*Report submitted in partial fulfilment of*

*Masters in Data Science & Spatial Analytics*

*For*

# *Machine Learning*

*For*

# *Glass Prediction*

***Submitted by***


***Sahil Bhosale (22070243010)***


*Under the guidance of*

**Prof. Sahil Shah**
**&**
**Dr. Vidya Patkar**


**SYMBIOSIS INSTITUTE OF GEO-INFORMATICS**

SYMBIOSIS INSTITUTE OF GEOINFORMATICS
CONSTITUENT OF SYMBIOSIS INTERNATIONAL
(DEEMED UNIVERSITY) , PUNE

***SYMBIOSIS INSTITUTE OF GEOINFORMATICS***
***CONSTITUENT OF SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY),PUNE***

# INDEX

# ***ABSTRACT***

This paper presents a machine learning approach for the prediction of glass properties using a large dataset. The goal of this project is to develop a model that can accurately predict the thermal and optical properties of glass based on its composition. The dataset used for this project contains information about the chemical composition and the corresponding thermal and optical properties of a variety of glass samples.

Several machine learning algorithms were compared and evaluated, including decision trees, random forests, KNN and Gradient Boosting. The best performing model was K-Nearest Neighbor which achieved high accuracy and low error rates. The model was trained on a portion of the dataset and tested on the remaining data to evaluate its performance.

The results of this project show that the developed model can accurately predict the thermal and optical properties of glass based on its composition. This model can be used by glass manufacturers to optimize their production processes and improve the quality of their products. Additionally, this model can be used by researchers in the field of materials science to gain a better understanding of the relationship between the chemical composition and properties of glass.

# *INTRODUCTION*

Glass is a ubiquitous material with a wide range of applications in various industries, including construction, electronics, and optics. The properties of glass, such as its thermal and optical behavior, are crucial in determining its suitability for different applications. The prediction of these properties is a challenging task, as they depend on a complex interplay between the chemical composition and microstructural characteristics of the glass.

Traditionally, the prediction of glass properties has been based on empirical models and trial-and-error experiments. However, these methods are time-consuming, expensive, and often lack accuracy. With the advent of machine learning, it is now possible to predict glass properties with greater accuracy and efficiency.

In this project, we aim to develop a machine learning model that can predict the thermal and optical properties of glass based on its chemical composition. The model will be trained on a large dataset containing information about the composition and properties of a variety of glass samples. The performance of the model will be evaluated using various metrics, and the best performing model will be selected for further analysis.

The results of this project will have significant implications for the glass industry, as they will allow manufacturers to optimize their production processes and improve the quality of their products. Additionally, the results will contribute to our understanding of the relationship between the chemical composition and properties of glass, which will inform future research in the field of materials science.

# *METHODOLOGY*

The methodology for this glass prediction machine learning project can be divided into several steps:

**Data Collection:** The first step is to collect a large dataset of glass samples and their corresponding chemical composition. The dataset should be diverse and representative of the glass samples that are typically used in different applications.

The main purpose is to understand the problem that the dataset aims to solve. This involves reading the dataset description, analyzing the variables, and considering the overall goal of the dataset. For the project the dataset has been taken from USA machine learning repository.
https://archive.ics.uci.edu/ml/index.php

**Preprocess the data:** Data preprocessing is an important step in the process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific task.

For the project steps which are performed are checking for null values, checking for duplicate values

```
df.isnull().sum()
```

```
ri      0
na      0
mg      0
al      0
si      0
k       0
ca      0
ba      0
fe      0
type    0
dtype: int64
```

There are no null values.

## *CHAPTER 1: Establishing a Connection with PostGre SQL.*

Importing the dataset is the most important feature of building a model. Dataset can be imported by using pandas by using the command **read_csv.** Loading the dataset is the most important part of building a model as you can perform various steps on the dataset and perform EDA on it.

Here the dataset is imported in the model using **psycopg2** library. The connection has been created with the table and the queries has been passed to copy the data from the csv file.

Once the data has been copied the csv file is read in the data and further EDA and model building has been carried out.

**CONNECTION TO PLSQL & Reading data from CSV file.**

```
connection=conn=ps.connect(

    database="employee",
    user="postgres",
    password="Sahil@2102",port=5432
)
```

```
cursor=connection.cursor()
cursor.execute("Drop table if exists glass")
cursor.execute("CREATE TABLE glass(RI float ,Na float,Mg float,Al float,Si float,K float,Ca float,Ba float,Fe float,Type int)")
```

```
with open('glass.csv', 'r') as f:
    reader = csv.reader(f)
    next(reader)
connection.commit()
```

**CALLING THE VALUES INSERTED IN THE TABLE.**

```
cursor.execute('select * from glass')
cursor.fetchall()
```

```
[(1.52101, 13.64, 4.49, 1.1, 71.78, 0.06, 8.75, 0.0, 0.0, 1),
 (1.51761, 13.89, 3.6, 1.36, 72.73, 0.48, 7.83, 0.0, 0.0, 1),
 (1.51618, 13.53, 3.55, 1.54, 72.99, 0.39, 7.78, 0.0, 0.0, 1),
 (1.51766, 13.21, 3.69, 1.29, 72.61, 0.57, 8.22, 0.0, 0.0, 1),
 (1.51742, 13.27, 3.62, 1.24, 73.08, 0.55, 8.07, 0.0, 0.0, 1),
 (1.51596, 12.79, 3.61, 1.62, 72.97, 0.64, 8.07, 0.0, 0.26, 1),
 (1.51743, 13.3, 3.6, 1.14, 73.09, 0.58, 8.17, 0.0, 0.0, 1),
 (1.51756, 13.15, 3.61, 1.05, 73.24, 0.57, 8.24, 0.0, 0.0, 1),
 (1.51918, 14.04, 3.58, 1.37, 72.08, 0.56, 8.3, 0.0, 0.0, 1),
 (1.51755, 13.0, 3.6, 1.36, 72.99, 0.57, 8.4, 0.0, 0.11, 1),
 (1.51571, 12.72, 3.46, 1.56, 73.2, 0.67, 8.09, 0.0, 0.24, 1),
 (1.51763, 12.8, 3.66, 1.27, 73.01, 0.6, 8.56, 0.0, 0.0, 1),
 (1.51589, 12.88, 3.43, 1.4, 73.28, 0.69, 8.05, 0.0, 0.24, 1),
 (1.51748, 12.86, 3.56, 1.27, 73.21, 0.54, 8.38, 0.0, 0.17, 1),
 (1.51763, 12.61, 3.59, 1.31, 73.29, 0.58, 8.5, 0.0, 0.0, 1),
 (1.51761, 12.81, 3.54, 1.23, 73.24, 0.58, 8.39, 0.0, 0.0, 1),
 (1.51784, 12.68, 3.67, 1.16, 73.11, 0.61, 8.7, 0.0, 0.0, 1),
 (1.52196, 14.36, 3.85, 0.89, 71.36, 0.15, 9.15, 0.0, 0.0, 1),
 (1.51911, 13.9, 3.73, 1.18, 72.12, 0.06, 8.89, 0.0, 0.0, 1),
```

```
df=pd.read_sql('select * from glass',connection)
```

```
df
```

| | ri | na | mg | al | si | k | ca | ba | fe | type |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.52101 | 13.64 | 4.49 | 1.10 | 71.78 | 0.06 | 8.75 | 0.00 | 0.0 | 1 |
| 1 | 1.51761 | 13.89 | 3.60 | 1.36 | 72.73 | 0.48 | 7.83 | 0.00 | 0.0 | 1 |
| 2 | 1.51618 | 13.53 | 3.55 | 1.54 | 72.99 | 0.39 | 7.78 | 0.00 | 0.0 | 1 |
| 3 | 1.51766 | 13.21 | 3.69 | 1.29 | 72.61 | 0.57 | 8.22 | 0.00 | 0.0 | 1 |
| 4 | 1.51742 | 13.27 | 3.62 | 1.24 | 73.08 | 0.55 | 8.07 | 0.00 | 0.0 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 209 | 1.51623 | 14.14 | 0.00 | 2.88 | 72.61 | 0.08 | 9.18 | 1.06 | 0.0 | 7 |
| 210 | 1.51685 | 14.92 | 0.00 | 1.99 | 73.06 | 0.00 | 8.40 | 1.59 | 0.0 | 7 |
| 211 | 1.52065 | 14.36 | 0.00 | 2.02 | 73.42 | 0.00 | 8.44 | 1.64 | 0.0 | 7 |
| 212 | 1.51651 | 14.38 | 0.00 | 1.94 | 73.61 | 0.00 | 8.48 | 1.57 | 0.0 | 7 |
| 213 | 1.51711 | 14.23 | 0.00 | 2.08 | 73.36 | 0.00 | 8.62 | 1.67 | 0.0 | 7 |

Instead of inserting the values manually we can enter the values from a csv file and carry out further operations.

It is one of the features of psycopg2 library.

