Handling Missing Stand Attribute Cases

– A documented workflow for ensuring valid effectiveness metrics in Optimizer

Workflow developed by Jeremy Fried and Cat Ducat
16 December 2024

Background and Rationale

Stand attributes calculated in FVS and loaded from FVSOut.DB to BioSum can have missing values when an attribute is undefined. For example, if calculating volume of trees expected to survive a fire with given surface flame length as a percent of the volume of all trees in that stand at the point in time being considered, as we commonly do using parameters derived from the First Order Fire Effects Model (FOFEM) to estimate mortality volume and subtracting that from all live volume, a stand that has no live trees cannot have a meaningful value for this attribute. The stand certainly does not have 100 percent survival if there are not trees present, nor can it have 0 percent survival, because there are no trees there to die. This attribute may show up as null in such instances, and if the attribute is to be used to characterize fire resistance or treatment effectiveness, plans must be made to account for these missing cases.

Other FVS calculated values, including several calculated by FFE and stored in the POTFIRE table, can be missing or be coded with a missing value flag, which in the case of canopy base height, is –1. This is typically assigned when there are no live trees or no live trees with a diameter greater than 1 inch (which prevents calculation of a meaningful canopy base height. Again, -1 is not a valid height nor should it be included in any weighted average of values over time.

Eventually, BioSum will be automated to handle missing cases like these. Until that time, a workaround is needed to address this issue when calculating weighted attribute values and when defining effectiveness in Optimizer.

The weighted average is calculated as the sum of the products of the analyst assigned weight (that can differ by PRE and POST and cycle) and the data value. Take the following weighted average calculation for example:

Cycle	Pre/Post	Data Value	Weight	Weighted Value (Data value x Weight)
1	PRE	-1	0	0
1	POST	-1	0.125	-0.125
2	PRE	13	0.125	1.625
2	POST	15	0.125	1.875
3	PRE	20	0.125	2.5
3	POST	20	0.125	2.5
4	PRE	23	0.125	2.875
4	POST	24	0.250	6.0

The calculation would be:

[(0 × (-1)) + (0.125× (-1)) + (0.125 × 13) + (0.125 × 15) + (0.125 × 20) + (0.125 × 20) + (0.125 × 23) + (0.25 × 24)] / (0+0.125+0.125+0.125+0.125+0.125+0.125+0.25) = **17.25**

However, BioSum recognizes negative values as invalid and substitutes a zero into the calculation such that the calculation becomes:

[(0×(0)) + (0.125×(0)) + (0.125×13) + (0.125×15) + (0.125×20) + (0.125×20) + (0.125×23) +(0.25×24)] / (0+0.125+0.125+0.125+0.125+0.125+0.125+0.25) = **17.375**

In this case, Cycle 1 Post was -1 but was treated as if null. BioSum's calculations currently substitute zeros for nulls in the weighed calculation, so this pulls the weighted value down from where we might want it to be (a weighted combination of the non-null values). For now, some data transformation is necessary to correct the weighted values calculated by BioSum in the calculated variables workflow.

A query-based workflow to adjust BioSum calculated values for the undefined cases

The following workflow generates a correction factor that can be applied, via query, to the already calculated weighted average of variables that contain null or negative values at some time points. CAUTION: recalculating weighted variable will overwrite the corrections made with this workflow and the workflow will need to be repeated.

In this example, we will first identify missing cases (NULLs or -1s, which we know that BioSum will handle the same as nulls, in the PREPOST table variables) in the variable we seek to correct. In this example, we'll do this for SURVRATE (in the FVS_CUSTOM

table), by calculating the weights of the missing cases, adjust the weighted average using the correction factor 1/(1-sum of weights of missing cases) when there are 4 or less missing values for a stand package combination and remove cases where values at more than four of the eight time points were NULL. If all variables with missing issues that are needed in the analysis are first copied to the PRE_FVS_CUSTOM and POST_FVS_CUSTOM tables, then this simplifies missing case handling by consolidating the workflow into one table (versus having to process SUMMARY, POTFIRE, COMPUTE, etc. separately).

