Livestock production imputation module

Author: Francesca Rosa

Description:

Inputs:

1. Production domain
2. Complete Key Table
3. Livestock Element Mapping Table
4. Identity Formula table
5. Share table
6. Elements code table
7. Range Carcass Weight table

Steps:

1. Impute Livestock Numbers

2. Impute Number of Slaughtered animal (assiciated to the animal item)

3. Transfer the animal slaughtered from animal commodity (parent) to the meat commodity (child)

4. Impute the meat triplet (production/animal slaughtered/carcass weight) based on the same logic as all other production imputation procedure.

5. Transfer the slaughtered animal from the meat back to the animal, as now certain slaughtered animal is imputed in step 3.

6. Transfer the slaughtered animal from the animal to all other child commodities. This includes items such as offals, fats and hides and impute missing values for non-meat commodities.

Initialisation

Load the necessary packages:

```
message("Step 0: Setup")

##' Load the libraries
suppressMessages({
    library(data.table)
    library(faosws)
    library(faoswsFlag)
    library(faoswsUtil)
    library(faoswsImputation)
    library(faoswsProduction)
    library(faoswsProcessing)
    library(faoswsEnsure)
    library(magrittr)
    library(dplyr)
    library(sendmailR)

})
```

Get the shared path and read the session info in case we are working locally (not throught the R plugin in the SWS):

```
R_SWS_SHARE_PATH = Sys.getenv("R_SWS_SHARE_PATH")

if(CheckDebug()){

    library(faoswsModules)
    SETTINGS = ReadSettings("modules/animal_stockFRANCESCA/sws.yml")

    ## If you're not on the system, your settings will overwrite any others
    R_SWS_SHARE_PATH = SETTINGS[["share"]]

    ## Define where your certificates are stored
    SetClientFiles(SETTINGS[["certdir"]])

    ## Get session information from SWS. Token must be obtained from web interface
    GetTestEnvironment(baseUrl = SETTINGS[["server"]],
                       token = SETTINGS[["token"]])

}
```

Create the directory in the share folder where the intermediate tabkes are saved for validation purposes (for example off-take rates can be browsed and validated looking at the following directory):

```
dir_to_save <- file.path(R_SWS_SHARE_PATH, "Livestock",
                         paste0("validation", gsub("/", "_",swsContext.username)))

if(!file.exists(dir_to_save)){
    dir.create(dir_to_save, recursive = TRUE)
}
```

Load and check the computation parameters. The parameters that have been set by the user in creting the "new debug session" (in case you are working locally, basically when the token has been created) or when the plugin has been launched (in case you are working directly on the SWS).

The "imputation selection" controls the commodities the module has to work on. ALL means that the module will loop on all the 17 animal items, while SESSION indicates that the module will loop only on the items queried in the current session.

The "imputation time window" allows to produce new imputations only for the last three available years.

```
imputationSelection = swsContext.computationParams$imputation_selection
if(!imputationSelection %in% c("session", "all"))
    stop("Incorrect imputation selection specified")



imputationTimeWindow = swsContext.computationParams$imputation_timeWindow
if(!imputationTimeWindow %in% c("all", "lastThree"))
    stop("Incorrect imputation selection specified")
```

Get data configuration and session:

```
sessionKey = swsContext.datasets[[1]]
datasetConfig = GetDatasetConfig(domainCode = sessionKey@domain,
                                 datasetCode = sessionKey@dataset)
```

Build processing parameters:

```
processingParameters =
    productionProcessingParameters(datasetConfig = datasetConfig)
```

Obtain the complete imputation key (this function pull the complete imputation key from a data-table stored in SWS: fbs_production_comm_codes) :

```
completeImputationKey = getCompleteImputationKey("production")
```

Extract the correspondence between the animal-parent and meat/non-meat-child items. This information is containend in the animal_parent_child_mapping table.

This table contains the parent (both item and element codes) which maps to the child item/element code. For example, the slaughtered animal element for cattle is 5315, while the slaughtered animal for cattle meat is 5320.

