Spam Classification Model

- The goal of this code is to develop an effective spam classification model using machine learning techniques.
- Multiple classification models are trained and evaluated to identify the best-performing one in terms of accuracy and precision.
- Additionally, ensemble methods, such as Voting and Stacking Classifiers, are explored to potentially improve classification performance.
- The code aims to provide insights into the performance of various machine learning algorithms for the task of spam detection and offers a comprehensive example of the end-to-end machine learning pipeline for text classification.

In [265]:
```python
import numpy as np
import pandas as pd
```

In [266]:
```python
df = pd.read_csv('dataset/spam.csv', encoding='latin1')
```

-> 'utf-8' encoding throws an error because it can't decode the csv file therefore i'm using 'latin1' encoding

In [267]:
```python
df.sample(5)
```

Out[267]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
2608	ham	Hello madam how are you ?	NaN	NaN	NaN
1639	spam	FreeMsg:Feelin kinda lnly hope u like 2 keep m...	NaN	NaN	NaN
1799	ham	If we hit it off, you can move in with me :)	NaN	NaN	NaN
2793	ham	The affidavit says <#> E Twiggs St, di...	NaN	NaN	NaN
3934	ham	You need to get up. Now.	NaN	NaN	NaN

In [268]:
```python
df.shape
```

Out[268]:
```
(5572, 5)
```

| Data Cleaning

-we have to clean our dataset since it has a lot of 'NaN' values and other possible faults in the data.

In [269]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

In [270]:
```
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

-> dropping the last 3 columns since the last 3 columns: "Unnamed" have very little non-null data (50, 12, 6)

In [271]:
```
df.sample(5)
```

Out[271]:

	v1	v2
5459	ham	If you hear a loud scream in about <#> m...
5053	spam	Double Mins & Double Txt & 1/2 price Linerenta...
464	ham	Ok i am on the way to railway
2701	ham	Hiya, sorry didn't hav signal. I haven't seen ...
3018	ham	Wat time do u wan 2 meet me later?

-> i'm changing the column names from v1 to target, and from v2 to text

In [272]:
```
df.rename(columns={'v1':'target','v2':'text'},inplace=True)
df.sample(5)
```

Out[272]:

	target	text
3866	ham	Alright we're hooked up, where you guys at
2924	ham	Are you coming to day for class.
563	spam	GENT! We are trying to contact you. Last weeke...
5066	spam	83039 62735=å£450 UK Break AccommodationVouche...
5229	ham	It means u could not keep ur words.

In [273]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

-> i'm using LabelEncoder to replace the values of our 'target' column values from ham/spam to 1/0 which will make it easy for us to make the model

ham - 0
spam - 1

In [274]:
```
df['target'] = encoder.fit_transform(df['target'])
```

In [275]:
```
df.head()
```

Out[275]:

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

-> now we need to check for Null and Duplicate values in our data

In [276]:
```
df.isnull().sum()
```

Out[276]:
```
target    0
text      0
dtype: int64
```

In [277]:
```
df.duplicated().sum()
```

Out[277]:
```
403
```

-> we need to remove these duplicate values from our data

In [278]:
```
df.shape
```
Out[278]:
```
(5572, 2)
```

In [279]:
```
df = df.drop_duplicates(keep='first')
```

In [280]:
```
df.duplicated().sum()
```
Out[280]:
```
0
```

In [281]:
```
df.shape
```
Out[281]:
```
(5169, 2)
```

| Data Analysis

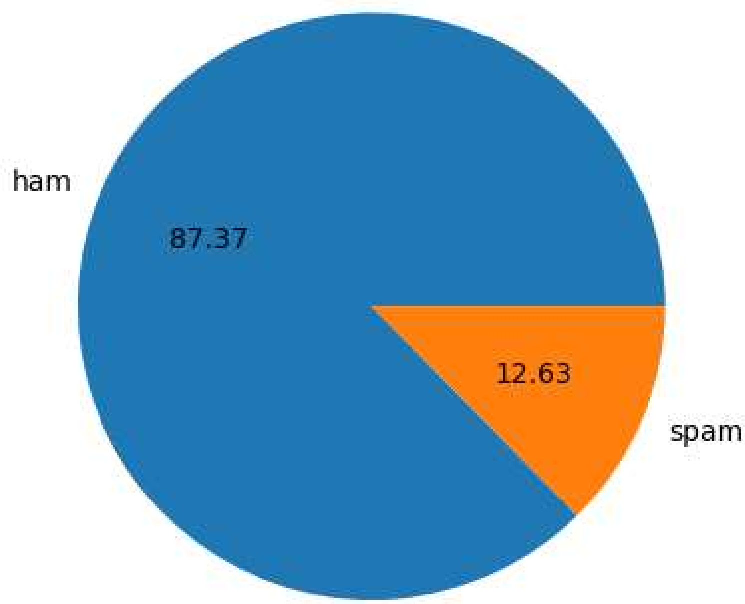
In [282]:
```
df.head()
```
Out[282]:

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

In [283]:
```
df['target'].value_counts()
```
Out[283]:
```
0    4516
1     653
Name: target, dtype: int64
```

- Now i'll graph this dataframe in form of a pie chart

In [284]:
```
import matplotlib.pyplot as plt
plt.pie(df['target'].value_counts(), labels=['ham','spam'],autopct="%0.2f")
plt.show()
```



-> we can see that we have a lot more 'non spam' data than 'spam' data, so when building the model we need to focus on the model precision a little more than the model accuracy

In [285]:
```
!pip install nltk
import nltk
```

