Objectives given to develp

1. Water data collection

2. Applying normalization techniques on data collected

2. a. data pre-processing : for the given dataset include only water parameters ( eg PH,Salinity,TDS etc

2. b. balancing the imbalanced data set,

3. Applying machine learning algorithms to

i. To detect water quality (safe=1 or unsafe=0 )last column of dataset should be 1or 0: calculation of WQI with any machine learning algorithm

ii. Impact of one parameter with the other (correlation between various parameters)

iii. find the best accuracy machine learning algorithm

iv. suggesting range of parameters impacting the water quality is last step of the project.

**1.About Dataset** [**link**](https://www.kaggle.com/datasets/sivapriyagarladinne/telangana-post-monsoon-ground-water-quality-data)

**Context**

Water is essential for all the crops and livestock’s as much as it is important for human beings. Crops and livestocks consume direct ground water, and if the quality is not up to the mark, or becomes unusable, then crops and livestocks are affected, which may lead to crop failure or livestocks developing diseases.  
By knowing the quality of the ground water, whether usable or not, the water can be put to appropriate use. Farmers can grow specific crops which can tolerate that quality of water.

**Usage**

By using these datasets one can train an ML Classification model to classify the water quality into one of the multiple classes and know where and how the water can be used.

**Data collected**

This data is collected from Telangana Open Data portal, Telangana State, India.  
This data contains samples tested from various districts.

There are 3 files, each or year 2018, 2019 and 2020 contains post-monsoon season groundwater quality details.  
ground\_water\_quality\_2018\_post.csv,  
ground\_water\_quality\_2019\_post.csv,  
ground\_water\_quality\_2020\_post.csv can be combinedly used.

Each dataset contains 26 columns such as:  
serial num (sno), District, Mandal, Village, Lattitude, Longitude, Chemicals (such as Ca, Mg, CO3 etc), Total Hardness of the water, Total dissolved solids, RSC, SAR, and the target variables 'Classification' and 'Classification1'.  
The feature columns can be used to predict water quality which are broadly classified into 9 classes, which are: C3S1, C2S1, C4S1,C4S2 , C3S2 ,C4S4 ,C3S3 ,C4S3, C1S1.

**C1S1**: Low salinity and low sodium waters are good for irrigation and can be used with  
most crops with no restriction on use on most of the soils.

**C2S1**: Medium salinity and low sodium waters are good for irrigation and can be used on  
all most all soils with little danger of development of harmful levels of exchangeable  
sodium if a moderate amount of leaching occurs. Crops can be grown without any special  
consideration for salinity control.

**C3S1**: The high salinity and low sodium waters require good drainage. Crops with good  
salt tolerance should be selected.

**C3S2**: The high salinity and medium sodium waters require good drainage and can be used  
on coarse - textured or organic soils having good permeability.

**C3S3**: These high salinity and high sodium waters require special soil management, good  
drainage, high leaching and organic matter additions. Gypsum amendments make feasible  
the use of these waters.

**C4S1**: Very high salinity and low sodium waters are not suitable for irrigation unless the  
soil must be permeable and drainage must be adequate. Irrigation waters must be applied  
in excess to provide considerable leaching. Salt tolerant crops must be selected.

**C4S2**: Very high salinity and medium sodium waters are not suitable for irrigation on fine  
textured soils and low leaching conditions and can be used for irrigation on coarse textured  
or organic soils having good permeability.

**C4S3**: Very high salinity and high sodium waters produce harmful levels of exchangeable  
sodium in most soils and will require special soil management, good drainage, high  
leaching, and organic matter additions. The Gypsum amendment makes feasible the use of  
these waters.

**C4S4**: Very high salinity and very high sodium waters are generally unsuitable for  
irrigation purposes. These are sodium chloride types of water and can cause sodium  
hazards. It can be used on coarse-textured soils with very good drainage for very high salt tolerant crops. Gypsum amendments make feasible the use of these waters.

**Classification of groundwater based on RSC**

RSC is defined as the excess of carbonate and bicarbonate amount over the alkaline  
earths (Ca2+ and Mg2+). Use of RSC beyond permissible limit (>2.5) adversely affects irrigation.  
The tendency of Ca2+ and Mg2+ to precipitate, as the water in the soil becomes more  
concentrated, as a result of evaporation and plant transpiration, and gets fixed in the soil by the  
process of base exchange, thereby decreasing the soil permeability.  
RSC = ((CO3 2-) + (HCO3-)) - ((Ca2+)+(Mg2+))  
Where concentrations are in meq/L.

RSC less than 1.25 is **safe**  
RSC between 1.25 and 2.50 is **marginal**  
RSC greater than 2.50 is **unsuitable**

**Use of ground water for livestock and poultry**

TDS < 1000 mg/L -- Excellent -- Excellent for all classes of livestock and poultry.

TDS b/w 1000-3000 -- very satisfactory -- Satisfactory for all classes of livestock. May cause temporary mild diarrhea in livestock not accustomed to them. Those waters approaching the upper limits may cause some watery droppings in poultry.

TDS b/w 3000-5000 -- Satisfactory for livestock Unfit for poultry -- Satisfactory for livestock but may be refused by animals not accustomed to it. If Sulphate salts predominate, animals may show temporary diarrhea. Poor waters for poultry, often causing watery faeces, increased mortality and decreasedgrowth especially in turkeys.

TDS b/w 5000-7000 -- Limited use for livestock Unfit for poultry -- This water can be used for livestock except for those that are pregnant or lactating. It may have some laxative effect and may be refused by animals until they become accustomed to it. It is unsatisfactory for poultry.

TDS b/w 7000-10,000 -- Very limited use -- Considerable risk for pregnant and lactating cows, horses, sheep and for the young of these species. It may be used for older ruminants or horses. Unfit for poultry and probably swine.

TDS > 10,000 -- Not recommended -- This water is unsatisfactory for all classes of livestock and poultry.

### **Imputing missing values using median and Linear Regression**

### **Normalisation and Dimentionality Reduction**

### **Model Training ( [Kaggle link](https://www.kaggle.com/code/sivapriyagarladinne/complete-eda-and-model-training))**

### **1. Imputing missing values using median and Linear Regression**

Importing dependencies

In [1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

pd.set\_option('display.max\_columns',None)

from sklearn.decomposition import PCA

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import sklearn.metrics

import math

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.metrics import accuracy\_score

import xgboost as xgb

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

In [2]:

data2018=pd.read\_csv("/kaggle/input/telangana-post-monsoon-ground-water-quality-data/ground\_water\_quality\_2018\_post.csv")

data2019=pd.read\_csv("/kaggle/input/telangana-post-monsoon-ground-water-quality-data/ground\_water\_quality\_2019\_post.csv")

data2020=pd.read\_csv("/kaggle/input/telangana-post-monsoon-ground-water-quality-data/ground\_water\_quality\_2020\_post.csv")

Exploring Data

In [3]:

data2018.shape

Out[3]:

(374, 26)

In [4]:

data2019.shape

Out[4]:

(364, 26)

In [5]:

data2020.shape

Out[5]:

(368, 27)

In [6]:

data2018.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 374 entries, 0 to 373

