```
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score



# Download required nltk data
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
import pandas as pd

# Load the dataset
df = pd.read_csv("/content/spam.csv", encoding="latin1")

# Display first few rows
print(df.head())
```

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

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

```
# Keep only necessary columns (adjust based on dataset structure)
df = df.iloc[:, [0, 1]]  # Keeping only first two columns
df.columns = ['label', 'message']  # Rename columns for clarity

# Convert labels to binary (spam → 1, ham → 0)
df['label'] = df['label'].map({'spam': 1, 'ham': 0})

print(df.head())
```

```
   label                                            message
0    NaN  Go until jurong point, crazy.. Available only ...
1    NaN                      Ok lar... Joking wif u oni...
3    NaN  U dun say so early hor... U c already then say...
4    NaN  Nah I don't think he goes to usf, he lives aro...
6    NaN  Even my brother is not like to speak with me. ...
<ipython-input-38-0b0b86c6e2b6>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df['label'] = df['label'].map({'spam': 1, 'ham': 0})
```

```
# Check for NaN values
df = df.dropna(subset=['label'])


# Ensure dataset has both spam & ham
if df['label'].nunique() < 2:
```

```
    raise ValueError("Dataset does not contain both spam and ham messages. Check preprocessing!")


# Class distribution before resampling
print("\nClass distribution before resampling:\n", df['label'].value_counts())
```

```
Class distribution before resampling:
 label
0    4825
1     747
Name: count, dtype: int64
```

```
# Separate spam and ham messages
ham = df[df['label'] == 0]
spam = df[df['label'] == 1]


# Resampling: Balance the dataset
if len(spam) < len(ham):
    spam_upsampled = resample(spam, replace=True, n_samples=len(ham), random_state=42)
    df = pd.concat([ham, spam_upsampled])



# Class distribution after resampling
print("\nClass Distribution After Resampling:")
print(df['label'].value_counts())
```

```
Class Distribution After Resampling:
label
0    4825
1    4825
Name: count, dtype: int64
```

```
# Text Preprocessing
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))



# Initialize stemmer and stop words
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))


def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    words = text.split()  # Tokenization
    words = [stemmer.stem(word) for word in words if word not in stop_words]  # Stemming & stopword removal
    return ' '.join(words)


# Example usage:
text = "This is a SAMPLE message to preprocess!?"
print(preprocess_text(text))
```

```
sampl messag preprocess
```

```
# Apply preprocessing to all messages
df['cleaned_message'] = df['message'].apply(preprocess_text)



# Feature Extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['cleaned_message']).toarray()
y = df['label'].values


# 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)
```

```
# Models
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100,max_depth=10, random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100, random_state=42),
    "MultinomialNB": MultinomialNB()
}


# Train & evaluate models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred)
    }
    print(f"{name} - Accuracy: {results[name]['Accuracy']:.4f}, Precision: {results[name]['Precision']:.4f}")
```

```
Random Forest - Accuracy: 0.9409, Precision: 0.9965
Gradient Boosting - Accuracy: 0.9528, Precision: 0.9659
MultinomialNB - Accuracy: 0.9839, Precision: 0.9844
```

```
# Model names and their accuracy
model_names = list(results.keys()) + ["Combined Model"]
accuracies = [results[name]['Accuracy'] for name in results] + ['accuracy_combined']


# Ensemble (Combining Random Forest & MultinomialNB)
rf_model = models["Random Forest"]
nb_model = models["MultinomialNB"]



rf_preds = rf_model.predict_proba(X_test)[:, 1]  # Probabilities from Random Forest
nb_preds = nb_model.predict_proba(X_test)[:, 1]  # Probabilities from MultinomialNB



combined_preds = (rf_preds + nb_preds) / 2  # Average probabilities
final_preds = (combined_preds > 0.5).astype(int)  # Convert to binary classification


# Evaluation for Combined Model
accuracy_combined = accuracy_score(y_test, final_preds)
precision_combined = precision_score(y_test, final_preds)

print(f"\nCombined Model - Accuracy: {accuracy_combined:.4f}, Precision: {precision_combined:.4f}")
```

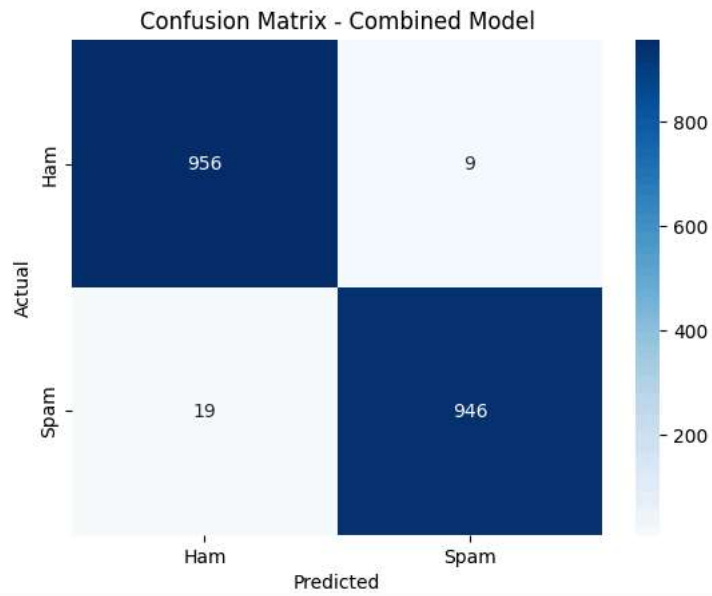
```
Combined Model - Accuracy: 0.9855, Precision: 0.9906
```

```
# Function to Predict User Input SMS
def predict_sms(message):
    processed_msg = preprocess_text(message)
    vectorized_msg = vectorizer.transform([processed_msg]).toarray()
    # Predict using both models
    rf_prob = rf_model.predict_proba(vectorized_msg)[:, 1]
    nb_prob = nb_model.predict_proba(vectorized_msg)[:, 1]

    combined_prob = (rf_prob + nb_prob) / 2
    prediction = "Spam" if combined_prob > 0.5 else "Ham"

    return prediction


# Confusion Matrix for the Combined Model
y_pred_combined = final_preds
cm = confusion_matrix(y_test, y_pred_combined)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Combined Model")
plt.show()
```




```
# Take user input
user_message = input("\nEnter an SMS message to classify: ")
print(f"\nOutput: {predict_sms(user_message)}")
```

```
Enter an SMS message to classify: FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it s..

Output: Spam
```