Synergies JMP® : Utiliser JMP® et JMP® Pro avec Python et R

LIVRE BLANC

Table des matières

Introduction

JMP est un ensemble de logiciels autonomes développé par SAS Institute, proposant un ensemble complet d'outils de visualisation de données et d'analyse statistique, pour Windows et Mac. Son interactivité et ses liens dynamiques rendent l'exploration de données captivante et instructive. JMP propose en outre de nombreuses options analytiques avancées et répond ainsi pleinement aux besoins des professionnels des données. De plus, il est possible, selon vos besoins, d'utiliser JMP en association avec des outils open source, tels que Python ou R.

Voici quelques exemples :

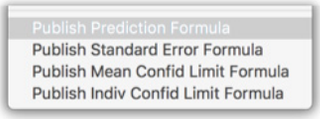
- Un statisticien dont l'entreprise lui interdit d'utiliser un logiciel non validé tel que R pour les besoins d'analyse primaire peut néanmoins vouloir essayer une nouvelle approche basée sur R mais utilisant des méthodes validées dans JMP.
- Un programmeur Python ou R peut vouloir profiter des liens dynamiques, des représentations visuelles avancées ou des puissantes analyses offertes par JMP.
- Le service d'analyse des données d'une entreprise peut vouloir développer un complément JMP appelant du code R ou Python. Les utilisateurs du complément accéderont alors par le biais d'une interface graphique et non par le biais de la programmation au code R ou Python.
- Un enseignant en statistique peut vouloir intégrer des packages open source précis dans ses cours, mais au sein d'une interface cliquable ne nécessitant aucun codage par les étudiants.

Quelle que soit la raison pour laquelle vous souhaitez connecter des outils open source (ou autres) à l'interface JMP, ce guide vous apprendra à utiliser les connexions à Python et R dans JMP[1].

JMP® et Python

Générer du code Python à partir d'un modèle JMP Pro

JMP Pro vous permet de générer du code Python pour les modèles prédictifs créés dans JMP. Cela s'avère utile lorsque vous souhaitez construire, examiner et comparer des modèles dans JMP Pro, mais les déployer dans un environnement de production différent. Le logiciel permet en effet de construire et de comparer facilement plusieurs modèles concurrents, et même de créer une moyenne de modèles (ensemble) dans la plate-forme Dépôt des formules.



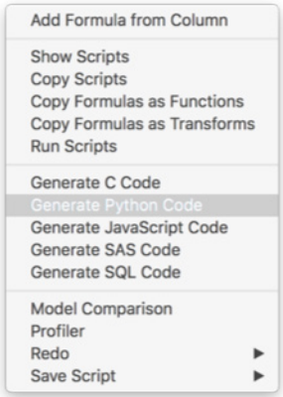
Tout d'abord, ajustez un modèle à l'aide des outils du menu Analyse de JMP. Pour enregistrer le modèle dans le Dépôt des formules, utilisez les options « Publier la formule de probabilité » (pour les modèles de classification) ou « Publier la formule de prévision » (pour les modèles prédictifs). En général, les deux options sont accessibles en cliquant sur le triangle rouge associé aux résultats du modèle ou sous « Enregistrer les colonnes ».

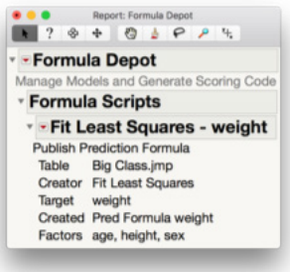
[1] JMP se connecte également à SAS et MATLAB (en plus de R et Python), et génère également du code de scoring de modèle en C, JavaScript, SAS et SQL (en plus de Python). Pour en savoir plus sur ces connexions JMP, et pour obtenir d'autres conseils et exemples de prise en main, rendez-vous sur jmp.com/support/help/en/15.1/#page/jmp/extending-jmp.shtml.

Une fois que vous avez sélectionné le mot-clé « Publier » (ou utilisé l'option « Ajouter une formule de la colonne »), votre modèle est envoyé au Dépôt des formules. La plate-forme Dépôt des formules vous permet d'enregistrer un ou plusieurs modèles concurrents, de les comparer, de créer des modèles d'ensemble et de les déployer dans divers environnements de production.

L'option « Générer le code Python », qui apparaît en cliquant sur le triangle rouge de chaque modèle (ici, le modèle « Fit Least Squares » ou « Ajustement par moindres carrés » pour la prédiction du **poids**) ou sur celui de Dépôt des formules, permet de générer le code pour l'ensemble des modèles enregistrés dans la fenêtre Dépôt des formules.

Le code Python qui en résulte (voir ci-dessous) comprend certaines dépendances obligatoires, qui sont appelées dans la première section du code. Suivent les informations sur les droits d'auteur, puis la définition des noms et des types de variables. Vient enfin la formule qui sera utilisée pour prédire la variable de réponse à l'aide des valeurs des variables de prévision (dans cet exemple, l'*âge, la taille et le sexe*).





Fit_Least_Squares_weight

```
from __future__ import division
import jmp_score as jmp
from math import *
import numpy as np


""" ================================================================
 Copyright(C) 2018 SAS Institute Inc.All rights reserved.

 Notice:
 The following permissions are granted provided that the
 above copyright and this notice appear in the score code and
 any related documentation. Permission to copy, modify
 and distribute the score code generated using
 JMP(R) software is limited to customers of SAS Institute Inc. ("SAS")
 and successive third parties, all without any warranty, express or
 implied, or any other obligation by SAS. SAS and all other SAS
 Institute Inc. product and service names are registered
 trademarks or trademarks of SAS Institute Inc. in the USA
 and other countries. Except as contained in this notice,
 the name of the SAS Institute Inc. and JMP shall not be used in
 the advertising or promotion of products or services without
 prior written authorization from SAS Institute Inc.
 ================================================================ """

""" Python code generated by JMP v14.1.0 """

def getModelMetadata():
    return {"creator": u"Fit Least Squares", "modelName": u"", "predicted": u"weight",
    "table": u"Big Class.jmp", "version": u"14.1.0", "timestamp": u"2018-08-08T04:02:07Z"}


def getInputMetadata():
    return {
        u"age": "float",
        u"height": "float",
        u"sex": "str"
    }


def getOutputMetadata():
    return {
        u"Pred Formula weight": "float"
    }


def score(indata, outdata):
    _temp_0 = np.nan
    _temp_1 = np.nan

    if (indata[u"sex"] == u"F"):
        _temp_0 = 2.0492068900361
    elif (indata[u"sex"] == u"M"):
        _temp_0 = -2.0492068900361
    else:
        _temp_0 = np.nan
    if (jmp.numeq(indata[u"age"], 12)):
        _temp_1 = 0
    elif (jmp.numeq(indata[u"age"], 13)):
        _temp_1 = -13.6855931672148
    elif (jmp.numeq(indata[u"age"], 14)):
        _temp_1 = -25.8472610743993
    elif (jmp.numeq(indata[u"age"], 15)):
        _temp_1 = -19.7698493666905
    elif (jmp.numeq(indata[u"age"], 16)):
        _temp_1 = -10.1590212259529
    elif (jmp.numeq(indata[u"age"], 17)):
        _temp_1 = 2.52025614257322
    else:
        _temp_1 = np.nan
    outdata[u"Pred Formula weight"] = -176.033203126788 + 4.72294023921341 * indata[u"height"]
    + _temp_0 + _temp_1

    return outdata[u"Pred Formula weight"]
```

Pour en savoir plus à ce sujet, rendez-vous sur jmp.com/support/help/en/15.1/#page/jmp/formula-depot.shtml.

