PROJECT DEVELOPMENT PHASE - SPRINT III

|  |  |
| --- | --- |
| Assignment Date | 10-11-2022 |
| Team ID | PNT2022TMIOD14930 |
| Project Name | Efficient Water Quality Analysis and Prediction  using Machine Learning |
| Maximum Marks | 8 Mark |

# Train and Develop the Model [Click here to view the Project(Hyperlink)](https://colab.research.google.com/drive/15Si0cbKycJTbFoaiq_PnMdrLHpjoPbD3)

## Data Collection:

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt import matplotlib as mpl

import matplotlib.patches as patches

from matplotlib.patches import ConnectionPatch from collections import OrderedDict

from matplotlib.gridspec import GridSpec

%matplotlib inline

df = pd.read\_csv('Final.csv') df

## Exploratory Data Analysis:

df.shape df.isnull().sum() df.info() df.describe()

df.fillna(df.mean(), inplace=True) df.isnull().sum()

df.Potability.value\_counts()

sns.countplot(df['Potability']) plt.show()

sns.distplot(df['ph']) plt.show()

df.hist(figsize=(14,14)) plt.show()

plt.figure(figsize=(13,8)) sns.heatmap(df.corr(),annot=True,cmap='terrain') plt.show()

df.boxplot(figsize=(14,7))

X = df.drop('Potability',axis=1) Y= df['Potability']

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, test\_size= 0.2, random\_state=101,shuffle=True)

## Train Decision Tree Classifier and check accuracy:

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report dt=DecisionTreeClassifier(criterion= 'gini', min\_samples\_split= 10, splitter= 'best') dt.fit(X\_train,Y\_train)

prediction=dt.predict(X\_test)

print(f"Accuracy Score = {accuracy\_score(Y\_test,prediction)\*100}") print(f"Confusion Matrix =\n {confusion\_matrix(Y\_test,prediction)}") print(f"Classification Report =\n {classification\_report(Y\_test,prediction)}")

df.head

res = dt.predict([[7.408985467,0.57139761,40,6.505923139,311.4526625,504.1459941, 11.53214401,81.10693773,3.772420928,0.0,100,0.0,16.5,0.0,11.24]])[0]

res

## Apply Hyper Parameter Tuning:

from sklearn.model\_selection import RepeatedStratifiedKFold from sklearn.model\_selection import GridSearchCV

# define models and parameters model = DecisionTreeClassifier() criterion = ["gini", "entropy"] splitter = ["best", "random"]

min\_samples\_split = [2,4,6,8,10,12,14]

# define grid search

grid = dict(splitter=splitter, criterion=criterion, min\_samples\_split=min\_samples\_split)

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1) grid\_search\_dt = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1, cv=cv,

scoring='accuracy',error\_score=0) grid\_search\_dt.fit(X\_train, Y\_train)

print(f"Best: {grid\_search\_dt.best\_score\_:.3f} using

{grid\_search\_dt.best\_params\_}")

means = grid\_search\_dt.cv\_results\_['mean\_test\_score'] stds = grid\_search\_dt.cv\_results\_['std\_test\_score'] params = grid\_search\_dt.cv\_results\_['params']

for mean, stdev, param in zip(means, stds, params): print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

print("Training Score:",grid\_search\_dt.score(X\_train, Y\_train)\*100) print("Testing Score:", grid\_search\_dt.score(X\_test, Y\_test)\*100)

# Modelling:

df.head(20) df.tail(5)

df[‘Potability’].value\_counts().to\_frame()

df\_filtered = df[df['Turbidity'].isin(["1,2,3,4,5,6,7,8,9,10"])] print(df\_filtered.head(15))

print(df\_filtered.shape)

# Model Evaluation

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_absolute\_error from sklearn.metrics import mean\_squared\_error print('R Squared=',r2\_score(X\_train,Y\_test)) print('MAE=',mean\_absolute\_error(X\_train,Y\_test)) print('MSE=',mean\_squared\_error(X\_train,Y\_tes

import joblib

joblib.dump(dt, 'classifier.pkl')

!pip install -U ibm-watson-machine-learning

from ibm\_watson\_machine\_learning import APIClient import json

import numpy as np

wml\_credentials =

{"apikey":"nFFWACn7pVNTQWlnb7pusoXVa63g0vFEq\_8Y2x2pxZSE", "url": "https://us-south.ml.cloud.ibm.com" }

wml\_client = APIClient(wml\_credentials) wml\_client.spaces.list()

SPACE\_ID = "3255cdbd-d2f9-4a9d-b816-efff2d706372" wml\_client.set.default\_space(SPACE\_ID) wml\_client.software\_specifications.list(500)

import sklearn sklearn. version

MODEL\_NAME = 'wqi' DEPLOYMENT\_NAME = 'Model'

