In [1]:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
dataset=load_breast_cancer()
df=pd.DataFrame(dataset['data'],columns=dataset['feature_names'])
df['target']=dataset['target']
df.head()
```

Out[1]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mear symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.180

5 rows × 31 columns

In [2]:

```
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 [7]:

```
from sklearn.model_selection import train_test_split
data=df.loc[:, df.columns != 'target']
target=df['target']
x_train,x_test,y_train,y_test=train_test_split(data,target,test_size=0.20,random_state=1)
```

In [8]:

```
print('train data of x_train is: ',x_train.shape)
print('train data of x_test is: ',x_test.shape)
print('train data of y_train is: ',y_train.shape)
print('train data of y_train is: ',y_test.shape)
```

```
train data of x_train is:  (455, 30)
train data of x_test is:  (114, 30)
train data of y_train is:  (455,)
train data of y_train is:  (114,)
```

In [9]:

```
print(y_train.value_counts())
print(y_test.value_counts())
```

```
1    285
0    170
Name: target, dtype: int64
1    72
0    42
Name: target, dtype: int64
```

In [10]:

```
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(x_train,y_train)

#
y_pred=logreg.predict(x_test)
```

```
C:\Users\91920\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:765: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [11]:

```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[11]:

```
array([[38,  4],
       [ 2, 70]], dtype=int64)
```

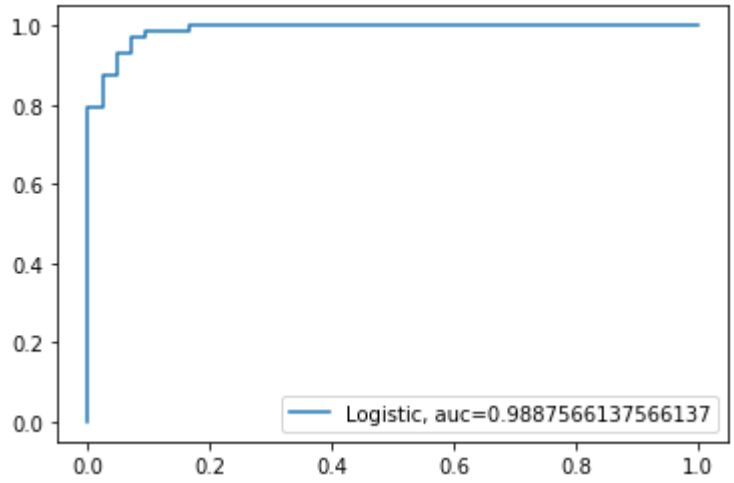
In [15]:

```
print(metrics.classification_report(y_test,y_pred,digits=3))
```

```
              precision    recall  f1-score   support

           0      0.950     0.905     0.927        42
           1      0.946     0.972     0.959        72

    accuracy                          0.947       114
   macro avg      0.948     0.938     0.943       114
weighted avg      0.947     0.947     0.947       114
```

In [37]:

```
import matplotlib.pyplot as plt
y_pred_proba = logreg.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="Logistic, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



In [33]:

```
y_pred_proba = logreg.predict_proba(x_test)[::,1]
y_pred_proba
```

Out[33]:

```
array([6.38650330e-01, 3.62099577e-01, 9.95729876e-01, 1.42059356e-03,
       2.68144191e-01, 1.55733834e-03, 2.06823582e-05, 1.46359015e-02,
       9.78352679e-01, 9.86557528e-01, 9.70041515e-01, 1.59699544e-01,
       1.52724104e-01, 9.82726335e-01, 8.06321011e-01, 8.80536887e-01,
       9.67544684e-01, 9.73968510e-01, 9.99124182e-01, 1.46687819e-06,
       9.89018113e-01, 9.55372997e-01, 1.71820076e-03, 9.75517257e-01,
       8.71135835e-02, 8.85733266e-01, 9.69286332e-01, 8.89055917e-07,
       2.08558239e-18, 4.56083303e-03, 3.31044495e-10, 9.95689742e-01,
       8.15842268e-05, 8.79295548e-02, 9.75610444e-01, 9.71271386e-01,
       1.09022178e-03, 9.13860853e-01, 3.81823267e-01, 9.94001472e-01,
       9.97532985e-01, 8.22593407e-01, 9.36689479e-01, 9.86996906e-01,
       9.77954293e-01, 1.15063032e-01, 9.92408287e-01, 9.97118036e-01,
       7.25342741e-01, 7.71810230e-03, 3.99245045e-05, 2.67445284e-13,
       9.96811900e-01, 9.95735225e-01, 9.87969966e-01, 9.63503695e-01,
       9.88351382e-01, 1.75963249e-01, 9.86817809e-01, 9.98034479e-01,
       9.93560454e-01, 2.26022883e-02, 7.83778642e-01, 2.98967505e-01,
       9.69064503e-01, 9.47338203e-01, 9.30134078e-01, 4.73672087e-08,
       9.96256846e-01, 9.54973319e-01, 9.98000692e-01, 9.98459297e-01,
       5.64985584e-01, 1.10754877e-03, 9.97535796e-01, 4.84610806e-07,
       9.46882613e-01, 7.94635912e-01, 9.95952430e-01, 7.06557477e-04,
       9.96812717e-01, 2.28940947e-06, 9.84902424e-01, 3.41277188e-03,
       9.92075769e-01, 9.91232552e-01, 3.44281498e-01, 9.21143176e-01,
       8.54666256e-03, 9.96456530e-01, 9.61485648e-01, 6.16742005e-04,
       9.96150674e-01, 9.99275750e-01, 4.68573045e-04, 2.62611673e-01,
       9.98780585e-01, 9.76419992e-01, 9.93951668e-01, 9.73139299e-01,
       9.26138966e-01, 9.90907352e-01, 9.94078398e-01, 8.00328104e-01,
       9.75900345e-01, 9.76945760e-01, 9.98873387e-01, 9.95971990e-01,
       1.05299384e-02, 1.69776143e-02, 8.83618008e-01, 9.84649347e-01,
       4.81298734e-01, 9.91868551e-01])
```

In [34]:

```
fpr
```

Out[34]:

```
array([0.        , 0.        , 0.        , 0.02380952, 0.02380952,
       0.04761905, 0.04761905, 0.07142857, 0.07142857, 0.0952381 ,
       0.0952381 , 0.16666667, 0.16666667, 1.        ])
```

In [35]:

```
tpr
```

Out[35]:

```
array([0.        , 0.01388889, 0.79166667, 0.79166667, 0.875     ,
       0.875     , 0.93055556, 0.93055556, 0.97222222, 0.97222222,
       0.98611111, 0.98611111, 1.        , 1.        ])
```

In [36]:

```
_
```

Out[36]:

```
array([1.99927575e+00, 9.99275750e-01, 9.47338203e-01, 9.46882613e-01,
       8.85733266e-01, 8.83618008e-01, 8.00328104e-01, 7.94635912e-01,
       6.38650330e-01, 5.64985584e-01, 4.81298734e-01, 3.44281498e-01,
       2.98967505e-01, 2.08558239e-18])
```

In [23]:

```
from sklearn.neighbors import KNeighborsClassifier
k=int(input('Enter number of neighbours'))
model=KNeighborsClassifier(n_neighbors=k)
#train model
model.fit(x_train,y_train)
predicteddata=model.predict(x_test)
```

```
Enter number of neighbours5
```

In [25]:

```
# import the metrics class
from sklearn import metrics
accuracy=metrics.accuracy_score(y_test,predicteddata)
print("Accuracy of model is : ",accuracy)
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,predicteddata)
cm
```

```
Accuracy of model is :  0.9385964912280702
```

Out[25]:

```
array([[37,  5],
       [ 2, 70]], dtype=int64)
```

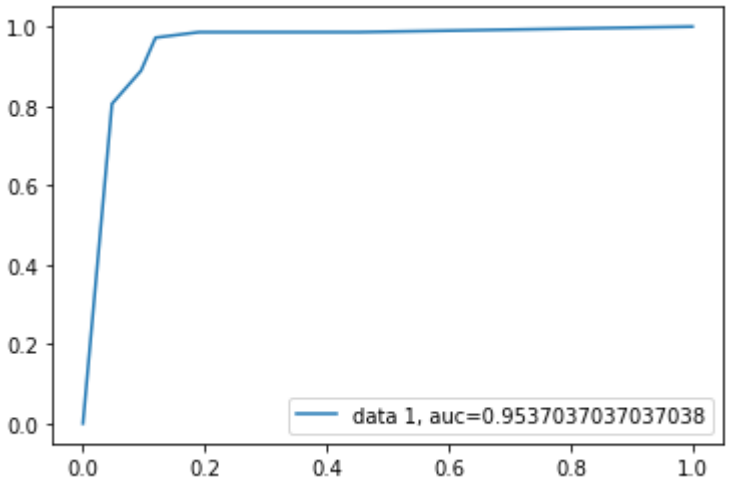
In [26]:

```
from sklearn.metrics import classification_report
cr=classification_report(y_test,predicteddata)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.95      0.88      0.91        42
           1       0.93      0.97      0.95        72

    accuracy                           0.94       114
   macro avg       0.94      0.93      0.93       114
weighted avg       0.94      0.94      0.94       114
```

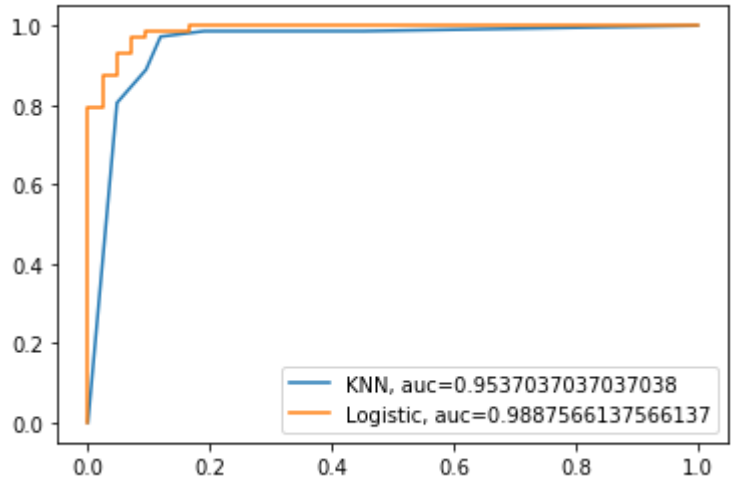
In [28]:

```
import matplotlib.pyplot as plt
y_pred_proba_knn = model.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba_knn)
auc = metrics.roc_auc_score(y_test, y_pred_proba_knn)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))



plt.legend()
plt.show()
```



In [39]:

```
import matplotlib.pyplot as plt
y_pred_proba_knn = model.predict_proba(x_test)[::,1]
fpr_k, tpr_k, _k = metrics.roc_curve(y_test,  y_pred_proba_knn)
auc = metrics.roc_auc_score(y_test, y_pred_proba_knn)
plt.plot(fpr_k,tpr_k,label="KNN, auc="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="Logistic, auc="+str(auc))



plt.legend()
plt.show()
```



In [ ]: