Experiment 3

Email Spam or Ham Classification using Naive Bayes, KNN, and SVM

Name: Vigneshwaran S

Reg no:3122237001059

1 Aim:

To develop and evaluate supervised machine learning models using K-Nearest Neighbors (KNN)

and Naive Bayes (Bernoulli, Multinomial, Gaussian) classifiers. To analyze and compare the performance

of these models on a given dataset using accuracy and other evaluation metrics.

2 Libraries used:

- Numpy
- Pandas
- Matplotlib
- Scikit-learn
- Seaborn

3 Objective:

- To implement KNN and Naive Bayes algorithms for classification, including preprocessing,

training, and testing using appropriate techniques such as cross-validation.

- To assess the models using metrics like accuracy, precision, recall, F1-score, confusion matrix,

ROC curve, and AUC, and to present a comparative analysis.

In [18]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Load dataset

df = pd.read_csv("Dataset/spambase_csv.csv")
print(df.shape)
print(df.head())
print(df.describe())
```

```
(4601, 58)
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0            0.00               0.64           0.64           0.0
1            0.21               0.28           0.50           0.0
2            0.06               0.00           0.71           0.0
3            0.00               0.00           0.00           0.0
4            0.00               0.00           0.00           0.0

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0           0.32            0.00              0.00                0.00
1           0.14            0.28              0.21                0.07
2           1.23            0.19              0.19                0.12
3           0.63            0.00              0.31                0.63
4           0.63            0.00              0.31                0.63

   word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0             0.00            0.00  ...           0.00          0.000
1             0.00            0.94  ...           0.00          0.132
2             0.64            0.25  ...           0.01          0.143
3             0.31            0.63  ...           0.00          0.137
4             0.31            0.63  ...           0.00          0.135

   char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0            0.0          0.778          0.000          0.000
1            0.0          0.372          0.180          0.048
2            0.0          0.276          0.184          0.010
3            0.0          0.137          0.000          0.000
4            0.0          0.135          0.000          0.000

   capital_run_length_average  capital_run_length_longest  \
0                       3.756                          61
1                       5.114                         101
2                       9.821                         485
3                       3.537                          40
4                       3.537                          40

   capital_run_length_total  class
0                       278      1
1                      1028      1
2                      2259      1
3                       191      1
4                       191      1

[5 rows x 58 columns]
       word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
count     4601.000000        4601.000000    4601.000000   4601.000000
mean         0.104553           0.213015       0.280656      0.065425
std          0.305358           1.290575       0.504143      1.395151
min          0.000000           0.000000       0.000000      0.000000
25%          0.000000           0.000000       0.000000      0.000000
50%          0.000000           0.000000       0.000000      0.000000
75%          0.000000           0.000000       0.420000      0.000000
max          4.540000          14.280000       5.100000     42.810000

       word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
count    4601.000000     4601.000000       4601.000000         4601.000000
mean        0.312223        0.095901          0.114208            0.105295
std         0.672513        0.273824          0.391441            0.401071
min         0.000000        0.000000          0.000000            0.000000
25%         0.000000        0.000000          0.000000            0.000000
```

```
50%         0.000000        0.000000          0.000000            0.000000
75%         0.380000        0.000000          0.000000            0.000000
max        10.000000        5.880000          7.270000           11.110000

       word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
count      4601.000000     4601.000000  ...    4601.000000    4601.000000
mean          0.090067        0.239413  ...       0.038575       0.139030
std           0.278616        0.644755  ...       0.243471       0.270355
min           0.000000        0.000000  ...       0.000000       0.000000
25%           0.000000        0.000000  ...       0.000000       0.000000
50%           0.000000        0.000000  ...       0.000000       0.065000
75%           0.000000        0.160000  ...       0.000000       0.188000
max           5.260000       18.180000  ...       4.385000       9.752000

       char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
count    4601.000000    4601.000000    4601.000000    4601.000000
mean        0.016976       0.269071       0.075811       0.044238
std         0.109394       0.815672       0.245882       0.429342
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.315000       0.052000       0.000000
max         4.081000      32.478000       6.003000      19.829000

       capital_run_length_average  capital_run_length_longest  \
count                 4601.000000                 4601.000000
mean                     5.191515                   52.172789
std                     31.729449                  194.891310
min                      1.000000                    1.000000
25%                      1.588000                    6.000000
50%                      2.276000                   15.000000
75%                      3.706000                   43.000000
max                   1102.500000                 9989.000000

       capital_run_length_total        class
count               4601.000000  4601.000000
mean                 283.289285     0.394045
std                  606.347851     0.488698
min                    1.000000     0.000000
25%                   35.000000     0.000000
50%                   95.000000     0.000000
75%                  266.000000     1.000000
max                15841.000000     1.000000

[8 rows x 58 columns]
```

In [19]:
```
#Missing Values

print(df.isnull().sum())
df.fillna(df.mean(), inplace=True)
```

```
word_freq_make               0
word_freq_address            0
word_freq_all                0
word_freq_3d                 0
word_freq_our                0
word_freq_over               0
word_freq_remove             0
word_freq_internet           0
word_freq_order              0
word_freq_mail               0
word_freq_receive            0
word_freq_will               0
word_freq_people             0
word_freq_report             0
word_freq_addresses          0
word_freq_free               0
word_freq_business           0
word_freq_email              0
word_freq_you                0
word_freq_credit             0
word_freq_your               0
word_freq_font               0
word_freq_000                0
word_freq_money              0
word_freq_hp                 0
word_freq_hpl                0
word_freq_george             0
word_freq_650                0
word_freq_lab                0
word_freq_labs               0
word_freq_telnet             0
word_freq_857                0
word_freq_data               0
word_freq_415                0
word_freq_85                 0
word_freq_technology         0
word_freq_1999               0
word_freq_parts              0
word_freq_pm                 0
word_freq_direct             0
word_freq_cs                 0
word_freq_meeting            0
word_freq_original           0
word_freq_project            0
word_freq_re                 0
word_freq_edu                0
word_freq_table              0
word_freq_conference         0
char_freq_%3B                0
char_freq_%28                0
char_freq_%5B                0
char_freq_%21                0
char_freq_%24                0
char_freq_%23                0
capital_run_length_average   0
capital_run_length_longest   0
capital_run_length_total     0
class                        0
dtype: int64
```

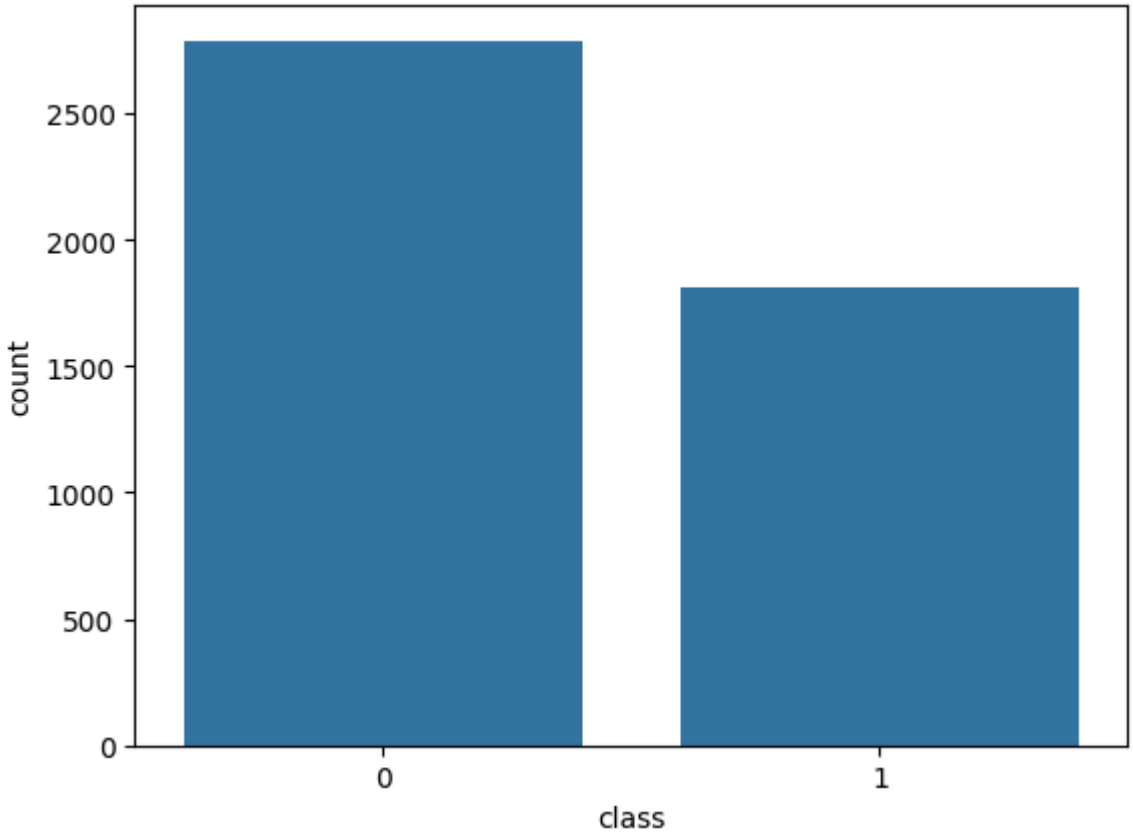
In [20]:
```
#Target

print(df['class'].nunique())
print(df['class'].unique())
```

```
2
[1 0]
```

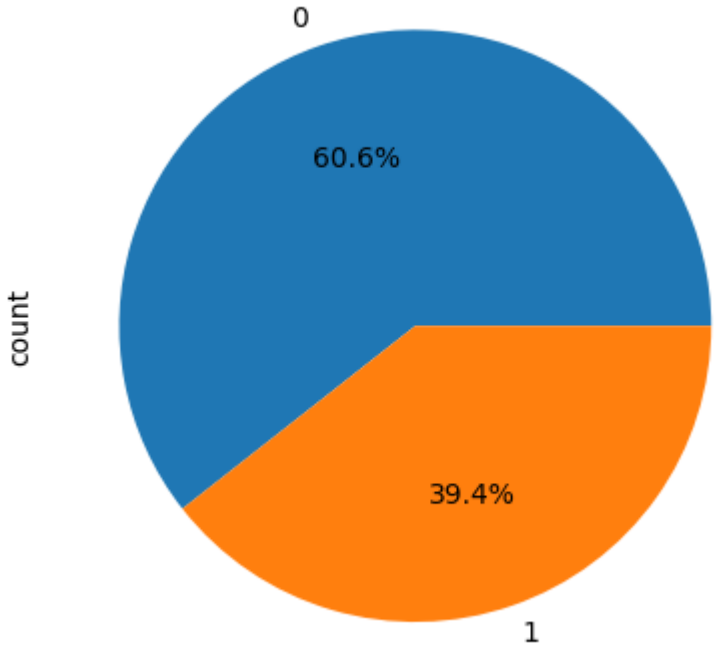
In [21]:
```
#Duplicate
df.duplicated().sum()
```

Out[21]: 391

In [22]:
```
#target Class Distribution

