ASSIGNMENT 3

- Name : Achal Rajesh Mate
- Roll No : 2203541
- Enroll No : MITU20BTCSD001
- Branch : CSE
- Class : TY CSE Is - 3
- Guided By : Prof Nagesh Jadhav Sir

Title:- **Download the any dataset from UCI or Data.org or from any other data repositories and Implement Single and multilayer perceptron on a dataset.**

Objectives:-

1. To learn about classification and regression
2. To learn MLP and backpropagation
3. To demonstrate and analyse the results

Theory:

Machine Learning

- Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
- The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
- Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without human intervention or assistance and adjust actions accordingly.

Types of Machine Learning

- Supervised Machine Learning
- Unsupervised Machine Learning
- Reinforcement Learning

1) Supervised Machine Learning

- Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output
- Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the out
- The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.
- The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is spam filtering.
- Supervised learning can be grouped further in two categories of algorithms:
  - Classification
  - Regression

2) Unsupervised Learning

- Unsupervised Learning is the training of a machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance.
- Unsupervised learning is a learning method in which a machine learns without any supervision.
- The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.
- In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classifieds into two categories of algorithms:
  - Clustering
  - Association

3) Reinforcement Learning

- Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.
- The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.
- Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones. In general, a reinforcement learning agent is able to perceive and interpret its environment, take actions and learn through trial and error.

Regression :

- Regression models are used to predict a continuous value.
- It falls under supervised learning wherein the algorithm is trained with both input features and output labels. It helps in establishing a relationship among the variables by estimating how one variable affects the other.
- Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of Market Trends, prediction of House prices, etc.
- The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

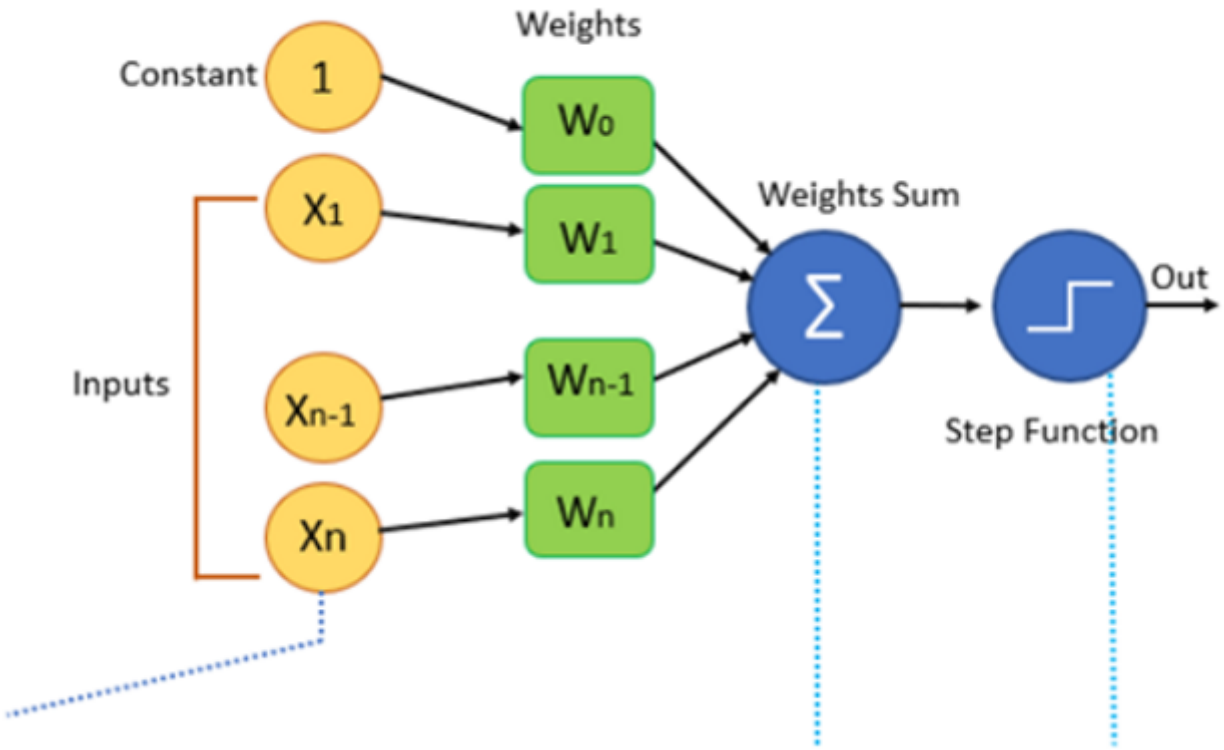
Classification

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.
- Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.
- In classification algorithm, a discrete output function(y) is mapped to input variable(x).
- y=f(x), where y = categorical output
- The best example of an ML classification algorithm is Email Spam Detector.
- The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

Perceptron

- The Perceptron is a linear machine learning algorithm for binary classification tasks. It may be considered one of the first and one of the simplest types of artificial neural networks.
- Perceptron Algorithm is used in a supervised machine learning domain for classification.

Perceptron Algorithm Block Diagram



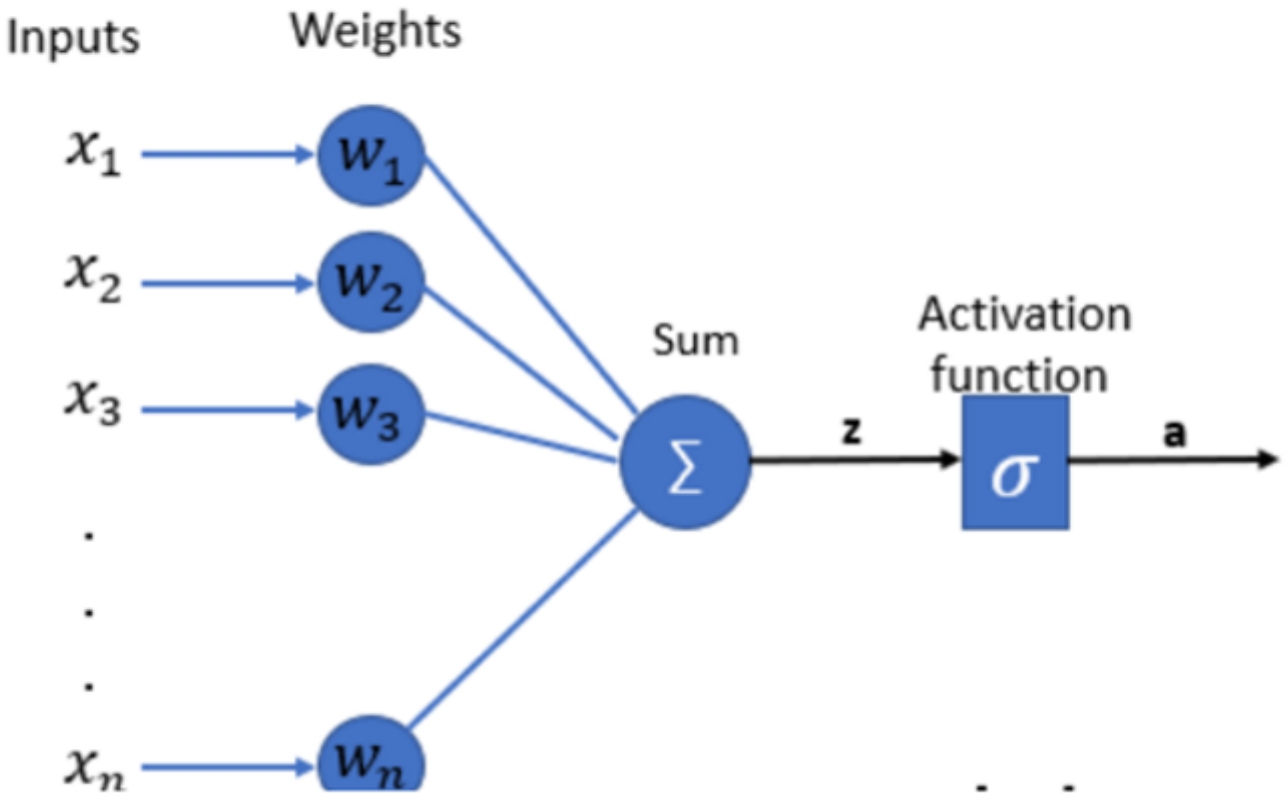
1. Input: All the features of the model we want to train the neural network will be passed as the input to it, Like the set of features [X1, X2, X3…..Xn]. Where n represents the total number of features and X represents the value of the feature.
2. Weights: Initially, we have to pass some random values as values to the weights and these values get automatically updated after each training error that is the values are generated during the training of the model. In some cases, weights can also be called as weight coefficients.
3. Weights Sum: Each input value will be first multiplied with the weight assigned to it and the sum of all the multiplied values is known as a weighted sum.

- Weights sum (Activation)= ∑Wi *Xi (from i=1 to i=n) + (W0* 1)
- The Perceptron algorithm is a two-class (binary) classification machine learning algorithm.
- It is a type of neural network model, perhaps the simplest type of neural network model.
- It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.
- If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.
    - Predict 1: If Activation > 0.0
    - Predict 0: If Activation <= 0.0

Single Layer Perceptron

- This is one of the easiest Artificial neural networks (ANN) types.
- A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model.
- The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.
- In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

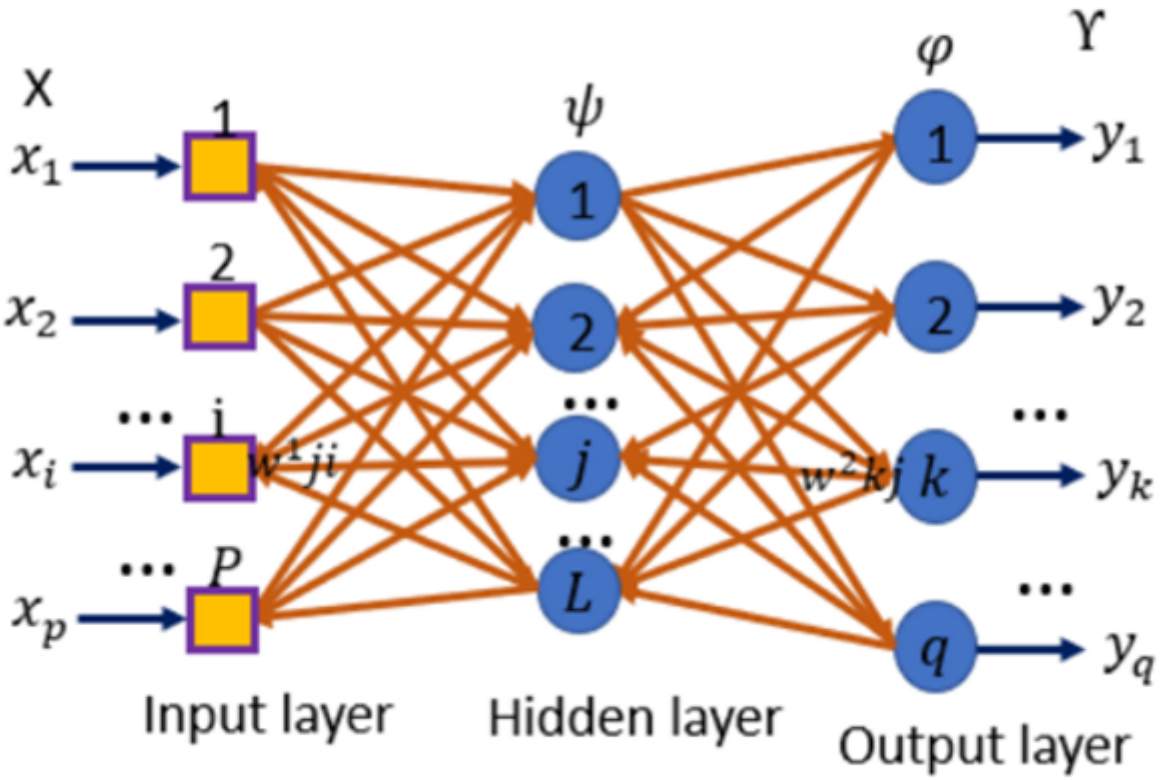
Single Layer Perceptron



Multi-Layer Perceptron

- Multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.
- The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:
  - Forward Stage: Activation functions start from the input layer in the forward stage and terminate on the output layer.

