# **USER MANUAL: PYTHON PROGRAMS FOR R CALCULATION**

# RasterManipulation\_2

The “RasterManipulation\_2” is the latest version of the Python program.

**Program location:** Y:\Abt1\hiwi\Oreamuno\GIS\03\_R\_Factor\Python\Python Programs\Raster Manipulation

**Objective**: Its main purpose is to convert the received precipitation and temperature into .csv files that can be used to calculate the R factor for each cell with the program RFactor\_5.

## Input data:

* *path\_precip*= folder location where the **precipitation** rasters are found
  + Files must be in .txt form
  + File name must be in the format: YearMonthDay\_Hour (YYYYMMDD\_0HH)
* *path\_temp*= folder location where the **temperature** rasters are found
  + Files must be in .txt form
  + File name must be in the format: YearMonthDay\_Hour (YYYYMMDD\_0HH)
  + There must be one temperature file for each precipitation file.
  + Temperature and precipitation rasters must have the same resolution (same number of cells)
  + It is not mandatory to introduce temperature values (see input file 4)
* *savepath*= folder location where the .csv files created by the program will be saved
* *temp*= binary variable
  + True means you are introducing temperature files.
  + False means you are not introducing temperature files. In the resulting file, the temperature value will be 9999

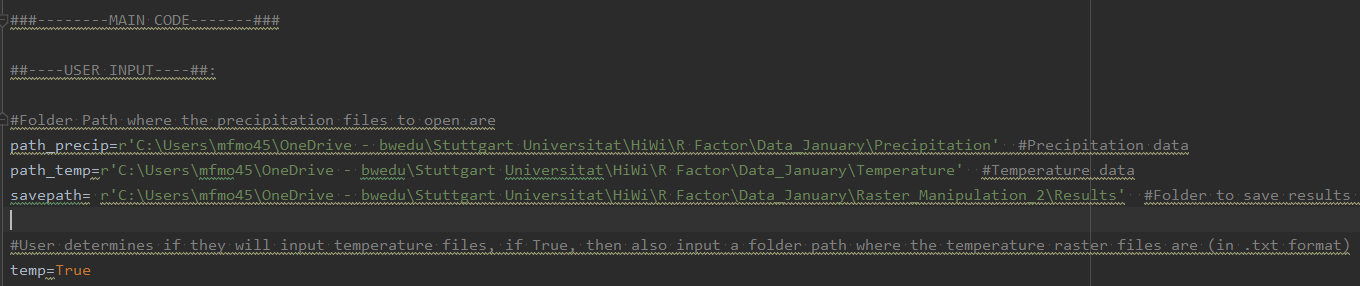


Figure . User input for RasterManipulation\_2 program

## Output data:

.csv file for each cell, containing the precipitation value for each time interval available:

* File name**:** OriginalRow\_OriginalColumn in the raster
  + This information will be used in the “RFactor\_5” program in order to recreate the original raster, assigning the corresponding R factor to each cell, in its original position
  + Comma delimited \*.csv file for each raster cell with the information shown in Figure 2.

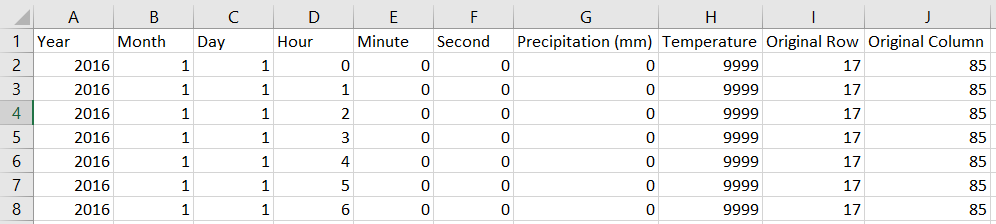


Figure . Information in resulting \*.csv file for each cell in the original rasters

## Error checks

The program checks for the following information:

* If the input file directories and the result directories exist
* If there are an equal number of precipitation and temperature files, in case the *temp* binary variable is *True*
* It does **NOT** check if the precipitation files and temperature files correspond to the same dates. It combines the precipitation with the temperature in the same order they are read in the folder