```
Requirement already satisfied: nltk in e:\anaconda\lib\site-packages (3.8.1)
Requirement already satisfied: click in e:\anaconda\lib\site-packages (from nlt
k) (8.0.4)
Requirement already satisfied: joblib in e:\anaconda\lib\site-packages (from nlt
k) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in e:\anaconda\lib\site-packages
(from nltk) (2022.7.9)
Requirement already satisfied: tqdm in e:\anaconda\lib\site-packages (from nltk)
(4.65.0)
Requirement already satisfied: colorama in e:\anaconda\lib\site-packages (from c
lick->nltk) (0.4.6)
```

In [286]:
```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to C:\Users\Arindal
[nltk_data]     Char\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[286]: True

In [287]:
```
df['num_characters'] = df['text'].apply(len)
```

-> this stores the number of characters in every text in a variable 'num_characters'

In [288]:
```
df.head()
```

Out[288]:

	target	text	num_characters
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

In [289]:
```
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
```

-> this stores the number of words in each 'text' inside variable 'num_words'

In [290]:
```
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

-> this stores the number of sentences in each 'text' inside variable 'num_words'

In [291]:
```
df[['num_characters','num_words','num_sentences']].describe()
```

Out[291]:

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.455794	1.965564
std	58.236293	13.324758	1.448541
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	220.000000	38.000000

In [292]:
```
df.head()
```

Out[292]:

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

- Describing 'ham' messages/mails

In [293]:
```
df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
```

Out[293]:

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.123782	1.820195
std	56.358207	13.493970	1.383657
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	38.000000

- Describing 'spam' messages/mails

In [294]:
```
df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
```

Out[294]:

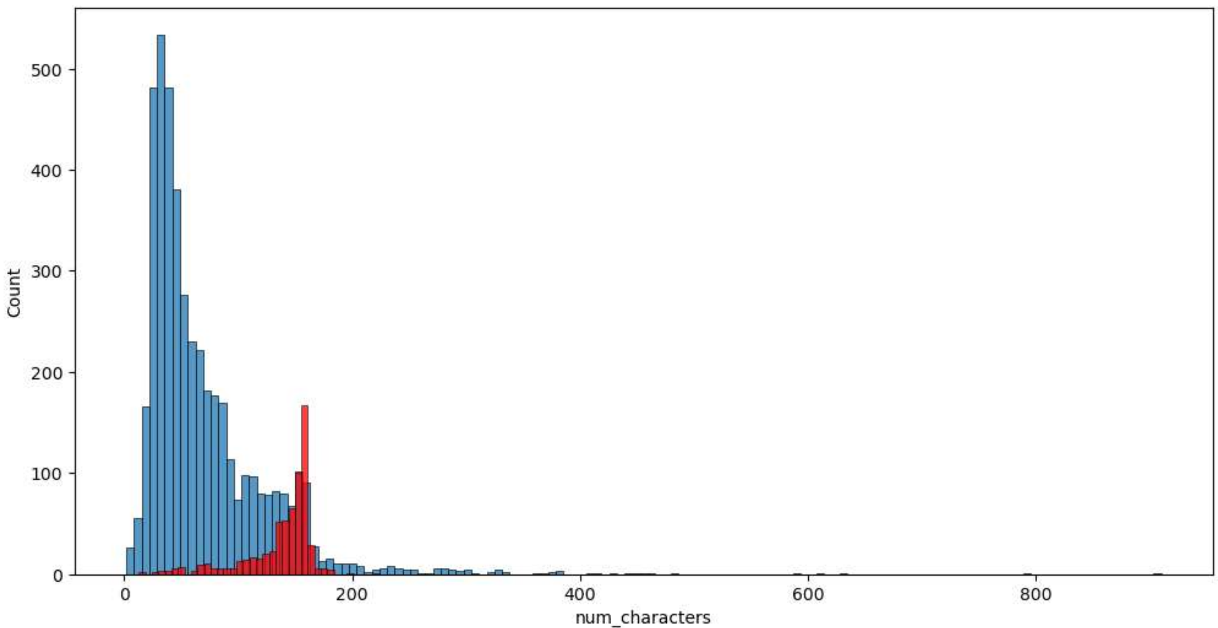
	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.970904
std	30.137753	7.008418	1.488425
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	9.000000

In [295]:
```
import seaborn as sns
```

- Lets plot these data in graphs

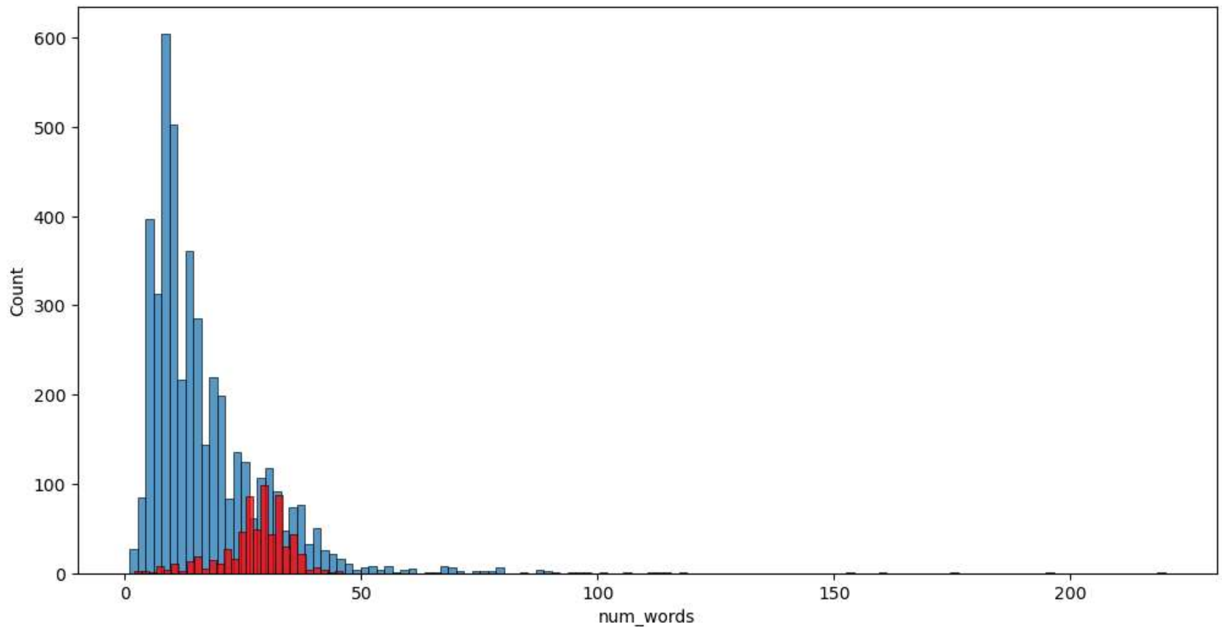
In [296]:
```
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_characters'])
sns.histplot(df[df['target'] == 1]['num_characters'],color='red')
```

Out[296]:
```
<Axes: xlabel='num_characters', ylabel='Count'>
```



In [297]:
```
plt.figure(figsize=(12,6))
sns.histplot(df[df['target'] == 0]['num_words'])
sns.histplot(df[df['target'] == 1]['num_words'],color='red')
```

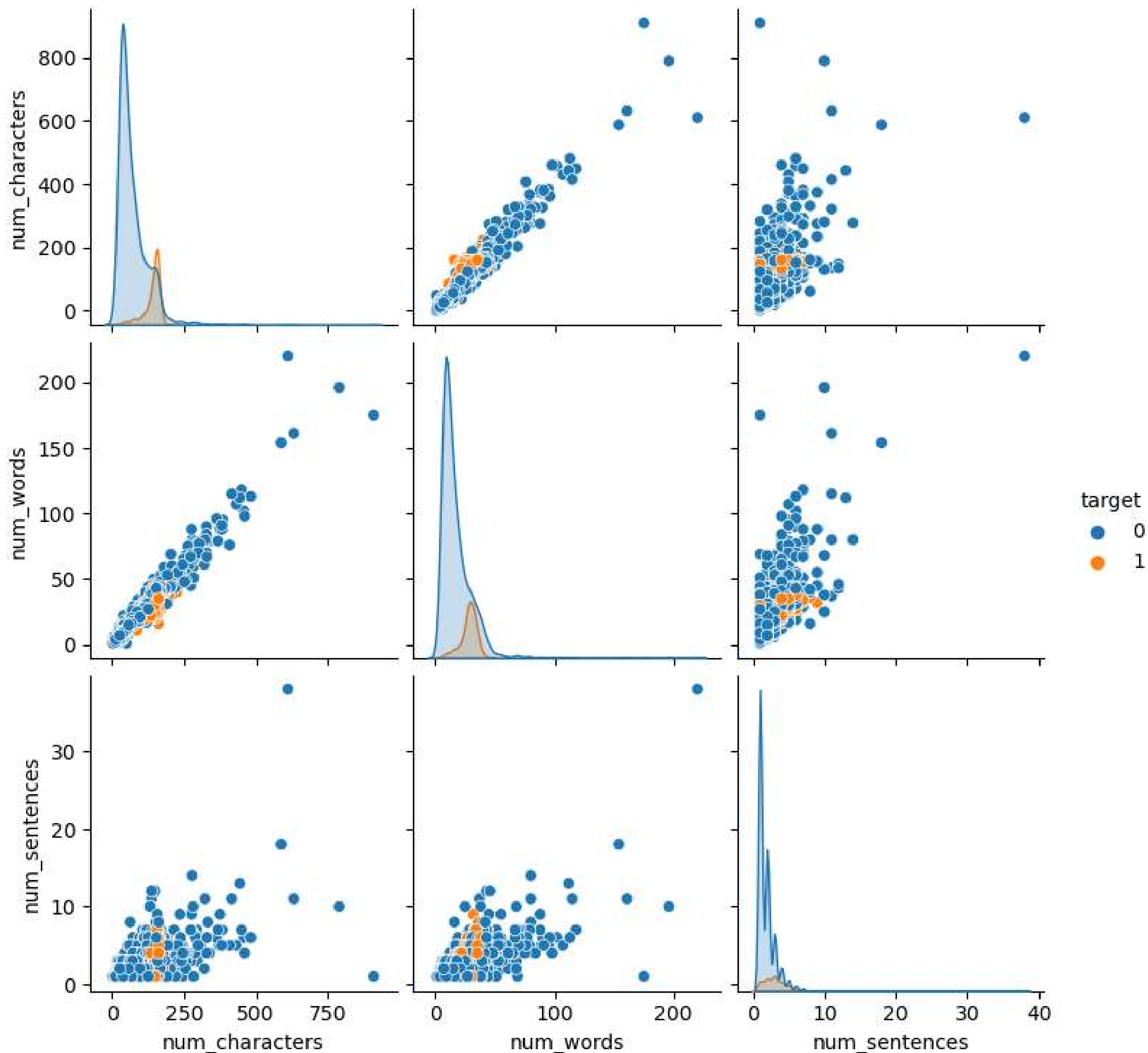
Out[297]: <Axes: xlabel='num_words', ylabel='Count'>



-> we can see that 'spam' messages are made up of more words or characters, compared to 'ham' texts

In [298]:
```
sns.pairplot(df,hue='target')
```

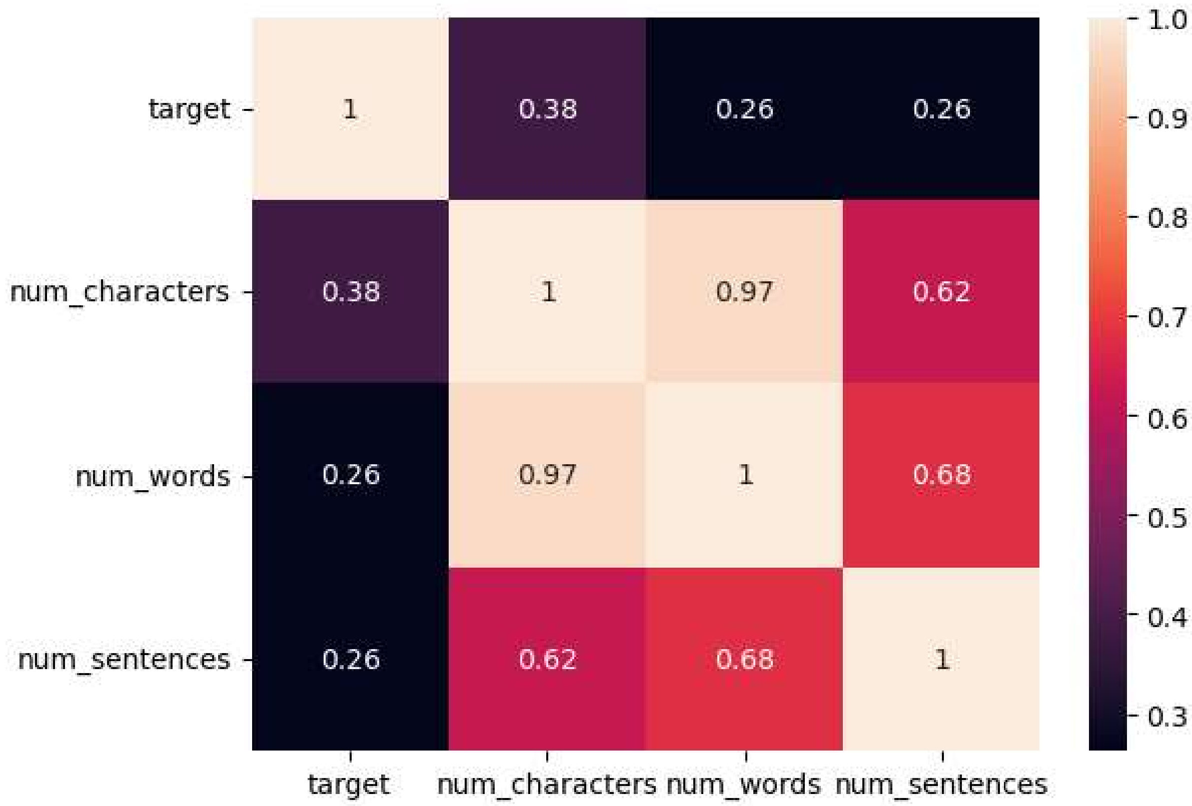
Out[298]: <seaborn.axisgrid.PairGrid at 0x25dadb8b150>



In [299]:
```
sns.heatmap(df.corr(),annot=True)
```

