Final Project

QH20 Water Potability Analysis

Group No. 01

B. Thanushan - S14025

W. S. S. Fernando - S13990

W. M. C. B. Weerakoon - S14028

1) Abstract

The main objective of this analysis is to build a model to predict the potability of water of a resource using water quality metrics. We are also interested in finding out which factors affect the most for the potability of the water. The dataset consists of data of 3276 different water bodies collected on 10 water quality metrics including the response variable. The descriptive analysis showed that pH value, amount of sulfate, amount of chloramines, and the number of trihalomethanes affect the potability of the water than other factors. Also, some relationships between the water quality metrics have been revealed. Advanced analysis was done by fitting classification techniques such as Logistic Regression, Decision Tree Classifier, Random Forest, and XG-Boost in two different ways. The first way was carried out by changing certain numerical variables to categories and the other way was carried out by taking the numerical values directly. The models with directly used numerical values performed better. The feature selection was applied to find the most important variables for each model. The models were evaluated using the AUC score and accuracy score. The XG-Boost technique outperformed other techniques with a 69.92% accuracy score. A web application was developed for the QH2O water purifier services with the functionality to predict whether the water body is safe for consumption given the values for the given factors. This application can help the water beverages companies who are the clients for QH2O to check whether the water body can be used or not.

2) Table of Contents

3) List of Figures

4) List of Tables

5) Introduction

Water is a basic human need. Also, water is a vital requirement in the agricultural sector, electricity generation, and many industrial sectors. Although 71% of the earth is covered with water, only around 0.8% of the earth's water is potable. Hence, clean, safe water is a scarce resource that people need to preserve.

The main causes of water pollution are not natural causes but human activities. Around 80% of the worlds' wastewater is dumped back into the water streams. Also, people dump garbage and oil into water streams too. Unsafe water kills more people each year than war and all other forms of violence combined. Since this has become a serious issue in the world, various organizations urge factories to treat their wastewater before releasing them to the environment.

6) Description of the Problem

With all the polluted water resources in the world, a system to identify clean, potable water has become a necessity. We are going to build a model to predict whether the water from different water streams is safe to consume or not. Also, we are interested in finding out which substances in the water that have a major impact on water pollution. Finally, creating a website that will predict whether the water from the stream is potable or not through the built model.

7) Description of The Dataset

Dataset is obtained from the Kaggle website. It consists of data of 3276 different water bodies collected on 10 water quality metrics. Potability is the response variable.

Variable	Type	Description
pH value	Continuous	pH value of the water body.
Hardness	Continuous	The capacity of water to precipitate soap in mg/L
Solids	Continuous	Total dissolved solids in ppm
Chloramines	Continuous	Amount of Chloramines in ppm
Sulfate	Continuous	Amount of Sulfates dissolved in mg/L
Conductivity	Continuous	Electrical conductivity of water in µS/cm
Organic_carbon	Continuous	Amount of organic carbon in ppm
Trihalomethanes	Continuous	Amount of Trihalomethanes in µg/L
Turbidity	Continuous	A measure of the light-emitting property of water in NTU (Nephelometric Turbidity Units)
Potability (Response)	Categorical	Indicates if water is safe for human consumption

Table 7-1: Description of the Dataset

8) Descriptive Analysis

Analysis of pH

According to (World Health Organization, 2007), the pH of most drinking water lies within the range of 6.5–8.5. According to the figure, it is clear that the highest percentage of pH values of potable water falls within this range. Also, it is evident that the pH value is not the only factor for the water to be potable since the water of other pH levels is also potable in some percentages.

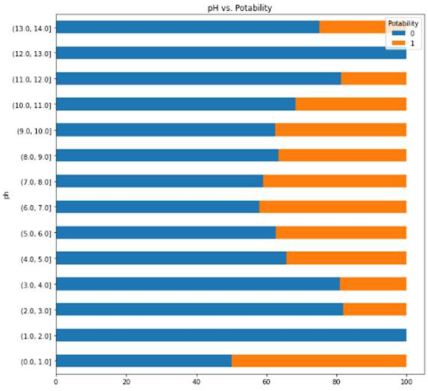

Figure 8-1: Stacked Bar Graph of pH vs. Potability

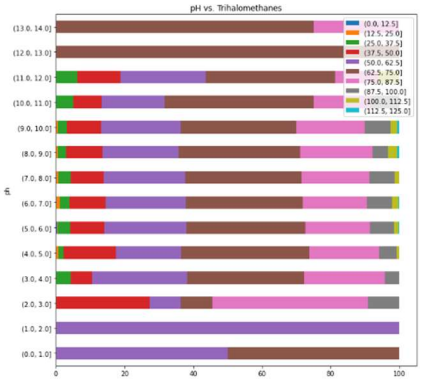

Figure 8-2: Stacked Bar Graph of pH vs. Trihalomethanes

Even a small change of pH exerts a marked effect on trihalomethanes formation. Therefore, even minute pH fluctuations may result in wide variations of trihalomethanes (Amarasooriya et al, 2018). But in our dataset, even though the pH levels change, trihalomethane levels do not fluctuate massively. This happens since a high percentage of data has pH levels 6.00 – 8.00. The data from this category overshadows the fluctuations.

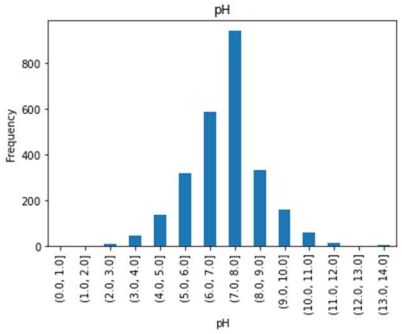

Figure 8-3: Bar Chart of pH

Analysis of Hardness

The simple definition of water hardness is the amount of dissolved calcium and magnesium in the water (Hardness of Water, 2021). Water supplies with a hardness greater than 200 mg/L are considered poor but have been tolerated by consumers; those in excess of 500 mg/L are unacceptable for most domestic purposes (Health Canada, 1995).

