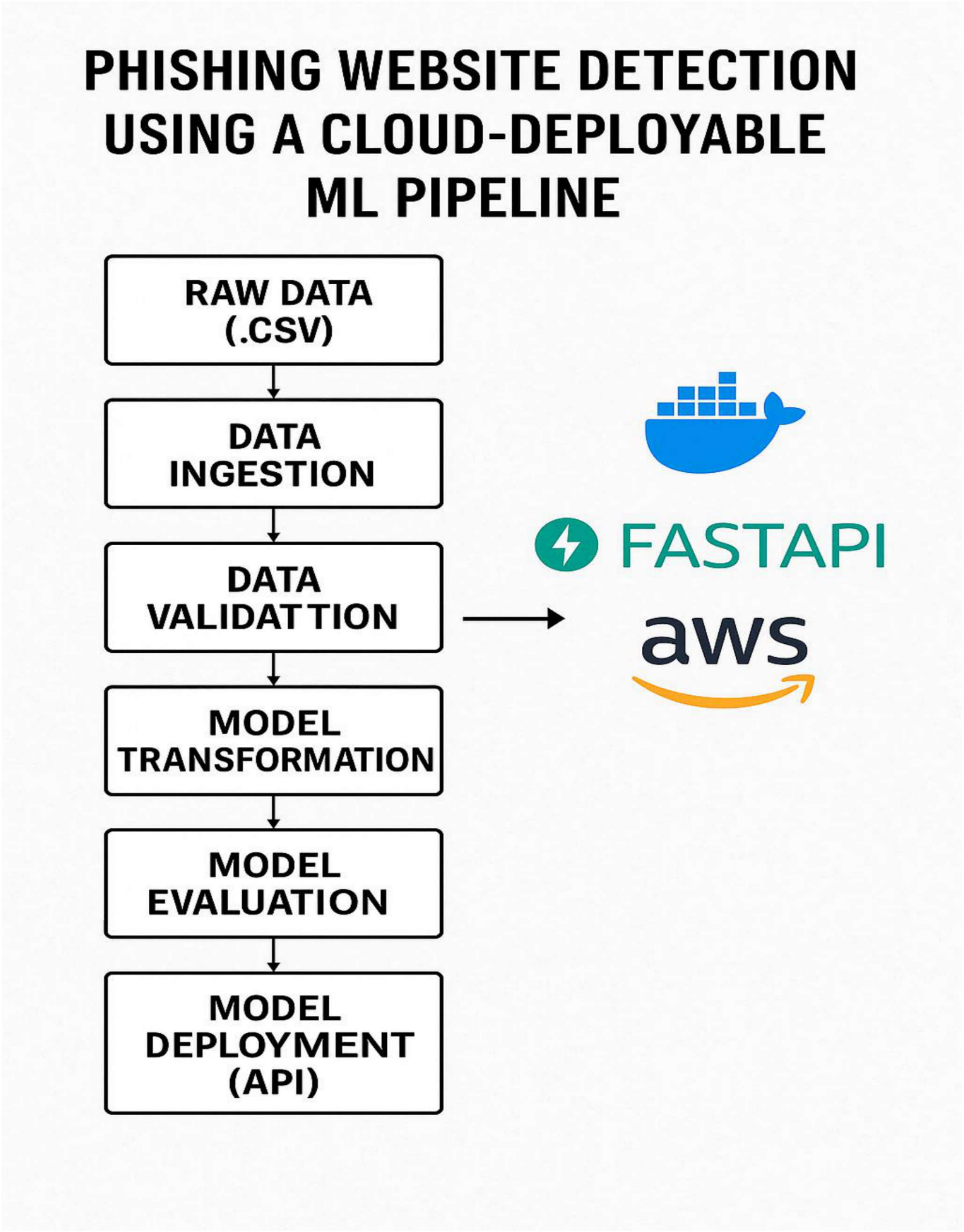
Phishing Website Detection – End-to-End ML Pipeline Project



By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Project Title

Network Security ML Pipeline: Phishing Website Detection and Cloud Deployment

---

Project Overview

This end-to-end machine learning project is designed to detect phishing websites using structured features derived from URLs and network behaviour.

The solution is implemented as a binary classification pipeline, where the objective is to accurately classify a given website as either phishing (malicious) or legitimate (safe).

Key Highlights of the Project

- **Problem Definition**: A supervised learning problem, specifically binary classification, aimed at identifying phishing websites based on feature-engineered data.
- **Data Source**: Structured CSV dataset containing a wide range of features such as:
    - URL characteristics (e.g., length, presence of special characters)
    - Security indicators (e.g., HTTPS, SSL status)
    - Network behavior flags
- **Architecture**: Built using a **modular pipeline structure**, split across major components:
    - Data Ingestion
    - Data Validation
    - Data Transformation
    - Model Training
    - Model Evaluation
    - Inference and Deployment
- **Robust Engineering**:
    - Custom **exception handling** and **logging mechanisms** for debuggability
    - Use of **dataclasses** to manage artifacts and configuration consistently
- **Cloud Integration**:
    - Data storage and retrieval via **MongoDB Atlas**
    - Model artifacts and metrics synced to **AWS S3**
- **Model Optimization**:
    - Hyperparameter tuning to improve **precision, recall, F1-score**
    - Tracking experiments with **MLflow**
- **Deployment Stack**:
    - REST API using **FastAPI**
    - Containerized using **Docker**
    - Deployed to **AWS EC2** for real-time predictions
- **CI/CD Automation**:
    - **GitHub Actions** automate Docker builds and push to **AWS ECR**
    - Ensures reproducible, one-click deployment pipeline

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Initial Project Structure Created in VS Code

```
📂 NETWORK_SECURITY_PROJECT/                Root directory of the project
├── .env                                    Stores sensitive secrets like database passwords or API keys
├── .gitignore                              Tells Git which files/folders to ignore (like .env, venv, __pycache__)
├── app.py                                  FastAPI app for serving predictions through API
├── Dockerfile                              Contains Docker instructions to package the entire app
├── main.py                                 Entry point for running the entire ML training pipeline
├── push_data.py                            Uploads raw CSV data into MongoDB
├── README.md                               Project summary and setup instructions
├── requirements.txt                        List of required Python libraries
├── setup.py                                Makes the code inside `Network_security` installable as a package
├── test_Mongo_DB.py                        Script to test MongoDB connection
├── 📂 venv                                 Python virtual environment (auto-created, not tracked in Git)
├── 📂 data_schema/                         Contains schema definition (`schema.yaml`) to validate input CSV columns
    - schema.yaml
├── 📂 Network_security/                    Main Python package with all logic
    - __init__.py
│   ├── 📂 cloud/                           Code for syncing data with AWS S3
    - __init__.py
    - S3_syncer.py
│   ├── 📂 components/                      All core ML pipeline steps: data ingestion, validation, etc.
    - __init__.py
    - data_ingestion.py
    - data_transformation.py
    - data_validation.py
    - model_trainer.py
│   ├── 📂 constants/                       Stores fixed configuration values like file paths and column names
    -   __init__.py
    -   📂 training_pipeline/
        ○   __init__.py (The file contains all the constant values)
│   ├── 📂 entity/                          Data classes for passing configuration and output between pipeline steps
    - __init__.py
    - artifact_entity.py
    - config_entity.py
│   ├── 📂 exception/                       Custom exception handling for better error tracking
    - __init__.py
    - exception.py
│   ├── 📂 logging/                         Centralized logging setup for debugging and monitoring
    - __init__.py
    - logger.py
│   ├── 📂 pipeline/                        Scripts to run full training or batch prediction pipelines
    - __init__.py
    - batch_prediction.py
    - training_pipeline.py
│   ├── 📂 utils/
    -   __init__.py
    -   📂 main_utils/                      Common utility functions used across the pipeline
        ○   __init__.py
        ○   utils.py
    -   📂 ml_utils/                        ML-specific helpers like metrics and model creation
        ○   __init__.py
        ○   📂 metric/
            ▪   __init__.py
            ▪   classification_report.py
        ○   📂 model/
            ▪   __init__.py
            ▪   estimator.py
├── 📂 Network_security_data/
    -   phisingData.csv                     Stores the raw dataset (`phisingData.csv`) used by the pipeline
```

Each folder contains an __init__.py file so it can be used as a Python package. This modular setup ensures clear separation of concerns and allows for easy scaling and maintenance.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Project Packaging with setup.py

setup.py > ...

```python
from setuptools import find_packages,setup
from typing import List

def get_requirements ()->List[str]:
    req_list:List[str]=[]
    try:
        with open('requirements.txt','r') as file:
            lines=file.readlines()
            for line in lines:
                req=line.strip()
                if req!='' and req!='-e .':
                    req_list.append(req)
    except FileNotFoundError:
        print("requirements.txt not found")
    return req_list

setup(
    name="Network_Security",
    version="0.0.1",
    author="kunal Shukla",
    author_email='kpshukla3@gmail.com',
    packages=find_packages(),
    install_requires=get_requirements()
)
```

It configures and installs project as a Python package.

List[str] is a type hint, not part of code execution. Improves readability and static checking.

-e . is useful during development, but it should not be passed to install requires, because it tells pip to install the current project — which is already being installed.

The setup() function defines metadata (name, version, author), auto-discovers packages, and sets installation dependencies using the helper function.

Logging and Exception Handling

Network_security > logging > logger.py > ...

```python
import logging
import os
from datetime import datetime

LOG_FILE=f"{datetime.now().strftime('%m_%d_%Y_%H_%M_%S')}.log"

logs_path=os.path.join(os.getcwd(),"logs",LOG_FILE)
os.makedirs(logs_path,exist_ok=True)

LOG_FILE_PATH=os.path.join(logs_path,LOG_FILE)

logging.basicConfig(
    filename=LOG_FILE_PATH,
    format="[ %(asctime)s ] %(lineno)d %(name)s - %(levelname)s - %(message)s",
    level=logging.INFO,
)
```

04_16_2025_12_35_10.log

Network_Security_Project/logs/04_16_2025_12_35_10.log

Defines where logs go, what format to use, and what level of messages to record

Network_security > exception > exception.py > ...

```python
import sys
from Network_security.logging import logger

class NetworkSecurityException(Exception):
    def __init__(self,error_message,error_details:sys):
        self.error_message = error_message
        _,_,exc_tb = error_details.exc_info()

        self.lineno=exc_tb.tb_lineno
        self.file_name=exc_tb.tb_frame.f_code.co_filename

    def __str__(self):
        return "Error occured in python script name [{0}] line number [{1}] error message [{2}]".format(
        self.file_name, self.lineno, str(self.error_message))
```

To access system-level info

Inheriting from Python's built-in Exception class makes it a valid custom error that can be raised and handled like built-in exceptions.

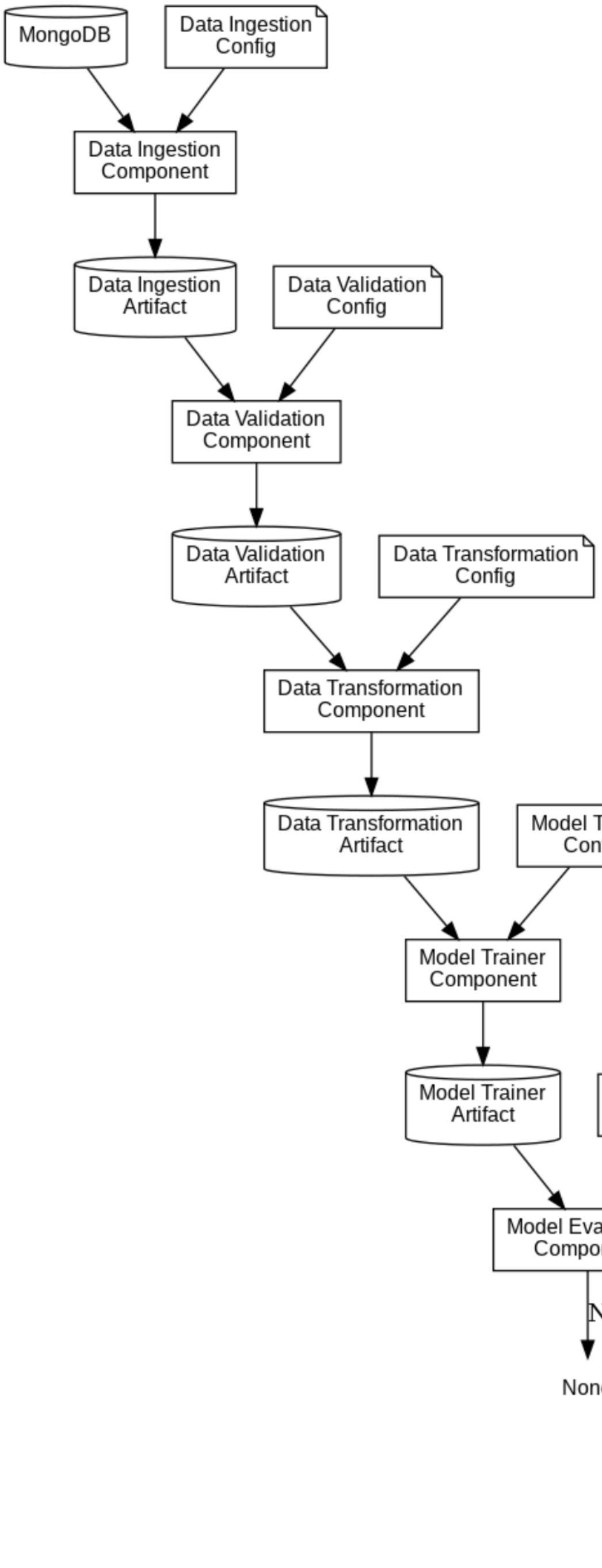
error_details.exc_info() returns a tuple of 3 items:
(exception_type, exception_value, traceback_object)

_, _, exc_tb is used to retrieve only the third element

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

End-to-end machine learning pipeline flowchart



By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Project Dependencies with requirements.txt

requirements.txt

```
python-dotenv
pandas
numpy
pymongo
certifi
pymongo[srv]
scikit-learn
mlflow
pyaml
fastapi
uvicorn
python-multipart
dill

-e .
```

- `python-dotenv` → Loads environment variables from .env file, used to manage sensitive credentials like database URIs and AWS
- `pymongo` → Python client for MongoDB to connect, read, and write to the MongoDB Atlas database
- `certifi` → Provides Mozilla's trusted CA certificates, used to securely connect to remote MongoDB
- `scikit-learn` → A YAML parser for reading schema.yaml and configuration files
- `pyaml` → Web framework used to build and expose the model inference API
- `fastapi` → ASGI server used to run the FastAPI application in production
- `uvicorn` → Required by FastAPI to handle file uploads in HTTP requests (used in batch prediction)
- `python-multipart` → Used to save and load complex Python objects like lambda functions or custom classes — works better than pickle in many ML use cases.
- `dill`

Why MongoDB ?

Instead of reading data from local CSV files, we use MongoDB to store and retrieve data in a centralized, scalable, and flexible way. This makes the pipeline cloud-ready and suitable for real-time access and updates, unlike static CSVs.

MongoDB Setup and URI Configuration

This project uses MongoDB Atlas, a cloud-based NoSQL database, for storing and retrieving structured data during ingestion and prediction. The MongoDB connection string (URI) is created in MongoDB Atlas and securely loaded from .env file using python-dotenv.

test_Mongo_DB.py – Testing MongoDB Connectivity Using URI

.env

```
Mongo_DB_URL=
```

generated via MongoDB Atlas

test_Mongo_DB.py > ...

```python
from pymongo.mongo_client import MongoClient
from dotenv import load_dotenv
import os

# Load environment variables
load_dotenv()

# Get MongoDB URI from environment
uri = os.getenv("MONGO_DB_URL")
# Create a new client and connect to the server
client = MongoClient(uri)
# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

push_data.py – Converting CSV to JSON and Inserting into MongoDB

push_data.py > NetworkDataExtract > csv_to_json_convertor

```python
import os
import sys
import json

from dotenv import load_dotenv
load_dotenv()
MONGO_DB_URL=os.getenv("MONGO_DB_URL")

import certifi
ca=certifi.where()

import pandas as pd
import numpy as np
import pymongo

from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging import logger

class NetworkDataExtract():
    def __init__(self):
        try:
            pass
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def csv_to_json_convertor(self,file_path):
        try:
            data=pd.read_csv(file_path)
            data.reset_index(drop=True,inplace=True)
            records=list(json.loads(data.T.to_json()).values())
            return records
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

certifi provides trusted SSL certificates
certifi.where() returns the path to those certificates, stored in ca, for secure MongoDB connection.

These imports will be included in all scripts going forward to ensure consistent exception tracking and logging across the entire pipeline

	0	1	2	3	4	5	6	7	8	9	...	11045	11046	11047
having_IP_Address	-1	1	1	1	1	-1	1	1	1	1	...	1	-1	-1
URL_Length	1	1	0	0	0	0	0	0	0	1	...	-1	-1	-1
Shortining_Service	1	1	1	1	-1	-1	-1	1	-1	-1	...	1	1	1
having_At_Symbol	1	1	1	1	1	1	1	1	1	1	...	1	1	1
double_slash_redirecting	-1	1	1	1	1	-1	1	1	1	1	...	1	1	1
Prefix_Suffix	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	...	-1	-1	-1
having_Sub_Domain	-1	0	-1	-1	1	1	-1	-1	1	-1	...	1	1	1

data.reset_index(drop=True, inplace=True)

data.T

data.T.to_json()

```
'{"0":{"having_IP_Address":-1,"URL_Length":1,"Shortining_Service":1,"having_At_Symbol":1,"double_slash_redirecting":-1,"Prefi
x_Suffix":-1,"having_Sub_Domain":-1,"SSLfinal_State":-1,"Domain_registeration_length":-1,"Favicon":1,"port":1,"HTTPS_token":-
1,"Request_URL":1,"URL_of_Anchor":-1,"Links_in_tags":1,"SFH":-1,"Submitting_to_email":-1,"Abnormal_URL":-1,"Redirect":0,"on_m
ouseover":1,"RightClick":1,"popUpWidnow":1,"Iframe":1,"age_of_domain":-1,"DNSRecord":-1,"web_traffic":-1,"Page_Rank":-1,"Goog
le_Index":1,"Links_pointing_to_page":1,"Statistical_report":-1,"Result":-1},"1":{"having_IP_Address":1,"URL_Length":1,"Shorti
ning_Service":1,"having_At_Symbol":1,"double_slash_redirecting":1,"Prefix_Suffix":-1,"having_Sub_Domain":0,"SSLfinal_State":
1,"Domain_registeration_length":-1,"Favicon":1,"port":1,"HTTPS_token":-1,"Request_URL":1,"URL_of_Anchor":0,"Links_in_tags":-
1,"SFH":-1,"Submitting_to_email":1,"Abnormal_URL":1,"Redirect":0,"on_mouseover":1,"RightClick":1,"popUpWidnow":1,"Iframe":
```

json.loads(data.T.to_json()

```
{'0': {'having_IP_Address': -1,
  'URL_Length': 1,
  'Shortining_Service': 1,
  'having_At_Symbol': 1,
  'double_slash_redirecting': -1,
  'Prefix_Suffix': -1,
  'having_Sub_Domain': -1,
  'SSLfinal_State': -1,
  'Domain_registeration_length': -1,
  'Favicon': 1,
  'port': 1,
  'HTTPS_token': -1,
  'Request_URL': 1,
  'URL_of_Anchor': -1,
  'Links_in_tags': 1,
  'SFH': -1,
  'Submitting_to_email': -1,
  'Abnormal_URL': -1,
  'Redirect': 0,
```

list(json.loads(data.T.to_json()).values()

```
[{'having_IP_Address': -1,
  'URL_Length': 1,
  'Shortining_Service': 1,
  'having_At_Symbol': 1,
  'double_slash_redirecting': -1,
  'Prefix_Suffix': -1,
  'having_Sub_Domain': -1,
  'SSLfinal_State': -1,
  'Domain_registeration_length': -1,
  'Favicon': 1,
  'port': 1,
  'HTTPS_token': -1,
  'Request_URL': 1,
  'URL_of_Anchor': -1,
  'Links_in_tags': 1,
  'SFH': -1,
  'Submitting_to_email': -1,
  'Abnormal_URL': -1,
  'Redirect': 0,
```

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

```python
def insert_data_mongodb(self,records,database,collection):
    try:
        self.database=database
        self.collection=collection
        self.records=records

        self.mongo_client=pymongo.MongoClient(MONGO_DB_URL)
        self.database=self.mongo_client[self.database]

        self.collection=self.database[self.collection]
        self.collection.insert_many(self.records)
        return(len(self.records))

    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Client acts as the gateway to interact with cloud database

Accesses the MongoDB database, which is like a schema that holds multiple collections.

Accesses the collection inside that database

Inserts all records into the MongoDB collection using batch insertion

```python

if __name__=='__main__':
    FILE_PATH="Network_security_data\phisingData.csv"
    DATABASE="KUNAL_AI"
    Collection="Networkdata"
    networkobj=NetworkDataExtract()
    records=networkobj.csv_to_json_convertor(file_path=FILE_PATH)
    print(records)
    no_of_records=networkobj.insert_data_mongodb(records,DATABASE,Collection)
    print(no_of_records)
```

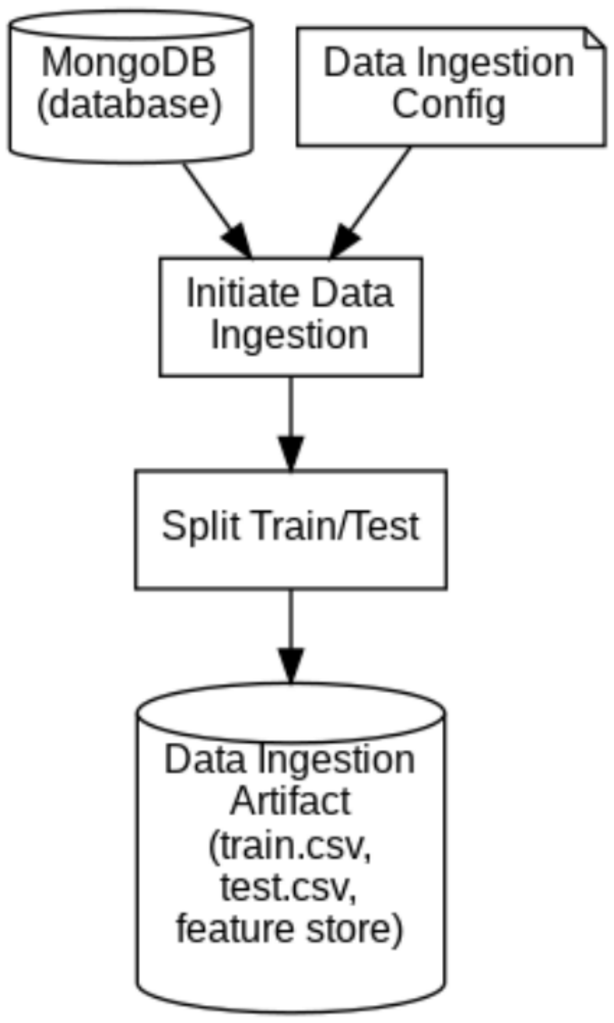
Sets the path to the input CSV file

Defines the target MongoDB database and collection

By Kunal P. Shukla

Data Ingestion Flow:

From MongoDB to Train/Test Split



training_pipeline/init.py – Constants for Data Ingestion Configuration

Constants are fixed values like directory names, filenames, collection names, or split ratios that do not change during execution. In this project, they are placed inside the training_pipeline module because they directly relate to the training process—keeping configuration centralized, organized, and easy to manage or modify later.

Network_security > constants > training_pipeline > __init__.py > ...

```python
import os
import sys
import numpy as np
import pandas as pd

TARGET_COLUMN="Result"
PIPELINE_NAME:str="NetworkSecurity"
ARTIFACT_DIR:str="Artifacts"
FILE_NAME:str="phisingData.csv"

TRAIN_FILE_NAME:str="train.csv"
TEST_FILE_NAME:str="test.csv"

DATA_INGESTION_COLLECTION_NAME:str="Networkdata"
DATA_INGESTION_DATABASE_NAME:str="KUNAL_AI"
DATA_INGESTION_DIR_NAME:str="data_ingestion"
DATA_INGESTION_FEATURE_STORE_DIR:str="feature_store"
DATA_INGESTION_INGESTED_DIR:str="ingested"
DATA_INGESTION_TRAIN_TEST_SPLIT_RATIO:float=0.2
```

data_ingestion_config.py- Setting up paths & parameters for Data Ingestion

Network_security > entity > config_entity.py > DataIngestionConfig > __init__

```python
from datetime import datetime
import os

from Network_security.constants import training_pipeline

print(training_pipeline.PIPELINE_NAME)
print(training_pipeline.ARTIFACT_DIR)

class TrainingPipelineConfig:
    def __init__(self,timestamp=datetime.now()):
        timestamp=timestamp.strftime("%m_%d_%Y_%H_%M_%S")
        self.pipeline_name=training_pipeline.PIPELINE_NAME
        self.artifact_name=training_pipeline.ARTIFACT_DIR
        self.artifact_dir=os.path.join(self.artifact_name,timestamp)
        self.model_dir=os.path.join("final_model")
        self.timestamp:str=timestamp
```

Imports predefined constants used for configuring the training pipeline

self.pipeline_name= NetworkSecurity
self.artifact_name=Artifacts

self.artifact_dir → Artifacts/04_17_2025_13_45_20

```python
class DataIngestionConfig:
    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.data_ingestion_dir:str=os.path.join(
            training_pipeline_config.artifact_dir,training_pipeline.DATA_INGESTION_DIR_NAME
        )
        self.feature_store_file_path:str=os.path.join(
            self.data_ingestion_dir,training_pipeline.DATA_INGESTION_FEATURE_STORE_DIR,training_pipeline.FILE_NAME
        )
        self.training_file_path:str=os.path.join(
            self.data_ingestion_dir,training_pipeline.DATA_INGESTION_INGESTED_DIR,training_pipeline.TRAIN_FILE_NAME
        )
        self.testing_file_path:str=os.path.join(
            self.data_ingestion_dir,training_pipeline.DATA_INGESTION_INGESTED_DIR,training_pipeline.TEST_FILE_NAME
        )
        self.train_test_split_ratio:float=training_pipeline.DATA_INGESTION_TRAIN_TEST_SPLIT_RATIO
        self.collection_name:str=training_pipeline.DATA_INGESTION_COLLECTION_NAME
        self.database_name:str=training_pipeline.DATA_INGESTION_DATABASE_NAME
```

Type hint indicates that an object of TrainingPipelineConfig must be passed during initialization

self.data_ingestion_dir → Artifacts/04_17_2025_13_45_20/data_ingestion

self.feature_store_file_path → Artifacts/04_17_2025_13_45_20/data_ingestion/feature_store/phisingData.csv

self.training_file_path → Artifacts/04_17_2025_13_45_20/data_ingestion/ingested/train.csv

self.testing_file_path → Artifacts/04_17_2025_13_45_20/data_ingestion/ingested/test.csv

Phishing Website Detection – End-to-End ML Pipeline Project

data_ingestion.py – Reading Data from MongoDB and Splitting into Train/Test

This module handles the complete data ingestion process. It pulls raw data from a MongoDB collection, stores it in a feature store (as CSV), and performs a train-test split. It returns a DataIngestionArtifact for downstream pipeline components.

Network_security > components > data_ingestion.py > DataIngestion > export_collection_as_dataframe

```python
import pymongo.mongo_client
from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging.logger import logging

from Network_security.entity.config_entity import DataIngestionConfig
from Network_security.entity.artifact_entity import DataIngestionArtifact

import os
import sys
import numpy as np
import pandas as pd
import pymongo
from typing import List
from sklearn.model_selection import train_test_split

from dotenv import load_dotenv
load_dotenv()
Mongo_DB_URL=os.getenv("Mongo_DB_URL")

class DataIngestion:
    def __init__(self,data_ingestion_config:DataIngestionConfig):
        try:
            self.data_ingestion_config=data_ingestion_config
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def export_collection_as_dataframe(self):
        try:
            database_name=self.data_ingestion_config.database_name
            collection_name=self.data_ingestion_config.collection_name
            self.mongo_client=pymongo.MongoClient(Mongo_DB_URL)
            collection=self.mongo_client[database_name][collection_name]
            df=pd.DataFrame(list(collection.find()))
            if "_id" in df.columns.to_list():
                df=df.drop(columns=["_id"],axis=1)
            df.replace({"na":np.nan},inplace=True)
            return df
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def export_data_into_feature_store(self,dataframe:pd.DataFrame):
        try:
            feature_store_file_path=self.data_ingestion_config.feature_store_file_path
            # Creating a folder
            dir_path=os.path.dirname(feature_store_file_path)
            os.makedirs(dir_path,exist_ok=True)
            dataframe.to_csv(feature_store_file_path,index=False,header=True)
            return dataframe
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Required for initializing data ingestion: config provides input settings, artifact stores output file paths

Retrieves database and collection names from the ingestion config object

Fetches all documents from the collection and converts them into DataFrame

Removes the MongoDB default _id column, which is not needed for model

Replaces any "na" string entries with proper NaN values for compatibility

Artifacts/04_17_2025_13_45_20/data_ingestion/feature_store/phisingData.csv

We convert the MongoDB data to CSV so it can be stored locally in the feature store and reused consistently across the pipeline, without repeatedly querying the database.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

```python
def split_data_as_train_test(self,dataframe:pd.DataFrame):
    try:
        train_set,test_set=train_test_split(
            dataframe,test_size=self.data_ingestion_config.train_test_split_ratio
            )
        logging.info("Performed train test split on DataFrame")
        logging.info(
            "Exited split_data_as_train_test method of Data_Ingestion class"
        )
        dir_path=os.path.dirname(self.data_ingestion_config.training_file_path)
        os.makedirs(dir_path,exist_ok=True)
        logging.info(f"Exporting train and test file path,")

        train_set.to_csv(
            self.data_ingestion_config.training_file_path,index=False,header=True
        )
        test_set.to_csv(
            self.data_ingestion_config.testing_file_path,index=False,header=True
        )
        logging.info(f"Exporting train and test file path,")

    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Splits data into training and testing sets using ratio from config

Artifacts/04_17_2025_13_45_20/data_ingestion/ingested/ folder will be created to store train.csv and test.csv

Artifacts/04_17_2025_13_45_20/data_ingestion/ingested/train.csv
Artifacts/04_17_2025_13_45_20/data_ingestion/ingested/test.csv

```python
def initiate_data_ingestion(self):
    try:
        dataframe=self.export_collection_as_dataframe()
        dataframe=self.export_data_into_feature_store(dataframe)
        self.split_data_as_train_test(dataframe)
        dataingestionartifact=DataIngestionArtifact(
            trained_file_path=self.data_ingestion_config.training_file_path,
            test_file_path=self.data_ingestion_config.testing_file_path)
        return dataingestionartifact


    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Fetching data from MongoDB into a DataFrame

Saving raw data into a local feature store CSV

Splitting data into train and test sets

Returning a DataIngestionArtifact, containing paths to train.csv and test.csv

artifact_entity.py – DataIngestionArtifact Definition

This dataclass represents the output (artifact) of the data ingestion step. It holds the file paths for both the training and testing datasets, which are used as inputs for the next stages of the ML pipeline.

Network_security > entity > artifact_entity.py > DataTransformationArtifact > transformed_train_file_path

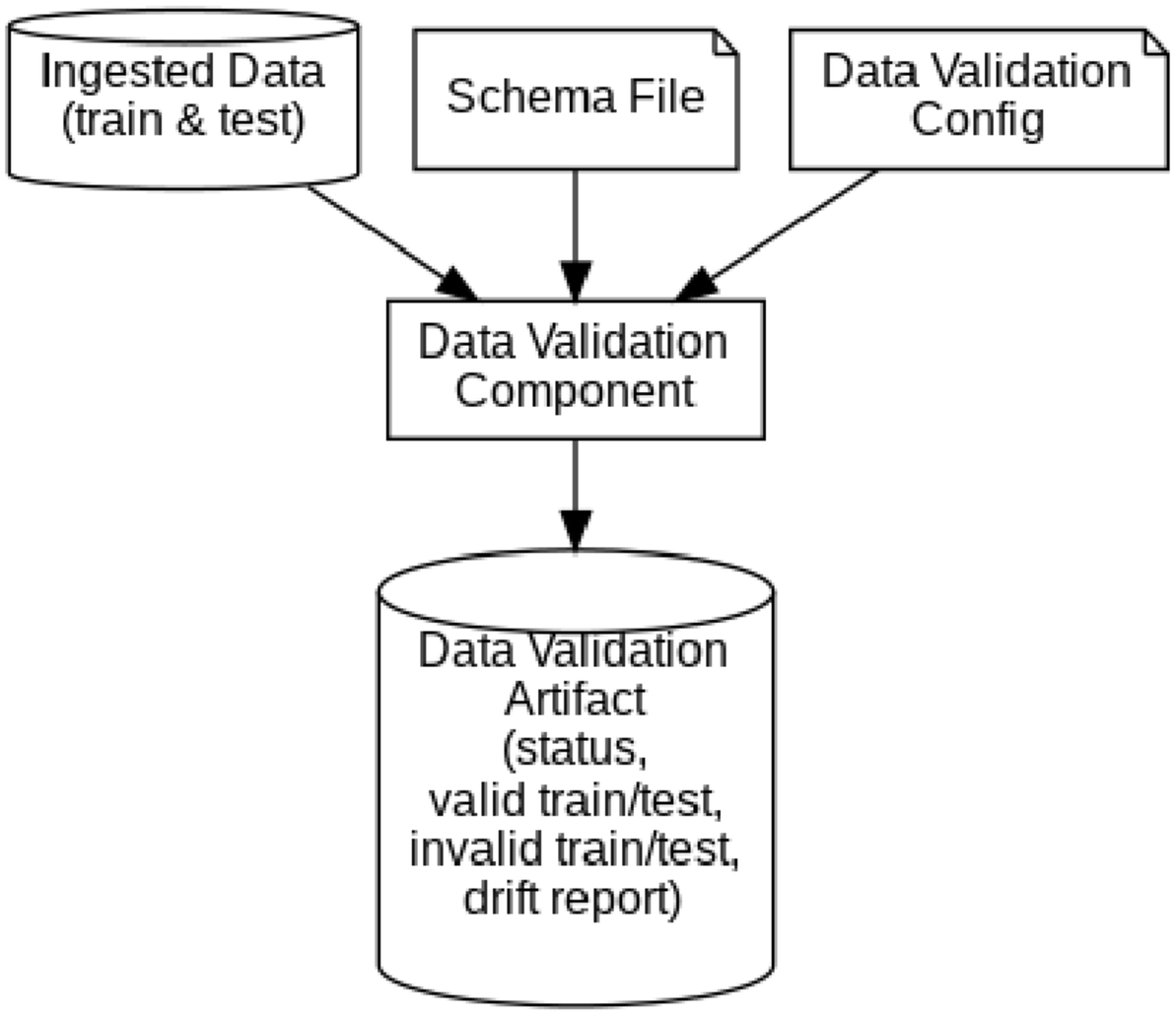
```python
from dataclasses import dataclass

@dataclass
class DataIngestionArtifact:
    trained_file_path:str
    test_file_path:str
```

Instead of returning artifacts directly from data ingestion, we define a structured dataclass here to store them and access them consistently across the pipeline.

By Kunal P. Shukla

Data Validation Flow:

From Schema Check to Drift Detection



training_pipeline/init.py – Constants for Data Validation Configuration

Network_security > constants > training_pipeline > __init__.py > ...

```python
DATA_VALIDATION_DIR_NAME: str="data validation"
DATA_VALIDATION_VALID_DIR: str="validated"
DATA_VALIDATION_INVALID_DIR: str="invalid"
DATA_VALIDATION_DRIFT_REPORT_DIR: str="drift_report"
DATA_VALIDATION_DRIFT_REPORT_DIR_NAME: str="report.yaml"
PREPROCESSING_OBJECT_FILE_NAME="preprocessing.pkl"
```

data_validation_config.py – defining rules and paths for Data Validation

Network_security > entity > config_entity.py > DataTransformationConfig > __init__

```python
class DataValidationConfig:
    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.data_validation_dir:str=os.path.join(training_pipeline_config.artifact_dir,
                                                  training_pipeline.DATA_VALIDATION_DIR_NAME)
        self.valid_data_dir:str=os.path.join(self.data_validation_dir,training_pipeline.DATA_VALIDATION_VALID_DIR)
        self.invalid_data_dir:str=os.path.join(self.data_validation_dir,training_pipeline.DATA_VALIDATION_INVALID_DIR)
        self.valid_train_file_path:str=os.path.join(self.valid_data_dir,training_pipeline.TRAIN_FILE_NAME)
        self.valid_test_file_path:str=os.path.join(self.valid_data_dir,training_pipeline.TEST_FILE_NAME)
        self.invalid_train_file_path:str=os.path.join(self.invalid_data_dir,training_pipeline.TRAIN_FILE_NAME)
        self.invalid_test_file_path:str=os.path.join(self.invalid_data_dir,training_pipeline.TEST_FILE_NAME)
        self.drift_report_file_path:str=os.path.join(self.data_validation_dir,
                                                     training_pipeline.DATA_VALIDATION_DRIFT_REPORT_DIR,
                                                     training_pipeline.DATA_VALIDATION_DRIFT_REPORT_DIR_NAME)
```

self.data_validation_dir → Artifacts/04_17_2025_13_45_20/data validation

self.valid_data_dir → Artifacts/04_17_2025_13_45_20/data validation/validated

self.invalid_data_dir → Artifacts/04_17_2025_13_45_20/data validation/invalid

self.valid_train_file_path → Artifacts/04_17_2025_13_45_20/data validation/validated/train.csv

self.valid_test_file_path → Artifacts/04_17_2025_13_45_20/data validation/validated/test.csv

self.invalid_train_file_path → Artifacts/04_17_2025_13_45_20/data validation/invalid/train.csv

self.invalid_test_file_path → Artifacts/04_17_2025_13_45_20/data validation/invalid/test.csv

self.drift_report_file_path → Artifacts/04_17_2025_13_45_20/data validation/drift_report/report.yaml

schema.yaml – Defining expected columns and numerical fields for validation

This file lists all required columns along with their expected data types. The numerical_columns section is added separately to help the validation component detect numerical drift. Both sections are used in data_validation.py to ensure the structure and types match the expected format before training begins.

data_schema > ! schema.yaml

```yaml
columns:
  - having_IP_Address: int64
  - URL_Length: int64
  - Shortining_Service: int64
  - having_At_Symbol: int64
  - double_slash_redirecting: int64
  - Prefix_Suffix: int64
  - having_Sub_Domain: int64
  - SSLfinal_State: int64
  - Domain_registeration_length: int64
  - Favicon: int64
  - port: int64
  - HTTPS_token: int64
  - Request_URL: int64
  - URL_of_Anchor: int64
  - Links_in_tags: int64
  - SFH: int64
  - Submitting_to_email: int64
  - Abnormal_URL: int64
  - Redirect: int64
  - on_mouseover: int64
  - RightClick: int64
```

data_schema > ! schema.yaml

```yaml
numerical_columns:
  - having_IP_Address
  - URL_Length
  - Shortining_Service
  - having_At_Symbol
  - double_slash_redirecting
  - Prefix_Suffix
  - having_Sub_Domain
  - SSLfinal_State
  - Domain_registeration_length
  - Favicon
  - port
  - HTTPS_token
  - Request_URL
  - URL_of_Anchor
  - Links_in_tags
  - SFH
  - Submitting_to_email
  - Abnormal_URL
  - Redirect
  - on_mouseover
  - RightClick
```

utils – Supporting Functions for File Handling, Model Saving, and Metric Evaluation

The utils directory contains helper scripts that are used across multiple pipeline components. It is organized into submodules to separate general-purpose utilities and ML-specific operations.

Network_security > utils > main_utils > utils.py > save_numpy_array > dir_path

```python
import yaml
from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging.logger import logging
import os,sys
import numpy as np
import dill
import pickle
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
def read_yaml_file(file_path: str)->dict:
    try:
        with open(file_path,"rb") as yaml_file:
            return yaml.safe_load(yaml_file)
    except Exception as e:
        raise NetworkSecurityException(e,sys)
def write_yaml_file(file_path: str, content: object, replace: bool = False) -> None:
    try:
        if replace:
            if os.path.exists(file_path):
                os.remove(file_path)
        os.makedirs(os.path.dirname(file_path), exist_ok=True)
        with open(file_path, "w") as file:
            yaml.dump(content, file)
    except Exception as e:
        raise NetworkSecurityException(e, sys)
```

Opens the YAML file in binary read mode and assigns the file object to the variable yaml_file

Loads the file content and converts it into a Python dictionary

If file already exists, it deletes the existing file before proceeding

Opens the file in write mode and writes the given Python object into YAML format and saves it to the file

By Kunal P. Shukla

data_validation.py – Validating Schema and Detecting Data Drift

Network_security > components > data_validation.py > DataValidation > detect_database_drift

```python
from Network_security.entity.artifact_entity import DataIngestionArtifact,DataValidationArtifact
from Network_security.entity.config_entity import DataValidationConfig

from Network_security.logging.logger import logging
from Network_security.exception.exception import NetworkSecurityException
from Network_security.constants.training_pipeline import SCHEMA_FILE_PATH
from scipy.stats import ks_2samp
import pandas as pd
import os, sys
from Network_security.utils.main_utils.utils import read_yaml_file,write_yaml_file

class DataValidation:
    def __init__(self,data_ingestion_artifact:DataIngestionArtifact,
                 data_validation_config:DataValidationConfig):
        try:
            self.data_ingestion_artifact=data_ingestion_artifact
            self.data_validation_config=data_validation_config
            self.schema_config=read_yaml_file(SCHEMA_FILE_PATH)
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Used to compare the distribution of numerical columns between train and test sets to detect data drift

Initializes the DataValidation class with ingestion artifacts, config settings, and the parsed schema.yaml.

Network_security > components > data_validation.py > DataValidation > detect_database_drift

```python
class DataValidation:
    @staticmethod
    def read_data(file_path)->pd.DataFrame:
        try:
            return pd.read_csv(file_path)
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def validate_number_of_columns(self,dataframe:pd.DataFrame)->bool:
        try:
            number_of_columns=len(self.schema_config['columns'])
            logging.info(f"Required no of columns:{number_of_columns}")
            logging.info(f"Data frame columns:{len(dataframe.columns)}")
            if len(dataframe.columns)==number_of_columns:
                return True
            return False
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Reads a CSV file using pd.read_csv() and returns a DataFrame; defined as a static method since it doesn't use class variables.

