Homework 3
Code

Name: Hanumantha Rao Vakkalanka
Email: hvakkala@uncc.edu
Student ID: 801333188

Course: ECGR 5105 Intro to ML
Lab Number: Spring 2023

Problem 1

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#imports the necessary libraries for creating visualizations (matplotlib),
scaling data (StandardScaler and MinMaxScaler), performing cross-validation
(KFold and cross_val_score), and fitting a logistic regression model
(LogisticRegression)
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

#import the necessary libraries for working with datasets (datasets),
evaluating model performance (metrics, confusion_matrix,
classification_report, and precision_recall_curve), loading the breast cancer
dataset from scikit-learn (load_breast_cancer), and fitting a Gaussian Naive
Bayes model (GaussianNaiveBayes)
from sklearn import datasets
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.datasets import load_breast_cancer

from sklearn.naive_bayes import GaussianNaiveBayes
```

```python
from sklearn.metrics import precision_recall_curve

# Load breast cancer dataset from scikit-learn datasets module
breast = load_breast_cancer()

# Assign the features (data) to the variable X
X = breast.data

# Assign the target (labels) to the variable Y
Y = breast.target

# Reshape X to have 569 rows and 30 columns
X =  np.reshape(X,(569,30))

# Reshape Y to have 569 rows and 1 column
Y = np.reshape(Y,(569,1))

# Concatenate X and Y arrays along the columns axis (axis=1)
#final_breast_data = np.concatenate([X,Y],axis=1)

# Convert final_breast_data array to a pandas DataFrame (optional)
#breast_dataset = pd.DataFrame(final_breast_data)

# Append 'labels' to the feature names and store in features_labels array
#features_labels = np.append(breast.feature_names,'labels')

# Set column names of breast_dataset to features_labels
#breast_dataset.columns = features_labels

# Display first 5 rows of the DataFrame (optional)
#breast_dataset.head()


# Create an instance of the MinMaxScaler class
Min_Max_Scaling = MinMaxScaler()

# Apply Min-Max scaling to the X array and store the result in breast_dataset
breast_dataset = Min_Max_Scaling.fit_transform(X)

# Create an instance of the StandardScaler class
sc = StandardScaler()

# Apply standardization to the breast_dataset array and store the result in
breast_dataset
breast_dataset = sc.fit_transform(breast_dataset)

# Convert breast_dataset to a pandas DataFrame
breast_dataset = pd.DataFrame(breast_dataset)
```

```python
# Display the resulting DataFrame
breast_dataset
```

```
           0         1         2         3         4         5         6   \
0    1.097064 -2.073335  1.269934  0.984375  1.568466  3.283515  2.652874   
1    1.829821 -0.353632  1.685955  1.908708 -0.826962 -0.487072 -0.023846   
2    1.579888  0.456187  1.566503  1.558884  0.942210  1.052926  1.363478   
3   -0.768909  0.253732 -0.592687 -0.764464  3.283553  3.402909  1.915897   
4    1.750297 -1.151816  1.776573  1.826229  0.280372  0.539340  1.371011   
..        ...       ...       ...       ...       ...       ...       ...   
564  2.110995  0.721473  2.060786  2.343856  1.041842  0.219060  1.947285   
565  1.704854  2.085134  1.615931  1.723842  0.102458 -0.017833  0.693043   
566  0.702284  2.045574  0.672676  0.577953 -0.840484 -0.038680  0.046588   
567  1.838341  2.336457  1.982524  1.735218  1.525767  3.272144  3.296944   
568 -1.808401  1.221792 -1.814389 -1.347789 -3.112085 -1.150752 -1.114873   

           7         8         9   ...        20        21        22  \
0    2.532475  2.217515  2.255747  ...  1.886690 -1.359293  2.303601   
1    0.548144  0.001392 -0.868652  ...  1.805927 -0.369203  1.535126   
2    2.037231  0.939685 -0.398008  ...  1.511870 -0.023974  1.347475   
3    1.451707  2.867383  4.910919  ... -0.281464  0.133984 -0.249939   
4    1.428493 -0.009560 -0.562450  ...  1.298575 -1.466770  1.338539   
..        ...       ...       ...  ...       ...       ...       ...   
564  2.320965 -0.312589 -0.931027  ...  1.901185  0.117700  1.752563   
565  1.263669 -0.217664 -1.058611  ...  1.536720  2.047399  1.421940   
566  0.105777 -0.809117 -0.895587  ...  0.561361  1.374854  0.579001   
567  2.658866  2.137194  1.043695  ...  1.961239  2.237926  2.303601   
568 -1.261820 -0.820070 -0.561032  ... -1.410893  0.764190 -1.432735   

           23        24        25        26        27        28        29  
0    2.001237  1.307686  2.616665  2.109526  2.296076  2.750622  1.937015  
1    1.890489 -0.375612 -0.430444 -0.146749  1.087084 -0.243890  0.281190  
2    1.456285  0.527407  1.082932  0.854974  1.955000  1.152255  0.201391  
3   -0.550021  3.394275  3.893397  1.989588  2.175786  6.046041  4.935010  
4    1.220724  0.220556 -0.313395  0.613179  0.729259 -0.868353 -0.397100  
..        ...       ...       ...       ...       ...       ...       ...  
564  2.015301  0.378365 -0.273318  0.664512  1.629151 -1.360158 -0.709091  
565  1.494959 -0.691230 -0.394820  0.236573  0.733827 -0.531855 -0.973978  
566  0.427906 -0.809587  0.350735  0.326767  0.414069 -1.104549 -0.318409  
567  1.653171  1.430427  3.904848  3.197605  2.289985  1.919083  2.219635  
568 -1.075813 -1.859019 -1.207552 -1.305831 -1.745063 -0.048138 -0.751207  

[569 rows x 30 columns]
```

```python
# Concatenate the standardized breast_dataset and Y arrays along the columns axis (axis=1)
final_breast_data = np.concatenate([breast_dataset,Y],axis=1)

# Convert final_breast_data to a pandas DataFrame
breast_dataset = pd.DataFrame(final_breast_data)
```

```
# Append the string 'labels' to the end of the array breast.feature_names
features_labels = np.append(breast.feature_names,'labels')

# Assign the resulting array as the column labels of the breast_dataset
DataFrame
breast_dataset.columns = features_labels

# Display the first few rows of the resulting DataFrame
breast_dataset.head()

   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0     1.097064     -2.073335        1.269934   0.984375         1.568466
1     1.829821     -0.353632        1.685955   1.908708        -0.826962
2     1.579888      0.456187        1.566503   1.558884         0.942210
3    -0.768909      0.253732       -0.592687  -0.764464         3.283553
4     1.750297     -1.151816        1.776573   1.826229         0.280372

