Seed Use Impputation Process

Author: Francesca Rosa

Description:

Inputs:

1. Seed use data (domain:agriculture - dataset: aproduction - element:5525)
2. Area sown data (domain:agriculture - dataset: aproduction - element: 5025)
3. Area Harvested data (domain:agriculture - dataset: aproduction - element:5312)
4. Climate data (domain: World Bank, dataset: climate, element:)
5. Utilization tables (Data-table in the SUA/FBS domain)

Main steps:

1. Get the area harvested/sown data using the R API (GetData)
2. Then get the seed rate data after loading them in to the data-base (GetData)
3. Perform data quality checks
4. Get climate data
5. Interpolate missing climate data
6. Performe the Hierarchical Linear model
7. Save the data back

Inizialization

```r
suppressMessages({
  library(bit64)
  library(curl)
  library(faosws)
  library(faoswsUtil)
  library(lme4)
  library(data.table)
  library(magrittr)
  library(reshape2)
  library(igraph)
  library(plyr)
  library(dplyr)
  ##library(RJDBC)
  library(ggplot2)
  library(faoswsFlag)
  library(faoswsProcessing)
  library(splines)
  library(faoswsImputation)
  library(faoswsEnsure)

})

library(faoswsSeed)
```

Setting up variables:

```r
areaVar = "geographicAreaM49"
yearVar = "timePointYears"
itemVar = "measuredItemCPC"
elementVar = "measuredElement"
areaSownElementCode = "5025"
areaHarvestedElementCode = "5312"
seedElementCode = "5525"
```

```
valuePrefix = "Value_measuredElement_"
flagObsPrefix = "flagObservationStatus_measuredElement_"
flagMethodPrefix = "flagMethod_measuredElement_"


updateModel = TRUE
```

Set up for the test environment and parameters:

```
library(faosws)

if(CheckDebug()){

  library(faoswsModules)
  SETTINGS <- ReadSettings("sws.yml")
  SetClientFiles(SETTINGS[["certdir"]])

  GetTestEnvironment(baseUrl = SETTINGS[["server"]],
                     token = SETTINGS[["token"]])

}


sessionKey = swsContext.datasets[[1]]
completeImputationKey=getCompleteImputationKey()
lastYear=as.numeric(max(completeImputationKey@dimensions$timePointYears@keys))
```

GetData: a specific funcion has been created to get only protected SEED data. The code referring to "seed use" is "5525":

```
  seed = getOfficialSeedData()
  seed=as.data.table(seed)
```

Seed imput data validation

Ensure that the seed official values belong to the tha range (0, inf):

```
  ensureValueRange(data = seed,
                   ensureColumn = "Value_measuredElement_5525",
                   min = 0,
                   max = Inf,
                   includeEndPoint = TRUE,
                   returnData = FALSE,
                   getInvalidData = FALSE)
```

Ensure the validity of all the Flag combination (ObsFlag, MethodFlag):

```
  seed=ensureFlagValidity(data = seed,
                    flagObservationVar = "flagObservationStatus_measuredElement_5525",
                      flagMethodVar = "flagMethod_measuredElement_5525",
                      returnData = FALSE,
                      getInvalidData = FALSE)
```

EnsureCorrectMissingValue eventually is not necessary because I pull only protected data.

```
  ensureCorrectMissingValue(data = seed,
                            valueVar = "Value_measuredElement_5525",
                            flagObservationStatusVar = "flagObservationStatus_measuredElement_5525",
                            missingObservationFlag = "M",
                            returnData = FALSE,
                            getInvalidData = FALSE)
```

Remove those figures that in the past have been simply copied from the previous year (the so-called "carry-forward" observations):

```
 seed = removeCarryForward(data = seed,
                             variable = "Value_measuredElement_5525")
```

Build the CPC hyerarchy, this step is particularly important for the Linear mixed model where the CPC levels identify the model hyerchy:

```
seed = buildCPCHierarchy(data = seed, cpcItemVar = itemVar, levels = 3)
```

Get AREA data

It is necessary to pull from the SWS both "area sown" and "area harvested". While for "area sown" only protected figures have to be taken into account, for "area harvested" all the available data is used to derive area Sown (or eventually used as proxy for area sown).

