Assignment - 01

Topic: Implementation of evaluation measures in Machine Learning Approach

Code:

Dataset: Breast Cancer Dataset by UCI Machine Learning Repository (Asuncion and Newman, 2007)

The Class Samples are either Benign (value = 2) or Malignant (value = 4)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Load dataset
cell_df = pd.read_csv('/cell_samples.csv')

# Create a random array of 699 elements with values 2 and 4
np.random.seed(42)  # for reproducibility
y_pred = np.random.choice([2, 4], size=699)

# Confusion matrix
y_true = cell_df['Class']
conf_matrix = confusion_matrix(y_true, y_pred)
```

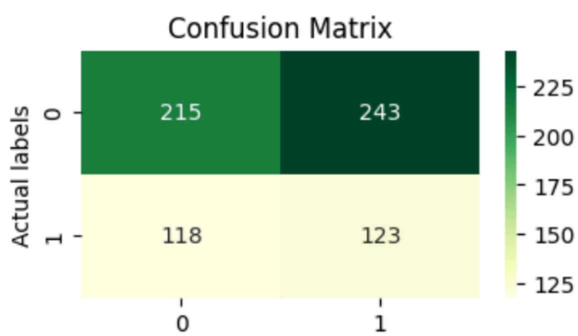
```
# Extract TP, TN, FP, FN from confusion matrix
TP = conf_matrix[1, 1]  # True Positives
TN = conf_matrix[0, 0]  # True Negatives
FP = conf_matrix[0, 1]  # False Positives
FN = conf_matrix[1, 0]  # False Negatives

# Calculate evaluation metrics
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
specificity = TN / (TN + FP)
f1 = 2 * (precision * recall) / (precision + recall)

# Calculate F-beta scores with beta=0.5
beta_05 = 0.5
fbeta_05 = (1 + beta_05**2) * (precision * recall) / (beta_05**2 * precision + recall)
```

```python
# Calculate all measures in percentages
accuracy_percent = accuracy * 100
precision_percent = precision * 100
recall_percent = recall * 100
specificity_percent = specificity * 100
f1_percent = f1 * 100
fbeta_05_percent = fbeta_05 * 100

# Display evaluation metrics
print("Evaluation Metrics:")
print("Accuracy:", accuracy_percent, "%")
print("Precision:", precision_percent, "%")
print("Recall (Sensitivity):", recall_percent, "%")
print("Specificity:", specificity_percent, "%")
print("F1 Score:", f1_percent, "%")
print("F-beta Score (beta=0.5):", fbeta_05_percent, "%")
```

```python
# Plot the confusion matrix
conf_df = pd.DataFrame(conf_matrix, columns=['Predicted Benign', 'Predicted Malignant'],
                       index=['Actual Benign', 'Actual Malignant'])
print("\nConfusion Matrix:")
print(conf_df)

# Plotting the confusion matrix
plt.figure(figsize=(4, 2))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="YlOrBr")
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
plt.title('Confusion Matrix')
plt.show()
```

Output:

```
Evaluation Metrics:
Accuracy: 48.354792560801144 %
Precision: 33.60655737704918 %
Recall (Sensitivity): 51.037344398340245 %
Specificity: 46.943231441048034 %
F1 Score: 40.527182866556835 %
F-beta Score (beta=0.5): 36.070381231671554 %

Confusion Matrix:
                  Predicted Benign  Predicted Malignant
Actual Benign                  215                  243
Actual Malignant               118                  123
```



Assignment - 02

Topic: Implementation of Python Machine Learning Libraries

Code:

Import necessary libraries

```
# @title Import necessary libraries
import numpy as np            # For numerical operations
import pandas as pd           # For data manipulation and analysis
import matplotlib.pyplot as plt  # For data visualization
import seaborn as sns         # For enhanced data visualization
from sklearn.model_selection import train_test_split  # For splitting the dataset
from sklearn.preprocessing import StandardScaler  # For feature scaling
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report  # For model evaluation
import tensorflow as tf       # For deep learning model creation
from tensorflow import keras # For high-level API to build deep learning models
import torch                  # For PyTorch operations

