```
import pandas as pd
import numpy as np
data= pd.read_csv('training.1600000.processed.noemoticon.csv', encoding='cp437')
data
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import string
nltk.download('stopwords')
nltk.download('punkt')
df= pd.DataFrame()
df['text']=  data['text of the tweetá'].apply(str.lower)
df['polarity_index']= data['polarity of tweetá']
df.head()
import string

def remove_punc(text):
    removed_text = ""
    for char in str(text):
        if char not in string.punctuation:
            removed_text+=char
    return removed_text
df['clean_text'] = df['text'].apply(remove_punc)
df.head()
df['tokenized']=df['clean_text'].apply(TweetTokenizer().tokenize)
df.info()
df.head()
def rem_stop(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]
    return filtered_tokens
df['stopwords']= df['tokenized'].apply(rem_stop)
df.head()
def lemma_tokens(tokens):
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return tokens
df['lemma_txt'] = df['stopwords'].apply(lemma_tokens)
df.head()
def concat_text(tokens):
  return " ".join([token for token in tokens])
df['preprocessed_text'] = df['lemma_txt'].apply(concat_text)
df.head()
### BOW
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
count_matrix = cv.fit_transform(df['preprocessed_text'].values.tolist())
from sklearn.model_selection import train_test_split
x_train_bow, x_test_bow, y_train_bow, y_test_bow = train_test_split(count_matrix,
df['polarity_index'], test_size=0.25, random_state=42)
### TFIDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf= TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(df['preprocessed_text'].values.tolist())
from sklearn.model_selection import train_test_split
x_train_tfidf, x_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_split(tfidf_matrix,
df['polarity_index'], test_size=0.25, random_state=42)
### CBOW
from gensim.models.word2vec import Word2Vec
```

```
cbow = Word2Vec(df['lemma_txt'], vector_size=100, window=5, min_count=2, sg=0)
vocab = cbow.wv.index_to_key

def get_mean_vector(model, sentence):
    words = [word for word in sentence if word in vocab]
    if len(words) >= 1:
        return np.mean(model.wv[words], axis=0)
    return np.zeros((100,))

cbow_vector = [get_mean_vector(cbow, sentence) for sentence in df['lemma_txt']]
x_train_cbow, x_test_cbow, y_train_cbow, y_test_cbow = train_test_split(cbow_vector, df['polarity_index'], test_size=0.25, random_state=42)
```

Skipgram

```
sg = Word2Vec(df['preprocessed_text'].values.tolist(), vector_size=100, window=5, min_count=2, sg=1)
vocab = sg.wv.index_to_key

def get_mean_vector(model, sentence):
    words = [word for word in sentence if word in vocab]
    if len(words) >= 1:
        return np.mean(model.wv[words], axis=0)
    return np.zeros((100,))

sg_vector = []
for sentence in df['preprocessed_text'].values.tolist():
    sg_vector.append(get_mean_vector(sg, sentence))

sg_vector = np.array(sg_vector)
x_train_sg, x_test_sg, y_train_sg, y_test_sg = train_test_split(sg_vector, df['polarity_index'], test_size=0.25, random_state=42)
```

Model implementation

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
dt = DecisionTreeClassifier(random_state=42,max_depth=7)
def Decision_Tree(x_train, x_test, y_train, y_test):

   dt = DecisionTreeClassifier(random_state=42,max_depth=7)
   dt.fit(x_train, y_train)
   y_pred = dt.predict(x_test)

   print("Accuracy:", accuracy_score(y_test, y_pred))
   print("Classification Report:\n", classification_report(y_test, y_pred))
   return dt
dt_bow = Decision_Tree(x_train_bow, x_test_bow, y_train_bow, y_test_bow)
dt_tfidf = Decision_Tree(x_train_tfidf, x_test_tfidf, y_train_tfidf, y_test_tfidf)
dt_cbow = Decision_Tree(x_train_cbow, x_test_cbow, y_train_cbow, y_test_cbow)
dt_sg = Decision_Tree(x_train_sg, x_test_sg, y_train_sg, y_test_sg)
test_para = pd.DataFrame({'text':['What is not to like about this product.',
'Not bad.',
'Not an issue.',
'Not buggy.',
'Not happy.',
'Not user-friendly.',
'Not good.',
'Is it any good?',
'I do not dislike horror movies.',
'Disliking horror movies is not uncommon.',
'Sometimes I really hate the show.',
'I love having to wait two months for the next series to come out!',
'The final episode was surprising with a terrible twist at the end.',
'The film was easy to watch but I would not recommend it to my friends.',
'I LOL’d at the end of the cake scene']})
test_para['text']= test_para['text'].apply(str.lower)
```

```
test_para['text']= test_para['text'].apply(remove_punc)
test_para['tokenized'] =test_para['text'].apply(TweetTokenizer().tokenize)
test_para['lemma_txt'] = test_para['tokenized'].apply(lemma_tokens)
test_para['preprocessed_text'] = test_para['lemma_txt'].apply(concat_text)
test_para.head()
from gensim.models.word2vec import Word2Vec

cbow_test = Word2Vec(test_para['lemma_txt'], vector_size=100, window=5, min_count=2, sg=0)
vocab_test = cbow_test.wv.index_to_key

def get_mean_vector(model, sentence):
    words = [word for word in sentence if word in vocab_test]
    if len(words) >= 1:
        return np.mean(model.wv[words], axis=0)
    return np.zeros((100,))

cbow_vector_test = [get_mean_vector(cbow_test, sentence) for sentence in test_para['lemma_txt']]
test_para['prediction']=dt_cbow.predict(cbow_vector_test)
for i,j in enumerate(test_para['prediction']):
    if test_para['prediction'][i] == 0:
        test_para['prediction'][i] = 'Negative'
    elif test_para['prediction'][i] == 4:
        test_para['prediction'][i] = 'Positive'
test_para
```