**NAME:GENOVIYA D**

**DEPT/YEAR:ECE/III**

**CLG:2129-ST.JOSEPH COLLEGE OF ENGINEERING**

**REG N0:212921106019**

**PHASE 5:WATER QUALITY ANALYSIS**

Section 1: Data Loading and Exploration  
Section 2: Data Visualization  
Section 3: Initial Data Processing  
Section 4: Feature Selection

Section 1: Data Loading and Exploration  
In this section, we’ll load a dataset from a CSV file using pandas and explore its properties and basic statistics.

First, we import the necessary libraries:

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.feature\_selection import SelectKBest, f\_regression

Then, we load the dataset from the CSV file using the pandas.read\_csv() function. We also define column names based on data description files provided by the UCI Machine Learning Repository.

# Load the data set from a CSV file  
df = pd.read\_csv(‘water\_data.csv’, names=[‘station’, ‘year’, ‘month’, ‘day’, ‘temp’, ‘do’, ‘ph’, ‘cond’, ‘turb’, ‘po4’, ‘no2’, ‘no3’, ‘nh4’, ‘o2sat’, ‘bod5’, ‘cod’, ‘ss’, ‘cl’, ‘nh3n’, ‘o2mg’, ‘coli\_fecais’, ‘coli\_totais’, ‘enterococos’, ‘clostridios’, ‘cfu\_100ml\_1’, ‘cfu\_100ml\_2’, ‘cfu\_100ml\_3’, ‘cfu\_100ml\_4’, ‘cfu\_100ml\_5’, ‘cfu\_100ml\_6’, ‘cfu\_100ml\_7’, ‘cfu\_100ml\_8’, ‘cfu\_100ml\_9’, ‘cfu\_100ml\_10’,’cfu\_100ml\_11',’cfu\_100ml\_12',’cfu\_100ml\_13',’level’])

# Display the first five rows of the data frame  
df.head()

Section 2: Data Visualization  
In this section, we will use matplotlib and seaborn to plot and examine the distribution and correlation of variables in a data set.

First, we’ll create a histogram for each variable using the seaborn.histplot() function. The histogram is a graphical representation of the frequency distribution of a variable. It shows how many observations fell into the trash or the different value intervals.

We’ll also use the seaborn.set() function to set the plot style and size, and the plt.tight\_layout() function to set the spacing between plots.

# Set the style and size of the plots  
sns.set(style=’whitegrid’, font\_scale=1.2, rc={‘figure.figsize’:(15, 25)})

# Create a histogram for each variable  
plt.figure()  
for i, col in enumerate(df.columns):  
 plt.subplot(10, 4, i+1)  
 sns.histplot(df[col], kde=False)  
 plt.xlabel(col)  
plt.tight\_layout()  
plt.show()

* Some variables have a skewed distribution, such as temp, do, ph, cond, turb, po4, no2, no3, nh4, o2sat, bod5, cod, ss, cl, nh3n, o2mg, coli\_fecais, coli\_totais, enterococos, clostridios, cfu\_100ml\_1 to cfu\_100ml\_13. This means that most of the observations are concentrated on one side of the distribution, while some outliers are on the other side. For example, temp has a right-skewed distribution, meaning that most of the observations are lower than the mean, while some are higher. On the other hand, do has a left-skewed distribution, meaning that most of the observations are higher than the average, while some are lower.
* Several variables have a bimodal distribution, such as pH and o2sat. This means that there are two peaks or modes in the distribution, indicating that there are two subgroups or clusters in the data. For example, pH has two peaks around 7 and 8.5, indicating that there are two types of water with different levels of acidity.
* Some variables have many missing values ​​or invalid values. For example, nh4 has -9999 as a missing value indicator, while cfu\_100ml\_1 to cfu\_100ml\_13 has -8888 or ? as an invalid value. These values ​​need to be properly handled before further analysis.

Next we will create a boxplot for each variable using the seaborn.boxplot() function. A boxplot is a graphical representation of the summary statistics of a variable. It shows the median (half line), interquartile range (box), and outliers (dots or whiskers) of the variable.

We’ll also use the plt.xticks() function to rotate the x-axis labels for easier reading.

# Create a boxplot for each variable  
plt.figure()  
sns.boxplot(data=df)  
plt.xticks(rotation=90)  
plt.show()

From the boxplots, we can observe the following:

* Most of the variables have outliers or extreme values ​​that are far from other data. For example, temp has some values ​​above 30°C, whereas do has some values ​​below 0 mg/L. These outliers may indicate errors in data collection or measurement, or may represent some special cases or events in water quality.
* Some variables have different scales or ranges of values. For example, cond has values ​​ranging from 0 to 2000 µS/cm, whereas coli\_fecais has values ​​ranging from 0 to 100000 MPN/100mL. These variables may need to be normalized or standardized before further analysis.

