

# cos482-hw1-pease

October 7, 2025

In this code, I went with an object oriented design in order to compartmentalize the major tasks. I wrote an Article class to handle storing the data for each parsed article, including methods for exporting this data to both dataframe formats and to parse the publicationInfo data. Additionally, I wrote a class to handle scraping all of the webpages and setting up the driver. This will result in a list of all the raw HTML files. Finally, I wrote a page parsing class, that takes that list of HTML pages and parses out the specific articles. This creates a master list of articles. Further on, this list of articles is converted to pandas dataframes using the methods defined in Article. These dataframes are used to generate the graphs and answers to the questions below.

```
[10]: from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
import matplotlib.pyplot as plt
import pandas as pd
import re
import time
```

```
[2]: class Article:
    containerSelector = lambda soup: soup.find_all("div", class_="gs_r gs_or_
↪gs_scl")
    titleSelector = lambda article: article.find("h3", class_="gs_rt").text
    authorSelector = lambda article: article.find("div", class_="gs_a").text
    citedBySelector = lambda article: article.find("div", class_="gs_fl_
↪gs_flb").find_all("a")[2].text

    def __init__(self, title, publicationInfo, cited_by):
        self.title = title
        self.publicationInfo = publicationInfo
        self.cited_by = int(cited_by.split(" ")[-1])

        # Process Additional Information from publicationInfo
        self.authors = self.publicationInfo.split("-")[0].strip()
        self.publisher = self.publicationInfo.split("-")[-1].strip()
        self.venue = self.publicationInfo.split("-")[-2].strip().split(",")[0].
↪strip()
```

```

        self.year = int(re.search(r'\b(20\d{2})\b', self.publicationInfo).
↪group(1))

        # Calculate average citations per year
        divisor = 2026 - self.year if self.year is not None else 1
        self.averageCitations = self.cited_by / divisor

    def print(self):
        print(f"Title: {self.title}")
        print(f"Publication Info: {self.publicationInfo}")
        print(f"Cited by: {self.cited_by}")
        print(f"Authors: {self.authors}")
        print(f"Publisher: {self.publisher}")
        print(f"Venue: {self.venue}")
        print(f"Year: {self.year}")
        print(f"Average Citations per Year: {self.averageCitations}")
        print()

    def to_DF1(self):
        return pd.DataFrame({
            'title': [self.title],
            'publication_info': [self.publicationInfo],
            'cited_by': [f"Cited by {self.cited_by}"],
        })

    def to_DF2(self):
        return pd.DataFrame({
            'title': [self.title],
            'authors': [self.authors],
            'year': [self.year],
            'venue': [self.venue],
            'publisher': [self.publisher],
            'citation_count': [self.cited_by],
            'avg_citations_per_year': [self.averageCitations]
        })

```

```

[3]: class WebScraper:
    def __init__(self, url):
        self.url = url
        self.pages = []

        # Constants
        self.scroll_pause_time = 3

        # Setup Selenium Webdriver
        self.service = Service(executable_path="./chromedriver.exe")
        self.options = webdriver.ChromeOptions()

```

```

self.options.add_argument('--ignore-certificate-errors')
self.options.add_argument('--incognito')
#self.options.add_argument('--headless')

self.driver = webdriver.Chrome(service=self.service, options=self.
↪options)
self.driver.get(self.url)
WebDriverWait(self.driver, 5).until(lambda s: s.find_element(By.ID, "
↪gs_hdr_lgo").is_displayed())
self.screen_height = self.driver.execute_script("return window.screen.
↪height;")

def close(self):
    self.driver.close()

def get_page(self):
    i = 0
    scroll_height = self.driver.execute_script("return document.body.
↪scrollHeight;")
    while (self.screen_height*i < scroll_height) and (i <= 20):
        self.driver.execute_script(f"window.scrollTo(0, {self.
↪screen_height}*{i});")
        i += 1
        time.sleep(self.scroll_pause_time)
        scroll_height = self.driver.execute_script("return document.body.
↪scrollHeight;")
        self.pages.append(self.driver.page_source)

def get_all_pages(self):
    self.get_page()
    while True:
        try:
            next_button = WebDriverWait(self.driver, 5).until(
                lambda s: s.find_element(By.LINK_TEXT, "Next"))
            next_button.click()
            try:
                WebDriverWait(self.driver, 5).until(
                    lambda s: s.find_element(By.ID, "gs_bdy_sb_in").
↪is_displayed())
            except TimeoutException:
                print("Captcha detected. Please solve it and press Enter to
↪continue...")
                input()
                self.get_page()
            except TimeoutException:
                print("No more pages")

```

```

        break
    self.close()

```

```

[4]: class PageParser:
    def __init__(self, pages):
        self.pages = pages
        self.articles = []

    def parse(self, page):
        soup = BeautifulSoup(page, "lxml")
        page_articles = Article.containerSelector(soup)
        for article in page_articles:
            title = Article.titleSelector(article)
            authors = Article.authorSelector(article)
            cited_by = Article.citedBySelector(article)
            self.articles.append(Article(title, authors, cited_by))

    def parse_all(self):
        for page in self.pages:
            self.parse(page)

```

```

[ ]: # Start Here - Execution
url = "https://scholar.google.com/scholar?
      ↪start=0&q=machine+learning&hl=en&as_sdt=0,20&as_ylo=2023"

# Task 1
# Scrape the pages
scraper = WebScraper(url)
scraper.get_all_pages()
pages = scraper.pages
scraper.close()

# Sideloader - Load previously scraped pages
# with open('pages_100.txt', 'r', encoding='utf-8') as file:
#     pages = file.read().
      ↪split('=====')

# Parse the pages
parser = PageParser(pages)
parser.parse_all()
articles = parser.articles

# Generate first DataFrame and save to CSV (Task 1)
df1 = pd.concat([article.to_DF1() for article in articles], ignore_index=True)
df1.to_csv('ml_articles_info.csv', index=False)

# Generate second DataFrame and save to CSV (Task 2, A)

```

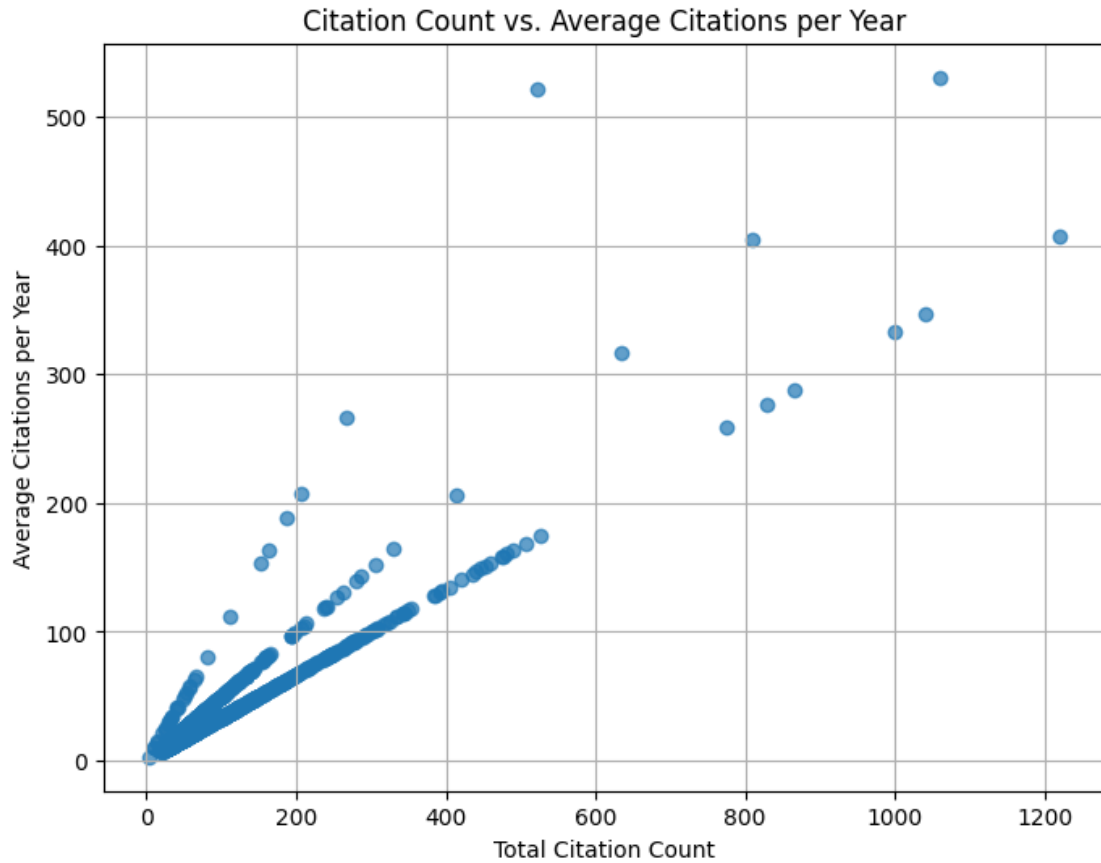
```
df2 = pd.concat([article.to_DF2() for article in articles], ignore_index=True)
df2.to_csv('ml_articles_info-cleaned.csv', index=False)
```

```
[15]: # Task 2, Question (b)
filtered_df = df2[
    (df2['year'] >= 2024) &
    (df2['citation_count'] > 100)
]

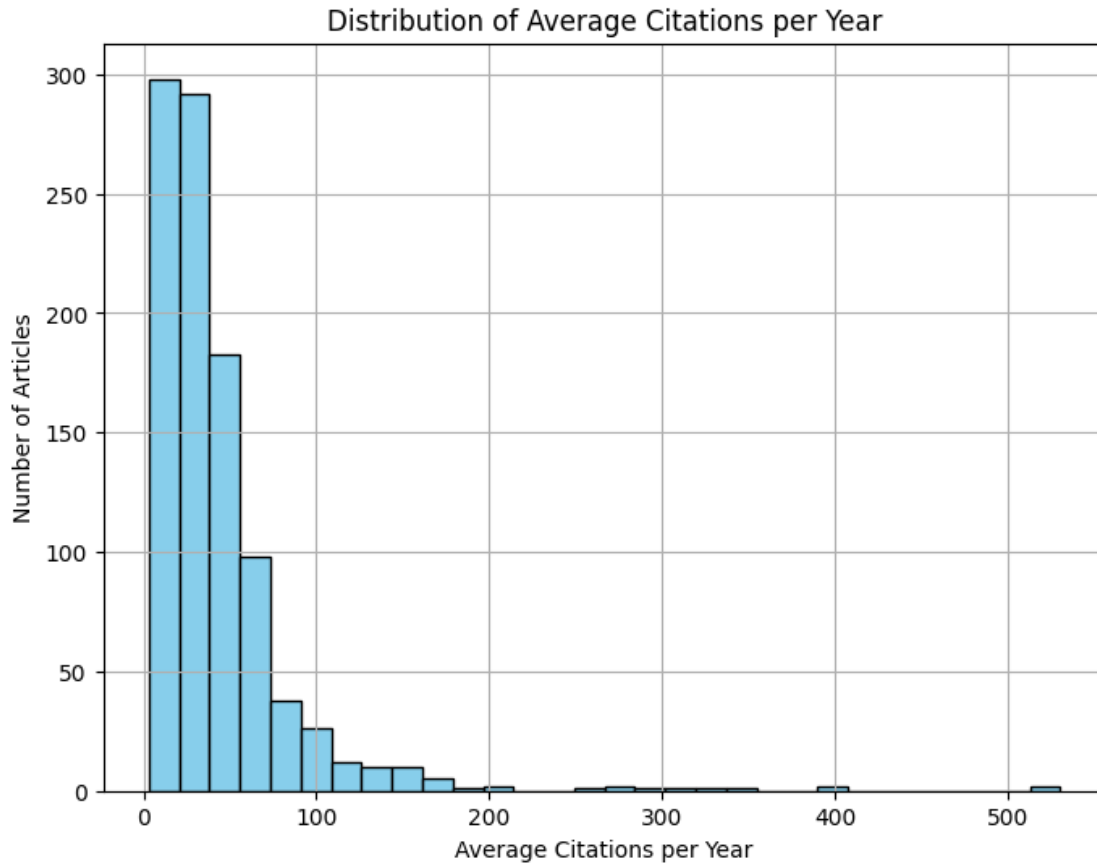
print("Articles from 2024 or later with over 100 citations:")
print(len(filtered_df))
```

Articles from 2024 or later with over 100 citations:  
88

```
[11]: # Task 2, Question (c)
plt.figure(figsize=(8, 6))
plt.scatter(df2['citation_count'], df2['avg_citations_per_year'], alpha=0.7)
plt.xlabel('Total Citation Count')
plt.ylabel('Average Citations per Year')
plt.title('Citation Count vs. Average Citations per Year')
plt.grid(True)
plt.show()
```



```
[12]: # Task 2, Question (d)
plt.figure(figsize=(8, 6))
plt.hist(df2['avg_citations_per_year'], bins=30, color='skyblue', edgecolor='black')
plt.xlabel('Average Citations per Year')
plt.ylabel('Number of Articles')
plt.title('Distribution of Average Citations per Year')
plt.grid(True)
plt.show()
```



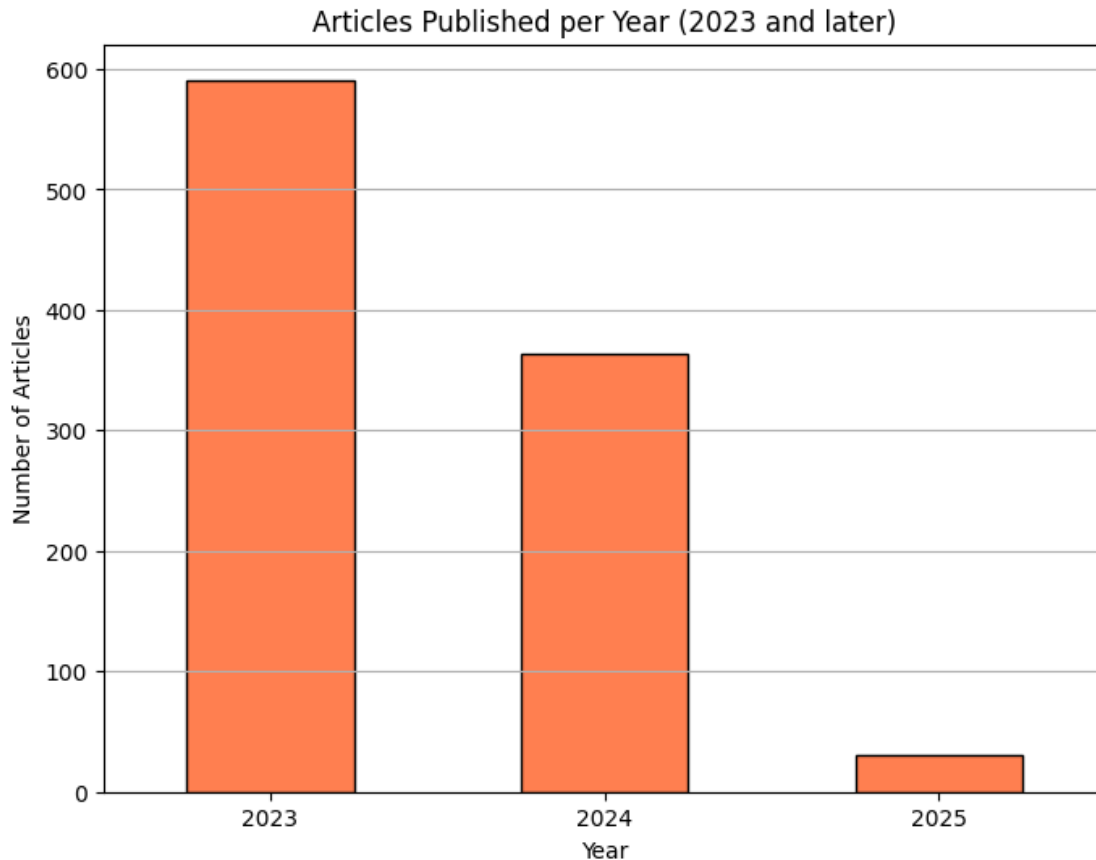
```
[13]: # Task 2, Question (e)
# Count number of articles published in each year since 2023
year_counts = df2[df2['year'] >= 2023]['year'].value_counts().sort_index()
print("Number of articles published in each year since 2023:")
print(year_counts)

# Bar plot
plt.figure(figsize=(8, 6))
year_counts.plot(kind='bar', color='coral', edgecolor='black')
plt.xlabel('Year')
plt.ylabel('Number of Articles')
plt.title('Articles Published per Year (2023 and later)')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

Number of articles published in each year since 2023:

|      |     |
|------|-----|
| year |     |
| 2023 | 591 |
| 2024 | 364 |

2025      31  
Name: count, dtype: int64



```
[14]: # Task 2, Question (f)
mean_citations_per_year = df2.groupby('year')['citation_count'].mean()
print("Mean citation count for each publication year:")
print(mean_citations_per_year)

# Bar plot
plt.figure(figsize=(8, 6))
mean_citations_per_year.plot(kind='bar', color='mediumseagreen',
                               edgecolor='black')
plt.xlabel('Publication Year')
plt.ylabel('Mean Citation Count')
plt.title('Mean Citation Count per Publication Year')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

Mean citation count for each publication year:



```
year
2023    130.954315
2024     79.027473
2025     80.354839
Name: citation_count, dtype: float64
```

