SAS4611-2020

Teaching SAS® Coding with Self-Checking Exercises

Jared Dean, SAS Institute Inc.

ABSTRACT

Programming is an essential skill for many people and for most STEM jobs. Teaching students to code is a standard part of statistics and computer science curriculums. In my opinion, students need individual hands-on practice to master the skill of writing code regardless of the programming language. Classroom lectures help explain concepts and design patterns, but they cannot substitute for students spending time practicing. Designing meaningful practice with timely feedback is difficult for faculty. Manual grading is either non-specific or has a significant lag between assignment completion and student feedback. In this paper, I outline how professors can teach SAS® using SAS® Analytics Cloud with self-grading assignments to help the students get immediate feedback on programming exercises. This case study leverages SAS Analytics Cloud and Jupyter Notebooks along with the SAS kernel and nbgrader.

INTRODUCTION

Programming is an essential skill for any professional in the analytics space. In the twenty years since I was first introduced to the SAS language, I strongly believe that the best way to master the skill of coding is to practice. In a classroom setting, practice is provided through homework. In this paper, I demonstrate how using SAS with cloud technologies and open-source tools professors can create self-checking homework exercises that provide students immediate feedback, which increases the speed of learning and confidence.

This paper has two main sections: an explanation of the tools used in creating the self-checking homework experience and a step-by-step guide of the process. There is also an appendix with SAS code used to compare student work to professors' solutions.

TOOLS

JUPYTER

On the Jupyter homepage, it states the mission of the tool: "Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages." In this context, Jupyter functions as a web-based interface for SAS programming where students can complete assignments and get immediate feedback on their ability to complete a programming task.

The only reason to have students use Jupyter as the programming interface instead of SAS Studio is because of the functionality that nbgrader provides.

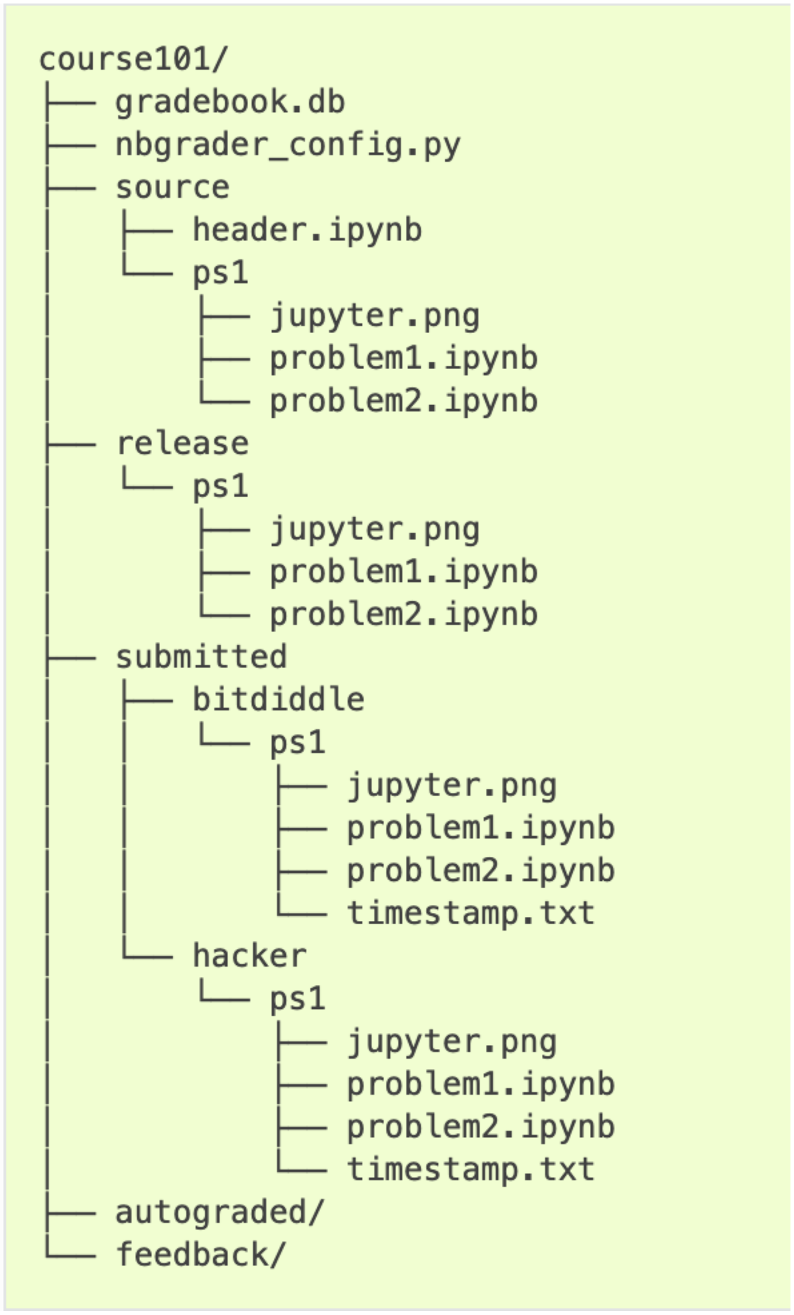
A Jupyter notebook is comprised of markdown cells and coding cells. Markdown cells are where the author can write prose. This can be documentation, instructions, links to videos, pictures, and so on. Coding cells contain code. For teaching SAS programming, the notebook uses the SAS Kernel. (Kernel is a Jupyter term that means the compute engine. Basically, it is where the code will be submitted for execution).

