Advance Data Science

Data Wrangling Using Edgar Data

Assignment – 1 Part - 1

Report by:

Ankit Bhayani

Rajat Agrawal

Vaisakha Sawant

OBJECTIVE

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

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 FILE

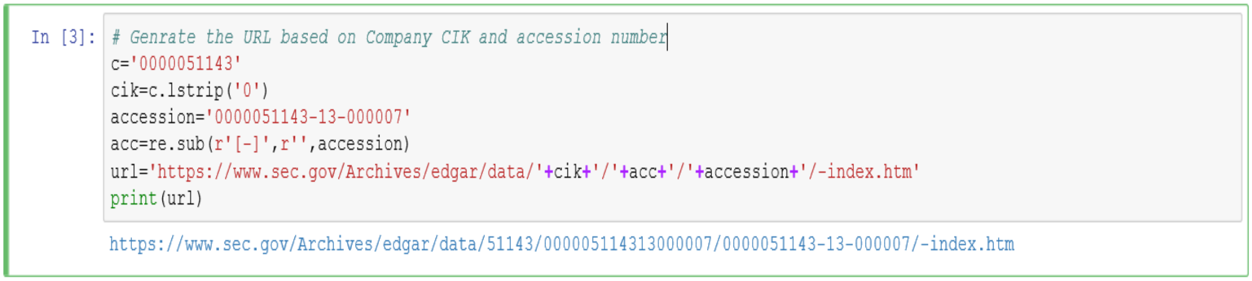
STEP 1:

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.



```
In [3]: # Genrate the URL based on Company CIK and accession number
        c='0000051143'
        cik=c.lstrip('0')
        accession='0000051143-13-000007'
        acc=re.sub(r'[-]',r'',accession)
        url='https://www.sec.gov/Archives/edgar/data/'+cik+'/'+acc+'/'+accession+'/-index.htm'
        print(url)

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

STEP 2:

CREATE & ACCESS THE EDGAR URL BASED ON PARAMETER PROVIDED

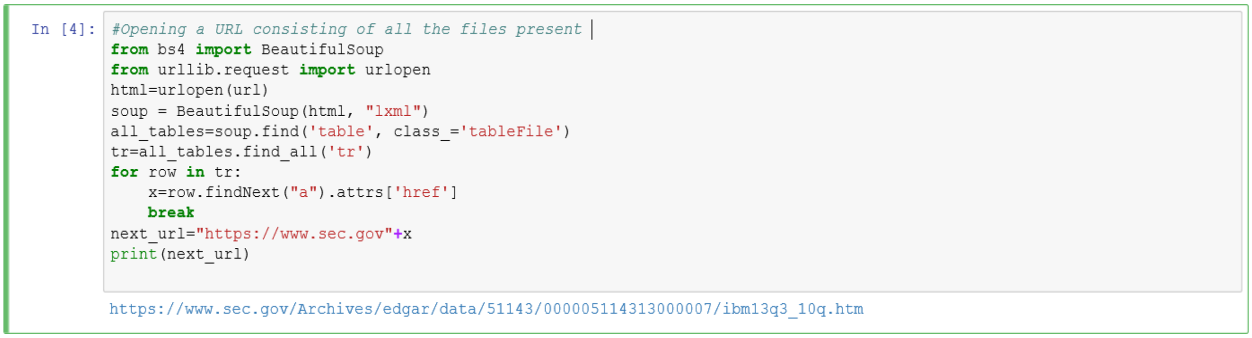
Since we are using python 3.2 for accessing the URL, we have to import the following python library to access the Edgar website.

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.



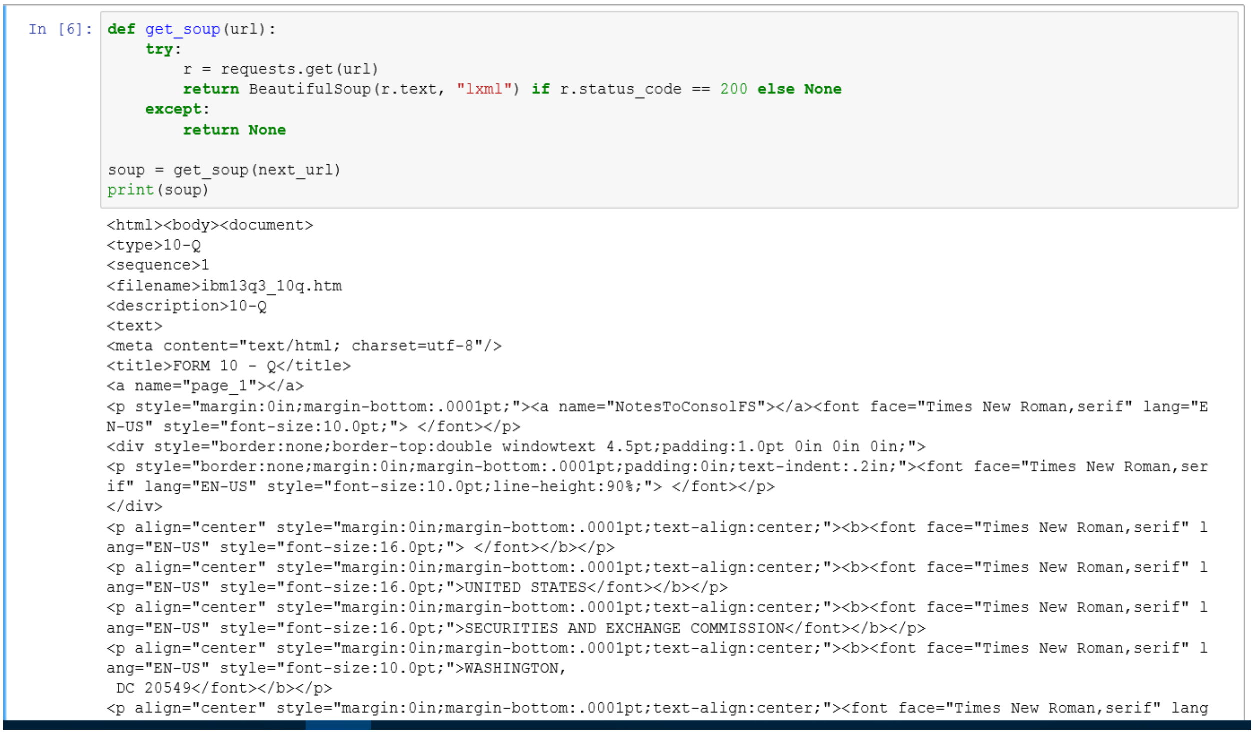
```
In [4]: #Opening a URL consisting of all the files present
        from bs4 import BeautifulSoup
        from urllib.request import urlopen
        html=urlopen(url)
        soup = BeautifulSoup(html, "lxml")
        all_tables=soup.find('table', class_='tableFile')
        tr=all_tables.find_all('tr')
        for row in tr:
            x=row.findNext("a").attrs['href']
            break
        next_url="https://www.sec.gov"+x
        print(next_url)

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

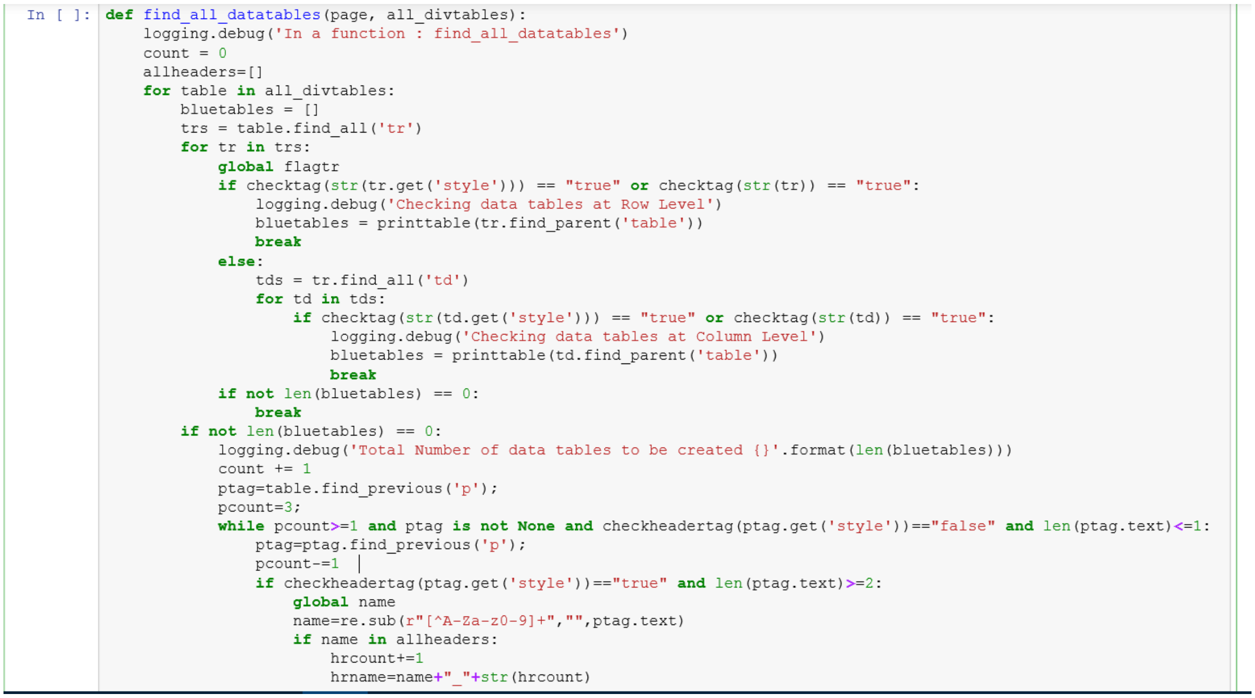


```
In [6]: def get_soup(url):
            try:
                r = requests.get(url)
                return BeautifulSoup(r.text, "lxml") if r.status_code == 200 else None
            except:
                return None

        soup = get_soup(next_url)
        print(soup)

        <html><body><document>
        <type>10-Q
        <sequence>1
        <filename>ibm13q3_10q.htm
        <description>10-Q
        <text>
        <meta content="text/html; charset=utf-8"/>
        <title>FORM 10 - Q</title>
        <a name="page_1"></a>
        <p style="margin:0in;margin-bottom:.0001pt;"><a name="NotesToConsolFS"></a><font face="Times New Roman,serif" lang="E
        N-US" style="font-size:10.0pt;"> </font></p>
        <div style="border:none;border-top:double windowtext 4.5pt;padding:1.0pt 0in 0in 0in;">
        <p style="border:none;margin:0in;margin-bottom:.0001pt;padding:0in;text-indent:.2in;"><font face="Times New Roman,ser
        if" lang="EN-US" style="font-size:10.0pt;line-height:90%;"> </font></p>
        </div>
        <p align="center" style="margin:0in;margin-bottom:.0001pt;text-align:center;"><b><font face="Times New Roman,serif" l
        ang="EN-US" style="font-size:16.0pt;"> </font></b></p>
        <p align="center" style="margin:0in;margin-bottom:.0001pt;text-align:center;"><b><font face="Times New Roman,serif" l
        ang="EN-US" style="font-size:16.0pt;">UNITED STATES</font></b></p>
        <p align="center" style="margin:0in;margin-bottom:.0001pt;text-align:center;"><b><font face="Times New Roman,serif" l
        ang="EN-US" style="font-size:16.0pt;">SECURITIES AND EXCHANGE COMMISSION</font></b></p>
        <p align="center" style="margin:0in;margin-bottom:.0001pt;text-align:center;"><b><font face="Times New Roman,serif" l
        ang="EN-US" style="font-size:10.0pt;">WASHINGTON,
         DC 20549</font></b></p>
        <p align="center" style="margin:0in;margin-bottom:.0001pt;text-align:center;"><font face="Times New Roman,serif" lang
```

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.

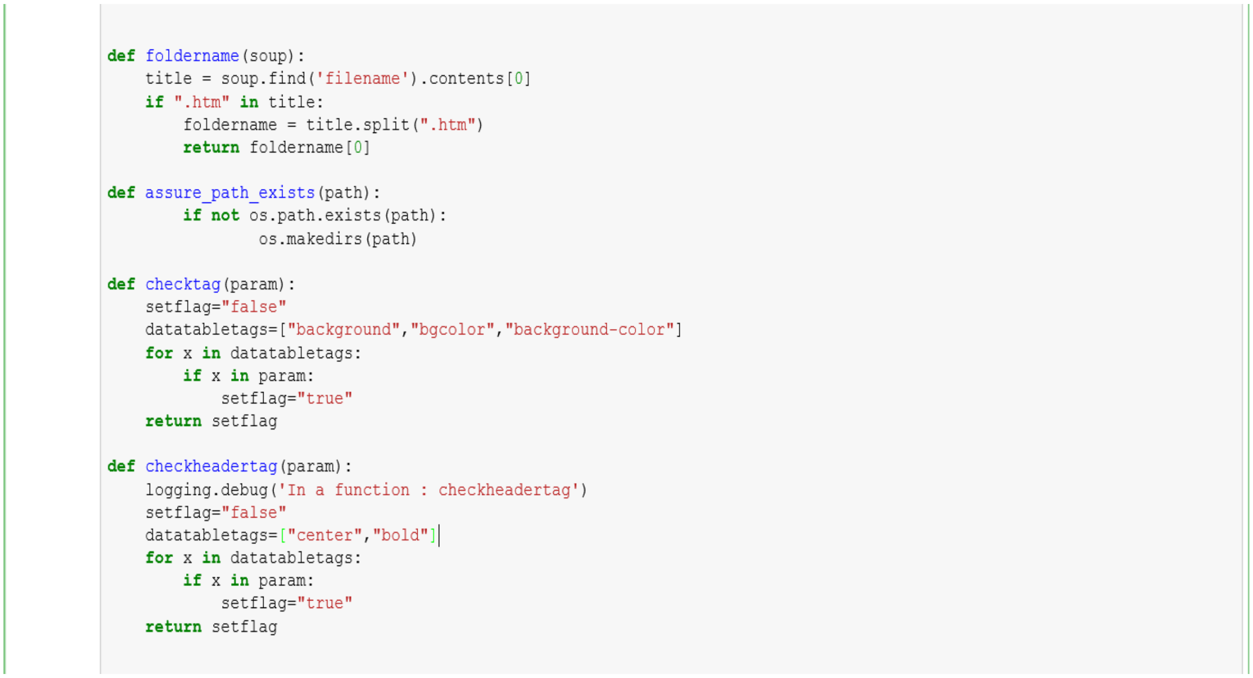


```
In [ ]: def find_all_datatables(page, all_divtables):
            logging.debug('In a function : find_all_datatables')
            count = 0
            allheaders=[]
            for table in all_divtables:
                bluetables = []
                trs = table.find_all('tr')
                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');
                    pcount=3;
                    while pcount>=1 and ptag is not None and checkheadertag(ptag.get('style'))=="false" and len(ptag.text)<=1:
                        ptag=ptag.find_previous('p');
                        pcount-=1
                        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
                                hrname=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.

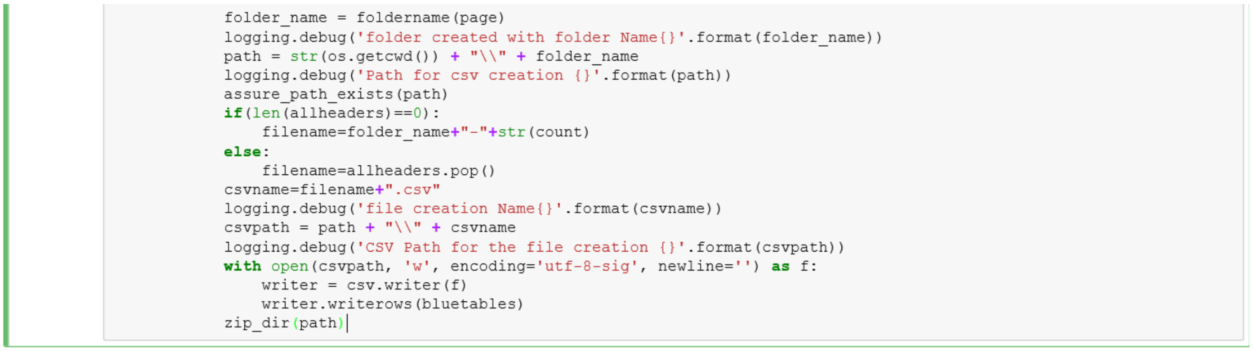


```
def foldername(soup):
    title = soup.find('filename').contents[0]
    if ".htm" in title:
        foldername = title.split(".htm")
        return foldername[0]

def assure_path_exists(path):
        if not os.path.exists(path):
                os.makedirs(path)

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

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

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.

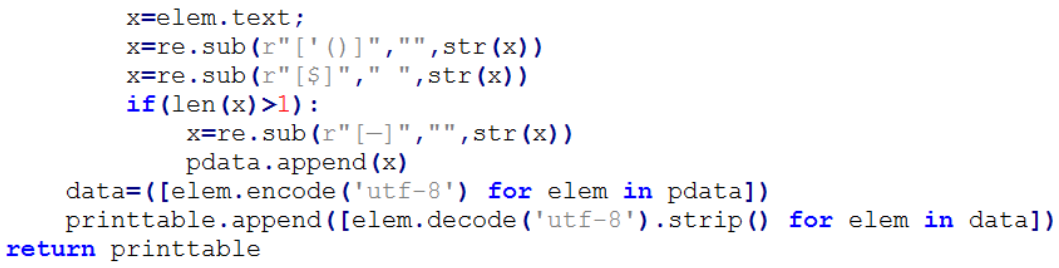


```
folder_name = foldername(page)
logging.debug('folder created with folder Name{}'.format(folder_name))
path = str(os.getcwd()) + "\\" + folder_name
logging.debug('Path for csv creation {}'.format(path))
assure_path_exists(path)
if(len(allheaders)==0):
    filename=folder_name+"-"+str(count)
else:
    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(bluetables)
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.

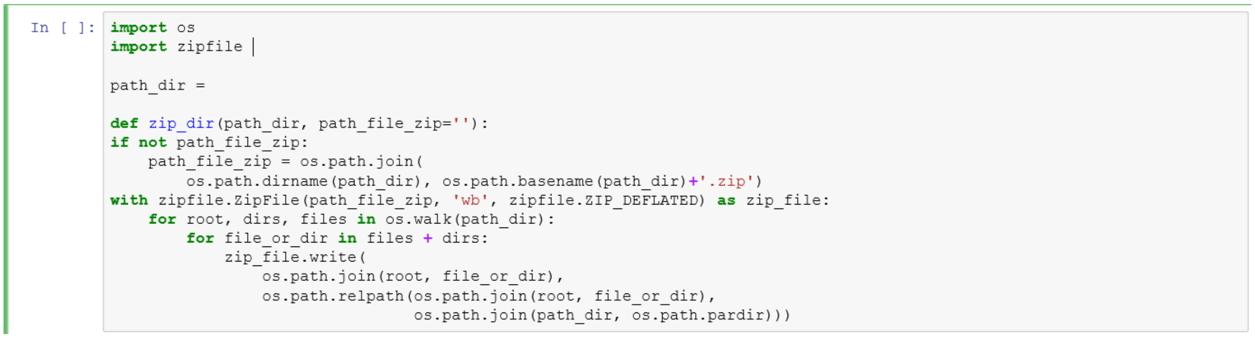


```
        x=elem.text;
        x=re.sub(r"['()]","",str(x))
        x=re.sub(r"[$]"," ",str(x))
        if(len(x)>1):
            x=re.sub(r"[—]","",str(x))
            pdata.append(x)
    data=([elem.encode('utf-8') for elem in pdata])
    printtable.append([elem.decode('utf-8').strip() for elem in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

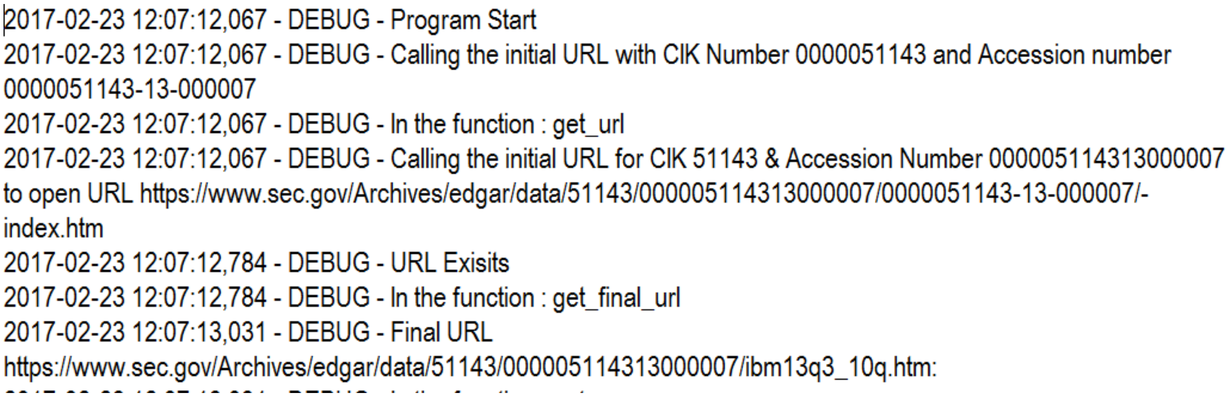


```
In [ ]: import os
        import zipfile

        path_dir =

        def zip_dir(path_dir, path_file_zip=''):
        if not path_file_zip:
            path_file_zip = os.path.join(
                os.path.dirname(path_dir), os.path.basename(path_dir)+'.zip')
        with zipfile.ZipFile(path_file_zip, 'wb', 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.



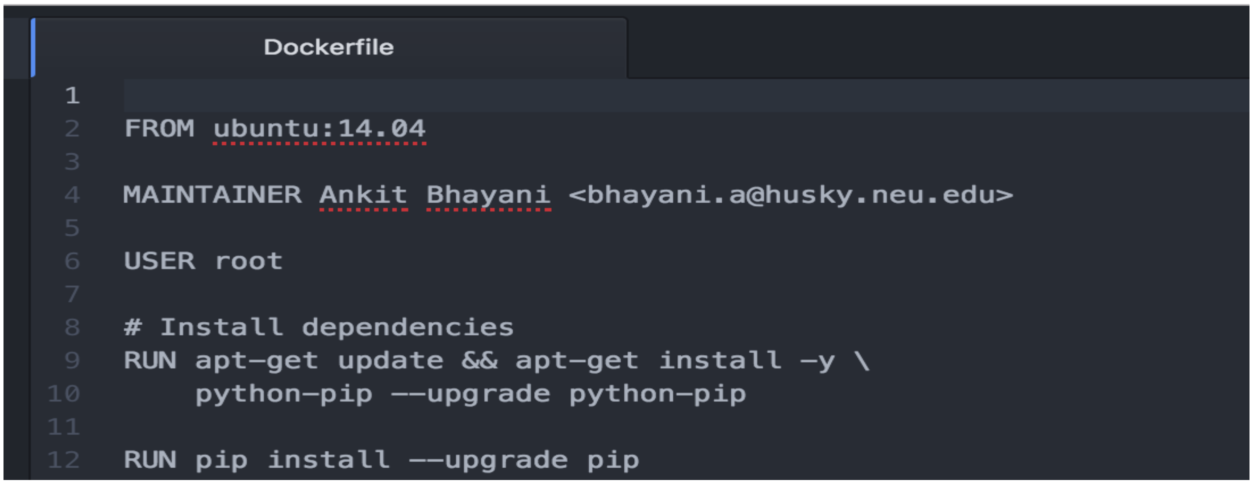
```
2017-02-23 12:07:12,067 - DEBUG - Program Start
2017-02-23 12:07:12,067 - DEBUG - Calling the initial URL with CIK Number 0000051143 and Accession number
0000051143-13-000007
2017-02-23 12:07:12,067 - DEBUG - In the function : get_url
2017-02-23 12:07:12,067 - DEBUG - Calling the initial URL for CIK 51143 & Accession Number 000005114313000007
to open URL https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007/-
index.htm
2017-02-23 12:07:12,784 - DEBUG - URL Exisits
2017-02-23 12:07:12,784 - DEBUG - In the function : get_final_url
2017-02-23 12:07:13,031 - DEBUG - Final URL
https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm:
```

SECTION - 2 DOCKERIZE PART 1

STEP 1:

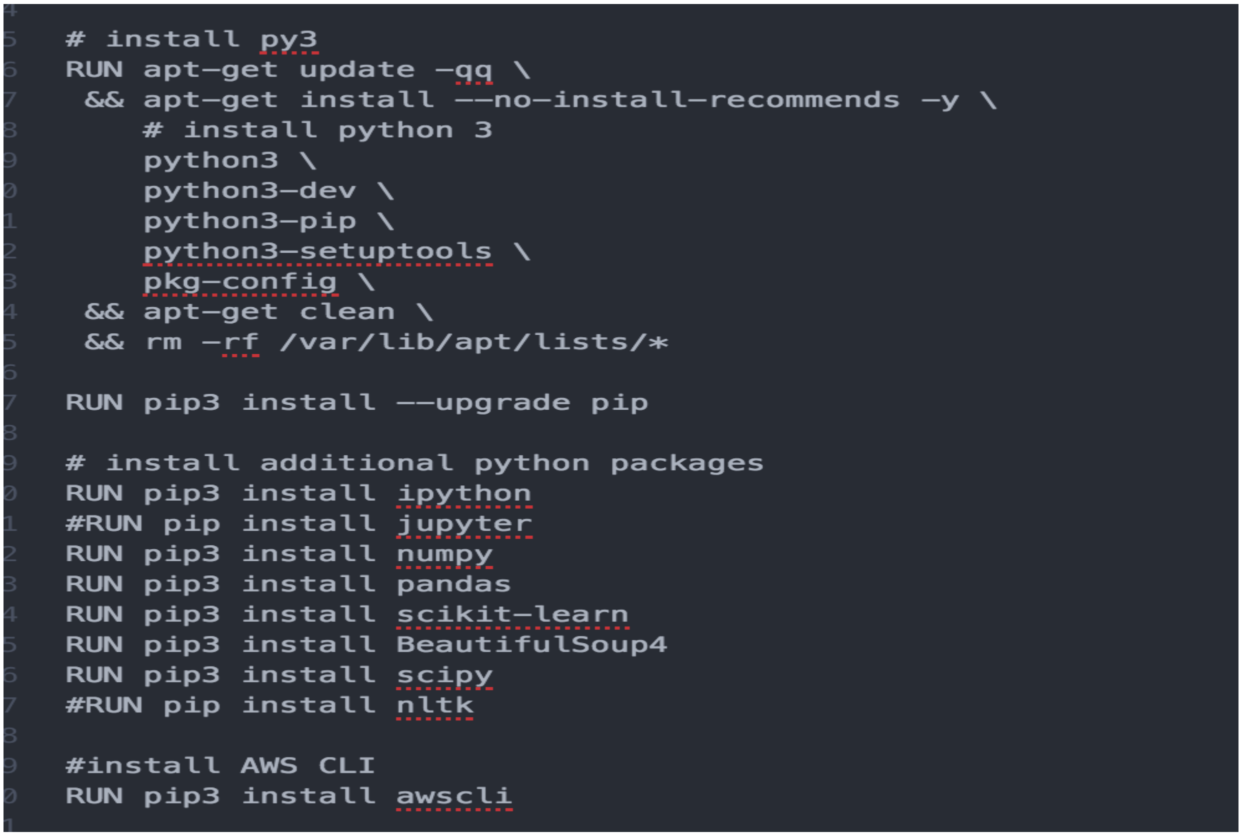
CREATE THE DOCKER IMAGE FROM DOCKERFILE

1.1 After taking the base UBUNTU 14.04 image.



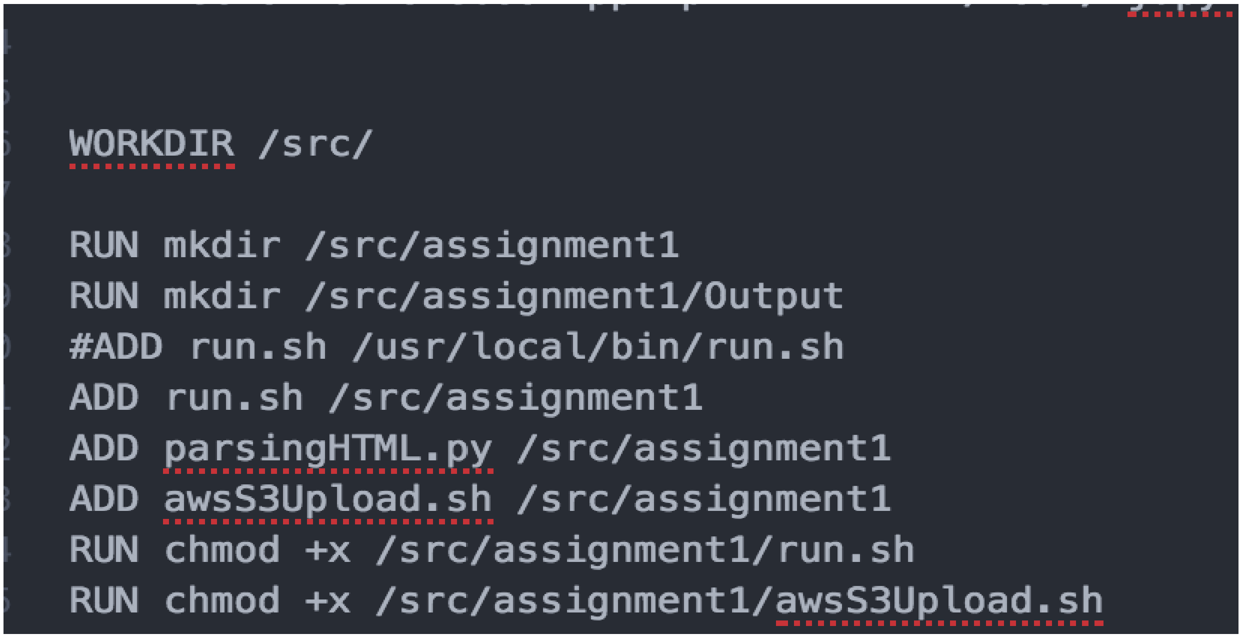
```

FROM ubuntu:14.04

MAINTAINER Ankit Bhayani <bhayani.a@husky.neu.edu>

USER root

# Install dependencies
RUN apt-get update && apt-get install -y \
    python-pip --upgrade python-pip

RUN pip install --upgrade pip
```

1.2 Install Python and all required packages which is used in source code



```
# install py3
RUN apt-get update -qq \
 && apt-get install --no-install-recommends -y \
    # install python 3
    python3 \
    python3-dev \
    python3-pip \
    python3-setuptools \
    pkg-config \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

RUN pip3 install --upgrade pip

# install additional python packages
RUN pip3 install ipython
#RUN pip install jupyter
RUN pip3 install numpy
RUN pip3 install pandas
RUN pip3 install scikit-learn
RUN pip3 install BeautifulSoup4
RUN pip3 install scipy
#RUN pip install nltk

#install AWS CLI
RUN pip3 install awscli
```

1.3 While building DockerFile; First creating the work directory and then copying the source python and shell files from local machine to DOCKER image.

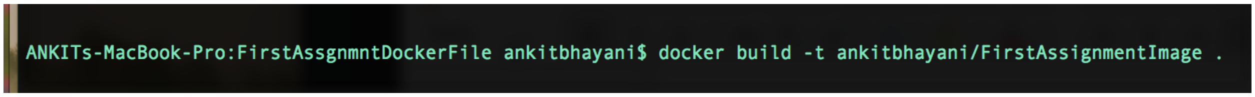


```
WORKDIR /src/

RUN mkdir /src/assignment1
RUN mkdir /src/assignment1/Output
#ADD run.sh /usr/local/bin/run.sh
ADD run.sh /src/assignment1
ADD parsingHTML.py /src/assignment1
ADD awsS3Upload.sh /src/assignment1
RUN chmod +x /src/assignment1/run.sh
RUN chmod +x /src/assignment1/awsS3Upload.sh
```

STEP 2:

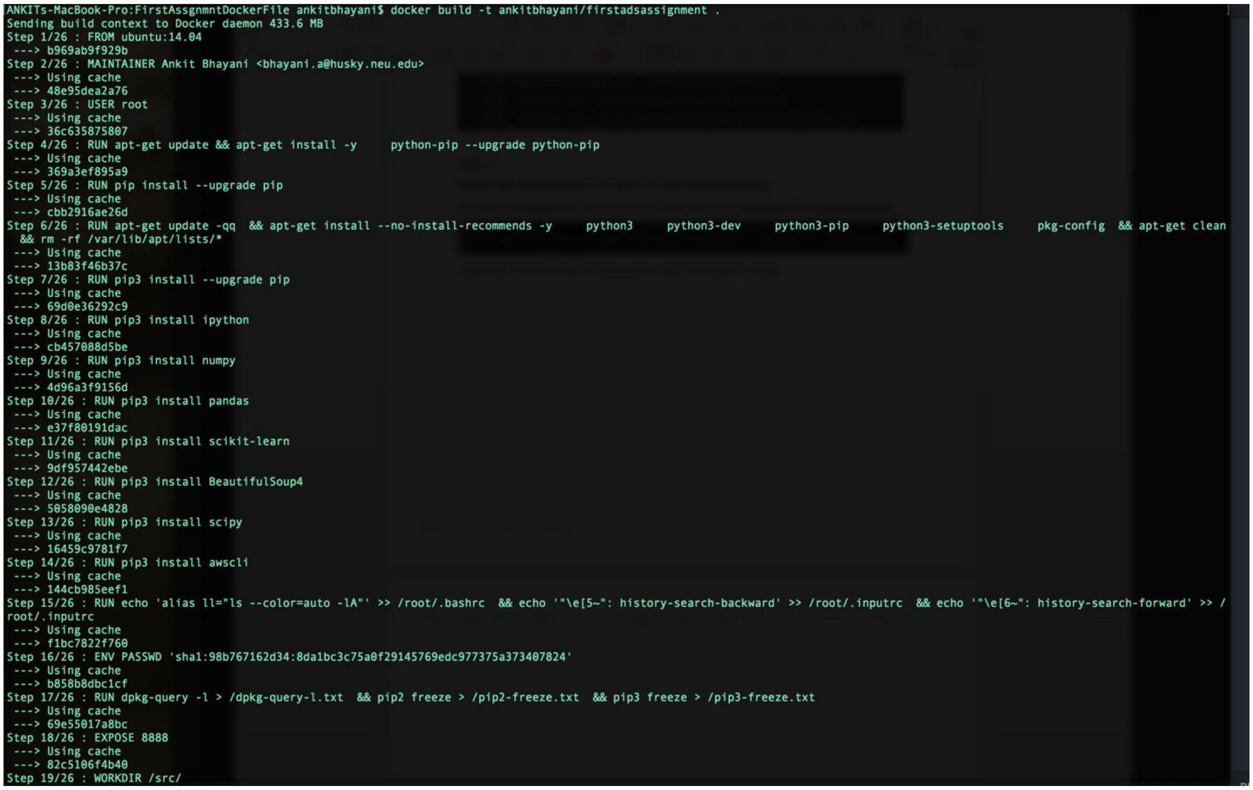
BUILD THE DOCKERFILE TO CREATE THE DOCKER IMAGE

Building this dockerfile will create a DOCKER image with name ankitbhayani/FirstAssignmentImage

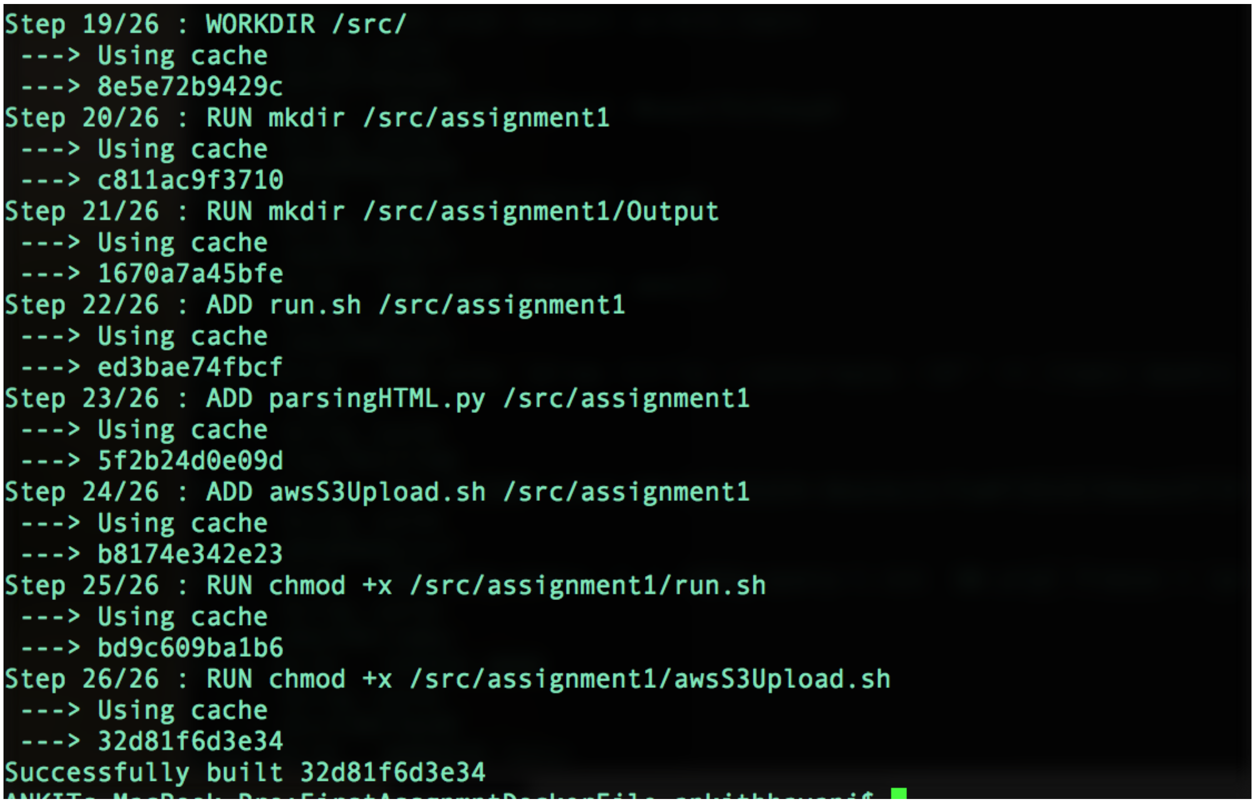


```
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$ docker build -t ankitbhayani/FirstAssignmentImage .
```

Each step in this file will be executed in order to create an image. AWS CLI is also integrated inside the dockerfile



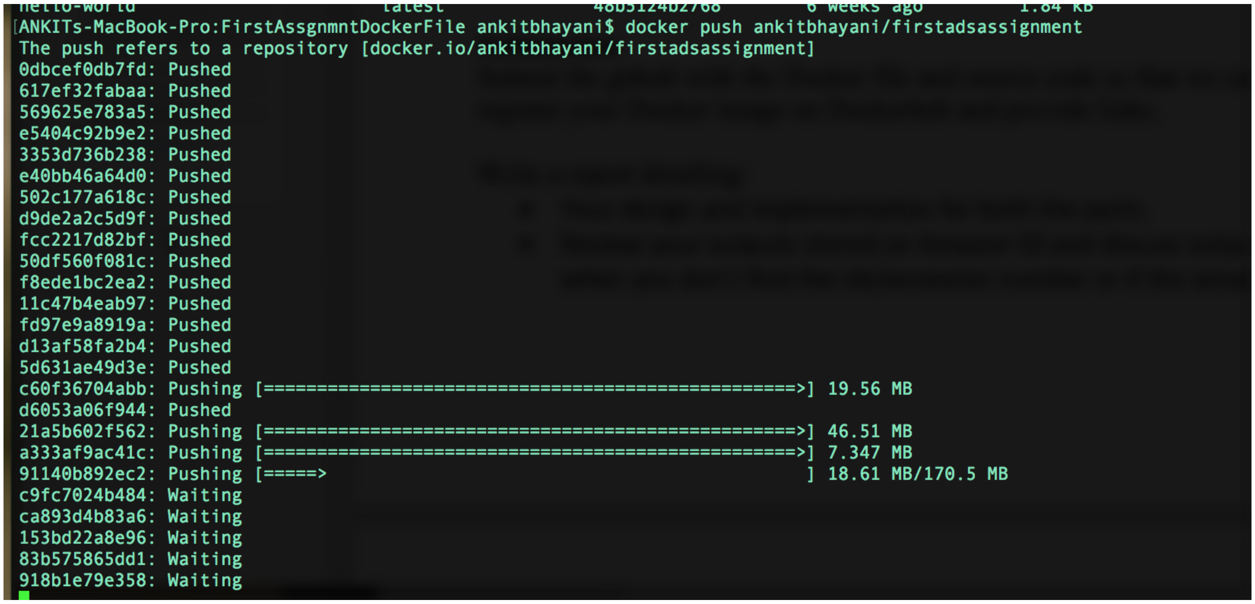
```
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$ docker build -t ankitbhayani/firstadsassignment .
Sending build context to Docker daemon 433.6 MB
Step 1/26 : FROM ubuntu:14.04
 ---> b969ab9f929b
Step 2/26 : MAINTAINER Ankit Bhayani <bhayani.a@husky.neu.edu>
 ---> Using cache
 ---> 48e95dea2a76
Step 3/26 : USER root
 ---> Using cache
 ---> 36c635875807
Step 4/26 : RUN apt-get update && apt-get install -y     python-pip --upgrade python-pip
 ---> Using cache
 ---> 369a3ef895a9
Step 5/26 : RUN pip install --upgrade pip
 ---> Using cache
 ---> cbb2916ae26d
Step 6/26 : RUN apt-get update -qq  && apt-get install --no-install-recommends -y     python3     python3-dev     python3-pip     python3-setuptools     pkg-config  && apt-get clean
  && rm -rf /var/lib/apt/lists/*
 ---> Using cache
 ---> 13b83f46b37c
Step 7/26 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> 69d0e36292c9
Step 8/26 : RUN pip3 install ipython
 ---> Using cache
 ---> cb457088d5be
Step 9/26 : RUN pip3 install numpy
 ---> Using cache
 ---> 4d96a3f9156d
Step 10/26 : RUN pip3 install pandas
 ---> Using cache
 ---> e37f80191dac
Step 11/26 : RUN pip3 install scikit-learn
 ---> Using cache
 ---> 9df957442ebe
Step 12/26 : RUN pip3 install BeautifulSoup4
 ---> Using cache
 ---> 5058090e4828
Step 13/26 : RUN pip3 install scipy
 ---> Using cache
 ---> 16459c9781f7
Step 14/26 : RUN pip3 install awscli
 ---> Using cache
 ---> 144cb985eef1
Step 15/26 : RUN echo 'alias ll="ls --color=auto -lA"' >> /root/.bashrc  && echo '"\e[5~": history-search-backward' >> /root/.inputrc  && echo '"\e[6~": history-search-forward' >> /
root/.inputrc
 ---> Using cache
 ---> f1bc7822f760
Step 16/26 : ENV PASSWD 'sha1:98b767162d34:8da1bc3c75a0f29145769edc977375a373407824'
 ---> Using cache
 ---> b858b8dbc1cf
Step 17/26 : RUN dpkg-query -l > /dpkg-query-l.txt  && pip2 freeze > /pip2-freeze.txt  && pip3 freeze > /pip3-freeze.txt
 ---> Using cache
 ---> 69e55017a8bc
Step 18/26 : EXPOSE 8888
 ---> Using cache
 ---> 82c5106f4b40
Step 19/26 : WORKDIR /src/
```



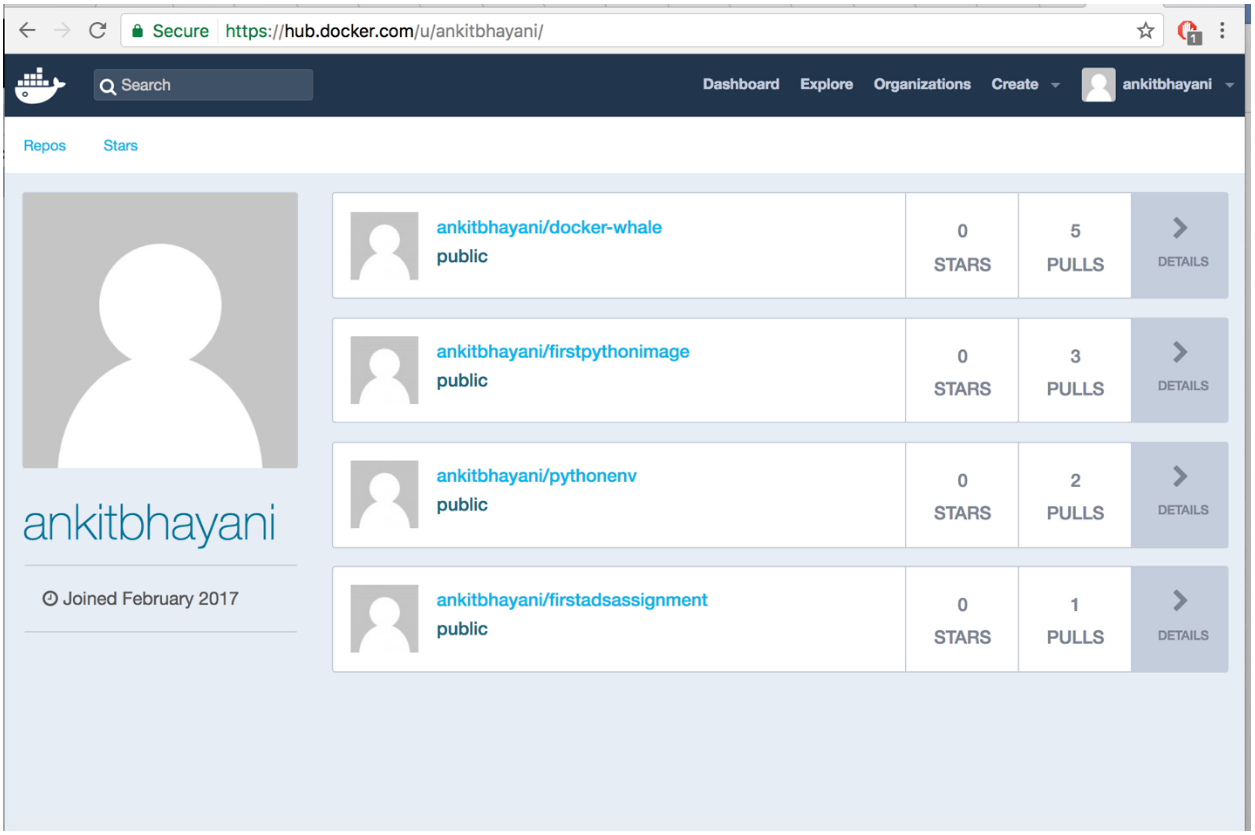
```
Step 19/26 : WORKDIR /src/
 ---> Using cache
 ---> 8e5e72b9429c
Step 20/26 : RUN mkdir /src/assignment1
 ---> Using cache
 ---> c811ac9f3710
Step 21/26 : RUN mkdir /src/assignment1/Output
 ---> Using cache
 ---> 1670a7a45bfe
Step 22/26 : ADD run.sh /src/assignment1
 ---> Using cache
 ---> ed3bae74fbcf
Step 23/26 : ADD parsingHTML.py /src/assignment1
 ---> Using cache
 ---> 5f2b24d0e09d
Step 24/26 : ADD awsS3Upload.sh /src/assignment1
 ---> Using cache
 ---> b8174e342e23
Step 25/26 : RUN chmod +x /src/assignment1/run.sh
 ---> Using cache
 ---> bd9c609ba1b6
Step 26/26 : RUN chmod +x /src/assignment1/awsS3Upload.sh
 ---> Using cache
 ---> 32d81f6d3e34
Successfully built 32d81f6d3e34
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$
```

STEP 3:

PUSH THIS DOCKER IMAGE to DOCKERHUB



```
 hello-world                    latest              48b5124b2768        6 weeks ago         1.84 kB
[ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$ docker push ankitbhayani/firstadsassignment
 The push refers to a repository [docker.io/ankitbhayani/firstadsassignment]
 0dbcef0db7fd: Pushed
 617ef32fabaa: Pushed
 569625e783a5: Pushed
 e5404c92b9e2: Pushed
 3353d736b238: Pushed
 e40bb46a64d0: Pushed
 502c177a618c: Pushed
 d9de2a2c5d9f: Pushed
 fcc2217d82bf: Pushed
 50df560f081c: Pushed
 f8ede1bc2ea2: Pushed
 11c47b4eab97: Pushed
 fd97e9a8919a: Pushed
 d13af58fa2b4: Pushed
 5d631ae49d3e: Pushed
 c60f36704abb: Pushing [==================================================>] 19.56 MB
 d6053a06f944: Pushed
 21a5b602f562: Pushing [==================================================>] 46.51 MB
 a333af9ac41c: Pushing [==================================================>] 7.347 MB
 91140b892ec2: Pushing [=====>                                             ] 18.61 MB/170.5 MB
 c9fc7024b484: Waiting
 ca893d4b83a6: Waiting
 153bd22a8e96: Waiting
 83b575865dd1: Waiting
 918b1e79e358: Waiting
```



Secure https://hub.docker.com/u/ankitbhayani/

ankitbhayani

Joined February 2017

ankitbhayani/docker-whale
public
0 STARS
5 PULLS
DETAILS

ankitbhayani/firstpythonimage
public
0 STARS
3 PULLS
DETAILS

ankitbhayani/pythonenv
public
0 STARS
2 PULLS
DETAILS

ankitbhayani/firstadsassignment
public
0 STARS
1 PULLS
DETAILS

STEP 4:

RUN THE DOCKER IMAGE

Command:

docker run -e ACCESS_KEY=**<YOUR_ACCESS_KEY>**

-e SECRET_KEY=**<YOUR_SECRET_KEY>**

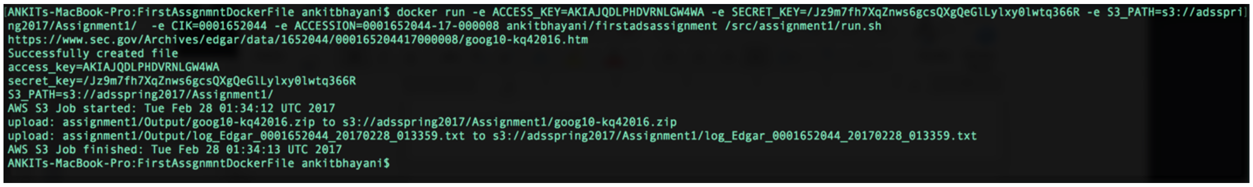
-e S3_PATH=**<YOUR_S3_PATH>**

-e CIK=**0001652044**

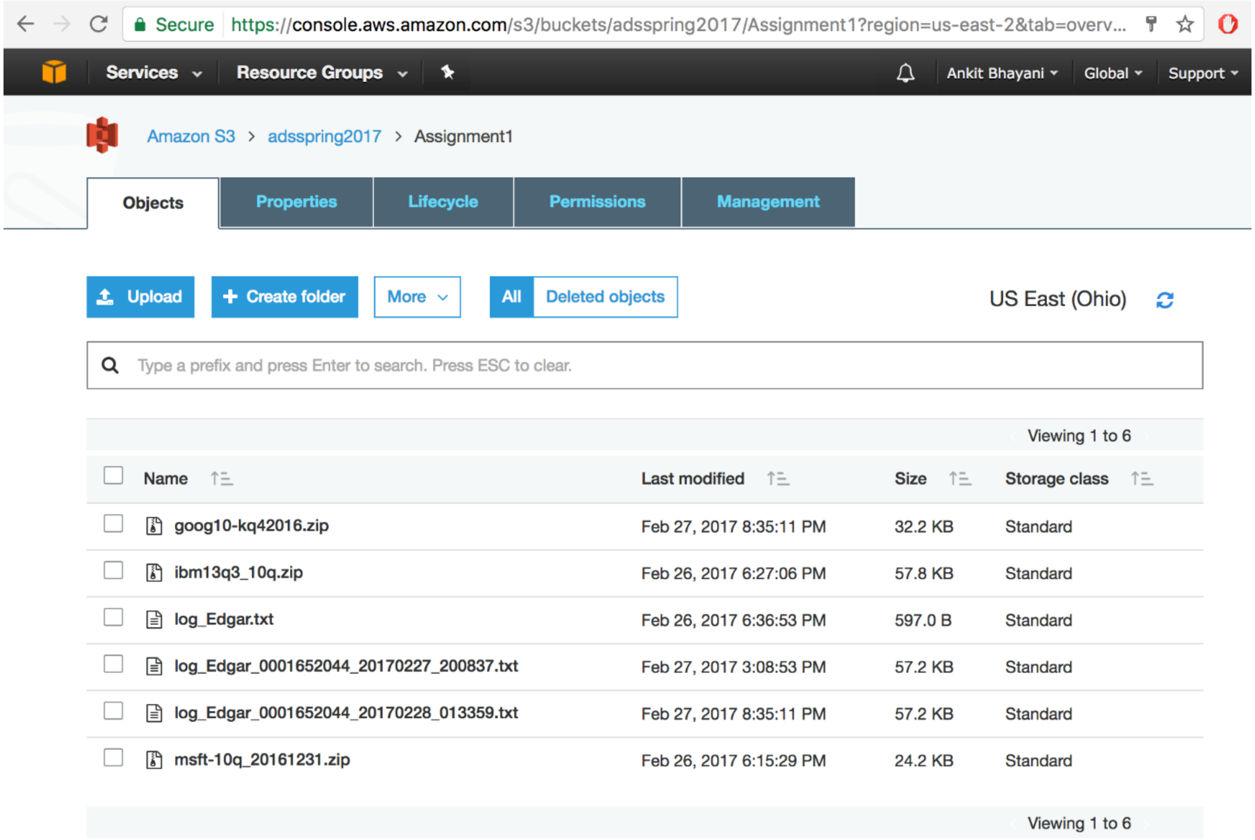
-e ACCESSION=**0001652044-17-000008**

ankitbhayani/firstadsassignment

/src/assignment1/run.sh



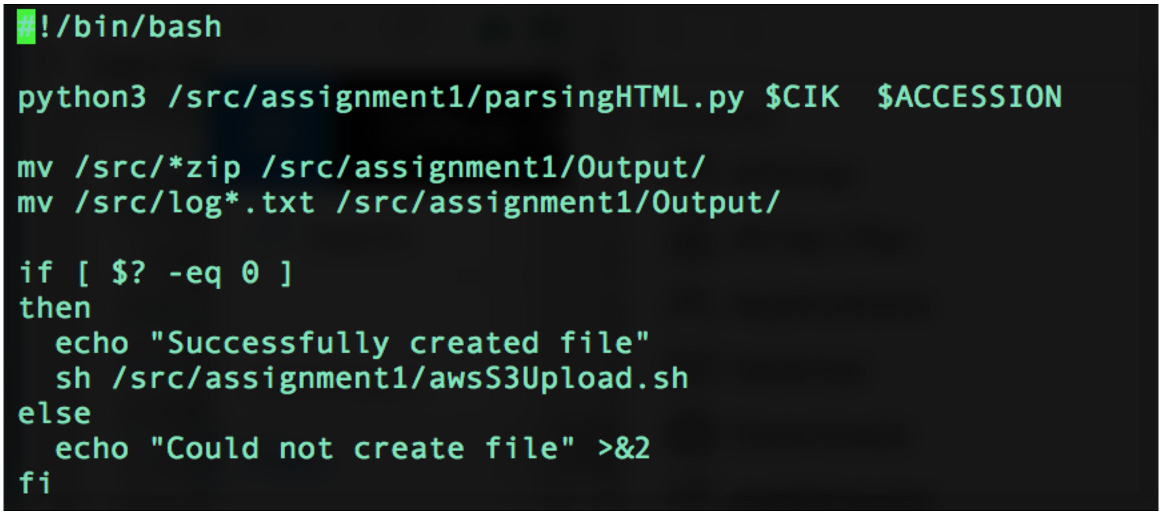
```
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$ docker run -e ACCESS_KEY=AKIAJQDLPHDVRNLGW4WA -e SECRET_KEY=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 -e S3_PATH=s3://adsspri
ng2017/Assignment1/  -e CIK=0001652044 -e ACCESSION=0001652044-17-000008 ankitbhayani/firstadsassignment /src/assignment1/run.sh
https://www.sec.gov/Archives/edgar/data/1652044/000165204417000008/goog10-kq42016.htm
Successfully created file
access_key=AKIAJQDLPHDVRNLGW4WA
secret_key=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
S3_PATH=s3://adsspring2017/Assignment1/
AWS S3 Job started: Tue Feb 28 01:34:12 UTC 2017
upload: assignment1/Output/goog10-kq42016.zip to s3://adsspring2017/Assignment1/goog10-kq42016.zip
upload: assignment1/Output/log_Edgar_0001652044_20170228_013359.txt to s3://adsspring2017/Assignment1/log_Edgar_0001652044_20170228_013359.txt
AWS S3 Job finished: Tue Feb 28 01:34:13 UTC 2017
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$
```



Secure
https://console.aws.amazon.com/s3/buckets/adsspring2017/Assignment1?region=us-east-2&tab=overv...
Services
Resource Groups
Ankit Bhayani
Global
Support
Amazon S3 > adsspring2017 > Assignment1
Objects
Properties
Lifecycle
Permissions
Management
Upload
Create folder
More
All
Deleted objects
US East (Ohio)
Type a prefix and press Enter to search. Press ESC to clear.
Viewing 1 to 6

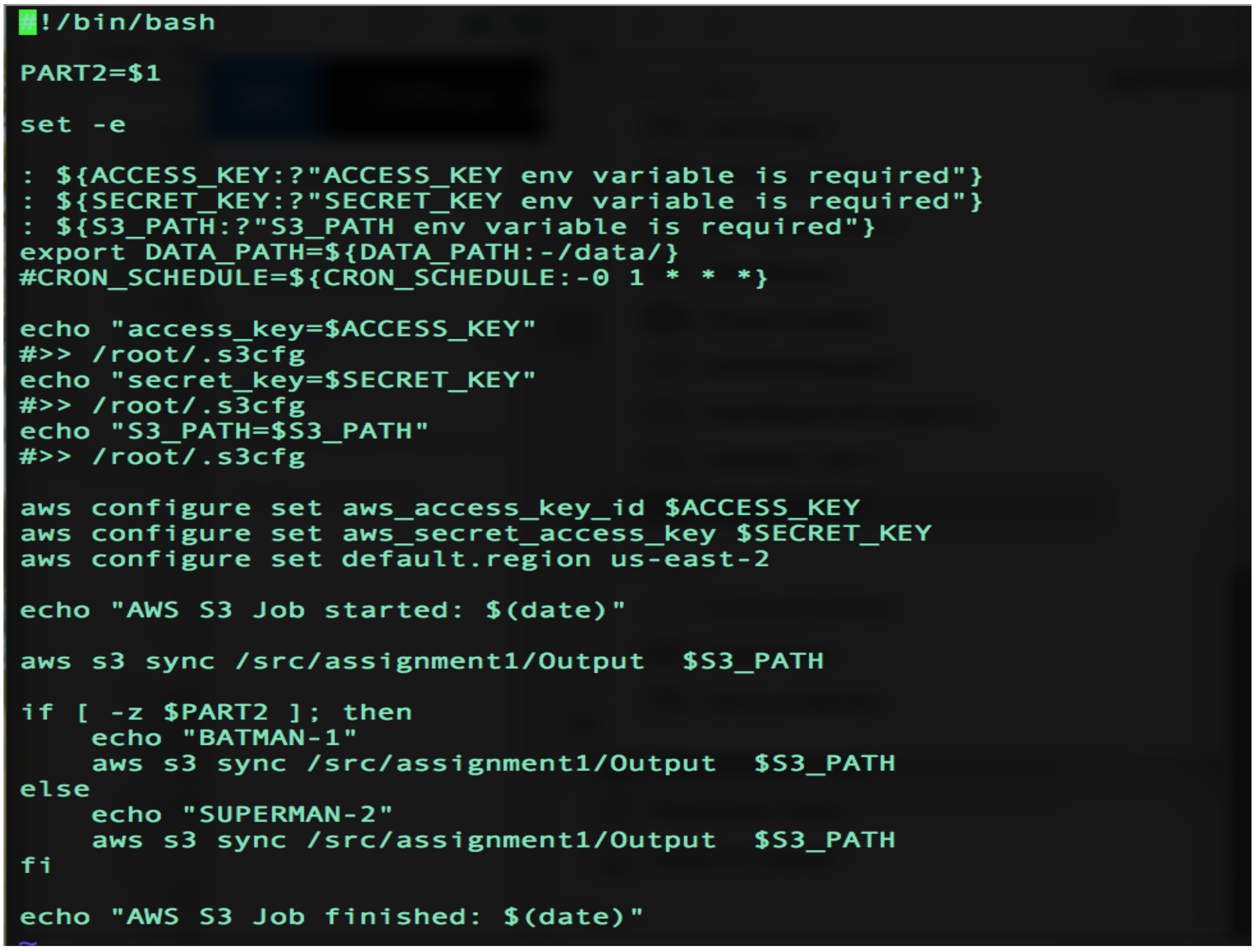
□ Name	Last modified	Size	Storage class
□ goog10-kq42016.zip	Feb 27, 2017 8:35:11 PM	32.2 KB	Standard
□ ibm13q3_10q.zip	Feb 26, 2017 6:27:06 PM	57.8 KB	Standard
□ log_Edgar.txt	Feb 26, 2017 6:36:53 PM	597.0 B	Standard
□ log_Edgar_0001652044_20170227_200837.txt	Feb 27, 2017 3:08:53 PM	57.2 KB	Standard
□ log_Edgar_0001652044_20170228_013359.txt	Feb 27, 2017 8:35:11 PM	57.2 KB	Standard
□ msft-10q_20161231.zip	Feb 26, 2017 6:15:29 PM	24.2 KB	Standard

DRIVER CODE HANDLING THROUGH SHELL FILE:



```
#!/bin/bash

python3 /src/assignment1/parsingHTML.py $CIK  $ACCESSION

mv /src/*zip /src/assignment1/Output/
mv /src/log*.txt /src/assignment1/Output/

if [ $? -eq 0 ]
then
  echo "Successfully created file"
  sh /src/assignment1/awsS3Upload.sh
else
  echo "Could not create file" >&2
fi
```

AWS S3 UPLOAD SHELL FILE:



```
#!/bin/bash

PART2=$1

set -e

: ${ACCESS_KEY:?"ACCESS_KEY env variable is required"}
: ${SECRET_KEY:?"SECRET_KEY env variable is required"}
: ${S3_PATH:?"S3_PATH env variable is required"}
export DATA_PATH=${DATA_PATH:-/data/}
#CRON_SCHEDULE=${CRON_SCHEDULE:-0 1 * * *}

echo "access_key=$ACCESS_KEY"
#>> /root/.s3cfg
echo "secret_key=$SECRET_KEY"
#>> /root/.s3cfg
echo "S3_PATH=$S3_PATH"
#>> /root/.s3cfg

aws configure set aws_access_key_id $ACCESS_KEY
aws configure set aws_secret_access_key $SECRET_KEY
aws configure set default.region us-east-2

echo "AWS S3 Job started: $(date)"

aws s3 sync /src/assignment1/Output  $S3_PATH

if [ -z $PART2 ]; then
    echo "BATMAN-1"
    aws s3 sync /src/assignment1/Output  $S3_PATH
else
    echo "SUPERMAN-2"
    aws s3 sync /src/assignment1/Output  $S3_PATH
fi

echo "AWS S3 Job finished: $(date)"
```

SECTION - 3 DISCUSS OUTPUT

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.

Review your outputs stored on Amazon S3 and discuss outputs

Output at Amazon S3 will consist of one zip file which will consist of all the csv files which are generated by our process through Docker. With this zip folder, there will be one log file which contains all the process/function/error occurred during our execution of program.