Gets the expected number of columns from the schema file

Compares the number of columns in the DataFrame with the expected count and returns True if they match.

Network_security > components > data_validation.py > DataValidation > detect_database_drift

```python
class DataValidation:
    def validate_number_of_numerical_columns(self,dataframe:pd.DataFrame)->bool:
        try:
            num_col_list=[]
            for i in dataframe.columns:
                if dataframe[i].dtype=='int64':
                    num_col_list.append(i)

            number_of_numerical_columns=len(self.schema_config['numerical_columns'])
            logging.info(f"Required no of numerical columns:{number_of_numerical_columns}")
            logging.info(f"Data frame numerical columns:{len(num_col_list)}")
            if len(num_col_list)==number_of_numerical_columns:
                return True
            return False
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Loops through the DataFrame columns and collects all columns with int64 type into num_col_list.

Gets the expected count of numerical columns in the schema

Returns True if the count of int64 columns matches the number defined in the schema.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Network_security > components > data_validation.py > DataValidation > initiate_data_validation

```python
class DataValidation:
    def detect_database_drift(self,base_df,current_df,thresold=0.05)->0.05:
        try:
            status=True
            report={}
            for column in base_df.columns:
                d1=base_df[column]
                d2=current_df[column]
                is_same_dist=ks_2samp(d1,d2)
                if thresold<=is_same_dist.pvalue:
                    is_found=False
                else:
                    is_found=True
                    status=False
                report.update({column:{
                                'p_value':float(is_same_dist.pvalue),
                                'drift_status':is_found
                                }})
            drift_report_file_path=self.data_validation_config.drift_report_file_path
            dir_path=os.path.dirname(drift_report_file_path)
            os.makedirs(dir_path,exist_ok=True)
            write_yaml_file(file_path=drift_report_file_path,content=report)

        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Iterates over each column in the base dataset and retrieves corresponding columns from both DataFrames.

Performs a Kolmogorov-Smirnov test to compare the distribution of the column in both datasets

If p-value is less than the threshold (default 0.05), drift is detected (is_found = True) and status is marked as False.

Adds drift info for the current column to the report dictionary — includes p-value and drift status

drift_report_file_path = Artifacts/04_17_2025_13_45_20/data validation/drift_report/report.yaml

Network_security > components > data_validation.py > DataValidation > initiate_data_validation

```python
class DataValidation:
    def initiate_data_validation(self)->DataValidationArtifact:
        try:
            train_file_path=self.data_ingestion_artifact.trained_file_path
            test_file_path=self.data_ingestion_artifact.test_file_path
            train_dataframe=DataValidation.read_data(train_file_path)
            test_dataframe=DataValidation.read_data(test_file_path)
            status=self.validate_number_of_columns(dataframe=train_dataframe)
            if not status:
                error_message=f"Train dataframe does not contain all columns. \n"
            status=self.validate_number_of_columns(dataframe=test_dataframe)
            if not status:
                error_message=f"Test dataframe does not contain all columns. \n"
            status=self.validate_number_of_numerical_columns(dataframe=train_dataframe)
            if not status:
                error_message=f"Train dataframe does not contain same numbers of numerical columns. \n"
            status=self.validate_number_of_numerical_columns(dataframe=test_dataframe)
            if not status:
                error_message=f"Test dataframe does not contain same numbers of numerical columns. \n"
```

Reads train and test CSV files from ingestion artifacts and loads them as DataFrames

Checks if train and test DataFrames match the expected column counts from the schema

Network_security > components > data_validation.py > DataValidation > initiate_data_validation

```python
class DataValidation:
    def initiate_data_validation(self)->DataValidationArtifact:
            status=self.detect_database_drift(base_df=train_dataframe,current_df=test_dataframe)
            dir_path=os.path.dirname(self.data_validation_config.valid_train_file_path)
            os.makedirs(dir_path,exist_ok=True)
            train_dataframe.to_csv(self.data_validation_config.valid_train_file_path,index=False,header=True)
            test_dataframe.to_csv(self.data_validation_config.valid_test_file_path,index=False,header=True)
            data_validation_artifact = DataValidationArtifact(
                validation_status=status,
                valid_train_file_path=self.data_ingestion_artifact.trained_file_path,
                valid_test_file_path=self.data_ingestion_artifact.test_file_path,
                invalid_train_file_path=None,
                invalid_test_file_path=None,
                drift_report_file_path=self.data_validation_config.drift_report_file_path,
            )
            return data_validation_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Creates a structured artifact containing validation result, file paths, and drift report path

By Kunal P. Shukla

artifact_entity.py – DataValidationArtifact Definition

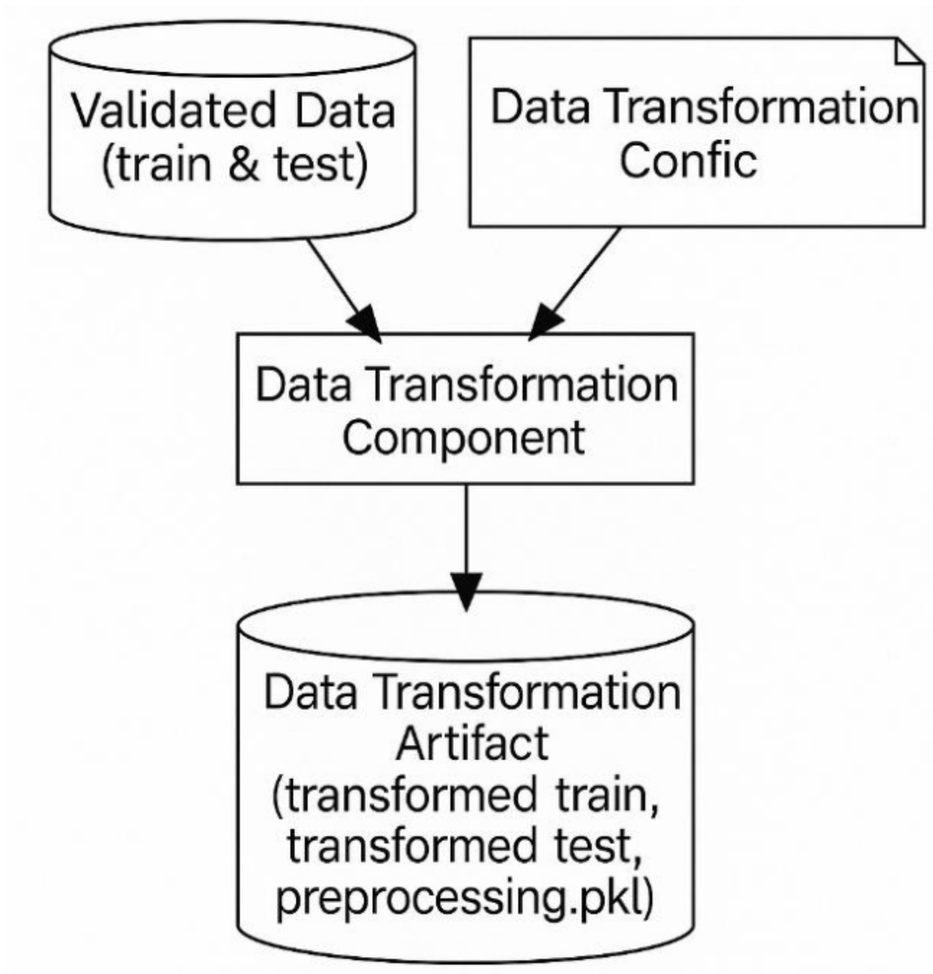
Defines the structure for storing data validation outputs like validation status, valid/invalid file paths, and drift report location. This artifact is returned at the end of the validation component.

Network_security > entity > artifact_entity.py > DataTransformationArtifact > transformed_train_file_path

```python
@dataclass
class DataValidationArtifact:
    validation_status:bool
    valid_train_file_path:str
    valid_test_file_path:str
    invalid_train_file_path:str
    invalid_test_file_path:str
    drift_report_file_path:str
```

Data Transformation Flow:

Transforming and Saving Data with preprocessing.pkl



training_pipeline/init.py - Constants for Data Transformation Configuration

Network_security > constants > training_pipeline > __init__.py > ...

```python
PREPROCESSING_OBJECT_FILE_NAME="preprocessing.pkl"

DATA_TRANSFORMATION_DIR_NAME:str="data transformation"
DATA_TRANSFORMATION_TRANSFORMED_DATA_DIR:str="transformed"
DATA_TRANSFORMATION_TRANSFORMED_OBJECT_DIR:str="transformed_object"
# knn Imputer for nan values
DATA_TRANSFORMATION_IMPUTER_PARAMS: dict={
    "missing_values":np.nan,
    "n_neighbors":3,
    "weights":"uniform",
}
```

KNNImputer replaces missing values using the average of the 3 nearest rows based on feature similarity, preserving local data patterns. It's more contextual than mean/median imputation and works well when data isn't too sparse

data_transformation_config.py – Defining Paths and Parameters

Network_security > entity > config_entity.py > ModelTrainerConfig > __init__

```python
class DataTransformationConfig:
    def __init__(self,training_pipeline_config:TrainingPipelineConfig):
        self.data_transformation_dir=os.path.join(training_pipeline_config.artifact_dir,
            training_pipeline.DATA_TRANSFORMATION_DIR_NAME)
        self.transformed_train_file_path: str = os.path.join( self.data_transformation_dir,
            training_pipeline.DATA_TRANSFORMATION_TRANSFORMED_DATA_DIR,
            training_pipeline.TRAIN_FILE_NAME.replace("csv", "npy"))
        self.transformed_test_file_path: str = os.path.join(self.data_transformation_dir,
            training_pipeline.DATA_TRANSFORMATION_TRANSFORMED_DATA_DIR,
            training_pipeline.TEST_FILE_NAME.replace("csv", "npy") )
        self.transformed_object_file_path: str = os.path.join(self.data_transformation_dir,
            training_pipeline.DATA_TRANSFORMATION_TRANSFORMED_OBJECT_DIR,
            training_pipeline.PREPROCESSING_OBJECT_FILE_NAME)
```

self.data_transformation_dir → Artifacts/04_17_2025_13_45_20/data transformation

self.transformed_train_file_path → Artifacts/04_17_2025_13_45_20/data transformation/transformed/train.npy

self.transformed_test_file_path → Artifacts/04_17_2025_13_45_20/data transformation/transformed/test.npy

self.transformed_object_file_path → Artifacts/04_17_2025_13_45_20/data transformation/transformed_object/preprocessing.pkl

utils.py – Saving NumPy Arrays and Python Objects for Pipeline Persistence

Network_security > utils > main_utils > utils.py > write_yaml_file

```python
def save_numpy_array(file_path:str,array:np.array):
    try:
        dir_path=os.path.dirname(file_path)
        os.makedirs(dir_path,exist_ok=True)
        with open(file_path,"wb") as file_obj:
            np.save(file_obj,array)
    except Exception as e:
        raise NetworkSecurityException(e,sys)

def save_object(file_path:str,obj:object)->None:
    try:
        logging.info("Entered the save_object method of MainUtils class")
        dir_path=os.path.dirname(file_path)
        os.makedirs(dir_path,exist_ok=True)
        with open(file_path,"wb") as file_obj:
            pickle.dump(obj,file_obj)
        logging.info("Exited the save_object method of MainUtils class")
    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Opens the file in binary write mode and saves the NumPy array using np.save(). This is used for .npy file saving

Serializes the object using pickle and writes it to disk in binary format (commonly used for saving models, transformers)

Phishing Website Detection – End-to-End ML Pipeline Project

data_transformation.py – Generating Transformed Arrays and Preprocessor for Model Training

Network_security > components > data_transformation.py > DataTransformation > get_data_transformer_object

```python
import sys,os
import numpy as np
import pandas as pd
from sklearn.impute import KNNImputer
from sklearn.pipeline import Pipeline
from Network_security.logging.logger import logging
from Network_security.exception.exception import NetworkSecurityException
from Network_security.constants.training_pipeline import TARGET_COLUMN
from Network_security.constants.training_pipeline import DATA_TRANSFORMATION_IMPUTER_PARAMS
from Network_security.entity.artifact_entity import DataTransformationArtifact,DataValidationArtifact
from Network_security.entity.config_entity import DataTransformationConfig
from Network_security.utils.main_utils.utils import save_numpy_array,save_object
class DataTransformation:
    def __init__(self,data_validation_artifact:DataValidationArtifact,
                 data_transforamtion_config:DataTransformationConfig):
        try:
            self.data_validation_artifact:DataValidationArtifact=data_validation_artifact
            self.data_transforamtion_config:DataTransformationConfig=data_transforamtion_config
        except Exception as e:
            raise NetworkSecurityException(e,sys)
    @staticmethod
    def read_data(file_path)->pd.DataFrame:
        try:
            return pd.read_csv(file_path)
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Network_security > components > data_transformation.py > DataTransformation > initiate_data_transformation

```python
class DataTransformation:
    def get_data_transformer_object(cls)->Pipeline:
        logging.info("Entered get_data_transformer_object method of DataTransformation class")
        try:
            imputer:KNNImputer=KNNImputer(**DATA_TRANSFORMATION_IMPUTER_PARAMS)
            logging.info("initialize KNNImputer with {DATA_TRANSFORMATION_IMPUTER_PARAMS}")
            processor:Pipeline=Pipeline([("imputer",imputer)])
            return processor
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

unpacks each key–value pair from the dictionary and passes them as individual keyword arguments to the KNNImputer

Even though Pipeline currently wraps only KNNImputer, using it ensures safe and consistent preprocessing. It becomes essential when adding steps like StandardScaler or PCA, as it prevents data leakage by fitting only on training data and applying the same logic to test data. This prevents the model from unintentionally learning patterns from the test set during preprocessing.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Network_security > components > data_transformation.py > DataTransformation > initiate_data_transformation

```python
class DataTransformation:
    def initiate_data_transformation(self)->DataTransformationArtifact:
        logging.info("Entered initiate_data_transformation method of DataTransformation class")
        try:
            logging.info("Starting data transformation")
            train_df=DataTransformation.read_data(self.data_validation_artifact.valid_train_file_path)
            test_df=DataTransformation.read_data(self.data_validation_artifact.valid_test_file_path)

            input_feature_train_df=train_df.drop(columns=TARGET_COLUMN,axis=1)
            target_feature_train_df=train_df[TARGET_COLUMN]
            target_feature_train_df=target_feature_train_df.replace(-1,0)

            input_feature_test_df=test_df.drop(columns=TARGET_COLUMN,axis=1)
            target_feature_test_df=test_df[TARGET_COLUMN]
            target_feature_test_df=target_feature_test_df.replace(-1,0)

            preprocessor=self.get_data_transformer_object()

            preprocessor_object=preprocessor.fit(input_feature_train_df)
            transformed_input_train_feature=preprocessor_object.transform(input_feature_train_df)
            transformed_input_test_feature=preprocessor_object.transform(input_feature_test_df)

            train_arr=np.c_[transformed_input_train_feature,np.array(target_feature_train_df)]
            test_arr=np.c_[transformed_input_test_feature,np.array(target_feature_test_df)]
```

Separates features and target from train set and converts label -1 to 0

Fits the preprocessing pipeline on training features and applies the same transformation to both train and test sets.

