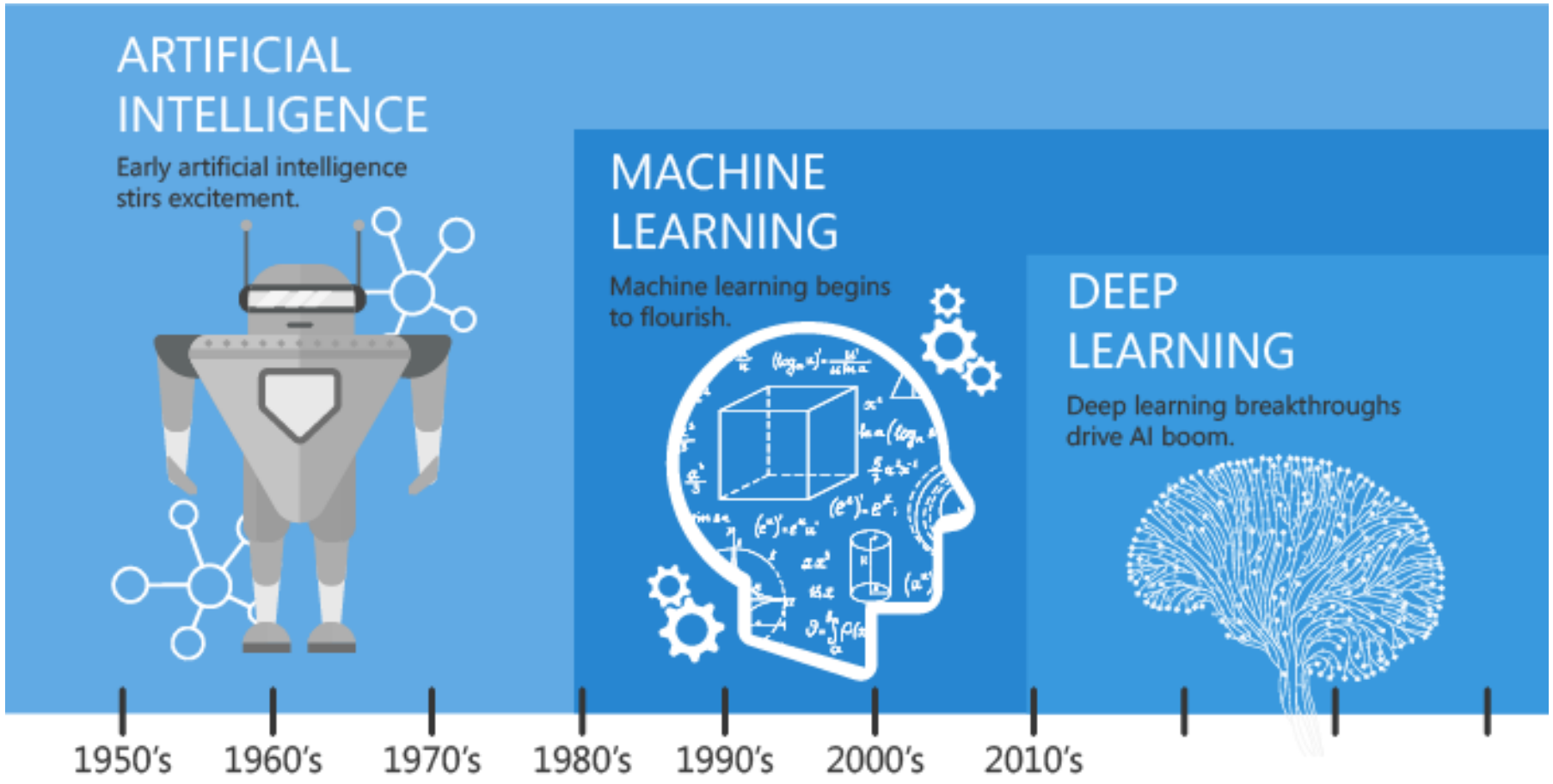
# Challenges of Machine Learning

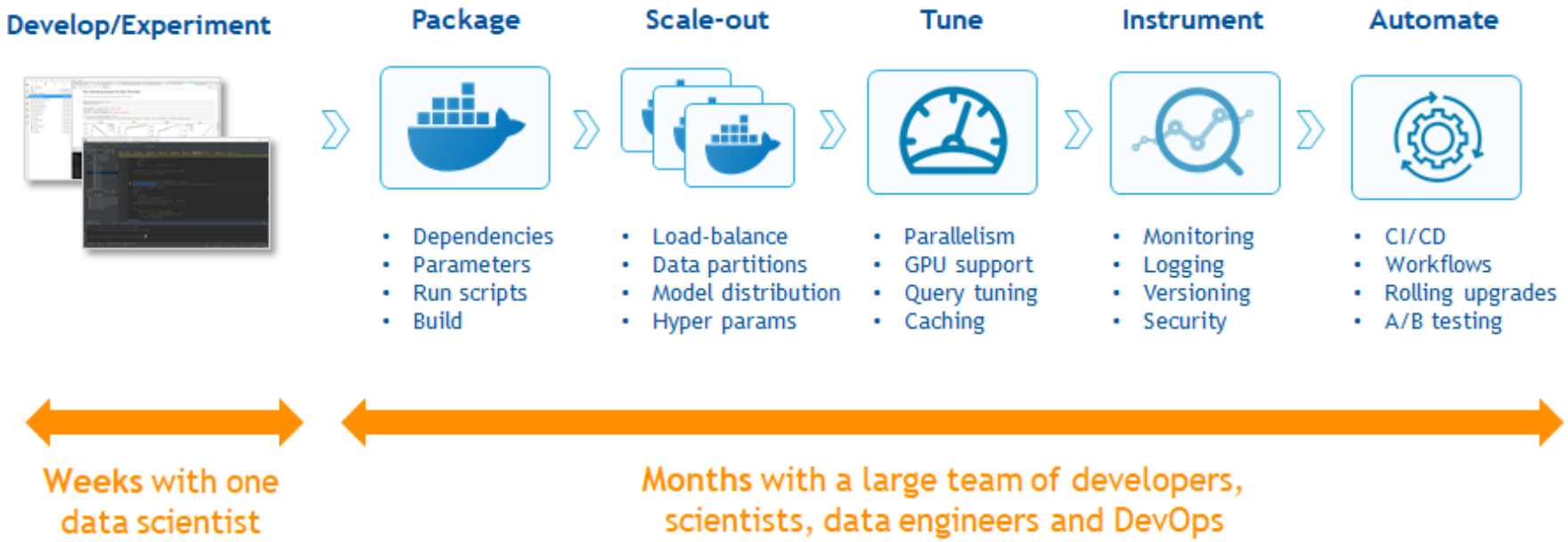
# History of AI



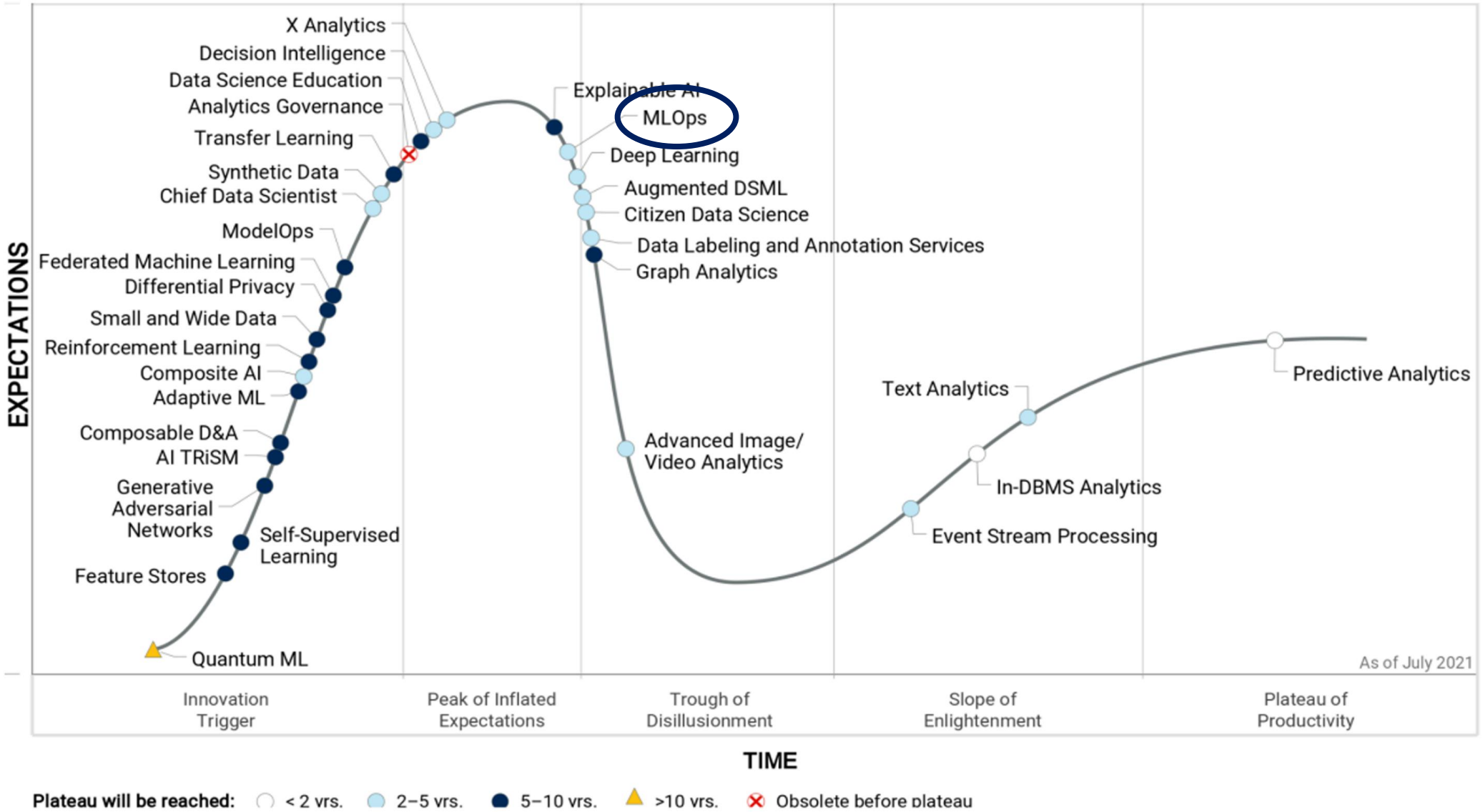

**85%** of models don´t
reach production

# Challenges

According to a survey, **55% of companies** have never deploy a model. Main **reasons**: lack of talent, lack of processes to manage change and lack of automated systems.



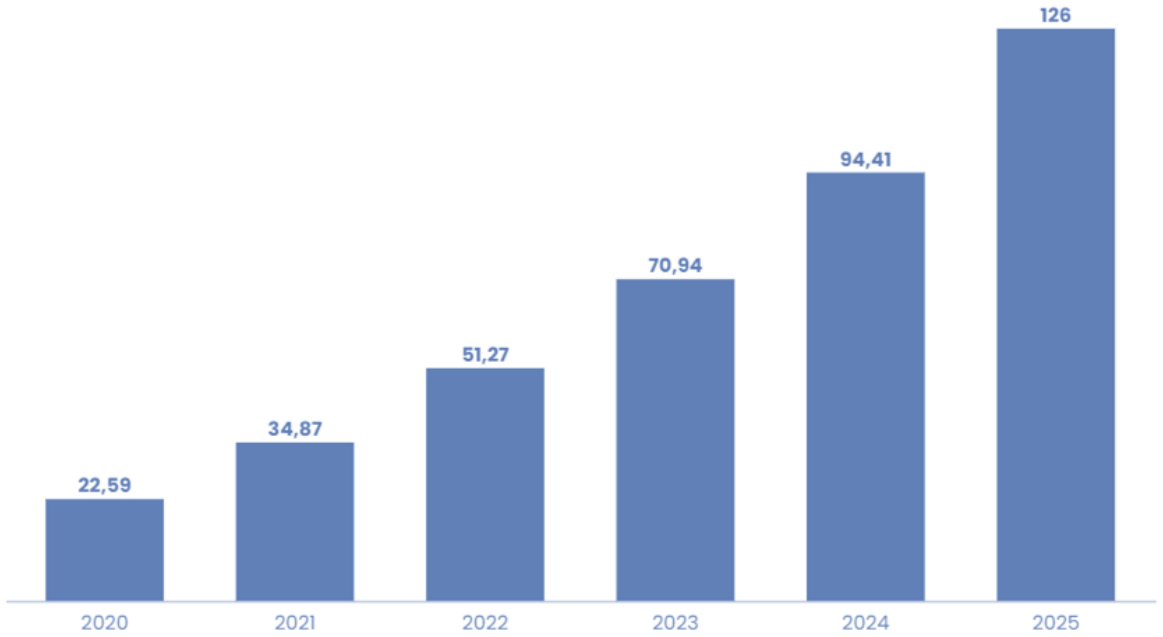
# Trends


X Analytics
Decision Intelligence
Data Science Education
Analytics Governance
Transfer Learning
Synthetic Data
Chief Data Scientist
ModelOps
Federated Machine Learning
Differential Privacy
Small and Wide Data
Reinforcement Learning
Composite AI
Adaptive ML
Composable D&A
AI TRiSM
Generative Adversarial Networks
Self-Supervised Learning
Feature Stores
Quantum ML
Explainable AI
MLOps
Deep Learning
Augmented DSML
Citizen Data Science
Data Labeling and Annotation Services
Graph Analytics
Advanced Image/ Video Analytics
Text Analytics
In-DBMS Analytics
Event Stream Processing
Predictive Analytics
As of July 2021
EXPECTATIONS
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
TIME
Plateau will be reached:
< 2 yrs.
2–5 yrs.
5–10 yrs.
>10 yrs.
Obsolete before plateau

# Benefits

Those organizations that put **AI into production** saw their profit margin increase from **3% to 15%.**

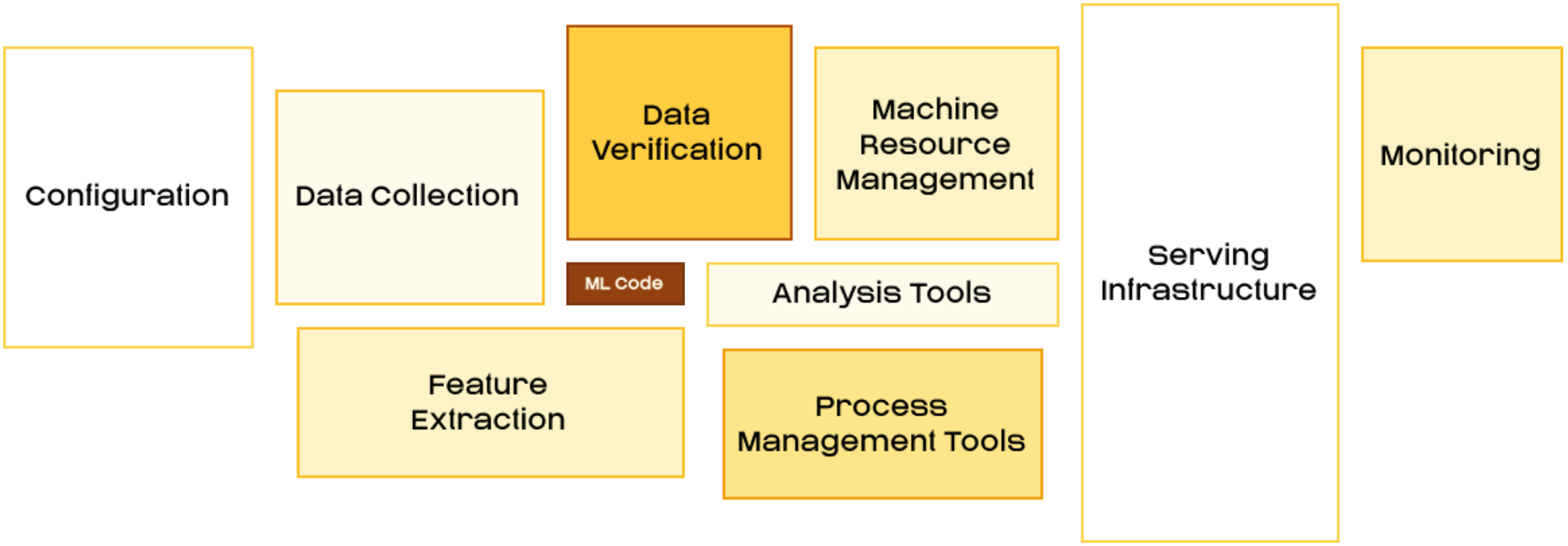
The MLOps market was estimated at **$23.2 billion** in 2019. It is projected to reach **$126 billion** by 2025 due to rapid adoption



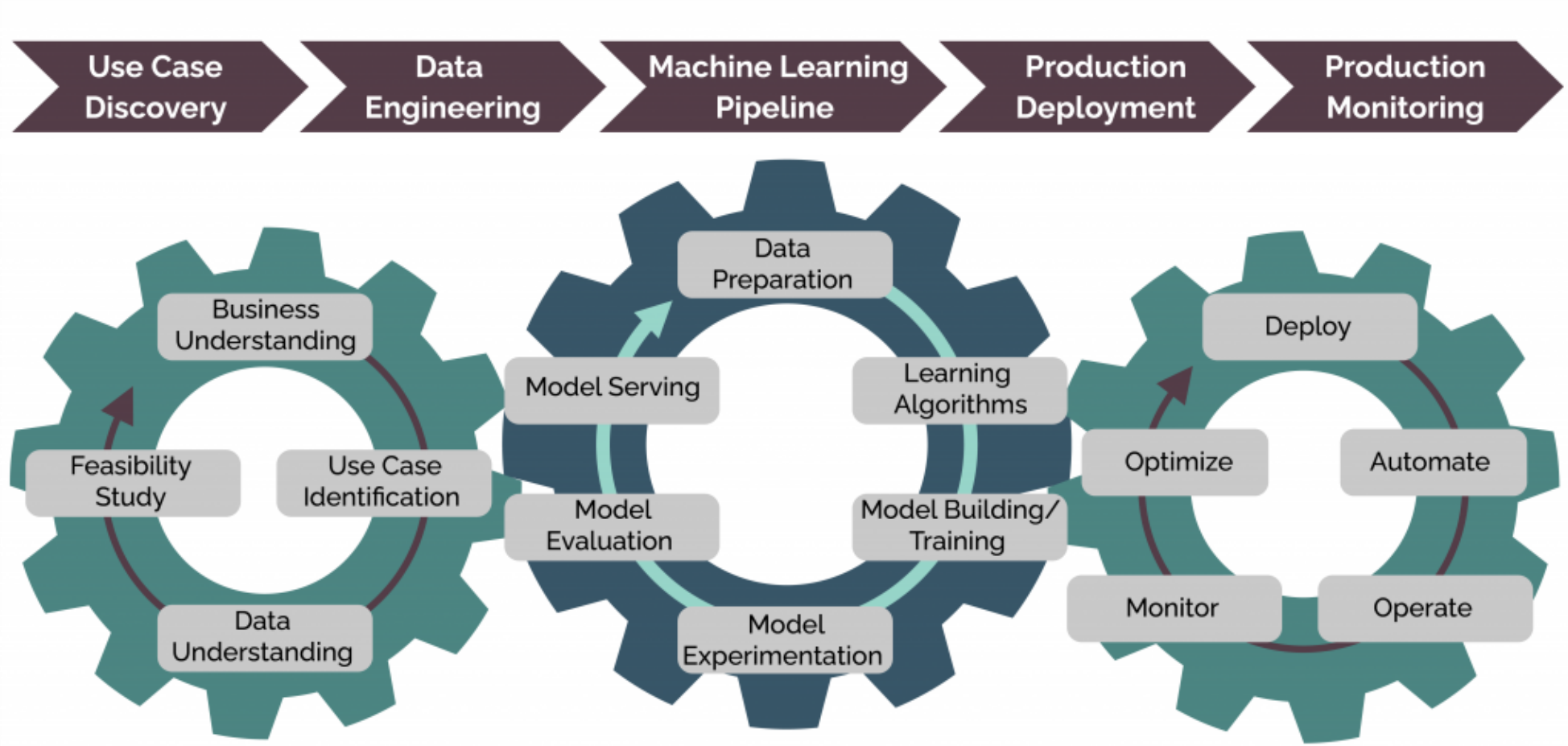

*AI software market revenue from 2020 to 2025 [billions of dollars]*

# What is MLOps?

Model creation must be **scalable, collaborative and reproducible**. The principles, tools and techniques that make models scalable, collaborative and reproducible are known as **MLOps.**

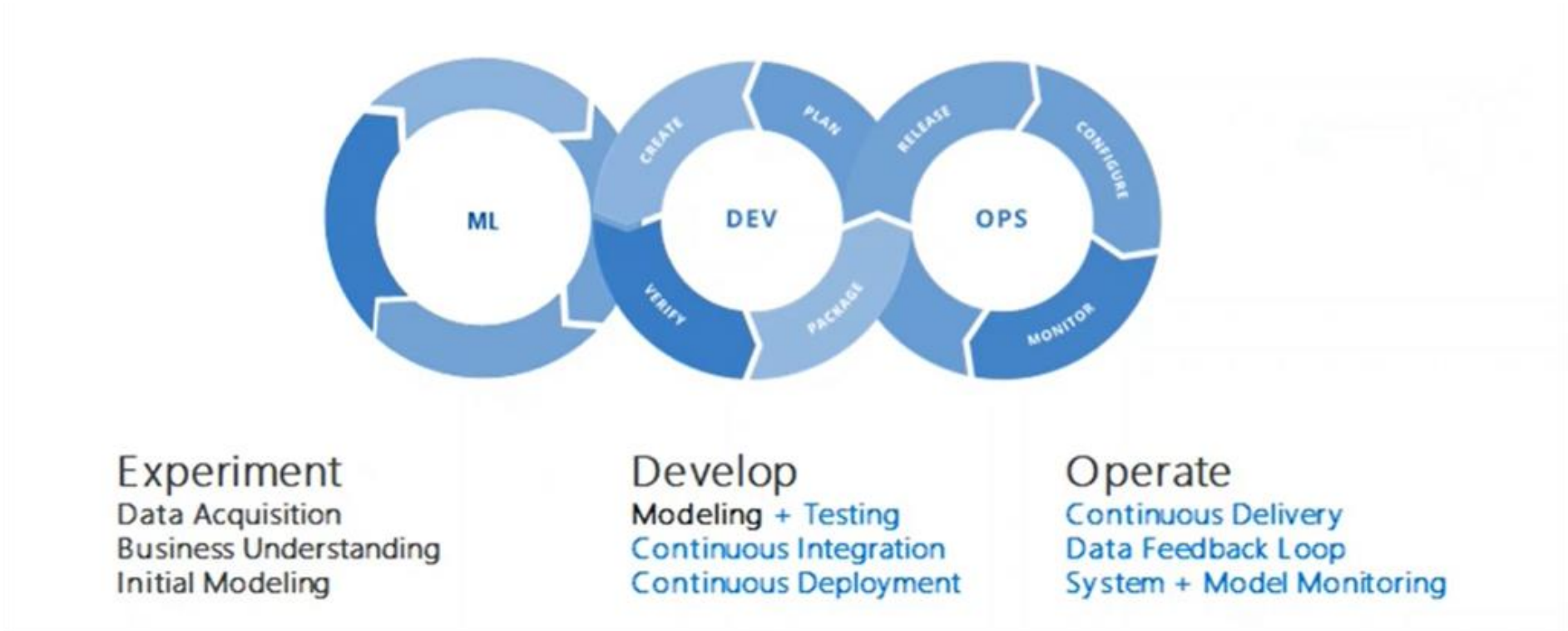


# MLOps Process

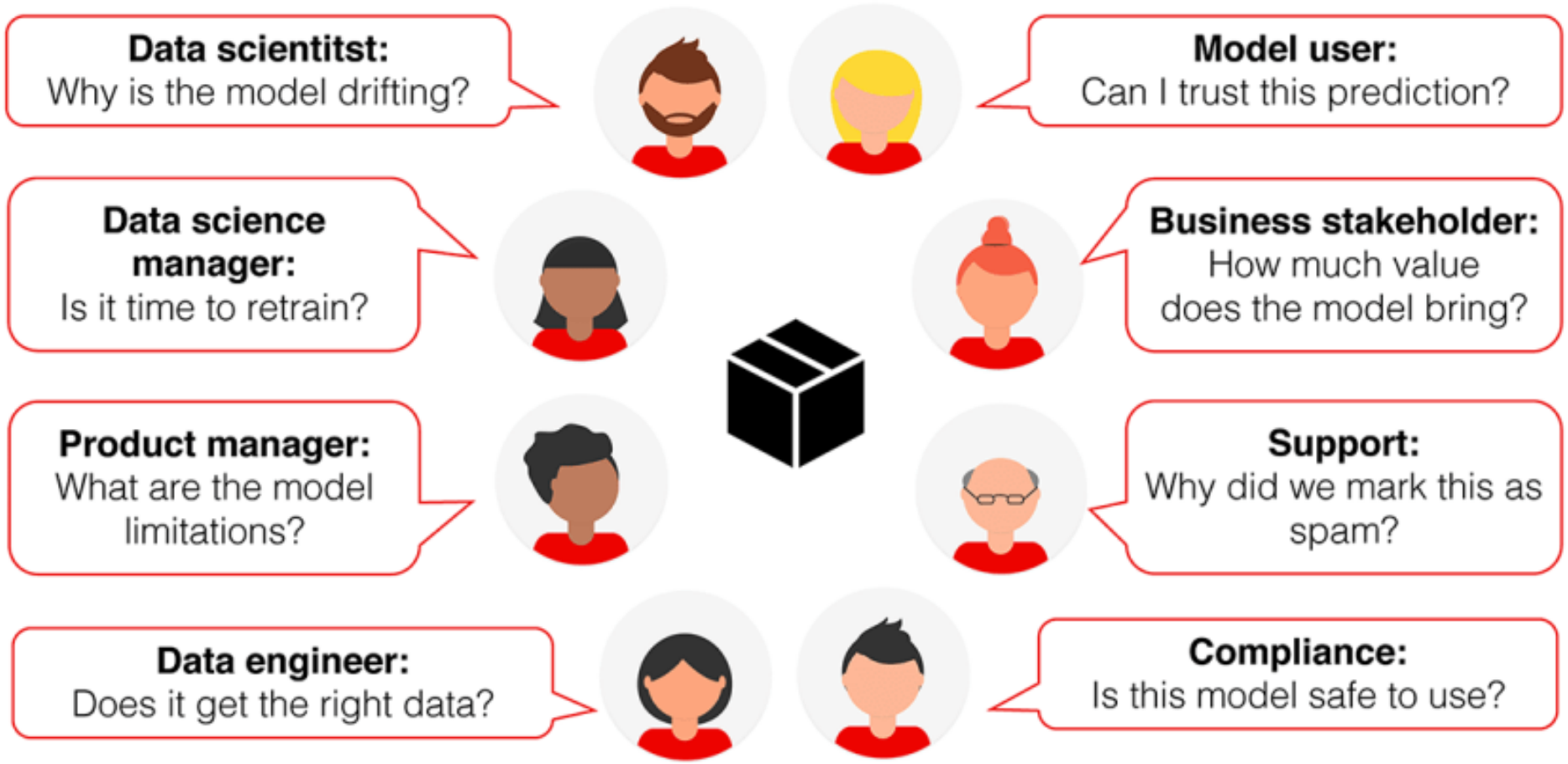


# DevOps & DataOps

**DevOps applied to Machine Learning** is known as MLOps. **DataOps** implies a set of rules that ensure a high quality of data to train models.



# Roles in MLOps



# MLOps Fundamentals

# Chanllenges addressed by MLOps

**Versioning**

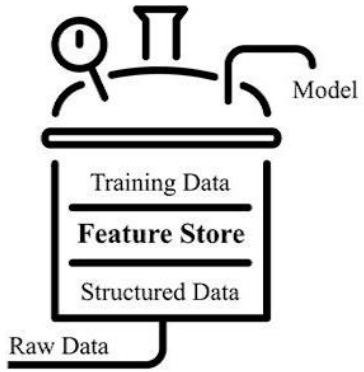
Tools such as Git and GitHub are used in code version control. Also, **data and artifacts** are versioned to ensure **reproducibility..**

**Model tracking**

Models in production can be **degraded** over time due to **data drift.**

**Feature Generation**

It requires a lot of resources. MLOps allows to **reuse functions**. So, you can focus on the design/test of the model.

# Parts of MLOps

**Feature store**
Stores the functions that has been used in model training

**Data Versioning**
Data version control ensures reproducibility and facilitates auditing.

**Metadata store**
It is critical for reproducibility. Everything should be registered, from model´ seed to evaluation metrics …

**Model Versioning**
Allows you to switch between models in real time or serve different models to monitor

**Model Registration**
Once a model has been trained, it is stored in a model registry with it´s metadata

**Model serving**
Serving a model means creating endpoints that can be used to run predictions

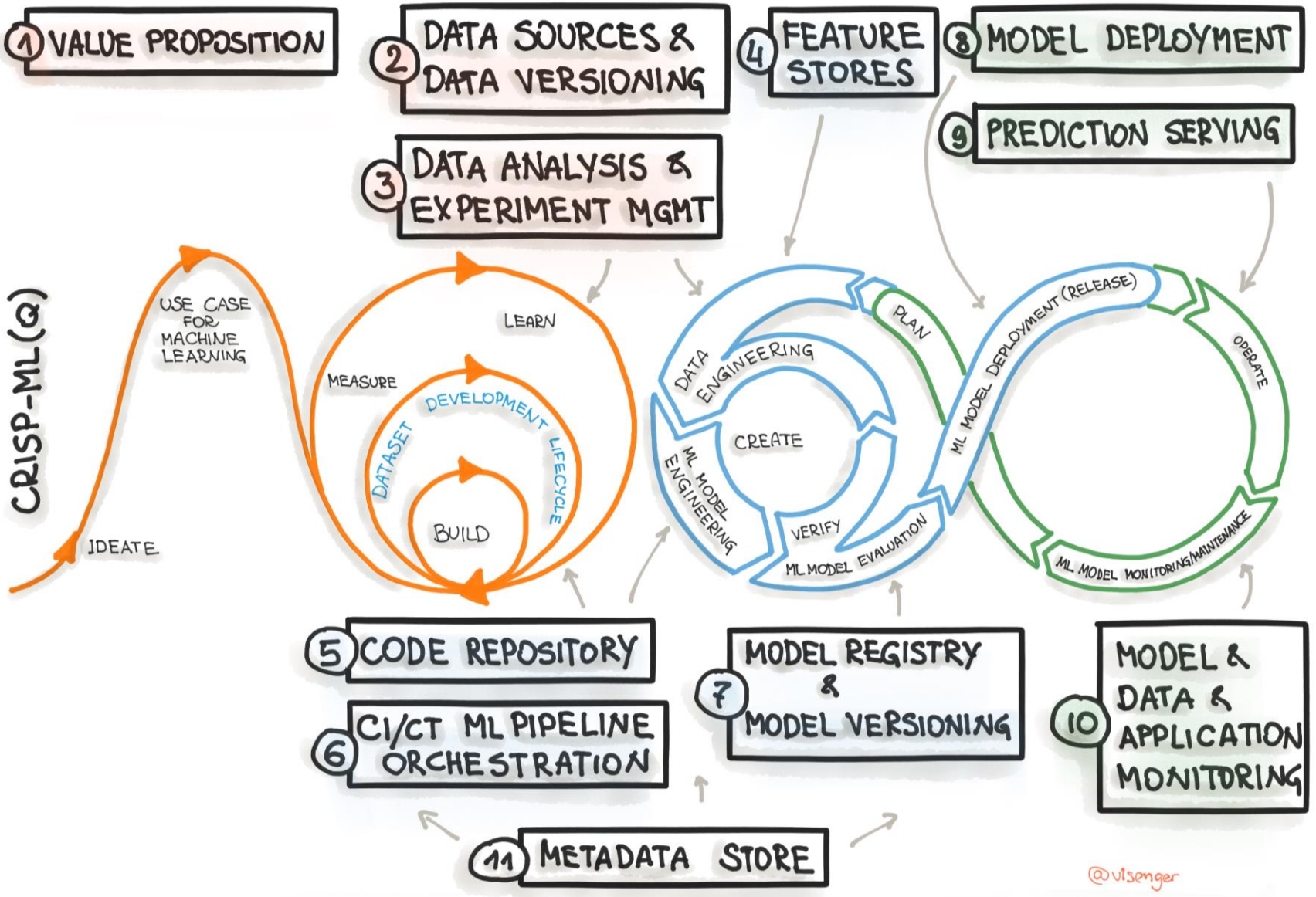
# Parts de MLOps

**Model Monitoring**
Models should be monitored for deviation and production bias

**Recycling of models**
Models can be retrained to improve performance or when there is new data

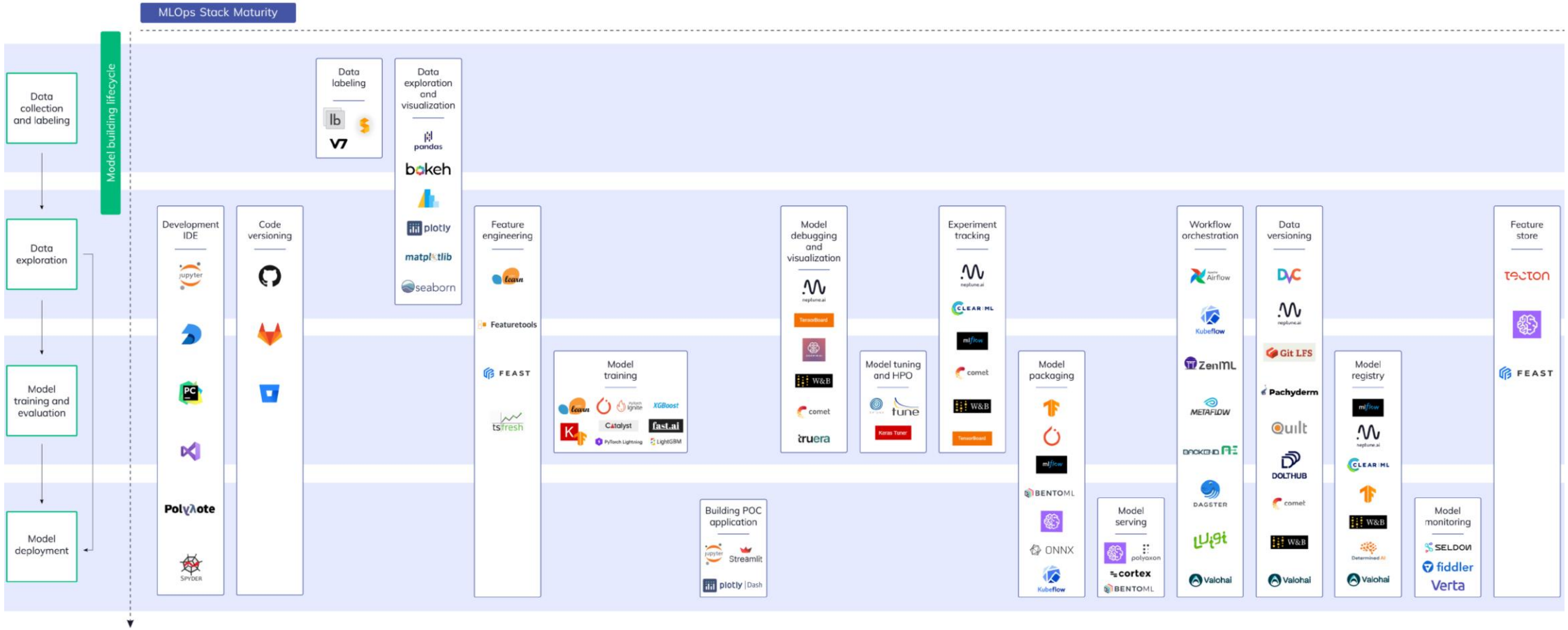
**CI/CD**
This ensures that code is frequently merged with automated process and tests.

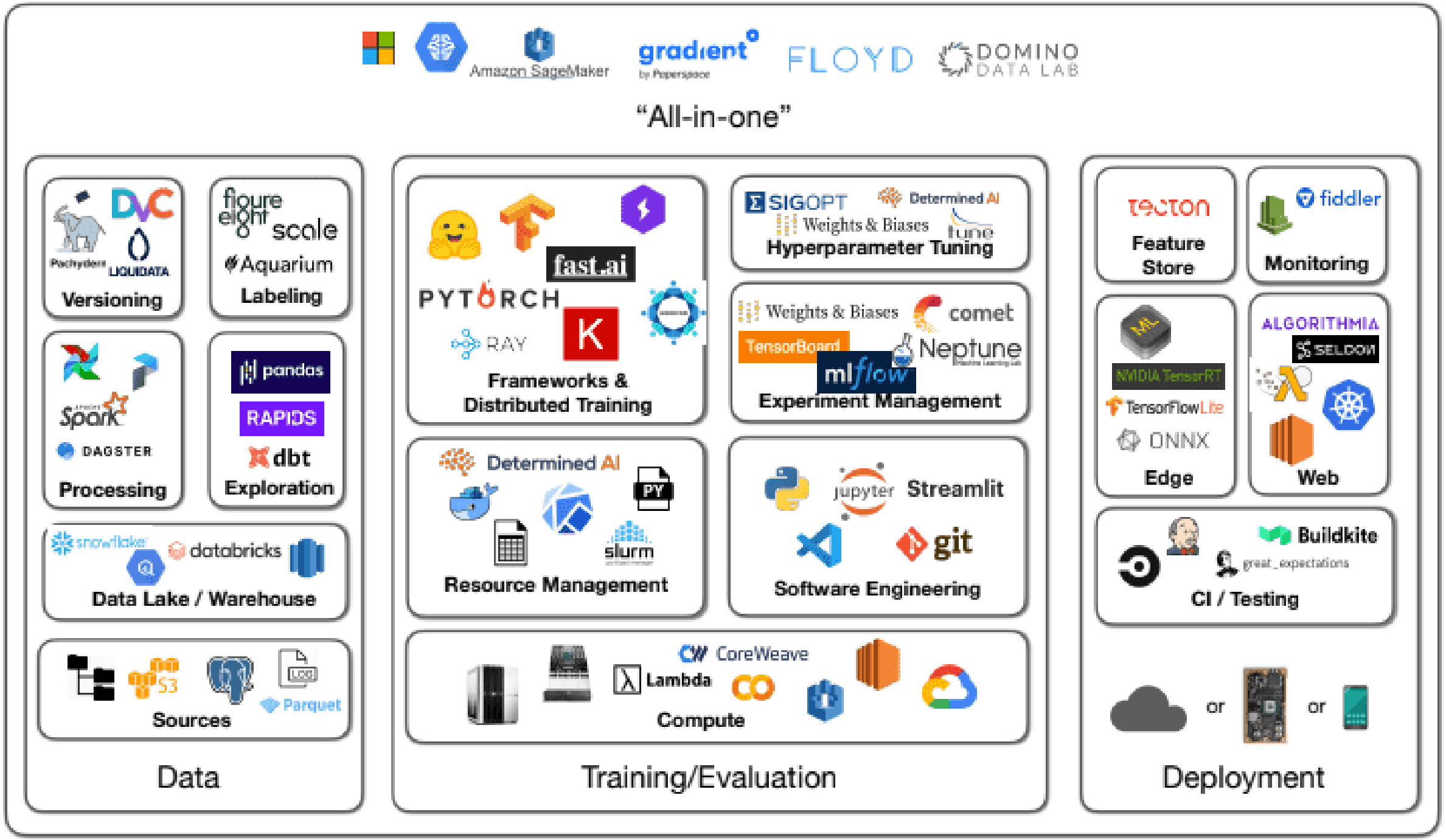
# Components of MLOps



# MLOps Tools

https://neptune.ai/blog/machine-learning-model-management#&gid=1&pid=1




Amazon SageMaker
gradient
by Paperspace
FLOYD
DOMINO DATA LAB
"All-in-one"
Versioning
Labeling
Processing
Exploration
Data Lake / Warehouse
Sources
Data
Frameworks & Distributed Training
Hyperparameter Tuning
Experiment Management
Resource Management
Software Engineering
Compute
Training/Evaluation
Feature Store
Monitoring
Edge
Web
CI / Testing
Deployment

# MLOps Stages

# MLOps stages

**Stage 1:** Model and data **Version Control**

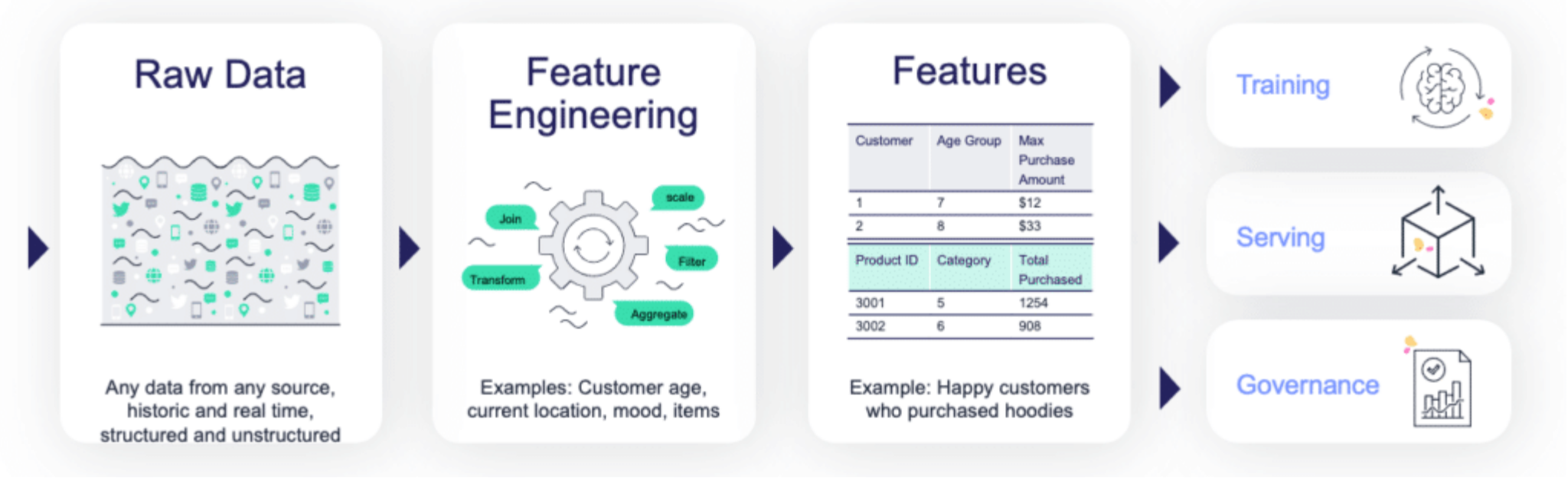
**Stage 2: AutoML** + Model and Data Version Control

**Stage 3:** AutoML + Model and Data Version Control + **Model Serving**

**Stage 4:** AutoM + Model and Data Version Control + Model Serving **+ Monitoring, Governance and Retraining**

# Stage 1: Data Collection and Preparation

**There is no ML without data**. ML teams need access to historical and/or online data from multiple sources. They must catalog and organize this data. Raw data cannot be used, they need to process this data.




Data Bootcamp
BEST DATA TRAINING

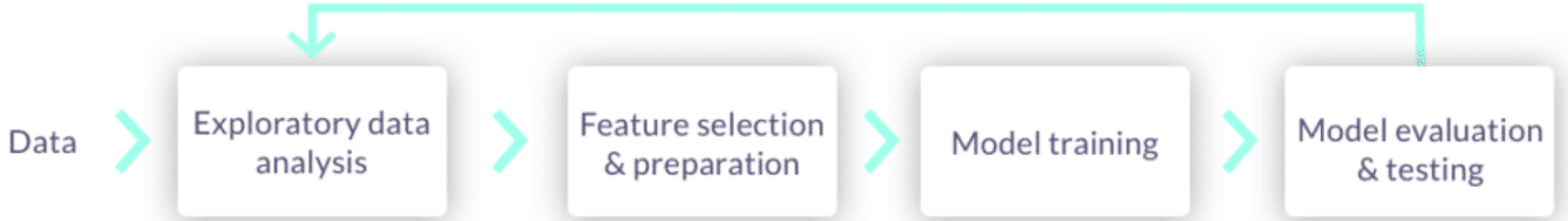
# Data collection and preparation with MLOps

MLOps solutions must incorporate a **Feature Storage** that:

1. Define data collection and transformations **only once** for batch and streaming scenarios
2. Process functions automatically **without manual intervention**
3. Serve functions from a **shared catalog** for training, service and government applications

# MLOps Stage 2: Automated Development

Model development generally follows the **same process**. Much of it can be **automated** thanks to **AutoML and MLOps**

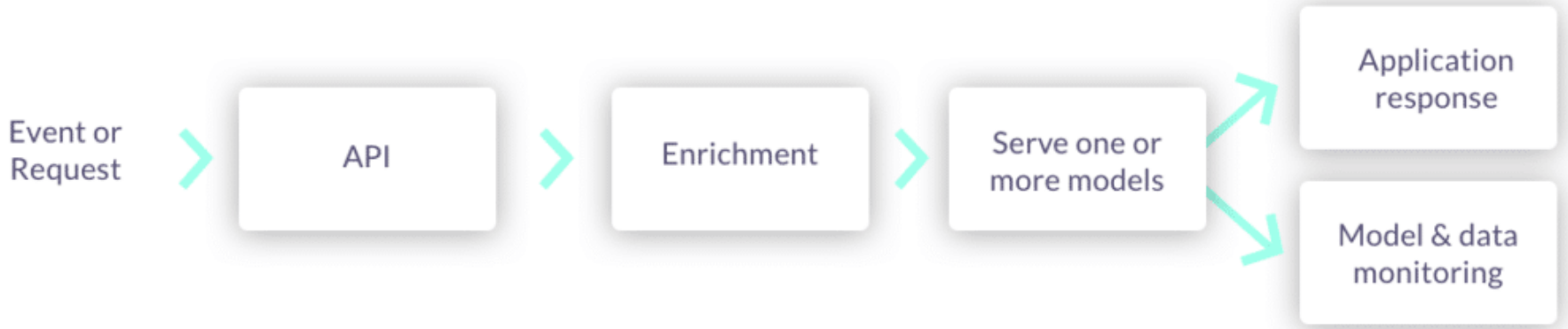



All runs, along with their data, metadata, code, and results, must be **versioned and logged**

# MLOps Stage 3: Create ML Services

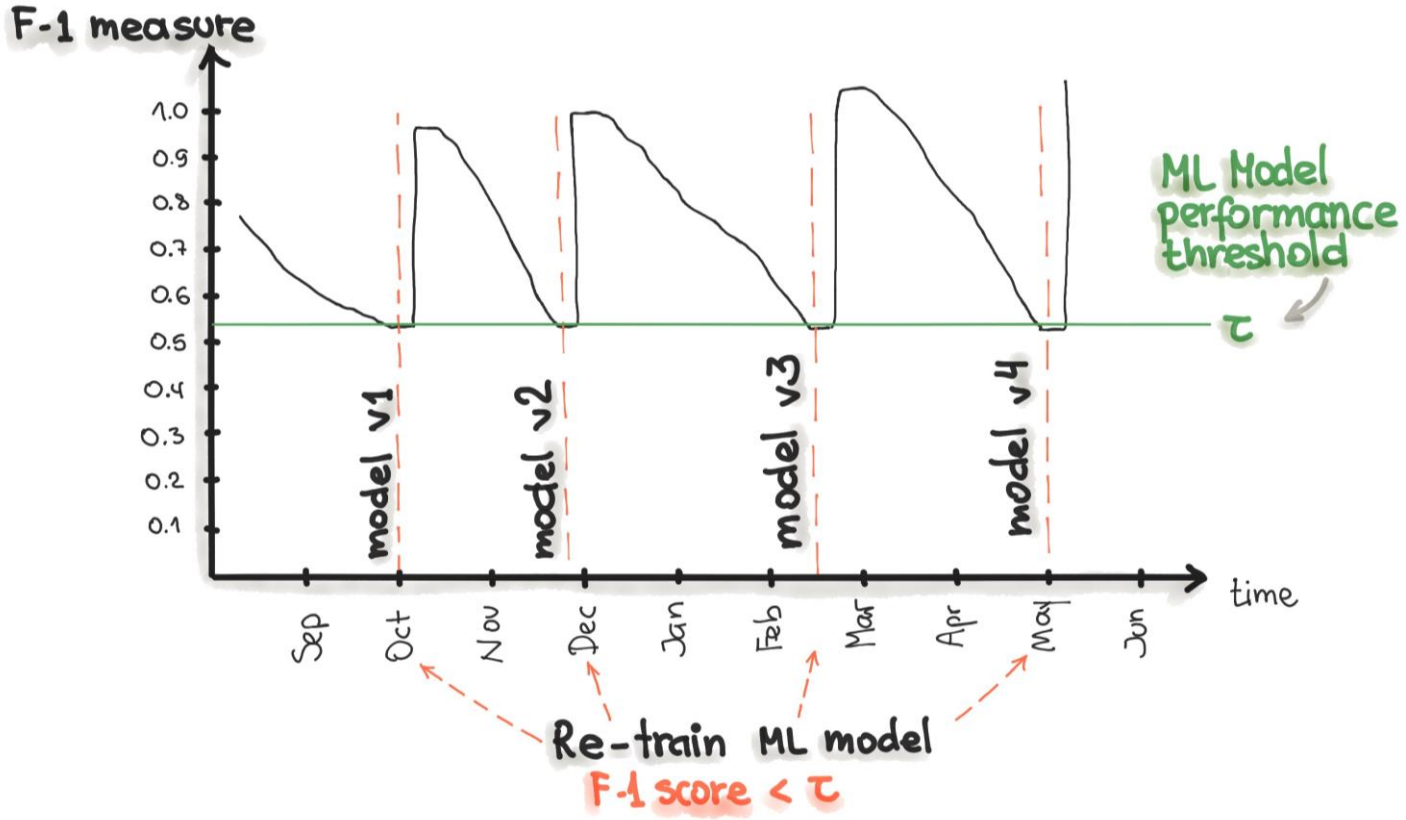
Once a model has been created, it must **be integrated** with the **business application** or **front-end** services. They must be implemented without interrupting service. Production pipelines implement:

- Real-time data collection, data validation, and feature engineering
- **API services** or application integration
- Data and **model monitoring** services
- **Resource monitoring** and alert services
- Telemetry and event logging services



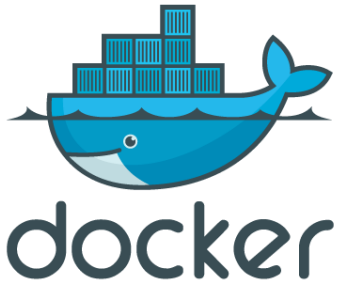
# MLOps Stage 4: Monitoring, Governance and Retraining

**Model monitoring** is a core component of MLOps to **keep models up-to-date** and predicting with maximum accuracy. It guarantees the validity of the model in the long term.



