IDS PROJECT REPORT
Breast Cancer Detection

Prepared By:

Samudro Sinha - 21ucs180
Priyanshu Jain - 21ucs162
Prianshu Shaw - 21ucs161
Hemant Rajpurohit - 21ucs094

November, 2023

Contents

Aim

Performing data preprocessing and preliminary analysis on Breast Cancer Diagnostic Dataset and getting inferences from the data using Machine Learning Classification techniques - whether or not a given tumor is malignant or benign.

Introduction

Breast cancer is one among the foremost common cancers among women worldwide, representing the bulk of latest cancer cases and cancer-related deaths consistent with global statistics, making it a big public health concern in today's society. The early diagnosis of Breast Cancer can improve the prognosis and chance of survival significantly, because it can promote timely clinical treatment to patients. Accurate classification of benign tumors can prevent patients from undergoing unnecessary treatments. Thus, the right diagnosis of Breast Cancer and therefore the classification of patients into malignant or benign groups is such a crucial topic of dialogue. Machine Learning provides advantages in detecting critical features from complex carcinoma datasets proving to be an appropriate methodology of choice in carcinoma pattern classification and forecast modeling. We will be using Machine Learning to predict whether a tumor is Benign (non-cancerous) or Malign (cancerous). Using prediction models to classify cases of carcinoma is statistical in nature, during this project, we'll be using multiple classifiers for the prediction of a carcinoma diagnosis, within the preprocessing phase. We'll even be analyzing what percentage true positives and false positives we get and the way much improvement in accuracy are often gained by feature selection.

Requirements

Python Version : Python 3.10.12
Libraries and Dependencies : NumPy, Pandas, Matplot Library, Sci-kit learn, Date-time, seaborn.
Execution environment : Google Colaboratory

Specification of dataset

Breast Cancer Wisconsin (Diagnostic)
Donated on 10/31/1995

Diagnostic Wisconsin Breast Cancer Database.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Health and Medicine	Classification
Feature Type	**# Instances**	**# Features**
Real	569	30

The dataset was imported from the UCI Machine Learning repository, the dataset contains 30 features and 569 instances of different kinds of samples.

Features

"Features" refer to the input variables or attributes used to make predictions about the target variable, which is the class or category you are trying to assign to each instance. Features are the measurable properties or characteristics of the data that the machine learning model analyzes to learn patterns and make predictions.

The dataset consists of the following attributes/features:

1. id: ID Number
2. diagnosis: The diagnosis of breast tissues (M = malignant, B = benign)
3. radius mean: mean of distances from center to points on the perimeter
4. texture mean: standard deviation of gray-scale values
5. perimeter mean: mean size of the core tumor
6. area mean
7. smoothness mean: mean of local variation in radius lengths
8. compactness mean: mean of perimeter∧2 / area - 1.0
9. concavity mean: mean of severity of concave portions of the contour
10. concave points mean: mean for number of concave portions of the contour
11. symmetry mean
12. fractal dimension mean: mean for ”coastline approximation” - 1
13. radius se: standard error for the mean of distances from center to points on the perimeter
14. texture se: standard error for standard deviation of gray-scale values
15. perimeter se
16. area se
17. smoothness se: standard error for local variation in radius lengths
18. compactness se: standard error for perimeter∧2 / area - 1.0
19. concavity se: standard error for severity of concave portions of the contour
20. concave points se: standard error for number of concave portions of the contour

21. symmetry se
22. fractal dimension se: standard error for ”coastline approximation” - 1
23. radius worst: ”worst” or largest mean value for mean of distances from center to points on the perimeter
24. texture worst: ”worst” or largest mean value for standard deviation of gray-scale values
25. perimeter worst
26. area worst
27. smoothness worst: ”worst” or largest mean value for local variation in radius lengths.
28. compactness worst: ”worst” or largest mean value for perimeter$\wedge$2 / area - 1.0
29. concavity worst: ”worst” or largest mean value for severity of concave portions of the contour
30. concave points worst: ”worst” or largest mean value for number of concave portions of the contour
31. symmetry worst
32. fractal dimension worst: ”worst” or largest mean value for ”coastline approximation” - 1

Out of these attributes given the attribute “ID” was dropped because it was not relevant in predicting the presence of breast cancer.

The features not explained are self-explanatory.
‘se’ stands for standard error.

Data Preprocessing

1. Importing Libraries

- pandas: for manipulation and analysis of data.
- numpy: contains a large collection of high-level mathematical functions to operate on large, arrays and matrices with multiple dimensions.
- sklearn: for various algorithms (classification, regression and clustering) including SVM, gradient boost□ing, k-means, random forests, DBSCAN etc. as well as the different evaluation metrics.
- matplotlib: for embedding plots into applications using general-purpose GUI toolkits.
- seaborn: for drawing attractive and informative statistical graphics.
- statmodels: for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.
- warnings: to warn programmers about changes in language or library features in anticipation of backwards incompatible changes coming with Python 3.0.

```
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

2. Loading the Dataset

```
# URL of the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data"

df = pd.read_csv(url, header=None)
```

We load the ‘Breast Cancer Wisconsin - (Diagnostic) dataset from the uci repository. We use pd.read_csv() function which is a predefined function in Pandas Library.

3. Finding dimensions

```
df.shape
```

```
(569, 32)
```

Thus, we can see that the dataset has 32 columns (indicating 32 features) and 569 tuples(indicating 569 datapoints).

Using the head function, we display the top 5 records from the dataset.

```
[ ]  df.head()
```

The output of the following is as follows:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	31
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 32 columns

df.info() function is a built-in function in pandas which provides information regarding each of the column present in the dataset. We know about the data-type, null/non-null values related information.

4. Looking for missing/duplicate/unique values

We use three handy functions:

1. df.duplicated().sum(): Counts the number of duplicated rows in a DataFrame.
2. df.isna().sum(): Counts the number of missing (NaN) values in each column of a DataFrame.
3. df.nunique(): Returns the number of unique values in each column of a DataFrame.

```
# Check Duplication
df.duplicated().sum()
```

```
0
```

There aren't duplicate values

```
# check Missing value
df.isna().sum()
```

```
27      0
28      0
29      0
30      0
31      0
dtype: int64
```

- No Missing Value is avalible

```
# Check the number of unique values of each column
df.nunique()
```

```
27      529
28      539
29      492
30      500
31      535
dtype: int64
```

5. Dropping useless columns/features

As a part of data preprocessing, we need to drop/remove the features from the dataset that do not provide us any insights for the classification algorithms/ data analysis algorithms to work.

```
[ ]  df = df.drop([0], axis= 1)
```

After removing the useless column(id in this case) the dataset looks like this:

```
df.head()
```

	1	2	3	4	5	6	7	8	9	10	...	22	23	24	25	26	27	28	29	30	31
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 31 columns

6. Renaming columns/features/values as per our convenience + encoding

Here. we have renamed the class column as target.
Also, in the target column, we denote malignant by 1 and benign by 0.
This will enable us to deploy machine learning algorithms, calculating loss functions much easily.

```
df = df.rename(columns={1 : 'target'})
```

changed target data in the dataset. I changed malignant to 1 and benign to 0.

```
df.target.replace({'M' : '1','B': '0'},inplace=True)
df.head()
```

Also, we have changed the datatype of the target column to float, with the help of 'df.target = df.target.astype('float64')'.

Finally, our data preprocessing step comes to an end. We can use various functions like .describe(), .info(), .head() to go through the dataset on a superficial level as well as in depth.

This is what the final dataset looks like:

```
df.head()
```

	target	2	3	4	5	6	7	8	9	10	...	22	23	24	25	26	27	28	29	30	31
0	1.0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	1.0	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	1.0	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	1.0	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	1.0	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 31 columns

```
# Check statistic of dataset
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
target	569.0	0.372583	0.483918	0.000000	0.000000	0.000000	1.000000	1.00000
2	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
3	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
4	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
5	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
6	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
7	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
8	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
9	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120

Exploratory Data Analysis

1.Visualisation : Data visualization plays a pivotal role in conveying insights from the dataset. Data visualization is the presentation of data in a graphical or pictorial format to help people understand patterns, trends, and insights in the data more easily. Instead of relying solely on raw numbers or textual information, data visualization leverages visual elements such as charts, graphs, maps, and other visual representations to convey complex information in a more accessible and understandable way.

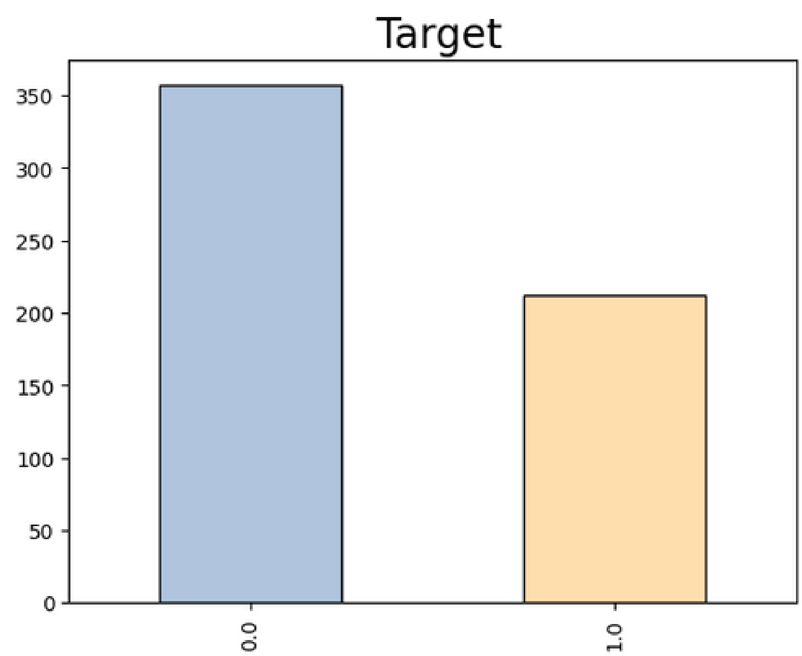
Numerical insights related to how many malignant and benign yields are there in the dataset.

```
df.target.value_counts()
```

```
0.0    357
1.0    212
Name: target, dtype: int64
```

Visualised target data in the dataset (we used bar chart)

```
df['target'].value_counts().plot(kind='bar',edgecolor='black',color=['lightsteelblue','navajowhite'])
plt.title(" Target",fontsize=20)
plt.show()
```



1-->Malignant
0-->Benign

2.Correlation Analysis: Correlation analysis is a statistical method used to evaluate the strength and direction of the linear relationship between two quantitative variables. The result of a correlation analysis is expressed as a correlation coefficient, which indicates the degree to which changes in one variable are associated with changes in another. The most common correlation coefficient is the Pearson correlation coefficient, denoted by *r*.
The Pearson correlation coefficient ranges from -1 to 1.

Correlation plays a significant role in feature selection as it helps identify relationships between different features in a dataset.

High correlation between features means they are more linearly dependent and hence have almost the same effect on the dependent variable. So, we can leave one of the two features when two they have high correlation.

Now, we generate the **Correlation matrix:**

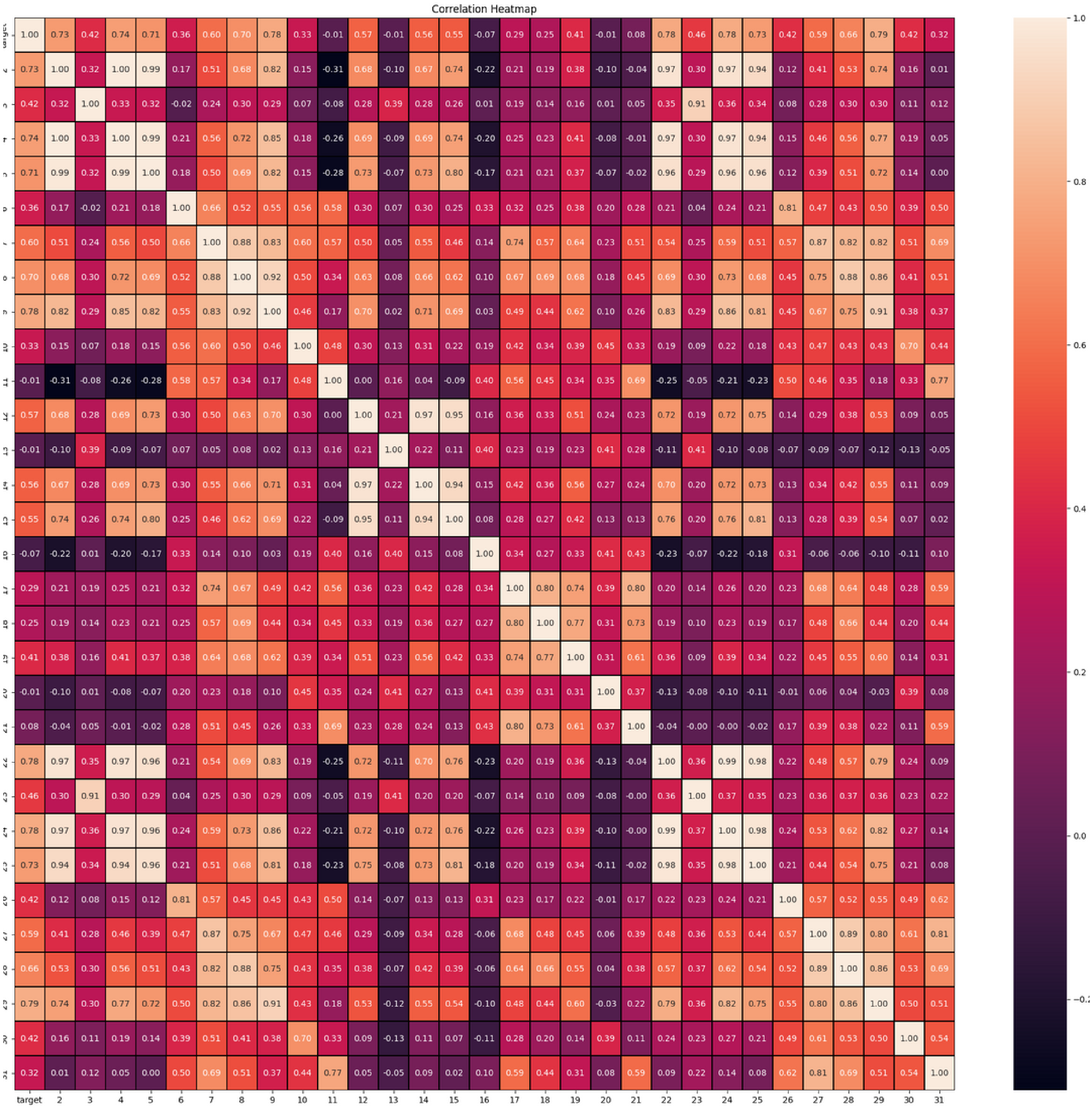
```
cor = df.corr()
```

	target	2	3	4	5	6	7	8	9	10	...	22	23	24	25	26	27	28	29	30	31
target	1.000000	0.730029	0.415185	0.742636	0.708984	0.358560	0.596534	0.696360	0.776614	0.330499	...	0.776454	0.456903	0.782914	0.733825	0.421465	0.590998	0.659610	0.793566	0.416294	0.323872
2	0.730029	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	...	0.969539	0.297008	0.965137	0.941082	0.119616	0.413463	0.526911	0.744214	0.163953	0.007066
3	0.415185	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	...	0.352573	0.912045	0.358040	0.343546	0.077503	0.277830	0.301025	0.295316	0.105008	0.119205
4	0.742636	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	...	0.969476	0.303038	0.970387	0.941550	0.150549	0.455774	0.563879	0.771241	0.189115	0.051019
5	0.708984	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	...	0.962746	0.287489	0.959120	0.959213	0.123523	0.390410	0.512606	0.722017	0.143570	0.003738
6	0.358560	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	...	0.213120	0.036072	0.238853	0.206718	0.805324	0.472468	0.434926	0.503053	0.394309	0.499316
7	0.596534	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	...	0.535315	0.248133	0.590210	0.509604	0.565541	0.865809	0.816275	0.815573	0.510223	0.687382
8	0.696360	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	...	0.688236	0.299879	0.729565	0.675987	0.448822	0.754968	0.884103	0.861323	0.409464	0.514930
9	0.776614	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	...	0.830318	0.292752	0.855923	0.809630	0.452753	0.667454	0.752399	0.910155	0.375744	0.368661
10	0.330499	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	...	0.185728	0.090651	0.219169	0.177193	0.426675	0.473200	0.433721	0.430297	0.699826	0.438413
11	-0.012838	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	...	-0.253691	-0.051269	-0.205151	-0.231854	0.504942	0.458798	0.346234	0.175325	0.334019	0.767297
12	0.567134	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050	0.303379	...	0.715065	0.194799	0.719684	0.751548	0.141919	0.287103	0.380585	0.531062	0.094543	0.049559
13	-0.008303	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480	0.128053	...	-0.111690	0.409003	-0.102242	-0.083195	-0.073658	-0.092439	-0.068956	-0.119638	-0.128215	-0.045655
14	0.556141	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650	0.313893	...	0.697201	0.200371	0.721031	0.730713	0.130054	0.341919	0.418899	0.554897	0.109930	0.085433
15	0.548236	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299	0.223970	...	0.757373	0.196497	0.761213	0.811408	0.125389	0.283257	0.385100	0.538166	0.074126	0.017539
16	-0.067016	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653	0.187321	...	-0.230691	-0.074743	-0.217304	-0.182195	0.314457	-0.055558	-0.058298	-0.102007	-0.107342	0.101480
17	0.292999	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424	0.421659	...	0.204607	0.143003	0.260516	0.199371	0.227394	0.678780	0.639147	0.483208	0.277878	0.590973
18	0.253730	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167	0.342627	...	0.186904	0.100241	0.226680	0.188353	0.168481	0.484858	0.662564	0.440472	0.197788	0.439329
19	0.408042	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634	0.393298	...	0.358127	0.086741	0.394999	0.342271	0.215351	0.452888	0.549592	0.602450	0.143116	0.310655
20	-0.006522	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351	0.449137	...	-0.128121	-0.077473	-0.103753	-0.110343	-0.012662	0.060255	0.037119	-0.030413	0.389402	0.078079
21	0.077972	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584	0.331786	...	-0.037488	-0.003195	-0.001000	-0.022736	0.170568	0.390159	0.379975	0.215204	0.111094	0.591328

