Edge Hill University

The Department of Computer Science

CIS4515
Practical Data Analysis
Level 7

Coursework 1
Task 2
Question 5 (Report)

2023/2024

Student: **Joseph Z. Moyo**

Student No: 25792334

1. Introduction

Amazon's Data Analysis department ran a mini project to analyse the positive customer reviews of gaming products that were sold. The task involved the use of the NLTK library in Python for natural language processing (NLP). This task was important to identify the common products that were referenced by customers in their positive reviews, enabling Amazon to note potential items to focus their stock budget on. In this exercise it was revealed that the top five best reviewed games were Resident Evil, Grand Theft Auto, Tony Hawk, Eternal Darkness and Mario Smash Bros. The X box gaming console was also highly referenced in the reviews, implying perhaps the gaming platform is the most popular for the customers.

2. Portfolio Task

The task involved importing text data of Amazon reviews, pre-processing it to query the collocations found in the data, revealing important information that could give Amazon clear understanding of its target market.

2.1. Question 1

The initial step in the exercise was to import the reviews from the text file into the python jupyter notebook workspace. Several modules were imported from the NLTK python library to read and pre-process the reviews text.

The regular expression tokenisation model 'regexp_tokenize' was used to tokenise the words in the reviews. It was called with an argument to ensure that words that have an apostrophe were not split (NLTK, 2023). The 'corpus()' module and its methods were used to remove common English language stop words. The tokens were subsequently tagged using the 'pos_tag()' module to identify the part-of-speech they belonged to. The tokens were further grouped in pairs called bigrams as they appeared in the reviews. This pairing helps to reduce the dimensions of the data while conveying more information (Medium, 2019). The first exercise's aim was to produce a collection of bigrams of token pairs with their respective part of speech tags.

The screenshots of the documented code are shown:

In [1]:

```python
from nltk import pos_tag, bigrams, regexp_tokenize, corpus
```

- The lines for each review instance were read from the txt file using python built in function 'open()'.
- The words in each line were set to lower case using the '.lower()' function to achieve uniformity and avoid counting the same word twice because of capitalisation.

In [2]:

```python
file = open(file='./positive_reviews_of_Video_Games.txt', mode = 'r')
lines = file.readlines()
doccie = ''

"""- The 'for loop' runs through each review line and adds it into a doccument called 'doccie'.
   - The lines are words are converted to lower case for consistency using '.lower()' method"""
for line in lines:
    doccie += line.split('\t')[2].lower()
file.close()

stop_words = set(corpus.stopwords.words('english'))

"""This is an alternative method for tokenising words. It allows us to set a filter pattern.
   In this case, words with an apostrophe are to be retained as is and will not be split into
   two tokens."""
wds = regexp_tokenize(doccie, pattern = r"[\w']+")

"""The tokenized list is filtered here to remove common English language stopwords."""
wds = [wrd for wrd in wds if not wrd in stop_words]
```

- The tokenization function from NLTK produces a list of all the words that appear in the read reviews column of the text.

In [4]:

```python
wds
```

Out[4]:

```
['13',
 'year',
 'old',
 'fps',
 'fan',
 'speaking',
 'word',
 'one',
 'best',
 'fps',
 'fan',
 'played',
 'ever',
```

- The 'pos_tag()' function was called with the tokenized words as input, giving each token alongside its part of speech tag.

In [6]:

```
wds_with_pos
for w, pos in wds_with_pos:
    print(w, pos)
```

```
good JJ
sloppy JJ
times NNS
sounds VBZ
better RBR
ghost NN
recon JJ
fact NN
blood NN
bad JJ
thing NN
means VBZ
people NNS
play VBP
ai JJ
acts NNS
smart JJ
stupid JJ
time NN
snipers NNS
```

- The 'bigrams()' function was called with the pos_tagged tokens as input, producing tuple pairs of tokens that appear next to each other in the reviews, with each token in a nested tuple with its respective pos_tag.

In [7]:

```
bigrams_pos = bigrams(wds_with_pos)

bigrams_pos = list(bigrams_pos)
```

In [8]:

```
bigrams_pos
```

Out[8]:

```
[(('13', 'CD'), ('year', 'NN')),
 (('year', 'NN'), ('old', 'JJ')),
 (('old', 'JJ'), ('fps', 'NN')),
 (('fps', 'NN'), ('fan', 'NN')),
 (('fan', 'NN'), ('speaking', 'VBG')),
 (('speaking', 'VBG'), ('word', 'NN')),
 (('word', 'NN'), ('one', 'CD')),
 (('one', 'CD'), ('best', 'JJS')),
 (('best', 'JJS'), ('fps', 'NN')),
```

- Finally in the cell below the bigrams are printed showing their respective member tokens and their pos_tags.