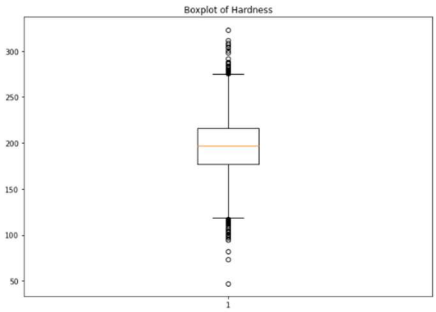

Figure 8-5: Boxplot of Hardness

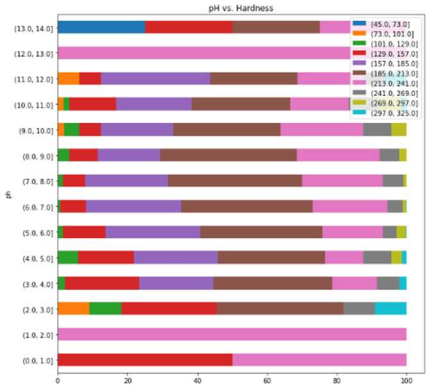

Figure 8-4: Stacked Bar Graph of pH vs. Hardness

According to the figure, the median hardness of the water is just below 200mg/l. It is stated in (Think Fish, 2013) that dissolved minerals tend to counter the effects of acids in the water, preventing the pH from dropping. Therefore, in most cases, hard water usually has a high (alkaline) pH, whilst soft water has a low (acidic) pH. But this does not reflect in the dataset. This can be due to the sulfate concentration being higher than the level of hardness in water since according to (Sulfate and water quality, n.d.), sulfates are most often related to their ability to form strong acids which change the pH.

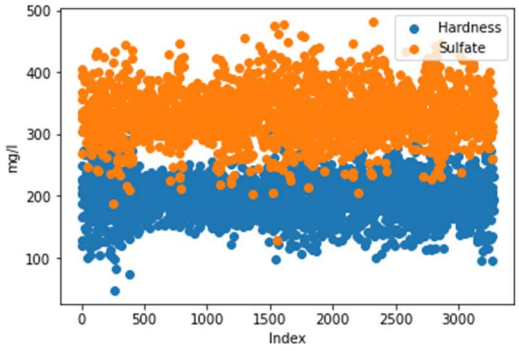

Figure 8-6: Scatterplots of Hardness and Sulfate

Analysis of Sulfate

According to (Oram, n.d.), The SMCL (Secondary Maximum Contaminant Level) for sulfate in drinking water is 250 mg/l. It can be seen that there is a high percentage of potability around 250 mg/l sulfate levels in the dataset too. There is a low water potability percentage in sulfate levels between 300 mg/l and 460mg/l. This can be due to the high percentage of solids in those levels as according to (Oram, n.d.), elevated total dissolved solids can result in your water having a bitter or salty taste which affects the potability of water.

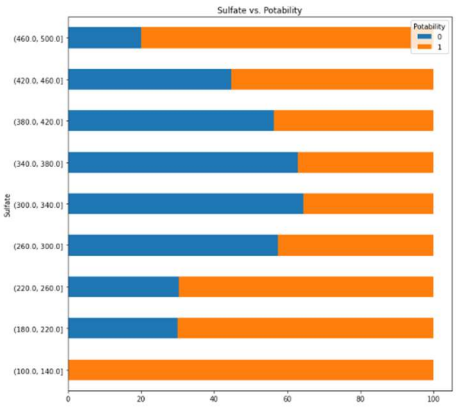

Figure 8-7: Stacked Bar Graph of Sulfate vs. Potability

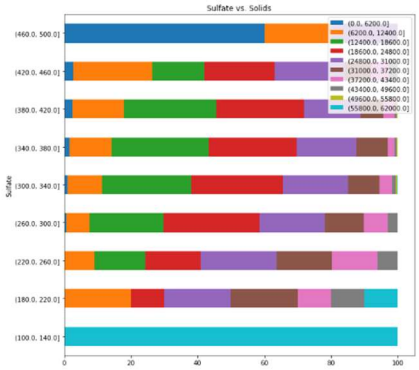

Figure 8-8: Stacked Bar Graph of Sulfate vs. Solids

Analysis of Chloramines

Chloramine levels up to 4 mg/l or 4 ppm are considered safe in drinking water according to (CDC, 2020). In the dataset, levels up to 3mg/l have a high percentage of potability. But in level 3mg/l to 4mg/l has a low percentage of potability. This is due to the presence of high levels of solid concentration in that level of chloramines since as earlier explained elevated total dissolved solids can result in your water having

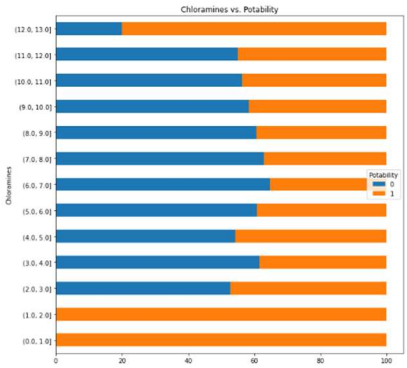

Figure 8-9 Stacked Bar Graph of Chloramines vs. Potability

a bitter or salty taste which affects the potability of water. Furthermore, (Friedler et al., 2021) stated that the regression coefficient of solids is negative when the level of solids is regressed against the level of chlorine. This implies that the solids in water affect negatively in the process of chlorine disinfection. This can be reflected in the figure since when the level of chloramine concentration is increasing, the level of solid concentration is decreasing.

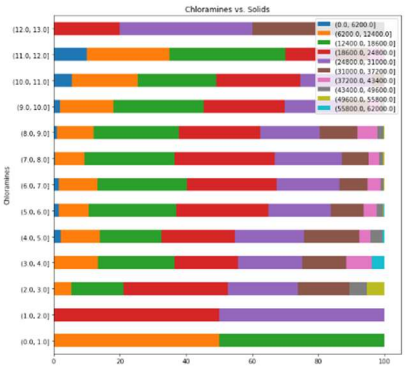

Figure 8-10: Stacked Bar Graph of Chloramines vs. Solids

Analysis of Trihalomethanes

Trihalomethanes are the result of a reaction between the chlorine used for disinfecting tap water and natural organic matter in the water (Hood, 2005). This can be seen from the figure as the level of trihalomethanes increases, the level of chloramines also increases. According to (EPA, 2016), the maximum allowable trihalomethanes level is 100 ppb. Since the level of trihalomethanes of almost all the data is below 100ppb but not all the data is potable, we can declare that other factors affect the water potability than trihalomethanes.

