```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,classification_report

data={ 'text':['hello section B','win the lottery','congratulations'],'label':['nspam','spam','spam']}
df=pd.DataFrame(data)
df
```

	text	label
0	hello section B	nspam
1	win the lottery	spam
2	congratulations	spam

Next steps: View recommended plots | New interactive sheet

```
# *Step 4: Split Data*
# Split the dataset into training and testing sets.

X = df['text']
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)



# *Step 5: Vectorize the Text*
# Convert text data into numerical format using CountVectorizer.

vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
 # Learns the vocabulary dictionary from the training data and Transforms the training data into the document-term matrix
X_test_counts = vectorizer.transform(X_test)
 #uses the already learned vocabulary from the training data to convert the test data into the same document-term matrix format.
doc_term_matrix = pd.DataFrame(X_train_counts.toarray(),columns= vectorizer.get_feature_names_out())
doc_term_matrix
```

	congratulations	lottery	the	win
0	0	1	1	1
1	1	0	0	0

```
# Step 6: Train the Naive Bayes Classifier*
# Now, we'll train a Multinomial Naive Bayes classifier.

model = MultinomialNB()
# multinomial Nave bayes theorem suitable for or text classification tasks where word counts are used as features.
model.fit(X_train_counts, y_train)
 #  model learns the relationship between the word counts (X_train_counts) and the corresponding labels (y_train).
```

```
MultinomialNB
MultinomialNB()
```

```
# *Step 7: Make Predictions*
# Use the trained model to make predictions on the test set.

y_pred = model.predict(X_test_counts)
# Uses the trained model to predict the labels ('spam' or 'not spam') for the test data (X_test_counts)



# *Step 8: Evaluate the Model*
# Evaluate the performance of the model using accuracy and a classification report.
#Calculates the proportion of correctly predicted labels out of all predictions.

# Formula:
# Accuracy=Number of Correct Predictions/Total Number of Predictions

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy}')
print('Classification Report:')
print(report)
```

```
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

       nspam       0.00      0.00      0.00       1.0
        spam       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
new_email=['congratulation you win the lottery']
new_email_vectorize=vectorizer.transform(new_email)
prediction=model.predict(new_email_vectorize)
print(f"Prediction:{'spam' if prediction[0] ==1 else 'notspam'}")
```

```
Prediction:notspam
```