MLOPS

Based on https://ml-ops.org/

MLOps

MLOps establishes effective practices and processes around the

design, development, test, deploy and maintain

ML models into production

MLOps we strive to avoid *"**technical debt**"* in machine learning applications

Agenda

1. CRISP-ML(Q)
2. Three Levels of ML Software
3. MLOps Principles
4. MLOps Stack Canvas

CRISP-ML(Q)

Cross-Industry Standard Process for the development of Machine Learning applications with Quality assurance methodology

CRISP-ML(Q)

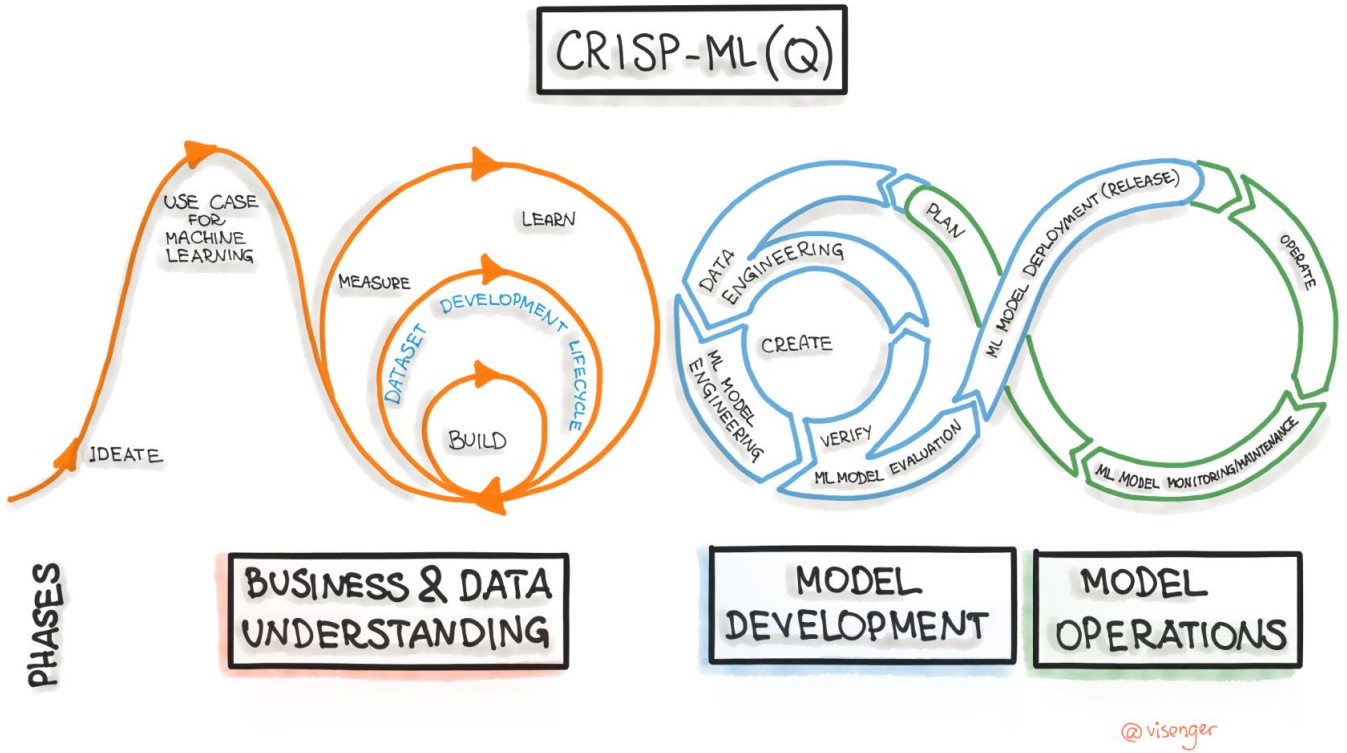
CRISP-ML(Q) is a

process model for machine learning software development

that creates an awareness of possible risks and

emphasizes quality assurance to diminish these risks

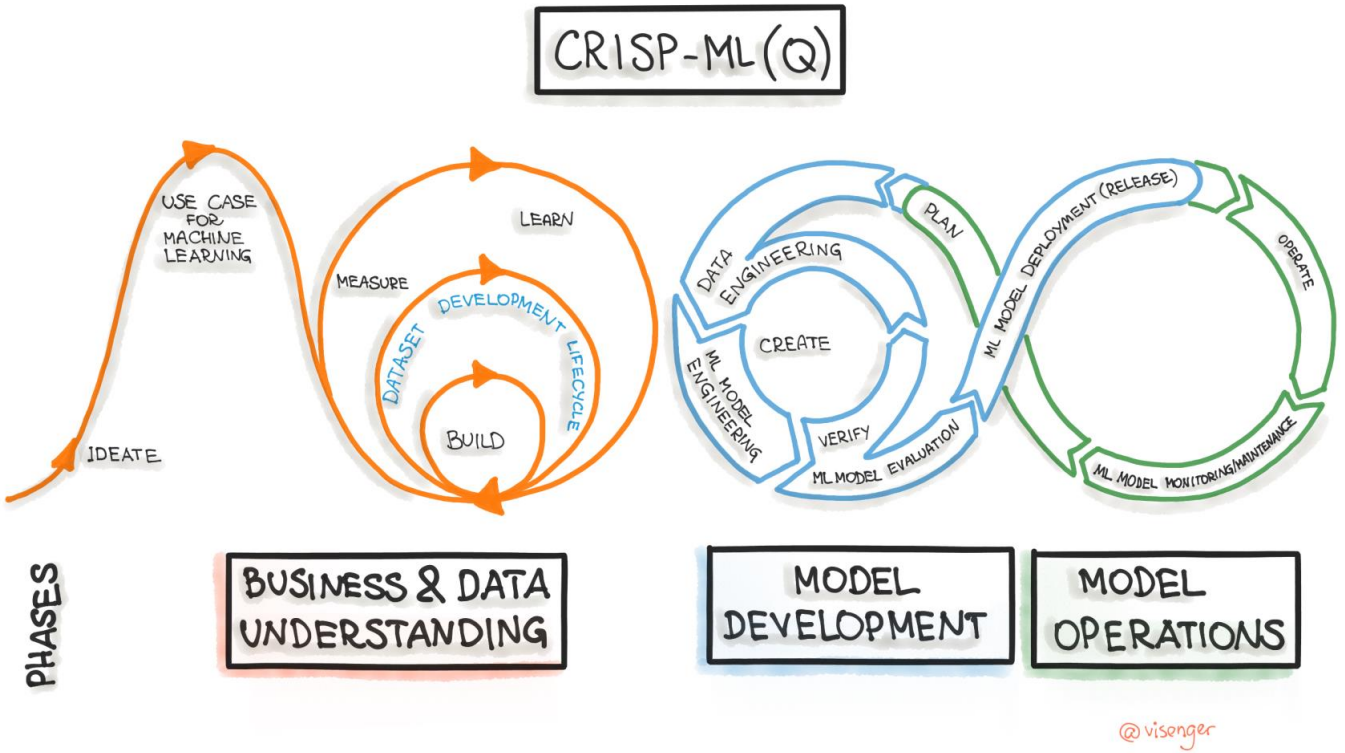
to ensure the ML project's success.



CRISP-ML(Q)

CRISP-ML(Q) process model describes six phases:

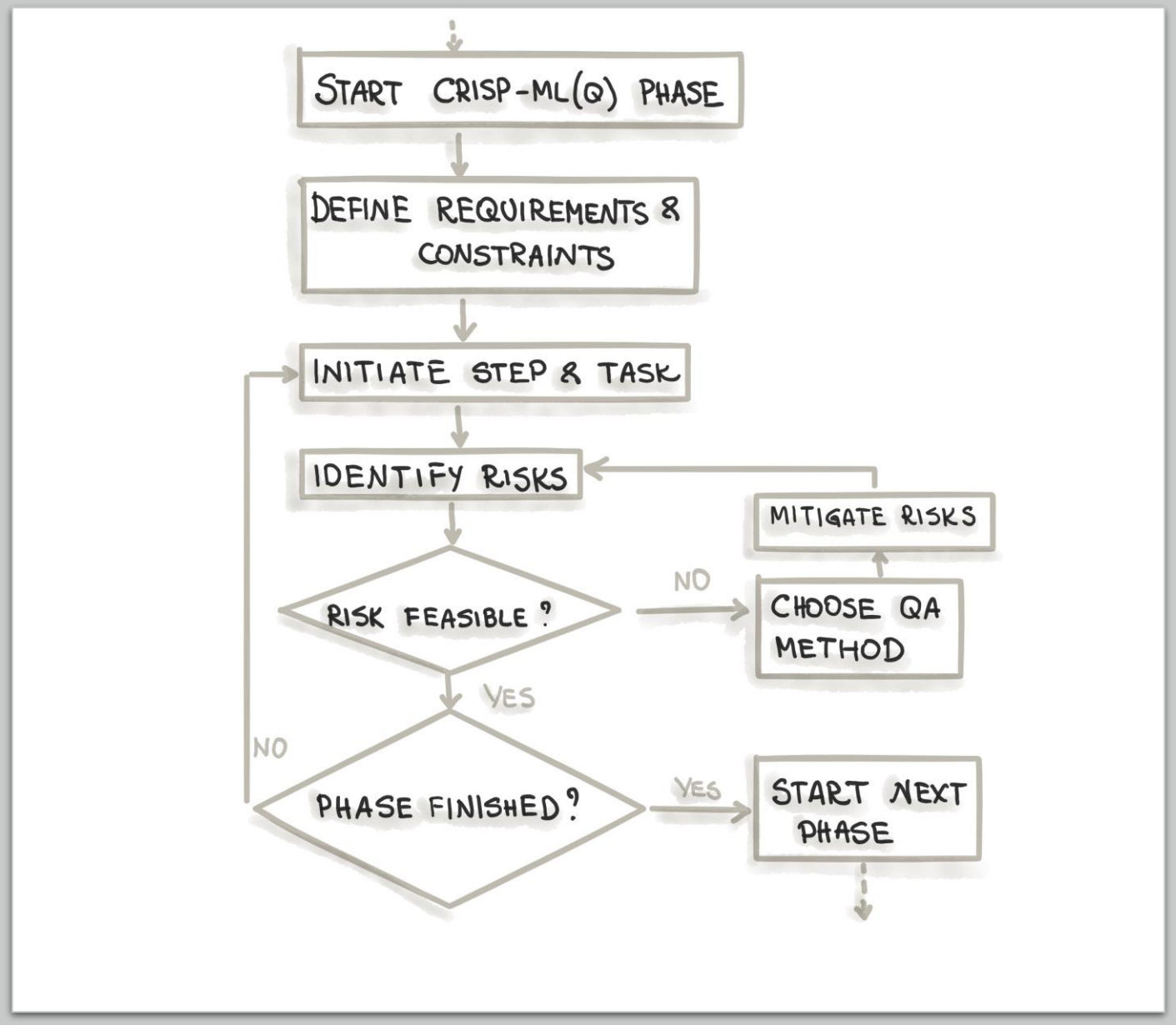
1. Business and Data Understanding
2. Data Engineering (Data Preparation)
3. Machine Learning Model Engineering
4. Model Evaluation
5. Deployment
6. Monitoring and Maintenance.

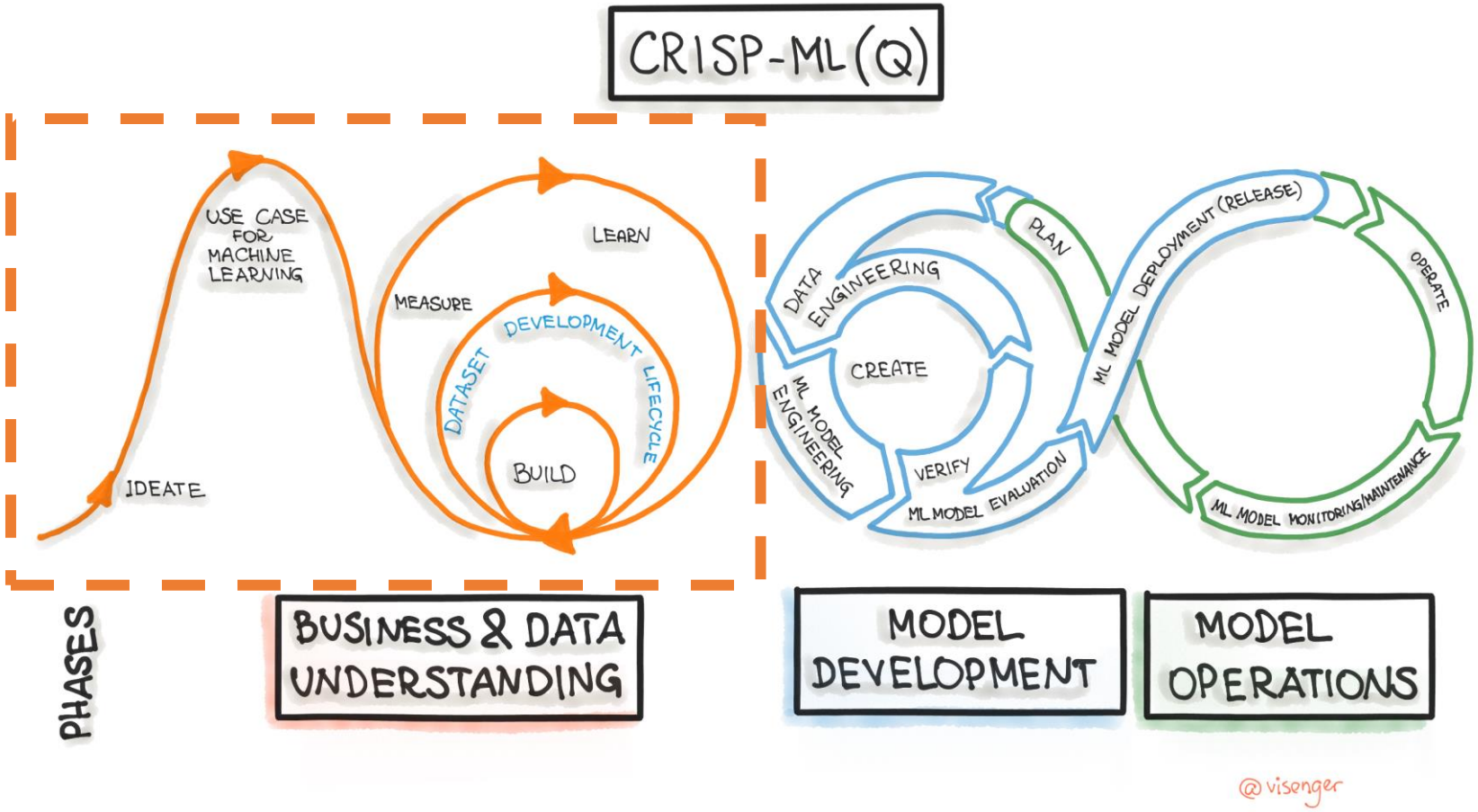


Quality Assurance Approach

For each phase of the process model the quality assurance requires:

Definition of requirements and constraints	Performance, data quality requirements, model robustness, etc.
Instantiation of the specific tasks	ML algorithm selection, model training, etc.
Specification of risks that might negatively impact the efficiency and success of the ML application	Bias, overfitting, lack of reproducibility, etc.),
Quality assurance methods to mitigate risks.	Cross-validation, documenting process and results

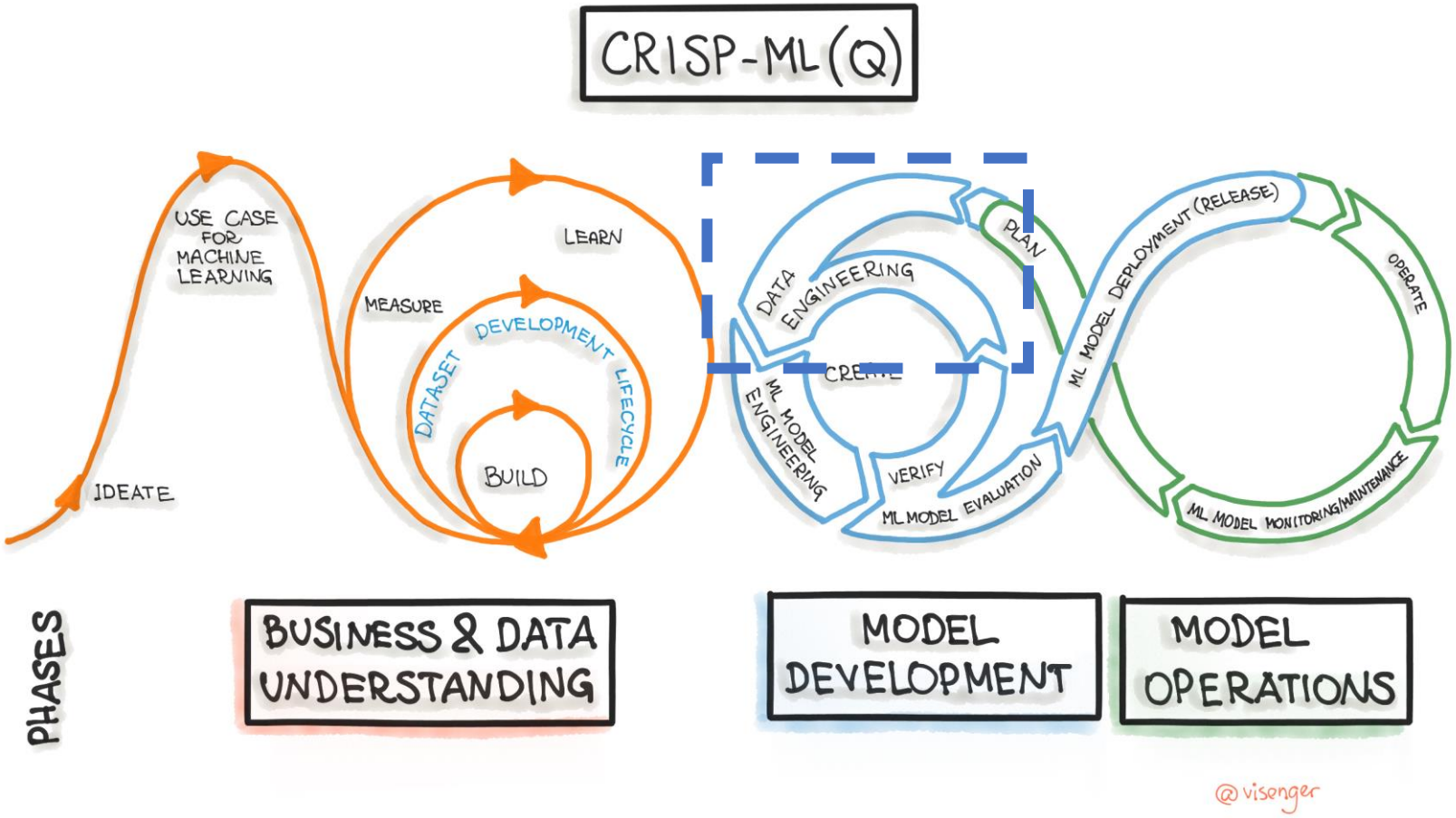




1.- Business and Data Understanding

Identify the scope of the ML application, the success criteria, and a data quality verification.