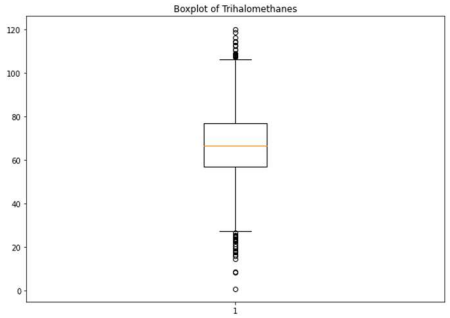

Figure 8-13: Boxplot of Trihalomethanes

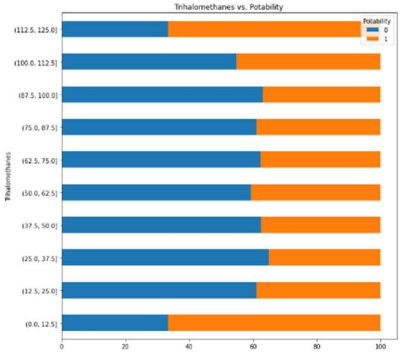

Figure 8-12: Stacked Bar Graph of Trihalomethanes vs. Potability

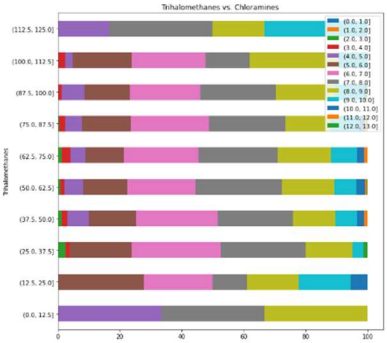

Figure 8-11: Stacked Bar Graph of Trihalomethanes vs. Chloramines

Analysis of Turbidity

Turbidity describes the cloudiness of water caused by suspended particles (WHO, 2017). Also, the WHO establishes that the turbidity of drinking water shouldn't be more than 5 NTU, and should ideally be below 1 NTU. According to the figure, the turbidity of the majority of the data is below 5NTU.

The stacked bar chart of turbidity vs. pH values show that when turbidity is increasing, the level of pH decreases. But, according to

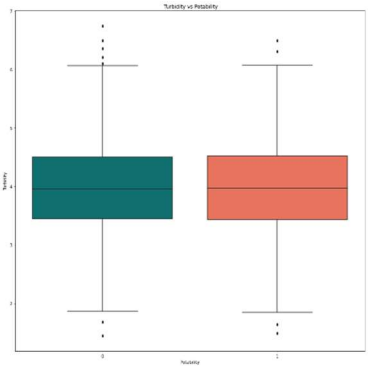

Figure 8-14: Boxplots of Turbidity vs. Potability

(Mohammed, 2015), there is no significant direct influence of pH on turbidity. But when the turbidity is increasing, the conductivity also increases. Hence, there are free ions in the water at high turbidity levels. According to (Hancock, n.d.), the coagulation process to eliminate turbidity requires positive ions. Therefore, there is a possibility of the coagulation process happening at high turbidity levels. It is stated in (Whitman.edu) that the coagulation process releases hydrogen ions into the water. Hence, the pH values of water will be decreased. Therefore, when turbidity is increasing, the pH values will decrease.

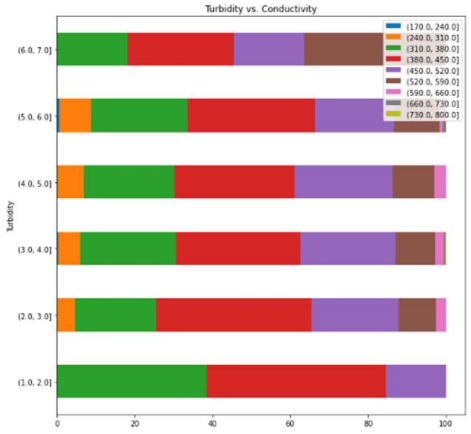

Figure 8-16: Stacked Bar Graph of Turbidity vs. Conductivity

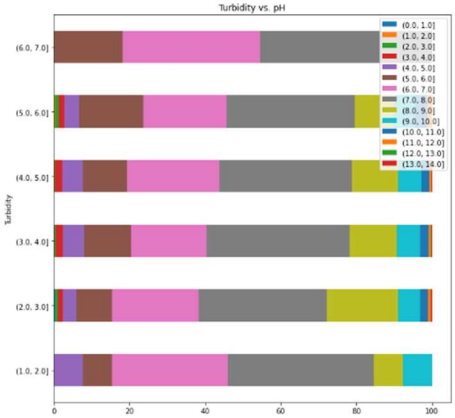

Figure 8-15: Stacked Bar Graph of Turbidity vs. pH

Correlation Plot

There are no significant correlations between variables.

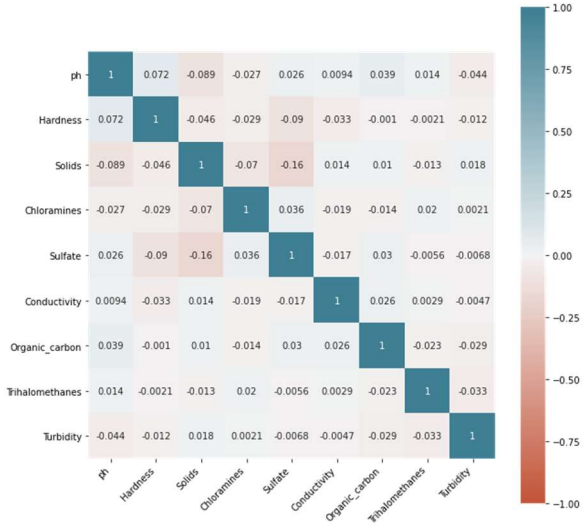

Figure 8-18: Correlation Plot of Predictor Variables

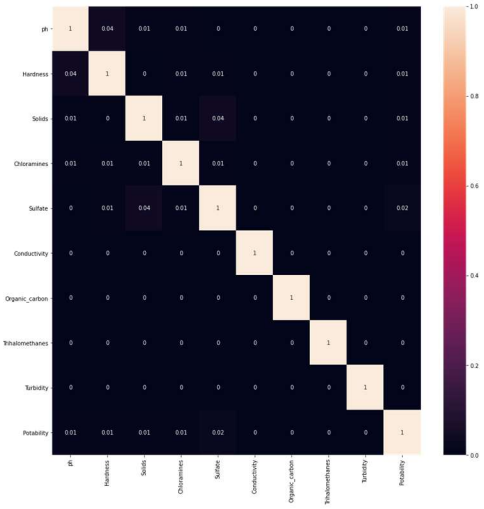

Figure 8-17: Correlation Heatmap of All Variables

9) Advanced Analysis

It has been concluded to use classification techniques such as Logistic Regression, Decision Tree Classifier, Random Forest, and XG Boosting from the results of the explorative data analysis. The AUC value and the Model Accuracy Score of models are used in evaluating the models. The advanced analysis was performed in two ways. The first way was done by changing some of the numerical variables to categorical variables in a proper way from the findings of a research paper. For example, hardness was classified into soft, slightly hard, moderately hard, hard, and very hard if values lie in between 0-17.1, 17.1-60, 60-120, 120- 180, and 180-above respectively. The second way was done by using the numerical variable directly in model fitting. From the results, it was found that using the numerical values directly gave a better result than the first way.