**Psycopg2 library:**

Psycopg2 is a popular Python library used for connecting to and interacting with PostgreSQL databases. It provides a simple and efficient API for executing SQL statements, fetching data from tables, and handling database transactions.

# CHAPTER 2: CHECKING THE DATAFRAME

After importing the data frame various operations needed to be carried out before building the models. The operations may include checking the top 5 columns, bottom columns and many other.

For checking this operations pandas library is imported and it has functions like head (), tail () and various other functions to work on the dataframe.

- head (): The **head function** in Python displays the first five rows of the dataframe by default. It takes in a single parameter: the *number of rows*. We can use this parameter to display the number of rows of our choice.

```
df.head(10)
```

| | ri | na | mg | al | si | k | ca | ba | fe | type |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.52101 | 13.64 | 4.49 | 1.10 | 71.78 | 0.06 | 8.75 | 0.0 | 0.00 | 1 |
| 1 | 1.51761 | 13.89 | 3.60 | 1.36 | 72.73 | 0.48 | 7.83 | 0.0 | 0.00 | 1 |
| 2 | 1.51618 | 13.53 | 3.55 | 1.54 | 72.99 | 0.39 | 7.78 | 0.0 | 0.00 | 1 |
| 3 | 1.51766 | 13.21 | 3.69 | 1.29 | 72.61 | 0.57 | 8.22 | 0.0 | 0.00 | 1 |
| 4 | 1.51742 | 13.27 | 3.62 | 1.24 | 73.08 | 0.55 | 8.07 | 0.0 | 0.00 | 1 |
| 5 | 1.51596 | 12.79 | 3.61 | 1.62 | 72.97 | 0.64 | 8.07 | 0.0 | 0.26 | 1 |
| 6 | 1.51743 | 13.30 | 3.60 | 1.14 | 73.09 | 0.58 | 8.17 | 0.0 | 0.00 | 1 |
| 7 | 1.51756 | 13.15 | 3.61 | 1.05 | 73.24 | 0.57 | 8.24 | 0.0 | 0.00 | 1 |
| 8 | 1.51918 | 14.04 | 3.58 | 1.37 | 72.08 | 0.56 | 8.30 | 0.0 | 0.00 | 1 |
| 9 | 1.51755 | 13.00 | 3.60 | 1.36 | 72.99 | 0.57 | 8.40 | 0.0 | 0.11 | 1 |

- tail (): The **tail function** in Python displays the last five rows of the dataframe by default. It takes in a single parameter: the *number of rows*. We can use this parameter to display the number of rows of our choice.

```
df.tail(10)
```

| | ri | na | mg | al | si | k | ca | ba | fe | type |
|---|---|---|---|---|---|---|---|---|---|---|
| **204** | 1.51817 | 14.95 | 0.0 | 2.27 | 73.30 | 0.00 | 8.71 | 0.67 | 0.0 | 7 |
| **205** | 1.51732 | 14.95 | 0.0 | 1.80 | 72.99 | 0.00 | 8.61 | 1.55 | 0.0 | 7 |
| **206** | 1.51645 | 14.94 | 0.0 | 1.87 | 73.11 | 0.00 | 8.67 | 1.38 | 0.0 | 7 |
| **207** | 1.51831 | 14.39 | 0.0 | 1.82 | 72.86 | 1.41 | 6.47 | 2.88 | 0.0 | 7 |
| **208** | 1.51640 | 14.37 | 0.0 | 2.74 | 72.85 | 0.00 | 9.45 | 0.54 | 0.0 | 7 |
| **209** | 1.51623 | 14.14 | 0.0 | 2.88 | 72.61 | 0.08 | 9.18 | 1.06 | 0.0 | 7 |
| **210** | 1.51685 | 14.92 | 0.0 | 1.99 | 73.06 | 0.00 | 8.40 | 1.59 | 0.0 | 7 |
| **211** | 1.52065 | 14.36 | 0.0 | 2.02 | 73.42 | 0.00 | 8.44 | 1.64 | 0.0 | 7 |
| **212** | 1.51651 | 14.38 | 0.0 | 1.94 | 73.61 | 0.00 | 8.48 | 1.57 | 0.0 | 7 |
| **213** | 1.51711 | 14.23 | 0.0 | 2.08 | 73.36 | 0.00 | 8.62 | 1.67 | 0.0 | 7 |

- shape ():  The **shape property** returns a tuple containing the shape of dataframe. The shape is the number of rows and columns in the dataframe.

```
df.shape
```

```
(214, 10)
```

- describe (): The **describe ()** method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains this information for each column:

count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.

75% - The 75% percentile*.
max - the maximum value.

```
df.describe()
```

| | ri | na | mg | al | si | k | ca | ba | fe | type |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 | 214.000000 |
| mean | 1.518365 | 13.407850 | 2.684533 | 1.444907 | 72.650935 | 0.497056 | 8.956963 | 0.175047 | 0.057009 | 2.780374 |
| std | 0.003037 | 0.816604 | 1.442408 | 0.499270 | 0.774546 | 0.652192 | 1.423153 | 0.497219 | 0.097439 | 2.103739 |
| min | 1.511150 | 10.730000 | 0.000000 | 0.290000 | 69.810000 | 0.000000 | 5.430000 | 0.000000 | 0.000000 | 1.000000 |
| 25% | 1.516522 | 12.907500 | 2.115000 | 1.190000 | 72.280000 | 0.122500 | 8.240000 | 0.000000 | 0.000000 | 1.000000 |
| 50% | 1.517680 | 13.300000 | 3.480000 | 1.360000 | 72.790000 | 0.555000 | 8.600000 | 0.000000 | 0.000000 | 2.000000 |
| 75% | 1.519157 | 13.825000 | 3.600000 | 1.630000 | 73.087500 | 0.610000 | 9.172500 | 0.000000 | 0.100000 | 3.000000 |
| max | 1.533930 | 17.380000 | 4.490000 | 3.500000 | 75.410000 | 6.210000 | 16.190000 | 3.150000 | 0.510000 | 7.000000 |

Every feature except for the target(Type) is continuous.

## *CHAPTER 3: Perform exploratory data analysis (EDA)*

EDA stands for Exploratory Data Analysis. It is an approach to analyzing and understanding data that involves exploring, summarizing, and visualizing the data to identify patterns, relationships, and anomalies.

The goal of EDA is to gain insights into the data, to understand the structure and content of the data, and to identify any potential problems or issues with the data that may need to be addressed before further analysis can be carried out.

EDA involves using a range of statistical and graphical techniques to explore the data, such as histograms, box plots, scatter plots, and correlation matrices. These techniques can be used to identify outliers, missing values, and other issues with the data that may need to be addressed before further analysis can be carried out.

EDA is an important part of the data analysis process, as it can help to identify interesting patterns and relationships in the data that may not be immediately obvious. It can also help to guide the selection of appropriate statistical models and methods for further analysis.

## WHY IS EDA NECESSARY?

- **To understand the data:** EDA is an important step in the data analysis process because it helps analysts to understand the structure, content, and quality of the data. EDA allows analysts to identify patterns, relationships, and anomalies in the data that may not be immediately apparent from summary statistics or raw data.

- **To identify potential problems with the data**: EDA can help to identify issues with the data, such as missing values, outliers, and other data quality problems. By identifying these problems early on, analysts can take steps to address them and ensure that the data is suitable for further analysis.

- **To guide further analysis:** EDA can help to guide the selection of appropriate statistical models and methods for further analysis. By identifying patterns and relationships in the data, analysts can determine which variables are most important and which statistical techniques are most appropriate.

- **To communicate findings:** EDA is also useful for communicating findings to others. By creating visualizations and summaries of the data, analysts can effectively communicate their findings to stakeholders who may not have a background in statistics or data analysis.

EDA is necessary because it helps analysts to understand, explore, and communicate their data in a meaningful way. Without EDA, analysts may miss important patterns and relationships in the data, and may make inappropriate decisions based on faulty or incomplete data.

Visualization used in the projects are as follows:

- Countplot
- Boxplot
- Pie Chart
- Kdeplot

Let’s understand each plot with their importance.

# Countplot

A countplot is a type of plot used in EDA (Exploratory Data Analysis) to display the count of observations in a categorical variable. It is a bar plot where the height of each bar represents the count of observations in a particular category.

In a countplot, the x-axis represents the categorical variable and the y-axis represents the count of observations. Each bar on the plot represents a category of the variable, and the height of the bar represents the count of observations in that category. The bars can be colored or shaded differently to indicate different levels or categories within a variable.

Countplots are particularly useful when dealing with categorical data, as they allow analysts to quickly visualize the distribution of data across different categories. They can be used to identify which categories are most common or frequent, and can help to identify any unusual or unexpected patterns in the data.

Countplots can be created using various visualization libraries in Python, such as Seaborn, Matplotlib, and Plotly. Seaborn, in particular, provides a high-level interface to create visually appealing countplots with just a few lines of code, making it a popular choice for EDA.

## FEATURES

- **Handling missing values:** Countplots can handle missing values by simply not plotting them, as they are not considered to be in any category. Alternatively, missing values can be plotted as a separate category, depending on the desired interpretation.