Combines transformed features and target arrays into final NumPy arrays for model training

Network_security > components > data_transformation.py > DataTransformation > initiate_data_transformation

```python
class DataTransformation:
    def initiate_data_transformation(self)->DataTransformationArtifact:
            save_numpy_array(self.data_transforamtion_config.transformed_train_file_path,array=train_arr)
            save_numpy_array(self.data_transforamtion_config.transformed_test_file_path,array=test_arr)
            save_object(self.data_transforamtion_config.transformed_object_file_path,preprocessor_object)

            save_object("final_model/preprocessor.pkl",preprocessor_object)


            data_transformation_artifact=DataTransformationArtifact(
                transformed_object_file_path=self.data_transforamtion_config.transformed_object_file_path,
                transformed_train_file_path=self.data_transforamtion_config.transformed_train_file_path,
                transformed_test_file_path=self.data_transforamtion_config.transformed_test_file_path
            )
            return data_transformation_artifact

        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

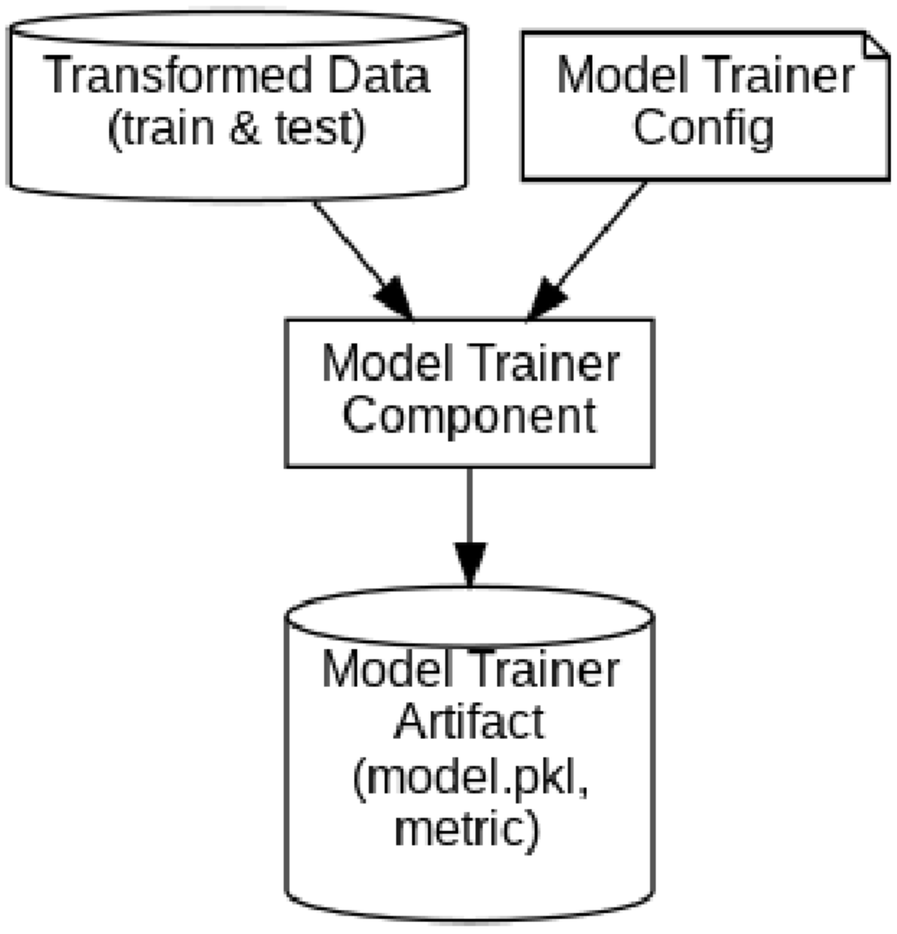
Saves transformed train and test arrays as .npy files

Serializes and saves the preprocessor object for reuse during inference

By Kunal P. Shukla

Model Trainer Flow

Training and Saving Model with model.pkl



training_pipeline/init.py - Constants for Model Trainer Configuration

Network_security > constants > training_pipeline > __init__.py > DATA_TRANSFORMATION_IMPUTER_PARAMS

```python
MODEL_TRAINER_DIR_NAME:str="model trainer"
MODEL_TRAINER_TRAINED_MODEL_DIR:str="trained_model"
MODEL_TRAINER_TRAINED_MODEL_NAME:str="model.pkl"
MODEL_TRAINER_EXPECTED_SCORE:float=0.6
MODEL_TRAINER_OVER_FITTING_UNDER_FITTING_THRESOLD:float=0.05

SAVED_MODEL_DIR=os.path.join("saved_models")
MODEL_FILE_NAME="model.pkl"

TRAINING_BUCKET_NAME="kunalawsbucketns"
```

model_trainer_config.py – Defining Paths and Parameters for Model Training

```python
class ModelTrainerConfig:
    def __init__(self,training_pipleline_config:TrainingPipelineConfig):
        self.model_trainer_config_dir=os.path.join(
            training_pipleline_config.artifact_dir,training_pipeline.MODEL_TRAINER_DIR_NAME
        )
        self.trained_model_file_path=os.path.join(
            self.model_trainer_config_dir,training_pipeline.MODEL_TRAINER_TRAINED_MODEL_DIR,
            training_pipeline.MODEL_TRAINER_TRAINED_MODEL_NAME
        )
        self.expected_accuracy:float=training_pipeline.MODEL_TRAINER_EXPECTED_SCORE
        self.overfitting_underfitting_thresold=training_pipeline.MODEL_TRAINER_OVER_FITTING_UNDER_FITTING_THRESOLD
```

self.model_trainer_config_dir → Artifacts/04_17_2025_13_45_20/model trainer

self.trained_model_file_path → Artifacts/04_17_2025_13_45_20/model trainer/trained_model/model.pkl

By Kunal P. Shukla

utils.py – Loading Pickle Objects and NumPy Arrays for Reuse in Pipeline

Network_security > utils > main_utils > utils.py > load_object

```python
def load_object(file_path:str,)->object:
    try:
        if not os.path.exists(file_path):
            raise Exception("The file: {file_path} does not exist")
        with open(file_path,'rb') as file_obj:
            print(file_obj)
            return pickle.load(file_obj)
    except Exception as e:
        raise NetworkSecurityException(e,sys)

def load_numpy_array(file_path:str)->np.array:
    try:
        with open(file_path,'rb') as file_obj:
            return np.load(file_obj)
    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Checks if the given file exists; raises an error if not found

Opens the file in binary mode and loads the object using pickle

Reads .npy file in binary mode and returns it as a NumPy array using np.load

Network_security > utils > main_utils > utils.py > load_object

```python
def evaluate_models(x_train,y_train,x_test,y_test,models,param):
    try:
        report={}
        for i  in range(len(list(models))):
            model=list(models.values())[i]
            para=param[list(models.keys())[i]]

            gs=GridSearchCV(model,para,cv=3)
            gs.fit(x_train,y_train)

            model.set_params(**gs.best_params_)
            model.fit(x_train,y_train)

            y_train_pred=model.predict(x_train)
            y_test_pred=model.predict(x_test)

            train_model_score = accuracy_score(y_train, y_train_pred)
            test_model_score = accuracy_score(y_test, y_test_pred)

            report[list(models.keys())[i]] = test_model_score

        return report

    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Iterates through each model in the models dictionary and fetches the model along with its corresponding hyperparameters for tuning

Performs hyperparameter tuning using GridSearchCV with 3-fold cross-validation

Updates the model with best-found parameters and retrains it on the full training set

Generates predictions for both training and testing data

Stores the test accuracy of each model in the report dictionary

classification_report.py – Calculating F1, Precision, and Recall Scores

Network_security > utils > ml_utils > metric > classification_report.py > get_classification_score

```python
import os,sys
from Network_security.entity.artifact_entity import ClassificationMetricArtifact
from Network_security.exception.exception import NetworkSecurityException
from sklearn.metrics import f1_score,recall_score,precision_score

def get_classification_score(y_true,y_pred):
    try:
        model_f1_score=f1_score(y_true,y_pred)
        model_recall_score=recall_score(y_true,y_pred)
        model_precision_score=precision_score(y_true,y_pred)

        classification_metric=ClassificationMetricArtifact(f1_score=model_f1_score,
                                                           recall_score=model_recall_score,
                                                           precision_score=model_precision_score)
        return classification_metric
    except Exception as e:
        raise NetworkSecurityException(e,sys)
```

Calculates F1, recall, and precision scores from predicted and actual labels, then wraps them into a structured artifact for downstream evaluation use

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

model_trainer.py – Training Models with Hyperparameter Tuning and Accuracy Evaluation

Network_security > components > model_trainer.py > ModelTrainer > __init__

```python
import os,sys
from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging.logger import logging
from Network_security.entity.config_entity import ModelTrainerConfig
from Network_security.entity.artifact_entity import DataTransformationArtifact,ModelTrainerArtifact
from Network_security.utils.main_utils.utils import save_object,load_object,load_numpy_array,evaluate_models
from Network_security.utils.ml_utils.metric.classification_report import get_classification_score
from Network_security.utils.ml_utils.model.estimator import NetworkModel

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier
import mlflow
```

```python
class ModelTrainer:
    def __init__(self,model_trainer_config:ModelTrainerConfig,data_transforamtion_artifact:DataTransformationArtifact):
        try:
            self.model_trainer_config=model_trainer_config
            self.data_transforamtion_artifact=data_transforamtion_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def track_mlflow(self,best_model,classification_train_metric):
        with mlflow.start_run():
            f1_score=classification_train_metric.f1_score
            precision_score=classification_train_metric.precision_score
            recall_score=classification_train_metric.recall_score

            mlflow.log_metric("f1_score",f1_score)
            mlflow.log_metric("precision_score",precision_score)
            mlflow.log_metric("recall_score",recall_score)
            mlflow.sklearn.log_model(best_model,"model")
```

Starts an MLflow tracking session to log metrics and the model

Extracts evaluation metrics from the classification artifact object

Logs evaluation metrics and the trained model to the current MLflow run for tracking, versioning, and retrieval

Network_security > components > model_trainer.py > ModelTrainer > track_mlflow

```python
class ModelTrainer:
    def train_model(self,x_train,y_train,x_test,y_test):
        models={
            "Random Forest":RandomForestClassifier(verbose=1),
            "Decision Tree":DecisionTreeClassifier(),
            "Gradient Boosting":GradientBoostingClassifier(verbose=1),
            "Logistic Regression":LogisticRegression(verbose=1),
            "AdaBoost":AdaBoostClassifier()
        }
        params={
            "Decision Tree": {
                'criterion':['gini', 'entropy', 'log_loss'],
                # 'splitter':['best','random'],
                # 'max_features':['sqrt','log2'],
            },
            "Random Forest":{
                # 'criterion':['gini', 'entropy', 'log_loss'],

                # 'max_features':['sqrt','log2',None],
                'n_estimators': [8,16,32,128,256]
            },
            "Gradient Boosting":{
                # 'loss':['log_loss', 'exponential'],
                'learning_rate':[.1,.01,.05,.001],
                'subsample':[0.6,0.7,0.75,0.85,0.9],
                # 'criterion':['squared_error', 'friedman_mse'],
```

Defines a dictionary of ML classifiers to be evaluated

Specifies the hyperparameter grids for each model to be used during GridSearchCV tuning

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Network_security > components > model_trainer.py > ModelTrainer > train_model

```
18    class ModelTrainer:
37        def train_model(self,x_train,y_train,x_test,y_test):
71            model_report:dict=evaluate_models(x_train=x_train,y_train=y_train,x_test=x_test,y_test=y_test,
72                                              models=models,param=params)
73            best_model_score=max(sorted(model_report.values()))
74            best_model_name=list(model_report.keys())[
75                list(model_report.values()).index(best_model_score)
76            ]
77            best_model=models[best_model_name]
78            y_train_pred=best_model.predict(x_train)
79            classification_train_metric=get_classification_score(y_true=y_train,y_pred=y_train_pred)
80            # Track the mlflow
81            self.track_mlflow(best_model,classification_train_metric)
82            y_test_pred=best_model.predict(x_test)
83            classification_test_metric=get_classification_score(y_true=y_test,y_pred=y_test_pred)
84            self.track_mlflow(best_model,classification_test_metric)
85            preprocessor=load_object(file_path=self.data_transforamtion_artifact.transformed_object_file_path)
86            model_dir_path=os.path.dirname(self.model_trainer_config.trained_model_file_path)
87            os.makedirs(model_dir_path,exist_ok=True)
88            Network_model=NetworkModel(preprocessor=preprocessor,model=best_model)
89            save_object(self.model_trainer_config.trained_model_file_path,obj=Network_model)
90            save_object("final_model/model.pkl",best_model)
```

Evaluates all models with hyperparameter tuning and returns a dictionary of model names and their test scores

Gets the model name with the highest test score by matching the score to its corresponding key

Logs training metrics and model into MLflow for tracking and versioning

Loads the saved preprocessor.pkl used during transformation to maintain consistency in inference

Wraps both preprocessor and model into a custom class, then saves it. Also saves the plain model separately for reference

Network_security > components > model_trainer.py > ModelTrainer > train_model