```
C:\Users\Arindal Char\AppData\Local\Temp\ipykernel_15052\4277794465.py:1: Future
Warning: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specify t
he value of numeric_only to silence this warning.
  sns.heatmap(df.corr(),annot=True)
```

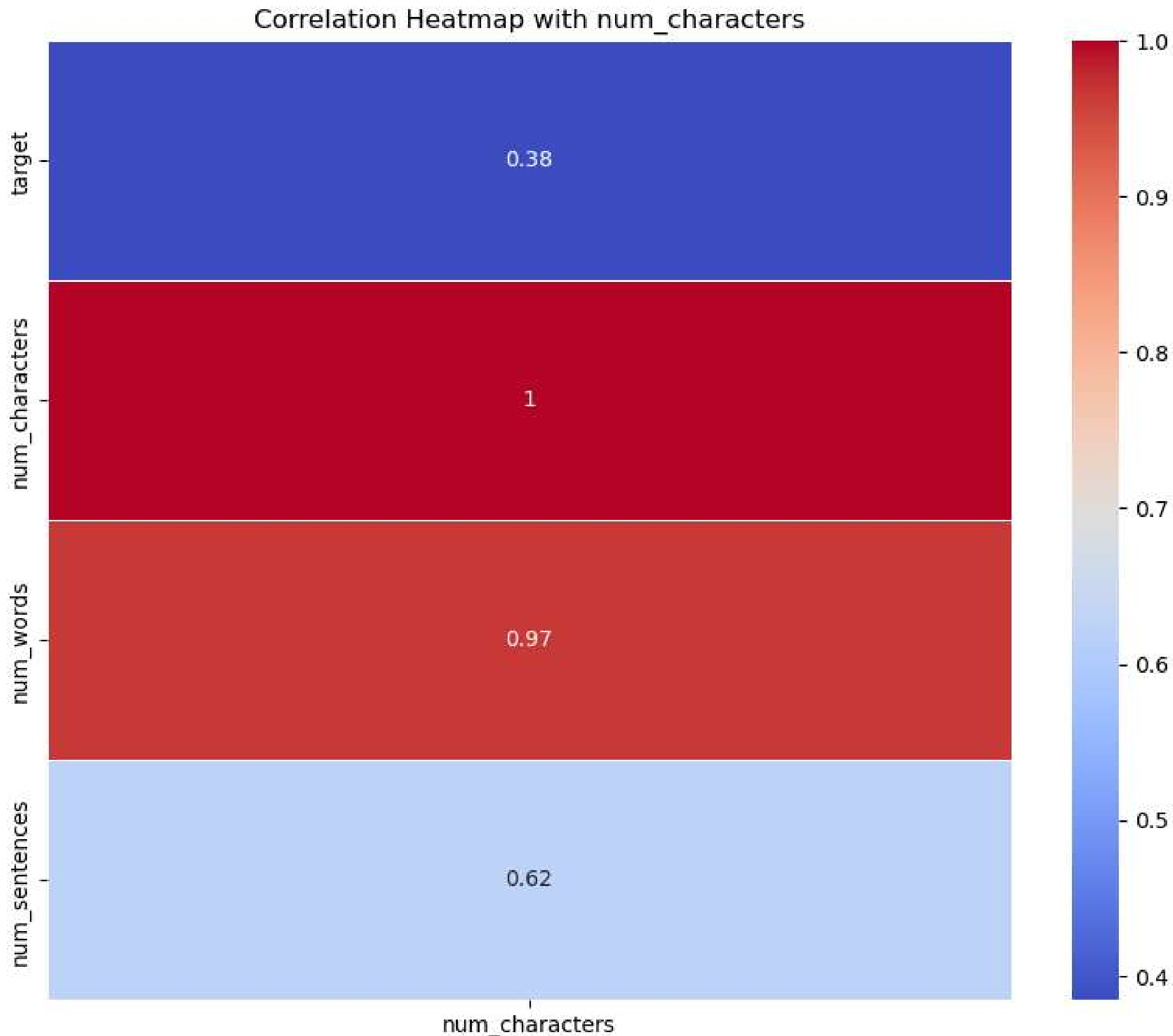
Out[299]: <Axes: >



-> this is a heat map of the correlation of our data, we can see that there is a high correlation between out different columns, so we should use only one valid column

In [300]:
```
correlation_matrix = df.corr(numeric_only=True)
correlation_with_num_characters = correlation_matrix['num_characters']