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sno 374 non-null int64

1 district 374 non-null object

2 mandal 374 non-null object

3 village 374 non-null object

4 lat\_gis 374 non-null float64

5 long\_gis 374 non-null float64

6 gwl 371 non-null float64

7 season 374 non-null object

8 pH 374 non-null float64

9 E.C 374 non-null int64

10 TDS 374 non-null float64

11 CO3 374 non-null int64

12 HCO3 374 non-null float64

13 Cl 374 non-null int64

14 F 374 non-null float64

15 NO3 374 non-null float64

16 SO4 374 non-null float64

17 Na 374 non-null float64

18 K 374 non-null float64

19 Ca 374 non-null int64

20 Mg 374 non-null float64

21 T.H 374 non-null float64

22 SAR 374 non-null float64

23 Classification 374 non-null object

24 RSC meq / L 374 non-null float64

25 Classification.1 374 non-null object

dtypes: float64(15), int64(5), object(6)

memory usage: 76.1+ KB

In [7]:

data2019.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 364 entries, 0 to 363

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sno 364 non-null int64

1 district 364 non-null object

2 mandal 364 non-null object

3 village 364 non-null object

4 lat\_gis 364 non-null float64

5 long\_gis 364 non-null float64

6 gwl 359 non-null float64

7 season 364 non-null object

8 pH 364 non-null float64

9 EC 364 non-null int64

10 TDS 364 non-null float64

11 CO\_-2 204 non-null float64

12 HCO\_ - 364 non-null int64

13 Cl - 364 non-null int64

14 F - 364 non-null float64

15 NO3- 364 non-null float64

16 SO4-2 364 non-null float64

17 Na+ 364 non-null float64

18 K+ 364 non-null float64

19 Ca+2 364 non-null float64

20 Mg+2 364 non-null float64

21 T.H 364 non-null float64

22 SAR 364 non-null float64

23 Classification 364 non-null object

24 RSC meq / L 364 non-null float64

25 Classification.1 364 non-null object

dtypes: float64(16), int64(4), object(6)

memory usage: 74.1+ KB

In [8]:

data2020.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 368 entries, 0 to 367

Data columns (total 27 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sno 368 non-null int64

1 district 368 non-null object

2 mandal 368 non-null object

3 village 368 non-null object

4 lat\_gis 368 non-null float64

5 long\_gis 368 non-null float64

6 gwl 365 non-null float64

7 season 368 non-null object

8 Unnamed: 8 0 non-null float64

9 pH 368 non-null object

10 E.C 368 non-null int64

11 TDS 368 non-null float64

12 CO3 368 non-null float64

13 HCO3 368 non-null int64

14 Cl 368 non-null int64

15 F 368 non-null float64

16 NO3 368 non-null float64

17 SO4 368 non-null float64

18 Na 368 non-null float64

19 K 368 non-null float64

20 Ca 368 non-null float64

21 Mg 368 non-null float64

22 T.H 368 non-null float64

23 SAR 368 non-null float64

24 Classification 368 non-null object

25 RSC meq / L 368 non-null float64

26 Classification.1 368 non-null object

dtypes: float64(16), int64(4), object(7)

memory usage: 77.8+ KB

Check for Null values

In [9]:

data2018.isnull().sum()

Out[9]:

sno 0

district 0

mandal 0

village 0

lat\_gis 0

long\_gis 0

gwl 3

season 0

pH 0

E.C 0

TDS 0

CO3 0

HCO3 0

Cl 0

F 0

NO3 0

SO4 0

Na 0

K 0

Ca 0

Mg 0

T.H 0

SAR 0

Classification 0

RSC meq / L 0

Classification.1 0

dtype: int64

In [10]:

data2019.isnull().sum()

Out[10]:

sno 0

district 0

mandal 0

village 0

lat\_gis 0

long\_gis 0

gwl 5

season 0

pH 0

EC 0

TDS 0

CO\_-2 160

HCO\_ - 0

Cl - 0

F - 0

NO3- 0

SO4-2 0

Na+ 0

K+ 0

Ca+2 0

Mg+2 0

T.H 0

SAR 0

Classification 0

RSC meq / L 0

Classification.1 0

dtype: int64

In [11]:

data2020.isnull().sum()

Out[11]:

sno 0

district 0

mandal 0

village 0

lat\_gis 0

long\_gis 0

gwl 3

season 0

Unnamed: 8 368

pH 0

E.C 0

TDS 0

CO3 0

HCO3 0

Cl 0

F 0

NO3 0

SO4 0

Na 0

K 0

Ca 0

Mg 0

T.H 0

SAR 0

Classification 0

RSC meq / L 0

Classification.1 0

dtype: int64

Renaming Features

In [12]:

column\_name\_mappings={'CO\_-2 ':'CO3', 'HCO\_ - ':'HCO3', 'Cl -':'Cl', 'F -':'F','NO3- ':'NO3 ', 'SO4-2':'SO4', 'Na+':'Na', 'K+':'K', 'Ca+2':'Ca', 'Mg+2':'Mg'

}

data2019.rename(columns=column\_name\_mappings,inplace=True)

In [13]:

data2019.rename(columns={'EC':'E.C'},inplace=True)

In [14]:

data2020.drop("Unnamed: 8",axis=1, inplace=True)

Imputing groundwater level missing values by median grouped by Districts

In [15]:

median\_groundwater\_by\_district\_2018 = data2018.groupby('district')['gwl'].median()

median\_groundwater\_by\_district\_2019 = data2019.groupby('district')['gwl'].median()

median\_groundwater\_by\_district\_2020 = data2020.groupby('district')['gwl'].median()

In [16]:

def impute\_missing\_with\_median(row):

if pd.isnull(row['gwl']):

return median\_groundwater\_by\_district\_2019[row['district']]

else:

return row['gwl']

data2018['gwl'] = data2018.apply(impute\_missing\_with\_median, axis=1)

data2019['gwl'] = data2019.apply(impute\_missing\_with\_median, axis=1)

data2020['gwl'] = data2020.apply(impute\_missing\_with\_median, axis=1)

there were 160 missing values in feature CO3. Lets see for how many districts the values are missing with the total districts

1 total number of unique districts in 2019 data

In [17]:

data2019['district'].value\_counts()

Out[17]:

NALGONDA 30

NIZAMABAD 23

KAMAREDDY 20

MEDAK 19

RANGAREDDY 18

VIKARABAD 16

KHAMMAM 15

YADADRI 15

JAGITYAL 14

BHADRADRI 14

NAGARKURNOOL 12

JOGULAMBA(GADWAL) 11

SANGAREDDY 11

NIRMAL 11

MAHABUBNAGAR 11

WARANGAL (R) 11

NARAYANPET 10

ADILABAD 9

MAHABUBABAD 8

HYDERABAD 8

MEDCHAL 7

JANGAON 7

SIDDIPET 7

SURYAPET 7

WANAPARTHY 7

WARANGAL (U) 7

MANCHERIAL 6

KUMURAM BHEEM 6

SIRCILLA 6

PEDDAPALLY 5

KARIMNAGAR 5

BHUPALPALLY 4

MULUGU 4

Name: district, dtype: int64

In [18]:

*#get the missing value rows and the districts*

missing2019=data2019[data2019['CO3'].isnull()]

missing2019['district'].value\_counts()

Out[18]:

NIZAMABAD 23

NALGONDA 22

KAMAREDDY 20

VIKARABAD 16

KHAMMAM 15

BHADRADRI 10

MAHABUBNAGAR 8

HYDERABAD 7

JOGULAMBA(GADWAL) 7

NARAYANPET 7

SANGAREDDY 7

SIDDIPET 7

MEDCHAL 6

SURYAPET 5

Name: district, dtype: int64

values of CO3 are completely missing for Districts Nizamabad, Kamareddy, Khammam, Vikarabad, Siddipet. For others out of total count only 2-3 are present. Out of 364 rows for 2019, 160 are missing which is43.95% and out of 1106 ( 3 datasets combined) ,160 are missing which is 14.46%

