Data Mining: Web Mining and Data Mining Applications

Background:

Data mining is a process to extract useful information from a vast amount of data. It is used to discover new, accurate and useful patterns in data, looking for meaning.

Data: refers to characteristics /numerical/categorical which are collected through observation.

Datasets: collections of items which are described by a set of attributes.

Data Mining starts with collection or sourcing of raw data. Raw data can have multiple issues like poor data quality such as noisy data, dirty data, missing values, inexact or incorrect values, inadequate data size and poor representation in data sampling.

Web Mining:

Web mining technique is used for automatic discovery and information extraction from Web documents and services. There are different types of web mining based on the applications and usage. They have been classified as below:

1. Web Content Mining:
   Web content material mining is an utility for extracting beneficial records from the content material of internet files. Web content material includes numerous varieties of information, consisting of textual content, images, audio, and video. Content information is a hard and fast of data designed with the aid of using an internet page. It can offer powerful and exciting styles in your needs. Text files are associated with textual content mining, gadget learning, and herbal language processing. This mining is likewise referred to as textual content mining. This sort of mining scans and mines agencies of textual content, images, and internet pages, relying on what you type.

2. Web Structure Mining:
   Web Structure mining is an application for retrieving structural information from the Internet. The structure of a web graph consists of web pages, which are nodes, and hyperlinks, which are edges that connect related pages. Structural analysis basically shows a structured summary of a specific website. Defines relationships between web pages with information or direct links. Analysing web structure can be very useful in determining the relationship between two commercial websites.

3. Web Usage Mining:
   Internet Usage Mining is an application for identifying or discovering interesting usage patterns in large data sets. And with these templates, you can understand user behaviour and more. In Internet mining analysis, users access data on the Internet and collect data in log form. Therefore, Internet-enabled mining is also known as log mining.

Q1:
You are provided with the following URL: http://eecs.qmul.ac.uk/~emmanouilb/income_table.html.
This webpage includes a table on individuals' income and shopping habits - the same that was used in the Week 3 lab.

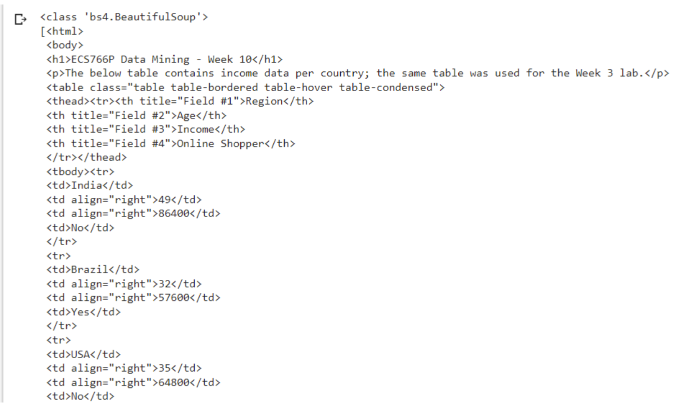
1. Inspect the HTML code of the above URL, and provide a short report on the various tags present in the code. What is the function of each unique tag present in the HTML code? [0.5 marks out of 5]

Following code has been written for the inspection of HTML tags present in the given “income_table” URL. From the code provided below and the HTML page the following tags have been extracted. Most of the tags are added in their respective hierarchical order.

- html: It states the content which is written in HTML.
- body: Contains all the content within a page with its starting and ending tags.
- h1: Utilized in headings. Largest heading variation.
- p: Utilized for indicating plain text.
- tr: Indicates rows of a table.
- thead: Indicates 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 indicated with a slash (/) operator.

#CODE	Output

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)
	

```
<class 'bs4.BeautifulSoup'>
[<html>
 <body>
 <h1>ECS766P Data Mining - Week 10</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>
```

2. Using Beautiful Soup, scrape the table and convert it into a pandas dataframe. Perform data cleaning when necessary to remove extra characters (no need to handle missing values). In the report include the code that was used to scrape and convert the table and provide evidence that the table has been successfully scraped and converted (e.g. by displaying the contents of the dataframe). [1 mark out of 5]

The BeautifulSoup library is used for HTML pages, and the contents of the tables on this page are stored in a panda's data frame in a structured way. This method is based on the tutorial content of this exercise. The code segments used to generate the data frame are shown below in the '#CODE' section.

