Water Quality Analysis

Objective:

To Access to safe drinking-water is essential to health, a basic human right and a component of effective policy for health protection. This is important as a health and development issue at a national, regional and local level. In some regions, it has been shown that investments in water supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions. comprehensively evaluate water quality by assessing its potability, detecting deviations from established standards, and explaining the interrelationships among key parameters.

DATA VISUALISATION USING IBM COGNOS:

IBM Cognos is a robust business intelligence and data visualization tool that can be effectively utilized for water quality analysis, specifically in the context of various parameters such as pH, hardness, solids, chloroamines, sulfate, conductivity, organic carbon, turbidity, and trihalomethanes. When considering the aspect of portability, Cognos provides several features and benefits that make it an ideal choice for ensuring that water quality data is accessible and usable from various locations and devices.

1. Web-Based Accessibility: IBM Cognos is web-based, which means that users can access their water quality analysis reports and dashboards from anywhere with an internet connection. This level of accessibility is crucial for water quality monitoring, as it allows stakeholders and decision-makers to view critical data without being tied to a specific physical location.

2. Responsive Design: Cognos dashboards are designed with responsiveness in mind. This means that the visualizations adapt to the screen size and orientation of the device being used. Whether it's a desktop computer, tablet, or smartphone, users can view and interact with water quality data without any loss of functionality or clarity.

3. Data Synchronization: Cognos can be set up to synchronize with data sources, ensuring that the most up-to-date water quality data is available regardless of the user's location. This is crucial for maintaining accuracy in water quality monitoring, especially in situations where conditions can change rapidly.

4. Offline Access: In cases where an internet connection is not readily available, Cognos also allows for offline access to previously accessed reports and data. Users can download reports and visualizations for offline use, ensuring data availability even in remote areas.

5. Sharing and Collaboration: IBM Cognos facilitates easy sharing of water quality reports and visualizations. Users can collaborate by sharing links or scheduled reports, which is valuable for researchers, regulatory bodies, and environmental organizations that need to exchange data and insights across different locations.

6. Security and Access Control: Water quality data is often sensitive and subject to privacy and regulatory requirements. Cognos provides robust security features, including role-based access control and data encryption, to protect the integrity and confidentiality of data while ensuring that authorized personnel can access it from anywhere securely.

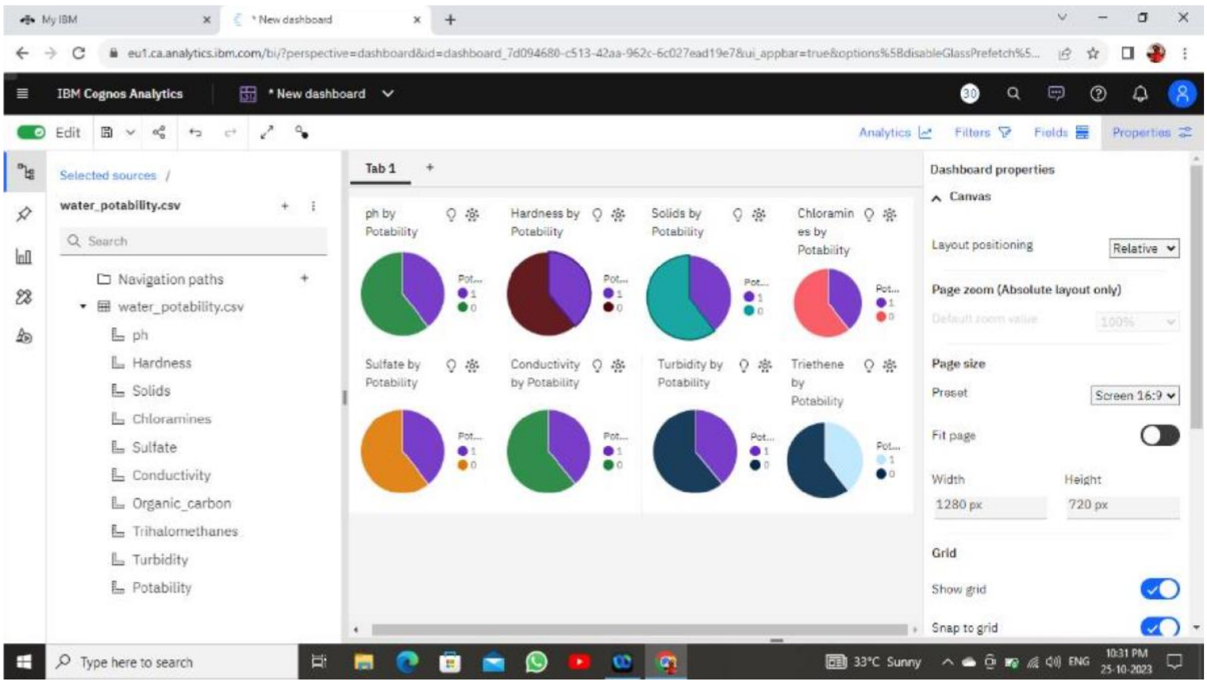
7. Interactivity and Drill-Down: Cognos allows users to interact with visualizations by drilling down into data for deeper insights. This feature is useful for remote teams and decision-makers who need to explore specific water quality parameters and trends even when they are not in the laboratory or at the water source.

