

# faoswsImputation: A package for the imputation of missing time series data in the Statistical Working System

Joshua M. Browning  
Food and Agriculture Organization  
of the United Nations

---

## Abstract

This vignette provides a detailed description of the usage of functions in the **faoswsImputation** package.

*Keywords:* Imputation, Linear Mixed Model, Ensemble Learning.

---

## 1. Setup

Before we begin, we will need to load the required libraries

```
## Load libraries
library(faoswsImputation)
library(faoswsUtil)
library(data.table)
```

Additionally, we'll load the production library for an example dataset

```
library(faoswsProduction)
```

To illustrate the functionality of the package, we take the Okra data set as an example. The implementation requires the data to be loaded as a *data.table* object. This is also the default when data are queried from the API of the Statistical Working System (SWS).

```
str(okrapd)

## Classes 'data.table' and 'data.frame': 912 obs. of 14 variables:
## $ geographicAreaM49 : int 3 3 3 3 3 3 3 3 3 3 ...
## $ areaName : chr "Albania" "Albania" "Albania" "Albania" ...
## $ itemCode : int 430 430 430 430 430 430 430 430 430 430 ...
## $ itemName : chr "Okra" "Okra" "Okra" "Okra" ...
## $ timePointYears : int 1995 1996 1997 1998 1999 2000 2001 2002 ...
## $ Value_measuredElement_5312 : num 800 600 750 750 740 780 790 780 700 600 ...
## $ flagObservationStatus_measuredElement_5312: chr "T" "T" "T" "T" ...
## $ flagMethod_measuredElement_5312 : logi NA NA NA NA NA NA ...
## $ Value_measuredElement_5416 : num 7.25 8 8 8 7.97 ...
## $ flagObservationStatus_measuredElement_5416: chr "T" "T" "T" "T" ...
```

```
## $ flagMethod_measuredElement_5416      : logi  NA NA NA NA NA NA ...
## $ Value_measuredElement_5510          : num   5800 4800 6000 6000 5900 6200 6500 67
## $ flagObservationStatus_measuredElement_5510: chr   "T" "T" "T" "T" ...
## $ flagMethod_measuredElement_5510      : logi   NA NA NA NA NA NA ...
## - attr(*, ".internal.selfref")=<externalptr>
## - attr(*, "index")= atomic
## ..- attr(*, "flagObservationStatus_measuredElement_5510")= int    9 10 11 35 36 37 49 50
## ..- attr(*, "flagObservationStatus_measuredElement_5416")= int    9 10 11 35 36 37 49 50
## ..- attr(*, "flagObservationStatus_measuredElement_5312")= int    9 10 11 35 36 37 49 50
```

Note: the okrapd dataset is a good example of the data available in the SWS. However, as such, the column names aren't very clear. As a quick explanation, 5312 refers to area harvested, 5416 refers to yield, and 5510 refers to production. Each variable has three columns for its value and the two status flags.

In addition to the data, the implementation also requires a table to map the hierarchical relation of the observation flags. It provides a rule for “flag aggregation” (the process of assigning a new observation flag to an observation which is computed from other observations). An example of the table is given below. For more details on flags and how to create/interpret such tables, please see the vignette of the **faoswsFlag** package.

```
swsOldFlagTable = faoswsFlagTable
faoswsFlagTable$flagObservationStatus =
  as.character(faoswsFlagTable$flagObservationStatus)
swsOldFlagTable
```

```
##   flagObservationStatus flagObservationWeights
## 1
## 2           T           0.80
## 3           E           0.75
## 4           I           0.50
## 5           M           0.00
```

## 2. Functions

This section describes the step-by-step usage of functions which are used to perform imputation. The steps/functions illustrated here are for demonstration purposes only, as usually these functions will all be called by a one-step imputation function `imputeProductionDomain` (i.e. a “wrapper function”).

### 2.1. Data processing

The first step of the imputation is to remove any previous imputations. Even when using the same methodology and settings, prior imputations will change as more information is received over time. This step is highly recommended but optional and depends on the judgement of the analyst.

To remove the prior imputations, one will need to specify the column name of the value and corresponding observation and method flags. Further, the character value which represents an imputation and the character value for a flag representing missing values must be provided.

The function will convert the previously imputed values to NA and the flags from previous imputations will be set to the missing flag value.

```
okraProcessed = copy(okrapd)

## Removing prior imputation for production
table(okraProcessed$flagObservationStatus_measuredElement_5510)

##
##      E      M      T
## 487 239 131  55

removeImputation(data = okraProcessed,
                  value = "Value_measuredElement_5510",
                  observationFlag =
                    "flagObservationStatus_measuredElement_5510",
                  methodFlag = "flagMethod_measuredElement_5510",
                  imputedFlag = "E",
                  missingObservationFlag = "M",
                  missingMethodFlag = "u")
table(okraProcessed$flagObservationStatus_measuredElement_5510)

##
##      M      T
## 487 370  55

## Removing prior imputation for area harvested
table(okraProcessed$flagObservationStatus_measuredElement_5312)

##
##      E      M      T
## 430 289 131  62

removeImputation(data = okraProcessed,
                  value = "Value_measuredElement_5312",
                  observationFlag =
                    "flagObservationStatus_measuredElement_5312",
                  methodFlag = "flagMethod_measuredElement_5312",
                  imputedFlag = "E",
                  missingObservationFlag = "M",
                  missingMethodFlag = "u")
table(okraProcessed$flagObservationStatus_measuredElement_5312)

##
##      M      T
## 430 420  62

## Removing prior imputation for yield
table(okraProcessed$flagObservationStatus_measuredElement_5416)
```

```
##
##      E      M      T
## 410 313 131   58

removeImputation(data = okraProcessed,
                  value = "Value_measuredElement_5416",
                  observationFlag =
                    "flagObservationStatus_measuredElement_5416",
                  methodFlag = "flagMethod_measuredElement_5416",
                  imputedFlag = "E",
                  missingObservationFlag = "M",
                  missingMethodFlag = "u")
table(okraProcessed$flagObservationStatus_measuredElement_5416)

##
##      M      T
## 410 444   58
```

After removing prior imputations, the next step is to replace zero values with a missing flag to values of NA. This is an issue from previous data: some observations will be labeled as missing but given a value of zero instead of a value of NA.

```
okraProcessed[geographicAreaM49 == 12 & timePointYears >= 2005,
              .(Value_measuredElement_5312,
                flagObservationStatus_measuredElement_5312)]

##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
## 1:                               0                                           M
## 2:                               0                                           M
## 3:                               0                                           M
## 4:                               0                                           M
## 5:                               0                                           M
## 6:                               19                                          
## 7:                               19                                          
## 8:                               19                                          
## 9:                               19                                           T

removeOM(data = okraProcessed,
          value = "Value_measuredElement_5312",
          flag = "flagObservationStatus_measuredElement_5312",
          naFlag = "M")
okraProcessed[geographicAreaM49 == 12 & timePointYears >= 2005,
              .(Value_measuredElement_5312,
                flagObservationStatus_measuredElement_5312)]

##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
## 1:                               NA                                           M
## 2:                               NA                                           M
## 3:                               NA                                           M
## 4:                               NA                                           M
## 5:                               NA                                           M
```

```
## 6: 19
## 7: 19
## 8: 19
## 9: 19 T
```

Note how the zeroes have been changed into NA's above. Let's do the same for production and area harvested:

```
removeOM(data = okraProcessed,
          value = "Value_measuredElement_5416",
          flag = "flagObservationStatus_measuredElement_5416",
          naFlag = "M")

removeOM(data = okraProcessed,
          value = "Value_measuredElement_5510",
          flag = "flagObservationStatus_measuredElement_5510",
          naFlag = "M")
```

In order for the linear mixed model (one of the models in the ensemble that we will later fit) to fit successfully, at least one observation is required for each country. Thus, this function removes countries which contain no non-missing observations.

```
okraProcessed[geographicAreaM49 == 245,
               .(Value_measuredElement_5416,
                  flagObservationStatus_measuredElement_5416)]

##      Value_measuredElement_5416 flagObservationStatus_measuredElement_5416
## 1: NA M
## 2: NA M
## 3: NA M
## 4: NA M
## 5: NA M
## 6: NA M
## 7: NA M
## 8: NA M
## 9: NA M
## 10: NA M
## 11: NA M
## 12: NA M
## 13: NA M
## 14: NA M
## 15: NA M
## 16: NA M
## 17: NA M
## 18: NA M
## 19: NA M

removeNoInfo(data = okraProcessed,
              value = "Value_measuredElement_5416",
              observationFlag = "flagObservationStatus_measuredElement_5416",
              byKey = "geographicAreaM49")
```

```
okraProcessed[geographicAreaM49 == 245,
               .(Value_measuredElement_5416,
                 flagObservationStatus_measuredElement_5416)]

## Empty data.table (0 rows) of 2 cols: Value_measuredElement_5416,flagObservationStatus_measuredElement_5416
```

Note for advanced users: All other `remove*` functions from the `utils` package perform by modifying the `data.table` in place (and thus you do not need to assign a new `data.table` to the result of a function). `removeNoInfo` should work in the same way, but there is currently not a way to delete rows in a `data.table` without copying the `data.table`. Thus, the object cannot be modified in place. For this function to behave like the other functions, then, we assign the `data.table` object in an environment (by default, the calling environment of `removeNoInfo`). This should be changed once the **data.table** package adds this functionality.

Next, we must create a list that contains specific parameters on how the processing should be performed.

```
processingParams = defaultProcessingParameters()
processingParams

## $productionValue
## [1] "Value_measuredElement_5510"
##
## $productionObservationFlag
## [1] "flagObservationStatus_measuredElement_5510"
##
## $productionMethodFlag
## [1] "flagMethod_measuredElement_5510"
##
## $yieldValue
## [1] "Value_measuredElement_5416"
##
## $yieldObservationFlag
## [1] "flagObservationStatus_measuredElement_5416"
##
## $yieldMethodFlag
## [1] "flagMethod_measuredElement_5416"
##
## $areaHarvestedValue
## [1] "Value_measuredElement_5312"
##
## $areaHarvestedObservationFlag
## [1] "flagObservationStatus_measuredElement_5312"
##
## $areaHarvestedMethodFlag
## [1] "flagMethod_measuredElement_5312"
##
## $yearValue
## [1] "timePointYears"
##
## $byKey
## [1] "geographicAreaM49"
```

```
##
## $removePriorImputation
## [1] TRUE
##
## $removeConflictValues
## [1] TRUE
##
## $imputedFlag
## [1] "E"
##
## $naFlag
## [1] "M"
```

Now, we will pass `processingParams` through all of the individual processing functions. The function `processProductionDomain` is a wrapper that executes all the data processing above.

```
okraProcessed = copy(okrapd)
processProductionDomain(data = okraProcessed,
                        processingParameters = processingParams)
```

## 2.2. Imputation

Now we are ready to perform the imputation. First, we'll impute the yield. The function `imputeVariable` allows the user to perform imputation on the dataset, and it accepts a list of imputation parameters which control how the imputation is done.

To run the imputation, we need to construct a list with the default imputation parameters (similar to the list with the processing parameters) and adjust them as necessary for our specific use case. The documentation page for `defaultImputationParameters()` provides some detail on what each of the different elements of this list are. Also, let's delete some of the data (having too many countries can clog up some of the later plots).

```
okraProcessed = okraProcessed[geographicAreaM49 <= 60, ]
imputationParams = defaultImputationParameters(variable = "yield")
sapply(imputationParams, class)

##           yearValue           forKey           ensembleModels
##           "character"           "character"           "list"
## restrictWeights    maximumWeights    plotImputation
##           "logical"           "numeric"           "logical"
##           errorType           errorFunction           groupCount
##           "character"           "function"           "numeric"
##           missingFlag           imputationFlag           newMethodFlag
##           "character"           "character"           "character"
##           flagTable           variable    imputationValueColumn
##           "data.frame"           "character"           "character"
## imputationFlagColumn imputationMethodColumn    newImputationColumn
##           "character"           "character"           "character"
##           estimateNoData
##           "logical"
```

One very important part of this list is the `ensembleModels` element. This element specifies all of the models which should be used to form the final ensemble. By default, ten models are used. However, let's use a simpler example with just three models:

```
names(imputationParams$ensembleModels)

## [1] "defaultMean"      "defaultLm"        "defaultExp"
## [4] "defaultLogistic"  "defaultLoess"     "defaultSpline"
## [7] "defaultArima"     "defaultMars"      "defaultNaive"
## [10] "defaultMixedModel"

imputationParams$ensembleModels =
  imputationParams$ensembleModels[1:3]
names(imputationParams$ensembleModels)

## [1] "defaultMean" "defaultLm"   "defaultExp"
```

You can also manually create your own model for use. See the documentation page for `?ensembleModel` for more details, and below for an example:

```
newModel = ensembleModel(
  model = function(data){
    rep(10, length(data))
  },
  extrapolationRange = 5,
  level = "countryCommodity")
is(newModel)

## [1] "ensembleModel"

imputationParams$ensembleModels = c(imputationParams$ensembleModels,
                                     newModel = newModel)
names(imputationParams$ensembleModels)

## [1] "defaultMean" "defaultLm"   "defaultExp"  "newModel"
```

