Literary Analysis Using Natural Language Processing

September 11, 2021

1 So You Want to Write a Short Story Masterpiece?: A Literary Analysis and Classification of Award-winning Stories by African and European Authors

1.0.1 Theme chosen: Literary Masterpiece

1.1 Background, Aims and Objectives

The Thing Around Your Neck, a collection of short stories by Chimamanda Ngozi Adichie, the award-winning Nigerian author, has unsurprisingly been called a masterpiece of African Literature with reviews such as "She makes storytelling seem as easy as birdsong" (Daily Telegraph) and "Stunning. Like all fine storytellers, she leaves us wanting more" (The Times). On the other hand, "The Grotesques" and "Mrs. Fox", two short stories by the British author, Sarah Hall, won the BBC National Short Story Award in two different years, establishing Hall as a writer of European short story masterpieces. But for writers who dream of following this path, what exactly makes a short story masterpiece, and what stories resonate with audiences of a particular region to be crowned? We will answer these questions and more (in two sections) by analysing a collection of works by both authors, Adichie and Hall, using the tools available to us.

1.1.1 In the first section, we will gain some insights into and answer the following questions (not necessarily in this order):

- What elements make up a short story masterpiece?
- Do these elements resonate more in one author's work than in another and how might the authors' environment and culture affect their writing?
- What important themes constitute a short story masterpiece?
- Based on the critical acclaim of these authors' works, what genre (specifically positive or negative) might be more widely received and does this differ by region? (We will use sentiment analysis to answer this question)

1.1.2 In the second section, we will use machine learning to aid writers in better targeting their audience or even in capturing audiences of a different region by:

- Classifying and therefore, determining what stories might be more relatable to, and thus considered to be geared towards, an African audience and what might be more relatable to, and thus considered to be geared towards, a European audience.
- Testing the accuracy of our machine model by using a test set, then stories not in our dataset, but from the same authors, and finally, stories not in our dataset and by different African and European authors.

- Critically evaluating the accuracy of our machine model and providing solutions for how our model can be improved to better serve as a tool for upcoming and established writers.

NOTE: This project uses the pronouns, "We" and "our" to keep the reader engaged. Where necessary, the pronoun defaults to "I" to explain some decisions taken. Furthermore, sources are cited inline and only sources not already mentioned are included in the references section at the end of the notebook (including learning resources used to provide further evidence of engagement with wider academic literature besides that provided in the main work). In addition, this notebook follows the PEP8 naming convention, especially for function names.

1.2 About the Authors

Chimamanda Ngozi Adichie is a Nigerian writer whose works range from novels to short stories to nonfiction. She was described in The Times Literary Supplement as "the most prominent" of a "procession of critically acclaimed young anglophone authors which is succeeding in attracting a new generation of readers to African literature", particularly in her second home, the United States.

[67]:
```
# About Sarah Hall (to be used in webscraping code block later)
about_sarah_hall = "Sarah Hall is the author of five novels - Haweswater, The␣
 ↪Electric Michelangelo (shortlisted for the Man Booker Prize), The Carhullan␣
 ↪Army, How to Paint a Dead Man and The Wolf Border. Her most recent work is␣
 ↪Madame Zero - a collection of short stories which includes Mrs Fox (the␣
 ↪story won the BBC National Short Story award in 2013)."
print(about_sarah_hall)
```

```
Sarah Hall is the author of five novels - Haweswater, The Electric Michelangelo
(shortlisted for the Man Booker Prize), The Carhullan Army, How to Paint a Dead
Man and The Wolf Border. Her most recent work is Madame Zero - a collection of
short stories which includes Mrs Fox (the story won the BBC National Short Story
award in 2013).
```

1.3 Firstly, some housekeeping: we need to import the following libraries.

[68]:
```
import nltk
import numpy as np
import matplotlib.pyplot as plt
import nltk.classify.util
import requests
import json
import math
import re
import sys  # For error handling
import os   # For operating system interaction
from bs4 import BeautifulSoup
from nltk import wordpunct_tokenize
from nltk import sent_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
```

```
from nltk.text import Text
from nltk.corpus import wordnet
from nltk.stem import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.stem import LancasterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag
nltk.download('averaged_perceptron_tagger')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
from nltk.classify import NaiveBayesClassifier
%matplotlib inline

try:
    import cPickle as pickle
except:
    import pickle
import pprint
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Brenda Akoda\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package vader_lexicon to C:\Users\Brenda
[nltk_data]     Akoda\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1.3.1 Discussing Ethics of Use limiting Project Scope

Before we begin our analysis, it is important to mention the other approach that was used before the decision to focus on the works of Sarah Hall and Chimamanda Adichie. Initially, this project was to compare the works of those who won the Commonwealth Short Story competition, particularly the winning works from Africa and Canada and Europe. These works were available on the Granta website, however, in considering the ethics of use of the data, I decided not to continue with the initial approach as the Granta website's terms and condition page (see: https://granta.com/terms/) clearly states that it is not acceptable to retrieve works from their site. Besides these terms, their website prevented access to these works via webscraping.

[69]:
```
# Get access to the data at a particular URL
def get_soup (URL, jar=None):
    if jar:
        r = requests.get(URL, cookies=jar)
    else:
        r = requests.get(URL) # Get URL
        jar = requests.cookies.RequestsCookieJar()
    print(r.url) # Display the URL of the accessed site.
    data = r.text # Get data from URL
    soup = BeautifulSoup(data, "html.parser") # Get BeautifulSoup parser.
```

```
    return soup, jar
```

Just for demonstration purposes, this is the prohibition message that we might get if trying to webscrape the Granta site.

NOTE: The function above is required to access the data at any URL specified for webscraping.

[70]:
```
soup, jar = get_soup("https://granta.com/wherever-mister-jensen-went/")
article = soup.find_all("p")
article
```

```
https://granta.com/wherever-mister-jensen-went/
```

[70]:
```
[<p>You don't have permission to access this resource.</p>]
```

Hence, the dataset was redefined to consider those works that were ethical to use for academic purposes.

1.4 Getting Sarah Hall's works

Three of Sarah Hall's works were freely and publicly available at certain sites (These sites are displayed below when we call the function get_soup). Hence, I scraped these sites to retrieve each short story and store them in a list for later data analysis.

[71]:
```
# Create a list to store Hall's 3 stories.
hall_stories = [""] * 3
```

-

1.5 Retrieving "The Grotesques" by Sarah Hall

[72]:
```
# Get the URL of the first work.
soup, jar = get_soup("https://www.theguardian.com/books/2020/oct/06/
 ↪master-of-short-story-sarah-hall-becomes-first-to-win-bbc-prize-twice")
```

```
https://www.theguardian.com/books/2020/oct/06/master-of-short-story-sarah-hall-
becomes-first-to-win-bbc-prize-twice
```

[73]:
```
# Find the sroty section
story_section = soup.find("div", {"class": "content__article-body"})
story_title1 = story_section.h2

# Append story to list.
story_body1 = story_title1.find_next_siblings("p")
for paragraph in story_body1:
    hall_stories[0] += paragraph.text + " "

# Get story title.
story_title1 = story_title1.text[0:14]
```

-

1.6 Retrieving "Wilderness" by Sarah Hall

[74]:
```
# Get the URL of the second story.
soup, jar = get_soup("https://www.theguardian.com/books/2013/mar/18/
↪wilderness-sarah-hall-short-story")
```

```
https://www.theguardian.com/books/2013/mar/18/wilderness-sarah-hall-short-story
```

[75]:
```
# Get story title.
story_title2 = soup.find("h1")
story_title2 = story_title2.text[0:10]

# Find and append story to list.
story_section = soup.find("div", {"class": "article-body-viewer-selector"})
if story_section != None:
    story_body2 = story_section.findChildren("p")
for paragraph in story_body2:
    hall_stories[1] += paragraph.text + " "

hall_stories[1] = hall_stories[1][:-70] # Strip off unrelated text towards the␣
↪end.
```

-

1.7 Retrieving "Mrs Fox" by Sarah Hall

[76]:
```
soup, jar = get_soup("https://www.toa.st/magazine/
↪mrs-fox-short-story-sarah-hall.htm")
```

```
https://www.toa.st/magazine/mrs-fox-short-story-sarah-hall.htm
```

[77]:
```
# Get story title.
story_title3 = soup.find("h1")
story_title3 = story_title3.text[0:7]

# Get story.
story_section = soup.findAll("p")
for paragraph in story_section:
    if paragraph.text == about_sarah_hall:
        break
    hall_stories[2] += paragraph.text + " "
```

1.8 Getting Chimamanda Adichie's works

Two of Chimamanda Adichie's works were freely and publicly available to use online at https://www.prospectmagazine.co.uk/author/Chimamanda-Ngozi-Adichie

The entire collection was also published on the Women of Dartmouth organisation website in pdf format: http://women.dartmouth.org/s/1353/images/gid294/editor_documents/events/new_york_book_club/chin
ngozi-adichie-the-thing-around-your.pdf?gid=294&pgid=61&sessionid=f133b40d-1b47-452e-863e-

11461097aa34&cc=1. Note that the choice of these three short stories from her short story collection, The Thing Around Your Neck, was made based on a poll of readers' favourite stories from the collection: https://www.goodreads.com/questions/848584-what-was-your-favourite-story-from-the.

That stated, for some variety and to combine different techniques, I chose to manually retrieve each story from these sites, save them in individual text files stored in a directory, and then use Python and the OS library to import all the files in that directory (using the function, import_file, below) and store them in a list for later data analysis.

[78]:
```
# Create a list to store Adichie's 3 stories.
adichie_stories = [""] * 3
```

[79]:
```
# Import, read and store the contents of a file.
def import_file (filename):
    try:
        # Open file.
        file = open(filename, mode="r", encoding="utf-8")
    except:
        # Error handling.
        print("Oops", sys.exc_info()[1], ".",
              "\nPlease confirm file path and try again.")
    else:
        # Read data from file and close after reading.
        data = file.read()
        file.close()

    return data
```

-

1.9 Retrieving "Ghosts", "The Thing Around Your Neck" and "Tomorrow Is Too Far" by Chimamanda Adichie

[80]:
```
try:
    i = 0
    # Iteratively import Adichie's stories from the specified directory.
    for filename in os.listdir("chimamanda_adichie_stories"):
        if filename.endswith('.txt'):
            adichie_stories[i] = import_file(os.path.
 ↪join("chimamanda_adichie_stories", filename))
        i += 1
except:
    pass
```

1.10 Preparing the data for analysis

Although our data has been cleaned through the procedures that we applied earlier, we still need to prepare the data for analysis by tokenising each story (i.e. chopping the story up into words),

removing punctuation marks and removing stopwords (these are words like “the”, “and”, etc, that do not give us much meaning and thus, must be removed before we can start to gain any meaningful insight).

Notice that the preprocess_data function below does not take into account words that are uppercase. This is intentional as we do not want to omit pronouns (also considered stopwords) which are important to any literary analysis as we will see later.

[81]:
```
# Tokenise and remove stopwords and punctuation marks from data.
def preprocess_data (data):
    # Split text (including at punctuations) into tokens.
    tokens = []
    tokens += wordpunct_tokenize(data)

    # Remove stopwords, punctuation marks, non-alphabetic characters (e.g. line␣
 ↪breaks, null data, etc.), and words of length 1.
    stop_words = stopwords.words("english")
    tokens = [word for word in tokens if word not in stop_words and word.
 ↪isalpha() and len(word) > 1]

    return tokens
```

[82]:
```
# Initialise containers to store tokens for numerical and context analysis.
adichie_stories_tokens = [""] * 3
hall_stories_tokens = [""] * 3

adichie_stories_text = [""] * 3
hall_stories_text = [""] * 3
```

[83]:
```
try:
    for story in range(len(adichie_stories)):
        # Prepare Adichie's and Hall's stories for data analysis.
        adichie_stories_tokens[story] = preprocess_data(adichie_stories[story])
        hall_stories_tokens[story] = preprocess_data(hall_stories[story])

        # Tokenise and convert to NLTK friendly format for further analysis␣
 ↪later.
        adichie_stories_text[story] = nltk.word_tokenize(adichie_stories[story])
        adichie_stories_text[story] = nltk.Text(adichie_stories_text[story])
        hall_stories_text[story] = nltk.word_tokenize(hall_stories[story])
        hall_stories_text[story] = nltk.Text(hall_stories_text[story])
except:
    pass
```

2 SECTION 1

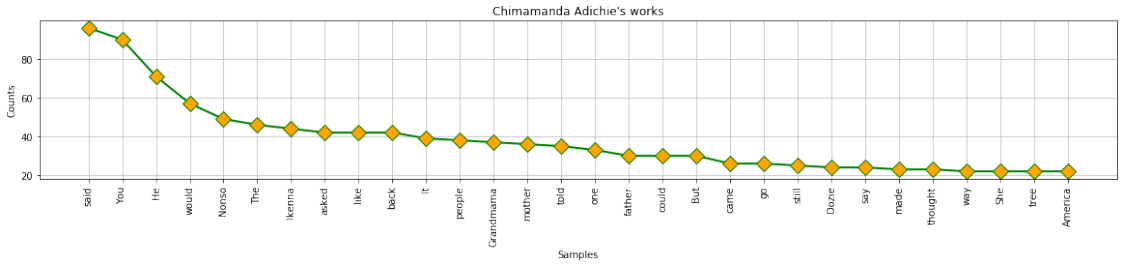
2.1 Analysing the data

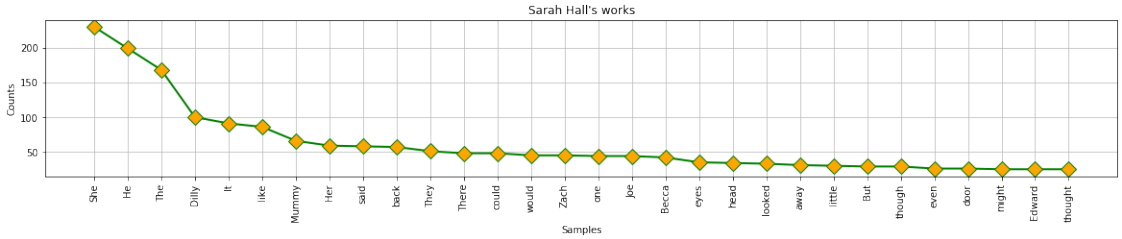
Having preprocessed our data, we may begin analysing it. Firstly, let us uncover the top 30 words from Adichie's three stories combined and Hall's three stories combined to see what insight we can gain from each author's writing.

[84]:
```
# Combine all three of Adichie's short stories.
all_adichie_stories = []
for story in adichie_stories_tokens:
    all_adichie_stories += story

# Combine all three of Hall's short stories.
all_hall_stories = []
for story in hall_stories_tokens:
    all_hall_stories += story
```

[85]:
```
def plot_line_graph(stories, wordCount, storyTitle=""):
    # Get frequency distribution of words
    freqDist = nltk.FreqDist(stories)

    # Plot frequency distribution of words
    plt.figure(figsize=(20,3))
    plt.title(storyTitle)
    freqDist.plot(wordCount, color='green', marker='D',␣
 ↪markerfacecolor='orange', markersize=12)
```

[86]:
```
# Plot a line graph showcasing the top 30 words used by each author in their␣
 ↪works.
plot_line_graph(all_adichie_stories, 30, "Chimamanda Adichie's works")
plot_line_graph(all_hall_stories, 30, "Sarah Hall's works")
```





2.2 FIRST INSIGHT: A UNIQUE POINT OF VIEW IS PREFERABLE WHEN CREATING A SHORT STORY MASTERPIECE

Often times, several short story authors narrate from a first person point of view, creating a perception that short stories are only successful when written from a first person point of view. While some of Adichie's stories in her short story collection are narrated from a first person perspective, not all narrate from this point of view. In fact, contrary to this perception, we can certainly view a pattern that cuts across these three works of Adichie and Hall: the use of the third person point of view, "She" and "He". Considering that two of Hall's works chosen for this analysis won the BBC National Short Story award, this could be an indication that perhaps:

2.2.1 A short story is more likely to be considered a masterpiece when narrated from a point of view that is different from the norm for stories in that genre.

This opinion is supported by Adichie's unconventional use of the second point of view in her story, "The Thing Around Your Neck" even as the protagonist is female and the story is told through her lens.

2.2.2 Observation:

It is interesting to observe that while both Adichie's and Hall's stories are female-focused as expressed in their synopses, they all seem to have an abundance of "He" pronouns, almost as much as "She" pronouns. To uncover exactly how many times the pronouns "He" and "She" appear in Hall's and Adichie's works, let us carry out a word count.

Note: We will add the count of the pronoun "You" in Adichie's work to the count of "She" (since the protagonist is female) to get an accurate representation.

[87]:
```
# Get count of She, He and You in Adichie's stories.
she_count_adichie = all_adichie_stories.count("She")
he_count_adichie = all_adichie_stories.count("He")
you_count_adichie = all_adichie_stories.count("You")

# Get count of She and He in Hall's stories.
she_count_hall = all_hall_stories.count("She")
he_count_hall = all_hall_stories.count("He")
```

```
print("In Adichie's stories, 'She' appears", she_count_adichie +␣
 ↪you_count_adichie, "times, while 'He' appears", he_count_adichie, "times.␣
 ↪Thus, 'She' appears", she_count_adichie + you_count_adichie -␣
 ↪he_count_adichie, "more times than 'He'. \n")
print("In Hall's stories, 'She' appears", she_count_hall, "times, while 'He'␣
 ↪appears", he_count_hall, "times. Thus, 'She' appears", she_count_hall -␣
 ↪he_count_hall, "more times than 'He'.")
```

```
In Adichie's stories, 'She' appears 112 times, while 'He' appears 71 times.
Thus, 'She' appears 41 more times than 'He'.

In Hall's stories, 'She' appears 230 times, while 'He' appears 199 times. Thus,
'She' appears 31 more times than 'He'.
```

The count of the words above certainly confirm that there are almost as many counts of 'He' as there are of 'She'. To be precise, 'She' appears only 41 and 31 times more than 'He' in Adichie's and Hall's works respectively.

This observation may indicate the authors' efforts to provide a balanced narrative in as much as the stories are female-focused or it may simply be reflective of the characters'–and by extension, the authors'–society, where stories supposedly focused on females still revolve around the men in their (i.e. these females') lives.

2.2.3 Formatting text to lowercase and removing pronouns

While we gained our first insight and observation by permitting capitalised words, particularly pronouns, we would most likely gain even more insight if we strip our stories of these words before further examination.

Note that in addition to the list of stopwords provided in the NLTK library, we will also check words against a more complete list as compiled by Zoheb Abai (https://gist.github.com/sebleier/554280#gistcomment-3431590) and provided at this link: https://gist.githubusercontent.com/ZohebAbai/513218c3468130eacff6481f424e4e64/raw/b70776f341a148293ff277a

[88]:
```
# Open and store contents of stopwords file.
gist_file = open("gist_stopwords.txt", "r")

try:
    content = gist_file.read()
    # Format contents of file.
    more_stop_words = content.split(",")
    more_stop_words= [i.replace('"',"").strip() for i in more_stop_words]
finally:
    # Close file.
    gist_file.close()
```

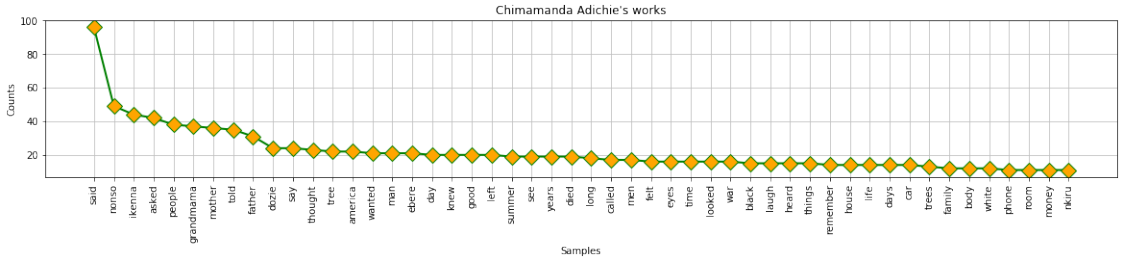
[89]:
```
stop_words = stopwords.words("english")

# Make all words lowercase.
```

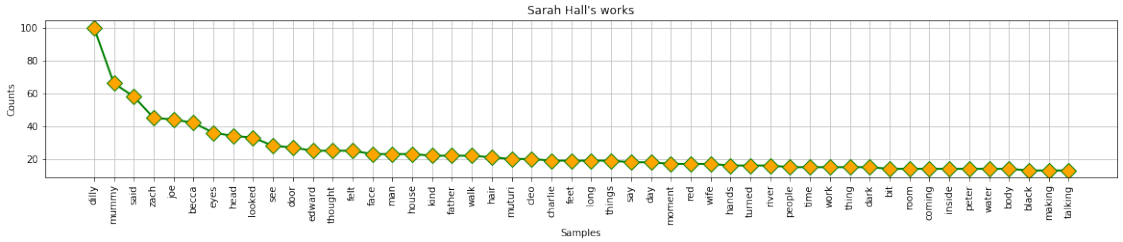
```
cleaned_adichie_stories = [word.lower() for word in all_adichie_stories]
cleaned_hall_stories = [word.lower() for word in all_hall_stories]

# Remove all remaining stopwords, including pronouns, using NLTK stopwords and␣
 ↪the additional stopwords list.
cleaned_adichie_stories = [word for word in cleaned_adichie_stories if word not␣
 ↪in stop_words and word not in more_stop_words and word.isalpha() and␣
 ↪len(word) > 2]
cleaned_hall_stories = [word for word in cleaned_hall_stories if word not in␣
 ↪stop_words and word not in more_stop_words and word.isalpha() and len(word)␣
 ↪> 2]
```

Having completely cleaned our data, we may now proceed to make further analysis.

Firstly, let us plot the line graph again for both authors' works.

[90]:
```
# Plot a line graph showcasing the top 50 words used by each author in their␣
 ↪works.
plot_line_graph(cleaned_adichie_stories, 50, "Chimamanda Adichie's works")
plot_line_graph(cleaned_hall_stories, 50, "Sarah Hall's works")
```





2.3 SECOND INSIGHT: BUILD A WORLD AROUND YOUR MAIN CHARACTER

From the graph of word frequencies above, it is easy to spot an abundance of names or reference to people. In Adichie's works, these names or references include Nonso, Ikenna, Grandmama, Mother, Father, Dozie, and Ebere. In Hall's works, these names or references include Dilly, Mummy, Zach,

Joe, Becca, Edward, Father, Muturi, Cleo, Charlie, Wife. Both authors' works also feature words such as house and people. Hence, these frequencies of names and references to names imply that:

2.3.1 Our main character is not an island, they need to be connected to other people to move the story and thus, building these people is just as important as building the main character.

Now, the question of what kind of characters to build around your main character seem to differ for both authors. This difference seems to be where Adichie's and Hall's story-telling diverge to become a reflection of the individual authors' cultures and societies. As the names in Adichie's works seem mostly related to family (as seen from the frequencies of such words), we will use the Text module in the NLTK library to uncover every instance of the word "Family" in both Adichie's and Hall's works and the context in which it is used.

[91]:
```
print("ADICHIE'S USE OF THE WORD, 'Family'\n")
for story in adichie_stories_text:
    # Convert tokens to NLTK friendly text format.
    story = Text(story)
    # Display every instance of the word and the context in which it is used.
    story.concordance("family")
    print("\n")
```

```
ADICHIE'S USE OF THE WORD, 'Family'

Displaying 3 of 3 matches:
t in Nigeria to see your father ' s family , especially Grandmama . You rememb
si name now , who would protect the family lineage . The neighbors came over w
 neck . He looked as if he were not family , as if he were one of the guests w


Displaying 5 of 5 matches:
" I asked . I was quite shaken . My family and I saw him on the day he died ,
ince ? " I asked . " Yes . My whole family was in Orlu when they bombed it . N
en we took her to the Staff Club on Family Day , because , he said , she was t
th one ' s eyes . " What about your family ? " I asked . " I never remarried.
ity of sand thrown on broken men by family members suspended between disbelief


Displaying 4 of 4 matches:
ho had put in the names of all your family members for the American visa lotte
d hide or should reveal only to the family members who wished them well . You
 he chanted . If you sell me and my family , you can not buy even one tire on
ticket , to go with you to see your family . You said no , you needed to go al
```

2.3.2 Family (nuclear and extended) remains a relatable and near indispensable element in African story-telling

As seen in the above articularly for Adichie, the word "family" is mentioned several times in all three works. Where family is mentioned, we get a sense of the importance of being close to family and preserving the family lineage in an African society. For instance, there are many mentions of seeing family, going with family or being with family. Furthermore, with words such as "Grandmama" and phrases such as "whole family" and "all your family members", there is the sense that family in this society automatically includes the extended family. The sacredness of family is also reflected in references to "Family Day" and the need to "protect the family lineage". In fact, the family is so intertwined with the individual such that an individual is nothing without a family (see: line 3 of 5 matches: "What about your family?").

[92]:
```
print("HALL'S USE OF THE WORD, 'Family'\n")
for story in hall_stories_text:
    # Convert tokens to NLTK friendly text format.
    story = Text(story)
    # Display every instance of the word and the context in which it is used.
    story.concordance("family")
    print("\n")
```

```
HALL'S USE OF THE WORD, 'Family'

Displaying 3 of 3 matches:
ut boundaries and identity within a family , he ' d used a fishing-net metapho
orst things to have happened to the family , her attachment , her over-attachm
 , wheaty , that safe , wonderful , family taste . Merrick had been wrong . Sh


Displaying 1 of 1 matches:
the burnout with the band , and the family arguments , she missed England - sh


no matches
```

On the other hand, in Sarah Hall's works, we see a fondness of family (with phrases like "safe, wonderful, family taste" and mentions of missing "family arguments") in two out of the three works, however these mentions (only 4 matches) differ from Adichie's references to family in two main ways: 1. Some of the phrases seem to reveal that the characters are currently apart from their families and there is no pressure to be with family. 2. There is no possessiveness when referring to family; that is, while Adichie uses "my family", "your family", etc, Hall uses the more casual phrases, "the family" and "a family". In other words, the family in the character's–and perhaps, by extension, in Hall's–society does not necessarily define the individual and there is no pressure to continue some family lineage. That stated, where characters are defined by family, that family seems to mean only a few central family figures like the mother as there is mention of "Mummy" 66 times, but only in one work (see 25 instances of "Mummy" printed below). There is also mention of "Wife" so the idea of family is certainly restricted to the nuclear and not extended.

In addition and going back to reconsider the mentions of "Mummy" in Hall's work (as printed below), the context in which "Mummy" is used indicates that the protagonist and her world view is strongly defined by her mother's views and perspectives. For instance, The protagonist usually confirms right or wrong by thinking about what "Mummy" would say or do (see 25 out of 66 instances below). This discovery, combined with what we have already seen in Adichie's works concerning the importance of family and the high frequency of the word "mother", certainly reveals that, more often than not:

2.3.3 No matter the society, mothers play an important role in shaping a character and that character's world view.

[93]:
```
print("HALL'S USE OF THE WORD, 'Mummy'\n")
for story in hall_stories_text:
    # Convert tokens to NLTK friendly text format.
    story = Text(story)
    # Display every instance of the word and the context in which it is used.
    story.concordance("mummy")
    print("\n")
```

```
HALL'S USE OF THE WORD, 'Mummy'

Displaying 25 of 66 matches:
top ; she was late getting home with Mummy ' s shopping . But the scene was too
. She could hear an internal voice , Mummy ' s voice : disgraceful , who are th
Edward had seen and reported back to Mummy , who was outraged and still talking
otchy smell . Silly girl , she heard Mummy say . Don ' t be so squeamish . Mumm
ummy say . Don ' t be so squeamish . Mummy was right , of course . She usually
ar up . Dilly sometimes thought that Mummy was like a truffle pig , rooting aro
 goodness ' sake . Engage ! By now , Mummy would have swept the degrading parod
t she was probably very late now and Mummy would be getting cross . Mummy had o
w and Mummy would be getting cross . Mummy had only sent Dilly out for a few it
e jam . Dilly couldn ' t remember if Mummy had asked for a particular kind , an
g her mind . Father Muturi , who was Mummy ' s favourite priest at St Eligius ,
k , and hadn ' t wanted to sing when Mummy had asked him to . When Dilly had su
eir French evening class this week . Mummy was making scones for the tea party
baby . And because of Peter , though Mummy maintained Peter had done nothing wr
t be spoken about , unless raised by Mummy , and then certain agreements were m
s , be witty but still seem humble - Mummy and Cleo were masters at that kind o
he water ' s curtain . It was one of Mummy ' s peeves , all the junk being toss
as it . An egg for breakfast was all Mummy had allowed , no toast because Dilly
n a lot of frustration in the room . Mummy and the lady , her name was possibly
 again . That was not uncommon with Mummy ' s acquaintances . One of the cakes
Dilly . Soon there would be scones , Mummy ' s speciality : warm , soft , comfo
ntention for a Nobel , people said . Mummy maintained Charlie-bo was from a sma
 the bottom of the hill . Recently , Mummy had arranged a session with Merrick
 It was strange seeing him away from Mummy ' s parties , where he was usually d
able and was glad when it was over . Mummy hadn ' t asked her about the session
```

```
no matches
```

```
no matches
```

It is important to note that I did not mention the frequency of the word, "father" in Hall's works being related to family as I did with "Mummy". This is because the father mentioned in Hall's works is actually a priest, not a regular father. To confirm this statement, we can derive the common contexts of the word, "father", (i.e. view which words are often used with "father").

[94]:
```
for story in hall_stories_text:
    story = Text(story)
    story.common_contexts(["father"])
```

```
those_muturi ._muturi saw_muturi or_muturi and_muturi their_with
their_had ?_muturi ,_muturi of_muturi
('The following word(s) were not found:', 'father')
('The following word(s) were not found:', 'father')
```

The underscore in the output above indicates where the word, "father" normally appears. As we can observe, "Muturi" comes right after "father". Hence, the father mentioned in one of Hall's works (as the others have no such mentions) is called "Father Muturi".

2.4 Applying Three Approaches of Stemming and Lemmatization to variations of a word to derive its root

From the line graph as earlier shown, we can observe the high frequency of the words, "said", "asked", "told", "say", and "called" in Adichie's works and also the high frequency of the words, "said" and "say" in Hall's works. Particularly in Hall's works, "said" appears as the third most frequent word while "say" appears as the 28th most frequent word. Now, as humans, we know that "said" and "say" mean exactly the same thing so in fact, there is a much higher frequency of the word, "say" than our graph is displaying, but how do we manipulate our data to consider all variations of a word and more accurately reflect its frequency?

To tackle this issue, we will explore different approaches using the words, "said" and "say" in Hall's works (and add the word "says" for better comparison) as test data, before formulating a final approach to be used on the whole dataset.

[95]:
```
test_data = ["said", "say", "says"]
```

Firstly, we will consider Porter's Stemmer which is the most often used stemmer to derive a root stem, though it is more computationally expensive:

[96]:
```
porter_stemmer = PorterStemmer()
stemmed_with_porter = [porter_stemmer.stem(word) for word in test_data]
print(stemmed_with_porter)
```

```
['said', 'say', 'say']
```

As we can see from the above output, while Porter Stemmer stems the word "says" to the root stem, "say", it fails to recognise that "said" and "say" are variations of the same word. Hence, we must consider a different approach to solving our problem.

Our second approach is to consider Snowball Stemmer: Note: For some context, although Snowball Stemmer is beyond the scope of the course, it is actually also called Porter 2 as it is an improvement over Porter Stemmer in terms of both stemming and computational time taken. Also note that when calling Snowball Stemmer, we must pass the language that we want to use.

[97]:
```
snowball_stemmer = SnowballStemmer("english")
stemmed_with_snowball = [snowball_stemmer.stem(word) for word in test_data]
print(stemmed_with_snowball)
```

```
['said', 'say', 'say']
```

As we can see from the above output, though Snowball Stemmer is considered to be an improvement over Porter Stemmer and it also stems the word "says" to "say", it still does not solve our problem of recognising the word, "said" as a variations of the word, "say". Hence, we must still consider a different approach.

Our third approach is to consider the Lancaster Stemmer which is a lot more aggressive at stemming than the other two stemmers and it is also the fastest algorithm of the three stemmers.

[98]:
```
lancaster_stemmer = LancasterStemmer()
stemmed_with_lanc = [lancaster_stemmer.stem(word) for word in test_data]
print(stemmed_with_lanc)
```

```
['said', 'say', 'say']
```

As seen from the above, in as much as Lancaster Stemmer is a very aggressive stemmer, it still does not stem the word, "said" to derive the root stem, "say".

At this point, it is important to reconsider the problem. Perhaps the word "said" is too different in spelling from the other two words for the stemmers to recognise it as a variation of "say" and therefore, to stem it properly. Perhaps what we need to do instead to solve our problem is to derive, not the stem of the words, but the meaning of the words. Luckily, this is exactly what a lemmatizer does and through the wordnet module in the NLTK library, we can use a lemmatizer to derive a lemma (or root word) of the words.

[99]:
```
lemmatizer = WordNetLemmatizer()
lemmatized_test_data = [lemmatizer.lemmatize(word) for word in test_data]
print(lemmatized_test_data)
```

```
['said', 'say', 'say']
```

Interestingly, the lemmatizer fails to solve our problem. However, this is likely because the default part of speech used by the lemmatizer to lemmatize words is the noun. Hence, we need to indicate the part of speech for each of our words (in this case, "v" for verb).

[100]:
```
lemmatized_test_data = [lemmatizer.lemmatize(word, "v") for word in test_data]
print(lemmatized_test_data)
```

```
['say', 'say', 'say']
```

Finally, the lemmatizer solves our problem, but only when we explicitly indicate the part of speech of each word; this means that we have to pass the part of speech of each word in our dataset to the lemmatizer to receive accurate results.

Luckily, we have NLTK Part of Speech (POS) tagging which is highly effective in discerning what part of speech applies to a word in a given context, even if that word is spelt the same. For instance, in the example below, "play" in the first list is correctly tagged as a noun (i.e. NN) while "play" in the second list is correctly tagged as a verb (i.e. VBP).

[101]:
```
nltk_noun = ["A", "play"]
nltk_verb = ["I", "play", "the", "drums"]
print(nltk.pos_tag(nltk_noun))
print(nltk.pos_tag(nltk_verb))
```

```
[('A', 'DT'), ('play', 'NN')]
[('I', 'PRP'), ('play', 'VBP'), ('the', 'DT'), ('drums', 'NNS')]
```

2.5 Using NLTK Part of Speech (POS) Tagging and Wordnet Lemmatizer for more accurate analysis of similar words

That stated, we cannot directly pass the NLTK parts of speech tags to the lemmatizer as it only recognises tags defined by its wordnet module. Hence, for a robust solution to this problem, we can carry out the following: - Use NLTK POS tagging to tag the part of speech of each word/token in our dataset; - Convert all NLTK tagged words to wordnet parts of speech tags; - Apply the results (tuples of word and wordnet tag) to the lemmatizer to lemmatize.

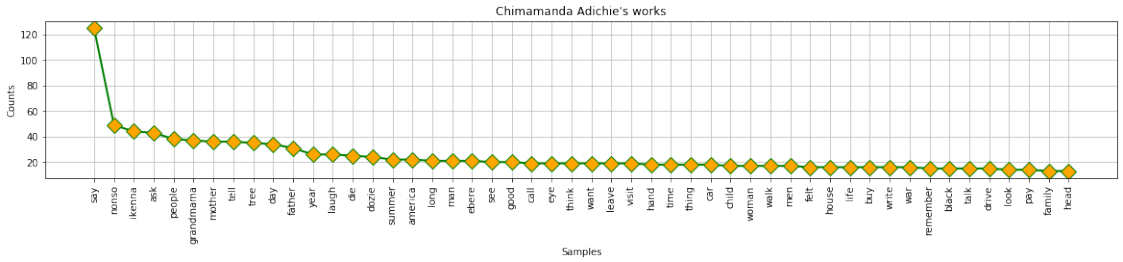
[102]:
```
# Source: Gaurav Gupta, https://medium.com/@gaurav5430/
 ↪using-nltk-for-lemmatizing-sentences-c1bfff963258

# Function to convert NLTK POS tag to wordnet tag
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None
```

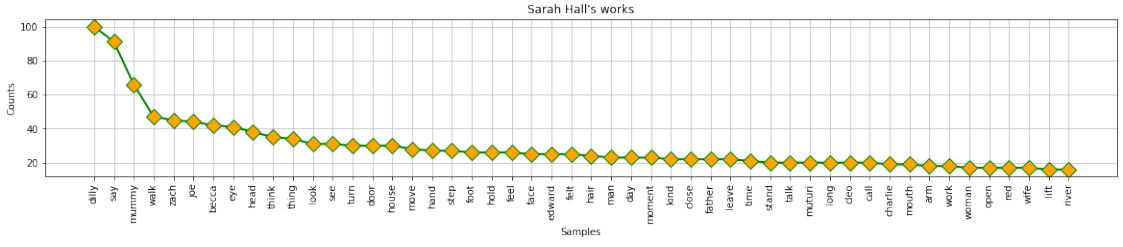
[103]:
```
# Code adapted from Gaurav Gupta, https://medium.com/@gaurav5430/
 ↪using-nltk-for-lemmatizing-sentences-c1bfff963258

# Lemmatize words in the provided list of words.
def lemmatize_words(words_list):
    # Tag each word in the provided list of words.
    nltk_tagged = nltk.pos_tag(words_list)
```

```
    # Convert NLTK tags to wordnet tags and create tuples of word and wordnet␣
↪tag.
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])),␣
↪nltk_tagged)
    lemmatized_words_list = []
    for word, tag in wordnet_tagged:
        if tag is None:
            # If there is no available tag, append the word as it is.
            lemmatized_words_list.append(word)
        else:
            # Otherwise, use the tag to lemmatize the word.
            lemmatized_words_list.append(lemmatizer.lemmatize(word, tag))
    return lemmatized_words_list
```

Having established functions that would enable us to complete the steps mentioned earlier and allows reusability, we may now apply these functions to lemmatize all the words in our datasets and then plot the output for a more accurate representation of the word frequencies.

[104]:
```
# Lemmatize all the words in Adichie's and Hall's works.
lemmatized_adichie_stories = lemmatize_words(cleaned_adichie_stories)
lemmatized_hall_stories = lemmatize_words(cleaned_hall_stories)

# Plot a line graph showcasing the top 50 words used by each author in their␣
↪works.
plot_line_graph(lemmatized_adichie_stories, 50, "Chimamanda Adichie's works")
plot_line_graph(lemmatized_hall_stories, 50, "Sarah Hall's works")
```





2.6 THIRD INSIGHT: DIALOGUE IS IMPORTANT IN THE CREATION OF A SHORT STORY MASTERPIECE

Upon exploring the different stemmers available in the NLTK library and finally choosing to combine the NLTK Parts of Speech (POS) tagging with Lemmatization to derive root words, we can more obviously notice from the above graph that "say" is the Number 1 and Number 2 most commonly used word in Adichie's and Hall's stories respectively. Other words like "ask" and "tell" are also in the top 10 most common words used in Adichie's works. With this observation, we can now clearly appreciate our third insight concerning how authors can produce a short story masterpiece: It is not enough to build a world around the main character, we must use dialogue to inform readers about this world, specifically about the characters, their personalities, actions, mindsets and how these affect our main character and perhaps ultimately leads to some end goal. Hence, in a more general sense:

2.6.1 Characters need to interact with one another to clue readers into what is going on and drive the story forward.

2.7 FOURTH INSIGHT: WHAT IS LEFT UNSAID IS SOMETIMES MORE POWERFUL THAN WHAT IS SAID

From the graph, we can see an abundance of the words, "see", "eye", "look", "feel", "felt" and "think". Indeed, as readers, we learn more about a character, not from what is being said, but rather from what is left unsaid–from their movements, particularly their gaze (i.e. as seen through the frequency of words such as "eye", "see", "look") and from how they really feel and think (i.e. as seen through the frequency of words such as "feel", "felt" and "think"). These movements, thoughts and feelings are most times only divulged to the reader to provide more insight into our character's psyche and thus, draw readers in and keep them engaged. Hence, Adichie and Hall show us that in writing a short story masterpiece, ### How your characters feel and move are just as important as what they say

2.8 Limitations of the POS Tagging and Lemmatizer technique used

As an aside, notice that while our technique of pos tagging and lemmatizing was mostly effective, it still had its limitations. For instance, the word, "felt" remained unchanged in the process and appears differently from the word, "feel" even though they are the same.

2.9 Sentiment Analysis of Adichie's and Hall Works

To gain our final insight, it might be beneficial to uncover what kind of stories move readers such that these stories are crowned masterpieces. Is a happy story more likely to attract critical acclaim or is a depressing one or is it a bit of both? Luckily, we can discover which one using sentiment analysis to get the polarity scores of each work (i.e. whether a work is negative, positive or neutral). As the sentiment analyzer takes in a string, we will have to use our original list of short stories for this task. The steps that we will take are as follows: * Tokenise each author's story by sentence: This will provide more accuracy by analysing each of the sentences and scoring each in terms of how positive, negative and neutral it is. A compound score is also provided that may have either negative or positive values indicating the overall sentiment of the sentence. * Aggregate the data to obtain a cumulative negative, positive, neutral and compound score for each of the stories: As we do not want to output hundreds or thousands of lines, we will add up the scores to determine the

overall sentiment of each story. * Visualise the data: As data is better understood when visualised, we will use a variety of techniques to visualise our data and then choose a final approach.

To accomplish the above steps, we will take an object-oriented approach to apply good practice (however, without initialising classes to avoid huge code blocks). The functions involved are written below with some comments for clarity concerning what the function does.

NOTE: I am using the VADER sentiment analyser instead of the TextBlob sentiment analysis. This is because as I discovered in my attempts to obtain the polarity scores of the texts, VADER works better as it is more accurate. From further research, I then confirmed that TextBlob is actually more suited to formal language. Hence, it is better to use VADER in this case.

[105]:
```
# Get the sentiment analyzer object.
sentimentAnalyzer = SentimentIntensityAnalyzer()
```

[106]:
```
# Function to get and cummulate polarity scores of all sentences in a story.
def get_polarity_scores(sentences):
    cummulative_polarity_scores = dict()
    for sentence in sentences:
        # Get the polarity score of the sentence.
        polarity_scores = sentimentAnalyzer.polarity_scores(sentence)
        for polarity in polarity_scores:
            if polarity not in cummulative_polarity_scores:
                cummulative_polarity_scores[polarity] =␣
 ↪polarity_scores[polarity]
            else:
                # Add up scores for each polarity.
                cummulative_polarity_scores[polarity] +=␣
 ↪polarity_scores[polarity]
    return cummulative_polarity_scores
```

NOTE: The function below, set_additional_bar_params, was initially part of the plot_polarity_bar_graph function, but separated to avoid huge code blocks.

[107]:
```
# Function to set additional parameters to bar charts.
def set_additional_bar_params(polarity_map, polarity_type, polarities):
        if polarity_map["compound"] < 0:
            # Colour compound bar red if its score is negative.
            plt.bar(3, polarity_map["compound"], color="#8b0000")
        else:
            # Colour compound bar green if its score is positive.
            plt.bar(3, polarity_map["compound"], color="#089000")
        plt.xlabel("Polarities")
        plt.ylabel("Scores")
        plt.xticks(polarity_type, polarities)
        plt.show()
```

[108]:
```
# Function to plot a bar graph displaying the cummulative polarity scores of␣
 ↪each story.
```

```
def plot_polarity_bar_graph(polarity_map):
    polarities = []
    scores = []
    for polarity in polarity_map:
        # Remove "neu" value to better compare between "pos" and "neg".
        if polarity != "neu":
            polarities.append(polarity)
            scores.append(polarity_map[polarity])
    # Change polarities from strings to numerical values to ensure display.
    polarity_type = np.arange(1, len(polarities) + 1, 1)

    plt.bar(polarity_type, scores, align="center", edgecolor="yellow",␣
 ↪facecolor="#003366")
    set_additional_bar_params(polarity_map, polarity_type, polarities)
```

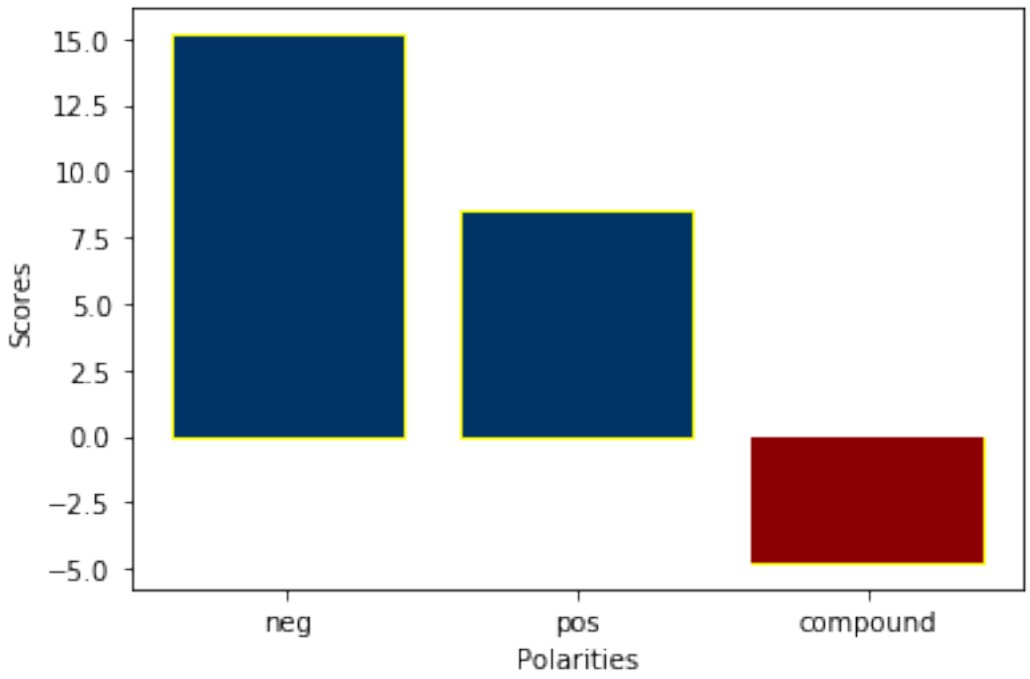
[109]:
```
# Initialisation function to start the sentiment analysis and interact with␣
 ↪other functions.
def start_sentiment_analysis(story):
    sentences_list = sent_tokenize(story)
    story_polarity_scores = get_polarity_scores(sentences_list)
    plot_polarity_bar_graph(story_polarity_scores)
    print("Absolute Difference between Negative and Positive Polarity Scores:␣
 ↪", abs(story_polarity_scores["neg"] - story_polarity_scores["pos"]), "\n")
    print("Neutral Polarity Score: ", story_polarity_scores["neu"], "\n\n")
```

Having written the functions needed for the sentiment analysis, we may now call the start_sentiment_analysis function to perform the analysis. Note that at this point, we do not have access to the other functions that calculate the polarity scores and plot the graph, we only have access to the entry function which will then handle the other operations. This approach fulfills the encapsulation and abstraction principles of Object-Oriented Programming as we hide the other functions away and abstract the analysis process such that we only have to call one function (essentially one line of code) without the need to understand its inner workings.

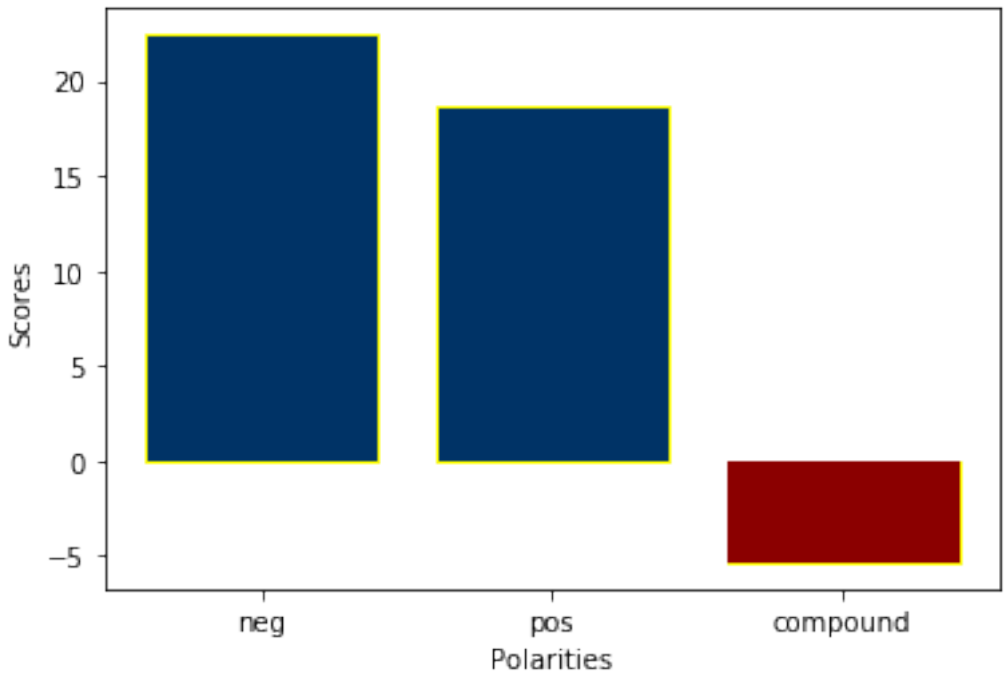
That explained, let us apply our sentiment analyser to all three of Adichie's works.

[110]:
```
print("\nPOLARITY SCORES OF ADICHIE'S WORKS \n")
for story in adichie_stories:
    start_sentiment_analysis(story)
```

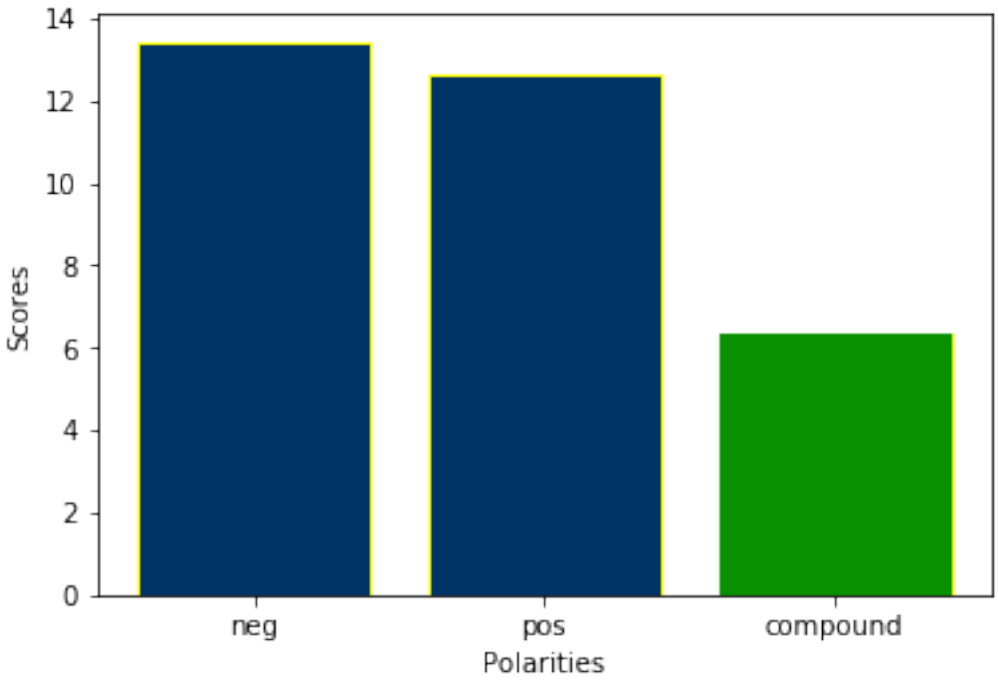
```
POLARITY SCORES OF ADICHIE'S WORKS
```




```
Absolute Difference between Negative and Positive Polarity Scores:
6.655999999999999

Neutral Polarity Score:  128.33800000000002
```




```
Absolute Difference between Negative and Positive Polarity Scores:
3.8049999999999997

Neutral Polarity Score:  258.9369999999999
```




```
Absolute Difference between Negative and Positive Polarity Scores:
0.7990000000000066

Neutral Polarity Score:  172.93499999999997
```

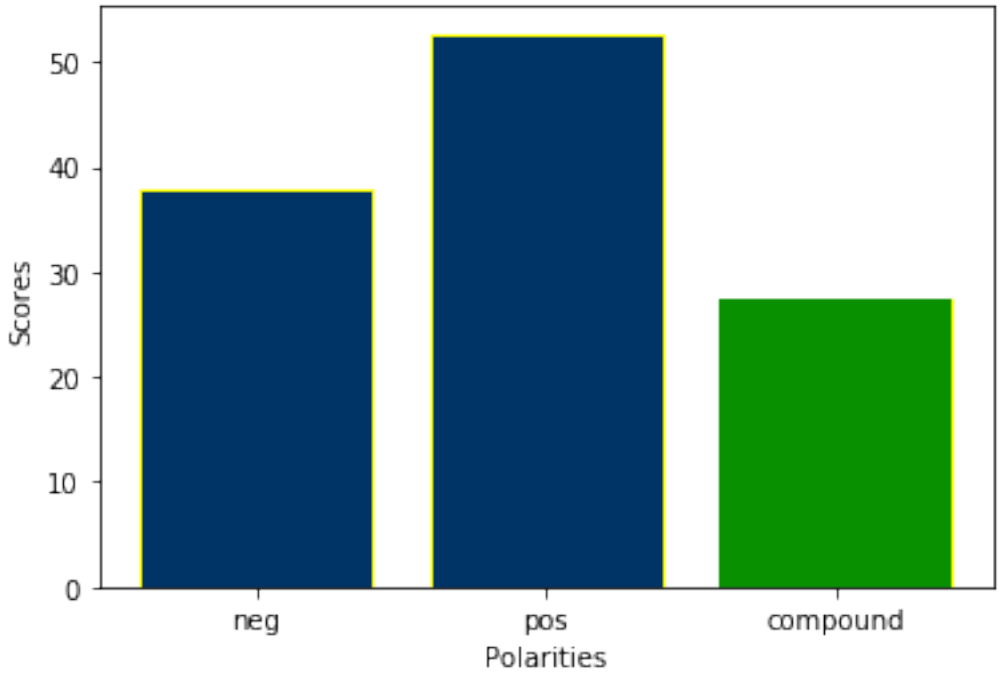
NOTE: The compound score indicates the overall sentiment of the story–with green being positive and red being negative.

From the above graphs, two out of three of Adichie's works are negative (or in other words, depressing). This discovery actually comes as no surprise with story titles such as "Ghosts" and "The Thing Around Your Neck" that already inform humans about the negative nature of their content. Notice though that the absolute difference between the negative and positive polarity scores is not much, indicating that while the stories were determined negative or positive, they were–to a large extent–almost balanced.

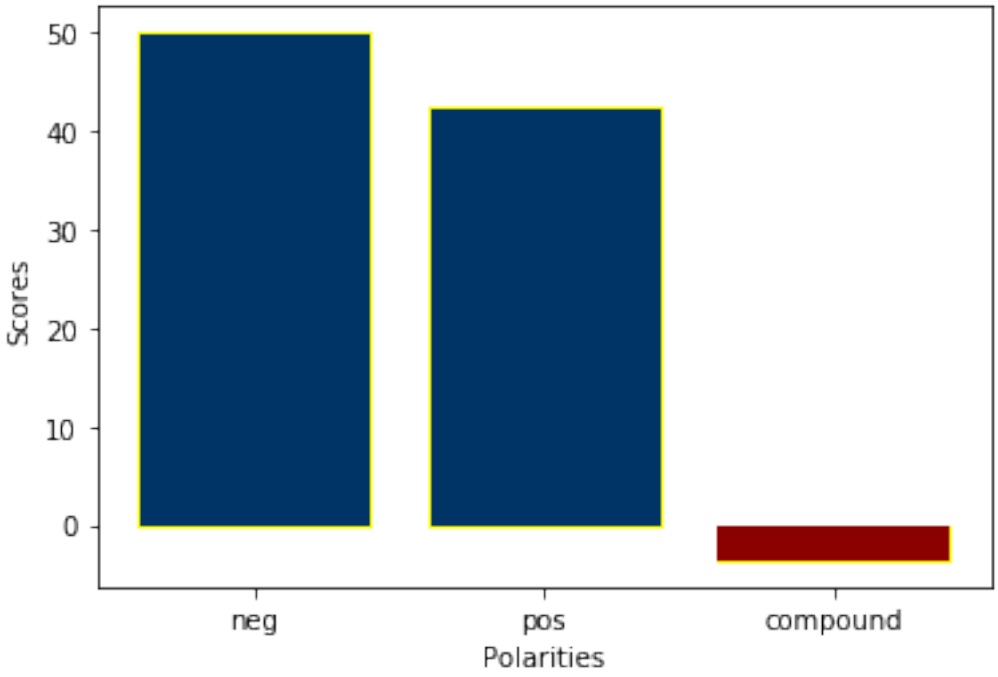
Let us now analyse all three of Hall's works to determine their polarity scores.

[111]:
```
print("\nPOLARITY SCORES OF HALL'S WORKS \n")
for story in hall_stories:
    start_sentiment_analysis(story)
```

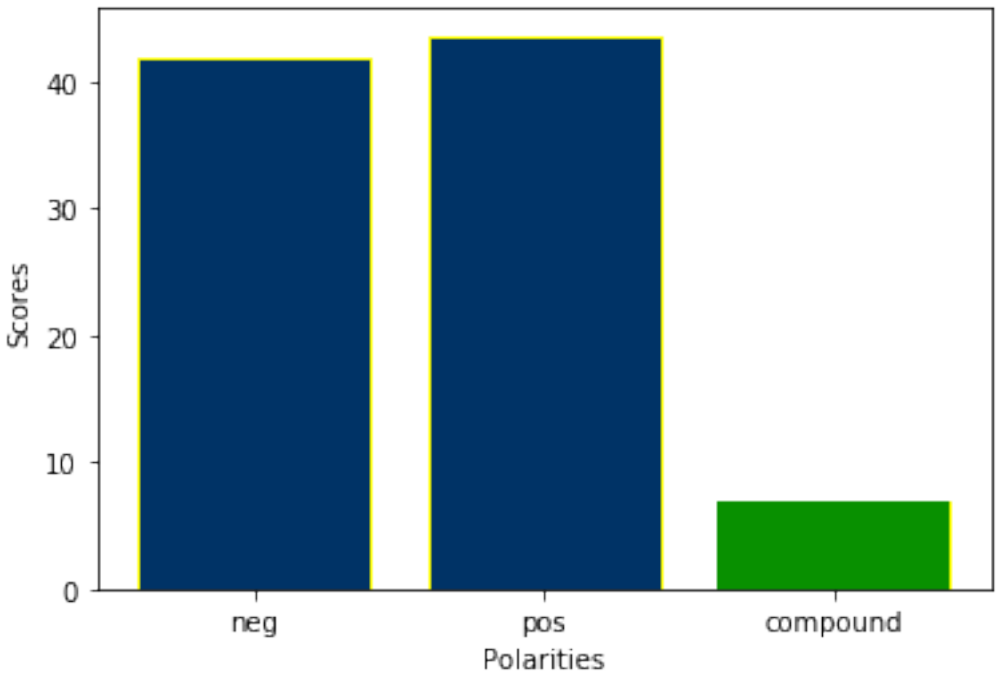
```
POLARITY SCORES OF HALL'S WORKS
```




```
Absolute Difference between Negative and Positive Polarity Scores:
14.861999999999988

Neutral Polarity Score:  453.50399999999985
```




```
Absolute Difference between Negative and Positive Polarity Scores:
7.531999999999975

Neutral Polarity Score:  511.63399999999984
```




```
Absolute Difference between Negative and Positive Polarity Scores:
1.6570000000000107

Neutral Polarity Score:  560.3940000000001
```

From the graphs above, two out of three of Hall's works have positive themes.

Similar to our observation with Adichie's works, notice that–except in the first work–the absolute difference between the negative and positive polarity scores is not much, also indicating that while the stories were determined negative or positive, they were–to a large extent–almost balanced.

2.10 FIFTH INSIGHT: POSITIVE OR NEGATIVE THEMES DO NOT MATTER, TELLING A BALANCED STORY DOES

As already observed from the previous graphs, considering the works of both authors combined, three out of the six works are positive and the other three are negative. Hence:

2.10.1 It does not matter whether a short story has an overall positive or negative theme.

However, it seems to be the balance in these stories, and not the theme, that make the stories masterpieces. Even with their overall negative or positive themes, these stories manage to walk a fine line between positivity and negativity, finding balance with their sentences ranging from positive to negative, and to largely neutral. Hence, when creating a short story masterpiece, write some positivity in the negativity and some negativity in the positivity. Put another way:

2.10.2 A character's world cannot be too happy and it cannot be too sad; it's a little bit of both.

3 SECTION 2

3.1 Using Machine Learning to identify stories relatable to audiences of a particular region

We have, thus far, uncovered some important elements that make a short story masterpiece, no matter the region. We have also uncovered some elements or aspects of some elements that resonate with one region more than they do with another. Hence, writers can be mindful of these distinctions when writing for a particular audience (in this case, either African or European) in order to ensure that the story created is relatable to the audience that it was written for.

In the age of Artificial Intelligence, rather than spending hours, days or months on reading and analysing texts by African and European authors, we can use machine learning to easily make the distinction between stories that are more relatable to an African audience vs a European audience. As mentioned earlier in the introduction, this project was initially to use stories by different authors to train a machine model, however, due to considerations about ethics of use, the dataset became limited to three of Adichie's and Hall's stories. Although renowned, these authors, by no means, represent their entire continents nor do their works speak for the wealth of short story literature in their regions as different authors, even in the same region, have different styles of writing. Nonetheless, this project serves as a first model in a series of models and research work required to help writers better target their audience (in this case, limited to Africa and Europe, where Europe in this context includes Britain) or capture new audiences.

- To begin, we will use Naive Bayes Classifier to train our model. As training a machine model, especially using the Naive Bayes Classifier can take a rather long time, I created functions below to save the classifier as a file after the training process and load the classifier for subsequent use. To do so, both functions uses the Pickle module to serialise our classifier object and enable us to load the classifier a lot more quickly than the first time that it was trained.

[112]:
```
# Function to save classifier as a file for faster classification later.
def save_classifier(classifier, filename):
    file = open(filename, 'wb')
    pickle.dump(classifier, file, -1)
    file.close()
```

[113]:
```
# Function to load the classifier when called for subsequent classifications.
def load_classifier(filename):
    file = open(filename, 'rb')
    classifier = pickle.load(file)
    file.close()
    return classifier
```

- Next, we will tokenise all three of Adichie's stories by sentence and add all those sentences to one big list. We will also do the same for Hall's stories.

[114]:
```
all_adichie_sentences = []
for i in range(len(adichie_stories)):
    sents_in_story = sent_tokenize(adichie_stories[i])
    all_adichie_sentences.extend(sents_in_story)

all_hall_sentences = []
for i in range(len(hall_stories)):
    sents_in_story = sent_tokenize(hall_stories[i])
    all_hall_sentences.extend(sents_in_story)
```

- After obtaining one list per author containing all the sentences in all of that author's stories, we need to clean the lists by removing irrelevant characters and by lemmatizing. Carrying out this step will increase the accuracy of our analysis (http://michael-harmon.com/blog/NLP2.html#second-bullet). For some variety of technique, we will use regular expression this time to clean our data.

[115]:
```
# Function to clean a list of sentences using REGEX and previously created␣
↪Lemmatizer function.
def preprocess_data_with_regex(sentencesList):
    cleaned_sentences_list = []
    for sentence in range(0, len(sentencesList)):
        cleaned_sentence = re.sub(r'\W', ' ', str(sentencesList[sentence])) #␣
↪Removing special characters
        cleaned_sentence = cleaned_sentence.lower() # Making sentence lowercase␣
↪for easier manipulation
        cleaned_sentence = re.sub(r'\s+[a-zA-Z]\s+', ' ', cleaned_sentence) #␣
↪Removing single characters
        cleaned_sentence = re.sub(r'\^[a-zA-Z]\s+', ' ', cleaned_sentence) #␣
↪Removing start single chars
        cleaned_sentence = re.sub(r'\s+', ' ', cleaned_sentence, flags=re.I) #␣
↪Removing multiple spaces
        cleaned_sentence = cleaned_sentence.split() # Tokenising sentence
        cleaned_sentence = lemmatize_words(cleaned_sentence) # Lemmatizing␣
↪using lemmatizer & POS tagging
        cleaned_sentence = ' '.join(cleaned_sentence) # Reconstructing sentence
        cleaned_sentences_list.append(cleaned_sentence) # Adding cleaned␣
↪sentence to new list
        return cleaned_sentences_list
```

[116]:
```
stopwords_set = list(set(stop_words + more_stop_words))

# Function to create a dictionary of words in each sentence.
def extract_features(sentencesList):
    words_in_sents_dict = dict()
    data = ' '.join(preprocess_data_with_regex([sentencesList])) # Passing the␣
↪list for preprocessing.
    # Tokenise cleaned sentences list into words list.
```

```
    for word in data.split():
        # Do not include irrelevant words.
        if (word not in stopwords_set and len(word) > 2):
            words_in_sents_dict[word] = True
    return words_in_sents_dict
```

- Now, we are ready to extract features from each sentence to add to a growing dictionary of words in each sentence. Note that we will only extract relevant features so that our analysis is more accurate. In the code block below, we call the extract_features function to do this. In that function, the words are also thoroughly cleaned by calling the preprocess_data_with_regex function and still cleaning the words further before their inclusion in the dictionary. As seen in the code block below, once that process is completed, the sentence is tagged as "african" or "european".

NOTE: Again, the use of "african" and "european" in tagging and naming the variables going forward does not imply that these features are representative of the entire European or African literature, at least not yet. For these features to be truly representative and to enhance our machine model's accuracy, we will need to include stories by various authors of those regions. This first model is simply a starting point and an invitation to further research and engineering of subsequent models building upon this one.

[117]:
```
african_features = [(extract_features(sentence), 'african') for sentence in␣
↪all_adichie_sentences]
european_features = [(extract_features(sentence), 'european') for sentence in␣
↪all_hall_sentences]
```

- To train and test our model effectively, we need to apply good machine learning practices, one of which is to split our datasets into training and test sets–with 80% used for training and the remaining 20% used for testing (https://www.askpython.com/python/examples/split-data-training-and-testing-set).

NOTE: I noticed from the list of sentences that the sentences of Hall's works were 1700+ in length while that of Adichie were 654. Hence, I set the cutoff based on the length of Adichie's sentences to make the sentences of both authors equal.

[118]:
```
# As european_features' length are 1700+, use african_features length (654)
# as cutoff for equality when analysing.
training_cutoff = math.floor((80/100) * len(african_features))

# 80% of all Adichie's sentences for more balanced training
african_train_set = african_features[0:training_cutoff] # 523
european_train_set = european_features[0:training_cutoff]
train_set = african_train_set + european_train_set

