**User Guide and Breakdown for The Station Processing Script**

**Author: Ved Bhoot**

**Outline**

1. **Part 1 – Get stations**
   1. Initial input – Page
2. **Part 2 – Process Functions**
   1. Process.Precip – Page
   2. Process.T.RH.P – Page
   3. Process.WD.WS – Page
   4. Process.SR (still need to make) – Page
   5. fmt.meteodata – Page
3. **Part 3 – Data Improving Functions**
   1. webscrape.CDEC – Page
   2. format.CDEC – Page
   3. date.format – Page
   4. CFI.station – Page
   5. impute.station – Page
   6. ESPI.implement – Page
   7. RainDay – by Avecedo (2013) – Page
   8. MarkovRain – by Avecedo (2013) – Page
   9. SSA.nafill – Page
   10. MAD.outlier – Page
   11. CF.station.OV – Page
   12. MkvWind – Page

**Part 1 – Get Stations**

**CDEC.url** - The variable name explains it well. This is the html of the CDEC website which the contains a list of all the stations within a rectangular boundary of the study area. You can see within the url there is ‘lat’ and ‘lon’, there are 2 of each of them. These indicate the max and mins for the website to search for.

**Station.variables.list** – This is a list of tables of all the stations with the variables of interest. The tables are pulled using the ‘webscrape.CDEC’ function. The explanation for the function is located in that function’s section. The tables contain the locations for the stations, including the elevation. It also contains the station name, sensor number, the temporal type (hourly or event), the start collection year, and end collection year (automatically ‘present’).

**stations** – Variable with the station names pulled out from ‘goodstations’.

**temporal.type** – Variable with the temporal types for the stations.

**start.source.date** – This is the input for setting the start date(s) of interest, if there is a specific date that you decide to start with then it is set as anything other than ‘yes’, and ‘state.date’ should be adjusted to the single date desired. If ‘yes’ is set then the start.date will be a vector of the dates that the sensor started collecting data.

**contain.event** – This is input as ‘yes’ or ‘no’, it is the command for setting whether the stations to process should contain stations that collect data in event form rather than hourly.

**Part 2 – Process Functions**

**\***All function inputs full explanation in their respective section

List of inputs and locations:

var.interest – Part 1

stations – Part 1

temporal.type – Part 1

skewremove.1 – Part 3 – format.CDEC

SkR.max – Part 3 – format.CDEC

SkR.min – Part 3 – format.CDEC

startset – Part 3 – format.CDEC

endset – Part 3 – format.CDEC

espi.apply – Part 3 – CFI.station

lagg.apply – Part 3 – CFI.station

skewremove.2 – Part 3 – CFI.station

skewvalmax – Part 3 – CFI.station

ssval – Part 3 – CFI.station / MkvWind

upper.threshold – Part 3 – MAD.outlier

lower.threshold – Part 3 – MAD.outlier

window – Part 3 – MAD.outlier

rm.outlier – Part 3 – MAD.outlier

showplot – Part 3 – MAD.outlier

L – Part 3 – SSA.nafill

NAconcurrent – Part 3 – MkvWind

okayNAnumber – Part 3 – MkvWind

**Process.Precip** – This function is used for processing precipitation data. Currently it is customized for CDEC. The first portion is a for loop where it pulls the station name. It then checks with an if and else statement whether it is ‘hourly’ or ‘event’ data by looking at ‘temporal.type’. The next three portions are specific to CDEC formatting. Once it pulls the dataset from the web it takes care of any more formatting issues through the ‘format.CDECdata’ function. Appropriate column names are then given and the set then begins processing. The ‘CFI.station’ function processes the data and then it is exported. The next iteration begins. For event data the process is similar, no conversion to numeric is needed as ‘format.CDEC’ takes care of that for it.

**Process.T.RH.P** – The idea is similar to ‘Process.Precip’. The main difference is the first step and the processing step. ‘is.RH’ is an input for ‘SSA.nafill’, its purpose is explained in the ‘SSA.nafill’ section in Part 3. It is initially automatically set as false, but the next if statements take care of this if the variable of interest is relative humidity. These if statements are for changing the sensor number in the url. The next for loop portion is like ‘Process.Precip’ so I will not explain this on here. The processing step takes the formatted data and first goes through the ‘MAD.outlier’ function to get rid of the outliers. Once this is done the ‘CF.station.OV’ (OV = other variables) is used for LOCF and linear interpolation of the data. Once this is done it goes to the ‘SSA.nafill’ function for Singular Spectrum Analysis NA imputation. It is then finished and exported.

**Process.WD.WS** – Similar idea as explained before for the for loop. This function first gets the sensor number. Both wind direction and speed are required for this to work. In the if/else statements both WD and WS data are pulled and formatted. ‘MkvWind’ processes this data using a Markov Chain based imputation method. The output is within one data frame so it is separated into two variables. The data is then exported.

**Process.SR** –

**fmt.meteodata** –

1. The inputs:
   1. meteo.var – variable of interest.
   2. file.names – names of the files containing the data that needs to be formatted.
   3. file.names.2 – names of the files containing the data for wind direction.
   4. folder.location – location for the files containing the data.
   5. folder.location.2 – location for the files containing the data for wind direction.
   6. espi – set as TRUE/FALSE, if the dataset was transformed with ESPI index.
   7. espi.lagged – set as TRUE/FALSE, if the ESPI index was lagged or not.
2. For formatting data to ParFlow guidelines. The conversions are specific to the outputted units from the functions in this script.
3. This function is very straightforward, conditionals containing the respective conversion for the meteorological variable of choice.

**Part 3 – Data Improving Functions**

**webscrape.CDEC** –

1. The inputs:
   1. ‘CDEC.url’ this was explained in Part 1.
   2. startyear – The maximum beginning year for data collection.
   3. endyear – The minimum end year, if the year is the current year set variable as ‘present’.
2. The first if statement checks if the input is correct.
3. The first for loop max number ‘numb.prelimstn’ gets changed per how many stations are located within the defined rectangular.
   1. This for loop pulls all the stations from the website. The url used is the search result from the CDEC website. Depending on the latitude and longitude bounds the results will likely be different. This loop web scrapes and extracts all the station names into the ‘namevar’ list. The list is then converted into a more usable format.
4. The next for loop for pulling the list of variables that a station has.
   1. The CDEC website has a page for each of the stations which shows the variables and other details of the variables that it has recorded.
   2. The next set of if statements are for checking and pulling the columns of the details that we want such as the time period it was recorded from.
      1. There is a lot of formatting attached by and in these statements.
         1. A difficult to follow part of the CDEC website is the html locations for the information that we’re looking for. A tool that tells the location on the webpage, selectorgadget, produces completely different results when checked in R. Because of this the script pulls the general location and in turn unnecessary information is attached to it. This unnecessary information contains certain elements within them that we do not want, such as random dates or words where there shouldn’t be.
   3. Lastly these are moved around and eventually placed in two lists ‘fullvar’ and ‘stnvarlist’.
      1. ‘fullvar’ is for the station location variables, elevation, latitude, and longitude.
      2. ‘stnvarlist’ contains a list of all the variables that the station has.
   4. The last for loop contains a for loop which aims to find out if the variable of interest is located within the stations in the area. The end goal is to output a list containing data frames for the details of the stations that contain the respective variable of interest
      1. The which statements are used to find within ‘stnvarlist’ if the variable and variables’ details are there.
         1. ‘enddate.check’ is there for formatting purposes, its full explanation is within the code.
         2. The if statement containing ‘checkvar.list[var.interest] >0’ is checking if the data is there or not.
      2. If not found then the next iteration is run.
      3. If it is found then it binds the station name and location variables together.
      4. The names() portion is for passing an error in R that the column names are different.
      5. Lastly the stations and its details are collected into ‘goodstationvars’.
      6. To finish the function the data table of details is formatted and then placed within the list, ‘all.variables’, and then iterated until the for loop is finished.

**format.CDEC** –

1. The inputs:
   1. ‘raw.CDEC.data’ – Data frame with the raw data pulled directly from CDEC. The specific format required is mentioned in the code right above the function.
   2. ‘formatv1’ – Selector of formatting. This is mainly for hourly data which needs some formatting.
   3. ‘formatv2’ – Second selector of formatting. This is for event data.
   4. ‘skewremove.1’ – Used to decide if you want to remove any unreasonable skews, this is before any of the processing for the data in Part 1 is done.
   5. ‘SkR.max’ – If ‘skewremove.1’ is true then this determines the maximum values that are okay to keep in the dataset
   6. ‘SkR.min’ – Same as ‘SkR.max’ except the minimum value.
   7. ‘startset’ – Start date of the dataset in the format month/day/year (--/--/----)
   8. ‘endset’ – End date of the dataset in the format month/day/year (--/--/----)
2. The first if statement checks if both ‘formatv1’ and ‘formatv2’ are the same.
   1. Only one can be set as true
3. ‘formatv1’ section
   1. Initial step is to remove excess stuff that is attached to the data.
   2. The while loop gets the dates necessary.
   3. These dates are then separated out month, day, year and repeated 24x (24 hours).
   4. The next for loop is for collecting a day count for the first column.
   5. The day count, dates, and variable data are bound together.
   6. The last for loop is to convert the class of the data into usable numeric, the dataset is then given appropriate column names.
4. ‘formatv2’ section
   1. Initial step is to remove excess stuff that is attached to the data.
   2. Next two for loops are for some class conversion into a more usable numeric class.
   3. The next portion from ‘locatorvar’ and the while loop is for identifying which recordings occurred multiple times in an hour.
      1. The while loop works with some indexing, where it checks the next set of values if within them there are repeated collections in the same hour.
         1. The first if statement is for ensuring the loop does not fail when checking that last few elements within the set.
            1. The statements inside do the checking and average the values for those that are repeats
   4. Once this is done the next portion is adding hours which did not have recordings as NA.
      1. First a set of the full dates and hours is produced with the while and for loop.
         1. This placed into ‘fulltimevar’
      2. The column where the data values are supposed to be are all set as NA and a tracking column, ‘rowtrack’, are added to ‘fulltimevar’
      3. The next for loop is for checking which elements in the event data are there for their respective hour and which are not, and placing them into the variable ‘whichvec.time’
         1. Instead of searching through the whole data set if the data for an hour exists or not, the data set is split into year 🡪 month 🡪 day as part of multiple for loops. This increases the efficiency of the loop as the whole dataset does not need to be checked iteration after iteration, rather the appropriate sections of the set are checked part by part.
   5. Lastly the data is put into ‘fulltimevar’, ordered, given a day count, named, and skews are removed if ‘skewremove.1’ is set as true.

**date.format –**

1. The input:
   1. the.date – this is the start date for the station.
2. The function’s goal is to adjust the start date by 1 day before. The reason for this is because the CDEC website for hourly data does not pull the value at the starting boundary time and thus the previous day at hour 23 is set on the url.
3. The logic behind the function itself is very straightforward and does not need much explanation.

**CFI.station** –

\*Some explanation is contained within the code

1. The inputs:
   1. station – Dataset for precipitation. MUST BE CUMULATIVE.
   2. espi.apply – Set as true or false, transform the data with the ESPI index.
   3. lagg.apply – Set as true or false, lag the ESPI index, currently preset as 1 year.
   4. skewremove.2 – Set as true or false for whether you want any skews removed (some skews may be an artifact of the data conversion and processing)
   5. skewvalmax – Maximum allowable rate (set in inches!) after conversion from cumulative to rate. This is important because the cleaning portion with the case specific LOCF is not perfect and therefore conversion to rates could lead to inaccurate rates. An example is with the first element in the set, let’s say it is 3.24 inches and the next is 3.25 inches, we do not know what came before the first so in conversion it is converted to 82.29 mm/hour but the next is 2.54 mm/hour. 82.29 mm/hour is definitely not accurate and thus it should not be included as is it a product of the data processing. It is unreasonable to manually search through the whole set so setting a reasonable max, where high correct rain rates are still present but incredibly skewed rates as mentioned, suffices.
   6. ssval – setseed() value.
2. There is a lot of indexing involved in this function, fully explained in (3.). The initial portion is for setting NA’s and producing the indices to use for the LOCF portion. The idea behind these indices is to track based on the cumulative nature of the data.
   1. The function begins by separating the data into two variables, one with only NA’s and another with all real values.
   2. The first goal is to adjust any abnormal values of 0, it does so by initially looking at singular cases of 0 where there is only a single 0 with no consecutive 0’s. The aim is to look for values where something might have happened to the sensor and in turn it resulted in a recorded value of 0 and not the proper value.
   3. After it finishes doing this it goes into checking consecutive zeros. The idea is to take boundary 0’s in a consecutive 0 set and look at the values by them.
      1. After obtaining the values the script then considers if the high-end boundary value is equal to or greater than the low-end boundary value. If this is the case then the middle values are filled in using LOCF, if this is not the case then in all likely hood the measurement was just reset (due to the cumulative nature).
   4. After finishing the 0’s the next task is to address values >0. This part is mostly reliant on LOCF and checks various cases for replacement. This section is not perfect, but for the most part it seems to address most cases relatively well. Issues can arise when a certain case does not appropriately fit the conditions, substitutes an inconsistent LOCF, and then continues to do so because of the initial skew. This is unlikely to occur but it is possible if such case is encountered. A method to check this is to only run through the replacement portion of the script/function and compare the original and replaced precipitation data. All these methods adhere to the cumulative nature of the data (unless, there is a case which isn’t appropriately addressed).
   5. After finishing this the data is converted from cumulative to a rate.
   6. The two variables ‘stationtemp1’ and ‘stationtemp2’ are combined, organized into the correct place (recall the placement of ‘organizevec’ into ‘station’ in the beginning), and put into the variable ‘stationfixed’
   7. Next is ‘impute.station’
3. How the indices work:
   1. ‘whichvec.1’ is first used to set skewed values or NaN’s as NA.
   2. ‘whichvec.na’ then collects the locations for all the NA’s.
   3. ‘stationtemp1’ and ‘stationtemp2’ split the data, with the prior containing all real values and the latter containing the NA’s. ‘organizevec’ is added at the beginning in order to keep the location for the NA’s in the correct spot when bringing it together at the end.
      1. They are split in order to efficiently look at cases for fixing.
   4. ‘whichvec.0’ gets us a set of TRUE’s and FALSE’s, which tells us where the 0’s are.
   5. ‘whichvec.0.T’ then tells us the location within ‘whichvec.0’ where the TRUE’s are.
   6. ‘whichvec.0.TS’ is used for ‘checkvec.0’ in which the goal is to get a set where 0’s tell us that the value in the initial set was consecutive.
   7. ‘whichvec.0.T.2’ is used right after to set 0’s as TRUE and all else FALSE.
      1. Example: a location vector c(1, 2, 3, 5, 10, 11, 12, 20)
      2. Shifting and subtracting 🡪 c(1, 2, 3, 5, 10, 11, 12) – c(1, 2, 4, 9, 10, 11, 19)
         1. Resulting vector: c(0,0,-1,-4,0,0,-7)
         2. T/F vector: c(T, T, F, F, T, T, F)
   8. Because we had to shift the set we need to add a FALSE at the end to get the full set. The reason for adding a false at the end is that regardless of whether the previous was TRUE or FALSE, the last element will end a set of consecutive 0’s or be a single set 0.
      1. A FALSE ends a set of any size, so going back to the example in (g.) we see the first set of consecutive values are c(1,2,3) and in the T/F vector we have c(T, T, F), the FALSE indicates the last element in that set which is 3. The next set is c(5) and thus is c(F), the next cases of c(11, 12) and c(20) are similar to the previous. A FALSE after a FALSE is always a single set, a TRUE after a FALSE begins a consecutive set, and a FALSE after a TRUE ends that set.
      2. So, adding a FALSE at the end of the example set gives us:
         1. c(T,T,F,F,T,T,F,F)
            1. c(T,T,F) 🡪 c(1,2,3) 🡪 consecutive
            2. c(F) 🡪 c(5) 🡪 single
            3. c(T,T,F) 🡪 c(10,11,12) 🡪 consecutive
            4. c(F) 🡪 c(20) 🡪 single
      3. Let’s say that the 20 in that set is 13 instead and thus consecutive.
         1. The resulting vector and T/F vector are:
            1. c(0,0,-1,-4,0,0,0) and c(T,T,F,F,T,T,T)
         2. We see that by adding a FALSE at the end of the T/F vector will close that consecutive set thus indicating c(10,11,12,13) (c(T,T,T,F)) is consecutive.
4. **impute.station –** 
   1. If ‘skewremove.2’ is true then skews are removed based on ‘skewvalmax’.
   2. If ‘espi.apply’ is true then ‘ESPI.implement’ is run, explained in (4.). If it is false then the data is not transformed and the only thing different is just the location of the data within ‘stationfixed’
   3. The goal for this function is implementing a Markov Chain imputation method.
      1. ‘stationtemp3’ collects the data and has values > 0 set as 1 (indicating wet hour).
      2. The first for loop splits imputation by month.
         1. The next for loop collects the counts for wet hours and dry hours, year by year.
            1. Within are two for loops, the first is for collecting the probability for a dry hour occurring after a dry hour and the other for dry if wet.
            2. After the counts are collected the probability is produced by summing the respective counts and dividing them by the total counts.

These are then used for the transition matrix.

* + - 1. Next the seed is set by ‘ssval’.
      2. The magnitudes of rain are collected, not including 0’s.
      3. ‘rainVals2’ is the vector used to fit the Weibull Probability Distribution Fuction
      4. Parameters for Avecdo’s (2013) function ‘MarkovRain’ are collected.
      5. After ‘MarkovRain’ is run we now have a set of values to place into the NA’s
      6. Once the values are placed in the NA’s the data is transformed (if ‘espi.appply’ was set as true) back to include ENSO

1. **ESPI.implement –**
   1. ‘lagg.apply’ is checked in the if statement to see if the ESPI index should be lagged or not.
      1. If it is lagged, then it is lagged 1 year forward.
   2. ESPI is collected directly from the web.
      1. In the for loop the values are then repeated for the length of the respective month
   3. ESPI values are attached ‘stationfixed’ and then transformed
2. **MarkovRain – by Avecedo (2013)** 
   1. **RainDay – by Avecedo (2013)**

**MAD.outlier** –

1. The inputs:
   1. upper.threshold – Tu
   2. lower.threshold – Tl
   3. window – This is the interval of time which to produce the median value in the equations above, input should be in days.
   4. rm.outlier – Set as true if detected outliers should be removed, false if not.
   5. showplot – Shows three plots in one panel if true, one with the original data, the next with the outliers labeled with points, and the data with outliers removed
2. The window is automatically multiplied by 24, so the input should be in days.
3. ‘z’ contains the upper thresholds and ‘z.1’ contains the lower threshold.
4. The locations for the outliers are pulled with the ‘which(y > z)’/‘which(y > z.1)’ statements.
5. The last if statements are self-explanatory.

**CF.station.OV** –

1. Some of the initial concepts in this function are similar to ‘CFI.station’.
   1. The initial portions with the whichvec’s are for indexing purposes.
2. The main idea behind the for loop is find the different occurrence cases for NA’s
   1. These cases are located using the indices.
      1. The concept behind these indices to find consecutive NA’s
         1. First get the locations for the NA’s.
         2. Shift them by subtracting 1 from those locations
         3. Find which in the locations variable are equal with the shifted one.
            1. This is set in another variable, this contains only trues and falses
         4. Every false begins a set of NA’s, if a true comes after a false then it is consecutive, if a false occurs after then that ends the set.
            1. False, true, true, false 🡪 3 consecutive NA’s
            2. False, false 🡪 One NA
            3. \*Hopefully I recalled this right, this one took a bit to come up with and was a bit tricky
   2. Singular case
      1. An NA with a value in front and before it.
      2. If this case is found then LOCF is used.
   3. Medium length case
      1. Consecutive NA’s, below a defined length (‘len.set’).
         1. This is set as <24.
      2. If this case occurs the missing data is linearly interpolated.
   4. Long length case
      1. Cosecutive NA’s above a defined length (‘len.set’).
         1. This is set as >24.
      2. If this case occurs the missing data is pulled from the previous year.
         1. If the previous year is not there/has no data then the next year is used.
            1. This does NOT guarantee all or any NA’s to be filled as both the previous and next year can have NA’s.
   5. This for loop can be run again until the number of NA’s do not change.

**SSA.nafill** –

1. The inputs:
   1. L – This is the window length for fitting a smoothed curve to the data. The window length must be equal to or below half the length of the data set. The window length is automatically set as half the set length if it is not defined.
   2. is.RH – explained in Part 2.
2. First the NA’s are located.
3. The SSA curves are fitted and the gaps are filled.
   1. If the variable of interest was relative humidity and if any of the fitted values were above 100 they are forced to be 100.

**MkvWind** –

1. The inputs:
   1. WSdata – Wind speed dataset
   2. WDdata – Wind direction dataset
   3. NAconcurrent – This is set as true or false. If true then if at a given time wind speed or wind direction have an NA and the other does not, the other is set as NA. The variable ‘okayNAnumber’ is for the maximum number of values that are okay to fill with NA’s. If more than this many are to be filled then they will not be filled with NA’s.
   4. okayNAnumber – The maximum number of values within a variable to be converted to NA if not concurrent with the other variable’s NA.
2. First any extreme skews are removed and values for wind direction that are above 360 are removed.
3. In the first if statement the length of NA’s that will be filled are checked.
   1. If the total number of those are under ‘okayNAnumber’ then ‘controller.TF’ will be set as true, otherwise it will be false.
   2. If ‘controller.TF’ is true then the if statement within runs and sets those values as NA’s.
4. Next is producing a table of all the wind directions and their respective angles.
5. This table is used in the first for loop to assign all the angles within the wind direction set directions.
6. ‘NA.pdata’ is there to check how many NA’s are currently there and also to check if the while loop needs to be iterated again if the initial run was unable to fill in all NA’s.
   1. ‘NA.pdata.2’ is set as 1 so the while loop can initially register it. The purpose for it is to check with ‘NA.pdata’ at the end to see if the number of NA’s in the iteration changed or not. One run through of the loop does not guarantee that all NA’s will be removed. This is because some months at the end might have NA’s and because of this a month afterward with NA’s in the beginning will have nothing to predict from. Therefore, multiple runs must be done until the only NA’s remaining are those at the beginning of the dataset where the is nothing to predict from.
7. Next is the while loop, the condition is to check if the number of NA’s changed or not, if not then the loop ends and goes onto LOCF filling.
   1. The first part is to remove the value in ‘NA.pdata.2’ so when ‘NA.pdata’ is given the length of NA’s ‘NA.pdata.2’ does not automatically equal ‘NA.pdata’ at first, but rather it gets checked at the end of an iteration.
   2. Next is the first for loop, this portion breaks down the set so that probabilities can be pulled on the temporal scale of month per all years.
      1. In this all the variables, lists, vectors, etc. are set up for use within the rest of the loop.
   3. The next for loop breaks down the monthly data by direction, this is done in order to collect all the state occurrence probabilities for the transition matrix.
      1. The for loop in it collects the data per year and collects counts.
   4. Once running through fully the counts are summed and converted to probabilities, which then forms the transition matrix. Along with this speed values (>0, probabilities used later take this into account) and angles for the directions are collected.
      1. The if statement containing ‘sum(dir.P.var.set[,2] >0)’ is there so that 0 does not get divided by 0.
   5. The next portion is collecting the Weibull Pdf shape and scale factors, and also collecting the density distribution for the angles.
      1. The angles use density distributions because they do not necessarily have any specific fit other than the amount they occurred. Along with this is that they are bound by the interval of values the respective direction can be.
   6. The next portion is for utilizing the transition matrix and the distributions.
      1. The first if statement is there so that the loop doesn’t fail as there are no values to predict from if the first value of the set is NA.
      2. The for loop iterates by the location for each NA.
         1. The first if statement in this is if there is a NA before, and thus the direction cannot be predicted from.
         2. First a direction is predicted.
            1. If there is already a value in a spot then it will not be filled in. otherwise it will be filled in and an angle will be pulled from the density distribution.

If the angle pulled is out of bounds it will be pulled again as long as it is within the bounds, this is part of the while loop.

North is special as it can be from 348.75 to 360 or 0 to 11.25, and thus has its own if statement to take this into account.

* + - 1. Once there is a predicted direction (if predicted is kept, otherwise the direction already in the set is used) the speed is predicted.
         1. Before any prediction can be done because the Weibull Pdf cannot take 0’s a probability method is used to first check the state of whether wind occurs or not.

The state is then predicted with the probabilities produced (0 = no wind, 1 = wind).

If 0, then the speed is set at 0.

If 1, then the speed is sampled from the appropriate Weibull Pdf.

* 1. Lastly ‘NA.pdata.2’ is collected, if it equals ‘NA.pdata” from the beginning then the while loop ends, but if not then the next iteration begins.

1. The last step is LOCF, this is very straightforward. The LOCF is with values in front and not a year ahead. After this is done there should be no more NA’s left, if the NA’s before LOCF were not in the beginning the entire function is ended as it implies something went wrong with the data or the function.