RESEARCH DATA UNIT

Scientific AI
Hamprecht Lab, IWR, Heidelberg University

SCIENTIFIC SOFTWARE CENTER

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

AI in research software: Best practices

Research Data Unit: Dr. Sebastian Zangerle
Scientific AI group: Peter Lippmann
Scientific Software Center: Dr. Inga Ulusoy, Dr. Harald Mack

February 2026

1. Requirements of “ML-based science”

What this course is not

- An introduction to data science
- An introduction to machine learning
- A course about different ML algorithms
- A course about different ML training approaches and libraries
- …

What this course is

- A best practices guide to creating machine learning based research software (MLBRS)
- A recommendation on how to manage and prepare your data
- A recommendation on how to train your models
- An introduction to software engineering best practices for MLBRS
- A guideline on how to generate independently reproducible scientific results using data-based approaches
- A guideline on how to publish your data and your models in an interoperable and responsible way

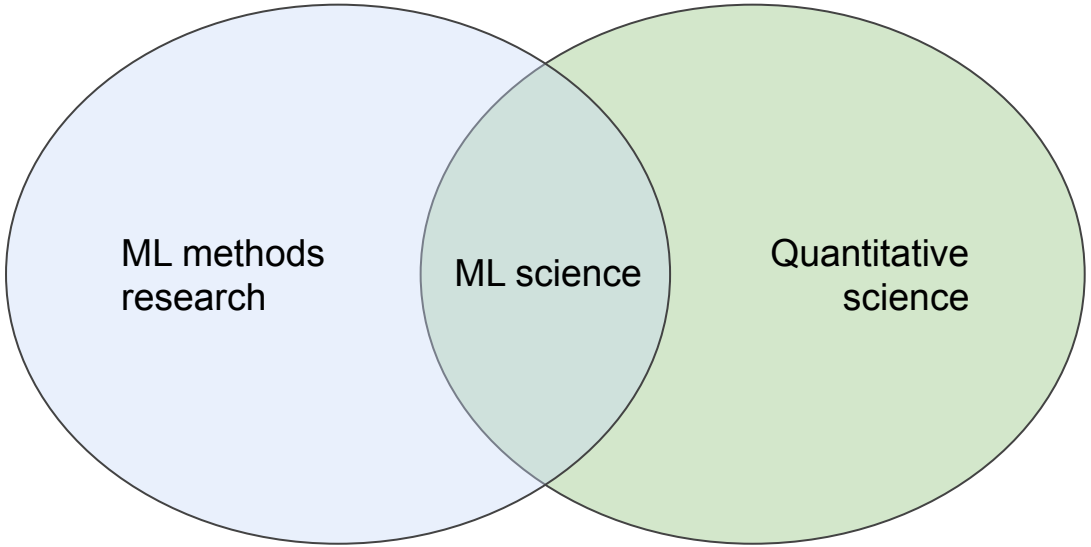
What is special about research software based on data? (“ML-based science”)

ML science

- Scientific research that uses machine learning models to extend scientific knowledge
- Answers a scientific question by using ML
- No restriction on algorithm, method, library, domain

Contrary to:

- ML methods research: Research on ML methods and algorithms with the goal to improve the field of ML



Sayash Kapoor *et al*, http://arxiv.org/abs/2308.07832

Research software

"... software that is developed and used in the context of research…"

Shifting requirements

A scientific question is answered using computation/simulation, but the way the problem is solved changes as part of the research process.

Passed along researchers

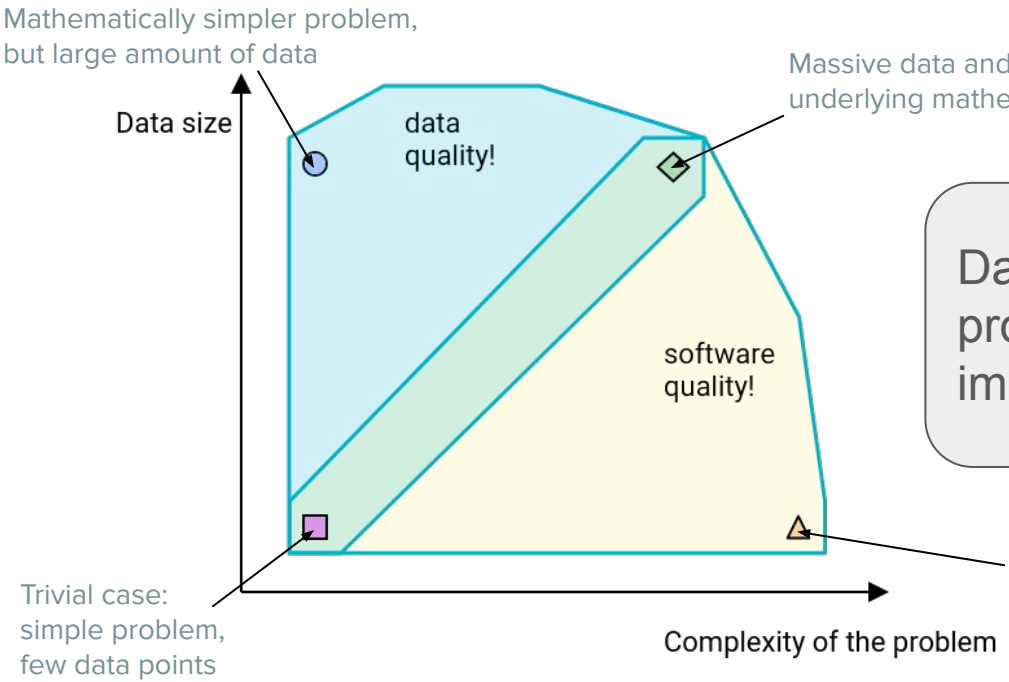
Initially developed for one purpose but then often organically extended depending on the researcher's needs.

Development Practices

Often created by researchers that have no fundamental training in software engineering and inherit practices from those around them.

ML-based research software

"... software that is developed and used in the context of research and predicts outcomes based on initial data…"



Data ***and*** software quality provide limits to quality and impact of research!

MLBRS: Data

Data is foundation for..

…model training, decision making and/or predictions.

Different kinds of data

For example, numerical data, textual data, images, audio, video.

Metadata

What is relevant metadata and should be included on the data card?

Availability and licensing

Will the data be publicly available to the community? What license does/will the dataset have?

Legal considerations

Where does the data come from? Is it licensed? Is it public or private data? In what form is the data stored and processed?

Ethical considerations

Does the data exploit work of others? Does it break some sort of confidentiality? Will it impact in a possible harmful way or can it be misconstrued to do harm?

Bias

Is there an inherent bias in the data itself, due to the data collection approach, or other reasons?

MLBRS: Software

Purpose

Will the software be more widely used, be an in-house code, or one-person software?

Software engineering best practices

Does the software follow software engineering best practices (version control, testing, documentation, …)?

Usability and reproducibility

Does the software include documentation on how models can be trained, and keeps track of training parameters? Does the software help to generate model cards and provide models in transferable format?

Accuracy and reliability

Does the software create robust and consistent results, even though it is based on a non-deterministic process?

Legal considerations

Does the software incorporate third-party models and/or code?

Legal considerations

What license is the software published under?
What license are models published under?

Security

Is the software secure against data injection?

Reproducibility

- Provide data to enable others to reproduce findings
- Provide code to enable others to reproduce findings

➔ ***Computational reproducibility (i)***

- Make sure your findings are true findings, and do not arise from problems with your data/code

➔ ***Independent reproducibility (ii) // "Responsible AI"***

Research software engineering generally targets (i), but with MLBRS we target (ii)

Why should you care?

Your research integrity, scientific best conduct (malpractice), can have long-lasting detrimental effect on science (impact on others and the field), affects society!

Key aspects

Software quality

Interoperability

Legal aspects

Reproducibility of the model's predictions

Reproducibility of the model training

Documentation on data collection, data cleaning, feature selection

Ethical aspects

Documentation on model training, hyperparameter tuning, model testing

Robustness of the model(s)

Data bias

Data leakage

Model bias

Software security

Responsible AI

2. Research Data Management

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

RESEARCH DATA UNIT HEIDELBERG

RESEARCH DATA UNIT

RESEARCH DATA UNIT HEIDELBERG

Research Data Unit at Heidelberg University

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

http://data.uni-heidelberg.de/

Project Planning

- Data Management Plans
- Courses & workshops
- Technical & organizational information

Data processing

- heiBOX
- heiCLOUD/nextcloud
- SDS@hd
- High Performance Computing
- RedCap

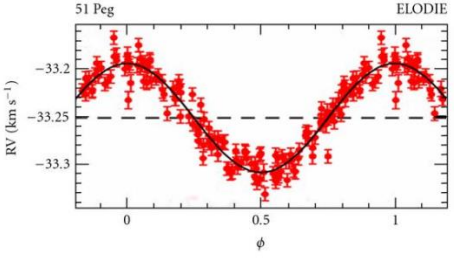
Data Archiving & Publication

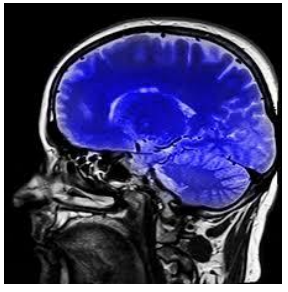
- heiDATA
- heidICON
- Archive – your data preserved
- heiARCHIVE

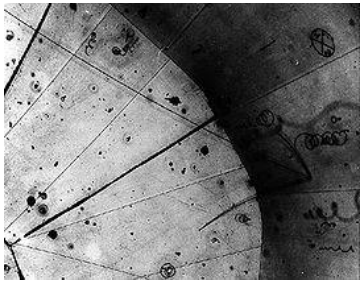
Pictures: © Universität Heidelberg, Kommunikation und Marketing

RESEARCH DATA UNIT HEIDELBERG

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386







WHAT IS RDM ABOUT?

RESEARCH DATA UNIT HEIDELBERG

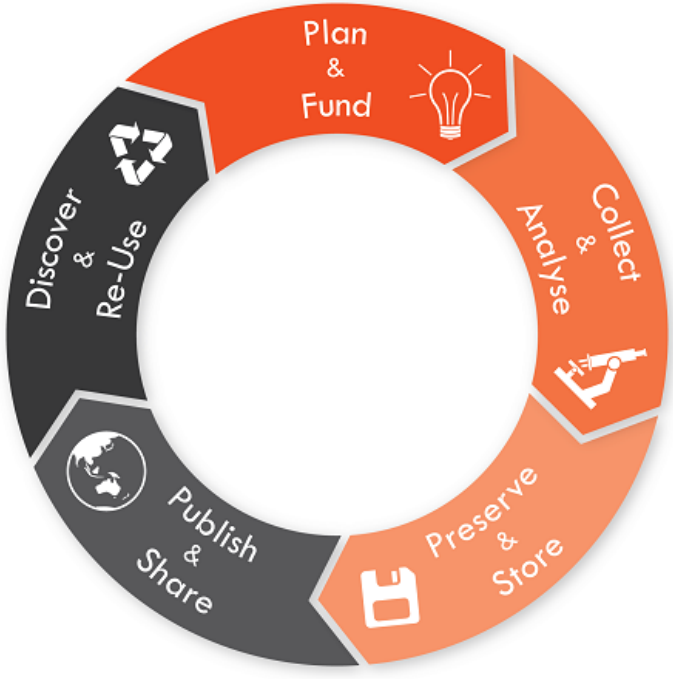
What is research data management?

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Research data management

"Research data management concerns the organisation of data, from its entry to the research cycle through to the dissemination and archiving of valuable results. It aims to ensure reliable verification of results, and permits new and innovative research built on existing information."

(Whyte, A., Tedds, J. (2011). 'Making the Case for Research Data Management'. DCC Briefing Papers. Edinburgh: Digital Curation Centre.)



https://library.sydney.edu.au/research/data-management/research-data-management.html

RESEARCH DATA UNIT HEIDELBERG

EXTERNAL REQUIREMENTS & POLICIES

Picture: https://www.flickr.com/photos/raummaschine/9214045295/

Policies & external requirements

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

SATZUNG ZUR SICHERUNG GUTER WISSENSCHAFTLICHER PRAXIS UND ZUM UMGANG MIT WISSENSCHAFTLICHEM FEHLVERHALTEN

in der Fassung vom 28.09.2021

Präambel

Zur Wahrnehmung ihrer Verantwortung in den drei Handlungsfeldern Forschung, Studium und Lehre sowie Wissenstransfer trifft die Universität Heidelberg im gesetzlichen Rahmen Vorkehrungen zur Verankerung einer Kultur der guten wissenschaftlichen Praxis. Der Senat hat deshalb in seiner Sitzung vom 28.09.2021 gemäß § 3 Abs. 5 S. 4 LHG i.V.m. § 19 Abs. 1 S. 2 Nr. 10 LHG die folgenden Regelungen beschlossen, durch die die Leitlinien zur Sicherung guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft (DFG) vom August 2019 rechtsverbindlich umgesetzt werden:

RESEARCH DATA UNIT HEIDELBERG

Policies & external requirements

Heidelberg University

Rules for safeguarding good academic practice and handling academic misconduct

§ 10 Documentation

(1) Researchers must document all information relevant to the establishment of a research result with the degree of transparency that is required and appropriate in the respective field. The same applies to individual results that do not support the research hypothesis. There must be no selection of results in such cases. Where research software is developed, the source code must be documented.

(2) The information required to understand the research, in particular research data and methodological, evaluation and analysis steps, is recorded. Third parties are to be given access to this information where this is possible.

Policies & external requirements

Heidelberg University

Rules for safeguarding good academic practice and handling academic misconduct

§ 11 Public access to research findings

“Researchers decide on their own responsibility whether, how and where to make their research findings publicly available. If they decide to publish their results, the data and principal materials upon which the published work is based must be stored in recognised archives and repositories where this is possible. The provisions of § 14 must be observed.”

Policies & external requirements

Heidelberg University

Rules for safeguarding good academic practice and handling academic misconduct

§ 16 Archiving

"(1) Once they have been made publicly available, research data and findings, and particularly the materials on which they are based, as well as the instruments and, where applicable, the research software used, must be backed up by adequate means according to the standards of the respective field and stored for the legally required time period (usually ten years). A shortening of this storage period must be justified. The storage period begins when the materials are first made publicly available.

(2) The materials are archived a) in the researchers' home institution or b) in repositories serving several locations. In case a) the university will provide the necessary infrastructure for archiving. The selected publication medium must make reference to the archiving location in an appropriate manner."

Policies & external requirements

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

RESEARCH DATA POLICY

RICHTLINIEN FÜR DAS MANAGEMENT VON FORSCHUNGSDATEN

Die Verfügbarkeit von Forschungsdaten ist die Gewähr für die Nachvollziehbarkeit und Überprüfbarkeit sowie die weitergehende Nutzung nach der Veröffentlichung. Sie ist ein zentraler Aspekt guter wissenschaftlicher Praxis der Universität. Ihr Management nach höchsten Standards baut auf diesem Prinzip auf und ist Teil der Exzellenzstrategie.

1. Die Verantwortlichkeit für den Lebenszyklus(*) von Forschungsdaten, insbesondere die Sicherstellung und Bereitstellung der Forschungsdaten zur langfristigen Archivierung liegt primär beim Projektverantwortlichen (PI).
2. Teil jedes Forschungsprojektes ist ein Plan für das Datenmanagement, der explizit adressiert, wie die Akkuratheit, Vollständigkeit, Authentizität, Integrität, Vertraulichkeit, Veröffentlichung und der offene Zugang von Daten gehandhabt werden. Dabei werden fachspezifische Besonderheiten berücksichtigt.
3. Die Universität unterstützt nach bestem Vermögen die PIs durch ein Kompetenzzentrum Forschungsdaten. Es bietet Beratung und Unterstützung bei der Entwicklung von Konzepten für ihr Datenmanagement an. Dafür ist eine frühzeitige Kontaktaufnahme vor oder zu Projektbeginn erforderlich.
4. Der Plan für das Management von Forschungsdaten stellt den Zugriff und die Nutzung unter Einhaltung von ethischen und Open Access-Prinzipien unter geeigneten Sicherheitsmaßnahmen sicher. Der Open-Access-Policy der Universität folgend ermuntert die Universitätsleitung Wissenschaftler ausdrücklich, Forschungsdaten gemäß der Grundsätze von Open Access, wie sie in der „Berliner Erklärung über offenen Zugang zu wissenschaftlichem Wissen" von 2003 beschrieben sind, zugänglich zu machen, solange keine entgegenstehenden rechtlichen Verpflichtungen bestehen (insb. Verträge mit Verlagen). Für Daten, die Grundlage von schutzfähigem, geistigem Eigentum sind, gilt grundsätzlich die Verpflichtung zur Einreichung einer Erfindungsmeldung gemäß Arbeitnehmererfindungsgesetz (§§ 5, 42 Nr. 2) und die IP-Policy der Universität Heidelberg vorrangig.
5. Die persönlichen Daten von Probanden, Patienten und andere von Datenerhebungen betroffenen Personen werden gemäß den Datenschutzrichtlinien geschützt.
6. Daten, die außerhalb der Universität als Teil des Datenmanagementplans bereitgehalten werden, sollten beim Kompetenzzentrum Forschungsdaten registriert werden. Das Kompetenzzentrum Forschungsdaten bietet eine Datenregistrierung an, die Datensätze sowohl aus universitären als auch externen Repositorien nachweist.
7. Alle Rechte an Daten, insbesondere das Recht, die Daten weitergehend zu nutzen oder zu publizieren, sollten den PIs vorbehalten sein und nicht an Dritte vergeben werden.

RESEARCH DATA UNIT HEIDELBERG

Policies & external requirements

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

RESEARCH DATA POLICY

RICHTLINIEN FÜR DAS MANAGEMENT VON FORSCHUNGSDATEN

Seven paragraphs

1) PI‘s are responsible for the whole research data lifecycle.
2) Every research project should develop a data management plan.
3) University offers support via the Research Data Unit.
4) University encourages researcher to publish open access if possible.
5) Importance of data privacy.
6) Data published outside of the university‘s webspace should be registered at the RDU.
7) PI‘s shall keep their right on data use and publication and shall not transfer it to third parties.

