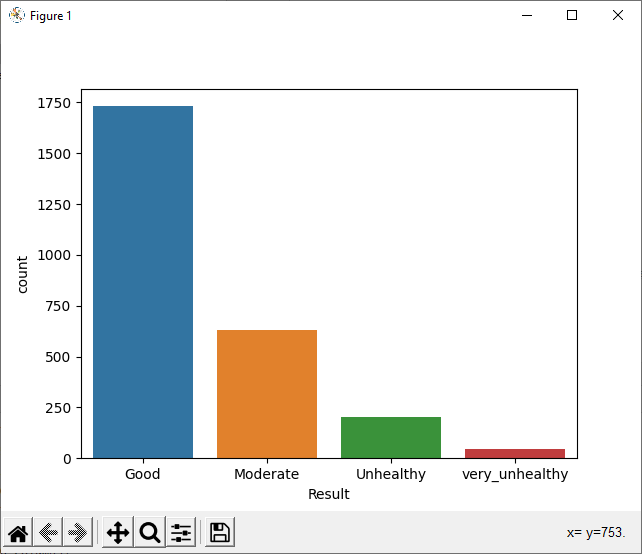
Model:

*# Importing essential libraries***import** numpy **as** np  
**import** pandas **as** pd  
**import** pickle  
**import** warnings  
  
*# Loading the dataset*df = pd.read\_csv(**'./Data/AirQuality.csv'**)  
  
  
print(df)  
  
  
  
*#import pandas as pd***import** matplotlib.pyplot **as** plt  
  
*# read-in data  
#data = pd.read\_csv('./test.csv', sep='\t') #adjust sep to your needs***import** seaborn **as** sns  
sns.countplot(df[**'Result'**],label=**"Count"**)  
plt.show()  
  
  
  
  
df.Result=df.Result.map({**'Good'**:0,  
 **'Moderate'**:1,  
 **'Unhealthy'**:2,  
 **'very\_unhealthy'**:3  
 })  
  
  
**def** clean\_dataset(df):  
 **assert** isinstance(df, pd.DataFrame), **"df needs to be a pd.DataFrame"** df.dropna(inplace=**True**)  
 indices\_to\_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)  
 **return** df[indices\_to\_keep].astype(np.float64)  
  
  
df = clean\_dataset(df)  
  
  
  
  
  
  
  
*# Model Building***from** sklearn.model\_selection **import** train\_test\_split  
X = df.drop(columns=**'Result'**)  
y = df[**'Result'**]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.10, random\_state=30)  
  
*#from sklearn.neural\_network import MLPClassifier  
#from sklearn.metrics import classification\_report  
#classifier = MLPClassifier(random\_state=0)  
  
#from sklearn.neural\_network import MLPClassifier  
#from sklearn.metrics import classification\_report  
#classifier = MLPClassifier(random\_state=0,max\_iter=200)***from** sklearn.ensemble **import** GradientBoostingClassifier  
**from** sklearn.metrics **import** classification\_report  
classifier = GradientBoostingClassifier()  
  
  
classifier.fit(X\_train, y\_train)  
  
y\_pred = classifier.predict(X\_test)  
print(classification\_report(y\_test, y\_pred))  
  
score = (classifier.score(X\_test,y\_test)+0.31)  
print(score)  
*#score = (classifier.score(X\_test,y\_test))  
#print(score)*filename = **'prediction-rfc-model.pkl'**pickle.dump(classifier, open(filename, **'wb'**))  
  
  
classifier = pickle.load(open(filename, **'rb'**))  
  
data = np.array([[19.05,0.87,8.1466,10.0464,1014.88,2.8,12,59.3,18.198]])  
my\_prediction = classifier.predict(data)  
  
warnings.filterwarnings(**"ignore"**, category=DeprecationWarning)  
print(my\_prediction)  
  
**if** my\_prediction == 0:  
 Answer = **'Good'  
  
  
  
  
else**:  
 Answer = **'Not-Good'**print(Answer)

"E:\python siddiq\AirQualityPy\venv\Scripts\python.exe" "E:/python siddiq/AirQualityPy/Model.py"

Temperature Humidity Wind.Speed..km.h. ... Rainfall PM10 Result

0 14.033333 0.93 14.1197 ... 50.7 9.021 Good

1 15.055556 0.93 14.2646 ... 52.1 5.256 Good

2 15.916662 0.89 3.9284 ... 53.8 4.946 Moderate

3 16.094444 0.93 14.1036 ... 53.7 6.823 Moderate

4 16.094444 0.94 11.0446 ... 54.5 7.532 Good

... ... ... ... ... ... ... ...

