Introduction to Web Science

Assignment 5

Jun Sun
junsun@uni-koblenz.de

Iryna Dubrovska
idubrovska@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science University of
Koblenz-Landau

Submission until:	December 7, 2021, 24:00 CET
Tutorial on:	December 9, 2021, 16:00 CET

This assignment focuses on the concepts of 1) How big is the Web, 2) Descriptive
Web Models and 3) Programming in Python. Some of the tasks may require you to do additional research extending the lecture. Please keep the citation rules in mind.

For all the assignment questions that require you to write a code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Date: 07/12/2021

Team Name: Boehm

Abhinav Ralhan	(abhinavr8@uni-koblenz.de)
Fatima Akram	(fatimaakram9396@uni-koblenz.de)
Hammad Ahmed	(hammadahmed@uni-koblenz.de)
Vishal Vidhani	(vvidhani@uni-koblenz.de)

1 Python Programming. Web Crawling (32 points)

1.1 Your task is write a crawler to “crawl” the Simple English Wikipedia. The data you collect while crawling should be arranged in a dataframe. (27 points)

You can start crawling from https://simple.wikipedia.org/wiki/Climate_change (it is your initial web page) and you can use the following libraries: requests, json, BeautifulSoup and pandas. Beautifulsoup provides higher level of abstraction when processing URLs which makes coding simpler, but if you prefer urllib, you can go for it instead. The algorithm can be represented as following:

1. **Crawl the initial web page and count links. The links should be separated into threegroups: internal (local), external, and links that reference content within the same page (hint: they usually start with “” in HTML code). Internal links are the links within the same domain (simple.wikipedia.org).**
2. **Extract internal links from the (initial) web page. Our goal is to extract those linksthat lead to other pages in the Simple English Wikipedia.**
3. **Extract the timestamp regarding when the article was last modified. Assign None to those web pages that do not have this attribute.**
4. **Crawl through the extracted internal links and repeat steps 1-3. Make sure that you only crawl through internal links, otherwise you would start crawling the entire web. Hint: it might be useful for you to have some output on the crawler’s console depicting which URL is currently being processed or how many URLs have been processed so far.**
5. **Organise the collected data into a dataframe with the following columns: Pagecount,INTcount (number of internal links), EXTcount (number of external links), URLfragments (number of links referencing content on the same page), timestamp (e.g., Figure 1).**

You are expected to handle any errors such as URL and HTTP errors.

Attention: you do not have to crawl through the whole Simple English Wikipedia! Please, set a limiter to <=200. 200 is the maximum number of pages you should crawl.

Remark:
In a real life setting you should rather do crawling on Wikipedia dumps. Doing “live” crawling can put additional pressure on wiki servers. Although, it highly depends on how your crawler is configured and what is the magnitude of your crawling.

