WESTYN HILLIARD

1.0.1 1. The Data Wrangling Workshop: Activity 7.01 -

1. Import the necessary libraries, inluding REGEX and BeautifulSoup -

[209]:
```
import urllib.request, urllib.parse, urllib.error
import requests
from bs4 import BeautifulSoup
import ssl
import re
```

2. Read the HTML from the URL -

[210]:
```
# Define the URL for the top 100 books on Project Gutenberg
top100url = 'https://www.gutenberg.org/browse/scores/top'
response = requests.get(top100url)
```

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

```
[211]: # Function to check the status of the request
       def status(r):
           if r.status_code==200:
               print("Success!")
           else:
               print("Failed!")

       # Check the status of the response
       status(response)
```

```
Success!
```

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

```
[212]: # Decode the content of the response and create a BeautifulSoup object
       contents = response.content.decode(response.encoding)
       soup = BeautifulSoup(contents, '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 -

```
[213]: # Empty list to hold all the http links in the HTML page
       links=[]
       # Find all the 'a' tags and store their 'href' attributes in the list of links
       for link in soup.find_all('a'):
           #print(link.get('href'))
           links.append(link.get('href'))

       # Display the first 30 links for inspection
       links[:30]
```

```
[213]: ['/',
        '/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/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/',
        '#books-last1',
        '#authors-last1',
        '#books-last7']
```

6 & 7. Use a regular expression to find the numeric digits in these links. These are the file numbers for the top 100 eBooks. Initialize the empty list to hold the file numbers over an appropriate rangeand use regex to find the numeric digits in the link href string. Hint: Use thefindall method.

[214]:
```python
# Empty list to hold the book numbers
booknumber=[]

# Loop through a specific range of links to extract book numbers
for i in range(19,119):
    link = links[i].strip()
    # Regular expression to find the numeric digits in the link (href) string
    n=re.findall('[0-9]+',link)
    if len(n)==1:
        # Append the filenumber casted as integer
        booknumber.append(int(n[0]))

# Print the extracted book numbers
print ("\nThe file numbers for the top 100 ebooks on Gutenberg are shown␣
↪below\n"+"-"*70)
print(booknumber)
```

```
The file numbers for the top 100 ebooks on Gutenberg are shown below
----------------------------------------------------------------------
[1, 1, 7, 7, 30, 30, 1513, 2701, 145, 2641, 100, 1342, 37106, 30251, 67979,
16389, 6761, 394, 5197, 4085, 2160, 6593, 1259, 84, 8581, 11, 28054, 2554,
74057, 25282, 345, 5200, 2000, 24022, 50150, 174, 2600, 4300, 64317, 2542,
42629, 24869, 1661, 98, 1998, 1952, 76, 43, 1400, 135, 1232, 33283, 844, 73988,
6130, 1080, 2591, 46, 30254, 27827, 996, 2650, 1260, 74051, 8492, 219, 768,
1497, 4363, 45, 31552, 244, 1727, 514, 1184, 67098, 36034, 2680, 5740, 205,
8800, 74, 16119, 408, 16, 73984, 36, 600, 74049, 31284, 55, 2814]
```

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

[215]:
```python
# Print the first 2000 characters of the page's text content for inspection
print(soup.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
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

Frequently Viewed or Downloaded
These listings are based on the number of times each eBook gets downloaded.
Multiple downloads from the same Internet address on the same day count as one download, and addresses that download more than 100 eBooks in a day are considered robots and are not counted.

Downloaded Books
2024-07-16400886
last 7 days2562561
last 30 days10822503

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

Romeo and Juliet by William Shakespeare (2746)
Moby Dick; Or, The Whale by Herman Melville (2486)
Middlemarch by George Eliot (1885)
A Room with a View by E. M. Forster (1837)
The Complete Works of William Shakespeare by William Shakespeare (1821)
Pride and Prejudice by Jane Austen (1754)
Little Women; Or, Meg, Jo, Beth, and Amy by Louisa May Alcott (1752)
Disqualified by Charles L. Fontenay (1681)
The Blue Castle: a novel by L. M. Montgomery (1626)
The Enchanted April by Elizabeth Von Arnim (1617)
The Adventures of Ferdinand Count Fathom - Complete by T. Smollett (1523)
Cranford by Elizabeth Cleghorn Gaskell (1500)
My Life - Volume 1 by Richard Wagner (1481)
The Adventures of Roderick Random by T. Smollett (1477)
The Expedition of Humphry Clinker by T. Smollett (1471)
History of Tom Jones, a Foundling by Henry

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

[216]:
```python
# Temp empty list of Ebook names
list_temp=[]
```

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

[217]:
```python
# Find the starting index for the list of top 100 ebooks
start_idx=soup.text.splitlines().index('Top 100 EBooks yesterday')
```

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

[218]:
```python
# Append the titles of the top 100 ebooks to the temporary list
for i in range(100):
    list_temp.append(soup.text.splitlines()[start_idx+2+i])
```

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

[219]:
```python
# Empty list to hold cleaned book titles
list_titles=[]

# Extract only the title part of each line
for i in range(100):
    match = re.match(r'(.+?) by (.+)', list_temp[i])
    if match:
        title, author = match.groups()
        list_titles.append(f"{title.strip()} by {author.strip()}")
    else:
        list_titles.append(list_temp[i].strip())
```

13. Print the list of titles.

[220]:
```python
# Print the cleaned book titles
for l in list_titles:
    print(l)
```

```
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

Romeo and Juliet by William Shakespeare (2746)
Moby Dick; Or, The Whale by Herman Melville (2486)
```

Middlemarch by George Eliot (1885)
A Room with a View by E. M. Forster (1837)
The Complete Works of William Shakespeare by William Shakespeare (1821)
Pride and Prejudice by Jane Austen (1754)
Little Women; Or, Meg, Jo, Beth, and Amy by Louisa May Alcott (1752)
Disqualified by Charles L. Fontenay (1681)
The Blue Castle: a novel by L. M. Montgomery (1626)
The Enchanted April by Elizabeth Von Arnim (1617)
The Adventures of Ferdinand Count Fathom - Complete by T. Smollett (1523)
Cranford by Elizabeth Cleghorn Gaskell (1500)
My Life - Volume 1 by Richard Wagner (1481)
The Adventures of Roderick Random by T. Smollett (1477)
The Expedition of Humphry Clinker by T. Smollett (1471)
History of Tom Jones, a Foundling by Henry Fielding (1468)
Twenty years after by Alexandre Dumas and Auguste Maquet (1446)
Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley (1403)
The art of money getting : by P. T. Barnum (1170)
Alice's Adventures in Wonderland by Lewis Carroll (1109)
The Brothers Karamazov by Fyodor Dostoyevsky (975)
Crime and Punishment by Fyodor Dostoyevsky (738)
A book of bridges by Walter Shaw Sparrow (721)
The Jewish State by Theodor Herzl (714)
Dracula by Bram Stoker (702)
Metamorphosis by Franz Kafka (675)
Don Quijote by Miguel de Cervantes Saavedra (668)
A Christmas Carol by Charles Dickens (667)
The Devil is an Ass by Ben Jonson (656)
The Picture of Dorian Gray by Oscar Wilde (636)
War and Peace by graf Leo Tolstoy (585)
Ulysses by James Joyce (561)
The Great Gatsby by F. Scott Fitzgerald (531)
A Doll's House : a play by Henrik Ibsen (520)
The Violoncello and Its History by Wilhelm Joseph von Wasielewski (516)
The Rámáyan of Válmíki, translated into English verse by Valmiki (509)
The Adventures of Sherlock Holmes by Arthur Conan Doyle (505)
A Tale of Two Cities by Charles Dickens (496)
Thus Spake Zarathustra: A Book for All and None by Friedrich Wilhelm Nietzsche (476)
The Yellow Wallpaper by Charlotte Perkins Gilman (455)
Adventures of Huckleberry Finn by Mark Twain (452)
The Strange Case of Dr. Jekyll and Mr. Hyde by Robert Louis Stevenson (446)
Great Expectations by Charles Dickens (429)
Les Misérables by Victor Hugo (428)
The Prince by Niccolò Machiavelli (424)
Calculus Made Easy by Silvanus P. Thompson (422)
The Importance of Being Earnest: A Trivial Comedy for Serious People by Oscar Wilde (421)
Electro-episoded in A.D. 2025 by E. D. Skinner (415)

The Iliad by Homer (408)
A Modest Proposal by Jonathan Swift (391)
Grimms' Fairy Tales by Jacob Grimm and Wilhelm Grimm (391)
A Christmas Carol in Prose; Being a Ghost Story of Christmas by Charles Dickens (380)
The Romance of Lust: A classic Victorian erotic novel by Anonymous (379)
The Kama Sutra of Vatsyayana by Vatsyayana (360)
Don Quixote by Miguel de Cervantes Saavedra (359)
Du côté de chez Swann by Marcel Proust (357)
Jane Eyre: An Autobiography by Charlotte Brontë (356)
The chest of tools by Madeline Leslie (348)
The King in Yellow by Robert W. Chambers (348)
Heart of Darkness by Joseph Conrad (348)
Wuthering Heights by Emily Brontë (344)
The Republic by Plato (339)
Beyond Good and Evil by Friedrich Wilhelm Nietzsche (336)
Anne of Green Gables by L. M. Montgomery (331)
Novo dicionário da língua portuguesa by Cândido de Figueiredo (330)
A Study in Scarlet by Arthur Conan Doyle (329)
The Odyssey by Homer (324)
Little Women by Louisa May Alcott (321)
The Count of Monte Cristo by Alexandre Dumas and Auguste Maquet (320)
Winnie-the-Pooh by A. A. Milne (320)
White Nights and Other Stories by Fyodor Dostoyevsky (314)
Meditations by Emperor of Rome Marcus Aurelius (313)
Tractatus Logico-Philosophicus by Ludwig Wittgenstein (306)
Walden, and On The Duty Of Civil Disobedience by Henry David Thoreau (302)
The divine comedy by Dante Alighieri (297)
The Adventures of Tom Sawyer, Complete by Mark Twain (293)
Doctrina Christiana (291)
The Souls of Black Folk by W. E. B. Du Bois (288)
Peter Pan by J. M. Barrie (280)
Sinking of the "Titanic" : by Jay Henry Mowbray (278)
The War of the Worlds by H. G. Wells (274)
Notes from the Underground by Fyodor Dostoyevsky (274)
Anthropology by Franz Boas (273)
Josefine Mutzenbacher by Felix Salten (269)
The Wonderful Wizard of Oz by L. Frank Baum (268)
Dubliners by James Joyce (264)
Treasure Island by Robert Louis Stevenson (263)
Les misérables Tome I: Fantine by Victor Hugo (261)
The Prophet by Kahlil Gibran (258)
Daddy Takes Us to the Garden by Howard Roger Garis (257)
The riddle of Three-Way Creek by Ridgwell Cullum (252)
Leviathan by Thomas Hobbes (247)

1.0.2 2. The Data Wrangling Workshop: Activity 7.02, page 390 -

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

[221]:
```
# Import necessary libraries
import urllib.request
import urllib.parse
import urllib.error
import json
import os
```

2, 3, and 4. Load the secret API key (you have to get one from the OMDb website and usethat; it has a daily limit of 1,000 API keys) from a JSON file, stored in the samefolder, in a variable. Obtain a key and store it in a JSON file as APIkeys.json. Open the APIkeys.json file. -

[222]:
```
# Load the secret API key from the JSON file
try:
    with open('APIkeys.json', 'r') as file:
        keys = json.load(file)
        secret_api_key = keys['OMDb_API_key']
        print(f"API key loaded successfully: {secret_api_key}")
except FileNotFoundError:
    print("The file 'APIkeys.json' was not found. Please ensure it is in the␣
 ↪same directory as this script.")
    raise
except json.JSONDecodeError:
    print("The file 'APIkeys.json' is not in valid JSON format.")
    raise
except KeyError:
    print("The key 'OMDb_API_key' was not found in the JSON file.")
    raise

print("JSON file successfully loaded.")
```

```
API key loaded successfully: 9067128c
JSON file successfully loaded.
```

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

[223]:
```
# Assign the OMDb portal URL to a variable
omdb_url = 'http://www.omdbapi.com/?'
print(f"OMDb URL: {omdb_url}")
```

```
OMDb 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. -

[224]:
```
# Create a variable for the API key portion of the URL
apikey = f'&apikey={secret_api_key}'
```

```python
print(f"API key portion of URL: {apikey}")
```

```
API key portion of URL: &apikey=9067128c
```

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

[225]:
```python
# Utility function to print JSON data from a dictionary in a readable format
def print_json(data):
    print("Printing JSON data:")
    try:
        json_str = json.dumps(data, indent=4)
        print("JSON string created successfully:")
        print(json_str)
    except Exception as e:
        print(f"Error while creating JSON string: {e}")

# Utility function to print JSON data from a file in a readable format
def print_json_from_file(file_path):
    print("Loading JSON data from file and printing it:")
    try:
        with open(file_path, 'r') as file:
            data = json.load(file)
            print_json(data)  # Use the print_json function to print data
    except FileNotFoundError:
        print(f"The file '{file_path}' was not found.")
    except json.JSONDecodeError:
        print(f"The file '{file_path}' is not in valid JSON format.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Path to the JSON file
file_path = 'Data.json'

# Call the print_json_from_file function with the file path
print_json_from_file(file_path)
```

```
Loading JSON data from file and printing it:
Printing JSON data:
JSON string created successfully:
{
    "Title": "Guardians of the Galaxy Vol. 2",
    "Year": "2017",
    "Rated": "PG-13",
    "Released": "05 May 2017",
    "Runtime": "136 min",
    "Genre": "Action, Adventure, Comedy",
    "Director": "James Gunn",
    "Writer": "James Gunn, Dan Abnett, Andy Lanning",
    "Actors": "Chris Pratt, Zoe Saldana, Dave Bautista",
    "Plot": "The Guardians struggle to keep together as a team while dealing
with their personal family issues, notably Star-Lord's encounter with his
father, the ambitious celestial being Ego.",
    "Language": "English",
    "Country": "United States",
    "Awards": "Nominated for 1 Oscar. 15 wins & 60 nominations total",
    "Poster": "https://m.media-amazon.com/images/M/MV5BNjM0NTc0NzItM2FlYS00YzEwL
WE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzIyNzg@._V1_SX300.jpg",
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "7.6/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "85%"
        },
        {
            "Source": "Metacritic",
            "Value": "67/100"
        }
    ],
    "Metascore": "67",
    "imdbRating": "7.6",
    "imdbVotes": "764,992",
    "imdbID": "tt3896198",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$389,813,101",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
```

8. Write a utility function to download a poster of the movie based on theinformation from the JSON dataset and save it in your local folder. Use the osmodule. The poster data is stored in a JSON key called Poster. Use the openPython command to open a file and write the poster data. Close the file afteryou're done. This function will save the poster data as an image file. -

[226]:
```python
# Utility function to download and save the movie poster
def download_poster(data, save_folder='posters'):
    try:
        poster_url = data.get('Poster')
        if poster_url and poster_url != 'N/A':
            if not os.path.exists(save_folder):
                os.makedirs(save_folder)
            poster_filename = os.path.join(save_folder, f"{data['Title']}.jpg")
            print(f"Downloading poster from {poster_url} to {poster_filename}")

            urllib.request.urlretrieve(poster_url, poster_filename)
            print(f"Poster downloaded and saved as {poster_filename}")
        else:
            print("No poster available for this movie.")
    except Exception as e:
        print(f"An error occurred while downloading the poster: {e}")

# Call the download_poster function to download and save the poster from Data.
 ↪json
try:
    with open(file_path, 'r') as file:
        movie_data = json.load(file)
        download_poster(movie_data)
except FileNotFoundError:
    print(f"The file '{file_path}' was not found.")
except json.JSONDecodeError:
    print(f"The file '{file_path}' is not in valid JSON format.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Downloading poster from https://m.media-amazon.com/images/M/MV5BNjM0NTc0NzItM2F1
YS00YzEwLWE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzIyNzg@._V1_SX300.jpg to
posters/Guardians of the Galaxy Vol. 2.jpg
Poster downloaded and saved as posters/Guardians of the Galaxy Vol. 2.jpg
```

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.Use a try-except loop for this. Use the previously created serviceurl andapikey variables. You have to pass on a dictionary with a key, t, and the moviename as the corresponding value to the urllib.parse.urlencode()function and then add the serviceurl and apikey variables to the output ofthe function to construct the full URL. This URL will be used to access the data.The JSON data has a key called Response. If it is True, that means the readwas successful. Check this before processing the data. If it's not successful, thenprint the JSON key Error, which will contain the appropriate error messagereturned by the movie database. -

[227]:
```
# Utility function to search for a movie by its name, print the JSON data, and␣
 ↪save the movie poster
def search_movie(movie_name):
    serviceurl = 'http://www.omdbapi.com/?'

    params = {'t': movie_name}
    query_string = urllib.parse.urlencode(params)
    full_url = serviceurl + query_string + apikey  # Using previously defined␣
 ↪apikey
```

```
    try:
        print(f"Making request to URL: {full_url}")
        response = urllib.request.urlopen(full_url)
        raw_data = response.read().decode()
        print("Raw data retrieved from OMDb API:")
        print(raw_data)

        try:
            movie_data = json.loads(raw_data)
        except json.JSONDecodeError as e:
            print(f"Failed to decode JSON response: {e}")
            return

        if movie_data.get('Response') == 'True':
            print("Movie data retrieved successfully. Printing JSON:")
            print_json(movie_data)
            download_poster(movie_data)
        else:
            print(f"Error: {movie_data.get('Error')}")
    except urllib.error.URLError as e:
        print(f"Failed to make a request to the OMDb API: {e.reason}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Example usage
movie_name = 'Guardians of the Galaxy Vol. 2'
search_movie(movie_name)
```

```
Making request to URL:
http://www.omdbapi.com/?t=Guardians+of+the+Galaxy+Vol.+2&apikey=9067128c
Raw data retrieved from OMDb API:
{"Title":"Guardians of the Galaxy Vol.
2","Year":"2017","Rated":"PG-13","Released":"05 May 2017","Runtime":"136
min","Genre":"Action, Adventure, Comedy","Director":"James Gunn","Writer":"James
Gunn, Dan Abnett, Andy Lanning","Actors":"Chris Pratt, Zoe Saldana, Dave
Bautista","Plot":"The Guardians struggle to keep together as a team while
dealing with their personal family issues, notably Star-Lord's encounter with
his father, the ambitious celestial being
Ego.","Language":"English","Country":"United States","Awards":"Nominated for 1
Oscar. 15 wins & 60 nominations total","Poster":"https://m.media-amazon.com/imag
es/M/MV5BNjM0NTc0NzItM2FlYS00YzEwLWE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzI
yNzg@._V1_SX300.jpg","Ratings":[{"Source":"Internet Movie
Database","Value":"7.6/10"},{"Source":"Rotten Tomatoes","Value":"85%"},{"Source"
:"Metacritic","Value":"67/100"}],"Metascore":"67","imdbRating":"7.6","imdbVotes"
:"764,992","imdbID":"tt3896198","Type":"movie","DVD":"N/A","BoxOffice":"$389,813
,101","Production":"N/A","Website":"N/A","Response":"True"}
Movie data retrieved successfully. Printing JSON:
```

```
Printing JSON data:
JSON string created successfully:
```

```
{
    "Title": "Guardians of the Galaxy Vol. 2",
    "Year": "2017",
    "Rated": "PG-13",
    "Released": "05 May 2017",
    "Runtime": "136 min",
    "Genre": "Action, Adventure, Comedy",
    "Director": "James Gunn",
    "Writer": "James Gunn, Dan Abnett, Andy Lanning",
    "Actors": "Chris Pratt, Zoe Saldana, Dave Bautista",
    "Plot": "The Guardians struggle to keep together as a team while dealing
with their personal family issues, notably Star-Lord's encounter with his
father, the ambitious celestial being Ego.",
    "Language": "English",
    "Country": "United States",
    "Awards": "Nominated for 1 Oscar. 15 wins & 60 nominations total",
    "Poster": "https://m.media-amazon.com/images/M/MV5BNjM0NTc0NzItM2FlYS00YzEwL
WE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzIyNzg@._V1_SX300.jpg",
    "Ratings": [
        {
            "Source": "Internet Movie Database",
            "Value": "7.6/10"
        },
        {
            "Source": "Rotten Tomatoes",
            "Value": "85%"
        },
        {
            "Source": "Metacritic",
            "Value": "67/100"
        }
    ],
    "Metascore": "67",
    "imdbRating": "7.6",
    "imdbVotes": "764,992",
    "imdbID": "tt3896198",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$389,813,101",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
```

```
Downloading poster from https://m.media-amazon.com/images/M/MV5BNjM0NTc0NzItM2Fl
YS00YzEwLWE0YmUtNTA2ZWIzODc2OTgxXkEyXkFqcGdeQXVyNTgwNzIyNzg@._V1_SX300.jpg to
posters/Guardians of the Galaxy Vol. 2.jpg
```

```
Poster downloaded and saved as posters/Guardians of the Galaxy Vol. 2.jpg
```

10. Test the search_movie function by entering Titanic. -

[228]:
```
# Test the search_movie function with "Titanic"
movie_name = 'Titanic'
search_movie(movie_name)
```

```
Making request to URL: http://www.omdbapi.com/?t=Titanic&apikey=9067128c
Raw data retrieved from OMDb API:
{"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, Mexico","Awards":"Won 11 Oscars. 126 wins & 83 nominations
total","Poster":"https://m.media-amazon.com/images/M/MV5BMDdmZGU3NDQtY2E5My00ZTl
iLWIzOTUtMTY4ZGI1YjdiNjk3XkEyXkFqcGdeQXVyNTA4NzY1MzY@._V1_SX300.jpg","Ratings":[
{"Source":"Internet Movie Database","Value":"7.9/10"},{"Source":"Rotten Tomatoes
","Value":"88%"},{"Source":"Metacritic","Value":"75/100"}],"Metascore":"75","imd
bRating":"7.9","imdbVotes":"1,292,276","imdbID":"tt0120338","Type":"movie","DVD"
:"N/A","BoxOffice":"$674,292,608","Production":"N/A","Website":"N/A","Response":
"True"}
Movie data retrieved successfully. Printing JSON:
Printing JSON data:
JSON string created successfully:
{
    "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, Mexico",
    "Awards": "Won 11 Oscars. 126 wins & 83 nominations total",
    "Poster": "https://m.media-amazon.com/images/M/MV5BMDdmZGU3NDQtY2E5My00ZTliL
WIzOTUtMTY4ZGI1YjdiNjk3XkEyXkFqcGdeQXVyNTA4NzY1MzY@._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": "1,292,276",
    "imdbID": "tt0120338",
    "Type": "movie",
    "DVD": "N/A",
    "BoxOffice": "$674,292,608",
    "Production": "N/A",
    "Website": "N/A",
    "Response": "True"
}
```

```
Downloading poster from https://m.media-amazon.com/images/M/MV5BMDdmZGU3NDQtY2E5
My00ZTliLWIzOTUtMTY4ZGI1YjdiNjk3XkEyXkFqcGdeQXVyNTA4NzY1MzY@._V1_SX300.jpg to
posters/Titanic.jpg
Poster downloaded and saved as posters/Titanic.jpg
```

11. Test the search_movie function by entering Random_error and retrievethe data for Random_error (obviously, this will not be found, and you shouldbe able to check whether your error-catching code is working properly). -

[ ]:

```
```

[229]:

```python
# Test the search_movie function with "Random_error"
movie_name = 'Random_error'
search_movie(movie_name)
```

```
Making request to URL: http://www.omdbapi.com/?t=Random_error&apikey=9067128c
Raw data retrieved from OMDb API:
{"Response":"False","Error":"Movie not found!"}
Error: Movie not found!
```

1.0.3 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.

[230]:

```python
import requests
import json

# Replace 'your_api_key_here' with your actual NASA API key
api_key = '80eoThX05qTqbfrjCGzdC4FVGrcRFMenUtWtGbqG'
```

```python
base_url = 'https://api.nasa.gov/planetary/apod?'

# Construct the complete API call URL
complete_url = base_url + 'api_key=' + api_key

# Make the GET request to the NASA APOD API
response = requests.get(complete_url)

# Check the HTTP status code
if response.status_code == 200:
    # Parse the JSON response
    apod_data = response.json()

    # Print the JSON data in a readable format
    print(json.dumps(apod_data, indent=4))

    # Extract and print specific data
    date = apod_data['date']
    explanation = apod_data['explanation']
    title = apod_data['title']
    url = apod_data['url']

    print(f"Title: {title}")
    print(f"Date: {date}")
    print(f"Explanation: {explanation}")
    print(f"URL: {url}")
else:
    print(f"Failed to retrieve data from NASA APOD API. HTTP Status code:␣
  ↪{response.status_code}")
```

```
{
    "copyright": "\nGabriel Mu\u00f1oz;\nText: Natalia Lewandowska \n(SUNY
Oswego)\n",
    "date": "2024-07-17",
    "explanation": "When Vulcan, the Roman god of fire, swings his blacksmith's
hammer, the sky is lit on fire. A recent eruption of Chile's Villarrica volcano
shows the delicate interplay between this fire -- actually glowing steam and ash
from melted rock -- and the light from distant stars in our Milky Way galaxy and
the Magellanic Clouds galaxies. In the featured timelapse video, the Earth
rotates under the stars as Villarrica erupts.  With about 1350 volcanoes, our
planet Earth rivals Jupiter's moon Io as the most geologically active place in
the Solar System. While both have magnificent beauty, the reasons for the
existence of volcanoes on both worlds are different. Earth's volcanoes typically
occur between slowly shifting outer shell plates, while Io's volcanoes are
caused by gravitational flexing resulting from Jupiter's tidal gravitational
pull.",
    "media_type": "video",
    "service_version": "v1",
    "title": "Villarrica Volcano Against the Sky",
    "url": "https://www.youtube.com/embed/aX4ozspTPQY?rel=0"
}
Title: Villarrica Volcano Against the Sky
Date: 2024-07-17
Explanation: When Vulcan, the Roman god of fire, swings his blacksmith's hammer,
the sky is lit on fire. A recent eruption of Chile's Villarrica volcano shows
the delicate interplay between this fire -- actually glowing steam and ash from
melted rock -- and the light from distant stars in our Milky Way galaxy and the
Magellanic Clouds galaxies. In the featured timelapse video, the Earth rotates
under the stars as Villarrica erupts.  With about 1350 volcanoes, our planet
Earth rivals Jupiter's moon Io as the most geologically active place in the
Solar System. While both have magnificent beauty, the reasons for the existence
of volcanoes on both worlds are different. Earth's volcanoes typically occur
between slowly shifting outer shell plates, while Io's volcanoes are caused by
gravitational flexing resulting from Jupiter's tidal gravitational pull.
URL: https://www.youtube.com/embed/aX4ozspTPQY?rel=0
```

1.1 4. Using one of the datasets provided in Weeks 5 & 6, or a dataset of your own, choose 3 of the following visualizations to complete.

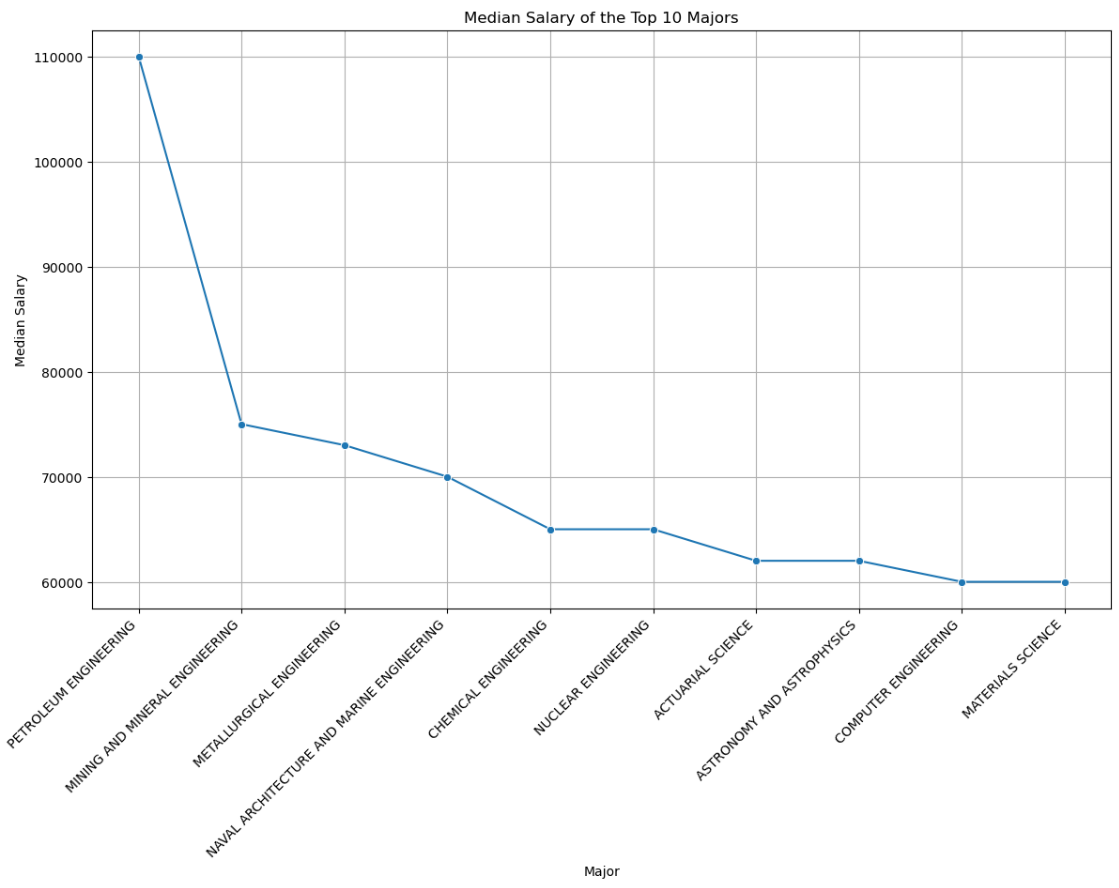
[231]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

# Load the dataset
recent_grads = pd.read_csv('recent-grads.csv')

# Display the first few rows of the dataset
print(recent_grads.head())
```

```
   index  Rank  Major_code                                   Major  \
0      0     1        2419                   PETROLEUM ENGINEERING
1      1     2        2416          MINING AND MINERAL ENGINEERING
2      2     3        2415               METALLURGICAL ENGINEERING
3      3     4        2417  NAVAL ARCHITECTURE AND MARINE ENGINEERING
4      4     5        2405                    CHEMICAL ENGINEERING

  Major_category  Total  Sample_size    Men  Women  ShareWomen  …  \
0    Engineering   2339           36   2057    282    0.120564  …
1    Engineering    756            7    679     77    0.101852  …
2    Engineering    856            3    725    131    0.153037  …
3    Engineering   1258           16   1123    135    0.107313  …
4    Engineering  32260          289  21239  11021    0.341631  …

   Part_time  Full_time_year_round  Unemployed  Unemployment_rate  Median  \
0        270                  1207          37           0.018381  110000
```

```
1        170                   388         85            0.117241    75000
2        133                   340         16            0.024096    73000
3        150                   692         40            0.050125    70000
4       5180                 16697       1672            0.061098    65000

   P25th   P75th  College_jobs  Non_college_jobs  Low_wage_jobs
0  95000  125000          1534               364            193
1  55000   90000           350               257             50
2  50000  105000           456               176              0
3  43000   80000           529               102              0
4  50000   75000         18314              4440            972

[5 rows x 22 columns]
```

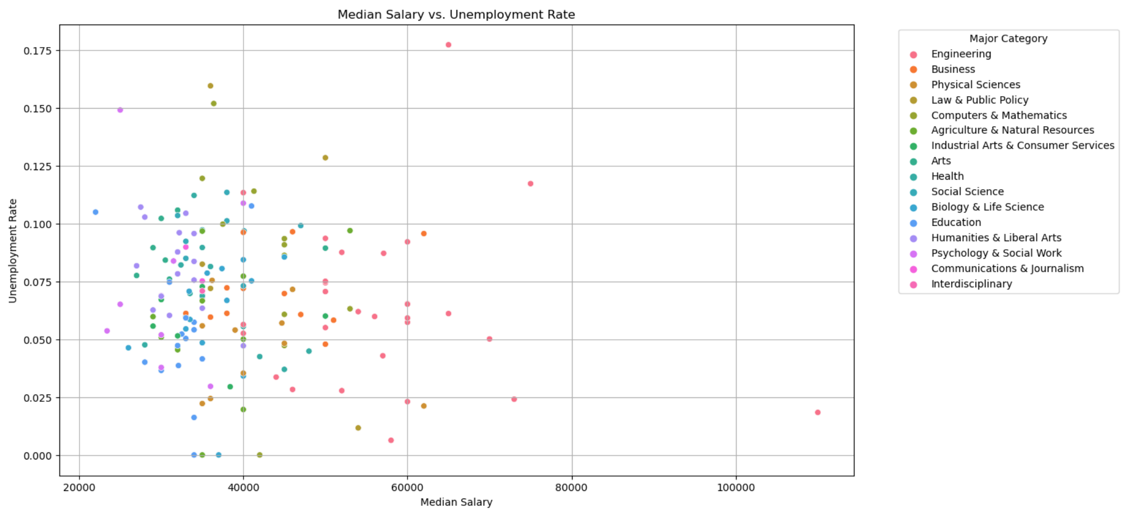
[232]:
```
# Sort the dataset by median salary
sorted_grads = recent_grads.sort_values(by='Median', ascending=False)

# Suppress specific warnings
warnings.filterwarnings('ignore', category=FutureWarning, module='seaborn')
```

[233]:
```
# 1. Line Plot: Median salary of the top 10 majors
plt.figure(figsize=(14, 8))
sns.lineplot(data=sorted_grads.head(10), x='Major', y='Median', marker='o')
plt.title('Median Salary of the Top 10 Majors')
plt.xlabel('Major')
plt.ylabel('Median Salary')
plt.xticks(rotation=45, ha='right')
plt.grid(True)
plt.show()
```



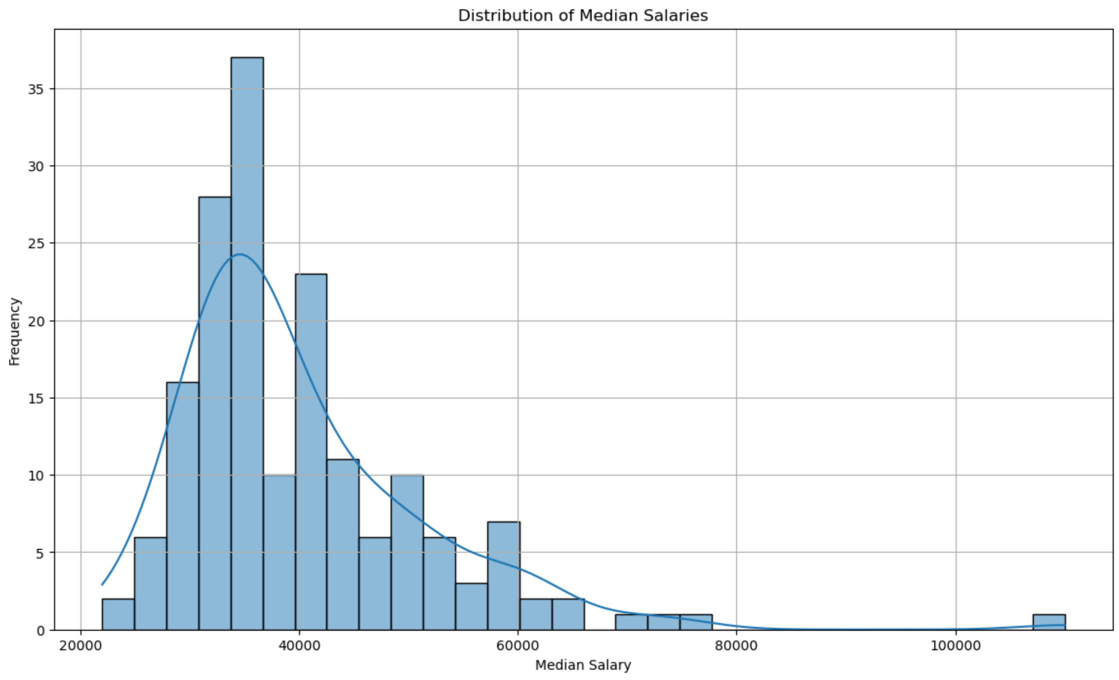
Line Plot: Shows the median salary for the top 10 majors by median salary. The x-axis labels are rotated for better alignment, and the figure size is increased for clarity.

[234]:
```python
# 2. Scatter Plot: Median salary vs Unemployment rate
plt.figure(figsize=(14, 8))
sns.scatterplot(data=recent_grads, x='Median', y='Unemployment_rate',␣
  ↪hue='Major_category')
plt.title('Median Salary vs. Unemployment Rate')
plt.xlabel('Median Salary')
plt.ylabel('Unemployment Rate')
plt.grid(True)
plt.legend(title='Major Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



Scatter Plot: Displays the relationship between the median salary and the unemployment rate for different majors, with different colors representing different major categories.

[235]:
```
# 3. Histogram: Distribution of Median Salaries
plt.figure(figsize=(14, 8))
sns.histplot(recent_grads['Median'], bins=30, kde=True)
plt.title('Distribution of Median Salaries')
plt.xlabel('Median Salary')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Histogram: Shows the distribution of median salaries across all majors, with a KDE (Kernel Density Estimate) overlay to visualize the distribution's shape.

```
[ ]:
```