Ideally, the two elements should be merged and have a single code in the classification. This will eliminate the change of code in the transfer procedure.

```
animalMeatMappingTable = ReadDatatable("animal_parent_child_mapping")
```

When the animal_parent_child_mapping table has been created in the SWS it was not possible to insert capital letters in the datatbla header. That's why the datatable stored in the SWS is not compliant (in terms of column names) with the other dataset stored in the SWS.

We change tha column names in order to ensure the compliancy between all the datatable the module works on (at least in terms of column names):

```
setnames(animalMeatMappingTable,c("measured_item_parent_cpc",
                                  "measured_element_parent",
                                  "measured_item_child_cpc",
                                  "measured_element_child"),
                                c("measuredItemParentCPC",
                                  "measuredElementParent",
                                  "measuredItemChildCPC",
                                  "measuredElementChild"))

animalMeatMappingTable= animalMeatMappingTable[,.(measuredItemParentCPC,
                                                  measuredElementParent,
                                                  measuredItemChildCPC,
                                                  measuredElementChild)]
```

Here we expand the session to include all the parent and child items. That is, we expand to the particular livestock tree.

For example, if 02111 (Cattle) is in the session, then the session will be expanded to also include 21111.01 (meat of cattle, freshor chilled), 21151 (edible offal of cattle, fresh, chilled or frozen), 21512 (cattle fat, unrendered), and 02951.01 (raw hides and skins of cattle).

The elements are also expanded to the required triplet (according to different typology of item).

```
livestockImputationItems =
    completeImputationKey %>%
    expandMeatSessionSelection(oldKey = .,
                               selectedMeatTable = animalMeatMappingTable) %>%
    getQueryKey("measuredItemCPC", datasetkey = .) %>%
    selectMeatCodes(itemCodes = .)

sessionItems =
    sessionKey %>%
    expandMeatSessionSelection(oldKey = .,
                               selectedMeatTable = animalMeatMappingTable) %>%
    getQueryKey("measuredItemCPC", datasetkey = .) %>%
    selectMeatCodes(itemCodes = .)
```

Select the range of items based on the computational parameter: this means that according to the parameters set by the user in opening a new debug session, we select the commodities the module has to work on (all or only the commodities included in the session.)

```
selectedMeatCode =
    switch(imputationSelection,
           session = sessionItems,
           all = livestockImputationItems)
```

The "last year" is a parameter used in many functions: since the user can choose if produce new imputations starting from 1990 or just for the last three years, we need to specify the last year for which we want to produce imputations in order to authomatically go three years back in overwriting non-protected figures.

```
lastYear=max(as.numeric(completeImputationKey@dimensions$timePointYears@keys))
```

When we work locally, we might be interested in deviate the "console output" on a txt file that can be properly read and browsed:

```
if( CheckDebug())
{logConsole1=file("log.txt",open = "w")
sink(file = logConsole1, append = TRUE, type = c( "message"))}
```

This empty datatable is created to host the list of item that may eventually report some problems. The user will recieve an email with the list of items for which the module failed.

```
imputationResult = data.table()
```

Perform Synchronisation and Imputation.

Here we iterate through the 17 meat items to perform the steps mentioned in the description. Essentially, we are looping over different livestock trees.

```
for(iter in seq(selectedMeatCode)){

    imputationProcess =    try({
```

```
    message("Processing livestock tree (", iter, " out of ",
            length(selectedMeatCode), ")")
    set.seed(070416)
```