# 20% of all Adichie's sentences for more balanced testing
african_test_set = african_features[training_cutoff:]
european_test_set = european_features[training_cutoff:len(african_features)]
test_set = african_test_set + european_test_set
```

- Finally, it is time to train our data using the train set. As mentioned earlier, when the model

is first trained, the classifier will be saved in a file and loaded for subsequent use. Hence, we need to add a condition that checks for the file so that we do not spend needless time if the classifier is already saved.

[119]:
```
if os.path.isfile("literary_classifier.pickle"):
    # Load classifier if file is available.
    classifier = load_classifier("literary_classifier.pickle")
else:
    # Train classifier if file is not available.
    classifier = NaiveBayesClassifier.train(train_set)
    save_classifier(classifier, "literary_classifier.pickle")
```

- We now test our classifier to find out if it works and how well. Firstly, let us test the classifier using the test set.

[120]:
```
# Test African test set.
first_tag = classifier.classify(african_test_set[0][0])
if (first_tag == "african"):
    print("Classifier correctly identifies test set as:", first_tag)
else:
    print("Wrong classification!")
```

```
Classifier correctly identifies test set as: african
```

[121]:
```
# Test European test set.
second_tag = classifier.classify(european_test_set[0][0])
if (second_tag == "european"):
    print("Classifier correctly identifies test set as:", second_tag)
else:
    print("Wrong classification!")
```

```
Classifier correctly identifies test set as: european
```

So our classifier correctly classifies the test sets. Let us check exactly how accurate it was at doing so:

[122]:
```
accuracy_score = nltk.classify.util.accuracy(classifier, test_set)
accuracy_score_percent = round(accuracy_score * 100, 1)

print ('Classifier accuracy:', accuracy_score)
print("Classifier is", accuracy_score_percent, "percent accurate")
```

```
Classifier accuracy: 0.7061068702290076
Classifier is 70.6 percent accurate
```

As seen from the above, our classifier has an accuracy of approximately 70.6%. While this score is average, the datasets were limited and our model would benefit from scaling up to a larger dataset.

Now the hard part would be classifying stories entirely outside of this set.

- Firstly, let us see if the classifier can correctly classify a different short story of Adichie's and Hall's. Note also that because we will try to classify other works, I have created two functions

to enable reusability for subsequent classification.

[123]:
```
def classify_african_stories(filename):
    test_story = import_file("african_test_data/" + filename)
    test_sents = sent_tokenize(test_story)
    tag = classifier.classify(extract_features(test_sents))
    if (tag == "african"):
        print("Classifier correctly identifies testset as:", tag)
    else:
        print("Incorrect classification for African story!")
```

[124]:
```
classify_african_stories("Adichie_The_Arrangers_of_Marriage.txt")
```

```
Classifier correctly identifies testset as: african
```

As seen from the above, our classifier correctly identifies a different work of Adichie's that was not part of our train or test sets. This indicates that our classifier is good at recognising the writing style of the author.

What about for Hall's work?

[125]:
```
def classify_european_stories(filename):
    test_story = import_file("european_test_data/" + filename)
    test_sents = sent_tokenize(test_story)
    tag = classifier.classify(extract_features(test_sents))
    if (tag == "european"):
        print("Classifier correctly identifies testset as:", tag)
    else:
        print("Incorrect classification for European story!")
```

[126]:
```
classify_european_stories("Hall_Evie_excerpt.txt")
```

```
Classifier correctly identifies testset as: european
```

As seen from the above, again our classifier correctly identifies a different work of Hall's that was not part of our train or test sets. This indicates that our classifier is good at recognising Hall's writing style.

- Nonetheless, this section was about identifying African vs European works. For this purpose, we will use award-winning works of other authors to test our classifier.

[127]:
```
classify_african_stories("afrtest1.txt")
classify_african_stories("afrtest2.txt")
```

```
Classifier correctly identifies testset as: african
Classifier correctly identifies testset as: african
```

[128]:
```
classify_european_stories("eurtest1.txt")
classify_european_stories("eurtest2.txt")
```

```
Incorrect classification for European story!
Incorrect classification for European story!
```

As seen from the above, the classifier correctly identifies both test stories by African authors, but fails to do the same for stories by the European authors.

To understand why, let us view the most informative features that our classifier uses to classify the stories.

[129]:
```
classifier.show_most_informative_features(20)
```

```
Most Informative Features
                      tree = True           africa : europe =     21.7 : 1.0
                       die = True           africa : europe =      9.4 : 1.0
                       man = True           africa : europe =      7.4 : 1.0
                       buy = True           africa : europe =      6.3 : 1.0
                       men = True           africa : europe =      6.3 : 1.0
                       war = True           africa : europe =      6.2 : 1.0
                       day = True           africa : europe =      6.1 : 1.0
                  daughter = True           africa : europe =      5.7 : 1.0
                      step = True           europe : africa =      5.7 : 1.0
                    summer = True           africa : europe =      5.0 : 1.0
                     write = True           africa : europe =      5.0 : 1.0
                     plate = True           europe : africa =      5.0 : 1.0
                      read = True           africa : europe =      5.0 : 1.0
                     guest = True           europe : africa =      5.0 : 1.0
                university = True           africa : europe =      5.0 : 1.0
                      year = True           africa : europe =      5.0 : 1.0
                     leave = True           africa : europe =      4.6 : 1.0
                     child = True           africa : europe =      4.4 : 1.0
                       art = True           europe : africa =      4.3 : 1.0
              conversation = True           europe : africa =      4.3 : 1.0
```

The listing above shows the words in the training set that appear more often for one category than another. For instance, according to the classifier, the word, "tree" relate to an African audience 21.7 times more often than it would to a European audience. Note that this determination is most likely because the word was not mentioned in any of Hall's stories.

According to the NLTK website, "these ratios are known as likelihood ratios, and can be useful for comparing different feature-outcome relationships" (https://www.nltk.org/book/ch06.html).

It is also now unsurprising that the two test stories by African authors were correctly categorised. This is because one of the test stories also has a theme about death (which is the second most informative feature) and the other test story has themes about men and buying. On the other hand, notice that while there are many informative features for categorising African literature, there are only four most informative features in the top twenty for identifying European literature. This is certainly not a good ratio as it means no strong european associations have been identified yet, using Hall's works, to create a strong likelihood ratio in favour of European literature.

Retraining our classifier to be more accurate is beyond the scope of this project; however, to provide a starting point for how this could be done, let us add some sentences from one of the European literary works used as test data.

[130]:
```
new_sent_list = ["On the telly, there was a woman with a pointy face talking␣
 ↪full screen. Fuck fuck I know that bird I know I do ."]
```

Let us train this excerpt (from eurtest1.txt in european_test_data) and test our classifier with another sentence in the same story.

NOTE: To save time, this training has already been carried out and saved in a new file called rev_literary_classifier.pickle

[131]:
```
# Prepare new feature for addition to trainset.
new_eur_feature = [(extract_features(sentence), 'european') for sentence in␣
 ↪new_sent_list]

# Load or train classifier.
if os.path.isfile("rev_literary_classifier.pickle"):
    new_classifier = load_classifier("rev_literary_classifier.pickle")
else:
    new_classifier = NaiveBayesClassifier.train(train_set + new_eur_feature)
    save_classifier(new_classifier, "rev_literary_classifier.pickle")
```

[132]:
```
# Prepare another sentence from the same story for testing.
test_data = "But what the fuck is that bird doing in my telly ?"
test_story = sent_tokenize(test_data)

# Test the new data on both classifiers.
print("Former classifier classifies data as:", classifier.
 ↪classify(extract_features(test_story)))
print("Retrained classifier classifies data as:",new_classifier.
 ↪classify(extract_features(test_story)))
```

```
Former classifier classifies data as: african
Retrained classifier classifies data as: european
```

As we can see, from simply adding to the dataset and increasing the diversity of features in our training set, our classifier gets one step closer to accurately identifying European literary works.

4 Conclusion

Using various techniques of Natural Language Processing, including sentiment analysis, we have uncovered five elements that are important in the creation of a short story masterpiece. * A short story is more likely to be considered a masterpiece when narrated from a point of view that is different from the norm for stories in that genre. * Our main character is not an island, they need to be connected to other people to move the story and thus, building these people is just as important as building the main character. * Characters need to interact with one another to clue readers into what is going on and drive the story forward. * How your characters feel and move are just as important as what they say * Positive or negative themes do not matter, telling a balanced story does.

We have also observed some elements that might resonate more with audiences of one region than

with audiences of another. For instance, the idea of family for African audiences seems to include both the nuclear and extended family.

Finally, we used machine learning to train a machine model to help writers better target their audience or even capture audiences of other regions by identifying which story will resonate more with an African audience than a European one and vice versa. Although, our classifier model is still crude and misidentifies some European stories, as I have already demonstrated, by increasing the dataset for European stories to enable our classifier identify more unique features, we can further this project and gradually increase the accuracy of our model to become a more useful tool to established and upcoming writers.

5 Additional References

Learning Resources: * http://www.nltk.org/howto/sentiment.html * https://www.opensourceforu.com/2016/12/analysing-sentiments-nltk/ * http://www.nltk.org/book/ * https://towardsdatascience.com/introduction-to-data-visualization-in-python-89a54c97fbed

Additional Stories Used for Testing: * afrtest1.txt: https://www.addastories.org/marry-african-president/ * afrtest2.txt: https://www.addastories.org/the-dawning/ * afrtest3.txt: https://johannesburgreviewofbooks.com/2019/02/04/new-short-fiction-the-neighbourhood-watch-by-remy-ngamije/ * eurtest1.txt: https://www.euprizeliterature.eu/authors/sophie-daull * eurtest2.txt: https://www.theguardian.com/books/2017/oct/03/read-the-edge-of-the-shoal-by-cynan-jones-winner-of-the-bbc-national-short-story-award