The dataframe has been divided into two parts which are described below:
For the first part the usage of 'th'tag is done which pulls the header information as a string. After which this string is processed for construction of a dataframe consisting of only header values.

In second part, the 'tr' tag is used for getting all the rows of given table and for each row the 'td' tag is used to extract information from columns. The 'td' search gives all the cells in the row simultaneously and extracted information is saved into a list. The list is then transformed into the a pandas dataframe with further processing applied on the list of row to debar unwanted characters from strings.

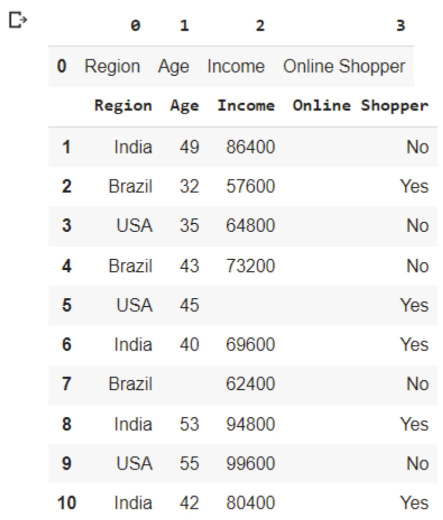
```
#CODE:

#A1-2

# 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)

Output:
Finally, the header and table dataframes are merged into single dataframe.

Note: Beautiful Soup is used to clean up the information extracted from both parts of the HTML related content. The result obtained is as shown in the figure below:



```
        0    1       2               3
0  Region  Age  Income  Online Shopper

   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
```

<u>**Q2:**</u>

The following steps have been implemented to solve the problem:

- Step 1: The data is loaded
- Step 2: The header information is extracted.
- Step 3: The content of the table is to be extracted.
- Step 4: Merging header of the table and content into a single dataframe.

#CODE:

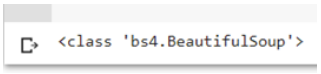
Part 1: Following code loads to webpage and handles necessary libraries that will be used for this question.

---

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)
```

Output:

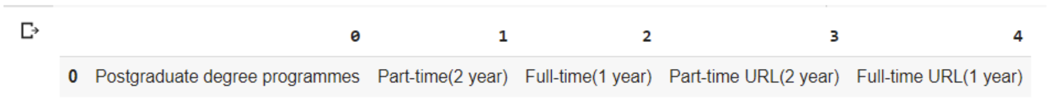


```
<class 'bs4.BeautifulSoup'>
```

Part 2: Header preparation:- Two new entries for part-time and full-time URLs are appended to the end of the dataframe.

```
# Header Preparation:
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())
```

Output:



	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)

Part 3:
For extracting the table contents the following algorithm is used:
1. Using 'tr' tags to get all table rows.
2. Using 'td' and BeautifulSoup

- **Using 'td' tags to extract all columns except the header row.**
- **Beautiful Soup library is used to extract information from texts.**

- **Output is in a string format stated as follows: [ProgramName, PartTimeCode, FullTimeCode]**

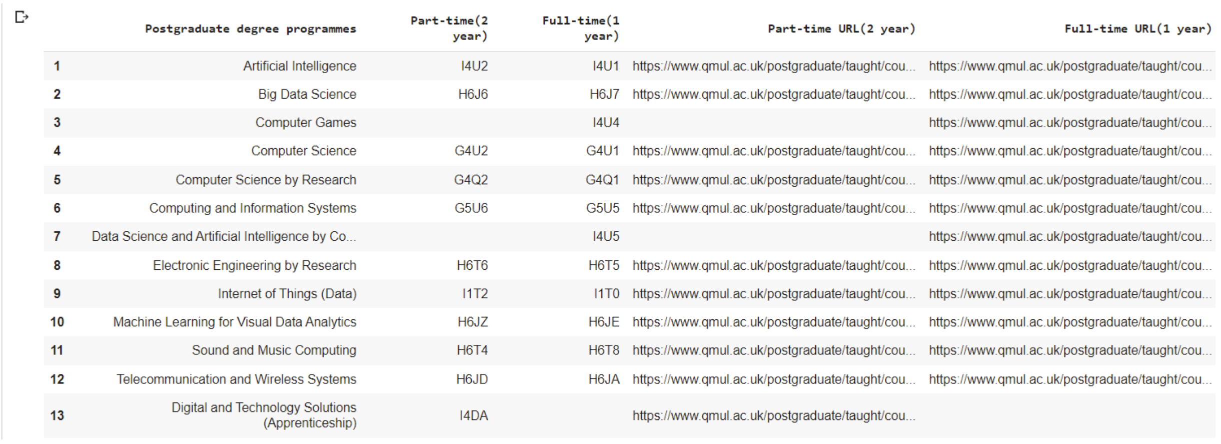
3. Tag 'a' and

- **Use the `a` tag to get the href value for each column in the row. If it exists, use a CSV rule to add it to the string in step 2.**
- **If the URL does not exist, add a comma-separated blank value.**