    - Backward Stage: In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.
- Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.
- A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.



BackPropagation Algorithm:

- The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

- In other words, backpropagation aims to minimize the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

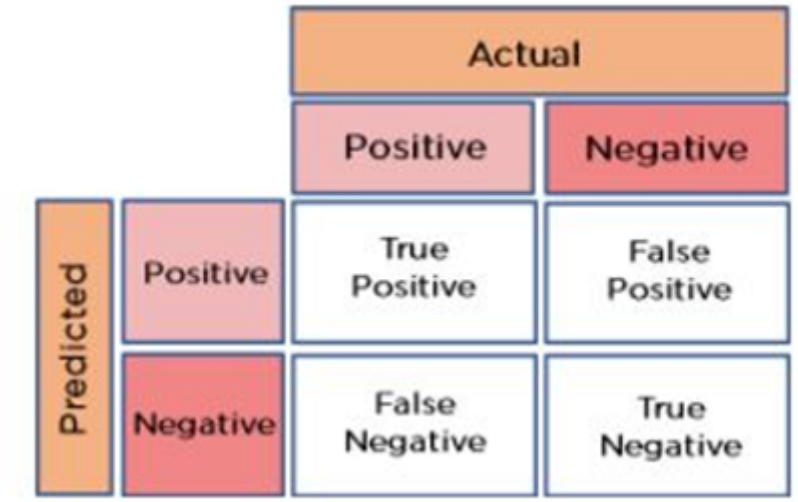
Activation function

- Activation functions also known non- linearity, describe the input-output relations in a non-linear way.
- This gives the model power to be more flexible in describing arbitrary relations. Here are some popular activation functions
- Sigmoid
- Relu
- TanH.

PERFORMANCE PARAMETERS

A. Confusion Matrix

- The confusion matrix provides the depth of prediction in a model as shown in fig.(fig no 5).
- The True Positive indicates that a positive value is predicted correctly[1,1].
- A False Positive occurs when the model forecasts the positive class erroneously as depicted.
- It indicate that the negative value is predicted as positive[0,1].
- False Negative indicate that a positive value is predicted as negative [1,0].
- Always False Negative should be less in our model performance.
- True Negative indicate that values that are actually negative and prediction also negative [0,0].



B. Accuracy

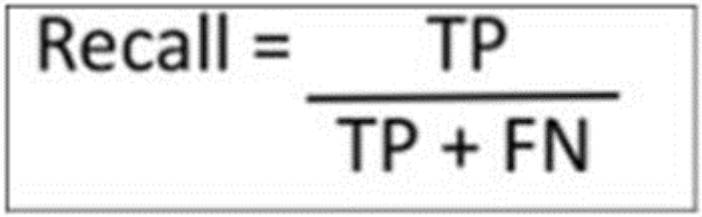
- The proportion of times the model correctly predicted the value. (AKA, the percentage of situation in which an industry's success or failure was correctly predicted).
- At first sight, this appears to be and excellent success statistic because the model was right the vast majority of time. However, it is dependent on the situation.
- The accuracy is defined by division of True Positive + True Negative by the sum of True Positive + True Negative + False Positive + False Negative

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

C. Recall:

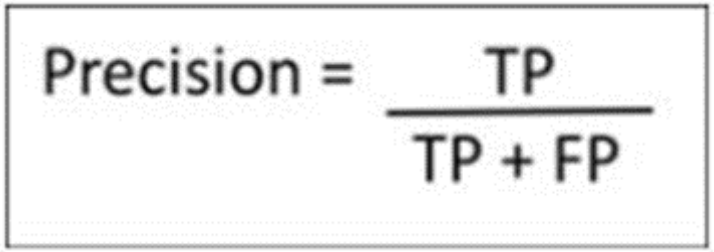
- Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

- The question recall answers is: Of all the passengers that truly survived,
- how many did we label? We have got recall of 0.631 which is good for this model as it's above 0.5.
- It is the division of the true positive values by true positive and false negative values



D. Precision:

- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- It indicates the percentage of spam messages that the classification system correctly identified as spam.
- It demonstrates that everything is correct.



E. F1-Score

- F1 Score is the weighted average of Precision and Recall.
- Therefore, this score takes both false positives and false negatives into account.
- Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution.
- Accuracy works best if false positives and false negatives have similar cost.
- If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

Dataset used and its attributes

- **Dataset Name** :-Glass Identification
- **Dataset Link** :- https://archive.ics.uci.edu/ml/datasets/Glass+Identification
- **Dataset Information**:-
  - Vina conducted a comparison test of her rule-based system and discriminant analysis.
  - BEAGLE is a product available through VRS Consulting, Inc.; 4676 Admiralty Way,Suite 206; Marina Del Ray, CA 90292 (213) 27-7890 and FAX: -3189.
  - In determining whether the glass was a type of "float" glass or not, the following results were obtained (# incorrect answers):

```
* Type of Sample                          Beagle   NN    DA
* Windows that were float processed (87)    10      12    21
* Windows that were not:            (76)    19      16    22
```

    - The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!
- **Number of Instances**: 214
- **Number of Attributes:** 10 (including an Id#) plus the class attribute -- all attributes are continuously valued
- **Attribute Information:**

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum

6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass: (class attribute)
    - 1 building_windows_float_processed
    - 2 building_windows_non_float_processed
    - 3 vehicle_windows_float_processed
    - 4 vehicle_windows_non_float_processed (none in this database)
    - 5 containers
    - 6 tableware
    - 7 headlamps

Import Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

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

In [2]:
```
from scipy.stats import norm
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import  mean_absolute_error,mean_squared_error,mean_squared_log_error
from sklearn.metrics import plot_confusion_matrix,accuracy_score
from sklearn import metrics
```

Read Dataset

In [3]:
```
df =pd.read_csv('D:\Downloads\DataSet\glass.csv')
```

View Top 5 Rows

In [4]:
```
df.head()
```

Out[4]:

	ID	Refractive_index	Sodium	Magnesium	Aluminum	Silicon	Potassium	Calcium	Barium	Iron	Target
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

View Last 5 rows

In [5]:
```
df.tail()
```

Out[5]:

	ID	Refractive_index	Sodium	Magnesium	Aluminum	Silicon	Potassium	Calcium	Barium	Iron	Target
209	210	1.51623	14.14	0.0	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.0	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.0	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.0	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.0	2.08	73.36	0.00	8.62	1.67	0.0	7

Dimensions of the Dataset

In [6]:
```

```

```
df.shape
```

Out[6]: (214, 11)

This dataset contains 214 rows and 11 columns

Columns in dataset

In [7]:
```
df.columns
```

Out[7]:
```
Index(['ID', 'Refractive_index', 'Sodium', 'Magnesium', 'Aluminum', 'Silicon',
       'Potassium', 'Calcium', 'Barium', 'Iron', 'Target'],
      dtype='object')
```

Concise Summary

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                214 non-null    int64  
 1   Refractive_index  214 non-null    float64
 2   Sodium            214 non-null    float64
 3   Magnesium         214 non-null    float64
 4   Aluminum          214 non-null    float64
 5   Silicon           214 non-null    float64
 6   Potassium         214 non-null    float64
 7   Calcium           214 non-null    float64
 8   Barium            214 non-null    float64
 9   Iron              214 non-null    float64
 10  Target            214 non-null    int64  
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

From above observation we can say that there is no null values in the dataset ,but lets varify by using pandas functions

Check Missing values

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
ID                0
Refractive_index  0
Sodium            0
Magnesium         0
Aluminum          0
Silicon           0
Potassium         0
Calcium           0
Barium            0
Iron              0
Target            0
dtype: int64
```

Hence there is no missing values in the dataset

Check Duplicate Value

In [10]:
```
df[df.duplicated()]
```

Out[10]:

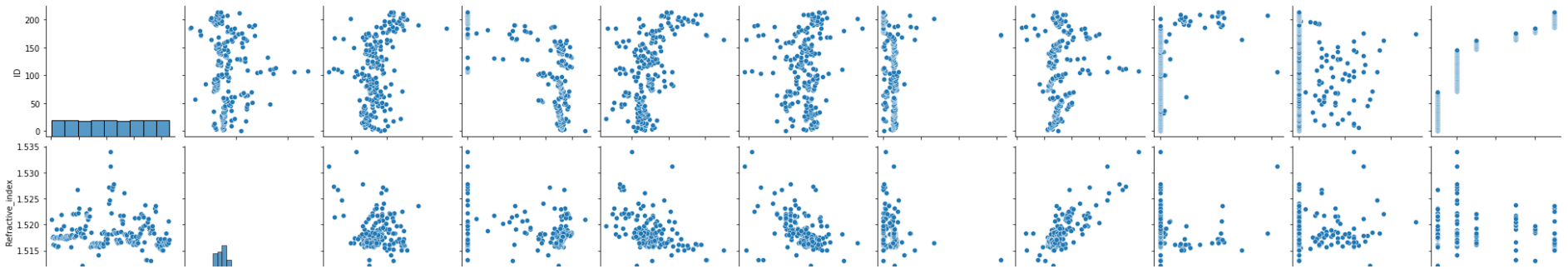
ID	Refractive_index	Sodium	Magnesium	Aluminum	Silicon	Potassium	Calcium	Barium	Iron	Target

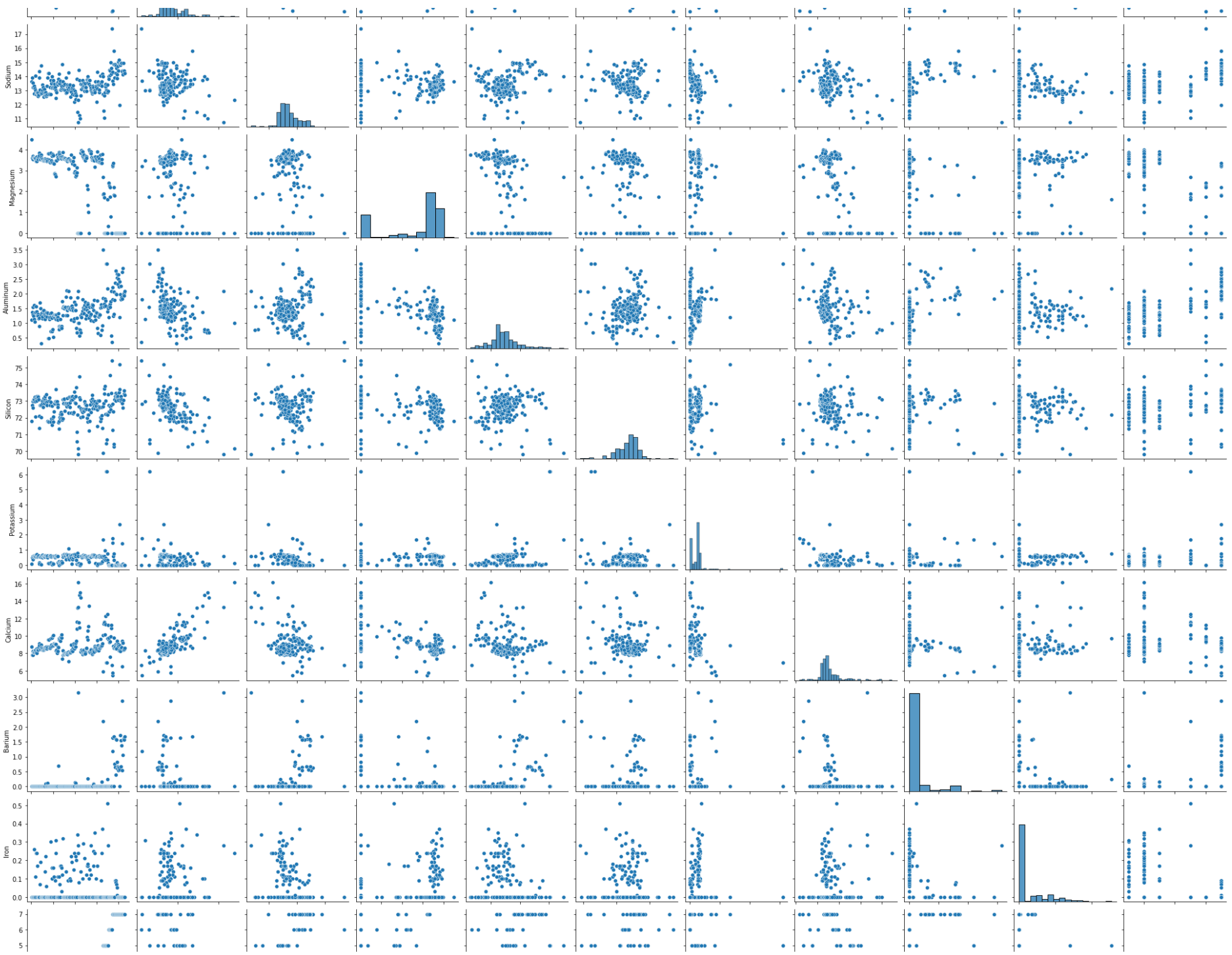
No duplicate values

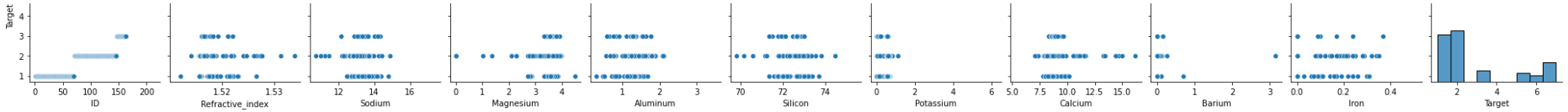
View Relationship between Data point in dataset

In [11]:
```
sns.pairplot(data= df)
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x24ae82250a0>
```




Sodium
Magnesium
Aluminum
Silicon
Potassium
Calcium
Barium
Iron



From above figure we can say that most of datapoints are not linearly co- related with each other

Statistical Summary of data

In [12]:
```
df.describe()
```

Out[12]:

	ID	Refractive_index	Sodium	Magnesium	Aluminum	Silicon	Potassium	Calcium	Barium	Iron	Target
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	107.500000	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009	2.780374
std	61.920648	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439	2.103739
min	1.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000	1.000000
25%	54.250000	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000	1.000000
50%	107.500000	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000	2.000000
75%	160.750000	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000	3.000000
max	214.000000	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000	7.000000

from above observation we can say that Potassium and Barium feature contains the outlier

Check the outliers from the dataset

In [13]:
```
for i in df.columns[1:-1]:
    Q1,Q3 = np.quantile(df[i],[0.25,0.75])
    IQR = Q3-Q1
    lower_bound = Q1 - (1.5*IQR)
    if lower_bound < 0:  # as in whole dataset there is no -ve value hence we set lowerbound to 0
        lower_bound=0
    upper_bound = Q3 + (1.5*IQR)
    print(i)
    print('Lower Bound =',np.round(lower_bound,2),' Upper Bound =',np.round(upper_bound,2))
```