Solution

```
from urllib.request import urlopen

from bs4 import BeautifulSoup

from dateutil import parser

import pandas as pd

import random as rd
import re

internal_urls = set()
external_urls =set()
reference_urls = set()
df = pd.DataFrame(columns=['PageCount','INTcount', 'EXTcount', 'URLfragments'])
url = "https://simple.wikipedia.org/wiki/Climate_change"


def Set_New_Url(internal_links_Count, internal_urls):
    url = ""
    if internal_links_Count != 0:
        if internal_links_Count == 1:
            index = 0
        else:
            index = rd.randint(1, int(internal_links_Count/2))
            internal_url_list = list(internal_urls)
            if r'https://' in internal_url_list[index]:
                url = internal_url_list[index]
            else:
                url = f'https://simple.wikipedia.org{internal_url_list[index]}'
    else:
        print(f'No any Internal Links in URL: {url}, so continue it with the
initial URL')
        url = 'https://simple.wikipedia.org/wiki/Climate_change'

    return url


for pageCount in range(200):

    print(f'\nVisiting Url {pageCount+1}: {url}')
    try:
        html = urlopen(url)
        bs = BeautifulSoup(html, "lxml")

        #Find internal links count
        internal_links = bs.find("div",{"id" : "content"}).findAll("a" ,
href=re.compile("(/wiki/)+([A-Za-z0-9_:()])+"))
        internal_links_Count = 0
        for link in internal_links:
            if(type(link['href']) == str):
                splitted = link['href'].split('/')
            if splitted:
                if len(splitted) >= 3:
                    if  splitted[1]=="wiki":
                        internal_urls.add(link['href'])
                        internal_links_Count += 1
            if r"simple.wikipedia.org" in link['href']:
                internal_urls.add(link['href'])
                internal_links_Count += 1

        #Find reference links count
        reference_links = bs.find("div",{"id" : "content"}).findAll("a" ,
href=re.compile("(#[A-Za-z0-9_:()])+"))
        reference_links_Count = 0
        for link in reference_links:
            reference_urls.add(link['href'])
            reference_links_Count += 1

        #Find external links count
        external_links = bs.find("div",{"id" : "content"}).findAll("a",
href=re.compile("([A-Za-z0-9_:()])+"))
        external_links_Count = 0
        for link in external_links:
            if r'/w/' not in link['href'] and link['href'] not in internal_urls
and link['href'] not in reference_urls:
                external_urls.add(link['href'])
                external_links_Count += 1

        #Fetch Last Modified DateTime
        timestamp = "None"
        timestamp_text = bs.find(id="footer-info-lastmod")
        if timestamp_text:
            res = parser.parse(timestamp_text.text, fuzzy=True)
            timestamp = (str(res).replace(" ", "T"))
        #print(f'Visiting link :- {url}')
        print(f'Internal Link Count:- {internal_links_Count}')
        print(f'External Link Count:- {external_links_Count}')
        print(f'Reference Link Count:- {reference_links_Count}')
        print(f'Time Stamp:- {timestamp}')
        df = df.append({'PageCount': pageCount+1, 'INTcount' :
internal_links_Count, 'EXTcount' : external_links_Count, 'URLfragments' :
reference_links_Count, 'TimeStamp': timestamp}, ignore_index=True)

        url = Set_New_Url(internal_links_Count, internal_urls, )
    except:
        url = Set_New_Url(internal_links_Count, internal_urls, )
        pass

print(df)
df.to_csv("LinkInformation.csv")
```

1.2 Briefly explain potential limitations of your crawler (max 200 words). You may think of the situations when it is used on some other websites other than the Simple English Wikipedia. (5 points)

There are few limitations in our code if we crawl on other sites with this code.

1. **Internal Links**: we have used the regular expression to fetch the internal links, as internal links are start with "/wiki/", so this link will only be accessible with this cite only.

2. **Reference Link:** Likewise internal links the reference links are also start with the "#" in a html href property of an anchor tag. Therefore, it will only be accessible with this cite only.

3. **TimeStamp**: As time stamp is defined in the footer tag with the id "footer-info-lastmod", so it will not be accessible by any other cite and will populate "None", if tag not found.

Finally, If no internal links on the other cites then it will try with the initial https://simple.wikipedia.org/wiki/Climate_change until loop run 200 times.

2 Web Crawler Results (15 points)

This task is based upon the dataset you obtained in task 1. If you have not succeeded with task 1, you can simulate the dataset. Having the dataset, proceed with the following:

1. Find mean and median of INTcount, EXTcount and URLfragments. What is morerelevant in this case, mean or median? Explain in which cases median is more informative than mean. (5 points)

Solution:

The median of INTcount, EXTcount, and URLfragments is 57.0, 6.5 , 8.0 respectively. The mean of INTcount, EXTcount, and URLfragments is 112.03, 18.595, 16.61

Median is more relevant in this case than mean.

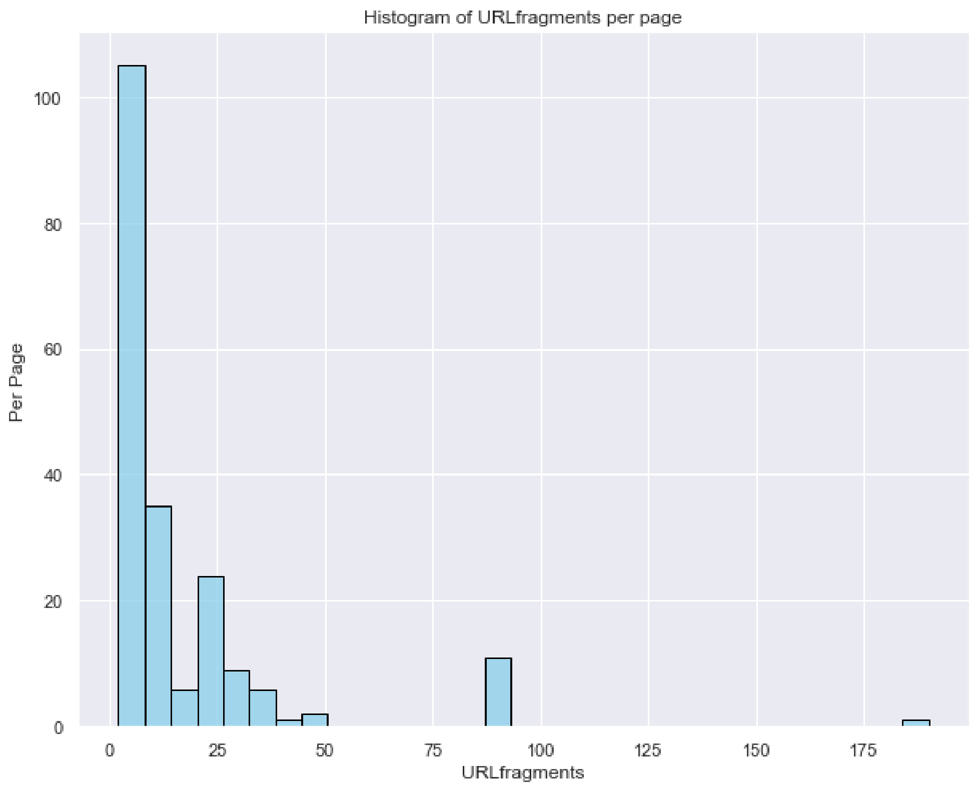
Median is more relevant than mean when our data is skewed, i.e., when the frequency distribution of our data is asymmetrical. If the data is normal, in that case our mean, median and mode, all three are identical. But when data gets skewed then mean loses the ability to represent the central data and mean becomes more relevant.

2. Plot a histogram showing the distribution of INTcount, EXTcount and URLfragments per web page. Please, include the screenshots. (5 points).

Solution

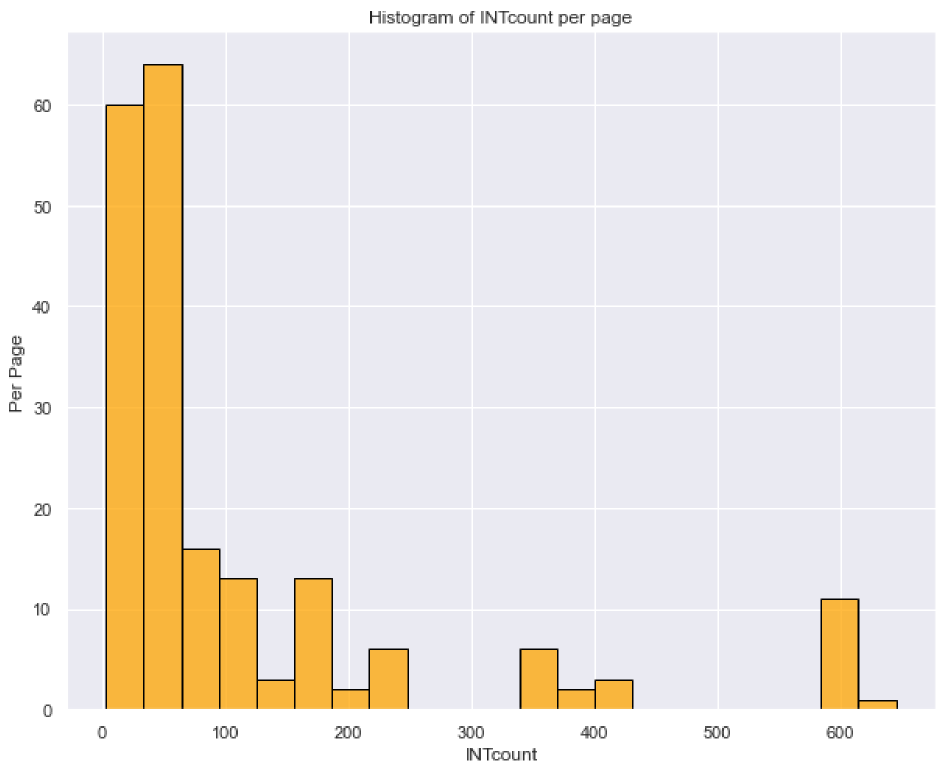
Histogram of URLfragments per page

```
import pandas as pd
import numpy as np

 import matplotlib.pyplot as plt

 import seaborn as sns

 from matplotlib.pyplot import
figure

df=pd.read_csv('LinkInformation.csv
')

figure(figsize=(10, 8))

sns.histplot(df.URLfragments, color = "skyblue",
ec="black")

plt.ylabel('Per Page')

plt.xlabel('URLfragments')

plt.title('Histogram of URLfragments per page')
```



Histogram of INTcount perpage:

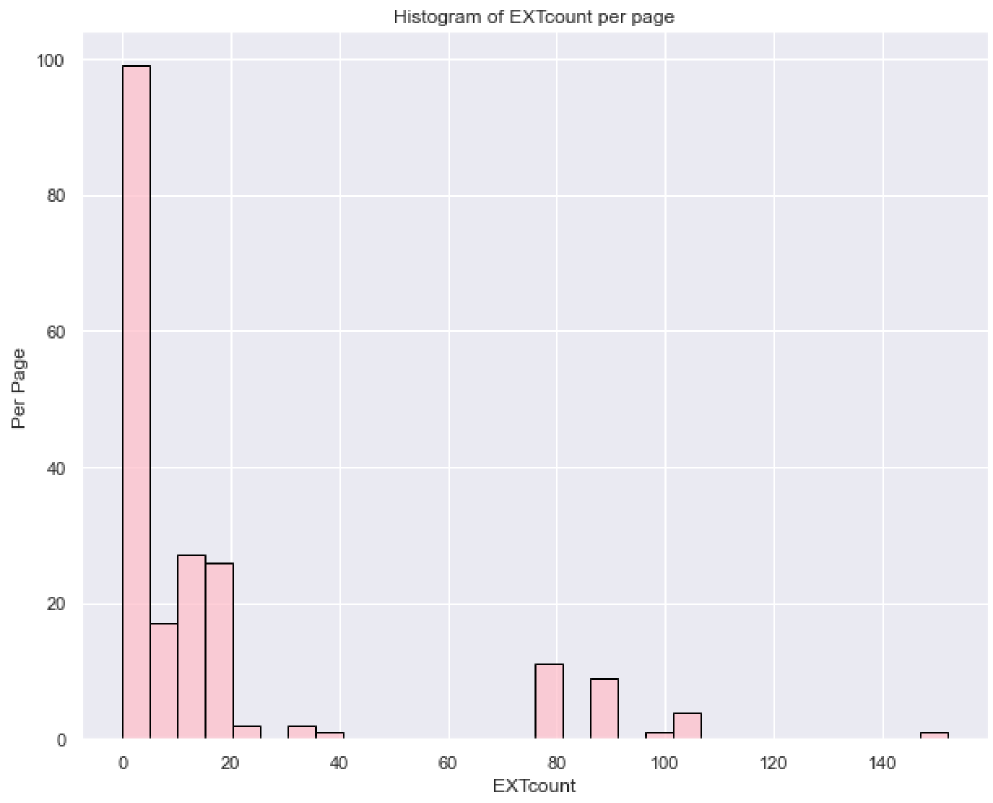
```
figure(figsize=(10, 8))

sns.histplot(df.INTcount, color = "orange", ec="black")
plt.ylabel('Per Page')
plt.xlabel('INTcount')
plt.title('Histogram of INTcount per page')
```



Histogram of EXTcount per

page:

```
 figure(figsize=(10, 8))

sns.histplot(df.EXTcount, color = "pink", ec="black")

plt.ylabel('Per                              Page')
plt.xlabel('EXTcount')
plt.title('Histogram of EXTcount per
page')
```

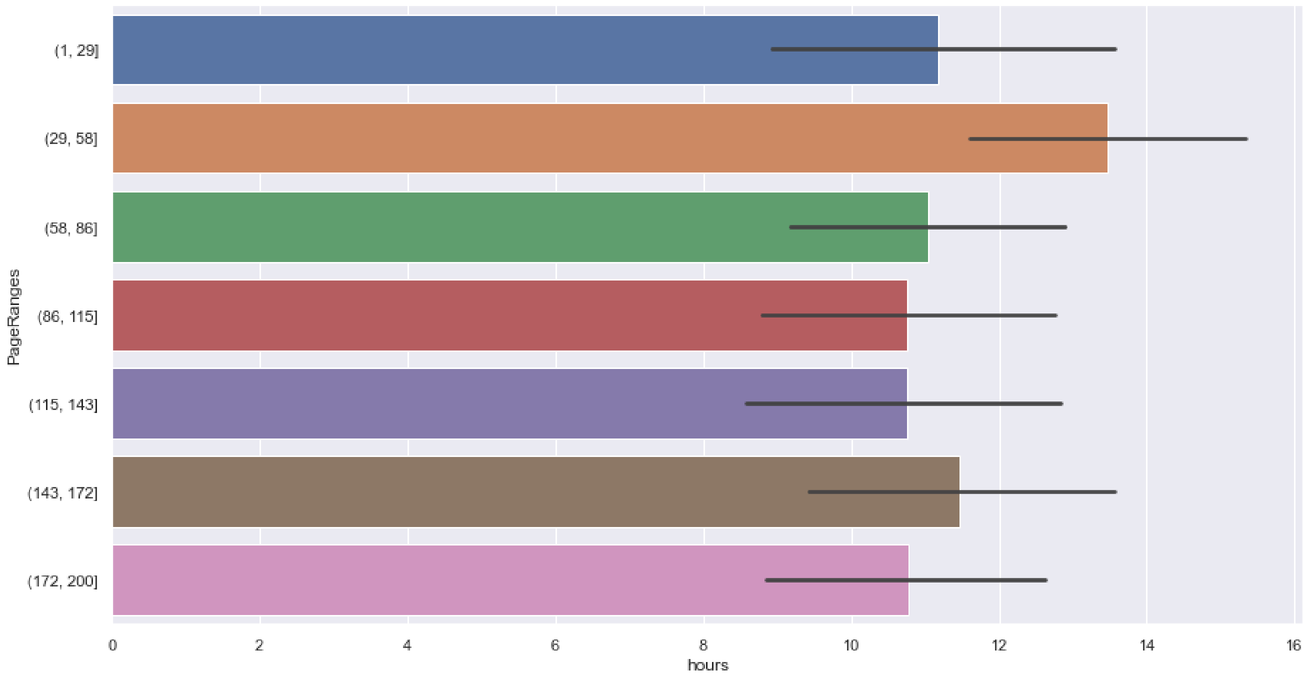


3. **Plot the hour of day at which the page was modified against the frequency (the numberof pages). Hint: you may want to perform a bit of data cleaning and remove records where the timestamp is None. Please, include the screenshots. The idea is to see in a simplified manner at what time of a day (GMT time) most modifications happen. (5 points)**

Solution

```
from datetime import
datetime

import pandas as pd

 import matplotlib.pyplot
as plt

import seaborn as sns

from matplotlib import
figure

#readinf making a copy of the csv file
df =
pd.read_csv('LinkInformation.csv')
df_using = df.copy()

#cleaning the data

df_using = df_using.mask(df_using.eq('None')).dropna()
```

```
df_using['formatted_time']      =      pd.to_datetime(df_using.TimeStamp)
df_using['hours'] = df_using['formatted_time'].dt.hour

#making ranges

df_using["ranges"] = pd.cut(df_using.PageCount, 7, right=False)

#making ranges into whole numbers

_, edges = pd.cut(df_using.PageCount, bins=7, retbins=True) labels =
[f'({abs(edges[i]):.0f}, {edges[i+1]:.0f}]' for i in range(len(edges)-1)]
df_using['PageRanges']      =      pd.cut(df_using.PageCount,      bins=7,
labels=labels)

#plotting the graph sns.set(rc =
{'figure.figsize':(15,8)})

sns.barplot(x= df_using.hours, y= df_using.PageRanges )
```



3. Short Questions (10 points)

3.1. Explain what falsifiable hypothesis is and give examples of both, falsifiable and non-falsifiable hypotheses. (5 points)

Solution

Hypothesis of falsifiability means that a theory is falsifiable if it can be contradicted by a statement with observable evidence. To ensure that a hypothesis is valid, you should be able to test it in different scenarios. Falsifiability is a requirement for a theory to be a scientific theory.

For example, Covid vaccines have different stages of trials before they are rolled out to the masses. Before they are tested, we can say that they are falsifiable as they have not been tested and

approved by the authorities.
Similarly, once Covid vaccines are tested in three different phases of trials - we can safely say that they have been tested with a specific percentage of accuracy and also within safety parameters. That means we can say that these vaccines are tested and non falsifiable under mentioned guidelines.

3.1. Explain what CDF plot shows and how to read it. You can provide an example if you want. (5 points)

Solution

CDF is also called Cumulative Density Function. It is used to understand probability for some function.
$F(x) = P(X \leq x)$ is the function for CDF.
The value ranges from [0,1].

In case of a continuous variable, it gives the area under the probability density from function from negative infinity to x.

For example, if X is the weight of a person and F(x) = 0.55, where x = 60kg.
Then we can say that there is a 45% chance that a person randomly selected from the dataset will have a weight greater than 60kg, and 55% that a person randomly selected from the dataset will have a weight lesser than or equal to 60kg.