Part 1 Analysis

Listing 1 - Loading and Reading the glass dataset

```
# Load dataset
filename = 'glass/glass.data'
names = ['Id', 'Refractive Index(RI)',
         'Sodium(Na)', 'Magnesium(Mg)',
         'Aluminum(Al)', 'Silicon(Si)',
         'Potassium(K)', 'Calcium(Ca)',
         'Barium(Ba)', 'Iron(Fe)', 'glassType']

dataset = read_csv(filename, names=names)

dataset = dataset.drop(['Id'], axis =1)
```

The “filename” specifies the location of the dataset file. The “names” contains a list of the column names for the dataset. This is important because the dataset being loaded might not include headers, and this explicitly defines what each column represents.

The columns are as follows: *Id*, a unique identifier for each record, which will later be dropped. *Refractive Index (RI),* a physical property of the glass. Another set of property are the chemical composition of the glass such as *Sodium (Na)*, *Magnesium (Mg)*, *Aluminum (Al)*, *Silicon (Si)*, *Potassium (K)*, *Calcium (Ca)*, *Barium (Ba)*, *Iron (Fe)*. And finally the *glassType*, which represents the type of glass.

The “read_csv” function is used to read the dataset from the file “filename”. It assigns the loaded data to a DataFrame called “dataset”. The names=names argument tells the function to use the predefined list of column names as the header for the dataset. If the file does not contain column headers, this ensures that the dataset is properly labeled.

The 'Id' column is then removed from the dataset. Since the 'Id' column is just a unique identifier and does not contain any useful information for analysis, it is dropped. This preprocessing step prepares the dataset for machine learning tasks and by dropping the 'Id' column, we ensure that only relevant features (RI, chemical composition, and glass type) are kept for further analysis.

Listing 2 - Dimensions of the dataset. Peek at the data itself. Statistical summary of all attributes. Breakdown of the data by the class variable.

a) Print the shape of the dataset.

```
# Summarize Data
# shape
print(dataset.shape)
```

```
(214, 10)
```

The line of code above, "print(dataset.shape)", outputs the **shape** of the dataset, which includes two numbers: the number of rows (samples) and the number of columns (features and target variable). This is a simple yet crucial step to understand the size of the dataset, as it tells us how many data points we are working with and how many features are available.

b) Print the first few rows of the dataset.

```
# print the first 10 rows of the data
print(dataset.head(10))
```

```
   Refractive Index(RI)  Sodium(Na)  ...  Iron(Fe)  glassType
0               1.52101       13.64  ...      0.00          1
1               1.51761       13.89  ...      0.00          1
2               1.51618       13.53  ...      0.00          1
3               1.51766       13.21  ...      0.00          1
4               1.51742       13.27  ...      0.00          1
5               1.51596       12.79  ...      0.26          1
6               1.51743       13.30  ...      0.00          1
7               1.51756       13.15  ...      0.00          1
8               1.51918       14.04  ...      0.00          1
9               1.51755       13.00  ...      0.11          1

[10 rows x 10 columns]
```

The "print(dataset.head(10))" command is used to print the **first 10 rows** of the dataset. This function allows us to preview the structure and contents of the dataset. By displaying the first few entries, we can confirm that the data has been loaded correctly and that the column names are accurately assigned

c) Print the statistical descriptions of the dataset.

```
# Descriptive statistics
print(dataset.describe())
```

```
       Refractive Index(RI)  Sodium(Na)  ...    Iron(Fe)   glassType
count            214.000000  214.000000  ...  214.000000  214.000000
mean               1.518365   13.407850  ...    0.057009    2.780374
std                0.003037    0.816604  ...    0.097439    2.103739
min                1.511150   10.730000  ...    0.000000    1.000000
25%                1.516522   12.907500  ...    0.000000    1.000000
50%                1.517680   13.300000  ...    0.000000    2.000000
75%                1.519157   13.825000  ...    0.100000    3.000000
max                1.533930   17.380000  ...    0.510000    7.000000

[8 rows x 10 columns]
```

The "print(dataset.describe())" command generates **descriptive statistics** for the numerical columns in the dataset. This function returns useful metrics such as the mean, standard deviation, minimum and maximum values, and percentiles (25th, 50th, and 75th). These summary statistics give an overview of the central tendencies and spread of the data, allowing us to detect potential issues such as outliers or large variability within the features.

d) Print the statistical descriptions of the dataset.

```
# class distribution by glass ttype
print (dataset.groupby('glassType').size())
```

