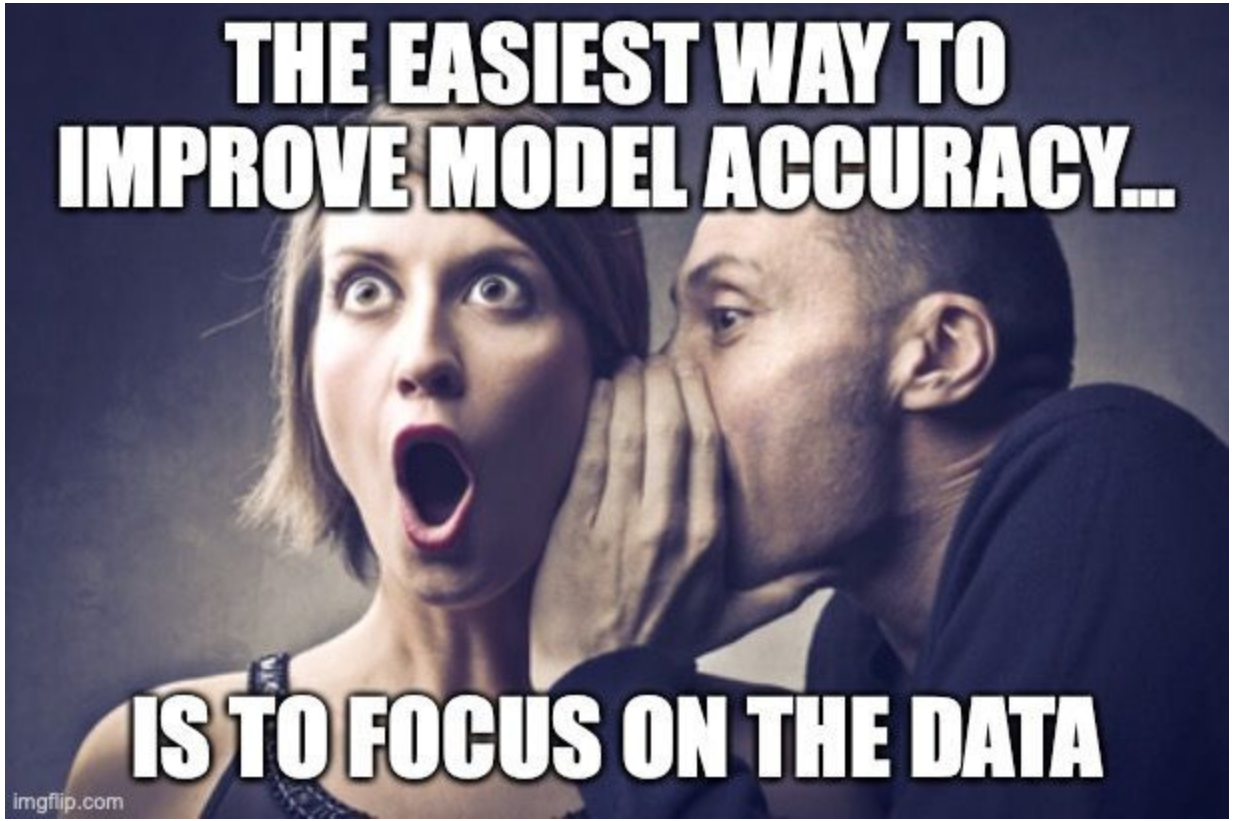
Decoupled storage and compute architecture leverages column-based partitioning simply to minimize the amount of data that slot workers read from disk.



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

Exploratory Data Analysis

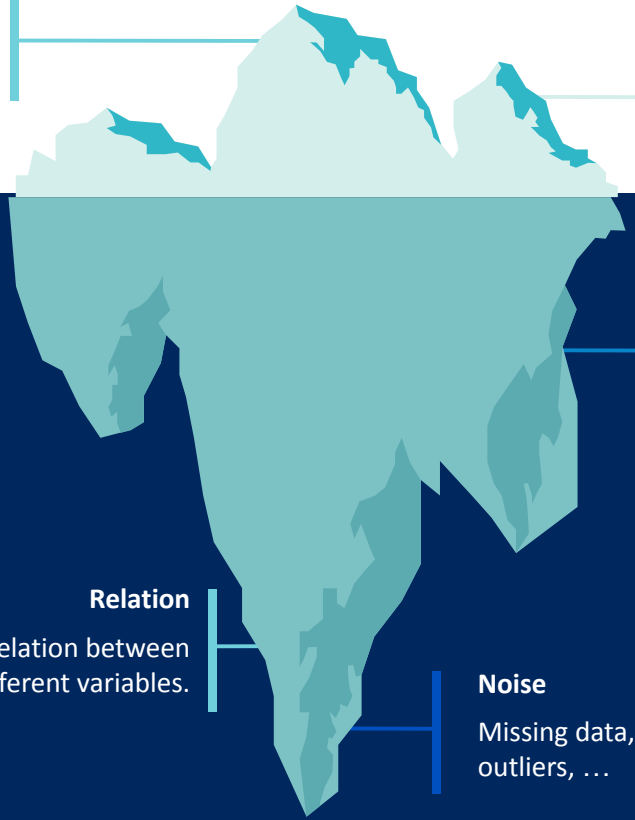
LIÈGE université


THE EASIEST WAY TO
IMPROVE MODEL ACCURACY...
IS TO FOCUS ON THE DATA
imgflip.com

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.