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0          3.283515        2.652874             2.532475       2.217515
1         -0.487072       -0.023846             0.548144       0.001392
2          1.052926        1.363478             2.037231       0.939685
3          3.402909        1.915897             1.451707       2.867383
4          0.539340        1.371011             1.428493      -0.009560

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                2.255747  ...      -1.359293         2.303601    2.001237
1               -0.868652  ...      -0.369203         1.535126    1.890489
2               -0.398008  ...      -0.023974         1.347475    1.456285
3                4.910919  ...       0.133984        -0.249939   -0.550021
4               -0.562450  ...      -1.466770         1.338539    1.220724

   worst smoothness  worst compactness  worst concavity  worst concave points
\
0          1.307686           2.616665         2.109526              2.296076
1         -0.375612          -0.430444        -0.146749              1.087084
2          0.527407           1.082932         0.854974              1.955000
3          3.394275           3.893397         1.989588              2.175786
4          0.220556          -0.313395         0.613179              0.729259

   worst symmetry  worst fractal dimension  labels
0        2.750622                 1.937015     0.0
1       -0.243890                 0.281190     0.0
2        1.152255                 0.201391     0.0
3        6.046041                 4.935010     0.0
4       -0.868353                -0.397100     0.0

[5 rows x 31 columns]

# Randomly select 80% of the rows in the breast_dataset DataFrame and assign
them to the train variable
```

```python
train=breast_dataset.sample(frac=0.8,random_state=0)

# Drop the rows in the test DataFrame that were selected for the train
DataFrame
test=breast_dataset.drop(train.index)

# Extract the values of the first 29 columns of the train and test DataFrames
and assign them to X_train and X_test, respectively
X_train = train.values[:,0:29]
X_test = test.values[:,0:29]

# Extract the values of the 'labels' column of the train and test DataFrames
and assign them to Y_train and Y_test, respectively
Y_train = train.values[:,30]
Y_test = test.values[:,30]

# Display the values of the 'labels' column of the test DataFrame
Y_test
```

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

```python
# Create a new instance of the Gaussian Naive Bayes classifier and assign it
to the variable 'model'
model = GaussianNaiveBayes()

# Train the Gaussian Naive Bayes classifier on the training data by calling
the 'fit()' method of the 'model' object, passing in X_train and Y_train as
arguments
model.fit(X_train, Y_train)
```

```
GaussianNaiveBayes()
```

```python
# Use the 'predict()' method of the 'model' object to predict the class
labels of the test data, passing in X_test as the argument
Y_predicted = model.predict(X_test)

# Print the predicted class labels of the test data
print(Y_predicted)
```

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

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

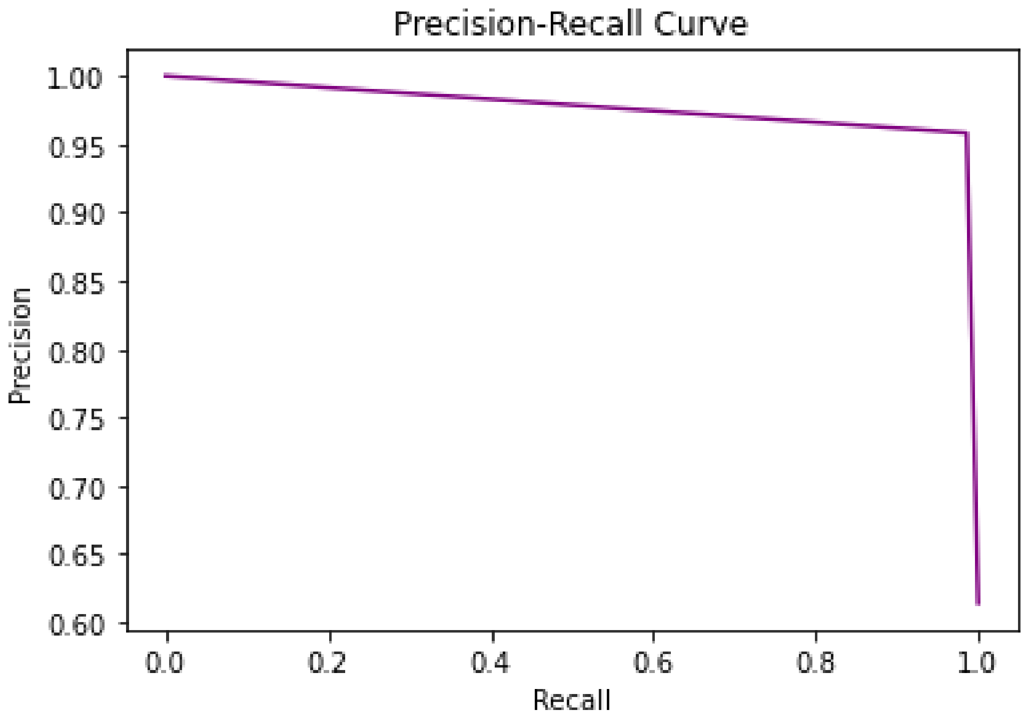
```python
# Use the 'classification_report()' function from the 'metrics' module to
generate a report of the classification performance of the model on the test
data, passing in Y_test and Y_predicted as arguments
report = metrics.classification_report(Y_test, Y_predicted)

# Print the classification report to the console
print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.93      0.95        44
         1.0       0.96      0.99      0.97        70

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

```python
# Use the 'confusion_matrix()' function from the 'metrics' module to generate
a confusion matrix of the classification performance of the model on the test
data, passing in Y_test and Y_predicted as arguments
matrix = metrics.confusion_matrix(Y_test, Y_predicted)

# Print the confusion matrix to the console
print(matrix)
```

```
[[41  3]
 [ 1 69]]
```

```python
#precision and recall
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#precision-recall curve is plotted using ax.plot(), with recall on the x-axis
and precision on the y-axis
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```



Problem 2

```python
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#imports the necessary libraries for creating visualizations (matplotlib),
scaling data (StandardScaler and MinMaxScaler), performing cross-validation
(KFold and cross_val_score), and fitting a logistic regression model
(LogisticRegression)
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

#import the necessary libraries for working with datasets (datasets),
evaluating model performance (metrics, confusion_matrix,
classification_report, and precision_recall_curve), loading the breast cancer
dataset from scikit-learn (load_breast_cancer), and fitting a Gaussian Naive
Bayes model (GaussianNaiveBayes)
from sklearn import datasets
```

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

from sklearn.datasets import load_breast_cancer

from sklearn.naive_bayes import GaussianNaiveBayes

from sklearn.metrics import precision_recall_curve

# Load breast cancer dataset from scikit-learn datasets module
breast = load_breast_cancer()

# Assign the features (data) to the variable X
X = breast.data

# Assign the target (labels) to the variable Y
Y = breast.target

# Reshape X to have 569 rows and 30 columns
X =  np.reshape(X,(569,30))

# Reshape Y to have 569 rows and 1 column
Y = np.reshape(Y,(569,1))

#create a pandas DataFrame object Dataflow from the target variable Y of the
breast cancer dataset
Dataflow = pd.DataFrame(Y)

# Importing PCA module from Scikit-learn's decomposition library
from sklearn.decomposition import PCA

# Initializing PCA with number of components as 2
pca = PCA(n_components=2)

# Applying PCA on the given data 'X' and getting the principal components
PrnComp = pca.fit_transform(X)

# Converting the principal components into a Pandas DataFrame
principalDataflow = pd.DataFrame(data = PrnComp
 , columns = ['principal component 1', 'principal component 2'])

# Concatenating the principalDataflow and Dataflow DataFrames horizontally
finalDataflow1 = pd.concat([principalDataflow, Dataflow], axis = 1)

# Printing the concatenated DataFrame
finalDataflow1

     principal component 1  principal component 2  0
0              1160.142574            -293.917544  0
1              1269.122443              15.630182  0
```

