```
# -*- coding: utf-8 -*-
"""JTHOMAS Week7 LAB.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1hwUx-vo1jwXqf-rb-fTePtBwxWM3XakZ
"""

from pip._internal.cli.cmdoptions import timeout

'''
Web Link:
=====================
"https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/#sample-review"
=====================
Download Link:
=====================
https://mcauleylab.ucsd.edu/public_datasets/data/amazon_v2/categoryFiles/
Video_Games.json.gz
=====================
"Video_Games.json.gz" Specification:
=====================
reviewerID -    ID of the reviewer, e.g. A2SUAM1J3GNN3B
asin -          ID of the product, e.g. 0000013714
reviewerName -  name of the reviewer
vote -          helpful votes of the review
style -         a dictionary of the product metadata, e.g., "Format" is "Hardcover"
reviewText -    text of the review
overall -       rating of the product
summary -       summary of the review
unixReviewTime -    time of the review (unix time)
reviewTime -    time of the review (raw)
image -         images that users post after they have received the product
=====================
'''
# Import Libraries and Tools
import json
import gzip
import spacy # For natural language processing
from collections import Counter
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer # for Sentiment
Analysis
from nltk.corpus import wordnet as wn # Definitions, Synonyms, Antonyms
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
import requests
from pathlib import Path
import re

nltk.download('vader_lexicon')
nltk.download('wordnet')

path="/Video_Games.json - Copy.gz"
url = 'https://www.gutenberg.org/cache/epub/1513/pg1513.txt'
'''
Function Definition
```

```python
'''
def parse(path):
  g = gzip.open(path, 'rb')
  for l in g:
    yield json.loads(l)

# declare empty Lists
overall_ratings = []
hundred_reviews = []


# AVERAGE VIDEO GAME RATING
# For...Each loop to loop through the JSON.GZ File
for review in parse(path):
    overall_ratings.append(review['overall'])

# Compute and Print the Average 2018 Video Game Rating
print("Average 2018 Video Game Rating: ", sum(overall_ratings) /
len(overall_ratings))


count = 0
# Gather and Store one hundred Review Texts for Natural Language Processing Tasks
for review in parse(path):
    text = review['reviewText']
    hundred_reviews.append(text)
    count += 1
    if count == 100:
        break

print("# ===========================================")
print("# Parts-of-speech Tagging")
print("# ===========================================")
# declare a natural langue processor using model "en_core_web_sm"
nlp = spacy.load("en_core_web_sm")
# set the count
count = 0
# For...Each Loop
for review in hundred_reviews:
    # Invoke natural language processing ON EACH [review].
    doc = nlp(review)
    print("Review Text at Index ", count, ":") # Display a section heading
    # tokenize what is in [doc] and output to the console.
    # one "token" is equivalent to one word out of each Review Text segment
    # nested For...Each loop
    for token in doc:
        print(f"{token.text:12} {token.pos_:8} {token.tag_:6} {token.dep_:8}")
    print()
    # Increment count.
    count += 1

print("# ===========================================")
print("# Sentiment Analysis")
print("# ===========================================")
# declare a Sentiment Intensity Analyzer object and store it for reuse.
sia = SentimentIntensityAnalyzer()

# Manually define an EXCLUDE THESE WORDS List
exclude = ["a", "an", "the", "The", "and", "or", "but", ".", ",", "\n", " ", ":",
";", "'", "\n\n", "?", "!", "it", "is", "in", "to", "I",
           "on", "was", "for", "with", "have", "you", "You", "been", "goes", "am",
"of", "&", "who", "product", "description", "..", "...",
           "n't", "so", "So", "are", "has", "over", "My", "shipping", "Not", "not",
"It", "For", "But", "Learning", "This", "this", "AND", "any",
           "way", "many" "town", "line", "than", "ordering", "Is", "will", "that",
"-", "even", "Does", "were", "their", "there", "they're",
           "did", "Then", "very", "time", "that", "game", "my", "work", "play",
"be", "get", "Trail", "since", "help", "after", "bye", "Best",
           "install", "see", "ordered", "daughter", "son", "nephew", "alright",
"move", "THIS", "to", "too", "two", "format", "about", "down"]

# Manually define POSITIVE words as it relates to video games
positive_words = ["exciting", "immersive", "experience", "innovative",
"captivating", "riveting", "addictive", "fluid", "balanced", "compatible",
"engaging",
                  "original", "worked", "playable", "universal", "skillful",
"wonderful", "enjoy", "classic", "fun", "loved", "recommend", "recommended",
                  "installs", "ideal", "impressive", "challenging"]

# Manually define NEGATIVE words as it relates to video games
negative_words = ["pixelated", "grainy", "cumbersome", "dark", "violent",
"immodest", "provocative", "incompatible", "not worth", "outdated", "hard", "easy",
                  "old", "slow", "misleading", "boring", "older", "horrible",
"complicated", "unsatisfying", "repetitive", "corny", "disappointed"]

# Declare an empty List
initial_word_list = []

# Load a Natural Language Processor object using a model
nlp = spacy.load("en_core_web_sm")

# For...Each loop
for review in hundred_reviews:
    doc = nlp(review)
    for token in doc:
        if token.text not in exclude:
            word = token.text
            initial_word_list.append(word)

# Invoke the Counter
frequency_dict = Counter(initial_word_list)
sorted_frequency_list = sorted(frequency_dict, key=lambda x: -frequency_dict[x])

text = " ".join(sorted_frequency_list)
scores = sia.polarity_scores(text) # Use of the Sentiment Intensity Analyzer object

# Interpret compound
compound = scores["compound"]
if compound >= 0.05:
    sentiment = "positive"
elif compound <= -0.05:
    sentiment = "negative"
else:
    sentiment = "neutral"

print(f"Compound: {compound:.3f}  Sentiment: {sentiment}")
print("Option 2:")
pos_count = 0
```

```
neg_count = 0

# Another way of doing Sentiment Analysis:
# 1. Count the number of Positive words
for word in initial_word_list:
    if word in positive_words:
        pos_count += 1
    if word in negative_words:
        neg_count += 1

print(f"Number of Positive Words in the 100 Reviews: {pos_count} Positivity
Percentage: {(pos_count / len(initial_word_list))*100} %")
print(f"Number of Negative Words in the same 100 Reviews: {neg_count}, Negativity
Percentage: {(neg_count / len(initial_word_list))*100} %")

print("# ================================================================")
print("# Word Definitions, Synonyms, and Antonyms")
print("# ================================================================")

'''
Function Definition
    Receives: A [word] in a List of Words
    Returns: [info]
'''
def get_word_info(word):
    info = {
        "word": word,
        "definitions": [],
        "synonyms": set(),
        "antonyms": set(),
        "examples": []
    }

    synsets = wn.synsets(word)
    for s in synsets:
        # definition and examples
        info["definitions"].append(f"{s.pos()} - {s.definition()}")
        for ex in s.examples():
            info["examples"].append(ex)

        # synonyms (lemma names) and antonyms
        for lemma in s.lemmas():
            info["synonyms"].add(lemma.name().replace("_", " "))
            for ant in lemma.antonyms():
                info["antonyms"].add(ant.name().replace("_", " "))

    info["synonyms"] = sorted(info["synonyms"])
    info["antonyms"] = sorted(info["antonyms"])
    info["definitions"] = list(dict.fromkeys(info["definitions"]))  # remove
duplicates, preserve order
    info["examples"] = list(dict.fromkeys(info["examples"]))
    return info

'''
Function Definition
    Receives: [info], which contains all pertinent information about a [word],
including:
        -Definition
        -Synonyms
        -Antonyms
    Returns: Nothing.
'''
def print_word_info(info):
    print(f"Word: {info['word']}")
    print("  Definitions:")
    if info["definitions"]:
        for d in info["definitions"]:
            print(f"   - {d}")
    else:
        print("   - (no definitions found)")

    print("  Synonyms:")
    print(f"   - {', '.join(info['synonyms']) if info['synonyms'] else '(none)'}")

    print("  Antonyms:")
    print(f"   - {', '.join(info['antonyms']) if info['antonyms'] else '(none)'}")

    print("  Examples:")
    if info["examples"]:
        for e in info["examples"]:
            print(f"   - {e}")
    else:
        print("   - (none)")
    print("-" * 40)

# Process the POSITIVE WORDS for Definitions, Synonyms, and Antonyms
# For...Each loop
for w in positive_words:
        info = get_word_info(w)
        print_word_info(info)

# Process the NEGATIVE WORDS for Definitions, Synonyms, and Antonyms
# For...Each loop
for w in negative_words:
        info = get_word_info(w)
        print_word_info(info)

print("# ============================================================")
print("# Word Cloud: Positive Words in Green and Negative Words In Red")
print("# ============================================================")
new_list = []

# Build a List of just the Positive and Negative Words
for word in initial_word_list:
    if word in positive_words:
        new_list.append(word)
    if word in negative_words:
        new_list.append(word)

# WordCloud function designating the color the words should appear in
def two_group_color_func(word, font_size, position, orientation, random_state=None,
**kwargs):
    if word in negative_words:
        return "rgb(200,0,0)"    # red
    if word in positive_words:
        return "rgb(0,150,0)"    # green
    return "black"
```

```
frequencies = Counter(new_list)

# Generate the word cloud
wc = WordCloud(width=400, height=400, background_color='white', max_font_size=70)
wc.generate_from_frequencies(frequencies)
wc.recolor(color_func=two_group_color_func)

#Display the Word Cloud
plt.figure(figsize=(5, 5))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.tight_layout()
plt.show()

print("# =============================================================")
print("# Named Entity Recognition and Similarity Detection")
print("# =============================================================")
nlp = spacy.load("en_core_web_sm")
# declare Document1 to be "Romeo and Juliet"
resp = requests.get(url)
soup = BeautifulSoup(resp.content, "html.parser")
text = soup.get_text(separator=" ")
doc = nlp(text).similarity(" ".join(positive_words))
doc = nlp(text).similarity(" ".join(negative_words))
```