WEB SCRAPING & SENTIMENT ANALYSIS

AIM: To Build an end-to-end machine learning project to conduct sentiment analysis on Steam game reviews.

Steam is the ultimate destination for playing, discussing, and creating games.

Here are some specific details of the project:

- The project uses Selenium to scrape reviews from the Steam website for the game Counter Strike 2.
- The data is cleaned and transformed using NLTK, including removing stop words and tokenizing the text.
- VADER is used to perform sentiment analysis on the reviews, and the results are stored in a new column called "polarity scores".
- A box plot is created to show the distribution of polarity scores for recommended and not recommended reviews.
- A word cloud is created to show the most common words used in the reviews.

What is Selenium?

Selenium is an open-source web automation tool that allows you to control a web browser through code. It's primarily used for automating browser testing, but it can also be effectively used for web scraping.

Why use Selenium for web scraping?

Selenium has several advantages over other web scraping tools:

- **Can handle dynamic content:** Selenium can execute JavaScript, which means it can scrape data from websites that rely on JavaScript to render content.
- **More flexible:** Selenium allows you to interact with web pages like a human user would, so you can scrape data from even complex websites.
- **Can handle different browsers:** Selenium supports a variety of web browsers, including Chrome, Firefox, and Edge.

Getting started with Selenium

1. **Install Selenium:**
   ```
   !pip install selenium
   ```

2. **Install a WebDriver:** You'll need to install a WebDriver for the browser you want to use.
   But since Safari has a built-in webdriver, I'll be using Safari

   ```
   !pip install webdriver-manager
   ```

3. **Write your Python script:**
   Here's a basic example of a Python script that uses Selenium to scrape the title of a web page:

Python
```python
from selenium import webdriver

# Create a webdriver instance
driver = webdriver.Chrome()

# Open the URL
driver.get("https://www.example.com")

# Get the title of the page
title = driver.title

# Print the title
print(title)

# Close the browser
driver.quit()
```

We can also use Selenium with other libraries like BeautifulSoup to parse the HTML content we scrape.

NLTK

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language. It supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.

VADER

VADER (Valence Aware Dictionary and sentiment Reasoner) is a pre-trained model that uses a lexicon and rules to analyse the sentiment of text. It's a Natural Language Toolkit (NLTK) module that provides sentiment scores based on the words used in a text. VADER can handle a variety of vocabularies, including abbreviations, capitalizations, repeated punctuations, and emoticons.

VADER uses a list of lexical features, such as words, that are labelled as positive or negative based on their semantic orientation. It uses a valence score for each word to determine its positivity or negativity. VADER is considered the best pre-trained sentiment classifier for social media conversations from networks such as Facebook or Twitter.

INTRODUCTION

This project aims to conduct sentiment analysis on Steam game reviews using an end-to-end machine learning pipeline. The process involves web scraping, data cleaning, exploratory data analysis (EDA), natural language processing (NLP), and sentiment analysis. The sentiment analysis is performed using the NLTK library, and the results are visualized using various plots.

LIBRARIES USED

- **Selenium:** For web scraping Steam game reviews.
- **Pandas:** For data manipulation and analysis.
- **NLTK:** For natural language processing tasks, such as tokenization, part-of-speech tagging, and sentiment analysis.
- **WordCloud**: For generating word clouds from the review text.
- **Plotly Express and Matplotlib:** For data visualization.
- **OS:** For operating system-related functionalities.

Code Structure

1. Web Scraping
   - The code starts by importing necessary libraries and defining the game ID and URL template.
   - Selenium is used to automate the web browser for navigating to the Steam game reviews page and scraping the data.

2. Data Cleaning
   - Extracted information includes review text, review rating, review length, play hours, and date posted.
   - The data is cleaned by removing unnecessary elements, converting date format, and saving it to a CSV file.

3. Exploratory Data Analysis (EDA)
   - Conducted EDA using bar plots to visualize the count of recommendations.

4. NLP and Sentiment Analysis
   - Text data is pre-processed by removing stopwords and tokenizing using NLTK.
   - Sentiment analysis is performed using the SentimentIntensityAnalyzer from NLTK.
   - Correlation between review sentiment and recommendation status is analysed.

5. Visualization
   - Box plots are created to visualize the distribution of polarity scores for Recommended and Not Recommended reviews.
   - Word cloud is generated to visually represent the most frequent words in the cleaned review text.

6. Documentation and Output
   - Proper documentation is provided throughout the code.
   - Generated outputs include a CSV file containing cleaned data and visualizations like bar plots, box plots, and a word cloud image.

UNDERSTANDING THE CODE

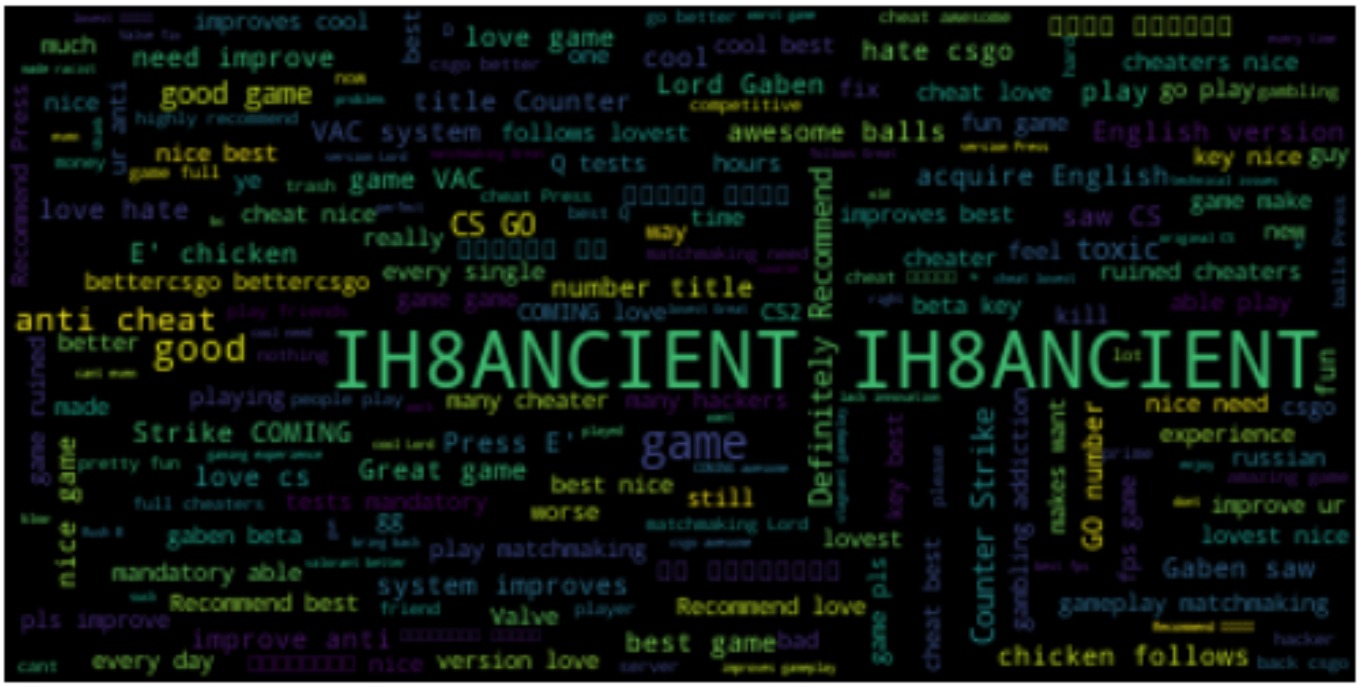
```
# Importing necessary libraries

from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException
from selenium import webdriver

import re
import time
from datetime import datetime
import pandas as pd

import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud

import plotly.express as px
import matplotlib.pyplot as plt
import os

# Defining the game id and url template

game_id = 730
url_template =
"https://steamcommunity.com/app/{}/reviews/?p=1&browsefilter=mostrecent&filterLanguage=english"
url = url_template.format(game_id)
```

```
# Initializing the webdriver

browser = webdriver.Safari()

# Navigating to the url

browser.get(url)

# Defining a function to get the current scroll position

def get_current_scroll_position(browser):
    return browser.execute_script("return window.pageYOffset;")

# Defining a function to scroll to the bottom of the page

def scroll_to_bottom(browser):
    browser.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(1)

# Defining a function to extract the steam id from a review card

def get_steam_id(card):
    profile_url =
card.find_element(By.XPATH,'.//div[@class="apphub_friend_block"]/a').get_attribute('href')
    steam_id = profile_url.split('/')[-1]
    return steam_id

# Defining a function to scrape review data from a review card

def scrape_review_data(card):
    date_posted_element =
card.find_element(By.XPATH,'.//div[@class="apphub_CardTextContent"]/div[@class="date_posted"]')
    date_posted = date_posted_element.text.strip()

    try:
        received_compensation_element =
card.find_element(By.CLASS_NAME,"received_compensation").text
    except NoSuchElementException:
        received_compensation_element = ""

    card_text_content_element = card.find_element(By.CLASS_NAME,"apphub_CardTextContent")
    review_content = card_text_content_element.text.strip()
    excluded_elements = [date_posted, received_compensation_element]

    for excluded_element in excluded_elements:
        review_content = review_content.replace(excluded_element, "")
    review_content = review_content.replace("\n", "").replace("\t", "")

    review_length = len(review_content.replace(" ", ""))

    thumb_text = card.find_element(By.XPATH,'.//div[@class="reviewInfo"]/div[2]').text
    play_hours = card.find_element(By.XPATH,'.//div[@class="reviewInfo"]/div[3]').text

    return review_content, thumb_text, review_length, play_hours, date_posted

# Initializing empty lists to store review data

reviews = []
steam_ids_set = set()

# Scraping review data from the webpage

try:
    last_position = get_current_scroll_position(browser)
    running = True
    while running:
        cards = browser.find_elements(By.CLASS_NAME, 'apphub_Card')

        for card in cards[-30:]:
            steam_id = get_steam_id(card)

            if steam_id in steam_ids_set:
                continue
            else:
                review = scrape_review_data(card)
                reviews.append(review)

            #steam_ids_set.add(steam_id)

        scroll_attempt = 0
        while scroll_attempt < max_scroll_attempts:
            scroll_to_bottom(browser)
            curr_position = get_current_scroll_position(browser)

            if curr_position == last_position:
                scroll_attempt+=1
                time_sleep(3)

                if curr_position >=3:
                    running = False
                    break

            else:
                last_position = curr_position
                break

except Exception as e:
    print(e)

# Closing the webdriver

finally:
    browser.quit()

# Creating a pandas dataframe from the scraped review data

df = pd.DataFrame(reviews, columns = ['ReviewText', 'Review','ReviewLength','PlayHours',
'DatePosted'])

# Printing the first few rows of the dataframe

df.head()

# Displaying the information about the dataframe

df.info()

# Printing the shape of the dataframe

df.shape

# Creating a bar plot of the count of recommendations

px.bar(df['Review'].value_counts(),
       title = 'Count of Recommendations')
```

```
# Cleaning the 'PlayHours' column

df['PlayHours'] = df['PlayHours'].str.replace("hrs on record", "")

# Displaying the 'PlayHours' column

df['PlayHours']

# Creating a dictionary to map month names to their corresponding numbers

month_mapping = {
    'January':'01',
    'February':'02',
    'March':'03',
    'April':'04',
    'May':'05',
    'June':'06',
    'July':'07',
    'August':'08',
    'September':'09',
    'October':'10',
    'November':'11',
    'December':'12'
}

# Extracting the day and month from the 'DatePosted' column

df[['Day', 'Month']] = df['DatePosted'].str.extract(r'(\d+) (\w+)', expand=True)

# Displaying the first few rows of the dataframe

df.head()

# Mapping the month names to their corresponding numbers

df['Month']=df['Month'].map(month_mapping)

# Displaying the first row of the dataframe

df.head(1)

# Creating a new 'DatePosted' column by combining the 'Day', 'Month', and a fixed year

df['DatePosted'] = df['Day'] + '/' + df['Month'] + '/2024'
df['DatePosted'] = pd.to_datetime(df['DatePosted'], format='%d/%m/%Y').dt.strftime('%d-%m-%Y')
df = df.drop(['Day', 'Month'], axis=1)

# Displaying the first few rows of the dataframe

df.head()

# Saving the dataframe to a csv file

df.to_csv('game_reviews.csv', encoding='utf-8', sep=';', index=False)

# Importing the 'stopwords' library from NLTK

import nltk
nltk.download('stopwords')

# Creating a set of English stopwords

stopwords = set(stopwords.words('english'))
```

```
# Defining a function to remove stopwords from a list of words

def remove_stopwords(sentences, stopwords):
    filtered_words = [item for item in sentences if item not in stopwords]
    return ' '.join(filtered_words)

# Applying the 'remove_stopwords' function to the 'ReviewText' column

df['cleanedReviewText'] = df['ReviewText'].astype(str).apply(lambda x:
remove_stopwords(x.split(), stopwords))

# Displaying the first few rows of the dataframe

df[['ReviewText', 'cleanedReviewText']]

# Tokenizing a sample review text

example = df['cleanedReviewText'][1]

# Tokenizing the review text using NLTK

from nltk.tokenize import word_tokenize
nltk.download('punkt')
tokens = nltk.word_tokenize(example)

# Displaying the tokens

tokens

# Part-of-speech tagging the tokens

nltk.download('averaged_perceptron_tagger')
tagged = nltk.pos_tag(tokens)

# Displaying the tagged tokens

tagged

# Importing the 'SentimentIntensityAnalyzer' class from NLTK

from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

# Creating an instance of the 'SentimentIntensityAnalyzer' class

sie = SentimentIntensityAnalyzer()

# Calculating the polarity score of a sample sentence

sie.polarity_scores('I love this game')

# Calculating the polarity scores of the 'cleanedReviewText' column

df['PolarityScores'] = [sie.polarity_scores(x)['compound'] for x in df['cleanedReviewText']]

# Displaying the first few rows of the dataframe

df.head()

# Create a new column 'ReviewValue' in the dataframe 'df' and assign a value of 1 to the rows
where the 'Review' column has the value 'Recommended', and 0 otherwise. This creates a new
column that can be used to analyze correlation between the review text and the review value.
df['ReviewValue'] = df['Review'].replace({'Recommended':1, 'Not Recommended':0})
```

```
# Calculate the Pearson correlation coefficient between the 'ReviewValue' and 'PolarityScores'
columns of the dataframe 'df'. This will give us a measure of the strength and direction of the
linear relationship between the two variables.
df[['ReviewValue', 'PolarityScores']].corr(method='pearson')

# Create a box plot using the Plotly Express library to visualize the distribution of polarity
scores for the two categories of reviews (Recommended and Not Recommended). This can help us
identify any differences in the sentiment of the reviews between the two categories.
px.box(df,x='Review', y='PolarityScores')

# Define a function 'plot_wordcloud' that takes in a pandas series and an optional output
filename, and generates a word cloud image from the series. The function uses the WordCloud
library to create the word cloud and saves it to a file with the specified output filename.
def plot_wordcloud(series, output_filename='wordcloud'):
    wordcloud = WordCloud().generate(' '.join(series.astype(str)))
    wordcloud.to_file(output_filename + '.png')
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")

# Generate a word cloud image from the 'cleanedReviewText' column of the dataframe 'df' using
the 'plot_wordcloud' function defined earlier. The word cloud image is saved to a file named
'wordcloud.png'.
plot_wordcloud(df['cleanedReviewText'])
```



Conclusion

This end-to-end machine learning project demonstrates the process of collecting, cleaning, and analysing Steam game reviews to perform sentiment analysis. The insights gained from this analysis can be valuable for understanding user sentiments and preferences for a specific game on the Steam platform.