**WORD2VEC CLASSIFICATION OF ALL THE CHURCHES BASED ON THE CHURCH DESCRIPTIONS.**

**CONTENTS**

**1) IMPORTING NECESSARY LIBRARIES**

**2) LOADING THE DATA**

**3) PRE-PROCESSING THE TEXT DATA**

**3)LOADING THE WORD2VEC MODEL**

**5)COSINE SIMILARITY SCORE CALCULATIONS**

**6) OVERLAP SCORE CALCULATION**

**7)OVERLAP OF METHODS**

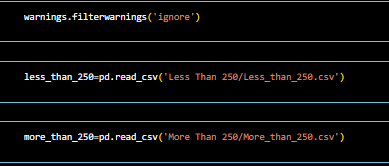
**8)OUTPUTS**

**1) IMPORTING NECESSARY LIBRARIES**



For explanation refer to fasttext test code documentation page number 2.

**2) LOADING THE DATA**



less\_than\_250

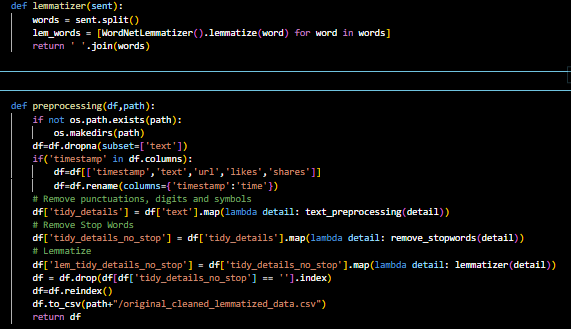
**3) PRE-PROCESSING THE TEXT DATA**

**I)**

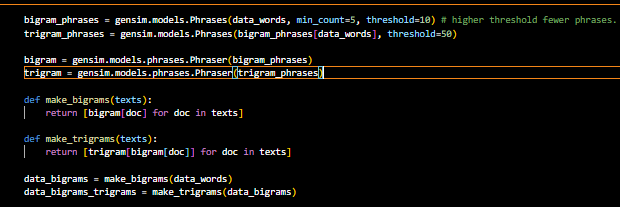


* + Convert to lowercase: Make all letters small.
  + Remove URLs: Deletes web addresses.
  + Remove HTML tags: Removes any HTML code.
  + Remove punctuation: Gets rid of symbols like ., !, and ?.
  + Remove newlines: Deletes line breaks.
  + Remove words with digits: Removes words that contain numbers.
  + Remove emojis: Deletes emojis
  + Load stopwords: Loads a list of common words to remove (like "and", "the").
  + Remove stopwords: Removes these common words from the text.
  + Special case for "church": If the cleaned text starts with "church", it removes this word

**II)**



1. **Lemmatizer Function:**
   * lemmatizer(sent):
     + Splits the sentence into words.
     + Converts each word to its base form (lemma).
     + Joins the words back into a sentence.
2. **Preprocessing Function:**
   * preprocessing(df, path):
     + Create directory: Creates the directory if it doesn't exist.
     + Drop missing texts: Removes rows with missing 'text'.
     + Rename columns: If 'timestamp' exists, rename it to 'time' and select specific columns.
     + Clean text: Applies text\_preprocessing to remove unwanted parts.
     + Remove stopwords: Applies remove\_stopwords to eliminate common words.
     + Lemmatize: Applies lemmatizer to convert words to their base form.
     + Remove empty rows: Deletes rows where 'tidy\_details\_no\_stop' is empty.
     + Reindex and save: Reindexes the DataFrame and saves it as a CSV file.
     + Return: Returns the processed DataFrame.



The code is using the `gensim` library to detect and generate bigram (two-word) and trigram (three-word) phrases from a given text corpus (`data\_words`). This is a common preprocessing step in natural language processing tasks, as it helps capture multi-word expressions and can improve the quality of text representation.

1. \*\*Generating Bigram and Trigram Phrases\*\*:

- `gensim.models.Phrases` is used to detect and generate bigram and trigram phrases from the `data\_words` corpus.

- `min\_count=5` means only words appearing at least 5 times will be considered for phrase detection.

- `threshold=10` (for bigrams) and `threshold=50` (for trigrams) determine the scoring threshold for forming new phrases. Higher thresholds result in fewer phrases.

- Trigram phrases are generated from the previously generated bigram phrases.