Extract the current ANIMAL, MEAT and NON-MEATcodes with their relative formula and mapping-table with shares to properly tranfer data from the parent to the child element (as already mensioned the elements referring to "number of slaughtered animals" are different in terms of code if it refers to the animal item or to the meat item, that's why we need to manage this correspondence):

```
    #Meat
    currentMeatItem = selectedMeatCode[iter]

    currentMappingTable =
        animalMeatMappingTable[measuredItemChildCPC == currentMeatItem, ]

    #animal
    currentAnimalItem = currentMappingTable[, measuredItemParentCPC]

    #All derived
    currentAllDerivedProduct =
    animalMeatMappingTable[measuredItemParentCPC == currentAnimalItem,measuredItemChildCPC]

    #Derived non meat
    currentNonMeatItem =
        currentAllDerivedProduct[currentAllDerivedProduct != currentMeatItem]

    message("Extracting the shares tree")
    shareData =
        getShareData(geographicAreaM49 =
                         getQueryKey("geographicAreaM49", completeImputationKey),
                     measuredItemChildCPC = currentAllDerivedProduct,
                     measuredItemParentCPC = currentAnimalItem,
                     timePointYearsSP =
                         getQueryKey("timePointYears", completeImputationKey)) %>%
        setnames(x = .,
                 old = c("Value", "timePointYearsSP"),
                 new = c("share", "timePointYears")) %>%
        mutate(timePointYears = as.numeric(timePointYears))
    shareData=as.data.table(shareData)
```

Get the animal data from the SWS:

```
    message("Extracting animal data ", currentAnimalItem,
            " (Animal)")

    #Get the animal formula
    animalFormulaTable =
        getProductionFormula(itemCode = currentAnimalItem) %>%
        removeIndigenousBiologicalMeat(formula = .)

    if(nrow(animalFormulaTable) > 1){
        stop("Imputation should only use one formula")}
```

Create the formula parameter list:

```
animalFormulaParameters =
    with(animalFormulaTable,
         productionFormulaParameters(datasetConfig = datasetConfig,
                                     productionCode = output,
                                     areaHarvestedCode = input,
                                     yieldCode = productivity,
                                     unitConversion = unitConversion))
```

Get the animal key, we take the complete key and then modify the element and item dimension to extract the current meat item and its corresponding elements.

It is not necessary to extract the triplet, but just Livestock and Slaughtered, the element that should play the role of the YIEL is, in this case the off-take rate that is endogenously computed (eventually using trade) and then imputed.

```
animalKey = completeImputationKey
animalKey@dimensions$measuredItemCPC@keys = currentAnimalItem
animalKey@dimensions$measuredElement@keys =
    with(animalFormulaParameters,
         c(productionCode))
```

Get the animal data (NB: the function preProcessing transforms the (0, M) figures pulled from the SWS into (NA, M) and transforms timePointYears in a numeric column while it a character column in the SWS).

```
animalData =
    animalKey %>%
    GetData(key = .) %>%
    preProcessing(data = .)
```

We remove all the non-protected figured that need to be overwritten.

The following "if"-condition allows to use also the NON-protected data to build the new imputations in case the time_window_imputation is equal to "last three years".

```
if(imputationTimeWindow=="all"){animalData=removeNonProtectedFlag(animalData)}else{
    animalData=removeNonProtectedFlag(animalData, keepDataUntil = (lastYear-2))}

animalData= expandYear(data = animalData,
                       areaVar = processingParameters$areaVar,
                       elementVar = processingParameters$elementVar,
                       itemVar = processingParameters$itemVar,
                       valueVar = processingParameters$valueVar,
                       newYears=lastYear)
```

The original idea was to include the TRADE data into the livestock imputation process. The basic hypothesis is that Countries import livestock for slaughtering purposes.

We made several test including and excluding trade data which in many cases was the source of outliers in the final results.

Apparently non feasible fluctuations into the meat production had been produced. That's why trade data had been finally excluded from the module.

We kept in the module the possibility to easily re-add the trade component.

```
itemMap = GetCodeList(domain = "agriculture", dataset = "aproduction", "measuredItemCPC")
itemMap = itemMap[,.(code,type)]
setnames(itemMap, "code", "measuredItemCPC")
data = merge(animalData, itemMap, by="measuredItemCPC")
```

This two lines contains info on the trade elements to be pulled in case it will be decided in the future to use trade to compute the number of animal Slaughtered

```
#itemCodeKey = ReadDatatable("element_codes")
#tradeElements = itemCodeKey[itemtype== unique(data[,type]),c(imports, exports)]
#factor= itemCodeKey[itemtype== unique(data[,type]),c(factor)]

#this is a conversion factor to be used in computing one element
#of the triplet from the others as identity I prefer to get the
#conversion factor from the data table: item_type_yield_elements
#which is the same where also the fuction getProductionFormula takes it.

getFactor = ReadDatatable(table = "item_type_yield_elements")
factor= getFactor[item_type== unique(data[,type]),c(factor)]
```

Pull trade data for the current Animal Item
In case you decide to use trade data: build the key using the most updated dataset!!!!

```
    ##   tradeData <- GetData(key = key)
    ##
    ##   setnames(tradeData, c("measuredElementTrade", "measuredItemCPC"),
    ##            c("measuredElement", "measuredItemCPC"))
    ##
    ##
    ##   tradeData=preProcessing(tradeData)
    ##
    ##   stockTrade=rbind(tradeData, animalData)
    ## At the moment it has been decided to NOT use trade data
    ##    stockTrade=animalData
    ##    stockTrade=denormalise(stockTrade,
    ##                           denormaliseKey = "measuredElement",
    ##                           fillEmptyRecords=TRUE )
```

Imputation of animal Stock

To impute livestock numbers we follow excatly the same approach (the ensemble approach) already developped. Here we are building the parameters :

```
animalData=denormalise(animalData, denormaliseKey = "measuredElement", fillEmptyRecords=TRUE )

animalStockImputationParameters=defaultImputationParameters()

## I am modifing the animalStockImputationParameters in order to specify
## that the variable to be imputed is the livestock (5111 for big animals, 5112 for small animals)
animalStockImputationParameters$imputationValueColumn=
        animalFormulaParameters$productionValue

animalStockImputationParameters$imputationFlagColumn=
        animalFormulaParameters$productionObservationFlag

animalStockImputationParameters$imputationMethodColumn=
        animalFormulaParameters$productionMethodFlag

animalStockImputationParameters$byKey=c("geographicAreaM49","measuredItemCPC")
animalStockImputationParameters$estimateNoData=TRUE

##This code is to see the charts of the emsemble approach
##animalStockImputationParameters$plotImputation="prompt"

message("Step 1: Impute missing values for livestock: item ", currentAnimalItem,
            " (Animal)")

stockImputed=imputeVariable(animalData,
                            imputationParameters = animalStockImputationParameters)
```

Imputation of number of slaughtered animals

Pull slaughtered Animail (code referrig to ANIMAL):

```
slaughterdKey=animalKey
slaughterdKey@dimensions$measuredElement@keys= with(animalFormulaParameters,
                                                    c(areaHarvestedCode))

slaughteredAnimalData =
        slaughterdKey %>%
        GetData(key = .) %>%
        preProcessing(data = .)

if(imputationTimeWindow=="all"){

slaughteredAnimalData=removeNonProtectedFlag(slaughteredAnimalData)}else{
slaughteredAnimalData=removeNonProtectedFlag(slaughteredAnimalData,
                                             keepDataUntil = (lastYear-2))}

slaughteredAnimalData= removeNonProtectedFlag(slaughteredAnimalData) %>%
        expandYear(data = .,
                   areaVar = processingParameters$areaVar,
                   elementVar = processingParameters$elementVar,
                   itemVar = processingParameters$itemVar,
                   valueVar = processingParameters$valueVar,
                   newYears=lastYear)

slaughteredAnimalData=denormalise(slaughteredAnimalData,
                                  denormaliseKey = "measuredElement",
                                  fillEmptyRecords=TRUE)
```

Prepare the table to be used to compute TOT slaughtered Animal: this approach has been follow in order to use trade data. In theory for some countries it would have been necessary to compute the Total number of animal slaughterd including the trade flows.

For some countries we may have slaughtered AnimalData, but not stockImputed Be careful with this merge:

```
stockSlaughtered=merge(stockImputed, slaughteredAnimalData,
                       by=c( "geographicAreaM49", "measuredItemCPC", "timePointYears"),
                       all.x =  TRUE,
                       all.y =  TRUE)



message("Step 2: Impute Number of Slaughtered animal for ",
        currentAnimalItem," (Animal)")
```

```
slaughteredParentData=computeTotSlaughtered(data =stockSlaughtered,
                                            FormulaParameters=animalFormulaParameters)
```

Before Saving this data in the shared folder I change the off-take method flag which is: “i”. It is now “c” because it was useful to protect it.

```
slaughteredParentData[TakeOffRateFlagMethod=="c", TakeOffRateFlagMethod:="i"]


if(!CheckDebug()){
write.csv(slaughteredParentData,
          file.path(dir_to_save, paste0("LivestockTriplet_",
                                        currentAnimalItem ,
                                        ".csv")), row.names = FALSE)
    }


slaughteredParentData = slaughteredParentData[,c("geographicAreaM49",
                                                 "measuredItemCPC",
                                                 "timePointYears",
                                      animalFormulaParameters$areaHarvestedValue,
                                      animalFormulaParameters$areaHarvestedObservationFlag,
                                      animalFormulaParameters$areaHarvestedMethodFlag),
                                      with=FALSE]


slaughteredParentData=normalise(slaughteredParentData, removeNonExistingRecords=FALSE)
```

message(“Step 3: Transfer animal slaughtered back from meat to animal commodity”)

Transfer the animal slaughtered from meat back to animal, this can be done by specifying parentToChild equal to FALSE.

We only subset the new calculated or imputed values to be transfer back to the animal (parent) commodity.

Since the animal element is not imputed nor balanced , we will not test whether it is imputed or the identity calculated.

Filter meatImputed:

```
    meatImputedFilterd =
        meatImputed[flagMethod == "i" | (flagObservationStatus == "I" & flagMethod == "e"), ]
```

```
slaughteredTransferedBackToAnimalData=transferParentToChild(parentData = slaughteredParentData,
                                                              childData = meatImputedFilterd,
                                                              mappingTable = animalMeatMappingShare,
                                                              transferMethodFlag="c",
                                                              imputationObservationFlag = "I",
                                                              parentToChild = FALSE)
```

Not all the tranfered figures have to be sent back to the SWS, bacause there are situation where only the flag is changed, and it would be better to keep the protected flag combination coming from the parent-data "slaughteredParentData"

```
    ensureProductionOutputs(data = meatImputed,
                            processingParameters = processingParameters,
                            formulaParameters = meatFormulaParameters,
                            testImputed = FALSE,
                            testCalculated = FALSE,
                            normalised = TRUE,
                            returnData = FALSE)
```

```
    livestockNumbers=normalise(stockImputed)
    message("Saving the synchronised and imputed data back")


    syncedData = rbind(meatImputed,
                       livestockNumbers,
                       slaughteredTransferedBackToAnimalData
    )
```

Send back also the (M,-) series otherwise it seems they are not updated!

```
syncedData=syncedData[(flagMethod!="u"),]
```

The transfer can over-write official and semi-official figures in the processed commodities as indicated by in the previous synchronise slaughtered module.

```
if(imputationTimeWindow=="lastThree")
{syncedData=syncedData[get(processingParameters$yearVar) %in% c(lastYear,
                                                                 lastYear-1,
                                                                 lastYear-2)]


syncedData=postProcessing(data =  syncedData)
syncedData=removeInvalidDates(syncedData)
ProtectedOverwritten=ensureProtectedData(syncedData[(flagObservationStatus=="I" &
                                                     flagMethod=="e") |
                                                     flagMethod=="i"|
                                                     flagMethod=="c",],
                                          getInvalidData = TRUE)

ProtectedOverwritten=ProtectedOverwritten[measuredElement!=
                                           imputationParameters$areaHarvestedParams$variable,]

ProtectedOverwritten=ProtectedOverwritten[Value!=i.Value]
SaveData(domain = sessionKey@domain,
             dataset = sessionKey@dataset,
             data = syncedData)
}else{
syncedData=postProcessing(data =  syncedData)
syncedData=removeInvalidDates(syncedData)

ProtectedOverwritten=ensureProtectedData(syncedData[(flagObservationStatus=="I" &
                                                     flagMethod=="e") |
                                                     flagMethod=="i"|
                                                     flagMethod=="c",],
                                         getInvalidData = TRUE)


ProtectedOverwritten=ProtectedOverwritten[measuredElement!=
                                           imputationParameters$areaHarvestedParams$variable]

ProtectedOverwritten=ProtectedOverwritten[Value!=i.Value]

SaveData(domain = sessionKey@domain,
             dataset = sessionKey@dataset,
             data = syncedData)
    }
```

```
    if(!CheckDebug() & length(ProtectedOverwritten)>0){



        createErrorAttachmentObject = function(testName,
                                               testResult,
                                               R_SWS_SHARE_PATH){
            errorAttachmentName = paste0(testName, ".csv")
            errorAttachmentPath =
                paste0(R_SWS_SHARE_PATH, "/rosa/", errorAttachmentName)
            write.csv(testResult, file = errorAttachmentPath,
                      row.names = FALSE)
            errorAttachmentObject = mime_part(x = errorAttachmentPath,
                                              name = errorAttachmentName)
            errorAttachmentObject
        }

        bodyWithAttachment=
            createErrorAttachmentObject(paste0("ToBeChecked_", currentMeatItem),
                                        ProtectedOverwritten,
                                        R_SWS_SHARE_PATH)


        sendmail(from = "sws@fao.org",
                 to = swsContext.userEmail,
                 subject = "Some protected figures have been overwritten",
                 msg = bodyWithAttachment)

    }
```

Now that we have computed and synchronized all the slaughtered we can proceed computig other derived items:

```
 if(length(currentNonMeatItem) > 0){
     nonMeatImputedList=list()


     message("Step 6: Transfer the slaughtered animal from the animal to all other child
             commodities. This includes items such as offals, fats and hides and
             impute missing values for non-meat commodities.")
```

Different triplet for different non-meat items, we need to loop through the different non-meat items:

```
     for(j in seq(currentNonMeatItem)){
         currentNonMeatItemLoop= currentNonMeatItem[j]

         message("Extracting production triplet for item ",
                 paste0(currentNonMeatItemLoop, collapse = ", "),
                 " (Non-meat Child)")
```

Get the non Meat formula:

```
         currentNonMeatFormulaTable =
             getProductionFormula(itemCode = currentNonMeatItemLoop) %>%
             removeIndigenousBiologicalMeat(formula = .)
```

Build the non meat key:

```
currentNonMeatKey = completeImputationKey
currentNonMeatKey@dimensions$measuredItemCPC@keys = currentNonMeatItemLoop
currentNonMeatKey@dimensions$measuredElement@keys =
with(currentNonMeatFormulaTable,
                   unique(c(input, output, productivity)))


nonMeatMeatFormulaParameters =
              with(currentNonMeatFormulaTable,
                   productionFormulaParameters(datasetConfig = datasetConfig,
                                               productionCode = output,
                                               areaHarvestedCode = input,
                                               yieldCode = productivity,
                                               unitConversion = unitConversion)
              )
```

Get the non meat data

```
         nonMeatData =
             currentNonMeatKey %>%
             GetData(key = .) %>%
             preProcessing(data = .) %>%
             denormalise(normalisedData = .,
                         denormaliseKey = "measuredElement") %>%
             createTriplet(data = .,
                           formula = currentNonMeatFormulaTable)
```

We have to remove (M,-) from the carcass weight: since carcass weght is usually computed ad identity, it results inutial that it exists a last available protected value different from NA and when we perform the function expandYear we risk to block the whole time series. I replace all the (M,-) carcass wight with (M,u) the triplet will be sychronized by the imputeProductionTriplet function.

```r
nonMeatData[get(nonMeatMeatFormulaParameters$yieldObservationFlag)==
                processingParameters$missingValueObservationFlag,

            ":="(c(nonMeatMeatFormulaParameters$yieldMethodFlag),
                 list(processingParameters$missingValueMethodFlag)) ]


nonMeatData = normalise(denormalisedData =nonMeatData,
                        removeNonExistingRecords = FALSE)

nonMeatData= expandYear(data = nonMeatData,
                        areaVar = processingParameters$areaVar,
                        elementVar = processingParameters$elementVar,
                        itemVar = processingParameters$itemVar,
                        ValueVar = processingParameters$valueVar,
                        newYears=lastYear)

message("Transfer Animal Slaughtered to All Child Commodities")

nonMeatMappingTable =
     animalMeatMappingTable[measuredItemChildCPC %in% currentNonMeatItemLoop, ]

animalNonMeatMappingShare =
              merge(nonMeatMappingTable, shareData, all.x = TRUE,
                    by = c("measuredItemParentCPC", "measuredItemChildCPC"))
```

In this tipology of commodity, there are still present old FAOSTAT imputations flagged as (I,-). At the moment the best we can do is to keep those figures as protected. We delete the figures flagged ad (I,e) end computed ad identity figures (method="i") coming from previus run of themodule:

```r
modifiedFlagTable=copy(flagValidTable)
modifiedFlagTable[flagObservationStatus=="I" & flagMethod=="-" , Protected:=TRUE]


if(imputationTimeWindow=="all"){nonMeatData=removeNonProtectedFlag(nonMeatData,
                                                  flagValidTable = modifiedFlagTable)}else{
                                nonMeatData=removeNonProtectedFlag(nonMeatData,
                                                  flagValidTable = modifiedFlagTable,


nonMeatData[measuredElement==nonMeatMeatFormulaParameters$areaHarvestedCode,
            ":="(c("Value",
                   "flagObservationStatus",
                   "flagMethod"),
                 list(NA_real_,"M","u"))]
```

Syncronize slaughteredTransferedBackToAnimalData to the slaughtered element associated to the non-meat item:

```
          slaughteredTransferToNonMeatChildData =
              transferParentToChild(parentData = slaughteredTransferedBackToAnimalData,
                                    childData = nonMeatData,
                                    transferMethodFlag="c",
                                    imputationObservationFlag = "I",
                                    mappingTable = animalNonMeatMappingShare,
                                    parentToChild = TRUE)



          nonMeatImputationParameters=
              with(currentNonMeatFormulaTable,
                   getImputationParameters(productionCode = output,
                                           areaHarvestedCode = input,
                                           yieldCode = productivity)
              )
```

Imputation without removing all the non protected figures for Production and carcass weight!

Some checks are requested because we cannot remove all the non protected values.

1. SLAUGHTERED: synchronized
2. YIELD: to stabilize imputations I have to keep non-protected figures
3. Non-MEAT PRODUCTION: remove non-protected figures, computed as IDENTITY (where possible), IMPUTED

```
slaughteredTransferToNonMeatChildData=denormalise(slaughteredTransferToNonMeatChildData,
                                                  denormalise="measuredElement",fillEmptyRecords=TRUE )
```

In addition, since the number of animal slaugheterd might have changed, we delete also the the figures previously calculated ad identity (flagMethod="i") if also production is available

remove those yields where both PRODUCTION and SLAUGHTERED are not NA:

```
noNAProd=slaughteredTransferToNonMeatChildData[,
                            !is.na(get(nonMeatMeatFormulaParameters$productionValue))]

noNASlaughterd=slaughteredTransferToNonMeatChildData[,
                            !is.na(get(nonMeatMeatFormulaParameters$areaHarvestedValue))]

filter= noNAProd & noNASlaughterd

slaughteredTransferToNonMeatChildData[filter,":="(c(nonMeatMeatFormulaParameters$yieldValue,
                                                    nonMeatMeatFormulaParameters$yieldObservationFlag,
                                                    nonMeatMeatFormulaParameters$yieldMethodFlag),
                                      list(NA_real_,"M","u"))]

nonMeatImputed = imputeProductionTriplet(data = slaughteredTransferToNonMeatChildData,
                                         processingParameters = processingParameters,
                                         imputationParameters = nonMeatImputationParameters,
                                         formulaParameters = nonMeatMeatFormulaParameters)

nonMeatImputedList[[j]] = normalise(nonMeatImputed)
```

```
slaughteredTransferToNonMeatChildData=rbindlist(nonMeatImputedList)


slaughteredTransferToNonMeatChildData=slaughteredTransferToNonMeatChildData[flagMethod!="u", ]

if(imputationTimeWindow=="lastThree"){
slaughteredTransferToNonMeatChildData=
slaughteredTransferToNonMeatChildData[get(processingParameters$yearVar)
                        %in% c(lastYear, lastYear-1, lastYear-2)]

slaughteredTransferToNonMeatChildData=removeInvalidDates(data=slaughteredTransferToNonMeatChildData,
                                                         context = sessionKey)

slaughteredTransferToNonMeatChildData=postProcessing(data =  slaughteredTransferToNonMeatChildData)


SaveData(domain = sessionKey@domain,
                  dataset = sessionKey@dataset,
                  data = slaughteredTransferToNonMeatChildData)
          }else{

slaughteredTransferToNonMeatChildData=postProcessing(data =  slaughteredTransferToNonMeatChildData)

slaughteredTransferToNonMeatChildData=removeInvalidDates(data = slaughteredTransferToNonMeatChildData,
                                                           context = sessionKey)

slaughteredTransferToNonMeatChildData=postProcessing(data =  slaughteredTransferToNonMeatChildData)

SaveData(domain = sessionKey@domain,
                  dataset = sessionKey@dataset,
                  data = slaughteredTransferToNonMeatChildData)
          }

      }


  }


  message("\nSynchronisation and Imputation Completed for\n",
            "Animal Parent: ", currentAnimalItem, "\n",
            "Meat Child: ", currentMeatItem, "\n",
            "Non-meat Child: ", paste0(currentNonMeatItem, collapse = ", "), "\n",
            rep("-", 80), "\n")


    })
```

Capture the items that failed :

```
    if(inherits(imputationProcess, "try-error"))
        imputationResult =
        rbind(imputationResult,
               data.table(item = currentItem,
                          error = imputationProcess[iter]))


}



## Initiate email
from = "sws@fao.org"
to = swsContext.userEmail
subject = "Livestock module"
body = paste0("Livestock production module successfully ran.
              You can browse results in the session: ", sessionKey@sessionId )
sendmail(from = from, to = to, subject = subject, msg = body)




##' ## Return Message

if(nrow(imputationResult) > 0){
    ## Initiate email
    from = "sws@fao.org"
    to = swsContext.userEmail
    subject = "Imputation Result"
    body = paste0("The following items failed, please inform the maintainer "
                  , "of the module")

    errorAttachmentName = "non_livestock_imputation_result.csv"
    errorAttachmentPath =
        paste0(R_SWS_SHARE_PATH, "/kao/", errorAttachmentName)
    write.csv(imputationResult, file = errorAttachmentPath,
              row.names = FALSE)
    errorAttachmentObject = mime_part(x = errorAttachmentPath,
                                      name = errorAttachmentName)

    bodyWithAttachment = list(body, errorAttachmentObject)

    sendmail(from = from, to = to, subject = subject, msg = bodyWithAttachment)
    stop("Production imputation incomplete, check following email to see where ",
         " it failed")
}
```

Sent the e-mail to notify to the user that the module finished running.

```
    if(!CheckDebug()){

        msg = "Imputation Completed Successfully"
        message(msg)

        ## Initiate email
        from = "sws@fao.org"
        to = swsContext.userEmail
        subject = "Crop-production imputation plugin has correctly run"
        body = paste0("Livestock production module successfully ran.
                      You can browse results in the session: ", sessionKey@sessionId)


        sendmail(from = from, to = to, subject = subject, msg = body)

    }

msg = "Imputation Completed Successfully"
```