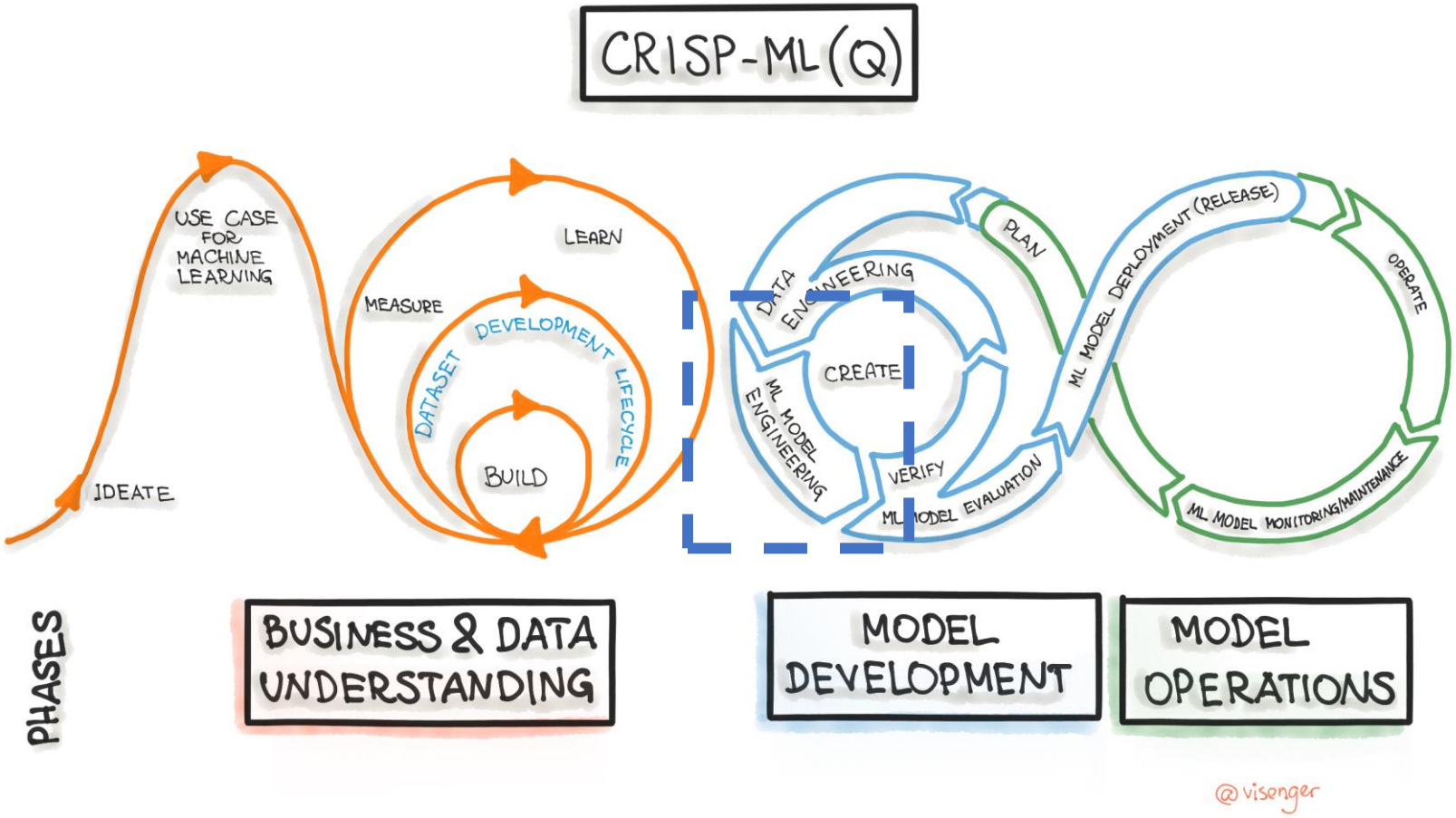
- **Define business objectives** (Feasibility? Scope?, Success criteria? KPI?, who is the final user?, increases user or app productivity?, use-cases?)
- **ML Canvas:** Translate business objectives into ML objectives. Data, learning, robustness, scalability, explainability, resource demand, etc
- **Collect and verify data** (EDA, quality, etc…)



2.- Data Engineering

Aims to prepare data for the following modeling phase.

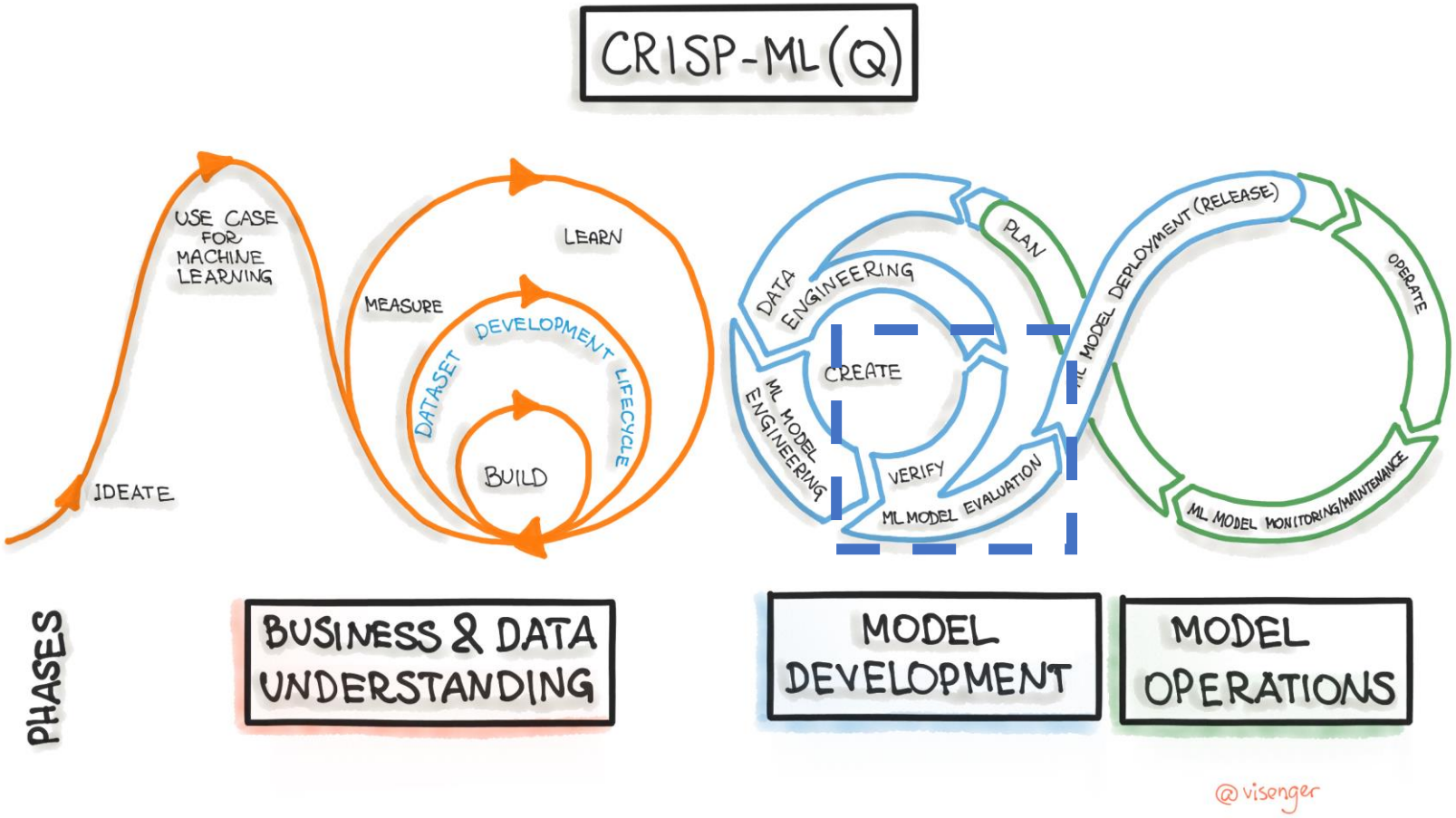
- **Data & Features selection**
- **Cleaning data**: noise reduction, data imputation, drop bad quality data.
- **Over/Under Sampling**, **Data augmentation** & **Preprocessing** (OHE, scaling, clustering, discretization, etc.)
- **Feature engineering** & Pipelines



3.- ML Model Engineering

Aims to specify one or several machine learning models to be deployed in the production.

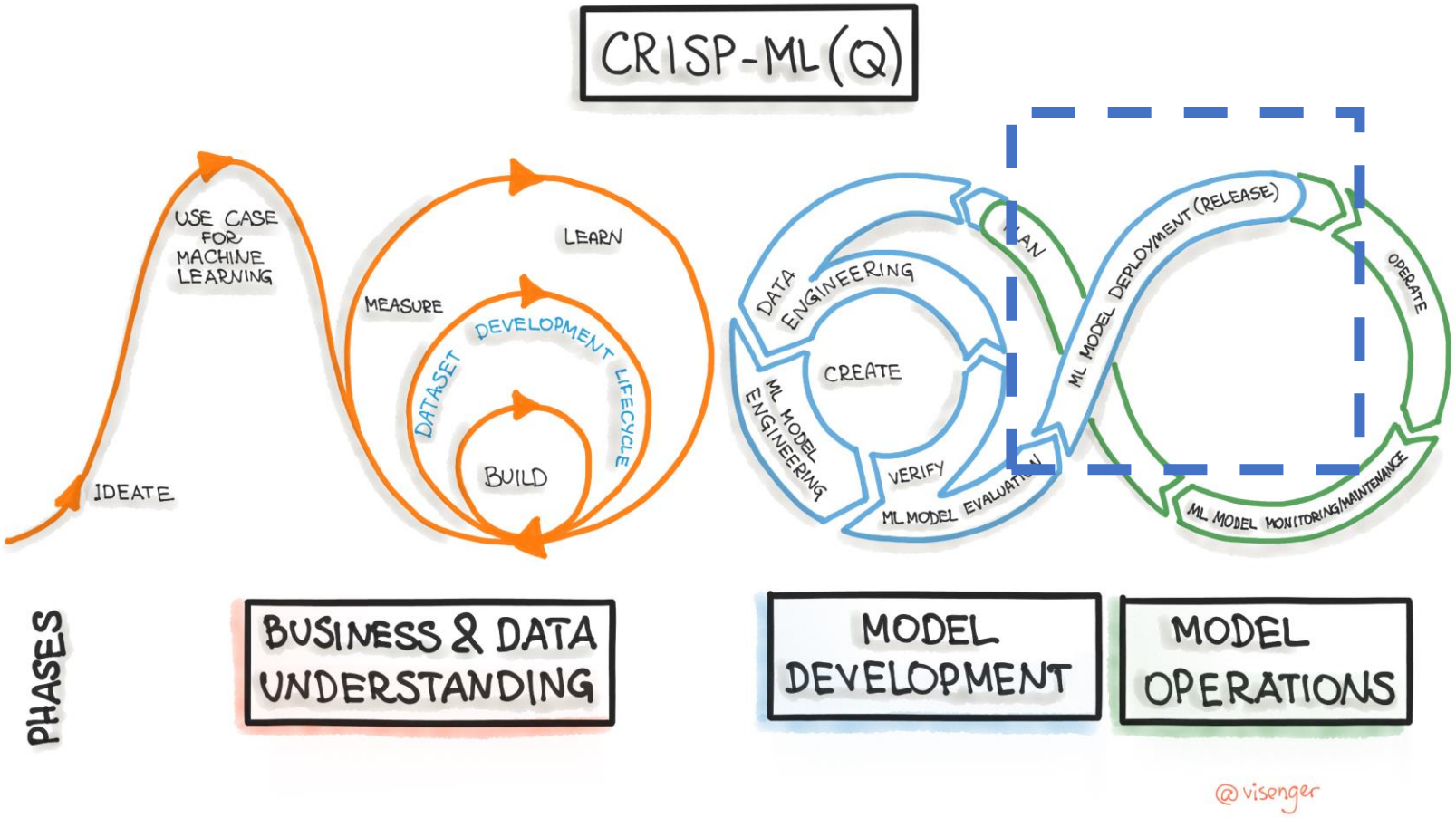
- **ML Model Engineering & Training**
- **ML model selection**
- **Ensure replicability collecting model metadata** : algorithm, training, validation and testing data set, hyper-parameters, and runtime environment description



4.- ML Model Evaluation

Validate the trained on a test set.

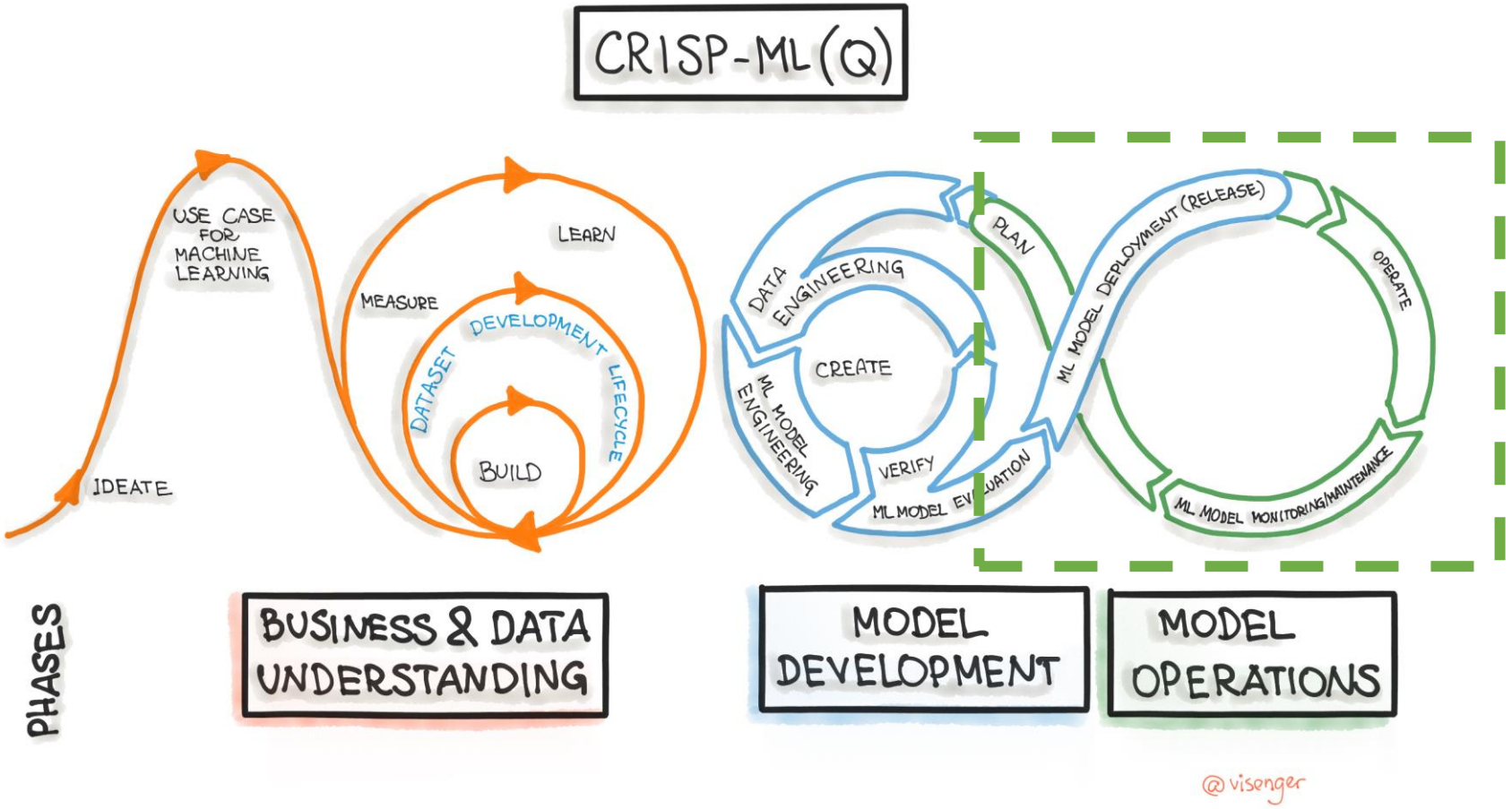
- **Validate model's performance** using validation set
- **Determine robustess** using noisy or wrong input data
- Increase model's **explainability**
- Make a decision whether to deploy the model



5.- Model Deployment

Model integration into the existing software system.

