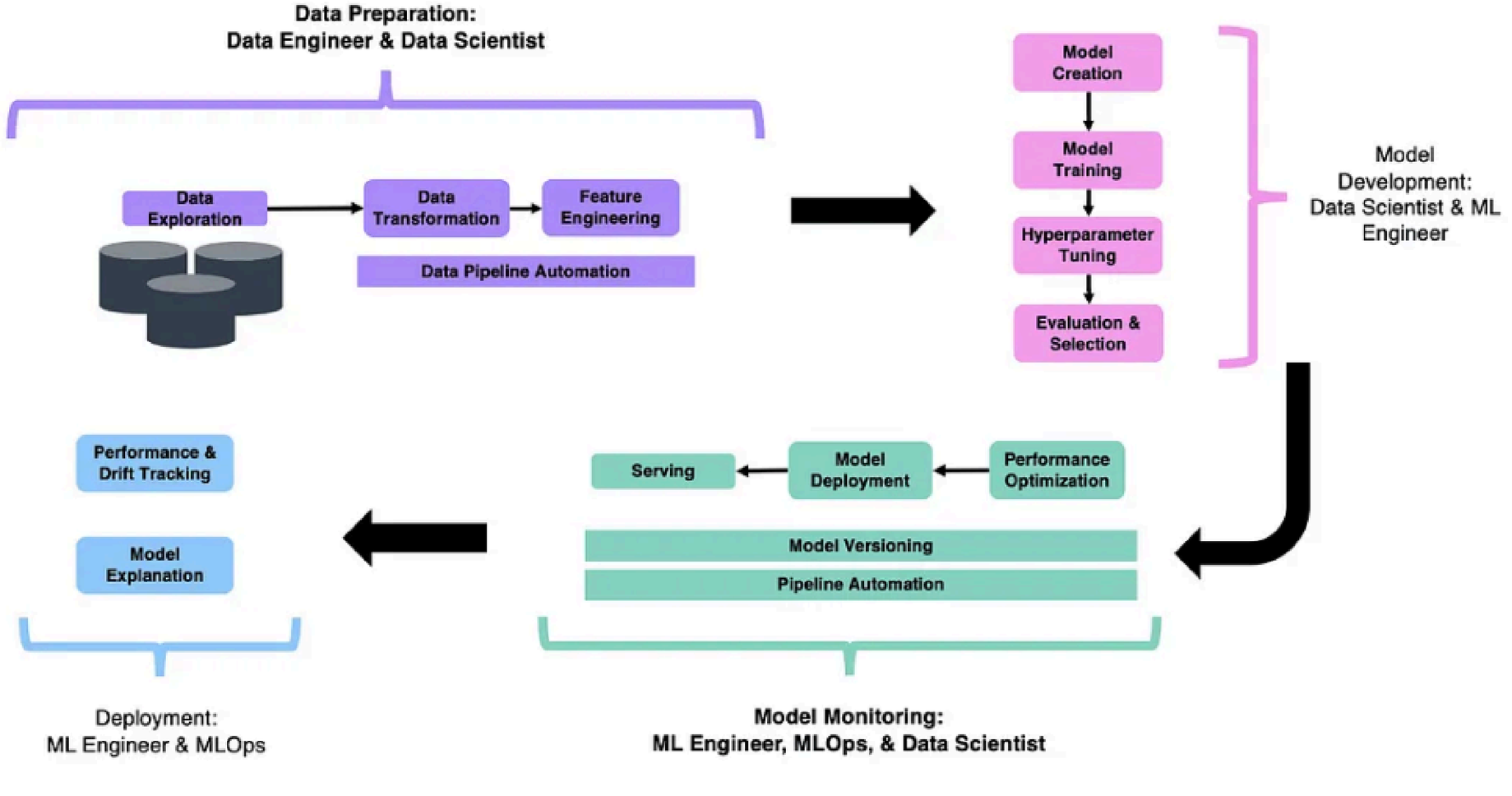
MACHINE LEARNING OPERATIONS

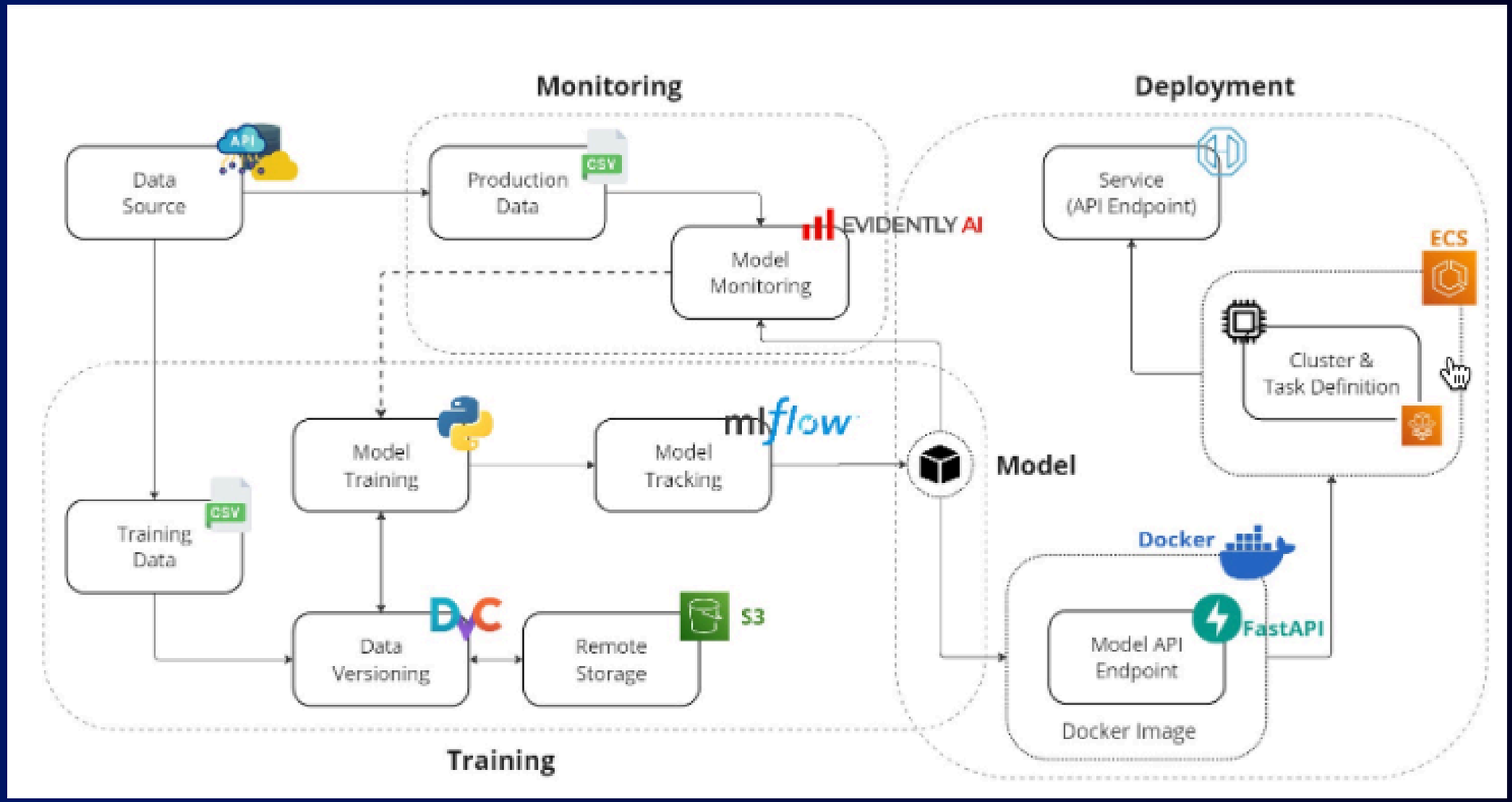
WEEK 7

UNLIMITED

Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**

Machine Learning Project Lifecycle




Monitoring
Deployment
Data Source
API
Production Data
CSV
EVIDENTLY AI
Model Monitoring
Service (API Endpoint)
ECS
Cluster & Task Definition
Model Training
Model Tracking
mlflow
Model
Training Data
CSV
Docker
Data Versioning
DVC
Remote Storage
S3
Model API Endpoint
FastAPI
Docker Image
Training

LAB 1. OmegaConf

Configuration Management in Python with OmegaConf

Why Use Configuration Management in Python Projects?

