INFO 7390 Advances in Data Sciences & Architecture

Working with EDGAR datasets

Electronic Data Gathering, Analysis, and Retrieval

Introduction

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). The goal of is to work with Edgar datasets.

Project Tools:

- Language: Python
- Process: Web Scrapping, Data Wrangling, Data Cleansing, Exploratory Data Analysis
- Tools used: Jupyter Notebooks, Amazon S3 bucket, Docker, Sublime text

Team

Ankur Kumar Jain - 1206900

Amandeep Singh - 001271649

Eklavya Saxen - 001850025

GitHub Link:

https://github.com/eklavyasaxena/Advances-in-Data-Sciences-and-Architecture.git

Docker Image on AWS S3

```
# Use the basic Python 3 image as launching point
FROM ubuntu

# Fetch and Install python
RUN ["apt-get", "update"]
RUN ["apt-get", "install", "python","-y"]
RUN ["apt-get", "install","python-pip","-y"]
RUN ["apt-get", "install","python3-pip","-y"]

RUN ["pip3", "install", "bs4"]
RUN ["pip3", "install", "lxml"]
CMD ["pip3", "install", "bs4"]
CMD ["pip3", "install", "lxml"]
CMD ["pip3", "install", "urllib3"]
RUN ["pip3", "install", "boto"]

# Add the script, argv input syntax, & requirements to the Dockerfile
ADD extract_tables.py /home
ADD argv_input_syntax.txt /home
```

Used an Ubuntu 16.4 image and updated its libraries. 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

Docker Summary:

- Dockers does not allow you to write on the root folder of an image, so create a folder within and run the python script there.
- Store all the heavy libraries and cache them , to save processing time.

Problem 1: Data Wrangling EDGAR data from Text Files

In this assignment, we have used beautiful soup to scrape the data from EDGAR website. First, we passed the CIK and Accession Number as arguments to extract the 10q link from the following URL https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/0000051143-13-000007-index.html. Once we get the URL of 10q i.e. https://www.sec.gov/Archives/edgar/data/51143/000005114313000007/ibm13q3_10q.htm and from this link we fetched all the data table and made csv for each table. We then implemented data cleansing which included extracting only tables which have cleaned data.

Process Overview

1. Web Scrapping:
    a. We have written a code to programmatically generate the URL for requested company CIK. If the company URL is valid then it will go to the next step.

```
cik = str(cik)
accession = str(accession)
cik = cik.lstrip('0')
acc = re.sub(r'[-]', r'', accession)
url = 'https://www.sec.gov/Archives/edgar/data/' + cik + '/' + acc + '/' + accession + '/-index.htm'
try:
    page_open = urllib.request.urlopen(url)
    generate_10q_link(url)
except:
    print("Invalid URL {}: ".format(url))
```

    b. To hit the actual URL, we will be using BeautifulSoup, a python library. To fetch the desired URL, we will find all the value of anchor tags and create the URL to open the file requested.

```
all_tables = soup.find('table', class_='tableFile')
tr = all_tables.find_all('tr')
for row in tr:
    final_url = row.findNext("a").attrs['href']
    break
final_link = "https://www.sec.gov" + final_url
get_next_page(final_link)
return (final_link)
```

    c. To find the data tables from the desired URL, we extracted all the tables which are having background color in row and column.

```
def all_datatables(g, a):
    count = 0
    allheaders=[]
    for table in a:
        bluetables = []
        trs = table.find_all('tr')
        for tr in trs:
            global flagtr
            if checktag(str(tr.get('style'))) == "true" or checktag(str(tr)) == "true":
                bluetables = get_table_name(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":
                        bluetables = get_table_name(td.find_parent('table'))
                        break
            if not len(bluetables) == 0:
                break
        if not len(bluetables) == 0:
            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
                        hrname=name+"_"+str(hrcount)
                        allheaders.append(hrname)
                    else:
                        hrname=name
                        allheaders.append(hrname)
                        break
```

d. Once we get all the data table which are having background color, we tried to fetch all the table header. If the header is present then the file will be created with table header name else file will be created with the company file name.

```
#function to check the header tag
def checkheadertag(param):
    flag="false"
    datatabletags=["center","bold"]
    for x in datatabletags:
        if x in param:
            flag="true"
    return flag


#function to check the style of td and tr tags
def checktag(param):
    flag = "false"
    datatabletags = ["background", "bgcolor", "background-color"]
    for x in datatabletags:
        if x in param:
            flag = "true"
    return flag
```

e. Once all the tables are fetched then we exported them into csv. We created a folder with the name of company.

```
folder_name = get_folder_name(g)
path = str(os.getcwd()) + "/" + folder_name
assure_path_exists(path)
if(len(allheaders)==0):
    filename=folder_name+"-"+str(count)
else:
    filename=allheaders.pop()
csvname=filename+".csv"
csvpath = path + "/" + csvname
with open(csvpath, 'w', encoding='utf-8-sig', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(bluetables)
zip_dir(path)
```

f. To clean the output csv file, we used regular expression to remove all the special character.

```
        x=z.text;
        x=re.sub(r"['()]","",str(x))
        x=re.sub(r"[$]"," ",str(x))
        if(len(x)>1):
            x=re.sub(r"[—]","",str(x))
            t.append(x)
    data=([z.encode('utf-8') for z in t])
    r.append([z.decode('utf-8').strip() for z in data])
```

g. We have created a zip folder which will be having all the csv files.

```
#function to create a zip folder
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, '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)))
```

h. To log the processing of the program, we created a log file which will be having all the minute details such as the start and end time of the execution, any error or exception will be logged into the log file.

```
2018-02-16 03:41:42,832 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:42:01,123 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:42:01,489 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:42:33,765 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:42:34,092 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:53:21,743 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:53:21,997 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:53:41,810 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:53:42,057 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:06,345 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:06,568 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:16,275 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:16,503 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:25,950 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:26,340 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:33,647 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:34,019 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:37,406 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:37,742 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 03:54:45,469 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 03:54:45,721 - DEBUG - https://en.wikipedia.org:443 "GET /
2018-02-16 13:07:10,300 - DEBUG - Starting new HTTPS connection (1):
2018-02-16 13:07:10,692 - DEBUG - https://en.wikipedia.org:443 "GET /
```

Problem 2: Missing Data Analysis of EDGAR Log File Data Set

Handling Null and missing values

Initially when data was inserted it has missing values in couple of columns. To handle this missing data, we have gone through following steps

1. First, we retrieved all the columns which has missing or Null values.
2. Next, we analyzed, which columns has numeric values with and missing values.
3. From the above we got only one column with missing values which is “size”.
4. Now to handle missing data we calculated mean of the existing values of size.
5. Now we imputed this mean value in all the missing values of the variable size.
6. Through above operation we have cleansed the data of size variable from all the missing and Null values.
7. To confirm that we have cleaned all the null values, we will check which comes to be zero.

Now we need to check if any other column has missing values, here we found one column which has categorial values which is “browser’

1. In the browser variable we have categorical values like win, mac etc. which denotes windows, mac and some more operating system.
2. To remove missing data from this column we have calculated the occurrence of each value in the variable “browser”.
3. Now we have replaced all the null values in the browser from the value which has most occurrence count.
4. In this way we have handle all the missing and null values in the dataset.

Now to recheck we calculated the sum of all missing values in the dataset which appears to be zero which concludes that the data has been cleaned completely.

```
#Checking count of null values in the data
data.isnull().sum()
```

```
ip               0
date             0
time             0
zone             0
cik              0
accession        0
extention        0
code             0
size         23341
idx              0
norefer          0
noagent          0
find             0
crawler          0
browser      91739
dtype: int64
```

Data Imputation with mean

```python
#Filling NULL data in 'size' by the mean of the all the existing values
data['size'].fillna(data['size'].mean(), inplace = True)
```

```python
#Now checking if size variable consists any null or not
data['size'].isnull().any()
```

```
False
```

```python
#Checking count of null values if still exist in the data
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      91739
dtype: int64
```

Data Imputation with count

```python
#checking counts of different types of browser
data['browser'].value_counts()
```

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

```python
#Assigning variable x to value of most occurring browser type
x=data['browser'].mode()
```

```python
#Imputing value of most occuring browser type in the Null values
data['browser'].fillna(x[0], inplace = True)
```

```python
#checking counts of different types of extention
data['browser'].value_counts()
```

```
win    266626
lin     39130
mie     27931
mac       413
opr        24
fox         7
rim         4
Name: browser, dtype: int64
```

Data is cleansed completely with all the missing and null values

```python
#Now checking if data still has any missing or null values
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      0
dtype: int64
```

After handling data, we encode all the objects into numeric in order to analyze relationship between the variables

```
#Encoding categorical data into numerical data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
for i in ['ip','date','time','accession','extention','browser']:
    data[i] = labelencoder.fit_transform(data[i])
data.head()
```

	ip	date	time	zone	cik	accession	extention	code	size	idx	norefer	noagent	find	crawler	browser
0	2511	0	0	500.0	771252.0	65538	6	200.0	123558.0	0.0	0.0	0.0	10.0	0.0	5
1	3013	0	0	500.0	849778.0	31313	13	200.0	38688.0	0.0	1.0	0.0	0.0	0.0	5
2	1289	0	1	500.0	1244190.0	97785	16	200.0	5683.0	0.0	1.0	1.0	0.0	0.0	5
3	3013	0	2	500.0	849778.0	31313	13	200.0	17038.0	0.0	1.0	0.0	0.0	0.0	5
4	3013	0	3	500.0	849778.0	31313	6	200.0	9025.0	0.0	1.0	0.0	0.0	0.0	5

Since all the variable data is converted into numeric, now we will plot a relationship diagram to analyze relationship between variables

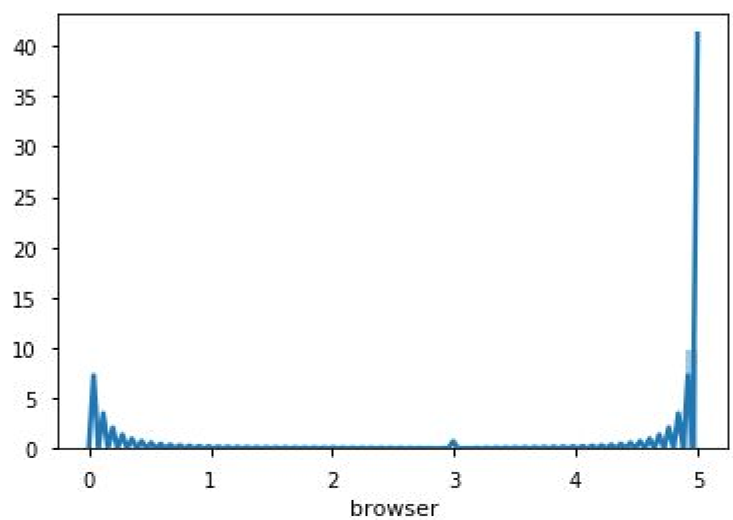
```
#Plotting Correlation and heat map
correlation = data.corr()
sns.set_context("notebook", font_scale = 1.0, rc = {"lines.linewidth" : 2.5})
plt.figure(figsize=(13, 7))
a = sns.heatmap(correlation,annot = True, fmt = '.2f')

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

From the above heat map, we have analyzed correlation between various columns such as between norefer - noagent, crawler – no agent etc.

To analyze more relationship, we plot some density plots

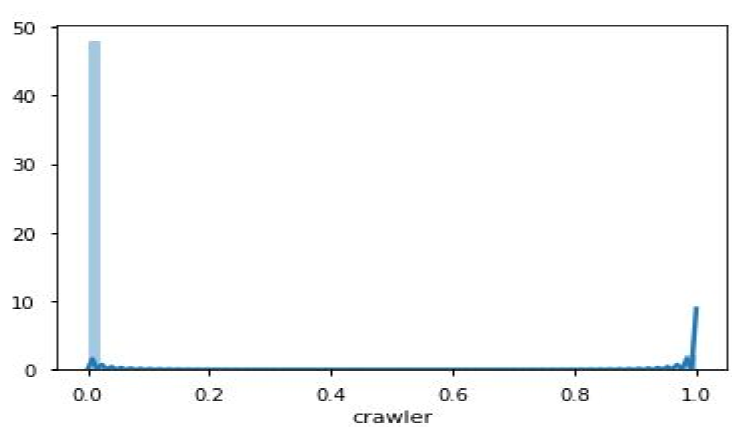
```
#Plotting density plot for browser
sns.distplot(data["browser"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1adc4728550>
```



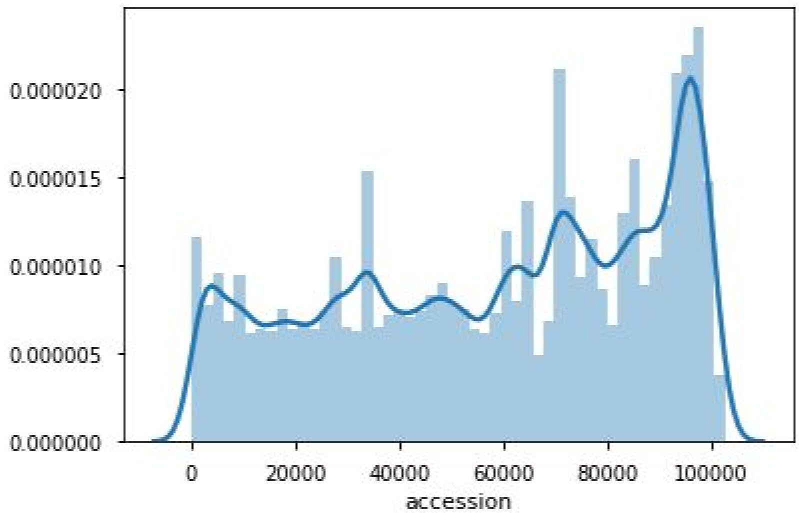

```
#Plotting density plot for crawler
sns.distplot(data["crawler"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cda37c8d30>
```



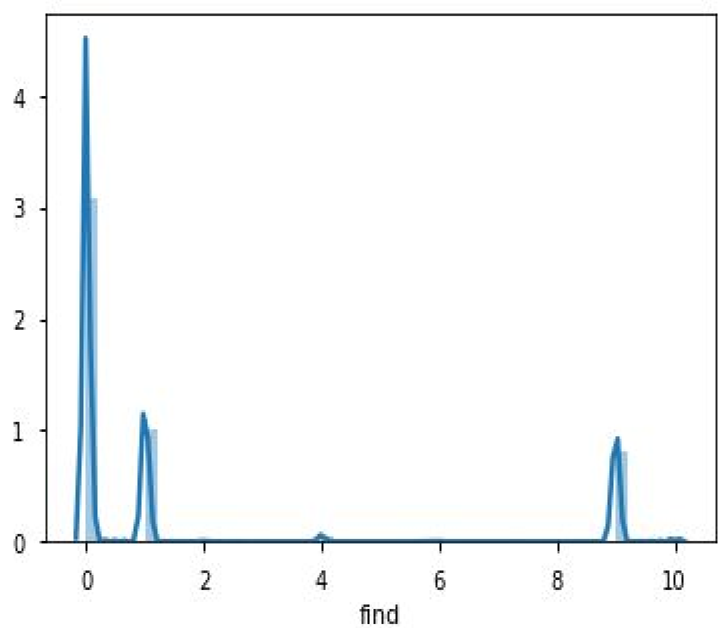
```
#Plotting density plot for accession
sns.distplot(data["accession"])
```

<matplotlib.axes._subplots.AxesSubplot at 0x1adbface320>




```
#Plotting density plot for find
sns.distplot(data["find"])
```

<matplotlib.axes._subplots.AxesSubplot at 0x1cda6baa6d8>



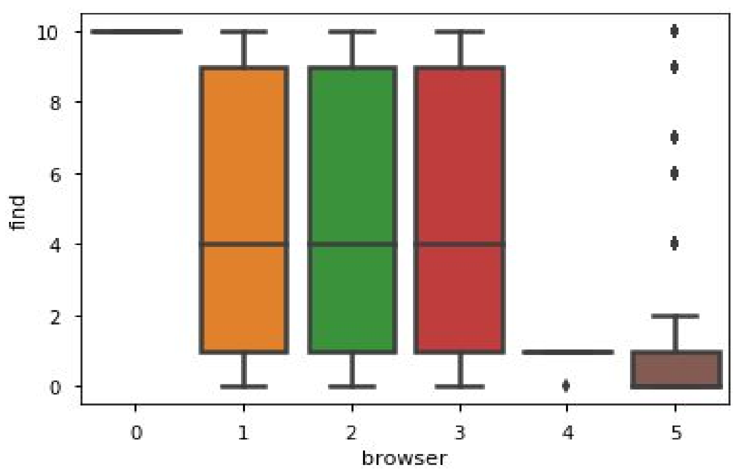
From the above plots we have concluded

1. Most of the data has “Windows” as the potential browser
2. As the most of data is taking 0 value of crawler which means most of users are not webcrawlers or has a user code of 404.
3. Most of the document requested arrived at the document link (internal EDGAR search)

Now Boxplots are plotted to evaluate relationship between variables

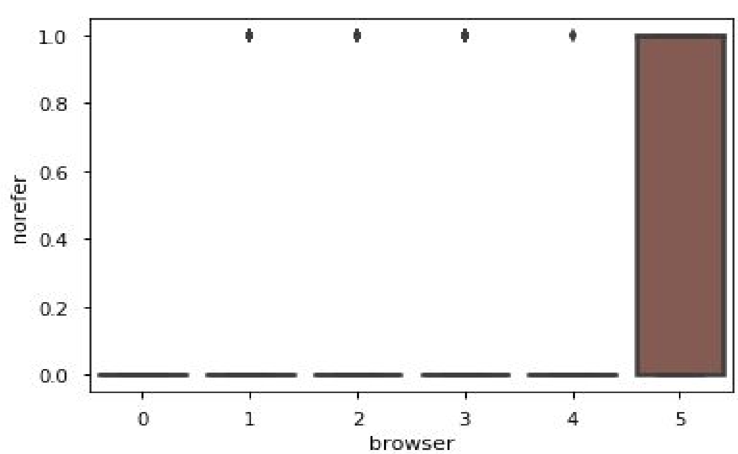
```
#Plotting boxplot to evaluate relationship between variables browser and find
sns.boxplot(x="browser", y="find", data=data_new)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cda71d3c50>
```



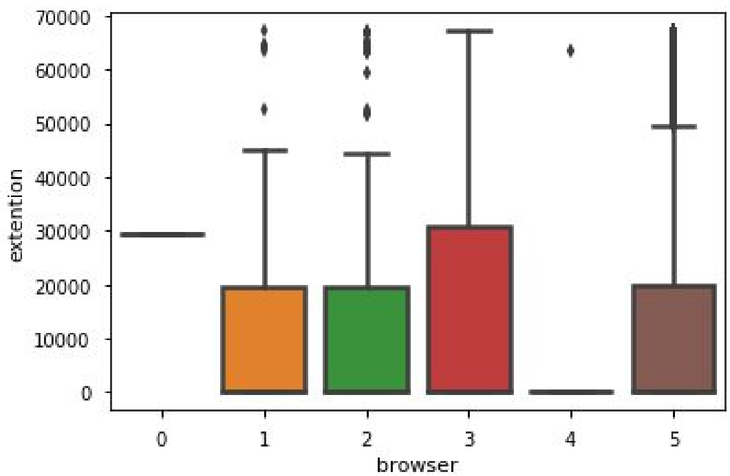

```
#Plotting boxplot to evaluate relationship between variables browser and norefer
sns.boxplot(x="browser", y="norefer", data=data_new)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cda70e8c18>
```



```python
#Plotting boxplot to evaluate relationship between variables browser and extension
sns.boxplot(x="browser", y="extention", data=data_new)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cda5e8ba20>
```



From the above plots we concluded the following:

1. The browser of values 1,2 and 3 are evenly distributed in among the type of document requested arrived.
2. Size of the file is depending on the type of extension.
3. Extension of file are even for the browser such as windows, mac etc.

From above we summarize following metrics from the data

1.Total number of records
2. Maximum size of file
3. Average size of file
4. Most common potential browser
5. Most occurring CIK
6. Most common document extention

These summaries are being calculated for every month and every year for further analysis.