Advances in Data sciences Case 1

Under the guidance of Professor Srikanth Krishnamurthy

Team 6: Vaidehi Deshpande,
Puneet Kumar,
Yamini Sehrawat

Table of Contents

Working with Edgar datasets

Abstract

EDGAR, the Electronic Data Gathering, Analysis, and Retrieval system, performs automated collection, validation, indexing, acceptance, and forwarding of submissions by companies and others who are required by law to file forms with the U.S. Securities and Exchange Commission (the "SEC").
The database is freely available to the public via the Internet (Web or FTP).

EDGAR Database of SEC Filings
Securities and Exchange Commission (SEC) EDGAR database which contains regulatory filings from publicly-traded US corporations.

Each company in EDGAR gets an identifier known as the CIK which is a 10-digit number. We can find the CIK by searching EDGAR using a name of stock market ticker.

For example, searching for IBM by ticker shows us that the CIK is 0000051143.

Note that the leading zeroes are often omitted (e.g. in the ftp access) so this would be come 51143.

Next each submission receives an 'Accession Number' (acc-no). For example, IBM's quarterly financial filing (form 10-Q) in October 2013 had accession number: 0000051143-13-000007.

Problem statement

Problem 1
Part 1: Parse Files

We have been provided with the link https://datahub.io/dataset/edgar to get information about how to access data from Edgar.
Our goal is to extract tables from 10Q (Quarterly report pursuant to Section 13) filings using R/Python.

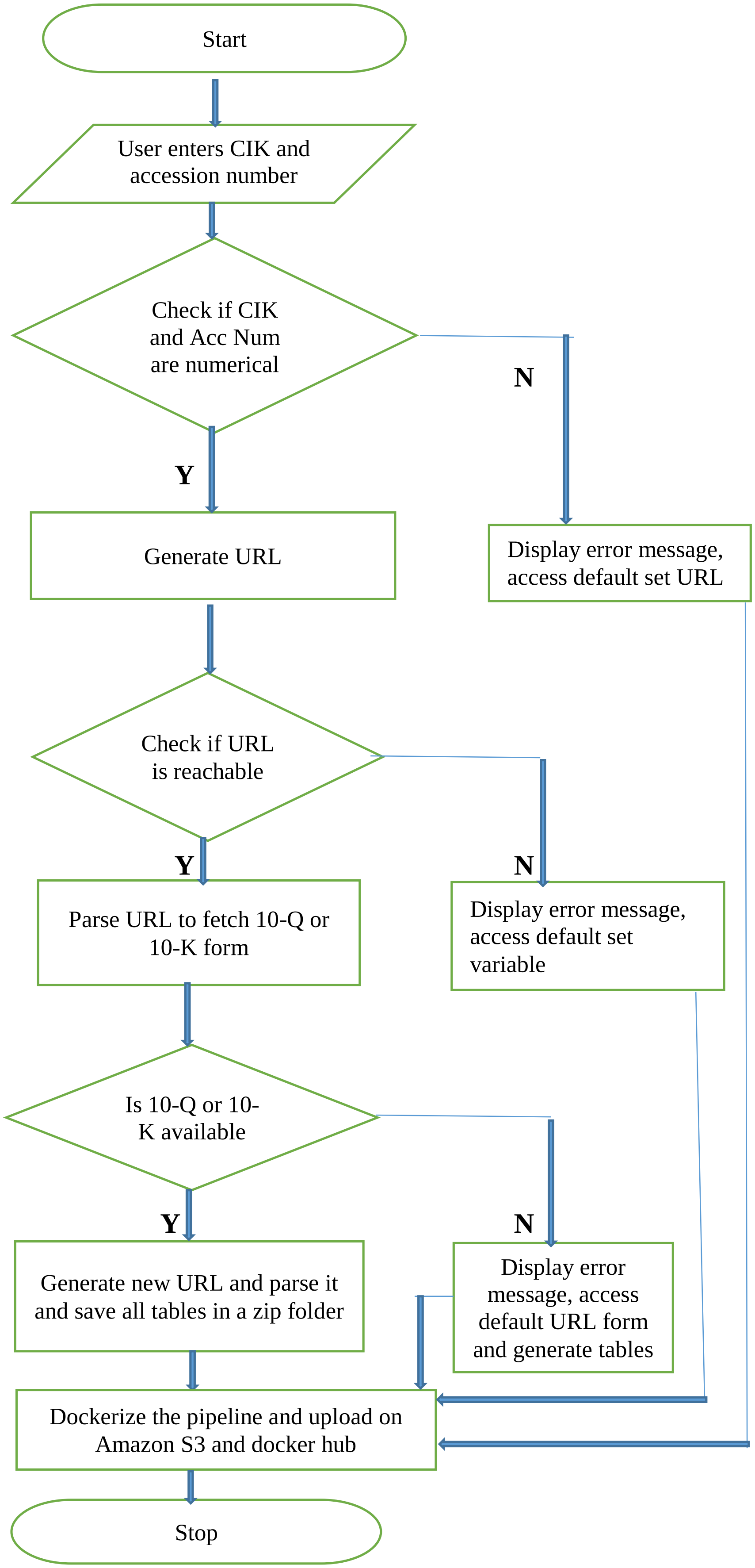
Given a company with CIK (company ID) XXX (omitting leading zeroes) and document accession number YYY (acc-no on search results), we are required to programmatically generate the url to get data (http://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007index.html for IBM for example). Then parse 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). Finally, parse this file to extract "all" tables in this filing and save them as csv files.

Part 2: Dockerize this pipeline

In this part, our goal is to build a docker image that can automate this task for any CIK and document accession number which could be parameterized in a config file. We should be able to replace IBM's CIK and document accession number with Googles to generate the url https://www.sec.gov/Archives/edgar/data/1652044/000165204417000008/0001652044-17-000008index.htm and then parse this document to look for the 10K filing https://www.sec.gov/Archives/edgar/data/1652044/000165204417000008/goog10-kq42016.htm and extract all tables from this filing. The program should log all activities, then zip the tables and upload the log file and the zip file to Amazon S3. Also, we should parameterize cik, accession number and amazon keys so that anyone can put their cik, accession number and amazon keys and locations and reuse our code.

Flow Chart:

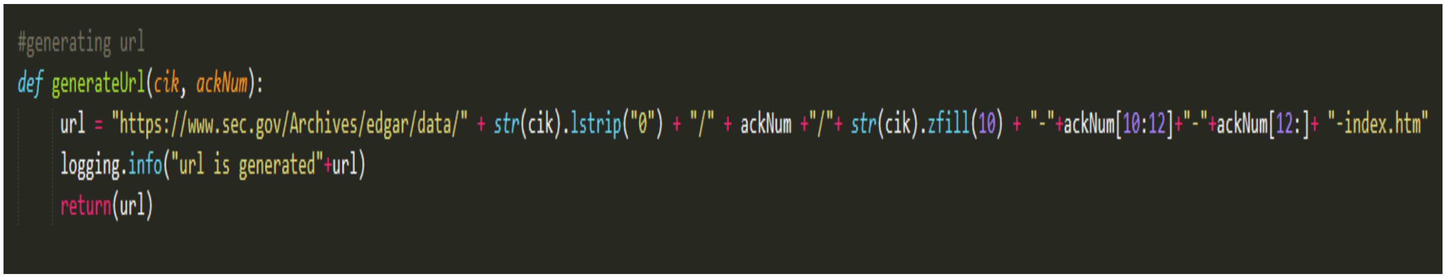
Problem 1


Start
User enters CIK and accession number
Check if CIK and Acc Num are numerical
Y
N
Generate URL
Display error message, access default set URL
Check if URL is reachable
Y
N
Parse URL to fetch 10-Q or 10-K form
Display error message, access default set variable
Is 10-Q or 10-K available
Y
N
Generate new URL and parse it and save all tables in a zip folder
Display error message, access default URL form and generate tables
Dockerize the pipeline and upload on Amazon S3 and docker hub
Stop

Data Parsing

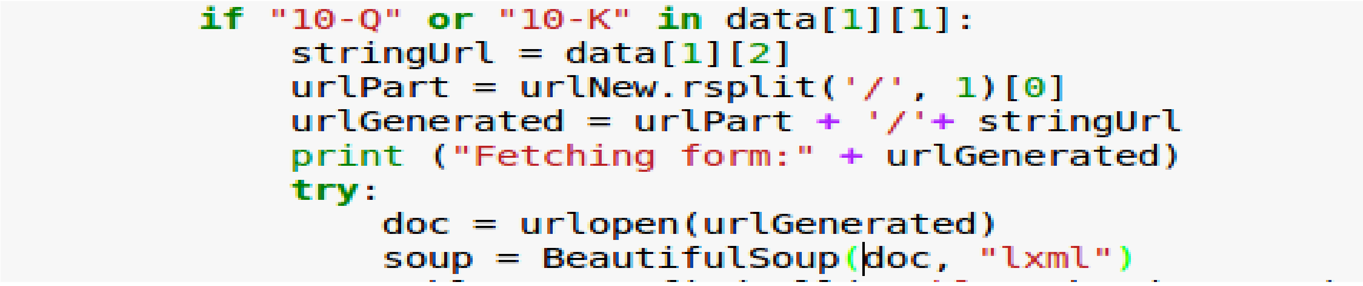
In Part 1, we analyzed how to access data from Edgar.

1. Our approach is to first write a function to generate URL based on cik and accession number that the user can provide. The function is as follows:



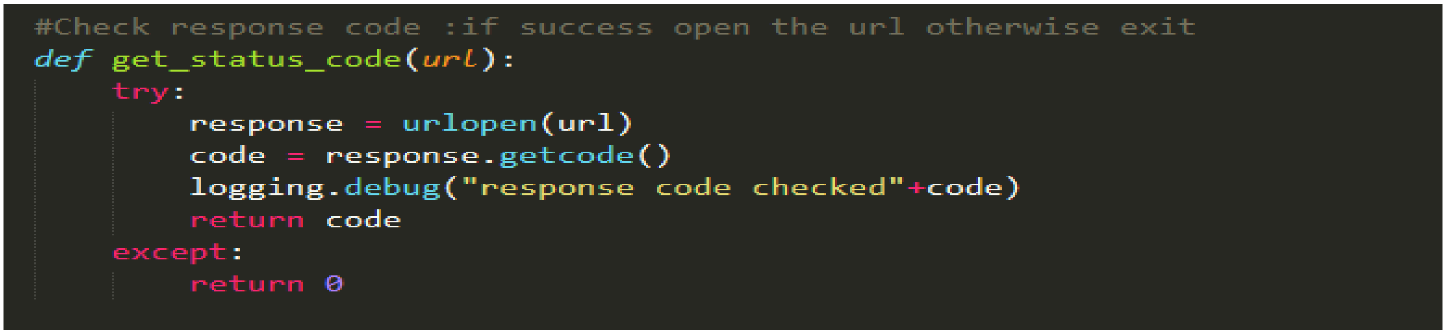
```
#generating url
def generateUrl(cik, ackNum):
    url = "https://www.sec.gov/Archives/edgar/data/" + str(cik).lstrip("0") + "/" + ackNum +"/"+ str(cik).zfill(10) + "-"+ackNum[10:12]+"-"+ackNum[12:]+ "-index.htm"
    logging.info("url is generated"+url)
    return(url)
```

