BRAVISA REPORTS

DOCUMENTATION FORMAT

Function Name: `function_name(parameters)`

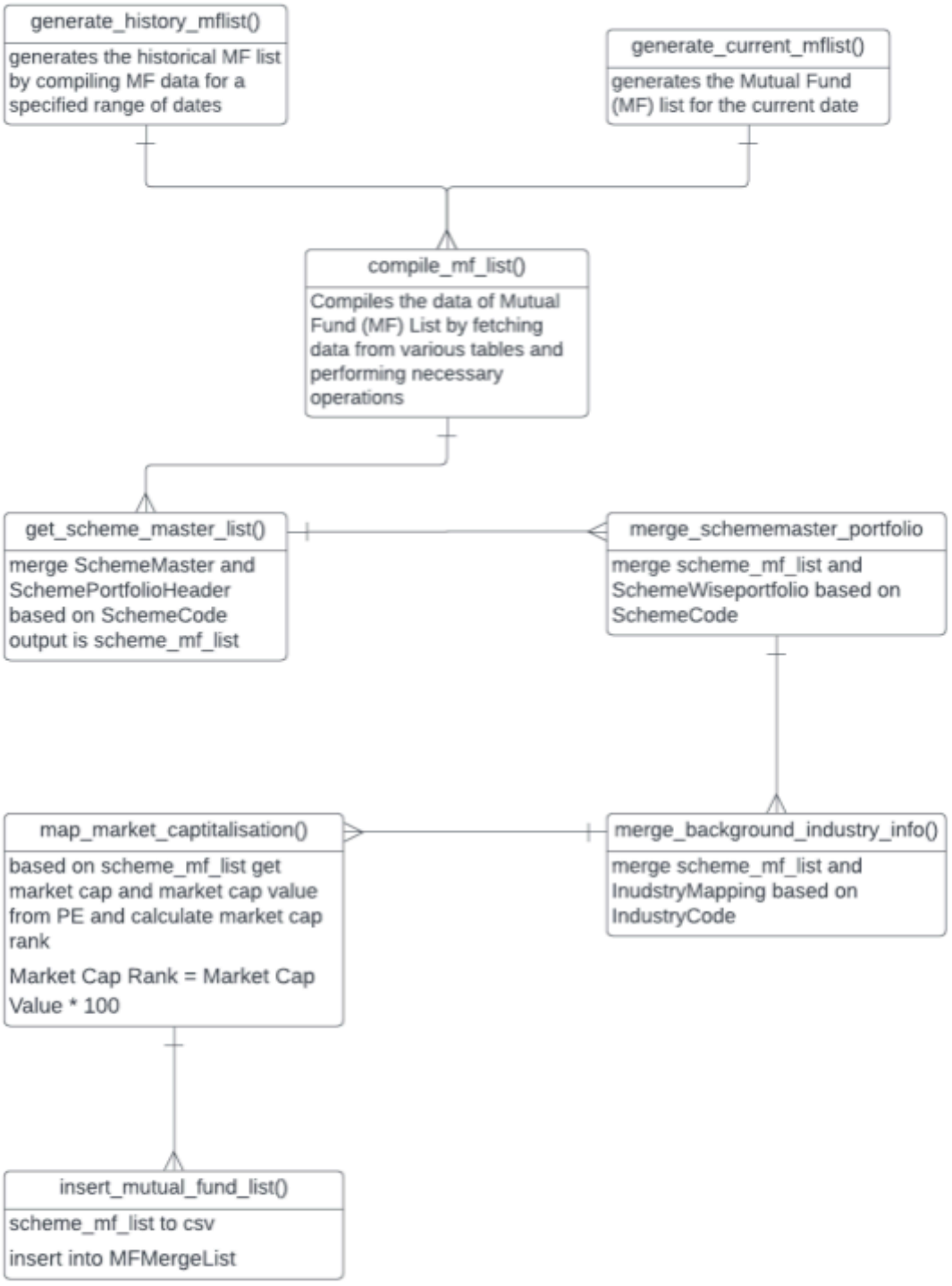
Parameters:
- `param1` (type): Description of param1.
- `param2` (type): Description of param2.

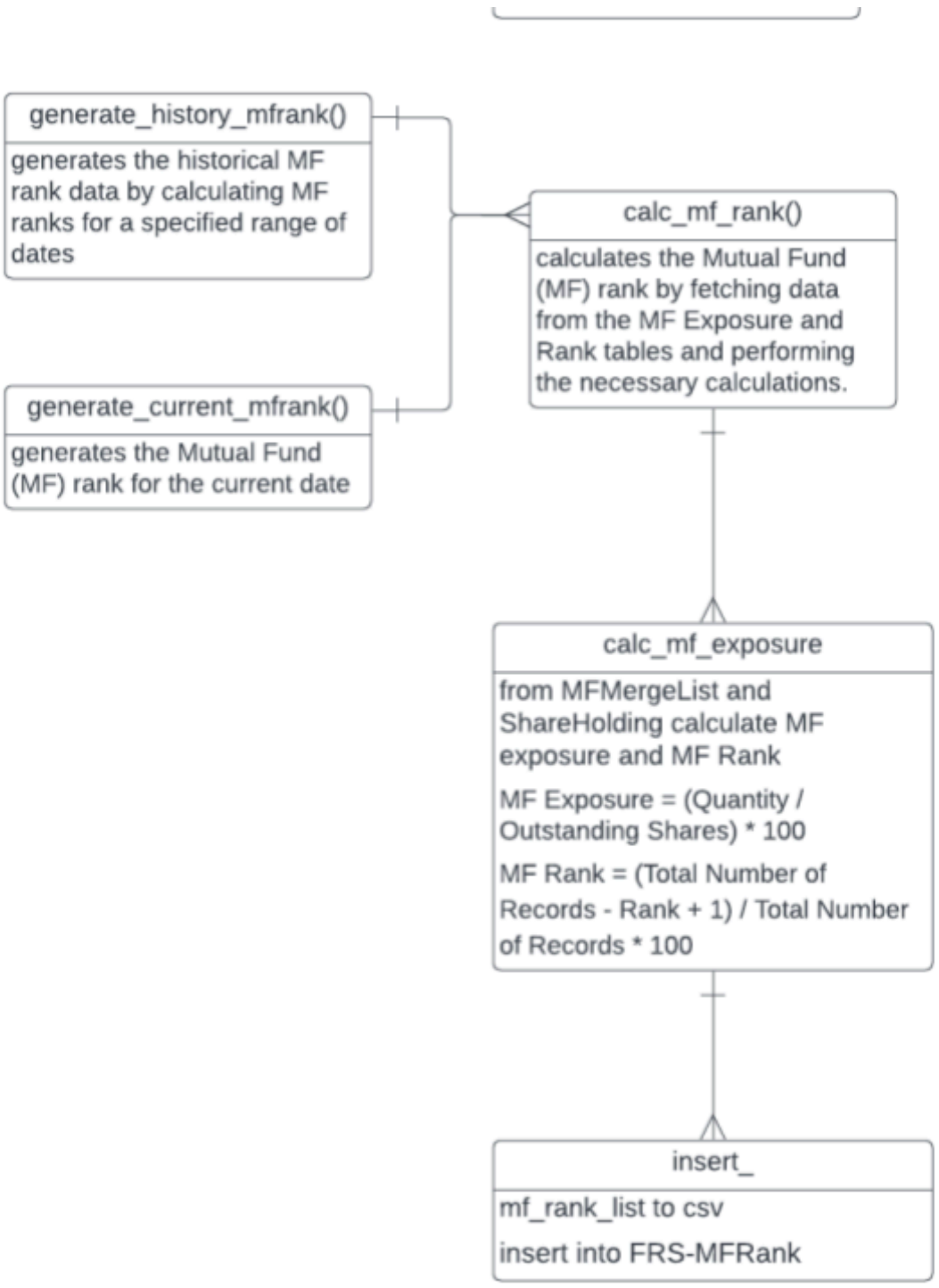
Description:
Provide a detailed step-by-step explanation of what the function does. Describe each step and its purpose in the function. You can include any formulas used, and provide descriptions for them.

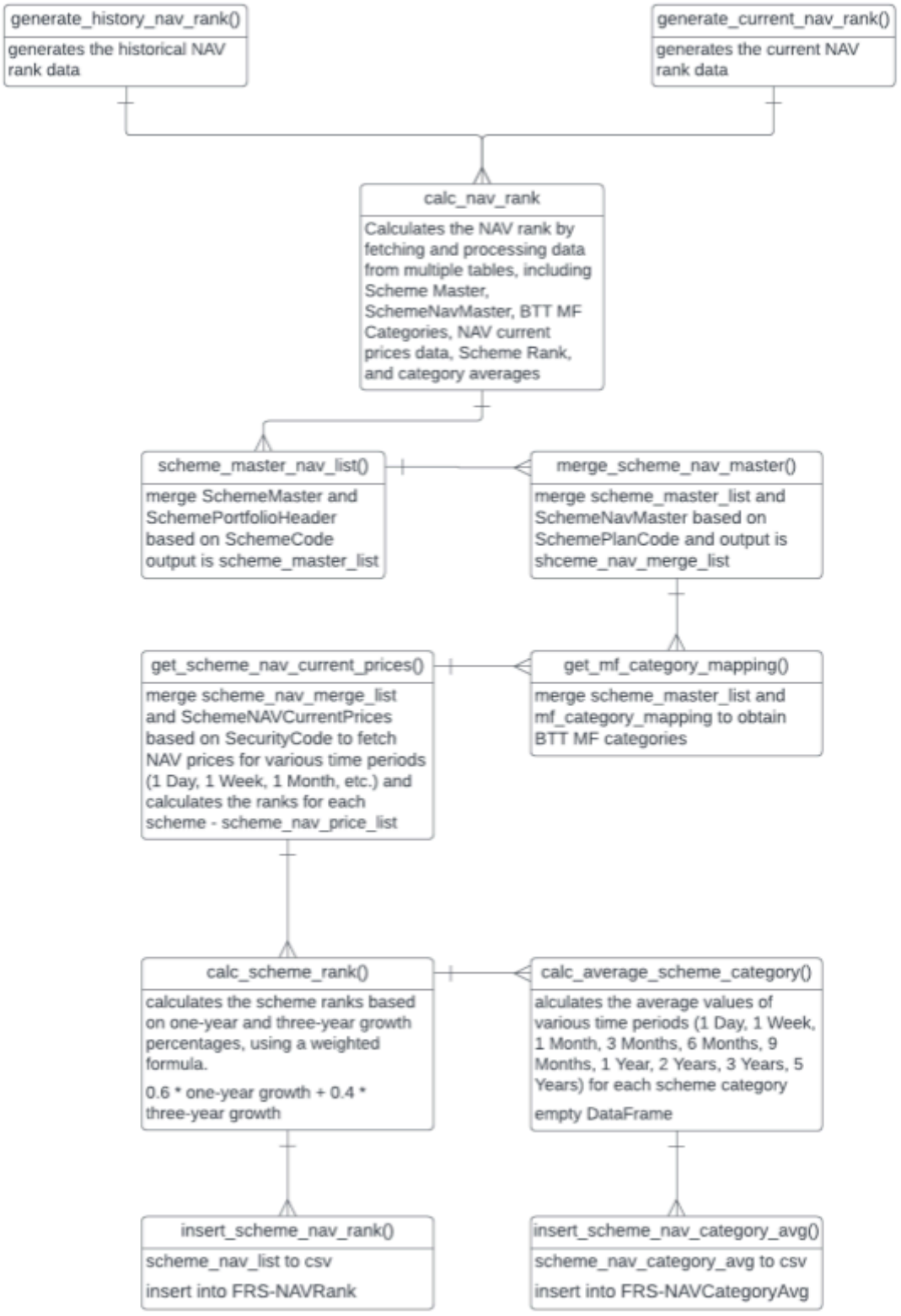
1. Step 1: Describe the first step.
    - Formula 1 (if used): Description of the formula.
    - ...
2. Step 2: Describe the second step.
    - Formula 2 (if used): Description of the formula.
    - ...

Return:
(type): Description of the return value.

FRS


generate_history_mflist()
generates the historical MF list by compiling MF data for a specified range of dates
generate_current_mflist()
generates the Mutual Fund (MF) list for the current date
compile_mf_list()
Compiles the data of Mutual Fund (MF) List by fetching data from various tables and performing necessary operations
get_scheme_master_list()
merge SchemeMaster and SchemePortfolioHeader based on SchemeCode output is scheme_mf_list
merge_schememaster_portfolio
merge scheme_mf_list and SchemeWiseportfolio based on SchemeCode
map_market_captitalisation()
based on scheme_mf_list get market cap and market cap value from PE and calculate market cap rank
Market Cap Rank = Market Cap Value * 100
merge_background_industry_info()
merge scheme_mf_list and InudstryMapping based on IndustryCode
insert_mutual_fund_list()
scheme_mf_list to csv
insert into MFMergeList


generate_history_mfrank()
generates the historical MF rank data by calculating MF ranks for a specified range of dates
generate_current_mfrank()
generates the Mutual Fund (MF) rank for the current date
calc_mf_rank()
calculates the Mutual Fund (MF) rank by fetching data from the MF Exposure and Rank tables and performing the necessary calculations.
calc_mf_exposure
from MFMergeList and ShareHolding calculate MF exposure and MF Rank
MF Exposure = (Quantity / Outstanding Shares) * 100
MF Rank = (Total Number of Records - Rank + 1) / Total Number of Records * 100
insert_
mf_rank_list to csv
insert into FRS-MFRank



get_last_date_prev_month

Parameters:

- `self` (object): The instance of the class.
- `target` (datetime.date): The target date for which you want to find the last day of the previous month.

Description:

This function calculates and returns the last day of the previous month based on the given target date.

Return:

(datetime.date): The last day of the previous month as a `datetime.date` object.

get_scheme_master_list

Parameters:

- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `date` (datetime.date): The reference date for the query.

Description:

This function retrieves a list of Scheme data from the "SchemeMaster" and "SchemePortfolioHeader" tables, specifically for SchemePlanCode 2066. It applies various filter conditions to select specific records from the tables, and then it merges the data from both tables based on the "SchemeCode" using a left join operation. The function also calculates and prints the length of the merged dataset.

Return:

(pd.DataFrame): Merged data of SchemeMaster and SchemePortfolioHeader.

Operation:

1. Construct SQL queries to retrieve data from the "SchemeMaster" and "SchemePortfolioHeader" tables with specific conditions.
2. Execute the SQL queries using the provided database connection (`conn`).
3. Merge the data from the "SchemeMaster" and "SchemePortfolioHeader" tables based on the "SchemeCode" using a left join.
4. Calculate and print the length of the merged dataset.

merge_schememaster_schemeportfolio

Parameters:

- `self` (object): The instance of the class.
- `scheme_mf_list` (pd.DataFrame): The merged data of SchemeMaster and SchemePortfolioHeader.
- `conn` (connection object): The connection to the database.
- `date` (datetime.date): The reference date for the query.

Description:

This function is responsible for fetching and merging data from the "SchemeMaster" and "SchemeWisePortfolio" tables in a database. It specifically targets records with a SchemePlanCode of 2066 and applies various filter conditions to select specific records from the "SchemeWisePortfolio" table. The function then calculates the length of the resulting data.

Return:

(pd.DataFrame): Merged data of scheme_mf_list and SchemeWisePortfolio.

Operation:

1. Calculate various `holding_date` values by calling the `get_last_date_prev_month` function multiple times.
2. Construct an SQL query to retrieve data from the "SchemewisePortfolio" table with specific conditions, including "SchemePlanCode," "InstrumentName," and date criteria.
3. Execute the SQL query using the provided database connection (`conn`).
4. Merge the data from the "scheme_mf_list" and "scheme_wise_portfolio_list" based on the "SchemeCode" using a left join.
5. Calculate and print the length of the merged dataset.

map_market_capitalisation

Parameters:

- `self` (object): The instance of the class.
- `scheme_mf_list` (pd.DataFrame): The merged data of SchemeMaster and IndustryMapping.
- `conn` (connection object): The connection to the database.
- `date` (datetime.date): The reference date for the query.

Description:

This function maps market capitalization values for the given data by fetching relevant information from the "PE" table and calculating "Market Cap Rank" based on "Market Cap Value."

Return:

(pd.DataFrame): The input `scheme_mf_list` with additional columns for "Market Cap Value," "Market Cap," and "Market Cap Rank."

Operation:

1. Construct an SQL query to retrieve data from the "PE" table with specific conditions, including "GenDate" date criteria.
2. Execute the SQL query using the provided database connection (`conn`) to obtain the "PE" data.
3. Iterate over each row in the input `scheme_mf_list` to map market capitalization data.

- Fetch "Market Cap Value" and "Market Cap Class" from the "PE" data for the corresponding "InvestedCompanyCode."
- Assign the values to the respective columns in `scheme_mf_list`.
4. Calculate "Market Cap Rank" by ranking "Market Cap Value" as a percentage and scaling by 100.
5. Sort `scheme_mf_list` based on "Market Cap Value" in descending order.
6. Return the updated `scheme_mf_list` with market capitalization information.

insert_mutualfund_list

Parameters:
- `self` (object): The instance of the class.
- `scheme_mf_list` (pd.DataFrame): data of market capitalization.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The cursor for database operations.
- `date` (datetime.date): The reference date for the data.

Description:
This function is responsible for exporting the data in `scheme_mf_list` to a CSV file and then inserting the data into the "MFMergeList" table in the database. It also performs data preprocessing and column renaming.

Operation:
1. Add a "GenDate" column to `scheme_mf_list` to store the provided date.
2. Rename columns in `scheme_mf_list` for consistency.
3. Remove redundant columns from `scheme_mf_list`.
4. Fill missing values and convert the "SchemePlanCode" column to integers.
5. Convert numeric columns to the appropriate data types.
6. Select a subset of columns for the final dataset.
7. Export the data to a CSV file named "schememerge_export.csv."
8. Prepare a SQL COPY statement to load data from the CSV file into the "MFMergeList" table.
9. Execute the COPY operation to import the data into the database.
10. Commit the transaction and remove the temporary CSV file.

Return:
None.

calc_mf_exposure

Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.

- `date` (datetime.date): The reference date for the query.

Description:
This function calculates the Mutual Fund (MF) exposure and MF Rank for a given date by fetching data from the "MFMergeList" and "ShareHolding" tables and applying a specific formula.

Operation:
1. Construct SQL queries to retrieve data from "MFMergeList" and "ShareHolding" tables based on the provided date.
2. Execute the SQL queries using the provided database connection (`conn`) to obtain relevant data.
3. Perform data preprocessing on the obtained data by converting numeric columns to the appropriate data types.
4. Calculate MF exposure for each unique "InvestedCompanyCode" by dividing the "Quantity" by the "Total" outstanding shares. The formula for MF Exposure is:
MF Exposure = (Quantity / Outstanding Shares) * 100
5. Calculate MF Rank based on MF Exposure values, where a higher MF Exposure results in a lower rank. The formula for MF Rank is calculated as a percentile rank:
MF Rank = (Total Number of Records - Rank + 1) / Total Number of Records * 100
6. Return the resulting dataset with MF Rank values.

Return:
(pd.DataFrame): The dataset containing MF exposure and MF Rank values.

Explanation for Formula Attributes:
- `MF Exposure`: The exposure of a Mutual Fund to a specific company's shares, calculated as the ratio of the quantity of shares held by the Mutual Fund to the total outstanding shares of the company. It is expressed as a percentage, providing insight into the fund's stake in the company.
- `MF Rank`: A ranking metric that assesses the relative MF Exposure of each company in comparison to others. A higher MF Exposure results in a lower rank. The formula calculates the percentile rank for each company based on its MF Exposure, indicating the company's position within the dataset.
    - shareholding_list: A Pandas DataFrame containing the shareholding data, including the company code and the total number of shares held.
    - stock_qty_list: A Pandas DataFrame containing the quantity of shares held for each invested company.
    - outstanding_shares_list: A Pandas DataFrame containing the outstanding shares for each company code.
    - quantity: The total quantity of shares held in the invested company.
    - outstanding_shares: The total outstanding shares for the invested company.

- mf_exposure: The MF exposure, calculated as (quantity / outstanding_shares) * 100.
- mf_rank_unique_list: A Pandas DataFrame containing the MF exposure and rank for each invested company.
- **Formula:**
- mf_exposure = (quantity / outstanding_shares) * 100
- This formula calculates the MF exposure as a percentage of the outstanding shares. The higher the MF exposure,
- the more concentrated the portfolio is in the invested company.

insert_mf_rank

Parameters:
- `self` (object): The instance of the class.
- `mf_rank_list` (pd.DataFrame): Data containing calculated MF Rank values.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The cursor for database operations.
- `date` (datetime.date): The reference date for the data.

Description:
This function is responsible for exporting the data in `mf_rank_list` to a CSV file and then inserting the data into the "FRS-MFRank" table in the database.

Operation:
1. Add a "Date" column to `mf_rank_list` to store the provided date.
2. Select a subset of columns from `mf_rank_list` for the final dataset.
3. Export the data to a CSV file named "mf_rank_export.csv."
4. Prepare a SQL COPY statement to load data from the CSV file into the "FRS-MFRank" table.
5. Execute the COPY operation to import the data into the database.
6. Commit the transaction and remove the temporary CSV file.

Return:
None.

scheme_master_nav_list

Parameters:

- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `date` (datetime.date): The reference date for the data.

Description:
This function fetches data from the "SchemeMaster" and "SchemePortfolioHeader" tables for a specific SchemePlanCode (2066) and merges the data based on the SchemeCode.

Operation:
1. Execute a SQL query (`scheme_sql`) to retrieve data from the "SchemeMaster" table. The query filters for SchemePlanCode 2066, SchemeTypeDescription "Open Ended," and SchemeName matching "!Direct & !Institutional" using a full-text search.
2. Execute another SQL query (`scheme_aum_sql`) to retrieve data from the "SchemePortfolioHeader" table. The query filters for SchemePlanCode 2066 and HoldingDate less than or equal to the provided `date`, ordering the results by SchemeCode and HoldingDate in descending order.
3. Merge the data from the two queries based on the "SchemeCode" column using a left join, resulting in a DataFrame (`scheme_master_list`).
4. Rename the "TotalMarketValue" column to "AUM" in the merged DataFrame.

Return:
A DataFrame containing the merged data of "SchemeMaster" and "SchemePortfolioHeader."

merge_scheme_nav_master
Parameters:
- `self` (object): The instance of the class.
- `scheme_master_list` (DataFrame): Data merged from SchemeMaster and SchemePortfolioHeader.
- `conn` (connection object): The connection to the database.

Description:
This function fetches data from the "SchemeNAVMaster" table for a specific SchemePlanCode (2066) and merges it with the provided `scheme_master_list`.

Operation:
1. Execute a SQL query (`scheme_nav_master_sql`) to retrieve data from the "SchemeNAVMaster" table, filtering for SchemePlanCode 2066.
2. Read the query result into a DataFrame (`scheme_nav_master_list`) using the provided database connection.
3. Merge the `scheme_master_list` and `scheme_nav_master_list` DataFrames based on the "SchemeCode" column using a left join, resulting in `scheme_nav_merge_list`.

4. Select specific columns from `scheme_nav_merge_list` and rename some of the columns to match the desired output.

Return:
A DataFrame containing merged data from SchemeMaster, SchemePortfolioHeader, and SchemeNAVMaster.

get_mf_category_mapping
Parameters:
- `self` (object): The instance of the class.
- `scheme_nav_merge_list` (DataFrame): Data merged from SchemeMaster, SchemePortfolioHeader, and SchemeNAVMaster.
- `conn` (connection object): The connection to the database.

Description:
This function retrieves data from the "mf_category_mapping" table and merges it with the provided `scheme_nav_merge_list` to obtain BTT MF Categories for schemes.

Operation:
1. Execute an SQL query (`sql`) to fetch data from the "mf_category_mapping" table, including columns "scheme_code," "btt_scheme_code," and "btt_scheme_category."
2. Read the query result into a DataFrame (`btt_mf_category`) using the provided database connection.
3. Merge the `btt_mf_category` and `scheme_nav_merge_list` DataFrames based on the "scheme_code" and "SchemeCode" columns using a left join, resulting in `scheme_nav_merge_list`.
4. Remove rows with null values in the "btt_scheme_category" column.
5. Drop the "scheme_code" column from the DataFrame.

Return:
A DataFrame containing merged data from "mf_category_mapping" and `scheme_nav_merge_list` with BTT MF Categories for schemes.

This code defines the `get_mf_category_mapping` function, which retrieves BTT MF Category mapping data from a database table and merges it with the previously merged scheme data.

get_scheme_nav_current_prices

Parameters:

- `self` (object): The instance of the class.
- `scheme_nav_merge_list` (DataFrame): Merged data of scheme_nav_merge_list, and SchemeNAVMaster .
- `conn` (connection object): The connection to the database.
- `date` (date): The date for which NAV prices are needed.

Description:

This function fetches the scheme NAV current prices for various time periods (1 Day, 1 Week, 1 Month, etc.) and calculates the ranks for each scheme.

