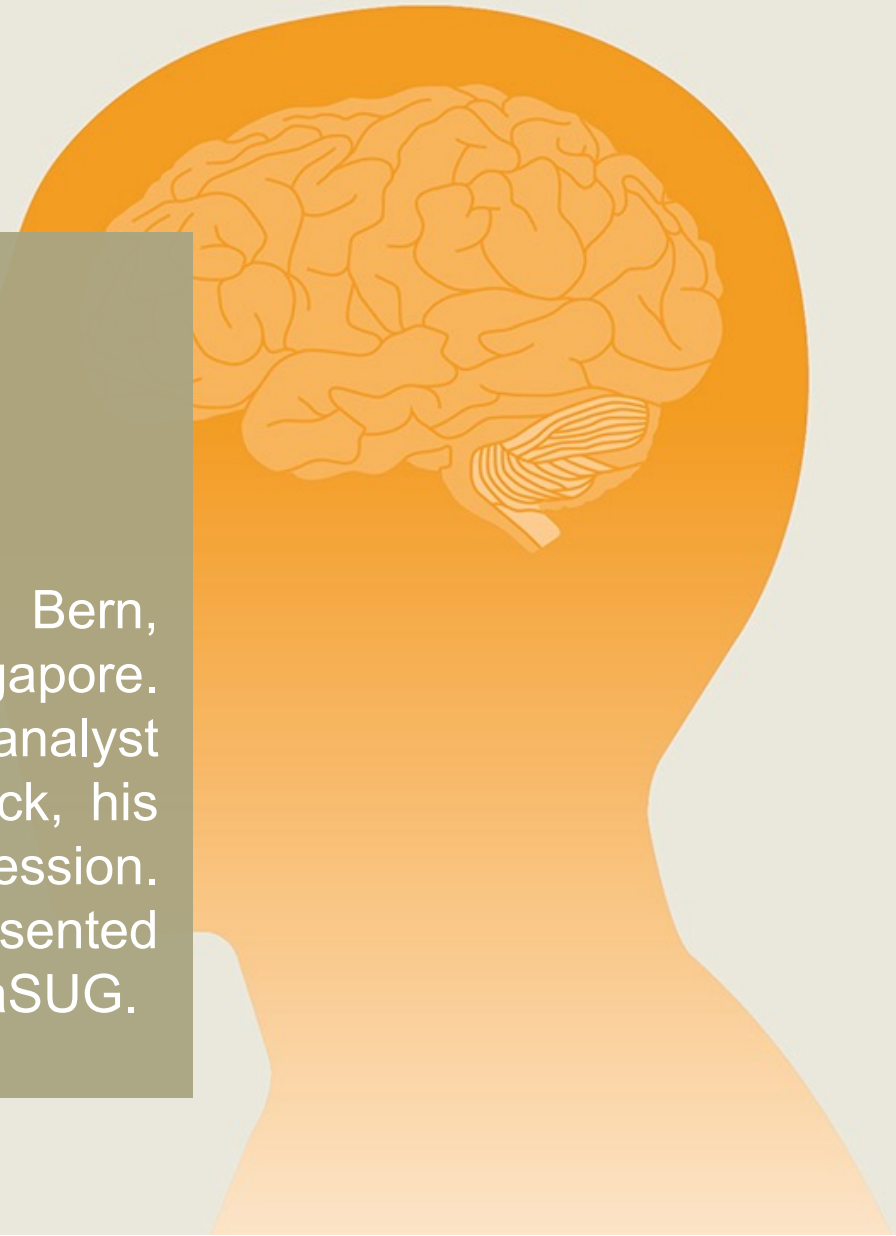


AIK HOE SEAH

Principal Statistical Programmer

H. Lundbeck A/S

Aik Hoe Seah received his M.Sc. from University of Bern, Switzerland, and his B.Sc from National University of Singapore. Before joining Lundbeck in 2017, Aik Hoe was a statistical analyst at Lilly where he supported the diabetes area. In Lundbeck, his work has been focused on late phase compounds for depression. He has been using SAS for more than 10 years and has presented in SAS Global Forum, local SAS forums and recently PharmaSUG.

