Data Processing Report

UNDERSTANDING FINANCIAL REPORTS USING NATURAL LANGUAGE PROCESSING

April 14, 2019

Supervisor: **Dr. RuiBang Luo**

Saripalli Varun Vamsi- UID : 3035242229
Tarun Sudhams- UID : 3035253876

Writen by Saripalli Varun Vamsi

Abstract

Credit Derivatives are considered excellent tools to hedge the credit risk of an underlying entity from one party to another without actually transferring the ownership of the entity. One such hedging tool is called Credit Default Swap (CDS), which are often known to be responsible for the 2007-2008 financial crisis. Upon further investigation, it was found that lack of regulation and information on how CDS works were the main culprits behind the crisis. Post the crisis, the United States Securities and Exchange Commission(SEC) has requested for frequent and more detailed reporting from the mutual funds about their current position on these derivatives. Given the lack of strict format for the report, it becomes extremely difficult to extract information from these reports and conduct in-depth analysis on how the mutual funds leverage credit derivatives and in particular, CDS.

This project aims at consolidating all the mutual fund holding reports on Credit Default Swap positions and investigating how mutual funds leverage credit derivatives by first scraping all the publicly available reports from the SEC using python with web scraping package Beautiful Soup. We then go through all the reports and extract relevant CDS information which is in both structured and unstructured format. The structured data is extracted in tables using python which is then processed and formatted to create a unified csv file containing all the tables. The unstructured information is also cleaned and then used for Natural Language Processing to understand the context of the sentences used in the reports and to extract the key entities from the sentences.

Furthermore, this report aims at providing a detailed motivation behind our project, and also explains in detail the methodology behind the data extraction of the structured data from the reports along with its processing and the creation of the unified csv file and also the preparation of the training data for use in Natural Language Processing. This enables us to successfully

create a database to empower future research studies. We also go over the key challenges and limitations faced by our team. Finally, we look at the results achieved.

Acknowledgment

We would like to thank our supervisor Dr.RuiBang Luo for his continuous guidance throughout the project by providing his expert opinion on various problems that we faced. We are also grateful to Dr.Ricky Ma to help us setup and troubleshoot the development environment.

Finally, we want to thank our co-supervisor Jun Yu Wang, who helped us with complex finance terminologies and supported us in creating a motivation for this project from a finance perspective.

Contents

List of Figures

List of Abbreviations

1. CDS - Credit Default Swap
2. NLP - Natural Language Processing

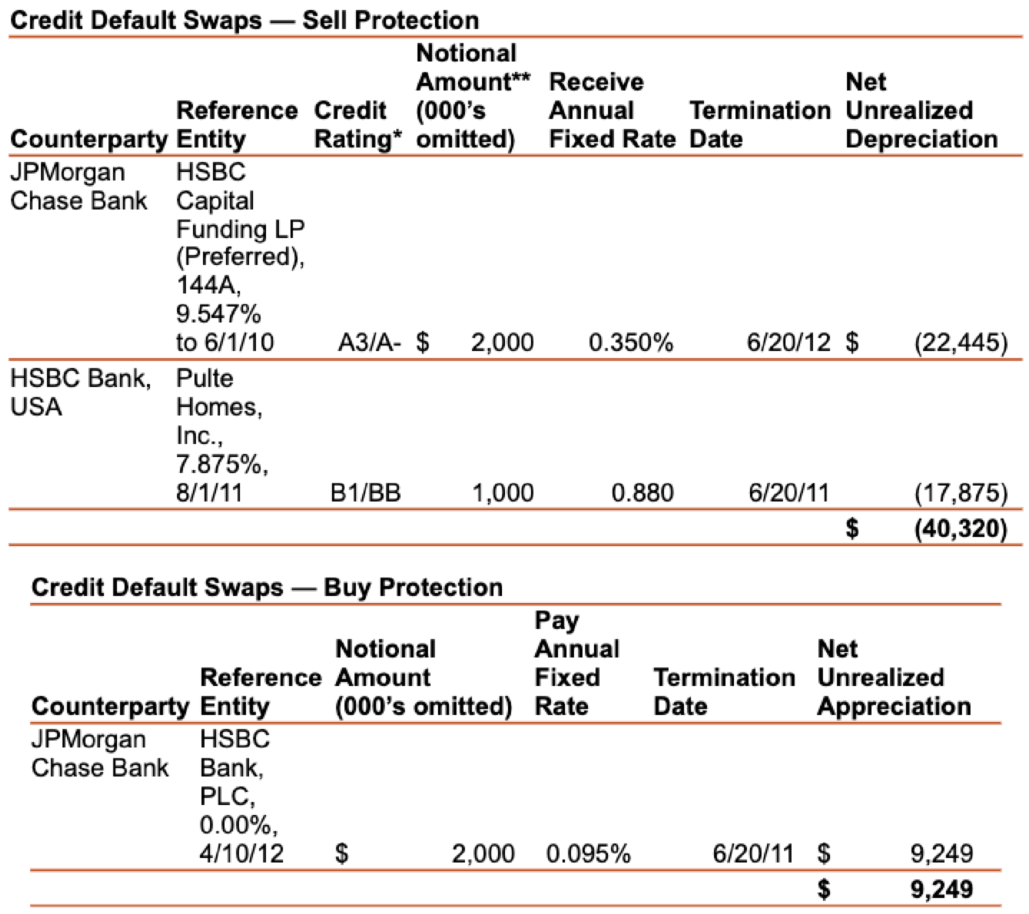
1 Introduction

The lack of a structured database of financial reports makes it difficult for Credit Default Swap related research studies to conduct a much more comprehensive and quantitative analysis. Therefore, a structured and well maintained database can help future researchers to analyze CDS and retrieve new and exciting information from it.

1.1 Background and Motivation

This project investigates how mutual funds leverage credit derivatives by studying their routine filings to the U.S. Securities and Exchange Commission. Credit Derivatives have a wide range of products and we will be studying a class of credit derivatives called Credit Default Swaps(CDS). Credit Default Swaps have a reference entity linked to them which are generally governments or corporations. The buyer has a credit asset with the reference entity and buys a CDS from the seller to insure himself against a default in the payment by the reference entity. It is thus used as a hedging tool to reduce the risk associated with a credit asset [4]. The buyer makes periodic payments to the seller till the date of the maturity of the contract and this constitutes the spread of the CDS. In the event of a credit default, the seller has to pay the buyer of the CDS the face value of the credit asset and all the interest payments that the buyer would have earned between that time till the date of the maturity of the asset.

Credit default swaps are traded over the counter and hence there isnt much information available on it. This resulted in it being extremely difficult to get relevant information from these reports to carry out further analysis. The reports also have varying formats of reporting which makes it extremely difficult to extract the data in an organized way .



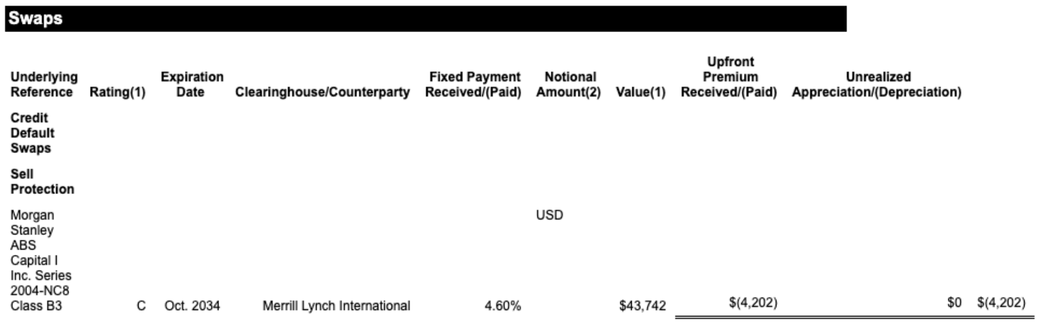
Credit Default Swaps — Sell Protection

Counterparty	Reference Entity	Credit Rating*	Notional Amount** (000's omitted)	Receive Annual Fixed Rate	Termination Date	Net Unrealized Depreciation
JPMorgan Chase Bank	HSBC Capital Funding LP (Preferred), 144A, 9.547% to 6/1/10	A3/A-	$ 2,000	0.350%	6/20/12	$ (22,445)
HSBC Bank, USA	Pulte Homes, Inc., 7.875%, 8/1/11	B1/BB	1,000	0.880	6/20/11	(17,875)
						$ (40,320)

Credit Default Swaps — Buy Protection

Counterparty	Reference Entity	Notional Amount (000's omitted)	Pay Annual Fixed Rate	Termination Date	Net Unrealized Appreciation
JPMorgan Chase Bank	HSBC Bank, PLC, 0.00%, 4/10/12	$ 2,000	0.095%	6/20/11	$ 9,249
					$ 9,249

Figure 1: Reporting Style of CDS in N-CSRS Report



Swaps

Underlying Reference	Rating(1)	Expiration Date	Clearinghouse/Counterparty	Fixed Payment Received/(Paid)	Notional Amount(2)	Value(1)	Upfront Premium Received/(Paid)	Unrealized Appreciation/(Depreciation)	
Credit Default Swaps									
Sell Protection									
Morgan Stanley ABS Capital I Inc. Series 2004-NC8 Class B3	C	Oct. 2034	Merrill Lynch International	4.60%	USD	$43,742	$(4,202)	$0	$(4,202)

Figure 2: Reporting Style of CDS in N-CSR Report

The two images above represent just two different ways in which CDS information is reported. We can observe the difficulty in extracting such information and formatting it and thus, organized information regarding CDS is extremely valuable. There have been a few previous studies exploring the usage of CDS by Mutual Funds[1],[5] but these reports examined only a small number of the institutions over a short period of time. Hence, we choose to comprehensively examine all the reports from 2004–2017. The ability to search for a companys name and get all its CDS dealings is something that has never been done before and it would be a powerful tool for many investors and would provide transparency, can be used to set appropriate capital requirements. This makes the results of our project extremely valuable for further research.

1.2 Objective

The objective of this project is to aggregate a structured database of credit default swap reportings from 2004–2017. This paper focuses on extraction and data processing of structured data and data processing and tagging of training data containing unstructured sentences for future NLP. This includes the methodology used to extract both structured and unstructured data from the reports, completely format, standardize and merge the structured data into one unified csv file. It also includes information about the Text Annotation Tool built in order to help us tag the unstructured sentences.