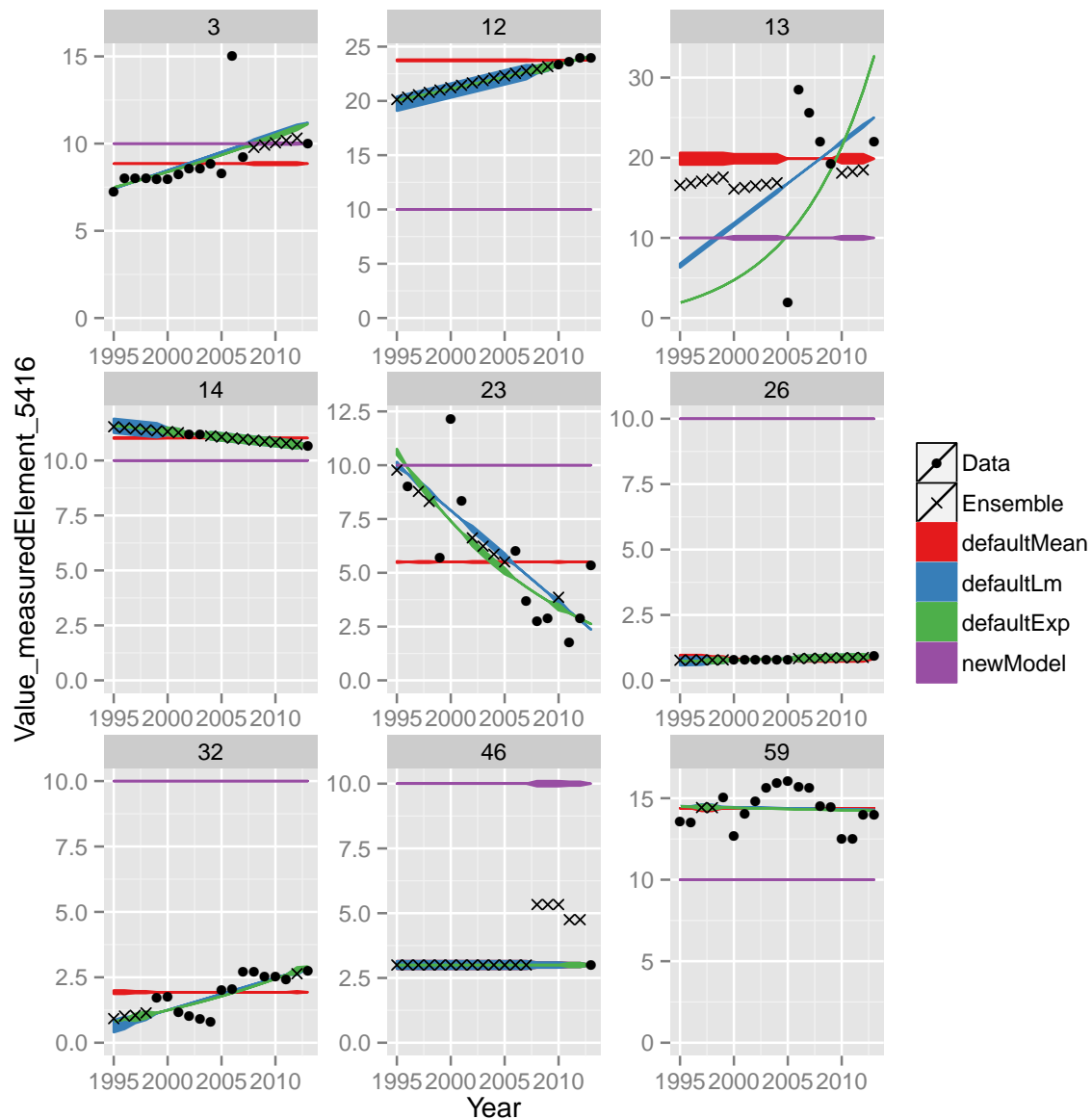
This new model returns a constant prediction of 10. It's not a good model, but it's a simple example of how to create a new model. The `extrapolationRange` specifies that the model can be used in an ensemble up to 5 observations outside the range of the data, but no more. The `level` argument specifies that the model should operate on data for all countries for a fixed commodity (as opposed to one model for each unique country-commodity pair).

```
imputationParams$newImputationColumn = "test"
imputeVariable(data = okraProcessed, imputationParameters = imputationParams)
colnames(okraProcessed)

## [1] "geographicAreaM49"
## [2] "areaName"
## [3] "itemCode"
## [4] "itemName"
```



```
## [5] "timePointYears"
## [6] "Value_measuredElement_5312"
## [7] "flagObservationStatus_measuredElement_5312"
## [8] "flagMethod_measuredElement_5312"
## [9] "Value_measuredElement_5416"
## [10] "flagObservationStatus_measuredElement_5416"
## [11] "flagMethod_measuredElement_5416"
## [12] "Value_measuredElement_5510"
## [13] "flagObservationStatus_measuredElement_5510"
## [14] "flagMethod_measuredElement_5510"
## [15] "Value_test"
## [16] "flagObservationStatus_test"
## [17] "flagMethod_test"
```



Before discussing the output, first note that the `newImputationColumn` parameter was updated in the `imputationParams` object. This parameter allows you to store the imputations from the model in three new columns (value, observation flag, and method flag). This allows

you to examine several different ensembles and compare their performances. The default value of `imputationParams$newImputationColumn` is just an empty string, and in this case the imputation will place the imputations into the processed `data.table`.

The graphs contain a lot of information. First, the dots represent observed values, and the crosses represent the imputations. The different colored lines show the different fits, and the thickness of the line is proportional to the weight it received in the ensemble. Of course, if the data point is an observation then no imputation is done, so all lines have the same thickness there. Also, the computed weights will be constant for all imputed values with one exception: models that are not allowed to extrapolate may have positive weights for some imputations and 0 for others. Moreover, if an observation is outside the extrapolation range of a model, then the weight of all other models will need to be rescaled so all values add to 1.

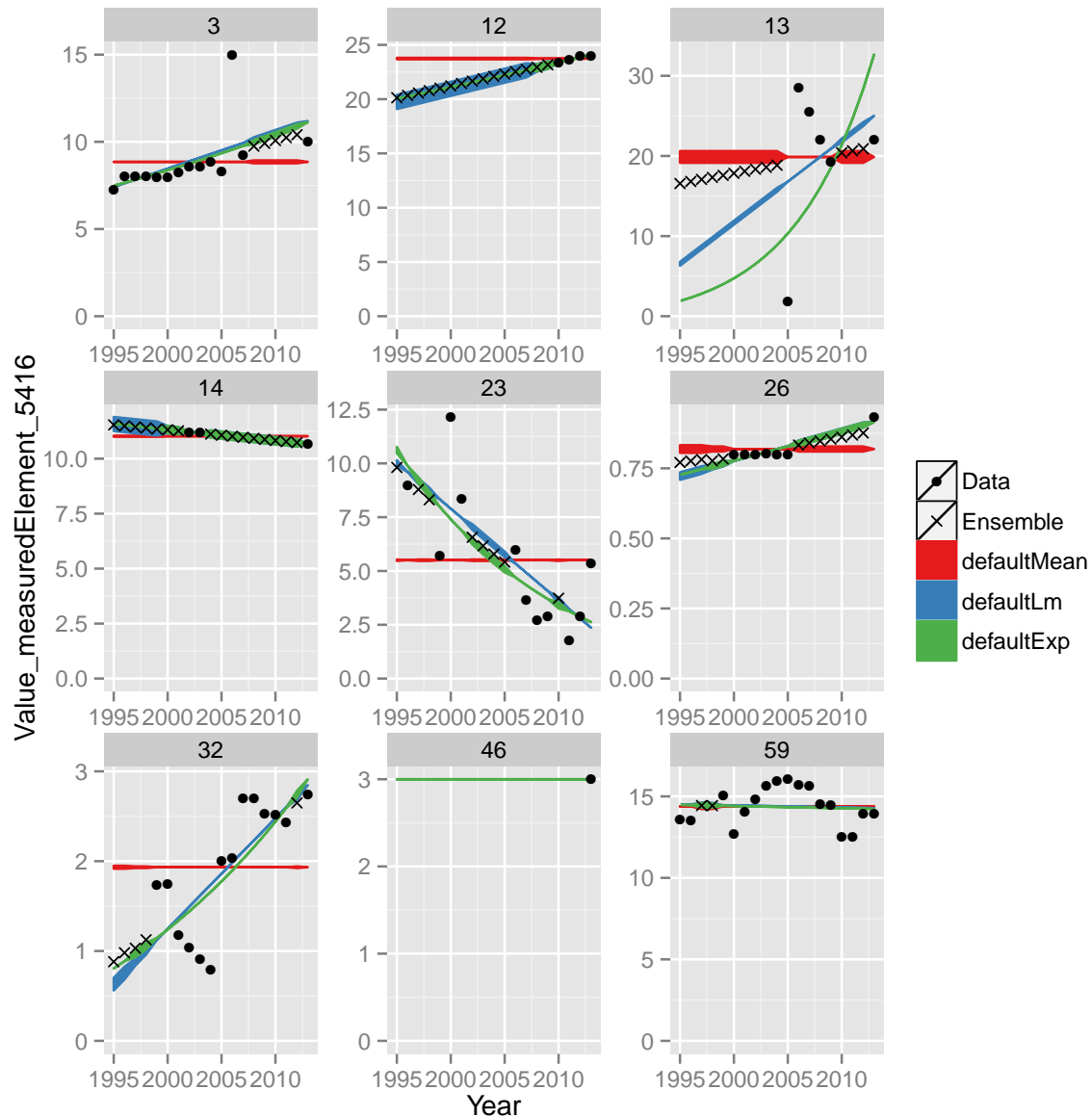
We see that the purple line (the model corresponding to our naive model which always estimates the value 10) rarely gets any weight. This makes sense, as it's not a very good model. However, in some particular cases (i.e. `geographicAreaM49 = 13`), no models do very well. The mean model thus gets most of the weight, but our naive model also gets a little weight. You can also see how it only has weight up to 5 observations outside of the range of the data; this is because we gave the model an extrapolation range of 5.

Now, suppose we wanted to remove the naive model which always predicts 10. We can do that and re-examine what our ensemble looks like:

```
imputationParams$ensembleModels = imputationParams$ensembleModels[-4]
names(imputationParams$ensembleModels)

## [1] "defaultMean" "defaultLm"   "defaultExp"

imputeVariable(data = okraProcessed, imputationParameters = imputationParams)
```



If we're happy with this model, we can assign these imputed values back to the original variable:

```
imputationParams$newImputationColumn = ""
imputeVariable(data = okraProcessed, imputationParameters = imputationParams)
okraProcessed[, c("Value_test", "flagObservationStatus_test",
                  "flagMethod_test")] := NULL]
```