31x31

Now, we generate the **Correlation heatmap:**

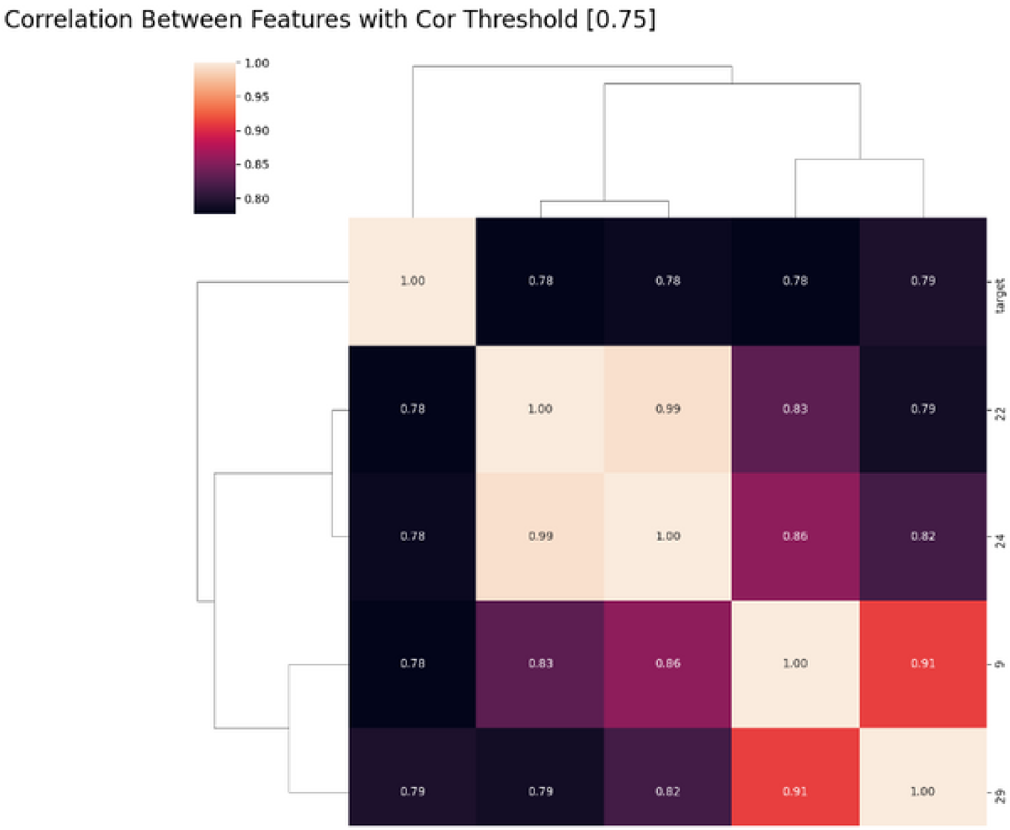
```
plt.figure(figsize=(25,23))
sns.heatmap(cor, annot= True, linewidths= 0.3 ,linecolor = "black", fmt = ".2f")
plt.title('Correlation Heatmap')
plt.show()
```



Thresholding:
Setting a correlation threshold, such as 0.75, in feature selection helps identify and eliminate highly uncorrelated features. This process reduces redundancy, improves model **interpretability**, and enhances **generalization performance**, preventing multicollinearity issues and ensuring that selected features contribute unique information to the model.

Data with a correlation greater than 0.75.

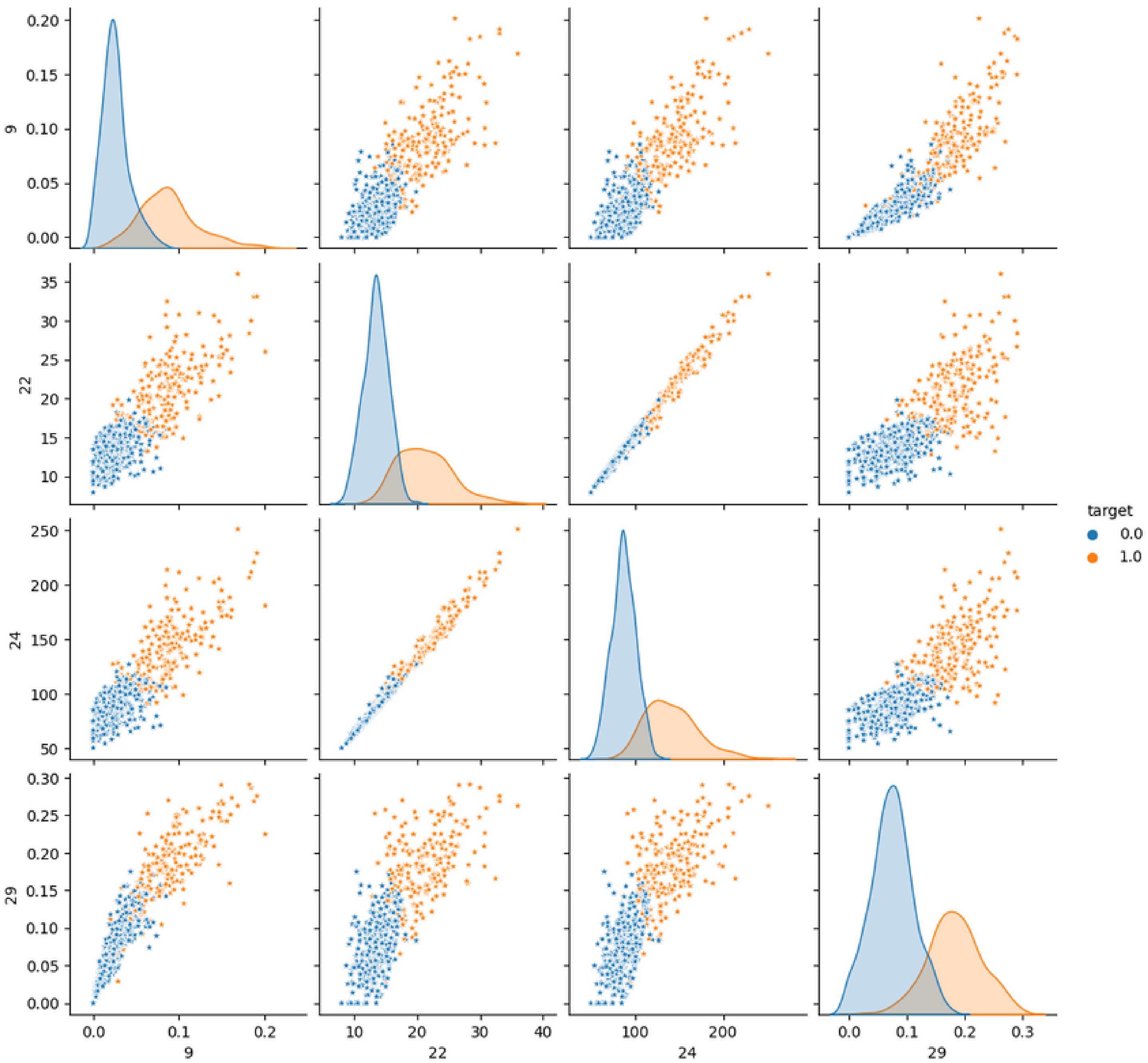
```
threshold = 0.75
filtre = np.abs(cor["target"] > threshold)
corr_features = cor.columns[filtre].tolist()
plt.figure(figsize=(10,8))
sns.clustermap(df[corr_features].corr(), annot = True, fmt = ".2f")
plt.title("\n                          Correlation Between Features with Cor Threshold [0.75]\n",fontsize=20)
plt.show()
```



visualized the data with a correlation greater than 0.75

3. Pairplot : Each scatterplot in the pair plot represents the relationship between two specific features, providing insights into potential patterns or distinctions between the two classes. The diagonal plots display kernel density estimates for individual features, showing their distributions.

```
[ ]  sns.pairplot(df[corr_features], diag_kind = "kde" , markers = "*", hue="target")
     plt.show()
```



Classification Algorithms

After carrying out basic data pre-processing and cleaning we can now move on to running the classifier models namely - K nearest neighbours, Random Forest, Decision Trees, Naive Bayes, Support Vector Machines and Logistic Regression. After the execution of all the above mentioned classifiers we can then compare the classifier evaluation metrics such as accuracy , precision, recall and F-scores to determine the best classification algorithm. Below is the step by step procedure so as to how we implemented the model:

Step 1 - Splitting and Standardising Data:
The code snippets provided below splits the dataset into features “x” and target variable “y”. We first create a new DataFrame x by dropping the column named 'target' along the columns (axis=1) from the original DataFrame df. Essentially, it extracts all the features except the target variable.
Then we move on to creating a series ‘y’ containing the values of the 'target' column from the original DataFrame df. We then move on to the train_test_split function for splitting data into training and testing sets.
Lastly, in the third snippet, we standardise the features in the training and testing sets using StandardScaler. This ensures that the features have a mean of 0 and a standard deviation of 1, which can be important for the classifiers that we are implementing.

```
# Splitting data
x= df.drop('target',axis=1)
y= df['target']
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.30,random_state=101)
```

```
[ ]  s= StandardScaler()
     x_train = s.fit_transform(x_train)
     x_test = s.fit_transform(x_test)
```

This code snippet creates a list called “algorithm” containing the names of various machine learning classifiers such as KNeighborsClassifier, RandomForestClassifier, DecisionTreeClassifier, GaussianNB, LogisticRegression, and SVC. Additionally, an empty list named “Accuracy” is initialised, intended to store accuracy values related to these classifiers.

```
algorithm = ['KNeighborsClassifier','RandomForestClassifier','DecisionTreeClassifier','GaussianNB','LogisticRegression', 'SVC' ]
Accuracy=[]
```

Step 2 - Model Evaluation:

This code snippet defines a function named “all” that takes a learning model as an argument. Inside the function, the model is trained on the training data (x_train and y_train), and predictions are made on the testing data (x_test). The accuracy of the model is then calculated and appended to a list named Accuracy.

Subsequently, the code generates and displays two confusion matrices: one without normalization and one with normalization. It also prints the confusion matrix, a classification report, and the accuracy score. This function is designed to provide a comprehensive evaluation of a machine learning model's performance on a given dataset.

The confusion matrix without normalisation provides raw counts of true positive, true negative, false positive, and false negative predictions, offering an absolute view of model performance. In contrast, the normalized confusion matrix presents these counts as proportions, offering a relative perspective by considering the distribution of true class instances. Both matrices contribute to a comprehensive evaluation of a machine learning model's classification accuracy and errors. We then use Matplot Library functions to print the various evaluation metrics which helps in better visualisation of the results.

```python
def all(model):
    model.fit(x_train,y_train)
    pred = model.predict(x_test)

    acc=accuracy_score(y_test,pred)
    Accuracy.append(acc)

    # confusion matrix without Normalization
    print('confusion matrix')
    # Calculate confusion matrix
    cm = confusion_matrix(y_test,pred)
    # Plot the confusion matrix
    sns.heatmap(cm, annot=True, fmt='d',cmap=['lightsteelblue','navajowhite'])
    plt.title('Confusion matrix')
    plt.xlabel('Predcted lablel')
    plt.ylabel('True lable')
    plt.show()

    # confusion matrix without Normalization
    print('Normalized confusion matrix')
    # Calculate confusion matrix
    cm1 = confusion_matrix(y_test,pred, normalize='true')
    # Plot the confusion matrix
    sns.heatmap(cm1, annot=True,cmap=['lightsteelblue','navajowhite'])
    plt.title('Normalized Confusion  matrix')
    plt.xlabel('Predcted lablel')
    plt.ylabel('True lable')
    plt.show()
```

```python
    # print Confusion matrix, Classification report and accuracy report
    print(cm)
    print(classification_report(y_test,pred))
    print('accuracy_score : ' , acc)
```

Hence the above snippets of code help us in setting up the dataset for classification algorithms to be executed on it by splitting it into training and testing splits. To ensure proper visualisation of the results we use several matplot library functions. We now can move on to individually running the classification algorithms as shown below.

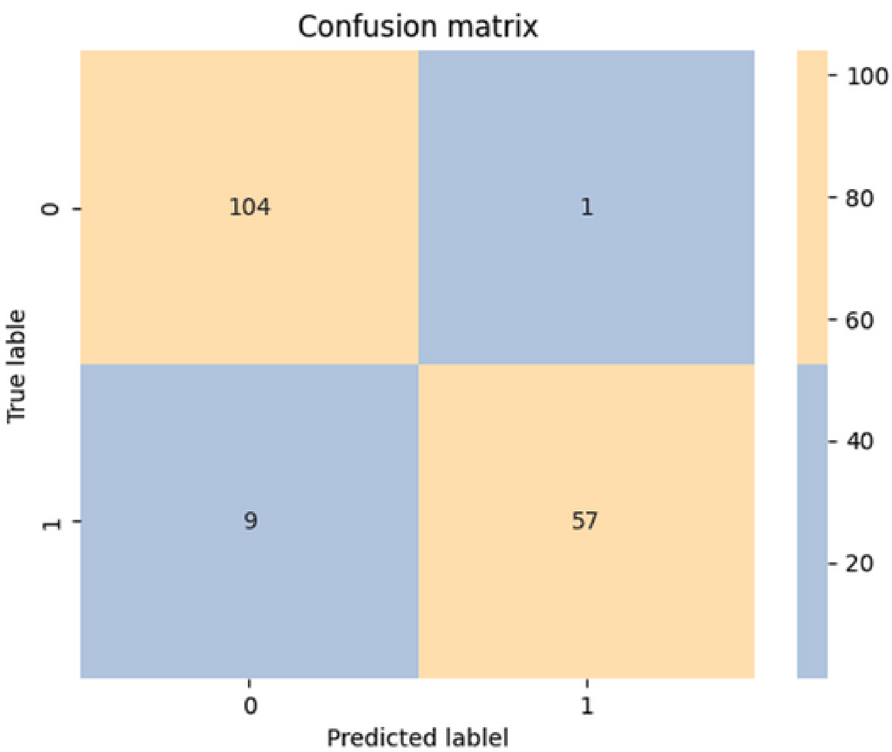
KNN

K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm used for classification and regression tasks. It relies on the principle of proximity, assigning a data point's label based on the majority class of its nearest neighbors. KNN is non-parametric and adaptable to various types of data.