NBGRADER

Nbgrader is a Jupyter extension designed for classroom assignments. The extension is only supported on Jupyter Notebook not the more modern Jupyter Lab so you, as a professor, must use Jupyter notebooks. Your students can complete assignments using Jupyter Lab. The default language for nbgrader is Python, so there are changes that must to be made in the configuration file to accommodate the syntax differences between Python and SAS. These changes are made in the nbgrader_config.py file.

```
## The code snippet that will replace code solutions
c.ClearSolutions.code_stub = {'python': '# YOUR CODE HERE\nraise
NotImplementedError()',
                              'sas':'/*** YOUR CODE HERE ***/\n%putlog ERROR: No Code
Written;'}
```

Nbgrader expects the directory structure, shown in Figure 1: Nbgrader Directory Structure, to take homework assignments from source to feedback. To work within this structure, all your assignment notebooks should be within the “source” directory. I divide my assignments by week, so I have week1, week2, … weekN as subdirectories under source. Within the week1 folder, I will have the notebooks that will be assigned for that week.



```
course101/
├── gradebook.db
├── nbgrader_config.py
├── source
│   ├── header.ipynb
│   └── ps1
│       ├── jupyter.png
│       ├── problem1.ipynb
│       └── problem2.ipynb
├── release
│   └── ps1
│       ├── jupyter.png
│       ├── problem1.ipynb
│       └── problem2.ipynb
├── submitted
│   ├── bitdiddle
│   │   └── ps1
│   │       ├── jupyter.png
│   │       ├── problem1.ipynb
│   │       ├── problem2.ipynb
│   │       └── timestamp.txt
│   └── hacker
│       └── ps1
│           ├── jupyter.png
│           ├── problem1.ipynb
│           ├── problem2.ipynb
│           └── timestamp.txt
├── autograded/
└── feedback/
```

Figure 1: Nbgrader Directory Structure

There are initial setup steps that you must complete before you can add the students in your section. Adding students can be done through the command line interface using the following code:

```
nbgrader db student add jld90 --last-name=dean --first-name=jared --email=jared.dean@sas.com
```

Or you can add students by using the Manage Student link, shown in Figure 3. Assignment Dashboard. The other folders for release, submitted, autograded, and feedback are created automatically as the assignment progresses, and student sub folders are created automatically based on the students you registered for the class.