We will check the corelation of this feature within 2018 and also with whole 3 datasets combined. If no significant relation is found lets remove the feature

In [19]:

correl\_2019=data2019.corr()

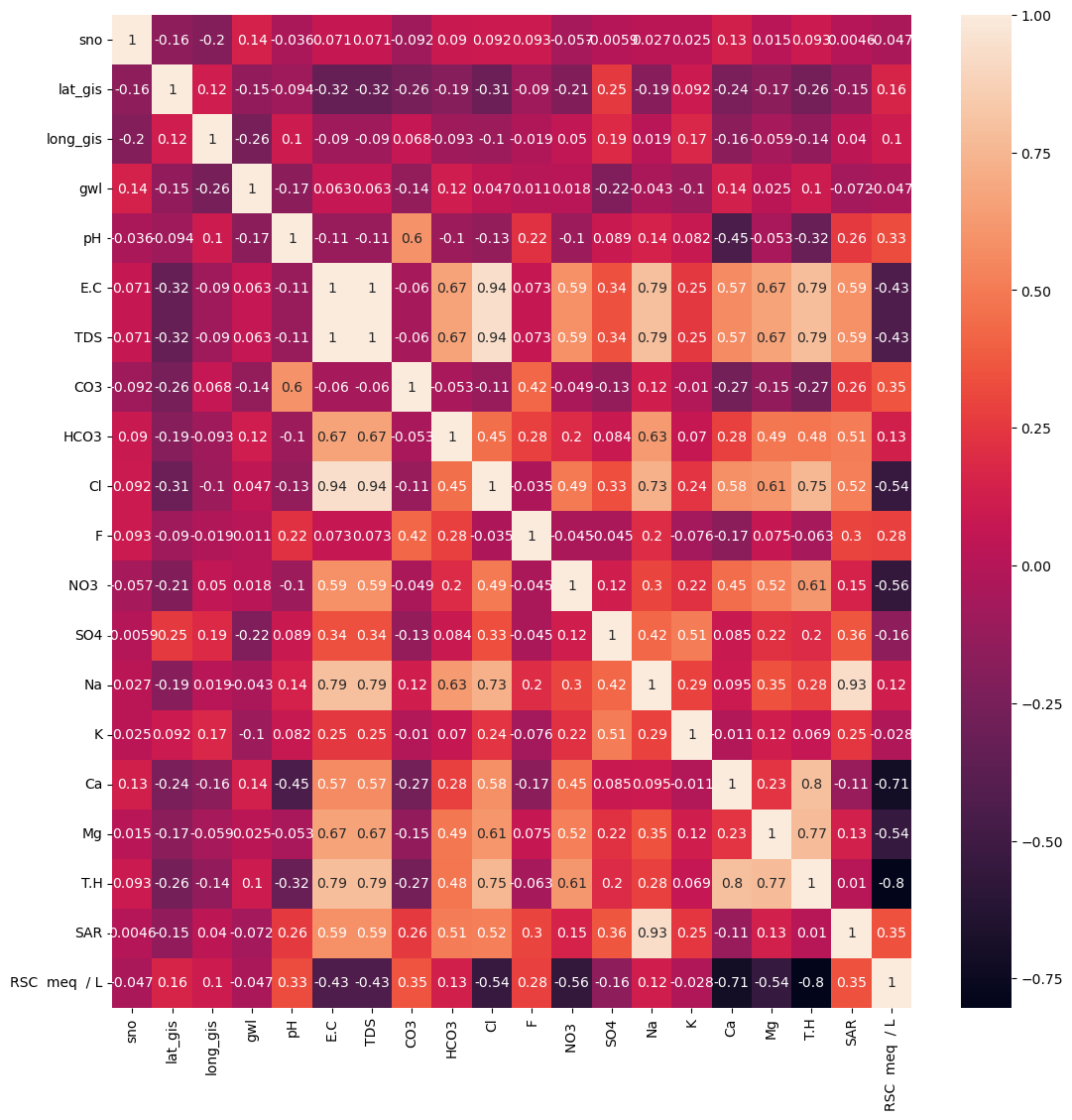
plt.figure(figsize=(13,13))

sns.heatmap(correl\_2019,annot=True)

plt.show()

/tmp/ipykernel\_20/2616675959.py:1: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric\_only to silence this warning.

correl\_2019=data2019.corr()



In [20]:

water\_data=pd.concat([data2018,data2019,data2020])

*#There was a val of ph as 8..05 which had to be changed to 8.05, the process for the same is below.*

water\_data.loc[261,'pH']=8.05

water\_data['pH']=water\_data.loc[:,'pH'].astype(float)

correl=water\_data.corr()

plt.figure(figsize=(13,13))

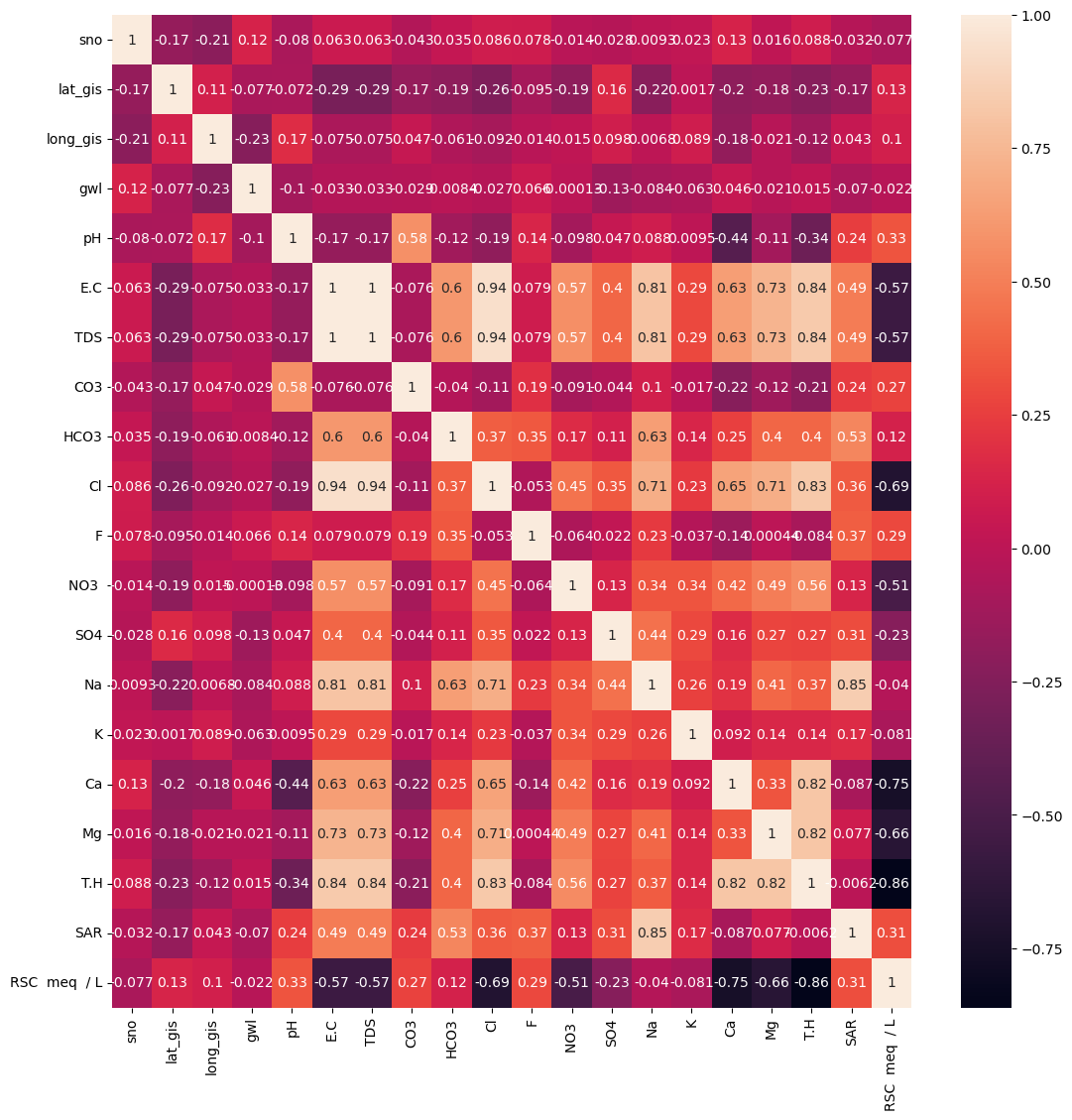
sns.heatmap(correl,annot=True)