1.3 Scope

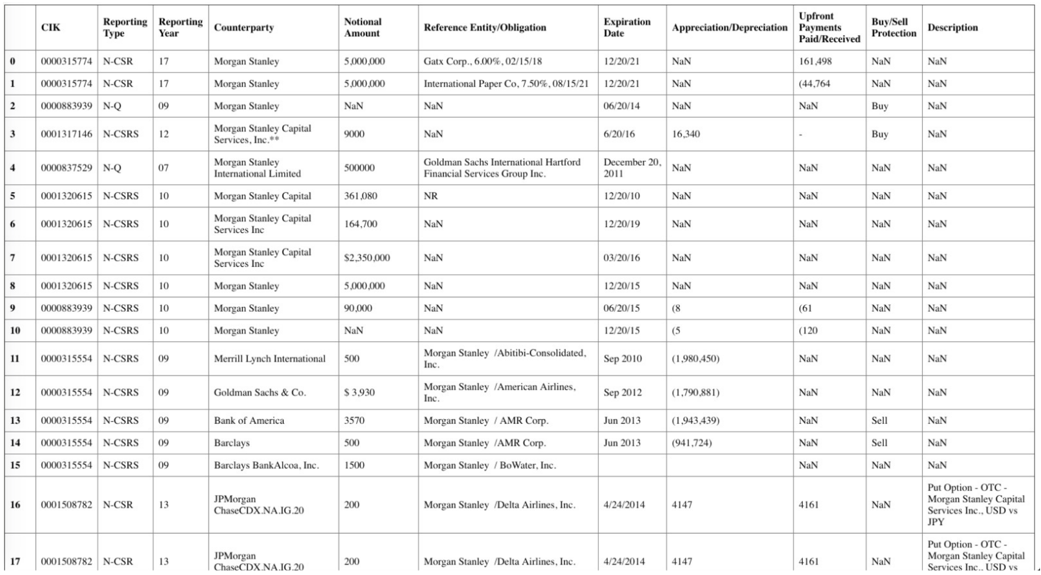
The project is divided into two parts Data Processing and Natural Language Processing. The project focuses on extracting and formatting CDS data in order to make a consolidated database of all the information.This data consists of both structured and unstructured data. We use rule based extraction methods for the structured data and since, this doesn't work on all the data, we need Natural Language Processing techniques to help us extract data from the unstructured data.

1.4 Deliverables

The complete implementation of the project is available on here. The project has a few major deliverables which have been outlined below:

1. **Text Annotation Tool**- A Django based web application developed to produce custom datasets for sequence labelling projects.

2. **Credit Default Swap Reportings Dataset**- A 16,813 rows dataset containing CDS reportings from 2004–2017 with categorical variables like Reference Entity, Notional Amount, Expiration Date, Counterparty etc.



	CIK	Reporting Type	Reporting Year	Counterparty	Notional Amount	Reference Entity/Obligation	Expiration Date	Appreciation/Depreciation	Upfront Payments Paid/Received	Buy/Sell Protection	Description
0	0000315774	N-CSR	17	Morgan Stanley	5,000,000	Gatx Corp., 6.00%, 02/15/18	12/20/21	NaN	161,498	NaN	NaN
1	0000315774	N-CSR	17	Morgan Stanley	5,000,000	International Paper Co, 7.50%, 08/15/21	12/20/21	NaN	(44,764	NaN	NaN
2	0000883939	N-Q	09	Morgan Stanley	NaN	NaN	06/20/14	NaN	NaN	Buy	NaN
3	0001317146	N-CSRS	12	Morgan Stanley Capital Services, Inc.**	9000	NaN	6/20/16	16,340	-	Buy	NaN
4	0000837529	N-Q	07	Morgan Stanley International Limited	500000	Goldman Sachs International Hartford Financial Services Group Inc.	December 20, 2011	NaN	NaN	NaN	NaN
5	0001320615	N-CSRS	10	Morgan Stanley Capital	361,080	NR	12/20/10	NaN	NaN	NaN	NaN
6	0001320615	N-CSRS	10	Morgan Stanley Capital Services Inc	164,700	NaN	12/20/19	NaN	NaN	NaN	NaN
7	0001320615	N-CSRS	10	Morgan Stanley Capital Services Inc	$2,350,000	NaN	03/20/16	NaN	NaN	NaN	NaN
8	0001320615	N-CSRS	10	Morgan Stanley	5,000,000	NaN	12/20/15	NaN	NaN	NaN	NaN
9	0000883939	N-CSRS	10	Morgan Stanley	90,000	NaN	06/20/15	(8	(61	NaN	NaN
10	0000883939	N-CSRS	10	Morgan Stanley	NaN	NaN	12/20/15	(5	(120	NaN	NaN
11	0000315554	N-CSRS	09	Merrill Lynch International	500	Morgan Stanley /Abitibi-Consolidated, Inc.	Sep 2010	(1,980,450)	NaN	NaN	NaN
12	0000315554	N-CSRS	09	Goldman Sachs & Co.	$ 3,930	Morgan Stanley /American Airlines, Inc.	Sep 2012	(1,790,881)	NaN	NaN	NaN
13	0000315554	N-CSRS	09	Bank of America	3570	Morgan Stanley / AMR Corp.	Jun 2013	(1,943,439)	NaN	Sell	NaN
14	0000315554	N-CSRS	09	Barclays	500	Morgan Stanley /AMR Corp.	Jun 2013	(941,724)	NaN	Sell	NaN
15	0000315554	N-CSRS	09	Barclays BankAlcoa, Inc.	1500	Morgan Stanley / BoWater, Inc.			NaN	NaN	NaN
16	0001508782	N-CSR	13	JPMorgan ChaseCDX.NA.IG.20	200	Morgan Stanley /Delta Airlines, Inc.	4/24/2014	4147	4161	NaN	Put Option - OTC - Morgan Stanley Capital Services Inc., USD vs JPY
17	0001508782	N-CSR	13	JPMorgan ChaseCDX.NA.IG.20	200	Morgan Stanley /Delta Airlines, Inc.	4/24/2014	4147	4161	NaN	Put Option - OTC - Morgan Stanley Capital Services Inc., USD vs

Figure 3: Combined CDS DataSet

3. **CRF Tagger for CDS Reportings Dataset**- A web application to allow researchers and other users to search for credit default swap

reportings by Counterparty, Reference Entity or Expiration Date.

4. **Credit Default Swap Search Engine**- A web application to allow researchers and other users to search for credit default swap reportings by Counterparty, Reference Entity or Expiration Date.

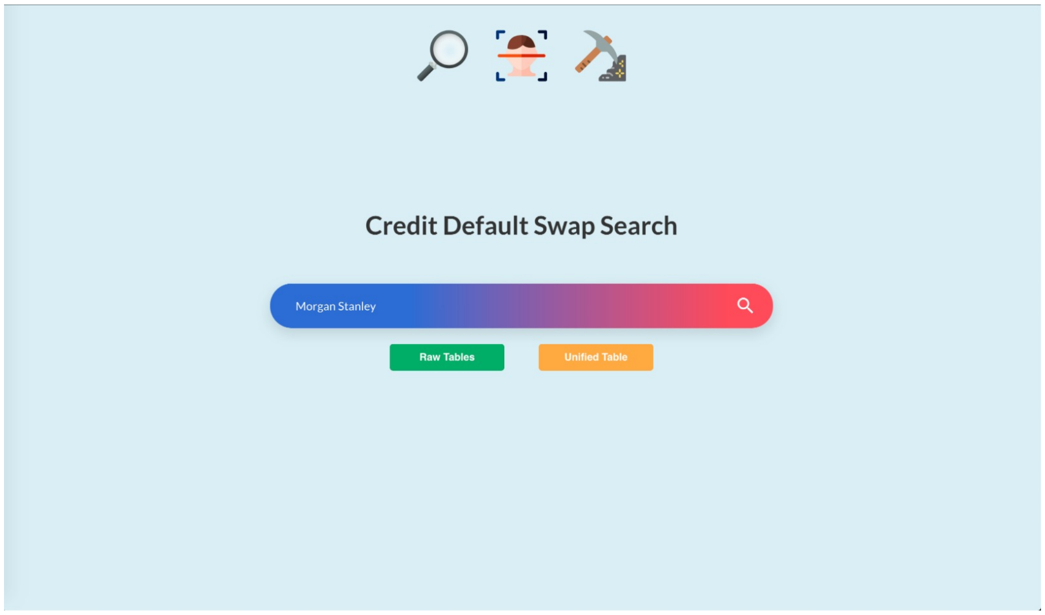


Figure 4: CDS Search Engine

5. **Report processing**- A web application built using Flask for users to upload reports containing structured CDS information and display the Credit Default Swap that it contains.

1.5 Outline of the report

The documentation for this project has been divided into two reports. Both the reports share the same background and motivation, however, the methodology and results for each of them have been specifically written to dive deeper into the implementation and difficulties encountered for each of the two aspects. So it is imperative that the reader must go through both the reports in order to completely understand each aspect of the project.

This report goes through **Data Processing**, which includes first extracting the structured data from the reports. The next step is the cleaning and formatting of the reports and then ensuring that all the data that has been extracted is accurate. The final step is normalizing the data and combining all the tables together into one unified csv file to achieve the Credit Default Swap reportings dataset. It also goes through how we get the unstructured sentences from the reports and the Text Annotation Tool built to help us tag the unstructured sentences to prepare them for the NLP.

The report that goes over the **NLP techniques** that we have implemented to conduct information extraction has been written by Tarun Sudhams. It provides an in-depth description of the implementation of the Conditional Random Fields on the CDS reporting dataset that we produced, analyses the results and compares them with CRF models used in Finance-specific datasets to analyse our performance.

2 Literature Review

2.1 Use of CDS by Corporate Bond Funds

CDS can be utilised not only as insurance but also as a profit generating tool by various investment funds. This research paper focuses on the effects that different factors such as the health of the Financial Markets, buyer sentiment have on the profit making ability of the investments made by Corporate Bond Funds. CDS can be used as a profit making tool when the seller of the CDS knows that the bond for which they are selling the CDS is unlikely to default. This will result in them getting the annual payments for offering the CDS protection while never having to actually pay out if the corporation which issued the bond doesnt default. It observes this during the Financial Crisis period(2007-2009). The result was that during the financial crisis, funds which were managed by teams performed poorly in comparison to single person managed funds [5]. The profit making ability of the Funds also decreased during the Financial Crisis as most of the corporations defaulted on their bond payments which led to these Corporate Bond Funds losing money.

2.2 Mutual Fund Holdings of CDS

This research paper sought to understand the motives behind the investments made by Mutual Funds into Credit Default Swaps [1]. They sought to analyse the relationship between liquidity needs of the fund and the buying/selling activities of CDS. It was found that funds start selling CDS when their liquidity condition was uncertain, when the bond was illiquid when compared to the CDS security. The paper also concludes that bigger Funds strategies regarding CDS investments move in the direction of yield enhancement and the smaller funds follow these bigger funds in risk taking.

3 Methodology

The below figure gives a good overview of our project and each step has been explained in detail in the subsequent parts below.

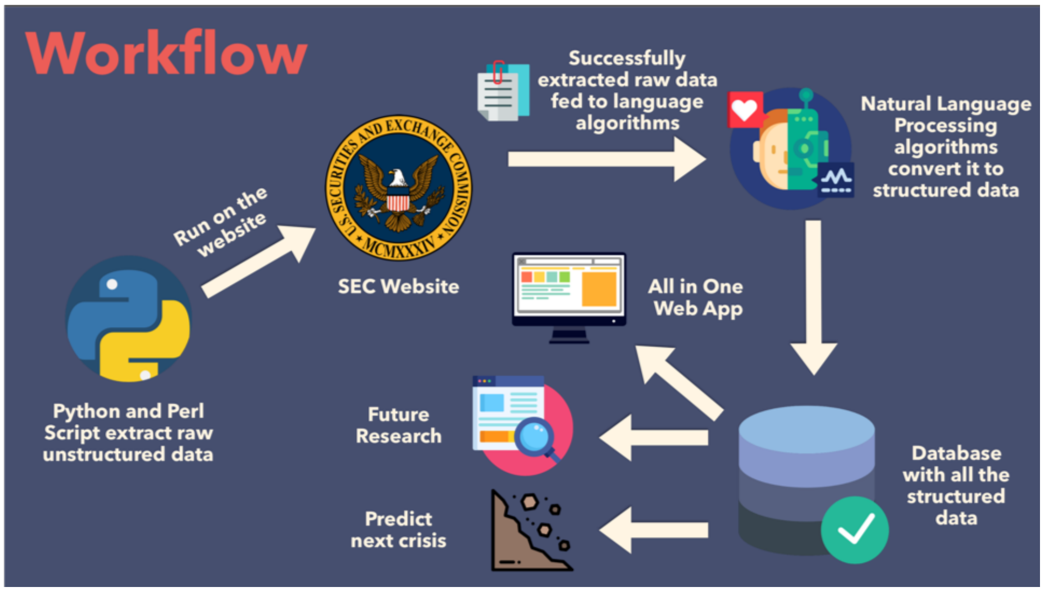

Workflow
Run on the website
SEC Website
Successfully extracted raw data fed to language algorithms
Natural Language Processing algorithms convert it to structured data
Python and Perl Script extract raw unstructured data
All in One Web App
Future Research
Predict next crisis
Database with all the structured data

Figure 5: Proposed Workflow

In a nutshell, the web crawling is done using Python scripts to collect gigabytes worth of financial reports from the United States Securities and Exchange Commission website will include our first step namely data collection process.We get both structured and unstructured data and we then clean and format the structured data after extracting it. Then we will move on to our natural language processing step which will help us convert the unstructured data collected in step 1 into a structured database.

3.1 Data Collection

This is the first step of our project which serves as the data collection process. Given the nature of our research subject, it is quite easy to locate the data as everything is available publicly. We employ key web scraping tools to consolidate the entire report filings from 2003 2017. There are few mature technologies which could be employed for this purpose such as NodeJS, C/C++, Python, PHP and Perl. However, there have been stability issues with NodeJS as it makes deploying multiple crawlers a complicated job. Development costs for C/C++ is very high and PHP has a very bad scheduler scheme which makes it harder to work with (Koshy, 2017). These reasons helped us identify Python and Perl as the best programming languages to write our scripts in.

3.2 Restructuring the Data

3.2.1 Building Corpus

In order to build a corpus for our NLP tasks, it was important for us to restructure the file structure such that it is easy to navigate between different kinds of reports. We had around 146 GB of data in one SEC folder which contained all the Funds' N-CSR, N-CSRS and N-Q reports along with their CIK number. The initial folder structure was that of the main SEC-Edgar-Data folder containing all the folders with the Mutual Funds names which in turn had a folder named with the Funds' CIK number which then had the folders N-CSR, N-CSRS, N-Q in them which respectively contained the Funds financial reports from 2004-2017. After going through the data, it was found that few of the Mutual Funds had changed the Fund names through the time period and this resulted in there being duplicate files under different Fund names but with the same CIK number. We first wrote a script to restructure the data such that we could split it into 3 parts from which we could extract the data sequentially.

Algorithm 1: Folder Restructure Pseudocode

```
Result: Restructured Folder
initialization;
for all the directories in SEC-Edgar-Data folder do
    for all the directories in the Fund Name folder do
        if the directory is N-CSR then
            Move the entire sub-directory to the N-CSR Folder created
              outside SEC-Edgar-Data;
        end
        if the directory is N-CSRS then
            Move the entire sub-directory to the N-CSRS Folder created
              outside SEC-Edgar-Data;
        end
        if the directory is N-Q then
            Move the entire sub-directory to the N-Q Folder created
              outside SEC-Edgar-Data;
        end
    end
end
```

The above algorithm goes over the pseudocode of the script which was run on the original folder and it segregated the original folder into three folders namely N-CSR, N-CSRS and N-Q. Each of these folders contained a folder with the CIK number and inside that folder was the folder with the Mutual Funds name containing the respective reports. In the event that a Mutual Fund had changed its name during the time period of 2004-2017, then this restructuring made it easier for us to identify the Fund by its unique CIK number. After restructuring , the folder structure was as follows(similar structuring for N-CSRS and N-Q):

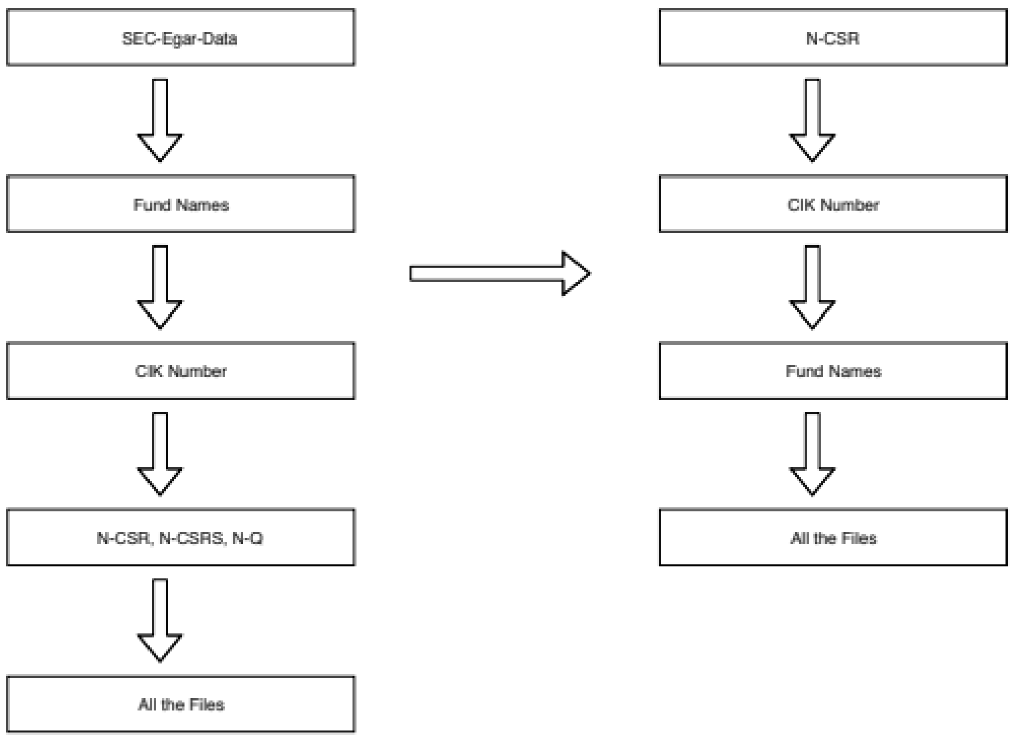


Figure 6: Updated File Structure

We then unzipped all the files to their original text files. After going through the data, we found out that along with files in there which had CDS information, there was also files which didnt have any mentions of Credit Default Swap. We hence decided to run another script to remove these files and cut down the size of our dataset to only the useful reports to speed up the processing time. This script checked all the text files for mentions of the word Credit Default Swap and other relevant words and when it found it, moved the respective text file to a new folder called Corpus_N-CSR(for N-CSR folder) where it also appended the CIK number along with N-CSR to the file name to make it easier for us to identify the files during future processes. This was similarly done for the N-CSRS and N-Q folders. The folder structure was as follows:

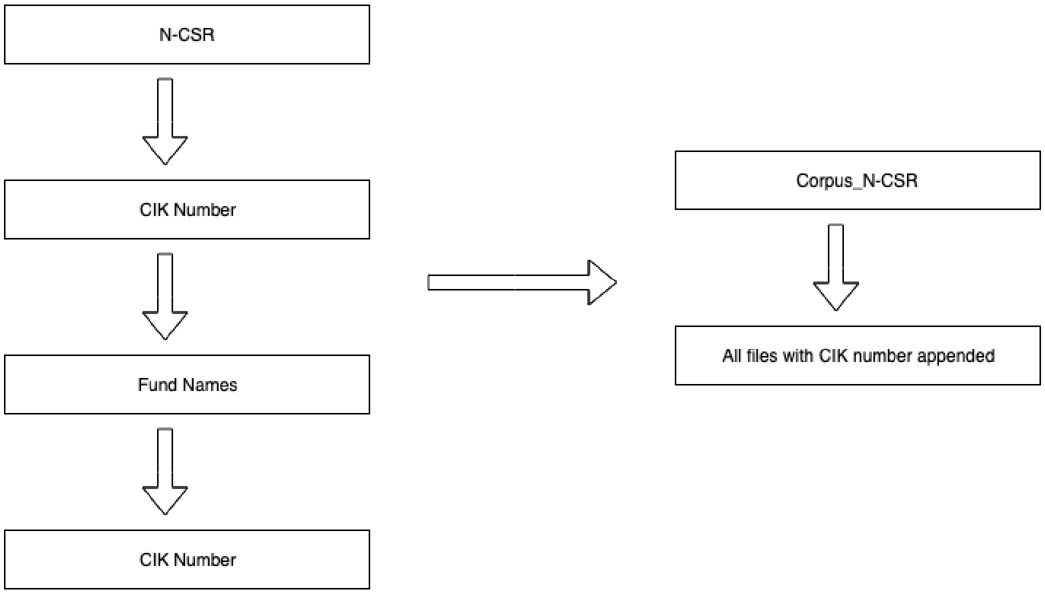
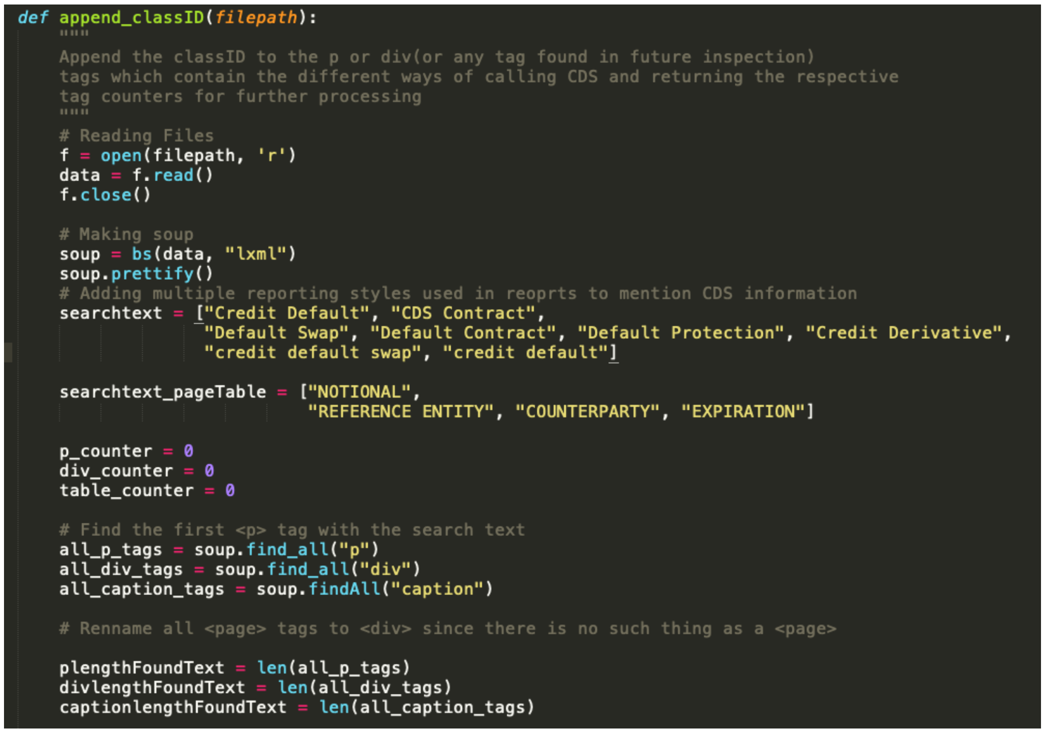


