

# CS 5710 Machine Learning

## Assignment-2

**Name:** Keerthi Reddy  
Gannapureddy

**ID:** 700743921

**GitHub Link:** <https://github.com/Keerthireddy860/Machine-Learning-Assignment-2>

### Importing the required libraries.

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
import random as rnd

# Loading the warnings module
import warnings
# setting a filter to ignore warnings that gets generated by the code
warnings.filterwarnings("ignore")

# Data Visualization Modules
import seaborn as sns
import matplotlib.pyplot as plt

# machine Learning modules
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing

# function is used to split the data into training and testing sets for machine learning modeling
from sklearn.model_selection import train_test_split

# importing functions used to evaluate the performance of machine learning models
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

## 1. Pandas:

### Question 1:

Read the provided CSV file '**data.csv**'.

<https://drive.google.com/drive/folders/1h8C3mLsso-R-sIOLsvoYwPLzy2fJ4IOF?usp=sharing>

#### *Source code:*

```
# Reading the CSV file using read_csv( ) function
df = pd.read_csv('data.csv')

# Printing the first 5 rows of the DataFrame
df.head()
```

**Output:**

|   | Duration | Pulse | Maxpulse | Calories |
|---|----------|-------|----------|----------|
| 0 | 60       | 110   | 130      | 409.1    |
| 1 | 60       | 117   | 145      | 479.0    |
| 2 | 60       | 103   | 135      | 340.0    |
| 3 | 45       | 109   | 175      | 282.4    |
| 4 | 45       | 117   | 148      | 406.0    |

**Explanation:**

Here in the code, I have used ***read\_csv()*** and ***head()*** functions to read CSV file and to display the first 5 rows of the *DataFrame* respectively.

**Question 2:**

Show the basic statistical description about the data.

**Source code:**

```
# Getting the basic statistical description about the data
description = df.describe()

# Printing the description
description
```

**Output:**

|       | Duration   | Pulse      | Maxpulse   | Calories    |
|-------|------------|------------|------------|-------------|
| count | 169.000000 | 169.000000 | 169.000000 | 164.000000  |
| mean  | 63.846154  | 107.461538 | 134.047337 | 375.790244  |
| std   | 42.299949  | 14.510259  | 16.450434  | 266.379919  |
| min   | 15.000000  | 80.000000  | 100.000000 | 50.300000   |
| 25%   | 45.000000  | 100.000000 | 124.000000 | 250.925000  |
| 50%   | 60.000000  | 105.000000 | 131.000000 | 318.600000  |
| 75%   | 60.000000  | 111.000000 | 141.000000 | 387.600000  |
| max   | 300.000000 | 159.000000 | 184.000000 | 1860.400000 |

**Explanation:**

Here in the code, I have used the ***describe()*** function to generate descriptive statistics of a *DataFrame*. Then the description is printed.

### **Question 3:**

Check if the data has null values.

**Source code:**

```
print('Are there any null values: ', df.isnull().values.any())

# Checking for null values in the DataFrame
null_values = df.isnull().sum()

# Printing the number of null values for each column
null_values
```

**Output:**

```
Are there any null values:  True

Duration      0
Pulse         0
Maxpulse      0
Calories      5
dtype: int64
```

**Explanation:**

Here in the code, I have used the *isnull()* function to identify null values in the *DataFrame*, and the *values.any()* function to check if any of the values are null.

Then *sum()* function is called on this Boolean *DataFrame* and displayed a new *DataFrame* containing the sum of **True** values for each column in *df*.

### **Question 3 (a):**

Replace the null values with the mean.