There were 491, 781, and 162 missing values in PH, Sulfate, and Trihalomethanes respectively. These values were less than 30% from the dataset and the variables were normally distributed. Hence, the missing values were imputed with the mean. The logistic regression is sensitive to outliers. Therefore, the outliers were handled by the capping method since the number of observations was small. The tree-based methods were not sensitive to outliers. Hence, the data was used without capping the outliers when fitting tree-based methods. The feature selection was done using recursive feature elimination.

The variables PH, hardness, chloramines, organic carbon, and turbidity were selected as the important variables in logistic regression. The parameters C and solver were chosen as 1 and 'lbfgs' by parameter tuning. The variables sulfate, solids, hardness, PH, and chloramines were selected as the important variables in the Decision Tree Method. The max depth, max features, min samples leaf, and min sample split were chosen as 9, auto, 5, and 0.01 respectively from parameter tuning. The variables PH, sulfate, hardness, solids, and chloramines were selected as the important variables in Random Forest Model. The criterion, max features, max depth, and the number of estimators were chosen as 'Gini', 9, auto, and 300 from parameter tuning. The variables PH, sulfate, hardness, solids, and chloramines were selected as the important variables in Random Forest Model. The gamma, learning rate, max depth, number of estimators,

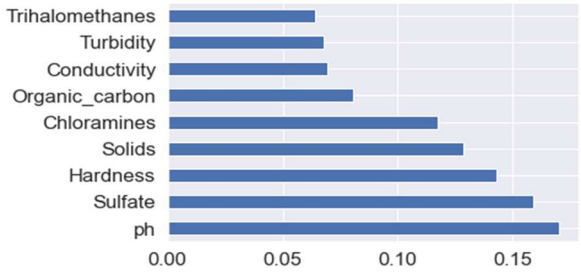

Figure 9-1: Feature Importance Plot of the XGBoost Model

and sub-sample were chosen as 0.1, 0.01, 8, 300, and 0.6 respectively from parameter tuning. The table shows the AUC scores and accuracy scores of the models. The figure shows the important variable plot of the XGBoost Model.

Models	AUC score	Accuracy Score
Logistic Regression	0.5	0.5991
Decision Tree	0.5720	0.6372
Random Forest	0.6069	0.6723
XG boost	0.6473	0.6992

Table 0-1: AUC Scores and the Accuracy Scores of the Models

10) Discussion and Conclusions

The main objective of the study is to predict whether the water body is safe for human consumption. The second objective is to find the factors that affect water purity. The objectives were achieved by the descriptive and advanced analysis.

The PH value, Sulfate, Hardness, Solids, and chloramines are the factors that mostly impact whether the water body can be used for the consumption or not since these variables were chosen as the important variables in most of the techniques. The XG boost model was selected as the best model to predict whether the water body is safe for human consumption. A website was created with the prediction feature using the best model.

In the advanced analysis, several ways have been tried to improve the model but no improvements were found in the accuracy and AUC scores. Therefore, it is suggested to take more observations to improve the model.

11) References

World Health Organisation. (2007). pH in drinking-water. *Guidelines for Drinking Water Quality*, *2*(2), 1–7. http://www.who.int/water_sanitation_health/dwq/chemicals/ph_revised_2007_clean_version.pdf

Usgs.gov. 2021. *Hardness of Water*. [online] Available at: <https://www.usgs.gov/special-topic/water-science-school/science/hardness-water?qt-science_center_objects=0#qt-science_center_objects>.

Health Canada. (1995). Sources and Levels of Hardness. *World Health*, *1979*(February 1979), 40–43.

Think Fish. 2013. [online] Available at: <https://www.thinkfish.co.uk/article/testing-for-ph-and-hardness-in-an-aquarium>.

State.ky.us. n.d. *Sulfate and water quality*. [online] Available at: <http://www.state.ky.us/nrepc/water/ramp/rmso4.htm>.

Oram, B., n.d. *Sulfate, Hydrogen Sulfide, Sulfate Reducing Bacteria - How to Identify and Manage*. [online] Water Research Center. Available at: <https://www.water-research.net/index.php/sulfates>.

Cdc.gov. 2020. *Water Disinfection with Chlorine and Chloramine | Public Water Systems | Drinking Water | Healthy Water | CDC*. [online] Available at: <https://www.cdc.gov/healthywater/drinking/public/water_disinfection.html>.

Friedler, E., Chavez, D. F., Alfiya, Y., Gilboa, Y., & Gross, A. (2021). Impact of suspended solids and organic matter on chlorine and UV disinfection efficiency of greywater. *Water (Switzerland)*, *13*(2). https://doi.org/10.3390/w13020214

Hood, E., 2005. Tap Water and Trihalomethanes: Flow of Concerns Continues. *Environmental Health Perspectives*, 113(7).

Archive.epa.gov. 2016. *EPA | Envirofacts | ICR | Regulations*. [online] Available at: <https://archive.epa.gov/enviro/html/icr/web/html/regulations.html>.

Amarasooriya, A., Weragoda, S., Makehelwala, M. and Weerasooriya, R., 2018. Occurrence of trihalomethane in relation to treatment technologies and water quality under tropical conditions. *H2Open Journal*, 1(1), pp.69-83.

WHO. (2017). WATER QUALITY AND HEALTH - REVIEW OF TURBIDITY: Information for regulators and water suppliers. *Who/Fwc/Wsh/17.01*, 10. https://www.who.int/water_sanitation_health/publications/turbidity-information-200217.pdf%0Ahttp://www.who.int/water_sanitation_health/publications/turbidity-information-200217.pdf

Mohammed, S., 2015. Effect of pH on the Turbidity Removal of Wastewater. *OALib*, 02(12), pp.1-9.

Hancock, N., n.d. *Conventional Water Treatment: Coagulation and Filtration — Safe Drinking Water Foundation*. [online] Safe Drinking Water Foundation. Available at: <https://www.safewater.org/fact-sheets-1/2017/1/23/conventional-water-treatment>.

Whitman.edu. n.d. [online] Available at: <https://www.whitman.edu/chemistry/edusolns_software/AlkalinityBackground.pdf>.