## Additional information

* The amount of files that can be read during one program run depends on the computer’s capability and storage availability.
* Running the program one month at a time takes about 10 min, using Computer\_063

## Main functions in the program

### Main loops:

* Precipitation Loop:
  + Reads through the precipitation files in the “*path\_precip*” folder
  + Saves the name (which contains the date of the precipitation records)
  + Saves the raster file in a Numpy Array, ignoring the first 6 rows (which contain raster properties information)
  + Calls **Precipitation** function, which returns a 3D matrix with the precipitation values for each cell. Each cell corresponds to an array in the 3D array.
* Temperature Loop:
  + Enters only if the user inputs a temperature folder and “*temp*” variable is equal to True.
  + Reads through the temperature files in the “*path\_temp*” folder directory.
  + Saves file name (which contains the date of the temperature records).
  + Calls **Temperature** function to save the temperature record to the corresponding precipitation value, for each cell, in the 3D array.

### Precipitation Function

* Receives:
  + *name*: File name
  + *storm\_array*: Numpy array with the data from the raster file being read
  + *tdm*: 3D matrix, which is modified in each loop. It is also the file that is returned, once the data from the “storm\_array” has been added
  + *i*: iteration number. It determines the row dimension in *tdm* being filled, for each array
* Calls **GetName** and, from *name*, it reads the date of the precipitation record and saves it to an array, where each column corresponds to Year, month, day, hour, minute and second (in that order).
  + If the name of the file doesn’t include minutes and seconds these values are 0
* Calls **ValueMatrix** and, from *storm\_array*, it creates a new Numpy array called with only the cells that have a data value (different from -9999). The function saves the precipitation value, the original row location and original column location for each cell.
* Calls **ThreeDArrayPrecipitation**, where the date, precipitation value and original row and column location are saved the 3D array *tdm*. Each array (dimension k) corresponds to each cell, and each array has as many rows as raster files introduced (dimension i, one for each date) and 10 columns (dimension j)
  + Since we have not introduced the temperature value, the temperature column is filled with 9999 °C.
  + The 3D arrays is filled one date at a time (since the 1st main loop reads through individual date raster files) and adds the new temperature data under the previous one.
  + Returns: 3D array *tdm*

### Temperature Function

* Receives the following variables:
  + *name*: File name
  + *temp\_array*: Numpy array with the temperature data from the raster file being read
  + *tdm*: 3D matrix, which was previously filled in the “Precipitation Loop” with the precipitation data”. It is modified in each loop. It is also the file that is returned, once the data from the “temp\_array” has been added
  + *i*: iteration number. It determines the row dimension in *tdm* being filled, for each array
* Calls **GetName** and, from *name*, it reads the date of the precipitation record and saves it to an array, where each column corresponds to Year, month, day, hour, minute and second (in that order).
* Calls **ValueMatrix** to save the data values (different to -9999) from *temp\_array* to a new Numpy array. The function saves the precipitation value, the original row location and original column location for each cell.
* Calls **ThreeDTemperature**, where the temperature value is added to the corresponding precipitation value row, in the 7th column, for each array in the 3D array *tdm*, which was previously filles with the precipitation data.
  + Function checks that the date and the original cell location of the temperature value correspond to the precipitation data’s location and temperature, for the given iteration. This means the precipitation and temperature data must be read in the same order.
  + Return tdm

### SaveFiles Function

* Receives:
* *tdm*: 3D matrix with the precipitation and temperature data for each cell
* *path:* save\_path
* Iterates through each array “k” in the 3D array, which corresponds to each of the cells in the original raster
* Extracts the kth array and transforms it into a 2D array using the “reshape” command
* Saves 2D array to a Pandas DataFrame, with column names in order for the user to be able to open files and easily understand it.
* Saves DataFrame to a comma-delimited .**csv file**, whose name corresponds to the original cell’s row\_column values. (See Figure 2 for file configuration format)

# RFactor\_5

(RFactor\_5 is the latest version of the Python program.)

**Location:** Y:\Abt1\hiwi\Oreamuno\GIS\03\_R\_Factor\Python\Python Programs\RFactor Calculation

**Objective:** Calculates the R factor on a per storm, per month and per year basis for each cell in the original Raster, and generates an output raster for each time frame with the corresponding R factors for each cell. The resulting Rasters can be opened in GIS program in order to be used in the RUSLE equation.

## Input data:

### Storm Event Input:

* *time\_interval*: User must input the time discretization of the precipitation records. This input will be checked with **one** of the input files. If the input value doesn’t coincide with input files, then an error is produced.
* *min\_6Hours*: minimum precipitation value (in mm) in a 6 hour time step that must be met in order to either start a storm event or decide when the event ends. If this value is set to 0, a storm will start when a precipitation record is greater than 0, and will end when, in a 6 hour window, the total precipitation is 0. Panagos et al, (2015) sets this value at 1.27 mm.
* *min\_P*: minimum total storm precipitation (in mm) in order to consider the given storm as erosive. If the value is set to 0, all storms will be considered, regardless of the total precipitation value. Panagos et. Al (2015) sets this value as 12.7 mm.
* *CF*: If the user input time interval is equal to 60 min (hourly discretization of data), the user can choose whether or not to consider an R Factor conversion factor, to take into consideration the error induced by an hourly discretization. **True** indicates the program to use the conversion, **False** to ignore the conversion factor.

### Temperature Input

* *min\_temp*: minimum temperature threshold (in °C). Temperatures below the given value will not be considered as erosive, so they will not be taken into consideration when determining the start and end of a precipitation event.

### Original Raster Data

This data must be modified if rasters different from the dirt-X rasters are being used. The data correspond to the coordinates (location) and cell size of the rasters being used. This information is the same for all original input rasters (precipitation and temperature) being used.

* *original\_rows*: number of rows in the original precipitation and temperature raster files
* *original\_columnss*: number of columns in the original precipitation and temperature raster files
* *xllcorner*: X coordinate of the origin of the original precipitation and temperature raster files
* *yllcorner*: Y coordinate of the origin of the original precipitation and temperature raster files
* *cellsize*: cell size of the original precipitation and temperature raster files

### Directory Paths

* *path*: folder directory where the .csv files for each raster cell is located. The files must be comma-delimited .csv files (generated by RasterManipulation\_2 program).
* *save\_path\_storm*: folder directory where the .csv files with the per-storm R factor results will be saved (no need for input if *SaveCSV(df\_storm, name, save\_path\_storm* command is commented out, see Figure 4)
* *save\_path\_month*: folder directory where the .csv files with the monthly R-factor results will be stored. (no need for input if *SaveCSV(df\_storm, name, save\_path\_month* command is commented out, see Figure 4)
* *Save\_path\_raster*: file directory where the monthly and yearly R Factor result rasters for the entire watershed will be saved.

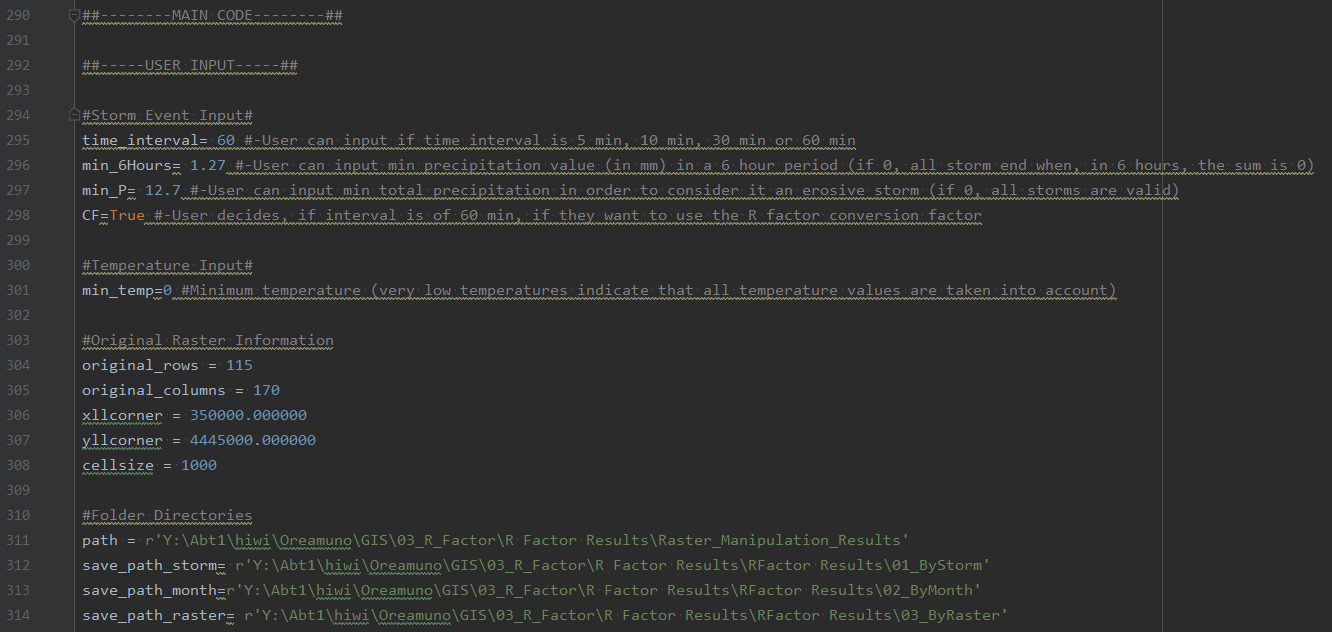


Figure . User input for program RFactor\_5

## Output data:

* **Per-Storm R Factor results**: Comma-delimited .csv file with the results for each storm in the given cell. File name corresponds to the original row\_column of the given cell. Includes storm start date, storm end date, total storm duration, total storm precipitation, maximum storm 30 minute intensity (mm/h), storm kinetic energy (KE), and storm R Factor (EI30) (see Figure 5)
  + Program generates a file for each cell in the original raster (2897 files)
  + User can decide to not generate the files by commenting the code “*SaveCSV(df\_storm, name, save\_path\_storm*) (see Figure 4)
* **Per-Month R Factor results**: Comma-delimited .csv file with the R factor result for each month in the given time frame, for each cell. Includes Month-Year and total R factor (see Figure 6).
  + Program generates a file for each cell in the original raster (2897 files)
  + User can decide to not generate the files by commenting the code “*SaveCSV(df\_storm, name, save\_path\_month*) (see Figure 4)
* **Raster files**: tab delimited .txt files (ASCII Raster format), for each month and for each year, with R factor results. Each cell contains the corresponding R Factor value.
  + Generated by command “*SaveRaster(R\_3D\_months, R\_3D\_years, save\_path\_raster)*” (see Figure 4)
  + Program generates file for each month-year combination and for each year in the given time frame.
  + File name corresponds to “month\_year” combination or “year”
  + In order to open the Raster files in ArcGIS, the command “ASCII to Raster” tool must be used.

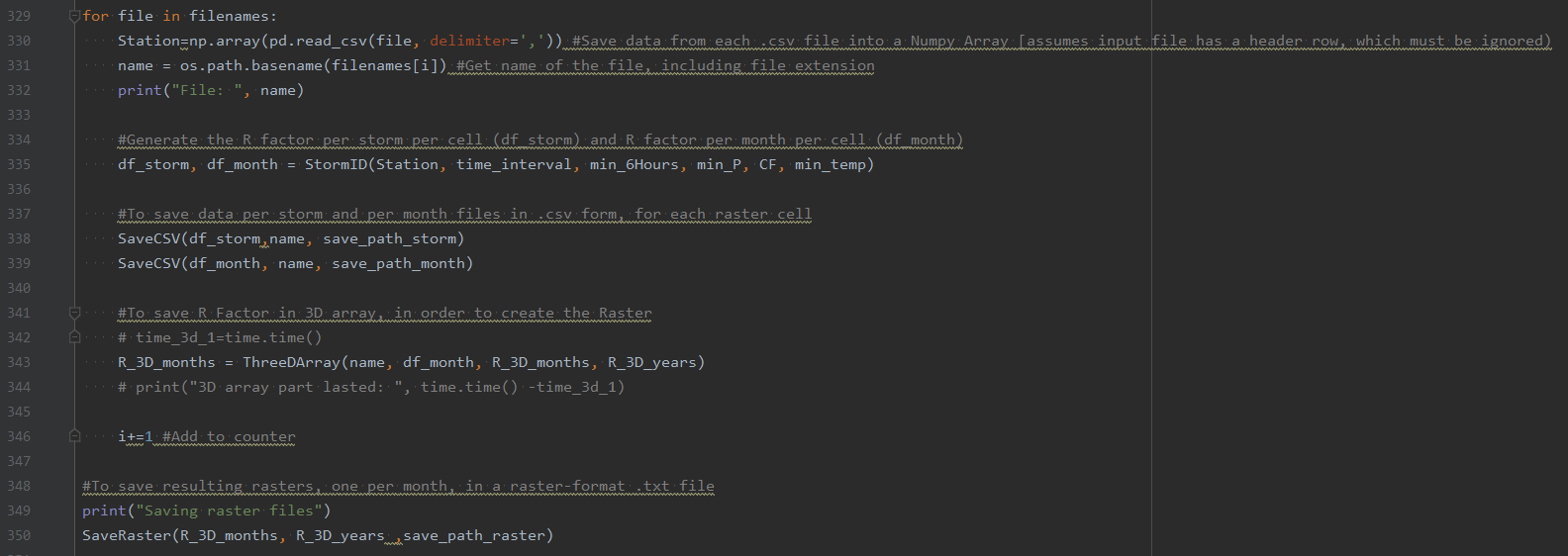


Figure . Code with Output Data commands

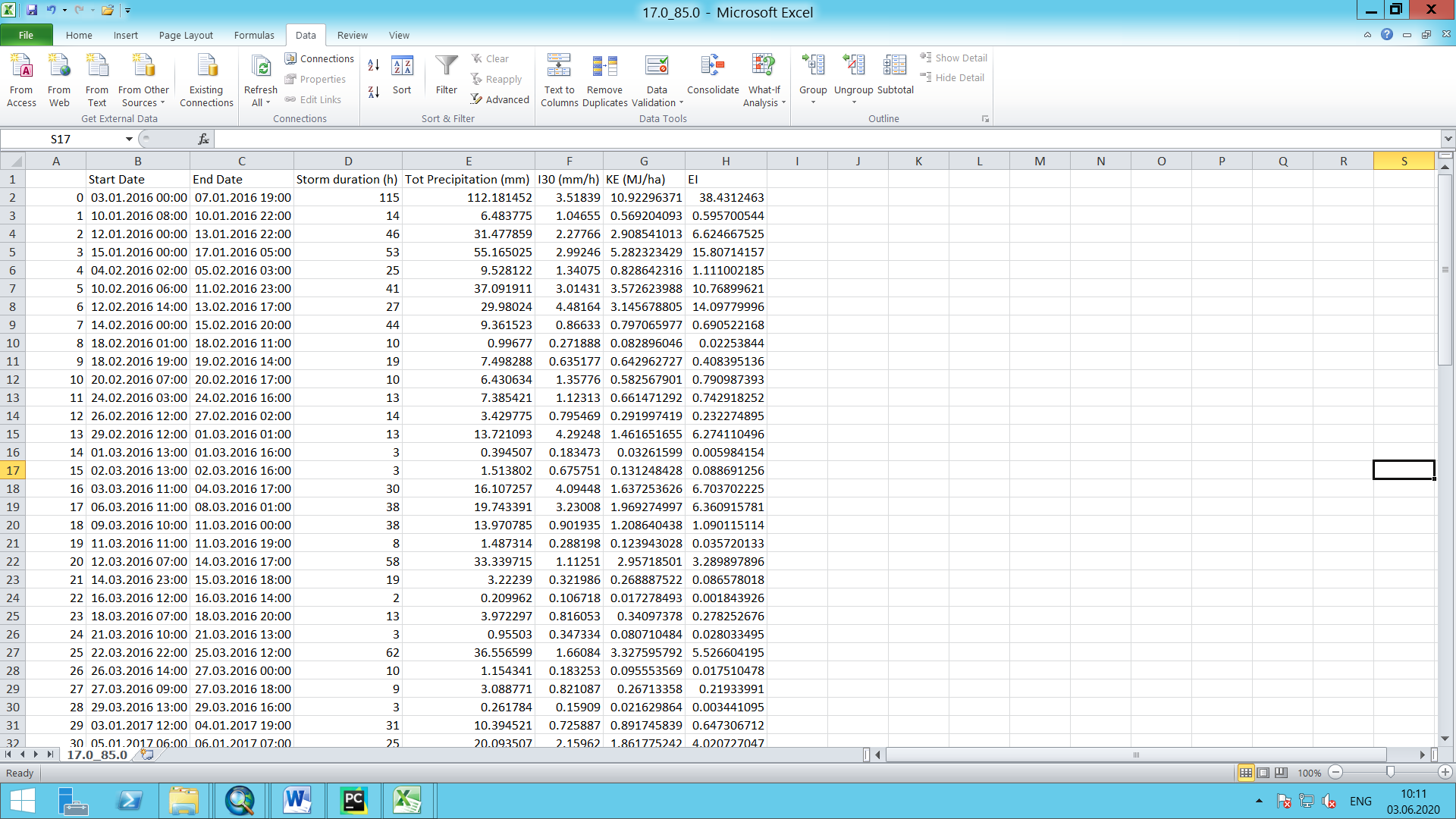


Figure . Per-Storm R Factor results example

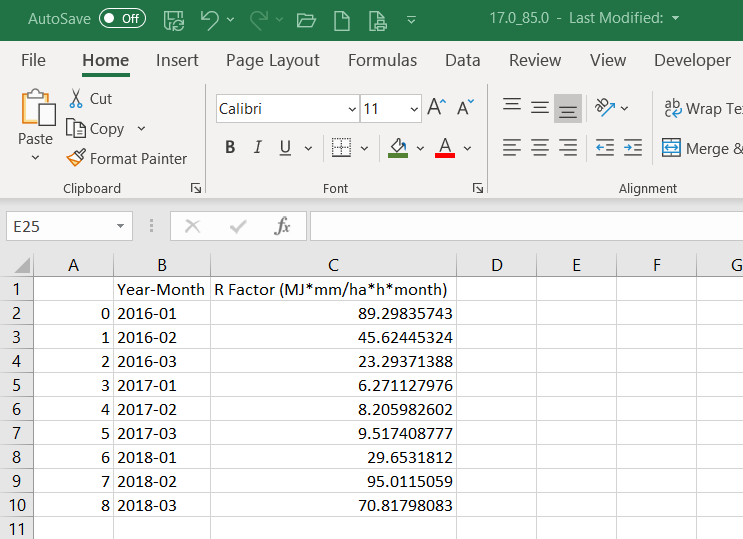


Figure . Per-Month R Factor results example

## Additional Information

* The program is independent of the RasterManipulation\_2 program in case the user wants to use the precipitation and temperature records of a given meteorological station instead of raster data. If this were the case, consider the following:
  + the input files must have the same configuration as the .csv files generated by the RasterManipulation\_2 program, without the “Original\_row” and “Original\_column” columns (see Figure 3).
  + The “*SaveRaster(R\_3D\_months, R\_3D\_years, save\_path\_raster*” command (see Figure 4) must be commented out, in order to not generate the output raster, which requires the input data to correspond to different cells.

## Error Checks

### CheckInput:

* Checks if the user input *time\_interval* is equal to either 10 min 15 min, 30 min or 60 min.
* Checks if the *min\_6Hours* and *min\_P* user input correspond to a number
* Checks that *CF* is “True” only if the user input time\_interval is equal to 60 min.

### CheckTimeInterval:

* Checks, for the first file in the file directory, if the user input “*time\_interva*l” coincides with the time interval in the file. It assumes the same result for all files in the directory. If this were not the case, an error will come up during the running of the program.

## Main functions in the program

### **MonthYear:**

Extracts the unique months and years in the input data files and saves them in different vectors (*unique\_months* and *unique\_years* respectively). These will later be used to generate the resulting raster files for each year and for each month-year combination. Additionally, it checks which years are complete years (have data for all 12 months). The results are printed out for the user to know which years are complete and which are incomplete.

* Receives a Data Frame with the data from the first data file in the file directory. The program assumes all input files have the same time frame.
  + If files don’t have the same time frame, and the other files include dates not contemplated in the first file, these dates won’t be considered in the final raster file generation. (Doesn’t affect the Per-Storm or Per Month calculations.

### **Main Loop:**

* Reads through the data files in the “path” folder
* Saves the name (which contains the row\_column combination for the given cell)
* Saves the data file in a Numpy Array
  + Program prints the name of each data file being read. This can be commented out.
* Calls “**StormID**” function, which returns two data frames, one with the R Factor results for each calculated storm (*df\_storm*, see Figure 5) and one with the R factors for each month (*df\_month*, see Figure 6).
* Calls “**SaveCSV**” function twice, for the *df\_storm* and *df\_month* data frames, in order to save the results as .csv files
  + These commands can be commented out if the user is not interested in these results.
* Calls “**ThreeDArray**”, which fills the monthly and yearly 3D matrices, and returns, for each loop, the updated 3D Arrays.
  + In each loop, the cell corresponding to the given data file, is updated in each array in the 3D array.

### **StormID**

* Receives:
  + *Station*: Numpy Array with data from input file
  + *time\_interval*: User input time\_interval
  + *min\_6Hours*: User input
  + *min\_P*: User input
  + *CF*: User Input
  + *Min\_temp*: User Input
* Loops through each row (each date) in *Station* in search for either the start or the end of a storm
  + A storm starts if, for a 6 hour time period, the total precipitation is equal or greater than *min\_6Hours*. If a storm starts, the variable “*hours”* becomes non zero, indicating that a storm has started.
    - A storm won’t start if, for a precipitation record, the temperature is below the temperature threshold (see explanation in section 2.5.4)
    - The row number where the storm starts is saved in variable *sd*.
  + A storm ends if, for a 6 hour time period, the total precipitation is less than *min\_6Hours*. Additionally, for a storm to end the recorded precipitation value must be equal to 0. (Panagos et.al) After a storm ends, the variable “*hours*” becomes 0, indicating that a new storm start must be determined.
    - The row number where the storm ends is saved in variable *ed*.
  + In order to determine if a storm starts or ends, the function calls the function **SumSixHours,** which returns the sum of the precipitation values for a 6 hour period, starting from the time being looped through.
  + When a storm ends, using the variables *sd* and *ed*, the total precipitation for the storm is calculated. If the total precipitation is greater than *min\_P* then the storm is considered as erosive. If the value is less than *min\_P*, then the storm data is not calculated.
    - Even though the temperature threshold is considered when determining if a storm starts or ends, it is not taken into consideration when adding the total precipitation for the storm event. This means that precipitation records with temperatures below the threshold are considered in the total precipitation value. (Taken according to C Code attached to Meusburger et. Al (2012) report).
  + For erosive storms, the maximum 30 minute intensity (I30), Kinetic Energy (KE) and R factor (EI30) are calculated for each storm, and stored in a data frame (*df\_storm*), calling the function **FillMatrix**. For a detailed explanation of the equations used see the section 2.6 in this document.
  + The function **RFacto**r is called, which sums the R factor for each month, using the *df\_storm* data frame, and saves it in a data frame (*df\_month).*

### **SumSixHours**

* Receives
  + *i*: row value being looped through
  + *Station*: Numpy array with data from file being looped through in **main loop**.
  + *min\_temp*: user input
  + *time\_interval*: user input
* Sums the precipitation values in a 6 hour period. Calls function **CalculateHour** to determine how many rows correspond to 6 hours, using user input *time\_interval*.
* If the temperature value of a given record is lower than *temp\_min*, the precipitation value is considered as 0, as below this threshold the precipitation is not considered to be erosive (snow).

### **ThreeDArray**

* Receives
  + *name*: file name, which contains the original row\_column of the cell corresponding to the file being looped through in the **main loop**.
  + *df\_month*: Data frame with R Factor for each month
  + *R\_3D\_months*: 3D array, originally created before the main loop and filled with the value -9999, and updated in each loop in the main loop. Has one array for each unique month in *unique\_months* vector, and each 2D array has the same number of rows (*original\_rows*) and columns (*original\_coumns*) as the original precipitation raster.
  + *R\_3D\_years*: 3D array, originally created before the main loop and filled with the value -9999, and updated in each loop in the main loop. Has one array for each unique year in *unique\_years* vector, and each 2D array has the same number of rows (*original\_rows*) and columns (*original\_coumns*) as the original precipitation raster.
* Function extracts the row and column location of the cell from the file name and saves it in a vector. This is the cell updated in each loop in the **main loop.**
* Function loops through each year in *unique\_years* vector and each month in *unique\_months* vector and finds the month-year combination in the *df\_month* data frame, and assigns the value to the cell in the corresponding array “*k*” in the 3D array R\_3D\_months.
  + These loops determine the order in which the data is saved in the 3D array, and is taken into consideration when naming each output raster.
* For each year in *unique\_years,* the function sums up the monthly R factors for the given year and assigns the value to the cell in the corresponding array “*y*” in the 3D array *R\_3D\_years*.

### SaveRaster

* Receives:
  + *R\_3D\_month*: 3D array, with one array for each month-year combination using *unique\_years* and *unique\_months* vectors, and filled in each loop in the main loop, through the function **ThreeDArray**.
  + *R\_3D\_years*: 3D array, with one array for each year in *unique\_years* and filled in each loop in the main loop, through the function **ThreeDArray**.
  + *Path*: *save\_path\_raster*, user input.
* Creates a data frame (*df\_head*) with information from the original raster, using the user input data for the original raster information. This is needed for the final raster to have an ASCII raster format and will be able to be read by a GIS program.
* Loops through each year in *unique\_years* vector and each month in *unique\_months* vector, in the same order as in the **ThreeDArray** function, and
  + Extracts each array “k” in the 3D array R\_3D\_month and converts it to a 2D array.
  + Concatenates df\_head with the 2D array into a single data frame and saves each data frame.
  + The file name corresponds to the year-month combination
* In each year “y” loop:
  + Extracts each array “y” in the 3D array R\_3D\_year and converts it to a 2D array.
  + Concatenates df\_head with the 2D array into a single data frame and saves each data frame.
  + The file name corresponds to the year

## Equations

### I30

* Corresponds to the maximum intensity in 30 minutes.
* Calculated in function **CalculateI30**, called from **StormID**.
* If *time\_interval* variable is less or equal to 30 min, calculates the rows corresponding to 30 minutes, sums the row precipitation values, and multiplies the sum by 2 to get intensity in mm/h.
* If *time\_interval* is 60 min, I30 is equal to the intensity in 60 min.

### Kinetic Energy (E)

* Corresponds to the total kinetic energy (in MJ/ha) for each storm.
* Calculatd in function **CalculateEnergy** called from **StormID**.
* First, it calculates the unit kinetic energy for each time step using the following equation (Panagos et.al, 2015)
* The total kinetic energy is equal to the sum of the individual unit kinetic energy value for each time step in the storm (Panagos et. Al, 2015):

### R Factor per storm

* Calculated in **CalculateEI** called from **StormID.**
* Calculates the R factor (in MJ\*mm/ha\*h) using the following equation (Panagos et.al, 2015)
* If *time\_interval* is equal to 60 min, and *CF* is True, the R factor is calculated using the following equation, applying a conversion factor to consider errors due to the larger time interval (Panagos et.al, 2015)

### R factor per month

* Calculated in function **RFactor**, called from **StormID**.
* Calculates the R Factor for each month

# References

Meusberger, K., Panagos, P., Montanarella, L., & Alewell, C. (2012). Spatial and temporal variability of rainfall erosivity factor for Switzerland. *Hydrology and Earth System Sciences*, 167-177.

Panagos, P., Ballabio, C., Borrelli, P., Meusburger, K., Klik, A., Rousseva, S., et al. (2015). Rainfall erosivity in Europe. *Science of the Total Environment*, 801-814.