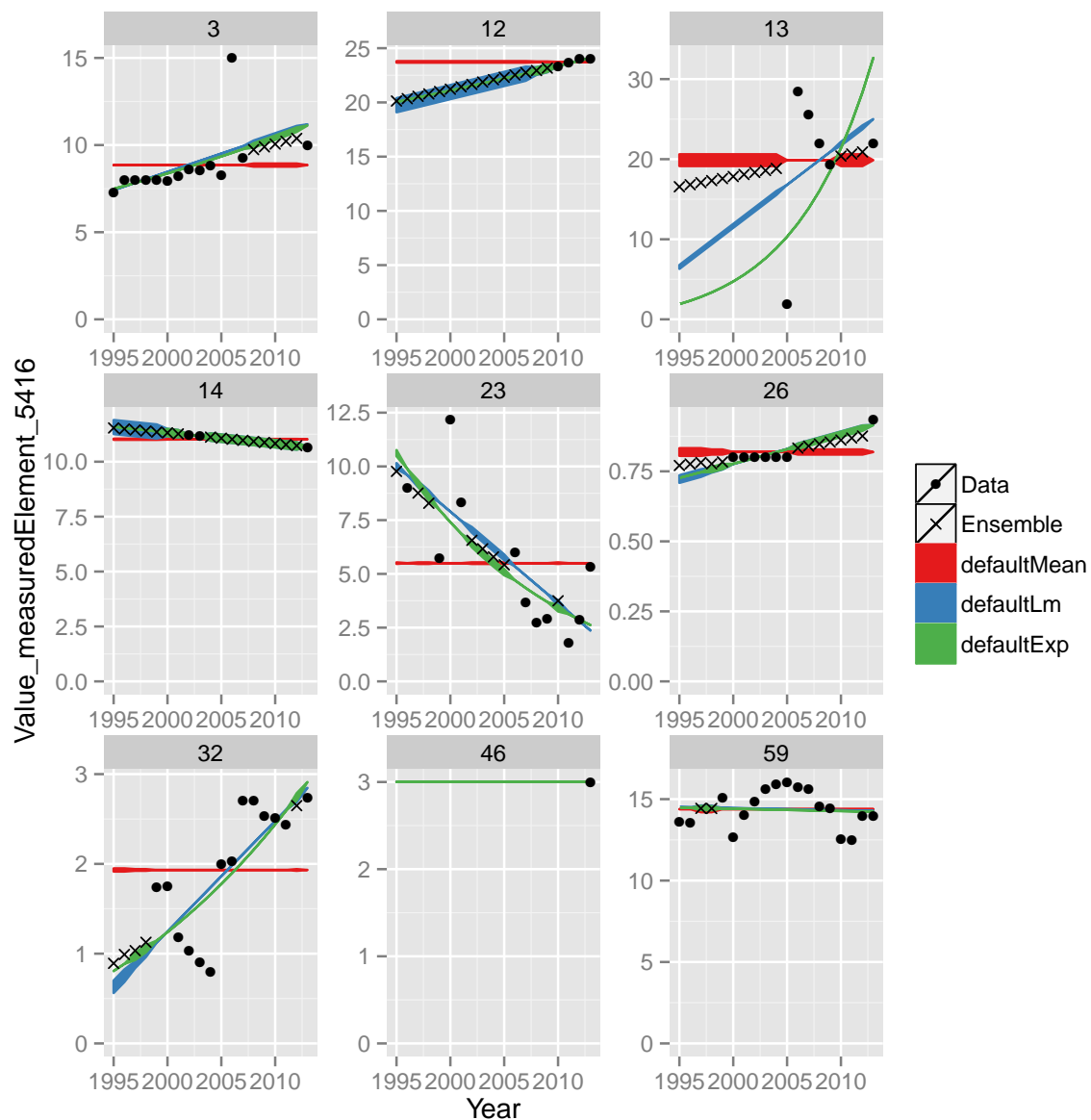
```
##      geographicAreaM49 areaName itemCode itemName timePointYears
##  1:                3  Albania    430    Okra      1995
##  2:                3  Albania    430    Okra      1996
##  3:                3  Albania    430    Okra      1997
##  4:                3  Albania    430    Okra      1998
##  5:                3  Albania    430    Okra      1999
##  ---
## 205:               59   Egypt    430    Okra      2009
## 206:               59   Egypt    430    Okra      2010
## 207:               59   Egypt    430    Okra      2011
```

```

## 208:          59   Egypt      430      Okra      2012
## 209:          59   Egypt      430      Okra      2013
##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
##    1:          800                                     T
##    2:          600                                     T
##    3:          750                                     T
##    4:          750                                     T
##    5:          740                                     T
## ---
## 205:          9325
## 206:          6885
## 207:          6725
## 208:          6957
## 209:          6957                                     T
##      flagMethod_measuredElement_5312 Value_measuredElement_5416
##    1:          NA          7.250000
##    2:          NA          8.000000
##    3:          NA          8.000000
##    4:          NA          8.000000
##    5:          NA          7.972973
## ---
## 205:          NA          14.441287
## 206:          NA          12.524619
## 207:          NA          12.496803
## 208:          NA          13.958315
## 209:          NA          13.958315
##      flagObservationStatus_measuredElement_5416 flagMethod_measuredElement_5416
##    1:          T                                     NA
##    2:          T                                     NA
##    3:          T                                     NA
##    4:          T                                     NA
##    5:          T                                     NA
## ---
## 205:          NA
## 206:          NA
## 207:          NA
## 208:          NA
## 209:          T                                     NA
##      Value_measuredElement_5510 flagObservationStatus_measuredElement_5510
##    1:          5800                                     T
##    2:          4800                                     T
##    3:          6000                                     T
##    4:          6000                                     T
##    5:          5900                                     T
## ---
## 205:          134665
## 206:          86232
## 207:          84041
## 208:          97108
## 209:          97108                                     T
##      flagMethod_measuredElement_5510

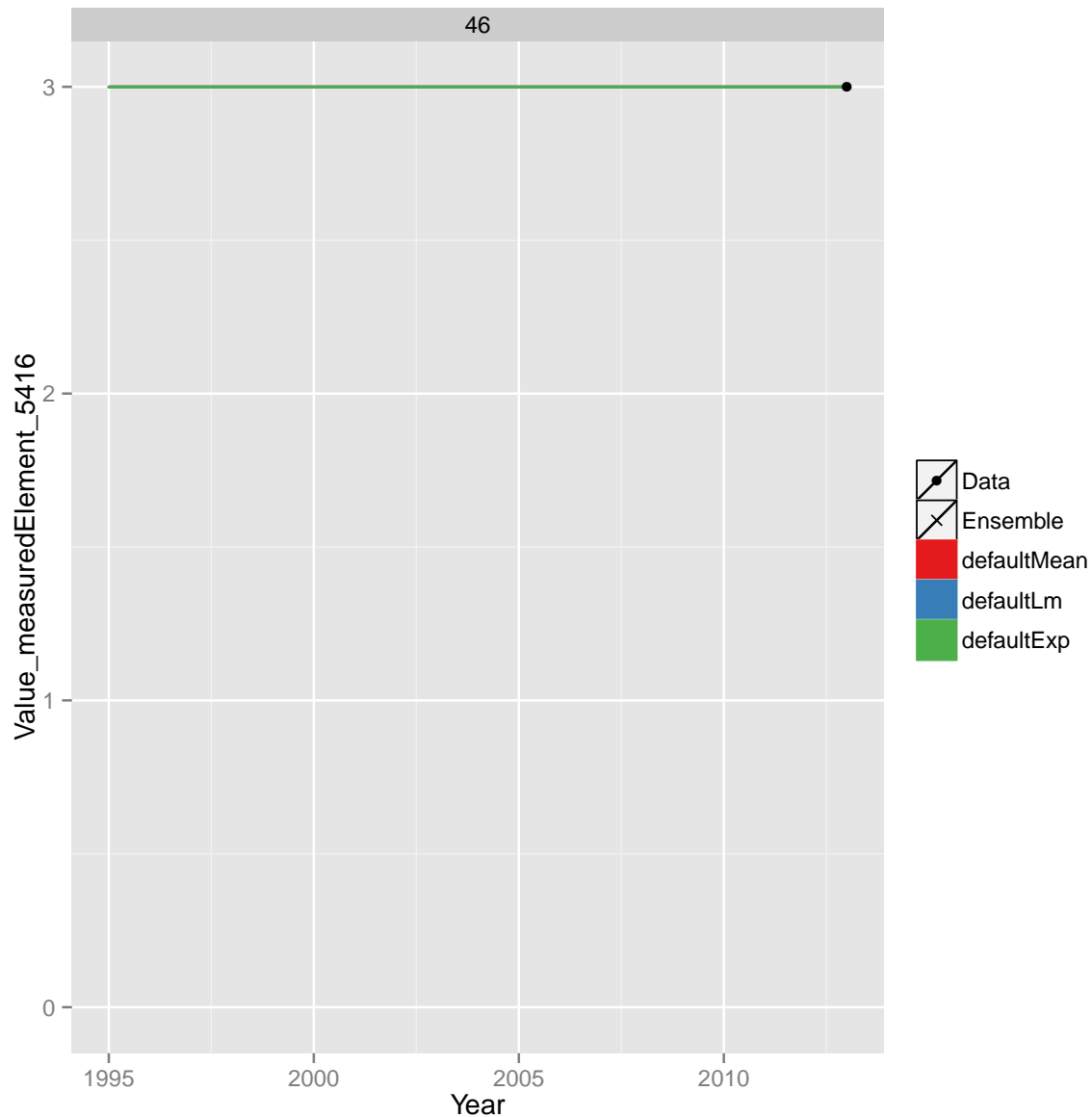
```

```
## 1: NA
## 2: NA
## 3: NA
## 4: NA
## 5: NA
## ---
## 205: NA
## 206: NA
## 207: NA
## 208: NA
## 209: NA
```



If we now try to impute again, we see that imputation fails because we have no missing observations. Well, to be more accurate, we have missing observations in one country with only one valid observation. This country was not possible to impute because no leave-one-out cross-validation error can be calculated with a single observation and thus no ensemble weights can be chosen.

```
imputeVariable(data = okraProcessed, imputationParameters = imputationParams)
```



After the imputation of yield, we proceed to impute the production. The function `imputeVariable` is used again, but we first need to impute by “balancing,” i.e. updating missing values of production when yield and area harvested both exist. This is because we have the relationship:

$$Y = P/A$$

where  $Y$  is yield,  $P$  is production, and  $A$  is the area harvested. If no value for area harvested is available, then the function proceeds to impute the remaining production values with ensemble learning. Let’s use some different models this time, just for the purpose of showing what’s available.

```
balanceProduction(data = okraProcessed,
                  imputationParameters = imputationParams,
                  processingParameters = processingParams)
imputationParams = defaultImputationParameters("production")
imputationParams$ensembleModels =
```

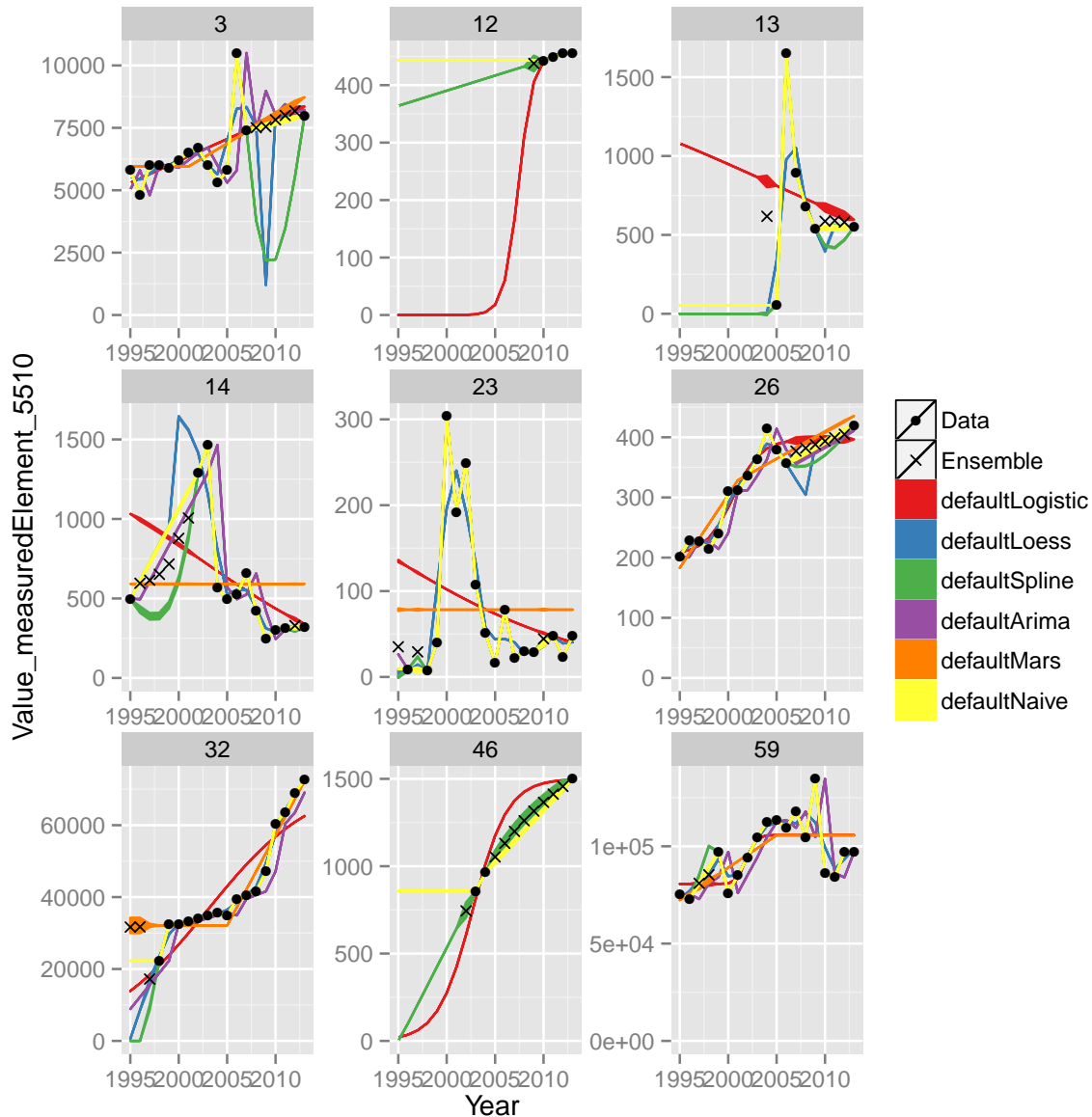
```

imputationParams$ensembleModels[4:9]
names(imputationParams$ensembleModels)

## [1] "defaultLogistic" "defaultLoess"      "defaultSpline"      "defaultArima"
## [5] "defaultMars"      "defaultNaive"

imputeVariable(data = okraProcessed,
               imputationParameters = imputationParams)

```



Note: imputations that are interpolations are always present, but some extrapolations are not imputed. The reason for this is that some models are not reasonable to extrapolate with (such as LOESS, Splines, etc.). For these models, an "extrapolation range" is defined, and this value dictates how far outside the range of the data a particular model is allowed to extrapolate. In some cases, no models are valid at a certain range and thus no imputation is performed. To avoid these kinds of issues, we recommend including a simple model that will rarely fail and that can be used to extrapolate (for example, defaultMean or defaultLm). Such models will ensure that most values are imputed. In our case, the extrapolation ranges are:

```
for(model in imputationParams$ensembleModels)
  print(model@extrapolationRange)

## [1] 1
## [1] 1
## [1] 1
## [1] Inf
## [1] Inf
## [1] 0
```

Thus, the Logistic, Arima, and Mars models are the only models that are allowed to extrapolate more than one observation away from the data. For most of the examples provided here, those three models all failed to fit to the data, and so imputations were not available.

