

```
import pandas as pd

# Load dataset
df = pd.read_csv("/content/spam_encoded.csv") # Update path if needed

# Display basic info
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Columns: 770 entries, spam to 767
dtypes: float64(768), int64(1), object(1)
memory usage: 32.7+ MB
None
spam                                original_message      0 \
0      0  Go until jurong point, crazy.. Available only ... -0.142966
1      0                        Ok lar... Joking wif u oni... -0.244984
2      1  Free entry in 2 a wkly comp to win FA Cup fina... -0.389005
3      0  U dun say so early hor... U c already then say... -0.090251
4      0  Nah I don't think he goes to usf, he lives aro...  0.056018

      1      2      3      4      5      6      7 ... \
0 -0.225385  0.268059  0.065721 -0.100783 -0.198441  0.390656  0.337182 ...
1 -0.052506  0.137510 -0.623836 -0.311277  0.074931  0.652979  0.200019 ...
2 -0.030240  0.203487 -0.166000  0.020918 -0.139901  0.331638  0.095457 ...
3  0.151952  0.069052 -0.610341 -0.220990  0.013123  0.301812  0.334780 ...
4  0.251125 -0.092620 -0.307521 -0.365733 -0.075228  0.414467  0.388034 ...

      758      759      760      761      762      763      764 \
0  0.069544 -0.396520  0.011688 -0.208539  0.012130 -0.056947 -0.025428
1  0.070855 -0.541656 -0.040561  0.047741 -0.063776 -0.138928  0.041252
2  0.381351 -0.158719 -0.000505 -0.167245  0.169160 -0.084802 -0.366647
3  0.196930 -0.408126  0.062470  0.140425  0.175495 -0.085699 -0.226141
4 -0.168950 -0.419398 -0.070503 -0.010642  0.323683  0.016842 -0.103515

      765      766      767
0 -0.057884  0.165974  0.302198
1 -0.045966  0.087562  0.546937
2 -0.106075  0.096242  0.567941
3 -0.002507  0.255287  0.496604
4 -0.076240  0.479977  0.274191

[5 rows x 770 columns]
```

```
!pip install wordcloud nltk scikit-learn seaborn
```

Show hidden output

```
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

nltk.download('stopwords')
```

ValueError ...

Please explain the error:

NameError: na

## Hypothes

The error message

NameError: name 'tfidf'

is not defined indicates that the variable tfidf is being used in your code (specifically, in the

joblib.dump line) before it has been assigned a value.

Looking at the code and the global variables, we see that you've correctly used a

TfidfVectorizer but have named it 'vectorizer' and not 'tfidf'. You likely intended to save this vectorizer object for later use, but accidentally referred to it as 'tfidf'.

## Suggeste Changes

import joblib

# Save the joblib.dump

# Save the joblib.dump

```

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Check if the input is a string before applying string methods
    if isinstance(text, str):
        text = text.lower() # Convert to lowercase
        text = re.sub(r'\W+', ' ', text) # Remove special characters
        text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words])
    # If not a string (e.g., an integer), potentially handle differently or skip
    # Here, we just return the original value if it's not a string
    return text

# Apply preprocessing, ensuring the 'spam' column is of string type
df['cleaned_text'] = df['spam'].astype(str).apply(preprocess_text)
print(df.head())

```

```

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

```

spam	original_message	0	\
0	0	Go until jurong point, crazy.. Available only ...	-0.142966
1	0	Ok lar... Joking wif u oni...	-0.244984
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	-0.389005
3	0	U dun say so early hor... U c already then say...	-0.090251
4	0	Nah I don't think he goes to usf, he lives aro...	0.056018

1	2	3	4	5	6	7	...	\
0	-0.225385	0.268059	0.065721	-0.100783	-0.198441	0.390656	0.337182	...
1	-0.052506	0.137510	-0.623836	-0.311277	0.074931	0.652979	0.200019	...
2	-0.030240	0.203487	-0.166000	0.020918	-0.139901	0.331638	0.095457	...
3	0.151952	0.069052	-0.610341	-0.220990	0.013123	0.301812	0.334780	...
4	0.251125	-0.092620	-0.307521	-0.365733	-0.075228	0.414467	0.388034	...