SAS KERNEL

The SAS Kernel for Jupyter is an open-source project (Apache 2 license) that Tom Weber and I maintain. It allows a SAS programmer to use Jupyter notebooks for their programming activities with any version of SAS 9.4 or later releases (including SAS® Viya®). It does not replace an existing SAS interface but is an addition to SAS® Studio, SAS® Enterprise Guide, and so on. The SAS Kernel source code is available on GitHub, and it can be installed using pip or conda. After the SAS Kernel is installed, you must modify a configuration file to identify how you connect to SAS. See the "Installing and Configuring Jupyter" link in the References section for more details on installation and configuration

SAS ANALYTICS CLOUD

I use SAS Analytics Cloud as a web-based programming platform. It leverages Kubernetes and Docker to provide compute resources. As a professor, the backend is not something I want to think about. SAS Analytics Cloud allows my students, and only my students, access to computing resources from any browser with internet connectivity. Each student has a home directory, and the section has a shared directory. I can distribute lecture materials and homework assignments by simply copying the files from my home directory (which is private to the students) to the shared directory. I have changed the file permissions on a few directories so that I'm the only person with Write access. This prevents the students from accidentally deleting course data or other materials relevant to the entire class.

The shared directory allows students to share their work outside of class and to seek help or ideas from the professor or fellow classmates.

PROCESS

This section walks through the lifecycle of an assignment from the creation by the professor to the final feedback sent to students after grading.

CREATING A HOMEWORK ASSIGNMENT

Creating a homework assignment starts with creating a notebook. I use the markdown cells to provide the question or coding task and then I put the solution in the subsequent coding cell. As an example, here is a question I might ask:

Q1 CREATE A TEMPORARY DATA SET

- Create a data set named phs703 that will be deleted at the end of your session.
- The data set should have one variable name semester.
- The value of semester should be fall2019.

In the subsequent coding cell, I would write the solution:

```
/*** BEGIN SOLUTION ***/
data phs703;
    semester = 'fall2019';
run;
/*** END SOLUTION ***/
```

The begin solution and end solution comments are key for the students when the notebook is processed by nbgrader. At run time, these comments are replaced with the following code:

```
/*** YOUR CODE HERE ***/
%putlog ERROR: No Code Written;
```

This code, if not modified, will produce an error in the log. This code helps students know that they have completed all of the assigned tasks. You can customize the replacement string in the `nbgrader_config.py` file.

In the cell below, the solution are the tests. In this case, I utilize SAS macros and the COMPARE procedure to validate that the student's outputs match mine. Except for the PRINT procedure, there is very little that cannot be validated this way by using ODS OUTPUT or output data sets. The advantages of this validation are that students get immediate feedback and that they are not limited to a specific syntax. For example, think of how many different WHERE clauses could be used to filter data each yielding the same result.

Below is an example of a solution cell:

```
/* Cell for testing your solution */
/*** BEGIN HIDDEN TESTS ***/
%MACRO hwae_q1 / store;
data answer;
    semester = 'fall2019';
run;
%mend hwae_q1;
/*** END HIDDEN TESTS ***/
/* Macro to generate the solution */
%hwae_q1
/* macro to compare your data set to the solution */
%compare(phs703);
```