- Define the hardware needed
- Assure user acceptance and usability testing
- Provide a fall-back plan for model outages
- Deploy according to the selected deployment strategy (A/B testing, multi-armed bandits)



6.- Model Monitoring and Maintenance

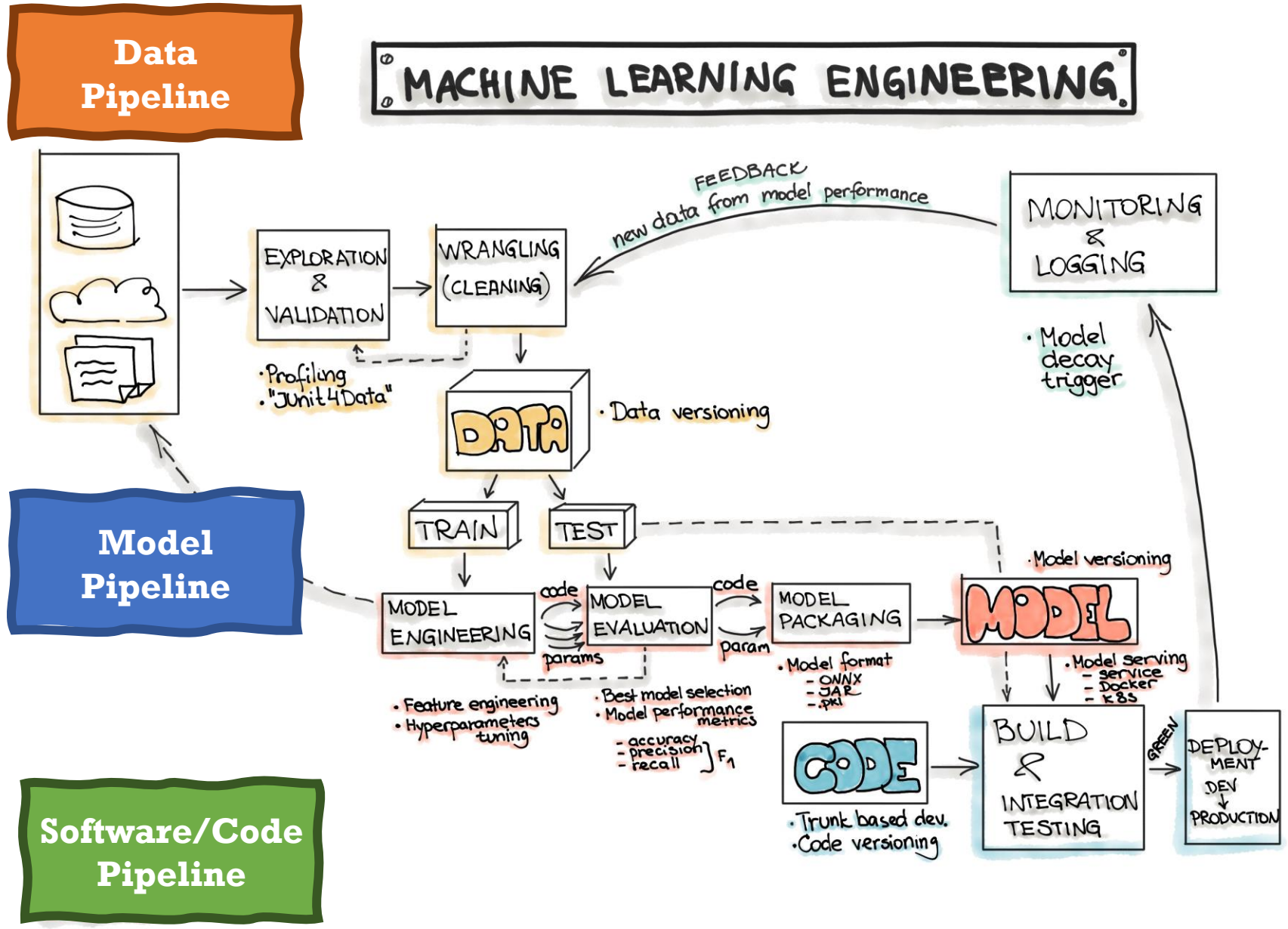
Monitor model's performance and maintain it

- Monitor the **efficiency and efficacy** of the model prediction
- **Retrain** model if required & **Collect new data**
- Continuous, **integration**, **training**, and **deployment** of the model

Three Levels of ML Software

Three Levels of ML Software

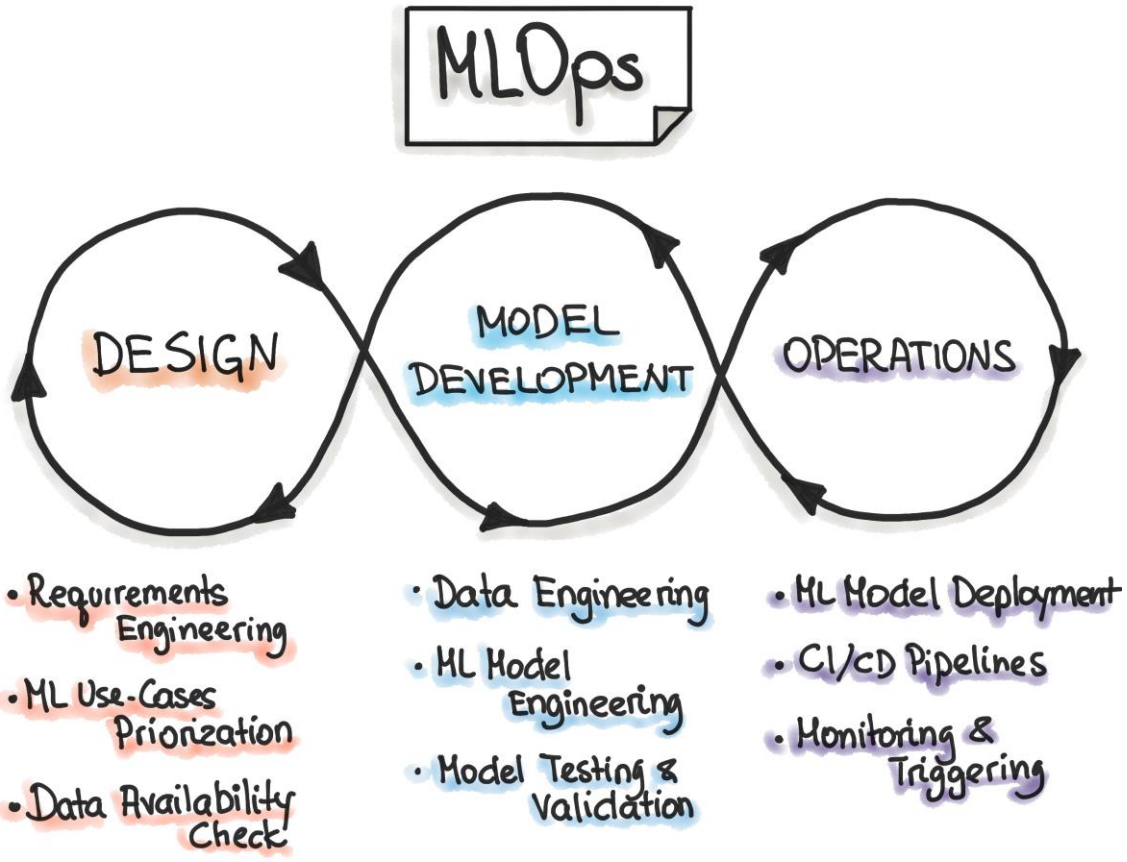
- Every ML-based software needs to manage three main assets: **Data**, **Model**, and **Code**.
- Each of these assets is managed in a **Pipeline**.



MLOps

Machine Learning Operations

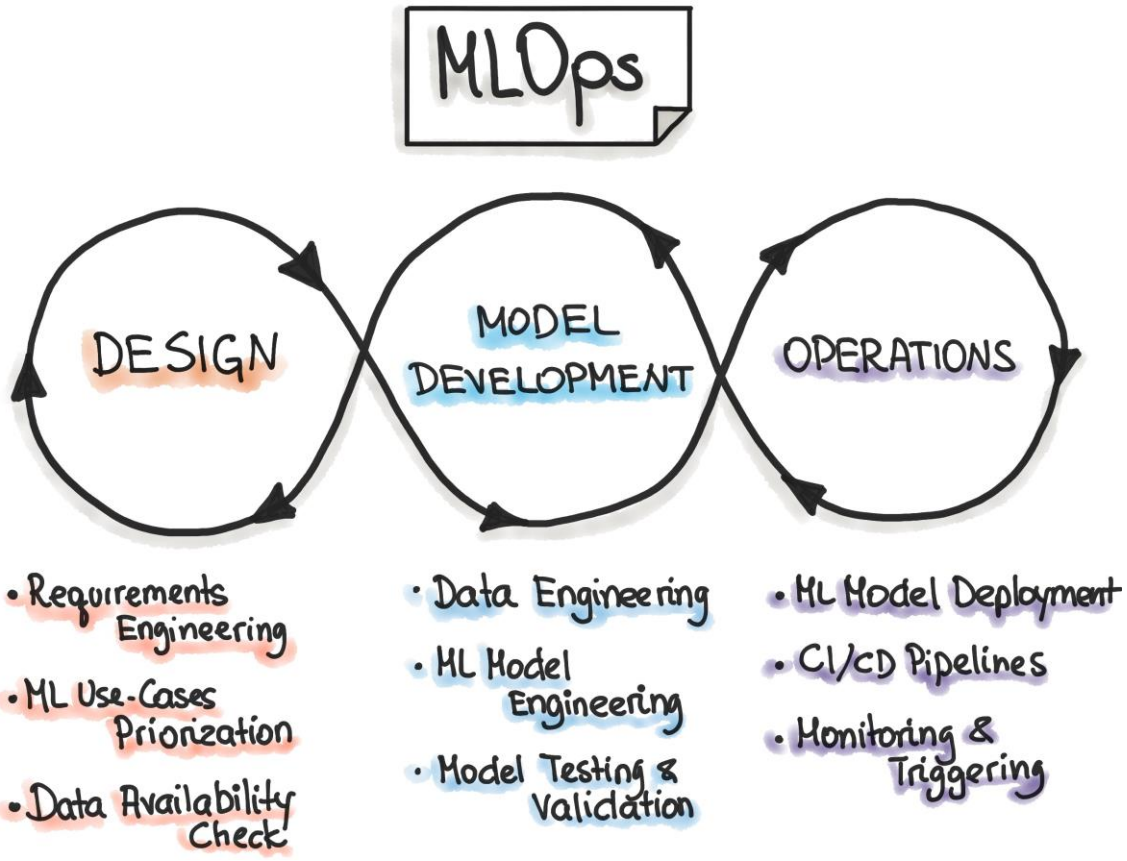
MLOps Principles



MLOps Principles

All three phases are interconnected and influence each other.

For example, the design decision during the design stage will propagate into the experimentation phase and finally influence the deployment options during the final operations phase.



MLOps Principles

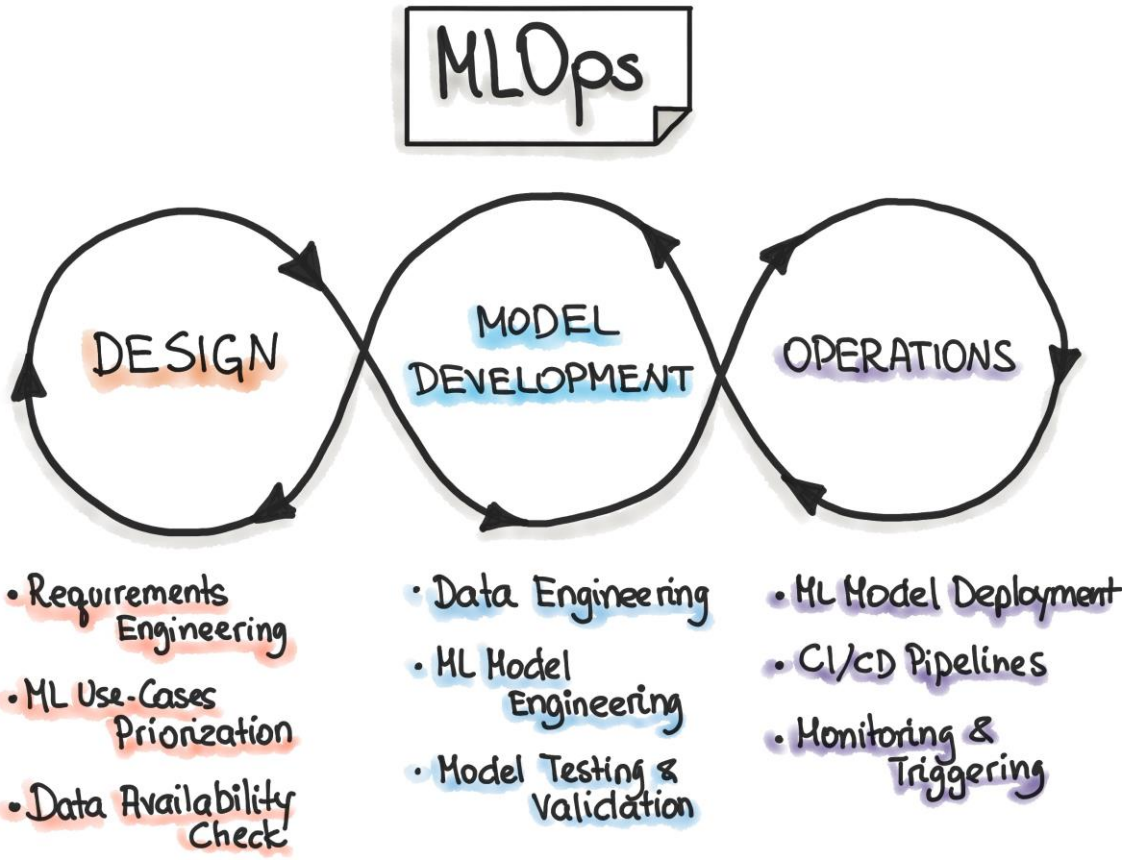
Continuous X and Automation

Versioning

Testing

Monitoring

Reproducibility



Continuous X

Continuous Integration (CI)	Continuous Delivery (CD)	Continuous Training (CT)	Continuous Monitoring (CM)
• Test and validate code, components data and models.	• Automatically deploy a model in the prediction service.	• Automatically retrains ML models for re-deployment.	• Monitor production data and model performance metrics

Automation Steps

1. Manual process

- Experimentation level.
- Every step in each pipeline, such as data preparation and validation, model training and testing, are executed manually (Jupyter Notebooks).

2. ML pipeline automation

- Data transformation and Feature creation and manipulation.
- Automatic model retraining triggered.
- Hyperparameter/Parameter selection

3. CI/CD pipeline automation

- Automatically build, test, and deploy the Model using CI/CD.

Versioning

ML models and data commonly changes:

- ML models can be retrained based upon new training data.
- Models may be retrained based upon new training approaches.
- Models may be self-learning.
- Models may degrade over time.
- Models may be deployed in new applications.
- Models may be subject to attack and require revision.
- Models can be quickly rolled back to a previous serving version.
- …

Versioning

The following processes should be tracked:

Data	ML Model	Code
1) Data preparation pipelines 2) Features store 3) Datasets 4) Metadata	1) ML model training pipeline 2) ML model (object) 3) Hyperparameters 4) Experiment tracking	1) Application code 2) Configurations

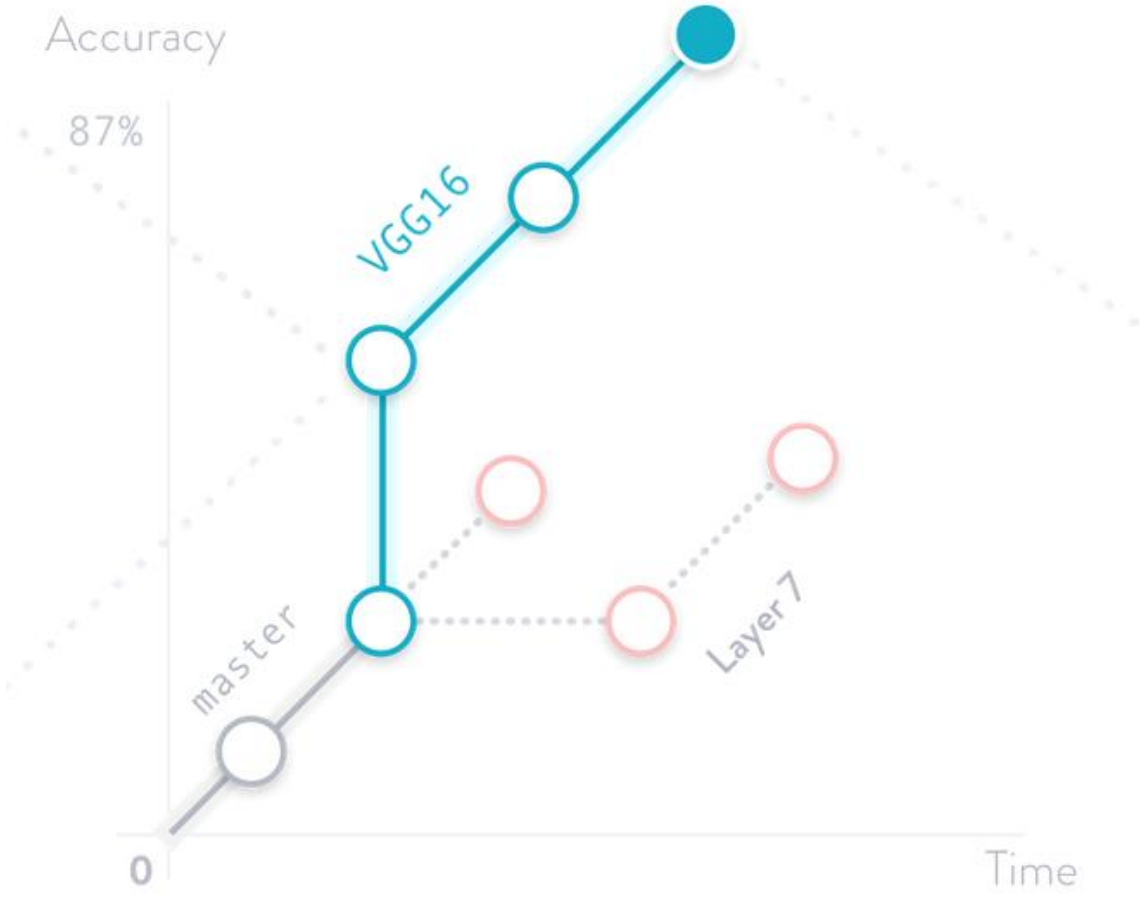
Also, every model and training code that creates an ML model **should go through a code review phase.**

Experiments Tracking

In ML, multiple experiments can be run in parallel before choosing the one to be promoted to production.

Each experiment should reside in a different branch and the result of this should be a new model that is integrated into the project.

