Objective: The objective of this assessment is to evaluate your understanding and ability to apply supervised learning techniques to a real-world dataset.

Dataset: Use the breast cancer dataset available in the sklearn library.

Key components to be fulfilled :

1.Loading and Preprocessing (2 marks) Load the breast cancer dataset from sklearn. Preprocess the data to handle any missing values and perform necessary feature scaling. Explain the preprocessing steps you performed and justify why they are necessary for this dataset. 2.Classification Algorithm Implementation (5 marks) Implement the following five classification algorithms: *Logistic Regression*Decision Tree Classifier *Random Forest Classifier*Support Vector Machine (SVM) *k-Nearest Neighbors (k-NN) For each algorithm, provide a brief description of how it works and why it might be suitable for this dataset. 3.Model Comparison (2 marks) Compare the performance of the five classification algorithms. Which algorithm performed the best and which one performed the worst? 4.Timely Submission (1 mark) Submission Guidelines: Provide your code in a Jupyter Notebook format and submit the GitHub link here. Ensure your explanations and answers are clear and concise.

In [ ]:

1.Loading and Preprocessing (2 marks) Load the breast cancer dataset from sklearn. Preprocess the data to handle any missing values and perform necessary feature scaling. Explain the preprocessing steps you performed and justify why they are necessary for this dataset.

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
data = load_breast_cancer(as_frame=True)
df = data.frame
df
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

569 rows × 31 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   mean radius             569 non-null    float64
 1   mean texture            569 non-null    float64
 2   mean perimeter          569 non-null    float64
 3   mean area               569 non-null    float64
 4   mean smoothness         569 non-null    float64
 5   mean compactness        569 non-null    float64
 6   mean concavity          569 non-null    float64
 7   mean concave points     569 non-null    float64
 8   mean symmetry           569 non-null    float64
 9   mean fractal dimension  569 non-null    float64
 10  radius error            569 non-null    float64
 11  texture error           569 non-null    float64
 12  perimeter error         569 non-null    float64
 13  area error              569 non-null    float64
 14  smoothness error        569 non-null    float64
 15  compactness error       569 non-null    float64
 16  concavity error         569 non-null    float64
 17  concave points error    569 non-null    float64
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
```

```
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [4]:
```
df.describe()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	sy
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	

8 rows × 31 columns

In [5]:
```
df.isnull().sum()
```

Out[5]:

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [6]:
```
df.duplicated().sum()
```

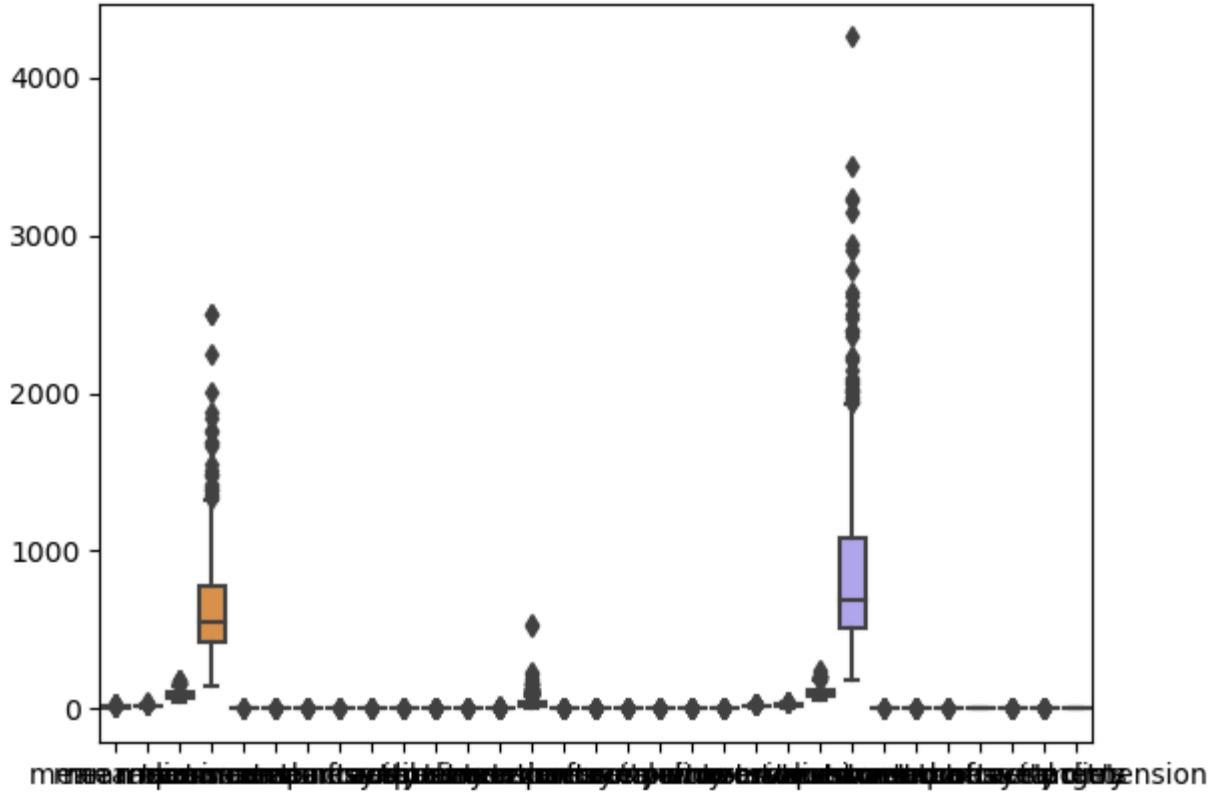
Out[6]:
```
0
```

In [7]:
```
df.skew()
```

Out[7]:

```
mean radius                0.942380
mean texture               0.650450
mean perimeter             0.990650
mean area                  1.645732
mean smoothness            0.456324
mean compactness           1.190123
mean concavity             1.401180
mean concave points        1.171180
mean symmetry              0.725609
mean fractal dimension     1.304489
radius error               3.088612
texture error              1.646444
perimeter error            3.443615
area error                 5.447186
smoothness error           2.314450
compactness error          1.902221
concavity error            5.110463
concave points error       1.444678
symmetry error             2.195133
fractal dimension error    3.923969
worst radius               1.103115
worst texture              0.498321
worst perimeter            1.128164
worst area                 1.859373
worst smoothness           0.415426
worst compactness          1.473555
worst concavity            1.150237
worst concave points       0.492616
worst symmetry             1.433928
worst fractal dimension    1.662579
target                    -0.528461
dtype: float64
```

In [8]:
```
sns.boxplot(df)
plt.show()
```



In [9]:
```
# data info, to get a concise summary of the DataFrame
# Skew for check the outliers
# Missing value check for avoid errors in model training and to ensure the data quality is ve
# Standardizing, it will help to improve the perfomance of machine learning algorithms
```

In [10]:
```
df.columns
```

Out[10]:
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

In [11]:
```
x = df.drop(columns="target")
y = df["target"]
```

In [12]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y)
```

In [13]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

In [ ]:
```

```

In [ ]:

2.Classification Algorithm Implementation (5 marks) Implement the following five classification algorithms: *Logistic Regression* Decision Tree Classifier *Random Forest Classifier* Support Vector Machine (SVM) *k-Nearest Neighbors (k-NN) For each algorithm, provide a brief description of how it works and why it might be suitable for this dataset.

- LogisticRegression - LogisticRegression is liner model used for binary clasification . It calculates the probability that a given input point belongs to a certain class. Its oftesn used in binary classification problems like this one
- DecisionTreeClassifier - Decision Tree buildes a tree like structure where each internal node represents a "test" on an attribute , and each branch represents the outcome of the test . It handles both numerical and categorical data and provides an interpretable model
- RandomForestClassifier - RandomForest is an ensemble method that buuilds multiple decision trees and merges them together to get more accurate and stable prediction. Its effective for larege datasets and can reduce overfitting by averaging multiple trees
- Support Vector Mechine (SVM) - SVM works by finding the hyperplane that best seperates the classes in the future spaces . Its especilally powerfull in high dimensiol spaces. Its effective in casses where the number of dimensions is greater than the number of the samples.
- K- Nearest Neighbors - K-NN is simple algorithm that stores all available cases and classifies new cases based on a similarity measure (eg.distance...). Its intuitive and works well with small datasets and simple classification