```
    print('min value =',df[i].min(), ' max value =', df[i].max())

    if df[i].min() < lower_bound:
        print('negative Outliers',len(df[(df[i]<lower_bound)]))

    if df[i].max() > upper_bound:
        print('positive Outliers', len(df[(df[i]>upper_bound)]))




    print('='*50)
```

```
Refractive_index
Lower Bound = 1.51  Upper Bound = 1.52
min value = 1.51115  max value = 1.53393
negative Outliers 3
positive Outliers 14
==================================================
Sodium
Lower Bound = 11.53  Upper Bound = 15.2
min value = 10.73  max value = 17.38
negative Outliers 5
positive Outliers 2
==================================================
Magnesium
Lower Bound = 0  Upper Bound = 5.83
min value = 0.0  max value = 4.49
==================================================
Aluminum
Lower Bound = 0.53  Upper Bound = 2.29
min value = 0.29  max value = 3.5
negative Outliers 5
positive Outliers 13
==================================================
Silicon
Lower Bound = 71.07  Upper Bound = 74.3
min value = 69.81  max value = 75.41
negative Outliers 8
positive Outliers 4
==================================================
Potassium
Lower Bound = 0  Upper Bound = 1.34
min value = 0.0  max value = 6.21
positive Outliers 7
```

```
==================================================
Calcium
Lower Bound = 6.84  Upper Bound = 10.57
min value = 5.43  max value = 16.19
negative Outliers 5
positive Outliers 21
==================================================
Barium
Lower Bound = 0.0  Upper Bound = 0.0
min value = 0.0  max value = 3.15
positive Outliers 38
==================================================
Iron
Lower Bound = 0  Upper Bound = 0.25
min value = 0.0  max value = 0.51
positive Outliers 12
==================================================
```

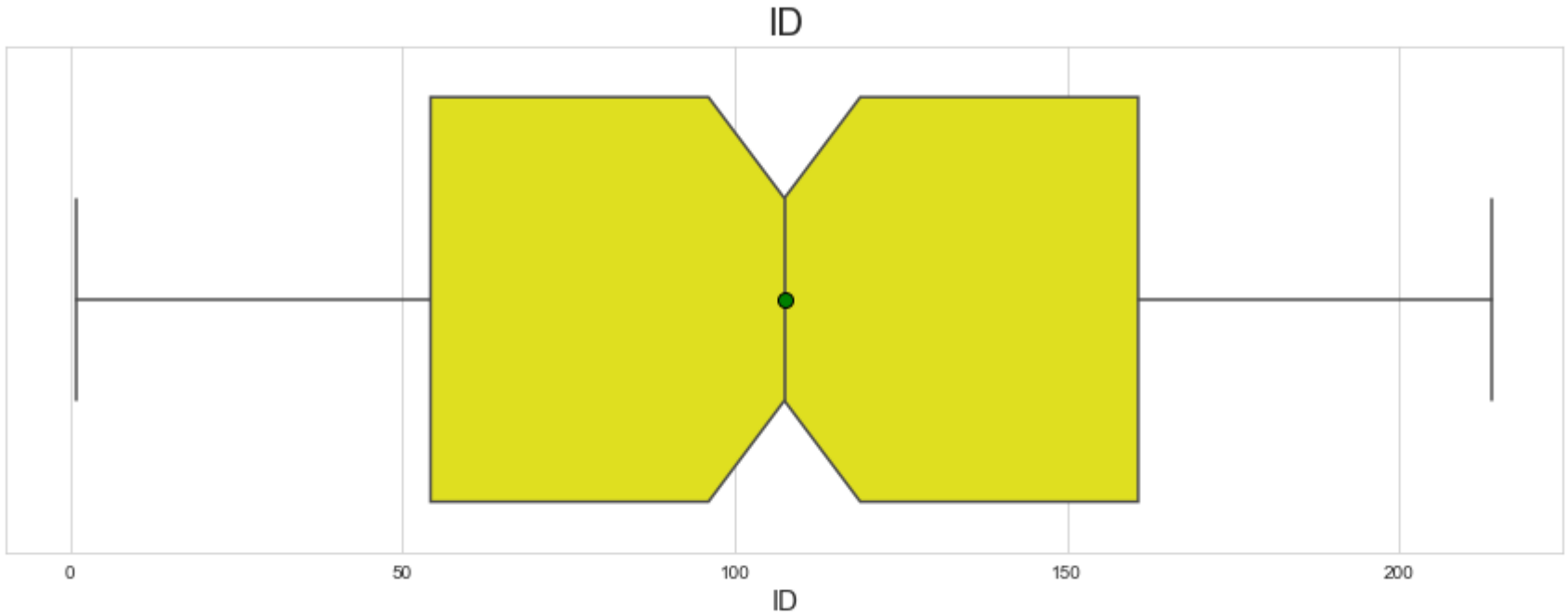
From above result we can say that Magnesium features does not contain any outlier and Barium and Calcium contains higher positive outlier repectively

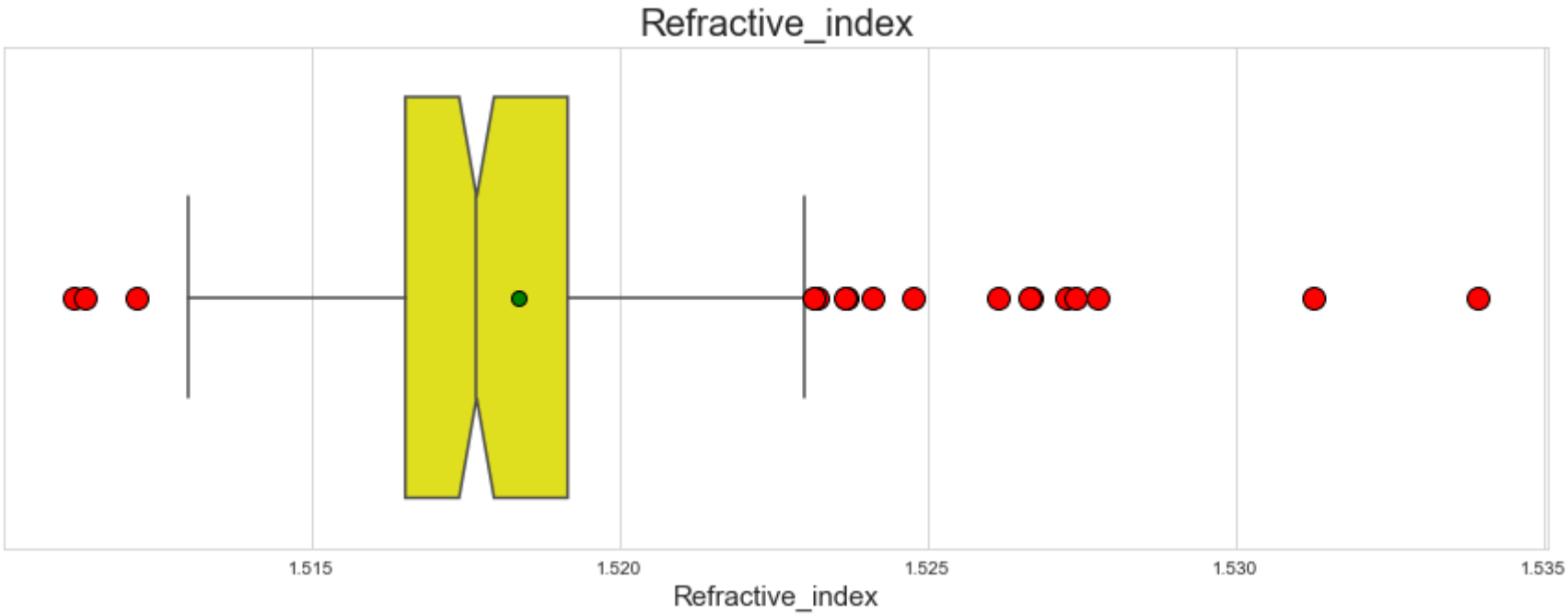
Plot the outlier using Boxplot

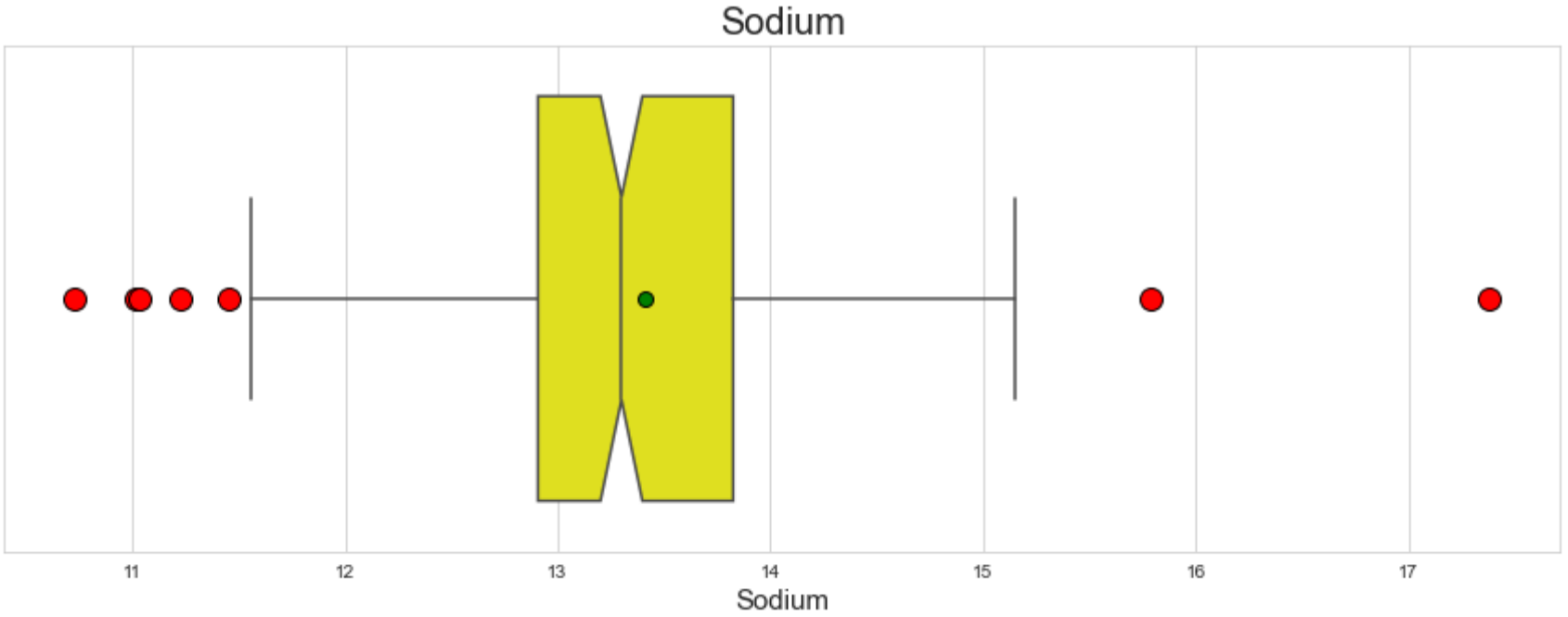
In [14]:
```
def box_plot_wiskers(feature):
    sns.set_style('whitegrid')
    out_marker = dict(markerfacecolor='red',markeredgecolor = 'black', marker='o', markersize = 12)
    plt.figure(figsize=(15,5))
    plt.title(feature ,fontsize = 20)
    plt.xlabel(feature , fontsize = 15)