sns.countplot(x='class', data=df)
plt.show()
df['class'].value_counts().plot.pie(autopct='%1.1f%%')
```



Out[22]: <Axes: ylabel='count'>

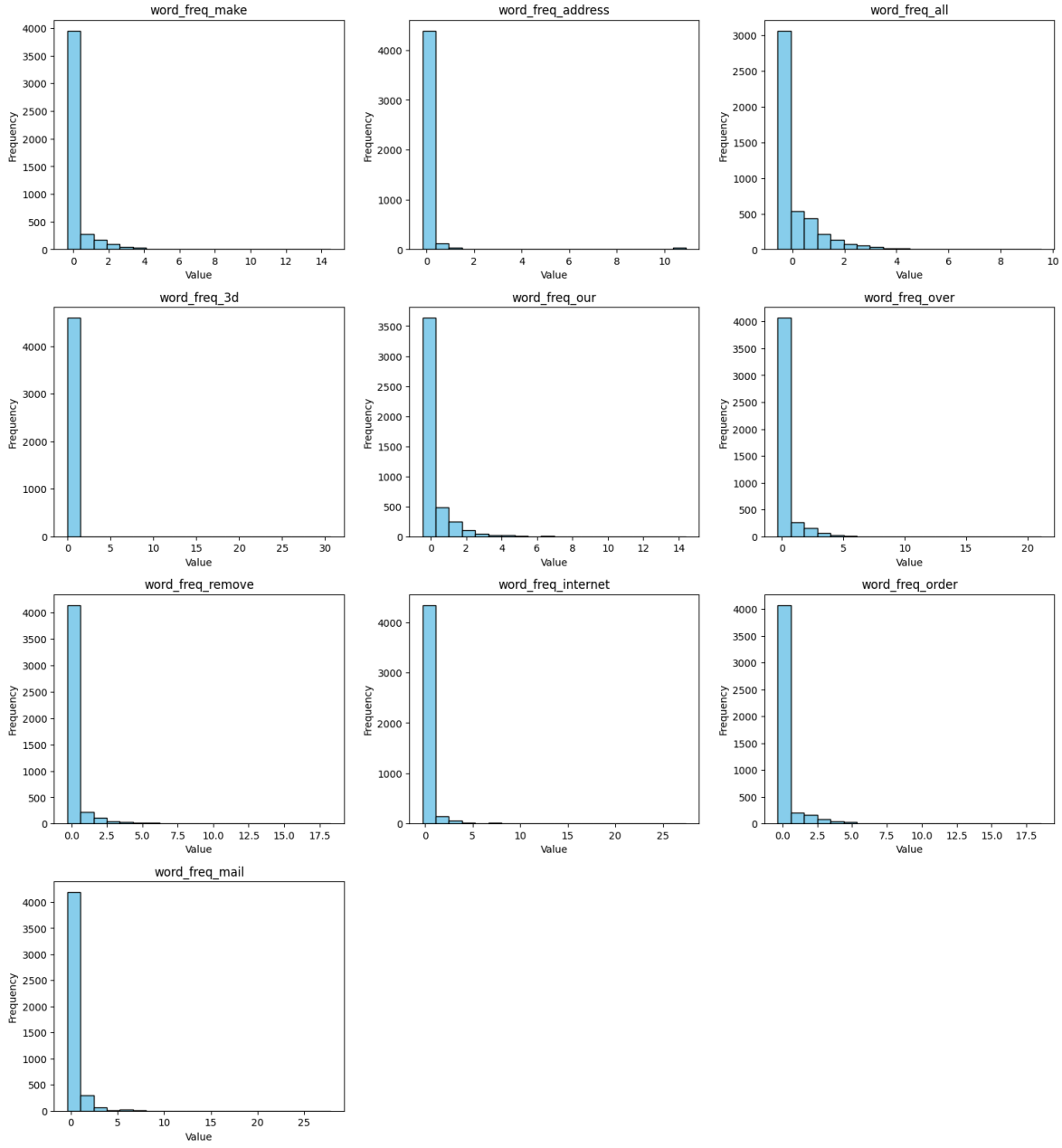


In [23]:
```
df.columns
```

Out[23]:
```
Index(['word_freq_make', 'word_freq_address', 'word_freq_all', 'word_freq_3d',
       'word_freq_our', 'word_freq_over', 'word_freq_remove',
       'word_freq_internet', 'word_freq_order', 'word_freq_mail',
       'word_freq_receive', 'word_freq_will', 'word_freq_people',
       'word_freq_report', 'word_freq_addresses', 'word_freq_free',
       'word_freq_business', 'word_freq_email', 'word_freq_you',
       'word_freq_credit', 'word_freq_your', 'word_freq_font', 'word_freq_000',
       'word_freq_money', 'word_freq_hp', 'word_freq_hpl', 'word_freq_george',
       'word_freq_650', 'word_freq_lab', 'word_freq_labs', 'word_freq_telnet',
       'word_freq_857', 'word_freq_data', 'word_freq_415', 'word_freq_85',
       'word_freq_technology', 'word_freq_1999', 'word_freq_parts',
       'word_freq_pm', 'word_freq_direct', 'word_freq_cs', 'word_freq_meeting',
       'word_freq_original', 'word_freq_project', 'word_freq_re',
       'word_freq_edu', 'word_freq_table', 'word_freq_conference',
       'char_freq_%3B', 'char_freq_%28', 'char_freq_%5B', 'char_freq_%21',
       'char_freq_%24', 'char_freq_%23', 'capital_run_length_average',
       'capital_run_length_longest', 'capital_run_length_total', 'class'],
      dtype='object')
```

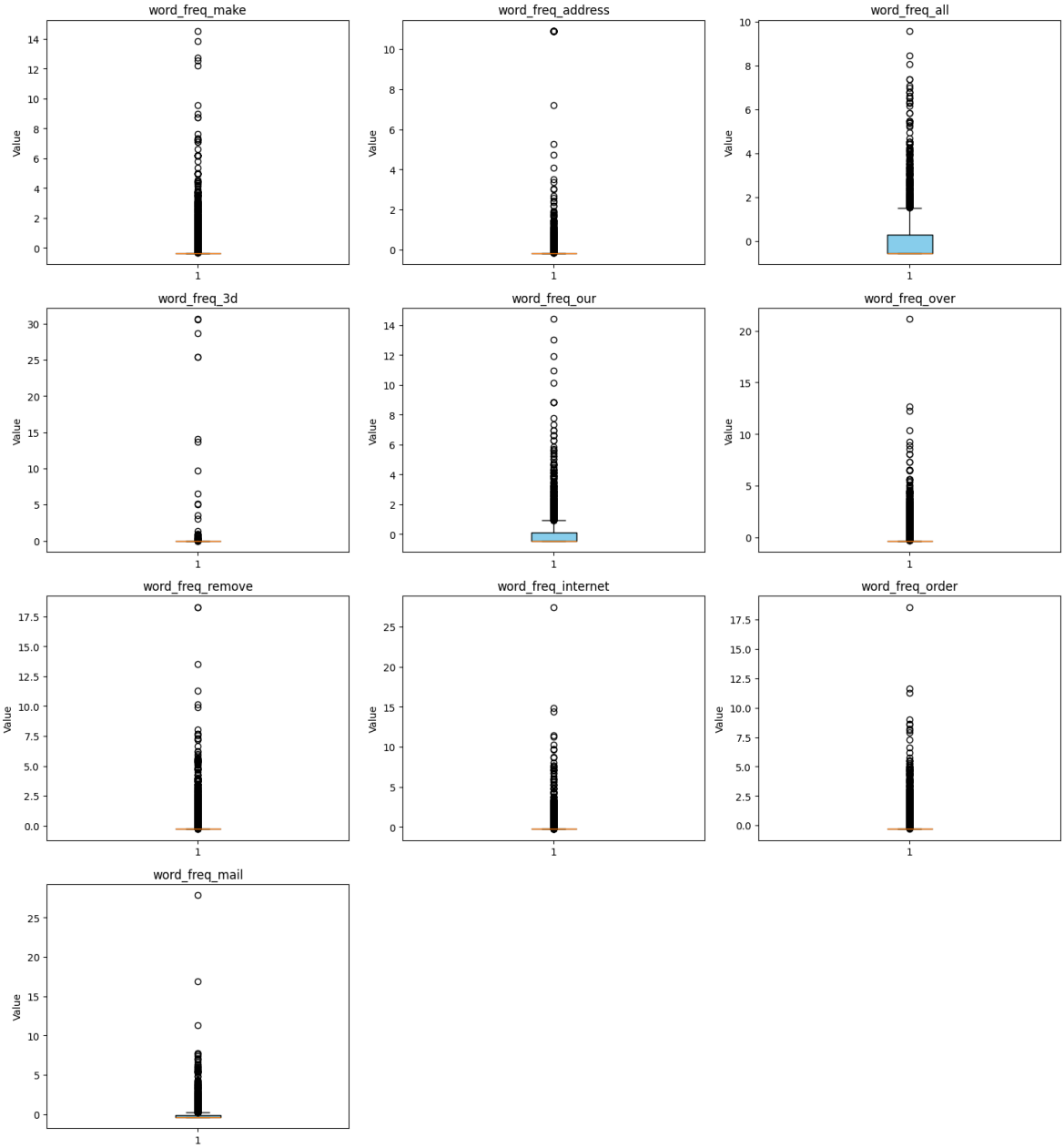
In [24]:
```
#Numeric Features
#Histogram

num_cols = len(df.columns)


n_cols = 3
n_rows = (num_cols + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(df.columns[:10]):
    axes[i].hist(df[col].dropna(), bins=20, color='skyblue', edgecolor='black')
    axes[i].set_title(col)
```

```
    axes[i].set_ylabel('Frequency')
    axes[i].set_xlabel('Value')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