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')
```

Data Preprocessing using Pandas and Numpy

```
# @title Data Preprocessing using Pandas and Numpy
# Replace missing values and convert columns to appropriate data types
cell_df.replace('?', np.nan, inplace=True)
cell_df.dropna(inplace=True)
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)

# Separate features and target variable
X = cell_df.drop(['ID', 'Class'], axis=1).values
y = cell_df['Class'].values
```

Use Scikit-learn for data splitting

```
# @title Use Scikit-learn for data splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling using Scikit-learn
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

A simple neural network model using Keras with TensorFlow backend

```
# @title A simple neural network model using Keras with TensorFlow backend
model = keras.Sequential([
    keras.layers.Dense(16, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_scaled, y_train == 4, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the model
y_pred_keras = model.predict(X_test_scaled).round().astype(int).flatten()
```

```
# Calculate evaluation metrics using Scikit-learn
accuracy = accuracy_score(y_test == 4, y_pred_keras)
conf_matrix = confusion_matrix(y_test == 4, y_pred_keras)
report = classification_report(y_test == 4, y_pred_keras)

# Display evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)
```

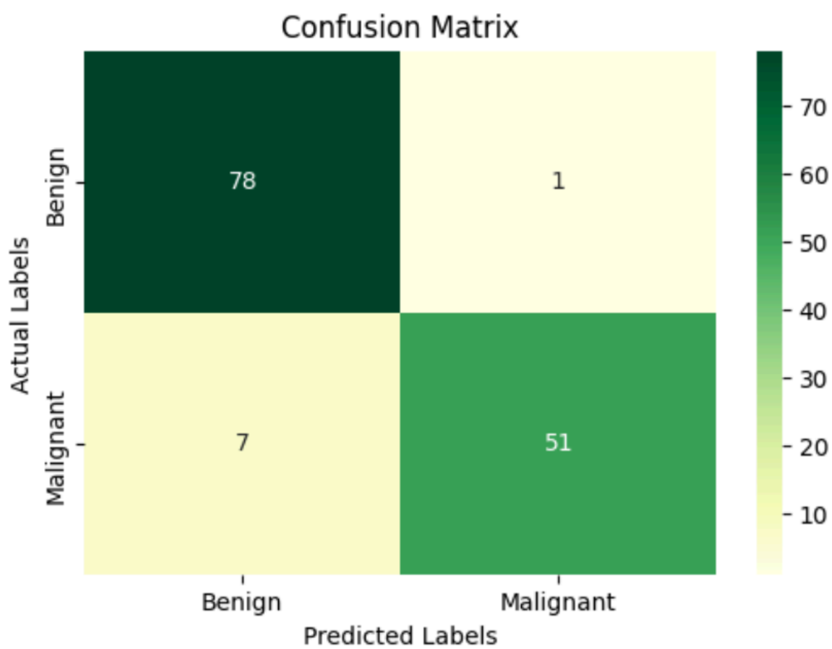
Output:

```
Epoch 1/10
16/16 [==============================] - 1s 16ms/step - loss: 0.6101 - accuracy: 0.7841 - val_loss: 0.5875 - val_accuracy: 0.7818
Epoch 2/10
16/16 [==============================] - 0s 5ms/step - loss: 0.5232 - accuracy: 0.8432 - val_loss: 0.5117 - val_accuracy: 0.7636
Epoch 3/10
16/16 [==============================] - 0s 4ms/step - loss: 0.4428 - accuracy: 0.8574 - val_loss: 0.4434 - val_accuracy: 0.8000
Epoch 4/10
16/16 [==============================] - 0s 4ms/step - loss: 0.3741 - accuracy: 0.8778 - val_loss: 0.3840 - val_accuracy: 0.8000
Epoch 5/10
16/16 [==============================] - 0s 5ms/step - loss: 0.3178 - accuracy: 0.9124 - val_loss: 0.3308 - val_accuracy: 0.8545
Epoch 6/10
16/16 [==============================] - 0s 5ms/step - loss: 0.2710 - accuracy: 0.9470 - val_loss: 0.2843 - val_accuracy: 0.9273
Epoch 7/10
16/16 [==============================] - 0s 7ms/step - loss: 0.2312 - accuracy: 0.9552 - val_loss: 0.2438 - val_accuracy: 0.9455
Epoch 8/10
16/16 [==============================] - 0s 7ms/step - loss: 0.1994 - accuracy: 0.9633 - val_loss: 0.2087 - val_accuracy: 0.9818
Epoch 9/10
16/16 [==============================] - 0s 7ms/step - loss: 0.1732 - accuracy: 0.9674 - val_loss: 0.1828 - val_accuracy: 0.9818
Epoch 10/10
16/16 [==============================] - 0s 6ms/step - loss: 0.1544 - accuracy: 0.9695 - val_loss: 0.1597 - val_accuracy: 0.9818
5/5 [==============================] - 0s 3ms/step
Accuracy: 0.9416058394160584
Confusion Matrix:
 [[78  1]
 [ 7 51]]
Classification Report:
               precision    recall  f1-score   support

       False       0.92      0.99      0.95        79
        True       0.98      0.88      0.93        58

    accuracy                           0.94       137
   macro avg       0.95      0.93      0.94       137
weighted avg       0.94      0.94      0.94       137
```

