

# Data Wrangling Using Edgar Data

## Part-1

Report by:

Ann Sara Sajee

Gunjan Ratan Lalwani

Rishabh Jain

## **OBJECTIVE**

The report summarizes the design and implementation of the data wrangling performed on the Edgar website. This report is divided into two sections.

**Section 1:** Describes the step by step implementation of fetching the 10 – Q data table provide company CIK and accession number.

**Section 2:** Describe the implementation to streamline the process using Docker and host the resultant output on Amazon S3.

## **SECTION 1 PARSE THE FILES**

### **STEP 1:**

#### **GET THE CIK AND ACCESSION NUMBER FROM THE USER**

Given Edgar company CIK code (maximum length of 10) and accession number by the user, we will be programmatically generating the URL for requested company CIK.

Before generating the URL, we have to make sure that we remove all the leading zeros from the CIK before hitting the URL.

For error handling, we will check if the response of the request is 200 (URL is valid) and go to the next step to fetch the request file form the next URL.

```

#importing libraries
import requests
from bs4 import BeautifulSoup
import pandas as pd
import re
import os

#input
def incheck(arg):
    default = '0000051143-13-000007'
    if re.match('[0-9]{10}-[0-9]{2}-[0-9]{6}',arg):
        return arg
    else:
        return default

avalid = input("Enter a value:")
acc_no = incheck(avalid)

# Generation of URL using acc_no
url = 'http://www.sec.gov/Archives/edgar/data/'
parts = acc_no.split('-')
cik = parts[0]
year = parts[1]
no = parts[2]
CIK = cik.lstrip('0')
link = url+CIK+'/'+cik+year+acc_no+'/'+acc_no+'-index.html'
print(link)

Enter a value:0001288776-15-000046
http://www.sec.gov/Archives/edgar/data/1288776/0001288776150001288776-15-000046/0001288776-15-000046-index.html

```

## STEP 2:

### CREATE & ACCESS THE EDGAR URL BASED ON PARAMETER PROVIDED

Since we are using python 3.2 for accessing the URL, we have to import the following python library to access the Edgar website.

`import urllib.request` :- To open a requested URL

In the next step, we will be hitting the actual URL which consist of the form need to be scrapped. To achieve this task, we will be using BeautifulSoup, a python library which is very efficient to work with the HTML files.

`from bs4 import BeautifulSoup` : - To fetch the entire HTML content from the requested file.

To fetch the next Url, we will find all the value of anchor tags and create the furl to open the file requested.

```
#Getting Html Document list
request = requests.get(link)
html_doc = request.text

#Getting 10q tag#
link10q = ""
soup = BeautifulSoup(html_doc, "html.parser")
all_tables = soup.find('table', class_='tableFile')
tr = all_tables.find_all('tr')
for row in tr:
    link10q = 'https://www.sec.gov'+ row.findNext("a").attrs['href']
    break
print(link10q)
```

<https://www.sec.gov/Archives/edgar/data/1288776/000128877615000046/goog10-qq32015.htm>

### STEP 3:

#### FIND THE REQUEST FILE & GET THE URL ASSOCIATED WITH THE FILE

After getting the actual URL for fetching the requested file content and parse the HTML content, we will be fetching the actual HTML content inside a soup object and analyze the data table required for the analysis.

For error handling, we will check if the response of the request is 200 (URL is valid) and go to the next step to fetch the requested file form from the URL.

```

try:
    request1 = requests.get(link10q)
    html_doc1 = request1.text
    BeautifulSoup(html_doc1,'lxml') if request1.status_code == 200 else None
except:
    None
    soup1 = BeautifulSoup(html_doc1,'lxml')
print(soup1)

```

---

```

<html><body><document>
<type>10-Q
<sequence>1
<filename>goog10-qq32015.htm
<description>FORM 10-Q
<text>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<!-- Document created using Wdesk 1 -->
<!-- Copyright 2015 Workiva -->
<title>10-Q</title>
<a name="s436AB9DAC474640F38853E137F6491C2"></a><div><div style="line-height:120%;text-align:center;font-size:1
nt:0px;line-height:normal;padding-top:10px;"><table cellpadding="0" cellspacing="0" style="font-family:Times New Roman
auto;width:99.609375%;border-collapse:collapse;text-align:left;"><tr><td colspan="2"></td></tr><tr><td width="67%"></td>
="vertical-align:bottom;padding-left:2px;padding-top:2px;padding-bottom:2px;padding-right:2px;"><div style="font-size:10;
E137FB9A2C8" style="font-family:Arial;font-size:10pt;"><font style="font-family:Arial;font-size:10pt;">Table of Contents<
gn:bottom;padding-left:2px;padding-top:2px;padding-bottom:2px;padding-right:2px;"><div style="text-align:right;font-size:1
e:10pt;color:#656565;font-weight:bold;">Google Inc.</font></div></td></tr></table></div></div></div><br><div style="1
v><div style="line-height:120%;text-align:center;"><hr></div><div style="line-height:120%;text-align:center;font-size:14pt
pt;font-weight:bold;">UNITED STATES</font></div><div style="line-height:120%;text-align:center;font-size:14pt;"><font

```

#### STEP 4:

#### FIND THE DATA TABLES FROM THE HTML CODE

In our analysis, we found that the form comprises of multiple tables and to fetch the table of interest that is our data tables, all of them were having a striped representation. We fetch all the tables having the background color in the row or column level as our final data table.

```

def find_all_datatables(page, all_divtables):
    logging.debug('In a function : find_all_datatables')
    count = 0
    allheaders=[]
    for table in all_divtables:
        bluetables = []
        trs = table.find_all('tr')
        for tr in trs:
            global flagtr
            if checktag(str(tr.get('style'))) == "true" or checktag(str(tr)) == "true":
                logging.debug('Checking data tables at Row Level')
                bluetables = printtable(tr.find_parent('table'))
                break
            else:
                tds = tr.find_all('td')
                for td in tds:
                    if checktag(str(td.get('style'))) == "true" or checktag(str(td)) == "true":
                        logging.debug('Checking data tables at Column Level')
                        bluetables = printtable(td.find_parent('table'))
                        break
                if not len(bluetables) == 0:
                    break
        if not len(bluetables) == 0:
            logging.debug('Total Number of data tables to be created {}'.format(len(bluetables)))
            count += 1
            ptag=table.find_previous('p');
            while ptag is not None and checkheadertag(ptag.get('style'))=="false" and len(ptag.text)<=1:
                ptag=ptag.find_previous('p')
                if checkheadertag(ptag.get('style'))=="true" and len(ptag.text)>=2:
                    global name
                    name=re.sub(r"^[^A-Za-z0-9]+", "",ptag.text)
                    if name in allheaders:
                        hrcount+=1
                        hname=name+" "+str(hrcount)

```

## STEP 5:

### FETCH THE TABLE HEADER

After getting all the data tables, we will try to fetch the table header for which we have created a function which we check the html paragraph for the table header. If the header is present, we will create a file with the table header name else we will create a file with company file name.

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

def checkheadertag(param):
    logging.debug("In a function : checkheadertag")
    setflag="false"
    datatagtags=["center","bold"]
    for x in datatagtags:
        if x in param:
            setflag="true"
    return setflag

def printtable(table):
    logging.debug("In a function : printtable")
    printtable = []
    printtrs = table.find_all('tr')
    for tr in printtrs:
        data=[]
        pdata=[]
        printtds=tr.find_all('td')
        for elem in printtds:
            x=elem.text;
            x=re.sub(r"[\()]", "",str(x))
            x=re.sub(r"[$]", " ",str(x))
            if(len(x)>1):
```

## STEP 6:

### CREATE CSV FOR EACH DATA TABLE

After getting all the data tables, we have to export data of each of these tables and create a csv file for each table. We will create a folder name with the name of the file and then create output csv inside that folder.

```

        break
    folder_name = foldername(page)
    logging.debug('folder created with folder Name {}'.format(folder_name))
    path = str(os.getcwd()) + "/" + folder_name
    logging.debug('Path for csv creation {}'.format(path))
    assure_path_exists(path)
    if(len(allheaders)==0):
        filename=folder_name+"-"+str(count)
    else:
        filename=allheaders.pop()
    csvname=filename+".csv"
    logging.debug('file creation Name {}'.format(csvname))
    csvpath = path + "/" + csvname
    logging.debug('CSV Path for the file creation {}'.format(csvpath))
    with open(csvpath, 'w', encoding='utf-8-sig', newline='') as f:
        writer = csv.writer(f)
        writer.writerows(blueables)
    zip_dir(path)

```

**STEP 7:****CLEAN THE OUTPUT CSV FILE**

For cleaning the final csv created, we have used the regular expression to filter all the special character which will be encountered while fetching the complete data table.

```

for elem in printtds:
    x=elem.text;
    x=re.sub(r"[\()]", "", str(x))
    x=re.sub(r"[$]", " ", str(x))
    if(len(x)>1):
        x=re.sub(r"[-]", "", str(x))
        pdata.append(x)
    data=([elem.encode('utf-8') for elem in pdata])
    printtable.append([elem.decode('utf-8').strip() for elem in data])
return printtable

```

**STEP 8:****CREATE ZIP FILE FOR THE ACTUAL RESULTANT FOLDER**

After getting all the csv, we have to create a zip file for the final output.



```
def zip_dir(path_dir, path_file_zip=""):
    if not path_file_zip:
        logging.debug('In a function : zip_dir')
        path_file_zip = os.path.join(
            os.path.dirname(path_dir), os.path.basename(path_dir) + '.zip')
    with zipfile.ZipFile(path_file_zip, 'w', zipfile.ZIP_DEFLATED) as zip_file:
        for root, dirs, files in os.walk(path_dir):
            for file_or_dir in files + dirs:
                zip_file.write(
                    os.path.join(root, file_or_dir),
                    os.path.relpath(os.path.join(root, file_or_dir),
                                    os.path.join(path_dir, os.path.pardir)))
```

### Step 9:

#### LOGGING EACH STEP IN A LOG FILE

After the final result, we also created and captured the various processing of the task in the log file named, **Edgar log** which will consist of minute details on which process is running and also capture the last point of failure based on the input parameters given. For each CIK and accession, e creates a log file with CIK name and timestamp so that user can easily find the problem associated with process in the log file.

```

2018-02-15 21:35:17,689 - DEBUG - Program Start
2018-02-15 21:35:17,689 - DEBUG - *****
2018-02-15 21:35:17,690 - DEBUG - Calling the initial URL with CIK Number 00003201
2018-02-15 21:35:17,690 - DEBUG - In the function : get_url
2018-02-15 21:35:17,691 - DEBUG - Calling the initial URL for CIK 320193 & Accessi
2018-02-15 21:35:19,363 - DEBUG - URL Exisits
2018-02-15 21:35:19,364 - DEBUG - In the function : get_final_url
2018-02-15 21:35:19,850 - DEBUG - Final URL https://www.sec.gov/Archives/edgar/dat
2018-02-15 21:35:19,851 - DEBUG - In the function : get_soup
2018-02-15 21:35:24,302 - DEBUG - In the function : find_all_tables
2018-02-15 21:35:24,339 - DEBUG - In a function : find_all_datatables
2018-02-15 21:35:24,366 - DEBUG - Checking data tables at Row Level
2018-02-15 21:35:24,366 - DEBUG - In a function : printtable
2018-02-15 21:35:24,369 - DEBUG - Total Number of data tables to be created 16
2018-02-15 21:35:24,381 - DEBUG - In the function : foldernamea10-qq1201812302017
2018-02-15 21:35:24,381 - DEBUG - folder created with folder Namea10-qq12018123020
2018-02-15 21:35:24,382 - DEBUG - Path for csv creation C:\Users\Deepak\spyder-py
2018-02-15 21:35:24,382 - DEBUG - In a function : assure_path_exists
2018-02-15 21:35:24,383 - DEBUG - file creation Namea10-qq1201812302017-1.csv
2018-02-15 21:35:24,383 - DEBUG - CSV Path for the file creation C:\Users\Deepak\
2018-02-15 21:35:24,409 - DEBUG - In a function : zip_dir
2018-02-15 21:35:24,517 - DEBUG - Checking data tables at Row Level
2018-02-15 21:35:24,518 - DEBUG - In a function : printtable
2018-02-15 21:35:24,529 - DEBUG - Total Number of data tables to be created 28
2018-02-15 21:35:24,555 - DEBUG - In the function : foldernamea10-qq1201812302017
2018-02-15 21:35:24,555 - DEBUG - folder created with folder Namea10-qq12018123020
2018-02-15 21:35:24,555 - DEBUG - Path for csv creation C:\Users\Deepak\spyder-py
2018-02-15 21:35:24,556 - DEBUG - In a function : assure_path_exists
2018-02-15 21:35:24,556 - DEBUG - file creation Namea10-qq1201812302017-2.csv
2018-02-15 21:35:24,557 - DEBUG - CSV Path for the file creation C:\Users\Deepak\
2018-02-15 21:35:24,562 - DEBUG - In a function : zip_dir
2018-02-15 21:35:24,671 - DEBUG - Checking data tables at Row Level
2018-02-15 21:35:24,672 - DEBUG - In a function : printtable

```

## **SECTION 2 DOCKERIZE THE PYTHON SCRIPT**

### **STEP 1:**

#### **CREATE THE DOCKER IMAGE FROM DOCKERFILE**

##### 1.1 Created a Dockerfile

```
1 FROM python:3
2
3 # Install app dependencies
4 RUN pip install Flask
5
6
7 # install additional python packages
8 RUN pip3 install ipython
9 #RUN pip install jupyter
10 RUN pip3 install numpy
11 RUN pip3 install pandas
12 RUN pip3 install scikit-learn
13 RUN pip3 install BeautifulSoup4
14 RUN pip3 install scipy
15 RUN pip3 install requests
16 #RUN pip install nltk
17
18 #install AWS CLI
19 RUN pip3 install awscli
20
21 #install Luigi
22 RUN pip3 install luigi
23
24 # install unzip utility
25 #RUN apt-get install unzip
26
27
28 # Bundle app source
29 COPY scrappingDataHtml.py /src/scrappingDataHtml.py
30
31 EXPOSE 8000
32 CMD ["python", "/src/scrappingDataHtml.py", "-p 8000"]
```

### **STEP 2:**

#### **BUILD THE DOCKERFILE TO CREATE THE DOCKER IMAGE**

```
Deepak@Deepak MINGW64 ~/Desktop/AdsAssignment1/Assignment1 (master)
$ Docker build -f Dockerfile .
Sending build context to Docker daemon 459.3kB
Step 1/14 : FROM python:3
--> c1e459c00dc3
Step 2/14 : RUN pip install Flask
--> Using cache
--> 45cbad4be326
Step 3/14 : RUN pip3 install ipython
--> Using cache
--> 0c82b9083cc1
Step 4/14 : RUN pip3 install numpy
--> Using cache
--> f667fae7eafe
Step 5/14 : RUN pip3 install pandas
--> Using cache
--> 16dc45f86dcc
Step 6/14 : RUN pip3 install scikit-learn
--> Using cache
--> 0ab12d761ca2
Step 7/14 : RUN pip3 install BeautifulSoup4
--> Using cache
--> 788d458e7ee4
Step 8/14 : RUN pip3 install scipy
--> Using cache
--> 14484632090b
Step 9/14 : RUN pip3 install requests
--> Using cache
--> f444dfacaba8
Step 10/14 : RUN pip3 install awscli
--> Using cache
--> 29b0f0b19b31
Step 11/14 : RUN pip3 install luigi
--> Using cache
--> 03446560d9fe
Step 12/14 : COPY scrappingDataHtml.py /src/scrappingDataHtml.py
--> a65b0d915aa1
Step 13/14 : EXPOSE 8000
--> Running in dec138816bc2
Removing intermediate container dec138816bc2
--> 2f5a576b70a7
Step 14/14 : CMD ["python", "/src/scrappingDataHtml.py", "-p 8000"]
--> Running in f091d1d9e4bd
Removing intermediate container f091d1d9e4bd
--> 7630b6c9abf0
Successfully built 7630b6c9abf0
```



**STEP 3:**

**PUSH THIS DOCKER IMAGE to DOCKERHUB**

```
Deepak@Deepak MINGW64 ~/Desktop/AdsAssignment1/Assignment1 (master)
$ Docker push lalwanigunjan/sc1
The push refers to repository [docker.io/lalwanigunjan/sc1]
6cee3b5a3454: Pushed
c6d95e9e3c57: Mounted from lalwanigunjan/sc
acd77ac35992: Mounted from lalwanigunjan/sc
97fca22711e1: Mounted from lalwanigunjan/sc
8457a13abd9c: Mounted from lalwanigunjan/sc
ab069d6c1306: Mounted from lalwanigunjan/sc
528fe1bf4a40: Mounted from lalwanigunjan/sc
85ace02fd9c4: Mounted from lalwanigunjan/sc
0818835a8c94: Mounted from lalwanigunjan/sc
4a0767d45c36: Mounted from lalwanigunjan/sc
44d37ab6f150: Mounted from lalwanigunjan/sc
6dce5c484bde: Mounted from lalwanigunjan/sc
057c34df1f1a: Mounted from lalwanigunjan/sc
3d358bf2f209: Mounted from lalwanigunjan/sc
0870b36b7599: Mounted from lalwanigunjan/sc
8fe6d5dcea45: Mounted from lalwanigunjan/sc
06b8d020c11b: Mounted from lalwanigunjan/sc
b9914afd042f: Mounted from lalwanigunjan/sc
4bcdffd70da2: Mounted from lalwanigunjan/sc
latest: digest: sha256:e1fcb38bb14dc004c855d2b77f08e091a845ea89a99f
6ca94b4 size: 4330
```

Secure | <https://hub.docker.com>

Type to filter repositories by name

 <a href="#">lalwanigunjan/edgardatacrapping</a> public	0 STARS	2 PULLS	<a href="#">DETAILS</a>
 <a href="#">lalwanigunjan/edgarscrapping</a> public	0 STARS	2 PULLS	<a href="#">DETAILS</a>
 <a href="#">lalwanigunjan/gunjandocker</a> public	0 STARS	1 PULLS	<a href="#">DETAILS</a>

#### STEP 4:

#### RUN THE DOCKER IMAGE

```
Deepak@Deepak MINGW64 ~/Desktop/AdsAssignment1/Assignment1 (master)
$ docker run -p 8000:8000 lalwanigunjan/sc2 python /src/scrappingDataHtml.py 00
01652044 0001652044-17-000008
https://www.sec.gov/Archives/edgar/data/1652044/000165204417000008/goog10-kq4201
6.htm
//goog10-kq42016/goog10-kq42016-1.csv
//goog10-kq42016/goog10-kq42016-2.csv
//goog10-kq42016/goog10-kq42016-3.csv
//goog10-kq42016/goog10-kq42016-4.csv
//goog10-kq42016/goog10-kq42016-5.csv
//goog10-kq42016/goog10-kq42016-6.csv
//goog10-kq42016/goog10-kq42016-7.csv
//goog10-kq42016/goog10-kq42016-8.csv
//goog10-kq42016/goog10-kq42016-9.csv
//goog10-kq42016/goog10-kq42016-10.csv
//goog10-kq42016/goog10-kq42016-11.csv
//goog10-kq42016/goog10-kq42016-12.csv
//goog10-kq42016/goog10-kq42016-13.csv
//goog10-kq42016/goog10-kq42016-14.csv
//goog10-kq42016/goog10-kq42016-15.csv
//goog10-kq42016/goog10-kq42016-16.csv
//goog10-kq42016/goog10-kq42016-17.csv
```

### SECTION 3 OUTPUT DISCUSSION

**How do you handle exceptions when you don't find the CIK/accession number or if the amazon keys aren't valid?**

In case, User did not provide the CIK/accession number, then the program will run for the default CIK/accession number which is this case is IBM.

In case, User provide the invalid CIK/accession number than the program will handle the error by displaying an error message ("URL Invalid"). Also log files are generated to capture each and every step that is executed by the program. In case of any issue or error, user can look into the log file to find the exact failure point.

The above rule also applies when the user provides with the invalid amazon keys.