**Structure of Json used to transfer data b/w GCP & AppsScript**

Documentation of the structure of the json that is used to describe the user’s intent & the structure of json used to display the result table & suggestions.

**Table of contents**

[JSON sent from AppsScript to Intents endpoint GCP](#_4adopjbt32n9)

[JSON sent from Intents endpoint GCP to AppsScript](#_x6imixa65ine)

[JSON sent from AppsScript to Date Detection endpoint GCP](#_4jjkcdn7tmur)

[JSON sent from Date Detection endpoint GCP to AppsScript](#_71cbwih3tu7w)

[Appendix](#_qdaxl7f572i4)

There are 2 GCP endpoints. Their functions are to -

1. Answers to the intent of the user & returns debiasing suggestions [Intents Endpoint]
2. Detect date columns in the user’s table [Date detection Endpoint]

**The json is sent using a http request.**

# JSON sent from AppsScript to Intents endpoint GCP

Json object that contains all aspects and table's data

Json (dictionary in python / obj in javascript)

{

‘intent’: string,

‘table’: 2d array,

‘metric’: string,

‘dimensions’: 1D array

‘summaryOperator’: string,

‘isAsc’: boolean,

‘topKLimit’: integer,

‘slices’: 1D array of objects,

‘dateRange’: object,

‘timeGranularity’: string,

‘comparisonValue’: object,

‘compareDateRange’: object,

‘correlationMetric’: object,

‘dateColumns’: object

}

**INFORMATION ABOUT EACH KEY**

* json[ “intent” ] contains a string - the intent in the query computed
  + show
  + topk
  + slice\_compare
  + time\_compare
  + trend
  + correlation
* json[ “table” ] is a 2d array containing the data of range selected by the user
* json[ “metric” ] is a string storing column name
* json[ “dimension” ] is a 1d array of strings storing name of columns
* json[ “summary” ] is a string storing the name of operation to be applied

Summary operations present :

* + None
  + Sum
  + Proportion of sum
  + Mean
  + Median
  + Count
  + Proportion of count
  + Count Distinct
  + Standard Deviation
  + Variance
  + First
  + Last
  + Minimum
  + Maximum
* json[ “isAsc” ] is a boolean variable which stores true for ascending order. It is set to false by default.
* json[ “topKLimit” ] is an Integer storing K . It has a default value of 10

There is a test that ensures k is not greater than the total number of rows in the table. (backend)

* json[ “slive” ] is a 1d array of objects

Each object has 3 keys -

* + sliceCol - string containing column name
  + sliceOp - string containing filter operation name
    - In,not in - array of string/number/boolean
    - Equal, not equal - string//number/boolean
    - <, <= , >, >= check number
  + sliceVal - 1d array for in/not in, string/number/boolean for =/!=, number for </<=/>/>=

Filter Operations Present :

* + Equal to
  + Not equal to
  + Less than
  + Less than equal to
  + Greater than
  + Greater than equal to
  + In
  + Not In
* json[ “dateRange” ] is an object with 3 keys
  + dateCol - string containing name of column which has dates
  + dateStart - date (yyyy-mm-dd)
  + dateEnd - date (yyyy-mm-dd)
* json[ “timeGranularity” ] is a string containing the selected value
  + Daily
  + Fortnightly (not supported yet)
  + Monthly
  + Quarterly (not supported yet)
  + Semi-Annually (not supported yet)
  + Annually
* json[ “comparisonValue” ] is an object with 3 keys
  + comparisonColumn - string containing name of column which contains the 2 specified values
  + slice1 - first value to compare
  + slice2 - second value to compare
* json[ “compareDateRange” ] is an object with 5 keys
  + dateCol - string containing name of column which contains the 2 specified date ranges
  + dateStart1 - start date of first range
  + dateEnd1 - end date of first range
  + dateStart2 - start date of second range
  + dateEnd2 - end date of second range
* json[ “correlationMetrics” ] is an object with 2 keys
  + metric1 - first metric used to apply correlation
  + metric2 - second metric used to apply correlation
* json[ “dateColumns” ] is an object with the same structure as the one returned from the detect date function on gcp.

# JSON sent from Intents endpoint GCP to AppsScript

Json object that results & suggestions of the query

Json (dictionary in python / obj in javascript)

{

‘outputTable’ : 2d array

‘suggestions’: **list of suggestions**

‘isSlicingPassed’ : 1d array of True/False values

‘isInTopK’ : 1d array of True/False values

}

**List of suggestions (array of objects)**

[

{

‘suggestion’ : ‘<Suggestion to be displayed>’

oversight\_name : ‘<As discussed it will help in getting feedback later>’

confidence\_score: ‘<As discussed it will help in getting feedback later>’

‘json’ : {<Empty if no new query suggested>}

‘is\_row\_level\_suggestion’ : <True/False>

row\_list :

* If is\_row\_level\_suggestion = True :
  + [List of dictionary {row:row\_name, confidence\_score: parameter} ]
  + Row number is returned ( header row is considered as 0)
* Else :
  + None

‘is\_column\_level\_suggestion’ : <True/False>

col\_list :

* If is\_column\_level\_suggestion = True :
  + [List of dictionary {column:column\_name, confidence\_score: parameter} ]
* Else :
  + None

},

{

# Other such suggestions

}

]

**NOTE:**

* Json key is set only if it provides json
* In case there are no suggestions, return an empty array

# 

# JSON sent from AppsScript to Date Detection endpoint GCP

JSON

{

table : 2d array containing the data of selected table

}

# JSON sent from Date Detection endpoint GCP to AppsScript

Json

{

‘date\_column\_name’:

{

‘type’: INCONSISTENT/CONSISTENT/ALL\_AMBIGUOUS

‘day\_first’: true/false/None

‘min\_date’:

{

‘day\_first\_true’:

‘day\_first\_false’: # one of these will be not present if column is consistent

}

‘max\_date’ :

{

‘day\_first\_true’:

‘day\_first\_false’:

}

}

}

**NOTE:**

* min\_date and max\_date contain the minimum and maximum dates present in the given date column. It is used for auto filling dates for the user.
* An inconsistent date column can’t be selected by the user.
* For an ambiguous date column, the user needs to specify the date format of the column. (select one of 10/29/2019 or 29/10/2019)

# Appendix

[preserves the discussion while coming up with the structure -- will not be uploaded to GitHub]

## Row level suggestions

Ex. MeanVs Median

We consider this as a new category of suggestion. So, add a key is\_row\_level = True / False In the suggestion structure.

And we also add a list determining the rows that have this suggestion.

Row\_list = [rows having this oversight]

Final suggestion structure -

List of suggestions.

[

{

‘suggestion’ : ‘<Suggestion to be displayed>’

oversight\_name : ‘<As discussed it will help in getting feedback later>’

confidence\_score: ‘<As discussed it will help in getting feedback later>’

‘json’ : {<Empty if no new query suggested>}

‘is\_row\_level\_suggestion’ : <True/False>

row\_list :

* If is\_row\_level\_suggestion = True :
  + [List of dictionary {row:row\_name, confidence\_score: parameter} ]
  + Row number is returned ( header row is considered as 0)
* Else :
  + None

‘is\_column\_level\_suggestion’ : <True/False>

col\_list :

* If is\_column\_level\_suggestion = True :
  + [List of dictionary {column:column\_name, confidence\_score: parameter} ]
* Else :
  + None

}

{

# Other such suggestions

}

]

# Final Return structure

{

‘outputTable’ : 2d array

‘suggestions’: [list of suggestions](#2h52o9hufrz1)

‘slicing\_passed\_list’ : list of indices in original table that pass the slicing condition

‘list\_topk\_indices’ : list of indices in the original table that are in the top-k. // no key present if intent other than top-k or grouping is done

}

## Implementing suggested query json at GCP level

As discussed we would attach the json for the new query with the suggestion string.

How to create the new json?

Note : **as the original json contains the entire table we assume it to be very bulky.**

(The new json would be very similar to the json received. Just the values of a key or two would change)

Options

1. We pass the json received to the intent layer then to the oversights layer and each oversight makes changes to the received json & return the new json with the suggestion.
   1. Con -  
        
      It’s very **inefficient** to let the entire json flow through all the code.  
        
      Pros-  
        
      Code would ‘look’ cleaner & will be ‘easy’ to write

1. (Is the entire json even necessary to create the suggested query?)  
   Ans - NO  
   We only need to note the change suggested in the json.  
     
   So, the oversights do not return the entire json BUT only the list of changes for the new query.  
     
   Changelist = dictionary of {keys : new\_value} # here keys whose values will not change will not be included.  
     
   Pro -  
   Efficient way.  
     
   Suboption
   1. We make the changes in the original json at the GCP level & pass the new json to UI  
      Will be approximately 10 times slower as each new json will be passed through the network. (assuming almost all suggestions will contain new query)
   2. We pass the changelist to the UI. And each new query button will be mapped to it’s changelist.  
      Best as nothing redundant is being used/passed

Option 2b is the best option according to me. But 2a is also fine if changelist is difficult to implement at the UI level. 2b would be much better than 1 as it reduces the load on the GCP at least.