Plot the confusion matrix using Matplotlib and Seaborn

```
# @title Plot the confusion matrix using Matplotlib and Seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGn', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')
plt.show()
```

Output:



Assignment - 03

Topic: Implementation of **Support Vector Machine (SVM), K-Nearest Neighbors (K-NN), and k-Means Clustering** by using Breast Cancer Dataset by UCI Machine Learning Repository

Code SVM:

Support Vector Machine (SVM)

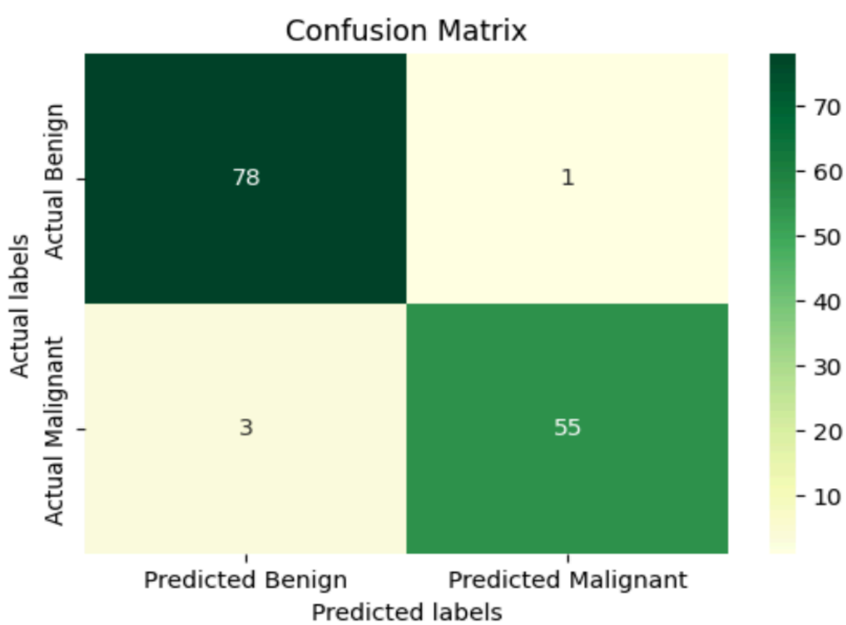
```
# @title Support Vector Machine (SVM)
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')

# Data Cleaning and Preprocessing
# Replace missing values marked as '?' and drop rows with missing values
cell_df.replace('?', np.nan, inplace=True)
cell_df.dropna(inplace=True)
```

```
# Convert 'BareNuc' column to numeric
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)

# Define features and target variable
X = cell_df.drop(['ID', 'Class'], axis=1).values
y = cell_df['Class'].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = svm_model.predict(X_test_scaled)
```

```python
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Display evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)

# Plot the confusion matrix
conf_df = pd.DataFrame(conf_matrix, columns=['Predicted Benign', 'Predicted Malignant'], index=['Actual Benign', 'Actual Malignant'])
plt.figure(figsize=(6, 4))
sns.heatmap(conf_df, annot=True, fmt='d', cmap='YlGn')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
plt.title('Confusion Matrix')
plt.show()
```