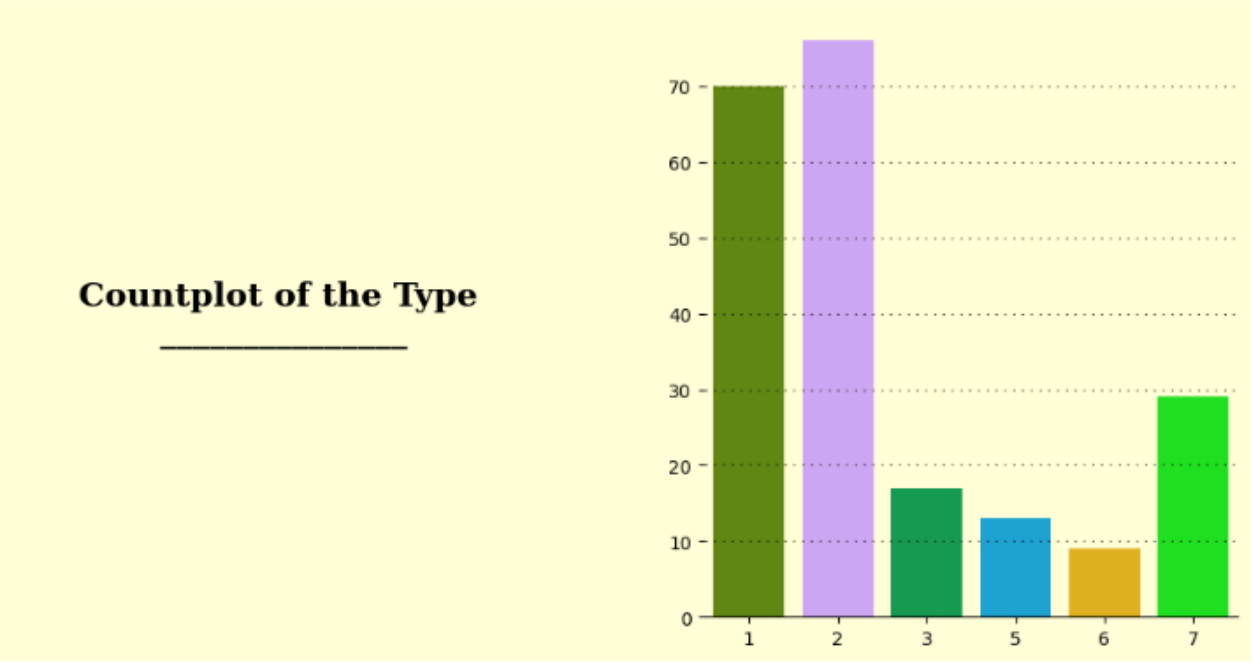
- **Comparison of multiple categorical variables:** Multiple categorical variables can be visualized on a single countplot by using different hues to

differentiate the variables. This allows for quick and easy comparisons between the variables.

- **Normalization:** Countplots can be normalized to show the relative frequency of each category, instead of the raw count. This is useful when dealing with datasets with significantly different sample sizes.

- **Adding labels and titles:** Countplots can be customized with various labels and titles to provide context and clarity. The x-axis and y-axis labels can be added to provide additional information about the categories and counts, and the plot title can be used to summarize the overall findings.

- **Styling options:** Countplots can be further customized with various styling options, such as changing the color palette, adding grid lines, or changing the font size. These options can help to make the plot more visually appealing and easier to interpret.



# BOXPLOT

A boxplot is a type of plot used in EDA (Exploratory Data Analysis) to display the distribution of numerical data through their quartiles. It is a standardized way of displaying the distribution of data based on five summary statistics: the minimum, the maximum, the first quartile (Q1), the median (Q2), and the third quartile (Q3).

In a boxplot, the box represents the interquartile range (IQR), which is the distance between the first and third quartiles (Q1 and Q3). The line inside the box represents the median. The whiskers extend from the box to the minimum and maximum values, and any data points outside of the whiskers are considered outliers.

Boxplots are particularly useful when dealing with numerical data, as they allow analysts to quickly visualize the spread of data and identify potential outliers. They can be used to compare the distribution of data across different groups or categories, and can help to identify any unusual or unexpected patterns in the data.

Boxplots can be created using various visualization libraries in Python, such as Seaborn, Matplotlib, and Plotly. Seaborn, in particular, provides a high-level interface to create visually appealing boxplots with just a few lines of code, making it a popular choice for EDA.

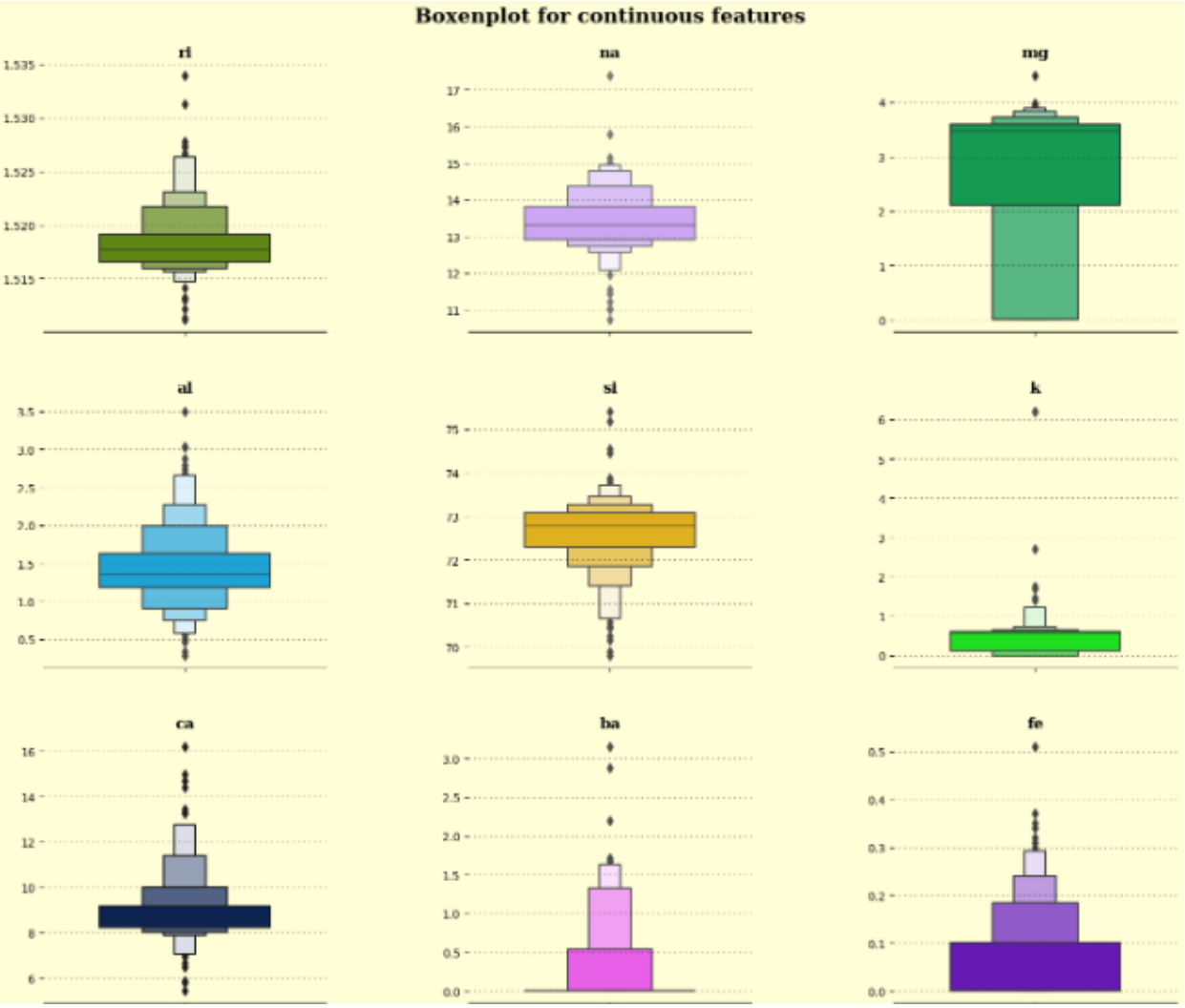
**Boxplots are used in EDA for several reasons, including:**

- **Identifying outliers:** Boxplots are particularly useful for identifying potential outliers in the data, which are data points that are significantly different from the rest of the data. Outliers can be important for identifying unusual patterns or errors in the data.

- **Comparing distributions:** Boxplots can be used to compare the distribution of data across different groups or categories. This can help to identify differences in the data and can provide insights into potential relationships or patterns.

- **Summarizing data:** Boxplots provide a concise summary of the distribution of data, including information about the median, quartiles, and potential outliers. This can be useful for providing a high-level overview of the data.

Overall, boxplots are a useful tool in EDA for analyzing numerical data. They allow for quick and easy visualization of the distribution of data, and can be customized with various options to provide additional context and insights.



