Paper JMP-xx

Web Scraping with JMP for Fun and Profit

Michael Hecht, SAS Institute Inc., Cary, NC

Abstract

JMP includes powerful tools for importing data from web pages, sometimes referred to as *web scraping*[1]. This paper presents a case study that retrieves Macintosh OS X[2] usage share data from the web, and transforms it into a JMP graph showing usage changes over time. When combined with JMP's built-in formulas, value labels, and summarization methods, the end result is a tool that can be used to quickly evaluate and make decisions based on OS usage trends.

Introduction

In the course of developing JMP, we need to make decisions on what operating system versions we will support. We wish we could support every version of an operating system, but in reality that is not feasible. When it is time to release a new version of JMP, we must decide if we will continue to support an older OS version, or if we will raise the minimum requirement[3].

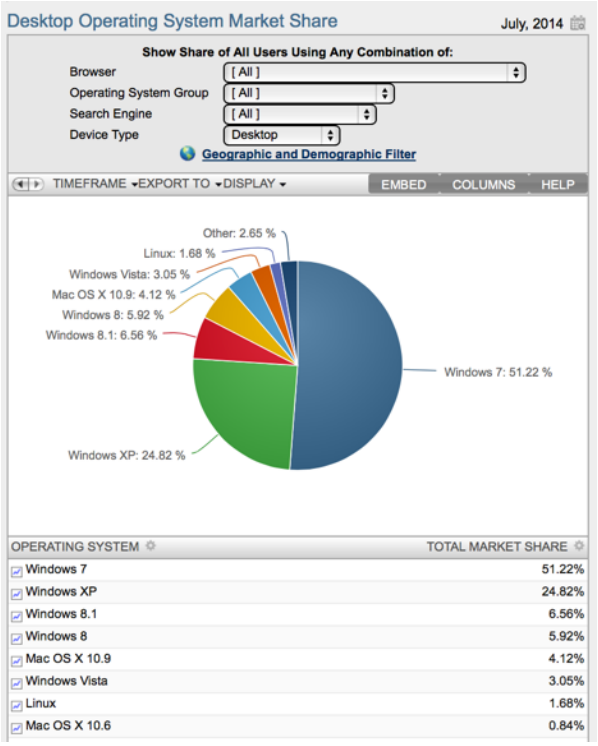
The decision to drop support for an OS version is not made lightly — we don't want to inconvenience our customers. But we cannot support older systems forever because this prevents us from adopting new technologies. Therefore, we want our decision to be informed, and that requires data. The data we want is an estimate of the percentages of OS versions in use by our customers. Moreover, we would like to see the historical usage of OS versions so we can predict how they will have changed when our next version of JMP is scheduled for release. Our respect for our customer's privacy prevents us from intrusively gathering OS version data while our customers use JMP. So we instead resort to a survey.

But we don't want to conduct the survey ourselves. Instead, we will access data from one of the various *network monitoring companies*. These companies monitor network traffic to a number of major internet web sites, and record the OS versions reported by the visitors' web browsers. So right away, we are assuming some things: that the OS versions used by our JMP customers is analogous to those used by the population as a whole; and that our customers will access the web with the same machines they use to run JMP.

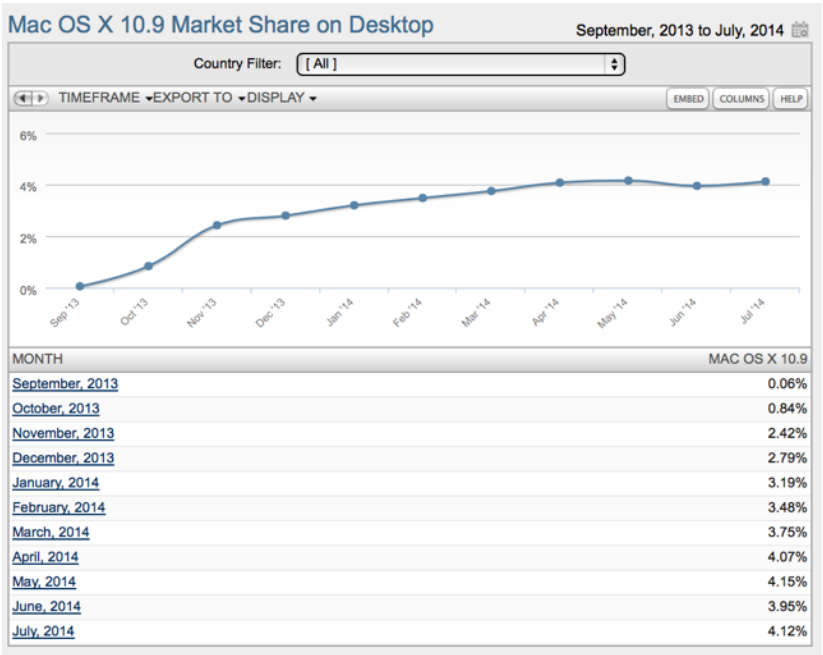
NetMarketShare℠

We collect our *usage share*[4] data from NetMarketShare[5]. This network monitoring company collects data from over 40,000 websites around the globe, ensuring that the visitor data they collect is unique and not fraudulent. Although they can provide detailed usage share statistics for a fee, the information we seek is freely available on their web pages.

If we go to their website and drill down to their Desktop Operating System by Version report, we see something like this:



This is a good chart, because it breaks down the usage share by operating system *and* version. But it doesn't show us trends over time. We can drill down to a specific version, like Mac OS X 10.9:



Here we see the trend for this OS version over the past 11 months. But we don't see how it relates to the other versions of OS X that are in current use.

Goal

We want to import the data for all Mac OS X versions from this website into JMP; then use JMP's graphing capabilities to produce the graph we want. Furthermore, we would like to view more than just 11 months of data. We would like to collect more data from NetMarketShare over time and accumulate it all into a JMP data table.

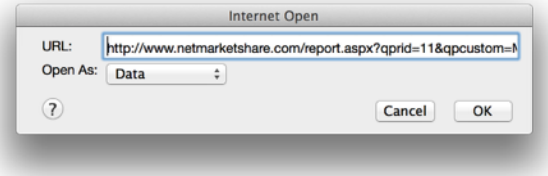
Step 1: Importing

The first step is to import the raw data from the NetMarketShare web pages into JMP. Fortunately, JMP has a built-in tool for doing just that[6].

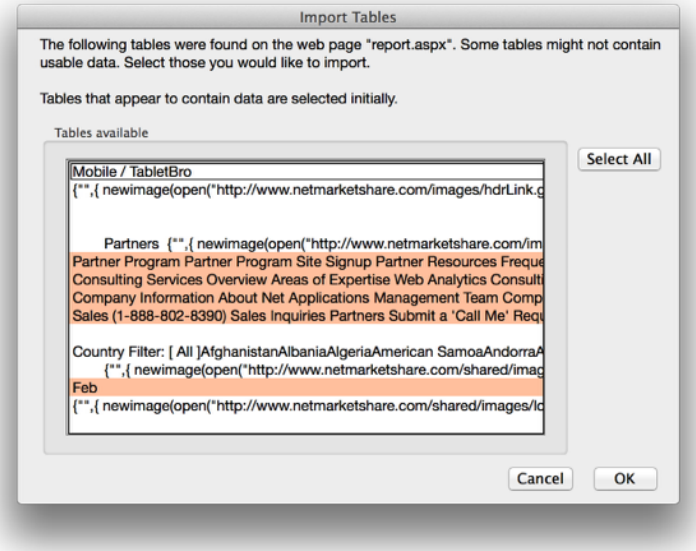
The Mac OS X 10.9 page has a specific URL that we can access from JMP. It is:

http://www.netmarketshare.com/report.aspx?qprid=11&qpcustom=Mac+OS+X+10.9

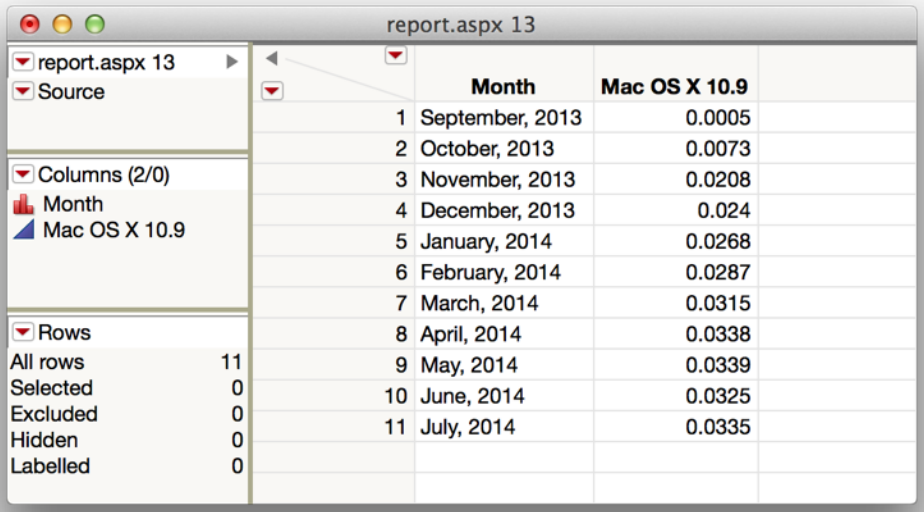
The URL may not be pretty, but it lets us access the data. In JMP, use the **Internet Open** command on the **File** menu, paste the URL into the dialog and click "OK".



JMP accesses the web page at netmarketshare.com and analyzes it, looking for <TABLE> elements within the page. It presents its findings in this dialog.



Of the 14 tables found, most of them are used to format page elements and do not contain useful data. The one we're interested in is Table 13, which shows "Feb" in the dialog list. If we select only that one and click "OK", JMP imports the data into this table, which is precisely what we want.



Import One Table
We can perform the same operation with JSL. If we examine the "Source" table property, we see code like this.

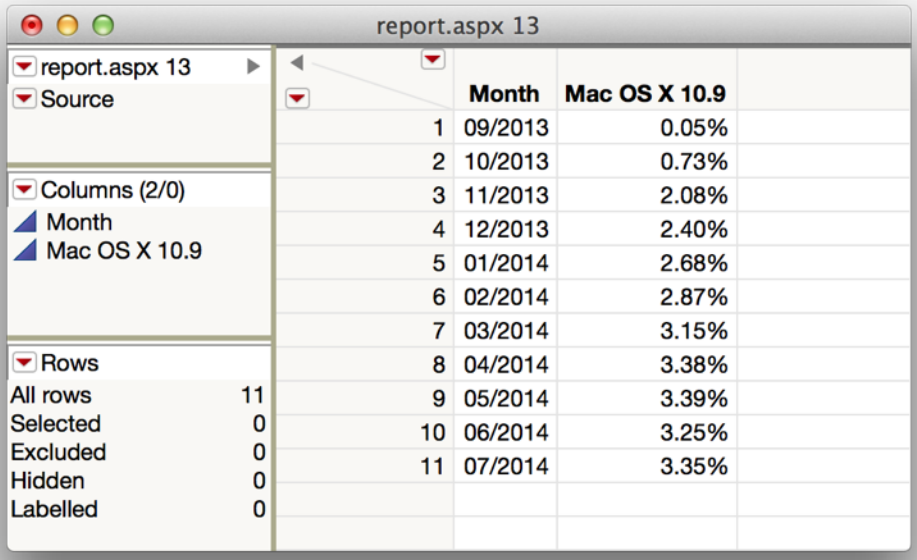
```
// Import a table from NetMarketShare
dt = Open(
       "http://www.netmarketshare.com/report.aspx?qprid=11&qpcustom=Mac+OS+X+10.9",
       HTML Table( 13 )
);
```

The **Internet Open** command just uses the JSL **Open()** function with a URL instead of a path. The **HTML Table()** option lets us specify that we want table #13 from the web page.

Once we've scraped this data, let's clean it up with a bit more code. We'll convert the *Month* column from character data into a standard numeric JMP date, and we'll use the *Percent* format for the *Mac OS X 10.9* column, which contains the *market usage share* (the usage share of this OS X version across the entire OS market).

```
// Map month names to numbers
monthmap = [
	"January" => "01",
	"February" => "02",
	"March" => "03",
	"April" => "04",
	"May" => "05",
	"June" => "06",
	"July" => "07",
	"August" => "08",
	"September" => "09",
	"October" => "10",
	"November" => "11",
	"December" => "12"
];

// Change the Date column to a date format that JMP will recognize: MM/YYYY
For( r = 1, r <= N Rows( dt ), r++,
	val = Column( dt, 1 )[ r ];
	If( val == "", Continue());
	w = Words( val, ", " );
	Column( dt, 1 )[ r ] = Eval Insert( "^monthmap[w[1]]^/^w[2]^" );
);

// Convert the Date column to a numeric value with the "M/Y" date format
Column( dt, 1 ) << Data Type( Numeric ) << Format( "M/Y", 7 ) << Set Modeling
Type( Continuous );

// Use the "Percent" format for the usage share column
Column( dt, 2 ) << Format( "Percent", 12, 2 );
```

Now our data table looks like this.



Import All Tables

From this starting point, we can import the NetMarketShare data for all OS X versions using a loop. (Note that NetMarketShare uses "Mac OS X 10.1" for Apple's most recent version: OS X 10.10 Yosemite.)

```
// Configuration

// The base URL for accessing OS X data from NetMarketShare
base url = "http://www.netmarketshare.com/report.aspx?qprid=11&qpcustom=Mac+OS+X+";

// Append one of these variants to the base URL to get the full URL
variant list = {
	"10.1", "10.9", "10.8", "10.7",
	"10.6", "10.5", "10.4", "(no+version+reported)" };

// Data is in this HTML table
html table i = 13;

// Map month names to numbers...


// Import all variants of Mac OS X from NetMarketShare
For( v = 1, v <= N Items( variant list ), v++,
	Write( Eval Insert( "\!NImporting data for ^variant list[v]^" ));
	dt = Open(
		Eval Insert( "^base url^^variant list[v]^" ),
		HTML Table( html table i )
	);

	// Change the Date column...
);
```

We create a JSL list named *variant list*, which holds the part of the URL that is different for each page we want to import. Next, we simply loop through the items of this list. For each variant, we write a message to the log and use **Open()** to import that variant. Both of these operations use the handy **Eval Insert()** function. This function returns its input string, after it evaluates each expression delimited by the caret symbols (^) and replaces that text with the result. So *^base url^* is replaced by the value of *base url*, and *^variant list[v]^* becomes the variant for this loop iteration. We follow the JMP convention of writing each line to the log starting with a newline sequence ("\!N").

This does the job — sort of. When we run it, we get eight data table windows, one for each variant we're importing. We really need to join all these data files into a single data table; and we don't need to see all the intermediate data tables in the process!

The first web page we import should become the beginning of our final data table. All others imported after that should be joined to it, adding their OS variant's usage data as an additional column. So we need a way to distinguish the first table from the rest. Here's how.

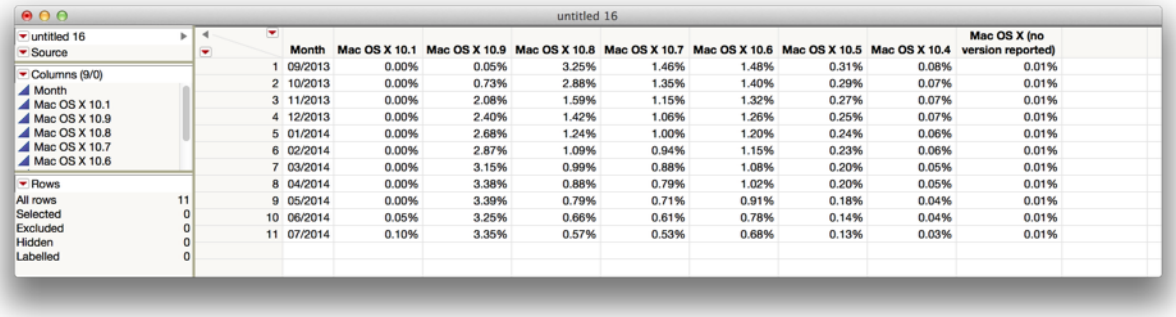
```
// Configuration...


// Import all variants of Mac OS X from NetMarketShare
dt = Empty();

For( v = 1, v <= N Items( variant list ), v++,
       Write( Eval Insert( "\!NImporting data for ^variant list[v]^" ));
       adt = Open(
              Eval Insert( "^base url^^variant list[v]^" ),
              HTML Table( html table i ),
              Invisible
       );

       // Change the Date column...

       // Join this table to dt join

       If( Is Empty( dt ),
              // First import
              dt = adt;
       ,
              // All subsequent imports
              jdt = dt << Join( With( adt ), Merge Same Name Columns,
                     By Matching Columns( :Month = :Month ),
                     Drop multiples( 0, 0 ), Name( "Include non-matches" )( 0, 0 ),
                     Invisible
              );

              // Done with dt, adt
              Close( dt, No Save ); Close( adt, No Save );

              // jdt is the new dt; get rid of the Match Flag column
              dt = jdt; dt << Delete Column( Match Flag );
       );
);

dt << New Data View;              // For debugging only
```

Each import places the reference to the scraped data table in the variable *adt*. Note the addition of the *Invisible* option on the **Open()**. That option tells JMP to not go to the effort to create a window for this data table. It imports the data and creates all the usual data table structures in memory, it just doesn't actually show us the table in a window. This actually speeds up the script's execution quite a bit!

The next **If()** determines whether this is the first import or a subsequent import. It doesn't work to use **If(** *dt* == **Empty()**, ... **)**. Instead, we must use the **Is Empty()** function to test for that. If true, this is the first import so we just move *adt* to *dt*.

The second half of the **If()** is used for all imports after the first one, because now *dt* is no longer empty. For these imports, we use the **<< Join** message to create a new data table *jdt*, with the columns from *dt* and *adt* joined together. Join is complex and has lots of options that control how the rows from its input tables are combined[7]. We won't go into those here. But after the join all our data is now in *jdt.* We don't need *dt* or *adt* anymore, so we close them. Now, *jdt* becomes the new *dt*. We delete the **Match Flag** column which the join added, and repeat for the next OS variant.

Once outside the loop, *dt* holds a reference to the data table that has all of our joined data in it. But it's still invisible. So we send it the **<< New Data View** message to tell JMP that it's time to make a window. This is just a temporary step for debugging, so we can see our progress along the way. This is what the final result looks like.



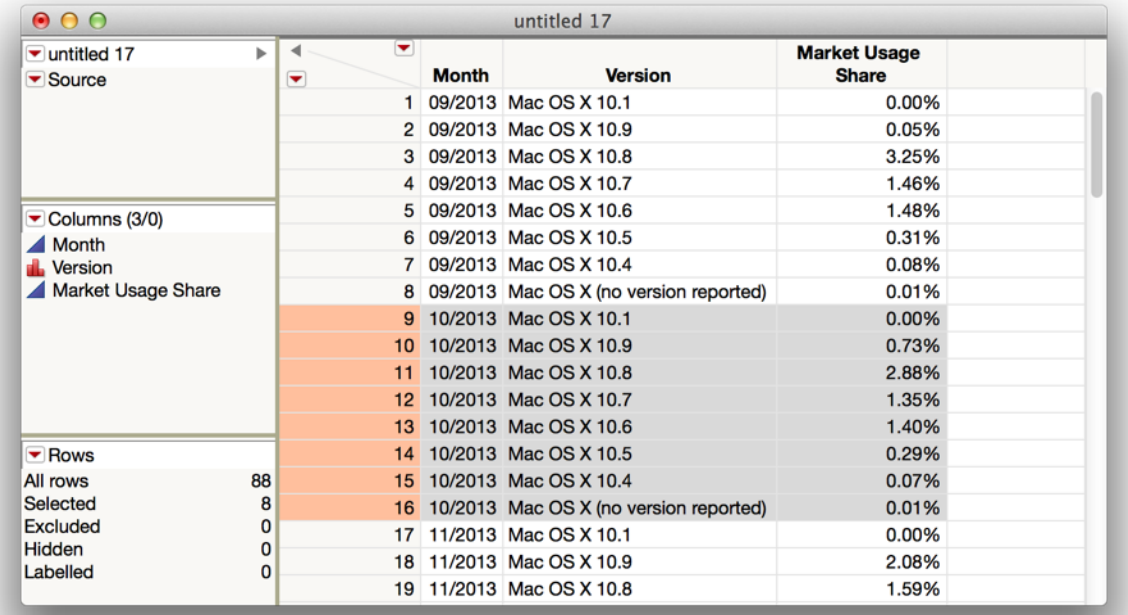
	Month	Mac OS X 10.1	Mac OS X 10.9	Mac OS X 10.8	Mac OS X 10.7	Mac OS X 10.6	Mac OS X 10.5	Mac OS X 10.4	Mac OS X (no version reported)
1	09/2013	0.00%	0.05%	3.25%	1.46%	1.48%	0.31%	0.08%	0.01%
2	10/2013	0.00%	0.73%	2.88%	1.35%	1.40%	0.29%	0.07%	0.01%
3	11/2013	0.00%	2.08%	1.59%	1.15%	1.32%	0.27%	0.07%	0.01%
4	12/2013	0.00%	2.40%	1.42%	1.06%	1.26%	0.25%	0.07%	0.01%
5	01/2014	0.00%	2.68%	1.24%	1.00%	1.20%	0.24%	0.06%	0.01%
6	02/2014	0.00%	2.87%	1.09%	0.94%	1.15%	0.23%	0.06%	0.01%
7	03/2014	0.00%	3.15%	0.99%	0.88%	1.08%	0.20%	0.05%	0.01%
8	04/2014	0.00%	3.38%	0.88%	0.79%	1.02%	0.20%	0.05%	0.01%
9	05/2014	0.00%	3.39%	0.79%	0.71%	0.91%	0.18%	0.04%	0.01%
10	06/2014	0.05%	3.25%	0.66%	0.61%	0.78%	0.14%	0.04%	0.01%
11	07/2014	0.10%	3.35%	0.57%	0.53%	0.68%	0.13%	0.03%	0.01%

Step 2: Stacking

We have now scraped all the web data from NetMarketShare into a single data table. But this data table needs to be reshaped so we can use it for graphing and analysis. We need a single column of all the market usage share percentages. For each month we will have multiple rows, one for each OS X version. We can do this reshaping with the **<< Stack** message.

```
// Build a list of all the market usage share columns
col list = {};
For( c = 2, c <= N Cols( dt ), c++,
	Insert Into( col list, Column( dt, c ));
);

// Stack these columns into a single column, along with a source label
// to identify the OS X version
dt stack = dt << Stack(
	Columns( col list ),
	Source Label Column( "Version" ),
	Stacked Data Column( "Market Usage Share" ),
	Invisible
);
Close( dt, No Save );

dt stack << New Data View;		// For debugging only
```

The **<< Stack** message requires a list of columns. But we don't want to specify all the market usage share columns explicitly. That would make our script more difficult to maintain when a new OS variant is introduced. Instead, we build a list of the columns dynamically in *col list*, starting at column 2. Then we can stack the data into a new data table referenced by *dt stack*. At this point, we're done with *dt* so we close it. We continue to use invisible data tables for efficiency; so as before the **<< New Data View** is used temporarily to show us the result of this intermediate step.



	Month	Version	Market Usage Share
1	09/2013	Mac OS X 10.1	0.00%
2	09/2013	Mac OS X 10.9	0.05%
3	09/2013	Mac OS X 10.8	3.25%
4	09/2013	Mac OS X 10.7	1.46%
5	09/2013	Mac OS X 10.6	1.48%
6	09/2013	Mac OS X 10.5	0.31%
7	09/2013	Mac OS X 10.4	0.08%
8	09/2013	Mac OS X (no version reported)	0.01%
9	10/2013	Mac OS X 10.1	0.00%
10	10/2013	Mac OS X 10.9	0.73%
11	10/2013	Mac OS X 10.8	2.88%
12	10/2013	Mac OS X 10.7	1.35%
13	10/2013	Mac OS X 10.6	1.40%
14	10/2013	Mac OS X 10.5	0.29%
15	10/2013	Mac OS X 10.4	0.07%
16	10/2013	Mac OS X (no version reported)	0.01%
17	11/2013	Mac OS X 10.1	0.00%
18	11/2013	Mac OS X 10.9	2.08%
19	11/2013	Mac OS X 10.8	1.59%

Each month is now represented by eight rows, one for every OS X version.

Step 3: Combining with Historical Data

We now have eleven months worth of market usage share data. But we want to see trends in OS X usage share over a longer period of time. This requires us to run our script again at the beginning of each month, to scrape the newly-posted data for the previous month. But now we need a way to combine the new month's data with the historical data we've already collected.

There are a few methods we could use to do this, each with their own benefits and drawbacks. The method I use is to filter out all but the new data, then concatenate that to the historical data. My definition of "new data" is data that I haven't already collected. I use this JSL to do the filtering.

```
dt historical << Select All Rows;
dt historical:Month << Set Selected( 1 );
dt historical << Select All Matching Cells;
dt historical << Invert Row Selection;
dt historical:Month << Set Selected( 0 );
dt stack << Delete Rows;
```

Let's break this down.

First, assume that *dt historical* is a reference to our data table of historical data. It has the same columns as the *dt stack* data table we just scraped from the web and combined, but it has more

rows that go back farther in time. The first thing we do is **<< Select All Rows** of *dt historical*. This is pretty self-explanatory.

Next, we select the *Month* column of *dt historical*, by sending **<< Set Selected(** 1 **)** to that specific column. This operation, combined with the previous one, has the effect of selecting all the *cells* of *Month*. This is different from only selecting the column or only selecting the rows. And it is a very important distinction, because the next operation works specifically on selected cells.

That next operation is **<< Select All Matching Cells**, and it is kind of magical! We send this message to *dt historical*, but the message doesn't really affect that data table. Instead, JMP looks at all other open data tables and selects any rows in them where any cell matches a cell that is selected in *dt historical*. So if *dt historical* has rows selected for *Month* = 09/2013 and *Month* = 10/2013, then **<< Select All Matching Cells** will cause the rows in *dt stacked* to also be selected, where *Month* has those same values. But the other rows of *dt stacked*, say where *Month* is 11/2013, will not be selected because *dt historical* doesn't have any row with that value. Magic!

The next two operations simply undo the selection in *dt historical*. **<< Invert Row Selection** is an easy way to say "select no rows". And **<< Set Selected(** 0 **)** just turns off the selected *Month* column.

At this point, *dt stacked* has all the rows selected that are already in *dt historical*. The next message **<< Delete Rows** deletes them. Now *dt stacked* contains only the new data.

The final step couldn't be easier.

```
dt historical << Concatenate( dt stack, Append to first table( 1 ));
Close( dt stack, No Save );
```

The Historical Data Table

We can do a few things to improve the reporting in our historical data table. First, the *Version* column needs a Value Label property. We label the raw OS X version numbers reported by NetMarketShare with their short code names. Some of these are the "big cat" names used in the first nine versions of OS X. The rest, starting with OS X 10.9, are "California" names. For the "(no version reported)" variant we use the label "Other". We also add a Value Ordering property to get these all in the correct order. Finally, we make the column a Label column.

We also want to balance the numbers so that we are focused on just the usage share of each OS X version within the realm of Macintosh users, rather than for the whole desktop PC market. To do this, we create a new column named *Usage Share*, which has this formula.

$$\frac{\textit{Market Usage Share}}{\text{Col Sum}\left[\textit{Market Usage Share}, \textit{Month}\right]}$$

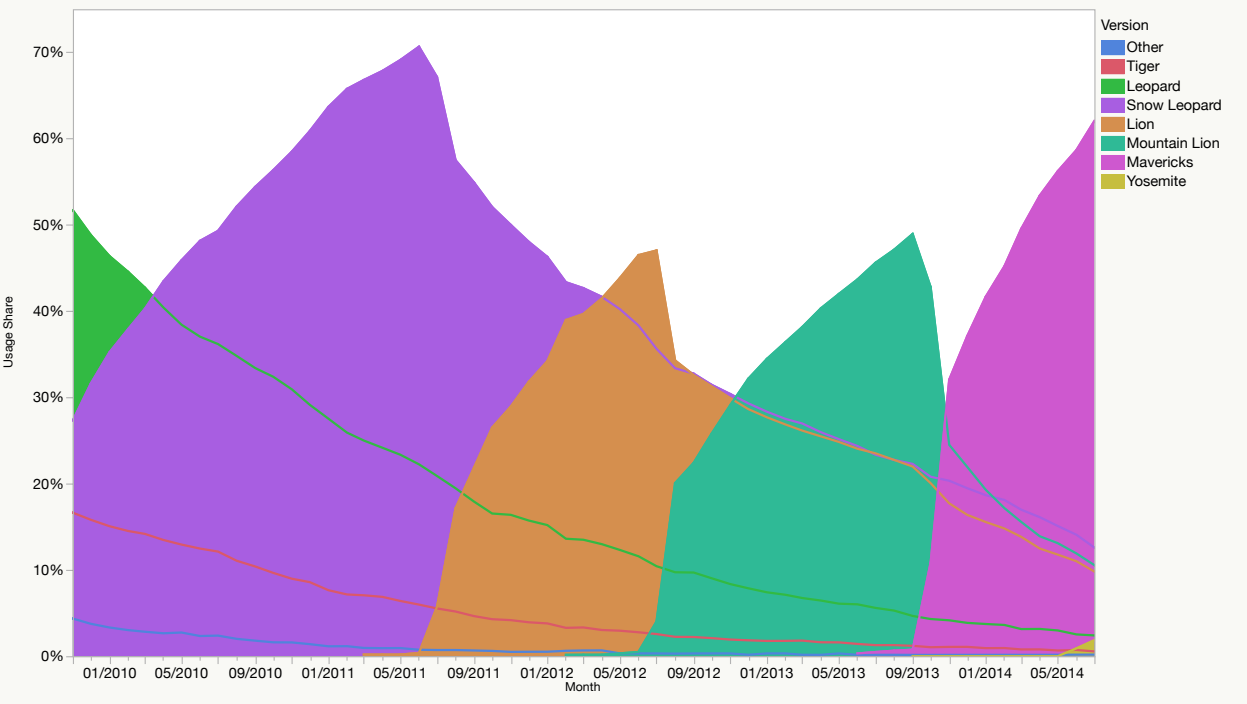
The Col Sum() function computes the sum of all the *Market Usage Share* values for a given *Month;* then the formula divides that into each individual *Market Usage Share*. For example, in September of 2013, Mountain Lion had 3.25% of the total desktop PC market usage share. But Mac users only accounted for 6.64% of all desktop PC usage that month. That means Mountain Lion had a usage share of 48.9% among Mac users (3.25/6.64 = 0.489).

Our collection script can now be added to our historical data table as a table script. When a table script is run, **Current Data Table()** is initially set to the data table that hosts the script. So we start our script with this.

```
dt historical = Current Data Table();
```

Graph and Analysis

I have collected data on OS X usage share going back to 2009. Let's graph that data and look at the trends. Here's a Graph Builder overlaid area plot.



The usage shares look like a mountain range, with each version peaking when the subsequent version is released. All the OS versions show a slow attrition rate, which probably reflects the replacement of old hardware with newer equipment that comes with the latest OS pre-installed.

The large purple mountain is Snow Leopard, and it had a very good run; especially when compared to its successor, Lion (the orange peak). In fact, Snow Leopard is still hanging on fairly well today, with more usage than both Lion and Lion's successor, Mountain Lion (teal). This is

most likely because Snow Leopard was the last OS version to support PowerPC applications. Snow Leopard proved so popular that in November of 2012, it caused Apple a real problem: their usage share was almost equally divided between three competing OS versions — Snow Leopard, Lion, and Mountain Lion.

Up until about December of 2013, it would not have been wise for us to drop support for Snow Leopard, because of its large continued usage. But then it dropped below 20% when Mavericks, which had been released in September of 2013, started to gain ground. Mavericks was the first OS X version that Apple distributed free of charge, which likely accounts for its quick uptake.

As of this writing, OS X 10.10 Yosemite is not yet released. Some usage statistics appear however, because Apple has conducted a public beta test prior to releasing it. It will be interesting to see if Yosemite finally puts Snow Leopard to rest.

Contact information

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

Michael Hecht
michael.hecht@jmp.com

References

[1] Wikipedia defines *web scraping* as "a computer software technique of extracting data from websites". This is done by "[simulating] human exploration of the World Wide Web by […] implementing low-level Hypertext Transfer Protocol (HTTP)". http://en.wikipedia.org/wiki/Web_scraping

[2] Wikipedia: OS X. http://en.wikipedia.org/wiki/Os_x

[3] JMP: System Requirements. http://www.jmp.com/support/system_requirements_jmp.shtml

[4] Wikipedia: Usage Share of Operating Systems. http://en.wikipedia.org/wiki/Usage_share_of_operating_systems

[5] NetMarketShare℠. http://www.netmarketshare.com

[6] JMP Online Documentation. Using JMP: Import Remote Files and Web Pages. http://www.jmp.com/support/help/Import_Remote_Files_and_Web_Pages.shtml#319726

[7] JMP Online Documentation. Discovering JMP: Joining Data Tables. http://www.jmp.com/support/help/Managing_Data.shtml#167540