Comparisons OpenLF Unicorn tools

Introduction

This pdf is created to explain the script named comparisons.py in path I:\Konstantinos Sidiropoulos\Documentation\Comparisons OpenLF_Unicorn.

The specified script creates a comparison between OpenLF's and Unicorn's daily LoadFlow reports. It calculates absolute and percentage differences (Unicorns variable is denominator) in current, active and reactive power between lines and boundary lines(X-lines). Also calculates absolute and percentage differences in voltage and theta between nodes and X-nodes. User can make daily or hourly comparisons depending on his needs.

Script was used in computer 10.91.100.15 with the following installations:

- Python version 3.12.0
- Visual Studio Cide: VSCodeUserSetup-x64-1.91.0.exe

1. Libraries and paths

The script:

➢ Imports the following libraries:

```
1. import pandas as pd
2. import numpy as np
3. from openpyxl import load_workbook
4. import os
```

Figure 1: Libraries import

- Pandas: Script uses pandas for reading, writing excel files and to do data manipulation.
- Numpy: Useful for handling special values in percentange results (infinite to nan)
- Load_workbook: Loads existing sheets in order to add headers in the final comparison results.
- Os: Useful to join folder paths with files.

➢ **def get_user_inputs()**:

timestamps = input("Enter the hours (comma-separated ex. 0030,0130 ... , or leave blank for default 0030-2330): "): Provides the possibility for user to specify the timestamps that he wants to compare the LoadFlow reports. If user does not specify timestamps, script considers all the timestamps from 0030 until 2330.

Also user specifies the path with the LoadFlow reports (**destination folder**) and the path where the comparisons excel is going to be saved (**destination folder1**). Fills the **'Date', 'File_type'**, and **'country_code'** variables with the correct information corresponding to the names of the LoadFlow reports. Numbers variable checks for versions of the LoadFlow reports and keeps the highest version for each hour (**numbers**).

```
1. def get_user_inputs():
2.     """
3.     Function to get user inputs for the comparison process.
```

```
    """
    # Prompt user to enter the necessary details
    timestamps = input("Enter the hours (comma-separated ex. 0030,0130 ... , or leave blank for default 0030-2330): ")
    if not timestamps:
        timestamps = ['0030', '0130', '0230', '0330', '0430', '0530', '0630', '0730', '0830', '0930', '1030', '1130','1230', '1330', '1430', '1530', '1630', '1730', '1830', '1930', '2030', '2130', '2230', '2330']
    else:
        timestamps = timestamps.split(',')

    destination_folder = input("Enter the base folder path where the load flows are located: ")
    destination_folder_1 = input("Enter the folder path where comparison results will be saved: ")
    Date = input("Enter the date (in YYYYMMDD format): ")
    File_type = input("Enter the file type (e.g., 'FO3'): ")
    country_code = input("Enter the country code (e.g., 'GR'): ")
    #Corresponds to different versions of loadflow reports
    numbers = range(0,15)

    return destination_folder, destination_folder_1, Date, File_type, country_code, numbers , timestamps

```

Figure 2: User inputs

2. Differencies and data loading:

def calculate_line_differencies(merged_df): Creates a threshold of 10^{-2} in order to clean Unicorn's and OpenLF's current, active and reactive power values less than 0.01 (Ampere, MW, MVAr) from data comparison. Merged_df variable contains both loadFlow reports (Unicorn's and OpenLF'S) of specific sheet (Nodes, Lines, X-nodes) for specific hour.

NOTE: apply(lambda x: 0 if abs(x) < threshold else x) function iterates through each value of the ' '_UNICORN and ''_OPENLF columns where ' ' corresponds to I,P,Q. Sets zero values less than 0,01.

I_diff, **P_diff, Q_diff** columns contain direct difference values between UNICORN and OPENLF for current (I), active power (P), and reactive power (Q) respectively.

It also computes the absolute differences **(I_diff_abs, P_diff_abs**, **Q_diff_abs**) and percentage differences **(I_diff_pct**, **P_diff_pct**, **Q_diff_pct**).

merged_df.replace([np.inf, -np.inf], np.nan, inplace=True): replaces infinite values with NAN.

