Test Procedures and Test Report for Data Pre-processing Pipeline

Version 1.0

prepared by: ALROY CHIANG

organization: edpr sunseap

Date: 04-03-2023

Contents

[Revision History 2](#_Toc129349210)

[1. Test Procedure 3](#_Toc129349211)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for changes** | **Version** |
| Alroy Chiang | 04-03-2023 | First version | 1.0 |
|  |  |  |  |
|  |  |  |  |

# Test Procedure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Procedures** | **Expected Outcome** | **Result** | **Remark** |
| **Test 1.1** | Use “staging” folder to place all raw data files to be processed | | | |
|  | * Place PowerPoint, text, word, pdf and csv files into “staging folder” * Execute program | * Files will be in “cleaned” or “rejected” folder | pass |  |
| **Test 1.2** | Raw data files are to be in CSV format | | | |
|  | * Place PowerPoint, text, word, pdf and csv files into “staging folder” * Execute program | * Csv files will be processed into “Cleaned” folder * All other files will be moved to “rejected” folder under “staging” directory | pass |  |
| **Test 1.3** | Raw data files will be deleted from “staging” folder once they have been processed | | | |
|  | * Place PowerPoint, text, word, pdf and csv files into “staging folder” * Execute program | * Files that have been processed will be in “Cleaned” folder or “Rejected” folder and the Raw data file will be deleted from “staging” folder | pass |  |
| **Test 1.4** | Raw data files with wrong extensions or file names that do not contain necessary information will not be processed and will be moved to “rejected” subfolder in “staging” folder | | | |
| **a** | * Place raw data files with names such as:   “example.txt”, “example.pdf”, “example.ppt” into “staging” folder   * Execute program | * Files with any other extensions other than “csv” will be moved to “rejected” folder under “staging” directory | pass |  |
| **b** | * Place raw data files with names such as: “Blk10ChaiCheeRd\_Agg”,   “Blk10ChaiCheeRd\_Sept2020\_Agg”, “Blk10ChaiCheeRdAgg”  “staging” folder   * Execute program | * Files with only 1 or none underscores will be moved to the “rejected” folder under “staging” directory | pass | Can demo both a and b files together in staging folder.  1.4 can demo 1.1, 1.2, 1.3, 1.5 altogether |
| **Test 1.5** | Raw data files will be processed sequentially | | | |
|  | * Place a set of ordinary raw data files into “Staging” folder * Execute program | * Files will process one by one in a random order. Console will output 2 statements “ \_\_file\_\_ has started processing” and “\_\_file\_\_ has finished processing” |  |  |
| **Test 1.6** | Raw data files do not necessarily contain a calendar month worth of data. Data files will be concatenated or segmented accordingly. | | | |
| **a** | * Raw data files containing different months’ worth of data are placed in “staging” folder   E.g 1 file containing raw data from 1st sept to 10th Sept. Another file containing raw data from 5th Sept to 15th Sept together having data of 1st Oct to 10th Oct. Placing both into “staging” folder.   * Execute program | * Files will be processed into “Cleaned” folder, under the postal code 467010 - > 2020 directories. Two files will be produced, one containing only Sept data while the other containing only Oct data. * Both files will have unique rows of data, duplicate rows will be removed. The data is a series indexed in time order. | pass |  |
| **b** | * First run 1.6a procedure. Run below procedures next. * Place the same set of files into staging folder * Execute program | * There should be no changes to output file. | pass |  |
| **Test 1.7** | Raw data files are to be named in the following format: location\_\*\_load.csv (case insensitive) | | | |
| **a** | * Place raw data files with names such as:   “Blk10ChaiCheeRd\_Sept2020\_agg”,  ”Blk10ChaiCheeRd\_Sept2020\_”,  “BlkChaiCheeRd\_Sept2020\_Agg(@lphAnUm3r!c)”  into “staging” folder.   * Execute program | * “Blk10ChaiCheeRd\_Sept2020\_” will be moved to the “rejected” folder under “staging” directory * “BlkChaiCheeRd\_Sept2020\_Agg(@lphAnUm3r!c)” will be processed into “Cleaned” folder. * Program console will output the number of files that have invalid filenames, 1 in this case | pass | Demo. Shows the load name variations |
