1. **Suppose we have Car data provided on Page 2 collected and the dataset contains three features. The first feature is the color, the second feature is Type, and the third feature is Origin. The target attribute is marked Stolen, which indicates whether a specific car is stolen or not. Suppose we have the following training data including 14 training samples or examples. Using Naive Bayes Classifier to classify a new instance which follows a condition New Instance = (Blue, SUV, Domestic) into (Yes or No). Please include the detailed calculation process.**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(Stolen|Blue, SUV, Domestic) = \frac{P(Stolen)P(Blue, SUV, Domestic|Stolen)}{P(Blue, SUV, Domestic)} \quad (1)$$

$$P(Notstolen|Blue, SUV, Domestic) = \frac{P(Notstolen)P(P(Blue, SUV, Domestic|Notstolen))}{P(Blue, SUV, Domestic)} \quad (2)$$

Color	Stolen	Not stolen
Red	3/6	4/8
Yellow	2/6	2/8
blue	1/6	2/8

Type	Stolen	Not stolen
Sports	4/6	3/8
SUV	2/6	5/8

Origin	Stolen	Not stolen
Domestic	2/6	5/8
Imported	4/6	3/8

$$P(Stolen|Blue, SUV, Domestic) = \frac{\frac{6}{14} * \frac{1}{6} * \frac{2}{6} * \frac{2}{6}}{(\frac{1}{6} * \frac{2}{6} * \frac{2}{6} * \frac{6}{14}) + (\frac{2}{8} * \frac{5}{8} * \frac{5}{8} * \frac{8}{14})} = 0.125$$

(1)

$$P(Notstolen|Blue, SUV, Domestic) = \frac{\frac{8}{14} * \frac{2}{8} * \frac{5}{8} * \frac{5}{8}}{(\frac{1}{6} * \frac{2}{6} * \frac{2}{6} * \frac{6}{14}) + (\frac{2}{8} * \frac{5}{8} * \frac{5}{8} * \frac{8}{14})} = 0.875$$

(2)

$$P(Notstolen|Blue, SUV, Domestic) > P(Stolen|Blue, SUV, Domestic)$$

So, the car classification is '*NOT STOLEN*'

$R(a_1|x) = \lambda_{11} P(C_1|x) + \lambda_{12} P(C_2|x)$

$R(a_1|x) = 0 + 6\,P(C_2|x) \longrightarrow (1)$

$R(a_2|x) = \lambda_{21} P(C_1|x) + \lambda_{22} P(C_2|x)$

$R(a_2|x) = 3\,P(C_1|x) + 0 \longrightarrow (2)$

$R(a_3|x) = \lambda_{31} P(C_1|x) + \lambda_{32} P(C_2|x)$

$R(a_3|x) = 2 + 2P(C_2|x) - 2P(C_2|x)$

Choose a_1 if $R(a_1|x) < R(a_3|x)$

$6(1 - P(C_1|x)) < 2$

$\boxed{P(C_1|x) > \frac{2}{3}}$

Choose a_2 if $R(a_2|x) < R(a_3|x)$

$3P(C_1|x) < 2$ $\boxed{P(C_1|x) < \frac{2}{3}}$

a_1: $P(C_1|x) > \frac{2}{3}$

a_2: $P(C_1|x) < \frac{2}{3}$

$\boxed{\frac{2}{3} < \text{Rejection} < \frac{2}{3}}$

Reject when $P(C_1|x) = \frac{2}{3}$

Part 2:

(a) Split the dataset into two parts as training data and test data. first 80 percent samples should be selected as training data and last 20 percent samples should be selected as test data. Please save these training and test datasets for remaining parts. Compute the confusion matrix and the accuracy of test data for Gaussian and Multinomial Naive Bayes Classifiers:

1) Code snippet of the block responsible for splitting the data as requested and initializing and training the models:

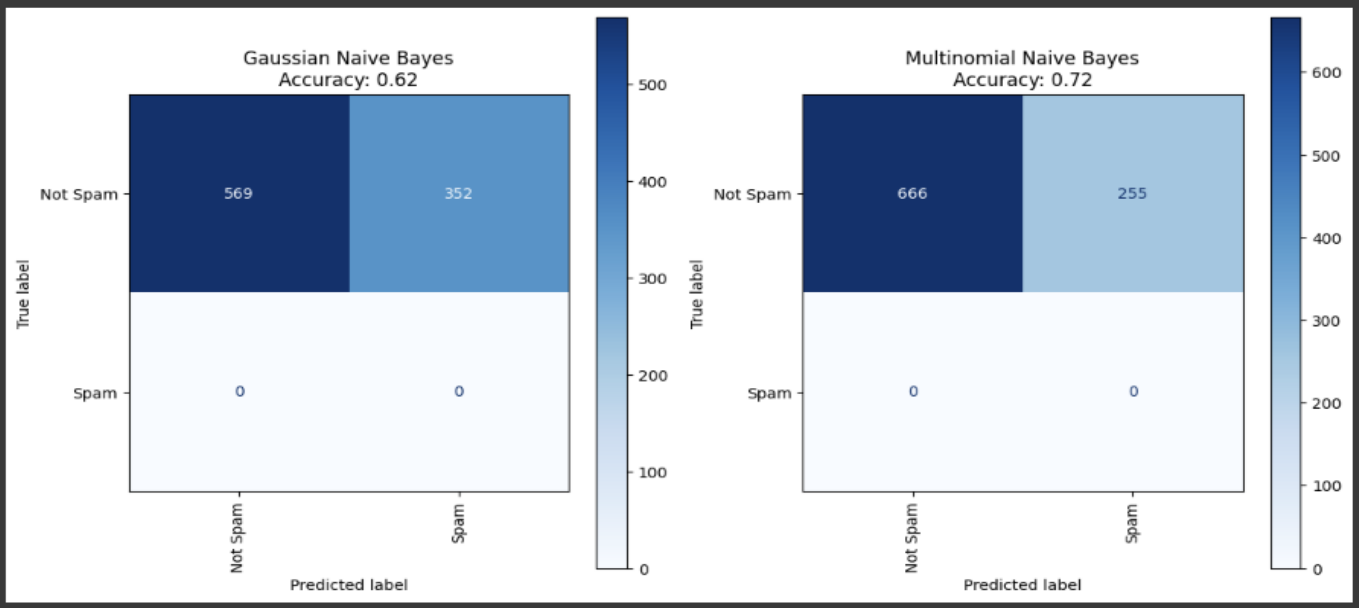
Split the dataset into two parts as training data and test data. first 80 percent samples should be selected as training data and last 20 percent samples should be selected as test data.

```
#splitting the data as reuested using data slicing technique and creating the classifiers
split_index = int(0.8 * len(data))

train_data_a = data[:split_index]
test_data_a = data[split_index:]

x_train_a = train_data_a.drop(columns=[57])
y_train_a = train_data_a[57]

x_test_a = test_data_a.drop(columns=[57])
y_test_a = test_data_a[57]

gnb = GaussianNB()
gnb.fit(x_train_a, y_train_a)

mnb = MultinomialNB()
mnb.fit(x_train_a, y_train_a)
```

```
▾ MultinomialNB
MultinomialNB()
```

2) Code snippet showing the code which draw the confusion matrix for the models' predictions:

```
#predicting the outputs and calculating the accuracy of the models and creating the confusion matrix for evaluation
y_pred_gnb_a = gnb.predict(x_test_a)
cm_gnb_a = confusion_matrix(y_test_a, y_pred_gnb_a)
accuracy_gnb_a = accuracy_score(y_test_a, y_pred_gnb_a)

y_pred_mnb_a = mnb.predict(x_test_a)
cm_mnb_a = confusion_matrix(y_test_a, y_pred_mnb_a)
accuracy_mnb_a = accuracy_score(y_test_a, y_pred_mnb_a)

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

disp_gnb_a = ConfusionMatrixDisplay(confusion_matrix=cm_gnb_a, display_labels=["Not Spam", "Spam"])
disp_gnb_a.plot(ax=axes[0], cmap='Blues', xticks_rotation='vertical')
axes[0].set_title("Gaussian Naive Bayes\nAccuracy: {:.2f}".format(accuracy_gnb_a))

disp_mnb_a = ConfusionMatrixDisplay(confusion_matrix=cm_mnb_a, display_labels=["Not Spam", "Spam"])
disp_mnb_a.plot(ax=axes[1], cmap='Blues', xticks_rotation='vertical')
axes[1].set_title("Multinomial Naive Bayes\nAccuracy: {:.2f}".format(accuracy_mnb_a))

plt.tight_layout()
plt.show()
```



