End To End MLOPS Data Science Project Implementation with Deployment (*Wine quality Prediction)*

1. Introduction & GitHub Repository Setup
2. Project Template Creation
3. Project Setup & Requirements Installation
4. Logging, Utils & Exception Module
5. Project Workflows
6. All Components Notebook Experiment
7. All Components Modular Code Implementation
8. Training Pipeline
9. MLflow (MLOps Tool) - For Experiments tracking & Model Registration
10. DVC (MLOps Tool) - For Pipeline Tracking & implementation
11. Prediction Pipeline & User App Creation
12. Docker
13. Final CI/CD Deployment on AWS

Step 1: Introduction & GitHub Repository Setup

- Create git repository Nikhil2893/End-to-end-Machine-Learning-Project-with-MLflow
- Add **README.md** file
- Select **.gitignore Python** template
- Git repository created.
- > git clone https://github.com/Nikhil2893/End-to-end-Machine-Learning-Project-with-MLflow.git
- Enter inside folder and open VS code

Problem Statement:

Here we will predict the quality of wine on the basis of given features. We use the wine quality dataset available on Internet for free. This dataset has the fundamental features which are responsible for affecting the quality of the wine.

Dataset: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-data.zip

Step 2: Project Template creation

- Create new file "**template.py**" in 'src/mlProject' folder.
- **template.py**

```
import os
from pathlib import Path
import logging

logging.basicConfig(level=logging.INFO, format='[%(asctime)s]: %(message)s:')

project_name = "mlProject"
```

```
•
•  list_of_files = [
•      ".github/workflows/.gitkeep",
•      f"src/{project_name}/__init__.py",
•      f"src/{project_name}/components/__init__.py",
•      f"src/{project_name}/utils/__init__.py",
•      f"src/{project_name}/utils/common.py",
•      f"src/{project_name}/config/__init__.py",
•      f"src/{project_name}/config/configuration.py",
•      f"src/{project_name}/pipeline/__init__.py",
•      f"src/{project_name}/entity/__init__.py",
•      f"src/{project_name}/entity/config_entity.py",
•      f"src/{project_name}/constants/__init__.py",
•      "config/config.yaml",
•      "params.yaml",
•      "schema.yaml",
•      "main.py",
•      "app.py",
•      "Dockerfile",
•      "requirements.txt",
•      "setup.py",
•      "research/trials.ipynb",
•      "templates/index.html",
•      "test.py"
•  ]
•
•  for filepath in list_of_files:
•      filepath = Path(filepath)
•      filedir, filename = os.path.split(filepath)
•
•      if filedir !="":
•          os.makedirs(filedir, exist_ok=True)
•          logging.info(f"Creating directory; {filedir} for the file: {filename}")
•
•      if (not os.path.exists(filepath)) or (os.path.getsize(filepath) == 0):
•          with open(filepath, "w") as f:
•              pass
•              logging.info(f"Creating empty file: {filepath}")
•
•      else:
•          logging.info(f"{filename} is already exists")
```

- Run > **python template.py**
- Above project folder structure will be created.
- Commit changes in github

Step 3: Project Setup and Requirements installation

3.1 Install requirements.txt

```
pandas
mlflow==2.2.2
notebook
numpy
scikit-learn
matplotlib
python-box==6.0.2
pyYAML
tqdm
ensure==1.0.2
joblib
types-PyYAML
Flask
Flask-Cors
-e .
```

3.2 setup.py & requirements installations

Application will be created as a package

```
import setuptools

with open("README.md", "r", encoding="utf-8") as f:
    long_description = f.read()

__version__ = "0.0.0"
REPO_NAME = "End-to-end-ML-Project-with-MLflow"
AUTHOR_USER_NAME = "Nikhil2893"
SRC_REPO = "mlProject"
AUTHOR_EMAIL = "nikhilwakode2893@gmail.com"

setuptools.setup(
    name=SRC_REPO,
    version=__version__,
    author=AUTHOR_USER_NAME,
    author_email=AUTHOR_EMAIL,
    description="A small python package for ml app",
    long_description=long_description,
    long_description_content="text/markdown",
    url=f"https://github.com/{AUTHOR_USER_NAME}/{REPO_NAME}",
    project_urls={
        "Bug Tracker": f"https://github.com/{AUTHOR_USER_NAME}/{REPO_NAME}/issues",
    },
    package_dir={"": "src"},
    packages=setuptools.find_packages(where="src")
)
```

- Create virtual environment : Create a conda environment after opening the repository
  > conda create -n mlproj python=3.8 -y
  > conda activate mlproj
  > pip install -r requirements.txt
  >-e. will automatically trigger setup.py
  >commit changes in github

Step 4: Logging Module

- It is important to track the flow of code, when trying to implement code.
- Create logging module in: **src/mlProject/__init__**
- I can create another folder also **src/logger.py BUT,** I will use above.

```
import os
import sys
import logging

logging_str = "[%(asctime)s: %(levelname)s: %(module)s: %(message)s]"

log_dir = "logs"
log_filepath = os.path.join(log_dir,"running_logs.log")
os.makedirs(log_dir, exist_ok=True)

logging.basicConfig(
    level= logging.INFO,
    format= logging_str,

    handlers=[
        logging.FileHandler(log_filepath),
        logging.StreamHandler(sys.stdout)  #it will prints all log in terminal as well
    ]
)

logger = logging.getLogger("cnnClassifierLogger")
```

- Create a file **test.py**
- run > **python test.py** to check above code running

```
from src.mlProject import logger

logger.info("Welcome to our custom Log")
```

Step 5: Utils Module

- Those functions, we are using frequently in a code or project, we call this as a utility function.
- So, I can write such function/method in utils, and whenever I require, I can call that function from **src/mlProject/utils**.
- Create "**common.py"** inside above folder.
- **from box.exceptions import BoxValueError:** it is used to handle exceptions

```python
import os
from box.exceptions import BoxValueError
import yaml
from mlProject import logger
import json
import joblib
from ensure import ensure_annotations
from box import ConfigBox
from pathlib import Path
from typing import Any

@ensure_annotations
def read_yaml(path_to_yaml: Path) -> ConfigBox:
    """reads yaml file and returns

    Args:
        path_to_yaml (str): path like input

    Raises:
        ValueError: if yaml file is empty
        e: empty file

    Returns:
        ConfigBox: ConfigBox type
    """
    try:
        with open(path_to_yaml) as yaml_file:
            content = yaml.safe_load(yaml_file)
            logger.info(f"yaml file: {path_to_yaml} loaded successfully")
            return ConfigBox(content)
    except BoxValueError:
        raise ValueError("yaml file is empty")
    except Exception as e:
        raise e

@ensure_annotations
def create_directories(path_to_directories: list, verbose=True):
    """create list of directories

    Args:
        path_to_directories (list): list of path of directories
        ignore_log (bool, optional): ignore if multiple dirs is to be created.
Defaults to False.
    """
    for path in path_to_directories:
        os.makedirs(path, exist_ok=True)
        if verbose:
            logger.info(f"created directory at: {path}")
```

```python
@ensure_annotations
def save_json(path: Path, data: dict):
    """save json data

    Args:
        path (Path): path to json file
        data (dict): data to be saved in json file
    """
    with open(path, "w") as f:
        json.dump(data, f, indent=4)

    logger.info(f"json file saved at: {path}")


@ensure_annotations
def load_json(path: Path) -> ConfigBox:
    """load json files data

    Args:
        path (Path): path to json file

    Returns:
        ConfigBox: data as class attributes instead of dict
    """
    with open(path) as f:
        content = json.load(f)

    logger.info(f"json file loaded succesfully from: {path}")
    return ConfigBox(content)


@ensure_annotations
def save_bin(data: Any, path: Path):
    """save binary file

    Args:
        data (Any): data to be saved as binary
        path (Path): path to binary file
    """
    joblib.dump(value=data, filename=path)
    logger.info(f"binary file saved at: {path}")


@ensure_annotations
def load_bin(path: Path) -> Any:
    """load binary data

    Args:
        path (Path): path to binary file
```

```
    Returns:
        Any: object stored in the file
    """
    data = joblib.load(path)
    logger.info(f"binary file loaded from: {path}")
    return data


@ensure_annotations
def get_size(path: Path) -> str:
    """get size in KB

    Args:
        path (Path): path of the file

    Returns:
        str: size in KB
    """
    size_in_kb = round(os.path.getsize(path)/1024)
    return f"~ {size_in_kb} KB"
```

- Use of @**ensure_annotations,Configbox** and **box.exceptions** is described in **research/trials.ipynb** file.
- Commit changes and push to github.
- Open **README.md**

Step 6: Project Workflow

Workflow:

1. Update config.yaml
2. Update schema.yaml #Here I mention all columns in data
3. Update params.yaml
4. Update the entity - (src/entity/config_entity.py)
5. Update the configuration manager (configuration.py) in src/config
6. Update the components
7. Update the pipeline
8. Update the main.py
#updat dvc.yaml ---------- Not incorporating in this project
9. Update the app.py

- **Workflows added →>> commit github**

Before that create different files in research folder, if everything works fine in notebook files, then convert to modular coding.

