Data

```
# Importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from sklearn import feature_extraction, model_selection, naive_bayes, metrics, svm
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

# Libraries for text processing
import re, nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#Data: https://drive.google.com/file/d/1UEc9IY2HgIAYOsm4qpXdqUO5a1ZyeyTV/view?usp=s
id = "1UEc9IY2HgIAYOsm4qpXdqUO5a1ZyeyTV"
path = "https://docs.google.com/uc?export=download&id=" + id
print(path)
```

```
https://docs.google.com/uc?export=download&id=1UEc9IY2HgIAYOsm4qpXdqUO5a1ZyeyT
```

```
!wget "https://docs.google.com/uc?export=download&id=1UEc9IY2HgIAYOsm4qpXdqUO5a1Zye
```

```
--2022-05-27 14:32:10--  https://docs.google.com/uc?export=download&id=1UEc9IY
Resolving docs.google.com (docs.google.com)... 142.250.157.101, 142.250.157.13
Connecting to docs.google.com (docs.google.com)|142.250.157.101|:443... connec
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0o-64-docs.googleusercontent.com/docs/securesc/ha0ro937c
Warning: wildcards not supported in HTTP.
--2022-05-27 14:32:11--  https://doc-0o-64-docs.googleusercontent.com/docs/sec
Resolving doc-0o-64-docs.googleusercontent.com (doc-0o-64-docs.googleuserconte
Connecting to doc-0o-64-docs.googleusercontent.com (doc-0o-64-docs.googleuserc
HTTP request sent, awaiting response... 200 OK
Length: 483640 (472K) [text/csv]
Saving to: ‘spam_clean.csv’
```

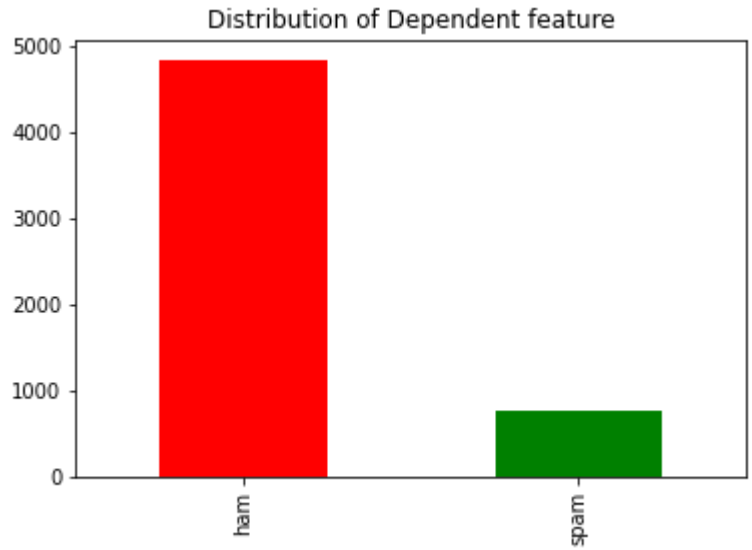
```
spam_clean.csv      100%[===================>] 472.30K  --.-KB/s    in 0.004s

2022-05-27 14:32:11 (121 MB/s) - ‘spam_clean.csv’ saved [483640/483640]
```

```
df = pd.read_csv('./spam_clean.csv', encoding='latin-1')
df.head()
```

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

```
freq = pd.value_counts(df["type"], sort= True)
freq.plot(kind= 'bar', color= ["red", "green"])
plt.title('Distribution of Dependent feature')
plt.show()
```



Text Cleaning & Preprocessing

```
def clean_tokenized_sentence(s):
    """Performs basic cleaning of a tokenized sentence"""
    cleaned_s = ""  # Create empty string to store processed sentence.
    words = nltk.word_tokenize(s)
    for word in words:
        # Convert to lowercase #
        c_word = word.lower()
        # Remove punctuations #
```

```
        c_word = re.sub(r'[^\w\s]', '', c_word)
        # Remove stopwords #
        if c_word != '' and c_word not in stopwords.words('english'):
            cleaned_s = cleaned_s + " " + c_word    # Append processed words to new
    return(cleaned_s.strip())


df["cleaned_message"] = df["message"].apply(clean_tokenized_sentence)
df.head(10)
```