Indeed, when the seed module is performed, imputed figured for "area harvested" have been alredy produced and validated through the "crop-production" imputation sub-module.

```
area=getAllAreaData()
##area=normalise(area)
##area=expandYear(area)
##area[is.na(Value), ":="(c("flagObservationStatus", "flagMethod"),list("I", "e"))]
##
##area= denormalise(area, denormaliseKey = "measuredElement")
```

```
ensureValueRange(data = area,
                 ensureColumn = "Value_measuredElement_5312",
                 min = 0,
                 max = Inf,
                 includeEndPoint = TRUE,
                 returnData = FALSE,
                 getInvalidData = FALSE)
```

```
ensureValueRange(data = area,
                 ensureColumn = "Value_measuredElement_5025",
                 min = 0,
                 max = Inf,
                 includeEndPoint = TRUE,
                 returnData = FALSE,
                 getInvalidData = FALSE)
```

```
areaPreProcessed= preProcessing(data= area,
                                normalised = FALSE)



areaConflict =areaRemoveZeroConflict(areaPreProcessed,
                                     value1= "Value_measuredElement_5025",
                                     value2= "Value_measuredElement_5312",
                                     observationFlag1= "flagObservationStatus_measuredElement_5025",
                                     methodFlag1= "flagMethod_measuredElement_5025",
                                     missingObservationFlag = "M",
                                     missingMethodFlag = "u"
 )
```

We need and authocorrection function that correct the invalid flag combinations (or flag-combinations coming from the old system):

```
areaConflictNormalised=normalise(areaConflict)
##invalid=   ensureFlagValidity(areaConflictNormalised,
##                              normalised =FALSE,
##                              getInvalidData = TRUE)
##

## we realise that there are still some invalid flag combinations

## empty empty ---> M u
## Flag (E, p) --> (E, f)
## Flag (E, e) --> (I, e)

autoFlagCorrection = function(data,
                            value= "Value",
                            flagObservationStatusVar = "flagObservationStatus",
                            flagMethodVar = "flagMethod"){
  dataCopy = copy(data)

  ## Correction (): (, ) --> (M, u)


  correctionFilter =
    dataCopy[[flagObservationStatusVar]] == "" &
    dataCopy[[flagMethodVar]] == ""

  dim2=dim(dataCopy[correctionFilter,])
  message("Number of (E, t) replaced with (E, -)", dim2[1])


  dataCopy[correctionFilter,
           `:=`(c(value,flagObservationStatusVar, flagMethodVar),
                .(NA_real_,"M", "u"))]

  ## Correction (2): (E, t) --> (E, -)


  correctionFilter =
    dataCopy[[flagObservationStatusVar]] == "E" &
    dataCopy[[flagMethodVar]] == "t"

  dim2=dim(dataCopy[correctionFilter,])
  message("Number of (E, t) replaced with (E, -)", dim2[1])



  dataCopy[correctionFilter,
           `:=`(c(flagObservationStatusVar, flagMethodVar),
                 .("E", "-"))]

  ## Correction (3): (E, e) --> (I, e)
  correctionFilter =
    dataCopy[[flagObservationStatusVar]] == "E" &
    dataCopy[[flagMethodVar]] == "e"


  dim3=dim(dataCopy[correctionFilter,])
  message("Number of (E, e) replaced with (I, e)", dim3[1])

  dataCopy[correctionFilter,
           `:=`(c(flagObservationStatusVar, flagMethodVar),
                .("I", "e"))]

  ## Correction (4): (E, p) --> (E, f)
  correctionFilter =
    dataCopy[[flagObservationStatusVar]] == "E" &
    dataCopy[[flagMethodVar]] == "p"


  dim4=dim(dataCopy[correctionFilter,])
  message("Number of (E, p) replaced with (E, f)", dim4[1])


  dataCopy[correctionFilter,
           `:=`(c(flagObservationStatusVar, flagMethodVar),
                .("E", "f"))]

  dataCopy

  ##   data.table(dim1[1],dim2[1],dim3[1],dim4[1])


}
```

```
areaConflictNormalised=autoFlagCorrection(areaConflictNormalised)

areaConflictNormalised=ensureFlagValidity(areaConflictNormalised,
                   normalised =TRUE,
                   getInvalidData = FALSE,
                   returnData = FALSE)


areaCleaned=denormalise(areaConflictNormalised, denormaliseKey = "measuredElement")
```

** Area sown imputation **

```
imputedArea=imputeAreaSown(data = areaCleaned,
                            byKey= c("geographicAreaM49","measuredItemCPC"))
```

AreaSown must be greater than or at least equal to area harvested. Check that in the AreaSown column all the values are not lower than AreaHarvested

```
ensureNoConflictingAreaSownHarvested (data= imputedArea,
                                      valueColumn1= "Value_measuredElement_5025",
                                      valueColumn2= "Value_measuredElement_5312",
                                      returnData = FALSE,
                                      normalised = FALSE,
                                      getInvalidData = FALSE)
```

** Pull Climate data ** From the World bank domain in the SWS, climate data is pulled. In particular, the Climate-element necessary to run the seed module is TEMPERATURE:

```
  climate = getWorldBankClimateData()
```

Merge all the necessary elements, pulled from the SWS, in one unique data.table:

```
finalModelData = mergeAllSeedData(seedData = seed, imputedArea, climate)
```

Extrapolate climateData: it means to fill the emply cells with the average country temperature

```
  finalModelData[, Value_wbIndicator_SWS.FAO.TEMP_mean:=mean(Value_wbIndicator_SWS.FAO.TEMP, na.rm=TRUE


  finalModelData[is.na(Value_wbIndicator_SWS.FAO.TEMP) ,
                 Value_wbIndicator_SWS.FAO.TEMP :=  Value_wbIndicator_SWS.FAO.TEMP_mean]

  finalModelData[,Value_wbIndicator_SWS.FAO.TEMP_mean:=NULL]
```

We have to remove cases where we do not have temperature, as we cannot create a model when independent variables are missing. The predictions would be NA anyways, and so we wouldn't be saving anything to the database if we left them in.

```
  finalModelData = finalModelData[Value_measuredElement_5525 > 1 &
                                    Value_measuredElement_5025 >1, ]






  finalModelData = finalModelData[!is.na(Value_wbIndicator_SWS.FAO.TEMP), ]
```

** The hierachical linear model **

```
  set.seed(260716)
  seedLmeModel =
    lmer(log(Value_measuredElement_5525) ~ Value_wbIndicator_SWS.FAO.TEMP +
           timePointYears +
           (log(Value_measuredElement_5025)|cpcLvl3/measuredItemCPC:geographicAreaM49),
         data = finalModelData)
```

Eventually create some plot to evaluate the results:

```
# par(mfrow=c(1,1))
# qqnorm(residuals(lossLmeModel))
# qqline(residuals(lossLmeModel))
```

Populate the seed dataset with predicted values:

```
seedAll = getSelectedSeedData()
seedAll= normalise(seedAll)

```

```
seedAll= expandYear(seedAll, newYears = lastYear)

seedAll[is.na(Value), ":="(c("flagObservationStatus",
                              "flagMethod"),
                            list("M", "u"))]
seedAll= denormalise(seedAll, denormaliseKey = "measuredElement")
```

```
seed1 = ensureFlagValidity(seedAll,normalised = FALSE,returnData = FALSE, getInvalidData = FALSE)
seed1= preProcessing(data= seed1, normalised = TRUE)
seed1=removeNonProtectedFlag(seed1,normalised = TRUE)

seed1 =denormalise(seed1, denormaliseKey = "measuredElement")


seed1 = buildCPCHierarchy(data = seed1, cpcItemVar = itemVar, levels = 3)
```

```
finalPredictData = merge(seed1,imputedArea, by=c("geographicAreaM49","measuredItemCPC","timePointYears"
finalPredictData= merge(finalPredictData,climate, by=c("geographicAreaM49","timePointYears"),all.x = TRU
```