01_data_ingestion_stage.ipynb
02_data_validation_stage.ipynb
03_data_transformation_stage.ipynb
04_model_trainer_stage.ipynb
05_model_evaluation_stage.ipynb

Step 7: Data Ingestion workflow

*Data is store as **winequality-data.zip***

Can be downloaded from Kaggle and store in github.

Link: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-data.zip

Follow research/01_data_ingestion_stage.ipynb to download from google drive.

```
## Donwload data from gdrive
```

- Before that follow the above workflow
- Open **config/config.yaml** and **update** it.
- Here get data from any url.

```
artifacts_root: artifacts

data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-data.zip
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion
```

- Before implementing actual components, will do experiment on notebook side. Once run fine, we will copy paste same code in modular coding format.
- Create stage 1 file in research folder: >> **research/01_data_ingestion_stage.ipynb (connect kernel to run file) and run the mentioned file.**

```
#Unzipping file
```

- **1.** Update config.yaml **completed**
- **2.** Update schema.yaml**, will use afterwards.** >>> so write **"key:val"** in it
- **3. Params.yaml >>>** we can't keep it empty>>> so write **"key:val"** in it**,** when I will create model, I will update it.(whenever I will write model trainer, I will use it)
- Go to >> **research/01_data_ingestion.ipynb >>** write code
- Whenever we create any methods or class, what would be its return type, is called **entity**.

```
from dataclasses import dataclass
from pathlib import Path
@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path

#This is a return type of configuration manager class
#This data, we are taking from config/config.yaml
```

- We have created **4. entity successfully**
- Now I need to write **5.** Configuration manager in **"research/01_data_ingestion_stage.ipynb" itself.**
- Go to **src/mlProject/constants/__init__.py**

```
from pathlib import Path

CONFIG_FILE_PATH = Path("config/config.yaml") #this will take care your path, whether you are in windows/linux machine
PARAMS_FILE_PATH = Path("params.yaml")
SCHEMA_FILE_PATH = Path("schema.yaml")
```

- I will come back to my notebook "**research/01_data_ingestion_stage.ipynb".**

- Go to **src/mlProject/utils/__init__.py** and paste below thing

```
from c.utils.common import *
```

- I will come back to my notebook "**research/01_data_ingestion_stage.ipynb"**

```
from mlProject.constants import *
from mlProject.utils import read_yaml, create_directories
```

Create configuration manager in **src/config/configuration.py**

```
# Create configuration manager class
    class ConfigurationManager:
        def __init__(
            self,
            config_filepath = CONFIG_FILE_PATH,
            params_filepath = PARAMS_FILE_PATH,
            schema_filepath = SCHEMA_FILE_PATH):
            self.config = read_yaml(config_filepath)
            self.params = read_yaml(params_filepath)
            self.schema = read_yaml(schema_filepath)
            create_directories([self.config.artifacts_root])

        def get_data_ingestion_config(self) -> DataIngestionConfig:
            config = self.config.data_ingestion

            create_directories([config.root_dir])

            data_ingestion_config = DataIngestionConfig(
                root_dir=config.root_dir,
                source_URL=config.source_URL,
                local_data_file=config.local_data_file,
                unzip_dir=config.unzip_dir
            )

            return data_ingestion_config
```

- 5.update the configuration manager in src config >>> **DONE**
- 6. Update the **components**>>> This is next task

```
import os
import urllib.request as request
import zipfile
from mlProject import logger
from mlProject.utils.common import get_size

class DataIngestion:
    def __init__(self, config: DataIngestionConfig):
        self.config = config


    def download_file(self):
        if not os.path.exists(self.config.local_data_file):
            filename, headers = request.urlretrieve(
                url = self.config.source_URL,
                filename = self.config.local_data_file
            )
            logger.info(f"{filename} download! with following info:
\n{headers}")
        else:
            logger.info(f"File already exists of size:
{get_size(Path(self.config.local_data_file))}")


    def extract_zip_file(self):
        """
        zip_file_path: str
        Extracts the zip file into the data directory
        Function returns None
        """
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path, exist_ok=True)
        with zipfile.ZipFile(self.config.local_data_file, 'r') as zip_ref:
            zip_ref.extractall(unzip_path)
```

7.Update the pipeline

```
try:
    config = ConfigurationManager()
    data_ingestion_config = config.get_data_ingestion_config()
    data_ingestion = DataIngestion(config=data_ingestion_config)
    data_ingestion.download_file()
    data_ingestion.extract_zip_file()
except Exception as e:
    raise e
```

Refer 01_data_ingestion_stage.ipynb

NOW IT IS IMPORTANT TO CONVERT ABOVE CODE AS MODULAR CODING

001*Step by step- Update the following files for *DATA INGESTION component******

- **Update config.yaml ---------- Already updated**
- **Update schema.yaml [Optional] ---------- Already updated**
- **Update params.yaml---------- Already updated**
- **4. Update src/mlProject/entity/config_entity.py**

```
from dataclasses import dataclass
from pathlib import Path

@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path
```

- **5. Update src/mlProject/config/configuration.py**

```
from mlProject.constants import *
from mlProject.utils.common import read_yaml, create_directories
from mlProject.entity.config_entity import (DataIngestionConfig)

class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH,
        schema_filepath = SCHEMA_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)
        self.schema = read_yaml(schema_filepath)

        create_directories([self.config.artifacts_root])

    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion

        create_directories([config.root_dir])
```

```
        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config
```

6. Update src/mlProject/components/data_ingestion.py

```
import os
import urllib.request as request
import zipfile
from mlProject import logger
from mlProject.utils.common import get_size
from pathlib import Path
from mlProject.entity.config_entity import (DataIngestionConfig)

class DataIngestion:
    def __init__(self, config: DataIngestionConfig):
        self.config = config


    def download_file(self):
        if not os.path.exists(self.config.local_data_file):
            filename, headers = request.urlretrieve(
                url = self.config.source_URL,
                filename = self.config.local_data_file
            )
            logger.info(f"{filename} download! with following info:
\n{headers}")
        else:
            logger.info(f"File already exists of size:
{get_size(Path(self.config.local_data_file))}")


    def extract_zip_file(self):
        """
        zip_file_path: str
        Extracts the zip file into the data directory
        Function returns None
        """
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path, exist_ok=True)
        with zipfile.ZipFile(self.config.local_data_file, 'r') as zip_ref:
            zip_ref.extractall(unzip_path)
```

➢ 7. Update src/mlProject/pipeline/stage_01_data_ingestion.py

```
➢ from mlProject.config.configuration import ConfigurationManager
➢ from mlProject.components.data_ingestion import DataIngestion
➢ from mlProject import logger
➢


➢ STAGE_NAME = "Data Ingestion stage"
➢
➢ class DataIngestionTrainingPipeline:
➢     def __init__(self):
➢         pass
➢
➢     def main(self):
➢         config = ConfigurationManager()
➢         data_ingestion_config = config.get_data_ingestion_config()
➢         data_ingestion = DataIngestion(config=data_ingestion_config)
➢         data_ingestion.download_file()
➢         data_ingestion.extract_zip_file()
➢
➢
➢ if __name__ == '__main__':
➢     try:
➢         logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
➢         obj = DataIngestionTrainingPipeline()
➢         obj.main()
➢         logger.info(f">>>>>> stage {STAGE_NAME} completed
  <<<<<<\n\nx==========x")
➢     except Exception as e:
➢         logger.exception(e)
➢         raise e
```