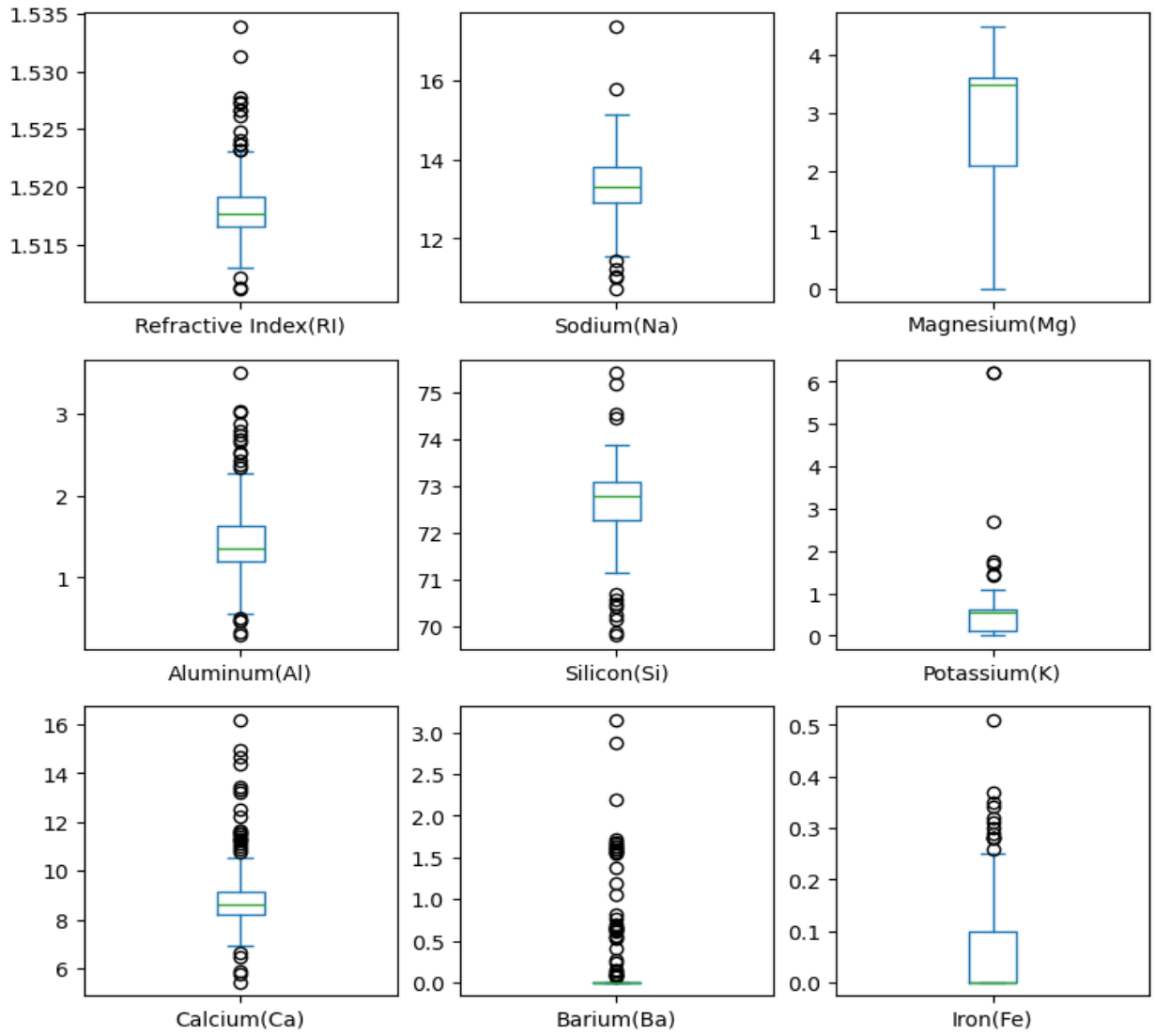
```
glassType
1    70
2    76
3    17
5    13
6     9
7    29
dtype: int64
```

The command "print(dataset.groupby('glassType').size())" calculates the **class distribution** of the target variable, glassType, by grouping the data based on the glass type and counting the number of instances in each class. This step is critical for understanding the balance of the dataset. If certain classes are overrepresented while others are underrepresented, it may indicate a class imbalance, which could impact the performance of machine learning models trained on the data.

Listing 3 - Univariate plots to better understand each attribute. Multivariate plots to better understand the relationships between attributes.

a) Univariate plot.