Note: Infinite values are due to very small values of I, P, Q in Unicorn's and OPENLF reports. We replace those inf percentages to zero because we want to focus on the huge percentage differencies due to different modeling of the software's.(sos)

merged_df[columns] = merged_df[columns].fillna(0): Fills empty cells with zero values.

merged_df = merged_df[~(merged_df[columns_to_check] == 0).all(axis=1)]: Removes rows where Unicorn's and OpenLF's I, P, Q values are zero.

Returns merged_df variable with the modifications.

```
def calculate_line_differencies(merged_df):
    """
```

```
    Calculate the differences and percentage differences for line data (current, active power, reactive power) and cleans unnecessary data.
    """
    #Clean data from near to zero values.
    threshold = 1e-2
    merged_df[['I_UNICORN', 'I_OPENLF', 'P_UNICORN', 'P_OPENLF', 'Q_UNICORN', 'Q_OPENLF']] = merged_df[['I_UNICORN', 'I_OPENLF', 'P_UNICORN', 'P_OPENLF', 'Q_UNICORN', 'Q_OPENLF']].applymap(lambda x: 0 if abs(x) < threshold else x)

    merged_df['I_diff'] = merged_df['I_UNICORN'] - merged_df['I_OPENLF']
    merged_df['P_diff'] = merged_df['P_UNICORN'] - merged_df['P_OPENLF']
    merged_df['Q_diff'] = merged_df['Q_UNICORN'] - merged_df['Q_OPENLF']

    # Absolute differences
    merged_df['I_diff_abs'] = merged_df['I_diff'].abs()
    merged_df['P_diff_abs'] = merged_df['P_diff'].abs()
    merged_df['Q_diff_abs'] = merged_df['Q_diff'].abs()

    # Percentage differences
    merged_df['I_diff_pct'] = (merged_df['I_diff'].abs() / merged_df['I_UNICORN'].abs()) * 100
    merged_df['P_diff_pct'] = (merged_df['P_diff'].abs() / merged_df['P_UNICORN'].abs()) * 100
    merged_df['Q_diff_pct'] = (merged_df['Q_diff'].abs() / merged_df['Q_UNICORN'].abs()) * 100

    # CLEANING DATA FROM DIVISION WITH NEAR TO ZERO OR ZERO VALUES. Their respectively unicorns and openlf's values have absolute differenccies near to zero so instead of inf we will have zero values
    merged_df.replace([np.inf, -np.inf], np.nan, inplace=True)
    columns = ['I_UNICORN', 'I_OPENLF', 'P_UNICORN', 'P_OPENLF', 'Q_UNICORN', 'Q_OPENLF','I_diff_abs', 'P_diff_abs' ,'Q_diff_abs' ,'I_diff_pct', 'P_diff_pct', 'Q_diff_pct']
    # Fill empty cells with zero values
    merged_df[columns] = merged_df[columns].fillna(0)
    columns_to_check = ['I_UNICORN', 'I_OPENLF', 'P_UNICORN', 'P_OPENLF', 'Q_UNICORN', 'Q_OPENLF']
    # Drop rows where I,P,Q UNICORN'S AND OPENLF'S values are zero
    merged_df = merged_df[~(merged_df[columns_to_check] == 0).all(axis=1)]

    return merged_df

```

Figure 3: Line differencies()

def calculate_Nodes_differencies(merged_df):

Merged_df variable contains both loadFlow reports of Unicorn and OpenLF for specific hour.

U_diff, **theta_diff** columns contain direct difference values between UNICORN and OPENLF for voltage magnitude (U) and theta (theta) respectively.

It also computes the absolute differences **(U_diff_abs, theta_diff_abs**) and percentage differences **(U_diff_pct**, **theta_diff_pct**).

merged_df.replace([np.inf, -np.inf], np.nan, inplace=True): replaces infinite values with NAN.

merged_df[columns] = merged_df[columns].fillna(0): Fills empty cells with zero values.

Returns merged_df variable with the modifications.

```
def calculate_Nodes_differencies(merged_df):
    """
    Calculate the voltage absolute and percentage differences for nodes data (voltage magnitude and angle) and cleans unnecessary data.
    """
```

```
    # Calculate voltage magnitude (U) and angle (theta) differences
    merged_df['U_diff'] = merged_df['U_UNICORN'] - merged_df['U_OPENLF']
    merged_df['theta_diff'] = merged_df['theta_UNICORN'] - merged_df['theta_OPENLF']

    # Absolute differences
    merged_df['U_diff_abs'] = merged_df['U_diff'].abs()
    merged_df['theta_diff_abs'] = merged_df['theta_diff'].abs()

    # Percentage differences (absolute percentage difference relative to UNICORN)
    merged_df['U_diff_pct'] = (merged_df['U_diff'].abs() / merged_df['U_UNICORN'].abs()) * 100
    merged_df['theta_diff_pct'] = (merged_df['theta_diff'].abs() /
merged_df['theta_UNICORN'].abs()) * 100

    # Replace inf or NaN values due to division by zero
    merged_df.replace([np.inf, -np.inf], np.nan, inplace=True)
    merged_df.fillna(0, inplace=True)

    return merged_df