Finally, we can balance the area harvested after both production and yield have been imputed.

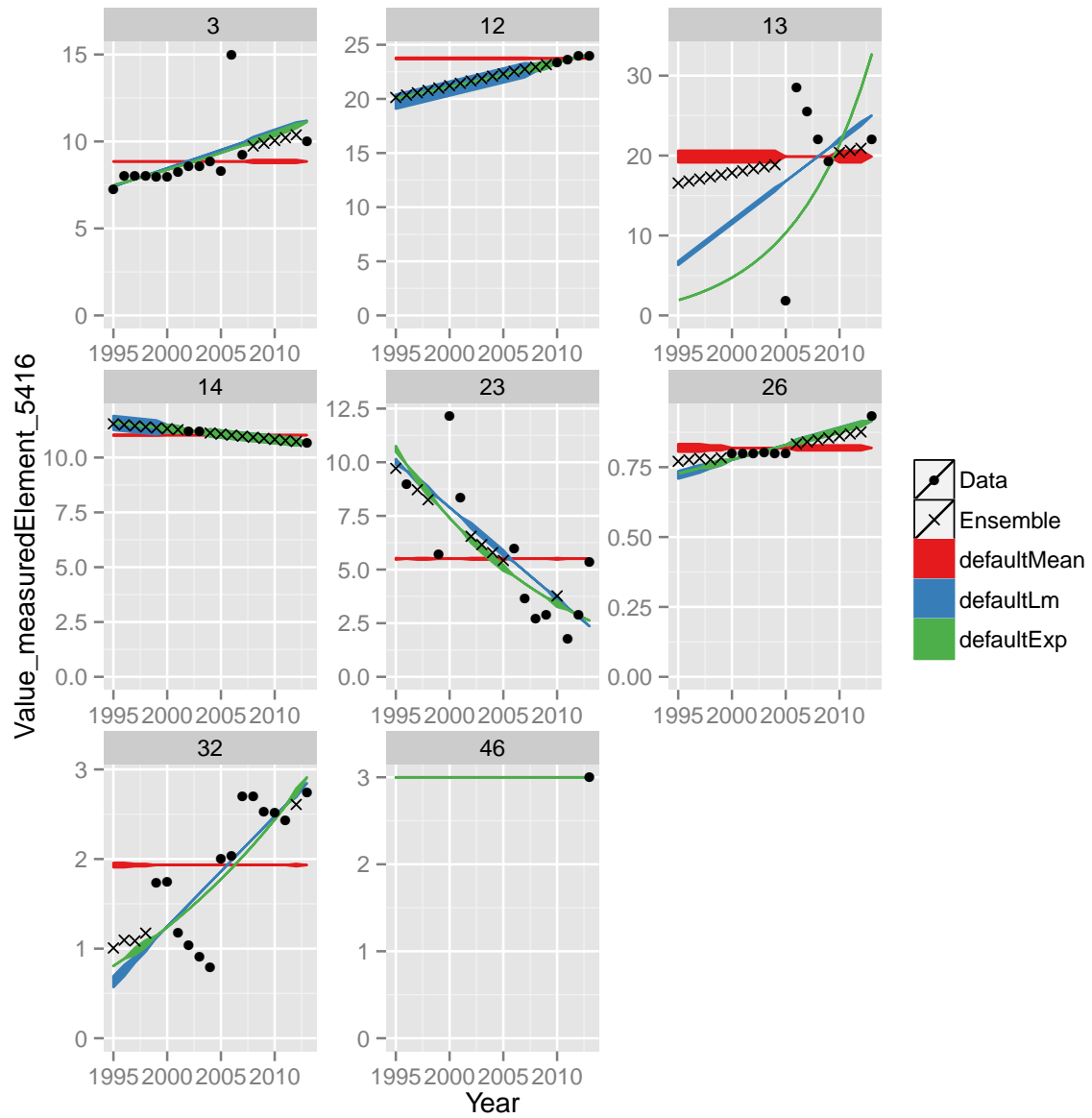
```
balanceAreaHarvested(data = okraProcessed,
                      imputationParameters = imputationParams,
                      processingParameters = processingParams)
```

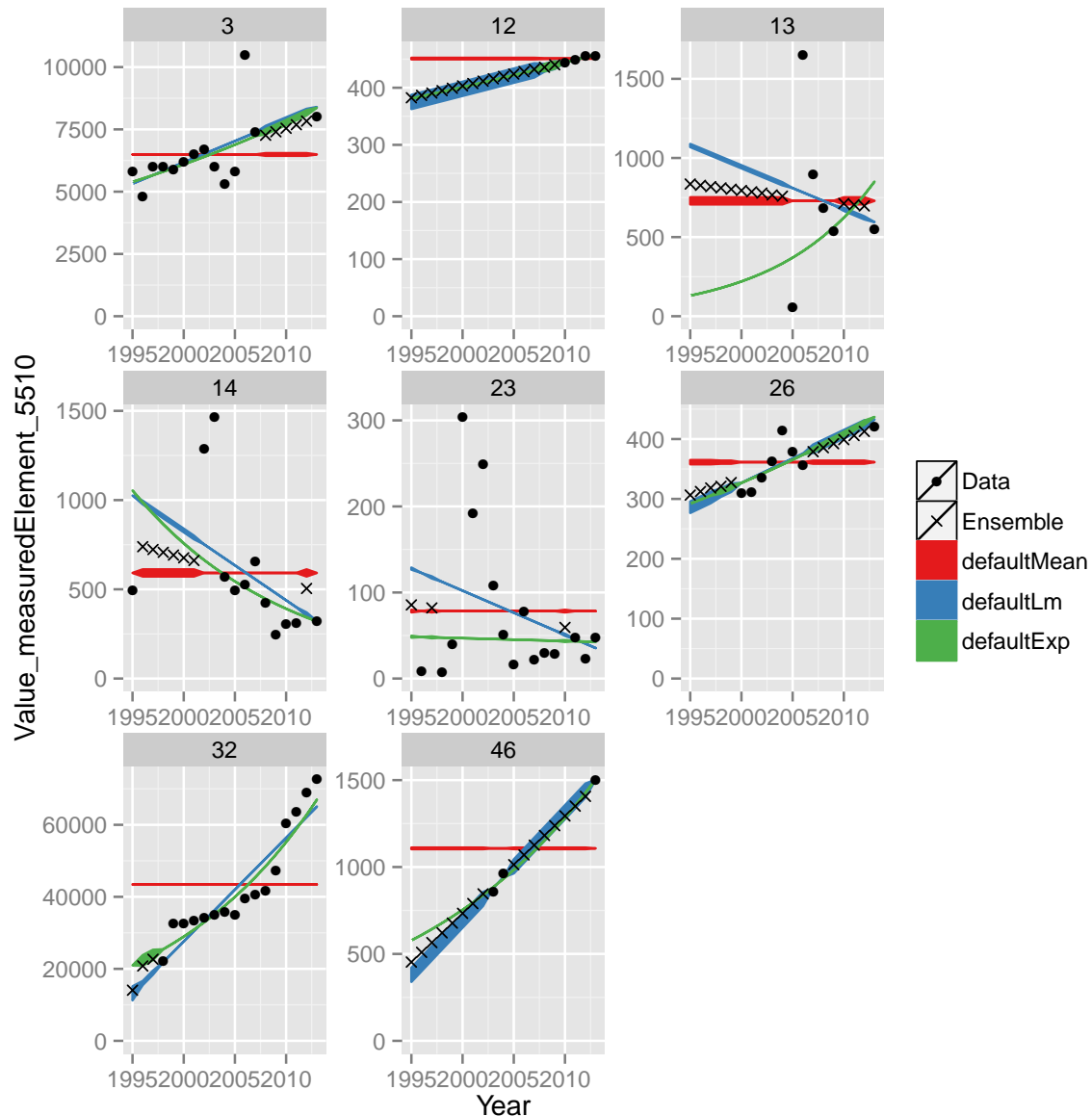
The full procedure outlined in this section can be performed by a single function `imputeProductionDomain`. You will need to specify three parameter lists: the processing parameters (1) and the imputation parameters for both yield and production (2).

```
yieldParams = defaultImputationParameters("yield")
yieldParams$ensembleModels = yieldParams$ensembleModels[1:3]
productionParams = defaultImputationParameters("production")
productionParams$ensembleModels = productionParams$ensembleModels[1:3]
okraProcessed = okrapd[geographicAreaM49 <= 55, ]
system.time(
{
  imputeProductionDomain(data = okraProcessed,
                        processingParameters = processingParams,
                        yieldImputationParameters = yieldParams,
                        productionImputationParameters =
                          productionParams)
})

## Initializing ...
## Imputing Yield ...
## Number of values imputed: 74
## Number of values still missing: 18
## Imputing Production ...
## Number of values imputed: 73
## Number of values still missing: 0
## Imputing Area Harvested ...
## Number of values imputed: 69
## Number of values still missing: 18
## user system elapsed
## 4.86 0.03 4.91
```







### 3. Ensemble model

Here we provide some details of how to implement user specific ensemble models.

First of all, the component models need to take a vector of values and return the fitted values. If the model failed, then a vector of NAs equal to the length of the input should be returned.

Shown below is the default linear regression model in the package. It is the analyst's job to ensure the component models return sensible values. For example, negative values are nonsensical for production, and in the current implementation negative values are replaced with zero.

```
defaultLm

## function (x)
## {
```

```
##      stopifnot(is.numeric(x))
##      stopifnot(length(x) > 1)
##      time = 1:length(x)
##      if (all(is.na(x)))
##          return(as.numeric(rep(NA_real_, length(x))))
##      lmFit = predict(lm(formula = x ~ time), newdata = data.frame(time = time))
##      lmFit[lmFit < 0] = 0
##      lmFit
##  }
## <environment: namespace:faoswsImputation>
```

Now, to create an `ensembleModel` object, two other pieces of information must be provided: the extrapolation range of the model (i.e. how many years it can extrapolate outside the support of the data) and the “level” of the model (see the class documentation):

```
mod = ensembleModel(model = defaultLm, extrapolationRange = Inf,
                    level = "countryCommodity")
is(mod)

## [1] "ensembleModel"
```

Now, `mod` is an object of type `ensembleModel`. We can construct a list of several of these models, but there are also some default models implemented. Calling `allDefaultModels()` returns a list of all of these models.

```
names(allDefaultModels())

## [1] "defaultMean"      "defaultLm"        "defaultExp"
## [4] "defaultLogistic"  "defaultLoess"     "defaultSpline"
## [7] "defaultArima"     "defaultMars"      "defaultNaive"
## [10] "defaultMixedModel"

sapply(allDefaultModels(), is)

##      defaultMean      defaultLm      defaultExp      defaultLogistic
## "ensembleModel" "ensembleModel" "ensembleModel" "ensembleModel"
##      defaultLoess      defaultSpline      defaultArima      defaultMars
## "ensembleModel" "ensembleModel" "ensembleModel" "ensembleModel"
##      defaultNaive      defaultMixedModel
## "ensembleModel" "ensembleModel"
```

Here we take the Okra production value of Bahrain as an illustration. After the component models have been designed and inserted into a list, we can compute the fits and weights then combine it to form the ensemble with the following functions.