IBM Cognos is a versatile tool for water quality analysis that prioritizes portability. It enables water quality data to be accessible, responsive, and secure from various locations and devices, making it an ideal solution for managing and monitoring water quality across different regions and scenarios. This portability ensures that stakeholders and decisionmakers have the information they need, whenever and wherever they need it, to make informed decisions about water quality management and regulation.

Visualizing or working with IBM Cognos:

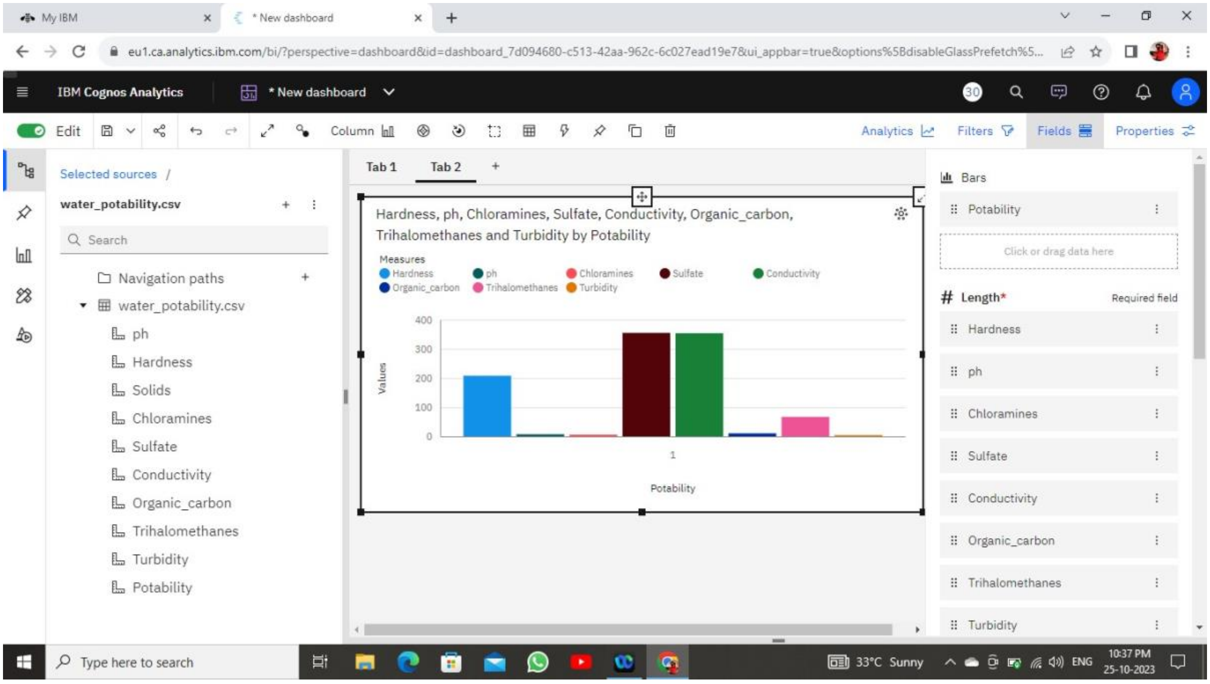
1. Creating dash board:

   a. Visualising data with respect to potability using pie chart representation:

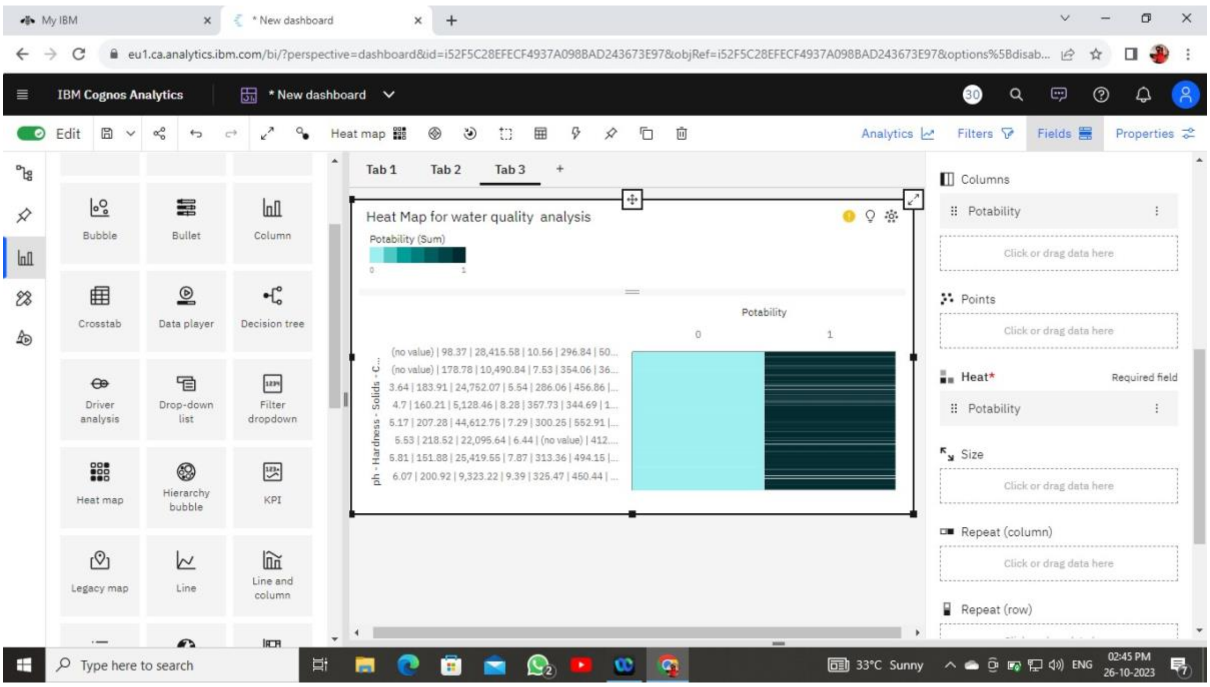


Visualising data with respect to potability using

b. bar chart representation:

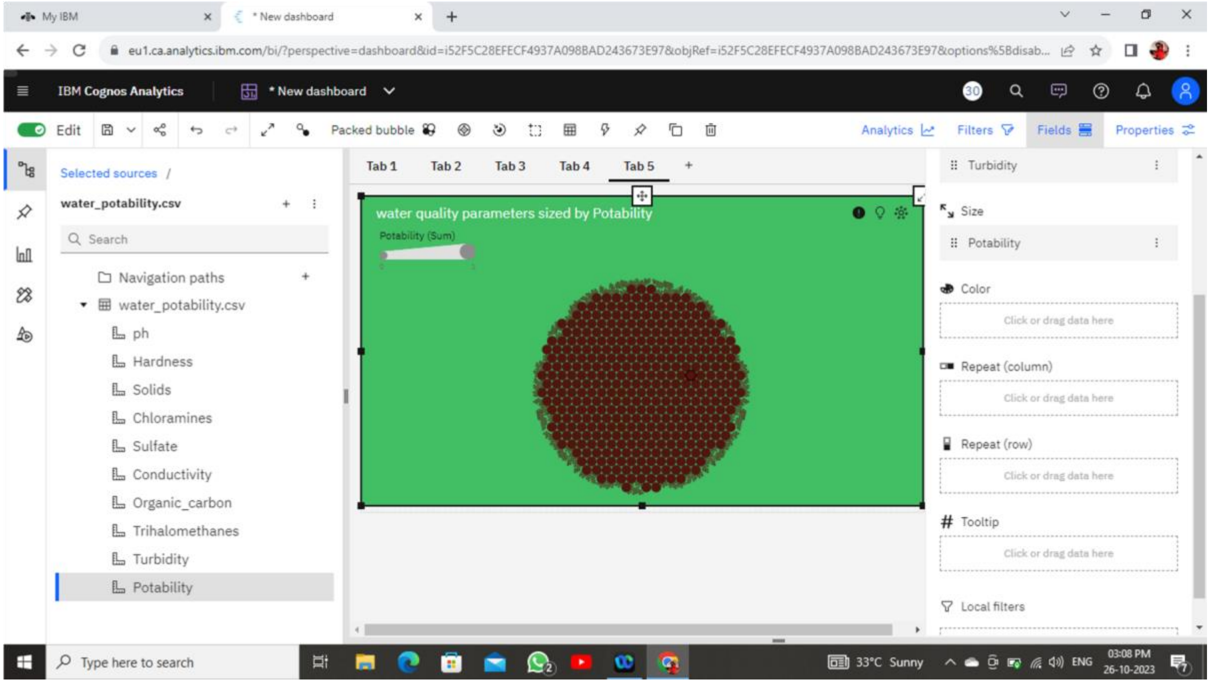


c. Visualising data with respect to potability using heat map representation:

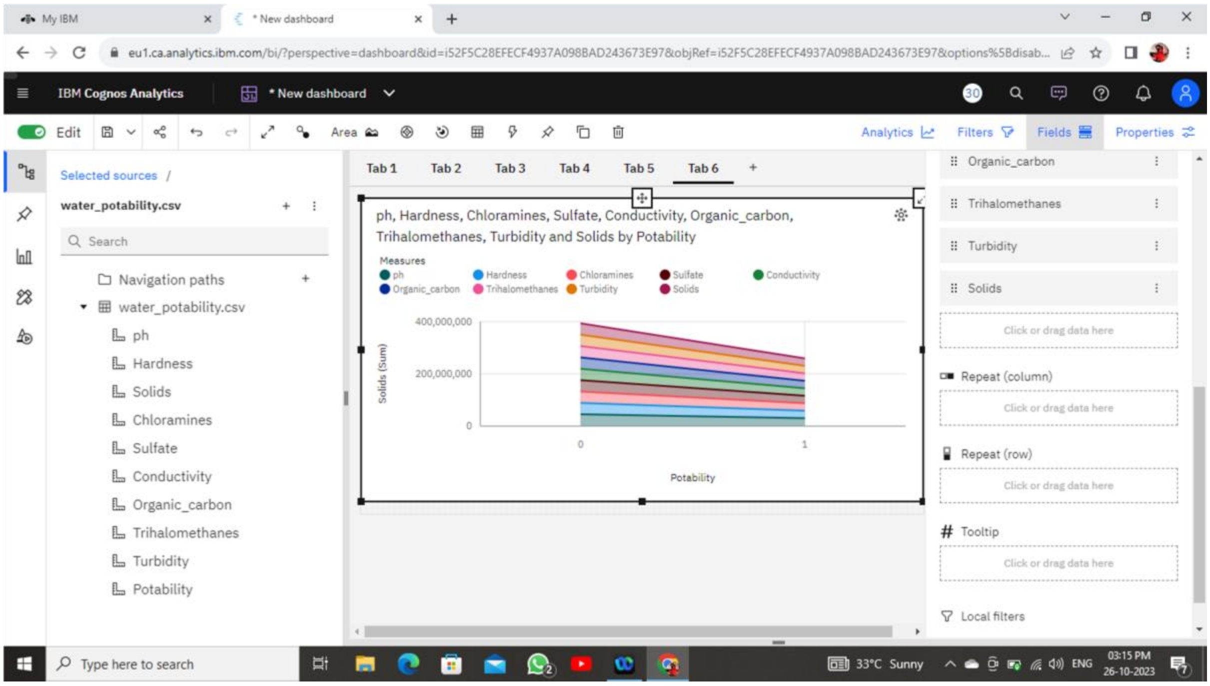


Visualising data with respect to potability using

d. bubble map representation:

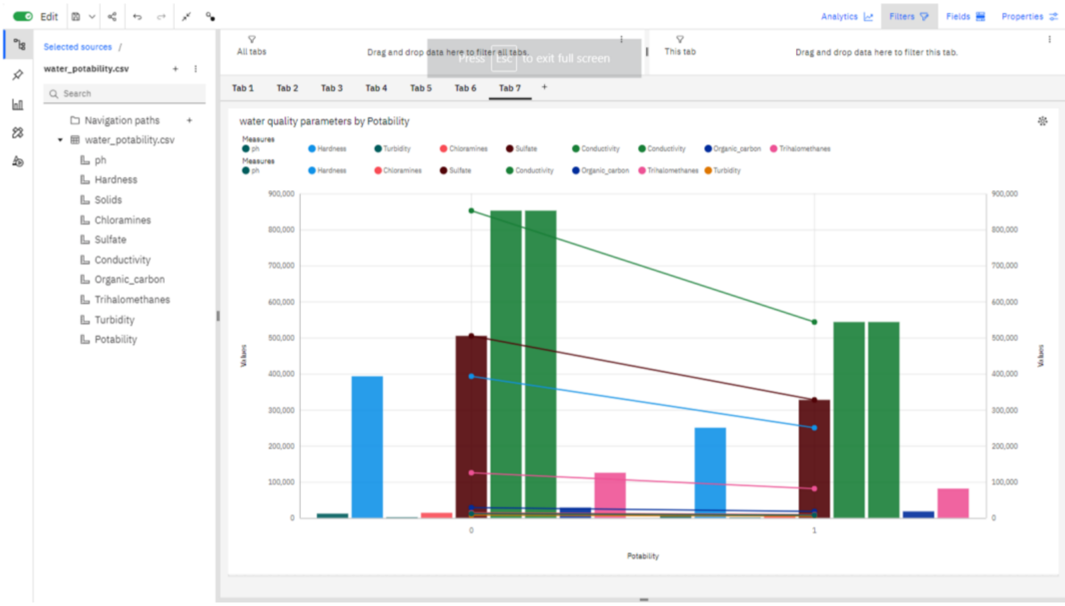


e. Visualising data with respect to potability using area map representation:

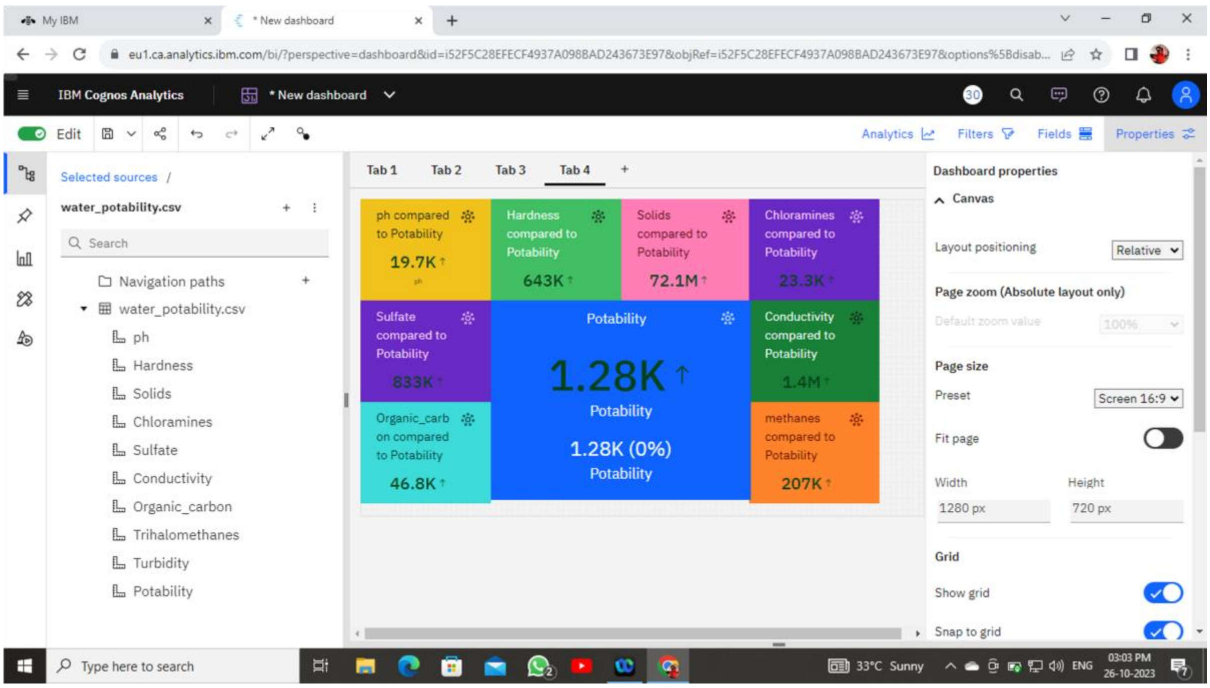


f. line bar map representation:

Visualising data with respect to potability using



g. Visualising data with respect to potability using summary map representation:



- Now its time to prepare the model, divide the data set into independent and dependent features. Then X contains all the independent features except the potability and Y contain the feature our target potability. Then we are splitting the data set into training and testing using train test function with some parameter like X, Y, test size, randomise to ignore the shuffle every time.

```
In [5]: X = sample.drop('Potability',axis=1)
        Y = sample['Potability']
```

```
In [11]: 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=101,shuffle=True)
```

- Train decision Tree classifier and check accuracy:

```
In [12]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
         sample=DecisionTreeClassifier(criterion= 'gini', min_samples_split= 10, splitter='best')
         sample.fit(X_train,Y_train)

Out[12]: DecisionTreeClassifier(min_samples_split=10)
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
In [33]: prediction = sample.predict(X_test)
         print(f"Accuracy Score = {accuracy_score(Y_test,prediction) * 100}")
         print(f"Confusion Matrix =\n{confusion_matrix(Y_test,prediction)}")
         print(f"Classification Report =\n{classification_report(Y_test,prediction)}")

         Accuracy Score = 59.29878048780488
         Confusion Matrix =
         [[276 126]
          [141 113]]
         Classification Report =
                       precision    recall  f1-score   support

                    0       0.66      0.69      0.67       402
                    1       0.47      0.44      0.46       254

             accuracy                           0.59       656
            macro avg       0.57      0.57      0.57       656
         weighted avg       0.59      0.59      0.59       656
```

Apply Hyper parameter tuning:(creation of dictionary for given data set)

```
In [44]: from sklearn.model_selection import RepeatedStratifiedKFold
         from sklearn.model_selection import GridSearchCV
         # define models and parameters
         model = DecisionTreeClassifier()
         criterion = ["gini", "entropy"]
         splitter = ["best", "random"]
         min_samples_split = [2,4,6,8,10,12,14]
         # define grid search
         grid = dict(splitter=splitter, criterion=criterion, min_samples_split=min_samples_split)
         cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
         grid_search_sample = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)
         grid_search_sample.fit(X_train,Y_train)

Out[44]: GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10, random_state=1),
                      error_score=0, estimator=DecisionTreeClassifier(), n_jobs=-1,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
                                  'splitter': ['best', 'random']},
                      scoring='accuracy')
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

- Checking the created model for given datas and accuracy:

```
In [51]: print(f"Best: {grid_search_sample.best_score_:.3f} using {grid_search_sample.best_params_}")
         means = grid_search_sample.cv_results_['mean_test_score']
         stds = grid_search_sample.cv_results_['std_test_score']
         params = grid_search_sample.cv_results_['params']
         for mean, stdev, param in zip (means, stds, params):
             print (f"{mean:.3f}({stdev:.3f}) with:{params}")

         print("Training Score:",grid_search_sample.score (X_train,Y_train)*100)
         print("Testing Score:", grid_search_sample.score (X_test,Y_test)*100)
```

```
Best: 0.591 using {'criterion': 'gini', 'min_samples_split': 14, 'splitter': 'random'}
0.586(0.036) with:[{'criterion': 'gini', 'min_samples_split': 2, 'splitter': 'best'}, {'criterion': 'gini', 'min_samples_spl
it': 2, 'splitter': 'random'}, {'criterion': 'gini', 'min_samples_split': 4, 'splitter': 'best'}, {'criterion': 'gini', 'min
_samples_split': 4, 'splitter': 'random'}, {'criterion': 'gini', 'min_samples_split': 6, 'splitter': 'best'}, {'criterion':
'gini', 'min_samples_split': 6, 'splitter': 'random'}, {'criterion': 'gini', 'min_samples_split': 8, 'splitter': 'best'},
{'criterion': 'gini', 'min_samples_split': 8, 'splitter': 'random'}, {'criterion': 'gini', 'min_samples_split': 10, 'splitte
r': 'best'}, {'criterion': 'gini', 'min_samples_split': 10, 'splitter': 'random'}, {'criterion': 'gini', 'min_samples_spli
t': 12, 'splitter': 'best'}, {'criterion': 'gini', 'min_samples_split': 12, 'splitter': 'random'}, {'criterion': 'gini', 'mi
n_samples_split': 14, 'splitter': 'best'}, {'criterion': 'gini', 'min_samples_split': 14, 'splitter': 'random'}, {'criterio
n': 'entropy', 'min_samples_split': 2, 'splitter': 'best'}, {'criterion': 'entropy', 'min_samples_split': 2, 'splitter': 'ra
ndom'}, {'criterion': 'entropy', 'min_samples_split': 4, 'splitter': 'best'}, {'criterion': 'entropy', 'min_samples_split':
4, 'splitter': 'random'}, {'criterion': 'entropy', 'min_samples_split': 6, 'splitter': 'best'}, {'criterion': 'entropy', 'mi
```

Training Score: 80.19083969465649

Testing Score: 58.079268292682926

Conclusion:

In conclusion, leveraging IBM Cognos for water quality analysis with a focus on parameters such as pH, hardness, solids, chloroamines, sulfate, conductivity, organic carbon, turbidity, and trihalomethanes brings a powerful and versatile solution to the forefront. The emphasis on portability within the Cognos framework significantly enhances the efficiency and effectiveness of water quality monitoring and management.The ability to access, visualize, and interact with water quality data from various locations and devices is essential for modern environmental monitoring. IBM Cognos accomplishes this by offering web-based accessibility, responsive design, mobile app support, data synchronization, offline access, and robust security features. This ensures that water quality stakeholders, whether in a lab, office, or in the field, can make informed decisions and respond to changing conditions promptly.The flexibility and user-friendliness of Cognos, combined with its secure and collaborative capabilities, facilitate seamless data sharing and collaboration among researchers, regulators, and environmental organizations, even across different geographic locations.A demo model has been created with respect to water quality analysis. In the realm of water quality analysis, where timely and accurate data is paramount, IBM Cognos with its portability features proves to be a vital tool in ensuring the availability, accessibility, and reliability of data, ultimately contributing to the preservation and management of water resources for a sustainable and healthy environment.