## Training

```
import numpy as np
import pandas as pd
# drop last 3 cols
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
import nltk
nltk.download('punkt')
import nltk
nltk.download('punkt_tab')
# num of words
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
df[['num_characters','num_words','num_sentences']].describe()
# ham

df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
#spam
df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_characters'])
sns.histplot(df[df['target'] == 1]['num_characters'],color='red')
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
# Calculate the correlation matrix only for numerical features

numerical_df = df.select_dtypes(include=np.number)  # Select columns with numerical data types
corr_matrix = numerical_df.corr()


# Display the heatmap using the correlation matrix of numerical features
sns.heatmap(corr_matrix, annot=True)
plt.show()



def transform_text(text):
```

```
    text = text.lower()
    text = nltk.word_tokenize(text)



    y = []

    for i in text:

if i.isalnum():

y.append(i)



    text = y[:]

    y.clear()



    for i in text:

        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)


    text = y[:]

    y.clear()



    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
import nltk
from nltk.corpus import stopwords
import string


nltk.download('stopwords')
```

```
def transform_text(text):
    text = text.lower()
```

```
    text = nltk.word_tokenize(text)



    y = []

    for i in text:

if i.isalnum():

y.append(i)



    text = y[:]

    y.clear()



    for i in text:

        if i not in stopwords.words('english') and i not in string.punctuation:

            y.append(i)



    text = y[:]

    y.clear()



    for i in text:

        y.append(ps.stem(i))



    return " ".join(y)

from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

ps.stem('loving')

df['transformed_text'] = df['text'].apply(transform_text)

from wordcloud import WordCloud

wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

```
spam_wc = wc.generate(df[df['target'] == 1]['transformed_text'].str.cat(sep=" "))
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
spam_corpus = []
```

```
for msg in df[df['target'] == 1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)



from collections import Counter

# Assuming the DataFrame has columns named 'word' and 'frequency'

spam_df = pd.DataFrame(Counter(spam_corpus).most_common(30), columns=['word', 'frequency'])



# Create the barplot using the 'x' and 'y' keywords
sns.barplot(x='word', y='frequency', data=spam_df)
plt.xticks(rotation='vertical')
plt.show()
ham_corpus = []
for msg in df[df['target'] == 0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
from collections import Counter
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd


# Assuming the DataFrame has columns named 'word' and 'frequency'

ham_df = pd.DataFrame(Counter(ham_corpus).most_common(30), columns=['word', 'frequency'])



# Create the barplot using the 'x' and 'y' keywords
sns.barplot(x='word', y='frequency', data=ham_df)
plt.xticks(rotation='vertical')
plt.show()

from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

```
#from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler()
#X = scaler.fit_transform(X)

from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
gnb.fit(X_train,y_train)
y_pred1 = gnb.predict(X_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
mnb.fit(X_train,y_train)
y_pred2 = mnb.predict(X_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_test,y_pred2))
print(precision_score(y_test,y_pred2))
bnb.fit(X_train,y_train)
y_pred3 = bnb.predict(X_test)
print(accuracy_score(y_test,y_pred3))
print(confusion_matrix(y_test,y_pred3))
print(precision_score(y_test,y_pred3))
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
```

```
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
mnb = MultinomialNB()

dtc = DecisionTreeClassifier(max_depth=5)

lrc = LogisticRegression(solver='liblinear', penalty='l1')

rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
xgb = XGBClassifier(n_estimators=50,random_state=2)
clfs = {

    'SVC' : svc,

    'KN' : knc,

    'NB': mnb,

    'DT': dtc,

    'LR': lrc,

    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,

    'GBDT':gbdt,

    'xgb':xgb

}

def train_classifier(clf,X_train,y_train,X_test,y_test):
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test,y_pred)
```

```
    precision = precision_score(y_test,y_pred)



    return accuracy,precision
accuracy_scores = []
precision_scores = []


for name,clf in clfs.items():



    current_accuracy,current_precision = train_classifier(clf, X_train,y_train,X_test,y_test)



    print("For ",name)

    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)


    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
sns.catplot(x = 'Algorithm', y='value',

            hue = 'variable',data=performance_df1, kind='bar',height=5)
plt.ylim(0.5,1.0)
plt.xticks(rotation='vertical')
plt.show()
# Voting Classifier

svc = SVC(kernel='sigmoid', gamma=1.0,probability=True)
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)



from sklearn.ensemble import VotingClassifier
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
        # Applying stacking
estimators=[('svm', svc), ('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier() clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred)) import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```

## Deployment