12) Appendix and Code

```
#!/usr/bin/env python
# coding: utf-8
# In[2]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.model_selection import
train_test_split
# In[109]:
data=pd.read_csv("D:/Studies/ST 4052 -
Statistical Learning II/Final
Project/water_potability.CSV")
# In[110]:
data.head()
# In[5]:
data.isna().any()
# In[6]:
data.isna().sum()
# In[7]:
data.isna().sum()/len(data)
# In[8]:
data_1=data.dropna(how="any")
data_1.head()
# In[9]:
stats.probplot(data_1["ph"], dist="norm",
plot=plt)
plt.show()
# In[9]:
stats.probplot(data_1["Sulfate"],
dist="norm", plot=plt)
plt.show()
# In[10]:
stats.probplot(data_1["Trihalomethanes"],
dist="norm", plot=plt)
plt.show()
# In[111]:
# split into train test sets
train, test =
train_test_split(data,test_size=0.2,random_
state= 20)
# In[114]:
train.shape
# In[115]:
test.shape
# In[94]:
train.isna().any()
# In[16]:
test.isna().any()
# In[16]:
train.isna().sum()
# In[18]:
test.isna().sum()
# In[19]:
train.isna().sum()/len(train)
# In[20]:
test.isna().sum()/len(test)
# In[116]:
train["ph"].fillna(train["ph"].mean(),inpla
ce=True)
train.head()
# In[117]:
train["Sulfate"].fillna(train["Sulfate"].me
an(),inplace=True)
train.head()
# In[118]:
train["Trihalomethanes"].fillna(train["Trih
alomethanes"].mean(),inplace=True)
train.head()
# In[119]:
train.isna().any()
# In[102]:
plt.hist(train["ph"], color="tomato")
plt.title("Histogram of pH")
# In[100]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of pH")
plt.boxplot(train["ph"])
# In[36]:
plt.hist(train["Hardness"], color="tomato")
plt.title("Histogram of Hardness")
# In[37]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Hardness")
plt.boxplot(train["Hardness"])
# In[38]:
plt.hist(train["Solids"], color="tomato")
plt.title("Histogram of Solids")
# In[39]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Solids")
plt.boxplot(train["Solids"])
# In[40]:
plt.hist(train["Chloramines"],
color="tomato")
plt.title("Histogram of Chloramines")
# In[41]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Chloramines")
plt.boxplot(train["Chloramines"])
# In[42]:
plt.hist(train["Sulfate"], color="tomato")
plt.title("Histogram of Sulfate")
# In[43]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Sulfate")
plt.boxplot(train["Sulfate"])
# In[44]:
plt.hist(train["Conductivity"],
color="tomato")
plt.title("Histogram of Conductivity")
# In[45]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Conductivity")
plt.boxplot(train["Conductivity"])
# In[103]:
plt.hist(train["Organic_carbon"],
color="tomato")
plt.title("Histogram of Organic Carbon")
# In[105]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Organic Carbon")
plt.boxplot(train["Organic_carbon"])
# In[48]:
plt.hist(train["Trihalomethanes"],
color="tomato")
plt.title("Histogram of Trihalomethanes")
# In[51]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Trihalomethanes")
plt.boxplot(train["Trihalomethanes"])
# In[52]:
plt.hist(train["Turbidity"],
color="tomato")
plt.title("Histogram of Turbidity")
# In[53]:
fig = plt.figure(figsize =(10, 7))
plt.title("Boxplot of Turbidity")
plt.boxplot(train["Turbidity"])
# In[62]:
plt.pie(train['Potability'].value_counts(),
labels=['Not Potable', 'Potable'],
colors=['tomato', 'teal'],
        startangle=90, shadow = True,
explode = (0, 0.1),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()
# plot title
plt.title('Potability')

# showing the plot
plt.show()
# In[466]:
import seaborn as sns
# In[83]:
my_pal = {1: "tomato", 0: "teal"}
for i,col in enumerate(train):
    sns.boxplot(x="Potability", y =
train[col], data=train, palette= my_pal)
    plt.title(f' {col} vs Potability')
    plt.figure(figsize=(15,15))
# In[120]:
bucket_ph = np.linspace(0, 14, 15)
bucket_ph
# In[121]:
train["ph"] = pd.cut(train["ph"],
bucket_ph)
train.head()
# In[100]:
train["ph"].value_counts().sort_index().plo
t.bar()
plt.xlabel("pH")
plt.ylabel("Frequency")
plt.title("pH")
# In[116]:
ph_Pot = pd.crosstab(index=train["ph"],
columns=train["Potability"])
ph_Pot
# In[117]:
ph_Pot = ph_Pot.div(ph_Pot.sum(axis = 1),
axis = 0)*100
ph_Pot
# In[138]:
ph_Pot.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("pH vs. Potability")
# In[123]:
ph_Pot1 = pd.crosstab(index=train["ph"],
columns=train["Potability"]).transpose()
ph_Pot1
# In[124]:
ph_Pot1 = ph_Pot1.div(ph_Pot1.sum(axis =
1), axis = 0)*100
ph_Pot1
# In[137]:
ph_Pot1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("pH vs. Potability")
plt.legend(loc='upper right')
# In[122]:
bucket_Har = np.linspace(45, 325, 11)
bucket_Har
# In[123]:
train["Hardness"] =
pd.cut(train["Hardness"], bucket_Har)
train.head()
# In[29]:
train["Hardness"].value_counts().sort_index
().plot.bar()
plt.xlabel("Hardness")
plt.ylabel("Frequency")
plt.title("Hardness")
# In[33]:
Har_Pot =
pd.crosstab(index=train["Hardness"],
columns=train["Potability"])
Har_Pot
# In[34]:
Har_Pot = Har_Pot.div(Har_Pot.sum(axis =
1), axis = 0)*100
Har_Pot
# In[35]:
Har_Pot.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Hardness vs. Potability")
# In[124]:
Har_Pot1 =
pd.crosstab(index=train["Hardness"],
columns=train["Potability"]).transpose()
Har_Pot1
# In[37]:
Har_Pot1 = Har_Pot1.div(Har_Pot1.sum(axis =
1), axis = 0)*100
Har_Pot1
# In[38]:
Har_Pot1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
```

```
plt.title("Hardness vs. Potability")
plt.legend(loc='upper right')
# In[125]:
bucket_Sol = np.linspace(0, 62000, 11)
bucket_Sol
