In [1]:
```
import pandas as pd
import string
import nltk
import joblib
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split


# Load the dataset
file_path = "updated_sms.csv"
df = pd.read_csv(file_path, encoding="utf-8")  # Using UTF-8 encodi

# Display basic info
df.info()

# Show first few rows
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6064 entries, 0 to 6063
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   label    6064 non-null   object
 1   message  6059 non-null   object
dtypes: object(2)
memory usage: 94.9+ KB
```

Out[1]:

	label	message
0	0	go until jurong point crazy available only in ...
1	0	ok lar joking wif u oni
2	1	free entry in a wkly comp to win fa cup final...
3	0	u dun say so early hor u c already then say
4	0	nah i dont think he goes to usf he lives aroun...

In [2]:
```
# Download stopwords if not available
nltk.download("stopwords")

# Initialize stemmer
ps = PorterStemmer()

# Define preprocessing function
def preprocess_text(text):
    if not isinstance(text, str):  # Ensure text is valid
        return ""

    text = text.lower()  # Convert to lowercase
    text = ''.join([char for char in text if char not in string.pun
    words = text.split()
    words = [word for word in words if word not in stopwords.words(
```

```
    words = [ps.stem(word) for word in words if len(word) > 2]  # A

    return ' '.join(words) if words else "empty"  # Ensure non-empt

# Apply preprocessing
df["message"] = df["message"].apply(preprocess_text)

# Show results
df.head()
```

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

Out[2]:

	label	message
0	0	jurong point crazi avail bugi great world buff...
1	0	lar joke wif oni
2	1	free entri wkli comp win cup final tkt may tex...
3	0	dun say earli hor alreadi say
4	0	nah dont think goe usf live around though

In [3]:

```
# Define a custom stopwords list (common English stopwords)
custom_stopwords = set([
    "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "y
    "yourselves", "he", "him", "his", "himself", "she", "her", "her
    "they", "them", "their", "theirs", "themselves", "what", "which
    "these", "those", "am", "is", "are", "was", "were", "be", "been
    "having", "do", "does", "did", "doing", "a", "an", "the", "and"
    "as", "until", "while", "of", "at", "by", "for", "with", "about
    "through", "during", "before", "after", "above", "below", "to",
    "on", "off", "over", "under", "again", "further", "then", "once
    "why", "how", "all", "any", "both", "each", "few", "more", "mos
    "nor", "not", "only", "own", "same", "so", "than", "too", "very
    "don", "should", "now"
])

# Update preprocessing function to use custom stopwords
def preprocess_text(text):
    if not isinstance(text, str):
        return ""

    text = text.lower()  # Convert to lowercase
    text = ''.join([char for char in text if char not in string.pun
    words = text.split()
    words = [word for word in words if word not in custom_stopwords
    words = [ps.stem(word) for word in words if len(word) > 2]  # A

    return ' '.join(words) if words else "empty"

# Apply preprocessing again
df["message"] = df["message"].apply(preprocess_text)
```

```
# Show processed text
df.head()
```

Out[3]:

	label	message
0	0	jurong point crazi avail bugi great world buff...
1	0	lar joke wif oni
2	1	free entri wkli comp win cup final tkt may tex...
3	0	dun say earli hor alreadi say
4	0	nah dont think goe usf live around though

In [4]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer()

# Transform messages into numerical format
X = vectorizer.fit_transform(df["message"])

# Extract labels
y = df["label"]

# Check shape of transformed data
X.shape, y.shape
```

Out[4]:

```
((6064, 6961), (6064,))
```

In [5]:

```
# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Check distribution of labels in train & test sets
y_train.value_counts(normalize=True), y_test.value_counts(normalize
```

Out[5]:

```
(label
 0      0.795712
 1      0.203875
 ham    0.000412
 Name: proportion, dtype: float64,
 label
 0    0.795548
 1    0.204452
 Name: proportion, dtype: float64)
```

In [6]:

```
# Ensure y_test and y_pred are integers
y_test = y_test.astype(int)
y_pred = y_pred.astype(int)

