```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import pandas as pd
import numpy as np
# In[2]:
import pandas as pd
# List of possible encodings to try
encodings=['utf-8','latin1','cp1252']
file_path='spam.csv' # Change this with different encodings
# Attempt to read the CSV file with different encodings
for encoding in encodings:
    try:
        df=pd.read_csv(file_path,encoding=encoding)
        print(f"file successfully read with encoding:{encoding}")
        break
    except UnicodeDecodeError:
            print(f"failed to read with encoding:{encoding}")
            continue # Try the next coding
if 'df' in locals():
    print("CSV file has been successfully loaded.")
else:
    print("All encoding attempts failed. Unable to read the CSV file.")
# In[3]:
df.sample(5)
# In[4]:
df.shape
# In[5]:
df.info()
# In[6]:
df.sample(5)
# In[7]:
df.rename(columns={'type':'target'},inplace=True)
df.sample(5)
# In[8]:
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
# In[9]:
df['target'] =encoder.fit_transform(df['target'])
# In[10]:
df.head()
# In[11]:
df.isnull().sum()
# In[12]:
df.duplicated().sum()
# In[13]:
df=df.drop_duplicates(keep='first')
df.duplicated().sum()
# In[14]:
df.shape
# In[15]:
df.head()
# In[16]:
df["target"].value_counts()
# In[17]:
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(),labels=['ham','spam'],autopct="%0.2f"
)
```

```
plt.show()
# In[18]:
import nltk
nltk.download('punkt')
# In[19]:
df['num_characters']=df['text'].apply(len)
# In[20]:
df.head()
# In[22]:
df['num_words']=df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
df.head()
# In[23]:
df['num_sentences']=df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
df.head()
# In[24]:
df[['num_characters','num_words','num_sentences']].describe()
# In[25]:
df[df['target']==0][['num_characters','num_words','num_sentences']].descr
ibe()
# In[26]:
import seaborn as sns
# In[27]:
plt.figure(figsize=(12,6))
sns.histplot(df[df['target']==0]['num_characters'])
sns.histplot(df[df['target']==1]['num_characters'],color='red')
# In[28]:
plt.figure(figsize=(12,6))
sns.histplot(df[df['target']==0]['num_words'])
sns.histplot(df[df['target']==1]['num_words'],color='red')
# In[29]:
sns.pairplot(df,hue='target')
# In[30]:
sns.heatmap(df.corr(),annot=True)
# In[31]:
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string
nltk.download('stopwords')
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:
        y.append(ps.stem(i))

    return " ".join(y)
transformed_text=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.")
```

```
print(transformed_text)
# In[32]:
df['text'][10]
# In[33]:
from nltk.stem.porter import PorterStemmer
ps=PorterStemmer()
ps.stem('loving')
# In[34]:
df['transformed_text']=df['text'].apply(transform_text)
df.head()
# In[35]:
from wordcloud import WordCloud
wc=WordCloud(width=500,height=500,min_font_size=10,background_color='whit
e')
# In[36]:
spam_wc =
wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
# In[37]:
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
# In[38]:
ham_wc =
wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
# In[39]:
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
# In[40]:
df.head()
# In[41]:
spam_corpus=[]
for msg in df[df['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)
# In[42]:
len(spam_corpus)
# In[45]:
get_ipython().system('pip install counter')
# In[53]:
ham_corpus=[]
for msg in df[df['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
# In[54]:
len(ham_corpus)
# In[55]:
df.head()
# In[56]:
from sklearn.feature_extraction.text import
CountVectorizer,TfidfVectorizer
cv= CountVectorizer()
tfidf=TfidfVectorizer(max_features=3000)
# In[57]:
X= tfidf.fit_transform(df['transformed_text']).toarray()
# In[58]:
X.shape
# In[59]:
y=df['target'].values
# In[60]:
```

```
from sklearn.model_selection import train_test_split
# In[61]:
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_s
tate=2)
# In[62]:
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import
accuracy_score,confusion_matrix,precision_score
# In[63]:
gnb=GaussianNB()
mnb=MultinomialNB()
bnb=BernoulliNB()
# In[64]:
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))
# In[65]:
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))
# In[66]:
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))
# In[67]:
get_ipython().system('pip install xgboost')
# In[68]:
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
# In[69]:
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)
# In[70]:
clfs={
    'SVC' : svc,
    'KN'  :knc,
    'NB'   :mnb,
    'DT' :dtc,
    'LR' :lrc,
    'RF'  :rfc,
    'AdaBoost':abc,
    'BgC':bc,
    'ETC':etc,
    'GBDT':gbdt,
    'xgb':xgb
}
# In[71]:
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
# In[72]:
train_classifier(svc,X_train,y_train,X_test,y_test)
# In[73]:
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)
# In[75]:
performance_df=pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_
scores,'Precision':precision_scores}).sort_values('Precision',ascending=F
alse)
# In[76]:
performance_df
# In[77]:
performance_df1=pd.melt(performance_df,id_vars="Algorithm")
performance_df1
# In[78]:
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()
# In[79]:
temp_df=pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accu
racy_scores,'Precision_max_ft_3000':precision_scores}).sort_values('Preci
sion_max_ft_3000',ascending=False)
# In[80]:
new_df=performance_df.merge(temp_df,on='Algorithm')
# In[81]:
new_df_scaled=new_df.merge(temp_df,on='Algorithm')
```

```
# In[82]:
temp_df=pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accura
cy_scores,'Precision_num_chars':precision_scores}).sort_values('Precision
_num_chars',ascending=False)
# In[83]:
new_df_scaled.merge(temp_df,on='Algorithm')
# In[84]:
svc=SVC(kernel='sigmoid',gamma=1.0,probability=True)
mnb=MultinomialNB()
etc=ExtraTreesClassifier(n_estimators=50,random_state=2)
from sklearn.ensemble import VotingClassifier
# In[85]:
voting=VotingClassifier(estimators=[('svm',svc),('nb',mnb),('et',etc)],vo
ting='soft')
# In[86]:
voting.fit(X_train,y_train)
# In[87]:
y_pred=voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
# In[88]:
estimators=[('svm',svc),('nb',mnb),('et',etc)]
final_estimator=RandomForestClassifier()
# In[89]:
from sklearn.ensemble import StackingClassifier
# In[90]:
clf=StackingClassifier(estimators=estimators,final_estimator=final_estima
tor)
# In[91]:
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))
# In[92]:
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
# In[93]:
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
X_train=["Sample text 1","Sample text 2","Sample text 3"]
y_train=[0,1,0]
tfidf=TfidfVectorizer(lowercase=True,stop_words='english')
X_train_tfidf=tfidf.fit_transform(X_train)
mnb.fit(X_train_tfidf,y_train)
with open('vectorizer.pkl','wb')as vectorizer_file:
    pickle.dump(tfidf,vectorizer_file)
with open('model.pkl','wb')as model_file:
        pickle.dump(tfidf,model_file)

# In[ ]:
```