In [9]:

```
"""- frst_e and snd_e denote first element and second element of the bigram.
   - frst_w and frst_pos imply first word and the respective first pos_tag.
   - snd in this case then means second."""

for frst_e, snd_e in bigrams_pos:
    frst_w, frst_pos = frst_e[0], frst_e[1]
    snd_w, snd_pos = snd_e[0], snd_e[1]
    print(frst_w, frst_pos, snd_w, snd_pos)
```

```
sloppy JJ times NNS
times NNS sounds VBZ
sounds VBZ better RBR
better RBR ghost NN
ghost NN recon JJ
recon JJ fact NN
fact NN blood NN
blood NN bad JJ
bad JJ thing NN
thing NN means VBZ
means VBZ people NNS
people NNS play VBP
play VBP ai JJ
ai JJ acts NNS
acts NNS smart JJ
smart JJ stupid JJ
stupid JJ time NN
time NN snipers NNS
snipers NNS must MD
must MD auto NN
```

2.2. Question 2

The follow up task involved extracting the 40 most important bigrams in the reviews text using the co-occurrence frequency algorithm introduced in class tutorial exercises. The data was imported and pre-processed in the same manner as in question 1. A function called ‘freq_of_bigrams()’ was created and it took as input the pos_tagged bigrams produced by the code from question_1.

The screenshot for the code can be seen below:

- A frequency calculation function called ***freq_of_bigrams()*** was created.
- The function takes pos_tagged bigrams as input and gives an output of each bigram and its frequency as observed from the tokenized word list.

```python
def freq_of_bigrams(bigrams_pos):
#An empty dictionary is used to instantiate the repository of bigrams and their frequencies.
    bigrams_freqs = {}

    for frst_e, snd_e in bigrams_pos:
        frst_w, frst_pos = frst_e[0], frst_e[1]
        snd_w, snd_pos = snd_e[0], snd_e[1]
#An 'if' statement is added evaluate if the bigram has been ecountered before in the
#the loop. If true, the counter adds 1 to the frequency value for the bigram.
        if (frst_w, snd_w) in bigrams_freqs:
            bigrams_freqs[(frst_w, snd_w)] += 1
#If the loop is encountering the bigram for the first time, it adds the bigram and
#assigns a value of 1 to it.
        else:
            bigrams_freqs[(frst_w, snd_w)] = 1
    return bigrams_freqs
```

```python
freq_of_bigrams(bigrams_pos)
```

```
('sloppy', 'times'): 1,
('times', 'sounds'): 1,
('sounds', 'better'): 1,
('better', 'ghost'): 1,
('ghost', 'recon'): 39,
('recon', 'fact'): 1,
('fact', 'blood'): 1,
('blood', 'bad'): 2,
('bad', 'thing'): 28,
('thing', 'means'): 1,
('means', 'people'): 1,
('people', 'play'): 12,
('play', 'ai'): 2,
('ai', 'acts'): 1,
('acts', 'smart'): 1,
('smart', 'stupid'): 1,
('stupid', 'time'): 1,
('time', 'snipers'): 1,
('snipers', 'must'): 1,
('must', 'auto'): 1
```

- The frequency list of the bigrams is then sorted in descending order using the code below:

```python
#The code sorts the dictionary produced by the freq_of_bigrams function by descending order of the values.
#The top 40 bigrams by frequency are given below.
sorted_freq_of_bigrams = dict(sorted(freq_of_bigrams(bigrams_pos).items(), key=lambda item: item[1], reverse=True)[:4
```

```python
sorted_freq_of_bigrams
```

```
{('great', 'game'): 319,
 ('game', 'play'): 240,
 ('one', 'best'): 227,
 ('resident', 'evil'): 220,
 ('x', 'box'): 189,
 ('replay', 'value'): 179,
 ('play', 'game'): 177,
 ('fun', 'game'): 176,
 ('game', 'ever'): 175,
 ('games', 'like'): 171,
 ('buy', 'game'): 170,
 ('game', 'great'): 169,
 ('super', 'smash'): 168,
 ('single', 'player'): 161,
 ('good', 'game'): 159,
 ('grand', 'theft'): 158,
 ('theft', 'auto'): 158,
 ('first', 'person'): 155,
```

The code produces the top 40 most frequently observed bigrams based on frequency of mentions. The top 10 mostly include common expressions for gaming experience such as ‘great game’, ‘game play’, ‘fun game’, ‘replay value’, and more. The top 10 also included actual video game titles such as ‘Resident Evil’, ‘Grand theft’ and ‘Super Smash’. There were also frequent references to a particular gaming platform in the form of the ‘x box’ console.

