Paper 3884-2019

# Comparing SAS® and Python – A Coder's Perspective

Daniel R. Bretheim, Willis Towers Watson

## ABSTRACT

When you see an interesting data set, report, or figure, do you wonder what it would take to replicate those outputs in SAS®? This paper does just that, by using SAS to re-create outputs that were originally generated by Python.

A key concept for understanding this comparison is that the starting point is the Python code. The paper associates snippets of Python with the corresponding SAS statements, attempting a reasonable apples-to-apples comparison. In other words, the resulting SAS code will not necessarily represent how it would have been written if we had started with SAS rather than Python. The paper illustrates how SAS code lines up with the widely used Python language.

We start with a brief summary of how SAS and Python compare across several environmental dimensions, followed by a simple example that introduces a comparison of code and syntax. A second example uses SAS to recreate a complex graph, where the code comparison will be broken down into the following task areas:

- Data manipulation
- Data aggregation
- Logic
- Graphing

The comparison summarizes the following measures:

- Lines of code
- Run time
- Readability

The appendices contain the complete programs and the output graphs for the graph example.

## INTRODUCTION

This is **not** a contest between SAS and Python. Rather, this is a comparison of two highly used and well known languages, intended to address the curiosity of SAS programmers that are interested in what another popular language **looks** like when performing similar tasks.

The code contained in this paper should not necessarily be considered to represent "best practice" or optimal coding. As any programmer knows, there are often many different ways to accomplish a coding task.

Finally, this paper is not necessarily intended to teach elements of either language. This is primarily a visual comparison of code and syntax.

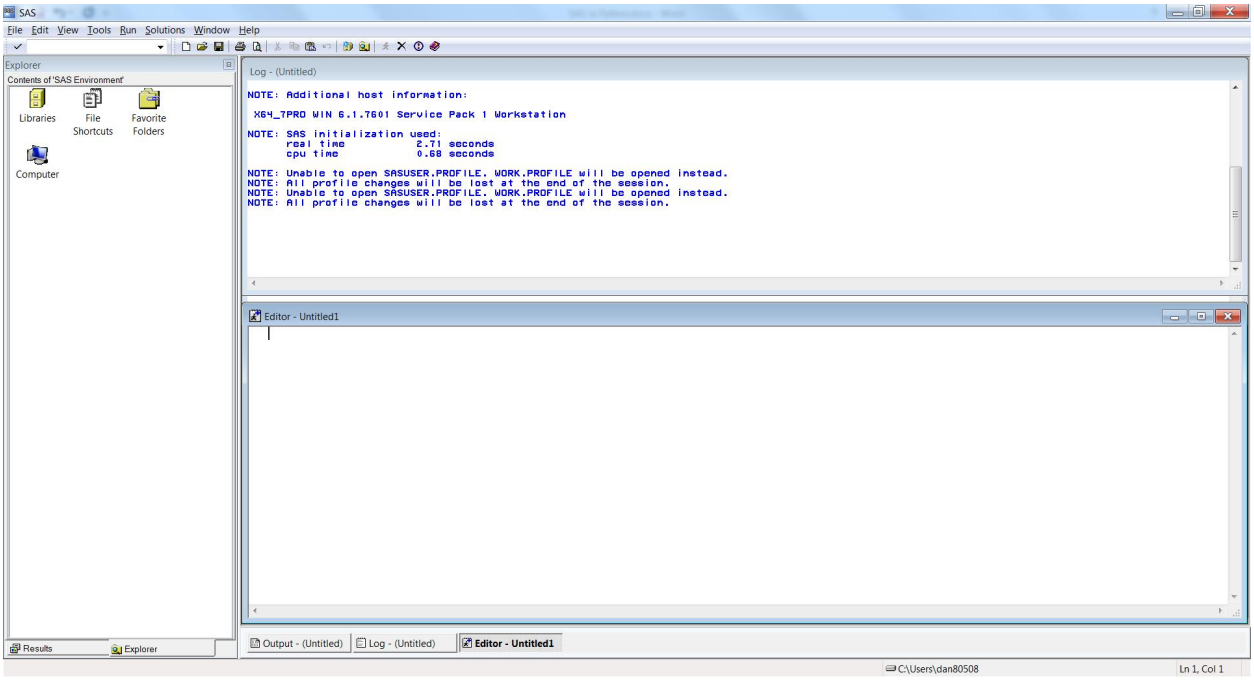
## ENVIRONMENT COMPARISON

| | **SAS** | **Python** |
|---|---|---|
| **Environment:** | | |
| Version | 9.4 | 3.6.2 |
| Type of language | Interpreted | Interpreted |
| Integrated Development Environment (IDE) | Display Manager | Spyder (one of many) |
| Data structures | SAS data sets | Series = array-like object with an |

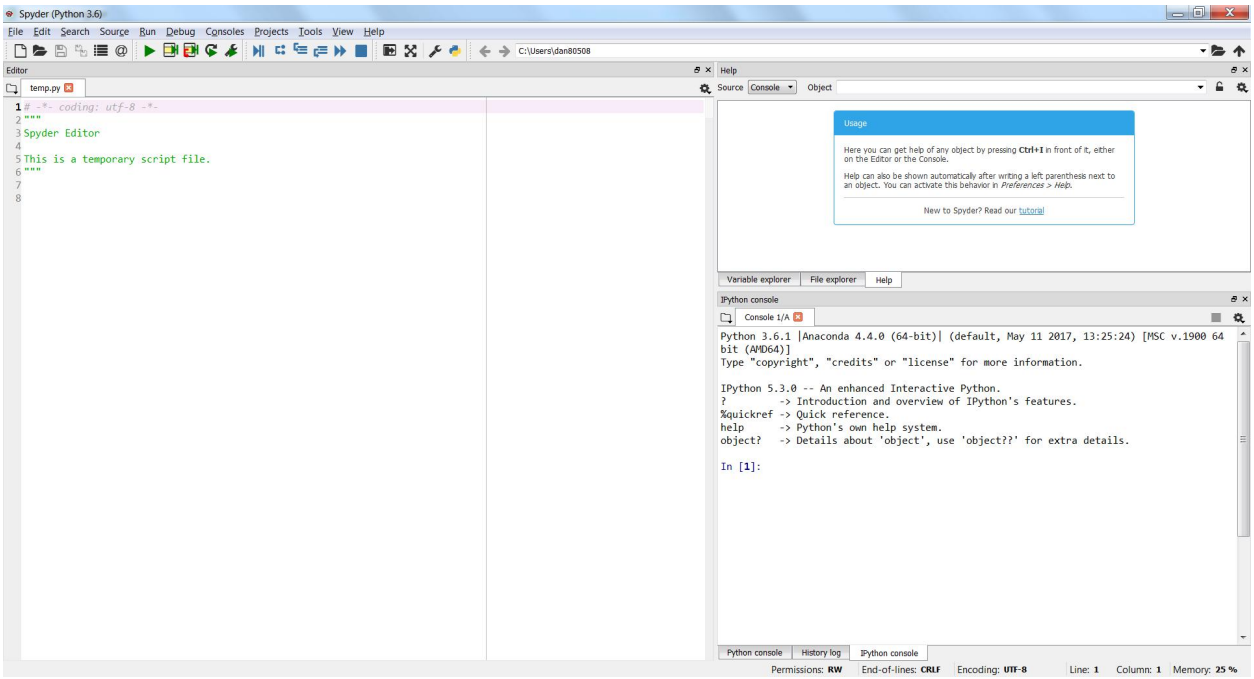
| | | |
|---|---|---|
| | | index<br>data frames = rows and columns with two indices |
| Defining a statement | Semi-colon | 4 space indentation |
| Comment indicator | * | # |
| Program structure | DATA and PROC steps | Single statements or function calls |
| Libraries | Called when needed. | Import required libraries, e.g.:<br>Pandas = data analysis library<br>Numpy = scientific computing library<br>Matplotlib = graphics library<br>Time = timer library |