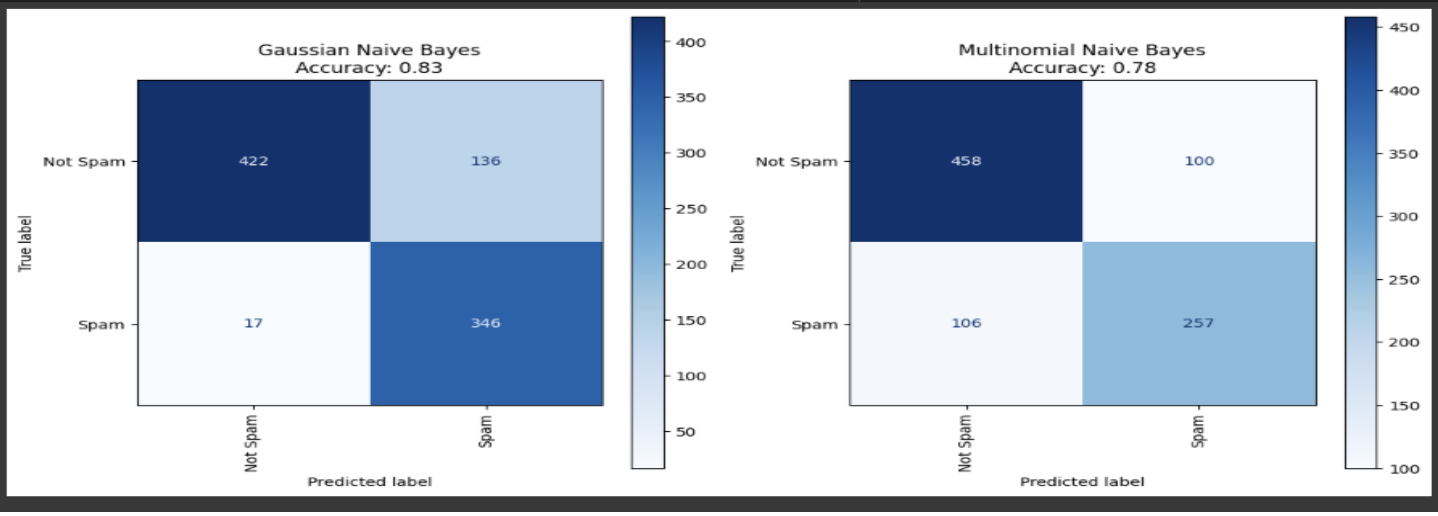
(b) Use train test split function on input and output of the whole data and utilize 80% of samples as train and 20% of samples as test data. Please save these training and test datasets for the remaining parts. After selecting the training and test dataset, compute the confusion matrix and the accuracy of test data for Gaussian and Multinomial Naive Bayes Classifiers:

1) Code responsible for splitting the data and training the models:

```
[312] data.columns

      Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                  17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                  34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
                  51, 52, 53, 54, 55, 56, 57],
                 dtype='int64')

[313] x = data.drop(57,axis=1)
      y = data[57]

      x_train_b , x_test_b , y_train_b ,y_test_b = train_test_split(x,y,stratify = y, test_size=0.2, random_state=42)

[315] # Initialize and train the guassian and Multinomial Naive Bayes classifier
      gnb = GaussianNB()
      gnb.fit(x_train_b,y_train_b)

      mnb = MultinomialNB()
      mnb.fit(x_train_b,y_train_b)

      ▾ MultinomialNB
      MultinomialNB()
```