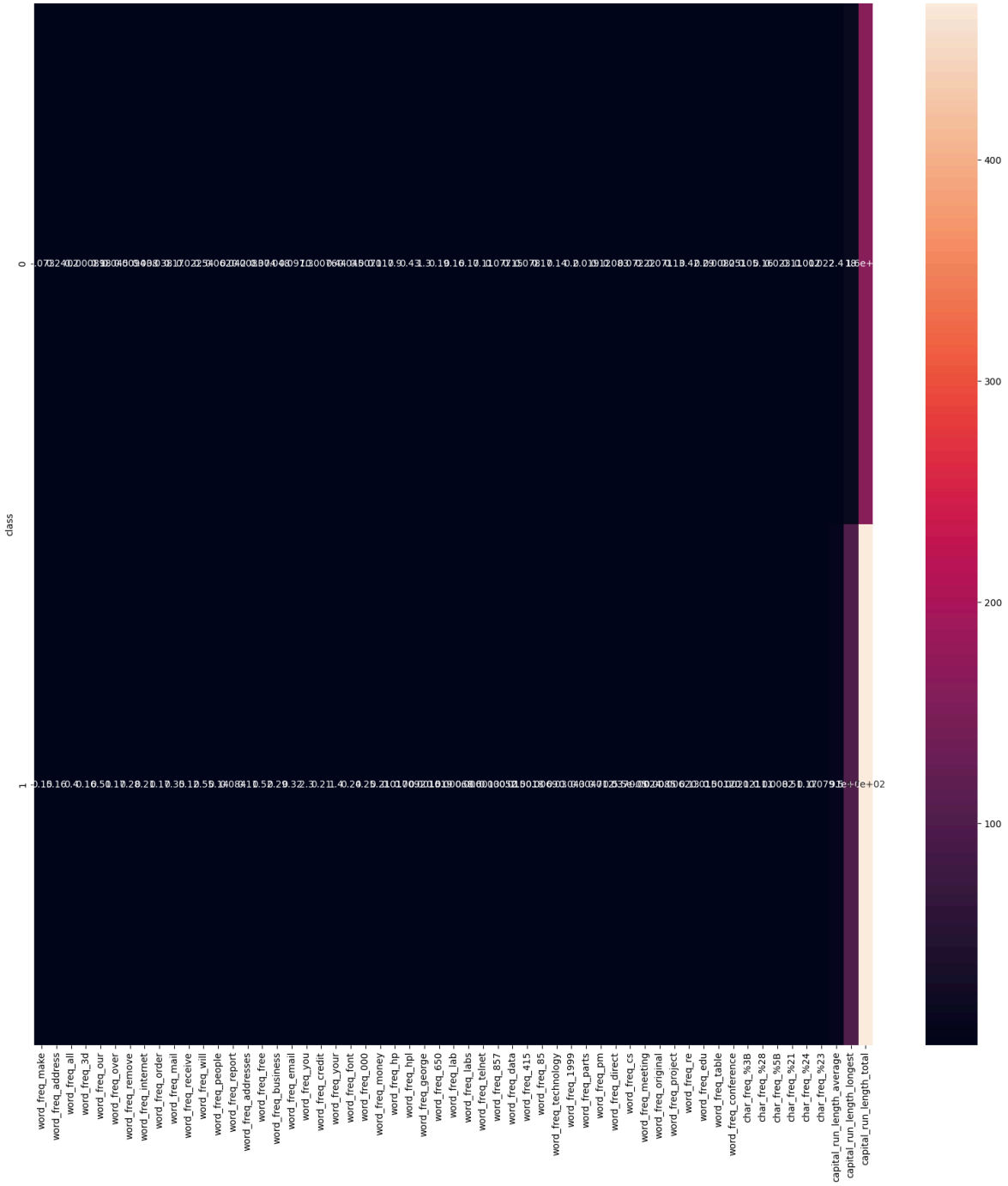
In [25]:
```
#Boxplot

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
num_cols = len(numeric_cols)

# Define subplot grid size
n_cols = 3  # number of plots per row
n_rows = (num_cols + n_cols - 1) // n_cols  # ceiling division

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))
axes = axes.flatten()
```

```
# Plot boxplot for each numeric column
for i, col in enumerate(numeric_cols[:10]):
    axes[i].boxplot(df[col].dropna(), vert=True, patch_artist=True,
                    boxprops=dict(facecolor='skyblue'))
    axes[i].set_title(col)
    axes[i].set_ylabel('Value')

# Remove unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



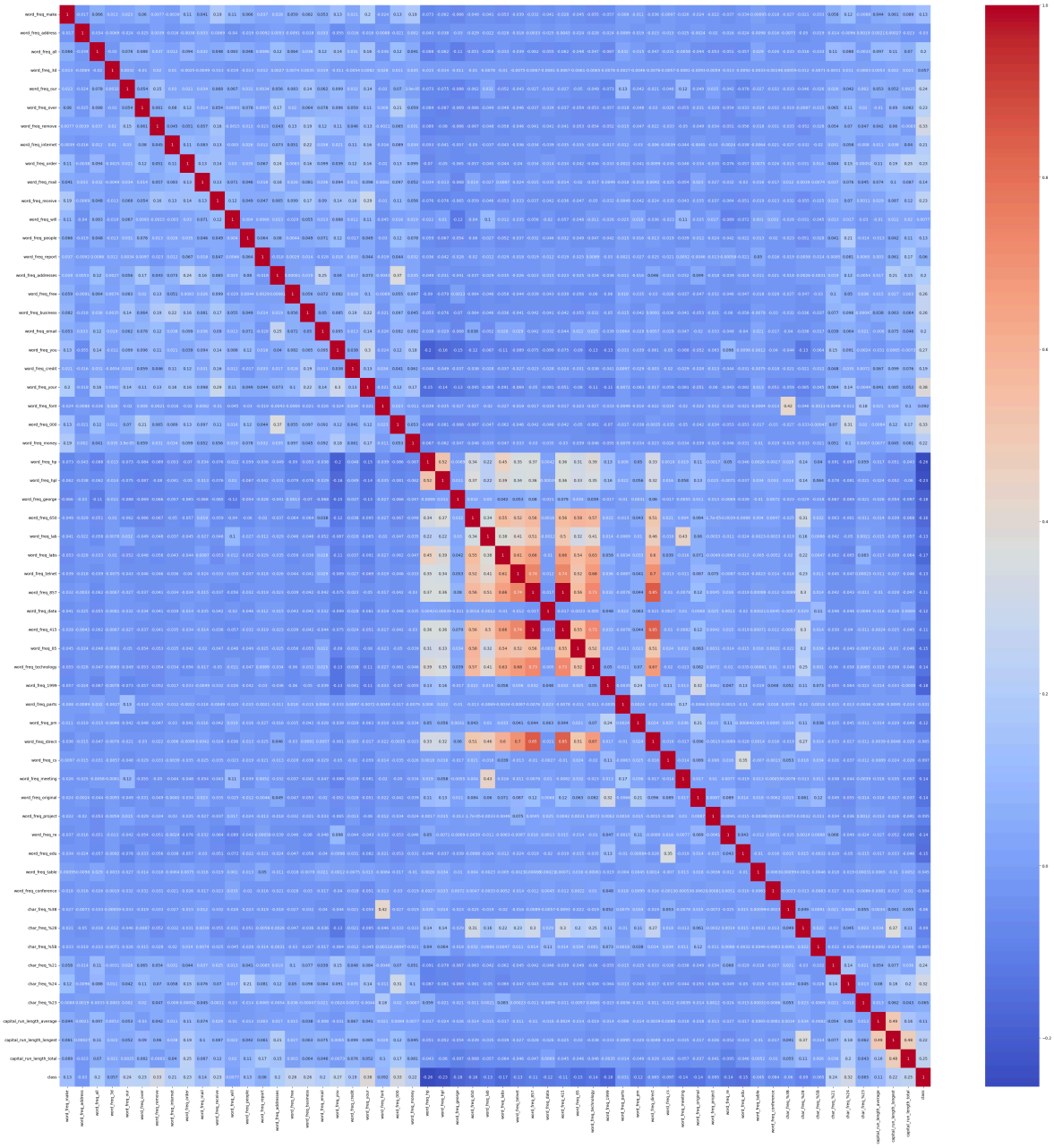
In [10]:
```
#Feature vs target relation
#HeattMap

plt.figure(figsize=(20,20))
sns.heatmap(df.groupby('class').mean(), annot=True)
```

Out[10]: <Axes: ylabel='class'>



In [11]:
```
#Correlation Heatmap

plt.figure(figsize=(50,50))
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[11]: <Axes: >



In [12]:
```
#Standardization

X = df.drop('class', axis=1)
y = df['class']

scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

df= pd.concat([X_scaled, y], axis=1)

print(df.head())
```

```
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0       -0.342434           0.330885       0.712859       -0.0469
1        0.345359           0.051909       0.435130       -0.0469
2       -0.145921          -0.165072       0.851723       -0.0469
3       -0.342434          -0.165072      -0.556761       -0.0469
4       -0.342434          -0.165072      -0.556761       -0.0469

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0       0.011565       -0.350266         -0.291794           -0.262562
1      -0.256117        0.672399          0.244743           -0.088010
2       1.364846        0.343685          0.193644            0.036670
3       0.472573       -0.350266          0.500237            1.308402
4       0.472573       -0.350266          0.500237            1.308402

   word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0       -0.323302       -0.371364  ...      -0.158453      -0.514307
1       -0.323302        1.086711  ...      -0.158453      -0.026007
2        1.974017        0.016422  ...      -0.117376       0.014684
3        0.789462        0.605857  ...      -0.158453      -0.007511
4        0.789462        0.605857  ...      -0.158453      -0.014910

   char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0      -0.155198       0.624007      -0.308355      -0.103048
1      -0.155198       0.126203       0.423783       0.008763
2      -0.155198       0.008496       0.440053      -0.079754
3      -0.155198      -0.161934      -0.308355      -0.103048
4      -0.155198      -0.164387      -0.308355      -0.103048

   capital_run_length_average  capital_run_length_longest  \
0                   -0.045247                    0.045298
1                   -0.002443                    0.250563
2                    0.145921                    2.221106
3                   -0.052150                   -0.062466
4                   -0.052150                   -0.062466

   capital_run_length_total  class
0                 -0.008724      1
1                  1.228324      1
2                  3.258733      1
3                 -0.152222      1
4                 -0.152222      1

[5 rows x 58 columns]
```