**Table 1. Examples of SAS® vs. Python Programming Environment Features**

## IDE COMPARISON




**Display 1. SAS® IDE – Display Manager**




**Display 2. Python IDE (example – there are many): Spyder**

## EXAMPLES

### EXAMPLE 1

We'll begin with a simple example where the purpose of the Python script is to read an Excel file, create several new data elements, subset the file, calculate summary statistics and output the results to Excel.

| **Task:** Import required packages. |
|---|
| **Python** |
| import pandas as pd |
| **SAS** |
| n/a |

| **Task:** Read Excel file. |
|---|
| **Python** |
| xls_file = pd.ExcelFile('C:\Python\data\Rawdata.xlsx') |
| **SAS** |
| **proc import** out = xls_file<br>datafile= "C:\Python\data\Rawdata.xlsx"<br>dbms=xlsx replace;<br>sheet="Final";<br>getnames=yes; |

| **Task:** Read into a dataframe, create new data elements, and subset the rows. |
|---|
| **Python** |
| # Read into a DataFrame<br>results = xls_file.parse('Final')<br><br># Calculate new data elements<br>results['Ret'] = results['DBER'] + results['DCER']<br>results['Health'] = results['MDER'] + results['RMER'] + results['DNER'] + results['DNRER'] + results['DTER'] + results['DTRER'] + results['LTER']<br>results['Total'] = results['Ret'] + results['Health']<br>results['PTO'] = results['SDER'] + results['VAER'] |

```
results['TotDB'] = results['DBER'] + results['DBERNQ']
results['TotDC'] = results['DCER'] + results['DCERNQ']
results['RetNQ'] = results['DBERNQ'] + results['DCERNQ']
results['TotRet'] = results['Ret'] + results['RetNQ']
results['TotalAll'] = results['Total'] + results['RetNQ']
results['TotRew'] = results['Base'] + results['Bonus'] + results['Total']
results['TotRewAll'] = results['TotRew'] + results['RetNQ']

# Create empty DataFrames
clients = []

# Select a subset of rows from the results DataFrame
clients = results.query('MarketID <= 2000 and ProgramID == 477')
```

**SAS**

```
data clients;

  set xls_file;

  * calculate new data elements ;
  Ret = DBER + DCER;
  Health = MDER + RMER + DNER + DNRER + DTER + DTRER + LTER;
  Total = Ret + Health;
  PTO = SDER + VAER;
  TotDB = DBER + DBERNQ;
  TotDC = DCER + DCERNQ;
  RetNQ = DBERNQ + DCERNQ;
  TotRet = Ret + RetNQ;
  TotalAll = Total + RetNQ;
  TotRew = Base + Bonus + Total;
  TotRewAll = TotRew + RetNQ;

  * subset rows ;
  if (MarketID <= 2000 and ProgramID = 477) then output;
```

**Task:** Calculate summary statistics.

**Python**

```
# Calculate means by MarketID from the client DataFrame and output to a new DataFrame
client = []

client=clients.groupby(['MarketID'])[['Age','Svc','Base','Bonus','PctFemale','TotDB','TotDC','DTER','DTRER','SDER','
LTER','MDER','RMER','DNER','DNRER','VAER','TotRet','Health','TotalAll','PTO','TotRewAll']].mean()
```

**SAS**

```
* Sort by EmployeeeID and ProgramID ;
proc sort data = clients;
  by MarketID;

* Calculate means by market ID ;
proc univariate data=clients noprint;
  var Age Svc Base Bonus PctFemale TotDB TotDC DTER DTRER SDER LTER MDER RMER DNER DNRER
      VAER TotRet Health TotalAll PTO TotRewAll;

  by MarketID;

  output out = client mean = Age Service Base Bonus PctFemale TotalPension TotalDC Death RetDth STD LTD
                             Medical PRM Dental Retden Vacation TotalRet Health TotalAll PTO TotRewAll;
```

**Task:** Output to Excel.

**Python**

```
writer = pd.ExcelWriter('client_py.xlsx')
client.to_excel(writer,'Sheet1')
writer.save()
```

**SAS**

```
proc export data= client
  outfile= "C:\SAS Work\SGF2019\client_sas.xlsx"
  dbms=xlsx replace;
run;
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **MarketID** | **Age** | **Svc** | **Base** | **Bonus** | **PctFemale** |
| 2 | **1114** | 54 | 10 | 67433.6 | 4046 | 100 |
| 3 | **1115** | 50 | 8 | 85051.2 | 5103 | 100 |
| 4 | **1116** | 36 | 4 | 83228 | 8323 | 100 |
| 5 | **1117** | 47 | 5 | 74172.8 | 4450 | 0 |
| 6 | **1118** | 36 | 5 | 101440.5 | 10144 | 0 |
| 7 | **1119** | 49 | 6 | 108139.2 | 6488 | 0 |
| 8 | **1120** | 40 | 7 | 101129.6 | 10113 | 100 |
| 9 | **1121** | 46 | 11 | 130338 | 15641 | 0 |
| 10 | **1122** | 46 | 10 | 136589 | 13659 | 0 |
| 11 | **1123** | 49 | 11 | 156251 | 23438 | 0 |
| 12 | **1124** | 50 | 11 | 187460 | 28119 | 0 |
| 13 | **1125** | 53 | 17 | 220531 | 44106 | 0 |
| 14 | **1126** | 54 | 12 | 261126.8 | 78338 | 0 |

**Output 1. First Six Columns in the Python Output**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | MarketID | Age | Service | Base | Bonus | PctFemale |
| 2 | 1114 | 54 | 10 | 67433.6 | 4046 | 100 |
| 3 | 1115 | 50 | 8 | 85051.2 | 5103 | 100 |
| 4 | 1116 | 36 | 4 | 83228 | 8323 | 100 |
| 5 | 1117 | 47 | 5 | 74172.8 | 4450 | 0 |
| 6 | 1118 | 36 | 5 | 101440.455 | 10144 | 0 |
| 7 | 1119 | 49 | 6 | 108139.2 | 6488 | 0 |
| 8 | 1120 | 40 | 7 | 101129.6 | 10113 | 100 |
| 9 | 1121 | 46 | 11 | 130338 | 15641 | 0 |
| 10 | 1122 | 46 | 10 | 136589 | 13659 | 0 |
| 11 | 1123 | 49 | 11 | 156251 | 23438 | 0 |
| 12 | 1124 | 50 | 11 | 187460 | 28119 | 0 |
| 13 | 1125 | 53 | 17 | 220531 | 44106 | 0 |
| 14 | 1126 | 54 | 12 | 261126.835 | 78338 | 0 |

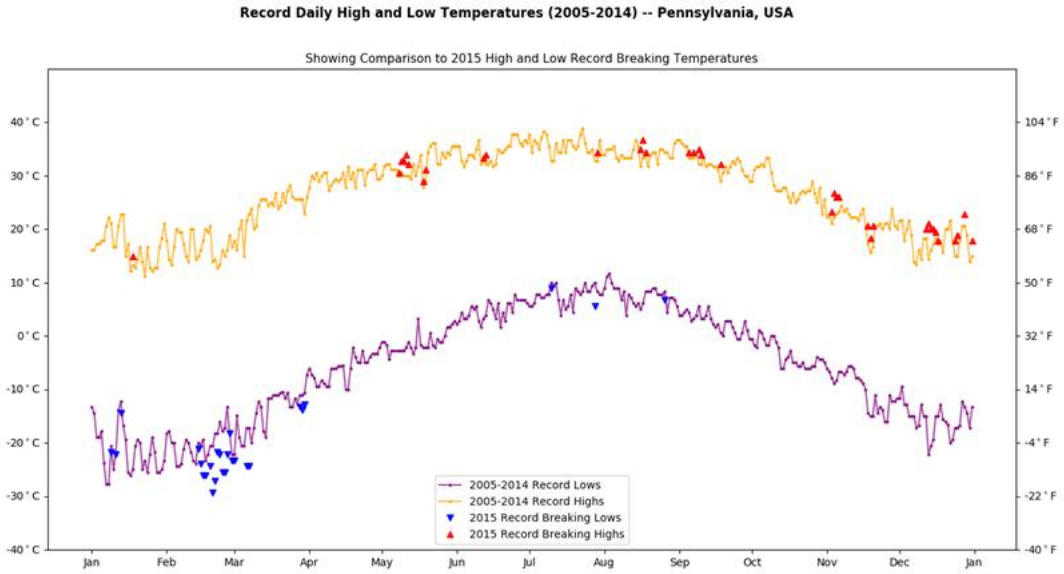
**Output 2. First Six Columns in the SAS® Output**

Some differences to note:

- A Python script must specify all the packages that will be required by the script.
- Python has nothing comparable to a DATA step.
- Some sorting in Python is done automatically.
- SAS uses the RUN statement to initiate processing of the final step.
- In this example, Python required 22 statements compared to 25 for SAS.
- Slightly different output formatting.

## EXAMPLE 2

Here we have a more complex example, where the objective is to replicate with SAS, the following graph originally produced using Python.




**Output 3. Original Python Graph**

The data for this more complex example come from a subset of The National Centers for Environmental Information (NCEI) Daily Global Historical Climatology Network. The NCEI is the data center that supports the National Oceanic and Atmospheric Administration (NOAA). The data are comprised of daily climate records from land surface stations across the globe. The data set selected for this project is from the Pittsburgh, PA area. It contains 83,943 rows and four columns with multiple observations per date.

Each row of the NOAA data set corresponds to a single observation consisting of the following variables:

- ID – station identification code
- Date – date in YYYY-MM-DD format
- Element – indicator of element type
    - TMAX – maximum temperature (tenths of degrees Celsius)
    - TMIN – minimum temperature (tenths of degrees Celsius)
- Data_Value – data value for Element (tenths of degrees Celsius)

**Sample Observations**

```
ID,Date,Element,Data_Value
USC00360861,2015-09-29,TMAX,239
USW00004726,2014-01-18,TMIN,-138
USC00364214,2014-09-16,TMAX,222
USC00362183,2009-04-07,TMAX,200
USC00367782,2009-02-08,TMAX,128
```

The tasks required to produce a single graph are:

- Return a line graph of the record high and record low temperatures by day of the year over the period 2005-2014.

- Overlay a scatter plot of the 2015 data for any points (highs and lows) for which the ten year record (2005-2014) record high or record low was broken in 2015.
- Identify and remove leap days (i.e., February 29).
- Add elements to the graph to enhance interpretation, e.g., legend, labels.

## Data Manipulation Tasks

**Task:** Read a csv file and output a data set.

**Python**

```
noaa = pd.read_csv('c:\Python\data\noaa_extract.csv')
```

**SAS**

```
proc import datafile='c:\Python\data\noaa_extract.csv'
  out=noaa
  dam's=csv replace;
```

**Task:** Print first few rows of a data set and the number of observations in the output file.

**Python**

```
print(noaa.head(5))
print('NOAA raw data rows ', lean(noaa))
```

**SAS**

```
proc print data=noaa (obs=5);
* SAS automatically prints the number of obs to the log. ;
```

**Task:** Convert text date field to date-time value.

**Python**

```
noaa['Date'] = pd.to_datetime(noaa['Date'])
```

**SAS**

SAS imports a YYYY-MM-DD field as a SAS date value.

**Task:** Create an index object for conversion to a string.

**Python**

```
noaa_index = pd.DatetimeIndex(noaa['Date'])
```

**SAS**

n/a

**Tasks:**

1. Delete an unnecessary column.
2. Convert date value to a string; create a new column from an existing data element.
3. Concatenation
4. Delete rows based on value.
5. Divide a data element by a constant.
6. Subset a data file.
7. Replace the Year with a constant (2015) to facilitate group-by processing.

**Python**

```
# 1.  Delete an unnecessary column.
noaa.drop(['ID'], axis=1, inplace=True)

# 2.  Convert date value to string; Create a new column from an existing data element.
noaa['Year']  = noaa_index.strftime("%Y")

# 3.  Concatenation
noaa['MoDay'] = noaa_index.strftime("%m-%d")

# 4.  Delete rows based on value.
noaa = noaa.loc[noaa['MoDay'] != "02-29"]

# 5.  Divide a data element by a constant.
noaa['Data_Value'] = noaa['Data_Value'] /10
```

```
# 6.  Subset a data file.
noaa_2015 = noaa.loc[noaa['Year'] == "2015"]
noaa = noaa.loc[noaa['Year'] < "2015"]

# 7.  Replace the year with a constant (2015) to facilitate group-by processing
noaa['Year'] = "2015".
```

**SAS**

```
data noaa_2015 noaa;
  length MoDay $5.;

  * 1. Delete an unnecessary column. ;
  set noaa (drop=ID);

  * 2. Convert a date value to string.  Create new columns from an existing data element. ;
  Year = year(date);
  Mo = put(month(date), $2.);
  Day = put(day(date), $2.);

  * 3. Concatenation ;
  if countc(Mo, ' ')=1 then Mo='0'||compress(Mo);
  if countc(Day, ' ')=1 then Day='0'||compress(Day);
  MoDay = Mo||'-'||Day;

  * 4. Delete rows based on value. ;
  if MoDay ne '02-29';

  * 5. Divide a data element by a constant. ;
  Data_Value = Data_Value/10;

  * 6. Subset a data file. ;
  if Year=2015 then output noaa_2015;
  else do;
     * 7. Replace the year with a constant (2015) to facilitate group-by processing. ;
     Year=2015;
     output noaa;
  end;
```

## Data Aggregation Tasks

**Task:** Calculate the 2005-2014 maximum temperatures.

**Python**

```
max_10 = noaa[noaa["Element"]=="TMAX"].groupby(["MoDay"])["Data_Value"].max()
# Change series to numpy array
max_10 = np.array(list(max_10))
```

**SAS**

```
proc sql;
  create table max_10 as
  select Moday, max(Data_Value) as max_10
  from noaa
  where Element = 'TMAX'
  group by Moday;
```

**Task:** Calculate the 2005-2014 minimum temperatures.

**Python**

```
min_10 = noaa[noaa["Element"]=="TMIN"].groupby(["MoDay"])["Data_Value"].min()
# Change series to numpy array
min_10 = np.array(list(min_10))
```

**SAS**

```
proc sql;
  create table min_10 as
```

```
  select Moday, min(Data_Value) as min_10
  from noaa
  where Element = 'TMIN'
  group by Moday;
```

**Task:** Calculate the 2015 maximum temperatures.

**Python**

```
max_2015 = noaa_2015[noaa_2015["Element"]=="TMAX"].groupby(["MoDay"])["Data_Value"].max()
```

**SAS**

```
proc sql;
  create table max_2015 as
  select Moday, max(Data_Value) as max_2015
  from noaa_2015
  where Element = 'TMAX'
  group by Moday;
```

**Task:** Calculate the 2015 minimum temperatures.

**Python**

```
min_2015 = noaa_2015[noaa_2015["Element"]=="TMIN"].groupby(["MoDay"])["Data_Value"].min()
```

**SAS**

```
proc sql;
  create table min_2015 as
  select Moday, min(Data_Value) as min_2015
  from noaa_2015
  where Element = 'TMIN'
  group by Moday;
```

## Logic Tasks

**Task:** Identify the record breaking high temperatures in 2015.

**Python**

```
max_break_points = (max_2015 > max_10).as_matrix()
```

**SAS**

```
proc sql;
  create table max_break_points as
  select Max_2015.MoDay, max_2015
    from Max_2015 full join Max_10
    on Max_2015.MoDay = Max_10.Moday
    where Max_2015.max_2015 > Max_10.max_10;
```

**Task:** Identify the record breaking low temperatures in 2015.

**Python**

```
min_break_points = (min_2015 < min_10).as_matrix()
```

**SAS**

```
proc sql;
  create table min_break_points as
  select Min_2015.MoDay, min_2015
    from Min_2015 full join Min_10
    on Min_2015.MoDay = Min_10.Moday
    where Min_2015.min_2015 < Min_10.min_10;
```

## Graphing Tasks

**Task:** Set the figure size.

**Python**

```
mpl.rcParams['figure.figsize'] = (16, 8)
```

**SAS**

```
goptions device=pug gsfname=graphout hsize=15in vsize=8in;
```

**Task:** Create 365 date values for the x-axis.

**Python**

```
x = pd.Series(range(0,365))
xdate=pd.to_datetime('01-01-2015')+pd.to_timedelta(pd.Series(x), 'D')
# Change list to numpy array
x_axis = np.array(list(xdate))
```

**SAS**

```
data dates365 (drop=i);
  format date mmddyy10.;
  if _n_ = 1 then do;
    date='01Jan2015'd;
    output;
  end;
  do i = 1 to 364;
    date = date+1;
    output;
  end;
```

**Task:** Format the x-axis as months.

**Python**

```
months = mdates.MonthLocator()
plt.gca().xaxis.set_major_locator(months)
myFmt=mdates.DateFormatter('%b')
plt.gca().xaxis.set_major_formatter(myFmt)
```

**SAS**

```
proc sql;
  create table tempx as
  select Max_10.*, Min_10.*, Max_break_points.*, Min_break_points.*, Max_10.max_10 *1.8 + 32 as F
  from Max_10 as one
    join Min_10 as two
      on one.MoDay = two.Moday
    left join Max_break_points as three
      on two.MoDay = three.Moday
    left join Min_break_points as four
      on two.MoDay = four.Moday;

proc sql;
  create table Dates365r as
  select monotonic() as row, *
  from Dates365;

proc sql;
  create table tempxr as
  select monotonic() as row, *
  from tempx;

proc sql;
  create table tempss as
  select Dates365r.*, tempxr.*
  from Dates365r as one
    join tempxr as two
      on one.row = two.row;

proc format;
  value mmm_fmt
  "01Jan2015"d ='Jan'
  "01Feb2015"d ='Feb'
  "01Mar2015"d ='Mar'
  "01Apr2015"d ='Apr'
  "01May2015"d ='May'
  "01Jun2015"d ='Jun'
  "01Jul2015"d ='Jul'
  "01Aug2015"d ='Aug'
  "01Sep2015"d ='Sep'
```

```
  "01Oct2015"d ='Oct'
  "01Nov2015"d ='Nov'
  "01Dec2015"d ='Dec'
  "01Jan2016"d ='Jan' ;

*  Apply this format in PROC GPLOT. ;
format date mmm_fmt.;
```

**Task:** Titles

**Python**

```
plt.suptitle("Record Daily High and Low Temperatures (2005-2014) -- Pennsylvania, USA", fontsize=12,
fontweight='bold')
plt.title("Showing Comparison to 2015 High and Low Record Breaking Temperatures", fontsize=11)
```

**SAS**

```
title1 bold 'Record Daily High and Low Temperatures (2005-2014) -- Pennsylvania, USA';
title2 ' ';
title3 'Showing Comparison to 2015 High and Low Record Breaking Temperatures';
```

**Tasks:**

1. Line graph the 2005-2014 min and max temperatures.
2. Overlay the 2015 min and max temperatures as a scatter plot.
3. Create and position a legend.
4. Plot a 2nd y-axis for Fahrenheit.
5. Display the graph.
6. Save the graph as a png file.

**Python**

```
# 1.  Line graph the 2005-2014 min and max temperatures.  Specify tick mark formatting.
plt.plot(x_axis, min_10, '-o', color='purple', ms=2, alpha=0.7)
plt.plot(x_axis, max_10, '-o', color='orange', ms=2, alpha=0.7)

# 2.  Overlay the 2015 min and max temperatures as a scatter plot.  Specify tick mark formatting.
plt.plot(x_axis[min_break_points], min_2015[min_break_points], 'bv', x_axis[max_break_points],
max_2015[max_break_points], 'r^', alpha=.9)

# 3.  Create and position a legend.
plt.legend(['2005-2014 Record Lows', '2005-2014 Record Highs', '2015 Record Breaking Lows', '2015 Record
Breaking Highs'], loc='lower center')

# 4.  Specify y-axes tick ranges and plot a 2nd y-axis for Fahrenheit.
ax1 = plt.gca()
ticks = np.arange(-40,50,10)
ax1.set_ylim(-40,50)
ax1.set_yticks(ticks)
ax1.set_yticklabels(map(lambda x: '{:}$^\circ$C'.format(x),ax1.get_yticks()))
ax2 = ax1.twinx()
ax2.set_ylim(ax1.get_ylim())
ax2.set_yticks(ax1.get_yticks())
ax2.set_yticklabels(map(lambda x: '{0:g}$^\circ$F'.format(x),(32+ax1.get_yticks()*1.8)))

# 5.  Display the graph.
plt.show()

# 6.  Save the graph as a png file.
plt.savefig("C:\SAS Work\Python\graphs\SGFy.png")
```

**SAS**

```
* 4. Specify x-axis and y-axes tick ranges. ;
axis1 order=("01Jan2015"d to "01Jan2016"d by month) offset=(3,3) label=none;
axis2 label=(height=3 pct ' ') minor=none offset=(3pct, 3pct) order=(-40 to 40 by 10);
axis3 label=(height=3 pct ' ') minor=none offset=(3pct, 3pct) order=(-40 to 104 by 18);

* 6. Save the graph as a png file. ;
```

```
filename graphout 'C:\SAS Work\Python\graphs\Plot_SAS.png';
* Set figure dimensions. ;
goptions device=png gsfname=graphout hsize=15in vsize=8in;

proc gplot data=temps;
  format date mmm_fmt.;

  * 1. Specify tick mark formatting
  symbol1 interpol=none
     value="D" font=MARKER
     color=blue;
  symbol2 interpol=none
     value="C" font=marker
     color=red;
  symbol3 interpol=join
     value=dot
     color=purple;
  symbol4 interpol=join
     value=dot
     color=orange;
  symbol5 value=none;

  * 1. Line graph the 2005-2014 min and max temperatures.
  * 2. Overlay the 2015 min and max temperatures as a scatter plot.
  plot min_10*date max_10*date min_2015*date max_2015*date / legend=legend1 overlay
    haxis=axis1 hminor=0 vaxis=axis2 vminor=0;

  * 3. Create and position a legend.
  legend1 label=none
     value=('2005-2014 Record Lows'  '2005-2014 Record Highs'
              '2015 Record Breaking Lows' '2015 Record Breaking Highs')
     across=1 down=4
     position=(bottom center inside)
     mode=share
     frame;

  * 4. Plot a 2nd y-axis for Fahrenheit.
  plot2 F*date / vaxis=axis3 vminor=0;

  * left y-axis label ;
  note move=(+.5,+.75) height=11 pt 'o'
        move=(+.3,-.5)  height=13 pt 'C';

  * right y-axis label ;
  note move=(123,48)  height=11 pt 'o'
        move=(+.3,-.5) height=13 pt 'F';

run;
quit;
```

| **Task:** Calculate run time. |
|---|
| **Python** |
| from datetime import timedelta<br>start_time2 = time.monotonic()<br>temps()<br>end_time2 = time.monotonic()<br>print(timedelta(seconds=end_time2 - start_time2)) |
| **SAS** |
| /* Start timer – place at the beginning of the program * /<br>%let _timer_start = %sysfunc(datetime());<br><br>/* Stop timer – place at the end of the program */ |

```
data _null_;
  dur = datetime() - &_timer_start;
  put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
run;
```

## CONCLUSION

"Beauty is in the eye of the beholder", is a phrase that first appeared in the 3rd century Greek. Whether your preference is strawberry ice cream over chocolate, or SAS over Python, there is value knowing something about the other flavor. Individual preferences are based on the many features of any programming language. In addition to the visual comparison of what "comparable" code looks like, the following results were obtained on the other comparisons of interest.

| Comparison | Python | SAS |
|---|---|---|
| Lines of essential code | 57 | 81 |
| Statement layout | horizontal | vertical |
| Run time | 2.44 seconds | 2.15 seconds |
| Readability | Very subjective – you decide | |

**Table 2. Examples of SAS® vs. Python Programming Comparisons**

Other issues that can influence software preference:

- IDE functionality
- Debugging aides
- Memory requirements
- Documentation
- Technical support
- Cost
- Legacy software in your organization

Another resource for SAS users that is related to Python is the SAS® Viya® product. SAS Viya is a cloud-enabled, in-memory analytics engine with a standardized code base that supports programming in SAS and other programming languages such as Python, R, Java, and Lua.

## CONTACT INFORMATION

Contact the author at:

Dan Bretheim
Willis Towers Watson
daniel.bretheim@willistowerswatson.com

## APPENDIX A – CODE

### PYTHON CODE

```
def temps():
    import pandas as pd
    import numpy as np
    import matplotlib as mpl
    import matplotlib.pyplot as plt
    from   matplotlib.ticker import FormatStrFormatter
    import matplotlib.dates as mdates

    # 1. Read csv file and output a dataframe – print the first few rows.
    noaa = pd.read_csv('c:\Python\data\noaa_extract.csv')
    print(noaa.head(5))
    print('NOAA raw data rows ', len(noaa))

    # 2. Convert Date to datetime values.
    noaa['Date'] = pd.to_datetime(noaa['Date'])

    # 3. Create a date index object.
    noaa_index = pd.DatetimeIndex(noaa['Date'])

    # 4. Delete an unnecessary column.
    noaa.drop(['ID'], axis=1, inplace=True)

    # 5. Convert the dates to strings, concatenate, and add two new columns.
    noaa['Year']  = noaa_index.strftime("%Y")
    noaa['MoDay'] = noaa_index.strftime("%m-%d")

    # 6. Delete leap day rows based on value.
    noaa = noaa.loc[noaa['MoDay'] != "02-29"]
    print('After dropping leap days ', len(noaa))

    # 7. Convert tenths of degrees Celsius to Celsius.
    noaa['Data_Value'] = noaa['Data_Value'] /10

    # 8. Separate the data to 2015 and 2005-2014.
    noaa_2015 = noaa.loc[noaa['Year'] == "2015"]
    print('2015 subset ', len(noaa_2015))
    noaa = noaa.loc[noaa['Year'] < "2015"]
    print('2005-2014 subset ', len(noaa))

    # 9. Replace the Year with 2015 to facilitate group-by processing.
    noaa['Year'] = "2015"
    print('After changing Year to 2015 ',len(noaa))

    # 10. Calculate the 2005-2014 maximum temperatures (n=365).
    max_10 = noaa[noaa["Element"]=="TMAX"].groupby(["MoDay"])["Data_Value"].max()
    print('# of max points ', len(max_10))
    # Change series to numpy array
    max_10 = np.array(list(max_10))

    # 11. Calculate the 2005-2014 minimum temperatures (n=365).
    min_10 = noaa[noaa["Element"]=="TMIN"].groupby(["MoDay"])["Data_Value"].min()
    print('# of min points ', len(min_10))
    # Change series to numpy array
    min_10 = np.array(list(min_10))
```

```
    # 12. Calculate the 2015 maximum temperatures (n=365).
    max_2015 = noaa_2015[noaa_2015["Element"]=="TMAX"].groupby(["MoDay"])["Data_Value"].max()
    print('max_2015 rows ', len(max_2015))

    # 13. Calculate the 2015 minimum temperatures (n=365).
    min_2015 = noaa_2015[noaa_2015["Element"]=="TMIN"].groupby(["MoDay"])["Data_Value"].min()
    print('min_2015 rows ', len(min_2015))

    # 14. Identify the record breaking temperatures in 2015.
    max_break_points = (max_2015 > max_10).as_matrix()
    print('# of max break points ', np.sum(max_break_points))
    min_break_points = (min_2015 < min_10).as_matrix()
    print('# of min break points ', np.sum(min_break_points))

    # 15. Set the figure size.
    mpl.rcParams['figure.figsize'] = (16, 8)

    # 16. Create 365 date values (0-364) for the x-axis.
    x = pd.Series(range(0,365))
    xdate=pd.to_datetime('01-01-2015')+pd.to_timedelta(pd.Series(x), 'D')
    # Change list to numpy array
    x_axis = np.array(list(xdate))
    print('len x_axis ', len(x_axis))

   # 17. Format x-axis as months.
    months = mdates.MonthLocator()
    plt.gca().xaxis.set_major_locator(months)
    # %b = month
    myFmt=mdates.DateFormatter('%b')
    plt.gca().xaxis.set_major_formatter(myFmt)

    # 18. Titles.
    plt.suptitle("Record Daily High and Low Temperatures (2005-2014) -- Pennsylvania, USA", fontsize=12,
fontweight='bold')
    plt.title("Showing Comparison to 2015 High and Low Record Breaking Temperatures", fontsize=11)

    # 19. Line graph the 2005-2014 min and max temperatures.
    plt.plot(x_axis, min_10, '-o', color='purple', ms=2, alpha=0.7)
    plt.plot(x_axis, max_10, '-o', color='orange', ms=2, alpha=0.7)

    # 20. Overlay the 2015 min and max temperatures as a scatter plot.
    plt.plot(x_axis[min_break_points], min_2015[min_break_points], 'bv', x_axis[max_break_points],
max_2015[max_break_points], 'r^', alpha=.9)

    # 21. Create and position a legend.
    plt.legend(['2005-2014 Record Lows', '2005-2014 Record Highs', '2015 Record Breaking Lows', '2015 Record
Breaking Highs'], loc='lower center')

    # 22. Plot a second y-axis for Fahrenheit on the right side and specify y-axis tick ranges and formatting.
    ax1 = plt.gca()
    ticks = np.arange(-40,50,10)
    ax1.set_ylim(-40,50)
    ax1.set_yticks(ticks)
    ax1.set_yticklabels(map(lambda x: '{:}$^\circ$C'.format(x),ax1.get_yticks()))
    ax2 = ax1.twinx()
    ax2.set_ylim(ax1.get_ylim())
    ax2.set_yticks(ax1.get_yticks())
    ax2.set_yticklabels(map(lambda x: '{0:g}$^\circ$F'.format(x),(32+ax1.get_yticks()*1.8)))
```

```
    # 23. Display the graph.
    plt.show()

    # 24. Save the graph as a png file.
    plt.savefig("C:\SAS Work\Python\graphs\Plot_Python.png")

# 25. Calculate run time.
import time
from datetime import timedelta
start_time2 = time.monotonic()
temps()
end_time2 = time.monotonic()
print(timedelta(seconds=end_time2 - start_time2))
```

## SAS CODE

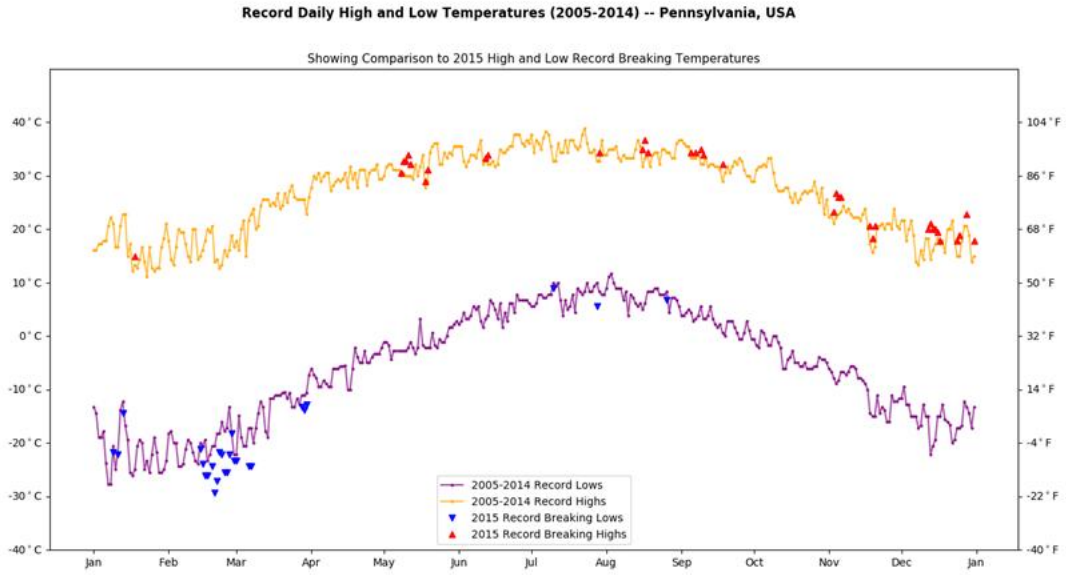
```
* Start timer. ;
%let _timer_start = %sysfunc(datetime());

* 1. Read csv file and output a SAS data set – print the first few rows. ;
proc import datafile='c:\Python\data\noaa_extract.csv'
  out=noaa
  dbms=csv replace;
proc print data=noaa (obs=5);
run;

data noaa_2015 noaa;
  length MoDay $5.;

  * 4. Delete an unnecessary column. ;
  set noaa (drop=ID);

  * 5. Convert the dates to strings, concatenate, and add new columns. ;
  Year = year(date);
  Mo = put(month(date), $2.);
  Day = put(day(date), $2.);
  if countc(Mo, ' ')=1 then Mo='0'||compress(Mo);
  if countc(Day, ' ')=1 then Day='0'||compress(Day);
  MoDay = Mo||'-'||Day;

  * 6. Delete leap day rows based on value. ;
  if MoDay ne '02-29';

  * 7. Divide a data element by a constant. ;
  Data_Value = Data_Value/10;

  * 8. Subset a data file. ;
  if Year=2015 then output noaa_2015;
  else do;
  * 9. Replace the year with a constant (2015) to facilitate group-by processing. ;
    Year=2015;
    output noaa;
  end;
run;

* 10. Calculate the 2005-2014 maximum temperatures.  ;
* 365 rows and 2 columns ;
proc sql;
  create table max_10 as
  select Moday, max(Data_Value) as max_10
  from noaa
  where Element = 'TMAX'
  group by Moday;
quit;

* 11. Calculate the 2005-2014 minimum temperatures.  ;
* 365 rows and 2 columns ;
proc sql;
  create table min_10 as
  select Moday, min(Data_Value) as min_10
  from noaa
  where Element = 'TMIN'
  group by Moday;
quit;
```

```
* 12. Calculate 2015 maximum temperatures.  ;
* 365 rows and 2 columns ;
proc sql;
  create table max_2015 as
  select Moday, max(Data_Value) as max_2015
  from noaa_2015
  where Element = 'TMAX'
  group by Moday;
quit;

* 13. Calculate the 2015 minimum temperatures.  ;
* 365 rows and 2 columns ;
proc sql;
  create table min_2015 as
  select Moday, min(Data_Value) as min_2015
  from noaa_2015
  where Element = 'TMIN'
  group by Moday;
quit;

* 14. Identify the record breaking temperatures in 2015. ;
* 36 record breaking highs ;
proc sql;
  create table max_break_points as
  select Max_2015.MoDay, max_2015
     from Max_2015 full join Max_10
     on Max_2015.MoDay = Max_10.Moday
     where Max_2015.max_2015 > Max_10.max_10;
quit;

* 26 record breaking highs ;
proc sql;
  create table min_break_points as
  select Min_2015.MoDay, min_2015
     from Min_2015 full join Min_10
     on Min_2015.MoDay = Min_10.Moday
     where Min_2015.min_2015 < Min_10.min_10;
quit;

* 16. Create 365 values for the x-axis.;
data dates365 (drop=i);
  format date mmddyy10.;
  if _n_ = 1 then do;
    date='01Jan2015'd;
    output;
  end;
  do i = 1 to 364;
    date = date+1;
    output;
  end;
run;
```

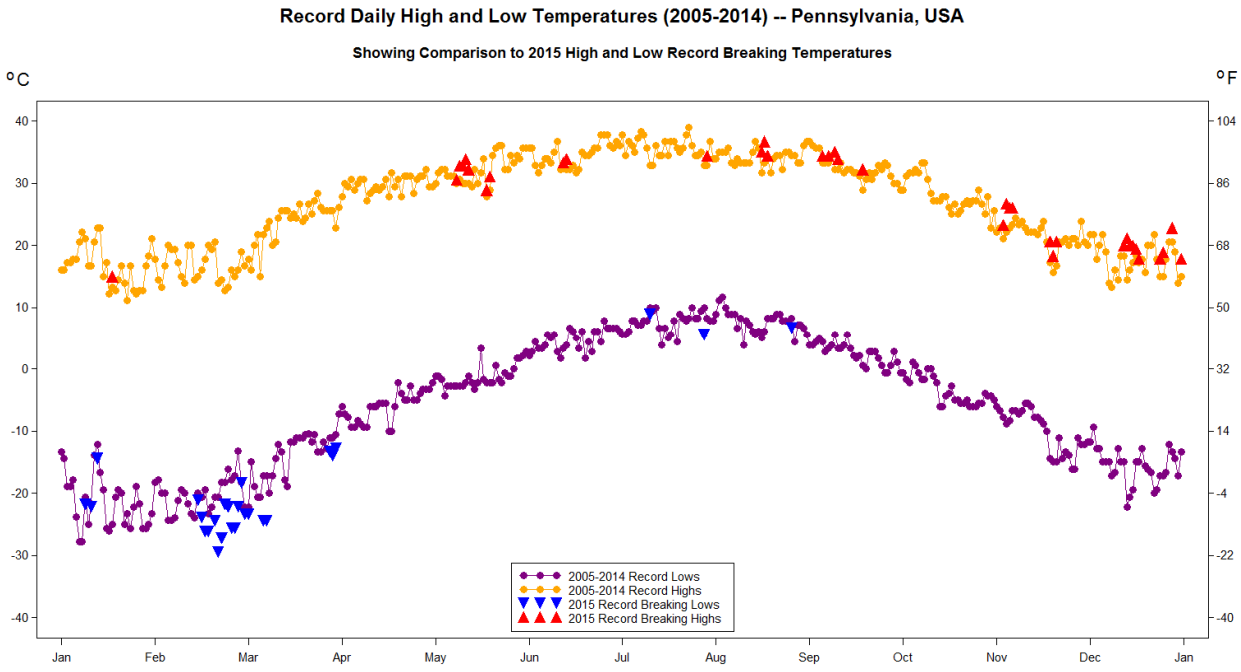
```
proc sql;
  create table tempx as
  select Max_10.*, Min_10.*, Max_break_points.*, Min_break_points.*, Max_10.max_10 *1.8 + 32 as F
  from Max_10 as one
    join Min_10 as two
      on one.MoDay = two.Moday
    left join Max_break_points as three
      on two.MoDay = three.Moday
    left join Min_break_points as four
      on two.MoDay = four.Moday;
quit;

proc sql;
  create table Dates365r as
  select monotonic() as row, *
  from Dates365;
quit;

proc sql;
  create table tempxr as
  select monotonic() as row, *
  from tempx;
quit;

proc sql;
  create table temps as
  select Dates365r.*, tempxr.*
  from Dates365r as one
    join tempxr as two
      on one.row = two.row;
quit;

* 17. Format x-axis as months. ;
proc format;
  value mmm_fmt
  "01Jan2015"d ='Jan'
  "01Feb2015"d ='Feb'
  "01Mar2015"d ='Mar'
  "01Apr2015"d ='Apr'
  "01May2015"d ='May'
  "01Jun2015"d ='Jun'
  "01Jul2015"d ='Jul'
  "01Aug2015"d ='Aug'
  "01Sep2015"d ='Sep'
  "01Oct2015"d ='Oct'
  "01Nov2015"d ='Nov'
  "01Dec2015"d ='Dec'
  "01Jan2016"d ='Jan' ;
run;

* 18. Titles ;
title1 bold 'Record Daily High and Low Temperatures (2005-2014) -- Pennsylvania, USA';
title2 ' ';
title3 'Showing Comparison to 2015 High and Low Record Breaking Temperatures';

* 22. Specify x-axis and y-axes tick ranges. ;
axis1 order=("01Jan2015"d to "01Jan2016"d by month) offset=(3,3) label=none;
axis2 label=(height=3 pct ' ') minor=none offset=(3pct, 3pct) order=(-40 to 40 by 10));
axis3 label=(height=3 pct ' ') minor=none offset=(3pct, 3pct) order=(-40 to 104 by 18);
```

```
* 24. Save the graph as a png file. ;
filename graphout 'C:\SAS Work\Python\graphs\Plot_SAS.png';

* 15. Set the figure size. ;
goptions device=png gsfname=graphout hsize=15in vsize=8in;

proc gplot data=temps;
  * Format the x-axis as months. ;
  format date mmm_fmt.;

  * 22. Specify tick mark formatting. ;
  symbol1 interpol=join
        value=dot
        color=purple;
  symbol2 interpol=join
        value=dot
        color=orange;
  symbol3 interpol=none
        value="D" font=MARKER
        color=blue;
  symbol4 interpol=none
        value="C" font=marker
        color=red;
  symbol5 value=none;

  * 19. Line graph the 2005-2014 min and max temperatures. ;
  * 20. Overlay the 2015 min and max temperatures as a scatter plot. ;
  plot min_10*date max_10*date min_2015*date max_2015*date / legend=legend1 overlay
     haxis=axis1 hminor=0 vaxis=axis2 vminor=0;

  * 21. Create and position a legend. ;
  legend1 label=none
        value=('2005-2014 Record Lows' '2005-2014 Record Highs'
                '2015 Record Breaking Lows' '2015 Record Breaking Highs' )
        across=1 down=4
        position=(bottom center inside)
        mode=share
        frame;

  * 22.  Plot a 2nd y-axis for Fahrenheit. ;
  plot2 F*date / vaxis=axis3 vminor=0;

  * Label for Celsius. ;
  note move=(+.5,+.75) height=11 pt 'o'
         move=(+.3,-.5)  height=13 pt 'C';

  * Label for Fahrenheit. ;
  note move=(148,48)  height=11 pt 'o'
         move=(+.3,-.5) height=13 pt 'F';
run;
quit;

* 25. Stop timer. ;
data _null_;
  dur = datetime() - &_timer_start;
  put 30*'-' / ' TOTAL DURATION:' dur time13.2 / 30*'-';
run;
```

## APPENDIX B – GRAPH OUTPUT




**Output 4. Original Python Graph**




**Output 5. SAS® Graph**