**SMS Spam Detection**

#library and packages

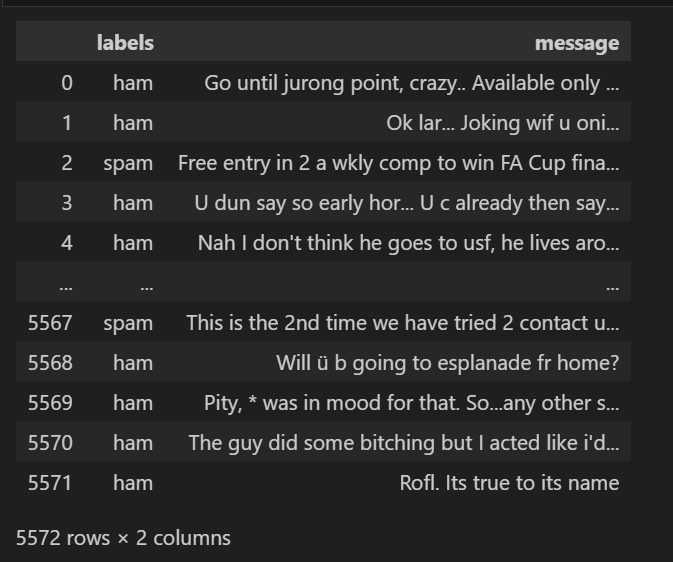
import numpy as np

import pandas as pd

#import dataset

dataset=pd.read\_csv("SMSSpamCollection",sep='\t',names=['labels','message'])

dataset

****

dataset.info()

**A screen shot of a computer

Description automatically generated**

dataset.describe() #to get count of labels

**A screenshot of a phone

Description automatically generated**

dataset['labels'] = dataset['labels'].map({'ham': 0, 'spam': 1}) # mapping with 0 and 1

dataset

**A screenshot of a black and white screen

Description automatically generated**

#visulation

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

# count plot for spam vs ham to check imbalance

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

g = sns.countplot(x="labels", data=dataset)

plt.title('Countplot for spam vs ham')

plt.xlabel('Is the SMS spam')

plt.ylabel('Count')

**A graph of a spam

Description automatically generated**

#so we can see the data is imbalance

#handling imbalance dataset with oversampling

only\_spam=dataset[dataset["labels"]==1]

only\_spam

**A screenshot of a black and white screen

Description automatically generated**

len(dataset)-len(only\_spam)

#so we need to replicate spam 6-7 times to make dataset imbaalnce

count=int((dataset.shape[0]-only\_spam.shape[0])/only\_spam.shape[0])

count

****

for i in range(0,count-1):

    dataset=pd.concat([dataset,only\_spam])

dataset.shape

****

# count plot for spam vs ham to check imbalance

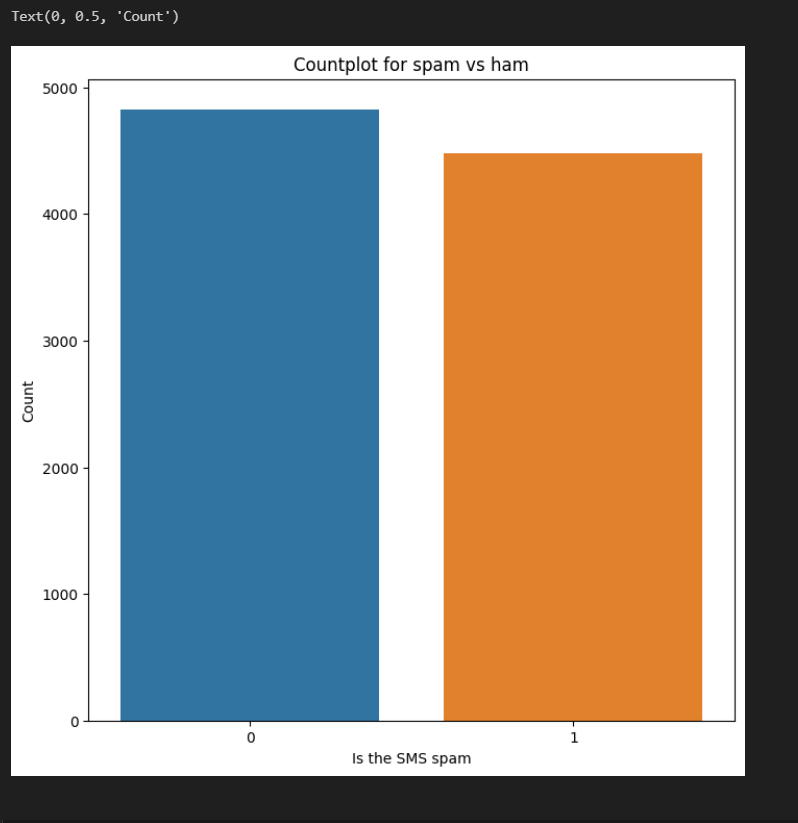
plt.figure(figsize=(8,8))

g = sns.countplot(x="labels", data=dataset)

plt.title('Countplot for spam vs ham')

plt.xlabel('Is the SMS spam')

plt.ylabel('Count')