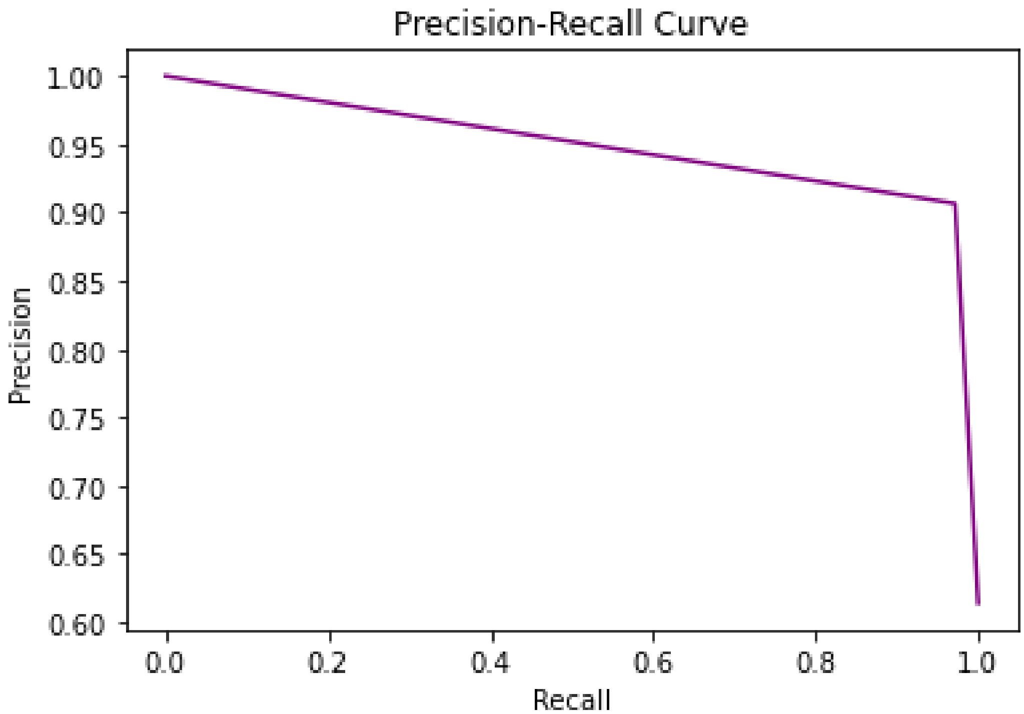
```
2              995.793889             39.156743  0
3             -407.180803            -67.380320  0
4              930.341180            189.340742  0
..                    ...                   ... ..
564           1414.126684            110.222492  0
565           1045.018854             77.057589  0
566            314.501756             47.553525  0
567           1124.858115             34.129225  0
568           -771.527622            -88.643106  1

[569 rows x 3 columns]
```

```python
# Creating a random training set of 80% of the data
train = finalDataflow1.sample(frac=0.8, random_state=0)

# Creating a test set containing the remaining 20% of the data
test = finalDataflow1.drop(train.index)

# Creating X_train and X_test datasets containing only the first principal
component
X_train = train.values[:,0:1]
X_test = test.values[:,0:1]

# Creating Y_train and Y_test datasets containing the target variable
#Y_train = train.values[:,2]
#Y_test = test.values[:,2]

# Creating a Logistic Regression model
model = LogisticRegression()

# Fitting the model with the training data
model.fit(X_train, Y_train)

# Predicting the target variable for the test data using the fitted model
Y_predicted = model.predict(X_test)

# Printing the classification report of the model's performance on the test
data
print(metrics.classification_report(Y_test, Y_predicted))

# Printing the confusion matrix of the model's performance on the test data
print(metrics.confusion_matrix(Y_test, Y_predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.84      0.89        44
         1.0       0.91      0.97      0.94        70

    accuracy                           0.92       114
   macro avg       0.93      0.91      0.91       114
```

```
weighted avg       0.92      0.92      0.92       114

[[37  7]
 [ 2 68]]
```

```python
# Calculate precision, recall, and thresholds using the precision_recall_curve function from the metrics module
precision, recall, thresholds = precision_recall_curve(Y_test, Y_predicted)

# Create a new figure and axis objects using the subplots method from the pyplot module
fig, ax = plt.subplots()

# Plot the precision-recall curve with recall on the x-axis and precision on the y-axis using the plot method of the axis object
ax.plot(recall, precision, color='purple')

# Add a title to the plot using the set_title method of the axis object
ax.set_title('Precision-Recall Curve')

# Add a label to the y-axis using the set_ylabel method of the axis object
ax.set_ylabel('Precision')

# Add a label to the x-axis using the set_xlabel method of the axis object
ax.set_xlabel('Recall')

# Display the plot using the show method of the pyplot module
plt.show()
```




```python
# Import the PCA class from Scikit-Learn's decomposition module
from sklearn.decomposition import PCA

# Create a new PCA object with 6 principal components
pca = PCA(n_components=6)

# Fit the PCA model to the input data X and transform the data into the new principal component space
PrnComp = pca.fit_transform(X)

# Create a new DataFrame called principalDataflow to hold the principal components, with column names for each component
principalDataflow = pd.DataFrame(data=PrnComp, columns=['principal component 1', 'principal component 2', 'principal component 3', 'principal component 4', 'principal component 5', 'principal component 6'])

# Concatenate the principal component DataFrame (principalDataflow) with the original data DataFrame (Dataflow) along the columns (axis=1)
finalDataflow2 = pd.concat([principalDataflow, Dataflow], axis=1)

# Print the final concatenated DataFrame
finalDataflow2
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
```

```
3             -407.180803             -67.380320              8.672848
4              930.341180             189.340742              1.374801
..                    ...                    ...                   ...
564           1414.126684             110.222492             40.065944
565           1045.018854              77.057589              0.036669
566            314.501756              47.553525            -10.442407
567           1124.858115              34.129225            -19.742087
568           -771.527622             -88.643106             23.889032

     principal component 4  principal component 5  principal component 6  0
0                -8.711975              32.000486               1.265415  0
1                17.861283              -4.334874              -0.225872  0
2                 4.199340              -0.466529              -2.652811  0
3               -11.759867               7.115461               1.299436  0
4                 8.499183               7.613289               1.021160  0
..                     ...                    ...                    ... ..
564               6.562240              -5.102856              -0.395424  0
565              -4.753245             -12.417863              -0.059637  0
566              -9.771881              -6.156213              -0.870726  0
567             -23.660881               3.565133               4.086390  0
568               2.547249             -14.717566               4.418123  1

[569 rows x 7 columns]
```

```
# Select a random sample of 80% of the data for training, with a fixed random state for reproducibility
training2 = finalDataflow2.sample(frac=0.8, random_state=0)

# Select the remaining 20% of the data for testing by dropping the rows that were selected for training
testing2 = finalDataflow2.drop(train.index)

# Extract the input features (principal components) for the training and test sets
X_training2 = training2.values[:,0:5]
X_testing2 = testing2.values[:,0:5]

# Extract the output labels (species) for the training and test sets
Y_training2 = training2.values[:,6]
Y_testing2 = testing2.values[:,6]
```