```
18    class ModelTrainer:
37        def train_model(self,x_train,y_train,x_test,y_test):
93            model_trainer_artifact=ModelTrainerArtifact(trained_model_file_path=
94                                                        self.model_trainer_config.trained_model_file_path,
95                                 train_metric_artifact=classification_train_metric,
96                                 test_metric_artifact=classification_test_metric)
97
98            logging.info(f"Model trainer artifact: {model_trainer_artifact}")
99            return model_trainer_artifact
100
101       def initiate_model_trainer(self)->ModelTrainerArtifact:
102           try:
103               trainer_file_path=self.data_transforamtion_artifact.transformed_train_file_path
104               test_file_path=self.data_transforamtion_artifact.transformed_test_file_path
105
106               train_arr=load_numpy_array(trainer_file_path)
107               test_arr=load_numpy_array(test_file_path)
108
109               x_train,y_train,x_test,y_test=(
110                   train_arr[:,:-1],
111                   train_arr[:,-1],
112                   test_arr[:,:-1],
113                   test_arr[:,-1]
114               )
```

Loads the NumPy arrays containing transformed features and target from disk

Splits the arrays into input features (x) and target labels (y) for both train and test sets

artifact_entity.py –Artifacts for Transformation, Metrics, and Model Trainer Output

```
23    @dataclass
24    class ClassificationMetricArtifact:
25        f1_score:float
26        precision_score:float
27        recall_score:float
28
29    @dataclass
30    class ModelTrainerArtifact:
31        trained_model_file_path:str
32        train_metric_artifact:ClassificationMetricArtifact
33        test_metric_artifact:ClassificationMetricArtifact
```

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

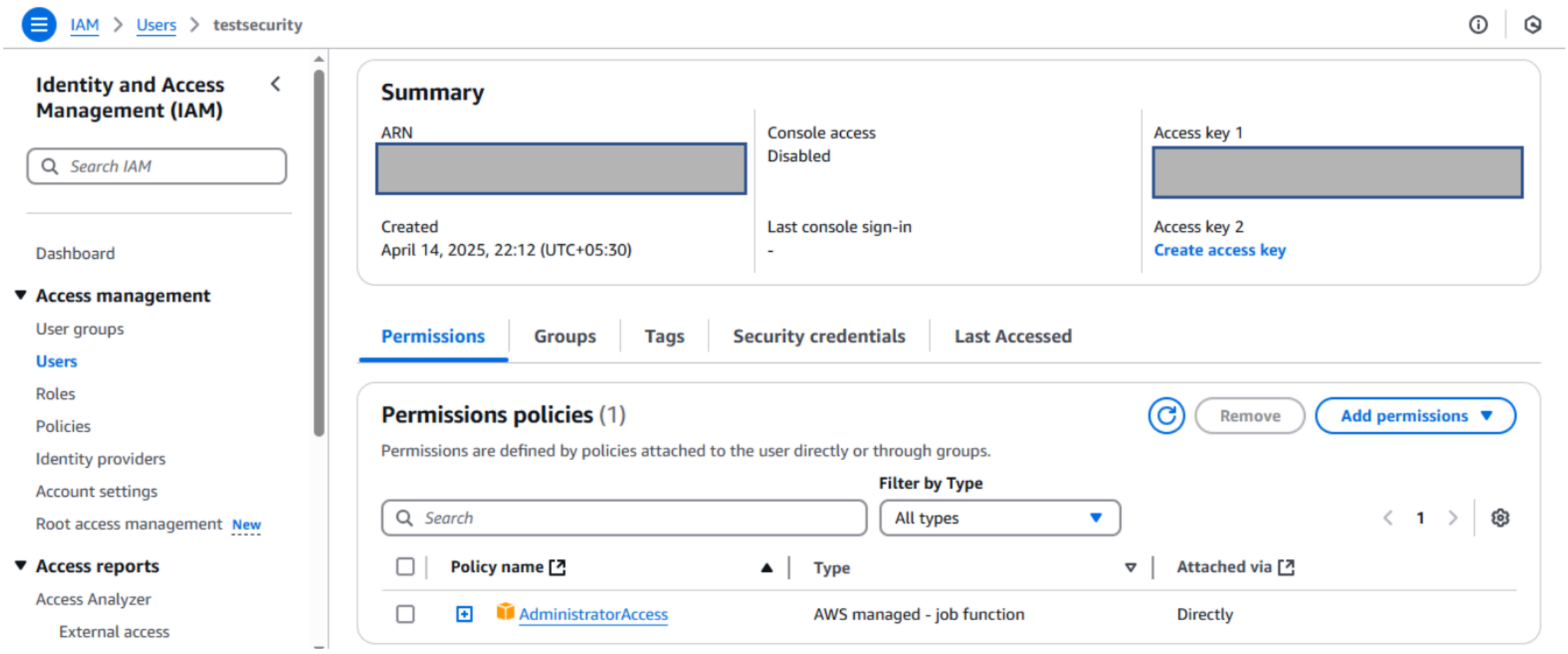
Create an IAM User

Foundation for Secure and Automated AWS Access

The very first step when working with AWS programmatically is to create an **IAM (Identity and Access Management) user**. This user allows secure, credential-based access to AWS services through CLI tools, SDKs like boto3, or automation pipelines.

Creating an IAM User

- Navigate to the AWS Management Console → IAM → Users → Add user
- Enable Programmatic access to generate the Access Key ID and Secret Access Key
- Assign required permissions: For full access: attach the AdministratorAccess policy



Why AWS S3?

Making ML Pipeline Cloud-Ready and Production-Friendly

AWS S3 is used in this project to store important pipeline artifacts like the trained model (model.pkl), preprocessing object (preprocessor.pkl), and drift reports securely in the cloud.

This solves three key problems:

1. **Storage** – Keeps your model and artifacts safe even if your local environment is cleaned or crashes.
2. **Access** – Makes your trained models accessible from anywhere (e.g., EC2, Lambda, or other APIs).
3. **Deployment-Ready** – Cloud storage like S3 is essential when moving towards real-time deployments, CI/CD pipelines, or serverless architecture.

By syncing both the artifact directory and final model to S3, the pipeline becomes portable, scalable, and ready for production.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

s3_syncer.py – Syncing Artifacts like preprocessor.pkl to AWS S3

This file marks the beginning of the cloud integration step.
The S3Sync class defines a method sync_folder_to_s3, which uses the AWS CLI to upload folders (like final models, drift reports, and preprocessing objects) to a specified S3 bucket for storage, sharing, or deployment.

Network_security > cloud > s3_syncer.py > ...

```python
import os

class S3Sync:
    def sync_folder_to_s3(self,folder,aws_bucket_url):
        command=f"aws s3 sync {folder} {aws_bucket_url}"
        os.system(command)
```

Defines a method that takes the local folder path and target S3 bucket URL as inputs

Builds the AWS CLI command to sync the local folder with the specified S3 bucket

main.py – Running the Complete Training Pipeline and Syncing Artifacts to S3

Network_security > pipeline > training_pipeline.py > ...

```python
import sys,os
from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging.logger import logging
from Network_security.components.data_ingestion import DataIngestion
from Network_security.components.data_validation import DataValidation
from Network_security.components.data_transformation import DataTransformation
from Network_security.components.model_trainer import ModelTrainer
from Network_security.entity.config_entity import(
TrainingPipelineConfig,DataIngestionConfig,DataValidationConfig,DataTransformationConfig,ModelTrainerConfig
)
from Network_security.entity.artifact_entity import(
DataIngestionArtifact,DataValidationArtifact,DataTransformationArtifact,ModelTrainerArtifact
)

from Network_security.constants.training_pipeline import TRAINING_BUCKET_NAME
from Network_security.cloud.s3_syncer import S3Sync
from Network_security.constants.training_pipeline import SAVED_MODEL_DIR

class TrainingPipeline:
    def __init__(self):
        self.training_pipeline_config=TrainingPipelineConfig()
        self.s3_sync = S3Sync()
```

Creates an instance of the S3Sync class to handle syncing local folders to AWS S3

Network_security > pipeline > training_pipeline.py > ...

```python
class TrainingPipeline:
    def start_data_ingestion(self):
        try:
            self.data_ingestion_config=DataIngestionConfig(training_pipeline_config=self.training_pipeline_config)
            logging.info("Initiate data ingestion")
            data_ingestion=DataIngestion(data_ingestion_config=self.data_ingestion_config)
            data_ingestion_artifact=data_ingestion.initiate_data_ingestion()
            logging.info(f"Data ingestion completed and artifact:{data_ingestion_artifact}")
            return data_ingestion_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def start_data_validation(self,data_ingestion_artifact:DataIngestionArtifact):
        try:
            data_validation_config=DataValidationConfig(training_pipeline_config=self.training_pipeline_config)
            logging.info("Initiate data validation")
            data_validation=DataValidation(data_ingestion_artifact=data_ingestion_artifact,
                                           data_validation_config=data_validation_config)
            dat_validation_artifact=data_validation.initiate_data_validation()
            logging.info(f"Data validation completed and artifact:{dat_validation_artifact}")
            return dat_validation_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Runs the ingestion process and returns a DataIngestionArtifact containing paths to the train and test CSV files

Validates the ingested data and returns a DataValidationArtifact with valid/invalid file paths and drift report

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

Network_security > pipeline > training_pipeline.py > ...

```python
class TrainingPipeline:
    def start_data_transforamtion(self,data_validation_artifact:DataValidationArtifact):
        try:
            data_transformation_config=DataTransformationConfig(training_pipeline_config=self.training_pipeline_config
            logging.info("Initiate data transformation")
            data_transformation=DataTransformation(data_transforamtion_config=data_transformation_config,
                                                   data_validation_artifact=data_validation_artifact)
            data_transformation_artifact=data_transformation.initiate_data_transformation()
            logging.info(f"Data transformation completed and artifact:{data_transformation_artifact}")
            return data_transformation_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)

    def start_model_trainer(self,data_transformation_artifact=DataTransformationArtifact):
        try:
            model_trainer_config=ModelTrainerConfig(training_pipleline_config=self.training_pipeline_config)
            logging.info("Initiate model trainer")
            model_trainer=ModelTrainer(model_trainer_config=model_trainer_config,
                                       data_transforamtion_artifact=data_transformation_artifact)
            model_trainer_artifact=model_trainer.initiate_model_trainer()
            logging.info(f"Model trainer completed and artifact{model_trainer_artifact}")
            return model_trainer_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Transforms the data and returns a DataTransformationArtifact with transformed arrays and the saved preprocessor

Trains and evaluates models, returning a ModelTrainerArtifact with model path and training/testing metrics

Network_security > pipeline > training_pipeline.py > TrainingPipeline > run_pipeline