**Source code:**

```
# Replacing null values with the mean of the respective column
mean_values = df.mean()
df.fillna(mean_values, inplace=True)

print('Are there any null values after replacing: ', df.isnull().values.any())

# Checking for null values in the DataFrame (should return all 0s)
null_values = df.isnull().sum()

# Printing the number of null values for each column
null_values
```

**Output:**

```
Are there any null values after replacing: False
Duration      0
Pulse         0
Maxpulse      0
Calories      0
dtype: int64
```

**Explanation:**

Here in the code, I have used **mean()** function to calculate the mean of each column in the DataFrame and **fillna()** function is used with calculated mean to replace all null values in **df**. Then used the **isnull()** function to identify null values in the *DataFrame*, and the **values.any()** function to check if any of the values are null.

**sum()** function is called on this Boolean DataFrame and displayed a new DataFrame containing the sum of **True** values for each column in **df**.

**Question 4:**

Select at least two columns and aggregate the data using: min, max, count, mean.

**Source code:**

```
# Selecting two columns maxpulse, calories and aggregating using min, max, count, and mean
agg_df = df.agg({'Maxpulse': ['min', 'max', 'count', 'mean'],
                 'Calories': ['min', 'max', 'count', 'mean']})

# Printing the aggregated data
agg_df
```

**Output:**

|       | Maxpulse   | Calories    |
|-------|------------|-------------|
| min   | 100.000000 | 50.300000   |
| max   | 184.000000 | 1860.400000 |
| count | 169.000000 | 169.000000  |
| mean  | 134.047337 | 375.790244  |

**Explanation:**

Here in the code, I have used **agg()** function to perform the aggregation operations on the selected columns, with a dictionary as an argument, where the keys are the column names to be aggregated and the values are lists of aggregation functions to be applied to each column.

Finally, the aggregated DataFrame is printed.

**Question 5:**

Filter the DataFrame to select the rows with calories values between 500 and 1000.

*Source code:*

```
# Filtering the DataFrame to select rows with calorie values between 500 and 1000
filtered_df = df.loc[(df['Calories'] >= 500) & (df['Calories'] <= 1000)]

# Printing the filtered DataFrame
filtered_df
```

*Output:*

|     | Duration | Pulse | Maxpulse | Calories |
|-----|----------|-------|----------|----------|
| 51  | 80       | 123   | 146      | 643.1    |
| 62  | 160      | 109   | 135      | 853.0    |
| 65  | 180      | 90    | 130      | 800.4    |
| 66  | 150      | 105   | 135      | 873.4    |
| 67  | 150      | 107   | 130      | 816.0    |
| 72  | 90       | 100   | 127      | 700.0    |
| 73  | 150      | 97    | 127      | 953.2    |
| 75  | 90       | 98    | 125      | 563.2    |
| 78  | 120      | 100   | 130      | 500.4    |
| 83  | 120      | 100   | 130      | 500.0    |
| 90  | 180      | 101   | 127      | 600.1    |
| 99  | 90       | 93    | 124      | 604.1    |
| 101 | 90       | 90    | 110      | 500.0    |
| 102 | 90       | 90    | 100      | 500.0    |
| 103 | 90       | 90    | 100      | 500.4    |
| 106 | 180      | 90    | 120      | 800.3    |
| 108 | 90       | 90    | 120      | 500.3    |

*Explanation:*

Here in the code, the ***loc[ ]*** function is used to select rows based on a boolean condition. The condition is specified inside the square brackets of the ***loc[ ]*** function using the 'Calories' column of ***df***. Specifically, ***df[Calories] >= 500*** and ***df[Calories] <= 1000*** are two separate boolean conditions, which are combined using the ***&*** operator to specify the filter condition.

Finally, the filtered DataFrame is printed.

### **Question 6:**

Filter the DataFrame to select the rows with calories values  $> 500$  and pulse  $< 100$ .