FOR THE POLYGLOT PROGRAMMER: VALIDATING STATISTICAL OUTPUTS USING MULTIPLE PROGRAMMING LANGUAGES

Aik Hoe Seah

Lundbeck


Salve
Sveiki
Tere
Gruezi
Dia dhuit
Selamat
Hallau
Hujambo
Bok
Xin chao
Saluton
Czesc
Sziasztok
Womenjeka
Hej
Halo
Konnichiwa
Ahoj
Kia ora
Ciao
Hello
Annyeong
Aloha
Bonjour
Mire dita
Ni Hao
Hallo
Bula
Zdravo
Gruss Gott
Hola
Oi
Merhaba
Sveiki
Terve
Hei

Introduction

- A statistical programmer:
  - uses a database → create computer programs that performs tasks like statistical modelling, summary tables and data visualizations.
- A polyglot programmer:
  - perform said tasks using multiple programming languages and at the same time, benefit from such efficiencies.

Introduction

- Program validation is an important quality control (QC) process.

- One person is the ‘main’ programmer, while a second programmer will perform the QC.
- Same set of specifications → independent programs development → outputs match → validated ✓

- We will discuss some benefits of creating a Python program, that can validate statistical output, utilising multiple programming languages like SAS, R and Python.

Initial Steps

Python program

SAS 9.4 or highe

(Jul 2013,
9.4 is a requirem
for SASPy to wo

stribution

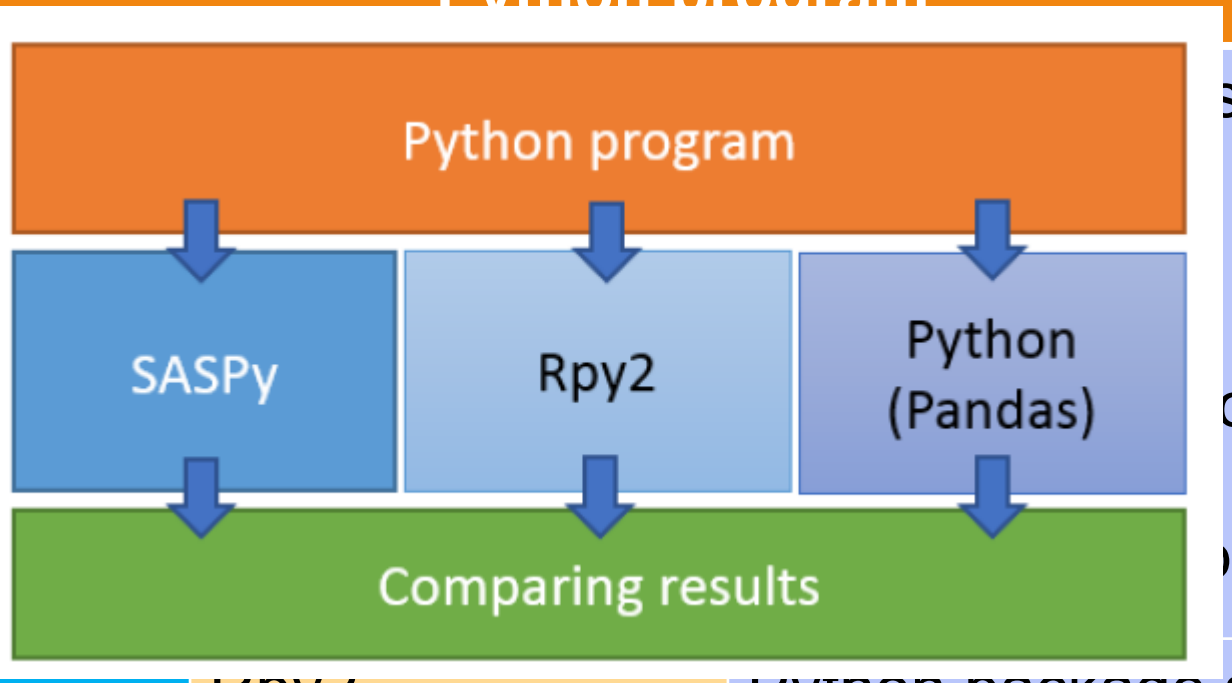
otebook, Spyder;

andas, Matplotlib)

SASPy

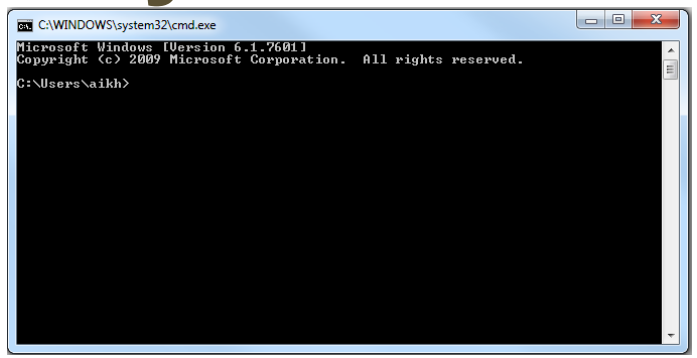
Rpy2

Python package statsmodels

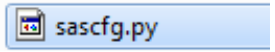


Installation and configuring SASPy

- On Windows, start cmd.exe
- Navigate to where Anaconda is installed
- Find pip.exe
- Key in command: pip install saspy



- Configure your sascfg.py (found in: Anaconda3/Lib/site-packages/saspy)




```
SAS_config_names=['winlocal']
winlocal = {'java'      : 'java',
            'encoding'  : 'windows-1252',
            'classpath' : cpW
            }
```

Initial Steps

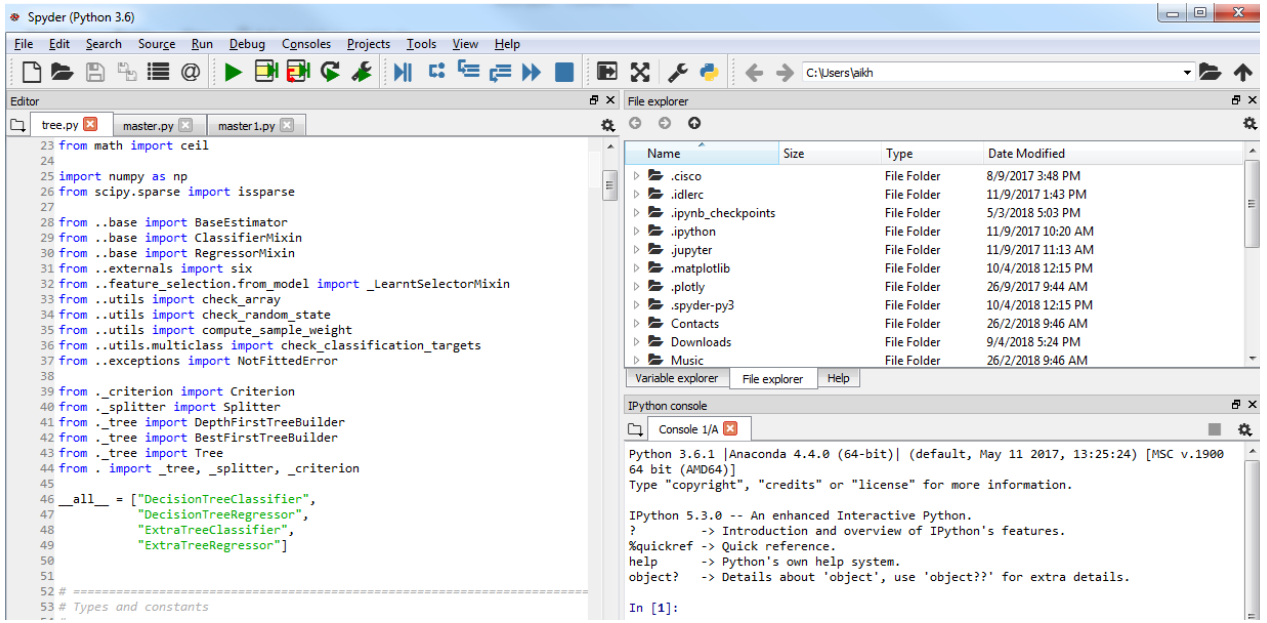
- Configure sascfg.py: may also need to update path of jar files

```
# build out a local classpath variable to use below for Windows clients
cpW  =  "C:\\Program Files\\SAS94\\SASDeploymentManager\\9.4\\products\\deploywiz__94360__prt__xx__sp0__1\\deploywiz\\sas.svc.connection.jar"
cpW += ";C:\\Program Files\\SAS94\\SASDeploymentManager\\9.4\\products\\deploywiz__94360__prt__xx__sp0__1\\deploywiz\\log4j.jar"
cpW += ";C:\\Program Files\\SAS94\\SASDeploymentManager\\9.4\\products\\deploywiz__94360__prt__xx__sp0__1\\deploywiz\\sas.security.sspi.jar"
cpW += ";C:\\Program Files\\SAS94\\SASDeploymentManager\\9.4\\products\\deploywiz__94360__prt__xx__sp0__1\\deploywiz\\sas.core.jar"
cpW += ";C:\\Users\\aikh\\AppData\\Local\\Continuum\\Anaconda3\\Lib\\site-packages\\saspy\\java\\saspyiom.jar"
```

- Testing out SASpy

```
In [1]: import saspy

In [2]: sas = saspy.SASsession(cfgname='winlocal')
SAS Connection established. Subprocess id is 1944
```



How Spyder IDE looks like

Installation and configuring Rpy2

- In cmd.exe, key in command: pip install rpy2
- No additional configuration needed
- Testing out rpy2

```
In [3]: import rpy2

In [4]: print(rpy2.__version__)
2.8.6
```

Running Your Analysis

- Reading in a dataset

```
In [5]: import pandas as pd
   ...: advs_sas = pd.read_sas("C:/test1.sas7bdat")

In [6]: advs_sas.head()
Out[6]:
     STUDYID                 USUBJID    BASETYPE      APHASE  NOMDAY  NOMWEEK  \
0  b'12345A'  b'12345A-AB1001-S1102'  b'PHASE A'  b'PHASE A'     0.0      0.0
1  b'12345A'  b'12345A-AB1001-S1102'  b'PHASE A'  b'PHASE B'    28.0      4.0
2  b'12345A'  b'12345A-AB1001-S1102'  b'PHASE B'  b'PHASE B'     0.0      0.0
3  b'12345A'  b'12345A-AB1001-S1102'  b'PHASE A'  b'PHASE B'    84.0     12.0
4  b'12345A'  b'12345A-AB1001-S1102'  b'PHASE B'  b'PHASE B'    56.0      8.0

      SVDT   ADY  AWTARGET        AVISIT AVISITN FUFL  VSSEQ
0  26986.0   0.0       NaN      b'BASE1'     2.0  NaN   15.0
1  27013.0   0.0       NaN      b'BASE2'     6.0  NaN   42.0
2  27013.0   0.0       NaN      b'BASE2'     6.0  NaN   42.0
3  27069.0  56.0       NaN  b'COMP/WITH'    12.0  NaN   84.0
4  27069.0  56.0       NaN  b'COMP/WITH'    12.0  NaN   84.0

[5 rows x 38 columns]
```

- Fix any encoding issues

```
In [7]: advs_sas['STUDYID'] = advs_sas['STUDYID'].str.decode('utf-8')
   ...: advs_sas['USUBJID'] = advs_sas['USUBJID'].str.decode('utf-8')
   ...: advs_sas['BASETYPE'] = advs_sas['BASETYPE'].str.decode('utf-8')
   ...: advs_sas['PARAMCD'] = advs_sas['PARAMCD'].str.decode('utf-8')
   ...: advs_sas['APHASE'] = advs_sas['APHASE'].str.decode('utf-8')
   ...: advs_sas['AVISIT'] = advs_sas['AVISIT'].str.decode('utf-8')

In [8]: advs_sas.head()
Out[8]:
  STUDYID              USUBJID BASETYPE   APHASE  NOMDAY  NOMWEEK    NOMWEEKC  \
0  12345A  12345A-AB1001-S1102  PHASE A  PHASE A     0.0      0.0   b'Week 0'
1  12345A  12345A-AB1001-S1102  PHASE A  PHASE B    28.0      4.0   b'Week 4'
2  12345A  12345A-AB1001-S1102  PHASE B  PHASE B     0.0      0.0   b'Week 0'
3  12345A  12345A-AB1001-S1102  PHASE A  PHASE B    84.0     12.0  b'Week 12'
4  12345A  12345A-AB1001-S1102  PHASE B  PHASE B    56.0      8.0   b'Week 8'

    ADY  AWTARGET     AVISIT AVISITN FUFL  VSSEQ
0   0.0       NaN      BASE1     2.0  NaN   15.0
1   0.0       NaN      BASE2     6.0  NaN   42.0
2   0.0       NaN      BASE2     6.0  NaN   42.0
3  56.0       NaN  COMP/WITH    12.0  NaN   84.0
4  56.0       NaN  COMP/WITH    12.0  NaN   84.0

[5 rows x 38 columns]
```

- Drop NaN columns, convert df to SAS dataset

```
In [9]: advs_sas2a = advs_sas.drop(['CRIT1', 'CRIT1FL', 'FUFL'], axis=1)
   ...: advs_sas2b = sas.df2sd(advs_sas2a, 'advs', 'WORK')
```

Creating Summary Tables

- SAS code wrapped in Python

```
In [6]: c1 = sas.submit("""
   ...: proc sort data=advs;
   ...:     by paramcd avisitn avisit;
   ...: run;
   ...: proc means data=advs n mean min max;
   ...:     where basetype="PHASE A" and avisit not in ("EARLY1", "EARLY2");
   ...:     by paramcd avisitn avisit;
   ...:     var chg;
   ...:     ods output summary=sum1;
   ...: run;
   ...: """)

In [7]: advs_sas2out = sas.sd2df('sum1', 'work')
   ...: advs_sas2out
Out[7]:
  PARAMCD  AVISITN   AVISIT  CHG_N  CHG_Mean  CHG_Min  CHG_Max
0     OPR         1  SCREEN     39  0.205128      -15       14
1     OPR         2   BASE1     40  0.000000        0        0
2     OPR         3   VIS03     40  1.150000      -16       10
3     OPR         4   VIS04     40 -0.225000      -21       11
4     OPR         5   VIS05     35  2.800000       -8       32
```

- Remove trailing spaces in strings

```
In [8]: advs_sas2out['PARAMCD'] = advs_sas2out['PARAMCD'].str.strip()
   ...: advs_sas2out['AVISIT'] = advs_sas2out['AVISIT'].str.strip()
   ...: advs_sas2out
Out[8]:
  PARAMCD  AVISITN  AVISIT  CHG_N  CHG_Mean  CHG_Min  CHG_Max
0     OPR        1  SCREEN     39  0.205128      -15       14
1     OPR        2   BASE1     40  0.000000        0        0
2     OPR        3   VIS03     40  1.150000      -16       10
3     OPR        4   VIS04     40 -0.225000      -21       11
4     OPR        5   VIS05     35  2.800000       -8       32
```

Creating Summary Tables

- R code wrapped in Python

```
In [17]: import rpy2.rinterface as rinterface
    ...: import rpy2.robjects as robjects
    ...: from rpy2.robjects.packages import importr
    ...: rinterface.initr()
    ...:
    ...: base = importr('base')
    ...: sas7bdat = importr('sas7bdat')
    ...: robjects.r('advs_r <- read.sas7bdat("C:/test1.sas7bdat")')
    ...:
    ...: plyr = importr('plyr')
```

```
In [21]: robjects.r('advs_r1 <- select(filter(advs_r, (advs_r$BASETYPE == "PHASE A") & !(advs_r$AVISIT == "EARLY1" | advs_r$AVISIT ==
"EARLY2")), c(1:38))')
    ...:
    ...: robjects.r('advs_r2 <- ddply(advs_r1, c("advs_r1$PARAMCD", "advs_r1$AVISITN", "advs_r1$AVISIT"), summarise, N = sum(!
is.na(CHG)), mean = mean(CHG, na.rm=TRUE), min = min(CHG, na.rm=TRUE), max = max(CHG, na.rm=TRUE))')
    ...: print(robjects.r('advs_r2'))
  advs_r1$PARAMCD  advs_r1$AVISITN advs_r1$AVISIT   N      mean   min   max
1             OPR              1.0         SCREEN  39  0.205128 -15.0  14.0
2             OPR              2.0          BASE1  40  0.000000   0.0   0.0
3             OPR              3.0          VIS03  40  1.150000 -16.0  10.0
4             OPR              4.0          VIS04  40 -0.225000 -21.0  11.0
5             OPR              5.0          VIS05  35  2.800000  -8.0  32.0
```

Creating Summary Tables

- Converting the R dataframe back to a Python dataframe
- Reset index: R (starts from 1) to Python (starts from 0)

```
In [22]: from rpy2.robjects import r, pandas2ri
    ...: pandas2ri.activate()
    ...:
    ...: r.data('advs_r2')
    ...: r['advs_r2'].head()
    ...:
    ...: advs_rout = r['advs_r2']
    ...: advs_rout1 = advs_rout
    ...: advs_rout1.columns = ['PARAMCD', 'AVISITN', 'AVISIT', 'CHG_N', 'CHG_Mean', 'CHG_Min', 'CHG_Max']
    ...: advs_rout1 = advs_rout.reset_index()
    ...: advs_rout2 = advs_rout1.drop(['index'], axis=1)
    ...: advs_rout2
Out[22]:
  PARAMCD  AVISITN  AVISIT  CHG_N  CHG_Mean  CHG_Min  CHG_Max
0     OPR      1.0  SCREEN     39  0.205128    -15.0     14.0
1     OPR      2.0   BASE1     40  0.000000      0.0      0.0
2     OPR      3.0   VIS03     40  1.150000    -16.0     10.0
3     OPR      4.0   VIS04     40 -0.225000    -21.0     11.0
4     OPR      5.0   VIS05     35  2.800000     -8.0     32.0
```

Creating Summary Tables

- Summary tables via using Python Pandas

```
In [24]: advs_py = advs_sas
    ...: advs_py.head()
    ...:
    ...: advs_py1 = advs_py[advs_py['BASETYPE'] =="PHASE A"]
    ...: advs_py2 = advs_py1[advs_py1['AVISIT'] !="EARLY1"]
    ...: advs_py3 = advs_py2[advs_py2['AVISIT'] !="EARLY2"]
    ...:
    ...: advs_py4 = advs_py3.groupby(["PARAMCD", "AVISITN", "AVISIT"]).agg({"CHG": ['count', 'mean', 'min', 'max']})
    ...: advs_py5 = advs_py4
    ...: advs_py5.columns = ['CHG_N', 'CHG_Mean', 'CHG_Min', 'CHG_Max']
    ...: advs_py6 = advs_py5.reset_index()
    ...: advs_py6
Out[24]:
  PARAMCD  AVISITN  AVISIT  CHG_N  CHG_Mean  CHG_Min  CHG_Max
0     OPR      1.0  SCREEN     39  0.205128    -15.0     14.0
1     OPR      2.0   BASE1     40  0.000000      0.0      0.0
2     OPR      3.0   VIS03     40  1.150000    -16.0     10.0
3     OPR      4.0   VIS04     40 -0.225000    -21.0     11.0
4     OPR      5.0   VIS05     35  2.800000     -8.0     32.0
```

Comparing Table Output Results

```
In [25]: (advs_py6 != advs_rout2).any(1)
Out[25]:
0    False
1    False
2    False
3    False
4    False
dtype: bool

In [26]: (advs_py6 != advs_sas3out).any(1)
Out[26]:
0     True
1    False
2    False
3    False
4    False
dtype: bool

In [27]: (advs_rout2 != advs_sas3out).any(1)
Out[27]:
0     True
1    False
2    False
3    False
4    False
dtype: bool
```

- Python vs R - no differences
- Python vs SAS - some differences
- R vs SAS - some differences

Comparing Table Output Results

- Further investigation of the differences between the 2 dataframes was not helpful. Visually, the 2 dataframes looked like they matched exactly.

```
In [28]: import numpy
    ...: ne_stacked = (advs_py6 != advs_sas3out).stack()
    ...: changed = ne_stacked[ne_stacked]
    ...: changed.index.names = ['id', 'col']
    ...: difference_locations = np.where(advs_py6 != advs_sas3out)
    ...: changed_from = advs_py6.values[difference_locations]
    ...: changed_to = advs_sas3out.values[difference_locations]
    ...: diff02 = pd.DataFrame({'from': changed_from, 'to': changed_to}, index=changed.index)
    ...: diff02
Out[28]:
                 from        to
id col
0  CHG_Mean  0.205128  0.205128
```

Comparing Table Output Results

- Define dataframe comparison as a function

```
In [29]: def compareDF(df1, df2, y):
    ...:     ne_stk = (df1.round(y) != df2.round(y)).stack()
    ...:     changed = ne_stk[ne_stk]
    ...:     if changed.shape==(0,):
    ...:         print ("no differences found")
    ...:     else:
    ...:         changed.index.names = ['variable', 'value']
    ...:         diff_locate = np.where(df1.round(y) != df2.round(y))
    ...:         changed_from = df1.values[diff_locate]
    ...:         changed_to = df2.values[diff_locate]
    ...:         diff = pd.DataFrame({'from': changed_from, 'to': changed_to}, index=changed.index)
    ...:         return diff

In [30]: compareDF(advs_py6, advs_sas3out, 12)
no differences found

In [31]: compareDF(advs_py6, advs_sas3out, 13)
no differences found

In [32]: compareDF(advs_py6, advs_sas3out, 14)
Out[32]:
                     from        to
variable value
0        CHG_Mean  0.205128  0.205128
```

Comparing Graphical Plot Results

SASPy

```
In [5]: c = sas.submit("""
        proc sgplot data=work.advs;
            where avisitn not in (4.1, 5.1);
          scatter x=avisitn y=chg;
        run;
        """)
        HTML(c['LST'])
```

Python (Seaborn)

```
In [6]: import seaborn as sns
        sns.regplot(x=advs_py2["AVISITN"], y=advs_py2["CHG"])
        sns.plt.show()
```

Rpy2

```
In [12]: import rpy2
         import rpy2.rinterface as rinterface
         import rpy2.robjects as robjects
         from rpy2.robjects.packages import importr
         from rpy2.robjects import Formula
         from rpy2.robjects.vectors import IntVector, FloatVector
         from rpy2.robjects.lib import grid

         rinterface.initr()
         base = importr('base')
         sas7bdat = importr('sas7bdat')
         rprint = robjects.globalenv.get("print")
         stats = importr('stats')
         grdevices = importr('grDevices')
         lattice = importr('lattice')
```

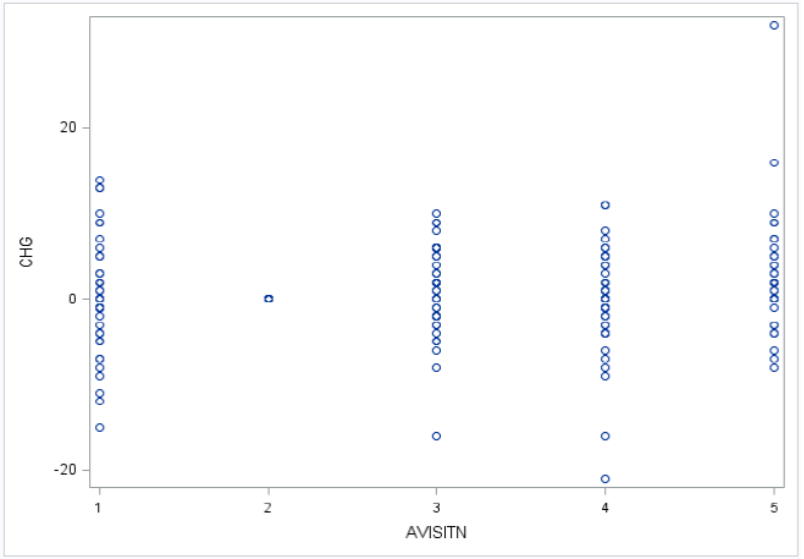
```
In [13]: robjects.r('advs_r <- read.sas7bdat("C:/test.sas7bdat")')
         robjects.r('advs_r2 <- advs_r[advs_r$BASETYPE == "PHASE A",]')
         robjects.r('advs_r3 <- advs_r2[advs_r2$AVISIT != "EARLY1",]')
         r4 = robjects.r('advs_r4 <- advs_r3[advs_r3$AVISIT != "EARLY2",]')
```

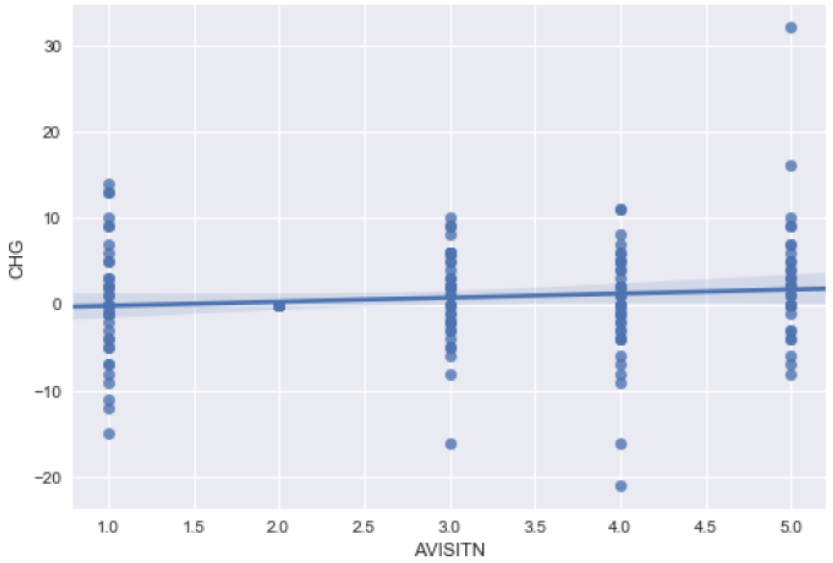
```
In [15]: xyplot = lattice.xyplot
```

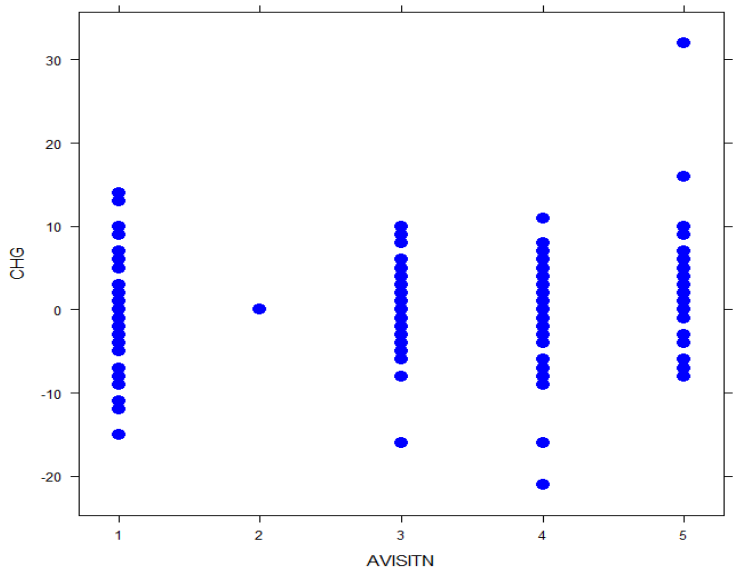
```
In [16]: formula = Formula('CHG ~ AVISITN')
         formula.getenvironment()['AVISITN'] = r4.rx2('AVISITN')
         formula.getenvironment()['CHG'] = r4.rx2('CHG')

         p = lattice.xyplot(formula)
         rprint(p)
```


Comparing Graphical Plot Results







- Plot using Proc SGplot (SASPy)
- Plot using Seaborn (Python)
- Plot using lattice (Rpy2)

Conclusion

- Suggested a method for validating statistical outputs utilising the 3 programming languages SAS, R and Python within a single Python program.
- By converting outputs to dataframes, the function introduced in this paper allows users to perform comparisons at decimal places of their own choice
- Validation continues to be an important aspect of statistical programming

Q&A

References

- SAS Institute. “SASPy”. Accessed January 26, 2018. Available at https://sassoftware.github.io/saspy/getting-started.html
- Gautier, Laurent and rpy2 contributers. “Introduction to rpy2”. Accessed January 26, 2018. Available at http://rpy2.readthedocs.io/en/version_2.8.x/introduction.html
- Les Shay. “Encoding and Decoding Strings (in Python 3.x)”. Accessed January 26, 2018. Available at http://pythoncentral.io/encoding-and-decoding-strings-in-python-3-x/
- Watson, Richann and Johnson, Patty. 2011. “Automated or Manual Validation: Which One is for You?”. *Proceedings of the 2011 PharmaSUG Conference*. Available at https://www.pharmasug.org/proceedings/2011/AD/PharmaSUG-2011-AD01.pdf
- Seabold, Skipper, and Josef Perktold. 2010. “Statsmodels: Econometric and statistical modeling with python.” *Proceedings of the 9th Python in Science Conference 2010*. Available at http://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf

Name: Aik Hoe Seah
Organization: Lundbeck Singapore
Address: 101 Thomson Rd, #14-05 United Square, Singapore
City, State ZIP: 307591
Work Phone: 65-6305-9143
E-mail: aikh@lundbeck.com