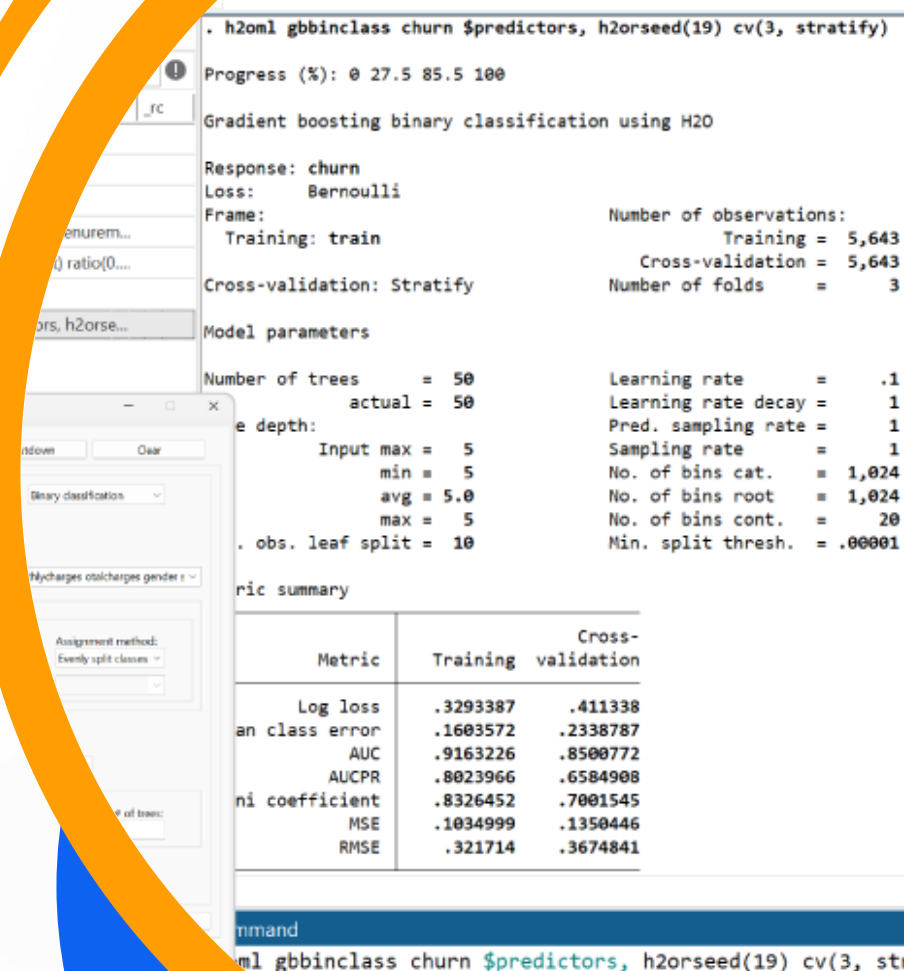


Teaching STATA through real failure

What went wrong, what worked, and
what we changed

Presented by
George Anyumba, DrPH
Paris Johnson, DrPH

Presented to
STATA 2026



The image shows a screenshot of the STATA H2O interface. The main window displays the results of a gradient boosting binary classification model using H2O. The command used is `. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify)`. The progress bar shows 100% completion. The response variable is 'churn', and the loss function is 'Bernoulli'. The model parameters are listed, including the number of trees (50), learning rate (0.1), and maximum depth (5). A summary table compares training and cross-validation metrics.

Metric	Training	Cross-validation
Log loss	.3293387	.411338
an class error	.1603572	.2338787
AUC	.9163226	.8500772
AUCPR	.8023966	.6584908
ni coefficient	.8326452	.7001545
MSE	.1034999	.1350446
RMSE	.321714	.3674841

The command window at the bottom shows the command: `. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify)`.

The Error That Changed Our Analysis

Our first analysis ran perfectly: no errors, no warnings, and a strong, significant result. It was also misleading. Then one command stopped us

Results

```
. isid patient_id visit_dt dept  
variables patient_id visit_dt dept do not uniquely identify the observations  
r(459);
```

This message told us our data was not structured the way we believed. One visit was stored as several rows, and our results were built on that mistake.