➢ 8. Update src/cnnClassifier/main.py

```
from mlProject import logger
from mlProject.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline

# logger.info("Welcome to our custom log")

STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

Go to project folder

Remove artifacts folder

Open terminal

>>python main.py

Add artifacts/* in .gitignore file

Remove .zip file for research folder and commit the changes in github.

002 **Step by step- Update the following files for *DATA VALIDATION component****

- Update the notebook **mlProject/research/02_data_validation_stage.ipynb**

1. Update config.yaml

```yaml
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-
data.zip
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


data_validation:
  root_dir: artifacts/data_validation
  unzip_data_dir: artifacts/data_ingestion/winequality-red.csv
  STATUS_FILE: artifacts/data_validation/status.txt
```

Follow the notebook **research/02_prepare_base_model.ipynb** for complete code:

2. Update schema.yaml

```yaml
COLUMNS:
  fixed acidity: float64
  volatile acidity: float64
  citric acid: float64
  residual sugar: float64
  chlorides: float64
  free sulfur dioxide: float64
  total sulfur dioxide: float64
  density: float64
  pH: float64
  sulphates: float64
  alcohol: float64
  quality: int64

TARGET_COLUMN:
  name: quality
```

3. Update params.yaml

➢ 4. Update src/mlProject/entity/config_entity.py

```
➢ from dataclasses import dataclass
➢ from pathlib import Path
➢
➢ @dataclass(frozen=True)
➢ class DataIngestionConfig:
➢     root_dir: Path
➢     source_URL: str
➢     local_data_file: Path
➢     unzip_dir: Path
➢
➢ @dataclass(frozen=True)
➢ class DataValidationConfig:
➢     root_dir: Path
➢     STATUS_FILE: str
➢     unzip_data_dir: Path
➢     all_schema: dict
```

➢ 5. Update src/mlProject/config/configuration.py

```
➢ from mlProject.constants import *
➢ from mlProject.utils.common import read_yaml, create_directories
➢ from mlProject.entity.config_entity import (DataIngestionConfig,
➢                                             DataValidationConfig)
➢
➢ class ConfigurationManager:
➢     def __init__(
➢         self,
➢         config_filepath = CONFIG_FILE_PATH,
➢         params_filepath = PARAMS_FILE_PATH,
➢         schema_filepath = SCHEMA_FILE_PATH):
➢
➢         self.config = read_yaml(config_filepath)
➢         self.params = read_yaml(params_filepath)
➢         self.schema = read_yaml(schema_filepath)
➢         create_directories([self.config.artifacts_root])
➢
➢     def get_data_ingestion_config(self) -> DataIngestionConfig:
➢         config = self.config.data_ingestion
➢
➢         create_directories([config.root_dir])
➢         data_ingestion_config = DataIngestionConfig(
➢             root_dir=config.root_dir,
➢             source_URL=config.source_URL,
➢             local_data_file=config.local_data_file,
➢             unzip_dir=config.unzip_dir
➢         )
➢
➢         return data_ingestion_config
```

```
    def get_data_validation_config(self) -> DataValidationConfig:
        config = self.config.data_validation
        schema = self.schema.COLUMNS

        create_directories([config.root_dir])

        data_validation_config = DataValidationConfig(
            root_dir=config.root_dir,
            STATUS_FILE=config.STATUS_FILE,
            unzip_data_dir = config.unzip_data_dir,
            all_schema=schema,
        )

        return data_validation_config
```

6. Update src/mlProject/components/data_validation.py

```
import os
from mlProject import logger
from mlProject.entity.config_entity import DataValidationConfig
import pandas as pd

class DataValiadtion:
    def __init__(self, config: DataValidationConfig):
        self.config = config

    def validate_all_columns(self)-> bool:
        try:
            validation_status = None

            data = pd.read_csv(self.config.unzip_data_dir)
            all_cols = list(data.columns)

            all_schema = self.config.all_schema.keys()

            for col in all_cols:
                if col not in all_schema:
                    validation_status = False
                    with open(self.config.STATUS_FILE, 'w') as f:
                        f.write(f"Validation status: {validation_status}")
                else:
                    validation_status = True
                    with open(self.config.STATUS_FILE, 'w') as f:
                        f.write(f"Validation status: {validation_status}")

            return validation_status

        except Exception as e:
            raise e
```

Here we have applied only one level of verification, another level of verification can also possible like checking datatype.

➢ 7. Update src/mlProject/pipeline/stage_02_data_validation.py

```
from mlProject.config.configuration import ConfigurationManager
from mlProject.components.data_validation import DataValiadtion
from mlProject import logger


STAGE_NAME = "Data Validation stage"

class DataValidationTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        data_validation_config = config.get_data_validation_config()
        data_validation = DataValiadtion(config=data_validation_config)
        data_validation.validate_all_columns()


if __name__ == '__main__':
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = DataValidationTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

➢ 8. Update src/mlProject/main.py

```
from mlProject import logger
from mlProject.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
from mlProject.pipeline.stage_02_data_validation import
DataValidationTrainingPipeline

STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
STAGE_NAME = "Data Validation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
```

```
    data_ingestion = DataValidationTrainingPipeline()
    data_ingestion.main()
    logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py >>>>Remove .zip file for research folder and commit the changes in github.

>>>will get status.txt, Validation_status : True , means data in proper format

003 **Step by step- Update the following files for *DATA TRANSFORMATION component****

Refer file **research/03_data_transformation_stage.ipynb**

- **1. Update config.yaml**

```
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-
data.zip
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


data_validation:
  root_dir: artifacts/data_validation
  unzip_data_dir: artifacts/data_ingestion/winequality-red.csv
  STATUS_FILE: artifacts/data_validation/status.txt


data_transformation:
  root_dir: artifacts/data_transformation
  data_path: artifacts/data_ingestion/winequality-red.csv
```

- **2.Update secrets.yaml [Optional] ---------- Already updated**
- **3.Update params.yaml**
-
- **4. Update src/mlProject/entity/config_entity.py**

```
from dataclasses import dataclass
from pathlib import Path
```

```
➢
➢  @dataclass(frozen=True)
➢  class DataIngestionConfig:
➢      root_dir: Path
➢      source_URL: str
➢      local_data_file: Path
➢      unzip_dir: Path
➢
➢  @dataclass(frozen=True)
➢  class DataValidationConfig:
➢      root_dir: Path
➢      STATUS_FILE: str
➢      unzip_data_dir: Path
➢      all_schema: dict
➢
➢  @dataclass(frozen=True)
➢  class DataTransformationConfig:
➢      root_dir: Path
➢      data_path: Path
```

➢ 5. Update src/mlProject/config/configuration.py

```
from mlProject.constants import *
from mlProject.utils.common import read_yaml, create_directories
from mlProject.entity.config_entity import (DataIngestionConfig,
                                            DataValidationConfig,
                                            DataTransformationConfig)

class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH,
        schema_filepath = SCHEMA_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)
        self.schema = read_yaml(schema_filepath)
        create_directories([self.config.artifacts_root])

    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion
        create_directories([config.root_dir])

        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
```

```
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config

    def get_data_validation_config(self) -> DataValidationConfig:
        config = self.config.data_validation
        schema = self.schema.COLUMNS

        create_directories([config.root_dir])

        data_validation_config = DataValidationConfig(
            root_dir=config.root_dir,
            STATUS_FILE=config.STATUS_FILE,
            unzip_data_dir = config.unzip_data_dir,
            all_schema=schema,
        )

        return data_validation_config

    def get_data_transformation_config(self) -> DataTransformationConfig:
        config = self.config.data_transformation

        create_directories([config.root_dir])

        data_transformation_config = DataTransformationConfig(
            root_dir=config.root_dir,
            data_path=config.data_path,
        )

        return data_transformation_config
```

➢ 6. Update src/mlProject/components/data_transformation.py

```
import os
from mlProject import logger
from sklearn.model_selection import train_test_split
import pandas as pd
from mlProject.entity.config_entity import DataTransformationConfig

class DataTransformation:
    def __init__(self, config: DataTransformationConfig):
        self.config = config
    ## Note: You can add different data transformation techniques such as Scaler,
PCA and all
    #You can perform all kinds of EDA in ML cycle here before passing this data to
the model
```

