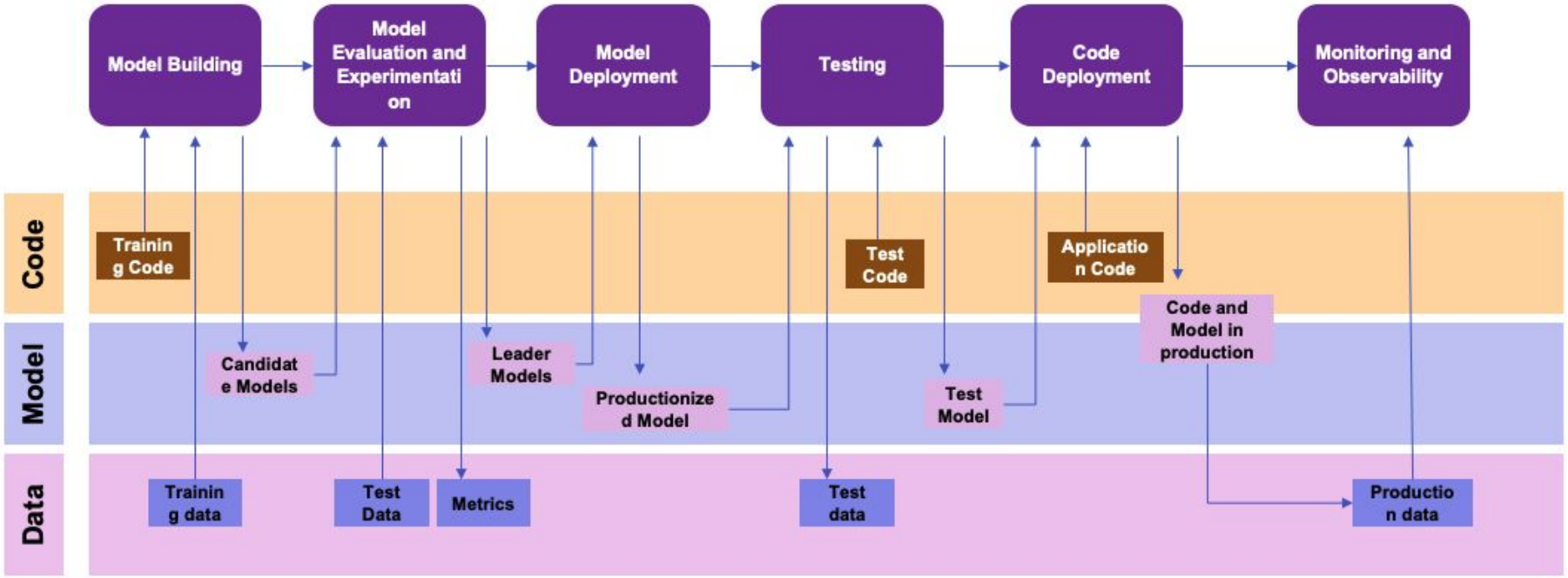
MLOps Hands-On

Part I: MLFlow

MLOps… how?


Model Building
Model Evaluation and Experimentation
Model Deployment
Testing
Code Deployment
Monitoring and Observability
Code
Model
Data
Training Code
Test Code
Application Code
Candidate Models
Leader Models
Productionized Model
Test Model
Code and Model in production
Training data
Test Data
Metrics
Test data
Production data

Hands-on idea and tooling

Target: to train a ML model implementing the ML lifecycle with **mlflow** and **dvc**

