SESUG 2025 Paper 46

Lookup Tables (LUT) in SAS® and Data-Driven Programming

Imelda C. Go, PhD, SC Department of Education, Columbia, SC
and Abbas S. Tavakoli, DrPH, MPH, ME, University of South Carolina

ABSTRACT

A lookup table (LUT) is a data structure that can help the programmer determine the outcome given the input. This paper illustrates the use of lookup tables to process data for an outcome instead of manually coded statements. The use of lookup tables has the potential to reduce processing time since computations are replaced by a lookup operation. Business rules and adequate documentation form the foundation of the lookup table. Embodying the processing logic in the lookup table provides a framework for data-driven programming, automation, and for the programmer to more efficiently complete programming tasks. There is the potential to reduce the amount of code because the lookup table already contains the information that would otherwise require hard coding. Using a lookup table can also simplify the code maintenance tasks and reduce human error when business rules need to change.

INTRODUCTION

SAS provides different table lookup tools including, 1-dimensional/multi-dimensional arrays, DATA step merges, SQL joins, formats, indexes, and hash objects. The examples here use the SQL join to illustrate data-driven programming that reduces hard coding and to create data structures that drive the processing. We can use lookup tables (LUT) to match input data with outcomes in lieu of using the more manual method of hard-coded conditional statements.

This may sound familiar to you at work regarding a recurring project. Someone says, "We already have code for this from last year. All we have to do is update it with this year's business rule changes. We still have to check with the client on a few things regarding the intent. If the rules didn't change, it would be easy. I took a look at the code and it's many, many lines of conditional statements. It's not going to be that easy to update it with the many changes this year. I'm afraid I might break the code when I start to update it."

In the perfect world, business rules would be written down, maintained, and documented and every rule would be defined clearly for the programmer. The written rules translate into data processing actions that the programmer must execute. Each business rule has a counterpart in programming code. It is ideal if the business rules are written in such a way that takes into consideration what the programmer must do. Often, clients may need to vet/approve the business rules and do not need to read the technical details that programmers need. Hence, there might be two types of documentation: one for the clients that provide a general overview and the second one with task-execution details for the programmers. If written a certain way, the LUT can be a bridge between the two types of documentation with columns that are understandable to a layperson and columns that are for the programmer.

Common business rule problems include:

- The business rules vary in detail provided. The programmers "fill in the blanks" to their best understanding and sometimes there is misinterpretation. Because programmers are not sure of what they need to do, delays will occur since they don't have the knowledge to finish the task.
- Some business rules are made/modified during meetings or after a lengthy e-mail interchange. These rules sometimes never make it into formal written business rules and remain in the SAS code.

It is not that easy when we have to update a program someone else wrote.

- When you have a set of complicated conditional statements (IF-THEN-ELSE, SELECT-WHEN, CASE-WHEN, etc.), it's not always easy to update it depending on the complexity of the code.
- Programming styles and techniques vary among programmers. How many programmers have been through the code? It can get complicated when multiple people touch the same code with different approaches.
    - Example: Some programmers will program for the general situation and then use statements at the end of the program to override the general cases to take care of the exceptions. This could lead to confusion because the reader takes this non-direct path to the get the desired outcomes. A more direct way is to use a lookup table.
- The code may have noise. The comments may not be updated and no longer reflect what is in the code. This can cause programmers to doubt what they are reading. Some people do not clean up their code and leave code that is not essential to the task.
- We change the code and think we did it correctly, but we break something in the process.

ADVANTAGES OF LOOKUP TABLES

- Using LUTs can reduce the amount of code because the inter-relationships between the input values and outcomes are already defined by each row in the LUT. Without the LUT framework, all those inter-relationships need to be programmed somehow.
- Once the LUT structure is defined, it is straightforward to populate it. Hard-coded lines are another story. When many processing rules are coded conditionally, there are several different ways a programmer can choose to do the task. If the code needs to change, it can be difficult and quite challenging to change the code because a change in one section might produce unintended consequences elsewhere. Therefore, using LUT is one way to reduce this type of difficulty.
- We can more clearly define the relationship between the input and the outcomes using as many rows in the LUT as we need. The LUT becomes a solid reference about what produces the desired outcomes.

EXAMPLES OF USING LOOKUP TABLES FOR DATA-DRIVEN PROGRAMMING

The three examples start with the same input data. The first example has a LUT with one record with the simplest information need. The second example has an LUT with 8 records, while the third one has 75 records. The LUT increased as the information needs changed.

