

# Machine Learning – Assignment 4

CS 5710 (CRN 22002)

Student ID: 700745451

Student Name: Kamala Ramesh

## Question 1:

- Import the pandas libraries. Read the csv file containing the data sets and display the basic statistical descriptions of the dataset.

```
✓ [109] from google.colab import drive  
1s drive.mount('/content/gdrive')
```

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force\_remount=True).

```
✓ [110] import pandas as pd  
2s import matplotlib.pyplot as plt
```



```
#read the csv file and assign it to a variable  
dataset = pd.read_csv('/content/gdrive/My Drive/data.csv')  
df = pd.DataFrame(dataset)  
  
#display the basic statistical description of the data  
df.describe()
```

	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



- Get the rows that has null values in any of their column values, copy their indexes into a list. Here we are storing the indexes just to see how our data looks after updating the null values. Update all the null values with the respective mean value. Now using the row indexes we can see how the rows look after updating.

```
▶ #find the rows that has null values
nullVal = pd.DataFrame(df[df.isna().any(axis=1)])
print("Rows that has null values:")
print(nullVal)

#store the rows indexes in a list
nullValInx = list(nullVal.index.values)

#replace the null values with the respective mean value of the column
df = df.fillna(round(df.mean(),1))

#display the updated row[s]
upd_val = pd.DataFrame(df,index=nullValInx)
print("\nRows that had null values, after update:")
upd_val
```

Rows that has null values:

	Duration	Pulse	Maxpulse	Calories
17	45	90	112	NaN
27	60	103	132	NaN
91	45	107	137	NaN
118	60	105	125	NaN
141	60	97	127	NaN

Rows that had null values, after update:

	Duration	Pulse	Maxpulse	Calories
17	45	90	112	375.8
27	60	103	132	375.8
91	45	107	137	375.8
118	60	105	125	375.8
141	60	97	127	375.8



- Here two columns, Pulse and Calories are selected and their respective max value, min value, count and mean are aggregated and displayed

```
✓ [113] #aggregated data of coulmns, Pulse and Calories
0s df.agg({'Pulse' : ['max', 'min', 'count', 'mean'], 'Calories' : ['max', 'min', 'count', 'mean']})
```

	Pulse	Calories
max	159.000000	1860.400000
min	80.000000	50.300000
count	169.000000	169.000000
mean	107.461538	375.790533

- Here we are displaying the rows whose Calories column values are between 500 and 1000. This is done in two steps. First we are filtering values greater than 500 and store it. Then filter the resulted data with values less than 1000.

```
✓ #to filter values between 500 and 1000 calories
0s df_great_500 = df[df['Calories']>=500] #filter rows with calories above 500
df_filter = df_great_500[df_great_500["Calories"]<=1000] #from the above result, filter the rows with calories below 1000
df_filter
```

	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

- Here we are displaying the rows whose Calories column values are greater than 500 and Pulse column values are less than 100. This is done in two steps. First, we are filtering Calories values greater than 500 and store it. Then filter the resulted data with Pulse values less than 1000.

```
df_great_500 = df[df['Calories']>500] #filter rows with calories above 500
df_pulse_100 = df_great_500[df_great_500["Pulse"]<100] #from the above result, filter the rows with pulse below 100
df_pulse_100
```

	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

- Creating a new data frame containing the all the columns except Maxpulse



```
#create a new dataframe containg all columns except Maxpulse
df_modified = df[["Duration","Pulse","Calories"]]
df_modified
```

	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
...	...	...	...
164	60	105	290.8
165	60	110	300.0
166	60	115	310.2
167	75	120	320.4
168	75	125	330.4

169 rows × 3 columns

- Delete the Maxpulse column from the main data frame.

```
#delete the Maxpulse column from the main frame
del df["Maxpulse"]
df
```





	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
...	...	...	...
164	60	105	290.8
165	60	110	300.0
166	60	115	310.2
167	75	120	320.4
168	75	125	330.4

169 rows × 3 columns

- Converting the datatype of Calories column from float to int.