```

Figure 4: Nodes comparison

def load_data(filepath, sheet_name):

Script reads specific sheet of Unicorn's or OpenLF's LoadFlow reports and prints error message in case of non-existence of the sheet. Those sheets will be used for Lines or Nodes comparisons. For every hour, two variables (**df1**, **df2**) are used to load sheet's of Unicorn's and OpenLF's reports correspondingly.

```
def load_data(filepath, sheet_name):
    """
    Load Excel data from a specified sheet.
    """
    try:
        return pd.read_excel(filepath, sheet_name=sheet_name)
    except Exception as e:
        print(f"Error loading {sheet_name} from {filepath}: {e}")
        return None

```

Figure 5: Load data()

3. Sheets data structure and merging

def rename_lines_data(df1 , df2): rename specific column names of Unicorn's (**df1**) and OpenLF's (**df2**) line sheet for later merging based on the common column names. Keeps the first 19 characters in both reports 'id' columns. Returns the structured df1, df2 sheets.

```
def rename_lines_data(df1 , df2):
    df1.rename(columns={'Name (mrid)': 'id'}, inplace=True)
    df1.rename(columns={'Terminal number': 'side'}, inplace=True)
    df2.rename(columns={'side_x': 'side'}, inplace=True)
    df1['id'] = df1['id'].astype(str).str[:19]
    df2['id'] = df2['id'].astype(str).str[:19]

    return df1, df2

```

Figure 6: Lines renamed columns

def rename_X_lines_data(df1, df2): Same process as **rename_lines_data** including different columns renaming.

```
1. def rename_X_lines_data(df1, df2):
2.     df2.rename(columns={'BUS' : 'Bus' }, inplace=True)
3.     df1.columns = df1.columns.str.strip()
4.     df1.rename(columns={'Name (mrid)': 'id'}, inplace=True)
5.     df1['id'] = df1['id'].astype(str).str[:19]
6.     df1['Bus'] = df1['Bus'].astype(str).str[:8]
7.
8.     return df1,df2
9.
```

Figure 7: X-lines renamed columns

def rename_X_Nodes_data(df1 , df2):

process_id() function is created to read the id column of X-Nodes Sheet in OpenLF's (**df2**) LoadFlow reports.

If the id starts with 'X' then it keeps the first 8 characters for each row of the id column(**X-node id**).

If it's not then 'X' character will be the 9th character of the id. Then process_id() returns 9th until 17th character (**X-node id)**.

Note: id column of X-Node sheet has boundary line names (ex. BZANDV14_XKR_ZA11_9 or XRI_PE11_0RIBAR11_1).

Function renames specific columns for later merging and keeps in df2 'id' column the processed id's.

Returns structured df1, df2 variables.

```
 1. def rename_X_Nodes_data(df1 , df2):
 2.     def process_id(id_str):
 3.                 id_str = str(id_str)
 4.                 if id_str.startswith('X'):
 5.                     return id_str[:8]
 6.                 elif len(id_str) > 9 and id_str[9] == 'X':
 7.                     return id_str[9:17]
 8.                 return id_str
 9.
10.     df1['Name (mrid)'] = df1['Name (mrid)'].astype(str).str[:8]
11.     df1.rename(columns={'Name (mrid)': 'id'}, inplace=True)
12.     df2.rename(columns={'boundary_v_mag': 'U', 'boundary_v_angle': 'theta'}, inplace=True)
13.     df2['id'] = df2['id'].apply(process_id)
14.
15.     return df1, df2
16.
```

Figure 8: X-Nodes renamed columns

def rename_Nodes_data(df1, df2):

Renames specific columns in 'Bus' sheet of df1, df2 LoadFlow reports for later merging. Returns structured df1, df2.

```
1. def rename_Nodes_data(df1, df2):
2.     df1['Name (mrid)'] = df1['Name (mrid)'].astype(str).str[:8]
```

```
3.     df1.rename(columns={'Name (mrid)': 'Bus'}, inplace=True)
4.     df2.rename(columns={ 'BUS' : 'Bus', 'v_mag': 'U', 'v_angle': 'theta'}, inplace=True)
5.
```

Figure 9: Nodes columns renaming

def merge_common_data(df1, df2, merge_columns, sort_columns = None):

Function that creates **merged_df**.