- **Maintainability:**
  - Separates configuration from code for easier maintenance and modularity.
  - Clearer separation of concerns.
- **Scalability:**
  - Manages complex systems and scaling efficiently.
  - Ensures consistency and reliability.
- **Error Prevention:**
  - Centralizes configuration data with validation mechanisms.
  - Reduces errors from incorrect values.
- **Flexibility:**
  - Allows easy switching between environments (development, staging, production).
  - Simplifies sharing configurations across instances.
- **Version Control:**
  - Tracks configuration changes over time.
  - Enables reverting to previous states and maintaining consistency.
- **Security:**
  - Separates sensitive data (e.g., API keys) from code.
  - Prevents accidental exposure and enforces validation.
- **Testing & Debugging:**
  - Enables test environments with tailored configurations.
  - Supports automated security testing.

Tools: Hydra and OmegaConf

- **OmegaConf**: Hierarchical configuration system for Python, merging YAML, dataclasses, and CLI inputs.

config.yaml

```yaml
server:
  host: localhost
  port: 80
client:
  url: http://${server.host}:${server.port}/
  server_port: ${server.port}
```

app.py

```python
from omegaconf import OmegaConf

# Load the YAML configuration file correctly
config = OmegaConf.load("config.yaml")

# Access values from the configuration
print("Server Host:", config.server.host)
print("Server Port:", config.server.port)
print("Client URL:", config.client.url)
print("Client Server Port:", config.client.server_port)
```

```
LAB01 % python main.py
Server Host: localhost
Server Port: 80
Client URL: http://localhost:80/
Client Server Port: 80
```

LAB: OmegaConf on Ubuntu

This lab focuses on working with `OmegaConf` for configuration management using YAML files in a Python environment. It assumes you are working on an Ubuntu system without Python pre-installed.

Objective:

- Learn how to set up Python in an Ubuntu system without Python installed.
- Use `OmegaConf` to load, modify, and merge YAML configurations.
- Work with environment variable resolutions using `OmegaConf`.

Prerequisites:

- Ubuntu system with no Python pre-installed.
- Internet access.
- Basic familiarity with the command line.

Lab Setup:

1. **Install Python and Pip:**

```
sudo apt update
sudo apt install python3 python3-pip -y
```

2. **Clone the Git Repository:**

```
git clone https://github.com/Tuchsanai/MLOps_Class.git
cd MLOps_Class/02_Data Versioning_and_pipeline /week07/LAB01
```

3. **Create a Python Virtual Environment:**

```
python3 -m venv venv
source venv/bin/activate
```

4. **Install Required Packages:**

```
pip install -r requirements.txt
```

If no `requirements.txt` exists, create it with the following content:

```
omegaconf
```

Lab Activities:

Load Configuration:

- Run the `app.py` script and observe how the `config.yaml` values are loaded and displayed.

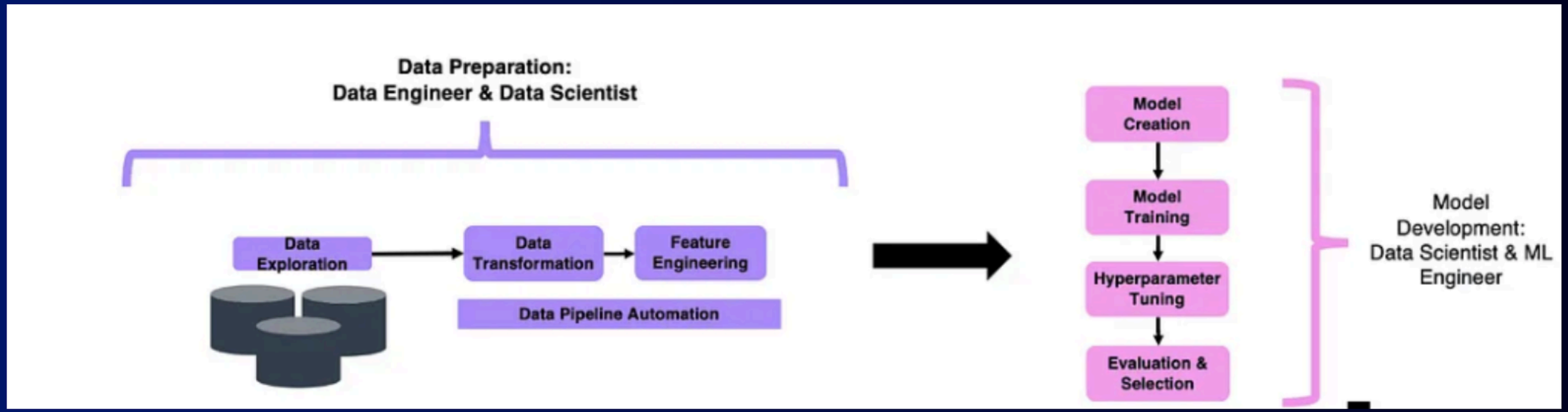
Running the Lab:

```
python app.py
```

Expected Output:

```
Server Host: localhost
Server Port: 80
Client URL: http://localhost:80/
Client Server Port: 80
```

