Edgar Financial Data Scraper

Introduction

Language used: Python

Tools used: Jupyter Notebooks, Amazon S3 bucket, Docker, Sublime text

Process: Web Scraping, Data Munging, Data Cleaning, Exploratory Data Analysis.

Sreerag Mandakathil Sreenath

001838559

mandakathil.s@husky.neu.edu

Shreya Chudasama

001828562

chudasama.s@husky.neu.edu

Aahana Khajanchi

001824402

khajanchi.a@husky.neu.edu

We are extracting all tables from 10Q filings of their Edgar finance records using R/Python. So given a company with CIK(ex. 51143) and document accession number (ex.000005114313000007) , we are programmatically generating the url (http://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007- index.html for IBM) to get data .
Then we are parsing the file to locate the link to the 10Q file: (https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm for the above example).
Parse this file to extract "all" tables in this filing and save them as csv files and then upload the csv and log files on S3 cloud.

Github Link: https://github.com/sreeragsreenath/AdavancedDataScience.git

Approach

Process for Part 1: Scraping the tables

1. We are passing "cik" , "accession number", "AWS_ACCESS_KEY_ID" and AWS_SECRET_ACCESS_KEY as system arguments and then programmatically generate the edgar url using cik and accession number.

Fetching the url using cik and accession number

```
base_url = 'https://www.sec.gov'
cik = "51143"
acc_num = "000005114313000007"
acc_num_index = acc_num[0:10]+"-"+acc_num[10:12]+"-"+acc_num[12:]+"-index.html"
url_rendered = base_url + "/Archives/edgar/data/" + cik + "/" + acc_num +"/" + acc_num_index
form_url = base_url + "/" + cik + "/" + acc_num
print(url_rendered)
```

```
https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007-index.html
```

2. Then we search for the 10Q link on that page using python library urlopen and Beautiful Soup to request the url and parse through the HTML page .

Fetching the 10q link from the URL

```
uClient = ureq(url_rendered)
page_html=uClient.read()
page_soup = soup(page_html, 'html.parser')
divs = page_soup.find('table',summary="Document Format Files")
url2=divs.find_all('tr')[1].find_all('td')[2].find('a')['href']
my_url2=urllib.parse.urljoin(base_url, url2)
```

```
'https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm'
```

3. Then we are finding all the href on the pages i.e. getting all the contents of Index and storing that in a list and discarding all the “None” elements from the list.

```
link_list=[]
link_text=[]

for link in page_soup2.find_all('a'):
    href_link=link.get('href')
    href_text=link.get_text()
    if href_link is not None:
        href_link=str(href_link).strip('#')
        is_Exists = page_soup2.find("a",{'name':href_link})
        if (is_Exists is not None):
            link_list.append(href_link)
            link_text.append(href_text)
logging.debug('List of all Href tags : '+str(link_list))
print(str(link_list))
```

```
['Part1FinInfo', 'ConsolidatedFS', 'ConsolEarnings', 'ConsolBS', 'ConsolCF', 'NotesToConsolFS', 'Controls', 'OtherIn
Proceedings', 'Unregistered', 'Exhibits']
```

4. Once we have got the list of all the href, we are trying to fetch the data between two tags. For example we will find contents between ‘Part1finInfo’ and .ConsolidatedFS’ and store it in a new Soup and search for tables in that data. We have defined a new function which takes soup, first tag and last tag as parameter and returns the content in between them.

```
def find_between( soup, firstTag, lastTag ):
    try:
        start = soup.index( firstTag ) + len( firstTag )
        end = soup.index( firstTag, start )
        return soup[start:end]
    except ValueError:
        return ""
```

5. This approach worked well for few firms but we faced issue when there was text data written under table tags, so our script captured all that text data and exportit in csv too.
In order to avoid this and scrape only tables with financial records we searched for the “style” tag of the table as only the tables with financial data have the style tag in their tables.
Below is the function used to check if a table row of a table has style tag with background colour.

```
def checktag(param):
    flag = "false"
    datatabletags = ["background", "bgcolor", "background-color"]
    for x in datatabletags:
        if x in param:
            flag = "true"
    return flag
```

To make it compatible with 10q files of all firms we are checking for “Background”, “bgcolor” and “background-color ” tags as there are few 10q files that are written in older version of HTML.