The steps for each of the three examples follow.

1. Identify the SAS data variables that are part of the data processing rules. Create an LUT where all the combinations of interest for the input variables and the corresponding outcome(s) appear.
2. Create formats for each of the input variables according to the criteria specified in the LUT.
3. Recode the numeric variables using formats and PUT function into meaningful labels that correspond to the values/labels in the LUT so that we can join the input data with the LUT. This is because our sample LUT has character variables or labels in the cells. If you have numeric variables in your LUT, you will make simple adjustments to the sample code so that you can join your data set with the LUT and arrive at the outcome without manually hard-coding conditionals in the SAS program.
4. Using a SQL join, match the SAS data with the Lookup Table using recoded variables.

The following summarizes the differences and similarities in the SAS code used for the three examples as the LUT increased in size. The code is mostly the same even if the LUT expanded greatly by the third example.

SAS Code	Example 1	Example 2	Example 3
Part 1: PROC Format	Same code	Same code	Needed change because LUT entries changed
Part 2: By recoding, produce dataset variables that can be joined to the LUT	Same code	Same code	Same code
Part 3: Read the LUT in an Excel file	Same code LUT has 1 record.	Same code LUT has 8 records.	Same code LUT has 75 records.
Part 4: SQL Join	Same code	Same code	Same code

Our sample input data set follows, which we will call `dataset`. It has the three columns relevant to the LUT in the following examples.

Obs	grade	age	score
1	9	3	1199
2	8	14	1199
3	8	15	1199
4	10	14	1199
5	.	.	1200
6	3	15	1200
7	5	19	1200
8	10	18	1200
9	11	19	1200
10	10	3	1199
11	.	.	1199
12	3	15	1199
13	5	19	1199
14	10	3	.
15	11	4	.
16	.	.	.
17	3	15	.
18	5	19	.
19	10	15	1240

EXAMPLE #1: IDENTIFY ELIGIBLE STUDENTS ONLY

We need to identify eligible students for an educational program in the 2025-26 school year. Eligibility is based on:

1. Being in the 10th grade
2. Being 15 years of age as of September 1, 2025
3. Having scored at least 1200 on a standardized test from the prior school year

Let us suppose we have a data set where `grade`, `age`, and `score` are all numeric variables.

Criteria	Programming Criteria
Grade 10	`Grade=10`
Age 15 as of September 1, 2025	`Age=15`
Score of at least 1200 on a statewide assessment in the prior year	`Score>=1200`

Programmers can choose to create a variable `eligible`, that is 1 when the criteria are all met and 0 if not.

```
If grade=10 and age=15 and score>=1200 then eligible=1; else eligible=0;
```

Instead of using this programming approach to arrive at the eligibility determination, we will use an LUT with one record. You can put any label in the last 3 columns of Row 1 because we will be using formats to map the values `grade`, `age`, and `score` in `dataset` to the labels below. The sample labels tell us how to define the formats.

Row Number	Category	CategoryCode	Grade Level	Age as of September 1, 2025	Score on statewide assessment in the prior school year
1	Eligible students	EL	Grade=10	Age=15	Score>=1200

The code to obtain the eligibility status follows.

Part	Code	Description

1	`proc format;`
	`value grade`
	`  **10**='Grade=10'`
	`  other='Grade<>10';`
	`value age`
	`  **15**='Age=15'`
	`  other='Age<>15';`
	`value score`
	`  other='Score<1200'`
	`  **1200**-high='Score>=1200';`
	`quit;`
		Since `grade`, `age`, and `score` columns are numeric. We will define formats for each, such that the labels match the labels in the LUT.
		
		Note that missing values are taken care of by the OTHER= lines. Since a numeric missing value is always less than any number, OTHER= for score definitely includes the missing values.
		
		Whatever is defined in each PROC FORMAT statement is stored in a dataset which can be obtained by the CNTLOUT= option. One may save these data sets and use them to load into PROC FORMAT with the CNTLIN= option and thereby further reduce manual hard coding.

2	`data dataset;`
	`set dataset;`
	`grade2=put(grade,grade.);`
	`age2=put(age,age.);`
	`score2=put(score,score.);`
		We apply the formats using the PUT function to the corresponding columns and look at the values. The PUT function always results in a character value. We can eliminate this step and apply the PUT function in the SQL join. This step emphasizes that we need to join our input data to the LUT.