/tmp/ipykernel\_20/2378862866.py:7: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric\_only to silence this warning.

correl=water\_data.corr()

Out[20]:

<Axes: >



CO3 is only moderately correlated with only pH values within data 0.6, complete data 0.58

Imputing missing values using a correlated feature as a predictor involves using the correlated feature to predict the missing values in the column with missing data. This can be achieved through a simple linear regression model or any other appropriate imputation technique that leverages the relationship between the correlated features.

Using Linear Regression

In [21]:

*#Getting non missing values of noth columns i.e., pH and CO3 from 2019*

df\_missing=data2019[['pH','CO3']]

df\_subset = df\_missing.dropna(subset=['CO3', 'pH'])

In [22]:

df\_subset.shape

Out[22]:

(204, 2)

In [23]:

*#taking X and y for prediction*

X\_train = df\_subset['pH'].values.reshape(-1, 1)

y\_train = df\_subset['CO3'].values

In [24]:

from sklearn.linear\_model import LinearRegression

*# Create and train the linear regression model*

model = LinearRegression()

model.fit(X\_train, y\_train)

Out[24]:

LinearRegression

LinearRegression()

In [25]:

X\_missing\_values = df\_missing.loc[df\_missing['CO3'].isnull(), 'pH'].values.reshape(-1, 1)

*# Predict the missing values*

predicted\_missing\_values = model.predict(X\_missing\_values)

*# Update the missing values in the original dataset with the predicted values*

df\_missing.loc[df\_missing['CO3'].isnull(), 'CO3'] = predicted\_missing\_values

/tmp/ipykernel\_20/1616279279.py:7: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

df\_missing.loc[df\_missing['CO3'].isnull(), 'CO3'] = predicted\_missing\_values

In [26]:

df\_missing.shape

Out[26]:

(364, 2)

In [27]:

df\_missing.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 364 entries, 0 to 363

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 pH 364 non-null float64

1 CO3 364 non-null float64

dtypes: float64(2)

memory usage: 5.8 KB

In [28]:

data2019['CO3']=df\_missing['CO3'] *#putting predicted CO3 values in data2019*

In [29]:

data2019.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 364 entries, 0 to 363

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sno 364 non-null int64

1 district 364 non-null object

2 mandal 364 non-null object

3 village 364 non-null object

4 lat\_gis 364 non-null float64

5 long\_gis 364 non-null float64

6 gwl 364 non-null float64

7 season 364 non-null object

8 pH 364 non-null float64

9 E.C 364 non-null int64

10 TDS 364 non-null float64

11 CO3 364 non-null float64

12 HCO3 364 non-null int64

13 Cl 364 non-null int64

14 F 364 non-null float64

15 NO3 364 non-null float64

16 SO4 364 non-null float64

17 Na 364 non-null float64

18 K 364 non-null float64

19 Ca 364 non-null float64

20 Mg 364 non-null float64

21 T.H 364 non-null float64

22 SAR 364 non-null float64

23 Classification 364 non-null object

24 RSC meq / L 364 non-null float64

25 Classification.1 364 non-null object

dtypes: float64(16), int64(4), object(6)

memory usage: 74.1+ KB

In [30]:

*#redefining water\_data*

water\_data=pd.concat([data2018,data2019,data2020])

water\_data.loc[261,'pH']=8.05

water\_data['pH']=water\_data.loc[:,'pH'].astype(float)

In [31]:

water\_data.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 1106 entries, 0 to 367

Data columns (total 26 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sno 1106 non-null int64

1 district 1106 non-null object

2 mandal 1106 non-null object

3 village 1106 non-null object

4 lat\_gis 1106 non-null float64

5 long\_gis 1106 non-null float64

6 gwl 1106 non-null float64

7 season 1106 non-null object

8 pH 1106 non-null float64

9 E.C 1106 non-null int64

10 TDS 1106 non-null float64

11 CO3 1106 non-null float64

12 HCO3 1106 non-null float64

13 Cl 1106 non-null int64

14 F 1106 non-null float64

15 NO3 1106 non-null float64

16 SO4 1106 non-null float64

17 Na 1106 non-null float64

18 K 1106 non-null float64

19 Ca 1106 non-null float64

20 Mg 1106 non-null float64

21 T.H 1106 non-null float64

22 SAR 1106 non-null float64

23 Classification 1106 non-null object

24 RSC meq / L 1106 non-null float64

25 Classification.1 1106 non-null object

dtypes: float64(17), int64(3), object(6)

memory usage: 265.6+ KB

Visualisations

In [32]:

*#visualise the correltaion after imputing*

cor=water\_data.corr()

plt.figure(figsize=(13,13))