Below is the code that checks if the contents fetched using find_between method has tables i.e.(len(tables)>0) and then check for the style tags in each row and retrieve the data and export them in a csv.

```
new_soup = find_between( page_soup2.prettify(), first, last )
new_bs = soup(new_soup, 'html.parser')
tables = new_bs.find_all("table")
if(len(tables)>0):
    table_all_rows=[]
    for table in tables:
        table_row = table.select('tr')
        tds = table.select('td')
        flag=0
        for td in tds:
            if checktag(str(td.get('style'))) == "true" or checktag(str(td)) == "true":
                flag=1
                break
        for tr in table_row:
            if(flag==1):
                table_column = tr.select('font')
                row = []
                for k in table_column:
                    k_text = k.text.replace(u'\xa0',u'')
                    k_text = k_text.replace(u'\n',u'')
                    if(k_text!='\n' ):
                        row.append(k_text)
                table_all_rows.append(row)
```

6. To log the process and time when it was done , we are using python library "logging".

```
ts = time.time()
st = datetime.datetime.fromtimestamp(ts).strftime('%Y%m%d_%H%M%S')
logfilename = 'log_Edgar_'+ cik + '_' + st + '.txt'
logging.basicConfig(filename=logfilename, level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(message)s')
logging.debug('Program Start')
logging.debug('CIK Number : '+cik+'  and Accession number : '+acc_num+ '  Url :'+ url_rendered)
```

We have added logs at different steps, few are:

- When the program starts and a URL is generated using the cik and accession number.
- If the url is not found or it is incorrect due to wrong cik or accession number.
- List of all the href tags.
- Creation of directory and csv.
- Successful or unsuccessful upload of contents on Amazon S3.
- Creation of zip folder after the csv has created.

Docker image on AWS

We have used an Ubuntu 16.4 image and updated its libraries. Then we installed Python3 and Python pip to execute the python script for web scraping. Required libraries has been stated in the dockerfiles itself which include :

- Lxml
- Beautiful soup 4
- Boto
- Urllib 3

Points learned:

- Dockers does not allow you to write on the root folder of an image, so we need to create a folder within and run the python script there.
- It is better to store all the heavy libraries and cache them , that will save our processing time.

Data Cleaning

The data has NaN values in the size and browser column. In the extention column, the data consist of many files, so we fetched it according to its extention.

1) We created a data frame for downloading the zip files and then reading those csv files using pandas to insert the data and formulated 'for' loop to iterate through all the files
2) The user will enter the year, for which log data is generated
3) For a particular year, it will generate the log files of all the 12 months
4) Used urllib.request to get the URL
5) It will download all the zip files in a particular folder, using python library zipfile to extract all the zip files of a particular year
6) To load the data, used pandas library read_csv

```
month=1
for i in range(12):

    if month in range(1,4):qtr = 1; month = "0"+str(month)
    elif month in range(4,7):qtr = 2; month = "0"+str(month)
    elif month in range(7,10): qtr = 3; month = "0"+str(month)
    elif month in range(10,13): qtr = 4
    else :pass

    r = requests.get("http://www.sec.gov/dera/data/Public-EDGAR-log-file-data/"+y+"/Qtr"+str(qtr)+"/log"+y+str(month)+"01.zip")
    z =zipfile.ZipFile(io.BytesIO(r.content))
    z.extractall(path)

    print(y+"/Qtr"+str(qtr)+"/log"+y+str(month)+"01.zip")
    month = int(month) + 1
```

```
2004/Qtr1/log20040101.zip
2004/Qtr1/log20040201.zip
2004/Qtr1/log20040301.zip
2004/Qtr2/log20040401.zip
2004/Qtr2/log20040501.zip
2004/Qtr2/log20040601.zip
2004/Qtr3/log20040701.zip
2004/Qtr3/log20040801.zip
2004/Qtr3/log20040901.zip
2004/Qtr4/log20041001.zip
2004/Qtr4/log20041101.zip
2004/Qtr4/log20041201.zip
```

```
data.head()
```

	ip	date	time	zone	cik	accession	extention	code	size	idx	norefer	noagent	find	crawler	browser
0	162.83.150.fab	2004-12-01	00:00:00	500.0	1065088.0	0001184529-04-000006	.txt	200.0	6524.0	0.0	1.0	0.0	0.0	0.0	win
1	162.83.150.fab	2004-12-01	00:00:00	500.0	1065088.0	0001184529-04-000006	-index.htm	200.0	2605.0	1.0	1.0	0.0	0.0	0.0	win
2	68.35.159.gjh	2004-12-01	00:00:00	500.0	889971.0	0001193125-03-055569	d10k.htm	200.0	930609.0	0.0	0.0	0.0	9.0	0.0	win
3	131.193.228.gae	2004-12-01	00:00:01	500.0	909793.0	0001193125-04-050920	d10k.htm	200.0	1104303.0	0.0	1.0	1.0	0.0	0.0	NaN
4	207.59.127.iee	2004-12-01	00:00:01	500.0	1173495.0	0001047469-04-029953	a2144175z8-k.htm	304.0	NaN	0.0	0.0	0.0	9.0	0.0	win

7) After loading the data, we calculated the null values using data.isnull().sum()

```
data.isnull().sum()
```

```
ip                0
date              0
time              0
zone              0
cik               0
accession         0
extention         0
code              0
size          26466
idx               0
norefer           0
noagent           0
find              0
crawler           0
browser      211839
dtype: int64
```

8) To get the information about each column, we have used data.info()

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440178 entries, 0 to 440177
Data columns (total 15 columns):
ip           440178 non-null object
date         440178 non-null object
time         440178 non-null object
zone         440178 non-null float64
cik          440178 non-null float64
accession    440178 non-null object
extention    440178 non-null object
code         440178 non-null float64
size         413712 non-null float64
idx          440178 non-null float64
norefer      440178 non-null float64
noagent      440178 non-null float64
find         440178 non-null float64
crawler      440178 non-null float64
browser      228339 non-null object
dtypes: float64(9), object(6)
memory usage: 50.4+ MB
```

9) For extraction of the extentions from the column extention, we have used regex

```
data['extention'] = data['extention'].apply(lambda x: re.findall("\..*", x)[0][1: ])
```

```
print(data['extention'])
```

```
0         htm
1         txt
2         xml
3         txt
4         htm
5         htm
6         xml
7         htm
8         txt
9         htm
10        htm
11        htm
12        htm
13        htm
14        txt
15        txt
16        htm
17        htm
18        xml
19        htm
20        xml
21        htm
```

```
data['extention'].unique()
```

```
array(['htm', 'txt', 'xml', 'paper', 'pdf', 'html', 'hdr.sgml', 'fil',
       'ht', 'x', 'sec.gov', 'xm', '', 'htm.ed2k', 'frm', 'e20-f.txt',
       'htmProxy'], dtype=object)
```

10) As there was only one numeric column, size which was having missing values, to handle that we have calculated the mean value of size column then replaced NaN values with the mean value.

```
# Filling Null data in 'size' by mean value
```

```
data['size'].fillna(data['size'].mean(), inplace = True)
```

```
data['size'].isnull().any()
```

```
False
```

```
data.isnull().sum()
```

```
ip                0
date              0
time              0
zone              0
cik               0
accession         0
extention         0
code              0
size              0
idx               0
norefer           0
noagent           0
find              0
crawler           0
browser      211839
dtype: int64
```

11) For the browser column, we replaced it's missing values using the most occuring value.

```
data.isnull().sum()
```

```
ip                0
date              0
time              0
zone              0
cik               0
accession         0
extention         0
code              0
size              0
idx               0
norefer           0
noagent           0
find              0
crawler           0
browser      211839
dtype: int64
```

```
x=data['browser'].value_counts().max()
```

```
data['browser'].value_counts()
```

```
win    216624
mie     11324
mac       237
lin       140
opr        13
iem         1
Name: browser, dtype: int64
```

```
x=data['browser'].mode()    # most occuring value
```

```
data['browser'].fillna(x[0], inplace = True)
```

```
x[0]
```

```
'win'
```

```
data['browser'].value_counts()
```

```
win    428463
mie     11324
mac       237
lin       140
opr        13
iem         1
Name: browser, dtype: int64
```

12) Now the data is cleaned and we don't have any missing or NaN values

```
data.isnull().any()
```

```
ip           False
date         False
time         False
zone         False
cik          False
accession    False
extention    False
code         False
size         False
idx          False
norefer      False
noagent      False
find         False
crawler      False
browser      False
dtype: bool
```

Checking for Anomalies

1) Checking if any missing values
2) To check if any values exist other than 0 or 1 in idx, norefer and noagent, if it would present, then it's an anomaly.

```
data['noagent'].value_counts()
```

```
0.0    282953
1.0    157225
Name: noagent, dtype: int64
```

```
data['norefer'].value_counts()
```

```
1.0    271794
0.0    168384
Name: norefer, dtype: int64
```

```
data['idx'].value_counts()
```

```
0.0    245456
1.0    194722
Name: idx, dtype: int64
```

Exploratory Data Analysis

Plotted the graphs to analyze the data:-

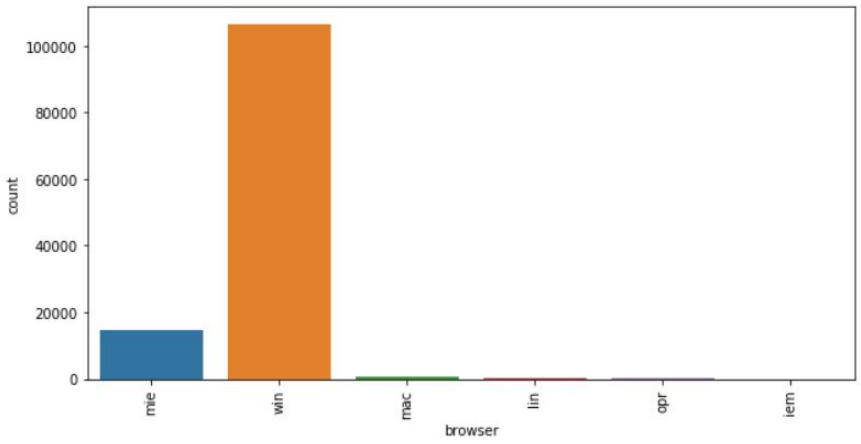
The browser ‘win’ has the maximum number of counts

```
In [73]: # matplotlib.figure.Figure(figsize= (w,h)) tuple in inches

from matplotlib import pyplot as plt
plt.figure(figsize=(10,5))


# seaborn.countplot - Show the counts of observations in each categorical bin using bars.
# A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable

sns.countplot(data['browser'])
plt.xticks(rotation = 'vertical')
#plt.title('Manufacturers distribution in dataset')
#plt.ylabel('Number of vehicles')
plt.show()
```




```
data['browser'].value_counts()
```

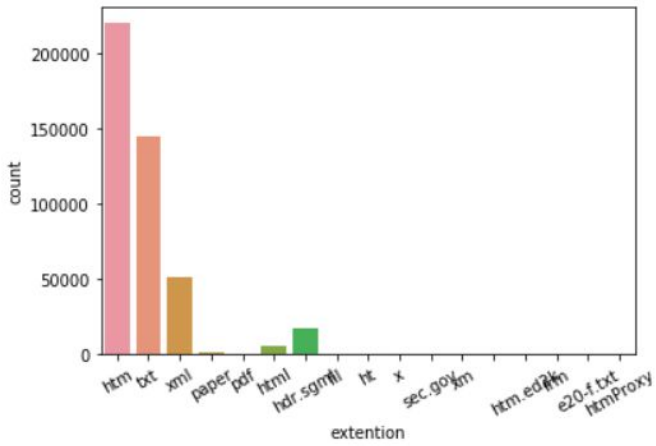
```
win    216624
mie     11324
mac       237
lin       140
opr        13
iem         1
Name: browser, dtype: int64
```

Browser's max frequency is in win and min in iem

Plotted countplot to find the frequency count of extention. The maximum count of extention is htm.

```
sns.countplot(data['extention'])
plt.xticks(rotation = '30')
```

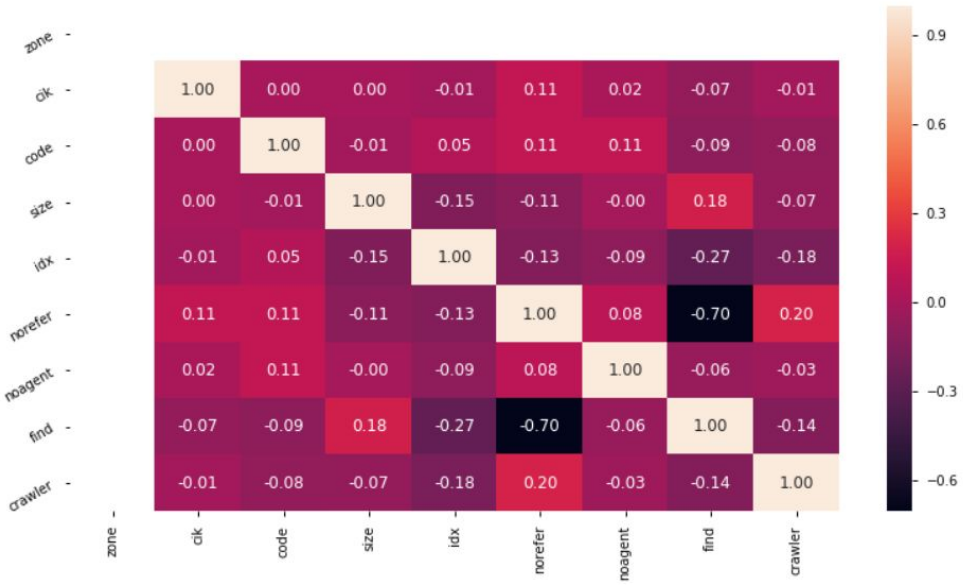
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]),
 <a list of 17 Text xticklabel objects>)
```



Plotted correlation and heat map to analyze the data

```
import seaborn as sns
correlation = data.corr()
sns.set_context("notebook", font_scale = 1.0, rc = {"lines.linewidth" : 2.5})
plt.figure(figsize=(13, 7))
a = sns.heatmap(correlation,annot = True, fmt = '.2f')

rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



Summary Metrics

```
date = data['date'].max()
```

```
date # Computing date of the data
```

```
'2004-12-01'
```

```
tot_record = len(data)
```

```
tot_record# computing total number of records
```

```
715514
```

```
max_c=data['cik'].value_counts()
```

```
max_cik= max_c.max()
```

```
max_cik# Max count value of cik
```

```
5157
```

```
max_size = data['size'].max()# Computing maximum size
```

```
max_size
```

```
46477728.0
```

```
max_size
```

```
46477728.0
```

```
sum_size = data['size'].sum()
```

```
length_size = len(data)
```

```
avg_size= sum_size/length_size# computing average of size
```

```
avg_size
```

```
46815.64302026236
```

```
max_ext = data['extention'].value_counts()
```

```
extention_max = data.extention[max_ext.max()]
```

```
extention_max #Computing maximum extention used
```

```
'.txt'
```

```
max_brw = data['browser'].value_counts()
```

```
max_brw.max()
```

```
579897
```

```
brw_max = data.browser[max_brw.max()]
```

```
brw_max # Computing most occur browser
```

```
'win'
```

```
row_entry = pd.Series([date, tot_record, max_cik, max_size, avg_size, extention_max, brw_max])
```

```
row_entry
```

```
0     2004-12-01
1         715514
2           5157
3    4.64777e+07
4        46815.6
5            .txt
6             win
dtype: object
```

```
# Creating Summary metrics
summary_metrics = pd.DataFrame(columns=['Date','Total Records','Most Cik',
                               'Maximum File Size','Average File size','Most used extention','Most used browser'])
```

```
summary_metrics
```

Date	Total Records	Most Cik	Maximum File Size	Average File size	Most used extention	Most used browser

```
import csv
```

```
courses_list=[]
```

```
summary_metrics = summary_metrics.append(
                   pd.Series([date, tot_record, max_cik, max_size, avg_size, extention_max, brw_max],
                            index=['Date','Total Records','Most Cik',
                              'Maximum File Size','Average File size','Most used extention','Most used browser'])
                   ,ignore_index=True)
```

```
summary_metrics
```

	Date	Total Records	Most Cik	Maximum File Size	Average File size	Most used extention	Most used browser
0	2004-12-01	715514	5157	46477728.0	46815.64302	.txt	win

The summary metrics will show the most cik, maximum file size, average file size, most used extention and the highest frequency count of the browser.