Building a Reproducible Model Workflow

Table of Contents

> "Your job as Data Scientist is ultimately not done until there is a **working model in production** helping the company/organization produce the Return of Investment (ROI) it's expected to achieve."

Business Reflections

Building a Reproducible Model Workflow

The Data Pipeline - Questions to ask yourself

1. How will the Data Pipeline need to work in production to support an **optimized ML** pipeline?
2. **Where** will the ML model run in production?
    a. Is it in the cloud, or in an edge device?
3. Will your ML model run on raw and **unstructured data**, or does it require data preparation and certain data quality levels?

Business Reflections

Building a Reproducible Model Workflow

The Data Rights - Questions to ask yourself

1. Do you own and **have access** to the data needed to train your ML model and run it in production?
2. If the access is limited, how does that potentially impact ML techniques used?
    a. For example, do you need "dummy data" for initial model training in the Lab, and then deploy and run your models using Federated Learning (distributed ML)?

Business Reflections

Building a Reproducible Model Workflow

The Data Quality - Questions to ask yourself

1. How are you approaching needed data quality in your ML pipeline?
2. How will data management be handled in an operational setting?
3. How will you **avoid bias** when selecting your training data set?

Business Reflections

Building a Reproducible Model Workflow

The Operation Aspects - Questions to ask yourself

1. Considering the needs of monitoring while in production at the start of the ML model development will impact choices and priorities.
2. Be sure to consider how you plan to re-train your models once they are in production.

Noah Gift • Following
MLOps Expert | Solopreneur | Author | Duke & Northwestern & UC Davis Adj...
1d • Edited •

In Chapter 12 of the O'Reilly Media book Practical MLOps I also spoke into the future about potential problems with ML predictions like the one facing Zillow.

"....A good summary of this dilemma is to be cautious about the confidence in a prediction or technique. One of the scariest and most talented grapplers I trained with, Dave Terrell, who fought for the UFC championship, told me, don't ever get in a fight with multiple opponents.

The more skill and "skin in the game" a practitioner has, the more they become aware of the epistemic risk. Why risk your life in a street fight with multiple people even if you are a world-class martial artist? Likewise, why be more confident than you should when you predict an election, stock prices, or natural systems?".

In practice, the best way to approach this problem in production is to limit the technique's complexity and assume a lower knowledge of epistemic uncertainty. This takeaway could mean a traditional machine learning high explainability is better than a complex deep learning model with marginally better accuracy."

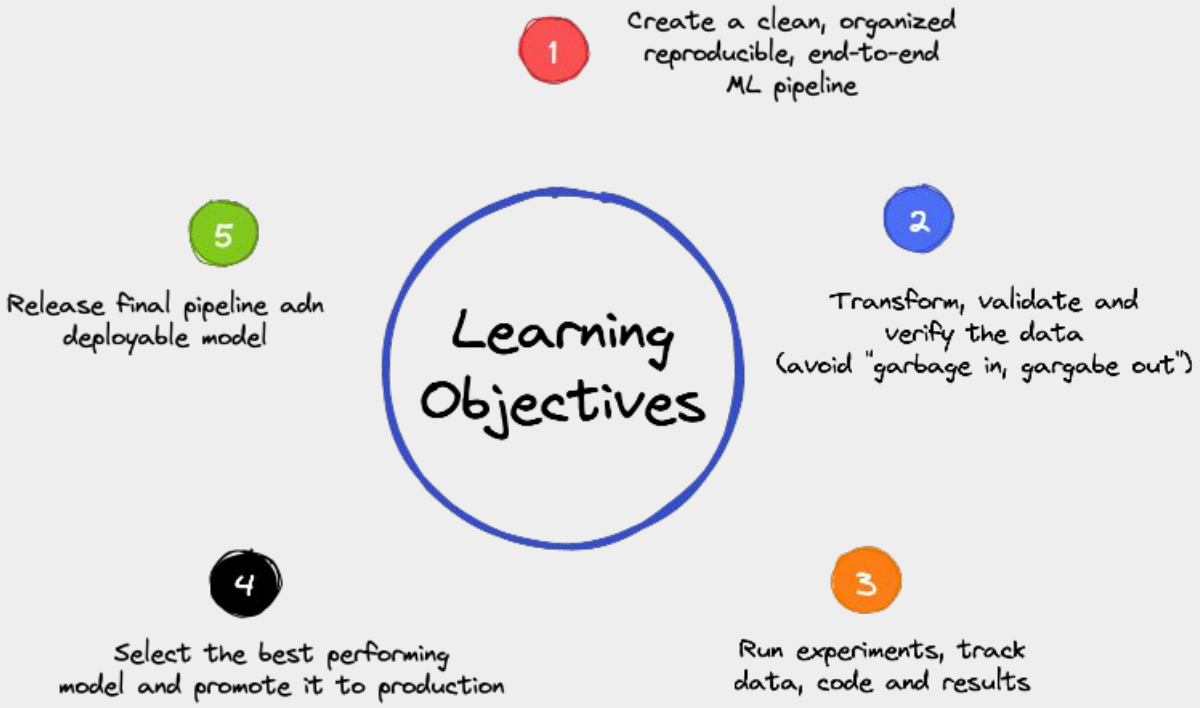
Zillow

https://www.bloomberg.com/news/articles/2021-11-08/zillow-z-home-flipping-experiment-doomed-by-tech-algorithms

https://edition.cnn.com/2021/11/08/homes/zillow-ibuyer-homes/index.html

What you will learn

Building a Reproducible Model Workflow



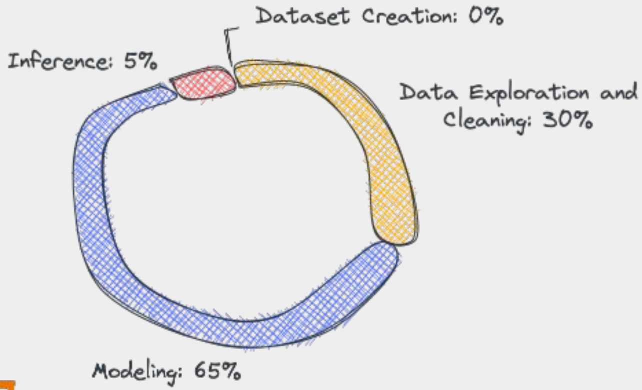
Introduction to Machine Learning Operations (MLOps)

