ANAND INSTITUTE OF HIGHER
TECHNOLOGY OLD MAHABALIPURAM ROAD,
KALASALINGAM NAGAR,
KAZHIPATTUR.

Water Quality Analysis
By
Data Analytics with Cognos

PHASE - 5

PROJECT : Water Quality Analysis

Project ID : Proj_229797_Team_2

Team members: Ranjith kumar v
Sharath p

Team leader: SURENTHER S

WATER QUALITY ANALYSIS

WATER QUALITY ANALYSIS:-

Discover the importance of water quality analysis and how it impacts our daily lives. Explore the methods and parameters used to test water quality, as well asthe relevant standards and regulations. Uncover the potential consequences of poor water quality and the steps we can take to ensure a clean and sustainable watersupply.

INTRODUCTION:-

Water quality analysis is a process of evaluating the physical, chemical, and biological characteristics of water to determine its suitability for various purposes, such as drinking, agriculture, industrial use, or aquatic ecosystems. This typically involves testing for parameters like pH, turbidity, dissolved oxygen, temperature, and concentrations of contaminants like heavy metals, bacteria, and organic compounds. Various techniques and instruments, such as spectrophotometers, pH meters, and microbial tests, are used to assess water quality.

DATA PREPROCESSING

Data preprocessing is a fundamental step in the data analysis pipeline. It involves cleaning and transforming raw data into a format suitable for analysis. The process typically includes tasks like handling missing values, removing duplicates, scaling and normalizing data, encoding categorical variables, and feature selection or extraction. Data preprocessing aims to enhance the quality of data, reduce noise, and make it compatible with machine learning algorithms or statistical analysis. Effective data preprocessing is critical for obtaining meaningful insights and building accurate predictive models.

METHODS FOR TESTING WATER QUALITY FIELD TESTING

On-site assessments enable immediate evaluation of water quality indicators using portable instruments. This method provides quick results and is commonly used for routine monitoring.

LABORATORY ANALYSIS This approach involves sophisticated equipment to measure a wide range of water quality parameters accurately. Samples are collected and analyzed in controlled settings to obtain comprehensive and reliable results.

BIOLOGICAL MONITORING By observing and analyzing the presence of indicator species and their health in aquatic ecosystems, scientists can assess water quality and detect potential issues, such as nutrient pollution or habitat degradation.

PARAMETERS MEASURED IN WATER QUALITY ANALYSIS

- pH levels: acidity or alkalinity of water
- Turbidity: clarity of water caused by suspended particles

- Dissolved oxygen: availability of oxygen for aquatic organisms
- Temperature: influences physical and biological processes
- Total dissolved solids: measure of inorganic substancesin water
- Nutrient concentrations: levels of nitrogen and phosphorus
- Microbial contamination: presence of harmful bacteria and viruses.

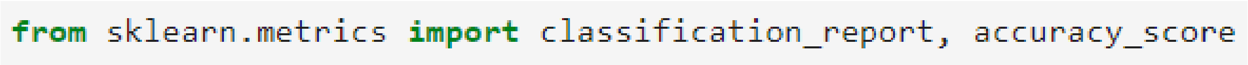
DATA SAVING

Data saving involves storing and managing datasets that are used for training, testing, and validating machine learning models. This process includes collecting, preprocessing, and archiving data in a structured format. Common data-saving formats for machine learning include CSV, JSON, or specialized formats like HDF5. Proper data saving is crucial for reproducibility, as it allows researchers and practitioners to access the same data for model development and evaluation. Additionally, version control systems may be used to track changes to datasets, ensuring that data remains consistent and traceable throughout the machine learning project.

MACHINE LEARNING ALGORITHMS :-

- Machine learning algorithms play a crucial role in data analysis by enabling automated data modeling, pattern recognition, and predictive analytics.Machine learning algorithms enhance data analysis by automating complex tasks, uncovering hidden patterns, and providing data-driven insights.

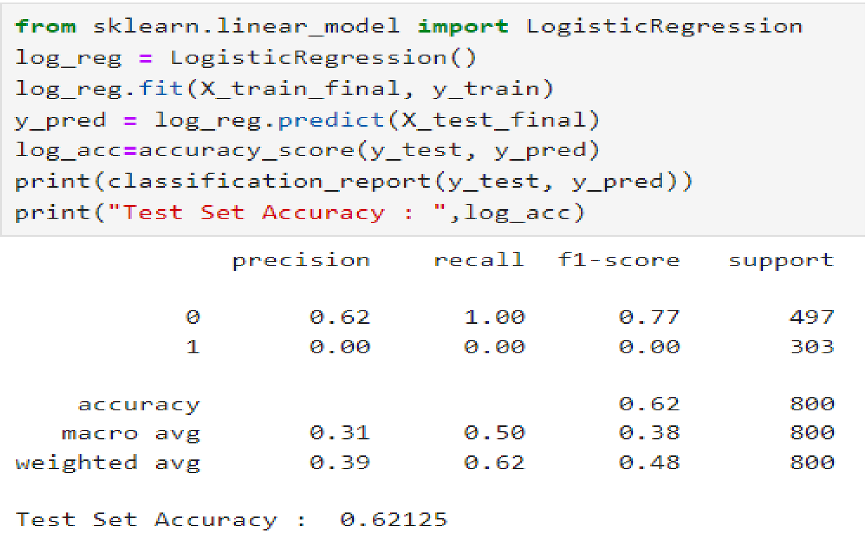
LIBRARY :-



```
from sklearn.metrics import classification_report, accuracy_score
```

LOGISTIC REGRESSION :-

Logistic Regression is a commonly used algorithm for binary and multiclass classification problems.It is a fundamental class algorithm that models the probability of class membership using the sigmoid function. It estimates parameters to create a decision boundary that separates data points into different classes.Accuracy is used to evaluate the model's performance by comparing its predictions to actual class labels.



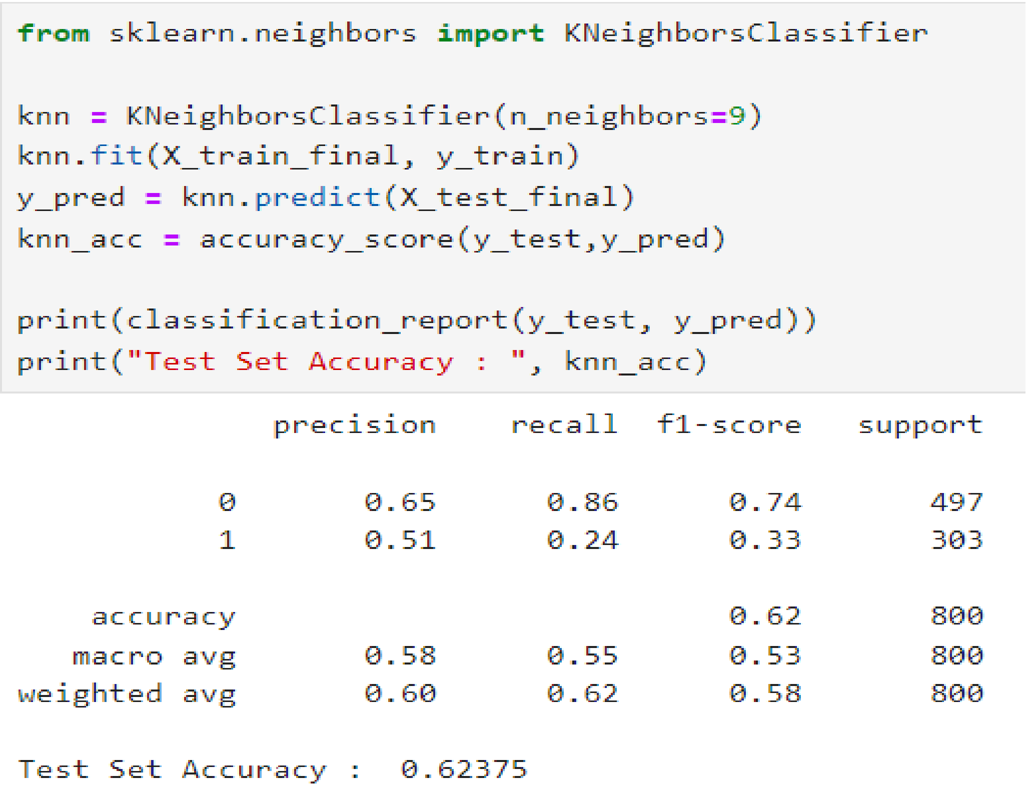
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train_final, y_train)
y_pred = log_reg.predict(X_test_final)
log_acc=accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
print("Test Set Accuracy : ",log_acc)
```

```
              precision    recall  f1-score   support

           0       0.62      1.00      0.77       497
           1       0.00      0.00      0.00       303

    accuracy                           0.62       800
   macro avg       0.31      0.50      0.38       800
weighted avg       0.39      0.62      0.48       800

Test Set Accuracy :  0.62125
```

The Test Accuracy of Logistic Regression is 62%

K-NEAREST NEIGHBOR CLASSFIER :-

KNN is a simple yet effective supervised machine learning algorithm used for both classification and regression tasks. It operates on the principle of similarity and is based on the idea that data points with similar features are more likely to belong to the same class or have similar target values. KNN is a straightforward algorithm that relies on the concept of similarity to classify or predict data points. It is non-parametric and lazy (as it doesn't build a model during training), making it suitable for various tasks.



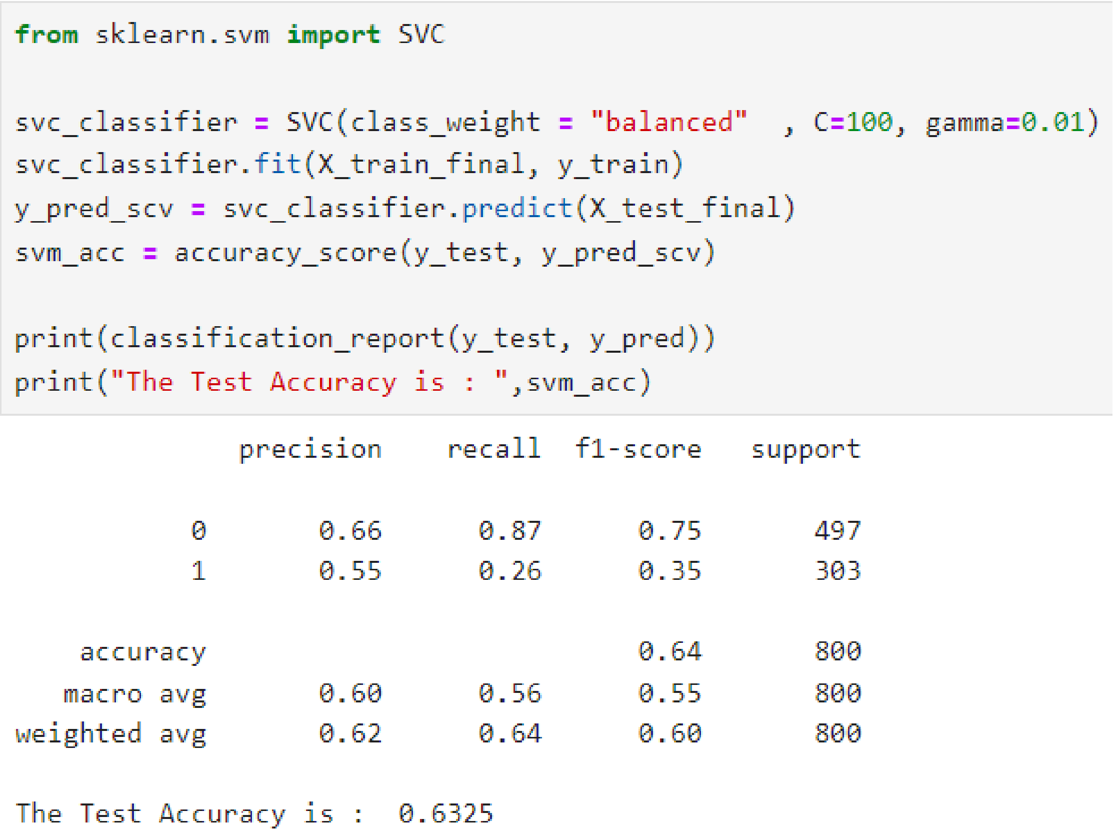
```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train_final, y_train)
y_pred = knn.predict(X_test_final)
knn_acc = accuracy_score(y_test,y_pred)

print(classification_report(y_test, y_pred))
print("Test Set Accuracy : ", knn_acc)
```

```
              precision    recall  f1-score   support

           0       0.65      0.86      0.74       497
           1       0.51      0.24      0.33       303

    accuracy                           0.62       800
   macro avg       0.58      0.55      0.53       800
weighted avg       0.60      0.62      0.58       800

Test Set Accuracy :  0.62375
```

The Test Accuracy of K-Nearest Neighbor is 62%

SUPPORT VECTOR CLASSIFIER :-

SVM is a powerful algorithm for classification and regression tasks that aims to find an optimal hyperplane to separate different classes while maximizing the margin between them. It can handle both linear and nonlinear data, and its performance is evaluated using accuracy metrics on training and test datasets.



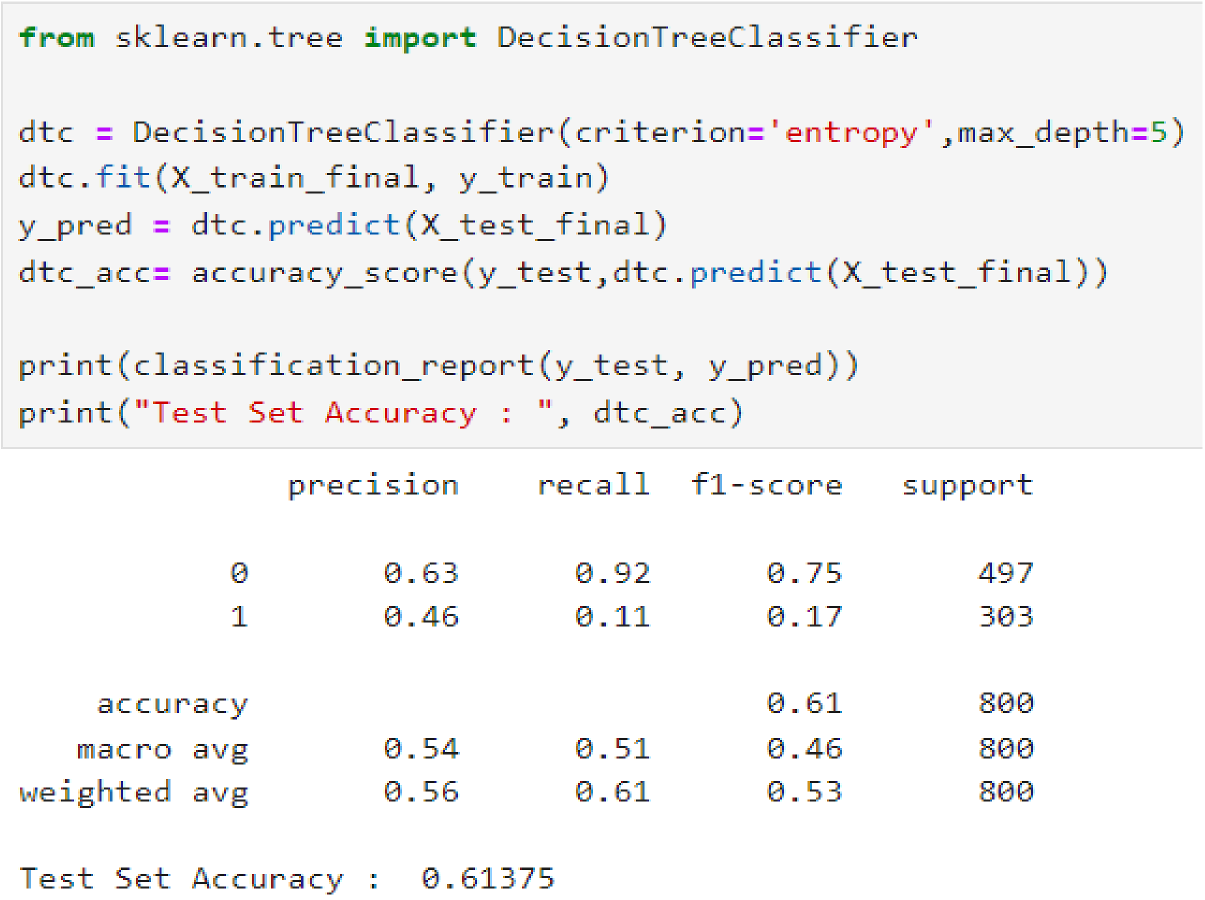
```
from sklearn.svm import SVC

svc_classifier = SVC(class_weight = "balanced"  , C=100, gamma=0.01)
svc_classifier.fit(X_train_final, y_train)
y_pred_scv = svc_classifier.predict(X_test_final)
svm_acc = accuracy_score(y_test, y_pred_scv)

print(classification_report(y_test, y_pred))
print("The Test Accuracy is : ",svm_acc)
```

```
              precision    recall  f1-score   support

           0       0.66      0.87      0.75       497
           1       0.55      0.26      0.35       303

    accuracy                           0.64       800
   macro avg       0.60      0.56      0.55       800
weighted avg       0.62      0.64      0.60       800

The Test Accuracy is :  0.6325
```

The Test Accuracy of Support Vector Classifier is 63%

DECISION TREE CLASSIFIER :-

A Decision Tree is a machine learning algorithm used for both classification and regression tasks. It builds a tree-like structure of decisions based on feature values to classify data. It makes use of impurity measures like entropy to determine the best feature splits. By controlling the tree's depth and evaluating its accuracy, one can create a model that balances between fitting the training data well and generalizing to new data.



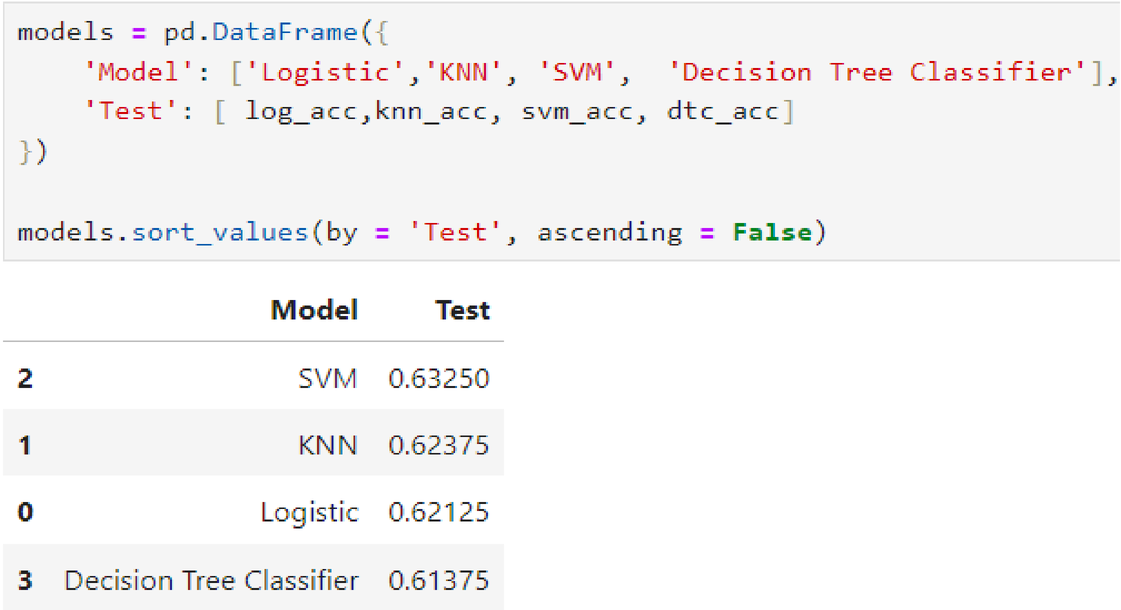
```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(criterion='entropy',max_depth=5)
dtc.fit(X_train_final, y_train)
y_pred = dtc.predict(X_test_final)
dtc_acc= accuracy_score(y_test,dtc.predict(X_test_final))

print(classification_report(y_test, y_pred))
print("Test Set Accuracy : ", dtc_acc)
```

```
              precision    recall  f1-score   support

           0       0.63      0.92      0.75       497
           1       0.46      0.11      0.17       303

    accuracy                           0.61       800
   macro avg       0.54      0.51      0.46       800
weighted avg       0.56      0.61      0.53       800

