NOVEL FEATURE SELECTION METHODS FOR WATER QUALITY DATA

MACHINE LEARNING – CS351

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL.

Done by:

Shashank D – 181CO248 – shashankd.181co248@nitk.edu.in

Tarun S Anur – 181CO255 – tarun.181co255@nitk.edu.in

1. ABSTRACT

Over the past few years, internet has grown widely and has also been able to collect drastically large amount of data. Most of this data is not yet structured as the data collection growth has grown exponentially. Also, there is a lot of unwanted data or redundant data that is also being collected which just adds large computational overheads to the machine learning analysis algorithms used to study them. In this document, we shall be discussing about the various techniques so as to reduce the redundancy of data and also how each feature contributes for the total detailed analysis of the machine learning model.

2. INTRODUCTION

Feature selection is a process of reducing the number of input variables during the development of a predictive model.

This is mostly done in order to reduce the computational costs and in some cases even increase the performance of the model to some extent. Feature selection is primarily focused on removing non-informative or redundant predictors from the model.

Statistical-based feature selection methods involve evaluating the relationship between each input variable and the target variable using statistics and selecting those input variables that have the strongest relationship with the target variable. These methods can be fast and effective, although the choice of statistical measures depends on the data type of both the input and output variables.

We can divide the feature selection methods broadly into three types, namely, **WRAPPER**, **FILTER** and **INTRINSIC** methods:

Wrapper feature selection methods create many models with different subsets of input features and select those features that result in the best performing model according to a given performance metric. These methods are unconcerned with the variable types, although they can be computationally expensive. **RFE (Recursive Feature Elimination)** is a good example of a wrapper feature selection method. Wrapper methods evaluate multiple models using procedures that add and/or remove predictors to find the optimal combination that maximizes model performance.

Filter feature selection methods use statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to choose and filter out those input variables that will be used in the model. Filter methods evaluate the relevance of the predictors outside of the predictive models and subsequently model only the predictors that pass some criterion. **Feature Importance** is the most commonly used filter feature selection method.

Intrinsic feature selection methods where automatic feature selection is performed automatically during the learning of the model. **Decision Tree classifier** is one the commonly used intrinsic feature selection method.

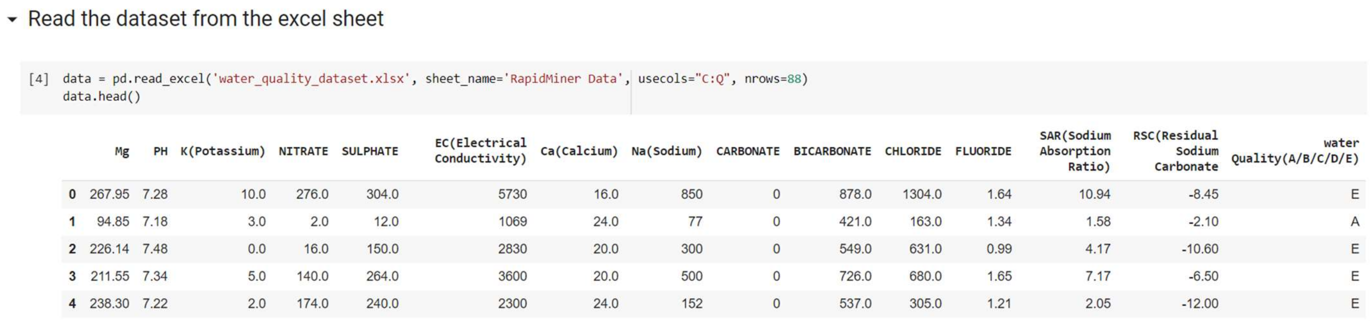
3. DATASET USED AND ITS DESCRIPTION

The dataset used had most of the water quality features needed for the classification. The various places from which the water was obtained was also mentioned. There were total of 88 tuples of data with the Water Quality Type (target feature) mentioned within the dataset. The features included:

- Magnesium (Mg^{2+})
- pH
- Potassium (K^{1+})
- Nitrate (NO_3^-)
- Sulphate (SO_4^{2-})
- Electrical Conductivity (EC)
- Calcium (Ca^{2+})
- Sodium (Na^{1+})
- Carbonate (CO_3^{2-})
- Bicarbonate (HCO_3^{1-})
- Chloride (Cl^{1-})
- Fluoride (F^{1-})
- Sodium Absorption Ratio (SAR)
- Residual Sodium Carbonate (RSC)

Water Quality type is basically of 5 types –

A. Drinking Water Source without conventional treatment but after disinfection
B. Outdoor Bathing (Organised)
C. Drinking Water Source after conventional treatment and disinfection
D. Propagation of Wildlife and Fisheries
E. Irrigation, Industrial Cooling, Controlled waste disposal



▾ Read the dataset from the excel sheet

```
[4] data = pd.read_excel('water_quality_dataset.xlsx', sheet_name='RapidMiner Data', usecols="C:Q", nrows=88)
    data.head()
```

	Mg	PH	K(Potassium)	NITRATE	SULPHATE	EC(Electrical Conductivity)	Ca(Calcium)	Na(Sodium)	CARBONATE	BICARBONATE	CHLORIDE	FLUORIDE	SAR(Sodium Absorption Ratio)	RSC(Residual Sodium Carbonate	water Quality(A/B/C/D/E)
0	267.95	7.28	10.0	276.0	304.0	5730	16.0	850	0	878.0	1304.0	1.64	10.94	-8.45	E
1	94.85	7.18	3.0	2.0	12.0	1069	24.0	77	0	421.0	163.0	1.34	1.58	-2.10	A
2	226.14	7.48	0.0	16.0	150.0	2830	20.0	300	0	549.0	631.0	0.99	4.17	-10.60	E
3	211.55	7.34	5.0	140.0	264.0	3600	20.0	500	0	726.0	680.0	1.65	7.17	-6.50	E
4	238.30	7.22	2.0	174.0	240.0	2300	24.0	152	0	537.0	305.0	1.21	2.05	-12.00	E

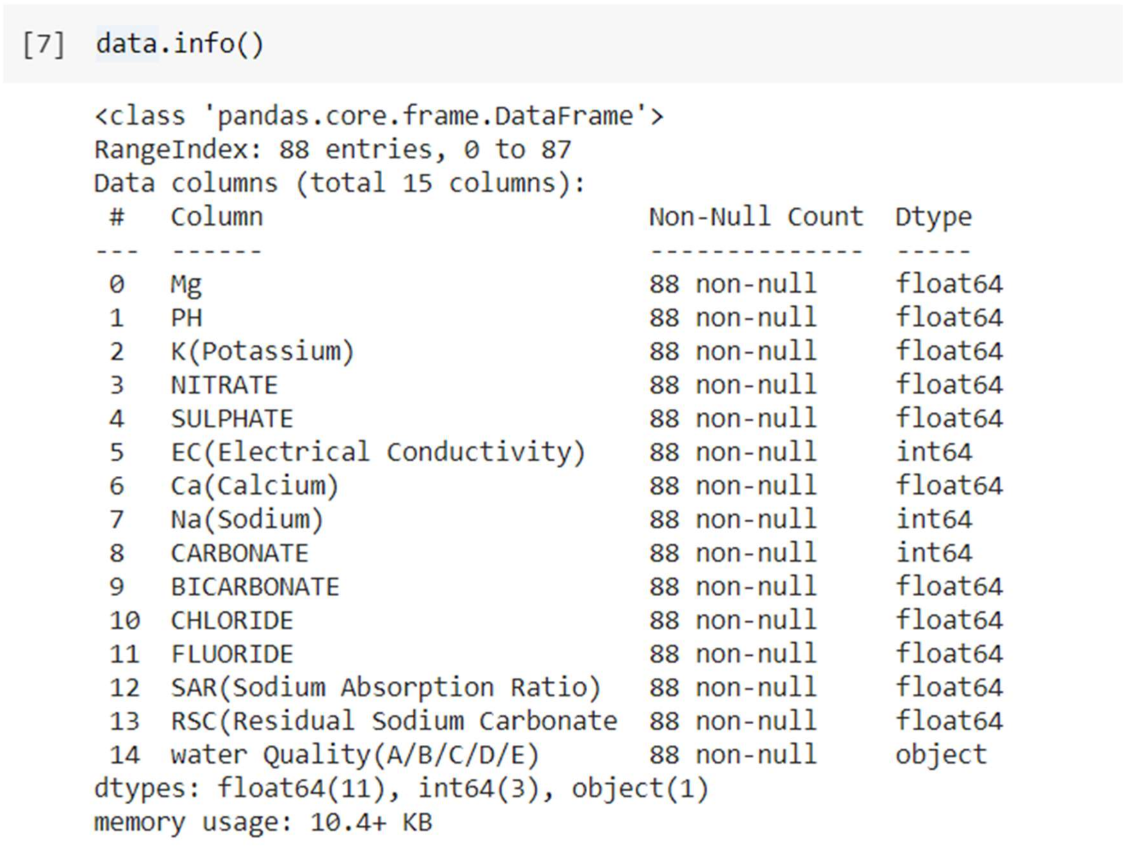
A little bit of pre-processing had to be done as one of the features had negative values associated with it. Also, we renamed some of the columns to access it easily.