```
import streamlit as st
import joblib
import pandas as
pd import sklearn
import pickle
import string
from nltk.corpus import stopwords
import nltk
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
if i.isalnum():
y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()



    for i in text:
```

```
        y.append(ps.stem(i))

    return " ".join(y)

tfidf = joblib.load('vectorizer.pkl')



model = pickle.load(open('model.pkl','rb'))
st.title("Email Spam Detection")
input_sms = st.text_area("Enter the message")
if st.button('Predict'):
    # 1. preprocess
    transformed_sms = transform_text(input_sms)
    # 2. vectorize
    vector_input = tfidf.transform([transformed_sms])
    # 3. predict
    result = model.predict(vector_input)[0]
    # 4. Display
    if result == 1:
        st.header("Spam")
    else:
        st.header("Not Spam”)
```

## Output

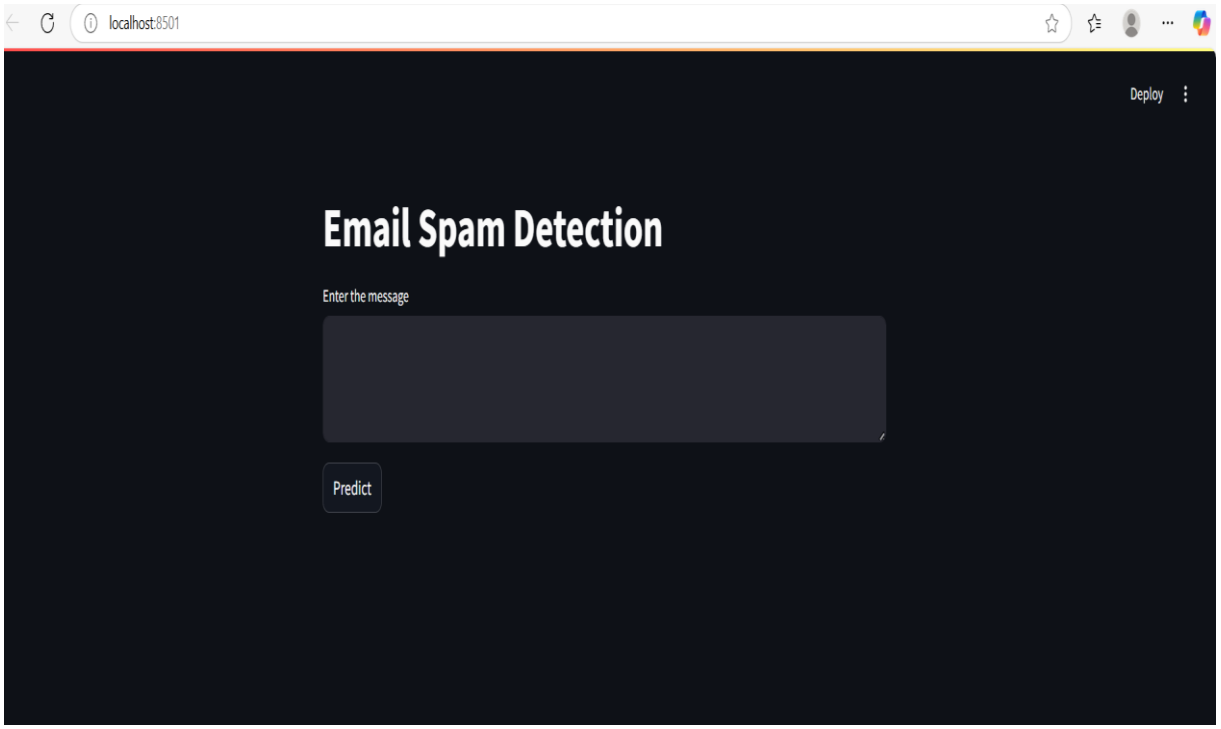


**Fig A** Web App

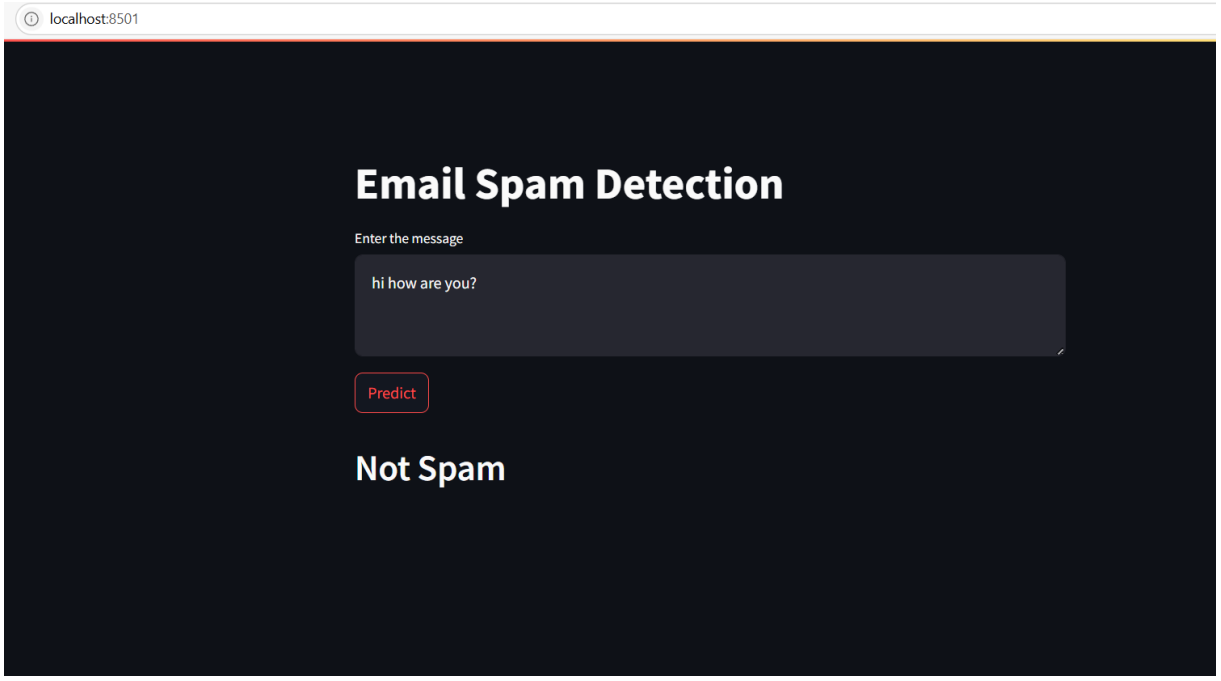

**Fig B** Prediction Result 1

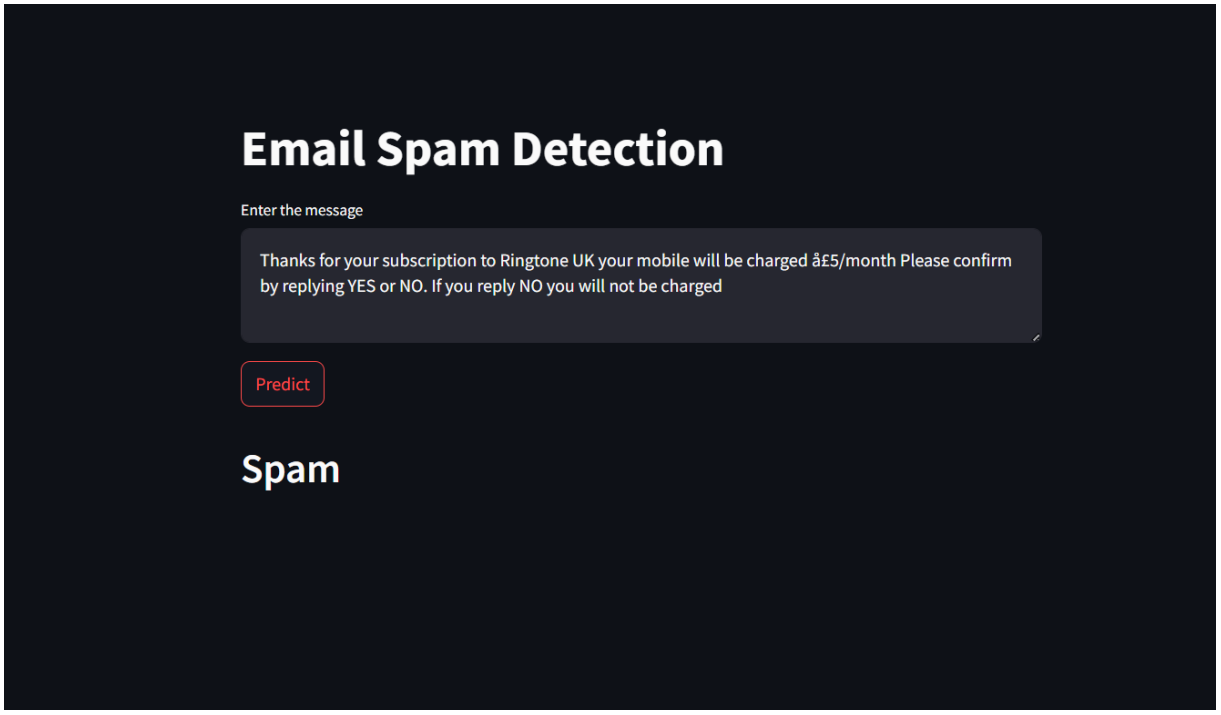


**Fig C** Prediction Result 2