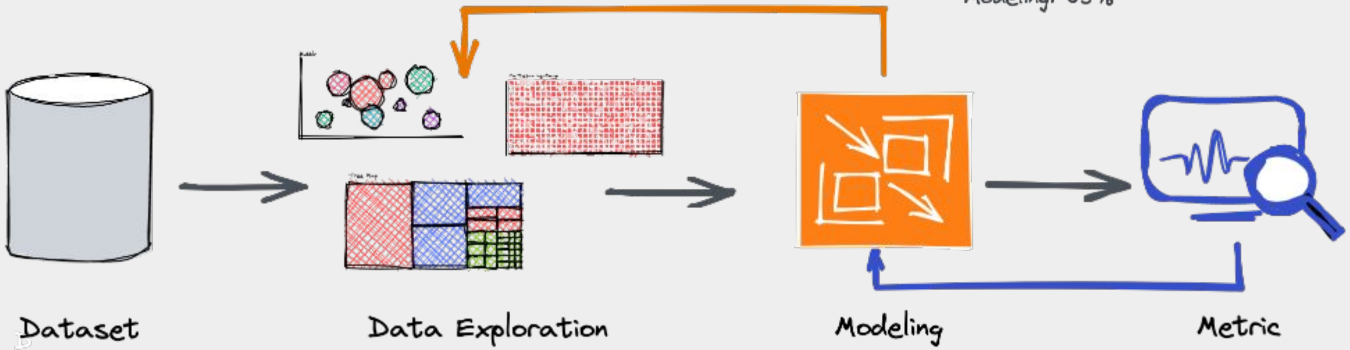
Academic/competitions settings vs real-world applications

Introduction to Machine Learning Operations (MLOps)

Academic/competitions settings

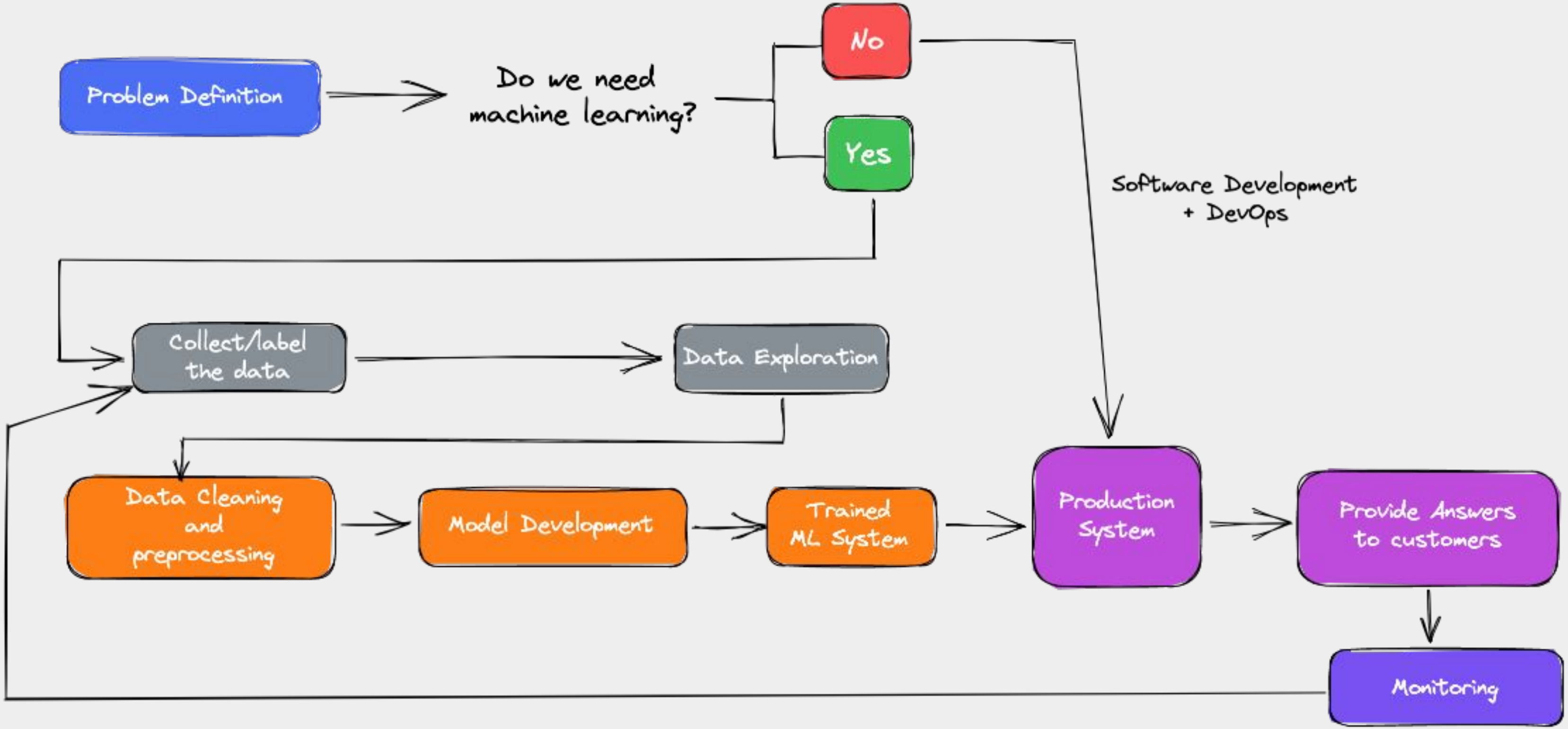
kaggle





Introduction to Machine Learning Operations (MLOps)

Machine Learning in the Wild



Introduction to Machine Learning Operations (MLOps)

Consequences of having the problem into production

Production

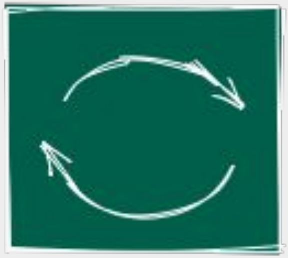
A model is 100% useless until it is in production

Usability

70% accuracy in production is infinitely better than 90% accuracy that can`t be used

Dependability

Avoid performance drift (monitoring, continuous trainning)