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

```python
# Create a new logistic regression model
model = LogisticRegression()

# Fit the logistic regression model to the training data
model.fit(X_training2, Y_training2)
```

```
LogisticRegression()
```

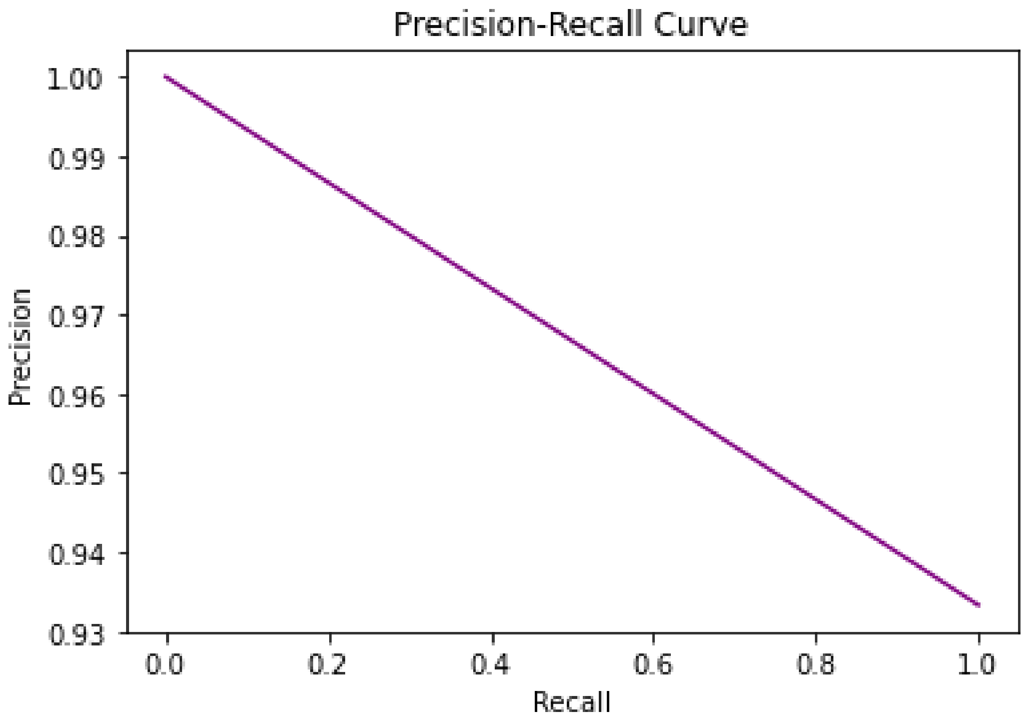
```python
# Predict the output labels for the test data using the trained model
Y_predicted = model.predict(X_testing2)
Y_predicted
```

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

```python
# Print the classification report for the logistic regression model
print(metrics.classification_report(Y_test, Y_predicted))

# Print the confusion matrix for the logistic regression model
print(metrics.confusion_matrix(Y_test,Y_predicted))

# Calculate the precision, recall, and thresholds for the logistic regression model
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)

# Create a new figure and axis for the precision-recall curve
fig, ax = plt.subplots()

# Plot the precision and recall values as a curve
ax.plot(recall, precision, color='purple')

# Set the title, x-label, and y-label for the plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

# Show the plot
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.89      0.94        44
         1.0       0.93      1.00      0.97        70

    accuracy                           0.96       114
```

```
   macro avg       0.97      0.94      0.95       114
weighted avg       0.96      0.96      0.96       114

[[39  5]
 [ 0 70]]
```




```
#Doing the LogisicReg for 18 imp components

from sklearn.decomposition import PCA

#Import the PCA class from scikit-learn.
pca = PCA(n_components=18)

#Create a PCA object with 18 principal components.
PrnComp = pca.fit_transform(X)

#Fit the PCA object to the input data X and transform the data to get the
principal components.
principalDataflow = pd.DataFrame(data = PrnComp
 , columns = ['principal component 1', 'principal component 2','principal
component 3',
              'principal component 4', 'principal component 5','principal
component 6',
              'principal component 7','principal component 8', 'principal
component 9',
              'principal component 10', 'principal component 11','principal
component 12',
```

```python
              'principal component 13','principal component 14', 'principal
component 15',
              'principal component 16','principal component 17', 'principal
component 18'])


#Create a pandas DataFrame principalDataflow to store the principal
components with column names corresponding to the component number
finalDataflow3 = pd.concat([principalDataflow, Dataflow], axis = 1)

#Concatenate the principalDataflow DataFrame with the original DataFrame
Dataflow along the column axis
finalDataflow3
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
3              -407.180803             -67.380320               8.672848
4               930.341180             189.340742               1.374801
..                     ...                    ...                    ...
564            1414.126684             110.222492              40.065944
565            1045.018854              77.057589               0.036669
566             314.501756              47.553525             -10.442407
567            1124.858115              34.129225             -19.742087
568            -771.527622             -88.643106              23.889032

     principal component 4  principal component 5  principal component 6  \
0                -8.711975              32.000486               1.265415
1                17.861283              -4.334874              -0.225872
2                 4.199340              -0.466529              -2.652811
3               -11.759867               7.115461               1.299436
4                 8.499183               7.613289               1.021160
..                     ...                    ...                    ...
564               6.562240              -5.102856              -0.395424
565              -4.753245             -12.417863              -0.059637
566              -9.771881              -6.156213              -0.870726
567             -23.660881               3.565133               4.086390
568               2.547249             -14.717566               4.418123

     principal component 7  principal component 8  principal component 9  \
0                 0.931337               0.148167               0.745463
1                -0.046037               0.200804              -0.485828
2                -0.779745              -0.274026              -0.173874
3                -1.267304              -0.060555              -0.330639
4                -0.335522               0.289109               0.036087
..                     ...                    ...                    ...
564              -0.786751               0.037082              -0.452530
565               0.449831               0.509154              -0.449986
566              -2.166493              -0.442279              -0.097398
```

```
567            -1.705401             -0.359964             0.385030
568            -2.815752              0.030039            -0.423451

     principal component 10  principal component 11  principal component 12
\
0                  0.589359               -0.307804                0.043452
1                 -0.084035                0.080642                0.033042
2                 -0.186994                0.279174               -0.020464
3                 -0.144155                0.927471               -0.174720
4                 -0.138502                0.042228               -0.062721
..                      ...                     ...                     ...
564               -0.235185                0.163649                0.052543
565                0.493247                0.007625                0.055832
566               -0.144667               -0.109147                0.076263
567                0.615467                0.307166               -0.028224
568               -0.301439                0.133353               -0.115105

     principal component 13  principal component 14  principal component 15
\
0                  0.034777                0.065069               -0.012934
1                  0.045485               -0.005534                0.021368
2                  0.083505                0.024824               -0.026887
3                  0.282556                0.080057                0.043201
4                 -0.114247                0.002274               -0.019548
..                      ...                     ...                     ...
564               -0.075032               -0.015211               -0.061390
565               -0.015163                0.009985                0.003312
566               -0.004448               -0.055285               -0.012459
567                0.060561               -0.037742               -0.031873
568               -0.019667                0.013734               -0.004134

     principal component 16  principal component 17  principal component 18
0
0                 -0.002670                0.018300                0.010263
0
1                 -0.028715                0.012371               -0.006009
0
2                 -0.041255                0.008218               -0.028044
0
3                 -0.034175                0.033742               -0.016965
0
4                  0.019932               -0.019201                0.004024
0
..                      ...                     ...                     ...
..
564               -0.054694               -0.004829               -0.011515
0
565               -0.020654                0.005197                0.002106
0
566               -0.005414                0.007866               -0.004484
```

```
0
567                   0.020126                0.015243                0.043651
0
568                   0.034264                0.009440               -0.028323
1

[569 rows x 19 columns]
```

