Imputation Processed Commodities

Author: Francesca Rosa

Description:

Inputs:

1. commodity tree
2. SUA data (restricted to the components I need: Prod, Trade, Stock, Seed)

Steps:

1. Build the imputation key and pull the data necessary

2. Compute Share "DownUp"

3. Compute Processing share

Data scope

- GeographicAreaM49: The user can decide if apply the procedure to the country queried in the session or to "all". "all" picks its Value from the production completeImputationKey (stored in the data.table: fbs_production_comm_codes)

- measuredItemCPC: the list of item involved in this process is stored in the SWS, in the table processed_item

- timePointYears: All years specified in tha range selected by the user start_year-end_year. It is requested also to specifi the first year for which we need to produce imputations The module authomatically protecte all the figures up to this year

---

Initialisation

```r
suppressMessages({
    library(data.table)
    library(faosws)
    library(faoswsFlag)
    library(faoswsUtil)
    library(faoswsImputation)
    library(faoswsProduction)
    library(faoswsProcessing)
    library(faoswsEnsure)
    library(magrittr)
    library(dplyr)
    library(sendmailR)
    library(faoswsStandardization)

})


R_SWS_SHARE_PATH <- Sys.getenv("R_SWS_SHARE_PATH")
```

```r
if(CheckDebug()){

 library(faoswsModules)
 SETTINGS = ReadSettings("modules/processedItems/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"]])

 batchNumber=122
 dir.create(paste0("C:\\Users\\Rosa\\Desktop\\ProcessedCommodities\\BatchExpandedItems\\Batch",
                  batchNumber),
           recursive=TRUE)

 dir=paste0("C:\\Users\\Rosa\\Desktop\\ProcessedCommodities\\BatchExpandedItems\\Batch",
           batchNumber)

}
sessionKey = swsContext.datasets[[1]]
oldData=FALSE


##Create a new directories in the share env to support the validation:


dir_to_save <- file.path(R_SWS_SHARE_PATH, "processedItem","validation")

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


dir_to_save_plot <- file.path(R_SWS_SHARE_PATH, "processedItem","validation", "plot")

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


dir_to_save_SUA <- file.path(R_SWS_SHARE_PATH, "processedItem","validation", "SUA")

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


dir_to_save_output <- file.path(R_SWS_SHARE_PATH, "processedItem","validation", "output")
```

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

This are those commodity that are pubblished on FAOSTAT

I build the key, always the same in the PRODUCTION sub modules:

```
FBScountries=ReadDatatable("fbs_countries")[,code]
```

Get default parameters

```
params = defaultProcessedItemParams()
```

Get the list of processed items to impute

```
processedCPC=ReadDatatable("processed_item")[,measured_item_cpc]
toBePubblished=ReadDatatable("processed_item")[faostat==TRUE,measured_item_cpc]
```

Get the commodity tree from the TREE DATASET: The country dimention depends on the session:

```
geoImputationSelection = swsContext.computationParams$imputation_country_selection
sessionCountry=getQueryKey("geographicAreaM49", sessionKey)
selectedCountry =
    switch(geoImputationSelection,
           "session" = sessionCountry,
           "all" = FBScountries)
```

The year dimention depends on the session:

```
startYear=swsContext.computationParams$startYear
imputationStartYear=swsContext.computationParams$startImputation
endYear=swsContext.computationParams$endYear

areaKeys=selectedCountry
```

Check on the consistency of startYear, andYear

```
if(startYear>=endYear){
    stop("You must select startYear lower than endYear")
}


timeKeys=as.character(c(startYear:endYear))

if(!(imputationStartYear>=startYear & imputationStartYear<=endYear)){
    stop("imputationStartYear must lie in the time window between startYear and EndYear")
}
```

I have to pull all the items in the parent and child columns:

```
itemKeysParent=GetCodeList(domain = "suafbs",
                            dataset = "ess_fbs_commodity_tree",
                            "measuredItemParentCPC")[,code]

itemKeysChild=GetCodeList(domain = "suafbs",
                           dataset = "ess_fbs_commodity_tree",
                           "measuredItemChildCPC")[,code]
```