    boxplot = sns.boxplot(df[feature],color='yellow',
                          showmeans=True,
                          notch = True,
                          flierprops=out_marker,
                          meanprops={ "marker":"o",
                                     "markerfacecolor":"green",
                                     "markeredgecolor":"black",
                                     "markersize":"8"})
```

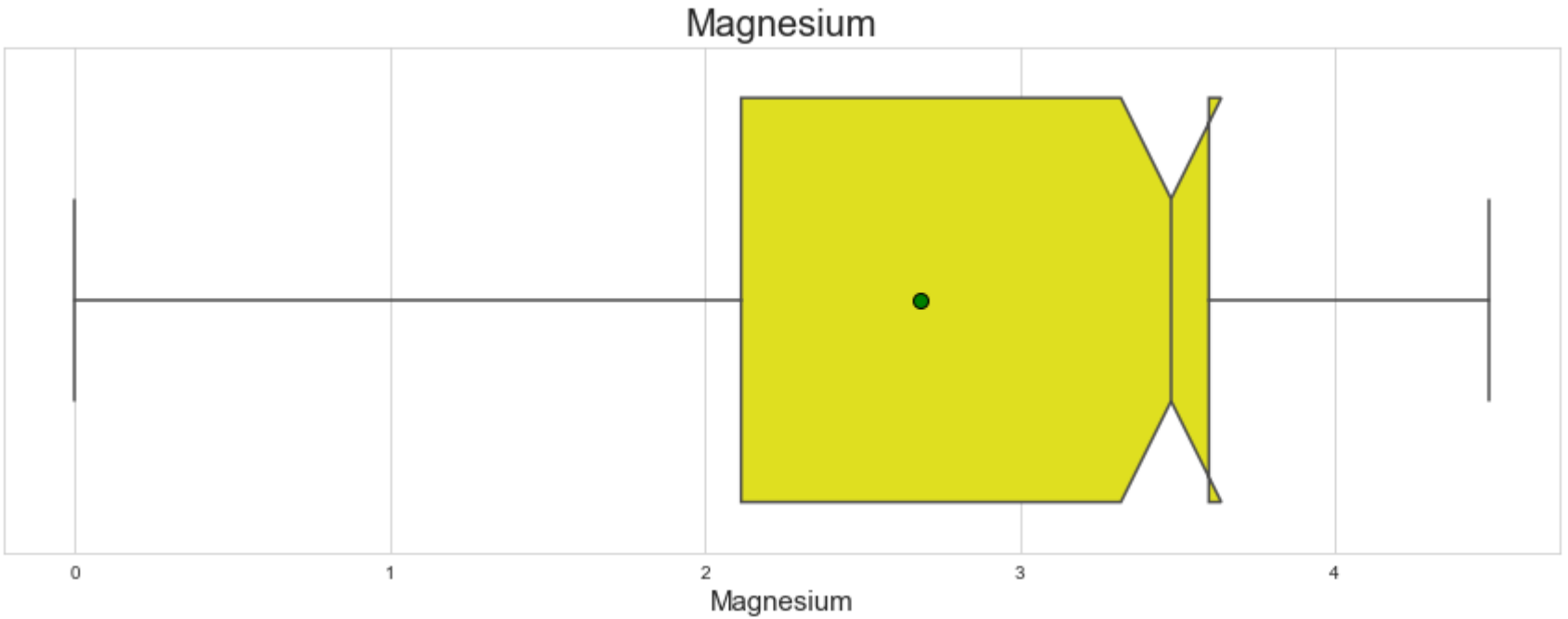
In [15]:
```
for clm in df.columns[:-1]:
```

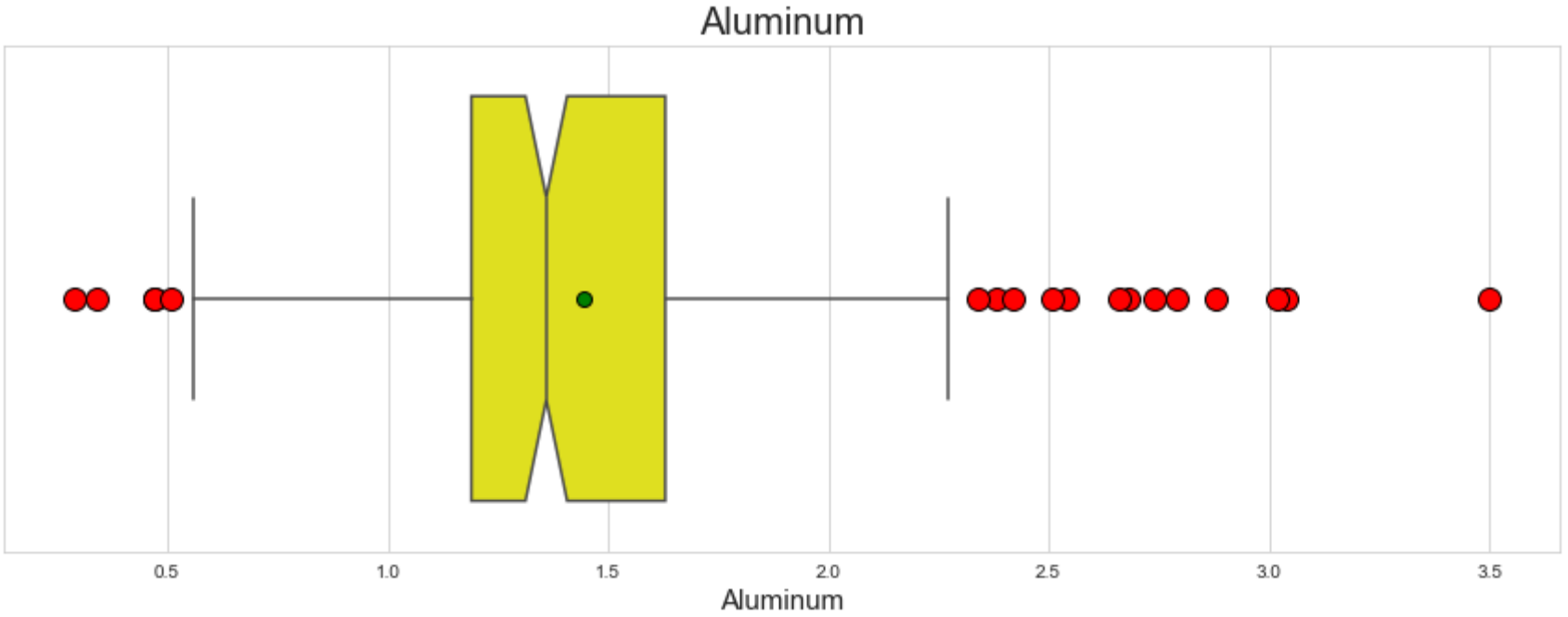
```
box_plot_wiskers(clm)
```

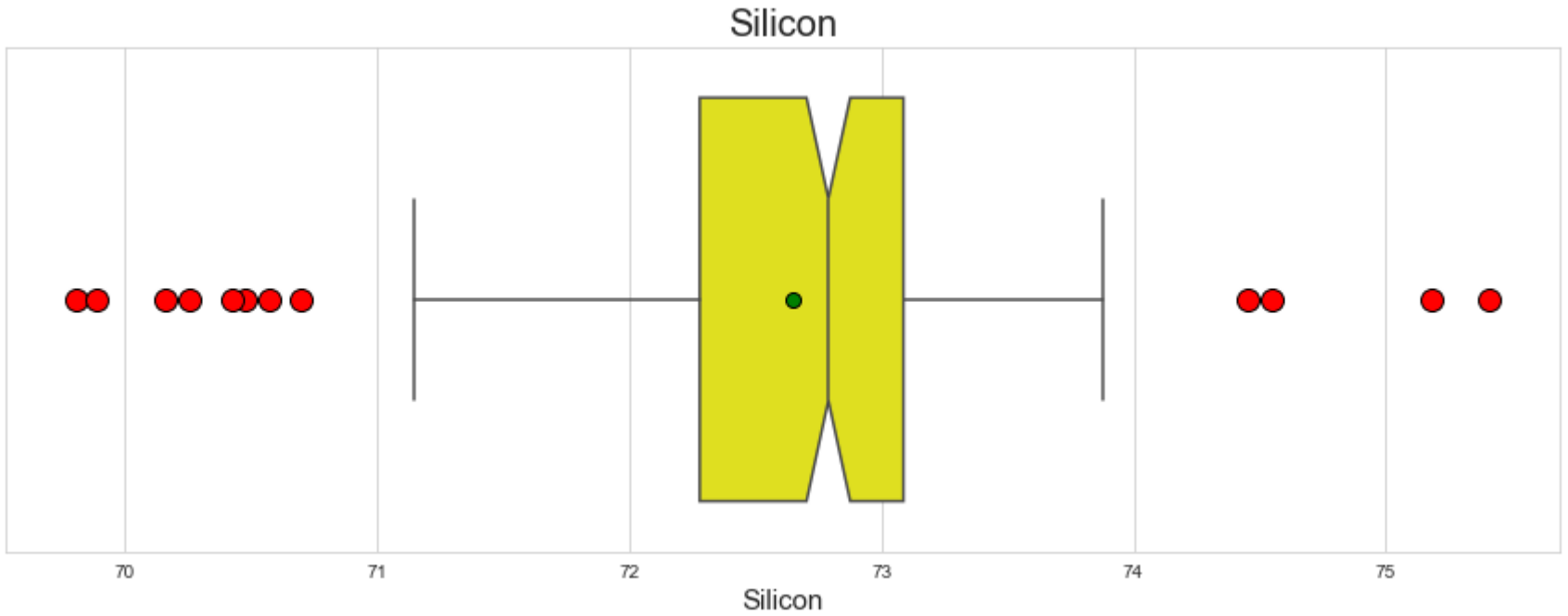


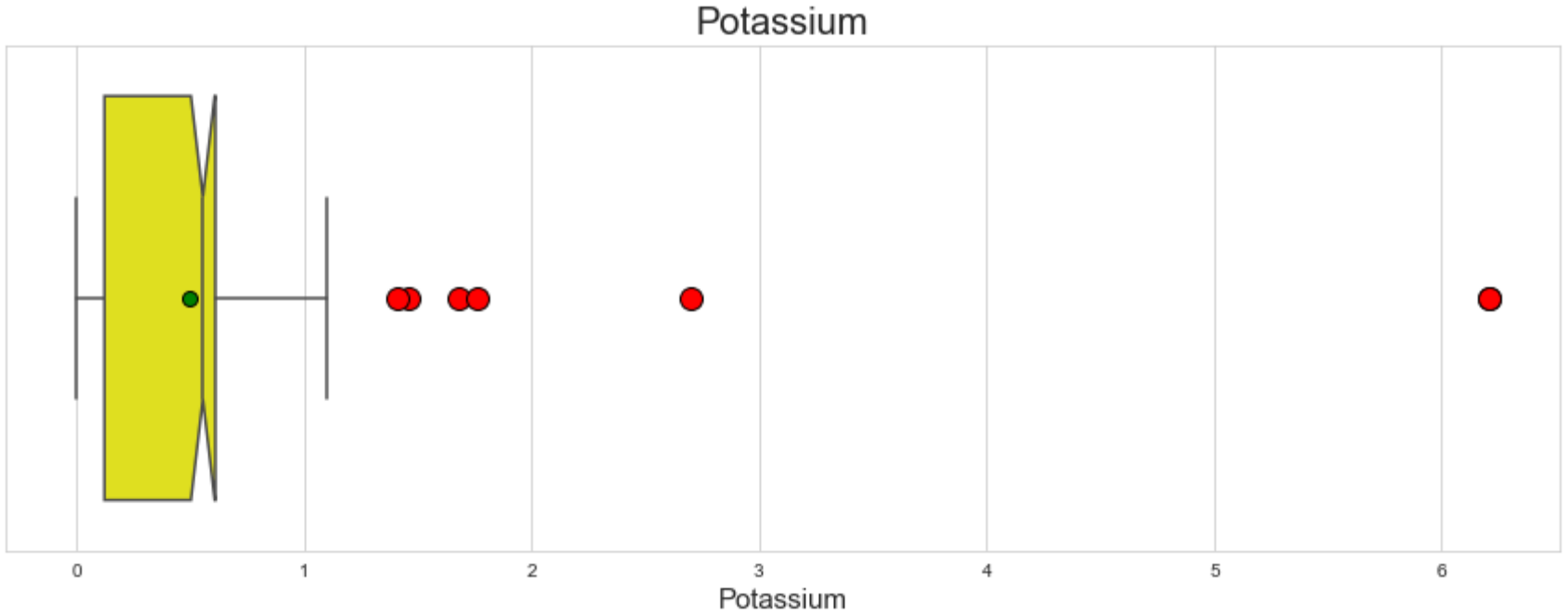

Refractive_index
1.515
1.520
1.525
1.530
1.535
Refractive_index

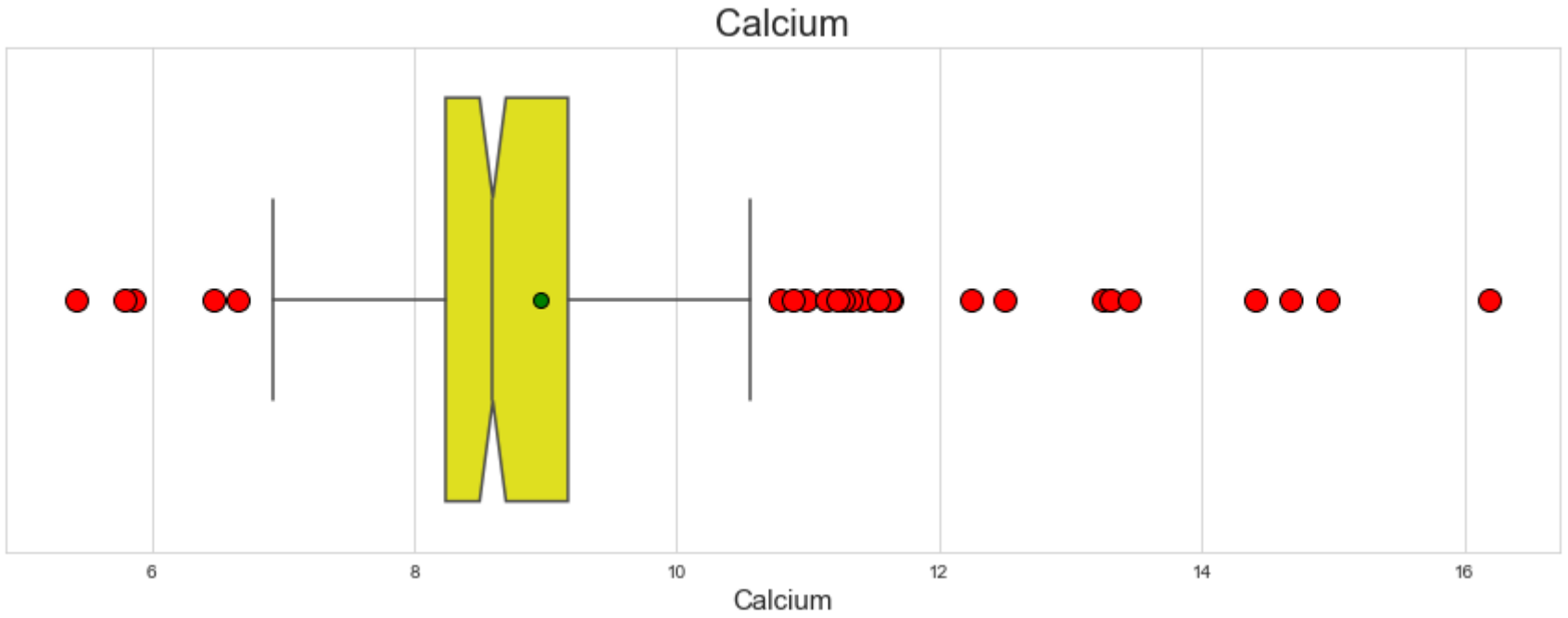

Sodium
11
12
13
14
15
16
17
Sodium

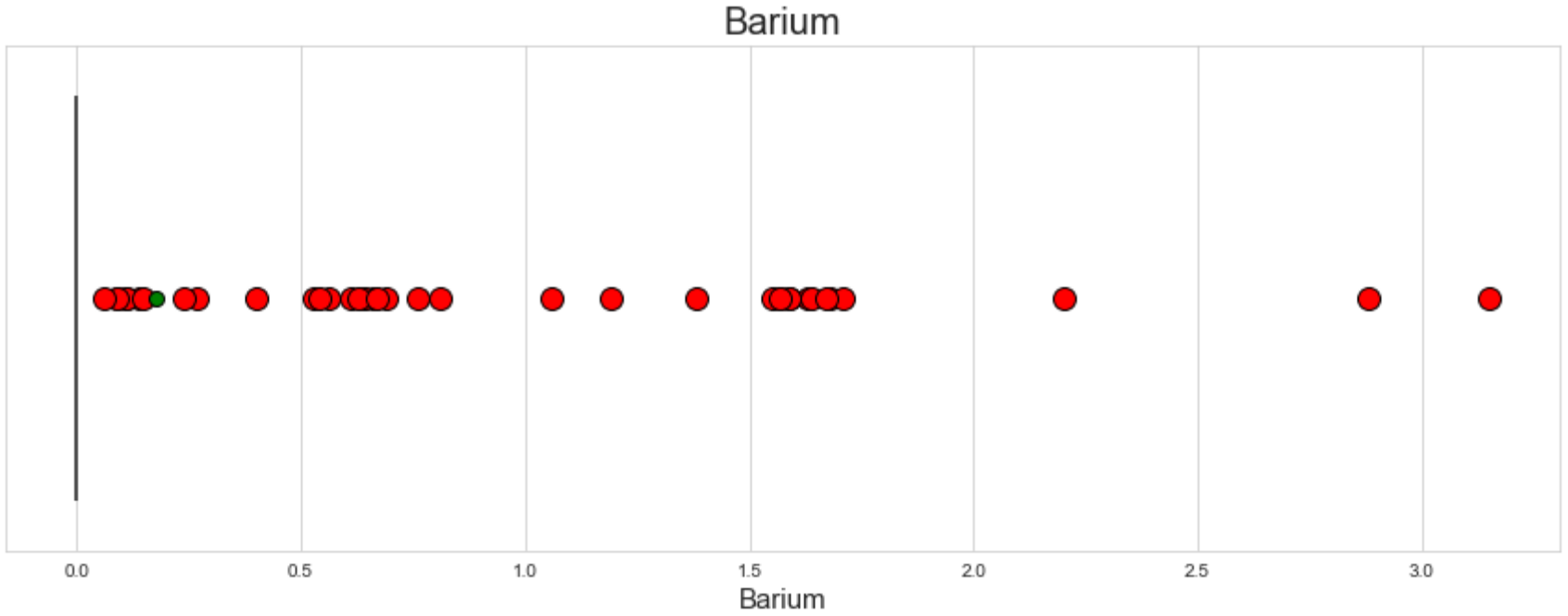

Magnesium
0
1
2
3
4
Magnesium

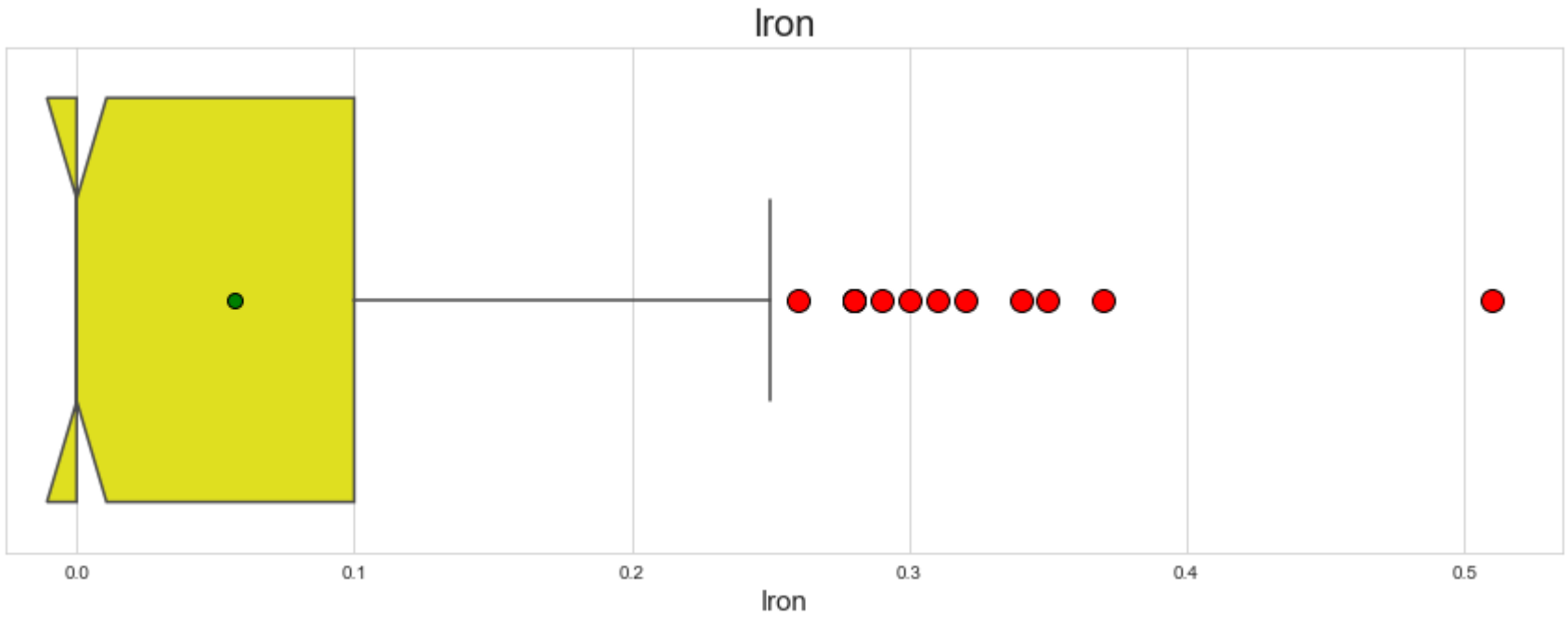

Aluminum
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Aluminum


Silicon
70
71
72
73
74
75
Silicon


Potassium
0
1
2
3
4
5
6
Potassium


Calcium
6
8
10
12
14
16
Calcium


Barium