[Research Data Policy - Universität Heidelberg (uni-heidelberg.de)]

RESEARCH DATA UNIT HEIDELBERG

Funders are pushing RDM & Open Data

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

DFG Deutsche Forschungsgemeinschaft

DFG Guidelines on the Handling of Research Data

"[…] For this reason, the handling of research data and the objects on which the data is based have to be carefully planned, documented and described. Wherever possible it is important to enable subsequent use of the research data and potentially also the objects by other users.
[…]
For this reason, the DFG expects research projects to include a description of how research data is handled. The description should be based on the checklist for handling research data
[…]
Costs incurred for the project-specific handling of research data should be requested in connection with the project.[…]"

RESEARCH DATA UNIT HEIDELBERG

Funders are pushing RDM & Open Data

Horizon 2020 & Horizon Europe: FAIR Data Management

- Participating projects will be required to develop a **Data Management Plan** (DMP)
- Participating projects are **required to deposit research data**, preferably into a research data repository
- "[…]as far as possible, projects must then **take measures to enable for third parties to access**, mine, exploit, reproduce and disseminate (free of charge for any user) this research data."
- http://www.dfg.de/foerderung/antrag_gutachter_gremien/antragstellende/nachnutzung_forschungsdaten/
- Guidelines on Open Access to Scientific Publications and Research Data in Horizon 2020 | Guidelines on Data Management in Horizon 2020

Journals: Nature

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

An inherent principle of publication is that others should be able to replicate and build upon the authors' published claims. A condition of publication in a Nature Portfolio journal is that **authors are required to make materials, data, code, and associated protocols promptly available to readers without undue qualifications.**

[…]Providing large datasets in supplementary information is strongly discouraged and the preferred approach is to make data available in repositories.

https://www.nature.com/nature-portfolio/editorial-policies/reporting-standards#availability-of-data

https://www.nature.com/sdata/policies/repositories

RESEARCH DATA UNIT HEIDELBERG

Journals: PLOS

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Data Availability

PLOS journals require authors to make all data necessary to replicate their study's findings publicly available without restriction at the time of publication. When specific legal or ethical restrictions prohibit public sharing of a data set, authors must indicate how others may obtain access to the data.
[…]
Publication is conditional on compliance with this policy. If restrictions on access to data come to light after publication, we reserve the right to post a Correction, an Editorial Expression of Concern, contact the authors' institutions and funders, or, in extreme cases, retract the publication. […]

https://journals.plos.org/plosone/s/data-availability

RESEARCH DATA UNIT HEIDELBERG

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

LEGAL ISSUES

RESEARCH
DATA UNIT
HEIDELBERG

Legal issues

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Research data and copyright

- Textual data typically are protected by copyright
- Copy right holder can grant simple or exclusive usage rights
- For publications in subscription journals: typically unlimited and irrevocable transfer of rights to the publishers

- Research data? Facts like measurements generally do not reach the threshold of originality, even though the data collection can be very sophisticated.
- Therefore: According to German copyright law, research data are in many cases not copyrighted.
- But many data are in databases and there is some kind of protection for these (EU directive 96/09/EG, UrhG §§ 87a-e). Virtually all data are useless without documentation. This documentation might very well be protected by copyright.

RESEARCH DATA UNIT HEIDELBERG

Legal issues

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

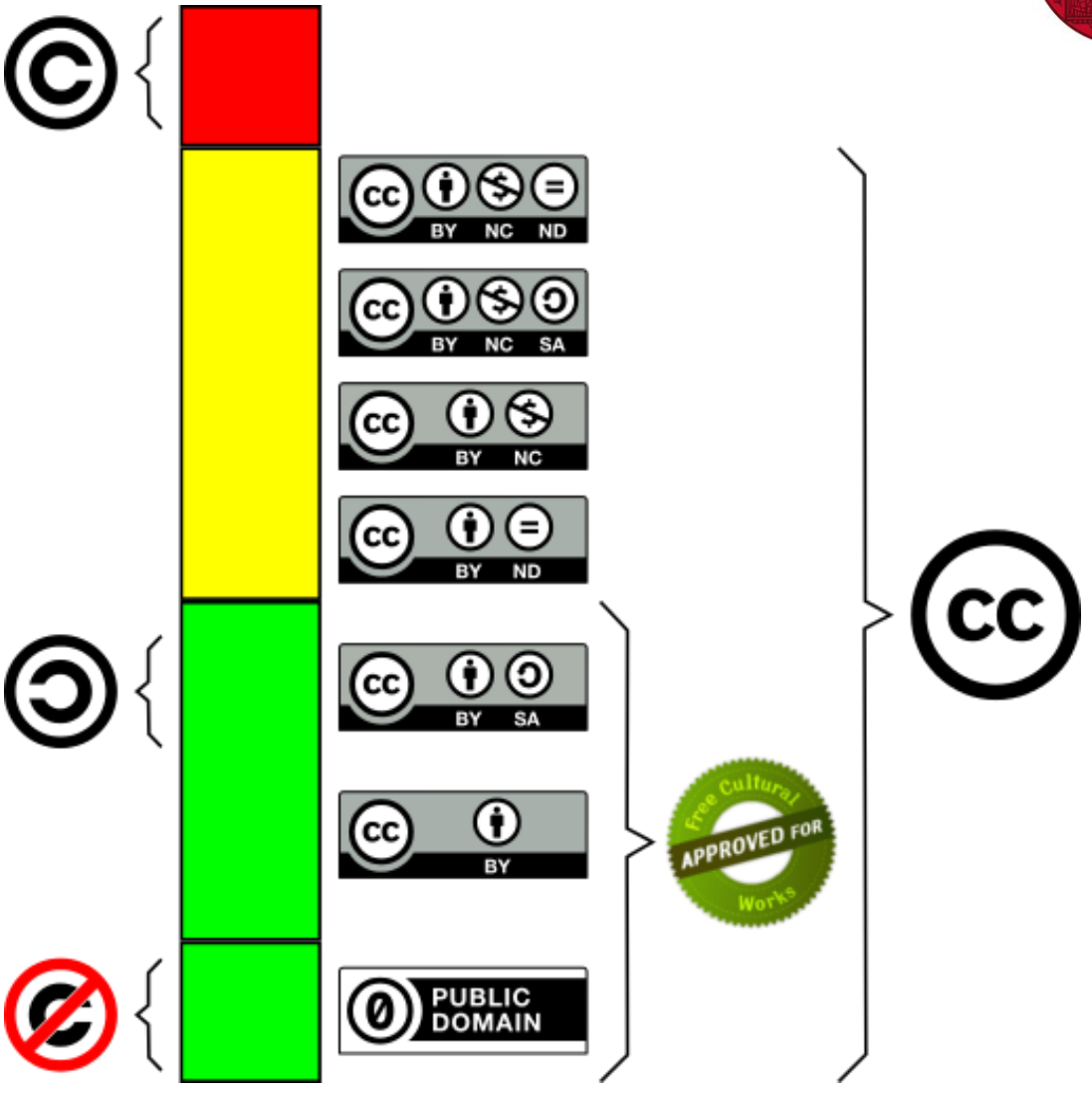
Creative Commons Licences

- Standard licences that determine the scope of use of a work
- Combination of layperson-friendly formulation and a legally proper license text adapted to the relevant national law.
- Licence content and metadata are available in machine readable form and can be added to a document. (→ TDM)
- Modular structure with differing “degrees of freedom”

- There are also alternatives, e.g. the Open Data Commons licenses.
- For Software there are specific software licenses

RESEARCH DATA UNIT HEIDELBERG

Legal issues

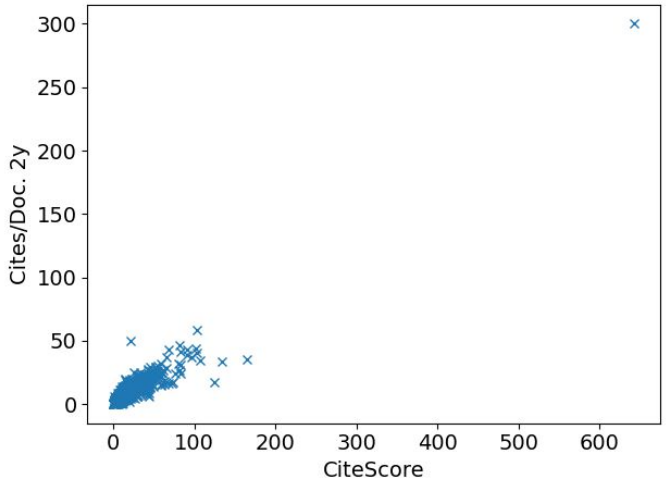


Picture by Marko Txopitea "Txopi" [CC0], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3ACreative_Commons_Semaforoa.svg

3. Research Data Quality

Collecting data

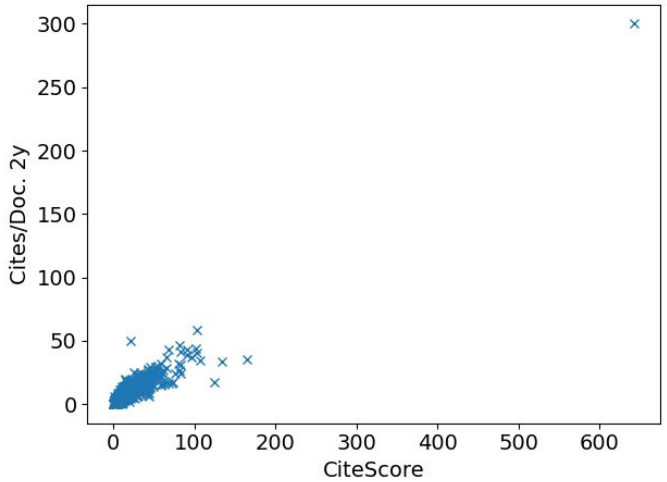
- Data must contain all ranges of the condition that is to be sampled
  - For example: To predict the impact of temperature on reactivity, all temperatures that are of interest need to be sampled (predictions only interpolate between data points but cannot extrapolate).
  - For example: CiteScore (Scopus citation index) vs. citations over all documents from last 2 years, for scientific journals.



Dataset: journal ranking dataset https://www.kaggle.com/datasets/xabirhasan/journal-ranking-dataset

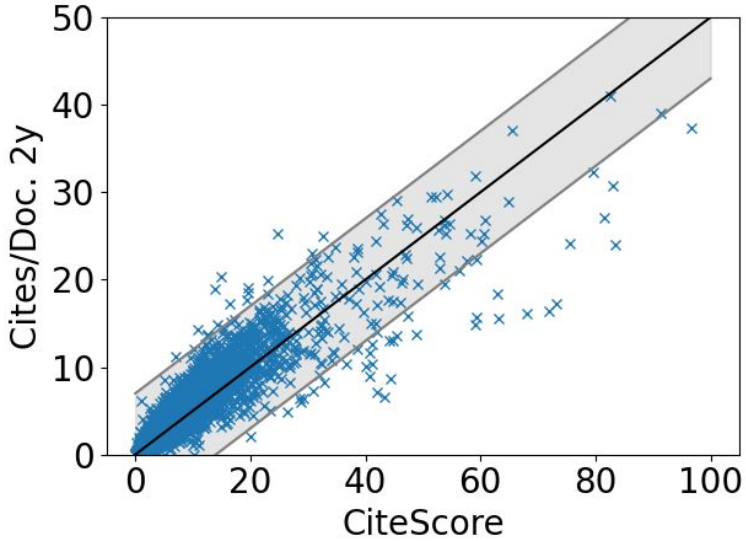
Collecting data

- Data must be homogeneous throughout feature space
  - For example: If temperature and pressure are both sampled, all combinations of features must be recorded for a homogeneous distribution of data points.
  - For example: CiteScore (Scopus citation index) vs. citations over all documents from last 2 years, for scientific journals.



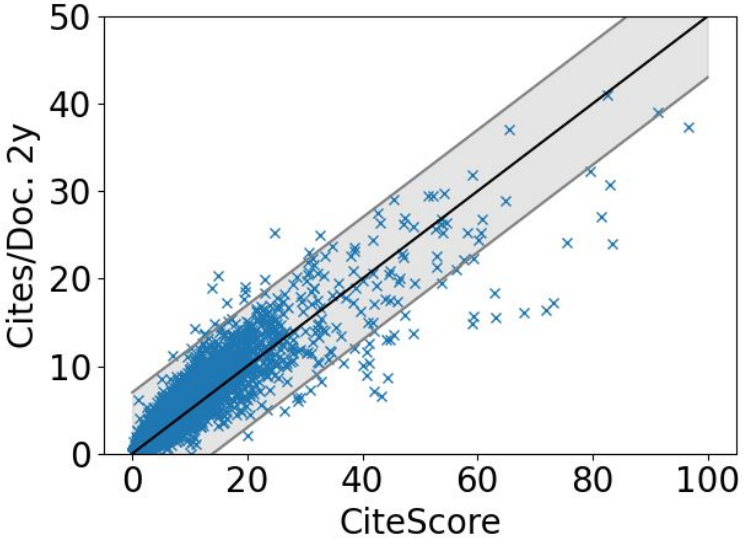
Collecting data

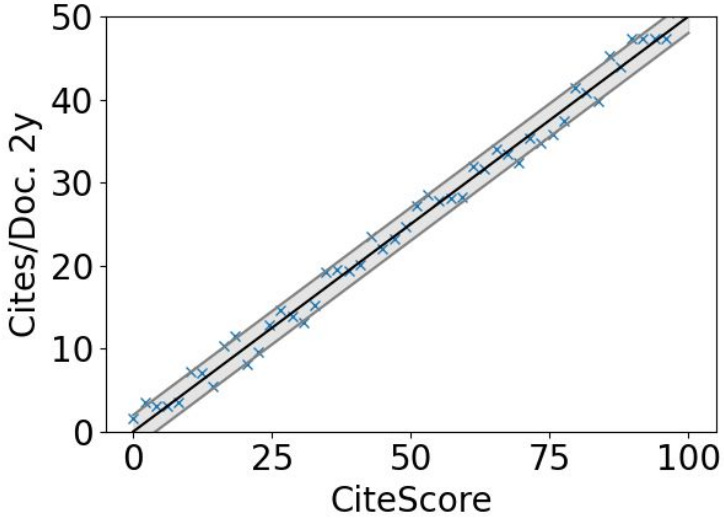
- Data must be of good quality
  - Whether it is real or synthetic data, the model can only make accurate predictions if the data itself is accurate.
  - For example: CiteScore (Scopus citation index) vs. citations over all documents from last 2 years, for scientific journals



Collecting data

- Data volume must be sufficient
  - Only with enough data can a model be trained to make accurate predictions.
  - For example: Complex data - more data points required; simpler data - fewer data points required





Collecting data

- Depending on the type of learning, data must be labeled and labeled correctly
  - Incorrect labelling interferes with the learning process.

Photo by nishizuka:
https://www.pexels.com/photo/brown-chihuahua-485294/

Photo by Maksim Goncharenok:
https://www.pexels.com/photo/a-chocolate-muffin-on-blue-surface-5994864/

Data preparation

- Make sure data is clean.
  - Correct typos, misidentified data types

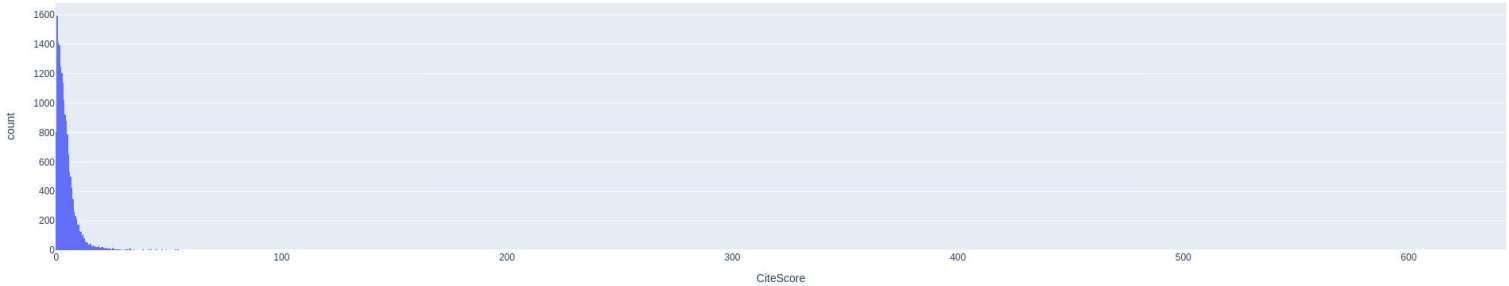
Chihuahuah →Chihuahua

Photo by nishizuka:
https://www.pexels.com/photo/brown-chihuahua-485294/

“26-04-24” →2024-04-26

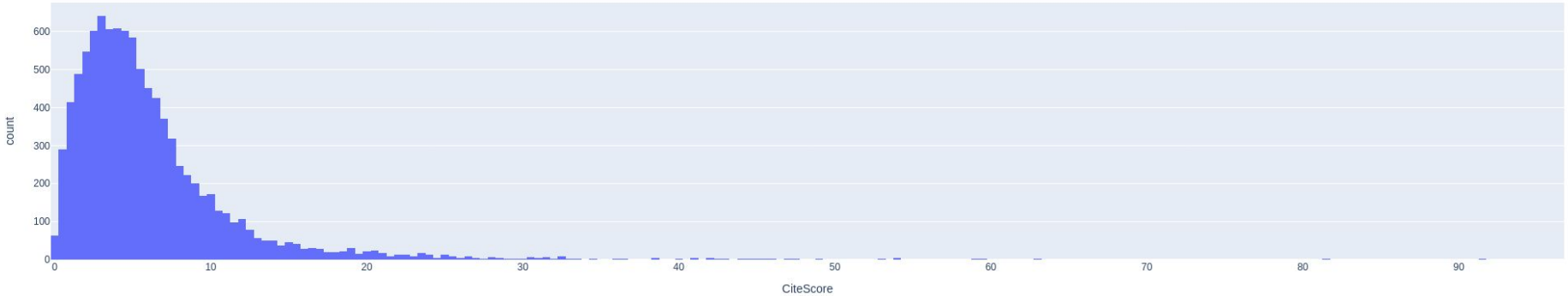
Data preparation

- Make sure data is homogeneous.
  - Visualize the data and use clustering analysis to identify outliers.
  - Use `df.describe()` and `plotly.express` to better understand your data