Operation:

1. Calculate a back date (3 days earlier than the given date).
2. Execute an SQL query to fetch distinct scheme NAV prices based on the "SecurityCode" for the specified back date.
3. Merge the fetched NAV data with `scheme_nav_merge_list` based on the "SecurityCode" using a left join.
4. Rename columns of the resulting DataFrame for clarity.
5. Convert percentage-related columns to float data types.
6. Remove spaces from the "SchemeCategoryDescription" column.
7. Calculate and add ranks (1 Day Rank, 1 Week Rank, etc.) based on various percentage change values, grouped by the BTT scheme category.
8. Select and reorder columns of interest to create the final DataFrame.

Return:

A DataFrame containing scheme NAV data, including percentage change and ranks for various time periods, for each scheme.

This code defines the `get_scheme_nav_current_prices` function, which calculates NAV ranks and fetches scheme NAV prices for different time periods.

calc_scheme_rank

Parameters:

- `self` (object): The instance of the class.
- `scheme_nav_list` (DataFrame): Data list of scheme NAV ranks.
- `conn` (connection object): The connection to the database.
- `date` (date): The date for which scheme NAV ranks are being calculated.

Description:
This function calculates the scheme ranks based on one-year and three-year growth percentages, using a weighted formula.

Operation:
1. For each row in the `scheme_nav_list` DataFrame, calculate the one-year and three-year growth percentages.
2. Handle cases where there are multiple values or no values for these growth percentages.
3. Calculate the "Scheme Rank Value" using a weighted formula (0.6 * one-year growth + 0.4 * three-year growth).
4. Calculate the "Scheme Rank" for each scheme based on their "Scheme Rank Value" within their scheme category.
5. Store the "Scheme Rank" as a percentage (0 to 100).

Return:
A DataFrame containing scheme NAV ranks, where "Scheme Rank" represents the calculated scheme ranks as percentages within their respective scheme categories.

This code defines the `calc_scheme_rank` function, which calculates scheme ranks based on a weighted formula involving one-year and three-year growth percentages and stores the results in a DataFrame. It handles various cases where there are multiple values for these growth percentages or no values, ensuring robustness in the calculation.

insert_scheme_nav_rank
Parameters:
- `self` (object): The instance of the class.
- `scheme_nav_list` (DataFrame): Data of NAV ranks to be inserted.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor.
- `date` (date): The date for which NAV ranks are being inserted.

Description:
This function inserts NAV rank data into the database.

Operation:
1. Clean and convert various columns of the `scheme_nav_list` DataFrame to the appropriate data types.
2. Add a "Date" column with the provided date in the "YYYY-MM-DD" format.
3. Rename columns to match the database schema.
4. Create a CSV file named "scheme_nav_rank_export.csv" and export the data.
5. Insert the exported data into the "FRS-NAVRank" table in the database.
6. Remove the temporary CSV file.

Return:
A DataFrame containing the inserted NAV rank data.

This code defines the `insert_scheme_nav_rank` function, which prepares and inserts NAV rank data into a database. It ensures that the data is properly formatted, matches the database schema, and is subsequently inserted into the designated table.

calc_average_scheme_category

Parameters:
- `self` (object): The instance of the class.
- `scheme_nav_list` (DataFrame): Data of scheme NAV values and ranks.
- `conn` (connection object): The connection to the database.

Description:
This function calculates the average values of various time periods (1 Day, 1 Week, 1 Month, 3 Months, 6 Months, 9 Months, 1 Year, 2 Years, 3 Years, 5 Years) for each scheme category.

Operation:
1. Create an empty DataFrame `scheme_nav_avg_list` to store the average values.
2. Calculate and populate the average values for each time period by grouping data based on the scheme category.
3. Reset the index of the `scheme_nav_avg_list` DataFrame to ensure proper formatting.
4. Return the DataFrame with the calculated average values.

Return:
A DataFrame containing the average values for each scheme category and time period.

This code defines the `calc_average_scheme_category` function, which calculates the average values of different time periods for each scheme category based on the provided scheme NAV data. The results are stored in a new DataFrame and returned for further analysis or use.

insert_scheme_nav_category_avg

Parameters:

- `self` (object): The instance of the class.
- `scheme_nav_category_avg` (DataFrame): Data of NAV category averages.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.
- `date` (datetime object): The date for which the data is inserted.

Description:
This function inserts the calculated average values for NAV categories into the database.

Operation:
1. Add a 'Date' column to the DataFrame `scheme_nav_category_avg` with the provided date.
2. Select specific columns and remove rows with any missing values (NaN).
3. Export the data to a CSV file named "scheme_nav_category_avg_export.csv."
4. Execute a SQL COPY statement to insert the data from the CSV file into the "FRS-NAVCategoryAvg" table.
5. Commit the transaction to save the changes to the database.
6. Remove the temporary CSV file used for data export.

Return:
None

This code defines the `insert_scheme_nav_category_avg` function, which takes the calculated NAV category average values, adds the date, and inserts the data into the database. It performs data export and uses a COPY statement to efficiently insert data into the target database table.

compile_mf_list
Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.
- `date` (datetime object): The date for which the MF list is compiled.

Description:
This function compiles the data of Mutual Fund (MF) List by fetching data from various tables and performing necessary operations.

Operation:

1. Set the global variable `today` to the provided date for reference within the function.
2. Print a message indicating that schemes are being filtered from the "Scheme Master" table.
3. Call the `get_scheme_master_list` function to obtain data from the "Scheme Master" table for the given date and store it in the `scheme_mf_list` DataFrame.
4. Check if the `scheme_mf_list` is not empty (i.e., data is available).
5. If data is available, perform the following operations:
a. Merge the filtered scheme list with scheme-wise portfolio data using the `merge_schememaster_schemeportfolio` function.
b. Merge the resulting data with industry mapping information using the `merge_background_industry_info` function.
c. Calculate market capitalization by mapping it to the scheme list using the `map_market_capitalisation` function.
d. Insert the final MF list data into the database using the `insert_mutualfund_list` function.
6. If the `scheme_mf_list` is empty, print a message indicating that data is not present for filtering schemes from the "Scheme Master" for the given date.

Return:
None

This code defines the `compile_mf_list` function, which compiles data for the Mutual Fund (MF) list by fetching, merging, and mapping data from various tables, and finally inserting the compiled data into the database. The function provides informative print messages to track the progress and handle scenarios where data might not be available for a specific date.

calc_mf_rank

Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.
- `date` (datetime object): The date for which the MF rank is calculated.

Description:
This function calculates the Mutual Fund (MF) rank by fetching data from the MF Exposure and Rank tables and performing the necessary calculations.

Operation:
1. Set the global variable `today` to the provided date for reference within the function.

2. Print a message indicating that MF exposure and rank are being calculated.
3. Call the `calc_mf_exposure` function to obtain data for MF exposure.
4. Calculate the MF rank based on the fetched data.
5. Print a message indicating that the calculated data is being inserted into the database.

generate_history_mflist

Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.

Description:
This function generates the historical MF list by compiling MF data for a specified range of dates.

Operation:
1. Define the start and end dates for the historical data range.
2. Iterate through the range of dates from the start date to the end date.
3. For each date, call the `compile_mf_list` function to compile the MF list data and insert it into the database.
4. Print messages to indicate the progress.

generate_history_mfrank

Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.

Description:
This function generates the historical MF rank data by calculating MF ranks for a specified range of dates.

Operation:
1. Define the start and end dates for the historical data range.
2. Iterate through the range of dates from the start date to the end date.
3. For each date, call the `calc_mf_rank` function to calculate the MF rank and insert it into the database.
4. Print messages to indicate the progress.

These functions are part of a broader process to calculate and maintain historical data for Mutual Fund (MF) lists and ranks. The code manages historical data for MFs, calculating ranks and maintaining a record of these calculations over time.

generate_current_mflist

Parameters:

- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.
- `curr_date` (datetime object): The current date for which the MF list is generated.

Description:

This function generates the Mutual Fund (MF) list for the current date by calling the `compile_mf_list` function.

Operation:

1. Print a message indicating the generation of the MF list for the provided current date.
2. Call the `compile_mf_list` function to compile the MF list for the current date.
3. Print a message indicating the completion of the compilation.

generate_current_mfrank

Parameters:

- `self` (object): The instance of the class.
- `curr_date` (datetime object): The current date for which the MF rank is generated.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.

Description:

This function generates the Mutual Fund (MF) rank for the current date by fetching data and calling relevant functions.

Operation:

1. Call the `generate_current_mflist` function to generate the MF list for the current date.
2. Call the `calc_mf_rank` function to calculate the MF rank for the current date.
3. Print a message indicating the completion of the MF rank generation.

export_table

Parameters:

- `self` (object): The instance of the class.
- `name` (str): The name of the table.
- `table` (DataFrame): Data to be exported.

Description:
This function exports a provided DataFrame into a CSV file with a specified name.

calc_nav_rank
Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.
- `date` (datetime object): The date for which NAV ranks are calculated.

Description:
This function calculates the NAV rank by fetching and processing data from multiple tables, including Scheme Master, SchemeNavMaster, BTT MF Categories, NAV current prices data, Scheme Rank, and category averages.

Operation:
1. Set the global variable `today` to the provided date for reference within the function.
2. Perform a series of operations to calculate NAV ranks, including filtering schemes, merging data, calculating ranks, and inserting results into tables.
3. Print progress messages for each step of the process.

generate_history_nav_rank
Parameters:
- `self` (object): The instance of the class.
- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.

Description:
This function generates the historical NAV rank data by calling the `calc_nav_rank` function for a specified range of dates.

Operation:
1. Define the start and end dates for the historical data range.
2. Iterate through the range of dates from the start date to the end date.
3. For each date, call the `calc_nav_rank` function to calculate the NAV ranks and category averages.
4. Print messages to indicate the progress.

generate_current_nav_rank
Parameters:
- `self` (object): The instance of the class.
- `curr_date` (datetime object): The current date for which NAV ranks are generated.

- `conn` (connection object): The connection to the database.
- `cur` (cursor object): The database cursor for executing SQL queries.

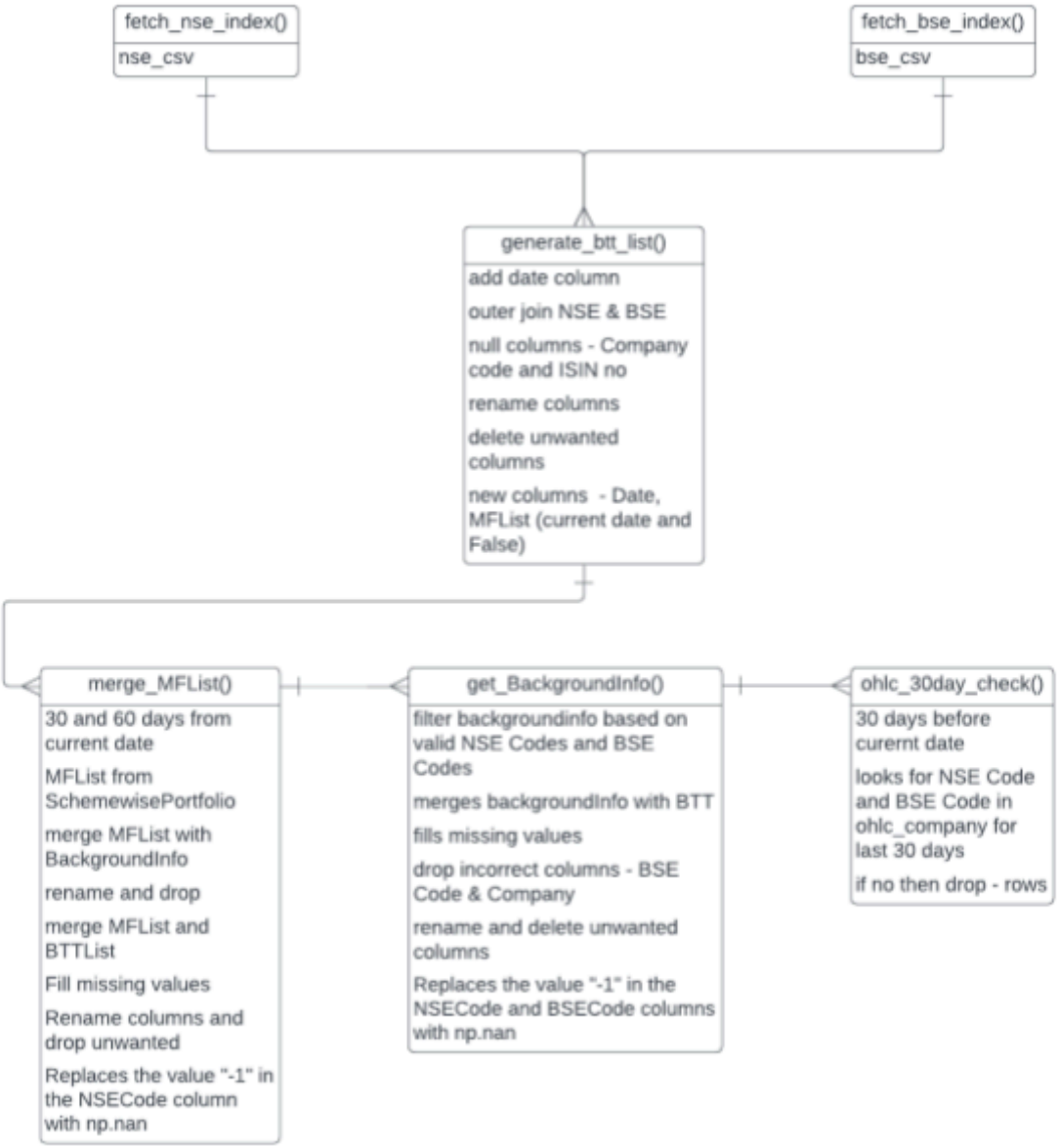
Description:
This function generates the current NAV rank data by calling the `calc_nav_rank` function for the provided current date.

Operation:
1. Print a message indicating the generation of NAV ranks for the provided current date.
2. Call the `calc_nav_rank` function to calculate the NAV ranks and category averages for the current date.
3. Print a message indicating the completion of the process.

These functions are part of a broader process to calculate and manage Mutual Fund (MF) lists and NAV ranks, which are essential for financial analysis and reporting.

BTT LIST


fetch_nse_index()
fetch_bse_index()
generate_btt_list()
merge_MFList()
ohlc_30day_check()
get_BackgorundInfo()
BTT_LIST



fetch_nse_index
Parameters:

- None

Description:

This function is responsible for fetching NSE (National Stock Exchange) index data from a specified URL, storing it in a local file, and reading it into a DataFrame.

Operation:

1. Define the `my_path` variable to store the absolute path of the script's directory.
2. Define the `filepath` variable to specify the directory where the NSE index file will be stored.

3. Build the full file path by joining `my_path` and "index-files/".
4. Create a dictionary `headers` containing necessary headers for accessing the NSE website.
5. Create a dictionary `cookies` with some cookie data.
6. Define the URL of the NSE index file as `file_url` and the file name as `file_name`.
7. Send an HTTP GET request to the `file_url` with the specified headers and cookies.
8. If the response status code is 200 (OK), download the file in chunks and store it in a local file at `path_to_store_file`.
9. Close the local file after downloading the entire content.
10. If the file download is successful, create the full file path `csv_file` by joining `filepath` and `file_name`.
11. Check if the file exists. If it does, read its content into a Pandas DataFrame named `table_nse`.
12. If the file doesn't exist, print an error message indicating that the file couldn't be fetched.
13. Return the `table_nse` DataFrame.

Return:
- table_nse (DataFrame): The NSE index data stored in a Pandas DataFrame.

Function Name: fetch_bse_index

Description:
This function is designed to fetch the BSE (Bombay Stock Exchange) index data from the BSE website. It performs a series of HTTP requests to access the required data. The BSE index data is downloaded and stored in a CSV file, which is then read into a Pandas DataFrame for further processing.

Parameters: None

Returns:
- table_bse: A Pandas DataFrame containing the BSE index data.

Code Details:
1. The function begins by printing a message to indicate that the BSE Index Fetch has been invoked.

2. It defines various request parameters, such as cookies, headers, params, and data, which are necessary to access the BSE website. These parameters simulate a web request that invokes a JavaScript function to download the BSE index data.

3. It specifies the URL for the BSE index data and sends an HTTP POST request to retrieve the data. The response is stored in the 'req' variable.

4. The code checks if the response status code is 200 (indicating success). If successful, it proceeds to download the data and save it as a CSV file.

5. The data is downloaded in chunks (100,000 bytes per chunk) and written to the CSV file. The file is stored in the specified file path with the name "BSE500_Index.csv."

6. If the response status code is not 200, the code prints an error message indicating that the file couldn't be fetched.

7. Finally, the code checks if the CSV file exists. If it does, the data is read into a Pandas DataFrame named 'table_bse.' If the file doesn't exist, an error message is printed.

8. The function returns the 'table_bse' DataFrame containing the BSE index data.

Note: The function includes code for downloading the data via an HTTP request, but it is currently commented out. This may be for testing purposes. The function is expected to be used with the uncommented code for fetching the data.

Function Name: generate_btt_list(table_nse, table_bse, conn, cur, curr_date)

Parameters:
- table_nse (pd.DataFrame): DataFrame containing NSE data.
- table_bse (pd.DataFrame): DataFrame containing BSE data.
- conn (connection object): Connection to the database.
- cur (cursor object): Cursor for the database.
- curr_date (datetime.date): The current date.

Description:
This function generates a list by merging the data from the NSE and BSE tables. It performs several operations such as filling missing values, renaming columns, deleting unwanted rows, and adding new columns like 'Date' and 'MFList' (with a default value of False). The function also includes a placeholder for populating the 'MFList' column by inferring data from an unspecified source. Additionally, it drops unwanted columns, ensuring that the final DataFrame consists of specific columns, including 'Company', 'ISIN', 'NSECode', 'BSECode', 'Date', and 'MFList'.

Operation:
1. Format the current date as a string in the 'YYYY-MM-DD' format.

2. Merge the data from the NSE and BSE tables using an outer join and store it in 'table_btt'.
3. Print the count of rows in 'table_nse' and 'table_bse'.
4. Export the 'NSE' and 'BSE' tables using an unspecified function ('export_table').
5. Print the count of rows in 'table_btt' after the merge.
6. Fill missing values in the 'Company Name' and 'ISIN Code' columns with values from 'COMPANY' and 'ISIN No.' columns, respectively.
7. Rename specific columns to ensure consistency and clarity in the dataset.
8. Identify the columns to be removed and drop them from the DataFrame.
9. Add columns 'Date' and 'MFList' with default values of the current date and 'False', respectively.
10. Implement a placeholder ('TODO') for populating the 'MFList' column from an unspecified source (FB).
11. Drop unwanted columns to maintain a specific set of columns in the final DataFrame.

Return:
(pd.DataFrame): The modified DataFrame 'table_btt' containing the required columns, with missing values filled and unwanted columns removed.

"""
Function Name: ohlc_30day_check

Description:
This function checks for OHLC (Open, High, Low, Close) data for each company in the provided 'table_btt' within a 30-day time frame prior to the 'curr_date'. It accesses the 'OHLC' data from the 'public' schema in the specified database connection 'conn'. Rows without relevant OHLC data are dropped from 'table_btt'.

Parameters:
- table_btt (DataFrame): A table containing company data
- conn (Connection): A database connection object
- curr_date (datetime): The current date for the analysis

Returns:
Modified 'table_btt' after dropping rows without relevant OHLC data within the specified time frame.

Function Description:
1. Converts the 'curr_date' to string format for use in SQL queries.
2. Calculates the 'month_back' date as 30 days before the 'curr_date'.
3. Constructs an SQL query to retrieve relevant 'OHLC' data within the specified date range.

4. Reads the SQL query results into the 'ohlc_company' DataFrame.
5. Iterates through each row in 'table_btt' and checks for corresponding 'NSECode' and 'BSECode' values.
6. If no relevant data is found in 'ohlc_company' for a company, the corresponding row in 'table_btt' is dropped.
7. Prints the count of dropped rows.
8. Returns the modified 'table_btt'.
"""

"""
Function Name: get_backgroundtable

Description:
This function retrieves data from the 'public."BackgroundInfo"' table in the specified database connection 'conn'. It also performs data cleaning by replacing missing or undesirable values in the 'NSECode' and 'BSECode' columns with standard values, and finally returns the cleaned 'background_info' DataFrame.

Parameters:
- conn (Connection): A database connection object

Returns:
A cleaned 'background_info' DataFrame containing company background information.

Function Description:
1. Constructs an SQL query 'background_info_sql' to select all data from the 'public."BackgroundInfo"' table.
2. Executes the SQL query using 'pd.read_sql_query' to fetch the data and store it in the 'background_info' DataFrame.
3. Performs data cleaning:
  - Replaces missing (NaN), empty (" "), and blank ("") values in the 'NSECode' column with "-1".
  - Replaces missing (NaN), zero (0), and negative zero values in the 'BSECode' column with "-1".
4. Returns the cleaned 'background_info' DataFrame.
"""

"""
Function Name: merge_MFList

Description:
This function merges mutual fund (MF) data with a given DataFrame 'BTTList' based on matching ISIN codes. It retrieves MF data from a database, processes it, and merges it with the input DataFrame. The merged DataFrame is returned.

Parameters:
- BTTList (DataFrame): A DataFrame containing data to be merged with MF data.
- conn (Connection): A database connection object.
- curr_date (datetime): The current date used for date calculations.

Returns:
A merged DataFrame containing combined data from 'BTTList' and MF data.

Function Description:
1. Calculate the 'month_back' and 'last_mfholding' dates based on the 'curr_date', which represents the current date minus 30 and 60 days, respectively.
2. Print the 'month_back' and 'last_mfholding' dates for informational purposes.
3. Construct an SQL query 'MFList_sql' to retrieve MF data. It selects distinct entries based on the "InvestedCompanyCode" and "HoldingDate" from the 'public."SchemewisePortfolio"' table where specific conditions are met, such as 'HoldingDate' being within the 'last_mfholding' and 'month_back' range and 'InvestedCompanyCode' being positive and not null.
4. Execute the SQL query using 'pd.read_sql_query' to fetch the MF data and store it in the 'MFList' DataFrame. Print the total row count in 'MFList' for reference.
5. Call the 'get_backgroundtable' function to retrieve company background information and store it in the 'background_info' DataFrame.
6. Filter 'background_info' by removing rows where 'CompanyCode' is not NaN. Print the total row count in 'background_info' for reference.
7. Merge 'MFList' with 'background_info' based on the 'InvestedCompanyCode' and 'CompanyCode' columns, using a left join. The resulting DataFrame is updated to include information about 'CompanyName', 'BSECode', 'NSECode', 'CompanyCode', and 'ISINCode'.
8. Add a new column 'MFList' to the 'MFList' DataFrame and set all its values to 'True'.
9. Drop the 'CompanyCode' and 'InvestedCompanyCode' columns from the 'MFList' DataFrame and rename the remaining columns as 'Company', 'Date', 'ISIN', and 'MFList'.
10. Filter rows in 'MFList' where the 'ISIN' column is not null.
11. Print the total row count in 'MFList' after merging with 'background_info'.
12. Merge the 'BTTList' DataFrame with 'MFList' based on the 'ISIN' column using an outer join. The resulting DataFrame is updated to include additional columns: 'Company_x', 'Date_x', 'NSECode_x', 'BSECode_x', and 'MFList_x'.
13. Fill missing values in 'BTTList' columns with corresponding values from 'Company_y', 'Date_y', 'NSECode_y', 'BSECode_y', and 'MFList_y'.
14. Rename columns in 'BTTList' as 'Company', 'Date', 'NSECode', 'BSECode', and 'MFList'.
15. Define a list 'table_columns' containing the expected column names in the final DataFrame.
16. Get the list of columns in the 'BTTList' DataFrame and store it in 'csv_columns'.

17. Identify columns to remove by finding the difference between 'csv_columns' and 'table_columns', then remove these columns from 'BTTList'.
18. Reorder the columns in 'BTTList' to match the order in 'table_columns'.
19. Replace values in the 'NSECode' column with NaN for rows where 'NSECode' is "-1".
20. Return the merged and cleaned 'BTTList' DataFrame.

Note: This function effectively combines mutual fund data with 'BTTList', performs data cleaning, and ensures the resulting DataFrame matches the expected format.
"""

"""
Function Name: get_BackgroundInfo

Description:
This function enhances a given DataFrame 'BTTList' with additional background information by merging it with data from the 'background_info' table in a database. The function first filters 'background_info' data for NSE and BSE codes, performs merges, and cleans the merged DataFrame. The resulting DataFrame is returned.

Parameters:
- BTTList (DataFrame): A DataFrame containing data to be enhanced with background information.
- conn (Connection): A database connection object.

Returns:
A DataFrame with enhanced data, including background information from the database.