2. \*\*Applying Phrase Transformations\*\*:

- `gensim.models.phrases.Phraser` is used to transform tokenized documents (lists of words) into lists of bigram or trigram phrases.

- `bigram` and `trigram` instances are created with the generated bigram and trigram phrases, respectively.

3. \*\*Helper Functions\*\*:

- `make\_bigrams` and `make\_trigrams` are custom functions to apply bigram and trigram phrase transformations to a list of tokenized documents (`texts`).

- `make\_bigrams` applies the `bigram` transformation, while `make\_trigrams` applies both `bigram` and `trigram` transformations.

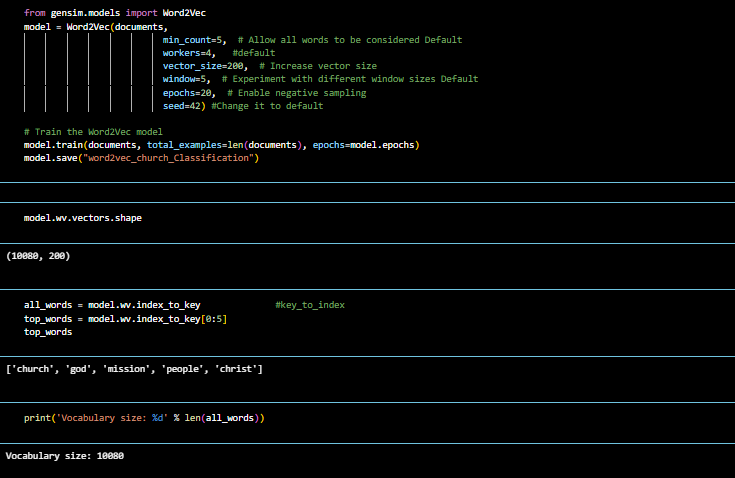
4. \*\*Applying Transformations to Data\*\*:

- `data\_bigrams` is a list of tokenized documents, where each document contains bigram phrases.

- `data\_bigrams\_trigrams` is a list of tokenized documents, where each document contains both bigram and trigram phrases.

The resulting `data\_bigrams` and `data\_bigrams\_trigrams` can be used as input for further natural language processing tasks, such as topic modeling or text classification. By capturing multi-word expressions through bigram and trigram phrases, the text representation can be improved, potentially leading to better performance in downstream tasks.

**4)LOADING THE WORD2VEC MODEL**



Refer to word2vec vs bert documentation for explanation of word2vec model and its parameters in detail.

**5)COSINE SIMILARITY SCORE CALCULATIONS**



**The above code snippet is iterating over each row in the `daily\_df1` dataset. For each row, it does the following:**

1. It takes the text from the `'tidy\_details\_no\_stop'` column and converts it into a list of words (`detail`). Then, it filters out the words that are not present in the model's vocabulary (`vocab`), and stores the remaining words in the `sent` list.

2. If there are any words left in the `sent` list, it creates a list of numerical representations (`word\_vectors`) for those words using the FASTEXT model's word vectors (`model.wv`).

3. It calculates the maximum cosine similarity between the `word\_vectors` and the `growth\_vectors` list, and **stores the result in `cos\_sims\_growth`**. It does the same for the `ideology\_vectors` list, and stores the result in **`cos\_sims\_ideology**`. Cosine similarity is a measure of how similar two sets of numbers are.

4. It sorts the `cos\_sims\_growth` and `cos\_sims\_ideology` scores lists in ascending order.

5. It calculates the mean of the top 3 cosine similarity scores from `cos\_sims\_growth` and `cos\_sims\_ideology`, and stores the results in `mean\_sim\_growth` and `mean\_sim\_ideology`, respectively.

6. It updates the `'growth\_sim\_score'` and `'ideology\_sim\_score'` columns in the `daily\_df1` dataset with the `mean\_sim\_growth` and `mean\_sim\_ideology` values for the current row.

7. It compares `mean\_sim\_growth` and `mean\_sim\_ideology`, and updates the `'Upper\_score\_Aim'` column in the `daily\_df1` dataset with 1 if `mean\_sim\_growth` is greater than `mean\_sim\_ideology`, 0 if it's less, and -1 if they're equal.

8. It calculates the mean of the top 6 cosine similarity scores (greater than 0.2) from `cos\_sims\_growth` and `cos\_sims\_ideology`, and stores the results in `mean\_sim\_growth` and `mean\_sim\_ideology`, respectively.

9. It updates the `'growth\_sim\_score\_lower'` and `'ideology\_sim\_score\_lower'` columns in the `daily\_df1` dataset with the new `mean\_sim\_growth` and `mean\_sim\_ideology` values for the current row.

10. It compares the new `mean\_sim\_growth` and `mean\_sim\_ideology` values, and updates the `'Lower\_score\_Aim'` column in the `daily\_df1` dataset with 1 if `mean\_sim\_growth` is greater than `mean\_sim\_ideology`, 0 if it's less, and -1 if they're equal or both are NaN (not a number).

11. It calculates the overall mean of `cos\_sims\_growth` and `cos\_sims\_ideology`, and stores the results in `mean\_sim\_growth` and `mean\_sim\_ideology`, respectively.

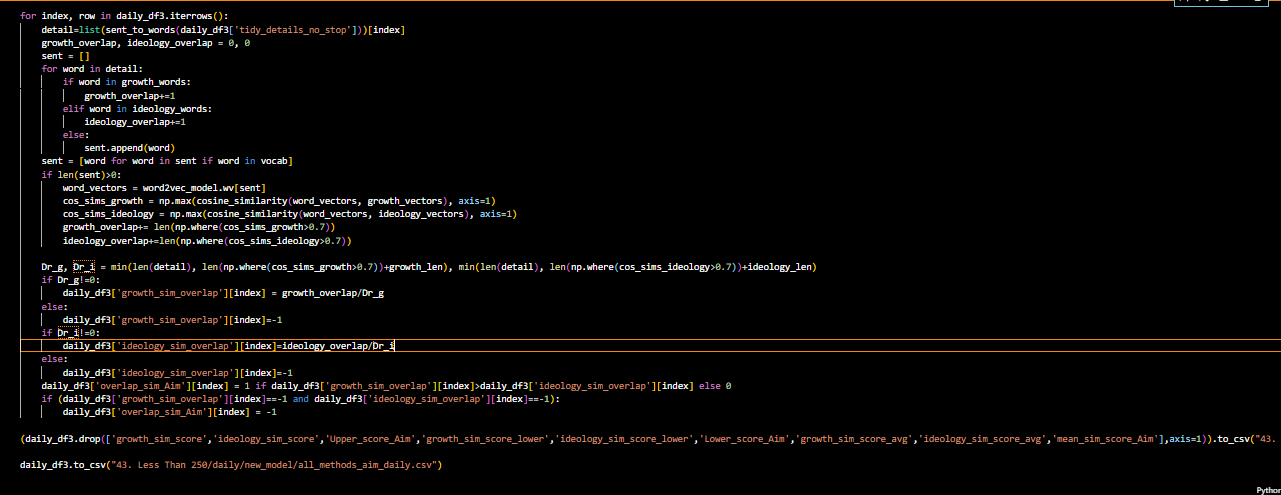
12. It updates the `'growth\_sim\_score\_avg'` and `'ideology\_sim\_score\_avg'` columns in the `daily\_df1` dataset with the overall `mean\_sim\_growth` and `mean\_sim\_ideology` values for the current row.

13. It compares the overall `mean\_sim\_growth` and `mean\_sim\_ideology` values, and updates the `'mean\_sim\_score\_Aim'` column in the `daily\_df1` dataset with 1 if `mean\_sim\_growth` is greater than `mean\_sim\_ideology`, and 0 if it's less.

Finally, the code saves different versions of the `daily\_df1` dataset to separate CSV files, excluding certain columns in each file.

**In summary, the code is calculating different measures of similarity between the text in each row and the predefined lists of growth and ideology words, and storing the results in various columns in the `daily\_df1` dataset. It then saves different versions of the dataset to separate CSV files, excluding certain columns in each file**

**6) OVERLAP SCORE CALCULATION**



The provided code seems to be written in Python and is performing some operations on a pandas DataFrame called `daily\_df3`. Here's a breakdown of what the code is doing:

1. It iterates over each row in the `daily\_df3` DataFrame using the `iterrows()` method.

2. For each row, it tokenizes the `tidy\_details\_no\_stop` column using the `sent\_to\_words` function and stores the resulting list of words in the `detail` variable.

3. It initializes two counters, `growth\_overlap` and `ideology\_overlap`, to keep track of the overlap between the `detail` words and predefined lists of `growth\_words` and `ideology\_words`, respectively.