Data preparation

- Remove duplicates.
  - Duplicates introduce bias.
  - Use df.drop_duplicates()



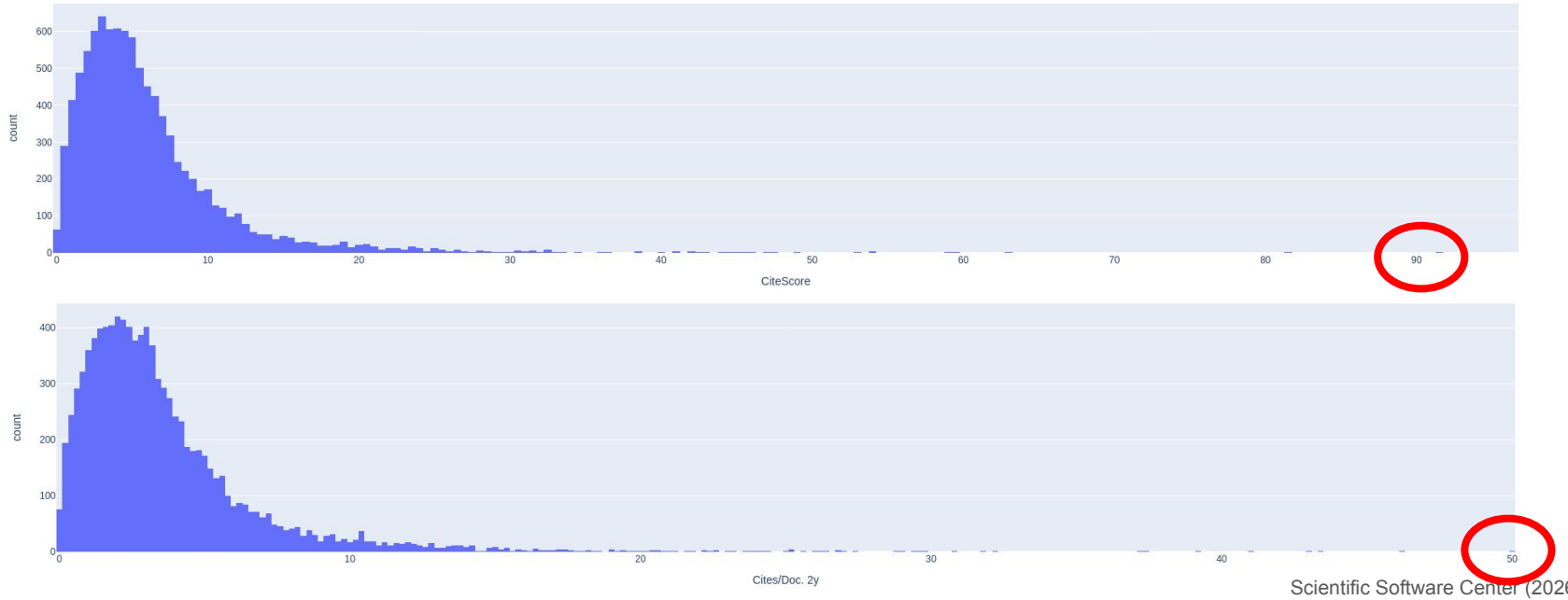
Data preparation

- Feature Engineering: Select influential features, remove unnecessary ones.
  - Unimportant features increase the complexity and reduce robustness.
  - For example: only choose features that are clearly correlated

	Rank	OA	SJR-index	CiteScore	H-index	Best Subject Rank	Total Docs.	Total Docs. 3y	Total Refs.	Total Cites 3y	Citable Docs. 3y	Cites/Doc. 2y
Rank	1.000000	0.111300	-0.503617	-0.485568	-0.625403	0.558208	-0.192069	-0.196795	-0.196338	-0.243070	-0.185484	-0.560625
OA	0.111300	1.000000	-0.069304	-0.056997	-0.178146	0.114037	0.061870	0.046403	0.058683	0.024084	0.048485	-0.045120
SJR-index	-0.503617	-0.069304	1.000000	0.878000	0.565015	-0.281225	0.091092	0.102424	0.094227	0.270083	0.081086	0.828618
CiteScore	-0.485568	-0.056997	0.878000	1.000000	0.527957	-0.279983	0.112000	0.127705	0.122350	0.285965	0.110357	0.943584
H-index	-0.625403	-0.178146	0.565015	0.527957	1.000000	-0.362788	0.331053	0.393130	0.313698	0.505095	0.362266	0.512423
Best Subject Rank	0.558208	0.114037	-0.281225	-0.279983	-0.362788	1.000000	-0.114754	-0.117089	-0.132615	-0.150247	-0.118463	-0.334142
Total Docs.	-0.192069	0.061870	0.091092	0.112000	0.331053	-0.114754	1.000000	0.934468	0.968011	0.806830	0.932626	0.150987
Total Docs. 3y	-0.196795	0.046403	0.102424	0.127705	0.393130	-0.117089	0.934468	1.000000	0.887417	0.854647	0.995085	0.148272
Total Refs.	-0.196338	0.058683	0.094227	0.122350	0.313698	-0.132615	0.968011	0.887417	1.000000	0.802696	0.893789	0.173401
Total Cites 3y	-0.243070	0.024084	0.270083	0.285965	0.505095	-0.150247	0.806830	0.854647	0.802696	1.000000	0.844114	0.308644
Citable Docs. 3y	-0.185484	0.048485	0.081086	0.110357	0.362266	-0.118463	0.932626	0.995085	0.893789	0.844114	1.000000	0.139525
Cites/Doc. 2y	-0.560625	-0.045120	0.828618	0.943584	0.512423	-0.334142	0.150987	0.148272	0.173401	0.308644	0.139525	1.000000
Refs./Doc.	-0.390894	-0.064572	0.267383	0.306913	0.247259	-0.299281	0.032381	0.019822	0.109949	0.076826	0.030626	0.382891
Life Sciences	-0.166150	0.073645	0.071380	0.125088	0.210414	-0.183625	0.044939	0.050866	0.068018	0.051387	0.051948	0.114416

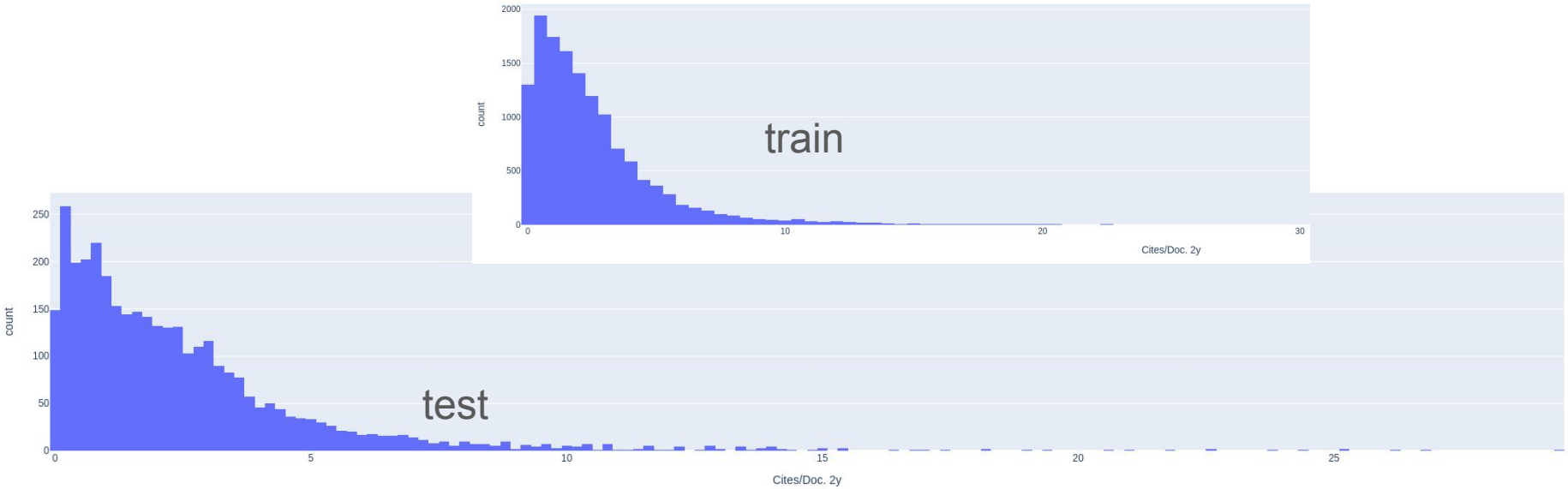
Data preparation

- Feature Engineering: Normalize features.
  - Features should have similar data ranges for the weights to be in similar ranges, and improved model robustness and faster training.



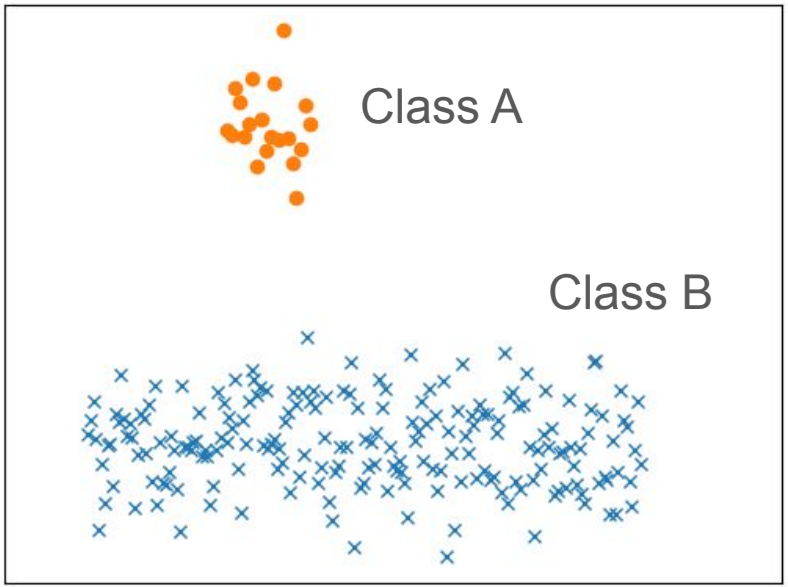
Data preparation

- Make sure to randomize your data.
  - Otherwise, your train and test data could contain more/less data of a certain kind (inhomogeneous data)



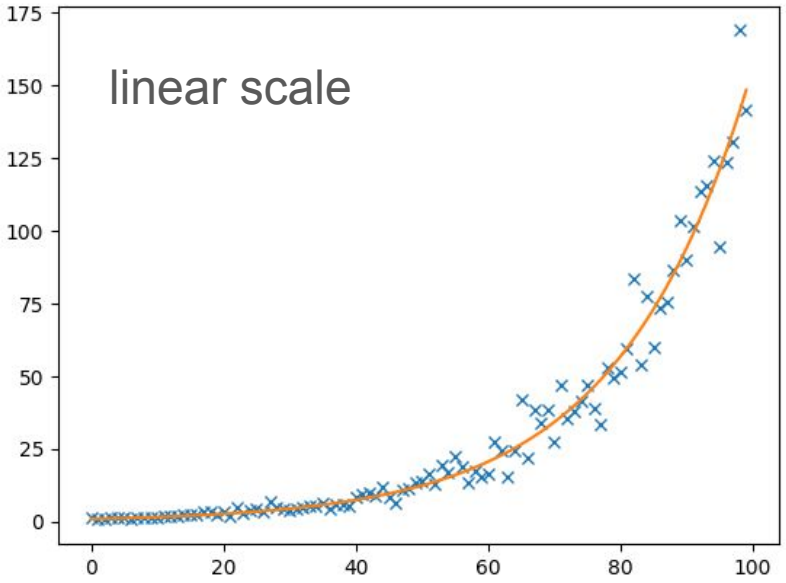
Data preparation

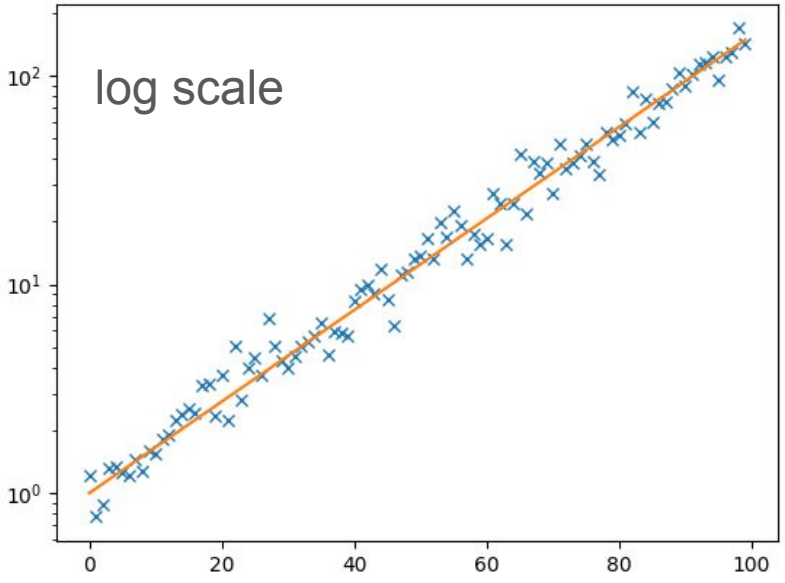
- Feature engineering: Make sure your dataset is balanced.
  - For classification tasks, all classes should have comparable sizes (similar numbers of examples).



Data preparation

- Feature engineering: Pick the right scale.
  - Visualize your data to see if you need to transform ie. onto a log scale.





4. Modeling of Research Data

Deep learning – Tools and Tricks

I wish I had known earlier

Peter Lippmann

05.02.2026

Scientific AI Lab, IWR, Heidelberg University

Deep learning overview

$$\min_{\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}} \ L(y, f_\theta(x))$$

$\mathcal{D}$: data distribution, typically approximated by a finite set of input-target pairs $\{x_i, y_i\}$

f_θ: neural network with learnable parameters θ

L: differentiable loss function, typically minimal if $y = f_\theta(x)$

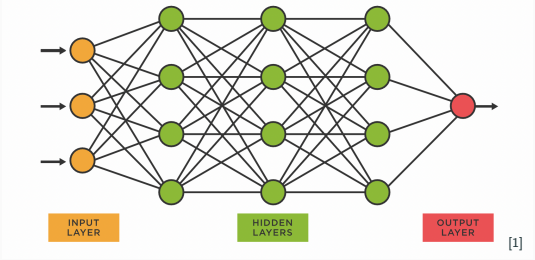
In practice:

$$\min_{\theta} \sum_{\substack{\text{sample } (x_i,y_i) \\ \text{in training set}}} L(y_i, f_\theta(x_i))$$

Optimization via (mini batch) gradient descent:

$$\theta^{(t+1)} = \theta^{(t)} \quad - \quad \alpha \sum_{\substack{\text{sample } (x_i,y_i) \\ \text{in mini batch}}} \nabla_\theta L(y_i, f_\theta(x_i))$$

The neural network



[1]

- Network types: CNNs for images, GNNs for graphs, sequence models for language, recurrent NNs for time series data, ... (often clear what to choose)
- BUT many representatives, e.g. many different CNN architectures (less clear)
- For given architecture, several hyperparameters (educated guess + trying out)

The loss function

$$\min_{\theta} \sum_{\substack{\text{sample } (x_i, y_i) \\ \text{in training set}}} L(y_i, f_\theta(x_i))$$

- Heavily depends on your task, e.g.:
  - classification $\leftrightarrow$ Cross Entropy loss,
  - regression $\leftrightarrow$ MSE (a.k.a. L2-loss) $\|y_i - f_\theta(x_i)\|^2$
- Tip: for regression L1-loss $|y_i - f_\theta(x_i)|$ can be more robust to outliers
- Sometimes a combination of losses is used (weighting them can be tricky)

The data

- More data is better, higher quality data is better
- Visualize your data before: PCA, UMAP, t-SNE
- Check your data is balanced (e.g. in instances per class)
- Split your data into train, validation and test set
- Standardize your data (both input and output)
- Use data augmentation if possible

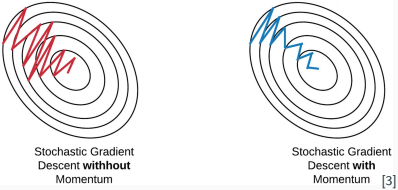
[2]

Data augmentation

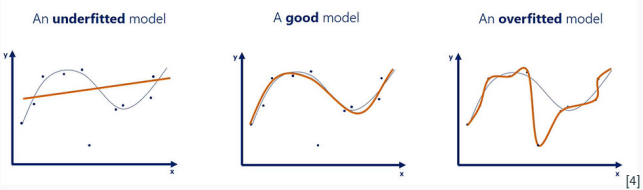
The optimizer

Use AdamW (Adaptive Moment Estimation + weight decay):

- Adaptive learning rate helps against too small or too large gradients
- Momentum stabalizes the gradient descent
- Weight decay helps against overfitting

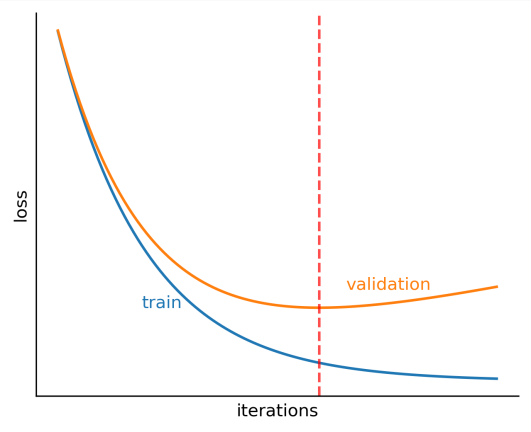


(a) Effect of momentum



(b) Overfitting model

More tricks against overfitting



- Dropout: randomly drop/ignore neurons during training
- Save checkpoints of your model: last (if training crashes) & k-many best

More Training Tricks

- Try to overfit on a single sample to debug your pipeline
- Set a seed during training for reproducibility
- Use enough CPU workers in dataloader to properly use GPU
- In dataloader use `shuffle=True` and `drop_last=True`
- Try gradient clipping in the optimizer against instable training
- Use a learning rate scheduler (e.g. cosine schedule)
- Try learning rate warmup
- Definitely try normalization layers: helps to standardize activations

Model evaluation

- During training, evaluate your model on validation set
- After training (when ready to publish), evaluate your model ONCE on the TEST set
- Use suitable metric to benchmark your model (e.g. accuracy for classification, see `https://metrics-reloaded.dkfz.de/` for image analysis)

Official implementation of the NeurIPS 23 spotlight paper of ♾️InfGCN♾️.