Test Set Accuracy :  0.61375
```

The Test Accuracy of Decision Tree Classifier is 61%

ALGORITHM ANALYSIS :-

This code facilitates the comparison of different machine learning models by recording and presenting their training and test accuracy in a structured table, sorted by test accuracy for easy assessment.



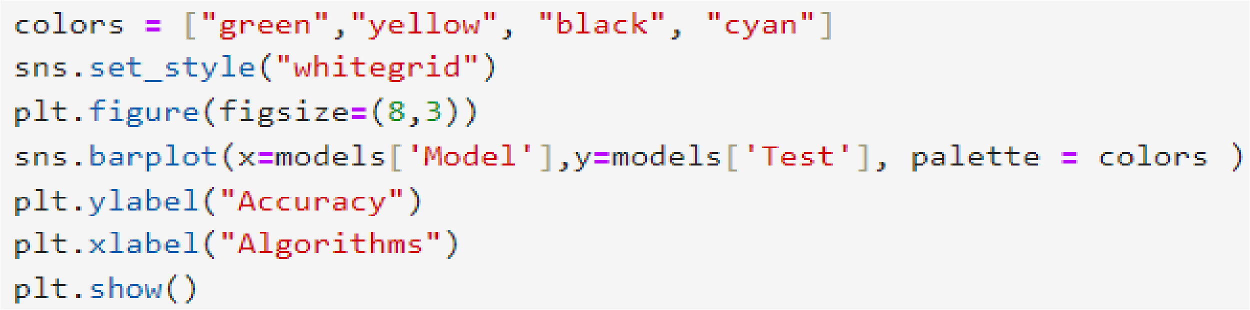
```
models = pd.DataFrame({
    'Model': ['Logistic','KNN', 'SVM',  'Decision Tree Classifier'],
    'Test': [ log_acc,knn_acc, svm_acc, dtc_acc]
})

models.sort_values(by = 'Test', ascending = False)
```

	Model	Test
2	SVM	0.63250
1	KNN	0.62375
0	Logistic	0.62125
3	Decision Tree Classifier	0.61375

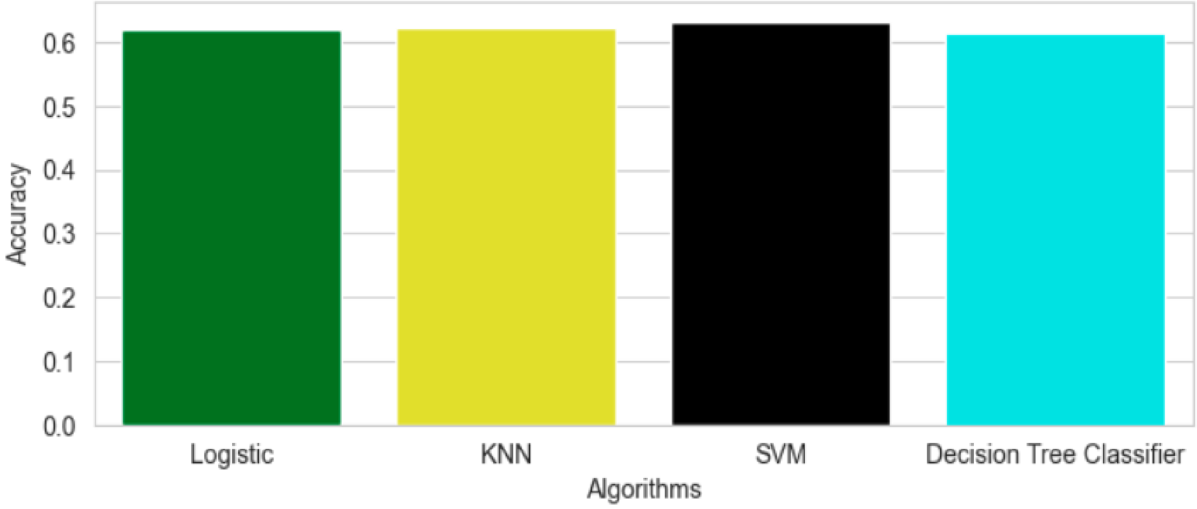
This code organizes the performance metrics of different machine learning models in a structured tabular format, sorts the models based on their test accuracy, and provides a clear comparison of how well each model performed on the test dataset. This is a common practice to help data analysts and machine learning practitioners choose the best model for a given task.

ALGORITHM VISUALIZATION :-

This code generates a bar plot using Seaborn to visually compare and present the test accuracy of different machine learning algorithms. The choice of colors, style, and figure size enhances the readability and presentation of the plot.



```
colors = ["green","yellow", "black", "cyan"]
sns.set_style("whitegrid")
plt.figure(figsize=(8,3))
sns.barplot(x=models['Model'],y=models['Test'], palette = colors )
plt.ylabel("Accuracy")
plt.xlabel("Algorithms")
plt.show()
```



The accuracy score for support vector machine is 63%. As compare with other models the accuracy score is much higher in support vector machine.

IMPACTS OF POOR WATER QUALITY

- Public Health Risks

  Poor water quality can lead to waterborne diseases, including diarrhoea, cholera, and typhoid, posing a significant threat to human health globally.

- Environmental Degradation

  Contaminated water adversely affects aquatic ecosystems, leading to the decline of fish populations, loss of biodiversity, and disruption of natural processes.

- Economic Consequences

Water pollution can impact industries, tourism, and agriculture, resulting in financial losses and hindering socio-economic development.

DATA COLLECTION

Data collection is a crucial phase in machine learning that involves gathering the necessary information or datasets to train, test, and validate machine learning models. Here are some key points about data collection in machine learning:

1. Data Sources: Data can be collected from various sources, such as databases, APIs, sensors, web scraping, surveys, or

existing datasets. The choice of data source depends on the specific problem and the type of data required.

2. Data Quality: Ensuring data quality is essential. This includes addressing issues like missing values, outliers, noise, and data inconsistencies. High-quality data is vital for training accurate and reliable models.

3. Data Annotation: In some cases, data may need to be manually annotated or labeled. This is common in supervised learning, where data points are assigned labels or categories.

4. Data Privacy and Ethics: Data collection should adhere to ethical standards and data privacy regulations. It's important to obtain consent and protect sensitive information when collecting data from individuals.

5. Data Volume: The amount of data needed depends on the complexity of the problem and the type of machine learning algorithm. More complex models often require larger datasets.

6. Data Splitting: Data is typically divided into training, validation, and testing sets. This splitting allows for model training, hyperparameter tuning, and evaluation on unseen data.

7. Data Versioning: Keeping track of different versions of the dataset is important for reproducibility. Version control systems can help manage data changes.

8. Bias and Fairness: Care should be taken to address bias in the collected data, as biased data can lead to biased machine learning models. Efforts should be made to ensure fairness and inclusivity in data collection.

9. Data Storage: Data should be stored in a secure and organized manner, making it easily accessible for model development and analysis.

Data collection is a critical step that lays the foundation for the success of a machine learning project. The quality and relevance of the data gathered significantly impact the performance and generalizability of the resulting models.

Importance of Visualizing the Dataset :

The importance of visualizing datasets cannot be overstated in the realm of data analysis. Visualization serves as a powerful bridge between raw data and human comprehension. It enhances our ability to understand, interpret, and extract meaningful insights from complex datasets.

By transforming numbers and statistics into charts, graphs, and interactive displays, visualization offers several key advantages. Firstly, it enables us to detect patterns, trends, and outliers that might remain hidden in tabular data, facilitating more accurate and timely decision-making. Moreover, it supports data exploration by allowing users to interact with the data, making it easier to uncover specific details and refine analysis.

This feature is particularly valuable in the age of big data, where sifting through vast datasets can be a formidable challenge. It is a time-saving tool that provides a rapid overview of data, streamlining the analysis process.

Visualizing the dataset :

Visualizing a dataset in Python is the process of using data visualization libraries like Matplotlib, Seaborn, or Plotly to create graphical representations of data. The goal is to gain insights, identify patterns, and present data in a visual format that is easy to understand.

Visualizing a dataset in Cognos is a fundamental aspect of data analysis and interpretation. It involves creating graphical representations of data to uncover patterns, relationships, and insights, making it an essential tool in data-driven decision-making and storytelling.

Identify The Dataset:

The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

Load The Dataset:

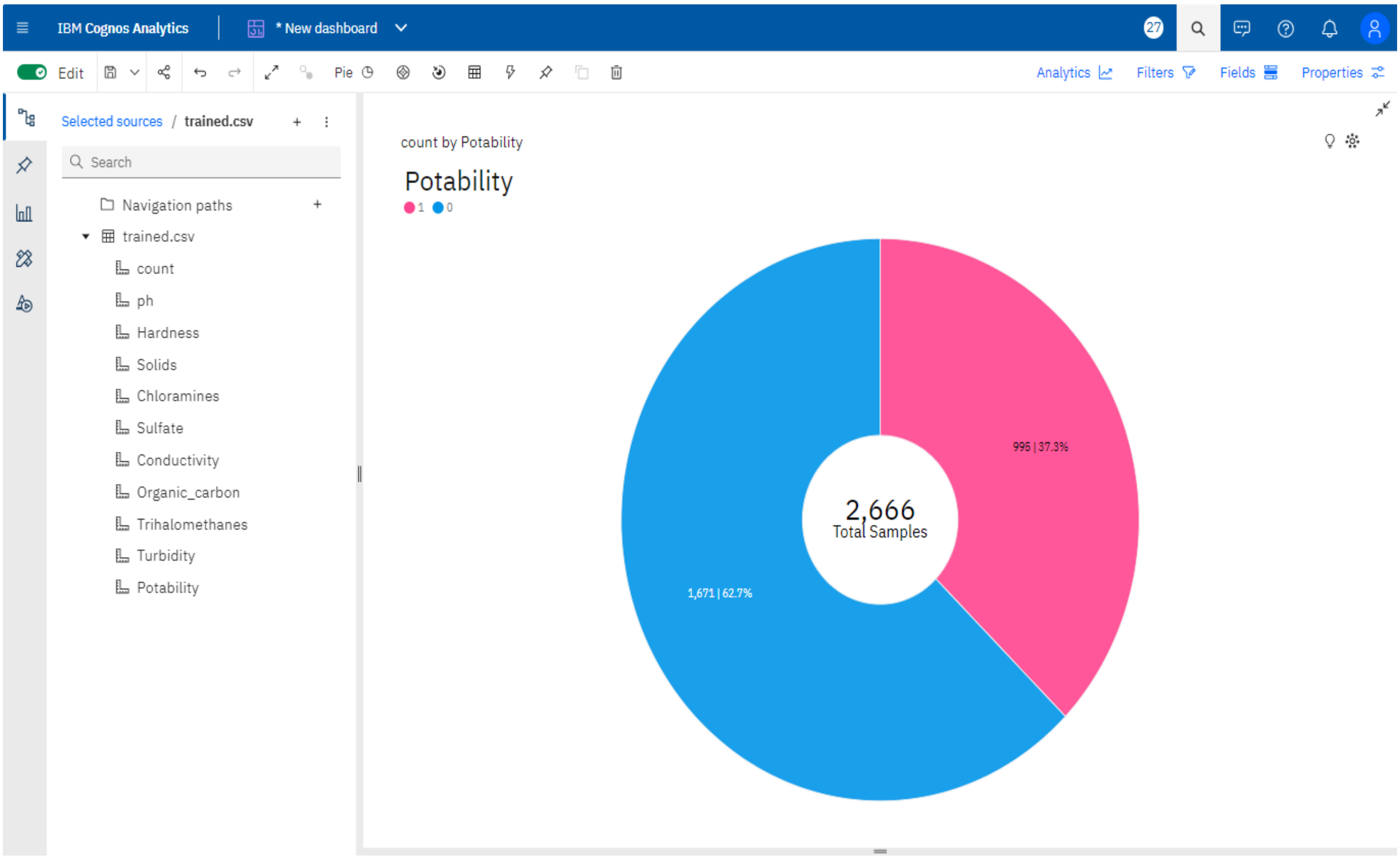
Once you have identified the dataset, you need to load it into the IBM Cognos Analytics. This may involve using a built-in function in the machine learning library, or it may involve writing our own code.

Preprocess The Dataset:

Once the dataset is loaded into the IBM Cognos Analytics Environment, you may need to think a logic before you can start visualizing the dataset.

Visualization of the dataset Using Cognos

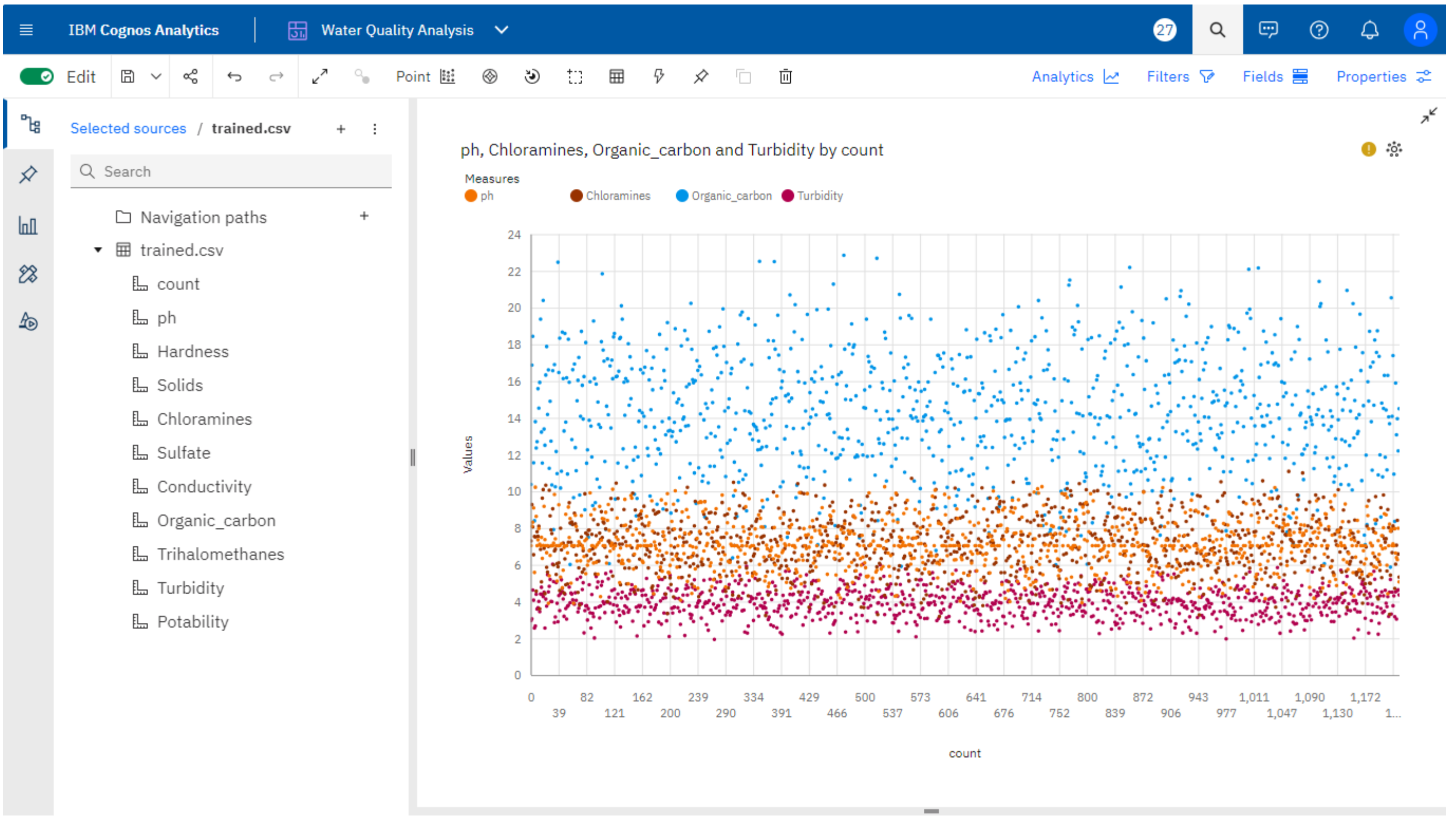
Visualization using Pie Chart :



Visualizing the Potability data using Pie Chart provided in the Cognos.

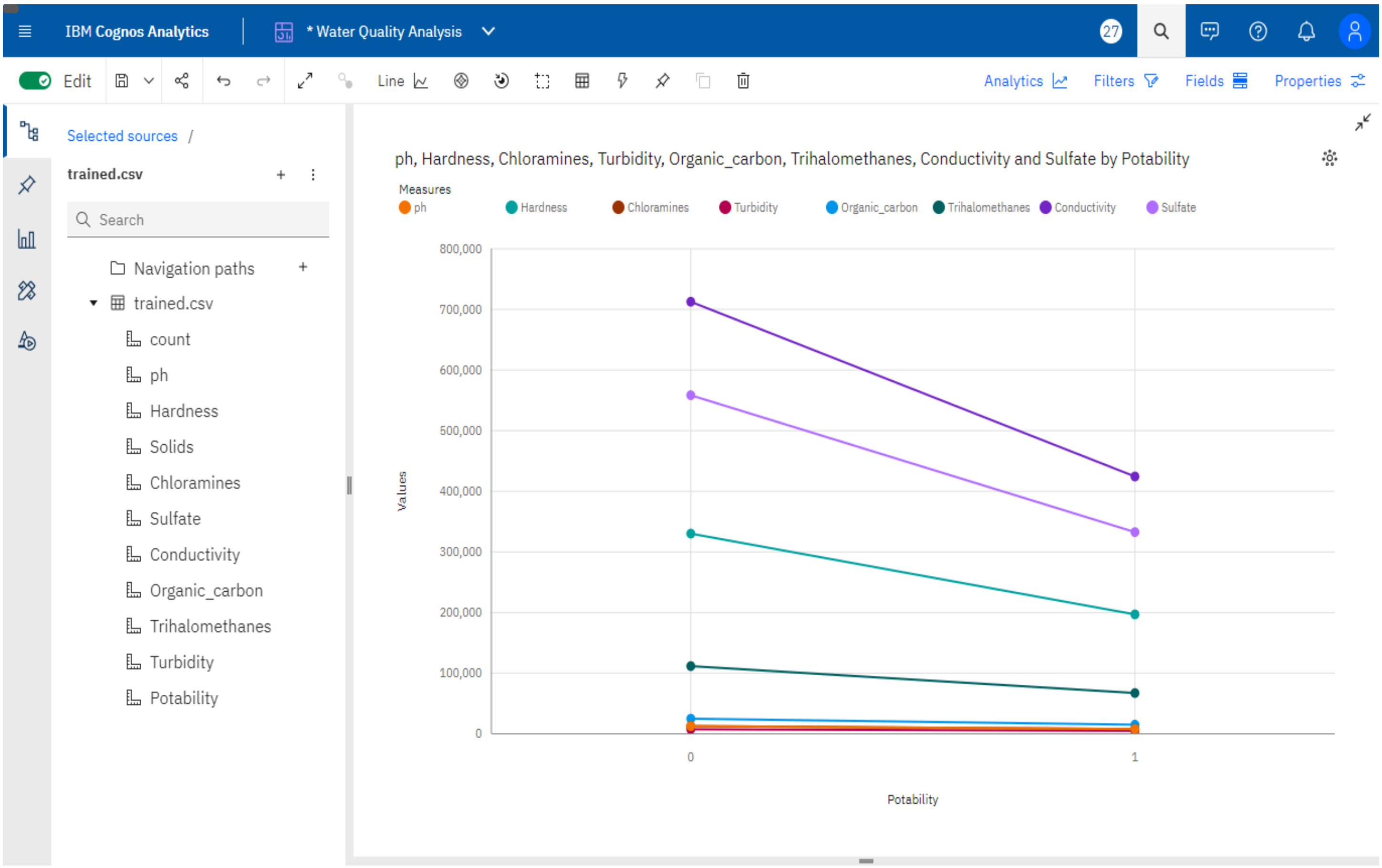
Here the insights of the Pie chart is the dataset contains 62.7% Not Potable and 37.3% Potable datas among the 2666 Water Samples.

Visualizing using the Points :



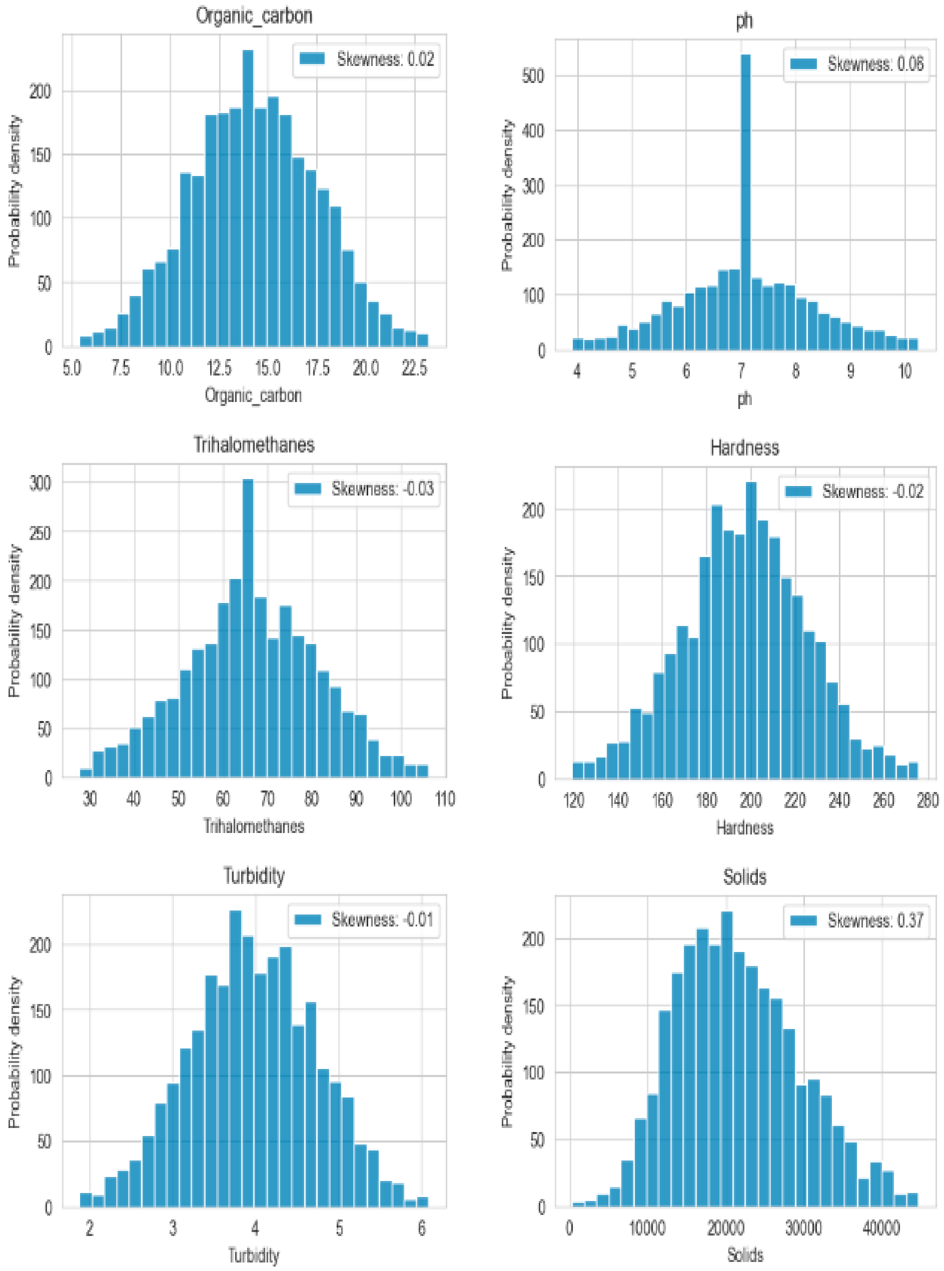
Visualizing the Distribution of each Water Quality Parameters provided in the Dataset using Point Chart provided in the Cognos.

Visualization using Line Chart :



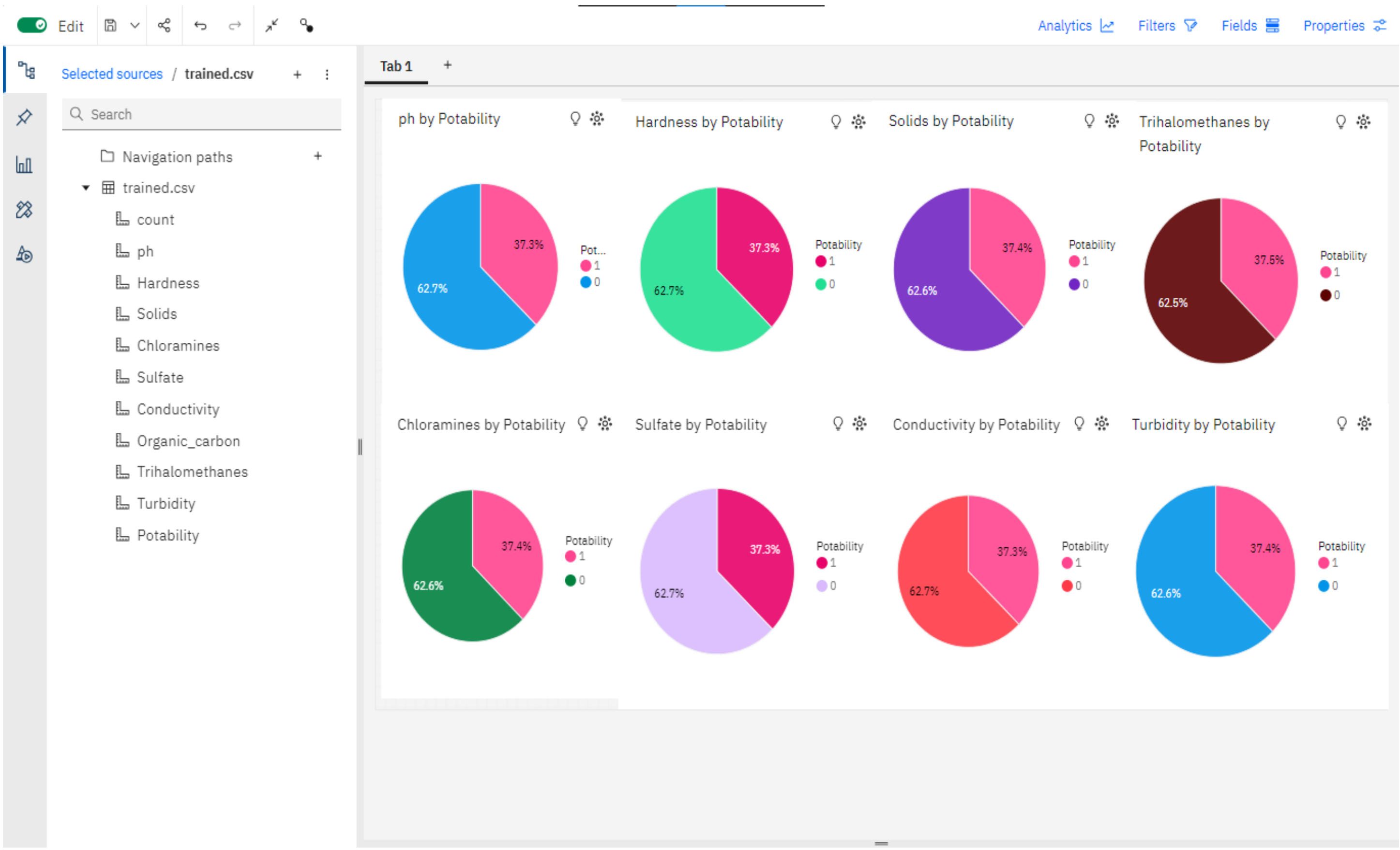
Visualizing the Potability of each water quality parameters using Line Chart provided in the Cognos.

Visualization using the Histplot :



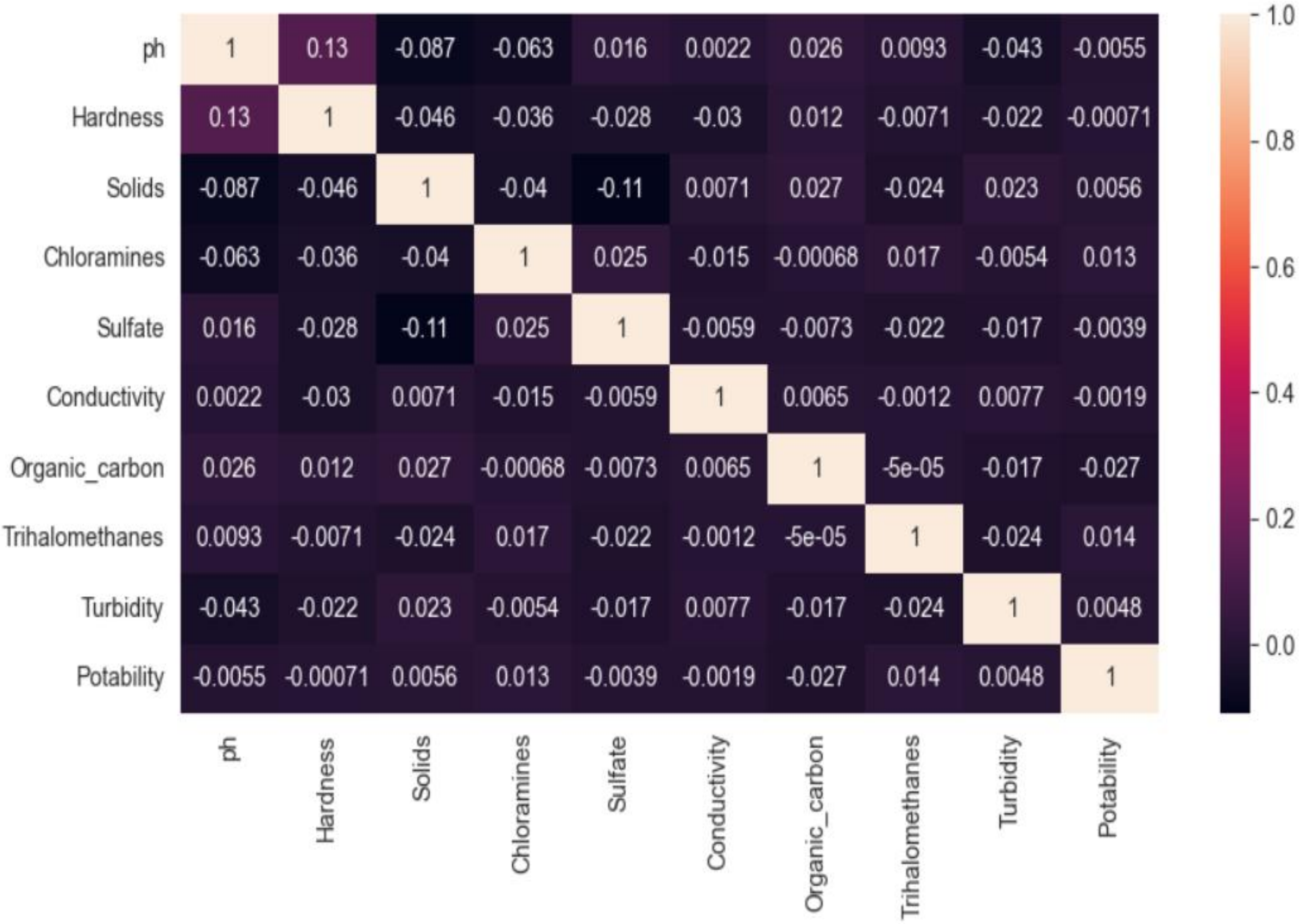
Visualizing the Potability Density of each water quality parameters using Histplot provided in the Cognos.

Visualization using the Pie Chart :



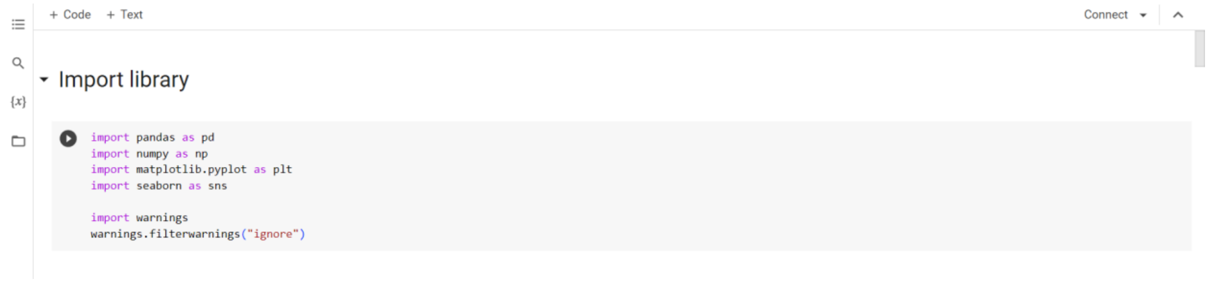
Visualizing the Potability of each water quality parameters using Pie Chart provided in the Cognos.

Visualizing Correaltion :



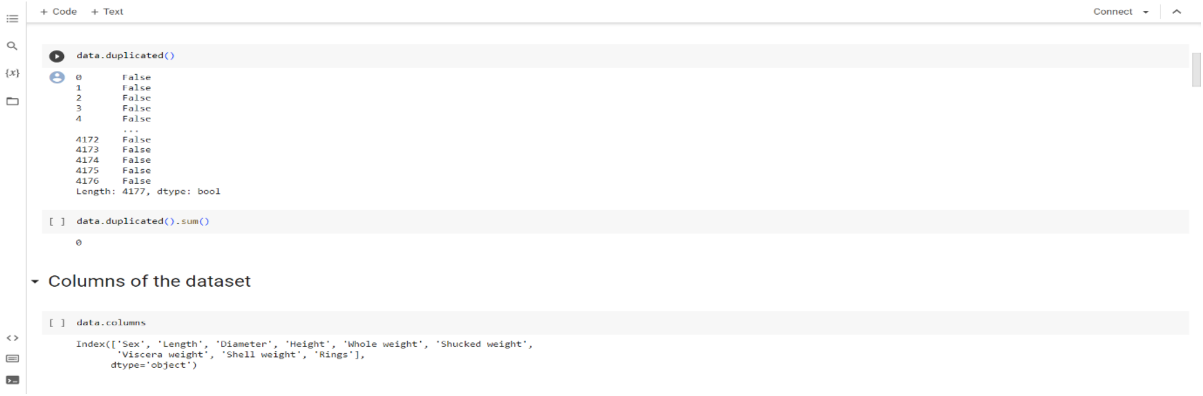
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1	0.13	-0.087	-0.063	0.016	0.0022	0.026	0.0093	-0.043	-0.0055
Hardness	0.13	1	-0.046	-0.036	-0.028	-0.03	0.012	-0.0071	-0.022	-0.00071
Solids	-0.087	-0.046	1	-0.04	-0.11	0.0071	0.027	-0.024	0.023	0.0056
Chloramines	-0.063	-0.036	-0.04	1	0.025	-0.015	-0.00068	0.017	-0.0054	0.013
Sulfate	0.016	-0.028	-0.11	0.025	1	-0.0059	-0.0073	-0.022	-0.017	-0.0039
Conductivity	0.0022	-0.03	0.0071	-0.015	-0.0059	1	0.0065	-0.0012	0.0077	-0.0019
Organic_carbon	0.026	0.012	0.027	-0.00068	-0.0073	0.0065	1	-5e-05	-0.017	-0.027
Trihalomethanes	0.0093	-0.0071	-0.024	0.017	-0.022	-0.0012	-5e-05	1	-0.024	0.014
Turbidity	-0.043	-0.022	0.023	-0.0054	-0.017	0.0077	-0.017	-0.024	1	0.0048
Potability	-0.0055	-0.00071	0.0056	0.013	-0.0039	-0.0019	-0.027	0.014	0.0048	1

CODES:-



Import library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```