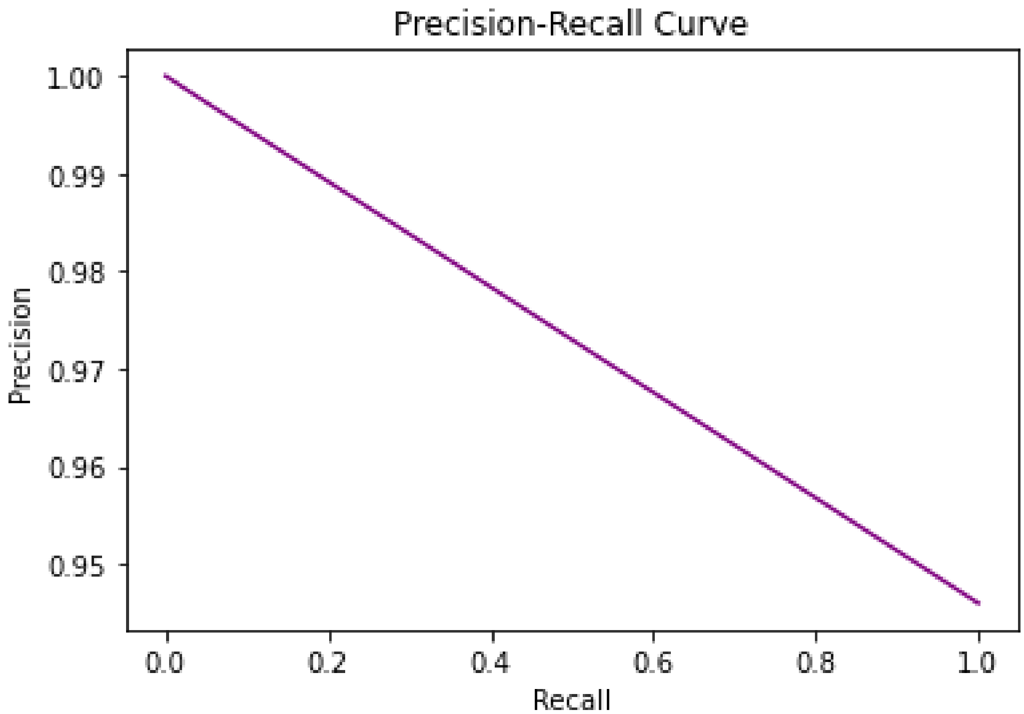
```
# Select a random sample of 80% of the data for training, with a fixed random state for reproducibility
training3=finalDataflow3.sample(frac=0.8,random_state=0)

## Select the remaining 20% of the data for testing by dropping the rows that were selected for training
testing3=finalDataflow3.drop(train.index)

# Extract the input features (principal components) for the training and test sets
X_training3 = training3.values[:,0:17]
X_testing3 = testing3.values[:,0:17]

# Extract the output labels (species) for the training and test sets
#Y_train = np.array(train.labels)
#Y_test = np.array(test.labels)
Y_training3 = training3.values[:,18]
Y_testing3 = testing3.values[:,18]
Y_testing3

# Create a new logistic regression model
model.fit(X_training3,Y_training3)

# Predict the output labels for the test data using the trained model
Y_predicted = model.predict(X_testing3)
Y_predicted
```

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

```
# Print the classification report for the logistic regression model
print(metrics.classification_report(Y_test, Y_predicted))

# Print the confusion matrix for the logistic regression model
print(metrics.confusion_matrix(Y_test,Y_predicted))
```

```python
# Calculate the precision, recall, and thresholds for the logistic regression
model
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)

# Create a new figure and axis for the precision-recall curve
fig, ax = plt.subplots()

# Plot the precision and recall values as a curve
ax.plot(recall, precision, color='purple')

# Set the title, x-label, and y-label for the plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')


# Show the plot
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.91      0.95        44
         1.0       0.95      1.00      0.97        70

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114

[[40  4]
 [ 0 70]]
```