```
    # I am only adding train_test_spliting cz this data is already cleaned up

    def train_test_spliting(self):
        data = pd.read_csv(self.config.data_path)

        # Split the data into training and test sets. (0.75, 0.25) split.
        train, test = train_test_split(data)

        train.to_csv(os.path.join(self.config.root_dir, "train.csv"),index =
False)
        test.to_csv(os.path.join(self.config.root_dir, "test.csv"),index = False)

        logger.info("Splited data into training and test sets")
        logger.info(train.shape)
        logger.info(test.shape)

        print(train.shape)
        print(test.shape)
```

➢ 7. Update src/mlProject/pipeline/stage_03_data_transformation.py

```
from mlProject.config.configuration import ConfigurationManager
from mlProject.components.data_transformation import DataTransformation
from mlProject import logger
from pathlib import Path

STAGE_NAME = "Data Transformation stage"

class DataTransformationTrainingPipeline:
    def __init__(self):
        pass
    def main(self):
        try:
            with open(Path("artifacts/data_validation/status.txt"), "r") as f:
                status = f.read().split(" ")[-1]

            if status == "True":
                config = ConfigurationManager()
                data_transformation_config = config.get_data_transformation_config()
                data_transformation = DataTransformation(config=data_transformation_config)
                data_transformation.train_test_spliting()

            else:
                raise Exception("You data schema is not valid")

        except Exception as e:
            print(e)
```

```
if __name__ == '__main__':
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = DataTransformationTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

➢ 8. Update src/mlProject/main.py

```
from mlProject import logger
from mlProject.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
from mlProject.pipeline.stage_02_data_validation import DataValidationTrainingPipeline
from mlProject.pipeline.stage_03_data_transformation import
DataTransformationTrainingPipeline
STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Data Validation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataValidationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Data Transformation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataTransformationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py

>>>>Remove .zip file for research folder and commit the changes in github

004 **Step by step- Update the following files for *MODEL TRAINER component****

Refer : research/04_model_trainer_stage.ipynb

➢ 1. Update config.yaml

```yaml
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-
data.zip
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


data_validation:
  root_dir: artifacts/data_validation
  unzip_data_dir: artifacts/data_ingestion/winequality-red.csv
  STATUS_FILE: artifacts/data_validation/status.txt


data_transformation:
  root_dir: artifacts/data_transformation
  data_path: artifacts/data_ingestion/winequality-red.csv


model_trainer:
  root_dir: artifacts/model_trainer
  train_data_path: artifacts/data_transformation/train.csv
  test_data_path: artifacts/data_transformation/test.csv
  model_name: model.joblib
```

➢ 3. Update params.yaml

```yaml
ElasticNet:
  alpha: 0.2
  l1_ratio: 0.1
```

➢ 4. Update src/mlProject/entity/config_entity.py

```python
from dataclasses import dataclass
from pathlib import Path

@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path

@dataclass(frozen=True)
class DataValidationConfig:
    root_dir: Path
    STATUS_FILE: str
    unzip_data_dir: Path
    all_schema: dict


@dataclass(frozen=True)
class DataTransformationConfig:
    root_dir: Path
    data_path: Path


@dataclass(frozen=True)
class ModelTrainerConfig:
    root_dir: Path
    train_data_path: Path
    test_data_path: Path
    model_name: str
    alpha: float
    l1_ratio: float
    target_column: str
```

➢ 5. Update src/mlProject/config/configuration.py

```python
from mlProject.constants import *
from mlProject.utils.common import read_yaml, create_directories
from mlProject.entity.config_entity import (DataIngestionConfig,
                                            DataValidationConfig,
                                            DataTransformationConfig,
                                            ModelTrainerConfig)
```

```python
class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH,
        schema_filepath = SCHEMA_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)
        self.schema = read_yaml(schema_filepath)

        create_directories([self.config.artifacts_root])

    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion

        create_directories([config.root_dir])

        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config


    def get_data_validation_config(self) -> DataValidationConfig:
        config = self.config.data_validation
        schema = self.schema.COLUMNS

        create_directories([config.root_dir])

        data_validation_config = DataValidationConfig(
            root_dir=config.root_dir,
            STATUS_FILE=config.STATUS_FILE,
            unzip_data_dir = config.unzip_data_dir,
            all_schema=schema,
        )

        return data_validation_config

    def get_data_transformation_config(self) -> DataTransformationConfig:
        config = self.config.data_transformation

        create_directories([config.root_dir])

        data_transformation_config = DataTransformationConfig(
```

```
            root_dir=config.root_dir,
            data_path=config.data_path,
        )

        return data_transformation_config

    def get_model_trainer_config(self) -> ModelTrainerConfig:
        config = self.config.model_trainer
        params = self.params.ElasticNet
        schema = self.schema.TARGET_COLUMN

        create_directories([config.root_dir])

        model_trainer_config = ModelTrainerConfig(
            root_dir=config.root_dir,
            train_data_path = config.train_data_path,
            test_data_path = config.test_data_path,
            model_name = config.model_name,
            alpha = params.alpha,
            l1_ratio = params.l1_ratio,
            target_column = schema.name

        )

        return model_trainer_config
```

6. Update src/mlProject/components/model_trainer.py

```
import pandas as pd
import os
from mlProject import logger
from sklearn.linear_model import ElasticNet
import joblib
from mlProject.entity.config_entity import ModelTrainerConfig


class ModelTrainer:
    def __init__(self, config: ModelTrainerConfig):
        self.config = config


    def train(self):
        train_data = pd.read_csv(self.config.train_data_path)
        test_data = pd.read_csv(self.config.test_data_path)


        train_x = train_data.drop([self.config.target_column], axis=1)
        test_x = test_data.drop([self.config.target_column], axis=1)
```

```
        train_y = train_data[[self.config.target_column]]
        test_y = test_data[[self.config.target_column]]


        lr = ElasticNet(alpha=self.config.alpha, l1_ratio=self.config.l1_ratio,
random_state=42)
        lr.fit(train_x, train_y)

        joblib.dump(lr, os.path.join(self.config.root_dir,
self.config.model_name))
```

7. Update src/mlProject/pipeline/stage_04_model_trainer.py

```
from mlProject.config.configuration import ConfigurationManager
from mlProject.components.model_trainer import ModelTrainer
from mlProject import logger


STAGE_NAME = "Model Trainer stage"

class ModelTrainerTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        model_trainer_config = config.get_model_trainer_config()
        model_trainer_config = ModelTrainer(config=model_trainer_config)
        model_trainer_config.train()


if __name__ == '__main__':
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = ModelTrainerTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

7. Update main.py

```
from mlProject import logger
from mlProject.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
```

```python
from mlProject.pipeline.stage_02_data_validation import
DataValidationTrainingPipeline
from mlProject.pipeline.stage_03_data_transformation import
DataTransformationTrainingPipeline
from mlProject.pipeline.stage_04_model_trainer import ModelTrainerTrainingPipeline

STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Data Validation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataValidationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e


STAGE_NAME = "Data Transformation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataTransformationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e


STAGE_NAME = "Model Trainer stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = ModelTrainerTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

005 **Step by step- Update the following files for *MODEL EVALUATION component****

Agenda: How we can integrate MLflow with this project and will be tracking experiment and how will register model?

Refer: research/stage_05_model_evaluation_with_MLflow.ipynb

```
## MLflow

[Documentation](https://mlflow.org/docs/latest/index.html)


##### cmd
- mlflow ui

### dagshub
[dagshub](https://dagshub.com/)

MLFLOW_TRACKING_URI=https://dagshub.com/Nikhil2893/End-to-end-Machine-Learning-Project-with-MLflow.mlflow \
MLFLOW_TRACKING_USERNAME=Nikhil2893 \
MLFLOW_TRACKING_PASSWORD=ff53be44bc1a7eb176f12bbfb8cee04567631847 \
python script.py

