DS Project_AP19110010217

October 17, 2021

1 Importing important libraries

[1]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from pandas.core.common import SettingWithCopyWarning
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)
```

1.1 Reading the Dataset "Water_potability"

[2]:
```
df=pd.read_csv("water_potability.csv")
df
```

[2]:
```
            ph    Hardness        Solids  Chloramines     Sulfate  \
0          NaN  204.890455  20791.318981     7.300212  368.516441
1     3.716080  129.422921  18630.057858     6.635246         NaN
2     8.099124  224.236259  19909.541732     9.275884         NaN
3     8.316766  214.373394  22018.417441     8.059332  356.886136
4     9.092223  181.101509  17978.986339     6.546600  310.135738
…          …           …             …           …           …
3271  4.668102  193.681735  47580.991603     7.166639  359.948574
3272  7.808856  193.553212  17329.802160     8.061362         NaN
3273  9.419510  175.762646  33155.578218     7.350233         NaN
3274  5.126763  230.603758  11983.869376     6.303357         NaN
3275  7.874671  195.102299  17404.177061     7.509306         NaN

      Conductivity  Organic_carbon  Trihalomethanes  Turbidity  Potability
0       564.308654       10.379783        86.990970   2.963135           0
1       592.885359       15.180013        56.329076   4.500656           0
2       418.606213       16.868637        66.420093   3.055934           0
3       363.266516       18.436524       100.341674   4.628771           0
4       398.410813       11.558279        31.997993   4.075075           0
…              …               …               …          …           …
3271    526.424171       13.894419        66.687695   4.435821           1
```

```
3272    392.449580       19.903225             NaN   2.798243           1
3273    432.044783       11.039070       69.845400   3.298875           1
3274    402.883113       11.168946       77.488213   4.708658           1
3275    327.459760       16.140368       78.698446   2.309149           1

[3276 rows x 10 columns]
```

[3]:
```python
df['ph'].describe()
```

[3]:
```
count    2785.000000
mean        7.080795
std         1.594320
min         0.000000
25%         6.093092
50%         7.036752
75%         8.062066
max        14.000000
Name: ph, dtype: float64
```

1.2 Data Analysis

[4]:
```python
# Printing first 5 rows of the data
df.head()
```

[4]:
```
         ph    Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0       NaN  204.890455  20791.318981     7.300212  368.516441    564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN    592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN    418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136    363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738    398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
```

[5]:
```python
# Printing last 5 rows of the data
df.tail(5)
```

[5]:
```
            ph    Hardness        Solids  Chloramines     Sulfate  \
3271  4.668102  193.681735  47580.991603     7.166639  359.948574
3272  7.808856  193.553212  17329.802160     8.061362         NaN
3273  9.419510  175.762646  33155.578218     7.350233         NaN
3274  5.126763  230.603758  11983.869376     6.303357         NaN
3275  7.874671  195.102299  17404.177061     7.509306         NaN
```

```
      Conductivity  Organic_carbon  Trihalomethanes  Turbidity  Potability
3271    526.424171       13.894419        66.687695   4.435821           1
3272    392.449580       19.903225              NaN   2.798243           1
3273    432.044783       11.039070        69.845400   3.298875           1
3274    402.883113       11.168946        77.488213   4.708658           1
3275    327.459760       16.140368        78.698446   2.309149           1
```

[6]:
```python
# Io understand the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

[7]:
```python
# Printing the shape of data
df.shape
```

[7]:
```
(3276, 10)
```

[8]:
```python
df.describe()
```

[8]:
```
                ph     Hardness        Solids  Chloramines      Sulfate  \
count  2785.000000  3276.000000   3276.000000  3276.000000  2495.000000
mean      7.080795   196.369496  22014.092526     7.122277   333.775777
std       1.594320    32.879761   8768.570828     1.583085    41.416840
min       0.000000    47.432000    320.942611     0.352000   129.000000
25%       6.093092   176.850538  15666.690297     6.127421   307.699498
50%       7.036752   196.967627  20927.833607     7.130299   333.073546
75%       8.062066   216.667456  27332.762127     8.114887   359.950170
max      14.000000   323.124000  61227.196008    13.127000   481.030642

       Conductivity  Organic_carbon  Trihalomethanes    Turbidity   Potability
count   3276.000000     3276.000000      3114.000000  3276.000000  3276.000000
```

```
mean    426.205111    14.284970     66.396293   3.966786   0.390110
std      80.824064     3.308162     16.175008   0.780382   0.487849
min     181.483754     2.200000      0.738000   1.450000   0.000000
25%     365.734414    12.065801     55.844536   3.439711   0.000000
50%     421.884968    14.218338     66.622485   3.955028   0.000000
75%     481.792304    16.557652     77.337473   4.500320   1.000000
max     753.342620    28.300000    124.000000   6.739000   1.000000
```

[9]:
```
 #Checking if there's any duplicated data is present
df.duplicated().sum()
```

[9]: 0

[10]:
```
## handling duplicate values first lets see how many null values are there
df.isnull().sum()
```

[10]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

[11]:
```
#Cleaning data and fixing missing values by their mean values
df['ph'] = df['ph'].fillna(df['ph'].mean())
df['Trihalomethanes'] = df['Trihalomethanes'].fillna(df['Trihalomethanes'].
 ↪mean())
df['Sulfate'] = df['Sulfate'].fillna(df['Sulfate'].mean())
```

