LOADING AND PREPROCESSING

```
from sklearn.datasets import load_breast_cancer
```

```
cancer=load_breast_cancer()
cancer
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
```

```
(standard error):                6.802  542.2\nsmoothness (standard error):         0.002  0.031\ncompactness (standard
error):         0.002  0.135\nconcavity (standard error):           0.0    0.396\nconcave points (standard error):      0.0
0.053\nsymmetry (standard error):            0.008  0.079\nfractal dimension (standard error):   0.001  0.03\nradius
(worst):                      7.93   36.04\ntexture (worst):                      12.02  49.54\nperimeter (worst):
50.41  251.2\narea (worst):                        185.2  4254.0\nsmoothness (worst):                   0.071
0.223\ncompactness (worst):                 0.027  1.058\nconcavity (worst):                     0.0    1.252\nconcave
points (worst):              0.0    0.291\nsymmetry (worst):                     0.156  0.664\nfractal dimension (worst):
0.055  0.208\n===================================== ====== ======\n\n:Missing Attribute Values: None\n\n:Class Distribution:
212 - Malignant, 357 - Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n:Donor: Nick
Street\n\n:Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic)
datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a
breast mass.  They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above
was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming."
Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification
method which uses linear\nprogramming to construct a decision tree.  Relevant features\nwere selected using an exhaustive
search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to obtain the separating
plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming
Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n..
dropdown:: References\n\n  - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction\n    for breast
tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n    Electronic Imaging: Science and Technology, volume 1905,
pages 861-870,\n    San Jose, CA, 1993.\n  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and\n
prognosis via linear programming. Operations Research, 43(4), pages 570-577,\n    July-August 1995.\n  - W.H. Wolberg, W.N.
Street, and O.L. Mangasarian. Machine learning techniques\n    to diagnose breast cancer from fine-needle aspirates. Cancer
Letters 77 (1994)\n    163-171.\n',
 'feature_names': array(['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',
```

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
print(cancer['feature_names'])
```

```
['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']
```

```
print(cancer['data'][0])
```

```
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

```
cancer['data'].shape
```

```
(569, 30)
```

```
import pandas as pd
import numpy as np
df_cancer=pd.DataFrame(np.c_[cancer['data'],cancer['target']],columns=np.append(cancer['feature_names'],['target']))
df_cancer
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimete
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.6
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.8
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.5
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.8
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.2
...	...	...	...	...	...	...	...	...	...	...	...	...	.
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.7
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.6
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.1

569 rows × 31 columns

```
df_cancer.shape
```

```
(569, 31)
```

```
df_cancer.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	..
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	..
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	..
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	..
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	..
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	..
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	..
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	..

8 rows × 31 columns

DATA PREPROCESSING

```
df_cancer.duplicated().sum()
```

```
0
```

```
df_cancer.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    float64
dtypes: float64(31)
memory usage: 137.9 KB
```

```
df_cancer.isnull().sum()
```

	0
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

```
x=df_cancer.drop('target',axis=1)
y=df_cancer['target']
x
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67
...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37

569 rows × 30 columns

```
y
```

	target
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
564	0.0
565	0.0
566	0.0
567	0.0
568	1.0

569 rows × 1 columns

dtype: float64

```
#split data to train and test

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y)
```

```
x_train.shape
```

```
(426, 30)
```

```
y_train.shape
```

```
(426,)
```

```
x_test.shape
```

```
(143, 30)
```

```
y_test.shape
```

```
(143,)
```

```
#scaling data

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()


x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
```

```
x_train
```

```
array([[-0.11260789,  0.92201873, -0.16545343, ..., -0.62077161,
        -0.42469392, -0.80372721],
       [-0.02244327, -1.40112566, -0.03815238, ..., -0.16175543,
        -1.24478977, -0.63770518],
       [ 1.62869645,  0.53636753,  1.56902327, ...,  1.07987099,
         0.30352397, -0.08319159],
       ...,
       [-0.8085661 , -1.42190926, -0.8313358 , ..., -0.87804278,
        -0.05548064, -0.90389383],
       [-0.44227231, -0.81687563, -0.35395689, ...,  0.95802194,
         1.86034493,  2.23281576],
       [-0.63668978, -0.25340919, -0.66282512, ..., -0.62293096,
        -1.55445252, -0.82475667]])
```