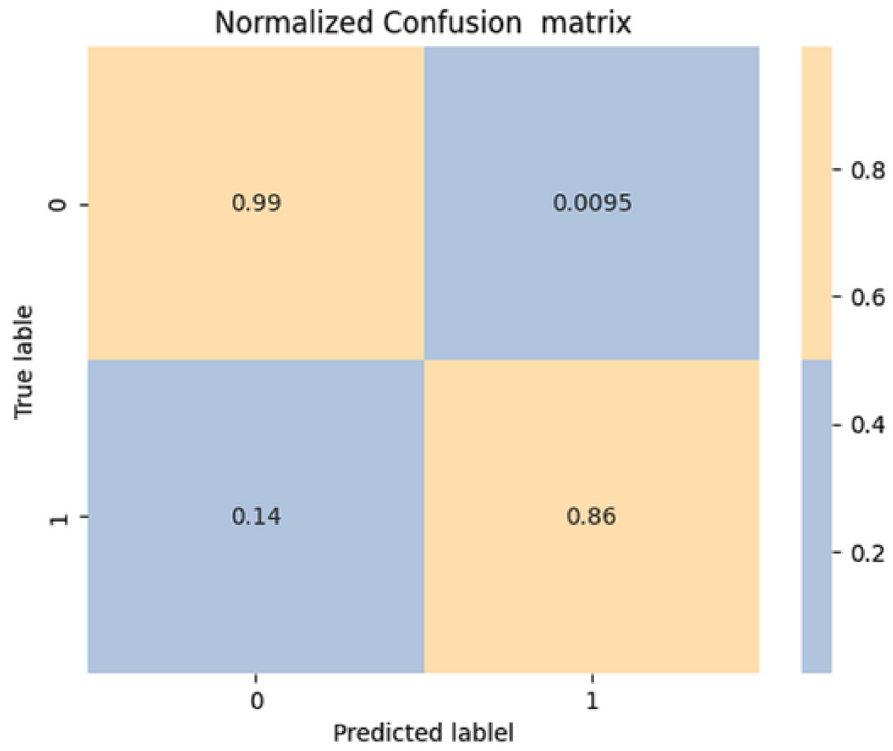
Implementation in Code :

```
[ ]  model_1 =KNeighborsClassifier(n_neighbors=2)
     all(model_1)
```

Confusion Matrix:



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[104   1]
 [  9  57]]
              precision    recall  f1-score   support

         0.0       0.92      0.99      0.95       105
         1.0       0.98      0.86      0.92        66

    accuracy                           0.94       171
   macro avg       0.95      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171

accuracy_score :  0.9415204678362573
```

RandomForest

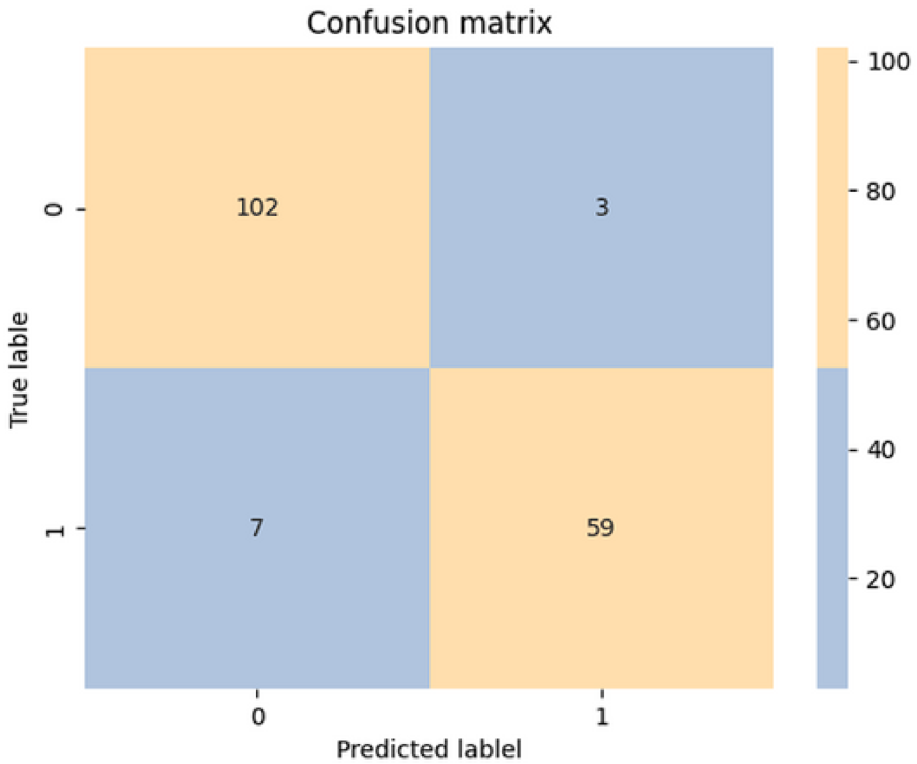
Random Forest is an ensemble learning algorithm widely used for classification and regression. It constructs multiple decision trees during training and merges their outputs to enhance accuracy and mitigate overfitting. By introducing randomness in tree building, Random Forest improves robustness and generalization, making it a popular choice in machine learning

Implementation in Code :

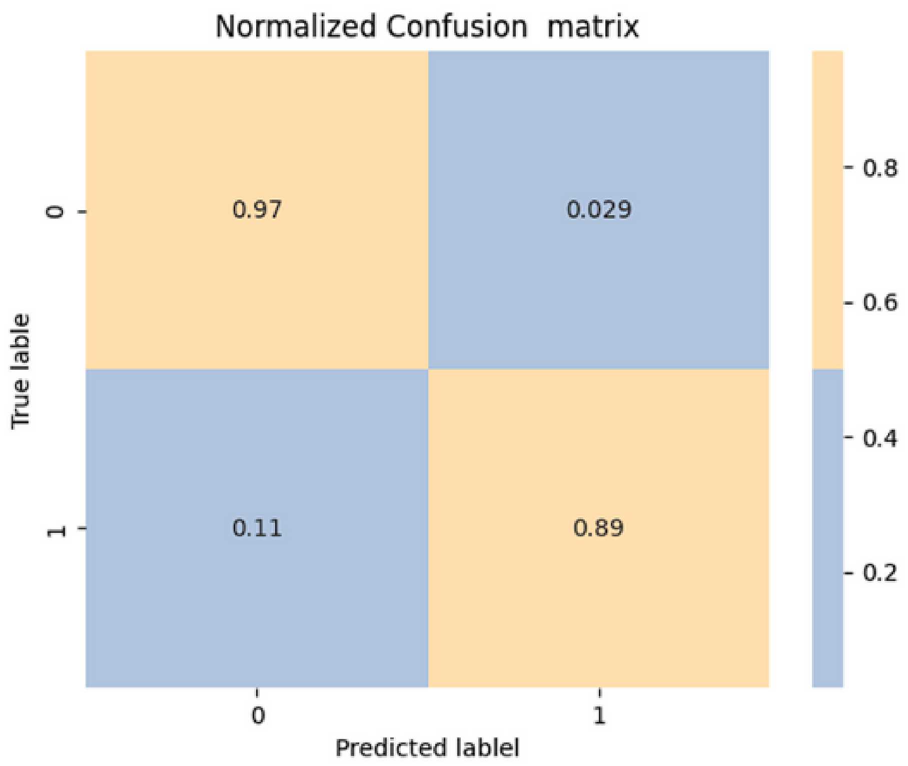
```
model_2= RandomForestClassifier(n_estimators=100,random_state=0)
all(model_2)
```

Confusion Matrix:

confusion matrix



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[102   3]
 [  7  59]]
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.95       105
         1.0       0.95      0.89      0.92        66

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.94       171
weighted avg       0.94      0.94      0.94       171

accuracy_score :  0.9415204678362573
```

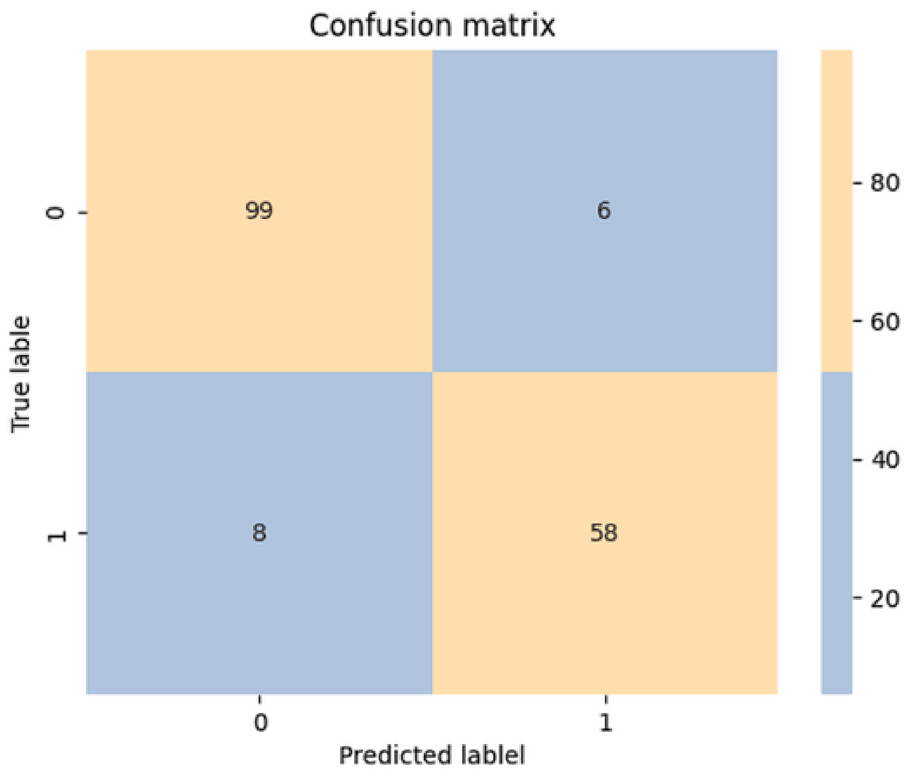
Decision Tree

Decision Trees are versatile machine learning models that make decisions by recursively partitioning data based on features. They're used for classification and regression tasks, visualizing complex decision-making processes. Decision Trees are interpretable, efficient, and handle non-linear relationships well, making them valuable in various domains like finance, healthcare, and marketing.