2.3. Question 3

The bigrams dictionary produced in question 1 was filtered using a list of pos_tag combinations given in the tutorials to remove pairs that aren’t considered as collocations.

The screenshots below present the code.

- A frequency calculation function called ***freq_o_bigrams()*** was created.
- The function takes pos_tagged bigrams as input and gives an output of each bigram and its frequency as observed from the tokenized word list.
- This time the function includes an 'if' statement to filter the bigrams using pos_tags, removing frequent bigrams that aren't actually collocations.

```
def freq_of_bigrams(bigrams_pos):
#An empty dictionary is used to instantiate the repository of bigrams and their frequencies.
    bigrams_freqs = {}
#A list of acceptable pos_tag pairs from the first and second words of the bigrams.
#This list is used by  the nested if statement to gauge whether the bigram is accepotble.
    pos_list = ['JJNN', 'NNPNNP', 'NNPNN', 'NNNN']

    for frst_e, snd_e in bigrams_pos:
        frst_w, frst_pos = frst_e[0], frst_e[1]
        snd_w, snd_pos = snd_e[0], snd_e[1]
#An 'if' statement is added evaluate if the bigram has been ecountered before in the
#the loop. If true, the next if statement is then evaluated.
        if (frst_w, snd_w) in bigrams_freqs:
#This 'if' statement matches the pos_tags of the two bigram words to see if their combination
#is acceptable as a collocation by observing if it is included in the 'pos_list' defined above.
#If true, the counter adds 1 to the frequency value for the bigram.
            if frst_pos + snd_pos in pos_list:
                bigrams_freqs[(frst_w, snd_w)] += 1
#If the loop is encountering the bigram for the first time, it then tests the condition
#of the 'if' statement nested in the else statement.
        else:
#This 'if' statement matches the pos_tags of the two bigram words to see if their combination
#is included in the 'pos_list' defined above. If true, the counter adds 1 to the frequency value of the bigram.
            if frst_pos + snd_pos in pos_list:
                bigrams_freqs[(frst_w, snd_w)] = 1
    return bigrams_freqs
```

- A pandas Dataframe ***df*** is then created to store and display the top 40 bigrams and their respective frequencies.

- A pandas Dataframe ***df*** is then created to store and display the top 40 bigrams and their respective frequencies.


```
df = pd.DataFrame(list(freq_of_bigrams(bigrams_pos)))
vals = pd.Series(list(freq_of_bigrams(bigrams_pos).values()))
df = pd.concat([df, vals], axis=1)
df.columns = ['1st_word', '2nd_word', 'bigram_frequency']
df.sort_values(by=['bigram_frequency'], ascending=False).head(40)
```

	1st_word	2nd_word	bigram_frequency
82	great	game	319
627	game	play	186
2929	x	box	163
429	fun	game	163
10	single	player	161
114	good	game	158
33139	super	smash	152
973	theft	auto	148
255	game	game	144
155	video	game	137
972	grand	theft	134

The observed collocations and their frequencies are different for some of the entries compared with the dictionary obtained in question 2. There is noticeable change in frequency for some of the pairs that were retained after filtration, e.g., the collocation 'game play' is still number two on the list but the frequency dropped from 240 to 186.

2.4. Question 4

The collocations were alternatively evaluated using the mutual information statistical metric which determines the importance of a pair from the statistical combination of its probability with the probabilities of the two member words, all normalised based on the total number of words in the tokens list (Icalem, 2018).

Two methods were explored:

- using the Mutual Information formula given in the tutorial to mechanically calculate the MI values from the bigram frequency function from question 3 along with a custom function to calculate token frequencies.
- utilising the NLTK library's modules for computing MI values.

The code used for the first method is shown below.

