github link : https://github.com/Yashasvee-second/MLlab-ex4-spam-classification

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,precision_score, recall_score, confusion_matrix
```

```
# Step 1: Loading the dataset
data = pd.read_csv("/content/drive/MyDrive/notes&lab-work/sem6/ai lab/ex4/spambase_csv.csv")
data.head()
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_intern
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.

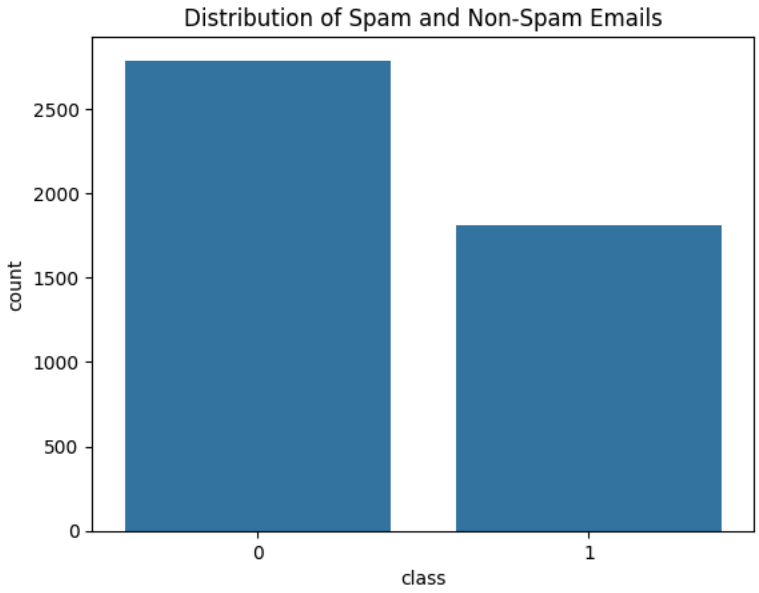
5 rows × 58 columns

```
data.info()
```

```
 2   word_freq_all              4601 non-null   float64
 3   word_freq_3d               4601 non-null   float64
 4   word_freq_our              4601 non-null   float64
 5   word_freq_over             4601 non-null   float64
 6   word_freq_remove           4601 non-null   float64
 7   word_freq_internet         4601 non-null   float64
 8   word_freq_order            4601 non-null   float64
 9   word_freq_mail             4601 non-null   float64
 10  word_freq_receive          4601 non-null   float64
 11  word_freq_will             4601 non-null   float64
 12  word_freq_people           4601 non-null   float64
 13  word_freq_report           4601 non-null   float64
 14  word_freq_addresses        4601 non-null   float64
 15  word_freq_free             4601 non-null   float64
 16  word_freq_business         4601 non-null   float64
 17  word_freq_email            4601 non-null   float64
 18  word_freq_you              4601 non-null   float64
 19  word_freq_credit           4601 non-null   float64
 20  word_freq_your             4601 non-null   float64
 21  word_freq_font             4601 non-null   float64
 22  word_freq_000              4601 non-null   float64
 23  word_freq_money            4601 non-null   float64
 24  word_freq_hp               4601 non-null   float64
 25  word_freq_hpl              4601 non-null   float64
 26  word_freq_george           4601 non-null   float64
 27  word_freq_650              4601 non-null   float64
 28  word_freq_lab              4601 non-null   float64
 29  word_freq_labs             4601 non-null   float64
 30  word_freq_telnet           4601 non-null   float64
 31  word_freq_857              4601 non-null   float64
 32  word_freq_data             4601 non-null   float64
 33  word_freq_415              4601 non-null   float64
 34  word_freq_85               4601 non-null   float64
 35  word_freq_technology       4601 non-null   float64
 36  word_freq_1999             4601 non-null   float64
 37  word_freq_parts            4601 non-null   float64
 38  word_freq_pm               4601 non-null   float64
 39  word_freq_direct           4601 non-null   float64
 40  word_freq_cs               4601 non-null   float64
 41  word_freq_meeting          4601 non-null   float64
 42  word_freq_original         4601 non-null   float64
 43  word_freq_project          4601 non-null   float64
 44  word_freq_re               4601 non-null   float64
```

```
 52  char_freq_%24               4601 non-null   float64
 53  char_freq_%23               4601 non-null   float64
 54  capital_run_length_average  4601 non-null   float64
 55  capital_run_length_longest  4601 non-null   int64
 56  capital_run_length_total    4601 non-null   int64
 57  class                       4601 non-null   int64
dtypes: float64(55), int64(3)
memory usage: 2.0 MB
```

```
# Step 2: Pre-Processing the data
# Handling missing values (replace missing data with data in previous row
# data has no null values
from sklearn.preprocessing import MinMaxScaler

data.fillna(method='ffill', inplace=True)

# Encoding (if categorical variables are present)

# Normalization
# Assuming all features are numeric, normalize them to [0, 1]
# below code applies norm , std to all cols so omitted
#################################################################
# data_normalized = (data - data.min()) / (data.max() - data.min())

# # Standardization
# scaler = StandardScaler()
# data_standardized = scaler.fit_transform(data_normalized)
# data_standardized = pd.DataFrame(data_standardized, columns=data.columns)
#################################################################

columns_to_scale = ['capital_run_length_average', 'capital_run_length_longest', "capital_run_length_total"]

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Perform Min-Max scaling on selected columns
data_scaled = data.copy()  # Make a copy of the original DataFrame
data_scaled[columns_to_scale] = scaler.fit_transform(data[columns_to_scale])


sns.countplot(x='class', data=data_scaled)
plt.title('Distribution of Spam and Non-Spam Emails')
plt.show()
```




```
# Step 4: Feature Engineering Techniques (if applicable)
# No feature engineering applied in this example

# Step 5: Split the data into training, testing, and validation sets
X = data_scaled.drop("class", axis=1)
y = data["class"]

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 GaussianNB
gnb = GaussianNB()
```

```
y_pred = gnb.fit(X_train, y_train).predict(X_test)


# Model Accuracy: how often is the classifier correct?
print("Accuracy:",accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",recall_score(y_test, y_pred))
```

```
Accuracy: 0.8219326818675353
Precision: 0.7233201581027668
Recall: 0.9384615384615385
```

```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

               0       0.94      0.74      0.83       531
               1       0.72      0.94      0.82       390

        accuracy                           0.82       921
       macro avg       0.83      0.84      0.82       921
    weighted avg       0.85      0.82      0.82       921
```

```
# Confusion Matrix
# from sklearn.metrics import confusion_matrix
# cm = confusion_matrix(y_test,y_pred)
# cm_display = metrics.ConfusionMatrixDisplay(cm)
# cm_display.plot()
conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)

# Step 9: Represent the results using graphs
# Visualize the confusion matrix
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[391 140]
 [ 24 366]]
```