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_with_num_characters.to_frame(), annot=True, cmap='coolwarı
plt.title('Correlation Heatmap with num_characters')
plt.show()
```



| Data Preprocessing

In [301]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\Arindal
[nltk_data]     Char\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[301]: True

In [302]:
```
from nltk.corpus import stopwords
import string
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
```

In [303]:
```
stopwords.words('english')
```

Out[303]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
```

In [304]:
```
string.punctuation
```

Out[304]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [305]:
```
ps.stem('loving')
```

Out[305]:
```
'love'
```

In [306]:
```
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)
```

-> the function 'transform_text' does these following tasks:

- **Convert Text to Lowercase:** It first converts the input text to lowercase to ensure consistent case handling.
- **Tokenization:** It uses nltk.word_tokenize to split the text into individual words or tokens. Tokenization is the process of breaking text into smaller units, typically words or punctuation marks.
- **Remove Non-Alphanumeric Characters:** It iterates through the tokens and keeps only those that are alphanumeric (letters and numbers) by checking i.isalnum(). This step removes special characters and symbols.
- **Stopword Removal:** It further filters the tokens by removing common English stopwords using stopwords.words('english'). Stopwords are words that are often removed from text because they are considered to be of little value in many NLP tasks.
- **Stemming:** It applies stemming to the remaining tokens using ps.stem(i). Stemming reduces words to their base or root form. For example, it converts words like "running" and "ran" to "run."
- **Join Tokens:** Finally, it joins the processed tokens back together into a single string and returns the resulting string.