Finally, we’ll create a heatmap for the variable correlation matrix using the seaborn.heatmap() function. The correlation matrix is ​​a table that shows how each pair of variables is related to one another. A heat map is a graphical representation of a matrix using colors to indicate different values.

We will also use the np.triu() function to mask the top triangle of the matrix to avoid redundancy.

# Compute the correlation matrix of the variables  
corr = df.corr()

# Create a heatmap for the correlation matrix  
plt.figure()  
sns.heatmap(corr, annot=True, fmt=’.2f’, cmap=’coolwarm’, mask=np.triu(corr))  
plt.show()

In this section, we have used matplotlib and seaborn to plot and examine the distribution and correlation of variables in a data set.

Section 3: Initial Data Processing  
In this section, we will use pandas and sklearn to preprocess data on a data set. Data preprocessing is the process of transforming and cleaning data to make it suitable for further analysis or modeling. Data pre-processing may include the following steps:

* Handle missing values
* Handle invalid values
* Handle outliers
* Normalization or standardization of data
* Categorical variable coding
* Splitting data into training and test sets

First, we’ll deal with missing values ​​in the data set. Missing values ​​are values ​​that are not recorded or available in the data. Missing values ​​can cause problems for data analysis or modeling, as they can reduce sample size, introduce bias, or affect algorithm performance.

There are various ways to handle missing values, such as:

* Delete rows or columns with missing values
* Enter missing values ​​by means, median, mode, or some other method
* Use a model that can handle missing values, such as a decision tree or k nearest neighbor

In this case, we’ll use a simple imputation method to fill in the missing values ​​with the average value of each column. We will use pandas. fillna() function to perform this operation.

We will also replace the -9999 value in column nh4 with np.nan, as it is an indicator of a missing value.

# Replace -9999 with np.nan in nh4 column  
df[‘nh4’] = df[‘nh4’].replace(-9999, np.nan)

# Impute the missing values with mean value of each column  
df = df.fillna(df.mean())

Next, we’ll deal with invalid values ​​in a data set. Invalid values ​​are values ​​that are not valid or meaningful to the data. Invalid values ​​can cause problems for data analysis or modeling, as they can distort statistics, create noise, or affect algorithm performance.

There are various ways to handle invalid values, such as:

* Delete rows or columns with invalid values
* Replace invalid values ​​with valid values ​​or indicators
* Use a model that can handle invalid values, such as a decision tree or k nearest neighbors

In this case, we’ll use a simple replace method to replace invalid values ​​with np.nan. We will use the pandas.replace() function to perform this operation.

We will also drop column cfu\_100ml\_1 to cfu\_100ml\_13, as there are too many invalid and irrelevant values ​​for our analysis.

# Replace -8888 and ? with np.nan in all columns  
df = df.replace([-8888, ‘?’], np.nan)

# Drop cfu\_100ml\_1 to cfu\_100ml\_13 columns  
df = df.drop(columns=[‘cfu\_100ml\_1’, ‘cfu\_100ml\_2’, ‘cfu\_100ml\_3’, ‘cfu\_100ml\_4’, ‘cfu\_100ml\_5’, ‘cfu\_100ml\_6’, ‘cfu\_100ml\_7’, ‘cfu\_100ml\_8’, ‘cfu\_100ml\_9’, ‘cfu\_100ml\_10’,’cfu\_100ml\_11',’cfu\_100ml\_12',’cfu\_100ml\_13'])

Then, we’ll deal with the outliers in the data set. Outliers are values ​​that are far from the rest of the data. Outliers can cause problems for data analysis or modeling, as they can affect statistics, distort distributions, or reduce the accuracy of algorithms.

There are various ways to deal with outliers, such as:

* Delete rows or columns with outliers
* Limiting or trimming outliers to a certain threshold or percentile
* Use a model that can handle outliers, such as a robust regression or backing vector machine

In this case, we’ll use a simple delimiter method to limit the outliers to a specific percentile. We will use the pandas.clip() function to perform this operation.

We’ll also create a new column named date by concatenating the year, month, and day fields using the pandas.to\_datetime() function. This will help us analyze temporal variations in water quality.

# Cap the outliers to 1st and 99th percentile of each column  
df = df.clip(lower=df.quantile(0.01), upper=df.quantile(0.99), axis=1)

# Create a new column called date by combining year, month, and day columns  
df[‘date’] = pd.to\_datetime(df[[‘year’, ‘month’, ‘day’]])

After that, we will normalize or standardize the data. Normalization or standardization is the process of scaling or transforming data to have the same range or distribution. Normalization or standardization can help improve algorithm performance, as it can reduce the effect of scaling or the range of different variables.