759	760	761	762	763	764	765	\
0	-0.396520	0.011688	-0.208539	0.012130	-0.056947	-0.025428	-0.057884
1	-0.541656	-0.040561	0.047741	-0.063776	-0.138928	0.041252	-0.045966
2	-0.158719	-0.000505	-0.167245	0.169160	-0.084802	-0.366647	-0.106075
3	-0.408126	0.062470	0.140425	0.175495	-0.085699	-0.226141	-0.002507
4	-0.419398	-0.070503	-0.010642	0.323683	0.016842	-0.103515	-0.076240

766	767	cleaned_text
0	0.165974	0.302198
1	0.087562	0.546937
2	0.096242	0.567941
3	0.255287	0.496604
4	0.479977	0.274191

[5 rows x 771 columns]

```

import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

nltk.download('stopwords')
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Check if the input is a string before applying string methods
    if isinstance(text, str):
        text = text.lower() # Convert to lowercase
        text = re.sub(r'\W+', ' ', text) # Remove special characters
        # Keep words with length greater than 1 to avoid single characters
        text = ' '.join([lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words and len(word) > 1])
    # If not a string (e.g., an integer), potentially handle differently or skip

```

[Use code](#)  
[with caution](#)

► Sources

Rate  
this  
answer



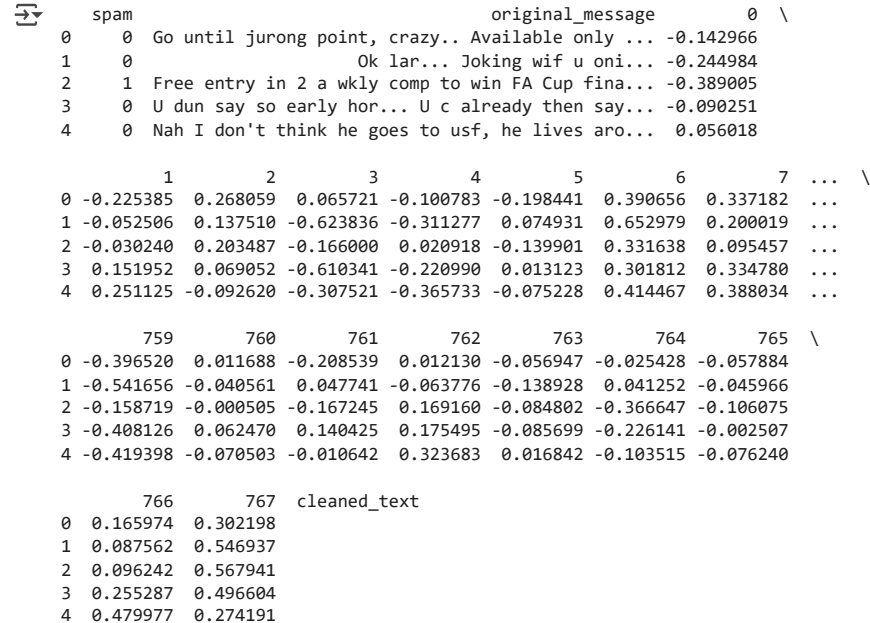
```

# Here, we just return the original value if it's not a string
return text

# Apply preprocessing, ensuring the 'spam' column is of string type
df['cleaned_text'] = df['spam'].astype(str).apply(preprocess_text)
print(df.head())

# Check if 'cleaned_text' contains any data after preprocessing
if any(df['cleaned_text']):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(df['cleaned_text']))
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.title("Most Common Words in SMS Messages")
    plt.show()
else:
    print("No words found for wordcloud generation. Check your preprocessing or data.")


```



```


```

spam	original_message	0	\
0	Go until jurong point, crazy.. Available only ...	-0.142966	
1	Ok lar... Joking wif u oni...	-0.244984	
2	Free entry in 2 a wkly comp to win FA Cup fina...	-0.389005	
3	U dun say so early hor... U c already then say...	-0.090251	
4	Nah I don't think he goes to usf, he lives aro...	0.056018	

1	2	3	4	5	6	7	...	\
-0.225385	0.268059	0.065721	-0.100783	-0.198441	0.390656	0.337182	...	
-0.052506	0.137510	-0.623836	-0.311277	0.074931	0.652979	0.200019	...	
-0.030240	0.203487	-0.166000	0.020918	-0.139901	0.331638	0.095457	...	
0.151952	0.069052	-0.610341	-0.220990	0.013123	0.301812	0.334780	...	
0.251125	-0.092620	-0.307521	-0.365733	-0.075228	0.414467	0.388034	...	

