Ones the juypter notebook testing is done move towards creation of data\_processing.py from src folder

Inside pathsconfig

######################## DATA PROCESSING ##########################

PROCESSED\_DIR = "artifacts/processed"

ANIMELIST\_CSV = "artifacts/raw/animelist.csv"

ANIME\_CSV = "artifacts/raw/anime.csv"

ANIMESYNOPSIS\_CSV = "artifacts/raw/anime\_with\_synopsis.csv"

X\_TRAIN\_ARRAY = os.path.join(PROCESSED\_DIR,"X\_train\_array.pkl")

X\_TEST\_ARRAY = os.path.join(PROCESSED\_DIR,"X\_test\_array.pkl")

Y\_TRAIN = os.path.join(PROCESSED\_DIR,"y\_train.pkl")

Y\_TEST = os.path.join(PROCESSED\_DIR,"y\_test.pkl")

RATING\_DF = os.path.join(PROCESSED\_DIR,"rating\_df.csv")

DF = os.path.join(PROCESSED\_DIR,"anime\_df.csv")

SYNOPSIS\_DF = os.path.join(PROCESSED\_DIR,"synopsis\_df.csv")

USER2USER\_ENCODED = "artifacts/processed/user2user\_encoded.pkl"

USER2USER\_DECODED = "artifacts/processed/user2user\_decoded.pkl"

ANIME2ANIME\_ENCODED = "artifacts/processed/anim2anime\_encoded.pkl"

ANIME2ANIME\_DECODED = "artifacts/processed/anim2anime\_decoded.pkl"

Create training pipeline code

Create folder called as pipeline inside that create a file training\_pipeline.py

from utils.common\_functions import read\_yaml

from config.paths\_config import \*

from src.data\_processing import DataProcessor

from src.model\_training import ModelTraining

if \_\_name\_\_=="\_\_main\_\_":

    data\_processor = DataProcessor(ANIMELIST\_CSV,PROCESSED\_DIR)

    data\_processor.run()

    model\_trainer = ModelTraining(PROCESSED\_DIR)

    model\_trainer.train\_model()

All the files in artificats folder needs to be pushed to another GCP Buckets, we will extract data from DVC, no need to add data ingestion in pipeline

$ python pipeline\training\_pipeline.py

Open gcp account

Create a new bucket name it (“my dvc”

Edit access give permission 🡪 add permission of service account name , and assign role, storage object viewer and admin

Now in requirements add “dvc”

In terminal

$git init

$ dvc init

Create file .gitignore -> venv, artifacts/raw/, artificates/model/,artifacts/weights/,artifacts/model\_checkpoint/MLOPS\_PROJECT\_2.egg-info

$git push origin main

$dvc add artifacts/raw 🡪 verify by going inside artifacts, raw.dvc

$dvc add artifacts/processed

Same for all

$dvc status 🡪 it should show Data and pipelines are up to date

$git add .

$git commit -m

You should see all .dvc files in github

################## Add origin of “my dvc” bucket ##############

In requirements add “dvc-gs” -> pip install -e .

Add remote origin

$ dvc remote add -d myremote <gcp bucket url> (gs://my-dvc……/) 🡪 add a back lash at last

$dvc status

$dvc push 🡪 this will push all the artifacts files into dvc bucket

Inside gcp storge it will look like 29,20,12, (hashing)

$dvc full 🡪 all files are up to date

Now delete model\_checkpoint inside artifacts and do dvc pull, it add model\_checkpoint

Create helper functions (similar users, similar annie etc.,)

Inside utils create helpers.py

import pandas as pd

import numpy as np

import joblib

from config.paths\_config import \*

############# 1. GET\_ANIME\_FRAME

def getAnimeFrame(anime,path\_df):

    df = pd.read\_csv(path\_df)

    if isinstance(anime,int):

        return df[df.anime\_id == anime]

    if isinstance(anime,str):

        return df[df.eng\_version == anime]

########## 2. GET\_SYNOPSIS

def getSynopsis(anime,path\_synopsis\_df):

    synopsis\_df = pd.read\_csv(path\_synopsis\_df)

    if isinstance(anime,int):

        return synopsis\_df[synopsis\_df.MAL\_ID == anime].sypnopsis.values[0]

    if isinstance(anime,str):

        return synopsis\_df[synopsis\_df.Name == anime].sypnopsis.values[0]