To save time I pull just the only element I need: "extraction rate" whose code is 5423

```
elemKeys=c("5423")

allCountries=areaKeys
```

```
finalByCountry=list()
allLevels=list()

#logConsole1=file("modules/processedItems/logProcessed.txt",open = "w")
#sink(file = logConsole1, append = TRUE, type = c( "message"))
```

Loop by country

```
start.time <- Sys.time()
for(geo in   seq_along(allCountries)){

    currentGeo=allCountries[geo]
```

Subset the data (SUA)

```
keyTree = DatasetKey(domain = "suafbs",
                     dataset = "ess_fbs_commodity_tree2",
                     dimensions = list(

    geographicAreaM49 = Dimension(name = "geographicAreaM49",
                                  keys = currentGeo),

    measuredElementSuaFbs = Dimension(name = "measuredElementSuaFbs",
                                      keys = elemKeys),

    measuredItemParentCPC_tree = Dimension(name = "measuredItemParentCPC_tree",
                                           keys = itemKeysParent),

    measuredItemChildCPC_tree = Dimension(name = "measuredItemChildCPC_tree",
                                          keys = itemKeysChild),

    timePointYears = Dimension(name = "timePointYears",
                               keys = timeKeys)
))

tree = GetData(keyTree)
#tree = try(setTimeOut({GetData(keyTree);},
#                    timeout=180, onTimeout="error"), silent = TRUE)
```

```
#if(!inherits(tree, "try-error")) {


if(nrow(tree)<1)
{message(paste0("The commodity-tree does not exist for the country: "), currentGeo)}else{
```

I modify the structure of the tree to make it compliant with the rest of the routine:

```
setnames(tree, c("measuredItemParentCPC_tree",
                 "measuredItemChildCPC_tree"),
               c("measuredItemParentCPC",
                 "measuredItemChildCPC"))

setnames(tree, "Value", "extractionRate")
tree[,flagObservationStatus:=NULL ]
tree[,flagMethod:=NULL ]
tree[,measuredElementSuaFbs:=NULL ]
```

I delete all the connection where no extraction rate is available:

```
tree=tree[!is.na(extractionRate)]
tree=tree[extractionRate!=0]
```

I associate the level to each parent-child combination, this operation is done by country-year combinations because the commodity tree may change across countries and over time.

The uniqueLevel represents all the country-year comb.

```
uniqueLevels = tree[, .N, by = c("geographicAreaM49", "timePointYears")]
uniqueLevels[, N := NULL]
levels=list()
treeLevels=list()
for (i in seq_len(nrow(uniqueLevels))) {
    filter=uniqueLevels[i, ]
    treeCurrent=tree[filter, , on = c("geographicAreaM49", "timePointYears")]
    levels=findProcessingLevel(treeCurrent,"measuredItemParentCPC","measuredItemChildCPC")
    setnames(levels, "temp","measuredItemParentCPC")
    treeLevels[[i]]= merge(treeCurrent, levels, by=c("measuredItemParentCPC"), all.x=TRUE)
}
tree=rbindlist(treeLevels)
```

Select all the commodity involved (what I need is just the dependence parent-child, all the possibilities, no country-year specific)

```
treeRestricted=tree[,.(measuredItemParentCPC,measuredItemChildCPC,processingLevel)]
treeRestricted=treeRestricted[with(treeRestricted, order(measuredItemChildCPC))]
treeRestricted=treeRestricted[!duplicated(treeRestricted)]
```

Get all the primary starting from the processed item (stored in the data.table: processed_item)

```
primaryInvolved=getPrimary(processedCPC, treeRestricted, params)
```

Get all the children of primary commodities involved (all levels)

```
primaryInvolvedDescendents=getChildren( commodityTree = treeRestricted,
                                        parentColname ="measuredItemParentCPC",
                                        childColname = "measuredItemChildCPC",
                                        topNodes =primaryInvolved )
```

For some countris (not included in the FBS country list) there are no processed commodities to be imputed, and the next steps cannot be run.

```
if(length(primaryInvolvedDescendents)==0)
    {message("No primary commodity involved in this country")}else{
```