Method 1

- Similar to ***Question 3***, a frequency calculation function called ***freq_o_bigrams()*** was created.
- The function takes pos_tagged bigrams as input and gives an output of each bigram and its frequency as observed from the tokenized word list.
- This time the function includes an 'if' statement to filter the bigrams using pos_tags, removing frequent bigrams that aren't actually collocations.

```python
def freq_of_bigrams(bigrams_pos):
#An empty dictionary is used to instantiate the repository of bigrams and their frequencies.
    bigrams_freqs = {}
#A list of acceptable pos_tag pairs from the first and second words of the bigrams.
#This list is used by  the nested if statement to gauge whether the bigram is accepatble.
    pos_list = ['JJNN', 'NNPNNP', 'NNPNN', 'NNNN']

    for frst_e, snd_e in bigrams_pos:
        frst_w, frst_pos = frst_e[0], frst_e[1]
        snd_w, snd_pos = snd_e[0], snd_e[1]
#An 'if' statement is added evaluate if the bigram has been ecountered before in the
#the loop. If true, the next if statement is then evaluated.
        if (frst_w, snd_w) in bigrams_freqs:
#This 'if' statement matches the pos_tags of the two bigram words to see if their combination
#is acceptable as a collocation by observing if it is included in the 'pos_list' defined above.
#If true, the counter adds 1 to the frequency value for the bigram.
            if frst_pos + snd_pos in pos_list:
                bigrams_freqs[(frst_w, snd_w)] += 1
#If the loop is encountering the bigram for the first time, it then tests the condition
#of the 'if' statement nested in the else statement.
        else:
#This 'if' statement matches the pos_tags of the two bigram words to see if their combination
#is included in the 'pos_list' defined above. If true, the counter adds 1 to the frequency value of the bigram.
            if frst_pos + snd_pos in pos_list:
                bigrams_freqs[(frst_w, snd_w)] = 1
    return bigrams_freqs
```

- A function for counting words called ***freq_of_wds*** is created in the cell below.
- The function returns a dictionary with all the tokenized words and their respective frequencies in the reviews.

```python
def freq_of_wds(wds):
#The function starts with an empty dictionary to instantiate the collection of words in and
# their frequencies.
    wds_freqs = {}
#A 'for-loop' goes over each over in the tokenized list
    for wrd in wds:
#An 'if' statement checks a condition if a word on the current loop has been added to the wds_freq
#dictionary. If true, 1 is added to the corresponding value of the word in the dictionary
        if wrd in wds_freqs:
            wds_freqs[wrd] += 1
#If the word has not been added before, the else statement is then activated and the word is
#entered into the dictionary for the first time and given a value of 1.
        else:
            wds_freqs[wrd] = 1
    return wds_freqs
```

```python
freq_of_wds = freq_of_wds(wds)
```

- A pandas Dataframe ***wds_df*** is then created to store and display the words/tokens and their respective frequencies.

```python
wds_df = pd.DataFrame(data = list(freq_of_wds))            #creating word freq dataframe
wrd_freqs = pd.Series(list(freq_of_wds.values()))
wds_df = pd.concat([wds_df, wrd_freqs], axis=1)
wds_df.columns = ['word', 'word_frequency']
```

- The top 40 most common tokens in the reviews. This is after the removal of the common English stop words.

```python
wds_df.sort_values(by=['word_frequency'], ascending=False).head(40)
```

	word	word_frequency
65	game	11137
72	games	3020
79	like	2790
7	one	2710
189	get	2393
25	play	2063

```python
plt.figure(1, figsize=(10,10))
sns.barplot(x = wds_df.sort_values(by=['word_frequency'], ascending=False).head(10)['word'],
            y = wds_df.sort_values(by=['word_frequency'], ascending=False).head(10)['word_frequency'], )
plt.title('The top 10 words by frequency in the reviews text')
plt.show()
```

Figure 1 below shows the top 10 most common words in the reviews text.

