Better together: Extending JMP® with open-source software

Nascif Abousalh-Neto | Meijian Guan | Ruth Hummel

JMP Division, SAS Institute, Cary, NC, 27513, USA
Correspondence
Ruth Hummel, JMP Division, SAS Institute, Cary, NC 27513, USA.
Email: ruth.hummel@jmp.com

JMP is commercial software designed for interactive data analysis and exploration. JMP's high-level, visual interface makes it an outstanding tool for teaching best practices, methods and model building techniques. JMP is also designed for extensibility, with features that allow the embedding of and deployment to open-source packages and environments. In this paper, we will explore use cases that illustrate how JMP users can take advantage of the broad support and rapid pace of change of open-source software while still benefiting from JMP's streamlined interface and robust capabilities.

KEYWORDS
data science, JMP, Python, R, software, teaching, visualization

1 | MOTIVATION

JMP, a statistical discovery application from SAS Institute (SAS Institute, n.d.), offers a highly integrated, easy-to-use user interface based on dialogues, menus, and drag-and-drop commands. JMP provides a comprehensive set of data access, cleaning, visualization, analysis, and modelling features. JMP allows customers to stay focused on their analytics workflow without any programming requirements.

In some scenarios, this streamlined workflow might not be sufficient. For example, the final step of a data modelling workflow might require that the model is exported from JMP so that it can be deployed to a production environment. Or perhaps the data from a particular domain might be better suited to a modelling technique that is not yet implemented in JMP.

When the streamlined workflow is not sufficient, JMP extensibility features come into play, allowing the creation of new workflows that are augmented by the rich capabilities available in open-source programming languages, such as R (R Core Team, 2017) and Python (Van Rossum, 1995), and their large repositories of statistical methods and data manipulation processes.

The following sections describe applications that we have created to show how to extend JMP with open-source languages and libraries. In Section 2, we show a web application that allows end users to obtain house price predictions from a JMP model that has been converted to Python and deployed to a cloud provider. In Section 3, we show how JMP users can access special packages in R or Python without leaving the JMP user interface and without any additional programming. The results are returned in JMP to make use of JMP's advanced graphics capabilities, including easy filtering, interactive selection, and easy hover labels or markers using images.

2 | CASE STUDY: DELIVERING PREDICTIVE CAPABILITIES TO END USERS FROM A CLOUD-DEPLOYED MODEL

Machine learning models can only generate value for organizations when the insights from those models are delivered to end users. (Patruno, 2020)
The JMP extensibility example in this section illustrates how to take a JMP model and deploy it into a production environment so that it can be accessed by users. First, the JMP user interacts with the JMP interface to select and fit a model and subsequently to generate the Python scoring code. The model scoring code is deployed to a cloud provider and accessed through a web application.

For the data modelling, data were downloaded from the housing website Redfin (Redfin, 2020) for the metropolitan area of St. Louis, MO. The data used in this application were downloaded on 9/27/2017. The data include dozens of characteristics, such as listing price, number of bedrooms, number of bathrooms, square footage, and other critical factors, for a few thousand homes for sale and listed on Redfin.com on the date of download.

The goal was to create a Price Predictor tool that could be accessed on a web application. The user might be a real estate agent going door to door to look for new inventory. The agent could ask the homeowners about their home characteristics, enter these into the Price Predictor, and instantly show the homeowners a reasonable listing price for their home, based on a prediction model developed in JMP. Other users might include home buyers and sellers.

JMP Pro has many tools to facilitate fitting, comparison, and selection of predictive models. One of these tools is the Profiler, which enables the user to explore the model predictions across the model space. JMP Pro’s Model Comparison enables the users to stack and easily compare these profilers (Figure 1).

After the potential models are compared and a best model (or a model average of several ‘best’ models) is selected, the scoring code can be created. In this use case, we generate scoring code in the Python programming language, one of the primary languages supported by Amazon Web Services (AWS) Lambda for deployment.

Movie S1 shows the creation of a Listing Price Predictor Model in JMP Pro 15.2, a comparison of three contender models and the generation of scoring code in Python for the final model.

The Python scoring code was then deployed to run as an AWS Serverless function, which is executed when a user enters the house information in the companion web application and clicks on a button to request a house price prediction (Figure 2).

Movie S2 shows the final model that was deployed in AWS using the JMP-generated Python scoring code.

Our use of a web application backed by a predictive model deployed as a serverless function illustrates one of many possible deployment strategies for JMP models. Python scoring code could also be wrapped in a database User Defined Function (UDF) so that predicted values could be returned as part of SQL queries. Or the user could generate JavaScript code for inclusion of the model directly in a web page, mobile app or Excel UDF. Or the user could generate C code for applications that require the models to be embedded in a high-performance inference pipeline.

This application allows a user to provide input values that are a passed in an HTTP request to a REpresentational State Transfer (REST) endpoint running as an AWS API Gateway. This endpoint triggers the execution of an AWS Lambda function, a serverless function that wraps the Python scoring code for the JMP model that predicts house prices.

The implementation of the web application used Bootstrap (Bootstrap Team, n.d.) for the user interface and OpenLayers (OpenLayers, n.d.) for the mapping services. The web application is hosted on S3 (AWS, n.d.-b) as a static website. The interface allows the user to enter values for the house attributes (e.g., the number of bathrooms and the house location) using a customized HTML form and a clickable map of St. Louis. When the user clicks the UI button that triggers the request of the house price prediction, the user input is collected and sent over to the compute layer, where it is used as input in the model evaluation.

**FIGURE 1** The final model uses characteristics of the house and its location to predict the list price.
The compute layer uses two Amazon services: API Gateway (AWS, n.d.-a) is the entry point for the REST calls sent from the client, which are mapped to an AWS Lambda function (AWS, n.d.-c). Lambda is Amazon’s infrastructure of choice for applications that implement a Serverless Architecture (Roberts, 2018), also referred to as Function-as-a-Service (FaaS). This architecture provides performance at scale with low management costs to stateless, low latency, high-throughput applications. This architecture is a great fit for scoring applications since they can be ‘easily divided into components that can be executed concurrently’, fitting the embarrassingly parallel paradigm (Herlihy & Shavit, 2012).

In our implementation, the Python scoring code generated by JMP was combined with a small Lambda adapter module and deployed as a Lambda service. Deployment of applications to AWS services can be done manually using their management console (AWS, n.d.-d). However, to streamline the process, we recommend using one of the many open-source wrappers that are available. In this exercise, we used the Serverless application framework (Serverless, n.d.).

Source code and additional details about this example are available at these two blogs (Abousalh-Neto, 2017; Abousalh-Neto & Valente, 2016).

3 | CASE STUDY: AN INTERACTIVE JMP USER INTERFACE TO THE R t-SNE PACKAGE

This example of hybrid JMP-R usage requires jsl and R programming to create, and the result is a basic interactive application for end users.

The application allows, from within JMP, the use of R packages for t-Distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) methods of low-dimensional visualizations of clusters (McInnes, Healy, & Melville, 2018; van der Maaten, 2014; van der Maaten & Hinton, 2012). These methods are available in R and Python and not in JMP. We also used interactive visualization elements that are natively available in JMP but are not easily built in R or Python. The t-SNE and UMAP methods have been widely applied to various research areas such as image processing, text mining, and genomics (Kobak & Berens, 2019).

For example, genomic researchers have begun, over the past decade, to collect data from individual cells from sampled groups of cells. The researchers need to determine to which cell type each individual cell belongs and determine the function of an individual cell in the context of its micro-environment. Cluster analysis is often a statistical tool used for determining how many cell types are present and which cells belong to each type. Once the clusters have been determined and the cells classified to clusters, it is useful to visualize the ‘nearness’ or similarity of these clusters to each other, explore cases of known misclassification, and identify differential gene expression patterns across cell types. Non-linear dimension-reduction and visualization algorithms, such as t-SNE and UMAP, enable the user to visualize a 2-D or 3-D distance map.

An analogous example can be found in handwriting analysis. Imagine that you want to scan handwritten numbers and correctly identify them. For each handwritten digit, you could create a 28-by-28 pixel grid covering that digit, recording the darkness detected within each pixel. This is exactly the scenario for the data collected by the National Institute of Standards and Technology and modified by Yann Lecun (1998).
modified data are available in the *MNIST database of handwritten digits* (LeCun, n.d.). The resulting dataset could be arranged with one row per handwritten digit with 784 columns to correspond to each of the 28-by-28 pixels. For the training data, where the truth is known, we could include an extra column for the label of the actual digit (0, 1, 2, 3, 4, 5, 6, 7, 8 or 9). In JMP, we could also optionally include the original image of the handwriting for comparison later in the misclassifications. The data table is shown in Figures 3 and 4.

The application provides access to both the Rtsne (Krijthe, 2018) and umap (Konopka, 2020) R packages through an interactive interface with the same style as other platforms in JMP. This user-friendly interface enables data table navigation, data quality control, sparsity handling, intuitive parameterization, and interactive results interpretation.

JMP’s R integration functions make it easy to communicate with R. Roughly speaking, you perform four steps for a successful R integration: (1) establish the connection, (2) send data to R, (3) submit R script to perform the main calculation, and (4) bring results generated by R to JMP. In practice, transferring data between JMP and R could take a considerable amount of time. Therefore, some data preprocessing and dimensionality reduction steps will greatly help to reduce the overall computing time. The application was built using R Version 3.5.2 for Windows (3.3.3 for Mac) and JMP 14.0. Movie S3 shows the use of the application.

Once the results are brought back from R into JMP, we benefit from the JMP interactive features, such as linked data selection and easy data filtering. Figure 5 shows the t-SNE visualization of the MNIST dataset coloured by the original label. Users can easily explore a subset of clusters with the Local Data Filter. Movie S4 shows this interactivity on the subset of data for which we had the original images.

This application shows the synergistic benefit of combining the two software packages. These methods are needed and are available in R (or Python) but not yet in JMP. The user-friendly interface prevents the need to rewrite the programming code. The JMP benefits of interactive and linked graphics allow for more exploration of the resulting visualizations, and more insight gained for the characteristics of the incorrectly classified digits, beyond what is straightforwardly possible in R.

---

**FIGURE 3** 10,000 rows from the MNIST database of handwritten digits

**FIGURE 4** A more interesting part of the data table in Figure 3
4 | CONCLUSIONS

The case studies illustrate two different approaches to extend JMP capabilities with open-source software: code generation for model deployment and the creation of hybrid applications that combine open-source packages and JMP user interface components. In both cases, the usefulness of the final product was greater than the sum of its parts.

We hope that these examples will inspire the statistics and data science community to develop new JMP + open-source applications.

5 | FURTHER READING

- Learn more about how to interface JMP with R and Python (Hummel, 2020).
- JMP add-ins (JMP, 2020) provide an infrastructure to wrap a JMP user interface to external libraries defined in other languages.
- Explore a JMP add-in that provides access to a Python implementation of the XGBoost algorithm (Richard, 2020), and a JMP add-in that accesses XGBoost through C++ (Wolfinger, 2020).

ORCID

Ruth Hummel https://orcid.org/0000-0003-1323-3834

REFERENCES

Abousalh-Neto, N. (2017). Sharing the ultimate boon: A journey from modeling to scoring. Retrieved August 2020, from community.jmp.com: https://community.jmp.com/t5/Discovery-Summit-2017/Sharing-the-Ultimate-Boon-A-Journey-From-Modeling-to-Scoring/ta-p/46020
Abousalh-Neto, N., & Valente, D. (2016). Scoring outside the box. Retrieved August 2020, from community.jmp.com: https://community.jmp.com/t5/Discovery-Summit-2016/Scoring-Outside-the-Box/tac-p/46361#M99
Amazon Web Services. (n.d.-a). Amazon API Gateway—Create, maintain, and secure APIs at any scale. Retrieved September 2017, from Amazon API Gateway: https://aws.amazon.com/api-gateway/
Amazon Web Services. (n.d.-b). Amazon S3—Object storage built to store and retrieve any amount of data from anywhere. Retrieved September 2017, from Amazon S3: https://aws.amazon.com/s3/
Amazon Web Services. (n.d.-c). AWS Lambda—Run code without thinking about servers. Retrieved September 2017, from AWS Lambda: https://aws.amazon.com/lambda/
Amazon Web Services. (n.d.-d). AWS Management Console—Everything you need to access and manage the AWS cloud. Retrieved September 2017, from AWS Management Console: https://aws.amazon.com/console/
Bootstrap Team. (n.d.). Bootstrap, the world’s most popular front-end open source toolkit. Retrieved September 2017, from Bootstrap: http://getbootstrap.com/
Herlihy, M., & Shavit, N. (2012). The art of multiprocessor programming, revised reprint (revised ed.). In The art of multiprocessor programming (p. 14). Elsevier. Retrieved September 2017
Hummel, R. (2020). Using JMP and JMP Pro with Python and R—JMP synergies with open source. Retrieved August 2020, from JMP Resource Center: https://www.jmp.com/en_us/whitepapers/jmp/jmp-synergies-using-jmp-and-jmp-pro-with-python-and-r.html
JMP. (2020). Compile add-ins with the JMP add-in builder. Retrieved August 2020, from JMP Scripting Guide: https://www.jmp.com/support/help/en/15.2/index.shtml#page/jmp/compile-add-ins-with-the-jmp-add-in-builder.shtml
Kobak, D., & Berens, P. (2019). The art of using t-SNE for single-cell transcriptomics. Nature Communications, 10, 5416. https://doi.org/10.1038/s41467-019-13056-x
Konopka, T. (2020). UMAP: Uniform Manifold Approximation and Projection. Retrieved August 2020, from CRAN: https://cran.r-project.org/web/packages/umap/index.html