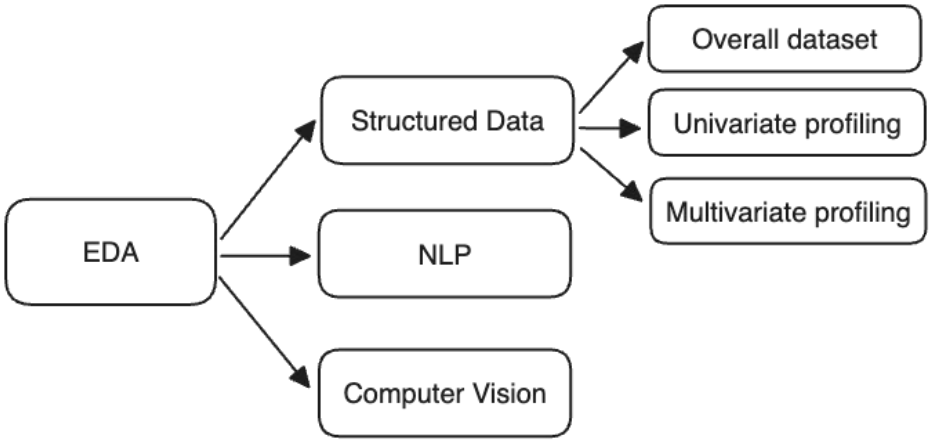
Relation

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

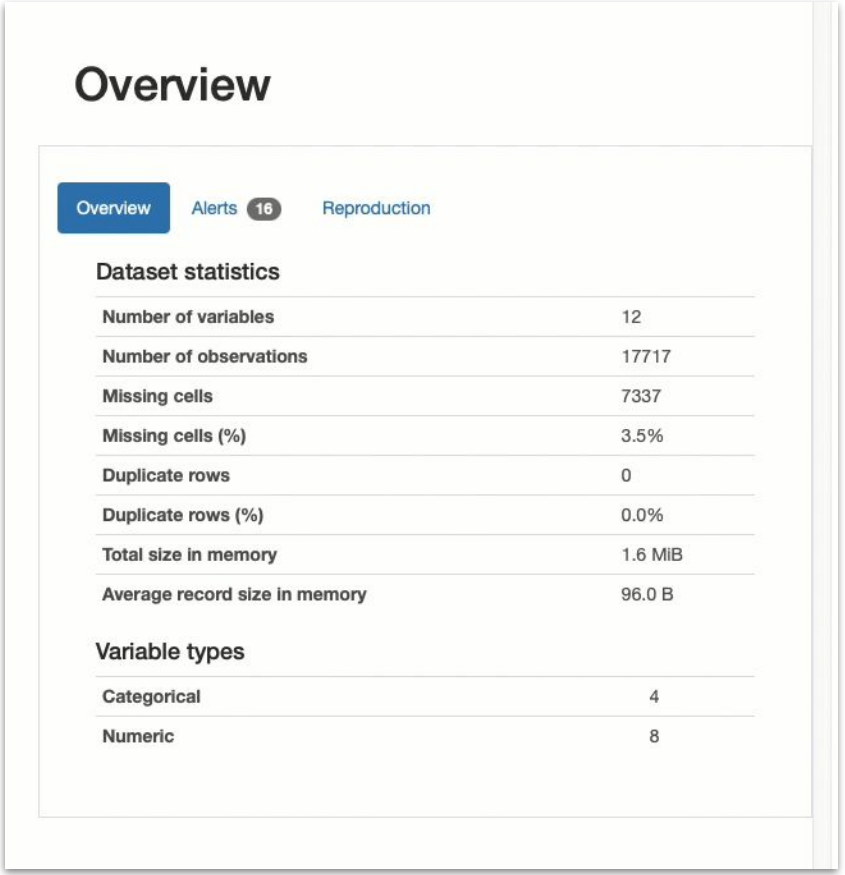
```
import pandas as pd
from pandas_profiling import ProfileReport


# 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

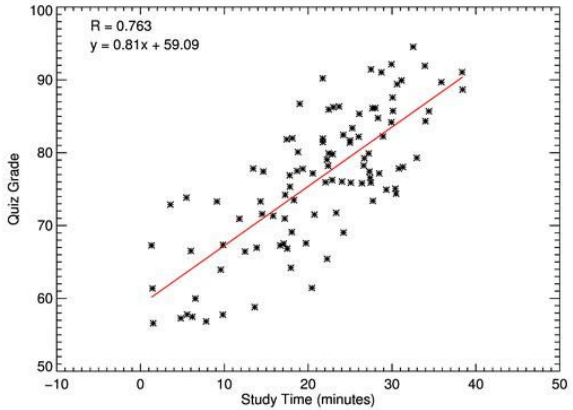
Univariate profiling

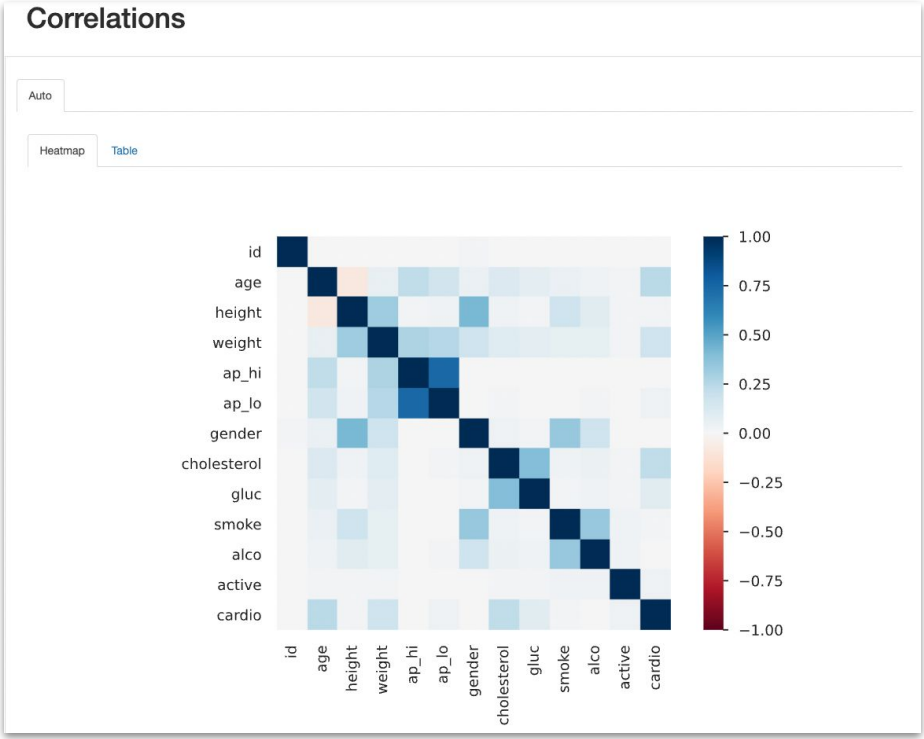
- Understanding **independent** variables