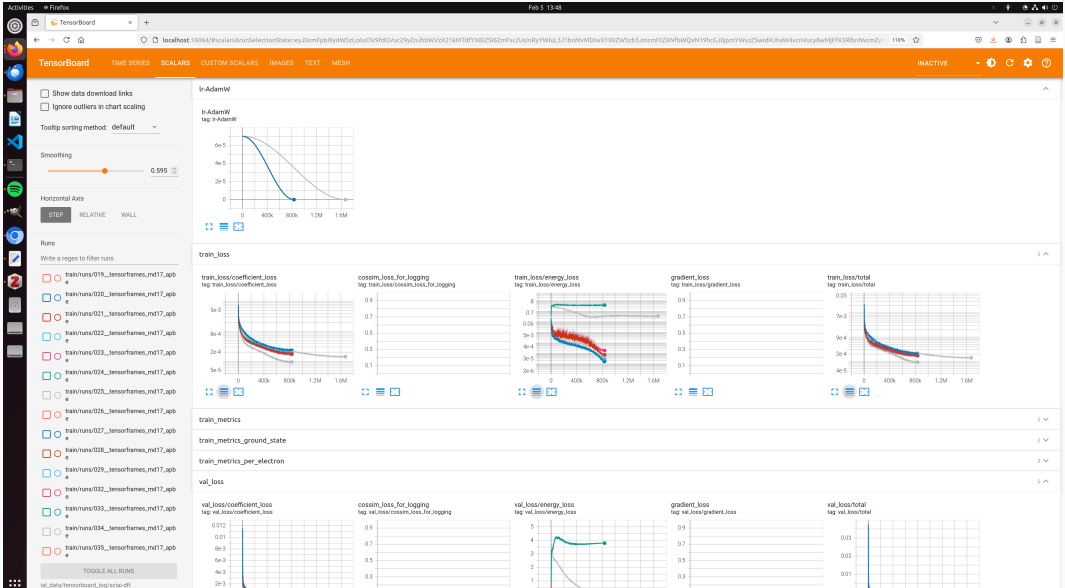
Code | Blame | Raw

```python
@register_dataset('small_density')
class SmallDensityDataset(Dataset):
    def __init__(self, root, mol_name, split):
        """
        Density dataset for small molecules in the MD datasets.
        Note that the validation and test splits are the same.
        :param root: data root
        :param mol_name: name of the molecule
        :param split: data split, can be 'train', 'validation', 'test'
        """
        super(SmallDensityDataset, self).__init__()
        assert mol_name in ('benzene', 'ethanol', 'phenol', 'resorcinol', 'ethane', 'm
        self.root = root
        self.mol_name = mol_name
        self.split = split
        if split == 'validation':
            split = 'test'

        self.n_grid = 50  # number of grid points along each dimension
        self.grid_size = 20.  # box size in Bohr
        self.data_path = os.path.join(root, mol_name, f'{mol_name}_{split}')
```

Scientific Integrity

Logging via Tensorboard



Pytorch Lightning

Pytorch lightning module combines pytorch model + optimizer + logging

- Abstracts away `to("cuda")`, `loss.backward()`, `model.eval()` and much more
- Makes complicated things which many people use easy, e.g. multi GPU support

```python
import lightning as L
import torch

from lightning.pytorch.demos import Transformer


class LightningTransformer(L.LightningModule):
    def __init__(self, vocab_size):
        super().__init__()
        self.model = Transformer(vocab_size=vocab_size)

    def forward(self, inputs, target):
        return self.model(inputs, target)

    def training_step(self, batch, batch_idx):
        inputs, target = batch
        output = self(inputs, target)
        loss = torch.nn.functional.nll_loss(output, target.view(-1))
        return loss

    def configure_optimizers(self):
        return torch.optim.SGD(self.model.parameters(), lr=0.1)
```

Many great tutorials at

`https://lightning.ai/docs/pytorch/stable/starter/introduction.html`

Config management with Hydra

- Save on boilerplate by "programming" in configs (customize models in config not in code)

```yaml
net:
  _target_: mldft.ml.models.components.graphformer.Graphformer
  edge_mlp:
    _target_: mldft.ml.models.components.mlp.MLP
    in_channels: 128
    hidden_channels: [768, 32]
    activation_layer:
      _target_: hydra.utils.get_class
      path: torch.nn.SiLU
    dropout: 0.
  energy_mlp:
    _target_: mldft.ml.models.components.mlp.MLP
    in_channels: 768
    hidden_channels: [768, 1]
    activation_layer:
      _target_: hydra.utils.get_class
      path: torch.nn.SiLU
    dropout: 0.
    disable_dropout_last_layer: True
    disable_activation_last_layer: True
    disable_norm_last_layer: True
```

- Easy to add new models, datasets, tasks and experiments
- Uses OmegaConf for configuration management

Great Hydra + Lightning template at
`https://github.com/ashleve/lightning-hydra-template`

How I use LLMs in my workflow

- Github co-pilot auto-completion in VS code
- Agent mode or LLMs for debugging: just paste the error traceback
- LLMs for speeding up code: from for-loops to vectorized code

```python
# kNN implementation with for-loop:
import torch

k = 3
x = torch.rand(10000, 5)  # 100 samples, 5 features
knn_indices = []
for x_i in x:
    # Compute distances from x_i to all other points
    distances = torch.norm(x - x_i, dim=1)

    # sort distances and get indices of k nearest neighbors
    knn_idx = torch.argsort(distances)[:k+1]  # +1 to exclude self
    knn_indices.append(knn_idx[1:])  # Exclude self

knn_indices = torch.stack(knn_indices)
```

✓ 9.8s

GPT

```python
# Pairwise squared Euclidean distances via broadcasting: (N, N)
# dist2[i, j] = ||x[i] - x[j]||^2
diff = x[:, None, :] - x[None, :, :]      # (N, N, D)
dist2 = (diff * diff).sum(dim=2)          # (N, N)

# Exclude self by setting diagonal to +inf
dist2.fill_diagonal_(float("inf"))

# Full sort per row, take k nearest
knn_indices_parallel = dist2.argsort(dim=1)[:, :k]  # (N, k)
```

✓ 0.6s

And check:

```python
torch.allclose(knn_indices, knn_indices_parallel)  # Should be True
```

✓ 0.0s

```
True
```

Caveat: Bugs introduced by LLMs are hard to spot! ALWAYS check LLM code line by line!

Thank You!

Any Questions?

Image References

[1] https://www.marktechpost.com/wp-content/uploads/2022/09/Screen-Shot-2022-09-23-at-10.46.58-PM.png

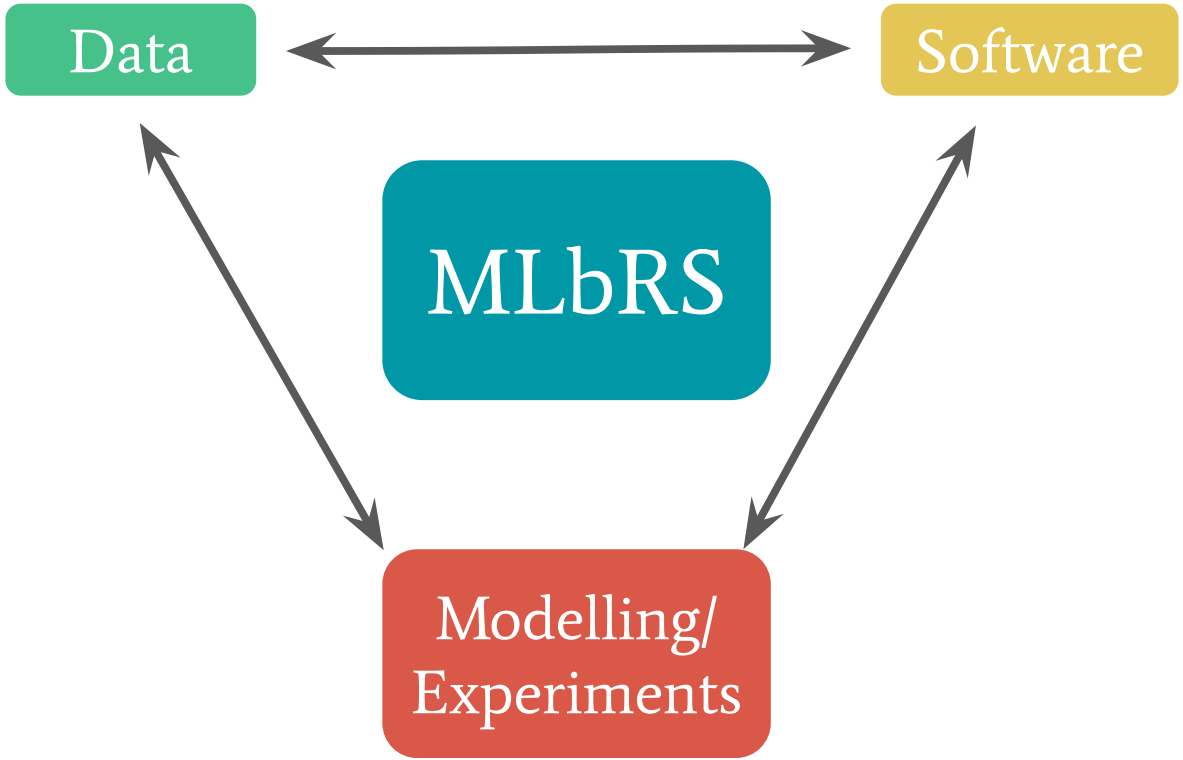
[2] https://media.datacamp.com/legacy/image/upload/v1669203370/Data_Augmentation_Header_f42227f2cb.png

[3] https://i.sstatic.net/epW89.jpg

[4] https://static.wixstatic.com/media/0ed3e8_a9b7d6d3dc6b4d5cbcb30c8b2fd4782b~mv2.jpg/v1/fill/w_1000,h_449,al_c,q_90,usm_0.66_1.00_0.01/0ed3e8_a9b7d6d3dc6b4d5cbcb30c8b2fd4782b~mv2.jpg

5. Reproducibility in MLbRS in practice

Machine-learning based research software



Established principles for sustainable software

- **S**ingle responsibility principle: One task per code entity
- **O**pen-closed principle: Open to extension, closed to change
- **L**iskov substitution: Allow substitution with derived version
- **I**nterface segregation: Only depend on things you need
- **D**ependency inversion: Depend on abstractions, not concretes

Apply to all aspects of MLbRS!

Separation of concerns/ Single responsibility

In software engineering

- Responsibility = 'reason to change'
- Any code entity should just have one reason to change
- Any entity implements only things concerned with a single problem

In MLbRS

- Every part has one 'reason to change'
- data, code and parameterization handled by different entities
- Raw, preprocessed, final data are distinct and don't mix
- every step in the pipeline does one thing only
- library code and experiments in separate repositories

Mental model : "functional purity" - no side effects, same input => same output

Open/closed principle

In software engineering

- new features can be added without modifying existing code
- existing code is stable even if new functionality is added
- established features have baseline correctness => tests

In MLbRS

- make use of modularity in your models, pipeline
- add new experiment runs without impacting previous ones
- Bind data + model def. to experiment

Practical tool : Use frameworks, version control (data + code)

Liskov substitution principle

In software engineering

- system should be able to run with derived classes without knowing it
- 'Design by contract'
- introduces guarantees: everything that adheres to a certain behavioral spec works

In MLbRS

- each stage in the pipeline corresponds to an abstract contract
- different versions of a stage => different implementations of its contract
- use a different model, different data transforms etc without touching our setup
- Guarantees allow faster iteration

Tool: Polymorphism, config templates, repo templates

Interface segregation

In software engineering

- don't force entities to depend on unused interfaces
- split large interfaces into smaller modular parts
- protects from undesired side effects
- sparsifies dependency graph

In MLbRS

- no unnecessary interdependencies between pipeline steps
- make pipeline minimal and specific: only pass data, model modules, evaluators needed
- can help performance too!

Practical tip : use workflow/experiment managers

Dependency inversion

In software engineering

- Functionality should depend on abstractions, not implementations
- introduces loose coupling
- relationships are ‘reversed’ : abstraction -> implementations

in MLbRS

- build pipeline on interacting abstractions
- then use those to derive implementations
- allows for easy scaling and testing

Tool: Encapsulation and abstraction

Summary

- SOLID principles from software engineering
- useful for all aspects of MLbRS
- introduce guarantees and invariants => reduce degrees of freedom in system

Compromises are sometimes necessary:

- Performance: GPU likes doing a lot at once -> violate separation of concerns
- Not all projects need everything: scale, goal, persistence
- Problem space evolves with experiments, often no static 'target' => not everything everywhere all at once
- 'By the spirit, not by the letter'

Practical example I

- have a library git repository + git experiment repository
- library: code, tests, pip releases, CI/CD (later)
- structure of experiment repo:
  - src: driver code
  - configs: parameterization
    - one config per run/sub-experiment
  - data: 'hot data'. This can be elsewhere too (directory, drive, network, cloud storage)
  - results: produced artifacts: config files, model snapshots, evaluation metrics

Practical tip : Use templates/cookiecutter for repository setup

Library: https://github.com/ssciwr/cookiecutter-python-package
Experiments: https://github.com/drivendataorg/cookiecutter-data-science

Practical example I

Done:

- Ensured reproducibility by saving parameterization together with results
- Code + dependencies under version control
- Parameterization and code separated
- stable library code separate from changing experiment code

Not done:

- Separation of concerns:
  - training logic, output setup, config handling all in one script
- Interface segregation:
  - Everything depends on full config
  - No easy substitution of data processing without changing main script…
- Manual versioning of experiments
- guarantees about behavior only in software part!

Practical example II

- MLbRS = Data + Code + Model => Version control for experiments/data?
- Automatic versioning of processed data, experiments, pipeline stages?
- using 'git for data' isn't ideal because
  - Git cannot diff binary files
  - gitlfs is often limited (github)
  - many big files make repositories big
  - separation of concerns: data should not be the concern of the source code VC
- possible solution: DVC (data version control)
  - https://dvc.org
  - adds a git-like metadata layer to get 'git for data and artifacts'
  - tracks the metadata with git
  - Handles data in content-addressable storage

Practical example II - DVC in more detail

Use cases

- Versioning of data and models
- Experiment pipeline setup
- Experiment tracking and sharing

DVC gives you:

- git-like interface for data, experiments
- Similar API to git: commit, branch, checkout…
- reproducible pipelines
- 'Smart run': only pipeline stages that change run again

DVC is not

- inference engine
- hyperparameter optimization system (use Optuna, W&B, Raytune…)
- not a full fledged experiment dashboard

git tracks recipes, dvc tracks results (processed data, model snapshots, experiment metrics)

Practical example II:
Setup dvc repository and add data

- Initialize dvc repository inside experiment git repository:

```
dvc init
```

=> .git, **.dvc**

- Add data to data/raw directory

```
dvc get https://huggingface.co/datasets/ylecun/mnist ./mnist -o
./data/raw/ylecun/mnist
```

=> raw data, not tracked by dvc

```
dvc add ./data/raw/ylecun/mnist
```

```
dvc commit
```

Practical example II: Define pipeline

- **Stage 1 - Data preprocessing**
  - Apply a set of transforms to raw data
  - Save transformed data to disk
  - Input: dataset name, dataset subset size
  - Output: transformed data

- **Stage 2 - model training**
  - Read preprocessed training data
  - Train model

- **Stage 3 - model evaluation**
  - Load trained model
  - Run on test data
  - Record metrics

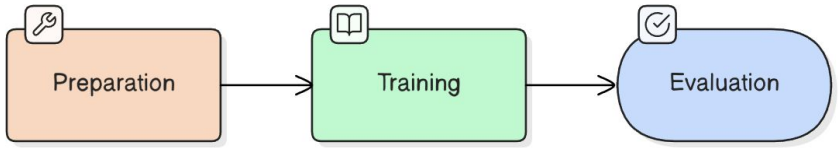
```
dvc stage add -n prepare_data \
-p data.name,data.train_size,data.test_size \
-d src/prepare_data.py -d data/raw/${data.name} \
-o data/processed/${data.name} \
python3 ./src/prepare_data.py --dataset ${data.name} --train_size
    ${data.train_size} --test_size ${data.test_size}
```

```
stages:
  preparation:
    cmd: python3 ./src/prepare_data.py --dataset ${data.name} --train_size ${data.train_size} --test_si
    deps:
      - ./src/prepare_data.py
      - ./data/raw/${data.name}
    params:
      - data.name
      - data.train_size
      - data.test_size
    outs:
      - data/processed
```

Practical example II: Run pipeline

- **Visualize defined pipeline with various backends**

```
dvc dag
```



- **Reproduce current pipeline**

```
dvc repro
```

- Use defined parameters to run the pipeline again
- Relies on data being available!
- Will skip stages for which the input/parameters has not changed

Practical example II: Manage experiments

Run pipeline with a modified set of parameters

```
dvc exp run --set-param configs/training.yaml:learning_rate=0.003
--set-param data.train_size=12000
```

- Modify parameter in params.yaml or referenced sub-configs
- Caches all current experiments
- Allows us to select some and commit it for further use, or branch on them
- DVC will keep track of experiments automatically: dvc.lock
- **Commit dvc.lock asap!**

Practical example II: Track and visualize results

- A pipeline stage can output 'metrics' to quantify experiment result
- `dvc exp show`

Experiment	Created	mean_loss	min_loss	max_loss
workspace	-	0.056889	0.0027314	0.16338
main	12:49 PM	0.083361	0.011967	0.19013
├── 20142e5 [duddy-yews]	01:02 PM	0.056889	0.0027314	0.16338
├── e34fee6 [bendy-clip]	01:02 PM	0.085162	0.0022099	0.20693
├── f46371e [amuck-fish]	12:57 PM	0.073541	0.0014057	0.2323
└── 6df4e2f [mangy-plot]	12:57 PM	0.083361	0.011967	0.19013

Practical example II: Manage experiments

- Select experiment to apply and continue with

```
dvc exp apply *experiment_name*
```

  - Apply parameters to the config file
  - **You must create new git commit with the changes!**

- Promote experiment to git branch to iterate further

```
dvc exp branch *experiment_name*
```

- What you want to keep must go into version control!
- Commit changes often!