# Installations

# Tools to use

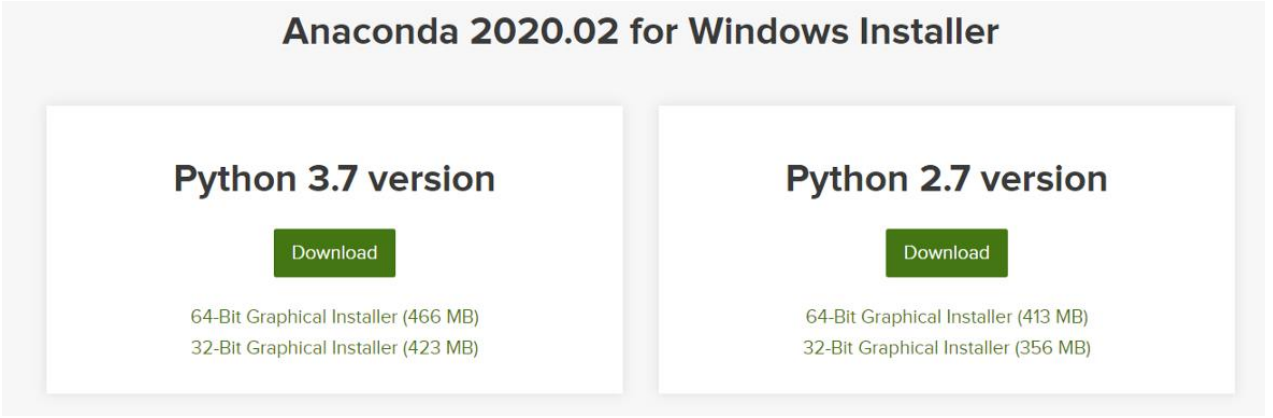




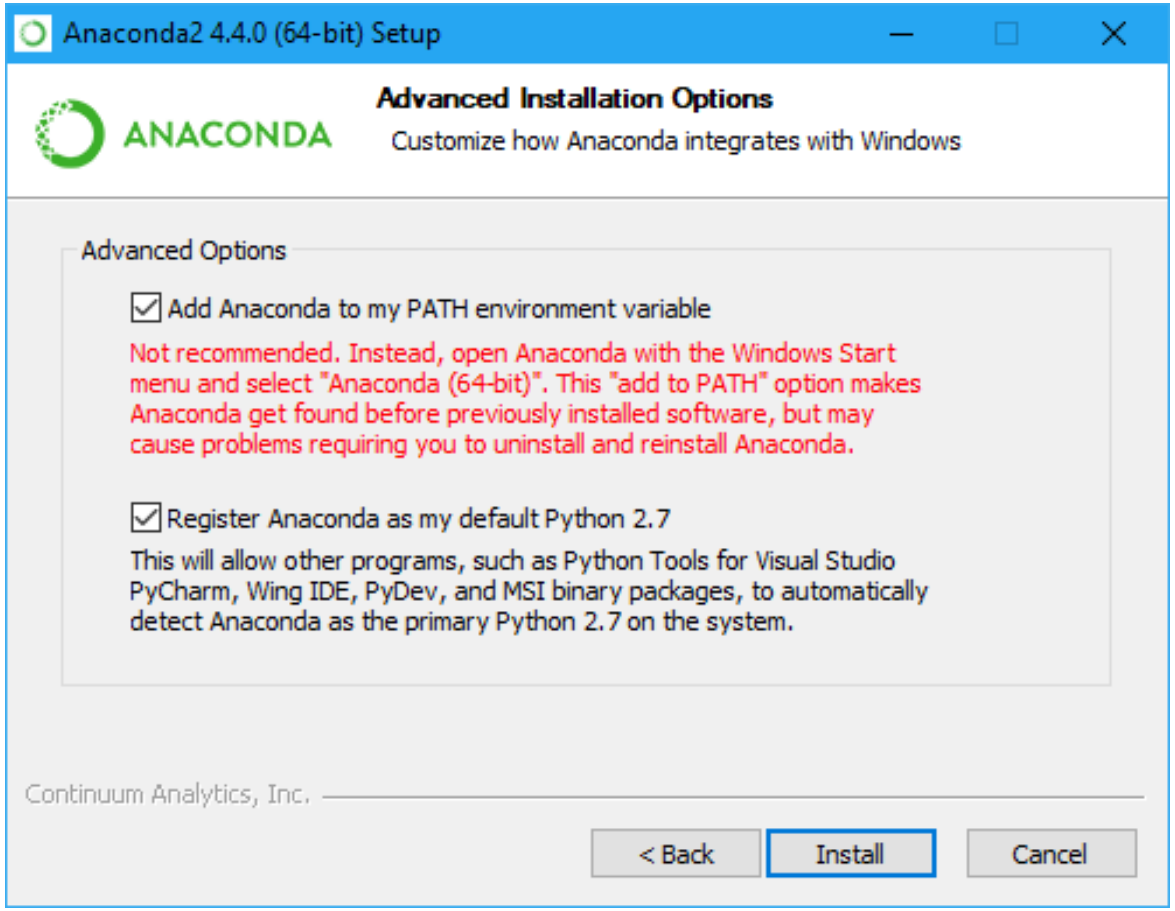
# Facility

If this is your first-time using Python, you must install **Anaconda Distribution** with **Python 3.7** or higher. Link:

https://www.anaconda.com/products/individual

# Environment variable



# Set the environment

**Step 1. Create a virtual environment**

Open the Anaconda Prompt from the start menu and run the following code: *conda create --name mlops python=3.7*

**Step 2 Activate the environment**

Execute the command: *conda activate mlops*

**Step 3. Install the necessary libraries**

From the Anaconda prompt write the following code, with the name of the library you want to install: *pip install [library—name]* Example with Pycaret: *pip install pycaret==2.3.5*

if you have requirements file: *pip install -r requirements.txt*

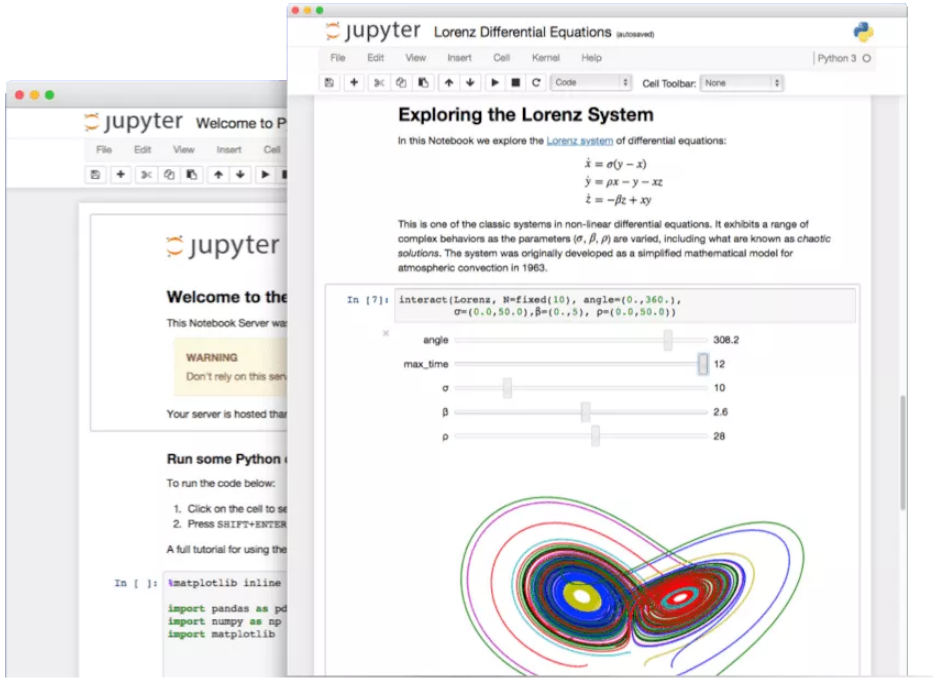
**Step 4. Jupyter Notebook**

From the correct environment (ml_pycaret) launch jupyter notebook with command *jupyter notebook*

# Jupyter Notebbok

**Jupyter Notebook** is a visual IDE for creating and sharing documents with code in different programing languages as:

Python, R, Scala, etc. It offers a **simple**, streamlined, document-centric experience.




## Jupyter Notebook: The Classic Notebook Interface

The Jupyter Notebook is the original web application for creating and sharing computational documents. simple, streamlined, document-centric experience.

Try it in your browser | Install the Notebook

### Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

### Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

### Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

### Big data integration

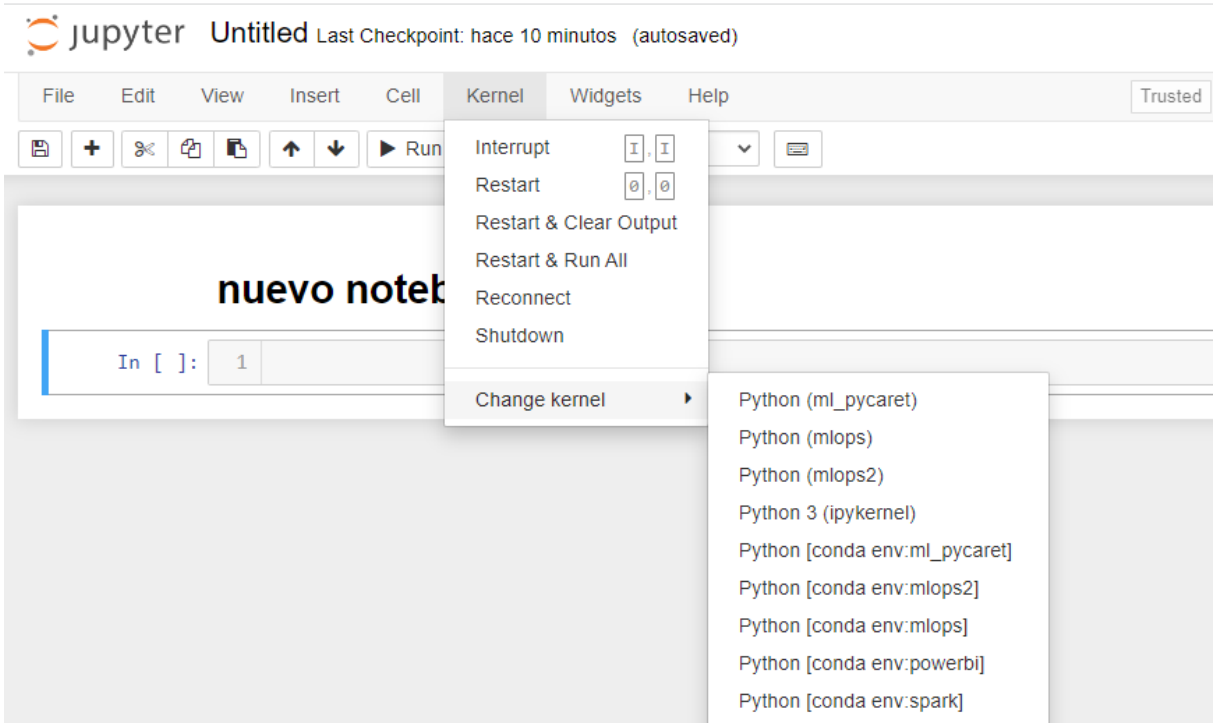
Leverage big data tools, such as Apache Spark, from Python, R, and Scala. Explore same data with pandas, scikit-learn, ggplot2, and TensorFlow.

# Jupyter Notebok Kernel

In some versions, to change enviroment in Jupyter Notebook we must install some aditional libreries. The commands are:

```
conda install -n python_env ipykernel
python -m ipykernel install --user --name mlops--display-name "Python (mlops)”
```

More information at: https://stackoverflow.com/questions/39604271/conda-environments-not-showing-up-in-jupyter-notebook



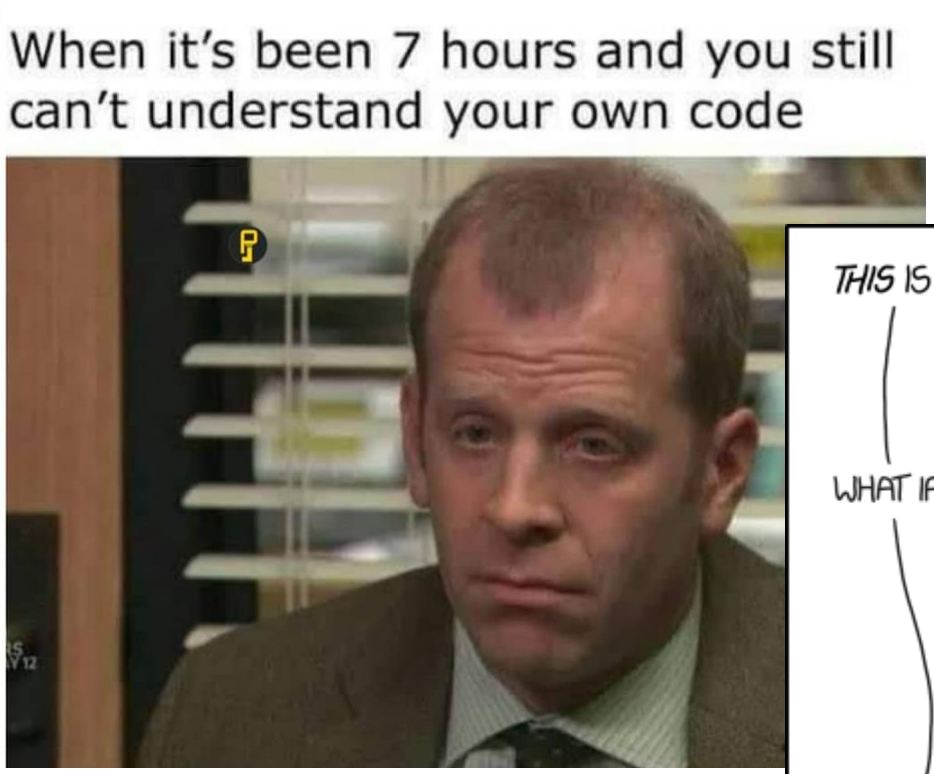
# Docker

In order to install Docker, we must follow those following steps:

1. Download **Docker Desktop** and install https://docs.docker.com/desktop/windows/install/
2. Install **Windows Subsystem for Linux** (Step 4) https://docs.microsoft.com/en-us/windows/wsl/install-manual
3. Step 5 from Powershell
4. Install **Ubuntu**

# Structuring ML projects

# The importance of organizing the project

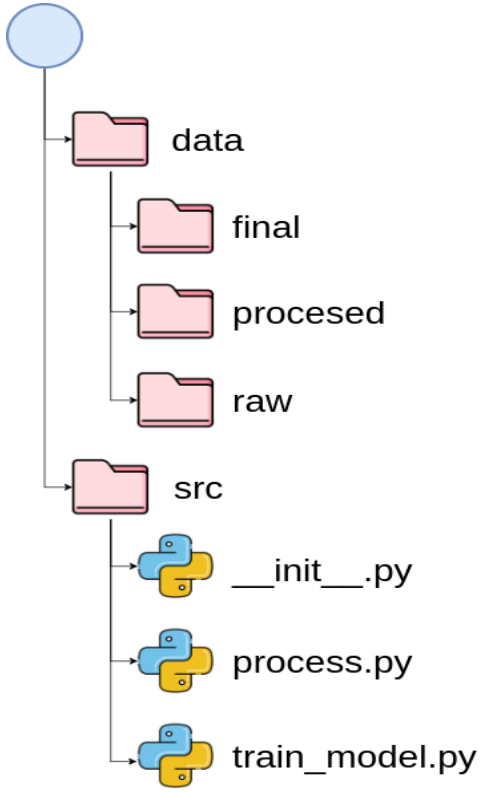


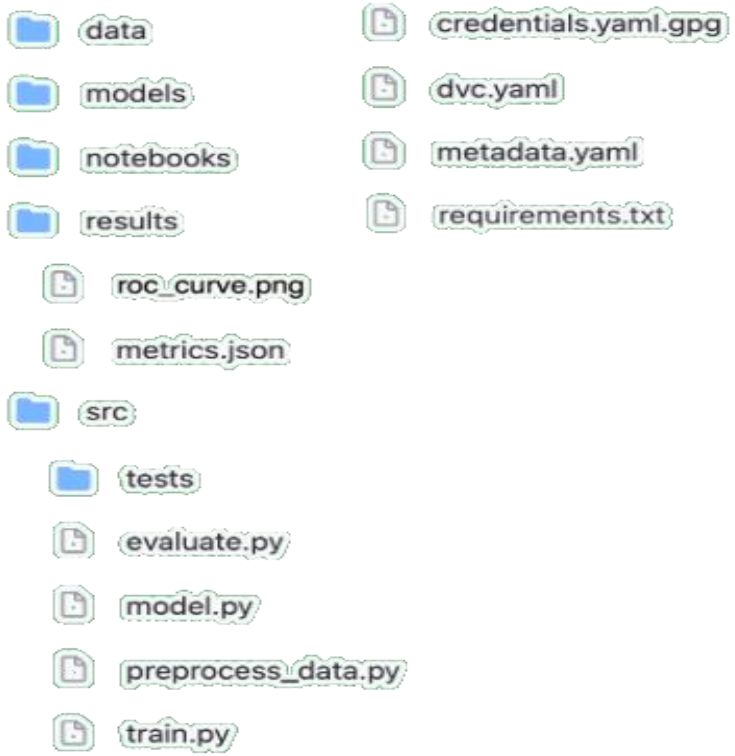


Data Bootcamp
BEST DATA TRAINING

# Structure ML projects

It is important to **structure** the project according to a **standard**. But what kind of standard should you follow?




```
.
├── LICENSE
├── README.md
├── data
│   └── README.md
├── metadata.yaml
├── models
│   └── README.md
├── notebooks
│   └── README.md
├── requirements.txt
├── results
│   └── README.md
└── src
    ├── scripts
    │   └── README.md
    └── tests
        ├── README.md
        └── test_australia_weather_predic

7 directories, 11 files
```

# Cookiecutter

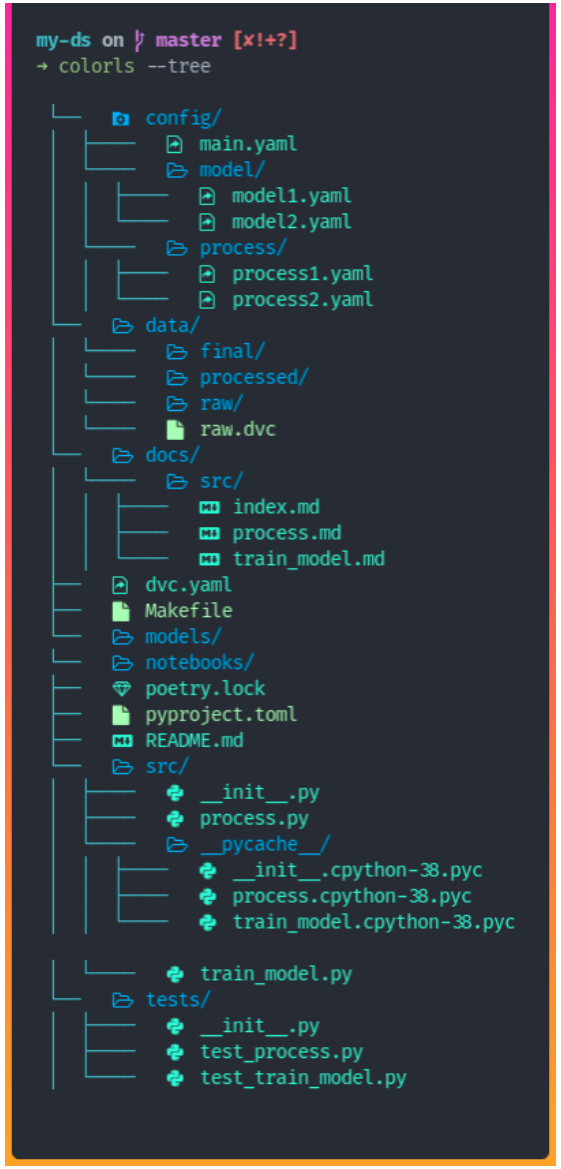
**Cookiecutter** is a tool for creating **projects folder structure** automatically **using templates**. You can create static file and folder structures based on input information.

*pip install cookiecutter*

*cookiecutter https://github.com/khuyentran1401/data-science-template*

# ML Tools

- **Poetry:** Dependency Management
- **Hydra:** To manage configuration files
- **Pre-commit plugins**: Automate code review and formatting
- **DVC**: Data Version Control
- **pdoc**: automatically create documentation for your project



# Poetry

An **alternative** to installing libraries with **pip** is using Poetry. **Poetry** allows to:

- Separate main dependencies and sub dependencies into two separate files (vs requirements.txt)
- Create readable dependency files
- Remove all unused sub-dependencies when removing a library
- Avoid installing new libraries in conflict with existing libraries
- Package the project with few lines of code

All the dependencies of the project are specified in **pyproject.toml.**

| Generate project | Install dependencies | To add a new PyPI library | To delete a library |
|---|---|---|---|
| poetry new <proyect-name> | poetry install | poetry add <library-name> | poetry remove <library-name> |

# Poetry

# Makefile

**Makefile** creates short and readable commands for configuration tasks. You can use Makefile to **automate tasks** such as setting up the environment.

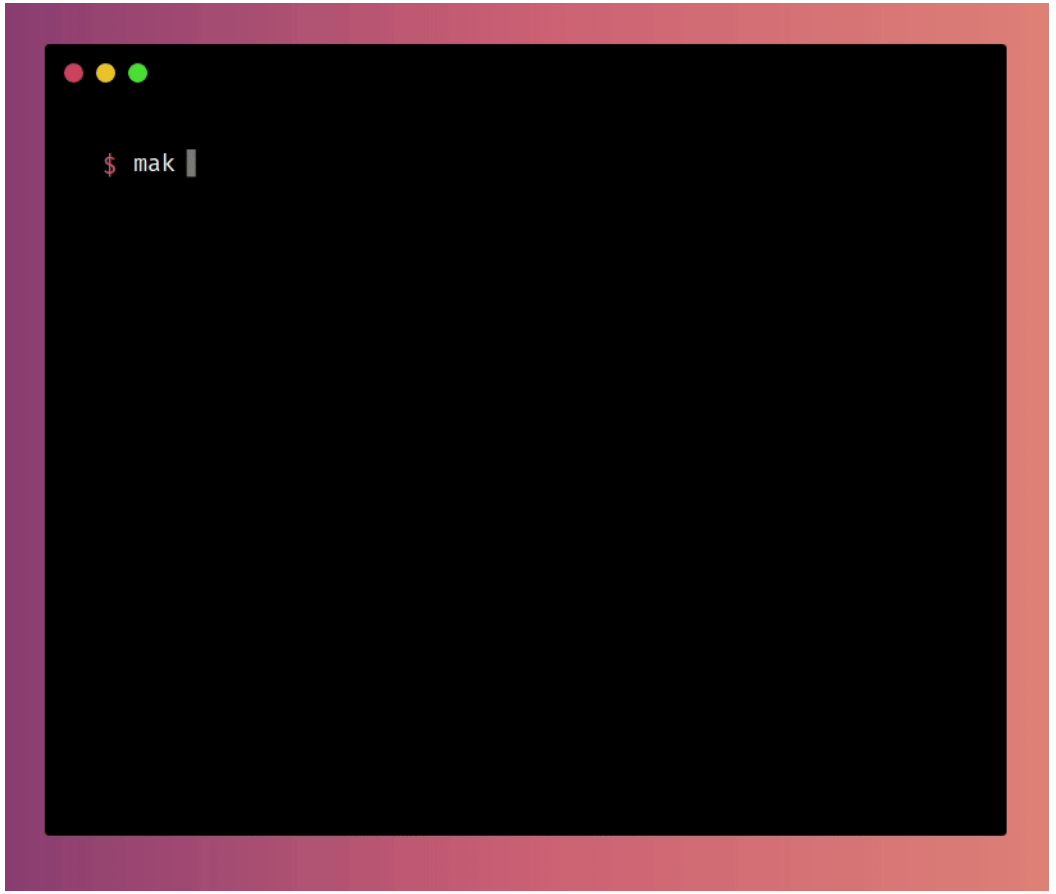
1

```
install:
        @echo "Installing..."
        poetry install
        poetry run pre-commit install

activate:
        @echo "Activating virtual environment"
        poetry shell

initialize_git:
  @echo "Initialize git"
        git init

setup: initialize_git install
```

2

```
make activate
make setup
```

3



# Hard-coding

In data science is common to **execute different configurations and models**, so configuration **should not be hardcoded**. For example, if we want to modify the input variables of model, it will take time to change them.

```
columns = ['iid', 'id', 'idg', 'wave', 'career']
df.drop(columns, axis=1, inplace=True)
```

Wouldn't it be better to set the columns in **a config file**?

```
variables:
  drop_features: ['iid','id','idg','wave','position','positin1', 'pid',  'field', 'from', 'career'

  # categorical variables to transform to numerical variables
  numerical_vars_from_numerical: ['income','mn_sat', 'tuition']

  # categorical variables to encode
  categorical_vars: ['undergra', 'zipcode']
  categorical_label_extraction: ['zipcode']
  categorical_onehot: ['undergra']
```

# Configuration file

A **configuration file** contains **parameters** that define the configuration of the program. It is good practice to avoid hard coding in Python scripts. **YAML** is a common language for a configuration file.

```python
# get current path
current_path = utils.get_original_cwd() + "/"

# read training data
data = pd.read_csv(current_path + config.dataset.data,  encoding=config.dataset.encoding)

# divide train and test
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(config.target.target, axis=1),
    data[config.target.target],
    test_size=0.1,
    random_state=0)
```

**Config.yaml**

```yaml
# data
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

pipeline:
  pipeline01: decisiontree

target: match
```

config.yaml hosted with ❤ by GitHub

# Hydra

There are some tools to manage a configuration file such as PyYaml or **Hydra**. Why Hydra?

- Change parameters in the **terminal**
- Switch between **setting groups**
- Automatically **record** results

```yaml
# data
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

pipeline:
  pipeline01: decisiontree

target: match
```

```python
import hydra
from hydra import utils
import pandas as pd

@hydra.main(config_path='preprocessing.yaml')
def run_training(config):
    """Train the model."""

    # Get current path
    current_path = utils.get_original_cwd() + "/"

    # read training data
    data = pd.read_csv(current_path + config.dataset.data,  encoding=config.dataset.encoding)

    # divide train and test
    X_train, X_test, y_train, y_test = train_test_split(
        data.drop(config.target, axis=1),
        data[config.target],
        test_size=0.1,
        random_state=0)

if __name__=='__main__':
    run_training()
```

1

2

# Hydra uses

1. **Modify the parameters from the terminal** without the need to modify the config file
2. Switch between different **configuration groups**
3. **Logging**

```yaml
dataset:
  data: data/raw.csv
  encoding: iso-8859-1

model: decisiontree
```

model.yaml hosted with ❤ by GitHub

```
python file.py model=logisticregression
```

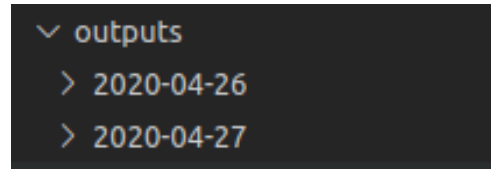
Run a logistic regression

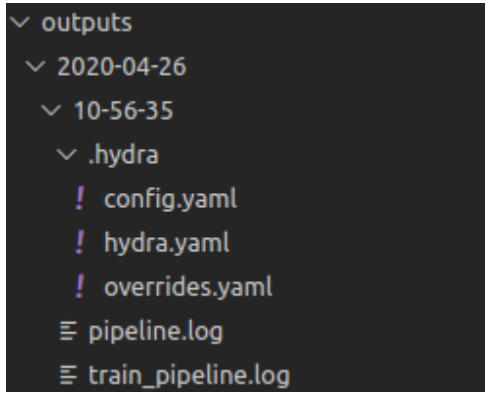
```
.
├── configs
│   └── model
│       ├── decisiontree.yaml
│       └── logistic.yaml
├── data
│   └── raw.csv
```

```yaml
hyperparamters:
  penalty: l1
  dual: False
  C: 1
```

logistic.yaml hosted with ❤

```
python file.py model=logistic
```



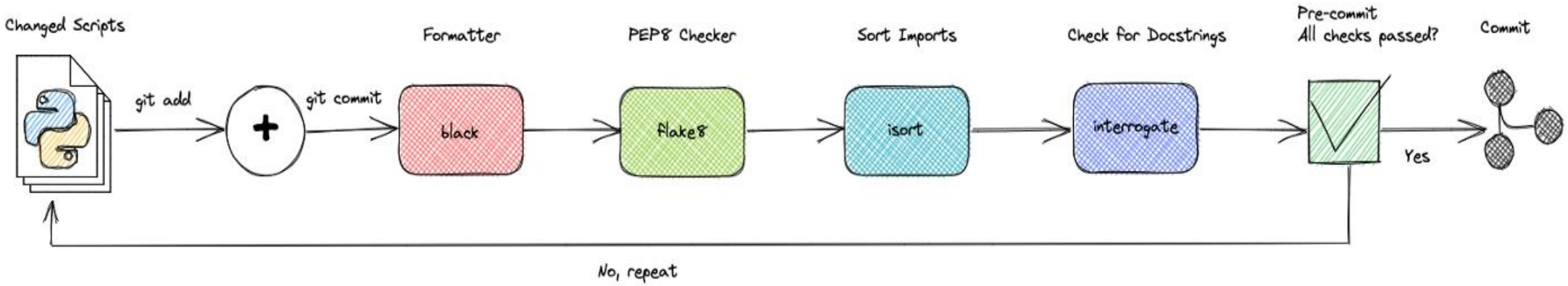



# Check code before commit

When committing Python code to Git, you must ensure that your code:

- It is correct
- It is organized
- Conforms to PEP 8 style guide
- Includes documentation (docstrings)

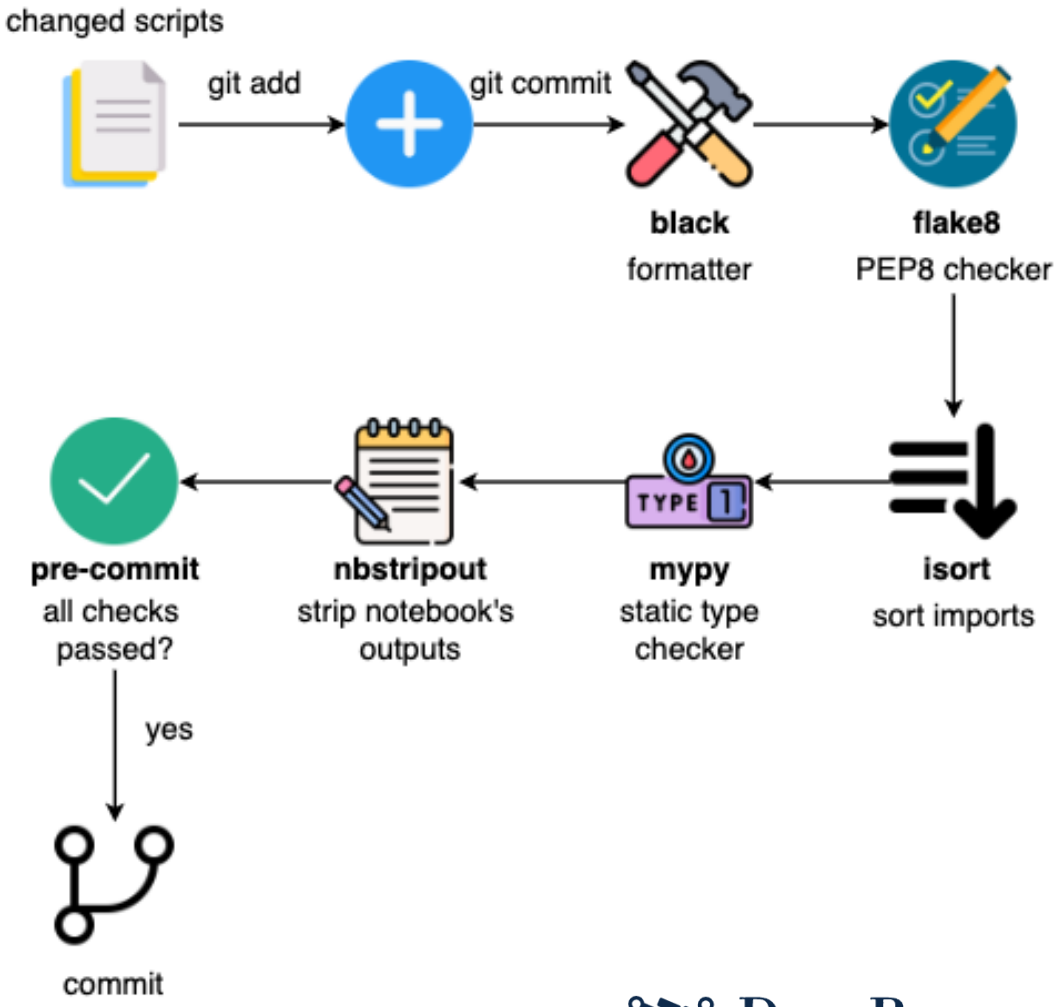
Different **plugins** can be added to **pre-commit** for automagical code **review.** Those plugins will review the code and will **correct it.**



# Commit plugins

In this example we will use the following plugins. Those plugins are specified in .pre-commit-config.yaml.

- **Black:** Format Python code
- **Flake8**: Check the style and quality of Python code
- **Isort:** Sort alphabetically imported libraries and separate them into types
- **Interrogate:** checks the code for missing docstrings



# Commit plugins in action

## black

```
1  def very_long_function(long_variable_name, long_variable_name2, long_variable_name3, long_variable_
2      pass
```

```
def very_long_function(
    long_variable_name,
    long_variable_name2,
    long_variable_name3,
    long_variable_name4,
    long_variable_name5,
):
    pass
```

## flake8

```
def very_long_function_name(var1, var2, var3,
var4, var5):
    print(var1, var2, var3, var4, var5)

very_long_function_name(1, 2, 3, 4, 5)
```

```
flake8_example.py:2:1: E128 continuation line under-indented for visual indent
flake8_example.py:5:1: E305 expected 2 blank lines after class or function definition, found 1
flake8_example.py:5:39: W292 no newline at end of file
```

```
def very_long_function_name(var1, var2, var3, var4, var5):
    print(var1, var2, var3, var4, var5)


very_long_function_name(1, 2, 3, 4, 5)
```

## isort

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from flake8_example import very_long_function_name
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, OrderedLogisticRegression, \
    LinearRegression, LogisticRegressionCV, LinearRegressionCV
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from flake8_example import very_long_function_name
from sklearn.linear_model import (
    LinearRegression,
    LinearRegressionCV,
    LogisticRegression,
    LogisticRegressionCV,
    OrderedLogisticRegression,
)
from sklearn.model_selection import train_test_split
```

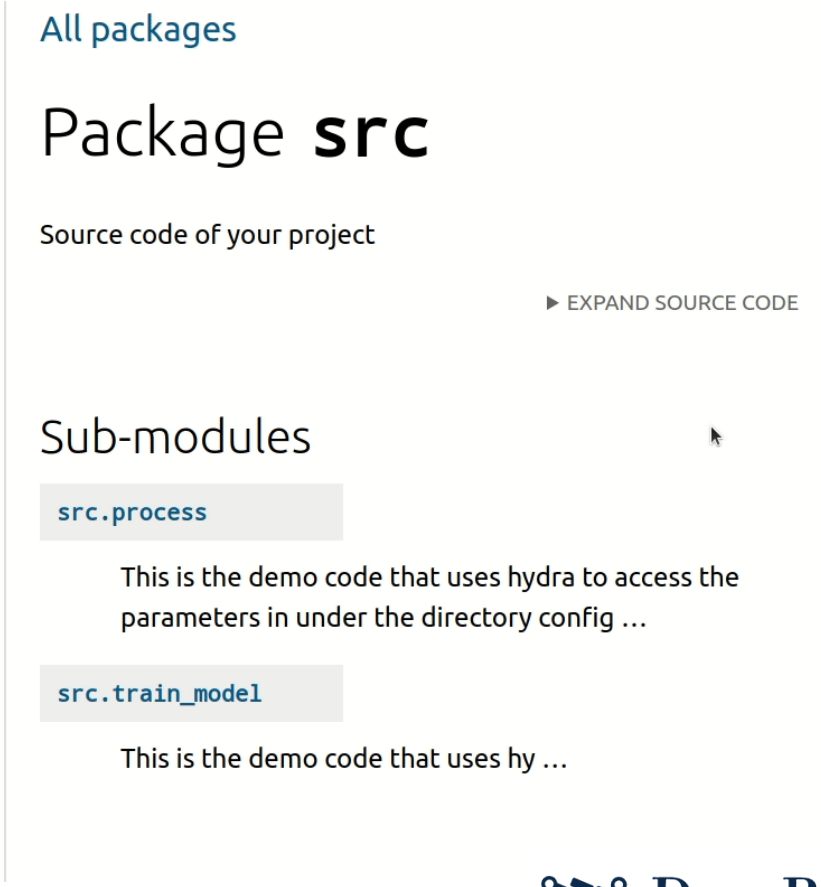
# Add documentation

As a data scientist, we will be **collaborating** a lot with other team members. Therefore, it is important to create a **good documentation** for the project. To create API documentation based on the **dockstrings** we can use Makefile.

```
make docs_view
```

```
Save the output to docs...
pdoc src --http localhost:8080
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
```

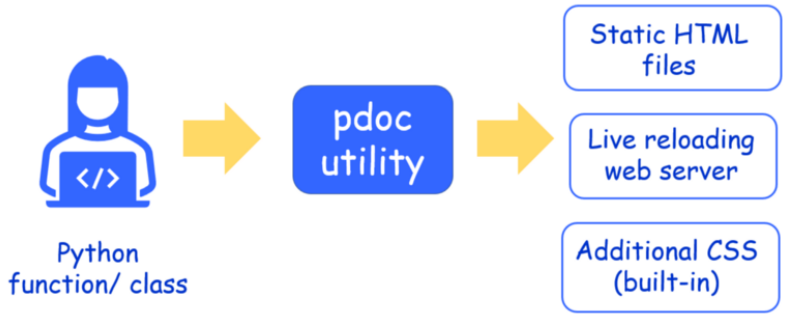
```
make docs_save
```




Source of : https://github.com/khuyentran1401/data-science-template/

# Pdoc3 to create Python documentation

It would be great to **generate beautiful web documentation** directly from functions docstrings without writing a single line of HTML/CSS code.




dockstrings

```
def addition(num1, num2=0):
    """
    Adds two numbers

    Args:

    num1: The first number

    num2: The second number, default 0

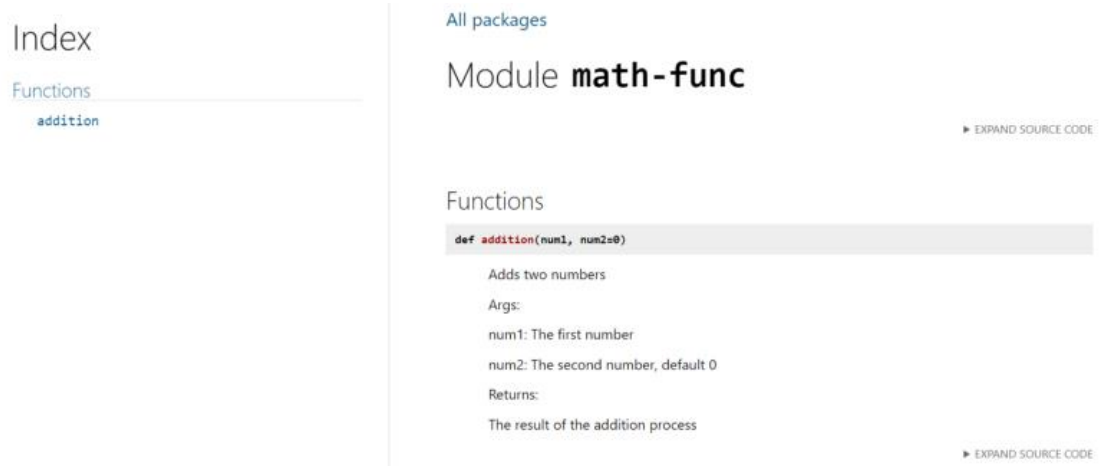
    Returns:

    The result of the addition process
    """

    return (num1+num2)
```

```
pip install pdoc3
pdoc --http localhost:8080 math-func.py
```

```
Starting pdoc server on localhost:8080
pdoc server ready at http://localhost:8080
```



# Markdown: the key to usability

The great thing about pdoc is that it allows seamless integration of **Markdown** text within the docstring.

```
"""

...

Handles exception by a check,

    ```python
    if num2 != 0:
        return (num1/num2)
    else:
        raise ValueError('The second argument cannot be zero')
    ```

 """
```

```
def divide(num1, num2=1)
```

Divides the first number by the second

***Args***:

`num1` : The first number

`num2` : The second number, default `1`

***Returns***:

The result of the **division** process

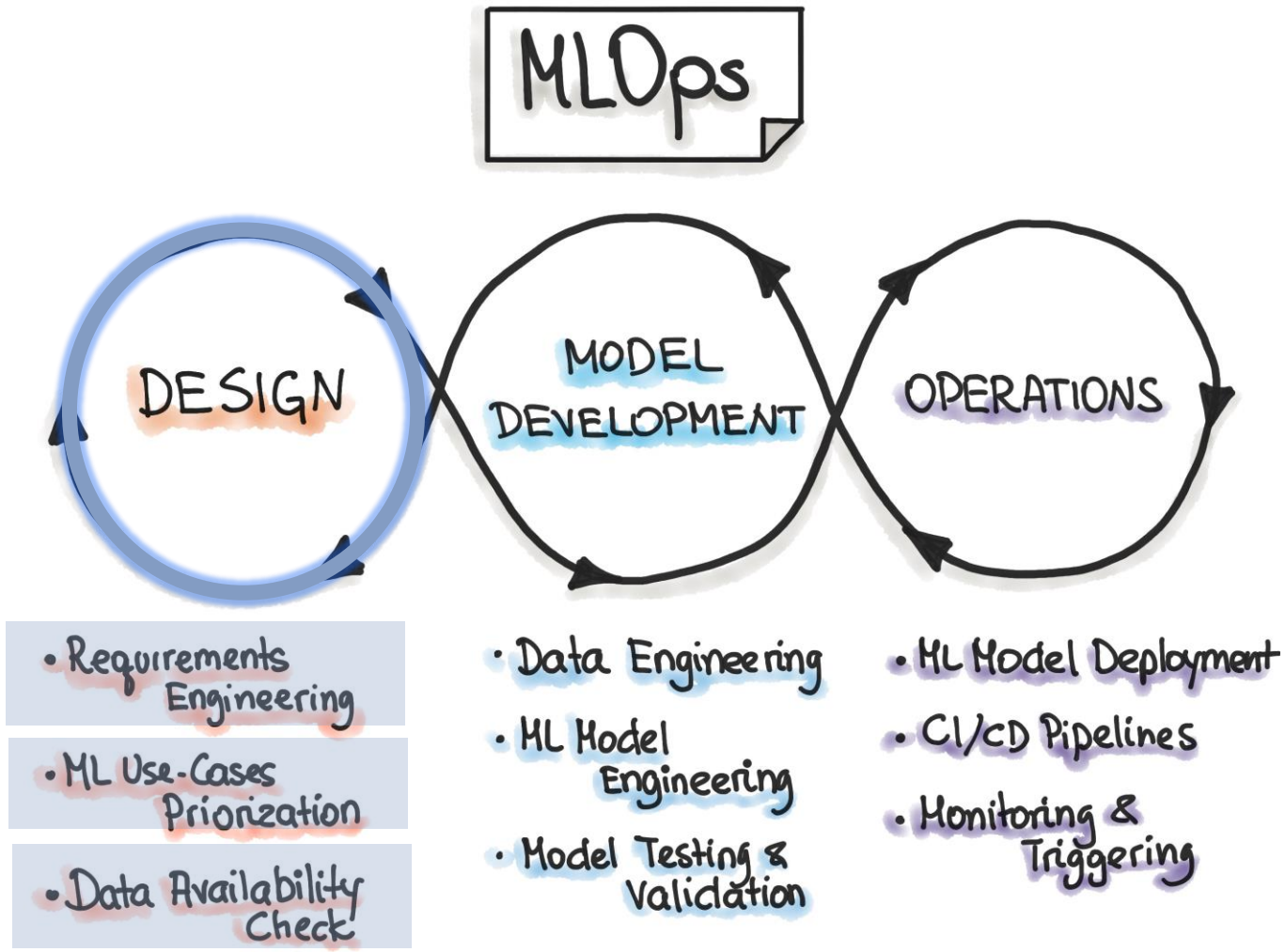
***Exception***:

Handles exception by a check,

