End to End ML Project with MLFlow/DVC

**Setup**

1. Create Template for project:
   1. E:\Data\_Science\MLOps-Exp>**pip install cookiecutter**

Create a template : **cookiecutter -c v1** [**https://github.com/drivendata/cookiecutter-data-science**](https://github.com/drivendata/cookiecutter-data-science)

Add:

Project name : mlops-project

Repo name

Author name

Open source licence-3 #for no licence

Skip E3 and AWS

Add python-3 as interpreter

**Now move into cd mlops-project project and open VSCode**

**Note: we get all the files and folder as a template as a project by cookiecutter (delete some of the \_\_init\_\_ files from folders due to we will create ourself)**

**Under SRC folder:**

Remove \_\_init\_\_ and .py files we will write again----**(don’t delete \_\_init\_\_ file from SRC folder for SRC)**

1. **Crete a virtual env**

**Python –m venv myenv**

**Set this before activation**

**Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass**

**✅ Why is it used for virtual environments or setup scripts?**

Many Python or ML projects (especially those using cookiecutter, venv, conda, etc.) may use:

* PowerShell scripts (.ps1)
* Virtual environment activation scripts (Activate.ps1)

Windows PowerShell might **block script execution** by default with errors like:

1. **We will set git for repo**

git init

add myenv into .gitignore (so that git not add this into gitfoler)

git add .

git commit –m “cookiecutter template”

-🡪 this is how we will add temp online into git

Lets move to github account-------------create a new repo to store this info into git

Once create repo foler on git appy to **set remote access**

E:\Data\_Science\MLOps-Exp\mlops-project> **git remote add origin** [**https://github.com/LakhanGitHub/MLOps\_Project.git**](https://github.com/LakhanGitHub/MLOps_Project.git)

**Upload data from local to git**

E:\Data\_Science\MLOps-Exp\mlops-project> **git push origin master**

1. **Set Dagshub for MLFlow**

**MLflow tracking remote**

https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow

**tracking code**

import dagshub

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

import mlflow

with mlflow.start\_run():

mlflow.log\_param('parameter name', 'value')

mlflow.log\_metric('metric name', 1)

MLFlowo URL: [MLflow](https://dagshub.com/LakhanGitHub/MLOps_Project.mlflow/)

* + Install dependencies
  + Pip install dagshub
  + Pip intall mlflow, seaborn, numpy ect…….xgboost

Check dagshub working properly, lets try write a file into **notebooks foder in VSCode**

**Dagshub\_test.py**

import mlflow

import dagshub

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

with mlflow.start\_run():

  mlflow.log\_param('parameter name', 'value')

  mlflow.log\_metric('metric name', 1)

E:\Data\_Science\MLOps-Exp\mlops-project> **python \notebooks\dagsbub\_test.py**

**Experiments**

Here we will do some experiments and track in mlflow

Exp1: Random forest classifier: random test to track matrix and pramameters **exp1.py**

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, f1\_score, recall\_score, precision\_score

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import mlflow.sklearn

import mlflow

import dagshub

#initilize dagshub and set mlflow for experiment tracking

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

mlflow.set\_experiment('Experiment1')

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

#load data

data = pd.read\_csv(r'E:\Data\_Science\MLOps-Exp\water\_potability.csv')

#tain test split

train\_data, test\_data = train\_test\_split(data, test\_size=0.20, random\_state=42)

#impute missing values

def fill\_missing\_value(df):

    for col in df.columns:

        if df[col].isnull().any():

            mean\_value = df[col].median()

            df[col].fillna(mean\_value, inplace=True)

    return df

train\_processed\_data = fill\_missing\_value(train\_data)

test\_processed\_data = fill\_missing\_value(test\_data)

#randomfortestclassifier

from sklearn.ensemble import RandomForestClassifier

import pickle

#seprate train(X) and test(y)

X\_train = train\_processed\_data.drop(columns=['Potability'], axis=1)

y\_train = train\_processed\_data['Potability']

X\_test = test\_processed\_data.iloc[:,0:-1].values

y\_test = test\_processed\_data.iloc[:,-1]

n\_enstimators = 100

with mlflow.start\_run():

    clf = RandomForestClassifier(n\_estimators=n\_enstimators)

    clf.fit(X\_train,y\_train)

    #save file into pikle

    pickle.dump(clf, open('model.pkl','wb'))

    y\_pred = clf.predict(X\_test)

    acc = accuracy\_score(y\_test, y\_pred)

    pre = precision\_score(y\_test, y\_pred)

    recall = recall\_score(y\_test, y\_pred)

    f1\_score = f1\_score(y\_test, y\_pred)