Find the so-called secondLoop commodities: select all the unique combination of country-year-commodity-level:

```
multipleLevels=unique(tree[,.( geographicAreaM49,
                               timePointYears,
                               measuredItemChildCPC,
                               processingLevel)])
```

count all the repeated country-commodity combinations.

```
multipleLevels[,n:=.N, by=c("measuredItemChildCPC", "geographicAreaM49", "timePointYears")]
```

the n>1 it means that that commodity appears at least for one country in more tham one level. If this situation occurs just for some countries (and not for all), it is not a problem, because at the end of the process we decide to keep only ONE production-contribution and in particular the contribution of the highest level in the tree- hierachy (if it is just one level, we are properly considering ALL the production components) see row 560 of this script.

```
secondLoop=unique(multipleLevels[n!=1,measuredItemChildCPC])
```

Get SUA data from QA: all components are stored in the SWS Note Restricted means that I am pulling only the components I need in this submodule: PRODUCTION, STOCK, SEED, TRADE

```
dataSuaRestricted=getSUADataRestricted()
```

```
data = elementCodesToNames(data = dataSuaRestricted, itemCol = "measuredItemFbsSua",
                           elementCol = "measuredElementSuaFbs")
```

Add all the missing PRODUCTION row: if the production of a derived product does not exist it even if it is created by this routine cannot be stored in the SUA table and consequently all the commodities that belongs to its descendents are not estimates or are estimated using only the TRADE and neglecting an important part of the supply components.

```r
prod=data[measuredElementSuaFbs=="production", .( geographicAreaM49,
                                                   timePointYears,
                                                   measuredItemFbsSua)]
```

All time point years

```r
timePointYears=timeKeys
```

All countries:

```r
geographicAreaM49=currentGeo
```

I have to build table containing the complete data key (I have called it: all)

```r
all1=merge(timePointYears, primaryInvolvedDescendents)
setnames(all1, c("x","y"),c("timePointYears","measuredItemFbsSua"))

all2=merge(geographicAreaM49, primaryInvolvedDescendents)
setnames(all2, c("x","y"),c("geographicAreaM49","measuredItemFbsSua"))
all1=data.table(all1)
all2=data.table(all2)

all=merge(all1, all2, by="measuredItemFbsSua", allow.cartesian = TRUE)
```

The object "all" contais a complete key (all countries, all years, all items) for the production element To avoid duplicates I select just those row that are not already included into the SUA table

Note that all the columns of the all datatable have class=factor

```r
all[,measuredItemFbsSua:=as.character(measuredItemFbsSua)]
all[,timePointYears:=as.character(timePointYears)]
all[,geographicAreaM49:=as.character(geographicAreaM49)]


noProd=setdiff(all,prod)
```

I add the value and flags columns:

```r
if(nrow(noProd)>0){
noProd=data.table(noProd,
                  measuredElementSuaFbs="production",
                  Value=NA,flagObservationStatus="M",
                  flagMethod="u" )
```

I add this empty rows into the SUA table

```r
data=rbind(data,noProd)

}
```

```
dataProcessed=dataSuaRestricted[measuredElementSuaFbs=="5510"]
setnames(dataProcessed,c("measuredElementSuaFbs",
                         "measuredItemFbsSua"),
                       c("measuredElement",
                         "measuredItemCPC"))

dataProcessed[,timePointYears:=as.numeric(timePointYears)]
dataProcessed=dataProcessed[,benchmark_Value:=Value]
```

Artificially protect production data between the time window external to the range I have to produce imputations:

```
dataProcessed=expandYear(dataProcessed,newYear=as.numeric(endYear))
dataProcessed=removeInvalidFlag(dataProcessed, "Value",
                                "flagObservationStatus",
                                "flagMethod", normalised = TRUE)

dataProcessed=removeNonProtectedFlag(dataProcessed, "Value",
                                     "flagObservationStatus",
                                     "flagMethod", normalised = TRUE,
                                     keepDataUntil =imputationStartYear)

setnames(dataProcessed, "measuredItemCPC", "measuredItemChildCPC")
```

At the moment we have SUA data for the time range 2000-2015 (I pulled data from QA) data:SUA (if I pulled data from SWS (the name of the ITEM column is different: measuredItemFbsSua)) I keep only the item theat I need: processed Items and all those items in them trees

```
data=data[measuredItemFbsSua %in% primaryInvolvedDescendents]
data=data[!is.na(measuredElementSuaFbs),]
```

Change the name of the ITEM columns: parents

```
setnames(data,"measuredItemFbsSua","measuredItemParentCPC")
```

The loop works on processing levels:

```
levels=unique(tree[, processingLevel])

data[,timePointYears:=as.numeric(timePointYears)]
tree[,timePointYears:=as.character(timePointYears)]
tree[,timePointYears:=as.numeric(timePointYears)]
```

Subset PRODUCTION data

```
for(lev in (seq_along(levels)-1))  {
    ##'  Loop by level
    treeCurrentLevel=tree[processingLevel==lev]
    message("Processing country ", currentGeo, " - level ", lev )
```

To compute the PRODUCTION I need to evaluate how much (which percentage) of the parent commodity availability is allocated in the productive process associate to a specific child item.