Practical example II: DVC Remotes

- Like git, dvc supports remotes to store tracked artifacts
- Remotes can be
  - Http address with login
  - SSH server
  - S3-based cloud storage
  - …
  - local

```
dvc remote add -d default_local ../path/to/local/remote
```

```
dvc commit, dvc push, dvc pull, dvc update…
```

- Add data to cache
- Push the data cache to remote
- Update data cache from remote
- Share repository **and** data remote with coworkers to share experiment status

Practical example II - summary

DVC approach is more SOLID

- Pipeline definition separate from code (Separation of concerns)
- Pipeline definition separate from pipeline parameterization
- Stage defined by command + parameters, not code (Liskov subst., Dependency inversion)
- Arbitrarily composable, stage interdependence minimal (Interface segregation)
- Extension is easy, even if current stages' code is unknown (Open/closed principle)
- Experiment management doesn't have to be hardcoded or manually managed
- Much more to it:
  - DVCLive Python module, e.g., for usage with jupyter notebooks
  - DVC extension for vscode
  - Support for hydra.cc: https://doc.dvc.org/user-guide/experiment-management/hydra-composition

6. Software Engineering best practices

Version Control: git

What is this?

A tool to allow you to track and revert changes, and collaborate with others (change management).

Why is this important?

- Allow versioning of the code and continuing functionality.
- Allow simultaneous changes to the same files.
- Fundamental for reproducing historic states in the line of development.
- Development follows a story and allows other users to build confidence in your work.

Version Control in practice: git

- Create a repo on GitHub
- Clone the repo to your local machine
- Checkout a branch and make changes
- `git add`, `git commit` and `git push`: IDEs such as VSCode make it easy for you!
- Observe how your repo changes on GitHub
- GitHub offers great learning labs: GitHub skills https://skills.github.com/
- roadmap.sh offers a git roadmap: https://roadmap.sh/git-github

Development workflows: GitHub-flow

What is this?

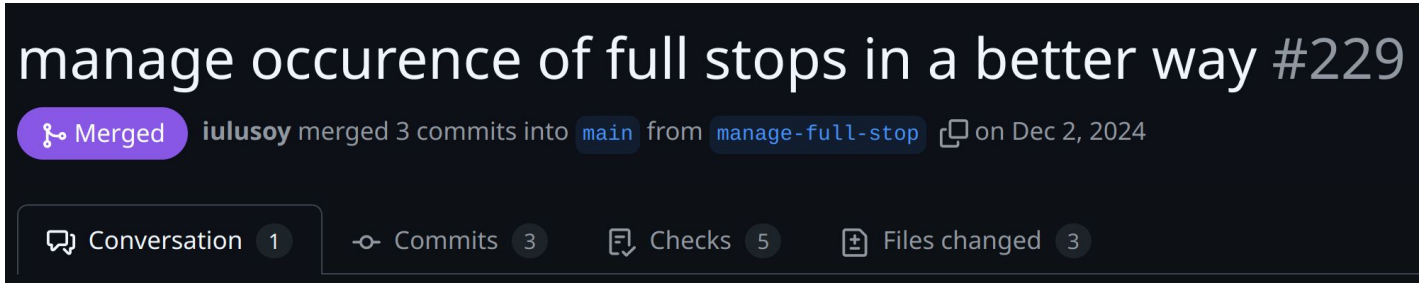
GitHub-flow is a lightweight workflow with creating branches, making changes, opening Pull Requests, running Continuous Integration, and requesting code review, together with Issues and Kanban project boards.

Why is this important?

- Manage how changes are incorporated in the software.
- Track progress in your project and highlight bottlenecks.
- Adhere to development guidelines, ensuring the implementation follows defined rules to ensure software quality.

Development workflows in practice: GitHub-flow

- After you made changes in your git branch, open a Pull Request on GitHub
- Observe how the PR highlights the changes in the line of development
- You can link issues, comment on the PR and run automated checks
- See GitHub skills https://skills.github.com/
- See roadmap.sh https://roadmap.sh/git-github



Requirements engineering and continuous delivery

What is this?

Requirements engineering is the process of translating stakeholder requirements on the research software into defined tasks. Early delivery and iteration over it allows refinement of the requirements and tasks.

Why is this important?

- Ensure that the software fulfills its purpose. In research software, requirements engineering is closely intertwined with the research process and subject to frequent changes.
- Understand the problem the software should solve and map this onto an efficient technically feasible solution considering all constraints.
- Allows prioritization of requirements/tasks and decision-making. Decisions should be documented together with the requirements.

Requirements engineering in practice

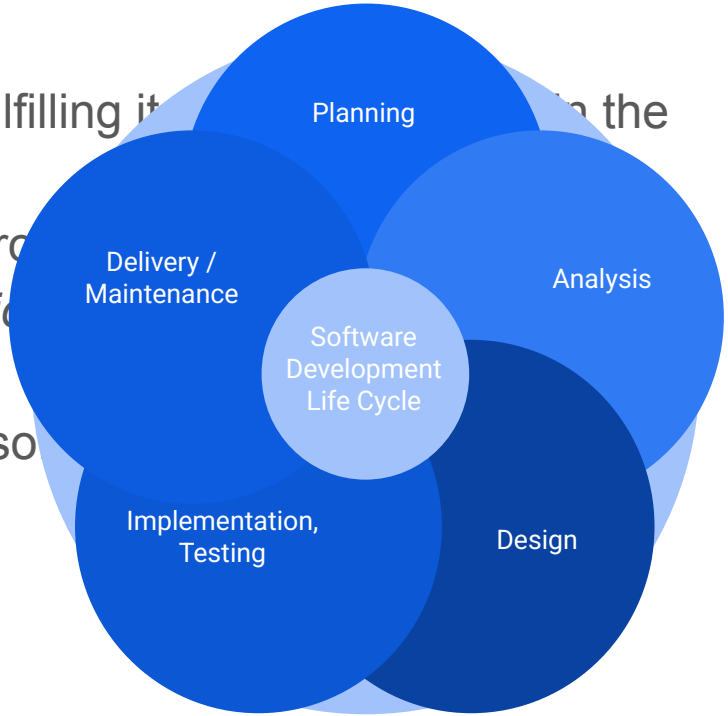
- Functional requirements (what the software should do: features)
- Non-functional requirements (how the software performs a task: ie performance, security)
- Domain requirements (specific to the domain: ie. Healthcare)
- Use tools such as draft.io or miro to gather requirements

As a <type of user>, I want <some goal> so that <some reason>.

As a <researcher>, I want <to obtain the probability of a class> so that <I can classify incoming images>.

Continuous delivery in practice

- Deliver early to find out if your software is fulfilling it[illegible] the right direction
- Use quality control to allow early delivery thro[illegible] GitHub-flow (*keep your main branch operati[illegible]*
- Following agile / lean principles
- Use git to allow consistent usability of your so[illegible]



Project management: Kanban boards

What is this?

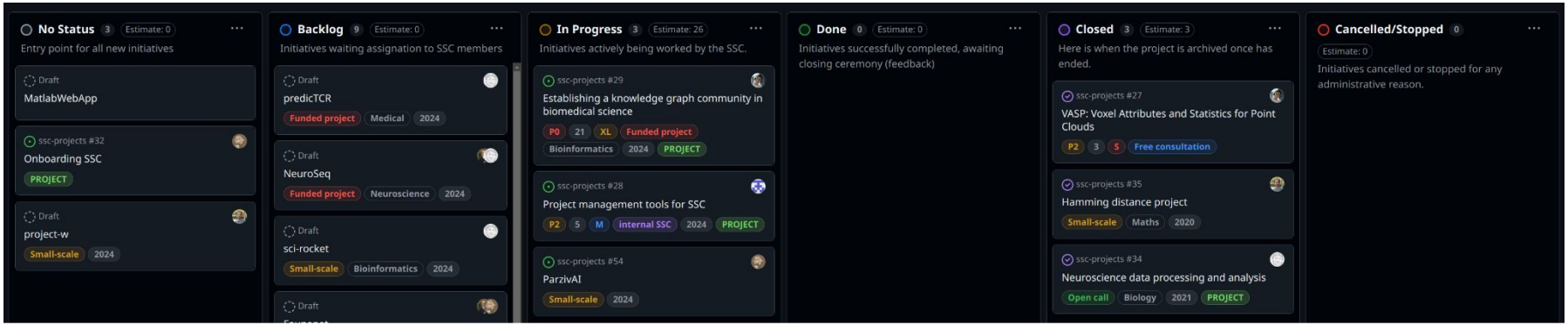
Project management is used to track progress, identify intertwined or dependent processes, and allows visual access to the project's status.

Why is this important?

- Ensure that the software development moves in the right direction.
- Increase the flow of ongoing work.
- Allow prioritization of tasks and understand interdependencies and bottlenecks in the development.

Project management in practice

- Use a Kanban board to organize tasks
- Separate tasks into backlog/todo, in progress, done
- Prioritize tasks and assign contributors/identify necessities
- For example, GitHub projects



Project planning: Architecture and design

What is this?

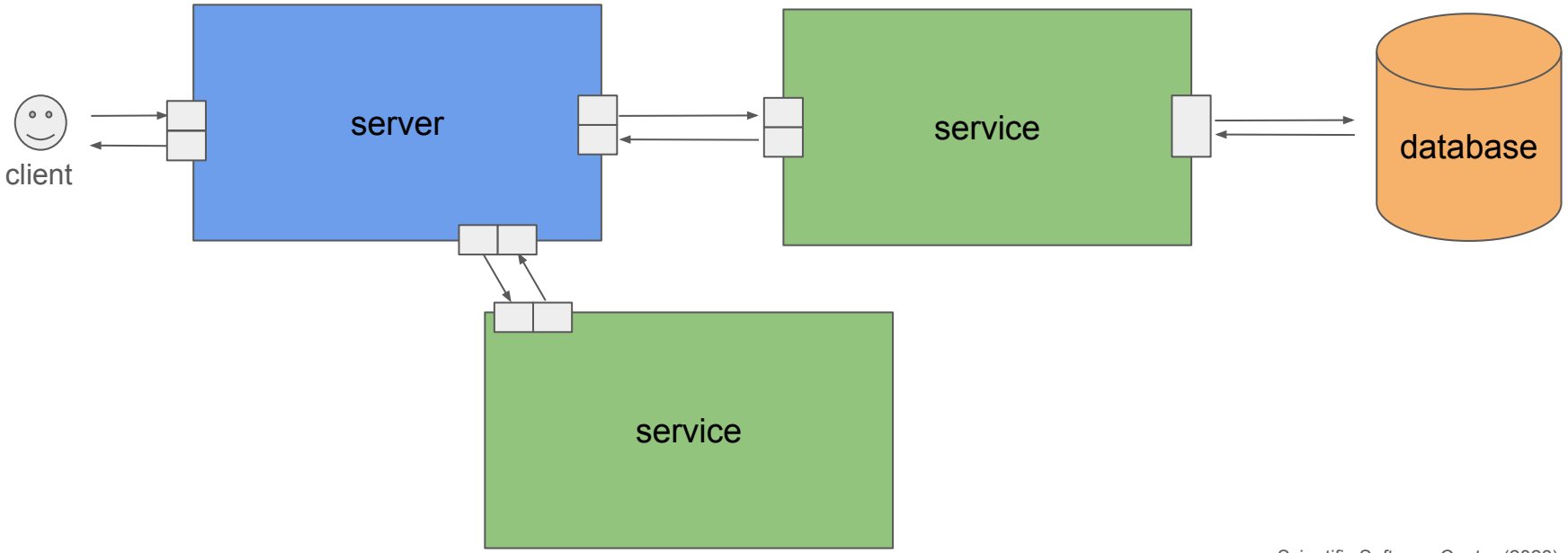
Software architecture describes how the system is composed of different pieces, and the interplay of the components. Design refers to the actual implementation of the requirements in the system as a whole and the different components.

Why is this important?

- Makes the software efficient and allow re-use of functionalities.
- Allows extensions and additions of features at a later stage without major refactoring.
- Makes the software maintainable.

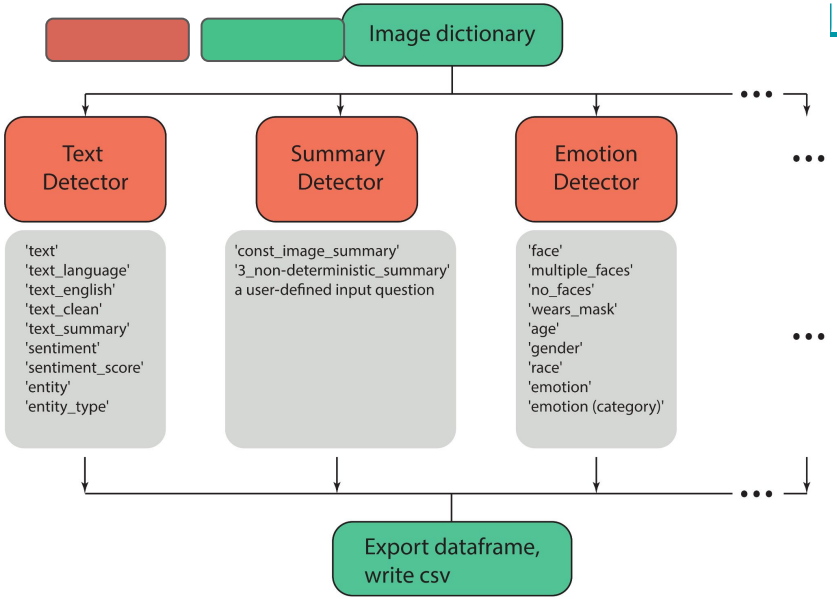
Architecture in practice

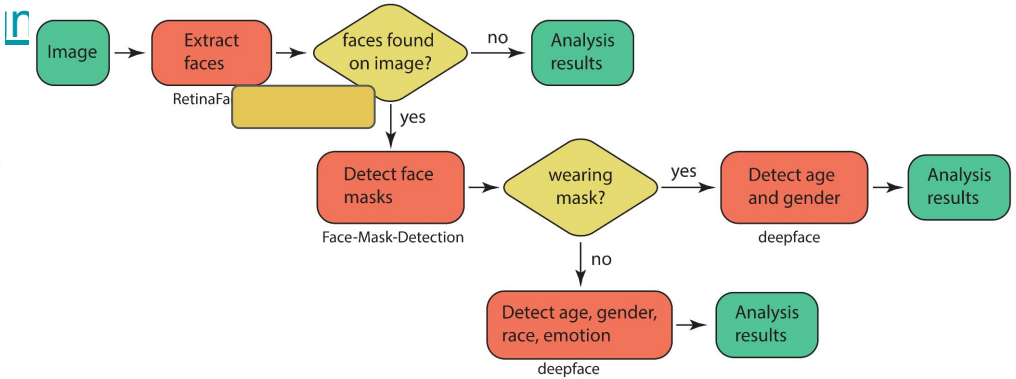
- Use (black/white) box diagrams to identify components and their interactions
- https://roadmap.sh/software-design-architecture / miro / draft.io / draw.io



Design in practice

- Map the input/output, data formats, transformations / logic / processes






```
image_summary_vqa_detector = ammico.SummaryDetector(image_dict,
    analysis_type="summary_and_questions",
    model_type="base")
for key in image_dict.keys():
    image_dict[key] = image_summary_vqa_detector.analyse_image(
        subdict=image_dict[key], analysis_type="summary_and_questions",
        list_of_questions = list_of_questions)
```

Quality management: Testing and continuous integration

What is this?

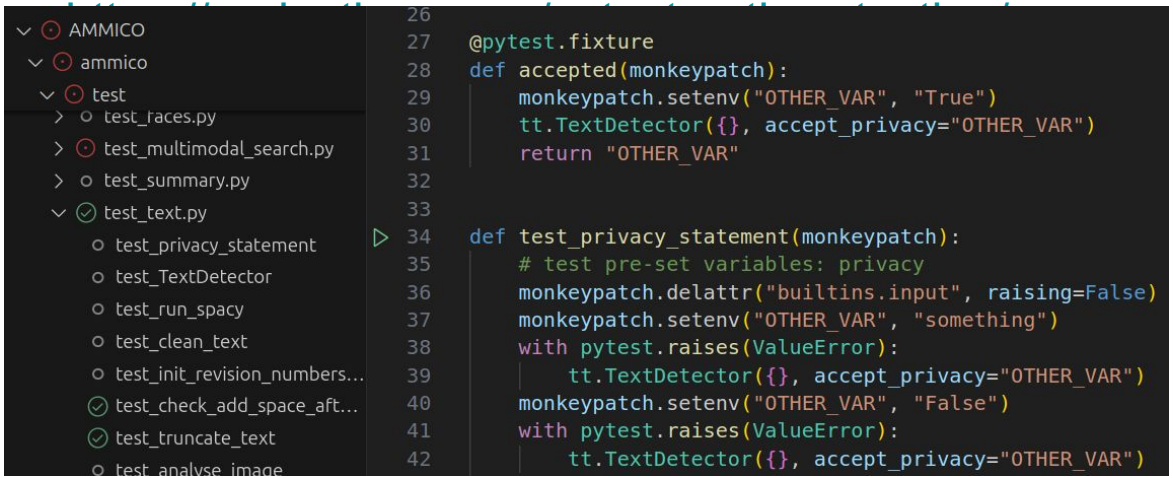
Quality management ensures that the software is producing correct results and that the results are reproducible.

Why is this important?

- As a scientist, you should take utmost care to conduct your science in accordance with the scientific code of conduct.
- Yourself or other researchers may need to reproduce results for further studies.
- Ensure that nothing breaks when you implement changes/new features.