Preprocessing

Cleaning data to meet Algorithms' requirements

```
[5] data['RSC(Residual Sodium Carbonate'] = data['RSC(Residual Sodium Carbonate'] - pd.DataFrame.min(data['RSC(Residual Sodium Carbonate'])
    data.describe()
```

	Mg	PH	K(Potassium)	NITRATE	SULPHATE	EC(Electrical Conductivity)	Ca(Calcium)	Na(Sodium)	CARBONATE	BICARBONATE	CHLORIDE	FLUORIDE	SAR(Sodium Absorption Ratio)	RSC(Residual Sodium Carbonate
count	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000	88.000000
mean	94.135089	7.683078	61.097427	22.434025	159.811496	2406.261364	100.642684	289.965909	4.750000	409.433644	482.641104	0.871853	4.779150	29.095838
std	76.780583	0.435190	73.617429	51.948068	133.507601	1757.151584	84.905110	260.322675	14.952223	181.804402	505.067125	0.442557	3.424933	7.498522
min	0.000000	6.820000	0.000000	0.000000	3.000000	346.000000	2.000000	20.000000	0.000000	98.000000	25.000000	0.020000	0.470000	0.000000
25%	35.965000	7.377500	3.000000	2.750000	57.250000	995.250000	39.000000	103.250000	0.000000	268.000000	113.000000	0.509000	2.342500	25.425000
50%	75.025000	7.605000	40.000000	10.000000	169.626593	2090.000000	80.000000	271.000000	0.000000	402.462302	339.000000	0.988176	4.622381	30.655000
75%	113.745000	7.812500	70.759255	18.079712	172.219945	3090.000000	123.500000	352.500000	4.000000	499.750000	588.250000	1.100000	5.697500	34.620000
max	350.140000	8.860000	315.000000	350.000000	744.000000	7440.000000	520.000000	1300.000000	96.000000	970.000000	2183.000000	2.280000	16.750000	40.880000



```
[7] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88 entries, 0 to 87
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Mg                              88 non-null     float64
 1   PH                              88 non-null     float64
 2   K(Potassium)                    88 non-null     float64
 3   NITRATE                         88 non-null     float64
 4   SULPHATE                        88 non-null     float64
 5   EC(Electrical Conductivity)     88 non-null     int64
 6   Ca(Calcium)                     88 non-null     float64
 7   Na(Sodium)                      88 non-null     int64
 8   CARBONATE                       88 non-null     int64
 9   BICARBONATE                     88 non-null     float64
 10  CHLORIDE                        88 non-null     float64
 11  FLUORIDE                        88 non-null     float64
 12  SAR(Sodium Absorption Ratio)    88 non-null     float64
 13  RSC(Residual Sodium Carbonate   88 non-null     float64
 14  water Quality(A/B/C/D/E)        88 non-null     object
dtypes: float64(11), int64(3), object(1)
memory usage: 10.4+ KB
```

4. DIFFERENT FEATURE SELECTION METHODS USED:

- UNIVARIATE SELECTION USING CHI-SQUARE TEST

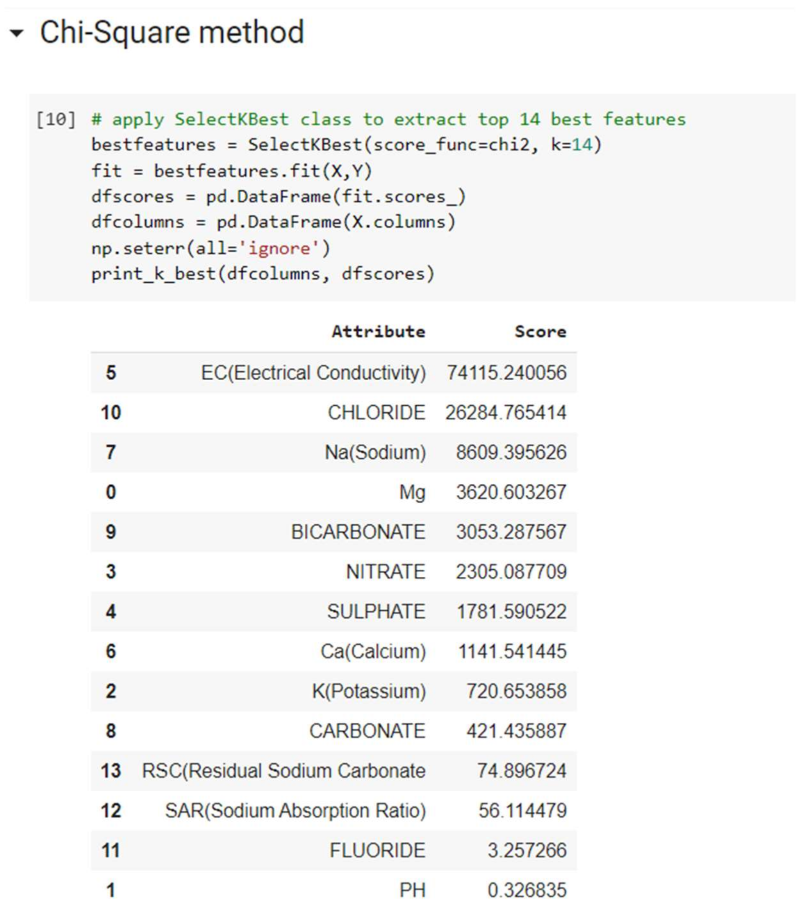
This method computes chi-squared stats between each non-negative feature and class. This score can then be used to select the n_features with the highest values for the test chi-squared statistic from X, which must contain only non-negative features such as Booleans or frequencies relative to the classes. It is a statistical test applied to sets of categorical data to evaluate how likely it is that any observed difference between the sets arose by chance. The chi-square score is calculated by the below formula:

$$X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

O_i = observed frequency counts in each category

E_i = expected frequency counts in each category

k = number of categories



Chi-Square method

```
[10] # apply SelectKBest class to extract top 14 best features
     bestfeatures = SelectKBest(score_func=chi2, k=14)
     fit = bestfeatures.fit(X,Y)
     dfscores = pd.DataFrame(fit.scores_)
     dfcolumns = pd.DataFrame(X.columns)
     np.seterr(all='ignore')
     print_k_best(dfcolumns, dfscores)
```

	Attribute	Score
5	EC(Electrical Conductivity)	74115.240056
10	CHLORIDE	26284.765414
7	Na(Sodium)	8609.395626
0	Mg	3620.603267
9	BICARBONATE	3053.287567
3	NITRATE	2305.087709
4	SULPHATE	1781.590522
6	Ca(Calcium)	1141.541445
2	K(Potassium)	720.653858
8	CARBONATE	421.435887
13	RSC(Residual Sodium Carbonate	74.896724
12	SAR(Sodium Absorption Ratio)	56.114479
11	FLUORIDE	3.257266
1	PH	0.326835