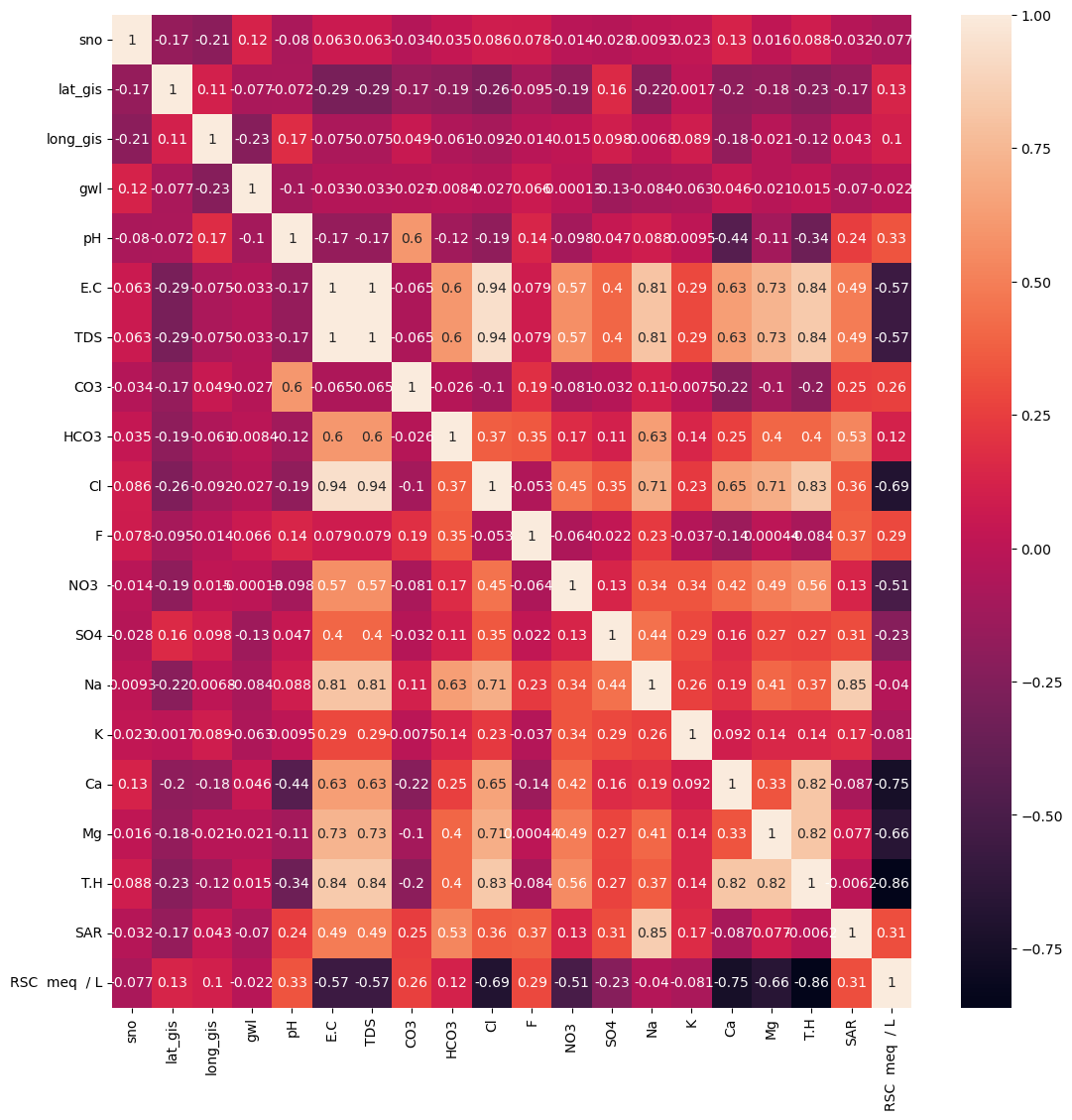
sns.heatmap(cor,annot=True)

/tmp/ipykernel\_20/606254922.py:4: FutureWarning: The default value of numeric\_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric\_only to silence this warning.

cor=water\_data.corr()

Out[32]:

<Axes: >



In [33]:

*#pie charts.*

*#According Standards there are limits to what extent the chemical level in water is safe*

*#By pie charts lets find out, of all the samples how much % of water samples are safe*

def counting(column,ll,ul):

*"""ll- lower limit, ul- upper limit"""*

c=0

c1=0

for i **in** column:

if i>ll **and** i<ul:

c=c+1

else:

c1=c1+1

return c,c1

def plotting(l,tit):

N=["permissible","not permissible"]

plt.pie(l,labels=N,autopct="**%0.1f%%**")

plt.title(tit)

plt.show()

l=counting(water\_data["pH"],6,8.5)

plotting(l,"pH")

l1=counting(water\_data["E.C"],500,3000)

plotting(l1,"E.C")

l2=counting(water\_data["TDS"],0,2000)

plotting(l2,"TDS")

l3=counting(water\_data["T.H"],0,600)

plotting(l3,"TH")

l4=counting(water\_data["Ca"],0,200)

plotting(l4,"Ca")

l5=counting(water\_data["HCO3"],100,800)

plotting(l5,"HCO3")

l6=counting(water\_data["Cl"],0,1000)

plotting(l6,"Cl")

l7=counting(water\_data['Mg'],1,30)

plotting(l7,"Mg")

l8=counting(water\_data['Na'],0,100)

plotting(l8,"Na")

l9=counting(water\_data['K'],0,12)

plotting(l9,"K")

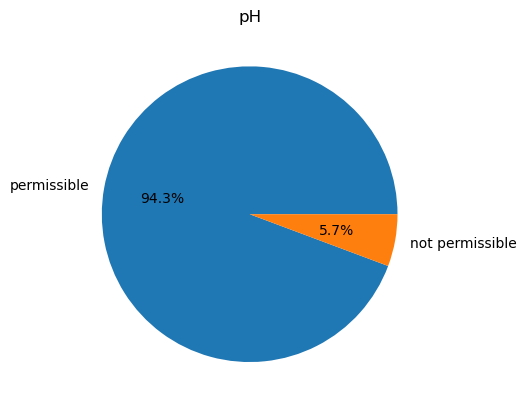
l10=counting(water\_data['SO4'],0,200)

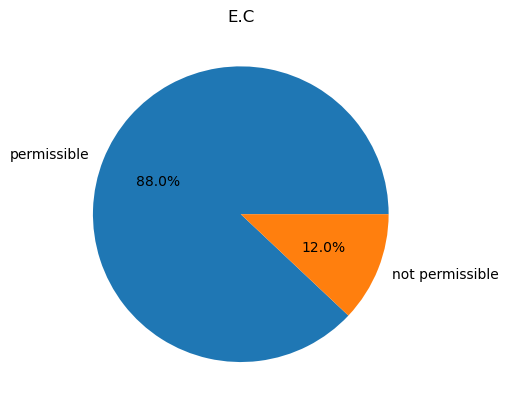
plotting(l10,"SO4")

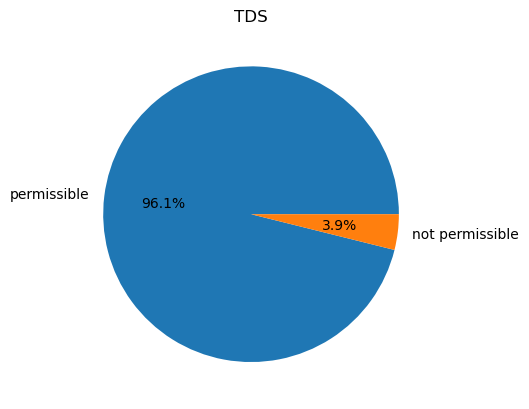
l11=counting(water\_data['CO3'],1.32,17.60)

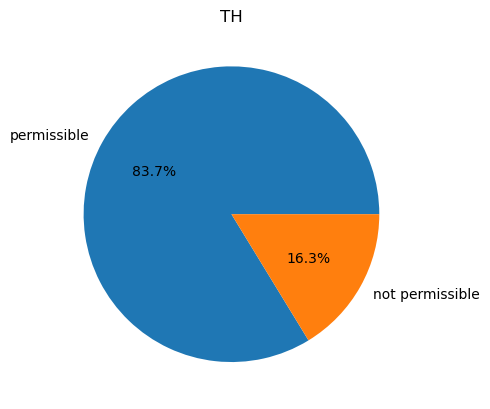
plotting(l11,"CO3")

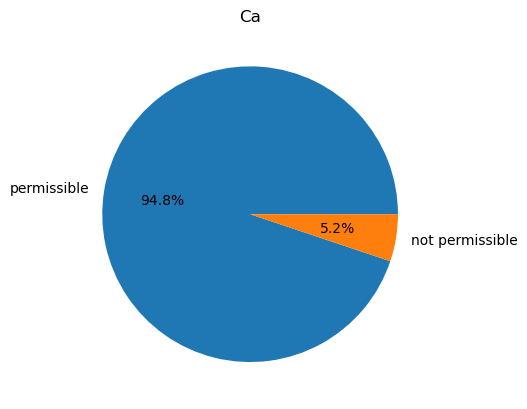
plt.show()