Testing in practice

- Use testing frameworks such as `pytest`
- Write tests in a `tests/` folder: unit tests, integration tests, system tests, compatibility tests, …
- To learn how to use pytest: https://docs.pytest.org/en/stable/,



Testing of non-deterministic processes

Try to make processes deterministic
For example: Use specified random seed.

Separate deterministic and non-deterministic processes and test separately
For example: Input processing can be tested separately from model prediction.

Test for output parameters and properties that remain constant
For example: Number of predictions, feature length, etc.

Include multiple valid outputs in your tests
For example: Three most probable classifications.

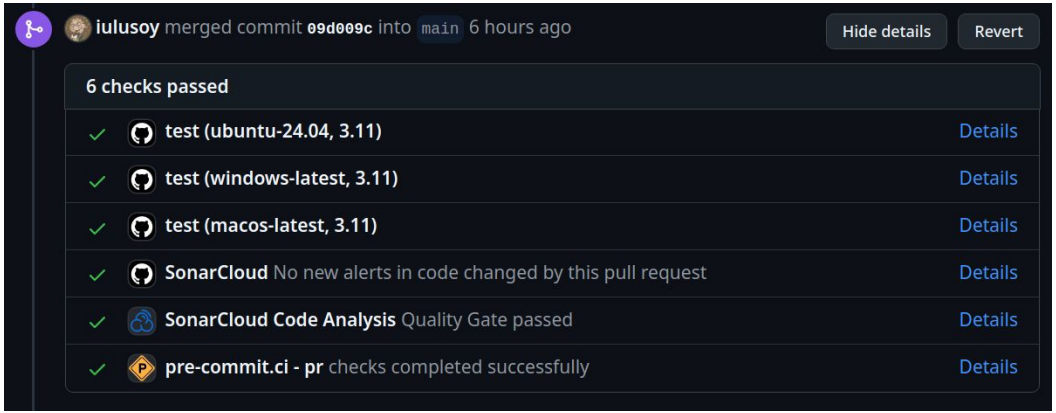
Robustness: A robust model is more likely to behave like a deterministic system
Make sure your model output is stable under a range of conditions.

Accuracy: The model accuracy will affect the testing strategy
A higher accuracy leads to more consistent predictions.

Distribution: You can also test for the distribution of results rather than the results themselves

Continuous integration in practice

- Set up your tests to be automatically run by GitHub actions
- Include code linter and quality control in your actions
- These should be set up to run automatically when you open a Pull Request
- GH actions, `codecov`, `sonarcloud`, `snyk`, `pre-commit`, code formatting (`black`), GitHub Guardian, `dependabot`



Software Management Plans

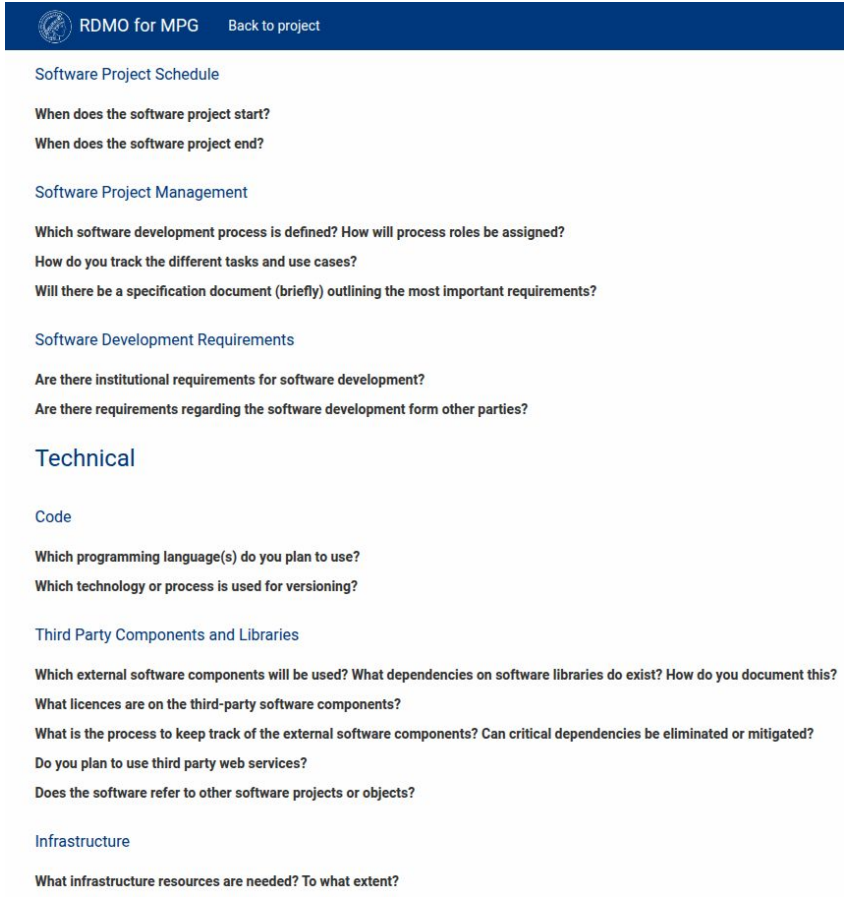
What is this?

Software Management Plans (SMPs) help to identify goals and the means required to pursue the goals in practice.

Why is this important?

- Identify criticality and required maturity of your software.
- Identify which measures are needed to ensure compliance of your software with the intended goals.
- Quantify milestones and tools for the intended purpose.

SMPs in practice



- Use the SMPs provided by the Max Planck digital library
- Helps you with your requirements and project management
- https://rdmo.mpdl.mpg.de/

Documentation

What is this?

Documentation can be comments, docstrings, readme's, tutorials, demonstration notebooks, and contains technical and domain-specific / application-specific descriptions of the software.

Why is this important?

- Document what the software can and cannot do, and parameter ranges.
- Allow others to install and use your software (or yourself, at a later time).
- Allow others to contribute to your software.

Documentation in practice

text module

class **text.PostprocessText**(*mydict: dict | None = None, use_csv: bool = False, csv_path: str | None = None, analyze_text: str = 'text_english'*)

Bases: `object`

analyse_topic(*return_topics: int = 3*) → tuple

Performs topic analysis using BERTopic.

Parameters: **return_topics** (*int, optional*) – Number of topics to return. Defaults to 3.
Returns: **tuple** – A tuple containing the topic model, topic dataframe, and most frequent topics.

get_text_df(*analyze_text: str*) → list

Extracts text from the provided dataframe.

Parameters: **analyze_text** (*str*) – Column name for the text field to analyze.
Returns: **list** – A list of text extracted from the dataframe.

get_text_dict(*analyze_text: str*) → list

Extracts text from the provided dictionary.

Parameters: **analyze_text** (*str*) – Key for the text field to analyze.
Returns: **list** – A list of text extracted from the dictionary.

- Use tools like `sphinx` and `mkdocs` to render docstrings and markdown at minimal effort
- Include jupyter notebooks that showcase use of your software - these can be run on google colab
- Document dependencies in a requirements file and provide installation instructions

Deployment: Runtime environment / containerisation

What is this?

Deployment information such as runtime environments or containers allow easy adaption as they provide direct access to running the software without installation and dependency conflicts.

Why is this important?

- A big step towards reproducibility and transferability of your approach.
- The software ecosystem changes quickly, and this allows to preserve a snapshot that can be shared and run easily.

Containerisation in practice

- Use docker to provide build instructions for containers, and possibly deploy the containers on Dockerhub for anyone to download and use
- Docker roadmap https://roadmap.sh/docker, official tutorial https://docs.docker.com/

Dockerfile > FROM

```
1  FROM jupyter/base-notebook
2
3  # Install system dependencies for computer vision packages
4  USER root
5  RUN apt update && apt install -y build-essential libgl1 libglib2.0-0 libsm6 libxrender1 libxext6
6  USER $NB_USER
7
8  # Copy the repository into the container
9  COPY --chown=${NB_UID} . /opt/ammico
10
11 # Install the Python package
12 RUN python -m pip install /opt/ammico
13
14 # Make JupyterLab the default for this application
15 ENV JUPYTER_ENABLE_LAB=yes
16
17 # Export where the data is located
18 ENV XDG_DATA_HOME=/opt/ammico/data
```

Software Licensing

What is this?

A software license states the terms of use, re-use and distribution, among others, without violating copyrights, and defines responsibilities.

Why is this important?

- So that others may use your code, and to prevent misuse.
- So that others may contribute to your code.
- So that the responsibilities for how the software is used are clear.
- Establishes the rights of all parties involved with the software.

Software licensing in practice

- Use the provided templates from GitHub: Either at repository creation or when adding a new file called LICENSE
- Permissive open-source license: BSD 2-Clause, MIT, Apache License 2.0
- Copyleft open-source license: GNU version 3, LGPL
- Proprietary licenses: Do not only keep your project close-source and potentially less visible, but also carry responsibilities for contract fulfillment
- https://opensource.org/licenses, https://choosealicense.com/
- **When using/incorporating third-party software**: ie. use of open-source libraries - is the third-party code distributed with your software? If so, compatibility needs to be confirmed!

Tooling and generative AI

Tooling: Integrated Development Environment (IDE)

What is this?

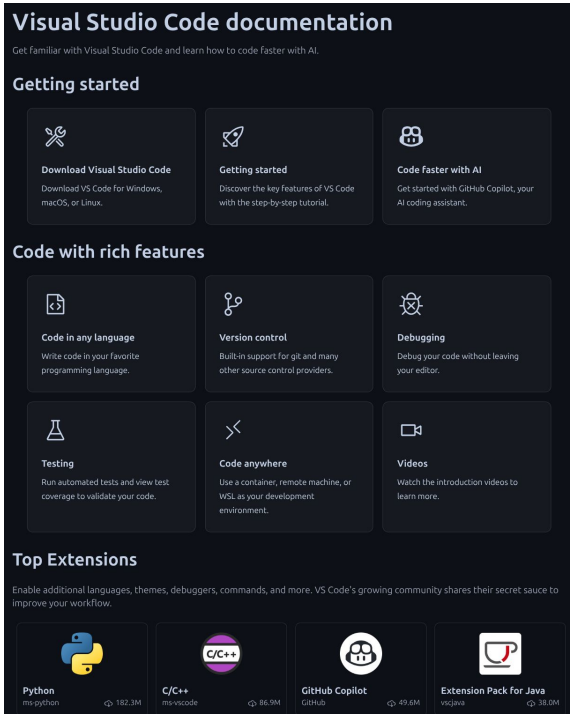
An IDE is a software application that combines tools for software development, such as a code editor, linter, compiler, debugger, and automation utilities.

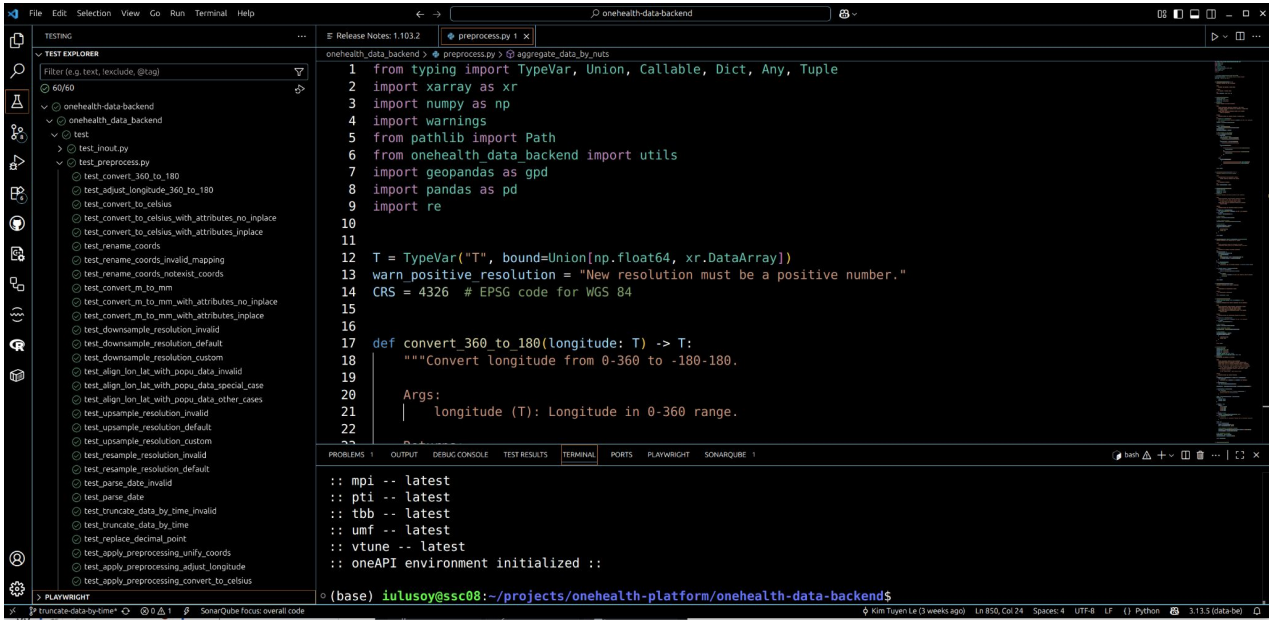
Why is this important?

- Provides you with all required tools in one place, making software development more efficient.
- Helps you keep your code clean.
- Many different IDE's available: https://github.com/zeelsheladiya/Awesome-IDEs

IDEs in practice

- For example: VSCode https://code.visualstudio.com/



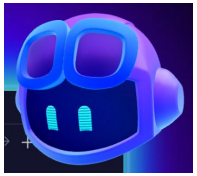


AI-supported software development

What is this?

The software development process can be supported by AI in many ways:

- Code completion
- Code generation
- Code analysis
- Code quality
- Documentation
- Debugging
- Security enhancement
- Architecture/UX design
- Full-cycle development (AI agents)



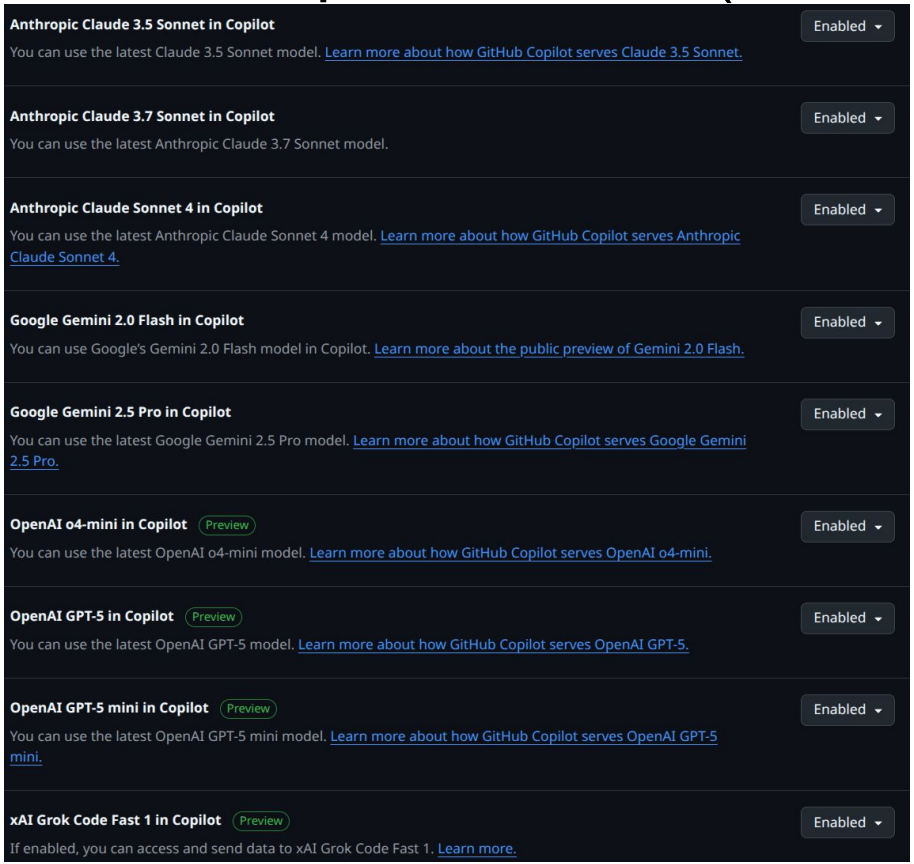
Why is this important?

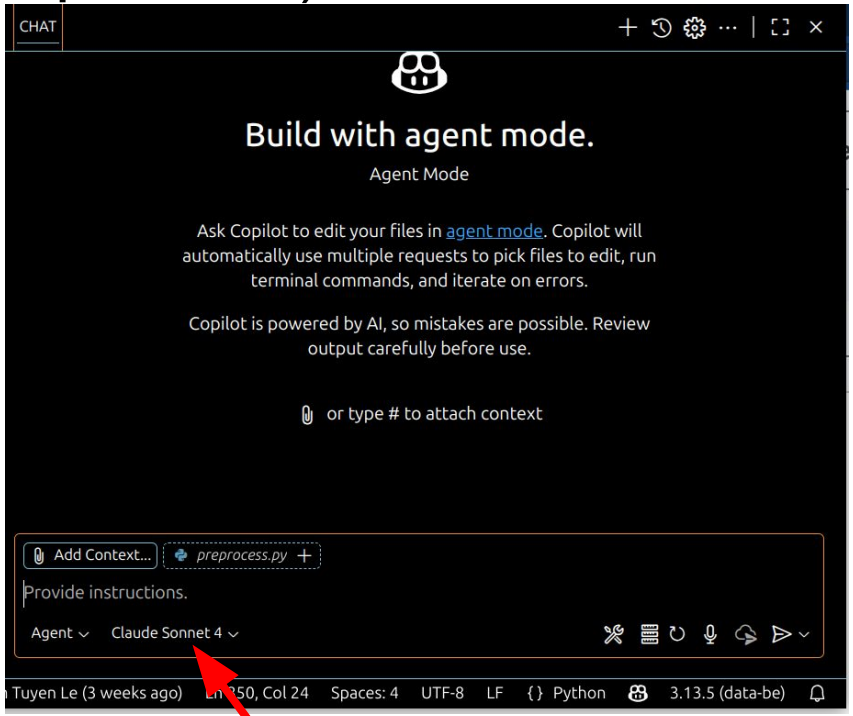
- AI has become an integral part of software development and can help you write better code.
- You can improve your skills by interacting with AI systems as you learn to understand why you should write something like this, or also why not.