759	760	761	762	763	764	765	\
-0.396520	0.011688	-0.208539	0.012130	-0.056947	-0.025428	-0.057884	
-0.541656	-0.040561	0.047741	-0.063776	-0.138928	0.041252	-0.045966	
-0.158719	-0.000505	-0.167245	0.169160	-0.084802	-0.366647	-0.106075	
-0.408126	0.062470	0.140425	0.175495	-0.085699	-0.226141	-0.002507	
-0.419398	-0.070503	-0.010642	0.323683	0.016842	-0.103515	-0.076240	

766	767	cleaned_text
0	0.165974	0.302198
1	0.087562	0.546937
2	0.096242	0.567941
3	0.255287	0.496604
4	0.479977	0.274191

```

[5 rows x 771 columns]
No words found for wordcloud generation. Check your preprocessing or data.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!

print(df['cleaned_text'].head(10)) # See first 10 rows
print(df['cleaned_text'].isnull().sum()) # Count missing values
print(df['cleaned_text'].apply(lambda x: len(str(x).strip())).describe()) # Check text length

```

```


```

0	0
1	0
2	1
3	0
4	0
5	1
6	0
7	0
8	1
9	1

```

Name: cleaned_text, dtype: object
0
count    5572.0
mean       1.0
std        0.0
min        1.0
25%        1.0
50%        1.0
75%        1.0
max        1.0
Name: cleaned_text, dtype: float64

```

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

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

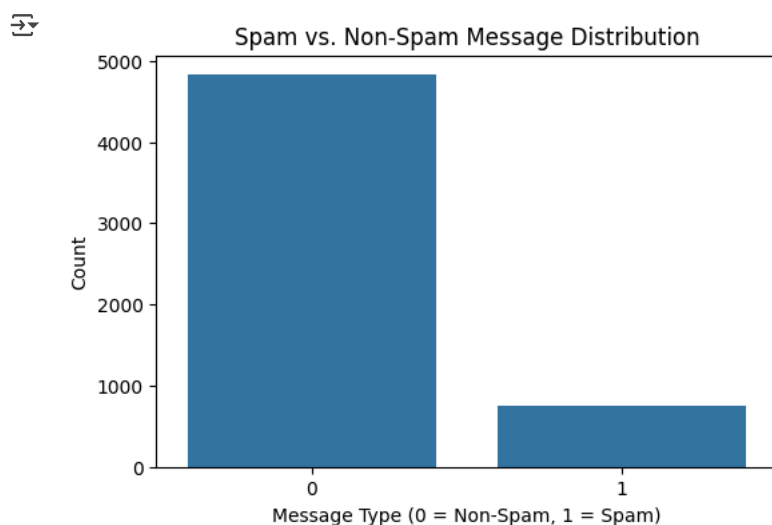
cleaned_text
0 go jurong point crazy available bugis n great ...
1           ok lar joking wif u oni
2 free entry 2 wkly comp win fa cup final tkts 2...
3           u dun say early hor u c already say
4           nah think go usf life around though

wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(df['cleaned_text']))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Most Common Words in SMS Messages")
plt.show()
```



<https://colab.research.google.com/drive/1E8rdTuRb-Att2bLkVeAEeERX6MSP6#scrollTo=BzxZcWNEAZHX&printMode=true>

```
# Print class counts
print(df['spam'].value counts())
```



spam	
0	4825
1	747

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Function to generate n-grams
```

```
def get_top_ngrams(corpus, ngram_range=(1,1), n=20):
    vectorizer = CountVectorizer(ngram_range=ngram_range, stop_words='english')
    X = vectorizer.fit_transform(corpus)
    word_counts = X.toarray().sum(axis=0)
    words_freq = sorted(list(zip(vectorizer.get_feature_names_out(), word_counts)), key=lambda x: x[1], reverse=True)
    return words_freq[:n]
```

```
# Get top unigrams
```

```
print("Top Unigrams:", get_top_ngrams(df['cleaned_text'], (1,1)))
```

```
# Get top bigrams
```

```
print("Top Bigrams:", get_top_ngrams(df['cleaned_text'], (2,2)))
```

```
# Get top trigrams
```

```
print("Top Trigrams:", get_top_ngrams(df['cleaned_text'], (3,3)))
```

Top Unigrams: [('ur', 385), ('gt', 318), ('lt', 316), ('ok', 292), ('free', 284), ('day', 273), ('know', 271), ('come  
Top Bigrams: [('lt gt', 276), ('let know', 42), ('sorry later', 38), ('good morning', 33), ('new year', 32), ('custom  
Top Trigrams: [('lt decimal gt', 23), ('happy new year', 19), ('lt gt min', 19), ('gt lt gt', 15), ('lt gt lt', 15),

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer = CountVectorizer()
X_bow = vectorizer.fit_transform(df['cleaned_text'])
```

```
print("Bow Shape:", X.bow.shape)
```

BoW Shape: (5572, 8016)

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

