

```

# =====
# 1. Import Libraries
# =====

import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt

from wordcloud import WordCloud

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# =====
# 2. Load Dataset
# =====

df = pd.read_csv(
    "/kaggle/input/datasets/mustafaadelibrahim/sms-spam-collection/SMSSpamCollection",
    sep="\t",
    header=None,
    names=["Label", "Text"],
    encoding="utf-8"
)

print("Dataset Shape:", df.shape)

print("\nFirst 5 rows:")
print(df.head())

# =====
# 3. Data Information
# =====

print("\nClass Distribution:")
print(df["Label"].value_counts())

# =====
# 4. Class Distribution Graph
# =====

plt.figure(figsize=(6,4))

df["Label"].value_counts().plot(
    kind="bar"
)

plt.title("Spam vs Ham Messages")
plt.xlabel("Message Type")
plt.ylabel("Number of Messages")
plt.xticks(rotation=0)
plt.show()

# =====
# 5. Data Preprocessing
# =====

df = df.dropna()

```

```

def clean_text(text):

    # Lowercase
    text = text.lower()

    # Remove punctuation and numbers
    text = re.sub(r"[^a-z\s]", "", text)

    # Remove extra spaces
    text = re.sub(r"\s+", " ", text)

    return text.strip()

df["Clean_Text"] = df["Text"].apply(clean_text)

print("\nOriginal Text:")
print(df["Text"].iloc[0])

print("\nCleaned Text:")
print(df["Clean_Text"].iloc[0])

# =====
# 6. Message Length
# =====

df["Message_Length"] = df["Text"].apply(len)

plt.figure(figsize=(8,5))

plt.hist(
    df[df["Label"] == "ham"]["Message_Length"],
    bins=30,
    alpha=0.6,
    label="Ham"
)

plt.hist(
    df[df["Label"] == "spam"]["Message_Length"],
    bins=30,
    alpha=0.6,
    label="Spam"
)

plt.title("Message Length Distribution")
plt.xlabel("Message Length")
plt.ylabel("Frequency")
plt.legend()

plt.show()

# =====
# 7. Word Cloud - Spam
# =====

spam_text = " ".join(
    df[df["Label"] == "spam"]["Clean_Text"]
)

spam_wordcloud = WordCloud(
    width=800,
    height=400,
    background_color="white"
).generate(spam_text)

plt.figure(figsize=(10,5))

plt.imshow(
    spam_wordcloud,
    interpolation="bilinear"
)

plt.axis("off")
plt.title("Spam Messages Word Cloud")

```

```

plt.show()

# =====
# 8. Word Cloud - Ham
# =====

ham_text = " ".join(
    df[df["Label"] == "ham"]["Clean_Text"]
)

ham_wordcloud = WordCloud(
    width=800,
    height=400,
    background_color="white"
).generate(ham_text)

plt.figure(figsize=(10,5))

plt.imshow(
    ham_wordcloud,
    interpolation="bilinear"
)

plt.axis("off")
plt.title("Ham Messages Word Cloud")

plt.show()

# =====
# 9. Convert Labels
# =====

df["Label"] = df["Label"].map({
    "ham": 0,
    "spam": 1
})

# =====
# 10. Split Dataset
# =====

X = df["Clean_Text"]
y = df["Label"]

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("\nTraining samples:", len(X_train))
print("Testing samples:", len(X_test))

# =====
# 11. Feature Representation
# =====

# ----- Bag of Words -----

bow = CountVectorizer(
    stop_words="english"
)

X_train_bow = bow.fit_transform(X_train)
X_test_bow = bow.transform(X_test)

# ----- TF-IDF -----

tfidf = TfidfVectorizer(
    stop_words="english"