Function Description:
1. Call the 'get_backgroundtable' function to retrieve background information from the database and store it in the 'background_info' DataFrame.
2. Create 'background_info_nse' by filtering 'background_info' to include only rows where 'NSECode' is not NaN, not an empty string (" "), not an empty string (""), and not equal to "-1".
3. Merge 'BTTList' with 'background_info_nse' based on the 'NSECode' column using a left join. The resulting DataFrame is updated to include 'CompanyName', 'BSECode', 'NSECode', and 'CompanyCode'.
4. Fill missing values in 'BTTList' columns 'CompanyName' with values from 'Company'.
5. Fill missing values in 'BTTList' column 'BSECode_x' with values from 'BSECode_y'.
6. Create 'background_info_bse' by filtering 'background_info' to include only rows where 'BSECode' is not NaN and greater than 0.
7. Merge 'BTTList' with 'background_info_bse' based on the 'BSECode_x' and 'BSECode' columns using a left join.
8. Replace values in the 'NSECode_y' column with NaN for rows where 'NSECode_y' is "-1".

9. Fill missing values in 'BTTList' column 'NSECode_x' with values from 'NSECode_y'.
10. Fill missing values in 'BTTList' column 'CompanyCode_x' with values from 'CompanyCode_y'.
11. Remove columns 'BSECode' and 'Company' from 'BTTList' in favor of merged columns.
12. Rename columns in 'BTTList' to 'CompanyName', 'NSECode', 'BSECode', and 'CompanyCode'.
13. Define a list 'table_columns' containing the expected column names in the final DataFrame.
14. Get the list of columns in the 'BTTList' DataFrame and store it in 'csv_columns'.
15. Identify columns to remove by finding the difference between 'csv_columns' and 'table_columns', then remove these columns from 'BTTList'.
16. Reorder the columns in 'BTTList' to match the order in 'table_columns'.
17. Replace values in the 'NSECode' column with NaN for rows where 'NSECode' is "-1".
18. Replace values in the 'BSECode' column with NaN for rows where 'BSECode' is -1.
19. Return the enhanced and cleaned 'BTTList' DataFrame.

Note: This function combines background information from the database with 'BTTList', cleans the data, and ensures the resulting DataFrame has the desired columns and format.
"""

"""
Function Name: export_table

Description:
This function exports a given DataFrame 'table' to a CSV file with the specified 'name'. It writes the data to the file, including column headers and a specified float format, and uses a custom line terminator.

Parameters:
- name (str): The name to be used for the exported CSV file.
- table (DataFrame): The DataFrame to be exported to the CSV file.

Returns:
None

Function Description:
1. Create an 'exportfilename' by concatenating the 'name' with "_export.csv" to form the export file name.

2. Open the export file for writing and store it in the 'exportfile' variable.
3. Use the 'to_csv' function to write the 'table' to the 'exportfile' in CSV format with the following options:
   - header=True: Include column headers in the CSV.
   - index=False: Exclude the index column in the CSV.
   - float_format="%2f": Format floating-point numbers with two decimal places.
   - line_terminator='\r': Use '\r' as the line terminator for the CSV file.
4. Close the 'exportfile' to save the data to the CSV file.

Note: This function exports a DataFrame to a CSV file with customization options for headers, index, float format, and line terminator.

"""

"""
Function Name: insert_BTT

Description:
This function prepares and inserts data from the 'BTTList' DataFrame into a database table. It also performs data cleaning and formatting before insertion.

Parameters:
- BTTList (DataFrame): A DataFrame containing data to be inserted into the database.
- conn (Connection): A database connection object.
- cur (Cursor): A database cursor object.

Returns:
None

Function Description:
1. Select specific columns from 'BTTList' to include in the insert operation, and store the result back in 'BTTList'.
2. Remove duplicate rows in 'BTTList' based on the combination of 'CompanyCode' and 'BTTDate' and store the result in 'BTTList'.
3. Fill missing values in the 'BSECode' column with -1 and update 'BTTList' in place.
4. Change the data type of the 'BSECode' column to integer.
5. Convert 'BSECode' values to strings by reassigning them as strings.
6. Replace any values of "-1" in the 'BSECode' column with NaN.
7. Drop rows in 'BTTList' where 'CompanyCode' or 'BTTDate' is missing (NaN).
8. Call the 'export_table' function to export 'BTTList' to a CSV file.
9. Define a SQL statement 'copy_sql' to copy data from the CSV file into the 'BTTList' table in the database.
10. Open the CSV file 'BTTList_export.csv' for reading.

11. Use the 'copy_expert' method of the 'cur' object to execute the SQL statement and copy data from the CSV file into the database table.
12. Commit the changes to the database using the 'conn' connection.
13. Remove the CSV file 'BTTList_export.csv'.

Note: This function prepares and inserts data from 'BTTList' into a database table, handling data cleaning and conversion along the way.

"""

"""
Function Name: main

Description:
This function serves as the main entry point of the program. It orchestrates various data processing steps, including fetching data from the database, generating and merging lists, adding background information, cleaning data, and inserting the final result into the database.

Parameters:
- curr_date (datetime): The current date to be used in the processing.

Returns:
None

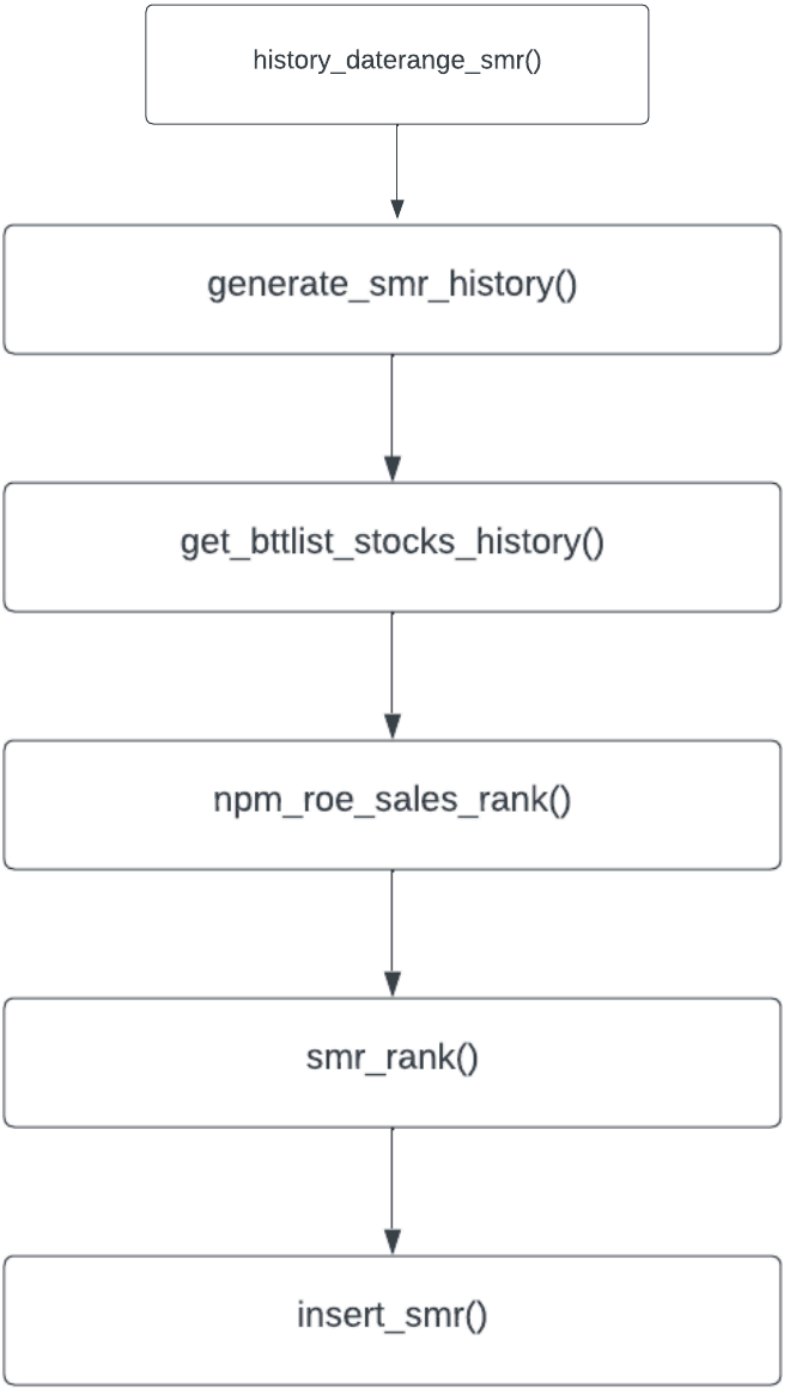
Function Description:
1. Establish a database connection using the 'db_connect' method of 'DB_Helper' and store the connection in 'conn'.
2. Create a database cursor object 'cur'.
3. Print a message to indicate the start of the BTT fetch service.
4. Fetch the 'NSE_january' data and export it to a CSV file.
5. Fetch the 'BSE_january' data and export it to a CSV file.
6. Print a message indicating that BTTList is being generated from 'NSE/BSE 500'.
7. Generate the 'BTTList' by calling the 'generate_btt_list' function and store the result in 'BTTList'.
8. Print the total row count of 'BTTList'.
9. Print a message indicating that 'MFList' is being merged into 'BTTList'.
10. Merge 'MFList' into 'BTTList' by calling the 'merge_MFList' function and update 'BTTList' with the result.
11. Export the 'MFList_january' to a CSV file and print the total row count of 'BTTList'.
12. Print a message indicating that data from 'BackgroundInfo' is being added.
13. Enhance 'BTTList' with background information by calling the 'get_BackgroundInfo' function.

14. Print a message indicating that stocks without OHLC data in the last 30 days are being dropped.
15. Call the 'ohlc_30day_check' function to remove stocks without OHLC data in the last 30 days.
16. Print the total row count of 'BTTList' after dropping stocks.
17. Get the current date in the format "%Y-%m-%d" and store it in 'today_date'.
18. Add a 'BTTDate' column to 'BTTList' with the value of 'today_date'.
19. Call the 'insert_BTT' function to insert 'BTTList' into the database.
20. Export 'BTTList_january' to a CSV file.
21. Print a message indicating the completion of the BTT fetch process.
22. Close the database connection 'conn'.
"""

SMR

Function Name:`SMR_Script()`


generate_smr_current()
generate_ratios_list_current()
compile_ratios_list_current()
compile_sales_npm_roe
merge_background_industry_info()
insert_ratios_merge_list()
get_bttlist_stocks_current()
npm_roe_sales()
smr_rank()
insert_smr()



Description:

This script is designed to compile financial data from various sources and calculate the Simple Monthly Return (SMR) and possibly other financial metrics.

Steps:

1. Import necessary libraries, such as `datetime`, `requests`, `psycopg2`, `pandas`, and more.

2. Define a class named `SMR` to encapsulate the process of generating SMR for a given date.

3. Initialize the `SMR` class in its constructor (`__init__`) without any specific initialization actions.

4. Define the `get_closest_quarter` method to find the closest quarter to a given target date.

   - Create a list of potential quarter-end dates and choose the closest one based on the target date.

   - Return the closest quarter date.

Usage:

- This script can be used to calculate SMR and other financial metrics by instantiating the `SMR` class and calling its methods, such as `get_closest_quarter`.

Function Name: get_closest_quarter(self, target)

Parameters:

self (class): An instance of the class.

target (datetime.date): The target date for which the closest quarter needs to be fetched.

Description:

This function fetches the closest quarter from the given target date. It considers fixed quarter-end dates (March 31, June 30, September 30, December 31) and selects the quarter-end date closest to the target date.

Step 1: Define a list of candidate quarter-end dates.

Candidate quarter-end dates are the last days of December, March, June, September, and December for the previous and current years.

Step 2: Select the closest quarter-end date.

Iterate through the candidate dates.

Remove dates from the list if they are greater than the target date.

Return the minimum date from the remaining candidates based on the absolute distance to the target date.

Return:

closest_quarter (datetime.date): The closest quarter-end date to the target date.

Function Name: `get_previous_quarter(target)`

Parameters:

- `target` (datetime.date): The target date for which you want to find the previous quarter.

Description:

This method fetches the previous quarter from the current quarter based on the provided `target` date.

1. Get the closest quarter to the `target` date by calling the `get_closest_quarter(target)` method.

2. Calculate `curr_qrt_dec` by taking the current quarter date and decrementing the day by 2.

3. Obtain the closest quarter to the `curr_qrt_dec` date using the `get_closest_quarter` method.

4. Return the previous quarter date.

Return:

(datetime.date): The date representing the previous quarter.

Function Name: `get_year_before_quarter(target)`

Parameters:

- `target` (datetime.date): The target date for which you want to find the quarter from one year ago.

Description:

This method fetches the last year's quarter from the closest quarter to the provided `target` date.

1. Get the closest quarter to the `target` date by calling the `get_closest_quarter(target)` method.

2. Call the `get_previous_quarter` method four times in a sequence to navigate back to the quarter of the same month but one year ago.

3. Return the quarter from one year before the provided `target` date.

Return:

(datetime.date): The date representing the quarter from one year before the target date.

Function Name: `get_four_years_before_quarter(target)`

Parameters:

- `target` (datetime.date): The target date for which you want to find the quarter from four years ago.

Description:

This method fetches the quarter of four years ago from the `target` date.

1. Get the quarter from one year before the `target` date by calling the `get_year_before_quarter(target)` method.

2. Call the `get_previous_quarter` method three more times to navigate back to the quarter of the same month but four years ago.

3. Return the quarter from four years before the `target` date.

Return:

(datetime.date): The date representing the quarter from four years before the target date.

Function Name: `compile_ratios_list_history(conn, date)`

Parameters:

- `conn` (psycopg2.connect): A database connection object used to execute SQL queries.

- `date` (datetime.date): The date for which you want to compile the ratios list history.

Description:

This method is responsible for compiling the history of ratios data. It involves fetching data from two database tables, `RatiosBankingVI` and `RatiosNonBankingVI`, for YearEnding data, and concatenating the obtained data.

Operation:

1. Initialize an empty Pandas DataFrame called `ratios_banking_list_init` with specific column names.

2. Execute an SQL query to fetch data from the `RatiosBankingVI` table where the "YearEnding" is less than or equal to the provided `date`. The retrieved data is stored in the `ratios_banking_list` Pandas DataFrame.

3. Execute a similar SQL query for the `RatiosNonBankingVI` table and store the data in the `ratios_nonbanking_list` Pandas DataFrame.

4. Rename columns in the `ratios_nonbanking_list` DataFrame to ensure consistency with the `ratios_banking_list`.

5. Select specific columns from the `ratios_nonbanking_list` DataFrame.

6. Iterate through the rows of the `ratios_banking_list` DataFrame and set the "Debt" column to 0 for each row.

7. Determine whether to use the original `ratios_banking_list` DataFrame or an empty DataFrame (`ratios_banking_list_init`) based on whether the `ratios_banking_list` DataFrame has rows.

8. Concatenate the `ratios_banking_list` and `ratios_nonbanking_list` DataFrames vertically to merge the data.

9. Return the merged data as `ratios_merge_list`.

Return:

(pandas.DataFrame): The merged data from `RatiosBankingVI` and `RatiosNonBankingVI` for generating the history data.

Function Name: `compile_ratios_list_current(conn, date)`

Parameters:

- `conn` (psycopg2.connect): A database connection object used to execute SQL queries.

- `date` (datetime.date): The date for which you want to compile the current ratios list.

Description:

This method is responsible for compiling the current ratios list by fetching data from database tables, including `RatiosBankingVI`, `RatiosNonBankingVI`, and `BTTList`, and merging the data based on the company code.

Operation:

1. Initialize an empty Pandas DataFrame called `ratios_banking_list_init` with specific column names.

2. Execute SQL queries to fetch data from the `RatiosBankingVI` and `RatiosNonBankingVI` tables. The queries select data where "YearEnding" is the maximum date for each company code.

3. Rename columns in the `ratios_nonbanking_list` DataFrame to ensure consistency with the `ratios_banking_list`.

4. Select specific columns from the `ratios_nonbanking_list` DataFrame.

5. Iterate through the rows of the `ratios_banking_list` DataFrame and set the "Debt" column to 0 for each row.

6. Determine whether to use the original `ratios_banking_list` DataFrame or an empty DataFrame (`ratios_banking_list_init`) based on whether the `ratios_banking_list` DataFrame has rows.

7. Concatenate the `ratios_banking_list` and `ratios_nonbanking_list` DataFrames vertically to merge the data.

8. Execute an SQL query to fetch data from the `BTTList` table for the maximum date before or on the provided `date`.

9. Merge the `BTTList` data with the compiled ratios data based on the "CompanyCode" column.

Return:

(pandas.DataFrame): The merged data from `RatiosBankingVI`, `RatiosNonBankingVI`, and `BTTList` for generating the current ratios list.

Function Name: `compile_sales_npm_roe(ratios_merge_list, date, conn)`

Parameters:

- `ratios_merge_list` (pandas.DataFrame): The merged data of Ratios Banking, Ratios Non-banking, and BTTList.

- `date` (datetime.date): The date for which you want to calculate the values.

- `conn` (psycopg2.connect): A database connection object used to execute SQL queries.

Description:

This method calculates the values of Sales Growth, Net Profit Margin (NPM), and other metrics. It involves fetching data from the TTM (Trailing Twelve Months) table for a specific date and performing calculations based on the retrieved data.

Operation:

1. Determine the date one year back and four years back from the provided `date`.

2. Execute SQL queries to fetch data from the TTM table where "YearEnding" is the maximum date for each company code.

3. Execute an additional SQL query to fetch data from the TTM table for the period between four years back and one year back.

4. Cleanse the "Sales" data by removing currency symbols and converting it to a float for both TTM dataframes.

5. Iterate through the rows of the `ratios_merge_list` DataFrame and calculate the values for each company:

   - Find the TTM YearEnding for the current company based on the company code.

   - Determine the previous year's quarter using the `get_year_before_quarter` method.

   - Retrieve the NPM and TTM Sales values for the current and previous quarters.

   - Calculate Sales Growth as a percentage.

6. Update the `ratios_merge_list` DataFrame with the calculated values.

Return:

(pandas.DataFrame): The `ratios_merge_list` DataFrame with additional columns for TTM YearEnding, NPM, and Sales Growth.

Function Name: `merge_background_industry_info(ratios_merge_list, conn)`

Parameters:

- `ratios_merge_list` (pandas.DataFrame): Data containing Sales Growth and NPM, as well as other related metrics.

- `conn` (psycopg2.connect): A database connection object used to execute SQL queries.

Description:

This method is responsible for merging background information and industry mapping data with the existing `ratios_merge_list` based on the "CompanyCode" and "IndustryCode" columns.

Operation:

1. Execute an SQL query to fetch data from the "BackgroundInfo" table and store it in the `background_info` Pandas DataFrame.

2. Merge the `ratios_merge_list` with the `background_info` DataFrame based on the "CompanyCode" column, adding information such as "CompanyName" and "IndustryCode" to the `ratios_merge_list`.

3. Execute another SQL query to fetch data from the "IndustryMapping" table and store it in the `industry_info` Pandas DataFrame.

4. Merge the `ratios_merge_list` with the `industry_info` DataFrame based on the "IndustryCode" column, adding information about the industry.

5. Drop the "IndustryCode" column from the resulting `ratios_merge_list` DataFrame.

Return:

(pandas.DataFrame): The `ratios_merge_list` DataFrame with additional columns for "CompanyName" and "Industry," obtained by merging data from "BackgroundInfo" and "IndustryMapping."

Additional Description:

This method plays a crucial role in enhancing the `ratios_merge_list` by enriching it with background information and industry mapping data. It provides more context by

incorporating the "CompanyName" and "Industry" details into the dataset. This enriched data is valuable for further analysis and reporting. The method ensures that the information is seamlessly integrated into the existing DataFrame, making it a comprehensive source for financial and industry-related insights.

Function Name: `insert_ratios_merge_list(ratios_merge_list, conn, cur, date)`

Parameters:

- `ratios_merge_list` (pandas.DataFrame): Data of ratios list to be inserted into the database.

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

- `cur` (psycopg2.cursor): A database cursor object used to execute SQL commands.

- `date` (datetime.date): The date for which the data is generated.

Description:

This method is responsible for exporting the data from the `ratios_merge_list` into a CSV file and then inserting this data into the "RatiosMergeList" table in the database.

Operation:

1. Create an empty DataFrame `ratios_merge_list_init` with the expected column structure.

2. Add a "GenDate" column to the `ratios_merge_list` containing the specified date.

3. Handle data type conversions and data cleaning for columns such as "Months," "FaceValue," "ROE," and "Debt."

4. Rename the "YearEnding" column to "ROEYearEnding."

5. Select specific columns from the `ratios_merge_list` to match the desired structure.

6. Print a message indicating that the Ratios Merge List is being inserted.

7. Export the data to a CSV file named "ratiosmerge_export.csv."

8. Use the COPY command to insert the data from the CSV file into the "RatiosMergeList" table.

9. Commit the changes to the database.

Return:

None

Additional Description:

This method handles the process of inserting the Ratios Merge List data into the database. It takes care of data formatting, cleaning, and exporting to ensure that the data is consistent with the database schema. The data is exported to a CSV file for further use and then efficiently loaded into the database using the COPY command. The "GenDate" column is added to track the date of data generation, providing valuable information for historical records.

Function Name: `get_bttlist_stocks_history(date, conn)`

Parameters:

- `date` (datetime.date): The date for which the data is to be fetched.

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

Description:

This method is responsible for fetching the historical data related to BTTList stocks and merging it with the data from the RatiosMergeList to generate SMR rank history.

Operation:

1. Define two date ranges, `BTT_back` and `BTT_next`, which specify the start and end dates for fetching BTTList data.

2. Construct a SQL query that retrieves data from both the BTTList and RatiosMergeList tables. The query performs a LEFT JOIN based on the "CompanyCode" column and selects data where the "BTTDate" falls within the specified date range.

3. Rename the "Company Code" column to "CompanyCode" in the result to match the expected column name.

4. Execute the SQL query and fetch the merged data into a DataFrame named `btt_smr_merge_list`.

Return:

(pandas.DataFrame) - Merge data of BTTList and RatiosMergeList, representing SMR rank history.

Additional Description:

This method retrieves historical data related to BTTList stocks within a specified date range and merges it with the data from the RatiosMergeList. The result is a DataFrame that provides SMR rank history for the specified date. The data is collected based on the "CompanyCode," and the resulting DataFrame can be used for further analysis and reporting.

Function Name: `get_bttlist_stocks_current(conn, today)`

Parameters:

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

- `today` (datetime.date): The current date for which the data is to be fetched.

Description:

This method is responsible for fetching the BTTList data for the current date and merging it with data from the RatiosMergeList, providing information about stocks available on the BTTList for the specified date.

Operation:

1. Calculate two date ranges, `BTT_back` and `BTT_next`, which specify the start and end dates for fetching BTTList data for the current month.

2. Construct a SQL query that retrieves data from both the BTTList and RatiosMergeList tables. The query performs a LEFT JOIN based on the "CompanyCode" column and selects data where the "BTTDate" falls within the specified date range.

3. Rename the "Company Code" column to "CompanyCode" in the result to match the expected column name.

4. Execute the SQL query and fetch the merged data into a DataFrame named `btt_smr_merge_list`.

Return:

(pandas.DataFrame) - Merge data of BTTList and RatiosMergeList, representing stocks available on the BTTList for the current month.

Additional Description:

This method retrieves the BTTList data for the current month, specifically the stocks available on the BTTList for the specified date. The data is merged with information from

the RatiosMergeList. The resulting DataFrame contains details about stocks available on the BTTList for the current month and can be used for various analyses or reports.

Function Name: `npm_roe_sales_rank(btt_ratios_list, conn)`

Parameters:

- `btt_ratios_list` (pandas.DataFrame): Data of BTTList and RatiosMergeList for the first of the month, including information about NPM, ROE, and Sales Growth.

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

Description:

This method is responsible for calculating the ranks and percentages of NPM, ROE, and Sales Growth based on the provided data. It assigns rank values and computes percentages for each of these financial metrics.

Operation:

1. Calculate the NPM Rank:

   - Rank the NPM values in descending order.

   - Calculate the NPM Rank as ((Total number of rows - NPM Rank + 1) / Total number of rows) 100.

2. Calculate the ROE Rank:

   - Rank the ROE values in descending order.

   - Calculate the ROE Rank as ((Total number of rows - ROE Rank + 1) / Total number of rows) 100.

3. Calculate the Sales Growth Rank:

   - Rank the Sales Growth values in descending order.

- Calculate the Sales Growth Rank as ((Total number of rows - Sales Growth Rank + 1) / Total number of rows) 100.

Return:

(pandas.DataFrame) - The input DataFrame `btt_ratios_list` with additional columns for NPM Rank, ROE Rank, and Sales Growth Rank, as well as their corresponding percentage values.

Additional Description:

This method takes a DataFrame containing data related to NPM, ROE, and Sales Growth and calculates rank values and percentages for each of these financial metrics. The resulting DataFrame includes NPM Rank, ROE Rank, and Sales Growth Rank, along with their corresponding percentage values, providing insights into the relative performance of these metrics within the dataset. These rankings and percentages can be useful for analyzing and comparing financial metrics for various stocks.

Function Name: `smr_rank(btt_ratios_list, conn)`

Parameters:

- `btt_ratios_list` (pandas.DataFrame): Data of BTTList and RatiosMergeList for the first of the month, including information about NPM, ROE, Sales Growth, and their respective ranks.

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

Description:

This method calculates the SMR (Sales, Margin, and Returns) Rank for each entry in the provided dataset, taking into account the ranks of NPM, ROE, and Sales Growth. SMR Rank is determined based on grade variations for each metric, and the final SMR Grade and Rank are computed.

Operation:

1. Calculate the Sales Growth (Sales Gr) Grade:

   - Determine the grade variation based on the Sales Growth Rank:

      - 'A' for ranks >= 80 to <= 100

      - 'B' for ranks >= 60 to < 80

      - 'C' for ranks >= 40 to < 60

      - 'D' for ranks >= 20 to < 40

      - 'E' for ranks >= 0 to < 20

      - Assign 'E' for other cases.

2. Calculate the NPM Grade:

   - Determine the grade variation based on the NPM Rank using the same grade variations as for Sales Growth.

3. Calculate the ROE Grade:

   - Determine the grade variation based on the ROE Rank using the same grade variations as for Sales Growth.

4. Calculate SMR Grade:

   - Combine the Sales Gr, NPM Gr, and ROE Gr to form the SMR Grade.

5. Assign SMR Ratings:

   - Map grade variations to numerical values using predefined grade-to-value maps.

   - Calculate the SMR Rating as the sum of the numerical values for Sales Gr, NPM Gr, and ROE Gr.

6. Calculate the SMR Rank:

   - Rank the SMR Ratings in descending order.

   - Calculate the SMR Rank as ((Total number of rows - SMR Rank + 1) / Total number of rows) 100.

Return:

(pandas.DataFrame) - The input DataFrame `btt_ratios_list` with additional columns for Sales Gr, NPM Gr, ROE Gr, SMR Grade, and SMR Rank, as well as their corresponding percentage values.

Additional Description:

This method takes a DataFrame containing data related to NPM, ROE, Sales Growth, and their respective ranks. It calculates the grade variations for each metric, combines them to determine the SMR Grade, and assigns an SMR Rating based on numerical values. The final SMR Rank is calculated and added to the dataset. This information provides insights into the relative performance of stocks in terms of Sales, Margin, and Returns, making it useful for stock analysis and comparison.

Function Name: `insert_smr(btt_ratios_list, conn, cur, date)`

Parameters:

- `btt_ratios_list` (pandas.DataFrame): Calculated data of SMR (Sales, Margin, and Returns) including information about SMR Rank, grades, and other stock-related details.

- `conn` (psycopg2.connect): A database connection object used to interact with the database.

- `cur` (psycopg2.cursor): A database cursor object used for executing database queries.

- `date` (datetime.date): The date for which the SMR data is being inserted into the database.

Description:

This method inserts the calculated SMR data into a database table named "SMR." It prepares the data by adding a "SMRDate" column with the specified date and ensures the correct data types for specific columns before exporting it to a CSV file. The CSV file is then imported into the "SMR" table in the database.

Operation:

1. Add a "SMRDate" column to the input DataFrame to specify the date for the SMR data.

2. Handle data type conversion for columns "BSECode" and "Months" to ensure they are of integer type where applicable and convert them to strings. Replace missing values with NaN where necessary.

3. Select specific columns from the DataFrame to include in the final data for insertion into the database.

4. Export the data to a CSV file named "smr_export.csv."

5. Use the PostgreSQL `COPY` command to insert the data from the CSV file into the "SMR" table in the database.

6. Commit the changes to the database.

Return:

None

Additional Description:

This method is used to insert SMR data into a database. It first prepares the data by adding a date, handling data types, and selecting the relevant columns. The data is then exported to a CSV file and subsequently inserted into the "SMR" table in the database. The "SMR" table stores information about stock performance and SMR ranks, which can be useful for financial analysis and reporting.

Function Name: `generate_ratios_list_history(date)`

Parameters:

- `date` (datetime.date): The date for which the Ratios List History is generated.

Description:

This method generates the Ratios List History by compiling data from various sources, calculating sales growth and net profit margin (NPM), merging background information, and inserting the resulting data into the Ratios Merge List table.

Operation:

1. Establish a database connection (`conn`) and create a database cursor (`cur`) to interact with the database.

2. Obtain the current date as `today` for the operation.

3. Display a message indicating the compilation date.

4. Compile the Ratios List History by performing the following steps:

   a. Compile data from the "RatiosBanking" and "RatiosNonBanking" tables, and store the result in the `ratios_merge_list` variable.

   b. Calculate the values of sales growth and NPM for the trailing twelve months (TTM) for the specified date.

   c. Merge background information (e.g., company name and industry) into the `ratios_merge_list`.

5. Insert the compiled data into the "RatiosMergeList" table in the database.

6. Close the database connection.

Return:

None

Additional Description:

This method serves as a comprehensive process for generating Ratios List History. It starts by establishing a database connection and cursor. Then, it compiles data from various

tables, calculates financial ratios, and merges additional information to create the Ratios List History. The resulting data is inserted into the "RatiosMergeList" table, which can be used for financial analysis and reporting. This function streamlines the process of collecting and storing financial data for historical analysis.

Function Name: `generate_ratios_list_current(conn, cur, today)`

Parameters:

- `conn` (database connection): The database connection used to access data.

- `cur` (database cursor): The database cursor for executing SQL queries.

- `today` (datetime.date): The current date for which the Ratios List is generated.

Description:

This method generates the Ratios List for the current date by fetching data from various sources, calculating sales growth and net profit margin (NPM), merging background information, and inserting the resulting data into the Ratios Merge List table.

Operation:

1. Display a message indicating the compilation process is starting.

2. Compile the Ratios List for the current date by performing the following steps:

   a. Compile data from the "RatiosBanking" and "RatiosNonBanking" tables and store the result in the `ratios_merge_list` variable.

   b. Calculate the values of sales growth and NPM for the trailing twelve months (TTM) for the specified date.

   c. Merge background information (e.g., company name and industry) into the `ratios_merge_list`.

3. Insert the compiled data into the "RatiosMergeList" table in the database.

Return:

None

Additional Description:

This method serves as a means to generate the Ratios List for the current date. It combines data from multiple sources and calculates essential financial ratios, facilitating financial analysis and reporting. The compiled data is inserted into the "RatiosMergeList" table, enabling users to access up-to-date financial information and make informed decisions.

Function Name: `generate_smr_history(date)`

Parameters:

- `date` (datetime.date): The date for which the SMR history is generated.

Description:

This method generates SMR (Sales, Margin, and Rank) history for a specific date. It involves fetching data from BTT stocks, calculating percentile values for Net Profit Margin (NPM), Return on Equity (ROE), and Sales Growth, and then determining the SMR rank.

Operation:

1. Establish a database connection and create a database cursor.

2. Retrieve the current date as `today`.

3. Display a message indicating the initiation of SMR generation for the specified date.

4. Fetch BTT (Buy Today, Target) stocks data for SMR ranking:

   a. Fetch BTT stocks data for the specified date using the `get_bttlist_stocks_history` method.

   b. Calculate the number of stocks retrieved and print this information.

5. Calculate the percentile values for NPM, ROE, and Sales Growth for the BTT stocks using the `npm_roe_sales_rank` method.

6. Calculate the SMR rank for the BTT stocks based on the calculated percentile values using the `smr_rank` method.

7. Insert the calculated SMR data into the SMR database table using the `insert_smr` method.

8. Close the database connection and cursor.

Return:

None

Additional Description:

This method is crucial for generating SMR history, which provides insights into the relative performance of stocks based on their NPM, ROE, and Sales Growth. By calculating percentile values and SMR ranks, investors and analysts can identify stocks with strong financial indicators and make informed investment decisions. The SMR history is useful for tracking changes in stock performance over time.

Function Name: `generate_smr_current(curr_date, conn, cur)`

Parameters:

- `curr_date` (datetime.date): The current date for which SMR is generated.

- `conn` (database connection): The database connection object.

- `cur` (database cursor): The database cursor object.

Description:

This method generates SMR (Sales, Margin, and Rank) data for the current date. It involves compiling ratios merge list, fetching data for BTT (Buy Today, Target) stocks, calculating percentile ranks for Net Profit Margin (NPM), Return on Equity (ROE), and Sales Growth, and determining the SMR rank for current date.

Operation:

1. Retrieve the current date as `today`.

2. Display a message indicating the compilation of Ratios Merge List for the current date.

3. Call the `generate_ratios_list_current` method to compile the Ratios Merge List for the current date.

4. Display a message indicating the fetching of BTT stocks for SMR ranking.

5. Fetch BTT stocks data for the current date using the `get_bttlist_stocks_current` method.

   - Calculate the number of stocks retrieved and print this information.

6. Check if BTT stocks data is not empty:

   a. If data is present, proceed with the following operations.

   b. Display a message indicating the calculation of percentile values for NPM, ROE, and Sales Growth using the `npm_roe_sales_rank` method.

   c. Display a message indicating the calculation of the SMR rank using the `smr_rank` method.

   d. Insert the calculated SMR data into the SMR database table using the `insert_smr` method.

7. If BTT stocks data is empty, print a message indicating that data is not present for BTT stocks for SMR ranking for the current date.

Return:

None

Additional Description:

This method is essential for generating SMR data for the current date. SMR data helps investors and analysts evaluate stocks based on their NPM, ROE, and Sales Growth, making it easier to identify stocks with strong financial indicators and make informed investment decisions. The method ensures that SMR data is up-to-date and reflects the latest stock performance metrics.

Function Name: `history_daterange_ratioslist`

Description:

This method is used to generate the RatiosMergeList for a historical date range. It iterates over a specified date range, generates the RatiosMergeList for each date, and updates the database with historical data.

Operation:

1. Define the start date (`start_date`) and end date (`end_date`) for the historical date range.

2. Get the closest quarter-end date to the `end_date` and assign it to `curr_date`.

3. While `curr_date` is greater than or equal to `start_date`, repeat the following steps:

   a. Call the `generate_ratios_list_history` method to generate the RatiosMergeList for `curr_date`.

   b. Update `curr_date` to the previous quarter-end date using the `get_previous_quarter` method.

Return:

None

Function Name: `history_daterange_smr`

Description:

This method is used to generate SMR (Sales, Margin, and Rank) data for a historical date range. It iterates over a specified date range, generates SMR data for each date, and updates the database with historical SMR data.

Operation:

1. Define the start date (`start_date`) and end date (`end_date`) for the historical date range.

2. While `end_date` is greater than or equal to `start_date`, repeat the following steps:

   a. Call the `generate_smr_history` method to generate SMR data for `start_date`.

   b. Increment `start_date` by one day using the `datetime.timedelta` function.

Return:

None

Additional Description:

These methods are useful for generating historical financial data, including the RatiosMergeList and SMR data, for a specified date range. They allow you to analyze the historical performance of stocks and make informed investment decisions based on past financial metrics.

Function Name: `export_table`

Description:

This method is used to export a pandas DataFrame to a CSV file. It takes a DataFrame and a name as input and creates a CSV file with a specified name for data export.

Arguments:

- `name`: A string that represents the name of the CSV file.

- `table`: A pandas DataFrame that contains the data to be exported.

Operation:

1. Create a variable `exportfilename` by concatenating the string "__SMR_" and the provided `name`, with ".csv" appended to the end. This forms the name of the CSV file to be created.

2. Open the `exportfilename` in write mode and assign it to the `exportfile` variable.

3. Use the `to_csv` method of the DataFrame to export its data to the opened `exportfile`. The `header` parameter is set to `True` to include column headers in the CSV, the `index` parameter is set to `False` to exclude row indices, and the `float_format` parameter is set to "%.2f" to format floating-point numbers with two decimal places. The `line_terminator` is set to '\r' to specify the line terminator used in the CSV.

4. Close the `exportfile` to save the exported data to the CSV file.

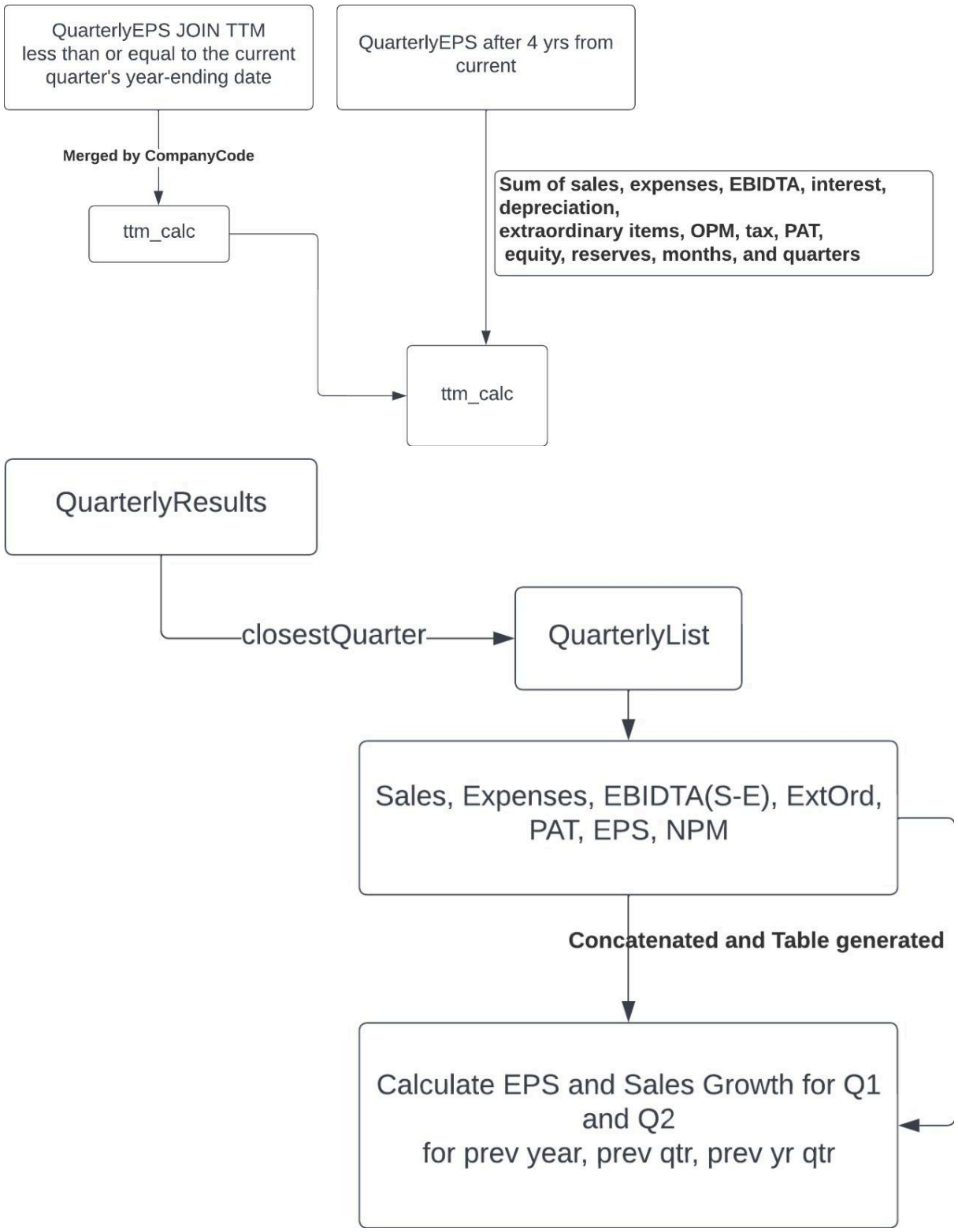
Return:

None

Additional Description:

This method is useful for saving data from a pandas DataFrame to a CSV file, making it easier to share or store data for analysis or other purposes. The `name` argument allows you to specify the name of the output file, and the method automatically handles the export process, including formatting options.

EPS



- quarterly_eps_history_insert

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function is responsible for inserting quarterly EPS history data into a database. The operation involves the following steps:

- Retrieve data from the "QuarterlyResults" table for the closest quarter.

- Rename columns in the retrieved data.

- Perform data transformations and calculations on the retrieved data.

- Calculate quarterly EPS growth and sales growth.

- Prepare the data for insertion into the database.

Return:

None

Details:

- The function first determines the start quarter and retrieves relevant data from the "QuarterlyResults" table.

- It renames columns to more meaningful names.

- Data transformation is performed on selected columns to convert values to float.

- Sales and expenses are calculated based on various components.

- Key financial metrics like EBITDA, PAT, EPS, and NPM are computed.

- The data is enriched with quarterly EPS growth and sales growth for Q1 and Q2.

- The final dataset is prepared for insertion into the database.

- The `insert_quarterly_eps_resulsts` method is called to insert the data into the database

- inset_ttm_history

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function is responsible for inserting TTM (Trailing Twelve Months) history data into a database. The operation involves the following steps:

- Iterates through a date range, calculating TTM values for each quarter.

- Calls `ttm_quarterly_list` to retrieve TTM data.

- Inserts the TTM data into the database.

Return:

None

Details:

- The function iterates through a date range starting from `today1` and goes back in time.

- For each quarter in the date range, it calculates TTM values using the `ttm_quarterly_list` method.

- The TTM values are then inserted into the database.

- set_daily_qtr_eps

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function fetches daily quarterly EPS data, performs calculations, and prepares the data for insertion into the database.

Return:

Quarterly EPS list.

Details:

- The function retrieves data from the "QuarterlyResults" table where certain conditions are met.

- It renames columns for better readability.

- Data transformation and calculations are performed on the data.

- Key financial metrics like EPS, NPM, and other flags are computed.

- The data is prepared as a Pandas DataFrame for further processing.

- Additional columns for Q1 and Q2 EPS growth, Q2 EPS, and Q2 sales growth are added to the DataFrame.

Please note that specific details of data insertion into the database and the implementation of methods like `ttm_quarterly_list` are not visible in the provided code snippet. The code focuses on data retrieval, transformation, and calculation.

- consolidated_set_daily_qtr_eps

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function fetches daily consolidated quarterly EPS data, compares it with existing data, and calculates various financial metrics, including EPS and NPM.

Return:

Quarterly EPS list.

Details:

- The function takes the `today` date and retrieves quarterly data from the "ConsolidatedQuarterlyResults" table.

- It renames columns for clarity.

- Data transformation and calculations are performed.

- The function calculates key financial metrics like EPS, NPM, and additional flags.

- The data is prepared as a Pandas DataFrame with extra columns for Q1 and Q2 metrics.

- insert_quarterly_eps_results

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function inserts the quarterly EPS data into the database. It also prepares a CSV file for exporting the data.

Return:

None

Details:

- The function starts by converting NaN values in certain columns to -1 and then to strings to handle missing data.

- It extracts specific columns from the DataFrame.

- The data is exported to a CSV file named "QuarterlyEPSListExport.csv."

- The data from the CSV file is copied into the "QuarterlyEPS" table in the database using the COPY command.

- The function updates the "Ext_Flag" column in the database to ensure it's correctly set.

Please note that the `copy_expert` method is used to efficiently copy data from a CSV file to the database. The function ensures that missing or null values are correctly handled during this process.

This code provides a comprehensive process for updating and inserting quarterly EPS data into the database, along with necessary data quality checks and transformations.

- insert_consolidated_quarterly_eps_results

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function inserts the quarterly consolidated EPS data into the database. It also prepares a CSV file for exporting the data.

Return:

None

Details:

- The function starts by filling NaN values in certain columns with -1, which are later converted to integers and then to strings for handling missing data gracefully.

- It specifically works on columns 'Months', 'Quarter', and 'Ext_Flag'.

- The data is extracted from the DataFrame, and unnecessary columns are dropped.

- The data is exported to a CSV file named "ConsolidatedQuarterlyEPSListExport.csv."

- The data from the CSV file is copied into the "ConsolidatedQuarterlyEPS" table in the database using the COPY command.

- The function includes an update query to ensure that the "Ext_Flag" column in the database is correctly set.

It's worth noting that there's a comment related to handling duplicate data, but it is not clear how duplicates are being managed in the function. You may want to further clarify the intention and purpose of this commented code section.

- quarterly_one_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the EPS and Sales Growth for the current quarter by comparing the data with the previous year's quarter.

Return:

- Pandas DataFrame containing the quarterly EPS data with added columns for 'Q1 EPS Growth' and 'Q1 Sales Growth'.

Details:

- The function determines the current quarter, the year of the quarter before the current year, two years before, and four years before based on the provided `today` date.

- It extracts quarterly EPS data for the previous four years from the "QuarterlyEPS" table, excluding rows with a null "Ext_Flag."

- The function ensures the 'Sales' column data is converted to float and removes currency symbols (e.g., "$").

- It sorts the `quarterly_eps_list` by 'YearEnding' in ascending order.

- For each row in the `quarterly_eps_list`, the function calculates 'Q1 EPS Growth' and 'Q1 Sales Growth':

- It retrieves the EPS and Sales data for the corresponding quarter of the previous year.

- If data is not found in `quarterly_eps_year_back`, it attempts to find it in the current year's data.

- 'Q1 EPS Growth' and 'Q1 Sales Growth' are calculated using the formulas provided.

- The calculated values are added to the DataFrame.

- The function returns the updated `quarterly_eps_list` with 'Q1 EPS Growth' and 'Q1 Sales Growth' columns.

The code seems to be well-documented with inline comments to explain the logic and calculations performed for EPS and Sales Growth. Ensure that the data retrieved from the database matches the expected structure and column names. Additionally, you may consider enhancing readability by creating separate helper functions for certain operations or error handling when data is not found.

- consolidated_quarterly_one_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the EPS and Sales Growth for the current quarter by comparing the data with the previous year's quarter. It specifically targets "ConsolidatedQuarterlyEPS" data.

Return:

- Pandas DataFrame containing the quarterly EPS data with added columns for 'Q1 EPS Growth' and 'Q1 Sales Growth'.

Details:

- The function determines the current quarter, the year of the quarter before the current year, two years before, and four years before based on the provided `today` date.

- It extracts quarterly EPS data from the "ConsolidatedQuarterlyEPS" table for the previous four years, excluding rows with a null "Ext_Flag."

- The function ensures the 'Sales' column data is converted to float and removes currency symbols (e.g., "$").

- It sorts the `quarterly_eps_list` by 'YearEnding' in ascending order.

- For each row in the `quarterly_eps_list`, the function calculates 'Q1 EPS Growth' and 'Q1 Sales Growth':

- It retrieves the EPS and Sales data for the corresponding quarter of the previous year from the `quarterly_eps_year_back` DataFrame.

- The function calculates 'Q1 EPS Growth' and 'Q1 Sales Growth' using the provided formulas.

- The calculated values are added to the DataFrame.

- The function returns the updated `quarterly_eps_list` with 'Q1 EPS Growth' and 'Q1 Sales Growth' columns.

The code appears to be similar to the previous function but specifically targeting "ConsolidatedQuarterlyEPS" data. Ensure that the data retrieved from the database matches the expected structure and column names. Additionally, you may consider enhancing readability by creating separate helper functions for certain operations or error handling when data is not found.

- quarterly_two_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the EPS and Sales Growth for the current quarter by comparing the data with the quarter from two years ago. It targets "QuarterlyEPS" data.

Return:

- Pandas DataFrame containing the quarterly EPS data with added columns for 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales.'

Details:

- The function determines the current quarter, the year of the quarter before the current year, and two years before based on the provided `today` date.

- It extracts quarterly EPS data from the "QuarterlyEPS" table for the previous two years, excluding rows with a null "Ext_Flag."

- The function ensures the 'Sales' column data is converted to float and removes currency symbols (e.g., "$").

- For each row in the `quarterly_eps_list`, the function calculates 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales':

- It retrieves the EPS and Sales data for the corresponding quarter and the previous year from the `quarterly_eps_year_back` DataFrame.

- The function calculates 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales' using the provided formulas.

- The calculated values are added to the DataFrame.

- The function returns the updated `quarterly_eps_list` with 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales' columns.

The code effectively calculates EPS and Sales Growth for the current quarter compared to the quarter from two years ago. Ensure that the data retrieved from the database matches the expected structure and column names. Additionally, you may consider enhancing readability by creating separate helper functions for certain operations or error handling when data is not found.

- consolidated_quarterly_two_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the EPS and Sales Growth for the current quarter by comparing the data with the quarter from two years ago. It targets "ConsolidatedQuarterlyEPS" data.

Return:

- Pandas DataFrame containing the consolidated quarterly EPS data with added columns for 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales.'

Details:

- The function determines the current quarter, the year of the quarter before the current year, and two years before based on the provided `today` date.

- It extracts consolidated quarterly EPS data from the "ConsolidatedQuarterlyEPS" table for the previous two years, excluding rows with a null "Ext_Flag."

- The function ensures the 'Sales' column data is converted to float and removes currency symbols (e.g., "$").

- For each row in the `quarterly_eps_list`, the function calculates 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales':

- It retrieves the EPS and Sales data for the corresponding quarter and the previous year from the `quarterly_eps_year_back` DataFrame.

- The function calculates 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales' using the provided formulas.

- The calculated values are added to the DataFrame.

- The function returns the updated `quarterly_eps_list` with 'Q2 EPS Growth,' 'Q2 EPS,' 'Q2 Sales Growth,' and 'Q2 Sales' columns.

The code effectively calculates EPS and Sales Growth for the current quarter based on "ConsolidatedQuarterlyEPS" data from two years ago. Ensure that the data retrieved from the database matches the expected structure and column names. Additionally, you may consider enhancing readability by creating separate helper functions for certain operations or error handling when data is not found.

