# The New HTTP Request in JMP ® 14

JMP Discovery Conference – Cary - 2018

Bryan Boone - SAS

Government and private entities are increasingly making their data available over the internet via RESTful services. JMP 14 has a new feature called *HTTP Request*, available through JSL, that allows you to access these RESTful services.

This paper describes what RESTful services are and provides several examples on how to use *HTTP Request* to bring new data into JMP.

## What is REST

REST, or Representational state transfer, is a way of providing interoperability between computer systems on the Internet. A RESTful service is a web-based service that utilizes this paradigm. A RESTful API is an application program interface (API) that uses HTTP requests to **GET**, **POST**, **PUT** and **DELETE** data to interact with the RESTful service.

The HTTP protocol defines a set of request HTTP Methods to indicate the action a client would like a server to perform on a resource. They are:

| | |
|---|---|
| • **GET** | Request the resource |
| • **HEAD** | Request only resource metadata |
| • **POST** | Requests to accept the resource |
| • **PUT** | Requests to store or create the resource |
| • **DELETE** | Requests to delete the resource |
| • **PATCH** | Request to modify the state of a resource |

## New Terminology, Familiar Activities

This may seem like completely new concepts, but you are already familiar with two of the most common request methods, **GET** and **POST**.

### GET

**GET** has two personalities.

The first personality is just **GET**-ting a resource. That is, the intent is merely accessing a web resource. This can be seen in Figure 1. Here, the browser is accessing the webpage.

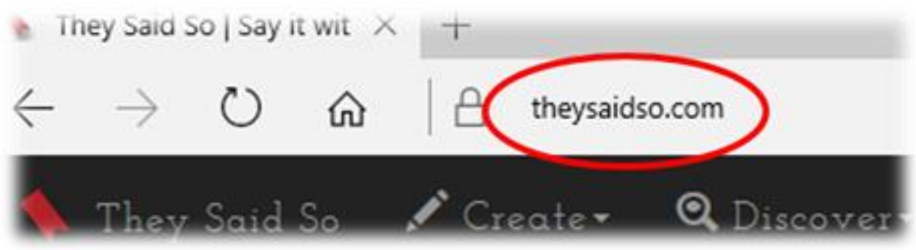
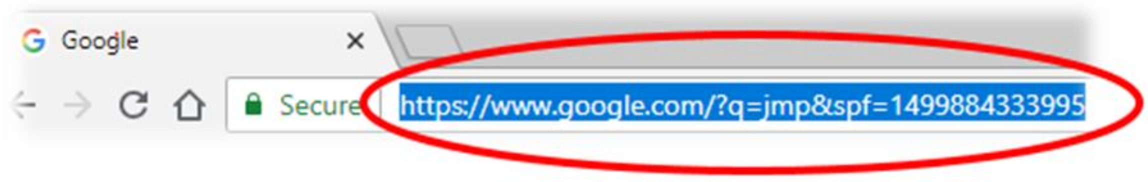


Figure 1. “**GET** for a web resource”

The second is **GET**-ting (accessing) a resource, along with what is known as a query string. A query string is a set of key/value pairs defined by the web resource and used by the client to request the server to filter or modify the **GET** request using these parameters. This can be seen in Figure 2. Here browser is asking the server to modify a **GET** request using:

- q=jmp
- spf=1499884333995



https://www.google.com/?q=jmp&spf=1499884333995

Figure 2. “**GET** using a query string for web resource”

## POST

**POST**, on the other hand, is more complex. A **POST** asks the server to change or create a resource. A **POST** typically contains an HTML Form which in turn contains a collection of key/value pairs of character data and/or key/value pairs of files. Figure 3 is an example that signs a user into Amazon. Here the key/value pairs are:

| | |
|---|---|
| • Email | Your email sign in |
| • Password | Your password |

The **“Sign in”** button performs the **POST**.

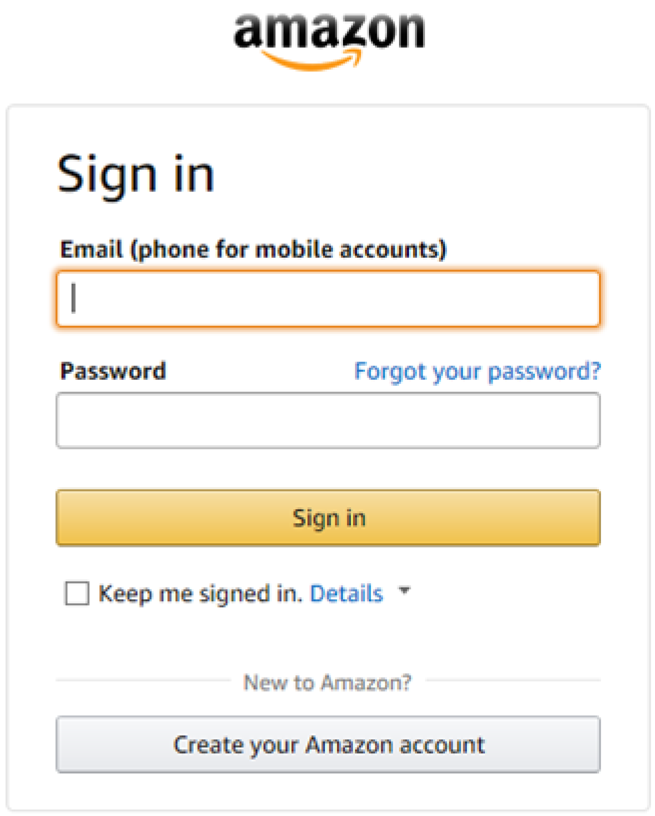

amazon
Sign in
Email (phone for mobile accounts)
Password
Forgot your password?
Sign in
Keep me signed in. Details
New to Amazon?
Create your Amazon account

Figure 3. “**POST** form data to a web resource”

## Why Use It

The HTTP protocol allows us to think of the information super highways as “The highways are paved; the street signs are in and bridges have been built.” The standard has allowed new data management companies to form. Additionally, the adoption of the HTTP standard has removed the information silos that are often found in established companies. Some of these companies include:

| | |
|---|---|
| Text-Processing.com (Sentiment Data) | Jenkins |
| Jira | The HUB |
| Office 365 | Google |
| Drop Box | One Drive |
| AWS (Amazon Web Services) | US Census Bureau |
| NASA | PubChem |
| New York Times | Fitbit |
| Yahoo | Bitcoin |
| Marvel Comics | SAS |
| OpenAQ | JMP |

## How To Use It

### JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is language independent but uses concepts are already familiar to most programmers.

JSON is built on two structures:

- A collection of name/value pairs. (dictionary, hash table, associative array, …)
- An ordered list of values. (array, vector, list, ...)

### New Terminology, Familiar Activities (again)

#### JSL

JSL is the scripting language of choice when JMP is involved. JSON may be a new term, but JSL supports the two data structures JSON requires.