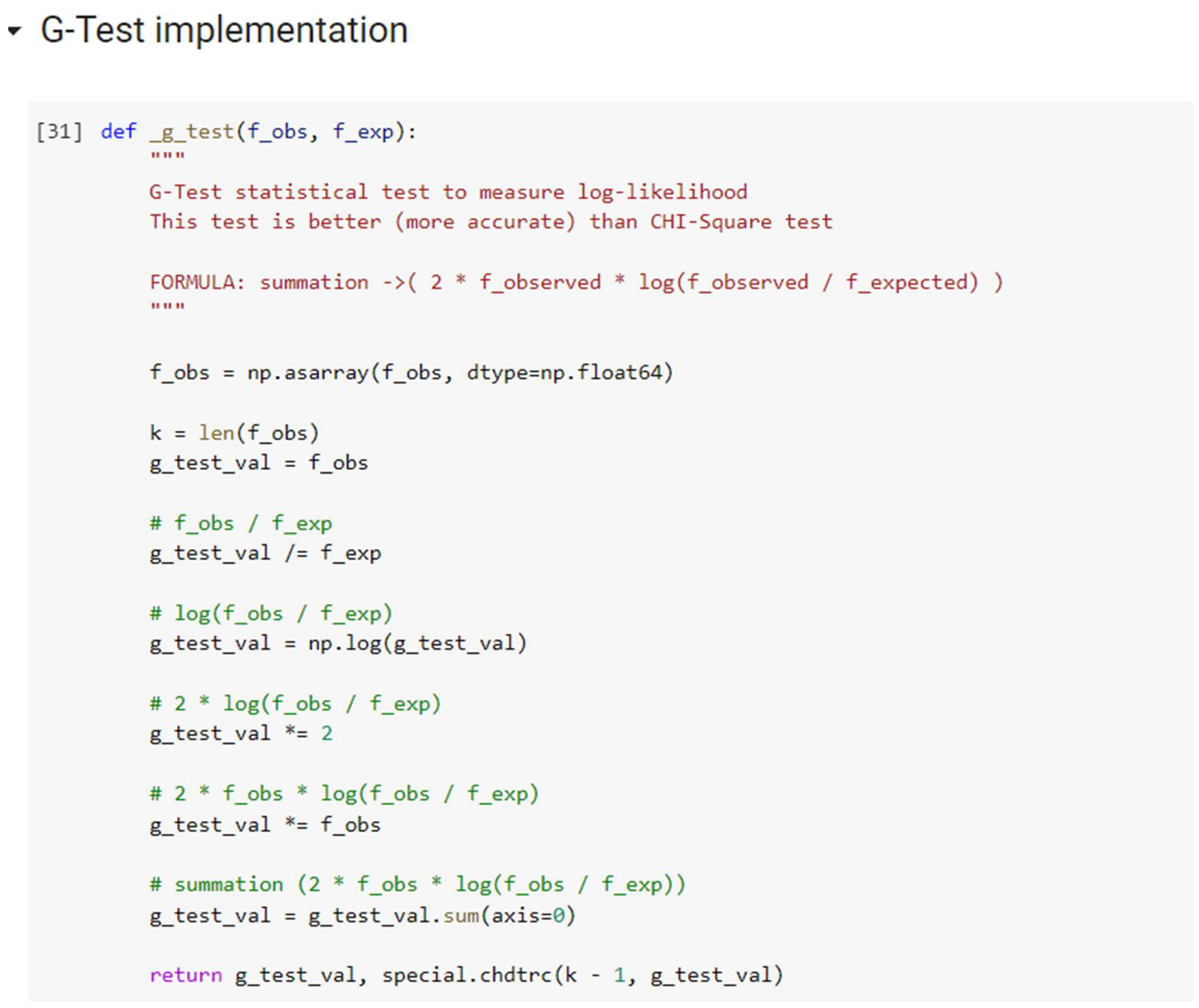
- UNIVARIATE SELECTION USING THE G-TEST

G_test is another statistical method by which we can find the feature importance. It is considered to be an improvement over the chi-squared test for small datasets. Using some approximations and assumptions, g_test can be proved as a stronger case over the chi-square test. For large datasets, other likelihood tests approach g_test values. It is also faster that the chi-square test as it involves logarithmic operations which is faster than the squaring operation. The G_test score is calculated by the below formula:

$$G = 2\sum_i O_i \cdot \ln\left(\frac{O_i}{E_i}\right),$$

Where, O_i = Observed frequency counts in each category

E_i = Expected frequency counts in each category



G-Test implementation

```
[31] def _g_test(f_obs, f_exp):
         """
         G-Test statistical test to measure log-likelihood
         This test is better (more accurate) than CHI-Square test

         FORMULA: summation ->( 2 * f_observed * log(f_observed / f_expected) )
         """

         f_obs = np.asarray(f_obs, dtype=np.float64)

         k = len(f_obs)
         g_test_val = f_obs

         # f_obs / f_exp
         g_test_val /= f_exp

         # log(f_obs / f_exp)
         g_test_val = np.log(g_test_val)

         # 2 * log(f_obs / f_exp)
         g_test_val *= 2

         # 2 * f_obs * log(f_obs / f_exp)
         g_test_val *= f_obs

         # summation (2 * f_obs * log(f_obs / f_exp))
         g_test_val = g_test_val.sum(axis=0)

         return g_test_val, special.chdtrc(k - 1, g_test_val)
```

Data checking for G-Test

[12]

```
def g_test(X, y):
    X = check_array(X, accept_sparse='csr')
    if np.any((X.data if issparse(X) else X) < 0):
        raise ValueError("Input X must be non-negative.")

    Y = LabelBinarizer().fit_transform(y)
    if Y.shape[1] == 1:
        Y = np.append(1 - Y, Y, axis=1)

    observed = safe_sparse_dot(Y.T, X)          # n_classes * n_features

    feature_count = X.sum(axis=0).reshape(1, -1)
    class_prob = Y.mean(axis=0).reshape(1, -1)
    expected = np.dot(class_prob.T, feature_count)

    return _g_test(observed, expected)
```

```
from scipy import special, stats
from scipy.sparse import issparse

from sklearn.base import BaseEstimator
from sklearn.preprocessing import LabelBinarizer
from sklearn.utils import (as_float_array, check_array, check_X_y, safe_sqr, safe_mask)
from sklearn.utils.extmath import safe_sparse_dot, row_norms
from sklearn.utils.validation import check_is_fitted

class Gtest_best_features(BaseEstimator):
    def __init__(self, score_func=g_test, *, k=10):
        self.k = k
        self.score_func = score_func

    def fit(self, X, y):

        if not callable(self.score_func):
            raise TypeError("G-Test function should be a callable, %s (%s) was passed." % (self.score_func, type(self.score_func)))

        self._check_params(X, y)
        score_func_ret = self.score_func(X, y)
        if isinstance(score_func_ret, (list, tuple)):
            self.scores_, self.pvalues_ = score_func_ret
            self.pvalues_ = np.asarray(self.pvalues_)
        else:
            self.scores_ = score_func_ret
            self.pvalues_ = None

        self.scores_ = np.asarray(self.scores_)

        return self

    def _check_params(self, X, y):
        if not (self.k == "all" or 0 <= self.k <= X.shape[1]):
            raise ValueError("k should be >=0")
```

```
[16] gtest_bestfeatures = Gtest_best_features()
     fit = gtest_bestfeatures.fit(X,Y)
     dfscores = pd.DataFrame(fit.scores_)
     dfcolumns = pd.DataFrame(X.columns)
     featrs = print_k_best(dfcolumns, dfscores)

     X_new = data[featrs['Attribute'][:8]]
     featrs
     # check_accuracy(X_new)
```

	Attribute	Score
3	NITRATE	5.168294
10	CHLORIDE	4.009121
5	EC(Electrical Conductivity)	2.993360
0	Mg	2.669468
7	Na(Sodium)	2.658597
2	K(Potassium)	2.409474
4	SULPHATE	2.136053
12	SAR(Sodium Absorption Ratio)	1.753163
6	Ca(Calcium)	1.243513
9	BICARBONATE	0.739006
11	FLUORIDE	0.737073
1	PH	-0.034115
13	RSC(Residual Sodium Carbonate	-0.274251

Mean accuracy of Decisicion Tree without feature Selection

```
[17] from sklearn.model_selection import cross_val_score
     clf = DecisionTreeClassifier()

     print(np.mean(cross_val_score(clf, X, Y, cv = 4)*100))

     53.40909090909091
```

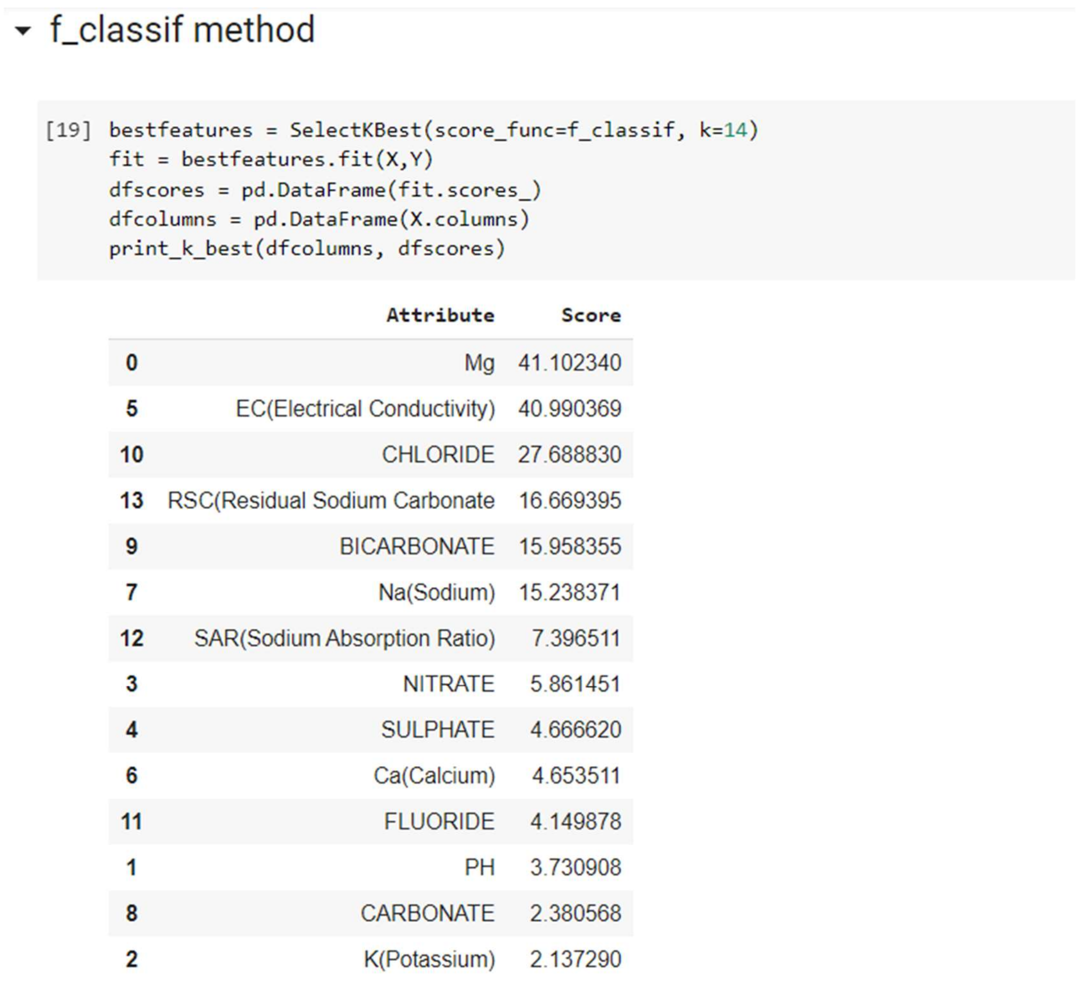
Mean accuracy of Decisicion Tree with feature Selection

```
[18] clf1 = DecisionTreeClassifier()

     ## Using X_new (i.e the top features obtained from G-test)

     print(np.mean(cross_val_score(clf1, X_new, Y, cv=4)*100))

     59.090909090909086
```