- ttm_quarterly_list

Parameters:

- `self` (object)

- `date` (datetime.date)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function generates TTM (Trailing Twelve Months) data for newly available data in the quarterly list. It fetches data from the "QuarterlyEPS" and "TTM" tables for the previous four years' quarters and calculates values for 'EPS' and 'NPM' (Net Profit Margin).

Return:

- Pandas DataFrame containing the TTM data for 'EPS' and 'NPM' alongside other relevant financial information.

Details:

- The function begins by identifying the current quarter and the quarter from one year ago based on the provided `date`.

- It fetches data from the "QuarterlyEPS" table where the year ending is earlier than or equal to the current quarter, and there is no corresponding data in the "TTM" table. It also ensures that 'Ext_Flag' is not null.

- The code processes the fetched data by removing currency symbols from columns and converting them to float.

- A new DataFrame named `ttm_calc` is created with columns for various financial data.

- The function then iterates through the quarterly data to calculate TTM values for each company:

- It determines the TTM months and quarters for a specific company within the time frame.

- It calculates sums for various financial parameters such as 'Sales,' 'Expenses,' 'EBIDTA,' 'Interest,' 'Depreciation,' 'Extraordinary,' 'OPM,' 'Tax,' 'PAT,' 'Equity,' and 'Reserves.'

- The 'Months' and 'Quarter' columns are updated with the corresponding sums.

- 'EPS' (Earnings Per Share) and 'NPM' (Net Profit Margin) are calculated and added to the DataFrame. These calculations depend on 'PAT' (Profit After Tax), 'Equity,' and 'Sales' among others.

- The final DataFrame is reordered to have specific columns and returned, containing 'CompanyCode,' 'YearEnding,' 'Months,' 'Quarter,' 'Sales,' 'Expenses,' 'EBIDTA,' 'Interest,' 'Depreciation,' 'Extraordinary,' 'OPM,' 'Tax,' 'PAT,' 'Equity,' 'Reserves,' 'EPS,' and 'NPM.'

The code effectively generates TTM data for 'EPS' and 'NPM' based on "QuarterlyEPS" and "TTM" data for the previous four years' quarters. Ensure that the data retrieved from the database matches the expected structure and column names. Consider improving readability by creating separate functions for certain calculations or providing more descriptive variable names.

- consolidated_ttm_quarterly_list

Parameters:

- `self` (object)

- `date` (datetime.date)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function generates TTM (Trailing Twelve Months) data for newly available data in the consolidated quarterly list. It fetches data from the "ConsolidatedQuarterlyEPS"

and "ConsolidatedTTM" tables for the previous four years' quarters and calculates values for 'EPS' and 'NPM' (Net Profit Margin).

Return:

- Pandas DataFrame containing the TTM data for 'EPS' and 'NPM' alongside other relevant financial information.

Details:

- The function begins by identifying the current quarter and the quarter from one year ago based on the provided `date`.

- It fetches data from the "ConsolidatedQuarterlyEPS" table where the year ending is earlier than or equal to the current quarter, and there is no corresponding data in the "ConsolidatedTTM" table. It also ensures that 'Ext_Flag' is not null.

- The code processes the fetched data by removing currency symbols from columns and converting them to float.

- A new DataFrame named `ttm_calc` is created with columns for various financial data.

- The function then iterates through the consolidated quarterly data to calculate TTM values for each company:

- It determines the TTM months and quarters for a specific company within the time frame.

- It calculates sums for various financial parameters such as 'Sales,' 'Expenses,' 'EBIDTA,' 'Interest,' 'Depreciation,' 'Extraordinary,' 'OPM,' 'Tax,' 'PAT,' 'Equity,' and 'Reserves.'

- The 'Months' and 'Quarter' columns are updated with the corresponding sums.

- 'EPS' (Earnings Per Share) and 'NPM' (Net Profit Margin) are calculated and added to the DataFrame. These calculations depend on 'PAT' (Profit After Tax), 'Equity,' and 'Sales' among others.

- The final DataFrame is reordered to have specific columns and returned, containing 'CompanyCode,' 'YearEnding,' 'Months,' 'Quarter,' 'Sales,' 'Expenses,' 'EBIDTA,' 'Interest,' 'Depreciation,' 'Extraordinary,' 'OPM,' 'Tax,' 'PAT,' 'Equity,' 'Reserves,' 'EPS,' and 'NPM.'

The code effectively generates TTM data for 'EPS' and 'NPM' based on "ConsolidatedQuarterlyEPS" and "ConsolidatedTTM" data for the previous four years' quarters. Ensure that the data retrieved from the database matches the expected structure and column names. Consider improving readability by creating separate functions for certain calculations or providing more descriptive variable names.

- ttm_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `today` (datetime.date)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function generates TTM (Trailing Twelve Months) data, sales growth, EPS (Earnings Per Share) growth, and an EPS rating based on the provided `quarterly_eps_list` and TTM data from the database.

Return:

- The modified `quarterly_eps_list` DataFrame with additional columns for TTM1 EPS, TTM1 Sales, TTM2 EPS, TTM2 Sales, TTM3 EPS, TTM3 Sales, and an EPS Rating.

Details:

- The function starts by retrieving the TTM data for the last five years, taking into account the current date. It also replaces currency symbols in the 'Sales' column and converts it to float.

- New columns are added to the `quarterly_eps_list` to store TTM1, TTM2, and TTM3 EPS and Sales data, along with an 'EPS Rating.'

- The function iterates through each row in `quarterly_eps_list` to calculate the TTM1, TTM2, and TTM3 EPS and Sales growth rates:

- For each time period, it retrieves the previous year's quarter data for the same company.

- It calculates EPS and Sales growth rates using the formulas provided in the comments.

- The results are added to the respective columns in the DataFrame.

- Additionally, the 'EPS Rating' is calculated based on TTM1 EPS values, and a weighted value is added to the 'EPS Rating' column.

- The code contains logging and debugging information in the form of writing to files and printing counts of zero values.

- The final `quarterly_eps_list` with the added columns is returned.

The code successfully calculates TTM1, TTM2, and TTM3 EPS and Sales growth rates, as well as an 'EPS Rating' based on TTM1 EPS values. However, there is a lot of debug information and comments about debugging within the code. Consider removing or commenting out these debugging elements in the final version to enhance code readability.

- consolidated_ttm_eps_sales_growth

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function generates TTM (Trailing Twelve Months) data, sales growth, EPS (Earnings Per Share) growth, and an EPS rating based on the provided `quarterly_eps_list` and TTM data from the database.

Return:

- The modified `quarterly_eps_list` DataFrame with additional columns for TTM1 EPS, TTM1 Sales, TTM2 EPS, TTM2 Sales, TTM3 EPS, TTM3 Sales, and an EPS Rating.

Details:

- The function calculates the current date and defines a `four_year_back` date by going four years back from the closest quarter to the current date.

- It retrieves TTM data from the database (`ConsolidatedTTM`) for the last five years.

- New columns are added to the `quarterly_eps_list` to store TTM1, TTM2, and TTM3 EPS and Sales data, along with an 'EPS Rating.'

- The function iterates through each row in `quarterly_eps_list` to calculate the TTM1, TTM2, and TTM3 EPS and Sales growth rates:

- For each time period, it retrieves the previous year's quarter data for the same company.

- It calculates EPS and Sales growth rates using the formulas provided in the comments.

- The results are added to the respective columns in the DataFrame.

- Additionally, the 'EPS Rating' is calculated based on TTM1 EPS values, and a weighted value is added to the 'EPS Rating' column.

- The final `quarterly_eps_list` with the added columns is returned.

The code effectively calculates the required growth rates and ratings based on TTM data and quarterly EPS data. However, it can be improved by removing unnecessary comments and redundant code, which will enhance readability and maintainability.

- get_all_ttm

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function fetches and inserts TTM (Trailing Twelve Months) data into a database.

Operation:

- Calls the `ttm_quarterly_list` function to generate TTM data for the provided date `today` and the connection `conn`.

- Exports the generated TTM data to a table named "TTM" in the database.

- Inserts the TTM data into the database using the `insert_ttm` method.

Details:

- The `ttm_quarterly_list` function generates the TTM data.

- The `export_table` function is used to save the TTM data into a table for reference.

- The `insert_ttm` method is used to insert the TTM data into the database.

- get_all_consolidated_ttm

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function fetches and inserts consolidated TTM (Trailing Twelve Months) data into a database.

Operation:

- Calls the `consolidated_ttm_quarterly_list` function to generate consolidated TTM data for the provided date `today` and the connection `conn`.

- Inserts the consolidated TTM data into the database using the `insert_c_ttm` method.

Details:

- The `consolidated_ttm_quarterly_list` function generates consolidated TTM data.

- The `insert_c_ttm` method is used to insert the consolidated TTM data into the database.

- insert_ttm

Parameters:

- `self` (object)

- `ttm` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function inserts TTM (Trailing Twelve Months) data into the database from a Pandas DataFrame.

Operation:

- Converts specified columns in the DataFrame to numeric using `pd.to_numeric` with error handling for coercion.

- Exports the TTM data to a CSV file named "ttm_export.csv".

- Uses the PostgreSQL `COPY` command to insert data from the CSV file into the "TTM" table in the database.

- Commits the changes to the database.

Details:

- The code prepares the data for insertion by converting specific columns to numeric data types.

- It exports the data to a CSV file with appropriate formatting.

- The `COPY` command is used for efficient bulk insertion of data into the database.

- After insertion, the code commits the transaction to save the changes.

- The code has been commented out for removing the CSV file after insertion. It should be enabled if needed.

- insert_c_ttm

Parameters:

- `self` (object)

- `ttm` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function inserts consolidated TTM (Trailing Twelve Months) data into the database from a Pandas DataFrame.

Operation:

- Converts specified columns in the DataFrame to numeric using `pd.to_numeric` with error handling for coercion.

- Exports the consolidated TTM data to a CSV file named "consolidated_ttm_export.csv".

- Uses the PostgreSQL `COPY` command to insert data from the CSV file into the "ConsolidatedTTM" table in the database.

-Removes the CSV file after insertion.

Details:

- The code prepares the data for insertion by converting specific columns to numeric data types.

- It exports the data to a CSV file with appropriate formatting.

- The `COPY` command is used for efficient bulk insertion of data into the database.

- After insertion, the code removes the CSV file.

- get_ttm

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function retrieves TTM (Trailing Twelve Months) data from the database for the closest quarter to the current date.

Operation:

- Retrieves TTM data from the "TTM" table in the database.

- Filters the data for quarters up to the closest quarter to the current date.

Return:

- A Pandas DataFrame containing the TTM data for the closest quarter.

Details:

- The function fetches TTM data based on the closest quarter to the current date.

- The data is filtered based on the "YearEnding" column for quarters less than or equal to the closest quarter.

- The retrieved data is returned as a Pandas DataFrame.

- eps_rating

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the EPS Rating based on quarterly EPS growth data.

Operation:

- It calculates the EPS Rating for each entry in the `quarterly_eps_list` DataFrame using the formula:

EPS Rating = (0.2 * Q1 EPS Growth) + (0.2 * Q2 EPS Growth)

- It retrieves the Q1 EPS Growth and Q2 EPS Growth values from the DataFrame, and if any of them are missing (NaN or None), they are treated as 0.

- Calls the `ttm_eps_sales_growth` function to further calculate TTM EPS and sales growth.

- Returns the updated `quarterly_eps_list` with EPS Rating.

Details:

- The code calculates EPS Rating by taking 20% of Q1 and Q2 EPS Growth values for each entry.

- It handles missing or invalid values, treating them as 0.

- After EPS Rating calculation, it calls another function (`ttm_eps_sales_growth`) for further data processing.

- The function returns the modified `quarterly_eps_list` DataFrame.

- consolidated_eps_rating

Parameters:

- `self` (object)

- `c_quarterly_eps_list` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function calculates the consolidated EPS Rating based on quarterly EPS growth data.

Operation:

- It calculates the consolidated EPS Rating for each entry in the `c_quarterly_eps_list` DataFrame using the formula:

EPS Rating = (0.2 * Q1 EPS Growth) + (0.2 * Q2 EPS Growth)

- It retrieves the Q1 EPS Growth and Q2 EPS Growth values from the DataFrame, and if any of them are missing (NaN or None), they are treated as 0.

- Calls the `consolidated_ttm_eps_sales_growth` function to further calculate consolidated TTM EPS and sales growth.

- Returns the updated `c_quarterly_eps_list` with consolidated EPS Rating.

Details:

- The code calculates consolidated EPS Rating by taking 20% of Q1 and Q2 EPS Growth values for each entry.

- It handles missing or invalid values, treating them as 0.

- After EPS Rating calculation, it calls another function (`consolidated_ttm_eps_sales_growth`) for further data processing.

- The function returns the modified `c_quarterly_eps_list` DataFrame.

- stock_percentile_ranking

Parameters:

- `self` (object)

- `quarterly_eps_list` (Pandas DataFrame)

Description:

This function calculates the Stock Percentile Ranking based on EPS Rating.

Operation:

- It calculates the EPS Ranking by ranking the EPS Rating column in descending order.

- It then calculates the Stock Percentile Ranking for each entry based on the EPS Ranking and the total number of entries.

- Returns the `quarterly_eps_list` DataFrame with an added 'EPS Ranking' column.

Details:

- The function ranks entries based on the EPS Rating.

- It calculates the percentile ranking based on the EPS Ranking and the total number of entries in the DataFrame.

- The percentile ranking is a percentage that indicates the position of an entry relative to others in the list.

- The result is added to the DataFrame and returned.

- insert_EPS

Parameters:

- `self` (object)

- `name` (str)

- `table` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `cur` (mysql.connector.connection_cext.CMySQLCursor)

- `today` (datetime.date)

Description:

This function inserts EPS data into the database.

Operation:

- It prepares the table for insertion by adding a 'EPS Date' column with the current date.

- It handles missing or invalid values in the 'BSECode,' 'Months,' and 'Quarter' columns, converting them to NaN.

- It renames 'Sales' to 'Q1 Sales' and 'EPS' to 'Q1 EPS' in the DataFrame.

- It selects specific columns from the DataFrame to create a structured table.

- The table is exported to a CSV file and subsequently imported into the database table 'Reports.EPS'.

Details:

- The code ensures the data is ready for insertion by adding a timestamp and handling missing values.

- It renames columns for consistent structure.

- The data is saved to a CSV file before being loaded into the database.

- insert_Consolidated_EPS

Parameters:

- `self` (object)

- `name` (str)

- `table` (Pandas DataFrame)

- `conn` (mysql.connector.connection.MySQLConnection)

- `cur` (mysql.connector.connection_cext.CMySQLCursor)

- `today` (datetime.date)

Description:

This function inserts consolidated EPS data into the database.

Operation:

- It prepares the table for insertion by adding a 'EPS Date' column with the current date.

- It handles missing or invalid values in the 'BSECode,' 'Months,' and 'Quarter' columns, converting them to NaN.

- It renames 'Sales' to 'Q1 Sales' and 'EPS' to 'Q1 EPS' in the DataFrame.

- It selects specific columns from the DataFrame to create a structured table.

- The table is exported to a CSV file and subsequently imported into the database table 'Reports.Consolidated_EPS'.

Details:

- The code ensures the data is ready for insertion by adding a timestamp and handling missing values.

- It renames columns for consistent structure.

- The data is saved to a CSV file before being loaded into the database.

- get_eps_list

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function retrieves EPS data by merging data from BTTList and QuarterlyEPS tables for the first day of the current month.

Operation:

- It calculates the first day of the current month and the first day of the next month.

- It constructs a SQL query to join the 'BTTList' and 'QuarterlyEPS' tables based on the 'CompanyCode.'

- The data from 'BTTList' is joined with the latest EPS data for each company.

- The result is a merged DataFrame containing EPS data for the current month.

- Some columns are dropped or renamed for consistency.

Return:

Merged data of 'BTTList' and 'QuarterlyEPS' tables for the current month.

Details:

- This function retrieves and merges data from 'BTTList' and 'QuarterlyEPS' for the current month.

- It constructs a SQL query to perform the join, ensuring that the EPS data is the latest available.

- The merged DataFrame contains relevant information for further processing.

- get_consolidated_eps_list

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

- `today` (datetime.date)

Description:

This function retrieves consolidated EPS data by merging data from BTTList and ConsolidatedQuarterlyEPS tables for the first day of the current month.

Operation:

- It calculates the first day of the current month and the first day of the next month.

- It constructs a SQL query to join the 'BTTList' and 'ConsolidatedQuarterlyEPS' tables based on the 'CompanyCode.'

- The data from 'BTTList' is joined with the latest consolidated EPS data for each company.

- The result is a merged DataFrame containing consolidated EPS data for the current month.

- Some columns are dropped or renamed for consistency.

Return:

Merged data of 'BTTList' and 'ConsolidatedQuarterlyEPS' tables for the current month.

Details:

- This function retrieves and merges data from 'BTTList' and 'ConsolidatedQuarterlyEPS' for the current month.

- It constructs a SQL query to perform the join, ensuring that the consolidated EPS data is the latest available.

- The merged DataFrame contains relevant information for further processing.

- get_eps_list_history

Parameters:

- `self` (object)

- `conn` (mysql.connector.connection.MySQLConnection)

Description:

This function retrieves EPS data by merging data from BTTList and QuarterlyEPS tables for a specific historical period.

Operation:

- It calculates the first day of December 2018 and the first day of January 2019.

- It retrieves the closest quarter to the current date.

- It constructs a SQL query to join the 'BTTList' and 'QuarterlyEPS' tables based on the 'CompanyCode.'

- The data from 'BTTList' is joined with the latest EPS data available for each company within the specified historical period.

- The result is a merged DataFrame containing EPS data for the specified historical period.

- Some columns are dropped or renamed for consistency.

Return:

Merged data of 'BTTList' and 'QuarterlyEPS' tables for the specified historical period.

Details:

- This function retrieves and merges data from 'BTTList' and 'QuarterlyEPS' for a specific historical period.

- It constructs a SQL query to perform the join, ensuring that the EPS data is the latest available within the specified period.

- The merged DataFrame contains relevant information for further analysis or reporting.

Function `ttm_history_insert`

- **Description:** This function appears to be an unfinished or empty method. It is meant to fetch TTM (Trailing Twelve Months) data for one quarter. However, it lacks the necessary code to perform this operation.

Function `today_quarterly_eps`

- **Parameters:**

  - `conn` (mysql.connector.connection.MySQLConnection): Database connection object.

  - `cur` (cursor): Database cursor.

  - `today` (datetime.date): The current date.

- **Description:** This function generates data for today's quarterly EPS (Earnings Per Share). It compiles quarterly EPS data by fetching data from various sources and calculates growth metrics.

Function `consolidated_today_quarterly_eps`

- **Parameters:**

  - `conn` (mysql.connector.connection.MySQLConnection): Database connection object.

  - `cur` (cursor): Database cursor.

  - `today` (datetime.date): The current date.

- **Description:** This function is similar to `today_quarterly_eps` but focuses on consolidated quarterly EPS data. It fetches data and calculates growth metrics for consolidated EPS data.

Function `history_eps_report`

- **Parameters:**

  - `date` (datetime.date): The date for which the historical EPS report should be generated.

  - `conn` (mysql.connector.connection.MySQLConnection): Database connection object.

  - `cur` (cursor): Database cursor.

- **Description:** This function generates a historical EPS report for a specified date. It calculates EPS rating, stock percentile ranking, and EPS data for the historical period.

Function `current_eps_report`

- **Parameters:**
  - `conn` (mysql.connector.connection.MySQLConnection): Database connection object.
  - `cur` (cursor): Database cursor.
  - `today` (datetime.date): The current date.

- **Description:** This function generates the EPS report for the current date. It compiles and calculates EPS data, including EPS rating and stock percentile ranking. It distinguishes between regular and consolidated data based on the TTM3 value.

Function `export_table`

- **Parameters:**
  - `name` (str): The name of the table to export.
  - `table` (pandas.DataFrame): The DataFrame to export.

- **Description:** This function exports a DataFrame to a CSV file with a specified name.

Function `Generate_History_Reports`

- **Description:** This function appears to be incomplete and lacks a specific purpose. It attempts to generate history reports for EPS but doesn't specify the range or other details for the historical data.

Function `Generate_Daily_Report`

- **Parameters:**

  - `curr_date` (datetime.date): The current date.

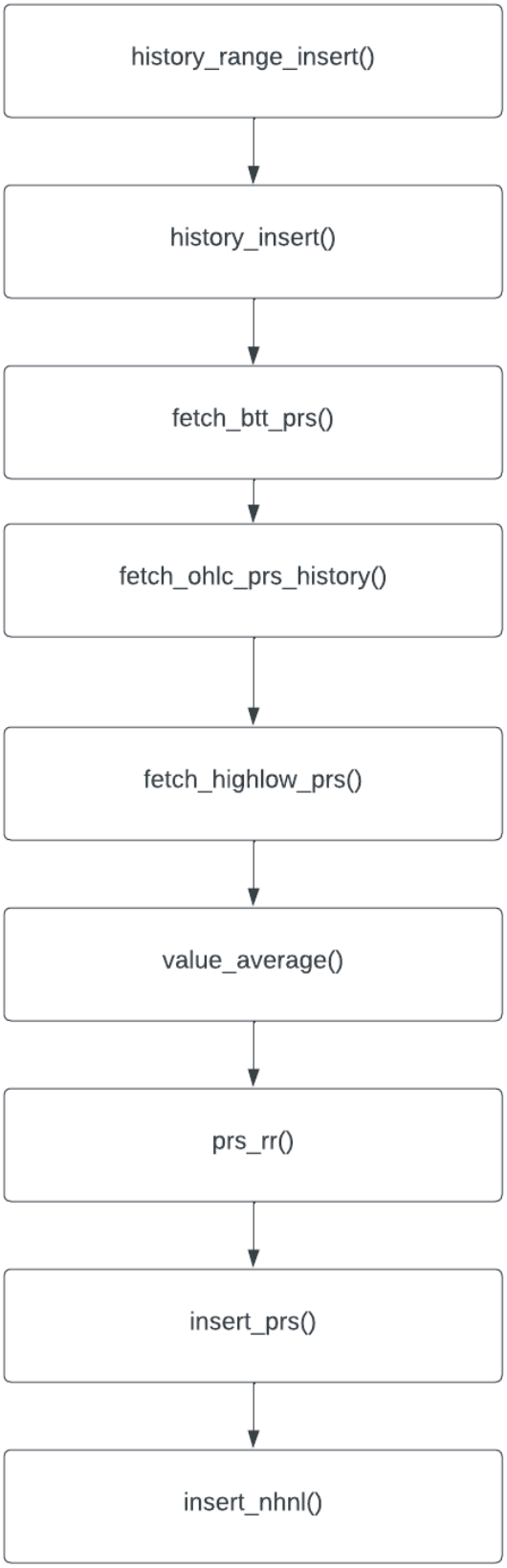
  - `conn` (mysql.connector.connection.MySQLConnection): Database connection object.

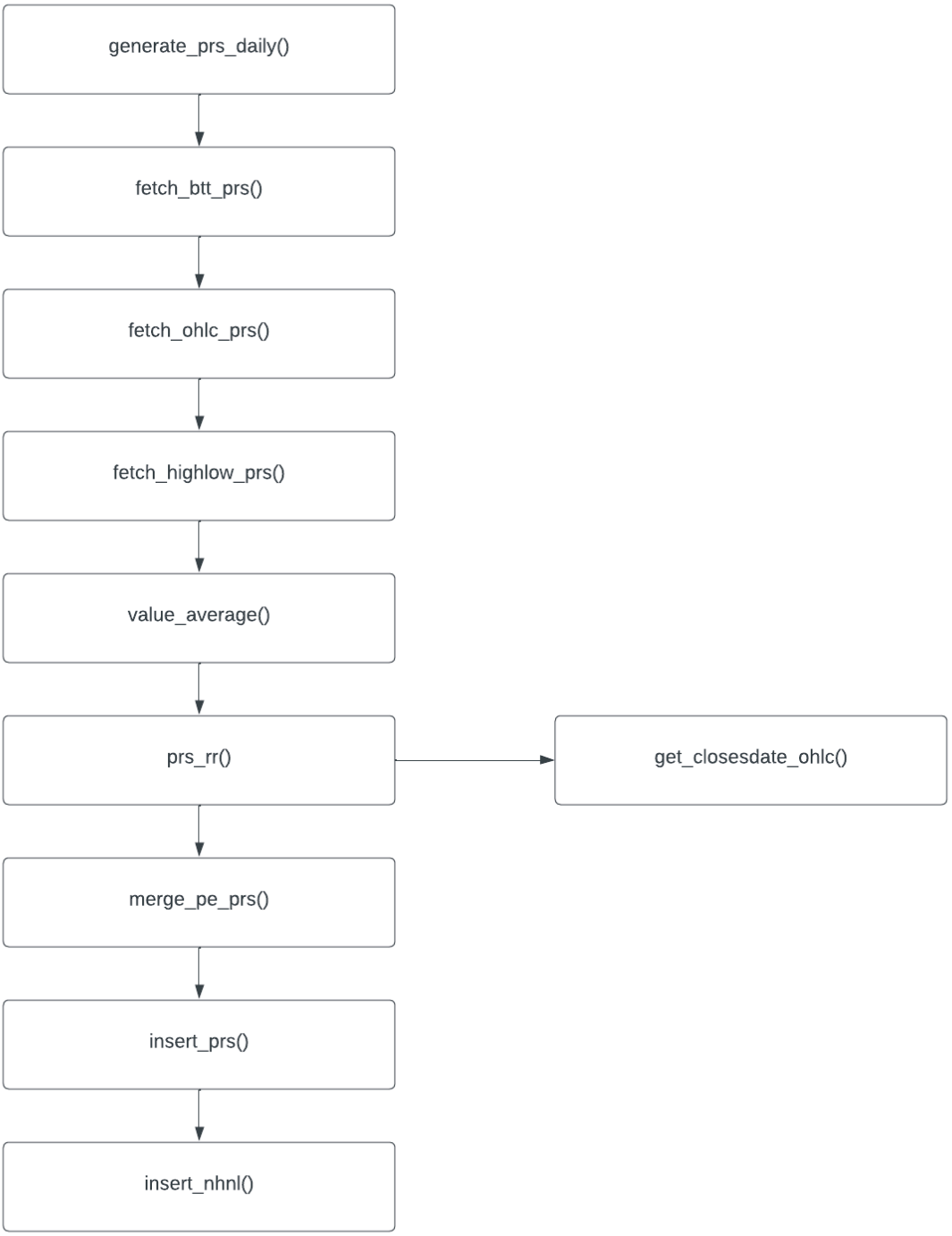
  - `cur` (cursor): Database cursor.