- You can use pandas **describe** and a **histogram** of value counts
- YData Profiling shows is in the **variable** tab

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**



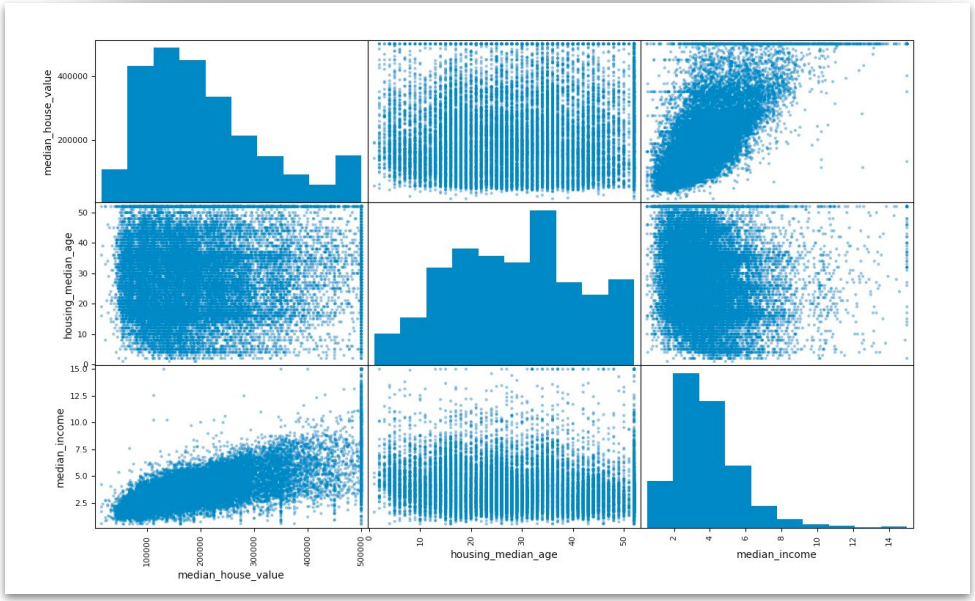


Ajibade, Samuel & Taiwo, Fele & Adedara,. (2016). A Systematic Review of Data Visualization Methods.

Multivariate profiling

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

```
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:

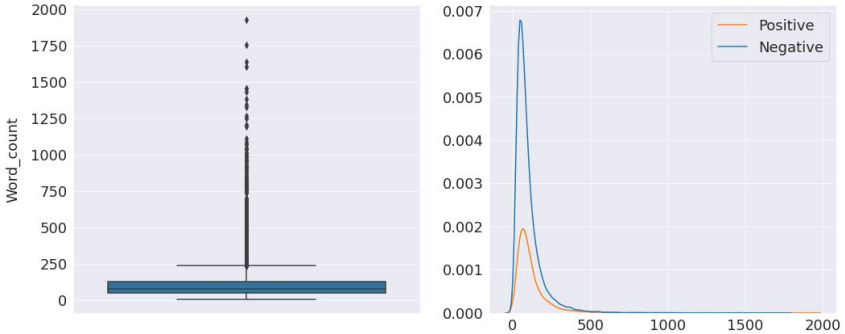
- Assume you work for a **publishing company** that has **300k news articles** from the last 30 years
- They want to build a **semantic search engine**
- The documents use domain specific language so you want to **fine-tune a text embedding model**
- For that, you need a labeled data set of **Question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

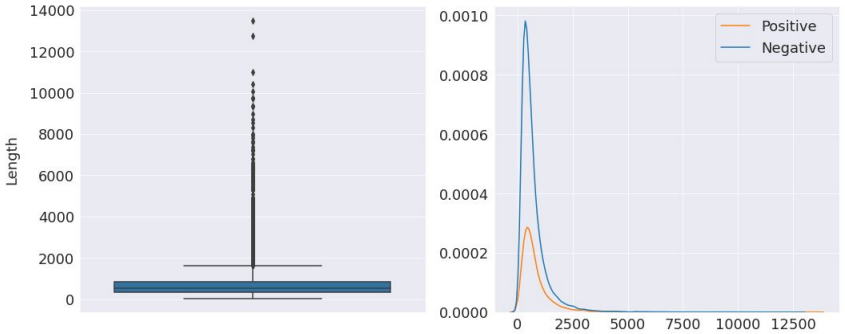
Open question:

- 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

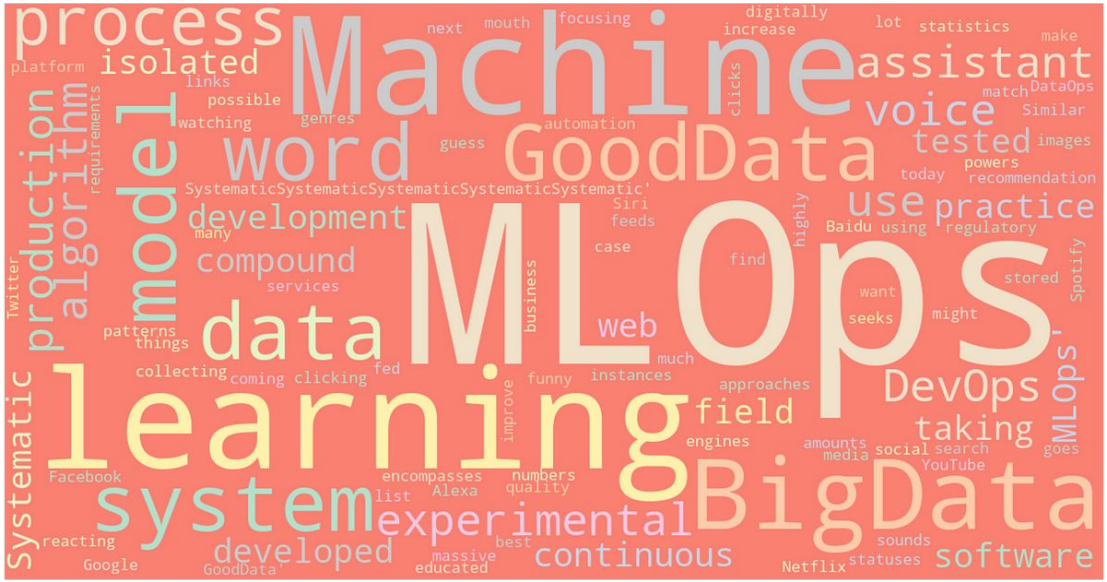
EDA for Natural Language Processing

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

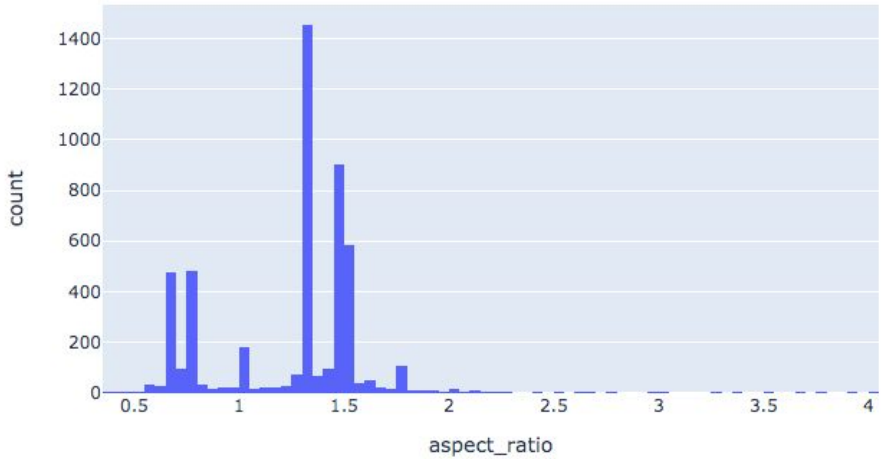
- Interesting to get a sense of topics
- In reality not super useful…