Run this to export as env variables:

```bash

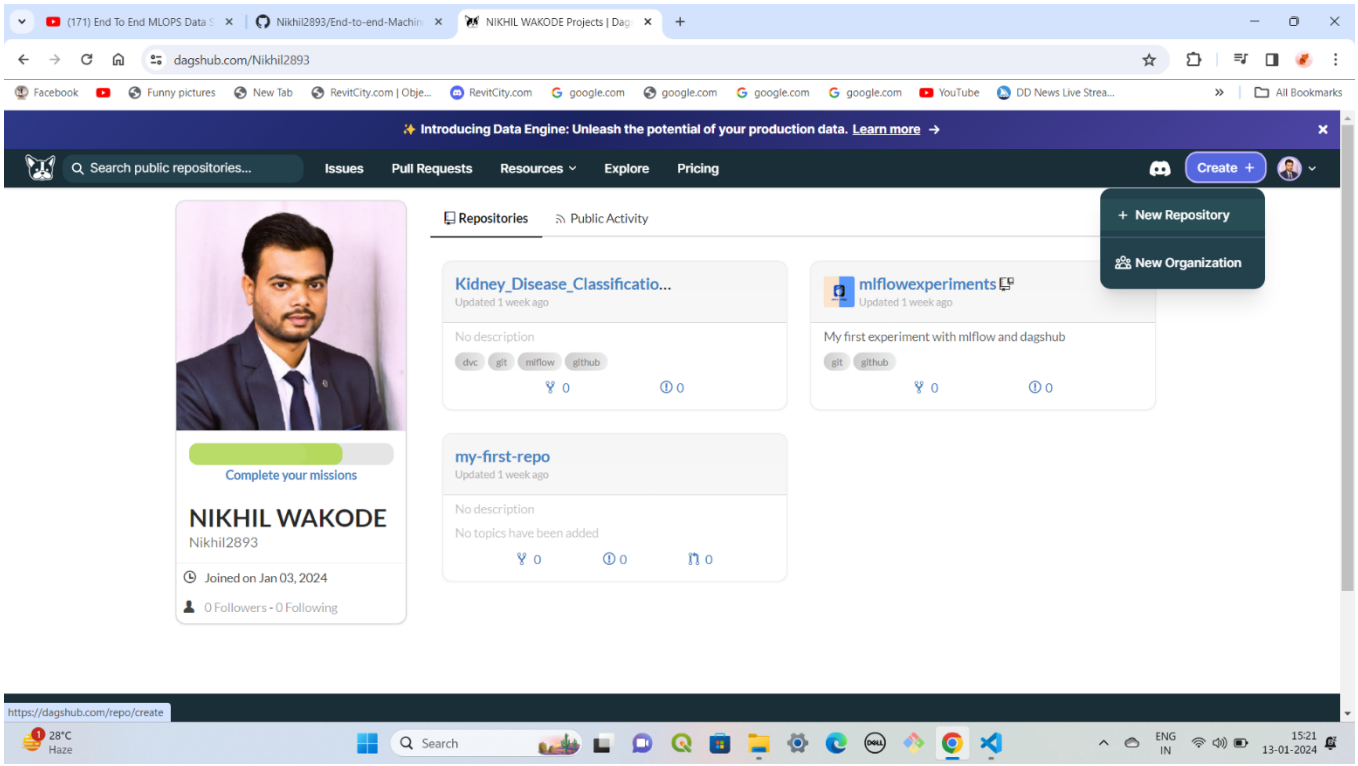
export MLFLOW_TRACKING_URI=https://dagshub.com/Nikhil2893/End-to-end-Machine-Learning-Project-with-MLflow.mlflow

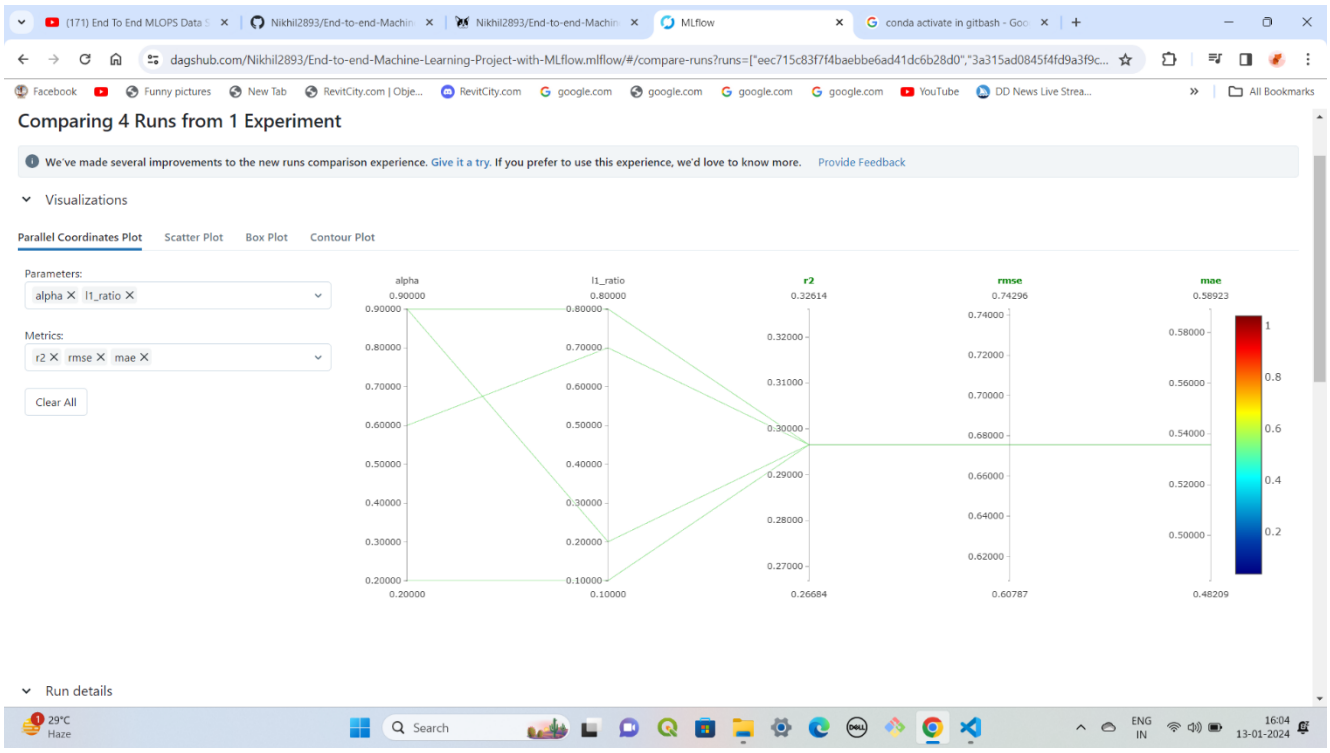
export MLFLOW_TRACKING_USERNAME=Nikhil2893

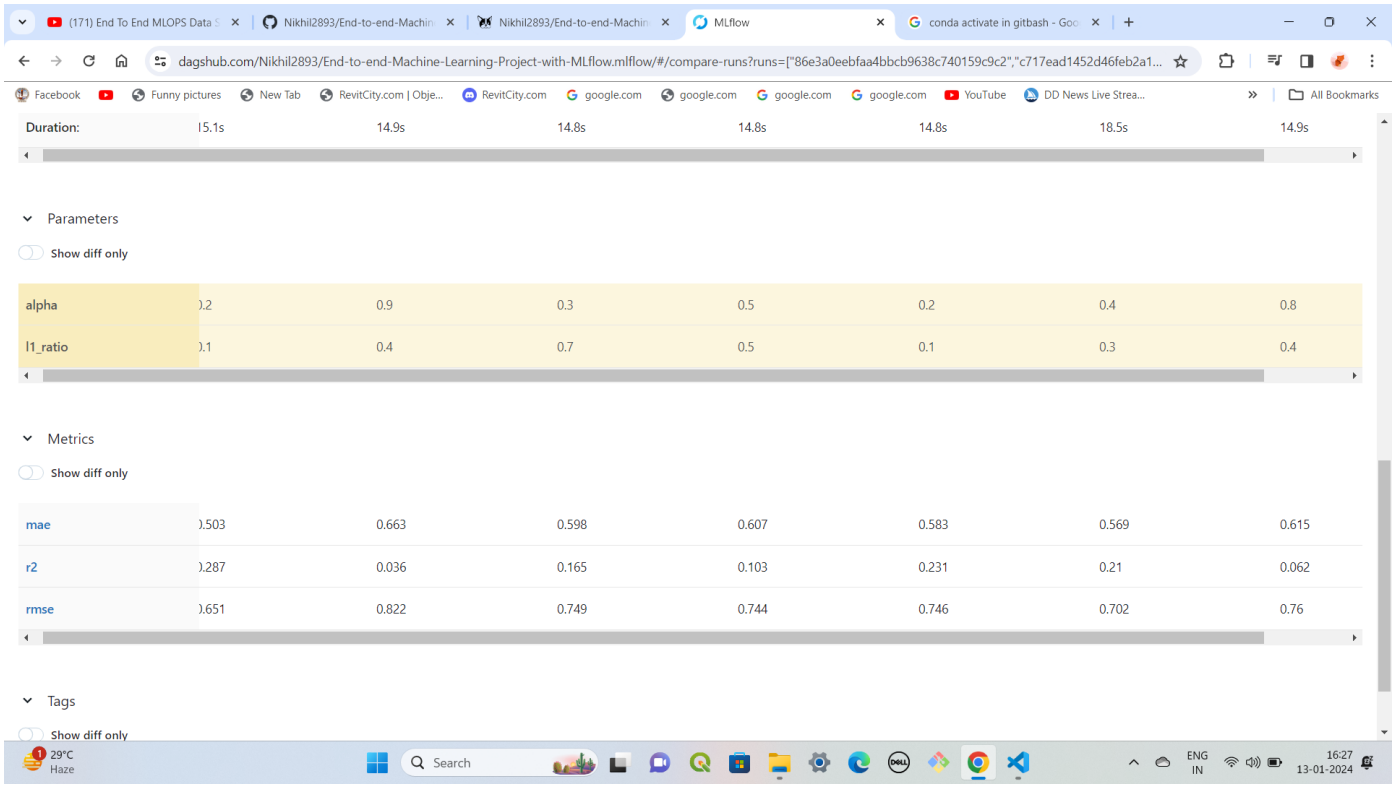
export MLFLOW_TRACKING_PASSWORD=ff53be44bc1a7eb176f12bbfb8cee04567631847