```python
class TrainingPipeline:
    def sync_artifact_dir_to_s3(self):
        try:
            aws_bucket_url = f"s3://{TRAINING_BUCKET_NAME}/artifact/{self.training_pipeline_config.timestamp}"
            self.s3_sync.sync_folder_to_s3(folder = self.training_pipeline_config.artifact_dir,aws_bucket_url=aws_buck
        except Exception as e:
            raise NetworkSecurityException(e,sys)
    def sync_saved_model_dir_to_s3(self):
        try:
            aws_bucket_url = f"s3://{TRAINING_BUCKET_NAME}/final_model/{self.training_pipeline_config.timestamp}"
            self.s3_sync.sync_folder_to_s3(folder = self.training_pipeline_config.model_dir,aws_bucket_url=aws_bucket_
        except Exception as e:
            raise NetworkSecurityException(e,sys)
    def run_pipeline(self):
        try:
            print(f"📂 Local artifact folder: {self.training_pipeline_config.artifact_dir}")
            print(f"📂 Local model folder: {self.training_pipeline_config.model_dir}")
            data_ingestion_artifact=self.start_data_ingestion()
            data_validation_artifact=self.start_data_validation(data_ingestion_artifact=data_ingestion_artifact)
            data_transformation_artifact=self.start_data_transforamtion(data_validation_artifact=data_validation_artif
            model_trainer_artifact=self.start_model_trainer(data_transformation_artifact=data_transformation_artifact)
            self.sync_artifact_dir_to_s3()
            self.sync_saved_model_dir_to_s3()
            return model_trainer_artifact
        except Exception as e:
            raise NetworkSecurityException(e,sys)
```

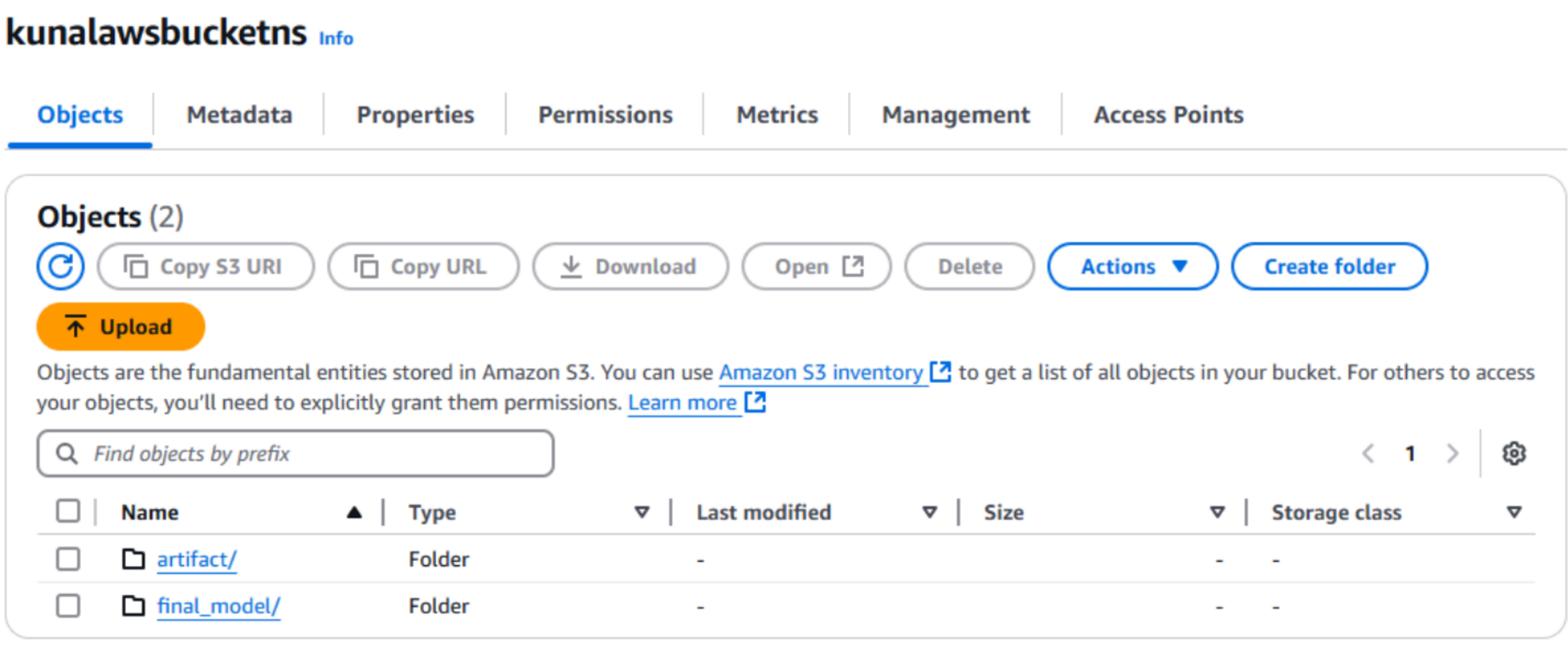
Builds the target S3 path for syncing all generated artifacts using the pipeline timestamp, and uploads the entire local artifact directory to the specified S3 path

Uploads the entire local artifact directory to the specified S3 path, and syncs the final model directory to S3

By Kunal P. Shukla

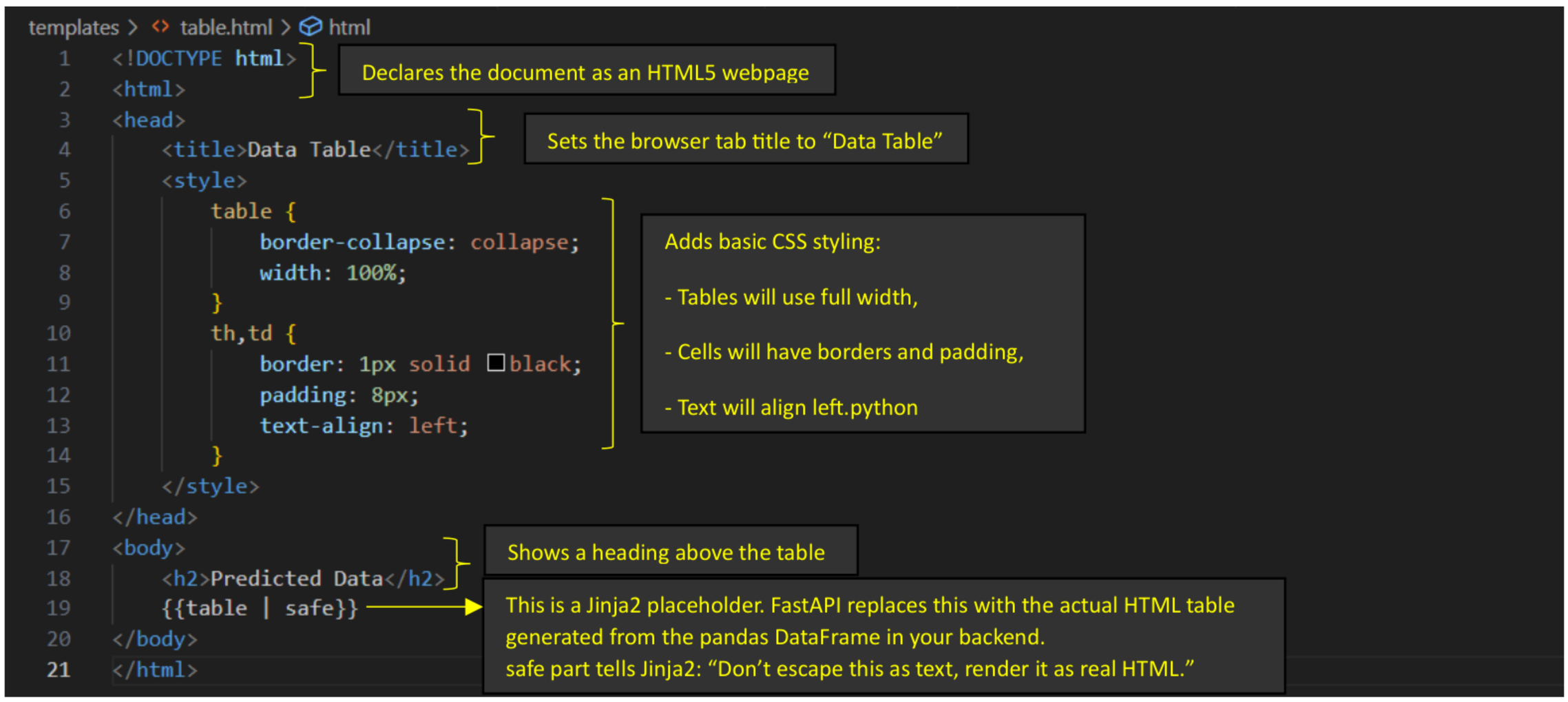
Phishing Website Detection – End-to-End ML Pipeline Project

S3 Bucket Overview – Uploaded Artifacts and Final Model from Training Pipeline



This screenshot shows the contents of your S3 bucket named **kunalawsbucketns**, where two folders—artifact/ and final_model/—have been successfully uploaded. These folders are automatically created and synced to AWS S3 at the end of the training pipeline. The artifact/ folder contains all intermediate outputs like ingested data, validation reports, transformed files, and other generated artifacts during each pipeline stage. The final_model/ folder contains the serialized trained model and preprocessing object, making them easily accessible for deployment. This sync happens as part of the run_pipeline() method, which triggers two S3 upload operations through the methods sync_artifact_dir_to_s3() and sync_saved_model_dir_to_s3() using the AWS CLI in the background.

table.html – Rendering Prediction Results in a Browser-Friendly Table



By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

app.py – FastAPI Interface for Training and Prediction

This file creates a FastAPI application with routes to trigger the training pipeline and generate predictions. It loads the trained model and preprocessor, accepts a CSV input, and returns predictions in a browser-rendered HTML table. The app integrates Jinja2 for templating and supports cross-origin requests for deployment.

app.py > ...

```python
import os,sys
import certifi
ca=certifi.where()
from dotenv import load_dotenv
load_dotenv()
mongo_db_url=os.getenv("MONGODB_URL_KEY")
print(mongo_db_url)
import pymongo
from Network_security.exception.exception import NetworkSecurityException
from Network_security.logging.logger import logging
from Network_security.pipeline.training_pipeline import TrainingPipeline
from fastapi.middleware.cors import CORSMiddleware
from fastapi import FastAPI,Request,UploadFile,File
from uvicorn import run as app_run
from fastapi.responses import Response
from starlette.responses import RedirectResponse
import pandas as pd

from Network_security.utils.main_utils.utils import load_object
from Network_security.utils.ml_utils.model.estimator import NetworkModel


from Network_security.constants.training_pipeline import DATA_INGESTION_DATABASE_NAME,DATA_INGESTION_COLLECTION_NAME
```

app.py > ...

```python
client=pymongo.MongoClient(mongo_db_url,tlsCAFile=ca)
database=client[DATA_INGESTION_DATABASE_NAME]
collection=database[DATA_INGESTION_COLLECTION_NAME]

app=FastAPI()
origins=["*"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)

from fastapi.templating import Jinja2Templates
templates = Jinja2Templates(directory="./templates")

@app.get("/",tags=["authentication"])
async def index():
    return RedirectResponse(url="/docs")
```

Establishes a secure MongoDB connection using certifi for TLS validation

Creates the FastAPI app instance

Allows any frontend (from any domain) to access your API. The * means "allow all"

Adds CORS middleware (Cross-Origin Resource Sharing).

This is required when your frontend (like React, Streamlit, or Postman) calls your backend API from another domain.

Without this, browsers block external requests for security reasons.

Tells FastAPI to look for HTML files (like table.html) inside the templates/ folder

/ redirects to the /docs page, which is the Swagger UI auto-generated by FastAPI — used for API testing

Jinja2 is a powerful templating engine used to generate HTML dynamically by inserting data from the backend into predefined template files. It is commonly used with web frameworks like Flask and FastAPI

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

app.py > index

```python
@app.get("/train")
async def train_route():
    try:
        train_pipeline=TrainingPipeline()
        train_pipeline.run_pipeline()
        return Response("Training is successful")
    except Exception as e:
        raise NetworkSecurityException(e,sys)
@app.post("/predict")
async def predict_route(request: Request,file: UploadFile = File(...)):
    try:
        df=pd.read_csv(file.file)
        preprocesor=load_object("final_model/preprocessor.pkl")
        final_model=load_object("final_model/model.pkl")
        network_model = NetworkModel(preprocessor=preprocesor,model=final_model)
        print(df.iloc[0])
        y_pred = network_model.predict(df)
        print(y_pred)
        df['predicted_column'] = y_pred
        print(df['predicted_column'])
        df.to_csv('prediction_output/output.csv')
        table_html = df.to_html(classes='table table-striped')
        return templates.TemplateResponse("table.html", {"request": request, "table": table_html})
    except Exception as e:
            raise NetworkSecurityException(e,sys)
```

Initializes the entire pipeline class

Runs the full process: ingestion → validation → transformation → model training → saving to S3

This route handles POST requests (with file upload). The user uploads a CSV file, and it's read as file

Loads the saved preprocessor and trained model using your utility functions

Converts the DataFrame into HTML using Bootstrap-like table styling

app.py > predict_route

```python
if __name__=="__main__":
    app_run(app,host="0.0.0.0",port=8080)

print(" Deployment trigger")
```

Runs your FastAPI app using Uvicorn on port 8080.

0.0.0.0 means it's accessible externally (e.g., from browser, Postman, or another server)

Dockerfile

Containerizing the FastAPI Application for AWS Deployment

This Dockerfile defines how the entire machine learning application — including the FastAPI app, model, and all dependencies — is packaged into a lightweight container image, ready for deployment on any platform such as AWS EC2

Dockerfile > ...