    #log the metircs and parameters

    mlflow.log\_metric('accurace',acc)

    mlflow.log\_metric('precision',pre)

    mlflow.log\_metric('recall',recall)

    mlflow.log\_metric('f1\_score',f1\_score)

    mlflow.log\_param('n\_estimators', n\_enstimators)

    cm = confusion\_matrix(y\_test, y\_pred)

    plt.figure(figsize=(5,5))

    sns.heatmap(cm, annot=True)

    plt.xlabel('Predicted')

    plt.ylabel('Actual')

    plt.title('Confution Metrics')

    plt.savefig('confution\_matrix.png')

    #log artifact

    mlflow.log\_artifact('confution\_matrix.png')

    mlflow.log\_artifact(\_\_file\_\_)

    #log mode

    mlflow.sklearn.log\_model(clf,'RandomForestClassifier')

    #set tags addition metadata

    mlflow.set\_tag('Author', 'Lakhan')

    mlflow.set\_tag('model', 'Random Forest')

    #print accuracy

    print('Accuracy', acc)

    print('Precision', pre)

    print('Recall', recall)

    print('f1\_score', f1\_score)

**##adding Second Experiment-------**

**With random model with filling missing value as median()**

models = {

    'Logistic Regression': LogisticRegression(),

    'Random Forest':RandomForestClassifier,

    'SVM': SVC(),

    'Decision Tree':DecisionTreeClassifier(),

    'K-Nearst Neighbors':KNeighborsClassifier(),

    'XGboost':XGBClassifier()

}

**##Third Experimane filling missing values with mean()**

**With random model with filling missing value as median**

models = {

    'Logistic Regression': LogisticRegression(),

    'Random Forest':RandomForestClassifier,

    'SVM': SVC(),

    'Decision Tree':DecisionTreeClassifier(),

    'K-Nearst Neighbors':KNeighborsClassifier(),

    'XGboost':XGBClassifier()

}

**#adding 4th exp**

**Whrere I am using randomd forest having max accury**

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

import mlflow.sklearn

import mlflow

import dagshub

import os

# Initialize dagshub and set mlflow for experiment tracking

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

mlflow.set\_experiment('Experiment4\_2')

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

# Load dataset

data\_path = r'E:\Data\_Science\MLOps-Exp\water\_potability.csv'

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

# Train-test split

train\_data, test\_data = train\_test\_split(data, test\_size=0.20, random\_state=42)

# Function to impute missing values with mean

def fill\_missing\_value(df):

    for col in df.columns:

        if df[col].isnull().any():

            mean\_value = df[col].mean()

            df[col].fillna(mean\_value, inplace=True)

    return df

train\_processed\_data = fill\_missing\_value(train\_data)

test\_processed\_data = fill\_missing\_value(test\_data)

# Separate features and target

X\_train = train\_processed\_data.drop(columns=['Potability'], axis=1)

y\_train = train\_processed\_data['Potability']

X\_test = test\_processed\_data.drop(columns=['Potability'], axis=1)

y\_test = test\_processed\_data['Potability']

# Define model and parameter grid

rf = RandomForestClassifier(random\_state=42)

param\_dict = {

    'n\_estimators': [100, 200, 300, 400, 500],

    'max\_depth': [4, 5, 6, 10]

}

# Setup RandomizedSearchCV

random\_search = RandomizedSearchCV(rf, param\_distributions=param\_dict, cv=5, n\_iter=50, n\_jobs=-1, verbose=False)

with mlflow.start\_run(run\_name='Random Forest tuning') as parent\_run:

    random\_search.fit(X\_train, y\_train)  # Fit the search

    for i in range(len(random\_search.cv\_results\_['params'])):

        with mlflow.start\_run(run\_name=f"combination\_{i+1}", nested=True) as child\_run:

            # Get parameter combination

            params = random\_search.cv\_results\_['params'][i]

            # Train model with current params

            model = RandomForestClassifier(random\_state=42, \*\*params)

            model.fit(X\_train, y\_train)

            # Predict and evaluate

            y\_pred = model.predict(X\_test)

            acc = accuracy\_score(y\_test, y\_pred)

            # Log parameters and metrics

            mlflow.log\_params(params)

            mlflow.log\_metric("mean\_cv\_score", random\_search.cv\_results\_['mean\_test\_score'][i])

            mlflow.log\_metric("test\_accuracy", acc)