#### ***Source code:***

```
# Filtering the DataFrame to select rows with calorie values > 500 and pulse values < 100
filtered_df = df.loc[(df['Calories'] > 500) & (df['Pulse'] < 100)]

# Printing the filtered DataFrame
filtered_df
```

#### ***Output:***

|     | Duration | Pulse | Maxpulse | Calories |
|-----|----------|-------|----------|----------|
| 65  | 180      | 90    | 130      | 800.4    |
| 70  | 150      | 97    | 129      | 1115.0   |
| 73  | 150      | 97    | 127      | 953.2    |
| 75  | 90       | 98    | 125      | 563.2    |
| 99  | 90       | 93    | 124      | 604.1    |
| 103 | 90       | 90    | 100      | 500.4    |
| 106 | 180      | 90    | 120      | 800.3    |
| 108 | 90       | 90    | 120      | 500.3    |

#### ***Explanation:***

Here in the code, the **loc[ ]** function is used to select rows based on a boolean condition. The condition is specified inside the square brackets of the **loc[ ]** function using the 'Calories' and 'Pulse' columns of *df*. Specifically, **df['Calories'] > 500** and **df['Pulse'] < 100** are two separate boolean conditions, which are combined using the **&** operator to specify the filter condition.

Finally, the filtered DataFrame is printed.

### **Question 7:**

Create a new “*df\_modified*” dataframe that contains all the columns from *df* except for “*Maxpulse*”.

#### ***Source code:***

```
# Dropping the 'Maxpulse' column and creating a new DataFrame
df_modified = df.drop(columns=['Maxpulse'])

# Printing the first 5 rows of new DataFrame
df_modified.head()
```

**Output:**

|   | Duration | Pulse | Calories |
|---|----------|-------|----------|
| 0 | 60       | 110   | 409.1    |
| 1 | 60       | 117   | 479.0    |
| 2 | 60       | 103   | 340.0    |
| 3 | 45       | 109   | 282.4    |
| 4 | 45       | 117   | 406.0    |

**Explanation:**

Here in the code, I have used **drop()** function is used to remove the specified column from the DataFrame, and the *columns* parameter is used to specify the name of the column to drop.

The resulting DataFrame is stored in **df\_modified** and is displayed.

**Question 8:**

Delete the “Maxpulse” column from the main *df* dataframe.

**Source code:**

```
# Dropping the 'Maxpulse' column from the original DataFrame
df.drop(columns=['Maxpulse'], inplace=True)

#printing the modified dataframe
df.head()
```

**Output:**

|   | Duration | Pulse | Calories |
|---|----------|-------|----------|
| 0 | 60       | 110   | 409.1    |
| 1 | 60       | 117   | 479.0    |
| 2 | 60       | 103   | 340.0    |
| 3 | 45       | 109   | 282.4    |
| 4 | 45       | 117   | 406.0    |

**Explanation:**

Here in the code, I have used **drop()** function is used with **inplace** parameter to remove the specified column from the DataFrame and save it to the original DataFrame, and the *columns* parameter is used to specify the name of the column to drop.

The resulting DataFrame is printed.

**Question 9:**

Convert the datatype of *Calories* column to *int* datatype.

***Source code:***

```
# astype() function is used to convert the data type
df['Calories'] = df['Calories'].astype('int64')
df.dtypes
```

***Output:***

```
Duration    int64
Pulse       int64
Calories    int64
dtype: object
```

***Explanation:***

Herein the code, I have used **astype()** function on the *Calories* column of **df** DataFrame to convert the data type to integer. **dtypes** attribute is used to display the data types.