Reproducibility

The process must be transparent and repeatable

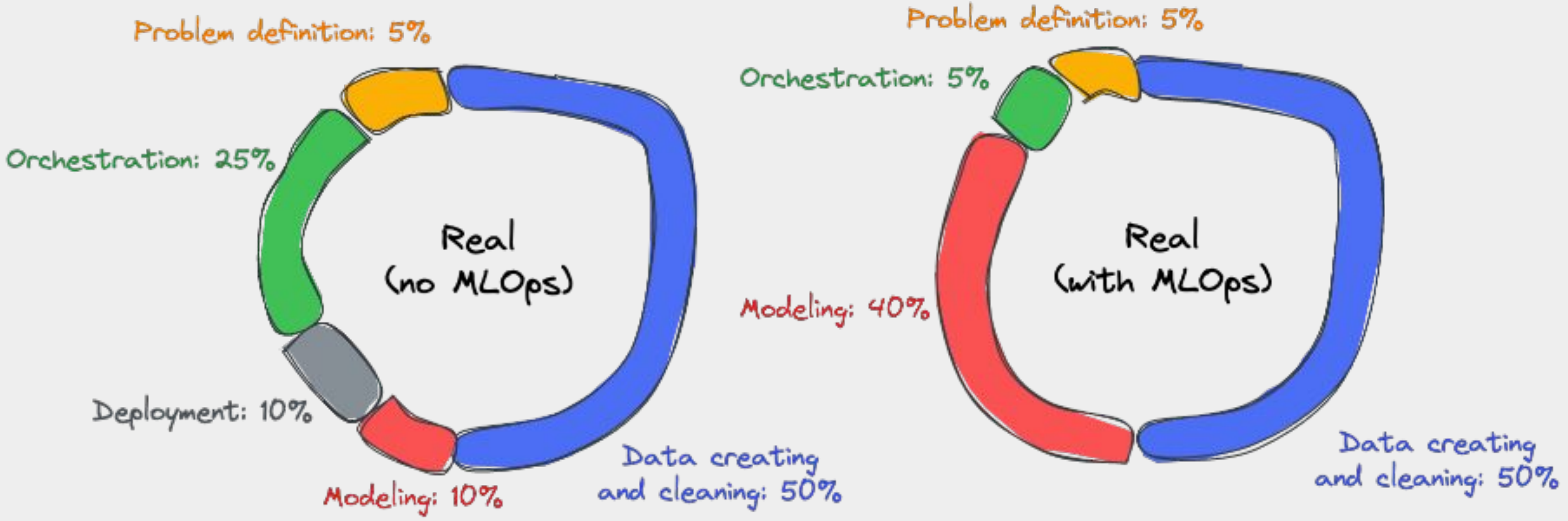
Introduction to Machine Learning Operations (MLOps)

So what is MLOps?

"A set of best practices and methods for an efficient end-to-end development and operation of performant, scalable, reliable, automated and reproducible ML solutions in a real production setting"

Introduction to Machine Learning Operations (MLOps)

So what is MLOps?



Introduction to Machine Learning Operations (MLOps)

Recap

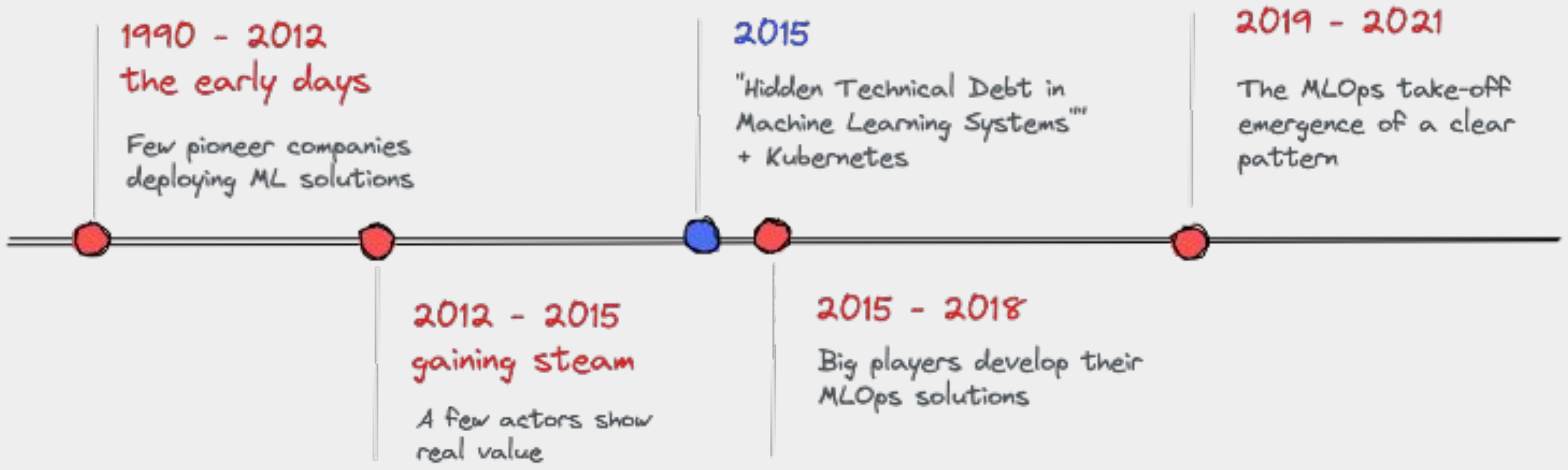
Academic or Competition

1) The dataset is fixed
2) Most dev time is spent in improving the model
3) Maximum performance on one or more metrics (e.g accuracy) is the most important aspect

Real-World Producton-Oriented model development

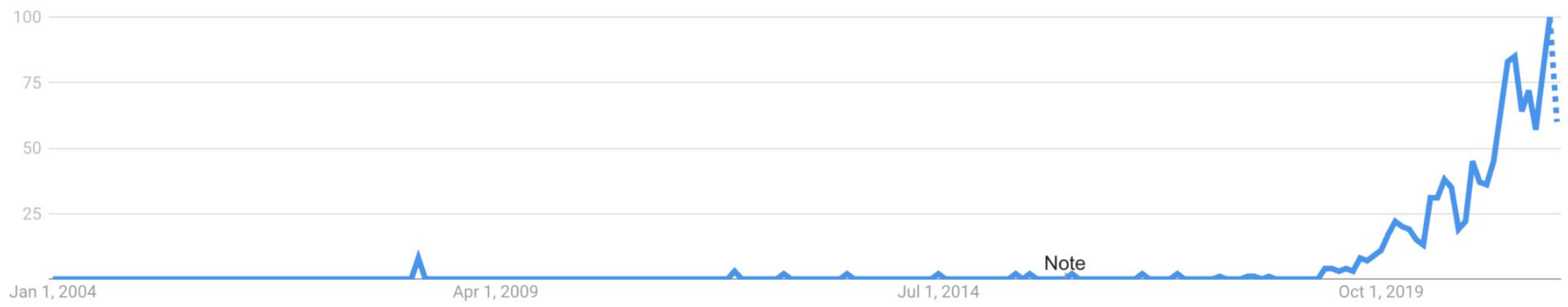
1) A careful problem definition is the first step towards a success project
2) The objective is a model with the best performance that is compatible with the constraints of the production system within in the allocated time and money budget
3) Monitoring after deployment and retrain, when necessary.