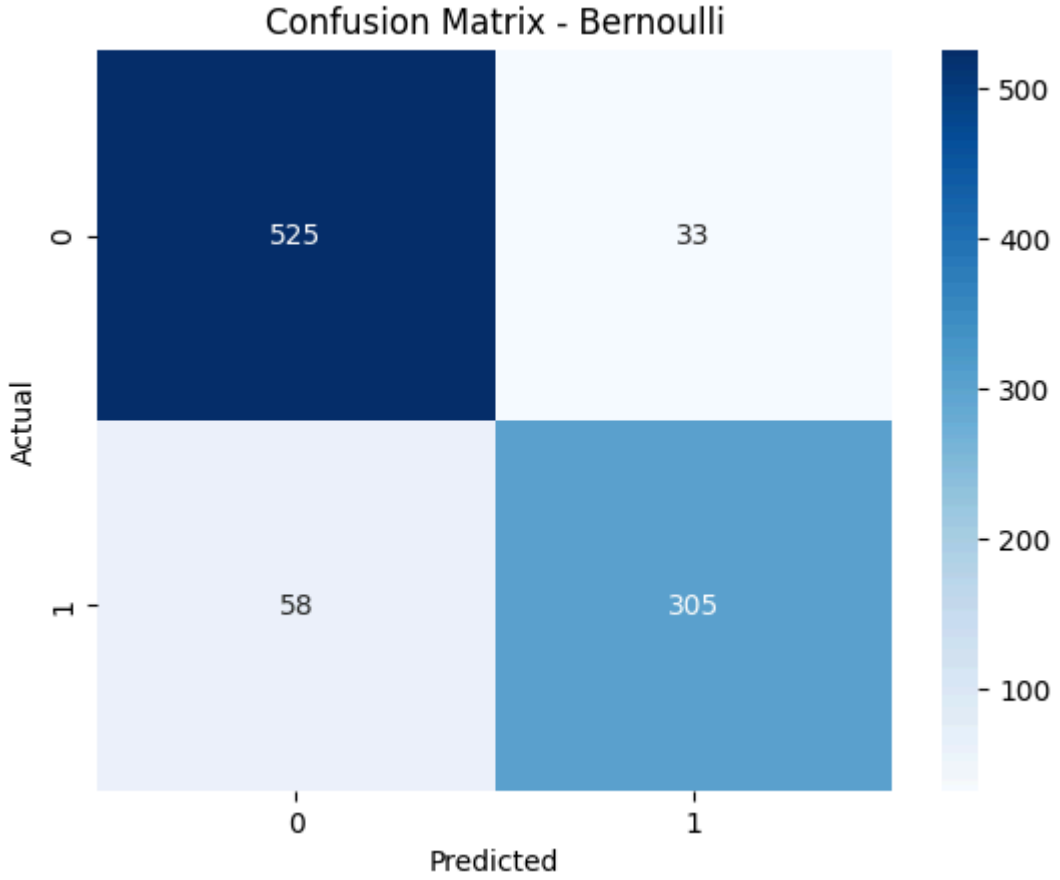
In [13]:
```
from sklearn.naive_bayes import BernoulliNB

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,

bnb = BernoulliNB()
bnb.fit(X_train, y_train)
y_pred = bnb.predict(X_test)
y_prob = bnb.predict_proba(X_test)[:, 1]
```

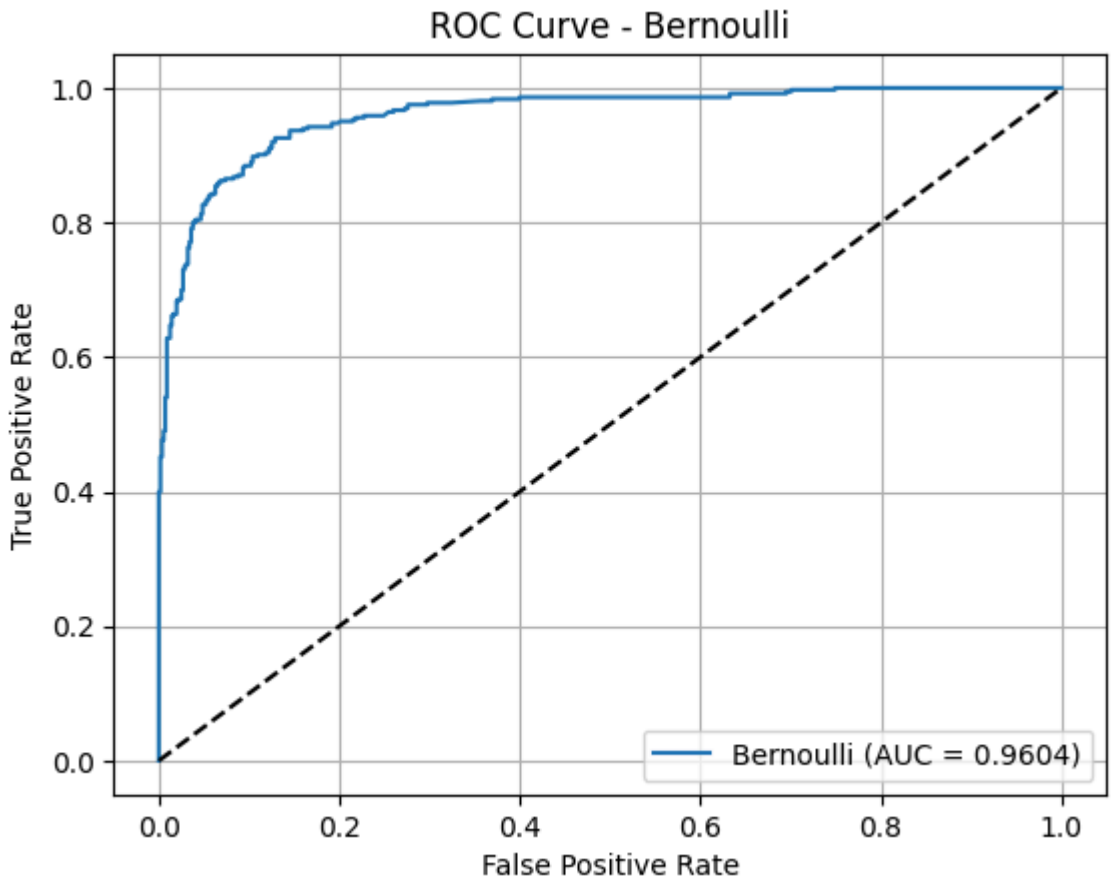
In [ ]:
```
from sklearn.naive_bayes import GaussianNB

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,

gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
y_pred = gnb.predict(X_test)
y_prob = gnb.predict_proba(X_test)[:, 1]
```

In [15]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

print(f"\n 'Bernoulli Results")
print(f"Accuracy:  {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall:    {rec:.4f}")
print(f"F1-Score:  {f1:.4f}")
print(f"AUC:       {roc_auc:.4f}")

    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix - Bernoulli')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

    # ROC Curve
plt.plot(fpr, tpr, label=f'Bernoulli (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.title(f'ROC Curve - Bernoulli')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```

```
 'Bernoulli Results
Accuracy:  0.9012
Precision: 0.9024
Recall:    0.8402
F1-Score:  0.8702
AUC:       0.9604
```





In [16]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score

print("----- K-Fold Accuracy Scores -----")
models = {
    "BernoulliNB": BernoulliNB()
```

```
}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in models.items():
    scores = cross_val_score(model, X_scaled, y, cv=kfold, scoring='accuracy')
    print(f"{name}: Mean Accuracy = {scores.mean():.4f}, Std = {scores.std():.4f
```

```
----- K-Fold Accuracy Scores -----
BernoulliNB: Mean Accuracy = 0.9022, Std = 0.0075
```

In [21]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
import matplotlib.pyplot as plt
import seaborn as sns
```

In [22]:
```
#Load dataset

df = pd.read_csv("Dataset/spambase_csv.csv")
print(df.shape)
print(df.head())
print(df.describe())
```

```
(4601, 58)
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0            0.00               0.64           0.64           0.0
1            0.21               0.28           0.50           0.0
2            0.06               0.00           0.71           0.0
3            0.00               0.00           0.00           0.0
4            0.00               0.00           0.00           0.0

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0           0.32            0.00              0.00                0.00
1           0.14            0.28              0.21                0.07
2           1.23            0.19              0.19                0.12
3           0.63            0.00              0.31                0.63
4           0.63            0.00              0.31                0.63

   word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0             0.00            0.00  ...           0.00          0.000
1             0.00            0.94  ...           0.00          0.132
2             0.64            0.25  ...           0.01          0.143
3             0.31            0.63  ...           0.00          0.137
4             0.31            0.63  ...           0.00          0.135

   char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0            0.0          0.778          0.000          0.000
1            0.0          0.372          0.180          0.048
2            0.0          0.276          0.184          0.010
3            0.0          0.137          0.000          0.000
4            0.0          0.135          0.000          0.000

   capital_run_length_average  capital_run_length_longest  \
0                       3.756                          61
1                       5.114                         101
2                       9.821                         485
3                       3.537                          40
4                       3.537                          40

   capital_run_length_total  class
0                       278      1
1                      1028      1
2                      2259      1
3                       191      1
4                       191      1

[5 rows x 58 columns]
       word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
count     4601.000000        4601.000000    4601.000000   4601.000000
mean         0.104553           0.213015       0.280656      0.065425
std          0.305358           1.290575       0.504143      1.395151
min          0.000000           0.000000       0.000000      0.000000
25%          0.000000           0.000000       0.000000      0.000000
50%          0.000000           0.000000       0.000000      0.000000
75%          0.000000           0.000000       0.420000      0.000000
max          4.540000          14.280000       5.100000     42.810000

       word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
count    4601.000000     4601.000000       4601.000000         4601.000000
mean        0.312223        0.095901          0.114208            0.105295
std         0.672513        0.273824          0.391441            0.401071
min         0.000000        0.000000          0.000000            0.000000
25%         0.000000        0.000000          0.000000            0.000000
```

```
50%         0.000000       0.000000          0.000000           0.000000
75%         0.380000       0.000000          0.000000           0.000000
max        10.000000       5.880000          7.270000          11.110000

       word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
count      4601.000000     4601.000000  ...    4601.000000    4601.000000
mean          0.090067        0.239413  ...       0.038575       0.139030
std           0.278616        0.644755  ...       0.243471       0.270355
min           0.000000        0.000000  ...       0.000000       0.000000
25%           0.000000        0.000000  ...       0.000000       0.000000
50%           0.000000        0.000000  ...       0.000000       0.065000
75%           0.000000        0.160000  ...       0.000000       0.188000
max           5.260000       18.180000  ...       4.385000       9.752000

       char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
count    4601.000000    4601.000000    4601.000000    4601.000000
mean        0.016976       0.269071       0.075811       0.044238
std         0.109394       0.815672       0.245882       0.429342
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.315000       0.052000       0.000000
max         4.081000      32.478000       6.003000      19.829000

       capital_run_length_average  capital_run_length_longest  \
count                 4601.000000                 4601.000000
mean                     5.191515                   52.172789
std                     31.729449                  194.891310
min                      1.000000                    1.000000
25%                      1.588000                    6.000000
50%                      2.276000                   15.000000
75%                      3.706000                   43.000000
max                   1102.500000                 9989.000000

       capital_run_length_total        class
count               4601.000000  4601.000000
mean                 283.289285     0.394045
std                  606.347851     0.488698
min                    1.000000     0.000000
25%                   35.000000     0.000000
50%                   95.000000     0.000000
75%                  266.000000     1.000000
max                15841.000000     1.000000

[8 rows x 58 columns]
```