**Question 10:**

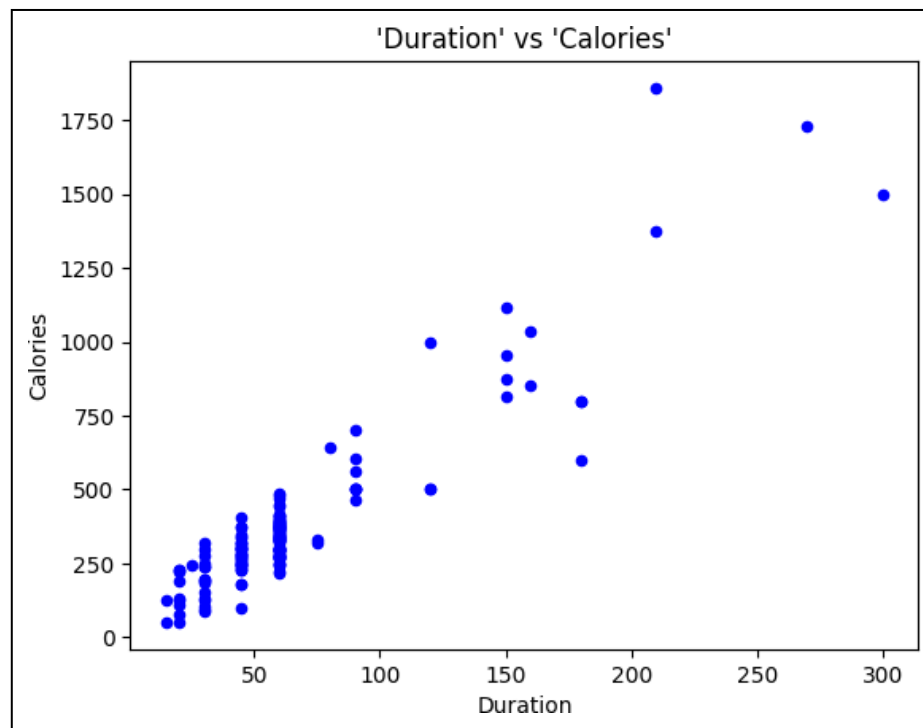
Using pandas create a scatter plot for the two columns (Duration and Calories).

***Source code:***

```
# Creating a scatter plot for 'Duration' vs 'Calories'
df.plot(x = 'Duration',
        y = 'Calories',
        kind = 'scatter',
        title = "'Duration' vs 'Calories'",
        c = 'blue')

# displaying the plot
plt.show()
```



**Output:**

## **1. (Glass Dataset):**

### **Question 1:**

Implement **Naïve Bayes method** using scikit-learn library.

- a. Use the glass dataset available in Link also provided in your assignment.

**Source code:**

```
# Loading the dataset
glass_df = pd.read_csv("glass.csv")
glass_df.head()
```

**Output:**

|   | 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 | 0.0 | 1    |
| 1 | 1.51761 | 13.89 | 3.60 | 1.36 | 72.73 | 0.48 | 7.83 | 0.0 | 0.0 | 1    |
| 2 | 1.51618 | 13.53 | 3.55 | 1.54 | 72.99 | 0.39 | 7.78 | 0.0 | 0.0 | 1    |
| 3 | 1.51766 | 13.21 | 3.69 | 1.29 | 72.61 | 0.57 | 8.22 | 0.0 | 0.0 | 1    |
| 4 | 1.51742 | 13.27 | 3.62 | 1.24 | 73.08 | 0.55 | 8.07 | 0.0 | 0.0 | 1    |

**Explanation:**

Here in the code, I have used ***read\_csv()*** and ***head()*** functions to read CSV file and to display the first 5 rows of the *DataFrame* respectively.

**Question 1 (b):**

Use *train\_test\_split* to create training and testing part.

***Source code:***

```
x1 = glass_df.iloc[:, :-1].values
y1 = glass_df['Type'].values

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test=train_test_split(x1, y1, test_size = 0.30, random_state = 0)
```

***Explanation:***

Here in the code, I have used *train\_test\_split( )* is used to split the dataset in to training and testing sets. The ***test\_size*** parameter specifies the percentage of the dataset to be used for testing, and the ***random\_state*** parameter is used to ensure reproducibility of the split.