Output SVM:

```
Accuracy: 0.9708029197080292
Confusion Matrix:
 [[78  1]
 [ 3 55]]
Classification Report:
               precision    recall  f1-score   support

           2       0.96      0.99      0.97        79
           4       0.98      0.95      0.96        58

    accuracy                           0.97       137
   macro avg       0.97      0.97      0.97       137
weighted avg       0.97      0.97      0.97       137
```



Code KNN:

K-Nearest Neighbors (K-NN)

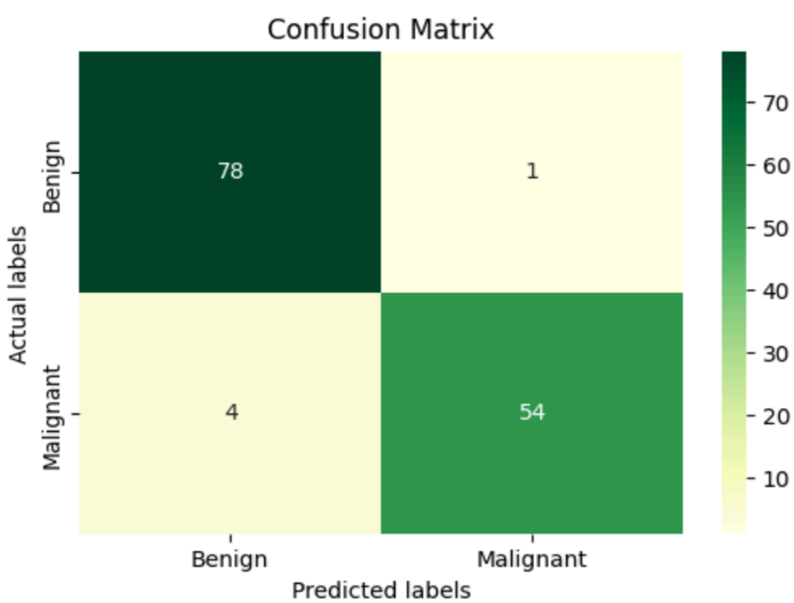
```
# @title K-Nearest Neighbors (K-NN)
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')

# Data Cleaning
# Replace missing values and convert 'BareNuc' column to integer
cell_df.replace('?', np.nan, inplace=True)
cell_df.dropna(inplace=True)
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)
```

```
# Separate features and target variable
X = cell_df.drop(['ID', 'Class'], axis=1)
y = cell_df['Class']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Data Preprocessing
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the k-NN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = knn_model.predict(X_test_scaled)
```

```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Display evaluation metrics
print("Evaluation Metrics:")
print("Accuracy:", accuracy * 100, "%")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='YlGn', xticklabels=['Benign', 'Malignant'],
            yticklabels=['Benign', 'Malignant'])
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
plt.title('Confusion Matrix')
plt.show()
```

