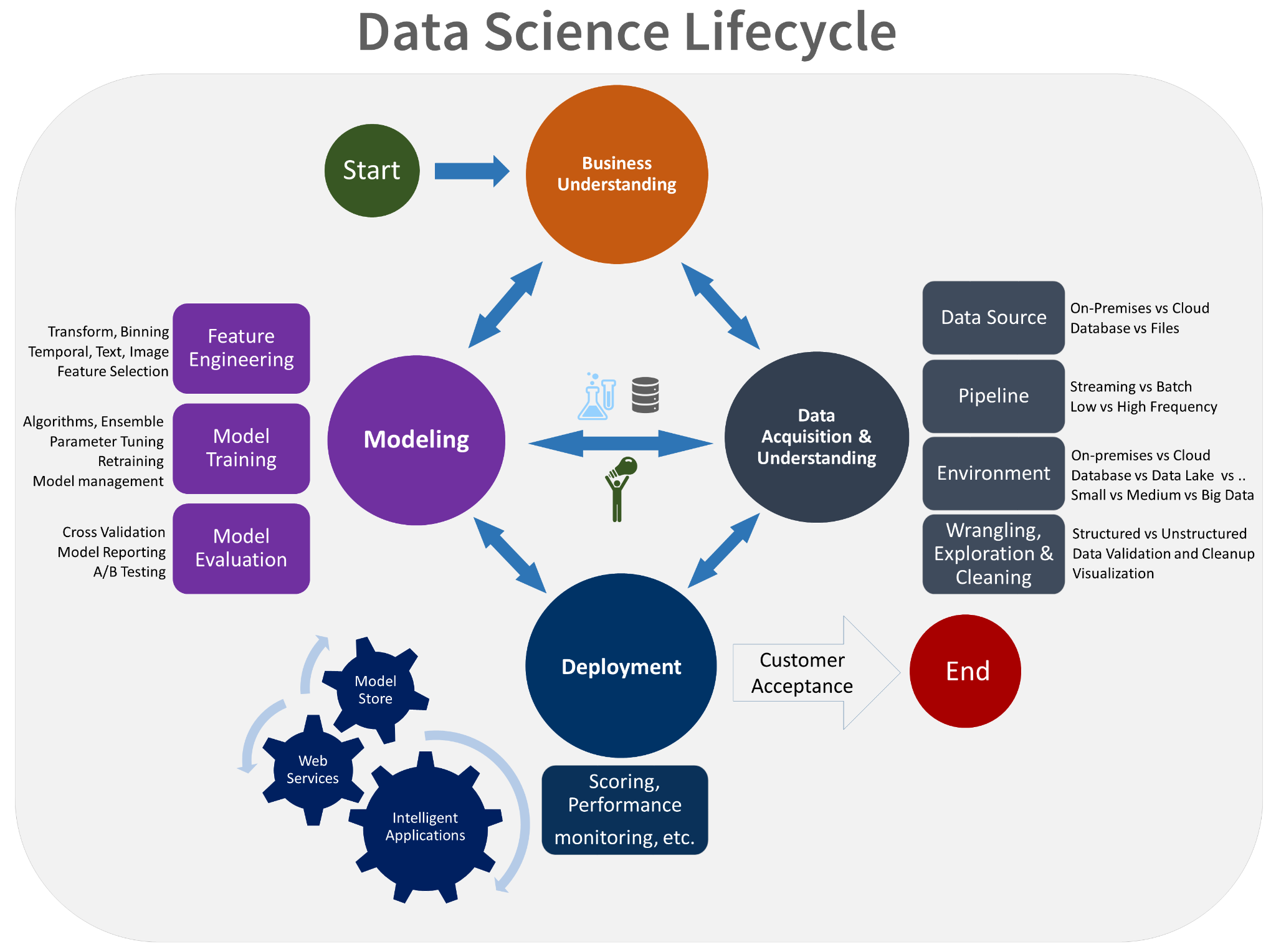
AI Standard Procedures

# 

# The Data Science Lifecycle (CRISP framework)

Every project should have their own lifecycle definition that fits better in its scope depending if concerning analytics or designing data products. However we can consider the following derivative model from the CRISP framework (Cross-Industry Standard Process for Data Mining) inspired in the Team Data Science Lifecycle from Microsoft:

1. Business understanding
2. Data Acquisition and Understanding (separate sometimes)
3. Modeling
4. Deployment (or Reporting if Analytics)
5. Customer Acceptance



## Business Understanding:

### Aims:

* Specify the key variables that are to serve as the model targets and whose related metrics are used to determine the success of the project.
* Identify the relevant data sources that the business has access to or needs to obtain.