First, we have to make sure we’ve correctly labeled any missing values as NA and not 0:

```
bahrainExample = okrapd[areaName == "Bahrain", ]
bahrainExample[1:4, .(areaName, timePointYears,
                      production = Value_measuredElement_5510,
                      productionFlag =
                        flagObservationStatus_measuredElement_5510)]
```

```
##      areaName timePointYears production productionFlag
## 1: Bahrain      1995           0           M
## 2: Bahrain      1996           0           M
## 3: Bahrain      1997           0           M
## 4: Bahrain      1998           0           M

removeOM(data = bahrainExample, value = "Value_measuredElement_5510",
         flag = "flagObservationStatus_measuredElement_5510")
bahrainExample[1:4, .(areaName, timePointYears,
                     production = Value_measuredElement_5510,
                     productionFlag =
                       flagObservationStatus_measuredElement_5510)]

##      areaName timePointYears production productionFlag
## 1: Bahrain      1995           NA           M
## 2: Bahrain      1996           NA           M
## 3: Bahrain      1997           NA           M
## 4: Bahrain      1998           NA           M
```

Next, we compute the model fits. We'll print the first three fits:

```
## Compute fit for all component models
imputationParameters = defaultImputationParameters("production")
modelFits = computeEnsembleFit(data = bahrainExample,
                              imputationParameters = imputationParameters)

modelFits[1:3]

## $defaultMean
## [1] 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333
## [9] 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333 659.3333
## [17] 659.3333 659.3333 659.3333
##
## $defaultLm
## [1] 1191.5667 1153.5500 1115.5333 1077.5167 1039.5000 1001.4833 963.4667
## [8] 925.4500 887.4333 849.4167 811.4000 773.3833 735.3667 697.3500
## [15] 659.3333 621.3167 583.3000 545.2833 507.2667
##
## $defaultExp
## [1] 181.4353 195.3650 210.3582 226.4962 243.8665 262.5630 282.6871 304.3477
## [9] 327.6622 352.7568 379.7676 408.8407 440.1336 473.8159 510.0700 549.0922
## [17] 591.0939 636.3026 684.9631

length(modelFits)

## [1] 10
```

To compute weights, we need to use cross-validation. Each observation is assigned a cross-validation group. To compute the error of a particular model, we estimate the observed values in group  $i$  with all values not in group  $i$ . This allows us to measure how well a model predicts the data, and can help prevent overfitting. The model weights are then computed. Note the NA's; these exist when observations are real and values are not being imputed.

```
## Calculate the weight for each component model
cvGroup = makeCvGroup(data = bahrainExample,
                      imputationParameters = imputationParameters)
cvGroup

## [1] NA NA NA NA NA NA NA NA NA NA NA 2 1 7 3 8 6 9 4 5

modelWeights = computeEnsembleWeight(data = bahrainExample,
                                     cvGroup = cvGroup,
                                     fits = modelFits,
                                     method = "inverse",
                                     imputationParameters =
                                         imputationParameters)
modelWeights[, .(defaultArima, defaultExp, defaultLm)]

##      defaultArima defaultExp defaultLm
## 1:              0 0.0000000 0.2529203
## 2:              0 0.0000000 0.2529203
## 3:              0 0.0000000 0.2529203
## 4:              0 0.0000000 0.2529203
## 5:              0 0.0000000 0.2529203
## 6:              0 0.0000000 0.2529203
## 7:              0 0.0000000 0.2529203
## 8:              0 0.0000000 0.2529203
## 9:              0 0.1393818 0.2176679
## 10:             0 0.1066930 0.1666189
## 11:             NA         NA         NA
## 12:             NA         NA         NA
## 13:             NA         NA         NA
## 14:             NA         NA         NA
## 15:             NA         NA         NA
## 16:             NA         NA         NA
## 17:             NA         NA         NA
## 18:             NA         NA         NA
## 19:             NA         NA         NA

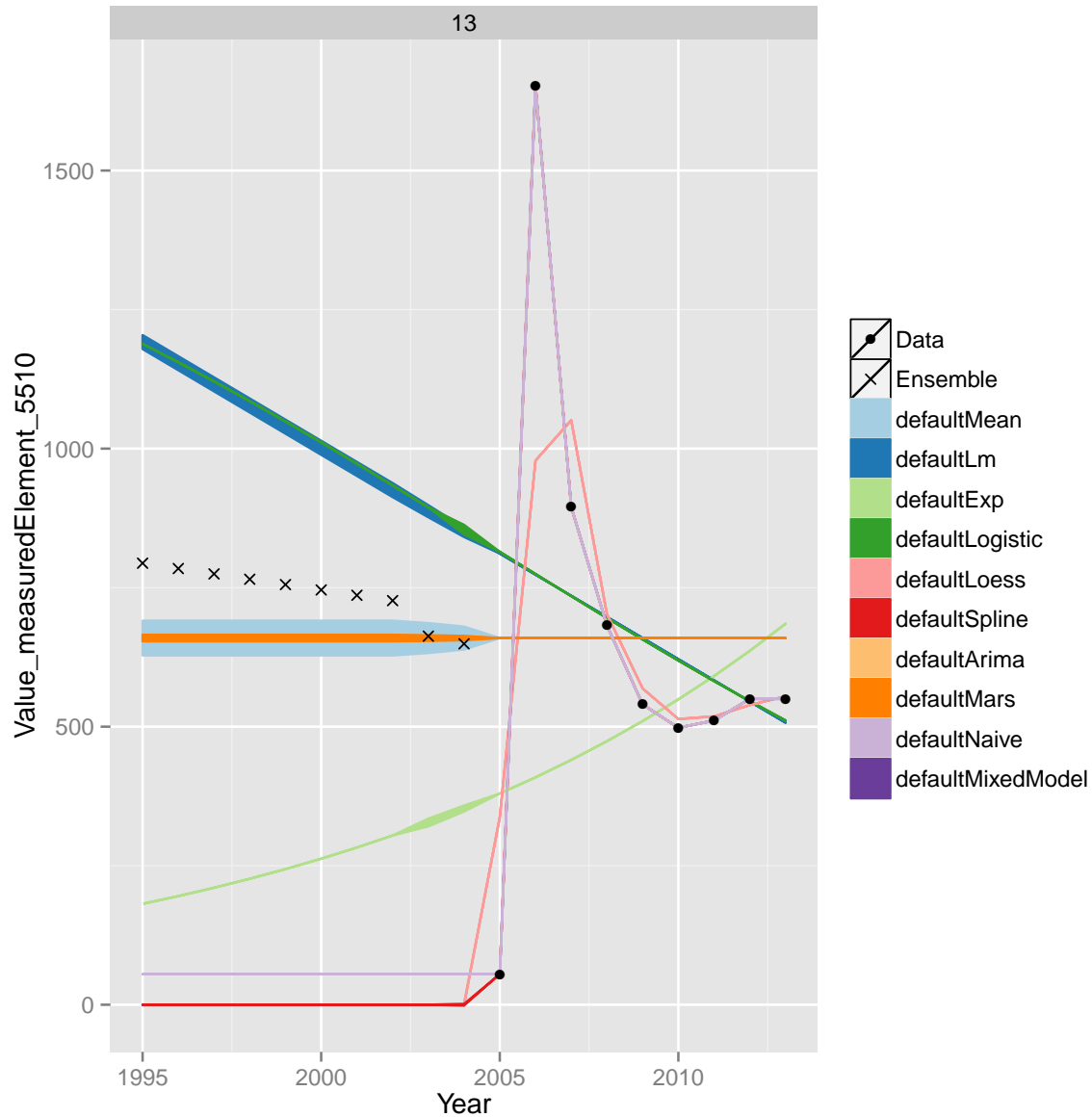
dim(modelWeights)

## [1] 19 10
```

Lastly, combine the fits with the estimated weights to produce the final ensemble, and then plot it!

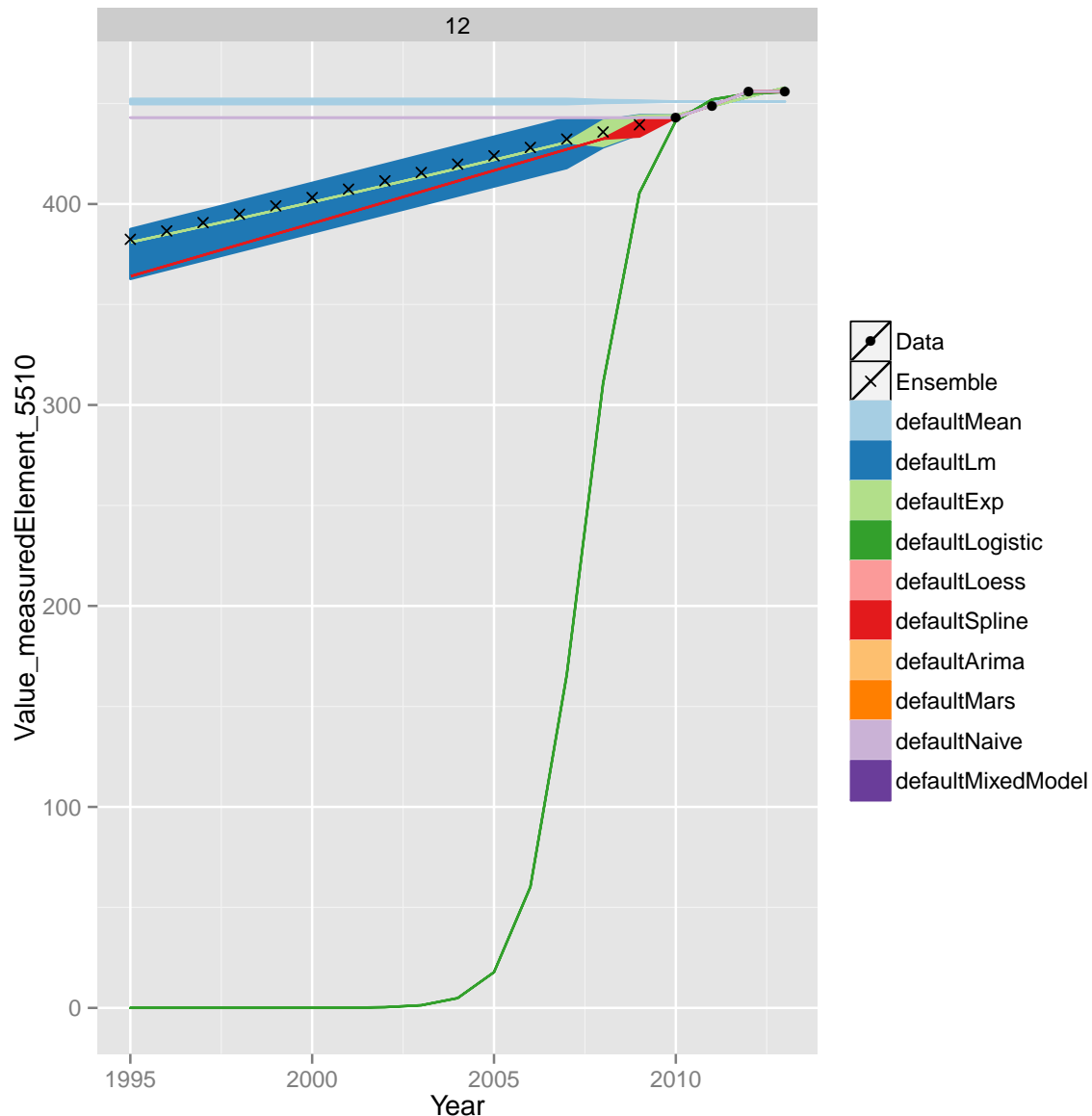
```
## Combine the models to obtain the ensemble
ensemble = bahrainExample[, Value_measuredElement_5510]
imputationFit = computeEnsemble(modelFits, modelWeights)
ensemble[is.na(ensemble)] = imputationFit[is.na(ensemble)]
plotEnsemble(data = bahrainExample, modelFits = modelFits,
             modelWeights = modelWeights, ensemble = ensemble,
             imputationParameters = imputationParameters)
```

## NULL



A one-step wrapper function is also available. There are also many other options you can specify when constructing an ensemble, such as the maximum weight that may be given to a model or a custom error function for choosing weights. See `defaultImputationParameters` for a description of all the options.

```
bahamasExample = okrapd[areaName == "Bahamas", ]
removeOM(data = bahamasExample, value = "Value_measuredElement_5510",
         flag = "flagObservationStatus_measuredElement_5510")
ensembleFit = ensembleImpute(data = bahamasExample,
                             imputationParameters = imputationParameters)
```



## 4. Models for Ensembling

This package implements many complex models that may not be familiar to the user, and so this section goes through the models and describes how the algorithm works as well as gives an example of the usage of that model. The order of this section is alphabetical, not by complexity. Let's first set up a dataset to use for this example, and we'll set

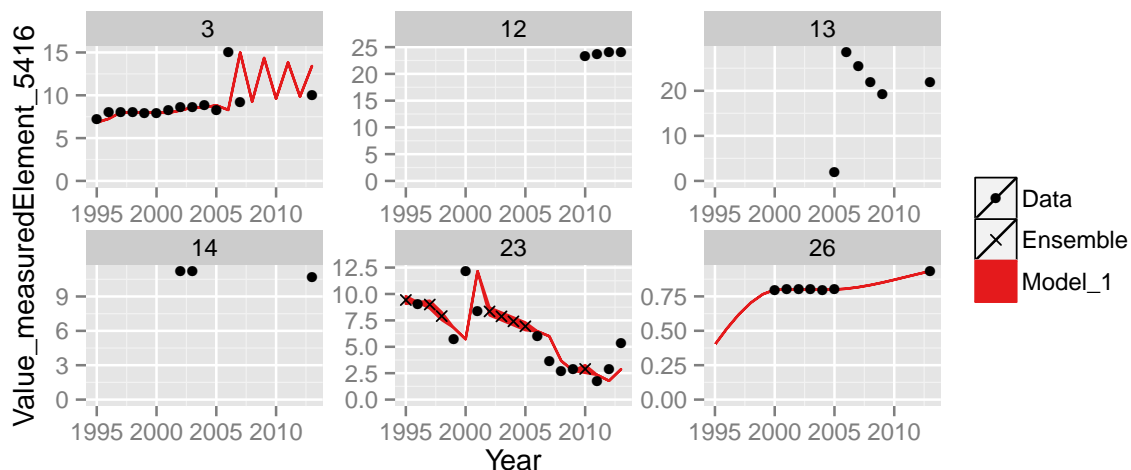
```
exampleData = okrapd[geographicAreaM49 < 30, ]
processProductionDomain(exampleData,
  processingParameters = defaultProcessingParameters()
imputationParameters = defaultImputationParameters("yield")
imputationParameters$newImputationColumn = "test"
invisible(exampleData[timePointYears %in% 2005:2007 &
  geographicAreaM49 == "91",
  c("Value_measuredElement_5416",
    "flagObservationStatus_measuredElement_5416") :=
```

```
list(NA, "M"])]
```

#### 4.1. defaultArima

The defaultArima model first fits an AutoRegressive, Integrated Moving Average (ARIMA) model to the time series provided, and it attempts to find the best model using the auto.arima function from the **forecast** package. If such a model is found, that model is used (along with KalmanSmooth) to generate new smoothed estimates of the time series.

```
model = ensembleModel(model = defaultArima, extrapolationRange = Inf,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```

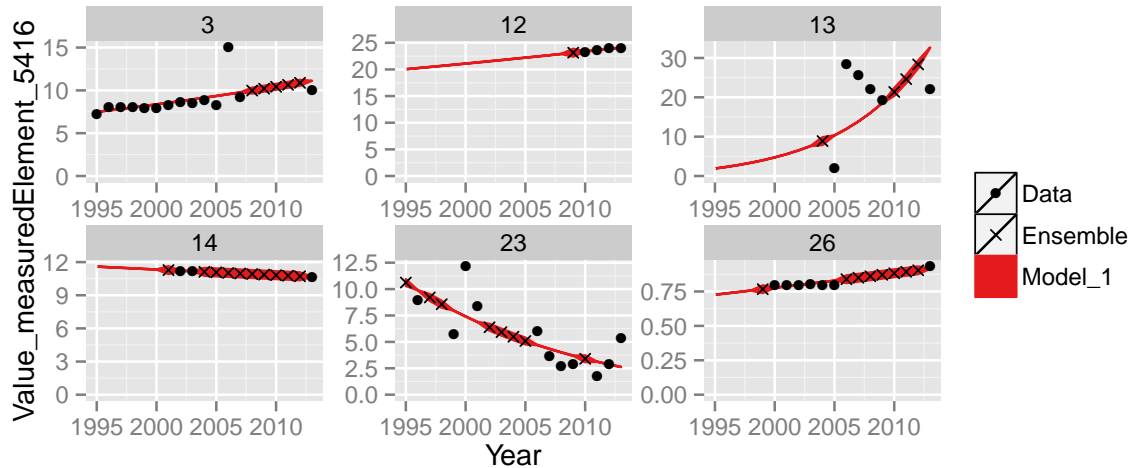


#### 4.2. defaultExp

This algorithm fits the following model:  $\log(Y + 1) = \beta_0 + \beta_1 t$  where  $Y$  is the dependent variable (i.e. production, seed rates, etc)  $t$  is time, and  $\beta_0, \beta_1$  are the estimated coefficients. This model is equivalent to  $Y + 1 = e^{\beta_0 + \beta_1 t}$ , hence the name exponential. The 1 in the formula ensures that  $\log(Y + 1)$  always exists (assuming  $Y \geq 0$ ).

```
model = ensembleModel(model = defaultExp, extrapolationRange = 1,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```

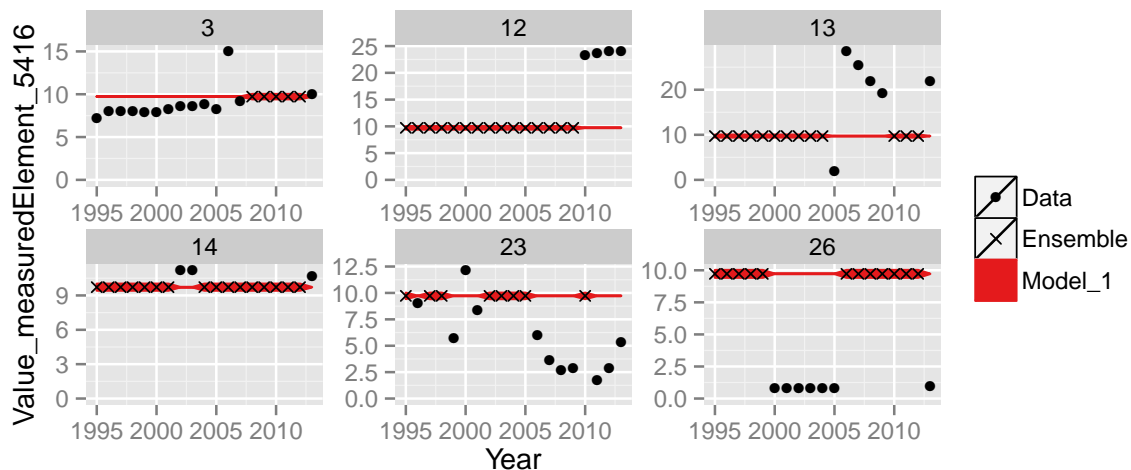




#### 4.3. defaultGlobalMean

This model is quite simple: it computes the mean from all available observations and uses that value to impute any missing values. This model is not recommended for most domains; however, it may perform reasonably well when imputing rates or proportions, as the average may not vary drastically from country to country. A variable like production is very different, values can vary drastically in scale and so a global mean is not appropriate.

```
model = ensembleModel(model = defaultGlobalMean, extrapolationRange = Inf,
                      level = "commodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



Note that in the above figure, the imputed values may appear to be different. However, this is simply due to the different scale in each of the grids; the imputed value is always about 8.

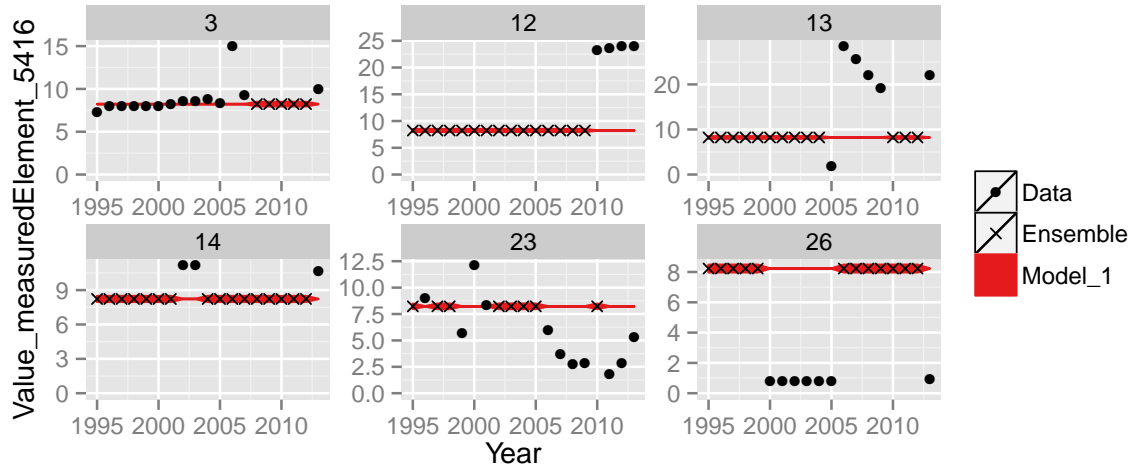
#### 4.4. defaultGlobalMedian

The global median works exactly the same as the global mean, but computes the median instead of the mean. Again, this type of model should only be used when imputing rates or something similar (i.e. no drastic differences in scale across groups).

```

model = ensembleModel(model = defaultGlobalMedian, extrapolationRange = Inf,
                      level = "commodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)

```



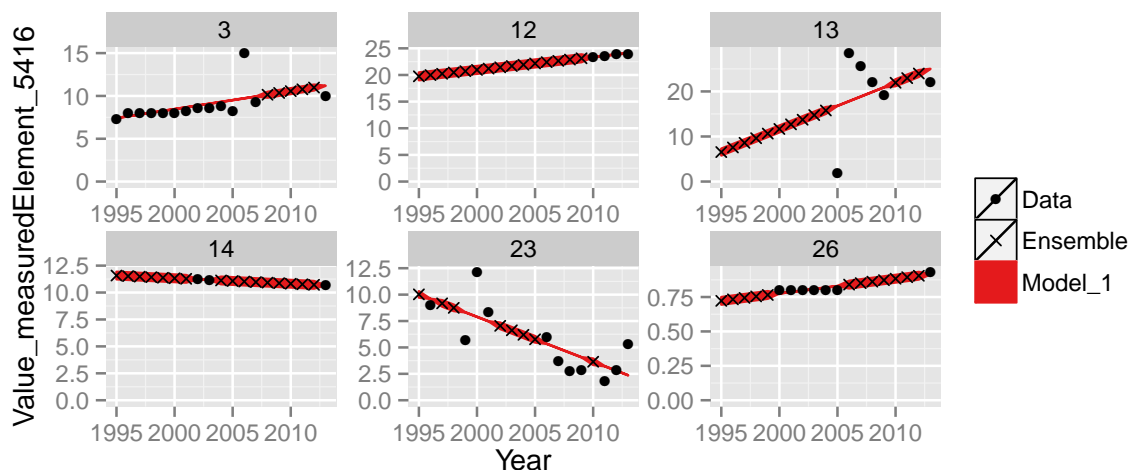
#### 4.5. defaultLm

The defaultLm model uses a simple linear regression model for imputation. It fits a model of the form:  $Y = \beta_0 + \beta_1 t$ , where  $Y$  is the value to impute,  $t$  is the time, and  $\beta_0, \beta_1$  are estimated coefficients.

```

model = ensembleModel(model = defaultLm, extrapolationRange = Inf,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)

```

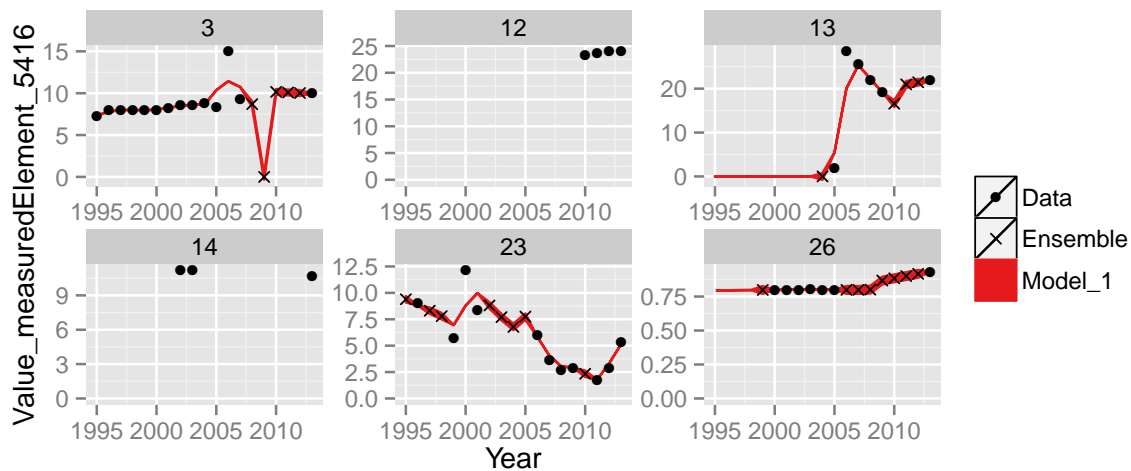


#### 4.6. defaultLoess

The defaultLoess model works by fitting a “local” linear regression model at each point in the model space. The model is local in the sense that the fit at time  $t$  uses only nearby time points, say  $t - k$  to  $t + k$ . Furthermore, points further away from  $t$  are given less weight in the

regression model. This type of model has several tuning parameters such as the size of the neighborhood and the degree of model to fit (i.e. we could fit local linear models, quadratic, etc.). For simplicity, we use a local linear model, and we choose the smallest span possible to allow for the most flexible model. Additionally, the local nature of the loess model means that it likely will not extrapolate well, so the recommended extrapolation range is 1.