Tables Needed in this example:

1. POST_FVS_CUSTOM (PREPOST_FVSOUT.db)
2. POST_FVS_CUSTOM_WEIGHTED (prepost_fvs_weighted.db, in Optimizer\db)
3. PRE_FVS_CUSTOM (PREPOST_FVSOUT.db)
4. PRE_FVS_CUSTOM_WEIGHTED (prepost_fvs_weighted.db, in Optimizer\db)
5. Cond (in \db\master.mdb and only needed if executing the optional query 2)

1. Append pre- and post- tables to a new table "PRE_FVS_CUSTOM"
    a. **Make FVS_CUSTOM query:** *SELECT "Pre" AS PrePost, PRE_FVS_CUSTOM.* INTO FVS_CUSTOM FROM PRE_FVS_CUSTOM;*
    b. **Append FVS_CUSTOM query:** *INSERT INTO FVS_CUSTOM (PrePost) SELECT POST_FVS_CUSTOM.*, "Post" AS PrePost FROM POST_FVS_CUSTOM;*
    c. Creates an FVS_CUSTOM table that is twice as long because both pre and post values are present
2. This is a diagnostic step. Skip it if you know you are only concerned about what you have placed into FVS_CUSTOM (the table this workflow addresses as an example) including variables known to have undefined cases. You will need a number of tables beyond the list above, depending on which ones contain variables to be evaluated for potential undefined values at one or more simulated time points. See query for examples of tables that may be needed.

   This query pulls together some of the more commonly useful metrics, some of which may be undefined at some point during the simulation. In essence, you are looking for cases where the attribute value is null or carries a missing value

flagged by, for example, a -1. The POTFIRE variables seem to be some of the 'problem' variables that carry these -1 flags for missing value

a. ***MakeMissingCaseExplore query:*** *SELECT cond.biosum_cond_id, PRE_FVS_SUMMARY.Year, PRE_FVS_POTFIRE.rxpackage, PRE_FVS_POTFIRE.rxcycle, PRE_FVS_POTFIRE.SURF_FLAME_SEV, PRE_FVS_POTFIRE.FIRE_TYPE_SEV, PRE_FVS_POTFIRE.TORCH_INDEX, PRE_FVS_POTFIRE.CANOPY_HT, PRE_FVS_POTFIRE.CANOPY_DENSITY, PRE_FVS_POTFIRE.POT_SMOKE_SEV, PRE_FVS_POTFIRE.MORTALITY_VOL_SEV, PRE_FVS_POTFIRE.MORTALITY_BA_SEV, PRE_FVS_COMPUTE.MORTRATE, PRE_FVS_COMPUTE.SURVRATE, PRE_FVS_COMPUTE.ALLVOL, cond.ba_ft2_ac, PRE_FVS_SUMMARY.BA, PRE_FVS_SUMMARY.TPA, PRE_FVS_SUMMARY.QMD, PRE_FVS_SUMMARY.RTPA, PRE_FVS_SUMMARY.TCUFT, PRE_FVS_SUMMARY.RTCUFT INTO MissingCaseExplore FROM ((cond INNER JOIN PRE_FVS_POTFIRE ON cond.biosum_cond_id = PRE_FVS_POTFIRE.biosum_cond_id) INNER JOIN PRE_FVS_SUMMARY ON (PRE_FVS_POTFIRE.rxcycle = PRE_FVS_SUMMARY.rxcycle) AND (PRE_FVS_POTFIRE.rxpackage = PRE_FVS_SUMMARY.rxpackage) AND (PRE_FVS_POTFIRE.biosum_cond_id = PRE_FVS_SUMMARY.biosum_cond_id)) INNER JOIN PRE_FVS_COMPUTE ON (PRE_FVS_POTFIRE.rxcycle = PRE_FVS_COMPUTE.rxcycle) AND (PRE_FVS_POTFIRE.rxpackage = PRE_FVS_COMPUTE.rxpackage) AND (PRE_FVS_POTFIRE.biosum_cond_id = PRE_FVS_COMPUTE.biosum_cond_id) WHERE (((cond.cond_status_cd)=1) AND ((cond.reservcd)=0));*

3. ***MakePreFVSCustomWeightedTbl query:***

```
SELECT PRE_FVS_CUSTOM_WEIGHTED.* INTO
PRE_FVS_CUSTOM_WEIGHTED_TBL
FROM PRE_FVS_CUSTOM_WEIGHTED;
```

a. Rather than working with linked tables, use local tables since we are manipulating values – don't want to change raw data
b. Do the same thing for the POST_FVS_CUSTOM_WEIGHTED_TBL table

c. To validate successful creation of these tables, compare row counts and the first few values of a variable of interest between the linked and local tables; if there are different values, then the tables are not populated with the same data

4. **Null_Wt_sum_per_combo:** Count the sum of the weights assigned to time points for which your attribute of interest is null or negative—essentially, when it is undefined as a meaningful attribute. This requires that a weight matrix be loaded into a table called WtMtx—these are the same weights used when building the corresponding weighted variable in the first task of the Optimizer module. WtMtx has three columns (populated here with the weights for this example) and it is essential that the first two be Short Text and Wt must be Double:

WtMtx		
RxCycle	**PrePost**	**Wt**
Short Text	Short Text	Double
1	PRE	0
1	POST	0.125
2	PRE	0.125
2	POST	0.125
3	PRE	0.125
3	POST	0.125
4	PRE	0.125
4	POST	0.25

SELECT FVS_CUSTOM.biosum_cond_id, FVS_CUSTOM.rxpackage, Sum(IIf(IsNull([SURVRATE]),[Wt],IIf([SURVRATE]<0,[Wt],0))) AS NullWt
FROM FVS_CUSTOM INNER JOIN WtMtx ON (FVS_CUSTOM.rxcycle = WtMtx.RxCycle) AND (FVS_CUSTOM.PrePost = WtMtx.PrePost)
GROUP BY FVS_CUSTOM.biosum_cond_id, FVS_CUSTOM.rxpackage;

Produces a view containing, for each combination of stand and RxPkg, the sum of the weights corresponding to undefined attribute values.

