**UNIVERSITY OF MUMBAI**

****

**DEPARTMENT OF COMPUTER SCIENCE**

**M.SC (Computer Science)**

**CERTIFICATE**

Certified that the work entered in this journal was

done in the computer laboratory by the student

**Mr. Harsh Shekhar Ghosalkar**

**Seat No.112042**

**Roll No.22** of the Class **- MSc computer Science**

**First Year Sem 2**

Semester during the year **2021-2022** in a Satisfactory

manner.

|  |  |  |
| --- | --- | --- |
| **Course**  **Code** | **Course Title** | **Page No.** |
| **PSCSP203** | **Practical Course on Web Mining** |  |
| **1** | Scrape an online E-Commerce Site for Data.  1. Extract product data from Amazon - be it any product and  put these details in the MySQL database. One can use  pipeline. Like 1 pipeline to process the scraped data and  other to put data in the database and since Amazon has  some restrictions on scraping of data, ask them to work on  small set of requests otherwise proxies and all would have  to be used.  2. Scrape the details like color, dimensions, material etc. Or  customer ratings by features. | 3 - 9 |
| **2** | Page Rank for link analysis using python  Create a small set of pages namely page1, page2, page3 and  page4 apply random walk on the same | 10 – 12 |
| **3** | Perform Spam Classifier. | 13 - 18 |
| **4** | Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online). | 19 – 22 |
| **5** | Apriori Algorithm implementation in case study. | 23 – 25 |
| **6** | Develop a basic crawler for the web search for user defined keywords. | 26 - 29 |
| **7** | Develop a focused crawler for local search. | 30 – 32 |
| **8** | Sentiment analysis for reviews by customers and visualize the same. | 33 - 36 |

**Practical 1**

**AIM :-** Scrape an online E-Commerce Site for Data.

1. Extract product data from Amazon - be it any product and

put these details in the MySQL database. One can use

pipeline. Like 1 pipeline to process the scraped data and

other to put data in the database and since Amazon has

some restrictions on scraping of data, ask them to work on

small set of requests otherwise proxies and all would have

to be used.

2. Scrape the details like color, dimensions, material etc. Or

customer ratings by features.

**THEORY : -**

Web scraping is the process of collecting structured web data in an automated fashion. It’s also called web data extraction. Some of the main use cases of web scraping include price monitoring, price intelligence, news monitoring, lead generation, and market research among many others.

**CODE / OUTPUT** **: -**

pip install kora -q

import csv

from bs4 import BeautifulSoup

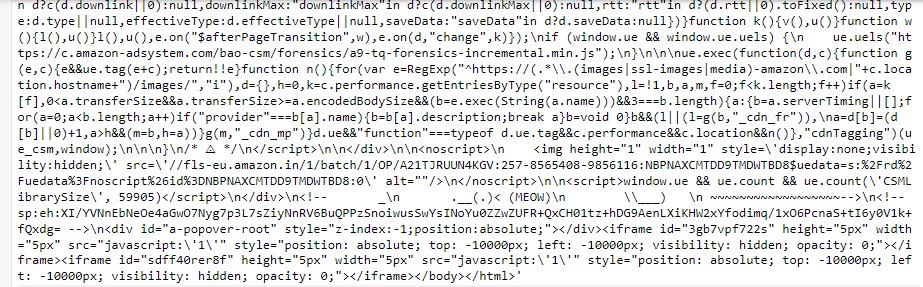
from kora.selenium import wd

wd.get('https://www.amazon.in/')

wd.page\_source







from selenium import webdriver

options = webdriver.ChromeOptions()

options.add\_argument('-headless')

options.add\_argument('-no-sandbox')

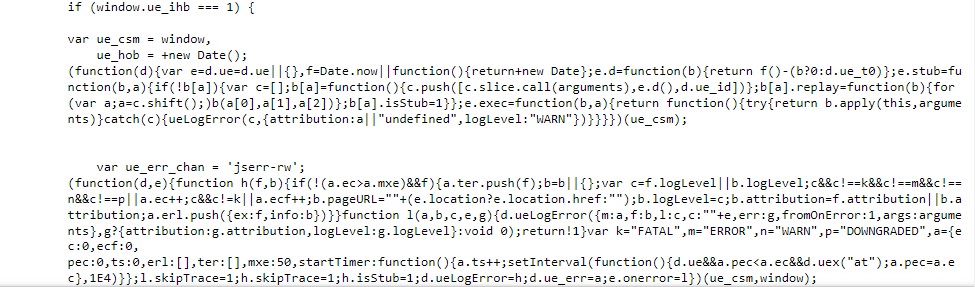
options.add\_argument('-disable-dev-shm-usage')

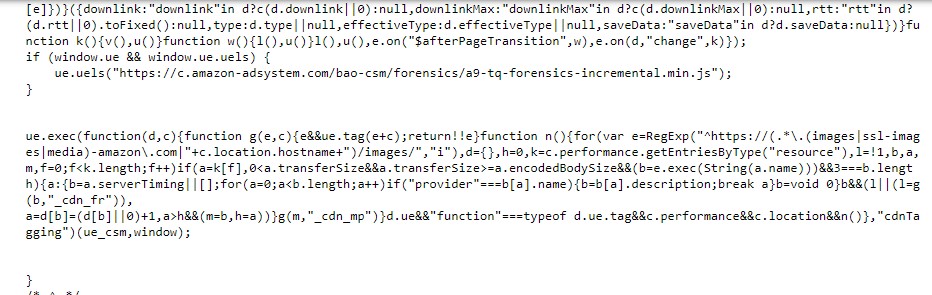
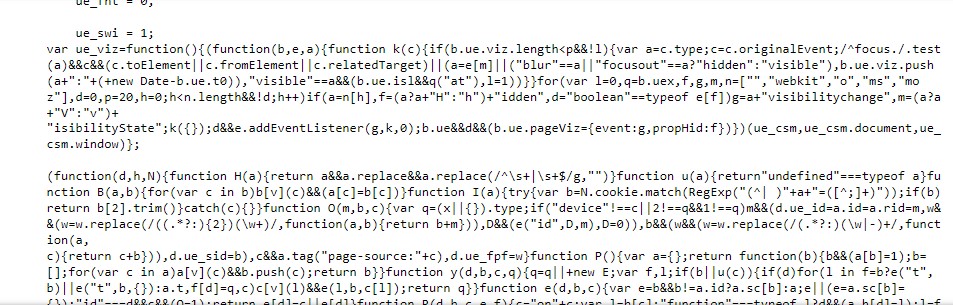
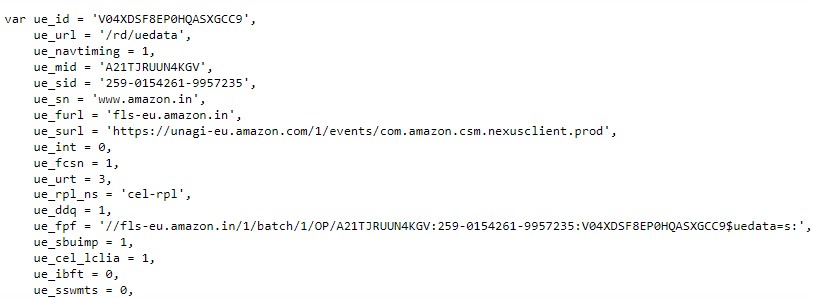
wd = webdriver.Chrome('chromedriver',options=options)

wd.get("https://www.amazon.in/")

print(wd.page\_source) # results







def get\_url(search\_term):

template = "https://www.amazon.in/s?k={}&rh=n%3A1389401031&ref=nb\_sb\_noss"

search\_term = search\_term.replace(' ','+')

return template.format(search\_term)

url = get\_url('laptops')

print(url)



wd.get(url)

soup = BeautifulSoup(wd.page\_source, 'html.parser')

result = soup.find\_all('div',{'data-component-type':'s-search-result'})

len(result)

print(result[2])



item = result[2]

atag = item.h2.a

atag.text



price\_parent = item.find('span','a-price')

price\_parent.find('span','a-offscreen').text



rating = item.i.text

print(rating)



review\_count = item.find('span', {'class':'a-size-base','dir':'auto'})

print(review\_count)



def extract\_record(item1):

atag = item1.h2.a

description = atag.text.strip()

url = "https://www.amazon.in/" + atag.get('href')

price\_parent = item1.find('span','a-price')

#price\_parent.find('span','a-offscreen').text

rating = ""

result = (description, price\_parent, rating)

return result

url = get\_url('mouse')

wd.get(url)

soup = BeautifulSoup(wd.page\_source, 'html.parser')

records = []

results = soup.find\_all('div',{'data-component-type':'s-search-result'})

for item in results:

records.append(extract\_record(item))

records[0]

print("printing records")

for x in range(len(records)):

print(records[x])



**Practical 2**

**AIM :-** Page Rank for link analysis using python

Create a small set of pages namely page1, page2, page3 and

page4 apply random walk on the same.

**THEORY : -**

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

**CODE / OUTPUT** **: -**

"""Random\_Walk\_Page\_Rank"""

"Practical 3"

import networkx as nx

import random

import numpy as np

# Add directed edges in graph

def add\_edges(g, pr):

for each in g.nodes():

for each1 in g.nodes():

if (each != each1):

ra = random.random()

if (ra < pr):

g.add\_edge(each, each1)

else:

continue

return g

# Sort the nodes

def nodes\_sorted(g, points):

t = np.array(points)

t = np.argsort(-t)

return t

# Distribute points randomly in a graph

def random\_Walk(g):

rwp = [0 for i in range(g.number\_of\_nodes())]

nodes = list(g.nodes())

r = random.choice(nodes)

rwp[r] += 1

neigh = list(g.out\_edges(r))

z = 0

while (z != 10000):

if (len(neigh) == 0):

focus = random.choice(nodes)

else:

r1 = random.choice(neigh)

focus = r1[1]

rwp[focus] += 1

neigh = list(g.out\_edges(focus))

z += 1

return rwp

g = nx.DiGraph()

N = 4

g.add\_nodes\_from(range(N))

# 2. Add directed edges in graph

g = add\_edges(g, 0.4)

# 3. perform a random walk

points = random\_Walk(g)

# 4. Get nodes rank according to their random walk points

sorted\_by\_points = nodes\_sorted(g, points)

print("PageRank using Random Walk Method")

print(sorted\_by\_points)

# p\_dict is dictionary of tuples

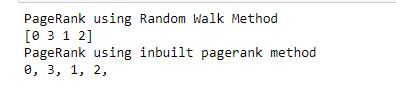
p\_dict = nx.pagerank(g)

p\_sort = sorted(p\_dict.items(), key=lambda x: x[1], reverse=True)

print("PageRank using inbuilt pagerank method")

for i in p\_sort:

print(i[0], end=", ")



**Practical 3**

**AIM :-** Perform Spam Classifier

**THEORY : -**

A spam filter/classifier is a program used to detect unsolicited, unwanted and virus-infected emails and prevent those messages from getting to a user's inbox. Like other types of filtering programs, a spam filter looks for specific criteria on which to base its judgments.

**CODE / OUTPUT** **: -**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from google.colab import drive

from sklearn import feature\_extraction, naive\_bayes, metrics, model\_selection

from sklearn.ensemble import RandomForestClassifier

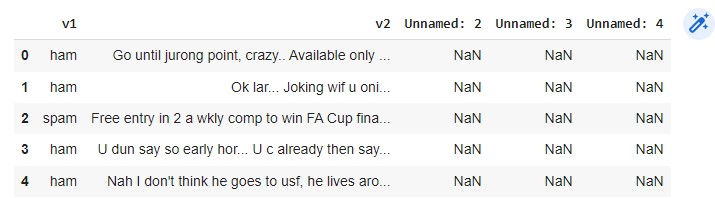
from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_recall\_fscore\_support as score

drive.mount('/content/drive')

dataset =pd.read\_csv("/content/drive/MyDrive/spam.csv", encoding='latin-1')

dataset.head()



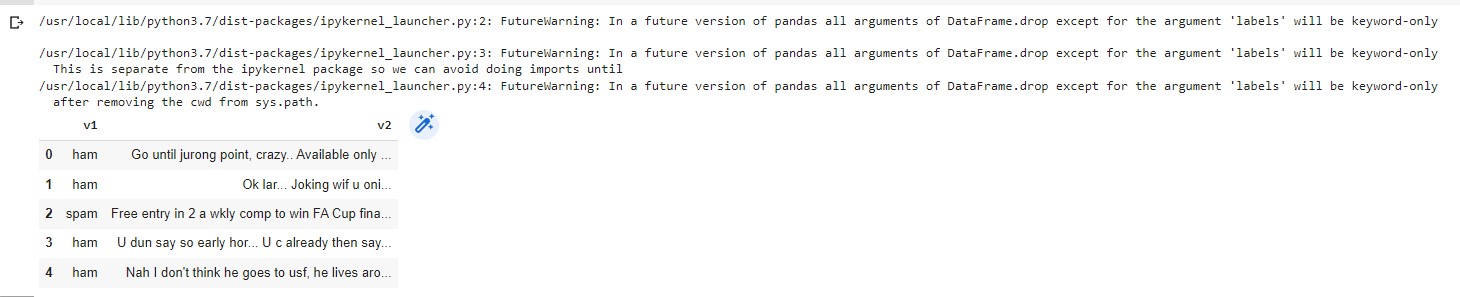
#removing unnamed columns

dataset = dataset.drop('Unnamed: 2', 1)

dataset = dataset.drop('Unnamed: 3', 1)

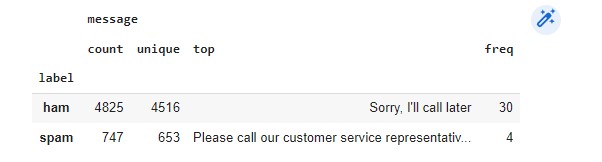
dataset = dataset.drop('Unnamed: 4', 1)

dataset.head()

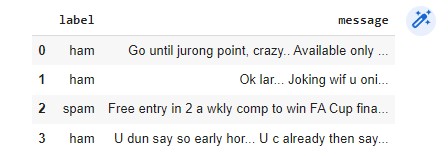


dataset = dataset.rename(columns = {'v1':'label','v2':'message'})

dataset.groupby('label').describe()



dataset.head(4)

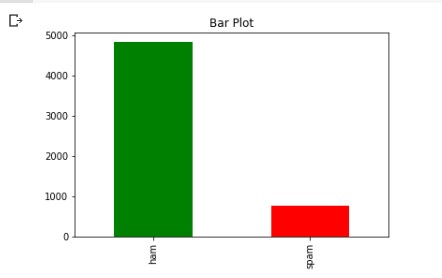


count\_Class=pd.value\_counts(dataset["label"], sort= True)

count\_Class.plot(kind = 'bar',color = ["green","red"])

plt.title('Bar Plot')

plt.show();



f = feature\_extraction.text.CountVectorizer(stop\_words = 'english')

X = f.fit\_transform(dataset["message"])

np.shape(X)



# Classifying spam and not spam msgs as 1 and 0

dataset["label"]=dataset["label"].map({'spam':1,'ham':0})

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, dataset['label'], test\_size=0.70, random\_state=42)

list\_alpha = np.arange(1/100000, 20, 0.11)

score\_train = np.zeros(len(list\_alpha))

score\_test = np.zeros(len(list\_alpha))

recall\_test = np.zeros(len(list\_alpha))

precision\_test= np.zeros(len(list\_alpha))

count = 0

for alpha in list\_alpha:

bayes = naive\_bayes.MultinomialNB(alpha=alpha)

bayes.fit(X\_train, y\_train)

score\_train[count] = bayes.score(X\_train, y\_train)

score\_test[count]= bayes.score(X\_test, y\_test)

recall\_test[count] = metrics.recall\_score(y\_test, bayes.predict(X\_test))

precision\_test[count] = metrics.precision\_score(y\_test, bayes.predict(X\_test))

count = count + 1

matrix = np.matrix(np.c\_[list\_alpha, score\_train, score\_test, recall\_test, precision\_test])

models = pd.DataFrame(data = matrix, columns =

['alpha', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precision'])

models.head(n=10)

best\_index = models['Test Precision'].idxmax()

models.iloc[best\_index, :]

rf = RandomForestClassifier(n\_estimators=100,max\_depth=None,n\_jobs=-1)

rf\_model = rf.fit(X\_train,y\_train)

y\_pred=rf\_model.predict(X\_test)

precision,recall,fscore,support =score(y\_test,y\_pred,pos\_label=1, average ='binary')

print('Precision : {} / Recall : {} / fscore : {} / Accuracy: {}'.format(round(precision,3),round(recall,3),round(fscore,3),round((y\_pred==y\_test).sum()/len(y\_test),3)))



import tensorflow as tf

from keras.preprocessing.text import Tokenizer

from keras.layers import Embedding, LSTM, Dropout, Dense

from keras.models import Sequential

from tensorflow.keras.utils import to\_categorical

#from keras.utils import to\_categorical

from keras.preprocessing.sequence import pad\_sequences

import tensorflow as tf

vocab\_size = 400

oov\_tok = "<OOV>"

max\_length = 250

embedding\_dim = 16

encode = ({'ham': 0, 'spam': 1} )

#new dataset with replaced values

dataset = dataset.replace(encode)

X = dataset['message']

Y = dataset['label']

tokenizer = Tokenizer(num\_words=vocab\_size, oov\_token=oov\_tok)

tokenizer.fit\_on\_texts(X)

# convert to sequence of integers

X = tokenizer.texts\_to\_sequences(X)

X = np.array(X)

y = np.array(Y)

X = pad\_sequences(X, maxlen=max\_length)

model = tf.keras.Sequential([

tf.keras.layers.Embedding(vocab\_size, embedding\_dim, input\_length=max\_length),

tf.keras.layers.GlobalAveragePooling1D(),

tf.keras.layers.Dense(24, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

model.summary()

num\_epochs = 50

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=.20, random\_state=7)

history = model.fit(X\_train, y\_train, epochs=num\_epochs, validation\_data=(X\_test,y\_test), verbose=2)

results = model.evaluate(X\_test, y\_test)

loss = results[0]

accuracy = results[1]

print(f"[+] Accuracy: {accuracy\*100:.2f}%")

from keras.preprocessing import sequence

#Defining the function

def get\_predictions(txts):

txts = tokenizer.texts\_to\_sequences(txts)

txts = sequence.pad\_sequences(txts, maxlen=max\_length)

preds = model.predict(txts)

if(preds[0] > 0.5):

print("SPAM MESSAGE")

else:

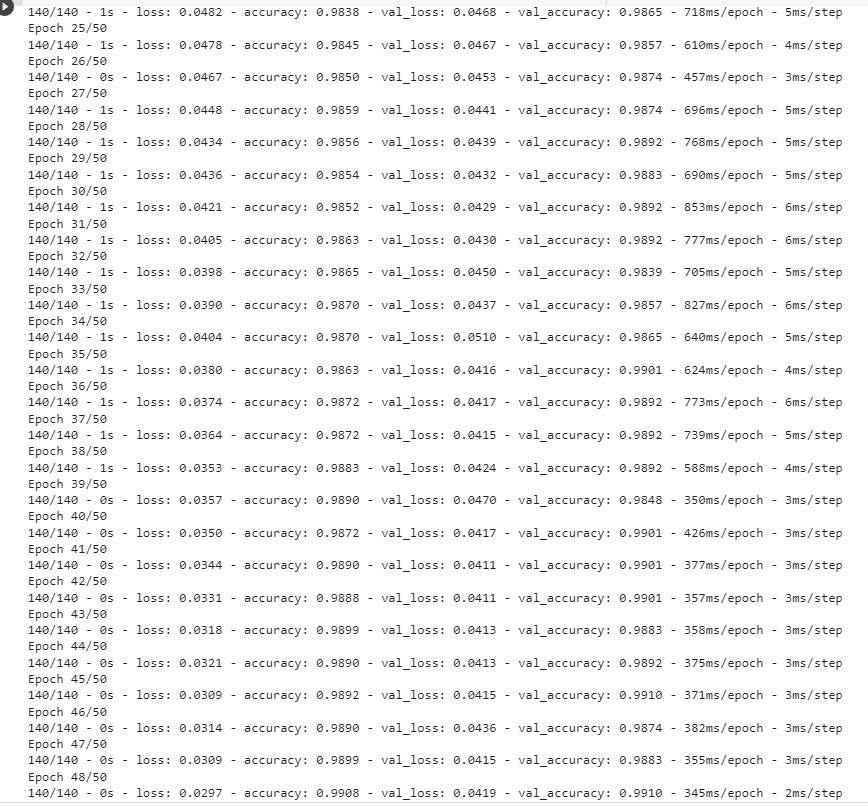
print('NOT SPAM')

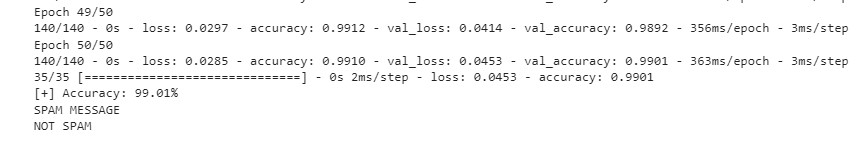
txts=["You have won a free ticket to las vegas. Contact now"]

get\_predictions(txts)

txts=["Hey there call me asap!!"]

get\_predictions(txts)





**Practical 4**

**AIM :-** Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online).

**THEORY : -**

Web mining is the application of data mining techniques to discover patterns from the World Wide Web. It uses automated methods to extract both structured and unstructured data from web pages, server logs and link structures. There are three main sub-categories of web mining.

Preprocessing stage helps to clean the records and determine the interesting user patterns and session creation. Data preprocessing is an important job of Web usage mining application. So, data must be processed before applying data mining methods to determine user access patterns from web log.

A meta description is an HTML element that provides a brief summary of a web page. A page's meta description tag is displayed as part of the search snippet in a search engine results page (SERP) and is meant to give the user an idea of the content that exists within the page and how it relates to their search query.

**CODE / OUTPUT** **: -**

"""Pract5.ipynb"""

# Commented out IPython magic to ensure Python compatibility.

# Imports

import requests

import numpy as np

import pandas as pd

from bs4 import BeautifulSoup

import matplotlib.pyplot as plt

# IMDB's homepage

imdb\_url = 'https://www.imdb.com'

# Use requests to retrieve data from a given URL

imdb\_response = requests.get(imdb\_url)

# Parse the whole HTML page using BeautifulSoup

imdb\_soup = BeautifulSoup(imdb\_response.text, 'html.parser')

# Title of the parsed page

imdb\_soup.title.text



# Find all links

links = [link.get('href') for link in imdb\_soup.find\_all('a')]

# Add homepage and keep the unique links

unique\_links = []

for link in links:

if not link in unique\_links:

unique\_links.append(imdb\_url + link)

# Box Office Mojo - UK Weekend box office

boxofficemojo\_url = 'https://www.boxofficemojo.com/intl/uk/?yr=2019&wk=33&currency=local'

# Use requests to retrieve data from a given URL

bom\_response = requests.get(boxofficemojo\_url)

# Parse the whole HTML page using BeautifulSoup

bom\_soup = BeautifulSoup(bom\_response.text, 'html.parser')

print(f"NUMBER OF TABLES IN THE PAGE: {len(bom\_soup.find\_all('table'))}")

table = bom\_soup.find\_all('table')[0]

table

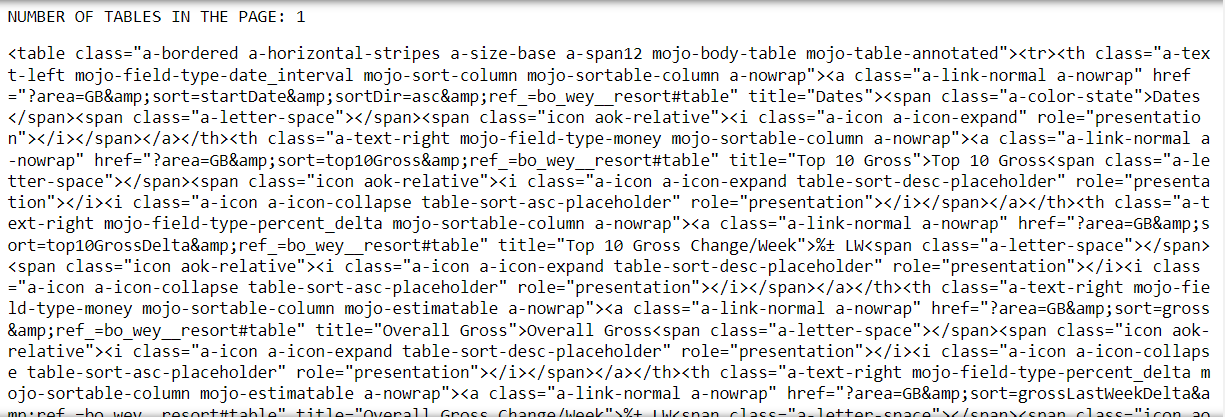


table.find\_all('tr')[0].contents

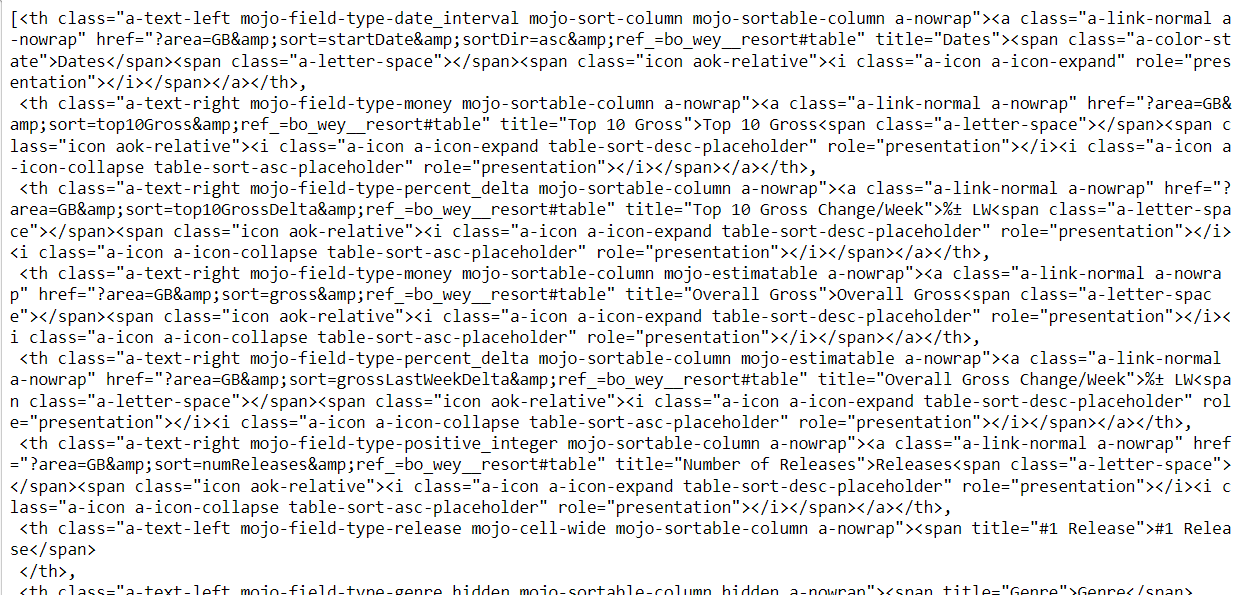
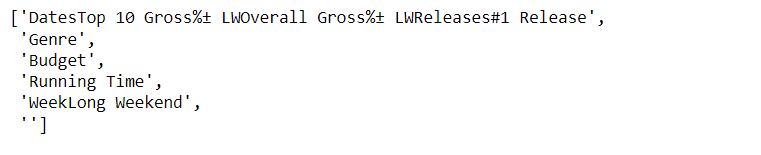


table.find\_all('tr')[0].text.split('\n')



lst = []

for row in table.find\_all('tr')[1:-1]:

s = pd.Series([data.text for data in row.find\_all('td')])

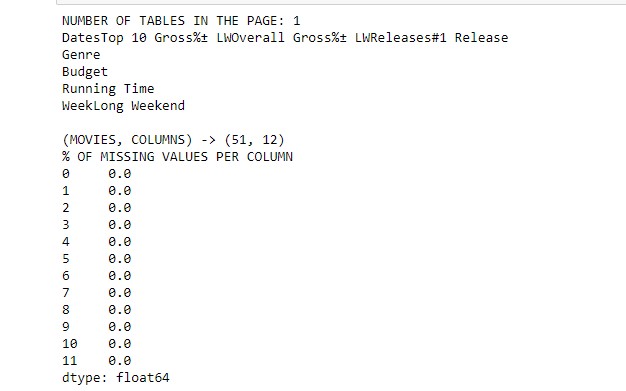
lst.append(s)

data = pd.concat(lst, axis=1).T

data.head(2)

print(f'(MOVIES, COLUMNS) -> {data.shape}')

print(f'% OF MISSING VALUES PER COLUMN\n{(data.isnull().sum() / data.shape[0]) \* 100}')

****

**Practical 5**

**AIM :-** Apriori Algorithm implementation in case study.

**THEORY : -**

Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.

**CODE / OUTPUT** **: -**

**Minimum Support = 50 %**

**Minimum Confidence = 70 %**

**Itemset ={ Bread, Chicken, Butter, Milk, Toast}**

|  |  |
| --- | --- |
| **Transaction ID** | **Items** |
| 100 | {Bread, Butter, Milk} |
| 200 | {Chicken, Butter, Toast} |
| 300 | {Bread, Chicken, Butter, Toast} |
| 400 | {Chicken, Toast} |

|  |  |
| --- | --- |
| **Item** | **Support** |
| **Bread** | **2 /4    = 0.5   = 50%** |
| **Chicken** | **3 /4   = 0.5   = 75%** |
| **Butter** | **3 /4    = 0.75   = 75%** |
| **Milk** | **1/4      = 0.25    = 25%** |
| **Toast** | **3/4     = 0.75   = 75%** |

**Itemset = { Bread, Chicken , Butter, Toast}**

|  |  |
| --- | --- |
| **Item** | **Support** |
| **{Bread, Chicken}** | **¼ = 0.25   =25%** |
| **{Bread, Butter}** | **2/4 =0.50   = 50%** |
| **{Bread, Toast}** | **¼  = 0.25   = 25%** |
| **{Chicken, Butter}** | **2/4  = 0.50  = 50 %** |
| **{Chicken, Toast}** | **¾ = 0.75 = 75%** |
| **{Butter, Toast}** | **2/4 = 0.50 = 50%** |

**Itemset = ({Bread, Butter}, {Chicken, Butter} , {Chicken, Toast}, {Butter, Toast})**

|  |  |
| --- | --- |
| **Item** | **Support** |
| {Bread, Butter, Toast} | 1/ 4   = 0.25   = 25% |
| {Chicken, Butter, Toast} | 2/4   =0.50    = 50 % |
| {Bread, Butter, Chicken} | ¼  = 0.25   = 25% |

**Minimum Support = 50%**

**Minimum Confidence = 70%**

Final Resultant Set based on Support = {Chicken, Butter, Toast}

Rules

1 .     (Chicken   &  Butter )   - > Toast      2   (50%)

2.       (Butter & Toast) -> Chicken 2   (50%)

3.       (Chicken & Toast) -> Butter   2   (50%)

4.      Chicken   - > (Butter & Toast)   2   (50%)

5.      Toast -> (Chicken & Butter)   2   (50%)

6.       Butter -> (Chicken & Toast)   2   (50%)

Confidence = S(A U B).count / S(A).count

**1 .     (Chicken   &  Butter )   - > Toast      2   (50%)**

S((Chicken &Butter) U (Toast))/ S(Chicken & Butter)

=2 /  2    = 1   = **100%**

**2.  .       (Butter & Toast) -> Chicken**

Confidence = S(A U B).count / S(A).count

S((Butter & Toast) U Chicken)) /S(Butter & Toast)

=2 /  2   = 1    = **100%**

**3.       (Chicken & Toast) -> Butter   2   (50%)**

Confidence = S(A U B).count / S(A).count

S((Chicken & Toast) U (Butter))/S(Chicken & Toast)

=2/3    = 0.666 **= 67%**

**4.      Chicken   - > (Butter & Toast)   2   (50%)**

Confidence = S(A U B).count / S(A).count

S((Chicken) U (Butter & Toast))/S(Chicken)

=2/3 = 0.666 = **67%**

**Minimum Support = 50%**

**Minimum Confidence = 70%**

**5.      Toast -> (Chicken & Butter)   2   (50%)**

Confidence = S(A U B).count / S(A).count

S((Toast) U (Chicken & Butter))/S(Toast)

=2/3 = 0.666 = **67%**

**6.       Butter -> (Chicken & Toast)   2   (50%)**

Confidence = S(A U B).count / S(A).count

S((Butter) U (Chicken & Toast))/S(Butter)

=2/3 = 0.666 = **67%**

**Final Associated Items rules are**

1 .     (Chicken   &  Butter )   - > Toast      2   (50%)

2.       (Butter & Toast) -> Chicken 2   (50%)

**Practical 6**

**AIM :-** Develop a basic crawler for the web search for user defined keywords.

**THEORY : -**

A Web crawler, sometimes called a spider or spiderbot and often shortened to crawler, is an Internet bot that systematically browses the World Wide Web and that is typically operated by search engines for the purpose of Web indexing (web spidering).

**CODE / OUTPUT** **: -**

import requests

url = 'https://en.wikipedia.org/wiki/Stock\_market'

response = requests.get(url, timeout=3) #timeout set to stop the request action in case of hanging

print('Status code: ',response.status\_code)

if response.status\_code==200:

print('Connection successfull.\n\n')

else:

print('Error. Check status code table.\n\n')



print(f"{'---'\*20}\n\tContents of Response.items():\n{'---'\*20}")

for k,v in response.headers.items():

print(f"{k:{25}}: {v:{40}}") # Note: add :{number} inside of a

print(f"{'---'\*20}\nStatus code: {response.status\_code}\n{'---'\*20}\n")

from bs4 import BeautifulSoup

# Feed the response's .content into BeauitfulSoup

page\_content = response.content

soup = BeautifulSoup(page\_content,'lxml') #'html.parser')

# Preview soup contents using .prettify()

print(soup.prettify()[:2000])

body = soup.body

for child in body.children:

# print child if its not empty

print(child if child is not None else ' ', '\n\n') # '\n\n' for visual separation

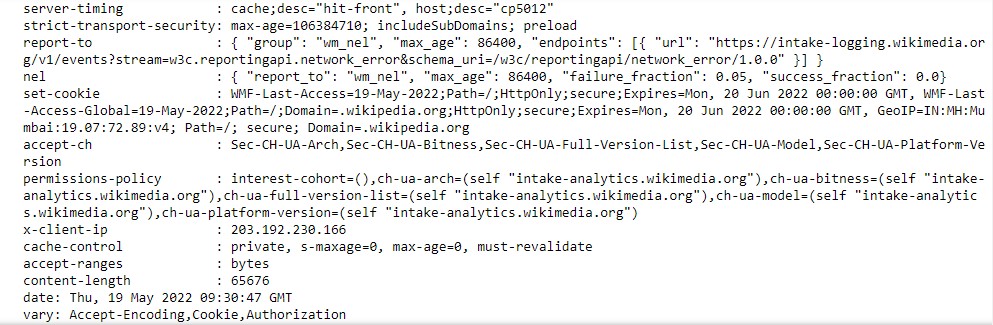
title = soup.head.title

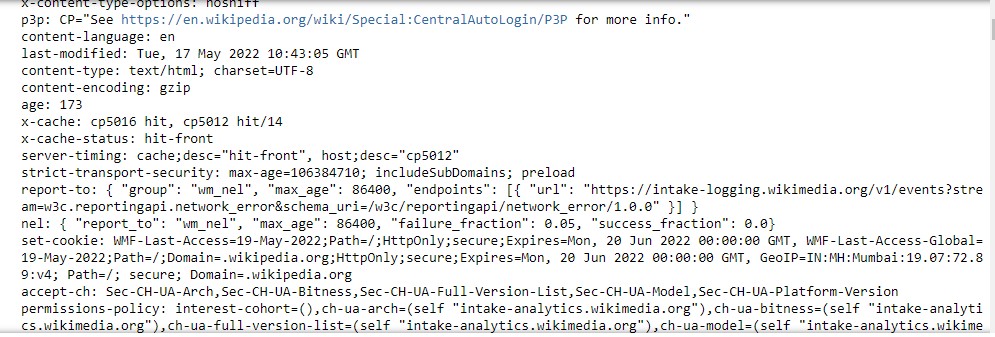
print(title.parent.name)

results = soup.find\_all()

results

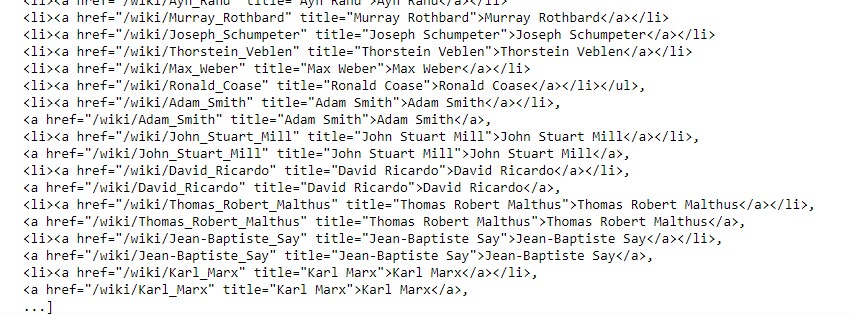












**Practical 7**

**AIM :-** Develop a focused crawler for local search .

**THEORY : -**

A focused crawler is a web crawler that collects Web pages that satisfy some specific property, by carefully prioritizing the crawl frontier and managing the hyperlink exploration process. Some predicates may be based on simple, deterministic and surface properties. For example, a crawler's mission may be to crawl pages from only the .jp domain. Other predicates may be softer or comparative, e.g., "crawl pages about baseball", or "crawl pages with large PageRank". An important page property pertains to topics, leading to 'topical crawlers'. For example, a topical crawler may be deployed to collect pages about solar power, swine flu, or even more abstract concepts like controversy while minimizing resources spent fetching pages on other topics.

**CODE / OUTPUT** **: -**

import requests

import lxml

from bs4 import BeautifulSoup

url = "https://www.rottentomatoes.com/top/bestofrt/"

f = requests.get(url)

soup = BeautifulSoup(f.content,'html')

#parse in normal html

movies = soup.find\_all('div',{'class':'discovery-tiles\_\_wrap'})[1].find\_all\_next('a')

movies\_list = []

num = 0

for movie in movies:

try:

url = 'https://www.rottentomatoes.com' + movie["href"]

movie\_f = requests.get(url)

movie\_page = BeautifulSoup(movie\_f.content,'html')

num += 1

title = movie\_page.find('h1', {'class': 'scoreboard\_\_title'})

movie\_content = movie\_page.find('div', {'class': 'movie\_synopsis'})

movies\_list.append({

"#": num,

"url": url,

"title": title.text,

"content": movie\_content.getText()

})

except:

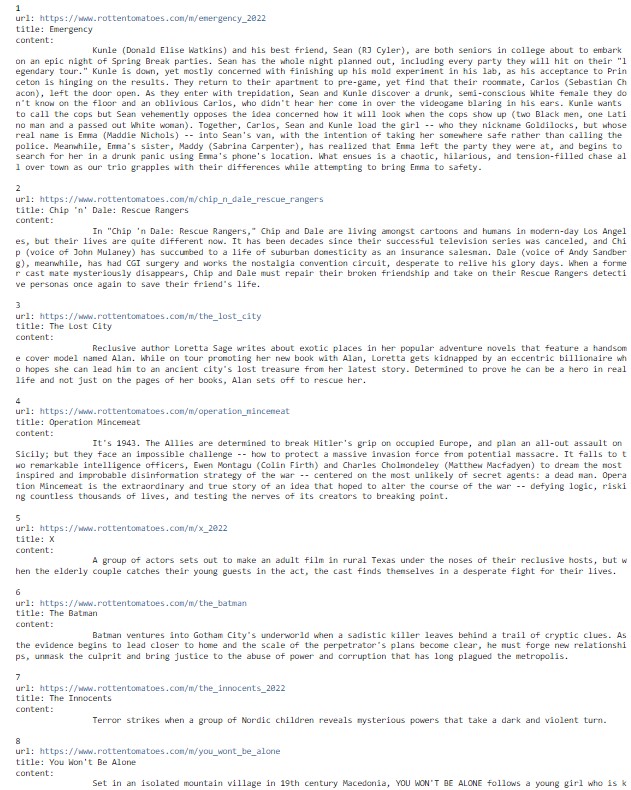
continue

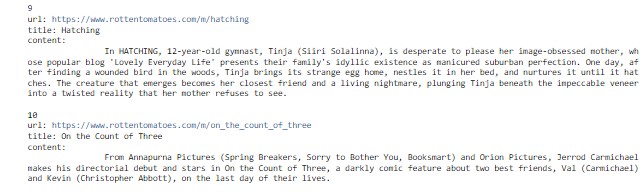
len(movies\_list)



for movie in movies\_list[:10]:

print(f"{movie['#']} \nurl: {movie['url']} \ntitle: {movie['title']} \ncontent: {movie['content']}")





**Practical 8**

**AIM :-** Sentiment analysis for reviews by customers and visualize the same.

**THEORY : -**

Sentiment analysis (also known as opinion mining or emotion AI) is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine. With the rise of deep language models, such as RoBERTa, also more difficult data domains can be analyzed, e.g., news texts where authors typically express their opinion/sentiment less explicitly.

**CODE / OUTPUT** **: -**

"""Pract10\_WM.ipynb"""

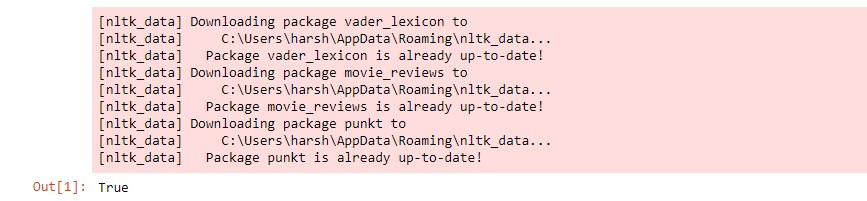
!pip install matplotlib pandas nltk textblob

import nltk

nltk.download('vader\_lexicon')

nltk.download('movie\_reviews')

nltk.download('punkt')



from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

sia = SIA()

sia.polarity\_scores("This restaurant was great, but I'm not sure if I'll go there again.")



text = "I just got a call from my boss - does he realise it's Saturday?"

sia.polarity\_scores(text)



text = "I just got a call from my boss - does he realise it's Saturday? :)"

sia.polarity\_scores(text)



text = "I just got a call from my boss - does he realise it's Saturday? 😊"

sia.polarity\_scores(text)



from textblob import TextBlob

from textblob import Blobber

from textblob.sentiments import NaiveBayesAnalyzer

blob = TextBlob("This restaurant was great, but I'm not sure if I'll go there again.")

blob.sentiment



blobber = Blobber(analyzer=NaiveBayesAnalyzer())

blob = blobber("This restaurant was great, but I'm not sure if I'll go there again.")

blob.sentiment



import pandas as pd

df = pd.DataFrame({'content': [

"I love love love love this kitten",

"I hate hate hate hate this keyboard",

"I'm not sure how I feel about toast",

"Did you see the baseball game yesterday?",

"The package was delivered late and the contents were broken",

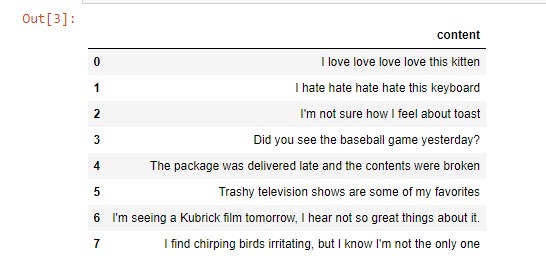
"Trashy television shows are some of my favorites",

"I'm seeing a Kubrick film tomorrow, I hear not so great things about it.",

"I find chirping birds irritating, but I know I'm not the only one",

]})

df



def get\_scores(content):

blob = TextBlob(content)

nb\_blob = blobber(content)

sia\_scores = sia.polarity\_scores(content)

return pd.Series({

'content': content,

'textblob': blob.sentiment.polarity,

'textblob\_bayes': nb\_blob.sentiment.p\_pos - nb\_blob.sentiment.p\_neg,

'nltk': sia\_scores['compound'],

})

scores = df.content.apply(get\_scores)

scores.style.background\_gradient(cmap='RdYlGn', axis=None, low=0.4, high=0.4)