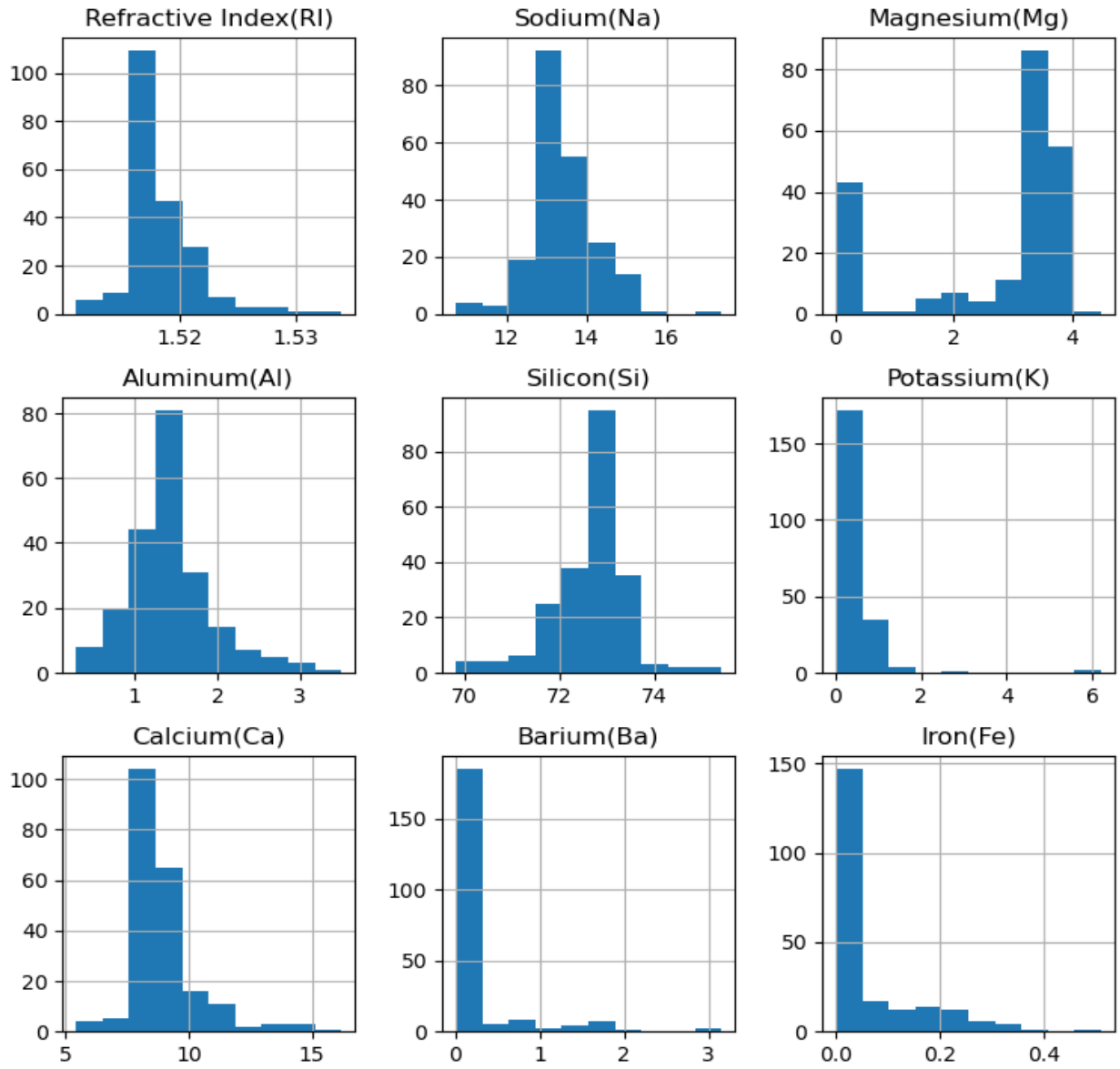
```
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(3,3), figsize=(9, 9), sharex=False, sharey=False)
pyplot.show()
```



The image above uses **box plots** with "dataset.plot(kind='box', subplots=True, layout=(3,3), figsize=(9, 9), sharex=False, sharey=False)". Box plots are a great way to visualize the **spread of data** and **detect outliers**. The box represents the interquartile range (IQR), which is the range between the 25th and 75th percentiles, while the line inside the box shows the median. By creating a box plot for each feature, you can easily compare their distributions and check for any significant **outliers** or skewed distributions. The box plots suggest that many features exhibit outliers, and some are skewed.

b) Visualize the dataset using histogram plots.