```python
if num2 != 0:
    return (num1/num2)
else:
    raise ValueError('The second argument cannot be zero')
```

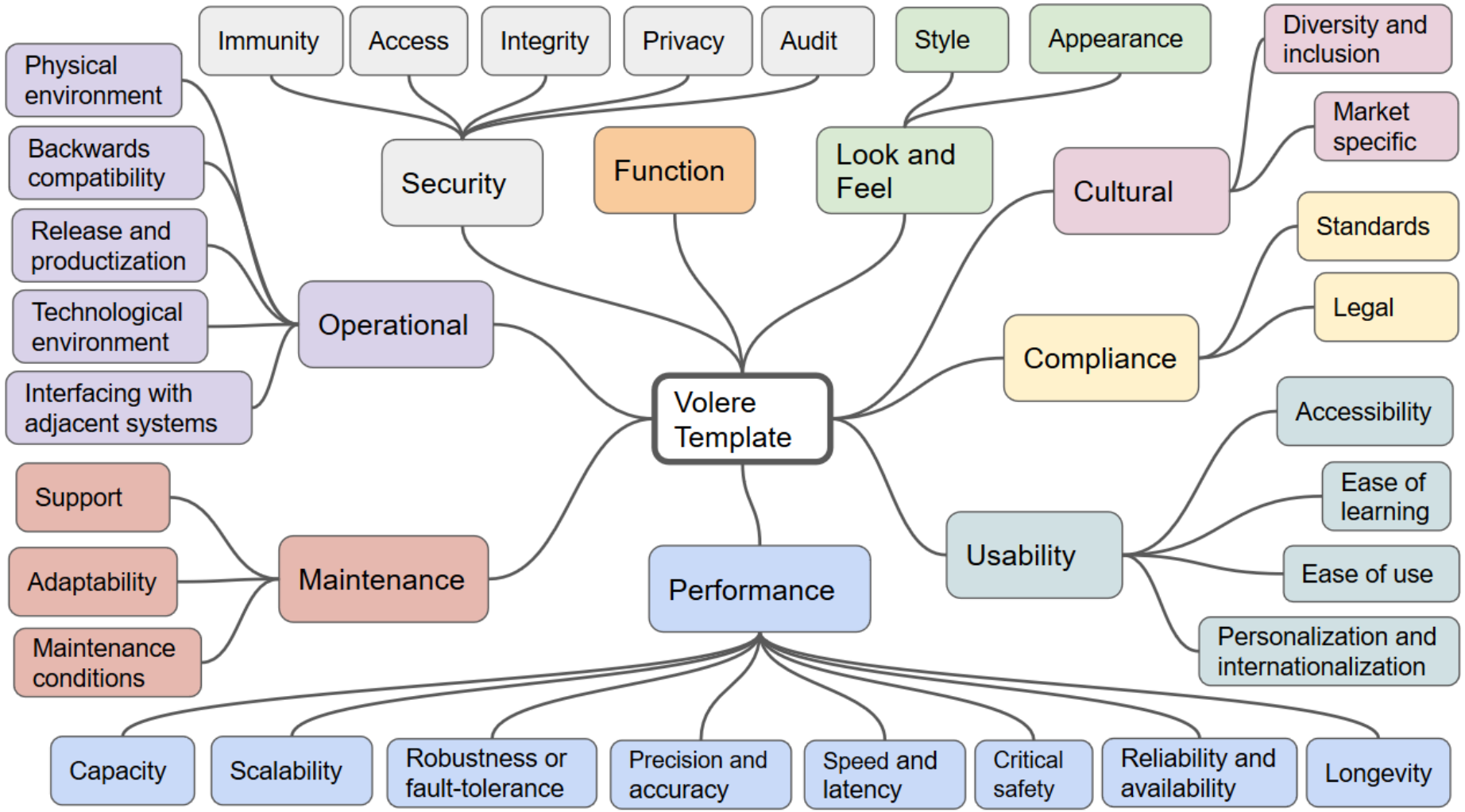
# ML product design

# MLOps stages



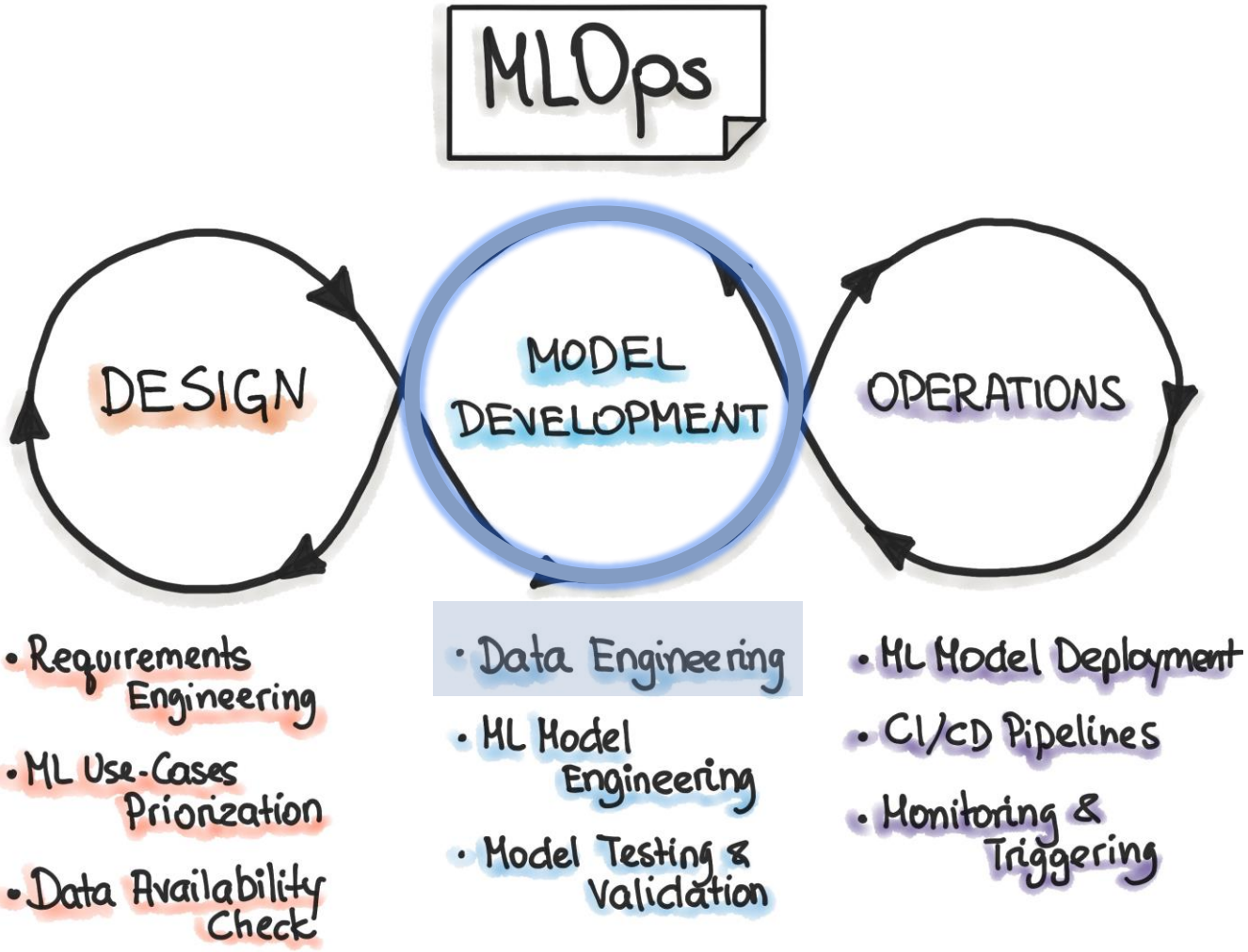
# Tools

Tool to complete the design phase of the model https://github.com/ttzt/catalog_of_requirements_for_ai_products



# Feature Store

# MLOps stages



# Reference tools

Feature Store is a data management layer for machine learning that allows you to share features and build more efficient **machine learning** pipelines.

| Platform | Open Source | Offline | Online | Real Time Ingestion | Feature Ingestion API | Write Amplification | Supported Platforms | Training API | Training Data |
|---|---|---|---|---|---|---|---|---|---|
| **Hopsworks** | AGPL-V3 | Hudi/Hive and pluggable | RonDB | Flink, Spark Streaming | (Py)Spark, Python, SQL, Flink | No | AWS, GCP, On-Prem | Spark | DataFrame (Spark or Pandas), files (.csv, .tfrecord, etc) |
| **Michelangelo** | No | Hive | Cassandra | Flink, Spark Streaming | Spark, DSL | None | Proprietary | Spark | DataFrame (Pandas) |
| **Zipline** | No | Hive | Unknown KV Store | Flink | DSL | None | Proprietary | Spark | Streamed to models? |
| **Twitter** | No | GCS | Manhatten, Cockroach | Unknown | Python, BigQuery | Yes. Ingestion Jobs | Proprietary | BigQuery | DataFrame (Pandas) |
| **Iguazio** | No | Parquet | V3IO, proprietary DB | Nuclio | Spark, Python, Nuclio | Unknown | AWS, Azure, GCP, on-prem | No details | DataFrame (Pandas) |
| **Databricks** | No | Delta Lake | Mysql or Aurora | None | Spark, SparkSQL | Unknown | Unknown | Spark | Spark Dataframes |

Reference
:
https://www.featurestore.org/

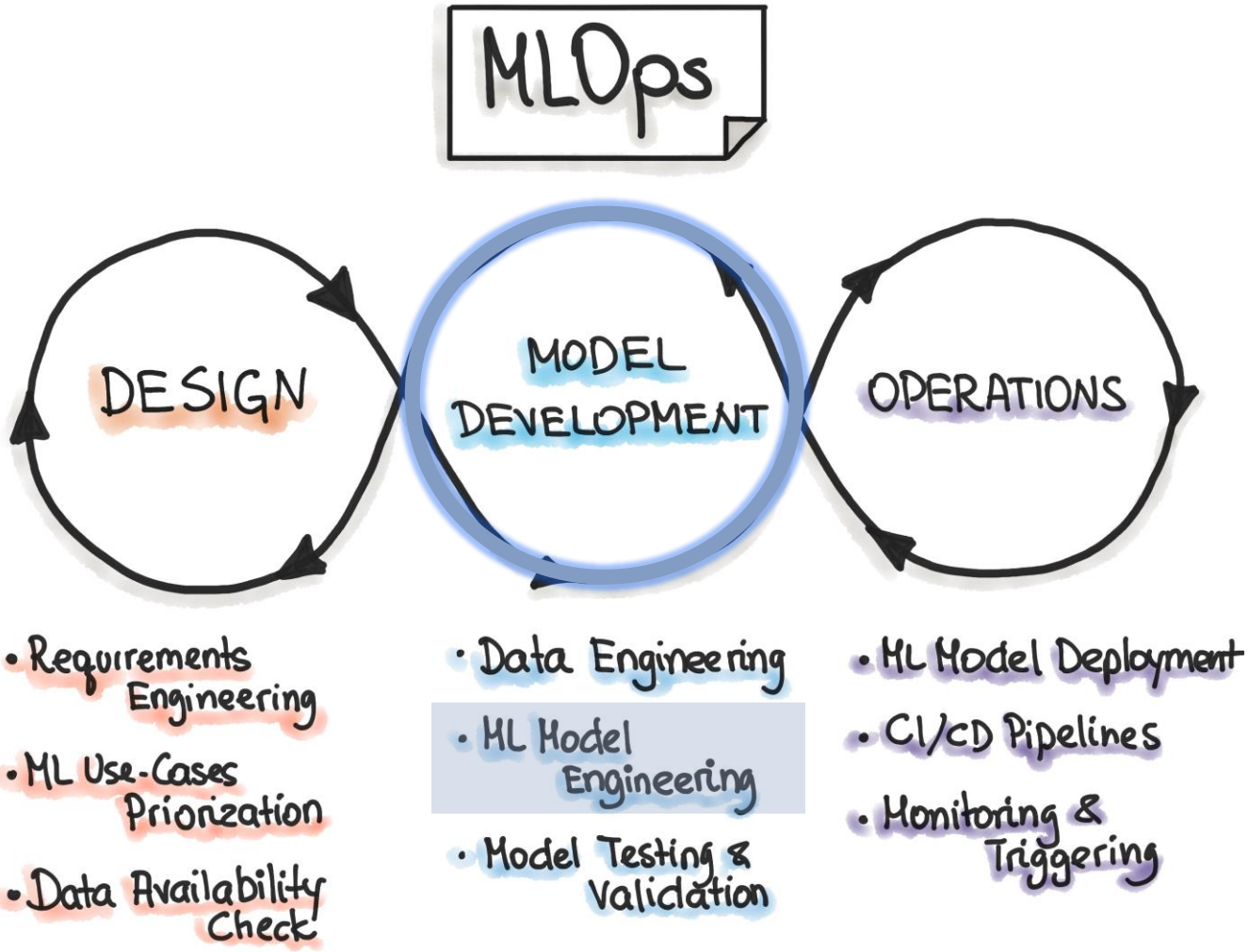
# DVC Studio

The **DVC Studio** interface allows us **to work with** data and also perform **experiments** from the web application:

- It helps us **manage data** and models,
- Allows you to run and track experiments
- Allows you to view and share results.
- It allows us to track our code, experiments, and data all the time.

# Automated model development

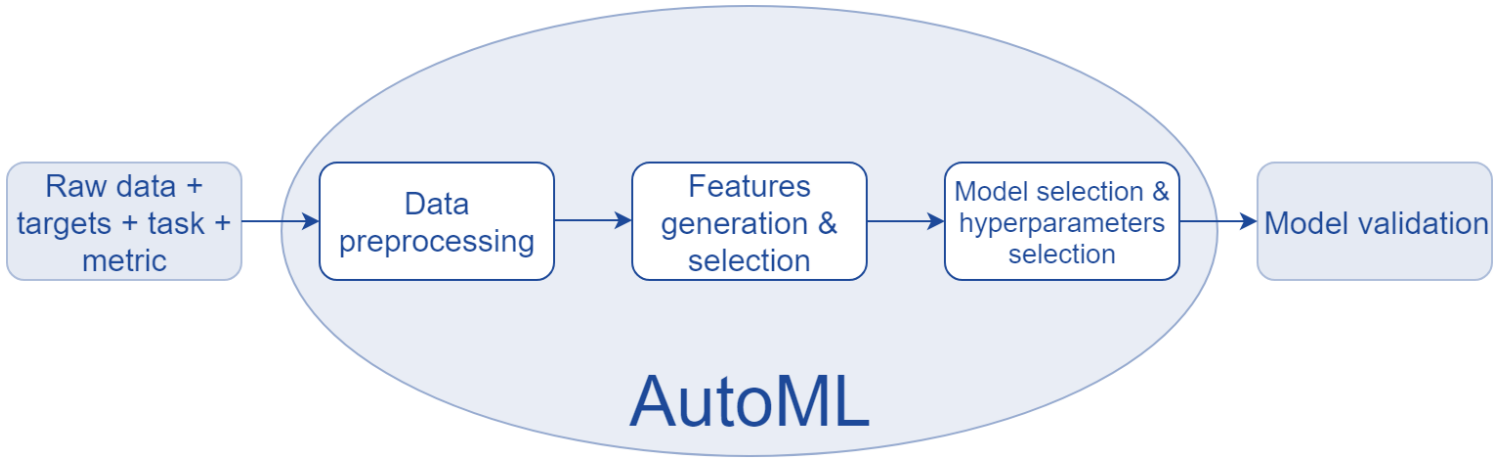
# MLOps stages



# AutoML

**AutoML** automates much of the model training process:

- AutoML helps to **Preprocess** the data
- AutoML generates new **variables** and selects the most significant ones
- AutoML trains and selects best **model**
- AutoML adjusts the **hyperparameters** of the chosen model
- AutoML makes model **evaluation** easy
- AutoML helps in model **deployment**



# Pycaret

**PyCaret** is an open source, low-code **machine learning** library. I has been developed in **Python** and reduce the time needed to create a model to minutes

Data Preparation

Model Training

Hyperparameter Tuning

Analysis & Interpretability

Model Selection

Experiment Logging

# Built-in Pycaret modules





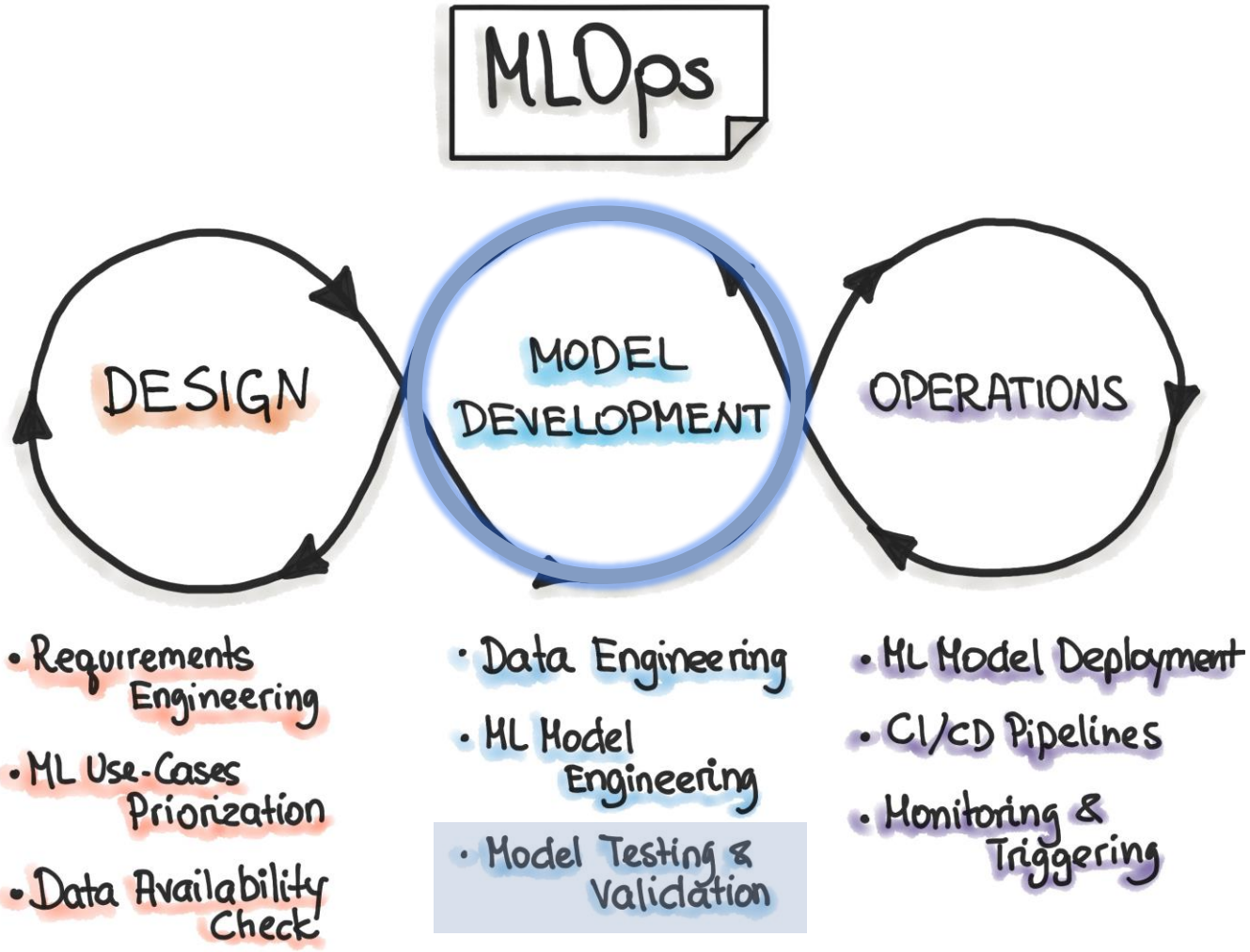







# Model interpretability

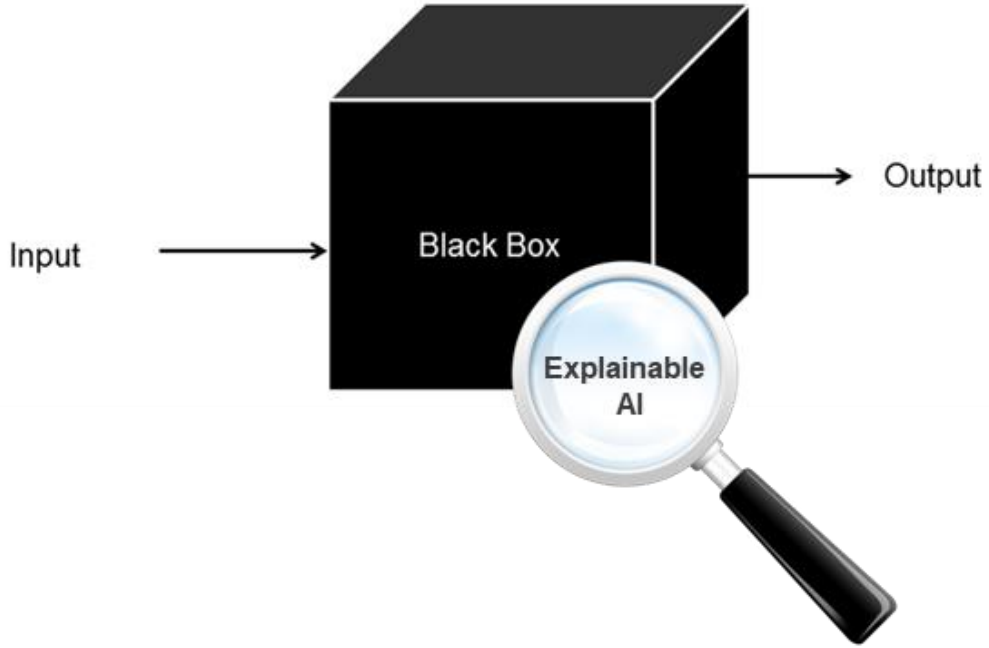
# MLOps stages



# Black box

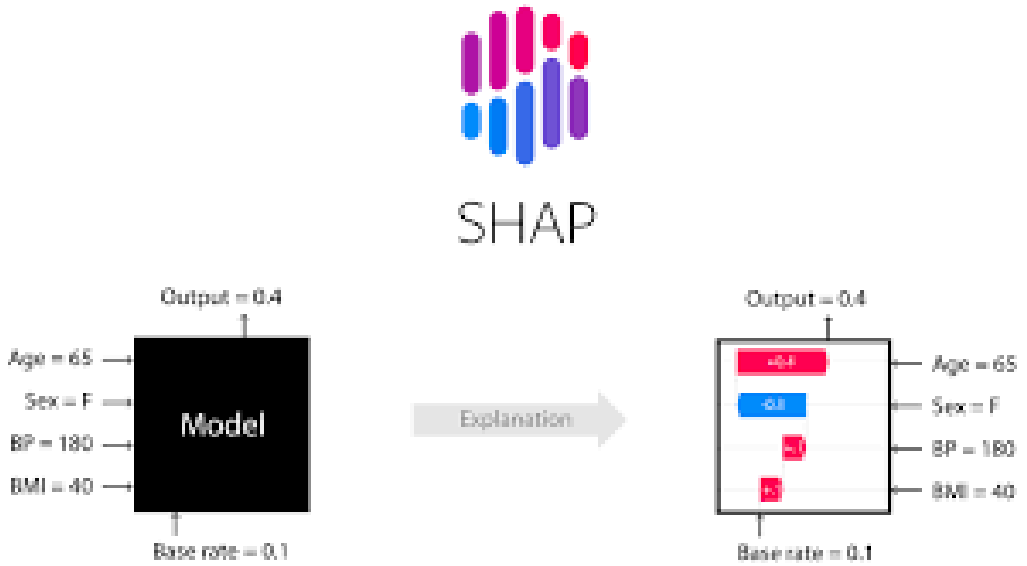
ML models are commonly known as "**Black Box**", due to their **difficult interpretability**. This can be a **problem** in many sectors.

Usage example: One model predicts that a bank should not lend someone money, and the bank is legally required to explain the basis for each loan refusal.



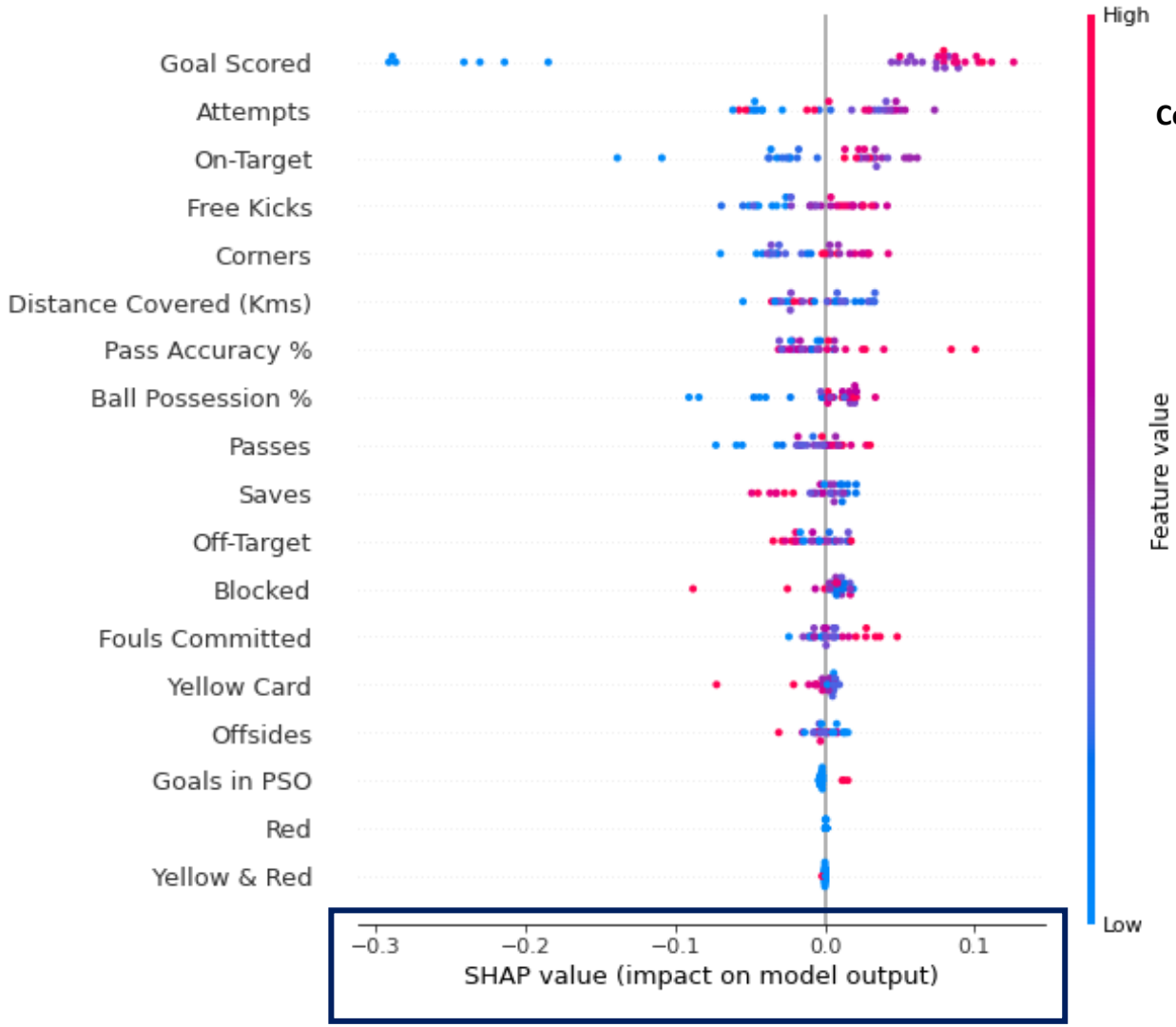
# SHAP

A widely used technique to understand the **impact** of a variable on the prediction of a model is **SHAP (SHapley Additive exPlanations).**



# SHAP Summary Plots

**Verticality:** Variables ordered by relevance

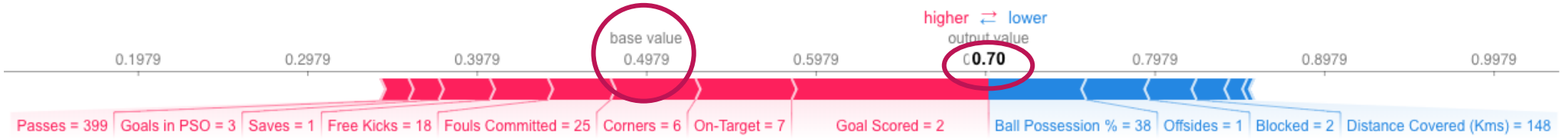



**Color:** Value of the variable. Pink for high values, blue for low

**Horizontality:** Effect in prediction. Left: less chance of winning, Right: more chance of winning

# SHAP Values

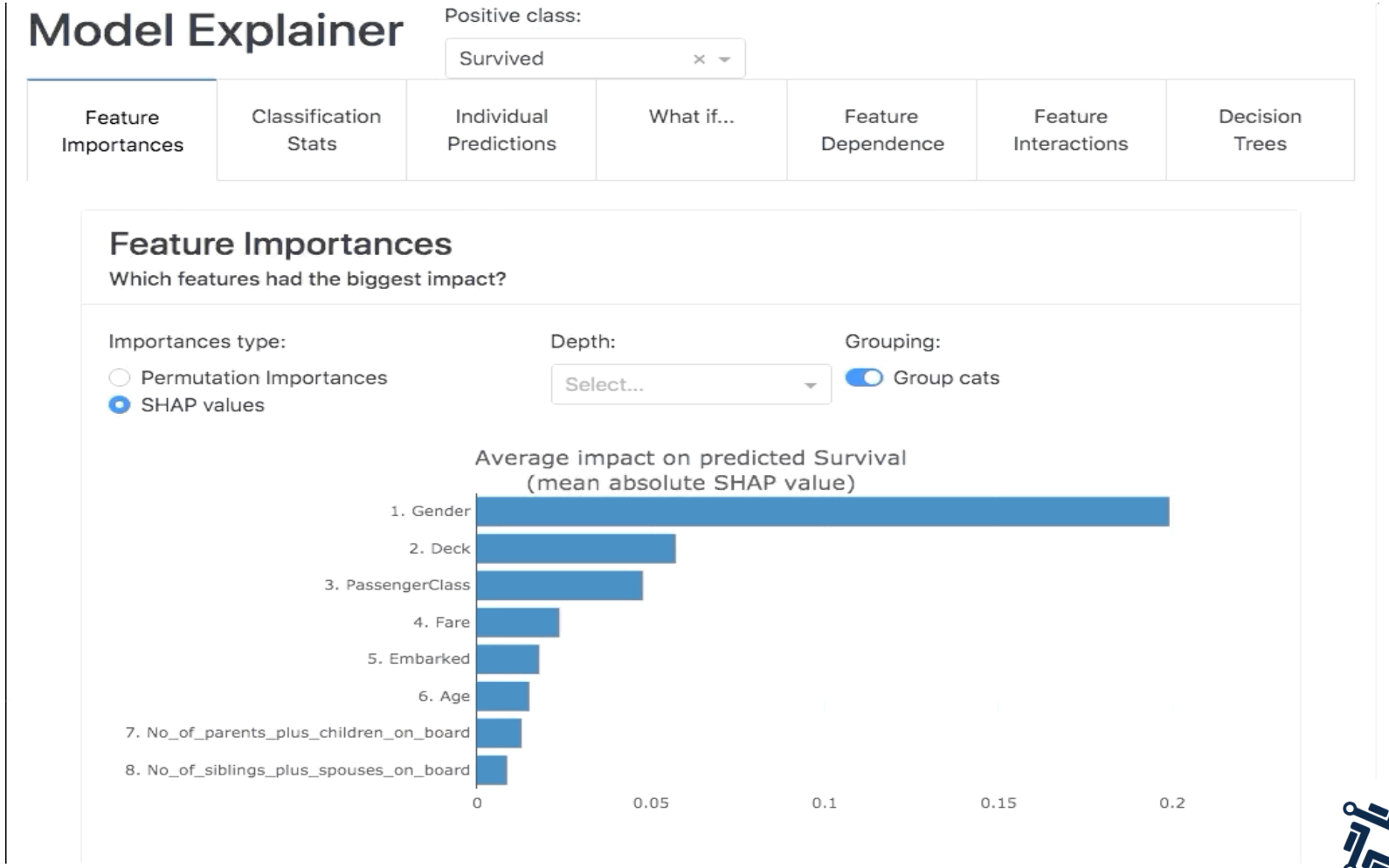
**SHAP values** interpret the **impact** of having a **certain value for a feature** compared to the prediction we would make if that feature took some reference value. Example: How much would a prediction change if the team scored 2 goals?




*0.7 vs base value: 0.4979. Features that cause an increase in predictions in pink and features that decrease prediction in blue. The biggest impact of the goal scored is 2. Possession of the ball has a significant effect that decreases the prediction.*

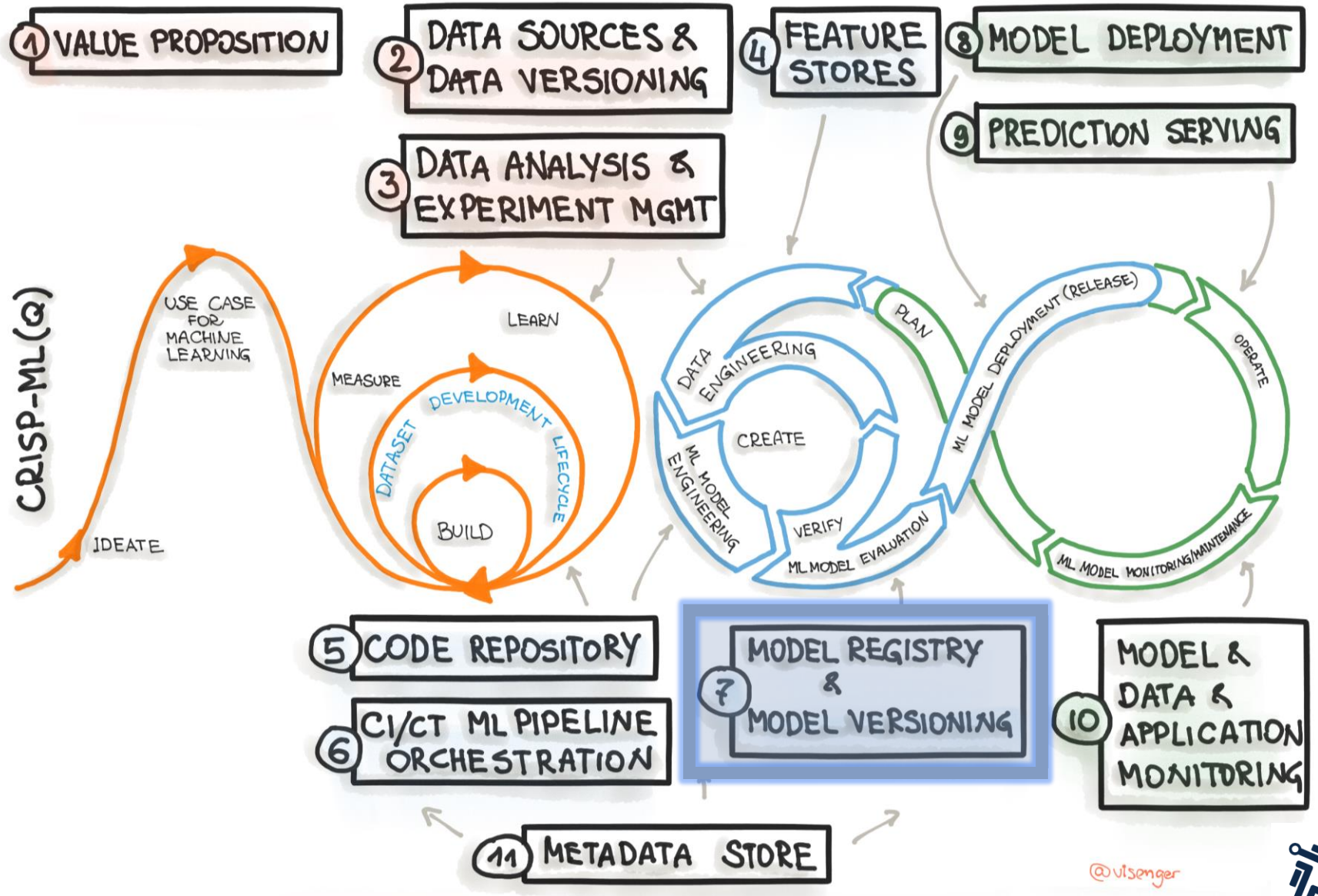
# Explainer Dashboard

The **Explainer dashboard** is a library for quickly creating interactive dashboards to analyze and explain the predictions and performance of machine learning models.



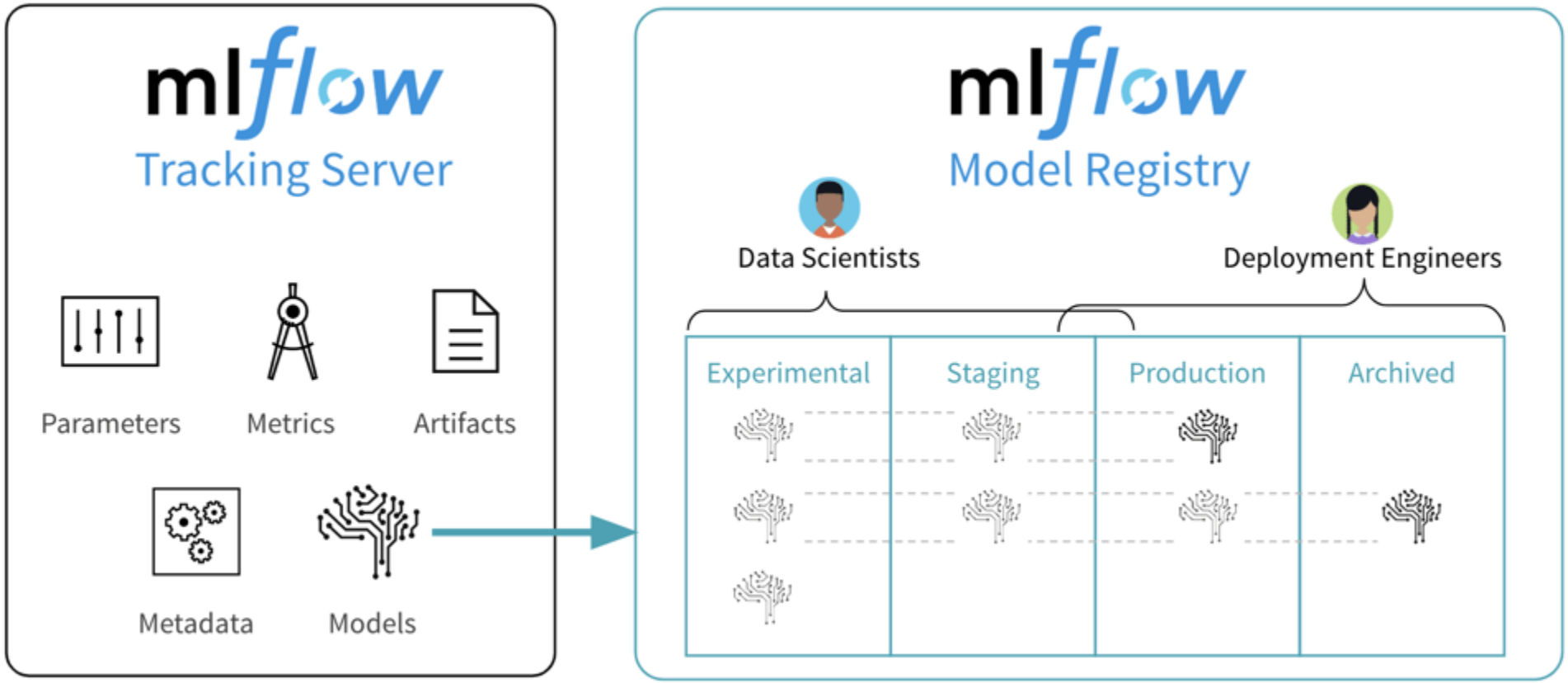
# Model registration and versioning

# MLOps stages

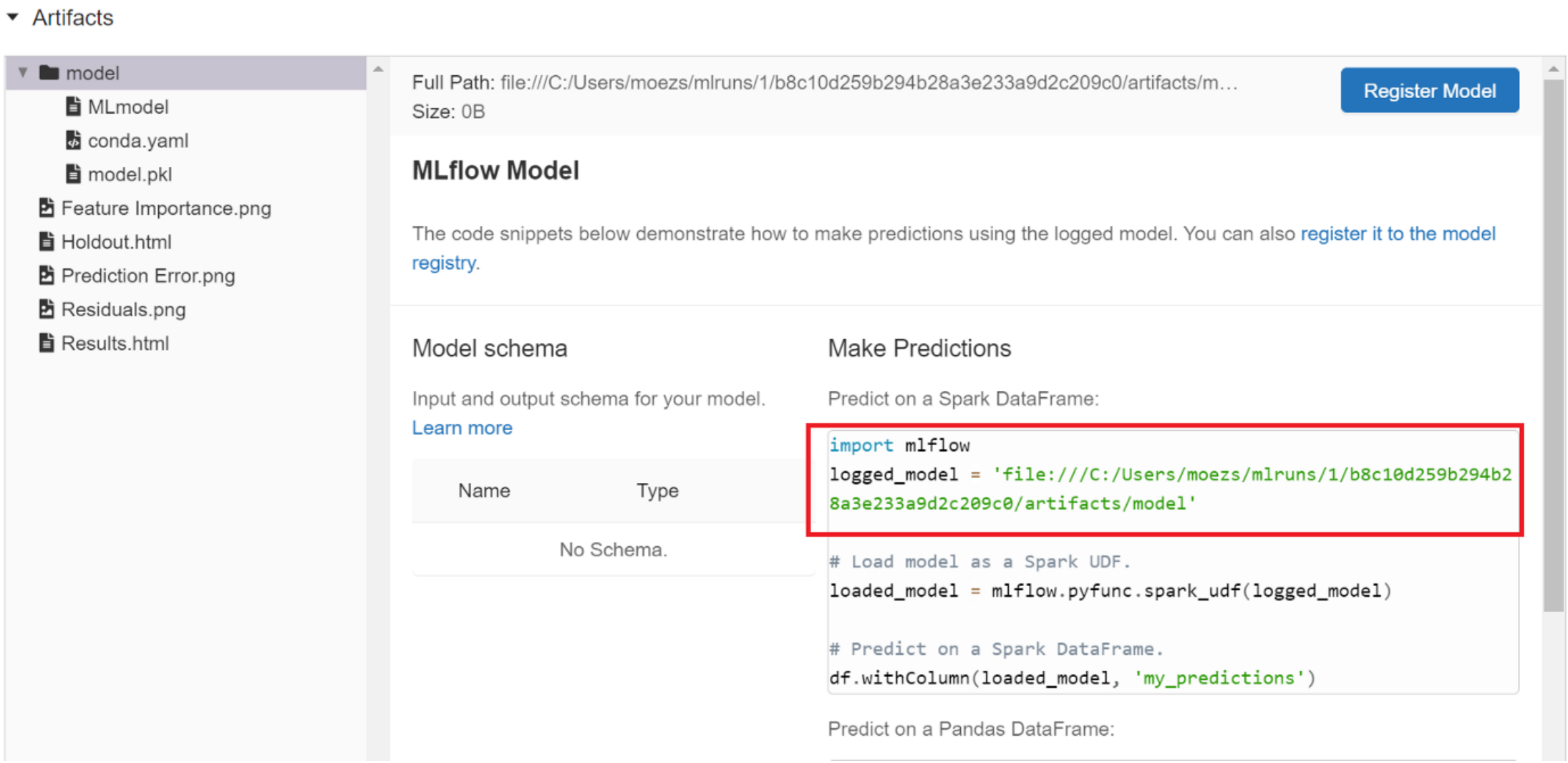


# Tools for MLOps

**MLflow** is an open source platform for managing the ML lifecycle, including model experimentation, reproducibility, deployment, and registration.



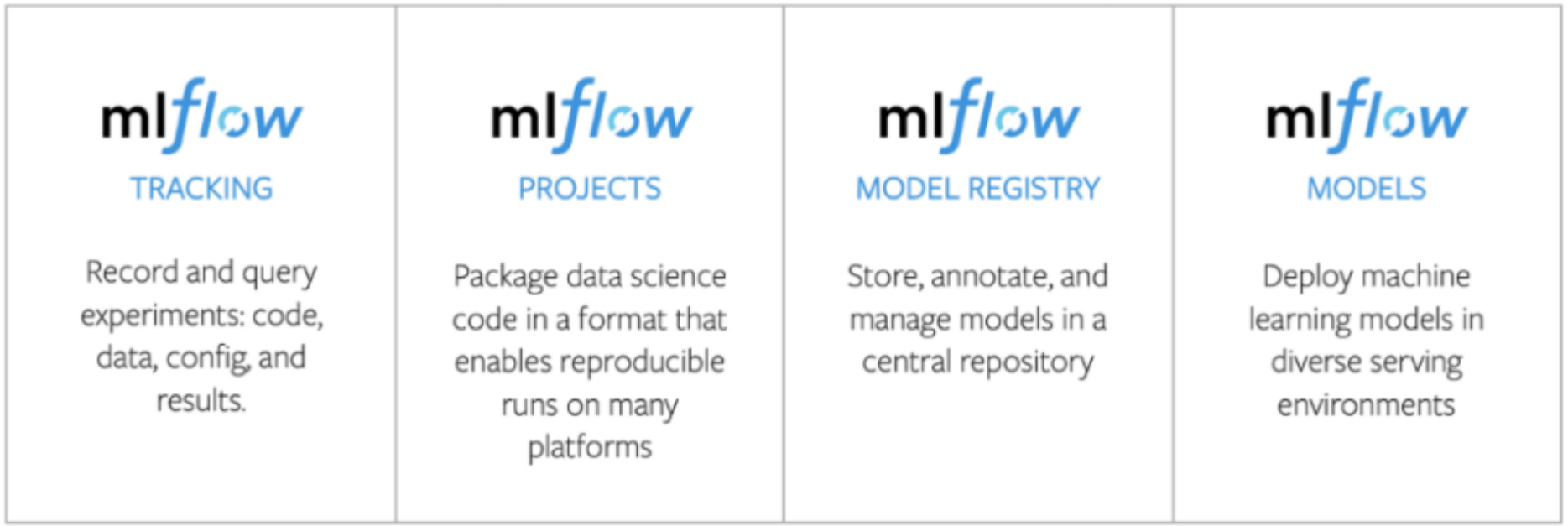
# MLFow platform



# Registration and versioning with MLFlow

To be able to enter a code from scratch in **MLFlow**, follow these steps:

https://www.mlflow.org/docs/latest/tutorials-and-examples/tutorial.html



# Pycaret & MLFlow

To be able to register models in MLFlow from Pycaret is very simple.

```
# initialize configuration
s = setup(data, target = 'Precio', transform_target = True, log_experiment = True, experiment_name
= 'diamond')

# within notebook (notice ! sign infront)
!mlflow ui
# on command line in the same folder
mlflow ui

# acceder a MLFlow
"localhost:5000"
```

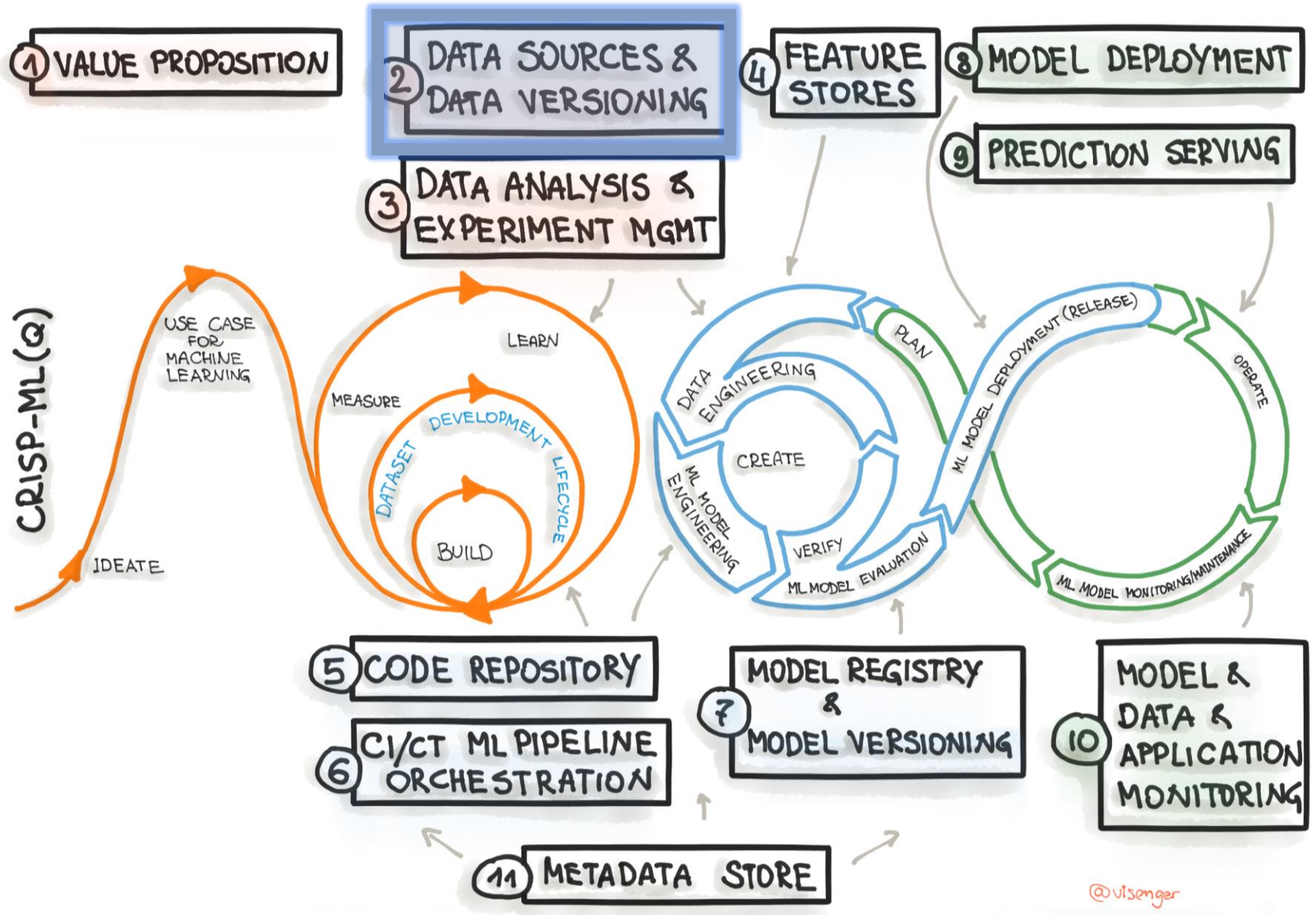
# Load the MLFlow model

In order to load the model and use it, we only have to access the MLFlow path that contains the model

```
# load model
from pycaret.regression import load_model
pipeline =
load_model('C:/Users/moezs/mlruns/1/b8c10d259b294b28a3e233a9d2c209c0/artifacts/model/model’)