#### *List*

*Lists* are containers to store items, such as the following:

- numbers
- variables
- character strings
- expressions (for example, assignments or function calls)
- matrices
- nested lists

#### *Associative Array*

- An *Associative Array* maps unique keys to values that can be non-unique. An associative array is also called a dictionary, a map, a hash map, or a hash table.

JSL also contains two functions that are specific to JSON processing.

#### *Parse JSON*

- Converts JSON text into an associative array or list based on the structure of the JSON data.

#### *As JSON Expr*

- Returns a JSON representation of the expression.

These functions allow the script writer to convert seamlessly to and from JSL containers and JSON.

## HTTP Request

*HTTP Request* is new in JMP 14. Originally, the technology was created to handle the interaction between JMP and JMP Public. It was soon realized that it had further applications than publishing data to JMP Public and the technology was surfaced in JSL as *HTTP Request*.

### HTTP Request Primer

*HTTP Request* seamlessly supports both http and https protocol and encapsulates much of the behavior that you'd expect from cURL.

"cURL is used in command lines or scripts to transfer data. It is also used in cars, television sets, routers, printers, audio equipment, mobile phones, tablets, settop boxes, media players and is the internet transfer backbone for thousands of software applications affecting billions of humans daily."

More importantly, you can think of *HTTP Request* as the glue that binds web resources to JSL. That is, *HTTP Request* communicates with a web resource and allows the script writer to process responses in JSL. The JSL complexity is controlled by the author.

The most common HTTP Methods that you use while interacting with a web resource are:

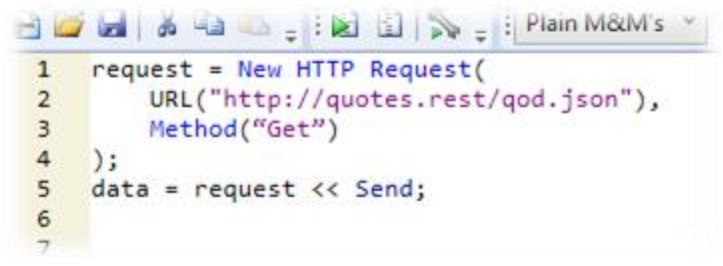
- **GET**
- **POST**

#### *Using* HTTP Request *in a GET action*

A **GET** action is probably the most common *HTTP Request* action. This action is a read-only action that can bring data into JMP.

Common messages for *HTTP Request*:

- *"Method"*
- *"URL"*
- *"Query String"*
- *"Send"*



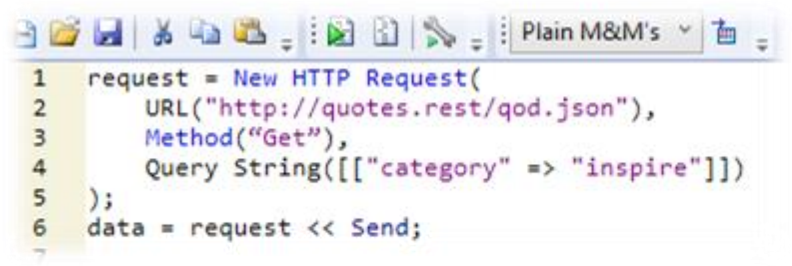
```
request = New HTTP Request(
    URL("http://quotes.rest/qod.json"),
    Method("Get")
);
data = request << Send;
```

Figure 4. "*HTTP Request* **GET**"

*HTTP Request* **GET** actions are often accompanied by a *"Query String"* to send parameters to a web resource. Figure 5 uses an *HTTP Request* with a *"Query String"*. In JSL, a *"Query String"* is an *Associative*

*Array* that maps a key to a character value or a *List* of character values. When the script in Figure 5 is run, the URL will be automatically expanded to:

http://quotes.rest/qod.json?category=inspire



```
request = New HTTP Request(
    URL("http://quotes.rest/qod.json"),
    Method("Get"),
    Query String([["category" => "inspire"]])
);
data = request << Send;
```

Figure 5. "*HTTP Request* **GET** using a '*Query String*'"

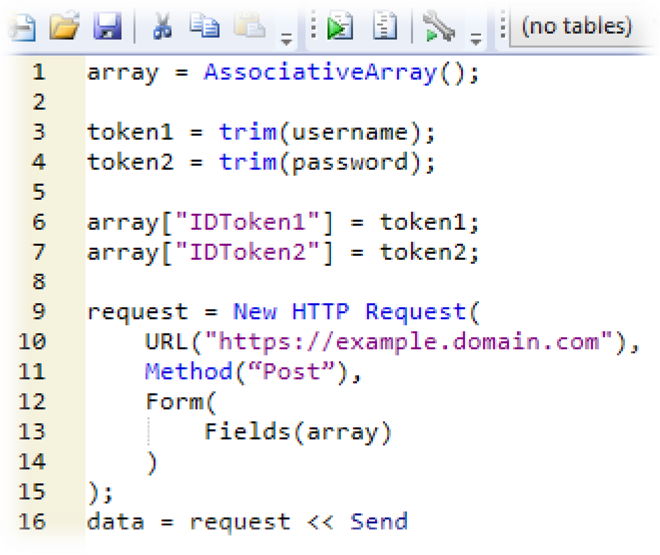
## *Using HTTP Request in a POST action*

*HTTP Request* **POST** actions are often more complex.

Common messages for *HTTP Request*:

- *"Method"*
- *"URL"*
- *"Form"*
  - *"Fields"*
  - *"Files"*
- *"JSON"*
- *"File"*
- *"Blob"*
- *"Send"*

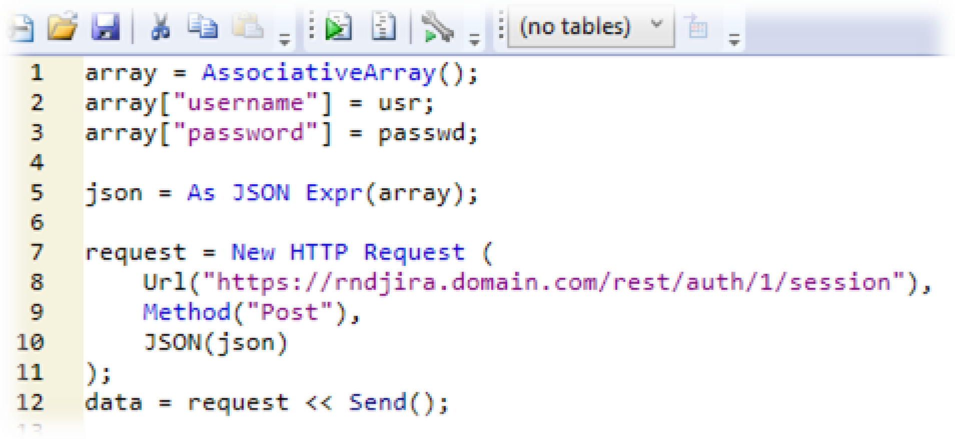
Figure 6 shows an *HTTP Request* "*Form*" data. In JSL, *"Form"* data can have two attributes, *"Fields"* and/or *"Files"*. They are both JSL *Associative Arrays* that map a key to a character value or a *List* of character values. In the case of the *"Files"*, the character value or *List* of character values are the filenames that you want to add to the "*Form*".



```
array = AssociativeArray();

token1 = trim(username);
token2 = trim(password);

array["IDToken1"] = token1;
array["IDToken2"] = token2;

request = New HTTP Request(
    URL("https://example.domain.com"),
    Method("Post"),
    Form(
        Fields(array)
    )
);
data = request << Send
```

Figure 6. "*HTTP Request* **POST** using a *Form*"

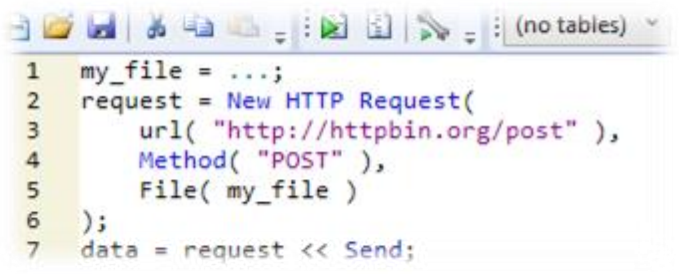
Figure 7 shows an *HTTP Request "JSON"* data. In JSL, *"JSON"* can be either an *Associative Array* or character data in JSON format. *As JSON Expr* will create a well-defined JSON object to use with *HTTP Request*. JSON data is sent as the body of the request.



```
array = AssociativeArray();
array["username"] = usr;
array["password"] = passwd;

json = As JSON Expr(array);

request = New HTTP Request (
    Url("https://rndjira.domain.com/rest/auth/1/session"),
    Method("Post"),
    JSON(json)
);
data = request << Send();
```

Figure 7. "*HTTP Request* **POST** using '*JSON*'"

Figure 8 shows an *HTTP Request "File"* data. In JSL, *"File"* is the filename you want to send as the body of the request.



```
my_file = ...;
request = New HTTP Request(
    url( "http://httpbin.org/post" ),
    Method( "POST" ),
    File( my_file )
);
data = request << Send;
```

Figure 8. “*HTTP Request* **POST** using ‘*File*’”

One last action for *HTTP Request* “*Blob”* data. In JSL, *“Blob”* is a binary blob data you want to send as the body of the request. This means you don’t have to write a file to disk before sending it as part of a request.

The **GET** action is the simplest request to make while **POST** is the most complex. The interaction between *HTTP Request* and the web resource is always defined by the server. Although the HTTP Protocol is a well-defined standard, the data (and data structures) are defined on a per web resource basis. That is, the API to One Drive is different than Dropbox, although they perform similar functions.

### *Return to Sender*

Data from an *HTTP Request* “*Send”* message can be either character data, binary blob data, or empty, depending upon web resource. Checking “*Is Empty”* on the returned data is the coarsest level of data investigation (besides none). However, if you need a fine-grained level of investigation you can use “*Is String”* to check for character data.

For even finer-grained level of investigation you can use *“Get MIME Type”*. Common MIME types can be found in the *Incomplete list of MIME types* article in the Reference section of the paper.

Finally, if needed, you can investigate low-level HTTP Status codes from a web resource using these messages:

| | |
|---|---|
| *“Has Information”* | 100-199 |
| *“Is Successful”* | 200-299 |
| *“Has Redirection”* | 300-399 |
| *“Has Client Error”* | 400-499 |
| *“Has Server Error”* | 500-599 |
| *“Has Error”* | Has either client or server error |
| *“Get Status”* | The HTTP Status code |

HTTP Status Codes descriptions can be found in the *Status Code Definitions* article in the Reference section of the paper.

## Where To Use It

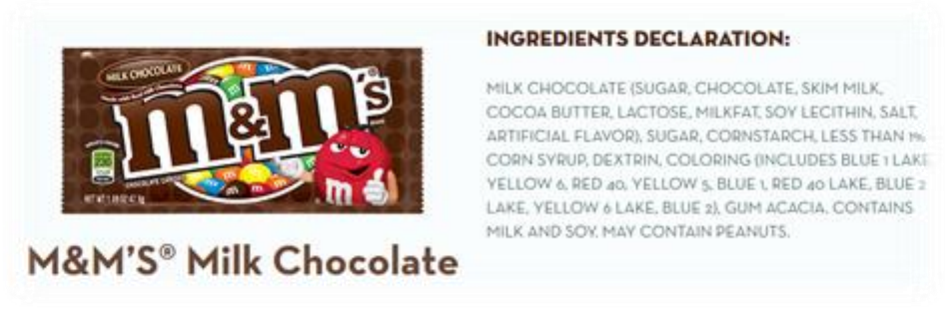
Although *HTTP Request* is capable of either character data or binary blob data from any web resource, we are going to focus on character data from RESTful services, namely JSON data.

### This Is Sweet

One of the new data management companies founded on the HTTP protocol standard that uses REST is PubChem. PubChem was formed in 2004 as an open chemistry data source at the National Institutes of Health (NIH). Hundreds of data sources from institutions such as universities, pharmaceutical companies, government agencies and chemical vendors are used to update the PubChem records each month. It is the largest collection of publicly available source of chemical information.

PubChem breaks down their data queries into substances, compounds, and assays, etc. From there, you can obtain a vast array of properties, descriptions, dose and other characteristics of interest.

I read that as M&M ingredients.



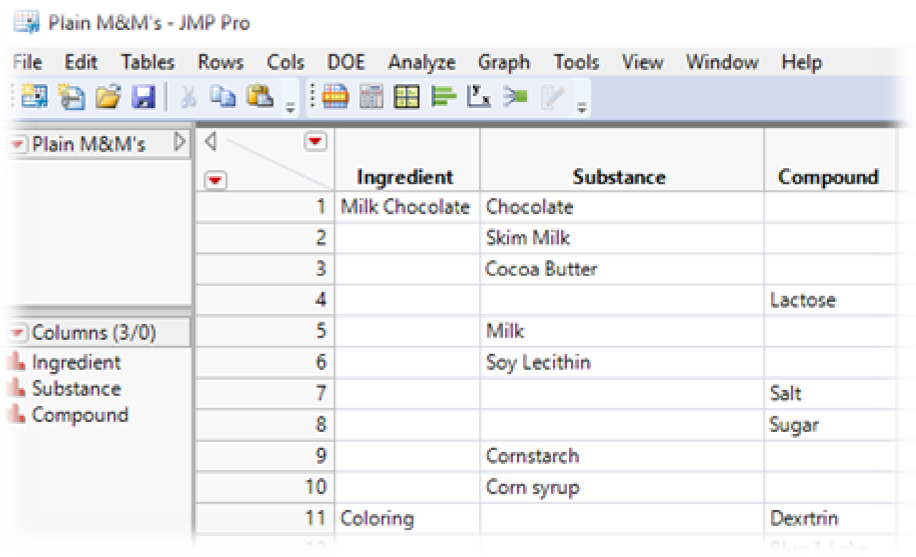
**INGREDIENTS DECLARATION:**

MILK CHOCOLATE (SUGAR, CHOCOLATE, SKIM MILK, COCOA BUTTER, LACTOSE, MILKFAT, SOY LECITHIN, SALT, ARTIFICIAL FLAVOR), SUGAR, CORNSTARCH, LESS THAN 1% CORN SYRUP, DEXTRIN, COLORING (INCLUDES BLUE 1 LAKE, YELLOW 6, RED 40, YELLOW 5, BLUE 1, RED 40 LAKE, BLUE 2 LAKE, YELLOW 6 LAKE, BLUE 2), GUM ACACIA. CONTAINS MILK AND SOY. MAY CONTAIN PEANUTS.

Figure 9. “Plain M&M ingredients”

I wrote a JSL script that created a JMP data table of the ingredients of Plain M&M’s to see what is in them. It was no surprise that the ingredients were heavy on the artificial coloring.

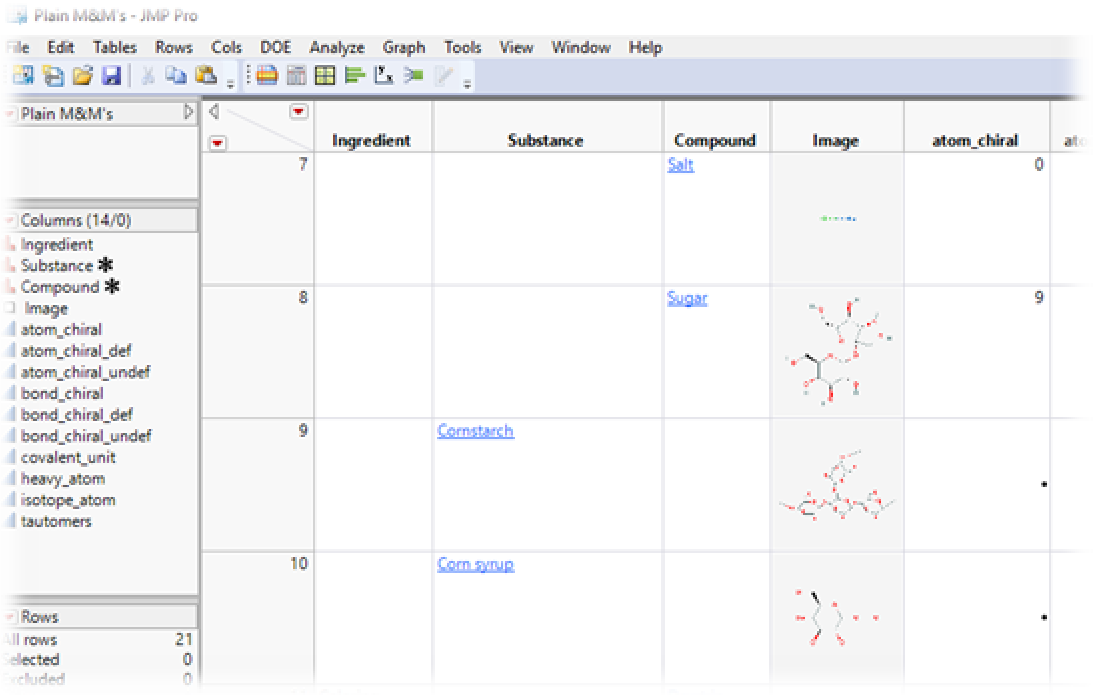
A JMP partial data table representation looks like this:



| | Ingredient | Substance | Compound |
|---|---|---|---|
| 1 | Milk Chocolate | Chocolate | |
| 2 | | Skim Milk | |
| 3 | | Cocoa Butter | |
| 4 | | | Lactose |
| 5 | | Milk | |
| 6 | | Soy Lecithin | |
| 7 | | | Salt |
| 8 | | | Sugar |
| 9 | | Cornstarch | |
| 10 | | Corn syrup | |
| 11 | Coloring | | Dexrtrin |

Figure 10. “Partial Plain M&M ingredient data table”

When the JSL script is run, *HTTP Request* is used to retrieve the substance or compound image as well as other properties provided by PubChem’s RESTful service.


Plain M&M's - JMP Pro
File Edit Tables Rows Cols DOE Analyze Graph Tools View Window Help
Plain M&M's

Columns (14/0)
- Ingredient
- Substance ✱
- Compound ✱
- Image
- atom_chiral
- atom_chiral_def
- atom_chiral_undef
- bond_chiral
- bond_chiral_def
- bond_chiral_undef
- covalent_unit
- heavy_atom
- isotope_atom
- tautomers

| Rows | |
|---|---|
| All rows | 21 |
| Selected | 0 |
| Excluded | 0 |

| | Ingredient | Substance | Compound | Image | atom_chiral |
|---|---|---|---|---|---|
| 7 | | | Salt | | 0 |
| 8 | | | Sugar | | 9 |
| 9 | | Cornstarch | | | • |
| 10 | | Corn syrup | | | • |

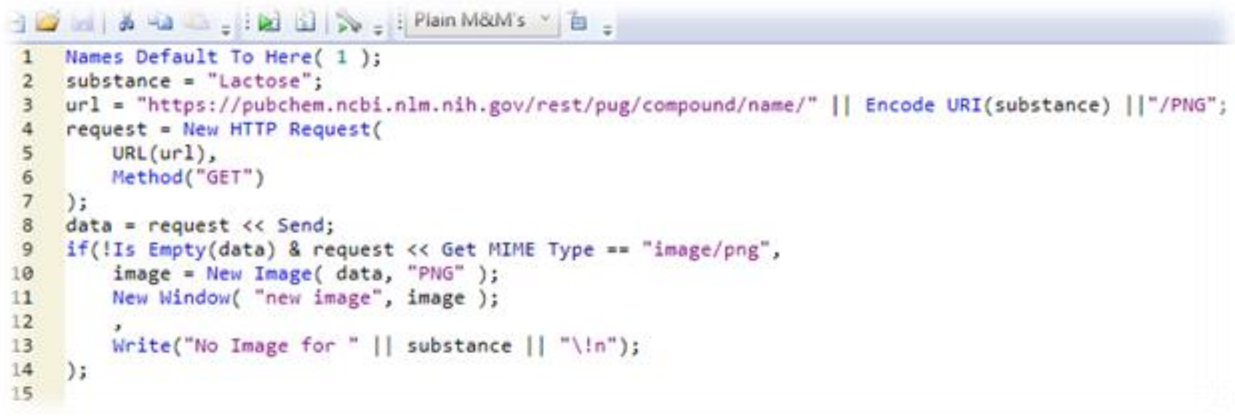
Figure 11. “Partial Plain M&M ingredient data table with PubChem data”

The RESTful service URLs of interest are:

- https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/name/[NAME]/PNG
- https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/[NAME]/PNG
- https://pubchem.ncbi.nlm.nih.gov/rest/pug/substance/name/[NAME]/JSON
- https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/[NAME]/JSON

(**[NAME]** is substance or compound name found in Figures 9 and 10.)

To retrieve the image data of a compound like Lactose, the *HTTP Request* looks like:



```
Names Default To Here( 1 );
substance = "Lactose";
url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/" || Encode URI(substance) ||"/PNG";
request = New HTTP Request(
    URL(url),
    Method("GET")
);
data = request << Send;
if(!Is Empty(data) & request << Get MIME Type == "image/png",
    image = New Image( data, "PNG" );
    New Window( "new image", image );
    ,
    Write("No Image for " || substance || "\!n");
);
```

Figure 12. “**GET**-ing the image of Lactose from PubChem”

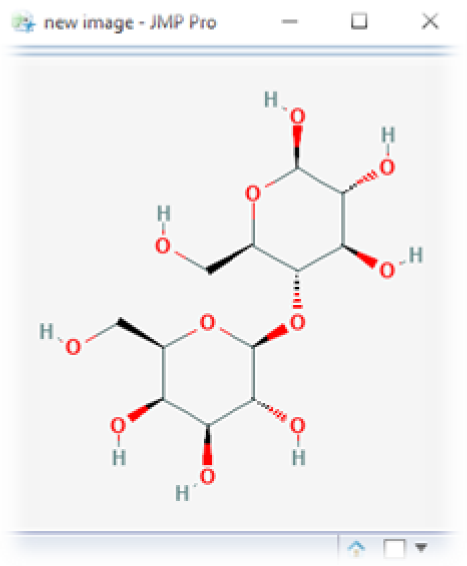
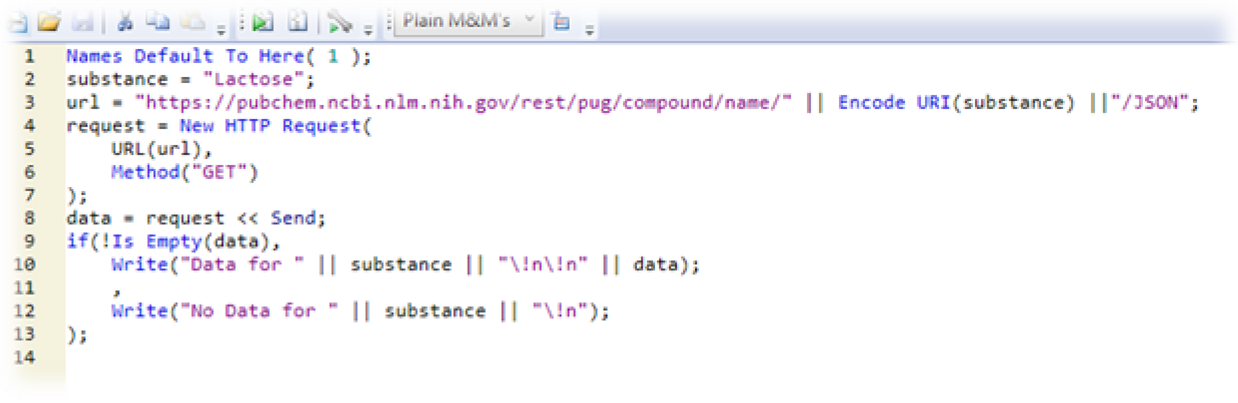


Figure 13. "**GET**-ing and displaying the image of Lactose from PubChem"

The Lactose request uses PubChem's endpoint for compound. There are two additional concepts used here. The first is *Encode URI*. URI encoding, also known as Percent Encoding, ensures that the RESTful URL contains only ASCII characters (luckily Lactose does). The second concept is *"Get MIME Type"*. The return from this PubChem endpoint is a binary blob data (a PNG picture). The *"Get MIME Type"* check ensures that the blob is indeed a PNG before it used with New Image since this endpoint will return character data if no PNG is found.

To retrieve the full record of Lactose in JSON data, the *HTTP Request* looks like:

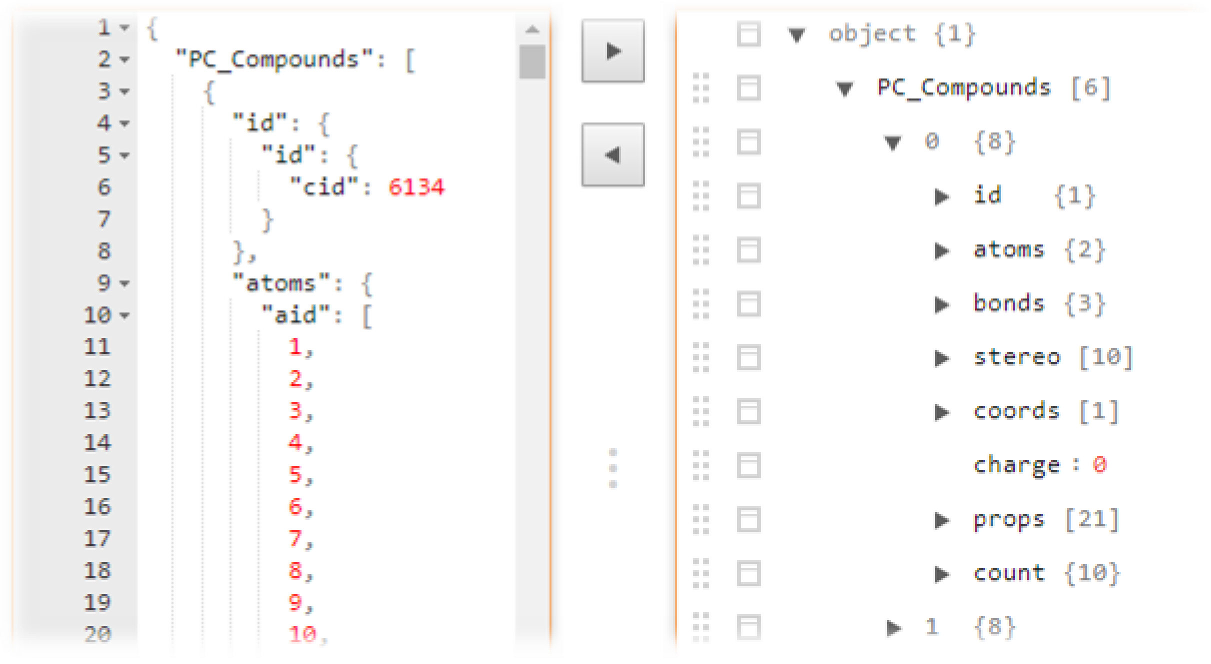


```
Names Default To Here( 1 );
substance = "Lactose";
url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/" || Encode URI(substance) ||"/JSON";
request = New HTTP Request(
    URL(url),
    Method("GET")
);
data = request << Send;
if(!Is Empty(data),
    Write("Data for " || substance || "\!n\!n" || data);
    ,
    Write("No Data for " || substance || "\!n");
);

```

Figure 14. "**GET**-ing the full record of Lactose from PubChem"

The endpoint defined by PubChem specifies that the **/JSON** suffix indicates the RESTful service should return JSON data, so checking the MIME type of *"data"* isn't necessary. However, we do have to process the JSON. There are [at least] two ways you can process the returned "*data*". The first is to use *"JSON To Data Table"*. This will try to create a JMP data table from the *"data"*. For rectangular or tabular data this works well (and fast). However, not all RESTful services return tabular data (PubChem is one that does not). The second way of processing returned data is *"Parse JSON"*. This creates a JSL *Associative Array*. This means you don't have to parse the JSON. However, you do need to know how the returned data is structured. That is, you'll need to know if it's a key/value pair or a key/values pair. This isn't as hard as it seems since you can capture the raw JSON in the JMP log and use any JSON editor to examine its structure.

A partial display of the JSON Data for Lactose looks like:



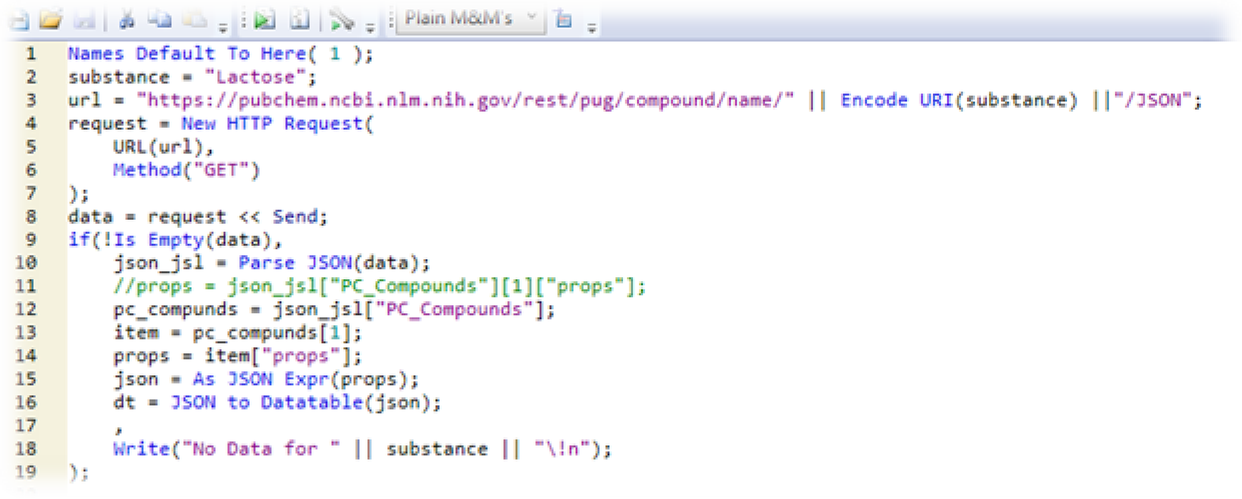
```
{
  "PC_Compounds": [
    {
      "id": {
        "id": {
          "cid": 6134
        }
      },
      "atoms": {
        "aid": [
          1,
          2,
          3,
          4,
          5,
          6,
          7,
          8,
          9,
          10,
```

```
▼ object {1}
    ▼ PC_Compounds [6]
        ▼ 0 {8}
            ▶ id    {1}
            ▶ atoms {2}
            ▶ bonds {3}
            ▶ stereo [10]
            ▶ coords [1]
              charge : 0
            ▶ props [21]
            ▶ count {10}
        ▶ 1 {8}
```

Figure 15. "Partial Lactose JSON data from PubChem"

This editor displays a side-by-side representation of raw JSON with corresponding JSON object. When you examine the data, a {..} in JSON, think JSL *Associative Array*. Likewise, when you encounter a [..] think JSL *List*.

Using this technique, you can modify the full record script shown in Figure 14 and use it to create a JMP data table for the properties of Lactose:

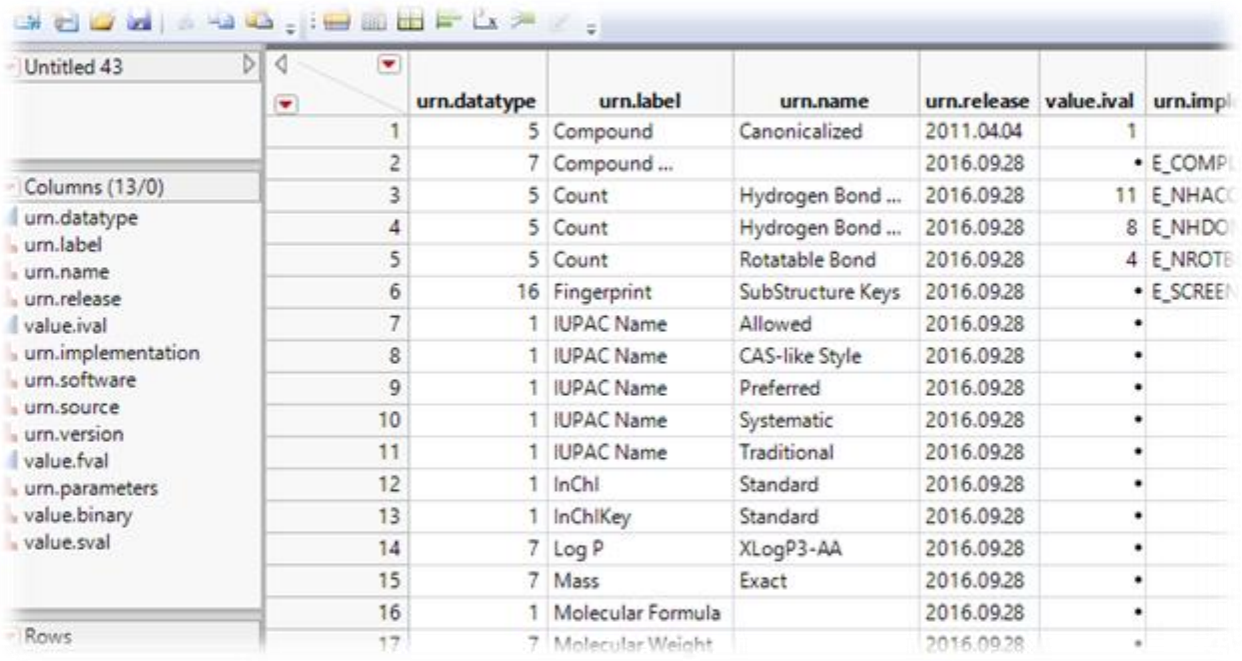


```
Names Default To Here( 1 );
substance = "Lactose";
url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/" || Encode URI(substance) ||"/JSON";
request = New HTTP Request(
    URL(url),
    Method("GET")
);
data = request << Send;
if(!Is Empty(data),
    json_jsl = Parse JSON(data);
    //props = json_jsl["PC_Compounds"][1]["props"];
    pc_compunds = json_jsl["PC_Compounds"];
    item = pc_compunds[1];
    props = item["props"];
    json = As JSON Expr(props);
    dt = JSON to Datatable(json);
    ,
    Write("No Data for " || substance || "\!n");
);
```

Figure 16. "**GET**-ing the full record of Lactose from PubChem and making "*props"* a data table"

This example uses *Parse JSON* to translate the raw JSON from PubChem to a JSL *Associative Array*. After investigating the JSON data structure, we realize that we are only interesting the "props" data in the Lactose JSON. We then can use the familiar JSL *Associative Array* navigation to retrieve the JSL *List* found at "PC_Compounds". We want only the first item in the list (since most of the "*props"* are repeated)

*"Props"* data is a JSL *Associative Array* so we can use *As JSON Expr* to translate *"props"* back to JSON and use it in *JSON to Datatable*. The result is in Figure 17.



Untitled 43

Columns (13/0)
urn.datatype
urn.label
urn.name
urn.release
value.ival
urn.implementation
urn.software
urn.source
urn.version
value.fval
urn.parameters
value.binary
value.sval

Rows

| | urn.datatype | urn.label | urn.name | urn.release | value.ival | urn.impl |
|---|---|---|---|---|---|---|
| 1 | 5 | Compound | Canonicalized | 2011.04.04 | 1 | |
| 2 | 7 | Compound ... | | 2016.09.28 | • | E_COMP |
| 3 | 5 | Count | Hydrogen Bond ... | 2016.09.28 | 11 | E_NHAC |
| 4 | 5 | Count | Hydrogen Bond ... | 2016.09.28 | 8 | E_NHDO |
| 5 | 5 | Count | Rotatable Bond | 2016.09.28 | 4 | E_NROTB |
| 6 | 16 | Fingerprint | SubStructure Keys | 2016.09.28 | • | E_SCREE |
| 7 | 1 | IUPAC Name | Allowed | 2016.09.28 | • | |
| 8 | 1 | IUPAC Name | CAS-like Style | 2016.09.28 | • | |
| 9 | 1 | IUPAC Name | Preferred | 2016.09.28 | • | |
| 10 | 1 | IUPAC Name | Systematic | 2016.09.28 | • | |
| 11 | 1 | IUPAC Name | Traditional | 2016.09.28 | • | |
| 12 | 1 | InChI | Standard | 2016.09.28 | • | |
| 13 | 1 | InChIKey | Standard | 2016.09.28 | • | |
| 14 | 7 | Log P | XLogP3-AA | 2016.09.28 | • | |
| 15 | 7 | Mass | Exact | 2016.09.28 | • | |
| 16 | 1 | Molecular Formula | | 2016.09.28 | • | |
| 17 | 7 | Molecular Weight | | 2016.09.28 | • | |

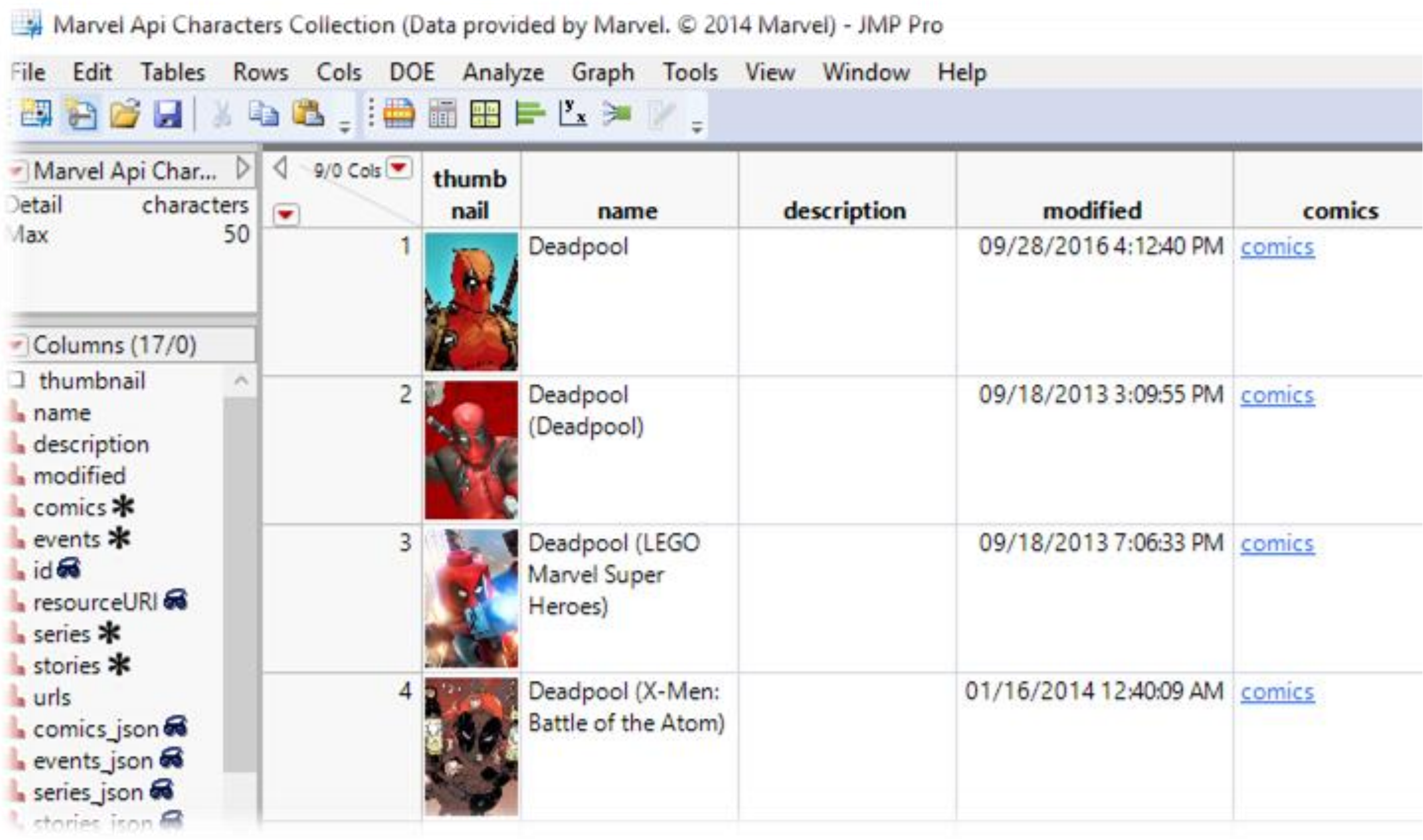
Figure 17. "Partial Lactose properties data table"

Remember, *HTTP Request* is just glue that binds web resources to JSL. Once you get your data into JSL, you can use it as simple or complex as according to your needs.

## Something To Marvel At

In 2014, Marvel Comics created a RESTful service that allows users to access 70 years of comic information. The Marvel Comics API is limited only by the scripter's imagination, so of course I had to register for a key and tap into it to see what I could find. I wrote a *marvel:API* JSL function that uses various columns that spiders through the Marvel Universe.

Being the Deadpool fan that I am I came with an initial query that created this data table:


Marvel Api Characters Collection (Data provided by Marvel. © 2014 Marvel) - JMP Pro

File Edit Tables Rows Cols DOE Analyze Graph Tools View Window Help

Marvel Api Char...

| | |
|---|---|
| Detail | characters |
| Max | 50 |

Columns (17/0)

- thumbnail
- name
- description
- modified
- comics ✱
- events ✱
- id
- resourceURI
- series ✱
- stories ✱
- urls
- comics_json
- events_json
- series_json
- stories_json

9/0 Cols

| | thumb nail | name | description | modified | comics |
|---|---|---|---|---|---|
| 1 | | Deadpool | | 09/28/2016 4:12:40 PM | comics |
| 2 | | Deadpool (Deadpool) | | 09/18/2013 3:09:55 PM | comics |
| 3 | | Deadpool (LEGO Marvel Super Heroes) | | 09/18/2013 7:06:33 PM | comics |
| 4 | | Deadpool (X-Men: Battle of the Atom) | | 01/16/2014 12:40:09 AM | comics |

Figure 18. "Partial Deadpool data table."

This unmasked 4 different Deadpool characters in the Marvel Comics Universe. I was interested in the identity of the first character and selected the series click event handler I had created. Given the origin of the click, the *marvel:API* function can spawn a new query to Marvel where the return value illustrates where Deadpool crosses over in the MCU. I limited the results to 50 rows; however, I was able to discover some interesting results. I did a quick Distribution Analysis on "*title*" and "*startYear*" just to see when Deadpool's popularity rose and fell (and rose again).

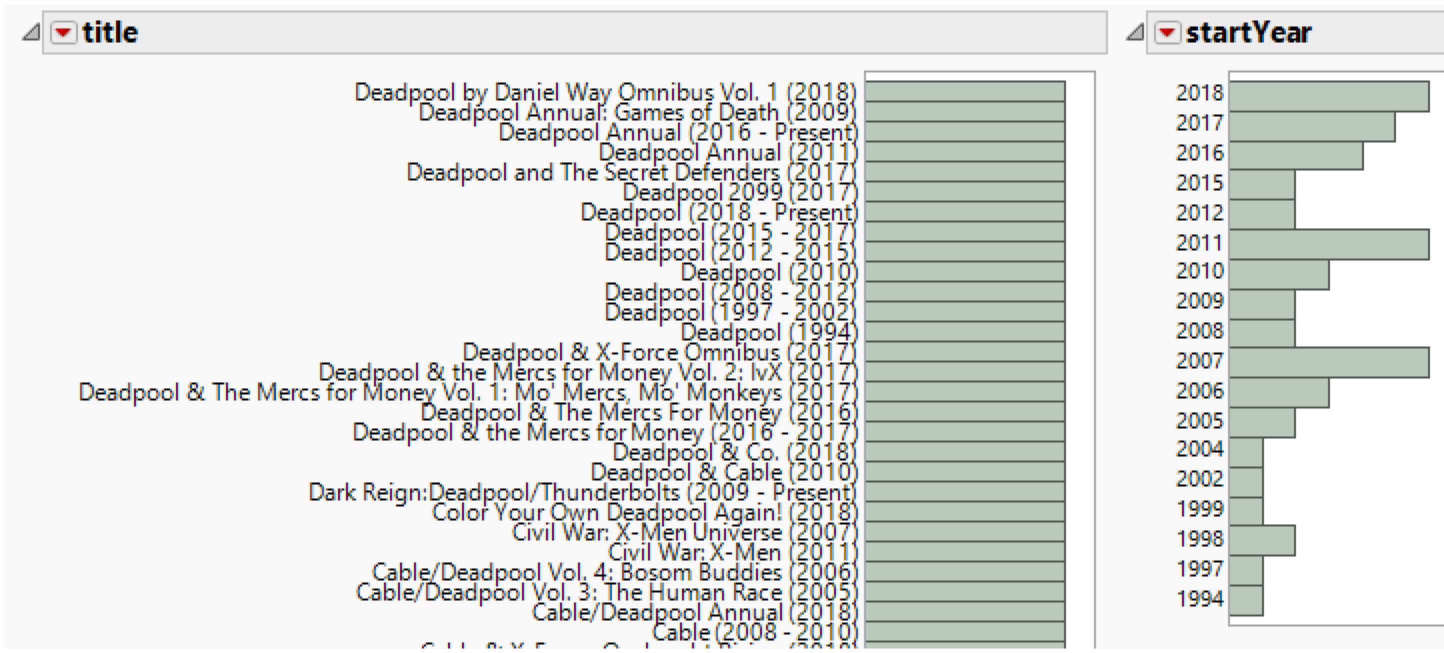


Figure 19. "Partial Deadpool Distribution."

Figure 19 shows that in 2007 Deadpool crossed over into many different Marvel Comic titles including:

- Avengers: The Initiative (2007-2010)
- Cable & Deadpool
- Civil War: X-Men Universe.

Interest in Deadpool waned again after 2007 but picked up again (as seen the analysis) after Ryan Reynolds first played the character in X-Men Origins: Wolverine (2009) so renewed interest in the character, reaching a peak in 2011.

In 2012-2015, the Marvel Comic Universe blows up in Deadpool Kills the Marvel Universe for a Marvel reboot.

Finally, no surprise for 2016-2018. That's when the movies came out.

Peter from Deadpool 2 sums up the Marvel Comics API the best, "I just saw the ad and thought it looked fun".

## Conventions used

The term RESTful service is used when the web resource is REST based

The term web resource is a generic term used when generic web access is implied.

HTTP actions are **bolded**

Buttons/Dialog controls are "**quoted and bolded**"

Hyperlinks are underlined

JSL Objects and Functions are *italicized*

JSL Variables/Columns/Messages names are "*quoted and italicized*"

## References

Hawk, Sandro (2011, December 16) *REST* retrieved from https://www.w3.org/2001/sw/wiki/REST

W3.org *Method Definitions* retrieved from https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

W3.org *Status Code Definitions* retrieved https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Mozilla.org (2018, May 13) *HTTP request methods* retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

JSON.org (2017, December) *Introducing JSON* retrieved from https://www.json.org

JMP.com (2018, July 12) *Lists* retrieved from https://www.jmp.com/support/help/14/lists.shtml

JMP.com (2018, July 12) *Work with Associative Arrays* retrieved from https://www.jmp.com/support/help/14/work-with-associative-arrays.shtml

*CURL command line tool and library for transferring data with URLs* retrieved from https://curl.haxx.se/

W3.org (2015, January 22) *Securing the Web* retrieved from https://www.w3.org/2001/tag/doc/web-https

W3.org (2014, June 11) *HTTP - Hypertext Transfer Protocol* retrieved from https://www.w3.org/Protocols/

Mozilla.org (2018, July 3) *Incomplete list of MIME types* retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Complete_list_of_MIME_types

PubChem.ncbi.nlm.nih.gov *About PubChem* retrieved from https://pubchemdocs.ncbi.nlm.nih.gov/

MMs.com *Ingredients* retrieved from http://www.mms.com/us/nutrition

W3Schools.com *HTML URL Encoding Reference* retrieved from https://www.w3schools.com/tags/ref_urlencode.asp

Developer.marvel.com *Developer Portal* retrieved from https://developer.marvel.com/