- **Description:** This function generates daily EPS reports for a given date. It calls the `current_eps_report` function for the current date.

Overall, the code contains functions for generating EPS reports for both historical and current data. However, there are some unfinished or empty methods, and the purpose and usage of some functions are not entirely clear. Further development and documentation are needed for a comprehensive understanding of the code's functionality.

PRS


history_range_insert()
history_insert()
fetch_btt_prs()
fetch_ohlc_prs_history()
fetch_highlow_prs()
value_average()
prs_rr()
insert_prs()
insert_nhnl()



Function Name: set_date(self, today_in=None)

Parameters:
- self (class): An instance of the class.
- today_in (datetime, optional): A datetime object representing a specific date. If not provided, it defaults to the current date.

Description:
This function is designed to set various date parameters, including the current date, a month back from the current date, and a year back from the current date. It checks for the optional input parameter `today_in` to set the current date. If `today_in` is not provided, it defaults to the current system date. It then computes the values for the current date, a month back, and a year back.

1. Step 1: Check the input `today_in` for None.
   - If `today_in` is None:
        - Set the variable `today` to the current system date without the time component.
   - If `today_in` is not None:
        - Set the variable `today` to the value of `today_in`.

2. Step 2: Convert dates to the desired format.
   - Set the variable `today_date` to the formatted string of the current date in the format "YYYY-MM-DD".
   - Compute the variable `month_back` by subtracting 30 days from the current date and formatting it in the "YYYY-MM-DD" format.
   - Compute the variable `year_back` by subtracting 366 days from the current date (considering leap years) and formatting it in the "YYYY-MM-DD" format.
   - Print the variable `year_back` and the current date.

Return:
None: The function does not explicitly return any value; it performs operations related to setting dates and printing them. The computed date values are stored in the global variables `today_date`, `month_back`, and `year_back`. The function does not return any specific value.

Function Name: fetch_btt_prs(self, curr_date, conn)

Parameters:
- self (class): An instance of the class.
- curr_date (datetime): A datetime object representing the current date from which data should be fetched.
- conn (database connection): A database connection used to execute a SQL query.

Description:
This function is responsible for fetching data from a database table named "BTTList" for a specific date range. The date range includes the beginning of the current month and the beginning of the next month. It uses the provided `curr_date` to determine this date range and executes a SQL query to retrieve the data.

1. Step 1: Calculate the start and end dates for the data retrieval.

- Set the variable `today` to the value of the `curr_date` parameter, representing the date from which data should be fetched.
- Calculate the variable `BTT_back` as the first day of the current month in the format "YYYY-MM-DD".
- Calculate the variables `next_month` and `next_year` to determine the first day of the next month. This handles the case when the next month is in a different year.
- Calculate the variable `BTT_next` as the first day of the next month in the format "YYYY-MM-DD".

2. Step 2: Construct and execute the SQL query.
- Build the SQL query string `btt_sql` to select distinct rows from the "BTTList" table where "BTTDate" falls within the date range specified by `BTT_back` and `BTT_next`.
- Execute the SQL query using the provided database connection `conn`.
- Store the result of the query in a DataFrame named `bttlist`.

Return:
- bttlist (DataFrame): A pandas DataFrame containing the fetched data from the "BTTList" table for the specified date range. The DataFrame includes columns such as "ISIN," "NSECode," "BSECode," "CompanyName," and "CompanyCode" for distinct rows within the specified date range. The function returns this DataFrame as its output.

Function Name: fetch_ohlc_prs(self, curr_date, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- curr_date (datetime): A datetime object representing the current date for which OHLC data should be fetched.
- bttlist (DataFrame): A pandas DataFrame containing data for the first of each month from the "BTTList" table.
- conn (database connection): A database connection used to execute a SQL query.

Description:
This function is responsible for fetching OHLC (Open, High, Low, Close) data from a database table named "OHLC" for a specific date. It matches the date provided as `curr_date` and retrieves OHLC data for that date. The function also performs some data manipulation to merge the OHLC data with the provided `bttlist` DataFrame.

1. Step 1: Convert `curr_date` to the desired date format.
- Set the variable `today_date` to the formatted string of the `curr_date` in the format "YYYY-MM-DD".

2. Step 2: Execute the SQL query to fetch OHLC data.
- Build the SQL query string `sql` to select all columns from the "OHLC" table where the "Date" matches the `today_date`.

- Execute the SQL query using the provided database connection `conn`.
- Store the result of the query in a DataFrame named `ohlc_company`.

3. Step 3: Merge the OHLC data with `bttlist`.
- Perform a left merge between the `bttlist` DataFrame and the `ohlc_company` DataFrame using the columns "NSECode" and "BSECode" as the merge keys.
- Rename columns in the merged DataFrame to match the expected column names by replacing "CompanyCode_x" with "CompanyCode" and "ISIN_x" with "ISIN".
- Remove columns "CompanyCode_y," "ISIN_y," and "Company" from the merged DataFrame.
- Fill missing values in the "Date" column with the `today_date`.

Return:
- bttlist (DataFrame): A pandas DataFrame containing the merged data of OHLC and `bttlist`. The merged DataFrame includes columns like "ISIN," "NSECode," "BSECode," "CompanyName," "CompanyCode," and OHLC data for the specified date. The function returns this DataFrame as its output.

Function Name: fetch_ohlc_prs_history(self, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- bttlist (DataFrame): A pandas DataFrame containing data from the "BTTList" table.
- conn (database connection): A database connection used to execute a SQL query.

Description:
This function is responsible for fetching historical OHLC (Open, High, Low, Close) data from a database table named "OHLC." It performs a left merge between the provided `bttlist` DataFrame and the OHLC data based on the "CompanyCode" column to retrieve historical OHLC data for companies listed in the "BTTList" table.

1. Step 1: Execute the SQL query to fetch historical OHLC data.
- Build the SQL query string `sql` to select all columns from the "OHLC" table where the "Date" matches the global variable `today_date` (which should have been set by a previous function).
- Execute the SQL query using the provided database connection `conn`.
- Store the result of the query in a DataFrame named `ohlc_company`.

2. Step 2: Merge the OHLC data with `bttlist`.
- Perform a left merge between the `bttlist` DataFrame and the `ohlc_company` DataFrame using the "CompanyCode" column as the merge key.
- Rename columns in the merged DataFrame to match the expected column names by replacing "NSECode_x" with "NSECode," "BSECode_x" with "BSECode," and "ISIN_x" with "ISIN."

- Remove columns "ISIN_y," "BSECode_y," and "Company" from the merged DataFrame.
- Fill missing values in the "Date" column with the global variable `today_date`.

Return:
- bttlist (DataFrame): A pandas DataFrame containing the merged data of OHLC and `bttlist` for historical data retrieval. The merged DataFrame includes columns like "ISIN," "NSECode," "BSECode," "CompanyName," "CompanyCode," and historical OHLC data. The function returns this DataFrame as its output.

Function Name: fetch_highlow_prs(self, curr_date, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- curr_date (datetime): A datetime object representing the current date for which the calculations are performed.
- bttlist (DataFrame): A pandas DataFrame containing merged data of OHLC and BTTList.
- conn (database connection): A database connection used to execute SQL queries.

Description:
This function calculates the values of "NewLow" and "NewHigh" for various time periods (52 weeks, 90 days, and 30 days) based on the provided OHLC and BTTList data. It iterates through the `bttlist` DataFrame, performs calculations, and updates the corresponding columns with the calculated values.

1. Step 1: Calculate date-related variables.
   - Set the variable `today` to the value of the `curr_date` parameter and ensure it has no time component.
   - Calculate `today_date`, `month_back`, and `year_back` based on the provided date and specified time intervals.

2. Step 2: Fetch historical OHLC data.
   - Build an SQL query to select OHLC data from the "OHLC" table for dates between `year_back` and `today_date`.
   - Execute the SQL query using the provided database connection `conn`.
   - Store the result in a DataFrame named `ohlc_full`.

3. Step 3: Calculate and update 52 Week High and Low.
   - Calculate the 52 Week High and Low for companies and NSE codes.
   - Iterate through the rows of `bttlist` and calculate the 52 Week High and Low for each company.

- Update the columns "52W High" and "52W Low" in the `bttlist` DataFrame with the calculated values and their corresponding dates.

4. Step 4: Calculate and update 52 Week NewHigh and NewLow.
   - Calculate the 52 Week NewHigh for companies and NSE codes.
   - Iterate through the rows of `bttlist` and calculate the 52 Week NewHigh for each company.
   - Update the column "52W NewHigh" in the `bttlist` DataFrame based on the comparison of the company's current "High" with the 52 Week High.

5. Step 5: Calculate and update 90 (65) Day NewHigh.
   - Filter the OHLC data for the last 90 days (or 65 days depending on the comment).
   - Calculate the 90 Day NewHigh for companies and NSE codes.
   - Iterate through the rows of `bttlist` and calculate the 90 Day NewHigh for each company.
   - Update the column "90D NewHigh" in the `bttlist` DataFrame based on the comparison of the company's current "High" with the 90 Day High.

6. Step 6: Calculate and update 65 (90) Day NewLow.
   - Calculate the 65 (90) Day NewLow for companies and NSE codes.
   - Iterate through the rows of `bttlist` and calculate the 65 (90) Day NewLow for each company.
   - Update the column "90D NewLow" in the `bttlist` DataFrame based on the comparison of the company's current "Low" with the 65 (90) Day Low.

7. Step 7: Calculate and update 20 Day NewHigh and NewLow.
   - Filter the OHLC data for the last 30 days.
   - Calculate the 20 Day NewHigh and NewLow for companies and NSE codes.
   - Iterate through the rows of `bttlist` and calculate the 20 Day NewHigh and NewLow for each company.
   - Update the columns "30D NewHigh" and "30D NewLow" in the `bttlist` DataFrame based on the comparisons of the company's current "High" and "Low" with the 20 Day High and Low.

Return:
- bttlist (DataFrame): A pandas DataFrame containing the updated data, including calculated values for "NewLow" and "NewHigh" for various time periods. The function returns this updated DataFrame as its output.

Function Name: value_average(self, curr_date, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- curr_date (datetime): A datetime object representing the current date for which the calculations are performed.
- bttlist (DataFrame): A pandas DataFrame containing data related to "NewLow" and "NewHigh" for various time periods.
- conn (database connection): A database connection used to execute SQL queries.

Description:
This function calculates the "Value Average" for a specified period based on the provided OHLC data and the `bttlist`. The Value Average is calculated by taking the average of the "Value" column for the most recent 50 data points (or fewer if less data is available) for each NSE code and Company code in the `bttlist`.

1. Step 1: Calculate date-related variables.
   - Set the variable `today` to the value of the `curr_date` parameter and ensure it has no time component.
   - Calculate `today_date` and `eightyday_back` based on the provided date and specified time intervals.

2. Step 2: Fetch OHLC data.
   - Build an SQL query to select OHLC data from the "OHLC" table for dates between `eightyday_back` and `today_date`.
   - Execute the SQL query using the provided database connection `conn`.
   - Store the result in a DataFrame named `ohlc_full`.

3. Step 3: Calculate and update Value Average.
   - Sort the `ohlc_full` DataFrame in descending order based on the "Date" column to have the most recent data first.
   - Iterate through the rows of the `bttlist` DataFrame.
   - For each row, filter the `ohlc_full` DataFrame to obtain data related to the NSE code and Company code.
   - Calculate the Value Average based on the "Value" column. If there are 50 or more data points, the average is calculated for the most recent 50 data points; otherwise, it's calculated for all available data points.
   - Update the "Value Average" column in the `bttlist` DataFrame with the calculated Value Average.

Return:
- bttlist (DataFrame): A pandas DataFrame containing the updated data, including the calculated "Value Average" for each NSE code and Company code in the `bttlist`. The function returns this updated DataFrame as its output.

The function takes the current date, the `bttlist` DataFrame, and a database connection as inputs, performs the Value Average calculation, and returns the updated DataFrame with the calculated values.

Function Name: getclosestdate_ohlc(self, companylist, s_date)

Parameters:
- self (class): An instance of the class.
- companylist (DataFrame): A pandas DataFrame containing data related to RR and OHLC.
- s_date (datetime): A specific date for which you want to retrieve the "Close" value.

Description:
This function is responsible for fetching the "Close" value for a given date. If the "Close" value for the exact date is not available in the `companylist`, the function returns the "Close" value for the closest available date based on a specific logic.

1. Step 1: Prepare the target date.
   - Transform the `s_date` parameter into a pure date value by removing the timestamp.
   - Convert the pure date value into a pandas Timestamp.

2. Step 2: Find the closest available date.
   - Check for data entries in the `companylist` DataFrame that match the target date and the following four dates: the target date itself, one day after, one day before, two days before, and three days before.
   - Determine the index of the `companylist` DataFrame.

3. Step 3: Return the "Close" value.
   - If there is data available for the exact target date in the `companylist`, return the "Close" value for that date.
   - If the exact date's data is not available, check for the closest available dates in the order of one day before, two days before, one day after, and three days before, and return the "Close" value for the closest available date.
   - If no data is found for any of these dates, return None to indicate that the "Close" value is not available for the specified date.

Return:
- close (float or None): The "Close" value for the specified date, or if it is not available, the "Close" value for the closest available date based on the logic described above. If no data is found for any of these dates, the function returns None.

Function Name: prs_rr(self, curr_date, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- curr_date (datetime): The current date for which RR and related metrics are calculated.
- bttlist (DataFrame): A pandas DataFrame containing data related to Value Average.
- conn: A database connection object used to query OHLC data.

Description:
This function calculates the Relative Rate of Change (RR) and related metrics for a set of stock data, which include RR for various time intervals (30 days, 60 days, 90 days, 365 days), change values, and off-high and off-low percentages. The RR is calculated as (current - previous) / previous * 100 for different time periods.

- Step 1: Prepare date values for the current date and back dates (30 days, 60 days, 90 days, 365 days).
- Step 2: Query OHLC data from the database for the specified date range.
- Step 3: Iterate over the rows of the `bttlist` DataFrame to calculate RR and related metrics.
- Step 4: Calculate RR for different time periods using the specified formulas.
- Step 5: Handle cases where RR data for the exact date is not available. It finds the closest available date's data.
- Step 6: Calculate additional metrics such as Change30, Change90, and Change52W.
- Step 7: Calculate OffHigh and OffLow percentages.
- Step 8: Update the `bttlist` DataFrame with calculated metrics.
- Step 9: Calculate relative strength (RS) metrics for RR30, RR90, and RR52W.
- Step 10: Calculate a combined RS metric.
- Step 11: Return the `bttlist` DataFrame with all calculated metrics.

Return:
- bttlist (DataFrame): The `bttlist` DataFrame with calculated RR and related metrics for each stock, including RR for various time periods, change values, off-high and off-low percentages, RS metrics, and a combined RS metric.

Function Name: merge_pe_prs(self, bttlist, conn, date)

Parameters:
- self (class): An instance of the class.

- bttlist (DataFrame): A pandas DataFrame containing data related to Relative Rate of Change (RR).
- conn: A database connection object used to query the PE data.
- date (datetime): The date for which PE data should be fetched.

Description:
This function merges data from the `bttlist` DataFrame and the `PE` table by matching records based on the 'CompanyCode'. It fetches PE data from the `PE` table for the provided date and selects specific columns of interest. Then, it merges this PE data into the `bttlist` DataFrame using a left join, ensuring that all records from the `bttlist` are included in the result.

- Step 1: Query PE data from the database for the specified date.
- Step 2: Select specific columns of interest from the PE data.
- Step 3: Merge the selected PE data into the `bttlist` DataFrame based on the 'CompanyCode' column using a left join.
- Step 4: Return the merged DataFrame containing data from both the `bttlist` and the `PE` table.

Return:
- bttlist (DataFrame): The `bttlist` DataFrame with additional PE data merged based on the 'CompanyCode'. The result includes data related to RR and PE for each stock.

Function Name: export_table(self, name, table)

Parameters:
- self (class): An instance of the class.
- name (str): The base name for the export file (excluding the file extension).
- table (DataFrame): The pandas DataFrame that you want to export to a CSV file.

Description:
This function exports the provided DataFrame to a CSV file with specific formatting options.

- `name`: A string that defines the base name for the exported file. The actual file name will include this name along with "_export.csv" as the file extension.
- `table`: The DataFrame that you want to export.

The function performs the following steps:

1. Constructs the export file name by appending "_export.csv" to the provided name.
2. Opens the export file in write mode.
3. Exports the DataFrame to the opened file in CSV format.

- The `header` parameter is set to `True` to include column headers in the exported file.
- The `index` parameter is set to `False` to exclude the index column.
- The `float_format` parameter is set to "%.2f" to format floating-point numbers with two decimal places.
- The `line_terminator` parameter is set to "\r" to ensure line terminators are compatible with certain systems.
4. Closes the export file.

The function does not remove the file ("os.remove") as the commented-out code suggests. If you intend to remove the exported file, you can uncomment this line. Additionally, there is a commented-out print statement that you can uncomment to indicate that the CSV file has been exported.

Return:
None

This function is primarily responsible for exporting a DataFrame to a CSV file with the specified formatting options and file name.

Function Name: insert_nhnl(self, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- bttlist (DataFrame): A pandas DataFrame containing data related to New Low and New High.

Description:
This function calculates NHNL data for different time intervals (30D, 90D, and 52W) and exports it to a CSV file. The NHNL data is then inserted into the database table "Reports"."NewHighNewLow."

- Step 1: Create a pandas DataFrame "nhnl" to store NHNL data.
- Step 2: Calculate NHNL data for each time interval and store it in the "nhnl" DataFrame.
- Step 3: Export the "nhnl" DataFrame to a CSV file named "NHNL_export.csv" using the "export_table" function.
- Step 4: Open the CSV file for reading.
- Step 5: Copy the data from the CSV file to the "Reports"."NewHighNewLow" table in the database.
- Step 6: Commit the transaction.
- Step 7: Remove the CSV file "NHNL_export.csv."

Return:
None

The "NHNL_export.csv" file is used to temporarily store the NHNL data before copying it to the database. Once the data is copied, the file is removed. Make sure the database table "Reports"."NewHighNewLow" exists and has the appropriate schema for data insertion.

Function Name: insert_prs(self, bttlist, conn)

Parameters:
- self (class): An instance of the class.
- bttlist (DataFrame): The pandas DataFrame containing the PE and RR data to be inserted into the database.
- conn: The database connection object.

Description:
This function inserts PRS data (PE Ratio and Relative Rate of Change) into the database.

- `bttlist`: The input DataFrame that contains the PRS data.
- `conn`: The database connection object used for database operations.

The function performs the following steps:

1. Adjusts data types and handles missing values in the input DataFrame:
   - It fills missing values in the 'Volume' column with 0 and converts the 'Volume' column to an integer data type.
   - It handles the 'BSECode' and 'Market Cap Rank' columns, ensuring that they are correctly typed and that missing values are appropriately replaced with NaN.

2. Constructs the export file name as "PRS_export.csv" and opens the export file in write mode.

3. Exports the input DataFrame (bttlist) to the opened file in CSV format:
   - The `header` parameter is set to `True` to include column headers in the exported file.
   - The `index` parameter is set to `False` to exclude the index column.
   - The `float_format` parameter is set to "%.2f" to format floating-point numbers with two decimal places.
   - The `line_terminator` parameter is set to "\r" to ensure line terminators are compatible with certain systems.

4. Closes the export file.

5. Defines a `copy_sql` statement for copying the data from the export file into the "PRS" table in the database using the PostgreSQL `COPY` command.

6. Opens the export file for reading.

7. Executes the `COPY` operation using the defined `copy_sql` statement and the contents of the export file.

8. Commits the changes to the database.

9. Closes the export file and removes it from the local file system.

10. There is a commented-out print statement that you can uncomment to indicate that the PRS data has been inserted.

Return:
None

This function is responsible for handling the data types, exporting the data to a CSV file, and then inserting the data into the "PRS" table in the database.

Function Name: history_range_insert(self, Dates)

Parameters:
- self (class): An instance of the class.
- Dates (list): A list of dates for which history data needs to be inserted.

Description:
This function iterates over the list of dates and performs history data insertion for each date using the `history_insert` function. It logs the progress to a text file named "PRS_History_Log.txt".

- `self`: An instance of the class.
- `Dates`: A list of date objects for which history data needs to be inserted.

The function performs the following steps:

1. Backup the standard output (`stdout`) using `stdout_backup`.

2. Print a message to indicate the start of history data insertion.

3. Iterate over each date (`date_in`) in the `Dates` list.

4. Inside the loop, the function does the following:
   - Prints a separator line and a message indicating the start of history data insertion for the current date.
   - Redirects the standard output to the "PRS_History_Log.txt" file using `sys.stdout`.
   - Repeats the same message to the log file.
   - Calls the `history_insert` function to insert history data for the current date.
   - Prints a separator line to the log file.

5. After processing all the dates, the standard output is reset to its original value using `stdout_backup`.

6. The function prints a message to indicate the completion of the insert operation.

7. A final separator line is printed to the standard output.

Return:
None

This function provides a structured way to insert history data for a range of dates and log the progress. It uses the `sys.stdout` redirection to capture the log messages in a file for each date, which can be useful for tracking and debugging the history insertion process.

Please note that to use this function, you should ensure that you have imported the `sys` module for working with the standard output and redirection.

Function Name: history_insert(self, today_in)

Parameters:
- self (class): An instance of the class.
- today_in (date): The date for which historical data is being inserted.

Description:
This function fetches and processes various data for a specific date, calculates relevant metrics, and inserts the historical data into the database. It performs the following steps:

1. Establish a database connection using the `DB_Helper` class and retrieve a database cursor (`conn`).

2. Set the current date to `today_in`.

3. Fetch BTTList data for the current date using the `fetch_btt_prs` function and the established database connection.

4. Create additional columns in the BTTList DataFrame without any data. These columns are used to store calculated metrics.

5. Fetch OHLC (Open, High, Low, Close) data for the current date using the `fetch_ohlc_prs_history` function and update the BTTList DataFrame.

6. Check if there is no Close data available for the current date. If so, log a message indicating that OHLC data was not found and raise a `ValueError`.

7. Calculate high, low, and Value Average based on the OHLC data for the current date using the `fetch_highlow_prs` and `value_average` functions.

8. Calculate the Relative Rate of Change (RR) for the current date using the `prs_rr` function.

9. Round the values in the BTTList DataFrame to two decimal places.

10. Reorder the columns in the BTTList DataFrame to ensure the desired order.

11. Insert the historical data into the "PRS" table in the database using the `insert_prs` function.

12. Calculate the NewHigh-NewLow (NHNL) totals using the `insert_nhnl` function.

13. Close the database connection.

The `history_insert` function is responsible for processing and storing historical data for a specific date. It performs a series of data retrieval and calculation steps before inserting the data into the database.

It's worth noting that this function works with a database, assumes that database connections and functions like `fetch_btt_prs`, `fetch_ohlc_prs_history`, `fetch_highlow_prs`, `value_average`, and `insert_prs` are defined elsewhere in your code.

Function Name: generate_prs_daily(self, curr_date, conn, cur)

Parameters:
- self (class): An instance of the class.
- curr_date (date): The current date for which PRS data is generated.
- conn: Database connection object.
- cur: Database cursor.

Description:
This function fetches and processes data from various sources, calculates relevant metrics, and generates PRS data for the specified date. Here are the key steps performed by this function:

1. Set the current date to `curr_date` using the `set_date` method.

2. Fetch BTTList data for the current date (`curr_date`) using the `fetch_btt_prs` function and the established database connection.

3. Create additional columns in the BTTList DataFrame without any data. These columns are used to store calculated metrics related to NewHigh, NewLow, and Rate Of Change (RR).

4. Fetch OHLC (Open, High, Low, Close) data for the current date using the `fetch_ohlc_prs` function and update the BTTList DataFrame with this data.

5. Check if there is no Close data available for the current date. If all Close values are null, log a message indicating that OHLC data was not found and raise a `ValueError`.

6. Calculate high, low, and Value Average metrics based on the OHLC data for the current date using the `fetch_highlow_prs` and `value_average` functions.

7. Calculate the Relative Rate of Change (RR) for the current date using the `prs_rr` function.

8. Merge the BTTList data with Price Earnings (PE) data for the current date using the `merge_pe_prs` function.

9. Round the values in the BTTList DataFrame to two decimal places.

10. Reorder the columns in the BTTList DataFrame to ensure the desired order.

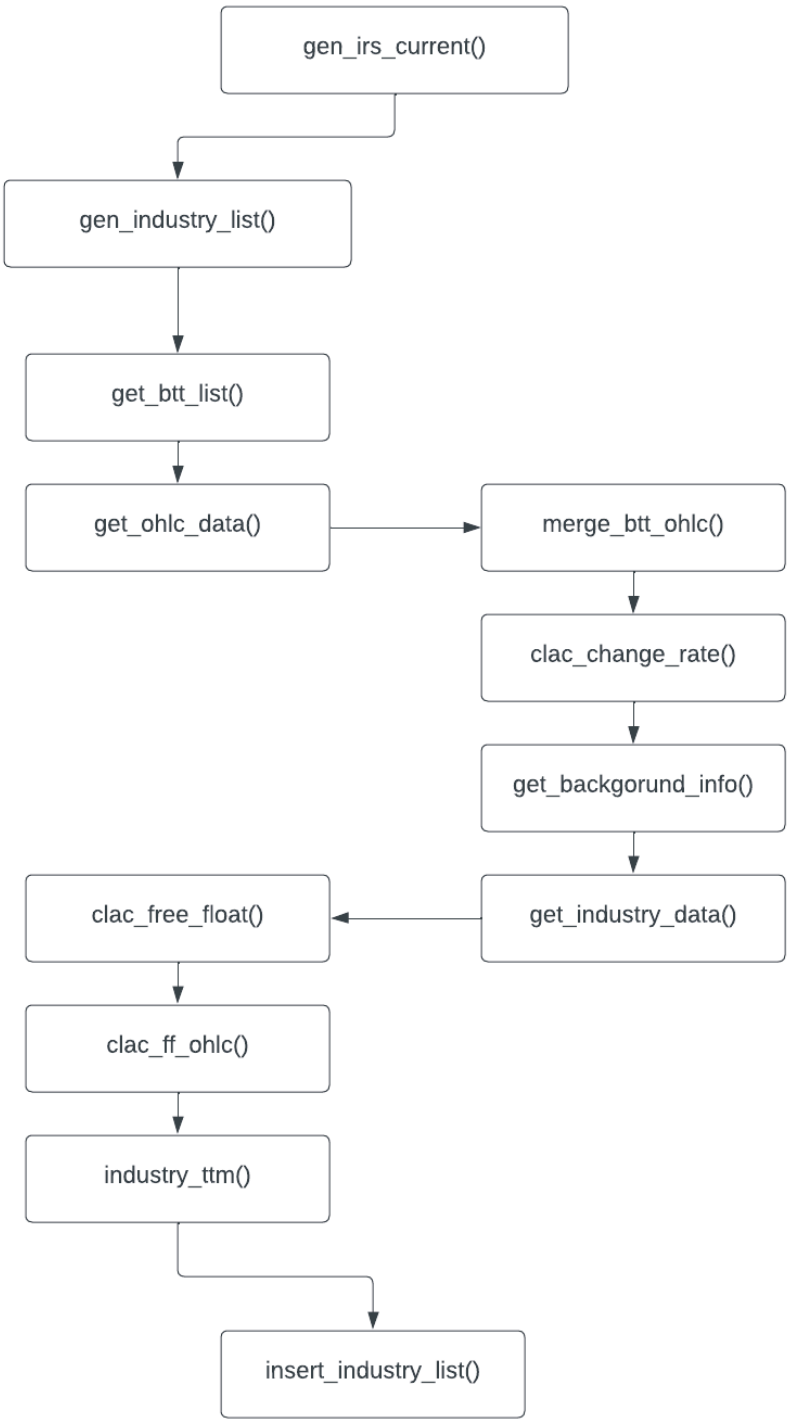
11. Insert the generated PRS data into the "PRS" table in the database using the `insert_prs` function.

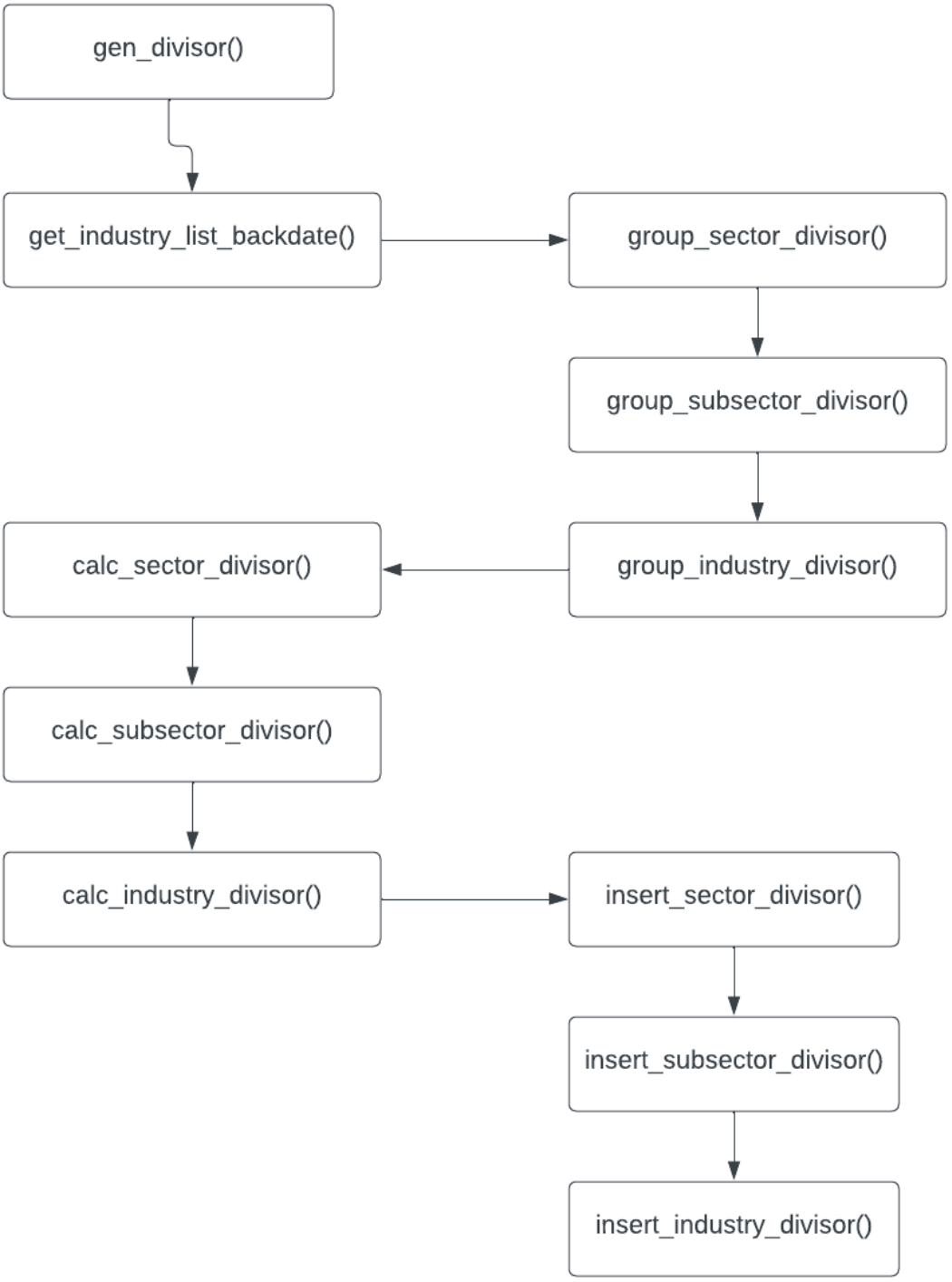
12. Calculate the NewHigh-NewLow (NHNL) totals using the `insert_nhnl` function.

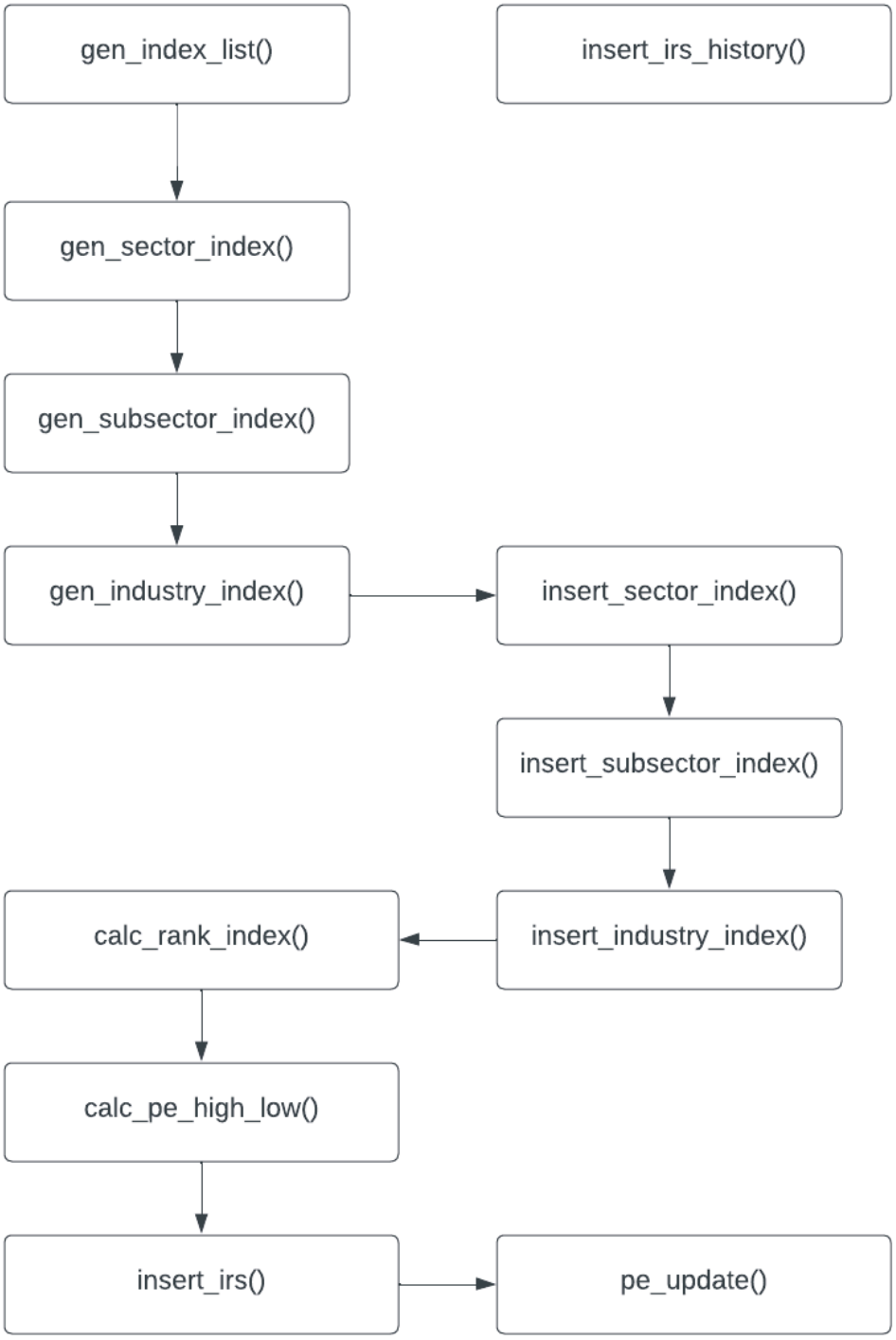
13. The function concludes by printing a separator line to indicate the end of the process.

The `generate_prs_daily` function performs data processing and metric calculation to generate PRS data for a specific date. It assumes the availability of functions like `fetch_btt_prs`, `fetch_ohlc_prs`, `fetch_highlow_prs`, `value_average`, `prs_rr`, `merge_pe_prs`, `insert_prs`, and `insert_nhnl` that are defined elsewhere in your code.

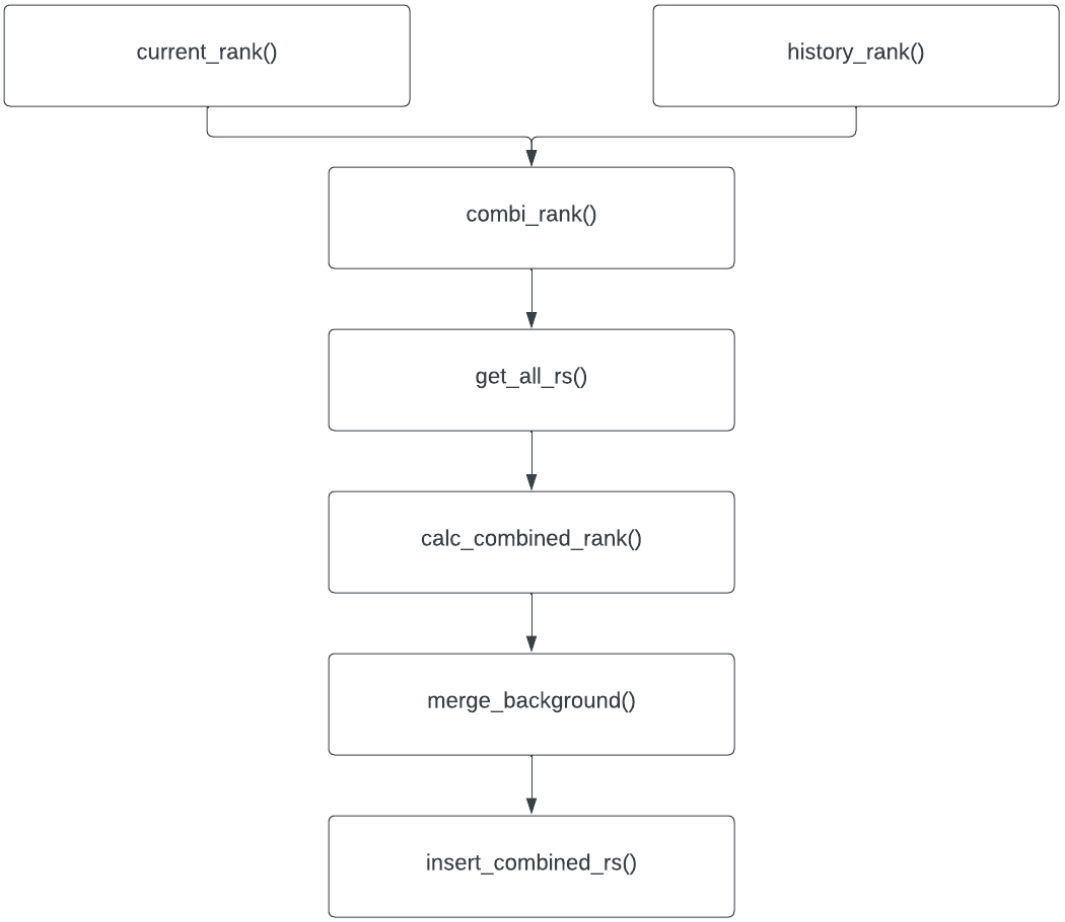
IRS


gen_irs_current()
gen_industry_list()
get_btt_list()
get_ohlc_data()
merge_btt_ohlc()
clac_change_rate()
get_backgorund_info()
get_industry_data()
clac_free_float()
clac_ff_ohlc()
industry_ttm()
insert_industry_list()

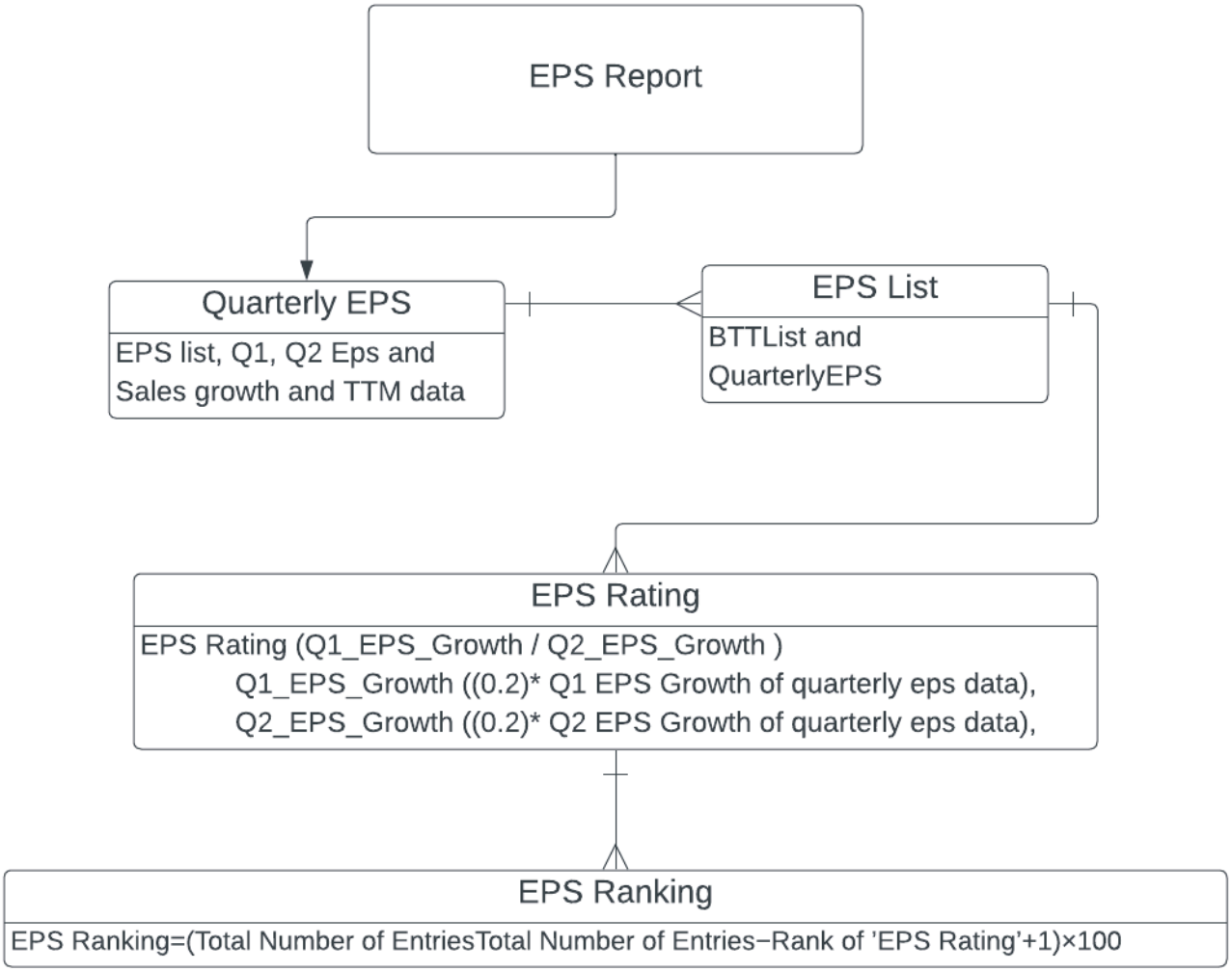

gen_divisor()
get_industry_list_backdate()
group_sector_divisor()
group_subsector_divisor()
group_industry_divisor()
calc_sector_divisor()
calc_subsector_divisor()
calc_industry_divisor()
insert_sector_divisor()
insert_subsector_divisor()
insert_industry_divisor()

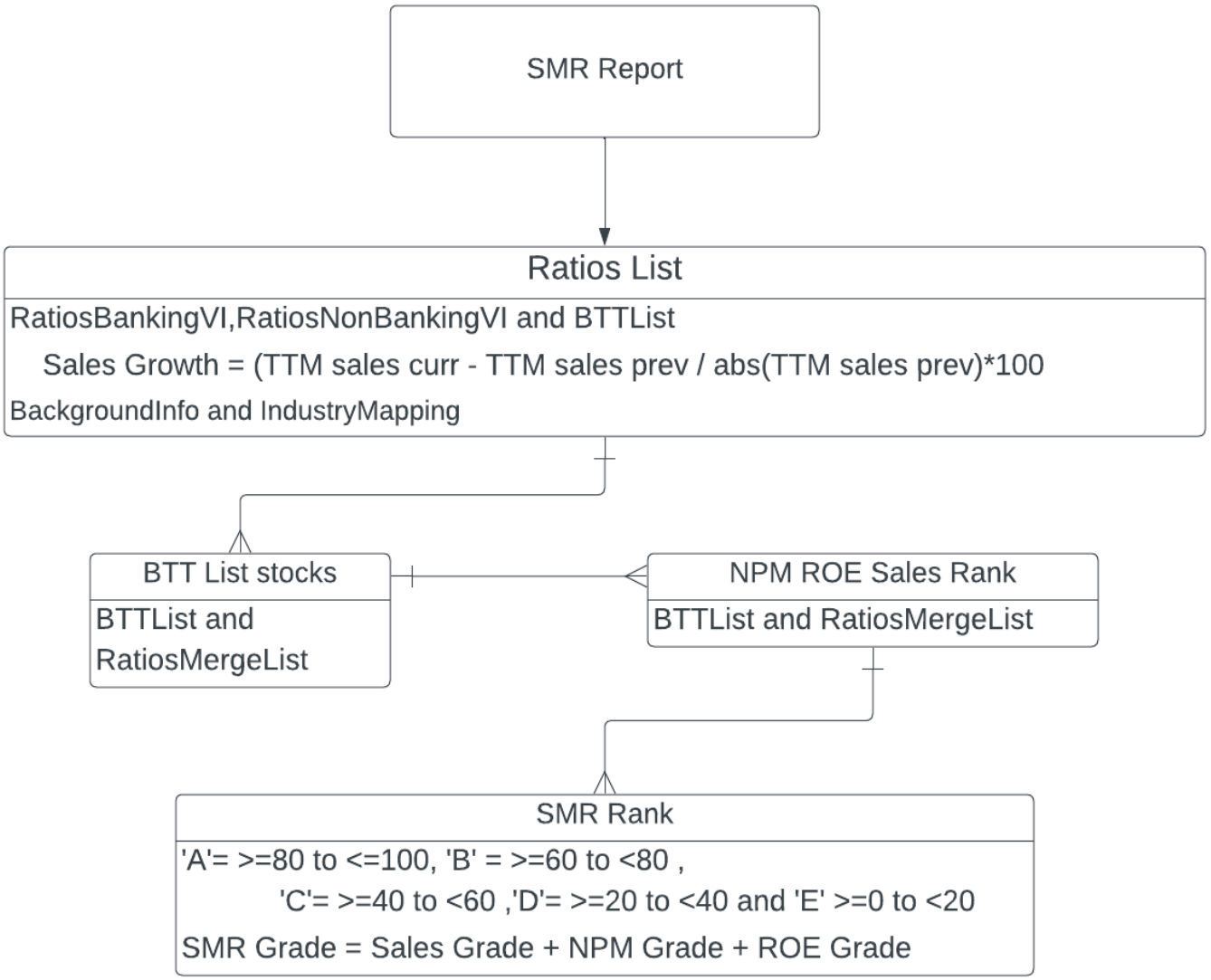

gen_index_list()
insert_irs_history()
gen_sector_index()
gen_subsector_index()
gen_industry_index()
insert_sector_index()
insert_subsector_index()
insert_industry_index()
calc_rank_index()
calc_pe_high_low()
insert_irs()
pe_update()

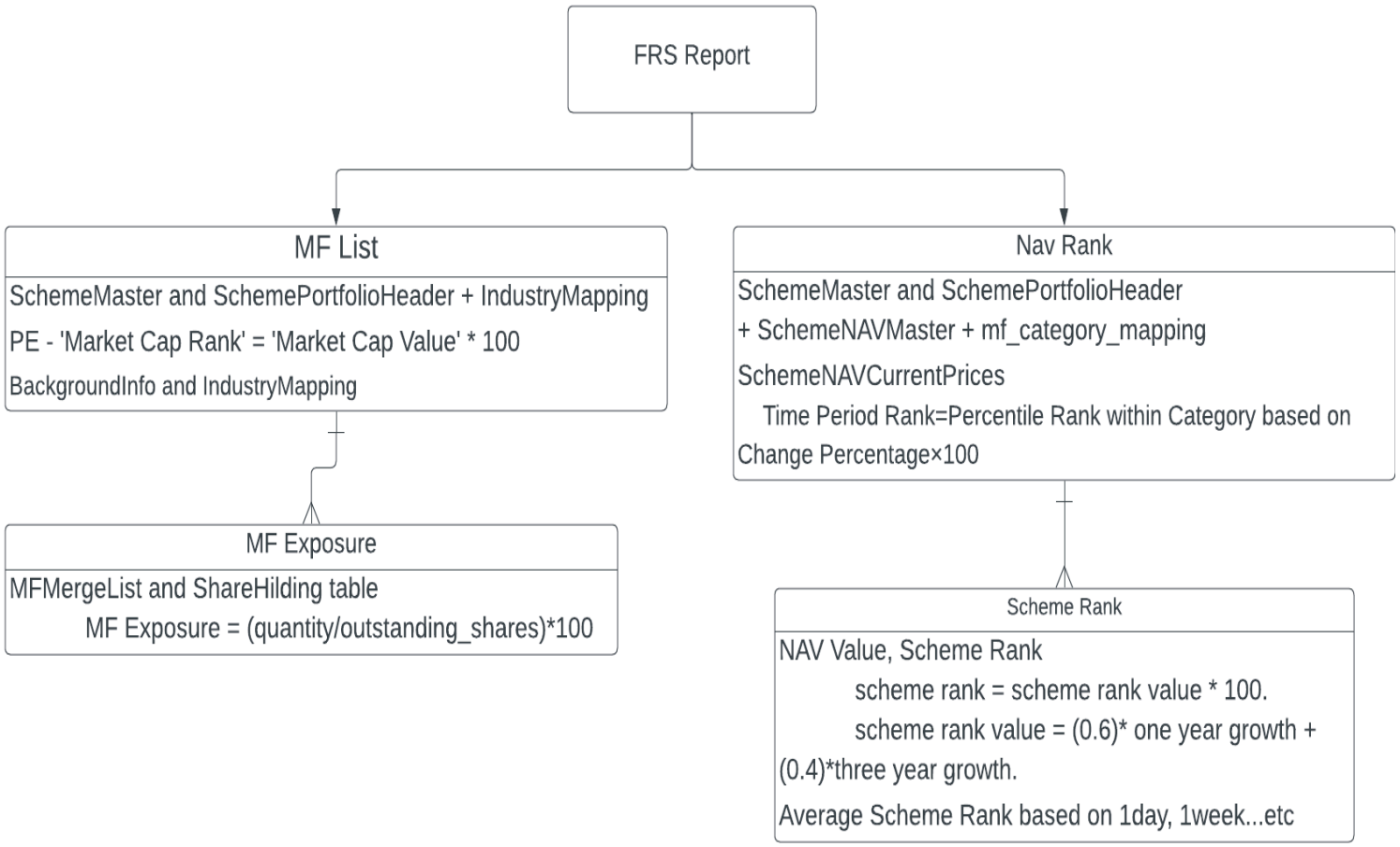
Combined_rank

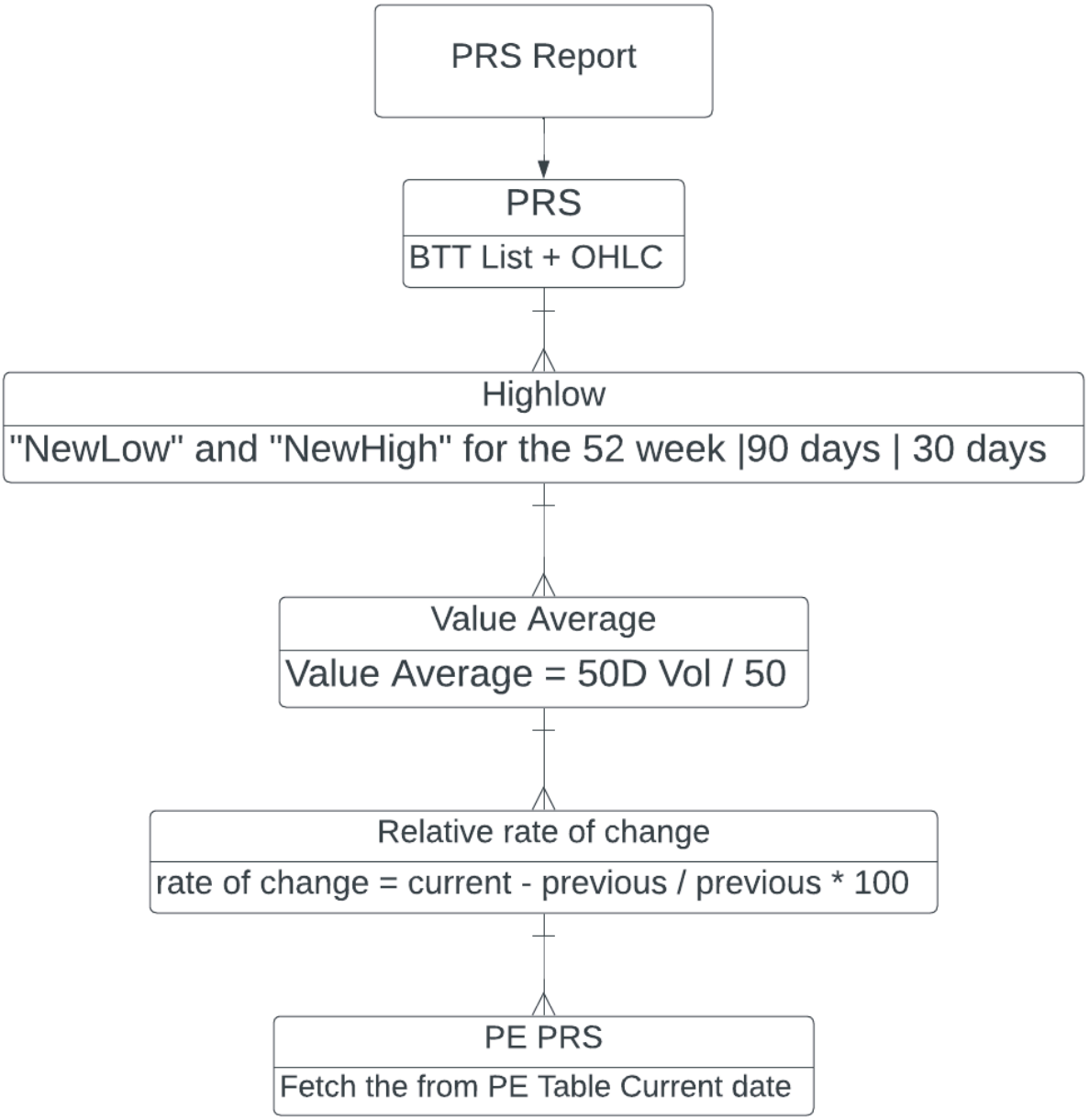


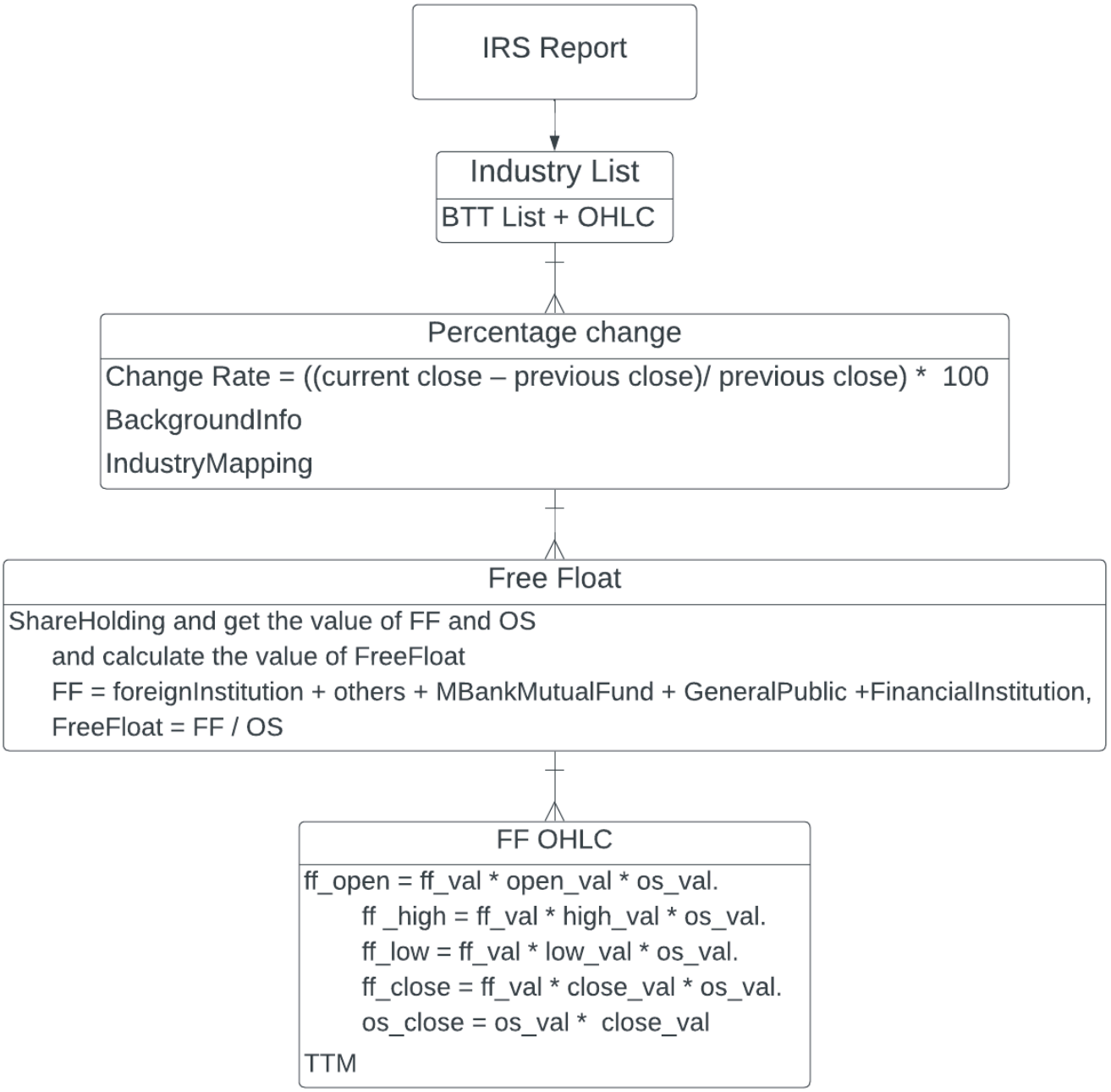
Process flows

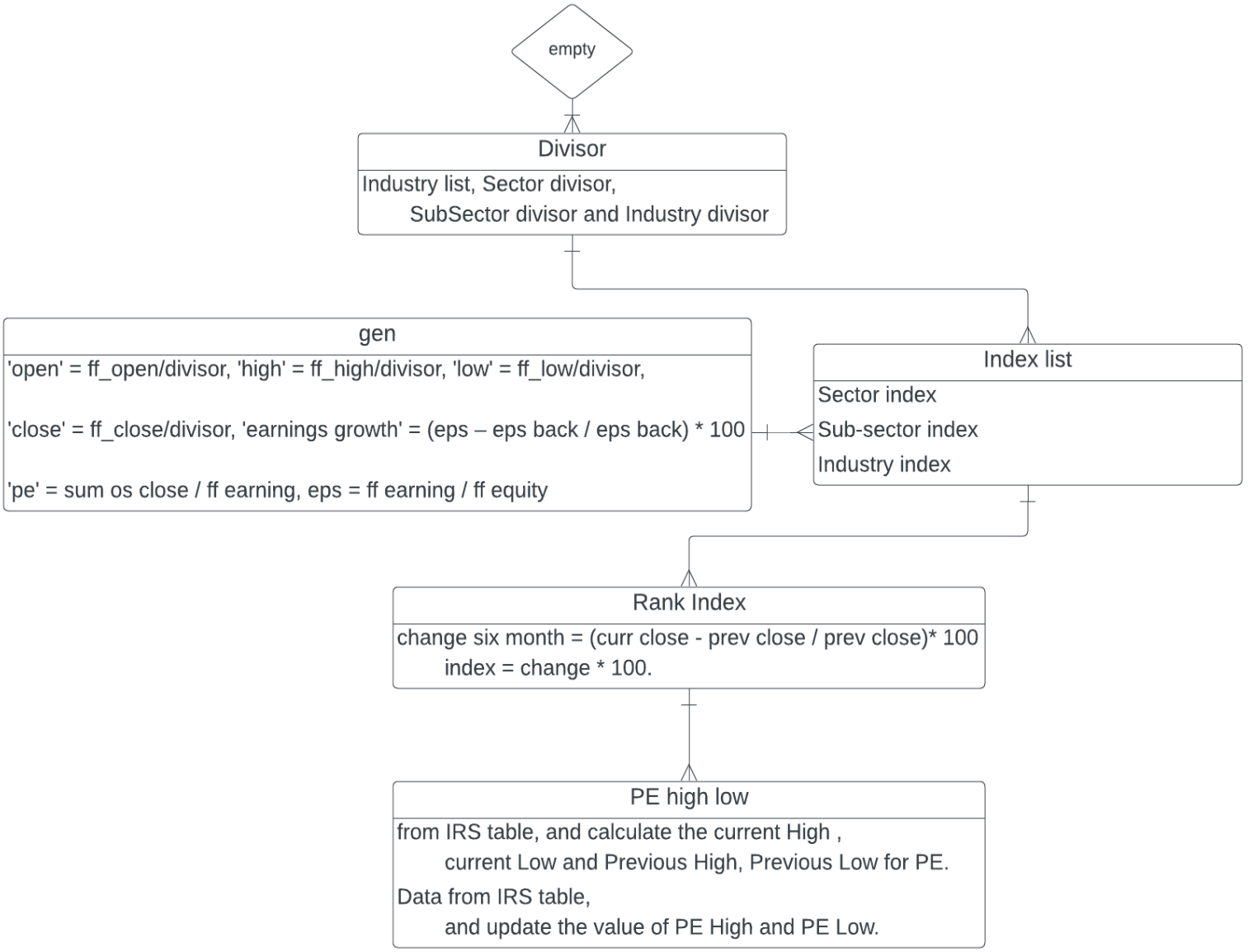

EPS Report
Quarterly EPS
EPS list, Q1, Q2 Eps and
Sales growth and TTM data
EPS List
BTTList and
QuarterlyEPS
EPS Rating
EPS Rating (Q1_EPS_Growth / Q2_EPS_Growth)
Q1_EPS_Growth ((0.2)* Q1 EPS Growth of quarterly eps data),
Q2_EPS_Growth ((0.2)* Q2 EPS Growth of quarterly eps data),
EPS Ranking
EPS Ranking=(Total Number of EntriesTotal Number of Entries−Rank of 'EPS Rating'+1)×100

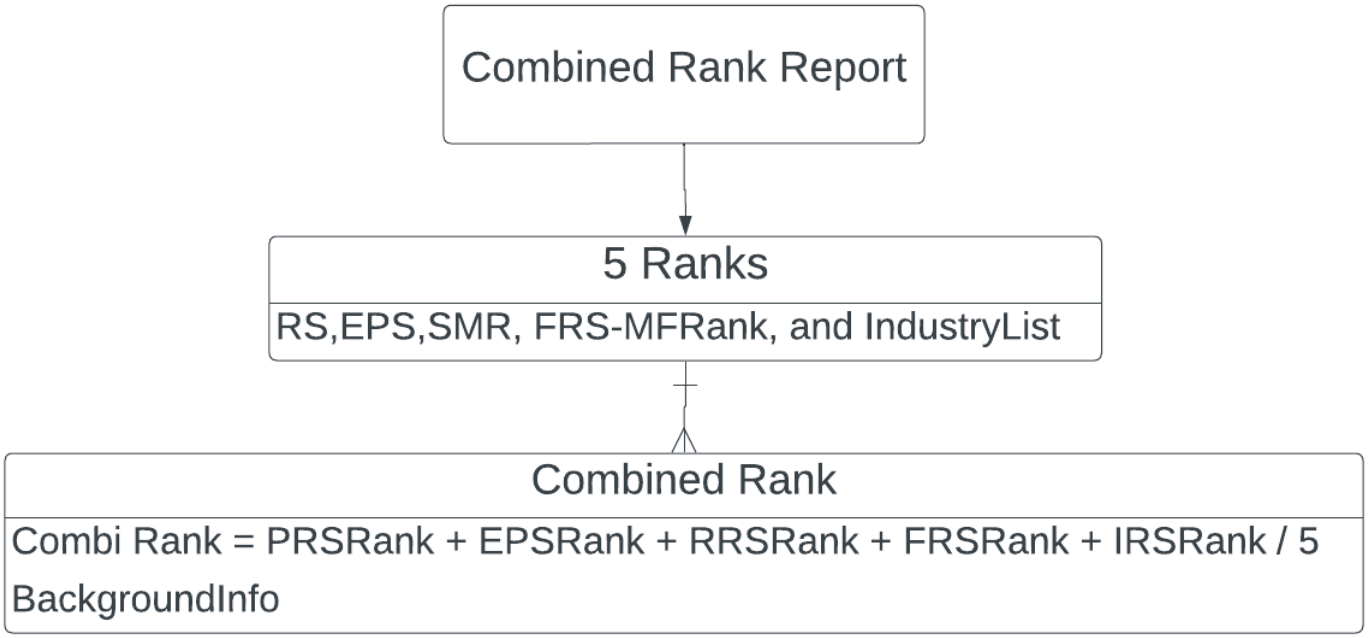

SMR Report
Ratios List
RatiosBankingVI,RatiosNonBankingVI and BTTList
Sales Growth = (TTM sales curr - TTM sales prev / abs(TTM sales prev)*100
BackgroundInfo and IndustryMapping
BTT List stocks
BTTList and RatiosMergeList
NPM ROE Sales Rank
BTTList and RatiosMergeList
SMR Rank
'A'= >=80 to <=100, 'B' = >=60 to <80 ,
'C'= >=40 to <60 ,'D'= >=20 to <40 and 'E' >=0 to <20
SMR Grade = Sales Grade + NPM Grade + ROE Grade

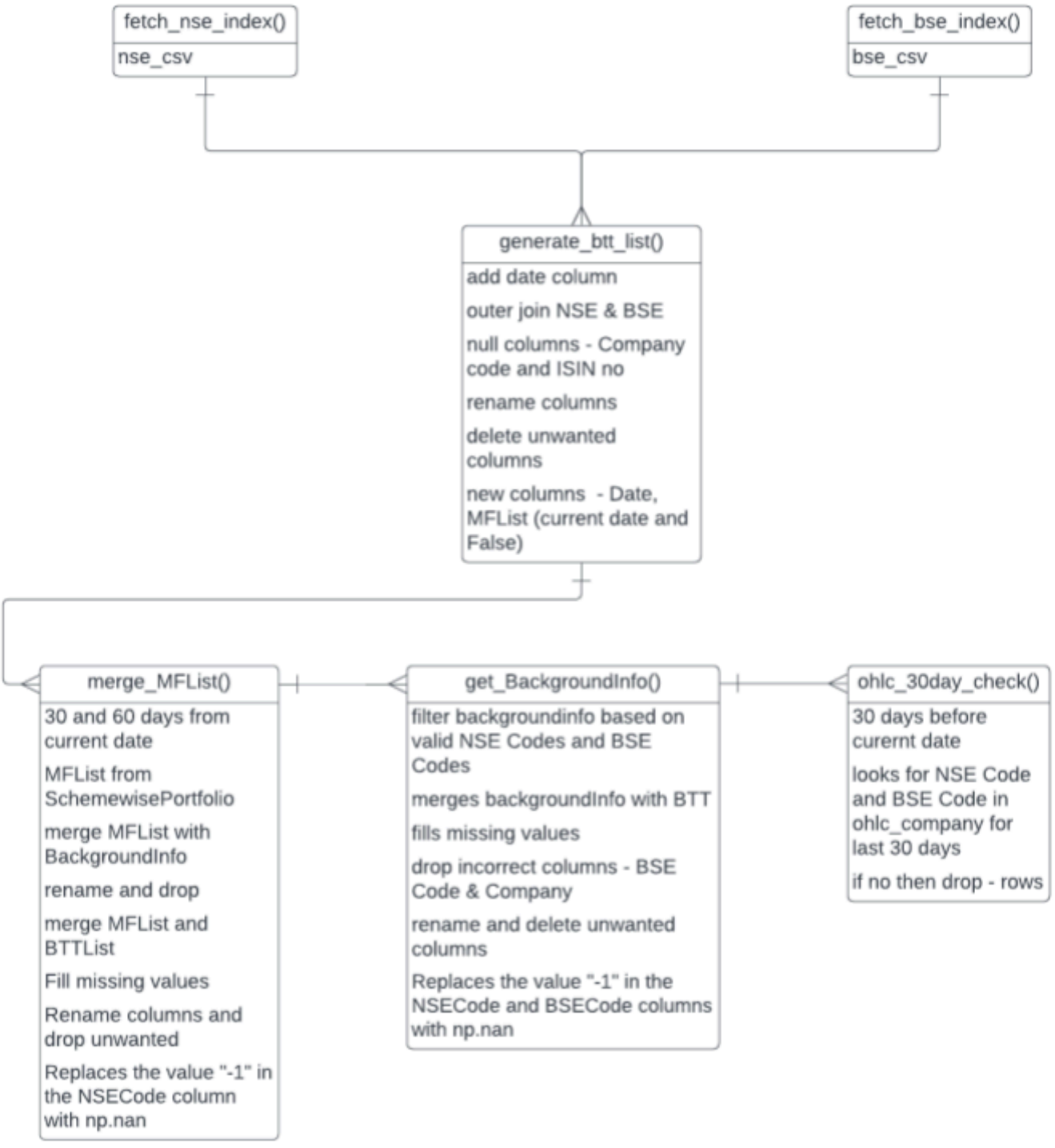

FRS Report
MF List
SchemeMaster and SchemePortfolioHeader + IndustryMapping
PE - 'Market Cap Rank' = 'Market Cap Value' * 100
BackgroundInfo and IndustryMapping
MF Exposure
MFMergeList and ShareHilding table
MF Exposure = (quantity/outstanding_shares)*100
Nav Rank
SchemeMaster and SchemePortfolioHeader
+ SchemeNAVMaster + mf_category_mapping
SchemeNAVCurrentPrices
Time Period Rank=Percentile Rank within Category based on Change Percentage×100
Scheme Rank
NAV Value, Scheme Rank
scheme rank = scheme rank value * 100.
scheme rank value = (0.6)* one year growth + (0.4)*three year growth.
Average Scheme Rank based on 1day, 1week...etc


PRS Report
PRS
BTT List + OHLC
Highlow
"NewLow" and "NewHigh" for the 52 week |90 days | 30 days
Value Average
Value Average = 50D Vol / 50
Relative rate of change
rate of change = current - previous / previous * 100
PE PRS
Fetch the from PE Table Current date


IRS Report
Industry List
BTT List + OHLC
Percentage change
Change Rate = ((current close – previous close)/ previous close) * 100
BackgroundInfo
IndustryMapping
Free Float
ShareHolding and get the value of FF and OS
and calculate the value of FreeFloat
FF = foreignInstitution + others + MBankMutualFund + GeneralPublic +FinancialInstitution,
FreeFloat = FF / OS
FF OHLC
ff_open = ff_val * open_val * os_val.
ff _high = ff_val * high_val * os_val.
ff_low = ff_val * low_val * os_val.
ff_close = ff_val * close_val * os_val.
os_close = os_val * close_val
TTM


empty
Divisor
Industry list, Sector divisor,
SubSector divisor and Industry divisor
gen
'open' = ff_open/divisor, 'high' = ff_high/divisor, 'low' = ff_low/divisor,
'close' = ff_close/divisor, 'earnings growth' = (eps – eps back / eps back) * 100
'pe' = sum os close / ff earning, eps = ff earning / ff equity
Index list
Sector index
Sub-sector index
Industry index
Rank Index
change six month = (curr close - prev close / prev close)* 100
index = change * 100.
PE high low
from IRS table, and calculate the current High ,
current Low and Previous High, Previous Low for PE.
Data from IRS table,
and update the value of PE High and PE Low.


Combined Rank Report
5 Ranks
RS,EPS,SMR, FRS-MFRank, and IndustryList
Combined Rank
Combi Rank = PRSRank + EPSRank + RRSRank + FRSRank + IRSRank / 5
BackgroundInfo


fetch_nse_index()
nse_csv
fetch_bse_index()
bse_csv
generate_btt_list()
add date column
outer join NSE & BSE
null columns - Company code and ISIN no
rename columns
delete unwanted columns
new columns - Date, MFList (current date and False)
merge_MFList()
30 and 60 days from current date
MFList from SchemewisePortfolio
merge MFList with BackgroundInfo
rename and drop
merge MFList and BTTList
Fill missing values
Rename columns and drop unwanted
Replaces the value "-1" in the NSECode column with np.nan
get_BackgroundInfo()
filter backgroundinfo based on valid NSE Codes and BSE Codes
merges backgroundInfo with BTT
fills missing values
drop incorrect columns - BSE Code & Company
rename and delete unwanted columns
Replaces the value "-1" in the NSECode and BSECode columns with np.nan
ohlc_30day_check()
30 days before curernt date
looks for NSE Code and BSE Code in ohlc_company for last 30 days
if no then drop - rows