This presentation will cover:

- ✓ **What went wrong:** how a clean-running analysis produced a misleading answer
- ✓ **What worked:** the questions and Stata checks that exposed the problem
- ✓ **What we changed:** how we rebuilt the workflow and how we teach it now



CASE ANALYSIS

Anysoma Health Group is a community clinic in Brooklyn with four departments. Leadership launched evening walk-in hours and asked one question: did evening hours improve 30-day follow-up? The answer lived in four data sources:

- ✓ EHR Encounter Log
- ✓ Scheduling System
- ✓ Lab Results Feed
- ✓ Enrollment Registry
- ✓ All Data Simulated

SCAN HERE



Running the Analysis

With a clear question and four data sources, the path looked simple. Load the visits, add patient information, and model follow-up. The entire analysis took five lines of Stata and ran without a single error or warning. Here is what the first pass did:

Load the Encounter Data

The EHR encounter log became the starting dataset, on the assumption that each row represented one clinic visit.

Merge Patient Demographics

Age and sex were added from the enrollment registry. Records that did not match were dropped, a routine step in most tutorials.

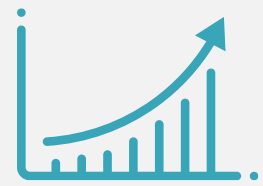
Model 30-Day Follow-Up

A logistic regression estimated whether evening walk-in hours were associated with completing follow-up, adjusting for age and sex.

```
* first pass
. use encounters, clear
. merge m:1 patient_id
> using registry
. keep if _merge == 3
. logit fu30 i.pilot
> age i.female, or
```

The Results

The model ran cleanly, and the output looked like a success story. Every number pointed in the same direction:



Strong Effect

Evening walk-in hours were linked to 84% higher odds of completing 30-day follow-up (odds ratio 1.84).



High Follow-Up

71.4% of records showed completed follow-up within 30 days.



Highly Significant

$p < 0.001$, with a tight 95% confidence interval of 1.71 to 1.98.



Large Sample

$N = 48,212$, a sample big enough to feel trustworthy.

```
. logit fu30 i.pilot age i.female, or
```

Logistic regression

Number of obs = 48,212

LR chi2(3) = 412.87

Prob > chi2 = 0.0000

Log likelihood = -28648.6

Pseudo R2 = 0.0072

	fu30	Odds ratio	Std. err.	z	P> z	[95% conf. int.]
1.pilot		1.840	.0688	16.30	0.000	1.710 1.980
age		1.006	.0011	5.47	0.000	1.004 1.008
1.female		1.213	.0262	8.94	0.000	1.163 1.265
_cons		.842	.0391	-3.70	0.000	.769 .922



Why Clean Output Can Still Be Wrong

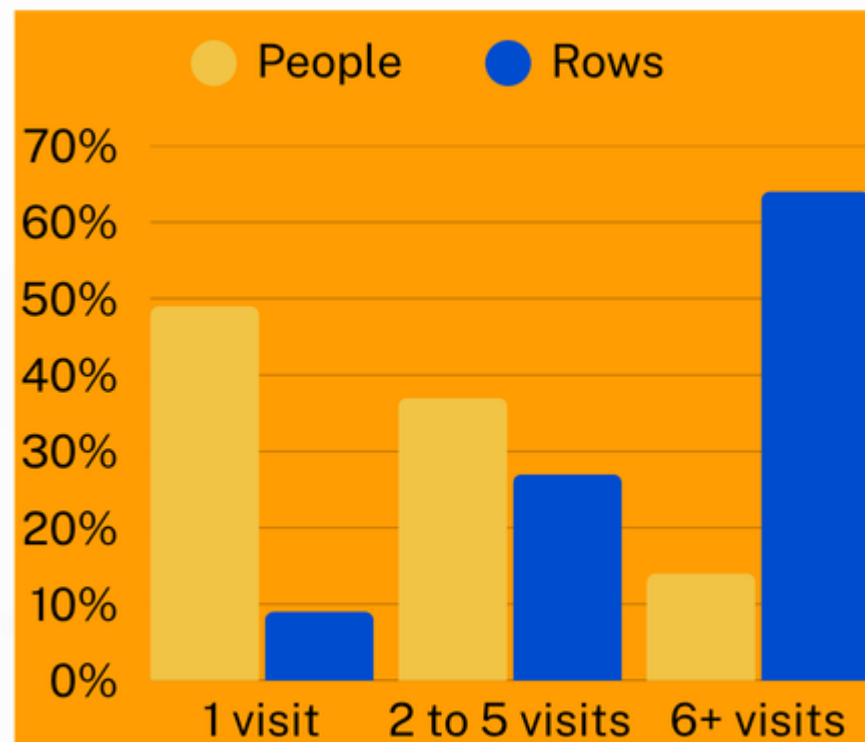
Our first analysis of Anysoma Health Group ran without a single error. It produced a strong, significant result that looked ready to report. But clean code is not the same as correct analysis. Five hidden problems shaped our answer before we ever ran the model:

- ✓ **One Visit, Many Rows**
- ✓ **One Patient, Several IDs**
- ✓ **Merges That Drop Data Silently**
- ✓ **The Wrong Unit of Analysis**
- ✓ **Confidence Without Checks**

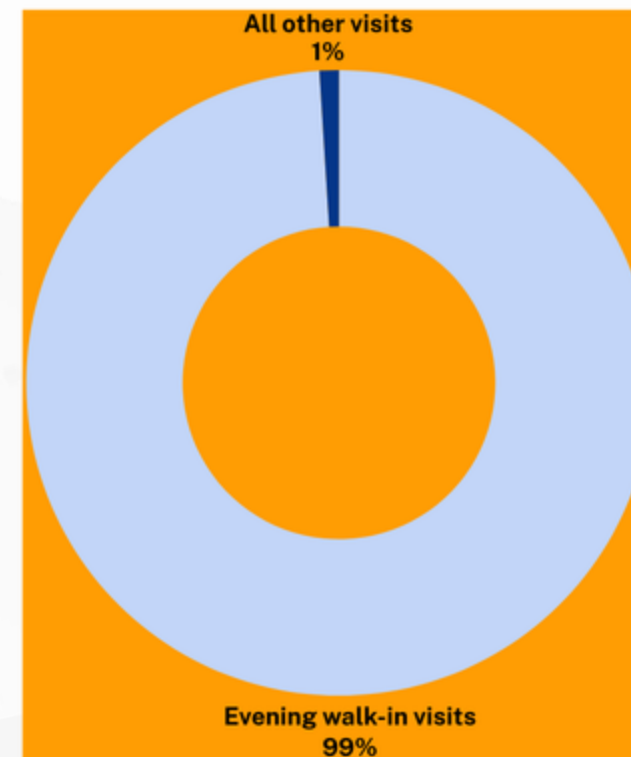
What Was Hiding in the Data

When we stopped trusting the output and looked at the data itself, the problems were visible right away. The numbers told a very different story about who was actually driving our results. Here is what we found:

Who Is Doing the Voting?



Who the Merge Removed



One Visit, Many Rows

Each workflow step was its own row. A single visit became four or five rows, and every row counted as a vote.

One Patient, Several IDs

Evening walk-ins received a kiosk ID the registry did not recognize, so one person could appear as several.

The Merge Removed the Pilot

Keeping only matched records deleted almost every evening walk-in visit, which was the exposure we were trying to study.

What Is N?

48,212 rows, 21,530 visits, 11,237 IDs, 9,406 people. The model used rows; leadership asked about people.

What Worked: Asking Simple Questions

We did not need a new model or a new method. We needed to stop trusting the output and ask the data two simple questions. Both answers had been available in Stata the whole time.

```
. isid patient_id visit_dt dept  
variables patient_id visit_dt  
dept do not uniquely identify  
the observations  
r(459);
```

```
> using registry  
  
Result      Number of obs  
-----  
Not matched      5,740  
from master      5,428 (_merge==1)  
from using        312 (_merge==2)  
  
matched      48,212  
( m
```

Ask What One Row Is

Before modeling again, we ran **duplicates report** and found many rows sharing the same patient, date, and department. Then we used **isid** to test our belief that each row was one visit. Stata answered with r(459): these variables do not uniquely identify the observations. For the first time, the data disagreed with us out loud. A quick count showed that each visit had been split into four or five rows.

Read the Merge as a Finding

The merge table had been telling us something from the start: 5,428 encounter rows never matched the registry. Instead of dropping them, we asked who they were. A simple cross-tab of the pilot flag by merge status showed that almost every unmatched row was an evening walk-in with a kiosk ID. The records we treated as housekeeping were the heart of the evaluation.

What We Changed: Rebuilding the Workflow

Once we knew what was wrong, we rebuilt the analysis in the order the data required, not the order the tutorial suggested. Structure came first. The model came last. Every step now states what one row represents and checks it before moving on.

Fix the Structure First

We started by linking every ID to one person. A crosswalk connected EHR, kiosk, and lab IDs to a single `person_id`, and `merge ..., assert(match using)` stopped the do-file if any encounter failed to link. Then we used `collapse` to turn four or five workflow rows into one row per visit, and confirmed it with `isid person_id visit_dt dept.` No record was dropped without an explanation.