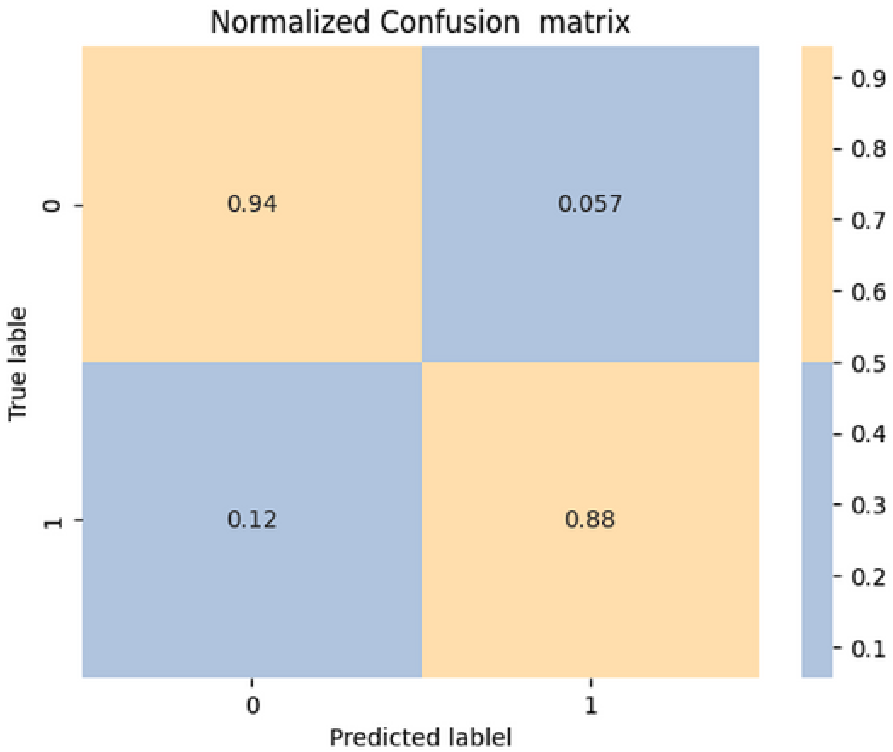
Implementation in Code :

```
[ ]  model_3 = DecisionTreeClassifier(random_state=42)
     all(model_3)
```

Confusion Matrix:



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[99  6]
 [ 8 58]]
              precision    recall  f1-score   support

         0.0       0.93      0.94      0.93       105
         1.0       0.91      0.88      0.89        66

    accuracy                           0.92       171
   macro avg       0.92      0.91      0.91       171
weighted avg       0.92      0.92      0.92       171

accuracy_score :  0.9181286549707602
```

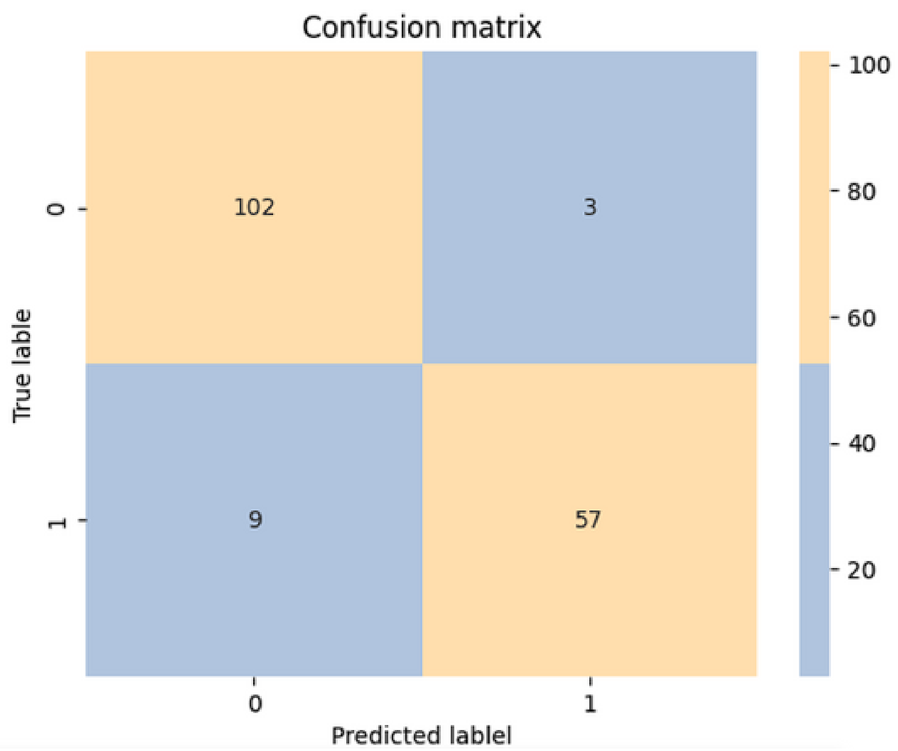
Naive Bayes

Naive Bayes, a probabilistic machine learning algorithm, is particularly suited for classification tasks. It relies on Bayes' theorem and assumes independence between features, simplifying computations. Despite its "naive" assumption, Naive Bayes often performs well in text categorization, spam filtering, and sentiment analysis, showcasing its efficiency and simplicity in diverse applications.

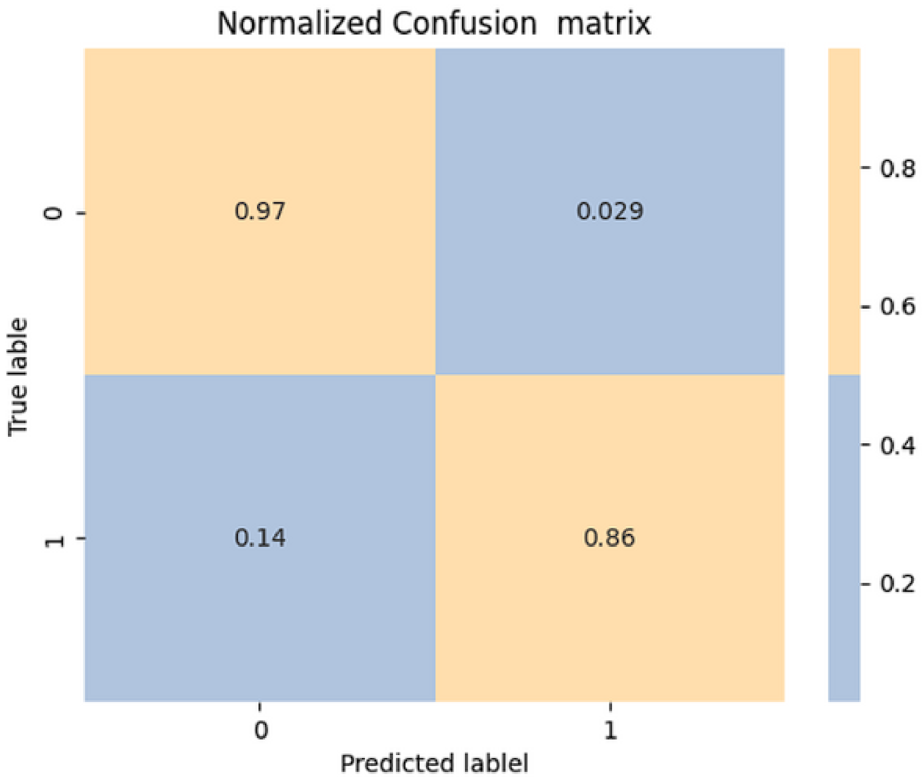
Implementation in Code :

```
model_4 = GaussianNB()
all(model_4)
```

Confusion Matrix:



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[102   3]
 [  9  57]]
              precision    recall  f1-score   support

         0.0       0.92      0.97      0.94       105
         1.0       0.95      0.86      0.90        66

    accuracy                           0.93       171
   macro avg       0.93      0.92      0.92       171
weighted avg       0.93      0.93      0.93       171

accuracy_score :  0.9298245614035088
```