	type	message	cleaned_message
0	ham	Go until jurong point, crazy.. Available only ...	go jurong point crazy available bugis n great ...
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry 2 wkly comp win fa cup final tkts 2...
3	ham	U dun say so early hor... U c already then say...	u dun say early hor u c already say
4	ham	Nah I don't think he goes to usf, he lives aro...	nah nt think goes usf lives around though
5	spam	FreeMsg Hey there darling it's been 3 week's n...	freemsg hey darling 3 week word back like fun ...
6	ham	Even my brother is not like to speak with me. ...	even brother like speak treat like aids patent

Most common words in Spam and Ham

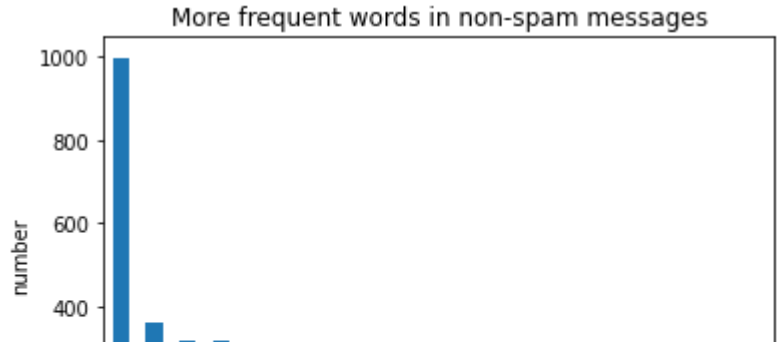
```
Counter(" ".join(df[df['type']=='ham']["cleaned_message"]).split())
```

```
             'granted': 1,
             'fulfil': 1,
             'wonderful': 15,
             'blessing': 2,
             'times': 25,
             'date': 11,
             'sunday': 8,
             'oh': 113,
             'watching': 34,
             'eh': 12,
             'remember': 32,
             '2': 309,
             'spell': 3,
             'name': 36,
             'yes': 75,
             'v': 47,
             'naughty': 6,
             'make': 88,
             'wet': 3,
             'fine': 48,
             'thatåõs': 5
```

```
 'chatdos': 5,
 'way': 101,
 'feel': 62,
 'gota': 1,
 'b': 56,
 'seriously': 9,
 'iûm': 5,
 'going': 168,
 'try': 42,
 'months': 7,
 'ha': 16,
 'ì_': 120,
 'pay': 30,
 'first': 56,
 'da': 143,
 'stock': 5,
 'comin': 11,
 'aft': 19,
 'finish': 42,
 'lunch': 46,
 'str': 3,
 'lor': 162,
 'ard': 22,
 '3': 53,
 'smth': 16,
 'ur': 241,
 'fffffffff': 1,
 'alright': 23,
 'meet': 74,
 'sooner': 4,
 'forced': 1,
 'eat': 38,
 'slice': 2,
 'really': 85,
 'hungry': 12,
 'tho': 18,
 'sucks': 7,
 'mark': 8,
 'getting': 46,
```

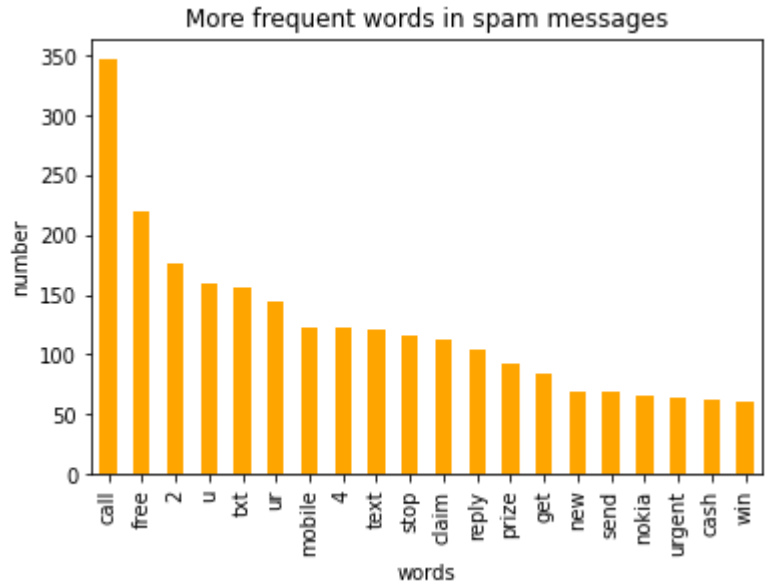
```
counter_ham = Counter(" ".join(df[df['type']=='ham']["cleaned_message"]).split()).m
df_ham = pd.DataFrame.from_dict(counter_ham)
df_ham = df_ham.rename(columns={0:"words in non-spam", 1:"count"})

df_ham.plot.bar(legend = False)
y_pos = np.arange(len(df_ham["words in non-spam"]))
plt.xticks(y_pos, df_ham["words in non-spam"])
plt.title('More frequent words in non-spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```