In [23]:
```
#Missing Values

print(df.isnull().sum())
df.fillna(df.mean(), inplace=True)
```

```
word_freq_make               0
word_freq_address            0
word_freq_all                0
word_freq_3d                 0
word_freq_our                0
word_freq_over               0
word_freq_remove             0
word_freq_internet           0
word_freq_order              0
word_freq_mail               0
word_freq_receive            0
word_freq_will               0
word_freq_people             0
word_freq_report             0
word_freq_addresses          0
word_freq_free               0
word_freq_business           0
word_freq_email              0
word_freq_you                0
word_freq_credit             0
word_freq_your               0
word_freq_font               0
word_freq_000                0
word_freq_money              0
word_freq_hp                 0
word_freq_hpl                0
word_freq_george             0
word_freq_650                0
word_freq_lab                0
word_freq_labs               0
word_freq_telnet             0
word_freq_857                0
word_freq_data               0
word_freq_415                0
word_freq_85                 0
word_freq_technology         0
word_freq_1999               0
word_freq_parts              0
word_freq_pm                 0
word_freq_direct             0
word_freq_cs                 0
word_freq_meeting            0
word_freq_original           0
word_freq_project            0
word_freq_re                 0
word_freq_edu                0
word_freq_table              0
word_freq_conference         0
char_freq_%3B                0
char_freq_%28                0
char_freq_%5B                0
char_freq_%21                0
char_freq_%24                0
char_freq_%23                0
capital_run_length_average   0
capital_run_length_longest   0
capital_run_length_total     0
class                        0
dtype: int64
```

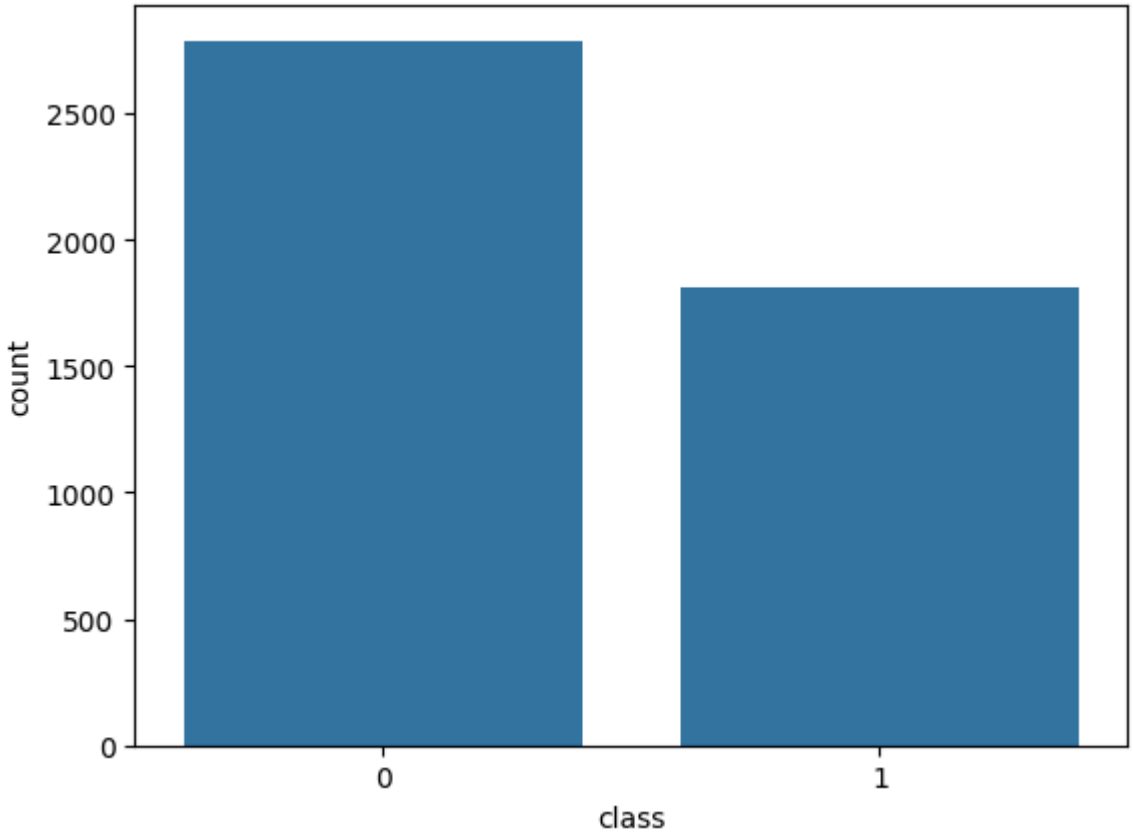
In [24]:
```
#Target

print(df['class'].nunique())
print(df['class'].unique())
```

```
2
[1 0]
```

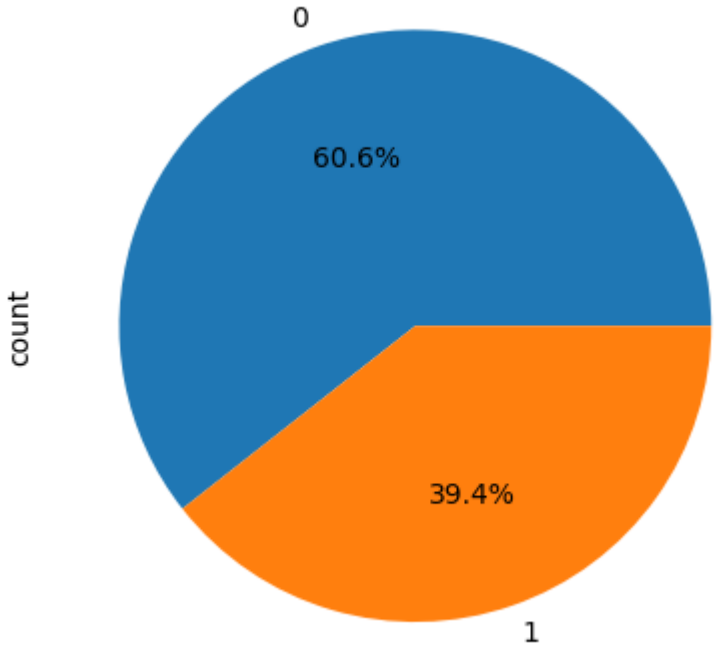
In [25]:
```
#Duplicate
df.duplicated().sum()
```

Out[25]: 391

In [26]:
```
#target Class Distribution

sns.countplot(x='class', data=df)
plt.show()
df['class'].value_counts().plot.pie(autopct='%1.1f%%')
```



Out[26]: <Axes: ylabel='count'>

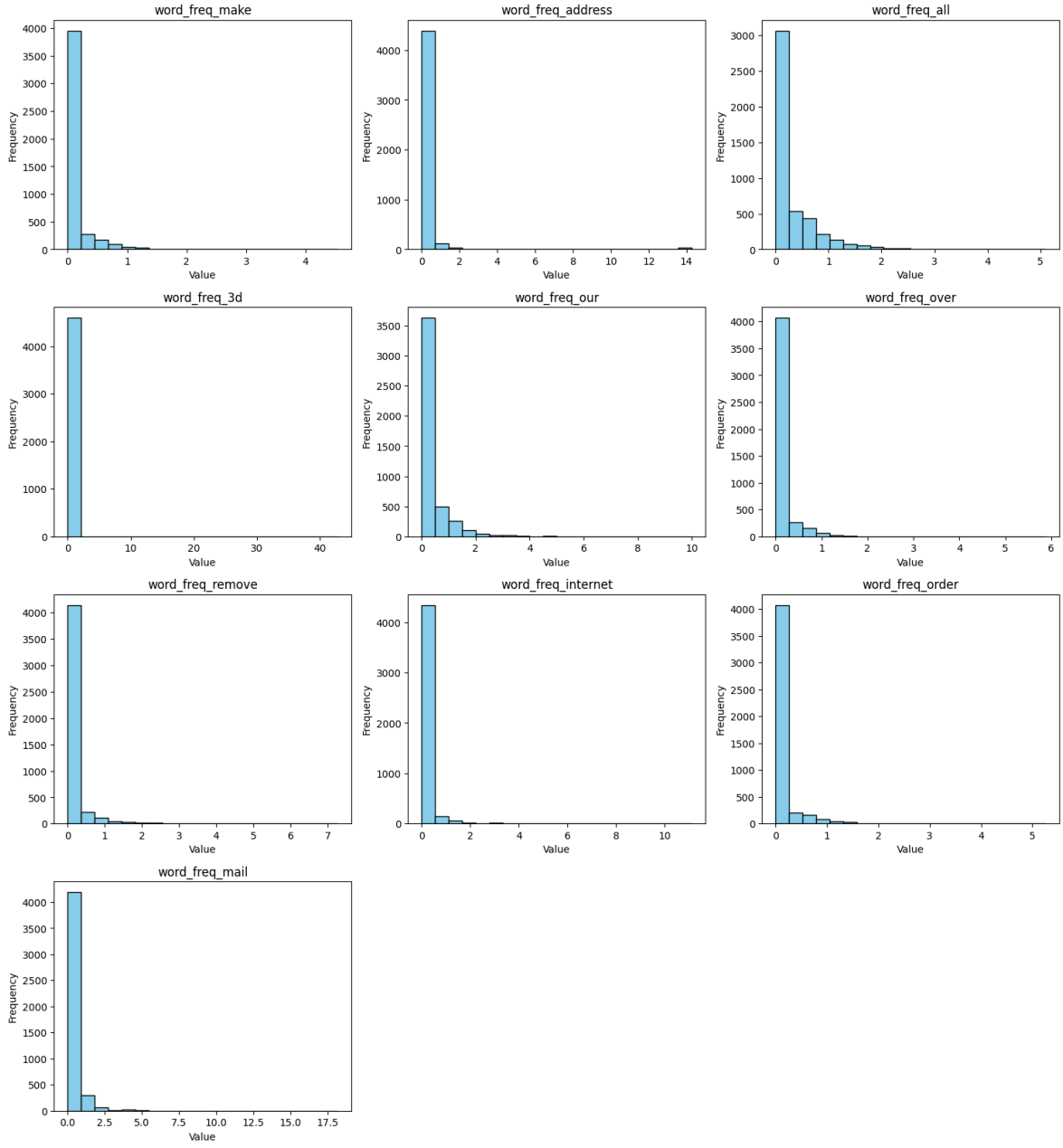


In [27]:
```
df.columns
```

Out[27]:
```
Index(['word_freq_make', 'word_freq_address', 'word_freq_all', 'word_freq_3d',
       'word_freq_our', 'word_freq_over', 'word_freq_remove',
       'word_freq_internet', 'word_freq_order', 'word_freq_mail',
       'word_freq_receive', 'word_freq_will', 'word_freq_people',
       'word_freq_report', 'word_freq_addresses', 'word_freq_free',
       'word_freq_business', 'word_freq_email', 'word_freq_you',
       'word_freq_credit', 'word_freq_your', 'word_freq_font', 'word_freq_000',
       'word_freq_money', 'word_freq_hp', 'word_freq_hpl', 'word_freq_george',
       'word_freq_650', 'word_freq_lab', 'word_freq_labs', 'word_freq_telnet',
       'word_freq_857', 'word_freq_data', 'word_freq_415', 'word_freq_85',
       'word_freq_technology', 'word_freq_1999', 'word_freq_parts',
       'word_freq_pm', 'word_freq_direct', 'word_freq_cs', 'word_freq_meeting',
       'word_freq_original', 'word_freq_project', 'word_freq_re',
       'word_freq_edu', 'word_freq_table', 'word_freq_conference',
       'char_freq_%3B', 'char_freq_%28', 'char_freq_%5B', 'char_freq_%21',
       'char_freq_%24', 'char_freq_%23', 'capital_run_length_average',
       'capital_run_length_longest', 'capital_run_length_total', 'class'],
      dtype='object')
```