Problem 3

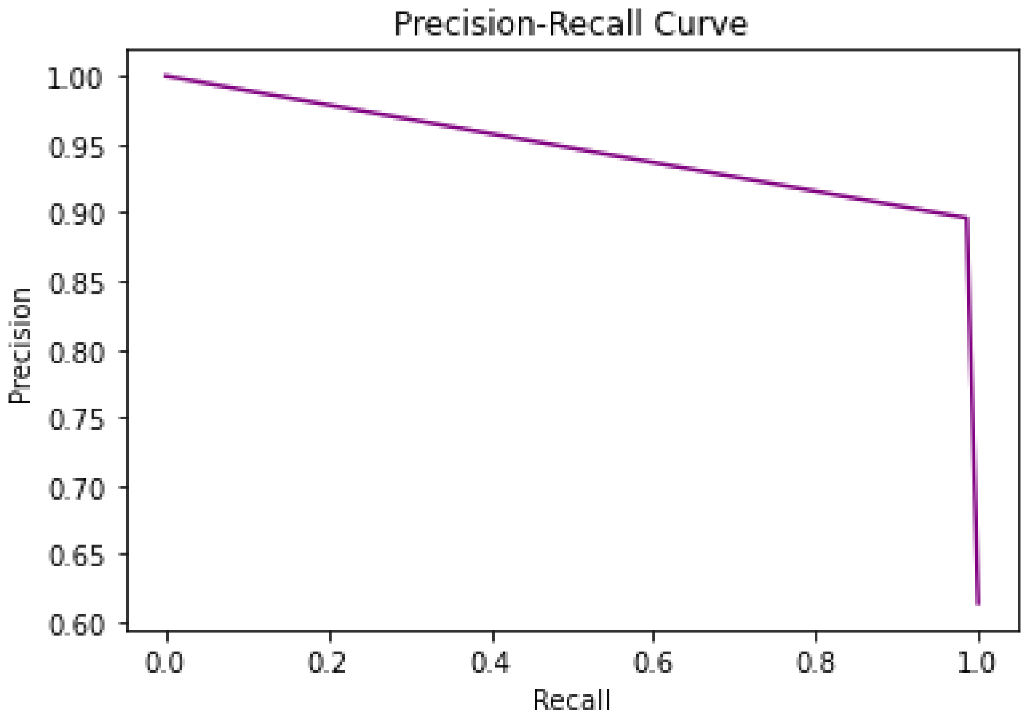
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#imports the necessary libraries for creating visualizations (matplotlib),
scaling data (StandardScaler and MinMaxScaler), performing cross-validation
(KFold and cross_val_score), and fitting a logistic regression model
(LogisticRegression)
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

#import the necessary libraries for working with datasets (datasets),
evaluating model performance (metrics, confusion_matrix,
classification_report, and precision_recall_curve), loading the breast cancer
```

```python
dataset from scikit-learn (load_breast_cancer), and fitting a Gaussian Naive
Bayes model (GaussianNaiveBayes)
from sklearn import datasets
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.datasets import load_breast_cancer

from sklearn.naive_bayes import GaussianNaiveBayes

from sklearn.metrics import precision_recall_curve

# Load breast cancer dataset from scikit-learn datasets module
breast = load_breast_cancer()

# Assign the features (data) to the variable X
X = breast.data

# Assign the target (labels) to the variable Y
Y = breast.target

# Reshape X to have 569 rows and 30 columns
X =  np.reshape(X,(569,30))

# Reshape Y to have 569 rows and 1 column
Y = np.reshape(Y,(569,1))

#create a pandas DataFrame object Dataflow from the target variable Y of the
breast cancer dataset
Dataflow = pd.DataFrame(Y)

#Doing the GaussianNaiveBayes for 2 imp components

# Importing PCA module from Scikit-learn's decomposition library
from sklearn.decomposition import PCA

# Initializing PCA with number of components as 2
pca = PCA(n_components=2)

# Applying PCA on the given data 'X' and getting the principal components
PrnComp = pca.fit_transform(X)

# Converting the principal components into a Pandas DataFrame
principalDataflow = pd.DataFrame(data = PrnComp
 , columns = ['principal component 1', 'principal component 2'])



# Concatenating the principalDataflow and Dataflow DataFrames horizontally
finalDataflow1 = pd.concat([principalDataflow, Dataflow], axis = 1)
```

```
# Printing the concatenated DataFrame
finalDataflow1

     principal component 1  principal component 2  0
0              1160.142574            -293.917544  0
1              1269.122443              15.630182  0
2               995.793889              39.156743  0
3              -407.180803             -67.380320  0
4               930.341180             189.340742  0
..                     ...                    ... ..
564            1414.126684             110.222492  0
565            1045.018854              77.057589  0
566             314.501756              47.553525  0
567            1124.858115              34.129225  0
568            -771.527622             -88.643106  1

[569 rows x 3 columns]

# Creating a random training set of 80% of the data
train = finalDataflow1.sample(frac=0.8, random_state=0)

# Creating a test set containing the remaining 20% of the data
test = finalDataflow1.drop(train.index)

# Creating X_train and X_test datasets containing only the first principal
component
X_train = train.values[:,0:1]
X_test = test.values[:,0:1]

# Creating Y_train and Y_test datasets containing the target variable
#Y_train = train.values[:,2]
#Y_test = test.values[:,2]

# Creating a Logistic Regression model
model = LogisticRegression()

# Fitting the model with the training data
model.fit(X_train, Y_train)

# Predicting the target variable for the test data using the fitted model
Y_predicted = model.predict(X_test)

# Printing the classification report of the model's performance on the test
data
print(metrics.classification_report(Y_test, Y_predicted))

# Printing the confusion matrix of the model's performance on the test data
print(metrics.confusion_matrix(Y_test, Y_predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.82      0.89        44
         1.0       0.90      0.99      0.94        70

    accuracy                           0.92       114
   macro avg       0.93      0.90      0.91       114
weighted avg       0.93      0.92      0.92       114

[[36  8]
 [ 1 69]]
```

```
# Calculate precision, recall, and thresholds using the precision_recall_curve function from the metrics module
precision, recall, thresholds = precision_recall_curve(Y_test, Y_predicted)

# Create a new figure and axis objects using the subplots method from the pyplot module
fig, ax = plt.subplots()

# Plot the precision-recall curve with recall on the x-axis and precision on the y-axis using the plot method of the axis object
ax.plot(recall, precision, color='purple')

# Add a title to the plot using the set_title method of the axis object
ax.set_title('Precision-Recall Curve')

# Add a label to the y-axis using the set_ylabel method of the axis object
ax.set_ylabel('Precision')

# Add a label to the x-axis using the set_xlabel method of the axis object
ax.set_xlabel('Recall')

# Display the plot using the show method of the pyplot module
plt.show()
```




```
# Import the PCA class from Scikit-Learn's decomposition module
from sklearn.decomposition import PCA

# Create a new PCA object with 6 principal components
pca = PCA(n_components=6)

# Fit the PCA model to the input data X and transform the data into the new
principal component space
PrnComp = pca.fit_transform(X)

# Create a new DataFrame called principalDataflow to hold the principal
components, with column names for each component
principalDataflow = pd.DataFrame(data=PrnComp, columns=['principal component
1', 'principal component 2', 'principal component 3', 'principal component
4', 'principal component 5', 'principal component 6'])

# Concatenate the principal component DataFrame (principalDataflow) with the
original data DataFrame (Dataflow) along the columns (axis=1)
finalDataflow2 = pd.concat([principalDataflow, Dataflow], axis=1)