In [14]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [15]:
```
cla_alg = {
    "LogisticRegression " : LogisticRegression(),
    "DecisionTreeClassifier " : DecisionTreeClassifier(),
    "RandomForestClassifier " : RandomForestClassifier(),
    "SVC " : SVC(),
    "KNeighborsClassifier" : KNeighborsClassifier(n_neighbors=3)
}
```

In [16]:
```
result = {}
for name,clf in cla_alg.items():
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_test,y_pred)
    con_mat = confusion_matrix(y_test,y_pred)
    cla_rep = classification_report(y_test,y_pred)
    result[name] = accuracy,con_mat
    print(f"{name} : {accuracy}\n")
    print(f"{name} : {cla_rep}")
    print(f"{name} : \n{con_mat}\n")

    sns.heatmap(con_mat,annot= True)
    plt.title(f"Confusion matix of {name}")
```

```
    plt.show()
    print("\n")
```

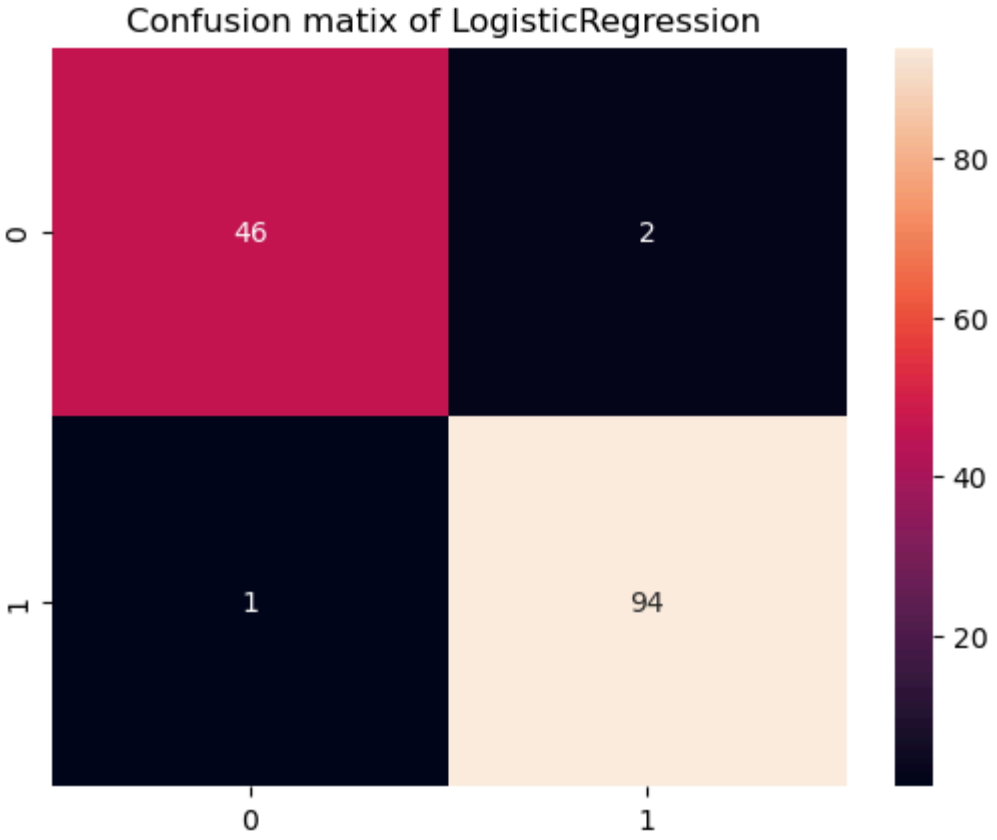
```
LogisticRegression  : 0.9790209790209791

LogisticRegression  :               precision    recall  f1-score   support

           0       0.98      0.96      0.97        48
           1       0.98      0.99      0.98        95

    accuracy                           0.98       143
   macro avg       0.98      0.97      0.98       143
weighted avg       0.98      0.98      0.98       143

LogisticRegression  :
[[46  2]
 [ 1 94]]
```