Figure 7: Corpus Folder Structure

3.3 Table Extraction

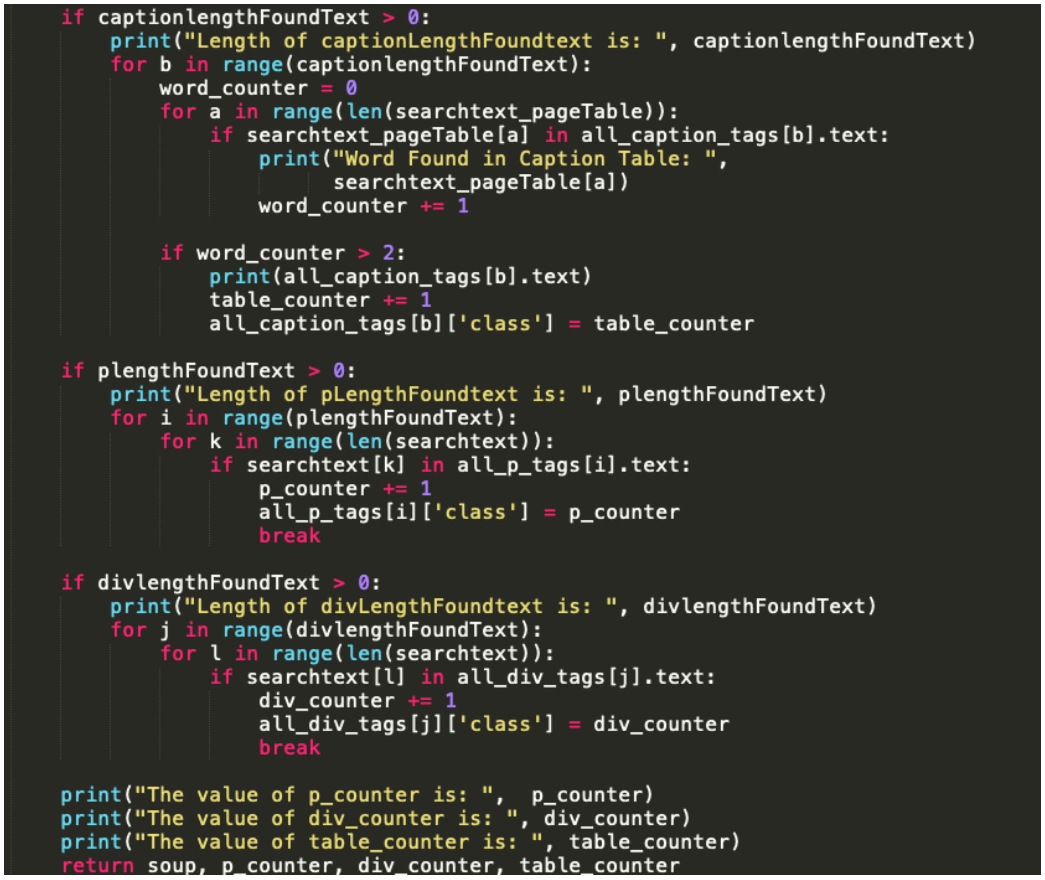
Once we got the folders containing the N-CSR, N-CSRS, N-Q reports, we found out that most of the reports had the CDS information in the form of tables. Hence it was necessary for us to figure out a way to extract these tables out of the reports. The reports had these tables enclosed within HTML tags such as the <div>and <table>tags. We decided to use beautiful soup with python and extract these tables out from the reports using the HTML tags. We wrote a python script to help us do this.



```
def append_classID(filepath):
    """
    Append the classID to the p or div(or any tag found in future inspection)
    tags which contain the different ways of calling CDS and returning the respective
    tag counters for further processing
    """
    # Reading Files
    f = open(filepath, 'r')
    data = f.read()
    f.close()

    # Making soup
    soup = bs(data, "lxml")
    soup.prettify()
    # Adding multiple reporting styles used in reoprts to mention CDS information
    searchtext = ["Credit Default", "CDS Contract",
                  "Default Swap", "Default Contract", "Default Protection", "Credit Derivative",
                  "credit default swap", "credit default"]

    searchtext_pageTable = ["NOTIONAL",
                            "REFERENCE ENTITY", "COUNTERPARTY", "EXPIRATION"]

    p_counter = 0
    div_counter = 0
    table_counter = 0

    # Find the first <p> tag with the search text
    all_p_tags = soup.find_all("p")
    all_div_tags = soup.find_all("div")
    all_caption_tags = soup.findAll("caption")

    # Rennname all <page> tags to <div> since there is no such thing as a <page>

    plengthFoundText = len(all_p_tags)
    divlengthFoundText = len(all_div_tags)
    captionlengthFoundText = len(all_caption_tags)
```

Figure 8: Table Extractor -1

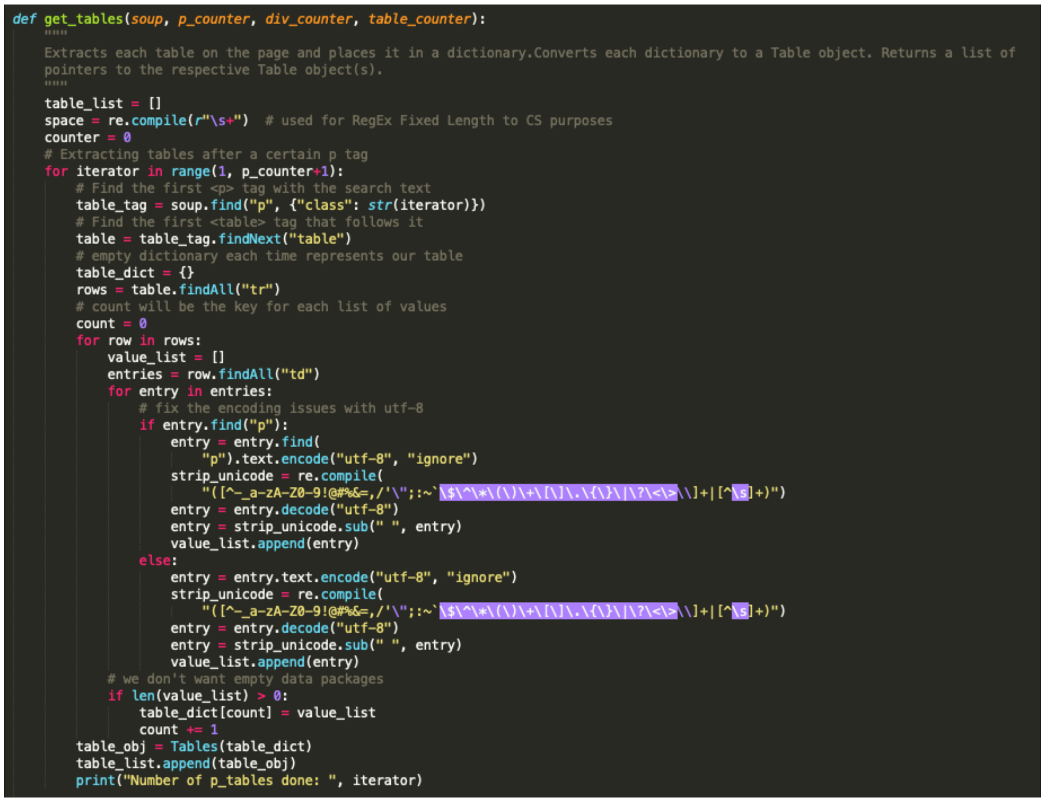
The above figure shows the first part of the first method that is called in the script. Here, we take the Financial Report with the HTML tags in it and then use beautiful soup to first make soup out of all the HTML tags. We then specify the words that the tables should contain in order for us to extract them. Furthermore, we find all the p, div, caption and store it in an array which we will later iterate through to find the respective tables within these tags. We have counters for the different types of tags as a way to count the number of tables being extracted from each type of tag.



```
    if captionlengthFoundText > 0:
        print("Length of captionLengthFoundtext is: ", captionlengthFoundText)
        for b in range(captionlengthFoundText):
            word_counter = 0
            for a in range(len(searchtext_pageTable)):
                if searchtext_pageTable[a] in all_caption_tags[b].text:
                    print("Word Found in Caption Table: ",
                          searchtext_pageTable[a])
                    word_counter += 1

            if word_counter > 2:
                print(all_caption_tags[b].text)
                table_counter += 1
                all_caption_tags[b]['class'] = table_counter

    if plengthFoundText > 0:
        print("Length of pLengthFoundtext is: ", plengthFoundText)
        for i in range(plengthFoundText):
            for k in range(len(searchtext)):
                if searchtext[k] in all_p_tags[i].text:
                    p_counter += 1
                    all_p_tags[i]['class'] = p_counter
                    break

    if divlengthFoundText > 0:
        print("Length of divLengthFoundtext is: ", divlengthFoundText)
        for j in range(divlengthFoundText):
            for l in range(len(searchtext)):
                if searchtext[l] in all_div_tags[j].text:
                    div_counter += 1
                    all_div_tags[j]['class'] = div_counter
                    break

    print("The value of p_counter is: ",  p_counter)
    print("The value of div_counter is: ", div_counter)
    print("The value of table_counter is: ", table_counter)
    return soup, p_counter, div_counter, table_counter
```