```
```

- Connect dagshub with github repository
- Using dagshub , we can launch MLFlow server
- Can track different experiments with different parameters
- Model registry also possible in MLflow, without doing it manually.


dagshub.com/Nikhil2893
Introducing Data Engine: Unleash the potential of your production data. Learn more
Search public repositories...
Issues
Pull Requests
Resources
Explore
Pricing
Create +
New Repository
New Organization
Repositories
Public Activity
Kidney_Disease_Classificatio...
Updated 1 week ago
No description
dvc git mlflow github
mlflowexperiments
Updated 1 week ago
My first experiment with mlflow and dagshub
git github
my-first-repo
Updated 1 week ago
No description
No topics have been added
Complete your missions
NIKHIL WAKODE
Nikhil2893
Joined on Jan 03, 2024
0 Followers - 0 Following
https://dagshub.com/repo/create


dagshub.com/Nikhil2893/End-to-end-Machine-Learning-Project-with-MLflow.mlflow/#/compare-runs?runs=["eec715c83f7f4baebbe6ad41dc6b28d0","3a315ad0845f4fd9a3f9c...
Comparing 4 Runs from 1 Experiment
We've made several improvements to the new runs comparison experience. Give it a try. If you prefer to use this experience, we'd love to know more. Provide Feedback
Visualizations
Parallel Coordinates Plot
Scatter Plot
Box Plot
Contour Plot
Parameters:
alpha l1_ratio
Metrics:
r2 rmse mae
Clear All
alpha
l1_ratio
r2
rmse
mae
Run details



In this after setting mlflow environment with dagshub start below steps

➢ 1. Update config.yaml

```
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://github.com/Nikhil2893/AIML_dataset/raw/main/winequality-
data.zip
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


data_validation:
  root_dir: artifacts/data_validation
  unzip_data_dir: artifacts/data_ingestion/winequality-red.csv
  STATUS_FILE: artifacts/data_validation/status.txt


data_transformation:
  root_dir: artifacts/data_transformation
  data_path: artifacts/data_ingestion/winequality-red.csv
```

```yaml
model_trainer:
  root_dir: artifacts/model_trainer
  train_data_path: artifacts/data_transformation/train.csv
  test_data_path: artifacts/data_transformation/test.csv
  model_name: model.joblib


model_evaluation:
  root_dir: artifacts/model_evaluation
  test_data_path: artifacts/data_transformation/test.csv
  model_path: artifacts/model_trainer/model.joblib
  metric_file_name: artifacts/model_evaluation/metrics.json
```

➢ 4. Update src/mlProject/entity/config_entity.py

```python
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path


@dataclass(frozen=True)
class DataValidationConfig:
    root_dir: Path
    STATUS_FILE: str
    unzip_data_dir: Path
    all_schema: dict


@dataclass(frozen=True)
class DataTransformationConfig:
    root_dir: Path
    data_path: Path


@dataclass(frozen=True)
class ModelTrainerConfig:
    root_dir: Path
    train_data_path: Path
    test_data_path: Path
    model_name: str
    alpha: float
    l1_ratio: float
```

```
    target_column: str

@dataclass(frozen=True)
class ModelEvaluationConfig:
    root_dir: Path
    test_data_path: Path
    model_path: Path
    all_params: dict
    metric_file_name: Path
    target_column: str
    mlflow_uri: str
```

➢ 5. Update src/mlProject/config/configuration.py

```
from mlProject.constants import *
from mlProject.utils.common import read_yaml, create_directories
from mlProject.entity.config_entity import (DataIngestionConfig,
                                            DataValidationConfig,
                                            DataTransformationConfig,
                                            ModelTrainerConfig,
                                            ModelEvaluationConfig)
class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH,
        schema_filepath = SCHEMA_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)
        self.schema = read_yaml(schema_filepath)

        create_directories([self.config.artifacts_root])

    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion

        create_directories([config.root_dir])

        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config
```

```python
    def get_data_validation_config(self) -> DataValidationConfig:
        config = self.config.data_validation
        schema = self.schema.COLUMNS

        create_directories([config.root_dir])

        data_validation_config = DataValidationConfig(
            root_dir=config.root_dir,
            STATUS_FILE=config.STATUS_FILE,
            unzip_data_dir = config.unzip_data_dir,
            all_schema=schema,
        )

        return data_validation_config

    def get_data_transformation_config(self) -> DataTransformationConfig:
        config = self.config.data_transformation

        create_directories([config.root_dir])

        data_transformation_config = DataTransformationConfig(
            root_dir=config.root_dir,
            data_path=config.data_path,
        )

        return data_transformation_config

    def get_model_trainer_config(self) -> ModelTrainerConfig:
        config = self.config.model_trainer
        params = self.params.ElasticNet
        schema =  self.schema.TARGET_COLUMN

        create_directories([config.root_dir])

        model_trainer_config = ModelTrainerConfig(
            root_dir=config.root_dir,
            train_data_path = config.train_data_path,
            test_data_path = config.test_data_path,
            model_name = config.model_name,
            alpha = params.alpha,
            l1_ratio = params.l1_ratio,
            target_column = schema.name

        )

        return model_trainer_config
```

```
    def get_model_evaluation_config(self) -> ModelEvaluationConfig:
        config = self.config.model_evaluation
        params = self.params.ElasticNet
        schema =  self.schema.TARGET_COLUMN

        create_directories([config.root_dir])

        model_evaluation_config = ModelEvaluationConfig(
            root_dir=config.root_dir,
            test_data_path=config.test_data_path,
            model_path = config.model_path,
            all_params=params,
            metric_file_name = config.metric_file_name,
            target_column = schema.name,
            mlflow_uri="https://dagshub.com/Nikhil2893/End-to-end-Machine-
Learning-Project-with-MLflow.mlflow",

        )    #change the uri as per requirement

        return model_evaluation_config
```

➢ 6. Update src/mlProject/components/model_evaluation.py

```
import os
import pandas as pd
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from urllib.parse import urlparse
import mlflow
import mlflow.sklearn
import numpy as np
import joblib
from mlProject.entity.config_entity import ModelEvaluationConfig
from mlProject.utils.common import save_json
from pathlib import Path



class ModelEvaluation:
    def __init__(self, config: ModelEvaluationConfig):
        self.config = config


    def eval_metrics(self,actual, pred):
        rmse = np.sqrt(mean_squared_error(actual, pred))
        mae = mean_absolute_error(actual, pred)
        r2 = r2_score(actual, pred)
        return rmse, mae, r2
```