**Question 2:**

Evaluate the model on testing part using score and

`classification_report(y_true, y_pred)`

***Source code:***

```
# GaussianNB()- function creates an instance of the Gaussian Naive Bayes classifier
classifier1 = GaussianNB()

# Training the classifier using the training data
classifier1.fit(x_train, y_train)

# Making predictions on the testing data and storing in y_pred
y_pred = classifier1.predict(x_test)

# generates a confusion matrix that summarizes the number of TP, FP, TN, FN for each class
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))

# accuracy of the classifier by comparing features (x_test) with true labels (y_test)
print("\n\t Accuracy by score:", classifier1.score(x_test, y_test))

# accuracy of the classifier by comparing the predicted values (y_pred) with true values (y_test)
print('\nAccuracy by accuracy_score:', accuracy_score(y_pred, y_test))

'''generating a summary of the predictions made by the classifier, including precision,
recall, and F1-score for each class, as well as an overall accuracy score. '''
print('\n', classification_report(y_test, y_pred))
```

***Output:***

```
Confusion Matrix:
[[18  1  0  0  1  1]
 [21  3  1  1  0  0]
 [ 7  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  0  2  0]
 [ 0  0  0  0  0  7]]

      Accuracy by score: 0.46153846153846156

Accuracy by accuracy_score: 0.46153846153846156
```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1            | 0.39      | 0.86   | 0.54     | 21      |
| 2            | 0.50      | 0.12   | 0.19     | 26      |
| 3            | 0.00      | 0.00   | 0.00     | 7       |
| 5            | 0.00      | 0.00   | 0.00     | 2       |
| 6            | 0.67      | 1.00   | 0.80     | 2       |
| 7            | 0.88      | 1.00   | 0.93     | 7       |
| accuracy     |           |        | 0.46     | 65      |
| macro avg    | 0.41      | 0.50   | 0.41     | 65      |
| weighted avg | 0.44      | 0.46   | 0.37     | 65      |

***Explanation:***

Here in the code, I have used ***GaussianNB()*** function to create Gaussian Naive Bayes classifier. The classifier is then trained on the training data using the ***fit()*** function. Predictions are made on the testing data using the ***predict()*** function, and the predicted labels are stored in ***y\_pred***. The results are then printed, including the confusion matrix using ***confusion\_matrix()*** function, accuracy scores using ***score()*** and ***accuracy\_score()*** functions, and classification report using ***classification\_report()*** functions.

**Question 1:**

Implement **linear SVM method** using scikit library.

- a. Use the glass dataset available in Link also provided in your assignment.

***Source code:***

```
# Loading the dataset
glass_df = pd.read_csv("glass.csv")
glass_df.head()
```

***Output:***

|   | 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 | 0.0 | 1    |
| 1 | 1.51761 | 13.89 | 3.60 | 1.36 | 72.73 | 0.48 | 7.83 | 0.0 | 0.0 | 1    |
| 2 | 1.51618 | 13.53 | 3.55 | 1.54 | 72.99 | 0.39 | 7.78 | 0.0 | 0.0 | 1    |
| 3 | 1.51766 | 13.21 | 3.69 | 1.29 | 72.61 | 0.57 | 8.22 | 0.0 | 0.0 | 1    |
| 4 | 1.51742 | 13.27 | 3.62 | 1.24 | 73.08 | 0.55 | 8.07 | 0.0 | 0.0 | 1    |

***Explanation:***

Here in the code, I have used ***read\_csv()*** and ***head()*** functions to read CSV file and to display the first 5 rows of the *DataFrame* respectively.

### **Question 1 (b):**

Use `train_test_split` to create training and testing part.

**Source code:**

```
x1 = glass_df.iloc[:, :-1].values
y1 = glass_df['Type'].values

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test=train_test_split(x1, y1, test_size = 0.30, random_state = 0)
```

**Explanation:**

Here in the code, I have used `train_test_split( )` is used to split the dataset in to training and testing sets. The ***test\_size*** parameter specifies the percentage of the dataset to be used for testing, and the ***random\_state*** parameter is used to ensure reproducibility of the split.