3	`proc import`
	` datafile="&path.\LUT.xlsx"`
	` out=lut  dbms=xlsx replace`
	` sheet="example";`
	`run;`
		The LUT is in an Excel workbook and is read by PROC IMPORT.

4	`proc sql;`
	`create table example as select`
	`a.*, b.* from dataset as a`
	`left join lut as b`
	`on`
	`catx('+',a.age2,a.grade2,a.score2)`
	`=catx('+',b.age,b.grade,b.score);`
	`quit;`
		Join the LUT with the records in dataset. `Grade2`, `age2`, and `score2` were defined so that their labels will match the labels used in the LUT.
		
		In lieu of using `age2`, `grade2`, and `score2`, we can use the PUT statement directly in the ON line:
		`on`
		`catx('+',put(a.age,age.),put(a.grade,`
		`grade.), put(a.score,score.))`
		`=catx('+',b.age,b.grade,b.score);`

Raw data with formatted values are shown below.

Obs	grade	grade2	age	age2	score	score2
1	9	Grade<>10	3	Age<>15	1199	Score<1200
2	8	Grade<>10	14	Age<>15	1199	Score<1200
3	8	Grade<>10	15	Age=15	1199	Score<1200
4	10	Grade=10	14	Age<>15	1199	Score<1200
5	.	Grade<>10	.	Age<>15	1200	Score>=1200
6	3	Grade<>10	15	Age=15	1200	Score>=1200
7	5	Grade<>10	19	Age<>15	1200	Score>=1200
8	10	Grade=10	18	Age<>15	1200	Score>=1200
9	11	Grade<>10	19	Age<>15	1200	Score>=1200
10	10	Grade=10	3	Age<>15	1199	Score<1200
11	.	Grade<>10	.	Age<>15	1199	Score<1200
12	3	Grade<>10	15	Age=15	1199	Score<1200
13	5	Grade<>10	19	Age<>15	1199	Score<1200
14	10	Grade=10	3	Age<>15	.	Score<1200
15	11	Grade<>10	4	Age<>15	.	Score<1200
16	.	Grade<>10	.	Age<>15	.	Score<1200
17	3	Grade<>10	15	Age=15	.	Score<1200
18	5	Grade<>10	19	Age<>15	.	Score<1200
19	10	Grade=10	15	Age=15	1240	Score>=1200

The LUT has a single record.

Obs	Category	CategoryCode	Grade	Age	Score
1	Eligible students	EL	Grade=10	Age=15	Score>=1200

We use a SQL JOIN on `dataset` and `lut`. Because we used a LEFT JOIN, `Category` and `CategoryCode` are blank whenever the `dataset` record had no matching record in `lut`. This helps us identify records where the values in `grade2`, `age2`, and `score2` are not defined in the `lut`. If we want to associate a category with these non-matching records, we should add records to the LUT for these combinations of values. We will expand the LUT in Example #2.

The joined data are shown below.

Obs	grade	age	score	grade2	age2	score2	Category	CategoryCode
1	8	14	1199	Grade<>10	Age<>15	Score<1200		
2	9	3	1199	Grade<>10	Age<>15	Score<1200		
3	5	19	.	Grade<>10	Age<>15	Score<1200		
4	5	19	1199	Grade<>10	Age<>15	Score<1200		
5	.	.	1199	Grade<>10	Age<>15	Score<1200		
6	11	4	.	Grade<>10	Age<>15	Score<1200		
7	.	.	.	Grade<>10	Age<>15	Score<1200		
8	5	19	1200	Grade<>10	Age<>15	Score>=1200		
9	11	19	1200	Grade<>10	Age<>15	Score>=1200		
10	.	.	1200	Grade<>10	Age<>15	Score>=1200		
11	10	3	.	Grade=10	Age<>15	Score<1200		
12	10	3	1199	Grade=10	Age<>15	Score<1200		
13	10	14	1199	Grade=10	Age<>15	Score<1200		
14	10	18	1200	Grade=10	Age<>15	Score>=1200		
15	8	15	1199	Grade<>10	Age=15	Score<1200		
16	3	15	1199	Grade<>10	Age=15	Score<1200		
17	3	15	.	Grade<>10	Age=15	Score<1200		
18	3	15	1200	Grade<>10	Age=15	Score>=1200		
19	10	15	1240	Grade=10	Age=15	Score>=1200	Eligible students	EL

In summary, we see only one eligible student.

The FREQ Procedure

grade2	age2	score2	Category	CategoryCode	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Grade<>10	Age<>15	Score<1200			7	36.84	7	36.84
Grade<>10	Age<>15	Score>=1200			3	15.79	10	52.63
Grade<>10	Age=15	Score<1200			3	15.79	13	68.42
Grade<>10	Age=15	Score>=1200			1	5.26	14	73.68
Grade=10	Age<>15	Score<1200			3	15.79	17	89.47
Grade=10	Age<>15	Score>=1200			1	5.26	18	94.74
Grade=10	Age=15	Score>=1200	Eligible students	EL	1	5.26	19	100.00

EXAMPLE #2: IDENTIFY ELIGIBLE STUDENTS AND WHY STUDENTS WERE INELIGIBLE

The first simple example was to illustrate how easy it is to join our data using recoding via PROC FORMAT even if there was just one record in the LUT.

The next request we receive is, since we have identified the eligible students, what were the reasons for ineligibility? Was it because of the grade level, the age, the score, or a combination of these?

To answer these questions, we can start writing SAS code to probe the reasons for ineligibility OR we simply expand the LUT and update PROC FORMAT if necessary. We add rows 2-8 in the following LUT to get more information on why students were not eligible. The updated LUT defines the relationship between grade level, age, score, eligibility, and reasons for ineligibility. Row number 1 defines an eligible student. The other rows define the ineligibility reason.

Row Number	Category	CategoryCode	Grade Level	Age as of September 1, 2025	Score on statewide assessment in the prior school year
1	Eligible students	EL	Grade=10	Age=15	Score>=1200
2	Ineligible due to score	IS	Grade=10	Age=15	Score<1200
3	Ineligible due to grade	IG	Grade<>10	Age=15	Score>=1200
4	Ineligible due to age	IA	Grade=10	Age<>15	Score>=1200
5	Ineligible due to grade, age	IGA	Grade<>10	Age<>15	Score>=1200
6	Ineligible due to grade, score	IGS	Grade<>10	Age=15	Score<1200
7	Ineligible due to age, score	IAS	Grade=10	Age<>15	Score<1200
8	Ineligible due to grade, age, score	IGAS	Grade<>10	Age<>15	Score<1200

Now that we've updated the LUT in the Excel file, we simply run the exact same code in EXAMPLE #1. No mess and no hassle with hard-coded statements. We just need to focus on making sure the LUT is correct and the corresponding formats are correct. The labels we used in the LUT are the same ones defined in PROC FORMAT in Example #1.

The updated LUT is shown below.

Obs	Category	CategoryCode	Grade	Age	Score
1	Eligible students	EL	Grade=10	Age=15	Score>=1200
2	Ineligible due to score	IS	Grade=10	Age=15	Score<1200
3	Ineligible due to grade	IG	Grade<>10	Age=15	Score>=1200
4	Ineligible due to age	IA	Grade=10	Age<>15	Score>=1200
5	Ineligible due to grade, age	IGA	Grade<>10	Age<>15	Score>=1200
6	Ineligible due to grade, age, score	IGAS	Grade<>10	Age<>15	Score<1200
7	Ineligible due to grade, score	IGS	Grade<>10	Age=15	Score<1200
8	Ineligible due to age, score	IAS	Grade=10	Age<>15	Score<1200

Raw data with formatted values are shown below.

Obs	grade	grade	age	age2	score	score2
1	9	HS	3	Bad Data	1199	Low Score
2	8	MS	14	Underaged	1199	Low Score
3	8	MS	15	Age=15	1199	Low Score
4	10	Grade=10	14	Underaged	1199	Low Score
5	.	No Grade	.	No Age	1200	Score>=1200
6	3	ES	15	Age=15	1200	Score>=1200
7	5	ES	19	Overaged	1200	Score>=1200
8	10	Grade=10	18	Overaged	1200	Score>=1200
9	11	HS	19	Overaged	1200	Score>=1200
10	10	Grade=10	3	Bad Data	1199	Low Score
11	.	No Grade	.	No Age	1199	Low Score
12	3	ES	15	Age=15	1199	Low Score
13	5	ES	19	Overaged	1199	Low Score
14	10	Grade=10	3	Bad Data	.	No Score
15	11	HS	4	Bad Data	.	No Score
16	.	No Grade	.	No Age	.	No Score
17	3	ES	15	Age=15	.	No Score
18	5	ES	19	Overaged	.	No Score
19	10	Grade=10	15	Age=15	1240	Score>=1200

We then join our raw data with the LUT on the appropriate columns to gain more insights about what caused ineligibility.