Calculate share down up. Please note that currentData contains the SUA table.

```
dataMergeTree=calculateShareDownUp(data=data,tree=treeCurrentLevel,
                                   params=params, printNegativeAvailability=FALSE)
```

Here I merge the SUA table (already merged with tree), with the PRODUCTION DATA This merge produces many NA because dataProcessed contains all the derived item (and not just those at the level of the loop)

```
        inputOutputdata= merge(dataMergeTree,dataProcessed,
                               by=c("geographicAreaM49",
                                    "measuredItemChildCPC",
                                    "timePointYears"),
                               all.y=TRUE)


        inputOutputdata[shareDownUp=="NaN",shareDownUp:=0]

        inputOutputdata=calculateProcessingShare(inputOutputdata, param=params)

        inputOutputdata[, newImputation:=availability*processingShare*extractionRate]

        finalByCountry[[lev+1]]=inputOutputdata
```

Update production in data in order to add the just computed production at each loop we compute production for the following level, this prodution should be used in the following loop to compute the availabilities

```
        updateData=inputOutputdata[,.(geographicAreaM49,
                                      timePointYears,
                                      measuredItemChildCPC,
                                      newImputation)]

        updateData[, newImputation:=sum(newImputation,na.rm = TRUE),
                   by=c("geographicAreaM49", "timePointYears", "measuredItemChildCPC")]
        updateData=unique(updateData)
```

I change the column names bacause the commodities that now are children will be parent in the next loop setnames(updateData,“measuredItemChildCPC”,“measuredItemParentCPC”)

```
I merge the new results into the SUA table
        data=merge(data,updateData, by=c("geographicAreaM49",
                                         "timePointYears",
                                         "measuredItemParentCPC"),
                   all.x=TRUE)
```

Olnly non-protected production figures have to be overwritten:

```
            data[,flagComb:=paste(flagObservationStatus,flagMethod,sep=";")]

            flagValidTable=copy(flagValidTable)
            flagValidTable=flagValidTable[Protected==TRUE,]
            protected=flagValidTable[,protectedComb:=paste(flagObservationStatus,
                                                           flagMethod,
                                                           sep=";")]
            protected=protected[,protectedComb]
            data[geographicAreaM49==currentGeo &
                     !(flagComb %in% protected)&
                     measuredElementSuaFbs=="production" &
                     !is.na(newImputation) &
                     !(measuredItemParentCPC %in% secondLoop) &
                     timePointYears>=imputationStartYear,
                 ":="(c("Value","flagObservationStatus","flagMethod"),
                      list(newImputation,"I","e"))]


        data[,newImputation:=NULL]
        data[,flagComb:=NULL]
}
```

```
    allLevelsByCountry=rbindlist(finalByCountry)

    secondLoopChildren=getChildren( commodityTree = treeRestricted,
                                    parentColname ="measuredItemParentCPC",
                                    childColname = "measuredItemChildCPC",
                                    topNodes =secondLoop )
```

Note that the so called "secondLoop" items are computed for each country separately:

```
secondLoop=secondLoopChildren
```

This passage is to sum up the productions of a derived commodities coming from more than one parent (those commodities are flagged as second-loop). To identify the highest level of the hierarchy I have to choose the min value in the processingLevel column.

```
allLevelsByCountry[, minProcessingLevel:=NA_real_]
```

The min of the processing-level is not only item-specific, but also country-time specific, since the the commodity-tree may change over the time and the countries.

```
allLevelsByCountry[measuredItemChildCPC %in% secondLoop, minProcessingLevel:=min(processingLevel),
                   by=c("geographicAreaM49","timePointYears", "measuredItemChildCPC")]

allLevelsByCountry[measuredItemChildCPC %in% secondLoop & extractionRate!=0,
                   minProcessingLevel:=min(processingLevel),
                       by=c("geographicAreaM49",
                            "timePointYears",
                            "measuredItemChildCPC")]
```