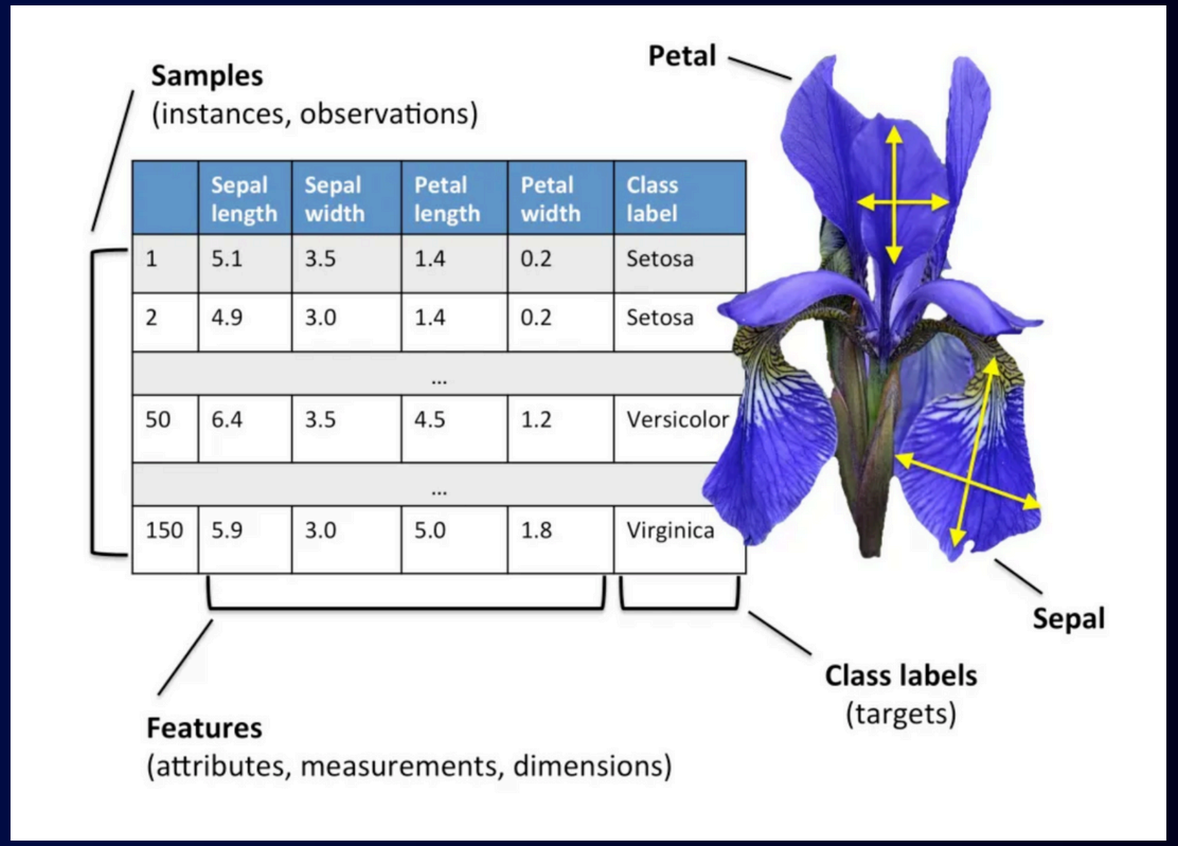
LAB 2 : Semi Auto MLCycle


Data Preparation:
Data Engineer & Data Scientist
Data Exploration
Data Transformation
Feature Engineering
Data Pipeline Automation
Model Creation
Model Training
Hyperparameter Tuning
Evaluation & Selection
Model Development:
Data Scientist & ML Engineer

