Executive Summary

Write a short (200 words max) summary of what the report contains and what you have discovered.

The main purpose of this project is to analyze multiple txt files. Each text file is a book, which is extracted from Project Gutenberg. There are 1176 books and they are fully and partially analyzed based on the tasks. The dataset is placed in the current working directory and has been loaded. First, the book is analyzed for metadata and other pieces of information that can be extracted.

The dataset has been used for natural language processing. First, the data has been cleaned and scrubbed using regular expressions. Tokenization for a sample of 300 books has been made from the total dataset. Frequency Distribution for tokens is presented. Further cleaning of our dataset StopWords has been excluded for further analysis. Frequency distribution is presented. Lemmatization is done for our selected book Frequency distribution is presented. Sentiment analysis is utilized on a scale of (-1 to 1). The is extracted from all books(1176) and converted to pandas dataframe. Frequency distribution of the books by Title and author is presented. Searching by title value and Author is included. Exploratory data analysis using Tables and graphs is presented Finally, Descriptive analysis is been done based on the interesting questions raised during exploratory analysis.

Data Cleaning and Preparations

In this section you should demonstrate how you have programmatically loaded the dataset (or those parts of it you intend to use) into memory, and cleaned and prepared it. Explain the steps you take.

Importing Libraries

In [1]:
```
import os
import nltk
import nltk.corpus
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.probability import FreqDist
from nltk.sentiment import SentimentIntensityAnalyzer
import re
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

Locating the Dataset and assigning variables

In [2]:
```
# Dataset is extracted on the same folder as of the notebook

# Assigning a variable to dataset directory
dir_path = (os.getcwd() + "\\dataset")
```

```
#changing the directory path to current working directory
os.chdir(dir_path)

#path of our Dataset
dataset = os.listdir(os.getcwd())
```

Checking and counting text files (books) in the dataset

In [3]:
```
# Iterate the directory for text files
# count the files when the condition is true
counter = 0
for textfile in dataset:
    if textfile.endswith(".txt"): # checking extension of files
        counter += 1              # counts every txt file

print(f"This is part of Gutenberg dataset \n\tThe dataset contains {len(dataset)} f

# Checking if there is an error on the given dataset for many reasons(extraction, v
if len(dataset) == counter: # checking only .txt files occur
    print('No errors found, the file contains only txt files')
else:
    print('Errors or other file types detected, on our dataset')
```

```
This is part of Gutenberg dataset
        The dataset contains 1176 files
        Out of which 1176 are text files
No errors found, the file contains only txt files
```

Natural Language Processing

This is a big function created for cleaning and tokenization

In [4]:
```
# Defining function for tokenization and cleaning
def tokenizer(file_location):                # Define
    with open(file_location, 'r') as fdata:  # Open
        text = fdata.read()                  # Read
    """
    Here analyzing the textbooks physically they start With Book concent
    and metadata(Title, Author, Release date, Language, Encoding)
    The book starts after mentioning *** START OF THIS PROJECT GUTENBERG EBOOK
    This information is forseen to analyze our dataset
    """
    # Skipping the metadata and go through the start of the book
    # find function is used to get index of book start and end
    starts = text.find("*** START OF THIS PROJECT GUTENBERG")
    ends = text.find("*** END OF THIS PROJECT GUTENBERG")
    book = text[starts:ends] # book part for analysis

    """Data Cleaning and preparation"""

    #Removing new line new tab, raw string, numbers and ofcourse there are extra sp
    for i in ["\n", "\t", "\r", "\d", "_",   "  ", "    "]:
        book = book.replace(i, " ")   ### Lowered Case

    # Extra cleaning
    # Case lowering to avoid repitition of capital and small letters
    book = book.lower()

    # This scrubs all punctuations using regular expression
    book = re.sub(r"[^\w\s]", "", book)

    # Now it is Great we are good to go to tokenization
```

```
    """Tokenization"""

    # Using nltk tokenizer from nltk library to tokenized our cleaned book
    tok = nltk.word_tokenize(book)
    return(tok)
```

Extracting and analyzing tokens for specfic dataset using the above function

In [5]:
```
"""For multiple datasets we can use the tokenizer function created above"""

#iterating through multiple datasets for tokens
# for loop with counter assigned 0 to count number of tokens
tok_counter = 0
for i in dataset[:4]:     # for counting tokens based on dataset index
    print(f"File {i} token = {len(tokenizer(i))}")
    tok_counter += len(tokenizer(i))

print(f"Total tokens = {tok_counter}")
```

```
File 10001.txt token = 5583
File 10002.txt token = 50973
File 10003.txt token = 61339
File 10004.txt token = 48121
Total tokens = 166016
```

Frequency Distrbution of tokens Output most frequent tokens

In [6]:
```
# Using FreqDist from nltk.corpus library to see most frequent words
# here Dataset[1] (for text file at index 1) is used
# FreqDist from nltk library is utilized to count repeated data
top_toks = FreqDist(tokenizer(dataset[1])).most_common(5)
# It is a good Idea to change to pandas DataFrame for later analysis
tokdf = pd.DataFrame(top_toks, columns=["Word","Frequency"])
display(tokdf)
```

	Word	Frequency
0	the	3634
1	i	2279
2	of	1708
3	and	1661
4	a	1346

Further cleaning of our dataset

Removing most common words, to visualize and describe our file

In [7]:
```
"""
Removing most common words that are
not used to describe our topic content
"""
# using nltk pakage to remove stopwords
# used the same text file as above to visualize
"""
This checks if the tokens created from tokenizer
are found on stopwords from nltk if not assign to stoks list
"""
```

```
stoks = [i for i in tokenizer(dataset[1])     # tokens assigned
         if i not in stopwords.words('english')] # tokens not in stopwords

#extracting top 5 tokens after removing stopwords
stop_top5 = FreqDist(stoks).most_common(5)
#Changing to pandas DataFrame
stdf = pd.DataFrame(stop_top5, columns= ["Word","Frequency"])
display(stdf)
```

	Word	Frequency
0	could	174
1	time	159
2	great	144
3	seemed	143
4	came	130

It is clear that that without stop words the most frequent were "the, i, of" after stopword cleaning "could, time, great" are most frequent words

Lemmatization

In [8]:
```
# Changing words to their root form for further analysis
def lemmatizer(toks):
    #lemmetizing each word from wordnet
    lemmatizer = WordNetLemmatizer()
    lemtoks = [lemmatizer.lemmatize(word) for word in toks] # lemmatizing each toke

    return(lemtoks)

## most frequent root words in our dataset
lemm_top5 = (FreqDist(lemmatizer(stoks)).most_common(5))  # stoks is used because i
# converting to panda dataframe
lmdf = pd.DataFrame(lemm_top5, columns= ["Word", "Frequency"])
display (lmdf)
```

	Word	Frequency
0	time	189
1	could	174
2	great	144
3	seemed	143
4	sun	138

There is a small difference in frequent words from cleaned data and from lemmatization from wordnet because of the root word change

Creating Bigrams

In [9]:
```
from nltk.collocations import BigramCollocationFinder
#this section is to analyze two specfic words or bigrams
#using nltk library importing bigram finder
#creating a function to generate bigram
def bigram_out (toks):
```

```
    #bigram creater
    bcf = BigramCollocationFinder.from_words(toks)
    #creating panda data frame for bigrams with frequency
    df_bigram = pd.DataFrame(data=bcf.ngram_fd.items(),
                             columns= ["Bigram","Frequency"])
    # using sort values, by default it sorts ascending
    #Decending is needed because to extract top bigrams, just not to use negetive i
    # no index is assigned on our data frame therefore after decending we reset our
    df_bigram= df_bigram.sort_values(by = ["Frequency"], ascending= False).reset_ir

    df_bigram = df_bigram[:5]
    return(df_bigram)

display(bigram_out(stoks)) # stoks is after cleaning stopwords
```

	Bigram	Frequency
0	(could, see)	52
1	(green, sun)	17
2	(made, way)	14
3	(could, hear)	14
4	(see, nothing)	13

sentiment analysis

Sentiment analysis based on a range -1 to 1 as -1 only negative and 1 only positive

In [10]:
```
# some idea from reference[2] has been utilized
def sentiment_analyzer(toks):
    #assigning variable to the sentiment analyzer
    sa = SentimentIntensityAnalyzer()

    # creating a counter values for our sentimentanalysis
    # we assign a 0 vaule since it is integer to add up,
    toks_positive = 0
    toks_negative = 0
    toks_neutral = 0
    overall_score = []
    #iterate through each tokens to get polarity +ve or -ve
    # using polarity scores from nltk sentiment pakages
    for word in toks:
        if sa.polarity_scores(word)["compound"] > 0:
            toks_positive += 1
            overall_score.append(sa.polarity_scores(word)["compound"])

        elif sa.polarity_scores(word)["compound"] < 0:
            toks_negative += 1
            overall_score.append(sa.polarity_scores(word)["compound"])

        elif sa.polarity_scores(word)["compound"] == 0:
            toks_neutral += 1
            overall_score.append(sa.polarity_scores(word)["compound"])
    #### convert the sentiment analysis scores to panda data frame
    overall_score_numbers = [i for i in overall_score if i != 0]
    df_sen = pd.DataFrame(data = {
        'Total toks' : len(toks),
        'Positive toks' : toks_positive,
        'Negative toks' : toks_negative,
```

```
        'Neutral toks' : toks_neutral,
        'Overall score' : sum(overall_score_numbers) / len(overall_score_numbers)
    }, index= [0])
    return(df_sen)

display(sentiment_analyzer(stoks))
```

	Total toks	Positive toks	Negative toks	Neutral toks	Overall score
0	24015	1388	1589	21038	-0.018286

we can catagorize this dataset (dataset[1]) is neutral

Extracting metadata from our files

In [11]:
```
# Creating metadata extractor function
def metadata_extractor(txtfile):
    f = open(txtfile, 'r')   # open specfic file
    for line in f:

        if "Title: " in line:              # if this characer found
            title_key = line.split(":")[0] # splits based on the given character le
            title_value = line.split(":")[1].strip('\n')
# when analyzing our data set there is editor in place of author for some books
# or operator is used to include this
# NB author key in our dataframe will show editor insted of Author because of index
# There are multiple authors for one book in that case we will search for authors i
        if "Author: " in line or "Editor: " in line or "Authors: " in line:
            author_key = line.split(":")[0]
            author_value = line.split(":")[1].strip('\n')

        if "Release Date: " in line:
            date_key = line.split(":")[0]
            date_value = line.split(":")[1].strip('\n')

        if "Language: " in line:
            language_key = line.split(":")[0]
            language_value = line.split(":")[1].strip('\n')
    # A dictionary to catch the the meta data for selected file
    metadata = {
        'Title': title_value,
        'Author': author_value,
        'Release Date': date_value,
        'Language': language_value
    }
    # Converting metadata to panda dataframe
    df = pd.DataFrame(metadata, index= [author_value])#
    return(df)
```

When analyzing our data set there is editor in place of author for some books. Although there are multiple authors for one book. This was an issue but solved.

Creating Big data frame that includes all of our data_set

- it iterates through each file and extracts meta data for the whole dataset

In [12]:
```
# creating new empty data frame
ndf = pd.DataFrame()

# Iterating each book for metadata(title, author...) values using the metadata extr
```

```
for i in dataset[:]:
    ndf = pd.concat([ndf, (metadata_extractor(i))])

# we can asign a specific column as an index
ndf.set_index("Title", inplace= True) # inplace= True is just change applied to cur
ndf.reset_index()

# Assigning our overall dataset bigdf ( Big data frame for easier use)
bigdf = ndf.reset_index()
```

Displaying Sample of our dataset

- 3 books from top index
- 3 books from bottom index

In [13]:
```
# displaying 3 books from top and bottom of our data frame
display(bigdf.head(3), bigdf.tail(3))
```

	Title	Author	Release Date	Language
0	Apocolocyntosis	Lucius Seneca	November 10, 2003 [EBook #10001]	English
1	The House on the Borderland	William Hope Hodgson	November 10, 2003 [EBook #10002]	English
2	My First Years As A Frenchwoman, 1876-1879	Mary King Waddington	November 10, 2003 [EBook #10003]	English

	Title	Author	Release Date	Language
1173	The Poetical Works of William Wordsworth, Vol...	William Wordsworth	May 19, 2004 [EBook #12383]	English
1174	The Italians	Frances Elliot	May 19, 2004 [eBook #12385]	English
1175	Samantha at the St. Louis Exposition	Marietta Holley	May 19, 2004 [eBook #12386]	English

Displaying information of our overall dataset (Big Dataset)

- the data frame is assigned "bigdf"

In [14]:
```
# displaying our data frame info
bigdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1176 entries, 0 to 1175
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Title         1176 non-null   object
 1   Author        1176 non-null   object
 2   Release Date  1176 non-null   object
 3   Language      1176 non-null   object
dtypes: object(4)
memory usage: 36.9+ KB
```

Explaining our Dataset on metadata values

- Unique, Frequency, Top, Count ...

In [15]:
```
# This function is used to describe the main statistics,the output explained it
display(bigdf.describe())
```

	Title	Author	Release Date	Language
count	1176	1176	1176	1176
unique	1053	680	1176	5
top	The Mirror of Literature, Amusement, and Inst...	Various	November 10, 2003 [EBook #10001]	English
freq	39	198	1	1172

Frequency Distrubutuon of books per title

In [16]:
```
# How many Books per title
# value counts displays frequency of the data value
# head is used to output the top 5 repeated values, can use tail function for low f
display(bigdf["Title"].value_counts().head(5))
```

```
The Mirror of Literature, Amusement, and Instruction             39
Audio                                                            31
The Mirror of Literature, Amusement, and Instruction, Vol. 12,   10
A Compilation of the Messages and Papers of the Presidents        9
The Mirror of Literature, Amusement, and Instruction.             8
Name: Title, dtype: int64
```

Frequency Distrubutuon of books per Author

In [17]:
```
# How many books per Author
# value counts displays frequency of the data value
# head is used to output the top 5 repeated values, can use tail function for low f
display(bigdf["Author"].value_counts().head(5))
```

```
Various             198
W.W. Jacobs          52
Anonymous            30
Roger McGuinn        30
Charles Kingsley     14
Name: Author, dtype: int64
```

searching our data frame by index, title, author ...

searching by index

In [18]:
```
# searching by index
display(bigdf.iloc[3:5])
```

	Title	Author	Release Date	Language
3	The Warriors	Lindsay, Anna Robertson Brown	November 10, 2003 [EBook #10004]	English
4	A Voyage to the Moon	George Tucker	November 7, 2003 [EBook #10005]	English

Searching by Title

In [19]:
```
# Searching by Title
# it is based on panda dataframe function we can search part of the title
search_title = "Apocol"
display (bigdf[(bigdf["Title"].str.contains(search_title))])
```

	Title	Author	Release Date	Language
0	Apocolocyntosis	Lucius Seneca	November 10, 2003 [EBook #10001]	English

Searching by Author

In [20]:
```
# Searching by Author
# it is based on panda dataframe function we can search part of the title
search_author = "Frances Ell"
display (bigdf[(bigdf["Author"].str.contains(search_author))])
```

	Title	Author	Release Date	Language
749	Sowing and Reaping	Frances Ellen Watkins Harper	February 10, 2004 [eBook #11022]	English
769	Minnie's Sacrifice	Frances Ellen Watkins Harper	February 12, 2004 [EBook #11053]	English
772	Trial and Triumph	Frances Ellen Watkins Harper	February 12, 2004 [EBook #11056]	English
1174	The Italians	Frances Elliot	May 19, 2004 [eBook #12385]	English

Exploratory Data Analysis

Programmatically explore the dataset. Tell the reader, how the dataset looks like and at least three most interesting observations that you can learn from the data. Propose at least three (3) questions to ask with the dataset.

How many books are there in our dataset

In [21]:
```
def book_counter():
    counter = 0
    for textfile in dataset:
        if textfile.endswith(".txt"): # checking extension of files
            counter += 1              # counts every txt file

    print(f"This is part of Gutenberg dataset \n\tThe dataset contains {len(dataset

    # Checking if there is an error on the given dataset for many reasons(extractic
    if len(dataset) == counter: # checking only .txt files occur
        print('No errors found, the file contains only txt files')
    else:
        print('Errors or other file types detected, on our dataset')

book_counter()
```

```
This is part of Gutenberg dataset
        The dataset contains 1176 files
        Out of which 1176 are text files
No errors found, the file contains only txt files
```

Opening the cover page (before the start of Project Gutenberg ebook) of random txt file from our dataset to see what information we can get

In [22]:
```
def cover_page(file_location):               # Define
    with open(file_location, 'r') as fdata:  # Open
        text = fdata.read()

    # Ebook starts after mentioning *** START OF THIS PROJECT GUTENBERG
    starts = text.find("*** START OF THIS PROJECT GUTENBERG")

    # Cover page is found above this line we can index it using starts index
    first_page = text[:starts]
    return(first_page)

print(cover_page(dataset[1]))
```

```
Project Gutenberg's The House on the Borderland, by William Hope Hodgson

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The House on the Borderland

Author: William Hope Hodgson

Release Date: November 10, 2003 [EBook #10002]
Last updated: January 19, 2009

Language: English

Character set encoding: ASCII
```

Listing the files with their basic info to better understand how our data looks like in tabular dataframe format

In [23]:
```
display(bigdf.head(3))
```

	Title	Author	Release Date	Language
0	Apocolocyntosis	Lucius Seneca	November 10, 2003 [EBook #10001]	English
1	The House on the Borderland	William Hope Hodgson	November 10, 2003 [EBook #10002]	English
2	My First Years As A Frenchwoman, 1876-1879	Mary King Waddington	November 10, 2003 [EBook #10003]	English

Tokens

selection of 300 books is analyzed for better analysis

Analyzing books for tokens and Knowing how many words are used with stastical discription

Nb the tokenizer function **excludes** the metadata before the start of the e book and after the end of e book i.e in between ***START OF THIS PROJECT GUTENBERG EBOOK and*** END OF THIS PROJECT GUTENBERG EBOOK***

Nb: I have seen files which appeared with 0 token count and I excluded them for further analysis, I decided to continue with this method because this method gets almost all files

In [24]:
```
# Nb there are files with 0 token counters and are excluded for further analysis

# All data selected
#Creating a list of tokens for overall dataset
overall_tokens = [len(tokenizer(i)) for i in dataset[:300] if len(tokenizer(i)) !=

# Creating panda dataframe of token for analysis
dftoken = pd.DataFrame(overall_tokens, index=[i for i in dataset[:300] if len(toker
```

Displaying Sample token count from our dataset

In [25]:
```
display(dftoken.head(3)) # head function counts from top
```

	Tokens
10001.txt	5583
10002.txt	50973
10003.txt	61339

Stastistics of our overall dataset based on tokens

In [26]:
```
# This is overall word count
# i.e without removing stop words and without lemmatization
display(dftoken.describe())
```

	Tokens
count	198.000000
mean	62904.767677
std	46178.829391
min	1395.000000
25%	28030.750000
50%	50888.000000
75%	87026.250000
max	203948.000000

Filtering most common words from our tokens and after that Lemmetizing

In [27]:
```
# using nltk pakage to remove stopwords
# used the same text file as above to visualize
"""
This checks if the tokens created from tokenizer
```

```
are found on stopwords from nltk if not assign to stoks list
"""
def stopword_lemm(files):
    stoks = [i for i in tokenizer(files)     # tokens assigned
         if i not in stopwords.words('english')] # tokens not in stopwords

    #Lemmetizing each word from wordnet
    lemmatizer = WordNetLemmatizer()
    lemtoks = [lemmatizer.lemmatize(word) for word in stoks] # lemmatizing each tok
    return(lemtoks)
```

In [28]:
```
var_lemm = stopword_lemm(dataset[1])
lemmtoken = pd.DataFrame(var_lemm, columns=["Lemmatized words"])
```

Statistics of our Lemmitized tokens for our selected data set

In [29]:
```
display(lemmtoken.describe())
```

	Lemmatized words
count	24015
unique	4950
top	time
freq	189

In [30]:
```
# Displaying most common frequent words
display(FreqDist(var_lemm).most_common(4))
```

```
[('time', 189), ('could', 174), ('great', 144), ('seemed', 143)]
```

Sentiment analysis of our selected book from (-1 to 1) -1 being most negative 1 being most positive

In [31]:
```
display(sentiment_analyzer(stopword_lemm(dataset[1])))
```

	Total toks	Positive toks	Negative toks	Neutral toks	Overall score
0	24015	1418	1582	21015	-0.014423

Here Dataset[1] sounds neutral or we can say 1.44% negative. overall scores can be categorize as ordinal values

Questions formulated after analyzing the Dataset and books

1. Type and amount of files for analysis
2. How many words are found on each book do statical analysis
3. Most frequent words (tokens)
4. How many unique words
5. How does it sound to the reader (positive or negative)
6. Books per Author
7. Books per Title

Descriptive Analytics

Interogate the dataset with descriptive analytics. Answer the three (3) questions that you found interesting in the previous section using descriptive analytics. Make each question a subsection of this section and follow the steps: 1. the question, 2. justify why it is interesting 3. the answer.

Question 1: Type and amount of files for analysis

Why it is interesting?

- Knowing the type of the docuement being analyzed is the basics for identifying the methods for analysis.
- Amount of files is used to know how the extent of our data analysis can go

In [32]:
```
book_counter()
```

```
This is part of Gutenberg dataset
        The dataset contains 1176 files
        Out of which 1176 are text files
No errors found, the file contains only txt files
```

Our data set is extracted from project Gutenberg and it contains 1176 books

Question 2: How many words are found on each book do statical analysis

Why is it interesting?

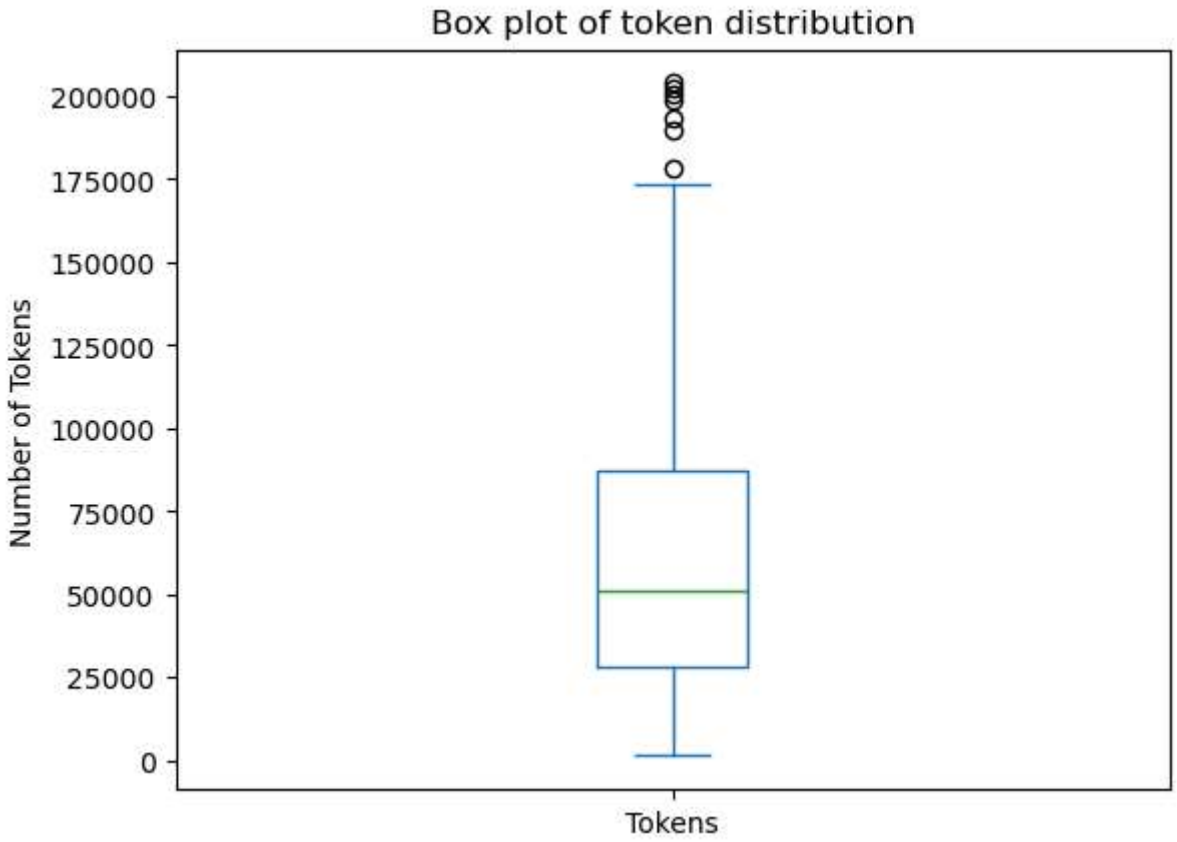
- We want to analyze statistical analysis of words Authors write.
- this may be used to predict how much to write
- The first 300 books are analyzed from our data set

In [33]:
```
display(dftoken.describe())
```

	Tokens
count	198.000000
mean	62904.767677
std	46178.829391
min	1395.000000
25%	28030.750000
50%	50888.000000
75%	87026.250000
max	203948.000000

Mean is closer to the minimium word count than maximum word count

In [34]:
```
dftoken.plot(kind = 'box', ylabel ='Number of Tokens', title = 'Box plot of token c
plt.show()
```



- Most books written tend to choose small number of words
- This shows that that the books are positively skiewed
- There are some outliers on the maximum diretion

Question 3: Most frequent words tokens

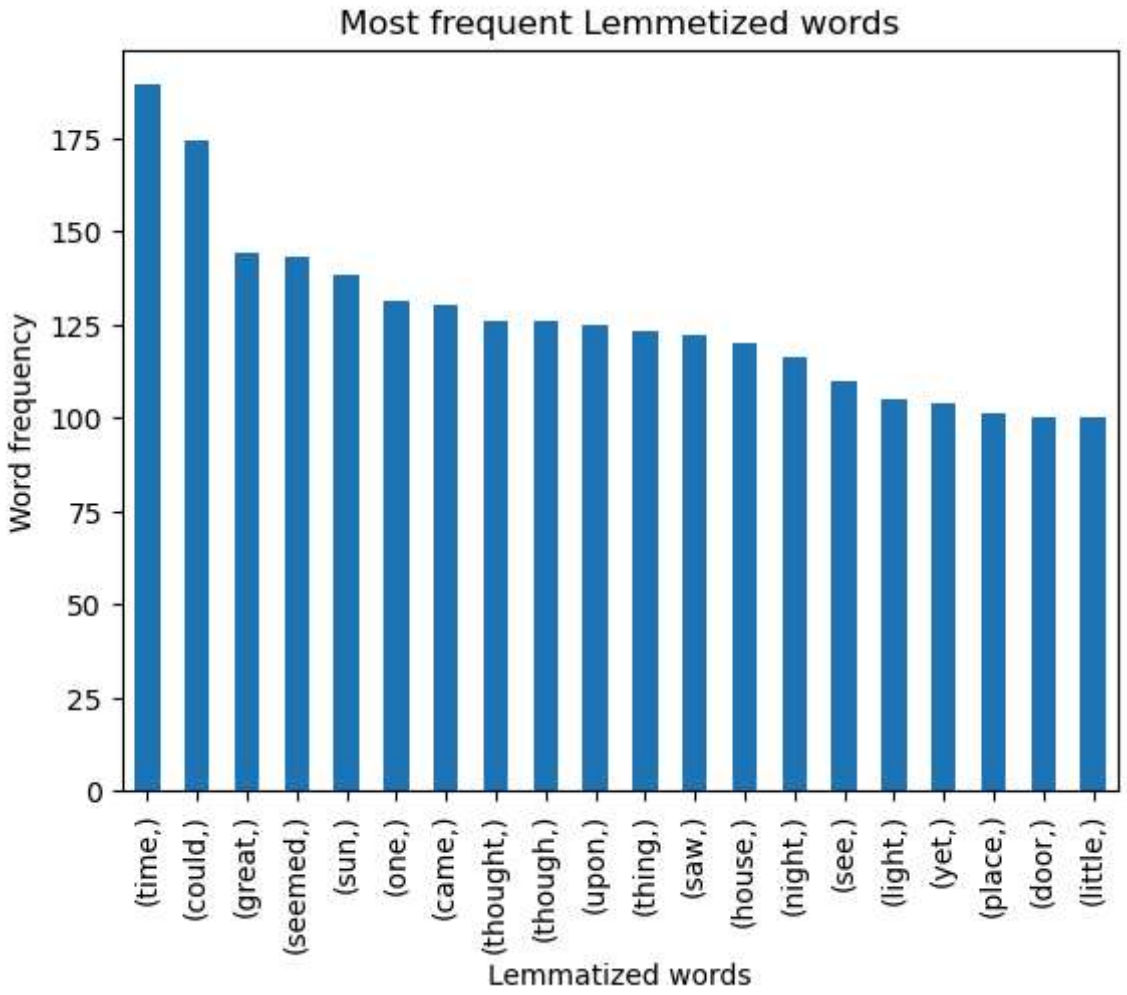
Why is it interesting?

- Using most frequent words we can analyze and categorize the book
- This book can be about romance, about religion, about science...

In [35]:
```
# Four of the top frequent words from dataset index 1 is displayed here
display(lemmtoken.value_counts().head(5))
```

```
Lemmatized words
time                189
could               174
great               144
seemed              143
sun                 138
dtype: int64
```

In [36]:
```
# Top 20 most frequent lemmetized words
lemmtoken.value_counts().head(20).plot(kind = 'bar', ylabel = 'Word frequency', tit
plt.show()
```



Question 4: How many unique words

why is it intersting?

- Unique words mostly tell about lexical diversity
- Wether the writer utlized many complex words or
- Wether it is easy to read based on small lexical diversity value

In [37]:
```
display(lemmtoken.describe())
```

	Lemmatized words
count	24015
unique	4950
top	time
freq	189

There are 4950 unique lemmatized words for our selected book , Lexica diversity can be determined based on unique words, word count, and frequency of words

Question 5: How does it sound to the reader

why is it interesting?

- This is widely used for social medias
- sentiment analysis can easly identify between negative and positive words
- it is a good idea to know how the written book sound to the reader.

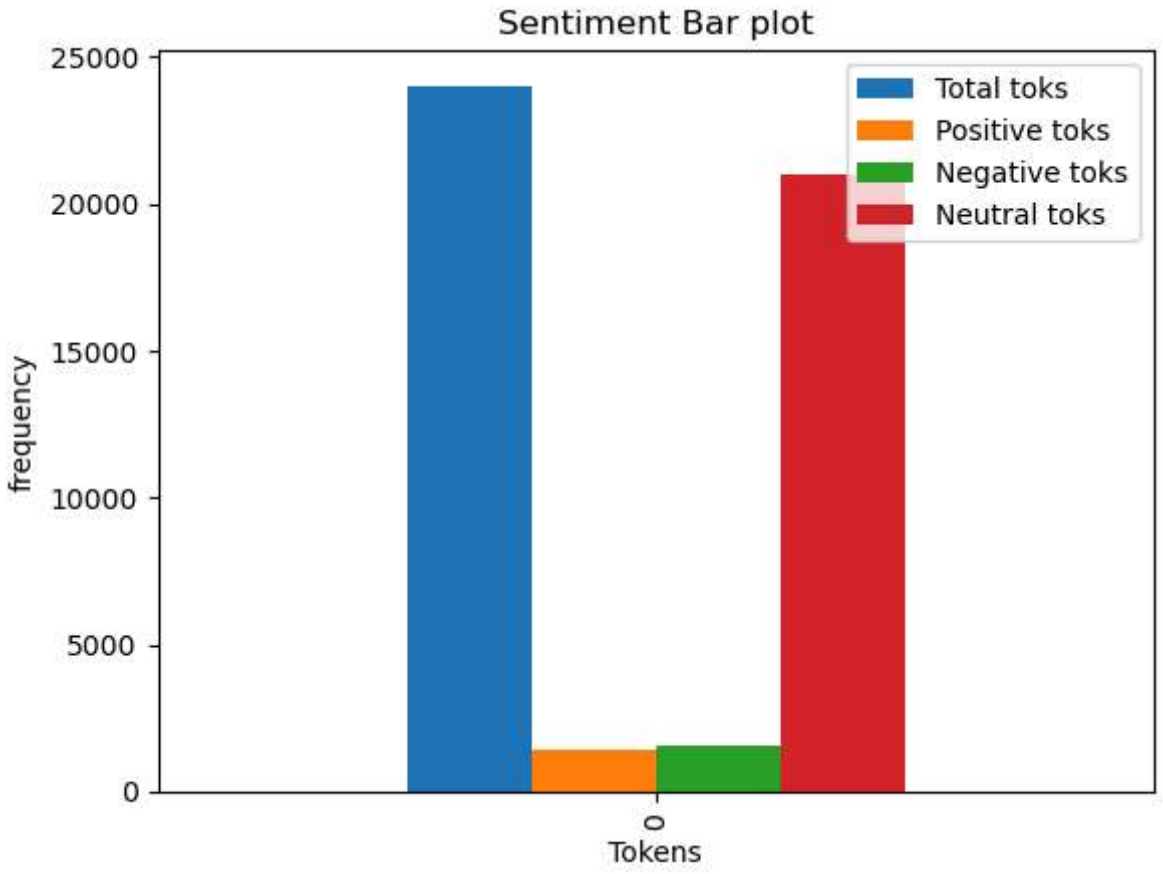
In [38]:
```
# one book is selected and (cleaning, stopwords, lemmitazation is done before)
se_selected = sentiment_analyzer(stopword_lemm(dataset[1]))
```

In [39]:
```
display(se_selected)
```

	Total toks	Positive toks	Negative toks	Neutral toks	Overall score
0	24015	1418	1582	21015	-0.014423

In [40]:
```
del se_selected["Overall score"]
```

In [41]:
```
se_selected.plot(kind ='bar', xlabel = 'Tokens', ylabel ='frequency', title= 'Senti
plt.show()
```



This bar plot describes how much it sounds to the reader, it is 1.4% negative

Question 6: Frequency of books per Author

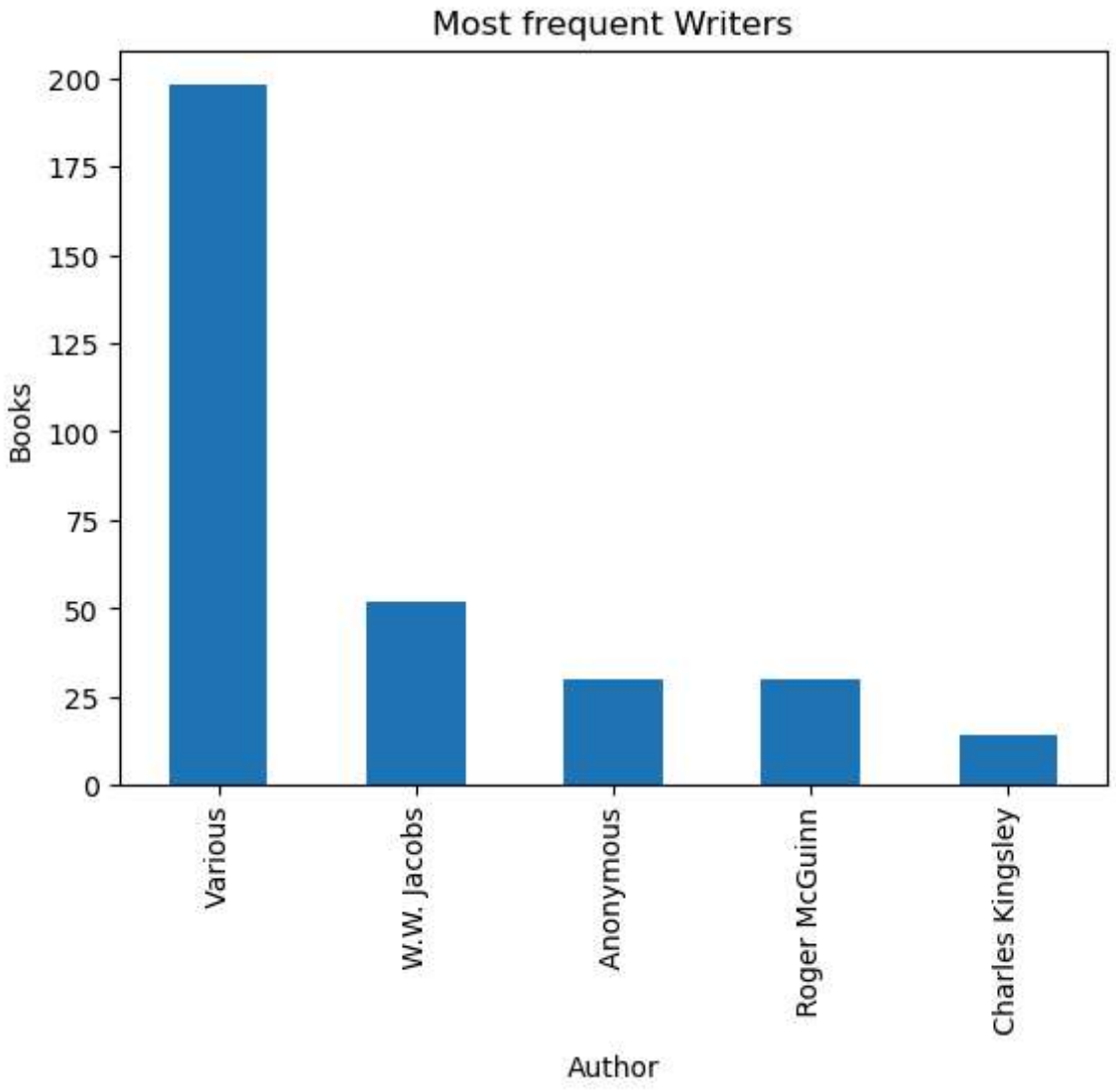
why is it intersting?

- it is used to identify who writes the most

In [42]:
```
display(bigdf["Author"].value_counts().head(5))
```

```
Various             198
W.W. Jacobs          52
Anonymous            30
Roger McGuinn        30
Charles Kingsley     14
Name: Author, dtype: int64
```

In [43]:
```
bigdf["Author"].value_counts().head(5).plot(kind = 'bar', xlabel = 'Author', ylabel
plt.show()
```



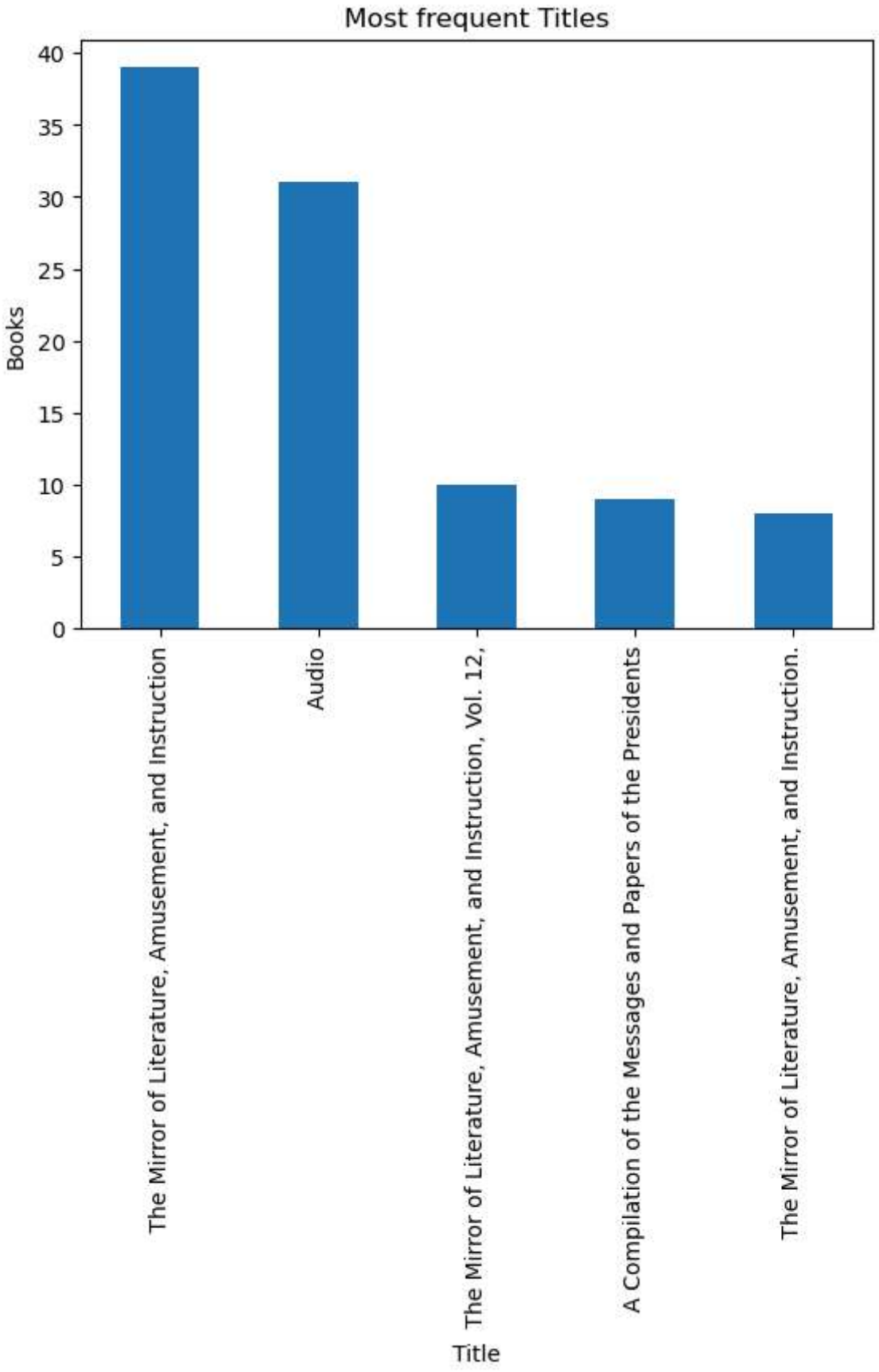
This bar graph identifies who writes the most

Question 7: Books per Title

why is it interesting?

- Used to idenify which books have volums
- which titles are repeated

In [44]:
```
bigdf["Title"].value_counts().head(5).plot(kind = 'bar', xlabel = 'Title', ylabel =
plt.show()
```



Bibliography

List all sources you have utilised in the making of this report here in this single markdown cell.

1 https://www.nltk.org/

2 https://odonnell31.medium.com/nlp-in-python-a-primer-on-nltk-with-project-gutenberg-fcc02be63d9a (Michael ODonnell, DataScientist, Adjunct Professor, Triathlete.

3 https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.html#pandas.DataFrame

4 https://www.w3schools.com/python/pandas/pandas_dataframes.asp

5 https://stackoverflow.com/