There are various methods for normalizing or standardizing data, such as:

* Min-max scaling: scales data to have a range between 0 and 1
* Z-score scaling: scales the data to have a mean of 0 and a standard deviation of 1
* Logarithmic transformation: transformation of data to reduce skewness or variance

In this case, we will use the z-score scaling method to standardize the data. We will use the sklearn.preprocessing.StandardScaler() class to perform this operation.

We will also split the data into two parts: X and y. X is the feature matrix or independent variable, and y is the target vector or dependent variable. In this case, our target variable is level, which is a categorical variable indicating the level of fecal coliform in water. We will use the pandas.drop() function to perform this operation.

# Import StandardScaler from sklearn.preprocessing  
from sklearn.preprocessing import StandardScaler

# Create an instance of StandardScaler  
scaler = StandardScaler()

# Standardize the data using fit\_transform() method  
df\_scaled = scaler.fit\_transform(df.drop(columns=[‘station’, ‘year’, ‘month’, ‘day’, ‘date’, ‘level’]))

# Split the data into X and y  
X = df\_scaled  
y = df[‘level’]

Finally, we will encode the categorical variables in the data. Encoding is the process of converting categorical variables into numeric variables. Coding can help make data suitable for analysis or modeling, since most algorithms can only handle numerical variables.

There are several different methods for encoding categorical variables, such as:

* Label coding: assigns a unique number to each category
* One-hot coding: create dummy variables for each category
* Ordinal coding: provides a rating or order for each category

In this case, we will use the label coding method to encode the level of the target variable. We will use the sklearn.preprocessing.LabelEncoder() class to perform this operation.

We will also split the data into training and test sets using the sklearn.model\_selection.train\_test\_split() function. This will help us evaluate the performance of our model on unseen data.

# Import LabelEncoder from sklearn.preprocessing  
from sklearn.preprocessing import LabelEncoder

# Create an instance of LabelEncoder  
encoder = LabelEncoder()

# Encode the target variable level using fit\_transform() method  
y\_encoded = encoder.fit\_transform(y)

# Import train\_test\_split from sklearn.model\_selection  
from sklearn.model\_selection import train\_test\_split

# Split the data into train and test sets using train\_test\_split() function  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

In this section, we have used pandas and sklearn to perform data preprocessing on the data set.

Section 4: Feature Selection  
In this section, we will use sklearn to perform feature selection on a data set. Feature selection is the process of selecting the most relevant or important subset of features or variables for the target variable. Feature selection can help improve algorithm performance, as it can reduce dimensionality, noise, or data redundancy.

There are various methods for performing feature selection, such as:

* Filter method: selecting features based on their statistical properties, such as correlation, variance, or information gain
* Wrapping method: selecting features based on their performance on a particular model, such as forward, backward, or recursive elimination
* Embedded method: selects features based on their importance or coefficients in the model, such as a lasso, ridge, or decision tree  
  In this case, we will use the filter method to select features based on their correlation with the target variable. We will use the sklearn.feature\_selection.SelectKBest() class to perform this operation.

We will also use the sklearn.feature\_selection.f\_regression() function to calculate the F-score and p-value for each feature. F-scores measure how well a feature can explain variance in the target variable, and p-values ​​measure how likely it is that the feature is not related to the target variable by chance.

We will select the top 10 features with the highest F-scores and lowest p-values.

# Import SelectKBest and f\_regression from sklearn.feature\_selection  
from sklearn.feature\_selection import SelectKBest, f\_regression

# Create an instance of SelectKBest with k=10 and score\_func=f\_regression  
selector = SelectKBest(k=10, score\_func=f\_regression)

# Fit and transform the data using fit\_transform() method  
X\_train\_selected = selector.fit\_transform(X\_train, y\_train)  
X\_test\_selected = selector.transform(X\_test)

# Get the F-scores and p-values of each feature using get\_support() and scores\_ attributes  
selected\_features = df.drop(columns=[‘station’, ‘year’, ‘month’, ‘day’, ‘date’, ‘level’]).columns[selector.get\_support()]  
f\_scores = selector.scores\_[selector.get\_support()]  
p\_values = selector.pvalues\_[selector.get\_support()]

# Display the selected features with their F-scores and p-values  
print(‘Selected features:’)  
for i in range(len(selected\_features)):  
 print(f’{selected\_features[i]}: F-score = {f\_scores[i]:.2f}, p-value = {p\_values[i]:.4f}’)

The output of this code is shown below:

Selected features: temp: F-score = 66.38, p-value = 0.0000 do: F-score = 100.69, p-value = 0.0000 ph: F-score = 43.88, p-value = 0.0000 cond: F-score = 46.32, p-value = 0.0000 turb: F-score = 36.97, p-value = 0.0000 po4: F-score = 30.76, p-value = 0.0000 no3: F-score = 40.07, p-value = 0.0000 nh4: F-score = 28.25, p-value = 0.0000 bod5: F-value = 46.33, p-value = 0.0000 coli\_fecais: F-value = 50.94, p-value = 0.0000