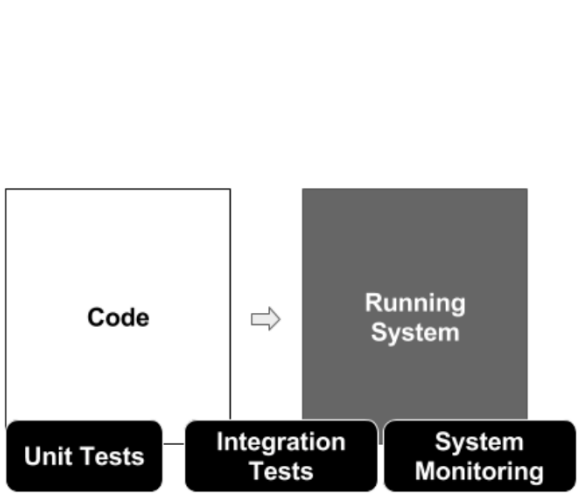
DVC is designed for this:
https://dvc.org/



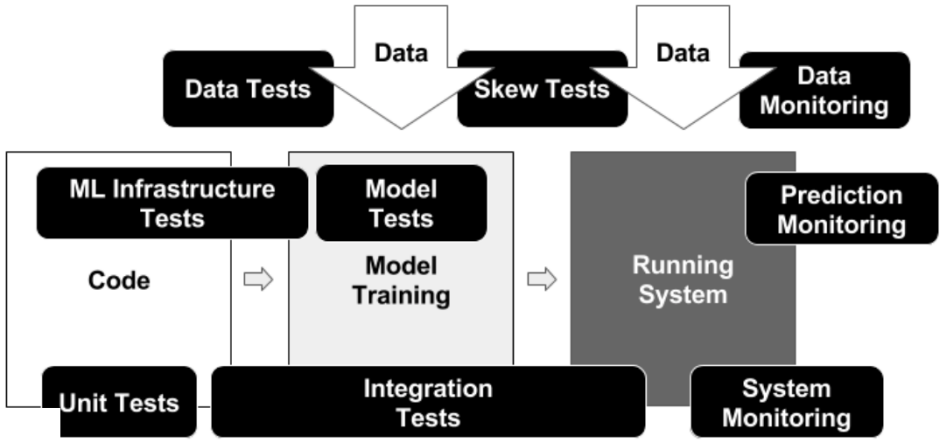
Testing

The complete development pipeline includes three essential components:

- **Data pipeline → Tests for features and data**
- **ML model pipeline → Tests for model development**
- **Application pipeline → Tests for ML infrastructure**



Traditional System Testing and Monitoring



ML-Based System Testing and Monitoring

Data Pipeline

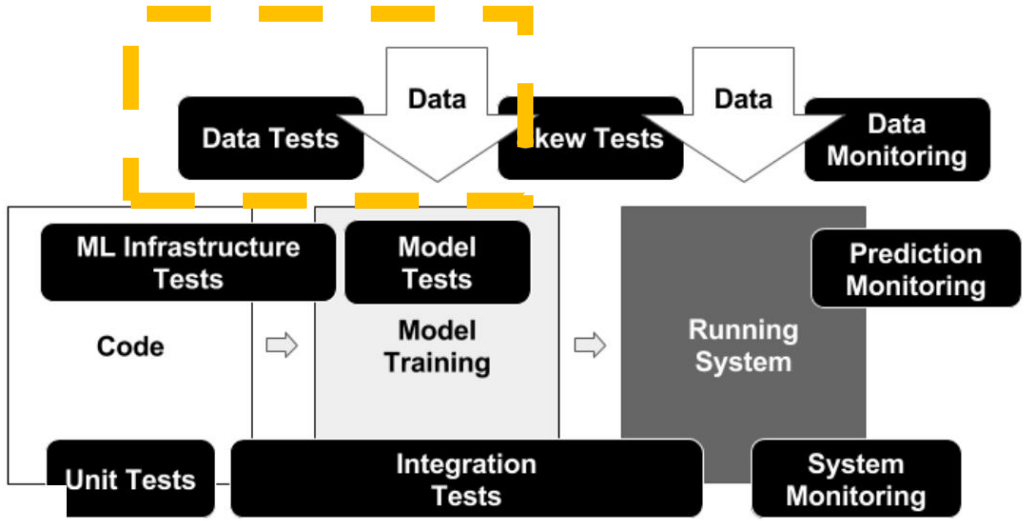
Features and Data Tests

Data validation

- Automatic check for data and features schema/domain.
  - Calculate statistics from the training data and build a schema.

Feature creation code

- Unit tests to capture bugs in features.



ML-Based System Testing and Monitoring

Model Pipeline

Tests for Reliable Model Development

Metric Selection

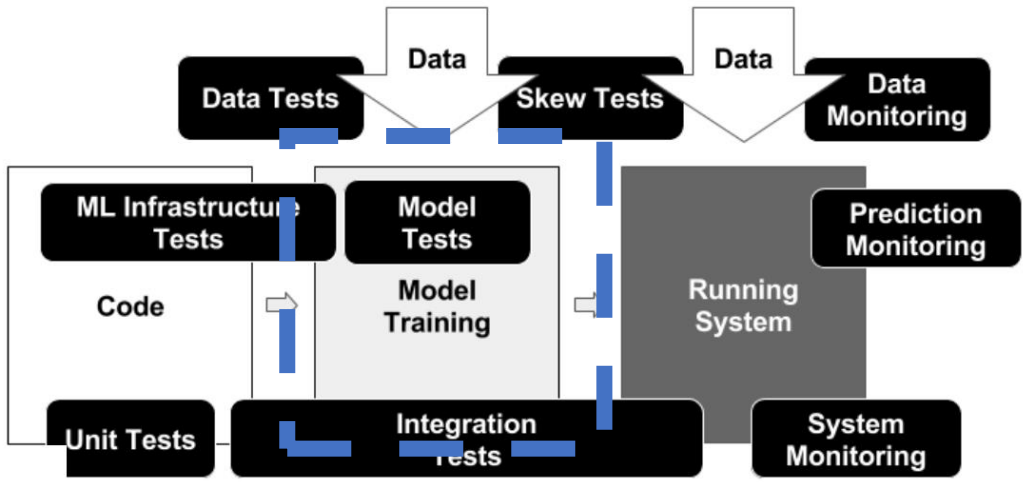
- **Choose a loss metric** (MSE, log-loss, etc.) **that correlate with business impact metrics** (revenue, user engagement, etc.)

Assessing the cost of more Sophisticated ML Models.

- Compare a simple baseline ML model with the sophisticated model.
- Example: linear model vs a deep neural network.

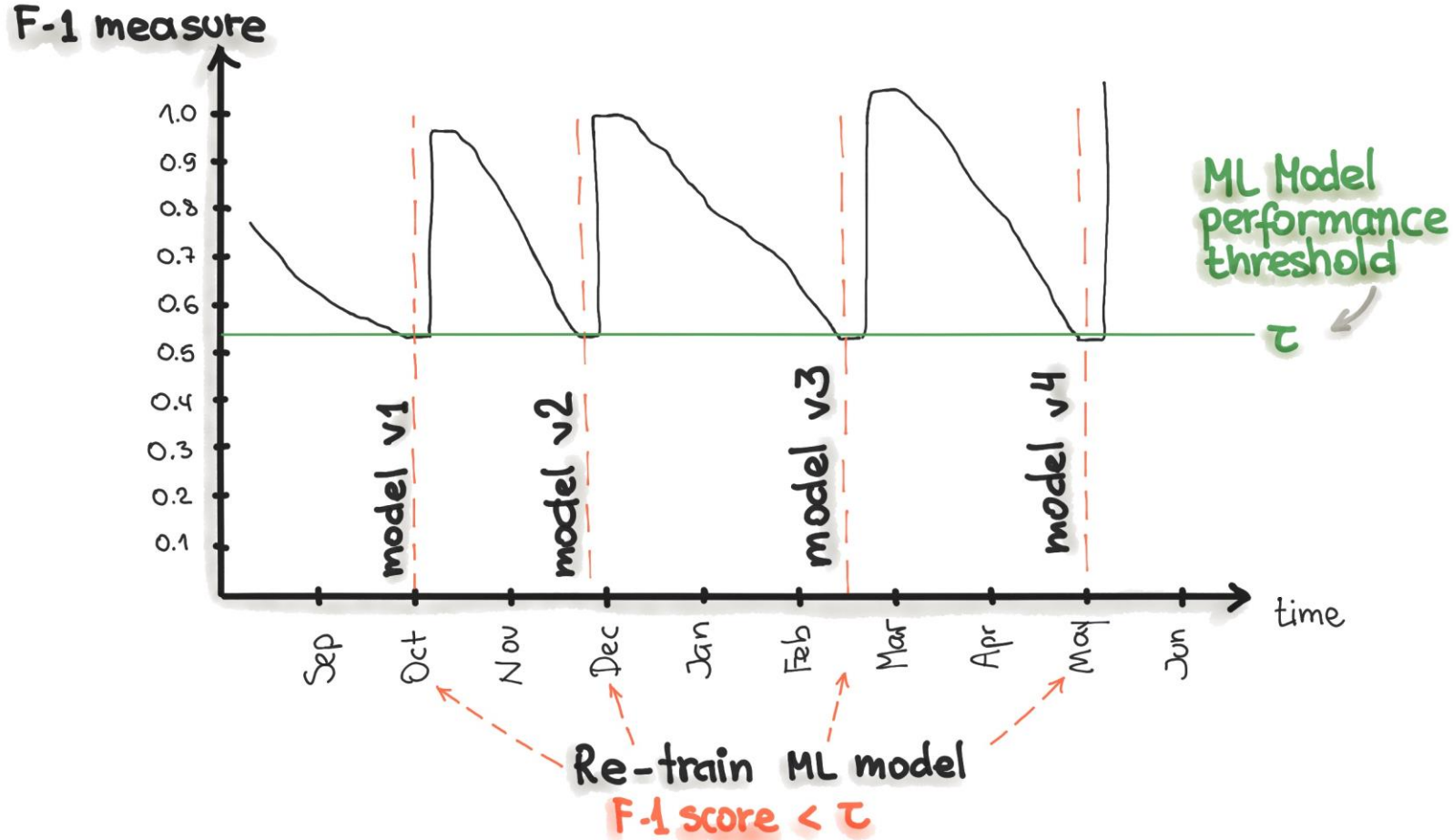
Model Staleness Test

- **Stale Model**: A model does not include up-to-date data and/or does not satisfy the business impact requirements.
- **A/B experiment with older models**. Produce an *Age vs. Prediction Quality* curve to understand of how often the ML model should be trained.



ML-Based System Testing and Monitoring

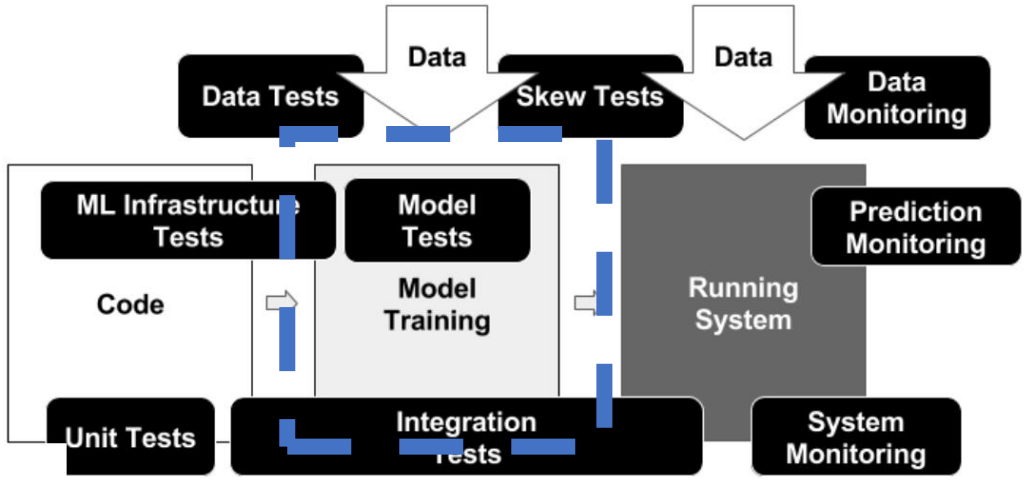
ML MODEL DECAY MONITORING



Model Pipeline

Tests for Reliable Model Development

Validating performance of a model	• Use an additional **test set** (disjoint from the training and validation sets) for a **final evaluation**.
Fairness/Bias/Inclusion Testing	• Action: Collect more data that includes **potentially under-represented categories**. • Action: Examine **input features** if they **correlate** with **protected user categories**.
Conventional Unit Testing	• **Feature creation,** • **ML model training and testing.**



ML-Based System Testing and Monitoring

Code Pipeline

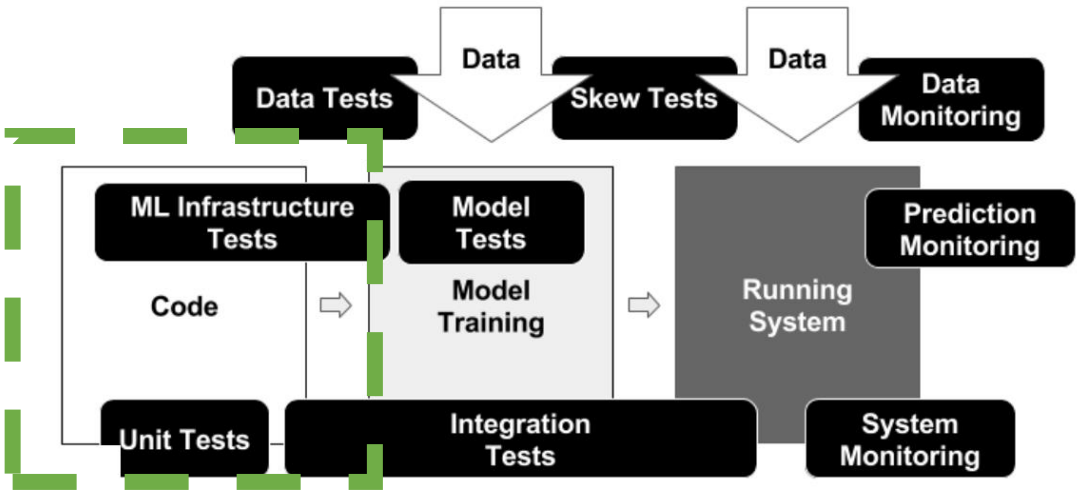
ML infrastructure Test

Reproducible Model Trainings

- Training the ML model on the same data should produce identical ML models.
- Action: determine the non-deterministic parts in the model training code base and try to minimize non-determinism.

Integration testing

- : The full ML pipeline should be integration tested: The test should validate that the data and code successfully finish each stage of training and the resulting ML model performs as expected.
- All integration tests should be run before the ML model reaches the production environment.



ML-Based System Testing and Monitoring

Code Pipeline

ML infrastructure Test

Test Model Degradation

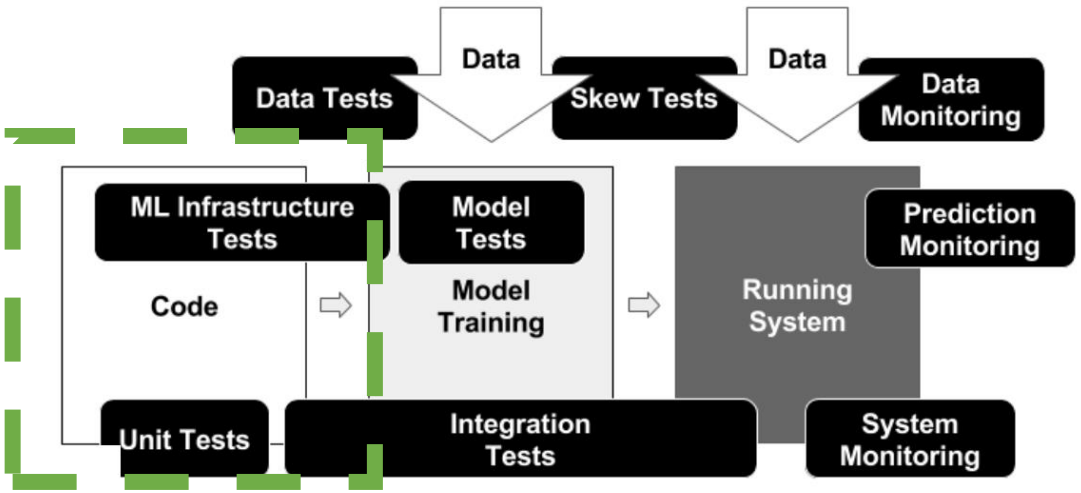
- Setting a threshold and testing for slow degradation in model quality over many versions on a validation set.
- Setting a threshold and testing for sudden performance drops in a new version of the ML model.

Canary Testing & Deployment

- Testing that an ML model successfully loads into production serving and the prediction on real-life data is generated as expected in a small group of users.

Model performance equal in dev and prod

- Prediction of a training example in dev and prod should result in the same prediction. A difference here indicates an engineering error.
- Check the difference between the performance on the holdout data and the "nextday" data. Some difference will always exist.
- Large differences may indicate that some time-sensitive features are causing troubles.



ML-Based System Testing and Monitoring

Monitoring

The deployed model needs to be monitored to ensure that it performs as expected.

In the Data Pipeline:

Monitor dependency changes

- Data version change.
- Changes in source system.
- Dependencies upgrade.

Monitor invariants in training and serving

- Alert if data does not match the schema
- Monitor whether training and serving features compute the same value.

Monitoring

The deployed model needs to be monitored to ensure that it performs as expected.

In the Model Pipeline:

Monitor the numerical stability of the ML model.	• Trigger alerts for the occurrence of any NaNs or infinities.
Monitor computational performance of an ML system.	• Measure the performance of versions and components of code, data, and model by pre-setting the alerting threshold. • Collect system usage metrics like GPU memory allocation, network traffic, and disk usage. These metrics are useful for cloud costs estimations.

Monitoring

The deployed model needs to be monitored to ensure that it performs as expected.