Null_wt_sum_per_combo		
biosum_cond_id	**rxpackage**	**NullWt**
120125302020070082187 0002	873	0
120125302020070082187 0002	874	0
120125302020070082187 0002	999	0

Null_wt_sum_per_combo		
biosum_cond_id	**rxpackage**	**NullWt**
120125302020070083877000 1	873	0.125
120125302020070083877000 1	874	0.125
120125302020070083877000 1	999	0.125

5. Adjust the current attribute weighted averages, that were already calculated in BioSum when calculated variables were defined or last recalculated, by applying a correction factor. We'll use the SURVRATE attribute as an example. This query identifies biosum_cond_id/rxpackage combinations (for both PRE and POST table cases) that have more than 4 NULL values out of the 8 PRE and POST cycle time points—in other words, where the attribute is undefined for more than half of the time points. The query creates a column called "ReplacementSurvRate" that represents the current SurvRate values multiplied by the correction factor that adjusts for the components of the weighted average calculated by BioSum that were null or negative. For cases where more than half of the values at the pre and post cycle time points are NULL, we intentionally recode the weighted attribute (in this case SURVRATE) as NULL. IMPORTANT: Effectiveness logic will be needed in optimizer to "trap" for these null cases so that they are not inadvertently characterized as effective owing to artifacts of how nulls play out in the Boolean logic of optimizer (probably as zeroes, which could generate errors of declaring effectiveness or ineffectiveness that is not real).

   Four queries are required to calculate and apply these adjustments: one each, for PRE and POST weighted tables to build local Access tables with the replacement values and one each to insert values from those replacement tables into the SQLite PRE and POST weighted tables. Attempting to query (for purposes of calculation of the replacement values) and actually replace the values in the SQLite tables in the same query seems ill-advised, if not impossible.

5.1 PostReplaceCalc_Make_Table Query:

SELECT Null_Wt_sum_per_combo.biosum_cond_id,
Null_Wt_sum_per_combo.rxpackage, Null_Wt_sum_per_combo.NullWt,
IIf([NullWt]<0.5,1/(1-[NullWt]),Null) AS AdjFactor,
POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1,
[AdjFactor]*[SURVRATE_1] AS ReplacementSurvRate INTO
PostAdjustedValues

FROM Null_Wt_sum_per_combo INNER JOIN
POST_FVS_CUSTOM_WEIGHTED_TBL ON
(Null_Wt_sum_per_combo.rxpackage =
POST_FVS_CUSTOM_WEIGHTED_TBL.rxpackage) AND
(Null_Wt_sum_per_combo.biosum_cond_id =
POST_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id)

WHERE (((POST_FVS_CUSTOM_WEIGHTED_TBL.rxcycle)="1"));

5.2 PreReplaceCalc Make Table Query:

SELECT Null_Wt_sum_per_combo.biosum_cond_id,
Null_Wt_sum_per_combo.rxpackage, Null_Wt_sum_per_combo.NullWt,
IIf([NullWt]<0.5,1/(1-[NullWt]),Null) AS AdjFactor,
PRE_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1,
[AdjFactor]*[SURVRATE_1] AS ReplacementSurvRate INTO PreAdjustedValues

FROM Null_Wt_sum_per_combo INNER JOIN
PRE_FVS_CUSTOM_WEIGHTED_TBL ON
(Null_Wt_sum_per_combo.rxpackage =
PRE_FVS_CUSTOM_WEIGHTED_TBL.rxpackage) AND
(Null_Wt_sum_per_combo.biosum_cond_id =
PRE_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id)

WHERE (((PRE_FVS_CUSTOM_WEIGHTED_TBL.rxcycle)="1"));

NOTE: Prior to running the update queries below, save a copy of the local prepost weighted tables made in Step 3 (PRE_FVS_CUSTOM_WEIGHTED_TBL and POST_FVS_CUSTOM_WEIGHTED_TBL), renaming them to "PRE_FVS_CUSTOM_WEIGHTED_TBLb4fix" and "POST_FVS_CUSTOM_WEIGHTED_TBLb4fix". Saving a copy of these tables prior to running the update queries allows you to compare the 'new' weighted tables against the originals.