Merge_columns and **sort_colums** variables contain a list of column names that are used as pairing key for merging Unicorn's and OpenLF's sheets and are sorted by ascending order (A to Z or smaller to larger).

common_ids = set(df1[merge_columns[0]]).intersection(set(df2[merge_columns[0]])): Saves a list with the common values of merge_columns first specified attribute.

df1 = df1[df1[merge_columns[0]].isin(common_ids)]: df1's first specified attribute will contain rows with only common_id's values. Same for df2.

EXAMPLE: If merge_columns['id', 'side'] then first specified attribute is 'id'. Script keeps common_id values of 'id' column and merges with those values and their common 'side' values.

Merged_df contains both sheets based on their common merged_columns. For same named columns '_Unicorn', '_OPENLF' suffixes are added.

```
 1. def merge_common_data(df1, df2, merge_columns, sort_columns = None):
 2.     """
 3.     Merge two DataFrames on common columns and sort them by columns provided.
 4.     """
 5.     common_ids = set(df1[merge_columns[0]]).intersection(set(df2[merge_columns[0]]))
 6.     df1 = df1[df1[merge_columns[0]].isin(common_ids)]
 7.     df2 = df2[df2[merge_columns[0]].isin(common_ids)]
 8.
 9.     merged_df = pd.merge(df1, df2, on=merge_columns, suffixes=('_UNICORN', '_OPENLF'))
10.     if sort_columns:
11.         merged_df.sort_values(by=sort_columns, ascending=[True] * len(sort_columns),
inplace=True)
12.     return merged_df
13.
```

Figure 10: merged_df

4. Final comparison sheets structure and paths generation

Script after merging specific sheets and calculating absolute and percentage differencies, changes the final data structure using:

def final_columns_rename_lines(merged_df , timestamp): Is used for lines/X-lines final sheets of merged_df. Returns renamed columns in **final_df** and a new 'timestamp' variable that contains the timestamp of LoadFlow reports comparison.

```
1. def final_columns_rename_lines(merged_df , timestamp):
2.     final_df = merged_df.rename(columns={
3.                 'I_UNICORN': 'I',
4.                 'I_OPENLF': 'I',
5.                 'P_UNICORN': 'P',
```

```
                'P_OPENLF': 'P',
                'Q_UNICORN' : 'Q',
                'Q_OPENLF' : 'Q',
                'I_diff_abs': 'I_diff_abs',
                'P_diff_abs': 'P_diff_abs',
                'Q_diff_abs': 'Q_diff_abs',
                'I_diff_pct': 'I_diff_pct',
                'P_diff_pct': 'P_diff_pct',
                'Q_diff_pct' :'Q_diff_pct',
            })
    final_df['Timestamp'] = timestamp
    return final_df , timestamp

```

Figure 11: Final data structure of Lines/X-lines

def final_columns_rename_buses(merged_df , timestamp): Same process as rename_lines for Nodes/X-nodes final sheet of merged_df. Returns structured **final_df** and timestamp variable.

```
def final_columns_rename_buses(merged_df , timestamp):
    final_df = merged_df.rename(columns={
                'U_UNICORN': 'U',
                'U_OPENLF': 'U',
                'theta_UNICORN': 'theta',
                'theta_OPENLF': 'theta',
                'U_diff_abs': 'U_diff_abs',
                'theta_diff_abs': 'theta_diff_abs',
                'U_diff_pct': 'U_diff_pct',
                'theta_diff_pct': 'theta_diff_pct'
            })

    final_df['Timestamp'] = timestamp
    final_df = final_df.dropna(subset=['U'])
    return final_df , timestamp

```

Figure 12: Final data structure of Nodes/X-Nodes

def make_adjustements_lines_to_excel(output_path, sheet_name): Adds headers to the final structured line/X-lines sheets.