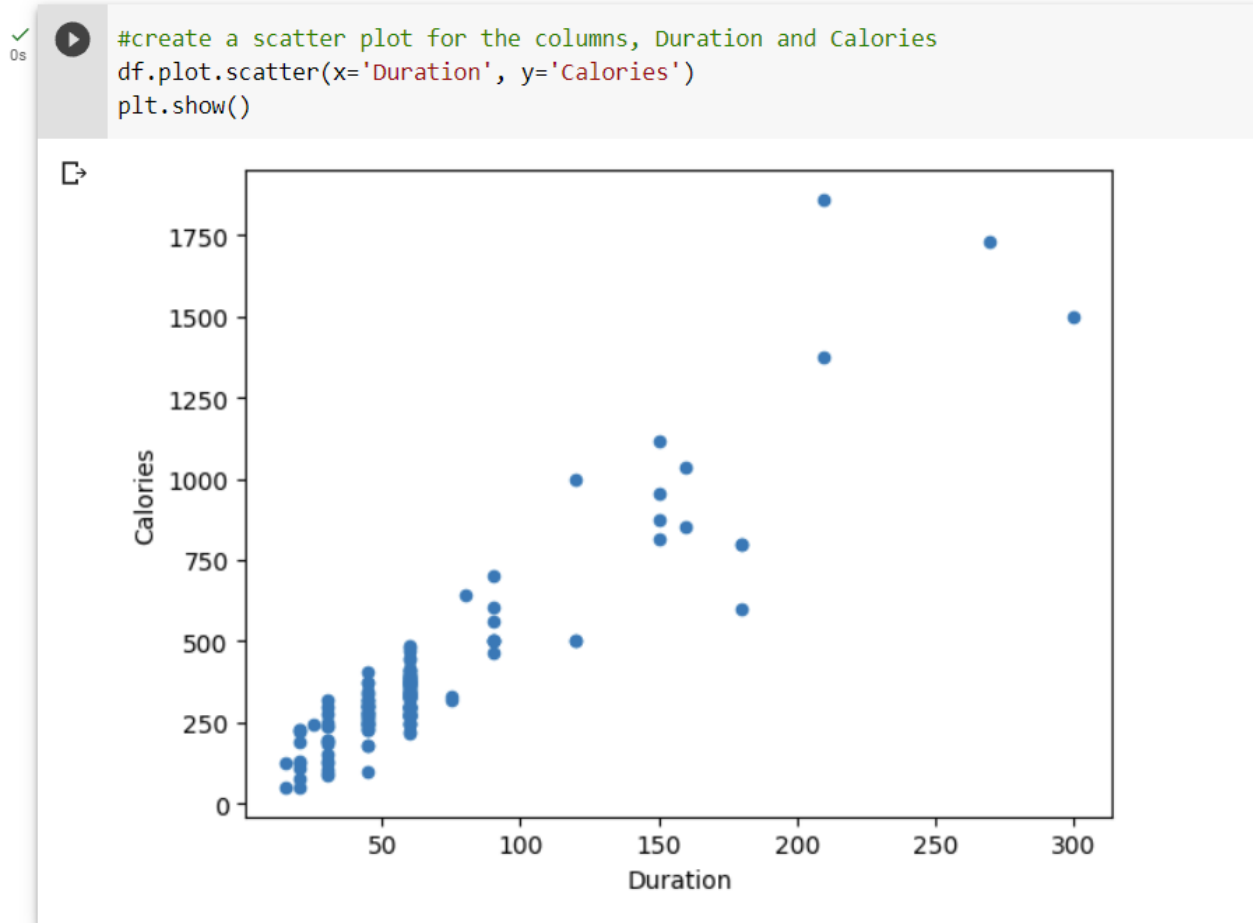
```
✓ 3s #convert the datatype of Calories column to int
df["Calories"] = df["Calories"].astype(int)
df
```



	Duration	Pulse	Calories
0	60	110	409
1	60	117	479
2	60	103	340
3	45	109	282
4	45	117	406
...	...	...	...
164	60	105	290
165	60	110	300
166	60	115	310
167	75	120	320
168	75	125	330

169 rows × 3 columns

- Creating a scatter plot for Duration and Calories.



## Question 2:

Import the libraries and read the titanic dataset

```
[1] from google.colab import drive
drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
#import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[21] #read dataset
titanic_df = pd.read_csv('/content/gdrive/My Drive/titanic_train.csv')
titanic_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

## Correlation Map of the given dataset

```
#correlation matrix of given data
titanic_df.corr()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

Encode the values in Column from str to int to calculate the correlation value with the target column.

```
[73] #encode columns with type str to type int
from sklearn.preprocessing import LabelEncoder

#Since correlation cannot be calculated when the column values are str type, we encode them with numbers
lb_make = LabelEncoder()
titanic_df["Sex_Encoded"] = lb_make.fit_transform(titanic_df["Sex"])
titanic_df["Sex_Encoded"].value_counts()

1    577
0    314
Name: Sex_Encoded, dtype: int64
```

```
[72] #Calculate Correlation between Survived and Encoded values of Column Sex
corr_Val = titanic_df["Survived"].corr(titanic_df["Sex_Encoded"])
print("Correlation between Survived and Sex Couolumn:",corr_Val)
```

Correlation between Survived and Sex Couolumn: -0.5433513806577555

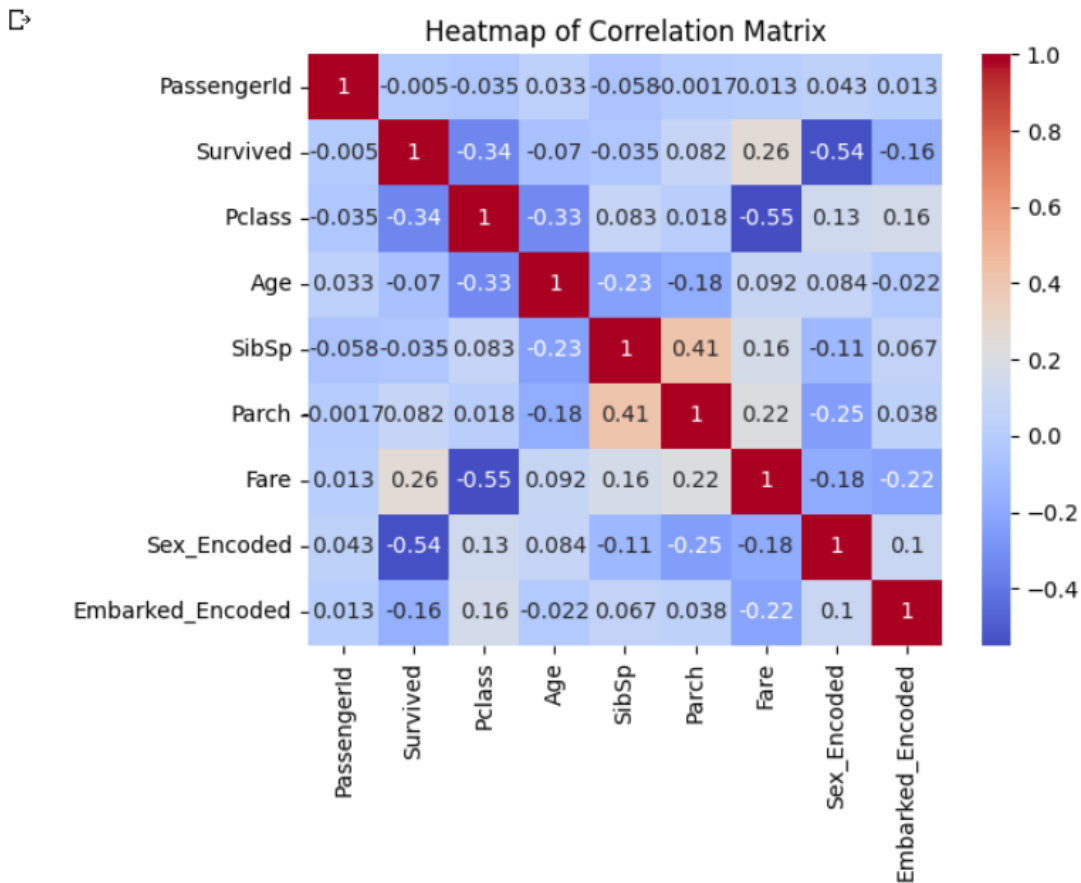
## Correlation after adding the Encoded Column

```
titanic_df.corr()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_Encoded	Embarked_Encoded
PassengerId	1.000000	-0.005007	-0.035144	0.033207	-0.057527	-0.001652	0.012658	0.042939	0.013083
Survived	-0.005007	1.000000	-0.338481	-0.069809	-0.035322	0.081629	0.257307	-0.543351	-0.163517
Pclass	-0.035144	-0.338481	1.000000	-0.331339	0.083081	0.018443	-0.549500	0.131900	0.157112
Age	0.033207	-0.069809	-0.331339	1.000000	-0.232625	-0.179191	0.091566	0.084153	-0.022239
SibSp	-0.057527	-0.035322	0.083081	-0.232625	1.000000	0.414838	0.159651	-0.114631	0.066654
Parch	-0.001652	0.081629	0.018443	-0.179191	0.414838	1.000000	0.216225	-0.245489	0.038322
Fare	0.012658	0.257307	-0.549500	0.091566	0.159651	0.216225	1.000000	-0.182333	-0.221226
Sex_Encoded	0.042939	-0.543351	0.131900	0.084153	-0.114631	-0.245489	-0.182333	1.000000	0.104057
Embarked_Encoded	0.013083	-0.163517	0.157112	-0.022239	0.066654	0.038322	-0.221226	0.104057	1.000000

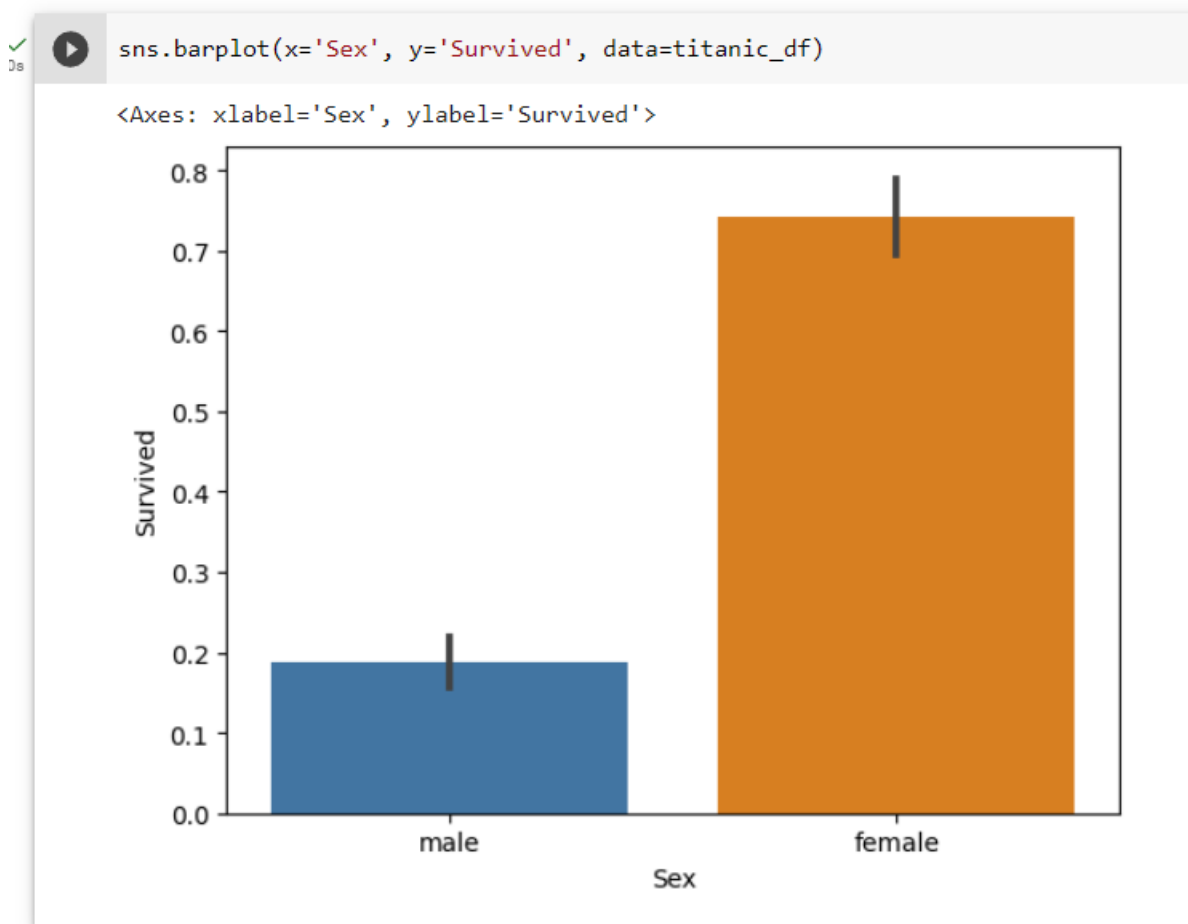
## HeatMap of Correlation