# print pipeline
print(pipeline)
```

# Versioning datasets with DVC

# MLOps stages



# DVC

**Git** cannot be used for datasets versioning, specially with large datasets. **For dataset versioning** we have **DVC**, which integrates with Git. It is an open-source command-line tool that mimics Git flows and commands.

DVC Characteristics:

- **Git-compliant**
- Easy data **version control**
- **Storage** independent
- Reproducible
- Language and **framework independent**
- Low friction **branching**
- Easy to use

# Tools Comparative

Next, we have a comparison of different data storage tools:

| | Open Source | Data Format Agnostic | Cloud/Storage Agnostic* (Supports most common cloud and storage types) | Simple to Use | Easy Support for BIG Data |
|---|---|---|---|---|---|
| DVC | ✅ (Apache 2.0) | ✅ | ✅ | ✅ | ❌ |
| DELTA LAKE | ✅ (Apache 2.0) | ❌ | ✅ | ❌ | ✅ |
| Git Large File Storage | ✅ (MIT) | ✅ | ❌ | ✅ | ❌ |
| Pachyderm | ✅⚠ (Non-standard license) | ✅ | ✅ | ❌ | ✅ |
| DOLT | ✅ (Apache 2.0) | ❌ | ❌ | ❌ | ❌ |
| lakeFS | ✅ (Apache 2.0) | ✅ | ❌ | ❌ | ✅ |

DAGsHub

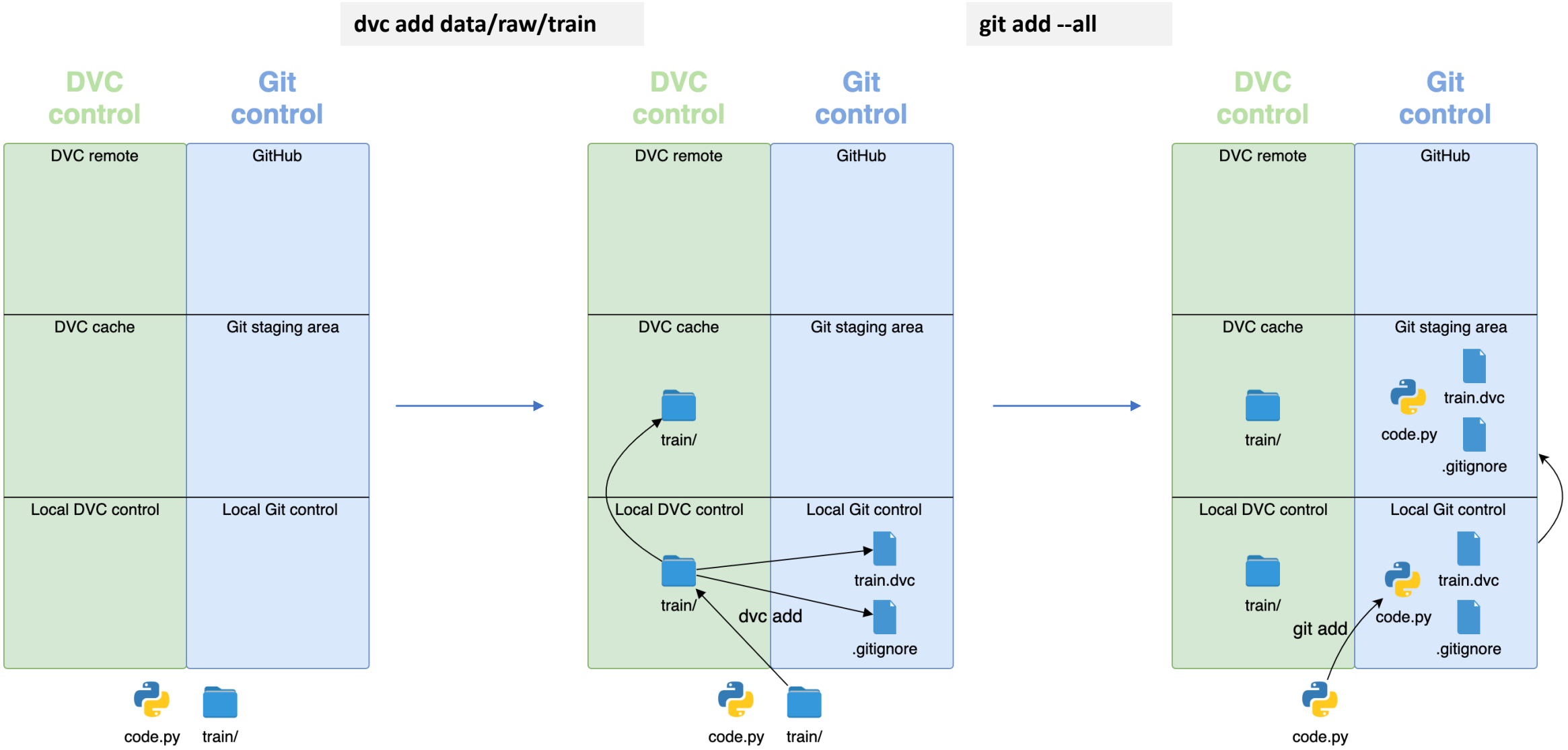
# DVC Commands

**DVC main commands** to upload data to a remote environment.

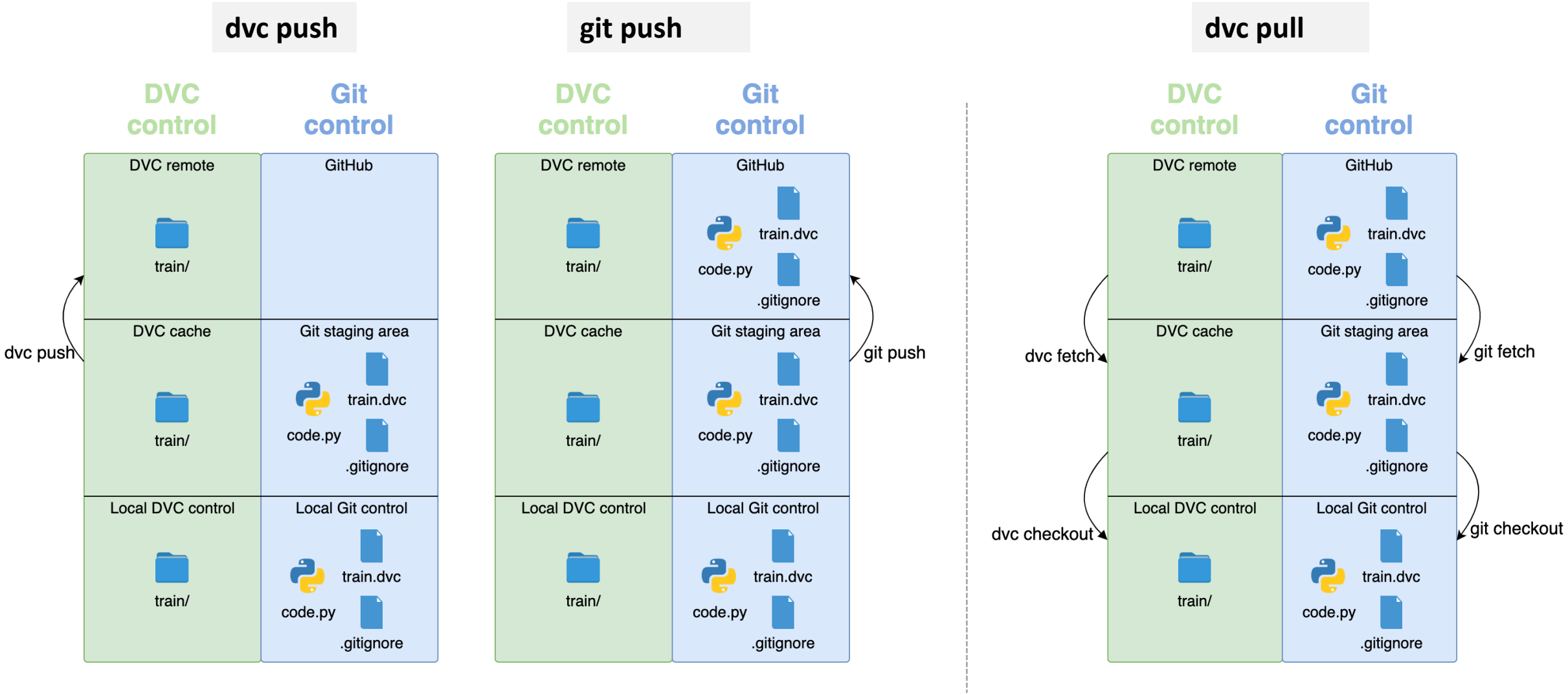
```
$ dvc init # initialize the repo
$ dvc add . # add the files that have been changed
$ dvc commit -m "making some changes" # commit the updates with a message
$ dvc remote add newremote s3://bucket/path # point the repo to an S3
bucket for storage
$ dvc push # push the changes to the DVC repo hosted in the default S3
bucket
$ dvc pull # pull the latest changes from the DVC repo hosted in the
default S3 bucket
```

# File tracking with DVC

**Large** data files and folders go to **DVC remote storage**, but **small .dvc** files go to **GitHub**



# Upload and download files with DVC



# DVC Files

DVC files are **YAML** files. Information is stored in **key-value** pairs and lists. The first key is **md5** followed by a string of characters. MD5 is a hash function. Two files that are exactly the same, will produce the same hash.

YAML

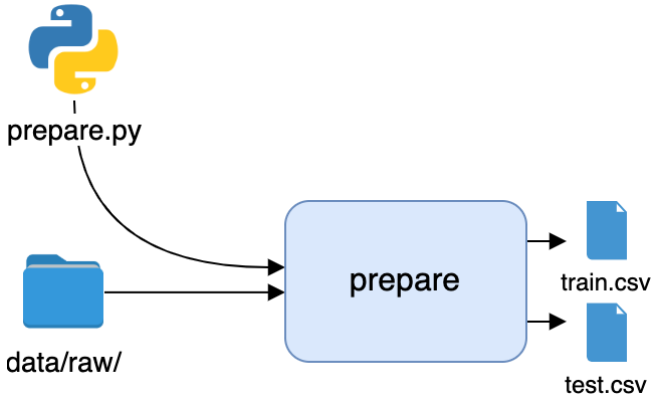
```yaml
md5: 62bdac455a6574ed68a1744da1505745
outs:
  - md5: 96652bd680f9b8bd7c223488ac97f151
    path: model.joblib
    cache: true
    metric: false
    persist: false
```

# DVC pipelines

**DVC** allows to **chain the files** of the **entire process** in a single execution called the **DVC pipeline** that requires a single command to be executed: **dvc repro**.

A pipeline consists of multiple stages, and each stage has three components:

- Dependencies
- Outputs
- Command




Shell

```
dvc run -n prepare \
      -d src/prepare.py -d data/raw \
      -o data/prepared/train.csv -o data/prepared/test.csv \
      python src/prepare.py
```

# DVC Commands

DVC will create two files:

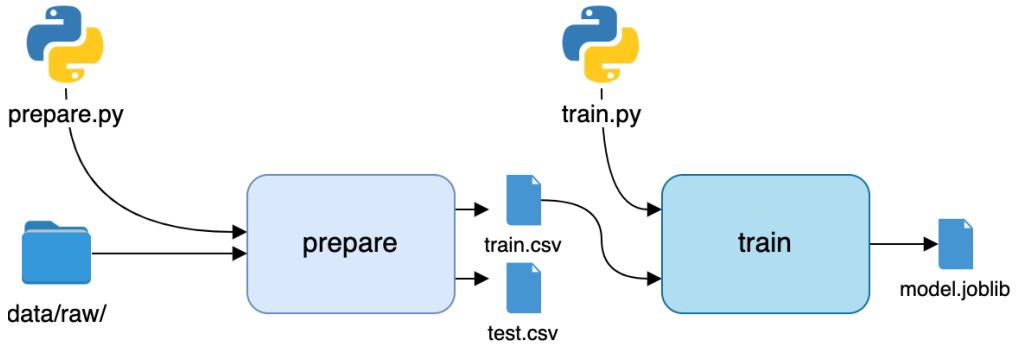
- **dvc.yaml**
- **dvc.lock**

The internal information of both files is similar, with the addition of MD5 hashes in dvc.lock

YAML

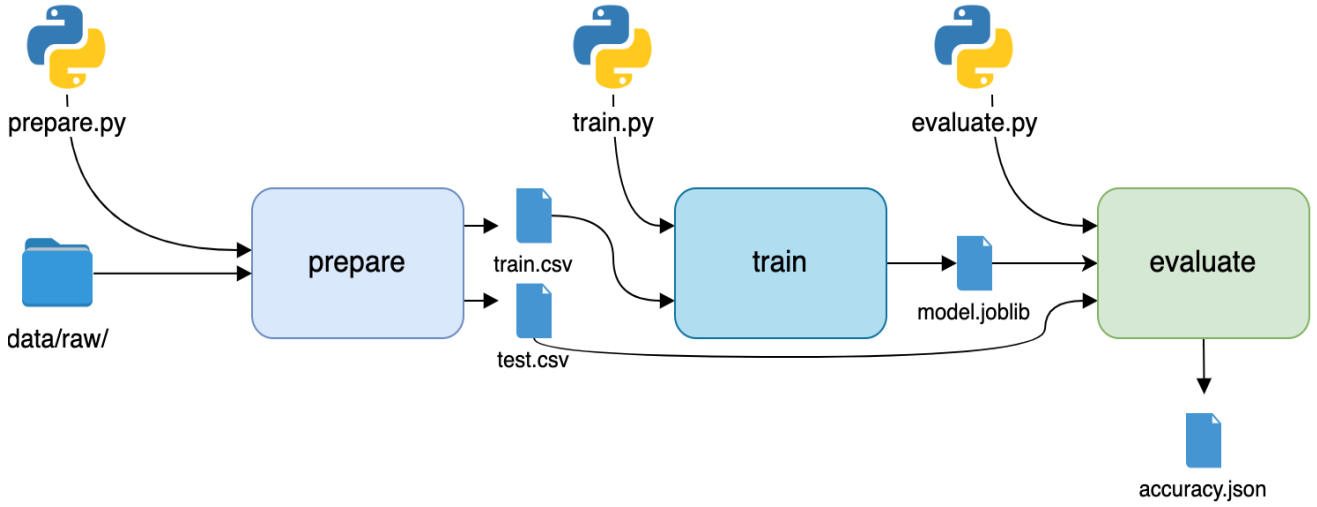
```yaml
prepare:
  cmd: python src/prepare.py
  deps:
    - path: data/raw
      md5: a8a5252d9b14ab2c1be283822a86981a.dir
    - path: src/prepare.py
      md5: 0e29f075d51efc6d280851d66f8943fe
  outs:
    - path: data/prepared/test.csv
      md5: d4a8cdf527c2c58d8cc4464c48f2b5c5
    - path: data/prepared/train.csv
      md5: 50cbdb38dbf0121a6314c4ad9ff786fe
```

# Examples of DVC pipelines




Shell

```
$ dvc run -n train \
        -d src/train.py -d data/prepared/train.csv \
        -o model/model.joblib \
        python src/train.py
```

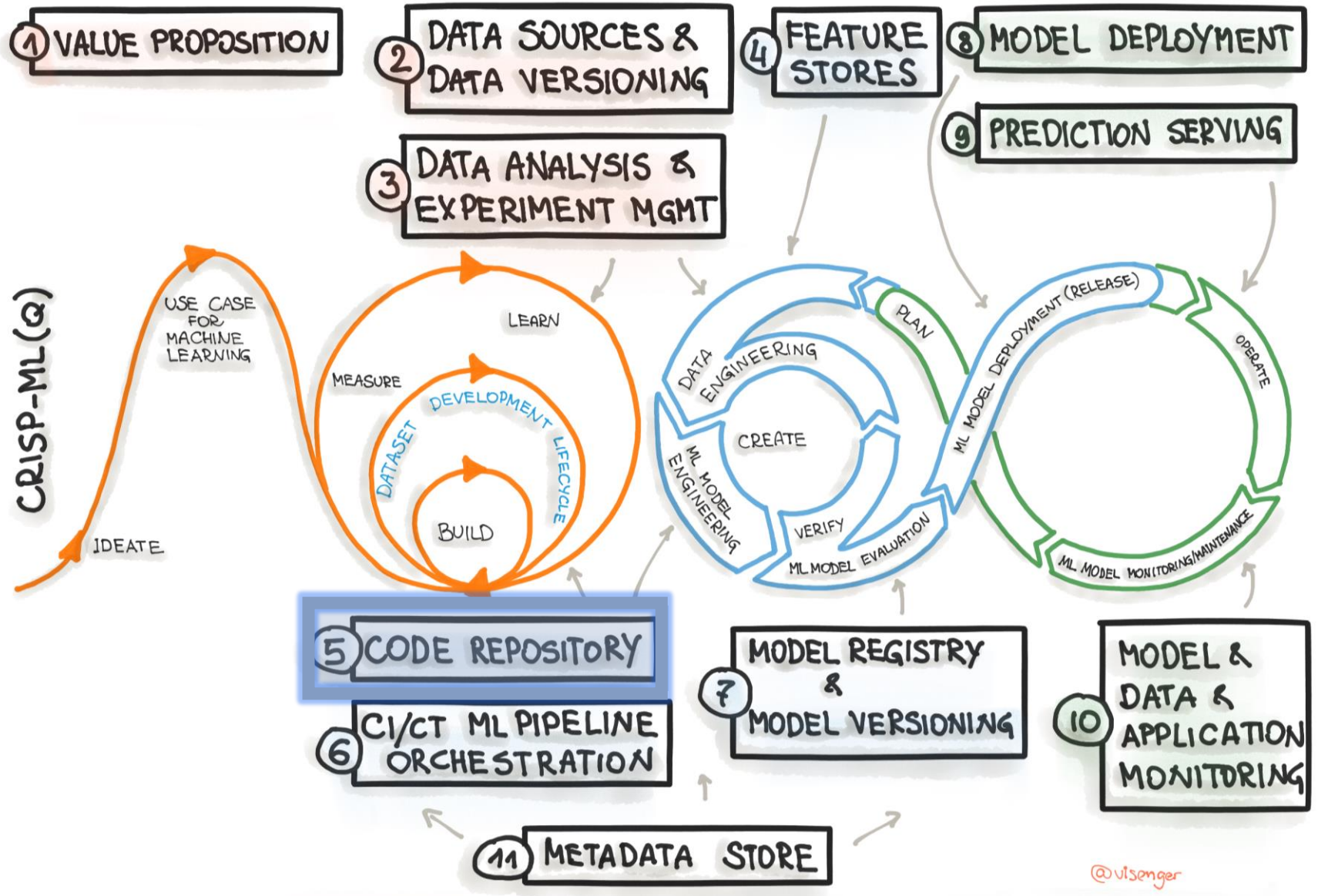



Shell

```
$ dvc run -n evaluate \
        -d src/evaluate.py -d model/model.joblib \
        -M metrics/accuracy.json \
        python src/evaluate.py
```

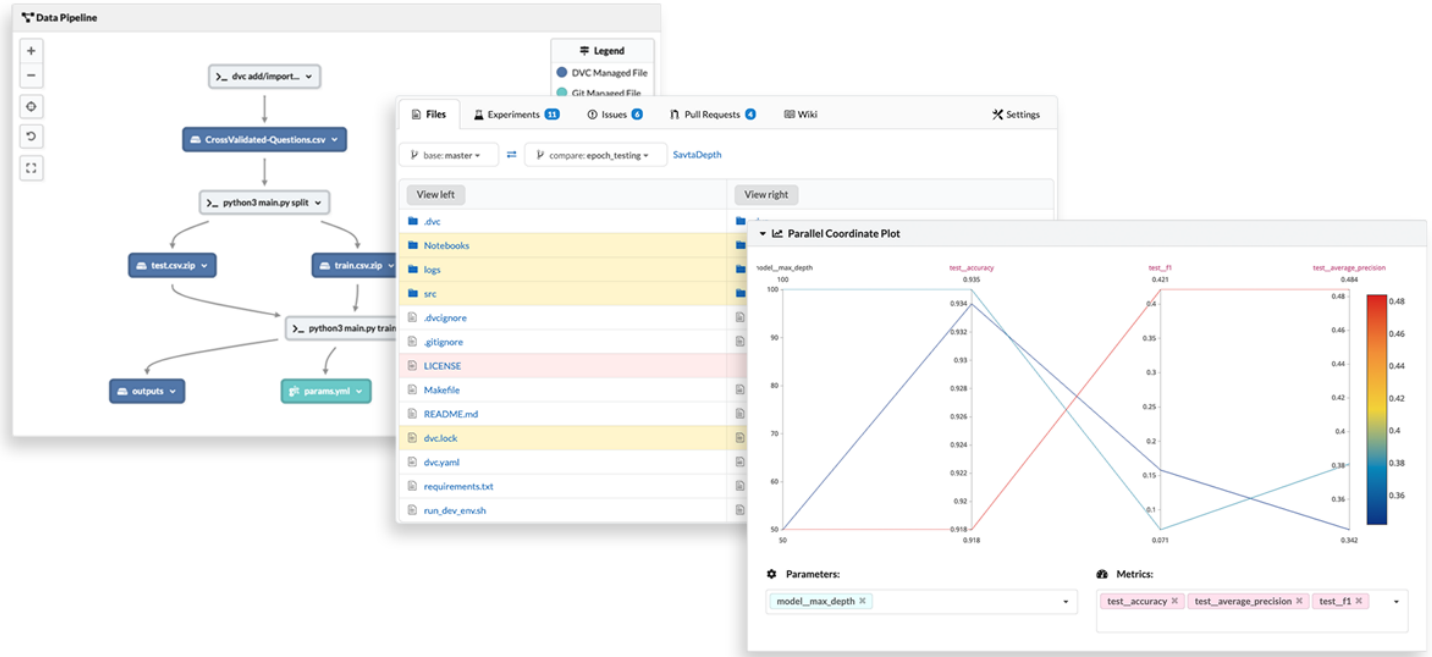
# Code Repository

# MLOps stages



# DagsHub

It is a platform **for versioning** data, models, experiments, and code. Easily share, review, and reuse teamwork with a **GitHub-like experience**. Based on open source.




# Dagshub Proyect

We will develop a **ML model** for predicting if the StackOverflow questions are related to Machine Learning topic.

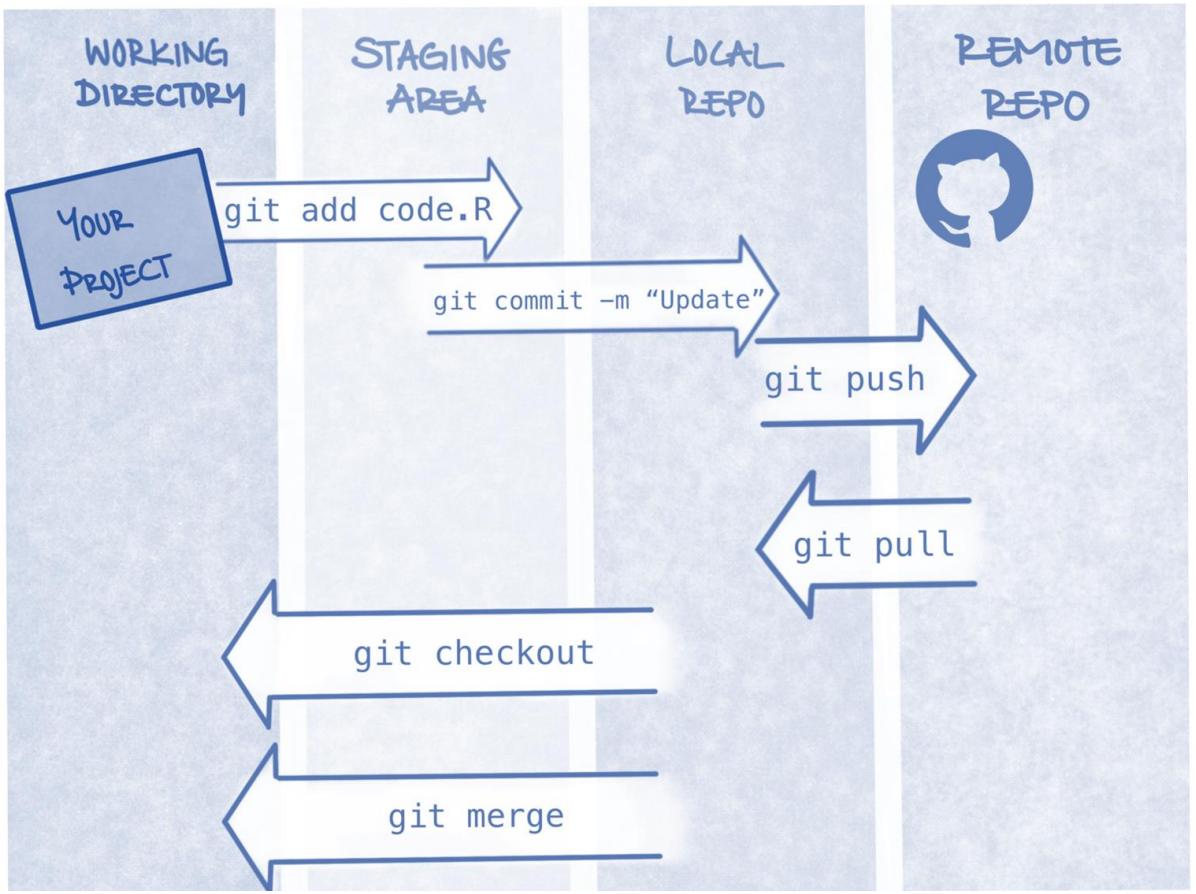
**Project link**: https://dagshub.com/docs/experiment_tutorial/

Levels:

- **Data exploration:** We will analyze the data to try to understand it and know which transformations should be applied.
- **DagsHub Setup**: We will create a DagsHub account and project.
- **Model and data version control**: We will develop a ML model, register it, and use DVC to version its dataset
- **Experimentation with DagsHub**: Experiment with model hyperparameters and configuration, and register experiments  in DagsHub

# Deployment process with Git

In order to deploy a local project in DagsHub through Git we must follow the following process:



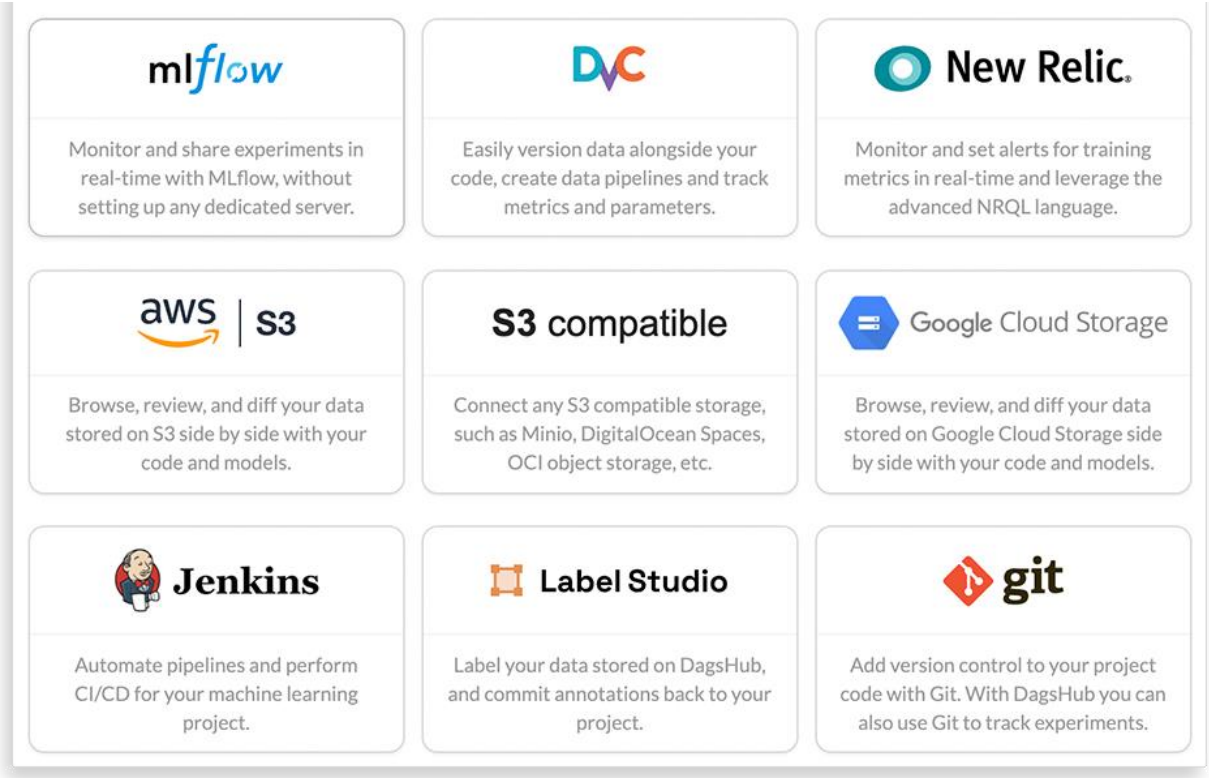
# Model improvements for Deployment

In order to have an ML model suitable for versioning and experimentation, code should follow some requirements. List of considerations for production model:

- **The dataset must be reproducible**. Our test dataset will be different each time we run the script. To compare different models and experiments, test dataset should not vary.
- **Cross validation.** It is necessary to apply a cross validation when classes are unbalanced.
- **Random seeds**. To get reproducible results code should have random_seed.
- **Model saving**. We should save our model; otherwise, we couldn't use it in production

# PyCaret and DagsHub integration

One of the **limitations of Mlflow** is that it only allows you to **register data locally**. To collaborate on Mlflow, **DagsHub** is very useful, as it provides a **remote MLflow server**. It also integrates DVC to version data.



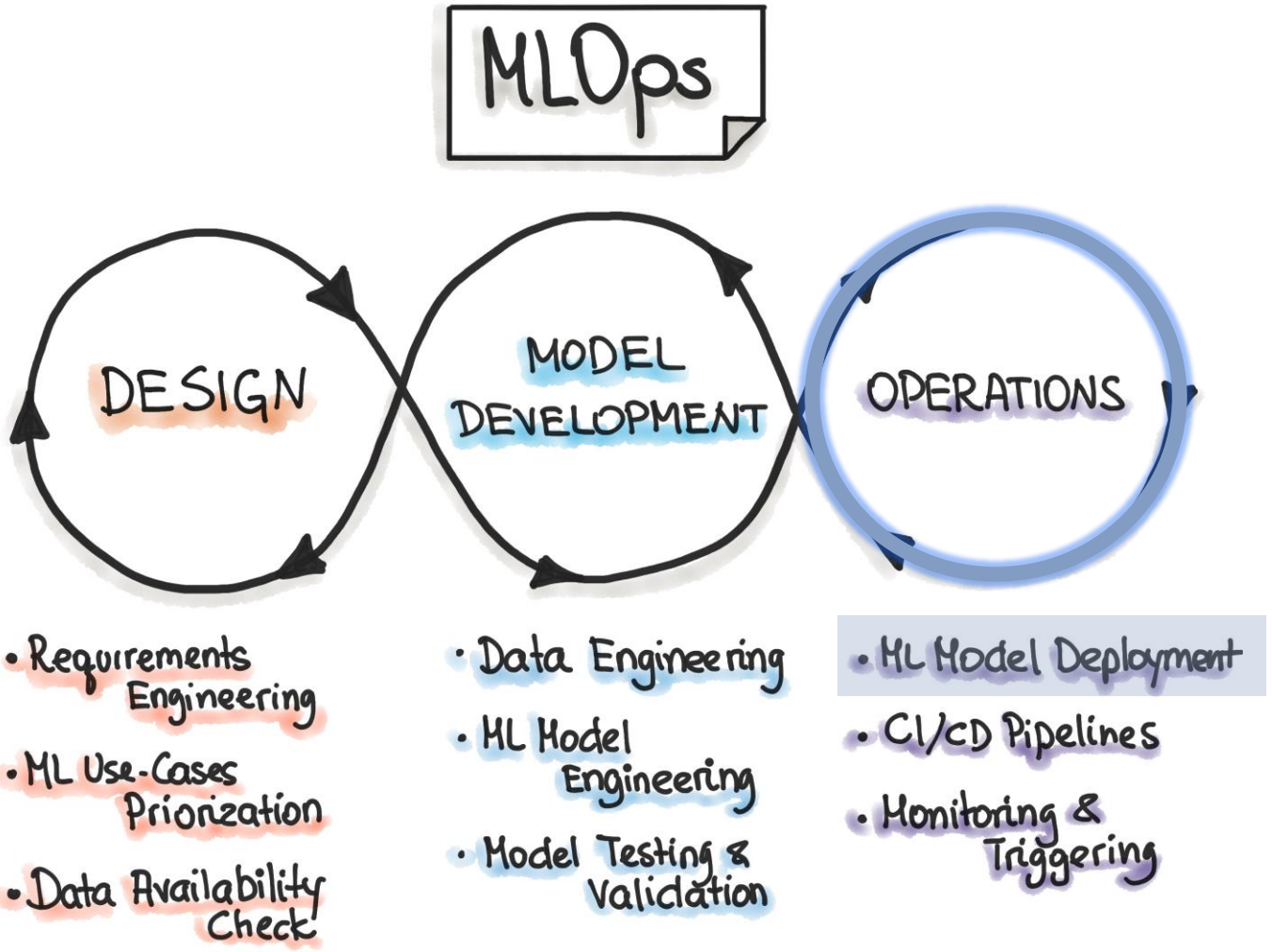
# DagsHub with Pycaret

To use DagsHub Logger with PyCaret, we must use *log_experiment = 'dagshub'* in *setup function.*

```
from pycaret.datasets import get_data
from pycaret.regression import *

data = get_data('diamond')

s = setup(data,
          target = 'Price',
          transform_target=True,
          log_experiment="dagshub",
          experiment_name='predict_price',
          log_data=True)
```

# Deploying Models in Production

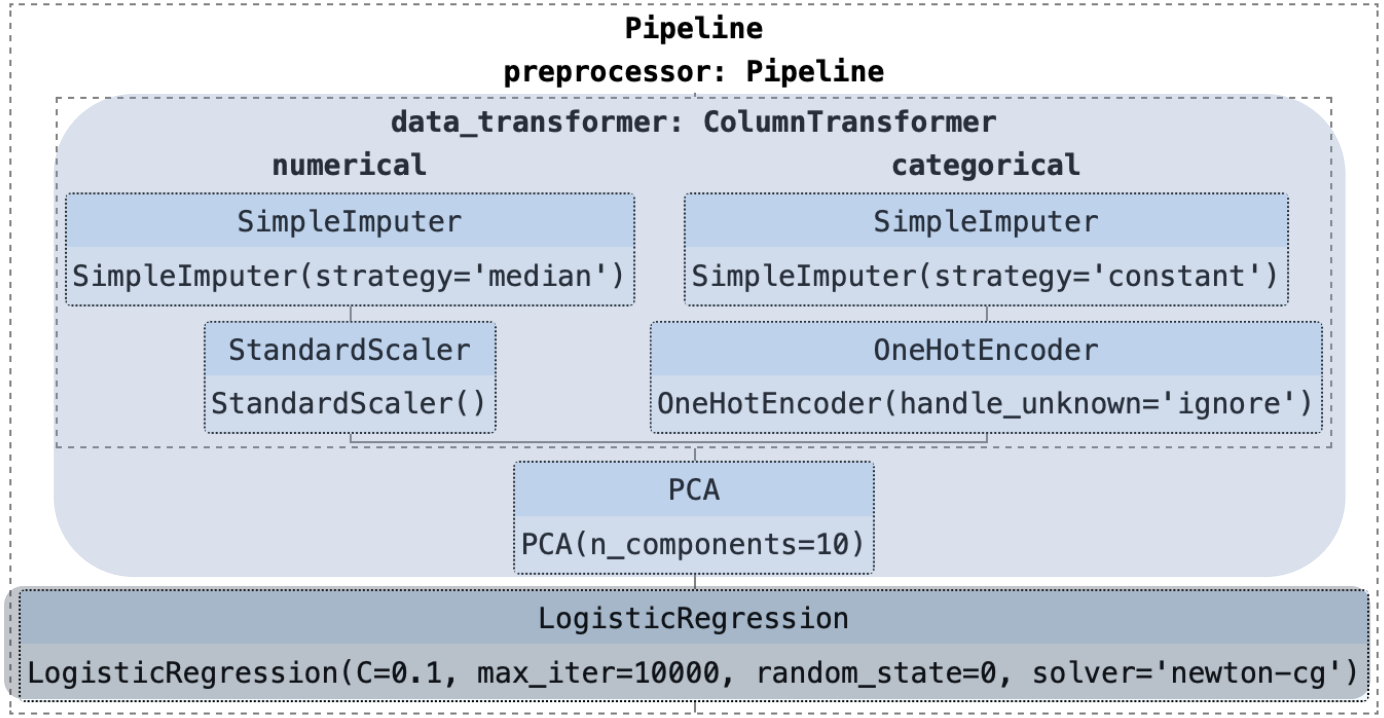
# MLOps stages



# Model Deployment

**Model Deployment** consists of integrating a model into **production environment** to make data-driven business decisions.

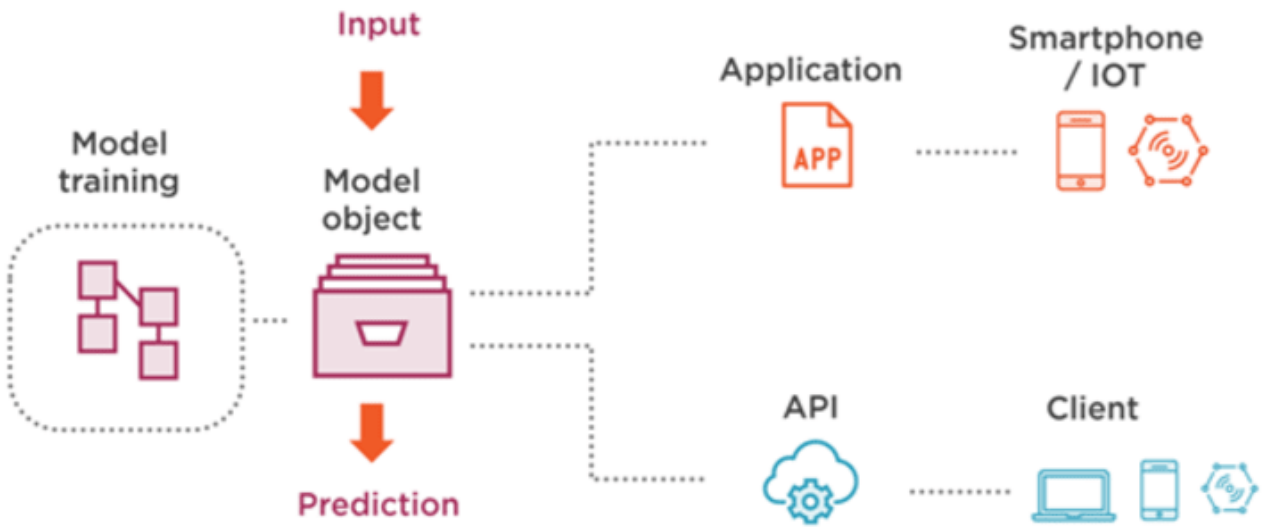
Models must be available for **web applications**, enterprise **software** (ERP), and **APIs**, to provide predictions for new data.



# Model Serving

There are different **alternatives to deploy a model** in a production environment:

1. Through **API**
2. Through **applications** (mobile/web)

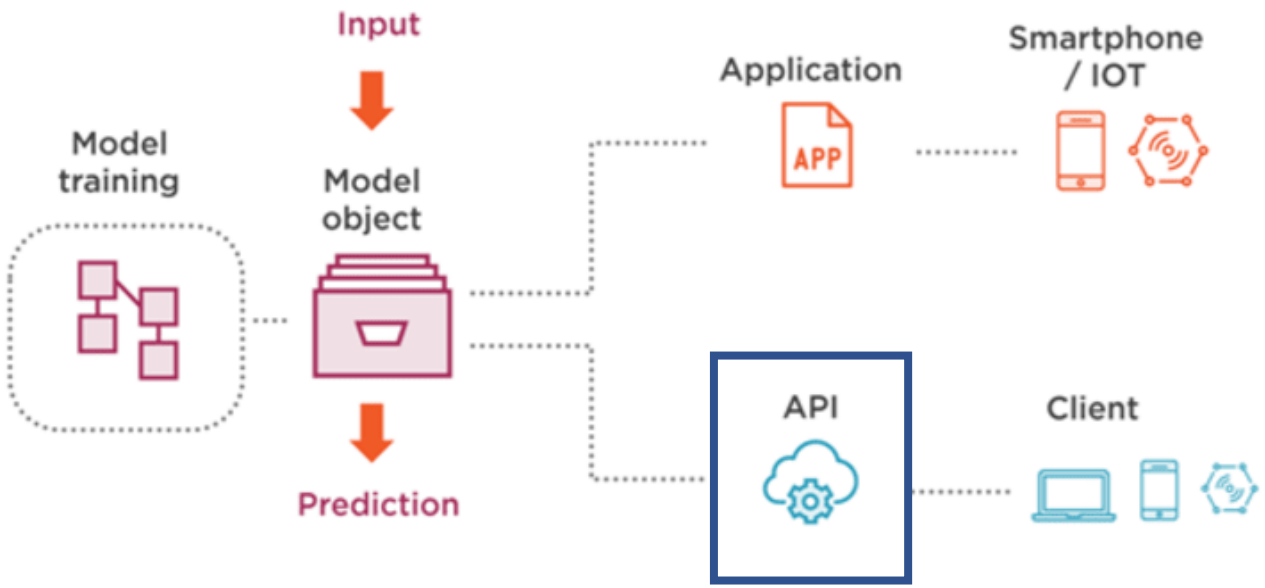


# API creation

# Model Serving

There are different **alternatives to deploy a model** in a production environment:

1. Through **API**
2. Through **applications** (mobile/web)



# What is an API?

**API** (Application Programming Interface) creates an entry point for an application, through HTTP requests.

API: Application Abstraction + Simplification of Third Party Integration

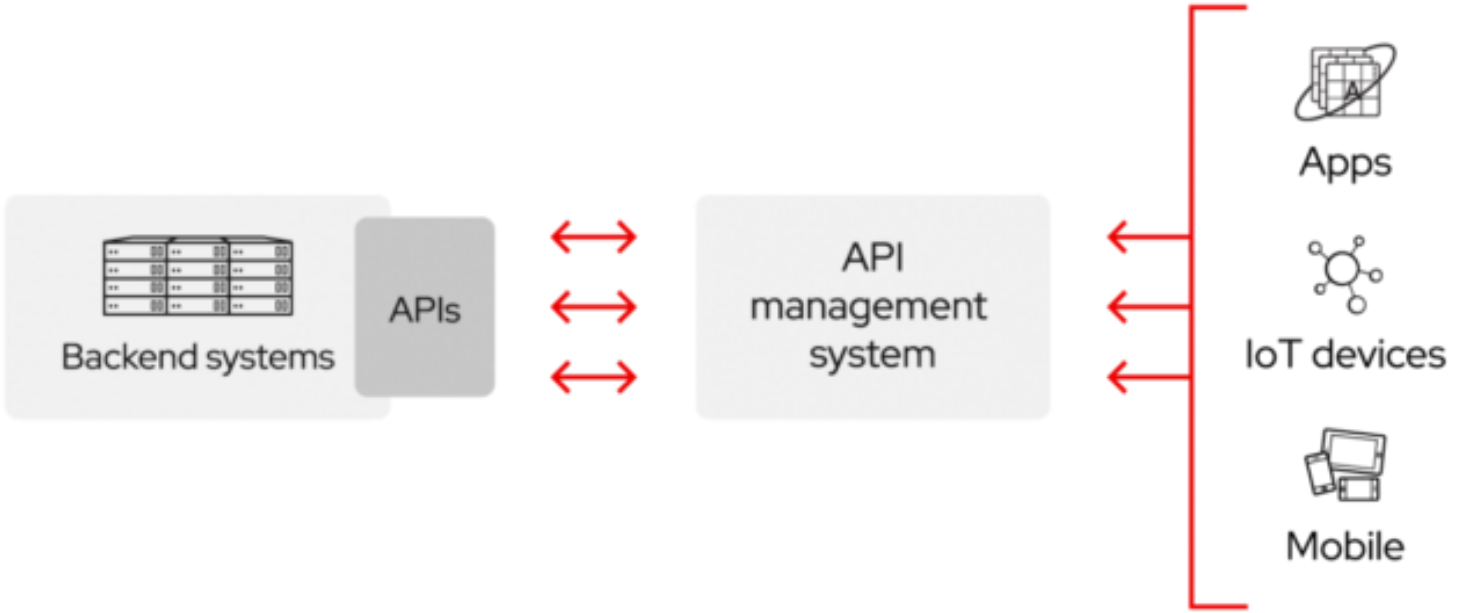



Image source: https://kunal3836

# Status codes and HTTP methods

## HTTP methods

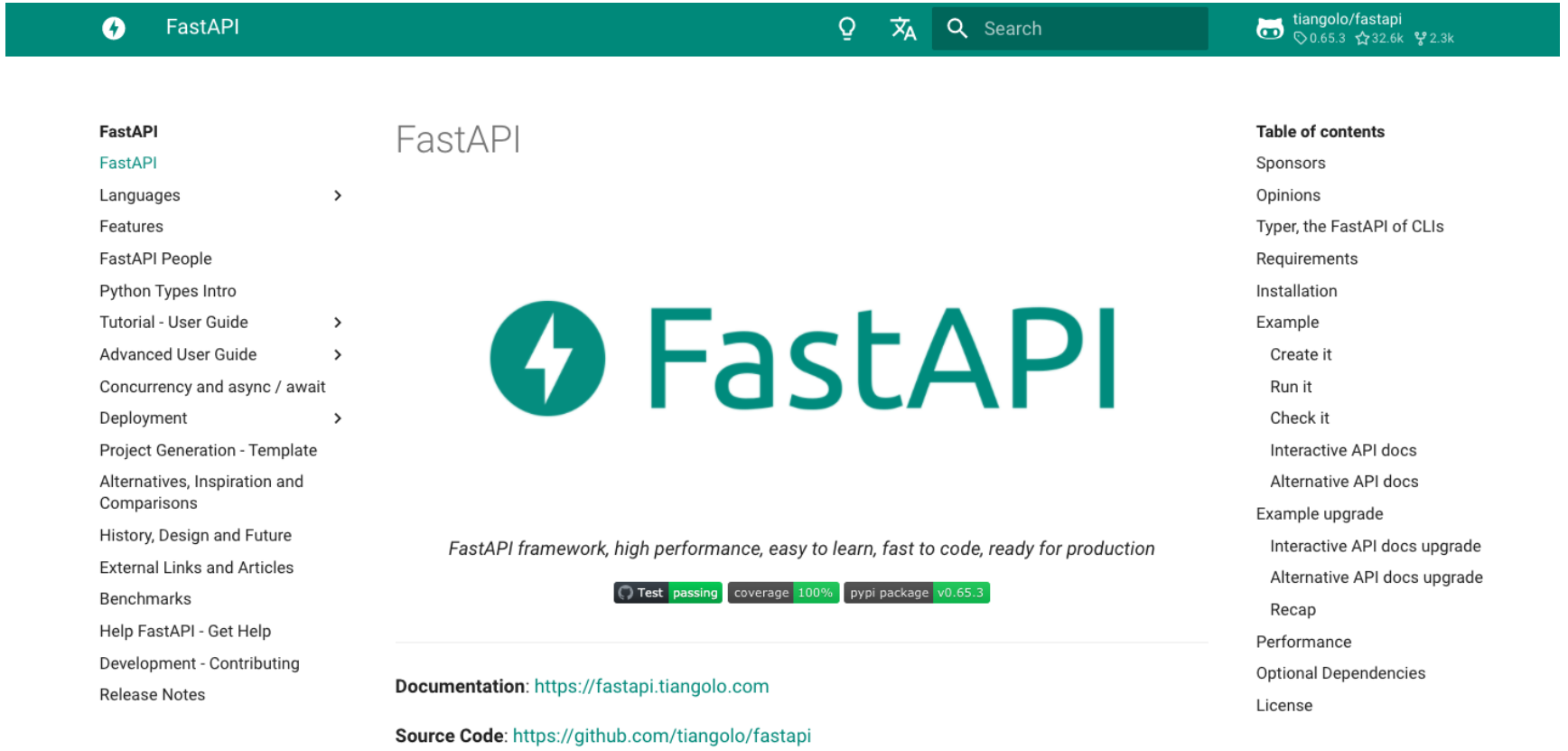
- **GET**: retrieve an existing resource (read only)
- **POST**: Create a new resource/send information
- **PUT**: Update an existing resource
- **PATCH**: partially update an existing resource
- **DELETE**: Delete a resource

## HTTP status code

- **2xx**: Successful operation
- **3xx**: Redirect
- **4xx**: client error
- **5xx**: Server Error

# FastAPI

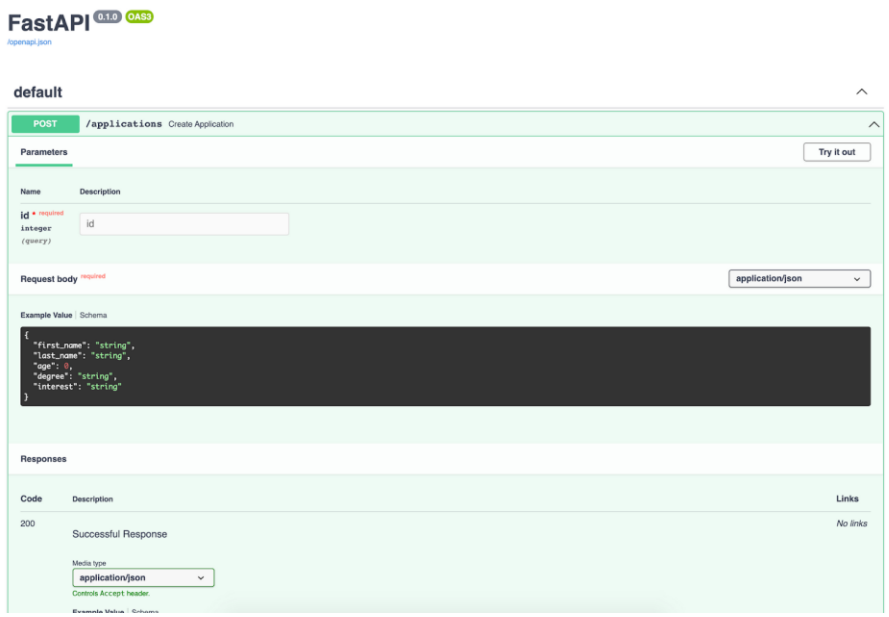
It is the reference framework for **creating robust and high-performance APIs** for production environments.



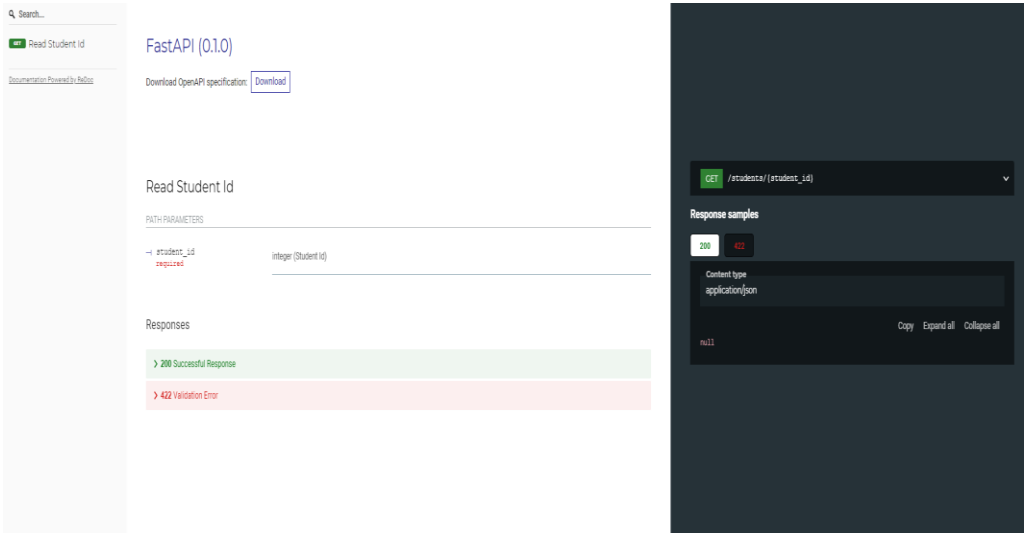
# API Documentation

Interactive API **exploration** and **documentation** are automatically generated

http://localhost:8000/docs




http://localhost:8000/redoc




Data Bootcamp
BEST DATA TRAINING

# Fast API Basics

**Basic function**

```
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
    uvicorn.run("hello_world_fastapi:app")
```

**Methods**

```
@app.get("/")
def home():
return {"Hello": "GET"}

@app.post("/")
def home_post():
return {"Hello": "POST"}
```

# Fast API Basics

## Query Parameters

```
@app.get("/employee")
def home(department: str):
   return {"department": department}
```

## Path parameters