4. It creates an empty list `sent` to store words from `detail` that are present in a `vocab` set.

5. For each word in `detail`, it checks if the word is present in `growth\_words` or `ideology\_words` and increments the corresponding overlap counter.

6. It then computes the cosine similarity between the word vectors of the words in `sent` and predefined `growth\_vectors` and `ideology\_vectors` using a pre-trained `word2vec\_model`.

7. It updates the `growth\_overlap` and `ideology\_overlap` counters by adding the number of word vectors with cosine similarity greater than 0.7 with respect to `growth\_vectors` and `ideology\_vectors`, respectively.

8. It calculates two denominators, `Dr\_g` and `Dr\_i`, based on the length of `detail` and the number of word vectors with high cosine similarity to `growth\_vectors` and `ideology\_vectors`, respectively.

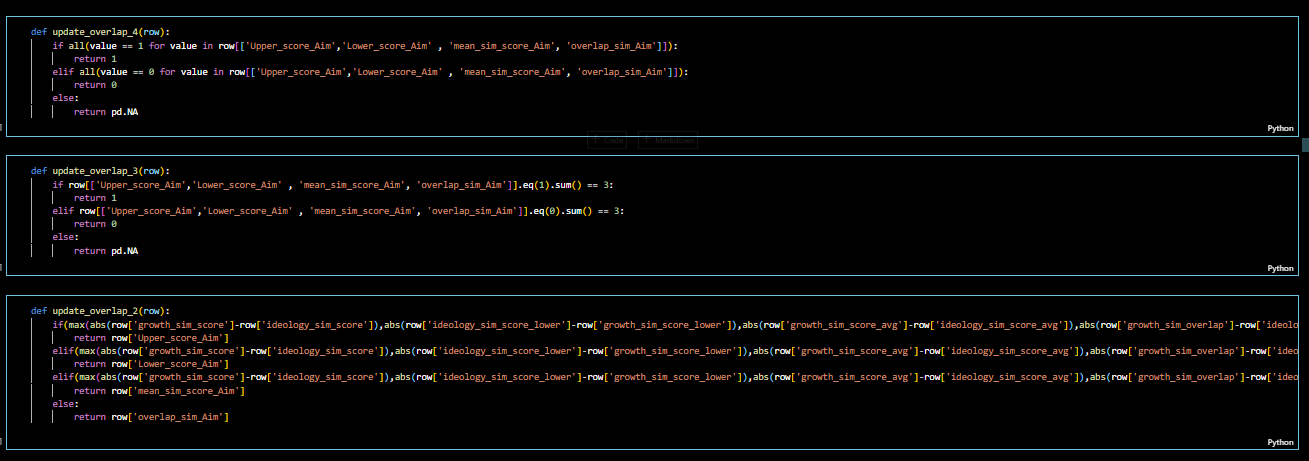
9. It computes the `growth\_sim\_overlap` and `ideology\_sim\_overlap` columns in `daily\_df3` by dividing the corresponding overlap counters by `Dr\_g` and `Dr\_i`, respectively. If the denominator is zero, it assigns a value of -1 to the corresponding column.

10. It computes the `overlap\_sim\_Aim` column in `daily\_df3` based on whether the `growth\_sim\_overlap` or `ideology\_sim\_overlap` is higher. If both values are -1, it assigns -1 to `overlap\_sim\_Aim`.

11. Finally, it saves the modified `daily\_df3` DataFrame with the new columns to a CSV file named "43. Less Than 250/daily/new\_model/all\_methods\_aim\_daily.csv", and also saves a subset of the DataFrame, excluding some columns, to another CSV file named "43. Less Than 250/daily/new\_model/overlap\_sim\_score\_aim\_daily.csv".

