Paper CTE037-2025

Becoming Trilingual: A Story of a SAS Programmer Learning R and Python

Aaron Brown, South Carolina Department of Education

ABSTRACT

This paper serves primarily as an introduction and encouragement to SAS programmers who are interested in learning another language. Using the free and open-source R and Python languages as examples, we discuss learning these languages and give an example of how to write the same (relatively simple) program in SAS, R, and Python. We also discuss some of the more powerful tools to leverage multiple languages, although a thorough discussion is beyond the scope of this paper.

INTRODUCTION

Years ago, I was frustrated when I saw several papers and presentations dealing with topics other than SAS at a SAS conference. However, I now realize how understanding multiple languages can be beneficial.

That realization is what has motivated me to write this paper. Its main purpose is to decrease how intimidating it can be to learn a new language by showing how to do something in SAS, then show how to do it in R and Python. Different people learn things differently. For me, I learn programming best when I need to do something for a particular task. Thus, for this paper, I took one of my SAS projects from work and replicated it in R and Python.

My hope is that, through this example, other SAS programmers will see similarities between the languages that will help them transition into being a programming polyglot.

BUT WHY LEARN ANOTHER LANGUAGE?

But why should one be a polyglot if you can already do what you need in SAS? That is a legitimate question, and that attitude is part of why I was originally reluctant to pursue a topic like this. One very pragmatic reason is that it increases job options, as some companies favor SAS and some favor other languages. Another reason is that it gives one more tools to do a job; and, as I'll mention near the end of this paper, there are tools to integrate SAS, R, Python, and others together into one "toolkit". But a final reason is that it can be fun. If you truly enjoy the intricacies and puzzles of programming syntax, learning a new language can be an engaging activity.

OUR EXAMPLE

My example for this paper is ED*Facts* file 170, which is a listing of school districts and whether or not they get a certain type of grant. ED*Facts* is the program by which the state departments of education send data to the federal government, and this is one relatively simple data file they require. The one formatting abnormality is that the header row contains metadata instead of the column names. Data is sent as CSV files.

The program is broken down into five parts. Appendices B, C, and D contain the full code in SAS, R, and Python, respectively. Any screenshots come from SAS Enterprise Guide, RGui, and Spyder, respectively.

- Part One is hard-coding the current year and then using that year to generate some metadata. It also sets up the output file directory.
- Part Two is reading our raw data in from a CSV file, then formatting and sorting the data. See Appendix A for the raw data. Note that it is currently not sorted by District Code. For SAS and R, the District Code was read in as numeric instead of character, so this step includes fixing that.
- Part Three is generating frequency tables to check that the formatting in Part Two was done correctly.
- Part Four is generating some more metadata and creating the header row.
- Part Five is generating the new CSV file.

PART ONE: CREATE VARIABLES

The SAS code utilizes macro variables to create the different elements. The year is hard-coded as *yr* an other macro variables are derived from that value. In our example, this is data for school year 2018-2019 so we start with a hard-coded 1819.

```
%let yr=1819;
%let myyear=20%substr(&yr,1,2)-20%substr(&yr,3,2);
%let today=%sysfunc(today());
%let filename=SCLEASUBGRANTS170%substr(&myyear,8,2)SA.csv;
%let dir=O:\SESUG 2025;
```

To create the same variables in R, we can use:

```
yr="1819"
myyear=paste("20",substr(yr,1,2),"-20",substr(yr,3,4),sep="")
today=as.character(Sys.Date()) #yyyy-mm-dd format
today=paste(substr(today,6,7),substr(today,9,10),substr(today,3,4),sep="")  #convert to mmddyy format
filename=paste("SCLEASUBGRANTS170",substr(myyear,8,10),"RR.csv",sep="")
dir="O:\\SESUG 2025"
```

The *yr* variable was created approximately the same way. In R, you combine character strings via the paste function, using the sep (separator) notation to state that there is no space between the different character strings. Also, while R uses a substr function that is very similar to SAS, its third argument is the *ending position* instead of the *length of the string*.

Today's date can be gleamed from the Sys.Date function, although some work was needed to render the Date-type object to character then to reformat it into the desired format.

filename and *dir* follow the same logic as *yr* and *myyear*, but note that R requires a double-slash instead of a single slash. The slash is an escape character and R requires two to read it as a "real" slash.

To create the same variables in Python, we can use:

```
yr="1819"
myyear="20"+yr[0:2]+"-20"+yr[2:4]
today=date.today()
today=today.strftime("%m%d%y")
filename="SCLEASUBGRANTS170"+myyear[7:9]+"PY.csv"
filepath="O:\SESUG 2025"
```

Most of this follows similar syntax to SAS or R. There is different work necessary to render *today* as desired.