### Main tasks:

* **Define objectives:** Work with your customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.
* **Identify data sources:** Find the relevant data that helps you answer the questions that define the objectives of the project.

**Define objectives:**

1. Identify the key business variables that the analysis needs to predict. We refer to these variables as the ***model targets***, and we use the metrics associated with them to determine the success of the project. Two examples of such targets are sales forecasts or the probability of an order being fraudulent.
2. Define the **project goals** by asking and refining "sharp" questions that are relevant, specific, and unambiguous. You typically use data science or machine learning to answer five types of questions:
   * How much or how many? (regression)
   * Which category? (classification)
   * Which group? (clustering)
   * Is this weird? (anomaly detection)
   * Which option should be taken? (recommendation)
3. Determine which of these questions you're asking and how answering it achieves your business goals.

4. Define the **success metrics**. For example, you might want to achieve a customer churn prediction. You need an accuracy rate of "x" percent by the end of this three-month project. With this data, you can offer customer promotions to reduce churn. The metrics must be **SMART**: Specific, Measurable, Achievable, Relevant and Time-bound.

**Identify data sources:** measures for the target and features of interest.

### **Deliverables:**

* **Charter document:** The charter document is a living document. You update the template throughout the project as you make new discoveries and as business requirements change. The key is to iterate upon this document, adding more detail, as you progress through the discovery process. Keep the customer and other stakeholders involved in making the changes and clearly communicate the reasons for the changes to them. You can use *Miro* or *Visio* here.
* **Data sources documentation:** This section specifies the original and destination locations for the raw data. In later stages, you fill in additional details like the scripts to move the data to your analytic environment (dataset builders).
* **Data dictionaries:** This document provides descriptions of the data that's provided by the client. These descriptions include information about the schema (the data types and information on the validation rules, if any) and the entity-relation diagrams, if available.

## **Data Acquisition and Understanding**

### **Aims:**

* Produce a clean, high-quality dataset whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so you are ready to model.
* Develop a solution architecture of the data pipeline that refreshes and scores the data regularly (dataset builder), ingesting the different data sources necessary to your analysis.

**Main tasks:**

* Ingest the data into the analytic environment.
* Explore the data to determine if the data quality is adequate to answer the question. EDA (Exploratory Data Analysis) report.
* Set up a data pipeline to score new or regularly refreshed data.

### 

* **Ingest the data:** Set up the process to move the data from the source locations to the target locations where you run analytics operations, like training and predictions.
* **Explore the data:** Before you train your models, you need to develop a sound understanding of the data. Real-world data sets are often noisy, have missing values or have a host of other discrepancies. You can use data summarization and visualization to audit the quality of your data and provide the information you need to process the data before it is ready for modeling. The next step is to better understand the patterns that are inherent in the data. This data analysis helps you choose and develop an appropriate predictive model for your target. Look for evidence for how well connected the data is to the target. Then determine whether there is sufficient data to move forward with the next modeling steps. Again, this process is often iterative. You might need to find new data sources with more accurate or more relevant data to augment the data set initially identified in the previous stage.

### **Set up a data pipeline:** set up a process to score new data or refresh the data regularly as part of an ongoing learning process. In this stage, you develop a solution architecture of the data pipeline. You develop the pipeline in parallel with the next stage of the data science project.

### **Deliverables:**

* **Data report/dashboard:** This report/dashboard includes data summaries, the relationships between each attribute and target, variable ranking, and more and results of the EDA.
* **Solution architecture:** The solution architecture can be a diagram or description of your data pipeline that you use to run scoring or predictions on new data after you have built a model. It also contains the pipeline to retrain your model based on new data.
* **Checkpoint decision:** Before you begin full-feature engineering and model building, you can reevaluate the project to determine whether the value expected is sufficient to continue pursuing it. You might, for example, be ready to proceed, need to collect more data, or abandon the project as the data does not exist to answer the question.

## Modeling:

### Aims:

* Determine an optimal set of features for the machine-learning tasks.
* Create an informative machine-learning model that predicts the target most accurately. Curate the model with statistical validation and a layer of interpretability.
* Create a machine-learning model that serves as a component of a data product suitable for production.