[12]:
```
#so we no longer have null values now
df.isnull().sum()
```

[12]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
```

```
dtype: int64
```

[13]:
```
#statistics of given data after removing null values
df.describe()
```

[13]:
```
                ph     Hardness        Solids  Chloramines      Sulfate  \
count  3276.000000  3276.000000   3276.000000  3276.000000  3276.000000
mean      7.080795   196.369496  22014.092526     7.122277   333.775777
std       1.469956    32.879761   8768.570828     1.583085    36.142612
min       0.000000    47.432000    320.942611     0.352000   129.000000
25%       6.277673   176.850538  15666.690297     6.127421   317.094638
50%       7.080795   196.967627  20927.833607     7.130299   333.775777
75%       7.870050   216.667456  27332.762127     8.114887   350.385756
max      14.000000   323.124000  61227.196008    13.127000   481.030642

       Conductivity  Organic_carbon  Trihalomethanes    Turbidity  Potability
count   3276.000000     3276.000000      3276.000000  3276.000000 3276.000000
mean     426.205111       14.284970        66.396293     3.966786    0.390110
std       80.824064        3.308162        15.769881     0.780382    0.487849
min      181.483754        2.200000         0.738000     1.450000    0.000000
25%      365.734414       12.065801        56.647656     3.439711    0.000000
50%      421.884968       14.218338        66.396293     3.955028    0.000000
75%      481.792304       16.557652        76.666609     4.500320    1.000000
max      753.342620       28.300000       124.000000     6.739000    1.000000
```

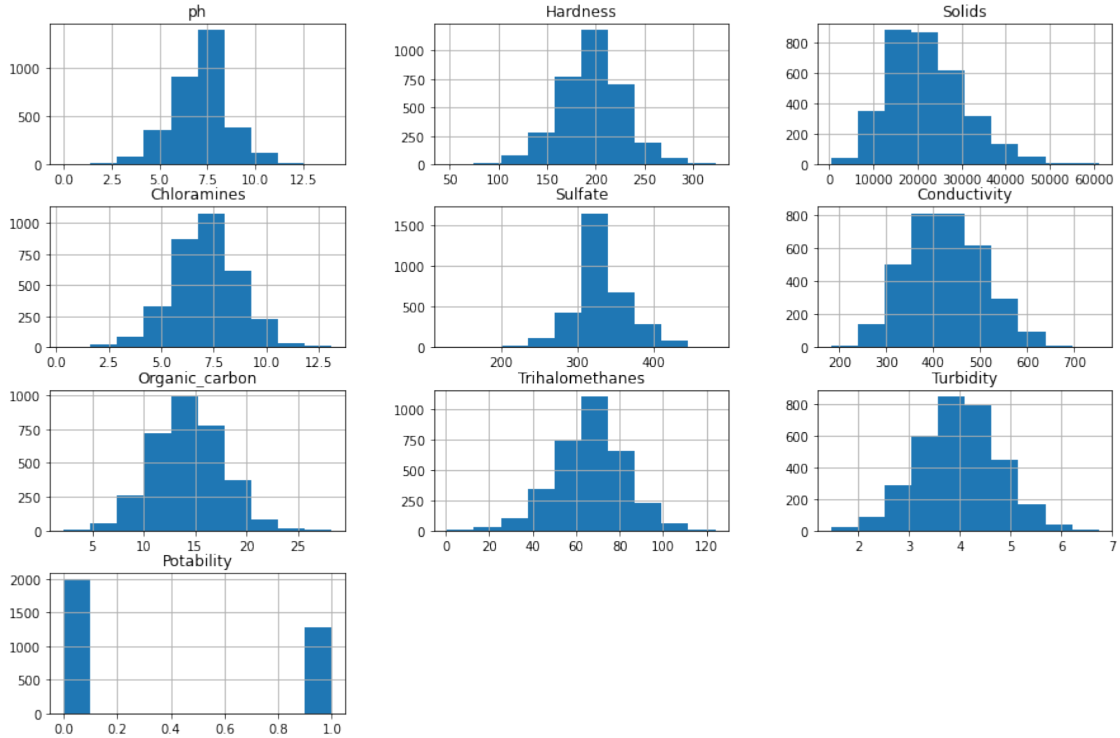
[14]:
```
#column names
df.columns
```

[14]:
```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