Obs	grade	age	Score	grade2	age2	score2	Category	Category Code
1	8	14	1199	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
2	9	3	1199	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
3	5	19	.	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
4	5	19	1199	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
5	.	.	1199	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
6	11	4	.	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
7	.	.	.	Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS
8	5	19	1200	Grade<>10	Age<>15	Score>=1200	Ineligible due to grade, age	IGA
9	11	19	1200	Grade<>10	Age<>15	Score>=1200	Ineligible due to grade, age	IGA
10	.	.	1200	Grade<>10	Age<>15	Score>=1200	Ineligible due to grade, age	IGA
11	10	3	.	Grade=10	Age<>15	Score<1200	Ineligible due to age, score	IAS
12	10	3	1199	Grade=10	Age<>15	Score<1200	Ineligible due to age, score	IAS
13	10	14	1199	Grade=10	Age<>15	Score<1200	Ineligible due to age, score	IAS
14	10	18	1200	Grade=10	Age<>15	Score>=1200	Ineligible due to age	IA
15	8	15	1199	Grade<>10	Age=15	Score<1200	Ineligible due to grade, score	IGS
16	3	15	1199	Grade<>10	Age=15	Score<1200	Ineligible due to grade, score	IGS
17	3	15	.	Grade<>10	Age=15	Score<1200	Ineligible due to grade, score	IGS
18	3	15	1200	Grade<>10	Age=15	Score>=1200	Ineligible due to grade	IG
19	10	15	1240	Grade=10	Age=15	Score>=1200	Eligible students	EL

In summary, we see more about what caused ineligibility. We also note that we do have missing data, but the LUT categories do not mention missing values.

The FREQ Procedure

grade2	age2	score2	Category	CategoryCode	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Grade<>10	Age<>15	Score<1200	Ineligible due to grade, age, score	IGAS	7	36.84	7	36.84
Grade<>10	Age<>15	Score>=1200	Ineligible due to grade, age	IGA	3	15.79	10	52.63
Grade<>10	Age=15	Score<1200	Ineligible due to grade, score	IGS	3	15.79	13	68.42
Grade<>10	Age=15	Score>=1200	Ineligible due to grade	IG	1	5.26	14	73.68
Grade=10	Age<>15	Score<1200	Ineligible due to age, score	IAS	3	15.79	17	89.47
Grade=10	Age<>15	Score>=1200	Ineligible due to age	IA	1	5.26	18	94.74
Grade=10	Age=15	Score>=1200	Eligible students	EL	1	5.26	19	100.00

This leads us to Example #3, where we mention missing values in the LUT. We will make the LUT even longer with 75 records and modify our PROC FORMAT statements to produce an even finer look at the data.

EXAMPLE #3: IDENTIFY ELIGIBLE STUDENTS AND DIGGING DEEPER INTO WHY STUDENTS WERE INELIGIBLE

Our next request is to provide a more detailed breakdown regarding ineligibility. We want to identify records with missing `grade`, `age`, or `score` values. We want to know more about the following:

1. Number of ineligible students who are in ES, MS, and HS
2. Number of ineligible students who do not have a grade level
3. Number of students with no age data
4. Number of students with a clearly bad age value
5. Number of students who are underaged
6. Number of students who are overaged
7. Number of students who do not have a score

To use the above framework for our code, we need to update the PROC FORMAT specifications and the LUT in the Excel file. The PROC FORMAT statements were updated as shown below and the LUT file now consists of 75 records (5X5X3=75 combinations).

PROC FORMAT		
Grade (5 labels)	**Age (5 labels)**	**Score (3 labels)**
Grade=10	No Age	No Score
No Grade	Bad Data	Low Score
ES	Underaged	Score>=1200
MS	Age=15	
HS	Overaged	

The LUT is shown below.