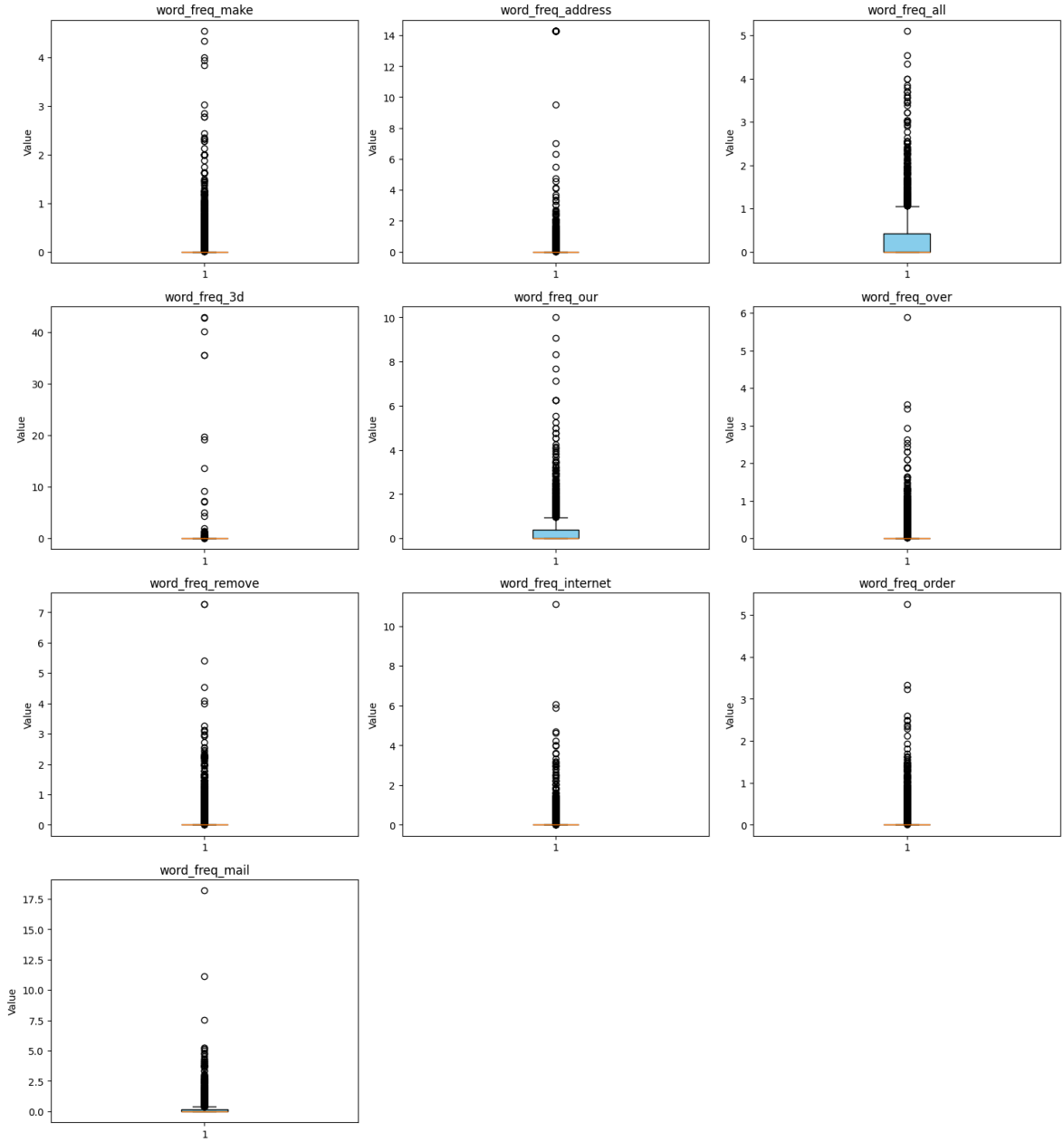
In [28]:
```
#Numeric Features
#Histogram

num_cols = len(df.columns)


n_cols = 3
n_rows = (num_cols + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(df.columns[:10]):
    axes[i].hist(df[col].dropna(), bins=20, color='skyblue', edgecolor='black')
    axes[i].set_title(col)
```

```
    axes[i].set_ylabel('Frequency')
    axes[i].set_xlabel('Value')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



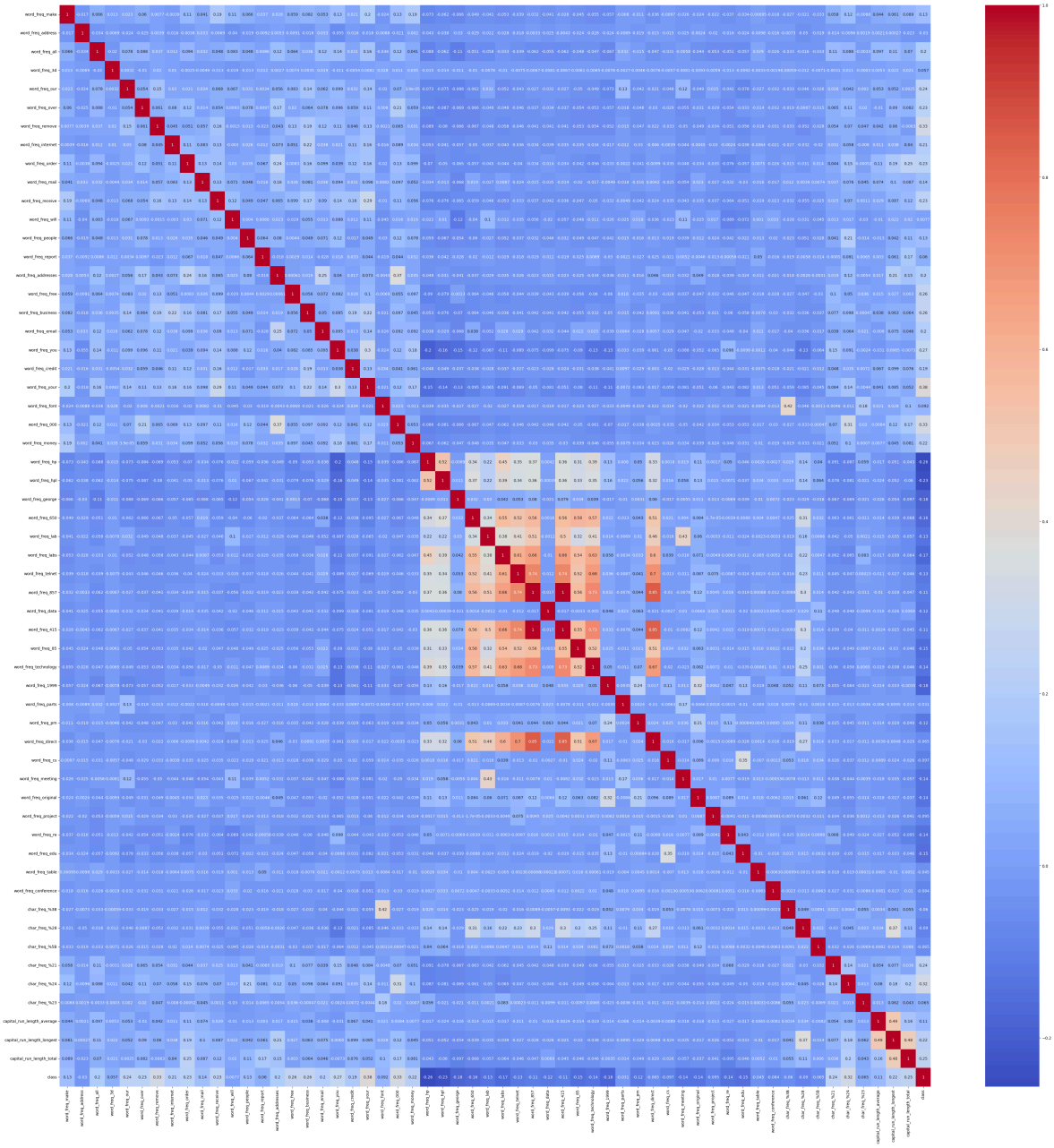
In [29]:
```
#Boxplot

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
num_cols = len(numeric_cols)

# Define subplot grid size
n_cols = 3  # number of plots per row
n_rows = (num_cols + n_cols - 1) // n_cols  # ceiling division

# Create subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))
axes = axes.flatten()
```

```
# Plot boxplot for each numeric column
for i, col in enumerate(numeric_cols[:10]):
    axes[i].boxplot(df[col].dropna(), vert=True, patch_artist=True,
                    boxprops=dict(facecolor='skyblue'))
    axes[i].set_title(col)
    axes[i].set_ylabel('Value')

