**Spam SMS Classification Project**

**Objective:**

The primary goal of this program is to classify SMS messages into **spam** (unwanted promotional or fraudulent messages) or **ham** (normal, non-promotional messages). This is achieved by preprocessing text, extracting relevant features, and training machine learning models.

**Overview of the Program:**

The program leverages Python and popular data science libraries like **pandas, numpy, matplotlib, seaborn, nltk, and sklearn** to accomplish the task. It performs data preprocessing, feature engineering, exploratory data analysis (EDA), and machine learning model training, followed by evaluation and prediction on unseen SMS messages.

**RAW CODE:**

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.naive\_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

# Load the dataset

dataset = pd.read\_csv("SMSSpamCollection", sep='\t', names=['label', 'message'])

# Convert labels to binary format: 'ham' to 0, 'spam' to 1

dataset['label'] = dataset['label'].map({'ham': 0, 'spam': 1})

# Visualize the dataset distribution

plt.figure(figsize=(8, 8))

sns.countplot(x='label', data=dataset)

plt.title('Countplot for Spam vs Ham as Imbalanced Dataset')

plt.xlabel('Is the SMS Spam?')

plt.ylabel('Count')

plt.show()

# Print details about the dataset

print(f'Total Number of SMS: {len(dataset)}')

print(f'Number of Spam SMS: {dataset["label"].sum()}')

print(f'Number of Ham SMS: {len(dataset) - dataset["label"].sum()}')

# Feature Engineering

dataset['word\_count'] = dataset['message'].apply(lambda x: len(x.split()))

dataset['contains\_currency\_symbols'] = dataset['message'].apply(lambda x: any(symbol in x for symbol in ['$', '£', '¥', '₹', '€']))

dataset['contains\_number'] = dataset['message'].apply(lambda x: any(char.isdigit() for char in x))

# Text Cleaning and Preprocessing

nltk.download('stopwords')

nltk.download("wordnet")

wnl = WordNetLemmatizer()

def preprocess\_text(sms):

message = re.sub('[^a-zA-Z]', ' ', sms) # Remove special characters and numbers

message = message.lower()

words = message.split()

filtered\_words = [wnl.lemmatize(word) for word in words if word not in stopwords.words('english')]

return ' '.join(filtered\_words)

dataset['cleaned\_message'] = dataset['message'].apply(preprocess\_text)

# TF-IDF Vectorization

tfidf = TfidfVectorizer(max\_features=500)

x = tfidf.fit\_transform(dataset['cleaned\_message']).toarray()

y = dataset['label']

# Train-test split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

# Naive Bayes Model

mnb = MultinomialNB()

mnb.fit(x\_train, y\_train)

y\_pred = mnb.predict(x\_test)

# Evaluation of Naive Bayes

print("Naive Bayes Model Performance:")

print(classification\_report(y\_test, y\_pred))

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(8, 8))

sns.heatmap(cm, annot=True, fmt='d', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'], cmap="Blues")

plt.title("Confusion Matrix for Naive Bayes")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

# Decision Tree Model

dt = DecisionTreeClassifier()

dt.fit(x\_train, y\_train)

y\_pred\_dt = dt.predict(x\_test)

# Evaluation of Decision Tree

print("Decision Tree Model Performance:")

print(classification\_report(y\_test, y\_pred\_dt))

cm\_dt = confusion\_matrix(y\_test, y\_pred\_dt)

plt.figure(figsize=(8, 8))

sns.heatmap(cm\_dt, annot=True, fmt='d', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'], cmap="Greens")

plt.title("Confusion Matrix for Decision Tree")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

# Prediction Function

def predict\_spam(sms):

processed\_sms = preprocess\_text(sms)

vectorized\_sms = tfidf.transform([processed\_sms]).toarray()

return mnb.predict(vectorized\_sms)[0]

# Predictions

sample\_messages = [

"IMPORTANT - You could be entitled up to $3,160 in compensation from mis-sold PPI on a credit card or loan.",

"Hey, are we still on for dinner tonight?",

"WINNER! You have won a lottery of $1000. Claim your prize now!",

"Meeting rescheduled to tomorrow at 10 AM. Please confirm attendance.",

"Get a discount on your next purchase! Visit our website for more details."