```
# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer()
```

```
# Fit the vectorizer to the 'cleaned_text' column and transform it
X_tfidf = vectorizer.fit_transform(df['cleaned_text'])
```

```
# Now you can use X_tfidf in train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df['spam'], test_size=0.2, random_state=42)
```

```
print("Train Set Shape:", X_train.shape)
print("Test Set Shape:", X_test.shape)
```

```
↗ Train Set Shape: (4457, 8016)
  Test Set Shape: (1115, 8016)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
# Predictions
y_pred = model.predict(X_test)
```

```
# Model Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
↗ Accuracy: 0.9426008968609866
```

	precision	recall	f1-score	support
0	0.94	0.99	0.97	965
1	0.95	0.61	0.74	150
accuracy			0.94	1115
macro avg	0.95	0.80	0.85	1115
weighted avg	0.94	0.94	0.94	1115

```
from sklearn.naive_bayes import MultinomialNB
```

```
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
```

```
print("Naïve Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb))
```

```
↗ Naïve Bayes Accuracy: 0.9659192825112107
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	965
1	1.00	0.75	0.85	150
accuracy			0.97	1115
macro avg	0.98	0.87	0.92	1115
weighted avg	0.97	0.97	0.96	1115

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

```
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
↗ Random Forest Accuracy: 0.979372197309417
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	965
1	0.99	0.85	0.92	150
accuracy			0.98	1115
macro avg	0.98	0.93	0.95	1115
weighted avg	0.98	0.98	0.98	1115

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
```

```
'C': [0.1, 1, 10],
'penalty': ['l1', 'l2']
}
```

```
grid_search = GridSearchCV(LogisticRegression(solver='liblinear'), param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
⚡ /usr/local/lib/python3.11/dist-packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to converge,
warnings.warn(
Best Parameters: {'C': 10, 'penalty': 'l1'}
Best Score: 0.9766639992350008
/usr/local/lib/python3.11/dist-packages/sklearn/svm/_base.py:1249: ConvergenceWarning: Liblinear failed to converge,
warnings.warn(
```

```
import joblib
```

```
# Save the trained model
joblib.dump(grid_search.best_estimator_, "spam_classifier.pkl")
```

```
# Save the vectorizer (TF-IDF) - Changed 'tfidf' to 'vectorizer'
joblib.dump(vectorizer, "tfidf_vectorizer.pkl")
```

```
⚡ ['tfidf_vectorizer.pkl']
```

```
# Load the model
model = joblib.load("spam_classifier.pkl")
vectorizer = joblib.load("tfidf_vectorizer.pkl")
```

```
# Example test message
new_message = ["Congratulations! You won a free lottery ticket."]
new_message_transformed = vectorizer.transform(new_message)
```

```
# Predict spam or ham
prediction = model.predict(new_message_transformed)
print("Prediction:", "Spam" if prediction[0] == 1 else "Ham")
```

```
⚡ Prediction: Ham
```

```
pip install flask
```

```
⚡ Requirement already satisfied: flask in /usr/local/lib/python3.11/dist-packages (3.1.0)
Requirement already satisfied: Werkzeug>=3.1 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from flask) (3.1.5)
Requirement already satisfied: itsdangerous>=2.2 in /usr/local/lib/python3.11/dist-packages (from flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from flask) (8.1.8)
Requirement already satisfied: blinker>=1.9 in /usr/local/lib/python3.11/dist-packages (from flask) (1.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->flask)
```

```
from flask import Flask, request, jsonify
import joblib
```

```
app = Flask(__name__)
```

```
# Load model & vectorizer
model = joblib.load("spam_classifier.pkl")
vectorizer = joblib.load("tfidf_vectorizer.pkl")
```

```
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get message from the request data
        data = request.json

        # Check if the message key is in the input data
        if 'message' not in data:
            return jsonify({"error": "Message not provided"}), 400
```

```

message = data['message']

# Transform the message using the TF-IDF vectorizer
transformed_message = vectorizer.transform([message])

# Make prediction using the trained model
prediction = model.predict(transformed_message)

# Return the prediction result as a JSON response
return jsonify({"prediction": "Spam" if prediction[0] == 1 else "Ham"})

except Exception as e:
    # Handle unexpected errors and return a 500 error with the error message
    return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)

* Serving Flask app '__main__'
* Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat

{
  "message": "Congratulations! You've won a free iPhone!"
}

{'message': "Congratulations! You've won a free iPhone!"}

!python app.py

python3: can't open file '/content/app.py': [Errno 2] No such file or directory

```

Enter a

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. [Learn more](#)