Figure 9: Table Extractor -2

In the above picture, is the second half of the first method that is called where we first check if there exists each of p, div, caption tags. We iterate through the array containing the text inside each of these tags and also iterate through the keywords that we want our tables to contain and compare them. If we find that, for example, in a certain p tag, we find a mention of one of our keywords, we then increase the value of the p_counter and also append the value of the p_counter as classID. We similarly do this for the other two tags. This method then returns all these values which will be used as the input arguments for the next method.



```python
def get_tables(soup, p_counter, div_counter, table_counter):
    """
    Extracts each table on the page and places it in a dictionary.Converts each dictionary to a Table object. Returns a list of
    pointers to the respective Table object(s).
    """
    table_list = []
    space = re.compile(r"\s+")  # used for RegEx Fixed Length to CS purposes
    counter = 0
    # Extracting tables after a certain p tag
    for iterator in range(1, p_counter+1):
        # Find the first <p> tag with the search text
        table_tag = soup.find("p", {"class": str(iterator)})
        # Find the first <table> tag that follows it
        table = table_tag.findNext("table")
        # empty dictionary each time represents our table
        table_dict = {}
        rows = table.findAll("tr")
        # count will be the key for each list of values
        count = 0
        for row in rows:
            value_list = []
            entries = row.findAll("td")
            for entry in entries:
                # fix the encoding issues with utf-8
                if entry.find("p"):
                    entry = entry.find(
                        "p").text.encode("utf-8", "ignore")
                    strip_unicode = re.compile(
                        "([^-_a-zA-Z0-9!@#%&=,/'\";:~`\$\^\*\(\)\+\[\]\.\{\}\|\?\<\>\\]+|[^\s]+)")
                    entry = entry.decode("utf-8")
                    entry = strip_unicode.sub(" ", entry)
                    value_list.append(entry)
                else:
                    entry = entry.text.encode("utf-8", "ignore")
                    strip_unicode = re.compile(
                        "([^-_a-zA-Z0-9!@#%&=,/'\";:~`\$\^\*\(\)\+\[\]\.\{\}\|\?\<\>\\]+|[^\s]+)")
                    entry = entry.decode("utf-8")
                    entry = strip_unicode.sub(" ", entry)
                    value_list.append(entry)
            # we don't want empty data packages
            if len(value_list) > 0:
                table_dict[count] = value_list
                count += 1
        table_obj = Tables(table_dict)
        table_list.append(table_obj)
        print("Number of p_tables done: ", iterator)
```

Figure 10: Extracting Tables based on Search Text

The above method is called next wherein we iterate through the list of p tags which contain the words and find the table tag and then go through the rows of the table and extract all the contents. We then return the list of tables at the end. Once we get the list of tables, we save the tables under the given file name in csv format. It is similarly done for the other tags.

3.4 Formatting of Data

3.4.1 checkTable.sh

We then wrote a bash script to run this on all the reports in the three folders. The folders from which we could extract tables were put into a separate structured folder and the rest of the files were put in an unstructured folder. This resulted in there being 3 structured folders each containing the tables from each report.

3.4.2 Format.py

Once we had the all the useful tables in different folders, we started working on formatting the tables. We found after going through the data that most of the tables had a different format for the data. This meant that we had to go through the csv files manually, try and identify a pattern which was followed by the majority of the tables and then try and write a python script to format those kinds of scripts.

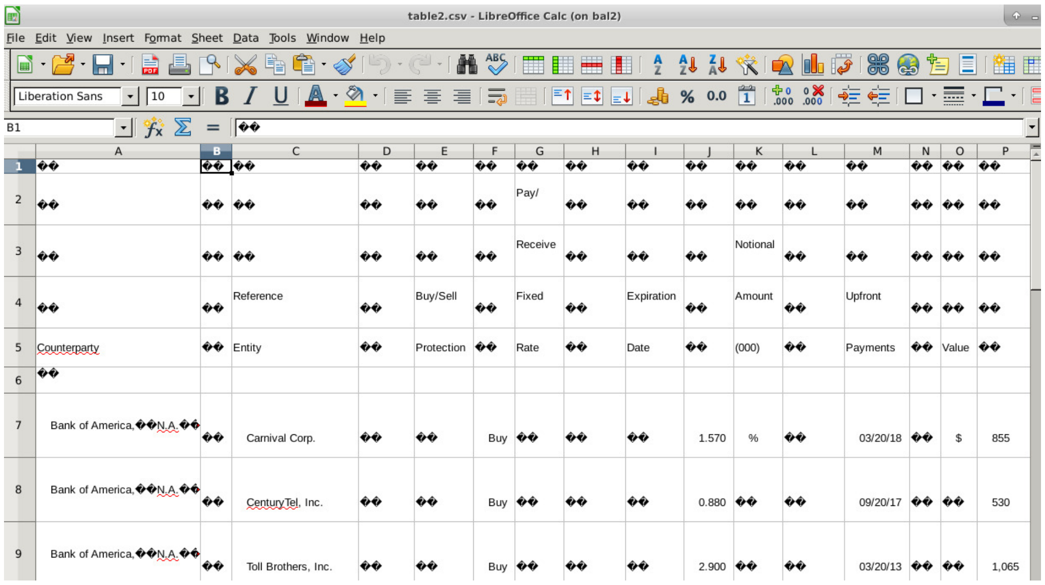


table2.csv - LibreOffice Calc (on bal2)
File Edit View Insert Format Sheet Data Tools Window Help
Liberation Sans
10
B1
��

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	��	��	��	��	��	��	��	��	��	��	��	��	��	��	��	��
2	��	��	��	��	��	��	Pay/	��	��	��	��	��	��	��	��	��
3	��	��	��	��	��	��	Receive	��	��	��	Notional	��	��	��	��	��
4	��	��	Reference	��	Buy/Sell	��	Fixed	��	Expiration	��	Amount	��	Upfront	��	��	��
5	Counterparty	��	Entity	��	Protection	��	Rate	��	Date	��	(000)	��	Payments	��	Value	��
6	��															
7	Bank of America,��N.A.��	��	Carnival Corp.	��	��	Buy	��	��	��	1.570	%	��	03/20/18	��	$	855
8	Bank of America,��N.A.��	��	CenturyTel, Inc.	��	��	Buy	��	��	��	0.880	��	��	09/20/17	��	��	530
9	Bank of America,��N.A.��	��	Toll Brothers, Inc.	��	��	Buy	��	��	��	2.900	��	��	03/20/13	��	��	1,065

Figure 11: Example 1 of an unformatted Table

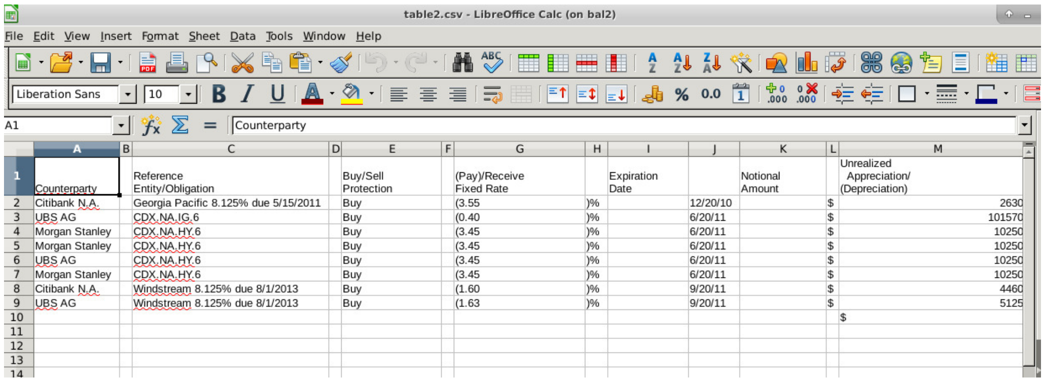


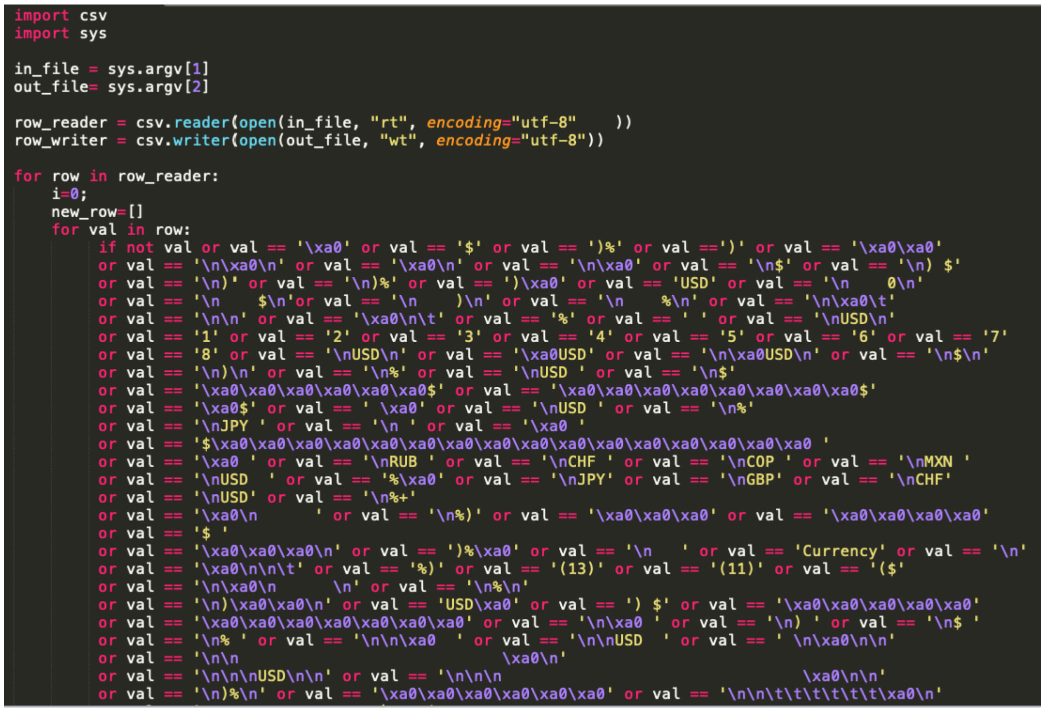
table2.csv - LibreOffice Calc (on bal2)

A1 Counterparty

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Counterparty		Reference Entity/Obligation		Buy/Sell Protection		(Pay)/Receive Fixed Rate		Expiration Date		Notional Amount		Unrealized Appreciation/ (Depreciation)
2	Citibank N.A.		Georgia Pacific 8.125% due 5/15/2011		Buy		(3.55	)%		12/20/10		$	2630
3	UBS AG		CDX.NA.IG.6		Buy		(0.40	)%		6/20/11		$	101570
4	Morgan Stanley		CDX.NA.HY.6		Buy		(3.45	)%		6/20/11		$	10250
5	Morgan Stanley		CDX.NA.HY.6		Buy		(3.45	)%		6/20/11		$	10250
6	UBS AG		CDX.NA.HY.6		Buy		(3.45	)%		6/20/11		$	10250
7	Morgan Stanley		CDX.NA.HY.6		Buy		(3.45	)%		6/20/11		$	10250
8	Citibank N.A.		Windstream 8.125% due 8/1/2013		Buy		(1.60	)%		9/20/11		$	4460
9	UBS AG		Windstream 8.125% due 8/1/2013		Buy		(1.63	)%		9/20/11		$	5125
10													$
11													
12													
13													