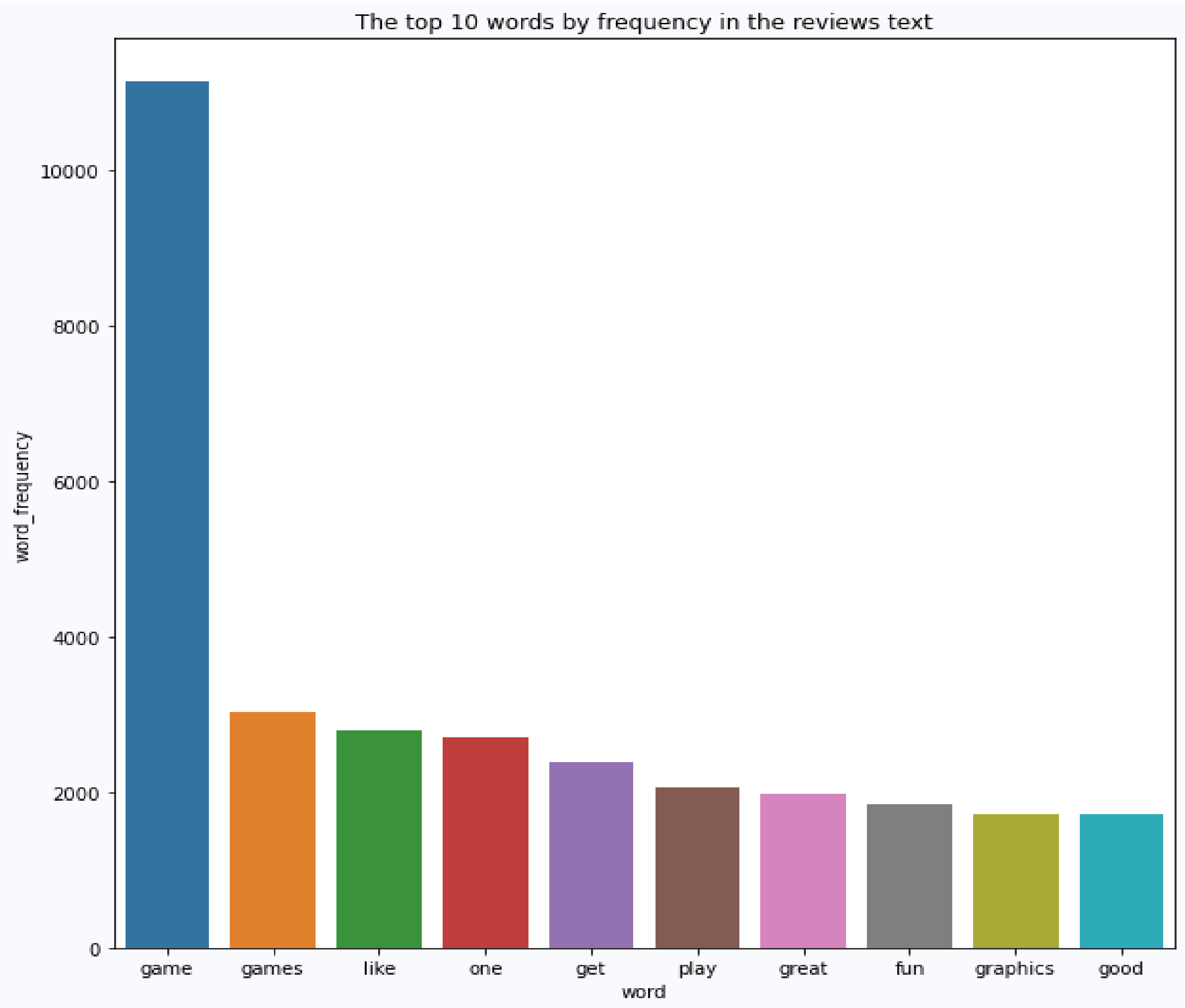

Figure 1: The most common words in the reviews text, showing that the word 'game' was highest. Positive words such as 'great', 'fun', 'like' and 'good' also appeared in the top 10.

- A pandas Dataframe ***df*** is then created to store and display the top 40 bigrams and their respective frequencies.

```
df = pd.DataFrame(list(freq_of_bigrams(bigrams_pos)))
vals = pd.Series(list(freq_of_bigrams(bigrams_pos).values()))
df = pd.concat([df, vals], axis=1)
df.columns = ['1st_word', '2nd_word', 'bigram_frequency']
df.sort_values(by=['bigram_frequency'], ascending=False).head(40)
```

	1st_word	2nd_word	bigram_frequency
82	great	game	319
627	game	play	186
2929	x	box	163
429	fun	game	163
10	single	player	161
114	good	game	158
33139	super	smash	152
973	theft	auto	148
255	game	game	144
155	video	game	137
972	grand	theft	134

- The Mutual Information of the bigrams calculations below are based on N = total number of words tokenized.

```
N = len(wds)
N
```

316022

- A 'for-loop' is used to go through each bigram in the bigrams dataframe ***df***.
- The code below demonstrates how the mutual information is calculated using the formula provided in week 3 tutorial.

```
#The mutual information list is initialised below:
```