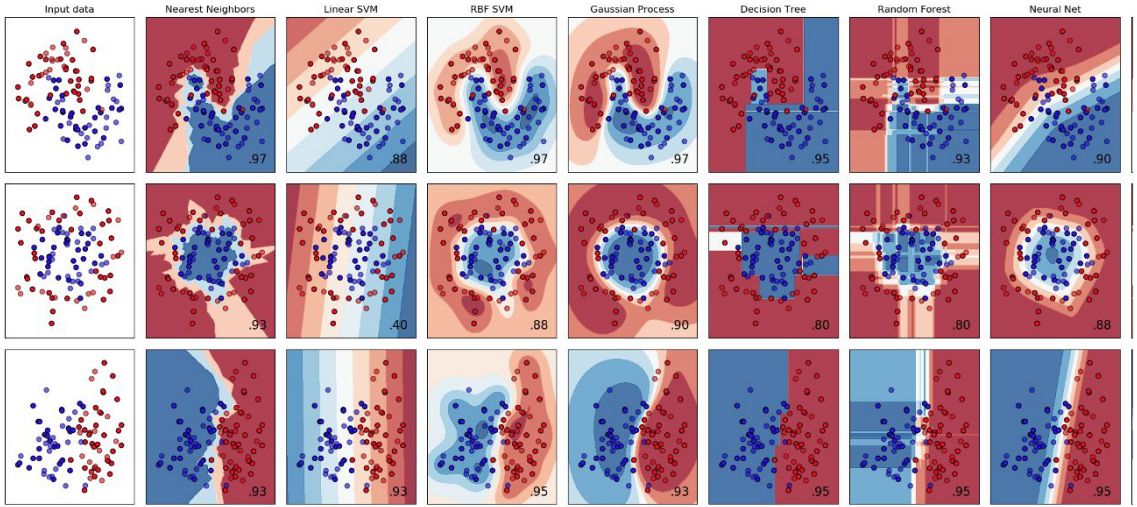
Features to accomplish:

- using python, train a model and register it
- control the model versioning
- register the model performance
- if there is any change in the code, retrain the model
- if there is any change in the data, retrain the model

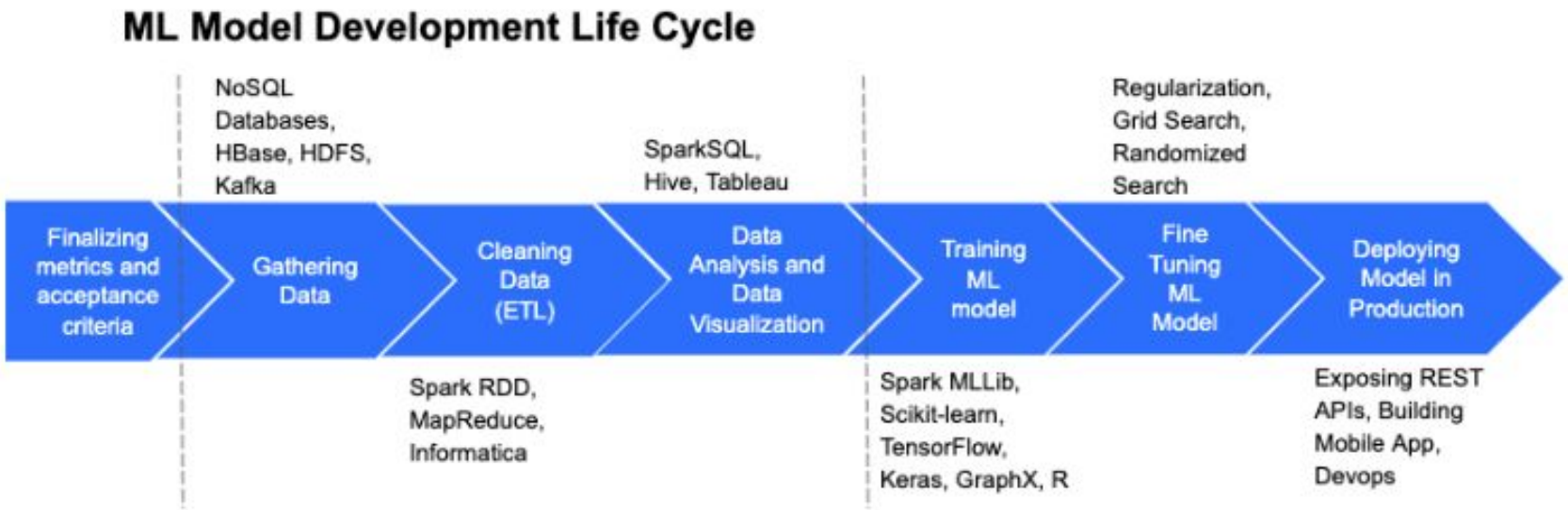
Tools and Frameworks

How can you track the models: MLFlow

- Remember, we want to track the models because…
  - Data always change
  - We want to try different algorithms and hyperparameters
  - The model is a binary digital asset, would be great to have it under control
  - .



