import numpy as np

import pandas as pd

df = pd.read\_csv('spam.csv')

df.sample(5)

df.shape

df.info()

df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'], inplace=True)

df.sample(5)

df.rename(columns={'v1':'target','v2':'text'}, inplace=True)

df.sample(5)

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['target'] = encoder.fit\_transform(df['target'])

df.head()

df.isnull().sum()

df.duplicated().sum()

df = df.drop\_duplicates(keep='first')

df.duplicated().sum()

df.shape

df.head()

df['target'].value\_counts()

import matplotlib.pyplot as plt

plt.pie(df['target'].value\_counts(), labels=['ham','spam'], autopct="%0.2f")

plt.show()

import nltk

!pip install nltk

nltk.download('punkt')

df['num\_characters'] = df['text'].apply(len)

df.head()

df['num\_words'] = df['text'].apply(lambda x:len(nltk.word\_tokenize(x)))

df.head()

df['num\_sentences'] = df['text'].apply(lambda x:len(nltk.sent\_tokenize(x)))

df.head()

df[['num\_characters','num\_words','num\_sentences']].describe()

df[df['target'] == 0][['num\_characters','num\_words','num\_sentences']].describe()

df[df['target'] == 1][['num\_characters','num\_words','num\_sentences']].describe()

import seaborn as sns

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')

sns.pairplot(df, hue='target')

sns.heatmap(df.corr(), annot=True)

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)

transform\_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.")

df['text'][10]

from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

ps.stem('loving')

df['transformed\_text'] = df['text'].apply(transform\_text)

df.head()

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(spam\_wc)

ham\_wc = wc.generate(df[df['target'] == 0]['transformed\_text'].str.cat(sep=" "))

plt.figure(figsize=(15,6))

plt.imshow(ham\_wc)

df.head()

spam\_corpus = []

for msg in df[df['target'] == 1]['transformed\_text'].tolist():

for word in msg.split():

spam\_corpus.append(word)

len(spam\_corpus)

from collections import Counter

sns.barplot(pd.DataFrame(Counter(spam\_corpus).most\_common(30))[0],

pd.DataFrame(Counter(spam\_corpus).most\_common(30))[1])

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)

len(ham\_corpus)

from collections import Counter

sns.barplot(pd.DataFrame(Counter(ham\_corpus).most\_common(30))[0],

pd.DataFrame(Counter(ham\_corpus).most\_common(30))[1])

plt.xticks(rotation='vertical')

plt.show()

df.head()

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

cv = CountVectorizer()

tfidf = TfidfVectorizer(max\_features=3000)

X = tfidf.fit\_transform(df['transformed\_text']).toarray()

X.shape

y = df['target'].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2)

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, AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier, 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

train\_classifier(svc, X\_train, y\_train, X\_test, y\_test)

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)

performance\_df = pd.DataFrame({'Algorithm': clfs.keys(), 'Accuracy': accuracy\_scores, 'Precision': precision\_scores}).sort\_values('Precision', ascending=False)

performance\_df

performance\_df1 = pd.melt(performance\_df, id\_vars = "Algorithm")

performance\_df1

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()

temp\_df = pd.DataFrame({'Algorithm': clfs.keys(), 'Accuracy\_max\_ft\_3000': accuracy\_scores, 'Precision\_max\_ft\_3000': precision\_scores}).sort\_values('Precision\_max\_ft\_3000', ascending=False)

temp\_df = pd.DataFrame({'Algorithm': clfs.keys(), 'Accuracy\_scaling': accuracy\_scores, 'Precision\_scaling': precision\_scores}).sort\_values('Precision\_scaling', ascending=False)

new\_df = performance\_df.merge(temp\_df, on='Algorithm')

new\_df\_scaled = new\_df.merge(temp\_df, on='Algorithm')

temp\_df = pd.DataFrame({'Algorithm': clfs.keys(), 'Accuracy\_num\_chars': accuracy\_scores, 'Precision\_num\_chars': precision\_scores}).sort\_values('Precision\_num\_chars', ascending=False)

new\_df\_scaled.merge(temp\_df, on='Algorithm')

# 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

voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)], voting='soft')

voting.fit(X\_train, y\_train)

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()

from sklearn.ensemble import StackingClassifier

clf = StackingClassifier(estimators=estimators, final\_estimator=final\_estimator)

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'))