A brief history of MLOps



A brief history of MLOps

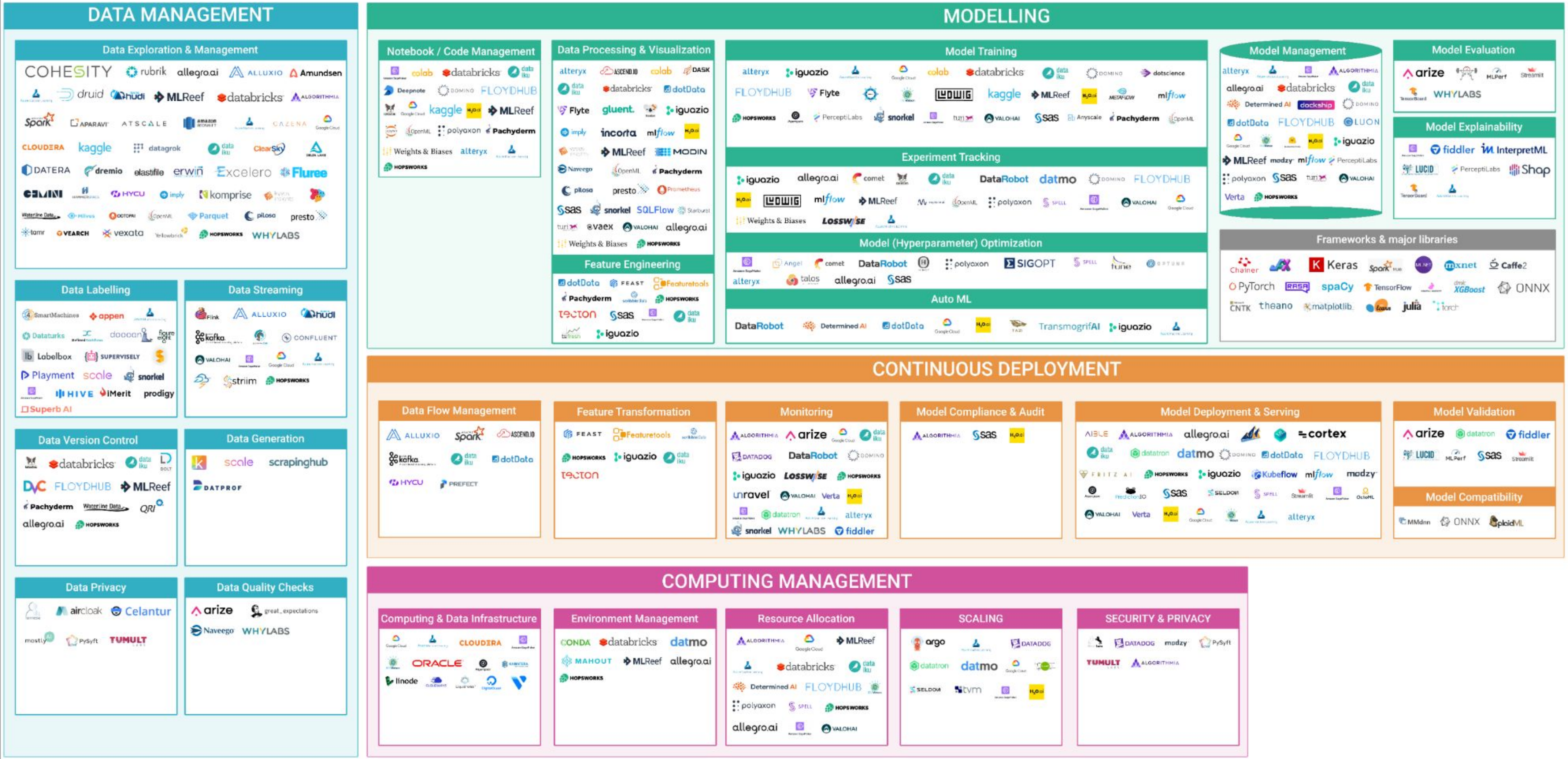
Interest in MLOps: explosion in MLOps tools



Source: Google Trends

Machine Learning tools & platforms landscape - v.1.1 February 2021

Presented by MLReef



https://about.mlreef.com/blog/global-mlops-and-ml-tools-landscape

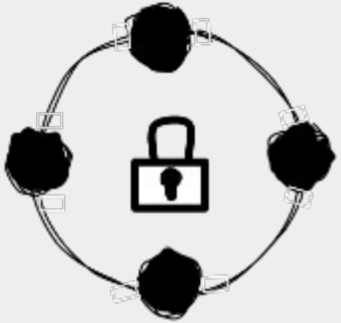
Criteria for choosing tools [in this course]

Tools & Environment

Battle-tested tools

Limited infrastructure

Simplicity



Focus on big picture

Free or freemium

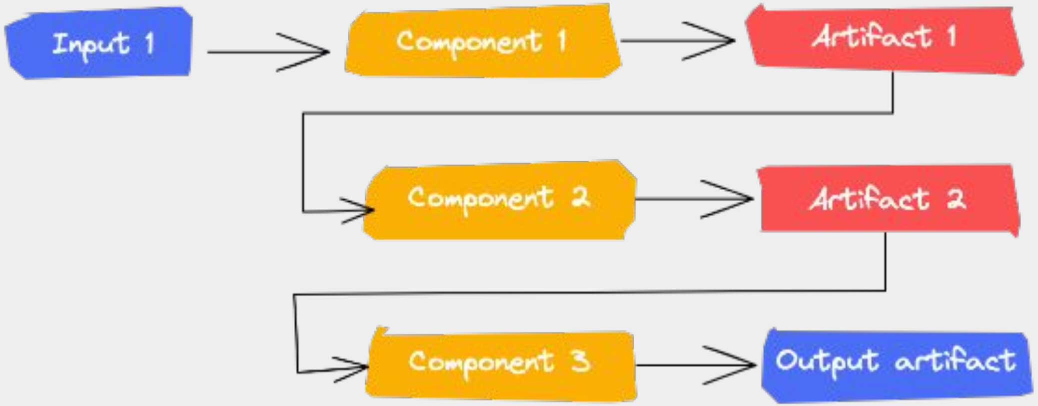
	Suggestion	Alternative tools
Code Tracking	Github	Gitlab, Bitbucket
Experiment Tracking	Weights & Biases	MLflow, Neptune, ...
Artifact Tracking	Weights & Biases	MLflow, Neptune, ...
Model Repository	Weights & Biases	MLflow, Paperspace
ML pipeline	MLflow + Hydra	Kuberflow, Metaflow, TFX
Environment Isolaton	conda	docker
Orchestration	MLflow + Hydra	k8s, AWS, GCP, Azure
Modeling	scikit-learn, TF	Pytorch

Machine Learning Pipelines

What is Machine Learning Pipelines?

A sequence of independent, modular and reusable components that automates the ML workflow.