**In summary, this code is calculating some additional measures of similarity between the text in each row and the predefined lists of growth and ideology words, and storing the results in new columns in the `daily\_df1` dataset. It then saves different versions of the dataset to separate CSV files, including or excluding certain column.**

**7)OVERLAP OF METHODS**



**The term "overlap of 4" refers to a scenario where all four similarity methods—mean similarity, lower bound similarity, upper bound similarity, and overlap similarity—yield the same result. In other words, when examining a dataset, if all four methods produce identical outcomes for a given set of keywords or concepts, it indicates a significant alignment across multiple measures of similarity.**

**Likewise, the concept of "overlap of 3" signifies that any three out of the four similarity methods lead to the same conclusion. This suggests a substantial degree of agreement among the majority of similarity metrics, further reinforcing the consistency in evaluating the relevance or association of keywords within the dataset.**

**Similarly, an "overlap of 2" implies that a pair of similarity methods among the four produce matching results. While not as comprehensive as an overlap of 3 or 4, this still indicates a notable level of concordance between specific pairs of similarity measures, highlighting areas of shared interpretation or relevance within the dataset.**

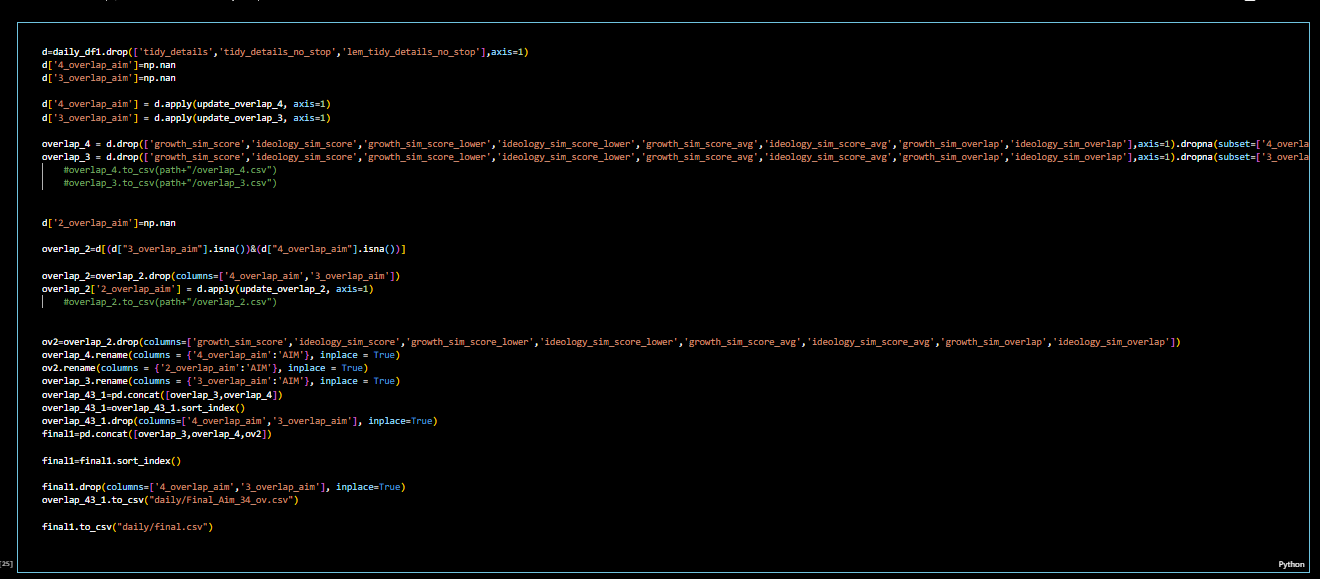
These are functions designed to update a column ('overlap\_sim\_Aim') in a DataFrame based on conditions involving other columns. Let's break down each function:

**1**. **update\_overlap\_4**: This function checks if all four specified columns have a value of 1. If they do, it returns 1. If all columns have a value of 0, it returns 0. Otherwise, it returns NA (null).

**2**. **update\_overlap\_3**: Similar to the previous function, but it only considers cases where exactly three out of the four columns have a value of 1. If this condition is met, it returns 1. If three columns have a value of 0, it returns 0. Otherwise, it returns NA.

**3**. **update\_overlap\_2**: This function is more complex. It calculates the absolute differences between pairs of values from various columns and selects the maximum difference. Based on which pair of columns has the maximum difference, it returns the corresponding value from another column ('Upper\_score\_Aim', 'Lower\_score\_Aim', 'mean\_sim\_score\_Aim', or 'overlap\_sim\_Aim'). If there's a tie or no conditions are met, it returns the value from 'overlap\_sim\_Aim'.

**In simpler terms, these functions evaluate conditions involving multiple columns and update a specific column based on those conditions. They help to automate decision-making based on certain criteria within the DataFrame.**



This script performs several operations on a DataFrame `daily\_df1` to update and manipulate columns based on certain conditions. Let's break down each step in detail:

1. **Data Preparation**:

- The script first creates a new DataFrame `d` by dropping certain columns ('tidy\_details', 'tidy\_details\_no\_stop', 'lem\_tidy\_details\_no\_stop') from `daily\_df1`.

