SESUG Paper VR-39-2025

Introducing Automatic Graphics from Python Models in SAS®

Dan Heath, SAS Institute Inc.

ABSTRACT

SAS users have the ability to automatically produce graphics with their tabular output from SAS analytic procedures using the ODS Graphics system. Many of these procedures provide options for controlling the content and appearance of these graphics. In addition, the ODS system provides statements for controlling what output is displayed, as well as a means for controlling the order of that output.

SAS Viya and SAS Workbench users have had these same capabilities when running SAS programs, but not when running Python programs — until now. Python users can now have graphics generated automatically from models that have graphics support. In addition, models now contain a new method called customize_results() that gives you the ability to control aspects of the graphics output, as well as filter tabular results and control result output order — all without rerunning the model computations.

INTRODUCTION

SAS users have enjoyed the benefits of having SAS procedures generate automatic graphics using the ODS Graphics System. Currently, there are over 125 procedures that use this system to automatically produce graphics applicable to the requested analysis. These results let you focus on the analysis without the distraction of writing code to create custom graphics from the output data.

In SAS Viya Workbench and SAS Viya 4, users can now call some of the same models used by SAS procedures using open-source languages, such as Python. However, if they want to visualize any results, they must write code using another package, such as Matplotlib or Seaborn.

Now, SAS has released the ability for Python-based SAS models to automatically produce graphics. As of this writing, this support is added to the following models:

- PCA – Principal component analysis
- LogisticRegression – Logistic regression classifier
- Lasso – Linear regression with L1 regularization
- SVC – Support vector classification model

In addition, every model now supports the new customize_results() method. This method can be used to generate customized graphical and tabular output. Graphical options are ignored for models that do not currently support graphics.

In the following sections, you will learn more about displaying automatic results and about environment variables use can use to affect them. You will also learn more about the customize_results() method and how you can use it to generate custom output from the original analysis. Also, I will discuss these capabilities primarily using the PCA model to focus more on the capabilities of the system rather than the plot types generated by each model.

AUTOMATIC GRAPHICS SUPPORT

In the code snippet below, A principal component analysis is performed on a fitness data set for six factors of fitness:

```
t = Table().load("fitness.csv", datalib=d)
factors = ["weight","oxy","runtime","rstpulse","runpulse","maxpulse"]
X_train = t[factors]
model = PCA()
model.fit(X_train)
```

At this point, both the graphical and tabular result are created, but not displayed. To display the results in Workbench using a Jupyter Notebook, run the following line after the fit() call:

```
display(model.details_)
```

When using a SAS Notebook or the PYTHON procedure in Viya 4, you should use the following call to display the results:

```
SAS.showMLA(model.details_)
```

An Eigenvalue plot (Figure 1), a pattern profile plot (Figure 2), and ten pattern plots (Figure 3, for example) are displayed, along with the tables.

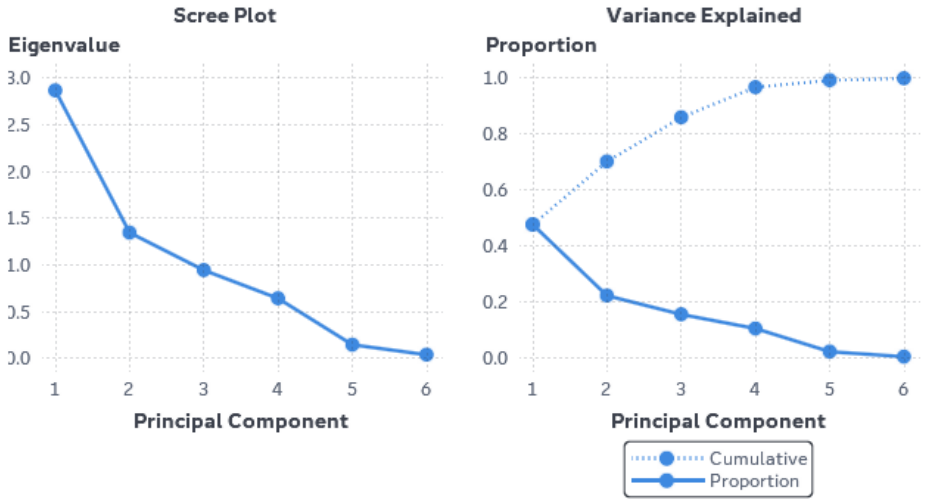

Figure 1. Eigenvalue Plot

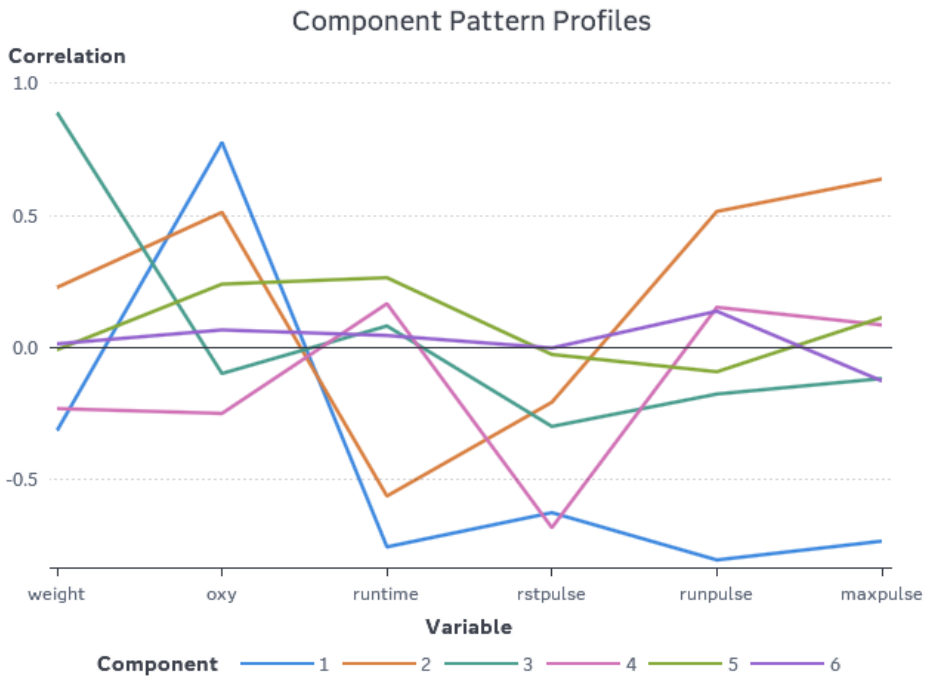

Figure 2. Pattern Profile Plot

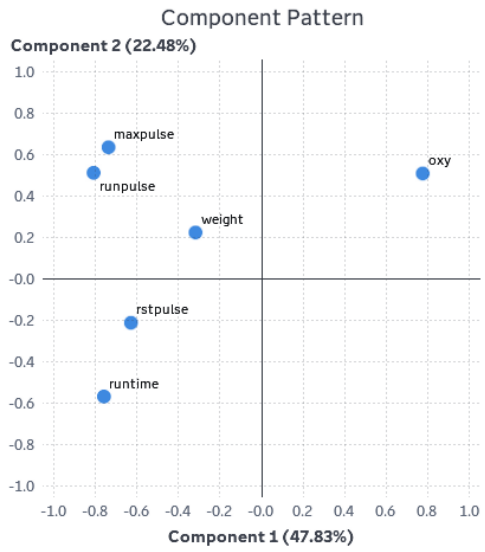

Figure 3. Pattern Plot

There are two environment variable that can be used to control aspects of the automatic graphics system:

- AUTO_GRAPHICS= true | false
- GRAPHICS_THEME=light | dark | highcontrast | opal | midnight | raven | htmlencore | grayscale | monochrome

To set these values in a Jupyter Notebook, use %set_env:

```
%set_env GRAPHICS_THEME=dark
```

In a SAS Notebook or the PYTHON procedure, you need to use the standard Python method:

```
os.environ["GRAPHICS_THEME"] = "dark"
```

The AUTO_GRAPHICS variable controls whether graphics are produced automatically. However, this variable has no impact on graph creation when using the customize_results() method. The default is to automatically produce graphics.

The GRAPHICS_THEME variable controls which theme is used to create graphics and can be useful for aligning the plot appearance with the rest of the report appearance. This variable affects both automatic graphics and graphics created with customize_results(). In addition, the customize_results() method has a parameter called “theme” which will override the environment theme. The default theme is “light”.

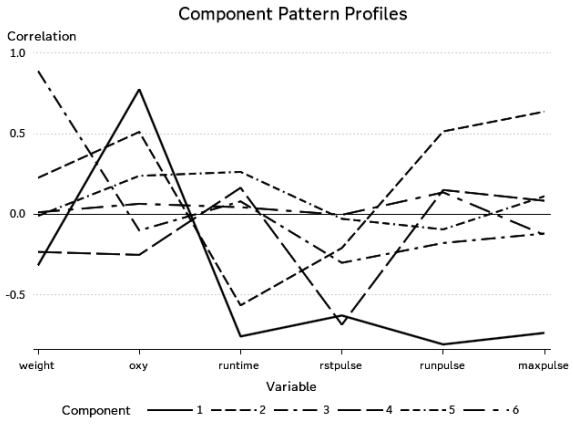

Figure 4. Pattern Profile Plot Using the Monochrome Theme

CUSTOMIZE RESULTS

Many times, you will want only a subset of the results to show in the notebook. You also might want to reorder the output or make customizations to the graphics. In these cases, you will want to use the customize_results() method to apply the desired changes, which will return a new result set that can be displayed using the statements described earlier. This method does NOT rerun the analysis – it uses the original result set to generate new results with the desired customizations.

The previously discussed display() methods can be used to display the results from the customize_results() method. For example:

```
display(model.customize_results(…))
```

The display() call is not shown in the examples below to focus on the customize_results() options.

GRAPHICS OPTIONS

The PLOTS parameter is the primary parameter for controlling what plots are rendered and what plot options are applied. This option is like the PLOTS option on many SAS statistical procedures; however, there are some important differences:

- In a SAS procedure, the list of valid plot types is specific that that procedure; but the PLOTS parameter must support plot types across multiple models. Therefore, the list of valid plot types will be specific to the model used. If an invalid plot type is specified for a model, it is just ignored. Consult the documentation for which plot types are supported by each model.
- While the PLOTS parameter will share many of the same sub-options as the PLOTS option in the SAS procedures, it does not support all of them. The PLOTS parameter might also support options not in the PLOTS option. Consult the documentation for what sub-options are supported for each plot type.

The general form of the option is the following:

```
plots="PlotType<(sub-option1, ..., sub-optionN)>" |
       ["PlotType1<(sub-option1, ..., sub-optionN)>", ...,
       "PlotTypeN<( sub-option1, ..., sub-optionN)>"]
```

The plot type can also be either the NONE keyword or the ALL keyword. The NONE keyword is used to suppress all graphics output in the customized results. This is useful for when you want custom results containing only tables. This case will be discussed further in the “table options” section. The ALL keyword will cause all possible plot types to be generated. Any sub-options on the ALL type will be passed to all plot types to be generated. If the plot type does not support the sub-option, the sub-option is just ignored for that type.

To illustrate the interaction of plot types and sub-options, consider the following two specifications for the PCA model:

- `plots="eigen(unpack)"`
- `plots="all(unpack)"`

The UNPACK sub-option requests for paneled plot be split into separate plots if possible. For the PCA model, only the Eigenvalue plot supports the UNPACK sub-option. In the first specification, I am requesting to generate only the Eigenvalue plot, but generate it unpacked (see Figure 5). In the second specification, I’m requesting that all plots be generated, and unpack any plots that support the sub-option. The result of that request is that I will get all the plots seen in the automatic result, with the exception that the Eigenvalue plot will be split into two plots.

It also possible to generate selected plots from an unpacked panel using name indexing. Consider the following requests:

- `plots="eigen(scree)"`
- `plots="eigen(variance)"`
- `plots="eigen(scree, variance)"`

Referring again to Figure 5, the first specification will generate only the left plot, the second the right plot, and the third both plots. The third result is the same as with the UNPACK sub-option in this case, but there are other cases where multiple name indexing be useful, such as with this criterion panel from the Lasso model (Figure 6).

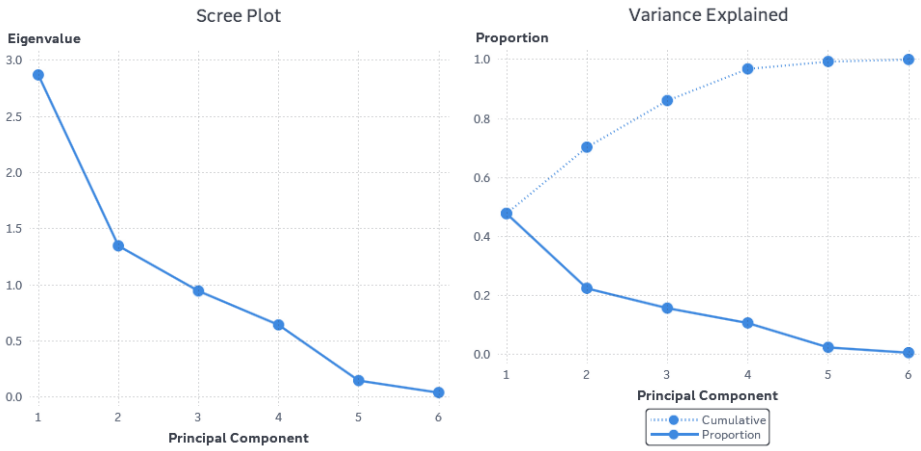

Figure 5. Eigenvalue Plot Split into Scree and Variance Plots

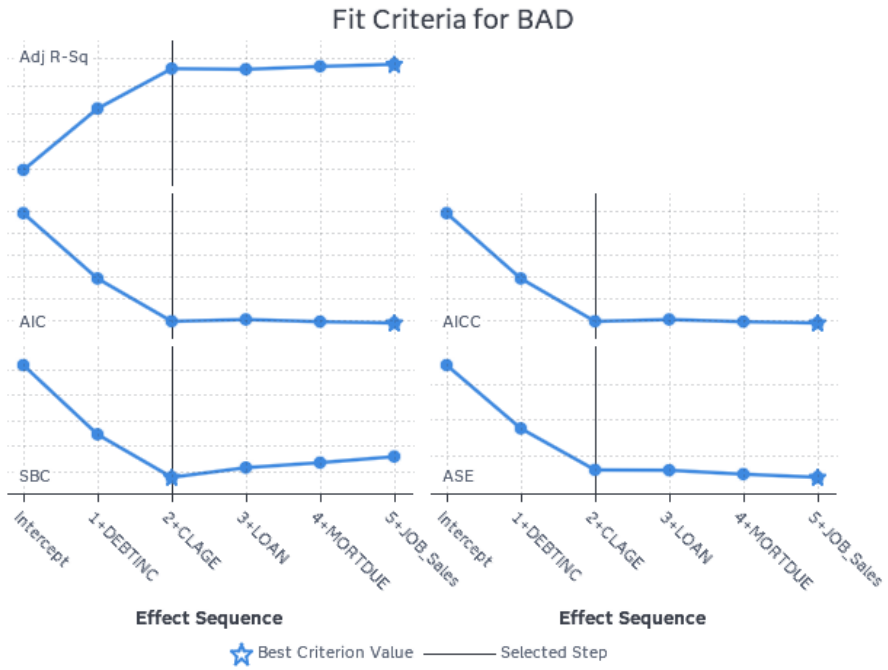

Figure 6. Criterion Panel from a Lasso Model

Some models have sub-options that can control the content of their plots. A good example of this support is in the Pattern plot for the PCA model. Consider the following request:

```
plots="pattern(vplot, circlelabel4)"
```

By default, the data for the Pattern plot is shown as a scatter plot. In this request, the VPLOT sub-option has the plot rendered as a vector plot instead of a scatter plot. The CIRCLELABEL4 option adds four reference circles on the plot with labels to better determine how much the data contributes to the principal components. The model supports up to five reference circles, both with labels (CIRCLELABEL1-CIRCLELABEL5) and without (CIRCLE1-CIRCLE5). See Figure 7 for comparison of the default output and this plot request.

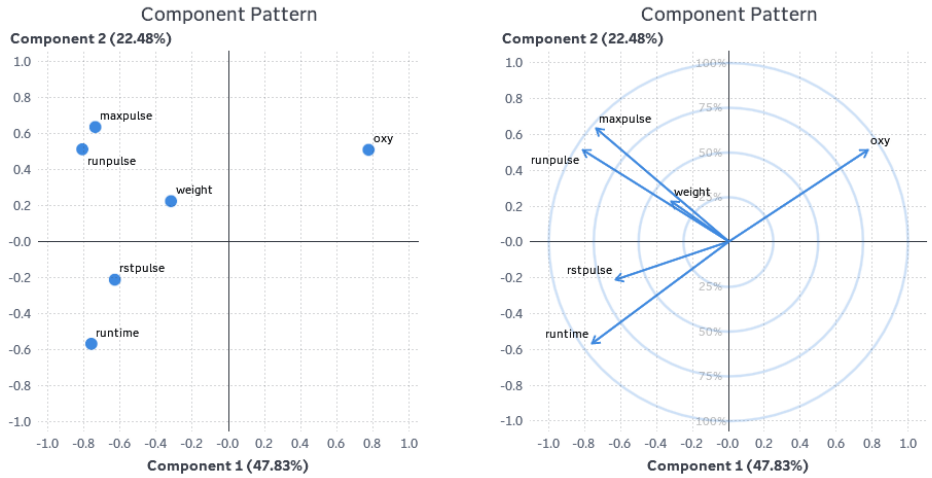

Figure 7. Default Pattern Plot and a Vector-based Pattern Plot with Reference Circles

The default number of pattern plots generated can quickly grow as you add more variables to the analysis. There are two sub-options for the Pattern plot to control the number of plots generated:

- `ncomp=<int>` -- Plot the first <int> number of principal components
- `index=<int> | [<int1>,...,<intN>]` – Generates plots based on the specified 1-based indices

To further elaborate on the INDEX sub-option, the original code snippet at the beginning for the paper generates ten pattern plots. If I were interested in the first and fourth plots, I can have customize_results() generate only those two plots, and add reference circles, by using the following PLOTS sub-option specification (Figure 8):

```
plots="pattern(index=[1,4], circlelabel5)"
```

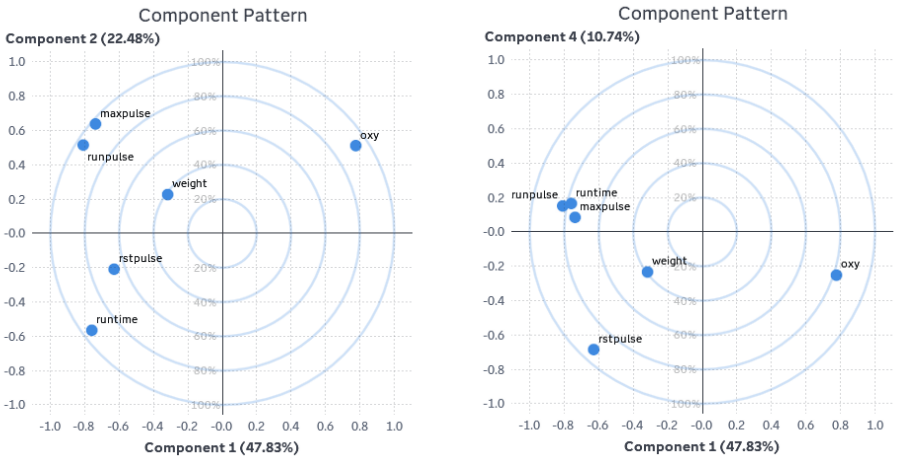

Figure 8. The First and Fourth Pattern Plots with Reference Circles

EXPORTING GRAPHICS

In addition to writing the graphics output into notebooks, customize_results() can export graphics output to a directory using the PATH parameter. When the path is specified, all graphics generated by the customize_results() request are written to that directory as well as the notebook. The filenames used are created by the result name and any required indexing.

The OUTPUT_TYPE parameter controls the type of graphics (either “svg” or “png”). The default output type is “svg”, which also produces data tip information which can be viewed by mousing over the plot data in the notebook. When exporting graphics as SVG, that data tip information will be saved with the file so that it can be viewed in another application, such as a web browser. In most cases, I would recommend “png” output, as it will guarantee the correct font appearance regardless of the system on which it is viewed, and the PNG file can be readily imported into almost any document or application. All the standalone graphs in this paper were generated by setting PATH and setting OUTPUT_TYPE=”png”.

Other parameters that are useful for exporting graphs include the following:

- width – Specifies the width of the graphics output in pixels.
- height – Specifies the height of the graphics output in pixels.
- dpi – Specifies the resolution of PNG image output. This default is 96.

Suppose you wanted to include an Eigenvalue plot in a report you are creating in Microsoft Word. You wanted this image to be 500px wide at 200dpi using the monochrome style. Your customize_results() call would look something like the following:

```
model.customize_results(plots="eigen", width=500, dpi=200,
                        path="/workspace/output", theme="monochrome",
                        output_type="png", include_tables="none")
```

The result of this call is in Figure 9. The INCLUDE_TABLES option will be discussed later in the “table options” section.

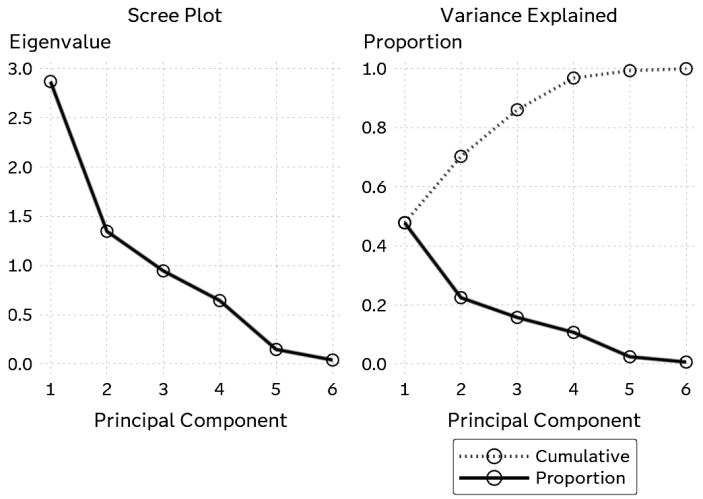

Figure 9. Exported Eigenvalue Plot

RESULT NAMES

Before discussing the table options and the option for ordering output, it is important to understand how to determine the name of each result.

§ NVars

Number of Variables

Description	Value
Number of Variables	6
Number of Principal Components	6

§ NObs

Number of Observations

Description	Value
Number of Observations Read	31.0
Number of Observations Used	31.0

Figure 10. NVars and NObs Table Output from Jupyter Notebook

Figure 10 contains a screenshot of the NVars and Nobs tables generated from the model fit. The name of the result is in the top-left corner. This name also appears for plot output; however, for plot output, multiple plots can be bound to the same name (the Pattern plot is a good example of this). Using the Pattern plot as an example, any operation using that result name will affect the entire group of plots. For example, if you reorder the output by name, the entire group of Pattern plots will move relative to the other results in the order list. I will discuss ordering results in a later section.

For SAS Notebooks and the PYTHON procedure, these names do not appear automatically. To make the names appear, use the SHOW_NAME parameter on the showMLA() method:

```
SAS.showMLA(model.details_, show_name=true)
```

The name will appear under the title of the table, or in the title location if there is no title. Typically, you would not use the SHOW_NAME parameter for your final output.

TABLE OPTIONS

In BASE SAS, there is an ODS SELECT statement and an ODS EXCLUDE statement. ODS SELECT renders only the object names on the statement and excludes all others. ODS EXCLUDE excludes all object names on the statement and renders all others. The INCLUDE_TABLES and EXCLUDE_TABLES parameters on the customize_results() method work in similar way for tabular results. The parameters take both result names and keywords:

- `include_tables = "all" | "none" | "<ResultName>" | ["<ResultName1>", ..., "<ResultNameN>"]`
- `exclude_tables = "all" | "none" | "graph_tables" | "<ResultName>" | ["<ResultName1>", ..., "<ResultNameN>"]`

The default behavior for customize_results() is to generate all tabular results (equivalent to include_tables=”all” or exclude_tables=”none”). If you have a situation where you want only graphs generated (as with the previous export example), specify either include_tables=”none” or exclude_tables=”all” to suppress all tables.

When you examine a complete result from model.details_, you will find tables that contain data that was used as input for creating graphs. You might want these tables dropped from the results, since the graphs also represent the data. To do this, simply specify exclude_tables=”graph_tables”, and all of those tables will be dropped.

Both the INCLUDE_TABLES and EXCLUDE_TABLES parameters let you include and exclude tables results by name. Referring again to Figure 10, if I only wanted to see those two tables, I would use the following specification:

```
model.customize_results(plots="none", include_tables=["NVars", "NObs"])
```

The names are case-sensitive. If you do not see a specified table included or excluded, be sure the case is correct for all characters in the name.

ORDER OPTION

In BASE SAS, you can have your output generated into an ODS Document and use the DOCUMENT procedure to replay pieces of output in any order you want. With the customize_results() method, you can use the ORDER parameter to generate results in any order you want.

The ORDER parameter takes a list of result names, as shown in the previous INCLUDE_TABLES example. The difference with ORDER is that the result names can be both graphs and tables. For example, I want to generate a result that contains only the Eigenvalues plot and the Eigenvalues table; but I want the table to come before the plot, which is different from the original result. The request would look like the following, with the result shown in Figure 11:

```
Model.customize_results(order=["Eigenvalues", "EigenvaluePlot"])
```

§ Eigenvalues

Eigenvalues

Number	Eigenvalue	Difference	Proportion	Cumulative
1	2.870008	1.521426	0.478335	0.478335
2	1.348582	0.401974	0.224764	0.703098
3	0.946608	0.301928	0.157768	0.860866
4	0.644680	0.495977	0.107447	0.968313
5	0.148704	0.107286	0.024784	0.993097
6	0.041418	NaN	0.006903	1.000000

§ EigenvaluesPlot

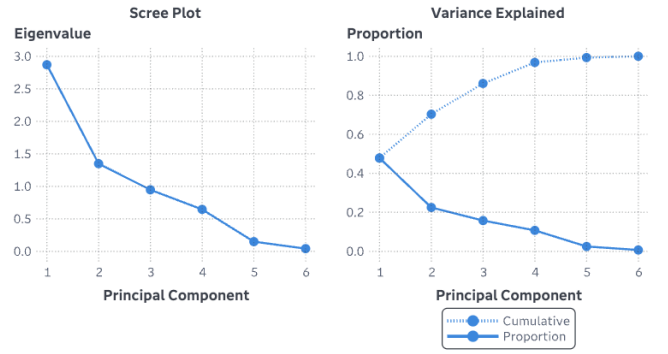

Figure 11. Table and Plot in Different Order

If other parameters cause a result from being generated, the ORDER parameter will ignore it in the list. For example, if the request were changed to the following, only the Eigenvalues table would appear because I requested that only the “profile” plot be generated:

```
Model.customize_results(plots="profile",
                        order=["Eigenvalues", "EigenvaluePlot"])
```

CONCLUSION

We have explored the new Python graphics functionality using the PCA model. I would encourage you to also explore these capabilities using the other models. As mentioned earlier, each model has different plot types and options, so be sure to consult the documentation to see what is available. The list of models supporting graphics will also continue to grow, so be sure to check the documentation when you upgrade.

RECOMMENDED READING

- *SAS® Viya®: Managing Plots in Python Automatic Graphics*
- *SAS® Viya® Workbench: Machine Learning Python API Guide*

CONTACT INFORMATION

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

Dan Heath
SAS Institute Inc.
Dan.Heath@sas.com