### **Question 2:**

Evaluate the model on testing part using score and

`classification_report(y_true, y_pred)`

**Source code:**

```
# LinearSVC( ) - function creates an instance of the Linear support vector classifier
classifier2 = LinearSVC(random_state = 1)

# Training the classifier using the training data
classifier2.fit(x_train, y_train)

# Making predictions on the testing data and storing in y_pred
y_pred2 = classifier2.predict(x_test)

# generates a confusion matrix that summarizes the number of TP, FP, TN, FN for each class
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred2))

# accuracy of the classifier by comparing features (x_test) with true labels (y_test)
print("\n\t Accuracy by score:", classifier2.score(x_test, y_test))

# accuracy of the classifier by comparing the predicted values (y_pred) with true values (y_test)
print('\n Accuracy by accuracy_score:', accuracy_score(y_pred2, y_test))

'''generating a summary of the predictions made by the classifier, including precision,
recall, and F1-score for each class, as well as an overall accuracy score. '''
print('\n', classification_report(y_test, y_pred2))
```

**Output:**

```
Confusion Matrix:
[[10 11  0  0  0  0]
 [ 8 18  0  0  0  0]
 [ 2  5  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  6  0  0  0  1]]

Accuracy by score: 0.4461538461538462
Accuracy by accuracy_score: 0.4461538461538462
```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1            | 0.39      | 0.86   | 0.54     | 21      |
| 2            | 0.50      | 0.12   | 0.19     | 26      |
| 3            | 0.00      | 0.00   | 0.00     | 7       |
| 5            | 0.00      | 0.00   | 0.00     | 2       |
| 6            | 0.67      | 1.00   | 0.80     | 2       |
| 7            | 0.88      | 1.00   | 0.93     | 7       |
| accuracy     |           |        | 0.46     | 65      |
| macro avg    | 0.41      | 0.50   | 0.41     | 65      |
| weighted avg | 0.44      | 0.46   | 0.37     | 65      |

**Explanation:**

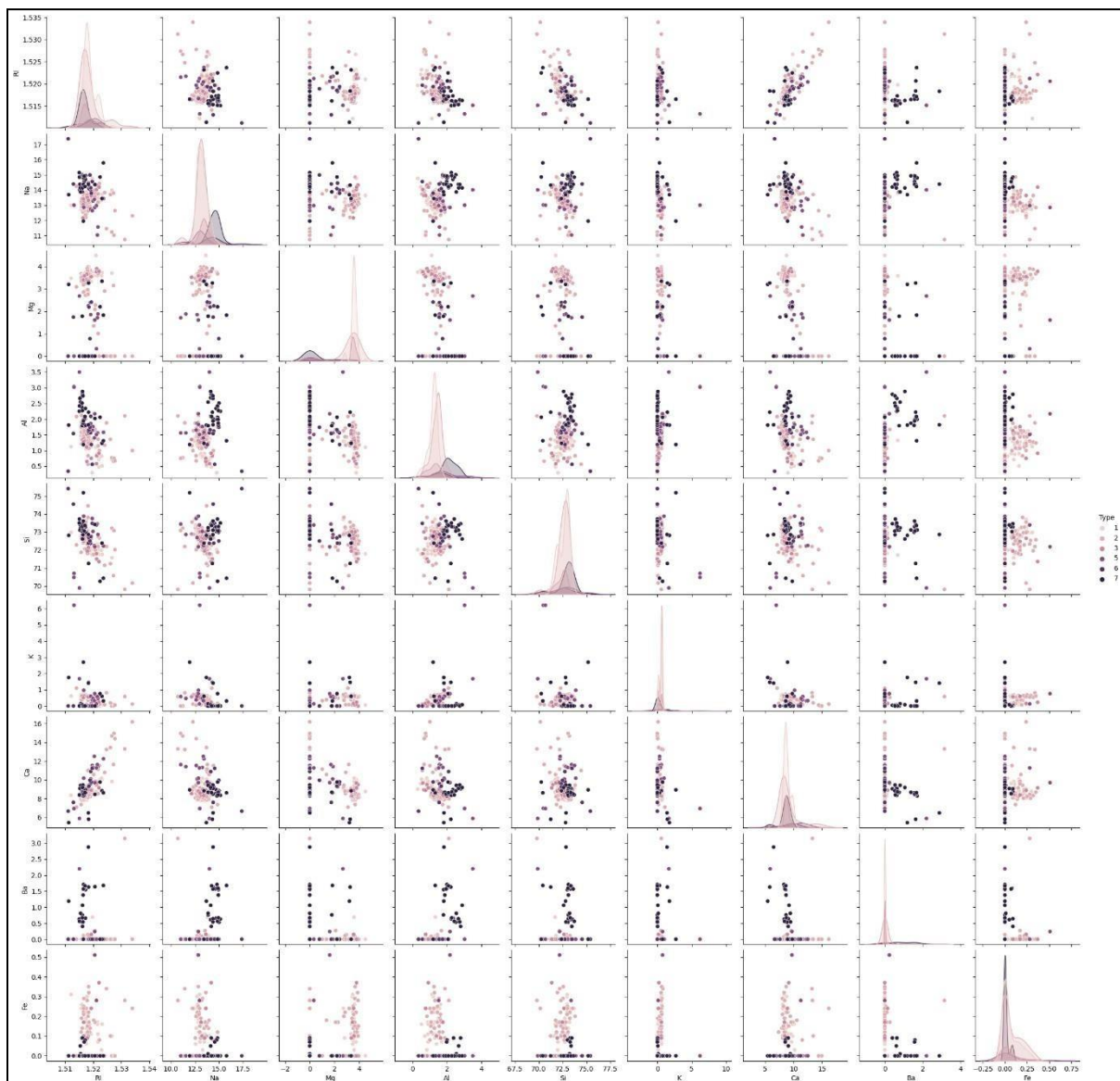
Here in the code, I have used ***LinearSVC()*** function to create Linear Support Vector Classifier. The classifier is then trained on the training data using the ***fit()*** function. Predictions are made on the testing data using the ***predict()*** function, and the predicted labels are stored in ***y\_pred2***. The results are then printed, including the confusion matrix using ***confusion\_matrix()*** function, accuracy scores using ***score()*** and ***accuracy\_score()*** functions, and classification report using ***classification\_report()*** functions.

