# COMPONENTS IN MLOPS

# Data Management and Processing

**Data Versioning:** Tools like DVC, Pachyderm, or Delta Lake track changes in datasets.

**Data Validation:** Ensures data consistency and integrity before training using tools like Great Expectations or TensorFlow Data Validation (TFDV).

**Feature Engineering & Feature Store:** Organizes reusable features for multiple models using platforms like Feast or Tecton.

# Modular Coding & Code Versioning

**Modular Code Structure:** Breaks down ML code into reusable modules for data preprocessing, training, and inference.

**Code Versioning:** Uses Git and DVC to track code and data versions for reproducibility.

**Automated Testing:** Implements unit tests (e.g., pytest) for data transformations, model predictions, and integrations.

main.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("data.csv")
X = df.drop(columns=["target"])
y = df["target"]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc}")
```

**Hard to reuse:** If we need another dataset, we must rewrite code.

**No separation of concerns:** Data loading, preprocessing, training, and evaluation are mixed together.

**Difficult debugging:** If something breaks, we have to go through the entire script.

**model_trainer.py**

```python
from sklearn.ensemble import RandomForestClassifier

def train_model(X_train, y_train, n_estimators=100, random_state=42):
    model = RandomForestClassifier(n_estimators=n_estimators, random_state=random_state)
    model.fit(X_train, y_train)
    return model
```

**evaluate.py**

```python
from sklearn.metrics import accuracy_score

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return accuracy_score(y_test, y_pred)
```

**data_loader.py**

```python
import pandas as pd
from sklearn.model_selection import train_test_split

def load_data(filepath):
    df = pd.read_csv(filepath)
    X = df.drop(columns=["target"])
    y = df["target"]
    return X, y

def split_data(X, y, test_size=0.2, random_state=42):
    return train_test_split(X, y, test_size=test_size, random_state=random_state)
```

**main.py**

```python
from data_loader import load_data, split_data
from model_trainer import train_model
from evaluator import evaluate_model

# Load and preprocess data
X, y = load_data("data.csv")
X_train, X_test, y_train, y_test = split_data(X, y)

# Train model
model = train_model(X_train, y_train)

# Evaluate
accuracy = evaluate_model(model, X_test, y_test)
print(f"Model Accuracy: {accuracy}")
```

# Experiment Tracking & Model Development

**Experiment Tracking:** Logs hyperparameters, datasets, and model metrics using MLflow, Weights & Biases, or Neptune.

**Automated Hyperparameter Tuning:** Uses Optuna, Ray Tune, or Hyperopt for optimizing model performance.

**Model Registry:** Stores trained models with metadata, ensuring traceability (MLflow Model Registry, Kubeflow Model Store).

# Model Deployment & Serving

**Model Packaging:** Converts models into deployable artifacts using Docker, TensorFlow Serving, or TorchServe.

**Model Deployment Strategies:**

**A/B Testing:** Compares two models in production to determine the best one.

**Canary Deployment:** Gradually rolls out a new model to a small portion of traffic.

**Shadow Deployment:** Runs the new model alongside the old one without affecting real users.

# Model Monitoring & Maintenance

**Performance Monitoring:** Tracks metrics like latency, accuracy, and throughput (Prometheus, Grafana, or Aws Sagemaker etc).

**Drift Detection:** Detects changes in input data (data drift) or model performance (concept drift) using tools like WhyLabs or Fiddler.

**Model Retraining & Continuous Learning:** Automates model retraining when performance degrades (Kubeflow Pipelines, Airflow).

# CI/CD for Machine Learning

**Continuous Integration (CI):** Automates testing and validation of ML code and models (Jenkins, GitHub Actions, AWS Tools).

**Continuous Deployment (CD):** Deploys models automatically to production environments (ArgoCD, Azure DevOps).

**Infrastructure as Code (IaC):** Automates provisioning of ML infrastructure using Terraform, AWS CloudFormation.