Script finds in **output_path** the specified **sheet_name** using **load_workbook** function. Adds headers with specific order that correspond to specific columns. Saves the structured sheet again in output_path.

```
def make_adjustements_lines_to_excel(output_path, sheet_name):
    header_titles = [
        'ID', 'SIDE', 'UNICORN', 'UNICORN', 'UNICORN', 'OPENLF', 'OPENLF', 'OPENLF',
        'ABSOLUTE DIFFERENCES', 'ABSOLUTE DIFFERENCES', 'ABSOLUTE DIFFERENCES',
        'PERCENTAGE DIFFERENCES', 'PERCENTAGE DIFFERENCES', 'PERCENTAGE DIFFERENCES',
'TIMESTAMP'
    ]

    # Open the workbook and the relevant sheet
    wb = load_workbook(output_path)
    ws = wb[sheet_name]

    # Insert custom headers
    ws.insert_rows(1)  # Insert an empty row at the top
    for col, value in enumerate(header_titles, start=1):
        ws.cell(row=1, column=col, value=value)  # Write the headers

    # Save the workbook with the updated headers
```

```
    wb.save(output_path)

```

Figure 13: Headers Lines/X-lines addition

def make_adjustements_nodes_to_excel(output_path, sheet_name): Similar process with previous function for Nodes/X-Nodes sheets.

```
def make_adjustements_nodes_to_excel(output_path, sheet_name):
    header_titles = ['ID', 'UNICORN', 'UNICORN', 'OPENLF', 'OPENLF', 'ABSOLUTE DIFFERENCES', 'ABSOLUTE DIFFERENCES', 'PERCENTAGE DIFFERENCES' , 'PERCENTAGE DIFFERENCES'  , 'TIMESTAMP']
    # Open the workbook and the relevant sheet
    wb = load_workbook(output_path)
    ws = wb[sheet_name]

    ws.insert_rows(1)
    for col, value in enumerate(header_titles, start=1):
        ws.cell(row=1, column=col, value=value)

    wb.save(output_path)

```

Figure 14: Headers Nodes/X-Nodes addition

def generate_file_paths(timestamp, number, Date, File_type, country_code, destination_folder, destination_folder_1): Function returns the paths (destination folder & filenames) to Unicorn's and OpenLF's LoadFlow reports and prints error messages in case reports are not found.

Note: Name structures of Unicorn's and OpenLF's reports are not always stable. For the specified project the filenames had the follow structure:
{Date}_{timestamp}_{File_type}_{country_code}_{number}_igmLfReport.xlsx.
{Date}_{timestamp}_{File_type}_{country_code}_0_OPENLF_REPORT.xlsx. (sos)

```
def generate_file_paths(timestamp, number, Date, File_type, country_code, destination_folder):
    # Define the paths for the input files
    df1_path = os.path.join(destination_folder, f'{Date}_{timestamp}_{File_type}_{country_code}_{number}_igmLfReport.xlsx') ####sos USER HAS TO FILL THE RIGHT NAME STRUCTURE OF UNICORN'S LOAD FLOW REPORTS
    df2_path = os.path.join(destination_folder, f'{Date}_{timestamp}_{File_type}_{country_code}_0_OPENLF_REPORT.xlsx') ###sos USER HAS TO FILL THE RIGHT NAME STRUCTURE OF OPENLF'S LOAD FLOW REPORTS
    print(f"Generated df1_path: {df1_path}")
    print(f"Generated df2_path: {df2_path}")
    # Check if both files exist
    if os.path.exists(df1_path) and os.path.exists(df2_path):

        return df1_path, df2_path
    else:
        # Notify the user about the missing files and offer guidance
        missing_files = []
        if not os.path.exists(df1_path):
            missing_files.append(f"'{df1_path}'")
        if not os.path.exists(df2_path):
            missing_files.append(f"'{df2_path}'")

        print(f"Warning: The following expected files were not found:\n{', '.join(missing_files)}")
        print("Please ensure the filenames match the expected pattern or adjust the filenames accordingly.")
        # Return None to indicate missing files
        return None, None
```

```
23.
```

Figure 15: Paths specification

5. Comparison excel creation

def find_highest_version_number(Date, timestamp, numbers, File_type, country_code , destination_folder):): function iterates through possible version numbers to identify the highest version of Unicorn's filename for a specific timestamp. Checks if each filename for each number from 0 to 15 exists, and returns the highest version number.

```
 1. def find_highest_version_number(Date, timestamp, numbers, File_type, country_code ,
destination_folder):
 2.     highest_number = -1
 3.
 4.     for number in numbers:
 5.         report_filename = os.path.join(destination_folder,
f'{Date}_{timestamp}_{File_type}_{country_code}_{number}_igmLfReport.xlsx')
 6.
 7.         if os.path.exists(report_filename):
 8.             if number > highest_number:
 9.                 highest_number = number
10.
11.     return highest_number
12.
```

Figure 16: Highest version

def process_files_and_accumulate_data(timestamps, numbers, Date, File_type, country_code, destination_folder, destination_folder_1): Combines all the predefined functions to in order to create a final structured dataframe that contains the comparisons of LoadFlow reports in a range of one day.

combined_output_path contains the folder and name of the final excel comparison file.

All_sheets_data creates a dictionary with specific names corresponding to the names of the final excel sheets.

for timestamp in timestamps: Loop iterates through a list of user specified timestamps. For each timestamp keeps the highest version number for Unicorn's LoadFlow reports and generates the paths to both Unicorn's and OpenLF's reports.

If function checks if LoadFlow reports exist to proceed with the comparisons. Else script continues to the next timestamp.

```
 1. def process_files_and_accumulate_data(timestamps, numbers, Date, File_type, country_code,
destination_folder, destination_folder_1):
 2.     # Use a single output Excel file for all timestamps
 3.     combined_output_path = os.path.join(destination_folder_1,
f'combined_results_OpenLF_Unicorn_{Date}.xlsx')
 4.     # Create dictionaries to store data for each category across all timestamps
 5.     all_sheets_data = {'Lines': [], 'X-lines': [], 'Nodes': [], 'X-Nodes': []}
 6.
 7.     for timestamp in timestamps:
 8.         number = find_highest_version_number(Date, timestamp, numbers, File_type, country_code,
destination_folder)
```

```
        # Generate file paths
        df1_path, df2_path = generate_file_paths(timestamp, number, Date, File_type,
country_code, destination_folder)
            # Check if the paths exist, continue to the next iteration if they don't
        if df1_path and df2_path and os.path.exists(df1_path) and os.path.exists(df2_path):

```

Figure 17: Loop creation

Lines sheet: Script loads Line sheet of Unicorn's and OpenLF's specified timestamp LoadFlow reports (**df1, df2**). Renames both sheets essential for merging columns. Merges sheets based on their common 'id' and 'side' columns and sorts rows by ascending form. Calculates line differencies (I, P, Q), drops unessential columns and renames the final data structure adding also column that contains the current timestamp. Drops rows where all columns except 'Timestamp' and 'Bus' contain zero values and saves data to Lines dictionary.

```
#Lines
                df1 = load_data(df1_path , 'Line')
                df2 = load_data(df2_path , 'Line')
                df1, df2 = rename_lines_data(df1, df2)
                merged_df = merge_common_data(df1, df2, merge_columns=['id', 'side'],
sort_columns=['id', 'side'])
                merged_df = calculate_line_differencies(merged_df)
                columns_to_drop = ['Terminal number', 'Bus ' ,'BUS', 'v_mag', 'v_angle',
'I_limit' , 'Area' ,  'Island number' , 'U' ,'theta',  'Base Voltage' , 'U' ,'theta',
'Bus',   'Imax' , 'loading' , 'Eq. type' , 'State' , 'r' , 'x' , 'side_x' , 'element_type' ,
'side_y' , 'name' , 'type' , 'value' , 'acceptable_duration' , 'I_diff' , 'P_diff' , 'Q_diff']
                merged_df.drop(columns=[col for col in columns_to_drop if col in
merged_df.columns] , inplace= True)
                final_df, timestamp = final_columns_rename_lines(merged_df, timestamp)
                # Drop rows where all columns except 'Timestamp' , 'Bus' , 'id' contain zeros
                columns_to_check = final_df.columns.difference(['Timestamp', 'side', 'id'])
                final_df = final_df.loc[~(final_df[columns_to_check] == 0).all(axis=1)]
                all_sheets_data['Lines'].append(final_df)

```

Figure 18: Lines Comparison sheet

X-Lines sheet: Script loads Line sheet from Unicorn's LoadFlow report (**df1**) and X-Nodes sheet from OpenLF's LoadFlow report (**df2**) for specified timestamp. Renames both sheet columns that are essential for merging. Merges sheets based on their common 'id', 'Bus' columns and sorts their rows by ascending form. Calculates boundary line differencies (I, P, Q), drops unnecessary columns and renames the final data structure including the addition of current timestamp column. Drops rows where all values except 'Timestamp', 'Bus', 'id' are zero and appends the final data to X-lines dictionary.

```
                #X-lines
                df1 = load_data(df1_path, 'Line')
                df2 = load_data(df2_path, 'X-Nodes')
                df1, df2 = rename_X_lines_data(df1, df2)
                merged_df = merge_common_data(df1, df2, merge_columns=['id', 'Bus'] ,
sort_columns=['id', 'Bus'])
                merged_df = calculate_line_differencies(merged_df)
                columns_to_drop = ['Area' , 'Terminal number' , 'v_angle' , 'I_limit' ,'Island
number' , 'U' , 'v_mag' , 'v_angle' 'theta' , 'Base Voltage' , 'U' ,'theta',   'side_x' , 'Imax' ,
'Unnamed: 0','loading' , 'Eq. type' , 'State' , 'r' , 'x'  , 'element_type' , 'side_y' , 'name' ,
'type' , 'value' , 'acceptable_duration' , 'I_diff' , 'P_diff' , 'Q_diff', 'boundary_v_mag' ,
'boundary_v_angle' , 'boundary_p' , 'boundary_q']
```

```
 8.                merged_df.drop(columns=[col for col in columns_to_drop if col in
merged_df.columns] , inplace= True)
 9.                final_df, timestamp = final_columns_rename_lines(merged_df, timestamp)
10.                # Drop rows where all columns except 'Timestamp' , 'Bus' , 'id' contain zeros
11.                columns_to_check = final_df.columns.difference(['Timestamp', 'Bus', 'id'])
12.                final_df = final_df.loc[~(final_df[columns_to_check] == 0).all(axis=1)]
13.                all_sheets_data['X-lines'].append(final_df)
14.
```

Figure 19: X-Lines comparison sheet

Nodes sheet: Script loads Bus sheet of Unicorn's and OpenLF's specified timestamp LoadFlow reports (**df1, df2**). Renames both sheet columns that are necessary for merging. Merges sheets based on their common 'Bus' column, calculates Nodes differencies (Voltage magnitude, theta) and drops unnecessary columns. Renames the final data structure, adds timestamp column (current timestamp value) and removes rows where all columns except 'Timestamp' and 'Bus' have zero values. Appends final data to Nodes dictionary.

```
 1.                #Nodes
 2.                df1 = load_data(df1_path , 'Bus')
 3.                df2 = load_data(df2_path, 'Bus')
 4.                df1, df2 = rename_Nodes_data(df1, df2)
 5.                merged_df = merge_common_data(df1, df2, merge_columns=['Bus'])
 6.                merged_df = calculate_Nodes_differencies(merged_df)
 7.                columns_to_drop = ['Bus type', 'Reference voltage_UNICORN', 'Pgen_UNICORN' ,
'Reference Voltage' ,'Qgen_UNICORN', 'Pload_UNICORN', 'Qload_UNICORN', 'Reference voltage_OPENLF',
'Pgen_OPENLF', 'Qgen_OPENLF', 'voltage_regulator_on','Pload_OPENLF', 'Qload_OPENLF' ,'Area', 'Final
bus type', 'Island number' ,  'Base Voltage' , 'Reference voltage' , 'target_v.1', 'max_q', 'min_q'
, 'Pgen' , 'Qgen' , 'Pload' , 'Qload' , 'Eq. type' , 'Eq. type' , 'connected_component' ,
'synchronous_component' , 'id_gen' , 'target_v' , 'p' , 'q' , 'p_load' , 'q_load' , 'U_diff' ,
'theta_diff' , 'State']
 8.                merged_df.drop(columns=[col for col in columns_to_drop if col in
merged_df.columns], inplace = True)
 9.                final_df, timestamp = final_columns_rename_buses(merged_df, timestamp)
10.                # Drop rows where all columns except 'Timestamp' and 'Bus' contain zeros
11.                columns_to_check = final_df.columns.difference(['Timestamp', 'Bus'])
12.                final_df = final_df.loc[~(final_df[columns_to_check] == 0).all(axis=1)]
13.                all_sheets_data['Nodes'].append(final_df)
14.
```

Figure 20: Nodes Comparison sheet

X-Nodes sheet: Script loads for specific timestamp Unicorn's Bus sheet (**df1**) and OpenLF's X-Nodes sheet(**df2**). Renames essential for merging columns of both sheets and merges sheets on their common 'id' column. Calculates X-Nodes differencies(Voltage magnitude, theta) and drops unnecessary columns. Renames final data structure, adds timestamp column (current timestamp value) and drops rows where their columns have zero values (exception 'Timestamp', 'id'). Appends final data to X-nodes dictionary.

```
 1.  #X-Nodes
 2.                df1 = load_data(df1_path, 'Bus')
 3.                df2 = load_data(df2_path, 'X-Nodes')
 4.                df1, df2 = rename_X_Nodes_data(df1, df2)
 5.                merged_df = merge_common_data(df1, df2, merge_columns=['id'])
 6.                merged_df = calculate_Nodes_differencies(merged_df)
 7.                columns_to_drop = ['Bus type', 'Area', 'BUS' , 'boundary_p' , 'boundary_q' ,
'v_mag' , 'v_angle' , 'I_limit'  ,'Final bus type', 'Island number' ,  'Base Voltage' , 'Reference
voltage' , 'Pgen' , 'Qgen' , 'Pload' , 'Qload' , 'Eq. type' , 'Eq. type' , 'bus_id' , 'I' , 'P' ,
'Q', 'boundary_p' , 'boundary_q' , 'connected_component' , 'synchronous_component' , 'id_gen' ,
'target_v' , 'p' , 'q' , 'p_load' , 'q_load' , 'U_diff' , 'theta_diff' , 'State' ]
```

```
                merged_df = merged_df.drop(columns=[col for col in columns_to_drop if col in
merged_df.columns])
                final_df, timestamp = final_columns_rename_buses(merged_df, timestamp)
                #DROP ROWS WITH ZERO COLUMNS
                columns_to_check = final_df.columns.difference(['Timestamp', 'id'])
                final_df = final_df.loc[~(final_df[columns_to_check] == 0).all(axis=1)]
                all_sheets_data['X-Nodes'].append(final_df)
        else:

            continue  # Continue to the next loop iteration

```

Figure 21: X-Nodes comparison sheet

NOTE: Same proccess is repeating for every timestamp.

After loop is finished, script creates an excel with four different sheets, containg all different timestamp comparisons.

with pd.ExcelWriter(combined_output_path, engine='openpyxl') as writer: Creates an excel in combined_output_path.

for sheet_name, dataframes in all_sheets_data.items(): Iterates through each key dictionary of all_sheets_data that contain the previous data frames for all user specified timestamps.

consolidated_df = pd.concat(dataframes, ignore_index=True): For each separate key dictionary, script concatenates all the different timestamp data creating the final sheet.

if not consolidated_df.empty: Checks if dataframe of sheet is empty. If it is, script prints an error message. If not, writes concatenate dataframe to the sheet.

Final, script adds specific headers to each sheet using the predefined make_adjustements functions.

```
# Once all data is collected, save it to the Excel file in different sheets
    with pd.ExcelWriter(combined_output_path, engine='openpyxl') as writer:
        for sheet_name, dataframes in all_sheets_data.items():
            if dataframes:
                consolidated_df = pd.concat(dataframes, ignore_index=True)
                if not consolidated_df.empty:
                    consolidated_df.to_excel(writer, sheet_name=sheet_name, index=False)
                else:
                   print(f"Sheet {sheet_name} has no data. Skipping sheet.")
            else:
                   print(f"No data for sheet {sheet_name}.")


    for sheet in all_sheets_data.keys():
        if sheet in ['Lines', 'X-lines']:
            # Use the function for lines
            make_adjustements_lines_to_excel(combined_output_path, sheet)
        elif sheet in ['Nodes', 'X-Nodes']:
            # Use the function for nodes
            make_adjustements_nodes_to_excel(combined_output_path, sheet)

```

Figure 22: Final excel

6. Main()

Script gets the user inputs and calls **process_files_and_accumulate_data()**.

```
1. if __name__ == "__main__":
2.     # User inforamtion
3.     destination_folder, destination_folder_1, Date, File_type, country_code, numbers, timestamps
= get_user_inputs()
4.     #Processing User's info for TCC
5.     process_files_and_accumulate_data(timestamps, numbers, Date, File_type, country_code,
destination_folder, destination_folder_1)
6.
```

Figure 23: Main()