**Question:**

Do at least two visualizations to describe or show correlations in the Glass Dataset.

**Visualization 1: Scatter Plot Matrix****Source code:**

```
# scatter plot matrix - shows the pairwise scatter plots of multiple variables in a dataset
# as the hue parameter is set to 'Type', it colors the scatter plot points based on the 'Type' column
sns.pairplot(glass_df, hue='Type')
```

**Output:****Explanation:**

Here in the code, I have used ***pairplot()*** function to generate scatterplots of all pairs. The ***hue*** parameter is set to ***'Type'***, it colors the scatter plot points based on the ***'Type'*** column.



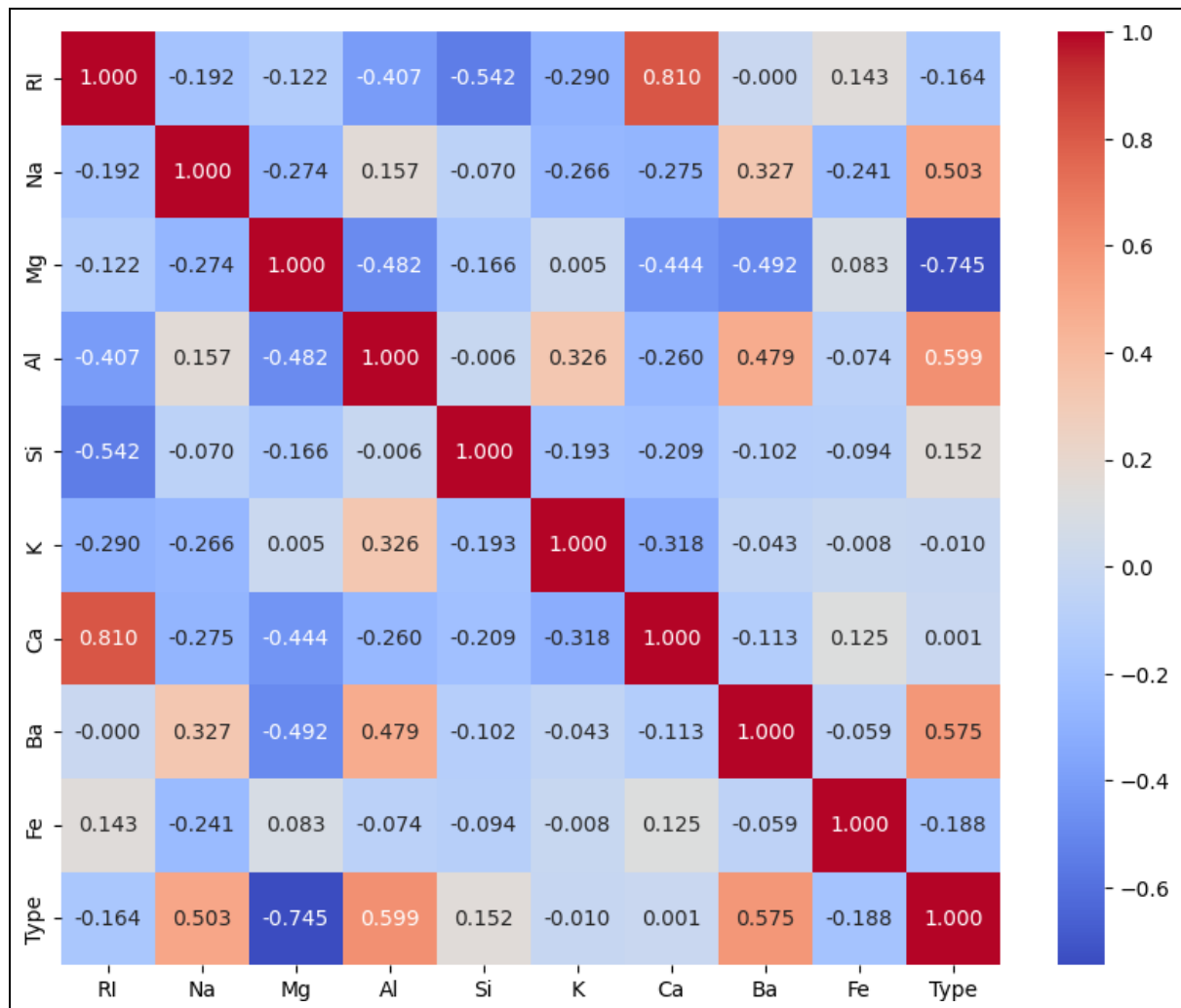
## Visualization 2: Heatmap

### Source code:

```
plt.figure(figsize=(10, 8))
corr = glass_df.corr()

#Heatmap: shows the correlation between pairs of variables, with a color scale indicating the strength of the relationship
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.3f')
plt.show()
```

### Output:



### Explanation:

Here in the code, I have used **corr()** function to compute the pairwise correlations between all columns and **heatmap()** function from the **seaborn** library to plot the correlations with a color scale.

The **figsize** parameter of the **figure()** function sets the size of the figure in inches. The **annot** parameter of the **heatmap()** function specifies that the numerical correlation values

should be annotated on the plot. The ***cmap*** parameter sets the color map to use for the heatmap, and the ***fmt*** parameter sets the format string for the annotation values to three decimal places.

***plt. show()*** function is used to display the plot.

### **Question:**

Which algorithm you got better accuracy? Can you justify why?

### **Answer:**

The accuracy scores for both classifiers are very low. Considering the accuracy scores, Naïve Bayes classifier performed slightly better than the Linear SVC classifier.

### **Justification:**

Naive Bayes analysis works well with probabilistic concepts whereas Linear SVM works better with linear regression logics. But to perform more accurately, SVM requires large amounts of data to train and test the data.

So, based on the amount of data given, Naive Bayes algorithm gives better accuracy compared to Linear SVM.