# Print the final concatenated DataFrame
finalDataflow2
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
```

```
3          -407.180803          -67.380320           8.672848
4           930.341180          189.340742           1.374801
..                 ...                 ...                ...
564        1414.126684          110.222492          40.065944
565        1045.018854           77.057589           0.036669
566         314.501756           47.553525         -10.442407
567        1124.858115           34.129225         -19.742087
568        -771.527622          -88.643106          23.889032

     principal component 4  principal component 5  principal component 6  0
0                -8.711975              32.000486               1.265415  0
1                17.861283              -4.334874              -0.225872  0
2                 4.199340              -0.466529              -2.652811  0
3               -11.759867               7.115461               1.299436  0
4                 8.499183               7.613289               1.021160  0
..                     ...                    ...                    ... ..
564               6.562240              -5.102856              -0.395424  0
565              -4.753245             -12.417863              -0.059637  0
566              -9.771881              -6.156213              -0.870726  0
567             -23.660881               3.565133               4.086390  0
568               2.547249             -14.717566               4.418123  1

[569 rows x 7 columns]
```

```python
# Select a random sample of 80% of the data for training, with a fixed random state for reproducibility
training2 = finalDataflow2.sample(frac=0.8, random_state=0)

# Select the remaining 20% of the data for testing by dropping the rows that were selected for training
testing2 = finalDataflow2.drop(train.index)

# Extract the input features (principal components) for the training and test sets
X_training2 = training2.values[:,0:5]
X_testing2 = testing2.values[:,0:5]

# Extract the output labels (species) for the training and test sets
Y_training2 = training2.values[:,6]
Y_testing2 = testing2.values[:,6]
Y_testing2
```

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

```python
# Create a new Gaussian model
model = GaussianNaiveBayes()

# Fit the Gaussian model to the training data
model.fit(X_training2,Y_training2)
```

```
GaussianNaiveBayes()
```

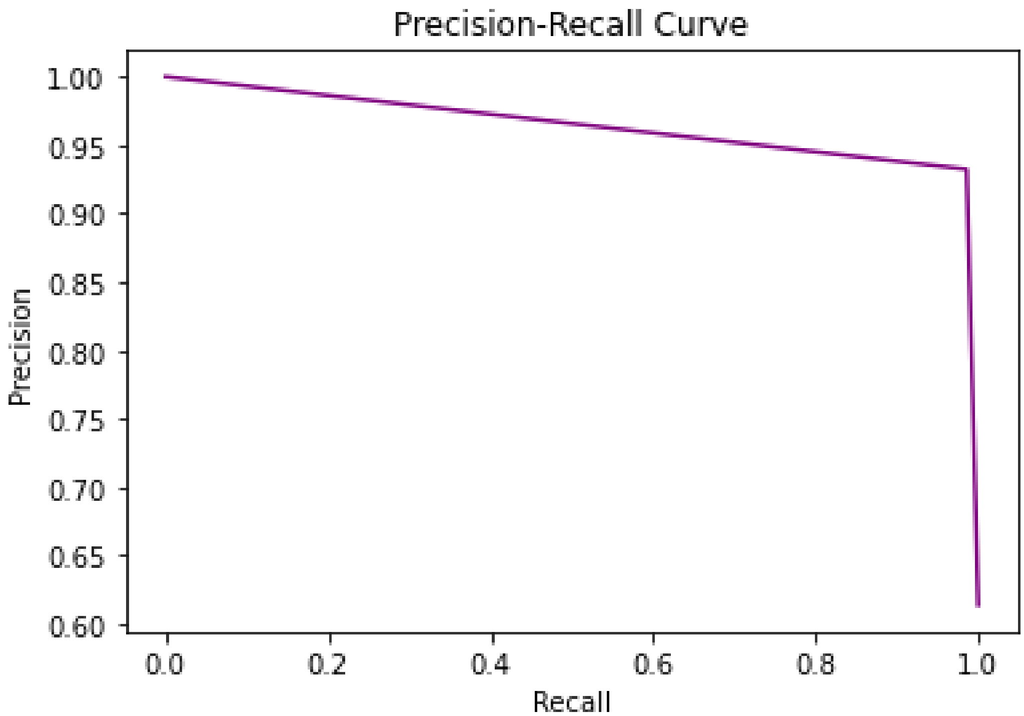
```python
# Predict the output labels for the test data using the trained model
Y_predicted = model.predict(X_testing2)
Y_predicted
```

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

```python
# Print the classification report for the logistic regression model
print(metrics.classification_report(Y_test, Y_predicted))

# Print the confusion matrix for the logistic regression model
print(metrics.confusion_matrix(Y_test,Y_predicted))

# Calculate the precision, recall, and thresholds for the logistic regression
model
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)

# Create a new figure and axis for the precision-recall curve
fig, ax = plt.subplots()

# Plot the precision and recall values as a curve
ax.plot(recall, precision, color='purple')

# Set the title, x-label, and y-label for the plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

# Show the plot
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.89      0.93        44
         1.0       0.93      0.99      0.96        70

    accuracy                           0.95       114
```

```
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

[[39  5]
 [ 1 69]]
```




```
#Doing the GaussianNaiveBayes for 18 imp components
from sklearn.decomposition import PCA

#Import the PCA class from scikit-learn.
pca = PCA(n_components=18)

#Create a PCA object with 18 principal components.
PrnComp = pca.fit_transform(X)

#Fit the PCA object to the input data X and transform the data to get the
principal components.
principalDataflow = pd.DataFrame(data = PrnComp
 , columns = ['principal component 1', 'principal component 2','principal
component 3',
              'principal component 4', 'principal component 5','principal
component 6',
              'principal component 7','principal component 8', 'principal
component 9',
              'principal component 10', 'principal component 11','principal
component 12',
              'principal component 13','principal component 14', 'principal
```

```
component 15',
              'principal component 16','principal component 17', 'principal
component 18'])


#Create a pandas DataFrame principalDataflow to store the principal
components with column names corresponding to the component number
finalDataflow3 = pd.concat([principalDataflow, Dataflow], axis = 1)

#Concatenate the principalDataflow DataFrame with the original DataFrame
Dataflow along the column axis
finalDataflow3
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
3              -407.180803             -67.380320               8.672848
4               930.341180             189.340742               1.374801
..                     ...                    ...                    ...
564            1414.126684             110.222492              40.065944
565            1045.018854              77.057589               0.036669
566             314.501756              47.553525             -10.442407
567            1124.858115              34.129225             -19.742087
568            -771.527622             -88.643106              23.889032

     principal component 4  principal component 5  principal component 6  \
0                -8.711975              32.000486               1.265415
1                17.861283              -4.334874              -0.225872
2                 4.199340              -0.466529              -2.652811
3               -11.759867               7.115461               1.299436
4                 8.499183               7.613289               1.021160
..                     ...                    ...                    ...
564               6.562240              -5.102856              -0.395424
565              -4.753245             -12.417863              -0.059637
566              -9.771881              -6.156213              -0.870726
567             -23.660881               3.565133               4.086390
568               2.547249             -14.717566               4.418123

     principal component 7  principal component 8  principal component 9  \
0                 0.931337               0.148167               0.745463
1                -0.046037               0.200804              -0.485828
2                -0.779745              -0.274026              -0.173874
3                -1.267304              -0.060555              -0.330639
4                -0.335522               0.289109               0.036087
..                     ...                    ...                    ...
564              -0.786751               0.037082              -0.452530
565               0.449831               0.509154              -0.449986
566              -2.166493              -0.442279              -0.097398
567              -1.705401              -0.359964               0.385030
```

