**Data Extraction from Unstructured Txt File**

**Goal**: To extract data from unstructured text files using Python and regular expressions (regex).

**Objective**: To identify and extract important information from unstructured text files and save it in a clean Excel format. The extracted information includes:

* Company Name
* Ticker
* Meeting Date
* Meeting Type
* Security
* ISIN
* Agenda Number
* Prop.#
* Proposal
* Proposal Type
* Proposal Vote
* For/Against Management

**Approach: -** The method consists of two parts:

**1.** Extracting proposal details — capturing data from Prop.# through For/Against Management fields.

* From the main file “appleton\_npx 1 1”, we extracted another text file named “proposals.txt” which contains details like Prop.#, Proposal, Proposal Type, Proposal Vote, For/Against Management.
* A specific regular expression is used to find all proposal tables.
* The regex pattern identifies blocks that start with a long dashed line and the header “Prop.# Proposal”, ending before the next dashed line or the </TABLE> tag to ensure all blocks are captured, including the last one.
* Using re.findall(), the script extracts these proposal blocks as strings.
* The output file is created by joining these blocks with double newlines for readability.
* A final dashed separator line is appended at the end for consistent formatting.
* Finally, a line reading “Management position unknown” \* is added as a definitive stopping condition for any subsequent scripts.

**2.** Extracting company information separately.

* The script is broken down into several modular functions, each with a specific responsibility.
* extract\_company\_headers(input\_file): Extracts the header block for each company meeting. It reads the Appleton file uses a single, multi-line regular expression to find and capture all header blocks and return a dataframe where each row corresponds to one company meetings.
* extract\_proposals(proposals\_file): It reads the file line by line, using the proposal header to detect where each company's data starts. Each company's block is given a unique ID, and proposal lines are stored as dictionaries in a list, tagged with that ID.
* post\_process\_proposals(rough\_data): This function cleans and structures the raw proposal data. It merges multi-line proposals and splits director lists into separate entries. The result is a clean DataFrame where each row represents a complete proposal or director.