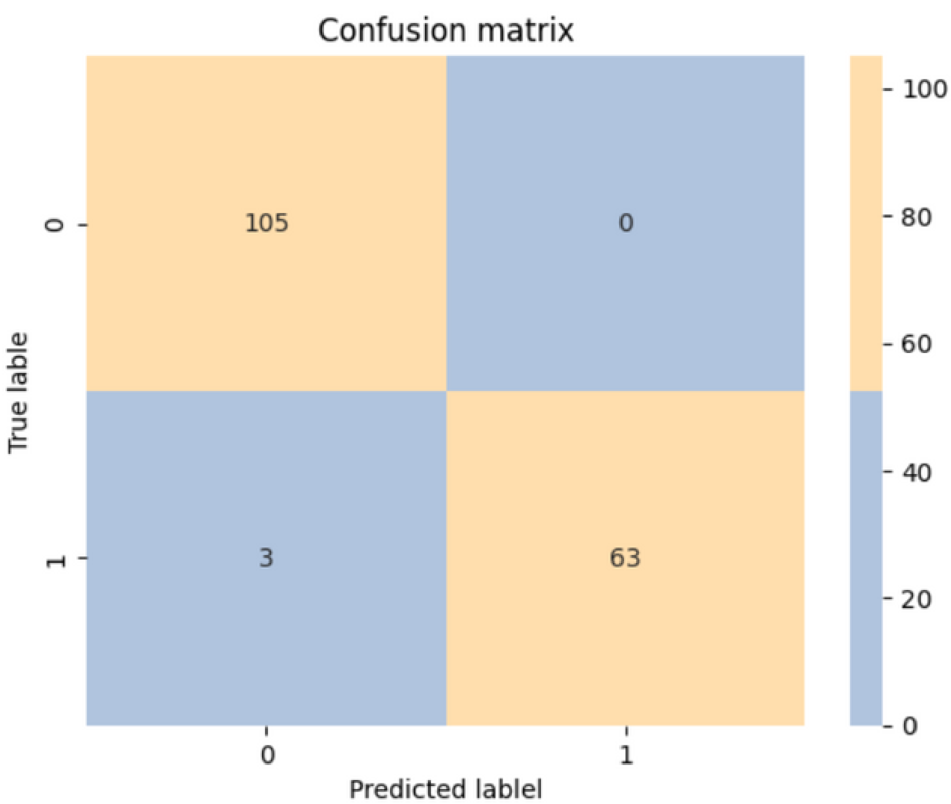
Logistic Regression

Logistic Regression is a fundamental and widely employed algorithm in data science for binary and multi-class classification tasks. Despite its name, it is used for classification rather than regression. The algorithm models the probability that an instance belongs to a particular class, employing the logistic function to constrain the output between 0 and 1.
In data science applications, Logistic Regression is favored for its simplicity, interpretability, and efficiency.

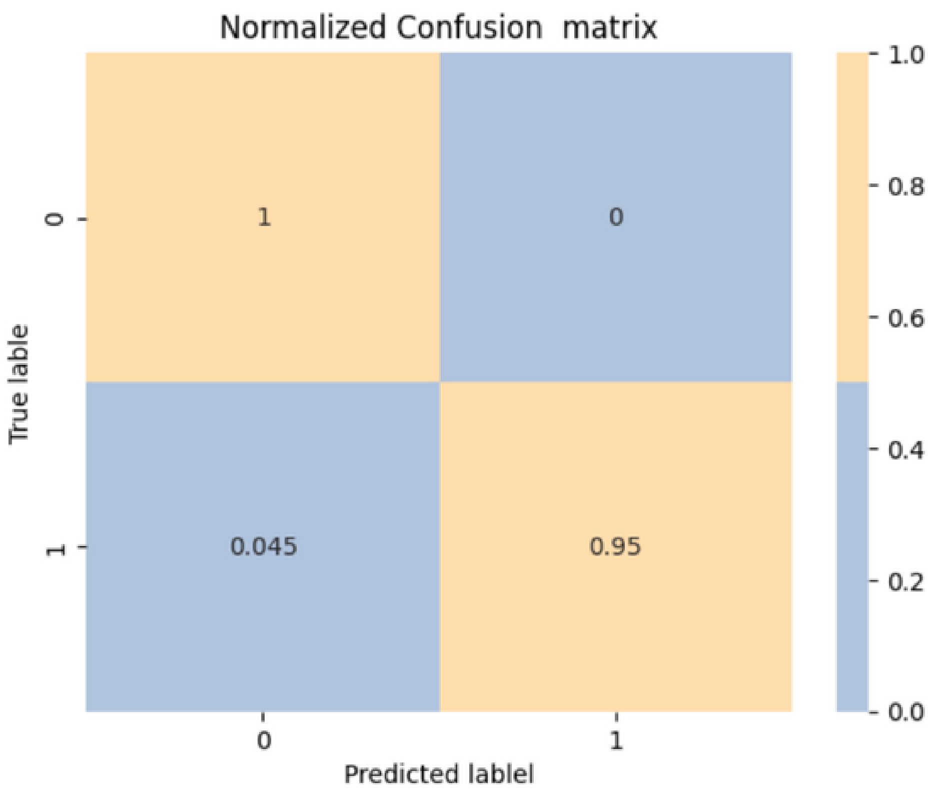
Implementation in Code :

```
model_5 = LogisticRegression()
all(model_5)
```

Confusion Matrix:



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[105   0]
 [  3  63]]
              precision    recall  f1-score   support

         0.0       0.97      1.00      0.99       105
         1.0       1.00      0.95      0.98        66

    accuracy                           0.98       171
   macro avg       0.99      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171

accuracy_score :  0.9824561403508771
```

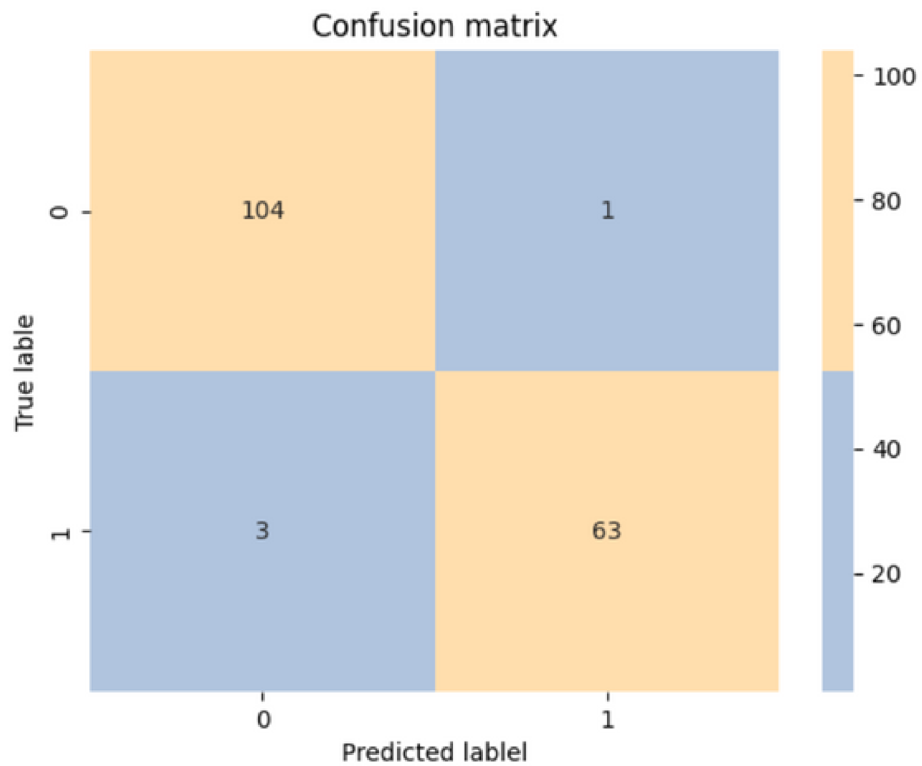
SVC

Support Vector Machines (SVM), including its variant Support Vector Classification (SVC), is a powerful algorithm in data science used for classification and regression tasks. SVC is particularly valuable in scenarios where the decision boundary between classes is complex or not easily linearly separable. SVC works by finding the hyperplane that best separates different classes while maximizing the margin between them

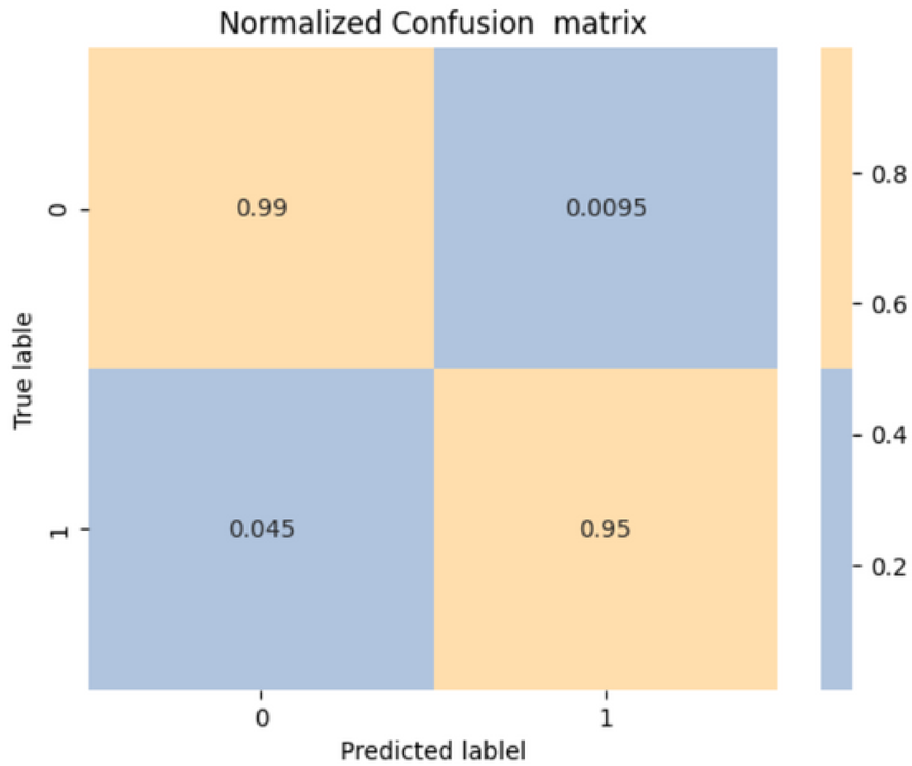
Implementation in Code :