Output KNN:

```
Evaluation Metrics:
Accuracy: 96.35036496350365 %
Confusion Matrix:
 [[78  1]
 [ 4 54]]
Classification Report:
               precision    recall  f1-score   support

           2       0.95      0.99      0.97        79
           4       0.98      0.93      0.96        58

    accuracy                           0.96       137
   macro avg       0.97      0.96      0.96       137
weighted avg       0.96      0.96      0.96       137
```



Code K-Means Clustering:

K-Means Clustering

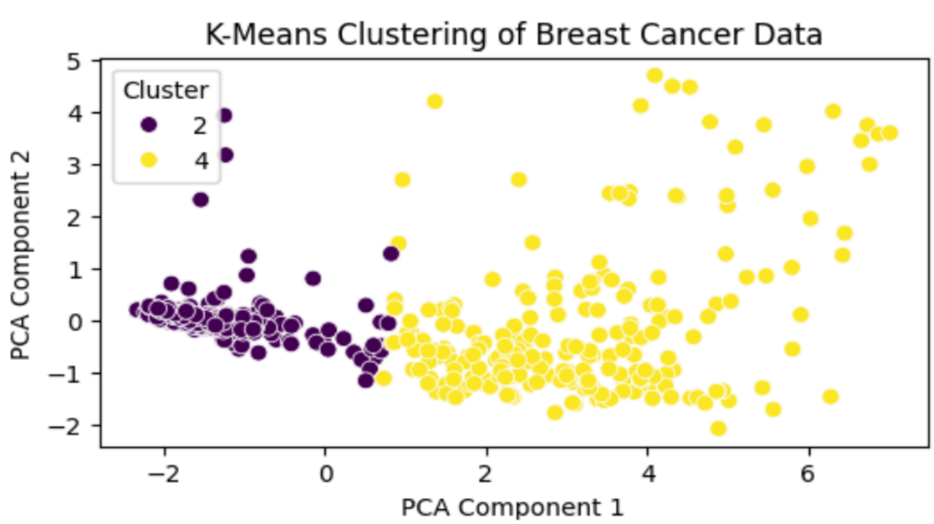
```
# @title K-Means Clustering
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')

# Data cleaning: Drop rows with missing values in 'BareNuc'
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)

# Feature selection
X = cell_df.iloc[:, 1:-1].values
```

```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-Means clustering with explicit value for n_init
kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Add cluster labels to the original dataframe
cell_df['Cluster'] = clusters

# Map clusters to class labels based on majority class in each cluster
cluster_labels = {0: 2, 1: 4} if cell_df[cell_df['Cluster'] == 0]['Class'].mode()[0] == 2 else {0: 4, 1: 2}
cell_df['Cluster'] = cell_df['Cluster'].map(cluster_labels)

# Evaluation Metrics
y_true = cell_df['Class']
y_pred = cell_df['Cluster']

accuracy = accuracy_score(y_true, y_pred)
conf_matrix = confusion_matrix(y_true, y_pred)
```

```python
# Print the required output
print("Number of data points in each cluster:")
print(cell_df['Cluster'].value_counts())

print("\nEvaluation Metrics:")
print("Accuracy:", accuracy * 100, "%")

print("Confusion Matrix:")
print(conf_matrix)

# PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualize the clusters
plt.figure(figsize=(6, 3))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=cell_df['Cluster'], palette='viridis', s=50)
plt.title('K-Means Clustering of Breast Cancer Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()
```

<u>Output K-Means Clustering:</u>

```
Number of data points in each cluster:
Cluster
2    453
4    230
Name: count, dtype: int64

Evaluation Metrics:
Accuracy: 95.75402635431918 %
Confusion Matrix:
[[434  10]
 [ 19 220]]
```



Assignment - 04

Topic: Implementation of **Naive Bayes** by using Breast Cancer Dataset by UCI Machine Learning Repository

Code Naive Bayes:

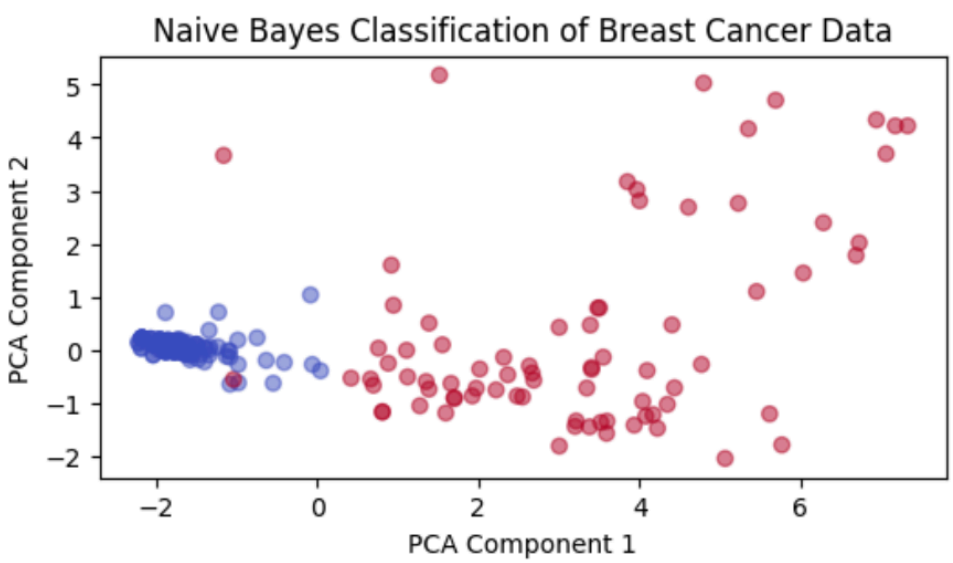
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')

# Data cleaning: Drop rows with missing values in 'BareNuc'
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)

# Feature selection
X = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
y = cell_df['Class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA to reduce to 2D for visualization
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train_scaled, y_train)

# Predict the labels
y_pred = model.predict(X_test_scaled)

# Evaluation Metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

```python
# Evaluation Metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the results
print("\nEvaluation Metrics:")
print("Accuracy:", accuracy * 100, "%")
print("Confusion Matrix:")
print(conf_matrix)

# Visualize the classification results
plt.figure(figsize=(6, 3))
plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, cmap='coolwarm', alpha=0.5)
plt.title('Naive Bayes Classification of Breast Cancer Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```

Output Naive Bayes:

```
Evaluation Metrics:
Accuracy: 96.58536585365853 %
Confusion Matrix:
[[123   4]
 [  3  75]]
```



Assignment - 05

Topic: Implementation of **Decision tree** by using Breast Cancer Dataset by UCI Machine Learning Repository

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn import tree
import matplotlib.pyplot as plt

# Load the dataset
cell_df = pd.read_csv('/content/sample_data/cell_samples.csv')

# Data cleaning: Drop rows with missing values in 'BareNuc'
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype(int)

# Feature selection
X = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
y = cell_df['Class']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the Decision Tree model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train_scaled, y_train)

# Make predictions
y_pred = clf.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

```python
# Print the results
print("Evaluation Metrics:")
print("Accuracy:", accuracy * 100, "%")
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# Visualize the Decision Tree
plt.figure(figsize=(20, 10))
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=['Benign', 'Malignant'], rounded=True)
plt.title("Decision Tree for Breast Cancer Classification")
plt.show()
```

Output Decision Tree:

```
Evaluation Metrics:
Accuracy: 93.43065693430657 %
Confusion Matrix:
[[77  2]
 [ 7 51]]

Classification Report:
              precision    recall  f1-score   support

           2       0.92      0.97      0.94        79
           4       0.96      0.88      0.92        58

    accuracy                           0.93       137
   macro avg       0.94      0.93      0.93       137
weighted avg       0.94      0.93      0.93       137
```