            # Log the trained model

            mlflow.sklearn.log\_model(model, artifact\_path="model")

            # Log datasets (train and test) as artifacts

            train\_csv\_path = "train\_data.csv"

            test\_csv\_path = "test\_data.csv"

            train\_processed\_data.to\_csv(train\_csv\_path, index=False)

            test\_processed\_data.to\_csv(test\_csv\_path, index=False)

            mlflow.log\_artifact(train\_csv\_path, artifact\_path="datasets")

            mlflow.log\_artifact(test\_csv\_path, artifact\_path="datasets")

            # Clean up temporary files

            os.remove(train\_csv\_path)

            os.remove(test\_csv\_path)

    # Log the best overall model

    mlflow.log\_params(random\_search.best\_params\_)

    mlflow.sklearn.log\_model(random\_search.best\_estimator\_, artifact\_path="best\_model")

    print("Best parameters found:", random\_search.best\_params\_)

    print("Training, evaluation, model, and dataset logging completed successfully.")

**#Built DVC pipeline**

* + **for data versioning we add a remote file into temp folter**
  + **>dvc init**

**>dvc remote add –d myremote C:\Windows\Temp**

**Name: C:\Windows\Temp\dvc**

**Dvc is a lolder under temp**

**Loc: C:\Windows\Temp**

**(this is temp foler for temp storate on production we upload data on AWS etc)**

**##Create oure best final model and log into mlflow**

import pandas as pd

import numpy as np

import pickle

import json

from mlflow.models.signature import infer\_signature

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import mlflow.sklearn

import mlflow

import dagshub

import os

# Initialize dagshub and set mlflow for experiment tracking

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

mlflow.set\_experiment('Best\_model')

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

test\_data = pd.read\_csv('./data/processed/test\_processed.csv')

X\_test = test\_data.iloc[:,0:-1].values

y\_test = test\_data.iloc[:,-1].values

model = pickle.load(open('model.pkl','rb'))

y\_pred= model.predict(X\_test)

with mlflow.start\_run(run\_name="DVC Model") as run:

    acc = accuracy\_score(y\_test, y\_pred)

    pre = precision\_score(y\_test, y\_pred)

    recall = recall\_score(y\_test, y\_pred)

    f1 = f1\_score(y\_test, y\_pred)

    # Log parameters and metrics

    mlflow.log\_params({

    'n\_estimators': 400,

    'max\_depth': 5

    })

    mlflow.log\_metric("accuracy", acc)

    mlflow.log\_metric("precision", pre)

    mlflow.log\_metric("recall", recall)

    mlflow.log\_metric("f1\_score", f1)

    mlflow.log\_artifact(\_\_file\_\_)

   # Log the trained model

    signature\_ = infer\_signature(X\_test, model.predict(X\_test))

    mlflow.sklearn.log\_model(model, 'Best Model', signature= signature\_)

    #save run id and model info into JSON file:

    run\_info = {'run\_id': run.info.run\_id, 'model\_name':'Best\_Model'}

    report\_path = 'reports/run\_info.json'

    with open(report\_path, 'w') as file:

        json.dump(run\_info,file,indent=4)

    metrics\_dict = {

        'acc':acc,

        'precision':pre,

        'recall':recall,

        'f1 score': f1

    }

    with open('metrics.json','w') as file:

        json.dump(metrics\_dict, file, indent=4)

**Capture final model run id from Dagshub………………**

with mlflow.start\_run(run\_name="DVC Model") as run: #add this as run here

**from below code by above code snipit**

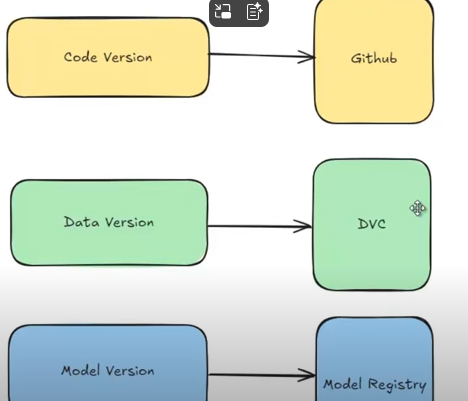
#save run id and model info into JSON file:

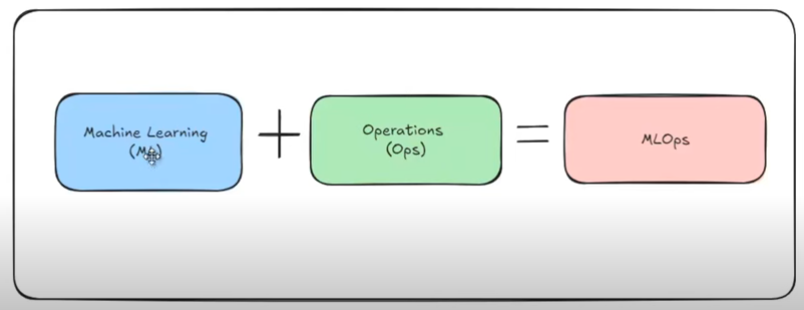
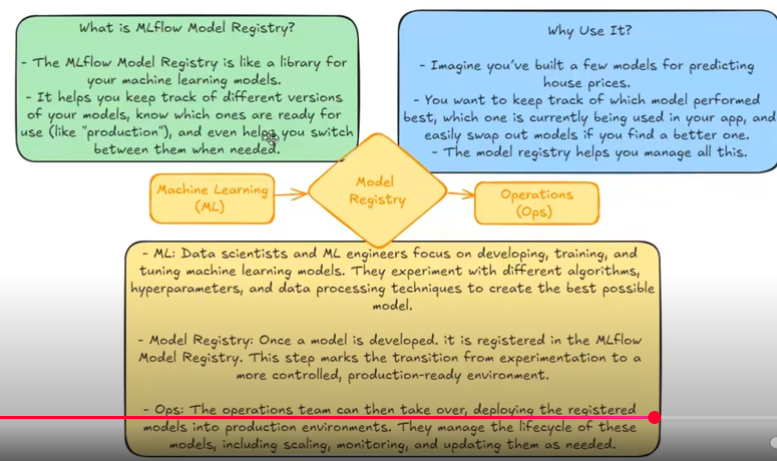
    run\_info = {'run\_id': run.info.run\_id, 'model\_name':'Best\_Model'}

    report\_path = 'reports/run\_info.json'

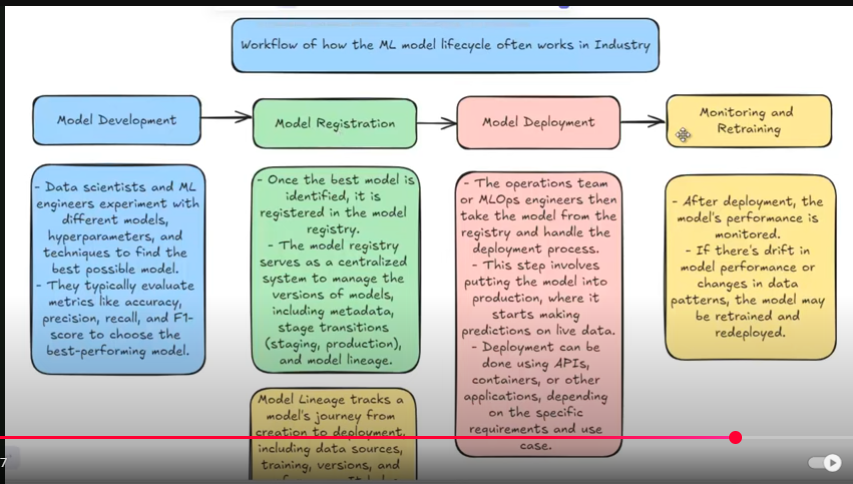
    with open(report\_path, 'w') as file:

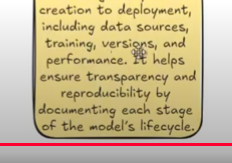
        json.dump(run\_info,file,indent=4)

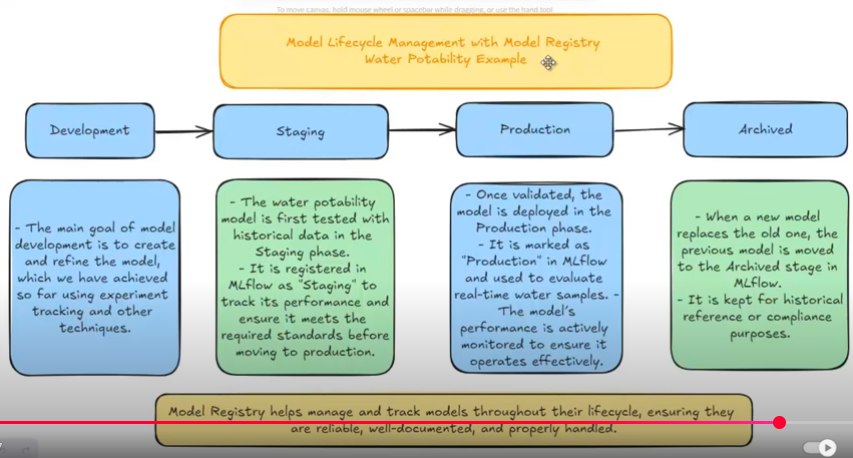
**By this be store model as version, this help to maintain model version at each run similar to git and dvc version control**