- A 'for-loop' is used to go through each bigram in the bigrams dataframe ***df***.
- The code below demonstrates how the mutual information is calculated using the formula provided in week 3 tutorial.

```
#The mutual information list is initialised below:
mi_list = []
#'for-loop' from index 0 to the last entry on the bigrams dataframe 'df'.
for i in range(len(df)):
#The block below takes the strings of the first and second words in each bigram in the loop
    frst_w = df.iloc[i][0]
    snd_w = df.iloc[i][1]
#The strings of words taken are then used to reference for the respective word frequencies
# in the the 'freq_wds' dataframe to calculate the probabilities of each word.
#The probability of the bigram itself is also calculated from its frequency recorded in 'df'
    p_frst_w = freq_of_wds[frst_w]/N
    p_snd_w = freq_of_wds[frst_w]/N
    p_bigram = df.iloc[i][2]/N
#The mutual information of the current bigram in the loop is then appended on the 'mi_list'
    mi_list.append(math.log(p_bigram/(p_frst_w*p_snd_w),2.0))

```

- A pandas Dataframe ***df*** is then created to store and display the top 40 bigrams and their respective mutual information values.
- It is worth noting that the formula given in the tutorial gives the same value for numerous bigrams.

```
mi = pd.Series(mi_list)
df = pd.concat([df, mi], axis=1)
df.columns = ['1st_word', '2nd_word', 'bigram_frequency', 'mi']
df.sort_values(by=['mi'], ascending=False).head(40)
```

	1st_word	2nd_word	bigram_frequency	mi
20276	hospitalbe	driver	1	18.269665
4823	ivory	bring	1	18.269665
15681	maneuverable	hovercraft	1	18.269665
35372	shinier	brawl	1	18.269665
15682	invulnerable	destruction	1	18.269665

The first method produced the top 40 collocations with the highest MI values. These pairs had a frequency of 1. They also had the same MI value, implying the method did not produce meaningful results.

The second method enlisted the modules in the NLTK library.

The code is presented in the screenshots below.

Method 2

- In the alternative method the NLTK library's modules are used for evaluating the mutual information metric:
  - 'FreqDist()' -> calculates the token frequencies and returns them in a list of tuples containing the token and its respective frequency value. It has a method call '.most_common()' which sorts the tokens by descending frequency.
  - 'collocations()' -> locates collocations in the tokenized list. The method call '.BigramAssocMeasures() is called and it gives a collection of bigram association measures which are used in this exercise.'
  - 'BigramCollocationFinder()' -> finds and ranks bigram collocations or other association measures.

```
#This function actually gives the same results as the frequency functions used in Method 1
wrd_freq_dist = FreqDist(wds)
wrd_freq_dist.most_common(40)
```

```
[('game', 11137),
 ('games', 3020),
 ('like', 2790),
 ('one', 2710),
 ('get', 2393),
 ('play', 2063),
 ('great', 1966),
 ('fun', 1839),
 ('graphics', 1722),
 ('good', 1709),
 ('time', 1566),
 ('first', 1436),
 ('best', 1371),
 ('really', 1329),
 ('also', 1321),
 ('even', 1316),
 ('well', 1229),
 ('much', 1140),
 ('2', 1110),
```

```
#Bigrams MI calculations
bigram_measure = collocations.BigramAssocMeasures()

bigrams_finder = BigramCollocationFinder.from_words(wds)

#The probabilities of each bigram are calculated using the code below:
p_bigram = bigrams_finder.score_ngrams(bigram_measure.raw_freq)
```

```
#The bigrams with the top 10 highest probabilities
p_bigram[:10]
```

```
[(('great', 'game'), 0.0010094233945737955),
 (('game', 'play'), 0.000759440798425426),
 (('one', 'best'), 0.0007183044218440489),
 (('resident', 'evil'), 0.0006961540652233073),
 (('x', 'box'), 0.0005980596287600231),
 (('replay', 'value'), 0.0005664162621589636),
 (('play', 'game'), 0.0005600875888387518),
 (('fun', 'game'), 0.0005569232521786458),
 (('game', 'ever'), 0.0005537589155185399),
 (('games', 'like'), 0.0005411015688878116)]
```

- The Mutual Information (MI) of the bigrams is then calculated by changing the method call on the code that was used to calculate the probabilities above.
- The method call used is '.mi_like'

```
#Mutual Information (MI)
mi_bigram = bigrams_finder.score_ngrams(bigram_measure.mi_like)
```

