# Machine Learning Pipeline and API Documentation

## Version: 1.0 ### Author: Sumaiya Mifria Akhter

---

## \*\*Introduction\*\*

This document provides a detailed guide for implementing a machine learning pipeline and interacting with it via RESTful API endpoints using FastAPI. The pipeline consists of the following stages:

1. \*\*Data Ingestion\*\*: Handles uploading and loading raw data.2. \*\*Data Transformation\*\*: Prepares the data for modeling.3. \*\*Model Training\*\*: Builds and evaluates a machine learning model.

---

## \*\*Project Structure\*\*

. ├── app.py # FastAPI application ├── main.py # Implementation of pipeline classes ├── uploaded\_files/ # Directory for storing uploaded files ├── transformed\_data.csv # Output of data transformation └── requirements.txt # Python dependencies

python

Copy code

---

## \*\*Pipeline Components\*\*

### \*\*1. DataIngestion\*\*

The `DataIngestion` class is responsible for handling raw data files. It includes methods for setting the data path and loading data.

#### \*\*Methods\*\*

- `\_\_init\_\_(self, data\_path: str = None)`: Initializes the class with an optional raw data path.

- `set\_data\_path(self, data\_path: str)`: Updates the data path dynamically.

- `load\_data(self, path: str)`: Reads the raw data from the specified file path into a DataFrame.

---

### \*\*2. DataTransformation\*\*

The `DataTransformation` class processes and transforms raw data to make it suitable for model training.

#### \*\*Methods\*\*

- `transform(self, data: pd.DataFrame)`: Applies transformations to the input DataFrame and returns the transformed data.

---

### \*\*3. ModelTrainer\*\*

The `ModelTrainer` class is responsible for building and evaluating the machine learning model.

#### \*\*Methods\*\*

- `train(self)`: Trains the machine learning model using the transformed data.

- `metrics`: A property that stores model performance metrics such as accuracy and F1 score.

---

## \*\*API Endpoints\*\*

### \*\*1. Data Ingestion\*\*

\*\*Endpoint\*\*: `/ingest-data/`

\*\*Method\*\*: `POST`

\*\*Description\*\*: Uploads a CSV file to the server and loads it into the pipeline.

#### \*\*Workflow\*\*1. Upload the file via the endpoint.2. Save the file to the `uploaded\_files/` directory.3. Load the data using the `DataIngestion` class.

#### \*\*Example Usage\*\*

- \*\*Request (Curl)\*\*:

```bash

curl -X POST "http://127.0.0.1:8000/ingest-data/" \

-F "file=@your\_dataset.csv"

```

- \*\*Response\*\*:

```json

{

"message": "Data ingestion completed",

"raw\_data\_sample": {

"column1": ["value1", "value2"],

"column2": ["value3", "value4"]

}

}

```

---

### \*\*2. Data Transformation\*\*

\*\*Endpoint\*\*: `/transform-data/`

\*\*Method\*\*: `POST`

\*\*Description\*\*: Transforms the raw data and saves the processed data to a CSV file.

#### \*\*Workflow\*\*1. Retrieve the file path from the `DataIngestion` instance.2. Load the raw data from the file.3. Apply transformations using the `DataTransformation` class.4. Save the processed data as `transformed\_data.csv`.

#### \*\*Example Usage\*\*

- \*\*Request (Curl)\*\*:

```bash

curl -X POST "http://127.0.0.1:8000/transform-data/"

```

- \*\*Response\*\*:

```json

{

"message": "Data transformation completed",

"transformed\_data\_sample": {

"transformed\_col1": ["value1\_trans", "value2\_trans"],

"transformed\_col2": ["value3\_trans", "value4\_trans"]

}

}

```

---

### \*\*3. Model Training\*\*

\*\*Endpoint\*\*: `/train-model/`

\*\*Method\*\*: `POST`

\*\*Description\*\*: Trains a machine learning model on the transformed data and returns the evaluation metrics.

#### \*\*Workflow\*\*1. Load the transformed data.2. Train the model using the `ModelTrainer` class.3. Return model performance metrics.

#### \*\*Example Usage\*\*

- \*\*Request (Curl)\*\*:

```bash

curl -X POST "http://127.0.0.1:8000/train-model/"

```

- \*\*Response\*\*:

```json

{

"message": "Model training completed",

"model\_metrics": {

"accuracy": 0.95,

"f1\_score": 0.93

}

}

```

---

## \*\*Implementation Details\*\*

### \*\*FastAPI Application (`app.py`)\*\*

The following code outlines the FastAPI endpoints and their interaction with the pipeline classes:

```pythonfrom fastapi import FastAPI, UploadFilefrom main import DataIngestion, DataTransformation, ModelTrainerimport osimport pandas as pd

app = FastAPI()

# Initialize pipeline components

data\_ingestion = DataIngestion()

data\_transformation = DataTransformation()

model\_trainer = ModelTrainer()

@app.post("/ingest-data/")async def ingest\_data(file: UploadFile):

file\_location = f"uploaded\_files/{file.filename}"

os.makedirs("uploaded\_files", exist\_ok=True)

with open(file\_location, "wb") as f:

f.write(await file.read())

data\_ingestion.set\_data\_path(file\_location)

raw\_data = data\_ingestion.load\_data(file\_location)

return {"message": "Data ingestion completed", "raw\_data\_sample": raw\_data.head(5).to\_dict()}

@app.post("/transform-data/")def transform\_data():

data\_path = data\_ingestion.raw\_data\_path

raw\_data = pd.read\_csv(data\_path)

transformed\_data = data\_transformation.transform(raw\_data)

transformed\_data.to\_csv("transformed\_data.csv", index=False)

return {"message": "Data transformation completed", "transformed\_data\_sample": transformed\_data.head(5).to\_dict()}

@app.post("/train-model/")def train\_model():

model\_trainer.train()

return {"message": "Model training completed", "model\_metrics": model\_trainer.metrics}