AI-supported software development in practice

- For example: GitHub Copilot https://github.com/features/copilot - get free access as a student https://github.com/education - many other tools available as well, for example
  - Code analysis: ie. SonarQube Cloud integration to GitHub, snyk
  - Security enhancement: ie. snyk
  - UX design: ie. Figma
  - A list of tools: https://github.com/jamesmurdza/awesome-ai-devtools, https://github.com/ikaijua/Awesome-AITools
- Copilot is integrated into VSCode: Code completion, code generation, code summarization, generation of docstrings and documentation content
- Chat mode or agentic mode
- Also allows to chats on GitHub and can carry out Pull Requests by itself through agentic mode

GitHub Copilot: Models (as of Sept. 2025)





select model

AI-supported software development: User roles

Chat user, ie. ChatGPT	Copilot user, ie. code completion and code generation	Chat-and-Copilot user, ie. code completion and code generation iterates with chat	Agentic user up to vibe coder
- Get help and learn through explanations and summarization	- Increase efficiency through passing over cumbersome tasks such as boilerplate code, docstring generation or straightforward pieces of code	- Increase efficiency and learn through interacting more closely with the LLM, refactoring suggestions in an iterative manner	- Pass of responsibility for pieces of code, or complete parts of the code to the AI agent without following the AI implementation in detail

AI-supported software development: caveats

- Not understanding your code
- Creating code that is harder to maintain
- Mixing of code for different releases of the same library (ie. langchain 0.2.x, 0.3.x, especially in frontend development)
- Rewriting code that is actually provided by a library through a simple keyword
- Subtle hallucinations that introduce hard-to-spot mistakes
- AI only as good as the context
- Does not generate code in the same style as existing codebase (lack of context-awareness and preservation)
- And of course, copyright, legal and ethical considerations

Starting a new project

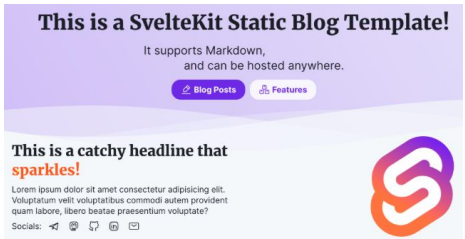
Project templates

What is this?

Project templates exist for example on GitHub. They include configurations for specific types of projects.

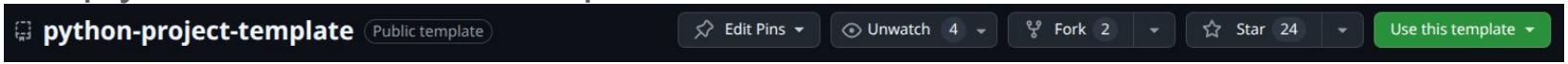
Why is this important?

- Project templates help you save time by providing complete configurations for testing, CI, and documentation, for example.
- Templates help you adhere to common practices, configurations, or

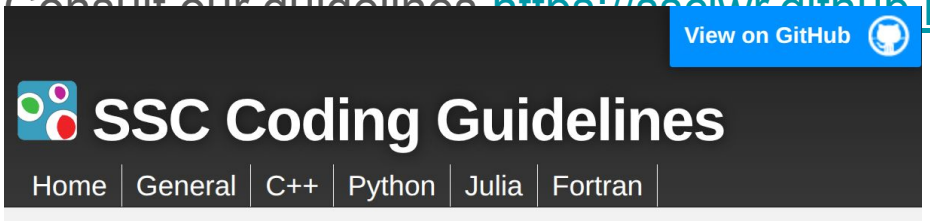


Project templates in practice

- For example: GitHub template repositories
- Simply click on “use this template”

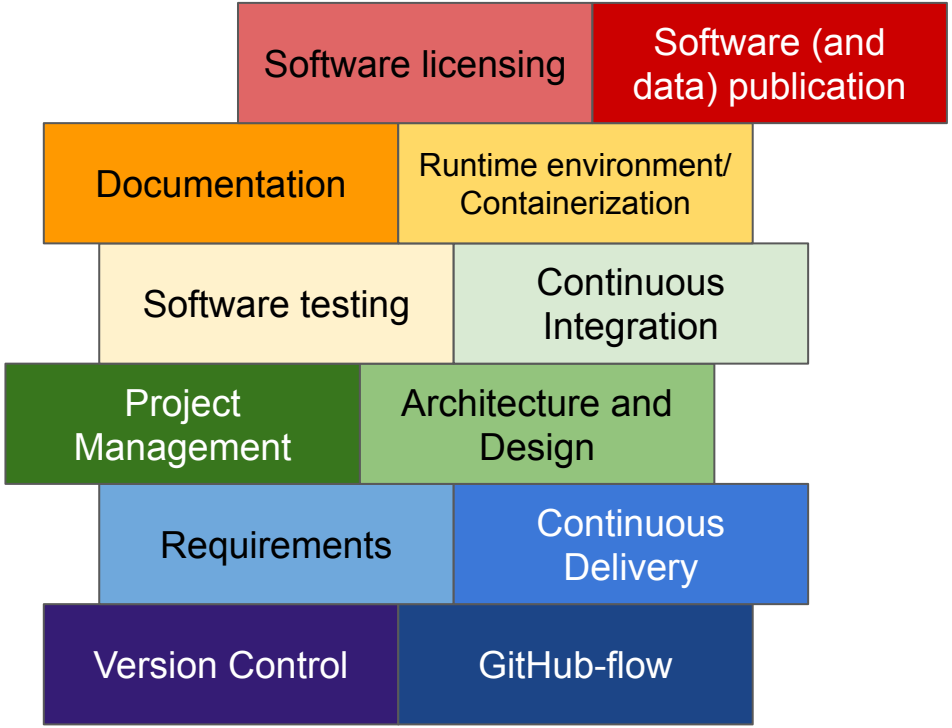


- More configurable templates: cookiecutters (also available for data science)
- The SSC offers such template repositories for
  - C++
  - Python
  - Fortran
  - Matlab
- Consult our guidelines https://escoiwr.github.io/guidelines/



Summary: Software Engineering best practices

Best practices: Building blocks for your software project



7. Publishing, interoperability and responsible AI (RAI)

Software publication

What is this?

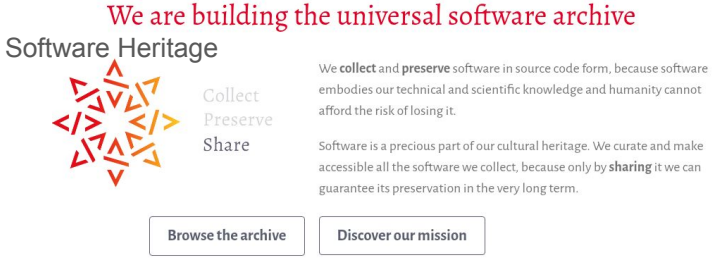
A software publication associates a DOI or other persistent identifier with your software.

Why is this important?

- This makes your software citable in scientific publications.
- You can reference your software plus providing the version number that you used.
- Your contribution to the scientific community becomes more visible.
- The publication helps you to structure the code and documentation for a user’s perspective.

Software publication in practice

- A list of software-specific journals: https://www.software.ac.uk/top-tip/which-journals-should-i-publish-my-software
- Write a short paper detailing your software, the architecture and design, in- and output, and purpose of the software
- Another option is to place a snapshot of your repo on zenodo, this also allows you to obtain a DOI
- Another option is to upload your project to Software Heritage and obtain a persistent SWHID https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html



Data publication

What is this?

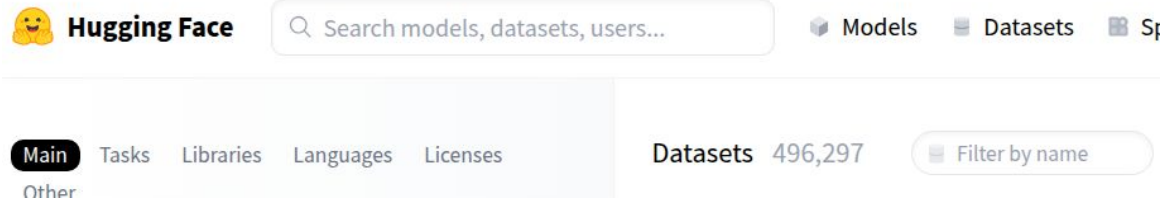
Data can be published in dedicated data repositories and obtain a DOI.

Why is this important?

- You can publish your datasets, making them available to the community.
- By associating a DOI, you may garner citations.
- Data often is the foundation for reproducing results.
- Data should be stored and preserved separately from the software once it reached a stage that it does not change anymore.

Data publication in practice

- Option A: Publish your data on zenodo and obtain a DOI
- Option B: Publish your data in your institutionally provided data repository, ie heiDATA and obtain a DOI
- Option C: Publish your data on community platforms such Hugging Face *if these provide a DOI*
- Make sure your data is clean and does not need to be updated anymore
- Cite your own data repository in your publications
- You can also download your own data dynamically within your code, ie,. using pooch https://www.fatiando.org/pooch/dev/ or datasets https://huggingface.co/docs/datasets/en/index



Why share your data?

- Create a larger impact of your scientific work
- Increase transparency and trust in your work
- Enable others to reproduce and build upon you
- Contribute to science as a whole
- BUT: Be careful with legal (copyright) and ethic advisor/institution.
- Follow the FAIR principle: Findable, Accessible

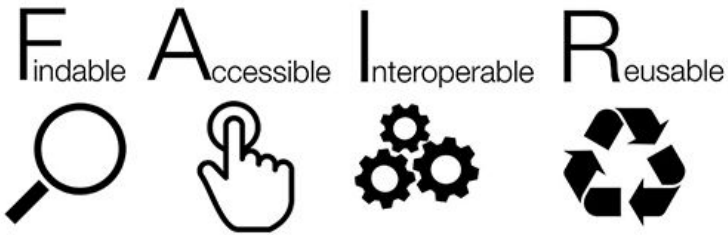


Image credit: SangyaPundir shared under CC-BY-SA 4.0
The FAIR Guiding Principles for scientific data management and stewardship
https://doi.org/10.1038/sdata.2016.18

Where to share your data

- HeiDATA
- Open Science Framework
- Zenodo
- Figshare
- Dryad
- Hugging face
- Kaggle: no DOI!
- Domain-specific repositories
  - i.e. https://lindat.mff.cuni.cz/repository/xmlui/
  - see https://www.nature.com/sdata/policies/repositories for a list
- For a comparison, see DOI: 10.5281/zenodo.3946720

Model sharing platforms

You can make models available for others on model sharing platforms like

- Hugging face,
- OpenML,
- Kaggle.

Advantages: Public platform with version control and model cards, you can link the data into the repo, allows others to use your model for production or fine-tuning.

Model card

- Model details
  - Architecture, parameters, citation information, license information
- Intended use
  - Use cases within the model's scope
- Performance metrics
  - Intended performance on given data
- Training data
  - Description of training data and data distribution
- Quantitative analysis
  - Potential biases and limitations
- Ethical consideration
  - Privacy and fairness concerns, impact on society
- https://huggingface.co/spaces/huggingface/Model_Cards_Writing_Tool
- https://github.com/openai/gpt-3/blob/master/model-card.md

Model deployment

In addition to making models and software available for others to use in their own code, you can also directly deploy the model - together with your code - directly so that it can be used.

Examples:

- Diffusers: google colab https://colab.research.google.com/github/huggingface/notebooks/blob/main/diffusers/stable_diffusion.ipynb
- https://lightning.ai/ for paid service and deployable models
- See https://www.freecodecamp.org/news/deploy-your-machine-learning-models-for-free/ for tutorials and services

REFORMS checklist

To ensure reproducibility, include a checklist such as the REFORMS checklist into the publication of your results.

There are a couple of domain-specific checklists, especially in health and life sciences (STARD, CLAIM, see the EQUATOR* network), that may be more appropriate - REFORMS aims to be most general.

REFORMS checklist: DOI 10.48550/arXiv.2308.07832, https://reforms.cs.princeton.edu/
*EQUATOR Network, https://www.equator-network.org/reporting-guidelines/

Interoperability and RAI

Interoperability

- Ability to interact with other systems interchangeably (i.e. transformers, pytorch)
- Exchange information with other systems
- Integrate into other systems

⇩

Use data and models in different contexts

RAI

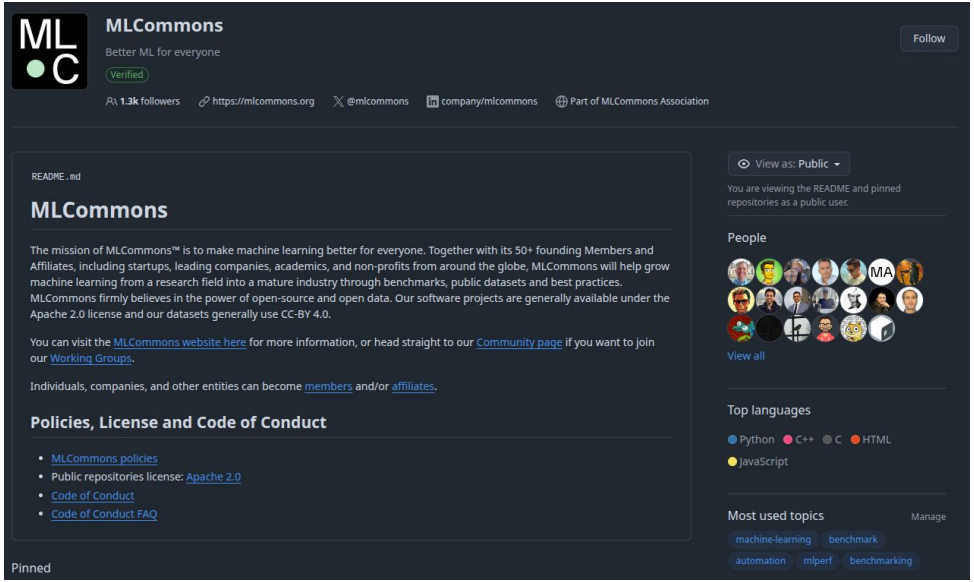
- Develop, assess, deploy AI systems safe and ethically
- Prioritize equitable and beneficial outcome of the AI system in the system development
- people and their goals at the center of design

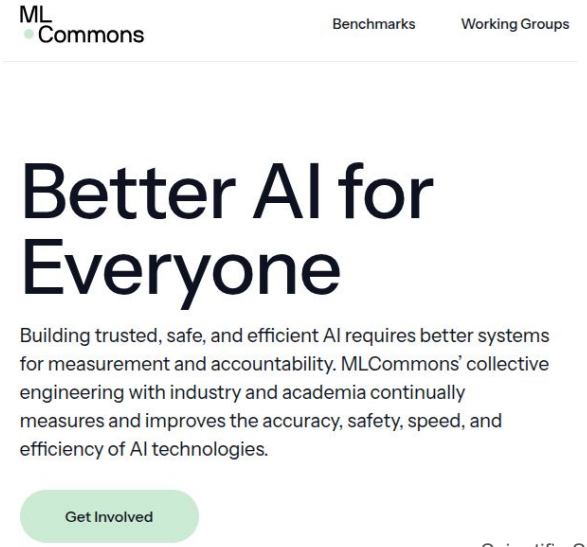
⇩

fairness, reliability, transparency, safety, privacy, security, inclusiveness, accountability

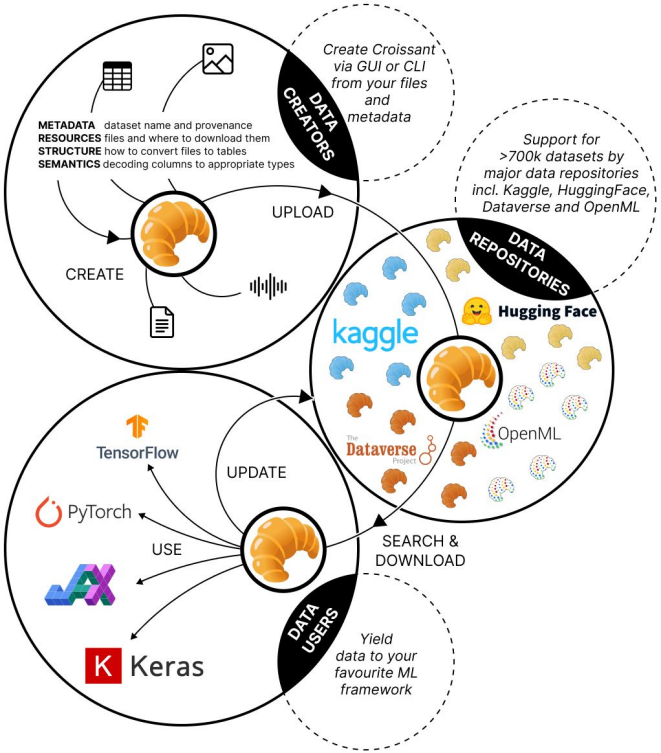
MLCommons

- To accelerate artificial intelligence innovation and increase its positive impact on society
- Targets industry and academia
- Democratize machine learning through open industry-standard benchmarks that measure quality and performance and build open, large-scale, and diverse datasets to improve AI models





Croissant: an interoperable data format



Croissant 🥐 is a high-level format for machine learning datasets

Metadata: standardized description of the dataset, including responsible ML aspects

Resources: one or more files or other sources containing the raw data

Structure: how the raw data is combined and arranged into data structures for use

ML semantics: how the data is most often used in an ML context

Find, inspect, and use the data in your favorite ML framework!