I remove (just for those commodities classified as secondLoop) the production-contributions coming from parents lower in the hierachy (it means with an higher processing-level)

```
    allLevelsByCountry=allLevelsByCountry[(is.na(minProcessingLevel) | processingLevel==minProcessingLe



    allLevels[[geo]]=allLevelsByCountry
```

Create an intermediate set of data to be saved in a shared folder

```
    forValidationFile[[geo]]=    allLevels[[geo]][,.(geographicAreaM49,
                                                     measuredItemChildCPC,
                                                     timePointYears,
                                                     measuredItemParentCPC,
                                                     extractionRate,
                                                     processingLevel,
                                                     processingShare,
                                                     availability,
                                                     shareDownUp,
                                                     newImputation,
                                                     minProcessingLevel,
                                                     benchmark_Value)]

    setnames(forValidationFile[[geo]], c("measuredItemChildCPC",
                                       "measuredItemParentCPC"),
                                     c("measuredItemChildCPC_tree",
                                       "measuredItemParentCPC_tree"))

    forValidationFile[[geo]]= nameData("suafbs",
                                     "ess_fbs_commodity_tree2",
                                     forValidationFile[[geo]])



    dataLabel=copy(data)
    dataLabel=dataLabel[geographicAreaM49==currentGeo]
    setnames(dataLabel,  "measuredItemParentCPC", "measuredItemChildCPC_tree")
    dataLabel=nameData("suafbs", "ess_fbs_commodity_tree2", dataLabel)
    dataLabel[,timePointYears_description:=NULL]
    setnames(dataLabel, c("measuredElementSuaFbs", "Value"),
             c("measuredElementSuaFbs_child", "Value_child"))


forValidationFile[[geo]]=merge(dataLabel,forValidationFile[[geo]],
                               by=c("geographicAreaM49",
                                    "measuredItemChildCPC_tree",
                                    "measuredItemChildCPC_tree_description",
                                    "geographicAreaM49_description",
                                    "timePointYears") ,
                               allow.cartesian =  TRUE)




forValidationFile[[geo]]=forValidationFile[[geo]][,.(geographicAreaM49,geographicAreaM49_description,
                                                     timePointYears,
                                                     measuredItemChildCPC_tree,
                                                     measuredItemChildCPC_tree_description ,
                                                     measuredItemParentCPC_tree,
                                                     measuredItemParentCPC_tree_description,
                                                     measuredElementSuaFbs_child ,
                                                     Value_child,
                                                     flagObservationStatus,
                                                     flagMethod, availability ,
                                                     extractionRate,
                                                     processingLevel,
                                                     benchmark_Value,
                                                     newImputation,
                                                     shareDownUp,
                                                     processingShare,
                                                     minProcessingLevel)]
```

Save all the intermediate output country by country:

```
if(!CheckDebug()){


res_recovery = try(write.csv(allLevels[[geo]],
                             file=file.path(dir_to_save_recovery,paste0(currentGeo,".csv")),
                             row.names = FALSE),
                   silent = TRUE)

res_validationFile = try(write.csv(forValidationFile[[geo]],
                                   file=file.path(dir_to_save_output,paste0(currentGeo,".csv")),
                                   row.names = FALSE), silent = TRUE)



}
}
}




#}else{message(paste0("TimeOut issues to GetData for country: ",currentGeo))}



}

end.time <- Sys.time()

output=rbindlist(allLevels)
```

Those commodities that appear in more than one level of the commodity-tree hierachy are considered children of the highest item in the hierachy.