It can be represented as a Direct Acyclic Graph (DAG)



Why use Machine Learning Pipelines?

The three levels of MLOps

	Dev. target	Retraining	Team size	Company	Prod. ready	Reusability	Infrastr. needs	Difficulty
Level 0	model	difficult, manual	1-5	Proof of concept				
Level 1	pipeline	easy, manual or triggered	1-20	Small/ medium				
Level 2	pipeline*	easy, automatic	>10	Medium or large				

https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Comparison Of the three Levels of MLOps

Project	MLOps Level
You are working on a startup idea centered on ML, and want show that your idea can work as soon as possible	0
You work in a large company with more than 50 ML models in production. You are going to build model 51.	2
You are finishing a class and you are working on the capstone project of the class. The project is fairly easy and you have only a few days to complete it.	0
Your startup has plans to move from one model in production to ten models in production within the next year. You wan to step up MLOps game.	1
You are finishing a class and you are working on the capstone project of the class. The project is pretty complex, and you plan to spend a month on it.	1

https://realpython.com/command-line-interfaces-python-argparse/

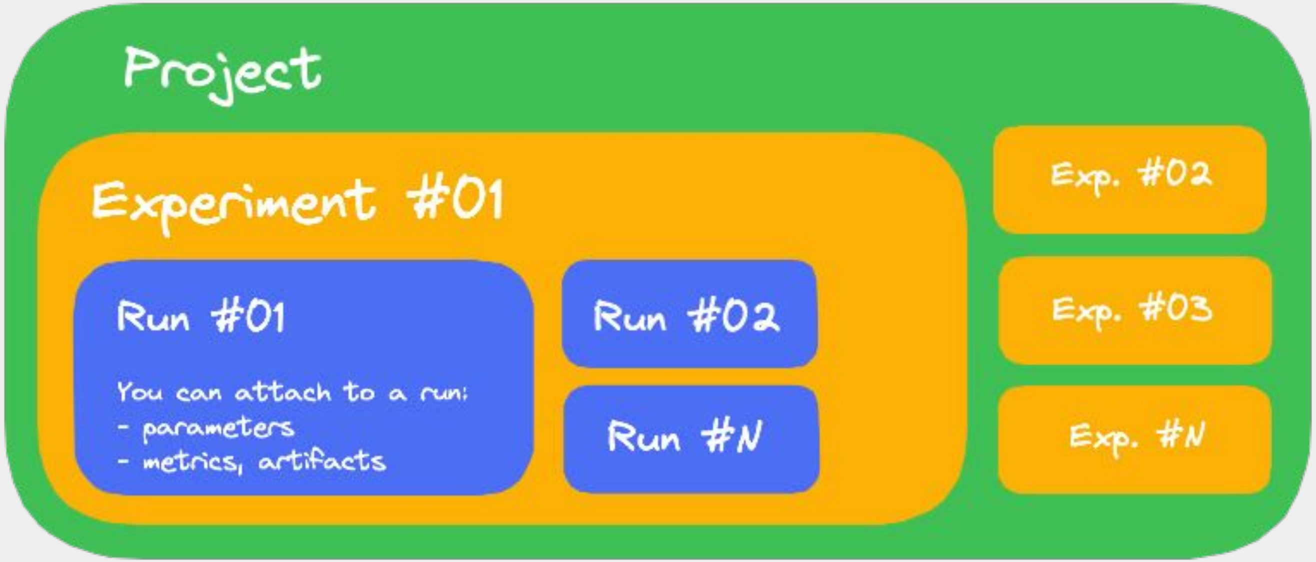
```python
import argparse

parser = argparse.ArgumentParser(
        description="This is a tutorial on argparse")

    # add the argument artifact_name
    parser.add_argument("--artifact_name",
                        type=str,
                        help="Name and version of artifact",
                        required=True)
```

```
❯ grep -h
usage: grep [-abcDEFGHhIiJLlmnOoqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
        [-e pattern] [-f file] [--binary-files=value] [--color=when]
        [--context[=num]] [--directories=action] [--label] [--line-buffered]
        [--null] [pattern] [file ...]
```

Versioning Data and Artifacts

Weights & Biases

```python
import wandb

wandb.init(
    project="my_project",
    group="experiment_1",
    job_type="data_cleaning")
```



Guided Exercise

Machine Learning Pipeline

Command Line Interface (CLI) +
Versioning Data and Artifacts

Weights & Biases

mlflow

https://github.com/mlflow/mlflow/

Integrations with:

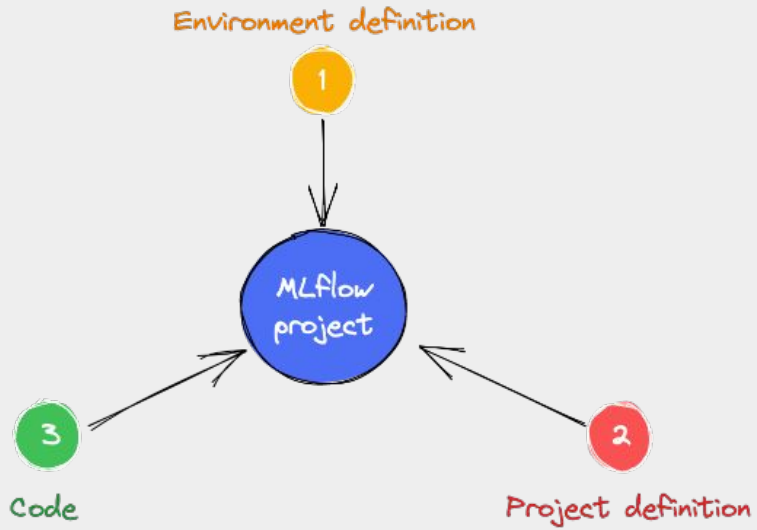
https://mlflow.org/

TensorFlow	PyTorch	Keras	APACHE Spark	scikit learn	RAPIDS
H_2O.ai	python	R	Java	mleap	ONNX
GLUON	XGBoost	LightGBM	spaCy	fast.ai	statsmodels
CatBoost	PYCARET	ALGORITHMIA	OPTUNA	RAY	CONDA
kubernetes	docker	Amazon SageMaker	Azure Machine Learning	Google Cloud	databricks

mlflow™

MLflow Tracking
MLflow Models
MLflow Model Registry
MLflow Projects

MLflow Projects is the component to package code in a Conda or Docker environment to ensure **reproducibility of code executions**.