2. Then we fetch 10-K or 10-Q form whichever is available for that particular CIK + accession number combination:



```
        if "10-Q" or "10-K" in data[1][1]:
            stringUrl = data[1][2]
            urlPart = urlNew.rsplit('/', 1)[0]
            urlGenerated = urlPart + '/'+ stringUrl
            print ("Fetching form:" + urlGenerated)
            try:
                doc = urlopen(urlGenerated)
                soup = BeautifulSoup(doc, "lxml")
```

3. We have checked for response codes when the function is called,

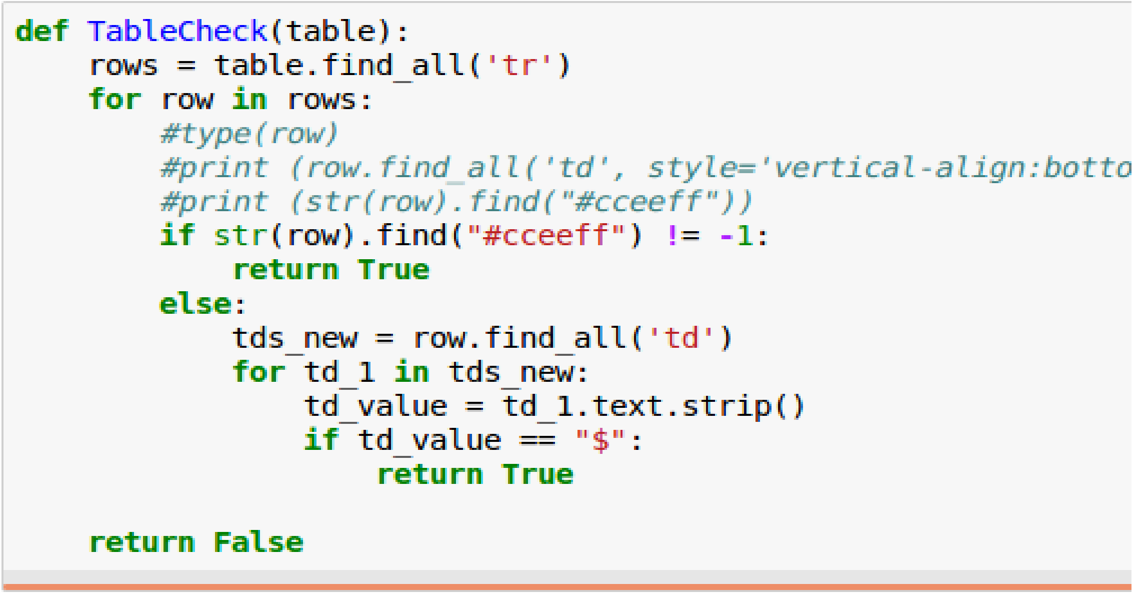


```
#Check response code :if success open the url otherwise exit
def get_status_code(url):
    try:
        response = urlopen(url)
        code = response.getcode()
        logging.debug("response code checked"+code)
        return code
    except:
        return 0
```

4. To fetch the accurate tables, we are using two conditions:
   - The table rows should be colored (because there are many tables which only contain text in table tag and not actual tabular content)
   - If table is not colored and does not contain any border like in url: https://www.sec.gov/Archives/edgar/data/315189/000155837017000952/de-20170129x10q.htm

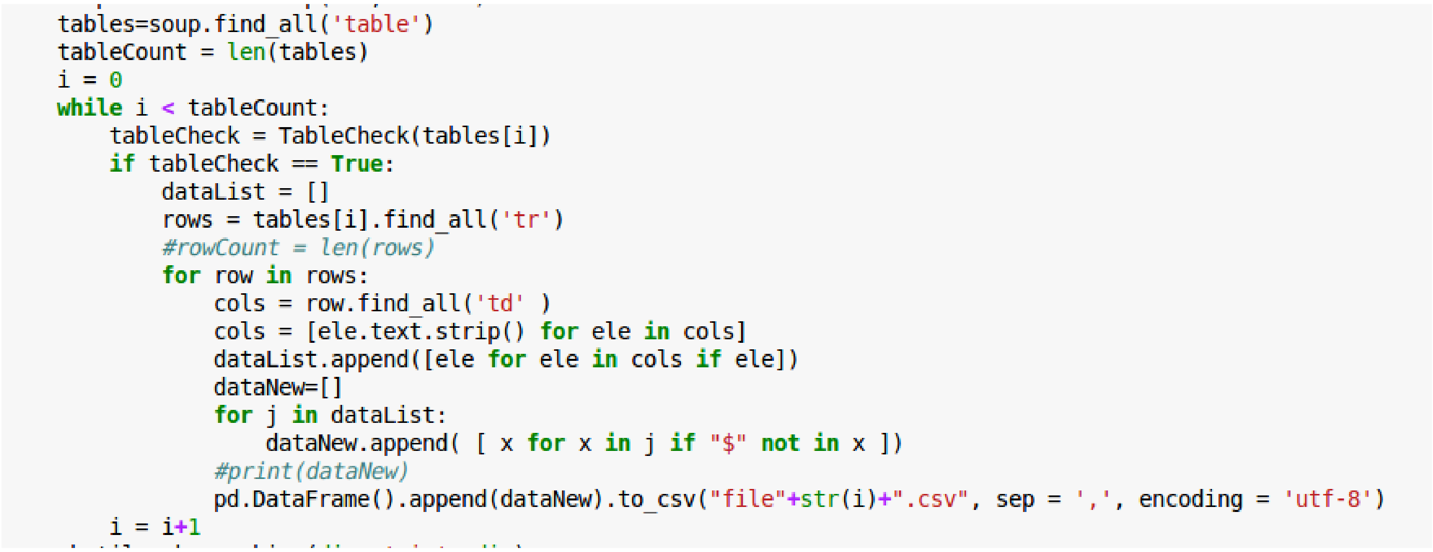
     Then we are checking if "$" is present in the table as a separate column.

Below is the function for checking above conditions:



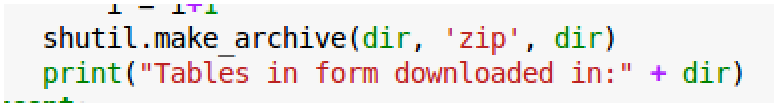
```
def TableCheck(table):
    rows = table.find_all('tr')
    for row in rows:
        #type(row)
        #print (row.find_all('td', style='vertical-align:botto
        #print (str(row).find("#cceeff"))
        if str(row).find("#cceeff") != -1:
            return True
        else:
            tds_new = row.find_all('td')
            for td_1 in tds_new:
                td_value = td_1.text.strip()
                if td_value == "$":
                    return True

    return False
```

5. We are calling this function in our main code as shown below:



```
tables=soup.find_all('table')
tableCount = len(tables)
i = 0
while i < tableCount:
    tableCheck = TableCheck(tables[i])
    if tableCheck == True:
        dataList = []
        rows = tables[i].find_all('tr')
        #rowCount = len(rows)
        for row in rows:
            cols = row.find_all('td' )
            cols = [ele.text.strip() for ele in cols]
            dataList.append([ele for ele in cols if ele])
            dataNew=[]
            for j in dataList:
                dataNew.append( [ x for x in j if "$" not in x ])
            #print(dataNew)
            pd.DataFrame().append(dataNew).to_csv("file"+str(i)+".csv", sep = ',', encoding = 'utf-8')
    i = i+1
```

6. Finally, we are saving all tables in a folder and then create a zip folder:

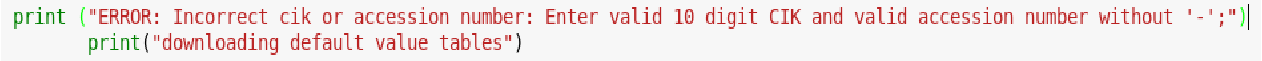


```
shutil.make_archive(dir, 'zip', dir)
print("Tables in form downloaded in:" + dir)
```

7. In case user enters incorrect CIK or accession number, we are performing following checks:
   - To check if CIK is valid we are checking the value entered by user in the CIK list present on EDGAR site.

- If CIK and accession code entered by user are invalid, the http url generated will not go through and return an error message to user prompting to enter valid CIK and accession number
- If the value is null for CIK or accession number, then also user will be prompted to enter correct values and program will not go through.
- For CIK if user enters with leading zeroes the zeros will be omitted while generating the url
- We are also ensuring if both cik and accession number are numeric

In any of the above case error message will be displayed to user and by default tables for IBM 10Q form will be downloaded.
Below is the message that we are printing:



```
print ("ERROR: Incorrect cik or accession number: Enter valid 10 digit CIK and valid accession number without '-';")
      print("downloading default value tables")
```

Dockerize the pipeline

Now that we have completed our program for parsing the data files, our next step is to dockerize the task so that any user can reuse our code to access data from EDGAR.

We have built a Docker image for that.
Building a docker image here requires three files: DockerFile, Source Code file and requirements file.
We have created a separate requirement.txt file to import the required Python libraries for our program. This is done to make our code cleaner and better understanding of the dependencies.
So, when the docker image run, all our libraries are imported.

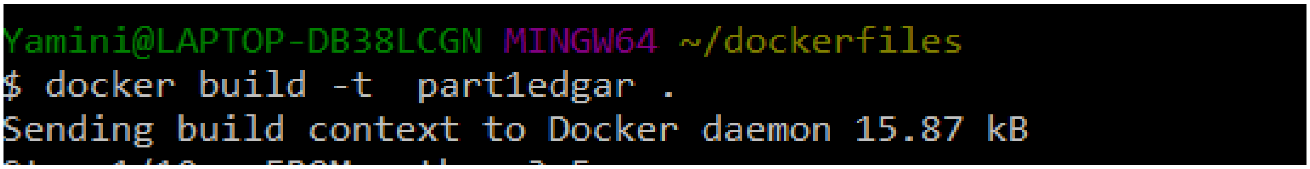
As per the requirement, we have programmed in a way so that the data files will be stored on Amazon S3 in user bucket. So in this case, the user should have a valid Amazon access and secret keys.
The user is required to specify the CIK and accession number for which company they require the data along with the Access keys and Secret key and locations.

So, our first step is building the docker image.

Step1. Build the image

We have created a directory – **dockerfiles** where we have put our source code file, DockerFile and requirements.txt file.
We have named our image as "**part1edgar**"

So, run the below command to build the image.