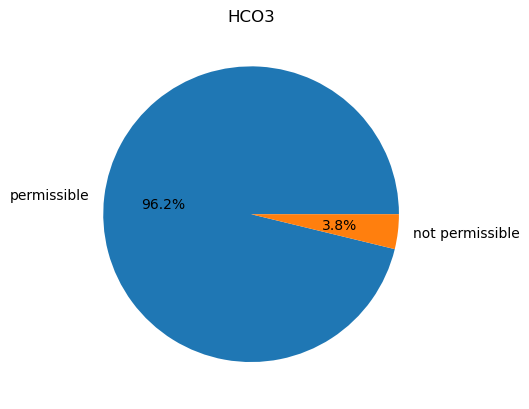


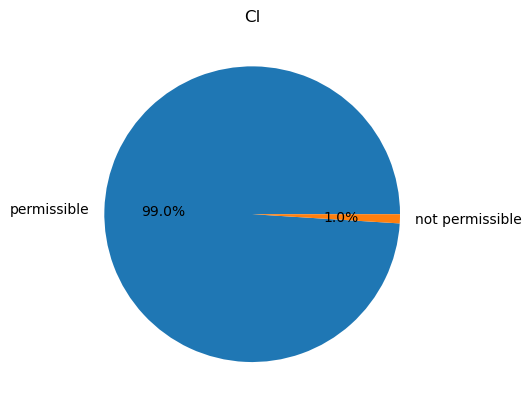


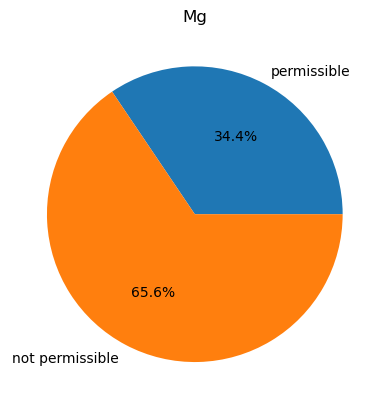


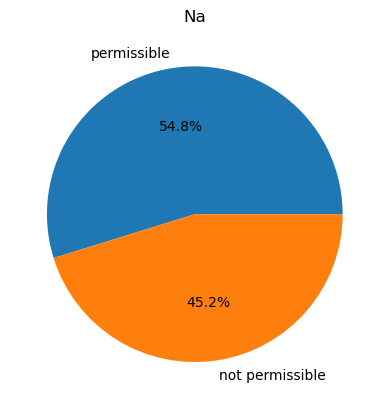


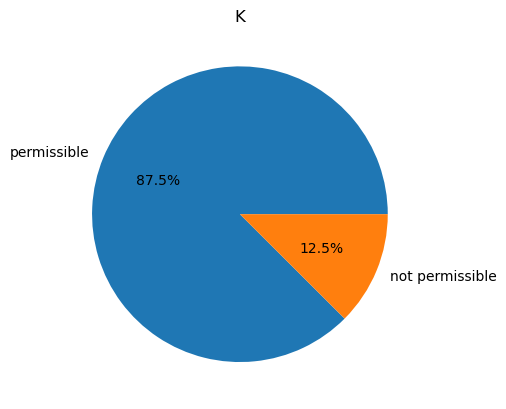


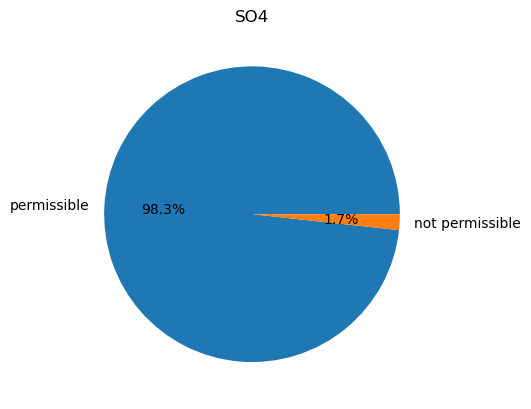


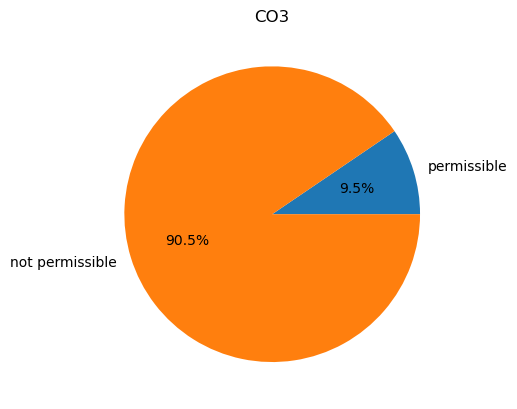












More Visualisations

In [34]:

clm\_tovis=['gwl', 'pH', 'E.C', 'TDS', 'CO3', 'HCO3', 'Cl', 'F', 'NO3 ', 'SO4',

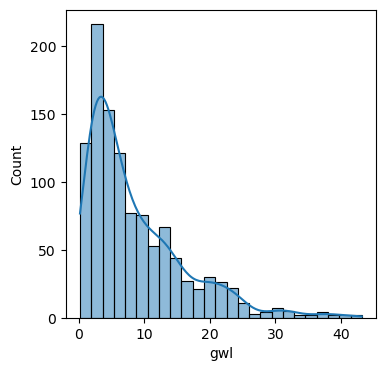
'Na', 'K', 'Ca', 'Mg', 'T.H', 'SAR', 'RSC meq / L']

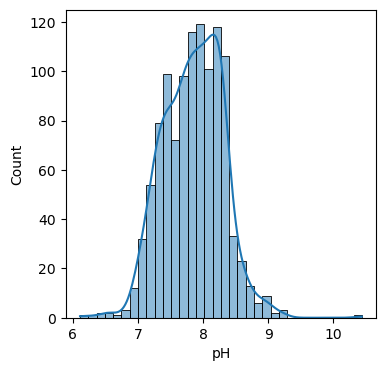
for i **in** clm\_tovis:

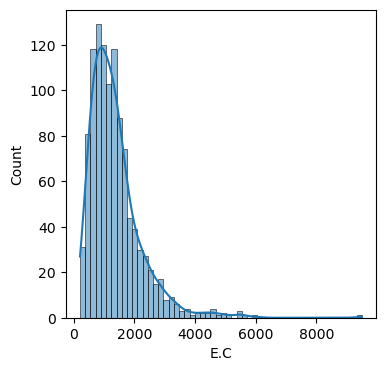
plt.figure(figsize=(4,4))

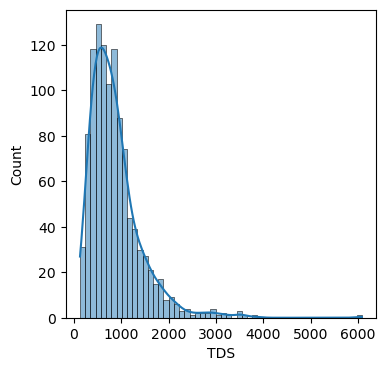
sns.histplot(water\_data[i],kde=True)

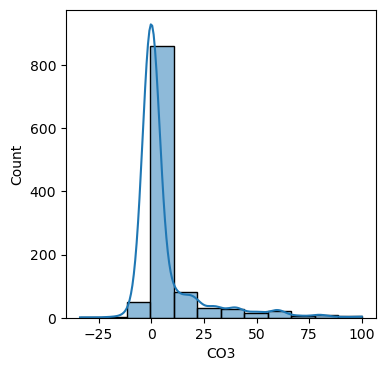
plt.show()