https://github.com/mlcommons/croissant
https://doi.org/10.52202/079017-2610

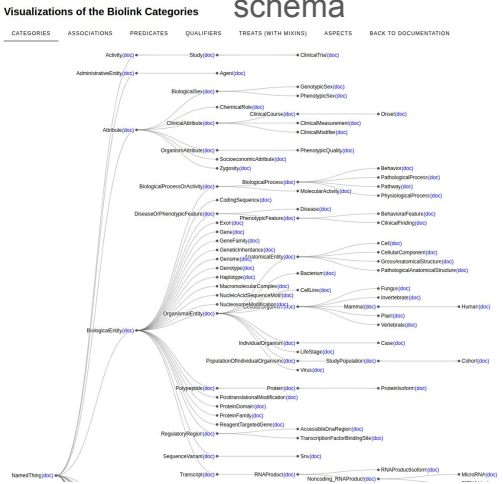
Croissant file

```
"@type": "sc:Dataset",
"dct:conformsTo": "http://mlcommons.org/croissant/1.0",
"name": "Collectri",
"description": "CollecTRI dataset",
"version": "0.0.1",
"license": "https://github.com/saezlab/CollecTRI/blob/main/LICENSE",
"url": "https://github.com/saezlab/CollecTRI",
"keywords": [
  "multi-omics",
  "regulatory interactions"
],
"datePublished": "2024-05-12",
"creator": [
  {
    "@type": "sc:Person",
    "name": "Sebastian Lobentanzer",
    "affiliation": "Helmholtz Munich",
    "identifier": "https://orcid.org/0000-0003-3399-6695"
  }
],
"citeAs": "This adapter was developed by Sebastian Lobentanzer to load and transform the Collectri dataset. For dataset c
"conformsTo": "http://mlcommons.org/croissant/1.0",
"distribution": [
  {
    "@type": "cr:FileObject",
    "@id": "collectri.csv",
    "name": "CollecTRI.csv",
    "contentUrl": "https://rescued.omnipathdb.org/CollecTRI.csv",
    "encodingFormat": "text/csv",
    "sha256": "86c90b30f2cc75c189da1f0a8c353d1547287cd656a9fac1c678634285bcb4e0"
  }
],
"recordSet": [
  {
```

Metadata layer (name, description, license)

Resources layer (source data included in dataset)

an ontology for a schema

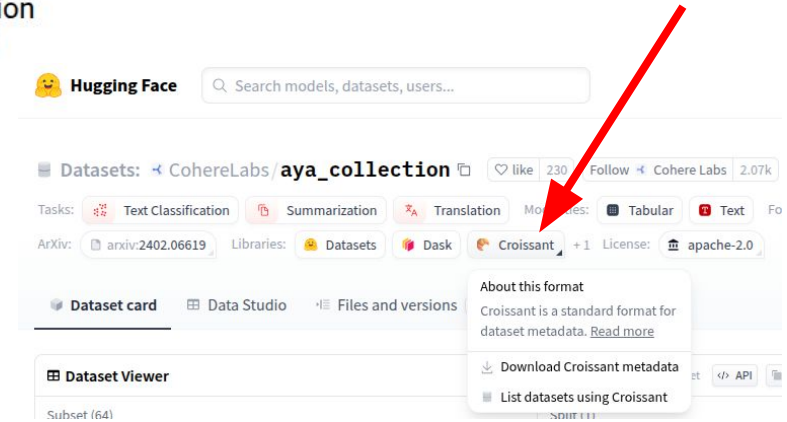


Croissant file


```
"recordSet": [
  {
    "@type": "cr:RecordSet",
    "@id": "collectri",
    "name": "CollecTRI",
    "description": "Transcription factor → target gene regulatory interactions with supporting evidence.",
    "field": [
      {
        "@type": "cr:Field",
        "@id": "collectri/source",
        "name": "source",
        "description": "Gene symbol of transcription factor (TF).",
        "dataType": "sc:Text",
        "examples": [
          "MYC"
        ],
        "source": {
          "fileObject": {
            "@id": "collectri.csv"
          },
          "extract": {
            "column": "source"
          }
        }
      },
      {
        "@type": "cr:Field",
        "@id": "collectri/target",
        "name": "target",
        "description": "Gene symbol of the regulated target gene.",
        "dataType": "sc:Text",
        "examples": [
          "TERT"
        ],
        "source": {
          "fileObject": {
            "@id": "collectri.csv"
          },
          "extract": {
            "column": "target"
          }
        }
      },
```


Structure Layer (data as RecordSets, data mapping and transformations)

Semantic layer (custom data types, dataset organization methods)



Work with croissant datasets

Loading:

```python
import mlcroissant as mlc

# load the collectri dataset
collectri_dataset = mlc.Dataset("../biocypher-components-registry/external_repos/adapter_collectri/collectri.json")
```

Using as a base for training:

```python
import tensorflow_datasets as tfds
import torch
from tqdm import tqdm

# Use Croissant dataset in your ML workload
builder = tfds.core.dataset_builders.CroissantBuilder(
    jsonld="../biocypher-components-registry/external_repos/adapter_collectri/collectri.json",
    record_set_ids=["collectri"],
    file_format='array_record',
)
builder.download_and_prepare()

# Split for training/testing and use as_dataset directly from the builder
train = builder.as_data_source(split='default[:80%]')
test = builder.as_data_source(split='default[20%:]')

ds = builder.as_data_source()
```

```python
# now load into pytorch
batch_size = 128
train_sampler = torch.utils.data.RandomSampler(train, num_samples=5_000)
train_loader = torch.utils.data.DataLoader(
    train,
    sampler=train_sampler,
    batch_size=batch_size,
)
test_loader = torch.utils.data.DataLoader(
    test,
    sampler=None,
    batch_size=batch_size,
)

class LinearClassifier(torch.nn.Module):
  def __init__(self, length, num_classes):
    super(LinearClassifier, self).__init__()
    self.classifier = torch.nn.Linear(length, num_classes)

  def forward(self, feature_x):
    # Convert to float32 if needed
    feature_x = feature_x.to(torch.float32)
    # Pass through the linear layer
    return self.classifier(feature_x)

# Create label mapping
print('Creating label mapping...')
label_mapping = {}
for example in train_loader:
  for label in example['sign_decision']:
    if label not in label_mapping:
      label_mapping[label] = len(label_mapping)
print(f"Label mapping: {label_mapping}")

# Update model with correct number of classes
num_classes = len(label_mapping)
model = LinearClassifier(length=1, num_classes=num_classes)
optimizer = torch.optim.Adam(model.parameters())
loss_function = torch.nn.CrossEntropyLoss()

print('Training...')
model.train()
```

- use tfds builder for loading
- builder compatible with tensorflow, pytorch, JAX
- eclair MCP server to use croissant with an AI agent
- Very active and open group of

Contributors

Albert Villanova (Hugging Face), Andrew Zaldivar (Google), Baishan Guo (Meta), Carole Jean-Wu (Meta), Ce Zhang (ETH Zurich), Costanza Conforti (Google), D. Sculley (Kaggle), Dan Brickley (Schema.Org), Eduardo Arino de la Rubia (Meta), Edward Lockhart (Deepmind), Elena Simperl (King's College London), Goeff Thomas (Kaggle), Joan Giner-Miguelez (UOC), Joaquin Vanschoren (TU/Eindhoven, OpenML), Jos van der Velde (TU/Eindhoven, OpenML), Julien Chaumond (Hugging Face), Kurt Bollacker (MLCommons), Lora Aroyo (Google), Luis Oala (Dotphoton), Meg Risdal (Kaggle), Natasha Noy (Google), Newsha Ardalani (Meta), Omar Benjelloun (Google), Peter Mattson (MLCommons), Pierre Marcenac (Google), Pierre Ruyssen (Google), Pieter Gijsbers (TU/Eindhoven, OpenML), Prabhant Singh (TU/Eindhoven, OpenML), Quentin Lhoest (Hugging Face), Steffen Vogler (Bayer), Taniya Das (TU/Eindhoven, OpenML), Michael Kuchnik (Meta)

Thank you for supporting Croissant! 🙂

RAI

Croissant RAI specification https://docs.mlcommons.org/croissant/docs/croissant-rai-spec.html

Use case 1: The data life cycle (level: dataset)

Key stages of the dataset life cycle include **motivation, composition, collection process, preprocessing/cleaning/labeling, uses, distribution, and maintenance** [5]. Documenting RAI-related properties of the dataset can encourage its creators to reflect on the process and improve understanding for users.

Information generated throughout the cycle addresses different aspects requiring consideration for responsible data usage, including (1) information regarding who created the dataset for which purpose, (2) information when the dataset was created, (3) which data sources were used, (4) information on the versioning of the dataset with timestamps for each version (5) how the data is composed and if it contains noise, redundancies, privacy-critical information, etc. (6) how data was processed (e.g. also containing information on crowdsourcing - see use case 2), (7) how the data is intended to be used, (8) how the dataset will be maintained. In conjunction, properties for documenting the provenance and lineage of the datasets that are derived from revision, modification or extension of existing datasets are also relevant for this use case.

We anticipate that this use case will be covered by the core vocabulary. The main purpose of the use case is to understand if there are any additional properties currently not included in Croissant.

Use case 2: Data labeling (level: dataset or record)

A portion of the metadata at dataset level will be aggregated from record-level annotations. These can be achieved either through some form of human input, in particular labels and annotations created via labeling services, including, but not restricted to crowdsourcing platforms (e.g. what platform has been used), how many human labels were extracted per record, any demographics about the raters if available, or by machine annotations (e.g. concept extraction, NER, and additional characteristics of the tools used for annotation to allow for replication or extension). These are important to automatically create distribution characteristics of datasets to better understand its composition, but also to be able to efficiently sample from different datasets. Information about the labeling process helps **understand how the data was created, the sample the labels apply to,** hence making the process easier to assess, repeat, replicate, and reproduce. This increases the reliability of the resulting data.

Software security

- Threat modelling: who, what, how
- Data-oriented attack: Access training data, poison data, inject trojan data
- Model-oriented attack: Modify training process (pre-trained malicious models), manipulate the deployed model (model patches, privacy information leakage, model inversion attacks)
- System-oriented attack: specialised hardware accelerators for ML software (SOC, trojan in GPU/TPU, for model corruption, backdoor insertion, model extraction, spoofing, information extraction, sybil attack)
- Possibility to carry out *pentests*

Security: best practices

Phases	Vulnerabilities Causes
Analysis phase	No risk analysis/ No security policy
	Biased risk analysis
	Unanticipated risks
Design phase	Relying on non-secure abstractions
	Security/Convenience tradeoff
	No logging
	Design does not capture all risks
Implementation phase	Insufficiently defensive input checking
	Non-atomic check and use
	Access validation errors
	Incorrect crypto primitive implementation
	Insecure handling of exceptional conditions
	Bugs in security logic
Deployment phase	Reuse in more hostile environments
	Complex or unnecessary configuration
	Insecure defaults
Maintenance phase	Feature interaction
	Insecure fallback

Chen, Barbar, Security for Machine Learning-based Software Systems, DOI 10.1145/3638531

Legal aspects

Legal aspects

- If you reuse data / models / code: Make sure the license terms allow this and that your license(s) is (are) compatible
- Make sure you do not violate the DSGVO / GDPR / European Data Act / Copyright / European AI Act
- Once a model is made available, it is impossible to restrict its use!
- Examples:
  - Models can put out near-exact copies of images/text in training data, ie Dall-E/Stable Diffusion generating images with Shutterstock/Getty Images watermarks, or reproducing artist's work
  - Code-generating tools such as GitHub copilot allow recreation of code, that is already contained in other software, regardless of the license terms of that software

Ethical aspects

Ethical aspects

- Be aware of people's tendency of overreliance!
  - ELIZA effect: the tendency of users to project human traits onto interactive software
  - Trusting into predictions above one's own assessment
- Misuse of AI by bad actors
  - Face recognition used to detect Uyghur population (China), Clearview used to track people's movement and employment status by police officers (even though use was prohibited)
  - Facial analysis used to track people's attention on billboards, change advertisement based on their demographic
- Source of truth during training
  - Data that foundation models are trained on does also contain false statements, ie law professor incorrectly accused of sexual harassment
  - Data in foundation models contains toxicity of the internet; human labor is used to label / annotate toxic text and images ie subcontracting workers in Kenya (OpenAI)

Infamous AI mistakes

AI in general

- **Automatization at amazon:** Experimental hiring tool, developed by a team of five, used artificial intelligence to give job candidates scores ranging from one to five stars
- Tool was trained on all resumes of the last 10 years
- Tool preferably suggested male candidates, penalyzing resumes that contained the word “women’s” or graduates from all-female colleges
- ***Why?***

https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G/ (2018)

Automated resume selection at amazon

- The dataset consisted of the resumes of the last 10 years: Predominantly male applicants
- Women are underrepresented in tech:

  https://fingfx.thomsonreuters.com/gfx/rngs/AMAZON.COM-JOBS-AUTOMATION/010080Q91F6/index.html
- Thus, the model learnt it was more often correct if it suggested male candidates: Unbalanced representation of the dataset

AI in general

- U.S. healthcare system uses commercial algorithms to guide health decisions
- Algorithm (Optum's Impact Pro) to target patients for "high-risk care management" programs
- Identify patients who benefit the most: https://www.science.org/doi/10.1126/science.aax2342
- Model is trained on healthcare spendings to determine the healthcare need
- Algorithm was much more likely to recommend white patients for these programs than black patients, even though the black patients were evidently sicker
- ***Why?***

Bias in healthcare need estimation

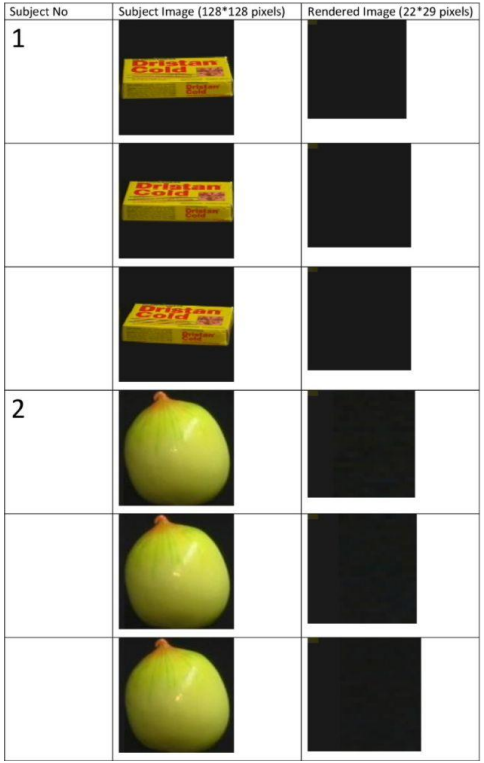
- Training based on healthcare spendings: But people of color are more likely to have lower incomes - making them less likely to access medical care even if they are insured
- Also, they may experience higher barriers to accessing health care (geography, transportation, work/childcare constraints), in addition to direct doctor-patient bias
- Data shows that race is correlated with substantial differences in health-care spendings: This results in a bias of the trained model

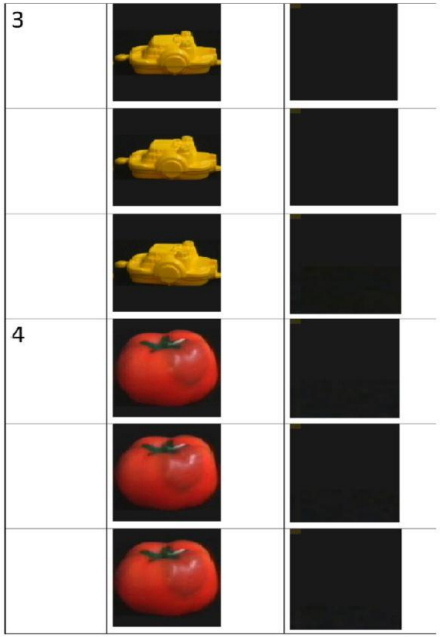
AI mistakes in research software

- Applying machine learning methods to COVID-19 radiological imaging for improving the accuracy of diagnosis
- Distinguish patients with COVID-19 from patients without COVID-19 but also bacterial pneumonia
- Tools were trained on public datasets with CT and CXR images
- Predictive tools failed practical tests: https://www.nature.com/articles/s42256-021-00307-0
- ***Why?***

Learning from the image background

- Image dataset was collected under controlled conditions: Does not represent the target distribution of interest
- CNN learnt to distinguish the image background rather than the image content
- Actually quite prevalent problem in computer vision





COIL-100 dataset https://www1.cs.columbia.edu/CAVE/publications/pdfs/Nene_TR96_2.pdf
https://doi.org/10.1016/j.visinf.2021.10.001

AI mistakes in research software

- Predict whether a country is likely to slide into civil war based on GDP, poverty rates, type of government structure, etc.
- Complex models using Random Forests and Adaboost outperform more standard statistical approaches like logistic regression by far
- Missing values in the dataset were constructed using imputation on the complete dataset
- Models proved to be over-optimistic and erroneous https://doi.org/10.1016/j.patter.2023.100804
- ***Why?***

Civil war predictions: Data leakage

- Data leakage: The data was imputed for missing values using the whole dataset
- Thus, the training dataset contained information about the test dataset
- This leads to an inflated estimate of the model performance

https://doi.org/10.1016/j.patter.2023.100804 supplemental material

Classification of failures/errors

Data leakage

Spurious relationship between independent variables and target variable

Artifact of collection, sampling, pre-processing

Leads to inflated estimates of model performance

Lack of clean separation training/test

- no test set
- pre-processing on training and test set (over/under sampling, imputation)
- feature selection on entire dataset
- duplicates in dataset

Kapoor, Narayanan, DOI: 10.1016/j.patter.2023.100804

Data leakage

Model uses features that are not legitimate

- for example, use of a certain drug when predicting illness (hypertensive drug, antibiotics)

Test set is not drawn from distribution of scientific interest

- temporal leakage (test set must not contain data from before the training set)
- non-independence between training and test samples (same people/units in both sets - use block cross-validation)
- sampling bias in test distribution (spatial bias, age, image settings)

Kapoor, Narayanan, DOI: 10.1016/j.patter.2023.100804

Resources

- Good practices in machine learning (Mathieu Bauchy) (https://www.youtube.com/watch?v=WScUQnU-ozQ&t=3213s
- Roadmaps https://roadmap.sh/roadmaps
- Kaggle https://www.kaggle.com/learn
- Hugging Face https://huggingface.co/learn
- REFORMS checklist https://reforms.cs.princeton.edu/
- MLCommons https://mlcommons.org/
- Scikit-learn resources https://scikit-learn.org/stable/common_pitfalls.html
- Testing of non-deterministic software https://bssw.io/blog_posts/testing-non-deterministic-research-software