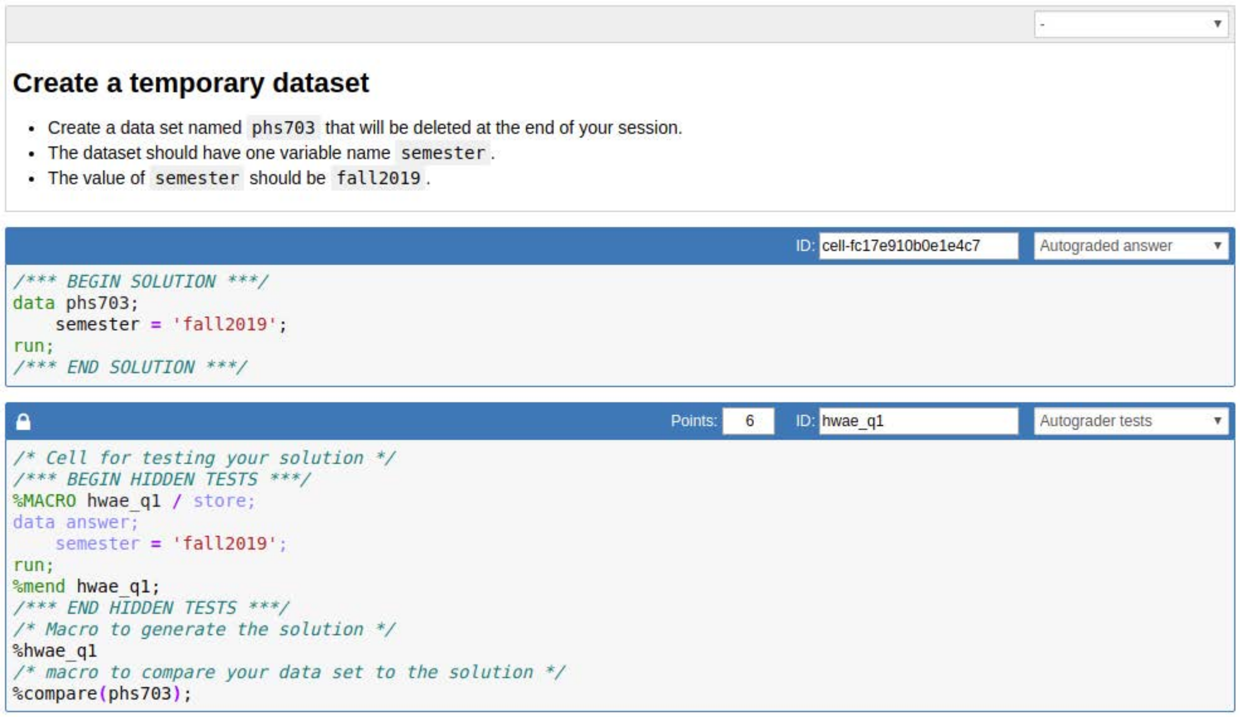
When the cell is processed by nbgrader for release, the students see the following code:

```
/* Cell for testing your solution */
/* Macro to generate the solution */
%hwae_q1
/* macro to compare your data set to the solution */
%compare(phs703);
```

The compare macro (which is included in the appendix of this paper) compares the students data set to the solution data set. In the log, the student sees either a SAS note that no differences were found or a warning pointing to the type of differences detected. This mechanism leverages the return codes from PROC COMPARE.

To prevent students from looking at the source code of the solutions, the macros are stored securely in the macro catalog. One of the major advantages of SAS Analytics Cloud is that I can update the macro catalog in place with each new homework assignment with no coordination or effort on the student's part.

The homework requirements are shown in context in Figure 2. Question Example. Note that assignment type (-, autograded answer, and autograded test) is very important to creating the release version for students. The most common mistake, through copy and paste, is adding the begin solution and end solution comment blocks within the autograded tests cell.



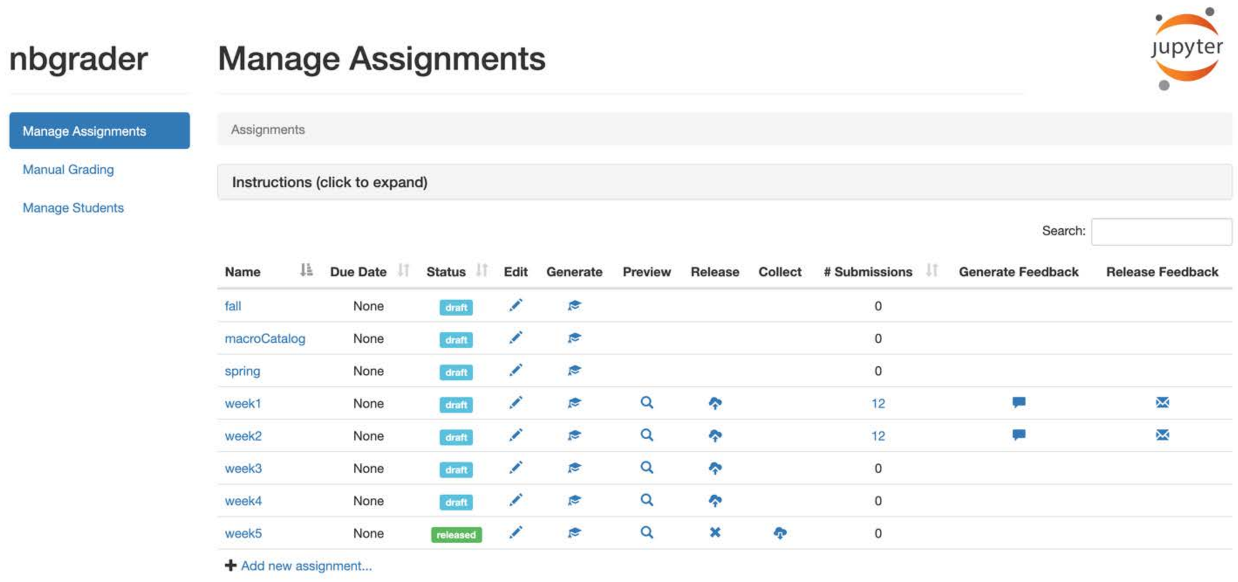
Create a temporary dataset

- Create a data set named `phs703` that will be deleted at the end of your session.
- The dataset should have one variable name `semester`.
- The value of `semester` should be `fall2019`.

```
/*** BEGIN SOLUTION ***/
data phs703;
    semester = 'fall2019';
run;
/*** END SOLUTION ***/
```

```
/* Cell for testing your solution */
/*** BEGIN HIDDEN TESTS ***/
%MACRO hwae_q1 / store;
data answer;
    semester = 'fall2019';
run;
%mend hwae_q1;
/*** END HIDDEN TESTS ***/
/* Macro to generate the solution */
%hwae_q1
/* macro to compare your data set to the solution */
%compare(phs703);
```

Figure 2. Question Example



Name	Due Date	Status	Edit	Generate	Preview	Release	Collect	# Submissions	Generate Feedback	Release Feedback
fall	None	draft	✎	🎓				0		
macroCatalog	None	draft	✎	🎓				0		
spring	None	draft	✎	🎓				0		
week1	None	draft	✎	🎓	🔍	☁		12	💬	✉
week2	None	draft	✎	🎓	🔍	☁		12	💬	✉
week3	None	draft	✎	🎓	🔍	☁		0		
week4	None	draft	✎	🎓	🔍	☁		0		
week5	None	released	✎	🎓	🔍	✖	☁	0		

Figure 3. Assignment Dashboard

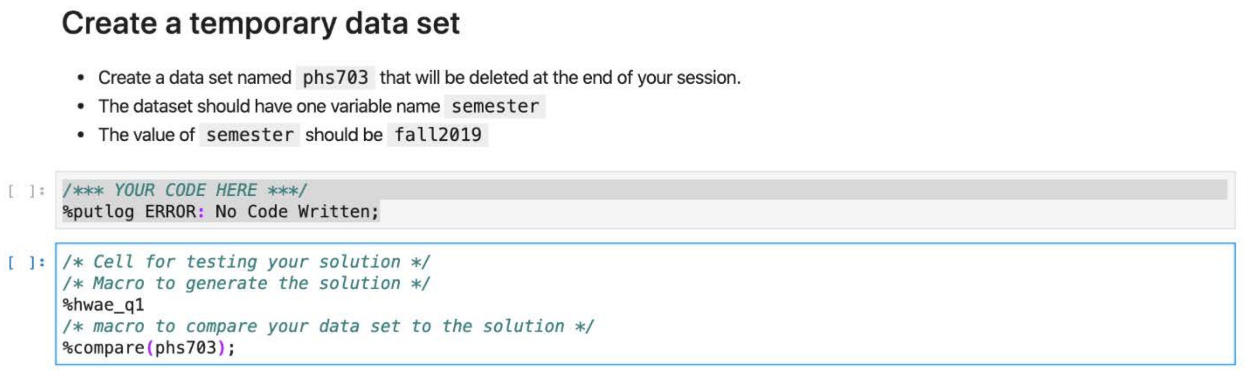
After you have completed making the assignment notebook, you will want to generate the student version. As shown in Figure 3. Assignment Dashboard, select the icon in the Generate column for the appropriate assignment. This will validate the notebook for correct use of the start and stop tags as well as create a student version in the release folder. The version in the source folder can be thought of as the key, and the version in the release folder is the blank worksheet.

I recommend reviewing the preview before sending the version to students. The best scenario is to have a teaching assistant or faculty member review the assignment prior to release. I have found several situations where students interpreted my instructions in a way I didn't intent.

RELEASE HOMEWORK TO STUDENTS

If you are using JupyterHub, there are mechanisms to distribute the assignments directly to students. That is not my current situation, so I manually upload the notebook to the course learning management system (LMS).

Students receive a notebook for completing their homework assignments. They can check each cell as many times as needed to get the correct answer, and they receive immediate feedback if they successfully completed the task. Figure 4 Released Student Question shows what the student would see when asked to complete this assignment.



Create a temporary data set

- Create a data set named `phs703` that will be deleted at the end of your session.
- The dataset should have one variable name `semester`
- The value of `semester` should be `fall2019`

```
[ ]: /*** YOUR CODE HERE ***/
     %putlog ERROR: No Code Written;
```

```
[ ]: /* Cell for testing your solution */
     /* Macro to generate the solution */
     %hwae_q1
     /* macro to compare your data set to the solution */
     %compare(phs703);
```

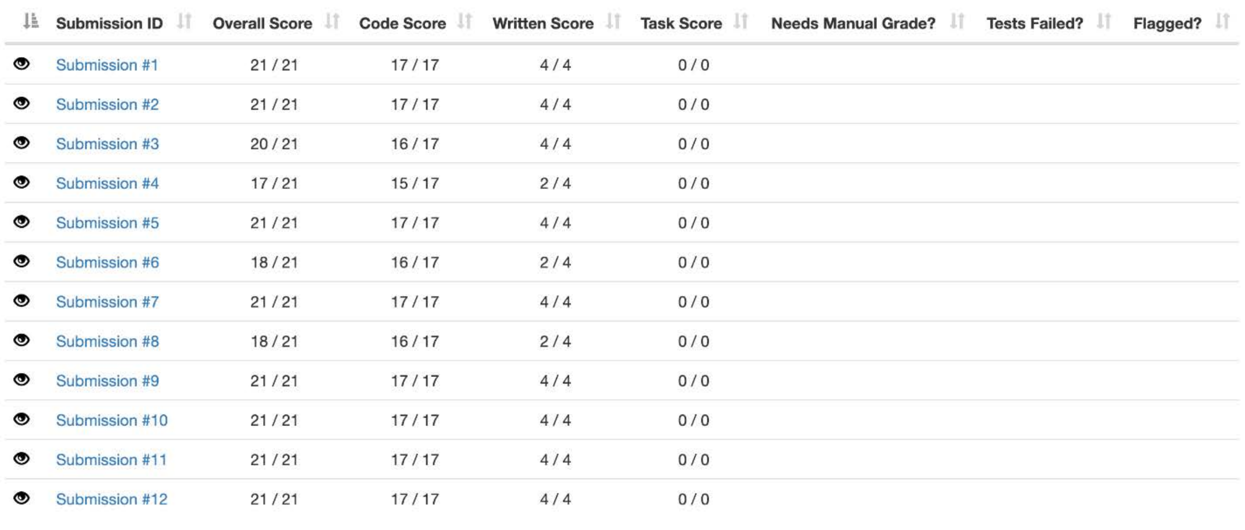
Figure 4 Released Student Question

COLLECT HOMEWORK

To collect the homework, I download a ZIP file from the university LMS. Then I have to unzip and place each student’s notebook in their folder to conform to the pattern that nbgrader expects. To do this, I wrote a Python notebook to automate the task.

GRADE HOMEWORK

Grading homework is a two-step process. The first step is to autograde the cells. When autograding the cells, the test cells are copied from the source notebook to the student’s submission. This copying prevents a student from altering the test cell as a way to improve their grades.



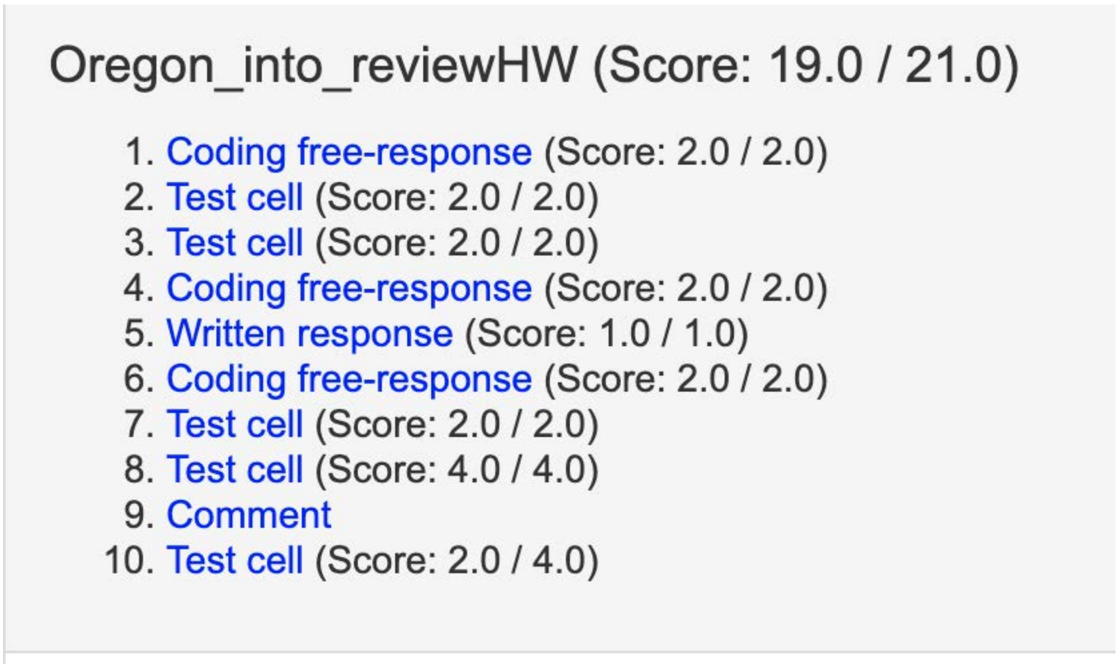
	Submission ID	Overall Score	Code Score	Written Score	Task Score	Needs Manual Grade?	Tests Failed?	Flagged?
	Submission #1	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #2	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #3	20 / 21	16 / 17	4 / 4	0 / 0			
	Submission #4	17 / 21	15 / 17	2 / 4	0 / 0			
	Submission #5	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #6	18 / 21	16 / 17	2 / 4	0 / 0			
	Submission #7	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #8	18 / 21	16 / 17	2 / 4	0 / 0			
	Submission #9	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #10	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #11	21 / 21	17 / 17	4 / 4	0 / 0			
	Submission #12	21 / 21	17 / 17	4 / 4	0 / 0			

Figure 5. Manual Grading Display

After the autograding is complete, manually grade the questions that couldn't be autograded. You can also review the student's code and award partial or full credit. There are shortcut keys that allow for efficient grading of the students. One other feature about nbgrader that I like is that student names are masked and randomized by default. In Figure 5. Manual Grading Display, you can see masked submissions and the breakdown of scores from automated and manual graded questions. Feedback can be provided for any question.

SHARE FEEDBACK

After the assignments have been graded, an HTML file that shows the student's code, the solution code, and any comments by the professor from grading is produced for each student. These HTML files can then be emailed to the student or uploaded to the LMS for the students to review. Figure 6. Feedback Table of Contents shows what the feedback HTML file contains. The hyperlinks in the document allow the student to quickly jump to the section they want to review.



Oregon_into_reviewHW (Score: 19.0 / 21.0)

1. Coding free-response (Score: 2.0 / 2.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 2.0 / 2.0)
4. Coding free-response (Score: 2.0 / 2.0)
5. Written response (Score: 1.0 / 1.0)
6. Coding free-response (Score: 2.0 / 2.0)
7. Test cell (Score: 2.0 / 2.0)
8. Test cell (Score: 4.0 / 4.0)
9. Comment
10. Test cell (Score: 2.0 / 4.0)

Figure 6. Feedback Table of Contents

APPENDIX

```
%MACRO compare(submission, solution=answer) /  store
         des="compare solution to student work";
   options nonotes;

   /* do the data sets exist */


   %if NOT (%sysfunc(exist(&submission))) %then
         %do;
               %put ERROR: &submission. dataset not found;
               %abort;
         %end;

   %if NOT (%sysfunc(exist(&solution))) %then
         %do;
```

```
                %put ERROR: &solution. dataset not found;
                %abort;
          %end;

    proc contents data=&submission out=__sublist (keep=name) noprint;
    run;

    proc sql noprint;
          %let sublist=;
          select distinct name into :sublist separated by ' ' from
__sublist;
    quit;

    proc sort data=WORK.&submission.;
          by &sublist.;
    run;

    proc contents data=&submission out=__sollist (keep=name) noprint;
    run;

    proc sql noprint;
          %let sollist=;
          select distinct name into :sollist separated by ' ' from
__sollist;
    quit;

    proc sort data=WORK.&solution.;
          by &sollist.;
    run;

    proc compare base=&solution. compare=&submission. criterion=0.00001
noprint;
    run;

    %let rc=&sysinfo;
    %put &rc=;

    /*     %put Proc Compare Return Code: &rc.; */
        data _null_;
         length message $ 600 decoded $ 600 text $60;
          array msg {17} $ 60 _temporary_ (
                 " ",
                "Data set labels differ",
                "Data set types differ",
                "Variable has different informat",
                "Variable has different format",
                "Variable has different length",
                "Variable has different label",
                "SOLUTION data set has observation(s) not in STUDENT",
                 "STUDENT data set has observation(s) not in SOLUTION",
                 "SOLUTION data set has BY group(s) not in STUDENT",
                 "STUDENT data set has BY group(s) not in SOLUTION",
                "SOLUTION data set has variable(s) not in STUDENT",
                "STUDENT data set has variable(s) not in SOLUTION",
                "At least one value comparison was unequal",
                "Conflicting variable types",
                 "BY variable(s) do not match",
                "Fatal error: comparison not done"
```

```
                    );

            testcode=&rc.;
            if testcode=0 then decoded="NO DIFFERENCE BETWEEN SOLUTION &
  STUDENT"; *<---(&SYSINFO=0 WHEN NO DIFFERENCE DETECTED);
            else do;
                decoded=" "; *<----------(VARIABLE 'DECODED' WILL STORE
  &SYSINFO MESSAGES);
                do k=1 to 16; *<----------(BECAUSE THERE ARE 16 POSSIBLE
  &SYSINFO MESSAGES);
                    binval=2**(k-1); *<--(CONVERT 0,1,2,3...16 TO BINARY
  1,2,4,8,16,32,64..);
                    match=band(binval, testcode); *<--(DO BITWISE TESTING WITH
  BAND FUNCTION);
                    key=sign(match)*k; *<---(REVERT BINARIES TO REGULAR
  NUMBERS FOR ARRAY- INDEXING);
                    text=msg(key+1); *<---(GET DECODED MESSAGE TEXT FROM ARRAY
  VALUE);
                    decoded=catx(", ",decoded,text); *<---(CONCATENATE IF MORE
  THAN ONE &SYSINFO MESSAGE);
                end;
            end;
            call symputx("message",decoded);
        run;
  /*
     proc delete data=&solution.;
     run;
  */
     options notes;

     %if &rc. eq 0 %then
           %put NOTE: &message.;
     %else
           %put WARNING: &message.;
  %mend compare;
```

CONCLUSION

This paper has introduced you to how student homework assignments could be made using a combination of SAS with open-source software. Creating homework assignments that are code based and self-grading gives all students confidence that they are learning and mastering the material.

Creating the assignments does take more time but the reusability and immediate feedback students receive as they work through the assignments is a worthwhile trade off.

REFERENCES

Jupyter. January 28, 2020. "Installing the Jupyter Software." Available at https://jupyter.org/install

Jupyter. "nbgrader documentation." Available at https://nbgrader.readthedocs.io/en/stable/index.html. Accessed on February 17, 2020.

SAS Institute Inc. July 21, 2017. "SAS Kernel Documentation." Available at https://sassoftware.github.io/sas_kernel/
.

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® For Dummies®*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jared Dean
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27518
919-677-8000
jared.dean@sas.com
@jaredldean