######################################################

#  Student Name: Devin Bailie

#  Student ID: bailied2 / W30644462

#  Course Code: CSCI 460 -- Fall 2025

#  Assignment Due Date: 10/14/2025

#  GitHub Link: https://github.com/bailied2/csci460-naive-bayes

######################################################

I used a Pandas DataFrame to import the dataset, then mapped each categorical variable to an integer value and cleared the DataFrame of rows with missing values. Initially, I was not mapping “unknown” values to a number and dropping rows that contained them, but after doing some research I decided that I was probably supposed to keep them since they aren’t literally empty cells, and my average accuracy improved from about 74% to about 84% when retaining rows with “unknown” values.

After prepping the data, the program then splits the DataFrame into two parts, one containing the ‘y’ column (y) and the other containing the rest of the columns (X). Then, I used the sklearn library to split the data in a stratified manner, create a Gaussian Naïve Bayes model and train it on the data.

After running the program for 10 iterations with 70% training and 30% testing data and comparing the predicted values to the test data, I got the following accuracy and F1-score values (rounded to 4 decimal places for readability):

|  |  |  |
| --- | --- | --- |
|  | Accuracy | F1-Score |
| Iteration 1 | 0.8409 | 0.8372 |
| Iteration 2 | 0.8452 | 0.8396 |
| Iteration 3 | 0.8413 | 0.8363 |
| Iteration 4 | 0.8444 | 0.8382 |
| Iteration 5 | 0.8444 | 0.8406 |
| Iteration 6 | 0.8439 | 0.8385 |
| Iteration 7 | 0.8449 | 0.8417 |
| Iteration 8 | 0.8421 | 0.8396 |
| Iteration 9 | 0.8484 | 0.8432 |
| Iteration 10 | 0.8350 | 0.8314 |
| Average | 0.8430 | 0.8386 |
| Standard Deviation | 0.0034 | 0.0031 |

For the next task, I looped through an array of training data proportions, running the test again for 9 different proportions as directed in the assignment instructions. I went ahead and wrote the code to calculate the average/standard deviation of the accuracies and F1-scores for each set of tests in task 3, to save having to calculate each one myself or copy all 20 values into a spreadsheet for all 9 tests. The table below shows my results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Accuracy | Accuracy  Standard Deviation | Average  F1-Score | F1-Score  Standard Deviation |
| 10% Training, 90% Testing | 0.8328 | 0.0126 | 0.8317 | 0.0075 |
| 20% Training, 80% Testing | 0.8442 | 0.0083 | 0.8400 | 0.0058 |
| 30% Training, 70% Testing | 0.8414 | 0.0045 | 0.8372 | 0.0044 |
| 40% Training, 60% Testing | 0.8385 | 0.0042 | 0.8357 | 0.0036 |
| 50% Training, 50% Testing | 0.8424 | 0.0042 | 0.8388 | 0.0033 |
| 60% Training, 40% Testing | 0.8435 | 0.0068 | 0.8393 | 0.0053 |
| 70% Training, 30% Testing | 0.8432 | 0.0060 | 0.8392 | 0.0038 |
| 80% Training, 20% Testing | 0.8430 | 0.0052 | 0.8388 | 0.0037 |
| 90% Training, 10% Testing | 0.8486 | 0.0058 | 0.8424 | 0.0061 |

The results are more similar than I initially expected for each test, but that may be partially due to the size of the dataset, or even more-so perhaps due to the stratification of the split. Doing some research about these results, I learned that this is also typical of the Gaussian Naïve Bayes model. Looking at the data, you can see that while the tests with less training data still performed relatively well, the standard deviation of both the accuracy and F1-scores are significantly higher in the first few tests than the tests with more training data and less testing data, up to a point. This is particularly visible in the first jump in accuracy standard deviation from 0.0126 with 10% training data to 0.0083 with 20% training data. However, at 90% training data and 10% testing data, the standard deviations spiked up again slightly, because the proportion of testing data was too small.

From the table, you can see that with more training data the model’s predictions become more consistent across iterations, until the testing data becomes too small. There are small ups and downs throughout, but this is to be expected due to noise.