```
x_test
```

```
array([[ 0.77055256,  1.83735121,  0.72832304, ...,  0.49012906,
         0.11709937,  0.38814075],
       [ 1.94451087,  1.69903497,  2.07199288, ...,  2.48666846,
         1.67094058,  2.49183469],
       [-0.62882918, -0.84550675, -0.65245768, ..., -1.099845  ,
        -0.63697063, -1.14455054],
       ...,
       [-0.38291272,  0.30871836, -0.34746595, ..., -0.06380694,
        -0.52128944, -0.10494729],
       [ 0.158689  ,  1.37232249,  0.06032718, ..., -1.38362612,
        -1.94659599, -1.59088984],
       [ 0.17625446, -0.089537  ,  0.13369582, ..., -0.15415479,
        -0.92260321, -0.7839173 ]])
```

LOGISTIC REGRESSION

```
#Fitting data to model

from sklearn.linear_model import LogisticRegression
log_reg=LogisticRegression()
log_reg.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
#model prediction

y_pred=log_reg.predict(x_test)
```

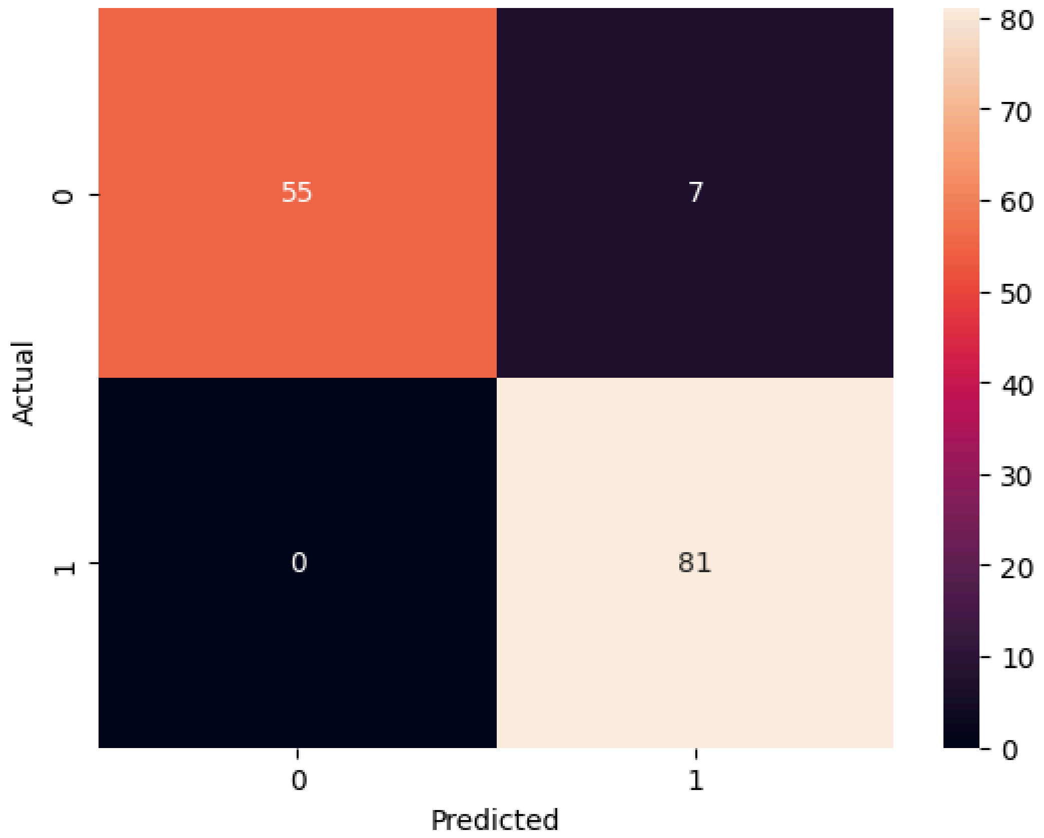
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
cm=confusion_matrix(y_test,y_pred)
print("Confusion Matrix")
print (cm)
```

```
Confusion Matrix
[[55  7]
 [ 0 81]]
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.89      0.94        62
         1.0       0.92      1.00      0.96        81

    accuracy                           0.95       143
   macro avg       0.96      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```

