In [3]:
```
import os
import string
import nltk
import random
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.tokenize import RegexpTokenizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

Out[3]:
```
True
```

In [4]:
```
def preprocess_text(raw_text):

  # Tokenize the raw text and lowercase
  tokenized_raw_text = word_tokenize(raw_text.lower())
  # print(tokenized_raw_text)

  # Filter out tokens to tokens that are alpha
  alpha_tokens = [token for token in tokenized_raw_text if token.isalpha()]
  # print(alpha_tokens)

  # Filter out tokens to tokens that not in the in the stopword list
  stopwords_tokens = [token for token in alpha_tokens if token not in stopwords.words('english')]
  # print(stopwords_tokens)

  return stopwords_tokens

# TESTING
text = "The game everyone’s talking about in the indie PC world right now doesn’t involve blasting aliens or co
print(preprocess_text(text)[:10])
```

```
['game', 'everyone', 'talking', 'indie', 'pc', 'world', 'right', 'involve', 'blasting', 'aliens']
```

In [5]:
```
proc_dir = 'process/'

# This list holds the all relevant terms
terms_list = []
corpus = ""
file_names = os.listdir(proc_dir)

# Go thru all files
for file in file_names:
  print("Processing %s" % file)

  file_loc = proc_dir + file
  with open(file_loc, 'r') as f:
    file_contents = f.read()
    corpus += file_contents
    processed_text = preprocess_text(file_contents)
    combined_string = ' '.join(processed_text)

    # Write all processed words into the list
    terms_list.append(combined_string)
```

```
Processing materiacollective-clean-18.txt
Processing eurogamer-clean-8.txt
Processing mcvuk-clean-11.txt
Processing pcgamer-clean-22.txt
Processing vg247-clean-7.txt
Processing destructoid-clean-9.txt
Processing polygon-clean-15.txt
Processing siliconera-clean-4.txt
Processing king5-clean-21.txt
Processing destructoid-clean-17.txt
Processing eurogamer-clean-12.txt
Processing usgamer-clean-10.txt
Processing gamesindustry-clean-20.txt
Processing en-wikipedia-clean-2.txt
Processing pastemagazine-clean-5.txt
Processing web-archive-clean-1.txt
Processing pcgamer-clean-3.txt
Processing gamespot-clean-19.txt
Processing materiacollective-bandcamp-clean-14.txt
```

In [6]:
```python
# Used https://www.youtube.com/watch?v=_RhHA_tYYXI&ab_channel=CodeWithAarohi as reference
def calc_tfidf_dict(tokens):

  # Computing TF-IDF

  tfidf_vectorizer = TfidfVectorizer()
  tfidf_fit = tfidf_vectorizer.fit_transform(tokens)
  # print(tfidf_fit)

  # Get feature names
  terms = tfidf_vectorizer.get_feature_names_out()
  # print(terms)

  # Create a dictionary
  term_tfidf_dict = {}
  for col in tfidf_fit.nonzero()[1]:
    # print(terms[col], '-', tfidf_fit[0, col])
    # Insert into dictionary
    term_tfidf_dict[terms[col]] = tfidf_fit[0, col]

  # Sorting dict
  sorted_dict = dict(sorted(term_tfidf_dict.items(), key=lambda item: item[1], reverse=True))
  return sorted_dict
```

In [7]:
```python
def get_important_terms(tfidf_dict, N):
  # N = number of terms extracted from the TF-IDF dictionary
  important_terms = []
  N_terms = dict(list(tfidf_dict.items())[:N])
  for key, value in N_terms.items():
    # print(key)
    important_terms.append(key)

  return important_terms
```

In [8]:
```python
tfidf_dict = calc_tfidf_dict(terms_list)
# print(tfidf_dict)

top_terms = get_important_terms(tfidf_dict, 40)
print("Top 40 important terms:")
for term in top_terms:
  print(term)
```

```
Top 40 important terms:
music
arrangements
materia
augustine
album
collective
concernedape
trademark
llc
illustrator
pianist
meadow
composed
stardew
game
improving
gonzales
mayuga
wife
composition
musical
bridgham
solo
eric
year
valley
taken
collections
piano
hope
based
barone
experience
spent
always
love
even
recordings
shines
liberties
```

In [9]:
```python
def check_word_in_sentence(word, sentence):
  # Convert both word and sentence to lowercase for case-insensitive comparison
  word = word.lower()
  sentence = sentence.lower()

  # Check if the word is present in the sentence
  if word in sentence.split():
    return True
  else:
    return False

# Testing
word1 = "word"
sentence1 = "Word is in this sentence"
sentence2 = "Werd isn't in this sentence"
print(check_word_in_sentence(word1, sentence1))
print(check_word_in_sentence(word1, sentence2))
```

```
True
False
```

In [10]:
```python
# Manually determine the top 10-15 terms based on your domain knowledge (not ranked)
# 1. Stardew Valley
# 2. ChuckleFish
# 3. Eric Barone
# 4. mobile
# 5. music
# 6. multiplayer
# 7.
# 8. ConcernedApe
# 9. album
# 10. android
# 11. version
# 12. iOS
# 13. update
```

In [11]:
```python
# Initial knowledge base
knowledge_base = {}

# Initial infomation for each term
stardew_valley_info = []
ChuckleFish_info = []
Eric_Barone_info = []
mobile_info = []
music_info = []
multiplayer_info = []
```

```
ConcernedApe_info = []
album_info = []
android_info = []
version_info = []
ios_info = []
update_info = []

# Tokenize the text into sentences
sentences = sent_tokenize(corpus)

# Update infomation
for sentence in sentences:
  if check_word_in_sentence("stardew", sentence) or check_word_in_sentence("valley", sentence):
    stardew_valley_info.append(sentence)

  if check_word_in_sentence("ChuckleFish", sentence):
    ChuckleFish_info.append(sentence)

  if check_word_in_sentence("Eric", sentence) or check_word_in_sentence("Barone", sentence):
    Eric_Barone_info.append(sentence)

  if check_word_in_sentence("ConcernedApe", sentence):
    ConcernedApe_info.append(sentence)

  if check_word_in_sentence("mobile", sentence):
    mobile_info.append(sentence)

  if check_word_in_sentence("music", sentence):
    music_info.append(sentence)

  if check_word_in_sentence("multiplayer", sentence):
    multiplayer_info.append(sentence)

  if check_word_in_sentence("album", sentence):
    album_info.append(sentence)

  if check_word_in_sentence("android", sentence):
    android_info.append(sentence)

  if check_word_in_sentence("version", sentence):
    version_info.append(sentence)

  if check_word_in_sentence("iOS", sentence):
    ios_info.append(sentence)

  if check_word_in_sentence("update", sentence):
    update_info.append(sentence)


knowledge_base["Stardew Valley"] = stardew_valley_info
knowledge_base["ChuckleFish"] = ChuckleFish_info
knowledge_base["Eric Barone"] = Eric_Barone_info
knowledge_base["ConcernedApe"] = ConcernedApe_info
knowledge_base["mobile"] = mobile_info
knowledge_base["music"] = music_info
knowledge_base["multiplayer"] = multiplayer_info
knowledge_base["album"] = album_info
knowledge_base["android"] = android_info
knowledge_base["version"] = version_info
knowledge_base["iOS"] = ios_info
knowledge_base["update"] = update_info

# Update infomation
#print(knowledge_base)

for key in knowledge_base.keys():
  print(key, len(knowledge_base[key]))
```

```
Stardew Valley 125
ChuckleFish 11
Eric Barone 72
ConcernedApe 8
mobile 8
music 12
multiplayer 15
album 8
android 6
version 10
iOS 11
update 14
```

In [13]:

```
import pickle

with open('knowledge_base.pickle', 'wb') as handle:
  pickle.dump(knowledge_base, handle)
```