# Remove unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



In [14]:
```
#Correlation Heatmap

plt.figure(figsize=(50,50))
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[14]: <Axes: >



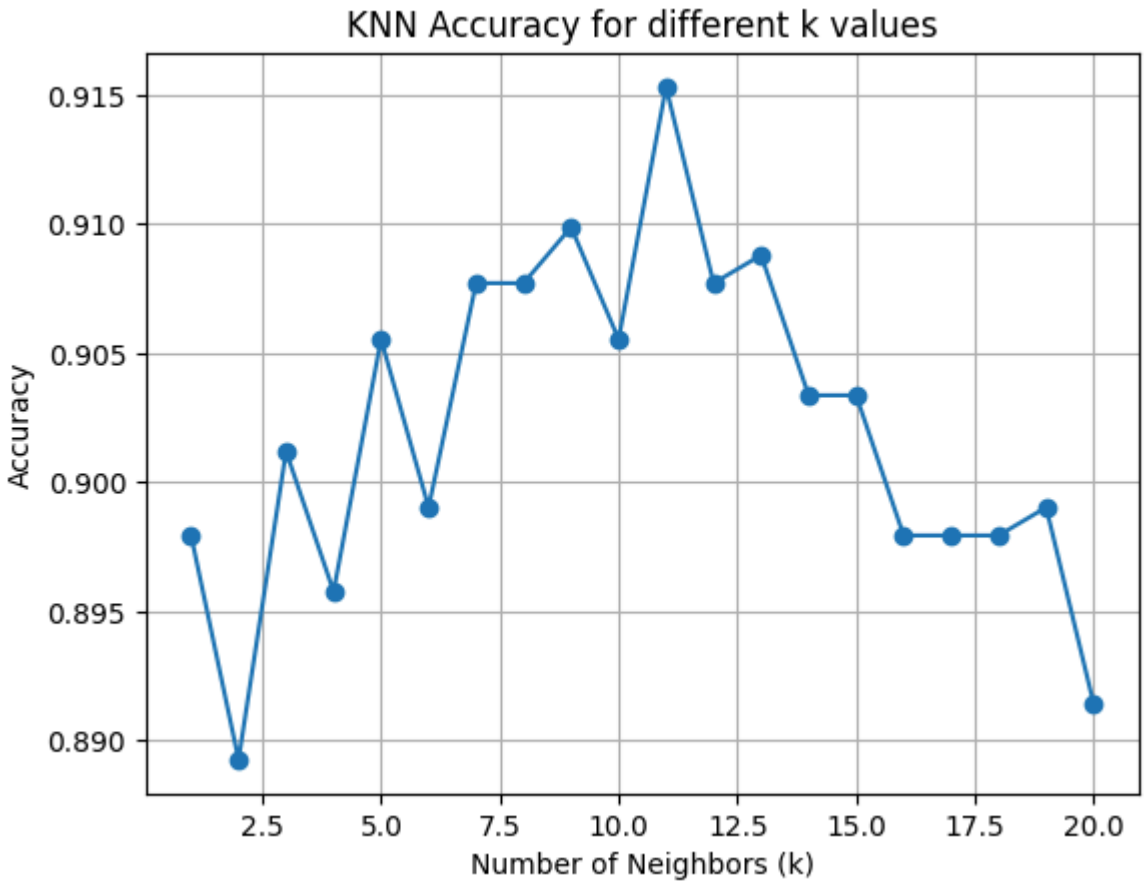
In [15]:
```
#Standardization

X = df.drop('class', axis=1)
y = df['class']

scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

df= pd.concat([X_scaled, y], axis=1)

print(df.head())
```

```
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0       -0.342434           0.330885       0.712859       -0.0469
1        0.345359           0.051909       0.435130       -0.0469
2       -0.145921          -0.165072       0.851723       -0.0469
3       -0.342434          -0.165072      -0.556761       -0.0469
4       -0.342434          -0.165072      -0.556761       -0.0469

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0       0.011565       -0.350266         -0.291794           -0.262562
1      -0.256117        0.672399          0.244743           -0.088010
2       1.364846        0.343685          0.193644            0.036670
3       0.472573       -0.350266          0.500237            1.308402
4       0.472573       -0.350266          0.500237            1.308402

   word_freq_order  word_freq_mail  ...  char_freq_%3B  char_freq_%28  \
0       -0.323302       -0.371364  ...      -0.158453      -0.514307
1       -0.323302        1.086711  ...      -0.158453      -0.026007
2        1.974017        0.016422  ...      -0.117376       0.014684
3        0.789462        0.605857  ...      -0.158453      -0.007511
4        0.789462        0.605857  ...      -0.158453      -0.014910

   char_freq_%5B  char_freq_%21  char_freq_%24  char_freq_%23  \
0      -0.155198       0.624007      -0.308355      -0.103048
1      -0.155198       0.126203       0.423783       0.008763
2      -0.155198       0.008496       0.440053      -0.079754
3      -0.155198      -0.161934      -0.308355      -0.103048
4      -0.155198      -0.164387      -0.308355      -0.103048

   capital_run_length_average  capital_run_length_longest  \
0                   -0.045247                    0.045298
1                   -0.002443                    0.250563
2                    0.145921                    2.221106
3                   -0.052150                   -0.062466
4                   -0.052150                   -0.062466

   capital_run_length_total  class
0                 -0.008724      1
1                  1.228324      1
2                  3.258733      1
3                 -0.152222      1
4                 -0.152222      1

[5 rows x 58 columns]
```

In [16]:

```python
#Model

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# -------------------------
# 1. Data Preprocessing
# -------------------------
X = df.drop('class', axis=1)
y = df['class']

# -------------------------
# 2. Train-Test Split
# -------------------------
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)


# -------------------------
# 4. KNN for varying k
# -------------------------
k_values = range(1, 21)  # vary k from 1 to 20
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

# -------------------------
# 5. Plot accuracy vs k
# -------------------------
plt.plot(k_values, accuracies, marker='o')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('KNN Accuracy for different k values')
plt.grid(True)
plt.show()

# -------------------------
# 6. Best k
# -------------------------
best_k = k_values[accuracies.index(max(accuracies))]
print(f"Best k = {best_k}, Accuracy = {max(accuracies):.4f}")
```



```
Best k = 11, Accuracy = 0.9153
```

In [17]:
```
knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print('k=11')
print('Accuracy:',acc)
```

```
k=11
Accuracy: 0.9153094462540716
```

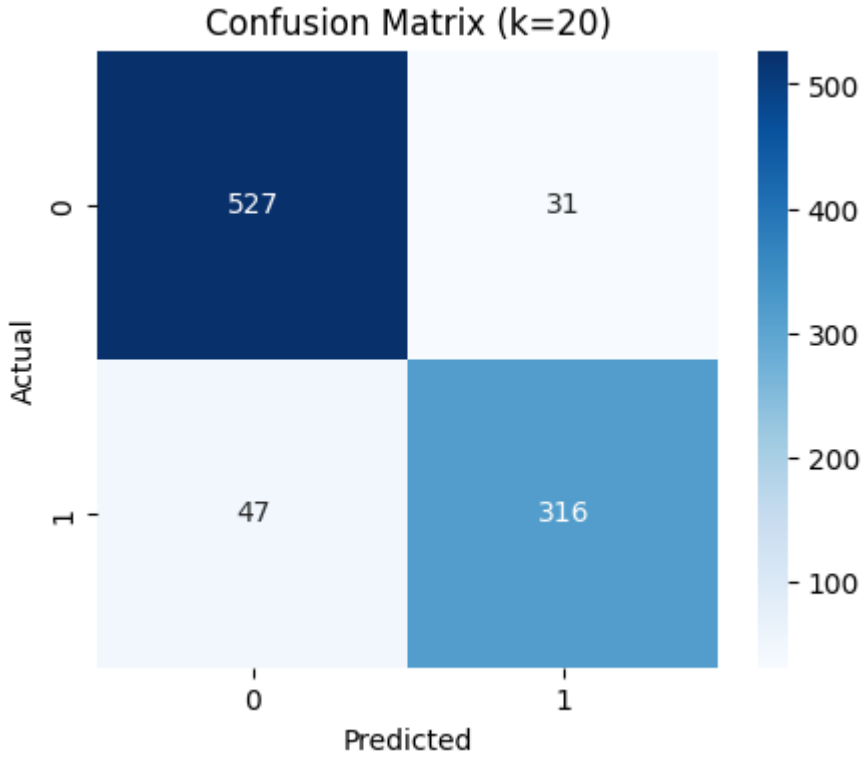
In [ ]:
```
algo='kd_tree'
k=11

knn = KNeighborsClassifier(n_neighbors=k, algorithm=algo)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print('k=11')
print('Accuracy:',acc)
```

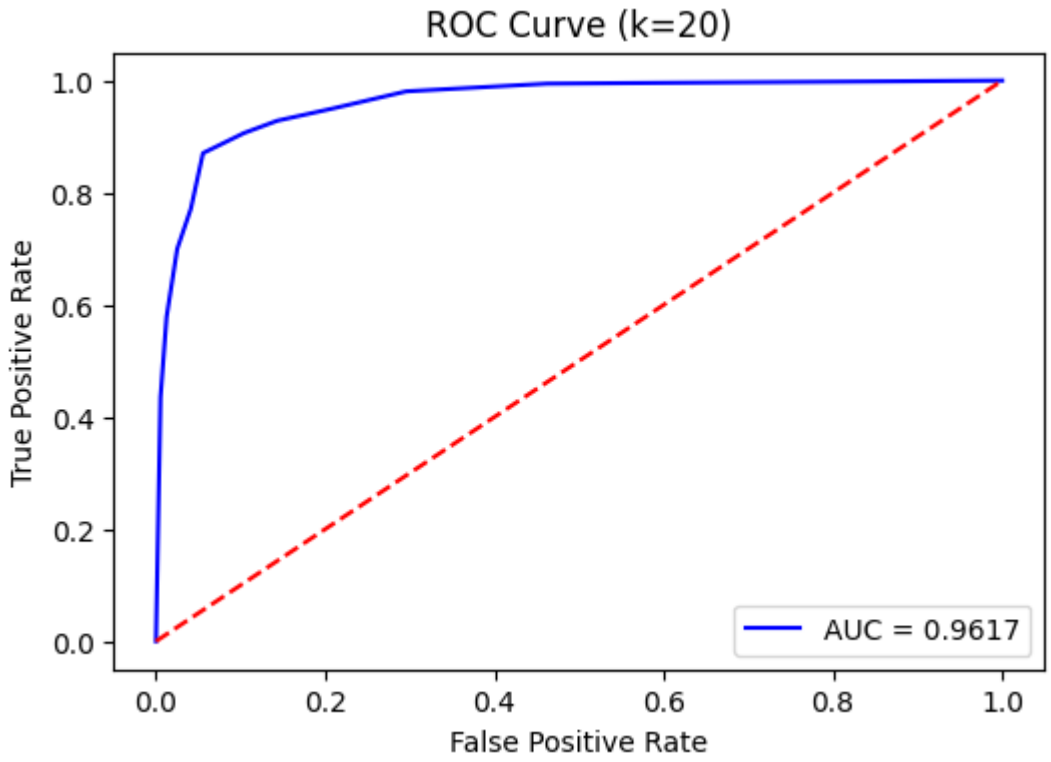
In [ ]:
```
algo='ball_tree'
k=11

knn = KNeighborsClassifier(n_neighbors=k, algorithm=algo)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print('k=11')
print('Accuracy:',acc)
```

In [18]:
```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_

y_pred = knn.predict(X_test)
y_pred_proba = knn.predict_proba(X_test)[:, 1]  # Probability for ROC

# ================================
# 6. Metrics
# ================================
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")

# ================================
# 7. Confusion Matrix
# ================================
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[0, 1], yticklabe
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix (k={k})')
```

```
plt.show()

# ==============================
# 8. ROC Curve
# ==============================
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {roc_auc:.4f}')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve (k={k})')
plt.legend()
plt.show()
```

```
Precision: 0.9107
Recall:    0.8705
F1 Score:  0.8901
```





In [19]:
```
# ================================
# 9. K-Fold Cross Validation (K=5)
# ================================
from sklearn.model_selection import StratifiedKFold, cross_val_score

print("\nK-Fold Cross Validation (k = 11)")