```
data.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...  
4172    False
4173    False
4174    False
4175    False
4176    False
Length: 4177, dtype: bool
```

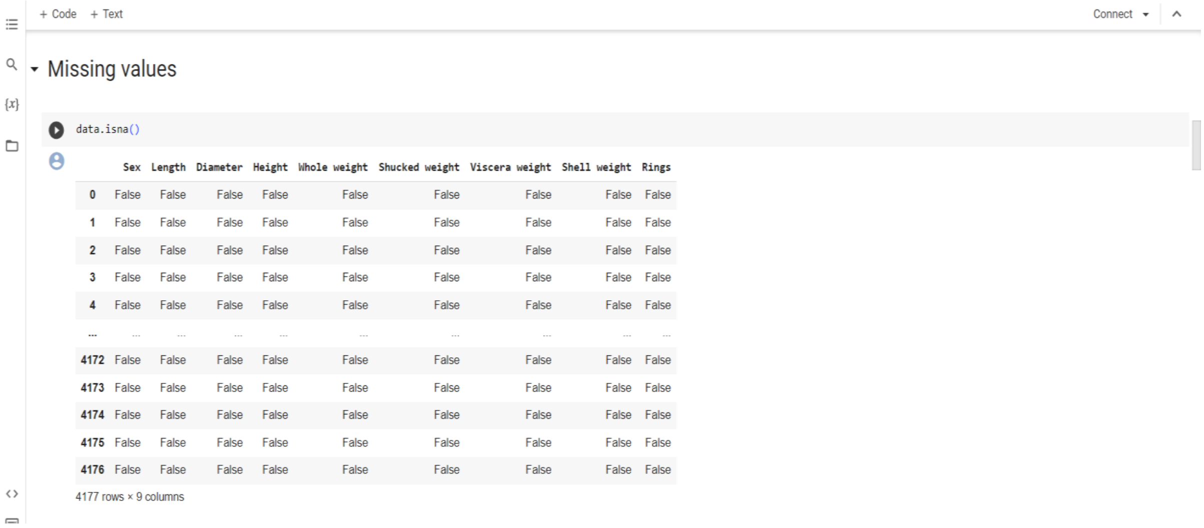
```
data.duplicated().sum()
```

```
0
```

Columns of the dataset

```
data.columns
```

```
Index(['Sex', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'Rings'],
      dtype='object')
```

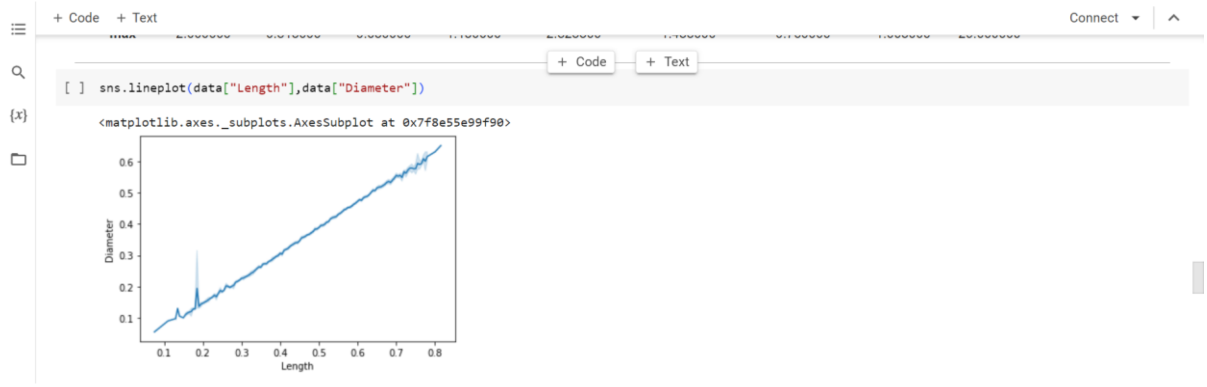


Missing values

```
data.isna()
```

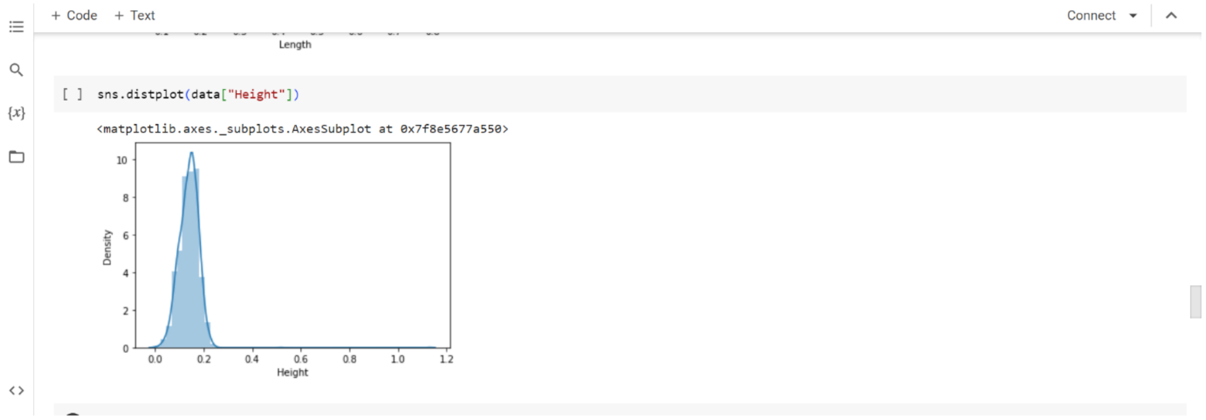
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
4172	False	False	False	False	False	False	False	False	False
4173	False	False	False	False	False	False	False	False	False
4174	False	False	False	False	False	False	False	False	False
4175	False	False	False	False	False	False	False	False	False
4176	False	False	False	False	False	False	False	False	False

4177 rows × 9 columns



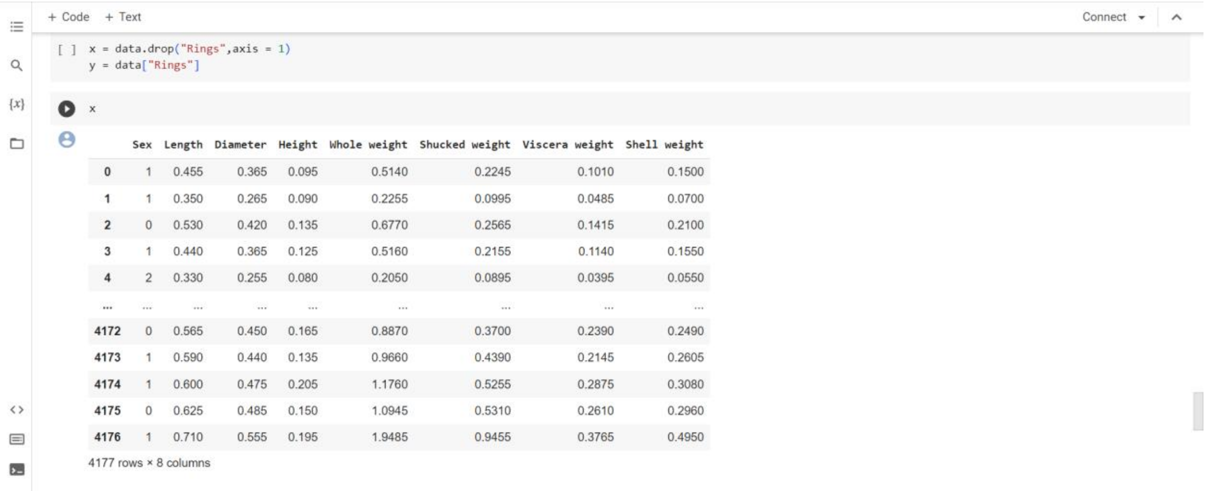
```
[ ] sns.lineplot(data["Length"],data["Diameter"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e55e99f90>
```



```
[ ] sns.distplot(data["Height"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e5677a550>
```

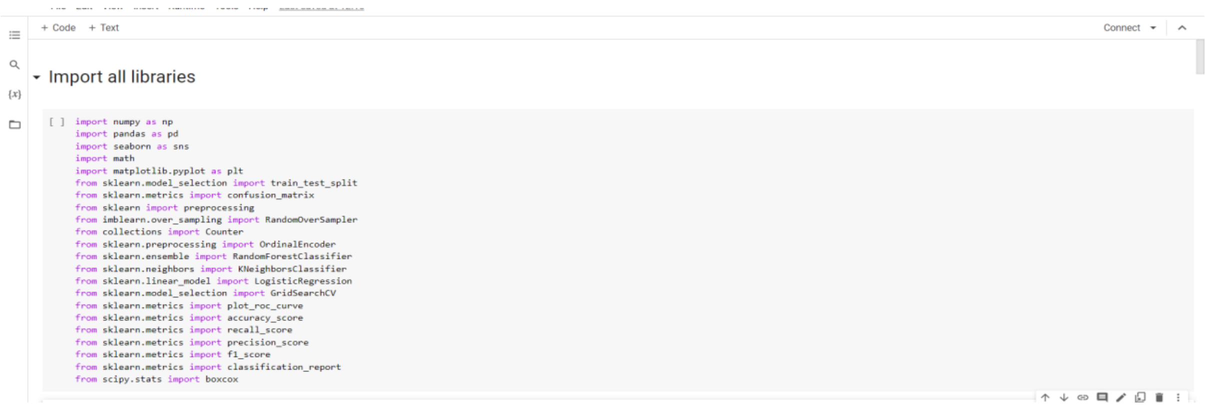


```
[ ] x = data.drop("Rings",axis = 1)
    y = data["Rings"]
```

```
x
```

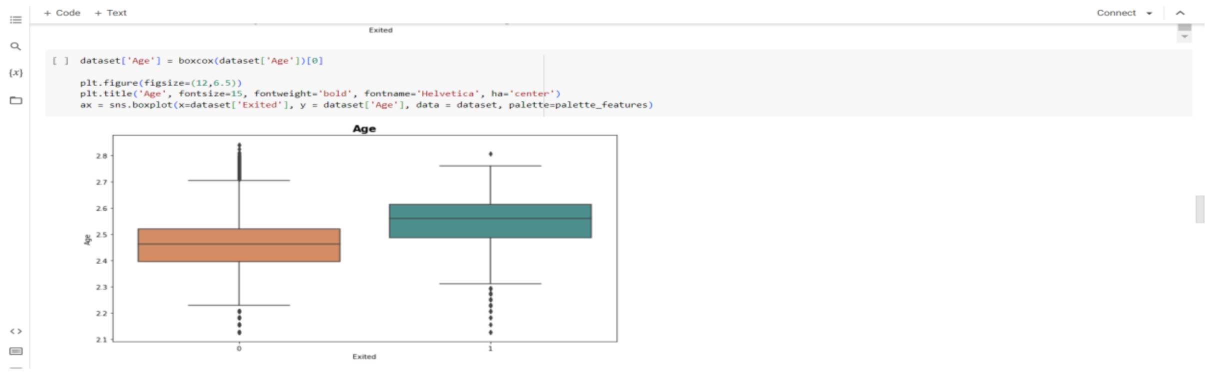
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550
...	...	...	...	...	...	...	...	...
4172	0	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490
4173	1	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605
4174	1	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080
4175	0	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960
4176	1	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950

4177 rows × 8 columns

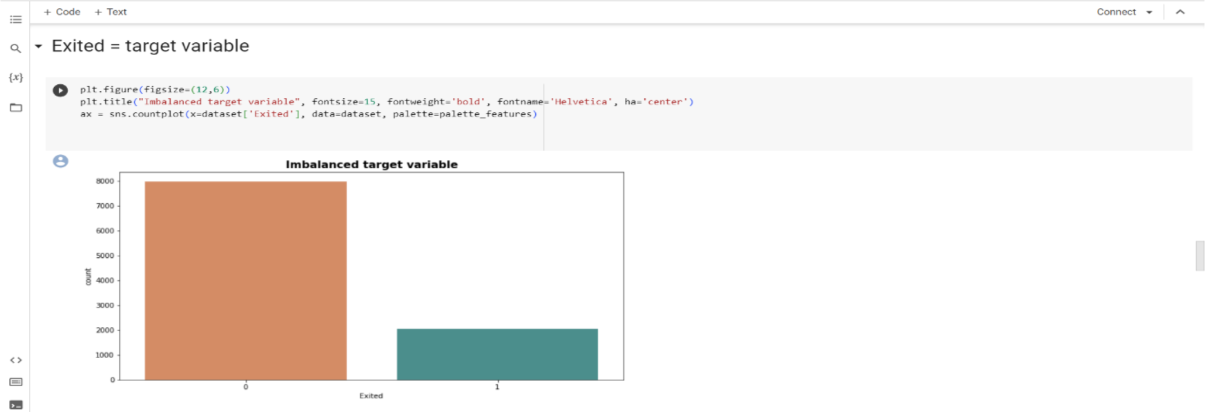


Import all libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import preprocessing
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from scipy.stats import boxcox
```

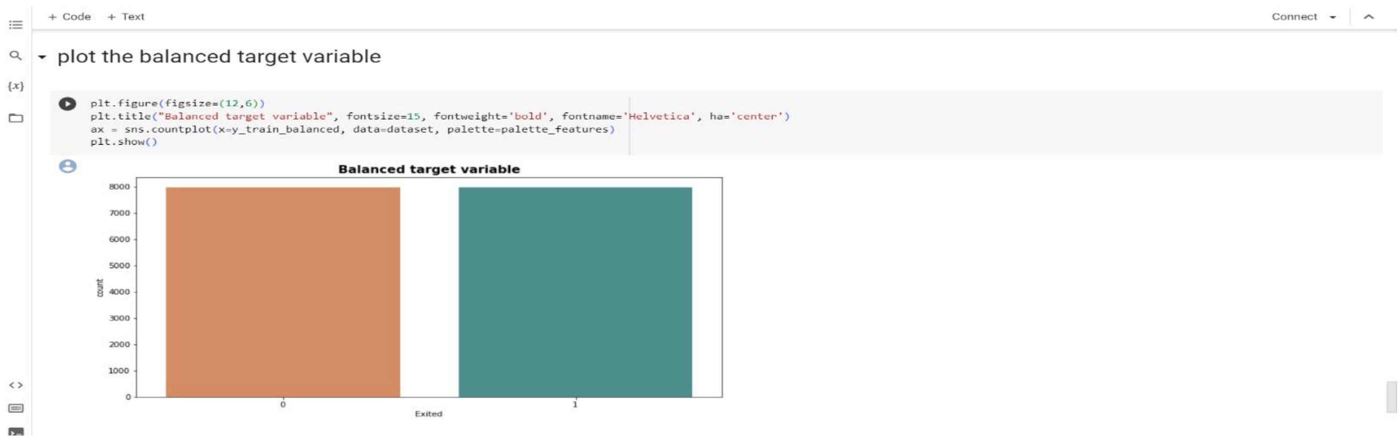


```
dataset['Age'] = boxcox(dataset['Age'])[0]

plt.figure(figsize=(12,6.5))
plt.title('Age', fontsize=15, fontweight='bold', fontname='Helvetica', ha='center')
ax = sns.boxplot(x=dataset['Exited'], y = dataset['Age'], data = dataset, palette=palette_features)
```



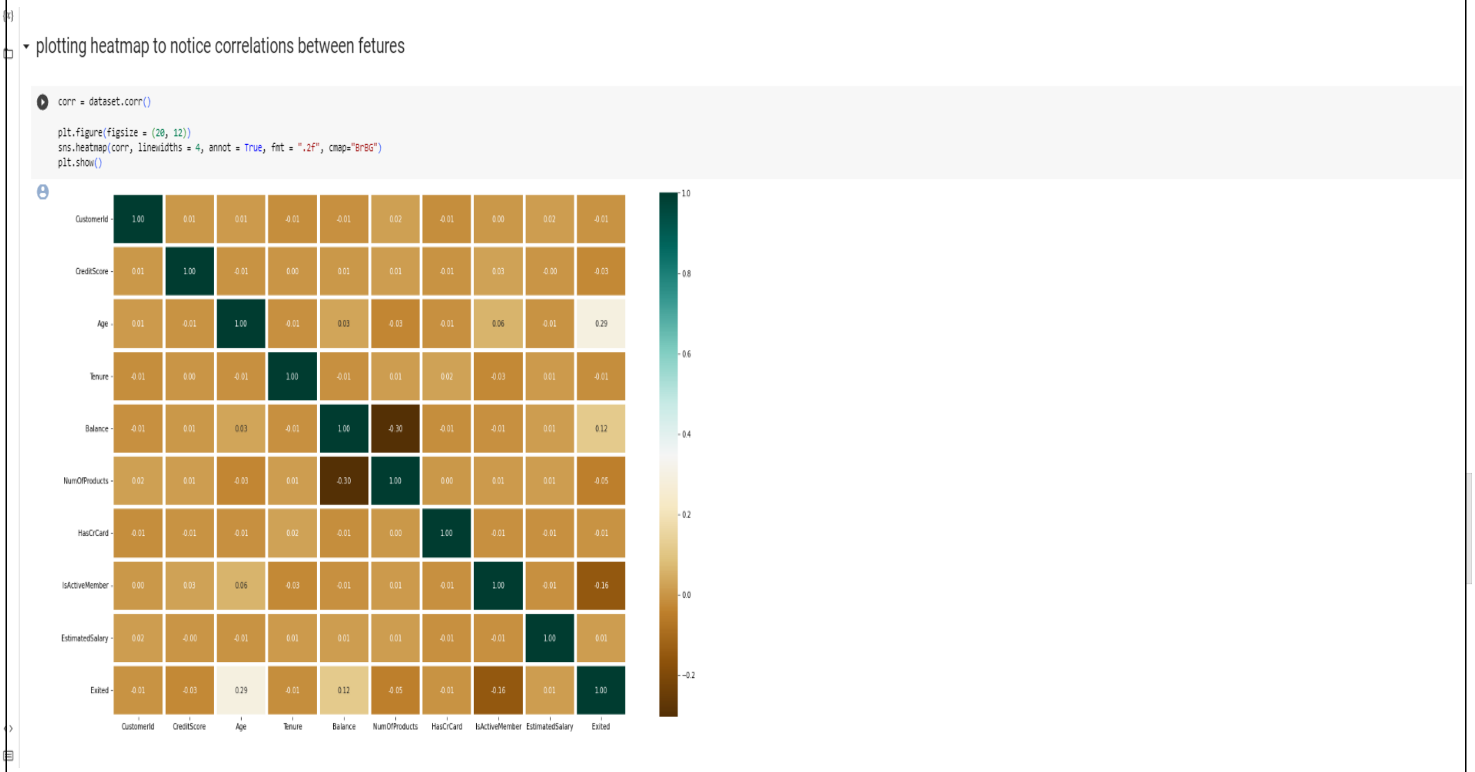
Exited = target variable

```
plt.figure(figsize=(12,6))
plt.title("Imbalanced target variable", fontsize=15, fontweight='bold', fontname='Helvetica', ha='center')
ax = sns.countplot(x=dataset['Exited'], data=dataset, palette=palette_features)
```



plot the balanced target variable

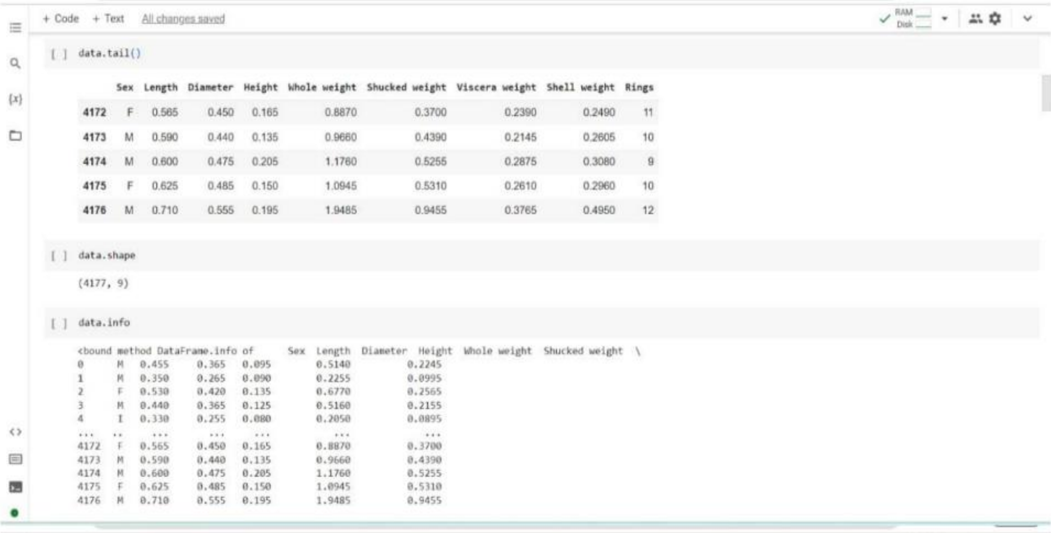
```
plt.figure(figsize=(12,6))
plt.title("Balanced target variable", fontsize=15, fontweight='bold', fontname='Helvetica', ha='center')
ax = sns.countplot(x=y_train_balanced, data=dataset, palette=palette_features)
plt.show()
```



plotting heatmap to notice correlations between fetures

```
corr = dataset.corr()

plt.figure(figsize = (20, 12))
sns.heatmap(corr, linewidths = 4, annot = True, fmt = ".2f", cmap="BrBG")
plt.show()
```

CustomerId 1.00 0.01 0.01 -0.01 -0.01 0.02 -0.01 0.00 0.02 -0.01
CreditScore 0.01 1.00 -0.01 0.00 0.01 0.01 -0.01 0.03 -0.00 -0.03
Age 0.01 -0.01 1.00 -0.01 0.03 -0.03 -0.01 0.06 -0.01 0.29
Tenure -0.01 0.00 -0.01 1.00 -0.01 0.01 0.02 -0.03 0.01 -0.01
Balance -0.01 0.01 0.03 -0.01 1.00 -0.30 -0.01 -0.01 0.01 0.12
NumOfProducts 0.02 0.01 -0.03 0.01 -0.30 1.00 0.00 0.01 0.01 -0.05
HasCrCard -0.01 -0.01 -0.01 0.02 -0.01 0.00 1.00 -0.01 -0.01 -0.01
IsActiveMember 0.00 0.03 0.06 -0.03 -0.01 0.01 -0.01 1.00 -0.01 -0.16
EstimatedSalary 0.02 -0.00 -0.01 0.01 0.01 0.01 -0.01 -0.01 1.00 0.01
Exited -0.01 -0.03 0.29 -0.01 0.12 -0.05 -0.01 -0.16 0.01 1.00
CustomerId
CreditScore
Age
Tenure
Balance
NumOfProducts
HasCrCard
IsActiveMember
EstimatedSalary
Exited
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



```
[ ] data.tail()
```

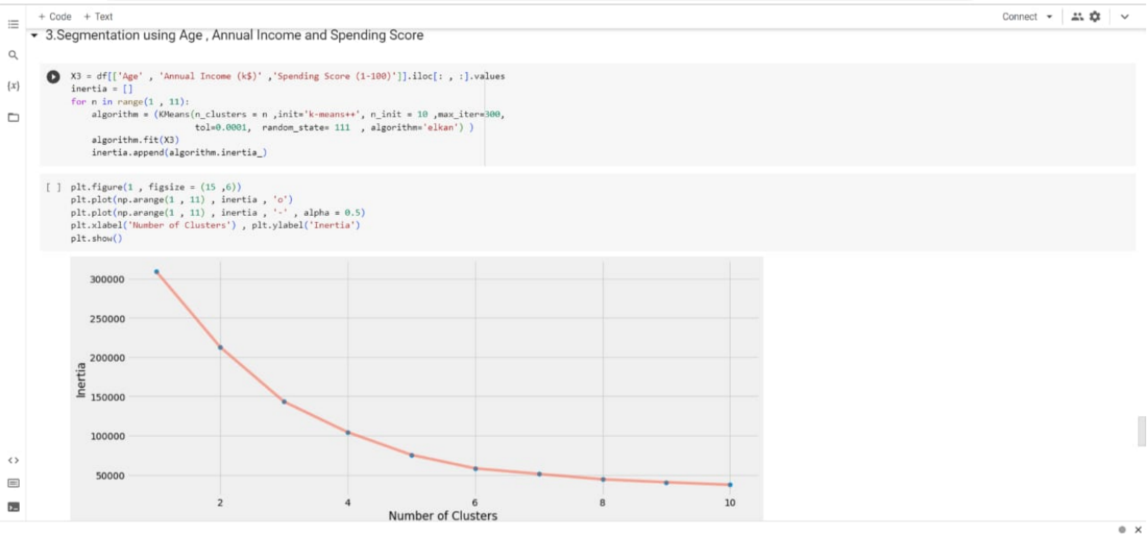
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

```
[ ] data.shape
```

```
(4177, 9)
```

```
[ ] data.info
```

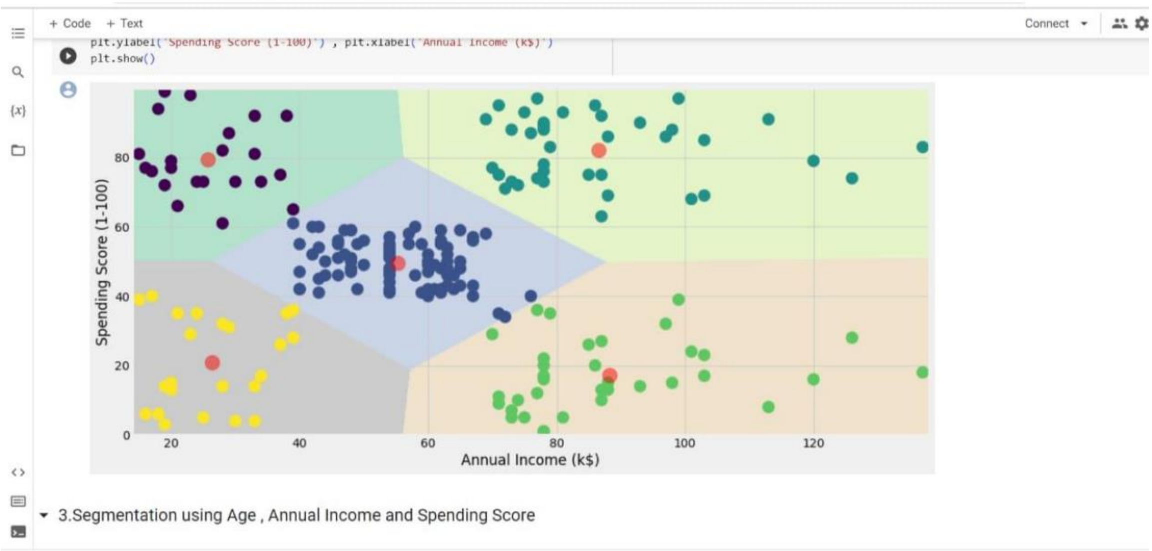
```
<bound method DataFrame.info of      Sex  Length  Diameter  Height  Whole weight  Shucked weight  \
0      M   0.455     0.365   0.095        0.5140          0.2245
1      M   0.350     0.265   0.090        0.2255          0.0995
2      F   0.530     0.420   0.135        0.6770          0.2565
3      M   0.440     0.365   0.125        0.5160          0.2155
4      I   0.330     0.255   0.080        0.2050          0.0895
...   ..     ...       ...     ...           ...             ...
4172   F   0.565     0.450   0.165        0.8870          0.3700
4173   M   0.590     0.440   0.135        0.9660          0.4390
4174   M   0.600     0.475   0.205        1.1760          0.5255
4175   F   0.625     0.485   0.150        1.0945          0.5310
4176   M   0.710     0.555   0.195        1.9485          0.9455
```



3.Segmentation using Age , Annual Income and Spending Score

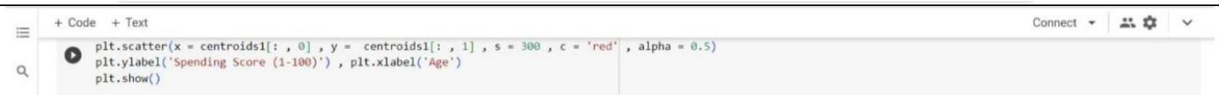
```
X3 = df[['Age' , 'Annual Income (k$)' ,'Spending Score (1-100)']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X3)
    inertia.append(algorithm.inertia_)
```

```
[ ] plt.figure(1 , figsize = (15 ,6))
    plt.plot(np.arange(1 , 11) , inertia , 'o')
    plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
    plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
    plt.show()
```

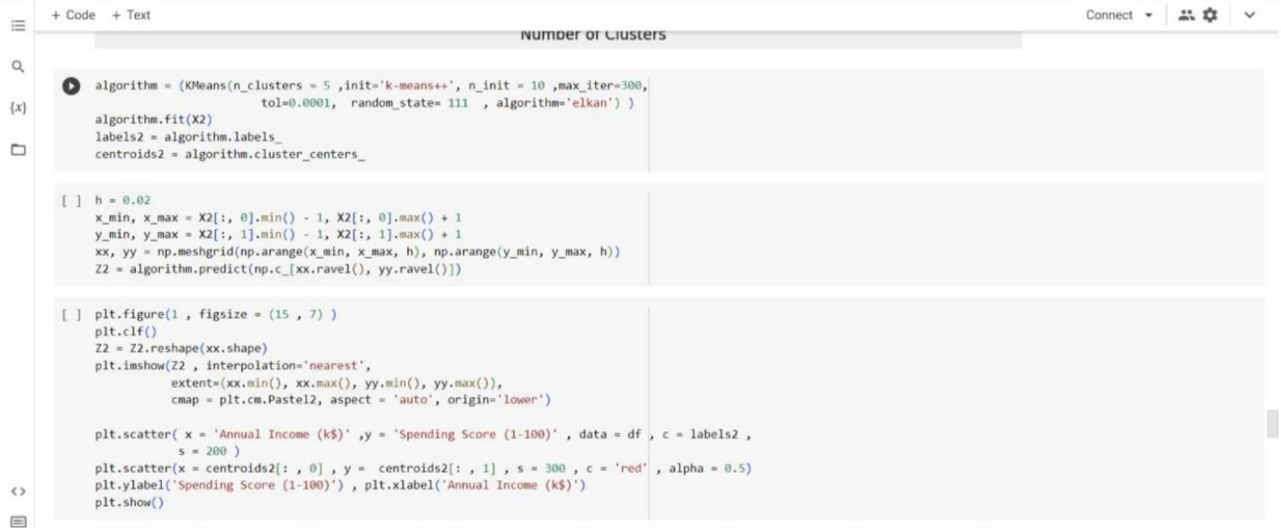


```
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Annual Income (k$)')
plt.show()
```

3.Segmentation using Age , Annual Income and Spending Score



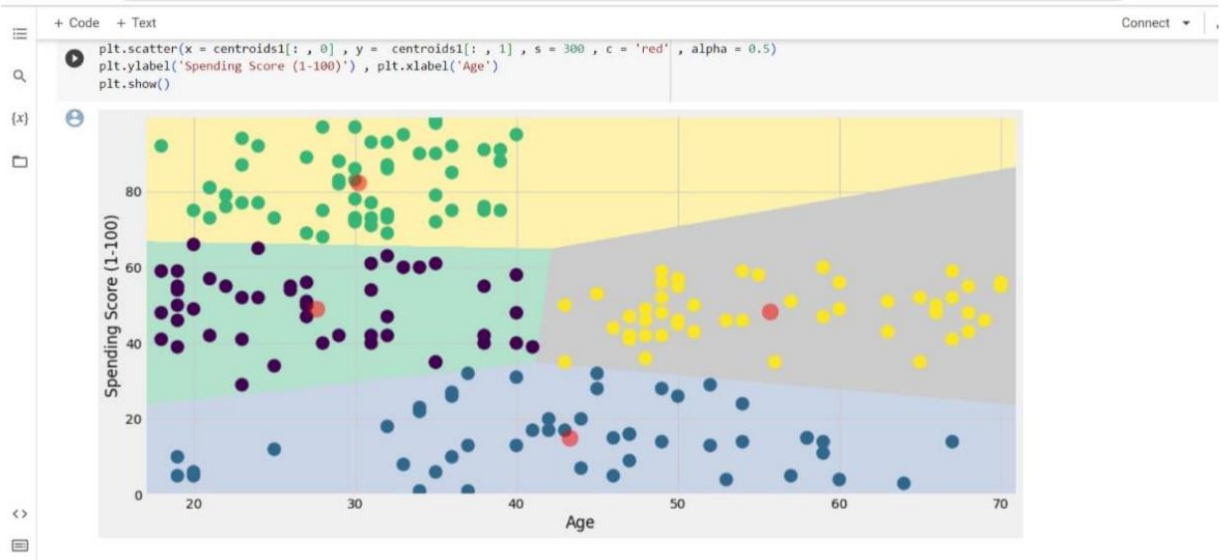
```
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
plt.show()
```



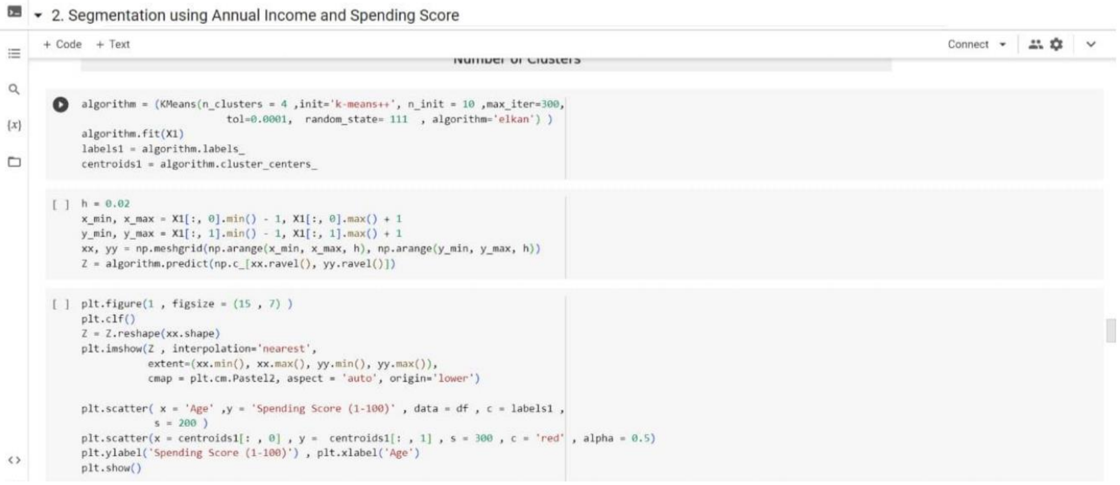
```
algorithm = (KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
algorithm.fit(X2)
labels2 = algorithm.labels_
centroids2 = algorithm.cluster_centers_
```

```
[ ] h = 0.02
    x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
    y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z2 = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
[ ] plt.figure(1 , figsize = (15 , 7) )
    plt.clf()
    Z2 = Z2.reshape(xx.shape)
    plt.imshow(Z2 , interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

    plt.scatter( x = 'Annual Income (k$)' ,y = 'Spending Score (1-100)' , data = df , c = labels2 ,
                s = 200 )
    plt.scatter(x = centroids2[: , 0] , y =  centroids2[: , 1] , s = 300 , c = 'red' , alpha = 0.5)
    plt.ylabel('Spending Score (1-100)') , plt.xlabel('Annual Income (k$)')
    plt.show()
```



```
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
plt.show()
```

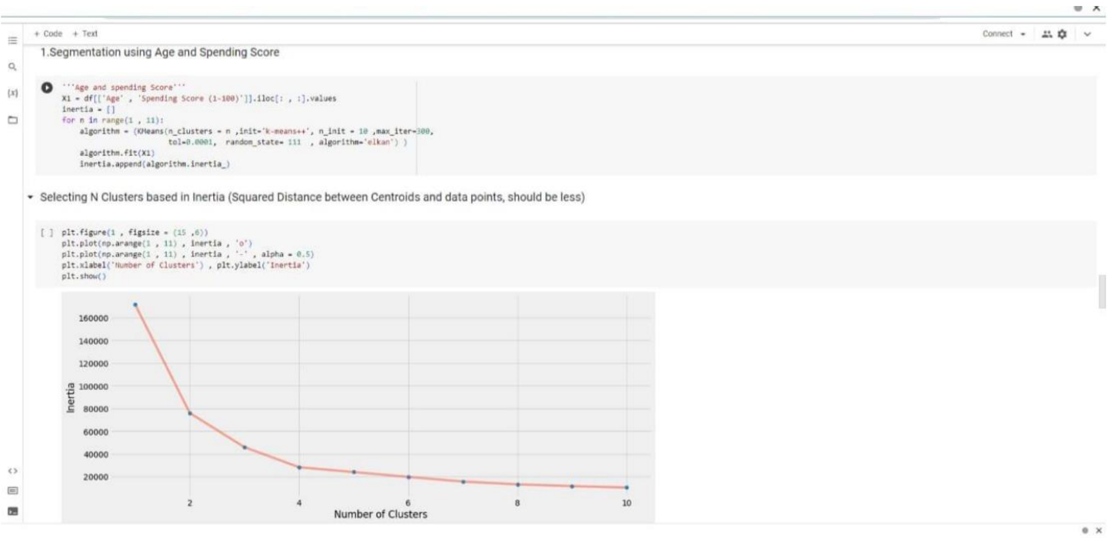


2. Segmentation using Annual Income and Spending Score

```
algorithm = (KMeans(n_clusters = 4 ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

```
[ ] h = 0.02
    x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
    y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
[ ] plt.figure(1 , figsize = (15 , 7) )
    plt.clf()
    Z = Z.reshape(xx.shape)
    plt.imshow(Z , interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

    plt.scatter( x = 'Age' ,y = 'Spending Score (1-100)' , data = df , c = labels1 ,
                s = 200 )
    plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c = 'red' , alpha = 0.5)
    plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
    plt.show()
```

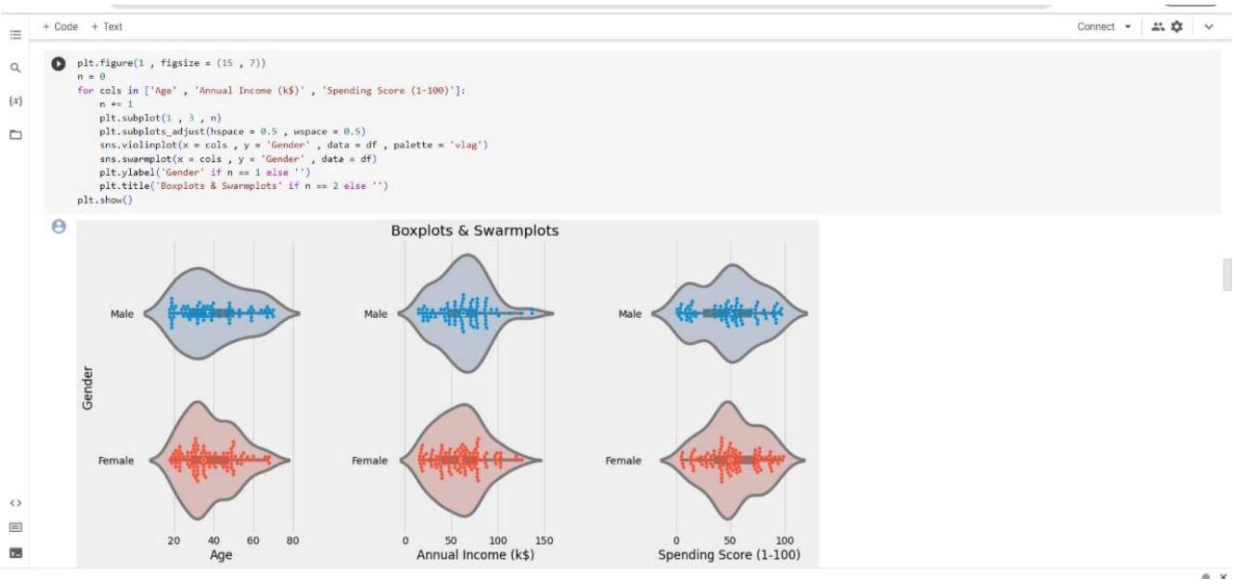


1.Segmentation using Age and Spending Score

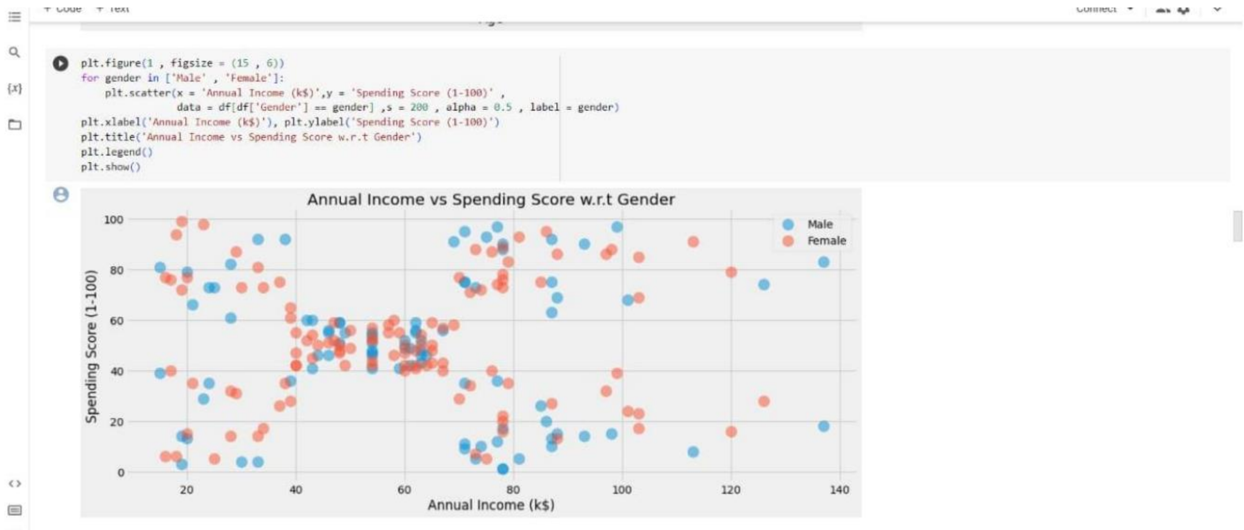
```
'''Age and spending Score'''
X1 = df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

Selecting N Clusters based in Inertia (Squared Distance between Centroids and data points, should be less)

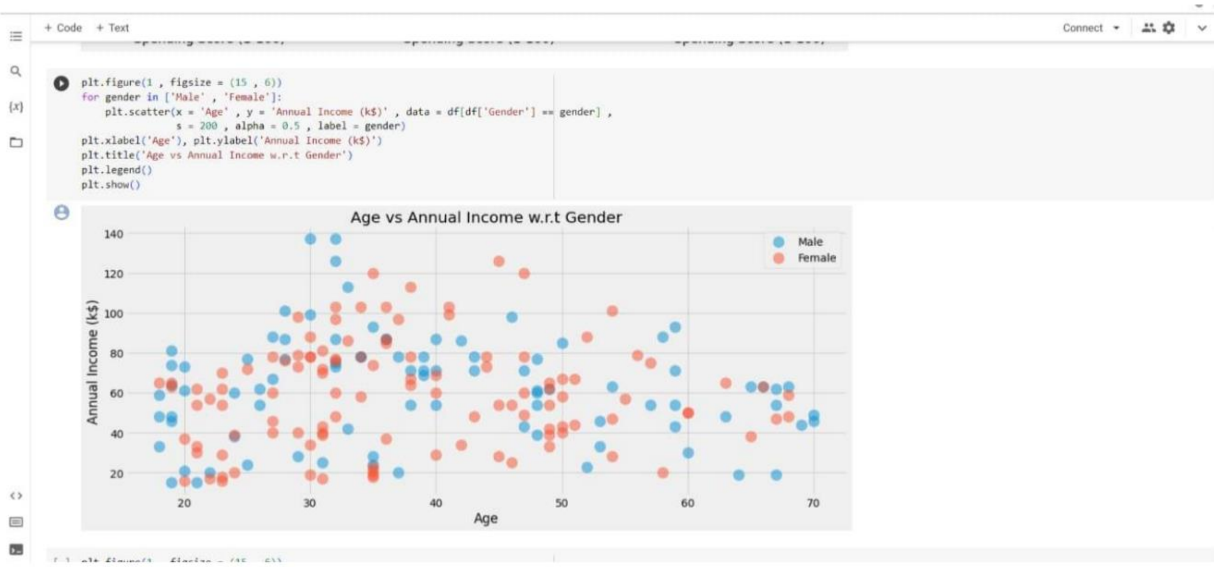
```
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```



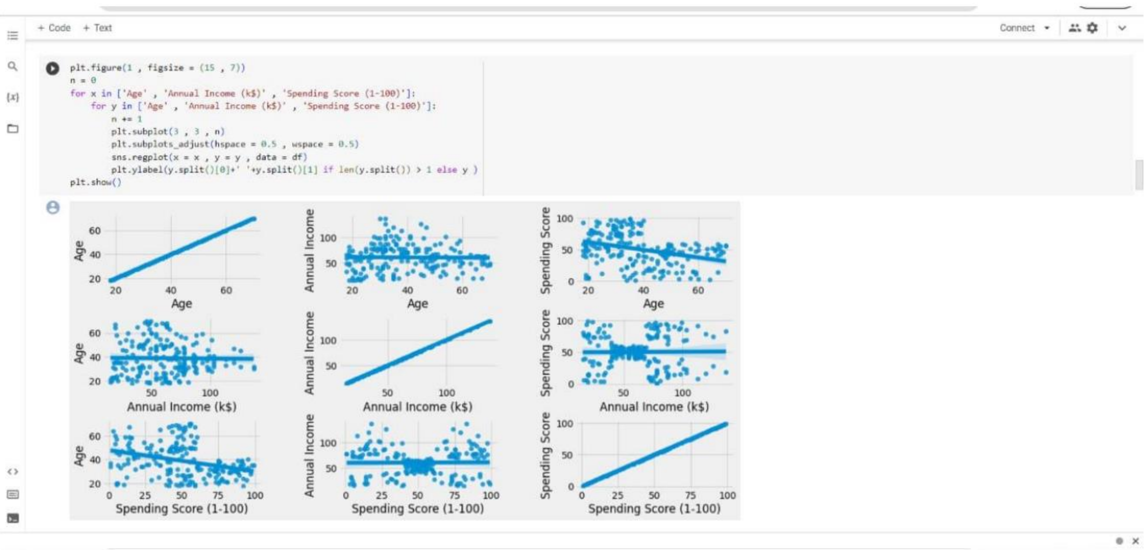
```
plt.figure(1 , figsize = (15 , 7))
n = 0
for cols in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
    sns.violinplot(x = cols , y = 'Gender' , data = df , palette = 'vlag')
    sns.swarmplot(x = cols , y = 'Gender' , data = df)
    plt.ylabel('Gender' if n == 1 else '')
    plt.title('Boxplots & Swarmplots' if n == 2 else '')
plt.show()
```



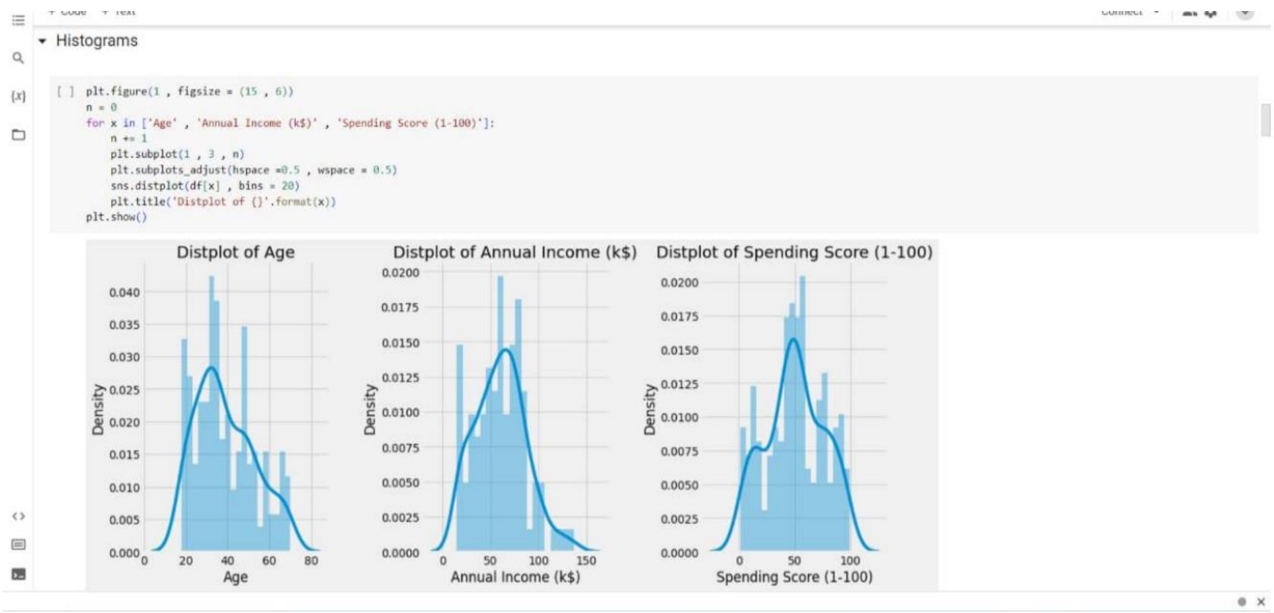
```
plt.figure(1 , figsize = (15 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Annual Income (k$)',y = 'Spending Score (1-100)' ,
                data = df[df['Gender'] == gender] ,s = 200 , alpha = 0.5 , label = gender)
plt.xlabel('Annual Income (k$)'), plt.ylabel('Spending Score (1-100)')
plt.title('Annual Income vs Spending Score w.r.t Gender')
plt.legend()
plt.show()
```



```
plt.figure(1 , figsize = (15 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Age' , y = 'Annual Income (k$)' , data = df[df['Gender'] == gender] ,
                s = 200 , alpha = 0.5 , label = gender)
plt.xlabel('Age'), plt.ylabel('Annual Income (k$)')
plt.title('Age vs Annual Income w.r.t Gender')
plt.legend()
plt.show()
```

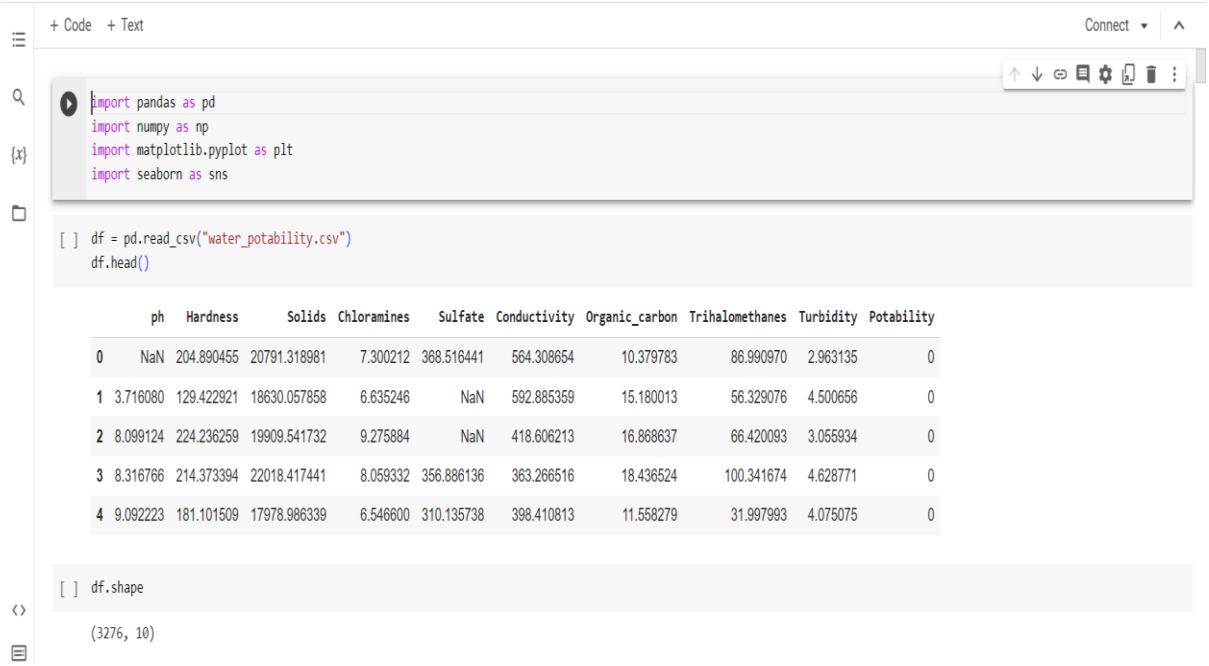


```
plt.figure(1 , figsize = (15 , 7))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    for y in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
        n += 1
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
        sns.regplot(x = x , y = y , data = df)
        plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else y )
plt.show()
```



Histograms

```
[ ] plt.figure(1 , figsize = (15 , 6))
    n = 0
    for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
        n += 1
        plt.subplot(1 , 3 , n)
        plt.subplots_adjust(hspace =0.5 , wspace = 0.5)
        sns.distplot(df[x] , bins = 20)
        plt.title('Distplot of {}'.format(x))
    plt.show()
```



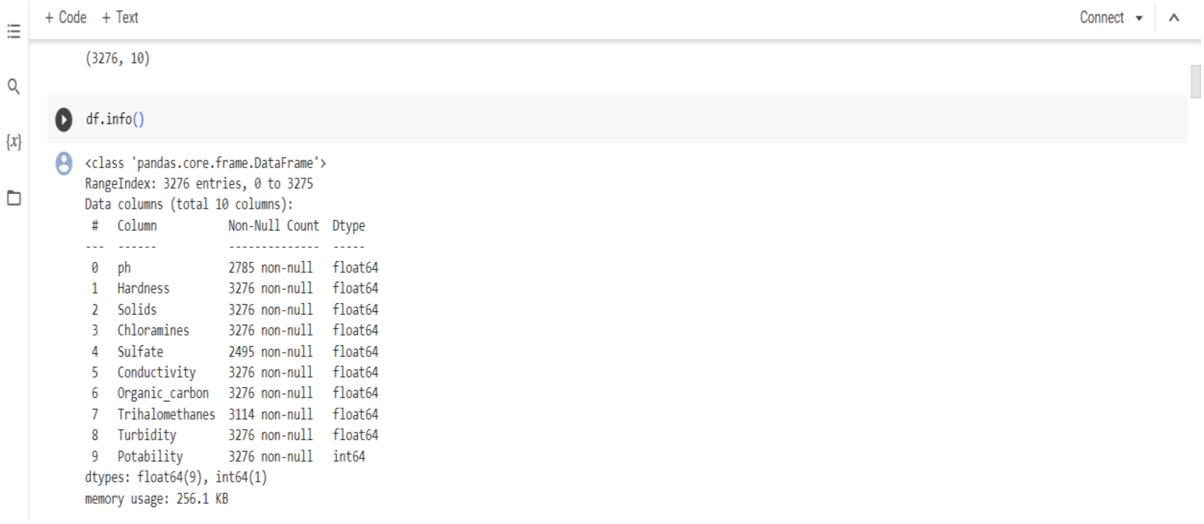
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

```
[ ] df.shape
```

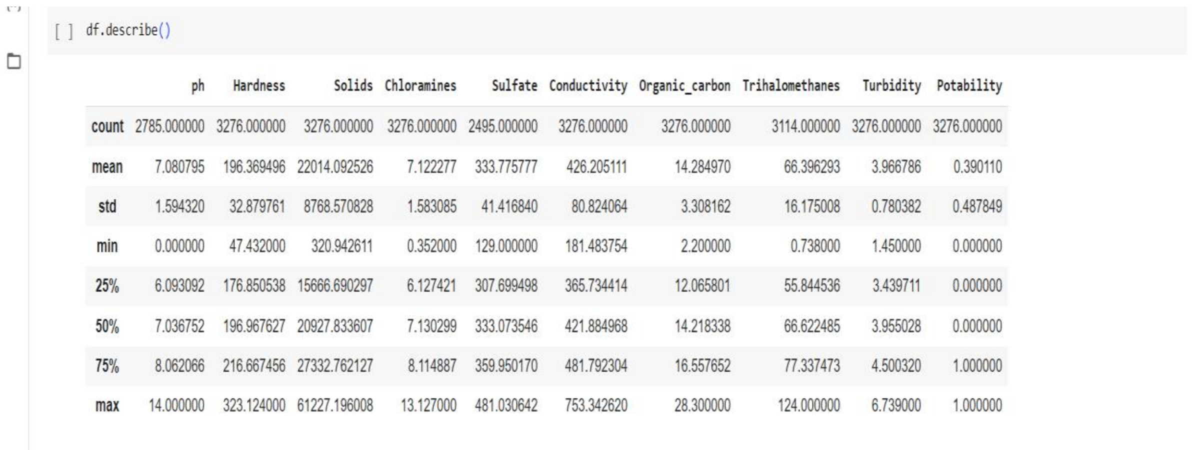
```
(3276, 10)
```



```
(3276, 10)
```

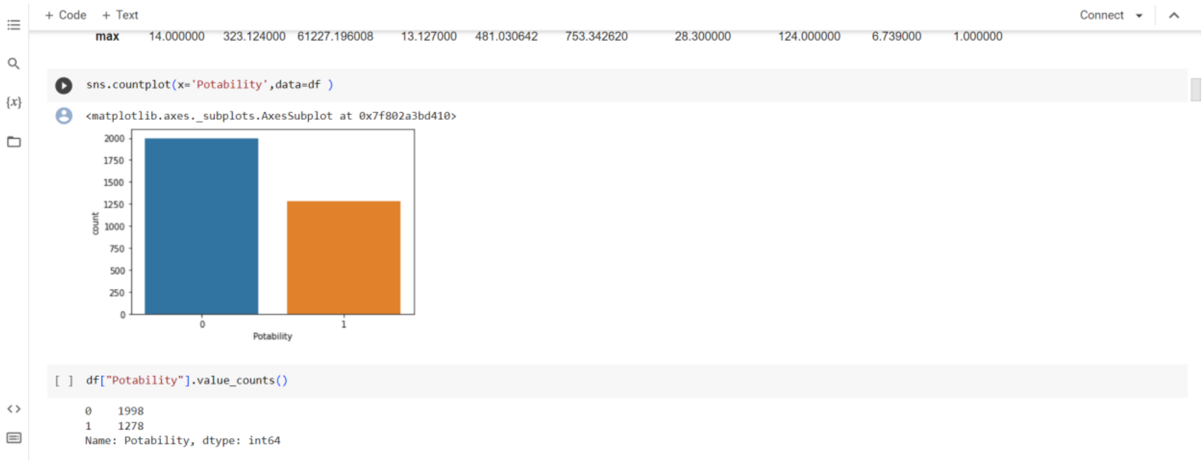
```
df.info()
```

```
<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
```



```
[ ] df.describe()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000



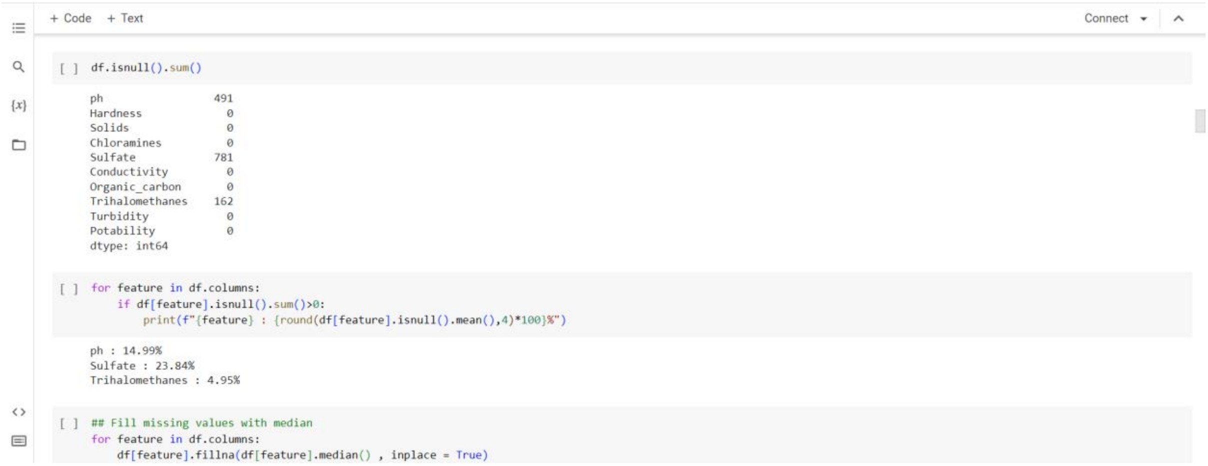
+ Code
+ Text
Connect
max 14.000000 323.124000 61227.196008 13.127000 481.030642 753.342620 28.300000 124.000000 6.739000 1.000000
2000
1750
1500
1250
1000
750
500
250
0
count
0
1
Potability

```
sns.countplot(x='Potability',data=df )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f802a3bd410>
```

```
[ ] df["Potability"].value_counts()
```

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



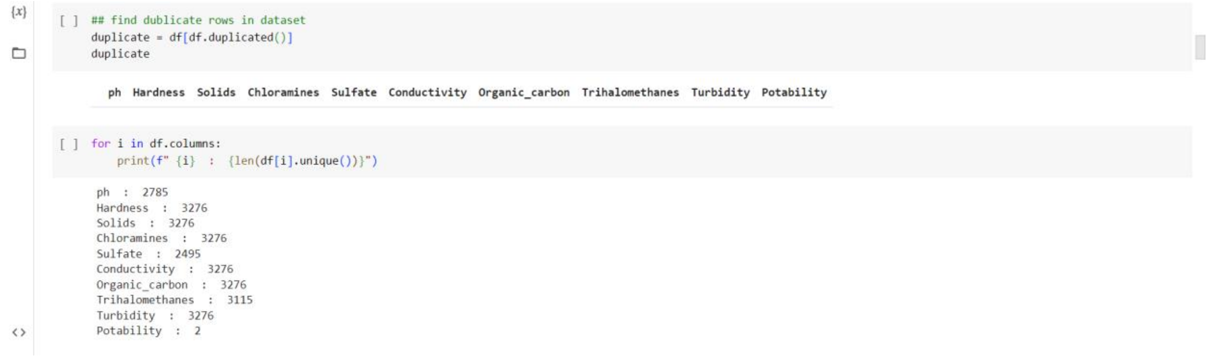
```
[ ] df.isnull().sum()
```

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

```
[ ] for feature in df.columns:
        if df[feature].isnull().sum()>0:
            print(f"{feature} : {round(df[feature].isnull().mean(),4)*100}%")
```

```
ph : 14.99%
Sulfate : 23.84%
Trihalomethanes : 4.95%
```

```
[ ] ## Fill missing values with median
    for feature in df.columns:
        df[feature].fillna(df[feature].median() , inplace = True)
```

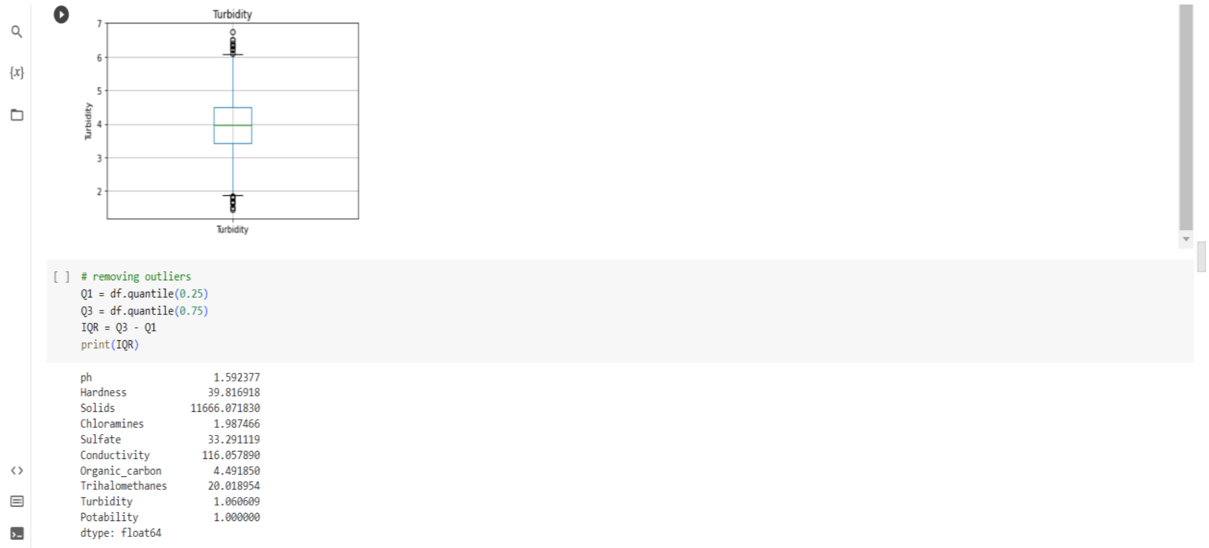


```
[ ] ## find dublicate rows in dataset
    duplicate = df[df.duplicated()]
    duplicate
```

```
ph  Hardness  Solids  Chloramines  Sulfate  Conductivity  Organic_carbon  Trihalomethanes  Turbidity  Potability
```

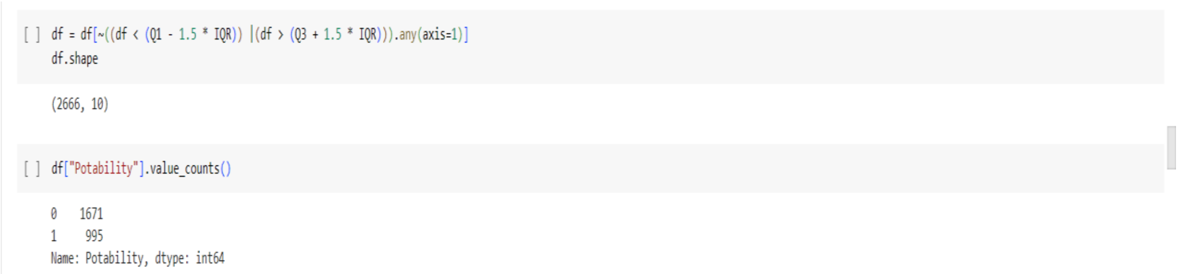
```
[ ] for i in df.columns:
        print(f" {i}  :  {len(df[i].unique())}")
```

```
 ph  :  2785
 Hardness  :  3276
 Solids  :  3276
 Chloramines  :  3276
 Sulfate  :  2495
 Conductivity  :  3276
 Organic_carbon  :  3276
 Trihalomethanes  :  3115
 Turbidity  :  3276
 Potability  :  2
```



```
[ ] # removing outliers
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    print(IQR)
```

```
ph                     1.592377
Hardness              39.816918
Solids             11666.071830
Chloramines            1.987466
Sulfate               33.291119
Conductivity         116.057890
Organic_carbon         4.491850
Trihalomethanes       20.018954
Turbidity              1.060609
Potability             1.000000
dtype: float64
```

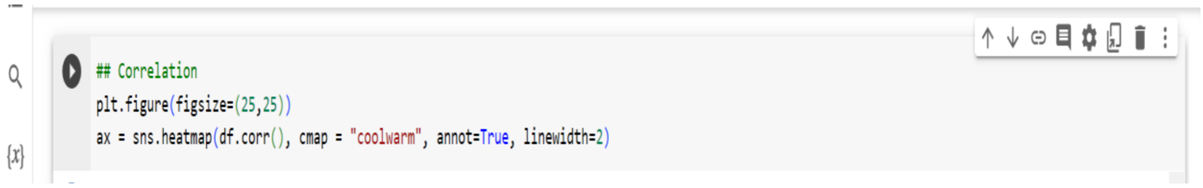


```
[ ] df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
    df.shape
```

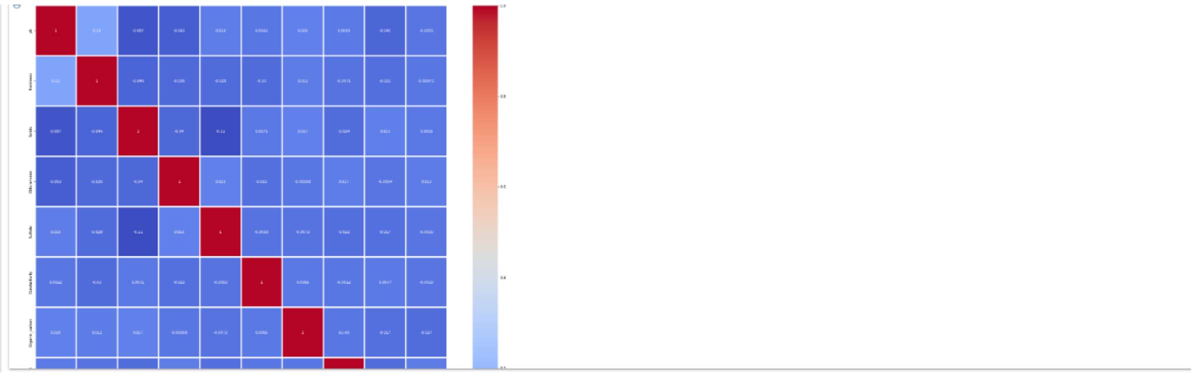
```
(2666, 10)
```

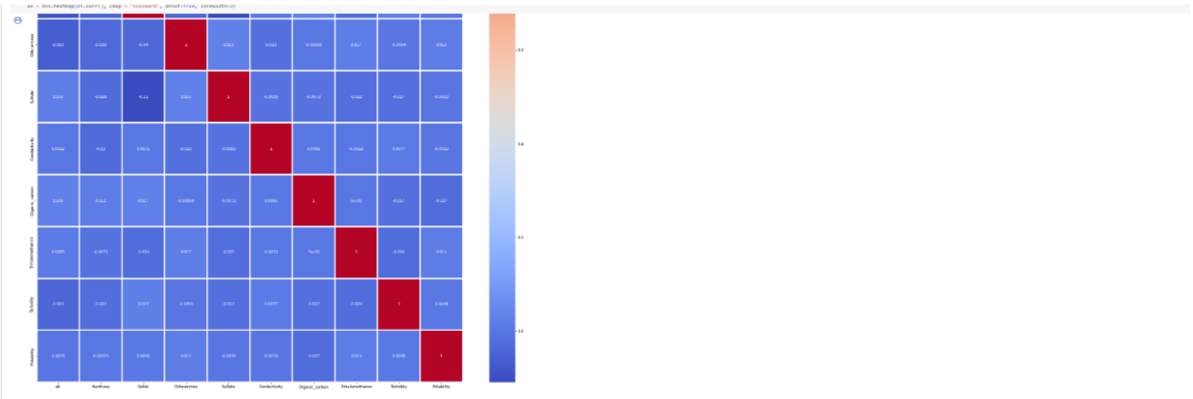
```
[ ] df["Potability"].value_counts()
```

```
0    1671
1     995
Name: Potability, dtype: int64
```

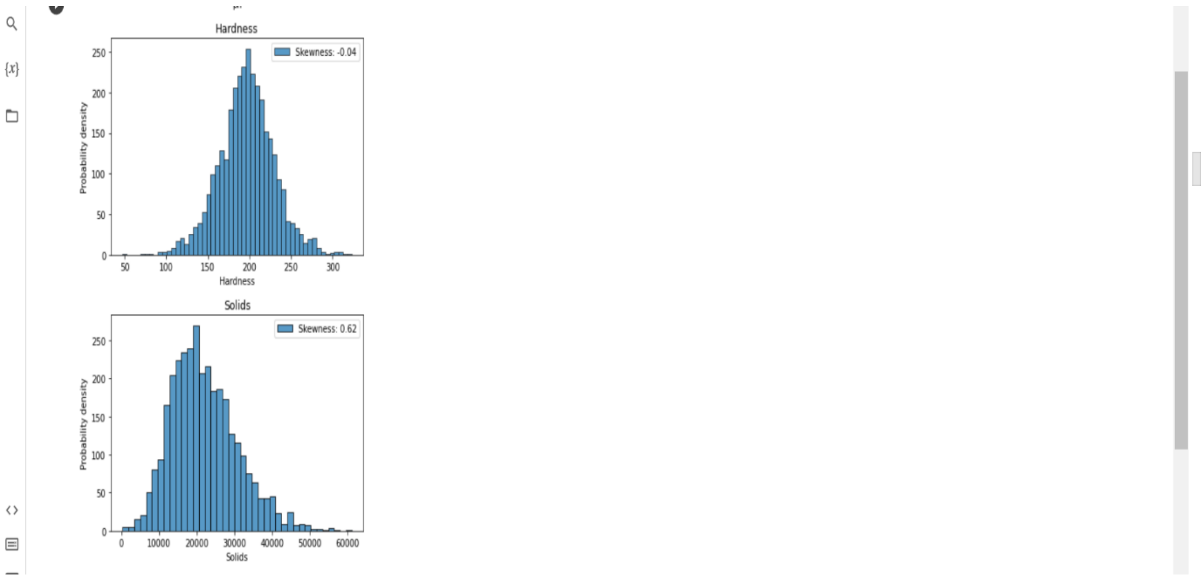


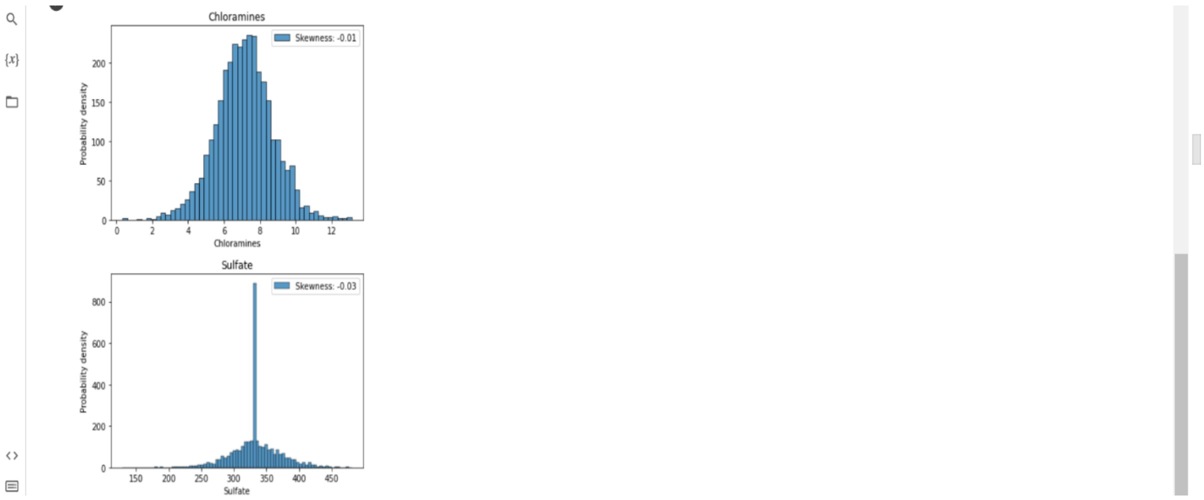
```
## Correlation
plt.figure(figsize=(25,25))
ax = sns.heatmap(df.corr(), cmap = "coolwarm", annot=True, linewidth=2)
```

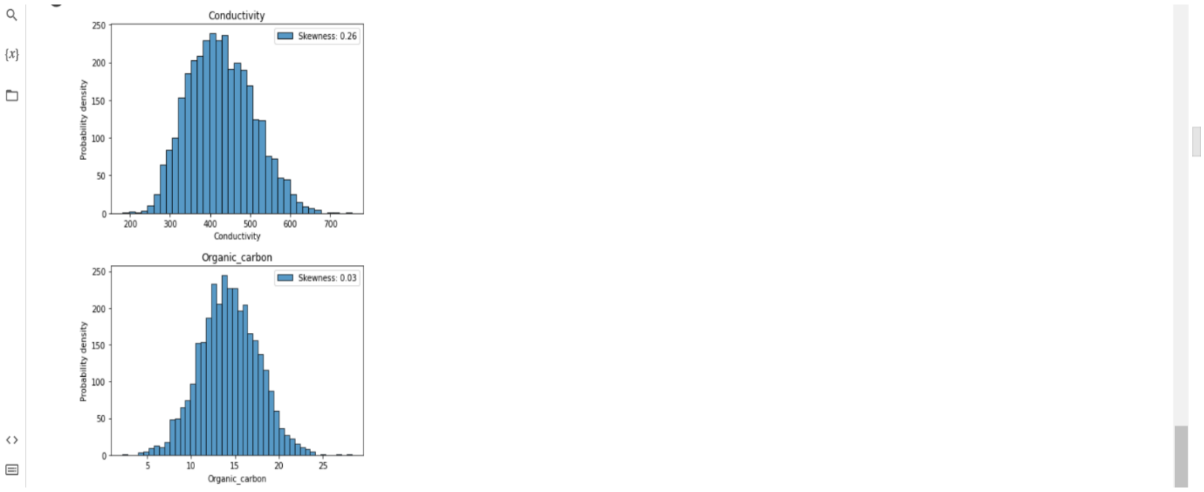


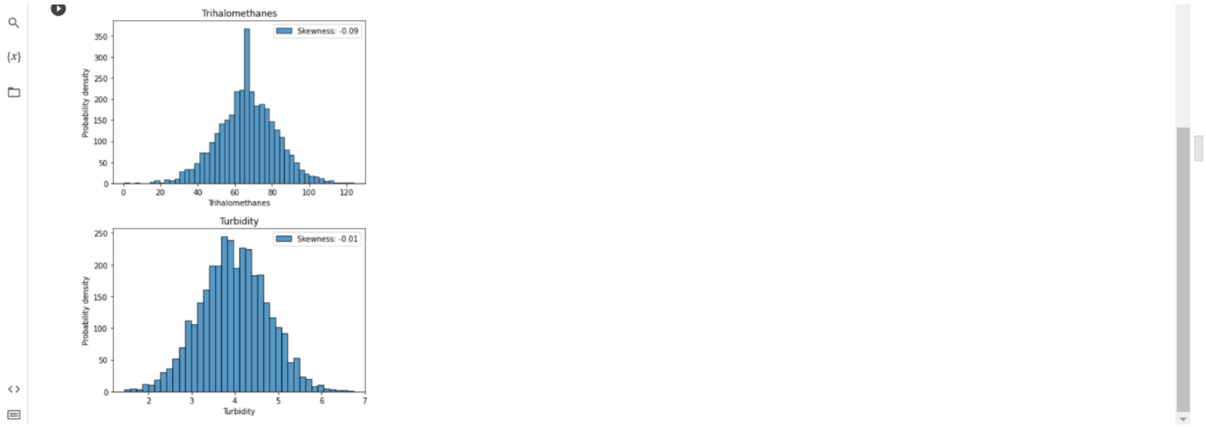


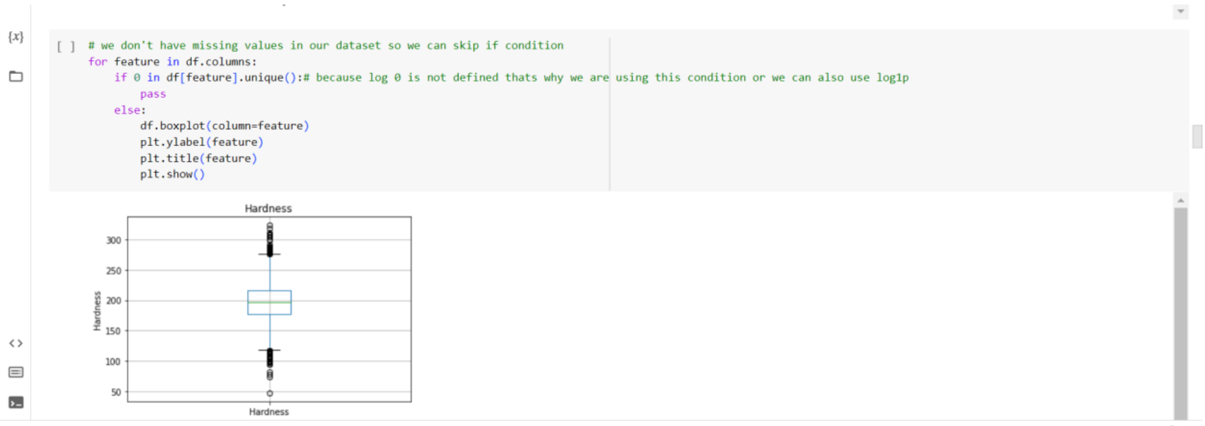
```
ax = sns.heatmap(df.corr(), cmap = "coolwarm", annot=True, linewidth=2)
```


Hardness
Skewness: -0.04
Probability density
0
50
100
150
200
250
50
100
150
200
250
300
Hardness
Solids
Skewness: 0.62
Probability density
0
50
100
150
200
250
0
10000
20000
30000
40000
50000
60000
Solids

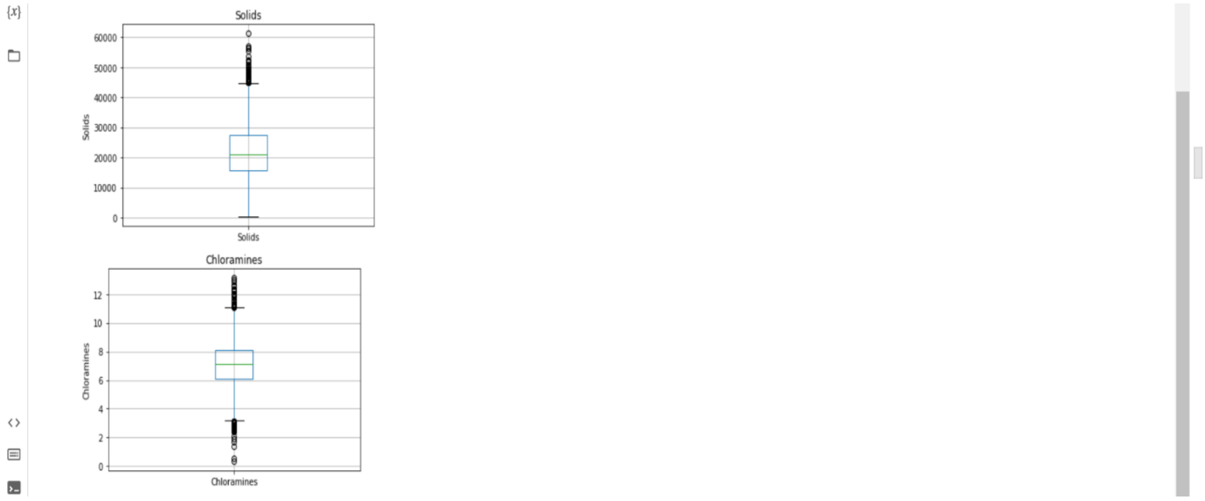

Chloramines
Skewness: -0.01
Probability density
0
50
100
150
200
0
2
4
6
8
10
12
Chloramines
Sulfate
Skewness: -0.03
Probability density
0
200
400
600
800
150
200
250
300
350
400
450
Sulfate

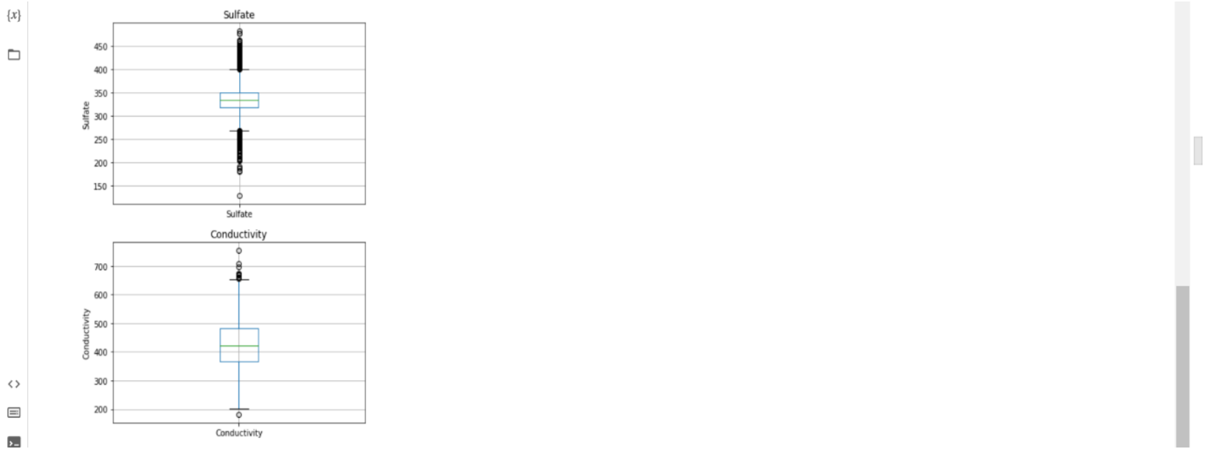

Conductivity
Skewness: 0.26
Probability density
0
50
100
150
200
250
200
300
400
500
600
700
Conductivity
Organic_carbon
Skewness: 0.03
Probability density
0
50
100
150
200
250
5
10
15
20
25
Organic_carbon

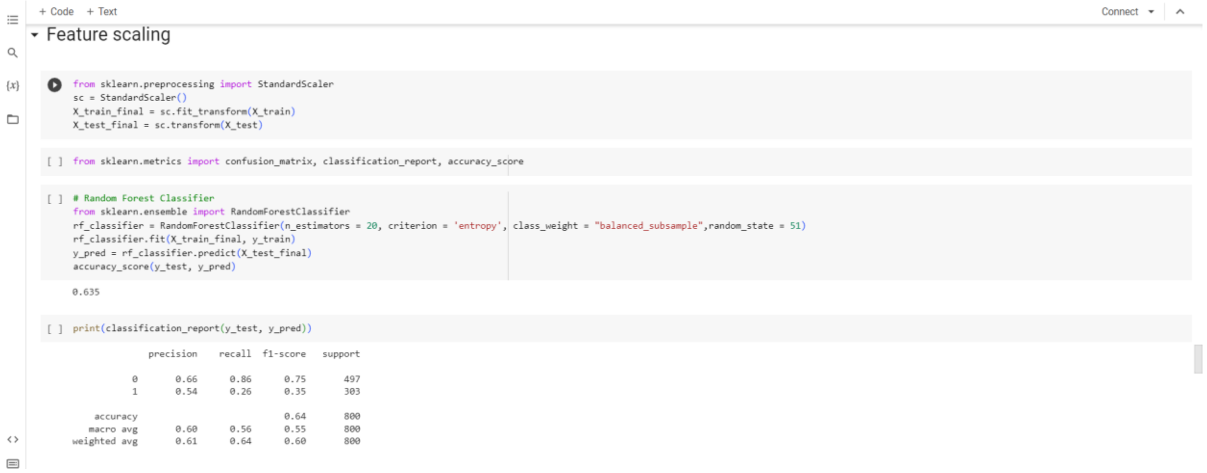




```
[ ] # we don't have missing values in our dataset so we can skip if condition
    for feature in df.columns:
        if 0 in df[feature].unique():# because log 0 is not defined thats why we are using this condition or we can also use log1p
            pass
        else:
            df.boxplot(column=feature)
            plt.ylabel(feature)
            plt.title(feature)
            plt.show()
```







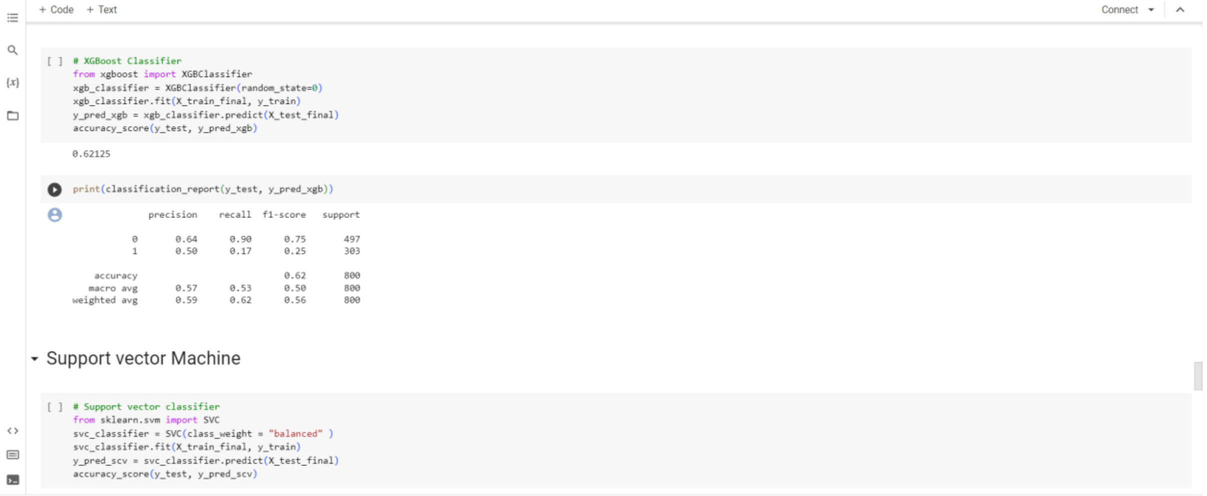
Feature scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_final = sc.fit_transform(X_train)
X_test_final = sc.transform(X_test)
```

```
[ ] from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
[ ] # Random Forest Classifier
    from sklearn.ensemble import RandomForestClassifier
    rf_classifier = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', class_weight = "balanced_subsample",random_state = 51)
    rf_classifier.fit(X_train_final, y_train)
    y_pred = rf_classifier.predict(X_test_final)
    accuracy_score(y_test, y_pred)

    0.635
```

```
[ ] print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.66      0.86      0.75       497
           1       0.54      0.26      0.35       303

    accuracy                           0.64       800
   macro avg       0.60      0.56      0.55       800
weighted avg       0.61      0.64      0.60       800
```

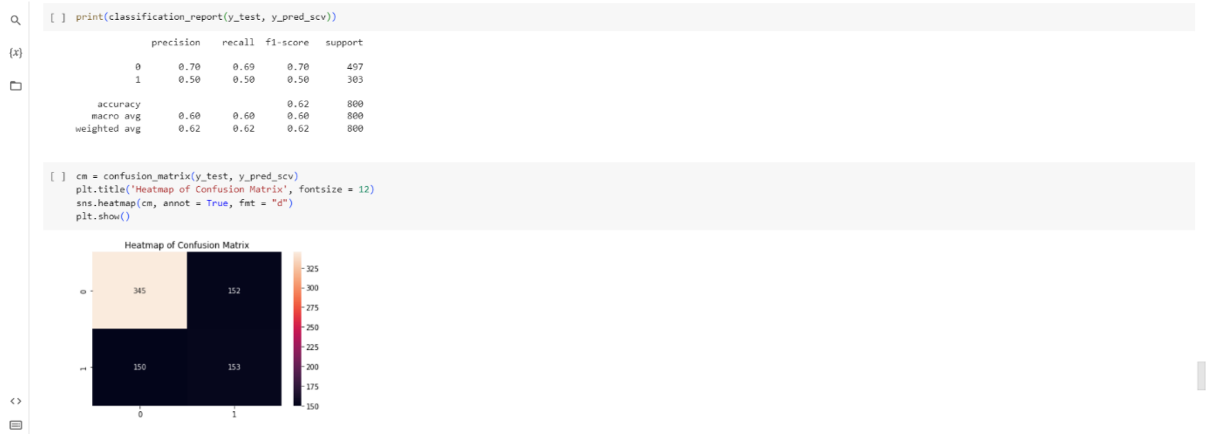


```
[ ] # XGBoost Classifier
    from xgboost import XGBClassifier
    xgb_classifier = XGBClassifier(random_state=0)
    xgb_classifier.fit(X_train_final, y_train)
    y_pred_xgb = xgb_classifier.predict(X_test_final)
    accuracy_score(y_test, y_pred_xgb)

    0.62125
```

```
print(classification_report(y_test, y_pred_xgb))

              precision    recall  f1-score   support

           0       0.64      0.90      0.75       497
           1       0.50      0.17      0.25       303

    accuracy                           0.62       800
   macro avg       0.57      0.53      0.50       800
weighted avg       0.59      0.62      0.56       800
```

Support vector Machine

```
[ ] # Support vector classifier
    from sklearn.svm import SVC
    svc_classifier = SVC(class_weight = "balanced" )
    svc_classifier.fit(X_train_final, y_train)
    y_pred_scv = svc_classifier.predict(X_test_final)
    accuracy_score(y_test, y_pred_scv)
```



```
print(classification_report(y_test, y_pred_scv))
```

```
              precision    recall  f1-score   support

           0       0.70      0.69      0.70       497
           1       0.50      0.50      0.50       303

    accuracy                           0.62       800
   macro avg       0.60      0.60      0.60       800
weighted avg       0.62      0.62      0.62       800
```

```
cm = confusion_matrix(y_test, y_pred_scv)
plt.title('Heatmap of Confusion Matrix', fontsize = 12)
sns.heatmap(cm, annot = True, fmt = "d")
plt.show()
```

Water Quality Predictive Model

In this project, we developed a water quality predictive model using the Support Vector Machine (SVM) algorithm within a Tkinter-based graphical user interface in Python. The model leverages historical water quality data to predict water quality levels based on various parameters, such as pH, Hardness, Solids, Chloramines, Sulphate, Conductivity, Organic_Carbon, Trihalomethanes and Turbidity. The Tkinter GUI allows users to input these parameters and receive real-time predictions, making it a valuable tool for environmental monitoring and ensuring the safety of water resources.

Code :

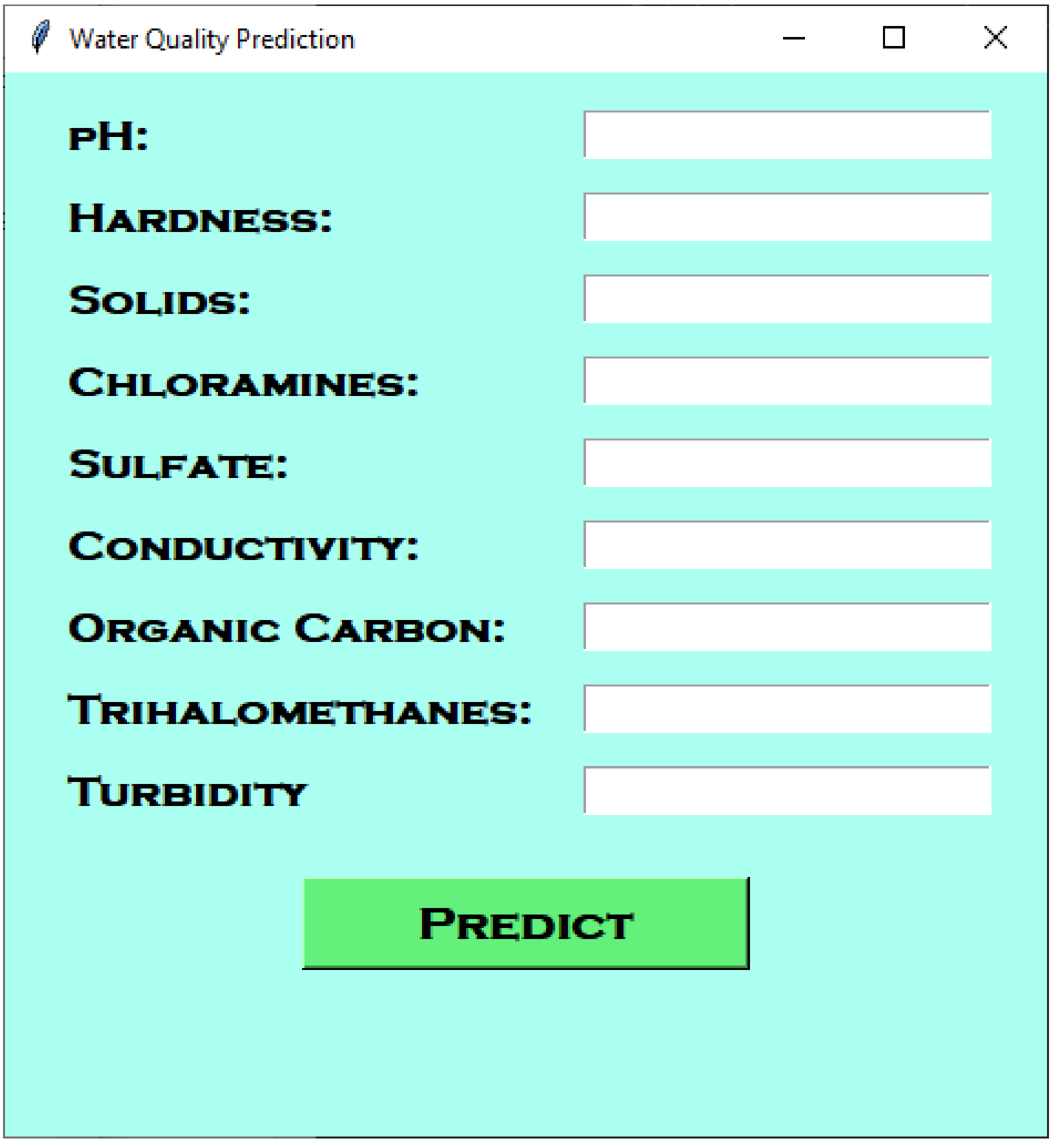
```
import tkinter as tk

from tkinter import Entry, Button, Label, Frame

import pickle

import pandas as pd

import numpy as np

import joblib

scaler = joblib.load("final_scaler.save")

model = pickle.load(open('final_model.pkl', 'rb'))
```

```python
# Function to make a prediction
def predict():
    input_features = [float(entry.get()) for entry in entry_widgets]
    features_value = [np.array(input_features)]
    feature_names = ["ph", "Hardness", "Solids", "Chloramines", "Sulfate", "Conductivity", "Organic_carbon","Trihalomethanes", "Turbidity"]
    df = pd.DataFrame(features_value, columns=feature_names)
    df = scaler.transform(df)
    output = model.predict(df)
    if output[0] == 1:
        prediction = "safe"
        result_label.config(text="Water is Safe for Human Consumption", fg="#68FF00")
    else:
        prediction = "not safe"
        result_label.config(text="Water is Not Safe for Human Consumption", fg="red")
# Create a Tkinter application window
app = tk.Tk()
app.title("Water Quality Prediction")
# Set the window geometry and background color
app.geometry("470x480")
app.configure(bg='#ABFFF1')
# Create a frame for the input fields and labels
input_frame = Frame(app, bg='#ABFFF1')
input_frame.pack(pady=10)
# Create input entry fields with labels
entry_labels = ["pH:", "Hardness:", "Solids:", "Chloramines:", "Sulfate:",
                "Conductivity:", "Organic Carbon:", "Trihalomethanes:", "Turbidity"]
entry_widgets = []
```

```
for label_text in entry_labels:

    label = Label(input_frame, text=label_text, bg='#ABFFF1',
font=("copperplate gothic bold", 14,"bold"), fg='black')

    label.grid(row=entry_labels.index(label_text), column=0, sticky='w',
padx=10, pady=5)

    entry = Entry(input_frame, font=("Arial", 12))

    entry.grid(row=entry_labels.index(label_text), column=1, padx=10, pady=5)

    entry_widgets.append(entry)

# Create a prediction button with custom style

predict_button = Button(app, text="Predict",command=predict,font=("copperplate
gothic bold", 16,"bold"), bg='#63F07B',
fg='black',width=12)predict_button.pack(pady=10)

# Create a label to display the prediction with increased font size and custom
style

result_label = Label(app, text="", font=("copperplate gothic bold", 14),
bg="#ABFFF1")

result_label.pack()

# Start the Tkinter main loop

app.mainloop()
```

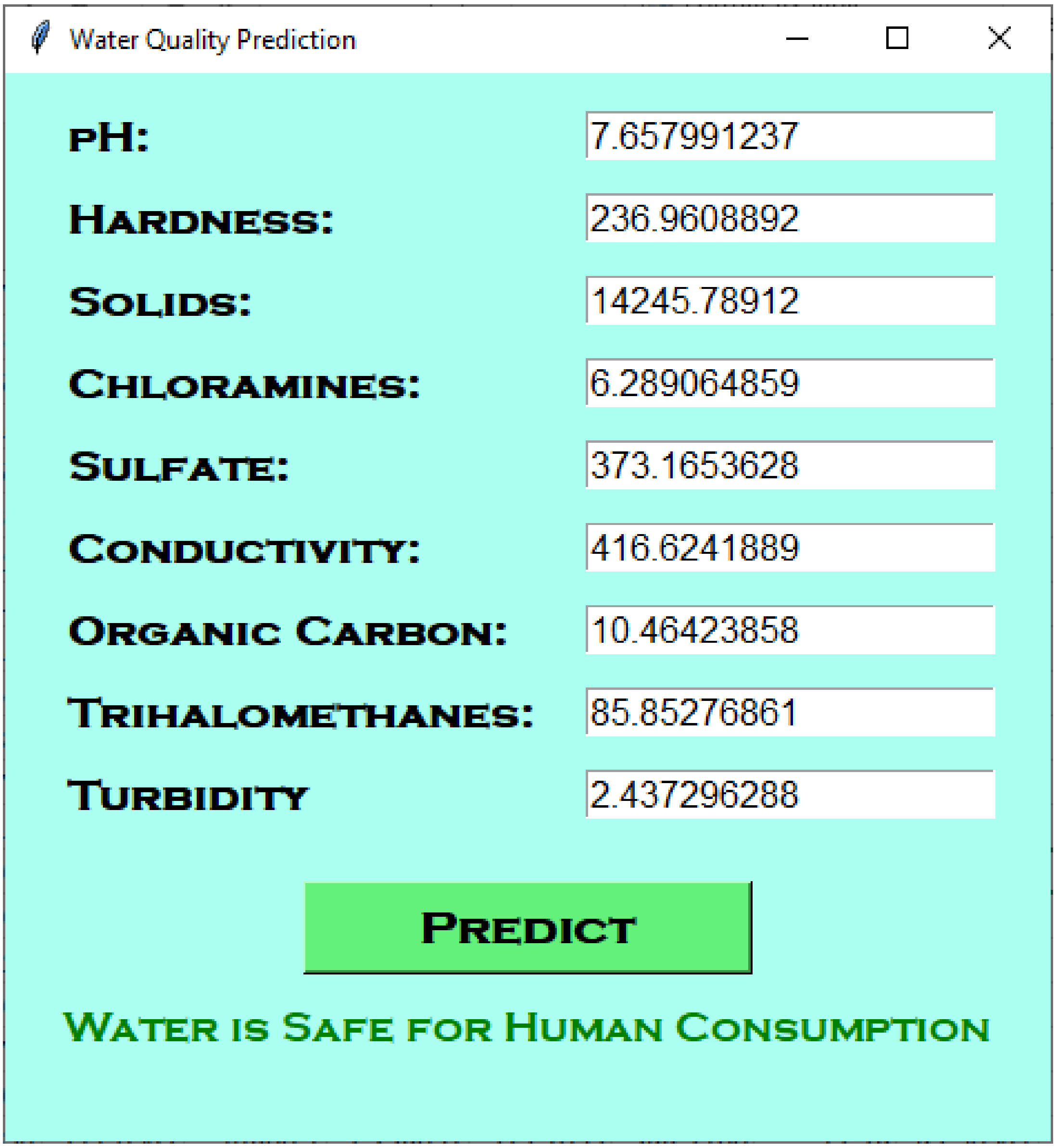
Output :

Home Interface :



Let’s test the predictive model using the water parameters provided in the dataset.

Testing with Potable values :



Water Quality Prediction

pH: 7.657991237

Hardness: 236.9608892

Solids: 14245.78912

Chloramines: 6.289064859

Sulfate: 373.1653628

Conductivity: 416.6241889

Organic Carbon: 10.46423858

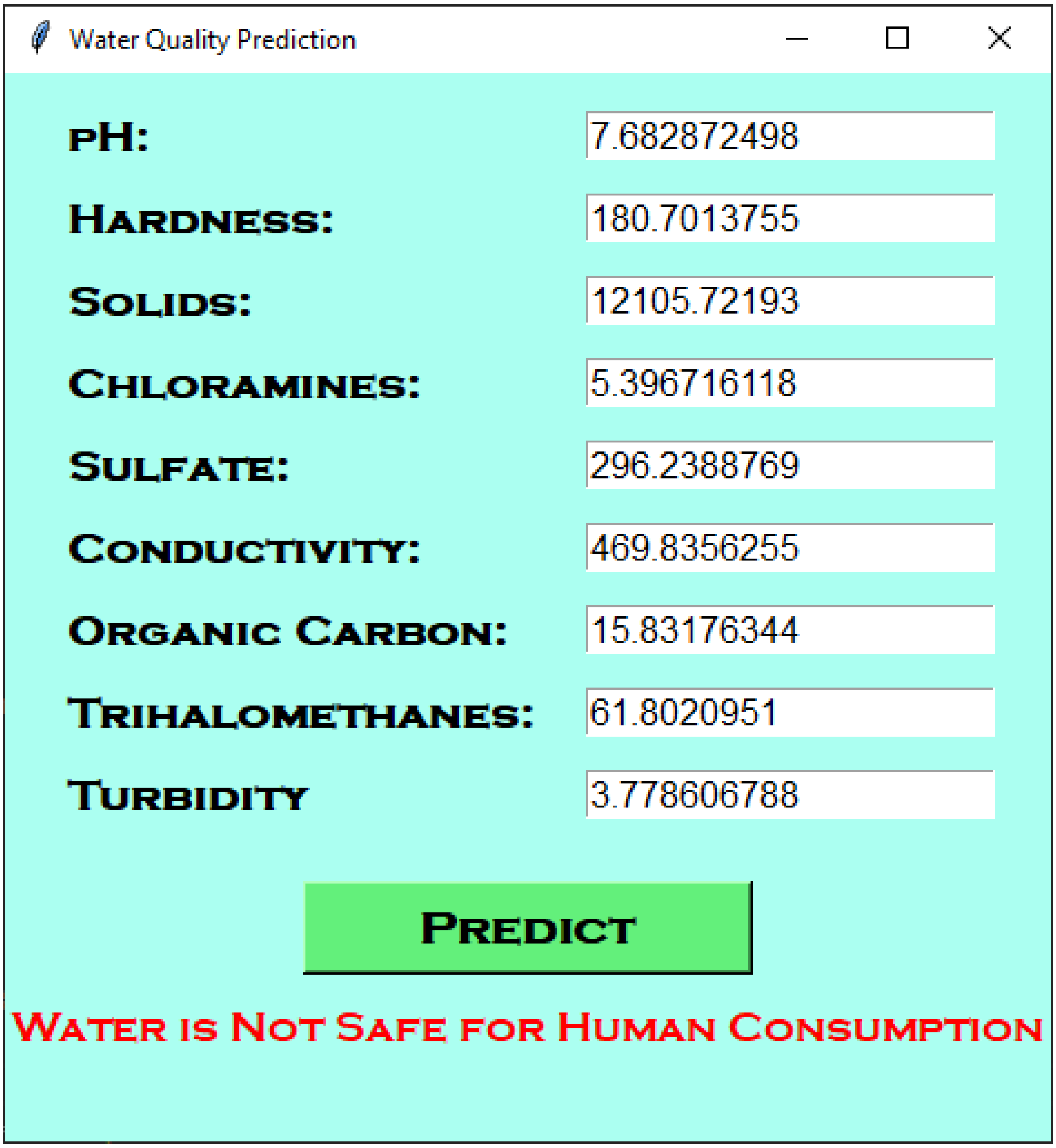
Trihalomethanes: 85.85276861

Turbidity 2.437296288

Water is Safe for Human Consumption

Here, we tested the Potable values provided in the dataset. So the Predictive model shows the result i.e "Water is Safe for Human Consumption".

Testing with Not Potable Values :



pH:	7.682872498
Hardness:	180.7013755
Solids:	12105.72193
Chloramines:	5.396716118
Sulfate:	296.2388769
Conductivity:	469.8356255
Organic Carbon:	15.83176344
Trihalomethanes:	61.8020951
Turbidity	3.778606788

Water is Not Safe for Human Consumption

Here, we tested the Not Potable values provided in the dataset. So the Predictive model shows the result i.e "Water is Not Safe for Human Consumption".

Conclusion :

- In the quest to build a Water Quality Prediction Model, we have embarked on a critical journey that begins with loading and preprocessing the Water_Potability dataset. We have traversed through essential steps, starting with importing the necessary libraries to facilitate data manipulation and analysis.

- Understanding the data's structure, characteristics, and any potential issues through exploratory data analysis (EDA) is essential for informed decision-making.

- Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.

- With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a Water Quality Prediction Model.

- Using multiple Machine Learning Algorithms, we concluded highest accuracy algorithm to make the water quality predictive model.

- The Water Quality Predictive Model is trained using SVM algorithm with the provided water samples data.

- Then the Water Quality Predictive Model is tested with several Water Quality Parameters and it provides the result whether its Potable or Not.

THANK
YOU...