2) Code responsible for creating the confusion matrix for the model's output:

```
y_pred_gnb = gnb.predict(x_test_b)
cm_gnb = confusion_matrix(y_test_b, y_pred_gnb)
accuracy_gnb = accuracy_score(y_test_b, y_pred_gnb)

y_pred_mnb = mnb.predict(x_test_b)
cm_mnb = confusion_matrix(y_test_b, y_pred_mnb)
accuracy_mnb = accuracy_score(y_test_b, y_pred_mnb)

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

disp_gnb = ConfusionMatrixDisplay(confusion_matrix=cm_gnb, display_labels=["Not Spam", "Spam"])
disp_gnb.plot(ax=axes[0], cmap='Blues', xticks_rotation='vertical')
axes[0].set_title("Gaussian Naive Bayes\nAccuracy: {:.2f}".format(accuracy_gnb))

disp_mnb = ConfusionMatrixDisplay(confusion_matrix=cm_mnb, display_labels=["Not Spam", "Spam"])
disp_mnb.plot(ax=axes[1], cmap='Blues', xticks_rotation='vertical')
axes[1].set_title("Multinomial Naive Bayes\nAccuracy: {:.2f}".format(accuracy_mnb))

plt.tight_layout()
plt.show()
```



(c) Use another Naive Bayes classifier of your choice to check for the improvement in terms of accuracy score of test data in (b) over Gaussian and Multinomial asked in (b) and provide an explanation for the improvement in performance (if any). Also, provide classification report in terms of precision, recall and F1-score and display the confusion matrix for only the selected classifier:

1) Code responsible for training the model and calculating it's accuracy score and the explanation why did it scored more than the gaussian and multinomial naïve bayes classifiers:

Use another Naive Bayes classifier of your choice to check for the improvement in terms of accuarcy score of test data in (b) over Gaussian and Multinomial asked in (b)

```
# Initialize and train the Bernoulli Naive Bayes classifier
bern = BernoulliNB(alpha=1.0, fit_prior=True)
bern.fit(x_train_b, y_train_b)
y_pred_bern = bern.predict(x_test_b)

accuracy = accuracy_score(y_test_b, y_pred_bern)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8762214983713354
```

Provide an explanation for the improvement in performance (if any).

> The Bernoulli Naive Bayes classifier might have achieved better accuracy in the previous problem compared to Gaussian and Multinomial Naive Bayes classifiers due to the following reasons:
> 1) Binary Feature Representation as it assume features are binary and it will suited with the binary feature representation.
> 2) Handling Irrelevant Features as it considers the presence or absence of features, disregarding their frequency or intensity.

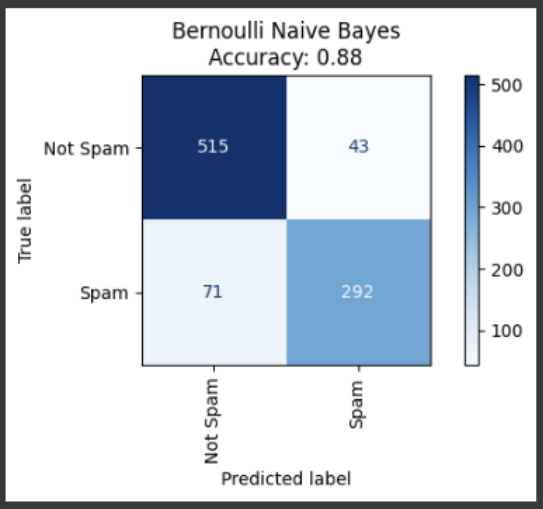
2) Code for the classification report and the confusion matrix of the Bernoulli naïve bayes classifier:

```
#generating the classification report
report = classification_report(y_test_b, y_pred_bern)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       558
           1       0.87      0.80      0.84       363

    accuracy                           0.88       921
   macro avg       0.88      0.86      0.87       921
weighted avg       0.88      0.88      0.88       921
```

Display the confusion matrix for only the selected classifier

```
[333] #drawing the confusion matrix
cm_bern = confusion_matrix(y_test_b, y_pred_bern)
fig, ax = plt.subplots(figsize=(6, 4))
disp_mnb = ConfusionMatrixDisplay(confusion_matrix=cm_bern, display_labels=["Not Spam", "Spam"])
disp_mnb.plot(ax=ax, cmap='Blues', xticks_rotation='vertical')
ax.set_title("Bernoulli Naive Bayes\nAccuracy: {:.2f}".format(accuracy))
plt.tight_layout()
plt.show()
```