```

```
)

X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print("\nBag of Words Shape:")
print(X_train_bow.shape)

print("\nTF-IDF Shape:")
print(X_train_tfidf.shape)

# =====
# 12. Model 1 - Naive Bayes + Bag of Words
# =====

model_bow = MultinomialNB()

model_bow.fit(
    X_train_bow,
    y_train
)

pred_bow = model_bow.predict(
    X_test_bow
)

# =====
# 13. Model 2 - Naive Bayes + TF-IDF
# =====

model_tfidf = MultinomialNB()

model_tfidf.fit(
    X_train_tfidf,
    y_train
)

pred_tfidf = model_tfidf.predict(
    X_test_tfidf
)

# =====
# 14. Calculate Metrics
# =====

results = pd.DataFrame({

    "Method": [
        "Bag of Words",
        "TF-IDF"
    ],

    "Accuracy": [
        accuracy_score(y_test, pred_bow),
        accuracy_score(y_test, pred_tfidf)
    ],

    "Precision": [
        precision_score(y_test, pred_bow),
        precision_score(y_test, pred_tfidf)
    ],

    "Recall": [
        recall_score(y_test, pred_bow),
        recall_score(y_test, pred_tfidf)
    ],

    "F1-Score": [
        f1_score(y_test, pred_bow),
        f1_score(y_test, pred_tfidf)
    ]
})
```

```

print("\n==== Model Results =====")
print(results)

# =====
# 15. Accuracy Comparison Graph
# =====

plt.figure(figsize=(8,5))

plt.bar(
    results["Method"],
    results["Accuracy"]
)

plt.title("Accuracy Comparison")
plt.xlabel("Feature Representation")
plt.ylabel("Accuracy")

plt.ylim(0, 1)

plt.show()

# =====
# 16. Compare All Metrics
# =====

results_plot = results.set_index("Method")

results_plot[
    ["Accuracy", "Precision", "Recall", "F1-Score"]
].plot(
    kind="bar",
    figsize=(10,6)
)

plt.title("Model Performance Comparison")
plt.xlabel("Feature Representation")
plt.ylabel("Score")

plt.ylim(0, 1)

plt.xticks(rotation=0)

plt.legend()

plt.show()

# =====
# 17. Confusion Matrix - TF-IDF
# =====

cm = confusion_matrix(
    y_test,
    pred_tfidf
)

plt.figure(figsize=(6,5))

plt.imshow(cm)

plt.title("Confusion Matrix - TF-IDF")

plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")

plt.xticks(
    [0,1],
    ["Ham", "Spam"]
)

plt.yticks(
    [0,1],
    ["Ham", "Spam"]
)

```

```

# Add numbers
for i in range(2):
    for j in range(2):

        plt.text(
            j,
            i,
            cm[i,j],
            ha="center",
            va="center"
        )

plt.colorbar()

plt.show()

# =====
# 18. Classification Report
# =====

print("\n==== Classification Report =====")

print(
    classification_report(
        y_test,
        pred_tfidf,
        target_names=[
            "Ham",
            "Spam"
        ]
    )
)

# =====
# 19. Test New Messages
# =====

new_messages = [

    "Congratulations! You won a free prize. Call now!",

    "Hey, are we meeting tomorrow at 5 pm?",

    "You have won a free lottery ticket. Claim now!",

    "Can you send me the lecture notes?"

]

new_messages_clean = [
    clean_text(message)
    for message in new_messages
]

new_features = tfidf.transform(
    new_messages_clean
)

predictions = model_tfidf.predict(
    new_features
)

print("\n==== New Message Predictions =====")

for message, prediction in zip(
    new_messages,
    predictions
):

    if prediction == 1:

```

```

        result = "SPAM"
    else:
        result = "HAM"

    print("\nMessage:", message)
    print("Prediction:", result)

```

Dataset Shape: (5572, 2)

First 5 rows:

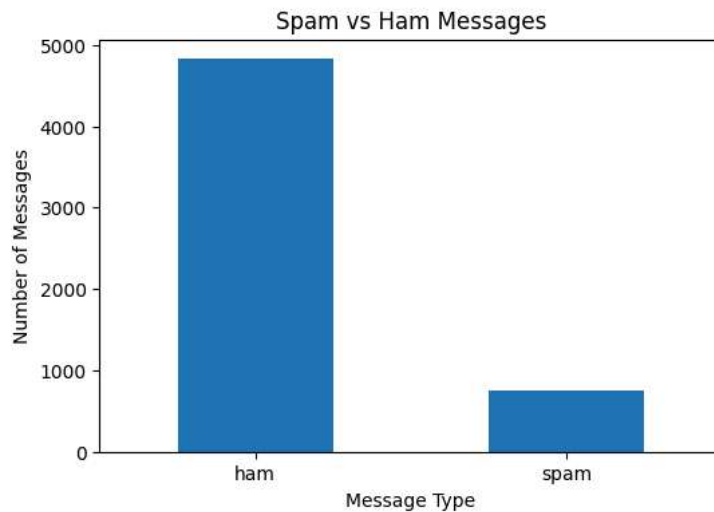
|   | Label | Text                                              |
|---|-------|---------------------------------------------------|
| 0 | ham   | Go until jurong point, crazy.. Available only ... |
| 1 | ham   | Ok lar... Joking wif u oni...                     |
| 2 | spam  | Free entry in 2 a wkly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

Class Distribution:

```

Label
ham    4825
spam    747
Name: count, dtype: int64

```



Original Text:

Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...

Cleaned Text:

go until jurong point crazy available only in bugis n great world la e buffet cine there got amore wat