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

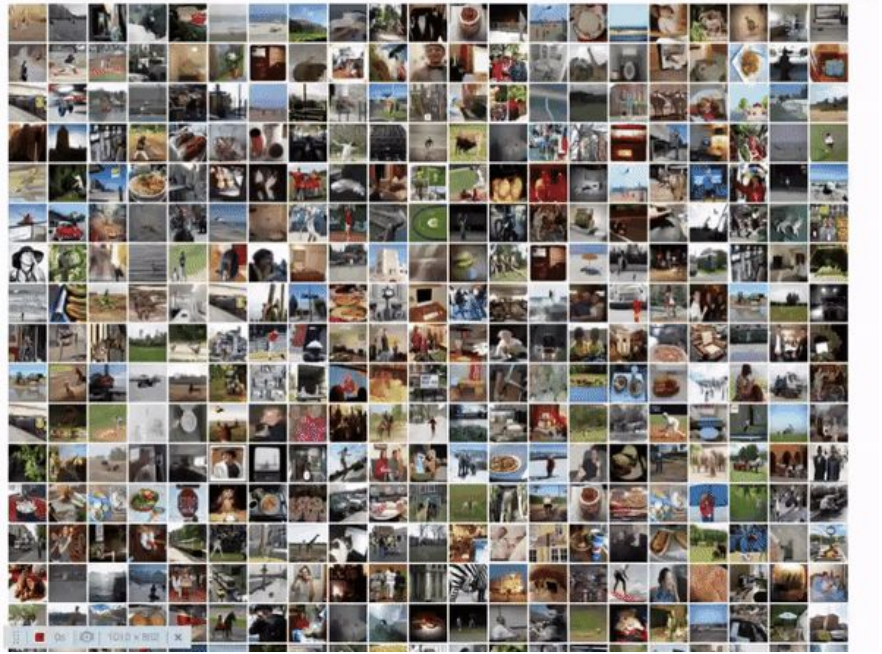
Organically explore and **look through data samples** of images

- 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

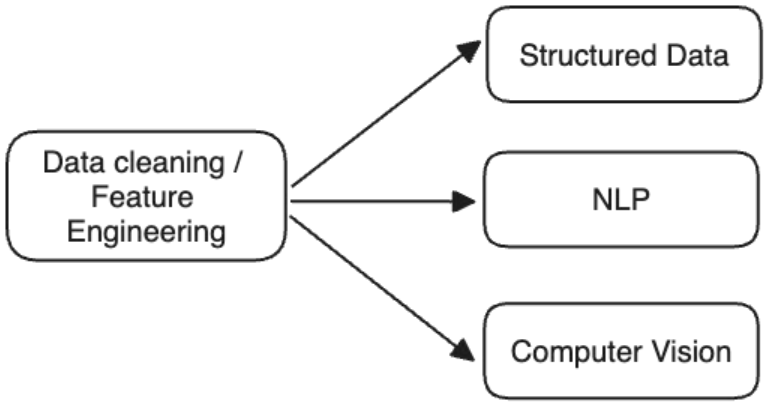
LAB: Ydata profiling

LIÈGE université

Data cleaning & feature engineering

LIÈGE université

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!

https://madewithml.com/courses/mlops/preprocessing/

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 normalization

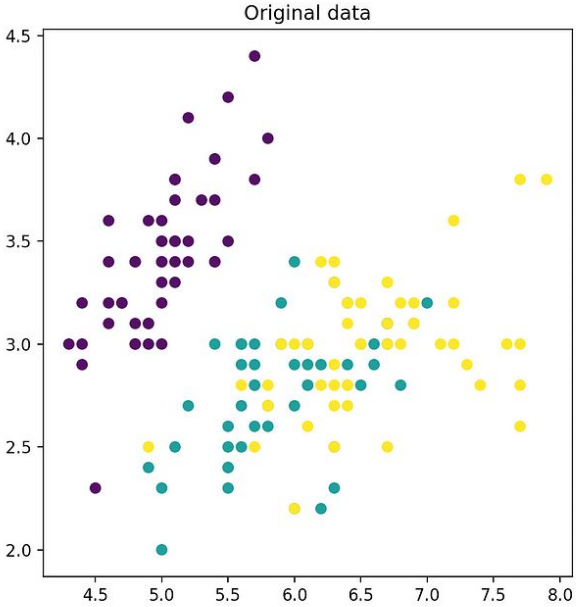
- You scale all values for variable X between 0 and 1
- Can be applied to **any kind of variable**
- Using the following formula:

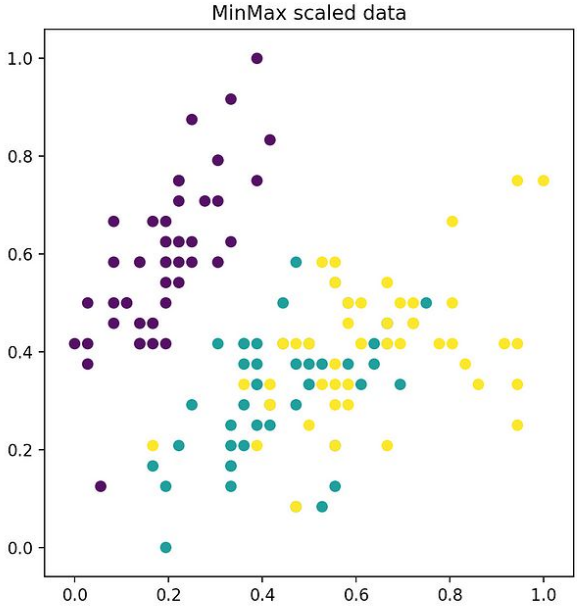
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- 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





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

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

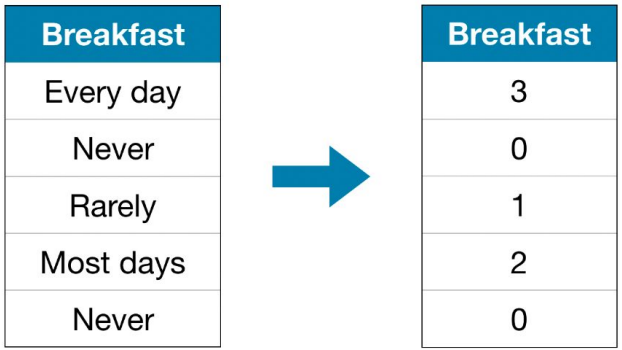
- Typically used for **skewed data** to reduce the disparity of extreme values
- Examples: Economics data (personal income) or demographics (population size)

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



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

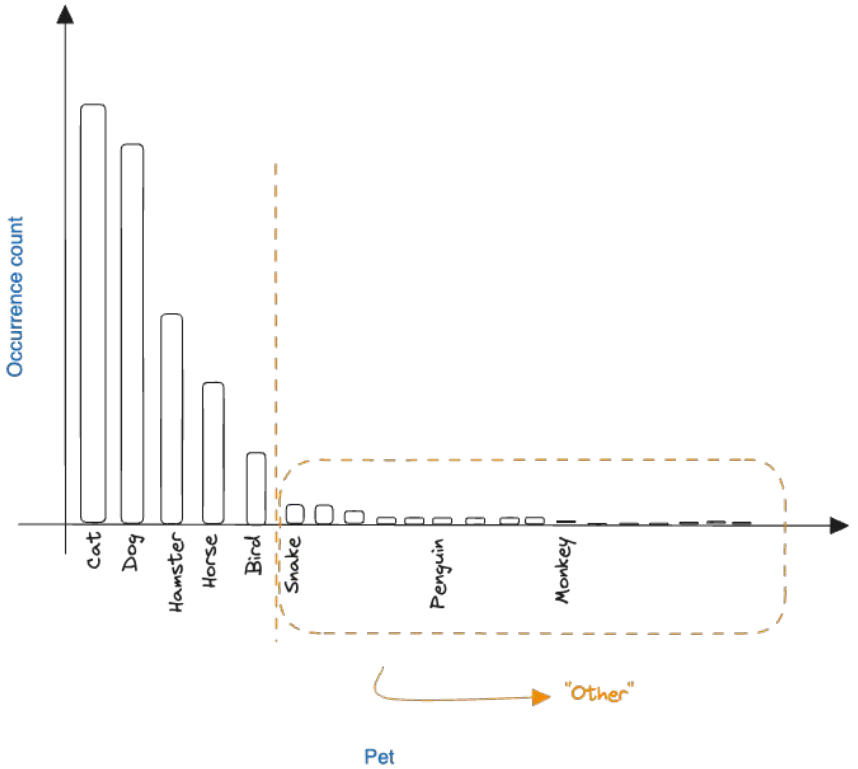
Encoding categorical features

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 **ur**

⇒ Create an “Oth

- 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.

CS 329S: Machine Learning Systems Design - Stanford (Chip Huey)
https://stanford-cs329s.github.io/syllabus.html

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.