```
@app.get("/employee/{id}")
def home(id: int):
   return {"id": id}
```

## Data Type Validation: Pydantic

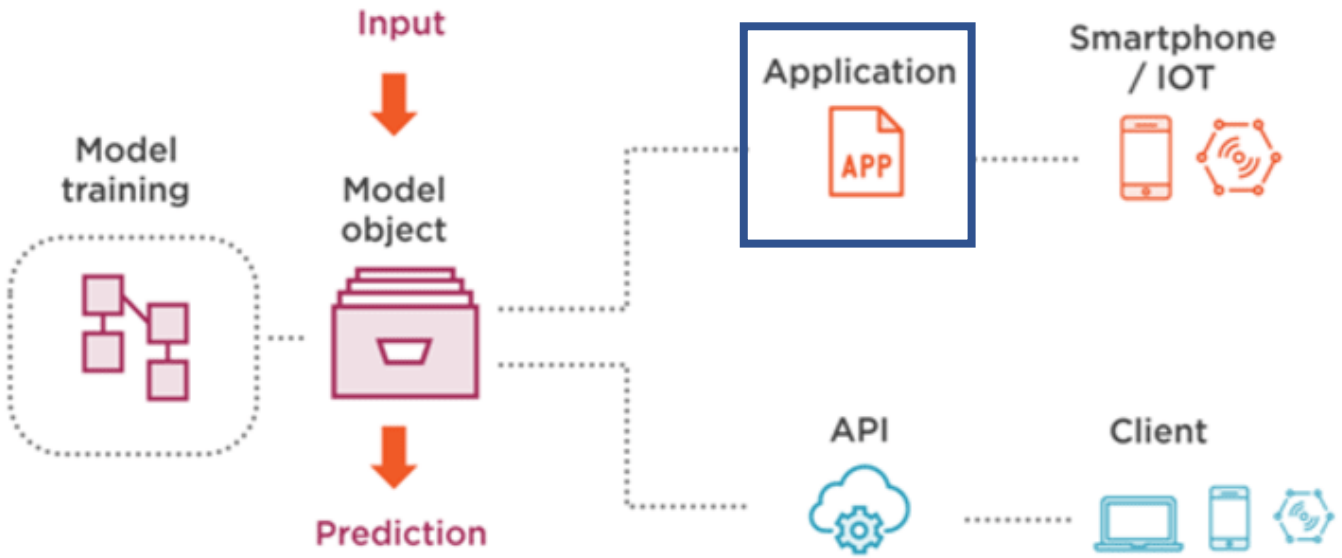
```
from fastapi import FastAPI, Query
from pydantic import BaseModel
from typing import Optional


class Application(BaseModel):
    first_name: str
    last_name: str
    age: int
    degree: str
    interest: Optional[str] = None

class Decision(BaseModel):
    first_name: str
    last_name: str
    probability: float
    acceptance: bool

app = FastAPI()

@app.post("/applications", response_model=Decision)
async def create_application(id: int, application: Application):

    first_name = application.first_name
    last_name = application.last_name
    proba = random.random()
    acceptance = proba > 0.5
```

# Developing web applications

# Model Serving

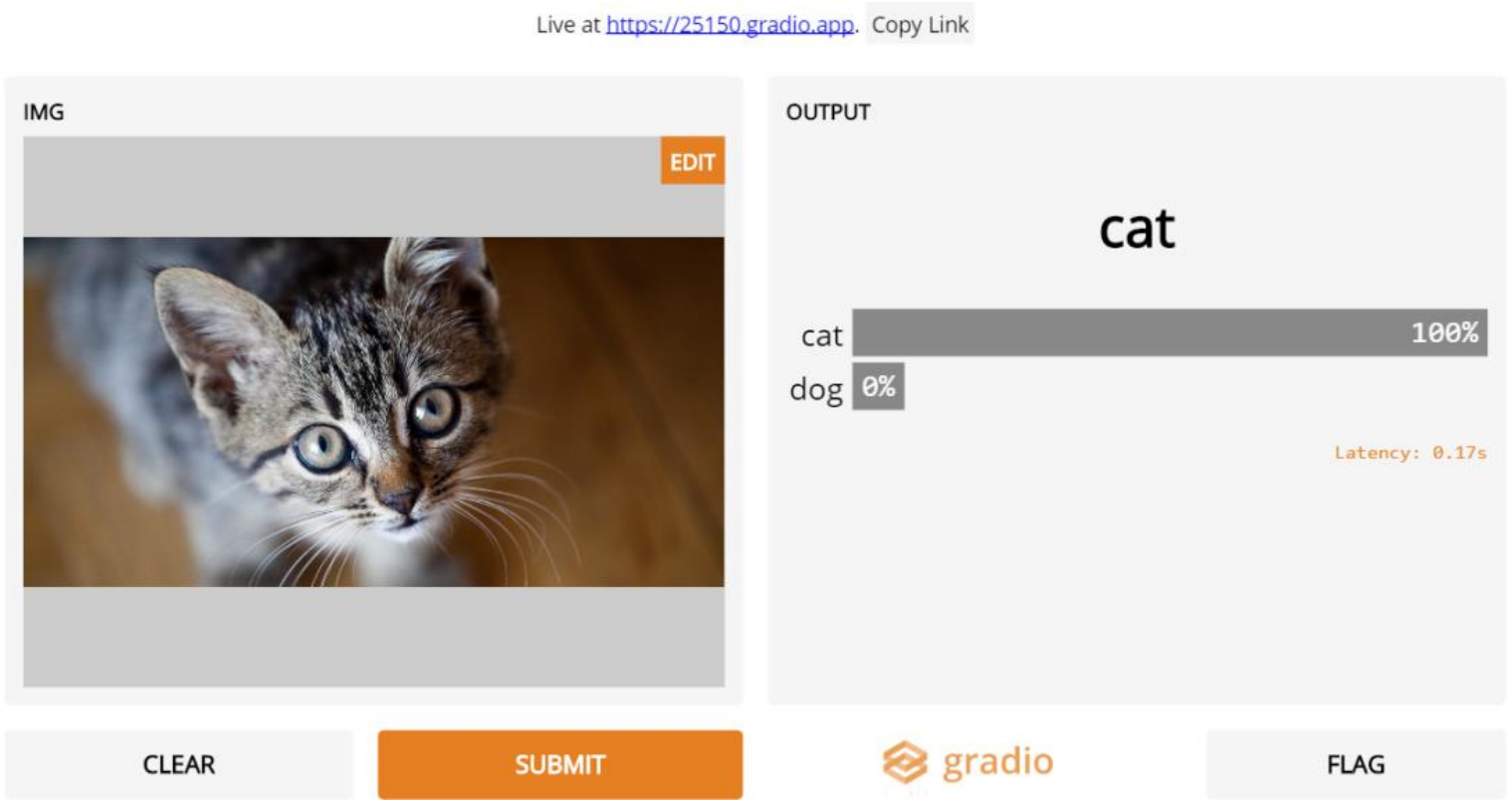
There are different **alternatives to deploy** a model in a **production** environment:

1. Through **API**
2. Through **Applications** (mobile/web)



# Create web application

A **business user** will not execute a code or notebook to get a prediction. For this reason, it is advisable to facilitate the process through **a front-end** (web application, mobile, etc.).



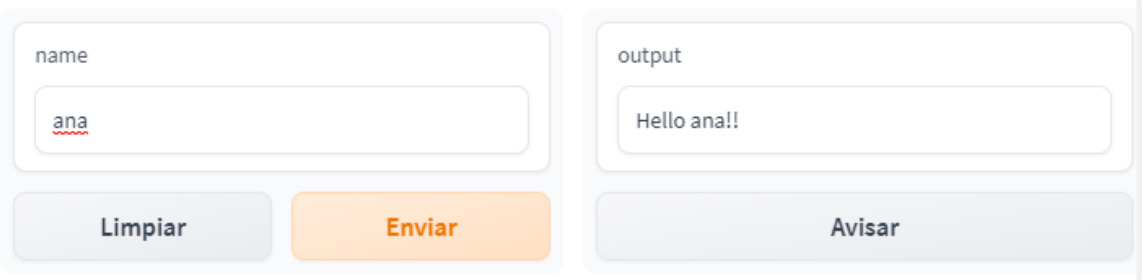
# Gradio Basics

Here you can access Gradio documentation https://gradio.app/getting_started/

```python
import gradio as gr

def greet(name):
    return "Hello " + name + "!!"

demo = gr.Interface(fn=greet, inputs="text", outputs="text")

demo.launch()
```

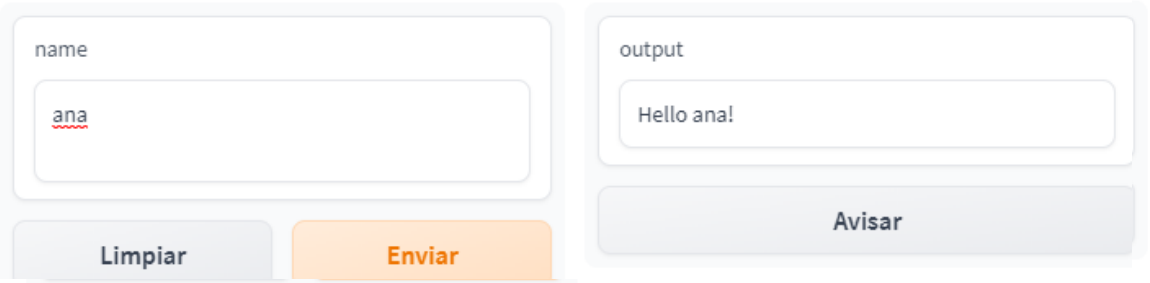
The Interface has three mandatory parameters:

- **fn:** the function to use in the user interface
- **inputs**: which component(s) to use for input, example: "text“, "image“ or "audio“
- **outputs**: which component(s) to use for the output, example: "text“, "image“, "label"

# Gradio Basics

```python
import gradio as gr

def greet(name):
    return "Hello " + name + "!"

demo = gr.Interface(
    fn=greet,
    inputs=gr.Textbox(lines=2, placeholder="Name Here..."),
    outputs="text",
)

demo.launch()
```

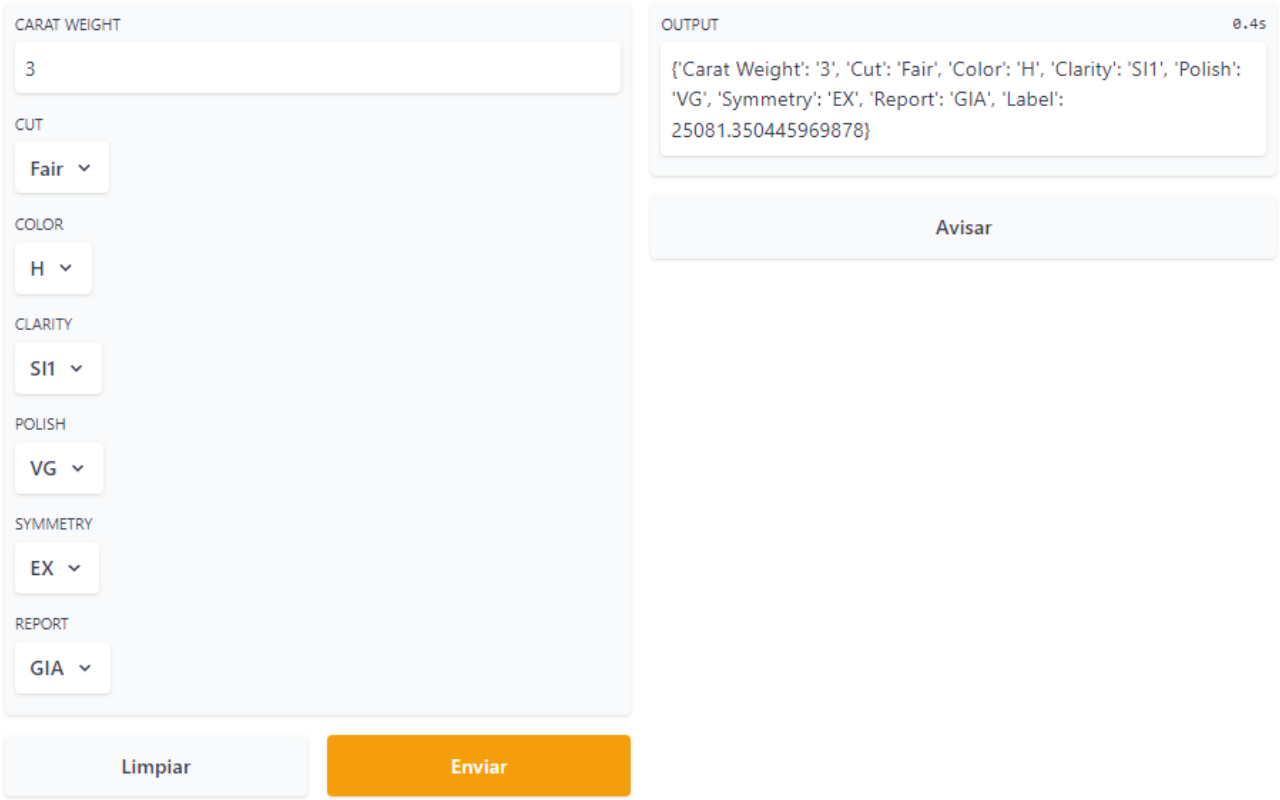



```python
import gradio as gr

def greet(name, is_morning, temperature):
    salutation = "Good morning" if is_morning else "Good evening"
    greeting = "%s %s. It is %s degrees today" % (salutation, name, temperatu
    celsius = (temperature - 32) * 5 / 9
    return greeting, round(celsius, 2)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "checkbox", gr.Slider(0, 100)],
    outputs=["text", "number"],
)
demo.launch()
```

# Create web application with Gradio

There are **libraries** that facilitate the development of applications such as Streamlit or Gradio. **Pycaret** implements a function to generate a basic web application with **Gradio**.

```
# create gradio app
create_app(lr)
```

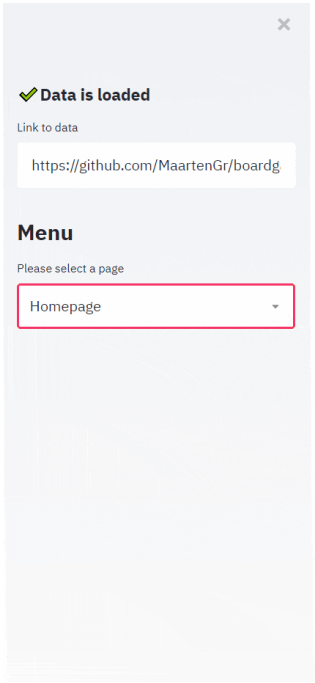


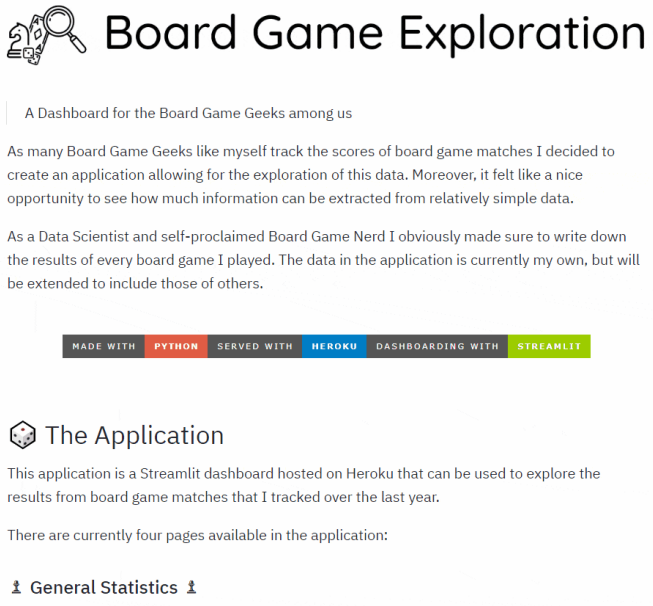
# Streamlit Fundamentals

With Streamlit, developing a **web application** for the ML model has become incredibly easy.

**Advantages** of using Streamlit:

- HTML knowledge not needed!
- Low code
- Widgets are variables
- Caching

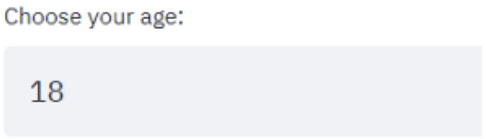


# Widgets

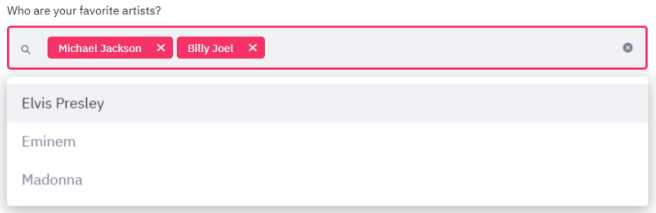
One of the main features of Streamlit is the use of **widgets**. There are many widgets available, including the following:

```
age = streamlit.selectbox("Choose your age: ",
np.arange(18, 66, 1))
```




```
age = streamlit.slider("Choose your age: ", min_value=16,
max_value=66, value=35, step=1)
```


```
artists = st.multiselect("Who are your favorite artists?",
["Michael Jackson", "Elvis Presley",
"Eminem", "Billy Joel", "Madonna"])
```



# Caching and graphics

In many tools data is **reloaded** in each selection. Streamlit can **cache data** and only run it if it hasn't been run before. **Supports** many **visualization libraries** (Matplotlib, Altair, Plotly, Bokeh, etc). Can even load audio and video.

Cache

```
import pandas as pd
import streamlit as st

@st.cache
def load_data():
    df = pd.read_csv("your_data.csv")
    return df

# Will only run once if already cached
df = load_data()
```

Visualization

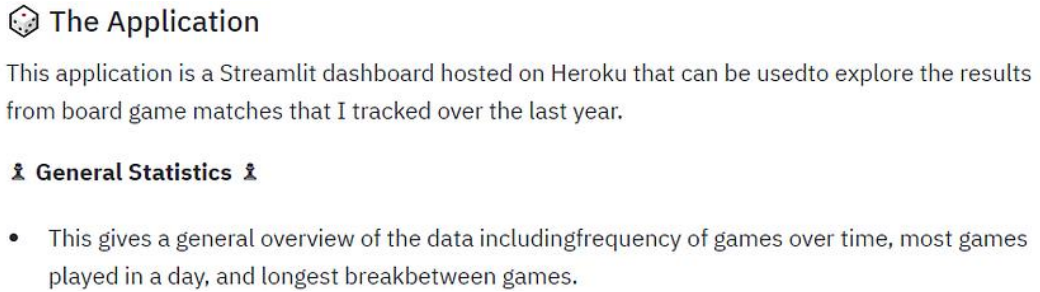
```
import pandas as pd
import numpy as np
import altair as alt
import streamlit as st

df = pd.DataFrame(np.random.randn(200, 3), columns=['a', 'b', 'c'])
c = alt.Chart(df).mark_circle().encode(x='a', y='b', size='c',
                                       color='c')
st.altair_chart(c, width=-1)
```

# Markdown and Write

Streamlit has many options for **Markdown** and customizing the readme. In addition, the **Write function** automatically **adapts** depending on the input data type.

```
import streamlit as st
st.markdown("### 🎲 The Application")
st.markdown("This application is a Streamlit dashboard hosted on Heroku that
            "to explore the results from board game matches that I tracked o
st.markdown("** 🏌 General Statistics 🏌**")
st.markdown("* This gives a general overview of the data including"
            "frequency of games over time, most games played in a day, and l
            "between games.")
```

- write( **string** ): Prints the formatted Markdown string
  write( **data_frame** ): Displays the DataFrame as a table.
- write( **dict** ): display the dictionary in an interactive widget.
- write( **keras** ): Shows a Keras model.
- write( **plotly_fig** ): Shows a Plotly figure.

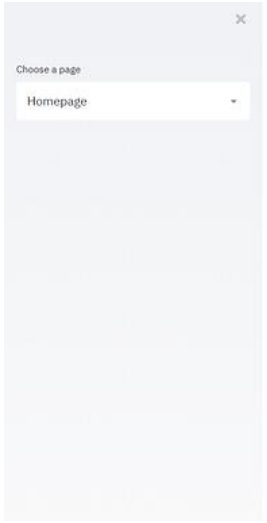


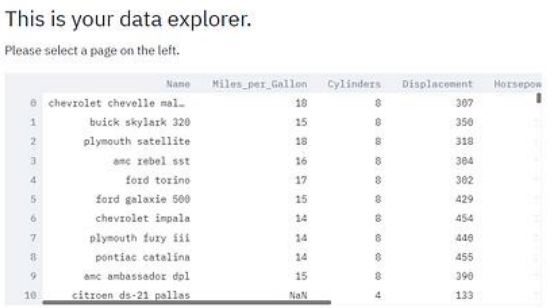
## 🎲 The Application

This application is a Streamlit dashboard hosted on Heroku that can be usedto explore the results from board game matches that I tracked over the last year.

**🏌 General Statistics 🏌**

- This gives a general overview of the data includingfrequency of games over time, most games played in a day, and longest breakbetween games.



This is your data explorer.

Please select a page on the left.

| | Name | Miles_per_Gallon | Cylinders | Displacement | Horsepow |
|---|---|---|---|---|---|
| 0 | chevrolet chevelle mal… | 18 | 8 | 307 | |
| 1 | buick skylark 320 | 15 | 8 | 350 | |
| 2 | plymouth satellite | 18 | 8 | 318 | |
| 3 | amc rebel sst | 16 | 8 | 304 | |
| 4 | ford torino | 17 | 8 | 302 | |
| 5 | ford galaxie 500 | 15 | 8 | 429 | |
| 6 | chevrolet impala | 14 | 8 | 454 | |
| 7 | plymouth fury iii | 14 | 8 | 440 | |
| 8 | pontiac catalina | 14 | 8 | 455 | |
| 9 | amc ambassador dpl | 15 | 8 | 390 | |
| 10 | citroen ds-21 pallas | NaN | 4 | 133 | |

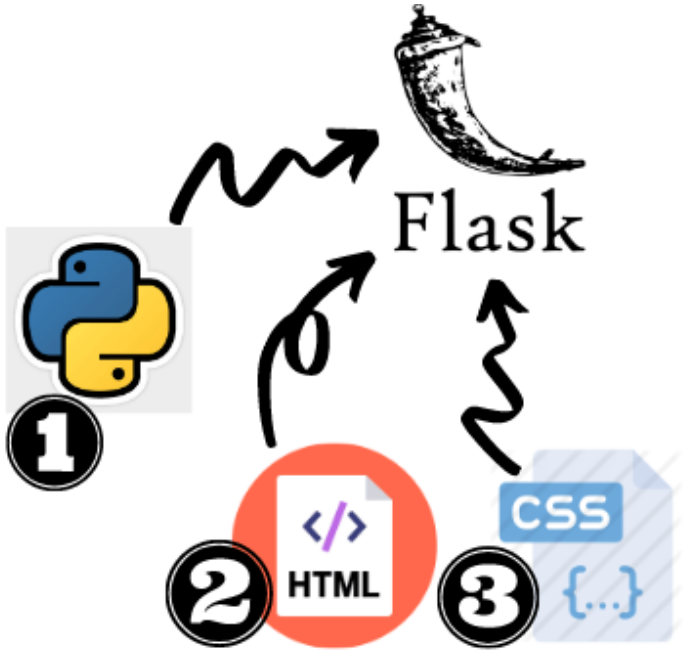
# Application development with Flask

# Flask

**Flask** is a web development framework written in **Python**. Supports multiple **extensions**. Flask is very easy to learn and quick to implement.

Benefits:

- Easy to use
- Flexible
- Allows testing

# Flask Features: Simple Syntax

Flask
web development, one drop at a time

FastAPI

**FLASK**

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
    app.run()
```

**FASTAPI**

```python
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":
    uvicorn.run("hello_world_fastapi:app")
```

# Flask Features: Define Routes

Next we see how we can **define the routes**

**FLASK**

```
from flask import request

@app.route("/", methods=["POST", "GET"])
def home():
    if request.method == "POST":
        return {"Hello": "POST"}

    return {"Hello": "GET"}
```

**FASTAPI**

```
@app.get("/")
def home():
    return {"Hello": "GET"}

@app.post("/")
def home_post():
    return {"Hello": "POST"}
```

# Features of Flask

Next we have **the route and query parameters**

**FLASK**

```
@app.route("/employee/<int:id>/")
def home():
    return {"id": id}
```

```
@app.route("/employee")
def home():
    department = request.args.get("department")
    return {"department": department}
```

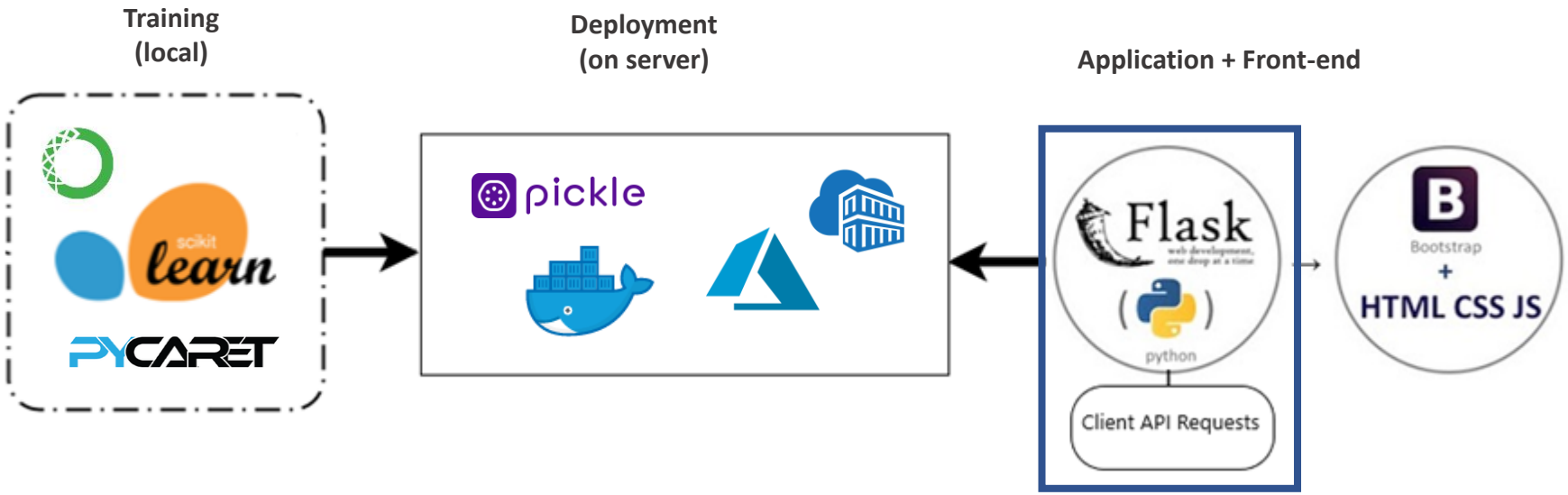
**FASTAPI**

```
@app.get("/employee/{id}")
def home(id: int):
    return {"id": id}
```

```
@app.get("/employee")
def home(department: str):
    return {"department": department}
```

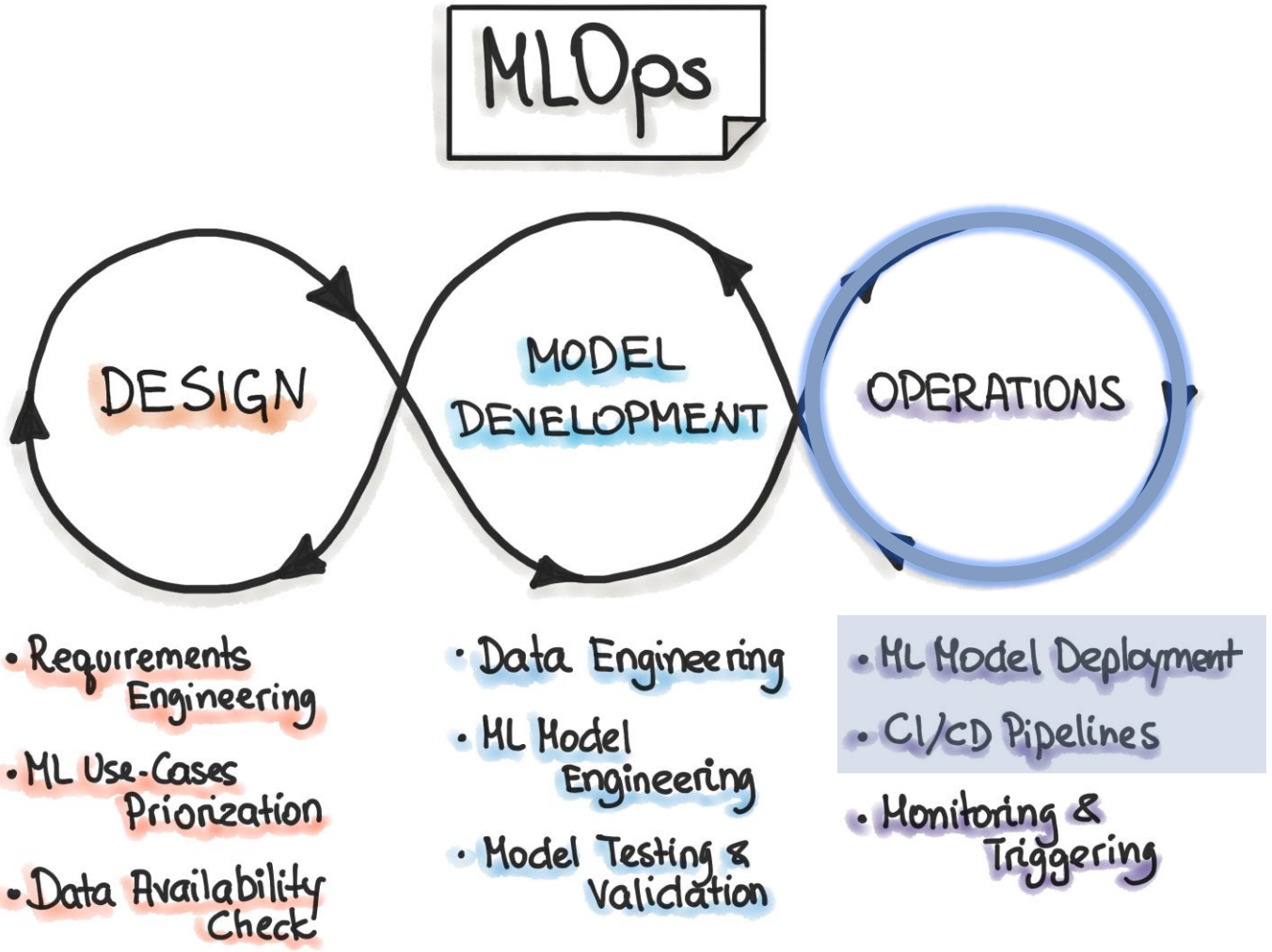
# Create an app with Flask and HTML

We are going to develop a web application based on Flask. You will need a Flask **back-end** and **a front-end** for example in **HTML**. Then that application must be deployed on a **server** (cloud or on-promise) such as Azure, Heroku, etc.



# Containers

# MLOps stages



# Problematica de los entornos

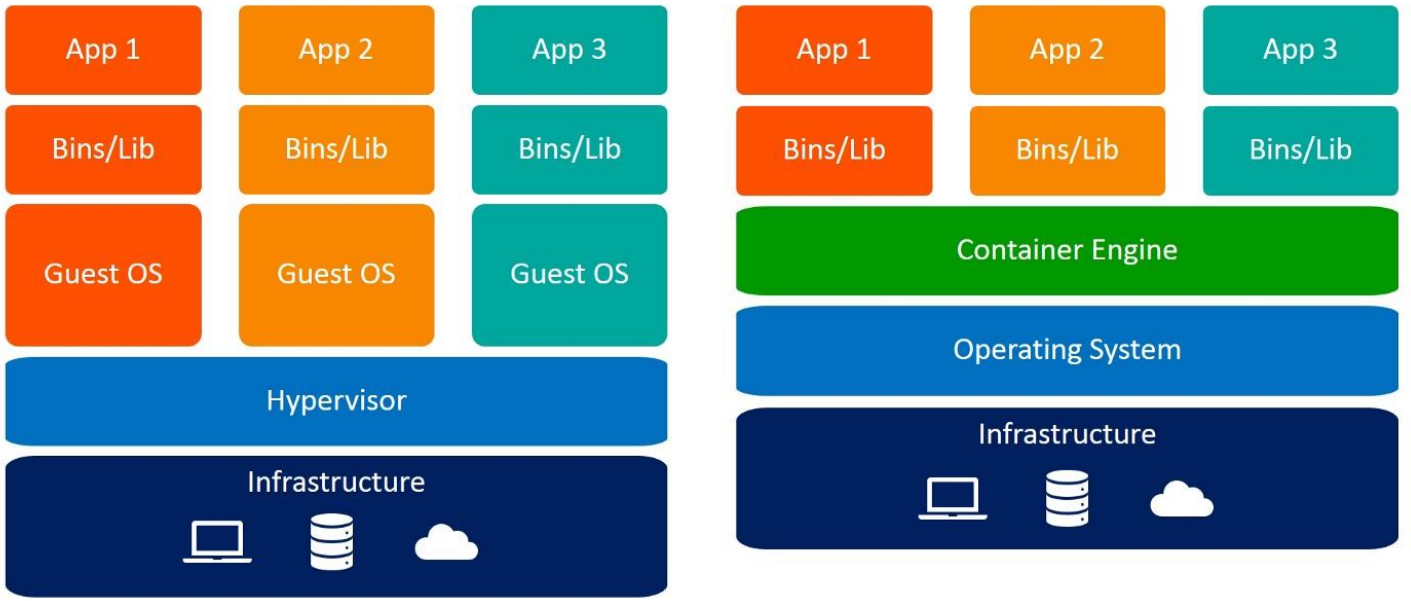
How many times has it happened to you that your code **works fine on your computer**, **but not on others**? The reason: your computer and the rest have different Python environments.

An **environment** includes all the libraries and dependencies used to create an application. If we can transfer that environment into a **container**, the model can be used elsewhere.

# Solutions for environments

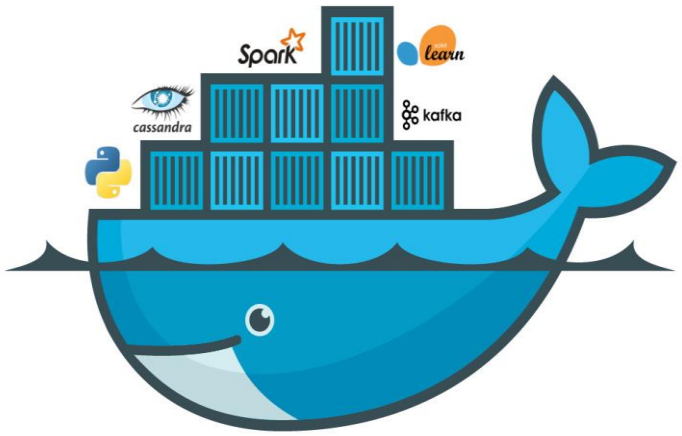
Alternatives to create an isolated environment for our application:

- Have a separate **machine**
- Use **virtual machines**
- **Container**



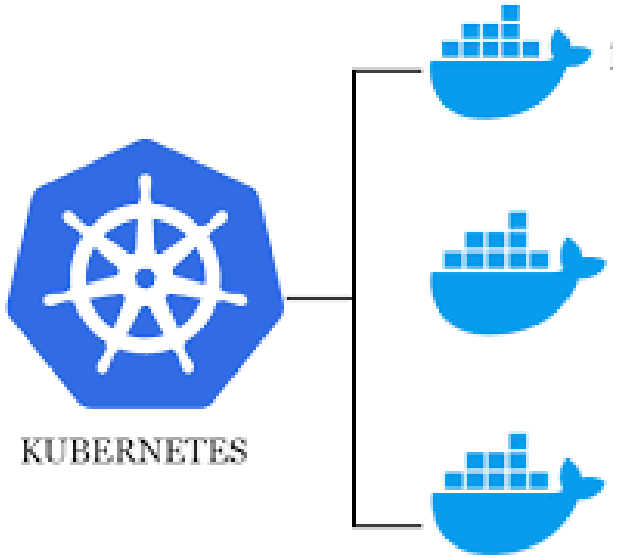
# Docker

**Docker** is a tool to facilitate the creation, deployment and execution of applications using **containers**. These allow you to bundle an application with all of its components and ship it as a single package.

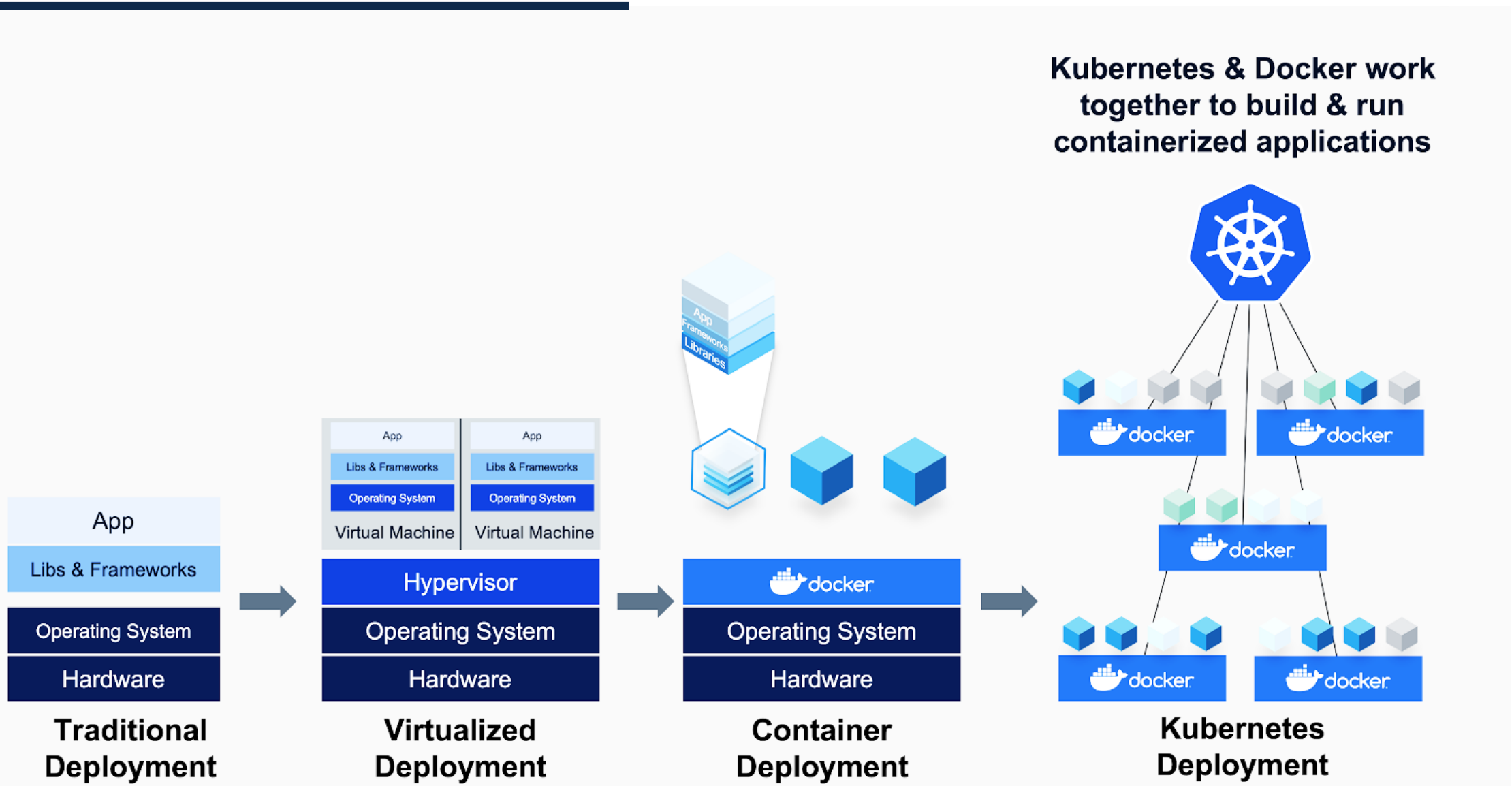



**Dockerfile**

```
FROM python:3.7

RUN pip install virtualenv
ENV VIRTUAL_ENV=/venv
RUN virtualenv venv -p python3
ENV PATH="VIRTUAL_ENV/bin:$PATH"

WORKDIR /app
ADD . /app

# install dependencies
RUN pip install -r requirements.txt

# expose port
EXPOSE 5000

# run application
CMD ["python", "app.py"]
```



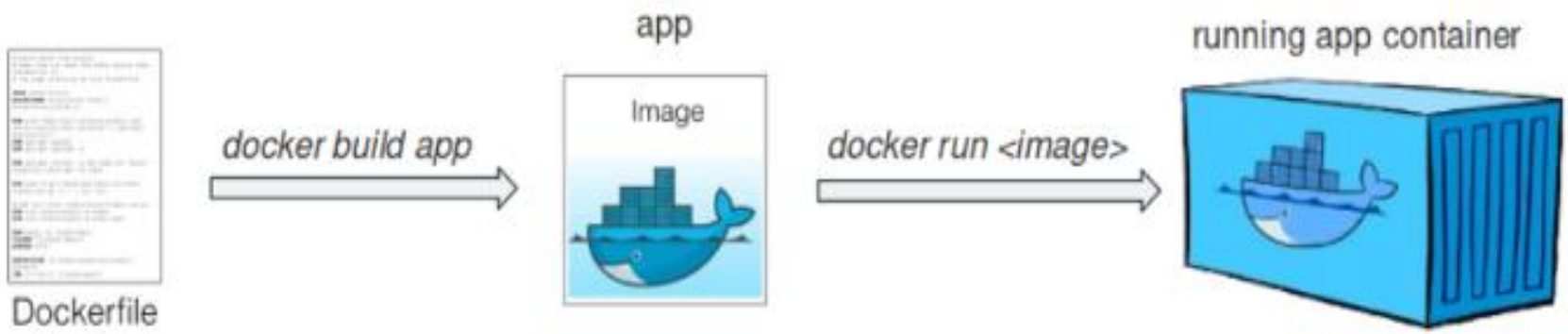
# Summary



# Creating a container

In order to create a container we will have to generate a **Dockerfile** with the necessary libraries for the model.

Then we generate the **Docker Image**, which contains all the information necessary for the execution of the model.



# Container for an API

**Pycaret** allows you to easily **create a container for an API** to consume a Machine Learning model.

1

```
create_docker('my_first_api')
```

```
Writing requirements.txt
Writing Dockerfile
Dockerfile and requirements.txt successfully created.
To build image you have to run --> !docker image build -f "Dockerfile" -t IMAGE_NAME:IMAGE_TAG .
```

2

```
# %load requirements.txt

pycaret
fastapi
uvicorn

```

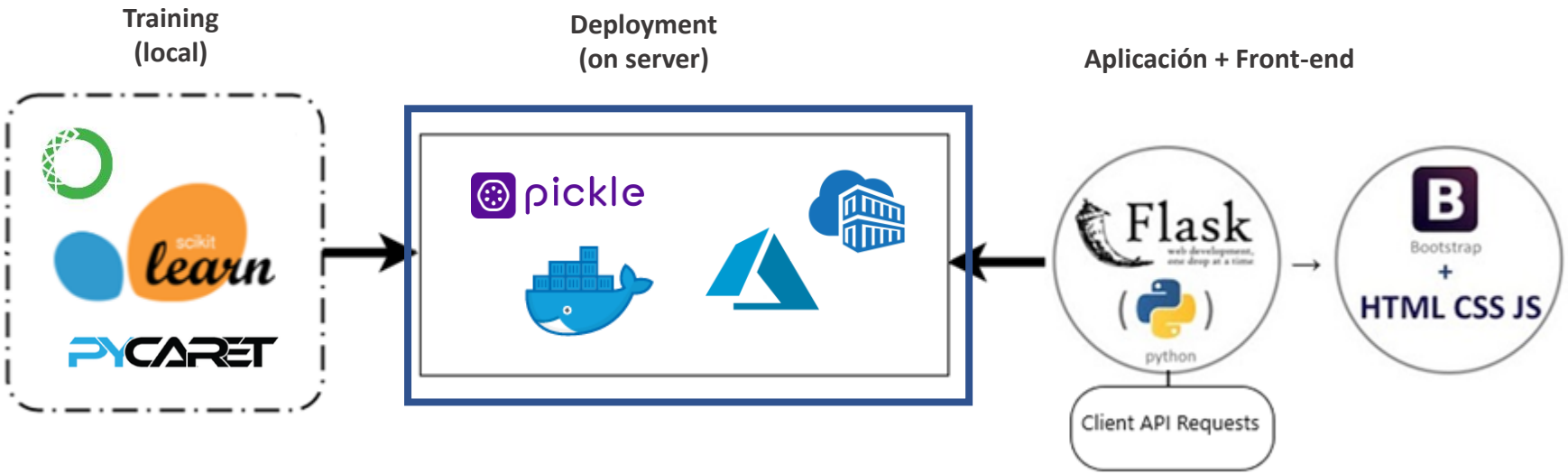
```
# %load Dockerfile


FROM python:3.8-slim

WORKDIR /app

ADD . /app

RUN apt-get update && apt-get install -y libgomp1

RUN pip install -r requirements.txt

EXPOSE 8000

CMD ["python", "my_first_api.py"]

```

3

```
!docker image build -f "Dockerfile" -t my_first_image:latest .
```

# Flask Application Container

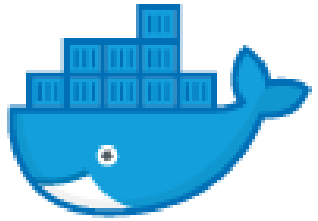
We will generate a **Docker container** for our web application. This application is developed with **Flask and HTML**, and will have a Machine Learning **model** embebed.



# Flask Application Container

We are going to generate a Docker container for our application.

For this we will follow the following steps:



1. Prepare the **environment** cd/path and pip install -r requirements.txt
2. Create the requirementx.txt and the Dockerfile
3. Build the Docker image with **docker build -t pycaret.azurecr.io/pycaret-insurance:latest** .
4. Test running **docker run -d -p 5000:5000 pycaret.azurecr.io/pycaret-insurance**

# ML services with BentoML

# BentoML

Wouldn't it be nice to automate the development of ML services? BentoML can help us with that.

BentoML is an open-source Python library that allows you to create an **ML service in minutes**.

1 Saving and registering the model

```
import bentoml

bentoml.keras.save_model("cnn16", cnn_model)
```

```
retrieved_cnn = bentoml.keras.load_model("cnn16:latest")
```

2 Access to model registry

```
$ bentoml models list

Etiqueta Módulo Tamaño Creación Tiempo Ruta cnn16
: 2uo5fkgxj27exuqj bentoml.keras   5.81 KiB     2022 - 12 - 19 0
: nb5vrfgwfgtjruqj   5.20 KiB     1 -20 bentoml.keras 21 : 36 :
```

# API development

Models usually use arrays, Dataframes, images, etc. and not APIs JSON format. BentoML's **bentoml.io** module supports and validates different formats, from Dataframes to Binary data.

3

```python
import bentoml
from bentoml.io import NumpyNdarray


runner = bentoml.sklearn.get("model_name:latest").to_runner()

svc = bentoml.Service("classifier", runners=[runner])

# The important part
@svc.api(input=NumpyNdarray(), output=NumpyNdarray())
def classify(input_series: np.ndarray) -> np.ndarray:

    result = runner.predict.run(input_series)
    return result
```

4

```
@svc.api(input=NumpyNdarray(shape=(-1,15), enforce_shape=True),
output=NumpyNdarray())
```

# Model serving and containerization

BentoML makes it easy to build a **Docker image**, without Docker knowledge. You must create the **bentofile.yaml** with the template. **Bentoctl** module allows deploying the containerized API in different cloud environments easily.

5

```
service: "service.py:service_name"
include:
- "*.py"
python:
 packages:
  - scikit_learn
  - numpy
  - tensorflow
```

```
$ bentoml build
```

```
$ bentoml containerize model_name:latest
```

6

```
$ pip install bentoctl terraform
$ bentoctl operator install aws-sagemaker
$ export AWS_ACCESS_KEY_ID=REPLACE_WITH_YOUR_ACCESS_KEY
$ export AWS_SECRET_ACCESS_KEY=REPLACE_WITH_YOUR_SECRET_KEY
$ bentoctl init
$ bentoctl build -b model_name:latest -f deployment_config.yaml
$ terraform init
$ terraform apply -var-file=bentoctl.tfvars -auto-approve
```

# BentoML Advantages

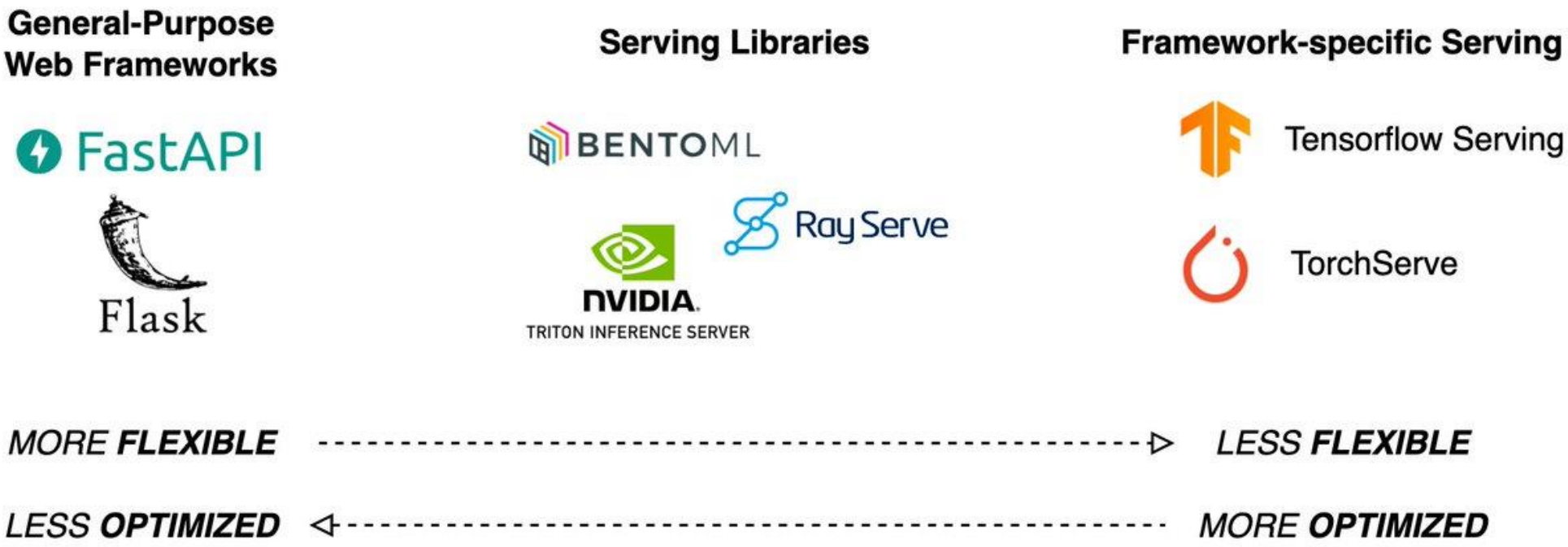
Advantages:

- **Starlette**: built on the same robust framework of ASGI (quick and easy) web applications
- Automatic documentation with **Swagger UI**
- **Asynchronous Requests**: Asynchronous requests to handle multiple requests simultaneously
- Easily save and load models
- Registration and **versioning of models**
- Input data **validation**
- Validation of **multiple formats**
- Automated **service** generation
- Automatic **Docker** Image Generation
- Facilitates deployment in different **Cloud** environments
- **GPU** service

Disadvantages:

- Bad integration with Conda

# Different tools



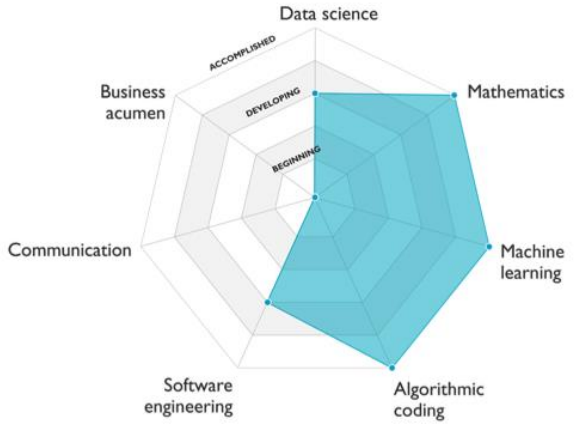
# Machine Learning in Cloud

# Importance of the Cloud

ML has been **out of reach** for most **companies** due to the high level of specialization it requires, high implementation costs and difficulties to scale. **ML in the Cloud** has a lot of benefits, such as:

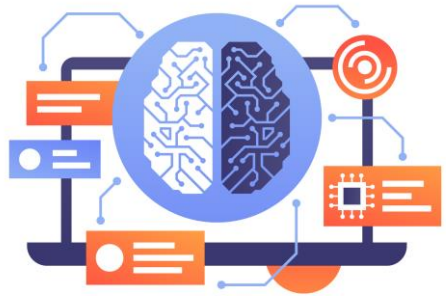
**Cost efficiency**


**Less specialization**




**Scalability**

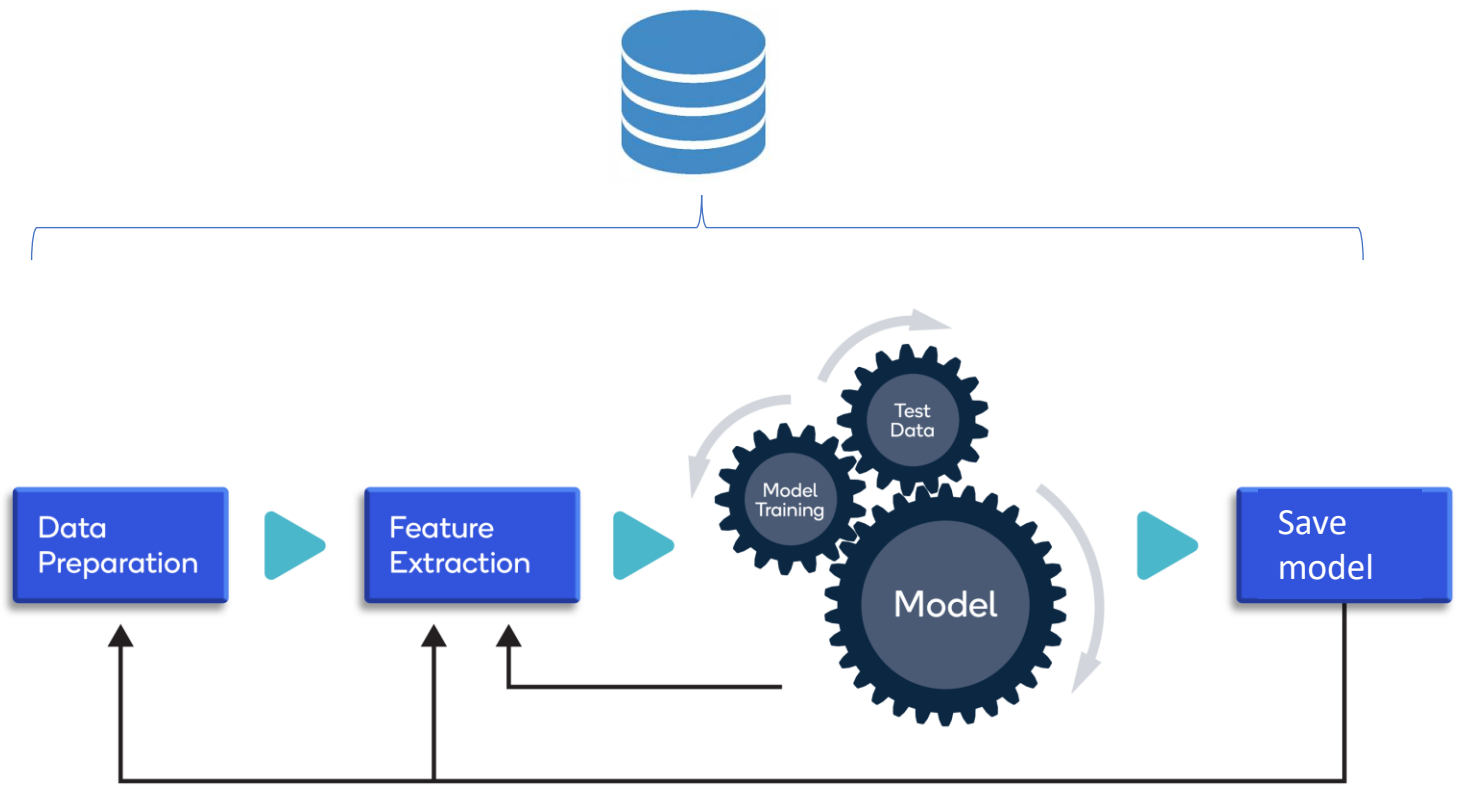
**Experimentation**



# Architecture

1. Train the model

Historical data




New data

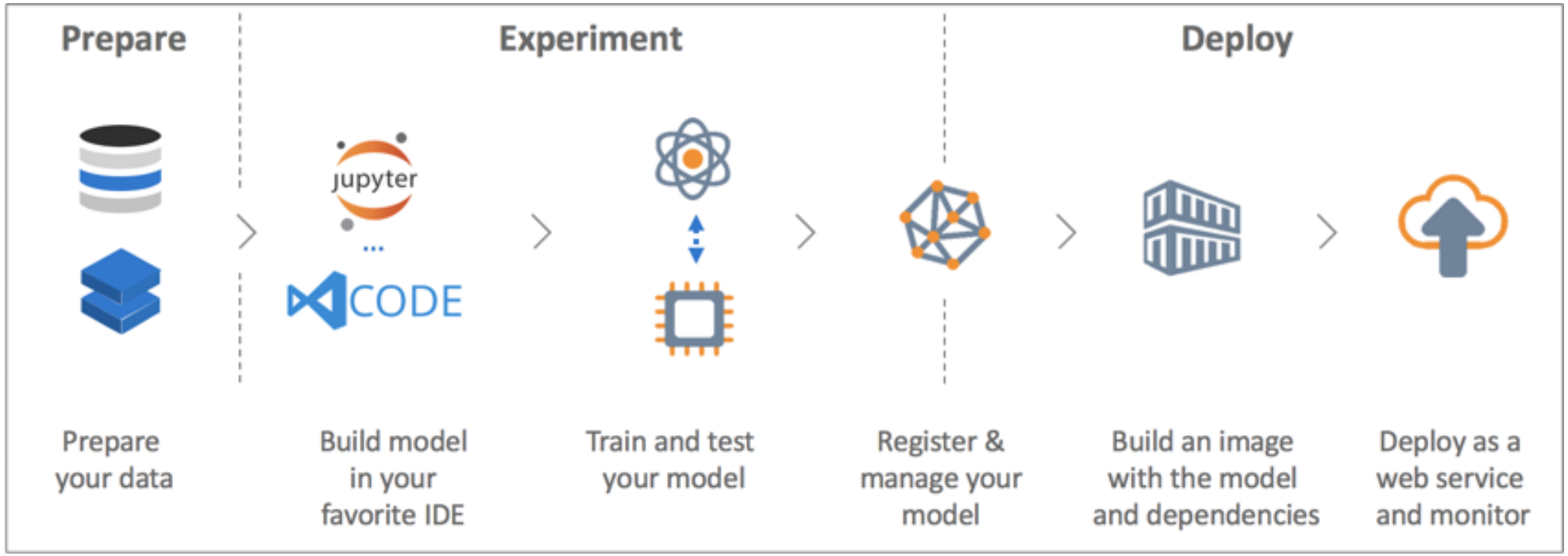
2. Consume the model


New Data with Predictions

3. Get predictions

# Deploy to Azure Container

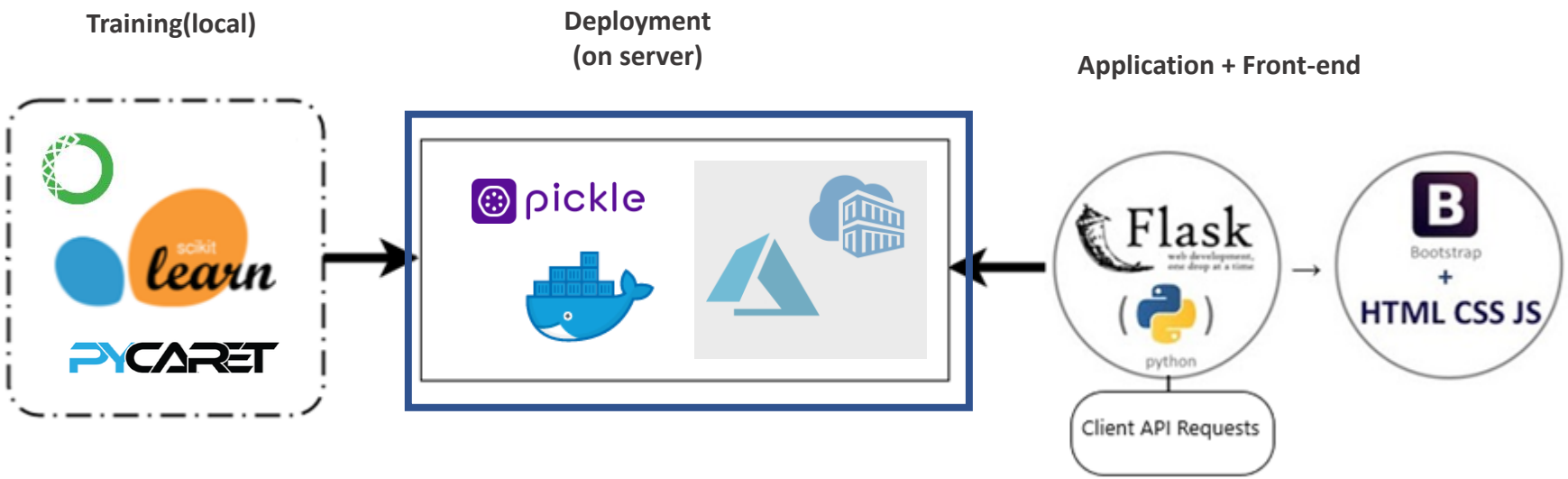
We can deploy our model in the cloud using **Docker Containers.** In this example, we will test deploying the model to Azure using the **Azure Container Registry.**



# Container deployment in Azure

We are going to **deploy the container** of the application that we had developed from Flask and HTML in Azure.

To do this, we will use the **Azure Container Registry** to upload the Docker image.

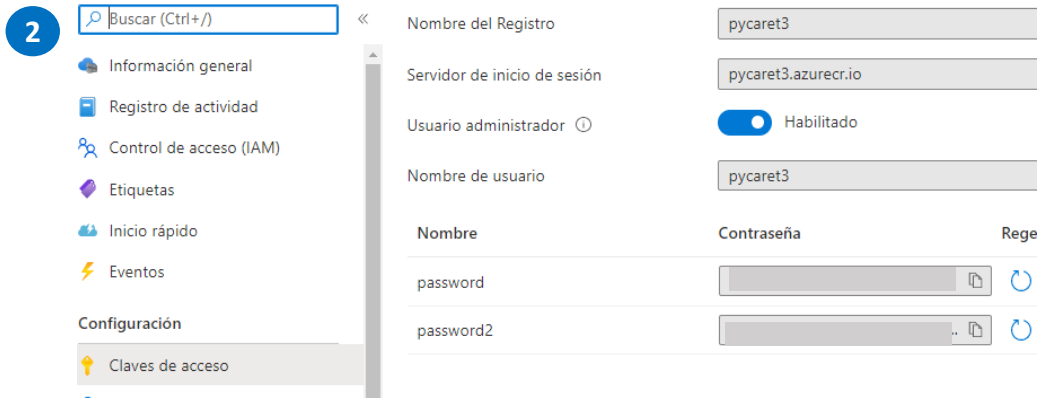


# Steps for deployment to Azure Container Registry

1 

Container Registry
Crear | Documentos | MS Learn

**Create an Azure Container Registry service**. You will create the service with pycaret3 subscription name

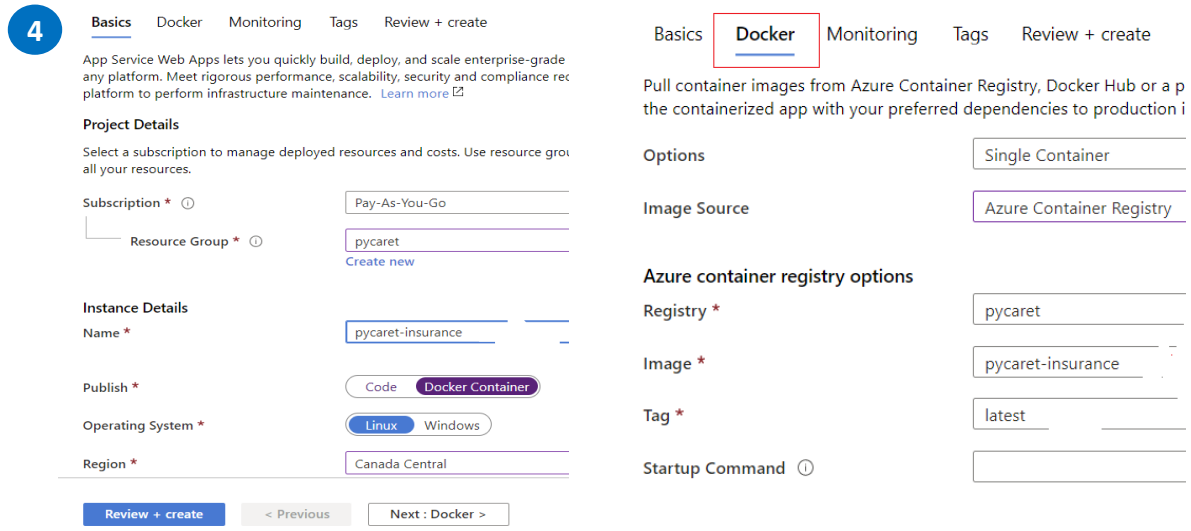
2 


**Login**. Log in Azure Container registry from Anaconda prompt with command "*docker login pycaret.azurecr.io*" . Enter name and credentials from"access codes".

3 

Aplicación web
Crear | Documentos | MS Learn

**Push to Azure container**. To do this, you will enter the command "*docker push pycaret.azurecr.io/pycaret-insurance:latest*"

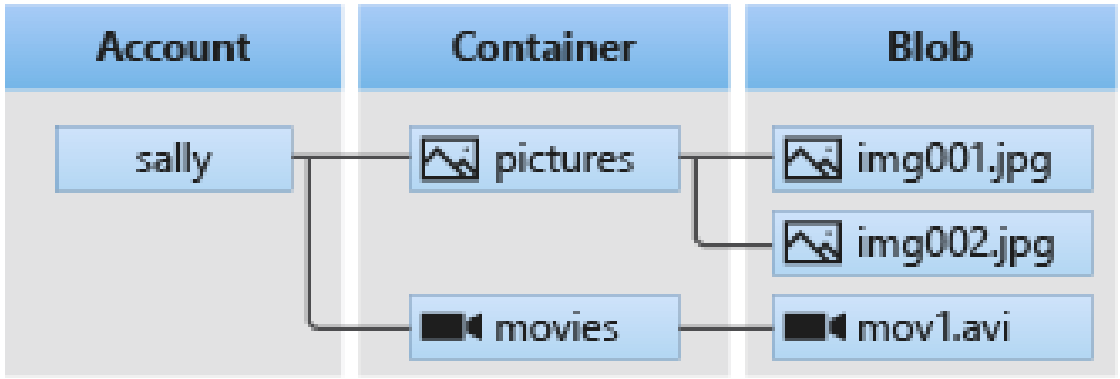
4 


**Web application service.** Create a web application service with this configuration:Publish = Docker Container, in Docker->Image Source = Azure Container Registry, Image = pycaret, Tag = latest

# Deploying models to a Blob storage

Another way to **deploy models** in the Azure Cloud is by saving them as **binary file** in **Azure Blob Storage.** To **consume this model**, we can download the model from the Blob Storage and use it to obtain predictions with new data.

Microsoft Azure
Blob Storage



# Azure SDKs

**Azure SDKs** are **collections and functions** created to facilitate the use of Azure services with **different languages**.

They are designed to be consistent, accessible and reliable.

| .NET | Java | JavaScript/TypeScript | Python |
|---|---|---|---|
| Obtención del SDK | Obtención del SDK | Obtención del SDK | Obtención del SDK |
| Documentación | Documentación | Documentación | Documentación |
| GitHub | GitHub | GitHub | GitHub |

| Ir | C++ | C | Android |
|---|---|---|---|
| Obtención del SDK | GitHub | GitHub | GitHub |
| Documentación | | | |
| GitHub | | | |

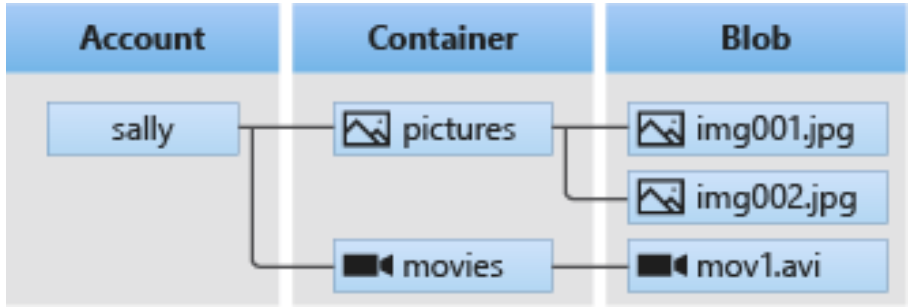
| iOS |
|---|
| GitHub |

https://azure.microsoft.com/es-es/downloads/

# Blob management with Python SDKs

**Blobs** are objects that can contain large amounts of unstructured data (text or binary data, images, documents, or files). There are the following Python classes to interact with these resources:

- *BlobServiceClient*:
- allows you to manipulate Azure Storage resources and containers.
- *ContainerClien*t: allows you to manipulate Azure Storage containers and their blobs.
- *BlobClient* : allows you to manipulate blobs from Azure Storage.

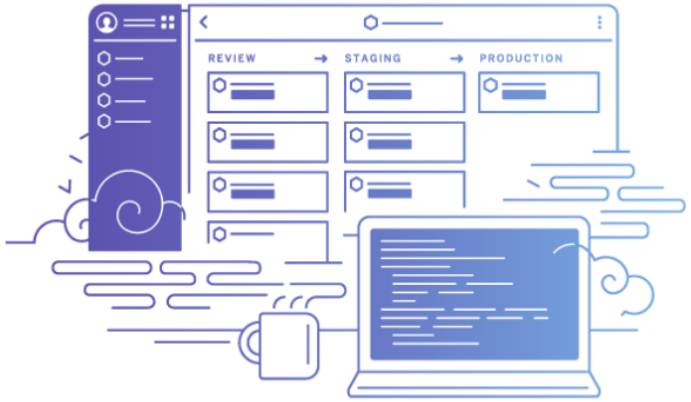



**Link**: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?toc=%2Fpython%2Fazure%2FTOC.json&tabs=environment-variable-windows

# Deploymento to Heroku

# Heroku

**Heroku** is a cloud service platform. Easy to use. Focused on supporting customer-centric applications. Heroku manages hardware and servers.




DEVELOPERS

## Focus on your apps

Invest in apps, not ops. Heroku handles the hard stuff — patching and upgrading, 24/7 ops and security, build systems, failovers, and more — so your developers can stay focused on building great apps.

Sign Up

Explore the Heroku Platform

OFFICIALLY SUPPORTED LANGUAGES

Node.js | Ruby | Java | PHP | Python | Go | Scala | Clojure

# Necessary files

**To deploy the service** on Heroku **three files** are needed in the code repository:

### requirements.txt

2 lines (2 sloc) | 25 Bytes

```
1  pycaret==1.0.0
2  streamlit
```

### Procfile

1 lines (1 sloc) | 40 Bytes

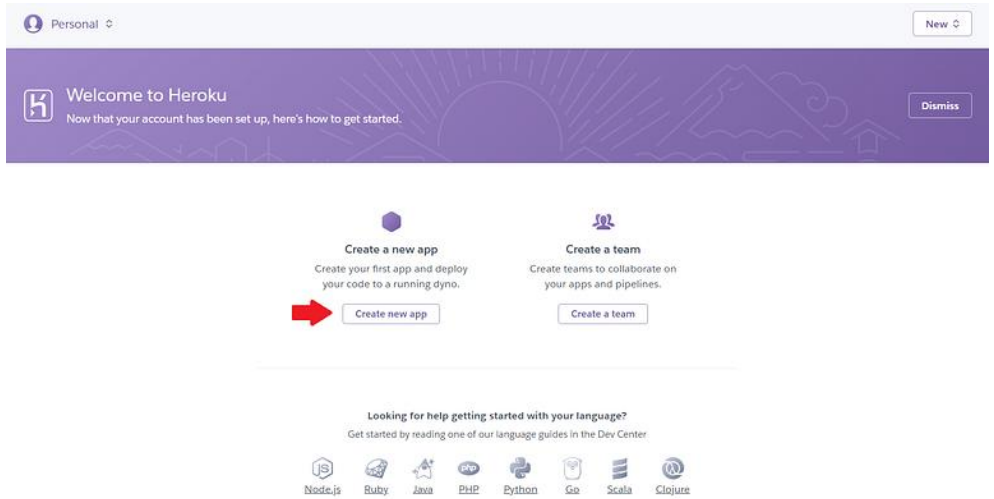
```
1  web: sh setup.sh && streamlit run app.py
```

### setup.sh

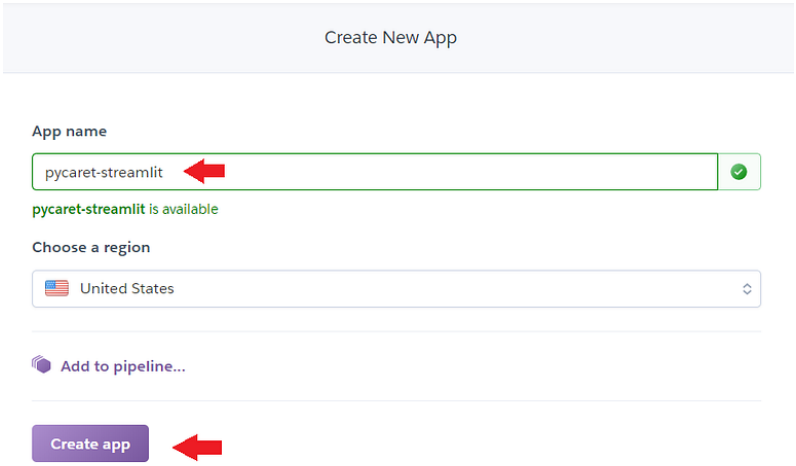
11 lines (11 sloc) | 219 Bytes

```
1  mkdir -p ~/.streamlit/
2  echo "\
3  [general]\n\
4  email = \"your-email@domain.com\"\n\
5  " > ~/.streamlit/credentials.toml
6  echo "\
7  [server]\n\
8  headless = true\n\
9  enableCORS=false\n\
10 port = $PORT\n\
11 " > ~/.streamlit/config.toml
```

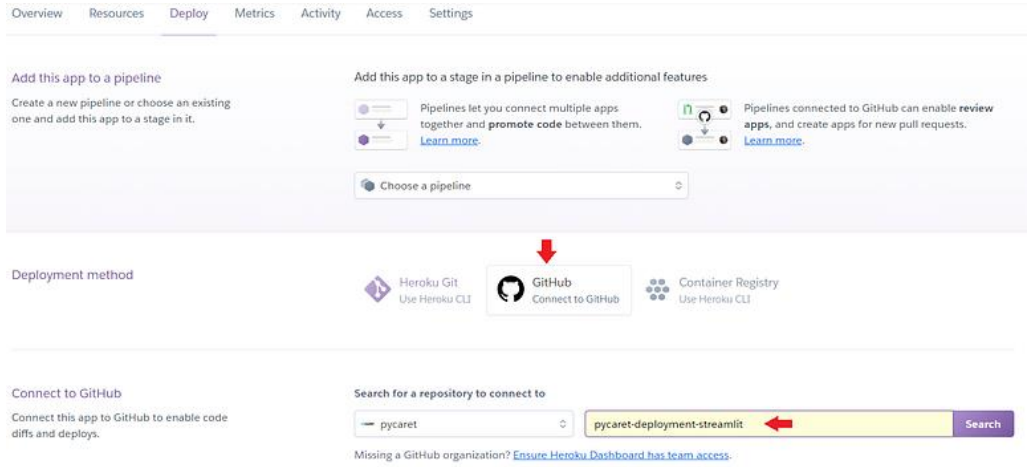
## 1 Sign up and click 'Create new app'



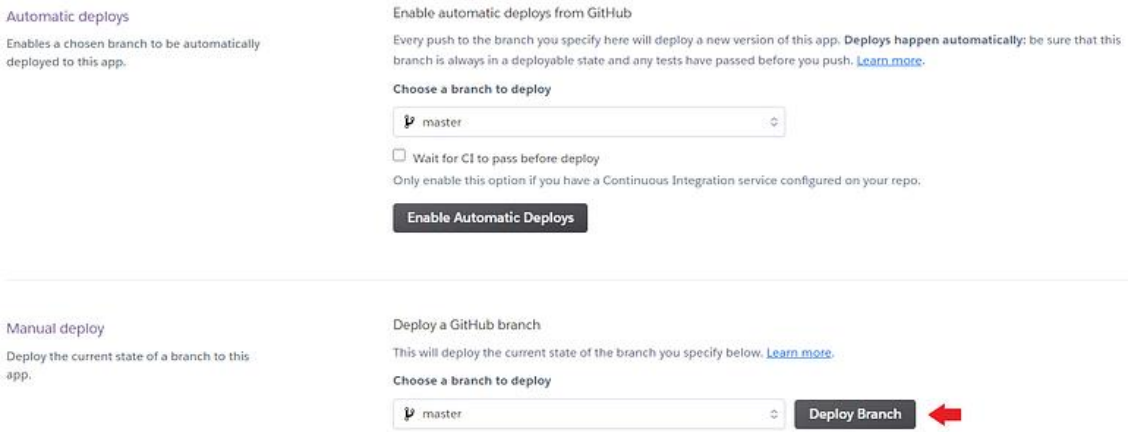

## 2 Add the app name and region




## 3 Connect to the GitHub repository

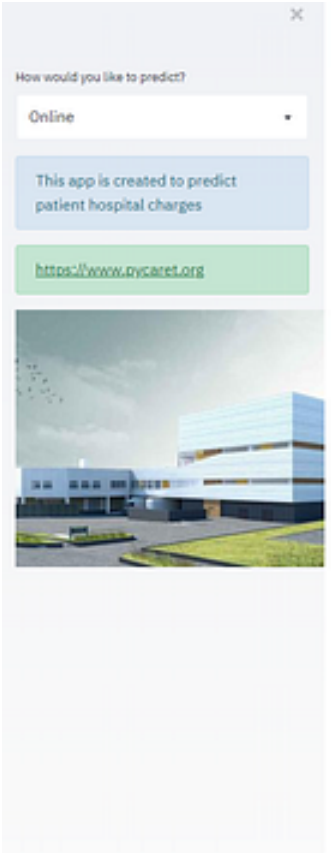


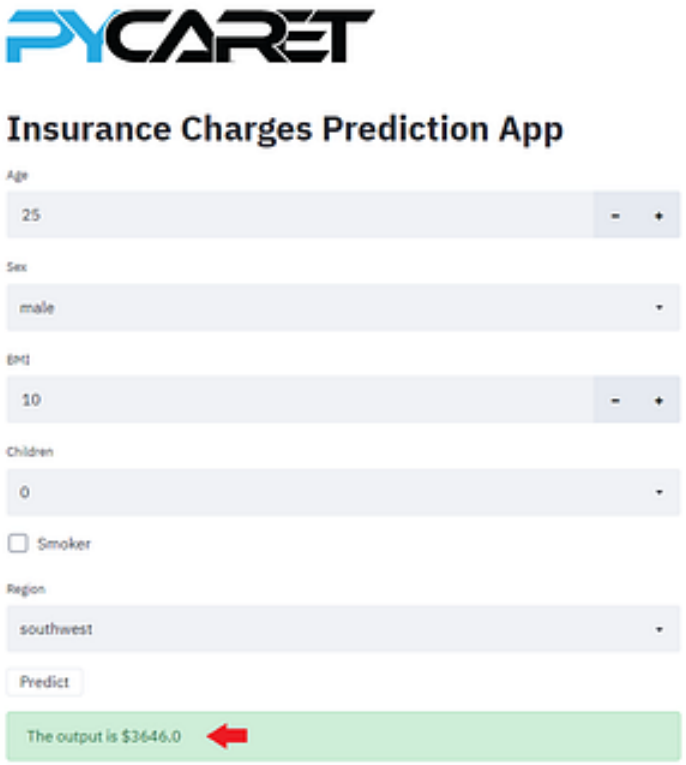

## 4 Deploy the branch and deploy



# Heroku project

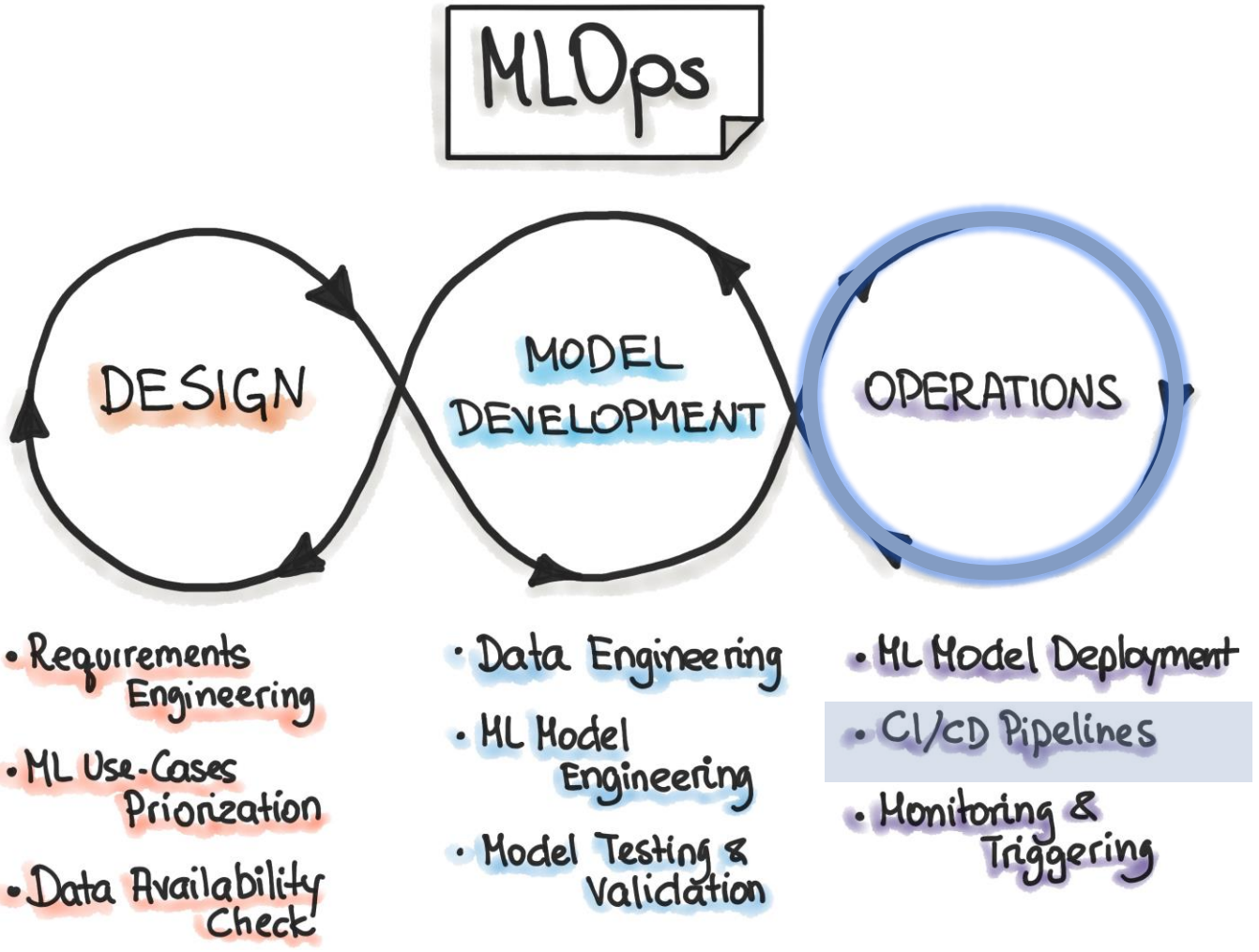
We are going to use Heroku to deploy the **healthcare charge prediction web** application developed with Pycaret and Streamlit.



# CI/CD

# MLOps Stages

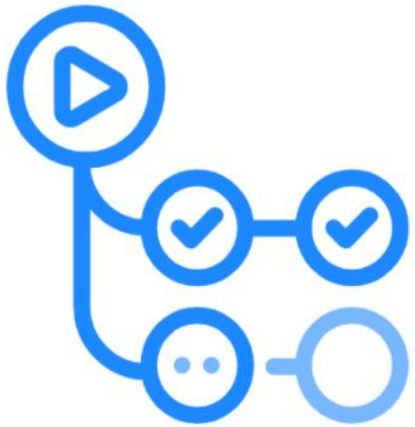


# For what?

Imagine that your company is creating a **service powered by ML**. Once you find a better model, how do you make sure the service **doesn't break** when you implement the **new model**?

# Introduction to Github Actions

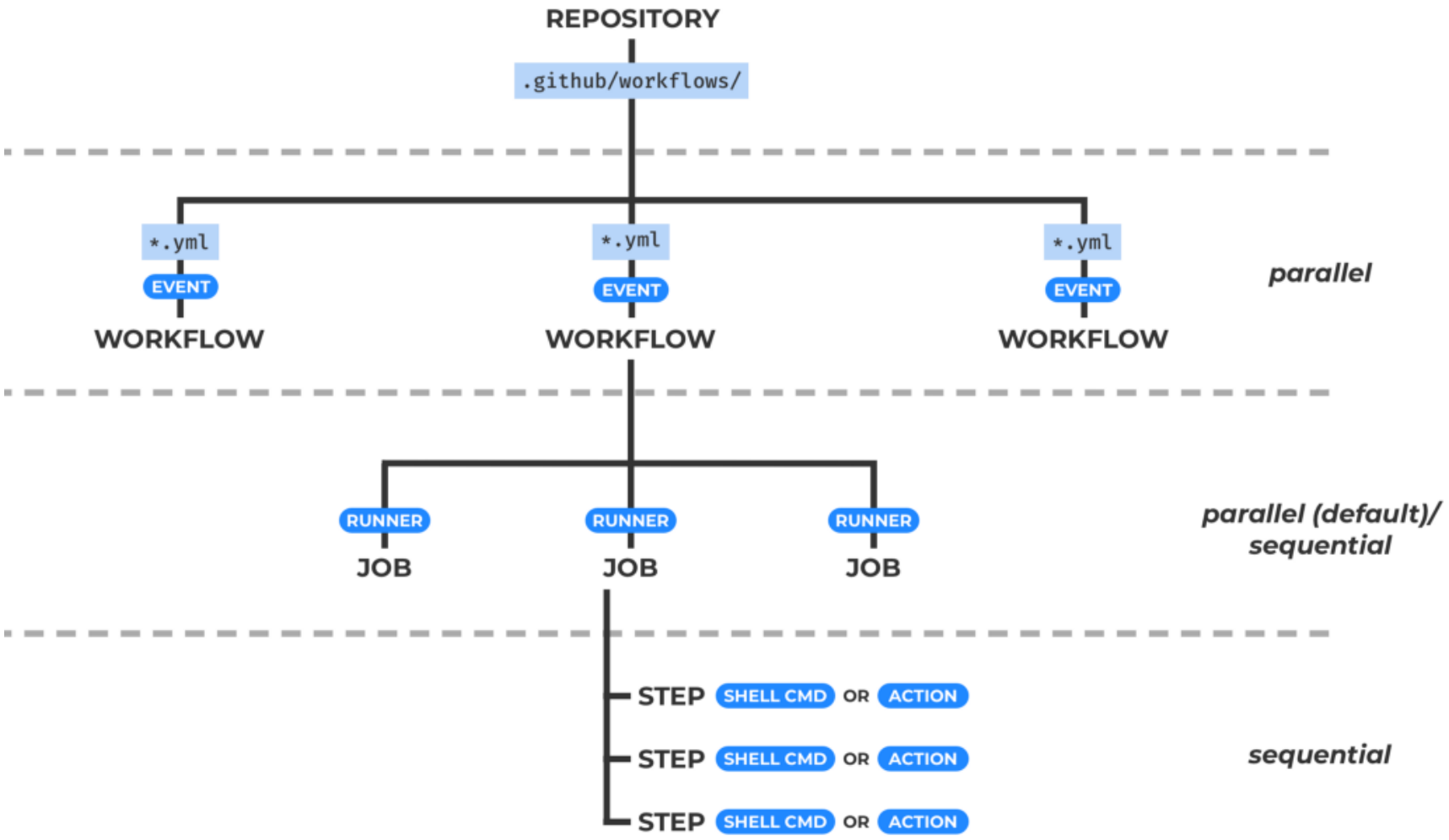
**Github Actions** is a platform that **automates** the creation, testing, and **deployment** of software. It also allows you to run code after a specific **event**.



GitHub Actions

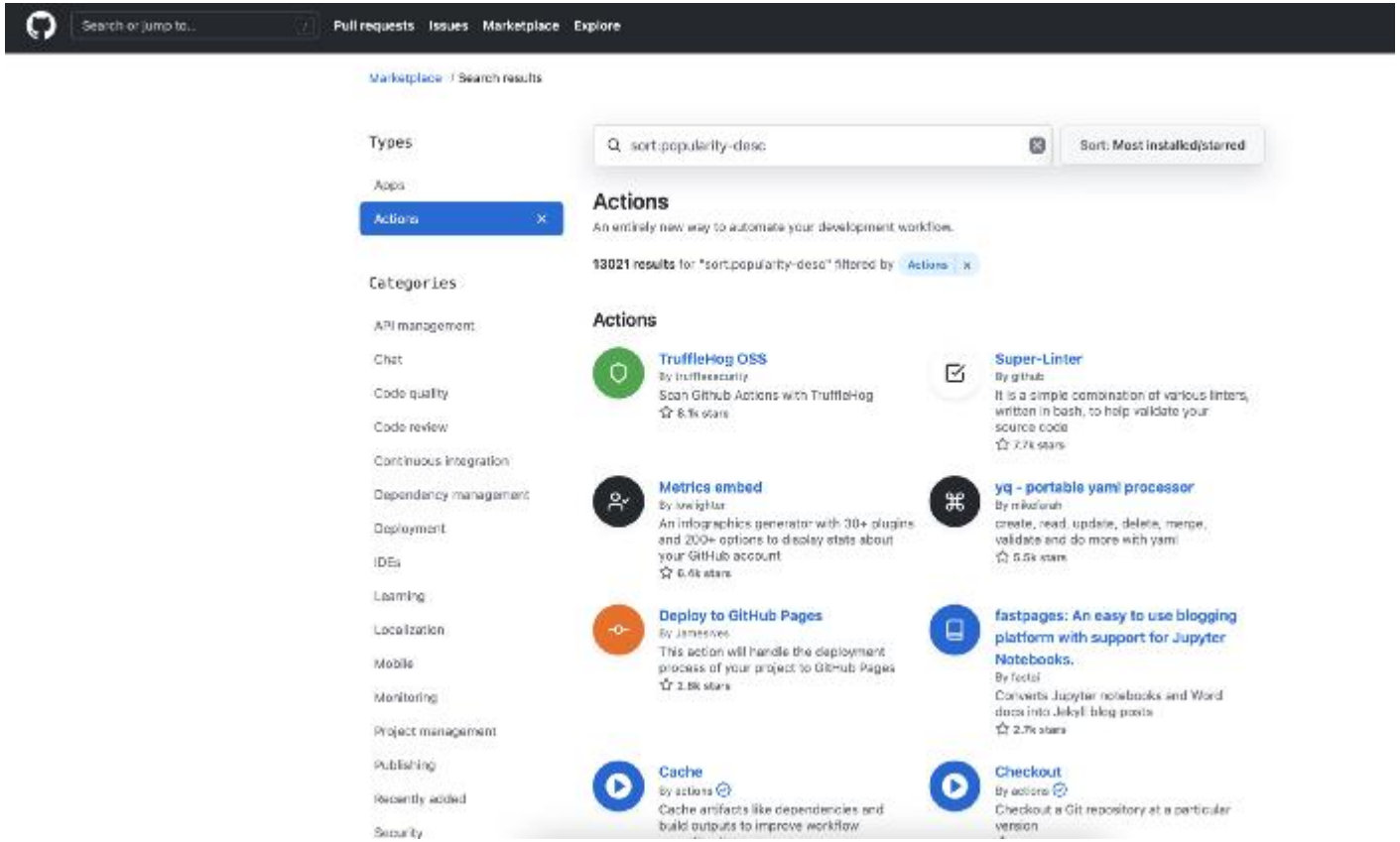
# Github Actions Components

Github Actions has different **components**



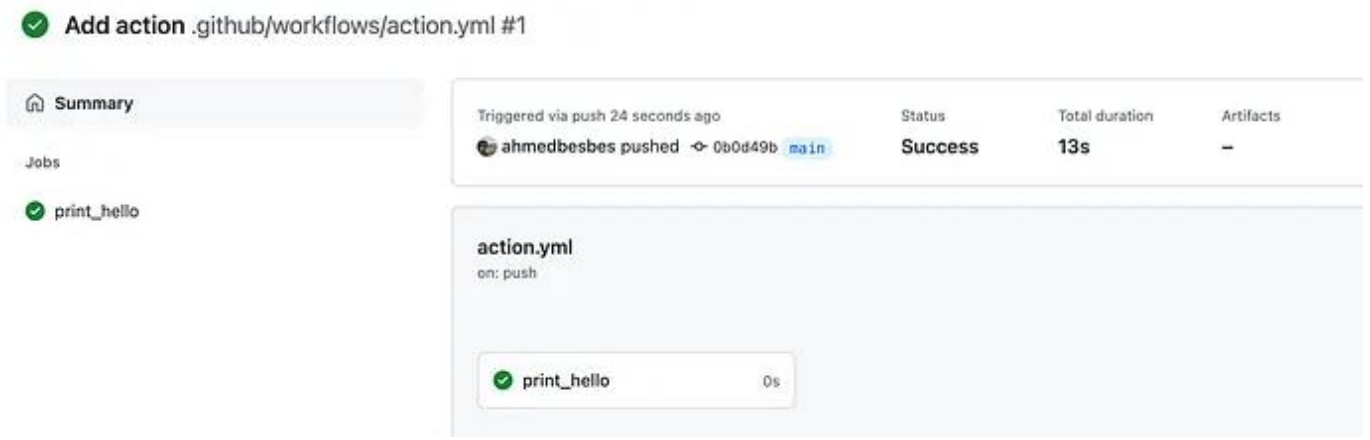
# Github Actions Marketplace

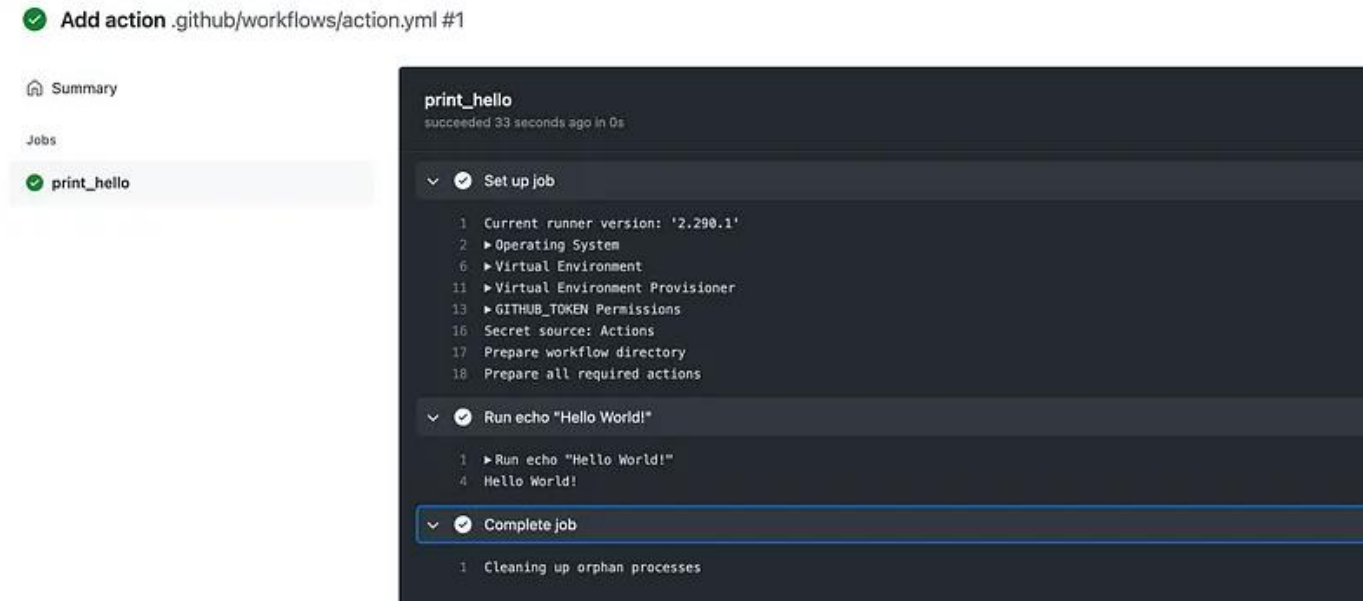
You can create your own GitHub Actions or reuse some **open-source** ones from the **Github marketplace**.