Figure 12: Example 2 of an unformatted Table

Above two figures show two examples of the types of format that the tables initially have. The other files may have a completely different format. We identified in majority of the files, there were a lot of random characters and spaces present in the cells in the csv files between the cells which actually had the information. We wrote a python script to eliminate all of these random characters and shift the cells with the actual information.



```python
import csv
import sys

in_file = sys.argv[1]
out_file= sys.argv[2]

row_reader = csv.reader(open(in_file, "rt", encoding="utf-8"    ))
row_writer = csv.writer(open(out_file, "wt", encoding="utf-8"))

for row in row_reader:
    i=0;
    new_row=[]
    for val in row:
         if not val or val == '\xa0' or val == '$' or val == ')%' or val ==')' or val == '\xa0\xa0'
         or val == '\n\xa0\n' or val == '\xa0\n' or val == '\n\xa0' or val == '\n$' or val == '\n) $'
         or val == '\n)' or val == '\n)%' or val == ')\xa0' or val == 'USD' or val == '\n    0\n'
         or val == '\n    $\n'or val == '\n    )\n' or val == '\n    %\n' or val == '\n\xa0\t'
         or val == '\n\n' or val == '\xa0\n\t' or val == '%' or val == ' ' or val == '\nUSD\n'
         or val == '1' or val == '2' or val == '3' or val == '4' or val == '5' or val == '6' or val == '7'
         or val == '8' or val == '\nUSD\n' or val == '\xa0USD' or val == '\n\xa0USD\n' or val == '\n$\n'
         or val == '\n)\n' or val == '\n%' or val == '\nUSD ' or val == '\n$'
         or val == '\xa0\xa0\xa0\xa0\xa0\xa0$' or val == '\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0$'
         or val == '\xa0$' or val == ' \xa0' or val == '\nUSD ' or val == '\n%'
         or val == '\nJPY ' or val == '\n ' or val == '\xa0 '
         or val == '$\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0\xa0 '
         or val == '\xa0 ' or val == '\nRUB ' or val == '\nCHF ' or val == '\nCOP ' or val == '\nMXN '
         or val == '\nUSD  ' or val == '%\xa0' or val == '\nJPY' or val == '\nGBP' or val == '\nCHF'
         or val == '\nUSD' or val == '\n%+'
         or val == '\xa0\n      ' or val == '\n%)' or val == '\xa0\xa0\xa0' or val == '\xa0\xa0\xa0\xa0'
         or val == '$ '
         or val == '\xa0\xa0\xa0\n' or val == ')%\xa0' or val == '\n   ' or val == 'Currency' or val == '\n'
         or val == '\xa0\n\n\t' or val == '%)' or val == '(13)' or val == '(11)' or val == '($'
         or val == '\n\xa0\n      \n' or val == '\n%\n'
         or val == '\n)\xa0\xa0\n' or val == 'USD\xa0' or val == ') $' or val == '\xa0\xa0\xa0\xa0\xa0'
         or val == '\xa0\xa0\xa0\xa0\xa0\xa0\xa0' or val == '\n\xa0 ' or val == '\n) ' or val == '\n$ '
         or val == '\n% ' or val == '\n\n\xa0  ' or val == '\n\nUSD  ' or val == ' \n\xa0\n\n'
         or val == '\n\n                              \xa0\n'
         or val == '\n\n\nUSD\n\n' or val == '\n\n\n                                    \xa0\n\n'
         or val == '\n)%\n' or val == '\xa0\xa0\xa0\xa0\xa0\xa0' or val == '\n\n\t\t\t\t\t\t\xa0\n'
```

Figure 13: Showcasing a portion of the different types of unwanted characters that we remove

Above figure represents a portion of the python script used to format the tables. The script works by taking in the unformatted file as the first argument and another filename which will contain the formatted table as the second argument. We loop over each row in the csv file and initialize an array which will contain values for the formatted row. We then go through each cell in the row and check if the character matches any of the random unwanted characters or spaces and if it does, we just ignore it. We initialize a counter i to 0 in the beginning of the loop and this gives us the index of the value and just increment it by 1, after checking if doing this won't exceed the length of the row, to ignore the unwanted characters. If the cell has useful information, we append it to the new empty array which we initialized and then increment the value of i. The unwanted characters which we are checking have all been hardcoded after going manually through all the reports as it would have otherwise been impossible to identify all the different types of these characters.

After it goes through all the rows, we have another check in place to see if the length of the row after removing all the unwanted character exceeds 3 and only if this condition is true, we write it to the new file. This check is in place to help us eliminate those rows and csv files which dont actually have any useful CDS values that we require. An example of this is shown in the image below.

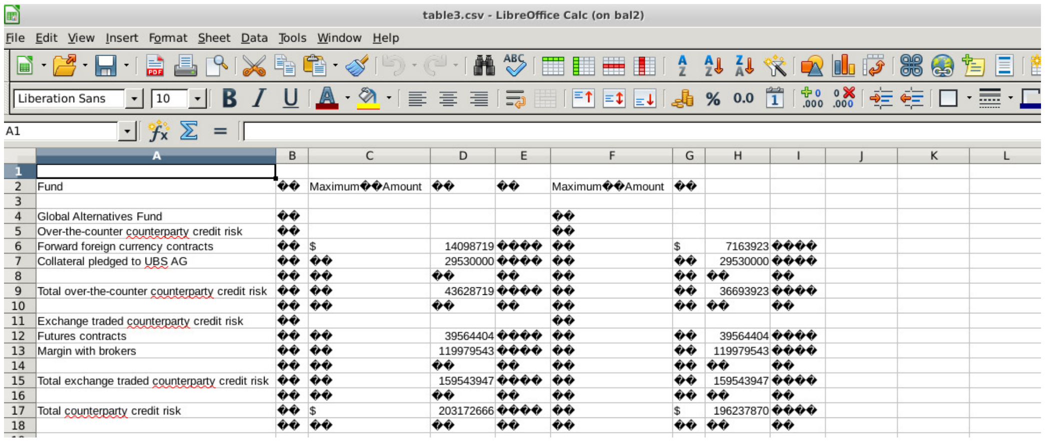


table3.csv - LibreOffice Calc (on bal2)
File Edit View Insert Format Sheet Data Tools Window Help
Liberation Sans
10
A1

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2	Fund	��	Maximum��Amount	��	��	Maximum��Amount	��					
3												
4	Global Alternatives Fund	��				��						
5	Over-the-counter counterparty credit risk	��				��						
6	Forward foreign currency contracts	��	$	14098719	����	��	$	7163923	����			
7	Collateral pledged to UBS AG	��	��	29530000	����	��	��	29530000	����			
8		��	��	��	��	��	��	��	��			
9	Total over-the-counter counterparty credit risk	��	��	43628719	����	��	��	36693923	����			
10		��	��	��	��	��	��	��	��			
11	Exchange traded counterparty credit risk	��				��						
12	Futures contracts	��	��	39564404	����	��	��	39564404	����			
13	Margin with brokers	��	��	119979543	����	��	��	119979543	����			
14		��	��	��	��	��	��	��	��			
15	Total exchange traded counterparty credit risk	��	��	159543947	����	��	��	159543947	����			
16		��	��	��	��	��	��	��	��			
17	Total counterparty credit risk	��	$	203172666	����	��	$	196237870	����			
18		��	��	��	��	��	��	��	��			

Figure 14: Example of Table containing useless CDS information

We see that this table came to be part of the current data set as it has the word counterparty in it and hence, it passed the first shell script. It however, doesnt contain any useful information that we require, and as it would have only 3 columns after formatting, with the final if condition, none of the rows would get written to the output file and hence it would only give a blank csv file as the output.

3.4.3 Blank.sh

We run this script on the formatted csv files after running the format.py script on all the folders. This script is to delete all the blank csv files that are present after formatting. It first runs a python script on all the files and it outputs the word empty if the csv files are blank. The script then uses the command grep to check the output file of the previous python script and deletes the csv file if it finds the word empty.

3.4.4 Formatword.py

This python script was written to remove those rows in the csv files which contained the word Total. This was done as after going through the data, it was found that most of the CDS tables had at the end of the table a row containing the Total Credit Default Swaps and Total Unrealized Appreciation(Depreciation) etc with corresponding values. This row was not needed, and it was necessary to remove this row in all the tables in order to make a neatly formatted unified csv file at the end. There were also tables which didn‘t have Total or a description written in the last row but just had the values of the total credit default swaps. These rows were however already removed using the last if condition in the format.py script and hence, we dont have to do anything additional to them.

3.4.5 Libor.sh

After running all the scripts above, we get a formatted table, but we also realized that along with Credit Default Swap tables, there were also a few tables containing Interest Rate Swaps and some other tables containing the overall figures for CDS being extracted and formatted. This was because the reporting style for those tables were similar and they had the same column headings as a CDS table. It was hence necessary to remove these csv files containing this other information.

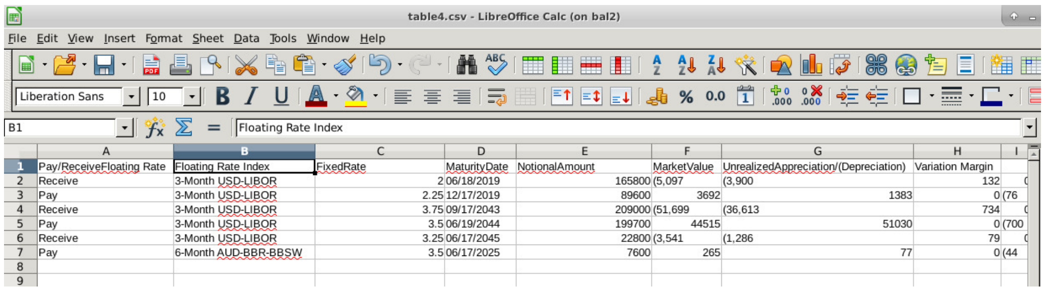


table4.csv - LibreOffice Calc (on bal2)
File Edit View Insert Format Sheet Data Tools Window Help
Liberation Sans
10
B1
Floating Rate Index

	A	B	C	D	E	F	G	H	I
1	Pay/ReceiveFloating Rate	Floating Rate Index	FixedRate	MaturityDate	NotionalAmount	MarketValue	UnrealizedAppreciation/(Depreciation)	Variation Margin	
2	Receive	3-Month USD-LIBOR	2	06/18/2019	165800	(5,097	(3,900	132	0
3	Pay	3-Month USD-LIBOR	2.25	12/17/2019	89600	3692	1383	0	(76
4	Receive	3-Month USD-LIBOR	3.75	09/17/2043	209000	(51,699	(36,613	734	0
5	Pay	3-Month USD-LIBOR	3.5	06/19/2044	199700	44515	51030	0	(700
6	Receive	3-Month USD-LIBOR	3.25	06/17/2045	22800	(3,541	(1,286	79	0
7	Pay	6-Month AUD-BBR-BBSW	3.5	06/17/2025	7600	265	77	0	(44
8									
9									

Figure 15: Example of an interest rate swap table extracted which we need to filter

This script searches for keywords such as LIBOR which is used to report Interest rate swaps in the Floating Rate Index column as seen in the above image, in the csv files and also other words such as Convertible Bonds and Forward which signifies Forward swaps. It then deletes the files which contains any of these words. This helps us to get a data set containing only Credit Default Swap tables and information.