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Barium

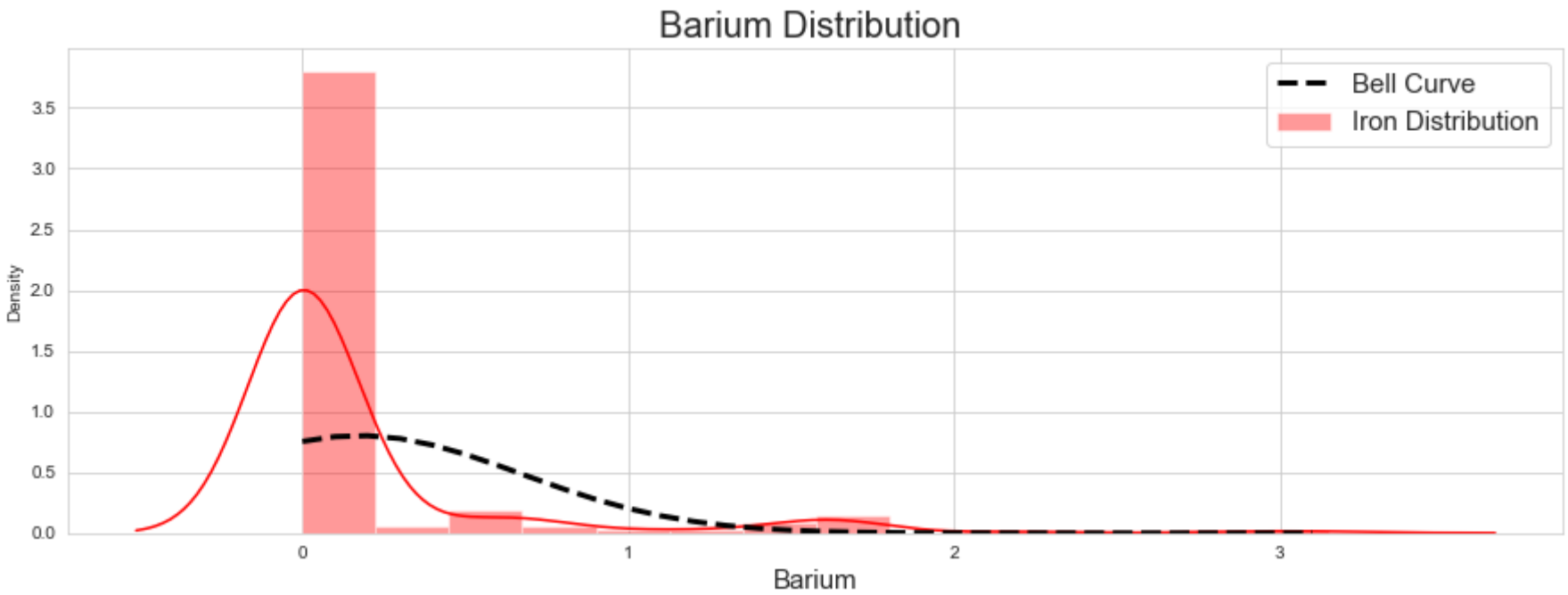


From above boxplot we can say that Barium has affect due to outlier, lets check the normal distribution of this feature

Normal Distribution of Barium Feature

In [16]:
```
plt.figure(figsize=(15,5))
plt.title('Barium Distribution ',fontsize = 20)
plt.xlabel('Barium' , fontsize = 15)
sns.distplot(df.Barium,color='r' , label=clm+' Distribution')
rang = np.arange(df.Barium.min(),df.Barium.max(),0.1)
plt.plot(rang, norm.pdf(rang,df.Barium.mean(),df.Barium.std()),color='black', ls='--',lw =3,label = 'Bell Curve')
plt.legend(fontsize= 15)
```

Out[16]:
```
<matplotlib.legend.Legend at 0x24ae8f9c2e0>
```



From above figure we can say that it is right-skewed Distribution(postive skeweness)

Check Types of Glasses

In [17]:
```
df.Target.unique()
```

Out[17]:
```
array([1, 2, 3, 5, 6, 7], dtype=int64)
```

In [18]:
```
df.Target = df.Target.map({1:1,2:2,3:3,5:4,6:5,7:6})
```

Value count of target variable

In [19]:
```
df.Target.value_counts()
```

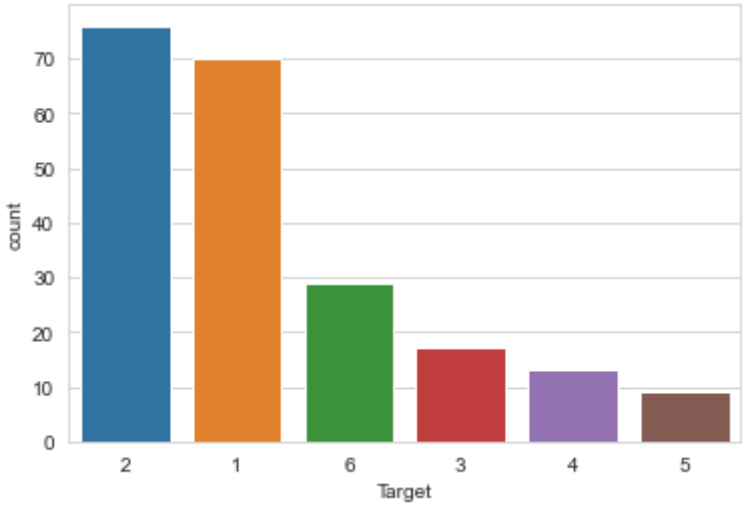
Out[19]:
```
2    76
1    70
6    29
```

