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

with **sklearn**

```
data=pd.read_csv("/content/spam.csv",encoding="latin-1")
```

```
data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
data=data.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1) # Pass all column names to be dropped as a single list. Specify a
```

```
data.shape
```

```
(5572, 2)
```

```
x=data.iloc[:,1]
y=data.iloc[:,0]
print(x)
print(y)
```

```
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                Will Ì_ b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                           Rofl. Its true to its name
Name: v2, Length: 5572, dtype: object
0        ham
1        ham
2       spam
3        ham
4        ham
        ...
5567    spam
5568     ham
5569     ham
5570     ham
5571     ham
Name: v1, Length: 5572, dtype: object
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
from sklearn.naive_bayes import BernoulliNB as naive_bayes
from sklearn.metrics import accuracy_score
```

```
accu=[]
pre=[]
re=[]
f1=[]
f12=[]
```

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer

x = data.iloc[:, 1]
y = data.iloc[:, 0]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

vectorizer = CountVectorizer()
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)
# print(x_test)
model = BernoulliNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(accuracy_score(y_test, y_pred))
accu.append(accuracy_score(y_test, y_pred))
pre.append(precision_score(y_test, y_pred, pos_label='ham'))
re.append(recall_score(y_test, y_pred, pos_label='spam'))
f1.append(f1_score(y_test, y_pred, pos_label='ham'))
f12.append(f1_score(y_test, y_pred, pos_label='spam'))
```

```
0.9748878923766816
```

```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(accuracy_score(y_test,y_pred))
accu.append(accuracy_score(y_test, y_pred))
pre.append(precision_score(y_test, y_pred, pos_label='ham'))
re.append(recall_score(y_test, y_pred, pos_label='spam'))
f1.append(f1_score(y_test, y_pred, pos_label='ham'))
f12.append(f1_score(y_test, y_pred, pos_label='spam'))
```

```
0.9838565022421525
```

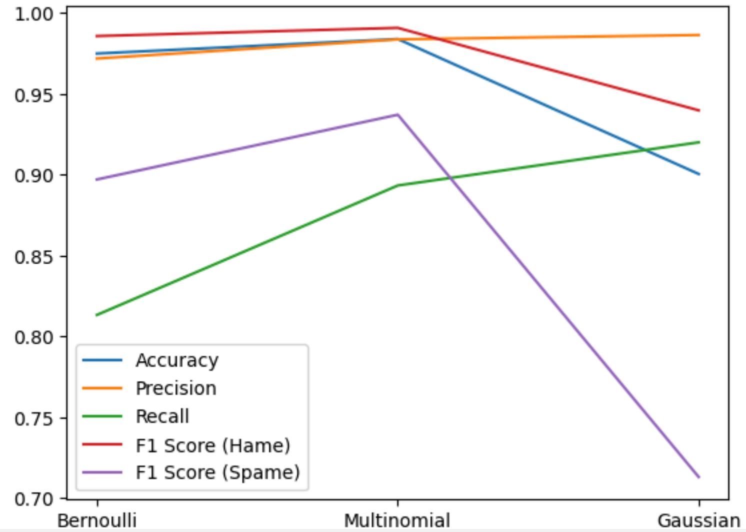
```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
x_train = x_train.toarray()
x_test = x_test.toarray()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(accuracy_score(y_test,y_pred))
accu.append(accuracy_score(y_test, y_pred))
pre.append(precision_score(y_test, y_pred, pos_label='ham'))
re.append(recall_score(y_test, y_pred, pos_label='spam'))
f1.append(f1_score(y_test, y_pred, pos_label='ham'))
f12.append(f1_score(y_test, y_pred, pos_label='spam'))
```

```
0.9004484304932735
```

```
import seaborn as sns
import matplotlib.pyplot as plt

h = ["Bernoulli", "Multinomial", "Gaussian"]
# Create the plot
sns.lineplot(x=h, y=accu, label="Accuracy")
sns.lineplot(x=h, y=pre, label="Precision")
sns.lineplot(x=h, y=re, label="Recall")
sns.lineplot(x=h, y=f1, label="F1 Score (Hame)")
sns.lineplot(x=h, y=f12, label="F1 Score (Spame)")
# Show the plot
plt.show()

```



without **sklearn**