Les bases pour connecter JMP et Python

En plus de générer le code de scoring des modèles Python à partir de JMP, vous pouvez également relier JMP et Python, puis envoyer et recevoir des données, des résultats et des actions (en mêlant du code JMP et Python) entre Python et JMP. (La connexion entre JMP et R fonctionne de la même manière.) Pour pouvoir connecter Python à JMP, vous devez utiliser le langage de script de JMP (JSL) pour encapsuler votre code Python.

Prenons un exemple simple. Python[2] devra être installé sur votre machine ; il est aujourd'hui présent de base sur la plupart des nouveaux ordinateurs.

(1) Ouvrez la connexion à Python à partir de JMP.

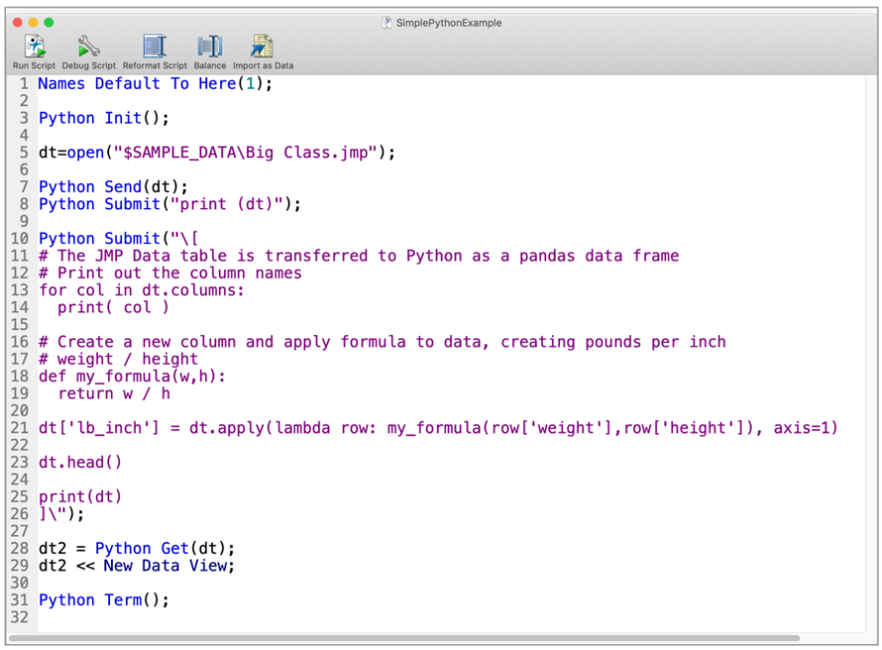
(2) Effectuez une opération, comme envoyer des données de JMP à Python.

(3) Effectuez une autre opération, par exemple envoyer du code Python directement à Python afin de créer une nouvelle variable.

(4) Effectuez une autre opération, comme renvoyer un résultat ou des données de Python à JMP (visualisez éventuellement le résultat dans JMP).

(5) Fermez la connexion à Python.

Voici le code de cet exemple simple. Pour procéder, ouvrez d'abord JMP. Cliquez ensuite sur Fichier > Nouveau > Nouveau script afin d'ouvrir une nouvelle fenêtre de script vierge. Tapez ensuite ce qui suit :




```
Names Default To Here(1);

Python Init();

dt=open("$SAMPLE_DATA\Big Class.jmp");

Python Send(dt);
Python Submit("print (dt)");

Python Submit("\[
# The JMP Data table is transferred to Python as a pandas data frame
# Print out the column names
for col in dt.columns:
  print( col )

# Create a new column and apply formula to data, creating pounds per inch
# weight / height
def my_formula(w,h):
  return w / h

dt['lb_inch'] = dt.apply(lambda row: my_formula(row['weight'],row['height']), axis=1)

dt.head()

print(dt)
]\");

dt2 = Python Get(dt);
dt2 << New Data View;

Python Term();
```

Le script ci-dessus contient neuf éléments de code JSL (indiqués en bleu). La première ligne permet de s'assurer que toute référence à un fichier ou à une variable que nous effectuons sera associée à ce script. La troisième ligne, « **`Python Init`**(); », ouvre la connection à Python[3].

[2] python.org.

[3] Vous trouverez des informations détaillées sur l'installation et les versions de Python ici : jmp.com/support/help/en/15.1/#page/jmp/install-python.shtml# et jmp.com/support/help/en/15.1/#page/jmp/troubleshooting-the-jmp-python-integration.shtml#ww822804.

La ligne 5, « **dt = Open**(**"$SAMPLE_DATA/Big Class.jmp"**); », demande à JMP d'ouvrir un jeu de données situé dans le répertoire des échantillons de données de JMP et lui attribue le nom « dt »[4]. Dans cet exemple, le jeu de données est appelé « Big Class.jmp ».

La ligne 7, « **Python Send(dt)** ; », envoie la table de données « dt » de JMP à Python. Elle est alors transférée à Python sous la forme d'une trame de données Pandas. La ligne 8, « **Python Submit("print(dt)");** », envoie du code Python à Python. L'information entre guillemets correspond au script Python, le reste de la ligne à l'encapsuleur du code JSL. Pensez à vérifier vos soumissions de code et à les corriger, le cas échéant, à l'aide de la fenêtre Log de JMP. Dans la barre de menu de JMP, cliquez sur Fenêtre → Log. Vous pouvez y voir tous les résultats ou les messages d'erreur Python internes. Par exemple, si nous exécutons ce code tel quel, nous obtiendrons le message suivant :

```
dt=open("$SAMPLE_DATA\Big Class.jmp");
/*:

Data Table( "Big Class.jmp" )
//:*/
Python Send(dt);
Python Submit("print (dt)");
/*:

      name  age sex  height  weight
0    KATIE   12   F      59      95
1   LOUISE   12   F      61     123
2     JANE   12   F      55      74
3   JACLYN   12   F      66     145
4   LILLIE   12   F      52      64
```

Les lignes 10 à 26 comprennent une instruction « **Python Submit();** » plus longue. Cet élément de code Python, intégré à l'instruction JSL « **Python Submit()** », imprimera le nom des colonnes (dans le log) et créera ensuite une nouvelle colonne selon la formule « lb_inch = weight/height », avant d'imprimer enfin la nouvelle table de données avec cette nouvelle colonne dans le log :

```
/*:

name
age
sex
height
weight
      name  age sex  height  weight   lb_inch
0    KATIE   12   F      59      95  1.610169
1   LOUISE   12   F      61     123  2.016393
2     JANE   12   F      55      74  1.345455
3   JACLYN   12   F      66     145  2.196970
4   LILLIE   12   F      52      64  1.230769
5      TIM   12   M      60      84  1.400000
6    JAMES   12   M      61     128  2.098361
```

Les lignes 28 et 29 renvoient la table de données actualisée dans la mémoire de travail du logiciel et l'ouvrent en tant que nouvelle table de données JMP.

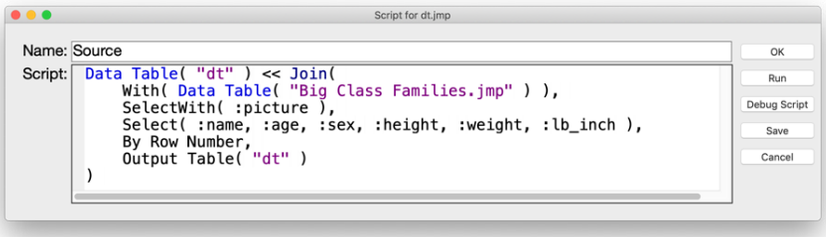
La dernière ligne du code, « **Python Term();** », met fin à la connexion à Python.

[4] Notez que nous avons utilisé une ligne de code JSL légèrement différente dans l'exemple R pour pointer vers un fichier de données ouvert. Il s'agit d'une autre façon de pointer vers des données, dans ce cas pour ouvrir un fichier de données, en utilisant JSL.

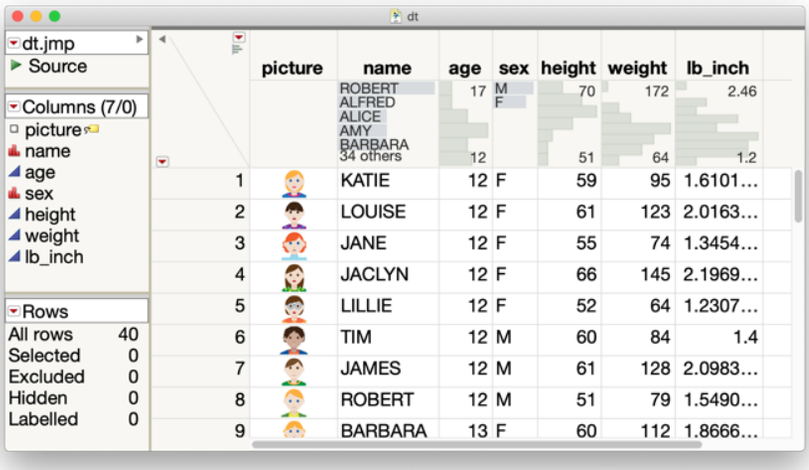
Aller plus loin avec JMP – Ajouter un marqueur d'image à un graphique

Bien que la dernière partie de cet exemple n'utilise plus la connexion JMP à Python, elle montre cependant les possibilités offertes lorsque l'on associe JMP avec Python (ou R ou un autre logiciel connecté). Il est possible notamment de travailler de manière interactive entre JMP et Python, en exécutant du code Python prédéfini ou des packages Python spéciaux, et d'améliorer ensuite les analyses ou les visualisations finales grâce aux fonctions dynamiques de JMP.

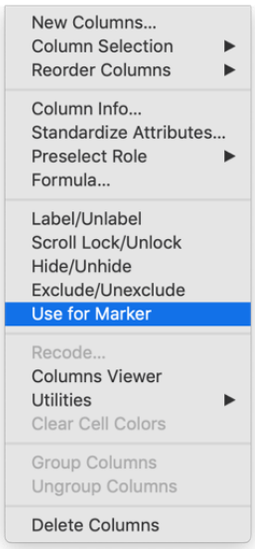
Après avoir renvoyé la table de données « dt » dans JMP, nous pouvons par exemple utiliser des outils de visualisation en quelques clics ou à l'aide de courts scripts JSL afin d'améliorer nos graphiques. Illustrons cela en joignant notre table « dt » au jeu d'échantillons de données intitulé « Big Class Families », disponible dans la bibliothèque d'échantillons de données de JMP, et en ajoutant une colonne d'images à notre nouveau « dt ». Commencez par ouvrir Big Class Families (en le sélectionnant depuis Aide > Bibliothèque d'échantillons de données) ou en exécutant le code « `dt = Open("$SAMPLE_ DATA/Big Class.jmp");` »), puis cliquez sur Tables > Joindre (ou le script suivant) pour joindre la colonne d'images à notre table « dt ».

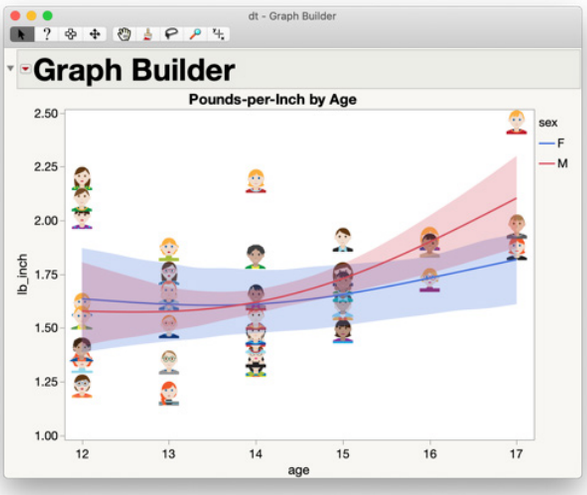


Ces étapes permettent d'ajouter une colonne appelée « picture » à notre jeu de données et d'importer les images des élèves de la table de données « Big Class Families » dans notre nouvelle table de données combinée.



Nous pouvons maintenant créer une représentation visuelle de la nouvelle variable « pounds-per-inch », en utilisant les images comme marqueurs dans notre graphique. Tout d'abord, définissez la colonne d'images en tant que marqueur en cliquant sur la colonne « image », puis sur Colonnes > Utiliser pour le marqueur.





Dans cet exemple simple, nous avons utilisé la connexion JMP à Python pour ouvrir des données dans JMP, les envoyer à Python, créer une nouvelle colonne dans Python et ajouter la nouvelle colonne à la trame de données Pandas de Python, avant de renvoyer les données actualisées sous forme de table de données JMP, que nous pourrons analyser et manipuler plus avant dans JMP par la suite.

JMP avec Python : autres exemples simples

Vous trouverez d'autres exemples simples sur les interactions entre JMP et Python dans l'Aide de JMP. Pour cela, cliquez sur Index de script → Étendre JMP Python Connection → ou en vous rendez-vous sur jmp.com/support/help/en/15.1/#page/jmp/additional-python-integration-examples.shtml.

Vous trouverez le code nécessaire pour :

- Envoyer une table de données vers Python.
- Créer des objets dans Python.
- Effectuer des opérations de matrice dans Python.

JMP® et R

Connecter JMP au logiciel open source R[5] vous permet de profiter facilement à la fois des capacités de R et de l'approche dynamique et interactive du logiciel JMP en matière d'analyse et de visualisation. Vous pouvez également créer des interfaces de menu faciles à utiliser pour les packages R et combiner des éléments JMP et R. Il vous suffit d'avoir à la fois JMP et R installés sur votre ordinateur. Les fonctions R permettant d'établir ces connexions sont répertoriées et décrites (avec des exemples) dans l'Index des scripts (Aide > Index des scripts > Fonctions R).

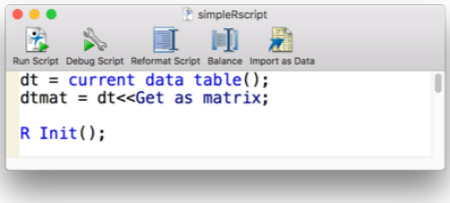
Les bases pour connecter JMP et R

Pour utiliser la connexion à R dans JMP, vous devez encapsuler votre code R à l'aide du langage de script de JMP (JSL).

Prenons un exemple. Transférons certaines données de JMP vers R en vue de les analyser et de les manipuler, puis renvoyons-les vers JMP. La démarche est relativement simple. Nous allons rédiger un bout de code JSL pour ouvrir R, envoyer les données de JMP à R, envoyer le code R à R puis renvoyer les données de R dans JMP.

Tout d'abord, vérifiez que R et JMP sont bien installés sur la même machine[6].

Puis, ouvrez une fenêtre de script dans JMP en cliquant sur **Fichier > Nouveau > Nouveau Script**.

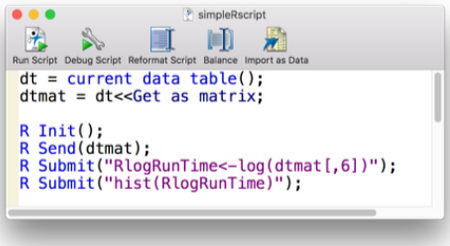


Ouvrons également un échantillon de données dans JMP. Accédez à **Aide > Bibliothèque d'échantillons de données** et ouvrez la table **Fitness.jmp**. Pour l'envoyer à R, nous avons seulement besoin de deux lignes de code JSL. La première identifie la table de données ouverte, **Fitness.jmp**, et la renomme « dt ». La deuxième convertit la table nouvellement nommée « dt » en matrice. Cette matrice sert de cadre commun à JMP et R, étant donné que R structure les données de manière légèrement différente de JMP. Ensuite, nous appelons R depuis JMP. Si R est installé sur la même machine, JMP le trouvera ; il n'est pas nécessaire d'établir une connexion ou de pointer JMP vers l'emplacement de R. Il suffit de saisir la ligne JSL « `R Init();` ». JMP recherchera alors R sur la machine, puis le lancera.

[5] cran.r-project.org.

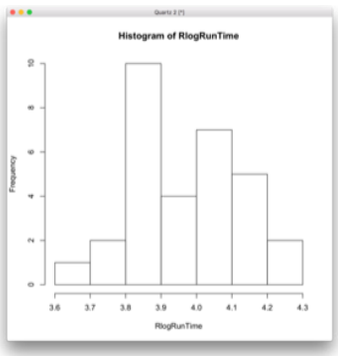
[6] Vous trouverez des informations détaillées sur l'installation et les versions de R ici : jmp.com/support/help/en/15.1/#page/jmp/installing-r.shtml.

Nous sommes maintenant prêts à envoyer des informations dans les deux sens entre JMP et R. La première ligne du code suivant envoie notre version matricielle des données, appelée dtmat, à R. La ligne suivante envoie le code R à exécuter dans R.



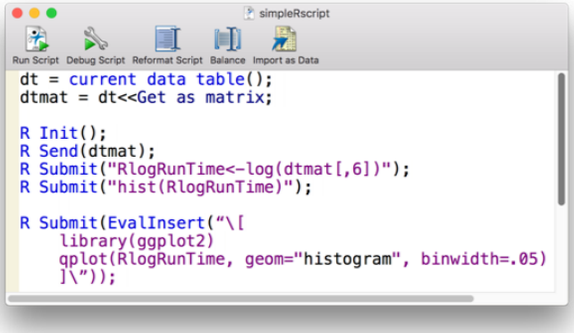
La ligne « `RlogRunTime<-log(dtmat[,6]);` » récupère la sixième colonne de la matrice d'entrée (l'« allure au mile » du jeu de données Fitness) et la convertit en log. Ces résultats seront désormais appelés « RlogRunTime » (c'est en effet le nom que nous lui avons donné dans notre extrait de code R).

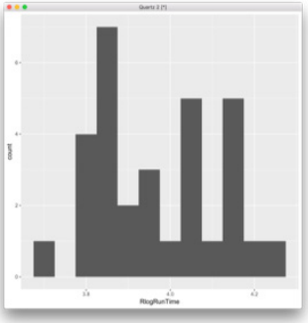
La ligne « `R Submit("hist(RlogRunTime)");` » génère un histogramme dans R, qui s'affichera à l'écran lors de l'exécution du code.



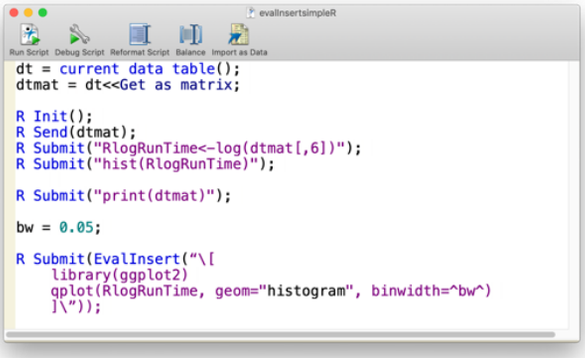
Pour exécuter le code, sélectionnez une ou plusieurs lignes (chaque ligne doit se terminer par « ; ») et cliquez sur le bouton « Exécuter le script » en haut de la fenêtre de script :

Si le bout de code R à envoyer via notre encapsuleur JSL comporte plusieurs lignes, nous pouvons utiliser « `R Submit(EvalInsert("\[____]\"))` » à la place de « `R Submit("___")` ». C'est d'ailleurs le cas dans l'exemple suivant, où nous utilisons le paquet « `ggplot2` » dans R pour créer un histogramme :

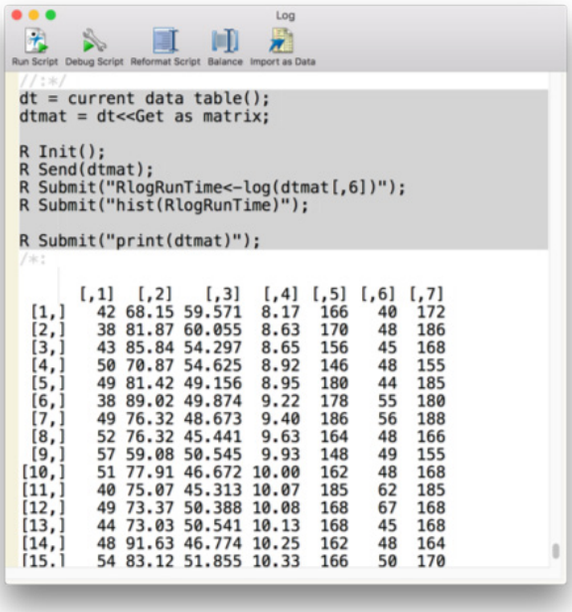




Par ailleurs, cet « **EvalInsert**() » est généralement utilisé pour remplacer ou évaluer des expressions au sein d'une chaîne de caractères dans JSL. Par exemple, nous pourrions préciser la largeur des intervalles de l'histogramme R via une variable dans le script JSL et ensuite appeler cette variable via le code « **EvalInsert**() », comme ceci :

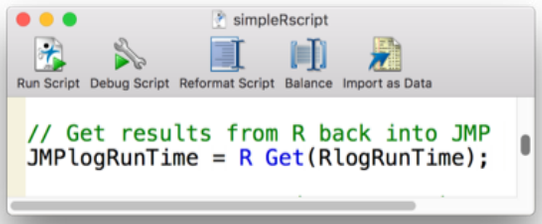


Pensez à vérifier vos soumissions de code et à les corriger, le cas échéant, à l'aide de la fenêtre Log de JMP. Dans la barre de menu de JMP, cliquez sur Fenêtre → Log. Vous pouvez y voir tous les résultats ou les messages d'erreur R internes.

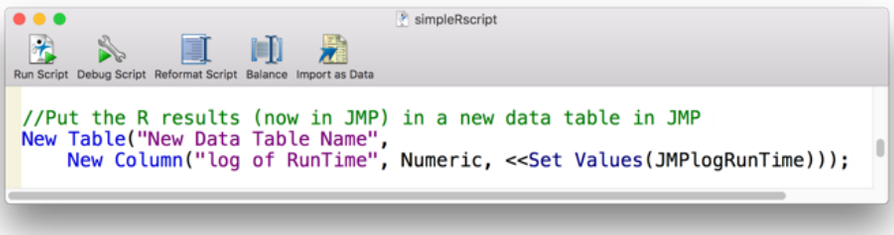


Bien entendu, il est également possible de créer cet histogramme dans JMP en utilisant la Plate-forme des distributions. Nous pouvons utiliser les fonctions JMP de deux manières différentes.

Premièrement, nous devons renvoyer « **RlogRunTime** » de R à JMP. Cette section de code prend les données nouvellement converties, intitulées « **RlogRunTime** » dans R, et les renvoie à JMP sous le nom « **JMPlogRunTime** ».



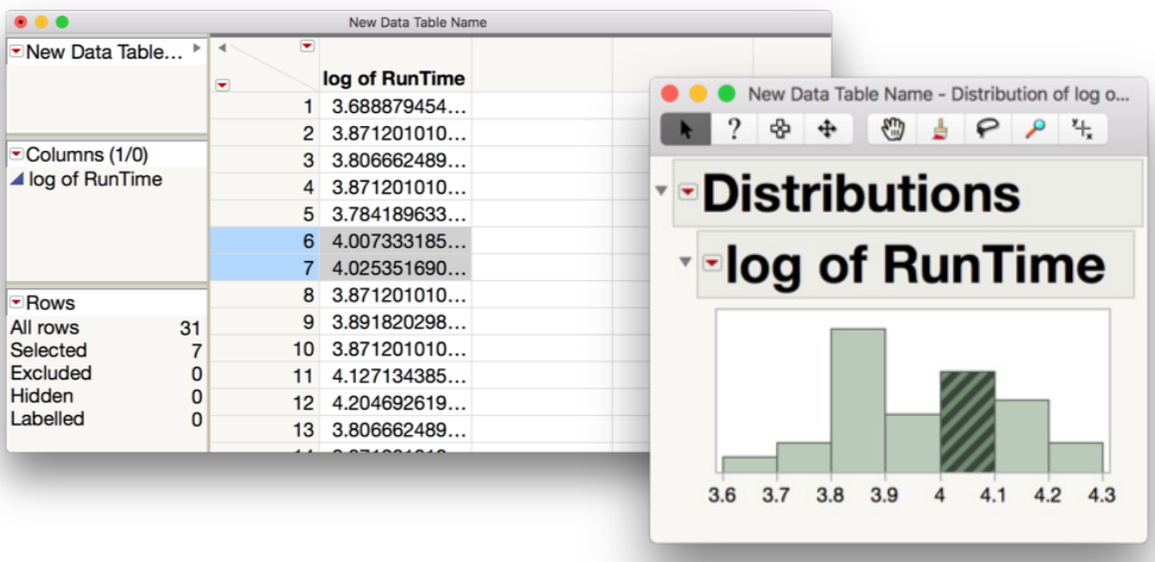
Nous devons également déplacer ces nouvelles données JMP de la mémoire interne vers une nouvelle table de données dans JMP afin de pouvoir les enregistrer et effectuer d'autres opérations sur celles-ci. Le code ci-après crée une nouvelle table de données JMP composée d'une seule colonne et des données de « **JMPlogRunTime** ».



Nous pouvons ensuite utiliser d'autres fonctions JMP de deux manières différentes.

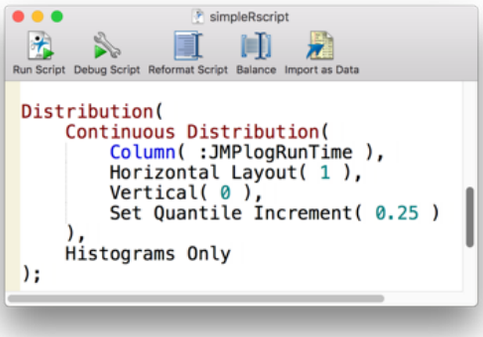
(1) À partir des menus cliquables pour effectuer des opérations sur la nouvelle table de données JMP.

Dans notre exemple, nous pouvons utiliser l'option Analyse > Distribution pour créer un histogramme interactif dans JMP.



(2) Via JSL toujours à partir d'une fenêtre de script.

Si vous maîtrisez la programmation JSL, vous pouvez rédiger d'autres actions JMP directement dans votre fenêtre de script. Sinon, vous pouvez facilement générer le code JSL correspondant en utilisant l'interface interactive pour créer les actions une première fois. Il vous suffit ensuite de cliquer sur le triangle rouge dans la fenêtre de résultats et de sélectionner Copier le script dans la fenêtre de script. Pour mettre à jour le script pour une nouvelle table de données, modifiez le script JSL afin de l'appliquer à la ou aux variables appropriées.



JMP avec R : autres exemples simples

Vous trouverez d'autres exemples simples sur la synergie entre JMP et R en recherchant « R » dans l'Aide de JMP et en accédant à Guide de scriptage → Étendre JMP → Travailler avec R.

Dans l'Aide de JMP, vous trouverez le code nécessaire pour :

- Envoyer une table de données vers R.
- Créer des objets dans R.
- Utiliser les fonctions et graphiques R.
- Ajouter facilement des matrices dans R.
- Obtenir des intervalles de confiance bootstrappés dans R.

Exemples avancés

Une fois que vous maîtrisez les bases des interfaces JMP avec R et Python, vous pourrez élargir ces connaissances de manière intéressante et interactive.

Voici quelques exemples avec lien :

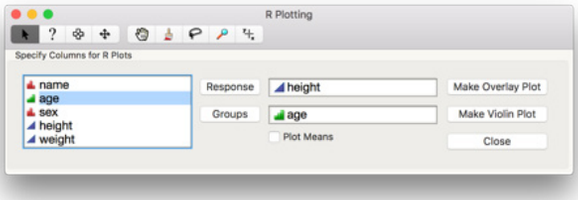
Appel interactif JMP à GGPlot

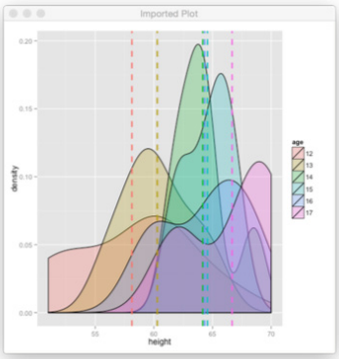
Ce script utilise la connexion JMP à R afin de créer des graphiques de densité à courbes lissées[7] à partir du package R ggplot2[8], décomposés par niveau à l'aide d'une variable de regroupement.

[7] Notez que ce type de vue d'histogramme est également une option native du Constructeur de graphiques JMP (version 15 et supérieure). L'utilisation de R dans cet exemple sert davantage d'illustration que de nécessité.

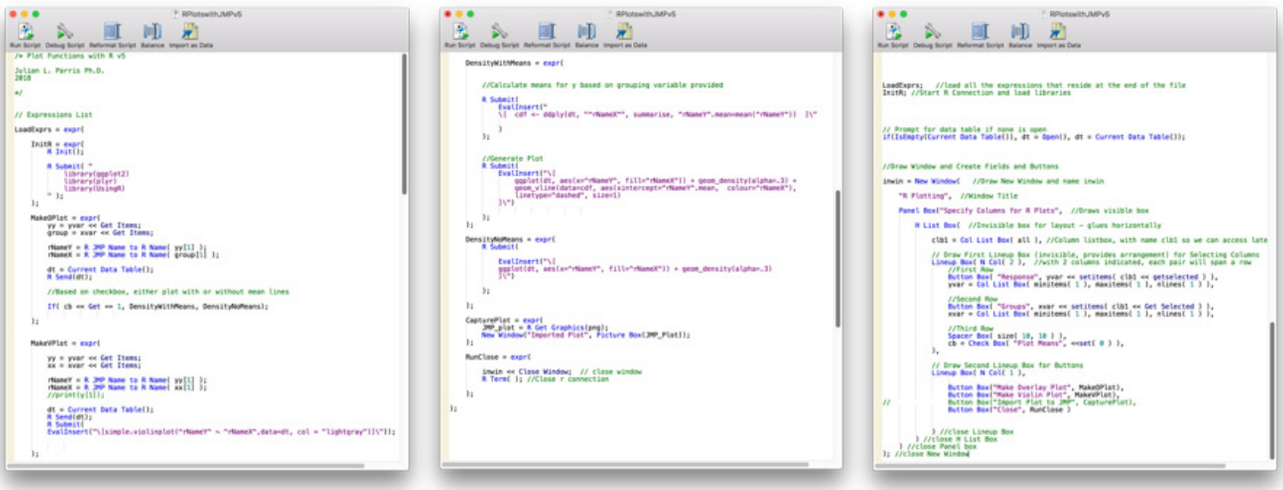
[8] Voir cran.r-project.org/web/packages/ggplot2/index.html.

Pour l'exécuter, choisissez un fichier de données, puis ouvrez et exécutez ce script. Vous verrez apparaître une boîte de dialogue contenant les colonnes de la table de données ouverte. Après quelques interactions avec cette invite, vous obtenez le graphique R correspondant.





L'utilisation de ce script ne nécessite pas de rédiger le code R ou JSL. Toutefois, si vous souhaitez créer quelque chose de similaire, le code JSL sous-jacent (qui contient des extraits du code R) est indiqué ici (téléchargeable à partir du lien vers cet exemple) :



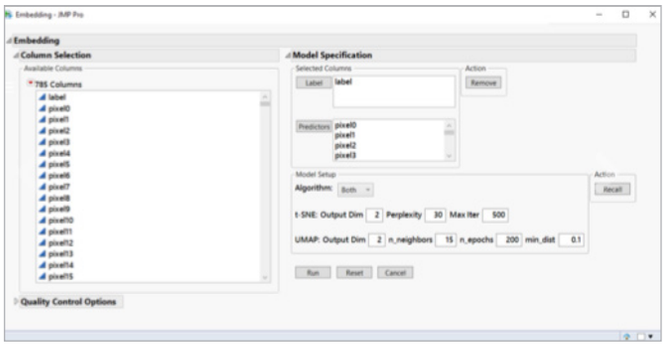
Pour en savoir plus et télécharger ce complément, consultez l'article suivant sur la communauté JMP : community.jmp.com/docs/DOC-6472.

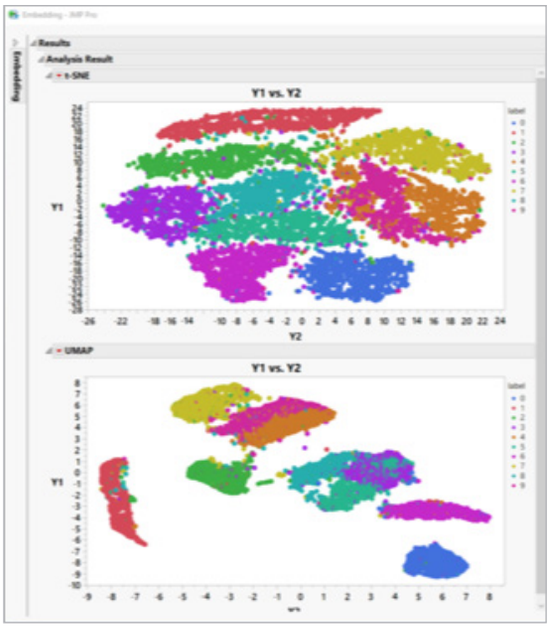
Visualiser des données avec le complément t-SNE et UMAP :

Ce script, qui utilise les packages t-SNE et UMAP[9] dans R, est encore plus convivial lorsqu'il est intégré à un complément JMP.

T-SNE (t-Distributed Stochastic Neighbor Embedding) et UMAP (Uniform Manifold Approximation and Projection) sont des algorithmes de réduction dimensionnelle et de visualisation non linéaires qui gagnent en popularité dans les domaines du traitement d'images, du text mining et de la génomique notamment. Ce complément fournit une interface conviviale à ces deux packages R. Il permet de naviguer dans les tables de données, de contrôler la qualité des données, d'en gérer la dispersion, de procéder à un réglage intuitif des paramètres et d'interpréter les résultats de manière interactive.

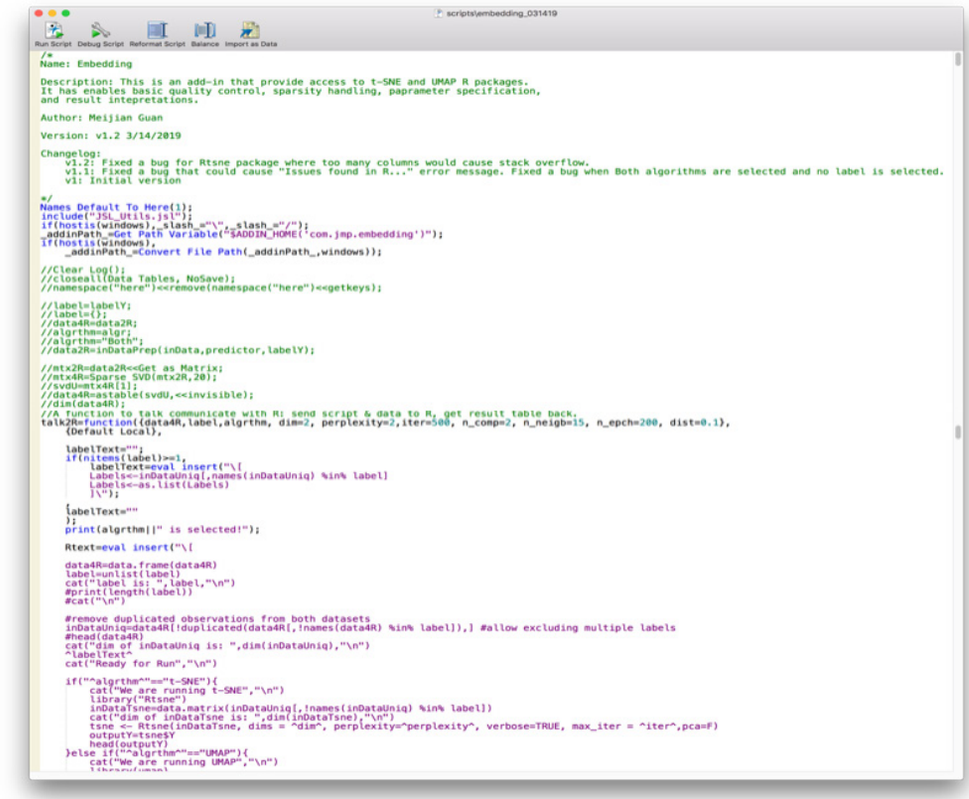
[9] Voir cran.r-project.org/web/packages/Rtsne/ et cran.r-project.org/web/packages/umap/.





Lors de la création d'un complément comme celui-ci, un seul utilisateur suffit pour élaborer le code R ou Python, ainsi que le code JSL correspondant. Il devra ensuite encapsuler le tout dans un complément JMP à l'aide du Générateur de compléments JMP[10]. Le fichier .jmpaddin obtenu peut être envoyé par e-mail à d'autres utilisateurs ou publié sur le Web[11]. Cela permet à d'autres utilisateurs d'exécuter le complément sans avoir à rédiger, consulter ou modifier le code, mais en utilisant à la place l'interface cliquable conviviale de JMP.

Le code invisible, qui s'exécute en arrière-plan, commence ainsi (le code R est indiqué en violet) :



```
/*
Name: Embedding

Description: This is an add-in that provide access to t-SNE and UMAP R packages.
It has enables basic quality control, sparsity handling, papramater specification,
and result intepretations.

Author: Meijian Guan

Version: v1.2 3/14/2019

Changelog:
    v1.2: Fixed a bug for Rtsne package where too many columns would cause stack overflow.
    v1.1: Fixed a bug that could cause "Issues found in R..." error message. Fixed a bug when Both algorithms are selected and no label is selected.
    v1: Initial version

*/
Names Default To Here(1);
include("JSL_Utils.jsl");
if(hostis(windows),_slash_="\",_slash_="/");
_addinPath_=Get Path Variable("$ADDIN_HOME('com.jmp.embedding')");
if(hostis(Windows),
    _addinPath_=Convert File Path(_addinPath_,windows));

//Clear Log();
//closeall(Data Tables, NoSave);
//namespace("here")<<remove(namespace("here")<<getkeys);

//label=labelY;
//labels={};
//data4R=data2R;
//algrthm=algr;
//algrthm="Both";
//data2R=inDataPrep(inData,predictor,labelY);

//mtx2R=data2R<<Get as Matrix;
//mtx4R=Sparse SVD(mtx2R,20);
//svdU=mtx4R[1];
//data4R=mastable(svdU,<<invisible);
//dim(data4R);
//A function to talk communicate with R: send script & data to R, get result table back.
talk2R=function({data4R,label,algrthm, dim=2, perplexity=2,iter=500, n_comp=2, n_neigbs=15, n_epch=200, dist=0.1},
    {Default Local},

    labelText="";
    if(nitems(label)>=1,
        labelText=eval insert("\[
        Labels<-inDataUniq[,names(inDataUniq) %in% label]
        Labels<-as.list(Labels)
        ]\");

    labelText=""
    );
    print(algrthm||" is selected!");

    Rtext=eval insert("\[

    data4R=data.frame(data4R)
    label=unlist(label)
    cat("label is: ",label,"\n")
    #print(length(label))
    #cat("\n")

    #remove duplicated observations from both datasets
    inDataUniq=data4R[!duplicated(data4R[,!names(data4R) %in% label]),] #allow excluding multiple labels
    #head(data4R)
    cat("dim of inDataUniq is: ",dim(inDataUniq),"\n")
    ^labelText^
    cat("Ready for Run","\n")

    if(""^algrthm^""=="t-SNE"){
        cat("We are running t-SNE","\n")
        library("Rtsne")
        inDataTsne=data.matrix(inDataUniq[,!names(inDataUniq) %in% label])
        cat("dim of inDataTsne is: ",dim(inDataTsne),"\n")
        tsne <- Rtsne(inDataTsne, dims = ^dim^, perplexity=^perplexity^, verbose=TRUE, max_iter = ^iter^,pca=F)
        output$Y=tsne$Y
        head(output$Y)
    }else if(""^algrthm^""=="UMAP"){
        cat("We are running UMAP","\n")
```

[10] Pour en savoir plus sur comment créer un complément JMP à partir d'un script JSL, consultez le lien suivant : jmp.com/content/dam/jmp/documents/en/academic/learning-library/01-add-in-builder.pdf.

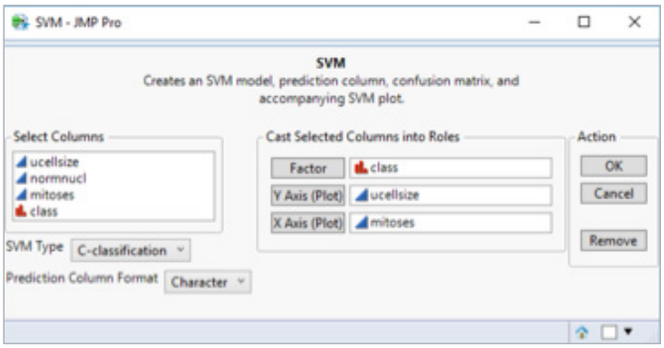
[11] Partagez-le avec la communauté d'utilisateurs JMP sur community.jmp.com afin que d'autres puissent l'utiliser.

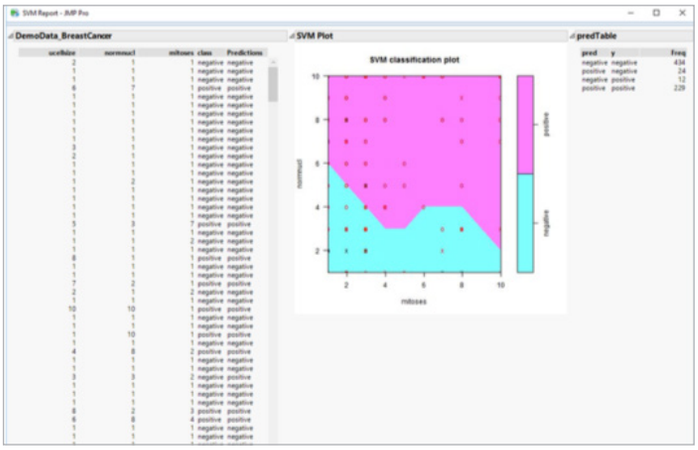
Une fois le code rédigé, les utilisateurs suivants n'auront plus du tout besoin ni de le voir, ni d'interagir avec lui. Cette fonctionnalité vous permet de le rédiger une bonne fois pour toutes et de l'intégrer dans un complément. Vous disposez désormais d'une interface cliquable pour appeler des fonctions R ou Python.

Pour en savoir plus et télécharger ce complément, consultez l'article suivant sur la communauté JMPcommunity.jmp.com/t5/JMP-Add-Ins/Data-visualization-with-t-SNE-and-UMAP/ta-p/177969.

Complément SVM :

Ce complément appelle la fonction SVM à partir du package R e1071[12]. Les utilisateurs interagissent avec cette boîte de dialogue afin d'obtenir le rapport de classification SVM[13] correspondant :





En revanche, les utilisateurs ne voient pas le code qui s'exécute en arrière-plan :

```
library(e1071)
library(gridExtra)

theText <- paste("x <- subset(dt, select=-", colnames(factor)[1], ")")
eval(parse(text=theText))
y <- factor
svm_model <- svm(x, y, type=svmType)
modelText <- paste("second.svm <- svm(", colnames(factor)[1], " ~ ., data = dt)")
eval(parse(text=modelText))

summary(svm_model)
summary(second.svm)
pred <- predict(svm_model, x)
pred <- as.vector(pred)

if(is.list(y)) {y <- unlist(y)}
predTable <- table(pred,y)
predTable <- as.data.frame(predTable)

eval(parse(text=paste(colnames(y_var)[1], "<- as.vector(as.matrix(y_var))")))
eval(parse(text=paste(colnames(x_var)[1], "<- as.vector(as.matrix(x_var))")))

max1 <- eval(parse(text=paste("max(", colnames(y_var)[1], ")")))
max2 <- eval(parse(text=paste("max(", colnames(x_var)[1], ")")))
gridSize <- eval(parse(text=paste("max(c(", max1, ",", max2, "))")))

eval(parse(text=paste("plot(second.svm, dt, ", colnames(y_var)[1], " ~ ",
colnames(x_var)[1], ", fill=TRUE, grid=gridSize)")))
```

Créer ce complément est encore plus facile à l'aide du Générateur de complément JMP pour R (voir l'exemple suivant). Le complément JMP pour R vous permet de sauter entièrement l'étape de programmation JSL et d'interagir simplement avec différentes boîtes de dialogue, en y déposant du code R ou Python.

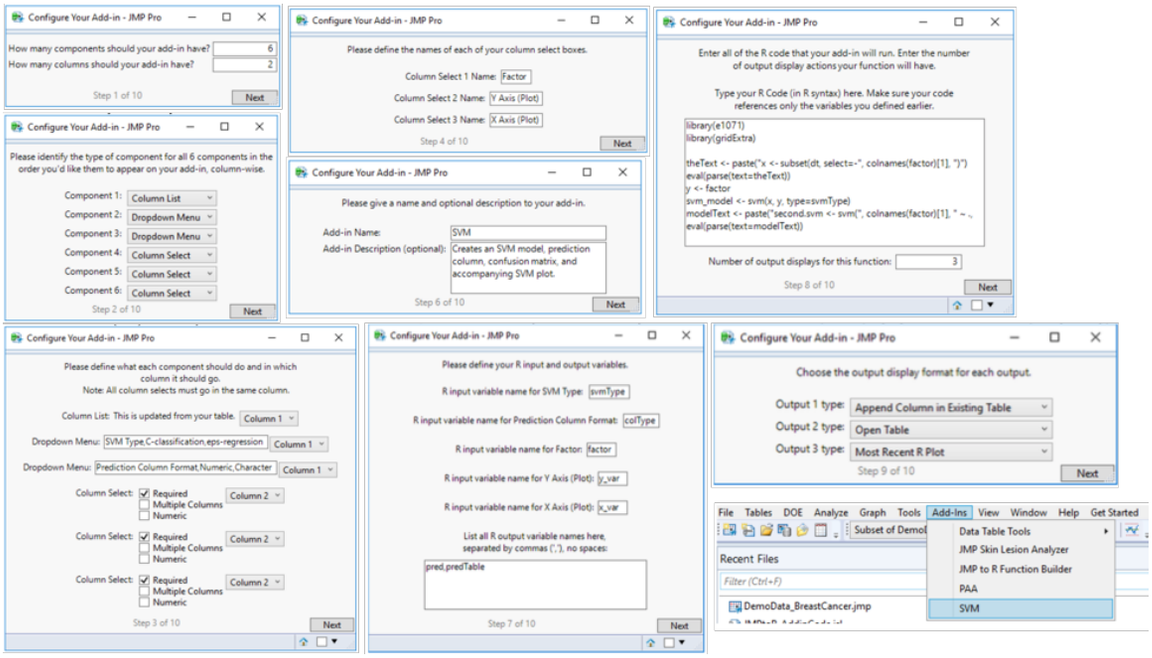
[12] Voir cran.r-project.org/web/packages/e1071/index.html.

[13] Le complément SVM a été ajouté à la suite de modélisation prédictive de JMP Pro 15. Encore une fois, cet exemple sert simplement à illustrer les avantages de la connexion à R. Les utilisateurs de JMP Pro 15 n'auront donc pas à réaliser cette opération.

Générateur de complément JMP pour R : Utiliser la connexion JMP à R sans JSL

Cet outil permet aux utilisateurs de définir et d'utiliser des compléments JMP personnalisés pour exécuter des fonctions R. Afin d'utiliser ce générateur d'interfaces, il vous faut tout d'abord créer le code R à envoyer à R sans toutefois devoir rédiger la moindre ligne de code JSL. De la même manière, les utilisateurs finaux de votre nouvelle interface cliquable n'auront pas besoin de voir ni d'utiliser ce code R. Ils verront simplement l'interface conviviale. (Remarque : ce complément fonctionne uniquement sur Windows, pas sur Mac).

Ce complément vous permet d'éviter la programmation JSL lorsque vous utilisez la connexion JMP à R. À la place, vous interagirez avec différentes boîtes de dialogue afin de créer les cases, les boutons et les invites que vous souhaitez ajouter à votre complément. Vous trouverez la plupart des étapes sur la page suivante :



L'exemple ci-dessus a été créé à l'aide du Générateur de complément JMP pour R.

Pour en savoir plus et télécharger le Générateur de complément JMP vers R, consultez l'article suivant sur la communauté JMP : community.jmp.com/t5/JMP-Add-Ins/The-JMP-to-R-Add-In-Builder/ta-p/43879.

Conclusion

Dans ce guide, nous vous avons présenté comment générer du code de scoring dans Python, mais aussi comment utiliser Python et R dans les scripts JMP. Nous avons également fourni des liens vers d'autres exemples, notamment des exemples avancés pour R faisant appel au Générateur de complément JMP. Pour en savoir plus, veuillez vous référer aux notes de bas de page et aux liens fournis tout au long de ce document. Si vous avez des questions ou besoin d'aide pour mettre en œuvre l'une de ces techniques, ou si vous souhaitez simplement nous faire part d'une découverte, envoyez un message à la communauté d'utilisateurs JMP à l'adresse suivante : community.jmp.com.

À propos de SAS et JMP

Leader mondial de l'analyse, SAS a créé JMP (prononcé « jump ») en 1989 pour permettre aux professionnels des données d'explorer et d'analyser les données de manière visuelle et interactive. Depuis, JMP s'est développé et se décline aujourd'hui en une gamme d'outils de découverte statistique qui répondent tous à des besoins spécifiques. John Sall, cofondateur et vice-président exécutif de SAS, est directeur de la division JMP.

SAS FRANCE Domaine de Grégy, Grégy-sur-Yerres, 77257 Brie Comte Robert Cedex +33 (1) 60 62 11 11

JMP est une solution logicielle développée par SAS Institute. Pour en savoir plus sur SAS, consultez le site **sas.com/fr**.
Pour contacter le service commercial JMP en France, composez le +33 (0)1 60 62 12 76 ou rendez-vous sur **jmp.com/fr**