Wrap-up

LIÈGE université

Lecture summary

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

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	W04	Train your ML model	Required
2.3	W04	Evaluate your ML model	Required
2.4	W04	Use W&B for step 2.2 - 2.3	Optional

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.
- The main purpose of the Milestones is to **provide feedback and guidance on the project**
  - The final project grade will be determined at the end of the project. If there are any issues highlighted during the milestone there is room for fixing it in the later development of 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 ask 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 be **online** and will take 15min (10min presentation + 5min QA)
- MS 1 will be between the 11th and 13th of March
- We will send a mail with a link to book a Google Meets slot (always between 13:00 and 16:30)

Cloud access

- In sprint 3 you will get to implement parts of your ML system in the Cloud
- You are free to use the Cloud of your preference - many will provide you free credits

- We contacted the main Cloud providers and can get you **35$** of **GCP Cloud credits** without having to enter any payment method
  - Let the teaching staff know if you require those credits and we will send you a specific voucher

LIÈGE université

APPENDIX

LIÈGE université

Trend to use more and more data to train ML models

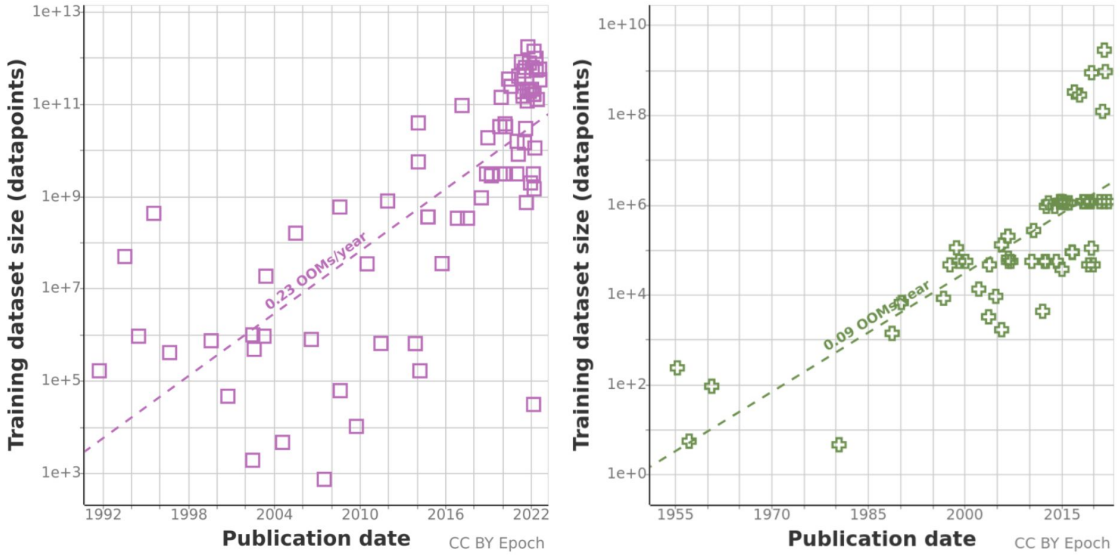
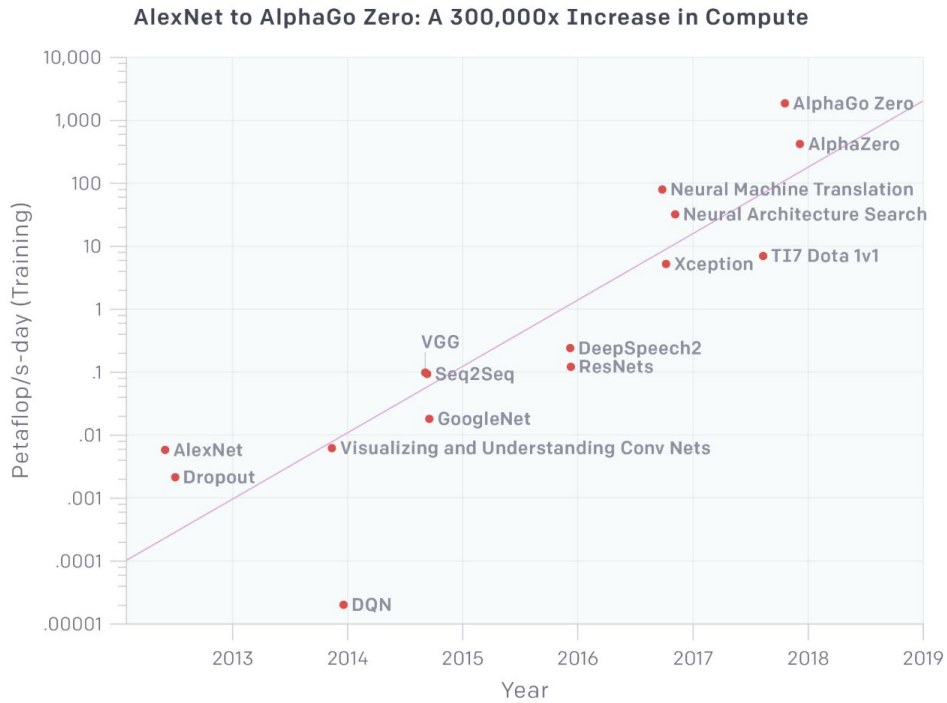