DEMO\_MODEL = dt

# Set Python Version software\_spec\_uid =

wml\_client.software\_specifications.get\_id\_by\_name('runtime-22.1-py3.9')

# Setup model meta model\_props = {

wml\_client.repository.ModelMetaNames.NAME: MODEL\_NAME, wml\_client.repository.ModelMetaNames.TYPE: 'scikit-learn\_1.0', wml\_client.repository.ModelMetaNames.SOFTWARE\_SPEC\_UID:

software\_spec\_uid

}

## SAVE THE MODEL:

#Save model

model\_details = wml\_client.repository.store\_model( model=DEMO\_MODEL, meta\_props=model\_props, training\_data=X\_train,

training\_target=Y\_train

)

model\_details

model\_id = wml\_client.repository.get\_model\_id(model\_details) model\_id

# Set meta deployment\_props = {

wml\_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT\_NA ME,

wml\_client.deployments.ConfigurationMetaNames.ONLINE: {}

}

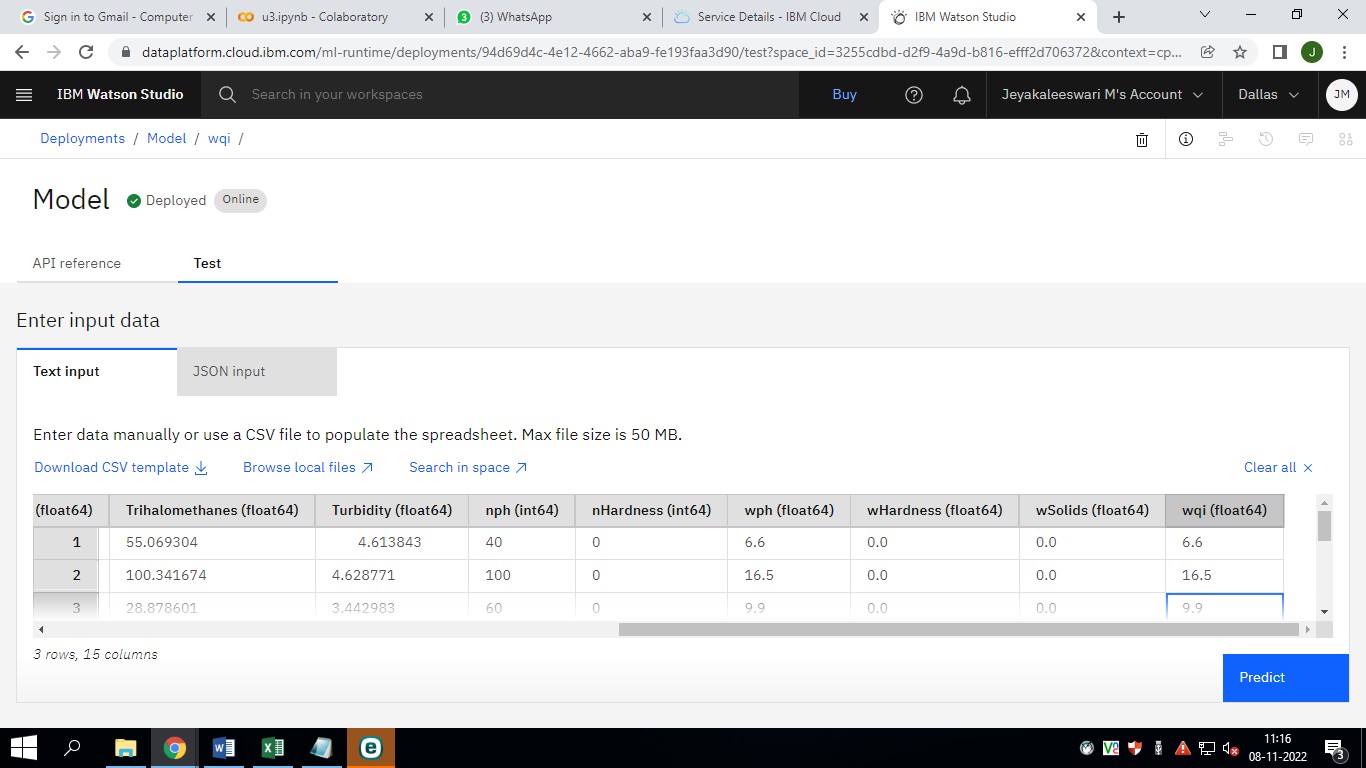
## DEPLOY:

# Deploy

deployment = wml\_client.deployments.create( artifact\_uid=model\_id, meta\_props=deployment\_props

)

# INPUT:



**OUTPUT:**