```
data=pd.read_csv("/content/Iris.csv")
```

```
data.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
data=data.drop(['Id'],axis=1)
```

```
data=data[data['Species'] != 'Iris-setosa']
```

```
data.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
53	5.5	2.3	4.0	1.3	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor

```
x=data.iloc[:,:-1]
y=data.iloc[:,-1]
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
a=np.mean(x_train[y_train=='Iris-versicolor'],axis=0)
b=np.mean(x_train[y_train=='Iris-virginica'],axis=0)

c=np.std(x_train[y_train=='Iris-versicolor'],axis=0)
d=np.std(x_train[y_train=='Iris-virginica'],axis=0)
```

```
print(a,b,c,d)
```

```
SepalLengthCm    5.947368
SepalWidthCm     2.839474
PetalLengthCm    4.257895
PetalWidthCm     1.339474
dtype: float64 SepalLengthCm    6.600000
```

```
SepalWidthCm     2.985714
PetalLengthCm    5.592857
PetalWidthCm     2.047619
dtype: float64 SepalLengthCm    0.507178
SepalWidthCm     0.268073
PetalLengthCm    0.461480
PetalWidthCm     0.188539
dtype: float64 SepalLengthCm    0.661888
SepalWidthCm     0.339868
PetalLengthCm    0.554373
PetalWidthCm     0.265687
dtype: float64
```

```python
tz=(x_test-a)/c
vz=(x_test-b)/d

p=np.exp(-0.5*((tz)**2))/(np.sqrt(2*np.pi)*c)
q=np.exp(-0.5*((vz)**2))/(np.sqrt(2*np.pi)*d)
```

```python
print(p,q)
```

```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
133       0.617699      1.472141       0.163563  1.472632e+00
103       0.617699      1.450735       0.012593  1.071379e-01
120       0.134789      0.602422       0.006550  4.892565e-06
95        0.698382      1.243921       0.857708  1.609443e+00
94        0.622138      1.299802       0.857708  2.070092e+00
89        0.533086      0.667469       0.739505  2.070092e+00
72        0.617699      0.667469       0.328367  1.472632e+00
130       0.013014      1.472141       0.000300  2.548066e-02
60        0.137435      0.011047       0.224424  4.183353e-01
50        0.091280      0.602422       0.546337  2.009690e+00
68        0.694819      0.086499       0.753339  1.472632e+00
80        0.533086      0.388202       0.528410  9.444681e-01
123       0.617699      1.299802       0.328367  1.071379e-01
83        0.782368      1.299802       0.163563  8.144885e-01
140       0.261555      0.928034       0.012593  2.850573e-07
54        0.434447      1.472141       0.656783  1.472632e+00
126       0.694819      1.472141       0.433617  1.071379e-01
127       0.751767      1.243921       0.328367  1.071379e-01
62        0.782368      0.086499       0.739505  4.183353e-01
81        0.533086      0.388202       0.416291  4.183353e-01      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
133       0.543896      1.011027       0.484706      0.179486
103       0.543896      1.137073       0.719568      0.972573
120       0.543896      0.962228       0.706312      0.956309
95        0.239133      1.172779       0.030644      0.009256
94        0.192515      0.824402       0.030644      0.028652
89        0.151487      0.422764       0.011599      0.028652
72        0.543896      0.422764       0.329551      0.179486
130       0.290337      1.011027       0.473570      1.286783
60        0.032451      0.017500       0.000579      0.000632
50        0.502134      0.962228       0.196717      0.076975
68        0.502134      0.081106       0.103095      0.179486
80        0.151487      0.265877       0.003855      0.002595
123       0.543896      0.824402       0.329551      0.972573
83        0.399657      0.824402       0.484706      0.363233
140       0.595894      1.109293       0.719568      0.623105
54        0.595894      1.011027       0.144745      0.179486
126       0.502134      1.011027       0.258790      0.972573
127       0.453117      1.172779       0.329551      0.972573
62        0.399657      0.081106       0.011599      0.000632
81        0.151487      0.265877       0.002116      0.000632
```

```python
n_versicolor=len(y_train[y_train=='Iris-versicolor'])/len(y_train)
n_virginica=len(y_train[y_train=='Iris-virginica'])/len(y_train)
```

```python
p_versicolor=np.prod(p,axis=1)*n_versicolor
p_virginica=np.prod(q,axis=1)*n_virginica
result = np.where(p_versicolor > p_virginica, 'Iris-versicolor', 'Iris-virginica')
```

```python
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,result))
```

```
0.95
```

```python
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.95
```