In the Code Pipeline:

Monitor how stale the system in production is.	• Measure the age of the model. Older ML models tend to decay in performance.
Monitor degradation of the predictive quality of the ML model on served data.	• If a label is available immediately after the prediction is made, we can measure the quality of prediction in real-time and identify problems.

Reproducibility

Every phase produce identical results given the same input.

Data	ML Model	Code
1) Backup data 2) Data versioning 3) Extract metadata 4) Versioning of feature engineering	1) Hyperparameter tuning is identical between dev and prod 2) The order of features is the same 3) Ensemble learning: the combination of ML models is same	1) Versions of all dependencies in dev and prod are identical 2) Same technical stack for dev and production environments 3) Reproducing results by providing container images or virtual machines

Project Structure

- Loosely Coupled Architecture (Modularity) principle

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- Make this project pip installable with `pip install -e`
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.readthedocs.io
```

Project Structure - Opinions

Data is immutable

- Don't ever edit your raw data, especially not manually, and especially not in Excel. Don't overwrite your raw data. Don't save multiple versions of the raw data. Treat the data (and its format) as immutable.
- Use Pipelines to move the data.

Project Structure - Opinions

Notebooks are for exploration and communication

- Notebook are very effective for EDA, however, these tools can be less effective for reproducing an analysis.
- Notebooks are **challenging objects for source control**. Recommendation: Not collaborating directly with others on Jupyter notebooks.
- **Naming convention:**