```
sns.heatmap(titanic_df.corr(), cmap='coolwarm', annot=True)
plt.title('Heatmap of Correlation Matrix')
plt.show()
```





Answer for a: As the survival rate of women is more compared to men, and more men died compared to women. Hence, we can keep this feature.



```
[126] #Find how many men and women survived and died
a = titanic_df[titanic_df['Survived'] == 0]
m0 = a[a['Sex'] == "male"]
f0 = a[a['Sex'] == "female"]
print("No. of Male Died:",m0["Sex"].count())
print("No. of Female Died:",f0["Sex"].count())

a = titanic_df[titanic_df['Survived'] == 1]
m1 = a[a['Sex'] == "male"]
f1 = a[a['Sex'] == "female"]
print("\nNo. of Male Survived:",m1["Sex"].count())
print("No. of Female Survived:",f1["Sex"].count())
```

No. of Male Died: 468  
No. of Female Died: 81

No. of Male Survived: 109  
No. of Female Survived: 233

Find the null values and fill the required columns

```
[80] #Since model cannot be trained when the column values are str type, we encode them with type int
lb_make = LabelEncoder()
titanic_df["Embarked_Encoded"] = lb_make.fit_transform(titanic_df["Embarked"])
titanic_df["Embarked_Encoded"].value_counts()

2    644
0    168
1     77
3      2
Name: Embarked_Encoded, dtype: int64
```

```
▶ #checking for null values
titanic_df.isnull().any()
```

```
PassengerId    False
Survived        False
Pclass          False
Name            False
Sex             False
Age            False
SibSp           False
Parch           False
Ticket         False
Fare           False
Cabin           True
Embarked        True
Sex_Encoded     False
Embarked_Encoded False
dtype: bool
```

```
[82] #filling null values with mean
titanic_df['Age'] = titanic_df['Age'].fillna(titanic_df['Age'].mean())
```

```
▶ #dropping Name, Ticket,Cabin and Passenger ID as they more unique values which might not be necessary
#Columns Sex and Embarked are also dropped as we have the encoded columns
X = titanic_df.drop(['Survived','Name','Sex','Ticket','Cabin','Embarked','PassengerId'], axis=1)
Y = titanic_df['Survived']

#splitting the dataset into training set and testing set
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size=0.25,random_state = 0)
```

```
▶ from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score

#instantiating the Naive Bayes model and fitting it with training set
gnb = GaussianNB()
gnb.fit(X_Train,Y_Train)

# Predicting the Test set result
Y_Pred = gnb.predict(X_Test)

#evaluating the model
print("Gaussian Naive Bayes Accuracy is:",round(accuracy_score(Y_Test,Y_Pred) * 100,2))
print("\nClassification Report:\n\n",metrics.classification_report(Y_Test,Y_Pred,zero_division=0))
```

🔗 Gaussian Naïve Bayers Accuracy is: 78.03

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.78	0.82	139
1	0.68	0.77	0.73	84
accuracy			0.78	223
macro avg	0.77	0.78	0.77	223
weighted avg	0.79	0.78	0.78	223

	precision	recall	f1-score	support
0	0.85	0.78	0.82	139
1	0.68	0.77	0.73	84
accuracy			0.78	223
macro avg	0.77	0.78	0.77	223
weighted avg	0.79	0.78	0.78	223

### Question 3:

To implement the Naïve Bayes method for the give dataset, glass.csv. Split the given dataset into trainingset and testing set. Train the model with the training data set. Predict the model for the test input and evaluate it.