```
3    17
4    13
5     9
Name: Target, dtype: int64
```

Plot the value count

In [20]:
```
sns.countplot('Target',data=df,label ="Target",order = df.Target.value_counts().index)
```

Out[20]: <AxesSubplot:xlabel='Target', ylabel='count'>



From above figure we can say that glass type 2(building_windows_non_float_processed) contains more values than other

Split the data into train and testing datasets

In [21]:
```
X =df.drop(['Target'], axis =1)
Y = df['Target']
```

In [22]:
```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2)
```

View the dimension of train and test dataset

In [23]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[23]:
```
((171, 10), (43, 10), (171,), (43,))
```

Build MLP Classifier Model

In [24]:
```
mlp = MLPClassifier(hidden_layer_sizes=(40,),learning_rate_init=0.01,max_iter=500,activation="logistic",alpha=0.01, tol=0.001)
```

In [25]:
```
mlp.fit(X_train,y_train)
mlp.score(X_train,y_train)
```

Out[25]:
```
1.0
```

In [26]:
```
mlp.score(X_test,y_test)
```

Out[26]:
```
1.0
```

In [27]:
```
predicted_classes = mlp.predict(X_test)

print(mlp.score(X_test,predicted_classes))
```

```
1.0
```

In [28]:
```
metrics.accuracy_score(predicted_classes,y_test)
```

Out[28]:
```
1.0
```

For Training, Testing and Prediction Accuracy = 100%

Print and Plot Confusion Matrix

In [29]:
```
target_names=['windows_float_processed','windows_non_float_processed','vehicle_windows','containers','tableware', 'headlamps']
```

In [30]:
```
confusion_matrix(y_test,predicted_classes)
```

Out[30]:
```
array([[16,  0,  0,  0,  0,  0],
       [ 0, 13,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  0,  0],
       [ 0,  0,  0,  3,  0,  0],
       [ 0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  8]], dtype=int64)
```

In [31]:
```
cm = confusion_matrix(y_test,predicted_classes)
row_sum = cm.sum(axis=0)
cm = np.append(cm,row_sum.reshape(1,-1),axis=0)
col_sum = cm.sum(axis=1)
cm = np.append(cm,col_sum.reshape(-1,1),axis=1)

plt.figure(figsize=(15,10))
sns.heatmap(cm,annot=True,cmap='summer',fmt='0.2f',xticklabels=target_names,
                yticklabels=target_names,linewidths=3,cbar=None,)

plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
windows_float_processed
windows_non_float_processed
vehicle_windows
containers
tableware
headlamps
Actual Values
16.00 0.00 0.00 0.00 0.00 0.00 16.00
0.00 13.00 0.00 0.00 0.00 0.00 13.00
0.00 0.00 2.00 0.00 0.00 0.00 2.00
0.00 0.00 0.00 3.00 0.00 0.00 3.00
0.00 0.00 0.00 0.00 1.00 0.00 1.00
0.00 0.00 0.00 0.00 0.00 8.00 8.00
16.00 13.00 2.00 3.00 1.00 8.00 43.00
windows_float_processed
windows_non_float_processed
vehicle_windows
containers
tableware
headlamps
Predicted Values

In [32]:
```
print('*'*30+'Classifcation Report'+'*'*30+'\n\n')
cr = classification_report(y_test,predicted_classes)
print(cr)
```

```
******************************Classifcation Report******************************


              precision    recall  f1-score   support

           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        13
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         8

    accuracy                           1.00        43
   macro avg       1.00      1.00      1.00        43
weighted avg       1.00      1.00      1.00        43
```

MLP Regressor

Dataset Name :- : Diamond Dataset

DataSet Link :- https://www.kaggle.com/datasets/shivam2503/diamonds

DataSet Info :-

- This classic dataset contains physical attributes and prices of 53940 diamonds
- There are 10 variables measuring various pieces of information about the diamonds.
- There are 3 variables with an ordered factor structure: cut, color, & clarity.
- An ordered factor arranges the categorical values in a low-to-high rank order. For example, there are 5 categories of diamond cuts with “Fair” being the lowest grade of cut to ideal being the highest grade.
- There are 6 variables that are of numeric structure: carat, depth, table, x, y, z
- There is 1 variable that has an integer structure: price

Dataset Attribute Description :-

Variable	Description	Values
price	price in US dollars	$326-$18,823
carat	weight of the diamond	0.2-5.01
cut	quality of the cut	Fair, Good, Very Good, Premium, Ideal
color	diamond color	J (worst) to D (best)
clarity	measurement of how clear the diamond is	I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)
x	length in mm	0-10.74
y	width in mm	0-58.9
z	depth in mm	0-31.8
depth	total depth percentage	43-79
table	width of top of diamond relative to widest point	43-95