# In[126]:
train["Solids"] = pd.cut(train["Solids"],
bucket_Sol)
train.head()
# In[127]:
bucket_Chl = np.linspace(0, 14, 15)
bucket_Chl
# In[128]:
train["Chloramines"] =
pd.cut(train["Chloramines"], bucket_Chl)
train.head()
# In[129]:
bucket_Sul = np.linspace(100, 500, 11)
bucket_Sul
# In[130]:
train["Sulfate"] = pd.cut(train["Sulfate"],
bucket_Sul)
train.head()
# In[131]:
bucket_Con = np.linspace(100, 800, 11)
bucket_Con
# In[132]:
train["Conductivity"] =
pd.cut(train["Conductivity"], bucket_Con)
train.head()
# In[133]:
bucket_Org = np.linspace(0, 30, 11)
bucket_Org
# In[134]:
train["Organic_carbon"] =
pd.cut(train["Organic_carbon"], bucket_Org)
train.head()
# In[135]:
bucket_Tri = np.linspace(0, 125, 11)
bucket_Tri
# In[136]:
train["Trihalomethanes"] =
pd.cut(train["Trihalomethanes"],
bucket_Tri)
train.head()
# In[137]:
bucket_Tur = np.linspace(1, 7, 7)
bucket_Tur
# In[138]:
train["Turbidity"] =
pd.cut(train["Turbidity"], bucket_Tur)
train.head()
# In[139]:
train["Solids"].value_counts().sort_index()
.plot.bar()
plt.xlabel("Solids")
plt.ylabel("Frequency")
plt.title("Solids")
# In[140]:
Har_Sol =
pd.crosstab(index=train["Solids"],
columns=train["Potability"])
Har_Sol
# In[141]:
Har_Sol = Har_Sol.div(Har_Sol.sum(axis =
1), axis = 0)*100
Har_Sol
# In[142]:
Har_Sol.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Solids vs. Potability")
# In[143]:
Har_Sol1 =
pd.crosstab(index=train["Solids"],
columns=train["Potability"]).transpose()
Har_Sol1
# In[145]:
Har_Sol1 = Har_Sol1.div(Har_Sol1.sum(axis =
1), axis = 0)*100
Har_Sol1
# In[146]:
Har_Sol1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Solids vs. Potability")
plt.legend(loc='upper right')
# In[147]:
train["Chloramines"].value_counts().sort_in
dex().plot.bar()
plt.xlabel("Chloramines")
plt.ylabel("Frequency")
plt.title("Chloramines")
# In[148]:
Har_Chl =
pd.crosstab(index=train["Chloramines"],
columns=train["Potability"])
Har_Chl
# In[149]:
Har_Chl = Har_Chl.div(Har_Chl.sum(axis =
1), axis = 0)*100
Har_Chl
# In[150]:
Har_Chl.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Chloramines vs. Potability")
# In[151]:
Har_Chl1 =
pd.crosstab(index=train["Chloramines"],
columns=train["Potability"]).transpose()
Har_Chl1
# In[152]:
Har_Chl1 = Har_Chl1.div(Har_Chl1.sum(axis =
1), axis = 0)*100
Har_Chl1
# In[153]:
Har_Chl1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Chloramines vs. Potability")
plt.legend(loc='upper right')
# In[159]:
for i,col in enumerate(train):

train[col].value_counts().sort_index().plot
.bar()
    plt.xlabel(f"{col}")
    plt.ylabel("Frequency")
    plt.title(f"{col}")
    plt.figure(figsize=(10,10))
# In[163]:
Har_Sul =
pd.crosstab(index=train["Sulfate"],
columns=train["Potability"])
Har_Sul
# In[164]:
Har_Con =
pd.crosstab(index=train["Conductivity"],
columns=train["Potability"])
Har_Con
# In[165]:
Har_Org =
pd.crosstab(index=train["Organic_carbon"],
columns=train["Potability"])
Har_Org
# In[166]:
Har_Tri =
pd.crosstab(index=train["Trihalomethanes"],
columns=train["Potability"])
Har_Tri
# In[167]:
Har_Tur =
pd.crosstab(index=train["Turbidity"],
columns=train["Potability"])
Har_Tur
# In[168]:
Har_Sul = Har_Sul.div(Har_Sul.sum(axis =
1), axis = 0)*100
Har_Sul
# In[169]:
Har_Con = Har_Con.div(Har_Con.sum(axis =
1), axis = 0)*100
Har_Con
# In[170]:
Har_Org = Har_Org.div(Har_Org.sum(axis =
1), axis = 0)*100
Har_Org
# In[171]:
Har_Tri = Har_Tri.div(Har_Tri.sum(axis =
1), axis = 0)*100
Har_Tri
# In[172]:
Har_Tur = Har_Tur.div(Har_Tur.sum(axis =
1), axis = 0)*100
Har_Tur
# In[268]:
Har_Sul.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Sulfate vs. Potability")
# In[174]:
Har_Con.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Conductivity vs. Potability")
# In[175]:
Har_Org.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Organic Carbon vs. Potability")
# In[176]:
Har_Tri.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Trihalomethanes vs. Potability")
# In[177]:
Har_Tur.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Turbidity vs. Potability")
# In[179]:
Har_Sul1 =
pd.crosstab(index=train["Sulfate"],
columns=train["Potability"]).transpose()
Har_Sul1
# In[180]:
Har_Con1 =
pd.crosstab(index=train["Conductivity"],
columns=train["Potability"]).transpose()
Har_Con1
# In[181]:
Har_Org1 =
pd.crosstab(index=train["Organic_carbon"],
columns=train["Potability"]).transpose()
Har_Org1
# In[182]:
Har_Tri1 =
pd.crosstab(index=train["Trihalomethanes"],
columns=train["Potability"]).transpose()
Har_Tri1
# In[183]:
Har_Tur1 =
pd.crosstab(index=train["Turbidity"],
columns=train["Potability"]).transpose()
Har_Tur1
# In[184]:
Har_Sul1 = Har_Sul1.div(Har_Sul1.sum(axis =
1), axis = 0)*100
Har_Sul1
# In[185]:
Har_Con1 = Har_Con1.div(Har_Con1.sum(axis =
1), axis = 0)*100
Har_Con1
# In[186]:
Har_Org1 = Har_Org1.div(Har_Org1.sum(axis =
1), axis = 0)*100
Har_Org1
# In[187]:
Har_Tri1 = Har_Tri1.div(Har_Tri1.sum(axis =
1), axis = 0)*100
Har_Tri1
# In[188]:
Har_Tur1 = Har_Tur1.div(Har_Tur1.sum(axis =
1), axis = 0)*100
Har_Tur1
# In[189]:
Har_Sul1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Sulfer vs. Potability")
plt.legend(loc='upper right')
# In[190]:
Har_Con1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Conductivity vs. Potability")
plt.legend(loc='upper right')
# In[191]:
Har_Org1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Organic Carbon vs. Potability")
plt.legend(loc='upper right')
# In[192]:
Har_Tri1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Trihalomethane vs. Potability")