Confusion matix of LogisticRegression
46
2
1
94
0
1
0
1
80
60
40
20

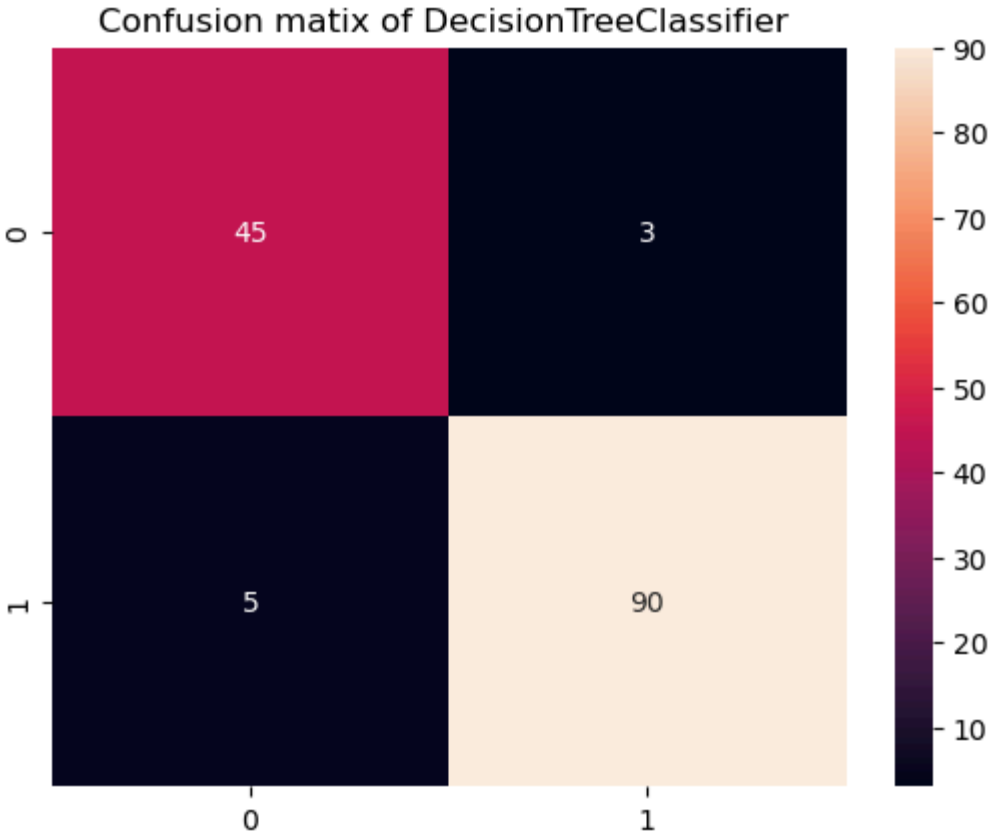
```
DecisionTreeClassifier  : 0.9440559440559441

DecisionTreeClassifier  :              precision    recall  f1-score   support

           0       0.90      0.94      0.92        48
           1       0.97      0.95      0.96        95

    accuracy                           0.94       143
   macro avg       0.93      0.94      0.94       143
weighted avg       0.95      0.94      0.94       143

DecisionTreeClassifier  :
[[45  3]
 [ 5 90]]
```


Confusion matix of DecisionTreeClassifier
45
3
5
90
0
1
0
1
90
80
70
60
50
40
30
20
10