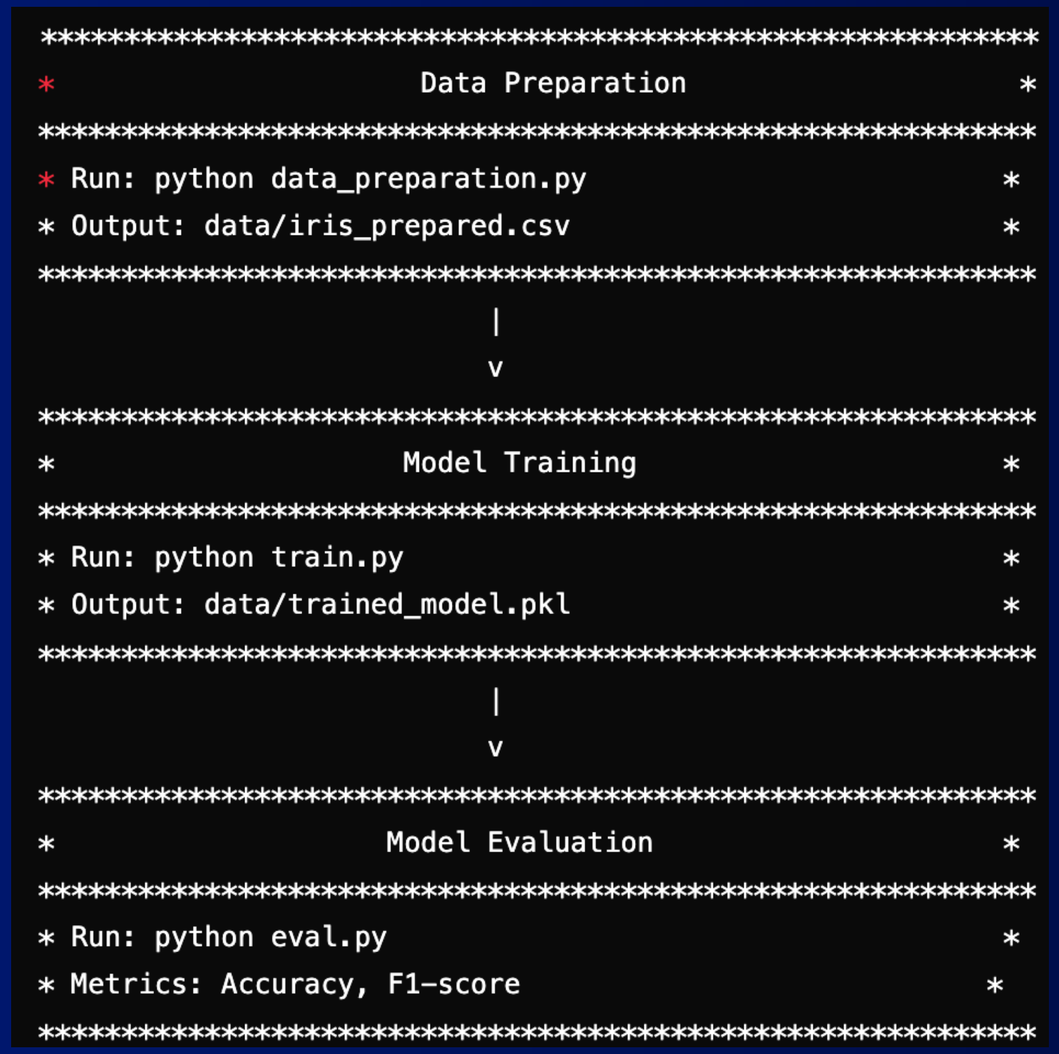
Machine learning-Iris dataset

Iris Flower:

Iris is the family in the flower which contains the several species such as the *iris.setosa,iris.versicolor,iris.virginica,etc.*



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



1. **Data Preparation:**
   - Run: `python data_preparation.py`
   - Output: `data/iris_prepared.csv`
2. **Model Training:**
   - Run: `python train.py`
   - Output: `data/trained_model.pkl`
3. **Model Evaluation:**
   - Run: `python eval.py`
   - Metrics: Accuracy, F1-score

OmegaConf

1. **Data Preparation:**
   - Run: `python data_preparation.py`
   - Output: `data/iris_prepared.csv`
2. **Model Training:**
   - Run: `python train.py`
   - Output: `data/trained_model.pkl`
3. **Model Evaluation:**
   - Run: `python eval.py`
   - Metrics: Accuracy, F1-score
4. **DVC Pipeline Execution:**
   - Command: `dvc repro`
   - Automates all stages.

4. Configuration File: `config.yaml`

Below is an **example** configuration file using **OmegaConf**. You can customize it as needed.

```yaml
data:
  path: "./data/"
  file_name: "iris.csv"
  target_column: "species"

model:
  type: "logistic_regression"  # Options: "logistic_regression" or "decision_tree"
  hyperparameters:
    # LogisticRegression params
    max_iter: 200
    # DecisionTreeClassifier params
    # max_depth: 3

training:
  test_size: 0.2
  random_state: 42

evaluation:
  metrics:
    - "accuracy"
    - "f1"
```

1. **Data Preparation:**
   - Run: `python data_preparation.py`
   - Output: `data/iris_prepared.csv`
2. **Model Training:**
   - Run: `python train.py`
   - Output: `data/trained_model.pkl`
3. **Model Evaluation:**
   - Run: `python eval.py`
   - Metrics: Accuracy, F1-score
4. **DVC Pipeline Execution:**
   - Command: `dvc repro`
   - Automates all stages.

Data Preparation: `data_preparation.py`

Below is a more **extensive** data preparation script that includes common cleaning tasks: removing duplicates, dropping NaNs, handling outliers via IQR, and optional label encoding for the target column.

```python
# data_preparation.py

import os
import pandas as pd
import numpy as np
from omegaconf import OmegaConf
from sklearn.preprocessing import LabelEncoder

def remove_outliers_iqr(df, column, threshold=1.5):
    """
    Removes rows from df where the specified column has outliers
    based on the IQR (Interquartile Range) method.

    threshold=1.5 is a common default; 3.0 is more conservative.
    """
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR

    # Filter out rows that are outside [lower_bound, upper_bound]
    mask = (df[column] >= lower_bound) & (df[column] <= upper_bound)
    df_cleaned = df[mask]
    return df_cleaned

def main():
    # 1. Load config
    config = OmegaConf.load("config.yaml")

    # 2. Read the raw data
    data_path = os.path.join(config.data.path, config.data.file_name)
    df = pd.read_csv(data_path)
    print("[INFO] Loaded raw data shape:", df.shape)

    # 3a. Remove duplicate rows (if any)
    initial_count = len(df)
    df.drop_duplicates(inplace=True)
    print(f"[INFO] Removed {initial_count - len(df)} duplicate rows.")

    # 3b. Drop rows with NaN values (if any)
    before_na = len(df)
    df.dropna(inplace=True)
    print(f"[INFO] Removed {before_na - len(df)} rows containing NaN values.")

    # 3c. Remove outliers using IQR for all numeric columns
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    for col in numeric_cols:
        df = remove_outliers_iqr(df, col, threshold=1.5)
    print("[INFO] Data shape after outlier removal:", df.shape)

    # 3d. Encode the target column if it is categorical
    target_col = config.data.target_column
    if df[target_col].dtype == object:
        le = LabelEncoder()
        df[target_col] = le.fit_transform(df[target_col])
        print(f"[INFO] Converted '{target_col}' to numeric labels.")

    # 4. Save prepared data
    prepared_data_path = os.path.join(config.data.path, "iris_prepared.csv")
    df.to_csv(prepared_data_path, index=False)
    print(f"[INFO] Data prepared and saved to {prepared_data_path}")
    print("[INFO] Final prepared data shape:", df.shape)

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

1. **Data Preparation:**
   - Run: `python data_preparation.py`
   - Output: `data/iris_prepared.csv`
2. **Model Training:**
   - Run: `python train.py`
   - Output: `data/trained_model.pkl`
3. **Model Evaluation:**
   - Run: `python eval.py`
   - Metrics: Accuracy, F1-score
4. **DVC Pipeline Execution:**
   - Command: `dvc repro`
   - Automates all stages.

6. Training: `train.py`

This script loads **prepared data**, splits it into training and test sets, and trains either a logistic regression or a decision tree classifier.

```python
# train.py
import os
import joblib
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from omegaconf import OmegaConf

def main():
    # 1. Load config
    config = OmegaConf.load("config.yaml")

    # 2. Load prepared data
    prepared_data_path = os.path.join(config.data.path, "iris_prepared.csv")
    df = pd.read_csv(prepared_data_path)

    # 3. Separate features and target
    target_column = config.data.target_column
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # 4. Split data into train & test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=config.training.test_size,
        random_state=config.training.random_state
    )

    # 5. Choose model type based on config
    model_type = config.model.type
    if model_type == "logistic_regression":
        model = LogisticRegression(
            max_iter=config.model.hyperparameters.get("max_iter", 100)
        )
    elif model_type == "decision_tree":
        model = DecisionTreeClassifier(
            max_depth=config.model.hyperparameters.get("max_depth", None),
            random_state=config.training.random_state
        )
    else:
        raise ValueError(f"Unknown model type: {model_type}")