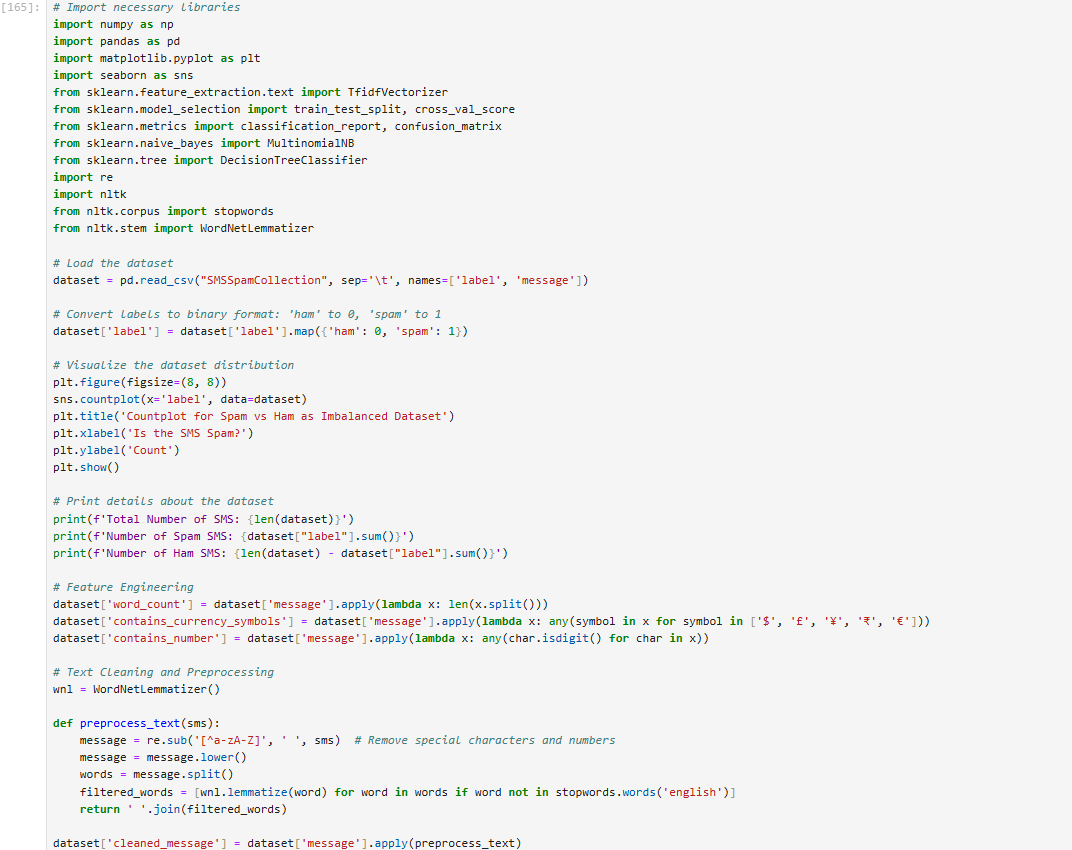
]

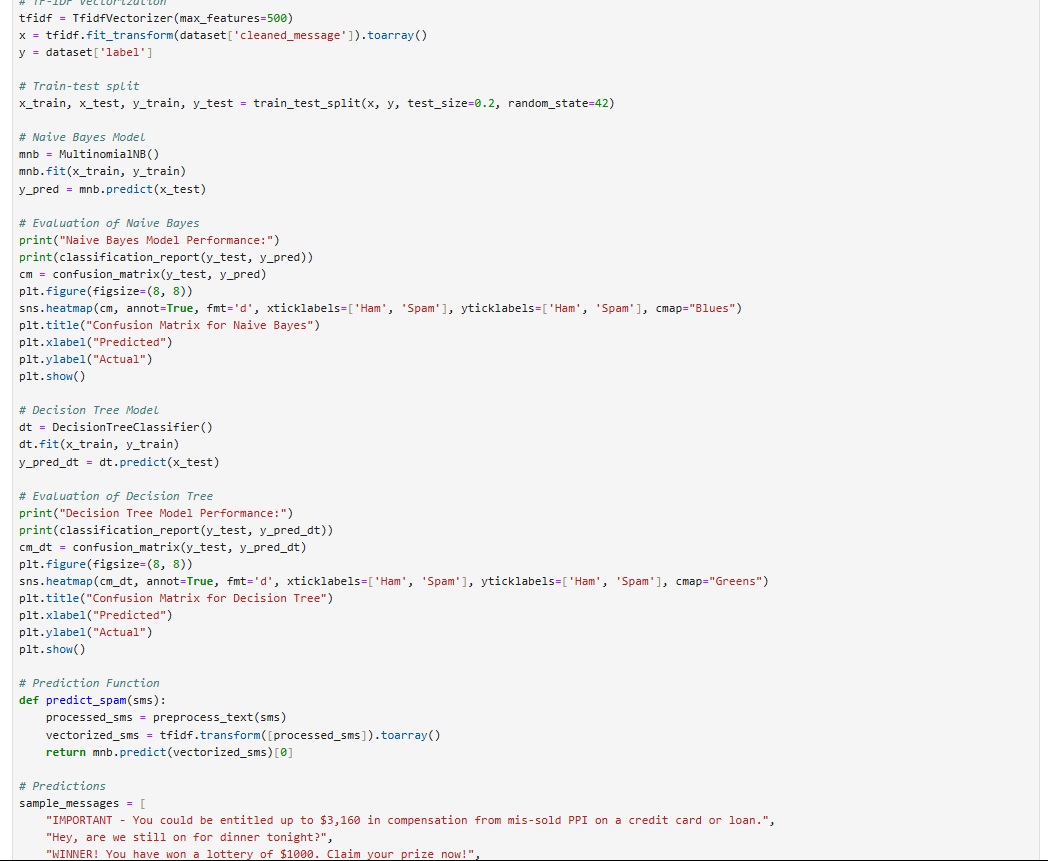
for msg in sample\_messages:

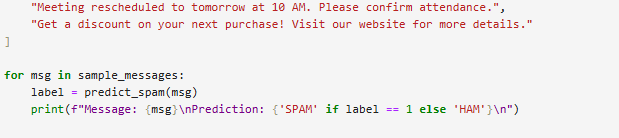
label = predict\_spam(msg)

print ( f"Message: {msg}\nPrediction: {'SPAM' if label == 1 else 'HAM'}\n")

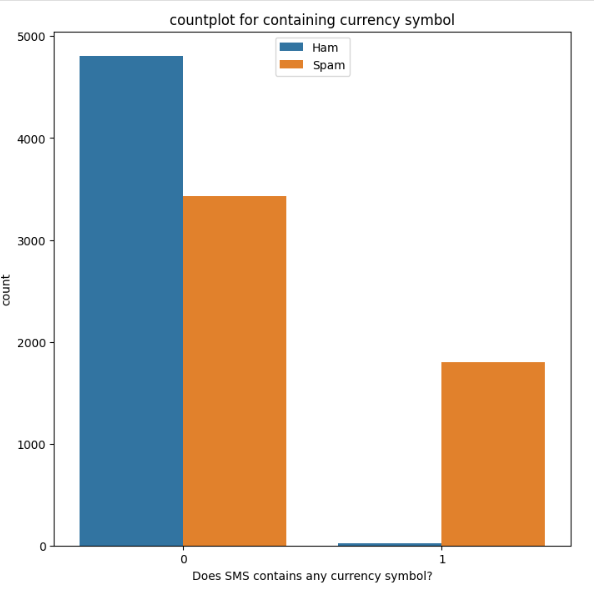
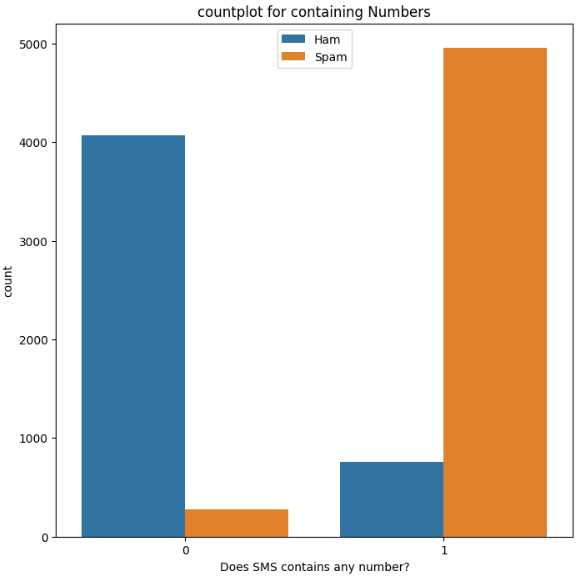
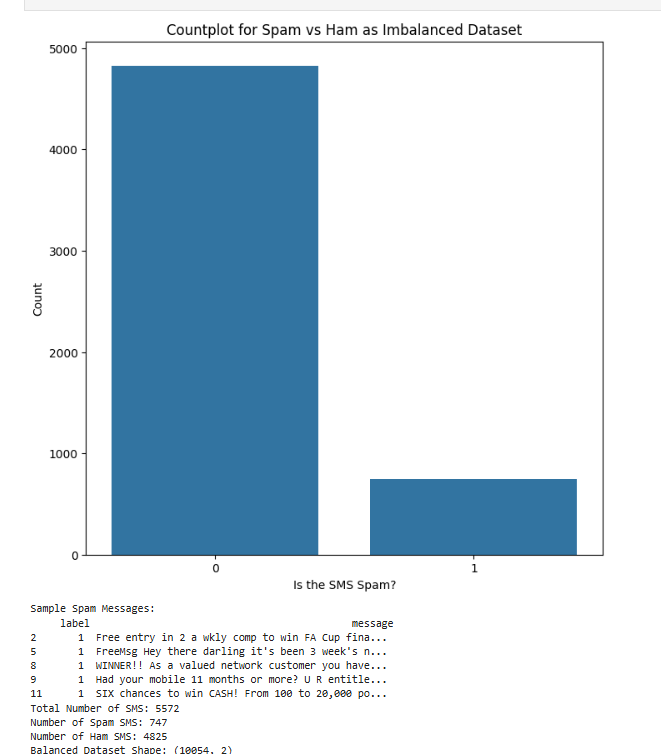
**CODE:**