Answer the Question That Was Asked

Leadership asked about people, so we analyzed people. We kept each patient's first visit, which defines whether they were exposed to evening hours, and confirmed one row per person with `isid person_id`. Only then did we run the same logistic model as before. We also tested the alternative: keeping every visit and using `vce(cluster person_id)` so repeat visits were not treated as independent.

Quality Checks Built into the Code

Our first analysis depended on someone noticing a problem. The rebuilt do-file does not. We added checks directly into the code, so the data is tested every time the analysis runs. If something changes, whether new data, a new analyst, or a new extract, the do-file catches it before the model does. These checks turn skepticism into a habit that runs automatically, even when no reviewer is in the room.

* checks that stop the code

```
. merge m:1 source_id  
> using id_crosswalk,  
> assert(match using)  
> keep(match) nogen  
  
. isid person_id  
> visit_dt dept
```

Checks That Stop the Code

Every time the unit of analysis changes, **isid** confirms what one row represents. Every merge uses **assert(match using)**, so unmatched records stop the do-file instead of disappearing. If a check fails, the analysis stops until someone explains why.

Checks That Explain the Code

We count people and use **assert** to compare the result with a number we trust, the 9,406 enrolled patients. **notes** records each decision inside the dataset, and **datasignature** confirms the file we modeled is the file we checked.

* checks that explain

```
. egen tagp = tag(person_id)  
. count if tagp  
. assert r(N) == 9406  
  
. notes: index visit  
> defines exposure  
. datasignature set
```

Same Data, Different Answer

We ran the same logistic model on the same data. The only thing that changed was the structure: one row per person instead of one row per workflow step, with every record accounted for. The strong effect disappeared.

- ✓ Follow-Up: 71.4% → 52.8%
- ✓ Odds Ratio: 1.84 → 1.12
- ✓ 95% CI: 0.97 to 1.29, Now Includes 1

FIRST PASS

REBUILT

N
48,212

9,406

Follow-up
71.4%

52.8%

Odds ratio
1.84

1.12

95% CI
1.71 to 1.98

0.97 to 1.29

```
. logit fu30 i.pilot age i.female, or
Logistic regression      Number of obs = 9,406
                          LR chi2(3)    = 14.82
                          Prob > chi2    = 0.0020
                          Pseudo R2     = 0.0011

-----+-----
fu30 | Odds ratio Std. err.      z    P>|z|    [95% conf. int.]
-----+-----
i.pilot | 1.120   .0814   1.36  0.179   .970   1.290
age | 1.004   .0015   2.66  0.008   1.001   1.007
i.female | 1.090   .0470   2.00  0.046   1.002   1.186
_cons | .910   .0750  -1.14  0.254   .774   1.070
```

Teaching Stata Through Failure

Step 1: Predict Before You Run

Before running any command, learners write down the N they expect and the direction of the effect. When the output disagrees, the surprise becomes the lesson.

The mistakes in this project were not embarrassing detours. They were the most valuable part of the analysis. So we now build them into how we teach. Instead of clean examples that work the first time, learners work with imperfect data and practice finding what is wrong before they trust what is right.

Step 2: Break It on Purpose

Learners receive a do-file that runs cleanly and answers the wrong question. Working in pairs, one writes the code and one asks the questions, and they find the hidden flaws together.

Step 3: Keep a U-Turn Log

Learners record every decision they reversed and why. We grade the reasoning, not just the final output, so changing course is rewarded instead of hidden.

```
. isid patient_id visit_dt dept  
variables patient_id visit_dt  
dept do not uniquely identify  
the observations  
r(459);
```

What We Want You to TAKEAWAY

Teach the Unit of Analysis First

Before any model, ask what one row represents. Every model answers a question about its rows, so make learners say it out loud.

Make Failure Visible and Safe

Plant errors, predict before running, and reward the learner who finds the flaw. Mistakes become method, not embarrassment.

Put Judgment Into the Code

Use `isid`, `assert`, and `notes` so the do-file stays skeptical even when no reviewer is in the room.

Questions and Resources

Thank you for joining us. Every failure in this talk came from a real pattern in applied work, and every fix is something you can use in your own classroom tomorrow. We would love to hear your questions, and your own stories of data that lied politely.

Explore Anysuma

Scan the code to explore the clinic, the simulated data, and the complete do-file. Everything is free to use in your own teaching.

SCAN HERE





Thank You



George Anyumba, DrPH



george.anyumba@baltimorecity.gov



Paris Johnson, DrPH



paris.johnson@baltimorecity.gov