5.3 UpdatePost Query: UPDATE PostAdjustedValues INNER JOIN POST_FVS_CUSTOM_WEIGHTED_TBL ON (PostAdjustedValues.rxpackage = POST_FVS_CUSTOM_WEIGHTED_TBL.rxpackage) AND (PostAdjustedValues.biosum_cond_id = POST_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id) SET POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1 = [ReplacementSurvRate] WHERE (((POST_FVS_CUSTOM_WEIGHTED_TBL.rxcycle)="1"));

5.4 UpdatePre Query:

UPDATE PreAdjustedValues INNER JOIN PRE_FVS_CUSTOM_WEIGHTED_TBL ON PreAdjustedValues.biosum_cond_id = PRE_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id SET PRE_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1 = [ReplacementSurvRate] WHERE (((PreAdjustedValues.rxpackage)="999") AND ((PRE_FVS_CUSTOM_WEIGHTED_TBL.rxcycle)="1"));

This will update the local Access tables.

Note that in the PRE_FVS weighted tables, the value for EVERY RxPkg is the value from the reference package (usually 999, grow-only).

After updating the local Access tables, a query like this will demo what got changed (if you first, before running the Update queries, make a copy of the local table and call it, for example, POST_CUSTOM_WEIGHTED_TBL_b4fix):

Query: CheckChangesInPost

SELECT POST_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id, POST_FVS_CUSTOM_WEIGHTED_TBL.rxpackage, POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1, POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.SURVRATE_1

FROM POST_FVS_CUSTOM_WEIGHTED_TBL INNER JOIN POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix ON (POST_FVS_CUSTOM_WEIGHTED_TBL.rxcycle = POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.rxcycle) AND (POST_FVS_CUSTOM_WEIGHTED_TBL.rxpackage = POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.rxpackage) AND

(POST_FVS_CUSTOM_WEIGHTED_TBL.biosum_cond_id = POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.biosum_cond_id)

WHERE (((POST_FVS_CUSTOM_WEIGHTED_TBL.rxcycle)="1") AND (([POST_FVS_CUSTOM_WEIGHTED_TBL].[SURVRATE_1]<>[POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix].[SURVRATE_1])=True)) OR (((POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1) Is Null) AND (Not (POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.SURVRATE_1) Is Null)) OR ((Not (POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1) Is Null) AND ((POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.SURVRATE_1) Is Null));

This query (CheckChangesInPost) identifies cases where SURVRATE was changed or where it was null and became not null, and vice versa, yielding this expected output:

CheckChangesInPost

biosum_cond_id	rxpackage	POST_FVS_CUSTOM_WEIGHTED_TBL.SURVRATE_1	POST_FVS_CUSTOM_WEIGHTED_TBL_b4fix.SURVRATE_1
120125302020470058968 0001	873	0.0299	0.0261625
120125302020470058968 0001	874	2.75571428571429E-02	0.0241125
120125302020470058968 0001	999	0.0273	0.0238875
120125302020470081109 0001	873	5.17857142857143E-02	0.0453125
120125302020470081109 0001	874	5.17857142857143E-02	0.0453125
120125302020470081109 0001	999	5.17857142857143E-02	0.0453125
120125306010070052701 0001	873	0.2	0.175
120125306010070052701 0001	874	0.2	0.175
120125306010070052701 0001	999	0.2	0.175
120125306010070086464 0002	873	0.2	0.175
120125306010070086464 0002	874	0.2	0.175
120125306010070086464 0002	999	0.2	0.175
120125306010070097178 0001	873		0
120125306010070097178 0001	874		0

If you like the result, then drop the _TBL from these last two queries to update the SQLite version of these tables, e.g., for the POST table:

UPDATE PostAdjustedValues INNER JOIN POST_FVS_CUSTOM_WEIGHTED ON (PostAdjustedValues.rxpackage = POST_FVS_CUSTOM_WEIGHTED.rxpackage) AND

(PostAdjustedValues.biosum_cond_id =
POST_FVS_CUSTOM_WEIGHTED.biosum_cond_id) SET
POST_FVS_CUSTOM_WEIGHTED.SURVRATE_1 = [ReplacementSurvRate]

WHERE (((POST_FVS_CUSTOM_WEIGHTED.rxcycle)="1"));

The adjustment of the targeted column is now permanent, unless the calculated values are recalculated using the recalculate button in optimizer. If they are recalculated, they will need to be adjusted again.

This workflow can be adapted for any FVS PREPOST tables by editing the series of queries presented above with the appropriate table names.

When resources permit, perhaps in 2026, BioSum code will be adapted to automate most of this workflow.