### Main tasks:

* **Feature Engineering:** inclusion, aggregation, and transformation of raw variables to create the features used in the analysis. An Intermediate point is to analyze how robust the features are, how they relate to each other and how the ML models are affected by them. This step requires a creative combination of domain expertise and the insights obtained from the data exploration step.
* **Model selection:** create a baseline ensemble of ML models that can serve as a first orientation on the performances you want to achieve. After selecting the baseline model that will serve as orientation and offer the best baseline performance in terms of the metric we chose to optimize, the next step is to create and optimize the ML model that will be offered for production.Create a layer of interpretability and statistical validation.
* **Determine if the ML model will be in production**.

### Deliverables:

* **The data model** (features and ML model), **a performance statistical report** (metrics on the test set bootstrapped, confidence intervals and layer of interpretability with LIME, SHAP or other ad-hoc tools).
* **A decision if the model goes into production.**

## Deployment:

### Aims:

* Deploy models with a **data pipeline** to a production or production-like environment for final user acceptance.

### Main tasks:

* **Model deployment:** the model and pipeline made operational to a production or production-like environment for application consumption.

**Model deployment:** predictions are usually made either in real time or on a batch basis. To deploy models, you expose them with an open API interface. The interface enables the model to be easily consumed from various applications, such as websites, dashboards, line-of-business applications or back-end applications.

### Deliverables:

* A **status dashboard** that displays the system health and key metrics.
* A **final modeling report** with deployment details.
* A **final solution architecture document**.

## Customer Acceptance:

### Aims: Finalize project deliverables: Confirm that the pipeline, the model, and their deployment in a production environment satisfy the customer's objectives.

### Main tasks:

* **System validation**: Confirm that the deployed model and pipeline meet the customer's needs.
* **Project hand-off**: Hand the project off to the entity that's going to run the system in production.

### Deliverables:

* **Exit report of the project for the customer**. This technical report contains all the details of the project that are useful for learning about how to operate the system.

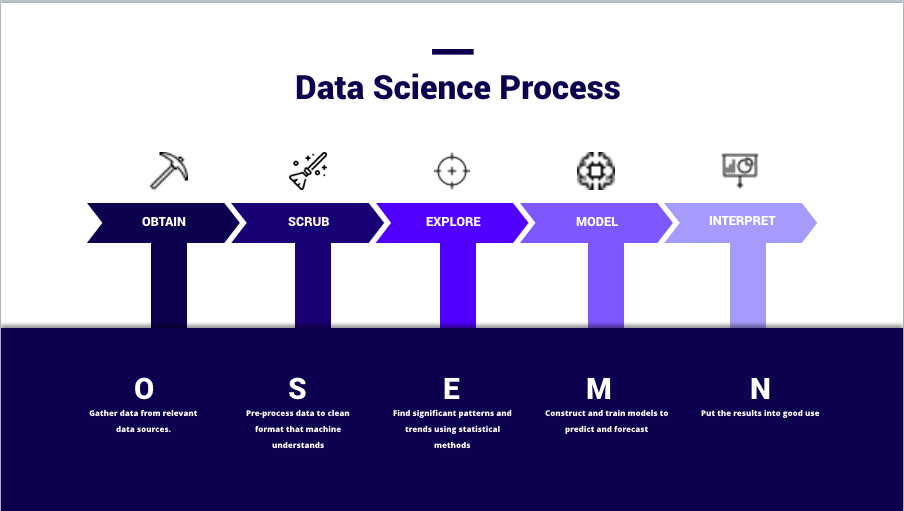
# 

# 

# 

# 

# The OSEMN framework and technologies that can be used step by step



**Technologies can that be used through the OSEMN framework:**

1. **O**btain: if involving connecting to databases one can use MySQL if the data is structured or MongoDB as an example of an unstructured database. Distributed storage can be achieved using tools such as Hadoop, Apache Spark to wrangle the data. Quite often one encounters data storage in a cloud environment so being effective with datalakes stored in services such as S3 from AWS or data warehouses such as Redshift is a must.
2. **S**crub: Alternatives to Pandas for cases of big data files: Polars, Vaex, PySpark…
3. **E**xplore: Dash, Streamlit and Plotly can be used as a more customized way to produce interactive dashboards to Tableau. Python traditional libraries for Data Analysis follow here. Pandas Profiling Report or Pycaret libraries offer a succinct way to describe datasets and potential models.
4. **M**odel: Traditional libraries in Python such as scikit-learn, tensorflow or pytorch are usual in this step.
5. **I**nterpret: the ML interpretability layer can be achieved with LIME, SHAP and other libraries for Interpretable AI. Models can be deployed in a cloud framework through Docker and services such as Sagemaker from AWS.