# Recalculate performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
f1 = f1_score(y_test, y_pred, pos_label=1)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
accuracy, precision, recall, f1, conf_matrix
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 3
      1 # Ensure y_test and y_pred are integers
      2 y_test = y_test.astype(int)
----> 3 y_pred = y_pred.astype(int)
      5 # Recalculate performance metrics
      6 accuracy = accuracy_score(y_test, y_pred)

NameError: name 'y_pred' is not defined
```

In [7]:
```
# Ensure labels are integers
y_test = y_test.astype(int)

# Loop through each alpha value and retrain
nb_results = {}
for alpha in alpha_values:
    nb_model = MultinomialNB(alpha=alpha)
    nb_model.fit(X_train, y_train)

    # Predict and convert to integers
    y_pred = nb_model.predict(X_test).astype(int)

    # Compute evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)

    # Store results
    nb_results[alpha] = (accuracy, precision, recall, f1)

# Convert results to DataFrame
nb_results_df = pd.DataFrame(nb_results, index=["Accuracy", "Precis
nb_results_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 6
      4 # Loop through each alpha value and retrain
      5 nb_results = {}
----> 6 for alpha in alpha_values:
      7     nb_model = MultinomialNB(alpha=alpha)
      8     nb_model.fit(X_train, y_train)

NameError: name 'alpha_values' is not defined
```

In [ ]:
```
# Reinitialize and fit LabelEncoder on the entire dataset (y)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)  # Fit on full labels

# Split the dataset again to keep consistency
y_train_encoded, y_test_encoded = train_test_split(y_encoded, test_

# Train Random Forest again with the corrected labels
rf_model.fit(X_train, y_train_encoded)
y_pred_rf = rf_model.predict(X_test)

# Recalculate performance metrics using "weighted" average to handl
accuracy_rf = accuracy_score(y_test_encoded, y_pred_rf)
precision_rf = precision_score(y_test_encoded, y_pred_rf, average="
recall_rf = recall_score(y_test_encoded, y_pred_rf, average="weight
f1_rf = f1_score(y_test_encoded, y_pred_rf, average="weighted")

# Display results
accuracy_rf, precision_rf, recall_rf, f1_rf
```

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

# Define a stratified K-Fold to balance class distribution
cv = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)  #

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

# Perform Grid Search with balanced CV
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv
grid_search.fit(X_train, y_train_encoded)

# Print best parameters
print("Best Hyperparameters:", grid_search.best_params_)
```

In [ ]:
```
# Train optimized Random Forest model
optimized_rf = RandomForestClassifier(
    max_depth=None,
    min_samples_split=5,
    n_estimators=200,
    random_state=42,
    class_weight="balanced"
)

optimized_rf.fit(X_train, y_train_encoded)

# Make predictions
y_pred_optimized = optimized_rf.predict(X_test)

# Evaluate performance
accuracy_opt = accuracy_score(y_test_encoded, y_pred_optimized)
```

```
precision_opt = precision_score(y_test_encoded, y_pred_optimized, a
recall_opt = recall_score(y_test_encoded, y_pred_optimized, average
f1_opt = f1_score(y_test_encoded, y_pred_optimized, average="weight

# Display final results
print("Final Optimized Random Forest Performance:")
print("✅ Accuracy:", accuracy_opt)
print("✅ Precision:", precision_opt)
print("✅ Recall:", recall_opt)
print("✅ F1 Score:", f1_opt)
```

In [ ]:
```
joblib.dump(optimized_rf, "optimized_sms_spam_model.pkl")
print("Model saved as optimized_sms_spam_model.pkl")
```

In [ ]:
```
# Find misclassified indices
misclassified_idx = (y_test_encoded != y_pred_optimized)

# Extract test set indices
test_indices = y_test.index  # Get the correct test set indices

# Extract misclassified messages
misclassified_messages = df.loc[test_indices[misclassified_idx], "m

# Display misclassified messages
print("Misclassified Messages:\n", misclassified_messages)
```

In [ ]:
```
import joblib

# Save the trained model
joblib.dump(optimized_rf, "optimized_spam_classifier.pkl")

# Save the TF-IDF vectorizer too
joblib.dump(vectorizer, "optimized tfidf_vectorizer.pkl")

print("✅ Model and vectorizer saved successfully!")
```

In [ ]:
```

```

In [ ]:
```

```