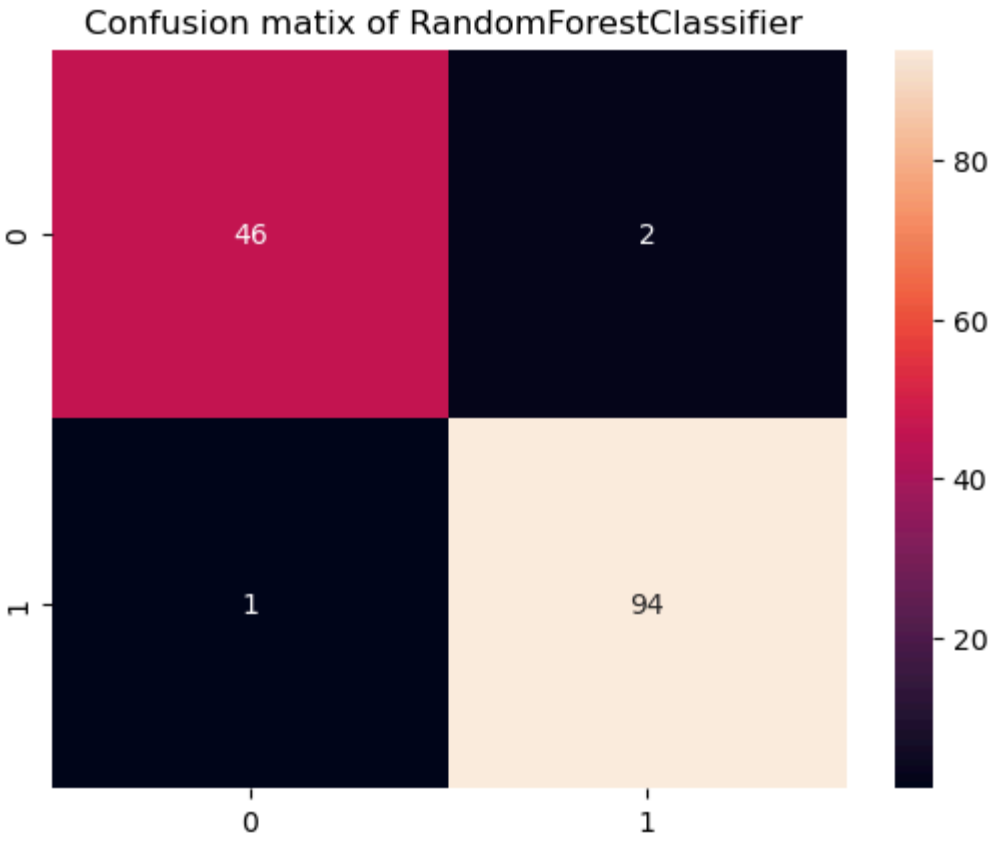
```
RandomForestClassifier  : 0.9790209790209791

RandomForestClassifier  :              precision    recall  f1-score   support

           0       0.98      0.96      0.97        48
           1       0.98      0.99      0.98        95

    accuracy                           0.98       143
   macro avg       0.98      0.97      0.98       143
weighted avg       0.98      0.98      0.98       143

RandomForestClassifier  :
[[46  2]
 [ 1 94]]
```


Confusion matix of RandomForestClassifier
46
2
1
94
0
1
0
1
80
60
40
20

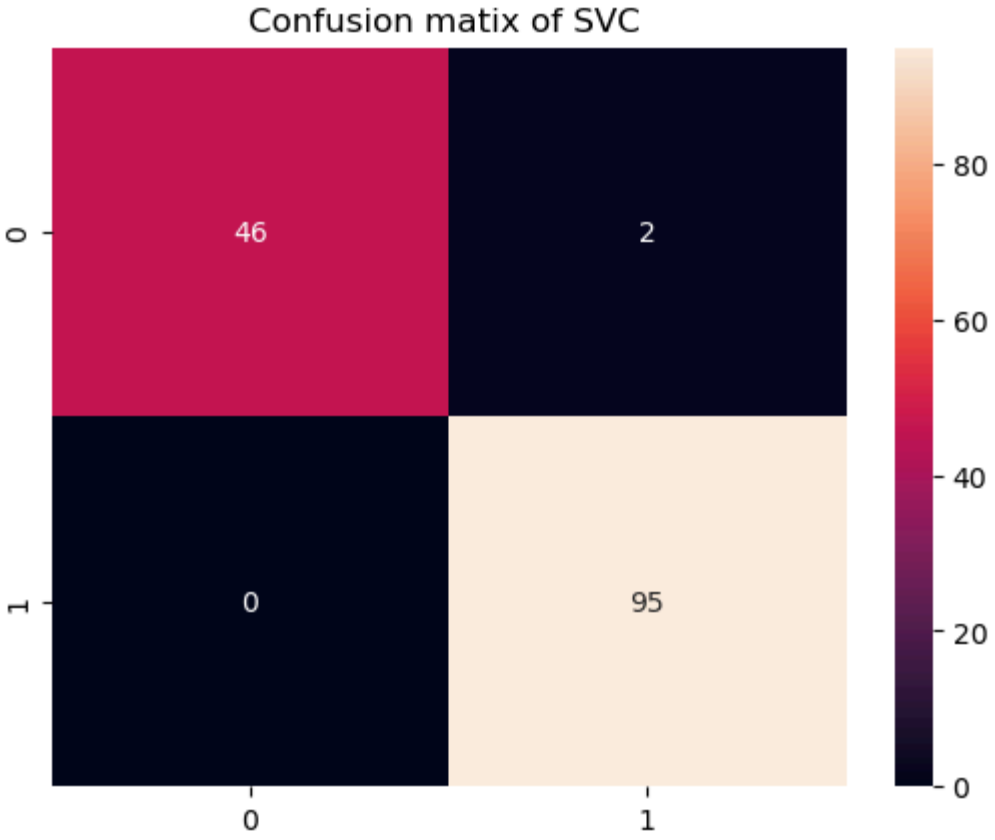
```
SVC  : 0.986013986013986

SVC  :               precision    recall  f1-score   support

           0       1.00      0.96      0.98        48
           1       0.98      1.00      0.99        95

    accuracy                           0.99       143
   macro avg       0.99      0.98      0.98       143
weighted avg       0.99      0.99      0.99       143

SVC  :
[[46  2]
 [ 0 95]]
```