In [307]:
```
df['text'][20]
```

Out[307]:
```
'Is that seriously how you spell his name?'
```

In [308]:
```
transform_text("Is that seriously how you spell his name?")
```

Out[308]:
```
'serious spell name'
```

In [309]:
```
df['transformed_text'] = df['text'].apply(transform_text)
```

In [310]:
```
df.head()
```

Out[310]:

	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1	nah think goe usf live around though

- Let's analyse the most occuring words in spam texts

In [311]:
```
spam_msg = []
for msg in df[df['target']==1]['transformed_text'].tolist():
    for word in msg.split():
        spam_msg.append(word)
```

In [312]:
```
spam_msg
```

```
 'txt',
 'rate',
 'c',
 'appli',
 '08452810075over18',
 'freemsg',
 'hey',
 'darl',
 '3',
 'week',
 'word',
 'back',
 'like',
 'fun',
 'still',
 'tb',
 'ok',
 'xxx',
 'std',
```

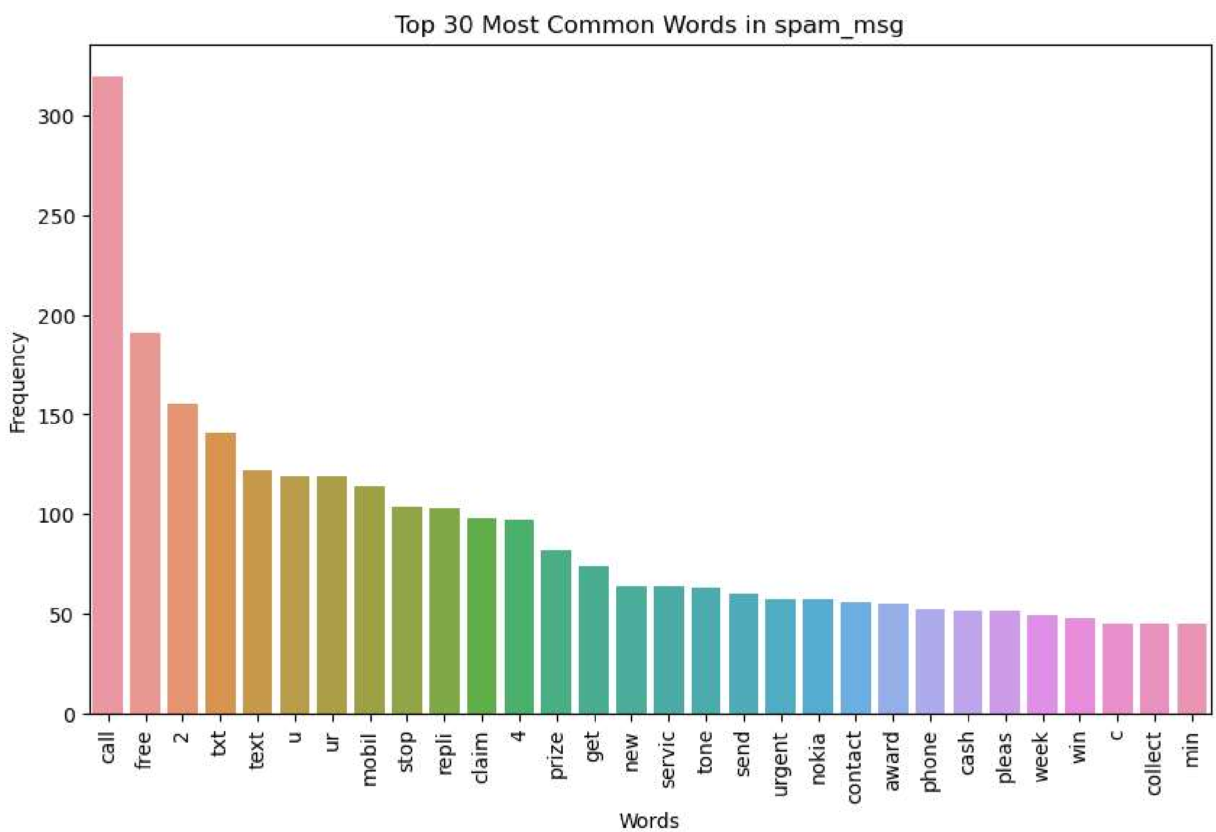
In [313]:
```
print('length: ', len(spam_msg))
```

```
length:  9939
```

In [314]:
```
from collections import Counter
counter = Counter(spam_msg)
most_common_items = counter.most_common(30)
```

In [315]:
```
commondf = pd.DataFrame(most_common_items)
```

In [316]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x=commondf[0], y=commondf[1])
plt.xticks(rotation='vertical')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 30 Most Common Words in spam_msg')
plt.show()
```



| Model Building

- Now we need to vectorize our data. We can do so using CountVectorizer or TfidVectorizer

In [327]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

In [328]:
```
# X = cv.fit_transform(df['transformed_text']).toarray()
```

In [329]:
```
X = tfidf.fit_transform(df['transformed_text']).toarray()
```

-> i decided to go with Tfid because it gave better results than Cv

In [361]:
```
print(X)
X.shape
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Out[361]:
```
(5169, 3000)
```

In [331]:
```
y = df['target'].values
```

In [332]:
```
from sklearn.model_selection import train_test_split
```

In [333]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2
```

-i want to compare different classification algorithms to see which performs the best with our data

i'm using GaussianNB, MultinomialNB and BernoulliNB for now

In [334]:
```
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

In [335]:
```
gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
```

In [336]:
```
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))
```

```
0.8694390715667312
[[788 108]
 [ 27 111]]
0.5068493150684932
```

In [337]:
```
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))
```

```
0.9709864603481625
[[896   0]
 [ 30 108]]
1.0
```

In [338]:
```
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))
```

```
0.9835589941972921
[[895   1]
 [ 16 122]]
0.991869918699187
```

-> we can see that MultinomialNB and BernoulliNB both give really good results both in terms of accuracy and precision