References:

* <https://medium.com/data-engineering-on-cloud/data-engineering-and-data-science-on-cloud-aws-f84e1e54346b>
* <https://towardsdatascience.com/8-alternatives-to-pandas-for-processing-large-datasets-928fc927b08c>
* <https://medium.datadriveninvestor.com/plotly-dash-everything-you-need-to-know-bc09a5e45395>
* <https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-deployment.html>
* <https://medium.com/swlh/machine-learning-model-deployment-in-docker-using-flask-d77f6cb551d6>
* <https://towardsdatascience.com/explainable-ai-xai-a-guide-to-7-packages-in-python-to-explain-your-models-932967f0634b>

# Software Engineering good practices

1. **Store data, artifacts, and code in different locations:** When working on a project, we’ll encounter three types of files: Data: All the raw data and any intermediate data results (e.g., CSV, txt, SQL dumps, etc.), Artifacts: Any other files generated during a pipeline execution (e.g., a serialized model, an HTML report with model metrics) and Source code. All the files to support the pipeline execution (e.g., .py, .R source code, Jupyter notebooks, tests). To keep things organized, separate each group into different folders.
2. **Declare paths and configuration parameters in a single file:** When working on a Machine learning project *flexibility* and *reusability* are very important to make your life easier while developing the solution. Create a yaml config file where you store the paths that you can use to call as global variables in the source code.
3. **Store credentials securely.** We can do this using keyring in a separate py file or just store them in a separate config file.
4. **Keep your repository clean by using a .gitignore file.** Missing this file can make your repository unnecessarily messy and might break the project for your colleagues (i.e. if you commit some user-dependent configuration file).
5. **Automate the workflow end-to-end.** You can use plumber here or when structuring the production code using a separate folder where we write in OOP format all the tasks we do through the data pipeline using Jupyter notebooks only as runners.
6. **Declare all software dependencies.** A dependency manager like Python Poetry helps you specify, install, and resolve external packages in your projects. This way, you can be sure that you always work with the right dependency version on every machine.
7. **Split production and development dependencies.** We might want to keep two dependency files (say requirements.prod.txt and requirements.dev.txt). Then, during development we can run: pip install -r requirements.dev.txt or pip install -r requirements.prod.txt.
8. **Have an informative and concise README:** What to include in the README file? Basic information about the project, at the very least, include a brief description of the project (what it does), list data sources (where does the data come from) and instructions for running the pipeline end-to-end.
9. **Document code:** as a general recommendation we should keep our code simple (good variable names, small functions), so others can understand it quickly and have some minimal documentation to support it.How to document code? Add a string right next to the function definition (aka a docstring) to provide a high-level description and convey as much information as possible using the code itself.
10. **Organize your repository in a hierarchical structure:** special attention when writing the code of the project. It can be thought of beforehand using the pythonic library Hydra. Check also an example of a DS project at the end of this doc.
11. **Package your code:** here it is extremely important to use a virtual environment in order to avoid dependency problems.
12. **Keep your Jupyter notebooks simple:** after using the modular paradigma the jupiter notebooks should be reserved to run experiments automatically. In case they train ML models they should create the model stored as a h5file or in json format as eg.
13. **Use the logging module (do not use print):** The problem with the example above is using the print statement. print sends a stream to standard output by default. So if you have a few print statements on every file, your terminal will print out dozens and dozens of messages every time you run your pipeline. The more print statements you have, the more complex it is to see through the noise, and they’ll end up being meaningless messages. Use the logging [module](https://docs.python.org/3/library/logging.html) instead. Among logging’s most relevant features are: filtering messages by severity, adding a timestamp to each record, including the filename and the line where the logging call originated, among others.
14. **Test your code:** is standard practice to use unit tests and integration tests.
15. **Take care of code quality:** the code is going to be read by humans.
16. **Delete dead and unreachable code.**