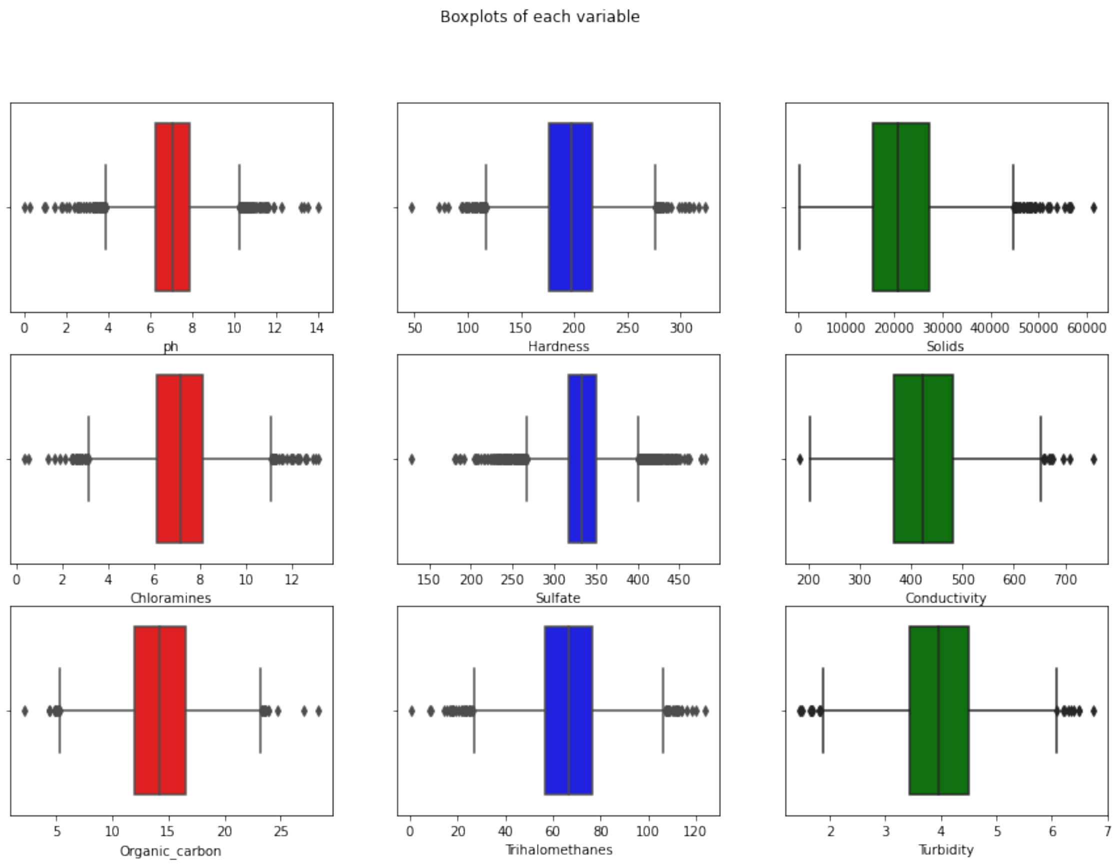
1.3 Data Visualisation

[15]:
```
#histogram
df.hist(figsize=(15,10))
plt.show
```

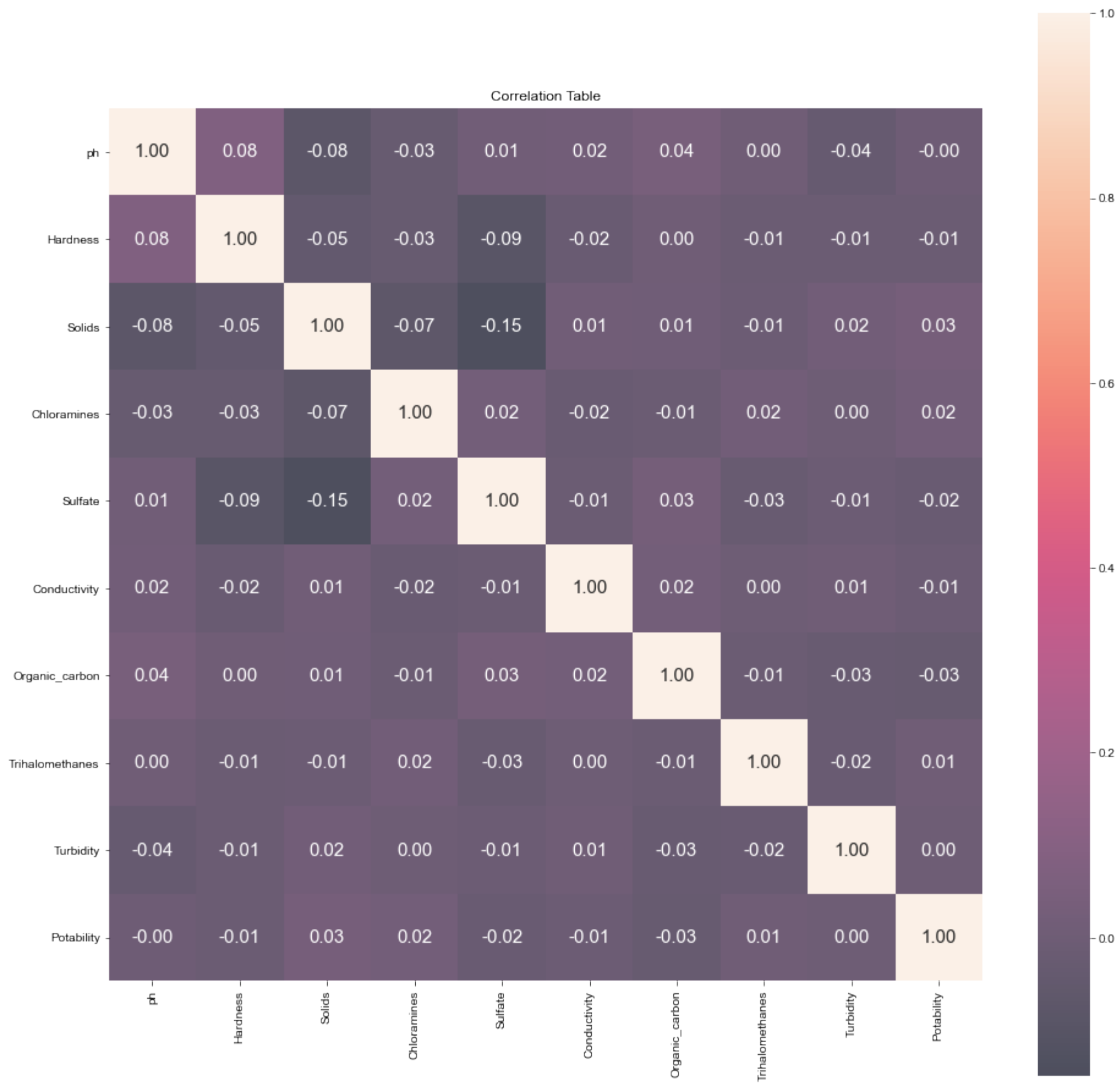
[15]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



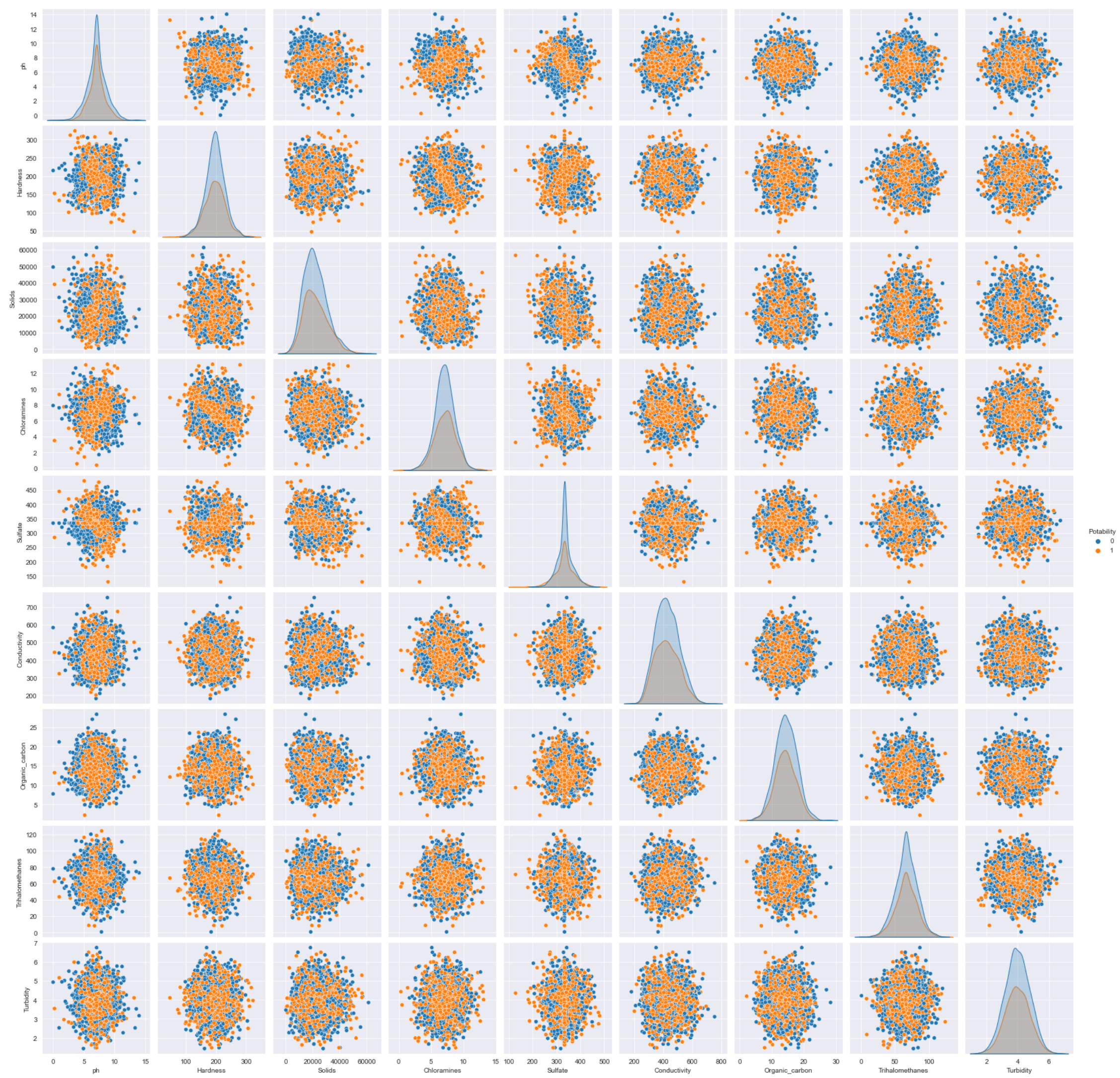
[16]:
```
#box plots
fig, axes = plt.subplots(3, 3,figsize=(15,10))
column = df.columns
fig.suptitle('Boxplots of each variable')
sns.boxplot(ax=axes[0,0],x=column[0],data=df,color='r')
sns.boxplot(ax=axes[0,1],x=column[1],data=df,color='b')
sns.boxplot(ax=axes[0,2],x=column[2],data=df,color='g')
sns.boxplot(ax=axes[1,0],x=column[3],data=df,color='r')
sns.boxplot(ax=axes[1,1],x=column[4],data=df,color='b')
sns.boxplot(ax=axes[1,2],x=column[5],data=df,color='g')
sns.boxplot(ax=axes[2,0],x=column[6],data=df,color='r')
sns.boxplot(ax=axes[2,1],x=column[7],data=df,color='b')
sns.boxplot(ax=axes[2,2],x=column[8],data=df,color='g')
plt.show()
```