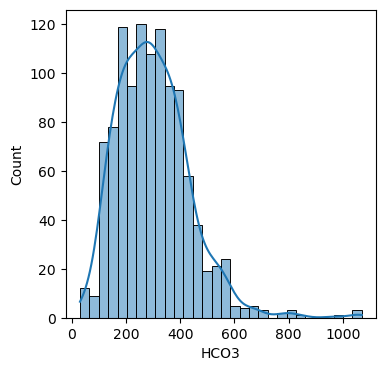


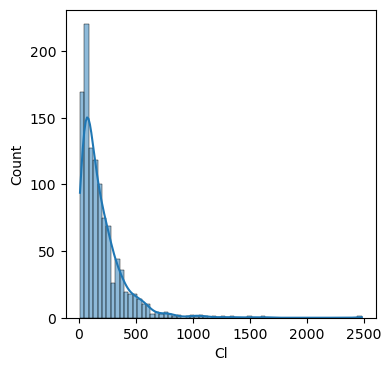


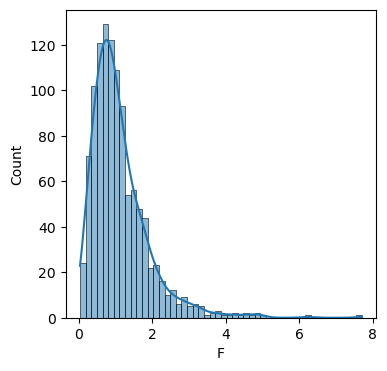


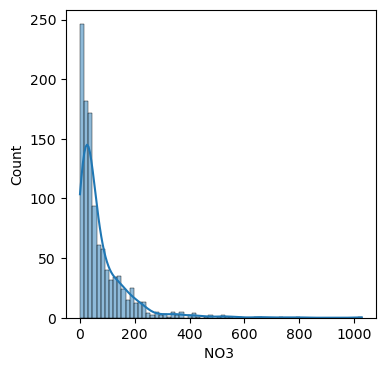


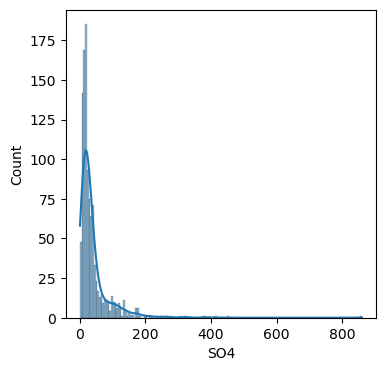


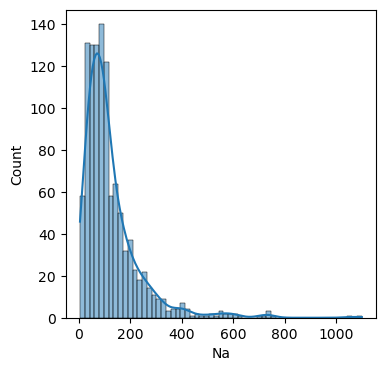


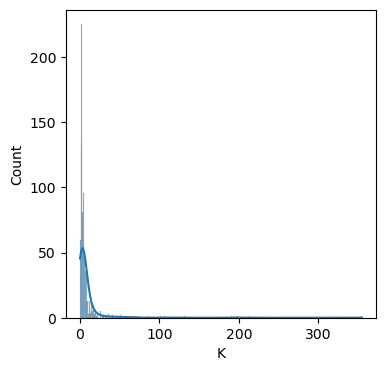


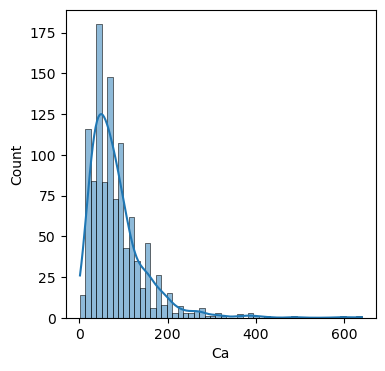


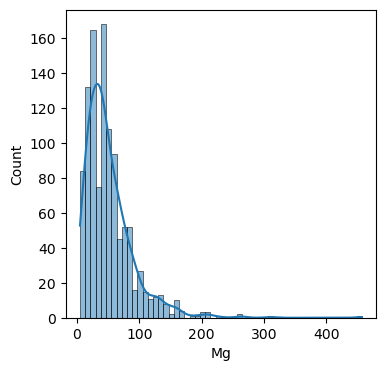


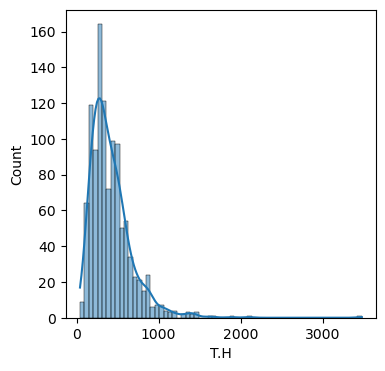


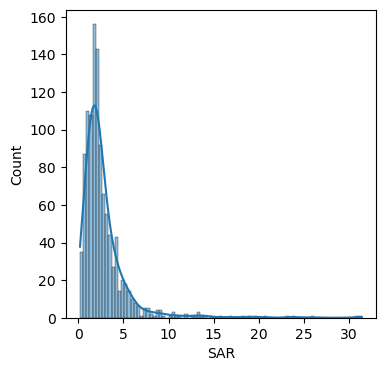


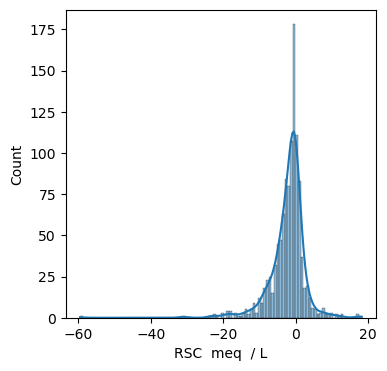












In [35]:

*#Boxplots*

num\_columns=17

fig, axs = plt.subplots(num\_columns, 1, figsize=(6, 4\*num\_columns)) *# Adjust figsize as needed*

*# Loop through each column and create a box plot in its respective subplot*

for i, column\_name **in** enumerate(clm\_tovis):

axs[i].boxplot(water\_data[column\_name])

axs[i].set\_title(column\_name)

*# Add a common x-axis label to all subplots (optional)*

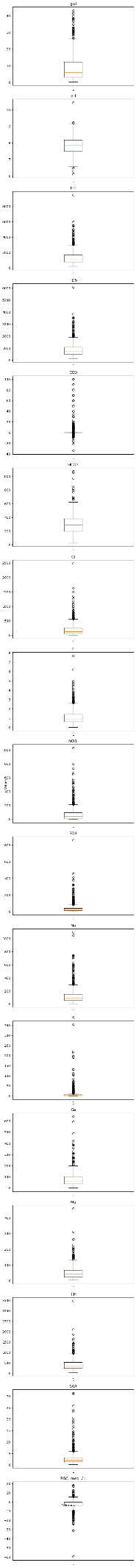
fig.text(0.5, 0.04, 'Columns', ha='center')

*# Add a common y-axis label to all subplots (optional)*

fig.text(0.04, 0.5, 'Values', va='center', rotation='vertical')

plt.tight\_layout() *# To prevent overlapping titles and labels*

plt.show()



In [36]:

water\_data.columns

Out[36]:

Index(['sno', 'district', 'mandal', 'village', 'lat\_gis', 'long\_gis', 'gwl',

'season', 'pH', 'E.C', 'TDS', 'CO3', 'HCO3', 'Cl', 'F', 'NO3 ', 'SO4',

'Na', 'K', 'Ca', 'Mg', 'T.H', 'SAR', 'Classification', 'RSC meq / L',

'Classification.1'],

dtype='object')