```
    def log_into_mlflow(self):

        test_data = pd.read_csv(self.config.test_data_path)
        model = joblib.load(self.config.model_path)

        test_x = test_data.drop([self.config.target_column], axis=1)
        test_y = test_data[[self.config.target_column]]


        mlflow.set_registry_uri(self.config.mlflow_uri)
        tracking_url_type_store = urlparse(mlflow.get_tracking_uri()).scheme


        with mlflow.start_run():

            predicted_qualities = model.predict(test_x)

            (rmse, mae, r2) = self.eval_metrics(test_y, predicted_qualities)

            # Saving metrics as local
            scores = {"rmse": rmse, "mae": mae, "r2": r2}
            save_json(path=Path(self.config.metric_file_name), data=scores)

            mlflow.log_params(self.config.all_params)

            mlflow.log_metric("rmse", rmse)
            mlflow.log_metric("r2", r2)
            mlflow.log_metric("mae", mae)


            # Model registry does not work with file store
            if tracking_url_type_store != "file":

                # Register the model
                # There are other ways to use the Model Registry, which depends on
the use case,
                # please refer to the doc for more information:
                # https://mlflow.org/docs/latest/model-registry.html#api-workflow
                mlflow.sklearn.log_model(model, "model",
registered_model_name="ElasticnetModel")
            else:
                mlflow.sklearn.log_model(model, "model")
```

➢ 7. Update src/mlProject/pipeline/stage_05_model_evaluation.py

```
from mlProject.config.configuration import ConfigurationManager
from mlProject.components.model_evaluation import ModelEvaluation
from mlProject import logger
```

```
STAGE_NAME = "Model evaluation stage"

class ModelEvaluationTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        model_evaluation_config = config.get_model_evaluation_config()
        model_evaluation_config = ModelEvaluation(config=model_evaluation_config)
        model_evaluation_config.log_into_mlflow()



if __name__ == '__main__':
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = ModelEvaluationTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

- **8. Update src/mlProject/main.py**

```
from mlProject import logger
from mlProject.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
from mlProject.pipeline.stage_02_data_validation import
DataValidationTrainingPipeline
from mlProject.pipeline.stage_03_data_transformation import
DataTransformationTrainingPipeline
from mlProject.pipeline.stage_04_model_trainer import ModelTrainerTrainingPipeline
from mlProject.pipeline.stage_05_model_evaluation import
ModelEvaluationTrainingPipeline

STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

```
STAGE_NAME = "Data Validation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataValidationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e



STAGE_NAME = "Data Transformation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataTransformationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e


STAGE_NAME = "Model Trainer stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = ModelTrainerTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e



STAGE_NAME = "Model evaluation stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = ModelEvaluationTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py

>>>>Remove .zip file for research folder and commit the changes in github.

>>>> **Save best model in *model/model.h5***

➢ 9. Update dvc.yaml

Use for PIPELINE TRACKING:

Whenever I am executing main.py

1. Main.py is connected with pipeline (**src/mlProject/pipeline**)

First it run and execute one by one

- **DataIngestionTrainingPipeline()**
- **DataValidationTrainingPipeline()**
- **DataTransformationTrainingPipeline()**
- **ModelTrainerTrainingPipeline()**
- **ModelEvaluationTrainingPipeline()**

Means all components are running one by one.

It will track pipeline. Will check which pipeline is available or not? It will update it.

If any stage is already executed, it will skip that stage and move to next stage.

(data version control can be used to track both ***data*** and ***pipeline***

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py

Now Instead executing python main.py, Just >>> **dvc repro**

Dvc.yaml>>> update it

```
stages:
  data_ingestion:
    cmd: python src/cnnClassifier/pipeline/stage_01_data_ingestion.py
    deps:                                                    #dependencies
      - src/cnnClassifier/pipeline/stage_01_data_ingestion.py
      - config/config.yaml
    outs:                                                    #output
```

```
    - artifacts/data_ingestion/kidney-ct-scan-image


prepare_base_model:
  cmd: python src/cnnClassifier/pipeline/stage_02_prepare_base_model.py
  deps:
    - src/cnnClassifier/pipeline/stage_02_prepare_base_model.py
    - config/config.yaml
  params:
    - IMAGE_SIZE
    - INCLUDE_TOP
    - CLASSES
    - WEIGHTS
    - LEARNING_RATE
  outs:
    - artifacts/prepare_base_model


training:
  cmd: python src/cnnClassifier/pipeline/stage_03_model_training.py
  deps:
    - src/cnnClassifier/pipeline/stage_03_model_training.py
    - config/config.yaml
    - artifacts/data_ingestion/kidney-ct-scan-image
    - artifacts/prepare_base_model

  params:
    - IMAGE_SIZE
    - EPOCHS
    - BATCH_SIZE
    - AUGMENTATION
  outs:
    - artifacts/training/model.h5


evaluation:
  cmd: python src/cnnClassifier/pipeline/stage_04_model_evaluation.py
  deps:
    - src/cnnClassifier/pipeline/stage_04_model_evaluation.py
    - config/config.yaml
    - artifacts/data_ingestion/kidney-ct-scan-image
    - artifacts/training/model.h5
  params:
    - IMAGE_SIZE
    - BATCH_SIZE
  metrics:
  - scores.json:
      cache: false
```

```
### DVC cmd

1. dvc init
2. dvc repro
3. dvc dag
```

```
$ dvc repro
Stage 'data_ingestion' didn't change, skipping
Stage 'prepare_base_model' didn't change, skipping
Stage 'training' didn't change, skipping
Stage 'evaluation' didn't change, skipping
Data and pipelines are up to date.
(kidney)
```

>>>> commit the changes in github.

➢ 10. Prediction pipeline and app.py

Create pipeline.py in src/mlProject/pipeline/prediction.py and update it

```python
import joblib
import numpy as np
import pandas as pd
from pathlib import Path



class PredictionPipeline:
    def __init__(self):
        self.model = joblib.load(Path('artifacts/model_trainer/model.joblib'))


    def predict(self, data):
        prediction = self.model.predict(data)

        return prediction
```