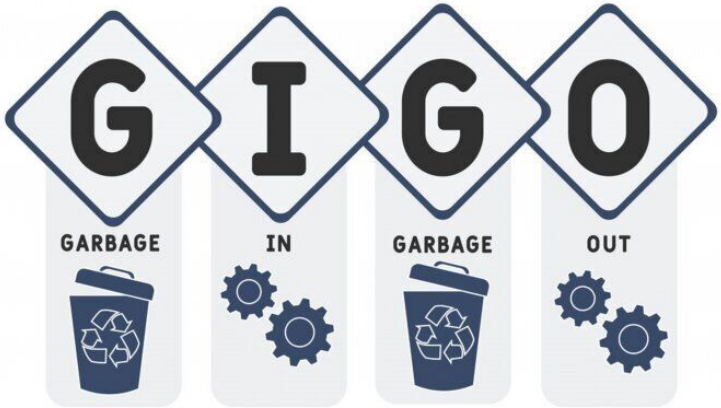
Figure 1: Training datasets for language (left) and vision (right).

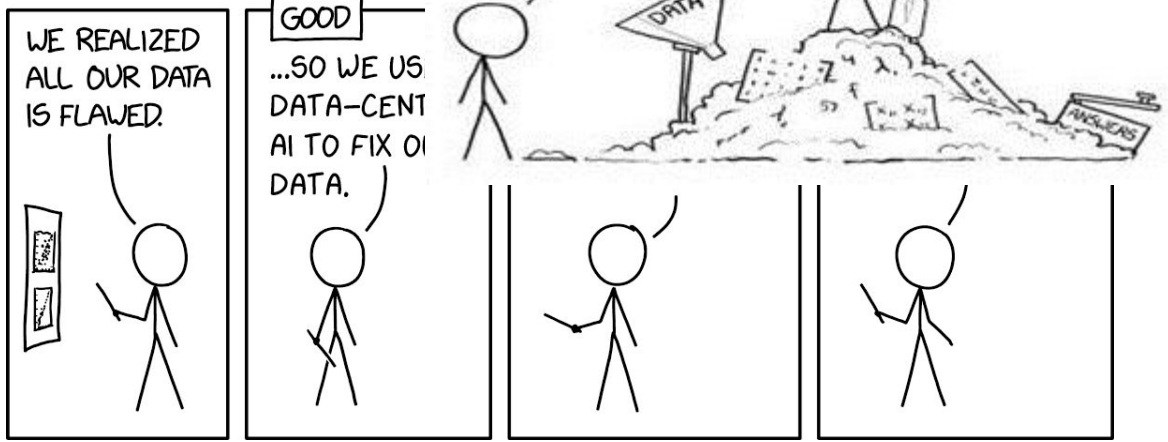
https://www.lesswrong.com/posts/asqDCb9XzXnLjSfgL/trends-in-training-dataset-sizes

“amount of compute used in the largest AI training runs has doubled every 3.5 months”



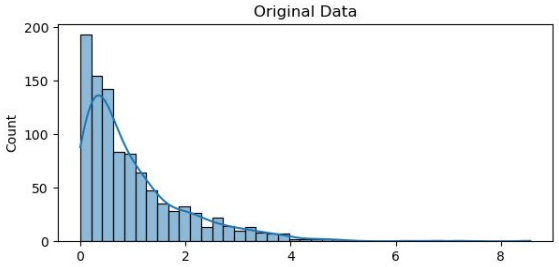
AI and Compute (OpenAI 2018)

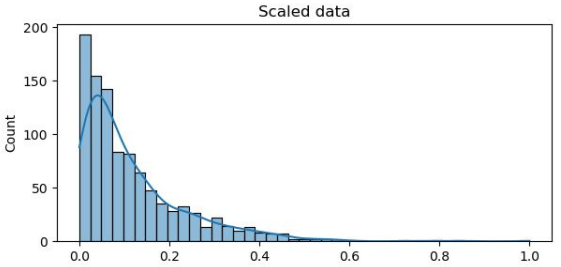




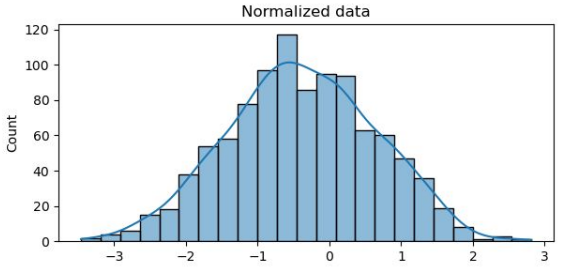
LIÈGE université

Difference between scaling and normalisation





- In **scaling**, you are changing the **range of your data** but without changing the shape of the distribution
- In **normalisation** you are changing the shape of the distribution of your data



LIÈGE université

Scaling

1. min/max normalization
   - Any -- no assumptions about variables
2. z-score normalization
   - When variables follow a **normal distribution**
3. log scaling
   - When variables follow an **exponential distribution**

Requires global statistics.
Helpful for linear models.