****

# creating new features

dataset['word\_count']=dataset['message'].apply(lambda x: len(x.split()))

dataset

**A screenshot of a black and white message

Description automatically generated**

plt.figure(figsize=(12,6))

#(1,1)

plt.subplot(1,2,1)

g=sns.histplot(dataset[dataset["labels"]==0].word\_count,kde=True)

p=plt.title('Distribution of word count for ham msgs')

#(1,2)

plt.subplot(1,2,2)

g=sns.histplot(dataset[dataset["labels"]==1].word\_count,color="red",kde=True)

p=plt.title('Distribution of word count for spam msgs')

plt.tight\_layout() ## padding in between

plt.show()

**A screenshot of a graph

Description automatically generated**

#createing new feature of containing\_currency\_symbol

def currency\_present(data):

    currency\_symbols=['¢','$','€','£','₹']

    for i in currency\_symbols:

        if i in data:

            return 1

    return 0

dataset["contains\_currency\_symbols"]=dataset["message"].apply(currency\_present)

dataset

**A screenshot of a computer

Description automatically generated**

#countplot for contains\_currency\_symbol

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

g=sns.countplot(x='contains\_currency\_symbols',data=dataset,hue="labels")

p=plt.title('Countplot for containing currency symbols')

p=plt.xlabel('does SMS contains currency\_symbol')

p=plt.ylabel('count')

p=plt.legend(labels=["Ham","Spam"],loc=9)

**A graph with numbers and a bar

Description automatically generated**

#creating new feature of containing number

#ord function to convert into ascii and then again to normal

def number(data):

    for i in data:

        if ord(i) >= 48 and ord(i) <= 57:

            return 1

    return 0

dataset["contains\_number"]=dataset['message'].apply(number)

dataset

**A screenshot of a computer

Description automatically generated**

#countplot for how many msgs have numbers

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

g=sns.countplot(x='contains\_number',data=dataset,hue="labels")

p=plt.title('Countplot for containing numbers')

p=plt.xlabel('does SMS contains numbers')

p=plt.ylabel('count')

p=plt.legend(labels=["Ham","Spam"],loc=9)

**A graph of numbers with blue and orange squares

Description automatically generated**

#data cleaning because unstructured data

import nltk

import re

nltk.download('stopwords')

nltk.download('wordnet')

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

**A screen shot of a computer

Description automatically generated**

corpus = []

wnl = WordNetLemmatizer()

for sms in list(dataset.message):

    message = re.sub(pattern='[^a-zA-Z]', repl=' ', string=sms)  # filtering out special characters and numbers

    message = message.lower()

    words = message.split()  # tokenizing (added parentheses after split)

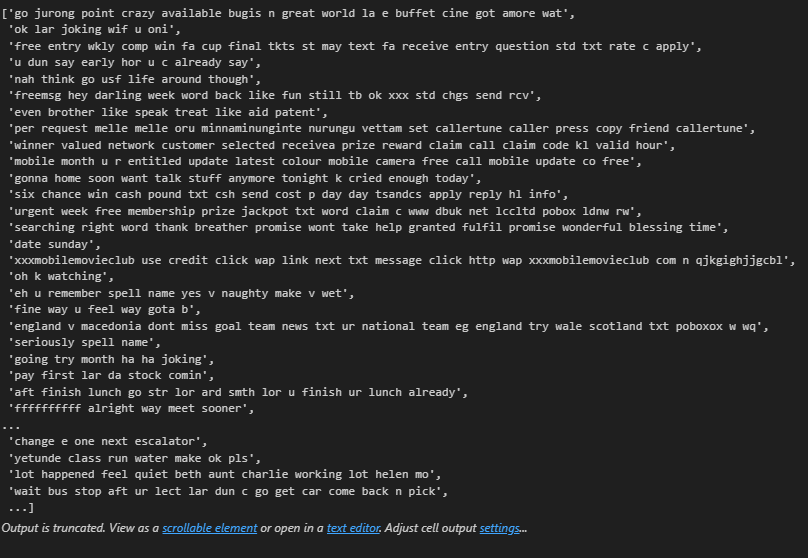
    filtered\_words = [word for word in words if word not in set(stopwords.words('english'))]

    lemm\_words = [wnl.lemmatize(word) for word in filtered\_words]

    message = ' '.join(lemm\_words)

    corpus.append(message)

corpus

****

#creating the bag of words model

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max\_features=500)

vectors = tfidf.fit\_transform(corpus).toarray()

feature\_names = tfidf.get\_feature\_names()