plt.legend(loc='upper right')
# In[193]:
Har_Tur1.plot(kind = "bar", stacked = True,
figsize=(10, 10))
plt.title("Turbidity vs. Potability")
plt.legend(loc='upper right')
# In[458]:
ph_Sul =
pd.crosstab(index=train["Turbidity"],
columns=train["Conductivity"])
ph_Sul
# In[459]:
ph_Sul = ph_Sul.div(ph_Sul.sum(axis = 1),
axis = 0)*100
ph_Sul
# In[462]:
ph_Sul.plot(kind = "barh", stacked = True,
figsize=(10, 10))
plt.title("Turbidity vs. Conductivity")
plt.legend(loc='upper right')
# In[488]:
def cramers_v(x, y):
    x = np.array(x)
    y = np.array(y)
    confusion_matrix = pd.crosstab(x, y)
    chi2 =
stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2-((k-1)*(r-
1))/(n-1))
    rcorr = r-((r-1)**2)/(n-1)
    kcorr = k-((k-1)**2)/(n-1)
    return np.sqrt(phi2corr/min((kcorr-
1),(rcorr-1)))
columns = train.columns
cramersv =
pd.DataFrame(index=train.columns,columns=tr
ain.columns)
cramersv
# In[487]:
fig = plt.figure(figsize=(15,15))
sns.heatmap(cramersv, annot=True)
fig.savefig('catcorr.jpg',
bbox_inches='tight', dpi=150)
plt.show()
# In[476]:
from scipy.stats import chi2_contingency
def cramers_V(var1,var2) :
  crosstab =np.array(pd.crosstab(var1,var2,
rownames=None, colnames=None)) # Cross
table building
  stat = chi2_contingency(crosstab)[0] #
Keeping of the test statistic of the Chi2
test
  obs = np.sum(crosstab) # Number of
observations
  mini = min(crosstab.shape)-1 # Take the
minimum value between the columns and the
rows of the cross table
  return (stat/(obs*mini))
# In[477]:
rows= []
for var1 in train:
  col = []
  for var2 in train :
    cramers =cramers_V(train[var1],
train[var2]) # Cramer's V test
    col.append(round(cramers,2)) # Keeping
of the rounded value of the Cramer's V
  rows.append(col)

cramers_results = np.array(rows)
df = pd.DataFrame(cramers_results, columns
= train.columns, index =train.columns)
df
# In[480]:
fig = plt.figure(figsize=(15,15))
sns.heatmap(df, annot=True)
fig.savefig('catcorr.jpg',
bbox_inches='tight', dpi=150)
plt.show()
# In[481]:
import scipy.stats as ss
```

```
def
cramers_corrected_stat(confusion_matrix):
    """ calculate Cramers V statistic for
categorial-categorial association.
        uses correction from Bergsma and
Wicher,
        Journal of the Korean Statistical
Society 42 (2013): 323-328
    """
    chi2 =
ss.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-
1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min( (kcorr-
1), (rcorr-1)))
# In[482]:
confusion_matrix =
pd.crosstab(train[column1], train[column2])
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import
StandardScaler
from sklearn.linear_model import
LogisticRegression
from sklearn.ensemble import
RandomForestClassifier
from sklearn.tree import
DecisionTreeClassifier

%matplotlib inline

df = pd.read_csv('water_potability.csv')
df.shape
df.head()
df.isnull().sum()
train.isnull().mean()*100
train, test =
train_test_split(df,test_size=0.2,random_st
ate= 20)

train['ph'] =
train['ph'].fillna(train['ph'].mean())
test['ph'] =
test['ph'].fillna(train['ph'].mean())
train['Sulfate'] =
train['Sulfate'].fillna(train['Sulfate'].me
an())
test['Sulfate'] =
test['Sulfate'].fillna(train['Sulfate'].mea
n())
train['Trihalomethanes'] =
train['Trihalomethanes'].fillna(train['Trih
alomethanes'].mean())
test['Trihalomethanes'] =
test['Trihalomethanes'].fillna(train['Triha
lomethanes'].mean())
X_train = train.iloc[:,:-1]
Y_train = train.iloc[:,-1:]
X_test = test.iloc[:,:-1]
Y_test = test.iloc[:,-1:]

from sklearn.tree import
DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.model_selection import
GridSearchCV
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import
confusion_matrix
from sklearn.metrics import
classification_report

params = {
    'max_depth': [2, 3, 4, 5,6,7,8,9,10],
    'min_samples_leaf': [5, 8, 12, 15, 20],
    'max_features' : ['auto', 'sqrt',
'log2'],
    'min_samples_split' : [0.01,0.05,0.1]
}

clf_TUNED =
DecisionTreeClassifier(random_state=20)
grid_search =
GridSearchCV(estimator=clf_TUNED,
param_grid=params,cv=5,


           n_jobs=-1, verbose=1, scoring =
"accuracy")
grid_search.fit(X_train, Y_train)
grid_search.best_params_

DT_clf = DecisionTreeClassifier(max_depth =
7, max_features = 'auto', min_samples_leaf
= 5,
min_samples_split=0.01,random_state=20)
DT_clf.fit(X_train, Y_train)
Y_pred = DT_clf.predict(X_test)

features = X_train.columns
importances = DT_clf.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)),
importances[indices], color='b',
align='center')
plt.yticks(range(len(indices)),
[features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
important_feat =
list(feat_imp_DT.nlargest(5).index)
X_train_imp = X_train.loc[:,important_feat]
X_test_imp = X_test.loc[:,important_feat]
grid_search.fit(X_train_imp, Y_train)
DT_clf_imp =
DecisionTreeClassifier(max_depth = 9,
max_features = 'auto', min_samples_leaf =
5, min_samples_split=0.01,random_state=20)
DT_clf_imp.fit(X_train_imp, Y_train)
Y_pred_imp = DT_clf_imp.predict(X_test_imp)
print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(Y_test, Y_pred_imp)))
false_positive_rate, true_positive_rate,
thresholds = roc_curve(Y_test, Y_pred_imp)
roc_auc = auc(false_positive_rate,
true_positive_rate)
print(confusion_matrix(Y_test, Y_pred_imp))
print(classification_report(Y_test,
Y_pred_imp))

#random forest
param_grid_RF = {
    'n_estimators': [100,200,300,400,500],