- UNIVARIATE SELECTION USING F_CLASSIF METHOD (ANOVA TEST)

ANOVA (Analysis of Variance) is another method to evaluate the dependence of two variables. ANOVA assumes a linear relationship between the variables and the target, and also that the variables are normally distributed. ANOVA is a collection of statistical models and their associated estimation procedures used to analyse the differences among means. ANOVA is based on the law of total variance, where the observed variance in a particular variable is partitioned into components attributable to different sources of variation.

$$SS = \sum x_i^{\,2} - \frac{(\sum x_i)^2}{N}$$



▾ f_classif method

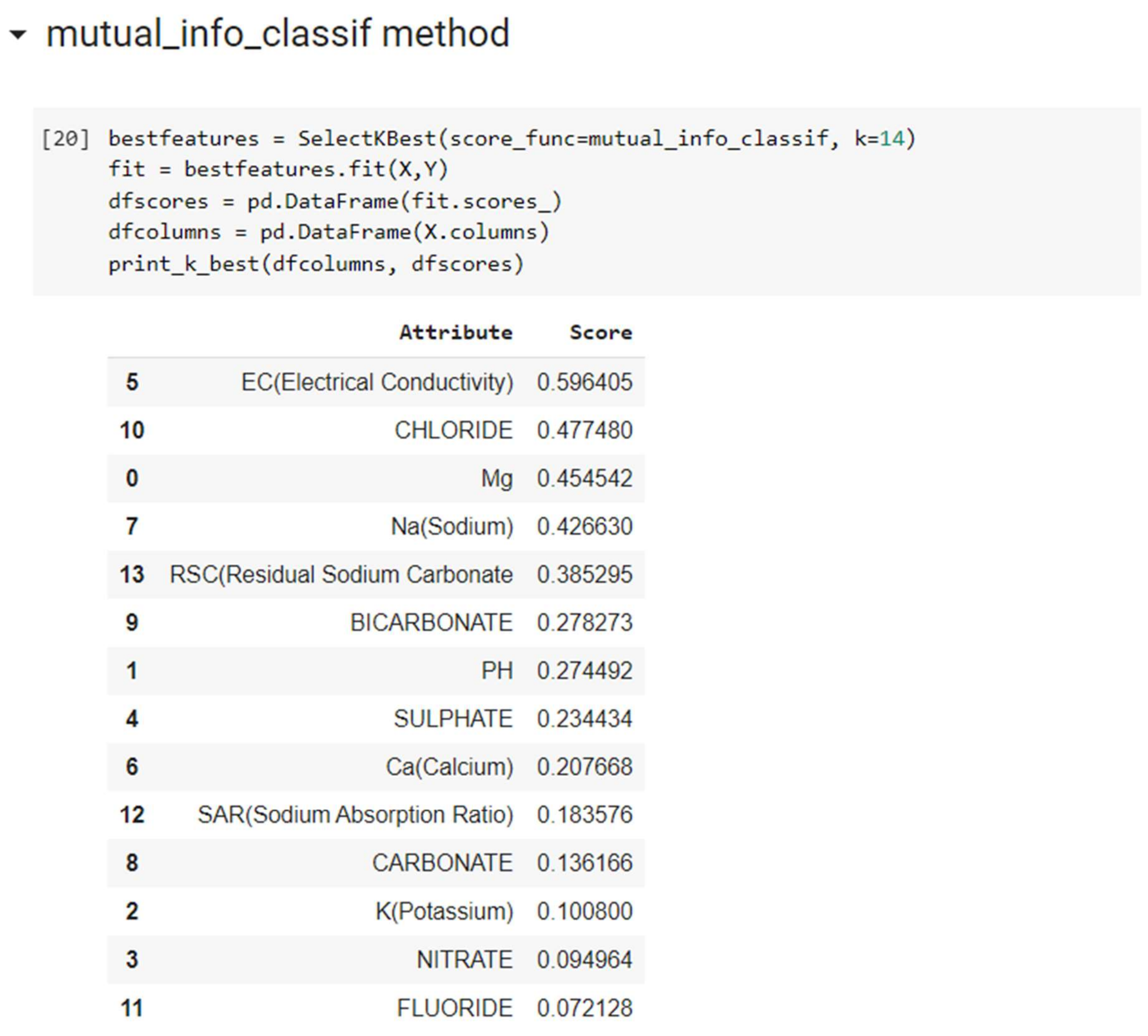
```
[19] bestfeatures = SelectKBest(score_func=f_classif, k=14)
     fit = bestfeatures.fit(X,Y)
     dfscores = pd.DataFrame(fit.scores_)
     dfcolumns = pd.DataFrame(X.columns)
     print_k_best(dfcolumns, dfscores)
```

	Attribute	Score
0	Mg	41.102340
5	EC(Electrical Conductivity)	40.990369
10	CHLORIDE	27.688830
13	RSC(Residual Sodium Carbonate	16.669395
9	BICARBONATE	15.958355
7	Na(Sodium)	15.238371
12	SAR(Sodium Absorption Ratio)	7.396511
3	NITRATE	5.861451
4	SULPHATE	4.666620
6	Ca(Calcium)	4.653511
11	FLUORIDE	4.149878
1	PH	3.730908
8	CARBONATE	2.380568
2	K(Potassium)	2.137290

- UNIVARIATE SELECTION USING MUTUAL_INFO_CLASSIF METHOD

This method estimates Mutual Information (MI) for a discrete target variable. It quantifies the "amount of information" obtained about one random variable through observing the other random variable. MI between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. MI measures how much information the presence/absence of a feature X contributes to making the correct prediction on Y. If X is deterministic of Y, then MI is the entropy of X, which is a notion in information theory that measures or quantifies the amount of information within a variable. True MI can never be negative. In case it turns out negative, it is replaced by 0.

$$I(X;Y) = \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)}$$



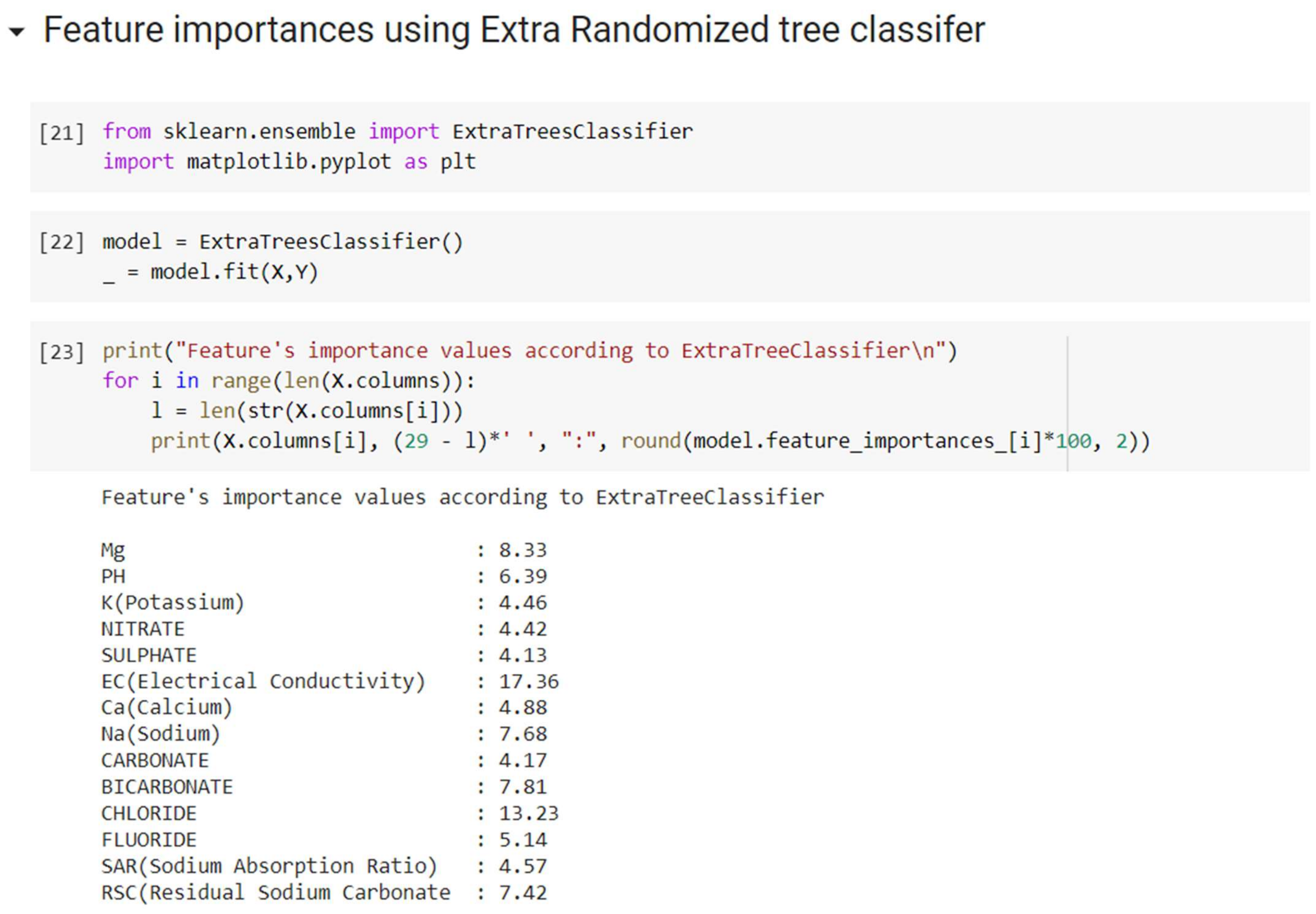
▾ mutual_info_classif method

```
[20] bestfeatures = SelectKBest(score_func=mutual_info_classif, k=14)
     fit = bestfeatures.fit(X,Y)
     dfscores = pd.DataFrame(fit.scores_)
     dfcolumns = pd.DataFrame(X.columns)
     print_k_best(dfcolumns, dfscores)
```