```
✓ 23s [2] from google.colab import drive
drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
▶ #import libraries
import numpy as np
import random as rnd
import pandas as pd

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics

from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
```

```

✓ [7] # reading the dataset file
0s df = pd.read_csv('/content/gdrive/My Drive/glass.csv')

X = df.drop(['Type'], axis=1)
Y = df["Type"]

#splitting the dataset into training set and testing set
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size=0.25, random_state = 0)

```

```

✓ #instantiating the Naive Bayes model and fitting it with training set
0s gnb = GaussianNB()
gnb.fit(X_Train, Y_Train)

# Predicting the Test set result
Y_Pred = gnb.predict(X_Test)

#evaluating the model
print("Gaussian Naive Bayes Accuracy is:", round(accuracy_score(Y_Test, Y_Pred) * 100, 2))
print("\nClassification Report:\n\n", metrics.classification_report(Y_Test, Y_Pred, zero_division=0))

```

Gaussian Naive Bayes Accuracy is: 46.3

Classification Report:

	precision	recall	f1-score	support
1	0.32	0.64	0.43	14
2	0.45	0.21	0.29	24
3	0.50	0.40	0.44	5
5	0.00	0.00	0.00	2
6	0.67	1.00	0.80	2
7	1.00	1.00	1.00	7
accuracy			0.46	54
macro avg	0.49	0.54	0.49	54
weighted avg	0.49	0.46	0.44	54

To implement the linear SVM method for the same dataset. Predicting the outputs for the test set and evaluating the model.

```
#instantiating the linear SVM model and fitting it with training set
svc = SVC(kernel='linear')
svc.fit(X_Train, Y_Train)

# Predicting the Test set result
Y_pred = svc.predict(X_Test)

#evaluating the model
print("SVM accuracy is:", round(accuracy_score(Y_Test,Y_pred) * 100, 2))
print("\nClassification Report:\n\n",metrics.classification_report(Y_Test,Y_pred,zero_division=0))
```

➤ SVM accuracy is: 55.56

Classification Report:

	precision	recall	f1-score	support
1	0.43	0.86	0.57	14
2	0.60	0.38	0.46	24
3	0.00	0.00	0.00	5
5	0.67	1.00	0.80	2
6	0.00	0.00	0.00	2
7	1.00	1.00	1.00	7
accuracy			0.56	54
macro avg	0.45	0.54	0.47	54
weighted avg	0.53	0.56	0.51	54

## Correlation matrix of Glass Dataset

✓ 0s df.corr()

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
RI	1.000000	-0.191885	-0.122274	-0.407326	-0.542052	-0.289833	0.810403	-0.000386	0.143010	-0.164237
Na	-0.191885	1.000000	-0.273732	0.156794	-0.069809	-0.266087	-0.275442	0.326603	-0.241346	0.502898
Mg	-0.122274	-0.273732	1.000000	-0.481799	-0.165927	0.005396	-0.443750	-0.492262	0.083060	-0.744993
Al	-0.407326	0.156794	-0.481799	1.000000	-0.005524	0.325958	-0.259592	0.479404	-0.074402	0.598829
Si	-0.542052	-0.069809	-0.165927	-0.005524	1.000000	-0.193331	-0.208732	-0.102151	-0.094201	0.151565
K	-0.289833	-0.266087	0.005396	0.325958	-0.193331	1.000000	-0.317836	-0.042618	-0.007719	-0.010054
Ca	0.810403	-0.275442	-0.443750	-0.259592	-0.208732	-0.317836	1.000000	-0.112841	0.124968	0.000952
Ba	-0.000386	0.326603	-0.492262	0.479404	-0.102151	-0.042618	-0.112841	1.000000	-0.058692	0.575161
Fe	0.143010	-0.241346	0.083060	-0.074402	-0.094201	-0.007719	0.124968	-0.058692	1.000000	-0.188278
Type	-0.164237	0.502898	-0.744993	0.598829	0.151565	-0.010054	0.000952	0.575161	-0.188278	1.000000

## HeatMap of Correlation

✓  
1s

```
sns.heatmap(df.corr(), annot=True)  
plt.title('Heatmap of Correlation Matrix')  
plt.show()
```

