Naïve Bayes code:

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import os

print(os.listdir("../input"))

message = [line.rstrip() for line in open('../input/SMSSpamCollection')]

print(len(message))

for message\_no,message in enumerate(message[:10]):

    print(message\_no,message)

    print('\n')

import pandas as pd

message=pd.read\_csv('../input/SMSSpamCollection' ,encoding='cp1252',sep='\t',names=["labels","message"])

message.head()

message.describe()

message.groupby('labels').describe()

message['length']=message['message'].apply(len)

message.head()

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

message['length'].plot(bins=50,kind='hist')

message.length.describe()

message[message['length']==910]['message'].iloc[0]

import string

mess = 'sample message!...'

nopunc=[char for char in mess if char not in string.punctuation]

nopunc=''.join(nopunc)

print(nopunc)

from nltk.corpus import stopwords

stopwords.words('english')[0:10]

nopunc.split()

clean\_mess=[word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

clean\_mess

def text\_process(mess):

    nopunc =[char for char in mess if char not in string.punctuation]

    nopunc=''.join(nopunc)

    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]

message.head()

message['message'].head(5).apply(text\_process)

message.head()

from sklearn.feature\_extraction.text import CountVectorizer

bow\_transformer = CountVectorizer(analyzer=text\_process).fit(message['message'])

print(len(bow\_transformer.vocabulary\_))

message4=message['message'][3]

print(message4)

bow4=bow\_transformer.transform([message4])

print(bow4)

print(bow4.shape)

print(bow\_transformer.get\_feature\_names()[4073])

print(bow\_transformer.get\_feature\_names()[9570])

messages\_bow = bow\_transformer.transform(message['message'])

print('Shape of Sparse Matrix: ',messages\_bow.shape)

print('Amount of non-zero occurences:',messages\_bow.nnz)

sparsity =(100.0 \* messages\_bow.nnz/(messages\_bow.shape[0]\*messages\_bow.shape[1]))

print('sparsity:{}'.format(round(sparsity)))

from sklearn.feature\_extraction.text import TfidfTransformer

tfidf\_transformer=TfidfTransformer().fit(messages\_bow)

tfidf4 = tfidf\_transformer.transform(bow4)

print(tfidf4)

print(tfidf\_transformer.idf\_[bow\_transformer.vocabulary\_['u']])

print(tfidf\_transformer.idf\_[bow\_transformer.vocabulary\_['university']])

messages\_tfidf=tfidf\_transformer.transform(messages\_bow)

print(messages\_tfidf.shape)

from sklearn.naive\_bayes import MultinomialNB

spam\_detect\_model = MultinomialNB().fit(messages\_tfidf,message['labels'])

print('predicted:',spam\_detect\_model.predict(tfidf4)[0])

print('expected:',message.labels[3])

all\_predictions = spam\_detect\_model.predict(messages\_tfidf)

print(all\_predictions)

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

print(classification\_report(message['labels'],all\_predictions))

print(confusion\_matrix(message['labels'],all\_predictions))

from sklearn.model\_selection import train\_test\_split

msg\_train,msg\_test,label\_train,label\_test = train\_test\_split(message['message'],message['labels'],test\_size=0.2)

print(len(msg\_train),len(msg\_test),len(label\_train),len(label\_test))

from sklearn.pipeline import Pipeline

pipeline = Pipeline([

   ( 'bow',CountVectorizer(analyzer=text\_process)),

    ('tfidf',TfidfTransformer()),

    ('classifier',MultinomialNB()),

])

pipeline.fit(msg\_train,label\_train)

predictions = pipeline.predict(msg\_test)

print(classification\_report(predictions,label\_test))