Confusion matix of SVC
46
2
0
95
0
1
0
1
80
60
40
20
0

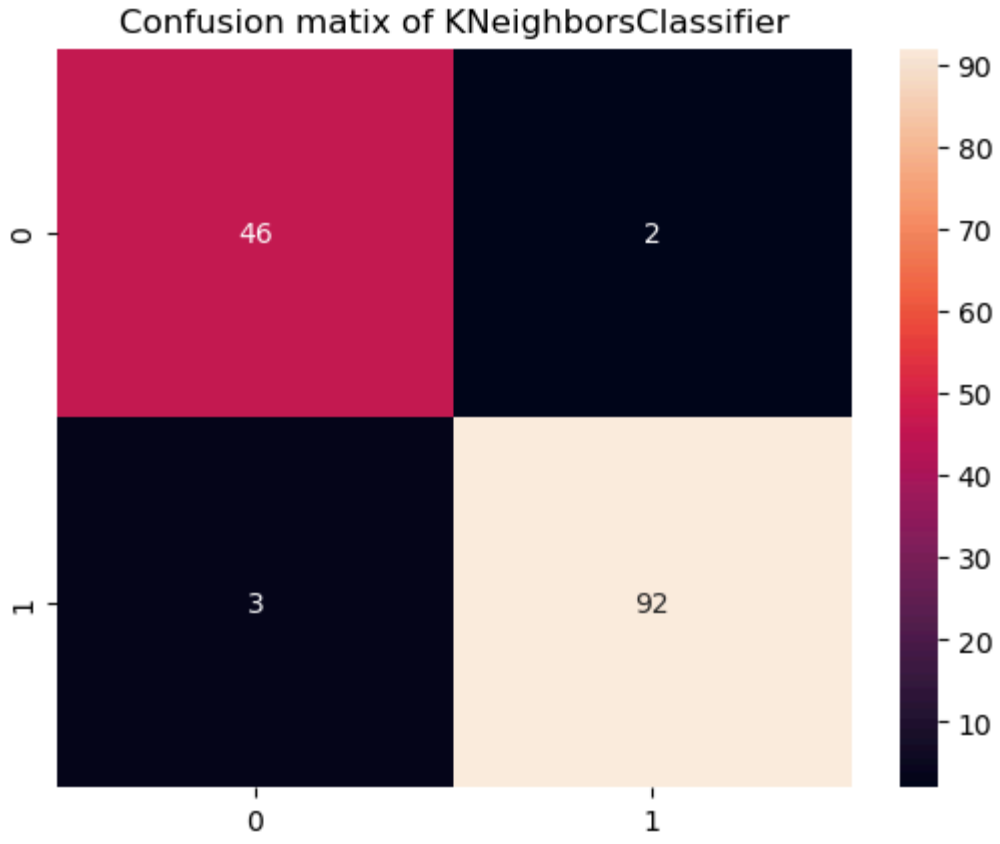
```
KNeighborsClassifier : 0.965034965034965

KNeighborsClassifier :               precision    recall  f1-score   support

           0       0.94      0.96      0.95        48
           1       0.98      0.97      0.97        95

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

KNeighborsClassifier :
[[46  2]
 [ 3 92]]
```


Confusion matix of KNeighborsClassifier
46
2
3
92
0
1
0
1
90
80
70
60
50
40
30
20
10

In [ ]:

In [ ]: