03-homework

February 25, 2025

1 STA 141B WQ 25 Homework Assignment 2

1.1 Instructions

- Complete the exercises below. Create more code chunks if necessary. Answer all questions. Show results for both the *test* and *run* cases.
- Export the Jupyter Notebook as an PDF file.
- Submit the PDF by **Sunday, February 23, at 11:59 PM PT** to Gradescope.
- For each exercise, indicate the region of your answer in the PDF to facilitate grading.

1.2 Additional information

- Complete this worksheet yourself.
- You may use the internet or discuss possible approaches to solve the problems with other students. You are not allowed to share your code or your answers with other students.
- No other libraries than those explicitly allowed can be used.
- Use code cells for your Python scripts and Markdown cells for explanatory text or answers to non-coding questions. Answer all textual questions in complete sentences.
- Late homework submissions will not be accepted. No submissions will be accepted by email.
- The total number of points for this assignment is 20. You can earn 5 bonus points.

Exercise 1

As a public organization, the compensations of employees of all institutions of the University of California are freely accessible. These reports cover UC's career faculty and staff employees, as well as part-time, temporary and student employees. It is accessible here. Internally, the data requested by the search mask is queried using an undocumented API. For this exercise, you may use:

```
import requests
import pandas

from json import loads
```

Hint: If you encounter an error when parsing the data, try to use string methods (e.g., `str.replace`*) to deal with them.*

(a) Get the compensation information of all UC Irvine professors in 2023. How many entries are being returned?

[184]:
```
import requests
```

```
url = "https://ucannualwage.ucop.edu/wage/search.do"
headers = {
  "accept": "application/json, text/javascript, */*; q=0.01",
  "accept-language": "en-US,en;q=0.9",
  "content-type": "application/x-www-form-urlencoded",
  "priority": "u=1, i",
  "sec-ch-ua": '"Not(A:Brand";v="99", "Google Chrome";v="133", "Chromium";
 ↪v="133"',
  "sec-ch-ua-mobile": "?0",
  "sec-ch-ua-platform": "\"macOS\"",
  "sec-fetch-dest": "empty",
  "sec-fetch-mode": "cors",
  "sec-fetch-site": "same-origin",
  "x-requested-with": "XMLHttpRequest"
}
data = {
  "_search": "false",
  "nd": "1740541745802",
  "rows": "32212",
  "page": "1",
  "sidx": "EAW_LST_NAM",
  "sord": "asc",
  "year": "2023",
  "location": "Irvine",
  "firstname": "",
  "lastname": "",
  "title": "prof",
  "startSal": "",
  "endSal": ""
}

response = requests.post(url, headers=headers, data=data)
response
```

```
[184]: <Response [200]>
```

```
[191]: json_data = response.json()
```

```
[198]: import pandas as pd

# Extracting 'cell' data from each row
df = pd.DataFrame([row["cell"] for row in json_data["rows"]])

# Dropping column 0
df.drop(columns=[0], inplace=True)

# Renaming columns
```

```
df.columns = ["year", "location", "first name", "last name", "title", "gross␣
 ↪pay", "regular pay", "overtime pay",
              "other pay"]

# Removing records where 'title' contains 'profl'
df = df[~df["title"].str.contains("profl", case=False, na=False)]

df.head()
```

[198]:
```
   year location first name last name                    title  gross pay  \
0  2023   Irvine      *****     *****             ASST PROF-AY  137826.00
1  2023   Irvine      *****     *****  HS ASST CLIN PROF-HCOMP  107270.00
2  2023   Irvine      *****     *****            VIS ASST PROF   25782.00
3  2023   Irvine      *****     *****             ASST PROF-AY  106447.00
4  2023   Irvine      *****     *****         ASST ADJ PROF-AY   31228.00

  regular pay overtime pay other pay
0   108850.00         0.00  28976.00
1   105433.00         0.00   1837.00
2    25782.00         0.00      0.00
3   105633.00         0.00    813.00
4    31228.00         0.00      0.00
```

[201]:
```
f'The number of entries is {df.shape[0]}'
```

[201]:
```
'The number of entries is 2242'
```

BONUS

(b, i) Use the UCI directory to learn each professors department, if available. How many professors with departments information do you find? **(ii)** Find the four departments that have the largest average gross pay, and the four departments that have the largest average base pay.

[203]:
```
df = df[~df["first name"].str.fullmatch(r'\*+', na=False)]
df = df[~df["last name"].str.fullmatch(r'\*+', na=False)]
df = df[["first name", "last name", "gross pay"]]
df.head()
```

[203]:
```
    first name  last name  gross pay
48         KEV  ABAZAJIAN  191699.00
49      ACKBAR      ABBAS  131615.00
50       PABLO     ABBONA  578970.00
51    GEOFFREY     ABBOTT  617177.00
52  HERMELINDA     ABCEDE       0.00
```

[252]:
```
import requests
from bs4 import BeautifulSoup
```

```
def get_department(firstName, lastName):
  # Define the URL
  url = "https://directory.uci.edu/render-list"

  # Define headers (if required)
  headers = {
    "accept": "*/*",
    "content-type": "application/x-www-form-urlencoded; charset=UTF-8"
  }

  # Define data to send in POST request
  data = {
    "uciKey": f"{firstName} {lastName}",
    "filter": "all"
  }

  # Make the POST request
  response = requests.post(url, headers=headers, data=data)

  # Parse the response HTML using BeautifulSoup
  soup = BeautifulSoup(response.json()['html'], "html.parser")

  # Find the table with class 'directory-info-table-section'
  table = soup.find("table", class_="directory-info-table-section")
  if not table:
    print('Department table not found')
    return None

  # Find the row where the first td has class 'directory-info-table-row' and␣
 ↪text 'Department'
  for row in table.find_all("tr"):
    first_td = row.find("td")
    if first_td and first_td.text.strip() == "Department":
      second_td = first_td.find_next_sibling("td")
      if second_td:
        return second_td.text.strip()
      else:
        print('Department field not found')
        return None
```

[254]:
```
df["department"] = df.apply(lambda row: get_department(row["first name"],␣
 ↪row["last name"]), axis=1)
df.head()
```

```
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
Department table not found
```

```
[254]:     first name  last name  gross pay                         department
       48         KEV  ABAZAJIAN  191699.00            Physics and Astronomy
       49      ACKBAR      ABBAS  131615.00           Comparative Literature
       50       PABLO     ABBONA  578970.00      SOM - Radiological Sciences
       51    GEOFFREY     ABBOTT  617177.00  SOM - Physiology & Biophysics
       52  HERMELINDA     ABCEDE       0.00                             None
```

```
[255]: # Save to CSV
       df.to_csv("HW3E1b.csv")
```

```
[264]: df = pd.read_csv("HW3E1b.csv")
```

```
[267]: df = df.dropna(subset=["department"])

       df_grouped = df.groupby("department", as_index=False)["gross pay"].mean()
       df_grouped.rename(columns={"gross pay": "average gross pay"}, inplace=True)
       df_grouped = df_grouped.sort_values(by="average gross pay", ascending=False)

       print("Here are the top four departments with the highest average gross pay")
       df_grouped.head(4)
```

```
Here are the top four departments with the highest average gross pay
```

```
[267]:                                        department  average gross pay
       46                            Clinic-Ophthalmology          1221374.0
       154  Physical Therapy - UCI Health Fountain Valley           803681.0
       9                   Ambulatory Care Administration           758856.0
       72                   Emergency Room - Placentia Linda           753857.0
```

Exercise 2

Lets play a variation of the wiki game to learn about this phenomenon. The rules are as follows: - Start using either a provided article or the random article link (wiki menu on the left hand side) - Click on the first non-italicized link outside of parentheses and info-boxes - Ignore external links (e.g., `/wiki/File:...` or `/wiki/Category:...`) - Stop when reaching "Philosophy", a dead end (page with no links, this should return `None`) or when a loop occurs

Use the test cases below to check your code:

```
>play('/wiki/Brigade_Commander_(video_game)')
['/wiki/Brigade_Commander_(video_game)',
 '/wiki/Amiga_Action',
 '/wiki/Amiga',
 '/wiki/Personal_computer',
 '/wiki/Computer',
 '/wiki/Machine',
 '/wiki/Power_(physics)',
 '/wiki/Energy',
 '/wiki/Physical_quantity',
 '/wiki/Quantification_(science)',
 '/wiki/Mathematics',
 '/wiki/Mathematical_theory',
 '/wiki/Reason',
 '/wiki/Consciousness',
 '/wiki/Awareness',
 '/wiki/Philosophy']
```

```
>play('/wiki/Keretapi_Tanah_Melayu')
['/wiki/Keretapi_Tanah_Melayu',
 '/wiki/Airline',
 '/wiki/Civil_aviation',
 '/wiki/Aviation',
 '/wiki/Flight',
 '/wiki/Motion_(physics)',
 '/wiki/Physics',
 '/wiki/Scientific',
 '/wiki/Scientific_method',
 '/wiki/Empirical_evidence',
 '/wiki/Evidence',
 '/wiki/Proposition',
 '/wiki/Philosophy_of_language',
 '/wiki/Language',
 '/wiki/Communication',
 '/wiki/Information',
 '/wiki/Abstraction',
 '/wiki/Rule_of_inference',
 '/wiki/Philosophy_of_logic',
 '/wiki/Philosophy']

>play('/wiki/Robert_Alfred_Tarlton')
['/wiki/Robert_Alfred_Tarlton',
 '/wiki/Birmingham',
 '/wiki/City_status_in_the_United_Kingdom',
 '/wiki/The_Crown',
 '/wiki/State_(polity)',
 '/wiki/Politics',
 '/wiki/Decision-making',
 '/wiki/Psychology',
 '/wiki/Mind',
 '/wiki/Thought',
 '/wiki/Cognition',
 '/wiki/Action_(philosophy)',
 '/wiki/Philosophy']

>play('/wiki/Ricky_Vallen')
['/wiki/Ricky_Vallen', None]
```

(a) Run:

```
play('/wiki/Yadav') # (i)
play('/wiki/1953_Arab_Games') # (ii)
play('/wiki/Save_Me_(Silver_Convention_song)') # (iii)
```

[271]:
```
import requests
from bs4 import BeautifulSoup, Tag, NavigableString
```

```python
import time
```

[272]:
```python
BASE_URL = 'https://en.wikipedia.org'


def extract_link(e: Tag) -> str | None:
  parenthesis_count = 0

  for child in e.children:
    if isinstance(child, NavigableString):
      for char in child:
        if char == '(':
          parenthesis_count += 1
        elif char == ')':
          parenthesis_count -= 1
    elif child.name == "a" and child.has_attr("href"):
      href = child["href"]

      # Valid link linking to another article
      if not href.startswith("/wiki/"):
        continue
      if ":" in href:
        continue

      # Not in parentheses
      if parenthesis_count > 0:
        continue

      return href
    elif not child.name == "i" and parenthesis_count == 0:
      if link := extract_link(child):
        return link

  return None  # Return None if no valid link is found


def get_link(curr_page: str) -> str | None:
  response = requests.get(f'{BASE_URL}/{curr_page}')
  soup = BeautifulSoup(response.text, "html.parser")

  # Only from body
  body_content = soup.find(class_="mw-content-ltr")

  def parse_body(el: Tag):
    if str(el.name) == "p":
      link = extract_link(el)
      return link
    elif str(el.name) in ('ol', 'ul'):
      for li in el.find_all(["li"]):
        link = extract_link(li)
        return link

  # For all sections
  for section in body_content.children:
    if section.name == 'meta':
      for e in section.children:
        if res := parse_body(e):
          return res
    else:
      if res := parse_body(section):
        return res

  return None
```

[274]:
```
MAX_REQUESTS_PER_SECOND = 160



def play(start, target="/wiki/Philosophy"):
  """Navigates from start Wikipedia page to target page using article links."""
  current = start
  visited = set()
  path = [start]

  while current != target:
    if current in visited:
      # A loop occurred
      return path

    visited.add(current)
    next_link = get_link(current)
    path.append(next_link)

    if not next_link:
      # Stuck
      return path

    current = next_link
    time.sleep(1 / MAX_REQUESTS_PER_SECOND)

  return path
```

[180]:
```
# Test cases
play('/wiki/Brigade_Commander_(video_game)')
```

```
[180]: ['/wiki/Brigade_Commander_(video_game)',
        '/wiki/Amiga_Action',
        '/wiki/Amiga',
        '/wiki/Personal_computer',
        '/wiki/Computer',
        '/wiki/Machine',
        '/wiki/Power_(physics)',
        '/wiki/Energy',
        '/wiki/Physical_quantity',
        '/wiki/Quantification_(science)',
        '/wiki/Mathematics',
        '/wiki/Mathematical_theory',
        '/wiki/Logical_reasoning',
        '/wiki/Mind',
        '/wiki/Thought',
        '/wiki/Cognition',
        '/wiki/Action_(philosophy)',
        '/wiki/Philosophy']
```

```
[181]: play('/wiki/Keretapi_Tanah_Melayu')
```

```
[181]: ['/wiki/Keretapi_Tanah_Melayu',
        '/wiki/Airline',
        '/wiki/Civil_aviation',
        '/wiki/Aviation',
        '/wiki/Flight',
        '/wiki/Motion_(physics)',
        '/wiki/Physics',
        '/wiki/Scientific',
        '/wiki/Scientific_method',
        '/wiki/Empirical_evidence',
        '/wiki/Evidence',
        '/wiki/Proposition',
        '/wiki/Philosophy_of_language',
        '/wiki/Philosophy']
```

```
[182]: play('/wiki/Ricky_Vallen')
```

```
[182]: ['/wiki/Ricky_Vallen', None]
```

```
[178]: play('/wiki/Yadav')  # (i)
```

```
[178]: ['/wiki/Yadav',
        '/wiki/Peasant',
        '/wiki/Pre-industrial_society',
        '/wiki/Industrial_Revolution',
        '/wiki/Second_Industrial_Revolution',
        '/wiki/Discovery_(observation)',
        '/wiki/Sciences',
        '/wiki/Scientific_method',
        '/wiki/Empirical_evidence',
        '/wiki/Evidence',
        '/wiki/Proposition',
        '/wiki/Philosophy_of_language',
        '/wiki/Philosophy']
```

```
[177]: play('/wiki/1953_Arab_Games')   # (ii)
```

```
[177]: ['/wiki/1953_Arab_Games',
 '/wiki/Arab_Games',
 '/wiki/Multi-sport_event',
 '/wiki/Sport',
 '/wiki/Physical_activity',
 '/wiki/Skeletal_muscle',
 '/wiki/Vertebrate',
 '/wiki/Animal',
 '/wiki/Multicellular',
 '/wiki/Organism',
 '/wiki/Life',
 '/wiki/Matter',
 '/wiki/Classical_physics',
 '/wiki/Physics',
 '/wiki/Scientific',
 '/wiki/Scientific_method',
 '/wiki/Empirical_evidence',
 '/wiki/Evidence',
 '/wiki/Proposition',
 '/wiki/Philosophy_of_language',
 '/wiki/Philosophy']
```

```
[179]: play('/wiki/Save_Me_(Silver_Convention_song)')   # (iii)
```

```
[179]: ['/wiki/Save_Me_(Silver_Convention_song)',
 '/wiki/Euro_disco',
 '/wiki/Electronic_dance_music',
 '/wiki/Electronic_music',
 '/wiki/Music_genre',
 '/wiki/Music',
 '/wiki/Sound',
 '/wiki/Physics',
 '/wiki/Scientific',
 '/wiki/Scientific_method',
 '/wiki/Empirical_evidence',
 '/wiki/Evidence',
 '/wiki/Proposition',
 '/wiki/Philosophy_of_language',
 '/wiki/Philosophy']
```

(b) Run this the game 200 times and report **(i)** How often did you end with *Philosophy*? **(ii)** What is the average and **(iii)** maximum length of your games? **(iv)** Print the ten most often visited articles and **(v)** the number of all visited articles.

[269]:
```python
import pandas as pd

wikipedia_articles = [
  "/wiki/Python_(programming_language)",
  "/wiki/Artificial_intelligence",
  "/wiki/Machine_learning",
  "/wiki/Deep_learning",
  "/wiki/Neural_network",
  "/wiki/Computer_vision",
  "/wiki/Natural_language_processing",
  "/wiki/Robotics",
  "/wiki/Data_science",
  "/wiki/Big_data",
  "/wiki/Cloud_computing",
  "/wiki/Cybersecurity",
  "/wiki/Quantum_computing",
  "/wiki/Internet_of_things",
  "/wiki/Blockchain",
  "/wiki/Cryptocurrency",
  "/wiki/Virtual_reality",
  "/wiki/Augmented_reality",
  "/wiki/Software_engineering",
  "/wiki/Operating_system",
  "/wiki/Computer_network",
  "/wiki/Database",
  "/wiki/Algorithm",
  "/wiki/Data_structure",
  "/wiki/Programming_language",
  "/wiki/Java_(programming_language)",
  "/wiki/JavaScript",
  "/wiki/C_(programming_language)",
  "/wiki/C%2B%2B",
  "/wiki/C_Sharp_(programming_language)",
  "/wiki/Swift_(programming_language)",
  "/wiki/Kotlin_(programming_language)",
  "/wiki/Rust_(programming_language)",
  "/wiki/Go_(programming_language)",
  "/wiki/PHP",
  "/wiki/R_(programming_language)",
  "/wiki/SQL",
  "/wiki/NoSQL",
  "/wiki/Software_testing",
  "/wiki/DevOps",
  "/wiki/Continuous_integration",
  "/wiki/Continuous_delivery",
  "/wiki/Microservices",
  "/wiki/Docker_(software)",
  "/wiki/Kubernetes",
  "/wiki/Cloud_native_computing",
  "/wiki/Serverless_computing",
  "/wiki/Edge_computing",
  "/wiki/5G",
  "/wiki/Wireless_network",
  "/wiki/Artificial_general_intelligence",
  "/wiki/Superintelligence",
  "/wiki/Computer_graphics",
  "/wiki/Game_development",
  "/wiki/Unreal_Engine",
  "/wiki/Unity_(game_engine)",
  "/wiki/OpenGL",
  "/wiki/Vulkan_(API)",
  "/wiki/Web_development",
  "/wiki/Frontend_and_backend",
  "/wiki/React_(JavaScript_library)",
  "/wiki/Angular_(web_framework)",
  "/wiki/Vue.js",
  "/wiki/Node.js",
  "/wiki/Express.js",
  "/wiki/Django_(web_framework)",
  "/wiki/Flask_(web_framework)",
  "/wiki/Ruby_on_Rails",
  "/wiki/ASP.NET",
  "/wiki/GraphQL",
  "/wiki/RESTful_API",
  "/wiki/Web_scraping",
  "/wiki/Selenium_(software)",
  "/wiki/Beautiful_Soup_(HTML_parser)",
  "/wiki/Scrapy",
  "/wiki/Data_mining",
  "/wiki/BigQuery",
  "/wiki/Spark_(software)",
  "/wiki/Hadoop",
  "/wiki/Elasticsearch",
  "/wiki/Kibana",
  "/wiki/Logstash",
  "/wiki/Git",
  "/wiki/GitHub",
  "/wiki/GitLab",
  "/wiki/Bitbucket",
  "/wiki/Cybersecurity",
  "/wiki/Ethical_hacking",
  "/wiki/Penetration_testing",
  "/wiki/Social_engineering_(security)",
  "/wiki/Encryption",
  "/wiki/Public-key_cryptography",
  "/wiki/Symmetric-key_algorithm",
  "/wiki/Hash_function",
  "/wiki/Artificial_neural_network",
  "/wiki/Recurrent_neural_network",
  "/wiki/Convolutional_neural_network",
  "/wiki/Generative_adversarial_network",
  "/wiki/Transformer_(machine_learning_model)",
  "/wiki/BERT_(language_model)",
  "/wiki/GPT-3",
  "/wiki/OpenAI",
  "/wiki/TensorFlow",
  "/wiki/PyTorch",
  "/wiki/Keras",
  "/wiki/Scikit-learn",
  "/wiki/Pandas_(software)",
  "/wiki/NumPy",
  "/wiki/Matplotlib",
  "/wiki/Seaborn_(software)",
  "/wiki/Plotly",
  "/wiki/LLM_(language_model)",
  "/wiki/AutoML",
  "/wiki/Federated_learning",
  "/wiki/Edge_AI",
  "/wiki/Reinforcement_learning",
  "/wiki/Q-learning",
  "/wiki/Markov_decision_process",
  "/wiki/Monte_Carlo_method",
  "/wiki/A_star_search_algorithm",
  "/wiki/Minimax",
  "/wiki/AlphaGo",
  "/wiki/AlphaZero",
  "/wiki/Chess_engine",
  "/wiki/Stockfish_(chess)",
  "/wiki/Leela_Chess_Zero",
  "/wiki/Computer_science",
  "/wiki/Discrete_mathematics",
  "/wiki/Graph_theory",
  "/wiki/Boolean_algebra",
  "/wiki/Complexity_theory",
  "/wiki/Turing_machine",
  "/wiki/Computational_complexity_theory",
  "/wiki/P_versus_NP_problem",
  "/wiki/Halting_problem",
  "/wiki/Quantum_algorithm",
  "/wiki/Shor%27s_algorithm",
  "/wiki/Grover%27s_algorithm",
  "/wiki/Quantum_error_correction",
  "/wiki/Quantum_supremacy",
  "/wiki/Quantum_entanglement",
  "/wiki/Quantum_cryptography",
  "/wiki/Post-quantum_cryptography",
  "/wiki/Information_theory",
  "/wiki/Shannon_entropy",
  "/wiki/Kolmogorov_complexity",
  "/wiki/Chaos_theory",
  "/wiki/Fractal",
  "/wiki/Mandelbrot_set",
  "/wiki/Lorenz_system",
  "/wiki/Cellular_automaton",
  "/wiki/Game_of_Life",
  "/wiki/Evolutionary_algorithm",
  "/wiki/Genetic_algorithm",
  "/wiki/Swarm_intelligence",
  "/wiki/Particle_swarm_optimization",
  "/wiki/Ant_colony_optimization",
  "/wiki/Neural_ODE",
  "/wiki/Meta-learning",
  "/wiki/Zero-shot_learning",
  "/wiki/Few-shot_learning",
  "/wiki/Civilization_VI",
  "/wiki/Assembly_language",
  "/wiki/Compiler",
  "/wiki/Interpreter_(computing)",
  "/wiki/Integrated_development_environment",
  "/wiki/Version_control",
  "/wiki/Continuous_deployment",
  "/wiki/Agile_software_development",
  "/wiki/Scrum_(software_development)",
  "/wiki/Kanban_(development)",
  "/wiki/Extreme_programming",
  "/wiki/Software_architecture",
  "/wiki/Design_pattern_(computer_science)",
  "/wiki/Model%E2%80%93view%E2%80%93controller",
  "/wiki/Service-oriented_architecture",
  "/wiki/Representational_state_transfer",
  "/wiki/Remote_procedure_call",
  "/wiki/Aspect-oriented_programming",
  "/wiki/Event-driven_programming",
  "/wiki/Functional_programming",
  "/wiki/Logic_programming",
  "/wiki/Procedural_programming",
  "/wiki/Object-oriented_programming",
  "/wiki/Concurrent_computing",
  "/wiki/Parallel_computing",
  "/wiki/Distributed_computing",
  "/wiki/Grid_computing",
  "/wiki/Cluster_computing",
  "/wiki/Supercomputer",
  "/wiki/Green_computing",
  "/wiki/High-performance_computing",
  "/wiki/Embedded_system",
  "/wiki/Real-time_computing",
  "/wiki/Ubiquitous_computing",
  "/wiki/Pervasive_computing",
  "/wiki/Human%E2%80%93computer_interaction",
  "/wiki/Usability",
  "/wiki/User_experience_design",
  "/wiki/Information_retrieval",
]

df = pd.DataFrame(wikipedia_articles, columns=["Wikipedia_Article"])
df.head()
```

[269]:
```
                       Wikipedia_Article
0  /wiki/Python_(programming_language)
1        /wiki/Artificial_intelligence
2               /wiki/Machine_learning
3                  /wiki/Deep_learning
4                 /wiki/Neural_network
```

[275]:
```
df["path"] = df["Wikipedia_Article"].apply(play)
```

[276]:
```
df.head()
```

[276]:
```
                       Wikipedia_Article  \
0  /wiki/Python_(programming_language)
1        /wiki/Artificial_intelligence
2               /wiki/Machine_learning
3                  /wiki/Deep_learning
4                 /wiki/Neural_network

                                                path
0  [/wiki/Python_(programming_language), /wiki/Hi…
```

```
1  [/wiki/Artificial_intelligence, /wiki/Intellig…
2  [/wiki/Machine_learning, /wiki/Field_of_study,…
3  [/wiki/Deep_learning, /wiki/Machine_learning, …
4  [/wiki/Neural_network, /wiki/Neurons, /wiki/Me…
```

[277]:
```
df.to_csv("HW3E2b.csv")
```

[279]:
```
from collections import Counter

# (i) Count occurrences of 'Philosophy' as the last article
philosophy_count = sum(path[-1] == '/wiki/Philosophy' for path in df['path'])

# (ii) Calculate average game length
avg_length = sum(len(path) for path in df['path']) / len(df)

# (iii) Find maximum game length
max_length = max(len(path) for path in df['path'])

# Flatten the list of paths to get individual article counts
all_articles = [article for path in df['path'] for article in path]
article_counts = Counter(all_articles)

# (iv) Get the ten most often visited articles
most_visited = article_counts.most_common(10)

# (v) Get the number of unique visited articles
unique_articles = len(article_counts)

# Print results
print(f"(i) Games ending in 'Philosophy': {philosophy_count}")
print(f"(ii) Average game length: {avg_length:.2f}")
print(f"(iii) Maximum game length: {max_length}")
print("(iv) Ten most visited articles:")
for article, count in most_visited:
  print(f"    {article}: {count} times")
print(f"(v) Number of unique visited articles: {unique_articles}")
```

```
(i) Games ending in 'Philosophy': 193
(ii) Average game length: 12.35
(iii) Maximum game length: 25
(iv) Ten most visited articles:
    /wiki/Philosophy: 193 times
    /wiki/Action_(philosophy): 149 times
    /wiki/Cognition: 135 times
    /wiki/Mind: 134 times
    /wiki/Thought: 134 times
    /wiki/Logical_reasoning: 131 times
    /wiki/Mathematics: 127 times
    /wiki/Mathematical_theory: 127 times
    /wiki/Computer_program: 42 times
    /wiki/Sequence: 42 times
(v) Number of unique visited articles: 491
```

(c) Print the articles that you obtain when starting from *Philosophy*.

```
[172]: play('/wiki/Philosophy')
```

```
[172]: ['/wiki/Philosophy']
```