ML Model Development Life Cycle



mlflow

The Machine Learning Workflow

Machine learning requires experimenting with a wide range of datasets, data preparation steps, and algorithms to build a model that maximizes some target metric. Once you have built a model, you also need to deploy it to a production system, monitor its performance, and continuously retrain it on new data and compare with alternative models.

Being productive with machine learning can therefore be challenging for several reasons:

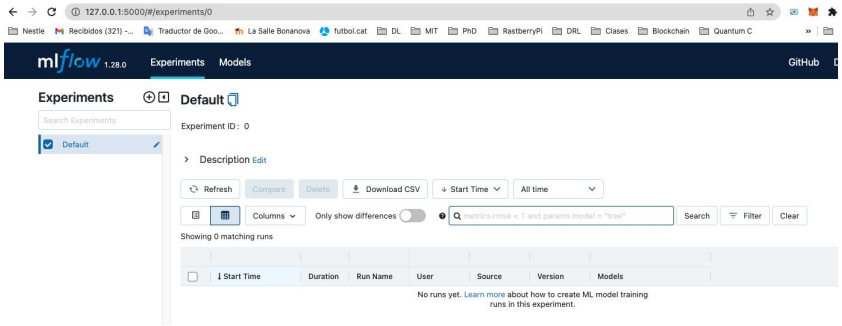
- **It's difficult to keep track of experiments.** When you are just working with files on your laptop, or with an interactive notebook, how do you tell which data, code and parameters went into getting a particular result?
- **It's difficult to reproduce code.** Even if you have meticulously tracked the code versions and parameters, you need to capture the whole environment (for example, library dependencies) to get the same result again. This is especially challenging if you want another data scientist to use your code, or if you want to run the same code at scale on another platform (for example, in the cloud).
- **There's no standard way to package and deploy models.** Every data science team comes up with its own approach for each ML library that it uses, and the link between a model and the code and parameters that produced it is often lost.
- **There's no central store to manage models (their versions and stage transitions).** A data science team creates many models. In absence of a central place to collaborate and manage model lifecycle, data science teams face challenges in how they manage models stages: from development to staging, and finally, to archiving or production, with respective versions, annotations, and history.

Moreover, although individual ML libraries provide solutions to some of these problems (for example, model serving), to get the best result you usually want to try *multiple ML libraries*. MLflow lets you train, reuse, and deploy models with any library and package them into reproducible steps that other data scientists can use as a "black box," without even having to know which library you are using.

Mlflow - Quickstart Lab

[Official doc]

1. Activate the conda environment
2. Install **mlflow**
3. Clone the *“quickstart”* repo: “`git clone https://github.com/mlflow/mlflow`”
4. Test it: run the mlflow ui: “`mlflow ui`”



MLFlow - Training a (simple) model without mlflow

```
warnings.filterwarnings("ignore")
np.random.seed(40)

# Read the wine-quality csv file from the URL
csv_url =\
    'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
try:
    data = pd.read_csv(csv_url, sep=';')
except Exception as e:
    logger.exception(
        "Unable to download training & test CSV, check your internet connection. Error: %s", e)

# Split the data into training and test sets. (0.75, 0.25) split.
train, test = train_test_split(data)

# The predicted column is "quality" which is a scalar from [3, 9]
train_x = train.drop(["quality"], axis=1)
test_x = test.drop(["quality"], axis=1)
train_y = train[["quality"]]
test_y = test[["quality"]]

# Set default values if no alpha is provided
if float(in_alpha) is None:
    alpha = 0.5
else:
    alpha = float(in_alpha)

# Set default values if no l1_ratio is provided
if float(in_l1_ratio) is None:
    l1_ratio = 0.5
else:
    l1_ratio = float(in_l1_ratio)

# Useful for multiple runs (only doing one run in this sample notebook)

# Execute ElasticNet
lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
lr.fit(train_x, train_y)

# Evaluate Metrics
predicted_qualities = lr.predict(test_x)
(rmse, mae, r2) = eval_metrics(test_y, predicted_qualities)

# Print out metrics
print("Elasticnet model (alpha=%f, l1_ratio=%f):" % (alpha, l1_ratio))
```

load data

transform data

Select model and hyperparameters

Train and evaluate the model

Lab: run it in your machines

- **Activate** your environment
- **Pull** the course repo
- run *jupyter notebook*
- find the notebook: *mlflow_labs/elasticnet_wine/train_without_mlflow.ipynb*

… and now… what?