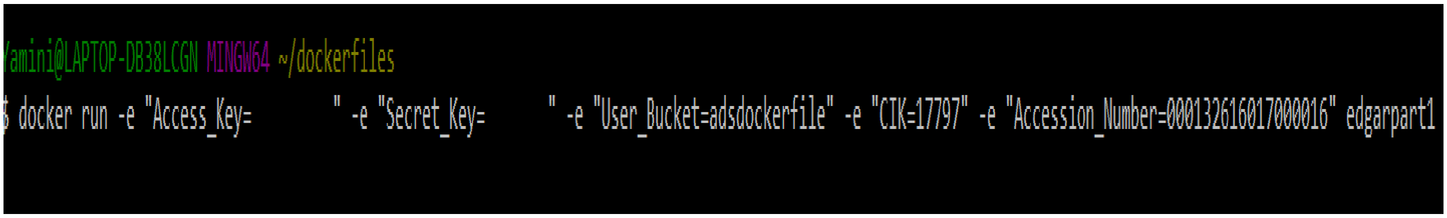
```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
$ docker build -t  part1edgar .
Sending build context to Docker daemon 15.87 kB
```

Step2. Run the Image

Next, once our image is build we will run the image. We can check image created using command – $docker images

The command use to run the image is:

```
docker run -e "Access_Key = _______"
           -e "Secret_Key = _______"
           -e "User_Bucket = ______"
           -e "CIK = _______"
           -e "Accession_Number = ______"
            part1edgar
```



```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
$ docker run -e "Access_Key=[redacted]" -e "Secret_Key=[redacted]" -e "User_Bucket=adsdockerfile" -e "CIK=17797" -e "Accession_Number=0001326160170000016" edgarpart1
```

-- OR—

For ex: to extract data for CIK: 17797 and Accession Number: 000132616017000016, You can copy this command and enter your credentials and simply run….

```
$ docker run -e "Access_Key = (Your AWS Access Key) "
              -e Secret_Key = (Your AWS Secret Key) "
              -e "User_Bucket= (Your Bucket) "
              -e "CIK=17797"
             -e "Accession_Number=000132616017000016"
              edgarpart1
```

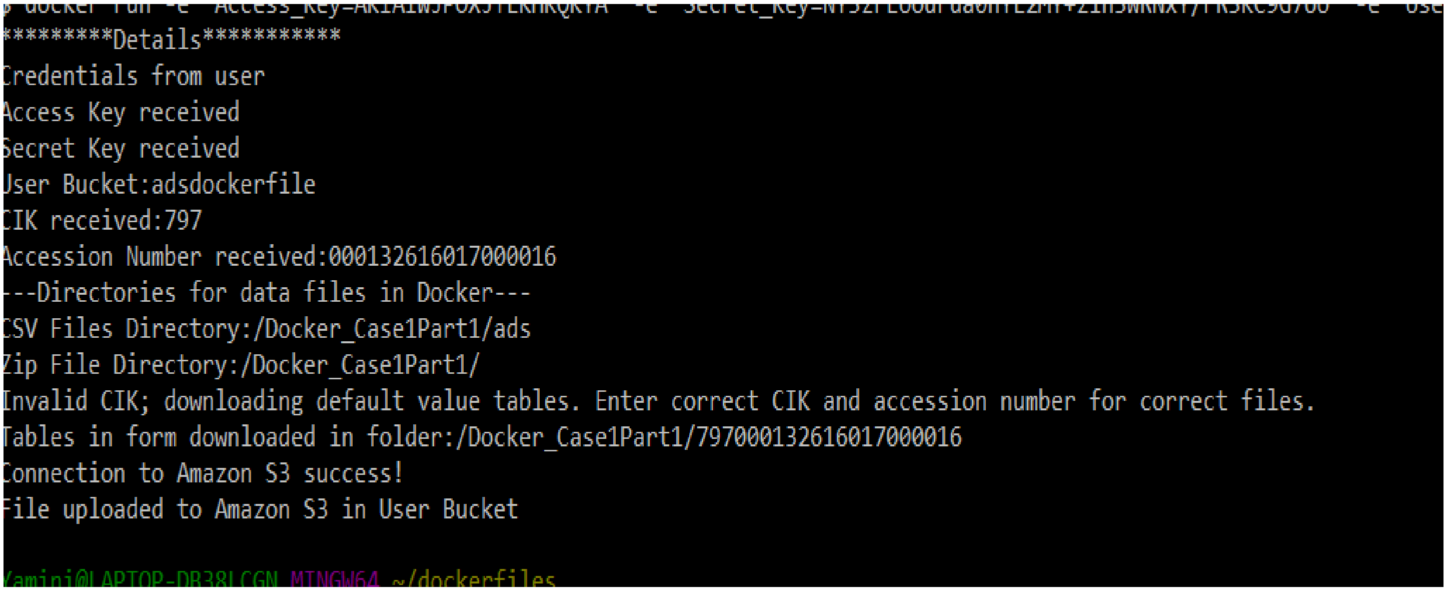
--------Note: Please use the credentials variables exactly as given in the instructions.

In our case, User bucket is adsdockerfile,
We have generated data for CIK: 51143 and Accession Number: 000005114313000007
As seen in the screenshot, user is required to pass following parameter:
Access_Key
Secret_Key
User_Bucket
CIK
Accession_Number

Access Keys and **Secret Keys** for connection to Amazon S3,
Bucket where the user's file data will be uploaded,
CIK of the company along with **Accession_Number**.

Step3. Success

After the image has successfully run, we can see success message with some details as in the docker:

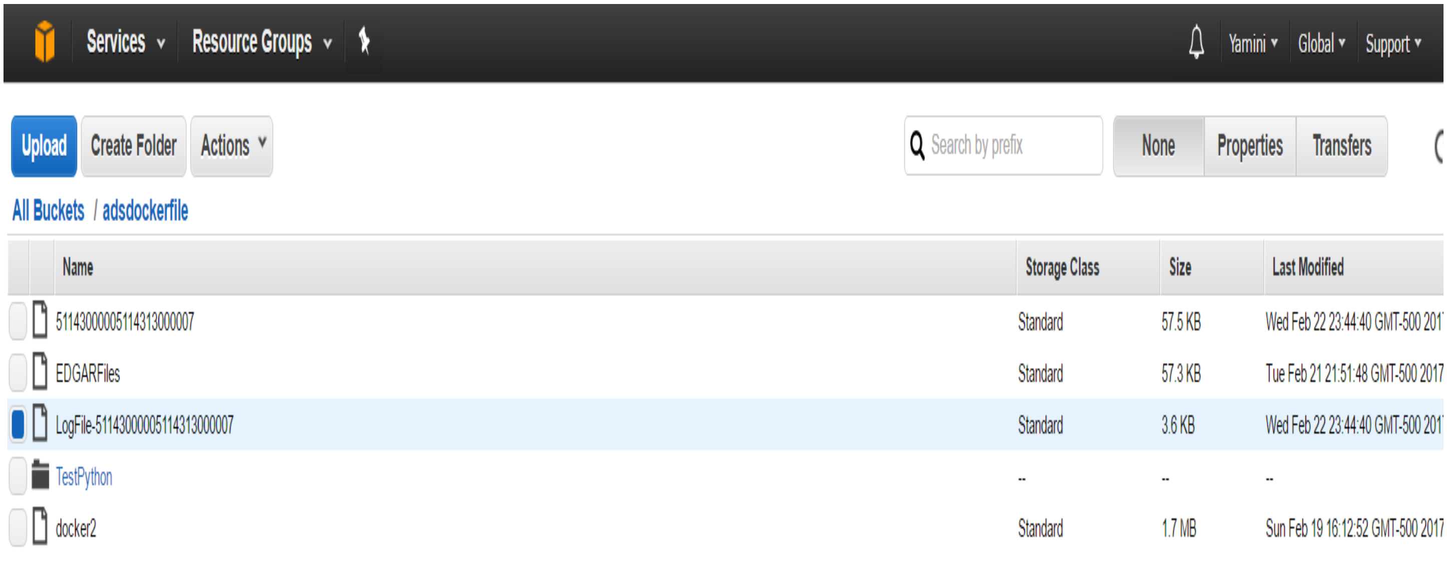


```
$ docker run -e "Access_Key=AKIAIWJFOXJYLKHKQKYA" -e "Secret_Key=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" -e "Use
*********Details***********
Credentials from user
Access Key received
Secret Key received
User Bucket:adsdockerfile
CIK received:797
Accession Number received:0001326160170000016
---Directories for data files in Docker---
CSV Files Directory:/Docker_Case1Part1/ads
Zip File Directory:/Docker_Case1Part1/
Invalid CIK; downloading default value tables. Enter correct CIK and accession number for correct files.
Tables in form downloaded in folder:/Docker_Case1Part1/7970001326160170000016
Connection to Amazon S3 success!
File uploaded to Amazon S3 in User Bucket

Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
```

Since we have not specified CIK, the program will take some default value and upload the files to Amazon. The user can see message here to enter correct CIK.

On Amazon S3, we can see the files are uploaded to the user bucket: Data file and Log file.

Note: The log file logs all the activities in our program with time stamps.

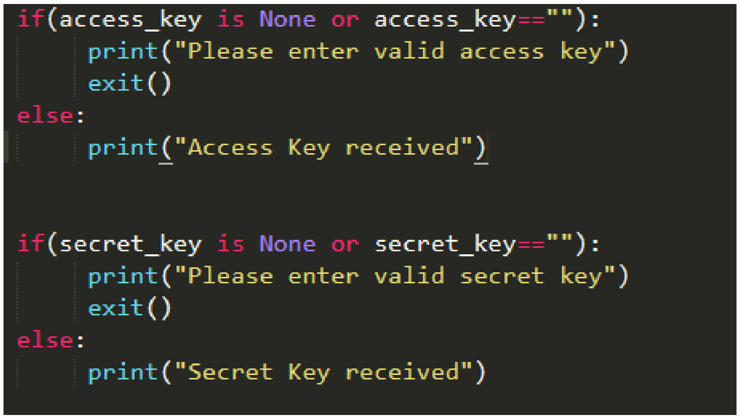


All Buckets / adsdockerfile

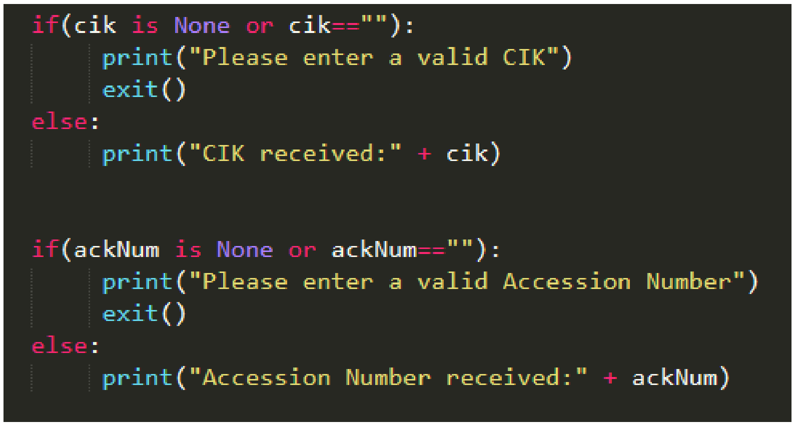
Name	Storage Class	Size	Last Modified
511430000051143130000007	Standard	57.5 KB	Wed Feb 22 23:44:40 GMT-500 201
EDGARFiles	Standard	57.3 KB	Tue Feb 21 21:51:48 GMT-500 2017
LogFile-511430000051143130000007	Standard	3.6 KB	Wed Feb 22 23:44:40 GMT-500 201
TestPython	--	--	--
docker2	Standard	1.7 MB	Sun Feb 19 16:12:52 GMT-500 2017

Exception/Error handling.