I have to expand the list of secondLoop commodities: I impose that all the descentents of a second-loop commodities are classified as “second-loop”:

The following lines create the dataset useful for validation purposes: I want to sum the newImputation into the totNewImputation, but I want to keep all the constributions to double check the final result

```
if(CheckDebug()){
    output[,  totNewImputation := sum(newImputation, na.rm = TRUE),
           by =c ("geographicAreaM49","measuredItemChildCPC","timePointYears")]

    ##'   outPutforValidation is the file created to validate the output from team B/C
    outPutforValidation=copy(output)


outPutforValidation=outPutforValidation[,.(geographicAreaM49,measuredItemChildCPC, timePointYears, meas
                                               processingLevel ,availability,shareDownUp,processingShar

    directory=paste0("C:/Users/Rosa/Desktop/ProcessedCommodities/BatchExpandedItems/Batch",
                     batchNumber,"/finalValidation")
    dir.create(directory)
    fileForValidation2(outPutforValidation,SUAdata=data ,  dir=directory)

    ##'  For validation purposes it is extremly important to produce validation files filtered for thos
    ##'  commodies plut flours (that are upposed to be pubblished)

}
```

Here I fisically sum by child the newImputation (that now are disaggregated by parent-contribution)

```
finalOutput=output[,  list(newImputation = sum(newImputation, na.rm = TRUE)),
                   by =c ("geographicAreaM49","measuredItemChildCPC","timePointYears")]
```

I subset the output datatable in order to make comparison between benchmark_Value (basically data already stored in the SWS) and newImputations (I keep just the Value and benchmark_Value columns)

orig=output[,.(geographicAreaM49,measuredItemChildCPC,timePointYears,Value,benchmark_Value,flagObservationStatus,fla
This subset of output is full of duplicated rows because each child can have more than one parents, and new imputations have been preduced for each of its parent. Each new imputations represent the contribution of one single parent item to the production of the derived item.

```
orig=orig[!duplicated(orig)]
```

FinalOutput contains all the newImputation (olready aggregated with respect to all the parent contribuions)

```
imputed=merge(finalOutput,orig, by=c("geographicAreaM49",
                                     "measuredItemChildCPC",
                                     "timePointYears"),
              allow.cartesian = TRUE)
```

Only M,u figures are overwritten by the new imputations.

```
imputed[flagObservationStatus=="M" & flagMethod=="u" & !is.na(newImputation),
        ":="(c("Value", "flagObservationStatus", "flagMethod"), list(newImputation,"I","e"))]
```

The column PROTECTED is created just to PLOT the output and distinguish between new imputations and protected figures.

```
imputed[,flagComb:=paste(flagObservationStatus,flagMethod,sep=";")]
imputed[, PROTECTED:=FALSE]
imputed[flagComb %in% protected, PROTECTED:=TRUE]
imputed[PROTECTED==TRUE,newImputation:=benchmark_Value]
toPlot=imputed
##'   This table is saved just to produce comparisond between batches:
```

```
if(CheckDebug()){
    write.csv(toPlot, paste0("C:\\Users\\Rosa\\Desktop\\ProcessedCommodities\\BatchExpandedItems\\Batch
                             batchNumber,"\\toPlot",batchNumber,".csv"), row.names=FALSE)
}

##Plots goes directly to the shared folder

if(!CheckDebug()){

res_plot = try(plotResult(toPlot, toPubblish=toBePubblished, pathToSave= dir_to_save_plot))

}
```

Save back

Remove the auxiliary columns created in the process

```
imputed[,newImputation:=NULL]
imputed[,benchmark_Value:=NULL]
imputed[,flagComb:=NULL]
imputed[,PROTECTED:=NULL]

imputed[,measuredElement:="5510"]
setnames(imputed, "measuredItemChildCPC", "measuredItemCPC")

imputed= removeInvalidDates(data = imputed, context = sessionKey)
imputed= postProcessing(data =  imputed)

imputed=imputed[flagObservationStatus=="I" & flagMethod=="e"]
imputed=imputed[,.(measuredElement,geographicAreaM49, measuredItemCPC,
                   timePointYears,Value,flagObservationStatus,flagMethod)]
```

Ensure that just data after the imputationStartYear are saved back and I am not overwriting protected figures

```
imputed=imputed[timePointYears>=imputationStartYear]
ensureProtectedData(imputed, returnData = FALSE)


SaveData(domain = sessionKey@domain,
         dataset = sessionKey@dataset,
         data =  imputed)
```

Sent an email to notify to the user that the module has successfully ran:

```
from = "sws@fao.org"
to = swsContext.userEmail
subject = "Derived-Products imputation plugin has correctly run"
body = paste0("The plug-in has saved the Production imputation in your session.",
              "You can browse charts and intermediate csv files in the shared folder: ",
                  dir_to_save)
```

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


print("The plugin ran successfully!")
```