Please, think about how you could save and send one version of your model (the best one) to the Production Team

MLflow - injecting management in the training

```python
# Useful for multiple runs (only doing one run in this sample notebook)
with mlflow.start_run():
    # Execute ElasticNet
    lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
    lr.fit(train_x, train_y)

    # Evaluate Metrics
    predicted_qualities = lr.predict(test_x)
    (rmse, mae, r2) = eval_metrics(test_y, predicted_qualities)

    # Print out metrics
    print("Elasticnet model (alpha=%f, l1_ratio=%f):" % (alpha, l1_ratio))
    print("  RMSE: %s" % rmse)
    print("  MAE: %s" % mae)
    print("  R2: %s" % r2)

    # Log parameter, metrics, and model to MLflow
    mlflow.log_param("alpha", alpha)
    mlflow.log_param("l1_ratio", l1_ratio)
    mlflow.log_metric("rmse", rmse)
    mlflow.log_metric("r2", r2)
    mlflow.log_metric("mae", mae)
```

Train and evaluate the model

Train and evaluate the model

Lab time…

Adapt the code in ***train_diabetes_without_mlflow.py*** to register

- the model
- the performance
- the png with the figure

into your MLFlow local server

https://www.mlflow.org/docs/latest/index.html

Ok, now…

- we can have under control all our:
  - experiments - hyperparameter tuning, algorithm selections, …
  - models - binaries, versions, associated experiments
  - environments - libraries, versions, …

Cookiecutter: Let’s have a common structure

1. run: `pip install cookiecutter`
2. run: `cookiecutter https://github.com/drivendata/cookiecutter-data-science`

```
(labs_docker) ➜  dvc_lab_test_01 ls -la
total 80
drwxr-xr-x  18 ESMoraEn  staff   576 Sep  9 16:34 .
drwxr-xr-x   3 ESMoraEn  staff    96 Sep  9 16:34 ..
-rw-r--r--   1 ESMoraEn  staff   459 Sep  9 16:34 .env
-rw-r--r--   1 ESMoraEn  staff  1003 Sep  9 16:34 .gitignore
-rw-r--r--   1 ESMoraEn  staff  1081 Sep  9 16:34 LICENSE
-rw-r--r--   1 ESMoraEn  staff  4683 Sep  9 16:34 Makefile
-rw-r--r--   1 ESMoraEn  staff  2895 Sep  9 16:34 README.md
drwxr-xr-x   6 ESMoraEn  staff   192 Sep  9 16:34 data
drwxr-xr-x   8 ESMoraEn  staff   256 Sep  9 16:34 docs
drwxr-xr-x   3 ESMoraEn  staff    96 Sep  9 16:34 models
drwxr-xr-x   3 ESMoraEn  staff    96 Sep  9 16:34 notebooks
drwxr-xr-x   3 ESMoraEn  staff    96 Sep  9 16:34 references
drwxr-xr-x   4 ESMoraEn  staff   128 Sep  9 16:34 reports
-rw-r--r--   1 ESMoraEn  staff   103 Sep  9 16:34 requirements.txt
-rw-r--r--   1 ESMoraEn  staff   212 Sep  9 16:34 setup.py
drwxr-xr-x   7 ESMoraEn  staff   224 Sep  9 16:34 src
-rw-r--r--   1 ESMoraEn  staff   632 Sep  9 16:34 test_environment.py
-rw-r--r--   1 ESMoraEn  staff    50 Sep  9 16:34 tox.ini
(labs_docker) ➜  dvc_lab_test_01
```

Docs

Create your github repo

1. go to github and create a repo
2. Run (with your repo name):

```
echo "# dvc_lab_test_01" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/enriquemoraayala/dvc_lab_test_01.git
git push -u origin main
```

3. Review the results