➔ For null/blank values of Access Key, Secret Key, UserBucket, CIK and Accession Number, we have put a condition if the values are null/blank, the program will exit with a message to the user otherwise will process.

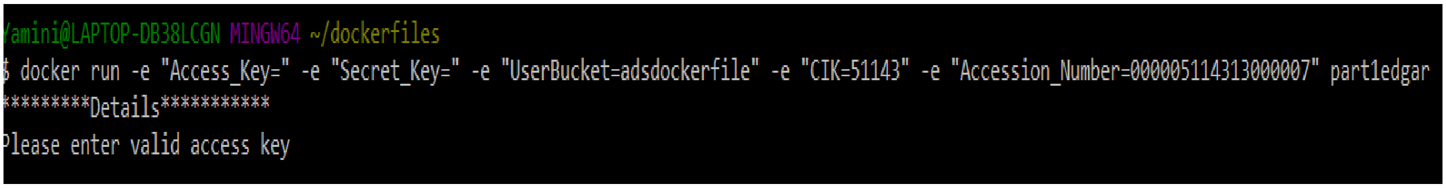


```
if(access_key is None or access_key==""):
    print("Please enter valid access key")
    exit()
else:
    print("Access Key received")


if(secret_key is None or secret_key==""):
    print("Please enter valid secret key")
    exit()
else:
    print("Secret Key received")
```



```
if(cik is None or cik==""):
    print("Please enter a valid CIK")
    exit()
else:
    print("CIK received:" + cik)


if(ackNum is None or ackNum==""):
    print("Please enter a valid Accession Number")
    exit()
else:
    print("Accession Number received:" + ackNum)
```

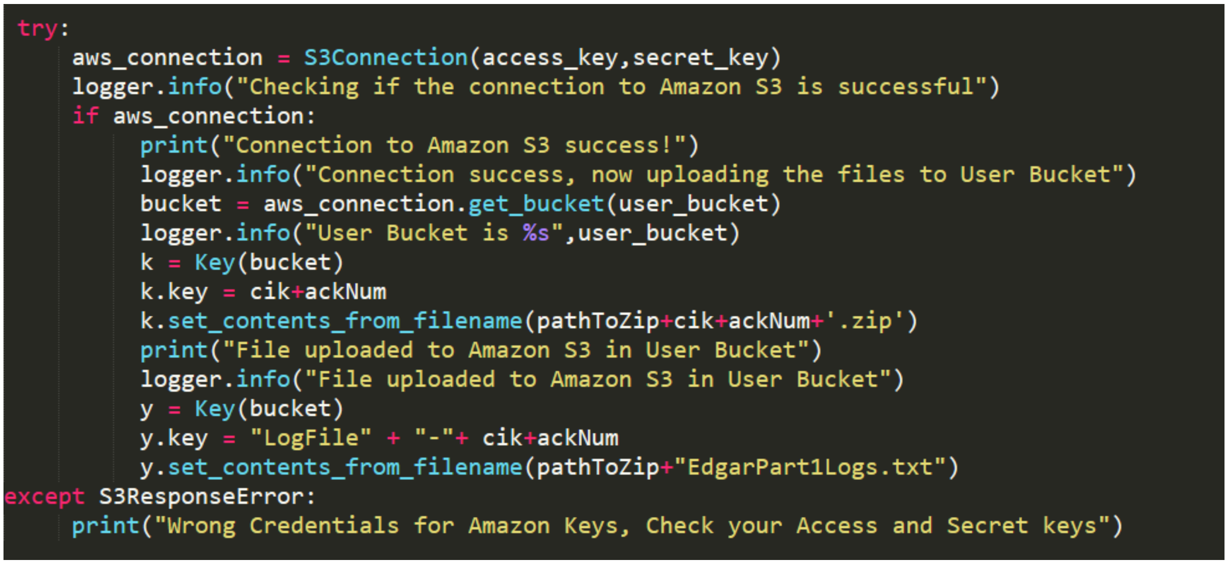
For ex: If the user left Access_key field as null: The program will exit with a message to the user.



```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
$ docker run -e "Access_Key=" -e "Secret_Key=" -e "UserBucket=adsdockerfile" -e "CIK=51143" -e "Accession_Number=000005114313000007" part1edgar
*********Details***********
Please enter valid access key
```

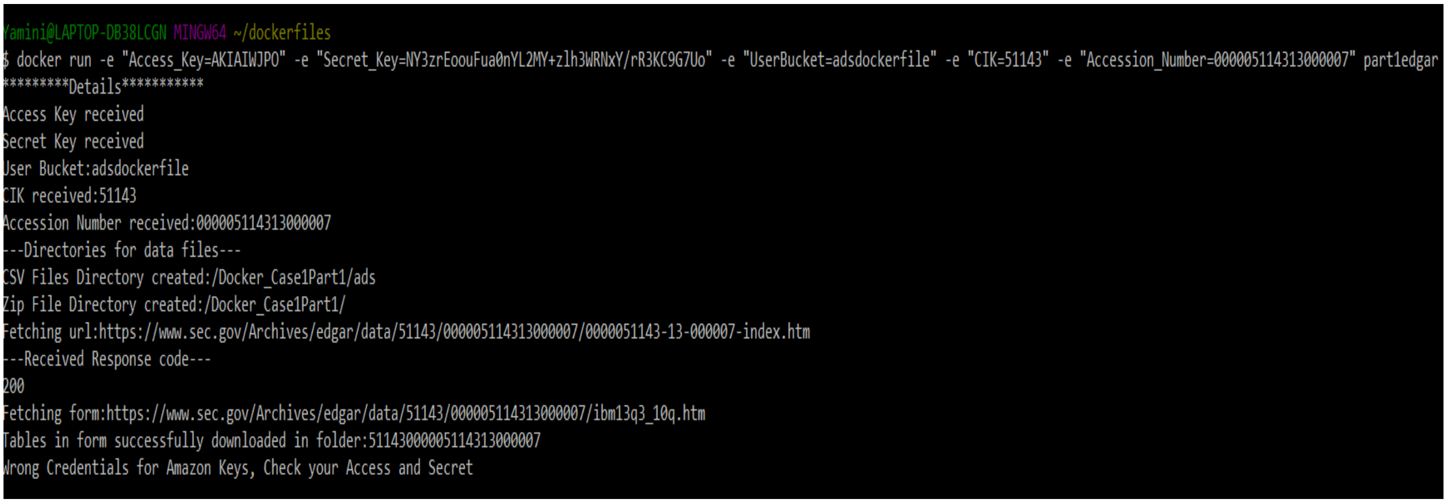
Same goes for other parameters. The program will exit if any of these mandatory parameters are not entered by user.

➔ For Invalid Access Keys and Secret Keys, the connection will not be made:



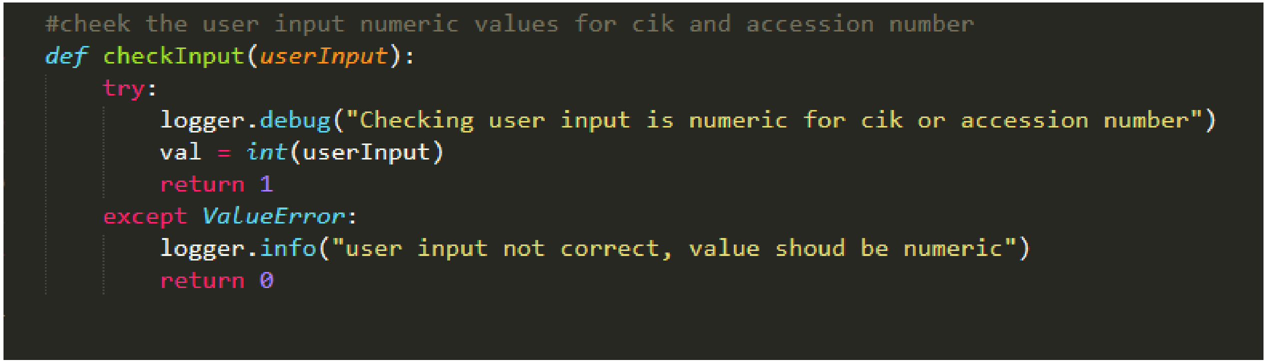
```
try:
    aws_connection = S3Connection(access_key,secret_key)
    logger.info("Checking if the connection to Amazon S3 is successful")
    if aws_connection:
        print("Connection to Amazon S3 success!")
        logger.info("Connection success, now uploading the files to User Bucket")
        bucket = aws_connection.get_bucket(user_bucket)
        logger.info("User Bucket is %s",user_bucket)
        k = Key(bucket)
        k.key = cik+ackNum
        k.set_contents_from_filename(pathToZip+cik+ackNum+'.zip')
        print("File uploaded to Amazon S3 in User Bucket")
        logger.info("File uploaded to Amazon S3 in User Bucket")
        y = Key(bucket)
        y.key = "LogFile" + "-"+ cik+ackNum
        y.set_contents_from_filename(pathToZip+"EdgarPart1Logs.txt")
except S3ResponseError:
    print("Wrong Credentials for Amazon Keys, Check your Access and Secret keys")
```

So, if the user enters some wrong Access Keys, the connection will be interrupted with a message to the user:



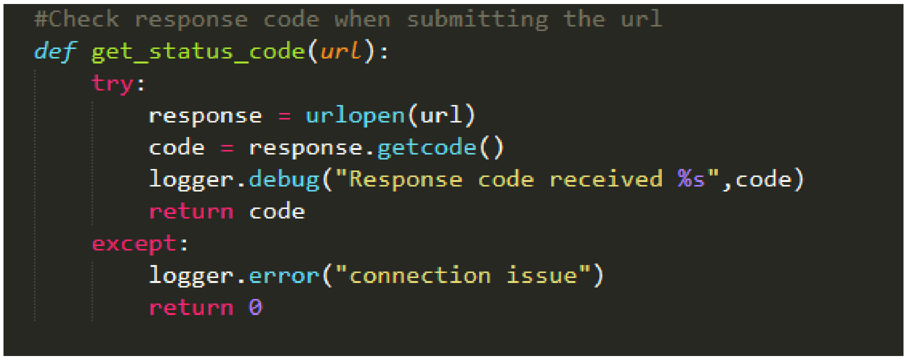
```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
$ docker run -e "Access_Key=AKIAIWJPO" -e "Secret_Key=NY3zrEoouFua0nYL2MY+zlh3WRNxY/rR3KC9G7Uo" -e "UserBucket=adsdockerfile" -e "CIK=51143" -e "Accession_Number=000005114313000007" part1edgar
*********Details***********
Access Key received
Secret Key received
User Bucket:adsdockerfile
CIK received:51143
Accession Number received:000005114313000007
---Directories for data files---
CSV Files Directory created:/Docker_Case1Part1/ads
Zip File Directory created:/Docker_Case1Part1/
Fetching url:https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007-index.htm
---Received Response code---
200
Fetching form:https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm
Tables in form successfully downloaded in folder:51143000005114313000007
Wrong Credentials for Amazon Keys, Check your Access and Secret
```

➔ Next, the values for CIK and Accession Number should be Numeric:



```
#cheek the user input numeric values for cik and accession number
def checkInput(userInput):
    try:
        logger.debug("Checking user input is numeric for cik or accession number")
        val = int(userInput)
        return 1
    except ValueError:
        logger.info("user input not correct, value shoud be numeric")
        return 0
```

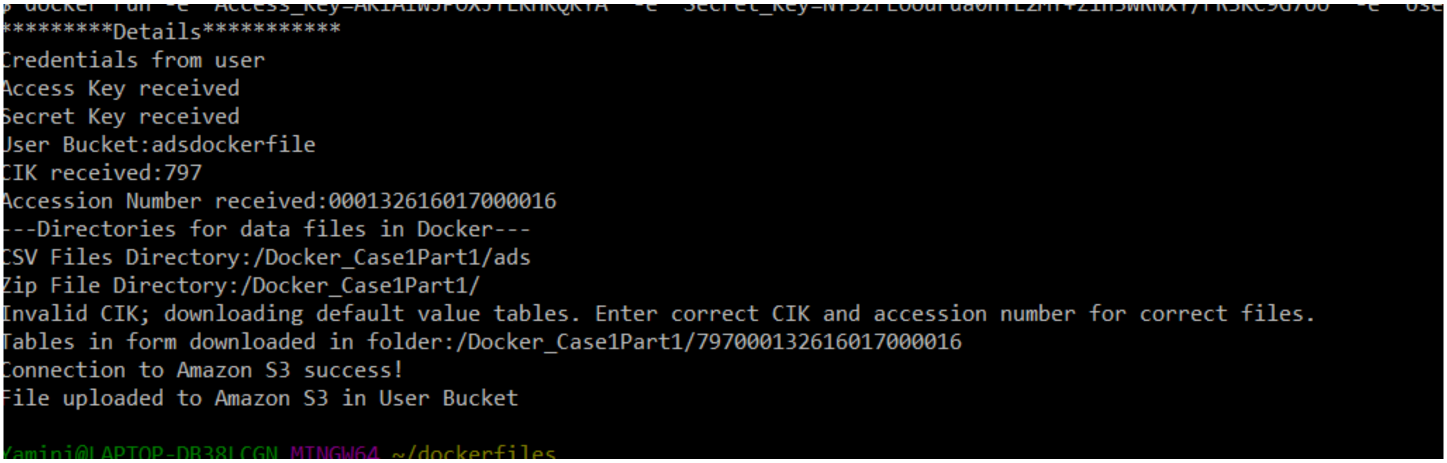
Check the response code we received when submitting the URL:



```
#Check response code when submitting the url
def get_status_code(url):
    try:
        response = urlopen(url)
        code = response.getcode()
        logger.debug("Response code received %s",code)
        return code
    except:
        logger.error("connection issue")
        return 0
```

➔ We have also checked the validity for CIK, it should exist in the data base:

The user will get a message when running the docker image if the CIK is not valid. And we have added some default value based on which tables will uploaded to Amazon.



```
$ docker run -e "Access_Key=AKIAIWJFOXJTLKHKQKYA" -e "Secret_Key=[illegible]" -e "Use
*********Details***********
Credentials from user
Access Key received
Secret Key received
User Bucket:adsdockerfile
CIK received:797
Accession Number received:000132616017000016
---Directories for data files in Docker---
CSV Files Directory:/Docker_Case1Part1/ads
Zip File Directory:/Docker_Case1Part1/
Invalid CIK; downloading default value tables. Enter correct CIK and accession number for correct files.
Tables in form downloaded in folder:/Docker_Case1Part1/797000132616017000016
Connection to Amazon S3 success!
File uploaded to Amazon S3 in User Bucket

Yamini@LAPTOP-DB38LCGN MINGW64 ~/dockerfiles
```

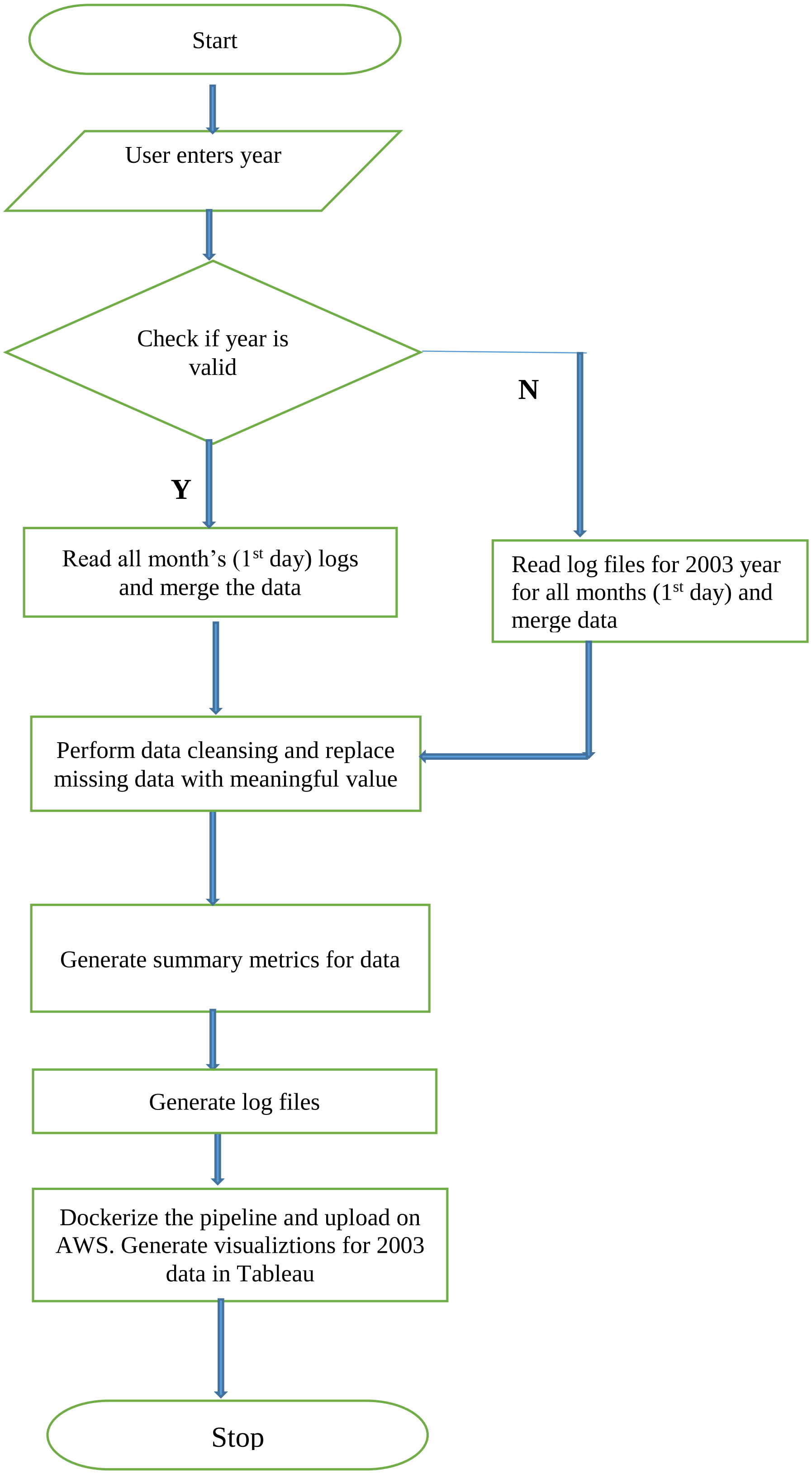
Problem 2: Missing Data Analysis and Visualization

In this problem our goal is to analyze the EDGAR Log File Data Set [https://www.sec.gov/data/edgar-log-filedata-set.html]. The page lists the meta data for the datasets and we are expected to develop a pipeline which does the following. Given a year, our program should get data for the first day of the month (programmatically generate the url http://www.sec.gov/dera/data/PublicEDGAR-log-file-data/2003/Qtr1/log20030101.zip for Jan 2003 for example) for every month in the year and process the file for the following:

- Handle missing data
- Compute summary metrics (Decide which ones)
- Check for any observable anomalies
- Your program should log all the operations (with time stamps) into a log file.
- Compile all the data and summaries of the 12 files into one file
- Upload this compiled data file and the log file you generated to your Amazon S3 bucket

The code should work for any year on the page. We should create a Docker image which runs the pipeline. Parameterize it so that anyone can put their specific keys and locations and reuse our code.
We are required to try our Docker image on AWS and run it for 2003 and share the locations for the AWS bucket with the processed data and log file in our report

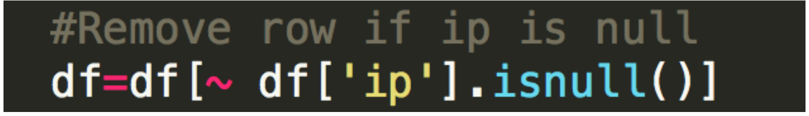
Flow Chart: Problem 2


Start
User enters year
Check if year is valid
N
Y
Read all month's (1st day) logs and merge the data
Read log files for 2003 year for all months (1st day) and merge data
Perform data cleansing and replace missing data with meaningful value
Generate summary metrics for data
Generate log files
Dockerize the pipeline and upload on AWS. Generate visualiztions for 2003 data in Tableau
Stop

Handling Missing Data

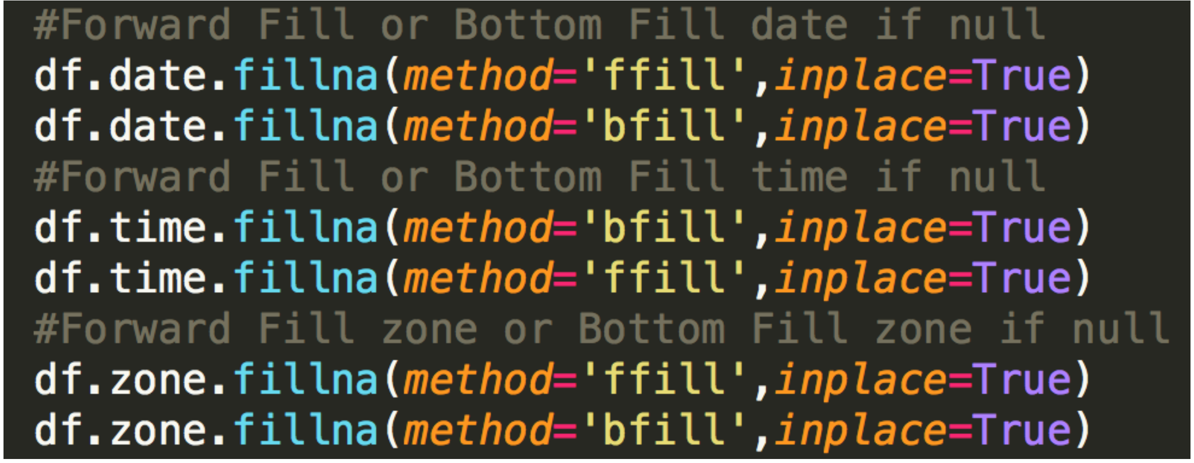
For each column of data null values are being handled using specific approach. There are a total of 15 columns in the log file CSV.

For IP Address column, if it is null, we are removing the row as it doesn't hold any information of the source.



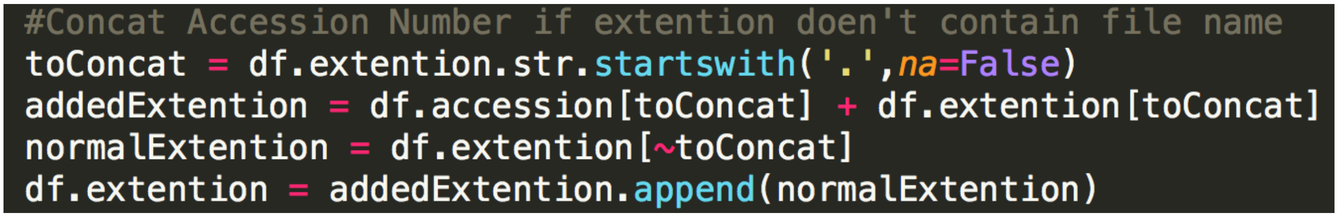
```
#Remove row if ip is null
df=df[~ df['ip'].isnull()]
```

For Date, Time, Time Zone we are using an approach of Forward fill and Bottom fill which fill null values with the immediate preceding value.



```
#Forward Fill or Bottom Fill date if null
df.date.fillna(method='ffill',inplace=True)
df.date.fillna(method='bfill',inplace=True)
#Forward Fill or Bottom Fill time if null
df.time.fillna(method='bfill',inplace=True)
df.time.fillna(method='ffill',inplace=True)
#Forward Fill zone or Bottom Fill zone if null
df.zone.fillna(method='ffill',inplace=True)
df.zone.fillna(method='bfill',inplace=True)
```

For extension column, if the data contains only extension then we are concatenating the extension with the data in the accession number column.



```
#Concat Accession Number if extention doen't contain file name
toConcat = df.extention.str.startswith('.',na=False)
addedExtention = df.accession[toConcat] + df.extention[toConcat]
normalExtention = df.extention[~toConcat]
df.extention = addedExtention.append(normalExtention)
```

For rest of columns below are the default values that replace nulls:

Column Name	New Value if NaN
Response code	404
Size & idx	0
Norefer & noagent	1
Find & crawler	0
browser	Unknown

Compute summary metrics

We have computed below summary matrices:

Numerical Analysis:

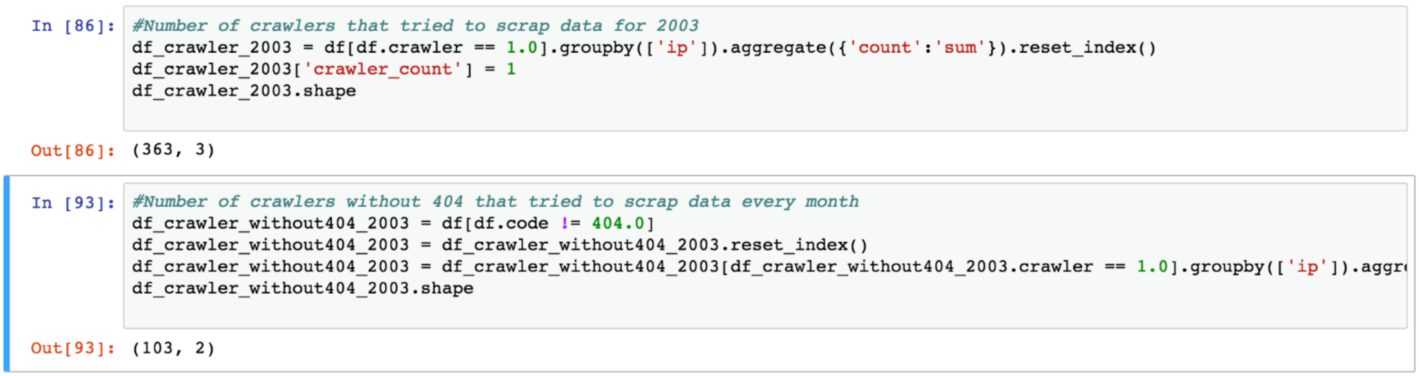
1. Number of requests that came from different browsers.
2. Number of requests that came in for each first day of month
3. Calculate Mean, Min, Max of file size's requested for each first day of month
4. Number of Distinct IPs that requested data for each first day of month

Categorical Analysis:

1. Based on HTTP Response code, Number of request that came in for each first day of month based on.
2. Differentiate crawlers with and without response code 404 for each month.

Observable Anomalies

As part of variables description, crawlers are being marked 1 if the response code is 404. But in the real case after analysis we found that is a difference between the number of crawlers with and without 404 response code in a year.



```
In [86]: #Number of crawlers that tried to scrap data for 2003
         df_crawler_2003 = df[df.crawler == 1.0].groupby(['ip']).aggregate({'count':'sum'}).reset_index()
         df_crawler_2003['crawler_count'] = 1
         df_crawler_2003.shape

Out[86]: (363, 3)

In [93]: #Number of crawlers without 404 that tried to scrap data every month
         df_crawler_without404_2003 = df[df.code != 404.0]
         df_crawler_without404_2003 = df_crawler_without404_2003.reset_index()
         df_crawler_without404_2003 = df_crawler_without404_2003[df_crawler_without404_2003.crawler == 1.0].groupby(['ip']).aggr
         df_crawler_without404_2003.shape

Out[93]: (103, 2)
```

The approach the we used as below:

1. Calculate number of unique crawlers based on IP in a year.
2. Calculate number of unique crawlers based on IP excluding the ones which have response code of 404.

Conclusion:

As in the above image the numbers are not nearly same, so the log file which marks crawler as True for response code 404 doesn't sound right.

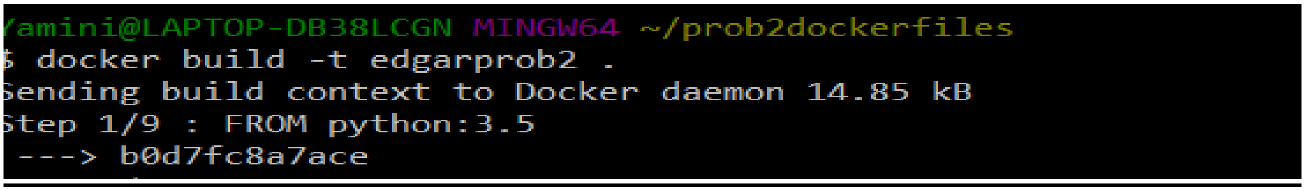
Dockerize the pipeline:

Next to dockerize the pipeline, our approach is same as in Part 1:

Step 1: Build the Image: edgarprob2

Our image is **edgarprob2**.

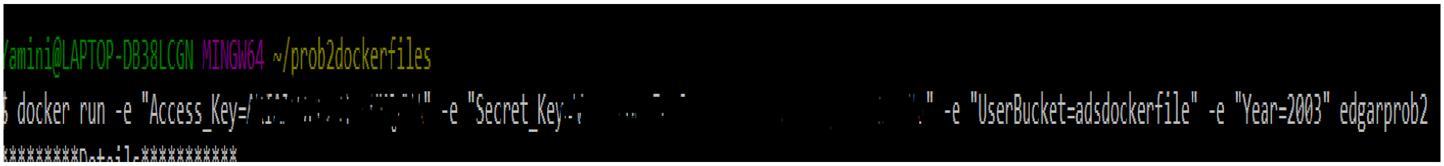
Here we have put our Source code file, Docker File and requirements.txt file in directory: prob2dockerfiles



```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/prob2dockerfiles
$ docker build -t edgarprob2 .
Sending build context to Docker daemon 14.85 kB
Step 1/9 : FROM python:3.5
 ---> b0d7fc8a7ace
```

Step2: Run the image

```
docker run -e “Access_Key = _______”
           -e “Secret_Key = _______”
           -e “User_Bucket = ______”
           -e “Year = _______”
           edgarprob2
```

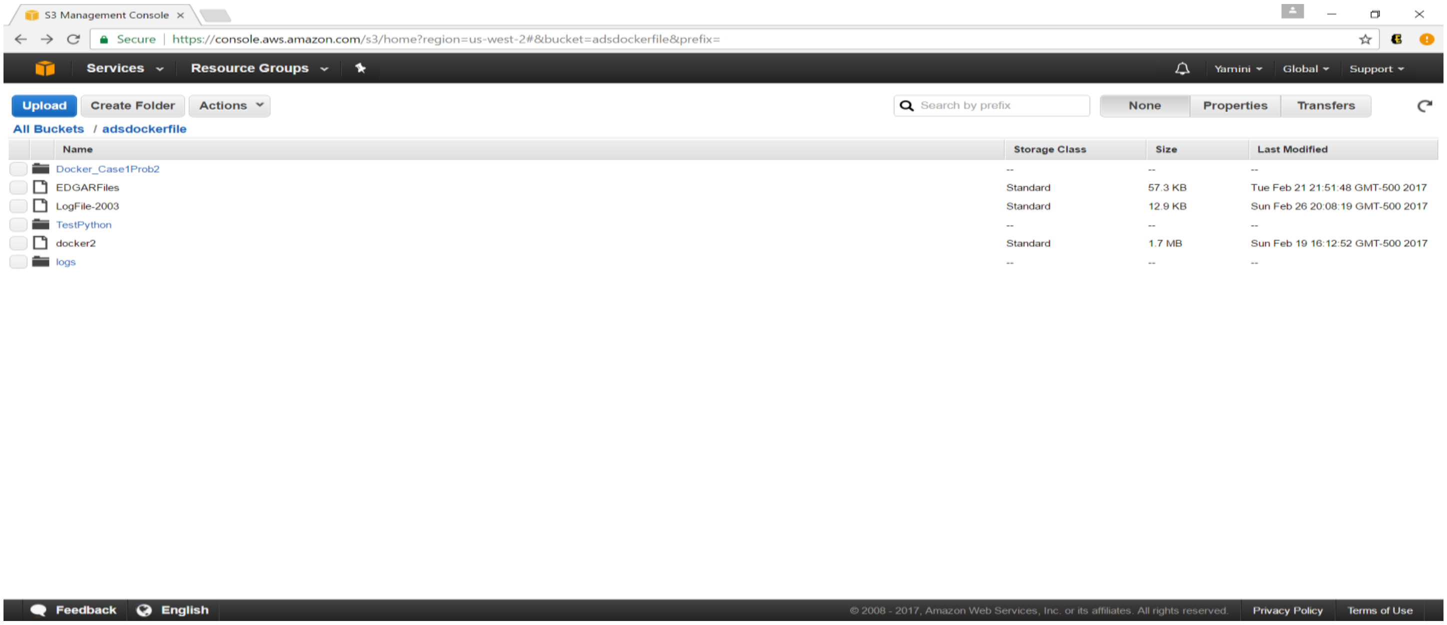


```
Yamini@LAPTOP-DB38LCGN MINGW64 ~/prob2dockerfiles
$ docker run -e "Access_Key=[redacted]" -e "Secret_Key=[redacted]" -e "UserBucket=adsdockerfile" -e "Year=2003" edgarprob2
********Detail-**********
```

The user is expected to provide Access Key and Secret Key for Amazon account along with bucket.
And Year for which the file is required.
If No year is specified, we have set default year as 2003.

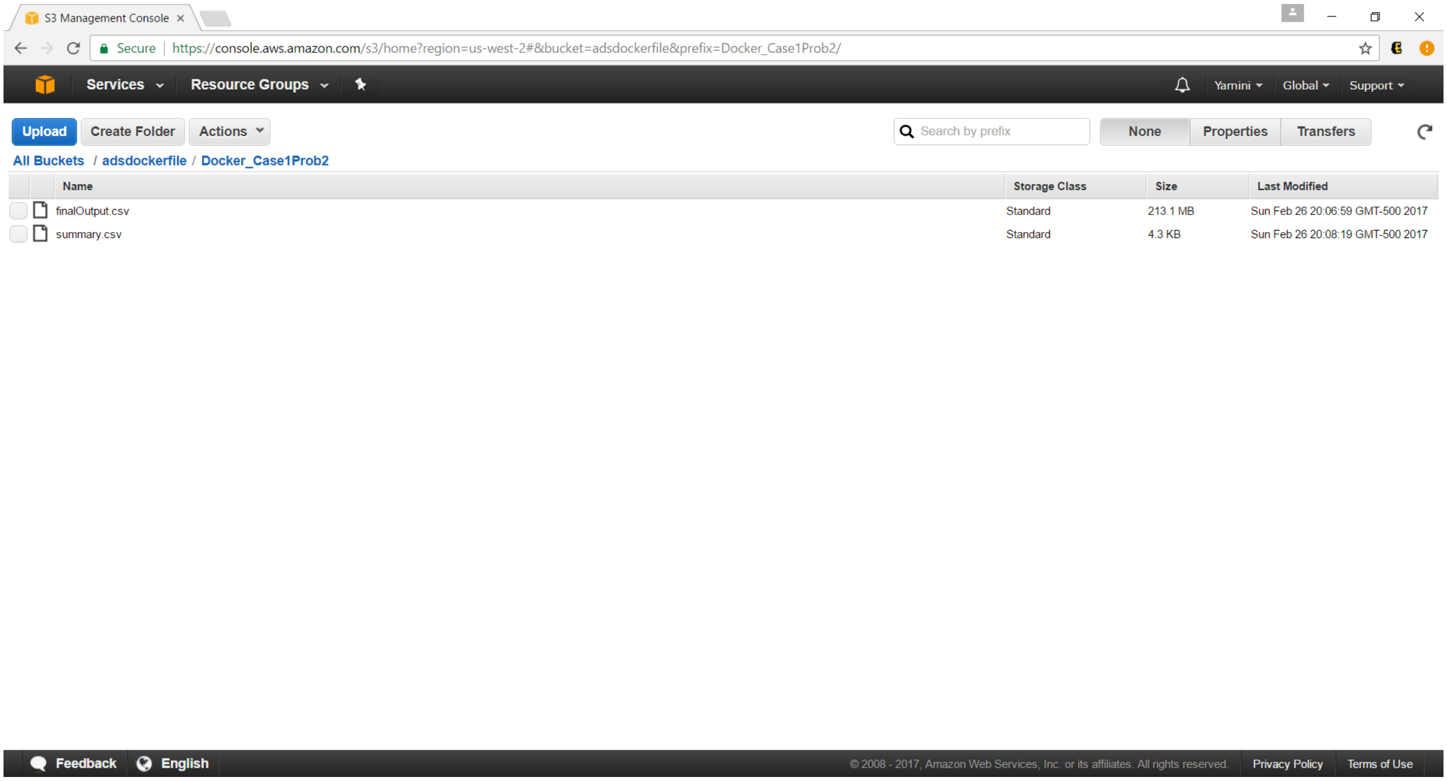
Step3: Success

The files are uploaded to Amazon S3 for year 2003 under folder: Docker_Case1Part2
And the logfile is created as LogFile – 2003(this is created outside the folder)


S3 Management Console
Secure | https://console.aws.amazon.com/s3/home?region=us-west-2#&bucket=adsdockerfile&prefix=
Services
Resource Groups
Yamini
Global
Support
Upload
Create Folder
Actions
Search by prefix
None
Properties
Transfers
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

All Buckets / adsdockerfile

Name	Storage Class	Size	Last Modified
Docker_Case1Prob2	--	--	--
EDGARFiles	Standard	57.3 KB	Tue Feb 21 21:51:48 GMT-500 2017
LogFile-2003	Standard	12.9 KB	Sun Feb 26 20:08:19 GMT-500 2017
TestPython	--	--	--
docker2	Standard	1.7 MB	Sun Feb 19 16:12:52 GMT-500 2017
logs	--	--	--


S3 Management Console
Secure | https://console.aws.amazon.com/s3/home?region=us-west-2#&bucket=adsdockerfile&prefix=Docker_Case1Prob2/
Services
Resource Groups
Yamini
Global
Support
Upload
Create Folder
Actions
Search by prefix
None
Properties
Transfers
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

All Buckets / adsdockerfile / Docker_Case1Prob2

Name	Storage Class	Size	Last Modified
finalOutput.csv	Standard	213.1 MB	Sun Feb 26 20:06:59 GMT-500 2017
summary.csv	Standard	4.3 KB	Sun Feb 26 20:08:19 GMT-500 2017

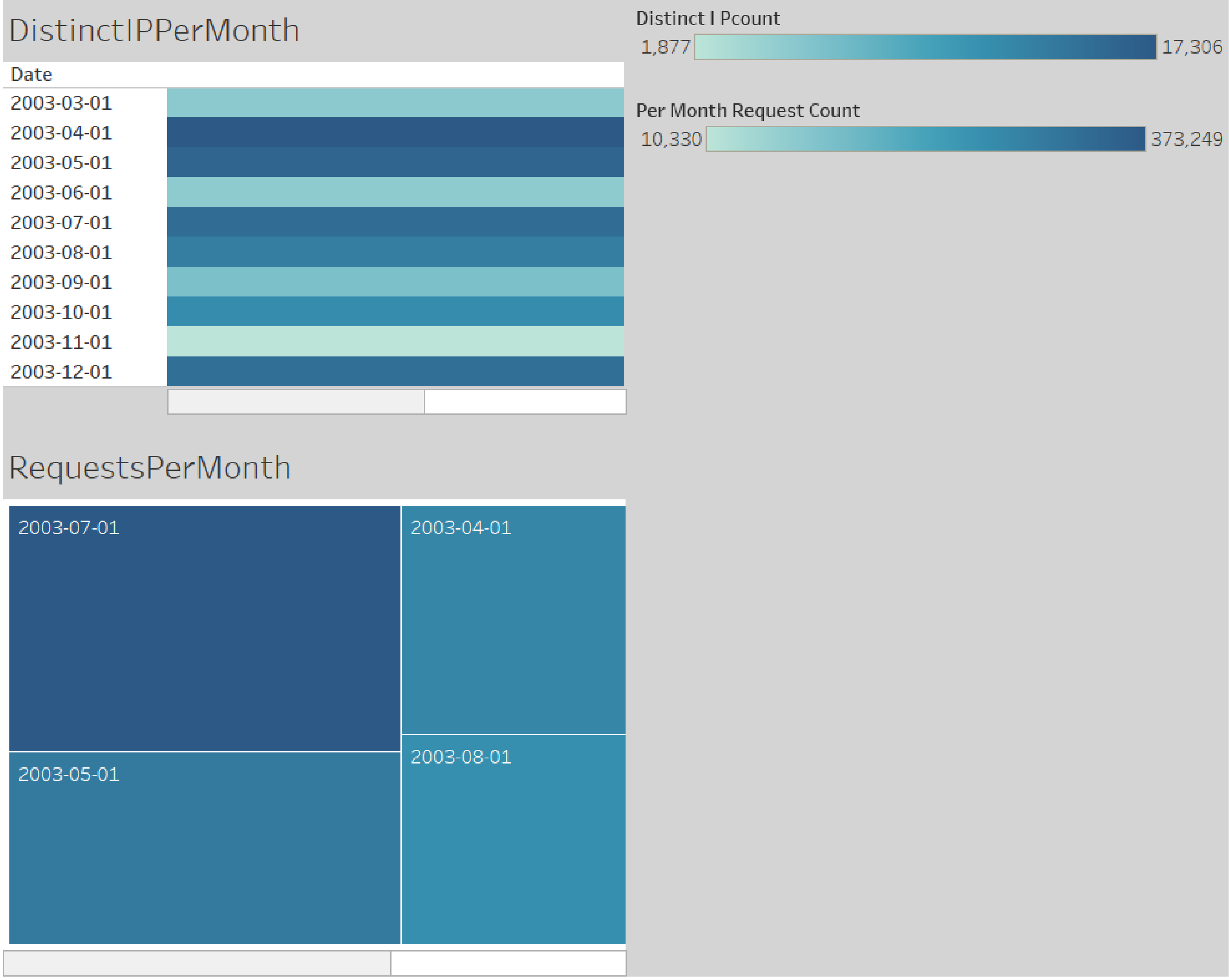
Visualization Using Tableau:

Following reports illustrate the summary metrics computed for 2003 year for all companies:

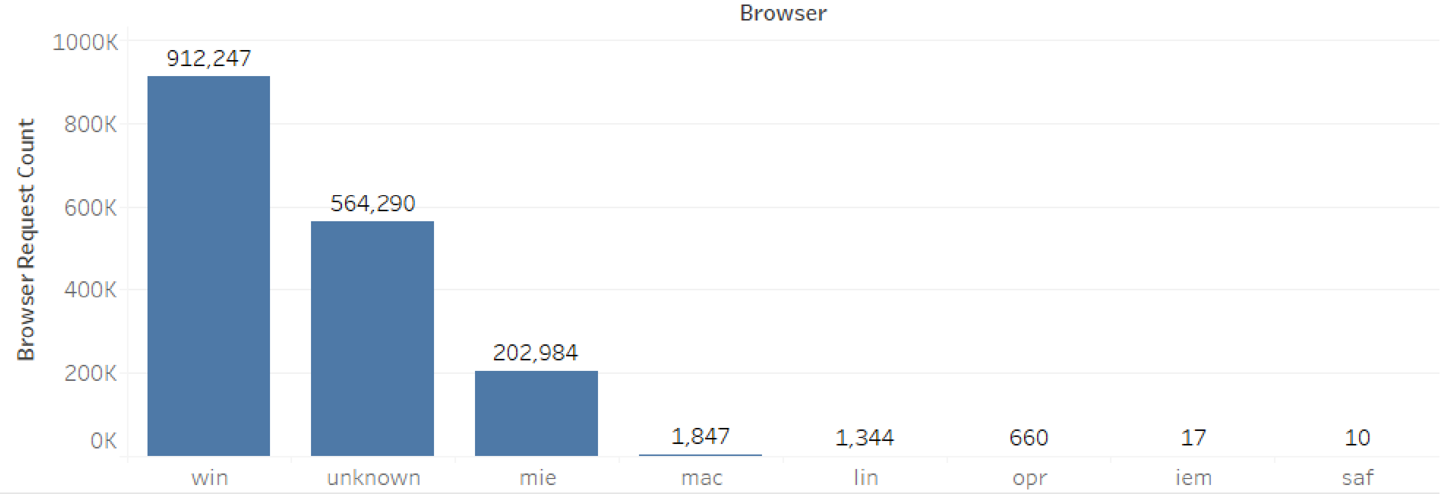
All the Dashboards and other workbooks are published on tableau: https://10az.online.tableau.com/#/site/tableauadsanalysis/workbooks/1209813/views

The data source is uploaded to Amazon S3:

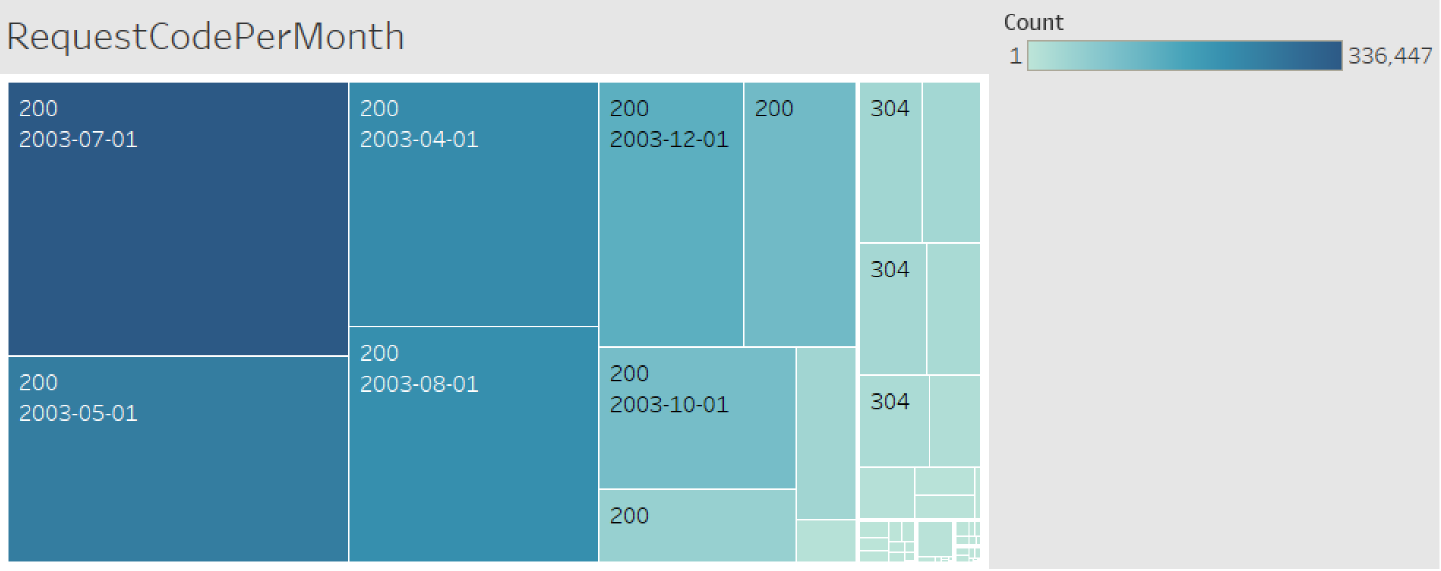
https://console.aws.amazon.com/s3/home?region=us-west-2#&bucket=adsdockerfile&prefix=

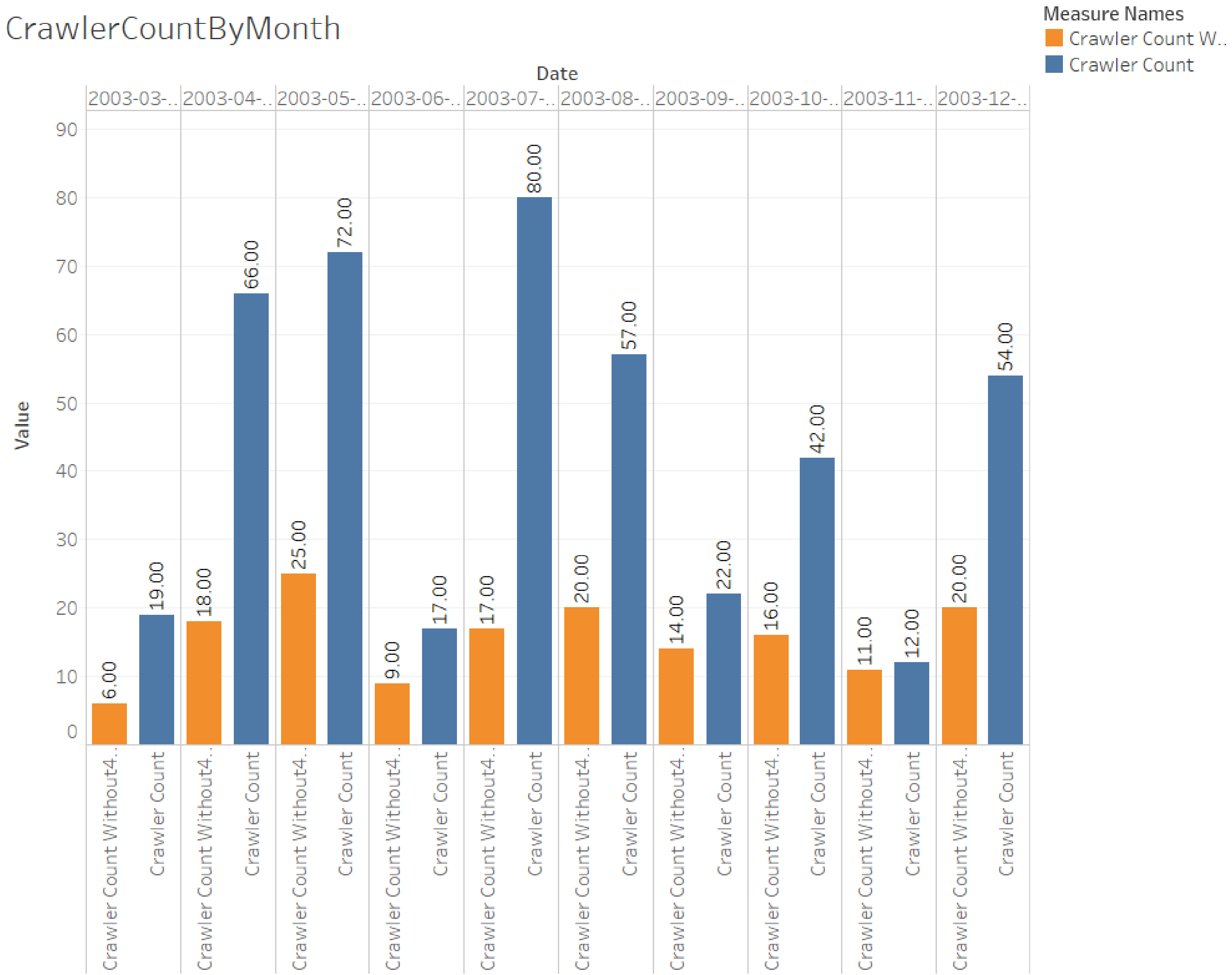


BrowserCount



RequestCodePerMonth




CrawlerCountByMonth
Measure Names
Crawler Count W..
Crawler Count
Date
2003-03-..
2003-04-..
2003-05-..
2003-06-..
2003-07-..
2003-08-..
2003-09-..
2003-10-..
2003-11-..
2003-12-..
Value
90
80
70
60
50
40
30
20
10
0
6.00
19.00
18.00
66.00
25.00
72.00
9.00
17.00
17.00
80.00
20.00
57.00
14.00
22.00
16.00
42.00
11.00
12.00
20.00
54.00
Crawler Count Without4..
Crawler Count

GitHub Links:

Github Repo: https://github.com/Yamini-S/ADS

Docker hub links:

We have register our docker image on docker hub and the links are:

Repo: yaminis/ads_assignment
Link: https://hub.docker.com/r/yaminis/ads_assignment/

For Problem 1:

Steps:

1. Pull the image on docker:
   Tag: edgarpart1

```
$docker pull yaminis/ads_assignment: edgarpart1
```

Note: Please give tag name as edgarpart1 after the ":", to pull image for problem1

2. You can see in docker images; the image is created.
3. Run the image as explained in above sections.

```
docker run -e "Access_Key = _______"
           -e "Secret_Key = _______"
           -e "User_Bucket = ______"
           -e "CIK = _______"
           -e "Accession_Number = ______"
           part1edgar
```

For Problem 2:
Steps:

1. Pull the image on docker:
   Tag: edgarprob2

```
$docker pull yaminis/ads_assignment: edgarprob2
```

Note: Please give tag name as edgarprob2 after the ":", to pull image for problem1

2. You can see in docker images; the image is created.
3. Run the image as explained in above sections.

```
docker run -e "Access_Key = ________"
           -e "Secret_Key = ________"
           -e "User_Bucket = _______"
           -e "Year = ________"
           edgarprob2
```

Amazon S3 Links:

For Problem 1:
Data File: https://s3.amazonaws.com/adsdockerfile/177970001326160170000016
LogFile: https://s3.amazonaws.com/adsdockerfile/LogFile-177970001326160170000016

For Problem2:
Data Files (CSV File including the Summary Metric):
https://s3.amazonaws.com/adsdockerfile/Docker_Case1Prob2/finalOutput.csv,

Summary Metrics file separate: https://s3.amazonaws.com/adsdockerfile/Docker_Case1Prob2/summary.csv

LogFile: https://s3.amazonaws.com/adsdockerfile/LogFile-2003

Summary Metrics file for Tableau Analysis: https://s3.amazonaws.com/adsdockerfile/summaryEdgarLogs.xlsx