```
model = ensembleModel(model = defaultLoess, extrapolationRange = 1,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



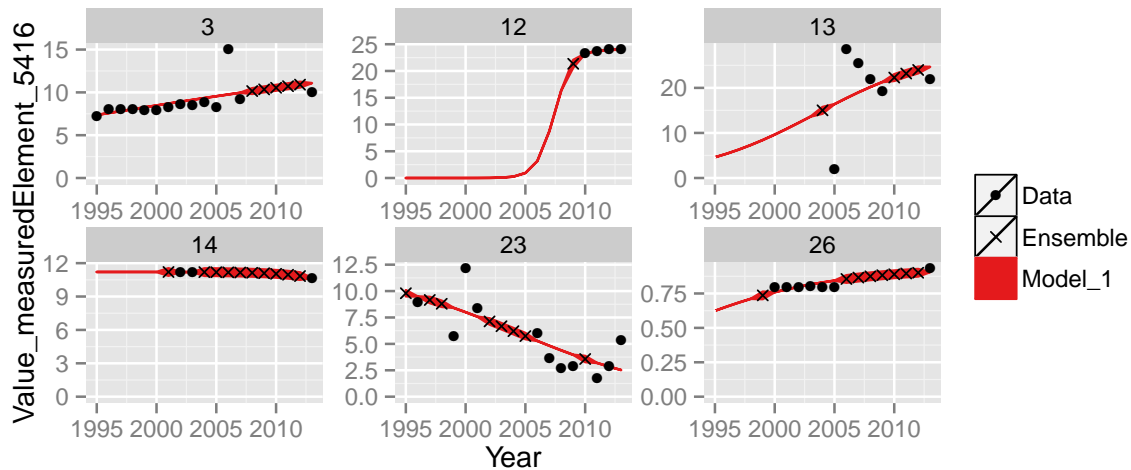
#### 4.7. defaultLogistic

Logistic curves are S-shaped curves of the form

$$f(x) = A + \frac{B}{1 + e^{-C(t-D)}}$$

. These types of functions make sense in scenarios where a variable is increasing but may have some upper bound (i.e. production may increase greatly as technology improves, but there is some maximum production level a country can obtain). This algorithm attempts to first fit all four parameters above via numerical least squares. If that approach fails,  $A$  is assumed to be 0 and numerical least squares are tried again. If that model also fails,  $B$  is assumed to be the largest value and model fitting proceeds via generalized least squares.

```
model = ensembleModel(model = defaultLogistic, extrapolationRange = 1,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```

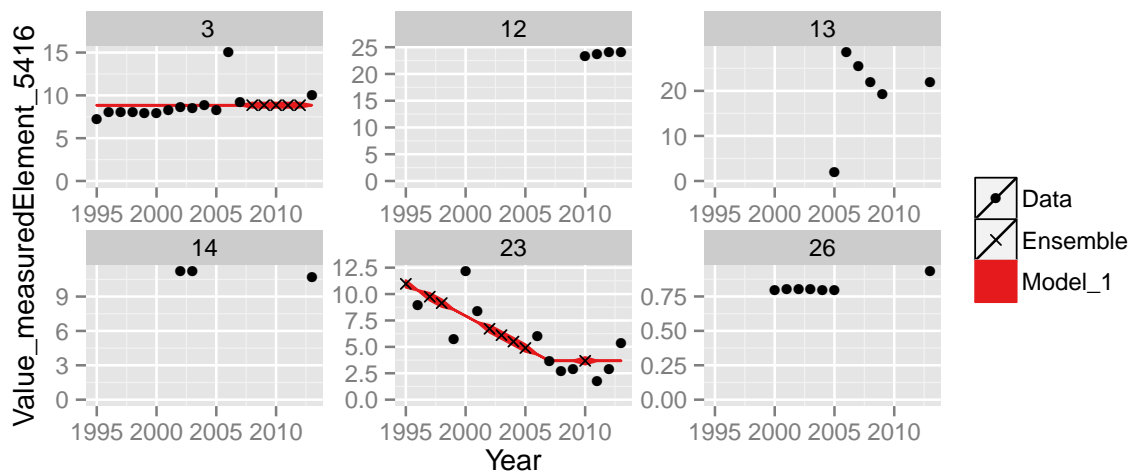


Note: in the first example, the logistic regression decays rapidly to 0. This may not be very reasonable, and thus we recommend using a small extrapolation range for this model.

#### 4.8. defaultMars

The defaultMars model uses a technique known as Multivariate Adaptive Regression Splines (MARS). This algorithm seeks to model the data using piecewise linear regression splines, and it determines the breakpoints of the splines using some optimization criterion. On our sample dataset, we don't see anything too interesting:

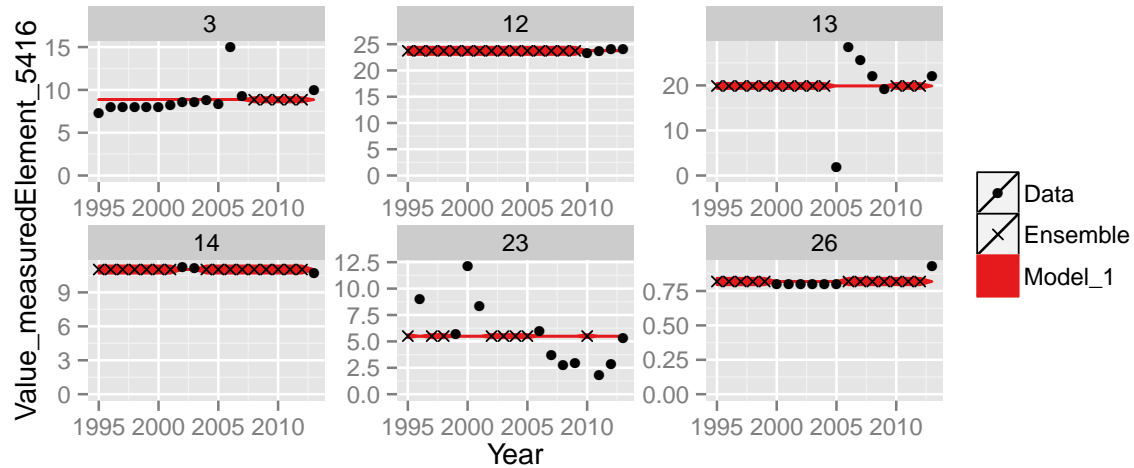
```
model = ensembleModel(model = defaultMars, extrapolationRange = Inf,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



#### 4.9. defaultMean

This model computes a mean on each subset of the data and uses that one value to impute any missing values.

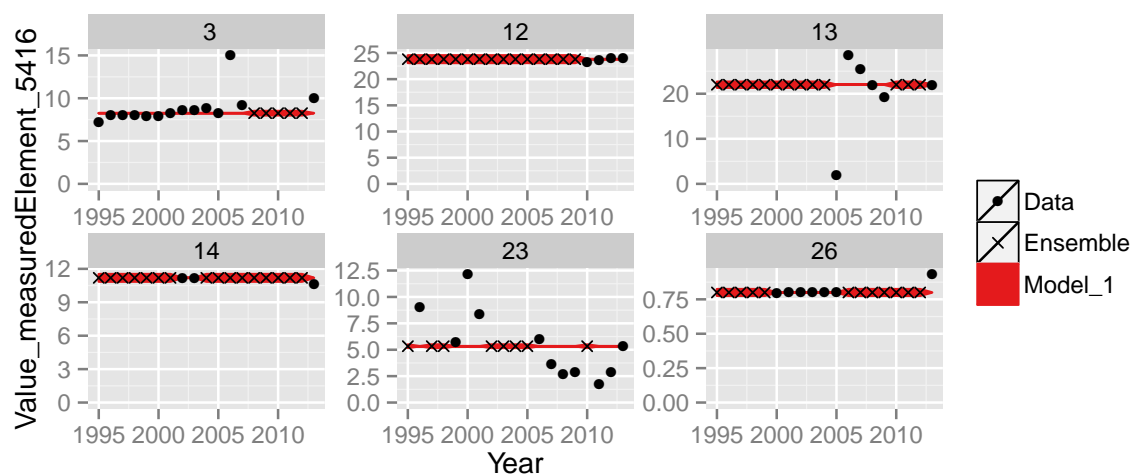
```
model = ensembleModel(model = defaultMean, extrapolationRange = Inf,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



#### 4.10. defaultMedian

This model computes a median on each subset of the data and uses that one value to impute any missing values.

```
model = ensembleModel(model = defaultMedian, extrapolationRange = Inf,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



#### 4.11. defaultMixedModel

The defaultMixedModel is a more complex model that's fit to global datasets (rather than each individual time-series). So, let's use a different dataset to run some analyses with this model:

```

mixedModelData = okrapd[geographicAreaM49 < 100, ]
processProductionDomain(mixedModelData,
  processingParameters = defaultProcessingParameters())
updateMissingFlags = function(data, value, flag, naFlag = "M"){
  missingIndex = which(is.na(data[[value]]))
  invisible(data[missingIndex, `:=`(c(flag), list(naFlag))])
}
updateMissingFlags(data = mixedModelData, value = "Value_measuredElement_5416",
  flag = "flagObservationStatus_measuredElement_5416")

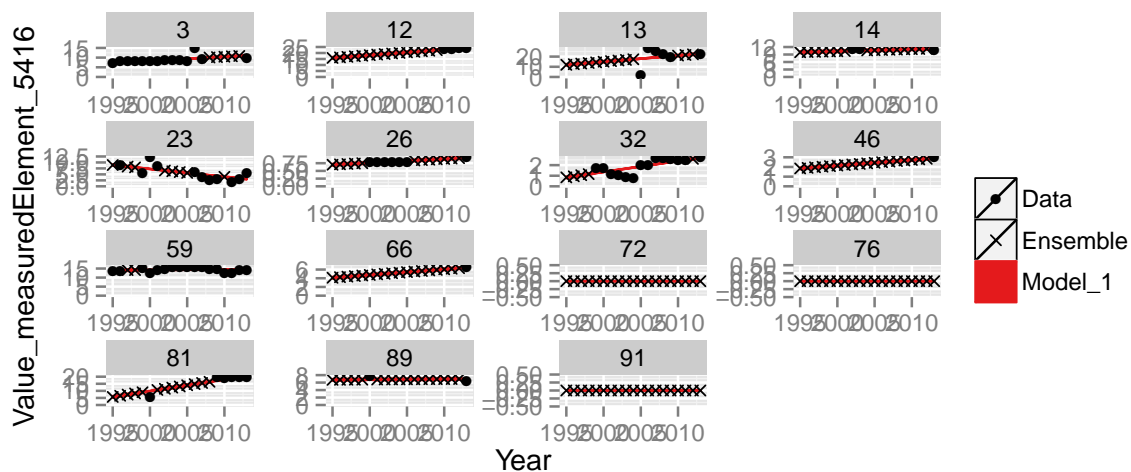
```

Now, let's run the ensemble imputation with the default imputation parameters. This example uses the **lme4** package to fit a mixed model.

```

newParameters = defaultImputationParameters("yield")
newParameters$newImputationColumn = "test"
newParameters$estimateNoData = TRUE
model = ensembleModel(model = defaultMixedModel, extrapolationRange = Inf,
  level = "commodity")
newParameters$ensembleModels = list(model)
imputeVariable(data = mixedModelData, imputationParameters = newParameters)

```



For most cases, we see the same results as we did with the linear regression: a simple least-squares curve is fit to the available data and then that model is used to impute the missing values. However, the mixed model fit to the data can also be used for estimation on time-series where very little data is available, for example on area 46 and 66.

More complex cases are also available: for example, we could fit a hierarchical model (also with the **lme4** package) that allows us to impute countries with missing data based on some hierarchy (for example, continents). In this example, we just made up arbitrary regions.

```

mixedModelData[geographicAreaM49 == "66", Value_measuredElement_5416 := NA]

```

```

##      geographicAreaM49 areaName itemCode itemName timePointYears
##  1:                   3  Albania    430     Okra         1995
##  2:                   3  Albania    430     Okra         1996
##  3:                   3  Albania    430     Okra         1997
##  4:                   3  Albania    430     Okra         1998

```

```

##      5:              3  Albania      430      Okra      1999
##      ---
## 319:              91  Guyana      430      Okra      2009
## 320:              91  Guyana      430      Okra      2010
## 321:              91  Guyana      430      Okra      2011
## 322:              91  Guyana      430      Okra      2012
## 323:              91  Guyana      430      Okra      2013
##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
##      1:              800                                          T
##      2:              600                                          T
##      3:              750                                          T
##      4:              750                                          T
##      5:              740                                          T
##      ---
## 319:              NA                                          M
## 320:              NA                                          M
## 321:              NA                                          M
## 322:              NA                                          M
## 323:              NA                                          M
##      flagMethod_measuredElement_5312 Value_measuredElement_5416
##      1:              NA              7.250000
##      2:              NA              8.000000
##      3:              NA              8.000000
##      4:              NA              8.000000
##      5:              NA              7.972973
##      ---
## 319:              NA              NA
## 320:              NA              NA
## 321:              NA              NA
## 322:              NA              NA
## 323:              NA              NA
##      flagObservationStatus_measuredElement_5416 flagMethod_measuredElement_5416
##      1:              T              NA
##      2:              T              NA
##      3:              T              NA
##      4:              T              NA
##      5:              T              NA
##      ---
## 319:              M              NA
## 320:              M              NA
## 321:              M              NA
## 322:              M              NA
## 323:              M              NA
##      Value_measuredElement_5510 flagObservationStatus_measuredElement_5510
##      1:              5800                                          T
##      2:              4800                                          T
##      3:              6000                                          T
##      4:              6000                                          T
##      5:              5900                                          T
##      ---
## 319:              NA                                          M