```
<step>-<ghuser>-<description>.ipynb
```

Project Structure - Opinions

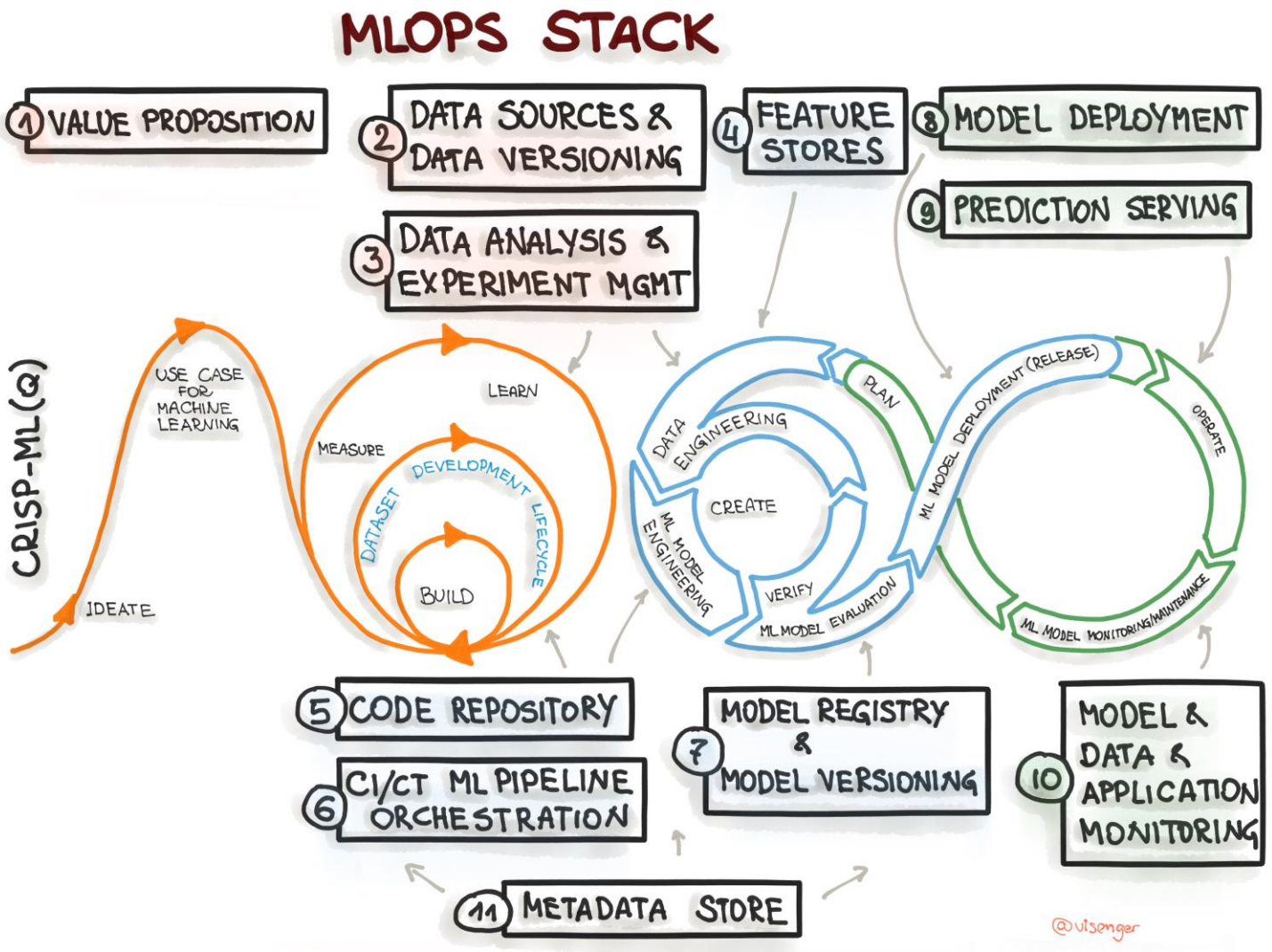
Analysis is a DAG:

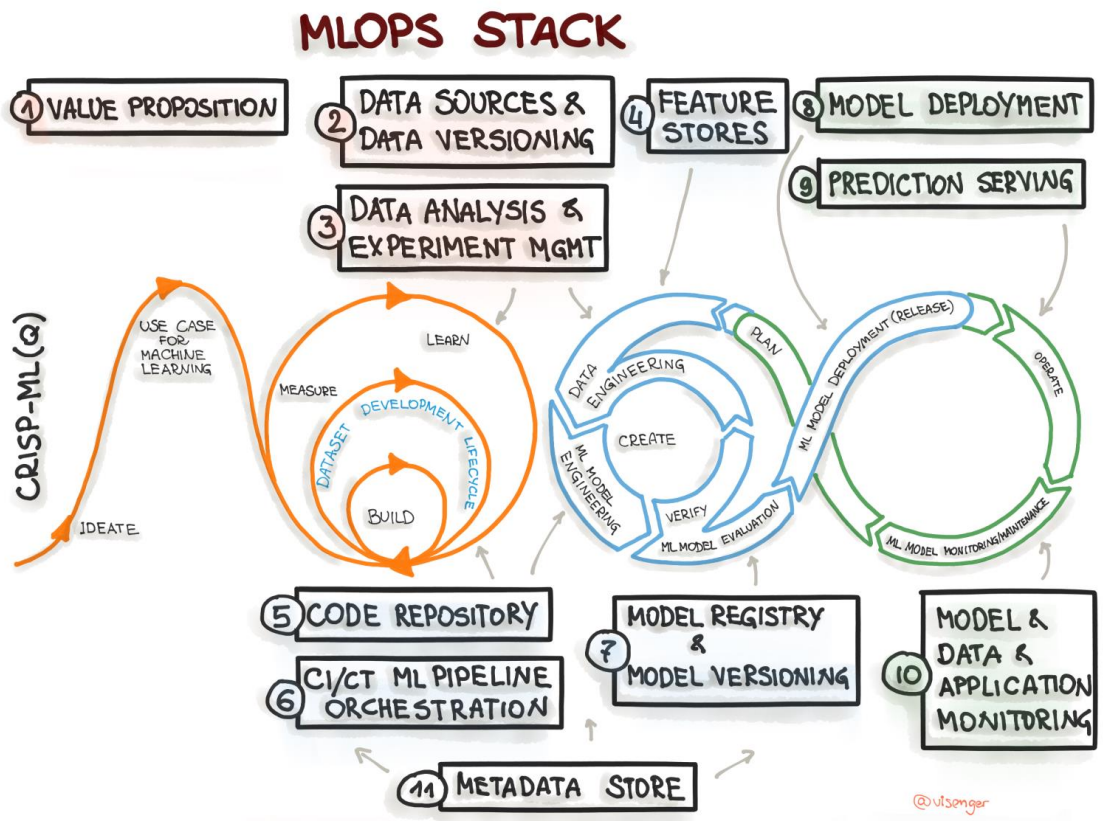
- Often in an analysis you have long-running steps that preprocess data or train models. If these steps have been run already you don't want to wait to rerun them every time.
- Use make for managing steps that depend on each other, especially the long-running ones.
- Other options: Airflow, Joblib, etc…

MLOps Stack Canvas

MLOps Stack Canvas

MLOps Stack Canvas allows to specify an architecture and infrastructure stack for Machine Learning Operations





MLOps Stack Canvas v1.0

PROJECT NAME: DATE:		TEAM MEMBERS:		
DATA ANALYSIS & EXPERIMENT MANAGEMENT 3	DATA SOURCES & DATA VERSIONING 2	VALUE PROPOSITION 1 *[Keep solving the real problem]*	CI/CT/CD: ML PIPELINE ORCHESTRATION 6 *[Building, testing, packaging and deploying of ML pipeline]*	MODEL REGISTRY & MODEL VERSIONING 7
				MODEL DEPLOYMENT 8
FEATURE STORE & WORKFLOWS 4 *[Keep feature computation consistent along the ML lifecycle]*	FOUNDATIONS 5 *[Reflecting DevOps]*	MLOPS DILEMMAS 0	MODEL & DATA & APPLICATION MONITORING 10	PREDICTION SERVING 9
METADATA STORE 11				

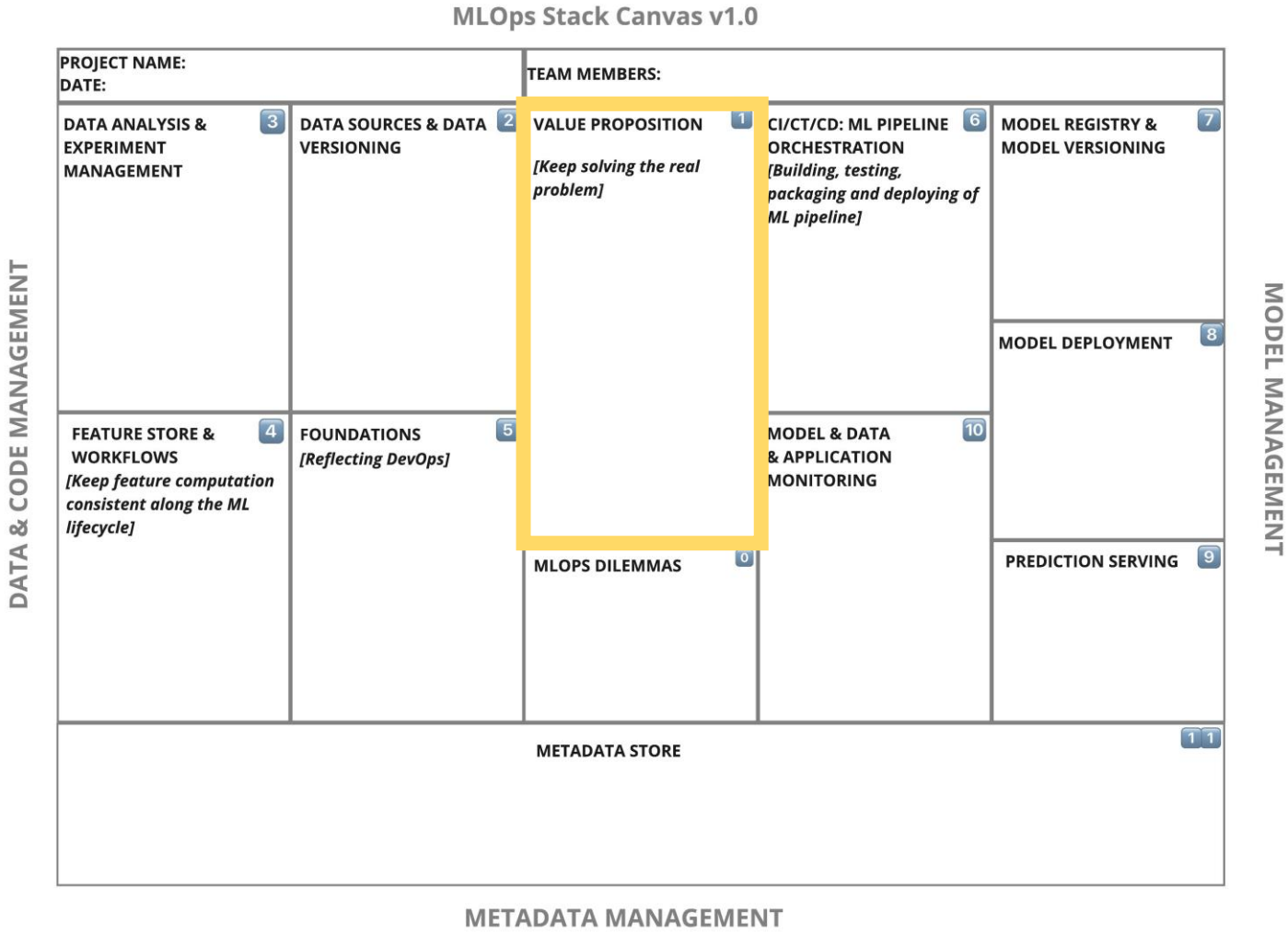
DATA & CODE MANAGEMENT

MODEL MANAGEMENT

METADATA MANAGEMENT

1.- Value Proposition

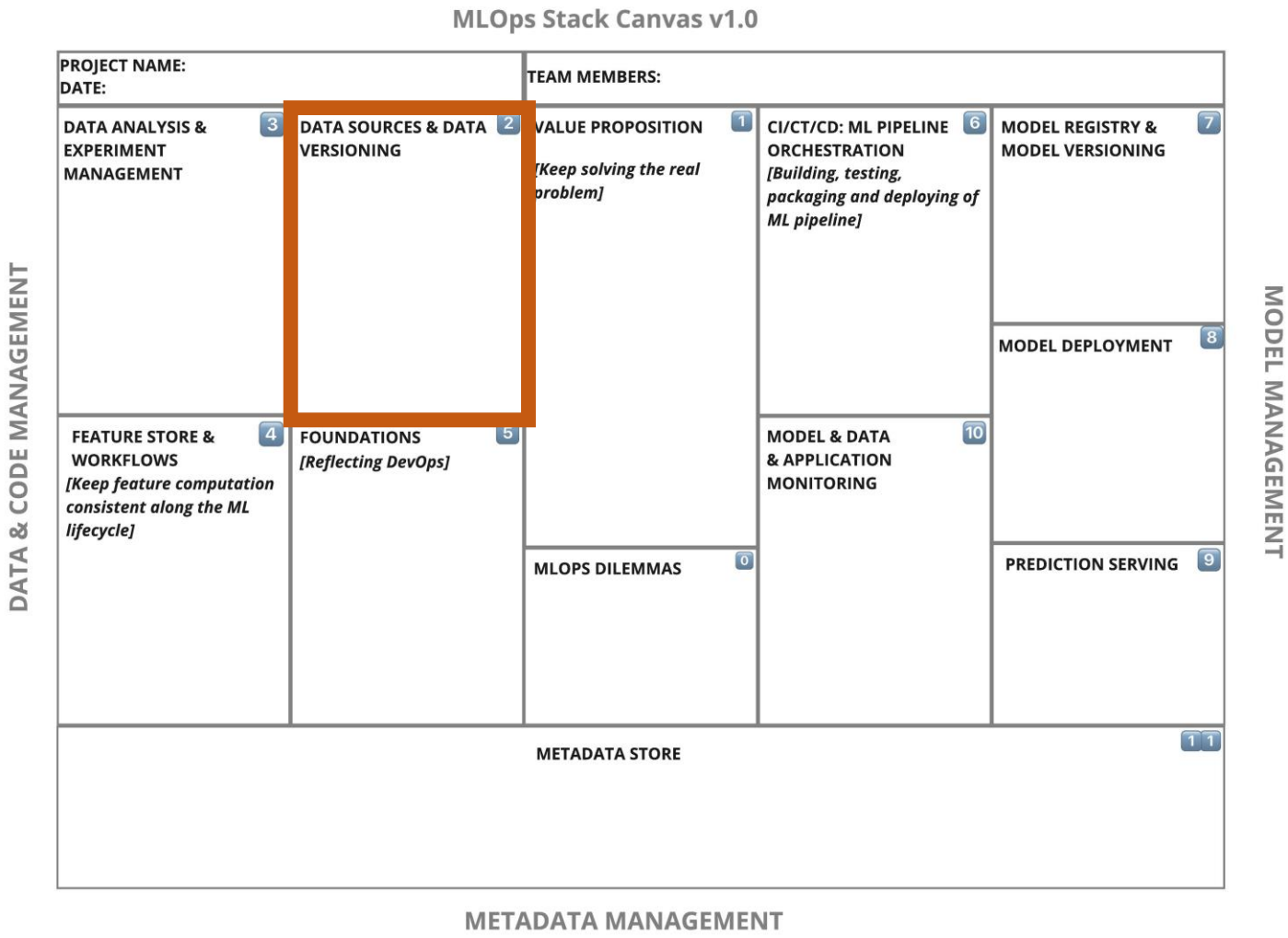
1. What are we trying to do for the end-user(s)?
2. What is the problem?
3. Why is this an important problem?
4. Who is our persona? (ML Engineer, Data Scientist, Operation/Business user)
5. Who owns the models in production?



Related:
https://www.louisdorard.com/machine-learning-canvas

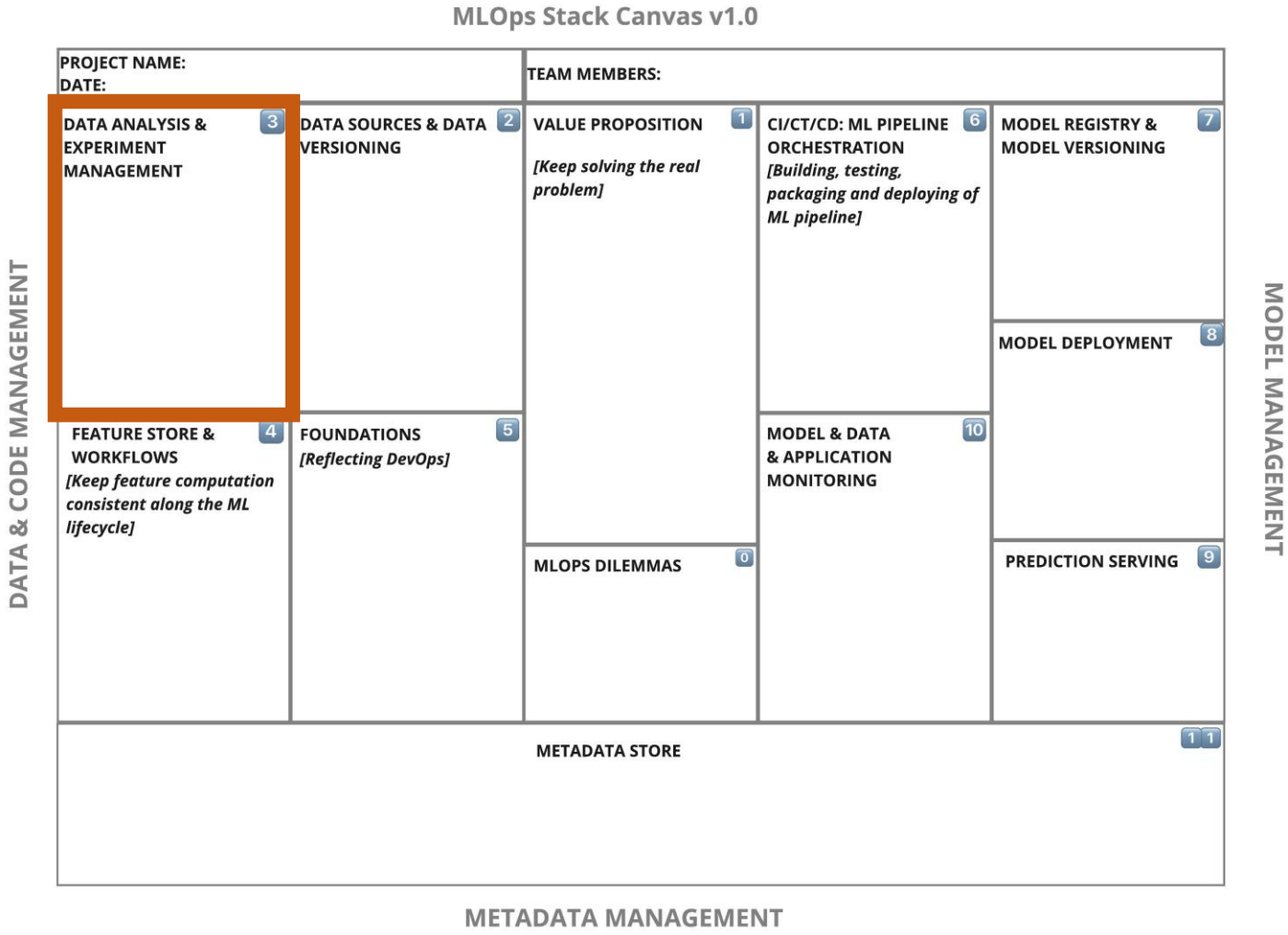
2.- Data Sources and Data Versioning

1. Is this data versioning optional or mandatory?
2. What data sources are available? (e.g., owned, public, earned, paid data)
3. What is the storage for the above data? (e.g., data lake, DWH)
4. Is manual labeling required? Do we have human resources for it?
5. How to version data for each trained model?
6. What tooling is available for data pipelines/workflows?



3.- Data Analysis and Experiment Management

1. What programming language to use for analysis? (R, Python, Scala, Julia. Or is SQL sufficient for analysis?)
2. Are there any infrastructure requirements for model training?
3. What ML-specific and business evaluation metrics need to be computed?
4. Reproducibility: What metadata about ML experiments is collected? (data sets, hyperparameters)
5. What ML Framework know-how is there?

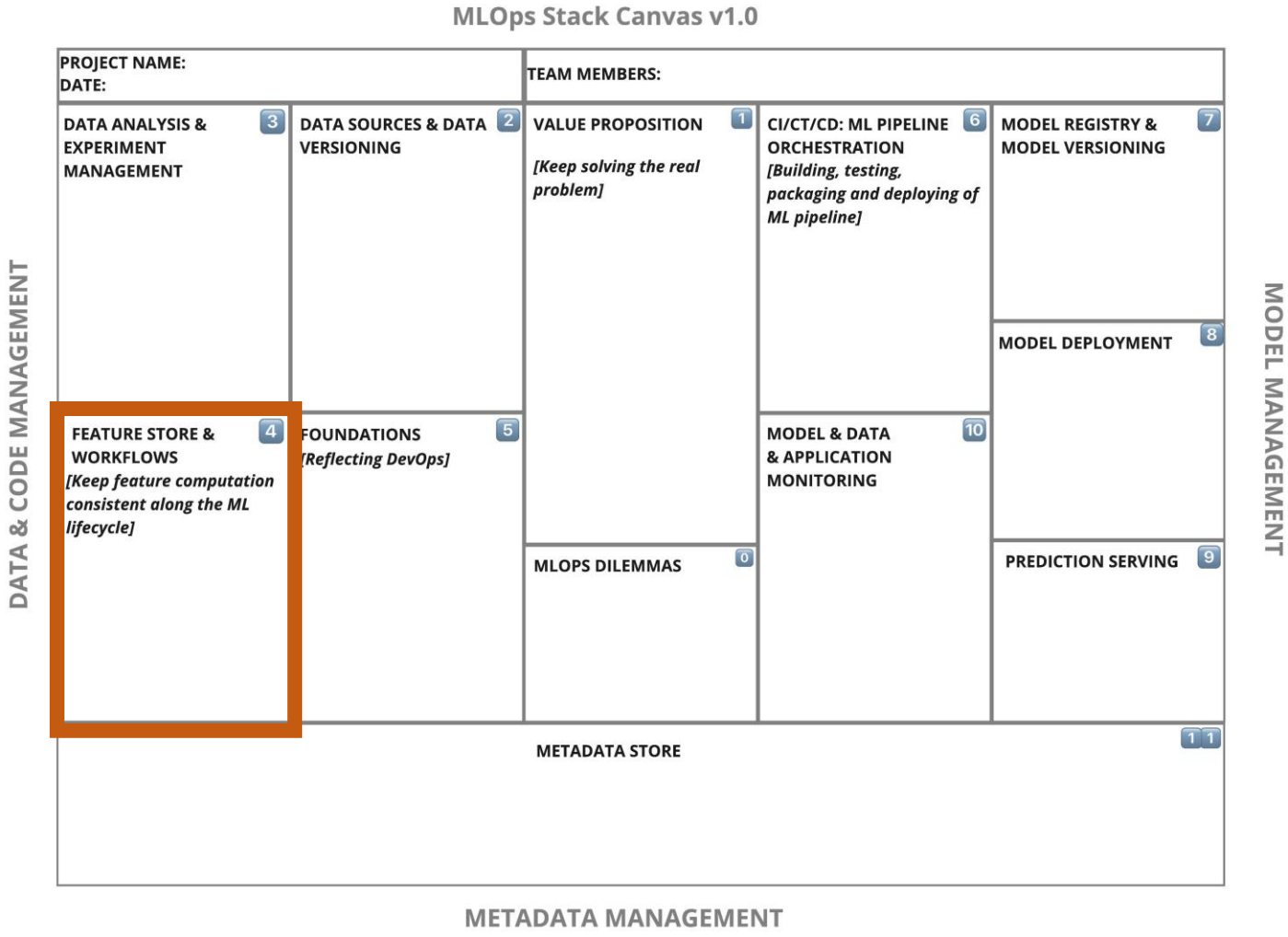


4.- Feature Store and Workflows

A feature store is defined as an interface for management, reproducibility, discovery, and reuse of features across ML projects and various data science teams.

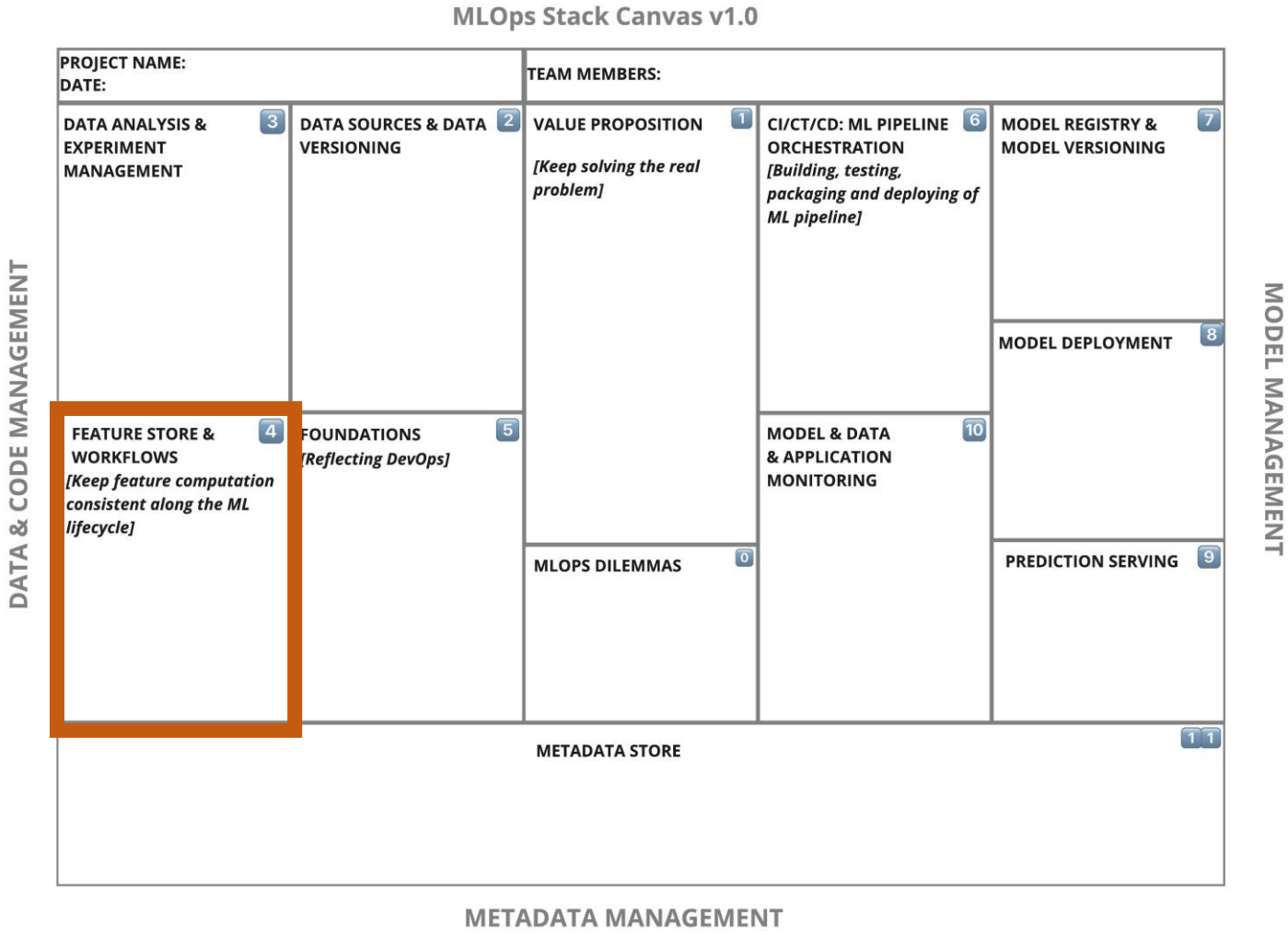
Feature stores promise the speed up in the development and operationalization of ML models.

*However, as an advanced component, feature stores **might add complexity**.*



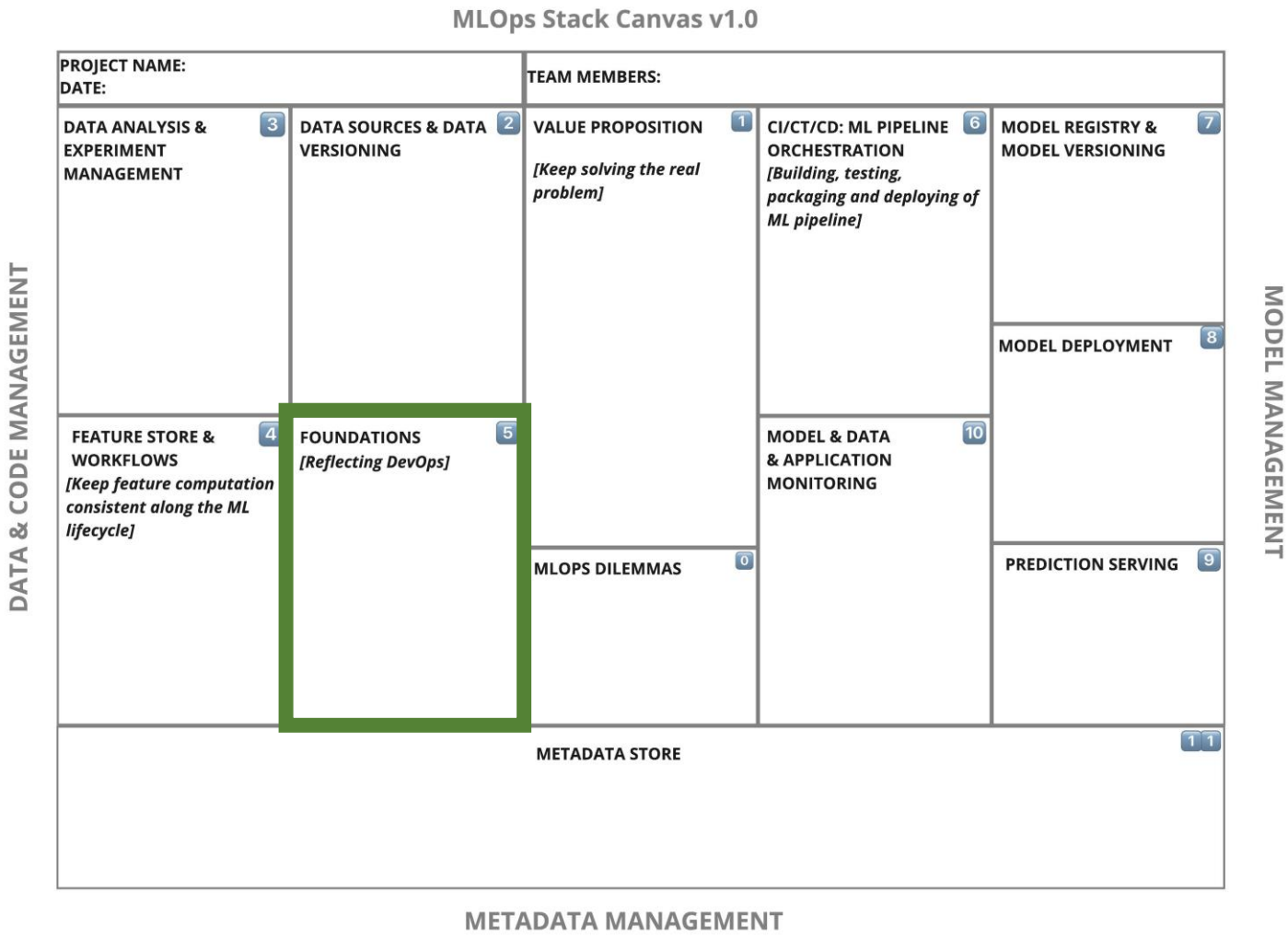
4.- Feature Store and Workflows

1. Is this optional or mandatory? Feature engineering has to be reproducible?
2. How are features computed (workflows) during the training and prediction phases?