    'max_features': ['auto', 'sqrt',
'log2'],
    'max_depth' : [6,7,8,9,10],
    'criterion' :['gini', 'entropy']
}
RFC =
RandomForestClassifier(random_state=20)
Grid_rf= GridSearchCV(estimator=RFC,
param_grid=param_grid_RF, cv= 5,n_jobs=-
1,verbose =1,scoring='roc_auc')
Grid_rf.fit(X_train, Y_train)
Grid_rf.best_params_
RF =
RandomForestClassifier(n_estimators=400,max
_features='auto',max_depth=10,criterion='en
tropy', random_state=20 ,verbose = 1)
feat_imp_RF =
pd.Series(RF.feature_importances_,
index=X_train.columns)
feat_imp_RF.nlargest(15).plot(kind='barh')
important_feat =
list(feat_imp_RF.nlargest(5).index)
X_train_imp = X_train.loc[:,important_feat]
X_test_imp = X_test.loc[:,important_feat]
Grid_rf= GridSearchCV(estimator=RFC,
param_grid=param_grid_RF, cv= 5,n_jobs=-
1,verbose =1,scoring='roc_auc')
Grid_rf.fit(X_train_imp, Y_train)
Grid_rf.best_params_
RF =
RandomForestClassifier(n_estimators=300,max
_features='auto',max_depth=9,criterion='gin
i', random_state=20 ,verbose = 1)
RF.fit(X_train_imp, Y_train)
Y_pred_imp = RF.predict(X_test_imp)
print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(Y_test, Y_pred_imp)))
false_positive_rate, true_positive_rate,
thresholds = roc_curve(Y_test, Y_pred_imp)
roc_auc = auc(false_positive_rate,
true_positive_rate)
print(confusion_matrix(Y_test, Y_pred_imp))
print(classification_report(Y_test,
Y_pred_imp))

#xgboost
import xgboost as xgb
from xgboost import XGBClassifier
xgb_model = XGBClassifier(
random_state=20,learning_rate=0.01,eval_met
ric='auc')
# fit the model with the training data
xgb_model.fit(X_train,Y_train)
feat_imp_xgb =
pd.Series(xgb_model.feature_importances_,
index=X_train.columns)
feat_imp_xgb.nlargest(15).plot(kind='barh')
imp_feat_xgb =
list(feat_imp_xgb.nlargest(5).index)
X_train_xgb = X_train.loc[:,imp_feat_xgb]
X_test_xgb = X_test.loc[:,imp_feat_xgb]
params_xgb = {
    'max_depth': [4,5,6,7,8],
    'n_estimators': [100,200,300,400,500],
    'learning_rate': [0.1, 0.01, 0.05],
    'subsample':[i/10.0 for i in
range(6,8)],
    'gamma':[i/10.0 for i in range(0,3)],
}
estimator = XGBClassifier(
    random_state=20,
    nthread=4
)
grid_search_xgb =
GridSearchCV(estimator=estimator,

param_grid=params_xgb,
                                cv=5, n_jobs=-1,
verbose=1)
grid_search_xgb.fit(X_train_xgb, Y_train)
grid_search_xgb.best_params_
final_xgb = XGBClassifier(base_score=0.5,
booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1,
colsample_bytree=1, gamma=0.1, gpu_id=-1,
              importance_type='gain',
interaction_constraints='',
              learning_rate=0.01,
max_delta_step=0, max_depth=8,
              min_child_weight=1,
monotone_constraints='()',
              n_estimators=300, n_jobs=4,
nthread=4, num_parallel_tree=1,
              random_state=20, reg_alpha=0,
reg_lambda=1, scale_pos_weight=1,
              subsample=0.6,
tree_method='exact', validate_parameters=1,
              verbosity=None)
final_xgb.fit(X_train_xgb, Y_train)
y_pred_xgb = final_xgb.predict(X_test_xgb)
print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(Y_test, y_pred_xgb)))
false_positive_rate, true_positive_rate,
thresholds = roc_curve(Y_test, y_pred_xgb)
roc_auc = auc(false_positive_rate,
true_positive_rate)

#handling outliers
#finding the Q1(25 percentile) and Q3(75
percentile)
q1 = train["ph"].quantile(0.25)
q2 = train["ph"].quantile(0.75)
#finding out the value of Inter Quartile
Range
IQR = q2 - q1
#defining max and min limits
max_limit = q2 + (1.5 * IQR)
min_limit = q1 - (1.5 * IQR)

#capping
train_ph= pd.DataFrame(np.where(train["ph"]
> max_limit, max_limit,
        (np.where(train["ph"] < min_limit,
min_limit, train["ph"]))), columns=["ph"])
train_ph
#finding the Q1(25 percentile) and Q3(75
percentile)
q1 = train['Hardness'].quantile(0.25)
q2 = train['Hardness'].quantile(0.75)
#finding out the value of Inter Quartile
Range
IQR = q2 - q1
#defining max and min limits
max_limit = q2 + (1.5 * IQR)
min_limit = q1 - (1.5 * IQR)

#capping
train_Hardness=
pd.DataFrame(np.where(train['Hardness'] >
max_limit, max_limit,
        (np.where(train['Hardness'] <
min_limit, min_limit, train['Hardness']))),
columns=["Hardness"])
train_Hardness
frames =
[train_ph,train_Hardness,train_Solids,train
_Chloramines,train_Sulfate,train_Conductivi
ty,train_Organic_carbon,

train_Trihalomethanes,train_Turbidity,train
_Potability]
new_train = pd.concat(frames, axis=1,
join='inner')

#logistic regression
LR = LogisticRegression(random_state=20)
LR.fit(X_train, Y_train)
#feature selection sing rfe
from sklearn.feature_selection import RFE
rfe = RFE(LR, 5)
fit = rfe.fit(X_train, Y_train)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" %
fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
columns = list(X_train.columns)
imp_feat =[]
for i in range(len(columns)):
    if (fit.support_[i] == True ):
        imp_feat.append(columns[i])
imp_feat
X_train_imp = X_train.loc[:,imp_feat]
X_test_imp = X_test.loc[:,imp_feat]
LR = LogisticRegression(random_state=20)
params = {
    'solver': ['lbfgs', 'liblinear', 'sag',
'saga'],
    'C' : [1.0,0.5, 0.1,0.05, 0.01]
}
grid_search = GridSearchCV(estimator=LR,

param_grid=params,
                           cv=5, n_jobs=-1,
verbose=1, scoring = "accuracy")
grid_search.fit(X_train_imp, Y_train)
LR_tuned =
LogisticRegression(C=1,solver='lbfgs',rando
m_state=20)
LR_tuned.fit(X_train_imp, Y_train)
Y_pred_imp = LR_tuned.predict(X_test_imp)
```