**Model registry is the mid point of ml and ops**







**##now regisger model with staging**

**Create model\_reg.py under model foler**

import json

import mlflow

import dagshub

from mlflow import MlflowClient

# Initialize dagshub and set mlflow for experiment tracking

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

mlflow.set\_experiment('Final\_Model')

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

#load the run id and model name from saved json file

report\_path = 'reports/run\_info.json'

with open(report\_path, 'r') as file:

    run\_info = json.load(file)

run\_id = run\_info['run\_id']

model\_name = run\_info['model\_name']

#create mlflow client

client = MlflowClient()

#create the model uri

model\_uri = f"runs:/{run\_id}/artifacts/{model\_name}"

#register model

reg = mlflow.register\_model(model\_uri,model\_name)

#get the model version

model\_version = reg.version

#transition the model into staging

new\_stage = 'Staging'

client.transition\_model\_version\_stage(

    name = model\_name,

    version=model\_version,

    stage = new\_stage,

    archive\_existing\_versions=True

)

print(f"model {model\_name} versioni {model\_version} transitioned to {new\_stage} stage.")

**here we have added stage as verion-1**

**now add this flow into dvc pipeline so that this staging will change automaticly once we have another version**

under **dvc.yaml** add below code to the bottom

 model\_registration:

    cmd: python src/model/model\_reg.py

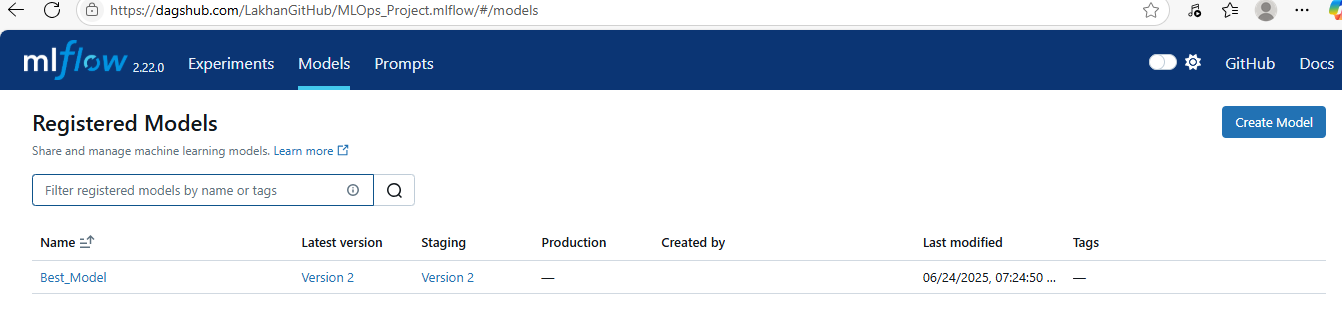
    deps:

    - reports/run\_info.json

    - src/model/model\_reg.py

This will register our model with pipeline

We can see this will add new version as we run pipeline automaticly



#Downlaod production version from mlflow and can do prediction

Create a file prediction.py outside near dvc.ymal

import json

import mlflow

import dagshub

from mlflow import MlflowClient

import pandas as pd

# Initialize dagshub and set mlflow for experiment tracking

dagshub.init(repo\_owner='LakhanGitHub', repo\_name='MLOps\_Project', mlflow=True)

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

#load the run id and model name from saved json file

model\_name = 'Best\_Model' #registered model

try:

    client = mlflow.tracking.MlflowClient()

    versions = client.get\_latest\_versions(model\_name, stages=['Staging'])

    if versions:

        latest\_version = versions[0].version

        run\_id = versions[0].run\_id #fatching the run id and latest version

        print(f'latest verion in staging :{latest\_version}, run id {run\_id}')

        logged\_model = f'runs:/{run\_id}/Best Model'  # artifact name "Best Model" 4f9f80232/artifacts/Best Model/model.pkl

        print(f'logged model :{logged\_model}')