We have to remove cases where we do not have temperature, as we cannot create a model when independent variables are missing. The predictions would be NA anyways, and so we wouldn't be saving anything to the database if we left them in.

```
finalPredictData[, Value_wbIndicator_SWS.FAO.TEMP_mean:=mean(Value_wbIndicator_SWS.FAO.TEMP, na.rm=TRUE
```

```
finalPredictData=finalPredictData[is.na(Value_wbIndicator_SWS.FAO.TEMP) ,
                                  Value_wbIndicator_SWS.FAO.TEMP :=  Value_wbIndicator_SWS.FAO.TEMP_mea

finalPredictData[,Value_wbIndicator_SWS.FAO.TEMP_mean:=NULL]

finalPredictData[,Value_wbIndicator_SWS.FAO.PREC:=NULL]


finalPredictData = finalPredictData[!is.na(Value_measuredElement_5025), ]

finalPredictData = finalPredictData[!is.na(Value_wbIndicator_SWS.FAO.TEMP), ]
```

Impute selected data

```
finalPredictData[, predicted := exp(predict(seedLmeModel,
                                            newdata = finalPredictData,
                                            allow.new.levels = TRUE))]



finalPredictData[,flagComb:=paste(flagObservationStatus_measuredElement_5525,
                                  flagMethod_measuredElement_5525,sep=";")]

protected=flagValidTable[Protected==TRUE,]
protected[,comb:=paste(flagObservationStatus,flagMethod,sep=";")]

tobeOverwritten=!finalPredictData[,flagComb] %in% protected[,comb]
```

This step should embed the idea that just non protected flag combinations shoul be overwritten while the previously version (the one now commented) was overwriting all the item characterised by observationFlag equal to c("E", "I", "T"). The only exception in the Seed context is the (M,-) flag combination: it has to be overwritten if it exists an areaSown different from (that's why we embedded into the conditions to select the items to be overwritten: !is.na(Value_measuredElement_5025))

```
finalPredictData[is.na(Value_measuredElement_5525)  & !is.na(predicted),
                 `:=` (c("Value_measuredElement_5525",
                         "flagObservationStatus_measuredElement_5525",
                         "flagMethod_measuredElement_5525"),
                       list(predicted, "I", "e"))]

finalPredictData[, predicted := NULL]
```

Output data validation

```
ensureValueRange(data = finalPredictData,
                 ensureColumn = "Value_measuredElement_5525",
                 min = 0,
                 max = Inf,
                 includeEndPoint = TRUE,
                 returnData = FALSE,
                 getInvalidData = FALSE)


ensureFlagValidity(data = finalPredictData,
                   flagObservationVar = "flagObservationStatus_measuredElement_5525",
                   flagMethodVar = "flagMethod_measuredElement_5525",
                   returnData = FALSE,
                   getInvalidData = FALSE)



finalPredictData_imputed=finalPredictData[, .(geographicAreaM49,
                                              timePointYears,
                                              measuredItemCPC,
                                              Value_measuredElement_5525,
                                              flagObservationStatus_measuredElement_5525,
                                              flagMethod_measuredElement_5525)]
```

Seed [t] should not be imputed for all the commodities which have an Area Harvested (or Area Sown):

```
finalPredictData_imputed=
  finalPredictData_imputed[,timePointYears:=as.character(timePointYears)]


finalPredictData_imputed =
  finalPredictData_imputed[flagObservationStatus_measuredElement_5525 == "I" &
                                                 flagMethod_measuredElement_5525 == "e",]



finalPredictData_imputed= normalise(finalPredictData_imputed)


finalPredictData_imputed=removeInvalidDates(data = finalPredictData_imputed,
                                            context = sessionKey)
```

Be sure that only the feasible country-commodity combinations will be saved in the session. This choice is based on the utilization_tables (based on the past allocation of the available primary commodity to the possible utilizations)

```
utilizationTable=ReadDatatable("utilization_table")


seedUtilization=unique(utilizationTable[element=="seed", .(area_code, item_code)])
setnames(seedUtilization, c("area_code", "item_code"), c("geographicAreaM49", "measuredItemCPC"))

seedInThePast=finalPredictData_imputed[seedUtilization,,on= c("geographicAreaM49", "measuredItemCPC")]
seedInThePast=seedInThePast[!is.na(timePointYears)]
```

Save data back in the session:

```
ensureProtectedData(finalPredictData_imputed,"production", "aproduction")

SaveData(domain = sessionKey@domain,
         dataset = sessionKey@dataset,
         data = seedInThePast)




"Module finished successfully"
```