- A pandas Dataframe ***mi_df*** is then created to store and display the top 40 bigrams and their respective mutual information values.

```
mi_df = pd.DataFrame(mi_bigram, columns=['bigram', 'mi'])
mi_df.sort_values(by=['mi'], ascending=False).head(40)
```

	bigram	mi
0	(resident, evil)	155.854801
1	(theft, auto)	132.071388
2	(grand, theft)	130.627985
3	(tony, hawk)	99.840474
4	(replay, value)	92.151724
5	(eternal, darkness)	82.455944
6	(smash, bros)	73.070560
7	(x, box)	71.900030
8	(golden, sun)	53.934750

```
plt.figure(3, figsize=(17,15))
sns.barplot(x = mi_df.sort_values(by=['mi'], ascending=False).head(10)['bigram'],
            y = mi_df.sort_values(by=['mi'], ascending=False).head(10)['mi'], )
plt.title('The top 10 bigrams by Mutual Information value')
plt.show()
```

The second method's results gave the top 10 collocations presented in Figure 2 below. Seven of the top 10 collocations were actual video game titles. One of the popular collocations was the X Box gaming console.

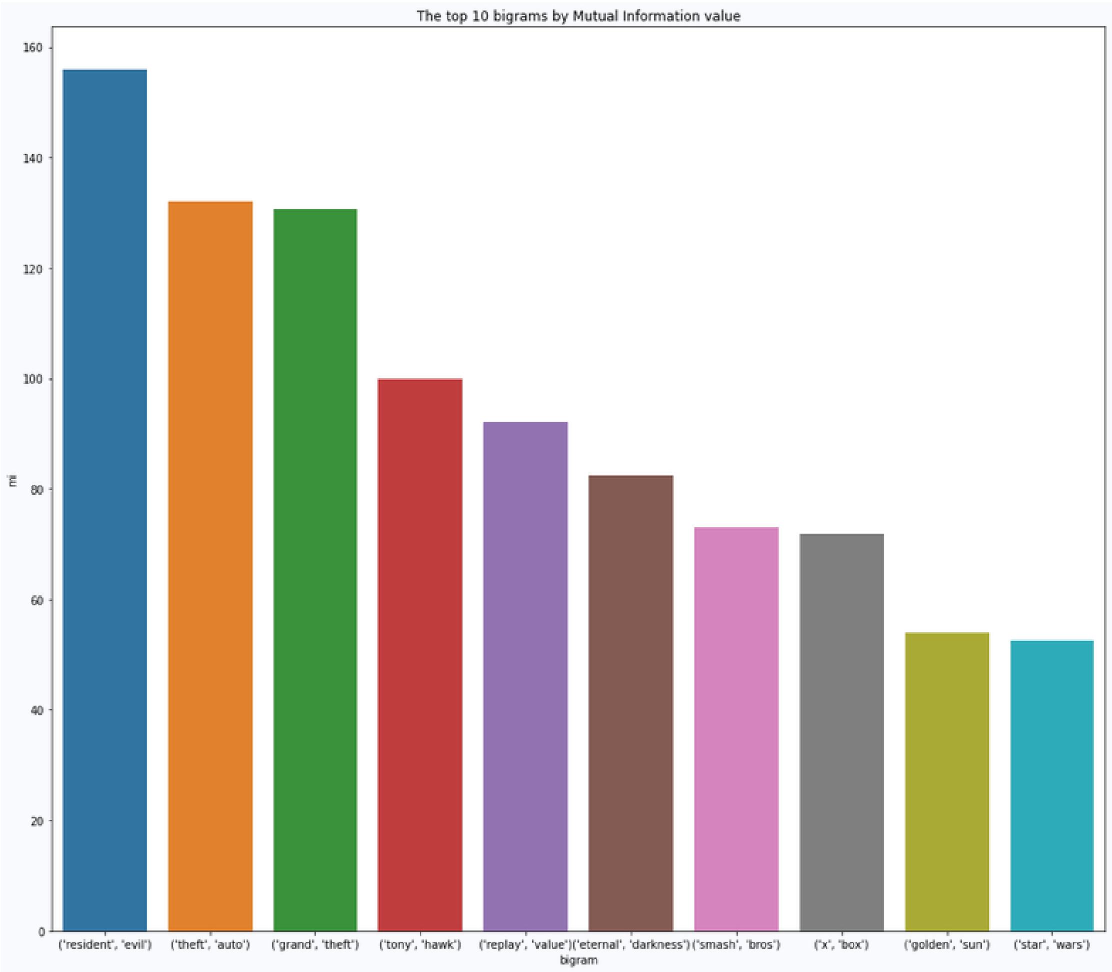

Figure 2: The top 10 collocations/bigrams by Mutual Information.