Now is a good moment to mention R packages and Python modules. In R, you sometimes need a bundle of functions that are not in the base R installation. You can download these Packages from one of several R servers. For Python, there are different modules you may need for different purposes. If you are using base Python, you may need to install these on your computer first. If you are using a particular user interface, like Anaconda, they might already be installed. For our project today, we do not need any R packages but we do need two Python modules. We can activate them via these lines of code:

```
from datetime import date
import pandas as pd
```

The first gives us access to a function we need to get today's date. The second is the general statistical software module in Python, and it includes what we will use for reading and writing CSVs as well as generating frequency tables. It also lets us create Python data frames, an object somewhat similar to a SAS dataset and very similar to an R data frame.

Another detail to remember is how the languages handle case-sensitivity and whitespace. In SAS, a variable name, dataset name, or function name means the same thing whether it is uppercase or lowercase. R and Python are different: case matters. SAS also does not "care about" whitespace: you could write an entire program all on one line of a text editor if you wanted, and it would still run. R cares about whitespace to a degree, and Python includes whitespace (especially indentation) as an integral part of its syntax. Be aware of when it matters.

PART TWO: CREATE A DATASET/DATAFRAME AND FORMAT DATA

The SAS code utilizes PROC IMPORT and the DATA step. We could read the data in via a DATA step, but the PROC IMPORT code is simpler. However, since PROC IMPORT does not let us control how data is read into SAS, we have an issue in that the District Code is read in as numeric and thus some leading zeroes are dropped. The DATA step code adds those back in. We also use the DATA step to create a new variable, called Grant, from the GrantStatus that is on our raw data.

```
proc import datafile="&dir\N170 data.csv" dbms=csv replace out=mydata;
      getnames=yes;
run;
data mydata;
      set mydata(RENAME=DistrictCode=DistrictCodeNumber);
      length DistrictCode $4. grant $20.;
      DistrictCode=put(DistrictCodeNumber,z4.);
      grant=upcase(compress('MVSUBG'||GrantStatus));
run;
proc sort data=mydata; BY DistrictCode; run;
```

To do the same in R, we use the code below.

```
mydata=read.csv(file=paste(dir,'\\N170 data.csv',sep=""), header=T)
mydata$DistrictCode=as.character(mydata$DistrictCode) #changes DistrictCode to character
#loop through the data frame and add back in leading zeroes.  While looping, also set value of Grant
obsnum=max(row(mydata))
mydata$Grant=rep(NA,obsnum) #create an empty column
for (i in 1:obsnum)
{
        if (nchar(mydata[i,1])==3) mydata[i,1]=paste('0',mydata[i,1],sep="") #fix DistrictCode
        mydata[i,4]=toupper(paste('MVSUBG',mydata[i,3],sep="")) #set Grant
}
#sort by DistrictCode, that is, the 1st column
mydata=mydata[order(mydata[,1]),]
```

The first line reads the CSV and creates the data frame. The second changes the character type of DistrictCode to character. Then we use a loop to go through each row (observation) in the data frame in order to add in leading zeroes to the DistrictCode and to create the values for the Grant variable. Lastly, we sort the data (using R's syntactically awkward *order* function.)

For Python, we use this code:

```
myinput=filepath+'\\N170 data.csv'
mydata=pd.read_csv(myinput, dtype={'DistrictCode':str})
#dtype sets DistrictCode as character, avoiding the issue we saw in PROC IMPORT
mydata['grant']='MVSUBG'+mydata['GrantStatus'].str.upper() #create and UPCASE grant status
mydata=mydata.sort_values('DistrictCode')
```

The *dtype* option allows us to read in the DistrictCode as a character string, so the leading zeroes are maintained as we create our data frame via the pandas function *read_csv*. We then create the Grant variable then sort.

PART THREE: FREQUENCY TABLES

For SAS, we can use PROC FREQ to create frequency tables so that we can check that we rendered the DistrictCode and Grant correctly.

```
PROC FREQ data=mydata;
      tables
      DistrictCode*DistrictCodeNumber
      grant*GrantStatus
      /list missing nopercent nocum;
      title 'Quality Control Check';
RUN;
```

For R, we use a similar construction with the table function.

```
print("Quality Control Check",quote=F)
table(mydata$DistrictCode)
table(mydata$Grant,mydata$GrantStatus)
```

For Python, we can use the *crosstab* functions built in as part of pandas data frames.

```
print("Quality Control Check")
print(pd.crosstab(mydata['DistrictCode'], columns='count'))
print(pd.crosstab(mydata['grant'], mydata['GrantStatus']))
```

As shown below, the output for all three are rather similar.

Quality Control Check

The FREQ Procedure

DistrictCode	DistrictCodeNumber	Frequency
0160	160	1
0201	201	1
0301	301	1
0401	401	1
0402	402	1
0403	403	1
0404	404	1
0501	501	1
1001	1001	1

grant	GrantStatus	Frequency
MVSUBGNO	No	7
MVSUBGYES	Yes	2

```
[1] Quality Control Check
> table(mydata$DistrictCode)

0160 0201 0301 0401 0402 0403 0404 0501 1001
   1    1    1    1    1    1    1    1    1
> table(mydata$Grant,mydata$GrantStatus)

          No Yes
  MVSUBGNO   7   0
  MVSUBGYES  0   2
```

```
Quality Control Check
col_0          count
DistrictCode
0160               1
0201               1
0301               1
0401               1
0402               1
0403               1
0404               1
0501               1
1001               1
GrantStatus  No  Yes
grant
MVSUBGNO      7    0
MVSUBGYES     0    2
```

Output 1. Quality Control Frequencies in SAS, R, and Python

Note that SAS is the only one with two variables for District Code, as we were able to change it to character in R and maintain it as character in Python without resorting to a different variable.

PART FOUR: CREATING ADDITIONAL METADATA

Now to create the rest of the metadata that we need for our header row, namely the number of observations in our data and the full contents of the header row.

In SAS, one way to do this is as follows:

```
DATA _null_;
      set mydata end=lastobs;
      if lastobs then CALL SYMPUTX("obsnum",_n_);
RUN;
%let firstline="LEA SUBGRANT STATUS,&obsnum,&filename,SAS
%sysfunc(putn(&today,mmddyy6.)),&myyear";
```

For R, we already created *obsnum* for the looping in part two. Thus, we only need one line of code in order to generate *firstline*.

```
firstline=paste("LEA SUBGRANT STATUS,",obsnum,",",filename,",R ",today,",",myyear,sep="")
```

In Python, we can use the syntax below to get the number of observations, then create *firstline*.

```
obsnum=mydata.shape[0]
firstline="LEA SUBGRANT STATUS,"+str(obsnum)+","+filename+",Python "+today+","+myyear
```

PART FIVE: OUTPUT THE CSV

The final CSV we have to make has a file format that requires specific columns in specific positions. It also contains a few "dummy" fields like a file number, some codes that are the same for every row, and some blank fields. We can create those and output the file in one DATA step in SAS, as shown below.

```
data _null_;
      file "&dir\&filename" dsd lrecl=500; *LRECL is arbitrarily large;
      set mydata;
      counter=_n_; StateNumber='45'; StateCode='01';

      if _n_=1 then put &firstline;
      put
      counter :8. StateNumber :$2. StateCode :$2.
      DistrictCode :$4. filler :$1. Grant :$20. filler :$1.;
run;
```

To do the same in R or Python requires a little more work, as we need to create the extra columns as part of the data frame before we export it to CSV. For both languages, I have added the columns to the *mydata* data frame and then isolated those I wanted as a new data frame named *outputThis*.

R:

```
#prepping the columns
mydata$counter=rep(1:obsnum)
mydata$StateNumber='45'; mydata$StateCode='01'; mydata$filler="
outputThis=data.frame(mydata$counter, mydata$StateNumber, mydata$StateCode,mydata$DistrictCode,
mydata$filler, mydata$Grant, mydata$filler)
#creating the csv
output=paste(dir,"\\",filename,sep="")
writeLines(firstline, con=output)
write.table(outputThis, output, append=T, quote=F, sep=",", row.names=F, col.names=F)
```

Python:

```
mydata['counter']=mydata.reset_index().index+1
mydata['StateNumber']='45'
mydata['StateCode']='01'
mydata['filler1']=''
mydata['filler2']=''
outputThis=mydata[['counter','StateNumber','StateCode','DistrictCode','filler1','grant','filler2']].copy()
#print(outputThis)
with open(filepath+'\\'+filename, mode='w') as file:
    file.write(firstline+"\r") #stuff at the end is a carriage return
    outputThis.to_csv(file, index=None, header=False, line_terminator="\n")
```

Note the use of whitespace at the end of the Python code to denote what code is part of the *with open* block.

CONCLUSION: INTEGRATING SAS, R, AND PYTHON

Being a polyglot gives you multiple tools. I hope the example I have given helps some programmers overcome any unease about learning new languages. But in addition to just coding in another language, there are also ways to integrate SAS, R, and Python together.

One example is having one language write and execute code for another language. In my personal experience, I have used SAS to dynamically write Python code as text file, then execute that Python code via an X command. Another example is how the FCMP procedure can use Python objects. The IML procedure can use R packages. Tools like these enable you to use another language without leaving SAS.

Another method is using software that integrates different programming languages. While a thorough discussion of such is outside the scope of this paper, I have included a paper by Isaiah Lankham and Matthew Slaughter in the Recommeded Readings that goes into some depth about SAS and Pyhon.

ACKNOWLEDGMENTS

The author thanks the SESUG chair and staff for this conference and the opportunity to present. He also thanks Isaiah Lankham for helping him get started with Python and discussing the merits of being a polyglot. And to the various bloggers, forum-goers, and Stack Overflow commenters who make the internet a great resource for figuring out how to program something.

ED*Facts* file specifications (data file layouts), including for our example of file 170, can be found online at https://www.ed.gov/data/edfacts-initiative/edfacts-resources/edfacts-file-specifications.

APPENDIX A: RAW DATA

```
DistrictCode,DistrictName,GrantStatus
0160,Abbeville 60,No
0201,Aiken 01,No
0301,Allendale 01,No
0501,Bamberg 01,No
0401,Anderson 01,Yes
0402,Anderson 02,No
0403,Anderson 03,No
0404,Anderson 04,No
1001,Charleston 01,Yes
```

APPENDIX B: SAS CODE

```
*ONE: hard-coded year ID and auto-generated variables based on year/date;
%let yr=1819;
%let myyear=20%substr(&yr,1,2)-20%substr(&yr,3,2);
%let today=%sysfunc(today());
%let filename=SCLEASUBGRANTS170%substr(&myyear,8,2)SA.csv;
%let dir=O:\SESUG 2025;

*TWO: read and clean input;
proc import datafile="&dir\N170 data.csv" dbms=csv replace out=mydata;
        getnames=yes;
run;
data mydata;
        set mydata(RENAME=DistrictCode=DistrictCodeNumber);
        length DistrictCode $4. grant $20.;
        DistrictCode=put(DistrictCodeNumber,z4.);
        grant=upcase(compress('MVSUBG'||GrantStatus));
run;
proc sort data=mydata; BY DistrictCode; run;

*THREE: check reformatting;
proc freq data=mydata;
        tables
        DistrictCode*DistrictCodeNumber
        grant*GrantStatus
        /list missing nopercent nocum;
        title 'Quality Control Check';
run;

*FOUR: get and format metadata;
data _null_;
        set mydata end=lastobs;
        if lastobs then CALL SYMPUTX("obsnum",_n_); *puts n-count in macro variable obsnum;
run;
%let firstline="LEA SUBGRANT STATUS,&obsnum,&filename,SAS
%sysfunc(putn(&today,mmddyy6.)),&myyear";

*FIVE: make the deliverable, a CSV file;
data _null_;
        file "&dir\&filename" dsd lrecl=500; *LRECL set arbitrarily large;
        set mydata;
        counter=_n_; StateNumber='45'; StateCode='01';

        if _n_=1 then put &firstline;
        put
        counter :8. StateNumber :$2. StateCode :$2.
        DistrictCode :$4. filler :$1. Grant :$20. filler :$1.;
run;
```

APPENDIX C: R CODE

```
#PART ONE
yr="1819"
myyear=paste("20",substr(yr,1,2),"-20",substr(yr,3,4),sep="")
today=as.character(Sys.Date()) #yyyy-mm-dd format
today=paste(substr(today,6,7),substr(today,9,10),substr(today,3,4),sep="")  #convert to mmddyy format
filename=paste("SCLEASUBGRANTS170",substr(myyear,8,10),"RR.csv",sep="")
dir="O:\\SESUG 2025"

#PART TWO
mydata=read.csv(file=paste(dir,'\\N170 data.csv',sep=""), header=T)
mydata$DistrictCode=as.character(mydata$DistrictCode) #as with SAS, DistrictCode was read in as
numeric
#loop through the data frame and add back in leading zeroes.  While looping, also set value of Grant
obsnum=max(row(mydata))
mydata$Grant=rep(NA,obsnum) #create an empty column
for (i in 1:obsnum)
{
        if (nchar(mydata[i,1])==3) mydata[i,1]=paste('0',mydata[i,1],sep="") #fix DistrictCode
        mydata[i,4]=toupper(paste('MVSUBG',mydata[i,3],sep="")) #set Grant
}
#sort by DistrictCode, that is, the 1st column
mydata=mydata[order(mydata[,1]),]

#PART THREE
print("Quality Control Check",quote=F)
table(mydata$DistrictCode)
table(mydata$Grant,mydata$GrantStatus)

#PART FOUR
#note that obsnum was made earlier, as we needed it in Part Two
firstline=paste("LEA SUBGRANT STATUS,",obsnum,",",filename,",R ",today,",",myyear,sep="")

#PART FIVE
#prepping the columns
mydata$counter=rep(1:obsnum)
mydata$StateNumber='45'; mydata$StateCode='01'; mydata$filler="
outputThis=data.frame(mydata$counter, mydata$StateNumber, mydata$StateCode,
mydata$DistrictCode, mydata$filler, mydata$Grant, mydata$filler)
#creating the csv
output=paste(dir,"\\",filename,sep="")
writeLines(firstline, con=output)
write.table(outputThis, output, append=T, quote=F, sep=",", row.names=F, col.names=F)
```

APPENDIX D: PYTHON CODE

```
import pandas as pd
from datetime import date

#PART ONE
yr="1819"
myyear="20"+yr[0:2]+"-20"+yr[2:4] #should be 2018-2019

today=date.today()
today=today.strftime("%m%d%y")

filename="SCLEASUBGRANTS170"+myyear[7:9]+"PY.csv"
filepath="O:\SESUG 2025"
#you can check any of these by typing a PRINT command like
#print(yr)

#PART TWO
myinput=filepath+'\\N170 data.csv'
mydata=pd.read_csv(myinput, dtype={'DistrictCode':str})
#dtype sets DistrictCode as character, avoiding the issue we saw in PROC IMPORT
mydata['grant']='MVSUBG'+mydata['GrantStatus'].str.upper() #create and UPCASE grant status
mydata=mydata.sort_values('DistrictCode')

#PART THREE
print("Quality Control Check")
print(pd.crosstab(mydata['DistrictCode'], columns='count'))
print(pd.crosstab(mydata['grant'], mydata['GrantStatus']))
#instead of using CROSSTAB for DistrictCode, you could use
#print(mydata['DistrictCode'])
#since only 1 row per district, but crosstab better replicates PROC FREQ

#PART FOUR
obsnum=mydata.shape[0] #note: mydata.count() does something slightly different than desired
firstline="LEA SUBGRANT STATUS,"+str(obsnum)+","+filename+",Python "+today+","+myyear
#print(firstline)

#PART FIVE
mydata['counter']=mydata.reset_index().index+1
mydata['StateNumber']='45'
mydata['StateCode']='01'
mydata['filler1']=''
mydata['filler2']=''
outputThis=mydata[['counter','StateNumber','StateCode','DistrictCode','filler1','grant','filler2']].copy()
#print(outputThis)
with open(filepath+'\\'+filename, mode='w') as file:
    file.write(firstline+"\r") #stuff at the end is a carriage return
    outputThis.to_csv(file, index=None, header=False, line_terminator="\n")
```

RECOMMENDED READING

Delwiche, Lora D. and Susan J. Slaughter. 2003. *The Little SAS Book: A Primer, Third Edition*. Cary, NC: SAS Institute Inc.

Hemedinger, Chris. "How to run SAS programs in Jupyter Notebook." https://blogs.sas.com/content/sasdummy/2016/04/24/how-to-run-sas-programs-in-jupyter-notebook/

Lankham, Isaiah and Matthew Slaughter. "Everything is better with friends: Executing SAS code in Python scripts with SASPY". https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3189-2019.pdf
See also: https://github.com/saspy-bffs/pharmasug-2023-proc-fcmp-python and https://github.com/saspy-bffs/pharmasug-2023-class

"Python for SAS Users." http://blog.rubypdf.com/2018/10/12/python-for-sas-users/

SAS Documentation. "Calling Functions in the R Language." https://documentation.sas.com/doc/en/imlug/15.2/imlug_r_toc.htm

SAS Documentation. "Using PROC FCMP Python Objects." https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lecompobjref/p18qp136f91aaqn1h54v3b6pkant.htm

VanderPlas, Jake. (2016). *A Whirlwind Tour of Python*. O'Reilly Media: Sebastopol, CA. Available at https://jakevdp.github.io/WhirlwindTourOfPython/

CONTACT INFORMATION

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

Aaron R. Brown
South Carolina Department of Education
1429 Senate Street, Columbia, SC 29201
Work Phone: 803-734-8858
E-mail: ARBrown@ed.sc.gov