########## 3. CONTENT RECOMMENDATION

def find\_similar\_animes(name, path\_anime\_weights, path\_anime2anime\_encoded, path\_anime2anime\_decoded, path\_anime\_df, n=10, return\_dist=False, neg=False):

    # Load weights and encoded-decoded mappings

    anime\_weights = joblib.load(path\_anime\_weights)

    anime2anime\_encoded = joblib.load(path\_anime2anime\_encoded)

    anime2anime\_decoded = joblib.load(path\_anime2anime\_decoded)

    # Get the anime ID for the given name

    index = getAnimeFrame(name, path\_anime\_df).anime\_id.values[0]

    encoded\_index = anime2anime\_encoded.get(index)

    if encoded\_index is None:

        raise ValueError(f"Encoded index not found for anime ID: {index}")

    # Compute similarity distances

    weights = anime\_weights

    dists = np.dot(weights, weights[encoded\_index])  # Ensure weights[encoded\_index] is a 1D array

    sorted\_dists = np.argsort(dists)

    n = n + 1

    # Select closest or farthest based on 'neg' flag

    if neg:

        closest = sorted\_dists[:n]

    else:

        closest = sorted\_dists[-n:]

    # Return distances and closest indices if requested

    if return\_dist:

        return dists, closest

    # Build the similarity array

    SimilarityArr = []

    for close in closest:

        decoded\_id = anime2anime\_decoded.get(close)

        anime\_frame = getAnimeFrame(decoded\_id, path\_anime\_df)

        anime\_name = anime\_frame.eng\_version.values[0]

        genre = anime\_frame.Genres.values[0]

        similarity = dists[close]

        SimilarityArr.append({

            "anime\_id": decoded\_id,

            "name": anime\_name,

            "similarity": similarity,

            "genre": genre,

        })

    # Create a DataFrame with results and sort by similarity

    Frame = pd.DataFrame(SimilarityArr).sort\_values(by="similarity", ascending=False)

    return Frame[Frame.anime\_id != index].drop(['anime\_id'], axis=1)

######## 4. FIND\_SIMILAR\_USERS

def find\_similar\_users(item\_input , path\_user\_weights , path\_user2user\_encoded , path\_user2user\_decoded, n=10 , return\_dist=False,neg=False):

    try:

        user\_weights = joblib.load(path\_user\_weights)

        user2user\_encoded = joblib.load(path\_user2user\_encoded)

        user2user\_decoded = joblib.load(path\_user2user\_decoded)

        index=item\_input

        encoded\_index = user2user\_encoded.get(index)

        weights = user\_weights

        dists = np.dot(weights,weights[encoded\_index])

        sorted\_dists = np.argsort(dists)

        n=n+1

        if neg:

            closest = sorted\_dists[:n]

        else:

            closest = sorted\_dists[-n:]

        if return\_dist:

            return dists,closest

        SimilarityArr = []

        for close in closest:

            similarity = dists[close]

            if isinstance(item\_input,int):

                decoded\_id = user2user\_decoded.get(close)

                SimilarityArr.append({

                    "similar\_users" : decoded\_id,

                    "similarity" : similarity

                })

        similar\_users = pd.DataFrame(SimilarityArr).sort\_values(by="similarity",ascending=False)

        similar\_users = similar\_users[similar\_users.similar\_users != item\_input]

        return similar\_users

    except Exception as e:

        print("Error Occured",e)

################## 5. GET USER PREF

def get\_user\_preferences(user\_id , path\_rating\_df , path\_anime\_df ):

    rating\_df = pd.read\_csv(path\_rating\_df)

    df = pd.read\_csv(path\_anime\_df)

    animes\_watched\_by\_user = rating\_df[rating\_df.user\_id == user\_id]

    user\_rating\_percentile = np.percentile(animes\_watched\_by\_user.rating , 75)

    animes\_watched\_by\_user = animes\_watched\_by\_user[animes\_watched\_by\_user.rating >= user\_rating\_percentile]

    top\_animes\_user = (

        animes\_watched\_by\_user.sort\_values(by="rating" , ascending=False).anime\_id.values

    )

    anime\_df\_rows = df[df["anime\_id"].isin(top\_animes\_user)]

    anime\_df\_rows = anime\_df\_rows[["eng\_version","Genres"]]

    return anime\_df\_rows

######## 6. USER RECOMMENDATION

def get\_user\_recommendations(similar\_users , user\_pref ,path\_anime\_df , path\_synopsis\_df, path\_rating\_df, n=10):

    recommended\_animes = []

    anime\_list = []

    for user\_id in similar\_users.similar\_users.values:

        pref\_list = get\_user\_preferences(int(user\_id) , path\_rating\_df, path\_anime\_df)

        pref\_list = pref\_list[~pref\_list.eng\_version.isin(user\_pref.eng\_version.values)]

        if not pref\_list.empty:

            anime\_list.append(pref\_list.eng\_version.values)

    if anime\_list:

            anime\_list = pd.DataFrame(anime\_list)

            sorted\_list = pd.DataFrame(pd.Series(anime\_list.values.ravel()).value\_counts()).head(n)

            for i,anime\_name in enumerate(sorted\_list.index):

                n\_user\_pref = sorted\_list[sorted\_list.index == anime\_name].values[0][0]

                if isinstance(anime\_name,str):

                    frame = getAnimeFrame(anime\_name,path\_anime\_df)

                    anime\_id = frame.anime\_id.values[0]

                    genre = frame.Genres.values[0]

                    synopsis = getSynopsis(int(anime\_id),path\_synopsis\_df)

                    recommended\_animes.append({

                        "n" : n\_user\_pref,

                        "anime\_name" : anime\_name,

                        "Genres" : genre,

                        "Synopsis": synopsis

                    })

    return pd.DataFrame(recommended\_animes).head(n)

Now, create prediction\_pipeline.py inside pipeline diretory

use last function from annie.ipynb

from config.paths\_config import \*

from utils.helpers import \*

def hybrid\_recommendation(user\_id , user\_weight=0.5, content\_weight =0.5):

    ## User Recommndation

    similar\_users =find\_similar\_users(user\_id,USER\_WEIGHTS\_PATH,USER2USER\_ENCODED,USER2USER\_DECODED)

    user\_pref = get\_user\_preferences(user\_id,RATING\_DF, DF)

    user\_recommended\_animes =get\_user\_recommendations(similar\_users,user\_pref,DF, SYNOPSIS\_DF,RATING\_DF)

    user\_recommended\_anime\_list = user\_recommended\_animes["anime\_name"].tolist()

    #### Content recommendation

    content\_recommended\_animes = []

    for anime in user\_recommended\_anime\_list:

        similar\_animes = find\_similar\_animes(anime, ANIME\_WEIGHTS\_PATH, ANIME2ANIME\_ENCODED, ANIME2ANIME\_DECODED, DF)

        if similar\_animes is not None and not similar\_animes.empty:

            content\_recommended\_animes.extend(similar\_animes["name"].tolist())

        else:

            print(f"No similar anime found {anime}")

    combined\_scores = {}

    for anime in user\_recommended\_anime\_list:

        combined\_scores[anime] = combined\_scores.get(anime,0) + user\_weight

    for anime in content\_recommended\_animes:

        combined\_scores[anime] = combined\_scores.get(anime,0) + content\_weight

    sorted\_animes = sorted(combined\_scores.items() , key=lambda x:x[1] , reverse=True)

    return [anime for anime , score in sorted\_animes[:10]]

Create Application.py and templates index.html, and css

Spin the VM and install docker and Jenkins

After all jenkins docker is installed

Create Jenkinsfile in root vscode

pipeline {

    agent any

stages{

 stage('Git Checkout'){

    steps{

        script{

            echo 'cloning from github'

            checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[credentialsId: 'github-token', url: 'https://github.com/Joelfernandes30/anime-recommendationproject.git']])

        }

    }

 }

}

}

Write Docker file

FROM python:3.8-slim

# Set environment variables to prevent Python from writing .pyc files & Ensure Python output is not buffered

ENV PYTHONDONTWRITEBYTECODE=1 \

    PYTHONUNBUFFERED=1

# Install system dependencies required by TensorFlow

RUN apt-get update && apt-get install -y \

    build-essential \

    libatlas-base-dev \

    libhdf5-dev \

    libprotobuf-dev \

    protobuf-compiler \

    python3-dev \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/\*

# Set the working directory

WORKDIR /app

# Copy the application code

COPY . .

