Bilenkin540Weeks_9_&_10_Exercises

May 8, 2025

Activity 7.01 – page 388

1. Import the necessary libraries, including regex and BeautifulSoup.

[610]:
```
# Importing necessary libraries
import re
import requests
from bs4 import BeautifulSoup
```

2. Read the HTML from the URL.

[611]:
```
# Requesting the top books from Gutenberg
url = "https://www.gutenberg.org/browse/scores/top"
response = requests.get(url)
```

3. Write a small function to check the status of the web request.

[612]:
```
# Function to check the status of the web request and print the result
def check_web_status(response):
    if response.status_code == 200:
        print("Request was successful.")
    else:
        print(f"Request failed with status code: {response.status_code}")

check_web_status(response)
```

```
Request was successful.
```

4. Decode the response and pass this on to BeautifulSoup for HTML parsing.

[613]:
```
# Decoding the HTML content and parsing it with BeautifulSoup
html_content = response.content.decode('utf-8')
soup = BeautifulSoup(html_content, 'html.parser')
```

5. Find all the href tags and store them in the list of links. Check what the list looks like – print the first 30 elements.

[614]:
```
# Extracting all href attributes from anchor tags
links = [a['href'] for a in soup.find_all('a', href=True)]

# Printing the first 30 links to inspect the structure
```

```
print(links[:30])
```

```
['/', '/about/', '/about/', '/policy/collection_development.html',
'/about/contact_information.html', '/about/background/',
'/policy/permission.html', '/policy/privacy_policy.html',
'/policy/terms_of_use.html', '/ebooks/', '/ebooks/', '/ebooks/categories',
'/ebooks/bookshelf/', '/browse/scores/top', '/ebooks/offline_catalogs.html',
'/help/', '/help/', '/help/copyright.html', '/help/errata.html',
'/help/file_formats.html', '/help/faq.html', '/policy/',
'/help/public_domain_ebook_submission.html',
'/help/submitting_your_own_work.html', '/help/mobile.html', '/attic/',
'/donate/', '/donate/', 'pretty-pictures', '#books-last1']
```

6. Use a regular expression to find the numeric digits in these links. These are the file numbers for the top 100 eBooks.

[615]:
```
# Initializing a list to store all the file numbers as eBook IDs
ebook_ids = []

# Using a loop to extract numeric digits from links
for link in links:
    match = re.findall(r'/ebooks/(\d+)', link)
    if match:
        ebook_ids.append(match[0])

# Printing the first 10 eBook IDs as a sample
print(ebook_ids[:10])
```

```
['84', '2701', '76038', '1342', '2542', '1513', '11', '64317', '844', '43']
```

7. Initialize the empty list to hold the file numbers over an appropriate range and use regex to find the numeric digits in the link href string. Hint: Use the findall method.

[616]:
```
# Initializing a list to store unique eBook file numbers
unique_ebook_ids = []

# Using regex to extract numbers from links and ensure uniqueness
for link in links:
    numbers = re.findall(r'/ebooks/(\d+)', link)
    for num in numbers:
        if num not in unique_ebook_ids:
            unique_ebook_ids.append(num)

# Printing the first 10 unique eBook IDs as a sample
print(unique_ebook_ids[:10])
```

```
['84', '2701', '76038', '1342', '2542', '1513', '11', '64317', '844', '43']
```

8. What does the soup object's text look like? Use the .text method and print only the first 2,000 characters (do not print the whole thing, as it is too long).

[617]:
```
# Cleaning up the text by removing excessive blank lines and whitespace
cleaned_text = "\n".join([line.strip() for line in soup_text.splitlines() if␣
 ↪line.strip()])

# Printing the first 2000 characters of the cleaned text
print(cleaned_text[:2000])
```

```
Top 100 | Project Gutenberg
Menu
About


About Project Gutenberg
Collection Development
Contact Us
History & Philosophy
Permissions & License
Privacy Policy
Terms of Use
Search and Browse


Book Search
Main Categories
Bookshelves
Frequently Downloaded
Offline Catalogs
Help


All help topics →
Copyright How-To
Errata, Fixes and Bug Reports
File Formats
Frequently Asked Questions
Policies →
Public Domain eBook Submission
Submitting Your Own Work
Tablets, Phones and eReaders
The Attic →
Donate
Ways to donate
To determine the ranking we count the times each file gets downloaded.
Both HTTP and FTP transfers are counted.
Only transfers from ibiblio.org are counted as we have no access to our mirrors
log files.
Multiple downloads from the same IP address on the same day count as one
download.
```

```
IP addresses that download more than 100 files a day are considered
robots and are not considered.
Books made out of multiple files like most audio books are counted
if any file is downloaded.
Downloaded Books
2025-05-05696538
last 7 days4906850
last 30 days19447270
Pretty Pictures
Top 100 EBooks yesterday -
Top 100 Authors yesterday -
Top 100 EBooks last 7 days -
Top 100 Authors last 7 days -
Top 100 EBooks last 30 days -
Top 100 Authors last 30 days
Top 100 EBooks yesterday
Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley (5509)
Moby Dick; Or, The Whale by Herman Melville (3268)
Pride and Prejudice by Jane Austen (2238)
A Doll's House : a play by Henrik Ibsen (2089)
The Great Gatsby by F. Scott  Fitzgerald (1993)
Romeo and Juliet by William Shakespeare (1962)
Alice's Adventures in Wonderland by Lewis Carroll (1776)
The Importance of Being Earnest: A Trivial Comedy for Serious People by Oscar
Wilde (1769)
The Strange Case of Dr. Jekyll and Mr. Hyde by Robert Louis Stevenson (1653)
The Picture of Dorian Gray by Oscar Wilde (1498)
Middlemarch by George Eliot (1407)
Dracula by Bram Stoker (1379)
The Complete Works of William Shakespeare by William Shakespear
```

9. Search in the extracted text (using a regular expression) from the soup object to find the names of the top 100 eBooks (yesterday's ranking).

[618]:
```
# Using a regular expression to extract the titles of the Top 100 eBooks from␣
 ↪the cleaned soup text
# Searching for lines like: "Title by Author (DownloadCount)"
top_100_matches = re.findall(r'^(.*?) by .*? \(\d+\)$', cleaned_text, re.
 ↪MULTILINE)

# Printing the first 10 titles as a sample
print(top_100_matches[:10])
```

```
['Frankenstein; Or, The Modern Prometheus', 'Moby Dick; Or, The Whale', 'Pride
and Prejudice', "A Doll's House : a play", 'The Great Gatsby', 'Romeo and
Juliet', "Alice's Adventures in Wonderland", 'The Importance of Being Earnest: A
Trivial Comedy for Serious People', 'The Strange Case of Dr. Jekyll and Mr.
Hyde', 'The Picture of Dorian Gray']
```

10. Create a starting index. It should point at the text Top 100 Ebooks yesterday. Use the splitlines method of soup.text. It splits the lines of text of the soup object.

[619]:
```
# Splitting the cleaned text into lines
lines = cleaned_text.splitlines()

# Finding the starting index for the "Top 100 EBooks yesterday" section
start_index = lines.index("Top 100 EBooks yesterday")

# Printing the index to confirm
print("Starting index for Top 100 eBooks yesterday:", start_index)
```

```
Starting index for Top 100 eBooks yesterday: 54
```

11. Run the for loop 1-100 to add the strings of the next 100 lines to this temporary list. Hint: use the splitlines method.

[620]:
```
# Initializing an empty list to store the next 100 eBook lines
top_100_lines = []

# Looping through the next 100 lines starting after the "Top 100 EBooks␣
 ↪yesterday" index
for i in range(start_index + 1, start_index + 101):
    top_100_lines.append(lines[i])

# Printing the first 5 lines to verify
print(top_100_lines[:5])
```

```
['Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley
(5509)', 'Moby Dick; Or, The Whale by Herman Melville (3268)', 'Pride and
Prejudice by Jane Austen (2238)', "A Doll's House : a play by Henrik Ibsen
(2089)", 'The Great Gatsby by F. Scott  Fitzgerald (1993)']
```

12. Use a regular expression to extract only text from the name strings and append it to an empty list. Use match and span to find the indices and use them.

[621]:
```
# Initializing an empty list to store just the book titles
top_100_titles = []

# Using regex to extract the title from each line
for line in top_100_lines:
    match = re.match(r'^(.*?) by .*? \(\d+\)$', line)
    if match:
        title = match.group(1)
        top_100_titles.append(title)

# Printing the first 10 titles as a sample
print(top_100_titles[:10])
```

```
['Frankenstein; Or, The Modern Prometheus', 'Moby Dick; Or, The Whale', 'Pride
```

```
and Prejudice', "A Doll's House : a play", 'The Great Gatsby', 'Romeo and
Juliet', "Alice's Adventures in Wonderland", 'The Importance of Being Earnest: A
Trivial Comedy for Serious People', 'The Strange Case of Dr. Jekyll and Mr.
Hyde', 'The Picture of Dorian Gray']
```

13. Print the list of titles.

[622]:
```
# Printing each title on a new line
for title in top_100_titles:
    print(title)
```

```
Frankenstein; Or, The Modern Prometheus
Moby Dick; Or, The Whale
Pride and Prejudice
A Doll's House : a play
The Great Gatsby
Romeo and Juliet
Alice's Adventures in Wonderland
The Importance of Being Earnest: A Trivial Comedy for Serious People
The Strange Case of Dr. Jekyll and Mr. Hyde
The Picture of Dorian Gray
Middlemarch
Dracula
The Complete Works of William Shakespeare
A Room with a View
Little Women; Or, Meg, Jo, Beth, and Amy
Crime and Punishment
The Blue Castle: a novel
Simple Sabotage Field Manual
The Enchanted April
The dinky ducklings
Metamorphosis
The Adventures of Ferdinand Count Fathom - Complete
Cranford
History of Tom Jones, a Foundling
Jane Eyre: An Autobiography
A Modest Proposal
The Adventures of Roderick Random
The Expedition of Humphry Clinker
A Tale of Two Cities
Twenty years after
Great Expectations
My Life - Volume 1
Thus Spake Zarathustra: A Book for All and None
Adventures of Huckleberry Finn
Heart of Darkness
Frankenstein; Or, The Modern Prometheus
The Yellow Wallpaper
The Scarlet Letter
```

Leviathan
The Adventures of Sherlock Holmes
Peppermint
The Souls of Black Folk
The pillow-book of Sei Shōnagon
The Brothers Karamazov
Grimms' Fairy Tales
Anne of Green Gables
The Prince
Don Quijote
Wuthering Heights
Ulysses
Second Treatise of Government
The Count of Monte Cristo
The Iliad
On Liberty
Honesty the best policy :
The Adventures of Tom Sawyer, Complete
White Nights and Other Stories
Beyond Good and Evil
A Christmas Carol in Prose; Being a Ghost Story of Christmas
Walden, and On The Duty Of Civil Disobedience
The Confessions of St. Augustine
The Republic
War and Peace
Tractatus Logico-Philosophicus
The Odyssey
Moby Word Lists
The divine comedy
The Philippines a Century Hence
The Reign of Greed
Plays
Les Misérables
Du côté de chez Swann
Treasure Island
The Works of Edgar Allan Poe - Volume 2
Dubliners
The Hound of the Baskervilles
The King in Yellow
A Study in Scarlet
Meditations
The Romance of Lust: A classic Victorian erotic novel
Woman-through a man's eyeglass
Little Women
Oliver Twist
The Wonderful Wizard of Oz
Gulliver's Travels into Several Remote Nations of the World
The Art of War

```
The Legend of Sleepy Hollow
The Tragical History of Doctor Faustus
Anna Karenina
Spoon River Anthology
The History of the Peloponnesian War
Josefine Mutzenbacher
Oedipus King of Thebes
Macbeth
The Prophet
Hard Times
The Kama Sutra of Vatsyayana
Carmilla
```

Activity 7.02 – page 390

1. Import urllib.request, urllib.parse, urllib.error, and json.

[623]:
```python
# Importing necessary libraries
import urllib.request
import urllib.parse
import urllib.error
import json
import os
```

2. Load the secret API key from a JSON file.

[624]:
```python
import json
import os

# Checking if the file exists
if not os.path.exists("APIkeys.json"):
    # If the file doesn't exist, creating it and writing the key
    with open("APIkeys.json", "w") as f:
        json.dump({"OMDB_API_KEY": "e8741595"}, f)
        print("APIkeys.json created and key written.")
else:
    # If the file exists, checking if the key exists
    with open("APIkeys.json", "r") as f:
        keys = json.load(f)

    if "OMDB_API_KEY" not in keys:
        # If the key is missing, updating the file
        keys["OMDB_API_KEY"] = "e8741595"
        with open("APIkeys.json", "w") as f:
            json.dump(keys, f)
        print("API key added to existing file.")
    else:
        print("API key already exists.")
```

```
API key added to existing file.
```

3. Obtain a key and store it in a JSON file as APIkeys.json.

[625]:
```
import json

# The API key obtained from OMDb
api_key = "e8741595"

# Storing the API key in a dictionary
api_data = {"apikey": api_key}

# Writing the dictionary to a JSON file
with open('APIkeys.json', 'w') as json_file:
    json.dump(api_data, json_file)

print("API key stored successfully in 'APIkeys.json'")
```

```
API key stored successfully in 'APIkeys.json'
```

4. Open the APIkeys.json file.

[626]:
```
import json

# Opening and reading the APIkeys.json file to load the API key
with open('APIkeys.json', 'r') as json_file:
    api_data = json.load(json_file)

# Extracting the API key from the loaded data
api_key = api_data["apikey"]

print("OMDb API key loaded successfully:", api_key)
```

```
OMDb API key loaded successfully: e8741595
```

5. Assign the OMDb portal (http://www.omdbapi.com/?) as a string to a variable.

[627]:
```
# Assigning the OMDb portal base URL to a variable
serviceurl = 'http://www.omdbapi.com/'

# Confirming the URL has been assigned
print("OMDb portal URL:", serviceurl)
```

```
OMDb portal URL: http://www.omdbapi.com/
```

6. Create a variable called apikey with the last portion of the URL(&apikey=secretapikey), where secretapikey is your own API key.

[628]:
```
# Creating a variable for your API key
apikey = f'&apikey={api_key}'

# Confirming the API key variable
print("API key:", apikey)
```

API key: &apikey=e8741595

7. Write a utility function called print_json to print the movie data from a JSONfile (which we will get from the portal).

[629]:
```
# Utility function to print JSON data in a readable format
def print_json(data):
    print(json.dumps(data, indent=4))
```

8. Write a utility function to download a poster of the movie based on the information from the JSON dataset and save it in local folder.

[630]:
```
import os
import urllib.request

def download_poster(data, folder="posters"):
    if 'Poster' not in data or data['Poster'] == 'N/A':
        print("No poster found for this movie.")
        return

    # Creating folder if it doesn't exist
    if not os.path.exists(folder):
        os.makedirs(folder)

    poster_url = data['Poster']
    poster_filename = os.path.join(folder, data['Title'].replace(" ", "_") + ".
 ↪jpg")

    try:
        urllib.request.urlretrieve(poster_url, poster_filename)
        print(f"Poster saved successfully as: {poster_filename}")
    except Exception as e:
        print("Failed to download poster:", e)
```

9. Write a utility function called search_movie to search for a movie by its name, print the downloaded JSON data, and save the movie poster in the local folder.

[631]:
```
def search_movie(title):
    try:
        # Constructing the full URL with query parameters
        params = {'t': title}
        query_string = urllib.parse.urlencode(params)
        full_url = serviceurl + '?' + query_string + apikey

        # Making the request and reading the response
        print(f"Requesting URL: {full_url}")
        with urllib.request.urlopen(full_url) as response:
            data = response.read().decode()
            json_data = json.loads(data)
```

```
        # Checking if the response is successful
        if json_data.get("Response") == "True":
            print_json(json_data)
            download_poster(json_data, folder='posters')
        else:
            print("Error:", json_data.get("Error"))

    except Exception as e:
        print("An error occurred:", e)
```

10. Test the search_movie function by entering Titanic.

[632]:
```
search_movie("Titanic")
```

```
Requesting URL: http://www.omdbapi.com/?t=Titanic&apikey=e8741595
{
    "Title": "Titanic",
    "Year": "1997",
    "Rated": "PG-13",
    "Released": "19 Dec 1997",
    "Runtime": "194 min",
    "Genre": "Drama, Romance",
    "Director": "James Cameron",
    "Writer": "James Cameron",
    "Actors": "Leonardo DiCaprio, Kate Winslet, Billy Zane",
    "Plot": "A seventeen-year-old aristocrat falls in love with a kind but poor
artist aboard the luxurious, ill-fated R.M.S. Titanic.",
    "Language": "English, Swedish, Italian, French",
    "Country": "United States",
    "Awards": "Won 11 Oscars. 126 wins & 83 nominations total",
    "Poster": "https://m.media-amazon.com/images/M/MV5BYzYyN2FiZmUtYWYzMy00MzViL
WJkZTMtOGY1ZjgzNWMwN2YxXkEyXkFqcGc@._V1_SX300.jpg",
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "7.9/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "88%"
        },
        {
            "Source": "Metacritic",
            "Value": "75/100"
        }
    ],
    "Metascore": "75",
```

```
    "imdbRating": "7.9",
    "imdbVotes": "1,334,872",
    "imdbID": "tt0120338",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$674,354,882",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
Poster saved successfully as: posters\Titanic.jpg
```

11. Test the search_movie function by entering Random_error and retrieve the data for Random_error.

[633]:
```
print(search_movie("Random_error"))
```

```
Requesting URL: http://www.omdbapi.com/?t=Random_error&apikey=e8741595
Error: Movie not found!
None
```

3. Connect to an API of your choice and do a simple data pull - you can use any API - except the API you have selected for your project.

a. Importing required libraries and setting up parameters.

[634]:
```
import requests

# My API key
api_key = '685a27e8d3ae37f288db402b5d7debf8'

# City I want to get the weather for
city_name = 'New York'

# Base URL for the OpenWeather API
base_url = 'https://api.openweathermap.org/data/2.5/weather'

# Parameters to send with the GET request
params = {
    'q': city_name,
    'appid': api_key,
    'units': 'imperial'
}
```

b. Connect to the API and do a "Get" call/operation on the API to return a subset of data from the API

[635]:
```
response = requests.get(base_url, params=params)

# Handling the response
```

```
if response.status_code == 200:
    data = response.json()
    print(f"Weather in {data['name']}, {data['sys']['country']}:")
    print(f"Temperature: {round(data['main']['temp'])}°F")
    print(f"Weather: {data['weather'][0]['description'].capitalize()}")
    print(f"Humidity: {data['main']['humidity']}%")
    print(f"Wind Speed: {round(data['wind']['speed'], 1)} m/s")
else:
    print(f"Error fetching data: {response.status_code} - {response.reason}")
```

```
Weather in New York, US:
Temperature: 62°F
Weather: Broken clouds
Humidity: 84%
Wind Speed: 14 m/s
```

4. Using one of the datasets provided in Weeks 7 & 8, or a dataset of your own, choose 3 of the following visualizations to complete.

Loading the Dataset

[636]:
```
import pandas as pd

# Loading the Excel file
file_path = r"C:\Users\maxim\OneDrive\Desktop\BU\DSC␣
 ↪540\CANDY-HIERARCHY-2015-SURVEY-Responses.xlsx"
df = pd.read_excel(file_path)

# Displaying the first 2 rows
df.head(2)
```

[636]:

	Timestamp	How old are you?	Are you going actually going trick or treating yourself?	[Butterfinger]	[100 Grand Bar]	[Anonymous brown globs that come in black and orange wrappers]	[Any full-sized candy bar]	[Black Jacks]	[Bonkers]	[Bottle Caps]	…
0	2015-10-23 08:46:20.451	35	No	JOY	NaN	DESPAIR					
1	2015-10-23 08:46:51.583	41	No	JOY	JOY	DESPAIR					

```
0                         JOY            NaN        NaN            NaN  …
1                         JOY        DESPAIR    DESPAIR            JOY  …

   [Necco Wafers] Which day do you prefer, Friday or Sunday?  \
0             NaN                                            NaN
1         DESPAIR                                            NaN

  Please estimate the degrees of separation you have from the following folks
[Bruce Lee]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[JK Rowling]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[Malala Yousafzai]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[Thom Yorke]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[JJ Abrams]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[Hillary Clinton]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[Donald Trump]  \
0                                                 NaN
1                                                 NaN

  Please estimate the degrees of separation you have from the following folks
[Beyoncé Knowles]
0                                                 NaN
1                                                 NaN
```

```
[2 rows x 124 columns]
```

Cleaning Column Names

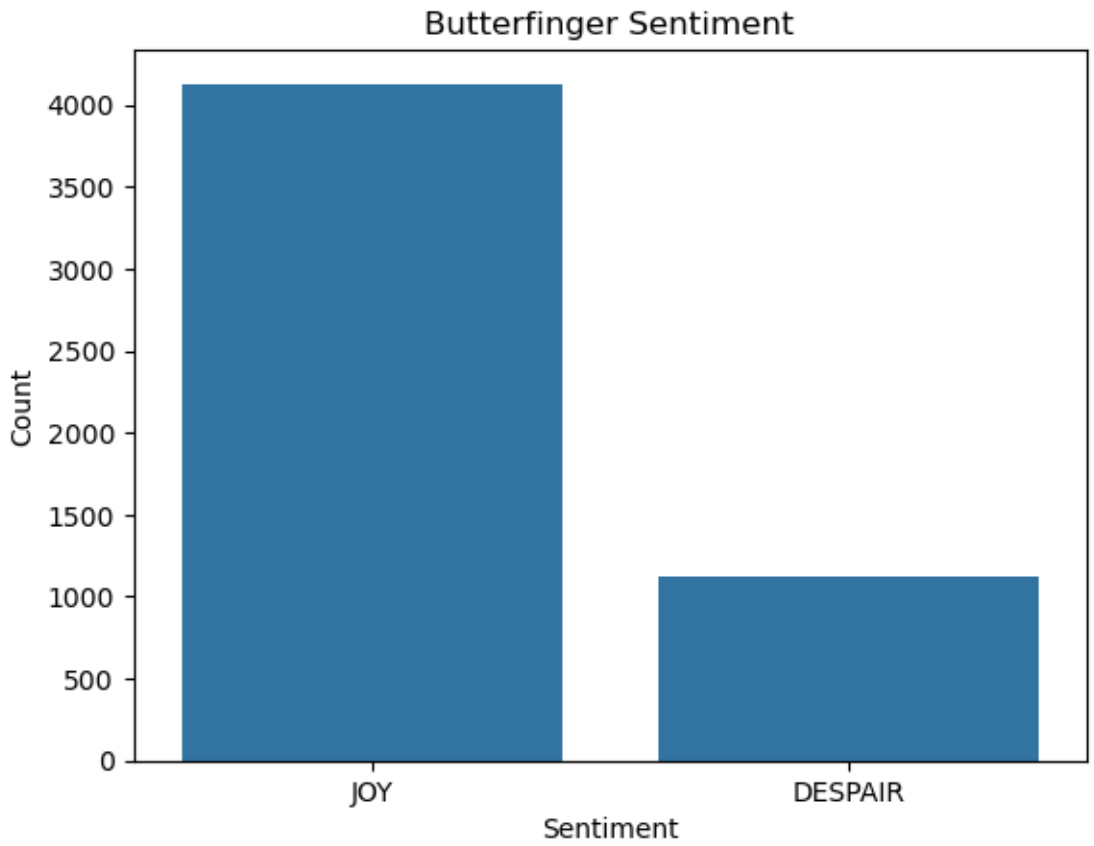
[637]:
```
df.columns = df.columns.str.strip()
```

1. Bar Plot – Butterfinger Ratings

[638]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Dropping NaN values and count responses for Butterfinger
butterfinger_counts = df['[Butterfinger]'].dropna().value_counts()

# Creating a bar plot
sns.barplot(x=butterfinger_counts.index, y=butterfinger_counts.values)
plt.title('Butterfinger Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



Butterfinger Sentiment Bar Chart

As we can see from the Butterfinger sentiment bar chart above, Butterfinger received an overwhelming number of "Joy" responses compared to "Despair." More than 4,000 participants expressed joy, while approximately 1,000 expressed despair. This chart clearly visualizes how survey participants responded to the candy Butterfinger. We can conclude that Butterfinger is a very likable candy, as it was more often selected as a preferred option.

2. Scatter Plot

Stripping any leading or trailing spaces and removing any square brackets ([ ]) from column names.

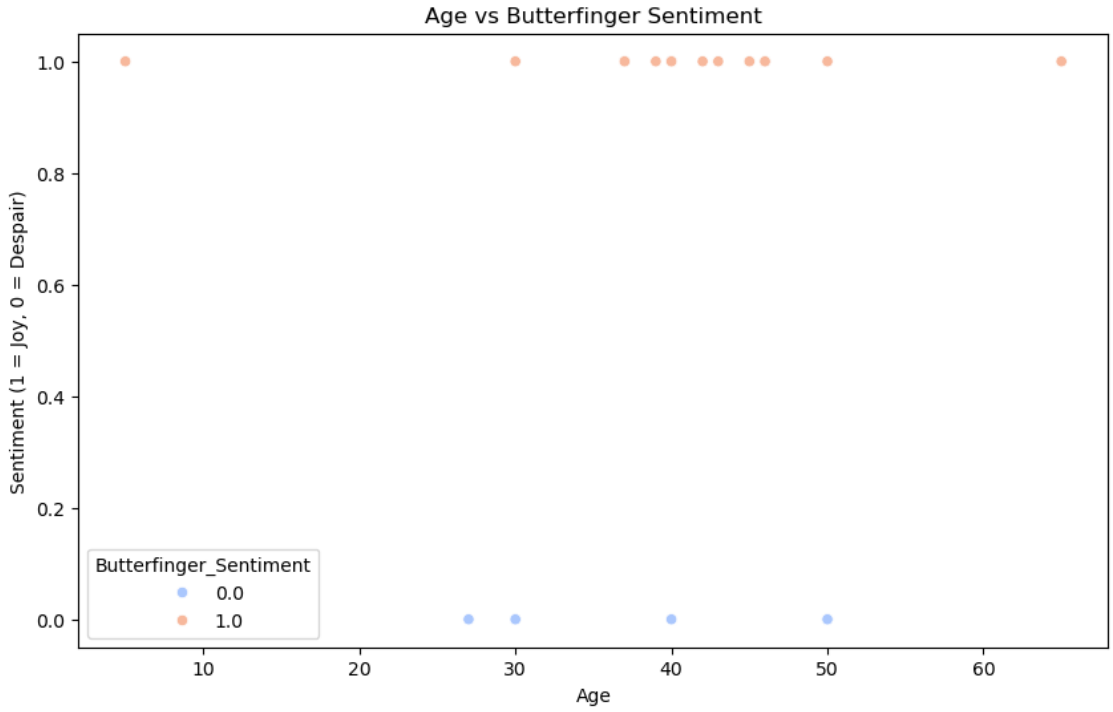
[639]:
```
# Stripping any leading/trailing spaces and removing unnecessary characters␣
↪like brackets in column names
df.columns = df.columns.str.strip().str.replace(r'[[]', '', regex=True).str.
↪replace(r'[]]', '', regex=True)
```

Removing non-numeric characters from the "How old are you?" column and converting the values to numeric format for analysis.

[640]:
```
# Removing non-numeric characters and converting age to numeric
df['Age_Clean'] = df['How old are you?'].str.extract(r'(\d+)')  # Extracting␣
↪digits
df['Age_Clean'] = pd.to_numeric(df['Age_Clean'], errors='coerce')  # Converting␣
↪to numeric
```

[641]:
```
# Mapping 'Joy' and 'Despair' to numerical values for plotting
df['Butterfinger_Sentiment'] = df['Butterfinger'].map({'JOY': 1, 'DESPAIR': 0})

# Scatter plot: Age vs Butterfinger sentiment (Joy = 1, Despair = 0)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['Age_Clean'], y=df['Butterfinger_Sentiment'],
                hue=df['Butterfinger_Sentiment'], palette='coolwarm',␣
↪marker='o')
plt.title('Age vs Butterfinger Sentiment')
plt.xlabel('Age')
plt.ylabel('Sentiment (1 = Joy, 0 = Despair)')
plt.show()
```



Cleaning Age Data

To ensure all values in the "How old are you?" column are numerical, we extracted only the numeric age values. This step helps remove non-numerical entries such as "30's" or "old enough" and prepares the data for numerical operations like scatter plots.

3. Pie Chart

Creating a pie chart to visualize the proportion of survey respondents who expressed "Joy" or "Despair" for the Butterfinger candy. This chart makes it easy to see the relative share of each sentiment category at a glance.

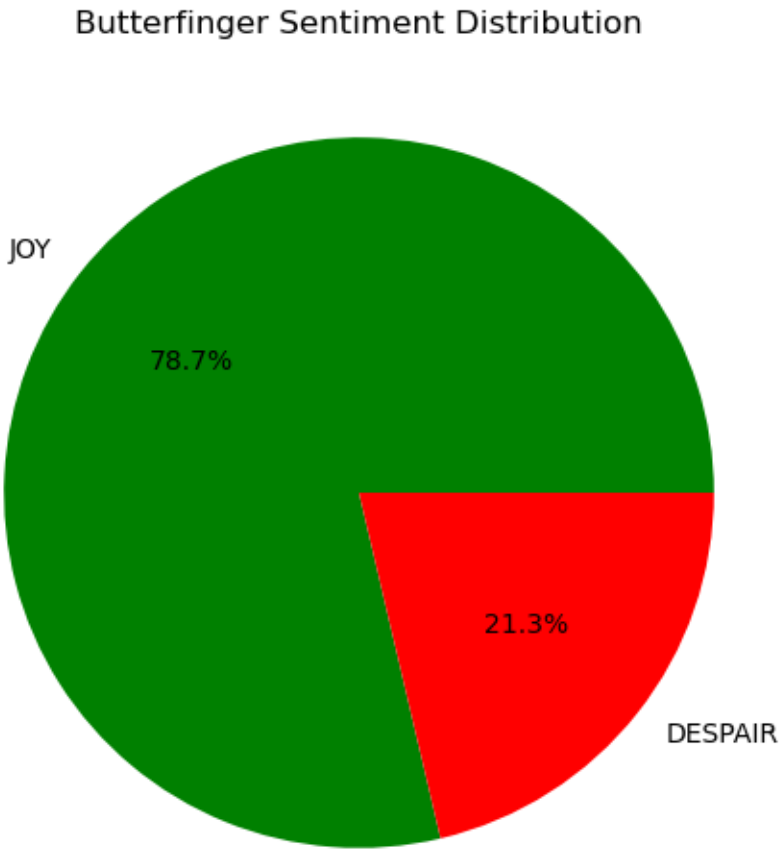
[642]:
```
import matplotlib.pyplot as plt

# Counting the number of each sentiment for Butterfinger
sentiment_counts = df['Butterfinger'].value_counts()

# Ensuring Joy is green and Despair is red
colors = ['green' if label == 'JOY' else 'red' for label in sentiment_counts.
 ↪index]

# Plotting the pie chart
plt.figure(figsize=(6, 6))
plt.pie(sentiment_counts, labels=sentiment_counts.index, autopct='%1.1f%%',␣
 ↪colors=colors)
```

```
plt.title('Butterfinger Sentiment Distribution')
plt.show()
```



Butterfinger Sentiment Pie Chart Summary

Using a pie chart for visualization, we can clearly see the proportions of Joy and Despair responses. The Joy responses, represented in green, occupy more than three-quarters of the pie, amounting to 78.7%. In contrast, the Despair responses, shown in red, make up 21.3%. Presenting the data in percentage form makes it easier to understand the scale of each group's sentiment toward Butterfinger.