```
568             -2.815752              0.030039            -0.423451

     principal component 10  principal component 11  principal component 12
\
0                  0.589359               -0.307804                0.043452
1                 -0.084035                0.080642                0.033042
2                 -0.186994                0.279174               -0.020464
3                 -0.144155                0.927471               -0.174720
4                 -0.138502                0.042228               -0.062721
..                      ...                     ...                     ...
564               -0.235185                0.163649                0.052543
565                0.493247                0.007625                0.055832
566               -0.144667               -0.109147                0.076263
567                0.615467                0.307166               -0.028224
568               -0.301439                0.133353               -0.115105

     principal component 13  principal component 14  principal component 15
\
0                  0.034777                0.065069               -0.012934
1                  0.045485               -0.005534                0.021368
2                  0.083505                0.024824               -0.026887
3                  0.282556                0.080057                0.043201
4                 -0.114247                0.002274               -0.019548
..                      ...                     ...                     ...
564               -0.075032               -0.015211               -0.061390
565               -0.015163                0.009985                0.003312
566               -0.004448               -0.055285               -0.012459
567                0.060561               -0.037742               -0.031873
568               -0.019667                0.013734               -0.004134

     principal component 16  principal component 17  principal component 18
0
0                 -0.002670                0.018300                0.010263
0
1                 -0.028715                0.012371               -0.006009
0
2                 -0.041255                0.008218               -0.028044
0
3                 -0.034175                0.033742               -0.016965
0
4                  0.019932               -0.019201                0.004024
0
..                      ...                     ...                     ...
..
564               -0.054694               -0.004829               -0.011515
0
565               -0.020654                0.005197                0.002106
0
566               -0.005414                0.007866               -0.004484
0
```

```
567                 0.020126                 0.015243                 0.043651
0
568                 0.034264                 0.009440                -0.028323
1

[569 rows x 19 columns]
```

```
# Select a random sample of 80% of the data for training, with a fixed random state for reproducibility
training3=finalDataflow3.sample(frac=0.8,random_state=0)

## Select the remaining 20% of the data for testing by dropping the rows that were selected for training
testing3=finalDataflow3.drop(train.index)

# Extract the input features (principal components) for the training and test sets
X_training3 = training3.values[:,0:17]
X_testing3 = testing3.values[:,0:17]

# Extract the output labels (species) for the training and test sets
#Y_train = np.array(train.labels)
#Y_test = np.array(test.labels)
Y_training3 = training3.values[:,18]
Y_testing3 = testing3.values[:,18]
Y_testing3
```

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

```
# Create a new Gaussian model
model = GaussianNaiveBayes()

# Fit the Gaussian model to the training data
model.fit(X_training3,Y_training3)
```

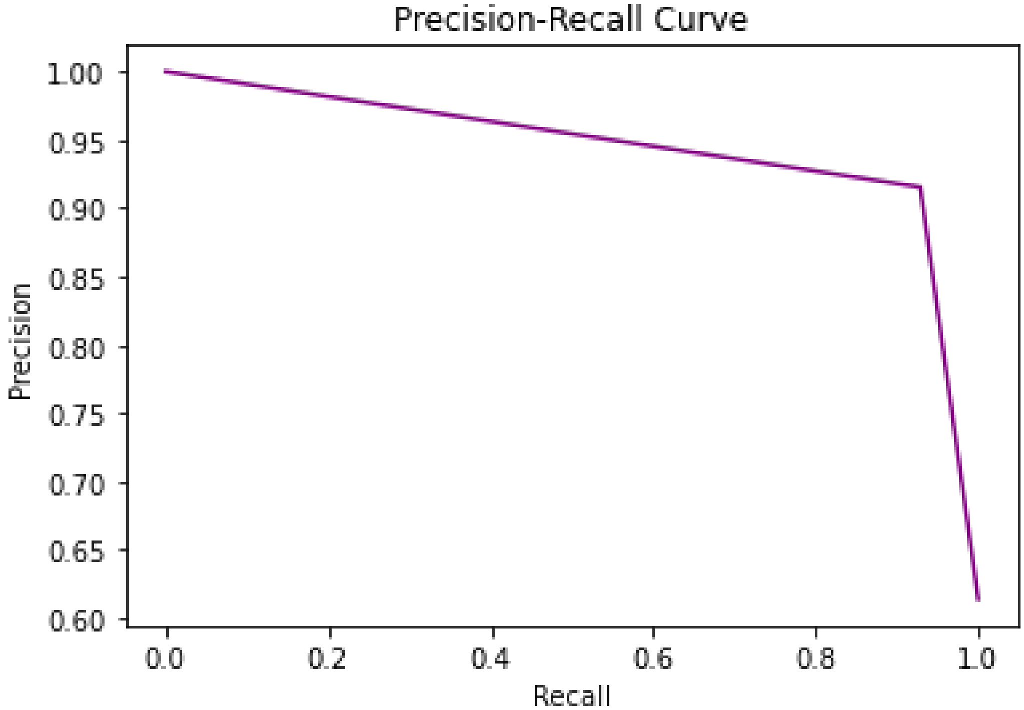
```
GaussianNaiveBayes()
```

```
# Predict the output labels for the test data using the trained model
Y_predicted = model.predict(X_testing3)
Y_predicted
```

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

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

# Print the classification report for the logistic regression model
print(metrics.classification_report(Y_test, Y_predicted))

# Print the confusion matrix for the logistic regression model
print(metrics.confusion_matrix(Y_test,Y_predicted))


# Calculate the precision, recall, and thresholds for the logistic regression
model
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)

# Create a new figure and axis for the precision-recall curve
fig, ax = plt.subplots()

# Plot the precision and recall values as a curve
ax.plot(recall, precision, color='purple')

# Set the title, x-label, and y-label for the plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')


# Show the plot
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.86      0.87        44
         1.0       0.92      0.93      0.92        70

    accuracy                           0.90       114
   macro avg       0.90      0.90      0.90       114
weighted avg       0.90      0.90      0.90       114

[[38  6]
 [ 5 65]]
```


Precision-Recall Curve
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.0
0.2
0.4
0.6
0.8
1.0