4. Appending the extracted row information into a list.
5. Constructing a dataframe from the list to exploit patterns in data to remove unnecessary characters.
6. Merge header and table dataframes into a single dataframe.

```
# 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)
```

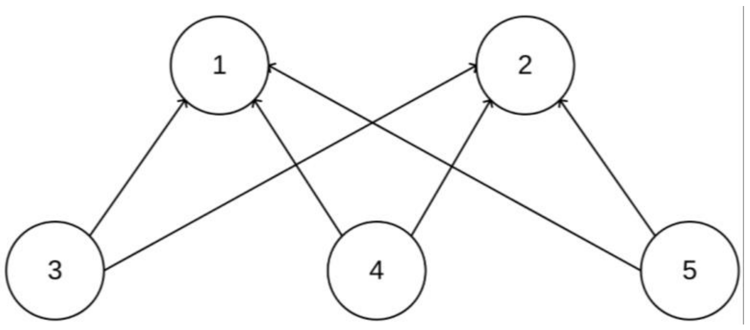
Output:



	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	Sound and Music Computing	H6T4	H6T8	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
12	Telecommunication and Wireless Systems	H6JD	H6JA	https://www.qmul.ac.uk/postgraduate/taught/cou...	https://www.qmul.ac.uk/postgraduate/taught/cou...
13	Digital and Technology Solutions (Apprenticeship)	I4DA		https://www.qmul.ac.uk/postgraduate/taught/cou...	

Note: Some programmes do not associated URLS as there is no option for full-time or part-time options.

<u>Q3:</u>



Here nodes 3, 4 and 5 are hubs as they have many outgoing connections and nodes 1 and 2 are authority nodes as they have many incoming connections. 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 occurs:

- 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.

With the above mentioned assumptions PageRank algorithm can be implemented for this graph. The equation for each node is as follows:

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

Derivation of PageRank for all nodes:

$\pi(1) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p11 + \pi(2)\, p21 + \pi(3)\, p31 + \pi(4)\, p41 + \pi(5)\, p51]$

$\quad = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\frac{1}{5} + \pi(2)\frac{1}{5} + \pi(3)\, 0.5 + \pi(4)\, 0.5 + \pi(5)\, 0.5]$

$\pi(2) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\, p11 + \pi(2)\, p21 + \pi(3)\, p31 + \pi(4)\, p41 + \pi(5)\, p51]$

$= \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\frac{1}{5} + \pi(2)\frac{1}{5} + \pi(3)\,0.5 + \pi(4)\,0.5 + \pi(5)\,0.5]$

$\pi(3) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\,p11 + \pi(2)\,p21 + \pi(3)\,p31 + \pi(4)\,p41 + \pi(5)\,p51]$
$= \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\frac{1}{5} + \pi(2)\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \pi(5)\,0]$

$\pi(4) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\,p11 + \pi(2)\,p21 + \pi(3)\,p31 + \pi(4)\,p41 + \pi(5)\,p51]$
$= \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\frac{1}{5} + \pi(2)\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \pi(5)\,0]$

$\pi(5) = \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\,p11 + \pi(2)\,p21 + \pi(3)\,p31 + \pi(4)\,p41 + \pi(5)\,p51]$
$= \alpha\frac{1}{5} + (1-\alpha)[\pi(1)\frac{1}{5} + \pi(2)\frac{1}{5} + \pi(3)\,0 + \pi(4)\,0 + \pi(5)\,0]$

For simplification we will convert to matrix form:

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

Data Mining Applications:

Q1:

- 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.

1. Construct and display the document-term matrix for the above documents. Remove all stop words (here consider as stop words: articles, prepositions, conjunctions, pronouns, and common verbs) and punctuation marks; convert any plural nouns/adjectives to their singular form; and convert verbs to the present tense and first person singular form, before you construct the matrix. [1 mark out of 5]

Document matrix:

	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

2. Using the above constructed document-term matrix, calculate the inverse document frequency idf(w) for all words w you have identified from question 1(a). [1 mark out of 5]

From the above table the computation of inverse document frequency(IDF) for all terms is done using following formula: $\log_{10}\left|\frac{D}{Dw}\right|$.

Example has been executed as shown below for word 'data'. Similar calculations have been carried out for other words and inserted in the table below.

Note: Here the denominator can be 1,2,3,4.

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

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

Q2:

Y = {0.1,0.15,0.2,0.2,0.3,0.4,0.25,0.6,0.5}

K = 3, the binning is done as follows:
- Y(1) = {0.1,0.15,0.2}
- Y(2) = {0.2,0.3,0.4}
- Y(3) = {0.25,0.6,0.5}

For assigning the values for binning the following equation is used: $y_{i+1} = \frac{\sum_{r=1}^{k} y_{ik+r}}{k}$

The equation computes the average of values present inside each bin. The output is as follows:
y = [0.15], [0.3], [0.45]
y = 0.15, 0.3, 0.45

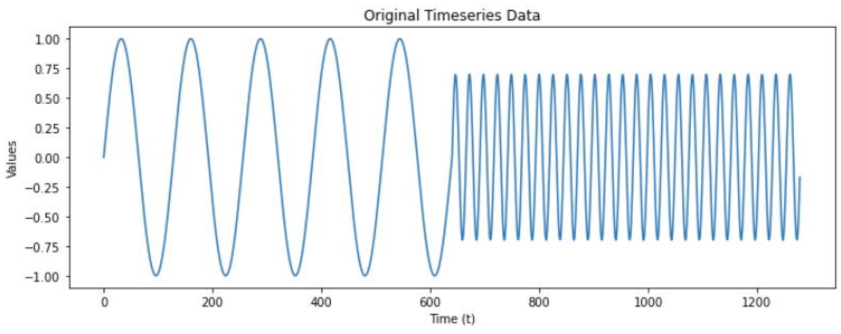
Q3:

#CODE:

#A3

Original time series is being plotted.

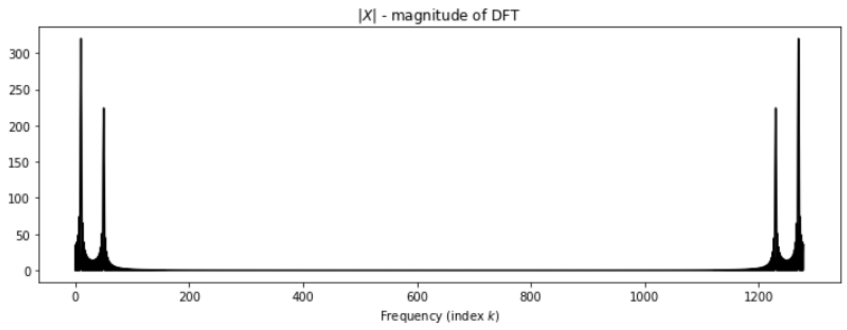
```
import numpy as np
import matplotlib.pyplot as plt
import pandas
from pandas import read_csv

# 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('Original Timeseries Data')
plt.plot(series)
plt.xlabel('Time (t)')
plt.ylabel('Values')
plt.tight_layout()
```



Similar to the approach provided in Lab Assignment below provided code has been written for extracting DFT of time series using built-in FFT method. As for the predominant frequencies there are two frequencies which are clearly visible in the output below.

```
# 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()
```

Output:



Q4:

Steps followed:

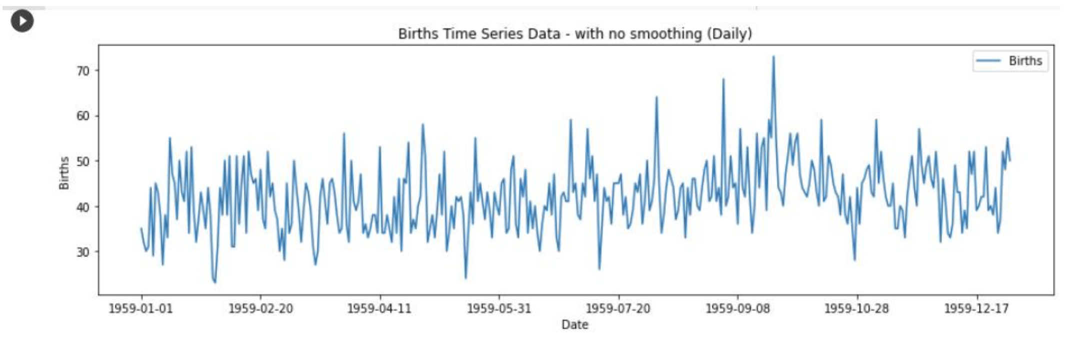
- Application of trailing moving average smoothing, window size = 7 (smoothing to be applied in a weekly level. The NaN values are replaced with 0.
- Smooth dataset is used for generation of predictions on the initial 5 days of 1960. Predictions are based on AR model p = 2 and ARMA model : p & q =2.

Note: Both implementations are based on the lab tutorial's implementations as they were suitable for this question as well.

In the code provided below, firstly the original dataset is plotted. Then rolling averaging with a window size of 7 is used to smooth the data. In the next step, use of .mean() and .fillna(0) operations NaN values are removed from the dataset. Lastly, the smoothed dataset is plotted as well. All outputs have been provided in the Output section of this question.

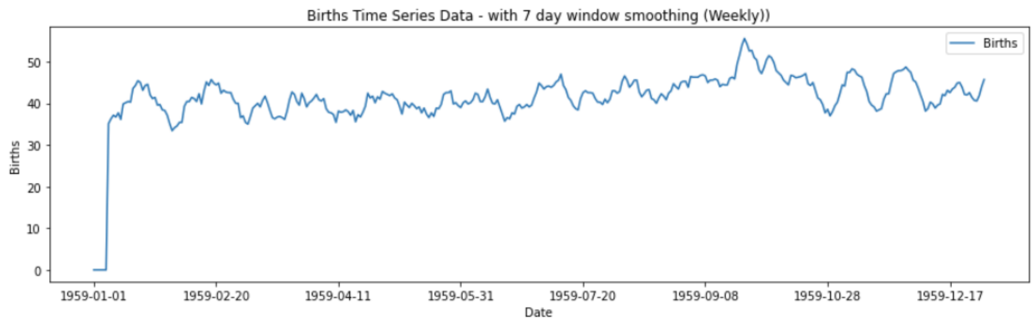
#CODE:

```
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()
```



```
Before 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
```

```
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
```



AR Model:

from statsmodels.tsa.ar_model import AutoReg
# 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
/usr/local/lib/python3.7/dist-packages/
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/
  UserWarning,
```

ARMA model:

```
import warnings
warnings.filterwarnings('ignore')

# 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    45.810249
1960-01-02    45.818768
1960-01-03    45.728095
1960-01-04    45.564020
1960-01-05    45.347310
Freq: D, Name: predicted_mean, dtype: float64
```

Q5:

1. The given set of Wikipedia pages are first loaded in memory using the code below.
2. Next, the TFIDF Vectorizer is used for pre-processing of the dataset to be used in K-Means clustering.
3. K-Means experiments are done in the range of 1 to 6 where 6 is the number of documents contained in this dataset. Elbow method is used here and a kink around 3.

#CODE:

```
#A5
import pandas as pd
import wikipedia

articles = ['anomaly detection', 'cluster analysis', 'k-means clustering','data warehouse', 'association rule learning', 'datamining']

wiki_lst=[]
title=[]

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

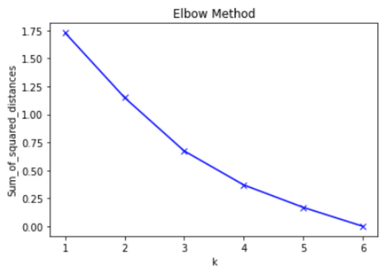
```
loading content:  anomaly detection
loading content:  cluster analysis
loading content:  k-means clustering
loading content:  data warehouse
loading content:  association rule learning
loading content:  datamining
```

```
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
```

```
(6, 3511)
```

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

Sum_of_squared_distances = []
K = range(1,6)

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()
```

```
[1.7283937109880239, 1.1491495907231792, 0.6730241474113106, 0.3694960185597148, 0.16884821376431258, 4.3298697960381105e-15]
```



```
# 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
3           data warehouse        0
5               datamining        0
4  association rule learning      1
0        anomaly detection        2
1         cluster analysis        3
2        k-means clustering       3
```

References : https://www.geeksforgeeks.org/web-mining/