# Basic workflow

```yaml
on:
  push:
    branches:
      - main

jobs:
  print_hello:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Hello World!"
```



# Why Github Actions?

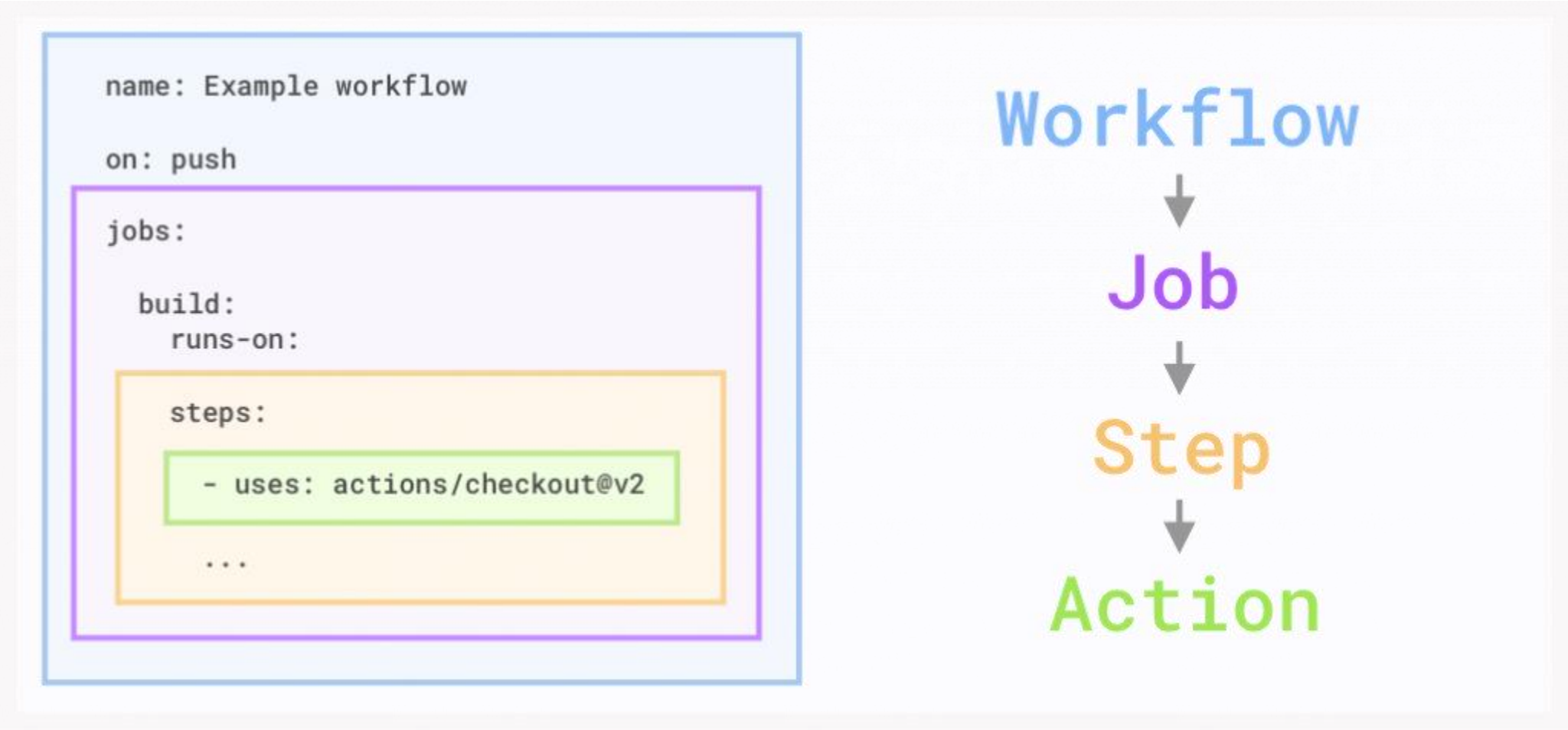
### Create a Docker image

```yaml
name: Docker Image CI

on:
  push:
    branches: [ $default-branch ]
  pull_request:
    branches: [ $default-branch ]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag my-image-name:$(date +%s)
```

### Run unit tests and code quality checks

```yaml
on:
  push:
    branches: [ $default-branch ]
  pull_request:
    branches: [ $default-branch ]

permissions:
  contents: read

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: "3.10"
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
    - name: Lint with flake8
      run: |
        # stop the build if there are Python syntax errors or undefined names
        flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
        # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
        flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
    - name: Test with pytest
      run: |
        pytest
```

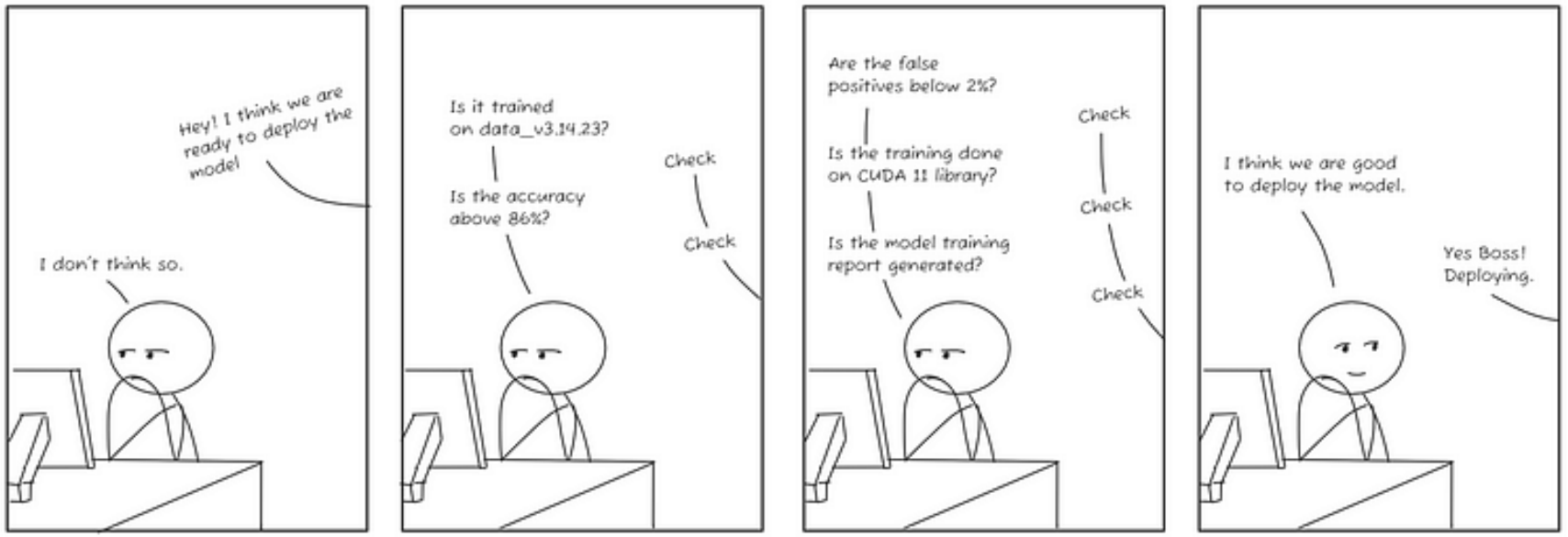
# Example


name: Example workflow
on: push
jobs:
build:
runs-on:
steps:
- uses: actions/checkout@v2
...
Workflow
Job
Step
Action

# Why would we need CML?

To take a model to production we would need multiple verifications, such as the following:

- Can we reproduce the model?
- Are we using the right version of the dataset?
- Are the performance metrics verifiable?
- etc

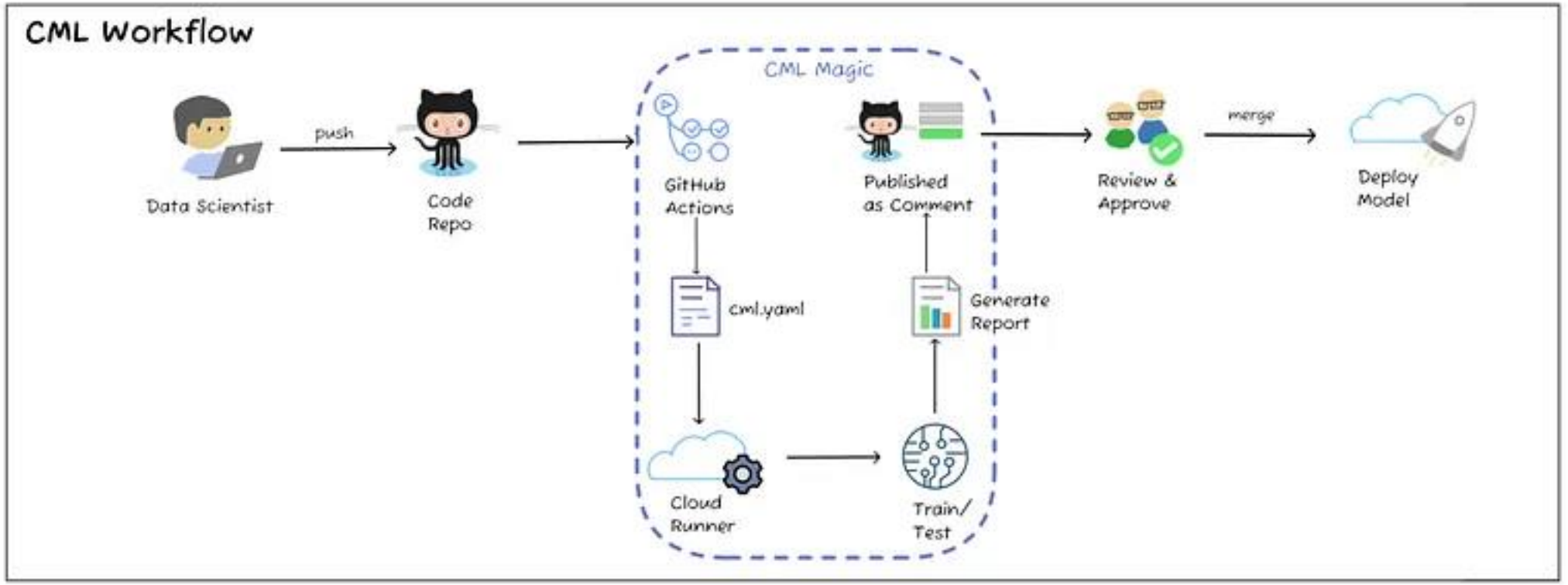


# What is CML

**CML** improves **deployment to production** of ML models, introducing a continuous workflow for provisioning cloud instances, training models on them, collecting metrics, evaluating performance, and publishing reports.

CML

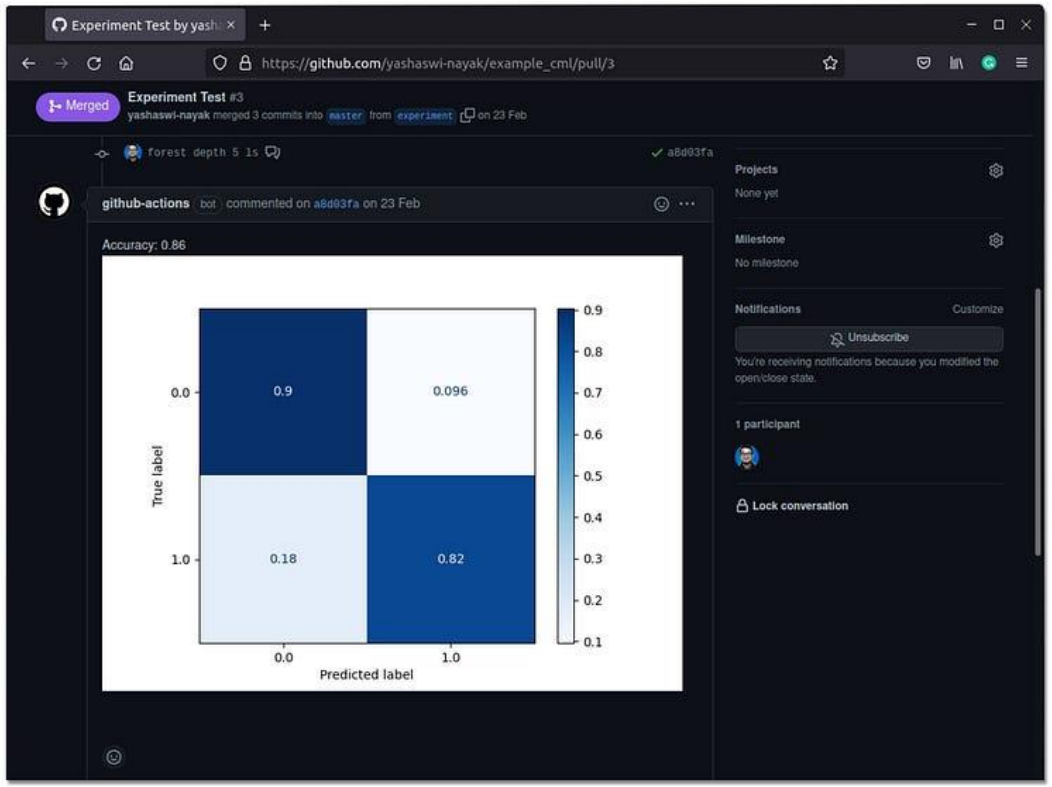
# How does CML work?



# CML Usage Example

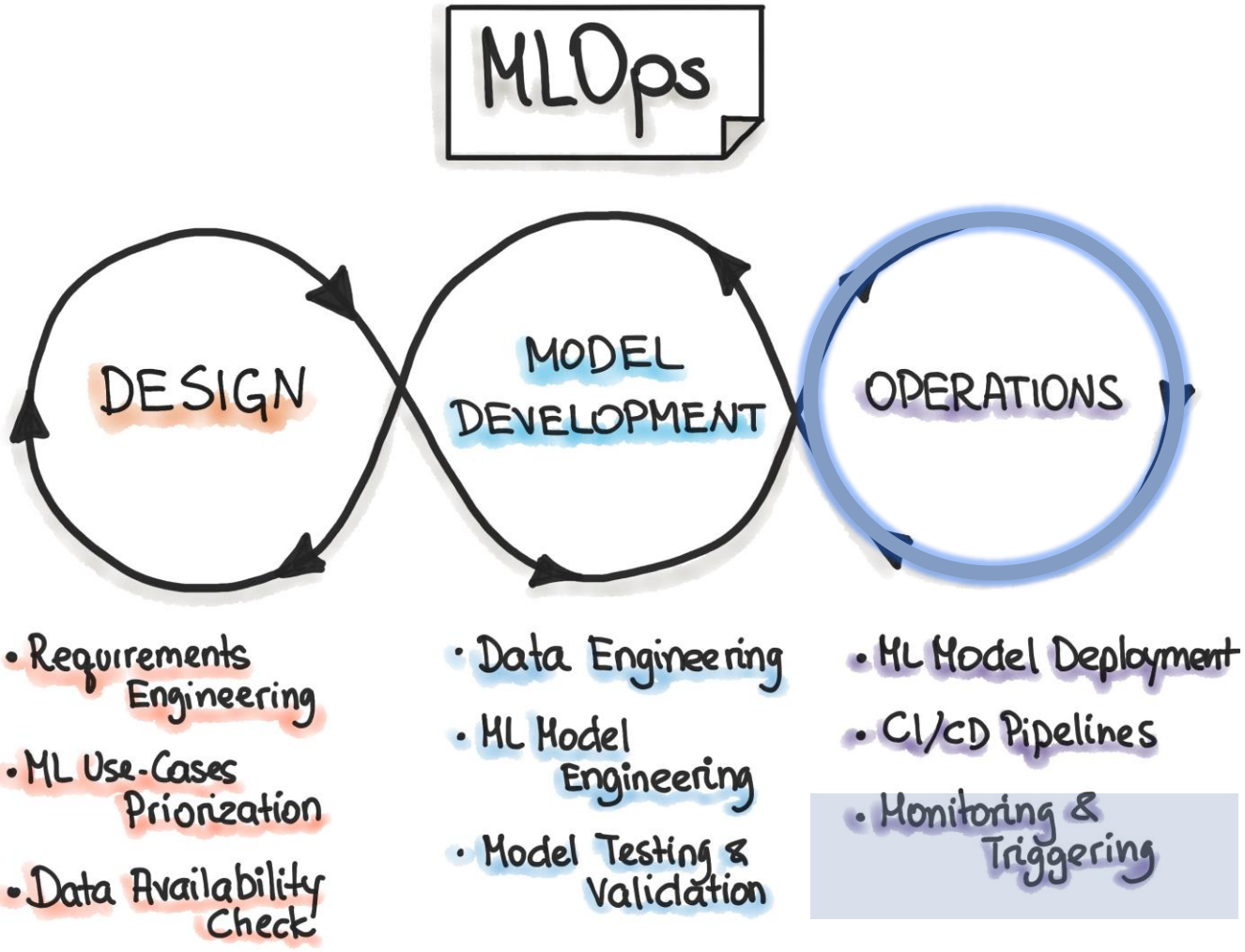
```yaml
name: model-training
on: [push]
jobs:
  train-model:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
      - uses: iterative/setup-cml@v1
      - name: Train model
        env:
          REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          pip install -r requirements.txt
          python train.py
          cat metrics.txt >> report.md
          cml publish plot.png --md >> report.md
          cml send-comment report.md

```

cml.yaml hosted with ❤ by GitHub



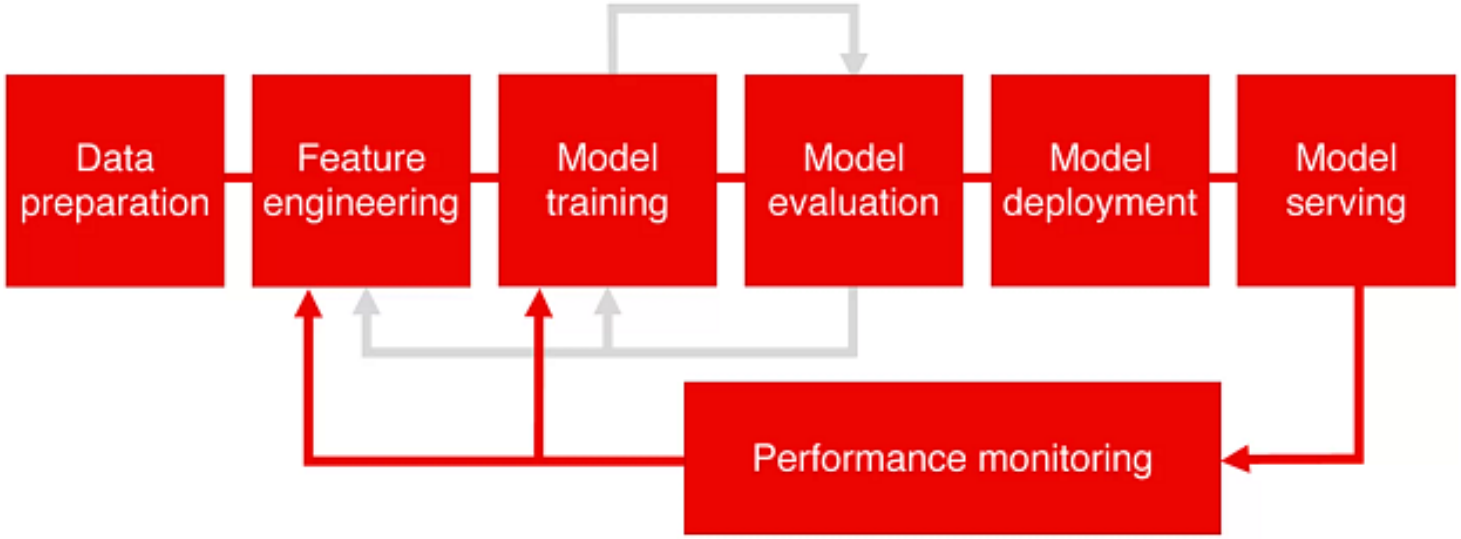
# Model monitoring with Evidently AI

# MLOps stages



# Maintain model quality

Model **performance** may **fluctuate** over time due to changes in the production dataset. Therefore, it is necessary to **monitor** the model and the service to ensure that it works as expected.

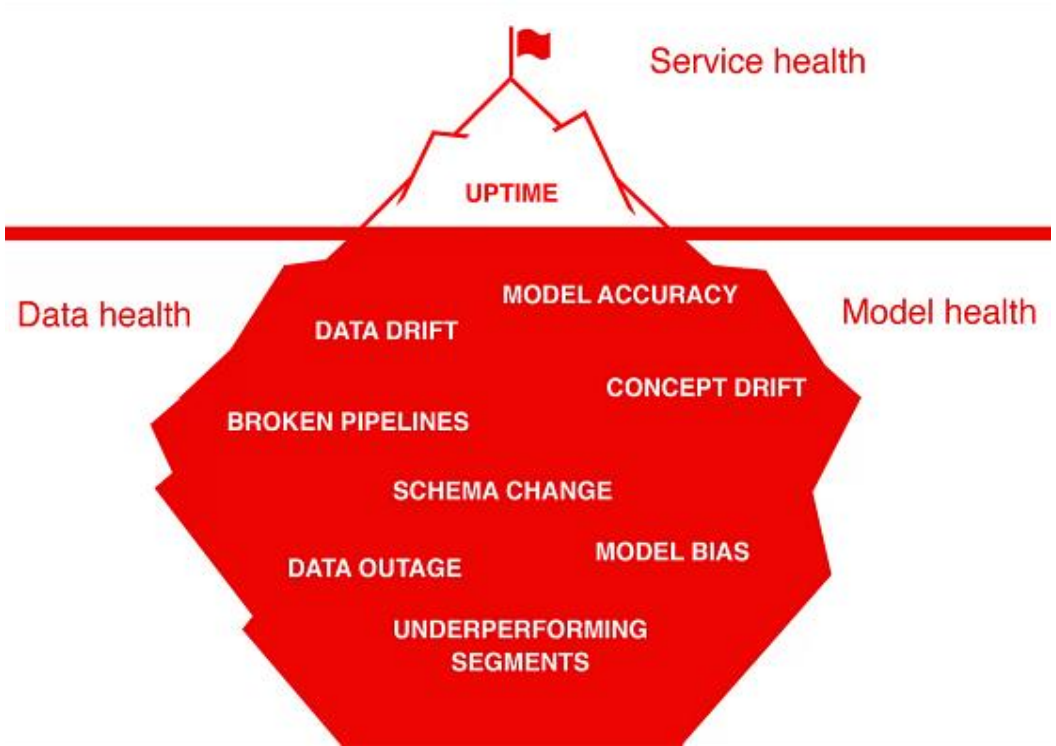


# Type of problems

There are different categories of problems that can occur to ML service:

- **Poor data** quality, broken pipelines, or **technical** issues cause a drop in performance.
- **Data drift.** It is the change in the distribution of data.

  Model works worse in unknown dataset regions.
- **Concept Drift**. The relationship between the target variable and input features changes.

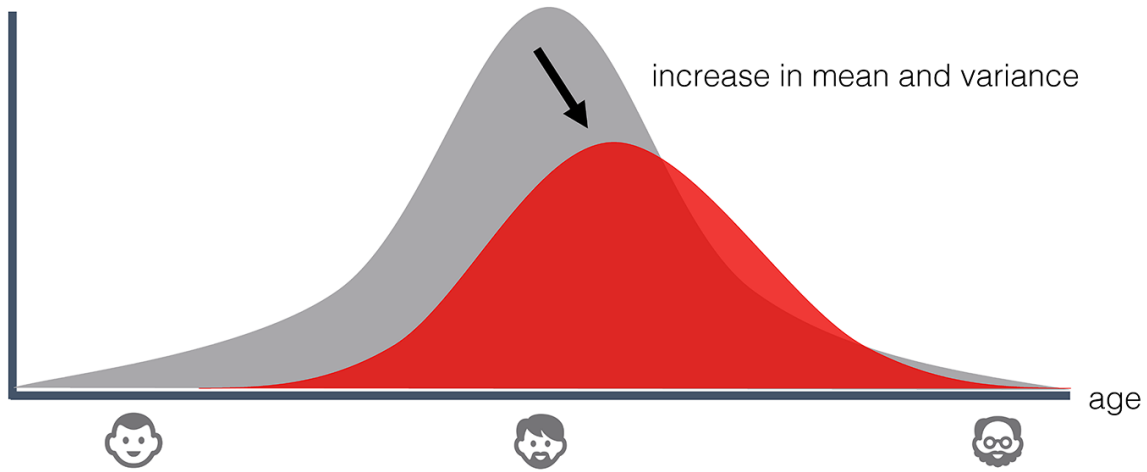


# Data Drift

It occurs due to **changes** in the **input data**. To detect it you must observe the input data in production and compare it with the training data.

Tests to detect changes in the distribution of the input data:

- Kolmogorov–Smirnov (**KS**) test
- Population Stability Index (**PSI**)
- **Z-score**

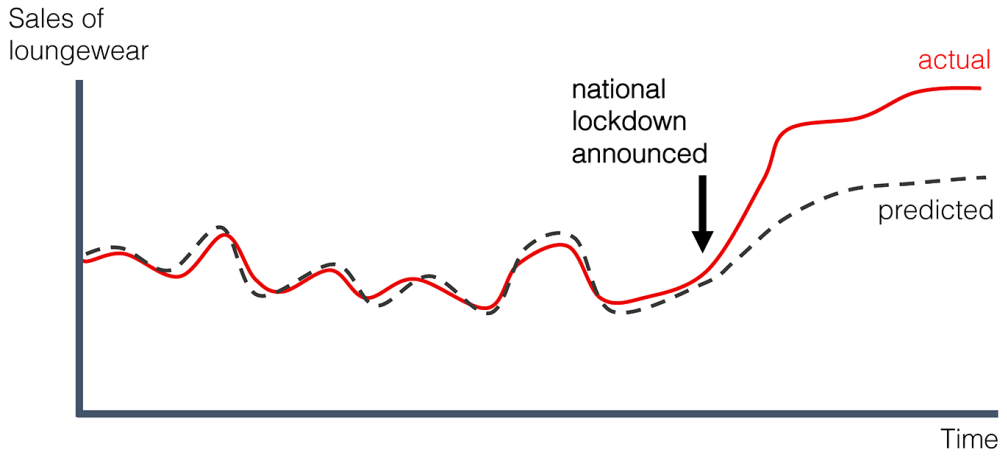


# Concept Drift

**Concept Drift** refers to the **change** in the relationships between input and output data in the underlying problem **over time**. You can detect it by looking at changes in the input prediction probabilities. Example: inflation in the prediction of house prices.

Prevent concept drift:

- Model **monitoring**
- Time based approach, **retraining** the model every X time
- **Continuous retraining**

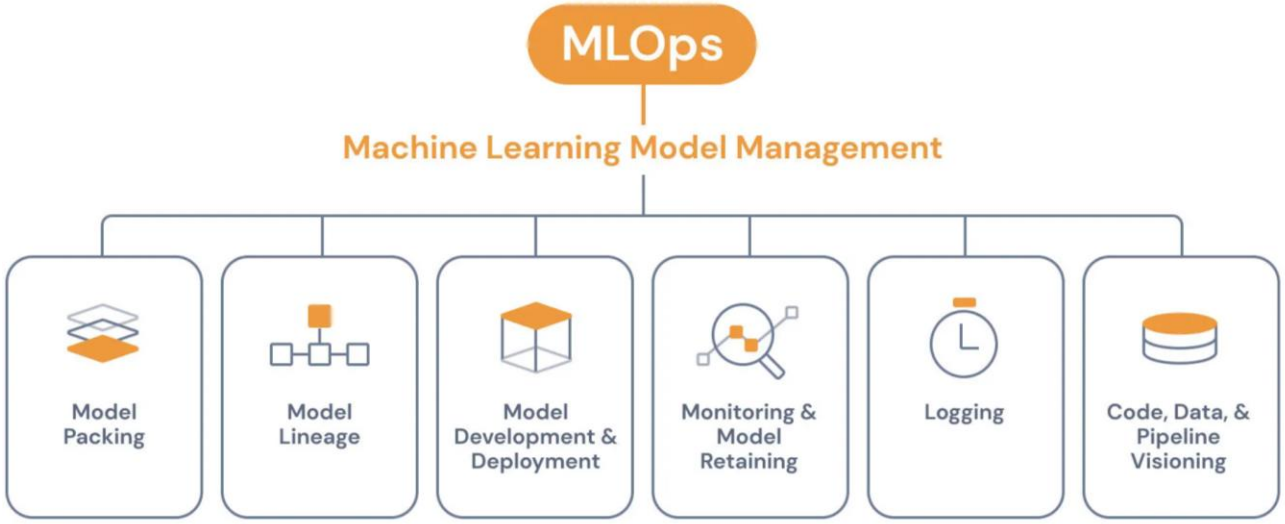


# Model performance

**Performance monitoring** helps us detect that a production model is **underperforming** and why it is underperforming. In addition, it is necessary to define what is considered **low performance**, because it will depend on the **use case**.

To improve performance:

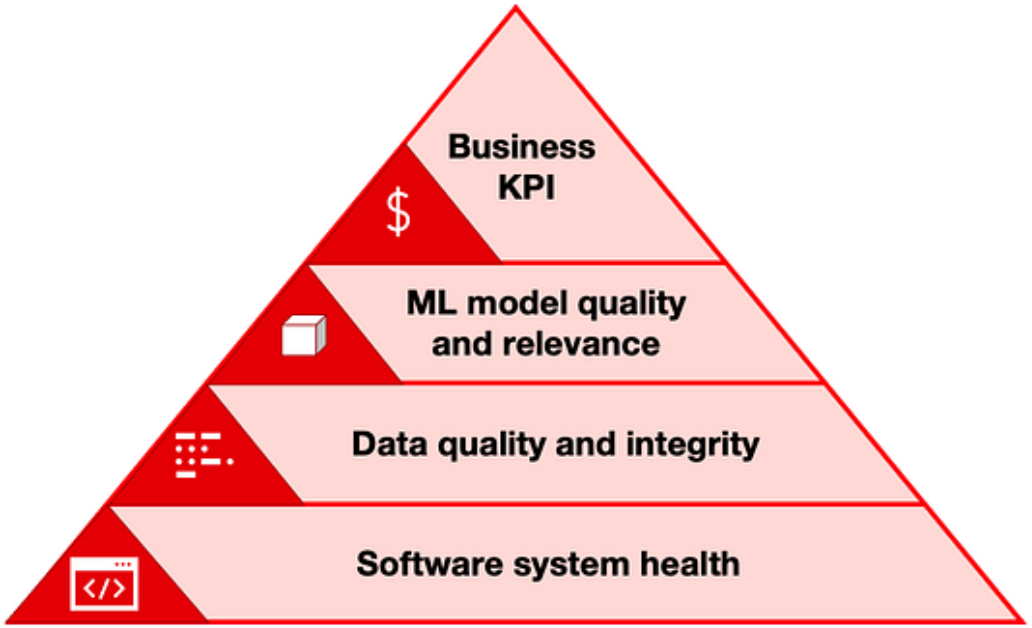
- Keep data preprocessing and ML model in **separate modules**
- Use a **reference** model
- Easily **retrainable** model architecture



# List of evaluations

An adequate monitoring system must cover the following revisions:

- Review data **distribution changes**
- Review **bias** between training and service
- Identify model **drift** and concept drift
- Identify issues in **pipelines**
- Identify **performance** issues
- **Data quality** review



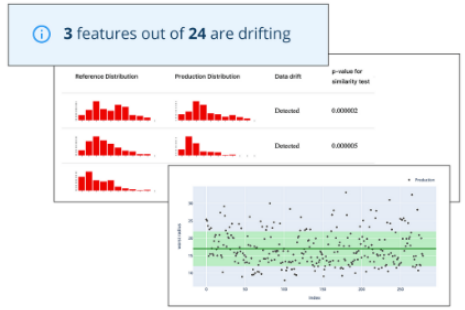
# Monitoring tools

There are different monitoring tools, although some of the most relevant are:

- Aporia
- Deepchecks
- Evidently AI
- MLRun


# Evidently AI

**Evidently AI** is one of the **best emerging tools**. Evidently generates interactive **dashboards** that can be used for model evaluation, debugging, and documentation. Each report covers a particular aspect of model performance




### Data Drift

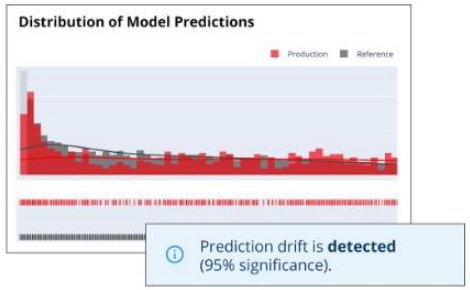
Run statistical tests to compare the input feature distributions, and visually explore the drift.

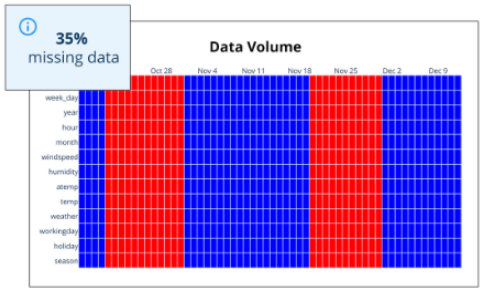
GET STARTED

### Target Drift

Understand how model predictions and target behavior change over time.

GET STARTED




### Data Quality

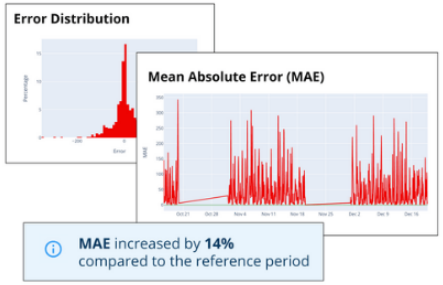
Get a snapshot of data health, and drill down to explore feature behavior and statistical properties.

GET STARTED

### Model Quality

Evaluate the ML model quality, and go beyond aggregate performance to discover where it fails.

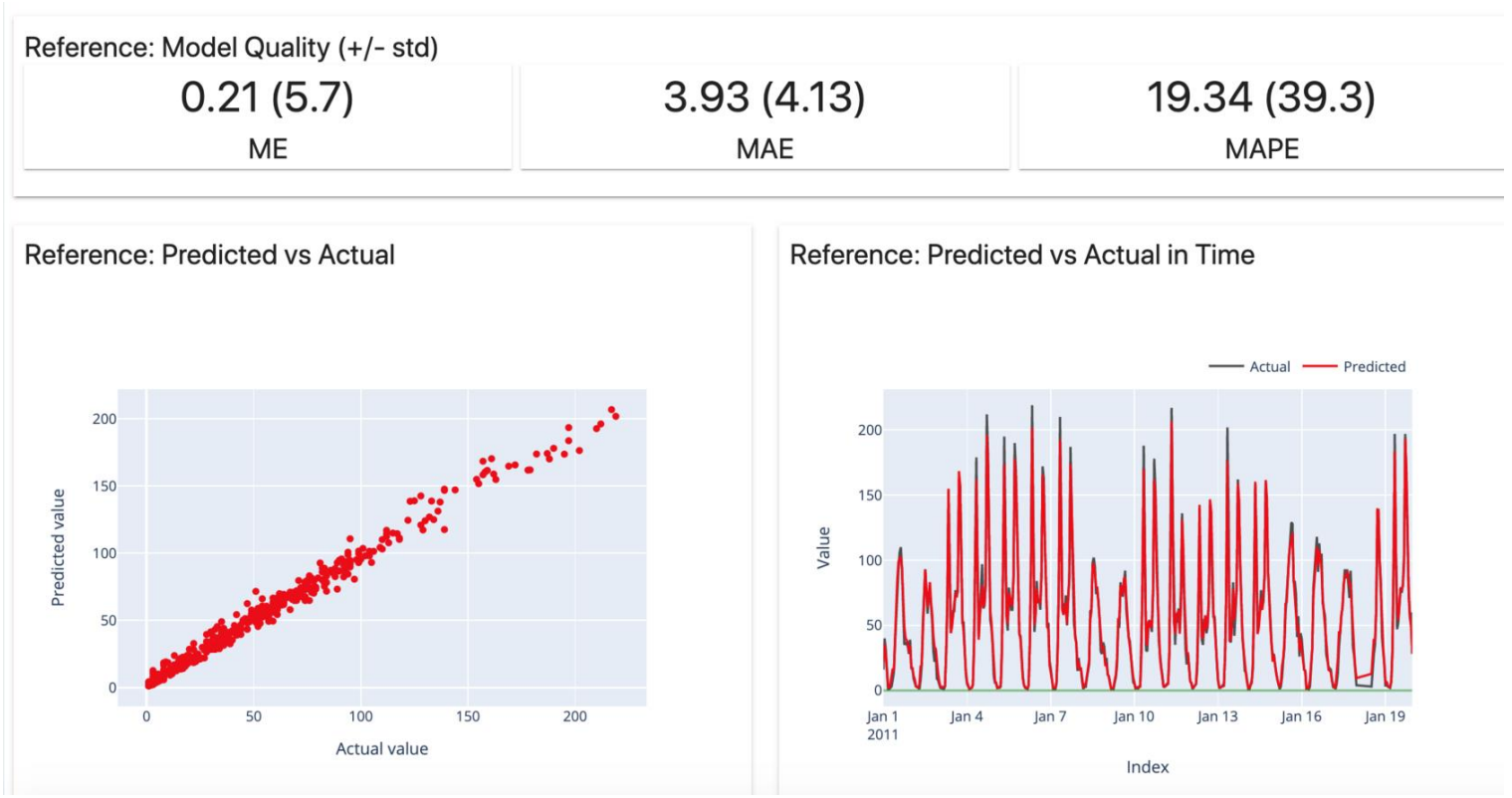
GET STARTED



# Evindetly Commands

To work with **Jupyter Notebook**, we must execute those commands to install **nbextension** in the prompt :

- *jupyter nbextension install --sys-prefix --symlink --overwrite --py evidently*
- *jupyter nbextension enable evidently --py --sys-prefix*



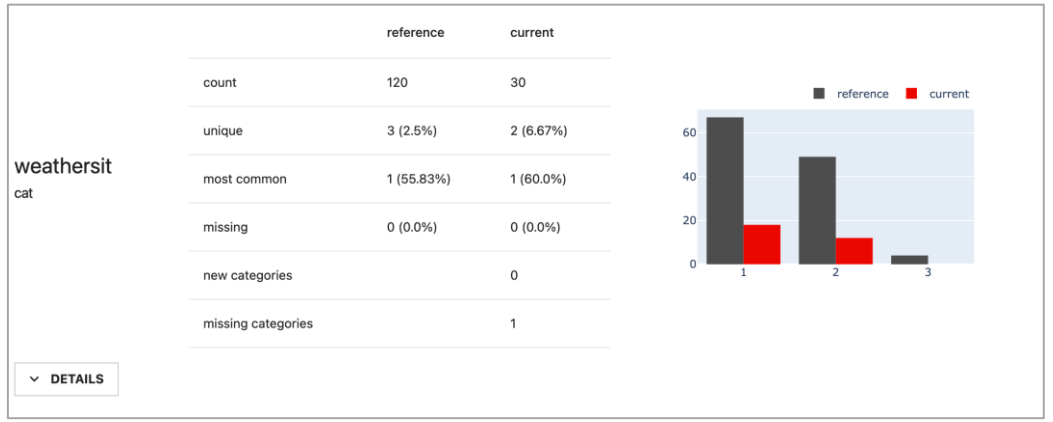
# Additional Information

There are two types of validations:

1. **Metric Presets**: Pre-built reports for visual exploration. Exportable to JSON.
2. **Test Presets**: Pre-built test suites for testing as part of the pipeline.

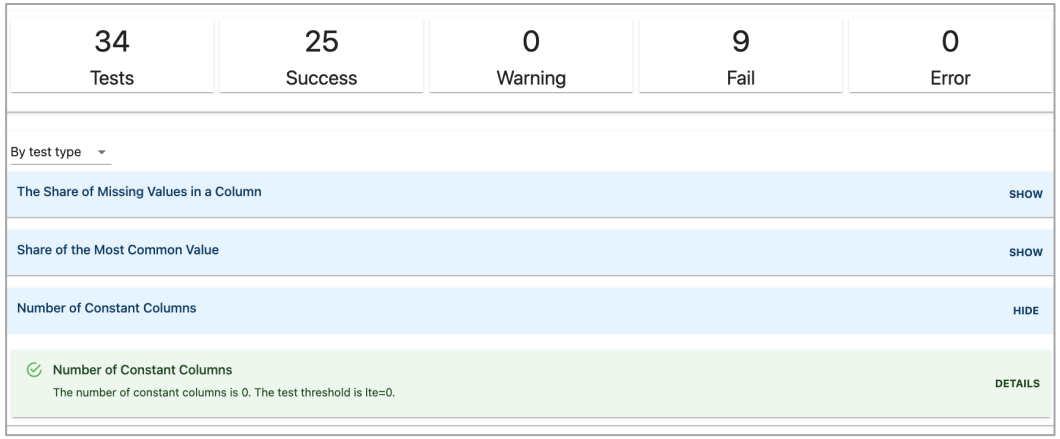
Report (DataQualityPreset)

```
data_quality_report = Report(metrics=[
    DataQualityPreset(),
])

data_quality_report.run(reference_data=adult_ref, current_data=adult_cur)
data_quality_report
```




Test Suite (DataQualityTestPreset, DataStabilityTestPreset)

```
data_quality_test_suite = TestSuite(tests=[
   DataQualityTestPreset(),
])

data_quality_test_suite.run(reference_data=ref, current_data=curr)
data_quality_test_suite
```



# Monitorization with DeepChecks

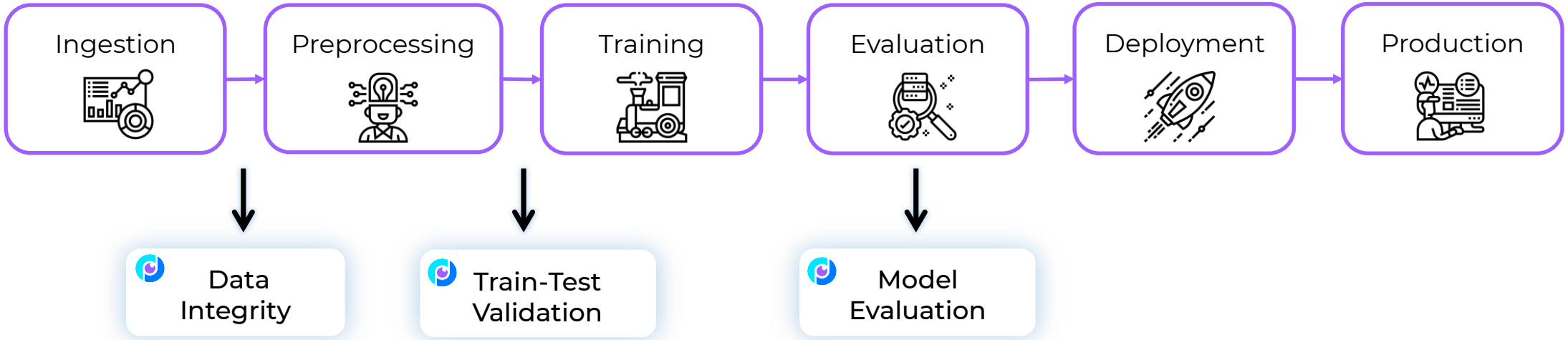
# Monitoring tools

There are different monitoring tools, although some of the most relevant are:

- Aporia
- Deepchecks
- Evidently AI
- MLRun


# DeepChecks

**Deepchecks** is a Python library for **validating models and data**. Includes checks for model performance, data integrity, or distribution misfit. Three components: **check, conditions and suite**.



# Componentes: Checks

Each "**check**" allows you to inspect a **specific aspect** of the data or models. It is the basic component and covers all types of analysis.

Two types of **results**: visual result + condition value

## Data Integrity

- Class Imbalance
- Columns Info
- Conflicting Labels
- Data Duplicates
- Feature Feature Correlation
- Feature Label Correlation
- Identifier Label Correlation
- Is Single Value
- Mixed Data Types
- Mixed Nulls
- Outlier Sample Detection
- Percent Of Nulls
- Special Characters
- String Length Out Of Bounds
- String Mismatch

## Train Test Validation

- Datasets Size Comparison
- Date Train Test Leakage Duplicates
- Date Train Test Leakage Overlap
- Feature Drift
- Feature Label Correlation Change
- Index Leakage
- Label Drift
- Multivariate Drift
- New Category
- New Label
- String Mismatch Comparison
- Train Test Samples Mix

## Model Evaluation

- Boosting Overfit
- Calibration Score
- Confusion Matrix Report
- Model Inference Time
- Model Info
- Multi Model Performance Report
- Performance Bias
- Prediction Drift
- Regression Error Distribution
- Regression Systematic Error
- ROC Report
- Segment Performance
- Simple Model Comparison
- Single Dataset Performance
- Train Test Performance
- Unused Features

## Custom Checks

- Create a Custom Check

# Components: Condition

A **condition** is a function that can be **added to a Check**, which returns **pass ✓**, **fail ✕** or **warning !**

It is intended to **validate** the value of the **Check**

Example:

```
from deepchecks.tabular.checks import BoostingOverfit
BoostingOverfit().add_condition_test_score_percent_decline_not_greater_than(threshold=0.05)
```

*Will return a check error if there is a difference of more than 5% between the best score in the test set and the score from the last iteration (overfit)*

# Components: Suite

A **Suite** is a **collection of checks**, to which you can add conditions. Shows a final report for the executed checks.

You can **edit** the preconfigured suites or create your own suite with a collection of optional checks and conditions.

```python
from deepchecks.tabular.suites import data_integrity

# Run data integrity suite
integ_suite = data_integrity()
result = integ_suite.run(dataset)

# Save as a HTML report
result.save_as_html(config.report.data_integrity)
```

```python
assert result.passed()
```

▾ Model Evaluation Suite

# Model Evaluation Suite

The suite is composed of various checks such as: Regression Systematic Error, Performance Report, Weak Segments Performance, etc...

Each check may contain conditions (which will result in pass / fail / warning / error , represented by ✓ / ✕ / ! / ?! ) as well as other outputs such as plots or tables.

Suites, checks and conditions can all be modified. Read more about custom suites.