---

- let's try some other classification algorithms and test their accuracy and precision

In [343]:
```
pip install xgboost
```

```
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  70.9/70.9 MB 3.9 MB/s eta 0:00:01
```

In [344]:
```
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 [345]:
```
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 [346]:
```
clfs = {
    'SVC' : svc,
    'KN' : knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT':gbdt,
    'xgb':xgb
}
```

In [347]:
```
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 [348]:
```
train_classifier(svc,X_train,y_train,X_test,y_test)
```

Out[348]:
```
(0.9758220502901354, 0.9747899159663865)
```

In [349]:
```
accuracy_scores = []
precision_scores = []

for name,clf in clfs.items():

    current_accuracy,current_precision = train_classifier(clf, X_train,y_train,X_

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

    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For  SVC
Accuracy -  0.9758220502901354
Precision -  0.9747899159663865
For  KN
Accuracy -  0.9052224371373307
Precision -  1.0
For  NB
Accuracy -  0.9709864603481625
Precision -  1.0
For  DT
Accuracy -  0.9313346228239845
Precision -  0.8252427184466019
For  LR
Accuracy -  0.9584139264990329
Precision -  0.9702970297029703
For  RF
Accuracy -  0.9758220502901354
Precision -  0.9829059829059829
For  AdaBoost
Accuracy -  0.960348162475822
Precision -  0.9292035398230089
For  BgC
Accuracy -  0.9584139264990329
Precision -  0.8682170542635659
For  ETC
Accuracy -  0.9748549323017408
Precision -  0.9745762711864406
For  GBDT
Accuracy -  0.9468085106382979
Precision -  0.9191919191919192
For  xgb
Accuracy -  0.9671179883945842
Precision -  0.9333333333333333
```

In [350]:
```
performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores
```

In [351]:
```
performance_df
```

Out[351]:

	Algorithm	Accuracy	Precision
1	KN	0.905222	1.000000
2	NB	0.970986	1.000000
5	RF	0.975822	0.982906
0	SVC	0.975822	0.974790
8	ETC	0.974855	0.974576
4	LR	0.958414	0.970297
10	xgb	0.967118	0.933333
6	AdaBoost	0.960348	0.929204
9	GBDT	0.946809	0.919192
7	BgC	0.958414	0.868217
3	DT	0.931335	0.825243

-> from this table we can conclude that our best performing algorithms are:

- Naive Bayer's
- Random Forest
- Support Vector Classifier
- Extra Trees Classifier

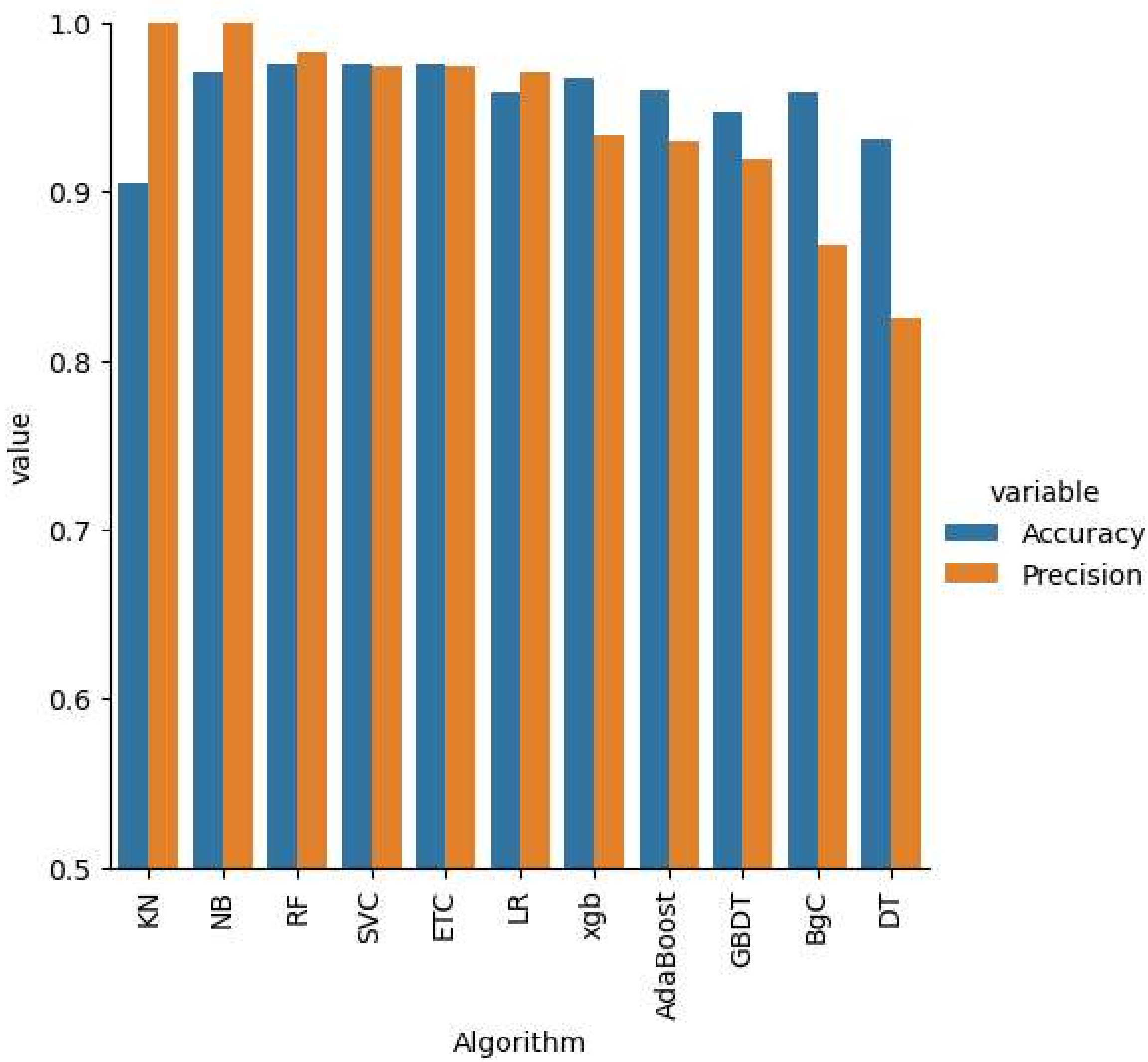
In [352]:
```
performance_df1 = pd.melt(performance_df, id_vars = "Algorithm")
```

In [353]:
```
performance_df1
```

Out[353]:

	Algorithm	variable	value
0	KN	Accuracy	0.905222
1	NB	Accuracy	0.970986
2	RF	Accuracy	0.975822
3	SVC	Accuracy	0.975822
4	ETC	Accuracy	0.974855
5	LR	Accuracy	0.958414
6	xgb	Accuracy	0.967118
7	AdaBoost	Accuracy	0.960348
8	GBDT	Accuracy	0.946809
9	BgC	Accuracy	0.958414
10	DT	Accuracy	0.931335
11	KN	Precision	1.000000
12	NB	Precision	1.000000
13	RF	Precision	0.982906
14	SVC	Precision	0.974790
15	ETC	Precision	0.974576
16	LR	Precision	0.970297
17	xgb	Precision	0.933333
18	AdaBoost	Precision	0.929204
19	GBDT	Precision	0.919192
20	BgC	Precision	0.868217
21	DT	Precision	0.825243

In [354]:
```
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 [355]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accuracy_s
```

In [356]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':accuracy_score
```