        #load the model using the loggged\_model variable

        loaded\_model = mlflow.pyfunc.load\_model(logged\_model)

        print(f'model loaded from {logged\_model}')

        #input data for prediction

        data  = pd.DataFrame({

            'ph':[3.71608],

            'Hardness':[204.89045],

            'Solids':[20791.31890],

            'Chloramines':[7.3000],

            'Sulfate':[381.5164],

            'Conductivity':[564.3086],

            'Organic\_carbon':[10.3700],

            'Trihalomethanes':[86.99],

            'Turbidity':[2.9634]

        })

        prediction = loaded\_model.predict(data)

        print(f'prediction {prediction}')

    else:

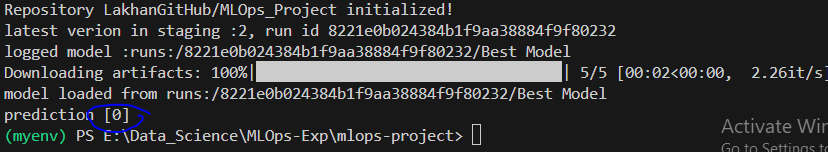
        print('no model exist in production stage')

except Exception as e:

    print(f'error fetching model :{e}')

this will doanload staging or production model like that and to prediction

here predictionis “0” water not portable:



By this production model we can create flask or streamlit App to build frontend and desing app.

**Stramlist app top of production staging:**import streamlit as st

import pandas as pd

import mlflow

from mlflow.tracking import MlflowClient

# Set MLflow tracking

mlflow.set\_tracking\_uri('https://dagshub.com/LakhanGitHub/MLOps\_Project.mlflow/')

model\_name = 'Best\_Model'

# Cache model loading to avoid re-downloading on every prediction

@st.cache\_resource

def load\_model\_from\_staging():

    try:

        client = MlflowClient()

        versions = client.get\_latest\_versions(model\_name, stages=["Staging"])

        if versions:

            run\_id = versions[0].run\_id

            logged\_model = f"runs:/{run\_id}/Best Model"  # Adjust artifact name if needed

            model = mlflow.pyfunc.load\_model(logged\_model)

            return model

        else:

            st.error("⚠️ No model found in Staging stage.")

            return None

    except Exception as e:

        st.error(f"Error loading model:\n{e}")

        return None

# Load model at app start

loaded\_model = load\_model\_from\_staging()

# Streamlit UI

st.title("💧 Water Potability Prediction App")

st.write("Provide water quality parameters below to predict if the water is \*\*safe to drink\*\*.")

# Input Form

with st.form("input\_form"):

    col1, col2, col3 = st.columns(3)

    with col1:

        ph = st.number\_input("pH Value", min\_value=0.0, max\_value=14.0, value=7.0, step=0.1)

        hardness = st.number\_input("Hardness", min\_value=0.0, value=150.0)

        solids = st.number\_input("Solids (ppm)", min\_value=0.0, value=10000.0)

    with col2:

        chloramines = st.number\_input("Chloramines (ppm)", min\_value=0.0, value=7.0)

        sulfate = st.number\_input("Sulfate (ppm)", min\_value=0.0, value=350.0)

        conductivity = st.number\_input("Conductivity (µS/cm)", min\_value=0.0, value=500.0)

    with col3:

        organic\_carbon = st.number\_input("Organic Carbon (ppm)", min\_value=0.0, value=10.0)

        trihalomethanes = st.number\_input("Trihalomethanes (µg/L)", min\_value=0.0, value=80.0)

        turbidity = st.number\_input("Turbidity (NTU)", min\_value=0.0, value=3.0)

    submit = st.form\_submit\_button("Predict")

# Prediction Logic

if submit:

    if loaded\_model is not None:

        input\_data = pd.DataFrame({

            'ph': [ph],

            'Hardness': [hardness],

            'Solids': [solids],

            'Chloramines': [chloramines],

            'Sulfate': [sulfate],

            'Conductivity': [conductivity],

            'Organic\_carbon': [organic\_carbon],

            'Trihalomethanes': [trihalomethanes],

            'Turbidity': [turbidity]

        })

        try:

            prediction = loaded\_model.predict(input\_data)[0]

            if prediction == 1:

                st.success("✅ The water is \*\*Safe to Drink\*\*.")

            else:

                st.error("The water is \*\*Not Safe to Drink\*\*.")

        except Exception as e:

            st.error(f"Error during prediction:\n{e}")