# PIECHART

A pie chart is a circular graph used in data visualization to represent the relative sizes of different categories or data points as parts of a whole. The entire circle represents the total data, and each slice of the pie represents a proportion of the total data. The size of each slice is proportional to the quantity it represents, and the slices are often colored or labeled to distinguish between the different categories.

Pie charts are particularly useful for displaying data that can be categorized or broken down into different parts, such as market share, survey results, or budget allocations. They are also useful for visualizing data that can be expressed as percentages, as the size of each slice can be directly interpreted as a percentage of the total.

Pie charts can be created using various visualization libraries in Python, such as Matplotlib and Plotly. They can be customized with various options, such as adding labels, changing the colors or styles, and adjusting the size or shape of the chart.

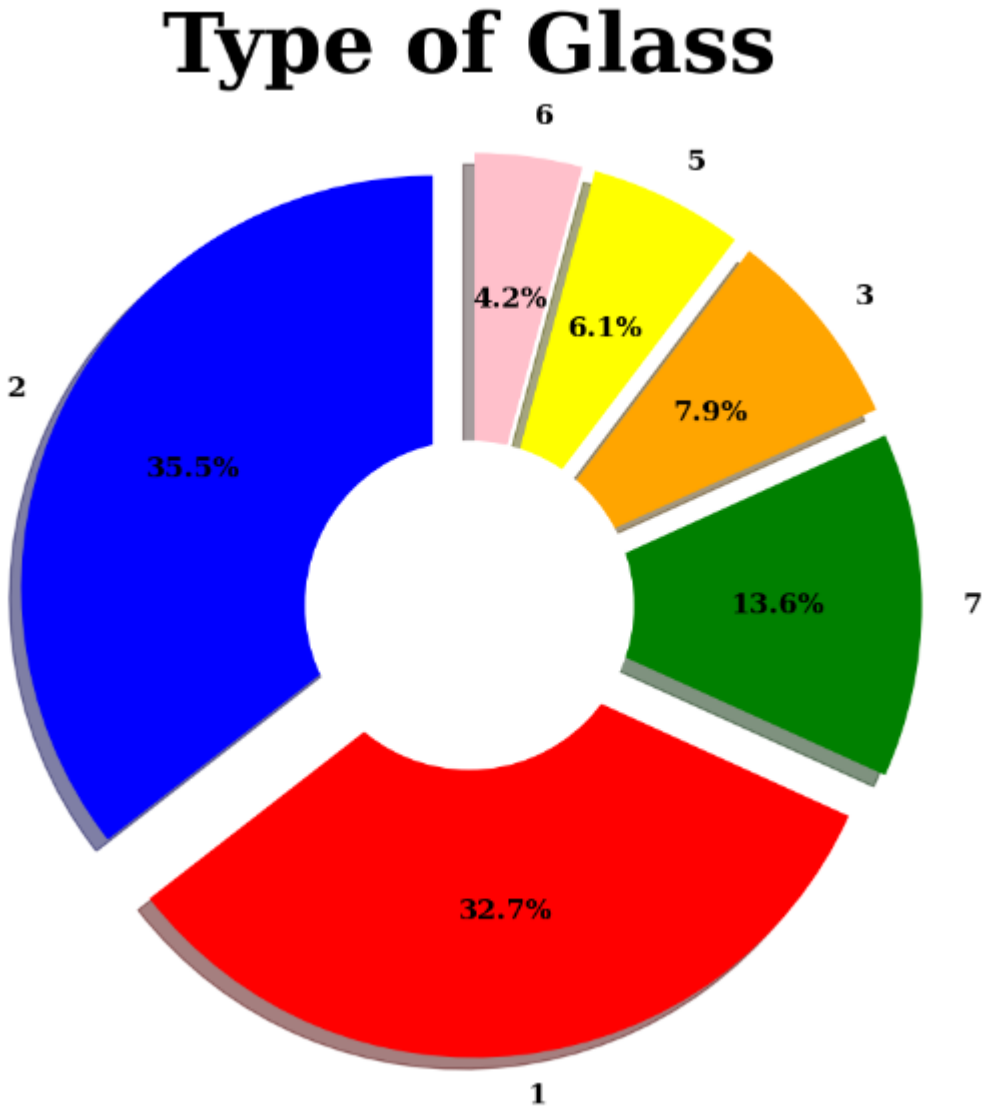
**Pie charts are used in data visualization for several reasons, including**:

- **Comparing proportions:** Pie charts are particularly useful for comparing the relative sizes of different categories or parts of a whole. They can quickly and easily show which categories are larger or smaller than others, and can help to identify patterns or trends in the data.

- **Providing a snapshot of data:** Pie charts provide a concise and intuitive snapshot of data, which can be useful for conveying information to a non-technical audience. They can also be used to quickly summarize complex data or findings.

- **Facilitating decision-making:** Pie charts can help to inform decision-making by providing a clear and easy-to-understand picture of the data. They can be used to identify areas that may require further investigation or attention, and can help to prioritize resources or actions.

Overall, pie charts are a useful tool in data visualization for displaying data that can be categorized or broken down into different parts. They provide a clear and intuitive way to compare the relative sizes of different categories, and can be used to inform decision-making and prioritize actions. However, it's important to use them judiciously, as they can become difficult to interpret when there are too many slices or the slices are too small.



# Kdeplot

A KDE (Kernel Density Estimation) plot is a type of data visualization technique that is used to visualize the distribution of a continuous variable. It is based on the kernel density estimation technique, which is a non-parametric way of estimating the probability density function of a random variable. In a KDE plot, the data is represented as a smooth curve, which is an estimate of the underlying probability density function.

KDE plots are particularly useful in EDA (Exploratory Data Analysis) as they allow analysts to visualize the distribution of data and identify patterns or trends that may not be visible in other types of plots. They can be used to compare the distribution of data across different groups or categories, and can help to identify potential outliers or unusual patterns in the data.

KDE plots can be created using various visualization libraries in Python, such as Seaborn, Matplotlib, and Plotly. Seaborn, in particular, provides a high-level interface to create visually appealing KDE plots with just a few lines of code, making it a popular choice for EDA.

**Some common uses of KDE plots include:**

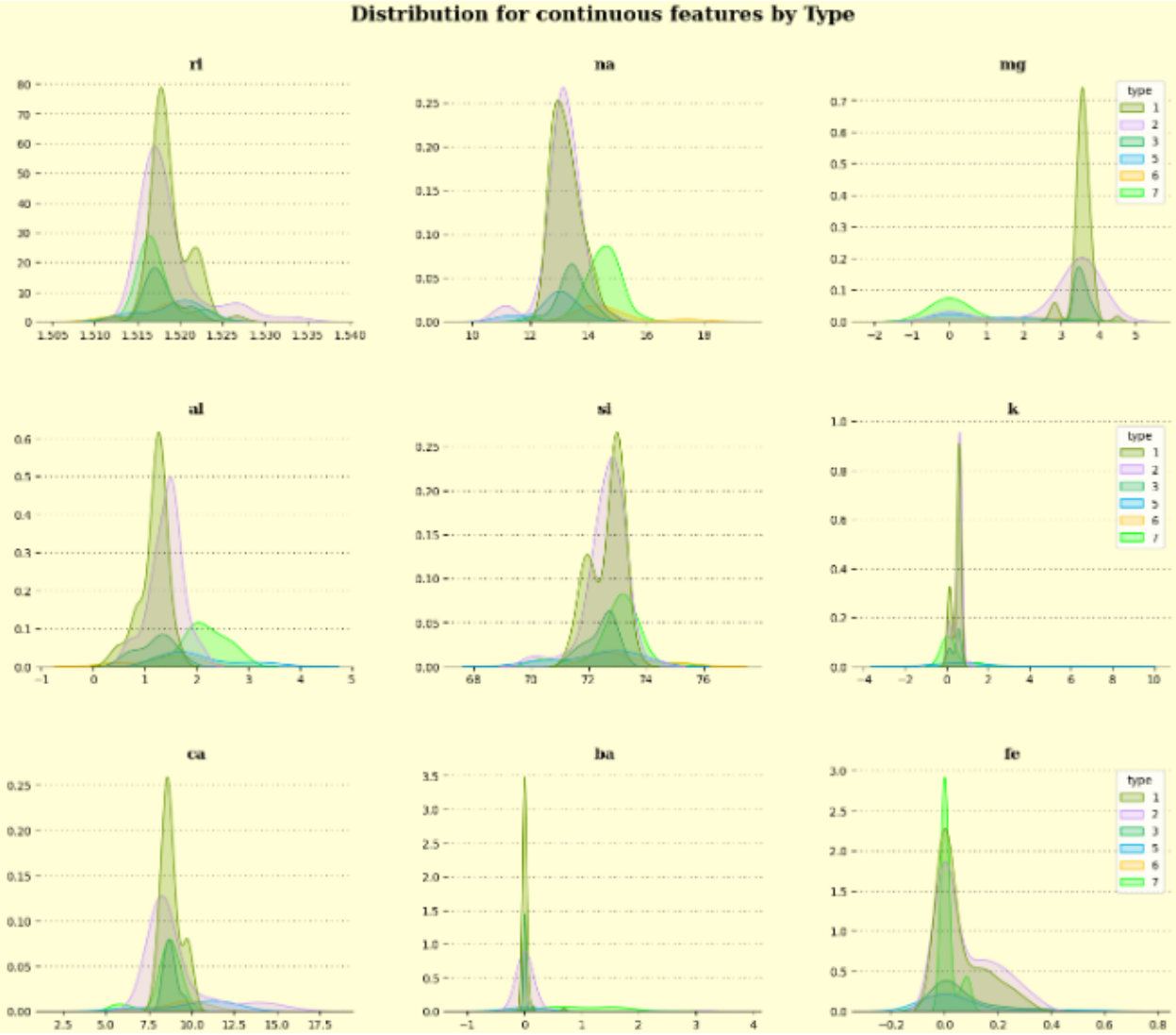
- **Visualizing the distribution of data:** KDE plots provide a smooth estimate of the underlying probability density function, which allows analysts to visualize the distribution of data and identify potential patterns or trends.