In [357]:
```
new_df = performance_df.merge(temp_df,on='Algorithm')
```

In [358]:
```
new_df_scaled = new_df.merge(temp_df,on='Algorithm')
```

In [359]:
```
temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accuracy_sco
```

In [360]:
```
new_df_scaled.merge(temp_df,on='Algorithm')
```

Out[360]:

	Algorithm	Accuracy	Precision	Accuracy_scaling_x	Precision_scaling_x	Accuracy_scaling_y	Pr
0	KN	0.905222	1.000000	0.905222	1.000000	0.905222	
1	NB	0.970986	1.000000	0.970986	1.000000	0.970986	
2	RF	0.975822	0.982906	0.975822	0.982906	0.975822	
3	SVC	0.975822	0.974790	0.975822	0.974790	0.975822	
4	ETC	0.974855	0.974576	0.974855	0.974576	0.974855	
5	LR	0.958414	0.970297	0.958414	0.970297	0.958414	
6	xgb	0.967118	0.933333	0.967118	0.933333	0.967118	
7	AdaBoost	0.960348	0.929204	0.960348	0.929204	0.960348	
8	GBDT	0.946809	0.919192	0.946809	0.919192	0.946809	
9	BgC	0.958414	0.868217	0.958414	0.868217	0.958414	
10	DT	0.931335	0.825243	0.931335	0.825243	0.931335	

| Voting Classifier

> A Voting Classifier is an ensemble machine learning model in which multiple base models (classifiers) are trained on a dataset, and their predictions are combined to make a final prediction.

In [363]:
```
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 [364]:
```
voting = VotingClassifier(estimators=[('svm', svc), ('nb', mnb), ('et', etc)],vot
```

In [365]:
```
voting.fit(X_train,y_train)
```

Out[365]:
```
VotingClassifier

VotingClassifier(estimators=[('svm',
                              SVC(gamma=1.0, kernel='sigmoid',
                                  probability=True)),
                             ('nb', MultinomialNB()),
                             ('et',
                              ExtraTreesClassifier(n_estimators=50,
                                                   random_state=2))],
                 voting='soft')

svm                    nb              et
SVC                    MultinomialNB   ExtraTreesClassifier
SVC(gamma=1.0, kernel='sigmoid', probability=True)
MultinomialNB()
ExtraTreesClassifier(n_estimators=50, random_state=2)
```

In [366]:
```
y_pred = voting.predict(X_test)
print("Accuracy",accuracy_score(y_test,y_pred))
print("Precision",precision_score(y_test,y_pred))
```

```
Accuracy 0.9816247582205029
Precision 0.9917355371900827
```

-> using a Voting Classifier gave me good results but nothing special, and there wasn't much difference
-> i also tried 'stacking' which is used to give a particular algorithm more dominance, but that too didn't give me much of a diiference

---

| Model Serialization

-we'll save the TF-IDF vectorizer and Multinomial Naive Bayes model as pickled files, which can be loaded and reused for future predictions

In [367]:
```
import pickle
pickle.dump(tfidf,open('vectorizer.pkl','wb'))
pickle.dump(mnb,open('model.pkl','wb'))
```

In [ ]:
```

```