[17]:
```python
# Heatmap
corr = df.corr()
plt.figure(figsize=(16,16))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.
 ↪2f',annot_kws={'size': 15},
           xticklabels=df.columns.values, yticklabels= df.columns.values, alpha␣
 ↪= 0.7)
plt.title("Correlation Table")
sns.set_style('darkgrid')
plt.show()
```



[18]:
```
sns.pairplot(df,hue='Potability')
sns.set_style('darkgrid')
```

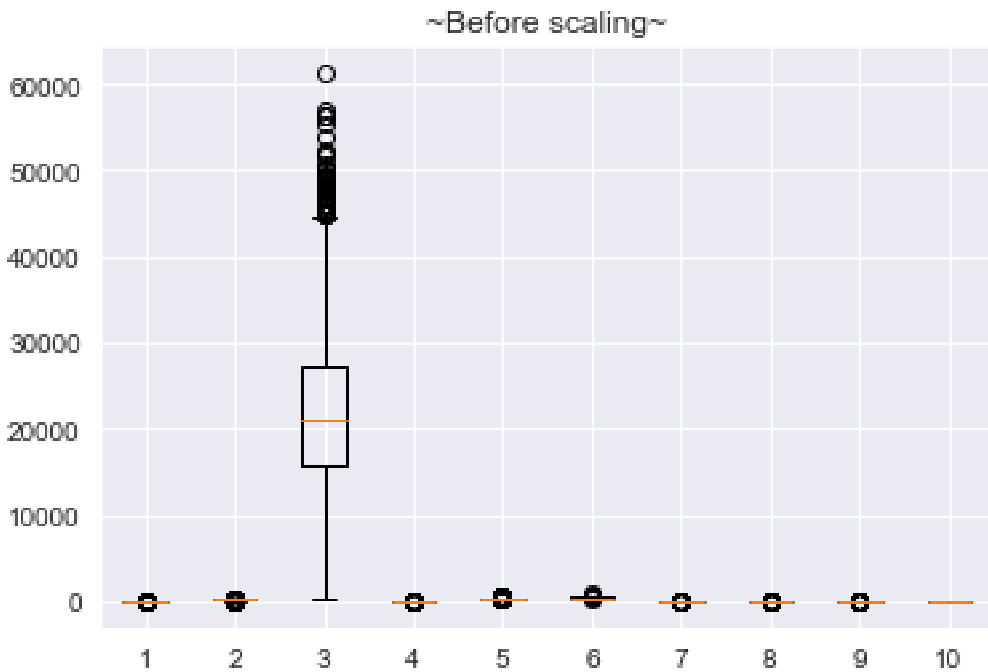


1.3.1 Standardising the data set by using different scalling methods

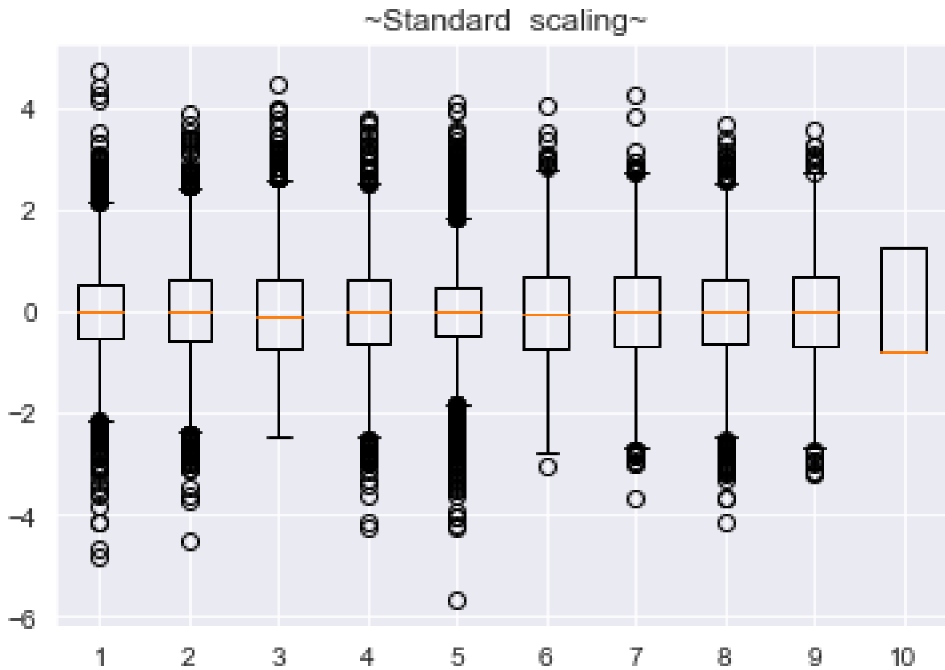
[19]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
s=StandardScaler()
m=MinMaxScaler()
r=RobustScaler()
n=Normalizer()

