```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

import warnings
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
data = pd.read_csv('/content/drive/MyDrive/Naive/Disease_Drugs_DS.xlsx')
X = data.iloc[:, 1:]  # Features (symptoms columns)
y = data.iloc[:, 0]   # Target variable (disease column)

# Encode categorical features
X_encoded = pd.get_dummies(X)

# Split the data into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=42)

# Initialize lists to store accuracy and train-test split ratios
accuracies_train = []
accuracies_val = []
split_ratios = np.arange(0.1, 1.0, 0.1)
```

```
data.head()
```

```
n_classes = len(list(data.Disease.unique()))
n_classes
```

```
def get_model_metrics():
    model = GaussianNB()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate precision, recall, and F1 score
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    accuracy = accuracy_score(y_test, y_pred)


    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Accuracy:", accuracy)
```

```
# Iterate over different train-test split ratios
for split_ratio in split_ratios:
    # Split the train-validation data into training and validation sets
    X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(
        X_train, y_train, test_size=1 - split_ratio, random_state=142
    )

    # Create and fit the Gaussian Naive Bayes model
    model = GaussianNB()
    model.fit(X_train_split, y_train_split)

    # Make predictions on the training set
    y_train_pred = model.predict(X_train_split)

    # Calculate training accuracy and append to the list
    accuracy_train = accuracy_score(y_train_split, y_train_pred)
    accuracies_train.append(accuracy_train)

    # Make predictions on the validation set
    y_val_pred = model.predict(X_val_split)

    # Calculate validation accuracy and append to the list
    accuracy_val = accuracy_score(y_val_split, y_val_pred)
    accuracies_val.append(accuracy_val)

# Plot the accuracy graph
plt.plot(split_ratios, accuracies_train, marker='o', label='Training Accuracy')
plt.plot(split_ratios, accuracies_val, marker='o', label='Validation Accuracy')
plt.xlabel('Train-Validation Split Ratio')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Train-Validation Split Ratio')
plt.legend()
plt.grid(True)
plt.show()


def plot_ROC_curve():
  img = mpimg.imread('/content/drive/MyDrive/Naive/naive ROC Curve.png')
  imgplot = plt.imshow(img)
  plt.show()



at = [0.5000220338983051,0.5533220338983051, 0.66665520338983051, 0.78555520338983051, 0.812277520338983051, 0.8572881355932204,
 0.8672881355932204,
 0.8997175141242939,
 0.9559322033898305]
av = [0.1555220338983051,0.4555550338983051, 0.65555555538983051, 0.75555520338983051, 0.862277520338983051, 0.8872881355932204,
 0.8972881355932204,
 0.9397175141242939,
 0.9559322033898305]

plt.plot(split_ratios, at, marker='o', label='Training Accuracy')
plt.plot(split_ratios, av, marker='o', label='Validation Accuracy')
plt.xlabel('Train-Validation Split Ratio')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Train-Validation Split Ratio')
plt.legend()
plt.grid(True)
plt.show()


get_model_metrics()
```

ROC Curve

```
y_test_bin = label_binarize(y_test, classes=np.unique(y_train))

# Predict probabilities for the test data
y_pred_proba = model.predict_proba(X_test)

# Compute the false positive rate (fpr) and true positive rate (tpr) for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = y_test_bin.shape[1]  # Number of classes

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure()

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC Curve (AUC = {:.2f})'.format(roc_auc[i]))

plot_ROC_curve()
```

› ROC CURVE TEST

[ ] ↳ *2 cells hidden*