3. What are infrastructure requirements for feature engineering?
4. “Buy or make” for feature stores?
5. What databases are involved in feature storage?
6. Do we design APIs for feature engineering?



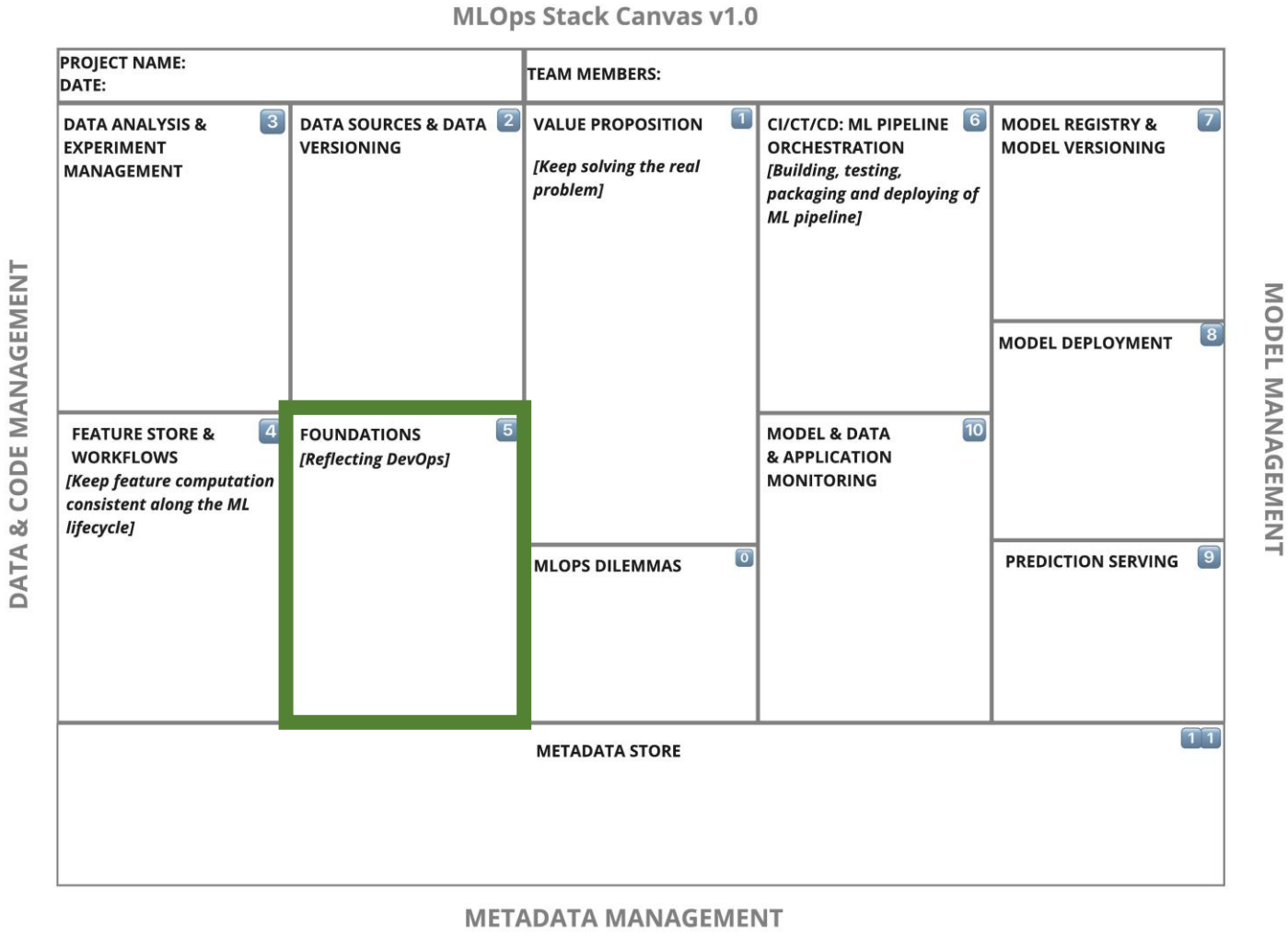
5.- Foundations (Reflecting DevOps)

1. How do we maintain the code? What source version control system is used?
2. How do we monitor the system performance?
3. Do we need versioning for notebooks?
4. Deployment and testing automation: What is the CI/CD pipeline for the codebase? What tools are used for it?



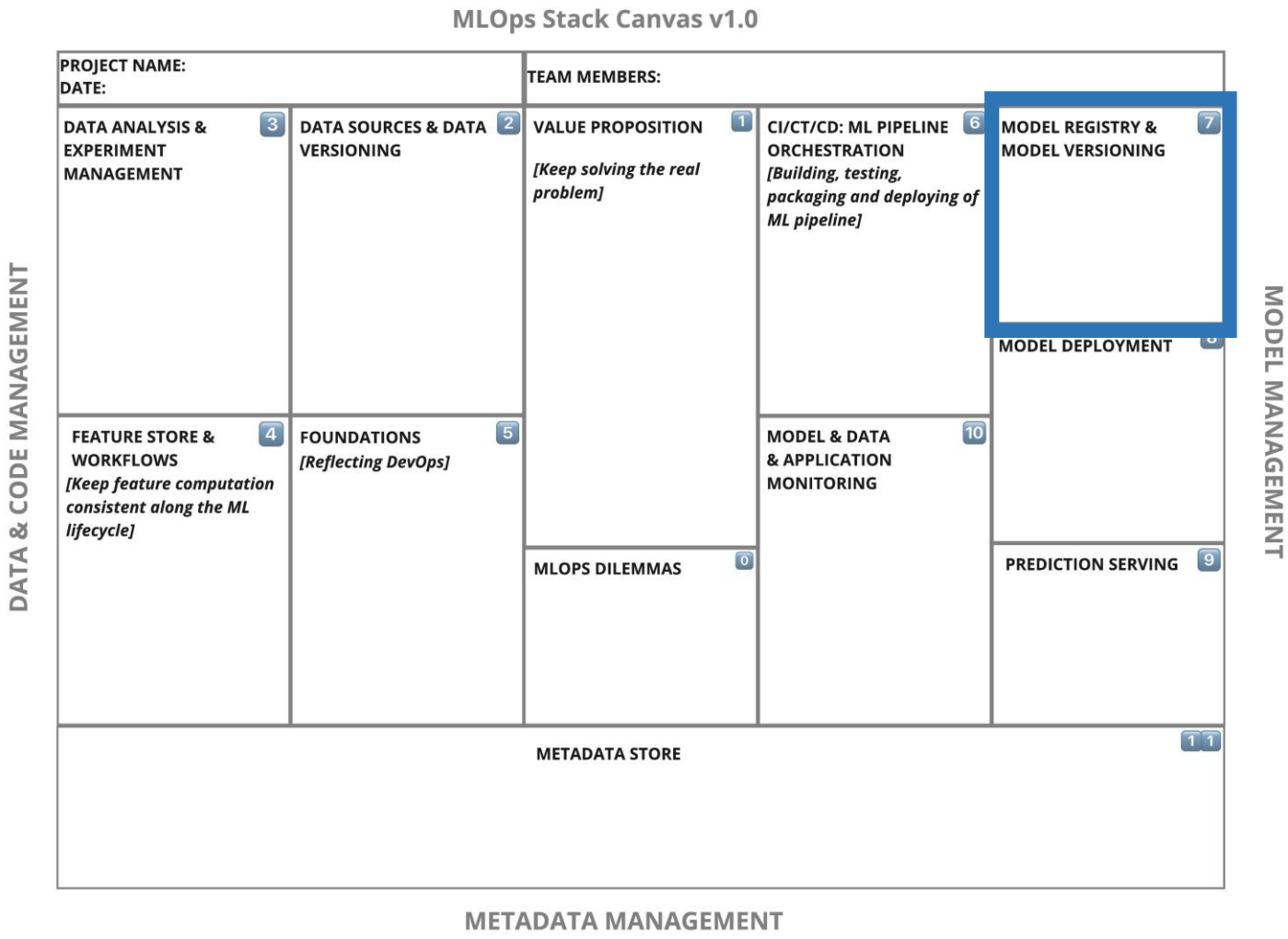
6.- CI, CT and CD: ML Pipeline Orchestration

1. How often are models expected to be retrained? What is the trigger for it (scheduled, event-based, or ad hoc)?
2. Where does this happen (locally or on a cloud)?
3. What is the formalized workflow for an ML pipeline? (e.g., Data prep -> model training -> model eval & validation) What tech stack is used?
4. Is distributed model training required? Do we have an infrastructure for the distributed training?
5. What is the workflow for the CI pipeline? What tools are used?
6. What are the non-functional requirements for the ML model (efficiency, fairness, robustness, interpretability, etc.)? How are they tested? Are these tests integrated into the CI/CT workflow?



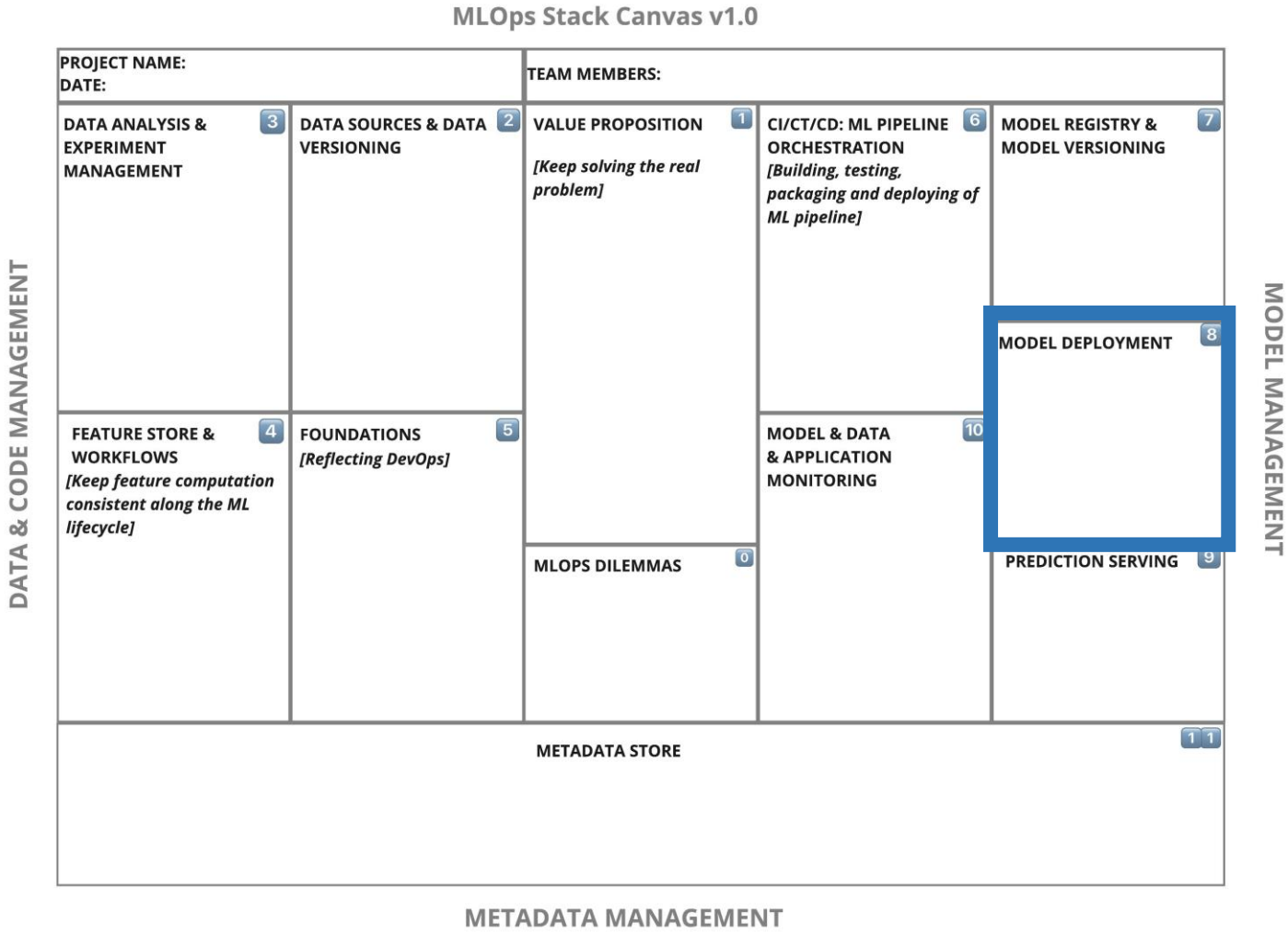
7.- Model Registry and Model Versioning

1. Is this optional or mandatory? The model registry might be mandatory if you have multiple models in production and need to track them all. The reproducibility requirement might be the reason that you need the model versioning.
2. Where should new ML models be stored and tracked?
3. What versioning standards are used? (e.g., semantic versioning)



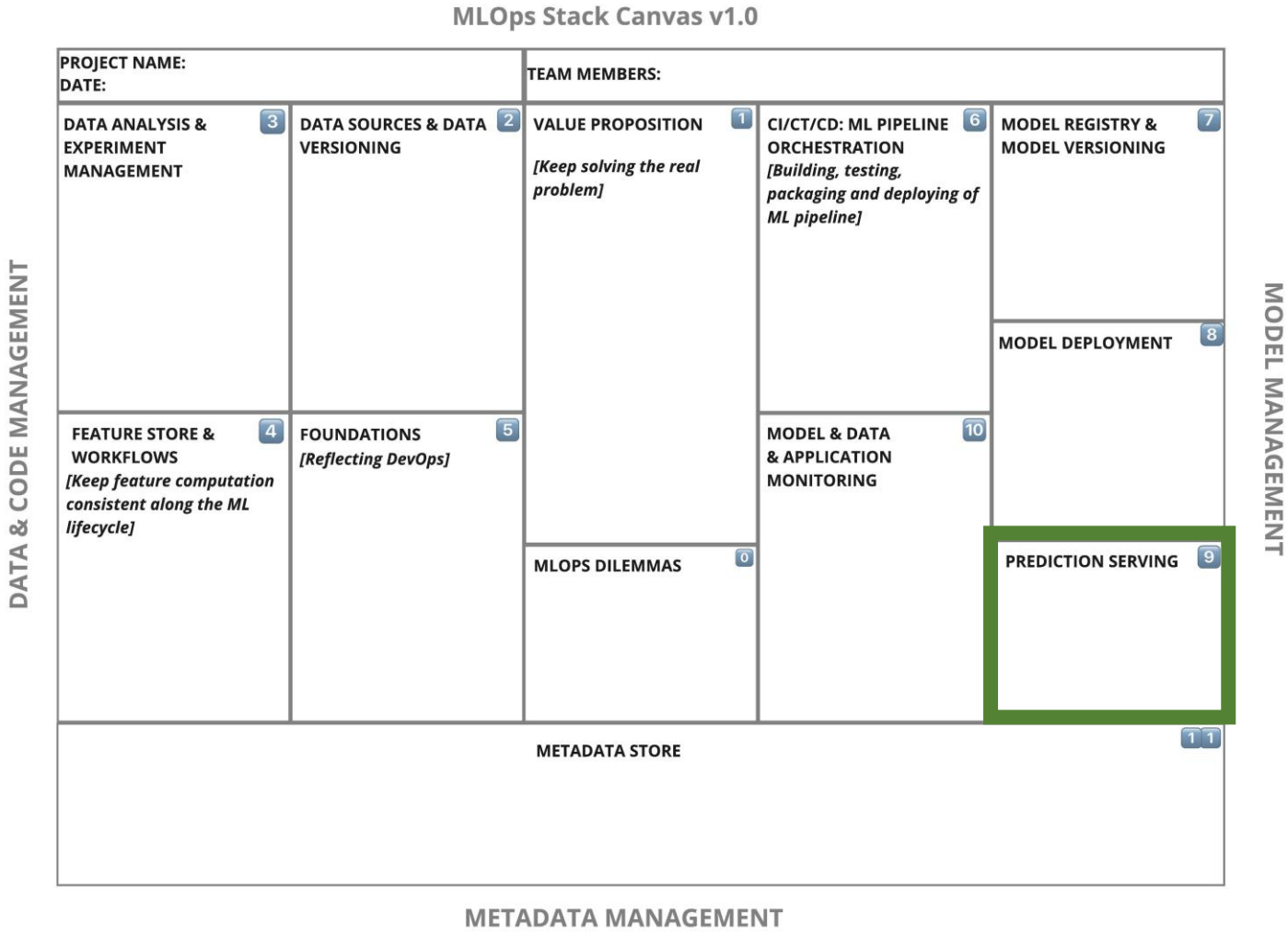
8.- Model Deployment

1. What is the delivery format for the model?
2. What is the expected time for changes? (Time from commit to production)
3. What is the target environment to serve predictions?
4. What is your model release policy? Is A/B testing or multi-armed bandits testing required? (e.g., for measuring the effectiveness of the new model on business metrics and deciding what model should be promoted into the production environment)
5. What is your deployment strategy? (e.g. shadow/canary deployment required?)



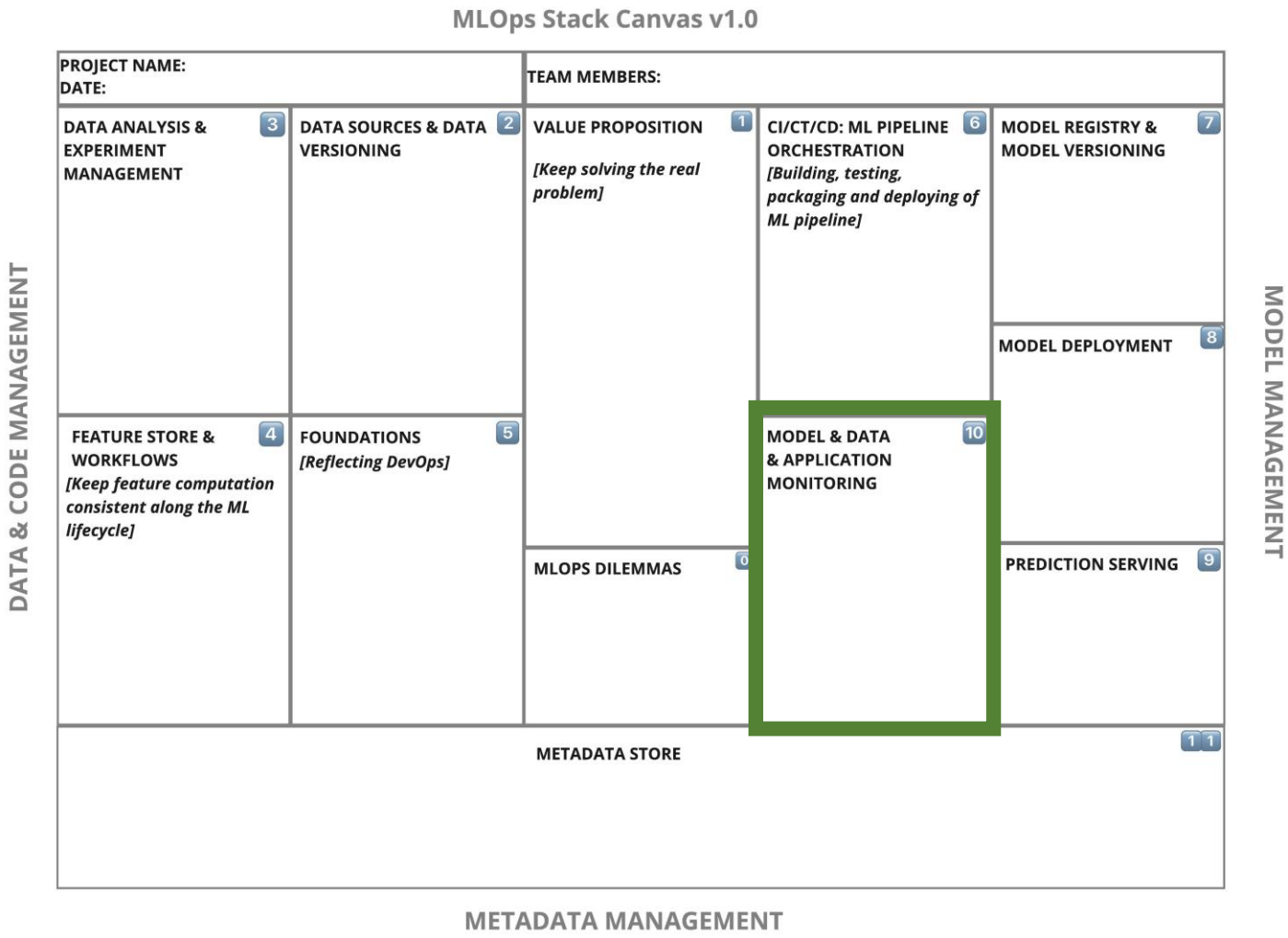
9.- Prediction Serving

1. What is the serving mode? (batch or online)
2. Is distributed model serving required?
3. Is multi-model prediction serving required?
4. Is pre-assertion for input data implemented?
5. What fallback method for an inadequate model output (post-assertion) is implemented? (Do we have a heuristic benchmark?)
6. Do you need ML inference accelerators (TPUs)?
7. What is the expected target volume of predictions per month or hours?



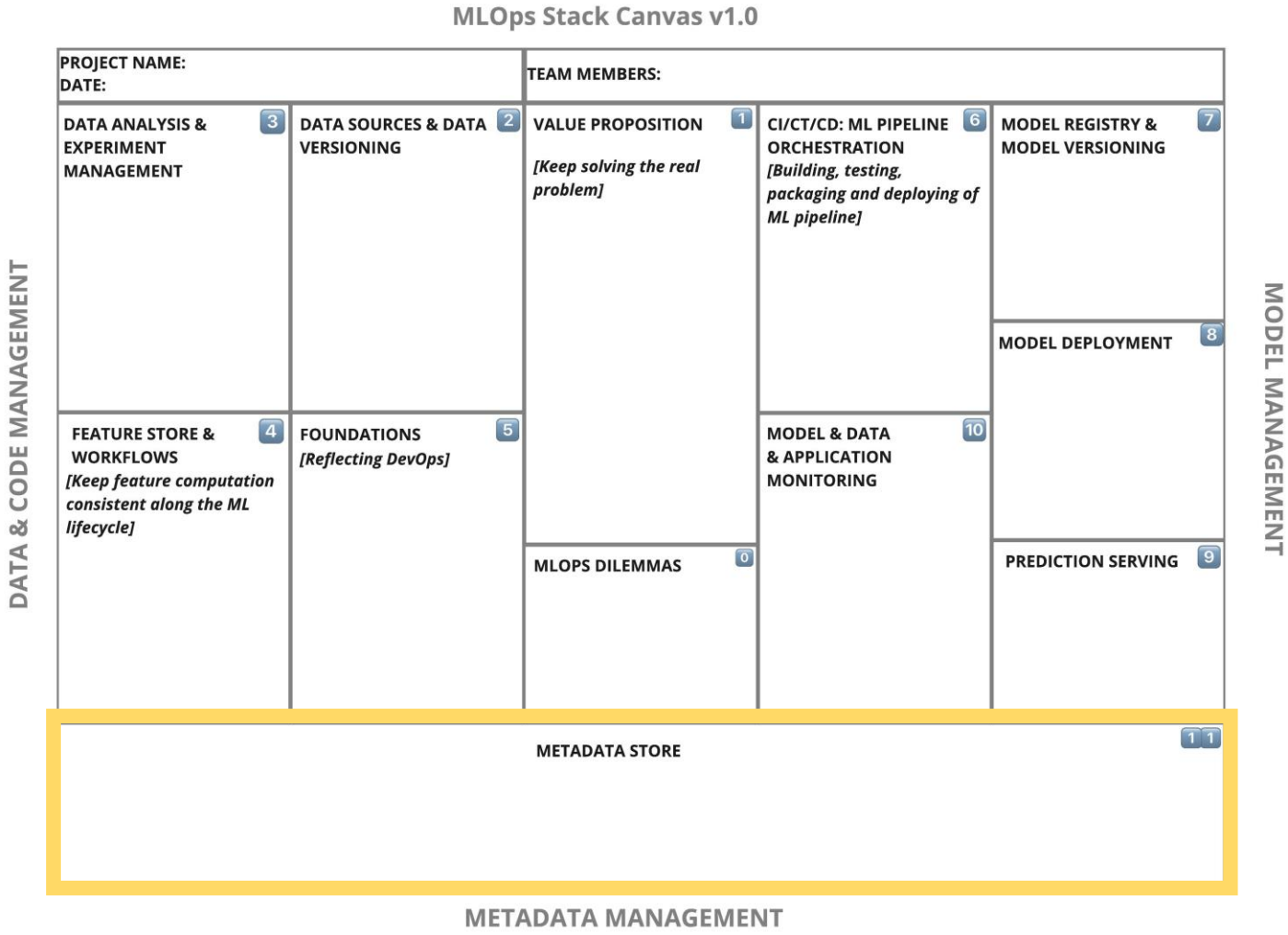
10.- ML Model, Data, and System Monitoring

1. Is this optional or mandatory? For instance, do you need to assess the effectiveness of your model during prediction serving? Do you need to monitor your model for performance degradation and trigger an alert if your model starts performing badly? Is the model retraining based on events such as data or concept drift?