```
FROM python:3.10-slim-buster
WORKDIR /app
COPY . /app

RUN apt update -y && apt install awscli -y

RUN apt-get update && pip install -r requirements.txt

CMD ["python3", "app.py"]
```

Uses a lightweight official Python 3.10 base image to keep the container small and efficient

Sets the working directory to /app and copies all local project files into that directory inside the container

Updates the package manager and installs the AWS CLI inside the container, enabling interaction with AWS services like S3

Installs all Python dependencies listed in requirements.txt

Defines the default command that will run when the container starts. This launches the FastAPI app (app.py) using Python

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

GitHub

Pushing Local Machine Learning Project to GitHub

cd path/to/your/project

git init

git add .

git commit -m "Commit message"

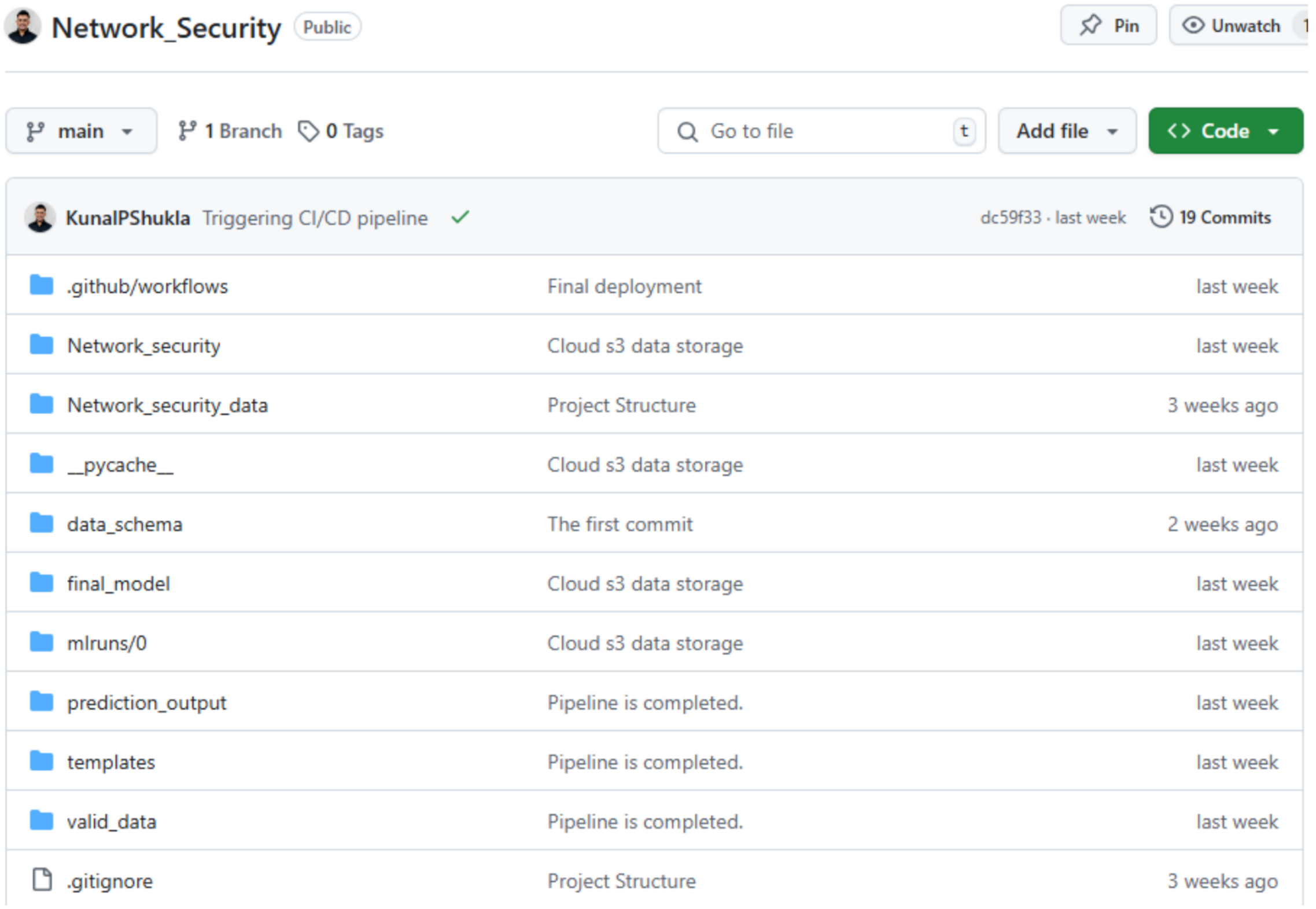
git remote add origin https://github.com/KunalPShukla/Network_Security.git

git branch -M main

git push -u origin main

It is good practice to regularly push changes to GitHub to maintain version control and project backups

Files and folders listed in .gitignore will not be pushed to the repository

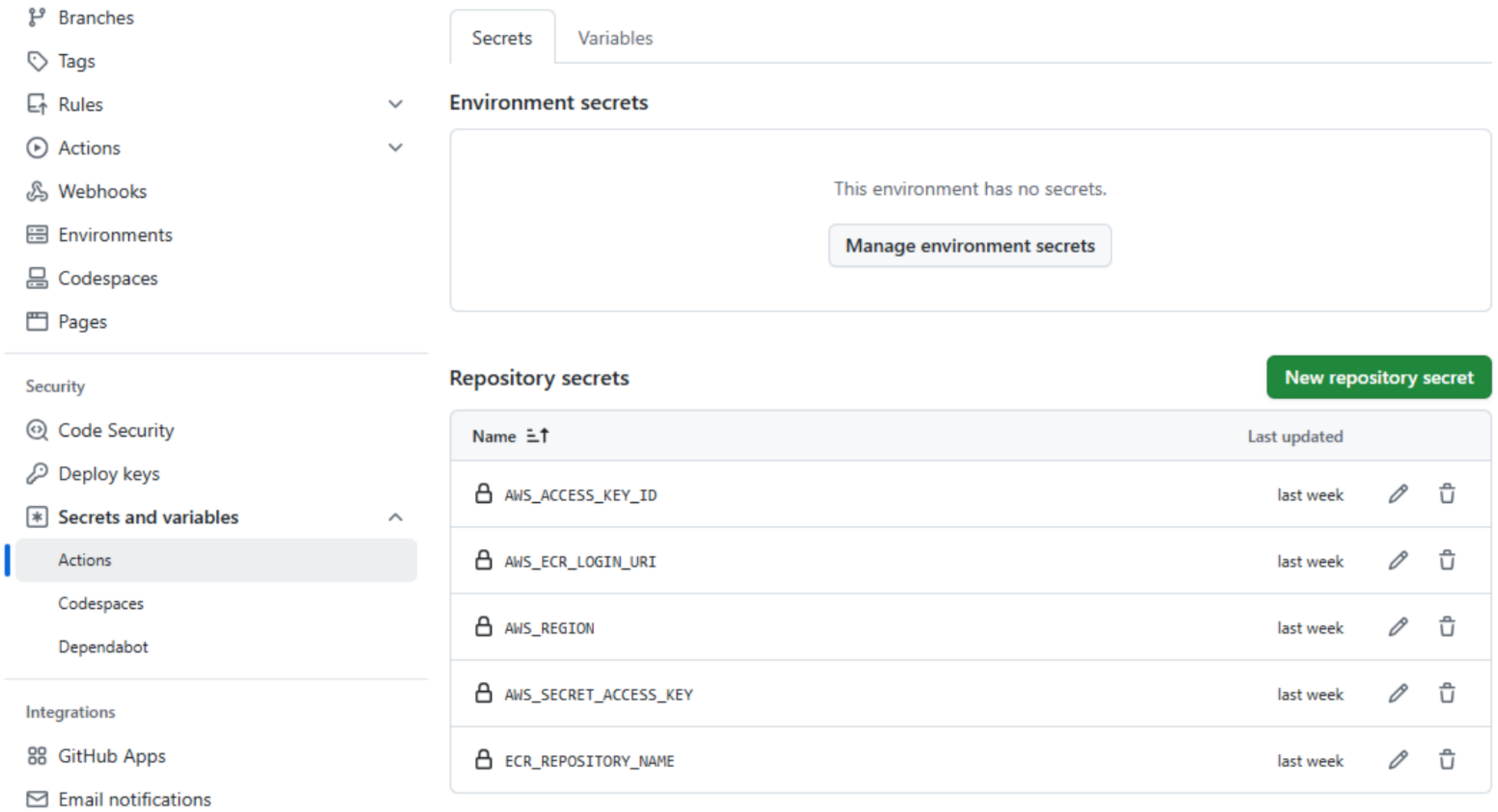


By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

GitHub Secrets – Securely Storing AWS Credentials for CI/CD Pipeline

It ensures that all sensitive AWS credentials (like access keys and login URIs) are safely stored and can be accessed inside the workflow without exposing them in the code



Amazon ECR

Private Repository for ML App Docker Image

This private ECR repository named networksecurity stores the Docker image of the ML application.
The image is built via GitHub Actions and pushed to this repository, from where it is later pulled by an EC2 instance for deployment.

By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

GitHub Actions Workflow

CI/CD Pipeline for Dockerized ML App to AWS ECR & EC2

This GitHub Actions workflow automates the **build, test, and deployment** process of your machine learning FastAPI app

Continuous Integration (CI)

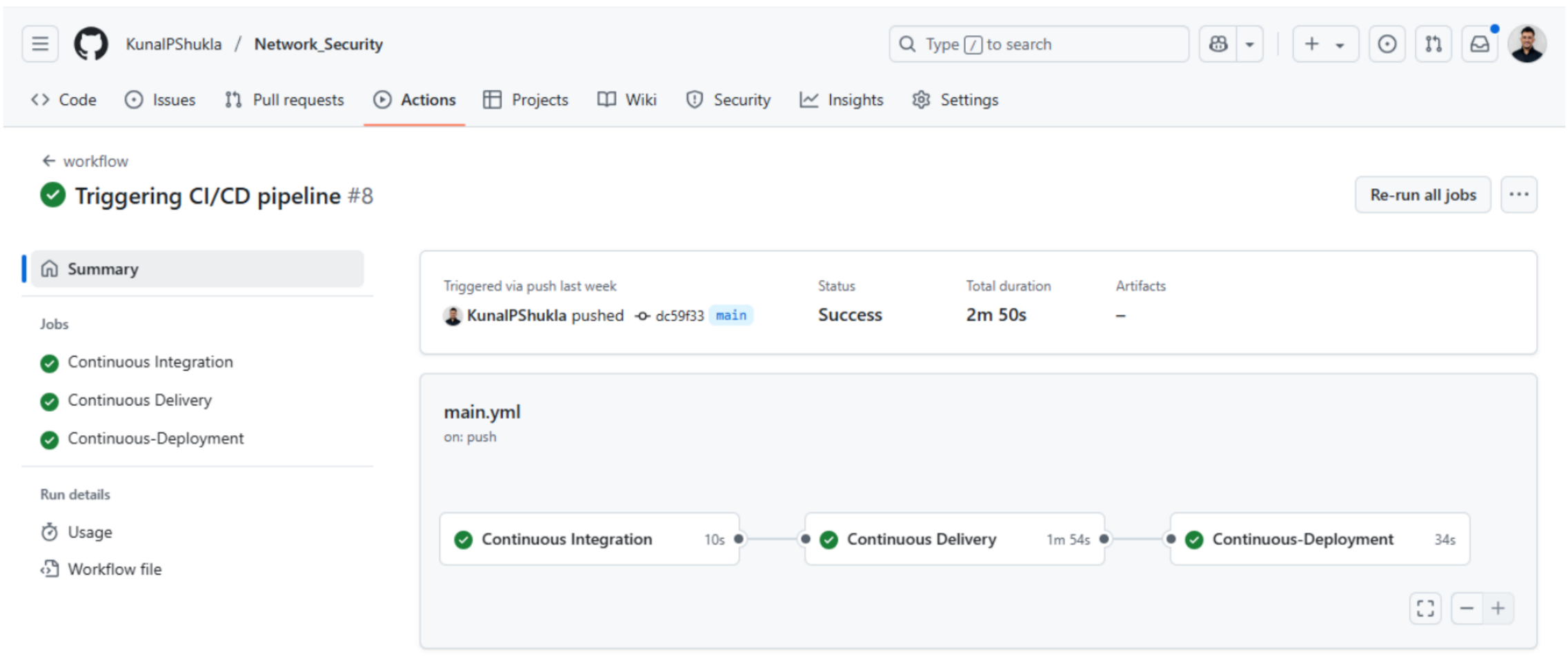
- Linting
- Unit testing
- Runs when a push is made to the main branch

Continuous Delivery (CD)

- Builds and tags a Docker image
- Pushes it to Amazon ECR

Continuous Deployment (CD)

- Runs on a self-hosted EC2 runner
- Pulls the latest image from ECR
- Removes existing containers if running
- Deploys the new image
- Cleans up old images and containers



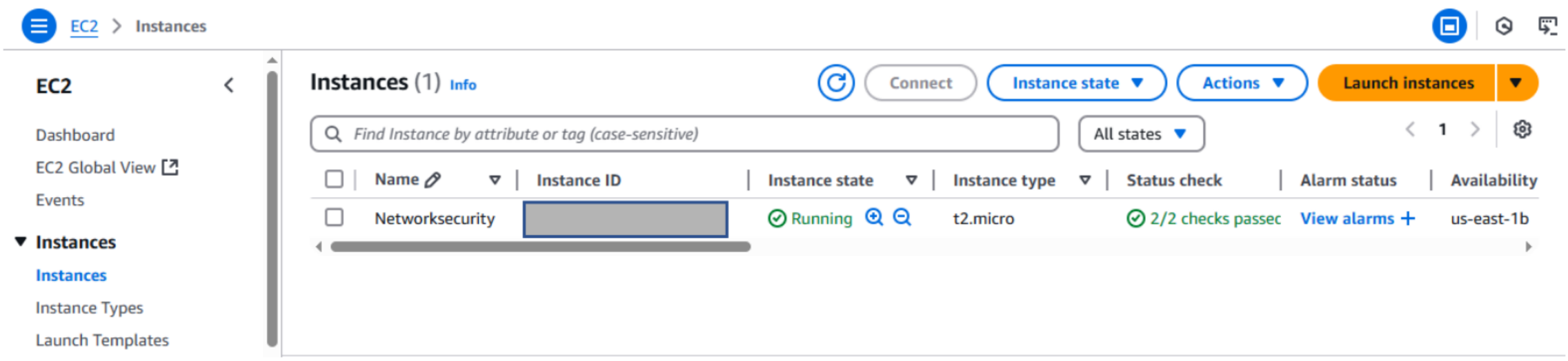
By Kunal P. Shukla

Phishing Website Detection – End-to-End ML Pipeline Project

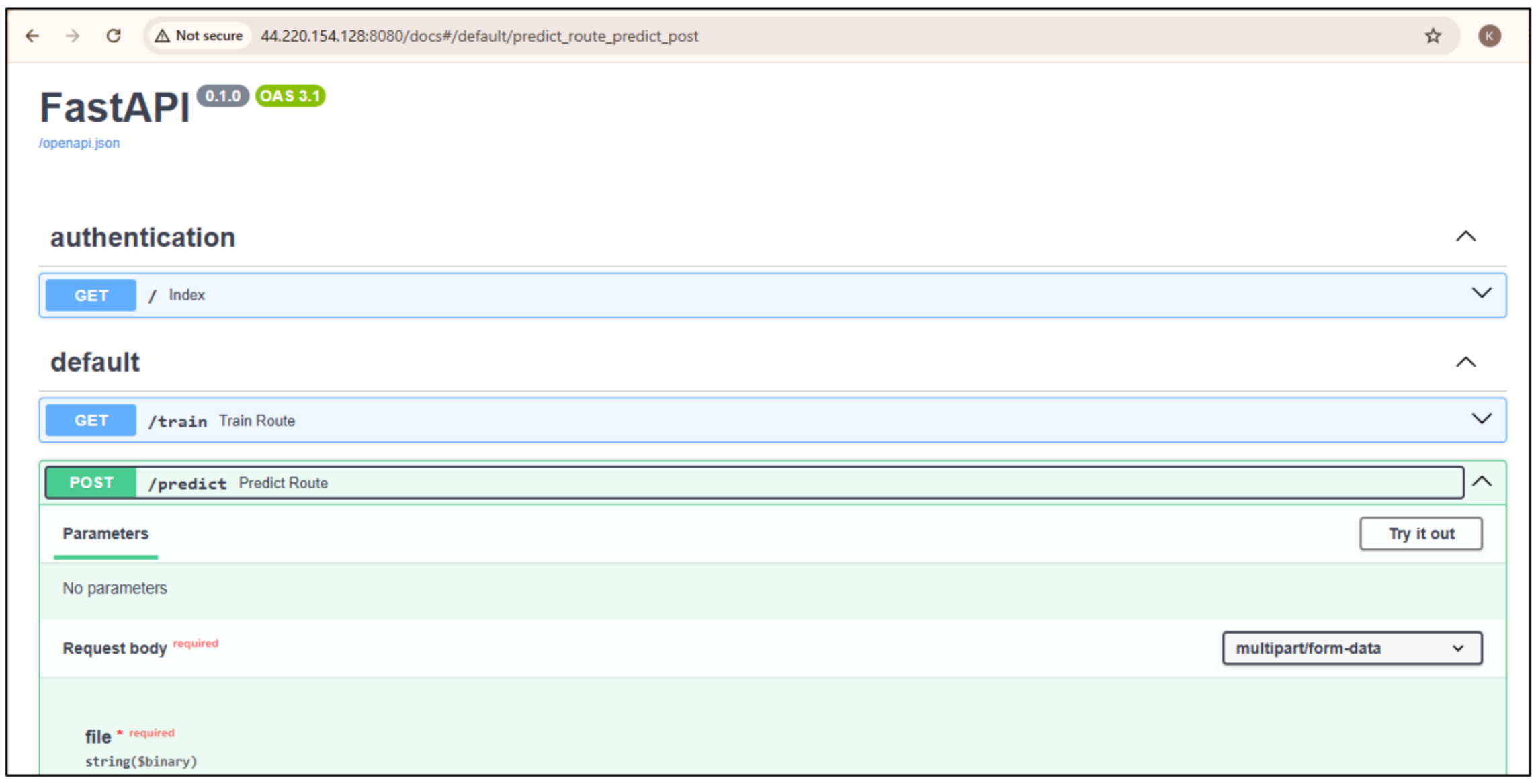
Amazon EC2

Hosting the Deployed ML Application

This EC2 instance acts as the **host machine** for running the Dockerized ML application. After the Docker image is pushed to **Amazon ECR** through GitHub Actions, this instance **pulls the image** and starts the containerized app — making it accessible via the public IP



This marks the successful deployment of the entire ML pipeline into a **cloud-accessible production environment**.



Credits:
This end-to-end machine learning deployment project structure, workflow, and guidance is inspired by **Krish Naik's Udemy course** on scalable ML pipelines and cloud deployment. Special thanks for the foundational knowledge and project framework.

By Kunal P. Shukla