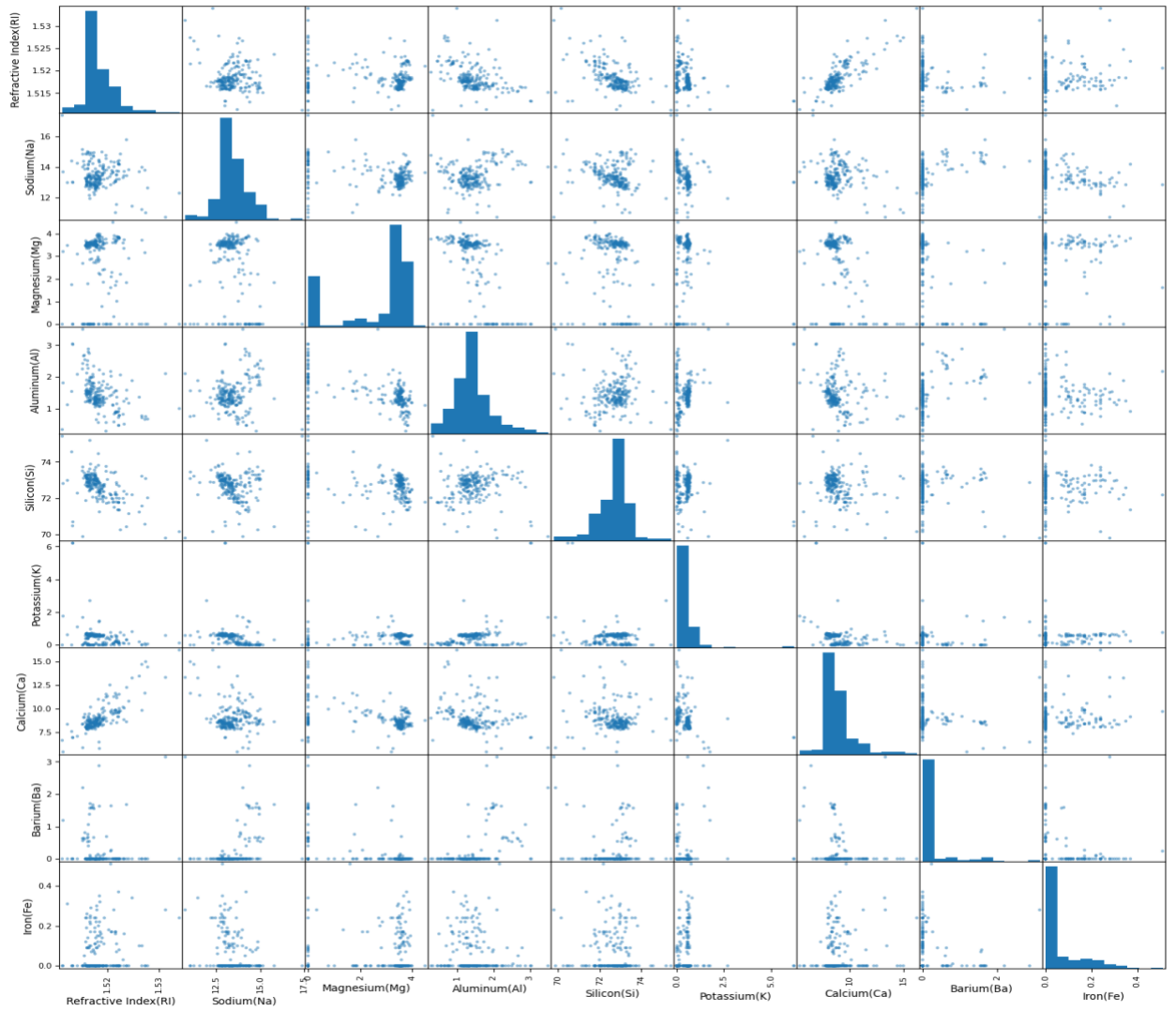
```
# histograms
dataset.hist(figsize=(9, 9))
pyplot.show()
```



Next, the code generates **histograms** for each feature using "dataset.hist(figsize=(9, 9))". A histogram shows the **frequency distribution** of a variable by grouping data into bins. This allows for an easy visualization of the **shape** of the data distribution (e.g., normal, skewed, or multimodal). By plotting histograms for all the features, you can observe how values are distributed and whether they follow a particular pattern, such as being **normally distributed** or having a skewed or uniform distribution. The histograms indicate that several features, such as sodium, potassium, and magnesium, have **right-skewed distributions**, meaning that most of the data points are clustered toward the lower values with a few high-value outliers.

c) Visualize the dataset using scatter plots.

```
# scatter plot matrix
scatter_matrix(dataset, figsize =(18,18))
pyplot.show()
```



Finally, the code produces a **scatter plot matrix** using "scatter_matrix(dataset, figsize=(18,18))". A scatter plot matrix creates pairwise scatter plots for all combinations of features, providing a visual representation of the **relationships between variables**. Each scatter plot shows how two variables are related—whether there is a linear relationship, a clustered pattern, or no clear association. This matrix is especially useful for detecting **correlations** between variables and understanding if there are any underlying patterns or structures in the data. For instance, if the scatter plot shows a diagonal line, it indicates a strong linear relationship between the two variables. The scatter plot matrix shows that **most feature pairs do not exhibit strong linear correlations**, as evidenced by the scattered nature of the points. However, there are clusters and patterns visible in certain pairs, which might suggest potential underlying groupings or relationships that could be explored further using clustering or classification techniques.