# Install dependencies from requirements.txt

RUN pip install --no-cache-dir -e .

# Train the model before running the application

RUN python pipeline/training\_pipeline.py

# Expose the port that Flask will run on

EXPOSE 5000

# Command to run the app

CMD ["python", "application.py"]

Configure gcp-key in global cred upload json file

stage("Making a virtual environment...."){

            steps{

                script{

                    echo 'Making a virtual environment...'

                    sh '''

                    python -m venv ${VENV\_DIR}

                    . ${VENV\_DIR}/bin/activate

                    pip install --upgrade pip

                    pip install -e .

                    pip install  dvc

                    '''

                }

            }

        }

stage('DVC Pull'){

            steps{

                withCredentials([file(credentialsId:'gcp-key' , variable: 'GOOGLE\_APPLICATION\_CREDENTIALS' )]){

                    script{

                        echo 'DVC Pul....'

                        sh '''

                        . ${VENV\_DIR}/bin/activate

                        dvc pull

                        '''

                    }

                }

            }

        }

Git add.

Git push

Run the pipeline

After the success go to workspace and look for files model, modelcheckpoint, processed, raw etc..,

Now,

apt-get update

apt-get install -y curl apt-transport-https ca-certificates gnupg

curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

echo "deb https://packages.cloud.google.com/apt cloud-sdk main" \

| tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

apt-get update && apt-get install -y google-cloud-sdk

gcloud –version

        environment {

        VENV\_DIR = 'venv'

        GCP\_PROJECT = 'probable-bebop-471914-q2'

        GCLOUD\_PATH = "/var/jenkins\_home/google-cloud-sdk/bin"

        KUBECTL\_AUTH\_PLUGIN = "/usr/lib/google-cloud-sdk/bin"

    }

Add this jenkins file

stage('Build and Push Image to GCR'){

            steps{

                withCredentials([file(credentialsId:'gcp-key' , variable: 'GOOGLE\_APPLICATION\_CREDENTIALS' )]){

                    script{

                        echo 'Build and Push Image to GCR'

                        sh '''

                        export PATH=$PATH:${GCLOUD\_PATH}

                        gcloud auth activate-service-account --key-file=${GOOGLE\_APPLICATION\_CREDENTIALS}

                        gcloud config set project ${GCP\_PROJECT}

                        gcloud auth configure-docker --quiet

                        docker build -t gcr.io/${GCP\_PROJECT}/ml-project:latest .

                        docker push gcr.io/${GCP\_PROJECT}/ml-project:latest

                        '''

                    }

                }

            }

        }

        stage('Deploying to Kubernetes'){

            steps{

                withCredentials([file(credentialsId:'gcp-key' , variable: 'GOOGLE\_APPLICATION\_CREDENTIALS' )]){

                    script{

                        echo 'Deploying to Kubernetes'

                        sh '''

                        export PATH=$PATH:${GCLOUD\_PATH}:${KUBECTL\_AUTH\_PLUGIN}

                        gcloud auth activate-service-account --key-file=${GOOGLE\_APPLICATION\_CREDENTIALS}

                        gcloud config set project ${GCP\_PROJECT}

                        gcloud container clusters get-credentials <name of kuberentes> --region us-central1

                        kubectl apply -f deployment.yaml

                        '''

                    }

                }

            }

        }

Enable APIS

GCR (Google cloud registry)

Artifact Registry (API)

Kubernetes engine API

Open Kubernetes engine 🡪 create cluster named ml-app

* Fleet registration
* Networking tick on Access using DNS
* Advance -> review and create

Wait for it to create

Create a file deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: ml-app

spec:

  replicas: 2

  selector:

    matchLabels:

      app: ml-app

  template:

    metadata:

      labels:

        app: ml-app

    spec:

      containers:

      - name: ml-app-container

        image: gcr.io/'probable-bebop-471914-q2'

/ml-project:latest

        ports:

        - containerPort: 5000  # Replace with the port your app listens on

---

apiVersion: v1

kind: Service

metadata:

  name: ml-app-service

spec:

  type: LoadBalancer

  selector:

    app: ml-app

  ports:

  - protocol: TCP

    port: 80

    targetPort: 5000

push code to github

build the pipeline

go to workloads in kubernetes and open it

At the end you will have your endpoint