* merge\_and\_save(): Combines the two separate datasets (headers and proposals) and produces the final output file.It performs a pandas.merge operation, joining the two DataFrames on the common IDs column. It then re-orders the columns into a logical sequence before saving.
* main (): The main controller function that defines the filenames and executes the entire workflow in the correct sequence.

**Tools Used: -**

* Rgex
* Python
* Pandas

**Problem faced: -**

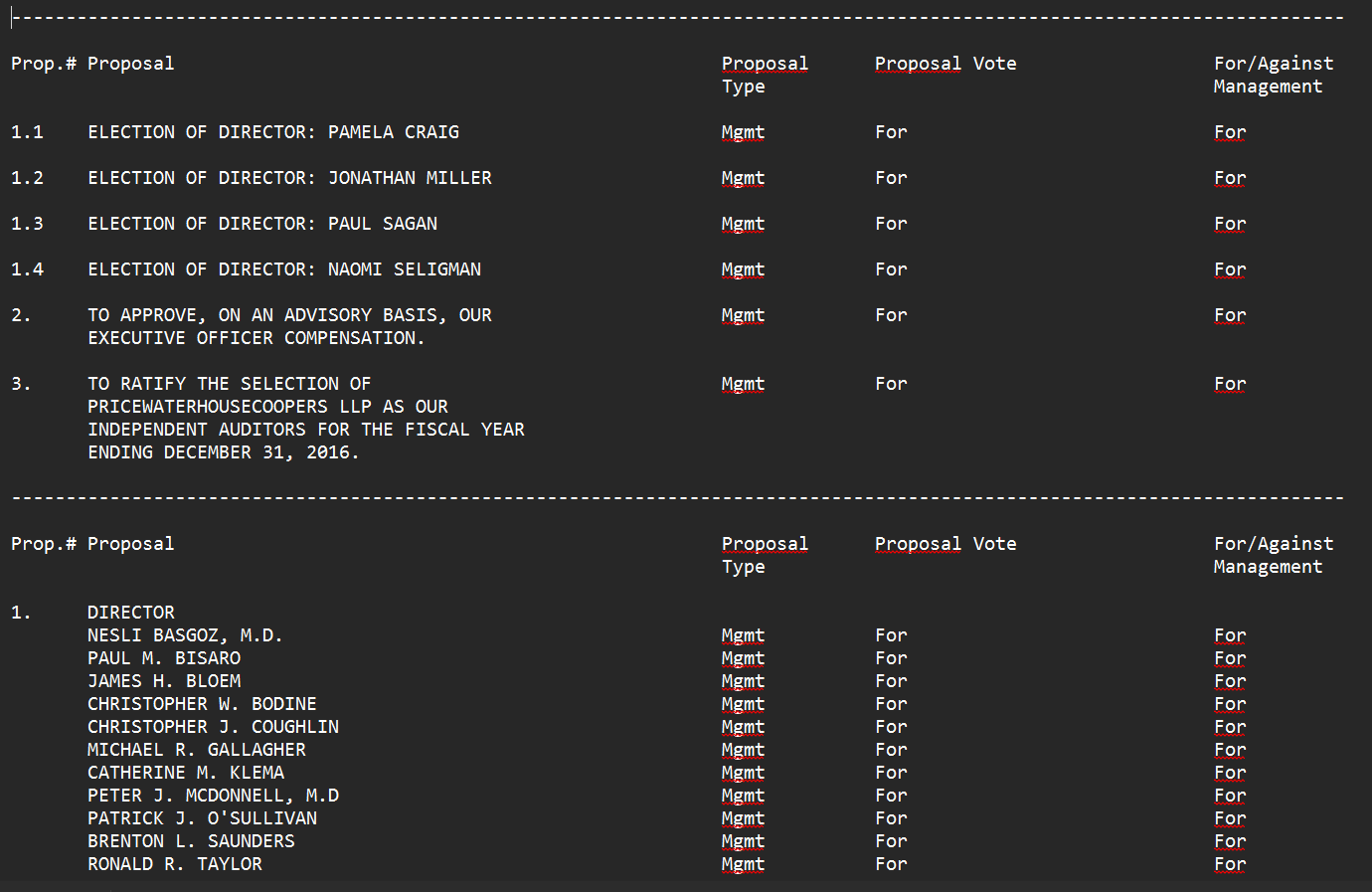
* Initially, I planned to use regex to directly extract the inner details, and while the data extraction was working well, the indentation didn't align properly when pasting the results into Excel.
* Handling Multi-Line Proposal Descriptions: A single proposal's text often spanned multiple lines. The initial script treated each line as a new, separate row, which was incorrect. We implemented a **post-processing loop**. This logic identifies any row with a blank Prop. # and correctly merges its proposal text with the preceding row.
* Differentiating Sub-Items from Multi-line Text (Director Lists): The Code needed to distinguish between a multi-line description and a list of individual directors under a single "DIRECTOR" heading. Merging these director names was a critical error. We created a more advanced, stateful parsing logic. The script now uses a flag (is\_in\_director\_list) that activates when it reads a "DIRECTOR" header. This flag tells the merging logic to treat subsequent lines without a proposal number as new, separate rows instead of merging them.
* When I created a separate TXT file for the proposals, we had already defined how to extract the data. However, in the last proposal section, it was followed by a </TABLE> tag instead of the expected --- separator required by the pattern, so we added an OR condition to handle the table tag as well.
* For parsing and merging the Excel file, we simply assigned an ID to each company and decided to merge it with the proposal number but there were two separate meetings for "Bank of America" under a single company ID. We replaced the logic with a more robust method that correctly identified a new company by its unique table header, ensuring each meeting was assigned a distinct ID.
* A meeting with a single proposal numbered "01" was followed by another company's meeting starting with "1A". Simple sorting logic failed to identify this as a new company and grouped them incorrectly.