▸ Didn't Pass

▸ Passed

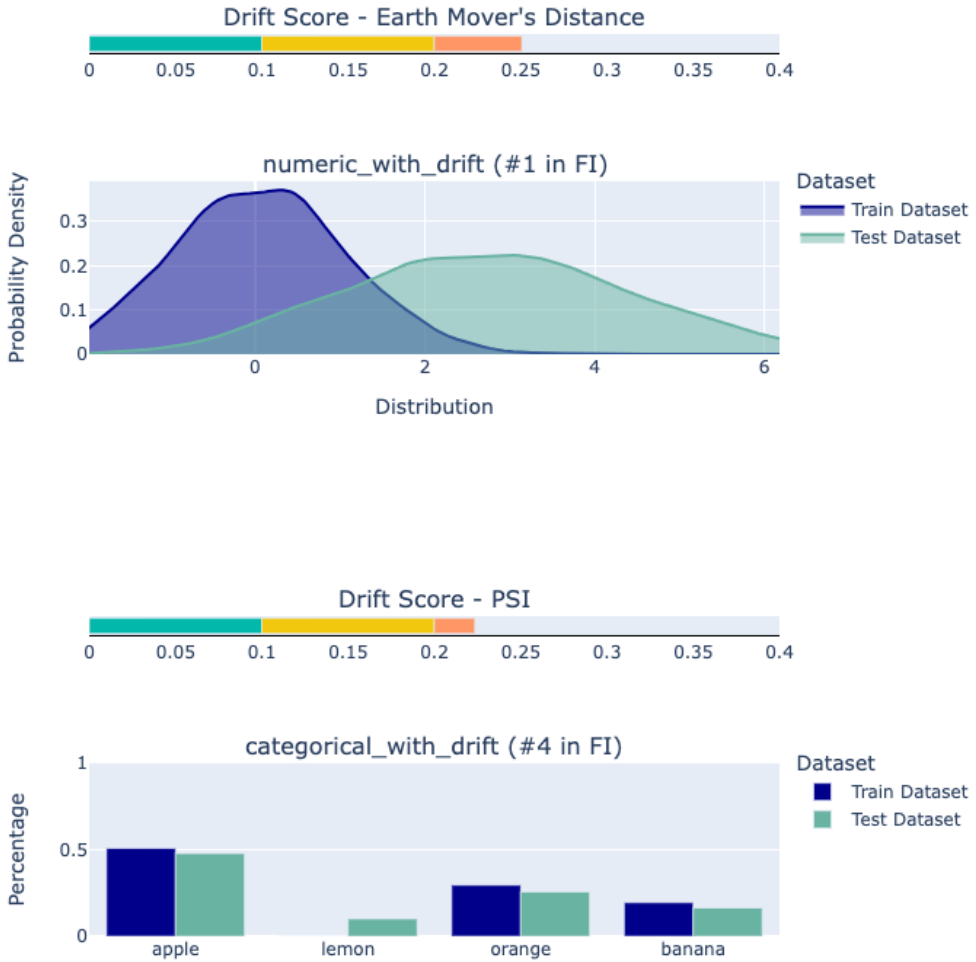
▸ Other

▸ Didn't Run

# Check Example

To run a specific **check**, all you need is to import it and then run it with the required input parameters.

```
from deepchecks.tabular.checks import FeatureDrift
import pandas as pd

train_df = pd.read_csv('train_data.csv')
test_df = pd.read_csv('test_data.csv')
# Initialize and run desired check
FeatureDrift().run(train_df, test_df)
```

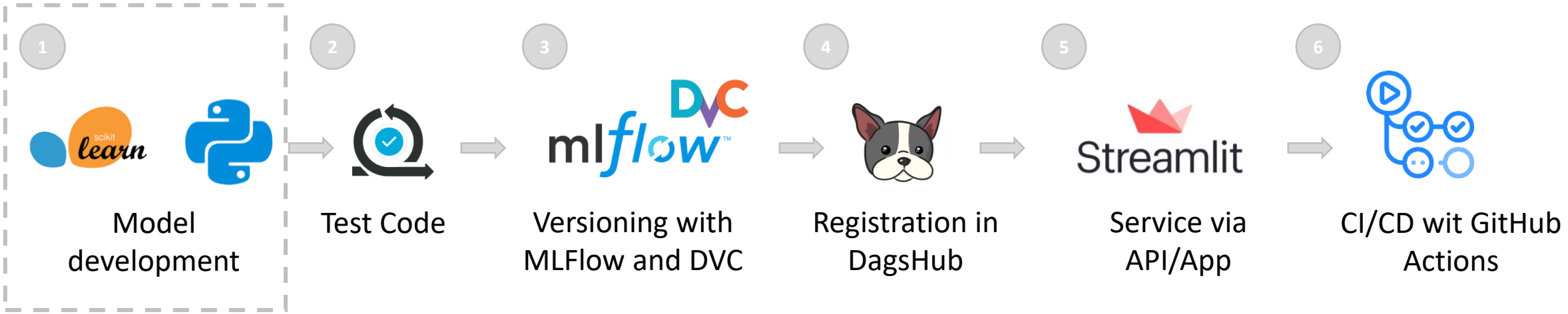


# End-to-end ML Project

# End-to-End Project

We will develop a project from start to finish where we will do the following:

** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



# Project Structure

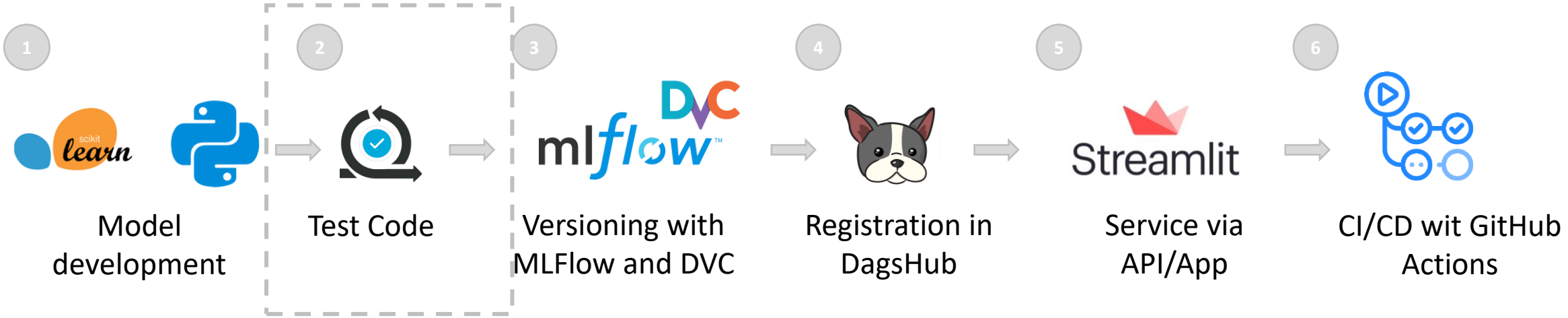
Project structure

- **application**: application and API with Streamlit and BentoML
- **config**: consists of configuration files
- **data**: where the data is stored
- **models**: where the already trained models are stored
- **notebooks**: notebook store to visually analyze the results/data
- **training**:
    - **src**: Python codes for model development
    - **tests**: consists of test files

# End-to-End Project

We will develop a project from start to finish where we will do the following:

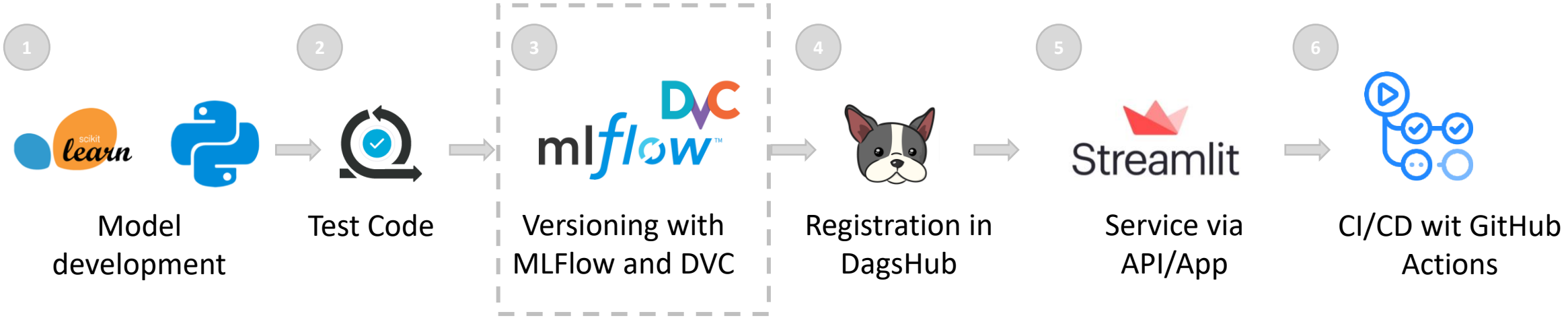
** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



# End-to-End Project

We will develop a project from start to finish where we will do the following:

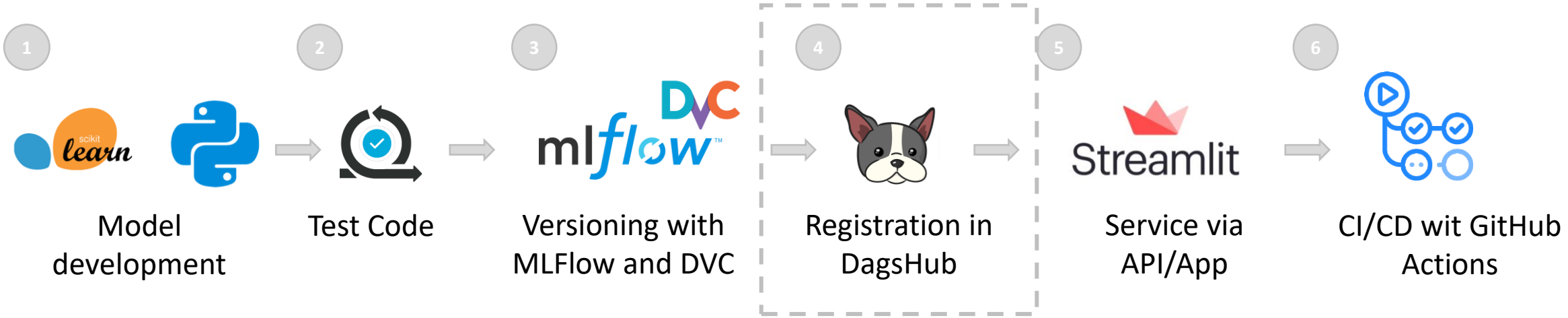
** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



# End-to-End Project

We will develop a project from start to finish where we will do the following:

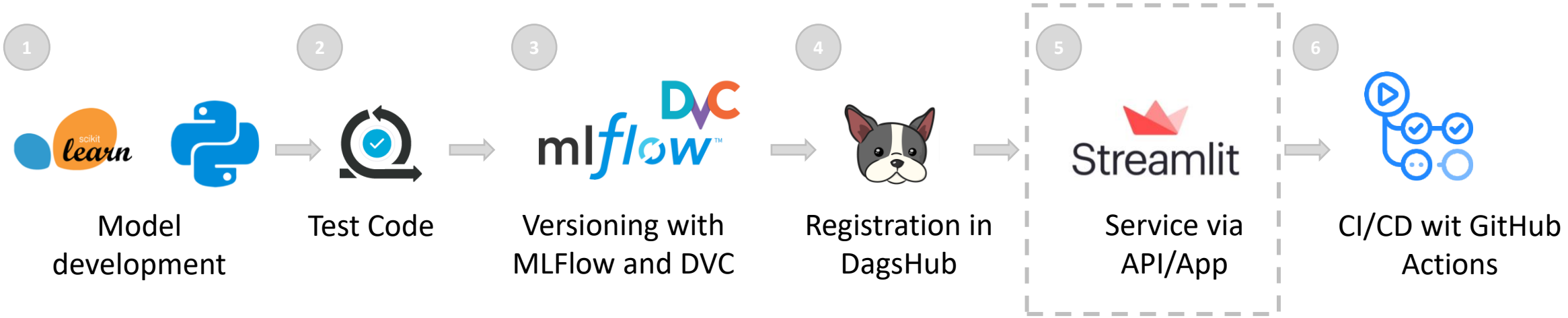
** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



# End-to-End Project

We will develop a project from start to finish where we will do the following:

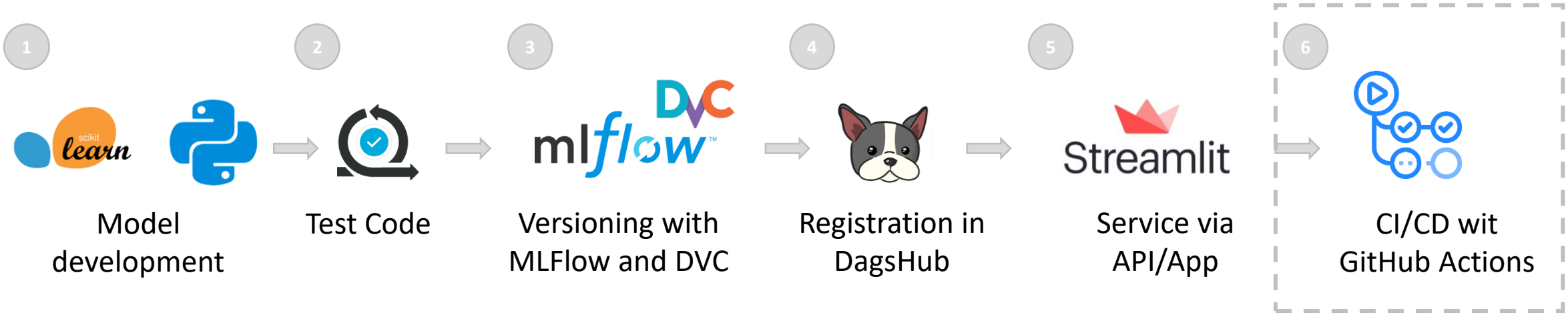
** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



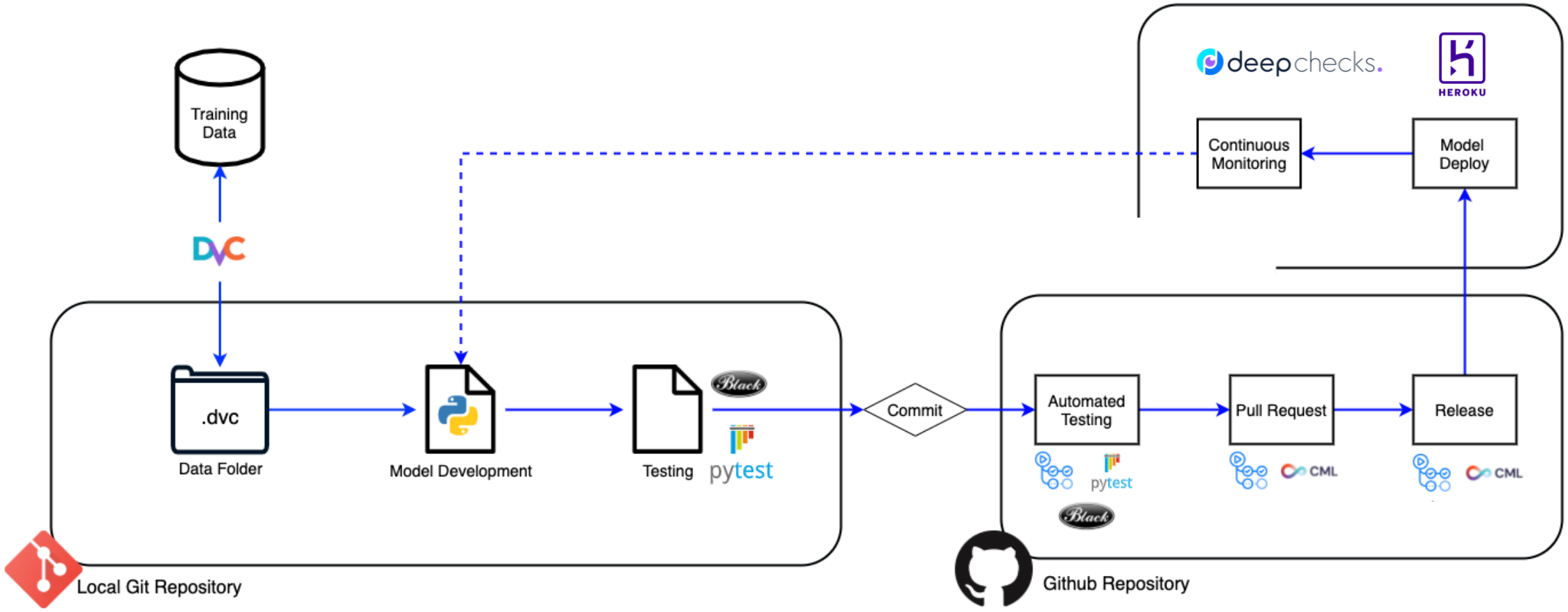
# End-to-End Project

We will develop a project from start to finish where we will do the following:

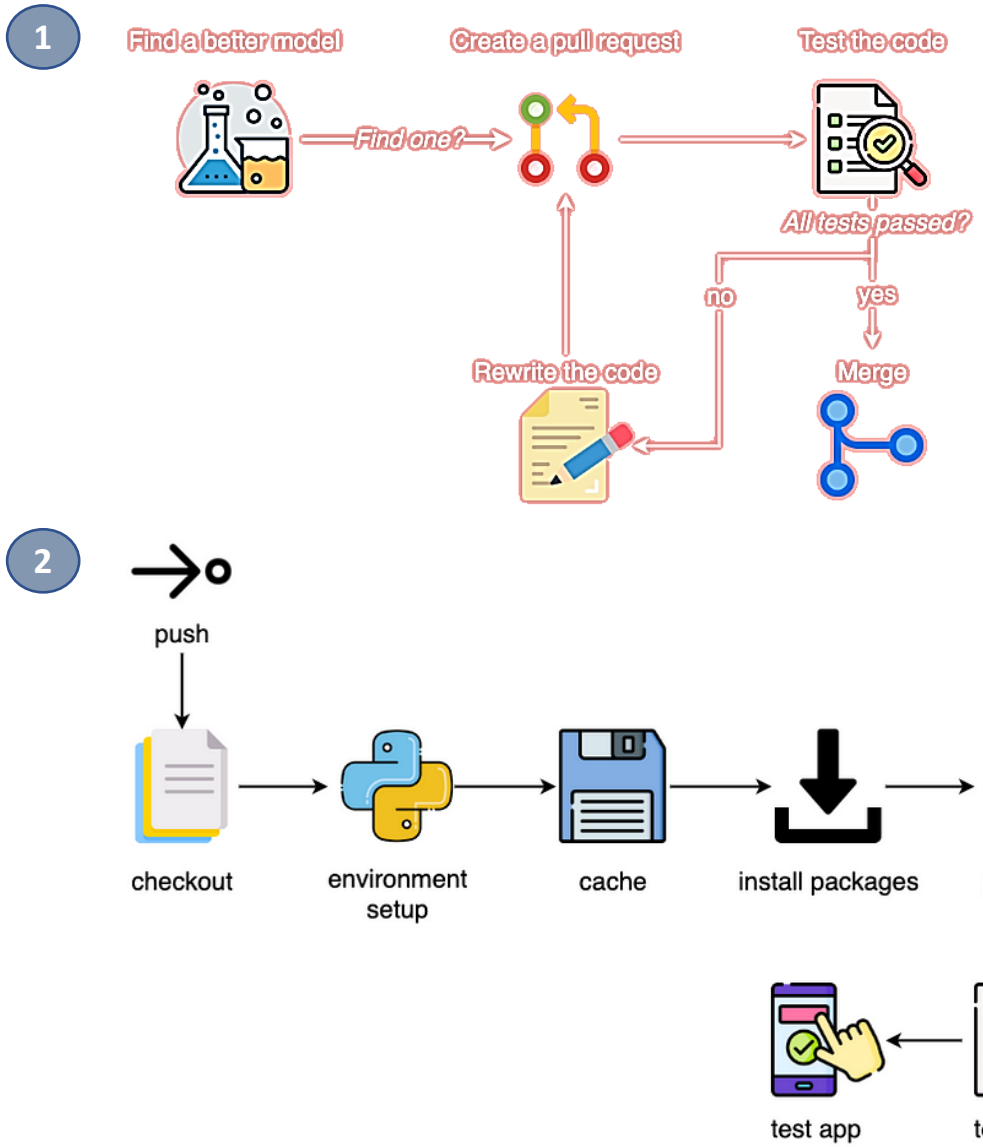
** Khuyen Tran Code (recommended to follow it): https://dagshub.com/khuyentran1401/employee-future-prediction/src/master



# CI/CD


Training Data
DVC
.dvc
Data Folder
Model Development
Testing
Black
pytest
Local Git Repository
Commit
Automated Testing
pytest
Black
Pull Request
CML
Release
CML
Github Repository
deepchecks.
HEROKU
Continuous Monitoring
Model Deploy

# GitHub Actions Workflows

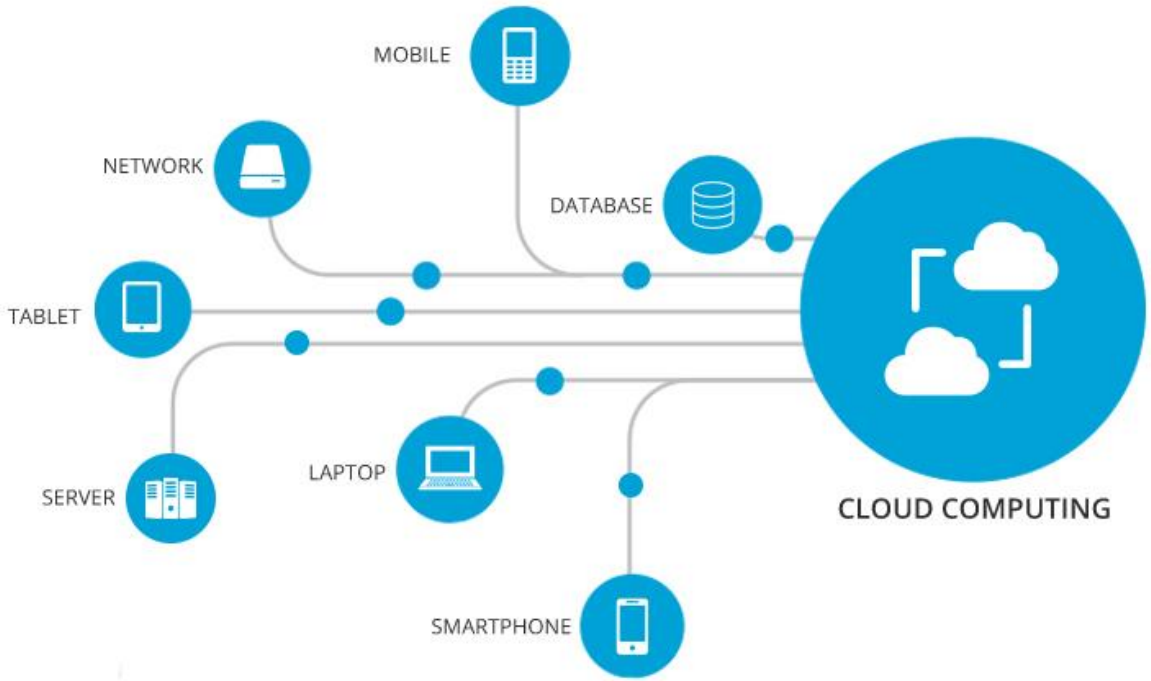


# Azure Fundamentals

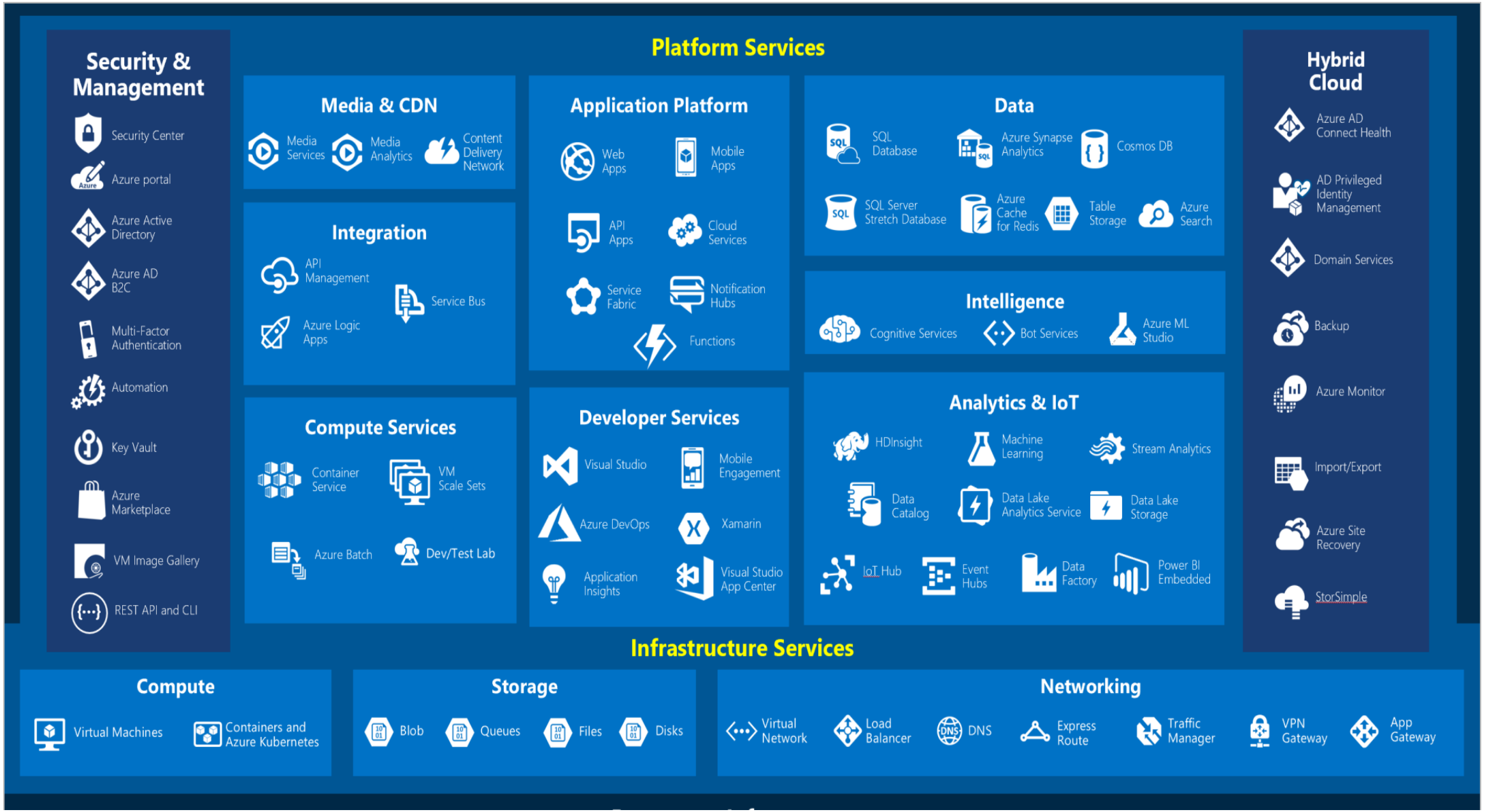
# Cloud Computing

Cloud computing is the provision of computing services over the Internet to offer **flexible resources** at scale.

Allows renting resources (CPU, RAM, storage) from a cloud provider (Azure) and paying only for what is used **(pay per use)**

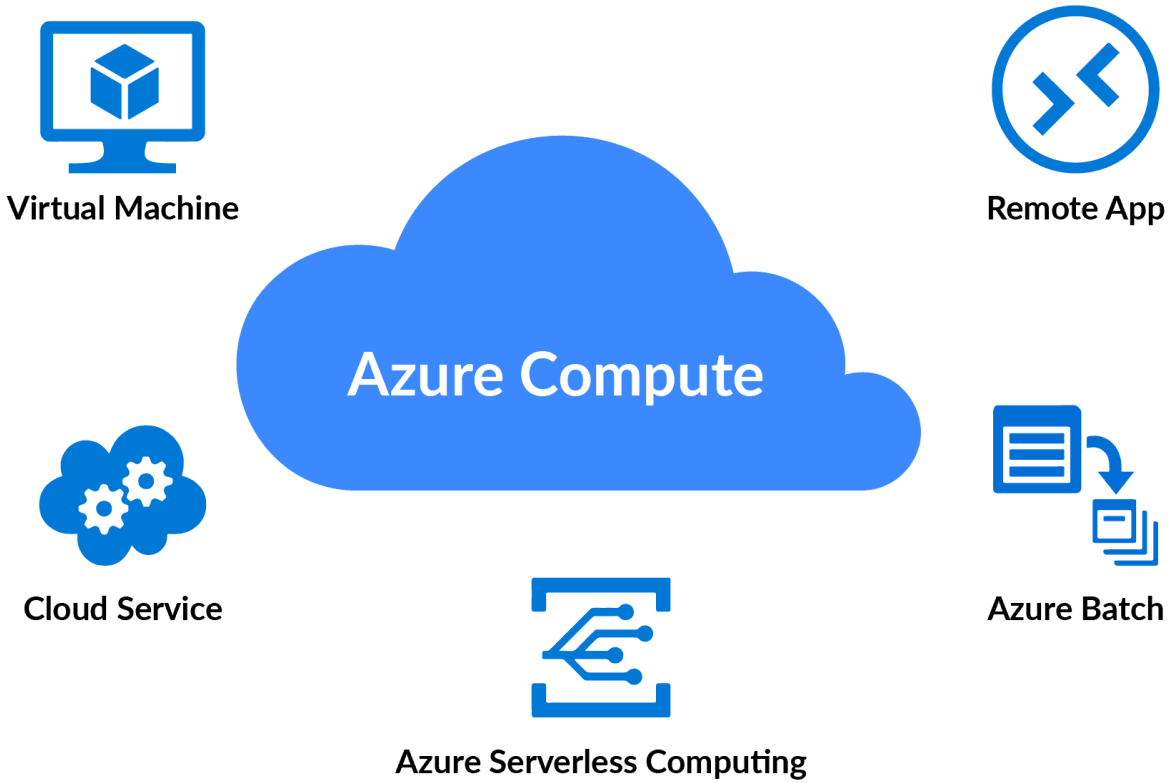


# Azure Fundamentals


Platform Services
Security & Management
Security Center
Azure portal
Azure Active Directory
Azure AD B2C
Multi-Factor Authentication
Automation
Key Vault
Azure Marketplace
VM Image Gallery
REST API and CLI
Media & CDN
Media Services
Media Analytics
Content Delivery Network
Integration
API Management
Service Bus
Azure Logic Apps
Compute Services
Container Service
VM Scale Sets
Azure Batch
Dev/Test Lab
Application Platform
Web Apps
Mobile Apps
API Apps
Cloud Services
Service Fabric
Notification Hubs
Functions
Developer Services
Visual Studio
Mobile Engagement
Azure DevOps
Xamarin
Application Insights
Visual Studio App Center
Data
SQL Database
Azure Synapse Analytics
Cosmos DB
SQL Server Stretch Database
Azure Cache for Redis
Table Storage
Azure Search
Intelligence
Cognitive Services
Bot Services
Azure ML Studio
Analytics & IoT
HDInsight
Machine Learning
Stream Analytics
Data Catalog
Data Lake Analytics Service
Data Lake Storage
IoT Hub
Event Hubs
Data Factory
Power BI Embedded
Hybrid Cloud
Azure AD Connect Health
AD Privileged Identity Management
Domain Services
Backup
Azure Monitor
Import/Export
Azure Site Recovery
StorSimple
Infrastructure Services
Compute
Virtual Machines
Containers and Azure Kubernetes
Storage
Blob
Queues
Files
Disks
Networking
Virtual Network
Load Balancer
DNS
Express Route
Traffic Manager
VPN Gateway
App Gateway

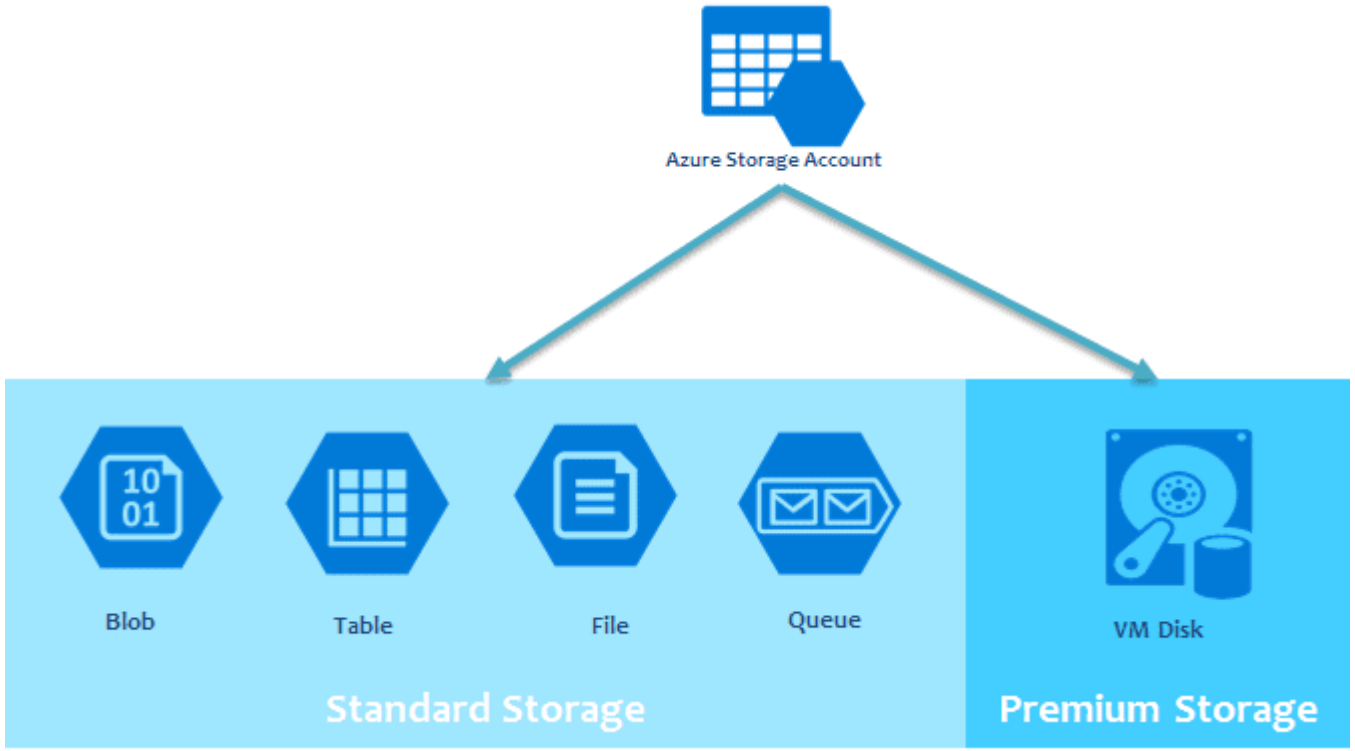
# Azure Compute Service

**Azure Compute service** is responsible for **hosting services** and running the application on workload with making sure of Availability of live application.



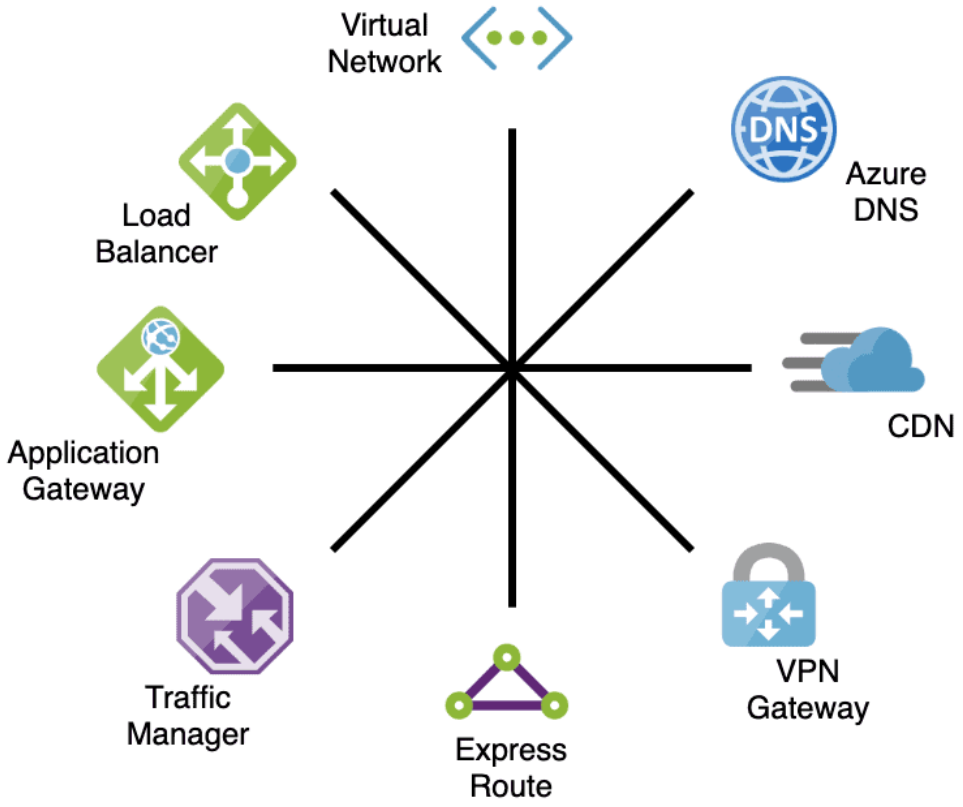
# Azure Storage Platform

The **Azure Storage** platform is Microsoft's **cloud storage solution** for modern data storage scenarios. Azure Storage offers highly available, massively scalable, durable, and secure storage for a variety of data objects in the cloud.



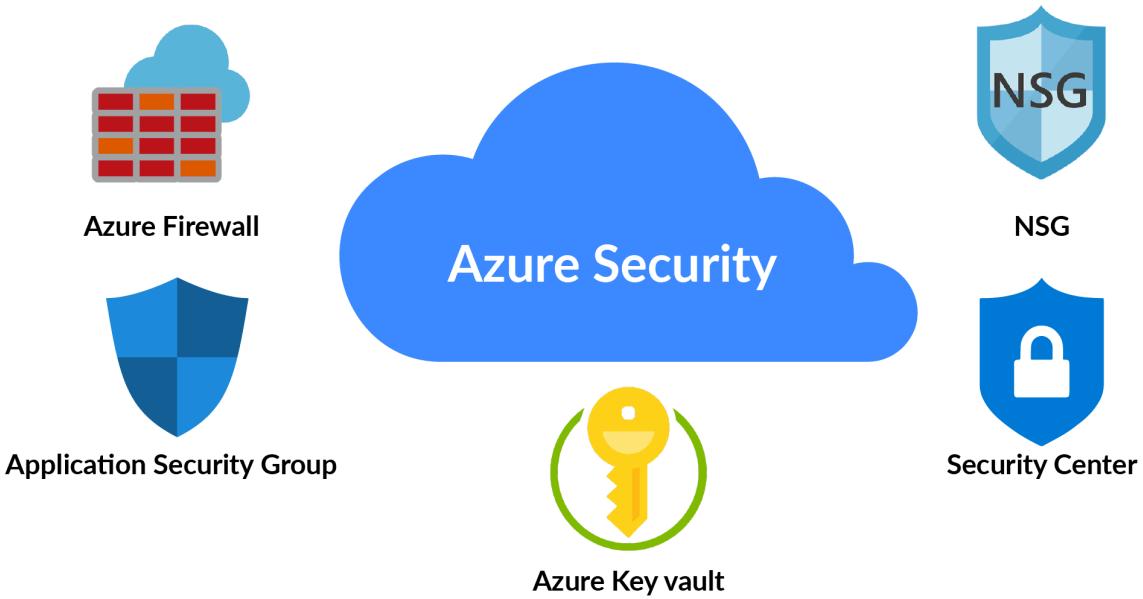
# Azure Network Services

**Microsoft Azure networking services** offer various capabilities to **connect** and manage cloud resources. Beyond virtual networks and connectivity options, Azure offers tools to monitor and manage traffic, perform load balancing and ensure secure user connections



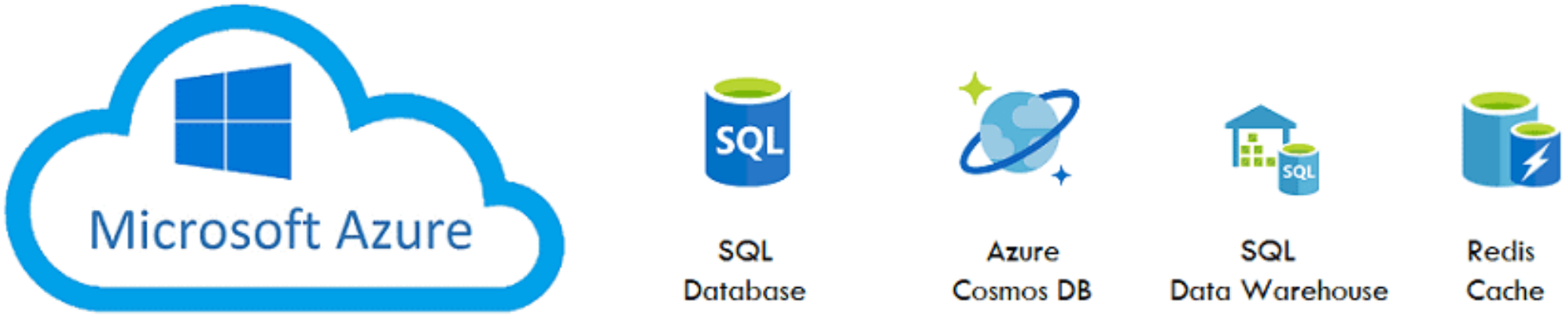
# Azure Security

Azure Security refers to **security tools and capabilities available on Azure cloud platform**. According to Microsoft, the tools for securing its cloud service encompasses “a wide variety of physical, infrastructure, and operational controls.”



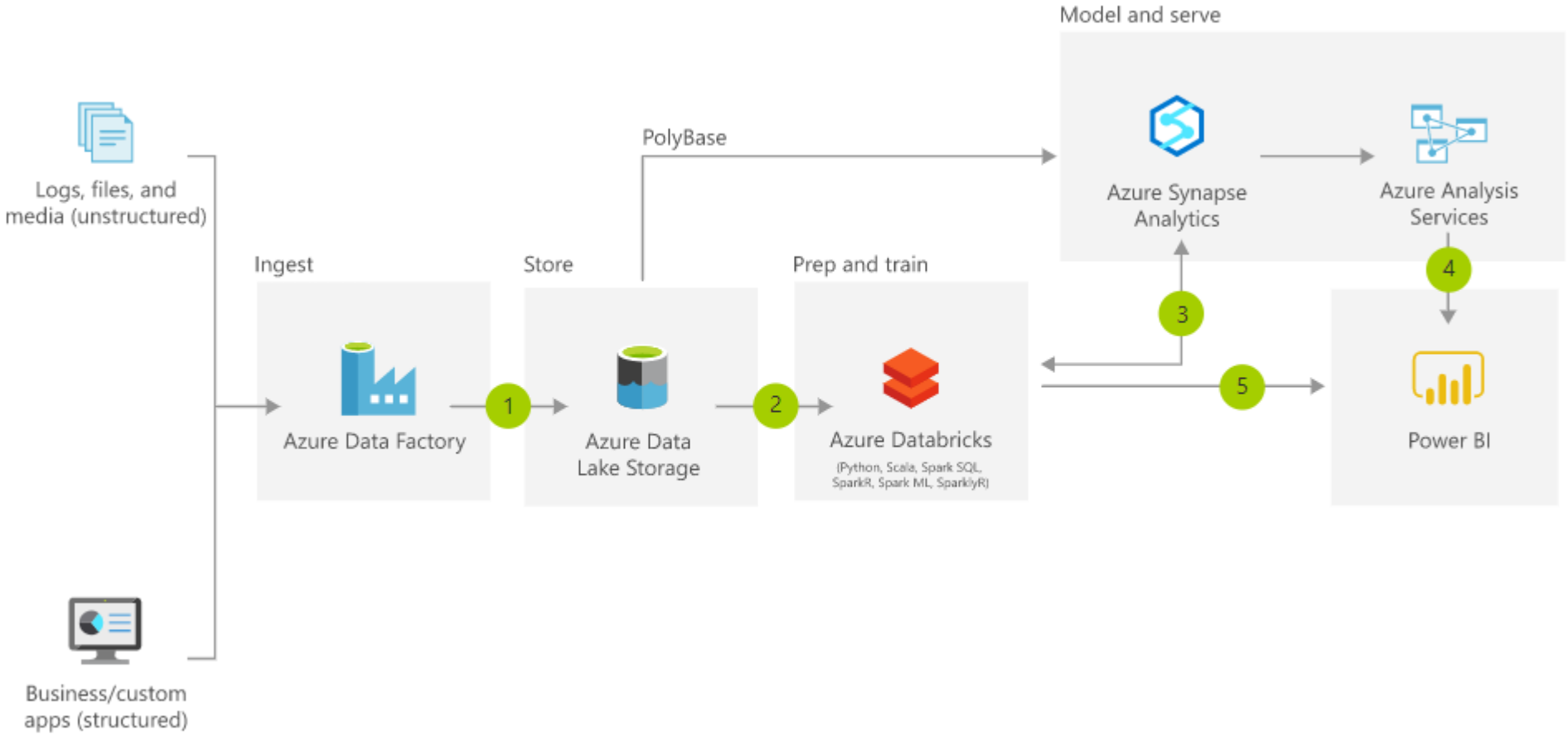
# Data Platform Service

**Data Platform Service** offers a choice of fully managed relational, NoSQL, and in-memory databases, spanning proprietary and open-source engines, to fit the needs of modern app developers.



# Azure Data Platform

**Azure Data Platform Service** provides the foundation for **managing business information**. It includes processes to ingest data, transform it and adequate governance to drive business insights.



# Container Services

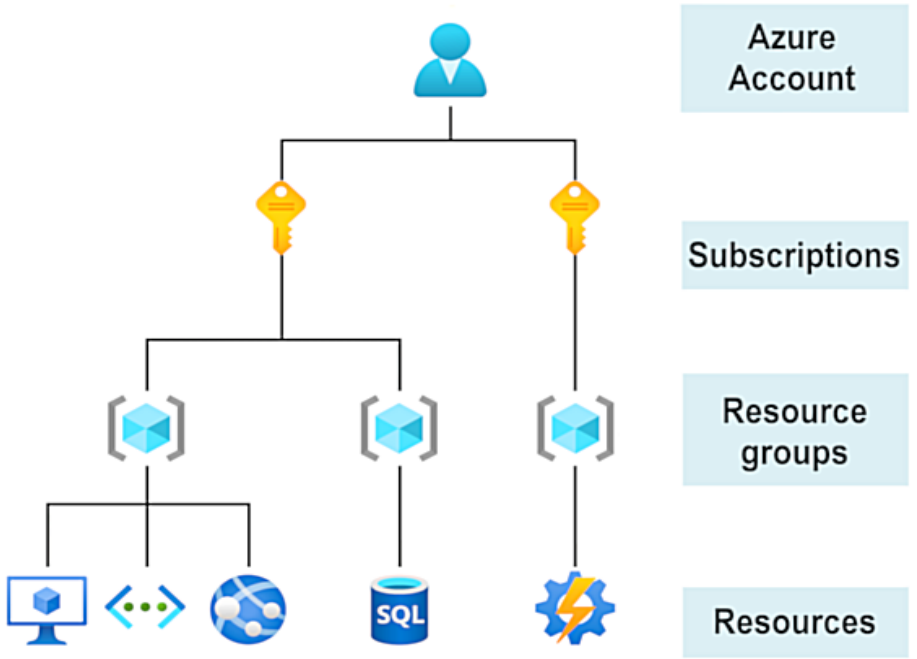
**Container services** are providing small **containers of particular services** or providing microservice architecture which enables faster delivery and high results. It enables Continous integration and continuous delivery tools for the development of Application.

# Azure accounts

To create and use Azure services, you need an **Azure account**. Each account can have one or more **subscriptions associated** with it. Example: A company may use a single account of Azure and separate subscriptions for departments.

Account Types:

- Payment
- Free
- Free for student

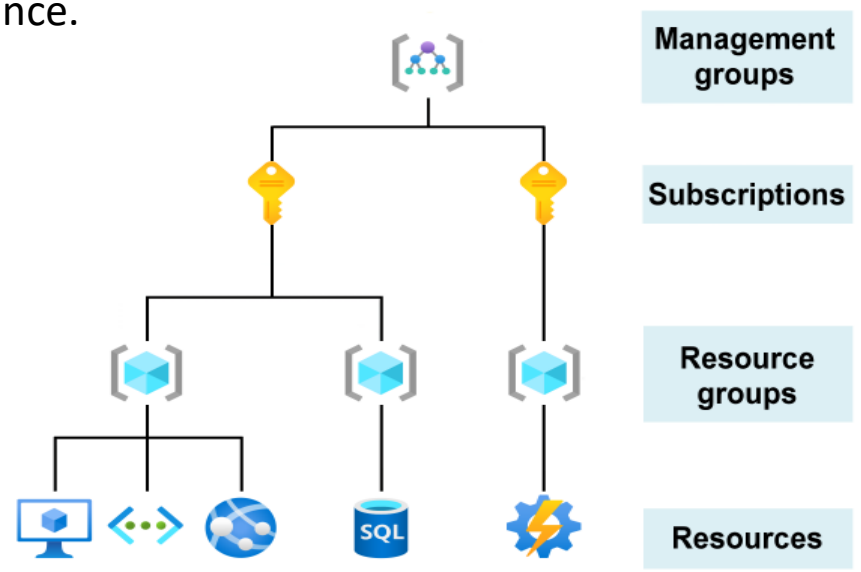


# Subscriptions, management groups, and resources

There are different organizational levels in Azure resources:

- **Resources:** These are instances of services, such as virtual machines, SQL databases, etc.
- **Resource groups:** Resources are combined into resource groups as logical containers.
- **Subscriptions:** A subscription groups user accounts and the resources. They have limits.
- **Administration Groups:** help manage access, policy and compliance.

  All subscriptions inherit the conditions applied to the group.



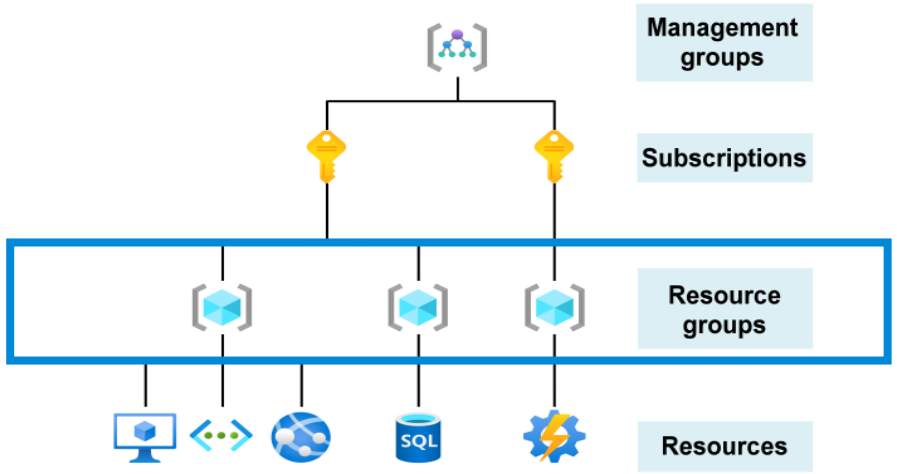
# Resource groups

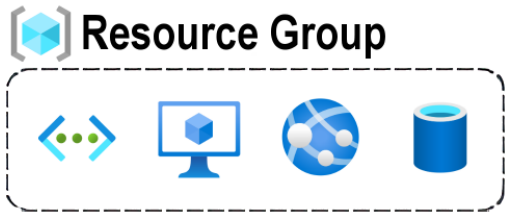
There are two important terms:

- **Resource:** is a single item.
- **Resource group:** is a grouping that contains related resources. They help manage and organize resources.

Characteristics of resource groups:

**Deleting** a resource group deletes all the resources it contains. They allow you to apply role-based **access control** (RBAC) permissions



# Azure Fundamentals

**Azure Storage** is Microsoft's cloud **data storage** solution. It has: security, high availability and scalability. Access via HTTP or HTTPS.

Azure Storage includes the following data services:


**Azure Blob Storage**: object storage

**Azure File Storage**: file share

**Azure Queue Storage**: for storing large numbers of messages

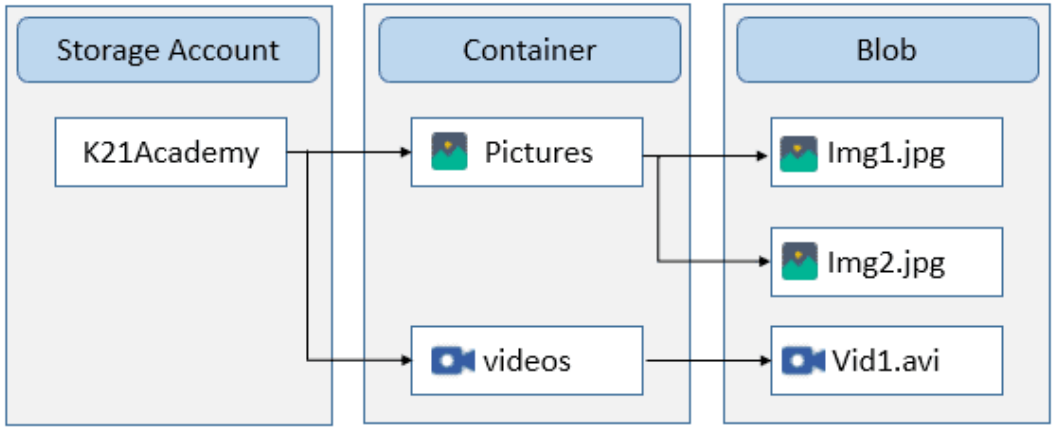
**Azure Table Storage**: for unstructured data (NoSQL)

**Azure Disk Storage:** disk storage for virtual machines

# Azure Blob Storage

**Azure Blob Storage** is an unstructured, **object** storage solution. It allows storing large amounts of data (**text or binary**). You can manage massive amounts of data. Blob Storage is great for:

- ✓ Images or documents
- ✓ Storage with access distributed.
- ✓ Video and audio streaming
- ✓ Backup and restore
- ✓ Storage 8TB for VM



# Blob storage access levels

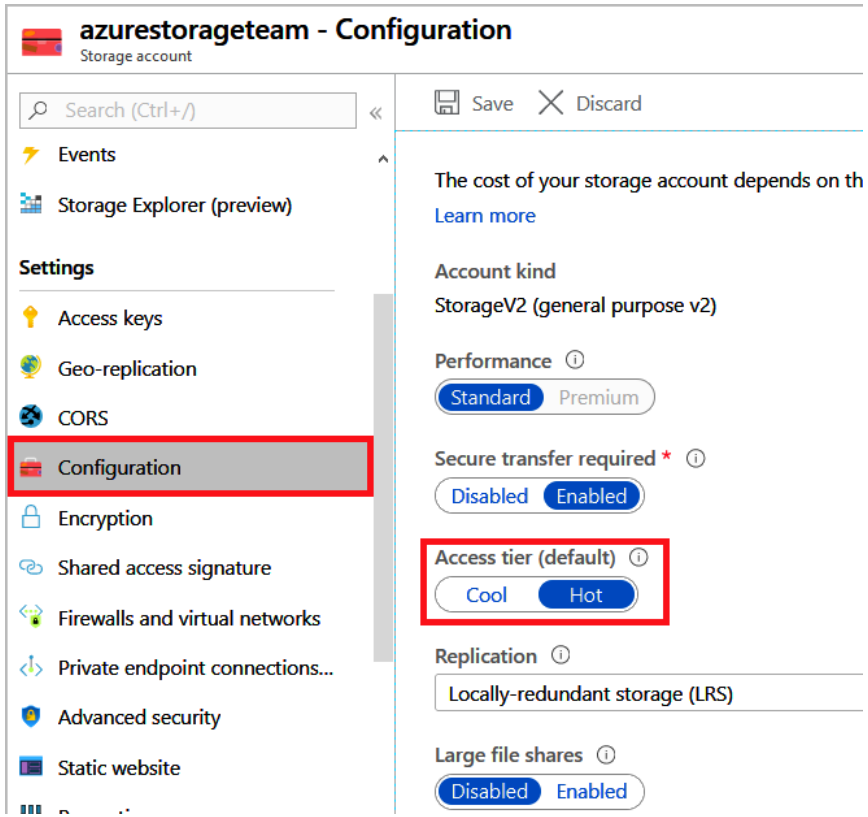
Azure provides multiple **access tiers**, which you can use to balance your storage costs with your access needs.

Azure Storage offers three levels of access:

- **Hot** access tier: frequently accessed data
- **Cool** access tier: infrequently accessed data (stored > 30 days)
- **Archive** access tier: data that is rarely accessed (stored > 180 days)

Storage Lifecycle Policy:

HOT -> COOL -> ARCHIVE



# Azure Machine Learning

**Azure Machine Learning** is a platform for machine learning that allows you to connect to data to **train models**, **deploy** and use them through **API endpoints**.

With Azure ML we can:

- Preprocess the data
- Train and evaluate predictive models
- Create Pipelines with compute-intensive experiments
- Deploy models as APIs