An alternative to the MI calculation in the form of the Pointwise Mutual Information (PMI) produced different values, giving values like those observed in method 1.

The screenshots below reveal the steps taken.

- An alternative approach is to call a different method instead of the 'mi_like'.
- The '.pmi' method actually gives the Pointwise Mutual Information, which is a variation of mutual information.

```
#Pointwise Mutual Information
#bigrams_finder.apply_freq_filter()
pmi_bigram = bigrams_finder.score_ngrams(bigram_measure.pmi)
```

- A pandas Dataframe ***pmi_df*** is then created to store and display the top 40 bigrams and their respective pointwise mutual information values.

```
pmi_df = pd.DataFrame(pmi_bigram, columns=['bigram', 'pmi'])
pmi_df.sort_values(by=['pmi'], ascending=False).head(40)
```

	bigram	pmi
0	("melody, pianistonio")	18.269665
723	(redman, del)	18.269665
634	(ofaction, swordfighting)	18.269665
635	(officier, wiggam)	18.269665
636	(ofhas, optiontional)	18.269665
637	(ofyour, whereabouts)	18.269665
638	(omar, tapia)	18.269665
639	(ooohing, ahhing)	18.269665
640	(oppressive, fascist)	18.269665
641	(optiontional, missionsdecent)	18.269665
642	(orginizm, equiped)	18.269665

The PMI calculation was further modified to filter out collocations that appear less than three times, resulting in the top PMI values of 16.685. The screenshot below shows the calculation. This tweak produced slightly meaningful collocations with examples, 'Hulk Hogan' and 'Ted Diase' who are actually WWF wrestlers, and 'Flight Simulator' and 'Striker 1945' which are game titles.

- The Pointwise Mutual Information method produced a top 40 collection with bigrams with the same value as the bigrams seen from method 1.
- The top 40 bigrams shown are not the same as those in method 1, but have the same values perhaps the algorithm followed a different arrangement.
- This shows that the formula given in the tutorial is not for mutual information but for pointwise mutual information.

- If we apply a filter to remove bigrams that occurred less than three times

```
bigrams_finder.apply_freq_filter(3)
pmi_bigram = bigrams_finder.score_ngrams(bigram_measure.pmi)

pmi_df_filtered = pd.DataFrame(pmi_bigram, columns=['bigram', 'pmi'])
pmi_df_filtered.sort_values(by=['pmi'], ascending=False).head(40)
```

	bigram	pmi
0	("flight, simulator")	16.684703
3	(oliver, platt)	16.684703
4	(shinji, mikami)	16.684703
1	(fw, 190)	16.684703
2	(hideki, kamiya)	16.684703
5	(billy, zane)	16.269665
6	(hulk, hogan)	16.269665
7	(kelly, slater)	16.269665
8	(striker, 1945)	16.269665
9	(ted, dibiase)	16.269665
10	(tick, tock)	16.269665

3. Evaluation

The application of NLP in this exercise aimed to generate customer insight by mining reviews text to give Amazon a commercial edge through a clear picture of the products that are popular. The first technique explored in question 3 produced a top 40 collocations list that included general expressions used in gaming and identified very few specific product names. The top 12 observed in the provided screenshot had only three actual gaming products, 'X box', 'Grand theft' and 'Super Smash'. The MI technique used in question 4 was approached in two ways; mechanically looping through the data using the MI formula, and by using NLTK library modules. The former produced results that were not informative consisting of random paired noun collocations that appeared only once. The latter avenue gave the best outcome with a top 40 mainly consisting of specific video games and platforms alongside common expressions. This approach would be ideal for identifying which products Amazon should focus its stock budget on.

4. References

Icalem, 2018. *Mutual Information.* [online]. Available from: https://lcalem.github.io/blog/2018/10/17/mutual-information [Accessed 15 February 2024]

Medium, 2019. *Feature Engineering with NLTK for NLP and Python*. [online]. Available from: https://towardsdatascience.com/feature-engineering-with-nltk-for-nlp-and-python-82f493a937a0 [Accessed 15 February 2024].

NLTK, 2023. *NLTK Documentation.* [online]. Available from: https://www.nltk.org/_modules/nltk/tokenize/regexp.html [Accessed 14 February 2024]