3.4.6 Emptydir.sh

This is a simple script which is run on all the folders after all the above scripts have been executed. It uses the find command to check if any of the folders are empty and then removes those folders. This was done as the blank.sh and libor.sh script would be removing the useless csv files and there were many instances in which the folder after that would be empty as all the csv files in it were useless. Hence it was necessary to remove these empty folders.

3.4.7 Master.sh

This is a master script which runs all the above scripts together one after the other. It runs the following scripts in order

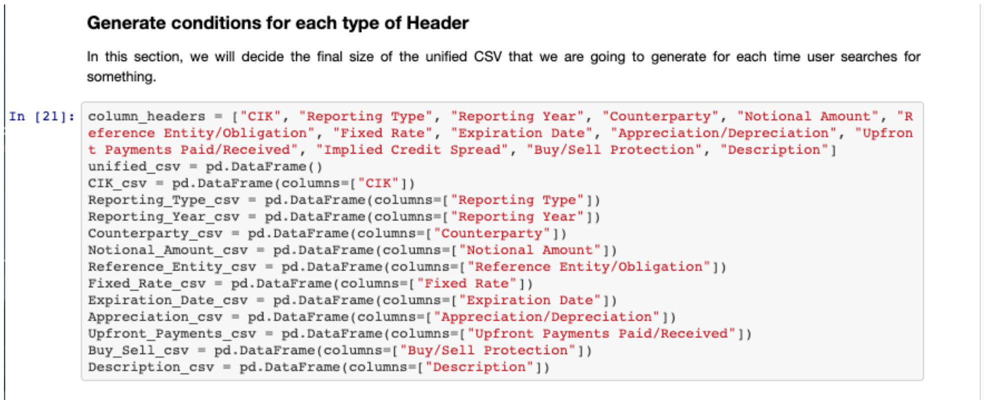
1. pythonExtractor.py script on all the files
2. checkTable.sh
3. format.sh (shell script to run the format.py script on all the csv files)
4. blank.sh
5. formatword.sh (shell script to run the formatword.py script on all the csv files)
6. libor.sh
7. Emptydir.sh

We can run this script on any file and it will extract and completely format the CDS tables and output them. The accuracy of this script was 76% (running this script on 100 random csv files containing the tables led to 76 files being completely formatted). The rest of the 24 files were around 90 percent formatted with there being some discrepancies in the way the column headers were reported. To ensure that the database consisting of all the tables was accurate and to help us build the unified csv file, we also manually went through each file and corrected the formatting of those files which werent fully formatted by the scripts.

3.5 Unified CSV File Containing all the CDS Data

The next step in the process is to take all of the data that we have and to merge all of it into one CSV file together. This would be helpful to present the data, for future research and also to help us do the time series analysis.

The first challenge in merging the data was that there were different ways in which the column headers were written. The header for Notional Amount could be "Notional Amount Due on Default" in one report, "Notional Value" in another or even "Notional Amount (000s)" in another report. Hence, we first gauge the different type of headers possible in all the reports and then come up with a master list containing the headers for the unified CSV file.



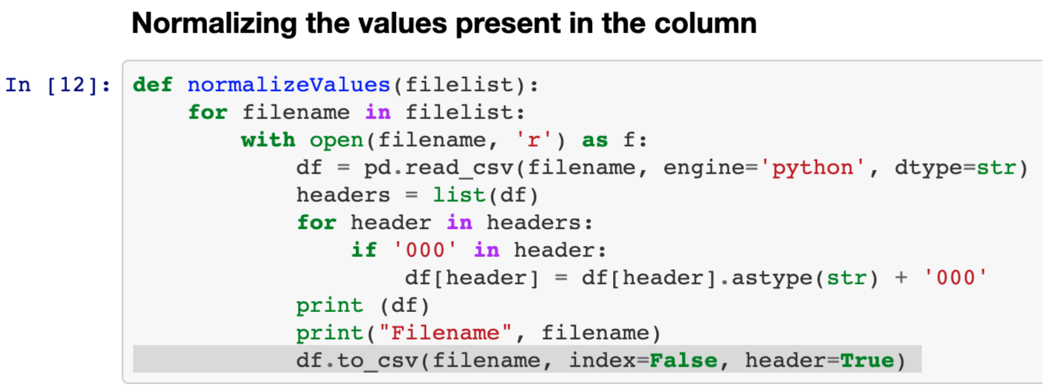
Generate conditions for each type of Header

In this section, we will decide the final size of the unified CSV that we are going to generate for each time user searches for something.

```
In [21]: column_headers = ["CIK", "Reporting Type", "Reporting Year", "Counterparty", "Notional Amount", "R
eference Entity/Obligation", "Fixed Rate", "Expiration Date", "Appreciation/Depreciation", "Upfron
t Payments Paid/Received", "Implied Credit Spread", "Buy/Sell Protection", "Description"]
unified_csv = pd.DataFrame()
CIK_csv = pd.DataFrame(columns=["CIK"])
Reporting_Type_csv = pd.DataFrame(columns=["Reporting Type"])
Reporting_Year_csv = pd.DataFrame(columns=["Reporting Year"])
Counterparty_csv = pd.DataFrame(columns=["Counterparty"])
Notional_Amount_csv = pd.DataFrame(columns=["Notional Amount"])
Reference_Entity_csv = pd.DataFrame(columns=["Reference Entity/Obligation"])
Fixed_Rate_csv = pd.DataFrame(columns=["Fixed Rate"])
Expiration_Date_csv = pd.DataFrame(columns=["Expiration Date"])
Appreciation_csv = pd.DataFrame(columns=["Appreciation/Depreciation"])
Upfront_Payments_csv = pd.DataFrame(columns=["Upfront Payments Paid/Received"])
Buy_Sell_csv = pd.DataFrame(columns=["Buy/Sell Protection"])
Description_csv = pd.DataFrame(columns=["Description"])
```

Figure 16: Code containing the master list for the unified csv

We also normalize the values in the Notional Amount column such that every time its reported in (000s), we add (000) to the value present in the column. The below image illustrates the code to achieve this.



Normalizing the values present in the column

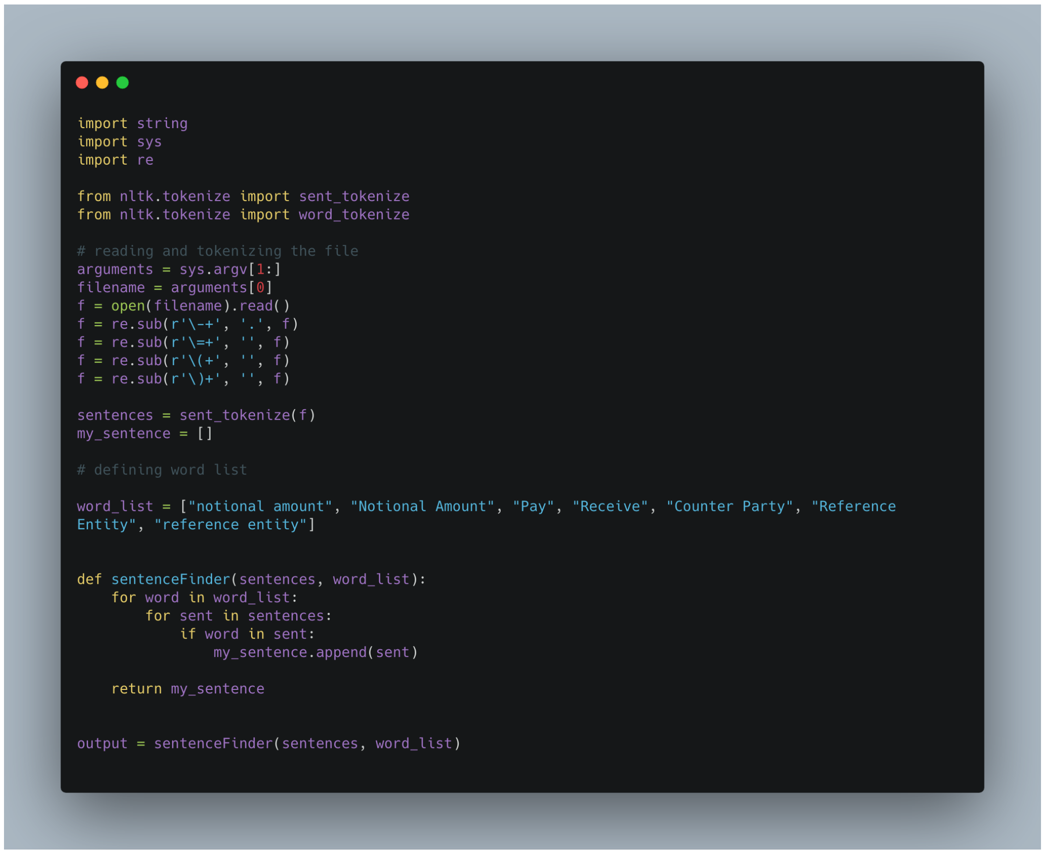
```
In [12]: def normalizeValues(filelist):
             for filename in filelist:
                 with open(filename, 'r') as f:
                     df = pd.read_csv(filename, engine='python', dtype=str)
                     headers = list(df)
                     for header in headers:
                         if '000' in header:
                             df[header] = df[header].astype(str) + '000'
                     print (df)
                     print("Filename", filename)
                     df.to_csv(filename, index=False, header=True)
```

Figure 17: Code for Normalization

We then merge all the csv files and perform basic data cleaning and add NaN for cells which don‘t contain values. We end up with a total of 16813 rows after merging all the files.