| **b** | * Place raw data files with names such as:   “\_Sept2020\_Agg”  “77HarbourFront\_Sept2020\_Agg”  “Blk10ChaiCheeRd\_Sept2020\_Agg”  “Front77Harbour\_Sept2020\_Agg”  in “staging” folder   * Execute program * When program prompts user to enter postal code, for the file 77Harbourfront, key in 776655. * When program prompts user to enter postal code for the file, blk10chaicheerd, key in 467010. * When the program prompts user to enter postal code for the file, front77harbour, key in 776655. | * The file “\_Sept2020\_Agg” will be moved to “rejected” folder * The files “77HarbourFront\_Sept2020\_Agg” and   “Front77Harbour\_Sept2020\_Agg”  Will be merged and be output into “Cleaned” folder as a single file   * “Blk10ChaiCheeRd\_Sept2020\_Agg” will be processed into “Cleaned” folder | pass | Demo. Shows the address name variations |
| **c** | * Place raw files with names such as: “77HARBOURFRONT\_2020sept\_Lift”   “77HarbourFront\_Sept2020\_Lift”  “Blk10ChaiCheeRd\_2020Sept\_Lift”  “Blk10ChaiCheeRd\_Sept2020\_LIFT”  Into “staging” folder   * Execute program * When the program prompts user to enter a new postal code for “77HarbourFront” file, key in: 776655 | * Both files “77HARBOURFRONT\_2020sept\_Lift” and   “77HarbourFront\_Sept2020\_Lift”  Will be merged and output into “Cleaned” folder regardless of address letter casings   * Both files “Blk10ChaiCheeRd\_2020Sept\_Lift” and   “Blk10ChaiCheeRd\_Sept2020\_LIFT”  Will be merged and output into “Cleaned” folder regardless of load letter casings | pass | Demo. Shows capitalization address name variation |
|  | | | | |
| **Test 2.1** | Each column will be checked for missing values. Any non-numbers such as NaN and NaT, will be treated as missing values. The number of missing values will be recorded in the meta data. | | | |
|  | * Place a raw data file:  “Blk10ChaiCheeRd\_MissingValues\_Refuse” that consist of missing values and strings within its rows into “staging” folder * Execute the program | * Console output will display the Non-Null value counts and datatypes of each column. * User will decide if the program should continue processing or exit the program. If user keys in “N” the raw data file will remain in the “staging” folder * If user keys in ‘Y”, the files will be processed into “Cleaned” folder. A metadata text file will be generated alongside the cleaned file in the same directory. * The metadata file has a row named “Missing Values” which will indicate the number of NaN values with respect to each data column | pass | Demo. User to assess whether should generate missing values or not. If data provided is inaccurate, strings mixed with floats etc.. |
| **Test 2.2** | Dataframe will be checked for any duplicate columns by comparing identical column headers and identical column values. | | | |
|  | * Place a raw data file: “Blk10ChaiCheeRd\_Dup\_Col\_BPump “ into “staging” folder. Raw data file will consist of duplicated column headings such as having four “IRMS\_B” columns and three “IRMS\_A” columns. “IRMS\_B” column values are distinct and different from each other. “IRMS\_A” columns values are identical to each other. There are a total of 18 columns in this raw data file. * Execute program | * The file “467010\_2020\_09\_bpump” will be processed into the “Cleaned” folder. It will have 16 columns remaining as no “IRMS\_B” columns will be deleted while 2 “IRMS\_A” columns will be deleted. |  |  |
| **Test 2.3** | Each column will be checked for outliers. | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder. | * Files will be processed into “Cleaned” folder alongside its metadata file generated. * Metadata file will display a row named “Outliers”, indicating the number of Outliers for each data column |  |  |
|  | | | | |
| **Test 3.1** | Transform all other date-time formats into the following format: DD-MM-YYYY HH:MM:SS (24 hours format) | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Raw data file consists of a “TIMESTAMP” column with values in datetime format: YYYY\_MM\_DD HH:MM:SS | * Files will be processed into “Cleaned” folder. * The “TIMESTAMP” data column |  | Show any final output file |
| **Test 3.2** | Calculate readings in log-scale from linear scale | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Open program and edit config.json file. Edit “LOG\_SCALE”: [] and input the column names you wish to convert from linear scale to log scale within the square brackets as such: “LOG\_SCALE”: [ “IRMS\_A”, “IRMS\_B”, “IRMS\_C” ] * Execute program | * Files will be processed into “Cleaned” folder. * The cleaned data file will contain additional columns on the right side labelled as: LOG( column\_name ) * Depending on how many column names within the “LOG\_SCALE”: [] square brackets, there would be as many additional columns in the cleaned data file. |  |  |
| **Test 3.3** | Compute reading difference in time for selected columns | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Open program and edit config.json file. Edit “TIME\_DIFFERENCING”: [] and input the column names & period interval within the square brackets that you wish to generate the time difference interval for. For example: “TIME\_DIFFERENCING”: [["I1", 2], ["I2", 2], ["I3", 2]] * Execute program | * Files will be processed into “Cleaned” folder. * The cleaned data file will contain additional columns on the right side with column heading such as: column\_name\_diff(period) where period is the respective period interval. * Depending on the number of list objects associated with the “TIME\_DIFFERENCING” key, there would be as many additional columns generated as well |  |  |
|  | | | | |
| **Test 4.1** | A meta data file in text format will be generated for each of the cleaned data file. The meta file will be co-located with the data file. | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Files will be processed into “Cleaned” folder alongside its metadata file generated. * Both files will be in the same directory. |  |  |
| **Test 4.2** | The meta data file will be named similarly to the data file. | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Both the cleaned data file and the metadata file generated will have identical names apart from their filename extensions |  |  |
| **Test 4.3** | The meta data file within will contain the file’s name itself | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file generated will have the format: postal\_code\_YYYY\_MM\_load |  |  |
| **Test 4.4** | The meta data fille will contain the number of entries in the dataset | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display a row named “count” with the number of values under each respective column |  |  |
| **Test 4.5** | The meta data file will contain the location information where the dataset was collected. | | | |
|  | * Place a raw data file “Blk10ChaiCheeRd\_Sep2020\_combined\_outer\_Light” with the appropriate format: location\_\*\_load.csv into the “staging” folder * Ensure address\_legend.txt file has the full address: 467010, "10 Chai Chee Rd, Singapore 467010” as an entry. * Execute the program * Program will prompt user to enter 467010 as a new postal code into the program | * Metadata file produced will display a row named “Address” with the formal address beside it. For example: 467010, "10 Chai Chee Rd, Singapore 467010” |  |  |
| **Test 4.6** | The meta data file will contain the time resolution of its associated data file. | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display a row named “Time Resolution” which indicates the smallest time gap between rows in the cleaned data file |  |  |
| **Test 4.7** | The meta data file will contain the time gap statistics of the dataset | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display rows named “Minimum Time Gap”, “Median Time Gap”, “Mean Time Gap”, “Maximum Time Gap”. |  |  |
| **Test 4.8** | The meta data file will contain the “time period”, the start and the end time of the dataset | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display a row named “Time Period” which represents the length of time the data was recorded. “Start Time” and “End Time” rows will also be displayed to show the starting timestamp where the first entry was recorded and the ending timestamp where the last entry was recorded |  |  |
| **Test 4.9** | The meta data file will contain the reading statistics of each column within the dataset | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display the “mean”, “std” (standard deviation), “25%” (1st quartile), “50%” (2nd quartile), “75%” (3rd quartile), “max” (maximum value), “min” (minimum value) of each column respectively |  |  |
| **Test 4.10** | The meta data file will contain the missing values count of each column within the dataset. | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * Metadata file will display a row named “Missing Values” which shows the number of missing values under each column respectively |  |  |
| **Test 4.11** | The meta data file will contain the outliers count of each column within the dataset | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Raw data file will consist of duplicate column headings * Execute program | * Files will be processed into “Cleaned” folder alongside its metadata file generated. * Metadata file will display a row named “Outliers”, indicating the number of Outliers for each data column |  |  |
|  | | | | |
| **Test 5.1** | Data pipeline uses the output folder structure as shown in Figure 3 to allow a systematic approach to locate a file of interest. Data files will be placed into their respective folders according to location and the year in which the data is collected. | | | |
|  | * Place 3 raw data files of different months and different loads with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute the program | * The cleaned data files will be located in their respective postal codes and year folders in this manner: “Cleaned” folder -> Respective Postal Codes’ folder -> Year folder. |  | demo |
| **Test 5.2** | Each data file will contain only data collected in a same calendar month | | | |
|  | * Place the first raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder. This data file shall consist only of September data * Place the second raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder. This data file shall consist overlapping September rows of data and October rows of data. This data file must have identical address and load names as with the first raw data file. | * The files will be processed into “Cleaned” folder. * 1 file contains purely September data while the other contains purely October data |  | # shown in point 1.6 test case |
| **Test 5.3** | All cleaned data files will only use ‘,’ as CSV delimiter | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute program | * The cleaned csv file is in the “Cleaned” folder * Upon inspection, all values will be separated by a ‘,’ instead of a ‘\t’ as initially seen in the raw csv data |  |  |
| **Test 5.4** | An address legend file address\_legend.csv will be created to translate all postal codes into their full address for readability. This legend will be created and maintained manually and placed in the cleaned folder | | | |
|  | * Place a raw data file :” rd10ChaiCheeblk\_Sep2020\_combined\_outer\_LiftB” into the “staging” folder * Locate the csv file “address\_legend” in the “Cleaned” folder and ensure that the formal address 467010, “ 10 Chai Chee Rd, Singapore 467010” exist in the file. * In address.json, remove value: rd10chaicheeblk from key: 467010 * Execute the program * Program will prompt you to key in the postal code of the informal filename’s address. In this case, key in “467010” * Program will match the postal code to the address\_legend.csv file and extract its formal address format out to be included in the metadata.txt file | * Metadata created will consist of a row named:   Address: 467010, “ 10 Chai Chee Rd, Singapore 467010” |  |  |
| **Test 5.5** | Cleaned data files will use the following filename format:  postal\_code\_YYYY\_MM\_load.csv  The filename consists of 6 digits postal code, 4 digits year data is collected, 2 digits calendar month data is collected, and the type of load | | | |
|  | * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder * Execute program | * The Cleaned data file will be processed into “Cleaned” folder with its name being formatted into postal\_code\_YYYY\_MM\_load.csv * For example: 822255\_2022\_12\_WaterPump.csv |  |  |
| **Test 5.6** | Data files with time gaps larger than a user defined threshold will be placed in the ***rejected***folder. | | | |
|  | * Open program and open the config.json file. Set the user defined threshold as 5. For example: “TIMEGAP\_THRESH”: 5 * Place 2 raw data files with the appropriate format: location\_\*\_load.csv into the “staging” folder. In each of the raw data file remove 10 rows worth of data. * Place a raw data file with the appropriate format: location\_\*\_load.csv into the “staging” folder. This data file will have a running time-series data with no missing rows. * Execute program | * The single raw data file without any missing rows will be processed into the “Cleaned” folder. * The 2 raw data files with 10 missing rows will be moved into “rejected” folder in the “processed” directory as they have time gaps exceeded the user defined threshold. * The console output will be as follows:   “Due to large amounts of time-gaps in ALL of the months in this file, Blk10ChaiCheeRd\_Only200RowsRemoved10\_outer\_Solar has been moved to the "Rejected" folder. Due to large amounts of time-gaps in ALL of the months in this file, Blk10ChaiCheeRd\_Only200RowsRemoved10\_outer\_ZPump has been moved to the "Rejected" folder. Blk10ChaiCheeRd\_Sep2020\_combined\_outer\_LiftA has been processed. The number of cleaned files is: 1. The number of files that have invalid filenames is: 0. The number of monthly dataframes that has been rejected due to large amounts time gaps is: 2” |  | demo |