Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS766P Data Mining – Assignment 4

Hasan Emre Erdemoglu

8 December 2020

Table of Contents:

Part 1:

Question 1.1 ..... **1**

Question 1.2 ..... **7**

Question 1.3 ..... **9**

Part 2:

Question 2.1 ..... **11**

Question 2.2 ..... **12**

Question 2.3 ..... **12**

Question 2.4 ..... **13**

Question 2.5 ..... **15**

Question 2.6 ..... **18**

Any code within this report could be provided upon request.

Part 1: Nov 17, 2020

Question 1.1:

The following code was written to inspect the HTML tags present in the given "income_table" URL.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from urllib.request import urlopen
from bs4 import BeautifulSoup

# Open the URL
url = "http://eecs.qmul.ac.uk/~emmanouilb/income_table.html"
html = urlopen(url)

# Get BS4 ready:
soup = BeautifulSoup(html, 'lxml')
#print(type(soup))

# Present the various tags present in the code: - This gives all.
soup.find_all(True)
```

Figure 1. Question 1.1 Part A Code

The following is a snippet of the last line of the code given above:

```
[<html>
 <body>
 <h1>ECS766P Data Mining - Week 9</h1>
 <p>The below table contains income data per country; the same table was used for the Week 3 lab.</p>
 <table class="table table-bordered table-hover table-condensed">
 <thead><tr><th title="Field #1">Region</th>
 <th title="Field #2">Age</th>
 <th title="Field #3">Income</th>
 <th title="Field #4">Online Shopper</th>
 </tr></thead>
 <tbody><tr>
 <td>India</td>
 <td align="right">49</td>
 <td align="right">86400</td>
 <td>No</td>
 </tr>
 <tr>
 <td>Brazil</td>
 <td align="right">32</td>
 <td align="right">57600</td>
 <td>Yes</td>
 </tr>
 <tr>
 <td>USA</td>
 <td align="right">35</td>
 <td align="right">64800</td>
 <td>No</td>
 </tr>
 <tr>
 <td>Brazil</td>
 <td align="right">43</td>
 <td align="right">73200</td>
 <td>No</td>
 </tr>
 <tr>
 <td>USA</td>
 <td align="right">45</td>
 <td align="right"></td>
 <td>Yes</td>
 </tr>
 <tr>
 <td>India</td>
 <td align="right">40</td>
 <td align="right">69600</td>
 <td>Yes</td>
 </tr>
```

Figure 2. Question 1.1 Part A - Output Snippet

Using the following code snippet, and the HTML page the following tags are extracted. Most of the tags are added in their respective hierarchical order.

- html: States that the content is written in HTML.
- body: Contains all the content within a page within its starting and ending tags.
- h1: Used in headings. Largest heading variation.
- p: Used for indicating plain text.
- tr: Used to indicate rows of a table.
- thead: Used to indicate table's header. 'tr' tag is used before thead; as the header is also a row.
    - th title: Makes the title more visible, bold; used to identify the fields.
- tbody: Indicates the body of the table. The data is stored within this section.
- td: Used to indicate columns of a table. It is used after constructing the row using 'tr' tag pointing to a single cell.
    - tdalign: Sets the alignment of the column entry.

Note that except for HTML tag all tags have their starting and ending tags. Ending tags are indicates with a slash (/) operator.

BeautifulSoup library can be used to mine this HTML page and the contents of the table of this page can be stored in a pandas dataframe in a structured manner. The method follows from the tutorial content of this lab. The code segment for generating the dataframe is shown below:

```
# Scrape and Clean the table - Form a dataframe

# Header Preperation:
header_list = []

column_labels = str(soup.find_all('th'))
cleantext_header = BeautifulSoup(column_labels, "lxml").get_text()  # extract the text without HTML tags
header_list.append(cleantext_header) # Add the clean table header to the list
# print(header_list)

df_header = pd.DataFrame(header_list)
df_header2 = df_header[0].str.split(', ', expand=True)
df_header2[0] = df_header2[0].str.strip('[')
df_header2[3] = df_header2[3].str.strip(']')
display(df_header2.head())

# Table Preperation:
table_list = []

# Print the first 10 table rows
rows = soup.find_all('tr')  # the 'tr' tag in html denotes a table row
#print(rows[:10])

# For every row in the table, find each cell element and add it to the list
for row in rows:
    row_td_cells = str(row.find_all('td'))
    row_cleantext = BeautifulSoup(row_td_cells, "lxml").get_text()  # extract the text without HTML tags
    table_list.append(row_cleantext)  # Add the clean table row to the list
#print(table_list)

df_table = pd.DataFrame(table_list)
df_table2 = df_table[0].str.split(', ', expand=True)
df_table2.head(10)

# Remove uneccesary characters
df_table2[0] = df_table2[0].str.strip('[')
df_table2[3] = df_table2[3].str.strip(']')

# Place everything in:
frames = [df_header2, df_table2]
df = pd.concat(frames)
df2 = df.rename(columns=df.iloc[0]) # We assign the first row to be the dataframe header
df3 = df2.drop(df2.index[0]) # We drop the replicated header from the first row of the dataframe
df3.head(10)
```

Figure 3. Question 1.1 Part B - Code Segment

The code above divides construction of the dataframe in two parts. In the first part; using ‘th’ tag; the header information is pulled as a string; then this string is processed to construct a dataframe only consisting of the header values.

In the second part, ‘tr’ tag is used to get all the rows of the given table; then for each row, using ‘td’ tag; column information is extracted. As ‘td’ search gives all the cells in the row simultaneously, the extracted information is saved to a list. This list is transformed into a pandas dataframe, then further processing is applied on the list of rows to remove unnecessary characters from these strings.

Finally, header and table dataframes are merged into a single dataframe. Note that Beautiful Soup is used to clean the extracted information from HTML related content in both parts. The following result is obtained:

	Region	Age	Income	Online Shopper
1	India	49	86400	No
2	Brazil	32	57600	Yes
3	USA	35	64800	No
4	Brazil	43	73200	No
5	USA	45		Yes
6	India	40	69600	Yes
7	Brazil		62400	No
8	India	53	94800	Yes
9	USA	55	99600	No
10	India	42	80400	Yes

Figure 4. Result Achieved by Question 1.2 Part B

Question 1.2:

This problem will be solved in 4 steps. First data will be loaded; then header information will be extracted. Later, the content of the table will be extracted. Finally, the header of the table and the content will be merged into a single dataframe.

The following code loads to webpage and handles necessary libraries that will be used for this question:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from urllib.request import urlopen
from bs4 import BeautifulSoup

# Open the URL
url = "http://eecs.qmul.ac.uk/postgraduate/programmes/"
html = urlopen(url)

# Get BS4 ready:
soup = BeautifulSoup(html, 'lxml')
#print(type(soup))

# Inspect HTML for finding a way to scrape this:
# soup.find_all(True)
```

Figure 5. Initialization for Question 1.2

The header is prepared in a similar fashion to what is used in this assignment. Two new entries for part-time and full-time URLs are appended to the end of the dataframe.

```
# Header Preperation:
header_list = []

column_labels = str(soup.find_all('th'))
cleantext_header = BeautifulSoup(column_labels, "lxml").get_text()  # extract the text without HTML tags
header_list.append(cleantext_header) # Add the clean table header to the list
#print(header_list)

df_header = pd.DataFrame(header_list)
df_header2 = df_header[0].str.split(', ', expand=True)
df_header2[0] = df_header2[0].str.strip('[')
df_header2[2] = df_header2[2].str.strip(']')
df_header2[3] = ['Part-time URL(2 year)']
df_header2[4] = ['Full-time URL(1 year)']
display(df_header2.head())
```

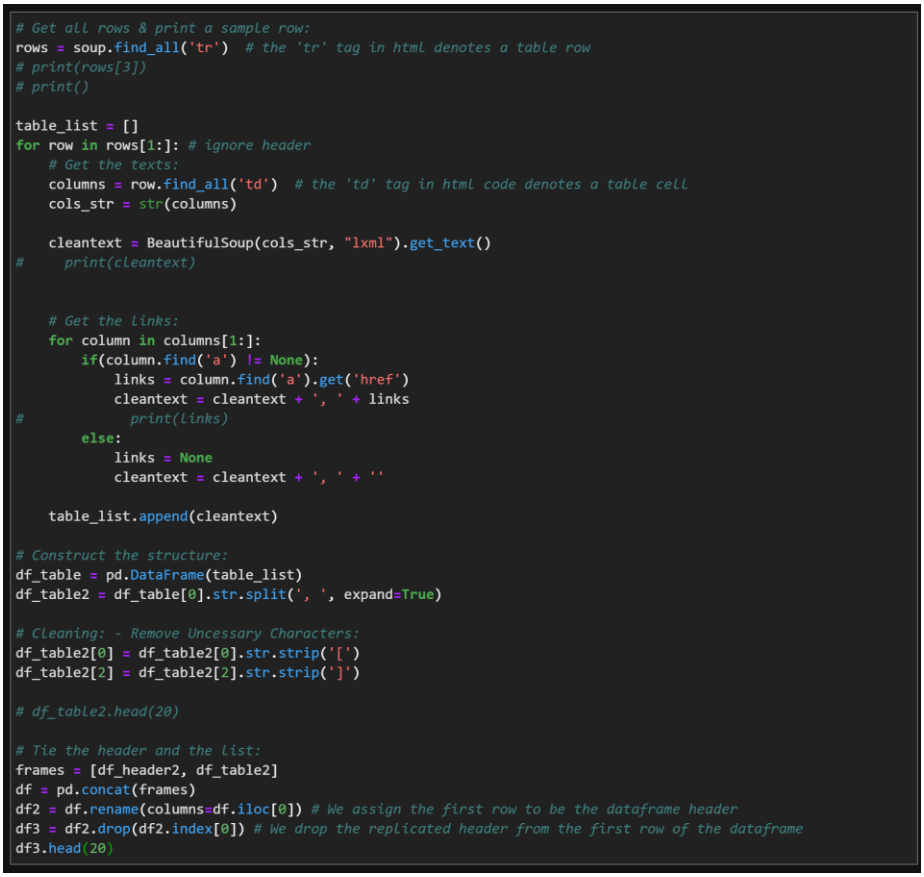
	0	1	2	3	4
0	Postgraduate degree programmes	Part-time(2 year)	Full-time(1 year)	Part-time URL(2 year)	Full-time URL(1 year)

Figure 6. Table Header Preparation

For extracting the table contents the following algorithm is used:

1. Get all table rows using ‘tr’ tags.
2. For each row, excluding the header row, use ‘td’ tags to extract all columns and using Beautiful Soup library extract the textual information. The output will be a string in the following order: [ProgramName, PartTimeCode, FullTimeCode]
3. For individual columns within a row, use ‘a’ tag to fetch href values, if they exist and add that to the string in Step 2, using CSV convention. If a URL does not exist; add an empty value separated by comma.
4. Append the extracted row information into a list.
5. Similar to the examples; construct a dataframe from the list; exploit the patterns in the data to remove unnecessary characters.
6. Combine header and table dataframes into a single dataframe.

The code and the output are given below:



```
# Get all rows & print a sample row:
rows = soup.find_all('tr')  # the 'tr' tag in html denotes a table row
# print(rows[3])
# print()

table_list = []
for row in rows[1:]: # ignore header
    # Get the texts:
    columns = row.find_all('td')  # the 'td' tag in html code denotes a table cell
    cols_str = str(columns)

    cleantext = BeautifulSoup(cols_str, "lxml").get_text()
#     print(cleantext)


    # Get the links:
    for column in columns[1:]:
        if(column.find('a') != None):
            links = column.find('a').get('href')
            cleantext = cleantext + ', ' + links
#             print(links)
        else:
            links = None
            cleantext = cleantext + ', ' + ''

    table_list.append(cleantext)

# Construct the structure:
df_table = pd.DataFrame(table_list)
df_table2 = df_table[0].str.split(', ', expand=True)

# Cleaning: - Remove Uncessary Characters:
df_table2[0] = df_table2[0].str.strip('[')
df_table2[2] = df_table2[2].str.strip(']')

# df_table2.head(20)

# Tie the header and the list:
frames = [df_header2, df_table2]
df = pd.concat(frames)
df2 = df.rename(columns=df.iloc[0]) # We assign the first row to be the dataframe header
df3 = df2.drop(df2.index[0]) # We drop the replicated header from the first row of the dataframe
df3.head(20)
```

Figure 7. The realization of Question 1.2’s algorithm

	Postgraduate degree programmes	Part-time(2 year)	Full-time(1 year)	Part-time URL(2 year)	Full-time URL(1 year)
1	Artificial Intelligence	I4U2	I4U1	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
2	Big Data Science	H6J6	H6J7	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
3	Computer Games		I4U4		https://www.qmul.ac.uk/postgraduate/taught/cou...
4	Computer Science	G4U2	G4U1	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
5	Computer Science by Research	G4Q2	G4Q1	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
6	Computing and Information Systems	G5U6	G5U5	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
7	Data Science and Artificial Intelligence by Co...		I4U5		https://www.qmul.ac.uk/postgraduate/taught/cou...
8	Electronic Engineering by Research	H6T6	H6T5	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
9	Internet of Things (Data)	I1T2	I1T0	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
10	Machine Learning for Visual Data Analytics	H6JZ	H6JE	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
11	Media and Arts Technology by Research		G4Q3		https://www.qmul.ac.uk/postgraduate/taught/cou...
12	Sound and Music Computing	H6T4	H6T8	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
13	Telecommunication and Wireless Systems	H6JD	H6JA	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
14	Digital and Technology Solutions (Apprenticeship)	I4DA		https://www.qmul.ac.uk/postgraduate/taught/cou...	

Figure 8. The output of the code listed in Figure 7.

Note that some programs do not have full-time or part-time options; hence they do not have associated URLs. The table also considers this.

Question 1.3:

The following image is given for this question:

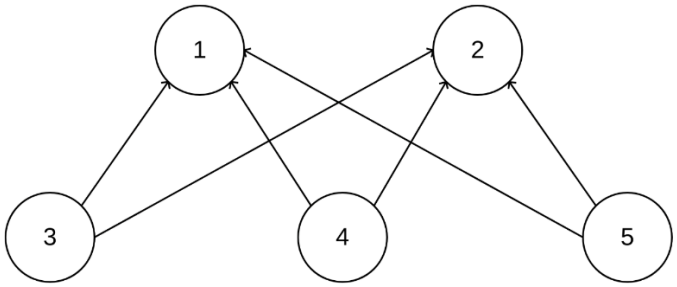


Figure 9. Graph for Question 1.3

Hubs are nodes which have many outgoing connections. In the image above, the hubs are 3, 4 and 5. Authority nodes have many incoming connections. These are nodes 1 and 2. As for the transition probabilities; if dead-end cases on nodes 1 and 2 are ignored and a uniform distribution is assumed the following situation will occur:

- Nodes 1 and 2 will only have transition probabilities to themselves with probability 1 as these are the authority nodes.
- Nodes 3,4,5 will have transition probabilities of 1/2 because each hub has 2 outgoing connections (no nodes 1 and 2). Due to probability measure rules; the probability measure of outgoing connections must add up to 1.

Using these values and assumptions, PageRank algorithm can be implemented for this graph. The equation for each node is as follows:

$$\pi(i) = \alpha\frac{1}{n} + (1-\alpha)\sum_{j \in \ln(i)} \pi(j)\, p_{ji}$$

This algorithm can be written for every node and the system of equations can be solved for this question. The teleportation factor alpha is kept as a unknown term.

Derivation of PageRank for all nodes:

$$\pi(1) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p_{11} + \pi(2)\, p_{21} + \pi(3)\, p_{31} + \pi(4)\, p_{41} + \pi(5)\, p_{51}]$$
$$= \alpha\frac{1}{5} + (1-\alpha)\left[\pi(1)\frac{1}{5} + \pi(2)\,\frac{1}{5} + \pi(3)\,0.5 + \pi(4)\,0.5\, + \pi(5)\,0.5\right]$$

$$\pi(2) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p_{12} + \pi(2)\, p_{22} + \pi(3)\, p_{32} + \pi(4)\, p_{42} + \pi(5)\, p_{52}]$$
$$= \alpha\frac{1}{5} + (1-\alpha)\left[\pi(1)\frac{1}{5} + \pi(2)\,\frac{1}{5} + \pi(3)\,0.5 + \pi(4)\,0.5 + \,\pi(5)\,0.5\right]$$

$$\pi(3) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p_{13} + \pi(2)\, p_{23} + \pi(3)\, p_{33} + \pi(4)\, p_{43} + \pi(5)\, p_{53}]$$
$$= \alpha\frac{1}{5} + (1-\alpha)\left[\pi(1)\frac{1}{5} + \pi(2)\,\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \,\pi(5)\,0\right]$$

$$\pi(4) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p_{14} + \pi(2)\, p_{24} + \pi(3)\, p_{34} + \pi(4)\, p_{44} + \pi(5)\, p_{54}]$$
$$= \alpha\frac{1}{5} + (1-\alpha)\left[\pi(1)\frac{1}{5} + \pi(2)\,\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \,\pi(5)\,0\right]$$

$$\pi(5) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p_{15} + \pi(2)\, p_{25} + \pi(3)\, p_{35} + \pi(4)\, p_{45} + \pi(5)\, p_{55}]$$
$$= \alpha\frac{1}{5} + (1-\alpha)\left[\pi(1)\frac{1}{5} + \pi(2)\,\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \,\pi(5)\,0\right]$$

Simplifying terms in matrix form:

$$\begin{bmatrix}\pi(1)\\ \pi(2)\\ \pi(3)\\ \pi(4)\\ \pi(5)\end{bmatrix} = \alpha\begin{bmatrix}1/5\\ 1/5\\ 1/5\\ 1/5\\ 1/5\end{bmatrix} + (1-\alpha)\begin{bmatrix}1/5 & 1/5 & 0.5 & 0.5 & 0.5\\ 1/5 & 1/5 & 0.5 & 0.5 & 0.5\\ 1/5 & 1/5 & 0 & 0 & 0\\ 1/5 & 1/5 & 0 & 0 & 0\\ 1/5 & 1/5 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\pi(1)\\ \pi(2)\\ \pi(3)\\ \pi(4)\\ \pi(5)\end{bmatrix}$$

Part 2: Nov 24, 2020

Question 2.1:

The following sentences are given for this question:

- Data refers to characteristics that are collected through observation.
- A dataset can be viewed as a collection of objects.
- Data objects are described by a number of attributes.
- An attribute is a characteristic or feature of an object.

Part A:

This part constructs the document term matrix for the documents. Each sentence is taken as a document. When stop words are removed and preparation is made, the following output is achieved:

- data refer characteristic collect observation
- dataset can view collection object
- data object describe number attribute
- attribute characteristic feature object

Finally, unique words from these documents can be taken and counted within the entire dataset for each document. Note that auxiliary verb "can" is also taken as a common verb (in general sense), hence removed from the calculations.

Table 1. Document-Term Matrix for given Dataset

	data	*refer*	*characteristic*	*collect*	*observation*	*dataset*	*view*	*collection*	*object*	*describe*	*number*	*attribute*	*feature*
1	1	1	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	1	1	1	0	0	0	0
3	1	0	0	0	0	0	0	0	1	1	1	1	0
4	0	0	1	0	0	0	0	0	1	0	0	1	1

Part B:

Part B will use the document-term matrix in Table 1 to calculate inverse document frequency (IDF) for all terms identified in Part A. IDF has the following formula: $IDF(w) = \log_{10}\left(\frac{|D|}{|D_w|}\right)$

Table 2.

Word	data	refer	characteristic	collect	observation	dataset	view	collection	object	describe	number	attribute	feature
IDF	0.301	0.602	0.301	0.602	0.602	0.602	0.602	0.602	0.125	0.602	0.602	0.301	0.602

A detailed example working on word *characteristic*. Other calculations follow from the same intuition. Note that denominator can be either 1,2,3 or 4 in this example so 4 set of idf values are present.

$$DF('characteristic') = \log_{10}\left(\frac{4}{|D_{characteristic}|}\right) = \log_{10}\frac{4}{2} = 0.301 \text{ (3 significant figure)}$$

Question 2.2:

The following timeseries data is given:

t	1	4	6
y	2	8	5

Using linear interpolation y values of the timeseries for t=3 and t=5 can be calculated using the following formula where i < j; y_i and y_j are the values of the timeseries at times t_i and t_j respectively.

$$y = y_i + \left(\frac{t - t_i}{t_j - t_i}\right) \cdot (y_j - y_i)$$

Using this for t = 3: (i = 1, j = 4)

$$y = 2 + \left(\frac{3 - 1}{4 - 1}\right) \cdot (8 - 2) = 6$$

For t = 5: (i = 4, j = 6)

$$y = 8 + \left(\frac{5 - 4}{6 - 4}\right) \cdot (5 - 8) = 6.5$$

Therefore, the results for t=3 and t=5 are 6 and 6.5 respectively.

Question 2.3:

Following timeseries data is given. Binning will be performed on this time series, using k=3 values per bin.

y = 0.1, 0.15, 0.2, 0.2, 0.3, 0.4, 0.25, 0.6, 0.5

When a bin size of 3 is used, the following will be the binned groups:

y = [0.1, 0.15, 0.2], [0.2, 0.3, 0.4], [0.25, 0.6, 0.5]

For assigning the values for binning the following equation is used:

$$y'_{i+1} = \frac{\sum_{r=1}^{k} y_{ik+r}}{k}$$

Basically, this equation averages the values inside the bins and summarize them into a single value. The output hence becomes:

y = [0.15], [0.3], [0.45]

y = 0.15, 0.3, 0.45

Question 2.4:

The following code was written to import the CSV file, turning it to a NumPy array and flattening this array into a single dimension. The original timeseries is also plotted:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas
from pandas import read_csv


# Load series, convert to numpy then flatten it.
series = read_csv('timeseries.csv', header=None).to_numpy()
series = series.flatten()
# print(series.shape)


# Plot of Original Time Series:
plt.figure(figsize=(10, 4))
plt.title('$|X|$ - magnitude of DFT')
plt.plot(series)
plt.xlabel('Frequency (index $k$)')
plt.tight_layout()
```

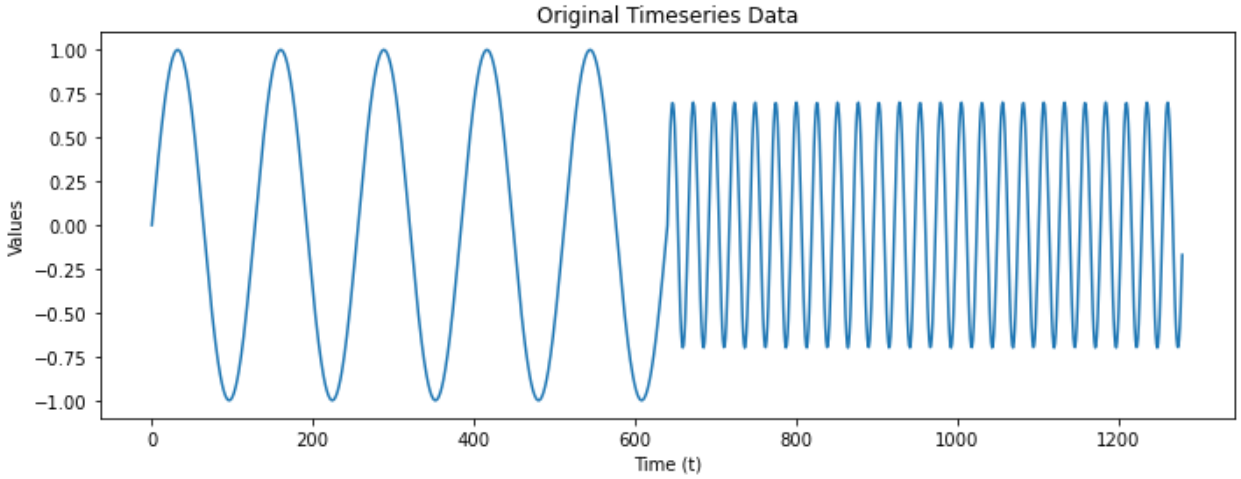


Figure 10. The plot of the Original Timeseries Data

Using a similar approach to the Lab Assignment; the following code can be written to extract the DFT of this time series data using NumPy's built-in FFT method. The amplitude response of the FFT of this dataset is given below along with the code that generated it:

```
# Compute DFT of timeseries using FFT:

Xfft = np.fft.fft(series)


# Plot of DFT

plt.figure(figsize=(10, 4))

plt.title('$|X|$ - magnitude of DFT')

plt.plot(np.abs(Xfft), 'k')

plt.xlabel('Frequency (index $k$)')

plt.tight_layout()
```

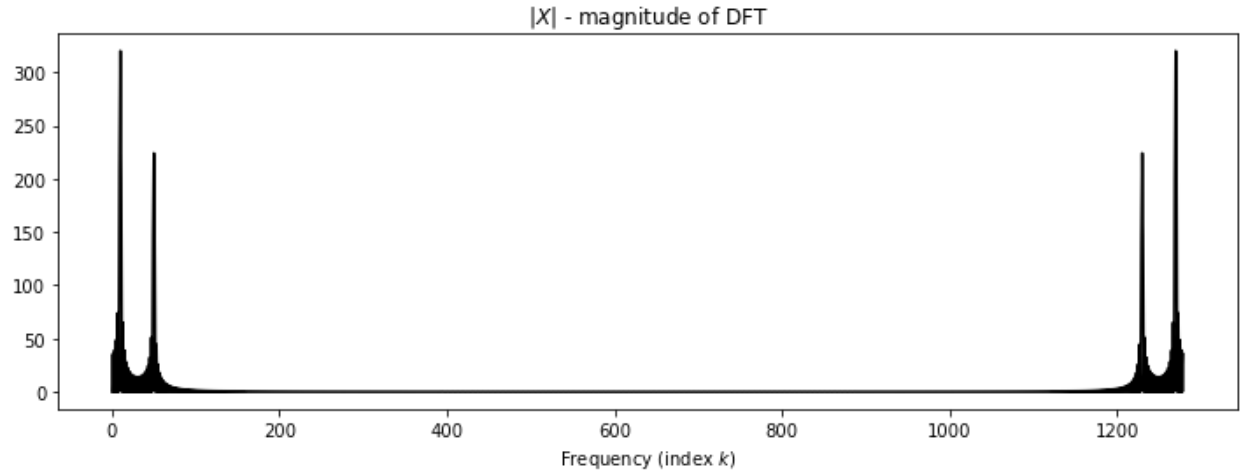


Figure 11. FFT of Given Timeseries Data - Magnitude Response

As for the predominant frequencies there are two frequencies which are clearly visible in Figure 11. This is also visible in Figure 10 as well the original timeseries data consists of two sinusoids. The dominant frequencies shown in Figure 11 is the frequencies of these sinusoids.

Question 2.5:

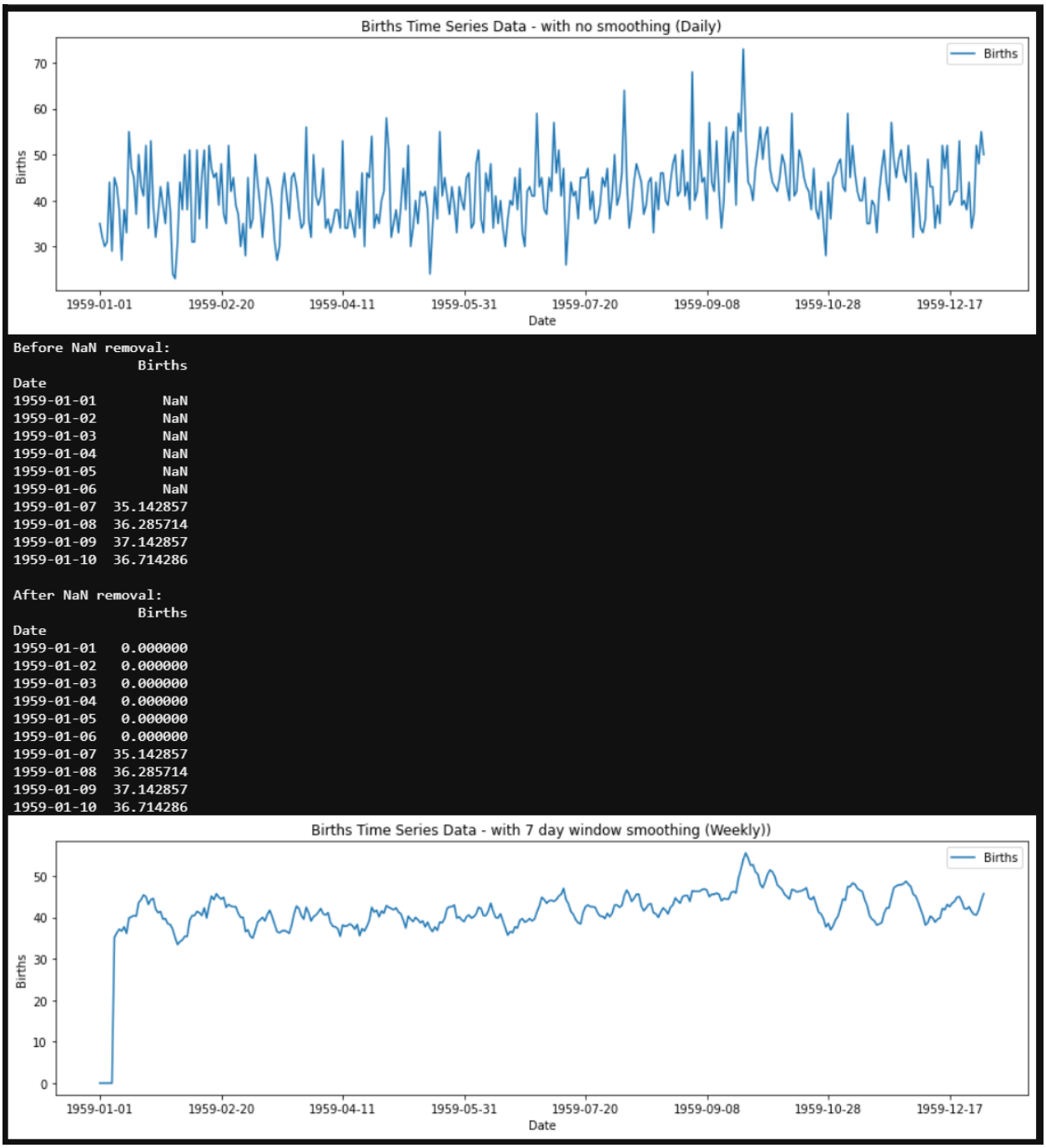
Births dataset will be used in this question. First, trailing moving average smoothing will be applied. The window size will be 7, as the question requires smoothing to be applied in a weekly level. Any NaN values will be replaced with zero (happens only on beginning of the dataset due to the smoothing effect). Later, smoothed dataset will be used to generate predictions for the first 5 days of 1960. Prediction will be based on AR model with p=2 and ARMA model with p and q = 2. Note that both implementations are based on the lab tutorial's implementations as they were suitable for this question as well.

Smoothing:

The following code is written for this section:

```
from pandas import read_csv
import matplotlib.pyplot as plt
series = read_csv('births.csv', header=0, index_col=0)
# print(series.head())
series.plot(figsize=(15,4))
plt.xlabel('Date')
plt.ylabel('Births')
plt.title('Births Time Series Data - with no smoothing (Daily)')
plt.show()
# Smooth by trailing moving average smoothing - Window size = 1 week
# Perform trailing moving average smoothing
rolling = series.rolling(window=7) # using a window of weekly samples
print('Before NaN removal:')
print(rolling_mean.head(10))
print()
# Replace NaN with 0.
# print(rolling.fillna(value=0))
rolling_mean = rolling.mean().fillna(0)
print('After NaN removal:')
print(rolling_mean.head(10))
rolling_mean.plot(figsize=(15,4))
plt.xlabel('Date')
plt.ylabel('Births')
plt.title('Births Time Series Data - with 7 day window smoothing (Weekly))')
plt.show()
```

In the code above, first the original dataset is plotted. Then, rolling averaging with a window size of 7 is used to smooth the data. Later using .mean() and .fillna(0) operations NaN values are removed from the dataset. Finally, the smoothed dataset is plotted as well. All outputs for this code is given below:




```
Before NaN removal:
              Births
Date
1959-01-01        NaN
1959-01-02        NaN
1959-01-03        NaN
1959-01-04        NaN
1959-01-05        NaN
1959-01-06        NaN
1959-01-07  35.142857
1959-01-08  36.285714
1959-01-09  37.142857
1959-01-10  36.714286

After NaN removal:
              Births
Date
1959-01-01   0.000000
1959-01-02   0.000000
1959-01-03   0.000000
1959-01-04   0.000000
1959-01-05   0.000000
1959-01-06   0.000000
1959-01-07  35.142857
1959-01-08  36.285714
1959-01-09  37.142857
1959-01-10  36.714286
```


Figure 12. Outputs of Smoothing Process

AR Model:

Predictions and outputs are shown below. To predict 5 values, in function called 'predict' a parameter of 4 is added. This is also used in ARMA model as well. Parameter p is changed in the "lags" portion; when initializing the Auto Regressive Model.

```
# Perform timeseries forecasting using the smoothed dataset
# in order to predict daily births for the first 5 days of 1960, using the models below.

# AR Model p=2:
# Initialise
from statsmodels.tsa.ar_model import AutoReg

# Fit Autoregressive model
model = AutoReg(rolling_mean, lags=2, old_names=False) # "lags" indicates the model order
model_fit = model.fit()

# Make prediction
yhat = model_fit.predict(len(rolling_mean), len(rolling_mean)+4) # arguments denote which dataset indices to predict
print(yhat)
```

```
1960-01-01    45.380177
1960-01-02    44.960852
1960-01-03    44.590676
1960-01-04    44.271699
1960-01-05    43.997395
Freq: D, dtype: float64
```

Figure 13. Code and results of AR model

ARMA Model:

Using a similar approach to AR Model, the following code is compiled to do predictions on ARMA model. Again, p and q values are set to 2 in initialization step. ARIMA model is used in this section; as leaving order as 0 makes ARIMA equivalent to ARMA.

```
# Fit ARMA model
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(rolling_mean, order=(2, 0, 2)) # p=2, q=2
model_fit = model.fit()

# Make prediction
yhat = model_fit.predict(len(rolling_mean), len(rolling_mean)+4) # arguments denote which dataset indices to predict
print(yhat)
```

```
1960-01-01    45.810247
1960-01-02    45.818766
1960-01-03    45.728092
1960-01-04    45.564016
1960-01-05    45.347305
Freq: D, Name: predicted_mean, dtype: float64
```

Figure 14. Code and Results on ARMA Model

Question 2.6:

For this section, the given set of Wikipedia pages are first loaded in memory using the code below:

```
import pandas as pd
import wikipedia


articles=['supervised learning', 'unsupervised learning', 'semi-supervised learning', 'association rule learning', 'anomaly detection', 'cluster analysis',
          'dimensionality reduction', 'regression analysis', 'statistical classification', 'data mining', 'data warehouse', 'online analytical processing']
wiki_lst=[]
title=[]

# Load wikipedia articles
for article in articles:
    print("loading content: ",article)
    wiki_lst.append(wikipedia.page(article).content)
    title.append(article)
```

```
loading content:  supervised learning
loading content:  unsupervised learning
loading content:  semi-supervised learning
loading content:  association rule learning
loading content:  anomaly detection
loading content:  cluster analysis
loading content:  dimensionality reduction
loading content:  regression analysis
loading content:  statistical classification
loading content:  data mining
loading content:  data warehouse
loading content:  online analytical processing
```

Figure 15. Loading Given Pages to Memory

Later TFIDF Vectorizer is used to pre-process the dataset so that it can be used in K-Means clustering:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words={'english'})
X = vectorizer.fit_transform(wiki_lst) # Create tf-idf feature of the wikipedia dataset

print(X.shape) # Print dimensions of tf-idf feature
```

```
(12, 4219)
```

Figure 16. Data Preparation

K-Means experiments are done in the range of 2 to 12; where 12 is the number of documents this dataset has. Elbow method is used here; and it is decided there is a kink around 4.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

Sum_of_squared_distances = []
K = range(1,12)

for k in K:
    km = KMeans(n_clusters=k, max_iter=200, n_init=10)
    km = km.fit(X)
    Sum_of_squared_distances.append(km.inertia_)

print(Sum_of_squared_distances)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method')
plt.show()
```

```
[4.885926455251841, 4.106571829664187, 3.532296087562986, 2.9076915753219086, 2.504059424387943, 2.0708074084742236, 1.623044290040304, 1.225902299541462, 0.8623317155060548, 0.5234699055386012, 0.2587147332133297]
```

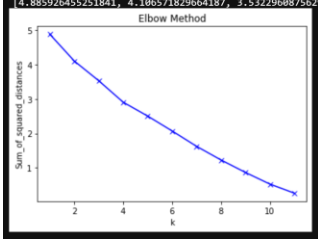


Figure 17. K-Means Experiment

This section runs the clustering algorithm for K=4 and displays the results:

```
# Fit k-means model with k=4
true_k = 4
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=200, n_init=10)
model.fit(X)

# Print list of documents and associated clusters
labels=model.labels_
wiki_cl=pd.DataFrame(list(zip(title,labels)),columns=['title','cluster'])
print(wiki_cl.sort_values(by=['cluster']))
```

```
                          title  cluster
9                     data mining        0
10                data warehouse        0
11  online analytical processing        0
5                cluster analysis        1
6       dimensionality reduction        1
8     statistical classification        1
0             supervised learning        2
1           unsupervised learning        2
2         semi-supervised learning        2
3       association rule learning        2
7             regression analysis        2
4               anomaly detection        3
```

Figure 18. Document Clustering with K=4