A

Group Project Final Report on

Extracting and Transforming the Compensation Data (DEF14A) from Text and HTML files

By

**Team Rocket**

Bilal Nasir

Chandu Yerragopu

Kiran Kadlag

Siva Manda

Under the esteemed guidance of

Course: ISQS 6339 Business Intelligence

Prof: Dr. David J. Lucus

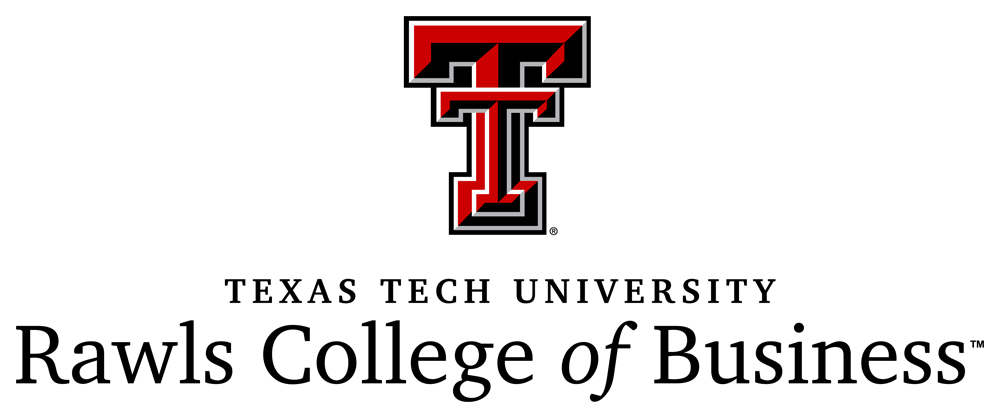


Table of contents

1 Introduction 3

1.1 How the data looks like? 3

1.2 What we want to do with this data? 3

2 List of Main Tasks 5

2.1 Parsing the Text Files 5

Original file table sample (File:0000001750\_1995-08-24.txt) 5

2.1.1 Extracting Table from Text File 5

2.1.2 Source Code for Table Extraction 6

2.2 Parsing the HTML Files 12

2.3 Parsing of footer notes of compensation table from ‘text files’ to ‘CSV’ files 16

2.4 Extracting the summary compensation table headers from text files 21

2.4.1 Logic Pseudo Code and Source Code for Table Header Extraction 23

3 Conclusion and Future Work 27

3.1 Conclusion 27

3.2 Future Work 27

# Introduction

This project titled **“Extracting and Transforming the Compensation Data (DEF14A) from Text and HTML files”** started with an aim of obtaining and studying the compensation related data tables from the huge volumes of data that is scrapped from various websites.

## How the data looks like?

This is the compensation related data that is obtained by scrapping the websites of around 14,000 organizations for a period of 1994 to 2016. The data is scrapped in December 2016.

**Types of files:**

We have worked with this type of files in this project: .txt (free form text), .html or .htm

In certain cases, we have found that .txt file extensions has html tags inside them. Thus, those files have to be treated as HTML for processing rather than a plain text.

**Some facts about source data processing:**

* Total Source Data Files: 118,843
* Total Text Files Parsed for Tables data: 63,650
* Time Process\_1 (Table): 1:11:11 (Hour:Min:Sec)
* Time Process\_2 (Table): 1:17:32 (Hour:Min:Sec)
* Total Time (Table extraction) 2 hours 28 minutes.

**Objectives completed:**

## What we want to do with this data?

*"Information is the oil of the 21st century, and analytics is the combustion engine” (Peter Sondergaard, Senior Vice President, Gartner)*

As rightly said in the above statement, when we can unhide the information present in this huge volume of data it has a lot of potential in understanding concepts and planning the next steps. The main point of interest in this data is to obtain the “Summary Compensation Tables” from each of the files and then analyze the data in those files to better understand the Executive moment across various organizations. The moment of executives across organizations may have some connection with the compensation that is offered by a company and with this research we will be able to understand the same.

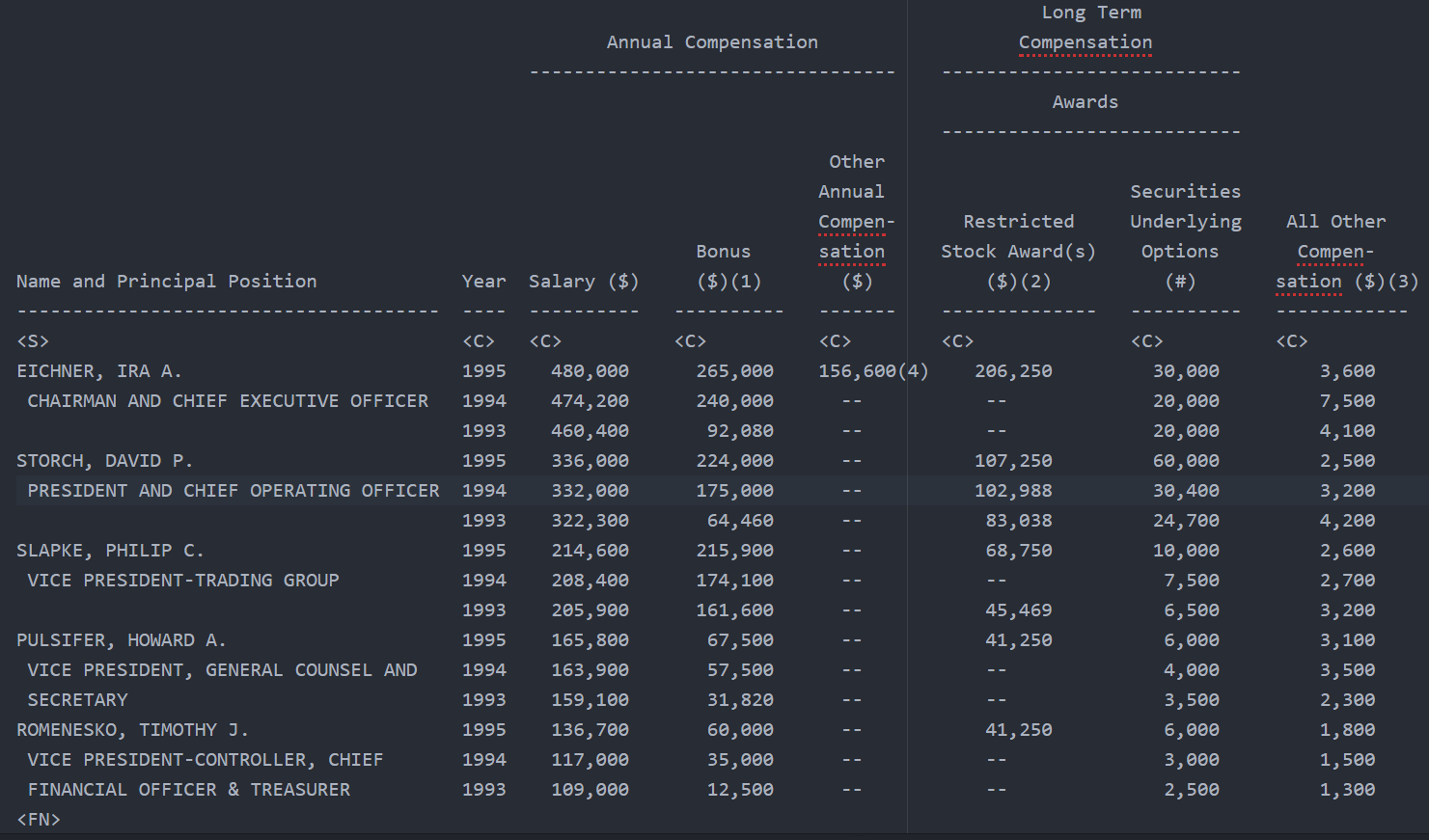
# List of Main Tasks

## Parsing the Text Files

The text files had executive compensation summary table embedded in each document along with its footnotes. We divided the work of extraction, cleaning and parsing of the table contents and footnotes contents into separate processes. Each task was taken by individual team members for its solution development. For text files we have two major tasks:

1. Extracting the compensation table
2. Extracting the footnotes of each table

### Original file table sample (File:0000001750\_1995-08-24.txt)



### Extracting Table from Text File

The table extraction goal was to parse and clean data correctly and load it into a CSV file. The CSV file of table may be used in future research purposes, and for loading into a database. The following process flow diagram illustrates the process in more detail:

DEF14A (CIK\_Date)

Stage\_Table.CSV

Final\_Table.CSV

.TxT ?

Python: Extract / Clean/Parse

Python(Dataframe):

Split Columns / Parse / Remove empty cells

Yes

### Source Code for Table Extraction

The python source code for this task is in two files as below:

* File 1: (Project\_Bilal4\_Txt1\_Parser\_Final.py) Extract the table from original DEF14A and save as a CSV file.
* File 2: (Project\_Bilal4\_Txt2\_Name\_Split\_Final.py) Further processing for splitting columns and formatting of the data in file.

#### File#1:

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 4 00:56:10 2018

@author: Muhammad Bilal Nasir: MS Data Science (Class of 2018)

"""

###############################################################################

# PROJECT WORK :OPEN,READ,PARSE WRITE TEXT FILE WITH REGEX

###############################################################################

#import pandas as pd

import os as os

import re

import csv

import numpy as np

from datetime import datetime

#Some Initializations (time, regular expressions, input filecounter)

startTime = datetime.now()

regex1 = re.compile('\s+')

file\_num=0

file\_read\_num=0

# Iterating my Files/directory structure

filepath = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/subset/'

filepathResults = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/Results/'

#filepath = 'D:/def\_14a\_data/data\_parsed/'

#filepathResults = 'D:/def\_14a\_data\_txt\_parsed/'

nPath=[] # Intialize

fIdx=0 #start index of filenames

for root, dirs, files in os.walk(filepath): # walk returns a tuple

for fIdx,name in enumerate(files):

nPath.append(os.path.join(root, name))

nPath=[row.replace('\\', '/') for row in nPath] #fixing my dir path

filename=name #List of files only

filename=filename.split(".")

fileExt=filename[1]

filePart1=filename[0].split("\_\_")

CIK=filePart1[0] # converting to string to retain the leading zeros of CIKs

Date\_def14a=filePart1[1]

if fileExt=="txt":

# Open File and read Contents

#file.close()

file = open(nPath[fIdx], 'r') #looping into the names

filedata=file.readlines()

startDataIdx = 0 # Data row index initilization

strtIDx=0 # Column row index initialization

Titleidx=0

#Filedata READER

for i,line in enumerate(filedata): #enumerating to get the tuple with index

#print(line)

if (line.find('COMPENSATION') > -1):

Titleidx=i

if ((line.find('Year') > -1 or line.find('YEAR') > -1) and (line.find('Salary') > -1 or line.find('SALARY') > -1) and (line.find('Bonus') > -1 or line.find('BONUS') > -1)):# and i>Titleidx and Titleidx!=0):

strtIDx=i #header row index

# Using Split

colheadingl10=[x.strip() for x in line.split(' ')]

colheadingl110=[x.strip() for x in filedata[strtIDx].split(' ')]

colheadingl20=[x.strip() for x in filedata[strtIDx-1].split(' ')]

colheadingl30=[x.strip() for x in filedata[strtIDx-2].split(' ')]

colheadingl40=[x.strip() for x in filedata[strtIDx-3].split(' ')]

#Removing empty cells- colheadingl10=colheadingl10.remove(' ')

colheading10 = list(filter(None, colheadingl10)) #Final String removing empty cell from the header

colheadingl110 = list(filter(None, colheadingl10))

colheadingl20 = list(filter(None, colheadingl20))

colheadingl30 = list(filter(None, colheadingl30))

colheadingl40 = list(filter(None, colheadingl40))

print(fIdx) # File index in each folder - for display/debugging

#if len(colheadingl10)!=0:

colheadingl110.append("CIK")

colheadingl110.append("Date\_DF14A")

#print(colheadingl110)

fileResult=filepathResults+str(filename[0])+'.csv'

myfile=open(fileResult, 'w')

wr = csv.writer(myfile,lineterminator='\n',delimiter=',') #quoting=csv.QUOTE\_ALL

wr.writerow(colheadingl110)

if (line.find('<S>') >-1 and line.find('<C>') >-1) and i > strtIDx and i <strtIDx+5 and strtIDx!=0: # Range check that these tags lie near the title row

startDataIdx = i

if i > startDataIdx and i < startDataIdx+50 and startDataIdx!=0:

#print(regex1.split(line))

#print(line)

record=[x.strip() for x in line.split(' ')] # Strip space levels set to 2 spaces

if len(record)>1: # if the record line is not empty

record1 = list(filter(None, record)) # Removing the empty cells in the list

#Replacing the -- value with empty string

record1= [w.replace('--', '0') for w in record1]

record1= [w.replace('-0-', '0') for w in record1]

record1= [w.replace('$', '') for w in record1]

record1= [w.replace(',', '') for w in record1] #re.sub(r'[^\w]', '', s)

record1= [w.replace('..', '') for w in record1]

record1= [w.replace('. .', '') for w in record1]

record1= [w.replace(' .', '') for w in record1]

record1= [w.strip() for w in record1] #removing the spaces between numeric (bonus and salary) left in case

#Filtering out the empty cells

record1 = list(filter(None, record1))

#Case: To split Title/Name String for Years

for lidx,w in enumerate(record1):

splitCell=w.split(' ') # split on 2 spaces and load a series of two elements

if len(splitCell)>1:

record1.insert(lidx+1,splitCell[len(splitCell)-1]) # My Magic potion Add the splitted value to a new next cell

record1.pop(lidx) # My Magic potion Add the splitted value to a new next cell

record1.insert(lidx,splitCell[0])

#print("SPLITING LIST CELLS:"+str(splitCell)+" BETWEEN: "+str(record1[lidx])+":AND:"+str(record1[lidx+1]))

#Case: First cell split if it has year at the end of the name and load to next cell

if len(record1)>1 and len(record1[1])!=4: #len(re.findall('([0-9]+)',record1[1])[0]): # alternatively len() !=4

splitCell2 = re.split(r'\s+(?=\d)|(?<=\d)\s+',record1[0]) #re.split(r'\s+(?=\d)|(?<=\d)\s+', s)

if len(splitCell2)>1: #= [re.split(w) for w in record1]

record1.insert(1,splitCell2[len(splitCell2)-1]) # My Magic potion Add the splitted value to a new next cell

#print("SPLITING LIST CELLS:"+str(splitCell2)+" BETWEEN: "+str(record1[0])+":AND:"+str(record1[0+1]))

#Case: moving the year at the correct column

if len(record1)>1 and len(record1[1])!=4 and len(record1[0])==4:

record1.insert(0,np.NaN)

#Appending CorporateID and Date Column

if len(record1)>3:

record1.append("=\""+CIK+"\"") #To retain the leading zeros OR ("\*"+CIK)

record1.append(Date\_def14a)

#Case: To fix record position in case name/title is empty but more data in trailing years

#for lidx,w in enumerate(record1):

if len(colheadingl110)>len(record1)-2 and len(record1)>2: # check id the record is short than the columns and more than 1 cells

diff=len(colheadingl110)-len(record1)

for s in range(diff):

record1.insert(s,np.NaN) # inserting Null at index s

# Checking the end of Table identifiers

if len(record1)>0 and (str(record1[0])!='<FN>' or str(record1[0])!='</TABLE>') and i<startDataIdx+40:

wr.writerow(record1)

#print(record1)

# Table End Check and loop exit

if (line.find('<FN>') >-1) or (line.find('</TABLE>') >-1 and i > startDataIdx and i < startDataIdx+50):

#Closing File Handles for this Read and Write

myfile.close()

file.close()

Titleidx=0

strtIDx=0

startDataIdx=0

#print('FILE SAVED:'+fileResult) #uncheck to check the file written (htmls will be skipped)

#file counter

file\_num+=1

print("Files Done:"+str(file\_num)) # Total files parsed

break # Get out and get a new file.

#Close any opened file at the end

myfile.close()

file.close()

#file counter

file\_read\_num+=1

print("Total Files Ckecked:"+str(file\_read\_num)+", Cumulative Time: "+str(datetime.now() - startTime))

#### File #1 Output (.csv) file sample (0000000020\_\_1997-03-20.csv)



#### File #2 Source Code.

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 20 00:56:10 2018

@author: Bilal Nasir- MS Data Science Fall 2018

"""

####################################################################################

# PROJECT WORK :OPEN,READ,PARSE WRITE TEXT TO SPLIT THE Name and Title

####################################################################################

import pandas as pd

import numpy as np

import os as os

from datetime import datetime

#Variable Initializations

startTime = datetime.now()

file\_num=0

# Looping my File structure

#filepath = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/Results/' #'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/subset/'

#filepathResults = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/ResultsFinalTxt/'

filepath = 'D:/def\_14a\_data\_txt\_parsed/'

filepathResults = 'D:/def\_14a\_data\_txt\_parsed\_final/'

nPath=[] # Intialize

fIdx=0 #start index of filenames

for root, dirs, files in os.walk(filepath): # walk returns a tuple

for fIdx,name in enumerate(files):

nPath.append(os.path.join(root, name))

nPath=[row.replace('\\', '/') for row in nPath] #fixing my dir path

filename=name #List of files only

filename=filename.split(".")

fileExt=filename[1]

filePart1=filename[0].split("\_\_")

corpID=filePart1[0]

reportDate=filePart1[1]

if fileExt=="csv":

# Open File and read Contents

df1 = pd.read\_csv(nPath[fIdx], index\_col=False)

df1.columns

len(df1.columns)

# Selecting 2 columns

df2 = df1.iloc[:,0:2].replace(np.NaN,0)

df2.iloc[:,1] = df2.iloc[:,1].astype(int) # converting years(str) to numbers

HYear=df2.iloc[:,1].max()

# creating new column for Titles

df2["Names"]=""

df2["Titles"]=""

Title=""

Name=""

rindex=[]

for index, row in df2.iterrows():

if row[1]==HYear:

#print(row[1]) #uncomment for Debuging

Name=row[0]

#print(Name) #uncomment for Debuging

df2.iloc[index,2]=Name

NameIdx=index

rindex.append(NameIdx)

if (row[1]<=HYear and index>NameIdx) or (row[1]=="" and index>NameIdx and row[0]!=""):

Title=row[0]

# Check zero for title

if Title==0:

Title=df2.iloc[index-1,0]

#print(Title) #uncomment for Debuging

df2.iloc[index,2]=Name

df2.iloc[index,3]=Title

Title=""

if (index==NameIdx+1) or (row[1]=="" and index==NameIdx+1 and row[0]!=""): # To append the title infront of the "Name" Cell.

Title=row[0]

if Title!="":

df2.iloc[index-1,3]=Title

else:

df2.iloc[index-1,3]="Missing"

Title=""

# Final data frame for Name and Titles only

df3=df2.iloc[:,2:]

# Droping the first column from the original DF

df1.drop(df2.columns[[0]], axis=1, inplace=True)

# append the New DF with the old DF.

result1 = df3.join(df1)

########### To remove the empty rows in the table ##########

Emp\_row=[];

for index, row in result1.iterrows():

if pd.isnull(row[3]): #(pd.isnull(df[1]))# result1.isnull().values.any()

#print("Empty Row") #uncomment for Debuging

Title\_Rem=row[1]

if pd.notnull(result1.iloc[index-1,3]): #iloc[row\_idx,column\_idx]

result1.iloc[index-1,1]+=" "+Title\_Rem

Emp\_row.append(index) # loading indexed to be removed

elif pd.notnull(result1.iloc[index-2,3]):

result1.iloc[index-2,1]+=" "+Title\_Rem

Emp\_row.append(index)

elif pd.notnull(result1.iloc[index-3,3]):

result1.iloc[index-3,1]+=" "+Title\_Rem

Emp\_row.append(index)

elif pd.notnull(result1.iloc[index-4,3]):

result1.iloc[index-4,1]+=" "+Title\_Rem

Emp\_row.append(index)

#Droping the row having empty cells and moving titles up

result2=result1.drop(Emp\_row) #result1.index[Emp\_row]

#print("\*DELETING Empty Rowssss") #uncomment for debuging

#Replacing Zeros in the Title column with Title from the earlier cells

for index, row in result2.iterrows():

if row[1]==0 and row[2]>=HYear: # If the cell=0 and row is not Maximum Year

#replacing the cell value with the title of the person.

result2.iloc[index,1]=result2.iloc[index-1,1]

#print(result2.iloc[index,1]) #Uncheck for

if row[1]==0 and row[2]==HYear: # If the cell=0 and row is not Maximum Year

#replacing the cell value with the title of the person.

result2.iloc[index,1]=result2.iloc[index+1,1]

#print(result2.iloc[index,1])

fileFinalresult=filepathResults+str(filename[0])+'\_finaltxt'+'.csv'

result2.to\_csv(fileFinalresult, sep=',', encoding='utf-8', index=False)

#Process End Time

file\_num+=1

print("Files Done:"+str(file\_num)+", Cumulative Time: "+str(datetime.now() - startTime))

#### File #2 Output (.csv) sample (0000000020\_\_1997-03-20\_finaltxt.csv)



The final out put above shows two new columns as “Name” and “Titles”. Secondly the empty rows have been removed.

## Parsing the HTML Files

**Problem Statement:**

As mentioned in the previous section, we have both text as well as HTML files and some of the files with .txt extension has HTML tags inside them. The task is to obtain the “Summary Compensation Table”from those files.

**Challenges**

* There is no much similarity between various html files so that same logic works for all the files.
* There is no much similarity between the .txt files as well to uniquely identify the table.
* Exporting the obtained table to .csv in a cleaner format is often difficult.

**Dealing with the problem statement:**

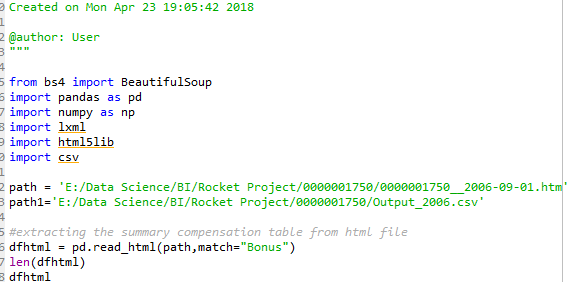
We have tried 2 approaches here to obtain the “Summary Compensation Table” from the html tags. They are:

1. Strip off all the HTML tags and then treat the output file with plain text as use that file for obtaining the Summary Compensation Table.
2. Use the python packages and functions to obtain the table.

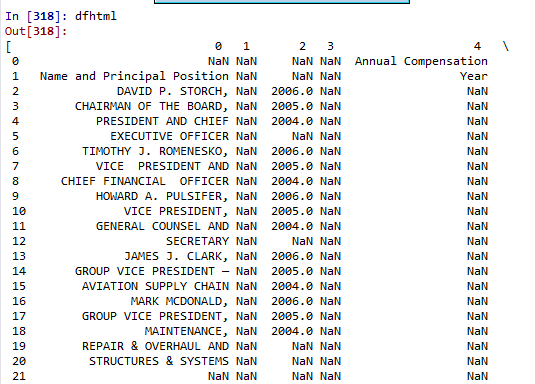
Since the first approach needs an intermediate file storage and further processing, we have decided to use the 2nd approach.

The code for that is as follows:

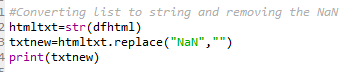
**Code:**



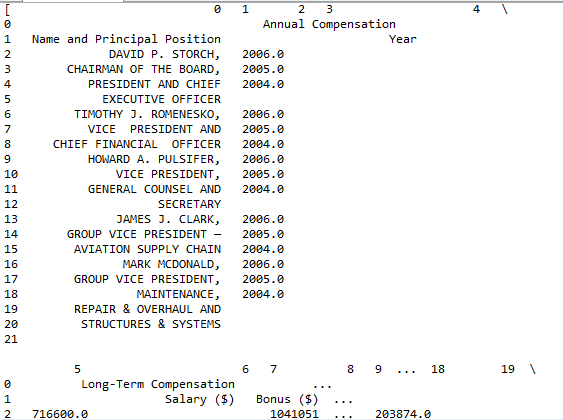
The above code obtains the “Sumary Compensation Table” into dfhtml variable. The sample output value looks like below:



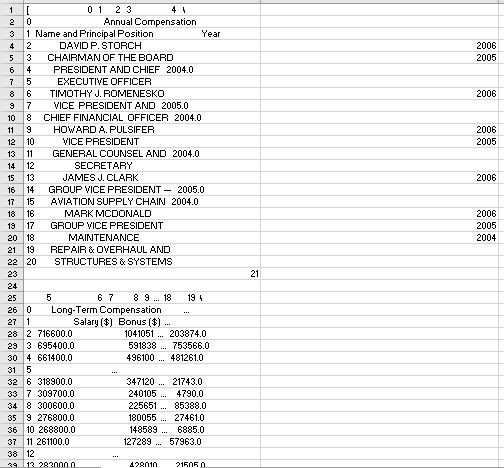
The next step is to remove the ‘NaN’ from dfhtml. The code for that is as follows:



The above code removes the NaN from the dfhtml and looks like below:



Now the next step is to save this output into a .csv file. When we tried saving the above output into a .csv file, it is not properly formatted. The sample output is as follows:



**Next Steps:**

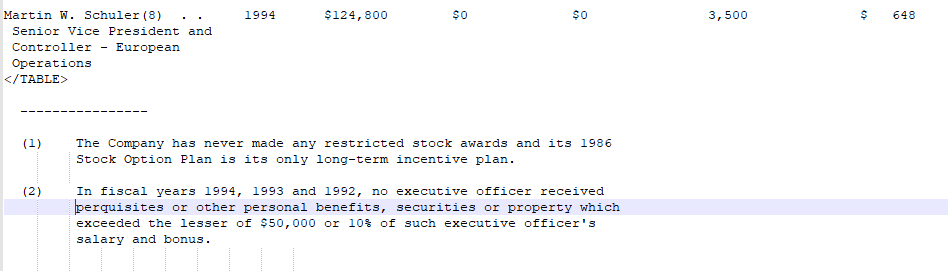
The output table has to be formatted with proper row and column alignments so that when we try to save it into a .csv file, it is saved in a proper format.

Thus, this section explains how the required “Summary Compensation Table” is obtained from the HTML tags present in both .html as well as .txt files with HTML tags.

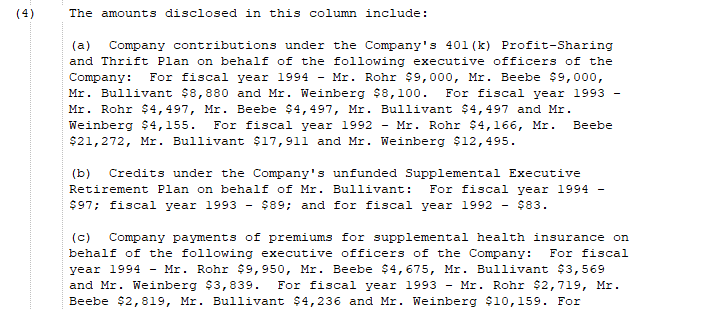
## Parsing of footer notes of compensation table from ‘text files’ to ‘CSV’ files

Many Compensation table have table summary and comments at the end of table, which includes promotion information, changes in salary structure, bonus and other salary benefits changes within a particular year etc. These notes also include 401(k) benefits details as per individual officer. This data is important to analyze the salary growth and benefit distribution as per year. While analyzing the salary data, this information will add value to understand promotional pattern and reasons to changes in salary.

**Sample Input file:**



These notes vary from 0 to many and size of comments are not fixed. Some notes having sub notes marked under the main note.



**Problem Statement:**

Parse all these notes under the summary table into csv file as per file date and company code. To develop a solution which will work for all files in database to get output, which will be used for data analysis.

**Challenges:**

* There is not particular start for foot notes in text file
* The size and number of notes are not fixed or similar
* The numbering to notes are also in string and which are not standard in all files.
* Some notes are divided in to two or more pages which should get captured in a solution program.
* There is no fixed space between two notes.

**Solution:**

**1. Read all files in database path folders one by one to find required text from each file**

**Steps:**

* Read the directory at given path
* Read all folders and files in parent directory
* Read only text files from child directory
* Set path for database and list all files and folders

**Methods and package:**

* ‘os’ package and ‘listdir()’ function to find all directories
* ‘os.path.isdir()’ – to check if respective output is directory or text file
* ‘os.chdir()’ – setting the path of respective directories to perform operation in particular directory
* ‘os.path.join()’ – to join path with new inside directory to access location

Nested ‘for’ loops for reaching all folders and files in a location.

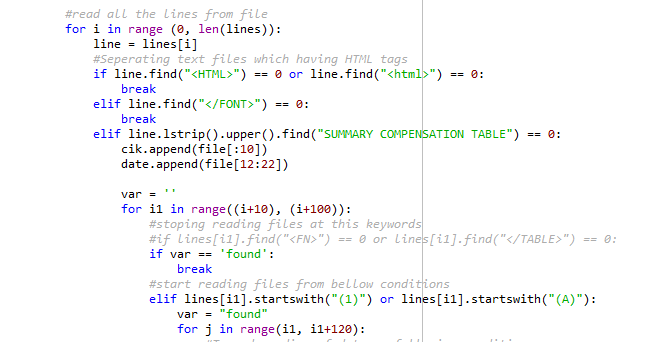


**2. Check files in folders are ‘HTML’ or ‘TEXT’ format, if file is text file then find compensation table**

**Solution:**

Functions:

* lstrip() :- remove the spaces before start of line to compare the string in ‘if condition’
* upper() :- convert the text in capitals to match with extract string
* break :- break ‘for’ loop on condition



Above part of program reading footnotes where it will find ‘(1)’ or ‘(A)’ within next 100 lines from compensation table header. ‘for’ loop starting from 10th line of table ‘for i1 in range((i+10), (i+100)):’ to avoid ‘(1)’ or ‘(A)’ in header of the table

**3. Constraint to stop reading file at the end of footnotes and inserting all data in list**



Many of conditions in ‘if’ statement is to find out the end of footnotes. If we find more data in foot notes other than comments, then we can add more satisfying conditions in ‘if’ statement

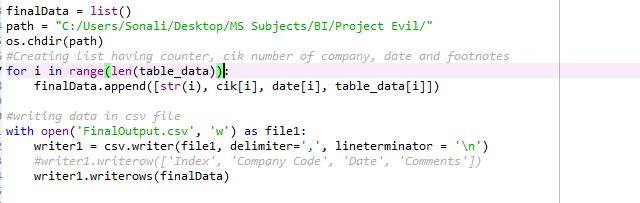
**4. Writing CIK code, date, foot notes in csv file**

**Steps:**

* Appends all comments in one list variable
* Setting index for each company
* Writing company code and date in a list from name of file

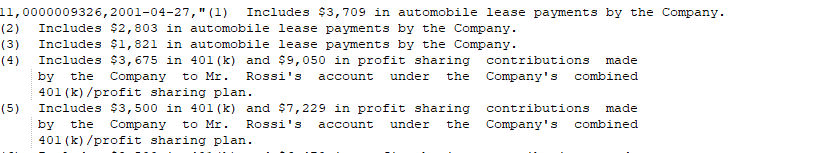


Following code setting code and writing respective CIK code, date and footnotes in a csv file by running a ‘for’ loop from range of 1 to length of list

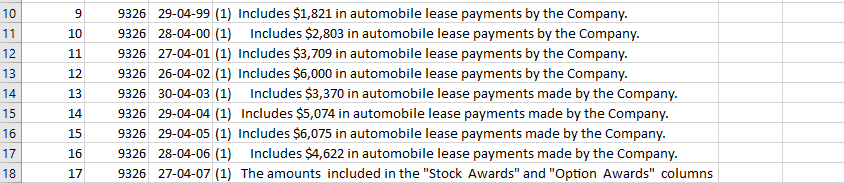


**Output:**

Opened in Notepad++ to see detail comments:



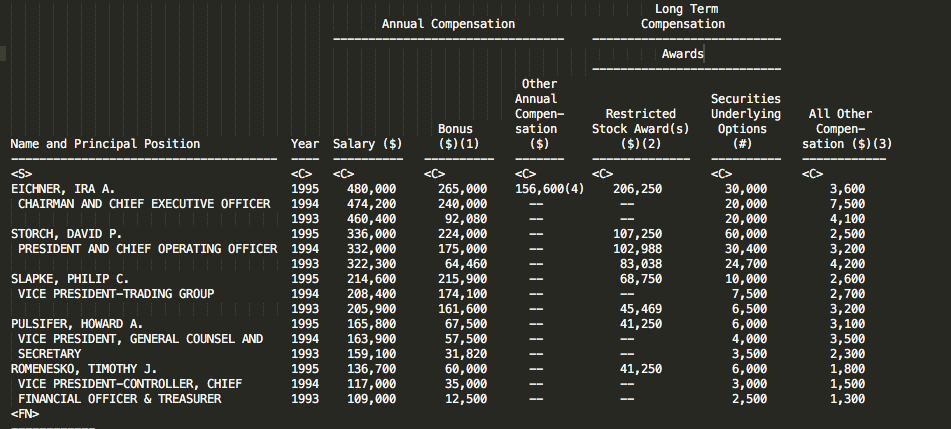
In Microsoft Excel:



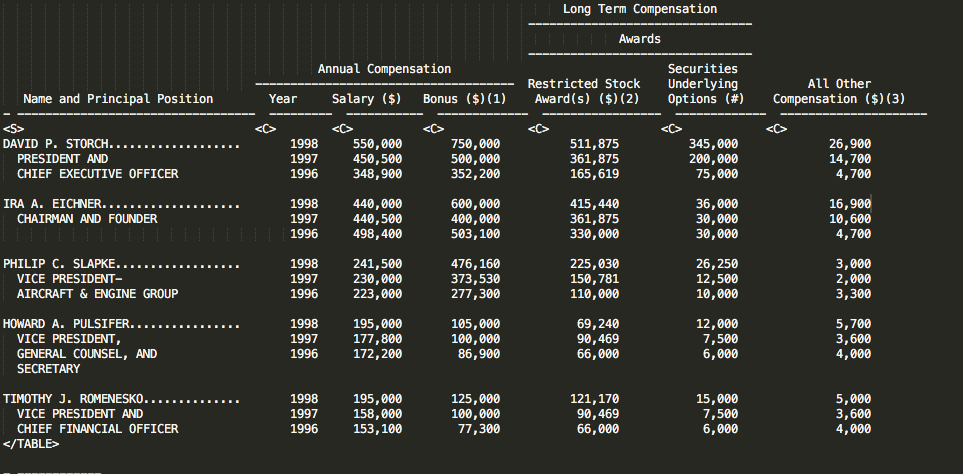
## Extracting the summary compensation table headers from text files

The goal of this task was to attempt and extract only the column headers of the tables in the text files and then load them into a separate CSV file than the ones with a full table. The challenge here is that the tables column headers are in a different format in each file, with a different number of spaces between column names and some column headers are spread over more number of lines per file than others. For example, let’s take a look two different text files in the file directory.

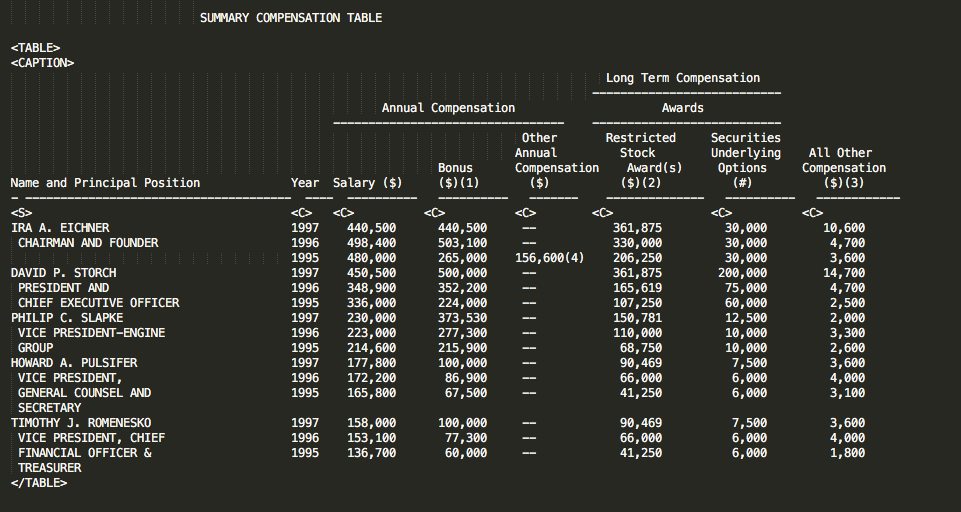
**File table sample 1** (File:0000001750\_1995-08-24.txt)



**File table sample 2** (File:0000001750\_1998-08-21.txt)



**File table sample 3** (File:0000001750\_1997-08-28.txt)



Here, we can see there is significant spacing between “Annual Compensation” and the next column headers in file table sample 1 and not as much in file table sample 2. Also in file sample 1, “Bonus ($)(1)” is spread over two lines and in file sample 2 “Bonus ($)(1)” occupies only one line. This formatting is a big challenge in extracting the column headers. This type of formatting of spacing again has another format in file sample 3 as we can see above.

### Logic Pseudo Code and Source Code for Table Header Extraction

The python source code for parsing the text file, extracting table headers and writing to the CSV is in **text\_headers.py.**

Here is the general logic of this file:

For all files in input file directory

If file is a text file

Open file and read contents

Find Summary Compensation Table

Loop and find the table headers

Put table headers column names into list

Write List into a CSV file

#### text\_headers.py:

import pandas as pd

import os as os

import re

import csv

import numpy as np

from sqlalchemy import create\_engine # Hit any Database server Required for Panda data load

#Project read files script

regex1 = re.compile('\s+')

# TESTING SINGLE FILE: PATHS asn code CODE

#filepath = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/subset/0000001750/'

#filepathResults = 'C:/Users/bilal/Documents/TTU/Courses/BusinessIntelligence/Project/Evil\_Project\_Data/Results/'

#fileResults = '0000001750\_\_1994-08-30.csv'

#file = open(filepath+'0000001750\_\_1994-08-30.txt', 'r')

#filedata=file.readlines()

# LOOPING my File structure

#Users/siva/Documents

filepath = '/Users/siva/Documents/Evil\_Proj/Input\_files'

filepathResults = '/Users/siva/Documents/Evil\_Proj/proj\_results/'

#fileResults = '0000001750\_\_1994-08-30.csv'

nPath=[] # Intialize

for root, dirs, files in os.walk(filepath): # walk returns a tuple

for name in files:

nPath.append(os.path.join(root, name))

nPath=[row.replace('\\', '/') for row in nPath] #fixing my dir path

filename = name # List of files only

filename = filename.split(".")

fileExt = filename[1]

filePart1 = filename[0].split("\_\_")

corpID = filePart1[0]

reportDate = filePart1[1]

print(filename)

#print(filePart1)

if fileExt == "txt":

# Open File and read Contents

#file.close()

file = open(nPath[0], 'r')

filedata = file.readlines()

#filedata

for i,line in enumerate(filedata): #enumerating to get the tuple with index

#startDataIdx = 0 # Data row index initilization

#strtIDx=0 # Column row index initialization

if (line.find('Year') > -1 and line.find('Salary') > -1):

print(line)

print(i)

strtIDx = i #header row index

print(filedata[strtIDx-1])

print(filedata[strtIDx-2])

# Using Split

colheadingl10 = [x.strip() for x in line.split(' ')]

colheadingl110 = [x.strip() for x in filedata[strtIDx].split(' ')]

colheadingl20 = [x.strip() for x in filedata[strtIDx-1].split(' ')]

colheadingl30 = [x.strip() for x in filedata[strtIDx-2].split(' ')]

#Removing empty cells- colheadingl10=colheadingl10.remove(' ')

colheading110 = list(filter(None, colheadingl110)) #Final String removing empty cell from the header

colheadingl110 = list(filter(None, colheadingl10))

colheadingl20 = list(filter(None, colheadingl20))

colheadingl30 = list(filter(None, colheadingl30))

colheadingl10

colheadingl20

colheadingl30

l1 = len(colheadingl10)

l2 = len(colheadingl20)

l3 = len(colheadingl30)

#colheadingl11[l1-1] = colheadingl3[l3-1]+colheadingl2[l2-1]+colheadingl1[l1-1]

for lastidx in range(l2):#max(l2,l3)

#print(colheadingl20[-(lastidx+1)]) # Merging the split headers back for column name

colheadingl110[-(lastidx+1)] = colheadingl20[-(lastidx+1)]+' '+colheadingl10[-(lastidx+1)]

#print(colheadingl110)

#Lhead=len(colheadingl110)

#Lhead

#Building Dataframe and writing it to a CSV File

#df = pd.DataFrame(some\_list, columns=["colummn"])

#df = pd.DataFrame(columns=colheadingl110) # Not required

# Using csv library

fileResult = filepathResults+str(filename[0])+'.csv'

myfile = open(fileResult, 'w')

wr = csv.writer(myfile,lineterminator='\n') #quoting=csv.QUOTE\_ALL

wr.writerow(colheadingl110)

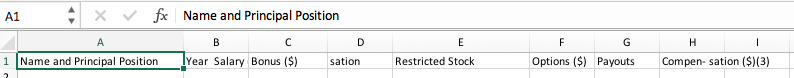
myfile.close()

file.close()

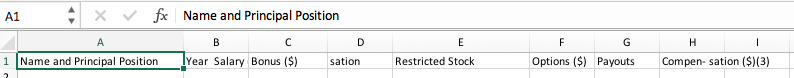
print('FILE SAVED:' + fileResult)

break # Get out and get anew file

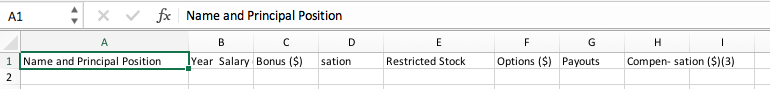
#### Text File Sample #1 Output (.csv) file sample (File:0000001750\_1995-08-24.csv)



#### Text File Sample #2 Output (.csv) file sample (File:0000001750\_1998-08-21.csv)



#### Text File Sample #3 Output (.csv) file sample (File:0000001750\_1997-08-28.csv)

****

Due to the challenges we see when looking at the formatting of the table column headers in our files, our script does not perfectly extract the file column headers from these text files. In the future, how to account for the differing multiple lines of text and the spacing between column headers should be worked on in the code.

# Conclusion and Future Work

## Conclusion

In this project, our main goal was to extract data and information from SEC DEF14A files and export various components of these files into CSV files. This was broken up into 4 major tasks that we carried out: Extracting the “Summary Compensation” table from text files, Extracting the “Summary Compensation” Table from HTML files, Extracting and enumerating the “Summary Compensation” table headers, extracting the footnotes for the executive “Summary Compensation” table. We were able to successfully extract the footnotes for this executive compensation table and put the resulting footnotes into a result CSV file for each of the thousands of files that we had.

Extracting the “Summary Compensation” table from the text files also proved to be pretty successful, as we were able to properly parse and extract these tables from the text files and put them into result CSV files while preserving the structure of the tables and dividing the components into cells of the CSV appropriately. Getting the column headers of these tables now from text and HTML files proved to be very challenging, due to the different format and spacing in the executive “Summary Compensation” Table between each of the files, and there were over 118,000 files so it was tough to create an algorithm that could perfectly perform this task for this many files. Also, we faced difficulties in extracting the executive “Summary Compensation” table form HTML as we used pandas and read\_html() to get this table into a list in python, but then had trouble exporting this to a CSV as our attempts would put the entire table into one cell. We tried transferring the table to a text file as an intermediary and then exporting it to CSV from the text file, but this did not work either. From our work and the project, we can conclude that seem of the tasks were successfully completed and some of the problems were able to be address, but due to the very large number of fires and the complexity of the data and difference in formatting between files, there still is a lot of work required to effectively get this data ready for any sort of analytics to be performed on it.

## Future Work

Reflecting on what issues we faced with extracting and cleaning the DEF14A data and what tasks were able to be completed, we saw that extracting the “Summary Compensation” table from text files and its footnotes form text and HTML files and putting them in result CSV files was a task that looks to have been successfully completed. However, due to time constraints of the project and the large number of files present (118,843) we were not able to look through every input file and compare them to result CSV files to properly ensure our scripts worked for every single file that we have. In the future, this is something that should be done to make sure the data was properly cleaned and extracted so that any analytics run on the data provides the most accurate results. This could show that our scripts for these parts might need to be changed or perfected a little. The tasks we faced the most challenges for with were extracting and enumerating the table column headers from both text and HTML files for the executive “Summary Compensation Table” and extracting the executive “Summary Compensation”. The challenge we faced with extracting the column headers was that each of the files had different formatting of the table headers as seen in section 2.4, so it was very tough to make an algorithm and write a script for this as there were 118,843 files which had different formats and we did not know what the structure was in all of them. This is something that should be looked at the future and will require a team of people to properly address this issue alone. In extracting this table from HTML, we were able to extract the table just fine into a list in python, but writing it to a result CSV with proper formatting was the problem we weren’t able to solve. This should also be address in the future which might require some more research on how to write list data to CSV while maintain the proper structure of the table for all the HTML files, which have differing formats and structure of the table.