```
counter_spam = Counter(" ".join(df[df['type']=='spam']["cleaned_message"]).split())
df_spam = pd.DataFrame.from_dict(counter_spam)
df_spam = df_spam.rename(columns={0:"words in spam", 1:"count_"})

df_spam.plot.bar(legend = False, color = 'orange')
y_pos = np.arange(len(df_spam["words in spam"]))
plt.xticks(y_pos, df_spam["words in spam"])
plt.title('More frequent words in spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```



Generate Datasets

```
df["type"] = df["type"].map({'spam':1,'ham':0})
```

```
df_X_train, df_X_test, y_train, y_test = train_test_split(df['cleaned_message'], df
print([np.shape(df_X_train), np.shape(df_X_test)])
```

```
[(4179,), (1393,)]
```

```
#Count Vectorizer
f = feature_extraction.text.CountVectorizer()
```

```
X_train = f.fit_transform(df_X_train)
X_test = f.transform(df_X_test)
```

```
# Standard Scaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-ae0768e59d91> in <module>()
      1 # Standard Scaler
      2 scaler = StandardScaler()
----> 3 X_train = scaler.fit_transform(X_train)
      4 X_test = scaler.transform(X_test)

                                 2 frames
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_data.py in parti
    869             if self.with_mean:
    870                 raise ValueError(
--> 871                     "Cannot center sparse matrices: pass `with_mean=Fa
    872                     "instead. See docstring for motivation and alterna
    873                 )

ValueError: Cannot center sparse matrices: pass `with_mean=False` instead. See
```

SEARCH STACK OVERFLOW

```
# Standard Scaler
scaler = StandardScaler(with_mean=False) # Problems with dense matrix
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test.todense())
```

```
print([np.shape(X_train), np.shape(X_test)])
```

```
[(4179, 7692), (1393, 7692)]
```

```
type(X_train)
```

```
scipy.sparse.csr.csr_matrix
```

Linear SVM Models

[ ] ↳ *2 cells hidden*

RBF SVM

[ ] ↳ *1 cell hidden*

Multinomial NB

```
df_X_train, df_X_test, y_train, y_test = train_test_split(df['cleaned_message'], df
print([np.shape(df_X_train), np.shape(df_X_test)])

#Count Vectorizer
f = feature_extraction.text.CountVectorizer()

X_train = f.fit_transform(df_X_train)
X_test = f.transform(df_X_test)

# No need of scaling


# Multinomial NB

from sklearn.model_selection import GridSearchCV

params = {
        'alpha':[0.01, 0.1, 1, 10]
        }
mnb = naive_bayes.MultinomialNB()
clf = GridSearchCV(mnb, params, scoring = "f1", cv=3)

clf.fit(X_train, y_train)

res = clf.cv_results_

for i in range(len(res["params"])):
  print(f"Parameters:{res['params'][i]} Mean_score: {res['mean_test_score'][i]} Ran
```

```
[(4179,), (1393,)]
Parameters:{'alpha': 0.01} Mean_score: 0.8921625905385584 Rank: 2
Parameters:{'alpha': 0.1} Mean_score: 0.8883633779156167 Rank: 3
Parameters:{'alpha': 1} Mean_score: 0.9006541826507674 Rank: 1
Parameters:{'alpha': 10} Mean_score: 0.8580328947757173 Rank: 4
```

✓ 0s completed at 20:14