3.6 Tagging of Unstructured Data for Natural Language Processing

After extracting and formatting the structured data, we move on to preparing the unstructured data for Natural Language Processing. We first need to remove all the HTML tags from the reports in order for us to get only the text out to be used for the NLP. We wrote a basic python script which would strip out all the HTML tags and then ran it on all the unstructured folders using a bash script. Once we got only the text from the report, we realized that we would run into a class imbalance problem as we would have a lot of text which would not be tagged along with a few useful unstructured sentences containing information about CDS in the report. We hence decided to write a python script using regex and substring to extract only those sentences which contained relevant information as this would greatly help increase the accuracy while training the data after tagging. Below is an extract of the script wherein we have defined a set of words and only extracted the sentences which contain these words.



```python
import string
import sys
import re

from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize

# reading and tokenizing the file
arguments = sys.argv[1:]
filename = arguments[0]
f = open(filename).read()
f = re.sub(r'\-+', '.', f)
f = re.sub(r'\=+', '', f)
f = re.sub(r'\(+', '', f)
f = re.sub(r'\)+', '', f)

sentences = sent_tokenize(f)
my_sentence = []

# defining word list

word_list = ["notional amount", "Notional Amount", "Pay", "Receive", "Counter Party", "Reference
Entity", "reference entity"]


def sentenceFinder(sentences, word_list):
    for word in word_list:
        for sent in sentences:
            if word in sent:
                my_sentence.append(sent)

    return my_sentence


output = sentenceFinder(sentences, word_list)
```

Figure 18: Code snippet of script to extract unstructured sentences containing keywords

Once we had the unstructured sentences, we could begin the tagging of the data. As this a cumbersome process, extreme precision and proper formatting of output data(post tagging) needs to be present in order to ensure higher success rate of the manual work being put. Keeping these as our requirements, we built a Text Annotation Tool based on the Django framework which allows us to upload and tag datasets manually. It supports up to 26 different entity labelling.

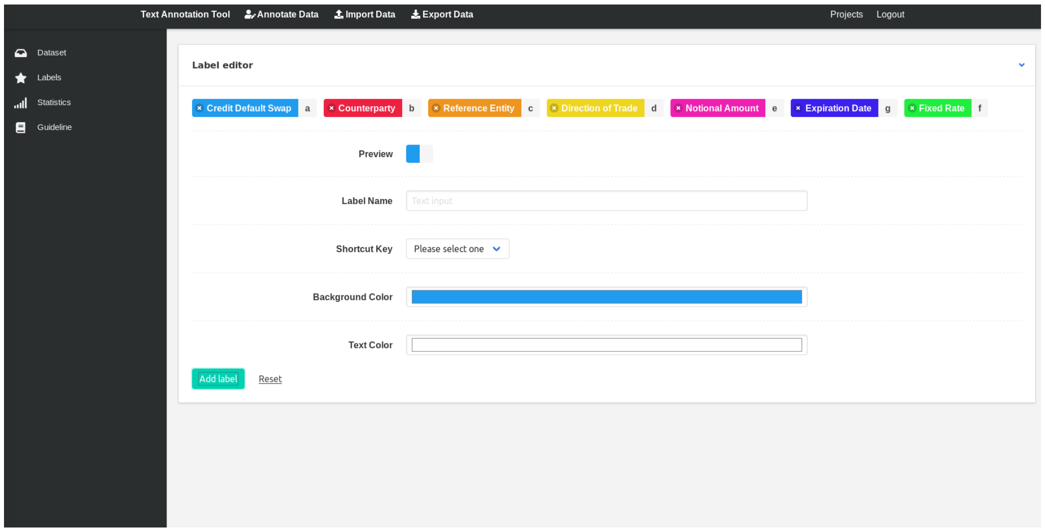
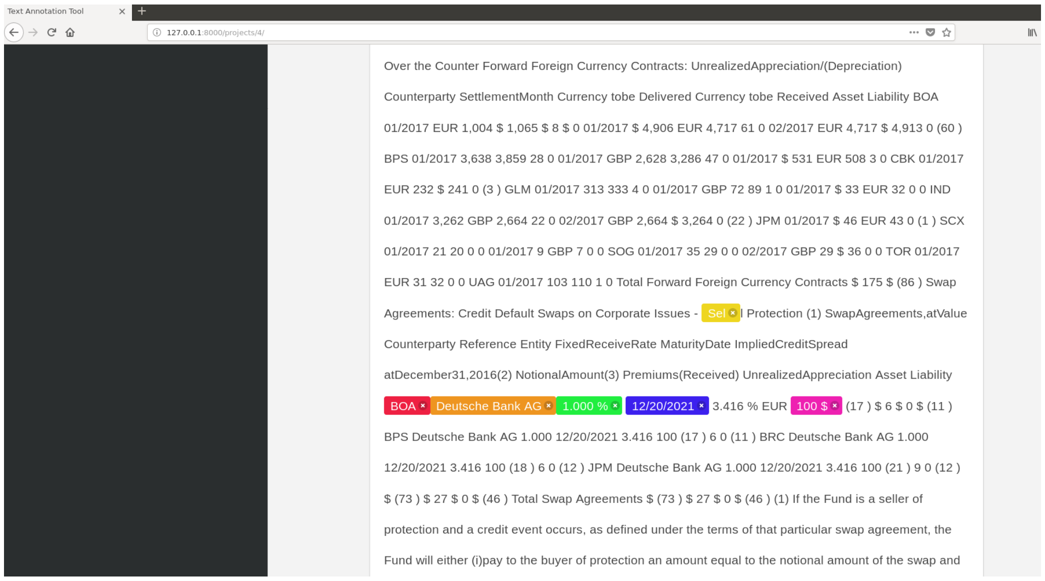


Figure 19: Defining Lables in the Tool



Over the Counter Forward Foreign Currency Contracts: UnrealizedAppreciation/(Depreciation) Counterparty SettlementMonth Currency tobe Delivered Currency tobe Received Asset Liability BOA 01/2017 EUR 1,004 $ 1,065 $ 8 $ 0 01/2017 $ 4,906 EUR 4,717 61 0 02/2017 EUR 4,717 $ 4,913 0 (60) BPS 01/2017 3,638 3,859 28 0 01/2017 GBP 2,628 3,286 47 0 01/2017 $ 531 EUR 508 3 0 CBK 01/2017 EUR 232 $ 241 0 (3) GLM 01/2017 313 333 4 0 01/2017 GBP 72 89 1 0 01/2017 $ 33 EUR 32 0 0 IND 01/2017 3,262 GBP 2,664 22 0 02/2017 GBP 2,664 $ 3,264 0 (22) JPM 01/2017 $ 46 EUR 43 0 (1) SCX 01/2017 21 20 0 0 01/2017 9 GBP 7 0 0 SOG 01/2017 35 29 0 0 02/2017 GBP 29 $ 36 0 0 TOR 01/2017 EUR 31 32 0 0 UAG 01/2017 103 110 1 0 Total Forward Foreign Currency Contracts $ 175 $ (86) Swap Agreements: Credit Default Swaps on Corporate Issues - Sel ×l Protection (1) SwapAgreements,atValue Counterparty Reference Entity FixedReceiveRate MaturityDate ImpliedCreditSpread atDecember31,2016(2) NotionalAmount(3) Premiums(Received) UnrealizedAppreciation Asset Liability BOA × Deutsche Bank AG × 1.000 % × 12/20/2021 × 3.416 % EUR 100 $ × (17) $ 6 $ 0 $ (11) BPS Deutsche Bank AG 1.000 12/20/2021 3.416 100 (17) 6 0 (11) BRC Deutsche Bank AG 1.000 12/20/2021 3.416 100 (18) 6 0 (12) JPM Deutsche Bank AG 1.000 12/20/2021 3.416 100 (21) 9 0 (12) $ (73) $ 27 $ 0 $ (46) Total Swap Agreements $ (73) $ 27 $ 0 $ (46) (1) If the Fund is a seller of protection and a credit event occurs, as defined under the terms of that particular swap agreement, the Fund will either (i)pay to the buyer of protection an amount equal to the notional amount of the swap and

Figure 20: Tagging in Action

Finally, it gives us a graphical user interface (GUI) for tagging datasets and helps us automatically format the manually tagged dataset into our desired format. Upon successfully tagging a report, you can export the final tagged report to contain sentences in either a .csv or .json format and download them. These tagged files are then used to train the CRF models which is discussed in the NLP report.

4 Results

4.1 Corpus Generation and Table Extraction

We had reduced the size of the original data set from 146 GB to 71GB (44GB for N-CSR, 15 GB for N-CSRS, 12 GB for N-Q) which is a 51.37 % reduction in size and made a corpus of the folders. We then ran our script to extract tables, the results of which are shown below.

Type of Report	Total number in Corpus	Total Extracted as Structured
N-CSR	7750	5198
N-CSRS	3304	2168
N-Q	3670	2720

Table 1: Number of reports which have structured data

To determine the accuracy of the script in extracting the data, we manually went through 100 random reports in each of the N-CSR, N-CSRS and N-Q folders and checked whether all the information was extracted.

Type of Report	Number of Reports Checked	Data accurately extracted
N-CSR	100	93
N-CSRS	100	89
N-Q	100	92

Table 2: Accuracy Test

We take a weighted average of all three folders and see that the overall accuracy of the script is **91.33%**. Since the accuracy of the script is dependent on the HTML tags also, it might have been the case that due to some inaccuracies in the tags, the accuracy has gone down. We then created a master script which would take in a financial report and extract and format the CDS tables and return it.

4.2 Unified CDS Table

We combined all the extracted data after cleaning it to generate a unified csv file containing 16813 rows of data. This can be accessed by anyone who visits our website and they can search for specific Funds' names and will get all the CDS dealings of the funds with an option to download it as a csv file.

4.3 Tagging of Unstructured Data

We successfully tagged around 1200 CDS sentences using our Text Annotation tool to help us create the training data for our CRF model which is discussed in detail in the NLP Report.

5 Difficulties Encountered

We encountered challenges in cleaning the data initially due to the volume of data that was present. We had to figure out how to restructure the initial data set and split it in order to proceed with the project in a sequential manner. There was also a lot of data that we didnt need and it was a challenge to identify this data and remove it in order to make our processing much faster. It was also impossible to come up with a script which would format all the CDS tables with 100 percent accuracy as each fund had their own style of reporting. We thus had to come up with a script after identifying patterns in the data. Even though this script worked on most of the reports, we still had to manually go through each report to make sure that the tables were formatted perfectly in order to create our unified csv file and this was a time consuming process.

6 Conclusion

Summarizing the report, we show the importance of Credit Default Swaps(CDS) and the power that it has on the economy of a country. There is currently no easy way to access the required information about CDS from the web. We thus highlight the need for a centralized database consisting of all the information regarding CDS dealings. We demonstrate the process that can be used to do this which would start with using Python and Perl scripts to get the data. Once we get the data, we process the data to get to the actual useful data and also divide it into two parts, namely the unstructured and structured part. We extract the structured information using rule based extraction approach. We find that even though the majority of the data is extracted through this, there is still unstructured data which cannot be extracted using this approach. This is the reason why we need to use Natural Language Processing methods which is further discussed in the NLP Report.

References

A.Guettler ,T.Adam. Pitfalls and Perils of Financial Innovation: The Use of CDS by Corporate Bond Funds. Jan 10, 2015.

Beautiful Soup Documentation. Available at:
`https://beautiful-soup-4.readthedocs.io/en/latest/`

Credit Default Swaps (n.d). PIMCO Funds. Available at:
`https://global.pimco.com/en-gbl/resources/education/understanding-credit-default-swaps`

P. Wayne (2018, October 6). Credit Default Swaps: An Introduction [Online]. Available: https://www.investopedia.com/articles/optioninvestor/08/cds.asp

W.Jiang, Z.Zhu. Mutual Funds Holdings of Credit Default Swaps. September 2016.