```

```

## 320:      7479
## 321:      1881
## 322:      2628
## 323:      3394
##      flagMethod_measuredElement_5510 Value_test flagObservationStatus_test
## 1:      NA      7.250000      NA
## 2:      NA      8.000000      NA
## 3:      NA      8.000000      NA
## 4:      NA      8.000000      NA
## 5:      NA      7.972973      NA
## ---
## 319:      NA      0.000000      I
## 320:      NA      0.000000      I
## 321:      NA      0.000000      I
## 322:      NA      0.000000      I
## 323:      NA      0.000000      I
##      flagMethod_test
## 1:      NA
## 2:      NA
## 3:      NA
## 4:      NA
## 5:      NA
## ---
## 319:      e
## 320:      e
## 321:      e
## 322:      e
## 323:      e

mixedModelData[geographicAreaM49 == "66",
                flagObservationStatus_measuredElement_5416 := "M"]

##      geographicAreaM49 areaName itemCode itemName timePointYears
## 1:      3 Albania      430      Okra      1995
## 2:      3 Albania      430      Okra      1996
## 3:      3 Albania      430      Okra      1997
## 4:      3 Albania      430      Okra      1998
## 5:      3 Albania      430      Okra      1999
## ---
## 319:      91 Guyana      430      Okra      2009
## 320:      91 Guyana      430      Okra      2010
## 321:      91 Guyana      430      Okra      2011
## 322:      91 Guyana      430      Okra      2012
## 323:      91 Guyana      430      Okra      2013
##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
## 1:      800      T
## 2:      600      T
## 3:      750      T
## 4:      750      T
## 5:      740      T
## ---

```



```

## 319:      NA      M
## 320:      NA      M
## 321:      NA      M
## 322:      NA      M
## 323:      NA      M
##      flagMethod_measuredElement_5312 Value_measuredElement_5416
## 1:      NA      7.250000
## 2:      NA      8.000000
## 3:      NA      8.000000
## 4:      NA      8.000000
## 5:      NA      7.972973
## ---
## 319:      NA      NA
## 320:      NA      NA
## 321:      NA      NA
## 322:      NA      NA
## 323:      NA      NA
##      flagObservationStatus_measuredElement_5416 flagMethod_measuredElement_5416
## 1:      T      NA
## 2:      T      NA
## 3:      T      NA
## 4:      T      NA
## 5:      T      NA
## ---
## 319:      M      NA
## 320:      M      NA
## 321:      M      NA
## 322:      M      NA
## 323:      M      NA
##      Value_measuredElement_5510 flagObservationStatus_measuredElement_5510
## 1:      5800      T
## 2:      4800      T
## 3:      6000      T
## 4:      6000      T
## 5:      5900      T
## ---
## 319:      NA      M
## 320:      7479
## 321:      1881
## 322:      2628
## 323:      3394
##      flagMethod_measuredElement_5510 Value_test flagObservationStatus_test
## 1:      NA      7.250000      NA
## 2:      NA      8.000000      NA
## 3:      NA      8.000000      NA
## 4:      NA      8.000000      NA
## 5:      NA      7.972973      NA
## ---
## 319:      NA      0.000000      I
## 320:      NA      0.000000      I
## 321:      NA      0.000000      I

```

```

## 322: NA 0.000000 I
## 323: NA 0.000000 I
##      flagMethod_test
## 1: NA
## 2: NA
## 3: NA
## 4: NA
## 5: NA
## ---
## 319: e
## 320: e
## 321: e
## 322: e
## 323: e

mixedModelData[,region := factor(ifelse(geographicAreaM49 < 15, 1,
                                         ifelse(geographicAreaM49 < 50, 2, 3)))]

##      geographicAreaM49 areaName itemCode itemName timePointYears
## 1: 3 Albania 430 Okra 1995
## 2: 3 Albania 430 Okra 1996
## 3: 3 Albania 430 Okra 1997
## 4: 3 Albania 430 Okra 1998
## 5: 3 Albania 430 Okra 1999
## ---
## 319: 91 Guyana 430 Okra 2009
## 320: 91 Guyana 430 Okra 2010
## 321: 91 Guyana 430 Okra 2011
## 322: 91 Guyana 430 Okra 2012
## 323: 91 Guyana 430 Okra 2013
##      Value_measuredElement_5312 flagObservationStatus_measuredElement_5312
## 1: 800 T
## 2: 600 T
## 3: 750 T
## 4: 750 T
## 5: 740 T
## ---
## 319: NA M
## 320: NA M
## 321: NA M
## 322: NA M
## 323: NA M
##      flagMethod_measuredElement_5312 Value_measuredElement_5416
## 1: NA 7.250000
## 2: NA 8.000000
## 3: NA 8.000000
## 4: NA 8.000000
## 5: NA 7.972973
## ---
## 319: NA NA
## 320: NA NA

```

```

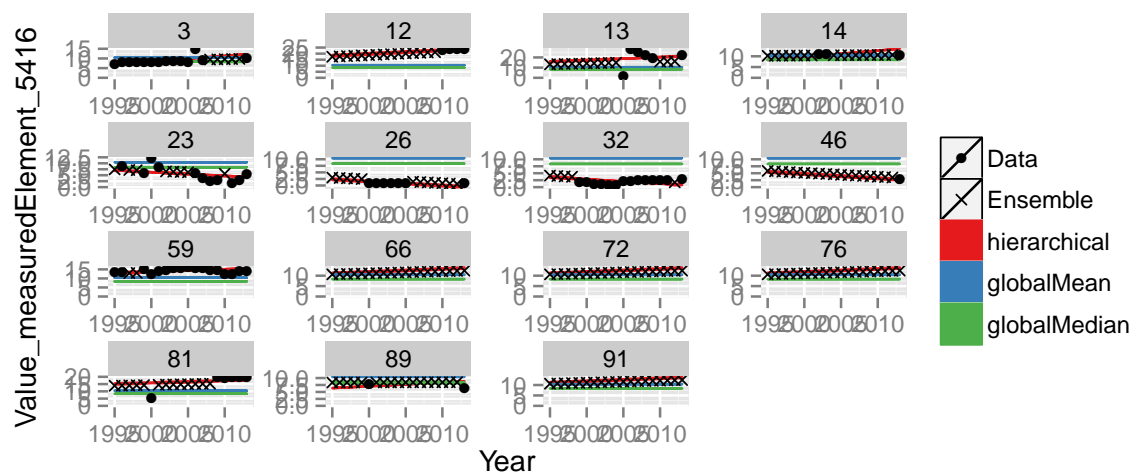
## 321:          NA          NA
## 322:          NA          NA
## 323:          NA          NA
##      flagObservationStatus_measuredElement_5416 flagMethod_measuredElement_5416
## 1:          T          NA
## 2:          T          NA
## 3:          T          NA
## 4:          T          NA
## 5:          T          NA
## ---
## 319:          M          NA
## 320:          M          NA
## 321:          M          NA
## 322:          M          NA
## 323:          M          NA
##      Value_measuredElement_5510 flagObservationStatus_measuredElement_5510
## 1:          5800          T
## 2:          4800          T
## 3:          6000          T
## 4:          6000          T
## 5:          5900          T
## ---
## 319:          NA          M
## 320:          7479
## 321:          1881
## 322:          2628
## 323:          3394
##      flagMethod_measuredElement_5510 Value_test flagObservationStatus_test
## 1:          NA      7.250000          NA
## 2:          NA      8.000000          NA
## 3:          NA      8.000000          NA
## 4:          NA      8.000000          NA
## 5:          NA      7.972973          NA
## ---
## 319:          NA      0.000000          I
## 320:          NA      0.000000          I
## 321:          NA      0.000000          I
## 322:          NA      0.000000          I
## 323:          NA      0.000000          I
##      flagMethod_test region
## 1:          NA      1
## 2:          NA      1
## 3:          NA      1
## 4:          NA      1
## 5:          NA      1
## ---
## 319:          e      3
## 320:          e      3
## 321:          e      3
## 322:          e      3
## 323:          e      3

```

```

formals(defaultMixedModel)$modelFormula = Value_measuredElement_5416 ~
  timePointYears*region + (timePointYears|geographicAreaM49/region)
hierarchical = ensembleModel(model = defaultMixedModel, level = "commodity",
  extrapolationRange = Inf)
globalMean = ensembleModel(model = defaultGlobalMean, level = "commodity",
  extrapolationRange = Inf)
globalMedian = ensembleModel(model = defaultGlobalMedian, level = "commodity",
  extrapolationRange = Inf)
newParameters$ensembleModels = list(hierarchical = hierarchical,
  globalMean = globalMean,
  globalMedian = globalMedian)
imputeVariable(data = mixedModelData, imputationParameters = newParameters)

```



In this complicated example, we can see several different types of imputation occurring. The most common case seems to be the hierarchical model, which simplifies to just a linear regression on countries with enough data to fit the model. The global mean and median don't seem to be as good of models, but in a few cases they provide some value.

Note that we are also able to impute values in region 66, where no data exists (because we deleted the one value that was present before we began). This imputation was performed by looking at the cross-validation error of the three models considered on all the other datasets and averaging the error across all available observations. Generally the mixed model performs best, and so it gets the most weight on this set of data where we have no available data.

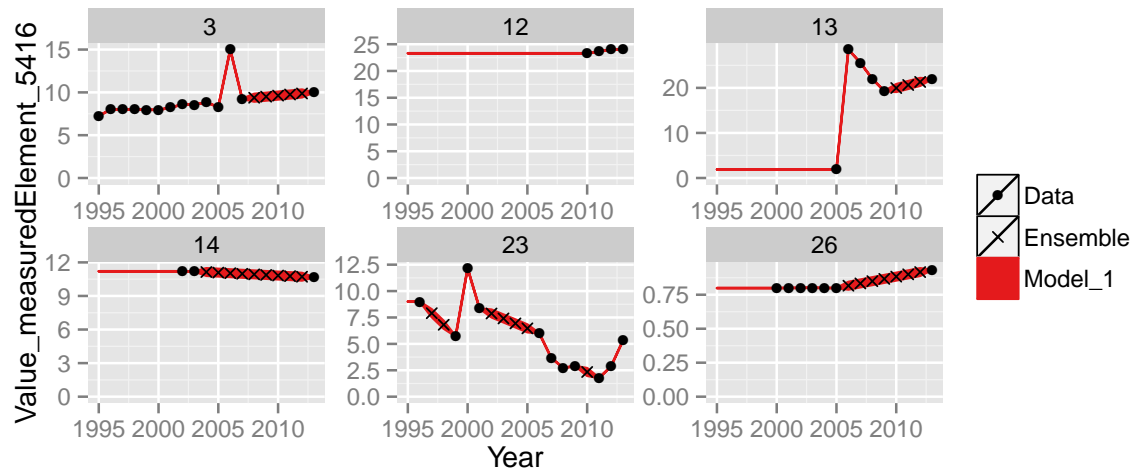
#### 4.12. defaultNaive

This model performs simple linear interpolation between available observations. However, if a missing value is outside the range of the data, then this model estimates that value by carrying back the first observation or carrying forward the last observation (depending on if the missing value is before the available data or after it). Because of this, this model is not recommended for extrapolation (and has a default `extrapolationRange` of 0 in `allDefaultModels`).

```

model = ensembleModel(model = defaultNaive, extrapolationRange = 0,
  level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)

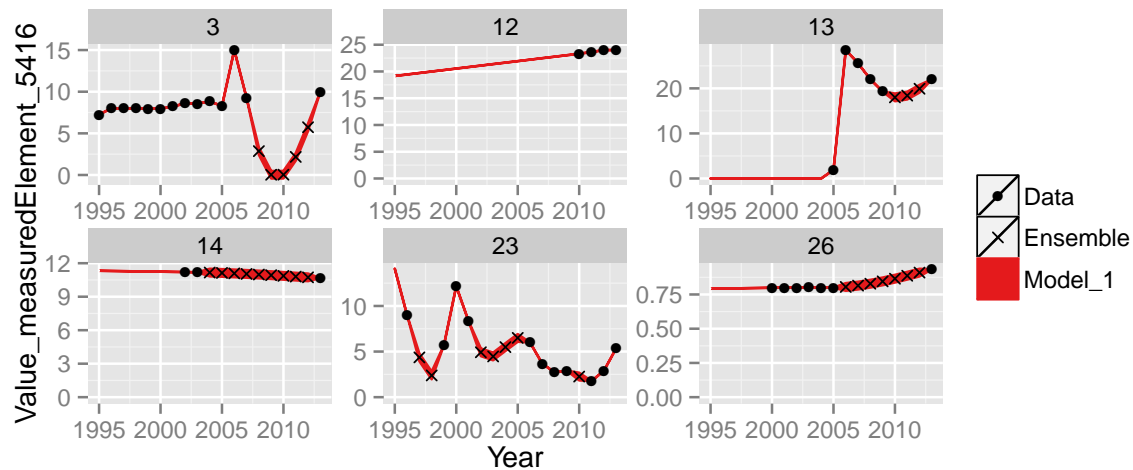
```



#### 4.13. defaultSpline

The defaultSpline model uses the spline function from the stats package (part of base) to fit a spline to the available observations. Missing observations are then imputed by the spline estimate at that location.

```
model = ensembleModel(model = defaultSpline, extrapolationRange = 0,
                      level = "countryCommodity")
imputationParameters$ensembleModels = list(model)
imputeVariable(data = exampleData, imputationParameters = imputationParameters)
```



#### Affiliation:

Joshua M. Browning

Economics and Social Statistics Division (ESS)

Economic and Social Development Department (ES)

Food and Agriculture Organization of the United Nations (FAO)

Viale delle Terme di Caracalla 00153 Rome, Italy

E-mail: [joshua.browning@fao.org](mailto:joshua.browning@fao.org)

URL: <https://svn.fao.org/projects/SWS/RModules/faoswsImputation/>