**Output and Input:**

Part I Input: -

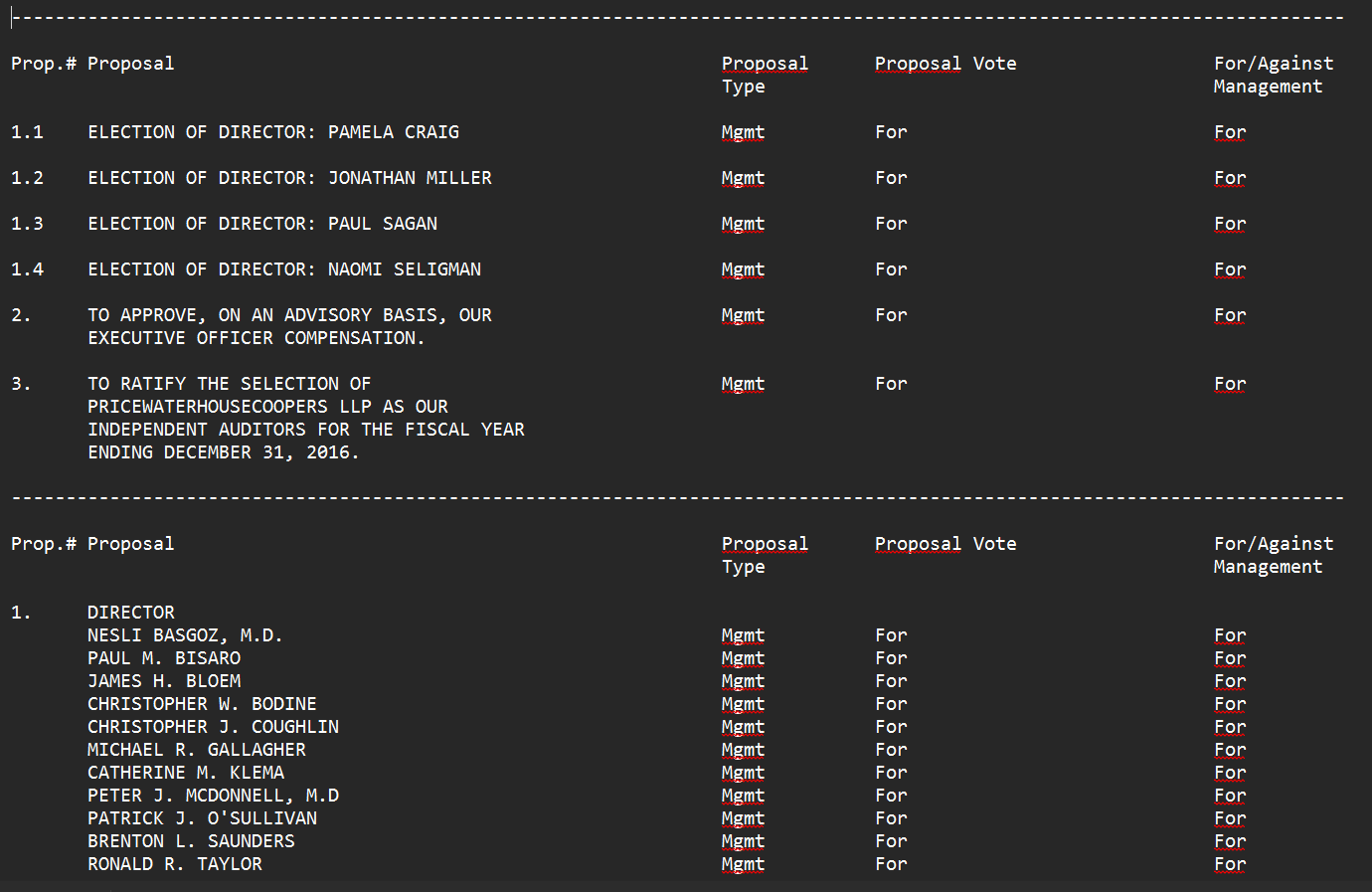


Output: -

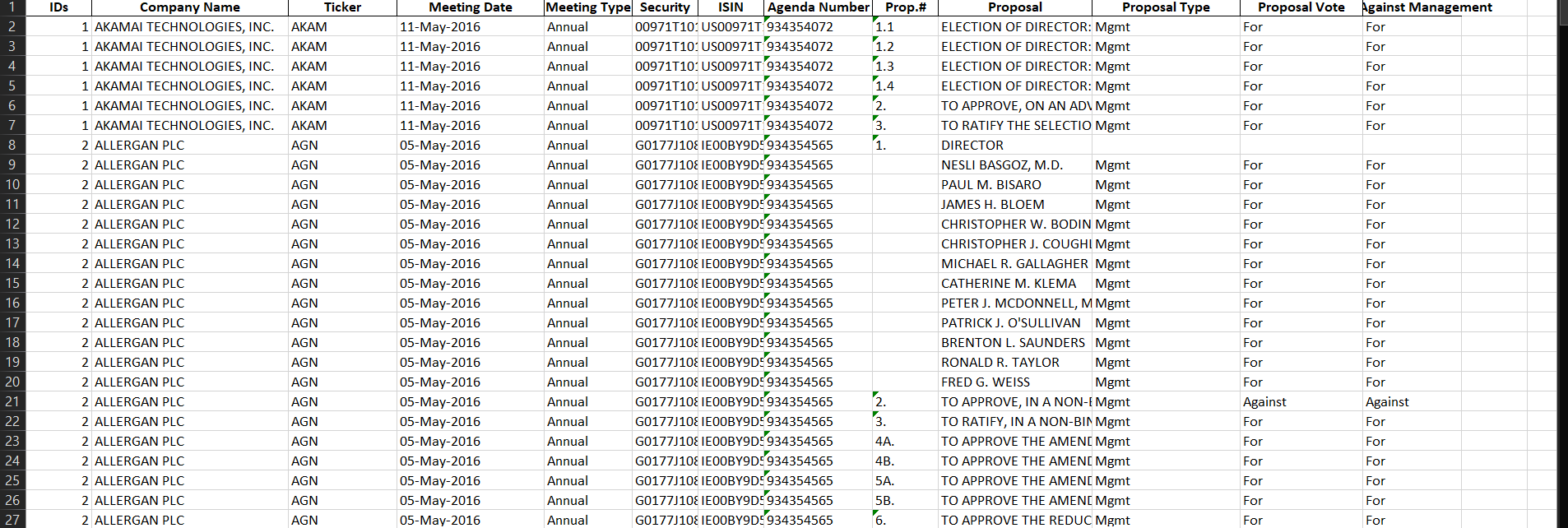


Part II

Input: -



Output: -



**Conclusion: -**

This project successfully developed a robust, automated workflow to parse, clean, and structure complex proxy voting filings. The primary objective—to convert a semi-structured text document into a clean, analysis-ready Excel spreadsheet—was fully achieved through an iterative development process.

The final solution consists of two key Python scripts. The first script effectively isolates and extracts all relevant proposal tables from the main source file using a precise regular expression. The second, more comprehensive script, utilizes the pandas library to parse this extracted data into a structured DataFrame, intelligently handle formatting challenges like multi-line descriptions, and merge it with the corresponding company header information.

Throughout the process, we systematically identified and solved a series of edge cases related to data grouping, pattern matching, and output formatting. The final result is a reliable, well-documented set of tools that efficiently transforms a difficult-to-use text file into a valuable and accessible dataset, demonstrating a successful application of data parsing and cleaning techniques.