In [37]:

*# to train data for prediction of water quality for crops*

Columns\_to\_drop=['sno', 'district', 'mandal', 'village', 'lat\_gis', 'long\_gis','season','RSC meq / L','Classification.1']

Water\_data=water\_data.drop(columns=Columns\_to\_drop)

In [38]:

Water\_data.head()

Out[38]:

|  | gwl | pH | E.C | TDS | CO3 | HCO3 | Cl | F | NO3 | SO4 | Na | K | Ca | Mg | T.H | SAR | Classification |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 5.09 | 8.28 | 745 | 476.80 | 0.0 | 220.0 | 60 | 0.44 | 42.276818 | 46.0 | 49.0 | 4.0 | 48.0 | 38.896 | 279.934211 | 1.273328 | C2S1 |
| 1 | 5.10 | 8.29 | 921 | 589.44 | 0.0 | 230.0 | 80 | 0.56 | 100.659091 | 68.0 | 42.0 | 5.0 | 56.0 | 63.206 | 399.893092 | 0.913166 | C3S1 |
| 2 | 4.98 | 7.69 | 510 | 326.40 | 0.0 | 200.0 | 30 | 0.66 | 41.471545 | 44.0 | 45.0 | 2.0 | 24.0 | 38.896 | 219.934211 | 1.319284 | C2S1 |
| 3 | 5.75 | 8.09 | 422 | 270.08 | 0.0 | 160.0 | 10 | 0.58 | 10.669864 | 35.0 | 27.0 | 1.0 | 32.0 | 19.448 | 159.967105 | 0.928155 | C2S1 |
| 4 | 2.15 | 8.21 | 2321 | 1485.44 | 0.0 | 300.0 | 340 | 2.56 | 128.843636 | 280.0 | 298.0 | 5.0 | 56.0 | 92.378 | 519.843750 | 5.682664 | C4S2 |

### **Normalisation, Dimentionality Reduction And Model Training**

In [39]:

*#Check the distribution of 'classification'*

Water\_data['Classification'].value\_counts()

Out[39]:

C3S1 696

C2S1 248

C4S1 87

C4S2 36

C3S2 12

C4S4 7

C3S3 6

C4S3 5

C1S1 3

OG 2

O.G 2

C3S4 1

C2S2 1

Name: Classification, dtype: int64

The distribution is uneven. So need to Use stratified method for tarining and testing. O.G and OG are not defined will remove them. C3S4 and C2S2 are only 1 which caused lot of error under Stratified method. For now remove them too

In [40]:

Water\_data = Water\_data[~Water\_data['Classification'].isin(['C3S4', 'C2S2','OG','O.G'])]

In [41]:

X=Water\_data.drop(['Classification'],axis=1)

y=Water\_data['Classification']

*#Splitting train test by startified*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=42)

In [42]:

*#Normalising the data, as for dimentionality reduction we need normalised data*

scaler = StandardScaler()

X\_train\_normalized = scaler.fit\_transform(X\_train)

X\_test\_normalized = scaler.transform(X\_test)

In [43]:

*#PCA*

pca = PCA(n\_components=12)

X\_train\_pca = pca.fit\_transform(X\_train\_normalized)

X\_test\_pca = pca.transform(X\_test\_normalized)

In [44]:

pca.explained\_variance\_ratio\_

Out[44]:

array([0.40849434, 0.15754707, 0.0864392 , 0.07664328, 0.0603712 ,

0.04550069, 0.04181629, 0.03975179, 0.03455754, 0.0241867 ,

0.01922031, 0.00444325])

In [45]:

sum(pca.explained\_variance\_ratio\_)

Out[45]:

0.9989716683888861

In [46]:

*#Label Encoding the Target feature*

encoder = LabelEncoder()

*# Fit and transform the training target variable 'y\_train'*

y\_train\_encoded = encoder.fit\_transform(y\_train)

*# Transform the test target variable 'y\_test'*

y\_test\_encoded = encoder.transform(y\_test)

In [47]:

*# 5 fold cross validation*

models=[xgb.XGBClassifier(n\_estimators=1000, max\_depth=5, learning\_rate=0.1, subsample=0.8, colsample\_bytree=0.8),

RandomForestClassifier(n\_estimators=50,random\_state=42)]

num\_folds = 5

skf = StratifiedKFold(n\_splits=num\_folds, shuffle=True, random\_state=42)

*# Perform cross-validation and evaluate models*

def model\_evaluation(model):

fold\_accuracies = []

models = []

for fold, (train\_index, val\_index) **in** enumerate(skf.split(X\_train\_pca, y\_train\_encoded)):

X\_train\_fold, X\_val\_fold = X\_train\_pca[train\_index], X\_train\_pca[val\_index]

y\_train\_fold, y\_val\_fold = y\_train\_encoded[train\_index], y\_train\_encoded[val\_index]

model = model *# Replace with your chosen model and set the appropriate hyperparameters*

model.fit(X\_train\_fold, y\_train\_fold)

*# Add the trained model to the list for later evaluation*

models.append(model)

*# Step 6: Make predictions on the current fold's validation data*

y\_pred\_fold = model.predict(X\_val\_fold)

*# Step 7: Calculate accuracy for the current fold*

accuracy = accuracy\_score(y\_val\_fold, y\_pred\_fold)

fold\_accuracies.append(accuracy)

print(f"Fold **{**fold+1**}** - Accuracy: **{**accuracy**:**.4f**}**")

*# Calculate and print the mean accuracy and standard deviation across all folds*

mean\_accuracy = sum(fold\_accuracies) / len(fold\_accuracies)

std\_accuracy = np.std(fold\_accuracies)

print(f"**\n**Mean Cross-Validation Accuracy: **{**mean\_accuracy**:**.4f**}** ± **{**std\_accuracy**:**.4f**}**")

In [48]:

for i **in** models:

model\_evaluation(i)

/opt/conda/lib/python3.10/site-packages/sklearn/model\_selection/\_split.py:700: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.

warnings.warn(

Fold 1 - Accuracy: 0.8896

Fold 2 - Accuracy: 0.9091

Fold 3 - Accuracy: 0.8961

Fold 4 - Accuracy: 0.9156

Fold 5 - Accuracy: 0.9221

Mean Cross-Validation Accuracy: 0.9065 ± 0.0120

/opt/conda/lib/python3.10/site-packages/sklearn/model\_selection/\_split.py:700: UserWarning: The least populated class in y has only 2 members, which is less than n\_splits=5.

warnings.warn(

Fold 1 - Accuracy: 0.8896

Fold 2 - Accuracy: 0.9221

Fold 3 - Accuracy: 0.8896

Fold 4 - Accuracy: 0.8961

Fold 5 - Accuracy: 0.9091

Mean Cross-Validation Accuracy: 0.9013 ± 0.0126

In [49]:

*#best model = XGBoost*

best\_model = xgb.XGBClassifier(n\_estimators=1000, max\_depth=5, learning\_rate=0.1, subsample=0.8, colsample\_bytree=0.8) *# Replace with the selected best model*

best\_model.fit(X\_train\_pca, y\_train\_encoded)

*# Step 8: Evaluate the final model on the test set*

y\_test\_pred = best\_model.predict(X\_test\_pca)

test\_accuracy = accuracy\_score(y\_test\_encoded, y\_test\_pred)

print(f"**\n**Final Model - Test Accuracy: **{**test\_accuracy**:**.4f**}**")

Final Model - Test Accuracy: 0.9303