Obs	Category	CategoryCode	Grade	Age	Score
1	Ineligible due to grade	IG	ES	Age=15	Score>=1200
2	Ineligible due to grade, score	IGS	ES	Age=15	Low Score
3	Ineligible due to grade, score	IGS	ES	Age=15	No Score
4	Ineligible due to grade, age	IGA	ES	Overaged	Score>=1200
5	Ineligible due to grade, age, score	IGAS	ES	Overaged	Low Score
6	Ineligible due to grade, age, score	IGAS	ES	Overaged	No Score
7	Ineligible due to grade, age	IGA	ES	Underaged	Score>=1200
8	Ineligible due to grade, age, score	IGAS	ES	Underaged	Low Score
9	Ineligible due to grade, age, score	IGAS	ES	Underaged	No Score
10	Ineligible due to grade, age	IGA	ES	No Age	Score>=1200
11	Ineligible due to grade, age, score	IGAS	ES	No Age	Low Score
12	Ineligible due to grade, age, score	IGAS	ES	No Age	No Score
13	Ineligible due to grade, age	IGA	ES	Bad Data	Score>=1200
14	Ineligible due to grade, age, score	IGAS	ES	Bad Data	Low Score
15	Ineligible due to grade, age, score	IGAS	ES	Bad Data	No Score

Obs	Category	CategoryCode	Grade	Age	Score
16	Ineligible due to grade	IG	MS	Age=15	Score>=1200
17	Ineligible due to grade, score	IGS	MS	Age=15	Low Score
18	Ineligible due to grade, score	IGS	MS	Age=15	No Score
19	Ineligible due to grade, age	IGA	MS	Overaged	Score>=1200
20	Ineligible due to grade, age, score	IGAS	MS	Overaged	Low Score
21	Ineligible due to grade, age, score	IGAS	MS	Overaged	No Score
22	Ineligible due to grade, age	IGA	MS	Underaged	Score>=1200
23	Ineligible due to grade, age, score	IGAS	MS	Underaged	Low Score
24	Ineligible due to grade, age, score	IGAS	MS	Underaged	No Score
25	Ineligible due to grade, age	IGA	MS	No Age	Score>=1200
26	Ineligible due to grade, age, score	IGAS	MS	No Age	Low Score
27	Ineligible due to grade, age, score	IGAS	MS	No Age	No Score
28	Ineligible due to grade, age	IGA	MS	Bad Data	Score>=1200
29	Ineligible due to grade, age, score	IGAS	MS	Bad Data	Low Score
30	Ineligible due to grade, age, score	IGAS	MS	Bad Data	No Score
31	Ineligible due to grade	IG	HS	Age=15	Score>=1200
32	Ineligible due to grade, score	IGS	HS	Age=15	Low Score
33	Ineligible due to grade, score	IGS	HS	Age=15	No Score
34	Ineligible due to grade, age	IGA	HS	Overaged	Score>=1200
35	Ineligible due to grade, age, score	IGAS	HS	Overaged	Low Score
36	Ineligible due to grade, age, score	IGAS	HS	Overaged	No Score
37	Ineligible due to grade, age	IGA	HS	Underaged	Score>=1200
38	Ineligible due to grade, age, score	IGAS	HS	Underaged	Low Score
39	Ineligible due to grade, age, score	IGAS	HS	Underaged	No Score
40	Ineligible due to grade, age	IGA	HS	No Age	Score>=1200
41	Ineligible due to grade, age, score	IGAS	HS	No Age	Low Score
42	Ineligible due to grade, age, score	IGAS	HS	No Age	No Score
43	Ineligible due to grade, age	IGA	HS	Bad Data	Score>=1200
44	Ineligible due to grade, age, score	IGAS	HS	Bad Data	Low Score
45	Ineligible due to grade, age, score	IGAS	HS	Bad Data	No Score
46	Ineligible due to grade	IG	No Grade	Age=15	Score>=1200
47	Ineligible due to grade, score	IGS	No Grade	Age=15	Low Score
48	Ineligible due to grade, score	IGS	No Grade	Age=15	No Score
49	Ineligible due to grade, age	IGA	No Grade	Overaged	Score>=1200
50	Ineligible due to grade, age, score	IGAS	No Grade	Overaged	Low Score
51	Ineligible due to grade, age, score	IGAS	No Grade	Overaged	No Score
52	Ineligible due to grade, age	IGA	No Grade	Underaged	Score>=1200
53	Ineligible due to grade, age, score	IGAS	No Grade	Underaged	Low Score
54	Ineligible due to grade, age, score	IGAS	No Grade	Underaged	No Score
55	Ineligible due to grade, age	IGA	No Grade	No Age	Score>=1200

Obs	Category	CategoryCode	Grade	Age	Score
56	Ineligible due to grade, age, score	IGAS	No Grade	No Age	Low Score
57	Ineligible due to grade, age, score	IGAS	No Grade	No Age	No Score
58	Ineligible due to grade, age	IGA	No Grade	Bad Data	Score>=1200
59	Ineligible due to grade, age, score	IGAS	No Grade	Bad Data	Low Score
60	Ineligible due to grade, age, score	IGAS	No Grade	Bad Data	No Score
61	Eligible students	EL	Grade=10	Age=15	Score>=1200
62	Ineligible due to score	IS	Grade=10	Age=15	Low Score
63	Ineligible due to score	IS	Grade=10	Age=15	No Score
64	Ineligible due to age	IA	Grade=10	Overaged	Score>=1200
65	Ineligible due to grade, age, score	IGAS	Grade=10	Overaged	Low Score
66	Ineligible due to grade, age, score	IGAS	Grade=10	Overaged	No Score
67	Ineligible due to age	IA	Grade=10	Underaged	Score>=1200
68	Ineligible due to age, score	IAS	Grade=10	Underaged	Low Score
69	Ineligible due to age, score	IAS	Grade=10	Underaged	No Score
70	Ineligible due to age	IA	Grade=10	No Age	Score>=1200
71	Ineligible due to age, score	IAS	Grade=10	No Age	Low Score
72	Ineligible due to age, score	IAS	Grade=10	No Age	No Score
73	Ineligible due to age	IA	Grade=10	Bad Data	Score>=1200
74	Ineligible due to age, score	IAS	Grade=10	Bad Data	Low Score
75	Ineligible due to age, score	IAS	Grade=10	Bad Data	No Score

We run the following code, where we just need to update the labels so that we can join our data to the LUT and get the outcomes.

Part	Code	Description
1	proc format; value grade 10='Grade=10' .='No Grade' 3-5='ES' 6-8='MS' 9,11,12='HS'; value age .='No Age' 1-4='Bad Data' 5-14='Underaged' 15='Age=15' 16-high='Overaged'; value score .='No Score' low-1199='Low Score' 1200-high='Score>=1200'; quit;	The specifications were updated to match the labels in the new LUT.
2-4	Use the same code in Example #1.	

Now that we've updated the LUT in the Excel file and updated Part 1 (PROC FORMAT), parts 2-4 are exactly the same code as in the last two examples. Once again, no mess and no hassle with hard-coded statements even if our LUT has grown to 75 records. You can save your energy by turning the data into information instead of laboring over hard-coded statements. Sure, there's some work in the LUT, but if you do a thorough job, you know you have analyzed every possible combination of interest and it's based on business rules that are clearly defined.

When needed, it's important that the LUT covers every single possible combination of interest in the data so that the business/processing rules update is complete. If the LEFT JOIN produces category codes that are missing, then the LUT could not account for all possible combinations of values in the data. In Example #1, all the non-matches were for ineligible students and at the time, we only wanted to know about the eligible students.

Raw data with formatted values follow.

Obs	grade	grade2	age	age2	score	score2
1	9	HS	3	Bad Data	1199	Low Score
2	8	MS	14	Underaged	1199	Low Score
3	8	MS	15	Age=15	1199	Low Score
4	10	Grade=10	14	Underaged	1199	Low Score
5	.	No Grade	.	No Age	1200	Score>=1200
6	3	ES	15	Age=15	1200	Score>=1200
7	5	ES	19	Overaged	1200	Score>=1200
8	10	Grade=10	18	Overaged	1200	Score>=1200
9	11	HS	19	Overaged	1200	Score>=1200
10	10	Grade=10	3	Bad Data	1199	Low Score
11	.	No Grade	.	No Age	1199	Low Score
12	3	ES	15	Age=15	1199	Low Score
13	5	ES	19	Overaged	1199	Low Score
14	10	Grade=10	3	Bad Data	.	No Score
15	11	HS	4	Bad Data	.	No Score
16	.	No Grade	.	No Age	.	No Score
17	3	ES	15	Age=15	.	No Score
18	5	ES	19	Overaged	.	No Score
19	10	Grade=10	15	Age=15	1240	Score>=1200

Joined data follow.

Obs	grade	age	score	grade2	age2	score2	Category	Category Code
1	3	15	1199	ES	Age=15	Low Score	Ineligible due to grade, score	IGS
2	3	15	.	ES	Age=15	No Score	Ineligible due to grade, score	IGS
3	3	15	1200	ES	Age=15	Score>=1200	Ineligible due to grade	IG
4	10	15	1240	Grade=10	Age=15	Score>=1200	Eligible students	EL
5	8	15	1199	MS	Age=15	Low Score	Ineligible due to grade, score	IGS
6	10	3	1199	Grade=10	Bad Data	Low Score	Ineligible due to age, score	IAS
7	10	3	.	Grade=10	Bad Data	No Score	Ineligible due to age, score	IAS
8	9	3	1199	HS	Bad Data	Low Score	Ineligible due to grade, age, score	IGAS
9	11	4	.	HS	Bad Data	No Score	Ineligible due to grade, age, score	IGAS
10	.	.	1199	No Grade	No Age	Low Score	Ineligible due to grade, age, score	IGAS
11	.	.	.	No Grade	No Age	No Score	Ineligible due to grade, age, score	IGAS
12	.	.	1200	No Grade	No Age	Score>=1200	Ineligible due to grade, age	IGA
13	5	19	1199	ES	Overaged	Low Score	Ineligible due to grade, age, score	IGAS
14	5	19	.	ES	Overaged	No Score	Ineligible due to grade, age, score	IGAS
15	5	19	1200	ES	Overaged	Score>=1200	Ineligible due to grade, age	IGA
16	10	18	1200	Grade=10	Overaged	Score>=1200	Ineligible due to age	IA
17	11	19	1200	HS	Overaged	Score>=1200	Ineligible due to grade, age	IGA
18	10	14	1199	Grade=10	Underaged	Low Score	Ineligible due to age, score	IAS
19	8	14	1199	MS	Underaged	Low Score	Ineligible due to grade, age, score	IGAS

Our summary provides us with much more information than the first two examples.

The FREQ Procedure

grade2	age2	score2	Category	CategoryCode	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ES	Age=15	Low Score	Ineligible due to grade, score	IGS	1	5.26	1	5.26
ES	Age=15	No Score	Ineligible due to grade, score	IGS	1	5.26	2	10.53
ES	Age=15	Score>=1200	Ineligible due to grade	IG	1	5.26	3	15.79
ES	Overaged	Low Score	Ineligible due to grade, age, score	IGAS	1	5.26	4	21.05
ES	Overaged	No Score	Ineligible due to grade, age, score	IGAS	1	5.26	5	26.32
ES	Overaged	Score>=1200	Ineligible due to grade, age	IGA	1	5.26	6	31.58
Grade=10	Age=15	Score>=1200	Eligible students	EL	1	5.26	7	36.84
Grade=10	Bad Data	Low Score	Ineligible due to age, score	IAS	1	5.26	8	42.11
Grade=10	Bad Data	No Score	Ineligible due to age, score	IAS	1	5.26	9	47.37
Grade=10	Overaged	Score>=1200	Ineligible due to age	IA	1	5.26	10	52.63
Grade=10	Underaged	Low Score	Ineligible due to age, score	IAS	1	5.26	11	57.89
HS	Bad Data	Low Score	Ineligible due to grade, age, score	IGAS	1	5.26	12	63.16
HS	Bad Data	No Score	Ineligible due to grade, age, score	IGAS	1	5.26	13	68.42
HS	Overaged	Score>=1200	Ineligible due to grade, age	IGA	1	5.26	14	73.68
MS	Age=15	Low Score	Ineligible due to grade, score	IGS	1	5.26	15	78.95
MS	Underaged	Low Score	Ineligible due to grade, age, score	IGAS	1	5.26	16	84.21
No Grade	No Age	Low Score	Ineligible due to grade, age, score	IGAS	1	5.26	17	89.47
No Grade	No Age	No Score	Ineligible due to grade, age, score	IGAS	1	5.26	18	94.74
No Grade	No Age	Score>=1200	Ineligible due to grade, age	IGA	1	5.26	19	100.00

Although we added even greater detail into the LUT, the cost of coding was relatively low in that the SQL join took care of matching our data with the right outcomes. Just think of how long the code would have to be if there was no lookup table and instead, consisted of conditional statements. It's easier to read the entries in the LUT than freestyle hard-coding that could take many different forms.

CONCLUSION

This paper is another example of data-driven programming where the goal is to get SAS to do the work with less hassle and hard-coding. Lookup tables are useful tools for automating work. They have the potential to reduce runtime, to simplify the coding, and to reduce code maintenance costs. If the input data, processing rules, and outcomes can be represented by a a lookup table, using lookup tables offers solid advantages.

CONTACT INFORMATION

Imelda C. Go
SC Department of Education
icgo@ed.sc.gov

Abbas S. Tavakoli
University of South Carolina
abbas.tavakoli@sc.edu

TRADEMARK NOTICE