	Attribute	Score
5	EC(Electrical Conductivity)	0.596405
10	CHLORIDE	0.477480
0	Mg	0.454542
7	Na(Sodium)	0.426630
13	RSC(Residual Sodium Carbonate	0.385295
9	BICARBONATE	0.278273
1	PH	0.274492
4	SULPHATE	0.234434
6	Ca(Calcium)	0.207668
12	SAR(Sodium Absorption Ratio)	0.183576
8	CARBONATE	0.136166
2	K(Potassium)	0.100800
3	NITRATE	0.094964
11	FLUORIDE	0.072128

- FEATURE IMPORTANCES USING EXTRA RANDOMIZED TREE CLASSIFIER

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees. This class implements a meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Extra-trees differ from classic decision trees in the way they are built. When looking for the best split to separate the samples of a node into two groups, random splits are drawn for each of the max_features randomly selected features and the best split among those is chosen.



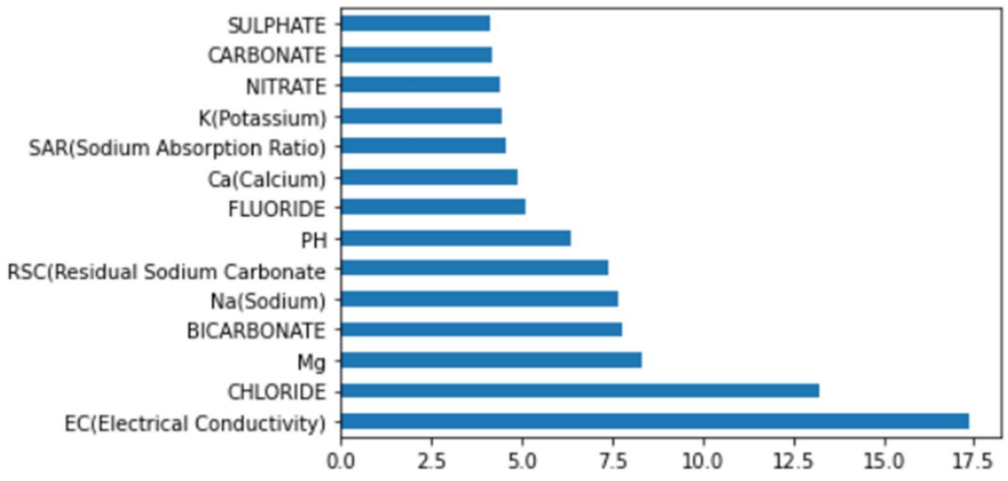
Feature importances using Extra Randomized tree classifer

```
[21] from sklearn.ensemble import ExtraTreesClassifier
     import matplotlib.pyplot as plt
```

```
[22] model = ExtraTreesClassifier()
     _ = model.fit(X,Y)
```

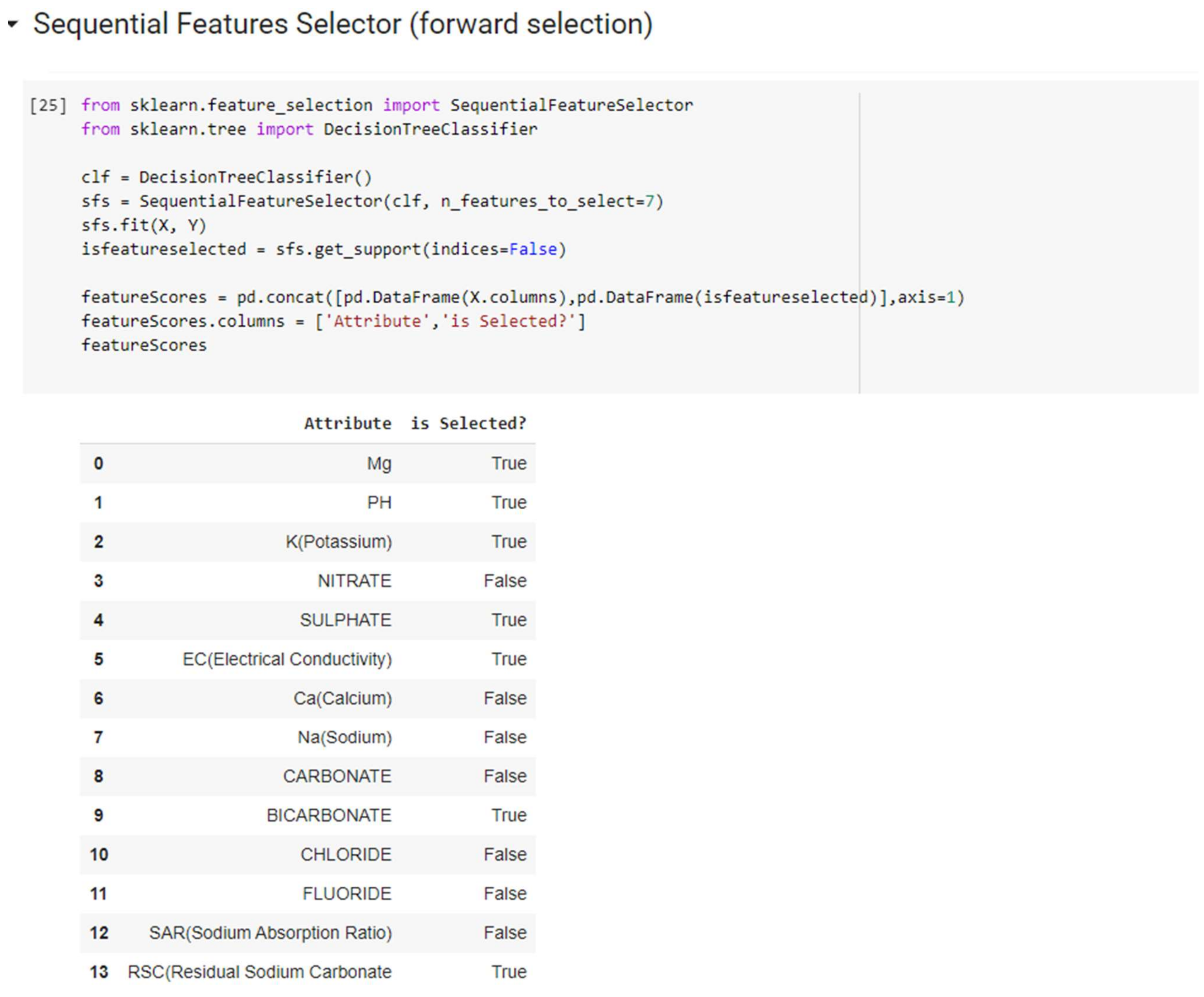
```
[23] print("Feature's importance values according to ExtraTreeClassifier\n")
     for i in range(len(X.columns)):
         l = len(str(X.columns[i]))
         print(X.columns[i], (29 - l)*' ', ":", round(model.feature_importances_[i]*100, 2))
```

```
Feature's importance values according to ExtraTreeClassifier

Mg                              : 8.33
PH                              : 6.39
K(Potassium)                    : 4.46
NITRATE                         : 4.42
SULPHATE                        : 4.13
EC(Electrical Conductivity)     : 17.36
Ca(Calcium)                     : 4.88
Na(Sodium)                      : 7.68
CARBONATE                       : 4.17
BICARBONATE                     : 7.81
CHLORIDE                        : 13.23
FLUORIDE                        : 5.14
SAR(Sodium Absorption Ratio)    : 4.57
RSC(Residual Sodium Carbonate   : 7.42
[24] # plot graph of feature importances for better visualization
     feat_importances = pd.Series(model.feature_importances_*100, index=X.columns)
     feat_importances.nlargest(14).plot(kind='barh')
     plt.show()
```



- SEQUENTIAL FEATURES SELECTOR (FORWARD SELECTION)

This Sequential Feature Selector adds (forward selection) features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator (decision tree classifier). Forward-SFS is a greedy procedure that iteratively finds the best new feature to add to the set of selected features. Concretely, we initially start with zero feature and find the one feature that maximizes a cross-validated score when an estimator is trained on this single feature. Once that first feature is selected, we repeat the procedure by adding a new feature to the set of selected features. The procedure stops when the desired number of selected features is reached.



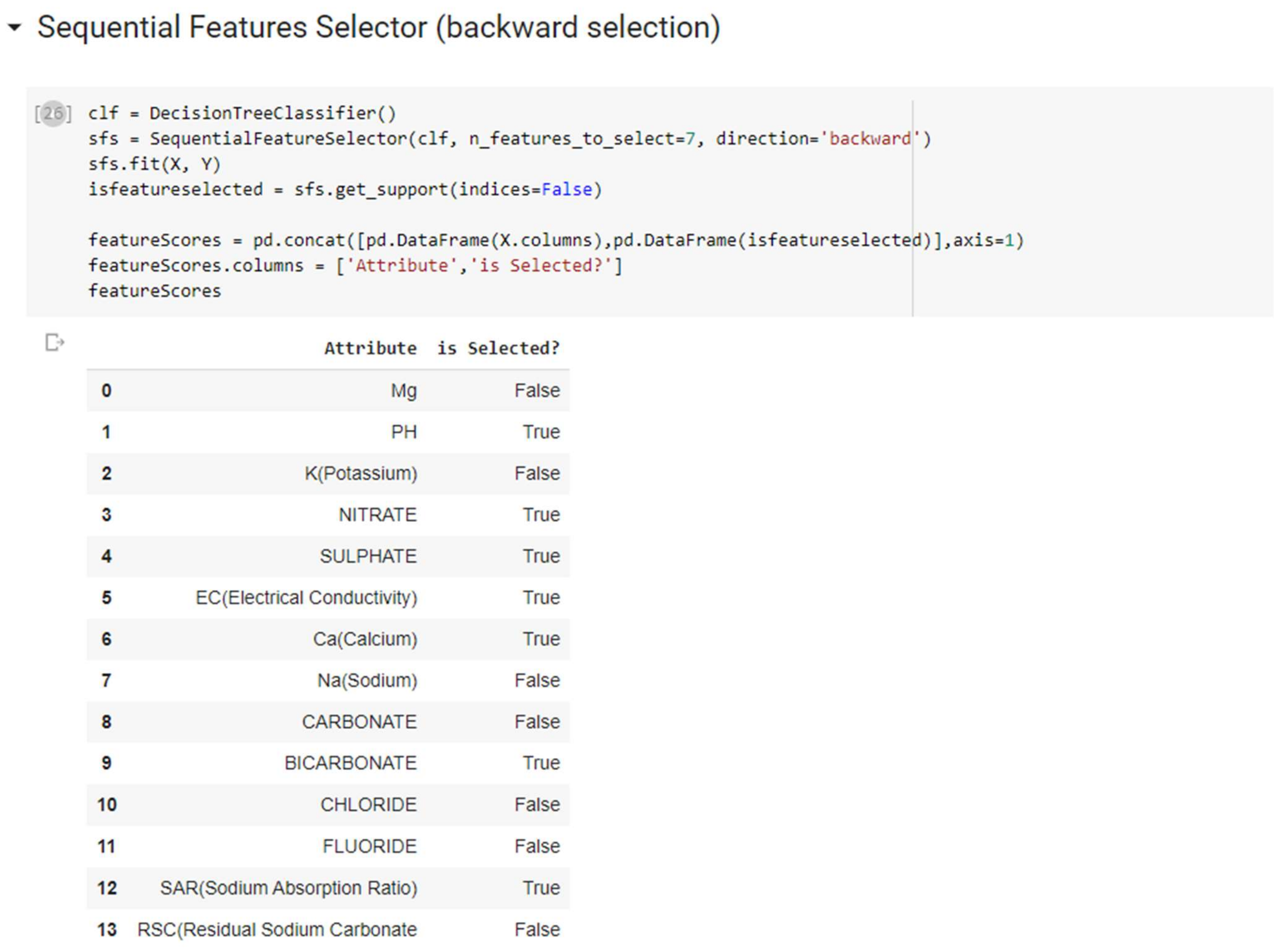
Sequential Features Selector (forward selection)

```
[25] from sklearn.feature_selection import SequentialFeatureSelector
     from sklearn.tree import DecisionTreeClassifier

     clf = DecisionTreeClassifier()
     sfs = SequentialFeatureSelector(clf, n_features_to_select=7)
     sfs.fit(X, Y)
     isfeatureselected = sfs.get_support(indices=False)

     featureScores = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(isfeatureselected)],axis=1)
     featureScores.columns = ['Attribute','is Selected?']
     featureScores
```

	Attribute	is Selected?
0	Mg	True
1	PH	True
2	K(Potassium)	True
3	NITRATE	False
4	SULPHATE	True
5	EC(Electrical Conductivity)	True
6	Ca(Calcium)	False
7	Na(Sodium)	False
8	CARBONATE	False
9	BICARBONATE	True
10	CHLORIDE	False
11	FLUORIDE	False
12	SAR(Sodium Absorption Ratio)	False
13	RSC(Residual Sodium Carbonate	True

- SEQUENTIAL FEATURES SELECTOR (BACKWARD SELECTION)

This Sequential Feature Selector removes (backward selection) features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to remove based on the cross-validation score of an estimator (decision tree classifier). Backward-SBS is a greedy procedure that iteratively finds the best new feature to remove from the set of selected features. Concretely, we initially start with all features and find the one feature that minimizes a cross-validated score when an estimator is trained using all these features. Once that first feature is removed, we repeat the procedure by removing a feature to the set of selected features. This feature can be also described as the least significant feature among the remaining available ones. The procedure stops when the desired number of selected features is reached.



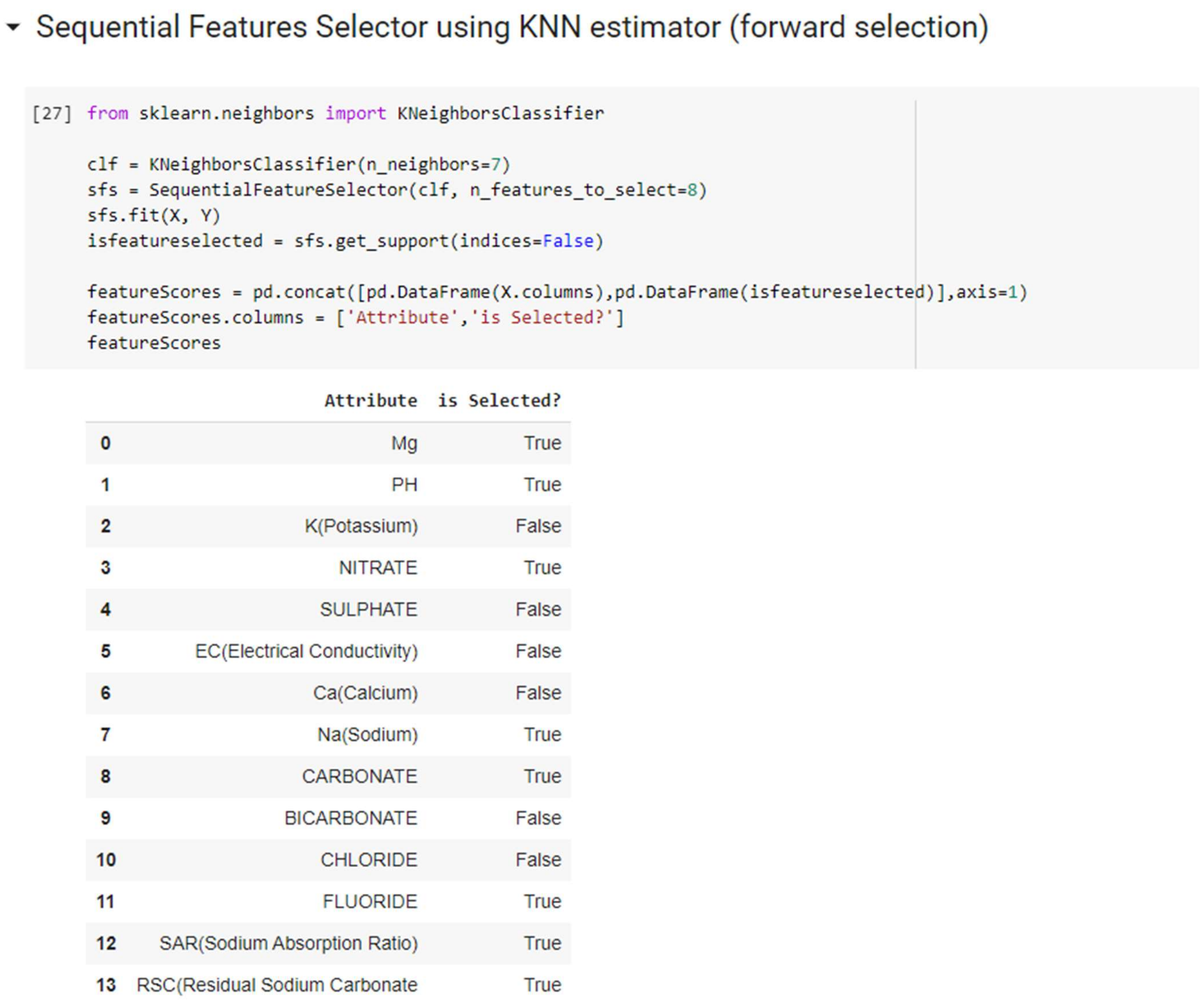
Sequential Features Selector (backward selection)

```
[26] clf = DecisionTreeClassifier()
     sfs = SequentialFeatureSelector(clf, n_features_to_select=7, direction='backward')
     sfs.fit(X, Y)
     isfeatureselected = sfs.get_support(indices=False)

     featureScores = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(isfeatureselected)],axis=1)
     featureScores.columns = ['Attribute','is Selected?']
     featureScores
```

	Attribute	is Selected?
0	Mg	False
1	PH	True
2	K(Potassium)	False
3	NITRATE	True
4	SULPHATE	True
5	EC(Electrical Conductivity)	True
6	Ca(Calcium)	True
7	Na(Sodium)	False
8	CARBONATE	False
9	BICARBONATE	True
10	CHLORIDE	False
11	FLUORIDE	False
12	SAR(Sodium Absorption Ratio)	True
13	RSC(Residual Sodium Carbonate	False

- SEQUENTIAL FEATURES SELECTOR USING KNN ESTIMATOR (FORWARD SELECTION)

This Sequential Feature Selector adds (forward selection) features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator (K – Nearest Neighbours). Forward-SFS is a greedy procedure that iteratively finds the best new feature to add to the set of selected features. Concretely, we initially start with zero feature and find the one feature that maximizes a cross-validated score when an estimator is trained on this single feature. Once that first feature is selected, we repeat the procedure by adding a new feature to the set of selected features. The procedure stops when the desired number of selected features is reached.



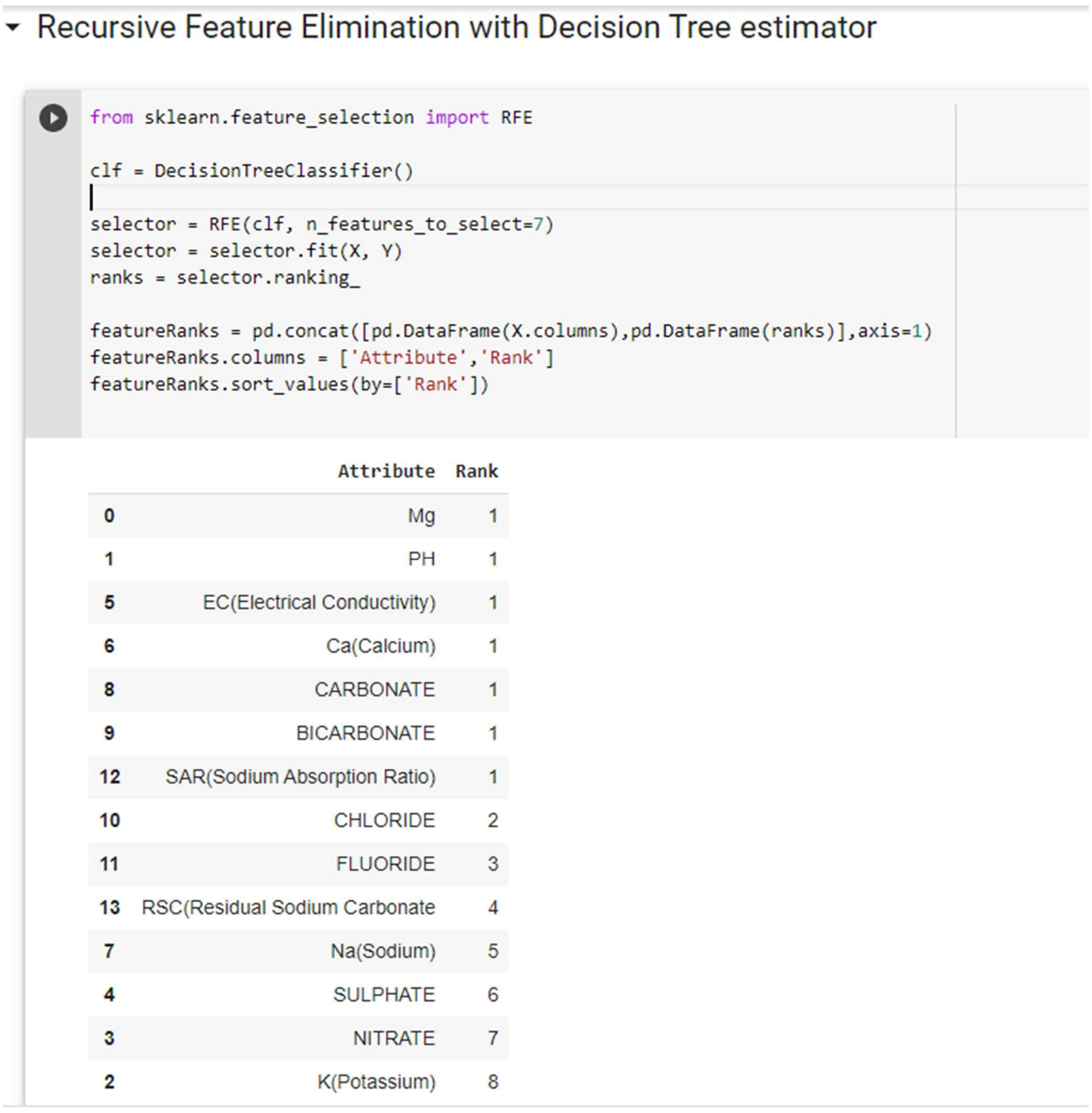
Sequential Features Selector using KNN estimator (forward selection)

```
[27] from sklearn.neighbors import KNeighborsClassifier

     clf = KNeighborsClassifier(n_neighbors=7)
     sfs = SequentialFeatureSelector(clf, n_features_to_select=8)
     sfs.fit(X, Y)
     isfeatureselected = sfs.get_support(indices=False)

     featureScores = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(isfeatureselected)],axis=1)
     featureScores.columns = ['Attribute','is Selected?']
     featureScores
```

	Attribute	is Selected?
0	Mg	True
1	PH	True
2	K(Potassium)	False
3	NITRATE	True
4	SULPHATE	False
5	EC(Electrical Conductivity)	False
6	Ca(Calcium)	False
7	Na(Sodium)	True
8	CARBONATE	True
9	BICARBONATE	False
10	CHLORIDE	False
11	FLUORIDE	True
12	SAR(Sodium Absorption Ratio)	True
13	RSC(Residual Sodium Carbonate	True

- RECURSIVE FEATURE ELIMINATION WITH DECISION TREE ESTIMATOR

The goal of Recursive Feature Elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. RFE works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains. This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.



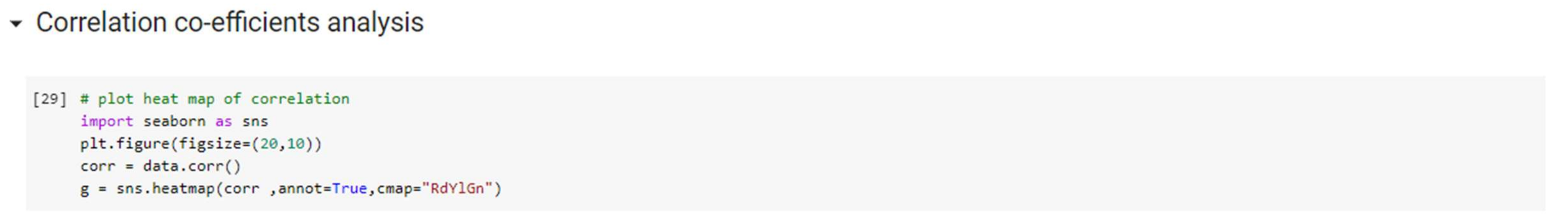
Recursive Feature Elimination with Decision Tree estimator

```
from sklearn.feature_selection import RFE

clf = DecisionTreeClassifier()

selector = RFE(clf, n_features_to_select=7)
selector = selector.fit(X, Y)
ranks = selector.ranking_

featureRanks = pd.concat([pd.DataFrame(X.columns),pd.DataFrame(ranks)],axis=1)
featureRanks.columns = ['Attribute','Rank']
featureRanks.sort_values(by=['Rank'])
```

	Attribute	Rank
0	Mg	1
1	PH	1
5	EC(Electrical Conductivity)	1
6	Ca(Calcium)	1
8	CARBONATE	1
9	BICARBONATE	1
12	SAR(Sodium Absorption Ratio)	1
10	CHLORIDE	2
11	FLUORIDE	3
13	RSC(Residual Sodium Carbonate	4
7	Na(Sodium)	5
4	SULPHATE	6
3	NITRATE	7
2	K(Potassium)	8

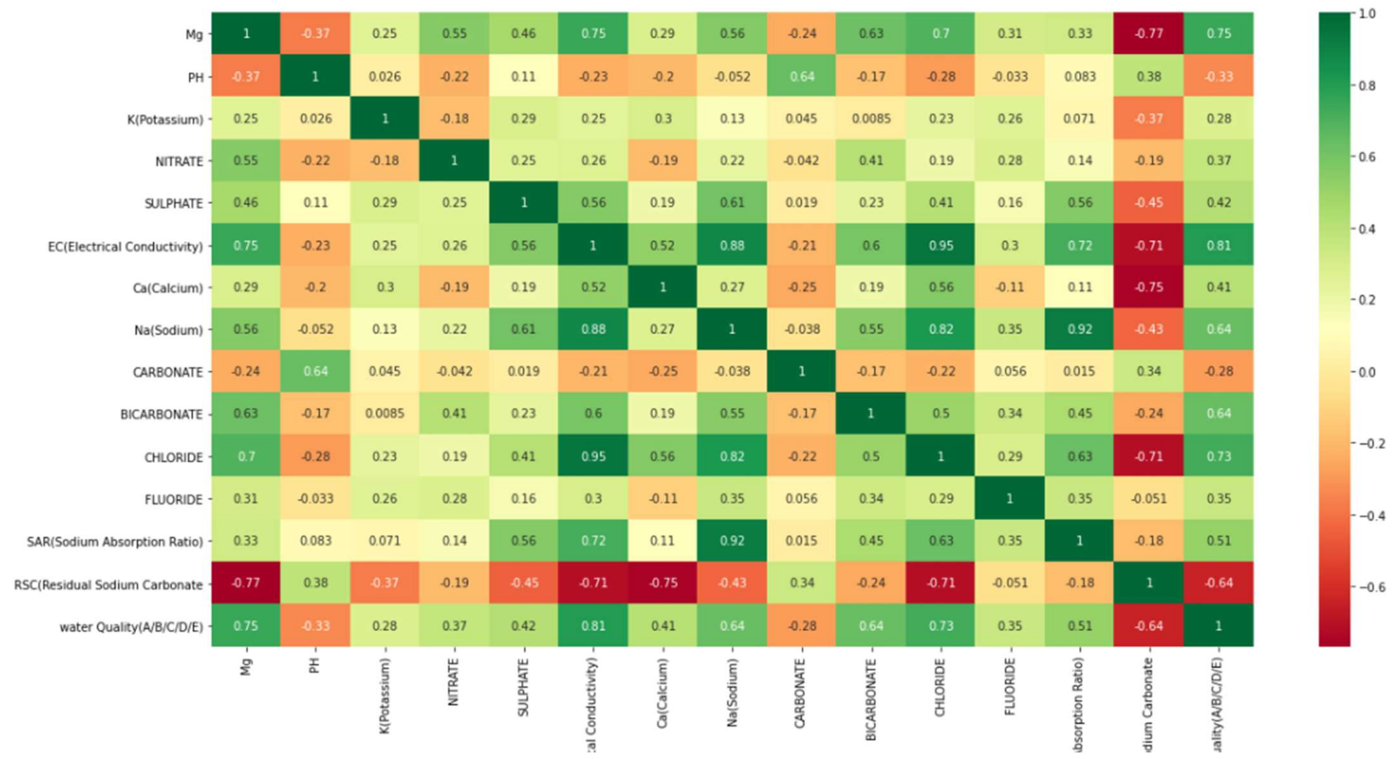
- CORRELATION CO-EFFICIENTS ANALYSIS

A correlation co-efficient is a numerical measure of some type of correlation, meaning a statistical relationship between two variables. It is used to find the pairwise correlation of all columns in the dataframe. They all assume values in the range from −1 to +1, where ±1 indicates the strongest possible agreement and 0 the strongest possible disagreement. The Pearson correlation coefficient, is a measure of the strength and direction of the linear relationship between two variables that is defined as the covariance of the variables divided by the product of their standard deviations. This is the best-known and most commonly used type of correlation coefficient.



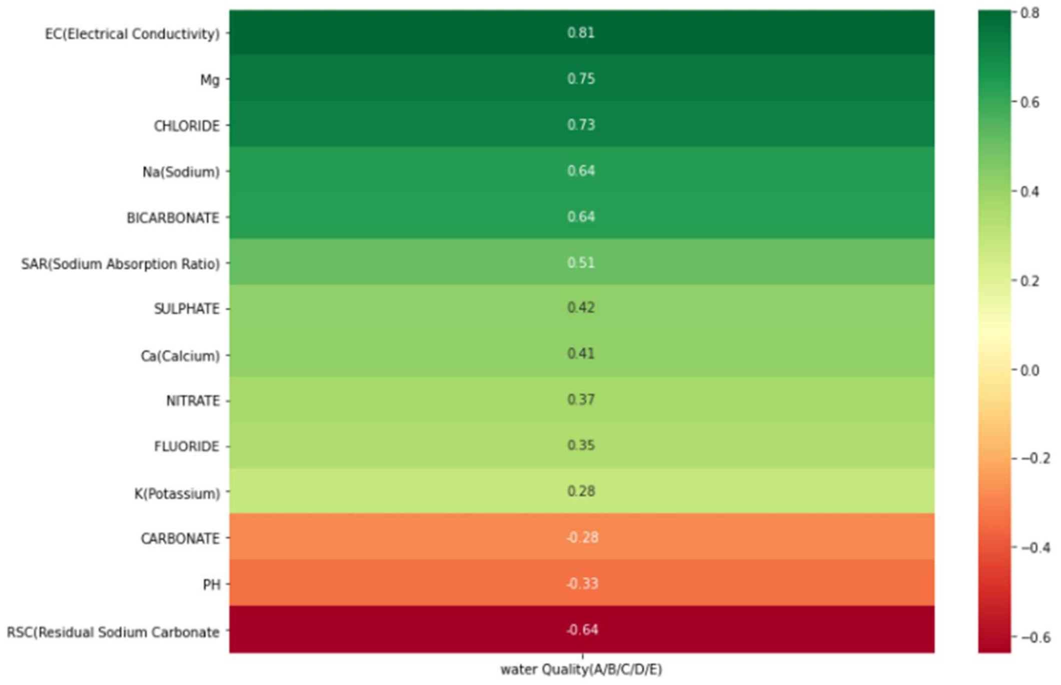
▾ Correlation co-efficients analysis

```
[29] # plot heat map of correlation
     import seaborn as sns
     plt.figure(figsize=(20,10))
     corr = data.corr()
     g = sns.heatmap(corr ,annot=True,cmap="RdYlGn")
```



	Mg	PH	K(Potassium)	NITRATE	SULPHATE	al Conductivity)	Ca(Calcium)	Na(Sodium)	CARBONATE	BICARBONATE	CHLORIDE	FLUORIDE	bsorption Ratio)	dium Carbonate	ality(A/B/C/D/E)
Mg	1	-0.37	0.25	0.55	0.46	0.75	0.29	0.56	-0.24	0.63	0.7	0.31	0.33	-0.77	0.75
PH	-0.37	1	0.026	-0.22	0.11	-0.23	-0.2	-0.052	0.64	-0.17	-0.28	-0.033	0.083	0.38	-0.33
K(Potassium)	0.25	0.026	1	-0.18	0.29	0.25	0.3	0.13	0.045	0.0085	0.23	0.26	0.071	-0.37	0.28
NITRATE	0.55	-0.22	-0.18	1	0.25	0.26	-0.19	0.22	-0.042	0.41	0.19	0.28	0.14	-0.19	0.37
SULPHATE	0.46	0.11	0.29	0.25	1	0.56	0.19	0.61	0.019	0.23	0.41	0.16	0.56	-0.45	0.42
EC(Electrical Conductivity)	0.75	-0.23	0.25	0.26	0.56	1	0.52	0.88	-0.21	0.6	0.95	0.3	0.72	-0.71	0.81
Ca(Calcium)	0.29	-0.2	0.3	-0.19	0.19	0.52	1	0.27	-0.25	0.19	0.56	-0.11	0.11	-0.75	0.41
Na(Sodium)	0.56	-0.052	0.13	0.22	0.61	0.88	0.27	1	-0.038	0.55	0.82	0.35	0.92	-0.43	0.64
CARBONATE	-0.24	0.64	0.045	-0.042	0.019	-0.21	-0.25	-0.038	1	-0.17	-0.22	0.056	0.015	0.34	-0.28
BICARBONATE	0.63	-0.17	0.0085	0.41	0.23	0.6	0.19	0.55	-0.17	1	0.5	0.34	0.45	-0.24	0.64
CHLORIDE	0.7	-0.28	0.23	0.19	0.41	0.95	0.56	0.82	-0.22	0.5	1	0.29	0.63	-0.71	0.73
FLUORIDE	0.31	-0.033	0.26	0.28	0.16	0.3	-0.11	0.35	0.056	0.34	0.29	1	0.35	-0.051	0.35
SAR(Sodium Absorption Ratio)	0.33	0.083	0.071	0.14	0.56	0.72	0.11	0.92	0.015	0.45	0.63	0.35	1	-0.18	0.51
RSC(Residual Sodium Carbonate	-0.77	0.38	-0.37	-0.19	-0.45	-0.71	-0.75	-0.43	0.34	-0.24	-0.71	-0.051	-0.18	1	-0.64
water Quality(A/B/C/D/E)	0.75	-0.33	0.28	0.37	0.42	0.81	0.41	0.64	-0.28	0.64	0.73	0.35	0.51	-0.64	1

```
[30] plt.figure(figsize=(12,9))
     corr = corr[[target_column]].sort_values(by=[target_column], ascending=False)[1:]
     g = sns.heatmap(corr ,annot=True,cmap="RdYlGn")
```


EC(Electrical Conductivity) 0.81
Mg 0.75
CHLORIDE 0.73
Na(Sodium) 0.64
BICARBONATE 0.64
SAR(Sodium Absorption Ratio) 0.51
SULPHATE 0.42
Ca(Calcium) 0.41
NITRATE 0.37
FLUORIDE 0.35
K(Potassium) 0.28
CARBONATE -0.28
PH -0.33
RSC(Residual Sodium Carbonate -0.64
water Quality(A/B/C/D/E)