s1=s.fit_transform(df)
m1=m.fit_transform(df)
r1=r.fit_transform(df)
```

```
n1=n.fit_transform(df)
```

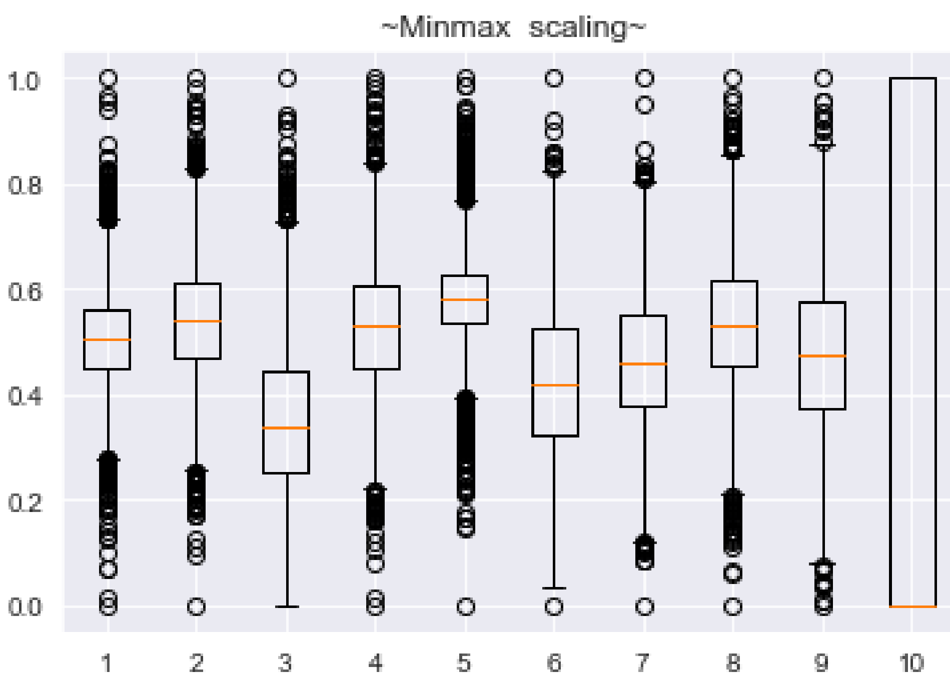
[20]:
```
plt.boxplot(df)
plt.title("~Before scaling~")
plt.show()
```



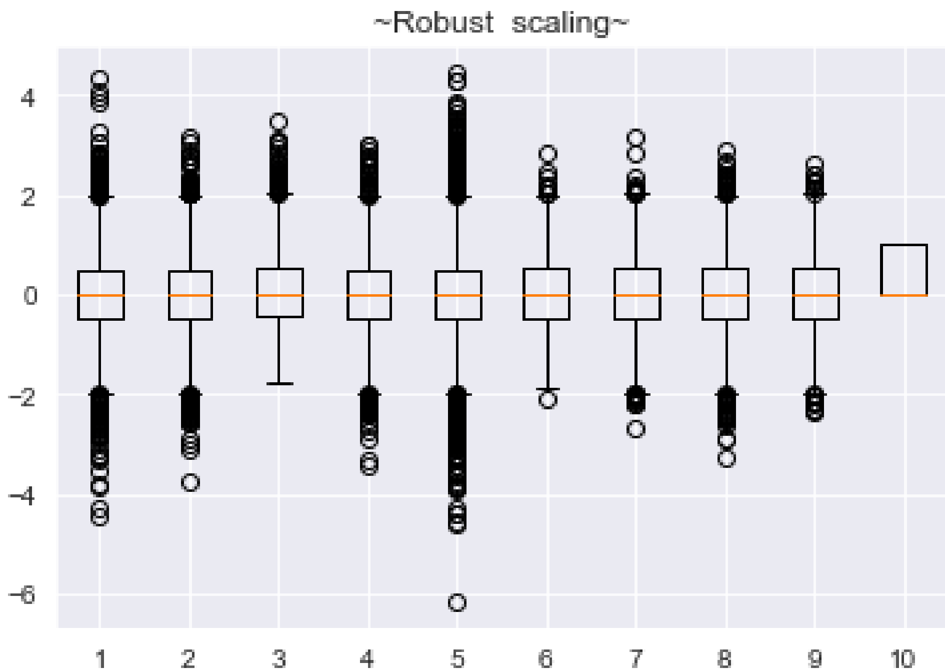
[21]:
```
plt.boxplot(s1)
plt.title(" ~Standard  scaling~")
plt.show()
```



[22]:
```
plt.boxplot(m1)
plt.title(" ~Minmax  scaling~")
plt.show()
```



[23]:
```
plt.boxplot(r1)
plt.title("~Robust  scaling~")
plt.show()
```

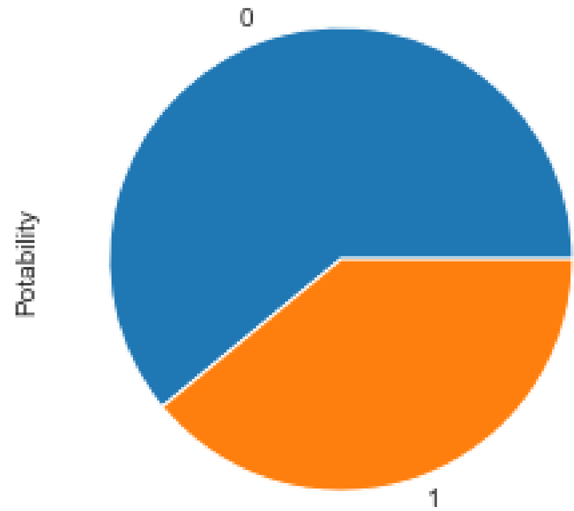


[24]:
```
## potability only have 2 values
df['Potability'].value_counts()
```

[24]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

[25]:
```
#pie chart
pota_count = df['Potability'].value_counts()
print(pota_count)
pota_count.plot.pie();
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```