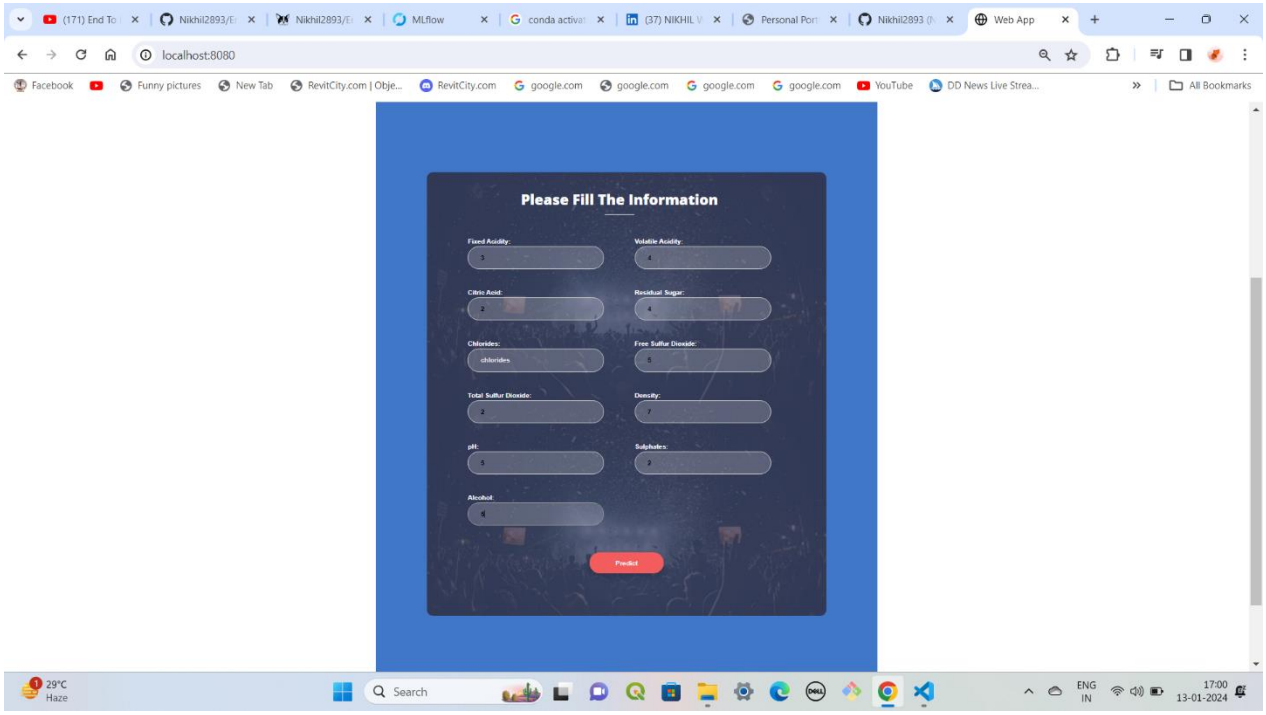
app.py

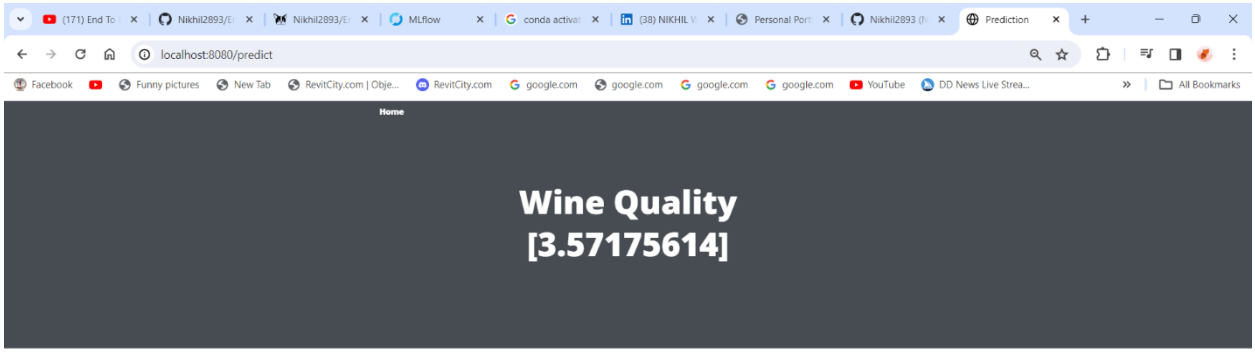
Refer templates/index.html for html code

```python
from flask import Flask, render_template, request
import os
import numpy as np
import pandas as pd
```

```
from mlProject.pipeline.prediction import PredictionPipeline


app = Flask(__name__) # initializing a flask app

@app.route('/',methods=['GET'])  # route to display the home page
def homePage():
    return render_template("index.html")


@app.route('/train',methods=['GET'])  # route to train the pipeline
def training():
    os.system("python main.py")
    return "Training Successful!"


@app.route('/predict',methods=['POST','GET']) # route to show the predictions in a
web UI
def index():
    if request.method == 'POST':
        try:
            #  reading the inputs given by the user
            fixed_acidity =float(request.form['fixed_acidity'])
            volatile_acidity =float(request.form['volatile_acidity'])
            citric_acid =float(request.form['citric_acid'])
            residual_sugar =float(request.form['residual_sugar'])
            chlorides =float(request.form['chlorides'])
            free_sulfur_dioxide =float(request.form['free_sulfur_dioxide'])
            total_sulfur_dioxide =float(request.form['total_sulfur_dioxide'])
            density =float(request.form['density'])
            pH =float(request.form['pH'])
            sulphates =float(request.form['sulphates'])
            alcohol =float(request.form['alcohol'])


            data =
[fixed_acidity,volatile_acidity,citric_acid,residual_sugar,chlorides,free_sulfur_diox
ide,total_sulfur_dioxide,density,pH,sulphates,alcohol]
            data = np.array(data).reshape(1, 11)

            obj = PredictionPipeline()
            predict = obj.predict(data)

            return render_template('results.html', prediction = str(predict))

        except Exception as e:
            print('The Exception message is: ',e)
            return 'something is wrong'

    else:
```

```
        return render_template('index.html')


if __name__ == "__main__":
    app.run(host="0.0.0.0", port = 8080, debug=True)
    # app.run(host="0.0.0.0", port = 8080)
```





About MLflow & DVC

MLflow

- Its Production Grade
- Trace all of your expriements
- Logging & taging your model

DVC

- Its very lite weight for POC only
- lite weight expriements tracker
- It can perform Orchestration (Creating Pipelines)

AWS-CICD-Deployment-with-Github-Actions

1. Login to AWS console.

2. Create IAM user for deployment

#with specific access

1. EC2 access : It is virtual machine
2. ECR: Elastic Container registry to save your docker image in aws

#Description: About the deployment

- Add main.yaml in **.github/workflows/main.yaml** ---require for CICD deploymeny

```
name: workflow

on:
  push:
    branches:
      - main
    paths-ignore:
      - 'README.md'

permissions:
  id-token: write
  contents: read
```

```yaml
jobs:
  integration:
    name: Continuous Integration
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Lint code
        run: echo "Linting repository"

      - name: Run unit tests
        run: echo "Running unit tests"

  build-and-push-ecr-image:
    name: Continuous Delivery
    needs: integration
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Install Utilities
        run: |
          sudo apt-get update
          sudo apt-get install -y jq unzip
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          ECR_REPOSITORY: ${{ secrets.ECR_REPOSITORY_NAME }}
          IMAGE_TAG: latest
        run: |
          # Build a docker container and
          # push it to ECR so that it can
          # be deployed to ECS.
          docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
          docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
```

```
            echo "::set-output
    name=image::$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG"


    Continuous-Deployment:
      needs: build-and-push-ecr-image
      runs-on: self-hosted
      steps:
        - name: Checkout
          uses: actions/checkout@v3

        - name: Configure AWS credentials
          uses: aws-actions/configure-aws-credentials@v1
          with:
            aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
            aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
            aws-region: ${{ secrets.AWS_REGION }}

        - name: Login to Amazon ECR
          id: login-ecr
          uses: aws-actions/amazon-ecr-login@v1


        - name: Pull latest images
          run: |
            docker pull ${{secrets.AWS_ECR_LOGIN_URI}}/${{
    secrets.ECR_REPOSITORY_NAME }}:latest

        # - name: Stop and remove container if running
        #   run: |
        #    docker ps -q --filter "name=mlproj" | grep -q . && docker stop
    mlproj && docker rm -fv mlproj

        - name: Run Docker Image to serve users
          run: |
            docker run -d -p 8080:8080 --name=mlproj -e 'AWS_ACCESS_KEY_ID=${{
    secrets.AWS_ACCESS_KEY_ID }}' -e 'AWS_SECRET_ACCESS_KEY=${{
    secrets.AWS_SECRET_ACCESS_KEY }}' -e 'AWS_REGION=${{ secrets.AWS_REGION
    }}'  ${{secrets.AWS_ECR_LOGIN_URI}}/${{ secrets.ECR_REPOSITORY_NAME }}:latest
        - name: Clean previous images and containers
          run: |
            docker system prune -f
```

1. **Build docker image of the source code**

```
FROM python:3.8-slim-buster

RUN apt update -y && apt install awscli -y
WORKDIR /app

COPY . /app
RUN pip install -r requirements.txt

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

2. Push your docker image to ECR

3. Launch Your EC2

4. Pull Your image from ECR in EC2

5. Lauch your docker image in EC2

#Policy:

1. AmazonEC2ContainerRegistryFullAccess

2. AmazonEC2FullAccess

3. Create ECR repo to store/save docker image

- Save the URI: 025027871758.dkr.ecr.us-east-1.amazonaws.com/kidney

4. Create EC2 machine (Ubuntu)

5. Open EC2 and Install docker in EC2 Machine:

#optinal

sudo apt-get update -y

sudo apt-get upgrade

#required

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo usermod -aG docker ubuntu

newgrp docker

6. Configure EC2 as self-hosted runner:

setting>actions>runner>new self hosted runner> choose os> then run command one by one

 # 7. Setup github secrets:

AWS_ACCESS_KEY_ID=

AWS_SECRET_ACCESS_KEY=

AWS_REGION = us-east-1

AWS_ECR_LOGIN_URI = demo>> 566373416292.dkr.ecr.ap-south-1.amazonaws.com

ECR_REPOSITORY_NAME = simple-app