Google Analytics website users data to Relational database service

The application is for pulling google analytics metrics, dimension and segments then perform data aggregation and send the same to a database.

Resources

Libraries.py

- Loads necessary third party libraries and user defined functions. Changing *date_range()* within this file as *date_range('weekly')* will produce weekly data points. It is daily by default.

Credentials.py

- Setting up google analytics reporting API authentication is done within this file. If success it will display "ACCESS ACQUIRED"

GAminer.py

- Imports both libraries and credentials files
- This is where the heavy lifting occurs, that is all data points needed from GA are configured and request sent using *get_report* function.
- If you need to add any data points this is where you will create a new *get_report ()* and configure it for either metrics, dimensions or segments.
- The functions will return json objects which will be passed for further processing downstream by data_aggregator function.
- "ALL REQUIRED DATA PULLED SUCCESSFULLY" is displayed

Data_aggregator.py

- This receives the various json data outputs from GAminer and uses various custom functions to generate single data points or data frames.

- The functions used here are both defined either here or within the libraries.py file. It will generate data frames ready for database insertion

- Whenever you add more data points on GAminer ensure to call their json outputs here as well.
- N/B: *If a new destination is added you have to add it to the destinations list here for its data points to be pulled.*

Schema.py

- Run this file in your chosen database to generate empty tables that will hold the data sets being pulled by the script daily or weekly.

★ *N/B : Ensure to create the empty tables before running the main.py file.*

Main.py

- This section will need database access credentials set-up done on schema.py. Adjust accordingly
- It will check if a table is empty then add data to it. If it has data it will append and adjust the id which is the primary key. The date variable will help to distinguish the data points
- “DATABASE UPDATE SUCCESS” will be displayed to mark the end.

Installation

Procedure

1. Create the tables on schema.sql on your server
2. Open main.py and change db host, username and password to match your server
3. Check libraries.py to confirm that you have installed all the libraries
4. Run python main.py
5. Finally check if you have 3 tables((primary_metrics, destination_users, virtual) with various data about website KPIs

TroubleShooting

Check for exceptions using the order below:

- Libraries -> credentials -> GAminer -> data_aggregator ->main
- Ensure that db is created, tables can either be empty or with data
- Ensure db access credentials are passed to main.py

Data Output

- The data output is daily by default meaning if you run it today it will pull data for yesterday. The data accuracy may vary a little depending on the time zone of the server but generally within 95% to 99% accuracy.
- The service can also be made to run weekly by modifying the date_range(duration=”weekly”) within GAminer.py
- Date from and date to will help in telling the difference between weekly and daily data since both will go on the same tables.
- The data are stored using the long format which can be queried to drill down and works best with visualization tools.

Example

If you want to use the script in the future to pull more data, all you need to know is GA metric, dimension and segment definitions

For example "sessions" is a GA metric, "organic search" is a GA dimension. Then apply the function on GAminer.py to pull the GA metric or dimensions by replacing the KPIs accordingly as outlined below.

```
def get_report(analytics,start,end):
 return analytics.reports().batchGet(
    body={
      'reportRequests': [
      {
        'viewId': VIEW_ID,
        'pageSize': 10000,
        'dateRanges': [{'startDate': start, 'endDate': end}],
        'metrics': [{'expression': 'ga:sessions'}],
        'dimensions': [{'name': 'ga:channelGrouping'}],
         "dimensionFilterClauses":[{
              "filters":[{
              "dimensionName":"ga:channelGrouping",
              "operator":"EXACT",
              "expressions":[
              "Organic Search"
              ]
           }]}]        }]
    }
 ).execute()
```

References
Use the links below to try different scenarios

Google Analytics metrics, dimensions and segments Overview:
https://ga-dev-tools.appspot.com/query-explorer/

Google Analytics Reporting API docs :
https://developers.google.com/analytics/devguides/reporting/core/v4/basics

Compiled by:
Mikes Kocholla