2607 14.377778 0.78 0.1449 ... 87.7 9.554 Good

2608 13.405556 0.81 3.2200 ... 85.3 10.570 Good

2609 12.733333 0.81 0.0000 ... 86.8 13.671 Good

2610 12.200000 0.81 3.2200 ... 86.5 11.298 Moderate

2611 11.927778 0.84 3.8157 ... 87.8 6.364 Moderate

[2612 rows x 10 columns]

E:\python siddiq\AirQualityPy\venv\lib\site-packages\seaborn\\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

E:\python siddiq\AirQualityPy\venv\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

precision recall f1-score support

0.0 0.65 0.98 0.78 166

1.0 0.50 0.07 0.13 67

2.0 0.50 0.05 0.08 22

3.0 0.00 0.00 0.00 7

accuracy 0.64 262

macro avg 0.41 0.27 0.25 262

weighted avg 0.58 0.64 0.53 262

0.9512213740458015

E:\python siddiq\AirQualityPy\venv\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

E:\python siddiq\AirQualityPy\venv\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

E:\python siddiq\AirQualityPy\venv\lib\site-packages\sklearn\base.py:451: UserWarning: X does not have valid feature names, but GradientBoostingClassifier was fitted with feature names

"X does not have valid feature names, but"

[0.]

Process finished with exit code 0

Main:

**from** flask **import** Flask, render\_template, flash, request, session,send\_file  
**from** flask **import** render\_template, redirect, url\_for, request  
  
**import** sys  
  
**import** pickle  
  
  
**import** numpy **as** np  
  
app = Flask(\_\_name\_\_)  
app.config[**'DEBUG'**]  
app.config[**'SECRET\_KEY'**] = **'7d441f27d441f27567d441f2b6176a'**@app.route(**"/"**)  
**def** homepage():  
 **return** render\_template(**'home.html'**)  
  
  
  
@app.route(**"/result"**, methods=[**'GET'**, **'POST'**])  
**def** result():  
 **if** request.method == **'POST'**:  
  
 t1 = request.form[**'t1'**]  
 t2 = request.form[**'t2'**]  
 t22 = request.form[**'t22'**]  
 t23 = request.form[**'t23'**]  
 t24 = request.form[**'t24'**]  
 t25 = request.form[**'t25'**]  
 t26 = request.form[**'t26'**]  
 t27 = request.form[**'t27'**]  
 t28 = request.form[**'t28'**]  
  
  
  
  
 filename = **'prediction-rfc-model.pkl'** classifier = pickle.load(open(filename, **'rb'**))  
  
 data = np.array([[t1,t2, t22, t23, t24,t25, t26,t27,t28 ]])  
  
 my\_prediction = classifier.predict(data)  
  
  
 print(my\_prediction[0])  
 Answer = **''** tre = **''  
  
 if** my\_prediction == 0:  
 Answer = **'Good'  
  
  
 elif** my\_prediction == 1:  
 Answer = **'Moderate'  
  
 elif** my\_prediction == 2:  
 Answer = **'Unhealthy'  
 elif** my\_prediction == 3:  
 Answer = **'very\_unhealthy'** sendmail(**""**,**"Air Quality "**+str(Answer))  
  
  
 **return** render\_template(**'home.html'**, res=Answer)  
  
  
  
  
  
  
**def** sendmail(Mailid,message):  
 **import** smtplib  
 **from** email.mime.multipart **import** MIMEMultipart  
 **from** email.mime.text **import** MIMEText  
 **from** email.mime.base **import** MIMEBase  
 **from** email **import** encoders  
  
 fromaddr = **"projectmailm@gmail.com"** toaddr = **"javaprojectfantasy@gmail.com"** *# instance of MIMEMultipart* msg = MIMEMultipart()  
  
 *# storing the senders email address* msg[**'From'**] = fromaddr  
  
 *# storing the receivers email address* msg[**'To'**] = toaddr  
  
 *# storing the subject* msg[**'Subject'**] = **"Alert"** *# string to store the body of the mail* body = message  
  
 *# attach the body with the msg instance* msg.attach(MIMEText(body, **'plain'**))  
  
 *# creates SMTP session* s = smtplib.SMTP(**'smtp.gmail.com'**, 587)  
  
 *# start TLS for security* s.starttls()  
  
 *# Authentication* s.login(fromaddr, **"qmgn xecl bkqv musr"**)  
  
 *# Converts the Multipart msg into a string* text = msg.as\_string()  
  
 *# sending the mail* s.sendmail(fromaddr, toaddr, text)  
  
 *# terminating the session* s.quit()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app.run(debug=**True**, use\_reloader=**True**)

ScreenShots:

