$DD := \text{READEXCEL}(\text{“..\..\6Agro 2015 – II\Proyecto Productivo\Datos clima 26..02.2014 – 1..01.2015}$

As I don't have your directory structure, I've slightly generalized the file name by splitting up the name and path, then combining them using concat in the call to READEXCEL.

$filename := \text{“Datos clima 26..02.2014 – 1..01.2015.xlsx”}$
$filepath := \mathbf{CWD}$

$DT := \text{READEXCEL}(\text{concat}(filepath, \text{“\”}, filename))$ DT = Data Table; includes all of t

It's often more convenient to work with a data set that doesn't contain any extraneous information (such as headers). So, I've used submatrix to remove headers.

$DD := \text{submatrix}(DT, 4, \text{rows}(DT) - 1, 0, \text{cols}(DT) - 1)$ DD = Data Data; header informat

One other observation is that the data contains NaNs where the xlsx file contains no data (eg, 3rd row of th header and the final several rows of the data). There are a couple of ways of dealing with this. A quick ch NaN's locations using matchNaN .. makes it difficult to see what the range of rows containing is

$$\text{matchNaN}(DD)^{\mathrm{T}} = \left[\ldots \begin{bmatrix} 7418 \\ 0 \end{bmatrix} \begin{bmatrix} 7405 \\ 1 \end{bmatrix} \begin{bmatrix} 7406 \\ 1 \end{bmatrix} \begin{bmatrix} 7407 \\ 1 \end{bmatrix} \begin{bmatrix} 7408 \\ 1 \end{bmatrix} \begin{bmatrix} 7409 \\ \vdots \end{bmatrix} \begin{bmatrix} 7410 \\ \vdots \end{bmatrix} \begin{bmatrix} 7411 \\ \vdots \end{bmatrix} \begin{bmatrix} 7412 \\ \vdots \end{bmatrix} \right.$$

So write a function to convert a nested vector of vectors to a 2D matrix, where each column of the returned matrix is the corresponding row of the nested vectors.

$$nst2mat(N) := \left\| \begin{array}{l} M \leftarrow 0 \\ \text{for } k \in N \\ \quad \left\| M^{\langle \text{cols}(M) \rangle} \leftarrow k \right. \\ M^{\mathrm{T}} \end{array} \right.$$

Sorting on the first column should give the minimum row containing a NaN in the first row.

$$\text{csort}(nst2mat(\text{matchNaN}(DD)), 0)^{\mathrm{T}} = \begin{bmatrix} 7405 & 7405 & 7405 & 7405 & 7405 & 7405 & 7405 & 7405 & 7405 & 7405 & 740 \\ 31 & 25 & 18 & 20 & 30 & 29 & 12 & 22 & 24 & 2 & \end{bmatrix}$$

A look at the data matrix shows that these rows are all NaNs, so we can ignore them from an analysis point of view. We could use the filterNaN to remove these rows without affecting the GDD calculation. However, there might be other occasions where the data set contains NaNs in one or more columns but not all columns. Removing the NaN rows would affect the data, so we'll write our functions with this in mind, rather than using

would affect the data, so we'll write our functions with this in mind, rather than using filterNaN.

We're going to calculate GDD by first creating a matrix with each day's temperatures in seperate columns. Once we have this matrix, we'll apply a gdd function to each column thus returning a vector of GDD values for each day.

It is also possible that the need will arise to analyze other quantities apart from temperature. So, as the principal of maximal laziness says there's no point in replicating work, we'll write a general function, getData, to extract the quantity in a given column, skipping any NaNs.

```
getData(dcol) :=
    "// define the time column"
    ctime ← 0
    "// get the number of the first day"
    day ← floor(DD_{0, ctime})
    "// initialize day/data item matrix"
    MT ← 0
    "// initialize day/data item matrix row and column counters"
    r ← −1
    c ← 0
    "// iterate over the Data to convert to day/data item matrix"
    for k ∈ 0..rows(DD) − 1
        "// add each day's value to the matrix "
        if IsNaN(DD_{k, ctime})
            continue
        if floor(DD_{k, ctime}) = day
            r ← r + 1
        else
            day ← floor(DD_{k, ctime})
            r ← 0
            c ← c + 1
        MT_{r, c} ← DD_{k, dcol}
    MT
```

Assuming column 3 is the desired temperature column, we'll call getData and assign the result to matrix MT.

$ctemp := 3$

$MT := getData(ctemp)$ $\quad \text{rows}(MT) = 24 \quad \text{cols}(MT) = 310$

Inspection of MT shows that the first few values of the first day are zeros and thus should not contribute to the GDD calculation. The solution below may look at bit complex, but it gives a number of useful tools for manipulating data, which can be used to put the data in a format that is easier to process.

First, we'll create a filter function that applies a predicate function p to each element x of a vector v, and assigns x to the output if p(x) is true (ie, equal 1). An example would be creating an even function to filter out odd numbers

$$filter(p,v) := \left\| \begin{array}{l} w \leftarrow 0 \\ \text{for } x \in v \\ \quad \left\| \begin{array}{l} \text{if } p(x) \\ \quad \left\| w_{\text{rows}(w)} \leftarrow x \right. \end{array} \right. \\ w \end{array} \right.$$

$$even(n) := \text{mod}(n,2) = 0$$

$$v := \text{stack}(1,2,3,4,5,6,7,8,9)$$

$$filter(even,v)^{T} = [2 \;\; 4 \;\; 6 \;\; 8]$$

$$filter(\lambda(n) \leftarrow \text{mod}(n,2), v)^{T} = [1 \;\; 3 \;\; 5 \;\; 7 \;\; 9]$$

The last example is important to note as it uses a local function definition to create an "odd" function. As there is no particular need to reuse this function, it doesn't really need a name, so by convention we choose lambda to represent an anonymous function

"The base temperature for the specified crop is 5 °C"

We'll now use the filter function to create a nonzeros function that removes the zeros from a vector. This will allow us to calculate an effective GDD.

$$nonzeros(v) := filter(\lambda(x) \leftarrow x \neq 0, v) \qquad nonzeros(\text{stack}(1,0,3,0,5))^{T} = [1 \;\; 3 \;\; 5]$$

The next step in the process is to create a function that will return some aggregate value of a vector (such as min, max or mean). We can choose to aggregate along either across the rows or down the columns. We only need aggregate down the columns in this case, but we'll throw in the row aggragation function for completeness.

$$rowagg(f,M) := \left\| \begin{array}{l} \text{for } k \in 0..\text{rows}(M)-1 \\ \quad \left\| A_{k} \leftarrow f\left(M^{\widehat{k}}\right) \right. \\ A \end{array} \right.$$

$$colagg(f,M) := \left\| \begin{array}{l} \text{for } k \in 0..\text{rows}(M)-1 \\ \quad \left\| A^{\langle k \rangle} \leftarrow f\left(M^{\langle k \rangle}\right) \right. \\ A \end{array} \right.$$

And demontrate their working with min, max and mean on two matrices, one containing zeros and the other integers > 0

$$testmatrix := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad testmatrix2 := \begin{bmatrix} 1 & 0 & 3 \\ 0 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

$$rowagg(\text{mean}, testmatrix) = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix} \qquad colagg(\text{mean}, testmatrix) = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

$$rowagg(\lambda(v) \leftarrow min(v), testmatrix) = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} \qquad rowagg(\lambda(v) \leftarrow \max(v), testmatrix) = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}$$

$$minmax(v) := \left\| \begin{array}{l} v \leftarrow nonzeros(v) \\ [min(v) \quad \max(v)]^{T} \end{array} \right|$$

The function minmax return the minimum and maximum non zero values from a vector

GDD = (Max

$$colagg(minmax, testmatrix2) = \begin{bmatrix} 1 & 5 & 3 \\ 7 & 8 & 6 \end{bmatrix} \qquad minmax(\text{stack}(1, 0, 3, 0, 5)) = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

We can apply minmax to MT to get each day's minimum and maximum values.

$$colagg(minmax, MT) = \begin{bmatrix} 19.19 & 16.86 & 18.9 & 17.99 & 18.23 & 19.04 & 17.45 & 17.36 & 17.9 & 16.45 & 17.12 & 17.48 \\ 30.35 & 32.02 & 29.33 & 31.14 & 29.82 & 27.98 & 30.52 & 29.14 & 29.37 & 31.93 & 29.2 & 31.73 \end{bmatrix}$$

And, at last!, we create a function to calculate the GDD for each day

$$T_{base} := 5$$

$$gdd(v) := \text{mean}(minmax(v)) - T_{base}$$

$$colagg(gdd, MT) = [19.77 \quad 19.44 \quad 19.115 \quad 19.565 \quad 19.025 \quad 18.51 \quad 18.985 \quad 18.25 \quad 18.635 \quad 19.19 \quad 18.16 \quad 19.60$$

Simply transpose the result and assign to a variable to use it elsewhere

$$GDD := colagg(gdd, MT)^{T}$$

Note that the built-in aggregate functions don't ignore NaNs ...

$$\text{mean}(\text{stack}(1, 2, 3, NaN)) = NaN$$

... but we could have writtten function that do so.

$$Mean(v) := \text{mean}(\text{filterNaN}(v))$$

$$Mean(\text{stack}(1, 2, 3, NaN)) = 2$$