1.4 Predicting potability using PH values

[26]:
```
# Data is in single dimension, so we reshape the input and output data(ph and␣
↪potability respectively)
x=df['ph'].values.reshape(-1,1)
y=df['Potability'].values.reshape(-1,1)
```

[27]:
```
#Splitting our data into train and test data by using  standard scalar
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[28]:
```
x=sc.fit_transform(x) # calculating mean and variance (x scaled = x-mean/sd)
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=10) # split our data into 80-20
```

[29]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(C=100000, fit_intercept=False) # fit intercept = False␣
↪is used to avoid zero biased fitting
model.fit(x_train,y_train.squeeze()) #squeeze 1d enteries
```

[29]:
```
LogisticRegression(C=100000, fit_intercept=False)
```

[30]:
```
pred=model.predict(x_test) # based on prior knowledge predict outputs for new␣
↪set of test data
```

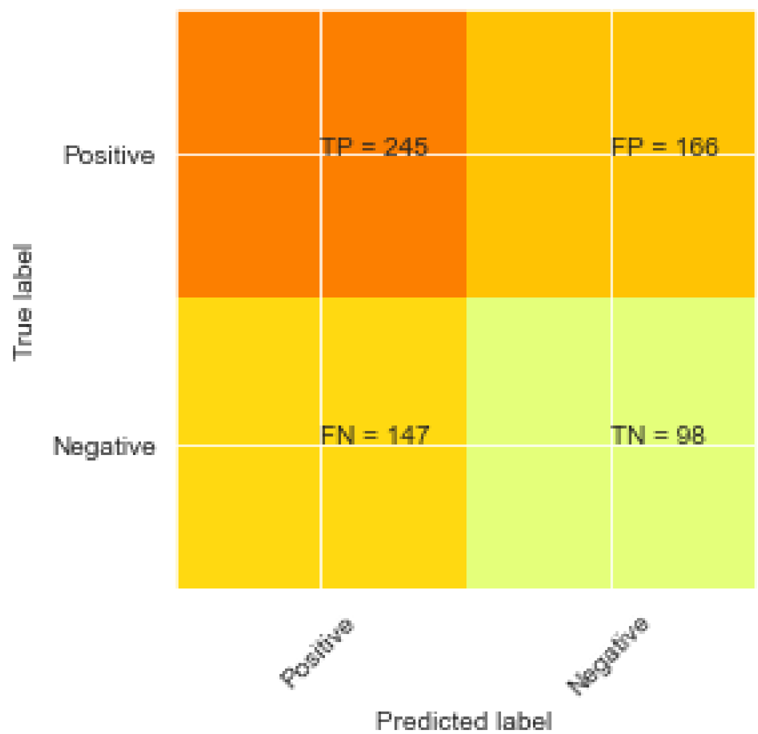
[31]:
```python
#checking the score on the training set
model.score(x_test,y_test) # how accurate our model predicted given test set␣
↪inputs
```

[31]: 0.5228658536585366

[32]:
```python
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[245 166]
 [147  98]]
```

[33]:
```python
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['Positive','Negative']
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TP','FP'], ['FN', 'TN']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
plt.show()
```



1.5 Predicting Potability using Hardness values

[34]:
```
# Data is in single dimension, so we reshape the input and output data(ph and␣
↪potability respectively)
x=df['Hardness'].values.reshape(-1,1)
y=df['Potability'].values.reshape(-1,1)
```

[35]:
```
#Splitting our data into train and test data by using  standard scalar
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[36]:
```
x=sc.fit_transform(x) # calculating mean and variance (x scaled = x-mean/sd)
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=10) # split our data into 80-20
```

[37]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(C=100000, fit_intercept=False) # fit intercept = False␣
↪is used to avoid zero biased fitting
model.fit(x_train,y_train.squeeze()) #squeeze 1d enteries
```

[37]:
```
LogisticRegression(C=100000, fit_intercept=False)
```

[38]:
```
pred=model.predict(x_test) # based on prior knowledge predict outputs for new␣
 ↪set of test data
```

