Project Document

Team ID	PNT2022TMID21489
Project Name	Essential Water Quality Analysis and Prediction using Machine learning

PROJECT FLOW:

• User interacts with the UI (User Interface) to enter Data
• The entered data is analyzed by the model which is integrated
• Once model analyses the input the prediction is showcased on the UI

STEPS FOLLOWED:

- Data Collection.
- Data Preprocessing.
- Model Building o Training and testing the model
- Application Building
- Train model in IBM cloud

1. **CODING & SOLUTIONING**
    a. **Feature 1: Predictive model**

```
#Splitting the data into dependent and independent variables
X= df[['year', 'DO', 'PH', 'CO','BOD','NI','Tot_col']]
df['wqi']=df['wqi'].astype('int')
Y= df[['wqi']]

X.shape

Y.shape

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import linear_model
from sklearn import metrics
import math
```

```
from sklearn.metrics import mean_squared_error

X_train, X_test, Y_train, Y_test = train_test_split( X, Y,test_size=0.2, random_state=10)

#from sklearn.preprocessing import StandardScaler
#sc_X = StandardScaler()
#X_train = sc_X.fit_transform(X_train)
#X_test = sc_X.transform(X_test)

#{Decision Tree Model}
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,Y_train)
clf_pred=clf.predict(X_test)
clf_accuracy=metrics.accuracy_score(Y_test,clf_pred)
print("1) Using Decision Tree Prediction, Accuracy is " + str(clf_accuracy))

#{K Neighbors Classifier}
knn = KNeighborsClassifier(n_neighbors=7)
knn=knn.fit(X_train,Y_train.values.ravel())
knn_pred=knn.predict(X_test)
knn_accuracy=metrics.accuracy_score(Y_test,knn_pred)
print ("2) Using K Neighbors Classifier Prediction, Accuracy is " + str(knn_accuracy))

#{using MLPClassifier}
mlpc = MLPClassifier()
mlpc.fit(X_train,Y_train.values.ravel())
mlpc_pred=mlpc.predict(X_test)
mlpc_accuracy=metrics.accuracy_score(Y_test,mlpc_pred)
print ("3) Using MLPC Classifier Prediction, Accuracy is " + str(mlpc_accuracy))

#{using MLPClassifier}
rfor = RandomForestClassifier()
rfor.fit(X_train,Y_train.values.ravel())
rfor_pred=rfor.predict(X_test)
rfor_accuracy=metrics.accuracy_score(Y_test,rfor_pred)
print ("4) Using RandomForest Classifier Prediction, Accuracy is " + str(rfor_accuracy))

#{using Linear Regression}
linreg=linear_model.LinearRegression()
linreg.fit(X_train,Y_train)
linreg_pred=rfor.predict(X_test)
linreg_accuracy=metrics.accuracy_score(Y_test,linreg_pred)
rmse = math.sqrt(mean_squared_error(Y_test,linreg_pred))
print ("5) Using Linear Regression Prediction, Accuracy is " + str(linreg_accuracy))
```

```
# Accuracy found maximum in RandomForest and Linear Regression


metrics.confusion_matrix(Y_test, rfor_pred)
print(metrics.classification_report(Y_test, rfor_pred))

# Saving the model
# loading library
import pickle
with open('WaterQuality_RFModel.pkl', 'wb') as files:
   pickle.dump(rfor, files)
```

OUTPUT:

Link for Colab :
https://colab.research.google.com/drive/1XuAKvroTOVjJUDhBld0QEW6uIxMCM422#scrollTo=t5cM9GC9W1ex

b. **Feature 2: Input from users**

```
from flask import Flask, render_template, request

app = Flask(__name__)

# interface between my server and my application wsgi
import pickle
model = pickle.load(open('WaterQuality_RFModel.pkl','rb'))

@app.route('/')#binds to an url
def helloworld():
   print("KIDDO>>")
   return render_template("index.html")

@app.route('/login', methods=['POST','GET'])#binds to an url
def login():
   print("Welcome to login")
   aa= request.form["year"]
   q= request.form["DO"]
   r= request.form["PH"]
   s= request.form["Conductivity"]
   pp = request.form["BOD"]
```

```
        qq = request.form["NI"]
        ss = request.form["Tot_col"]

        #t=request.form
        t=[[int(aa),float(q),float(r),float(s),float(pp),float(qq),float(ss)]]
        output= model.predict(t)
        print(output)
        output=output[[0]]
        if (output>=95 and output<=100):
              return render_template("index.html", y="Excellent. The WQI predicted value is " +
str(output))  # we can only concatenate string to str
        elif (output>=89 and output<=94):
             return render_template("index.html", y="Very Good. The WQI predicted value is " +
str(output))  # we can only concatenate string to str
        elif(output>=80 and output<=88):
               return render_template("index.html", y="Good. The WQI predicted value is " +
str(output))  # we can only concatenate string to str
        elif (output >=65 and output <=79):
                return render_template("index.html", y="Fair. The WQI predicted value is " +
str(output))  # we can only concatenate string to str
        elif (output >=45 and output <=64):
              return render_template("index.html", y="Marginal. The WQI predicted value  is " +
str(output))  # we can only concatenate string to str
        else:
                return render_template("index.html", y="Poor. The WQI predicted value is " +
str(output))  # we can only concatenate string to str


    @app.route('/admin')#binds to an url
    def admin():
       return "Hey Admin How are you?"

    if __name__ == '__main__' :
       app.run(debug=True)
```

OUTPUT:

After inputs are being entered,