**References:**

* [**https://towardsdatascience.com/5-reasons-to-use-yaml-files-in-your-machine-learning-projects-d4c7b9650f27**](https://towardsdatascience.com/5-reasons-to-use-yaml-files-in-your-machine-learning-projects-d4c7b9650f27)
* [**https://odsc.medium.com/develop-and-deploy-a-machine-learning-pipeline-in-45-minutes-with-ploomber-f2caf9ba6e93**](https://odsc.medium.com/develop-and-deploy-a-machine-learning-pipeline-in-45-minutes-with-ploomber-f2caf9ba6e93)
* [**https://realpython.com/dependency-management-python-poetry/**](https://realpython.com/dependency-management-python-poetry/)
* [**https://towardsdatascience.com/how-to-structure-a-data-science-project-for-readability-and-transparency-360c6716800**](https://towardsdatascience.com/how-to-structure-a-data-science-project-for-readability-and-transparency-360c6716800)
* [**https://medium.com/pytorch/hydra-a-fresh-look-at-configuration-for-machine-learning-projects-50583186b710**](https://medium.com/pytorch/hydra-a-fresh-look-at-configuration-for-machine-learning-projects-50583186b710)
* [**https://python-course.eu/python-tutorial/modules-and-modular-programming.php**](https://python-course.eu/python-tutorial/modules-and-modular-programming.php)
* [**https://python-course.eu/python-tutorial/modules-and-modular-programming.php**](https://python-course.eu/python-tutorial/modules-and-modular-programming.php)
* [**https://www.guru99.com/unit-test-vs-integration-test.html**](https://www.guru99.com/unit-test-vs-integration-test.html)
* [**https://realpython.com/pytest-python-testing/**](https://realpython.com/pytest-python-testing/)
* [**https://towardsdatascience.com/virtual-environments-104c62d48c54#:~:text=A%20virtual%20environment%20is%20a,a%20system%2Dwide%20Python**](https://towardsdatascience.com/virtual-environments-104c62d48c54#:~:text=A%20virtual%20environment%20is%20a,a%20system%2Dwide%20Python)**).**
* [**https://git-scm.com/docs/gitglossary**](https://git-scm.com/docs/gitglossary)

Example of a Data Science Project structure

.

└── model-engineering-framework/

├── config/

│ ├── AWS.config

│ ├── credentials.cfg

│ └── paths.cfg

└── model-engineering/

├── datasets/

│ ├── dataset\_builder\_utils.py

│ └── dataset\_builder.py

├── experiments/

│ ├── exp1/

│ │ ├── exp1\_config.yaml

│ │ └── exp\_runner.yaml

│ ├── exp2/

│ │ ├── exp1\_config.yaml

│ │ └── exp\_runner.yaml

│ └── experiment\_backgrounds/

│ ├── exp1\_trainer.py

│ └── exp2\_trainer.py

├── learners/

│ ├── trainer\_mtl.py

│ ├── trainer\_skeleton.py

│ ├── trainer.py

│ └── utils.py

├── mlops/

│ └── clearml\_wrapper.py

├── models/

│ ├── regression\_tasks/

│ │ └── trust\_model\_multipercepron.py

│ └── classification/

│ └── exp\_layers.py

├── notebooks/

│ ├── example\_dataset\_building.ipynb

│ ├── example\_multiperceptron\_runner.ipynb

│ └── example\_experimental\_layers\_runner.ipynb

├── poc/

│ └── nn\_model\_experiment/

│ ├── pytorch\_model.py

│ ├── tensorflow\_model.py

│ ├── model\_runner.ipynb

│ └── output\_models/

│ ├── nn\_model\_pytorch\_timestamp.json

│ └── nn\_model\_tensorflow\_timestamp.json

├── project\_trust/

│ ├── create\_diagrams.ipynb

│ └── evaluation\_report/

│ ├── config\_report.yaml

│ └── report\_generator.ipynb

├── poetry.lock

├── pyproject.toml

├── README.md

└── tests/

└── data/

├── test\_data\_utily.py

└── test\_training\_pipeline.py

# References:

* <https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining>
* <https://towardsdatascience.com/crisp-dm-methodology-leader-in-data-mining-and-big-data-467efd3d3781>
* <https://learn.microsoft.com/de-de/azure/architecture/data-science-process/overview>
* <https://datarundown.com/data-science-life-cycle/>