bernoulli

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer()
x_train=vectorizer.fit_transform(x_train)
x_test=vectorizer.transform(x_test)
x_trainh=x_train[y_train=='ham'].toarray()
x_trains=x_train[y_train=='spam'].toarray()
print(x_trainh)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
x_test=x_test.toarray()
```

```
x_trainh.shape
x_trains.shape
x_test.shape
```

```
(1115, 7735)
```

```
x=[]
y=[]
for i in range(len(x_trainh[0])):
  count=0
  for j in range(len(x_trainh)):
    if x_trainh[j][i]>=1:
      count+=1
  x.append(count)
for i in range(len(x_trains[0])):
  count=0
  for j in range(len(x_trains)):
    if x_trains[j][i]>=1:
      count+=1
  y.append(count)
```

```
y_pred=[]
for i in range(len(x_test)):
  p=len(x_trainh)/(len(x_trainh)+len(x_trains))
  q=len(x_trains)/(len(x_trainh)+len(x_trains))
  for j in range(len(x_test[i])):
    if x_test[i][j]>=1:
      p*=(x[j])/(len(x_trainh[0]))
      q*=(y[j])/(len(x_trains[0]))
    else:
      p*=1-((x[j])/(len(x_trainh[0])))
      q*=1-((y[j])/(len(x_trains[0])))
  if p>q:
    y_pred.append("ham")
  else:
    y_pred.append("spam")
```

```
from sklearn.metrics import confusion_matrix

# Initialize lists before appending
accu = []
pre = []
re = []
f1_scores = []  # Changed from f1 to f1_scores to avoid conflict

# confusion matrix returns a 2x2 array
cm = confusion_matrix(y_test, y_pred)  # y_test and y_pred were swapped in your original call

# Access the values by their index
tn, fp, fn, tp = cm.ravel()

# Calculate metrics
ac = (tp + tn) / (tp + tn + fp + fn)
pr = tp / (tp + fp)
r = tp / (tp + fn)
f1 = (2 * pr * r) / (pr + r)

# Append the calculated values to respective lists
```

```
accu.append(ac)
pre.append(pr)
re.append(r)
f1_scores.append(f1)
```

```
x=[]
y=[]
c=0
for i in range(len(x_trainh[0])):
  count=0
  for j in range(len(x_trainh)):
    if x_trainh[j][i]>=1:
      count+=x_trainh[j][i]
      c+=x_trainh[j][i]
  x.append(count)
for i in range(len(x_trains[0])):
  count=0
  for j in range(len(x_trains)):
    if x_trains[j][i]>=1:
      count+=x_trains[j][i]
      c+=x_trains[j][i]
  y.append(count)
```

```
y_pred=[]
for i in range(len(x_test)):
  p=len(x_trainh)/(len(x_trainh)+len(x_trains))
  q=len(x_trains)/(len(x_trainh)+len(x_trains))
  for j in range(len(x_test[i])):
    p*=((x[j]+1)/(c+len(x_test[i])))**x_test[i][j]
    q*=((y[j]+1)/(c+len(x_test[i])))**x_test[i][j]
  if p>q:
    y_pred.append("ham")
  else:
    y_pred.append("spam")
```

```
cm = confusion_matrix(y_test, y_pred)  # y_test and y_pred were swapped in your original call

# Access the values by their index
tn, fp, fn, tp = cm.ravel()

# Calculate metrics
ac = (tp + tn) / (tp + tn + fp + fn)
pr = tp / (tp + fp)
r = tp / (tp + fn)
f1 = (2 * pr * r) / (pr + r)