2. What ML metrics are collected?
3. What domain-specific metrics are collected?
4. How is the model performance decay detected? (Data Monitoring)
5. How is the data skew detected? (Data Monitoring)
6. What operational aspects need to be monitored? (e.g., model prediction latency, CPU/RAM usage)
7. What is the alerting strategy? (thresholds)
8. What triggers the model re-training? (ad hoc, event-based, or scheduled)



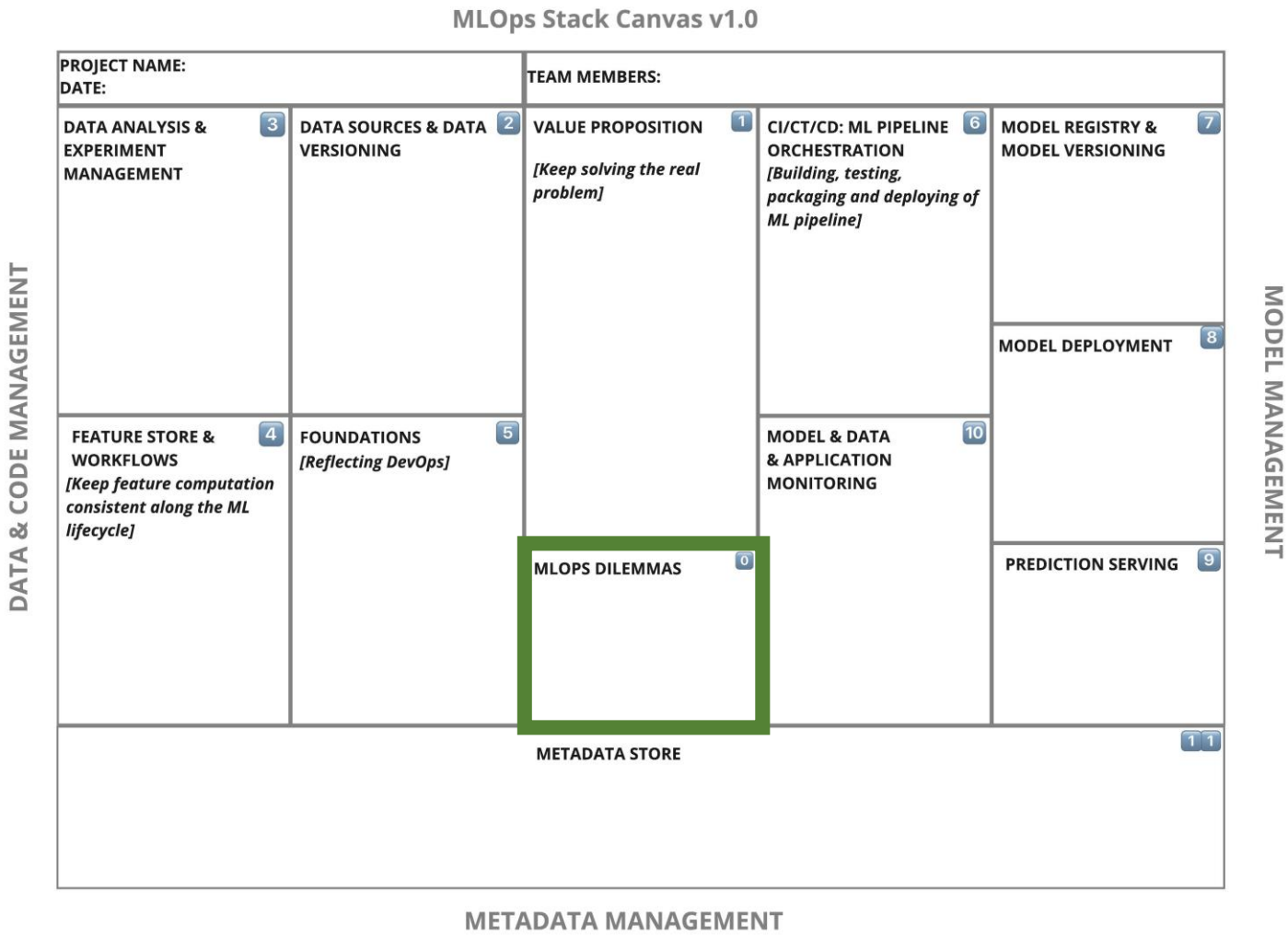
10.- Metadata Store

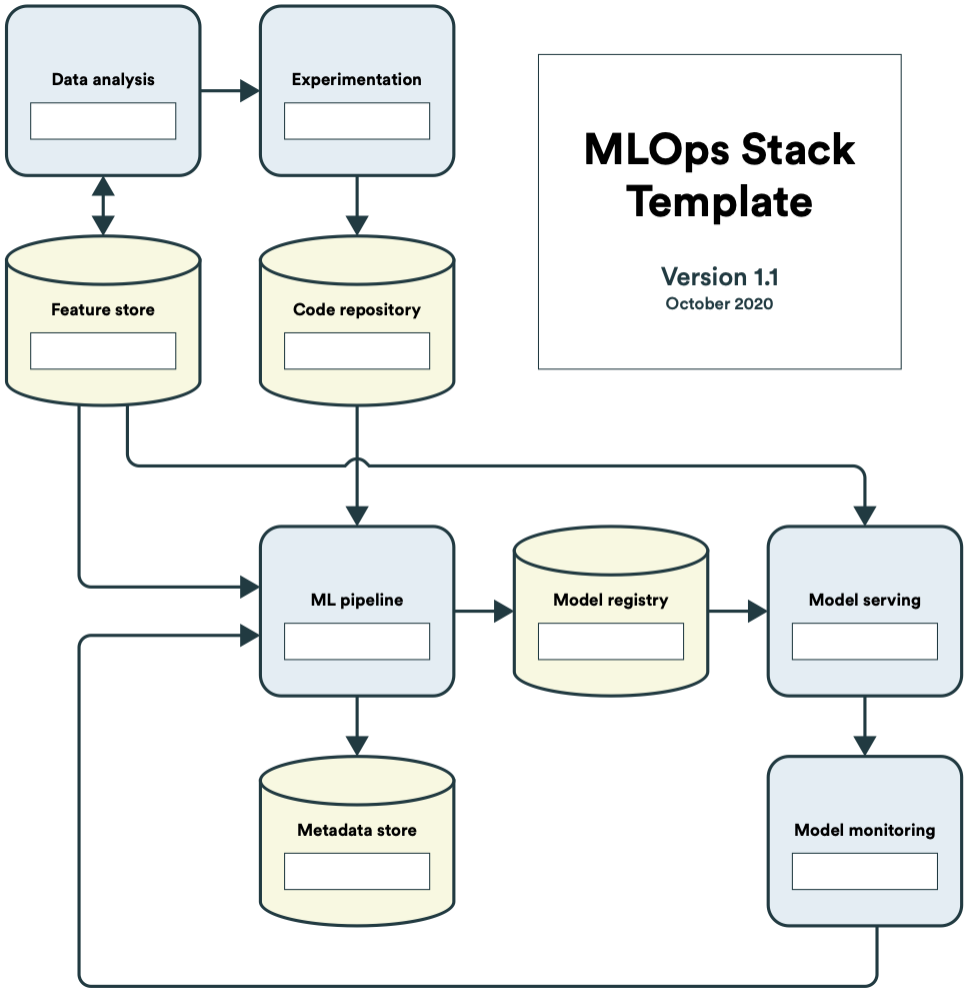
1. What kind of metadata in code, data, and model management need to be collected? (e.g., the pipeline run ID, trigger, performed steps, start/end timestamps, train/test dataset split, hyperparameters, model object, various statistics/profiling, etc.)
2. Are any ML governance processes included in the MLOps lifecycle? What metadata will be required?
3. What is the documentation strategy: Do we treat documentation as a code? (examples: Datasheets for Datasets and Model Card for Model Reporting)
4. What operational metrics need to be collected? E.g., time to restore, change fail percentage.



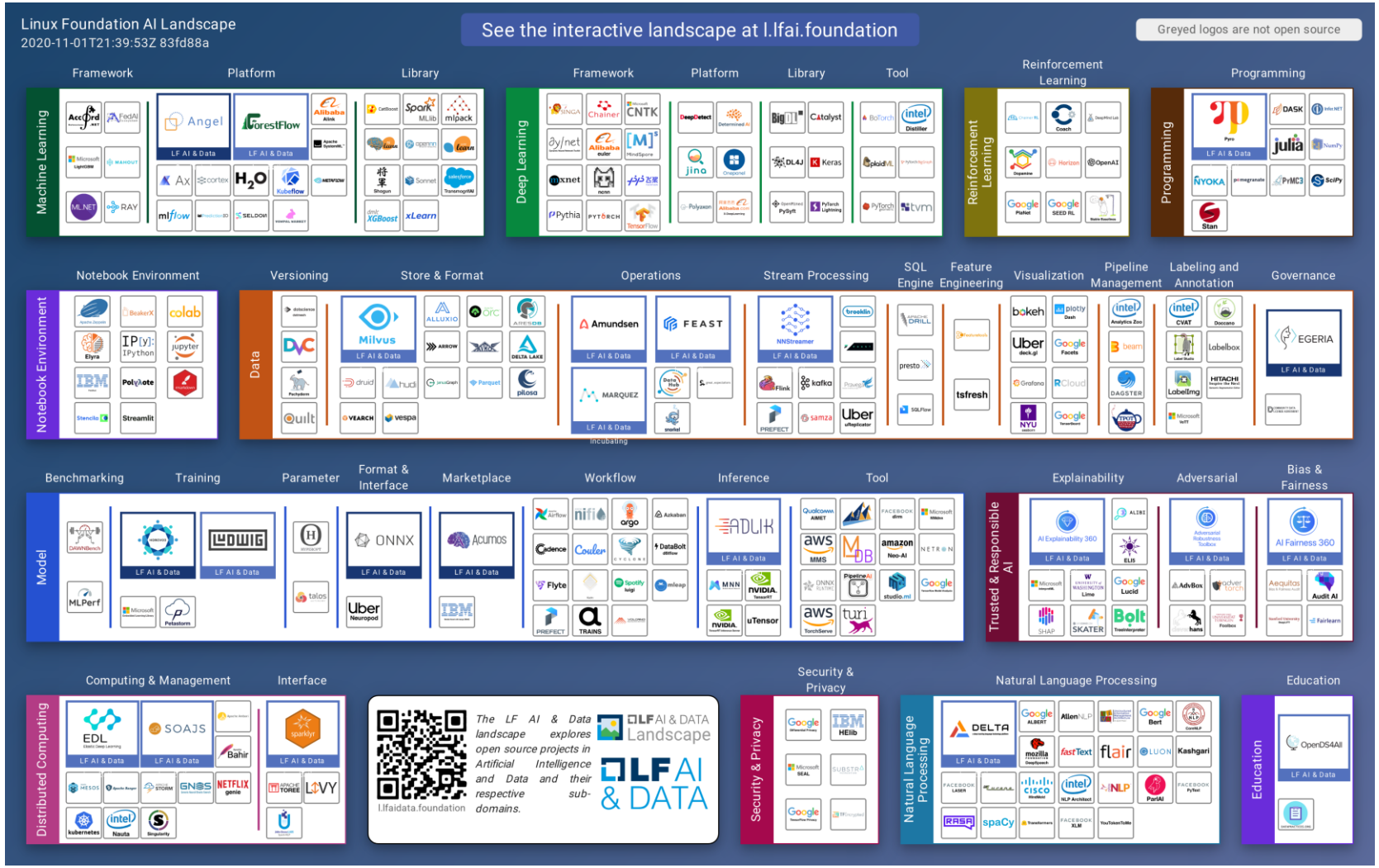
0.- Dilemmas of MLOps

1. **Tooling**: Should we buy, use existing open-source or build in-house tools for any of the MLOps components? What are the risks, trade-offs, and impacts of each of the decisions?
2. **Platforms**: Should we agree on one MLOps platform or create a hybrid solution? What are the risks, trade-offs, and impacts of each of the decisions?
3. **Skills**: How expensive is it to either acquire or educate our own machine learning engineering talents?




Data analysis
Experimentation
MLOps Stack Template
Version 1.1
October 2020
Feature store
Code repository
ML pipeline
Model registry
Model serving
Metadata store
Model monitoring

MLOps Setup Components	Tools
Data Analysis	Python, Pandas
Source Control	Git
Test & Build Services	PyTest & Make
Deployment Services	Git, DVC
Model & Dataset Registry	DVC[aws s3]
Feature Store	Project code library
ML Metadata Store	DVC
ML Pipeline Orchestrator	DVC & Make


Linux Foundation AI Landscape
2020-11-01T21:39:53Z 83fd88a
See the interactive landscape at l.lfai.foundation
Greyed logos are not open source
Machine Learning
Framework
Platform
Library
Deep Learning
Framework
Platform
Library
Tool
Reinforcement Learning
Reinforcement Learning
Programming
Programming
Notebook Environment
Notebook Environment
Data
Versioning
Store & Format
Operations
Stream Processing
SQL Engine
Feature Engineering
Visualization
Pipeline Management
Labeling and Annotation
Governance
Model
Benchmarking
Training
Parameter
Format & Interface
Marketplace
Workflow
Inference
Tool
Trusted & Responsible AI
Explainability
Adversarial
Bias & Fairness
Distributed Computing
Computing & Management
Interface
The LF AI & Data landscape explores open source projects in Artificial Intelligence and Data and their respective sub-domains.
l.lfaidata.foundation
LF AI & DATA Landscape
LF AI & DATA
Security & Privacy
Security & Privacy
Natural Language Processing
Natural Language Processing
Education
Education
Incubating

Fuente

https://ml-ops.org/