- Two new columns, '4\_overlap\_aim' and '3\_overlap\_aim', are added to `d` with NaN (missing) values.

2. \*\***Updating '4\_overlap\_aim' and '3\_overlap\_aim**'\*\*:

- The script applies the functions `update\_overlap\_4` and `update\_overlap\_3` row-wise to update the values in '4\_overlap\_aim' and '3\_overlap\_aim' columns, respectively.

- Two DataFrames, `overlap\_4` and `overlap\_3`, are created by dropping rows with NaN values in '4\_overlap\_aim' and '3\_overlap\_aim', respectively.

3. \*\***Updating '2\_overlap\_aim'**\*\*:

- A new column '2\_overlap\_aim' is added to `d` with NaN values.

- Rows with NaN values in both '4\_overlap\_aim' and '3\_overlap\_aim' are selected to update '2\_overlap\_aim' column.

- The function `update\_overlap\_2` is applied row-wise to update '2\_overlap\_aim' column.

4. \*\***Data Formatting**\*\*:

- Unnecessary columns related to similarity scores are dropped from `overlap\_2` DataFrame.

- Column names are renamed to 'AIM' in `overlap\_4`, `ov2`, and `overlap\_3` DataFrames.

- `overlap\_3` and `overlap\_4` DataFrames are concatenated into `overlap\_43\_1` DataFrame and sorted by index.

- Duplicate columns '4\_overlap\_aim' and '3\_overlap\_aim' are dropped from `overlap\_43\_1` DataFrame.

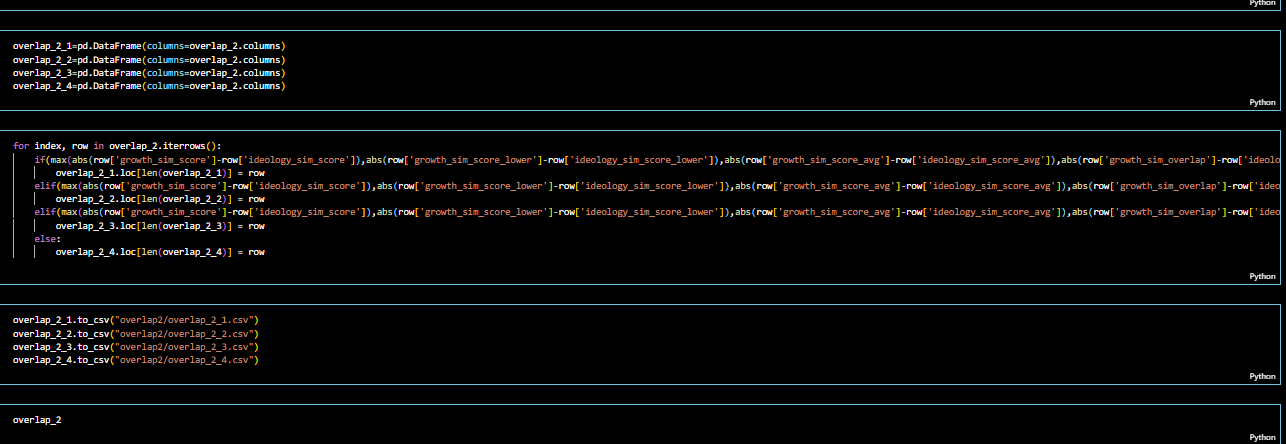
- The final DataFrame `final1` is created by concatenating `overlap\_3`, `overlap\_4`, and `ov2` DataFrames, sorted by index, and duplicate columns dropped.

5. \*\***Data Expor**t\*\*:

- `overlap\_43\_1` DataFrame is exported to a CSV file named "Final\_Aim\_34\_ov.csv" in the "daily" directory.

- `final1` DataFrame is exported to a CSV file named "final.csv" in the "daily" directory.