# Append the calculated values to respective lists
accu.append(ac)
pre.append(pr)
re.append(r)
f1_scores.append(f1)
```

```
from sklearn.metrics import accuracy_score
mh=np.mean(x_trainh,axis=0)
ms=np.mean(x_trains,axis=0)

sh=np.std(x_trainh,axis=0)
ss=np.std(x_trains,axis=0)

zh=(x_test-mh)/sh
zs=(x_test-ms)/ss

p=np.exp(-0.5*((zh)**2))/(np.sqrt(2*np.pi)*sh)
q=np.exp(-0.5*((zs)**2))/(np.sqrt(2*np.pi)*ss)

n_ham=len(y_train[y_train=='ham'])/len(y_train)
n_spam=len(y_train[y_train=='spam'])/len(y_train)

p_ham=np.prod(p,axis=1)*n_ham
p_spam=np.prod(q,axis=1)*n_spam
result = np.where(p_ham > p_spam, 'ham', 'spam')
y_pred=result

cm = confusion_matrix(y_test, y_pred)  # y_test and y_pred were swapped in your original cal

# Access the values by their index
tn, fp, fn, tp = cm.ravel()

# Calculate metrics
```

```
ac = (tp + tn) / (tp + tn + fp + fn)
pr = tp / (tp + fp)
r = tp / (tp + fn)
f1 = (2 * pr * r) / (pr + r)

# Append the calculated values to respective lists
accu.append(ac)
pre.append(pr)
re.append(r)
```

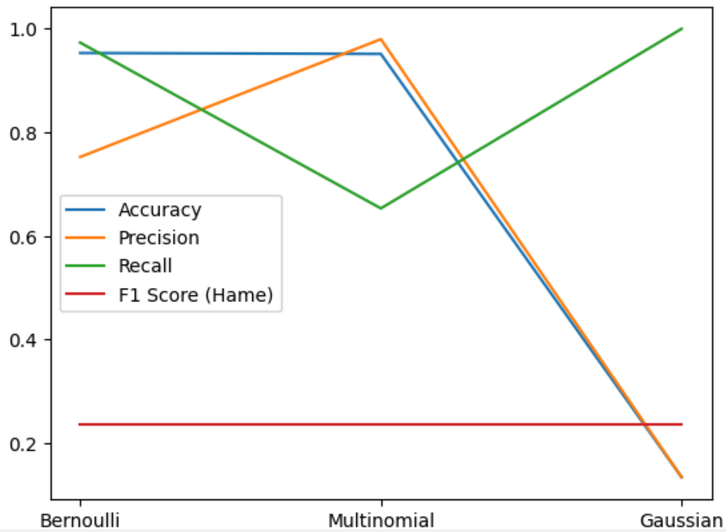
```
<ipython-input-44-618831411234>:8: RuntimeWarning: divide by zero encountered in divide
  zh=(x_test-mh)/sh
<ipython-input-44-618831411234>:8: RuntimeWarning: invalid value encountered in divide
  zh=(x_test-mh)/sh
<ipython-input-44-618831411234>:9: RuntimeWarning: divide by zero encountered in divide
  zs=(x_test-ms)/ss
<ipython-input-44-618831411234>:9: RuntimeWarning: invalid value encountered in divide
  zs=(x_test-ms)/ss
<ipython-input-44-618831411234>:11: RuntimeWarning: invalid value encountered in divide
  p=np.exp(-0.5*((zh)**2))/(np.sqrt(2*np.pi)*sh)
<ipython-input-44-618831411234>:12: RuntimeWarning: invalid value encountered in divide
  q=np.exp(-0.5*((zs)**2))/(np.sqrt(2*np.pi)*ss)
```

```
import seaborn as sns
import matplotlib.pyplot as plt

h = ["Bernoulli", "Multinomial", "Gaussian"]
# Create the plot
sns.lineplot(x=h, y=accu, label="Accuracy")
sns.lineplot(x=h, y=pre, label="Precision")
sns.lineplot(x=h, y=re, label="Recall")
sns.lineplot(x=h, y=f1, label="F1 Score (Hame)")

# Show the plot
plt.show()

```




```
Start coding or generate with AI.
```