- **Comparing distributions:** KDE plots can be used to compare the distribution of data across different groups or categories. This can help to identify differences in the data and can provide insights into potential relationships or patterns.

- **Identifying outliers:** KDE plots can help to identify potential outliers in the data, which are data points that are significantly different from the rest of the data. Outliers can be important for identifying unusual patterns or errors in the data.

- **Providing a high-level overview:** KDE plots provide a concise and visually appealing summary of the distribution of data, which can be useful for providing a high-level overview of the data.

Overall, KDE plots are a useful tool in EDA for analyzing continuous data. They allow for quick and easy visualization of the distribution of data, and can be customized with various options to provide additional context and insights.



# *CHAPTER 4: COORELATION MATRIX*

A correlation matrix is a table that shows the correlation coefficients between different variables in a dataset. Correlation coefficients measure the strength and direction of the linear relationship between two variables, with values ranging from -1 to 1.

A positive value indicates a positive correlation, meaning that the variables tend to increase or decrease together. A negative value indicates a negative correlation, meaning that the variables tend to have an inverse relationship.

The importance of a correlation matrix lies in the insights it can provide into the relationships between variables in a dataset. It can help to identify which variables are strongly related to each other, which can be useful in understanding the underlying structure of the data and in developing predictive models.

**Some common applications of a correlation matrix include:**

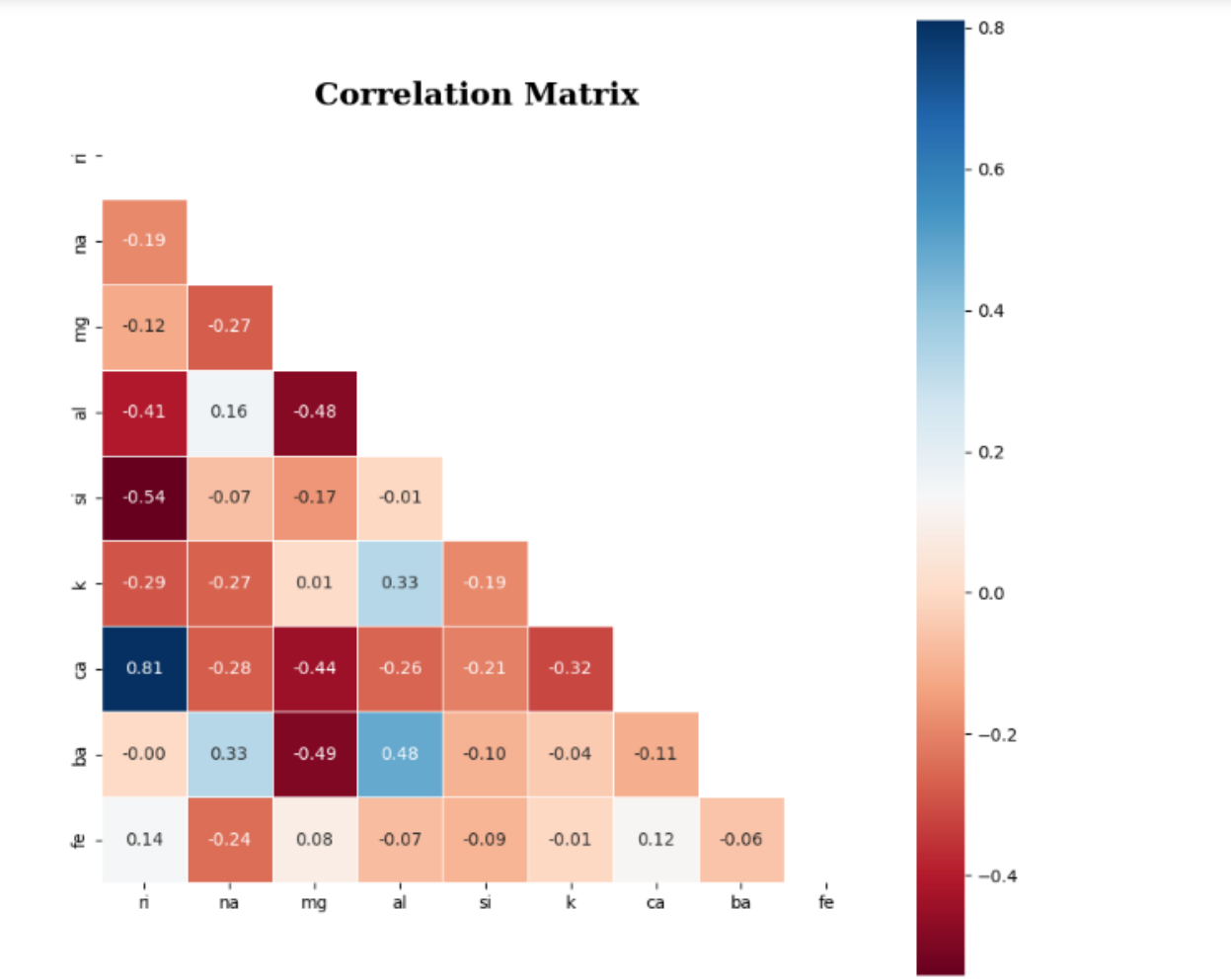
**Feature selection:** Correlation matrices can be used to identify which variables are strongly related to the target variable, which can be useful in selecting features for a predictive model.

**Multicollinearity detection:** Correlation matrices can also be used to identify multicollinearity, which occurs when two or more variables are highly correlated with each other. Multicollinearity can cause problems in some statistical models, such as linear regression, and may require corrective action.

**Exploratory data analysis:** Correlation matrices can provide insights into the underlying structure of the data, and can help to identify relationships that may not be visible in other types of data visualization.

**Hypothesis testing:** Correlation matrices can be used to test hypotheses about the relationships between variables, such as whether a particular variable is significantly correlated with the target variable.

Overall, a correlation matrix is an important tool in data analysis and modeling. It can help to identify important relationships between variables, which can be useful in developing predictive models and in gaining insights into the structure of the data. However, it is important to note that correlation does not necessarily imply causation, and care should be taken in interpreting the results of a correlation analysis.




In the above matrix we can see the relation between the elements of glass.
As the range is from -1 to 1 we can easily get a relation between the elements for further analysis and model building.

# SCATTER PLOT AND COR

A scatterplot is a type of graphical display that is used to represent the relationship between two variables. It is a plot that uses dots or points to represent the values of the two variables, with one variable being plotted along the x-axis and the other variable being plotted along the y-axis.

The importance of scatterplots lies in their ability to help us understand and visualize the relationship between two variables. By plotting the variables on a scatterplot, we can quickly see if there is a positive or negative correlation between the two variables, as well as the strength of that correlation.

Scatterplots also allow us to identify any outliers or clusters in the data, which can provide valuable insights into the underlying patterns of the data.
Scatterplots are commonly used in fields such as statistics, data science, economics, and social sciences, where they are used to analyze and interpret large datasets. They are also useful in business and marketing, where they can be used to identify trends and patterns in customer behavior, as well as to make predictions about future trends.

Overall, scatterplots are a powerful tool for visualizing data and gaining insights into complex relationships between variables

## CORELATION COFFECIENT(COR)

"Cor" in statistics is short for "correlation coefficient," which is a numerical measure that quantifies the strength and direction of the linear relationship between two variables. Correlation coefficients are values that range from -1 to 1, where a correlation of -1 indicates a perfectly negative linear relationship (i.e., as one variable increases, the other decreases), a correlation of 0 indicates no linear relationship, and a correlation of 1 indicates a perfectly positive linear relationship (i.e., as one variable increases, the other also increases).

