Paper 7320-2016

Analytics of Things: Golf a Good Walk Spoiled?

David Shannon, Amadeus Software

ABSTRACT

This paper demonstrates techniques using SAS® software to combine data from devices and sensors for analytics. Base SAS, Data Integration Studio and Visual Analytics are used to query external services, import, fuzzy match, analyze and visualize data. Finally, the benefits of SAS® Event Stream Processing models and alerts will be discussed.

To bring the analytics of things to life the following data are collected: GPS data from countryside walks, GPS and score card data from smart phones whilst playing golf and meteorological data feeds. These data are combined to test the old adage that golf is a good walk spoiled. Further, the use of alerts and potential for predictive analytics is discussed.

INTRODUCTION

The Internet of Things (IoT) is an idiom meaning connected devices and applications, sharing data that are analysed in real time, informing decisions and actions.

This presents an opportunity for businesses to better understand the behaviour of products, services and customers. In turn businesses can create timely services, propositions and actions not previously possible.

The volumes of data generated may be small or large. Traditional technologies or Hadoop services all apply. Ultimately businesses are not simply adopting IoT because it's cool, rather to open consumer opportunity and drive automated decision making.

WHAT ARE THE INTERNET OF THINGS AND ANALYTICS OF THINGS?

The Internet of Things has three parts (reference 1):

1. The things with embedded sensors
2. The networks to integrate the things
3. The systems which process and analyse their data.

Data collected from Internet of Things are no more than that: Data. The complex, disparate data generated by devices and sensors are a typical example of Big Data.

What is the benefit of the Analytics of Things?

Competitive advantage leads to quicker return on investment, which in turn comes from exploiting data to the highest degree of intelligence (reference 2). It is the statistical analysis, forecasting, predictive modelling and ultimately optimization which makes the Internet of Things viable and progressive for our business services.

Machine learning is the automated optimisation of processes through pattern recognition and algorithm refinement. As data sources increase from the prevalence of interconnected devices and sensors, so the prominence of machine learning will increase.

Therefore, the Analytics of Things is: The optimization of products and services by analysing complex, disparate data sources generated by the Internet of Things, supplemented with existing data sources.

WHERE DO IOT DATA COME FROM?

The following groups are recognised to have adopted connected devices and sensors which are exploiting the internet of things:

- Manufacturing, Goods and Services
- Wearable technology
- Utilities
- Environment

The wider discussion of industry uses and benefit is left to for your further reading, but I recommend beginning with a SAS Voices blog published around the same time as this paper is being written (reference 3).

HOW DO WE USE SAS FOR ANALYTICS OF THINGS?

The following figure visualises the flow of data from Internet to Analytics of Things:

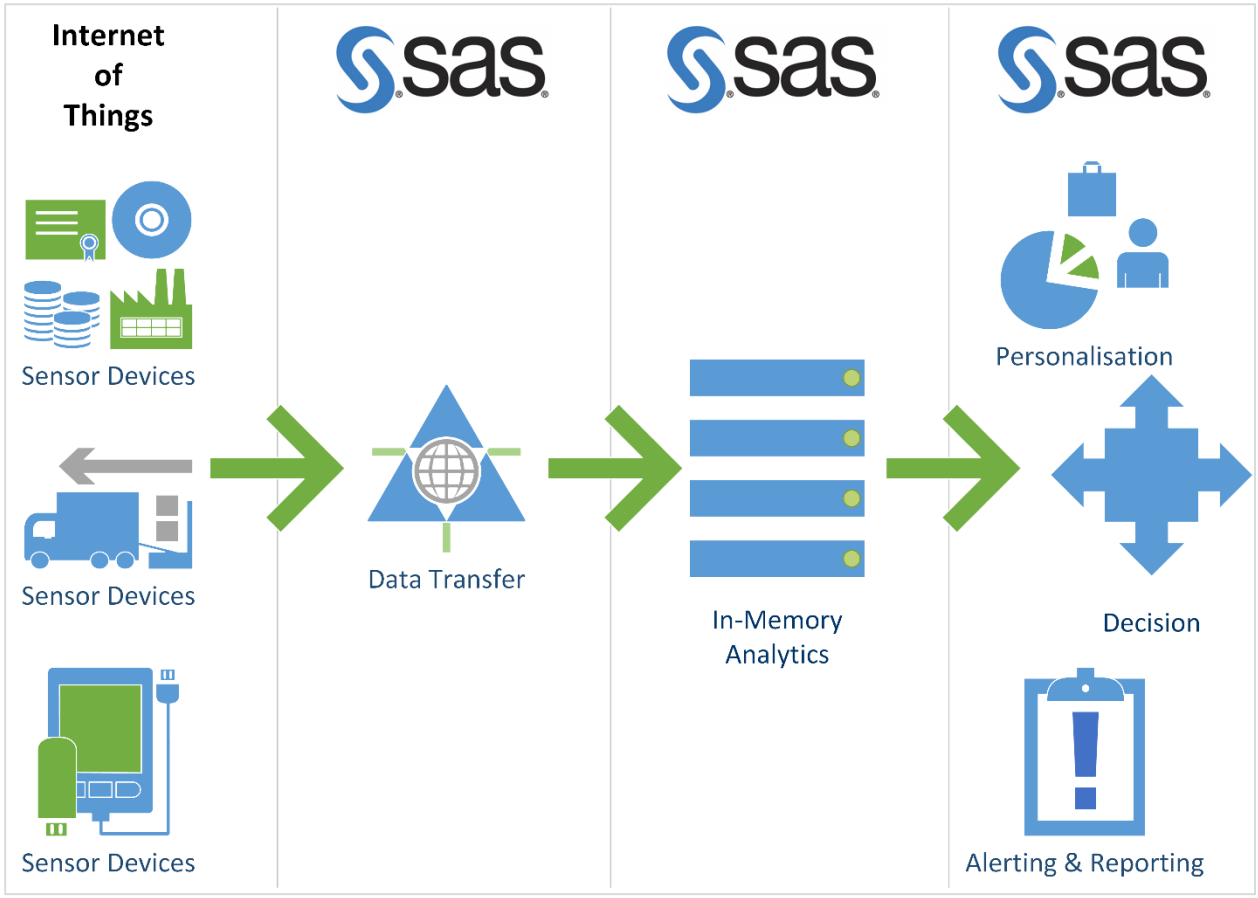


Figure 1: Data flow from Internet to Analytics of Things

The following three sets of tasks describe the Internet of Things to Analytics:

1. **IoT**: Sensors and devices generate data which are synchronised with secure repositories, usually in the cloud

2. **Data Transfer**: Data are queried by SAS for processing. Industry standard REST web services return data in JSON or XML format. These formats are readily imported by SAS. Increasingly, data are streamed via event stream processing, the so called *continuous query*.
3. **In-memory Analytics**: Manages the rapidly changing big data to deliver the Analytics of Things, informing and automating decision making.

GOLF A GOOD WALK SPOILED?

Let's bring things to life by statistically testing whether golf is a good walk spoiled. This seems to be an adage dating to from the early 1900's where novelist Harry Wilson uses the phrase in his books. Later Mark Twain is associated with the saying, although some say falsely. Either way, I am not a particularly keen golfer, or have persuasion towards the outcome of this hypothesis. It is simply an accessible way of demonstrating the Analytics of Things. Anyone with SAS can reproduce the techniques described in this paper, either to build a business case in your organization, or even just for fun.

METHODOLOGY

Let's assume a good walk is measured by the time required to walk between the tee and the hole (reference 4). The time taken between leaving and arriving at the hole will be the primary endpoint.

To minimize bias so far as reasonably possible, the following conditions were set when collecting data:

- A single player wore the same devices and repeatedly played 18-hole golf on the same course
- The player consistently played alone
- The same player walked directly around the same golf course, using the same sensor devices without playing the game, representing the countryside walk.

Sensors are worn by the player collected GPS, number of shots and vital signs at one second intervals. Weather details from the nearest weather station will be captured at the time of the golf events, or as near as possible.

Average times from the repeated rounds of golf are calculated between tee and hole. Whilst more sophisticated statistical methods may be appropriate, in this paper a paired t-test is used to compare the group means times from tee to hole.

SENSOR DEVICES

Data are collected from the following sensors and services:

- **The Met Office:** An open license is used from the United Kingdom's national weather service, where by the nearest weather station is identified and a query performed to extract weather observations corresponding to the observed times associated with playing golf. Instructions are found in Appendix A for obtaining an API key and deriving the necessary URL's for querying these data.
- **Microsoft Band and Sports Tracker:** The latest generation of this wearable device allows tracking of your golf game. Specifically, data extracted from the Microsoft Health Dashboard allow the GPS coordinates, shots and other vital signs data.

DATA TRANSFER

Familiarity with the following techniques are necessary when querying data from web services:

- Querying data from RESTful HTTP web services via Proc HTTP.
- Querying data from RESTful HTTP web services via Proc DS2.
- Importing JSON
- Importing XML

- Web Service authentication

Choosing between Proc HTTP or Proc DS2 is a matter of personal choice. The former presents easier syntax whilst the latter offers greater control, particularly when parsing JSON responses.

In the following section HTTP addresses are presented that fetch responses from web services. Understanding the API of those web services requires studying the API documentation from the web service provider.

Querying REST web services via a Data Integration Studio REST Transform or Proc HTTP

In SAS Data Integration Studio, add a REST transform from the Access category onto a job. Figure 2 shows the endpoint completed (private key truncated).

A complete explanation of this URL is found in Appendix A with the Proc HTTP step for those using alternative SAS programming environments.

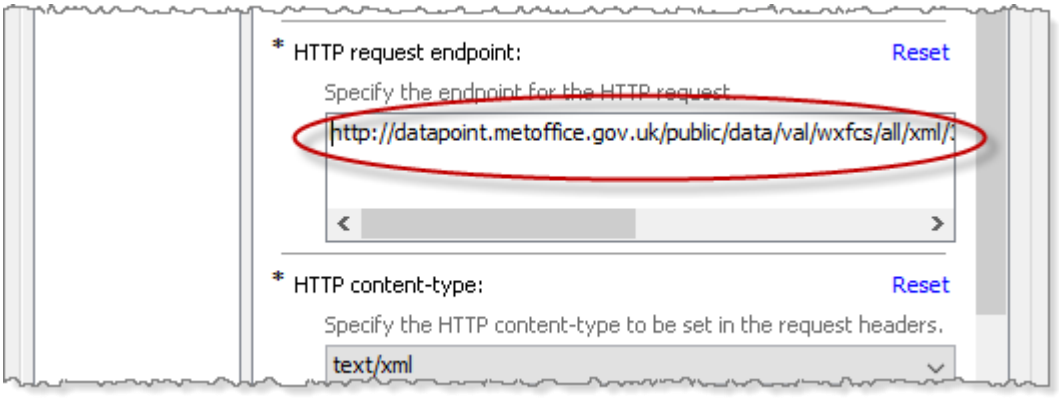


Figure 2: REST Transform with endpoint specified

Parameters passed into the web service via its API have requested results be returned in XML format. The REST transform is configured to handle the file. On the Input and Output section, Figure 3, set the output type to "File". Rather than an explicit Output filename, a fileref "_fore_" is typed into the field:

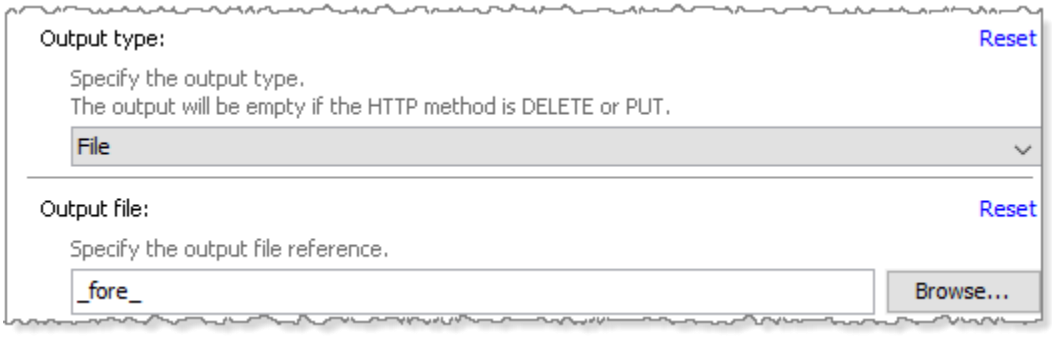


Figure 3: REST Transform Input and Output

The _fore_ fileref must be assigned prior to the transform code executing. The precode section of the transform is used to assign the fileref (Figure 4, below). The temp engine causes a random filename to be generated in the SAS WORK library.

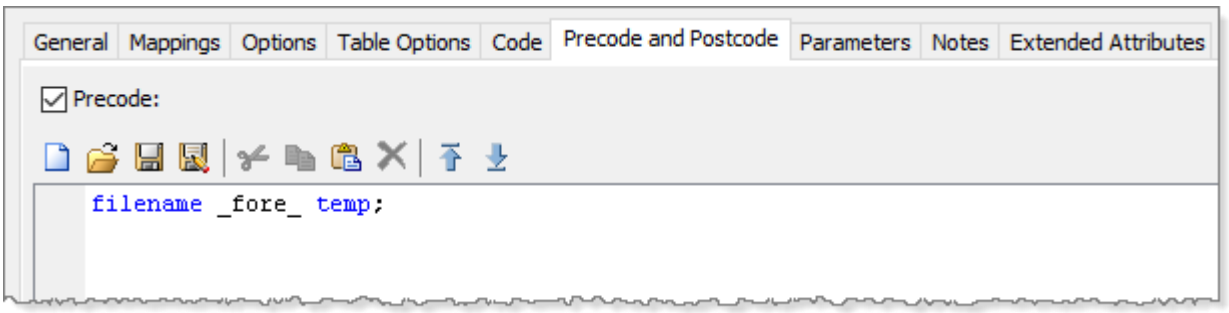


Figure 4: REST Transform Precode

Importing the XML file into SAS is simplified with code generated by The SAS XML Mapper. If in any doubt a video can be found on YouTube (reference 5) demonstrating this technique. The input XML file is then referenced using the _fore_ fileref, rather than an explicit path and filename.

Querying data from REST web services via Proc DS2

Alternatively, Proc DS2 is an extremely capable propriety programming language in Base SAS. Amongst its strengths are packages which interact with web services and parse JSON.

Code is written or inserted into a user transform for use in a SAS Data Integration Studio job, Figure 5.

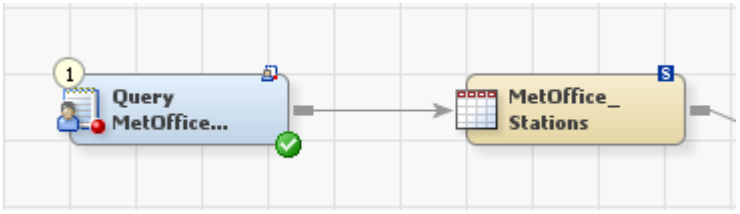


Figure 5: User Transform for Proc DS2

DS2 code is organized into methods, which contain objects that your code interacts with. Those who have previously written SAS SCL will be familiar with the default INIT, RUN and TERM code blocks, ability to write your own methods and the concept of instantiating objects before calling their methods and properties.

The following is an extract of the DS2 statements found within the INIT block of the Proc DS2 used to query a list of weather stations and their locations. In this example I requested the results are returned in JSON. The full program with comments is found within Appendix B.

```
      * Declare HTTP package with reference w…;
      dcl package http w();

      * Delcare fields…;
      dcl int i rc tokenType parseFlags;
      dcl nvarchar(128) token;

      * Initialise the HTTP GET method with the URL to query met office (note CAT
        function allows the string to be wrapped making it more readable);
     w.createGetMethod(
cat('http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/sitelist?',
    '&key=INSERT-YOUR-API-KEY')
    );

      * Execute the HTTP GET method;
      w.executeMethod();

      * Store the response and return code;
```

```
w.getResponseBodyAsString(response, rc);

* Pass the response field into the JSON parser (j is instantiated elsewhere);
rc = j.createParser( response );

* Loop over each token in the JSON until the Location tag is found.  Then
   call a user written method called parsesites to parse the JSON;
do while (rc = 0);
     j.getNextToken( rc, token, tokenType, parseFlags);
     * When the keyword in the JSON is 'Location' call the parseSites method;
     if (token='Location') then parseSites();
end;
```

Web Service Authentication

Many open web services require little more than registration to gain access to their functionality. Identification is managed through API keys which remain confidential to you. Identification is typically used to limit the number of requests your solution makes in a given period of time.

Secure services, such as Google, Microsoft, social media websites, use the OAuth 2 protocol. OAuth requires a bearer token to be requested, captured and then passed back in the header of your query. Obtaining the bearer token may be difficult, some suggest impossible, without interactive user login.

Dealing with those web services using OAuth 2 required, by far, more time than all the other examples put together for this paper. Interacting with the Microsoft API is worthy of a paper of its own.

COMBINING DATA

Having followed methods described above, data from my sensor devices and the met office weather observations are now stored in SAS data sets.

Data from the GPS tracking are collected to a precision of one observation per second, as illustrated in Figure 6:

trkpt_lat	trkpt_lon	ele	time	e...	Position	hole
51.608376667	-1.21582	59.6	2016-03-13T14:17:53	Golf		
51.608388333	-1.215853333	59.6	2016-03-13T14:17:54	Golf		
51.6084	-1.215883333	59.6	2016-03-13T14:17:55	Golf		
51.608408333	-1.215915	59.5	2016-03-13T14:17:56	Golf	Tee	1
51.60842	-1.21594	59.5	2016-03-13T14:17:57	Golf		1
51.60843	-1.21596	59.6	2016-03-13T14:17:58	Golf		1
51.60844	-1.215973333	59.7	2016-03-13T14:17:59	Golf		1

Figure 6: Partial Data Set of Microsoft Band and Sports Tracker Events

However, other data sources may not be collected to the same level of detail. Specifically, weather observations are logged at three hourly intervals. Rules may be considered for joining these imprecise records. Techniques for fuzzy merging, both numeric and character data are well documented in papers and blogs. See references 6 for a discussion relating to fuzzy matching of numeric values.

For simplicity, I deemed a weather observation to be relevant for the full three hours, hence derived the observations expiry date with a data step. The resulting structure is seen in Figure 7:

Observation_Start	Observation_Expiry	Screen Tem...	Wind Gust	Wind Directi...	Wind Speed
13MAR16:00:00:00	13MAR2016:02:59:59	2	15	N	1
13MAR16:03:00:00	13MAR2016:05:59:59	0	4	SSW	1
13MAR16:06:00:00	13MAR2016:08:59:59	0	8	NE	5
13MAR16:09:00:00	13MAR2016:11:59:59	4	8	ENE	4

Figure 7: Partial Weather Observations data set

Back in Data Integration Studio, it makes joining these data straight forward. A SQL Join transform is added to a job diagram Figure 8:

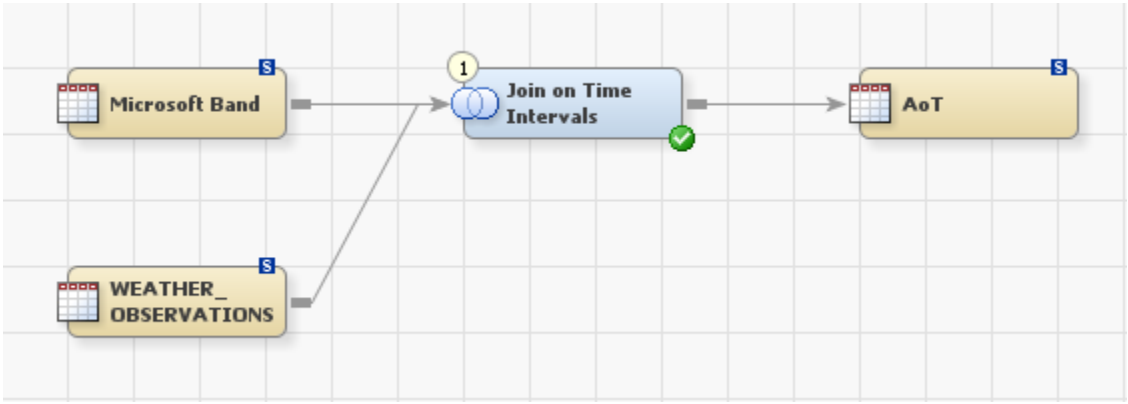


Figure 8: Joining tables containing unequal time intervals using SQL Join in SAS Data Integration Studio

The Join criteria are defined so that the GPS Time column must be between the weather Observation Start and End dates to fulfill the join. This is illustrated by Figure 9:

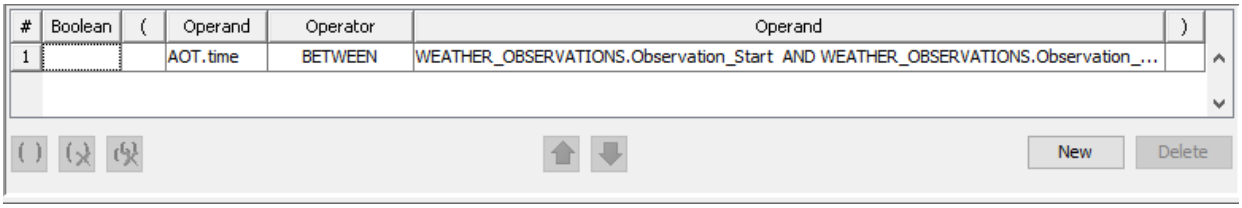


Figure 9: Logic for joining data with unequal grain.

This is trivial logic for anyone already familiar with SQL, the following syntax being generated by the transform in Data Integration Studio:

```
proc sql;
   create table aotdtl.AOT_Weather as
   select *
   from
      aotdtl.AOT as AOT left join
      aotdtl.WEATHER_OBSERVATIONS as WEATHER_OBSERVATIONS
         on
          (AOT.time BETWEEN WEATHER_OBSERVATIONS.Observation_Start
        AND WEATHER_OBSERVATIONS.Observation_Expiry
          )
   ;
quit;
```

IN-MEMORY ANALYTICS

The SAS LASR Analytical Server (LASR) is used to hold data in physical memory (RAM) across one or more servers.

In addition to creating Explorations and Reports in SAS Visual Analytics, programmers are encouraged to study Proc IMSTAT. Amongst much other functionality, this procedure allows statistical methods to be executed directly against in-memory LASR tables.

Loading data into the LASR server may be achieved in a number of ways. Connecting to the LASR server is achieved with a SAS library to where data sets are then written. SAS Enterprise Guide and Data Integration Studio have tasks and transforms to this end. Personally, I find importing data via the Visual Analytics Administrator is more complete. It automates the data copy, metadata generation and creates a SAS Job which may then be maintained within Data Integration Studio. See Figure 10, below:

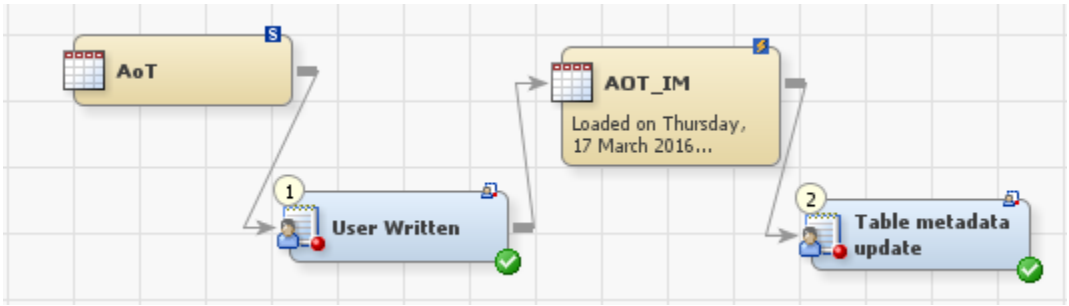


Figure 10: Job generated by SAS Visual Analytics to load SAS data into SAS LASR Analytical Server

With data loaded into the LASR server, we now begin visualizing observations through SAS Visual Analytics. Figure 11 shows two Geo Maps. The left map is a Walking event and the map on the right is a single golfing event.

My golfing is clearly more erratic than walking. Nonetheless; is the difference statistically significant, i.e. is erraticism spoiling my walk? A stored process was created to perform the paired t-test.

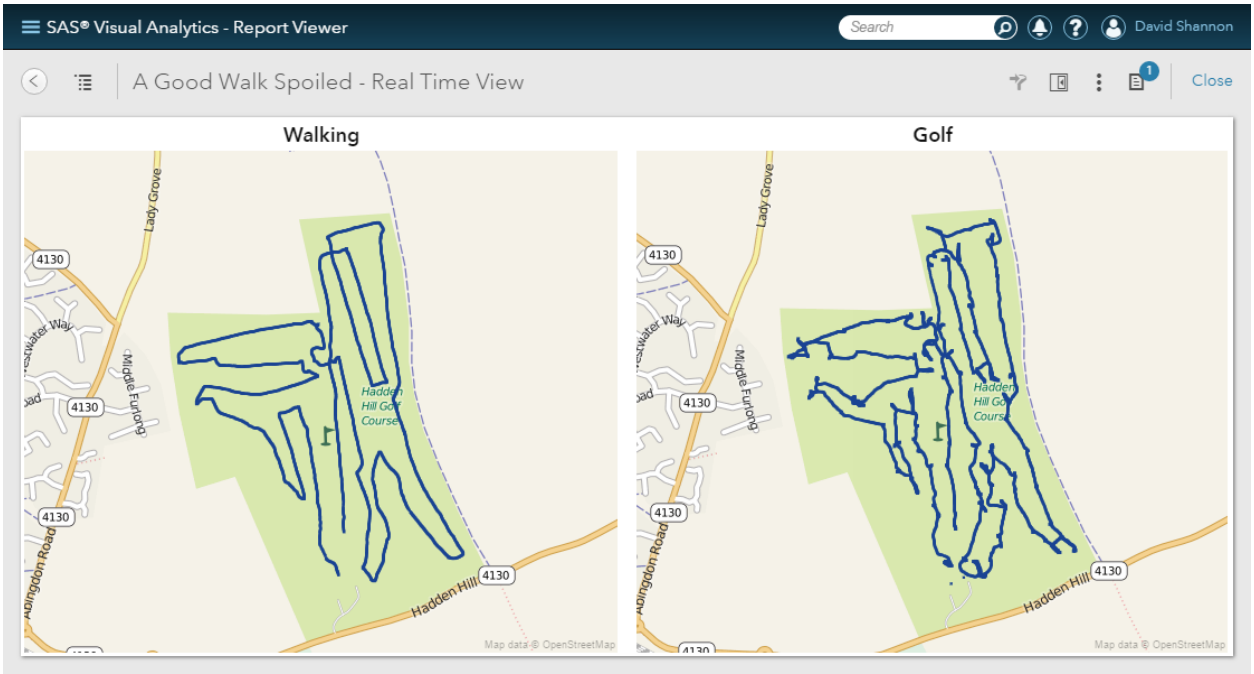


Figure 11: [Near] real time view of GPS data

The following chart (Figure 12) presents the averaged times between tee and hole. The blue bars (golf) are clearly taller than the green bars (walking), indicating longer durations. On the right, output is shown from the stored process used to execute a Students paired t-test.

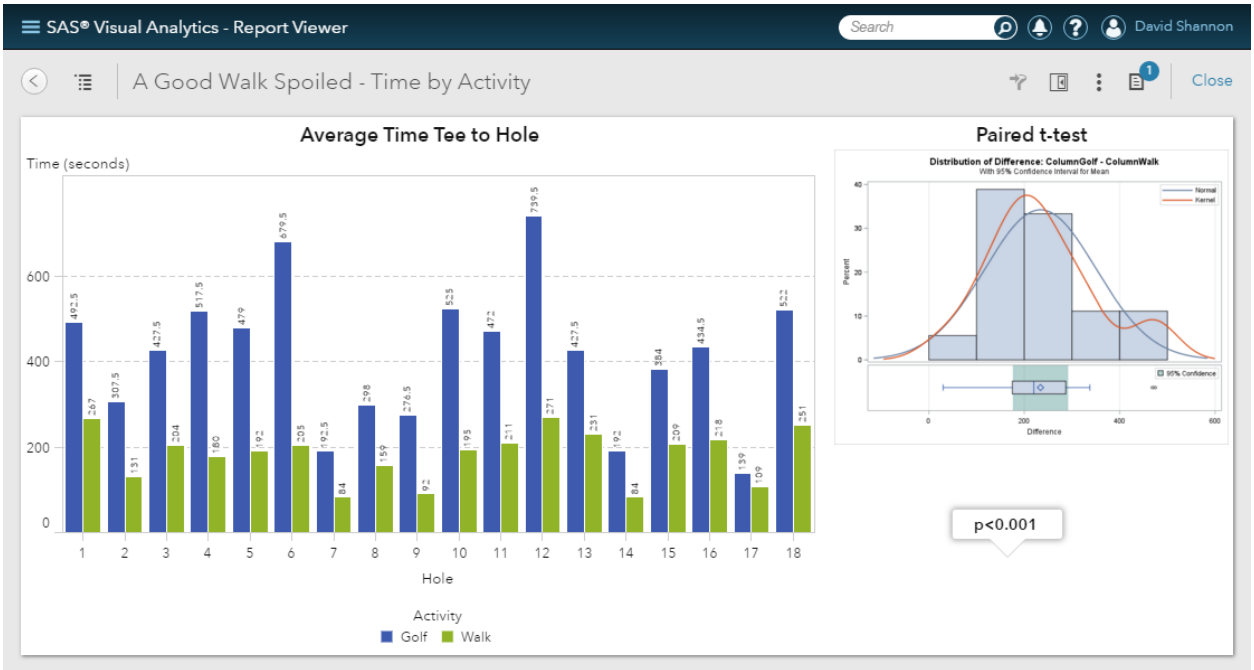


Figure 12: Average (mean) golf and walking times

The histogram and box-and-whisker plot are diagnostics from Student's paired t-test. The confidence interval (green) does not straddle zero. Hence, there is strong statistical evidence of a difference between the mean golf and walking times from tee to hole.

In my analyses there is no account taken of my golf score, the weather observations or vital signs collected from the devices I wore whilst playing. Consider the possibility of being able to warn me of impending weather that could affect my game, or an intervention as my heart rate rises to a level causing inconsistency in my shots. Consider how machine learning enables models to understand the factors affecting my golf score and provide automated interventions, even whilst on the course. Ultimately these interventions or alerts may improve my golf score, satisfaction from the sport and increase the likelihood of me booking another next round of golf, or perhaps a tuition session may be more appropriate!

SAS® EVENT STREAM PROCESSING

The greatest deficiency with the Data Integration jobs I created whilst constructing this paper, is that data are refreshed in batch. My devices must be synchronised with their companion apps on my phone, which in turn upload data to the internet, from where I can access those data for analysis.

To process and analyse real time events, with potentially vast volumes of data, an appropriate technology is required.

Effectively a continuous query, SAS Event Stream Processing is a solution which streams data into SAS in real time. Projects define how events are processed and the actions may be based on native text parsing, data quality and various statistical methods. Users may define rules for decisions and alerts taken on individual and cumulative data as they are streamed into SAS.

We expect to see this solution become integral to many solutions in the future.

CONCLUSION

The paper has demonstrated data can be collected in near real time from devices with sensors, joined with further data enriching the data set for analysis. Actions and interventions are possible much closer in time to the event, presenting opportunities for goods and services.

Event streaming technology overcomes the latency between sensors generating data and its analysis. This is still dependent on the capability of sensors, any companion devices, being able to stream data in real or near time.

Decision making is likely to gain autonomy through machine learning. Actions will therefore be taken more quickly and accurately to a subject level. This will provide efficiencies to many organizations and therefore investment in IoT will continue to be observed.

Finally, golf *is* a good walk spoiled. That is true assuming you play golf at my pace and agree a spoiled walk to be a slower walk than otherwise possible. Perhaps that is a subjective conclusion rather than scientific, however techniques for integrated IoT data with analytics have been demonstrated.

REFERENCES

1. SAS Institute Inc. 2016. "What is the Internet of Things?" Accessed 9th March, 2016. www.sas.com/iot.
2. Anderson J, Rainie L. "The Internet of Things Will Thrive by 2025", Pew Research Centre. May 2014. Available at http://www.pewinternet.org/files/2014/05/PIP_Internet-of-things_0514142.pdf .
3. SAS Voices, February 2016, "Ten IoT Articles to catch you up on the latest trend", Accessed 16th March 2016.
4. Golf Digest 2012, "A Good Walk Measured", Accessed 13th December 13, 2015 http://www.golfdigest.com/story/david-owen-my-tech-2012-10
5. SAS Institute, 2011, How to Generate an XMLMap File Using the SAS XML Mapper https://www.youtube.com/watch?v=Kva6augvJOM, Accessed 16th March 2016.
6. Foley, MJ, 1999, Fuzzy Merges: Examples and Techniques, SUGI24, p46-24.

CONTACT INFORMATION

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

David Shannon
Amadeus Software
+44 (0)1993 848010
david.shannon@amadeus.co.uk
www.amadeus.co.uk

APPENDIX A: QUERYING THE UK MET OFFICE DATAPOINT SERVICE STEP-BY-STEP

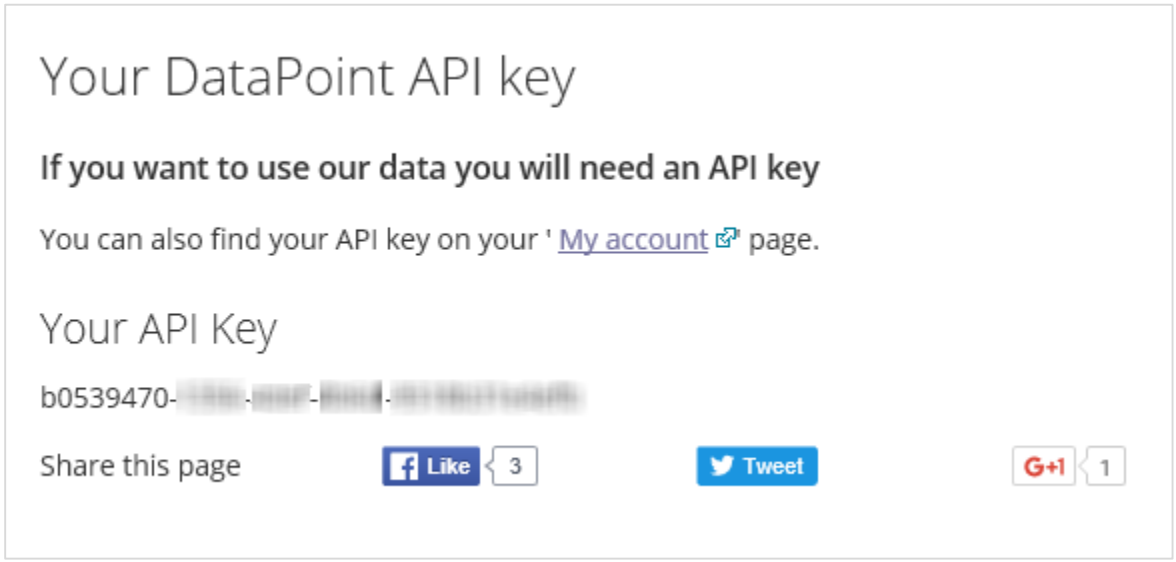
Met Office provides various weather feeds of varying granularity and detail, for open and commercial use. Freely available access is provided via a project called DataPoint which is intended for innovative applications, professionals, research, etc.

At the time of writing, the website describing DataPoint's API is accessed from the following URL:

`http://www.metoffice.gov.uk/datapoint`

You will need an API Key to use this service. The API Key forms part of the URL used to request information from the Met Office. It uniquely identifies you (or your application) to the Met Office each time a query is executed; therefore, the key should be considered sensitive.

After registering for a DataPoint account you will be able to access you API key, appearing similarly to the following, the final four parts of my API Key are disguised:



The base URL for querying DataPoint is:

`http://datapoint.metoffice.gov.uk/public/data/`***`resource`***`?key=`***`APIkey`***

Where:

resource is any of the services provided by DataPoint. In this paper two resources are used:

`val/wxfcs/all/`***`datatype`***`/sitelist`	Returns a list of all the UK location ID's providing daily and three hourly forecasts.
`val/wxfcs/all/`***`datatype`***`/`*`locationId`*	Substitute the locationID keyword with the ID of the desired forecast location and this resource will return three hourly forecasts for the next three days.

datatype describes the output format desired, from either of: **JSON** or **XML**

APIKey is your API Key.

For example, the following URL will generate a list of UK weather station ID's in JSON format:

```
http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/sitelist?key=<insert-your-key-here>
```

Turning this into a SAS program, requires Proc HTTP. Again, note that I have truncated the screenshot to disguise my API Key:

```
filename _sites_ temp;
proc http
   method="GET"
   url="http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/sitelist?key=b0539470-
   out=_sites_;
run;
quit;

data _null_;
  infile _sites_;
  input;
  putlog _infile_;
run;
```

This validates the query as accurate by printing the raw JSON into the SAS log, partial output is shown below:

```
6          out=_sites_;
7       run;

NOTE: PROCEDURE HTTP used (Total process time):
      real time           0.75 seconds
      cpu time            0.09 seconds

NOTE: 200 OK

8       quit;
9
10      data _null_;
11        infile _sites_;
12        input;
13        putlog _infile_;
14      run;

NOTE: The infile _SITES_ is:
      Filename=C:\SAS94\SASWORK\David.Shannon\_TD2504_AC101_\#LN00010,
      RECFM=V,LRECL=32767,File Size (bytes)=916163,
      Last Modified=10 March 2016 11:20:48 o'clock,
      Create Time=10 March 2016 11:20:48 o'clock

{"Locations":{"Location":[{"elevation":"933.0","id":"3072","latitude":"56
2","name":"Cairnwell","nationalPark":"Cairngorms National Park","region":
"Perth and Kinross"},{"elevation":"134.0","id":"3088","latitude":"56.852"
```

APPENDIX B: QUERYING A LIST OF MET OFFICE WEATHER STATIONS WITH PROC DS2

Appendix A proves the query from SAS to the Met Office DataPoint service works. However, the JSON needs to be parsed, storing the data in a data set. There are various blogs and articles for parsing JSON, however Proc DS2 contains two packages which allow the HTTP request and JSON parsing to be completed in a single step.

The following program illustrates performing a HTTP GET request, with JSON response and parsing the JSON into a SAS data set named SITELIST.

The parseSites method is constructed to parse the JSON file returned by this Met Office DataPoint web service. Creating the correct steps and attributes required examination of the JSON output, along with an understanding of the API documentation from the service being queried:

```
proc ds2;
  * Create a new data set - allowing it to be overwritten;
  data work.sitelist(overwrite=yes);

      * Declare global JSON package with reference j;
      dcl package json j();

      * Declare global output columns using native SAS types;
      dcl char(128) name nationalpark region unitaryautharea;
      dcl double id elevation latitude longitude;

      * Declare global a field to store the query results from the Met Office
         DataPoint feed;
      dcl varchar(9999999) character set utf8 response;

      * Declare global return code and counter fields;
      dcl int rc;

      * Drop unrequired columns from the output data set;
      drop response rc;

      * Define a method which parses the JSON into SAS fields;
      method parseSites();

        dcl int tokenType parseFlags;
        dcl nvarchar(128) token;
        rc=0;

        * iterate over all message entries;
        do while (rc=0);
            j.getNextToken( rc, token, tokenType, parseFlags);
            if (token = 'elevation') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              elevation=inputn(token,'8.1');
              nationalPark='';
            end;
            if (token = 'id') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              id=inputn(token,'8.0');
            end;
            if (token = 'latitude') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              latitude=inputn(token,'8.3');
```

```
            end;
            if (token = 'longitude') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              longitude=inputn(token,'8.3');
            end;
            if (token = 'name') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              name=token;
            end;
            if (token = 'nationalPark') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              nationalPark=token;
            end;
            if (token = 'region') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              region=token;
            end;
            if (token = 'unitaryAuthArea') then do;
              j.getNextToken( rc, token, tokenType, parseFlags);
              unitaryAuthArea=token;
              output;
            end;
          end;
          return;
        end;

        * The INIT method is always called;
        method init();

          * Declare HTTP package with reference w that has scope of INIT code
              block;
          dcl package http w();

          * Delcare fields with scope of INIT code block;
          dcl int i rc tokenType parseFlags;
          dcl nvarchar(128) token;

          * Initialise the HTTP GET method with the URL to query met office
              (note CAT function allows the string to be wrapped making it more
                 readable);
          w.createGetMethod(

cat('http://datapoint.metoffice.gov.uk/public/data/val/wxfcs/all/json/sitelis
t?',
                    '&key=INSERT-YOUR-API-KEY'));

          * Execute the HTTP GET method;
          w.executeMethod();

          * Store the response and return code. Response;
          w.getResponseBodyAsString(response, rc);

          * Pass the response field into the JSON parser;
          rc = j.createParser( response );

          * Loop over each token in the JSON until the Location tag is found.  Then
            call a user written method called parsesites to parse the JSON.
```

```
        do while (rc = 0);
            j.getNextToken( rc, token, tokenType, parseFlags);
            * When the keyword in the JSON is 'Location' call the parseSites
              method;
            if (token='Location') then parseSites();
        end;
      end;

      * Term always runs at the end of the step;
      method term();
        * Release resources;
        rc = j.destroyParser();
      end;

  enddata;

run;
quit;
```