```
[ ]  model_6 = SVC(kernel = 'linear')
     all(model_6)
     model_6.fit(x_train, y_train)
```

Confusion Matrix:



Normalized Confusion Matrix:



Evaluation Matrix:

```
[[104   1]
 [  3  63]]
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98       105
         1.0       0.98      0.95      0.97        66

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.98       171
weighted avg       0.98      0.98      0.98       171

accuracy_score :  0.9766081871345029
```

Final Results

We now move on to plotting the final accuracy results using Matplot Library functions, which will help us in visualising the performance of different models using a simple line graph. Initially we create a data frame to store the corresponding pairs of accuracy of the models in two different lists.

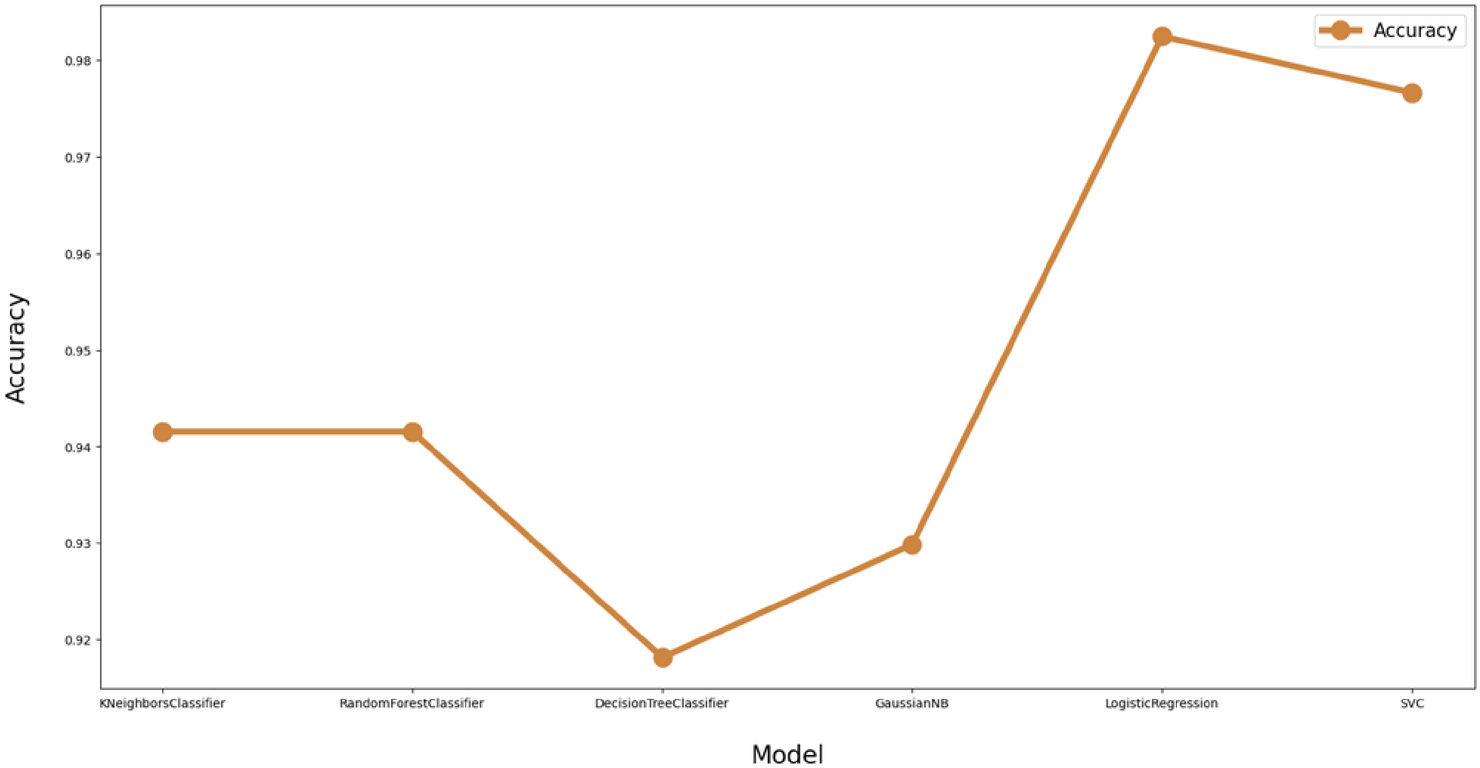
```
df = pd.DataFrame({'Algorithm':algorithm,'Accuracy':Accuracy })
```

The provided code creates a pandas DataFrame (df) to store information about the accuracy of different machine learning algorithms.

```
fig = plt.figure(figsize=(20,10))
plt.plot(df.Algorithm,df.Accuracy,label='Accuracy',lw=5,color='peru',marker='o',markersize = 15)
plt.legend(fontsize=15)
plt.xlabel('\nModel',fontsize= 20)
plt.ylabel('Accuracy\n',fontsize= 20)
plt.show()
```

The provided code utilizes Matplotlib library functions to create a line plot visualizing the accuracy scores of different machine learning algorithms. The x-label denotes the type of model that was implemented and the y-label denotes the corresponding accuracy , then we finally move on to use the plt.show() function to print the line graph depicting the final results shown on the next page under the conclusions tab.

Conclusion



The line plot depicting accuracy scores for various machine learning algorithms reveals notable performance variations. Among the algorithms tested. This goes on t

Highest Accuracy: Logistic Regression
Logistic Regression stands out with the highest accuracy score (close to 0.982) , showcasing its effectiveness in this particular context. Its predictive capability surpasses other algorithms, making it a promising choice for this classification task.

Noteworthy Performers:
KNeighbour Classifier and Random Forest Classifier both achieve a commendable accuracy (close to 0.941) , reflecting robust predictive capabilities.
Support Vector Classifier (SVC) follows closely with an accuracy of 0.977, indicating strong performance in classifying instances.

Lowest Accuracy: Decision Tree Classifier
While still delivering a respectable accuracy close to 0.918, the Decision Tree Classifier exhibits a slightly lower performance compared to other algorithms in this evaluation.

Final Inference

After going through the various steps of data pre-processing, data analysis, and then deploying various machine learning algorithms on the same dataset, performing the same task, here's what we could conclude by the final results:

Logistic Regression tends to outperform other classification algorithms when the relationship between features and the outcome is linear, interpretability is essential, the task is binary classification, the dataset has limited complexity, outliers are present, and dealing with imbalanced datasets. Logistic Regression's simplicity and ability to provide clear coefficients contribute to its effectiveness in these scenarios.

Code [*CLICK HERE*]