Read Dataset

In [33]:
```
dfr = pd.read_csv('D:\Downloads\DataSet\diamonds.csv')
```

View Top 5 Rows of Dataset

In [34]:
```
dfr.head()
```

Out[34]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Columns in Dataset

In [35]:
```
dfr.columns
```

Out[35]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

Dimension of Dataset

In [36]:
```
dfr.shape
```

Out[36]:
```
(53940, 10)
```

The dataset contains 53940 instace or rows with 10 different features including target feature

Concise Summary

In [37]:
```
dfr.info()
```

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

```
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

from above observation it is show that cut,color and clarity features contains categorical data points and no misssing value in dataset but lets recheck it by using pandas method

In [38]:
```
dfr.isnull().sum()
```

Out[38]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

Hence it is clear that dataset does not contain any null points

Check if dataset contains duplicate rows

In [39]:
```
dfr[dfr.duplicated()]
```

Out[39]:

	carat	cut	color	clarity	depth	table	price	x	y	z
1005	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66

	carat	cut	color	clarity	depth	table	price	x	y	z
1006	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1007	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1008	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
2025	1.52	Good	E	I1	57.3	58.0	3105	7.53	7.42	4.28
...	...	...	...	...	...	...	...	...	...	...
47969	0.52	Ideal	D	VS2	61.8	55.0	1919	5.19	5.16	3.20
49326	0.51	Ideal	F	VVS2	61.2	56.0	2093	5.17	5.19	3.17
49557	0.71	Good	F	SI2	64.1	60.0	2130	0.00	0.00	0.00
50079	0.51	Ideal	F	VVS2	61.2	56.0	2203	5.19	5.17	3.17
52861	0.50	Fair	E	VS2	79.0	73.0	2579	5.21	5.18	4.09

146 rows × 10 columns

From above result it is stated that dataset contains 146 duplicate rows hence we need to drop this duplicate rows

In [40]:
```
dfr.drop(dfr[dfr.duplicated()].index,axis=0,inplace=True)
```

Check whether duplicate rows are drop or not

In [41]:
```
dfr[dfr.duplicated()]
```

Out[41]:

carat	cut	color	clarity	depth	table	price	x	y	z

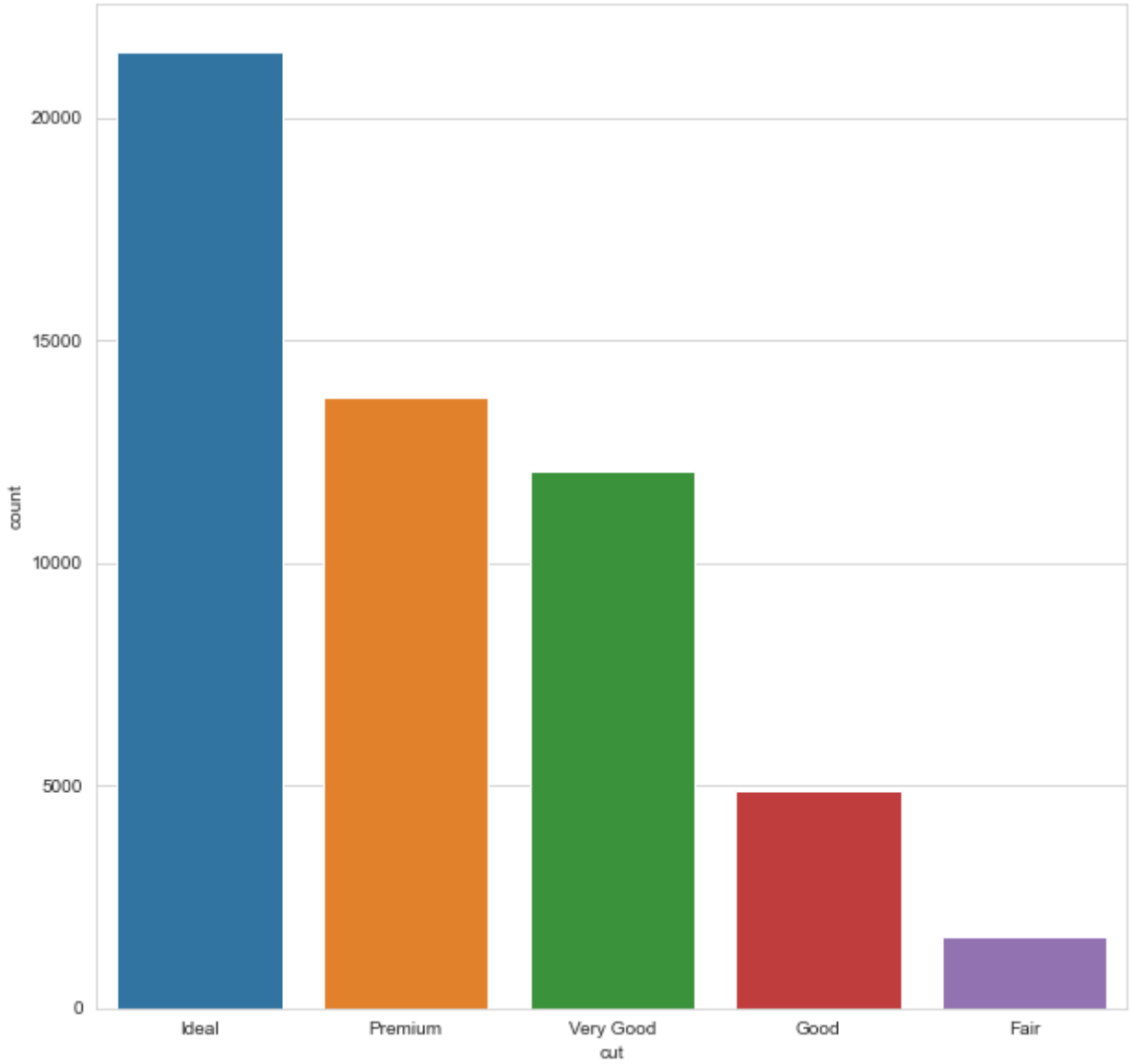
In [42]:
```
dfr.shape
```

Out[42]: (53794, 10)

Duplicated rows are drop.

In [43]:
```
plt.figure(figsize=(10,10))
sns.countplot(x= dfr.cut,order = dfr['cut'].value_counts().index)
```

Out[43]:
```
<AxesSubplot:xlabel='cut', ylabel='count'>
```

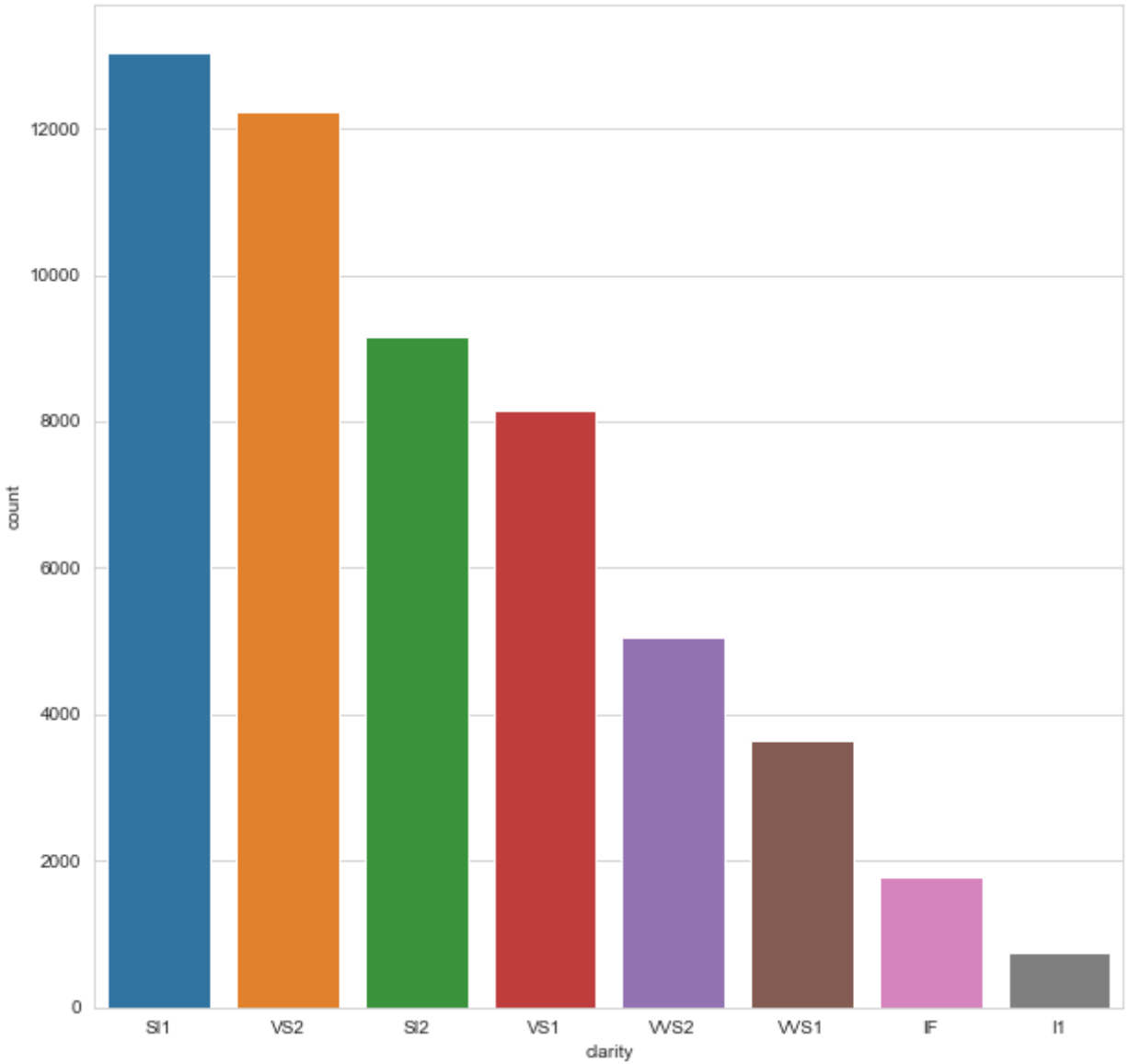


In the above observation ideal contains high number of datapoints the other categories of cut feature ,where as fair contains less number of datapoints

In [44]:

```
plt.figure(figsize=(10,10))
sns.countplot(x= dfr.clarity,order = dfr.clarity.value_counts().index)
```

Out[44]: <AxesSubplot:xlabel='clarity', ylabel='count'>

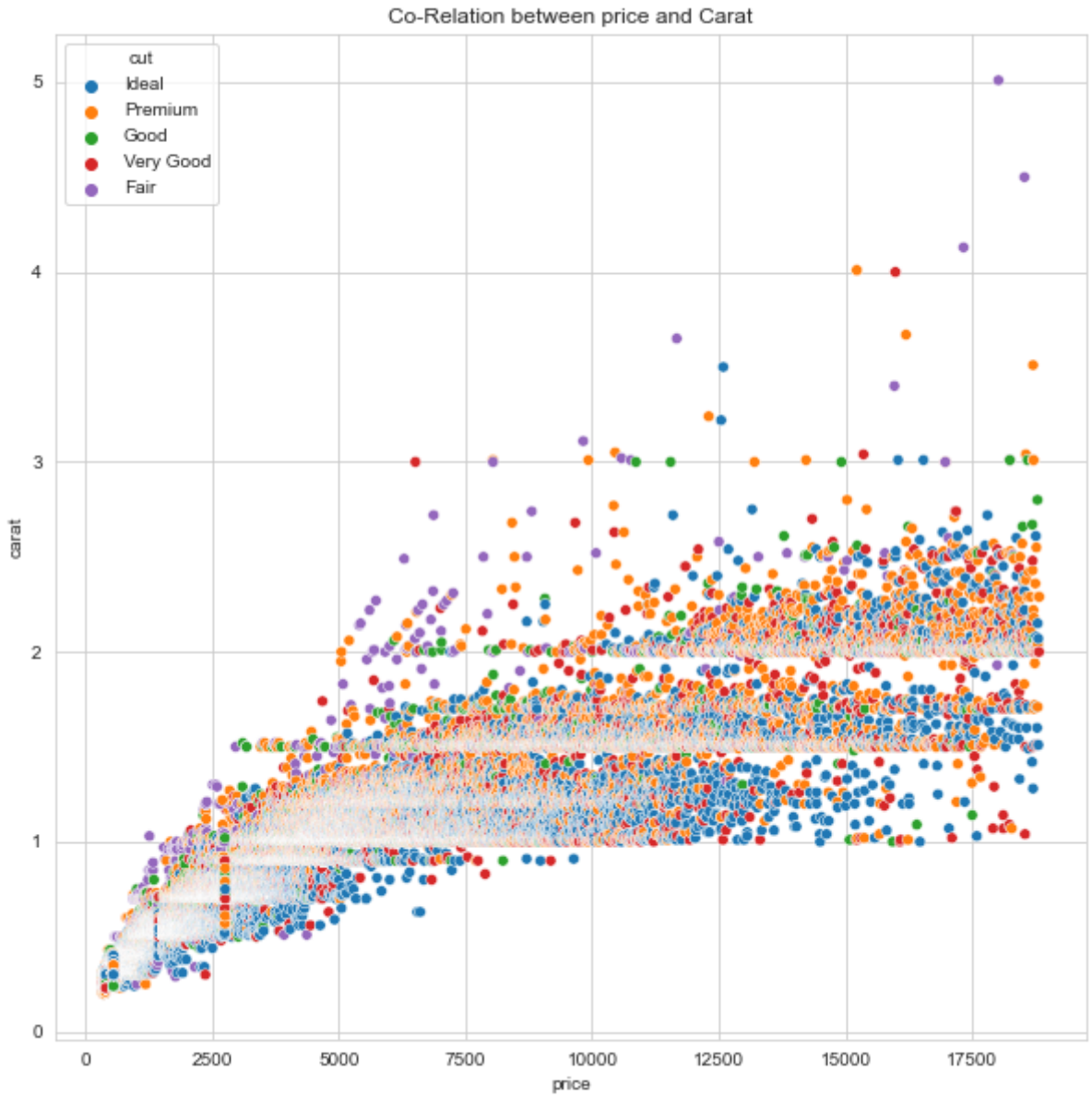


In the above obervation SI1 category of clarity feature contains more number of datapoints then other categories where I1 contains less number of data points

In [45]:
```
plt.figure(figsize=(10,10))
```

```
plt.title("Co-Relation between price and Carat")
sns.scatterplot(x = dfr.price,y = dfr.carat,hue=dfr.cut)
```

Out[45]: <AxesSubplot:title={'center':'Co-Relation between price and Carat'}, xlabel='price', ylabel='carat'>



from the above observation it is clear that price and carat features are linearly co-related with each other
ideal category of cut feature contains more datapoints than other
Fair category of cut feature contains highes price and weight

Converting Categorical features into numerical

In [46]:
```
le = LabelEncoder()
```

In [47]:
```
dfr['cut'] = le.fit_transform(dfr['cut'])
```

In [48]:
```
dfr['color'] = le.fit_transform(dfr['color'])
```

In [49]:
```
dfr['clarity'] = le.fit_transform(dfr['clarity'])
```

In [50]:
```
dfr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53794 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53794 non-null  float64
 1   cut      53794 non-null  int32  
 2   color    53794 non-null  int32  
 3   clarity  53794 non-null  int32  
 4   depth    53794 non-null  float64
 5   table    53794 non-null  float64
 6   price    53794 non-null  int64  
 7   x        53794 non-null  float64
 8   y        53794 non-null  float64
 9   z        53794 non-null  float64
dtypes: float64(6), int32(3), int64(1)
memory usage: 5.9 MB
```

All Categorical features are converted into numerical

Statistical Summary of data

In [51]:
```
dfr.describe()
```

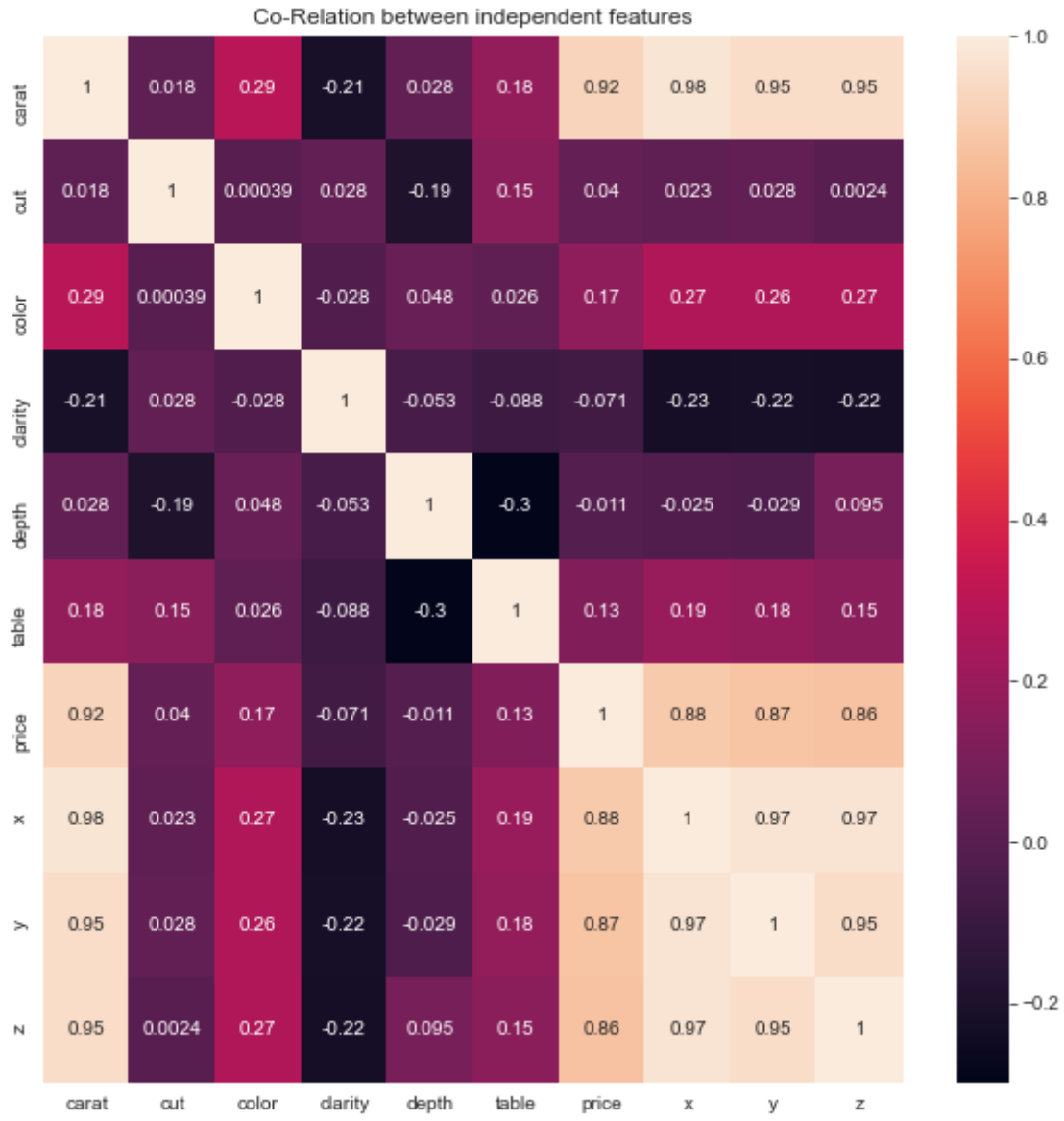
Out[51]:

	carat	cut	color	clarity	depth	table	price	x	y	
count	53794.00000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.0000
mean	0.79778	2.553947	2.593914	3.835762	61.748080	57.458109	3933.065082	5.731214	5.734653	3.5387
std	0.47339	1.027569	1.701117	1.724872	1.429909	2.233679	3988.114460	1.120695	1.141209	0.7050
min	0.20000	0.000000	0.000000	0.000000	43.000000	43.000000	326.000000	0.000000	0.000000	0.0000
25%	0.40000	2.000000	1.000000	2.000000	61.000000	56.000000	951.000000	4.710000	4.720000	2.9100
50%	0.70000	2.000000	3.000000	4.000000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.5300
75%	1.04000	3.000000	4.000000	5.000000	62.500000	59.000000	5326.750000	6.540000	6.540000	4.0300
max	5.01000	4.000000	6.000000	7.000000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.8000

Plot correlation between datapoints

In [52]:
```
plt.figure(figsize=(10,10))
plt.title("Co-Relation between independent features")
sns.heatmap(data = dfr.corr(),annot=True)
```

Out[52]:
```
<AxesSubplot:title={'center':'Co-Relation between independent features'}>
```



From the above observation it is clear that carat, price,x,y,z features are highly correlated with each other

Splitting training and testing datapoints

In [53]:

```
X = dfr.drop(['price'],axis= 1)
```

```
y = dfr['price']
```

In [54]:
```
scalar = StandardScaler()
```

In [55]:
```
X1 = scalar.fit_transform(X)
```

In [56]:
```
X = pd.DataFrame(X1,columns=X.columns)
```

In [57]:
```
X.head()
```

Out[57]:

	carat	cut	color	clarity	depth	table	x	y	z
0	-1.199402	-0.539089	-0.936990	-0.484540	-0.173495	-1.100486	-1.589399	-1.537553	-1.572574
1	-1.241651	0.434090	-0.936990	-1.064299	-1.362393	1.585691	-1.642938	-1.660231	-1.742780
2	-1.199402	-1.512269	-0.936990	0.095218	-3.390512	3.376475	-1.500168	-1.458689	-1.742780
3	-1.072656	0.434090	1.414429	0.674977	0.455922	0.242603	-1.366321	-1.318485	-1.288899
4	-1.030407	-1.512269	2.002284	-0.484540	1.085338	0.242603	-1.241397	-1.213332	-1.118694

In [58]:
```
X.shape
```

Out[58]:
```
(53794, 9)
```

In [59]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

In [60]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[60]:
```
((43035, 9), (10759, 9), (43035,), (10759,))
```

In [61]:
```
regressor  = MLPRegressor(solver='lbfgs',hidden_layer_sizes=(100,10),verbose=True)
```

In [62]:
```
regressor.fit(X_train,y_train)
```

Out[62]:
```
MLPRegressor(hidden_layer_sizes=(100, 10), solver='lbfgs', verbose=True)
```

In [63]:
```
print("Training Accuracy is : ",regressor.score(X_train,y_train))
```

```
Training Accuracy is :  0.9742601060810401
```

In [64]:
```
print("Testing  Accuracy is : ",regressor.score(X_test,y_test))
```

```
Testing  Accuracy is :  0.9704004478504896
```

In [65]:
```
train_pred = regressor.predict(X_train)
print("R^2: ",metrics.r2_score(y_train, train_pred))
print("MSE: ",mean_squared_error(y_train, train_pred))
print('RMSE: ',np.sqrt(mean_squared_error(y_train, train_pred)))
print('MAE: ', mean_absolute_error(y_train,train_pred))
```

```
R^2:  0.9742601060810401
MSE:  414599.10904020316
RMSE:  643.8937094274203
MAE:  373.75365195206734
```

In [66]:
```
test_pred = regressor.predict(X_test)
print("R^2: ",metrics.r2_score(y_test, test_pred))
print("MSE: ",mean_squared_error(y_test, test_pred))
print('RMSE: ',np.sqrt(mean_squared_error(y_test, test_pred)))
print('MAE: ', mean_absolute_error(y_test,test_pred))
```

```
R^2:  0.9704004478504896
MSE:  446673.0971440941
RMSE:  668.3360660207513
MAE:  374.1980584648587
```

Conclusion

Thus we have successfully completed the implementation of Multilayer perceptron