mlflow

Defining the Project

The MLproject file (without .yml nor .yaml)

```
name: download_data
conda_env: conda.yml

entry_points:
  main:
    parameters:
      file_url:
        description: URL of the file to download
        type: uri
      artifact_name:
        description: Name for the W&B artifact that will be created
        type: str

    command: >-
      python download_data.py --file_url {file_url} \
                              --artifact_name {artifact_name}
  other_script:
    parameters:
        parameter_one:
          description: First parameter
          type: str
    command: julia other_script.jl {parameter_one}
```

Defining the Environment

The conda.yml file

```
name: download_data
channels:
  - conda-forge
  - defaults
dependencies:
  - requests=2.24.0
  - pip=20.3.3
  - hydra-core=1.0.6
  - pip:
      - wandb==0.10.21
       - mlflow==1.14.1
```

Running the Project

MLproject file

```
name: download_data
conda_env: conda.yml

entry_points:
  main:
    parameters:
      file_url:
        description: URL of the file to download
        type: uri
      artifact_name:
        description: Name for the W&B artifact
                     that will be created
        type: str

    command: >-
      python download_data.py --file_url {file_url} \
                              --artifact_name {artifact_name}
  other_script:
    parameters:
        parameter_one:
          description: First parameter
          type: str
    command: julia other_script.jl {parameter_one}
```

```
# Run default script from a local folder
mlflow run ./my_project -P file_url=https://.... \
            -P artifact_name=my_data.csv

# Run a different entry point from a local folder
mlflow run ./my_project \
            -e other_script  \
            -P parameter_one=27

# Run default script directly from Github
mlflow run git@github.com/mysuername/myrepo.git \
                  -P file_url=https://...   \
                  -P artifact_name=my_data.csv   \

# Run a specific release or tag
mlflow run git@github.com/mysuername/myrepo.git \
                 -v 2.2.8
                 -P file_url=https://...   \
                 -P artifact_name=my_data.csv   \
```

Introduction to YAML

A simple list
This is how you define a list in Python:

```
my_list = ['a word', 'b', 1, 3.5]
```

And this is how the same is represented in a YAML file:

```
- a word
- b
- 1
- 3.5
```

A nested list
Of course, in Python you can define lists containing other lists:

```
my_list = ['a word', [1, 2, 'a'], 1, 3.5]
```

This is how that data structure is represented in YAML:

```
- a word
- - 1
  - 2
  - a
- 1
- 3.5
```

Introduction to YAML

A key-value mapping (a dictionary)

This is how you define a dictionary in Python:

```
d = {'key_1': 1, 'key_2': "a string", 'another_key': 2.5}
```

And this is how the same is represented in a YAML file:

```
key_1: 1
key_2: a string
another_key: 2.5
```

A nested dictionary

In Python you can of course define nested dictionaries, i.e., dictionaries containing other dictionaries:

```
d = {
 "a": "a value",
 "b": {
   "c": 1.2,
   "d": 1,
   "e": "a string"
 },
 "c": 5
}
```

In YAML this is represented as:

```
a: a value
b:
  c: 1.2
  d: 1
  e: a string
c: 5
```

Introduction to YAML

Mixing dictionaries and lists
You can of course mix lists and dictionaries in Python:

```python
d = {
 "a": "a value",
 "b": {
   "c": 1.2,
   "d": 1,
   "e": "a string"
 },
 "c": [1, 2, "another string"]
}
```

Such a data structure is represented in YAML as:

```yaml
a: a value
b:
  c: 1.2
  d: 1
  e: a string
c:
  - 1
  - 2
  - another string
```

The YAML of conda.yml and MLproject*

conda.yml

```yaml
name: download_data
channels:
  - conda-forge
  - defaults
dependencies:
  - requests=2.24.0
  - pip=20.3.3
  - pip:
      - wandb==0.12.6
```

python version of conda.yml

```python
{
    "name": "download_data",
    "channels": [
        "conda-forge",
        "defaults"
    ],
    "dependencies": [
        "requests=2.24.0",
        "pip=20.3.3",
        {
            "pip": [
                "wandb==0.12.6"
            ]
        }
    ]
}
```

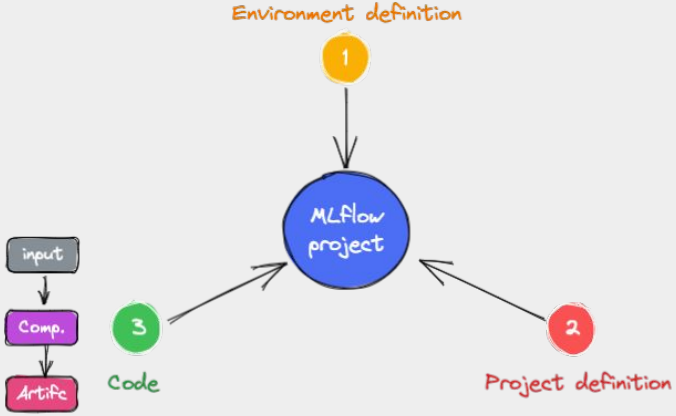
conversion yml to python

```python
import yaml

with open("conda.yml") as fp:
   d = yaml.safe_load(fp)
```