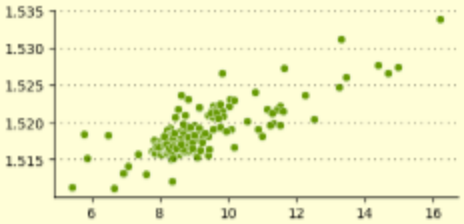
Correlation coefficients are used in a variety of statistical analyses, including regression analysis and hypothesis testing. They can be used to determine whether two variables are related, and if so, the direction and strength of that relationship. For example, in a medical study, a correlation coefficient might be used to determine if there is a relationship between a certain treatment and patient outcomes. In marketing research, a correlation coefficient might be used to determine if there is a relationship between advertising spend and sales.
It's important to note that correlation does not necessarily imply causation.

A correlation between two variables does not necessarily mean that one causes the other; rather, there may be other factors that are driving the relationship. Additionally, correlation coefficients are only appropriate for measuring linear relationships between variables; if the relationship is non-linear, other statistical techniques may be required.

In out project we have plot the scatter plot for comparing two elements and to find the correlation of the two elements.

**Scatter plot for Ca and RI**

**cor = 0.8104**

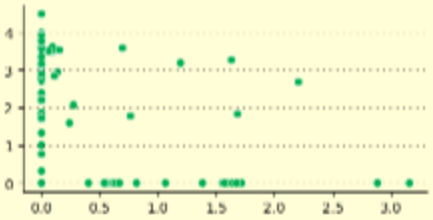



**Scatter plot for Si and RI**

**cor = -0.5421**


**Scatter plot for Ba and Mg**

**cor = -0.4923**

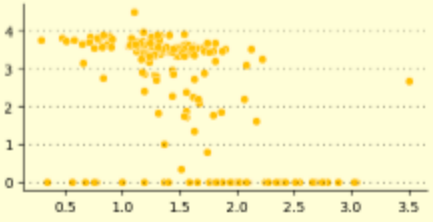



**Scatter plot for Ba and Al**

**cor = 0.4794**


**Scatter plot for Al and Mg**

**cor = -0.4818**



# ***CHAPTER 5: MODELING***

Modeling in machine learning refers to the process of building a mathematical representation or algorithm that can be used to make predictions or decisions based on input data. The process of modeling involves selecting an appropriate machine learning algorithm or approach, feeding in training data to the algorithm, and then refining the model until it accurately predicts or classifies new, unseen data.

The model is typically built using a training dataset, which contains both input data (also known as features) and their corresponding labels or outputs. The model then uses this data to learn patterns and relationships between the input and output variables. Once the model has been trained, it can be used to predict outcomes or make decisions on new data.

Modeling in machine learning involves several important steps, including data preparation and cleaning, feature selection, algorithm selection, hyperparameter tuning, and model evaluation. Data preparation and cleaning involves ensuring that the input data is in a suitable format for the algorithm to use. Feature selection involves choosing the most relevant variables to include in the model.

Algorithm selection involves selecting the appropriate machine learning algorithm for the problem at hand. Hyperparameter tuning involves selecting the best values for the parameters of the algorithm, such as the learning rate or regularization term. Model evaluation involves measuring the accuracy or performance of the model on a separate, unseen test dataset.

Overall, modeling is a crucial step in machine learning, as the quality and accuracy of the model can have a significant impact on the performance of the machine learning system. A well-designed and trained model can lead to better predictions or decisions, and ultimately, better outcomes in the problem domain.

**For the project we have used the following models:**

- Logistic Regression

- Random Forest
- Gradient Boosting
- Decision Tree
- K-Nearest Neighbors
- Support Vector Classifiers.

Lets get to know regarding each model for better understanding how the data has performed on the following models.

# LOGISTIC REGRESSION

A glass classifiers model based on logistic regression can be used to predict the type of glass based on certain characteristics. Logistic regression is a type of supervised learning algorithm used for classification problems, which means that it can be used to predict the class of a sample based on its features.

To create a glass classifiers model based on logistic regression, we would start by gathering a dataset that contains features of different types of glass and their corresponding classifications. Some features that might be included in the dataset could be the refractive index, sodium oxide content, magnesium oxide content, aluminum oxide content, silicon oxide content, potassium oxide content, calcium oxide content, and barium oxide content.

Once the dataset is collected and prepared, we would split it into training and testing sets. The training set would be used to train the logistic regression model to learn the relationship between the features and the classifications, while the testing set would be used to evaluate the performance of the model.

We would then use a logistic regression algorithm to train the model on the training set, adjusting the weights of the features to minimize the error between the predicted classifications and the actual classifications. Once the model is trained, we would use it to make predictions on the testing set and evaluate its

performance by comparing the predicted classifications to the actual classifications.

Overall, a glass classifiers model based on logistic regression can be a useful tool for predicting the type of glass based on its characteristics. However, the accuracy of the model will depend on the quality of the dataset and the chosen features, as well as the performance of the logistic regression algorithm.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
X = df.drop('type', axis=1)
y = df['type']

scaler = RobustScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,random_state=40)
```

```
logreg = LogisticRegression(random_state=1010)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
score_logreg = accuracy_score(y_pred, y_test)*100

print('Accuracy Score of Logistic Regression :', round(score_logreg,2))
```

```
Accuracy Score of Logistic Regression : 63.95
```

# RANDOM FOREST

Glass classifiers based on random forest are a type of machine learning model that can be used to classify different types of glass based on their chemical and physical properties. Random forest is an ensemble learning method that combines multiple decision trees to produce a more accurate and robust model.

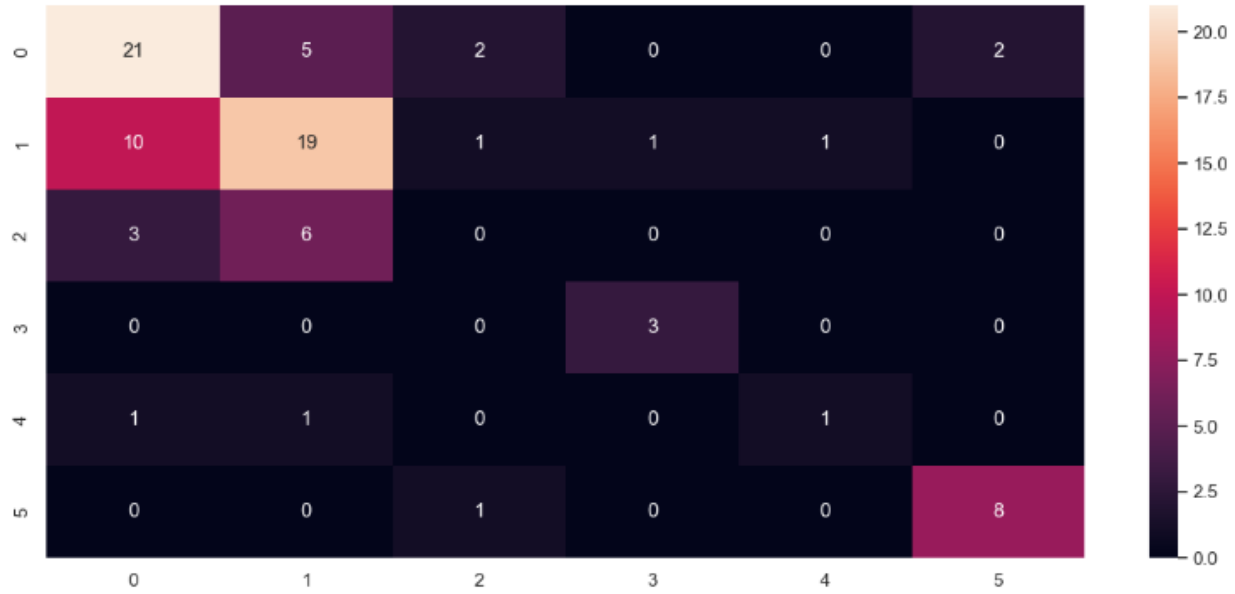
To build a glass classifier based on random forest, you would first need a dataset of glass samples with known properties, such as their refractive index, sodium, magnesium, and aluminum content, among others. You would then need to split the dataset into training and testing sets to evaluate the model's accuracy.

Next, you would use a random forest algorithm to train the model on the training data. The algorithm would randomly select subsets of the features and data points to create multiple decision trees. Each tree would vote on the glass type for each sample, and the final prediction would be the mode of all the tree predictions.

Finally, you would evaluate the model's accuracy on the testing set by comparing its predictions with the actual glass types. You could also use techniques like cross-validation to ensure that the model is robust and not overfitting to the training data.

Overall, a glass classifier based on random forest can be an effective way to identify different types of glass based on their chemical and physical properties.