#seprating dependent and independent

x=pd.DataFrame(vectors,columns=feature\_names)

y=dataset['labels']

#cross validation report

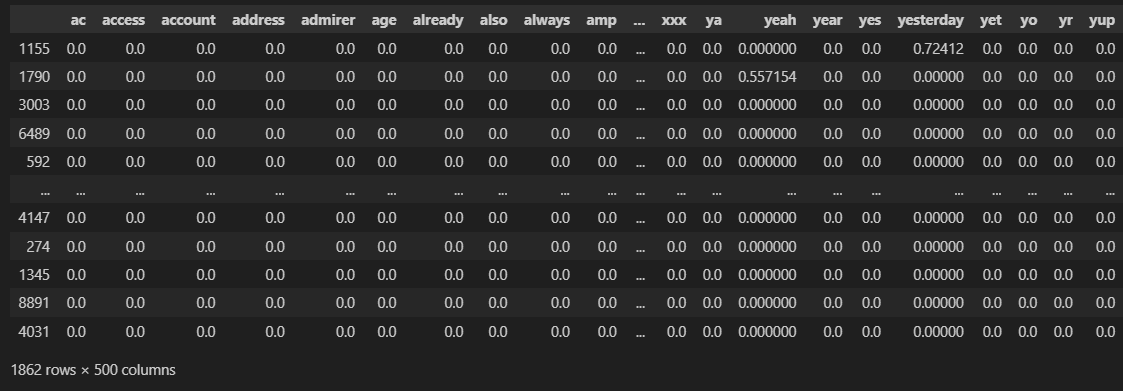
#creating cnfusion matrix

from sklearn.model\_selection import cross\_val\_score,train\_test\_split

from sklearn.metrics import classification\_report,confusion\_matrix

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

x\_test



#trining using naive bayes

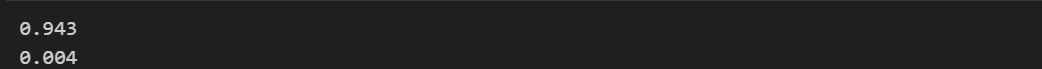
from sklearn.naive\_bayes import MultinomialNB

mnb=MultinomialNB()

cv=cross\_val\_score(mnb,x,y,scoring= 'f1',cv=10)

print(round(cv.mean(),3))

print(round(cv.std(),3))



mnb.fit(x\_train,y\_train)

y\_pred=mnb.predict(x\_test)

print(classification\_report(y\_test,y\_pred))

A screenshot of a computer screen

Description automatically generated

cm = confusion\_matrix(y\_test, y\_pred)

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

axis\_labels = ['ham', 'spam']

g = sns.heatmap(data=cm, xticklabels=axis\_labels, yticklabels=axis\_labels, annot=True, fmt='g', cbar\_kws={"shrink": 0.5})

plt.title("Confusion Matrix of our Multinomial Naive Bayes Model")

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.show()

A screenshot of a graph

Description automatically generated

#using decision tree

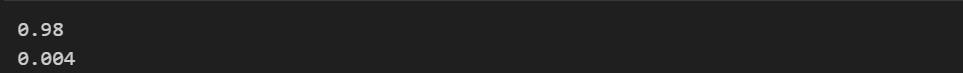
from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()

cv1=cross\_val\_score(dt,x,y,scoring='f1',cv=10)

print(round(cv1.mean(),3))

print(round(cv1.std(),3))



dt.fit(x\_train,y\_train)

y\_pred1=dt.predict(x\_test)

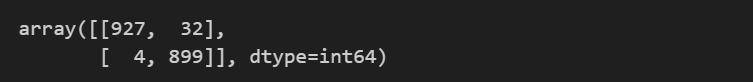
print(classification\_report(y\_test,y\_pred1))

A screenshot of a computer screen

Description automatically generated

cm = confusion\_matrix(y\_test, y\_pred1)

cm



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

axis\_labels = ['ham', 'spam']

g = sns.heatmap(data=cm, xticklabels=axis\_labels, yticklabels=axis\_labels, annot=True, fmt='g', cbar\_kws={"shrink": 0.5})

plt.title("Confusion Matrix of our Multinomial Naive Bayes Model")

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.show()

A screenshot of a graph

Description automatically generated

def predict\_spam(sms):

    message = re.sub(pattern='[^a-zA-Z]', repl=' ', string=sms)  # filtering out special characters and numbers

    message = message.lower()

    words = message.split()  # tokenizing (added parentheses after split)

    filtered\_words = [word for word in words if word not in set(stopwords.words('english'))]

    lemm\_words = [wnl.lemmatize(word) for word in filtered\_words]

    message = ' '.join(lemm\_words)

    temp=tfidf.transform([message]).toarray()

    return dt.predict(temp)

#prediction 1

sample\_message=""

if predict\_spam(sample\_message):

    print('this is a spam message')

else:

    print('this is not a spam message')