Preprocessing

In this lab, we will be exploring how to preprocess tweets for sentiment analysis. We use the NLTK package to perform a preprocessing pipeline for Twitter datasets.

In [1]:

```
import nltk                                # Python library for NLP
from nltk.corpus import twitter_samples    # sample Twitter dataset from NLTK
import matplotlib.pyplot as plt            # library for visualization
import random                              # pseudo-random number generator
```

About the Twitter dataset

The dataset contains 5000 positive tweets and 5000 negative tweets exactly. (Not a Real World Scenario !!!)

In [2]:

```
#downloads sample twitter dataset.
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Unzipping corpora/twitter_samples.zip.
```

Out[2]:

```
True
```

We can load the text fields of the positive and negative tweets by using the module's `strings()` method like this:

In [3]:

```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

In [4]:

```
print('Number of positive tweets: ', len(all_positive_tweets))
print('Number of negative tweets: ', len(all_negative_tweets))

print('\nThe type of all_positive_tweets is: ', type(all_positive_tweets))
print('The type of a tweet entry is: ', type(all_negative_tweets[0]))
```

```
Number of positive tweets:  5000
Number of negative tweets:  5000

The type of all_positive_tweets is:  <class 'list'>
The type of a tweet entry is:  <class 'str'>
```

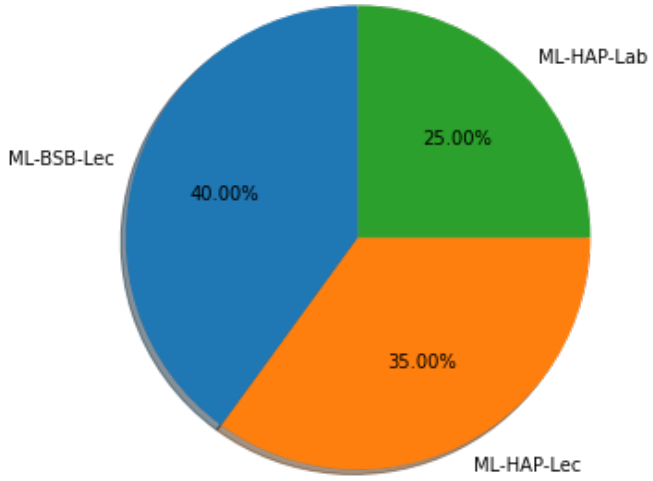
We can see that the data is stored in a list and individual tweets are stored as strings.

Let's have more visually appealing report by using Matplotlib's pyplot library.

In [18]:

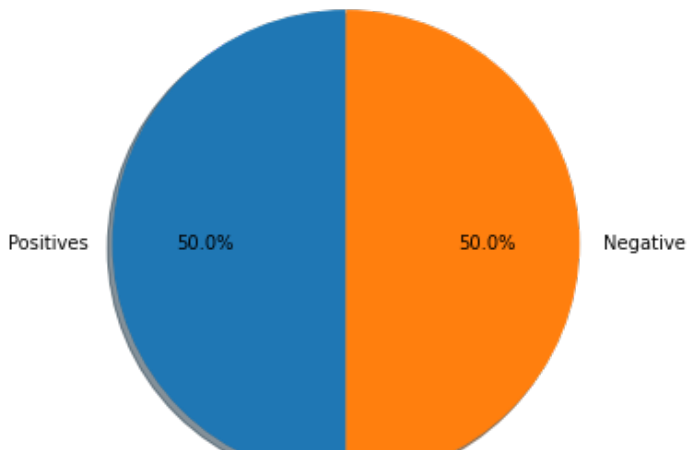
```
# Declare a figure with a custom size
fig = plt.figure(figsize=(5, 5))

# labels for the classes
labels = 'ML-BSB-Lec', 'ML-HAP-Lec','ML-HAP-Lab'
```

```
# Sizes for each slide
sizes = [40, 35, 25]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise:
plt.pie(sizes, labels=labels, autopct='%.2f%%',
        shadow=True, startangle=90)

#autopct enables you to display the percent value using Python string formatting.
#For example, if autopct='%.2f', then for each pie wedge, the format string is '%.2f' and



# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show()
```



In [19]:

```
# Declare a figure with a custom size
fig = plt.figure(figsize=(5, 5))

# labels for the two classes
labels = 'Positives', 'Negative'

# Sizes for each slide
sizes = [len(all_positive_tweets), len(all_negative_tweets)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise:
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show()
```



Looking at raw texts

In [22]:

```
# print positive in greeen
print('\033[92m' + all_positive_tweets[random.randint(0,5000)])

# print negative in red
print('\033[91m' + all_negative_tweets[random.randint(0,5000)])
```

```
@aliendovecote oooooooooohh! i just saw your #PetJam! i've been toying with a virtual pet
game idea for weeks! now i will pounce!! :D
@WordOfTheFree ago hogo vishaya all adu. BJP madatte anta vishwas ne illa. :(
```

One observation you may have is the presence of emoticons and URLs in many of the tweets.

Preprocess raw text for Sentiment analysis

Data preprocessing:

For NLP, the preprocessing steps are comprised of the following tasks:

- **Tokenizing the string**
- **Lowercasing**
- **Removing stop words and punctuation**
- **Stemming**

Let's see how we can do these to a given tweet. We will choose just one and see how this is transformed by each preprocessing step.

In [23]:

```
# Our selected sample
tweet = all_positive_tweets[2277]
print(tweet)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #
Friday off… https://t.co/3tfYom0N1i
```

Let's import a few more libraries for this purpose.

In [24]:

```
# download the stopwords from NLTK
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[24]:

```
True
```

In [25]:

```
import re                                   # library for regular expression operations
import string                               # for string operations

from nltk.corpus import stopwords           # module for stop words that come with NLTK
from nltk.stem import PorterStemmer         # module for stemming
from nltk.tokenize import TweetTokenizer    # module for tokenizing strings
```

Remove hyperlinks, Twitter marks and styles

Since we have a Twitter dataset, we'd like to remove some substrings commonly used on the platform like the hashtag, retweet marks, and hyperlinks. We'll use the re library to perform regular expression operations on our tweet. We'll define our search pattern and use the `sub()` method to remove matches by substituting with an empty character (i.e. `''`)

In [27]:

```
print('\033[92m' + tweet)
print('\033[94m')

# remove hyperlinks
tweet2 = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet2)

# remove hashtags
# only removing the hash # sign from the word
tweet2 = re.sub(r'#', '', tweet2)

print(tweet2)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #
Friday off… https://t.co/3tfYom0N1i

My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Frid
ay off…
```

Tokenize the string

To tokenize means to split the strings into individual words without blanks or tabs. In this same step, we will also convert each word in the string to lower case. The tokenize module from NLTK allows us to do these easily:

In [29]:

```
print()
print('\033[92m' + tweet2)
print('\033[94m')

# instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False)

# tokenize tweets
tweet_tokens = tokenizer.tokenize(tweet2)

print()
print('Tokenized string:')
print(tweet_tokens)
```

```
My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Frid
ay off…


Tokenized string:
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', '
sunflowers', 'favourites', 'happy', 'friday', 'off', '…']
```

Remove stop words and punctuations

The next step is to remove stop words and punctuation. Stop words are words that don't add significant meaning to the text.

In [30]:

```
#Import the english stop words list from NLTK
stopwords_english = stopwords.words('english')

print('Stop words\n')
print(stopwords_english)
```

```
print('\nPunctuation\n')
print(string.punctuation)
```

```
Stop words

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himsel
f', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'the
m', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if
', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'ag
ainst', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to'
, 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 't
hen', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'sa
me', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'shoul
d', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'c
ouldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn
't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren',
"weren't", 'won', "won't", 'wouldn', "wouldn't"]

Punctuation

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

We can see that the stop words list above contains some words that could be important in some contexts.

These could be words like *i, not, between, because, won, against*.

You might need to customize the stop words list for some applications.

For our exercise, we will use the entire list.

For the punctuation, certain token like ':)' should be retained when dealing with tweets because they are used to express emotions.

In [31]:

```
print()
print('\033[92m')
print(tweet_tokens)
print('\033[94m')

tweets_clean = []

for word in tweet_tokens: # Go through every word in your tokens list
    if (word not in stopwords_english and  # remove stopwords
        word not in string.punctuation):  # remove punctuation
        tweets_clean.append(word)

print('removed stop words and punctuation:')
print(tweets_clean)
```

```
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', '
sunflowers', 'favourites', 'happy', 'friday', 'off', '…']

removed stop words and punctuation:
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites
', 'happy', 'friday', '…']
```

Stemming

Stemming is the process of converting a word to its most general form, or stem. This helps in reducing the size of our vocabulary.

Consider the words:

- **learn**
- **learning**
- **learned**
- **learnt**

All these words are stemmed from its common root learn.

However, in some cases, the stemming process produces words that are not correct spellings of the root word. For example, happi and sunni. That's because it chooses the most common stem for related words. For example, we can look at the set of words that comprises the different forms of happy:

- **happy**
- **happiness**
- **happier**

We can see that the prefix happi is more commonly used. We cannot choose happ because it is the stem of unrelated words like happen.

NLTK has different modules for stemming and we will be using the PorterStemmer

In [32]:

```
print()
print('\033[92m')
print(tweets_clean)
print('\033[94m')

# Instantiate stemming class
stemmer = PorterStemmer()

# Create an empty list to store the stems
tweets_stem = []

for word in tweets_clean:
    stem_word = stemmer.stem(word)  # stemming word
    tweets_stem.append(stem_word)  # append to the list

print('stemmed words:')
print(tweets_stem)
```

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites
', 'happy', 'friday', '…']

stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit', 'happi', 'f
riday', '…']
```

EXERCISE:

Perform all of the above preprocessing tasks on any other text dataset