```
rf= RandomForestClassifier(n_estimators= 10, criterion="entropy")
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize = (14,6))
sns.heatmap(cm_rf, annot = True)
plt.show()
score_rf = accuracy_score(y_pred, y_test)*100
print('Accuracy Score of Random Forest Classifier :', round(score_rf,2))
```




```
Accuracy Score of Random Forest Classifier : 67.44
```

# GRADIENT BOOSTING

Glass classifiers based on gradient boosting are another type of machine learning model that can be used to classify different types of glass based on their chemical and physical properties. Gradient boosting is also an ensemble learning method that combines multiple weak models (such as decision trees) to produce a more accurate and robust model.

To build a glass classifier based on gradient boosting, you would first need a dataset of glass samples with known properties, such as their refractive index, sodium, magnesium, and aluminum content, among others. You would then need to split the dataset into training and testing sets to evaluate the model's accuracy. Next, you would use a gradient boosting algorithm to train the model on the training data.

The algorithm would iteratively create new decision trees that try to correct the errors of the previous trees. Each new tree would focus on the samples that were incorrectly classified by the previous trees, thereby improving the overall accuracy of the model.

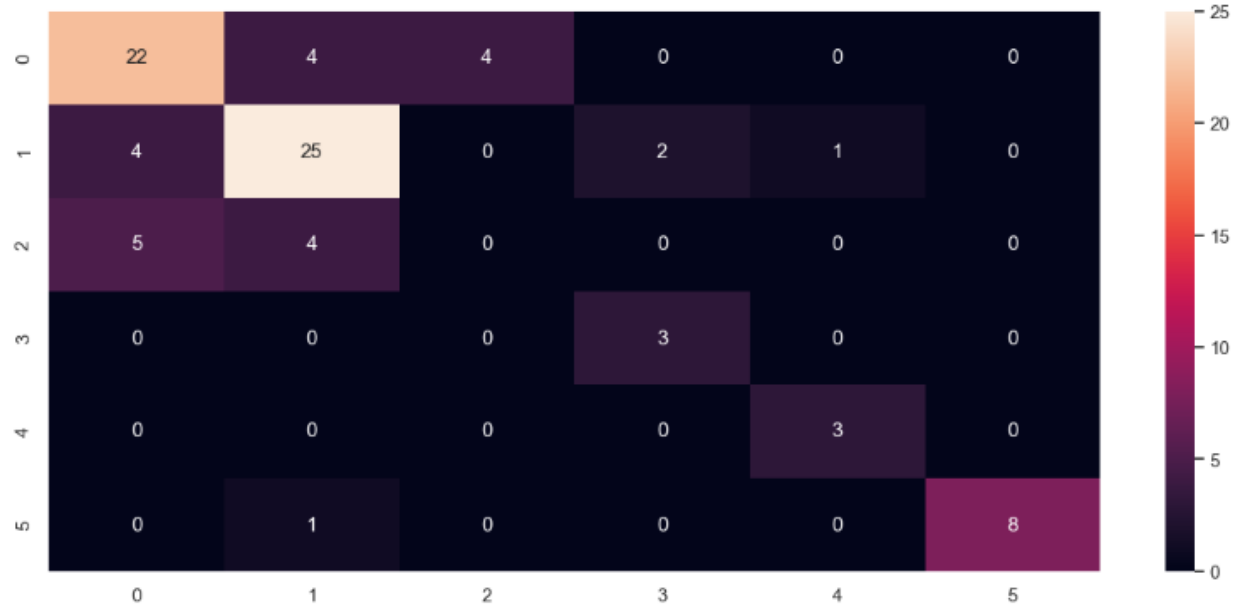
Finally, you would evaluate the model's accuracy on the testing set by comparing its predictions with the actual glass types. You could also use techniques like cross-validation to ensure that the model is robust and not overfitting to the training data.

Overall, a glass classifier based on gradient boosting can be a powerful and accurate way to identify different types of glass based on their chemical and physical properties.

However, gradient boosting can be more computationally expensive than other methods, so it may be less practical for large datasets or real-time applications.

```
gbc = GradientBoostingClassifier(random_state=200)
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)
score_gbc = accuracy_score(y_pred, y_test)*100
cm_gb = confusion_matrix(y_test, y_pred_gb)
plt.figure(figsize = (14,6))
sns.heatmap(cm_gb, annot = True)
plt.show()

print('Accuracy Score of Gradient Boosting Classifier :', round(score_gbc,2))
```




```
Accuracy Score of Gradient Boosting Classifier : 73.26
```

# DECISION TREE CLASSIFICATION

Glass classifiers based on decision tree are a type of machine learning model that can be used to classify different types of glass based on their chemical and physical properties. Decision trees are simple but powerful models that can represent complex decision-making processes based on a sequence of yes-or-no questions.

To build a glass classifier based on decision trees, you would first need a dataset of glass samples with known properties, such as their refractive index, sodium, magnesium, and aluminum content, among others. You would then need to split the dataset into training and testing sets to evaluate the model's accuracy.

Next, you would use a decision tree algorithm to train the model on the training data. The algorithm would iteratively split the data based on the most informative features (e.g., those with the highest information gain) to create a tree of decision rules. Each internal node of the tree would represent a decision based on a feature, and each leaf node would represent a glass type.

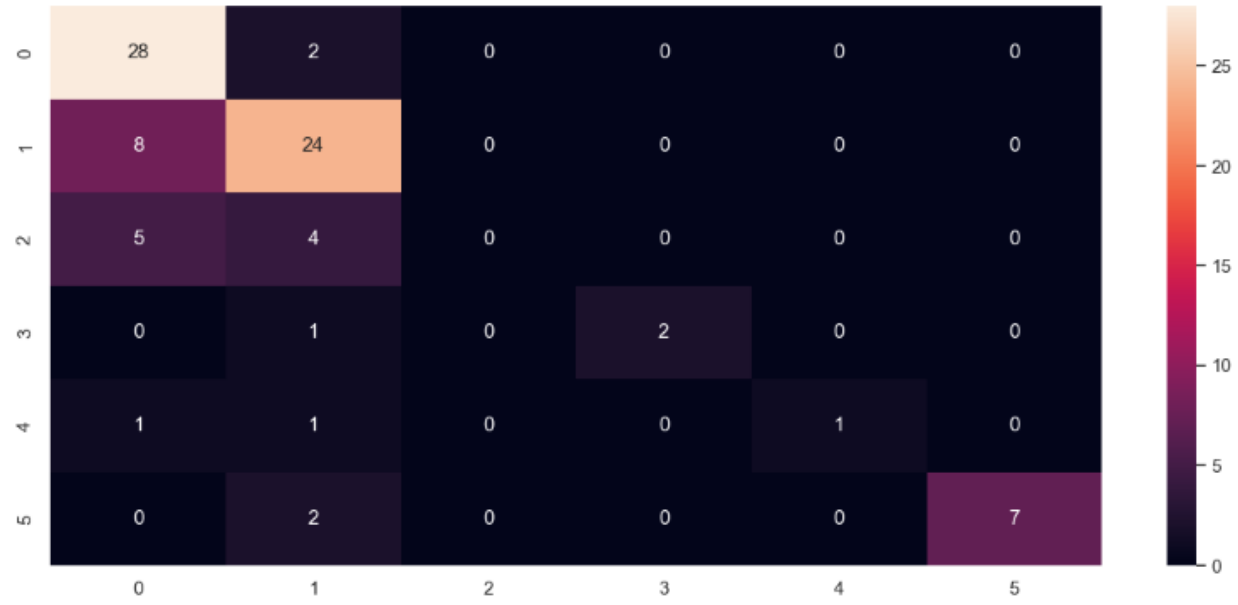
Finally, you would evaluate the model's accuracy on the testing set by comparing its predictions with the actual glass types. You could also use techniques like cross-validation to ensure that the model is robust and not overfitting to the training data.

Overall, a glass classifier based on decision tree can be a simple and interpretable way to identify different types of glass based on their chemical and physical properties. However, decision trees can be prone to overfitting to the training data, especially if the tree is deep or complex.

Therefore, it is important to use techniques like pruning or ensemble learning to avoid overfitting and improve the model's generalization performance.

```
dt = DecisionTreeClassifier(criterion='entropy', random_state=0)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
plt.figure(figsize = (14,6))
sns.heatmap(cm_dt, annot = True)
plt.show()
score_dt = accuracy_score(y_pred_rf, y_test)*100

