

The Best of Three Worlds: Combining IBM® SPSS® Statistics with Your R and Python Code for Extended Algorithms, Ease of Use, and Presentation Output

Jon K. Peck, IBM

Introduction

IBM SPSS Statisticsⁱⁱⁱ is a comprehensive statistical package providing a large number of statistical algorithms, data transformation and management features, a matrix language, graphics, and presentation formats. Although it has an extensive command language in addition to its point and click interface, it was not designed to be a platform for users to develop new statistical or graphical algorithms, although this was possible to some degree. As such, development of new procedures was in the main a task that required SPSS developers to perform.

The system was also limited in its ability to react to results and to use the output from statistical procedures as input to other procedures. In version 12, the Output Management System was introduced to allow all results to be used as data, and a version of the Basic programming language provided some higher level control, but there was little ability to apply programming logic in an integrated way.

Beginning in version 14 of the system, Statistics introduced the ability to use a general purpose, open source programming language to interact with and control the system. In that version, the Python programming language was integrated into the system and provided with a set of APIs that could be used to interact with and control Statistics. Provision for Microsoft .NET languages such as VB.NET and C# to run Statistics and call similar APIs was also introduced, but this can only be used in external mode and will not be discussed in this paper. There is also a C-callable API set made available through a SDK (Software Development Kit).

Beginning in version 16 of Statistics, support for the R language was introduced, providing access to the large library of statistical and graphical packages implemented in that language, and a mechanism for better integration of Python and R programs appeared. In version 17, an integrated tool for easy construction of dialog boxes in the regular Statistics style was added, and in version 18, a packaging mechanism was added to simplify distribution of user-written packages. The SPSS Developer product was introduced to provide an inexpensive platform without most statistical capabilities to facilitate use by customers who wish to supply their own statistical routines but take advantage of the infrastructure capabilities of Statistics.

In the current version, 20, it is therefore possibly and practical for anyone to extend and customize SPSS Statistics using Python and/or R, to provide a point and click interface to these extensions, and to provide traditional Statistics syntax. In short, user-created enhancements appear as if native to the product. This paper will illustrate several applications of these tools and show the basics of how to use these features. By combining the strengths of Statistics with the capabilities of R and Python, custom features can be rapidly developed and made available to any user.

The Python and R Languages

Pythonⁱⁱⁱ is an open source, highly portable, easily integrated, object-oriented language that is both powerful and easy to learn. Invented by Guido van Rossum and now supported by the Python Software Foundation, it appears in the top ten languages listed in the TIOBE^{iv} programming community index. The Python Package Index^v, which is a repository of software for the Python language, currently lists 20672 available packages, of which 1282 are classified as scientific/engineering. The classification system is, however, incomplete. Statistics version 20 is integrated with Python version 2.7

R is "a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc"^{vi} It is widely used in statistics, and CRAN currently lists 3761 available packages. Statistics Version 20 is integrated with R version 2.12. The integrated R version changes with the Statistics release.

The two languages are very different, and both are very different from the traditional Statistics syntax language. Both have excellent facilities for statistical and graphical development, and both are integrated with Statistics, albeit with some differences in functionality. Both are supported by a large community of users and developers. R and most of the contributed packages are distributed under the GNU General Public License. Python is distributed under the less restrictive PSF license^{vii}, which does not require that modifications be made available as open source and does not have the GPL "viral" property.

The Benefits and Drawbacks of Using Python or R with Statistics

Using Python or R with Statistics provides four types of benefits

- Generalization of jobs
- Automation of manual steps
- Extension of capabilities
- Integration with other applications

It is common in organizations using Statistics or other statistical packages to accumulate many versions of basically the same job. By using programmability, similar jobs can be generalized, abstracting away from specific variable names, file locations, and other details.

It is common for a project to spend as much or more time on the result reporting aspects as on the analysis. The ability to inspect and programmatically edit publication quality results means that time consuming but routine parts of the reporting of a statistical analysis can often be done without human intervention.

The ability to read and write the Statistics active dataset, inspect and manipulate the metadata, run arbitrary R and Python code, and write text, pivot tables and charts to the Statistics Viewer means that procedures implemented in such code can take advantage of Statistics built-in capabilities and can behave the same as built-in procedures. In the case of Python, using open source libraries such as numPy and sciPy^{viii}, which are the fundamental Python scientific computing packages, is a powerful aid to algorithm implementation. The Partial Least Squares procedure in Statistics is implemented in Python with the use of these libraries. R, of course, also provides many tools and packages for this purpose.

The ability to inspect the environment for inputs and to route results by various channels to downstream consumers using programmability logic means that integration is easier and more smoothly accomplished.

There are two drawbacks to using the integrated programmability features. Simply because Python and R are different from the Statistics command language, developers need to know more, and users may be confused when encountering different languages. Extension commands and custom dialogs can be used to mitigate this difficulty for users.

The integration requires that Python or R code run in a different computer process from the main Statistics application. While the provided APIs mean that developers need not concern themselves with this, if large volumes of data must be transmitted between the processes using interprocess communication protocols, performance will suffer to a degree. Passing case data through R or Python code is slower than native data passing. The ability to aggregate data or start with results from Statistics helps to mitigate this problem as well as reducing the R memory constraint, but this cannot always be avoided.

How Programmability Works

Python or R code is included in the regular Statistics command stream between BEGIN PROGRAM and END PROGRAM commands. When END PROGRAM is read, the entire program is executed, after which regular Statistics command processing resumes. The program state is preserved from one program to another within a session. For Python, but not for R, external mode can also be used. In that mode, the Statistics functionality is embedded in a Python program, and the normal Statistics user interface does not appear.

A large set of APIs is provided for use by the Python or R code. The Python and R versions are similar, but there are some differences in functionality. The APIs provide for

- reading and writing Statistics datasets
- reading and writing variable metadata
- examining the state of Statistics
- reading Statistics results into Python or R objects
- running Statistics commands (Python only)
- creating Statistics pivot tables
- manipulating Viewer contents (Python only)

R charts and ordinary Python or R print statement output automatically appears in the Statistics Viewer. The R browser function can be used to work interactively in R within the program. Python IDEs that provide for external debugging such as Wingware IDE^x can be useful in developing Python programs.

Detailed documentation for the R and Python APIs is accessible from the Help menu in Statistics, and see this book^{xi}, which can be downloaded in PDF form from the SPSS Community site for an expository treatment.

The examples in the following sections will illustrate these points, focusing particularly on extending Statistics with new procedures. The examples to be shown all use material from the SPSS Community

website (www.ibm.com/developerworks/spssdevcentral) from which these and other useful materials can be downloaded.

R Examples

A simple integrated R program can be written in a few lines of code.

```
begin program r.
print("Hello, JSM")
dta = spssdata.GetDataFromSPSS("jobcat educ salary",
  missingValueToNA=TRUE, factorMode="levels")
summary(dta)
plot(dta$educ, dta$salary)
end program.
```

This program prints a greeting and then reads three variables from the active Statistics dataset and creates an R data frame containing them. The output from print and the summary function appears in the Statistics Viewer along with a scatterplot. Statistics system-missing values are converted to the R NA value, and categorical variables are converted to R factors.

The following program computes a linear model and displays its results as a Statistics pivot table.

```
begin program r.
dta = spssdata.GetDataFromSPSS(c("salary salbegin jobcat prevexp"),
  missingValueToNA=TRUE, factorMode="levels")
res <- lm(salary ~ salbegin + jobcat + prevexp, data=dta, na.action=na.omit)
if (!is.null(res$message)) {
  print(res$message)
  return
}

spsspkg.StartProcedure("My linear model", "LINEAR")
ressum = summary(res)
caption = "put other useful information here"
spsspivottable.Display(coefficients(ressum),
  templateName="SPSSLM",
  caption=caption, isSplit=FALSE)
spsspkg.EndProcedure()
end program.
```

This is the output.

➔ My linear model

[DataSet1] C:\spss20\Samples\English\Employee data.sav

Rtable

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	9,086.494	1,024.206	8.872	.000
salbegin	1.479	.068	21.781	.000
jobcat2	6,900.799	1,614.480	4.274	.000
jobcat3	12,052.724	1,400.426	8.606	.000
prevexp	-24.651	3.622	-6.806	.000

put other useful information here

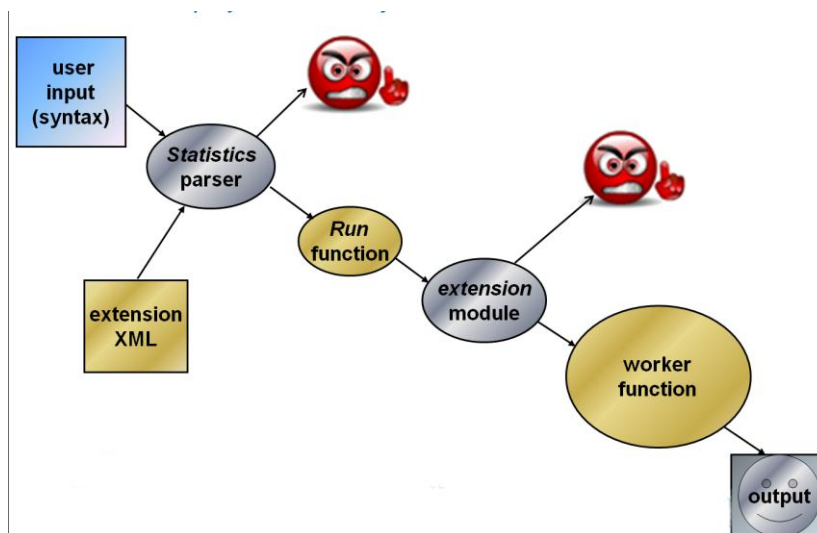
Parameterizing Programs and Creating Extension Commands

To make these programs useful, the inputs need to be parameterized. There are two ways to do this. One is to build a custom dialog box where the parameter values are substituted into the code from the control field values in the dialog. The second and more powerful way is to create a Statistics extension command.

An extension command has three parts

- A syntax definition expressed in a small XML file
- A Run function that receives the parsed syntax and maps it to program variables
- An implementation function that does the work.

The following diagram illustrates the process of executing an extension command



The user input is parsed by the Statistics parser using the xml file that defines valid syntax and is passed, if correct, to the user-written Run function. The Run function uses the extension module provided by Statistics for further checking and mapping to program variables. The worker function does the

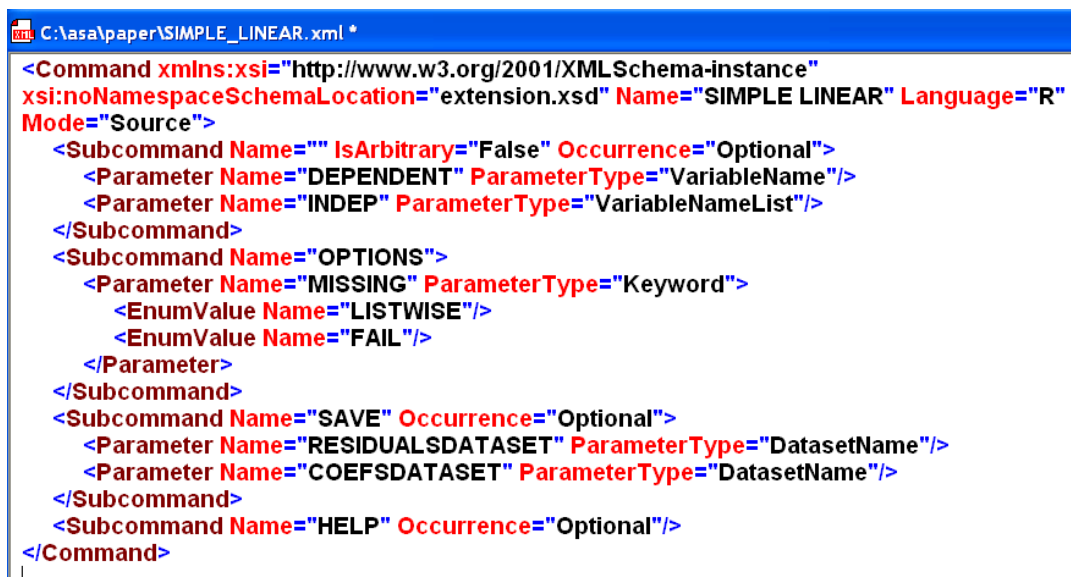
computations and produces the output. Because of the work done by the parser and extension module, the worker code is relieved of most of the need for validating the inputs and can focus on the task.

Statistics commands are generally structured as a command that has subcommands, including an anonymous unnamed subcommand. Subcommands have keywords that take values of various types. For the simple linear model example above, one might design the following extension command syntax.

```
SIMPLE LINEAR DEPENDENT=varname INDEP = list-of-varnames
/OPTIONS MISSING=LISTWISE* | FAIL
/SAVE RESIDUALSDATASET=datasetname COEFSDATASET=datasetname
/HELP.
```

where all of the explicit subcommands are optional, and the missing value treatment defaults to LISTWISE. The first line implicitly contains the anonymous subcommand with the DEPENDENT and INDEP keywords. The SAVE subcommand can specify a dataset to be created containing the residuals and another containing the estimated coefficients. The HELP subcommand will display some help text and do nothing else.

This structure is expressed in a small extension XML file that looks like this.



```
C:\asa\paper\SIMPLE_LINEAR.xml *
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="extension.xsd" Name="SIMPLE LINEAR" Language="R"
Mode="Source">
  <Subcommand Name="" IsArbitrary="False" Occurrence="Optional">
    <Parameter Name="DEPENDENT" ParameterType="VariableName"/>
    <Parameter Name="INDEP" ParameterType="VariableNameList"/>
  </Subcommand>
  <Subcommand Name="OPTIONS">
    <Parameter Name="MISSING" ParameterType="Keyword">
      <EnumValue Name="LISTWISE"/>
      <EnumValue Name="FAIL"/>
    </Parameter>
  </Subcommand>
  <Subcommand Name="SAVE" Occurrence="Optional">
    <Parameter Name="RESIDUALSDATASET" ParameterType="DatasetName"/>
    <Parameter Name="COEFSDATASET" ParameterType="DatasetName"/>
  </Subcommand>
  <Subcommand Name="HELP" Occurrence="Optional"/>
</Command>
```

The command name is SIMPLE LINEAR; it is implemented in R source form. It takes a single dependent variable and a list of independent variables. The possible keyword types are described in the schema installed with Statistics and provided at <http://www.ibm.com/software/analytics/spss/xml/> and in the online documentation. The types include VariableName, VariableNameList, DatasetName, Integer, Number, Keyword, QuotedString, InputFile, OutputFile and others. The structure of the XML file is the same for both Python and R. For R, the mode can be Source or Package, which refers to a compiled R package.

When the command is executed, the Statistics parser first compares the user input with the definition in the XML file. Any violations cause the command execution to be halted and an appropriate error message to be displayed.

Otherwise, the command parameters are packaged into an appropriate Python or R structure, and the Run method of the implementation file, which must match the command name, is called with that structure as the parameter. All installed extension commands are automatically available in a session without any user action.

The Run function for this command adds details to the syntax definition and prepares the input for execution by the target function. For SIMPLE LINEAR, the R code looks like this.

```
Run<-function(args){
  args <- args[[2]]
  oobj<-spsspkg.Syntax(templ=list(
    spsspkg.Template("DEPENDENT", subc="", ktype="existingvarlist", var="dep", islist=FALSE),
    spsspkg.Template("INDEP", subc="", ktype="existingvarlist", var="indep", islist=TRUE),
    spsspkg.Template("MISSING", subc="OPTIONS", ktype="str", var="missing"),
    spsspkg.Template("RESIDUALSDATASET", subc="SAVE", ktype="literal", var="residualsdataset"),
    spsspkg.Template("COEFSDATASET", subc="SAVE", ktype="literal", var="coefsdataset")
  ))
  if ("HELP" %in% attr(args, "names"))
    writeLines(helptext)
  else
    res <- spsspkg.processcmd(oobj, args, "simpleLinear")
}
```

This specification adds the information that the variable names have to be not only syntactically valid as variable names but that they have to appear in the active dataset (ktype="existingvarlist"). The parameter type can be bool (values of "true", "false", "yes", or "no"), str (string or quoted literal), int (integer with optional range), float (number with optional range), varname (arbitrary, syntactically valid variable or dataset name), or existingvarlist (names of variables that exist in the dataset.).

processcmd also maps each input, either as a list or a single value, to a parameter of the implementing function in R or Python – simpleLinear in this case. The implementing function is introspected to determine whether each parameter is required or optional, and an error is raised if a required parameter is not supplied. The call to spsspkg.processcmd validates the inputs, maps the variables appropriately, and calls the target function. The Run function handles the display of the help text.

At the point where the implementing function receives control, all of the input specifications have been checked, so the function can assume that its input is valid. In some cases, additional validation would need to be carried out by that function for conditions that cannot be expressed in the XML or Run specification. Error messages that appear prior to the worker function automatically appear in the current Statistics output language, and there is a mechanism available to worker functions to display translated output.

The implementing function is the same as it would be if it were not an extension command, but it can assume valid input. If the implementing function terminates by raising an error condition, the error

message is automatically turned into a Statistics Warnings object that is displayed in the Viewer. We show here only the function signature for this example.

```
simpleLinear<-function(dep, indep, missing="listwise", residualsdataset=NULL, coefsdataset=NULL)
```

The dep and indep arguments are required while the other three have default values.

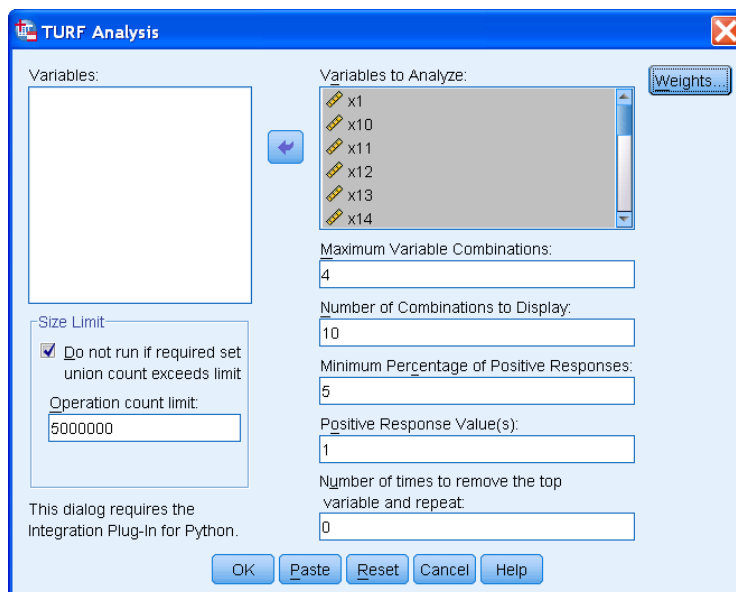
A Python Example: TURF

TURF, Total Unduplicated Reach and Frequency, is a popular technique in market research. Given a survey of preferences, it finds the combination of a specified number of variables in a survey that gives the best coverage (reach) of positive responses. Examples include

- Which frozen yoghurt flavors to offer in shops that have three dispensing machines to maximize the chance that a visitor to the shop will like at least one flavor
- What sports events to advertise on to maximize coverage of sports viewers
- Where to locate a set of repeated regional conference offerings to maximize the number of attendees who will attend one of them.

While conceptually simple, TURF is computationally expensive, since all possible combinations of variables must be tried, and the number of combinations grows rapidly with the number of choices. In Statistics TURF is implemented as the SPSSINC TURF extension command using Python. First we will show the user interface and output for a set of yoghurt flavors and then discuss how this was implemented.

The package includes a custom dialog box built with the Statistics Custom Dialog Builder. The CDB provides a no-programming, drag and drop tool for building dialog boxes that can be added to Statistics and appear in other SPSS products as well.



The extension command syntax for this procedure looks like this.


```
SPSSINC TURF VARIABLES = x1 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x2 x20 x3 x4 x5 x6 x7 x8 x9
/OPTIONS BESTN = 4 NUMBERTODISPLAY = 10 THRESHOLD = 5 CRITERIA = 1
REMOVALCYCLES = 0. MAXSETS=5000000
/IMPORTANCE STRICT=YES.
```

Here is some of the output.

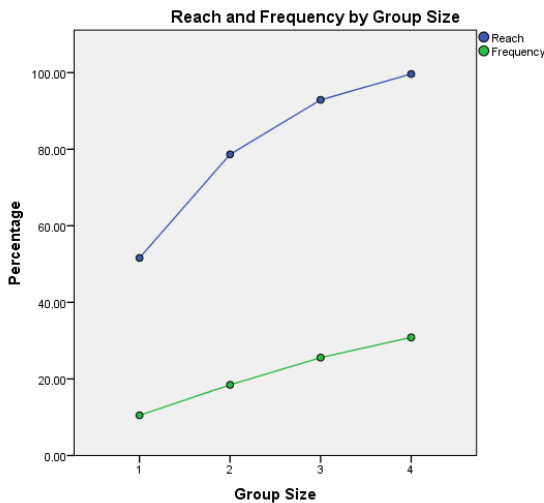
Maximum Group Size: 4. Reach and Frequency.

Variables	Statistics			
	Reach	Pct of Cases	Frequency	Pct of Responses
x11, x12, x17, x2	280	99.6	426	30.8
x11, x12, x17, x18	274	97.5	475	34.4
x11, x12, x18, x20	272	96.8	442	32.0
x11, x12, x17, x19	272	96.8	440	31.9
x11, x12, x18, x19	271	96.4	464	33.6
x11, x12, x14, x18	270	96.1	476	34.5
x11, x12, x17, x4	269	95.7	422	30.6
x10, x11, x12, x17	268	95.4	387	28.0
x11, x12, x14, x17	267	95.0	452	32.7
x11, x12, x18, x6	267	95.0	436	31.6

Variables: x1, x10, x11, x12, x13, x14, x15, x16, x17, x18, x19, x2, x20, x3, x4, x5, x6, x9

Best Reach and Frequency by Group Size

Variables	Statistics				
	Group Size	Reach	Pct of Cases	Frequency	Pct of Responses
ADDED: x12	1	145	51.6	145	10.5
ADDED: x11 KEPT: x12	2	221	78.6	255	18.5
ADDED: x17 KEPT: x11, x12	3	261	92.9	353	25.6
ADDED: x2 KEPT: x11, x12, x17	4	280	99.6	426	30.8



The XML syntax definition indicates that the command is implemented in Python, and that it has an anonymous main subcommand with one keyword, and OPTIONS, IMPORTANCE, and HELP subcommands. Keyword types used include VariableNameList, Integer, Number, OutputFile, and TokenList. A TokenList is a more general form of a keyword list.

```
<Command xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="extension-1.0.xsd" Name="SPSSINC
TURF" Language="Python">
  <Subcommand Name="" IsArbitrary="False" Occurrence="Optional">
    <Parameter Name="VARIABLES" ParameterType="VariableNameList">
  </Subcommand>
  <Subcommand Name="OPTIONS">
    <Parameter Name="BESTN" ParameterType="Integer">
    <Parameter Name="MINTABLE" ParameterType="Integer">
    <Parameter Name="NUMBERTODISPLAY" ParameterType="Integer">
    <Parameter Name="THRESHOLD" ParameterType="Number">
    <Parameter Name="CRITERIA" ParameterType="NumberList">
    <Parameter Name="REMOVALCYCLES" ParameterType="Integer">
    <Parameter Name="MAXSETS" ParameterType="Integer">
    <Parameter Name="LOGFILE" ParameterType="OutputFile">
    <Parameter Name="LOGFREQ" ParameterType="Integer">
  </Subcommand>
  <Subcommand Name="IMPORTANCE" Occurrence="Optional">
    <Parameter Name="IWEIGHTS" ParameterType="TokenList">
    <Parameter Name="STRICT" ParameterType="Keyword">
  </Subcommand>
  <Subcommand Name="HELP" Occurrence="Optional">
</Command>
```

Statistics will call the Run function in SPSSINC_TURF.py. That function augments the Statistics parser specification with this code.

```
oobj = Syntax([
  Template("VARIABLES", subc="", ktype="existingvarlist", var="variables", islist=True),
  Template("BESTN", subc="OPTIONS", ktype="int", var="bestn", vallist=[1]),
  Template("MINTABLE", subc="OPTIONS", ktype="int", var="mintable", vallist=[1]),
  Template("NUMBERTODISPLAY", subc="OPTIONS", ktype="int", var="number", vallist=[1]),
  Template("THRESHOLD", subc="OPTIONS", ktype="float", var="threshold"),
  Template("CRITERIA", subc="OPTIONS", ktype="int", var="criteria", islist=True),
  Template("REMOVALCYCLES", subc="OPTIONS", ktype="int", var="removalcycles", vallist=[0]),
  Template("MAXSETS", subc="OPTIONS", ktype="int", var="maxsets"),
  Template("LOGFILE", subc="OPTIONS", ktype="literal", var="logfile"),
  Template("LOGFREQ", subc="OPTIONS", ktype="int", var="logfreq", vallist=[1]),
  Template("IWEIGHTS", subc="IMPORTANCE", ktype="str", var="iweights", islist=True),
  Template("STRICT", subc="IMPORTANCE", ktype="bool", var="strict"),
  Template("HELP", subc="", ktype="bool")])
```

This code also defines value checking through the vallist keyword. The BESTN parameter, for example, has a minimum value of 1 – show at least the best one variable. Three of these parameters are lists. The definition code is nearly identical to the equivalent code in R.

The final validation, variable mapping, and invocation of the computational module is specified by this code.

```
processcmd(oobj, args, turf, vardict=spssaux.VariableDict())
```

The turf function definition begins with this.

```
def turf(variables, bestn, number, threshold=0, criteria=None, removalcycles = 0,
         maxsets=None, mintable=1, logfile=None, logfreq=500000, iweights=None, strict=True):
```

The first three arguments are required while all others have default values. Again, introspection is used to determine whether all required arguments were supplied. Because of the way that Python defines

functions, certain defaults, for example, the criteria default, which is a list of values, are specified in the definition as None and created in the code.

Altogether, the SPSSINC TURF implementation is 900 lines of code, including comments, docstrings, and help text. The original version, which was about half the current size, was produced in only two days, including the dialog box interface, the syntax definition, the code implementation, and the help. Working with these tools enables a developer to be extremely productive.

Python is a scripting language and is generally slower than would be compiled code in languages such as C and Java, but Python, like R, allows C code to be integrated. For a computationally intensive procedure such as TURF, this could be a disadvantage, but the largest part of the time goes into calculating set unions and other set operations. These operations are built into the Python language and are implemented in C code. TURF builds a set of positive responses for each question. The set members are the positive response case ids. The set unions for all possible question combinations of a given size are computed for each size up to the requested number of variables. For the "best four" example above, this requires 6,175,000 set union operations. Doubling to the "best eight" variables requires 263,929,00 set union operations, so this work and calculating the set sizes quickly comes to dominate the execution time. This larger problem executed in ten seconds on a Lenovo W500 laptop computer.

Python or R?

For many purposes either Python or R can be used, and the choice may be based on personal preference and knowledge, organizational constraints, licensing restrictions, or the existence of specific packages and tools. There are also differences in functionality with SPSS Statistics stemming from the different roles intended for these two integrations. Python was originally implemented primarily for controlling and interacting with Statistics and secondarily for extending functionality. R, on the other hand, was implemented primarily for extending the statistical and graphical capabilities of Statistics. Thus using Python one can build and run Statistics syntax while this is not possible with R. Python also provides a set of apis in the SpssClient module that can access and manipulate objects in the Statistics Viewer and in the user interface. These APIs do not have an R equivalent. With either language, however, the Output Management System (OMS) can be used to write Statistics output objects to an in-memory XML workspace or as a new dataset, and both of these are accessible from either language. Python can be run in either external or internal mode while R can only be run in internal mode.

Packaging and Distribution

A complete package generally consists of a custom dialog box, a syntax definition, and Python or R code files. It may also include translation files. Combining these into an Extension Bundle via the Statistics Utilities >Extension Bundles > Create Extension Bundle menu makes it easy for a user to install the package without concern about the individual pieces. Any required R packages are downloaded at installation time if possible. Using this mechanism also facilitates managing bundles as there is a standard way to see what bundles are installed and view their descriptions and dependencies. Since the installation process for a bundle consists only of copying files, it is possible to do a push install for high volume distribution of a package and the prerequisite Python or R plugins.

Conclusion

Using IBM SPSS Statistics in combination with R and Python provides these advantages

- High developer productivity due to the ability to use the Statistics infrastructure for data acquisition and manipulation, output management, graphics and other facilities while coding in R or Python
- Ease of development of robust packages due to the extension command mechanism for validation of input and error handling
- Ease of producing publication quality output and exporting in various formats
- Ease of producing a user interface consistent with the general appearance of Statistics using the Custom Dialog Builder and the extension mechanism
- Ease of distributing packages to SPSS users via the Extension Bundle Mechanism

All of the materials discussed here except for SPSS Statistics itself are free and available from the SPSS Community website at www.ibm.com/developerworks/spssdevcentral.

ⁱ Older versions were known as SPSS or PASW Statistics. This paper will use the current name when referring to any version.

ⁱⁱ <http://www-01.ibm.com/software/analytics/spss/products/statistics/>

ⁱⁱⁱ <http://www.python.org/>

^{iv} <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

^v <http://pypi.python.org/pypi>

^{vi} <http://cran.r-project.org/>

^{vii} <http://docs.python.org/license.html>

^{viii} <http://numpy.scipy.org/>

^{ix} "NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more."

"SciPy is an Open Source library of scientific tools for Python. It depends on the NumPy library, and it gathers a variety of high level science and engineering modules together as a single package. SciPy provides modules for

statistics
 optimization
 numerical integration
 linear algebra
 Fourier transforms
 signal processing
 image processing
 ODE solvers
 special functions"

^x <http://wingware.com/>

^{xi} Programming and Data Management