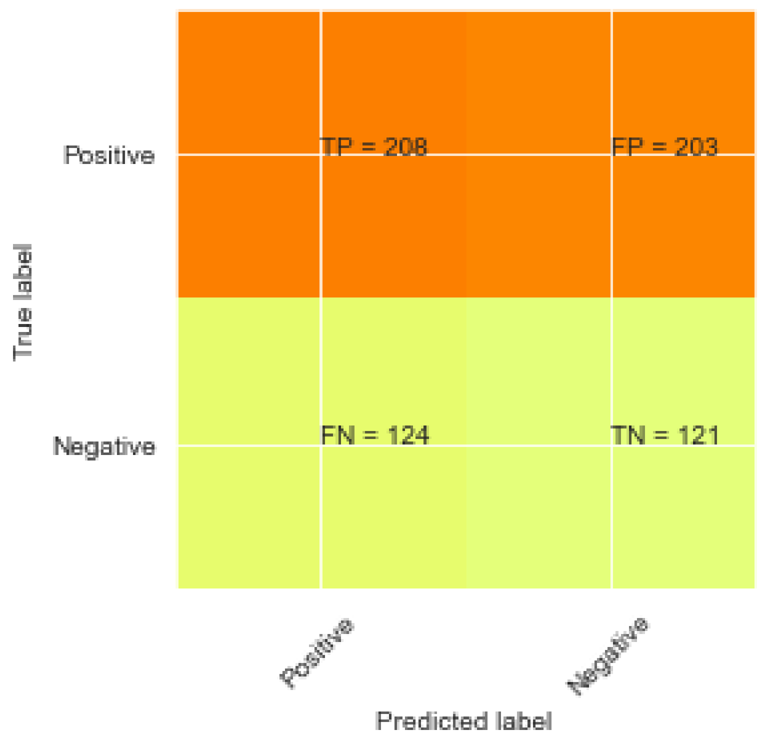
[39]:
```
#checking the score on the training set
model.score(x_test,y_test) # how accurate our model predicted given test set␣
 ↪inputs
```

[39]: 0.5015243902439024

[40]:
```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[208 203]
 [124 121]]
```

[41]:
```
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['Positive','Negative']
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TP','FP'], ['FN', 'TN']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
plt.show()
```



1.6 Predicting Potability using Sulphate values

[50]:
```
# Data is in single dimension, so we reshape the input and output data(ph and␣
↪potability respectively)
x=df['Solids'].values.reshape(-1,1)
y=df['Potability'].values.reshape(-1,1)
```

[51]:
```
#Splitting our data into train and test data by using  standard scalar
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[52]:
```
x=sc.fit_transform(x) # calculating mean and variance (x scaled = x-mean/sd)
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=10) # split our data into 80-20
```

[53]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(C=100000, fit_intercept=False) # fit intercept = False␣
↪is used to avoid zero biased fitting
model.fit(x_train,y_train.squeeze()) #squeeze 1d enteries
```

[53]:
```
LogisticRegression(C=100000, fit_intercept=False)
```

[54]:
```
pred=model.predict(x_test) # based on prior knowledge predict outputs for new␣
↪set of test data
```

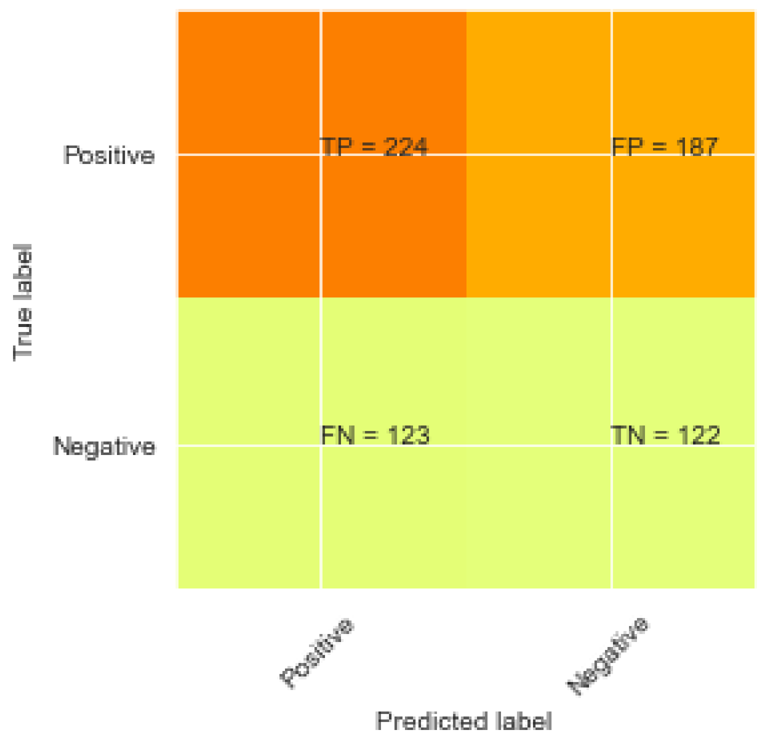
[55]:
```
#checking the score on the training set
model.score(x_test,y_test) # how accurate our model predicted given test set␣
↪inputs
```

[55]: 0.5274390243902439

[56]:
```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[224 187]
 [123 122]]
```

[57]:
```
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['Positive','Negative']
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TP','FP'], ['FN', 'TN']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
plt.show()
```



So far we checked the potability based on Ph, Hardness and Solids Values. Almost all are giving the same prediction but using Ph values has given 2% more accurate results, and in this case we also got highest true positives.

```
[ ]:
```