1. Load and simplify the dataset

Our SMS text messages dataset has 5 columns if you read it in pandas: v1 (containing the class labels ham/spam for each text message), v2 (containing the text messages themselves), and three Unnamed columns which have no use. We'll rename the v1 and v2 columns to class_label and message respectively while getting rid of the rest of the columns.

```
import pandas as pd
df =
pd.read_csv(r'spam.csv',encoding='ISO-
8859-1')
df.rename(columns = {'v1':'class_label',
'v2':'message'}, inplace = True)
df.drop(['Unnamed: 2', 'Unnamed: 3',
'Unnamed: 4'], axis = 1, inplace = True)


df
```

	class_label	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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

Check out the fact that '5572 rows x 2 columns' means that our dataset has 5572 text messages!

2. Explore the dataset: Bar Chart

It's a good idea to carry out some Exploratory Data Analysis (EDA) in a classification problem to visualize, get some information out of, or find any issues with your data before you how many spam/ham messages we have and create a bar chart for it.

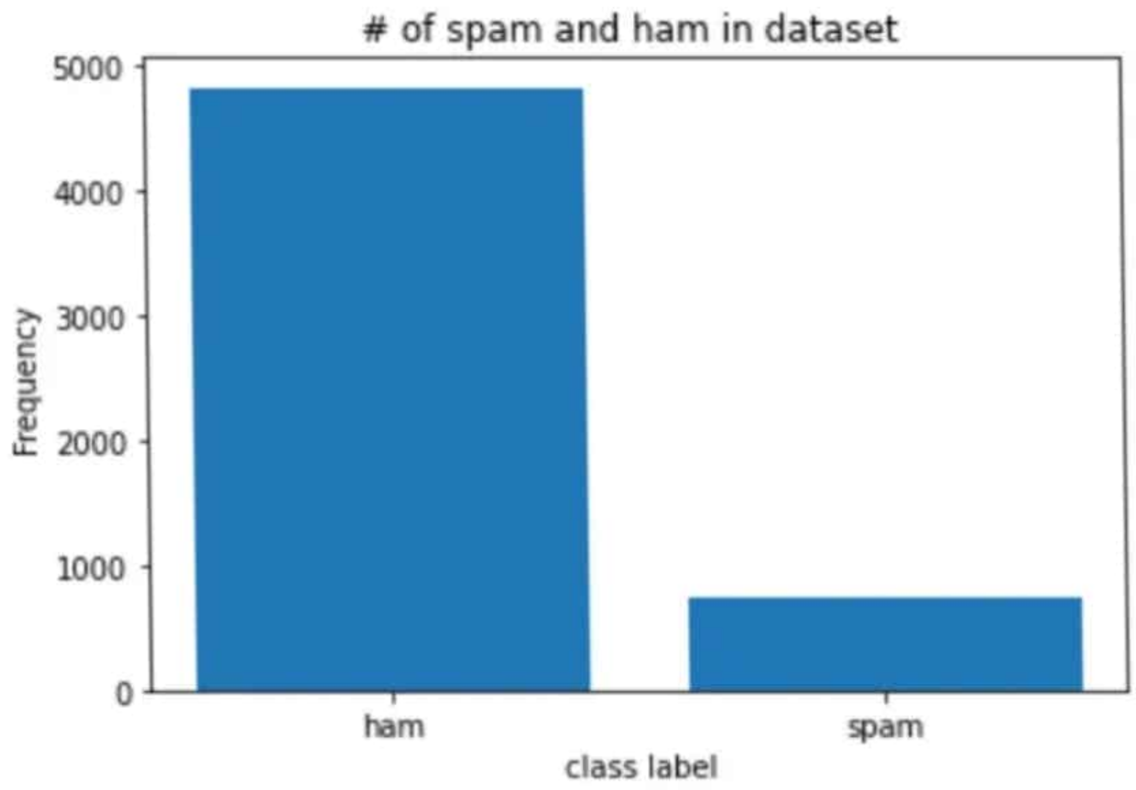
```
#exploring the dataset

df['class_label'].value_counts()
```

```
ham      4825
spam      747
Name: class_label, dtype: int64
```

Our dataset has 4825 ham messages and 747 spam messages. This is an imbalanced dataset; the number of ham messages is much higher than those of spam! This can potentially cause our model to be biased. To fix this, we could resample our data to get an equal number of spam/ham messages.

To generate our bar chart, we use NumPy and pyplot from Matplotlib.

To generate our bar chart, we use NumPy and pyplot from Matplotlib.



3. Explore the dataset: Word Clouds

For my project, I generated word clouds of the most frequently occurring words in my spam messages.

```
df_spam = df[df.class_label=='spam']


df_spam
```

	class_label	message
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...
11	spam	SIX chances to win CASH! From 100 to 20,000 po...
...	...	...
5537	spam	Want explicit SEX in 30 secs? Ring 02073162414...
5540	spam	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...
5547	spam	Had your contract mobile 11 Mnths? Latest Moto...
5566	spam	REMINDER FROM O2: To get 2.50 pounds free call...
5567	spam	This is the 2nd time we have tried 2 contact u...

Next, we'll convert our DataFrame to a list, where every element of that list will be a spam message. Then, we'll join each element of our list into one big string of spam messages. The lowercase form of that string is the required format needed for our word cloud creation.

our list into one big string of spam messages. The lowercase form of that string is the required format needed for our word cloud creation.

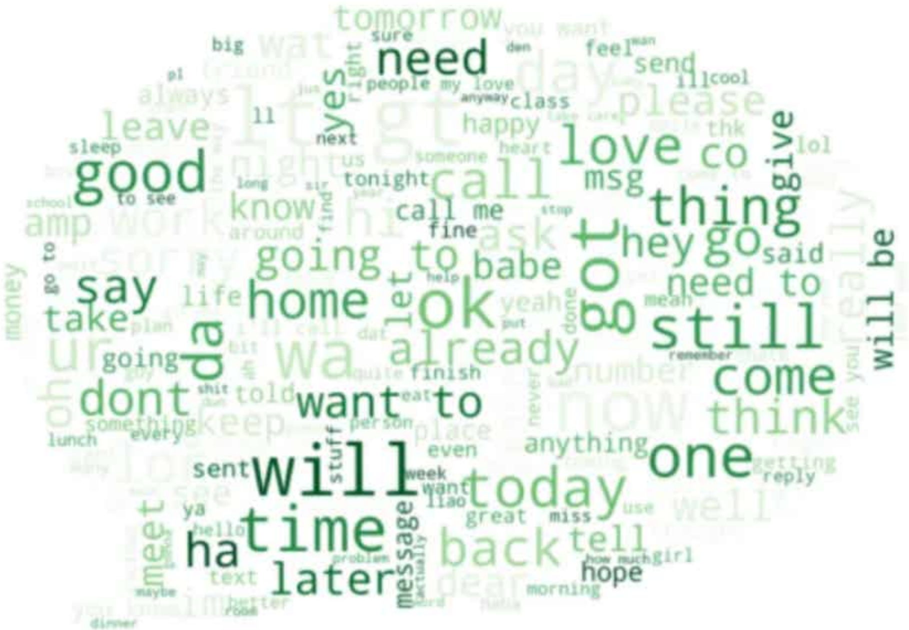
```
spam_list= df_spam['message'].tolist()


filtered_spam = filtered_spam.lower()
```

Finally, we'll import the relevant libraries and pass in our string as a parameter:

After displaying it:

Pretty cool, huh? The most common words in spam messages in our dataset are 'free,' 'call now,' 'to claim,' 'have won,' etc.

For this word cloud, we needed the Pillow library only because I've used masking to create that nice speech bubble shape. If you want it in square form, omit the mask parameter.



4. Handle imbalanced datasets

To handle imbalanced data, you have a variety of options. I got a pretty good f-measure in my project even with unsampled data, but if you want to resample, see this.

5. Split the dataset

First, let's convert our class labels from string to numeric form:

```
df['class_label'] =
df['class_label'].apply(lambda x: 1 if x
== 'spam' else 0)
```

In Machine Learning, we usually split our data into two subsets — train and test. We feed the train set along with the known output values for it (in this case, 0 or 1 corresponding to spam or ham) to our model so that it learns the patterns in our data. Then we use the test set to get the model's predicted labels on this subset. Let's see how to split our data.

First, we import the relevant module from the sklearn library:

```
from sklearn.model_selection import
train_test_split
```

And then we make the split:

```
x_train, x_test, y_train, y_test
= train_test_split(df['message'],
df['class_label'], test_size = 0.3,
random_state = 0)
```

Let's now see how many messages we have for our test and train subsets:

```
print('rows in test set: ' +
str(x_test.shape))
print('rows in train set: ' +
str(x_train.shape))
```

```
rows in test set: (1672,)
rows in train set: (3900,)

pandas.core.series.Series
```

So we have 1672 messages for testing, and 3900 messages for training!

6. Apply Tf-IDF Vectorizer for feature extraction

Our Naïve Bayes model requires data to be in either Tf-IDF vectors or word vector count. The latter is achieved using Count Vectorizer, but we'll obtain the former through using Tf-IDF Vectorizer.

```
lst = x_train.tolist()
vectorizer = TfidfVectorizer(
input= lst ,  # input is the actual text
lowercase=True,      # convert to
lowercase before tokenizing
stop_words='english' # remove stop words
)



features_train_transformed =
vectorizer.fit_transform(list) #gives tf
idf vector for x_train
features_test_transformed  =
vectorizer.transform(x_test) #gives tf
idf vector for x_test
```

7. Train our Naive Bayes Model

We fit our Naïve Bayes model, aka MultinomialNB, to our Tf-IDF vector version of x_train, and the true output labels stored in y_train.

```
from sklearn.naive_bayes import
MultinomialNB
# train the model
classifier = MultinomialNB()
classifier.fit(feature
s_train_transformed,
y_train)
```

```
labels = classifier.predict(features_test
_transformed)
from sklearn.metrics import f1_score
from sklearn.metrics import
confusion_matrix
from sklearn.metrics import
accuracy_score
from sklearn.metrics import
classification_report



actual = y_test.tolist()
predicted = labels
results = confusion_matrix(actual,
predicted)
print('Confusion Matrix :')
print(results)
print ('Accuracy
Score :',accuracy_score(actual,
predicted))
print ('Report : ')
print (classification_report(actual,
predicted) )
score_2 = f1_score(actual, predicted,
average = 'binary')
print('F-Measure: %.3f' % score_2)
```

```
Confusion Matrix :
[[1434    0]
 [  61  177]]
Accuracy Score : 0.9635167464114832
Report :
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1434
           1       1.00      0.74      0.85       238

    accuracy                           0.96      1672
   macro avg       0.98      0.87      0.92      1672
weighted avg       0.97      0.96      0.96      1672

F-Measure: 0.853
```

We have an f-measure score of 0.853, and our confusion matrix shows that our model is making only 61 incorrect classifications. Looks pretty good to me 😊

10. Heatmap for our Confusion Matrix (Optional)

You can create a heatmap using the seaborn library to visualize your confusion matrix. The code below does just that.