print('Accuracy Score of Decision Tree Classifier :', round(score_dt,2))
```




```
Accuracy Score of Decision Tree Classifier : 60.47
```

# K-NEAREST NEIGHBOUR

Glass classifiers based on k-nearest neighbors (KNN) are a type of machine learning model that can be used to classify different types of glass based on their chemical and physical properties. KNN is a non-parametric algorithm that can be used for both classification and regression tasks.

To build a glass classifier based on KNN, you would first need a dataset of glass samples with known properties, such as their refractive index, sodium, magnesium, and aluminum content, among others. You would then need to split the dataset into training and testing sets to evaluate the model's accuracy.

Next, you would use the KNN algorithm to train the model on the training data. The algorithm would first calculate the distance between the test sample and each of the training samples based on their features. It would then identify the k-nearest training samples (i.e., the samples with the smallest distances) and assign the test sample to the glass type that is most common among the k-nearest neighbors.

Finally, you would evaluate the model's accuracy on the testing set by comparing its predictions with the actual glass types. You could also use techniques like cross-validation to ensure that the model is robust and not overfitting to the training data.

Overall, a glass classifier based on KNN can be a simple and effective way to identify different types of glass based on their chemical and physical properties. However, KNN can be sensitive to the choice of k, which can affect both the model's accuracy and its computational efficiency.

Therefore, it is important to use techniques like grid search or cross-validation to optimize the choice of k and ensure the best performance of the model.

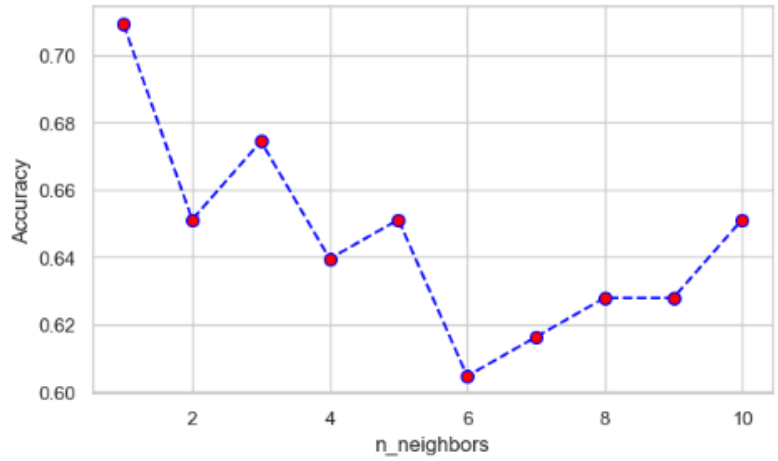
```
# Find n_neighbors for best score
accuracy = []

for i in range(1, 11):
    model = KNeighborsClassifier(n_neighbors = i)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy.append(model.score(X_test, y_test))

score_knn = max(accuracy)*100
print('Accuracy Score of K-Nearest Neighbors Classifier : ', round(score_knn,2))

plt.figure(figsize=(7, 4))
plt.plot(range(1,11), accuracy, linestyle='dashed', marker='o', color='blue',
         markersize=7, markerfacecolor='red')
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
sns.set_theme(style='whitegrid')
plt.show()
```

```
Accuracy Score of K-Nearest Neighbors Classifier :  70.93
```



# SUPPORT VECTOR CLASSIFIER

Glass classifiers based on support vector machines (SVMs) are a type of machine learning model that can be used to classify different types of glass based on their chemical and physical properties. SVMs are a powerful and versatile algorithm that can be used for both linear and non-linear classification tasks.

To build a glass classifier based on SVMs, you would first need a dataset of glass samples with known properties, such as their refractive index, sodium, magnesium, and aluminum content, among others. You would then need to split the dataset into training and testing sets to evaluate the model's accuracy.

Next, you would use the SVM algorithm to train the model on the training data. The algorithm would try to find a hyperplane in the feature space that can separate the different glass types with the largest margin. In cases where the data is not linearly separable, the algorithm would use a kernel function to map the data into a higher-dimensional space where it can be separated by a hyperplane.

Finally, you would evaluate the model's accuracy on the testing set by comparing its predictions with the actual glass types. You could also use techniques like cross-validation to ensure that the model is robust and not overfitting to the training data.

Overall, a glass classifier based on SVMs can be a powerful and accurate way to identify different types of glass based on their chemical and physical properties. However, SVMs can be sensitive to the choice of kernel and the regularization parameter, which can affect both the model's accuracy and its computational efficiency.

Therefore, it is important to use techniques like grid search or cross-validation to optimize the choice of hyperparameters and ensure the best performance of the model.

```
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
score_svm = accuracy_score(y_pred, y_test)*100

print('Accuracy Score of Support Vector Classifier :', round(score_svm, 2))
```

```
Accuracy Score of Support Vector Classifier : 72.09
```

# *CHAPTER 6: COMAPRISON OF MODEL*

**Random Forest and Gradient Boosting:** Random Forest and Gradient Boosting are both ensemble methods that use multiple decision trees to classify the glass types. These models can handle complex, high-dimensional data and are less prone to overfitting than single decision trees. They are also less sensitive to hyperparameters than other models, making them easier to tune.

**Decision Tree:** Decision tree classifiers are simple and interpretable models that can be used to classify different types of glass based on their chemical and physical properties. They are easy to understand and can be visualized, making them useful for exploring the underlying decision-making process. However, they are prone to overfitting and can become too complex for high-dimensional data.

**KNN:** KNN classifiers are based on the idea that similar glass types are more likely to belong to the same class. This method can be effective for low-dimensional data and can be easily implemented. However, it can be computationally expensive for high-dimensional data and can be sensitive to the choice of k.

**SVMs:** SVM classifiers are based on finding a hyperplane that separates different glass types in the feature space with the largest margin. This method can handle both linear and non-linear data and can be robust to outliers. However, it can be sensitive to the choice of kernel and regularization parameter.

Overall, the choice of the machine learning model depends on the specific characteristics of the glass dataset and the goals of the analysis. Random Forest and Gradient Boosting can be effective for complex, high-dimensional data, while Decision Trees can be useful for exploring the decision-making process.

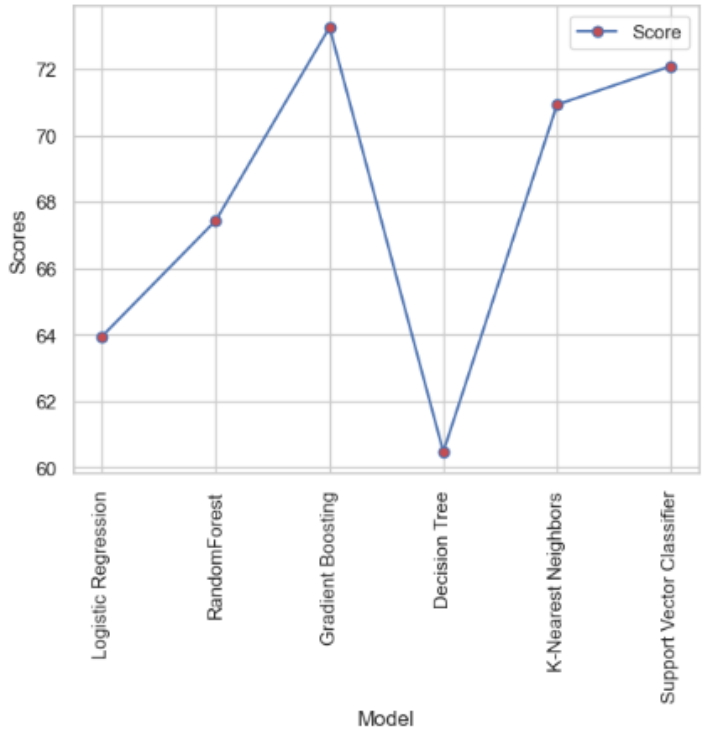
KNN can be effective for low-dimensional data, while SVMs can handle both linear and non-linear data. It is also important to consider the computational cost and ease of implementation when selecting a model.

```
In [55]: mes=pd.DataFrame({'Model' : ['Logistic Regression', 'RandomForest', 'Gradient Boosting',
                                      'Decision Tree', 'K-Nearest Neighbors', 'Support Vector Classifier'],
                           'Score' : [round(score_logreg,2),round(score_rf,2), round(score_gbc,2), round(score_dt,2), round(score_knn,2), roun
         mes
```

Out[55]:

| | Model | Score |
|---|---|---|
| 0 | Logistic Regression | 63.95 |
| 1 | RandomForest | 67.44 |
| 2 | Gradient Boosting | 73.26 |
| 3 | Decision Tree | 60.47 |
| 4 | K-Nearest Neighbors | 70.93 |
| 5 | Support Vector Classifier | 72.09 |

```
plt.plot(mes['Score'],'-o',mfc='r')
labels=mes['Model']
x=[0,1,2,3,4,5]
plt.xticks(x, labels, rotation ='vertical')
plt.legend(['Score','Model'])
plt.xlabel("Model")
plt.ylabel("Scores")
plt.show()
sns.set_theme(style='whitegrid')
```



# ***CONCLUSION***

In conclusion, glass classifiers based on machine learning models can be effective and accurate ways to identify different types of glass based on their chemical and physical properties.

Random Forest and Gradient Boosting are both ensemble methods that can handle complex, high-dimensional data and are less prone to overfitting than single decision trees. Decision Tree classifiers are simple and interpretable models that can be useful for exploring the underlying decision-making process.

KNN classifiers are based on the idea that similar glass types are more likely to belong to the same class, making them effective for low-dimensional data. SVM classifiers are based on finding a hyperplane that separates different glass types in the feature space with the largest margin and can handle both linear and non-linear data.

When selecting a model, it is important to consider the specific characteristics of the glass dataset and the goals of the analysis. The computational cost and ease of implementation are also important factors to consider. Techniques like grid search or cross-validation can be used to optimize the choice of hyperparameters and ensure the best performance of the model.

Overall, glass classifiers based on machine learning models can provide a powerful tool for glass classification and can have broad applications in various industries, including manufacturing, engineering, and materials science.