(d) Take same first 80 percent as asked in (a) training samples and split the data into four equal parts according to order such as the first 25% of training data (subset 1), the second 25% of training data (subset 2), the third 25% of training data (subset 3) and the fourth 25% of training data (subset 4). The train selected classifier chosen in (c) for each subset and predicted the accuracy score by evaluating on last 20 percent of test data assumed in (a). Plot bar chart to show all subsets' accuracy on the figure. Add your comment:

1) Code for splitting the data into subsets:

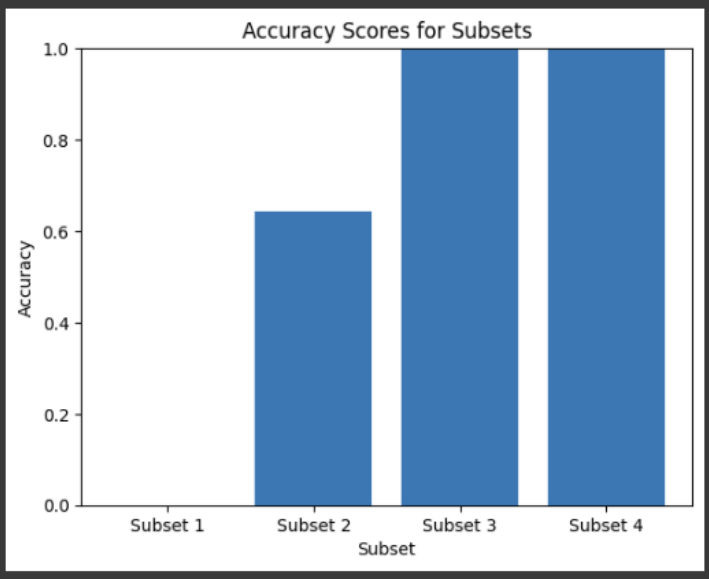
```
#splitting the part A training data into 4 equal subsets
split_index = int(len(train_data_a))
subset_size = int(split_index * 0.25)

subset_1 = data[:subset_size]
subset_2 = data[subset_size:2*subset_size]
subset_3 = data[2*subset_size:3*subset_size]
subset_4 = data[3*subset_size:split_index]

x_subset_1 = subset_1.drop(columns=[57])
y_subset_1 = subset_1[57]

x_subset_2 = subset_2.drop(columns=[57])
y_subset_2 = subset_2[57]

x_subset_3 = subset_3.drop(columns=[57])
y_subset_3 = subset_3[57]

x_subset_4 = subset_4.drop(columns=[57])
y_subset_4 = subset_4[57]
```

2) Code for training the model chose in problem “c” on each set and calculating the accuracy score with each subset:

```
#fitting the bernoulli model ont he 4 subsets and generating accuracy scores for each subset
accuracy_scores = []
for i in range(1,5):
  x = globals()['x_subset_'+str(i)]
  y = globals()['y_subset_'+str(i)]
  bern.fit(x,y)
  y_pred = bern.predict(x_test_a)
  accuracy_scores.append(accuracy_score(y_test_a,y_pred))
```

3) Drawing the bar plot for the accuracy scores of each subset and commenting on the results:

```python
#plotting bar plot for the accuracy scores of the 4 subsets
subset_labels = ['Subset 1', 'Subset 2', 'Subset 3', 'Subset 4']

plt.bar(subset_labels, accuracy_scores)
plt.xlabel('Subset')
plt.ylabel('Accuracy')
plt.title('Accuracy Scores for Subsets')
plt.ylim(0, 1)
plt.show()
```



Add your comment

As we can see:

1)The first subset training data set didn't provide the model with a variation between the spam and not spam classes as the first portion of the data are all from '1' class.

2)For the second subset it did contain a small portion of the '0' class so the model was able to capture some more info to classify better and did increase the accuracy .

3) For the third and fourth subset the accuracy was 100% because the test data is all from the '0' class as the subsets so they were most likely to assign all the coming predictions to class '0'.