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(KNeighborsClassifier(n_neighbors=11), X, y, cv=kfold

print(f"Mean Accuracy: {cv_scores.mean():.4f}")
print(f"Standard Deviation: {cv_scores.std():.4f}")
```

```
K-Fold Cross Validation (k = 11)
Mean Accuracy: 0.9081
Standard Deviation: 0.0097
```

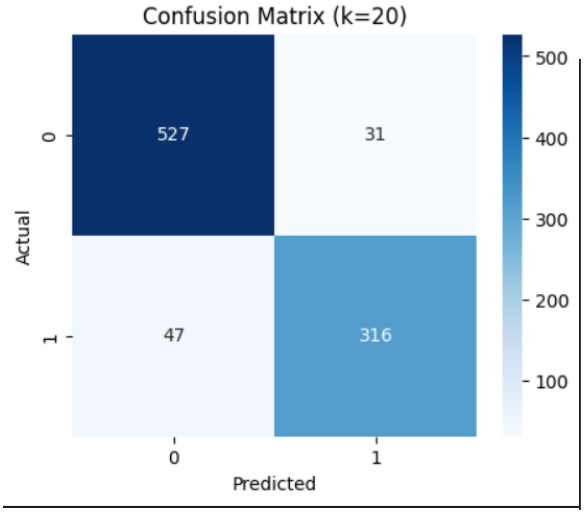
In [20]:
```
# ================================
# 10. Compare and Record Observations
# ================================
import pandas as pd
best_k=11

metrics_data = {
    "Best_k": [best_k],
    "Accuracy": [acc],
    "Precision": [precision],
    "Recall": [recall],
    "F1-Score": [f1],
    "ROC AUC": [roc_auc],
    "CV Mean Accuracy": [cv_scores.mean()],
    "CV Std Dev": [cv_scores.std()]
}

results_df = pd.DataFrame(metrics_data)
```

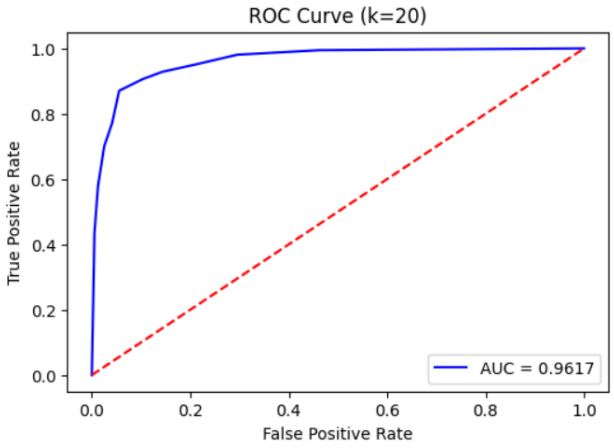
```
print("\nFinal Evaluation Metrics Summary:")
print(results_df)
```

```
Final Evaluation Metrics Summary:
   Best_k  Accuracy  Precision    Recall  F1-Score   ROC AUC  \
0      11  0.915309   0.910663  0.870523  0.890141  0.961714

   CV Mean Accuracy  CV Std Dev
0          0.908064     0.00968
```

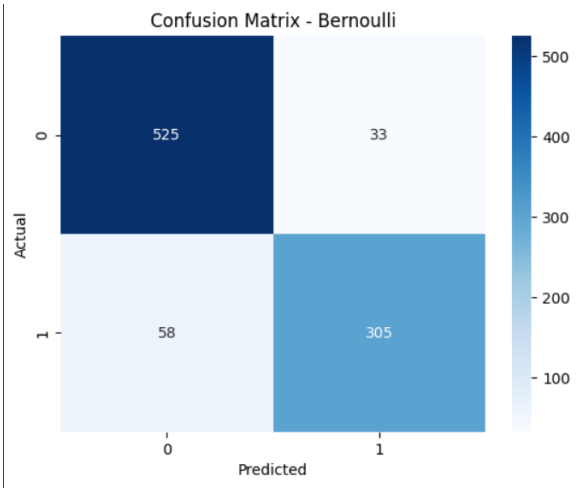
5 Confussion Matrix and ROC curve

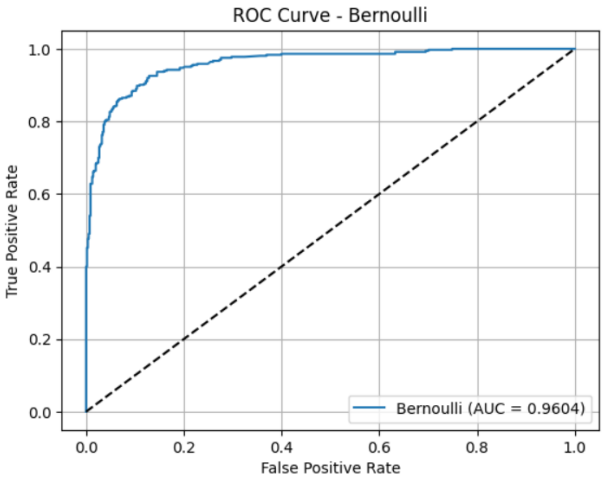
Naïve Bayes,



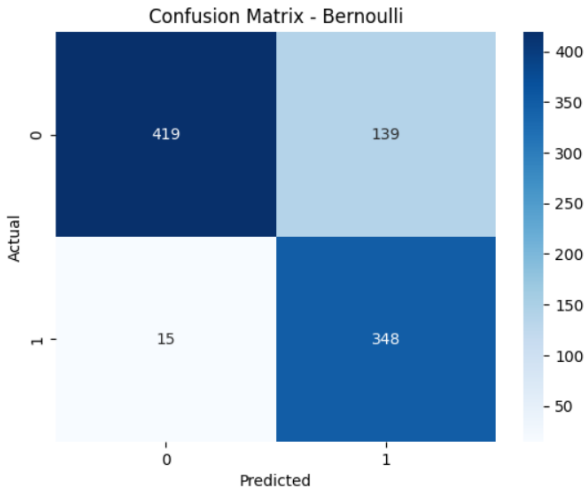


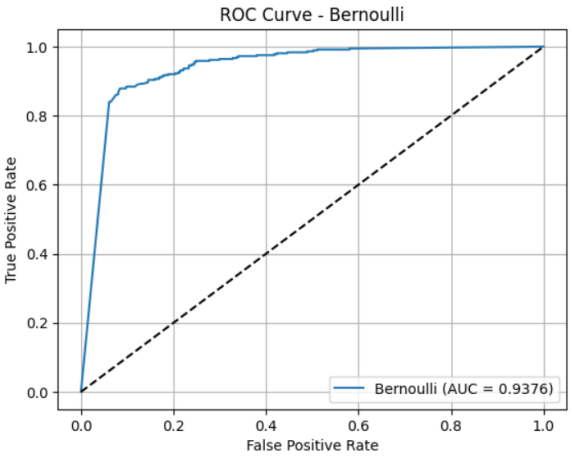
Bernoulli,



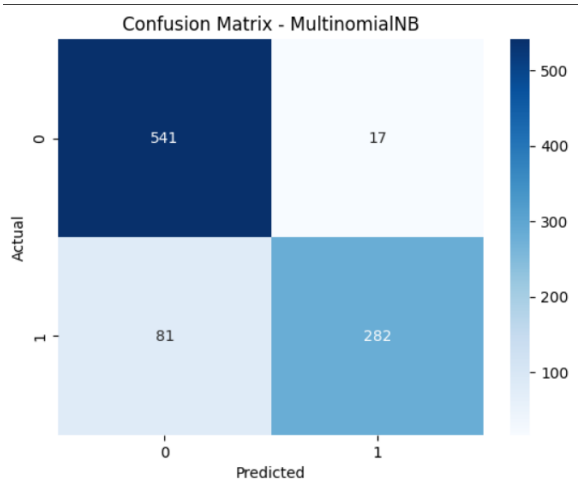


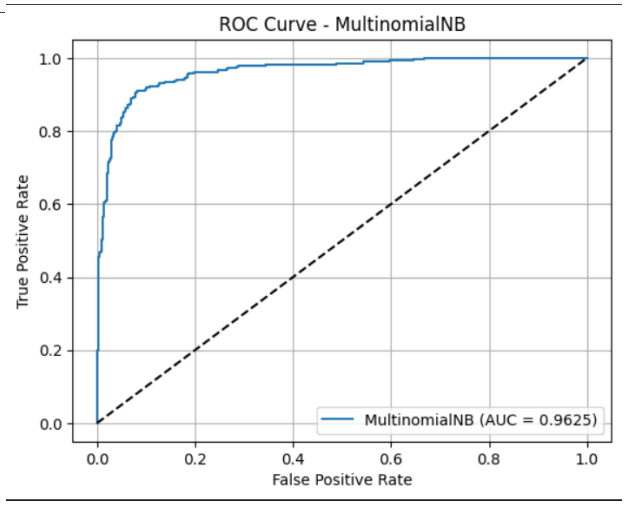
Gaussian,





Multinomial,





6 Comparison Table

Performance Comparison of Naïve Bayes

Model	Accuracy	Precision	Recall	F1 Score
Bernoulli NB	0.9012	0.9024	0.8402	0.8702
Multinomial NB	0.8936	0.9431	0.7769	0.8520
Gaussian NB	0.8328	0.7146	0.9587	0.8188

KNN Performance for different k values

K	Accuracy	Precision	Recall	F1 Score
1	0.8899	0.8551	0.8676	0.8613
3	0.8892	0.8741	0.8652	0.8574
5	0.8993	0.8828	0.8585	0.8705
7	0.8950	0.8815	0.8474	0.8641

KD tree vs Ball tree

Metric	KD Tree	Ball Tree
Accuracy	0.9153	0.9153
Precision	0.9107	0.9107
Recall	0.8705	0.8705
F1 Score	0.8901	0.8901
Training Time	0.4470	0.4058

7 Observation:

• KNN (k=1) consistently achieved higher accuracy across all 5 folds compared to Multinomial

Na¨ıve Bayes, with an average accuracy of 90.39 percentage vs 88.63 percentage.

• Na¨ıve Bayes showed slightly more stable performance across folds, whereas KNN had slightly

higher variance but better peak performance (e.g., Fold 3 with 91.52 percentage).

• KNN’ s superior accuracy suggests that instance-based learning works better for this dataset,

possibly due to well-separated class boundaries that KNN can capture using distance metrics.

8 Conclusion:

• KNN (k=1) outperformed Multinomial Na¨ıve Bayes in terms of average accuracy in 5-fold

cross-validation, making it the better choice for this dataset.

• However, Na¨ıve Bayes remains a strong baseline due to its simplicity, speed, and competitive

accuracy, especially for high-dimensional or text data.