Guided Exercise

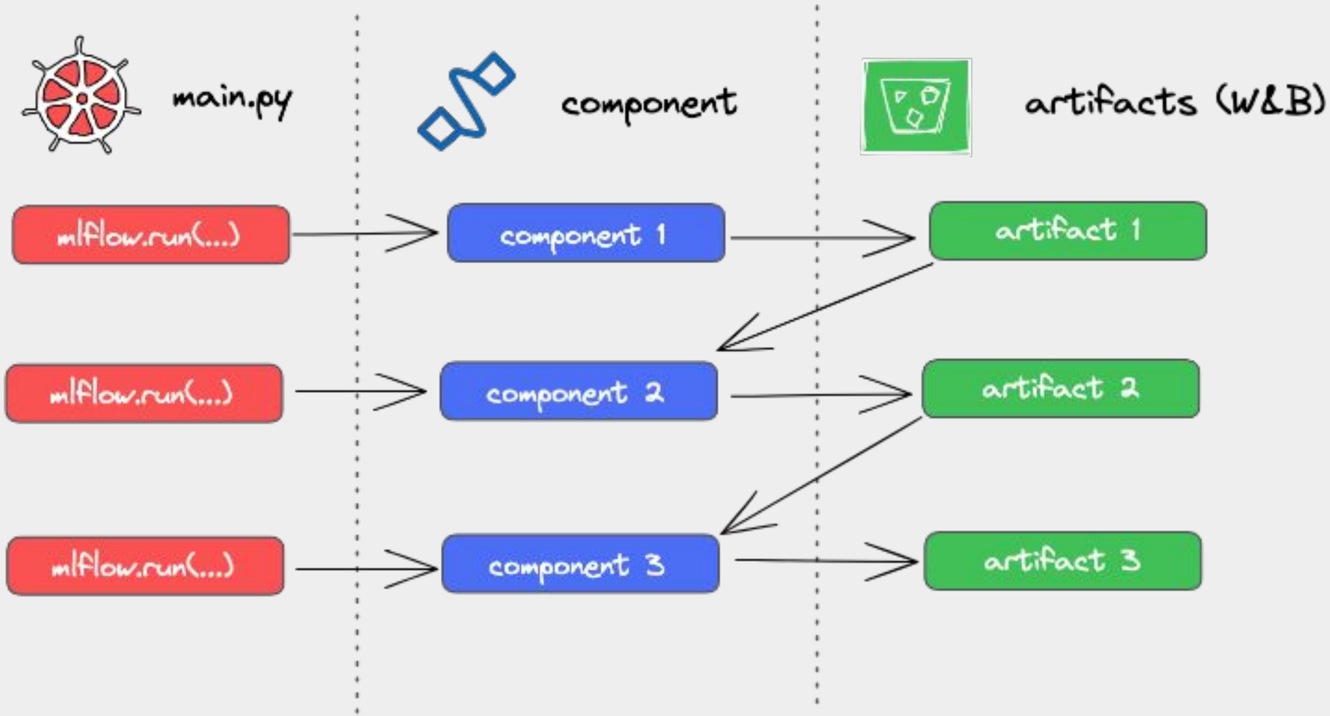
Machine Learning Pipeline

convert (Command Line Interface (CLI) + Versioning Data and Artifacts) to MLflow Component



Linking Together the Components

Writing a pipeline with mlflow

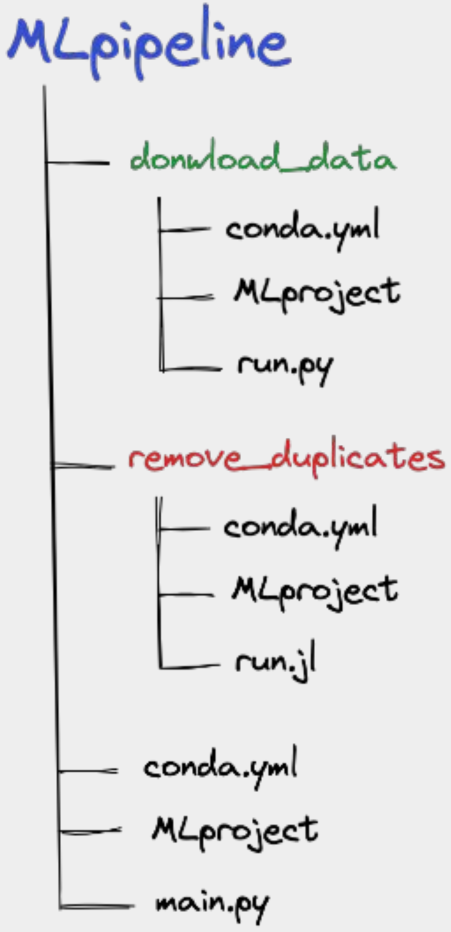


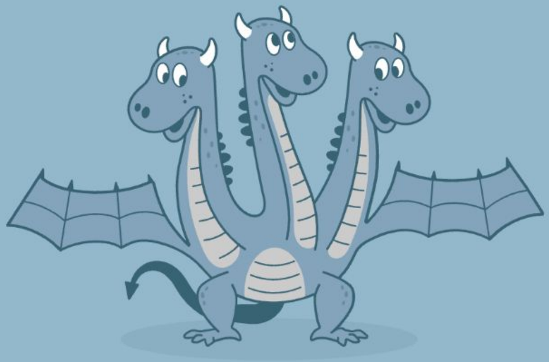
ML Pipeline in mflow

A project that calls other projects (the components)

main.py

```
import mlflow

mlflow.run(
  uri="download_data",
  entry_point="main",
  parameters={
    "file_url": "https://...",
    "output_artifact": "raw_data.csv"
  }
)

mlflow.run(
  uri="remove_duplicates",
  entry_point="main",
  parameters={
    "input_artifact": "raw_data.csv:latest",
    "output_artifact": "clean_data.csv"
  }
)
```





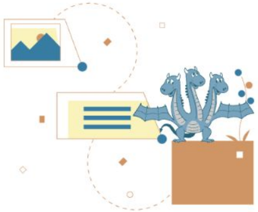
Hydra

A framework for elegantly configuring complex applications

Get Started

Star 5,094

https://hydra.cc/



No Boilerplate

Hydra lets you focus on the problem at hand instead of spending time on boilerplate code like command line flags, loading configuration files, logging etc.

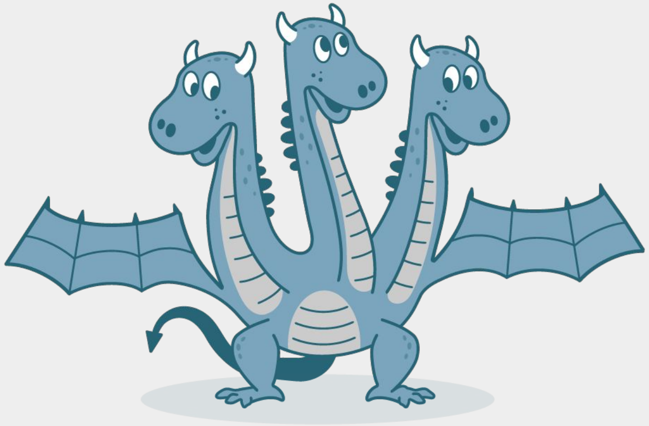
Powerful Configuration

With Hydra, you can compose your configuration dynamically, enabling you to easily get the perfect configuration for each run. You can override everything from the command line, which makes experimentation fast, and removes the need to maintain multiple similar configuration files.

Pluggable Architecture

Hydra has a pluggable architecture, enabling it to integrate with your infrastructure. Future plugins will enable launching your code on AWS or other cloud providers directly from the command line.

Hydra defines configuration files containing default values for all the parameters, so that it is easier to keep track of them and to know what they are for.