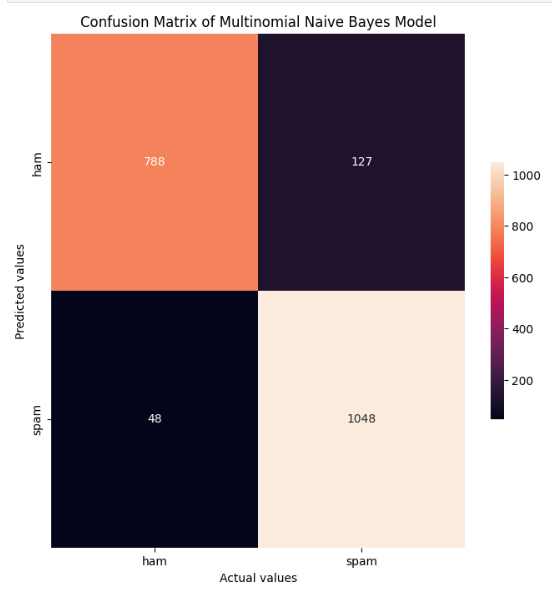
****

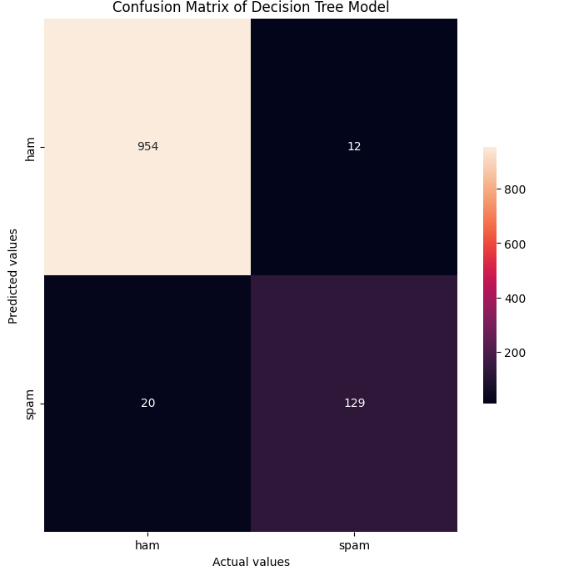
****

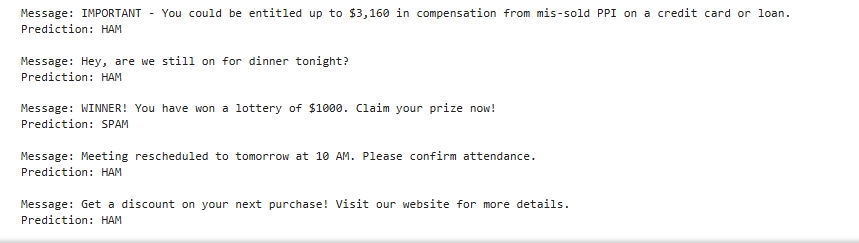
****

**OUTPUTS:**

****





****

**Visualizations:**

**Imbalanced vs Balanced Dataset**:

* + Count plots for spam and ham before and after balancing.

**Word Count Distribution**:

* + Histograms showing the distribution of word counts for spam and ham.

**Confusion Matrix**:

* + Heatmap visualizing true positives, true negatives, false positives, and false negatives for each model.

**Conclusion:**

The program successfully classifies SMS messages as spam or ham by combining text preprocessing, feature engineering, and machine learning. The Naive Bayes model demonstrated robust performance with interpretable predictions, making it suitable for SMS spam detection.