© Akshay Kulkarni and Adarsha Shivananda 2019
Akshay Kulkarni and Adarsha Shivananda, *Natural Language Processing Recipes*
https://doi.org/10.1007/978-1-4842-4267-4_1

1. Extracting the Data

Akshay Kulkarni[1] and Adarsha Shivananda[1]

(1) Bangalore, Karnataka, India

In this chapter, we are going to cover various sources of text data and ways to extract it, which can act as information or insights for businesses.

- Recipe 1. Text data collection using APIs
- Recipe 2. Reading PDF file in Python
- Recipe 3. Reading word document
- Recipe 4. Reading JSON object
- Recipe 5. Reading HTML page and HTML parsing
- Recipe 6. Regular expressions
- Recipe 7. String handling
- Recipe 8. Web scraping

Introduction

Before getting into details of the book, let's see the different possible data sources available in general. We need to identify potential data sources for a business's benefit.

CLIENT DATA For any problem statement, one of the sources is their own data that is already present. But it depends on the business where they store it. Data storage depends on the type of business, amount of data, and cost associated with different sources.

- SQL databases
- Hadoop clusters
- Cloud storage
- Flat files

FREE SOURCE A huge amount of data is freely available over the internet. We just need to streamline the problem and start exploring multiple free data sources.

- Free APIs like Twitter
- Wikipedia
- Government data (e.g. `http://data.gov` (http://data.gov))

- Census data (e.g. http://www.census.gov/data.html (http://www.census.gov/data.html))
- Health care claim data (e.g. https://www.healthdata.gov/

)

WEB SCRAPING Extracting the content/data from websites, blogs, forums, and retail websites for reviews with the permission from the respective sources using web scraping packages in Python.

There are a lot of other sources like crime data, accident data, and economic data that can also be leveraged for analysis based on the problem statement.

Recipe 1-1. Collecting Data

As discussed, there are a lot of free APIs through which we can collect data and use it to solve problems. We will discuss the Twitter API in particular (it can be used in other scenarios as well).

PROBLEM

You want to collect text data using Twitter APIs.

SOLUTION

Twitter has a gigantic amount of data with a lot of value in it. Social media marketers are making their living from it. There is an enormous amount of tweets every day, and every tweet has some story to tell. When all of this data is collected and analyzed, it gives a tremendous amount of insights to a business about their company, product, service, etc.

Let's see how to pull the data in this recipe and then explore how to leverage it in coming chapters.

HOW IT WORKS

Step 1-1 Log in to the Twitter developer portal

Create your own app in the Twitter developer portal, and get the keys mentioned below. Once you have these credentials, you can start pulling data. Keys needed:

- consumer key: Key associated with the application (Twitter, Facebook, etc.).
- consumer secret: Password used to authenticate with the authentication server (Twitter, Facebook, etc.).
- access token: Key given to the client after successful authentication of above keys.
- access token secret: Password for the access key.

Step 1-2 Execute below query in Python

Once all the credentials are in place, use the code below to fetch the data.

```
# Install tweepy
!pip install tweepy

# Import the libraries

import numpy as np
import tweepy
import json
import pandas as pd
from tweepy import OAuthHandler

# credentials

consumer_key = "adjbiejfaaoeh"
consumer_secret = "had73haf78af"
access_token = "jnsfby5u4yuawhafjeh"
access_token_secret = "jhdfgay768476r"

# calling API

auth = tweepy.OAuthHandler(consumer_key,
consumer_secret)
```

```
auth.set_access_token(access_token,
access_token_secret)
api = tweepy.API(auth)

# Provide the query you want to pull the data. For
example, pulling data for the mobile phone ABC

query ="ABC"

# Fetching tweets

Tweets = api.search(query, count =
10,lang='en',exclude='retweets',tweet_mode='extended')
```

The query above will pull the top 10 tweets when the product ABC is searched. The API will pull English tweets since the language given is 'en' and it will exclude retweets.

Recipe 1-2. Collecting Data from PDFs

Most of the time your data will be stored as PDF files. We need to extract text from these files and store it for further analysis.

PROBLEM

You want to read a PDF file.

SOLUTION

The simplest way to do this is by using the PyPDF2 library.

HOW IT WORKS

Let's follow the steps in this section to extract data from PDF files.

Step 2-1 Install and import all the necessary libraries

Here are the first lines of code:

```
!pip install PyPDF2
import PyPDF2
from PyPDF2 import PdfFileReader
```

NOTE You can download any PDF file from the web and place it in the location where you are running this Jupyter notebook or Python script.

Step 2-2 Extracting text from PDF file

Now we extract the text.

```
#Creating a pdf file object

pdf = open("file.pdf","rb")

#creating pdf reader object

pdf_reader = PyPDF2.PdfFileReader(pdf)

#checking number of pages in a pdf file

print(pdf_reader.numPages)

#creating a page object

page = pdf_reader.getPage(0)

#finally extracting text from the page

print(page.extractText())

#closing the pdf file

pdf.close()
```

Please note that the function above doesn't work for scanned PDFs.

Recipe 1-3. Collecting Data from Word Files

Next, let us look at another small recipe by reading **Word** files in Python.

PROBLEM

You want to read Word files .

SOLUTION

The simplest way to do this is by using the docx library.

HOW IT WORKS

Let's follow the steps in this section to extract data from the Word file.

Step 3-1 Install and import all the necessary libraries

Here are the first lines of code:

```
#Install docx
!pip install docx
```

```
#Import library
from docx import Document
```

NOTE You can download any Word file from the web and place it in the location where you are running this Jupyter notebook or Python script.

Step 3-2 Extracting text from word file

Now we get the text:

```
#Creating a word file object

doc = open("file.docx","rb")

#creating word reader object

document = docx.Document(doc)

# create an empty string and call this document. This
document variable store each paragraph in the Word
document.We then create a for loop that goes through
each paragraph in the Word document and appends the
paragraph.

docu=""
for para in document.paragraphs:
       docu += para.text

#to see the output call docu
print(docu)
```

Recipe 1-4. Collecting Data from JSON

Reading a JSON file/object.

PROBLEM

You want to read a JSON file/object.

SOLUTION

The simplest way to do this is by using requests and the JSON library.

HOW IT WORKS

Let's follow the steps in this section to extract data from the JSON.

Step 4-1 Install and import all the necessary libraries

Here is the code for importing the libraries.

```
import requests
import json
```

Step 4-2 Extracting text from JSON file

Now we extract the text.

```
#json from "https://quotes.rest/qod.json"
r = requests.get("https://quotes.rest/qod.json")
res = r.json()
print(json.dumps(res, indent = 4))
```

```
#output
{
    "success": {
        "total": 1
    },
    "contents": {
        "quotes": [
            {
                "quote": "Where there is ruin, there
is hope for a treasure.",
                "length": "50",
                "author": "Rumi",
                "tags": [
                    "failure",
                    "inspire",
                    "learning-from-failure"
                ],
                "category": "inspire",
                "date": "2018-09-29",
                "permalink":
"https://theysaidso.com/quote/dPKsui4sQnQqgMnXHLKtfweF
/rumi-where-there-is-ruin-there-is-hope-for-a-
treasure",
                "title": "Inspiring Quote of the day",
                "background":
"https://theysaidso.com/img/bgs/man_on_the_mountain.jp
g",
                "id": "dPKsui4sQnQqgMnXHLKtfweF"
            }
        ],
        "copyright": "2017-19 theysaidso.com"
    }
}
```

```
#extract contents
q = res['contents']['quotes'][0]
q
```

```
#output
```

```
{'author': 'Rumi',
 'background':
'https://theysaidso.com/img/bgs/man_on_the_mountain.jp
g',
 'category': 'inspire',
 'date': '2018-09-29',
 'id': 'dPKsui4sQnQqgMnXHLKtfweF',
 'length': '50',
 'permalink':
'https://theysaidso.com/quote/dPKsui4sQnQqgMnXHLKtfweF
/rumi-where-there-is-ruin-there-is-hope-for-a-
treasure',
 'quote': 'Where there is ruin, there is hope for a
treasure.',
 'tags': ['failure', 'inspire', 'learning-from-
failure'],
 'title': 'Inspiring Quote of the day'}
```

```
#extract only quote
print(q['quote'], '\n--', q['author'])
```

```
#output
It wasn't raining when Noah built the ark....
-- Howard Ruff
```

Recipe 1-5. Collecting Data from HTML

In this recipe, let us look at reading HTML pages.

PROBLEM

You want to read parse/read HTML pages.

SOLUTION

The simplest way to do this is by using the bs4 library.

HOW IT WORKS

Let's follow the steps in this section to extract data from the web.

Step 5-1 Install and import all the necessary libraries

Let's import the libraries:

```
!pip install bs4
import urllib.request as urllib2
from bs4 import BeautifulSoup
```

Step 5-2 Fetch the HTML file

Pick any website from the web that you want to extract. Let's pick Wikipedia for this example.

```
response =
urllib2.urlopen('https://en.wikipedia.org/wiki/Natural
_language_processing')
html_doc = response.read()
```

Step 5-3 Parse the HTML file

Now we get the data:

```
#Parsing
soup = BeautifulSoup(html_doc, 'html.parser')
# Formating the parsed html file
strhtm = soup.prettify()

# Print few lines
print (strhtm[:1000])

#output

<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Natural language processing - Wikipedia
  </title>
  <script>
   document.documentElement.className =
document.documentElement.className.replace(
/(^|\s)client-nojs(\s|$)/, "$1client-js$2" );
  </script>
  <script>
   (window.RLQ=window.RLQ||[]).push(function()
{mw.config.set({"wgCanonicalNamespace":"","wgCanonical
SpecialPageName":false,"wgNamespaceNumber":0,"wgPageNa
me":"Natural_language_processing","wgTitle":"Natural
language
processing","wgCurRevisionId":860741853,"wgRevisionId"
:860741853,"wgArticleId":21652,"wgIsArticle":true,"wgI
sRedirect":false,"wgAction":"view","wgUserName":null,"
wgUserGroups":["*"],"wgCategories":["Webarchive
template wayback links","All accuracy
disputes","Articles with disputed statements from June
2018","Wikipedia articles with NDL
identifiers","Natural language
processing","Computational linguistics","Speech
recognition","Computational fields of stud
```

Step 5-4 Extracting tag value

We can extract a tag value from the first instance of the tag using the following code.

```
print(soup.title)
print(soup.title.string)
print(soup.a.string)
print(soup.b.string)

#output
 <title>Natural language processing -
Wikipedia</title>
Natural language processing - Wikipedia
None
```

```
Natural language processing
```

Step 5-5 Extracting all instances of a particular tag

Here we get all the instances of a tag that we are interested in:

```
for x in soup.find_all('a'): print(x.string)
```

```
#sample output
 None
Jump to navigation
Jump to search
Language processing in the brain
None
None
automated online assistant
customer service
[1]
computer science
artificial intelligence
natural language
speech recognition
natural language understanding
natural language generation
```

Step 5-6 Extracting all text of a particular tag

Finally, we get the text:

```
for x in soup.find_all('p'): print(x.text)
```

```
#sample output
Natural language processing (NLP) is an area of
computer science and artificial intelligence concerned
with the interactions between computers and human
(natural) languages, in particular how to program
computers to process and analyze large amounts of
natural language data.

Challenges in natural language processing frequently
involve speech recognition, natural language
understanding, and natural language generation.

The history of natural language processing generally
started in the 1950s, although work can be found from
earlier periods.
In 1950, Alan Turing published an article titled
"Intelligence" which proposed what is now called the
Turing test as a criterion of intelligence.
```

If you observe here, using the 'p' tag extracted most of the text present in the page.

Recipe 1-6. Parsing Text Using Regular Expressions

In this recipe, we are going to discuss how regular expressions are helpful when dealing with text data. This is very much required when dealing with raw data from the web, which would contain HTML tags, long text, and repeated text. During the process of developing your application, as well as in output, we don't need such data.

We can do all sort of basic and advanced data cleaning using regular expressions.

PROBLEM

You want to parse text data using regular expressions.

SOLUTION

The best way to do this is by using the "re" library in Python.

HOW IT WORKS

Let's look at some of the ways we can use regular expressions for our tasks. Basic flags: the basic flags are I, L, M, S, U, X:

- `re.I`: This flag is used for ignoring casing.

- `re.L`: This flag is used to find a local dependent.
- `re.M`: This flag is useful if you want to find patterns throughout multiple lines.
- `re.S`: This flag is used to find dot matches.
- `re.U`: This flag is used to work for unicode data.
- `re.X`: This flag is used for writing regex in a more readable format.

Regular expressions' functionality:

- Find the single occurrence of character a and b:

  `Regex: [ab]`
- Find characters except for a and b:

  `Regex: [^ab]`
- Find the character range of a to z:

  `Regex: [a-z]`
- Find a range except to z:

  `Regex: [^a-z]`
- Find all the characters a to z as well as A to Z:

  `Regex: [a-zA-Z]`
- Any single character:

  `Regex:`
- Any whitespace character:

  `Regex: \s`
- Any non-whitespace character:

  `Regex: \S`
- Any digit:

  `Regex: \d`
- Any non-digit:

  `Regex: \D`
- Any non-words:

  `Regex: \W`
- Any words:

  `Regex: \w`
- Either match a or b:

  `Regex: (a|b)`
- The occurrence of a is either zero or one:
  - Matches zero or one occurrence but not more than one occurrence

    `Regex: a? ; ?`
  - The occurrence of a is zero times or more than that:

    `Regex: a* ; * matches zero or more than that`
  - The occurrence of a is one time or more than that:

    ```
    Regex: a+ ; + matches occurrences one or more that
    one time
    ```
- Exactly match three occurrences of a:

  `Regex: a{3}`
- Match simultaneous occurrences of a with 3 or more than 3:

  `Regex: a{3,}`
- Match simultaneous occurrences of a between 3 to 6:

  `Regex: a{3,6}`
- Starting of the string:

  `Regex: ^`
- Ending of the string:

  `Regex: $`
- Match word boundary:

```
Regex: \b
```

- Non-word boundary:

```
Regex: \B
```

`re.match()` and `re.search()` functions are used to find the patterns and then can be processed according to the requirements of the application.

Let's look at the differences between `re.match()` and `re.search()`:

- `re.match()`: This checks for a match of the string only at the beginning of the string. So, if it finds the pattern at the beginning of the input string, then it returns the matched pattern; otherwise; it returns a noun.

- `re.search()`: This checks for a match of the string anywhere in the string. It finds all the occurrences of the pattern in the given input string or data.

Now let's look at a few of the examples using these regular expressions.

Tokenizing

You want to split the sentence into words – tokenize. One of the ways to do this is by using `re.split`.

```
# Import library

import re

#run the split query

re.split('\s+','I like this book.')

['I', 'like', 'this', 'book.']
```

For an explanation of regex, please refer to the main recipe.

Extracing email IDs

The simplest way to do this is by using `re.findall`.

1. Read/create the document or sentences

```
doc = "For more details please mail us at:
xyz@abc.com, pqr@mno.com"
```

2. Execute the re.findall function

```
addresses = re.findall(r'[\w\.-]+@[\w\.-]+', doc)
for address in addresses:
    print(address)

#Output
xyz@abc.com
pqr@mno.com
```

Replacing email IDs

Here we replace email ids from the sentences or documents with another email id. The simplest way to do this is by using `re.sub`.

1. Read/create the document or sentences

```
doc = "For more details please mail us at xyz@abc.com"
```

2. Execute the `re.sub` function

```
new_email_address = re.sub(r'([\w\.-]+)@([\w\.-]+)',
r'pqr@mno.com', doc)
print(new_email_address)

#Output
For more details please mail us at pqr@mno.com
```

For an explanation of regex, please refer to Recipe 1-6.

Extract data from the ebook and perform regex

Let's solve this case study by using the techniques learned so far.

1. Extract the content from the book

```
# Import library

import re
import requests

#url you want to extract
```

```
url = 'https://www.gutenberg.org/files/2638/2638-
0.txt'

#function to extract
def get_book(url):
 # Sends a http request to get the text from
project Gutenberg
 raw = requests.get(url).text
 # Discards the metadata from the beginning of the
book
 start = re.search(r"\*\*\* START OF THIS PROJECT
GUTENBERG EBOOK .* \*\*\*",raw ).end()
 # Discards the metadata from the end of the book
 stop = re.search(r"II", raw).start()
 # Keeps the relevant text
 text = raw[start:stop]
 return text

# processing
def preprocess(sentence):
 return re.sub('[^A-Za-z0-9.]+' , ' ',
sentence).lower()

#calling the above function

book = get_book(url)
processed_book = preprocess(book)
print(processed_book)

# Output
 produced by martin adamson david widger with
corrections by andrew sly the idiot by fyodor
dostoyevsky translated by eva martin part i i.
towards the end of november during a thaw at nine o
clock one morning a train on the warsaw and
petersburg railway was approaching the latter city
at full speed. the morning was so damp and misty
that it was only with great difficulty that the day
succeeded in breaking and it was impossible to
distinguish anything more than a few yards away
from the carriage windows. some of the passengers
by this particular train were returning from abroad
but the third class carriages were the best filled
chiefly with insignificant persons of various
occupations and degrees picked up at the different
stations nearer town. all of them seemed weary and
most of them had sleepy eyes and a shivering
expression while their complexions generally
appeared to have taken on the colour of the fog
outside. when da
```

2. Perform some exploratory data analysis on this data using regex

```
# Count number of times "the" is appeared in the
book
len(re.findall(r'the', processed_book))

#Output
302

#Replace "i" with "I"
processed_book = re.sub(r'\si\s', " I ",
processed_book)
print(processed_book)

#output
 produced by martin adamson david widger with
corrections by andrew sly the idiot by fyodor
dostoyevsky translated by eva martin part I i.
towards the end of november during a thaw at nine o
clock one morning a train on the warsaw and
petersburg railway was approaching the latter city
at full speed. the morning was so damp and misty
that it was only with great difficulty that the day
succeeded in breaking and it was impossible to
distinguish anything more than a few yards away
from the carriage windows. some of the passengers
by this particular train were returning from abroad
but the third class carriages were the best filled
chiefly with insignificant persons of various
occupations and degrees picked up at the different
stations nearer town. all of them seemed weary and
most of them had sleepy eyes and a shivering
expression while their complexions generally
appeared to have taken on the colour of the fog
outside. when da

#find all occurance of text in the format "abc--
xyz"
re.findall(r'[a-zA-Z0-9]*--[a-zA-Z0-9]*', book)
```

```
#output
 ['ironical--it',
 'malicious--smile',
 'fur--or',
 'astrachan--overcoat',
 'it--the',
 'Italy--was',
 'malady--a',
 'money--and',
 'little--to',
 'No--Mr',
 'is--where',
 'I--I',
 'I--',
 '--though',
 'crime--we',
 'or--judge',
 'gaiters--still',
 '--if',
 'through--well',
 'say--through',
 'however--and',
 'Epanchin--oh',
 'too--at',
 'was--and',
 'Andreevitch--that',
 'everyone--that',
 'reduce--or',
 'raise--to',
 'listen--and',
 'history--but',
 'individual--one',
 'yes--I',
 'but--',
 't--not',
 'me--then',
 'perhaps--',
 'Yes--those',
 'me--is',
 'servility--if',
 'Rogojin--hereditary',
 'citizen--who',
 'least--goodness',
 'memory--but',
 'latter--since',
 'Rogojin--hung',
 'him--I',
 'anything--she',
 'old--and',
 'you--scarecrow',
 'certainly--certainly',
 'father--I',
 'Barashkoff--I',
 'see--and',
 'everything--Lebedeff',
 'about--he',
 'now--I',
 'Lihachof--',
 'Zaleshoff--looking',
 'old--fifty',
 'so--and',
 'this--do',
 'day--not',
 'that--',
 'do--by',
 'know--my',
 'illness--I',
 'well--here',
 'fellow--you']
```

Recipe 1-7. Handling Strings

In this recipe, we are going to discuss how to handle strings and dealing with text data.

We can do all sort of basic text explorations using string operations.

PROBLEM

You want to explore handling strings.

SOLUTION

The simplest way to do this is by using the below string functionality.

`s.find(t)` index of first instance of string t inside s (-1 if not found)

`s.rfind(t)` index of last instance of string t inside s (-1 if not found)

`s.index(t)` like s.find(t) except it raises ValueError if not found

`s.rindex(t)` like s.rfind(t) except it raises ValueError if not found

`s.join(text)` combine the words of the text into a string using s as the glue

`s.split(t)` split s into a list wherever a t is found (whitespace by default)

`s.splitlines()` split s into a list of strings, one per line

`s.lower()` a lowercased version of the string s

`s.upper()` an uppercased version of the string s

`s.title()` a titlecased version of the string s

`s.strip()` a copy of s without leading or trailing whitespace

`s.replace(t, u)` replace instances of t with u inside s

HOW IT WORKS

Now let us look at a few of the examples.

Replacing content

Create a string and replace the content. Creating Strings is easy, and it is done by enclosing the characters in single or double quotes. And to replace, you can use the `replace` function.

1. Creating a string

```
String_v1 = "I am exploring NLP"

#To extract particular character or range of
characters from string

print(String_v1[0])

#output
"I"

#To extract exploring

print(String_v1[5:14])

#output
exploring
```

2. Replace “exploring” with “learning” in the above string

```
String_v2 = String_v1.replace("exploring",
"learning")
print(String_v2)

#Output
I am learning NLP
```

Concatenating two strings

Here’s the simple code:

```
s1 = "nlp"
s2 = "machine learning"
s3 = s1+s2
print(s3)

#output
'nlpmachine learning'
```

Searching for a substring in a string

Use the `find` function to fetch the starting index value of the substring in the whole string.

```
var="I am learning NLP"
f= "learn"
var.find(f)

#output
```

5

Recipe 1-8. Scraping Text from the Web

In this recipe, we are going to discuss how to scrape data from the web.

CAUTION Before scraping any websites, blogs, or e-commerce websites, please make sure you read the terms and conditions of the websites on whether it gives permissions for data scraping.

So, what is web scraping, also called web harvesting or web data extraction?

It is a technique to extract a large amount of data from websites and save it in a database or locally. You can use this data to extract information related to your customers/users/products for the business's benefit.

Prerequisite: Basic understanding of HTML structure.

PROBLEM

You want to extract data from the web by scraping. Here we have taken the example of the IMDB website for scraping top movies.

SOLUTION

The simplest way to do this is by using beautiful soup or scrapy library from Python. Let's use beautiful soup in this recipe.

HOW IT WORKS

Let's follow the steps in this section to extract data from the web.

Step 8-1 Install all the necessary libraries

```
!pip install bs4
!pip install requests
```

Step 8-2 Import the libraries

```
from bs4 import BeautifulSoup
import requests
import pandas as pd
from pandas import Series, DataFrame
from ipywidgets import FloatProgress
from time import sleep
from IPython.display import display
import re
import pickle
```

Step 8-3 Identify the url to extract the data

```
url = 'http://www.imdb.com/chart/top?ref_=nv_mv_250_6'
```

Step 8-4 Request the url and download the content using beautiful soup

```
result = requests.get(url)
c = result.content
soup = BeautifulSoup(c,"lxml")
```

Step 8-5 Understand the website page structure to extract the required information

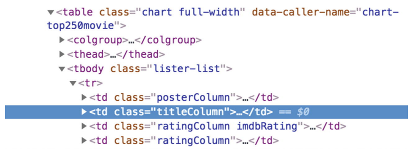
Go to the website and right-click on the page content to inspect the html structure of the website.

Identify the data and fields you want to extract. Say, for example, we want the Movie name and IMDB rating from this page.

So, we will have to check under which div or class the movie names are present in the HTML and parse the beautiful soup accordingly.

In the below example, to extract the movie name, we can parse our soup through `<table class ="chart full-width">` and `<td class="titleColumn">`.

Similarly, we can fetch the other details. For more details, please refer to the code in step 8-6.



```
▼<table class="chart full-width" data-caller-name="chart-top250movie">
  ▶<colgroup>…</colgroup>
  ▶<thead>…</thead>
  ▼<tbody class="lister-list">
    ▼<tr>
      ▶<td class="posterColumn">…</td>
      ▶<td class="titleColumn">…</td> == $0
      ▶<td class="ratingColumn imdbRating">…</td>
      ▶<td class="ratingColumn">…</td>
```

Step 8-6 Use beautiful soup to extract and parse the data from HTML tags

```
summary = soup.find('div',{'class':'article'})

# Create empty lists to append the extracted data.

moviename = []
cast = []
description = []
rating = []
ratingoutof = []
year = []
genre = []
movielength = []
rot_audscore = []
rot_avgrating = []
rot_users = []

# Extracting the required data from the html soup.

rgx = re.compile('[%s]' % '()')
f = FloatProgress(min=0, max=250)
display(f)
for row,i in
zip(summary.find('table').findAll('tr'),range(len(summ
ary.find('table').findAll('tr')))):
    for sitem in row.findAll('span',
{'class':'secondaryInfo'}):
        s = sitem.find(text=True)
        year.append(rgx.sub(", s))
    for ritem in row.findAll('td',
{'class':'ratingColumn imdbRating'}):
        for iget in ritem.findAll('strong'):
            rating.append(iget.find(text=True))
            ratingoutof.append(iget.get('title').split
(' ', 4)[3])
    for item in row.findAll('td',
{'class':'titleColumn'}):
        for href in item.findAll('a',href=True):
            moviename.append(href.find(text=True))
            rurl =
'https://www.rottentomatoes.com/m/'+
href.find(text=True)
            try:
                rresult = requests.get(rurl)
            except
requests.exceptions.ConnectionError:
                status_code = "Connection refused"
            rc = rresult.content
            rsoup = BeautifulSoup(rc)
            try:
                rot_audscore.append(rsoup.find('div',
{'class':'meter-value'}).find('span',
{'class':'superPageFontColor'}).text)
                rot_avgrating.append(rsoup.find('div',
{'class':'audience-info hidden-xs
superPageFontColor'}).find('div').contents[2].strip())
                rot_users.append(rsoup.find('div',
{'class':'audience-info hidden-xs
superPageFontColor'}).contents[3].contents[2].strip())
            except AttributeError:
                rot_audscore.append("")
                rot_avgrating.append("")
                rot_users.append("")
            cast.append(href.get('title'))
            imdb = "http://www.imdb.com" +
href.get('href')
            try:
                iresult = requests.get(imdb)
                ic = iresult.content
                isoup = BeautifulSoup(ic)
                description.append(isoup.find('div',
{'class':'summary_text'}).find(text=True).strip())
                genre.append(isoup.find('span',
{'class':'itemprop'}).find(text=True))
                movielength.append(isoup.find('time',
{'itemprop':'duration'}).find(text=True).strip())
            except
requests.exceptions.ConnectionError:
                description.append("")
                genre.append("")
                movielength.append("")
    sleep(.1)
    f.value = i
```

Note that there is a high chance that you might encounter an error while executing the above script because of the following reasons:

- Your request to the URL has failed, so maybe you need to try again after some time. This is common in web scraping.
- Web pages are dynamic. The HTML tags of websites keep changing. Understand the tags and make small changes in the code in accordance with HTML, and you are good to go.

Step 8-7 Convert lists to data frame and you can perform the analysis that meets the business requirements

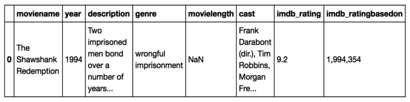
```
# List to pandas series

moviename = Series(moviename)
cast = Series(cast)
description = Series(description)
rating = Series(rating)
ratingoutof = Series(ratingoutof)
year = Series(year)
genre = Series(genre)
movielength = Series(movielength)
rot_audscore = Series(rot_audscore)
rot_avgrating = Series(rot_avgrating)
rot_users = Series(rot_users)

# creating dataframe and doing analysis

imdb_df =
pd.concat([moviename,year,description,genre,movielengt
h,cast,rating,ratingoutof,rot_audscore,rot_avgrating,r
ot_users],axis=1)
imdb_df.columns =
['moviename','year','description','genre','movielength
','cast','imdb_rating','imdb_ratingbasedon','tomatoes_
audscore','tomatoes_rating','tomatoes_ratingbasedon']
imdb_df['rank'] = imdb_df.index + 1
imdb_df.head(1)

#output
```



	moviename	year	description	genre	movielength	cast	imdb_rating	imdb_ratingbasedon
0	The Shawshank Redemption	1994	Two imprisoned men bond over a number of years...	wrongful imprisonment	NaN	Frank Darabont (dir.), Tim Robbins, Morgan Fre...	9.2	1,994,354

Step 8-8 Download the data frame

```
# Saving the file as CSV.

imdb_df.to_csv("imdbdataexport.csv")
```

We have implemented most of the ways and techniques to extract text data from possible sources. In the coming chapters, we will look at how to explore, process, and clean this data, followed by feature engineering and building NLP applications.

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Front Matter

NEXT
2. Exploring and Processing Text Data