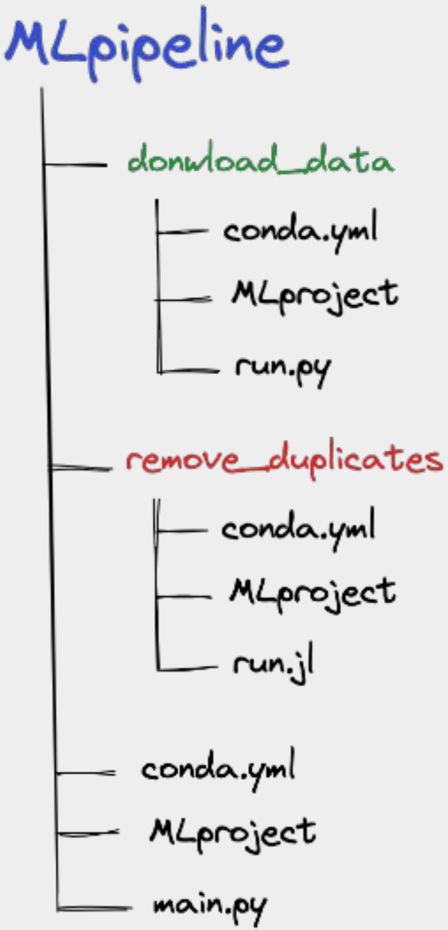
config.yaml

```yaml
main:
  project_name: my_project
  experiment_name: dev
data:
  train_data: "exercise_3/data_train.csv:latest"
random_forest_pipeline:
  random_forest:
    n_estimators: 100
    criterion: gini
    max_depth: null
```

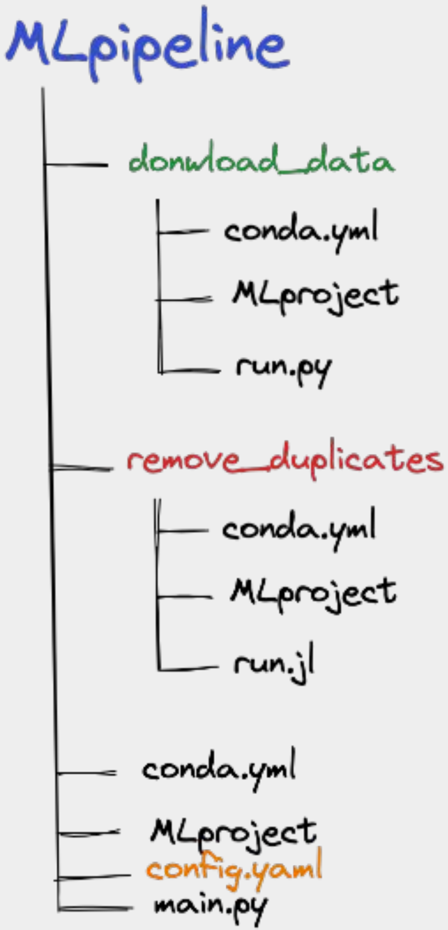
Parameters can be overridden from the command line, and multiple runs can be generated with one single command main.py

```python
import mlflow
import hydra

@hydra.main(config_name="config")
def go(config):
  # Now here config is a dictionary with
  # our configuration
  # For example, to access the parameter
  # train_data in the data
  # section we can just do
  train_data = config["data"]["train_data"]

  ...


if __name__=="__main__":
  go()
```

Before



After Hydra

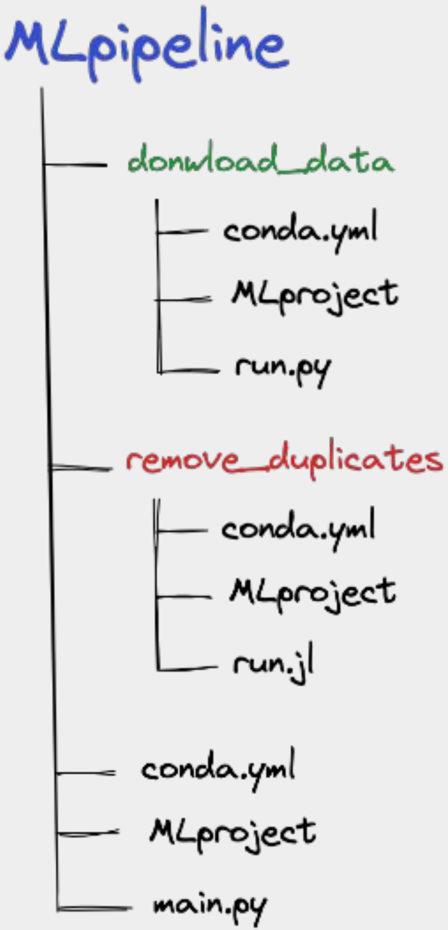


MLproject

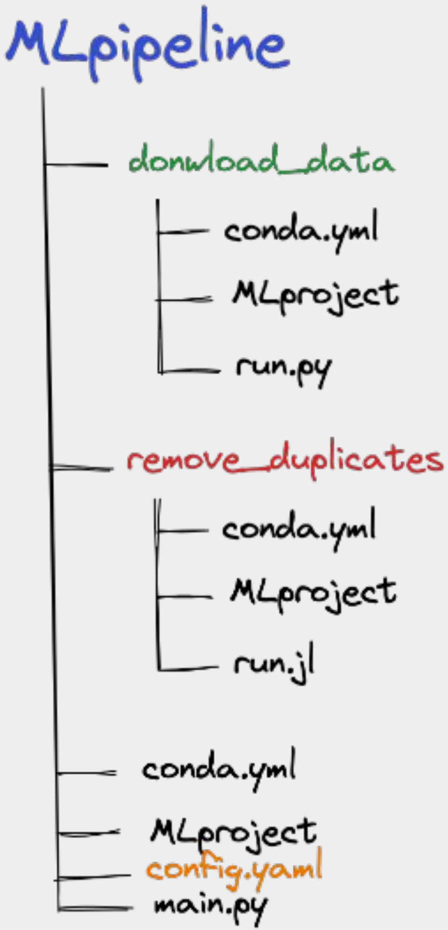
```yaml
name: main
conda_env: conda.yml

entry_points:
  main:
    parameters:
      hydra_options:
        description: Hydra parameters to override
        type: str
        default: ''
    command: >-
      python main.py $(echo {hydra_options})
```

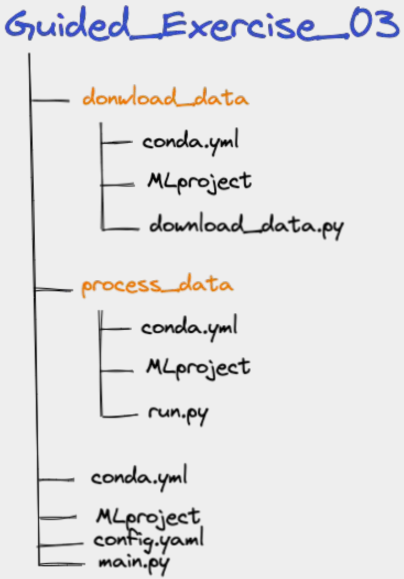
```
> mlflow run . \
          -P  hydra_options="main.experiment_name='Prod'"
```

Before



After Hydra





MLflow Pipeline + Hydra

Multiples mlflow workflows
input, component, artfacts