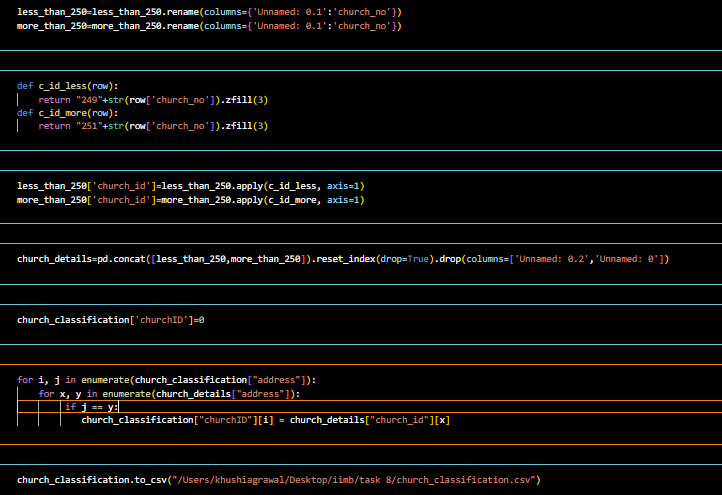
In summary, this script updates and manipulates columns in the DataFrame `daily\_df1` based on certain conditions, creates new DataFrames based on these updates, formats the data, and exports the final results to CSV files for further analysis.

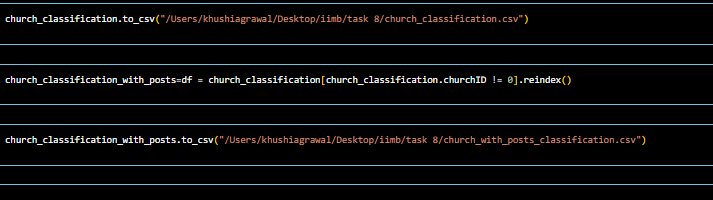


This code is performing filtering operations on a pandas DataFrame called overlap\_3 and creating subsets based on specific conditions. It appears that overlap\_3 contains columns related to similarity scores and overlap measures for some data. Here's a summary of what the code is doing:

1. **Filtering for Equal Scores**:
   * The code creates four new DataFrames: overlap\_3\_123, overlap\_3\_124, overlap\_3\_134, and overlap\_3\_234.
   * Each of these DataFrames is a subset of overlap\_3 where certain columns have equal values.
   * For example, overlap\_3\_123 contains rows where the values in Upper\_score\_Aim, Lower\_score\_Aim, and mean\_sim\_score\_Aim columns are all equal.
   * Similarly, overlap\_3\_124 contains rows where Upper\_score\_Aim, Lower\_score\_Aim, and overlap\_sim\_Aim have equal values.
   * The other two DataFrames (overlap\_3\_134 and overlap\_3\_234) filter for different combinations of equal values across these columns.
2. **Saving Filtered DataFrames to CSV**:
   * After creating the filtered DataFrames, the code saves them as separate CSV files using the to\_csv method.
   * The file names follow a pattern like overlap\_3\_123.csv, overlap\_3\_124.csv, overlap\_3\_134.csv, and overlap\_3\_234.csv, where the numbers correspond to the columns that have equal values.
   * These files are saved in a directory called overlap3.
3. **Saving Original DataFrames**:
   * Finally, the code saves the original overlap\_4 and overlap\_3 DataFrames to CSV files named overlap\_4.csv and overlap\_3.csv, respectively.

The purpose of this code is to extract and save specific subsets of the overlap\_3 DataFrame, where certain similarity score and overlap measure columns have equal values. These subsets may be useful for further analysis or processing in the broader context of the project. Additionally, the original DataFrames (overlap\_4 and overlap\_3) are also saved for reference or future use.





1. \*\*Renaming Columns\*\*:

- The code renames columns in the `overlap\_4`, `ov2`, and `overlap\_3` DataFrames.

- Specifically, it renames the columns `'4\_overlap\_aim'`, `'2\_overlap\_aim'`, and `'3\_overlap\_aim'` to `'AIM'`.

- Similarly, it renames the columns `'4\_overlap\_aim\_ind'`, `'2\_overlap\_aim\_ind'`, and `'3\_overlap\_aim\_ind'` to `'AIM\_ind'`.

- These rename operations are performed inplace, meaning the original DataFrames are modified.

2. \*\*Concatenating DataFrames\*\*:

- The code concatenates the `overlap\_3`, `overlap\_4`, and `ov2` DataFrames into a single DataFrame called `final`.

- The `pd.concat` function from pandas is used for this operation.

3. \*\*Defining a Function for Overlap Calculation\*\*:

- A function called `overlap\_3\_and\_more` is defined.

- This function takes a row of data as input and checks if at least three of the columns `'Upper\_score\_Aim'`, `'Lower\_score\_Aim'`, `'mean\_sim\_score\_Aim'`, and `'overlap\_sim\_Aim'` have the value 'Growth' or 'Ideology'.

- If at least three columns have the value 'Growth', it returns 'Growth'.

- If at least three columns have the value 'Ideology', it returns 'Ideology'.

- Otherwise, it returns 'Invalid'.

4. \*\*Applying the Function to the DataFrame\*\*:

- The `overlap\_3\_and\_more` function is applied to each row of the `final` DataFrame using the `apply` method.

- The result is stored in a new column called `'3\_and\_more\_overlap'`.

- Another column called `'3\_and\_more\_overlap\_ind'` is created by applying the `categorize\_value` function to the `'3\_and\_more\_overlap'` column.

5. \*\*Dropping a Column\*\*:

- The code drops the `"2\_overlap\_aim"` column from the `final` DataFrame.

6. \*\*Merging DataFrames\*\*:

- The code merges the `final\_word2vec` and `final\_fasttext` DataFrames based on the columns `'name'`, `'address'`, `'denomination'`, `'sub\_denomination'`, `'contacts'`, `'web'`, `'size'`, `'details'`, `'link'`, `'facebook'`, `'instagram'`, `'twitter'`, `'youtube'`, `'state'`, `'region'`, and `'division'`.

- The resulting merged DataFrame is called `church\_classification`.

- The `suffixes` parameter is used to handle overlapping column names by appending `'\_W2V'` and `'\_FT'` to the respective columns.

7. \*\*Defining Functions for Church ID Assignment\*\*:

- Two functions, `c\_id\_less` and `c\_id\_more`, are defined.

- These functions take a row of data as input and generate a `'church\_id'` based on the `'church\_no'` column.

- If `'church\_no'` is less than 250, the `'church\_id'` starts with "249" followed by the `'church\_no'` padded with leading zeros to make it three digits.

- If `'church\_no'` is greater than or equal to 250, the `'church\_id'` starts with "251" followed by the `'church\_no'` padded with leading zeros to make it three digits.

8. \*\*Applying Functions to Assign Church IDs\*\*:

- The `c\_id\_less` function is applied to the `less\_than\_250` DataFrame using the `apply` method, and the resulting `'church\_id'` values are stored in a new column.

- Similarly, the `c\_id\_more` function is applied to the `more\_than\_250` DataFrame to assign `'church\_id'` values.

9. \*\*Updating Church IDs in the Merged DataFrame\*\*:

- The code iterates over the `"address"` column in `church\_classification` and the `"address"` column in `church\_details`.

- If the addresses match, the corresponding `"churchID"` value in `church\_classification` is updated with the `"church\_id"` value from `church\_details`.

10. \*\*Saving the Final DataFrame\*\*:

- Finally, the `final` DataFrame is saved to a CSV file named "Final\_Aim.csv" using the `to\_csv` method.

In summary, the code performs various operations such as renaming columns, concatenating DataFrames, applying custom functions, merging DataFrames, assigning church IDs based on specific conditions, updating church IDs in a merged DataFrame, and saving the final DataFrame to a CSV file.

**8)OUTPUTS:**

**FOR LESS THAN 250 FILE:**

[**https://www.dropbox.com/scl/fo/t2yz5f966s46y8abxthsa/AM-LmcS-bQXrXLKOw29LvAg?rlkey=ytwy2f75t4dt6vlfoh6epmt6k&st=35z5rq9p&dl=0**](https://www.dropbox.com/scl/fo/t2yz5f966s46y8abxthsa/AM-LmcS-bQXrXLKOw29LvAg?rlkey=ytwy2f75t4dt6vlfoh6epmt6k&st=35z5rq9p&dl=0)

**FOR MORE THAN 250 FILE:**

[**https://www.dropbox.com/scl/fo/cbr7xw4uxk1zmlcvf4zof/AFwhU4OFWl3u3Sm\_83xYNOM?rlkey=g52npyyecz9tt285c7vmubfy5&st=df3cr5a2&dl=0**](https://www.dropbox.com/scl/fo/cbr7xw4uxk1zmlcvf4zof/AFwhU4OFWl3u3Sm_83xYNOM?rlkey=g52npyyecz9tt285c7vmubfy5&st=df3cr5a2&dl=0)