```
#accuracy score
log_reg_acc=accuracy_score(y_test,y_pred)
print(log_reg_acc)
```

```
0.951048951048951
```

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier ⓘ ?
DecisionTreeClassifier()
```

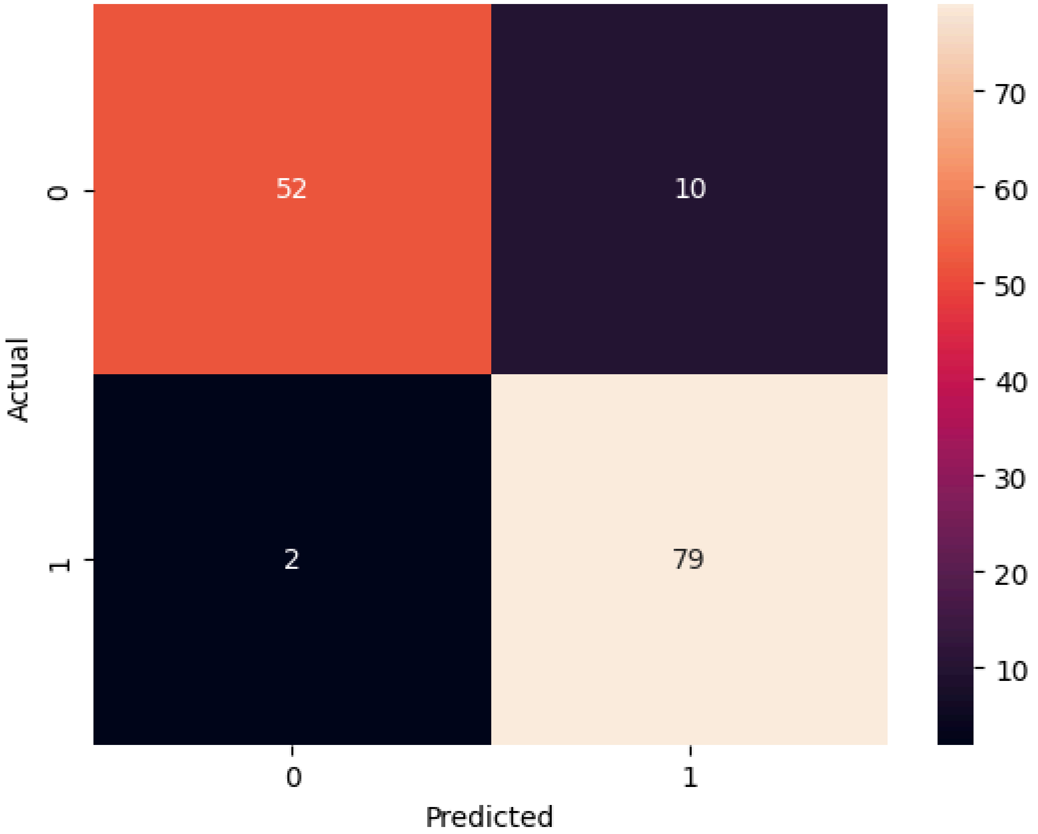
```
y_pred=dtc.predict(x_test)
y_pred
```

```
array([0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
       1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
       1., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 0., 1., 0.,
       1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1.,
       1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1.,
       1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1.,
       1., 0., 1., 1., 1., 0., 1.])
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[52 10]
 [ 2 79]]
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.84      0.90        62
         1.0       0.89      0.98      0.93        81
```

```
    accuracy                           0.92       143
   macro avg       0.93      0.91      0.91       143
weighted avg       0.92      0.92      0.92       143
```

```
#accuracy score

dtc_acc=accuracy_score(y_test,y_pred)
print(dtc_acc)
```

```
0.916083916083916
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
rand_clf=RandomForestClassifier()
rand_clf.fit(x_train,y_train)
```

```
▾ RandomForestClassifier i ?
RandomForestClassifier()
```

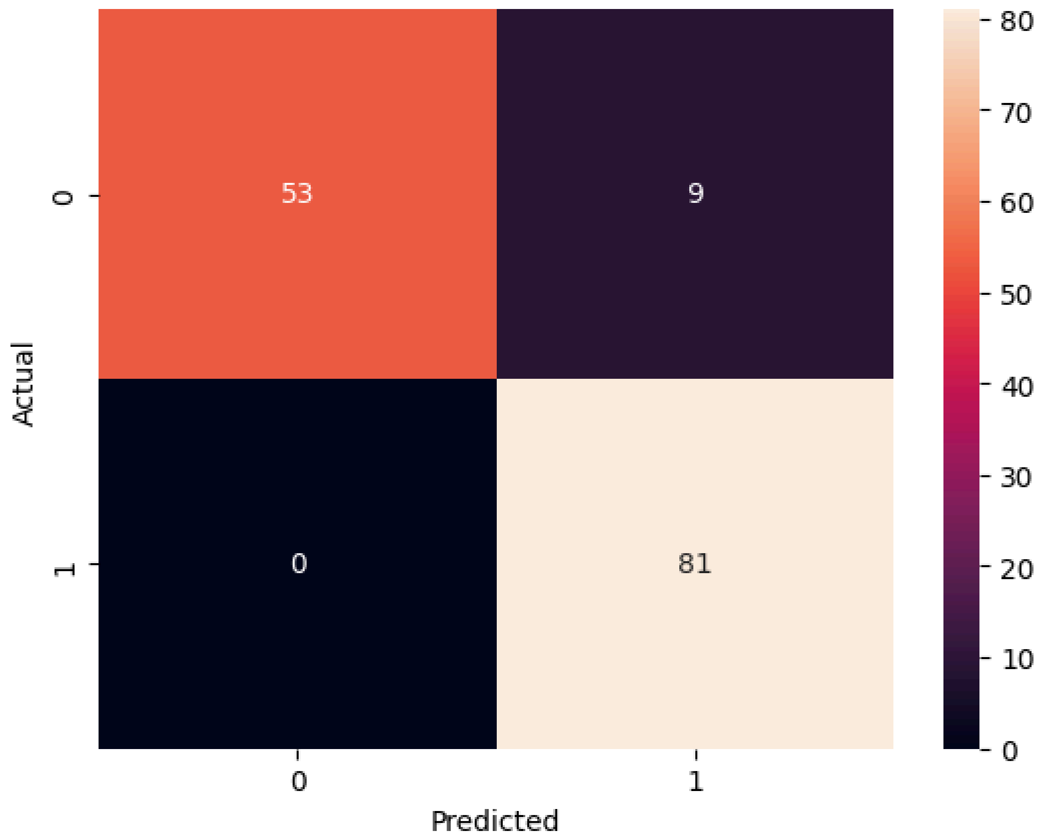
```
y_pred=rand_clf.predict(x_test)
y_pred
```

```
array([0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
       1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
       1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1.,
       1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
       1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1.,
       1., 0., 0., 1., 1., 1., 1.])
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[53  9]
 [ 0 81]]
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.85      0.92        62
         1.0       0.90      1.00      0.95        81

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

```
#accuracy score
ran_clf_acc=accuracy_score(y_test,y_pred)
print(ran_clf_acc)
```

```
0.9370629370629371
```

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
svc_clf=SVC()
svc_clf.fit(x_train,y_train)
```

```
▾ SVC
SVC()
```

```
y_pred=svc_clf.predict(x_test)
y_pred
```

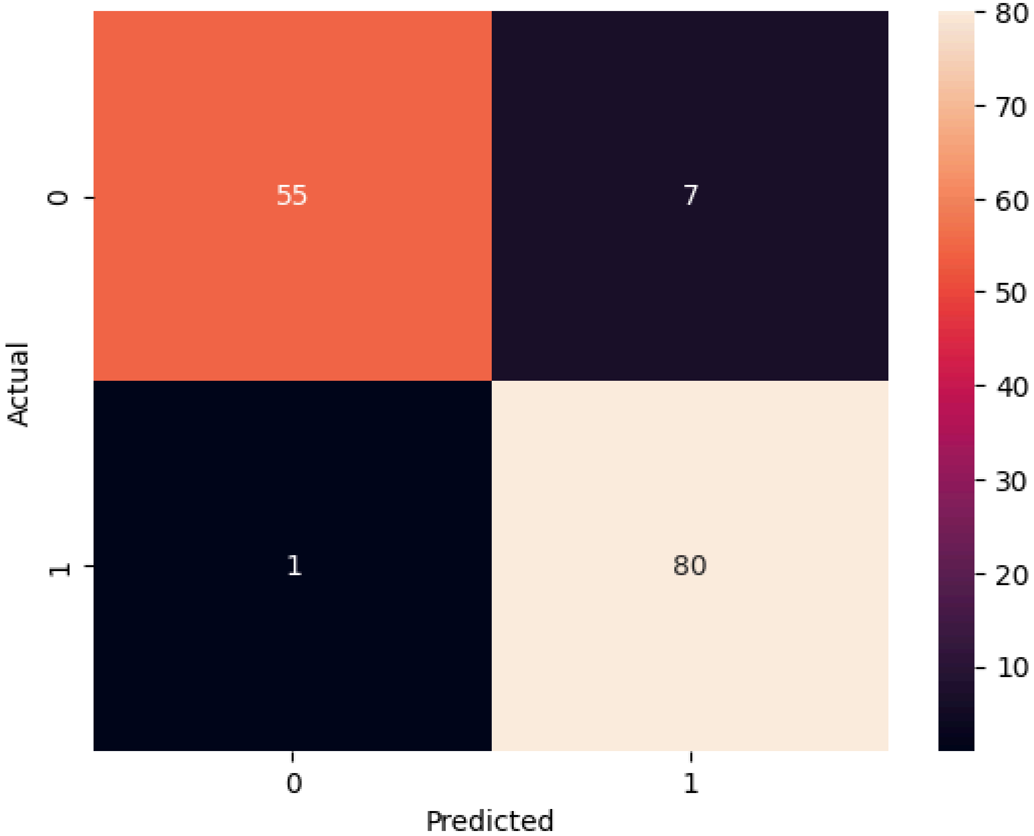
```
array([0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
       0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
```

```
       1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1.,
       1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
       1., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1.,
       1., 0., 0., 1., 1., 0., 1.])
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[55  7]
 [ 1 80]]
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.89      0.93        62
         1.0       0.92      0.99      0.95        81

    accuracy                           0.94       143
   macro avg       0.95      0.94      0.94       143
weighted avg       0.95      0.94      0.94       143
```

```
#accuracy score
svc_clf_acc=accuracy_score(y_test,y_pred)
print(svc_clf_acc)
```

```
0.9440559440559441
```

K-NEAREST NEIGHBOUR

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf=KNeighborsClassifier()
knn_clf.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier i ?
KNeighborsClassifier()
```

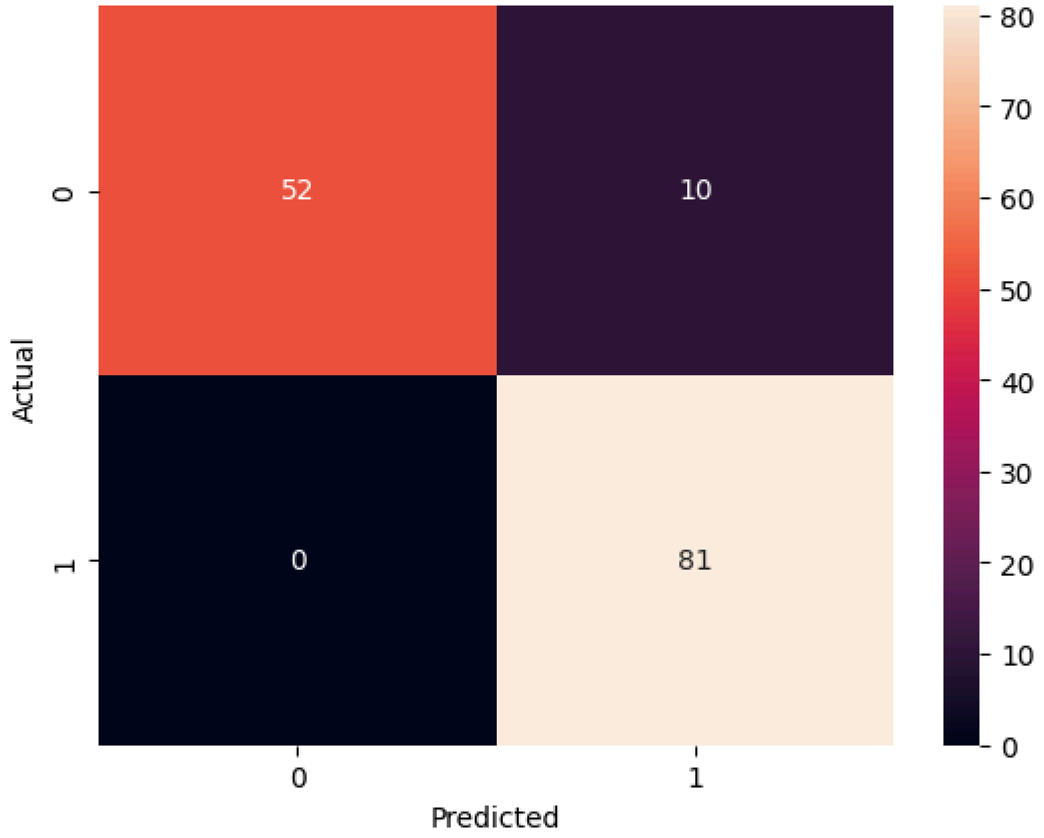
```
y_pred=knn_clf.predict(x_test)
y_pred
```

```
array([0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1.,
       1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1.,
       1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
       1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1.,
       1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
       1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1.,
       1., 0., 1., 0., 1., 1., 1.])
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[52 10]
 [ 0 81]]
```

```
cm=confusion_matrix(y_test,y_pred)
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.84      0.91        62
         1.0       0.89      1.00      0.94        81
```

```
    accuracy                           0.93       143
   macro avg       0.95      0.92      0.93       143
weighted avg       0.94      0.93      0.93       143
```

```
#accuracy score
knn_clf_acc=accuracy_score(y_test,y_pred)
print(knn_clf_acc)
```

```
0.9300699300699301
```

MODEL COMPARISON

```
dict={"model":['LogisticRegression','Decision tree classifier','Random forest classifier','Support vector machine','K-nearest neight

df_model=pd.DataFrame(dict)
df_model.sort_values(by='Score',ascending=False)
```

	model	Score
0	LogisticRegression	0.950
3	Support vector machine	0.940
2	Random forest classifier	0.937
4	K-nearest neighbor	0.930
1	Decision tree classifier	0.916

Logistic regression perfomed the best and Decision tree classifier performed the worst.