7390 Homework 1 report

Problem 1: Data wrangling Edgar data from text files (50 points)

How to run our dockers image? (Parameterized your own params)

We are using two different way to pass parameters. For part one, the application is imbedded in a Flask application.

Part One:

Step 1: docker run -p 5000:5000 --name p1instance -t yinking422/p1

Step 2: run in browser:

0.0.0.0:5000/accessKey&secretAccessKey&cik&AcessionNo

Part Two:

docker run -ti yinking422/p2

How to dynamic generated url?

We are using Beautiful Soap library to wrangle the website. We could get the content using html tag.

In order to get dynamic url, we first analyse the format of IBM and Google data page.

IBM

```
<tr>
<td scope="row">1</td>
<td scope="row">10-Q</td>
<td scope="row"><a
href="/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm">ibm13q3_10q.h
tm</a></td>
<td scope="row">10-Q</td>
<td scope="row">7305482</td>
</tr>
```

Google

```
<tr>
<td scope="row">1</td>
<td scope="row">FORM 10-Q</td>
<td scope="row"><a href="/Archives/edgar/data/1288776/000128877615000046/goog10-
qq32015.htm">goog10-qq32015.htm</a></td>
<td scope="row">10-Q</td>
<td scope="row">2236200</td>
</tr>
```

why used formtype = soup.findAll('td', text=formname)[0].parent vs # formtype = soup.findAll('td', text=formname)[0].find_previous_sibling('td')

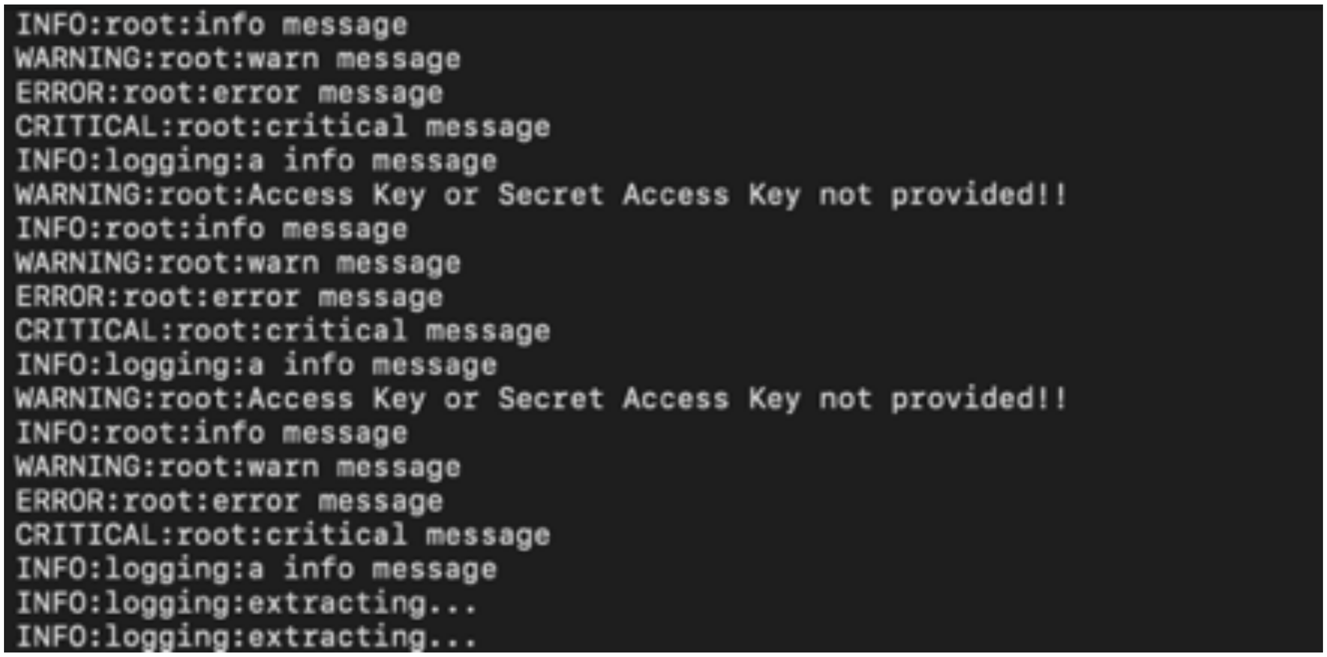
There is identical type equals 10-Q. We locate this then to find the parent of it, So we could get the url.

```
        tr = soup.findAll('td', text=formname)[0].parent
          for a in tr.find_all('a'):
            href = a.get("href")
            url2 = "https://www.sec.gov/" + href
            print(url2)
```

Problem 2 Missing Data Analysis

We took the following steps to solve problem 2

Step 1 initialize a log file
By using logging package we create a log file named "logger.log" and using logging.handler to create a handler with format to write into a log file and the log file look like this:



```
INFO:root:info message
WARNING:root:warn message
ERROR:root:error message
CRITICAL:root:critical message
INFO:logging:a info message
WARNING:root:Access Key or Secret Access Key not provided!!
INFO:root:info message
WARNING:root:warn message
ERROR:root:error message
CRITICAL:root:critical message
INFO:logging:a info message
WARNING:root:Access Key or Secret Access Key not provided!!
INFO:root:info message
WARNING:root:warn message
ERROR:root:error message
CRITICAL:root:critical message
INFO:logging:a info message
INFO:logging:extracting...
INFO:logging:extracting...
```

Step 2 Check all the input
In this program we want to user to enter their AWS access key and secret key also the year, before processing any further we must check whether those input data are right. We use sys.argv function to get the input for each string in says.argv[] we extract it and get the data we want. We firstly check the amazon keys, by using boto.connect_s3(ACCESS_KEY,ACCESS_SECRET_KEY) if this function worked then we will create a unique bucket with timestamp else we will send a warning message and exit the program. If AWS keys works fine and then we will check the year input. First we check wether the user enter something, if the user didn't enter anything we will use 2003 as default year and

send the info message. If user enter something, then we will check the wether the year is between 2003 and 2017, if the input year is not valid the program will be exited.

Step 3 Parsing the data from website
As the result of a thorough inspection into the naming pattern of EDGAR log files, the team realized that each file url consist of its year, quarter of year, and date information. Such a pattern avails the program of generation the url programmatically. And since the url is in a format of zip, the program can download and extract the zip directly by using the “zip” functions integrated in python 3.

Step 4 Handle the missing data
We read the csv file into a DataFrame and then do some data cleaning as follow:
- For the row which have no ip, date, time, cik or accession, delete it.
- Replace the NaN value in browser with ‘unknown’
- Replace NaN idx with the most frequent one
- Replace all the norefer, crawler and noagent with 0
- Replace NaN in find with most frequent find
- Replace NaN in extension with most frequent extension
- Replace NaN in zone with most frequent zone
- Find mean of the size and replace null value with mean

Then write those cleaned data frame into a new csv file ends like “….imput.csv”

Step 5 Check for any observable anomalies
For this step, we divided the problem as categorical and numerical, and solve them separately.

As for categorical columns, we hard code some lists of values according to the readme file downloaded along with original csv files, and then check against the imputed csv file against these lists. For example, the [‘noagent’] column should only consists of 0 or 1, thus the program checks each value of this column agains a list of only 0 and 1.
Hard coding a specific list of for each column may help reduce the times of calculation significantly.
As for the numerical column [‘size’], the team assumed the data obeys a normal distribution, and considered only those values within 5 sigma as valid. Although the team found this data set may not strictly obey a normal distribution which means a further regression analysis according to categorical variables are indicated, such an approach did reduce the anomalies sufficiently.

After detection of anomalies, the program handles them by replacing them by the same value used in imputation.

Step 6 Compute summary metrics
For every input.csv files we read it into a DataFrame and then we do some summary analysis and write the result into a new DataFrame and the result should be one row in the summary metrics file which looks like this:
As you can see we summarize the number of IP address, the most frequent IP, the number of CIK the most frequent CIK number and the extension, the mean of the size and the most frequent browser.

Step 7 Compile all data into one file
After handle the missing and anomalies data, we create a new csv file and use the year and timestamp to create a unique name. For each column in the csv file we just need the data so the header in each csv file should be removed. Use a list to contain each DataFrame from each csv and use pd.concat() function to merge it together

Step 8 Zip the summary metrics, log file and compiled data into the zip file
Use zipfile.Zipfile() function to generate the zip file and find each file name and write it into the zip file. Since the directory of log file is different from the summary metrics and allcsv file, we need to change the directory, use os.chdir('..') to go back and then we can find the log file.

Step 9 Upload the zip file to the Amazon S3
Since we create the bucket in the first step and we got the zip file. Just go to where the zip file directory is and then use Key(bucket) function to generate the key object which is used for store data in the S3 bucket. By binding the key and the file name then the file will be uploaded.

you can see one zip file here:
https://s3.console.aws.amazon.com/s3/buckets/team10ads20181013093840/?region=us-east-1&tab=overview