    # 6. Train model
    model.fit(X_train, y_train)

    # 7. Save the trained model
    model_path = os.path.join(config.data.path, "trained_model.pkl")
    joblib.dump(model, model_path)
    print(f"[INFO] Model trained and saved to {model_path}")

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

1. **Data Preparation:**
   - Run: `python data_preparation.py`
   - Output: `data/iris_prepared.csv`
2. **Model Training:**
   - Run: `python train.py`
   - Output: `data/trained_model.pkl`
3. **Model Evaluation:**
   - Run: `python eval.py`
   - Metrics: Accuracy, F1-score
4. **DVC Pipeline Execution:**
   - Command: `dvc repro`
   - Automates all stages.

7. Evaluation: `eval.py`

This script loads the **trained model**, re-splits the data in the same manner, and evaluates performance metrics.

```python
# eval.py
import os
import joblib
import pandas as pd
from omegaconf import OmegaConf
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split

def main():
    # 1. Load config
    config = OmegaConf.load("config.yaml")

    # 2. Load prepared data
    prepared_data_path = os.path.join(config.data.path, "iris_prepared.csv")
    df = pd.read_csv(prepared_data_path)

    # 3. Separate features and target
    target_column = config.data.target_column
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # 4. Split data (same split as in train.py)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y,
        test_size=config.training.test_size,
        random_state=config.training.random_state
    )

    # 5. Load the trained model
    model_path = os.path.join(config.data.path, "trained_model.pkl")
    model = joblib.load(model_path)

    # 6. Predict on test set
    y_pred = model.predict(X_test)

    # 7. Evaluate metrics
    metrics_list = config.evaluation.metrics
    for metric in metrics_list:
        if metric.lower() == "accuracy":
            acc = accuracy_score(y_test, y_pred)
            print(f"[INFO] Accuracy: {acc:.4f}")
        elif metric.lower() == "f1":
            # Weighted average for multi-class
            f1 = f1_score(y_test, y_pred, average="weighted")
            print(f"[INFO] F1 (weighted): {f1:.4f}")
        else:
            print(f"[WARN] Metric '{metric}' is not implemented.")

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

Ubuntu Setup: Python & Virtual Environment

Since the Ubuntu system has no Python installed, let's install Python and set up a **virtual environment**:

1. **Update apt** and install Python 3, pip, and venv:

```
sudo apt-get update
sudo apt-get install -y python3 python3-pip python3-venv
```

2. **Create a virtual environment** (e.g., named `venv`):

```
python3 -m venv venv
```

3. **Activate** the virtual environment:

```
source venv/bin/activate
```

4. **Install required Python packages** (e.g., `pandas`, `scikit-learn`, `OmegaConf`):

Usage

Once you've **cloned** the repo, installed **Python** and **dependencies**, and **activated** your virtual environment:

1. **Check your** `config.yaml`
   - Verify the paths (`data.path`, `data.file_name`) and the `target_column`.
   - Switch model type (e.g., `"logistic_regression"` or `"decision_tree"`) if desired.
2. **Run data preparation**:

```
python data_preparation.py
```

   - This will clean and transform `iris.csv` → `iris_prepared.csv`.
3. **Train the model**:

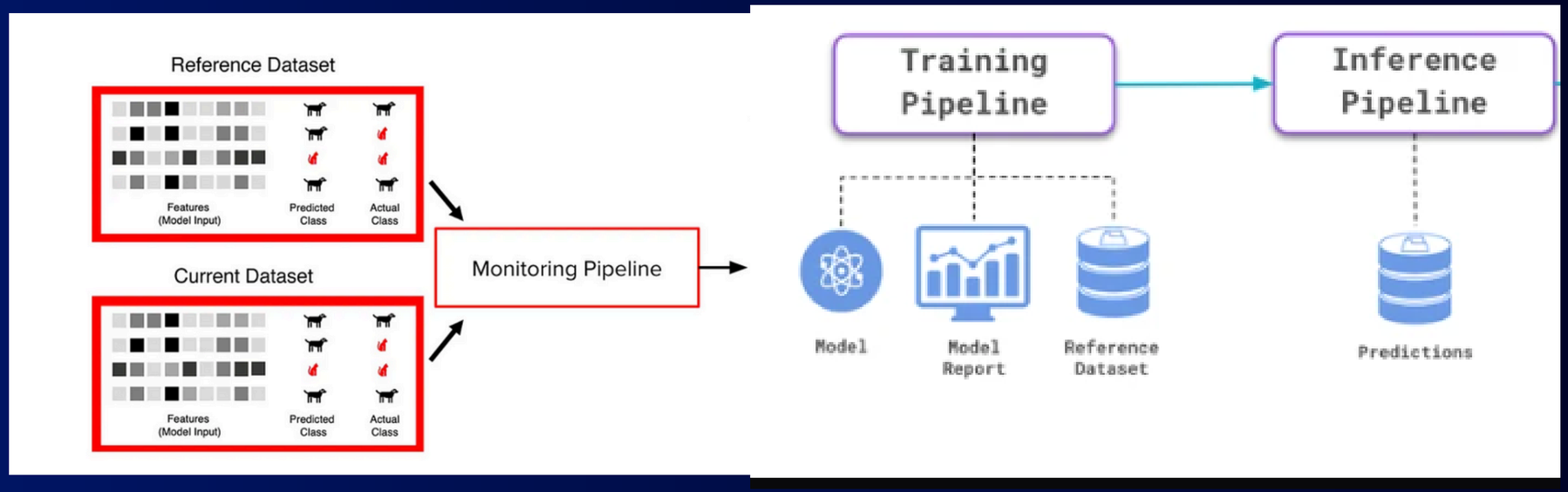
```
python train.py
```

   - This trains your chosen model and saves it to `trained_model.pkl`.
4. **Evaluate the model**:

```
python eval.py
```

   - This will print out the requested metrics (e.g., accuracy, F1).

LAB 3. Auto MLCycle with DVC Pipeline




```
************************************************************
*                    Data Preparation                      *
************************************************************
* Run: python data_preparation.py                          *
* Output: data/iris_prepared.csv                           *
************************************************************
                          |
                          v
************************************************************
*                    Model Training                        *
************************************************************
* Run: python train.py                                     *
* Output: data/trained_model.pkl                           *
************************************************************
                          |
                          v
************************************************************
*                    Model Evaluation                      *
************************************************************
* Run: python eval.py                                      *
* Metrics: Accuracy, F1-score                             *
************************************************************
```

Create the DVC Pipeline

1. **Create or update the `dvc.yaml` file** (in the current directory):

```yaml
stages:
  prepare:
    cmd: python data_preparation.py
    deps:
      - data/iris.csv
      - data_preparation.py
    outs:
      - data/iris_prepared.csv

  train:
    cmd: python train.py
    deps:
      - data/iris_prepared.csv
      - train.py
    outs:
      - data/trained_model.pkl

  evaluate:
    cmd: python eval.py
    deps:
      - data/iris_prepared.csv
      - eval.py
      - data/trained_model.pkl
```

6. Run the Pipeline

1. **Reproduce (run) the pipeline:**

```
dvc repro
```

Example output:

```
'data/iris_prepared.csv' didn't exist. Stage 'prepare' is being run:
> python data_preparation.py
[INFO] Loaded raw data shape: (150, 5)
[INFO] Removed 0 duplicate rows.
[INFO] Removed 0 rows containing NaN values.
[INFO] Data shape after outlier removal: (150, 5)
[INFO] Converted 'species' to numeric labels.
[INFO] Data prepared and saved to data/iris_prepared.csv
...
Stage 'train' is being run:
> python train.py
[INFO] Model trained and saved to data/trained_model.pkl


...
Stage 'evaluate' is being run:
> python eval.py
[INFO] Accuracy: 0.8961
[INFO] F1 (weighted): 0.8961



Pipeline completed successfully.
```

Create the DVC Pipeline

1. **Create or update the `dvc.yaml` file** (in the current directory):

```yaml
stages:
  prepare:
    cmd: python data_preparation.py
    deps:
      - data/iris.csv
      - data_preparation.py
    outs:
      - data/iris_prepared.csv

  train:
    cmd: python train.py
    deps:
      - data/iris_prepared.csv
      - train.py
    outs:
      - data/trained_model.pkl

  evaluate:
    cmd: python eval.py
    deps:
      - data/iris_prepared.csv
      - eval.py
      - data/trained_model.pkl
```

2. **Visualize the pipeline (optional):**

```
dvc dag
```

Example output: