Breast Cancer

```
# Library import
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools

np.random.seed(42)  # for reproducibility
sns.set(rc={"figure.figsize": (8, 8)})
sns.set_style("ticks")
```

Dataset

```
# load the dataset
data = load_breast_cancer()
print(data.DESCR)  #print short description.
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
```

```
radius (worst):                    7.93   36.04
```

```
1 print(f"types of cancer (targets) are {data.target_names}")
```

```
types of cancer (targets) are ['malignant' 'benign']
```

```
1 X = data.data
2 X.shape
```

```
(569, 30)
```

569 features, 30 examples.

```
1 X = data.data #feature
2 y = data.target  #labels
3 print(f"Shape of feature is {X.shape}, and shape of target is {y.shape}")
```

```
Shape of feature is (569, 30), and shape of target is (569,)
```

good idea to split the data three parts train data, test data, and validation data. 369 examples for training and 200 example for testing.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=200, random_state=42, stratify=y)
```

```
1 y_train[:10]           # it prints only last 10 values from the array.
```

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

```
1 X_train[:10]
```

```
array([[1.387e+01, 2.070e+01, 8.977e+01, 5.848e+02, 9.578e-02, 1.018e-01,
        3.688e-02, 2.369e-02, 1.620e-01, 6.688e-02, 2.720e-01, 1.047e+00,
        2.076e+00, 2.312e+01, 6.298e-03, 2.172e-02, 2.615e-02, 9.061e-03,
        1.490e-02, 3.599e-03, 1.505e+01, 2.475e+01, 9.917e+01, 6.886e+02,
        1.264e-01, 2.037e-01, 1.377e-01, 6.845e-02, 2.249e-01, 8.492e-02],
       [1.176e+01, 2.160e+01, 7.472e+01, 4.279e+02, 8.637e-02, 4.966e-02,
        1.657e-02, 1.115e-02, 1.495e-01, 5.888e-02, 4.062e-01, 1.210e+00,
        2.635e+00, 2.847e+01, 5.857e-03, 9.758e-03, 1.168e-02, 7.445e-03,
        2.406e-02, 1.769e-03, 1.298e+01, 2.572e+01, 8.298e+01, 5.165e+02,
        1.085e-01, 8.615e-02, 5.523e-02, 3.715e-02, 2.433e-01, 6.563e-02],
       [1.495e+01, 1.877e+01, 9.784e+01, 6.895e+02, 8.138e-02, 1.167e-01,
        9.050e-02, 3.562e-02, 1.744e-01, 6.493e-02, 4.220e-01, 1.909e+00,
        3.271e+00, 3.943e+01, 5.790e-03, 4.877e-02, 5.303e-02, 1.527e-02,
        3.356e-02, 9.368e-03, 1.625e+01, 2.547e+01, 1.071e+02, 8.097e+02,
        9.970e-02, 2.521e-01, 2.500e-01, 8.405e-02, 2.852e-01, 9.218e-02],
       [1.203e+01, 1.793e+01, 7.609e+01, 4.460e+02, 7.683e-02, 3.892e-02,
        1.546e-03, 5.592e-03, 1.382e-01, 6.070e-02, 2.335e-01, 9.097e-01,
        1.466e+00, 1.697e+01, 4.729e-03, 6.887e-03, 1.184e-03, 3.951e-03,
        1.466e-02, 1.755e-03, 1.307e+01, 2.225e+01, 8.274e+01, 5.234e+02,
        1.013e-01, 7.390e-02, 7.732e-03, 2.796e-02, 2.171e-01, 7.037e-02],
       [1.348e+01, 2.082e+01, 8.840e+01, 5.592e+02, 1.016e-01, 1.255e-01,
        1.063e-01, 5.439e-02, 1.720e-01, 6.419e-02, 2.130e-01, 5.914e-01,
        1.545e+00, 1.852e+01, 5.367e-03, 2.239e-02, 3.049e-02, 1.262e-02,
        1.377e-02, 3.187e-03, 1.553e+01, 2.602e+01, 1.073e+02, 7.404e+02,
        1.610e-01, 4.225e-01, 5.030e-01, 2.258e-01, 2.807e-01, 1.071e-01],
       [1.086e+01, 2.148e+01, 6.851e+01, 3.605e+02, 7.431e-02, 4.227e-02,
        0.000e+00, 0.000e+00, 1.661e-01, 5.948e-02, 3.163e-01, 1.304e+00,
        2.115e+00, 2.067e+01, 9.579e-03, 1.104e-02, 0.000e+00, 0.000e+00,
        3.004e-02, 2.228e-03, 1.166e+01, 2.477e+01, 7.408e+01, 4.123e+02,
        1.001e-01, 7.348e-02, 0.000e+00, 0.000e+00, 2.458e-01, 6.592e-02],
       [1.157e+01, 1.904e+01, 7.420e+01, 4.097e+02, 8.546e-02, 7.722e-02,
        5.485e-02, 1.428e-02, 2.031e-01, 6.267e-02, 2.864e-01, 1.440e+00,
        2.206e+00, 2.030e+01, 7.278e-03, 2.047e-02, 4.447e-02, 8.799e-03,
        1.868e-02, 3.339e-03, 1.307e+01, 2.698e+01, 8.643e+01, 5.205e+02,
        1.249e-01, 1.937e-01, 2.560e-01, 6.664e-02, 3.035e-01, 8.284e-02],
       [1.094e+01, 1.859e+01, 7.039e+01, 3.700e+02, 1.004e-01, 7.460e-02,
        4.944e-02, 2.932e-02, 1.486e-01, 6.615e-02, 3.796e-01, 1.743e+00,
        3.018e+00, 2.578e+01, 9.519e-03, 2.134e-02, 1.990e-02, 1.155e-02,
        2.079e-02, 2.701e-03, 1.240e+01, 2.558e+01, 8.276e+01, 4.724e+02,
        1.363e-01, 1.644e-01, 1.412e-01, 7.887e-02, 2.251e-01, 7.732e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02],
       [1.277e+01, 2.141e+01, 8.202e+01, 5.074e+02, 8.749e-02, 6.601e-02,
        3.112e-02, 2.864e-02, 1.694e-01, 6.287e-02, 7.311e-01, 1.748e+00,
        5.118e+00, 5.365e+01, 4.571e-03, 1.790e-02, 2.176e-02, 1.757e-02,
        3.373e-02, 5.875e-03, 1.375e+01, 2.350e+01, 8.904e+01, 5.795e+02,
        9.388e-02, 8.978e-02, 5.186e-02, 4.773e-02, 2.179e-01, 6.871e-02]])
```

```
1 classifier = svm.SVC(kernel='linear', C=1.0, probability=True, verbose=True)
```

```
classifier.fit(X_train, y_train)
```

```
[LibSVM]
SVC
SVC(kernel='linear', probability=True, verbose=True)
```

```
y_preds = classifier.predict(X_test)
y_proba = classifier.predict_proba(X_test)
```

```
print(y_preds)    # 1d vactor.
```

```
[1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 0
 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1
 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 1 1
 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0
 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1]
```

```
print(y_proba)     # 2d vactor.
```

```
[[3.02737643e-03 9.96972624e-01]
 [9.18633556e-01 8.13664441e-02]
 [8.32311692e-06 9.99991677e-01]
 [3.19060750e-02 9.68093925e-01]
 [9.28066587e-07 9.99999072e-01]
 [9.54586789e-01 4.54132111e-02]
 [1.96117026e-01 8.03882974e-01]
 [9.97152046e-01 2.84795425e-03]
 [2.71094161e-01 7.28905839e-01]
 [3.09278404e-01 6.90721596e-01]
 [6.67386770e-03 9.93326132e-01]
 [1.69121369e-01 8.30878631e-01]
 [3.67050125e-06 9.99996329e-01]
 [7.38527308e-02 9.26147269e-01]
 [9.71706716e-02 9.02829328e-01]
 [9.93091588e-01 6.90841228e-03]
 [3.61124844e-03 9.96388752e-01]
 [1.19150625e-02 9.88084937e-01]
 [8.11452981e-07 9.99999189e-01]
 [8.87272640e-01 1.12727360e-01]
 [7.82947638e-02 9.21705236e-01]
 [4.50327433e-01 5.49672567e-01]
 [8.64097247e-01 1.35902753e-01]
 [1.90903295e-02 9.80909671e-01]
 [7.45247086e-01 2.54752914e-01]
 [9.87861349e-01 1.21386513e-02]
 [2.37531842e-02 9.76246816e-01]
 [9.99933632e-01 6.63683734e-05]
 [9.99993625e-01 6.37522729e-06]
 [9.52646125e-01 4.73538747e-02]
 [9.87992642e-01 1.20073583e-02]
 [5.05182638e-02 9.49481736e-01]
 [2.22011019e-02 9.77798898e-01]
 [2.82295358e-01 7.17704642e-01]
 [7.12979727e-02 9.28702027e-01]
 [1.69408054e-02 9.83059195e-01]
 [2.24947270e-10 1.00000000e+00]
 [3.16233198e-01 6.83766802e-01]
 [9.98006141e-01 1.99385948e-03]
 [5.39530862e-07 9.99999460e-01]
 [2.16874611e-01 7.83125389e-01]
 [1.44239116e-02 9.85576088e-01]
 [7.51102681e-03 9.92488973e-01]
 [3.51462676e-02 9.64853732e-01]
 [1.35490659e-02 9.86450934e-01]
 [6.40041054e-02 9.35995895e-01]
 [2.62412795e-03 9.97375872e-01]
 [1.24983427e-01 8.75016573e-01]
 [7.26211678e-03 9.92737883e-01]
 [1.44570708e-01 8.55429292e-01]
 [7.24400896e-03 9.92755991e-01]
 [4.42901065e-02 9.55709893e-01]
 [6.96503543e-02 9.30349646e-01]
 [1.61404295e-01 8.38595705e-01]
 [2.12588430e-01 7.87411570e-01]
 [9.99791847e-01 2.08152930e-04]
 [1.01361773e-02 9.89863823e-01]
 [9.87274866e-01 1.27251341e-02]
```

```
y_proba = y_proba[:,1].reshape((y_proba.shape[0],))
```

we need to reshape y_prova to a 1D vactor

```
y_proba[:5], y_preds[:5], y_test[:5]
```

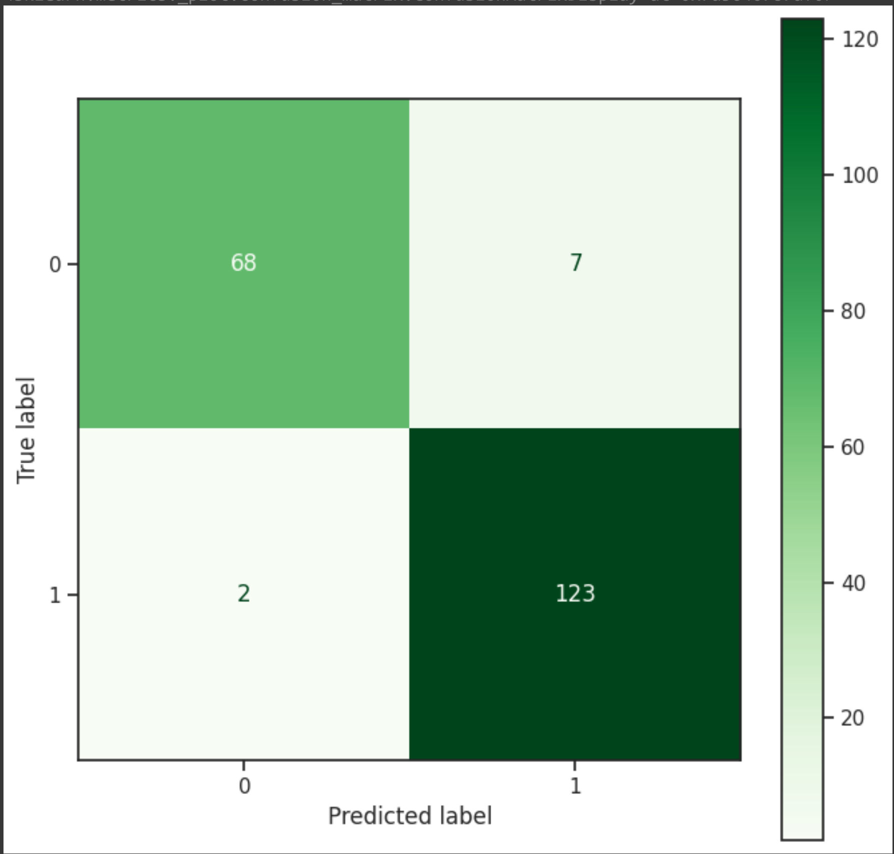
```
(array([0.99697262, 0.08136644, 0.99999168, 0.96809392, 0.99999907]),
 array([1, 0, 1, 1, 1]),
 array([1, 0, 1, 1, 1]))
```

```
conf = metrics.confusion_matrix(y_test, y_preds)
conf
```

```
array([[ 68,   7],
       [  2, 123]])
```

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test, cmap=plt.cm.Greens)
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7a3640787af0>

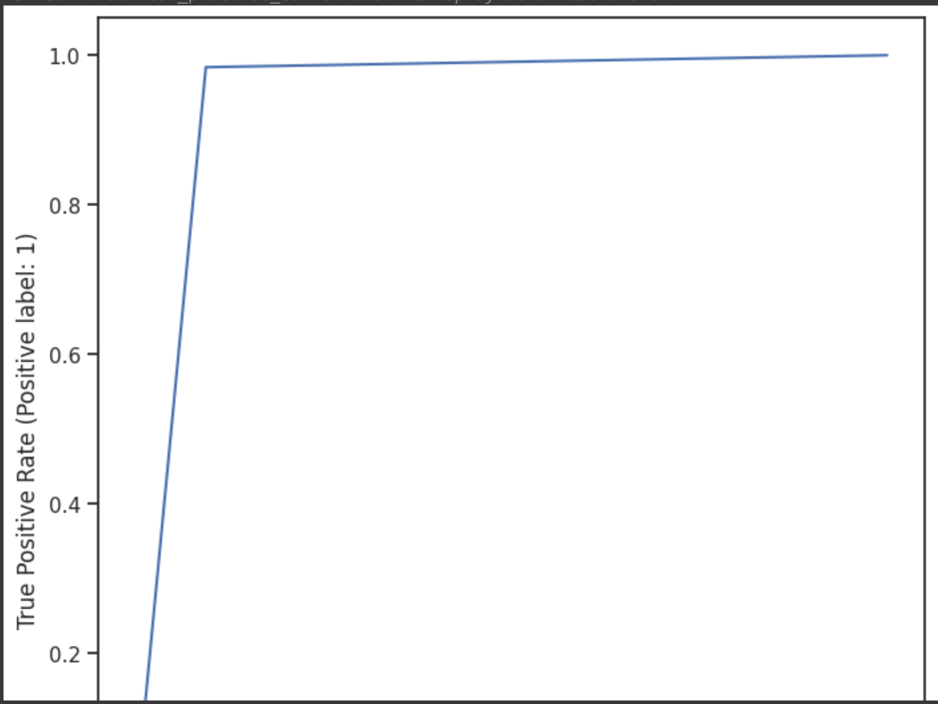



```
from sklearn.metrics import (classification_report)
print (classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        75
           1       0.95      0.98      0.96       125

    accuracy                           0.95       200
   macro avg       0.96      0.95      0.95       200
weighted avg       0.96      0.95      0.95       200
```

```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_predictions(y_test, y_preds)
```

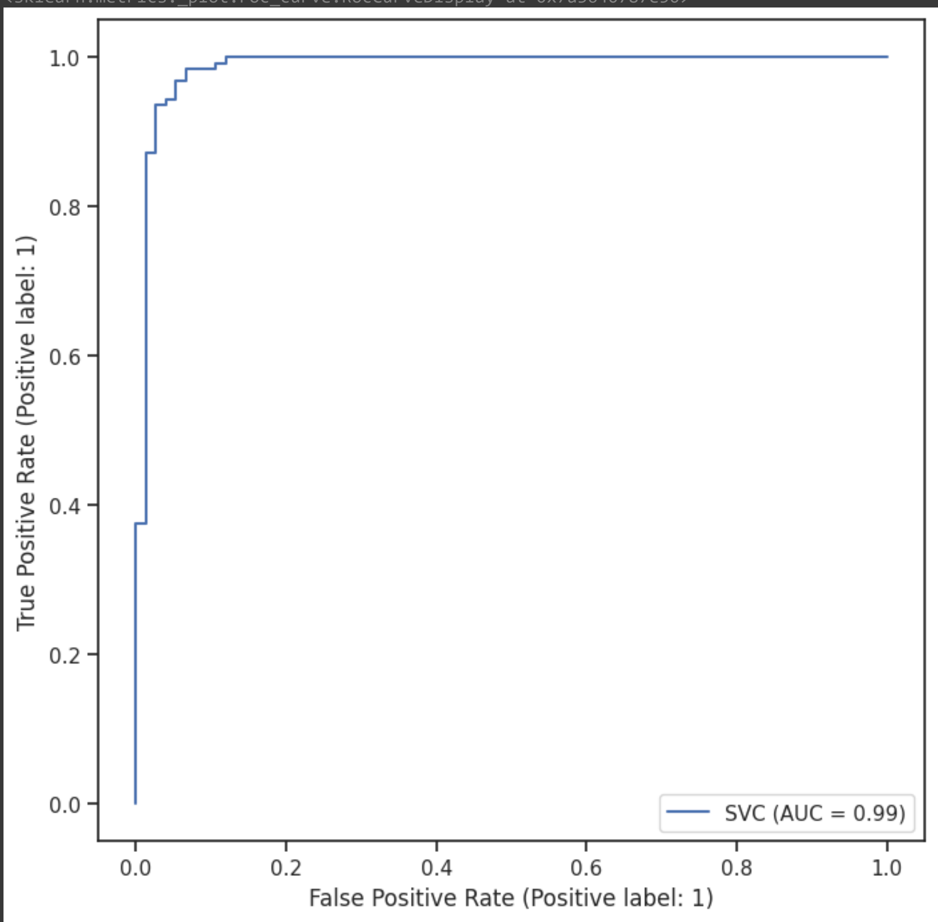
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7a3640787610>
```



Discuss: ROC curve used for Binary. ROC curve close to 1 is good.

```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(classifier, X_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7a3640787e50>
```



Discuss: ROC curve used for Binary. ROC curve close to 1 is good.