Interactive Data Analytics Dashboard for RDF Knowledge Graphs

Ava Sharafi, Negar Shariat, Aniqa Riaz

February 2021

Contents

1 Introduction

Interactive Analytics Dashboard for RDF knowledge graphs is an interactive dashboard for analyzing and visualizing RDF datasets. The dashboard has been deployed as a web app on heroku and provides an opportunity to visualize SPARQL query results on both local and online rdf data sets. To make the dashboard more interactive and dynamic, different features have been provided to the users to allow them to adjust the interface according to their requirements.

2 How to access

2.1 WebApp

The dashboard has been deployed as a webapp on heroku and can be accessed using this link `https://dashboard-rdf.herokuapp.com`

2.2 Local Environment

- You have to either download or clone github repository to your machine.
- Install python environment.
- Install all dependencies using the following steps:
  - open terminal
  - cd to the directory where requirements.txt is located
  - pip install -r requirements.txt
- Run dashboard through python files using the following steps
  - cd to the directory (src) where python files are located.
  - type and run the command below in a terminal.

    python main.py
- Run dashboard through Jupyter Notebooks using the following steps:
  - cd to the directory where notebooks are located.
  - type and run the command below in a terminal.

    voila main.ipynb
  - A window will open in the browser containing the link for running the app on localhost.

3 Features

3.1 Query online endpoint

Users can query any SPARQL endpoint. Then, the results can be displayed in different Formats. We already tested the dashboard with the endpoints below:

https://query.wikidata.org/sparql

http://dbpedia.org/sparql

https://agrovoc.uniroma2.it/sparql/

3.2 Query local RDF dataset

User can upload different rdf format data sets and then can run queries on them. Results can then be visualized in different forms. Different formats allowed include RDF/XML, N3, NTriples, N-Quads, Turtle, TriX and RDFa.

3.3 Tabular view of data

Result obtained from querying rdf data from online endpoint or local file can be visualized in the form of a table. To make the visualization more interactive tables include sorting and filtering of data. Option to select and delete specific rows and colunms is also provided.

3.4 Analysis of data through different plot types

The results can be displayed to the user in a form of a chart. The user can change the type of chart, the X-Axis and Y-Axis.

3.5 Analysis of data containing coordinates through maps

The results can be visualized in a form of a map, if the coordinates exist in the results. If the dashboard fails to distinguish the coordinates, still the map will be displayed and the user can select the latitudes and longitudes manually. Also, if the user hovers over coordinates on the map the selected labels will be shown.

3.6 Download results

Data in tabular form can be downloaded in csv format and all plots and charts can be downloaded as images.

3.7 Comparison of two queries and their visualization

Through this feature, the user is able to compare two queries. After submitting the first query in the main query box and the second one in the compare box, the results (including tables, charts, and maps) will be shown in front of each query and provide the possibility of visually comparing both queries' outcomes.

3.8 Query samples

Some query samples are provided for the user to test the dashboard tools in a table of query descriptions. When the user clicks on one of the queries, the query text and endpoint will appear on the query box and after submitting, the results can be seen in the table, chart, and map tabs.

3.9 Tutorial

In order to help the user use the dashboard easier, a step-by-step tutorial including screenshots and explanations of the main features is provided.

4 Structure of files

- Main files : Contains code for building all files and running server
- Layout files : Contains code for main layout and the layout of tabs
- Utility files: Contains code for different helper and utility functions used throughtout the app
- Querying files: Contains code for updating the user interface of query and endpoints textboxes after user selection and querying online and local datasets.
- Table files: Contains code to update the table tab after query result has been updated
- Chart files: Contains code to update the Charts tab after query result has been updated
- Map files: Contains code to update the Maps tab after query result has been updated

All the files and their functions are explained in detail below.

4.1 Main

- Main:
  Compiles all the files and runs the server using the app object defined in MainApp

- MainApp:
  Defines the app object which initializes the dash object in case of python files and jupyter dash object in jupyter notebook version of the code.

4.2 Layout

- Layout:
  The layout file defines the main app layout which contains query box that is fixed on the left-hand side and the tabs.

  - Query Box:
    The query box has one text area for the endpoint, upload area and one text area for the query text. There is also a submit button and a toggle button that opens and closes the compare query box. The components that are used are:

    * Textarea from dash_core_components
    * Upload from dash_uploader
    * Button from dash_html_components

  - Tabs:
    The tabs let the user navigate among the pages of the dashboard. The number of tabs and their names are defined here while the individual code for each tabs is in the tabs file. The components that are used are:

    * Tabs and Tab from dash_core_components

- Tabs:
  The tabs file contains code for the layout of each individual tab.

  - Render_content:
    Render Content function is used as a callback for the tabs to display the selected tab to user.
    It takes one input.
    Value of the tab that has been clicked by the user.
    It returns one output.
    The layout of the tab that has been selected by the user. Layout of the following different tabs have been defined and can be returned by this function.

    * Tables

* Charts
* Maps
* Query Samples
* Tutorial

4.3 Helper Functions

- Start_table_df:
  Initializes an empty pandas data frame for table in the tables tab.

- Update_output Callbacks:
  Both callbacks have the same functionality. The second callback is fired when a second query has also been submitted for compare. The callbacks are called after user has finished uploading a file for local querying to update a hidden field attribute for the endpoint which in turn will update the value in the actual query endpoint text box later on.
  Both callbacks take two inputs.
  FileNames which is a list incase multiple files are uploaded.
  IsCompleted property of upload component which is a boolean variable showing status of upload.
  Both callbacks return a single variable.
  Title which is a property of a hidden component used to set value of the query endpoint textbox.

- Log_File_Data:
  This function is currently not being used. It was initially created to log file upload time but then the upload component was changed to allow larger file uploads. The new component didn't execute any callbacks or function on start of execution which could allow for a logger to measure time. Hence this functionality had to be removed.

- Log_Query_Data:
  This function is used to log query data and the time taken to exectue a query to a csv file after a query has been executed.
  It takes fives inputs.
  The endpoint of a query, the query itself, number of rows returned in the result, number of columns returned in the result and time taken by the query.

- Log_Parse_Data:
  This function is used to log the time taken to parse a rdf file.
  It takes two inputs.
  The endpoint of a query and time taken to parse the file.

- Read Samples Files:
  Reads sample queries from a csv file and stores them in a dataframe to be used for displaying in query samples tab.

- Toggle_compare:
  Shows and hides the compare input box on user input

4.4 Querying

- Query Text Updates:
  Contains code for update of query text box and query endpoint text box on user actions.
  - Update_endpoint Callbacks:
    Both callbacks have the same functionality. The second callback is fired when a file has been uploaded for compare query. Each callback updates the query endpoint value and type according to the user input.
    Both callbacks take three inputs.
    endpoint-query1-intermediate-value1 containing the file name of the rdf file incase user has uploaded a file.
    endpoint-query1-intermediate-value2 which is a hidden variable containing the endpoint value selected by user through a sample query.
    fileNames property of upload component.
    The callbacks return two values.
    Value property of query-endpoint.
    Title property of a hidden variable containing the type of endpoint i.e. whether it is local (1 in this case) or online endpoint (2 in this case).
  - Get_active_cell Callback:
    Sets the value of query textbox and a hidden variable for query endpoint on selection in query samples.
- Query Result Callbacks:
  To perform the data processing steps including querying endpoint, getting data, extract latitude and longitude and parsing long strings in data in one callback, serialize the output at JSON, and provide it as an input to the other callbacks inside a hidden Div.
  - Callbacks:
    Both callbacks have the same functionality. The second callback is fired when the user click on compare button.
    The callback gets n_clicks property of the component html.Button that has the ID submit–btn.
    as an Input. The two first outputs are is_open property of alert_value and alert_except which dispaly errors to the user.
    The output which shares the data with other callbacks is the children property of the component with the ID intermediate–value.
    We used state in callback to pass along extra values without firing the callbacks. The values of the three components dcc.Textarea

with IDs query–text, query–endpoint and endpoint–query–type, are passed into the callback.
n_clicks property of the html.Button component fires the callback after the component has been clicked on, which means the fucntion "get_data_function" will be called if both query and endpoint boxes are not empty.

- get_data_function:
  To query online endpoint or local endpoint.
  If user uploads a local rdf file and a query (In this case, endpoint type is 1) rdflib is used to parse the file and store the parsed data in graph type structure. Then the user submitted query is run on this graph structure. If the output of the query is None then an alert message will be shown to the user and the "get_data_function" returns None. If the output is not None then no alert message will be shown and "get_data_function" serialize the result called "ResultListdataframe" at JSON and returns it.

  If users submits an online endpoint and a query(In this case, endpoint type is 2), the "sql" function will be called. If the output of the "sql" function is None then an alert message will be shown to the user and the "get_data_function" returns None. If the output of "sql" function is not None then no alert message will be shown and "get_data_function" serialize the result called "ResultListdataframe" at JSON and returns it.

- sql function:
  It gets the query and endpoint's URL as inputs.
  First, it checks if the URL is a valid SPARQL endpoint by checking if the string 'sparql' is a part of the URL. If it is, SPARQLWrapper executes the query and returns the results in JSON format. If the executing fails, SPARQLWrapper exceptions catch the errors. Then, the "sql" function returns None.

  In the case that there are results, we get the names of columns and all the values for each column. Then we save them into a pandas data frame namely "ResultListdataframe".

  After getting the results, we parse the long string in data and remove the unuseful information. For example, sometimes the URL of endpoints repeats in the results.

  If the coordinates exist in results but there are no latitude and longitude, we need to parse coordinate to extract longitude and latitude. So, we define the "extract_lot_lan" function to parse the coordinates into latitude and longitude.

4.5 Tables

- Callbacks: Both callbacks have the same functionality. The second callback is fired when a second query has also been submitted for compare. These callbacks are called when the user clicks on the table tab. The callback takes the query results and returns a dictionary of data to be shown in the table along with the column names.
  Each callback takes three inputs.
  n_clicks which is a property of the dash html button component. It is an integer that represents the number of times that this element has been clicked on. For first callback this is for submit button and for second it is for compare button.
  Jsonified_ResultListdataframe which is the result of the query. It is passed on to gen_table_function.
  derived_virtual_selected_rows that are rows that have been selected in the table by the user.It is passed on to gen_table_function.
  Each callback returns four variables.
  Is_open property of an alert for the table which tells if the table is empty and there are no results. It is a Boolean variable.
  Data property of the table which is set to a dictionary containing results of the query provided by the user.
  Columns property of the table which is set to a list of columns contained by the query result with properties deletable and selectable for dash_table.
  Style property for the table Div which contains the display variable set to 'None' if there are no results. Otherwise it is set to 'block'.

- gen_table_function: This function is used to take the JSON data frame result of the query and convert it into a pandas data frame type compatible with dash table.
  It takes two inputs.
  Jsonified_ResultListdataframe which is the result of the query.
  derived_virtual_selected_rows that are rows that have been selected in the table by the user. When the table is first rendered,'derived_virtual_selected_rows' will be 'None'. This is due to an idiosyncrasy in Dash (Dash calls the dependent callbacks when the component is first rendered). So, if rows is 'None', then the component was just rendered and its value will be the same as the component's dataframe.
  Output contains two variables.
  resultListdataframe which is a pandas data frame containing the result of the query in the form readable by the dash_table.
  mycolumns which is a List of columns contained by the query result with properties deletable and selectable for dash_table.

4.6 Charts

- Callbacks:Both callbacks have the same functionality. The second callback is fired when the user click on compare button.
  The inputs of these callback are n_clicks property of the component html.Button that has the ID submit–btn.
  and the value property of the hidden Div component that has ID intermediate–value.
  The first output is is_open property of alert_chart dispaly an error to the user if such an errors exist. The last two outputs are the figure and style properties of the dcc.Graph and html.Div components respectively.

  When the user clicks on the Chart's tab, this callback will be fired. After that, if the submit button has been clicked, then the "gen_graph" function will be called.

- gen_graph:
  This function gets the results as an input. First, it checks if the results are empty it returns None and also displays an error to the users.
  We also have to check if there is more than one column in the results. If there is, we use the first and second columns to generate the chart.
  We also used update menus to modify the type of chart and the X–axis and Y–axis.

4.7 Maps

- Callbacks:Both callbacks have the same functionality. The second callback is fired when the user click on compare button.

  The inputs of these callbacks are n_clicks property of the component html.Button that has the ID submit–btn.
  and the value property of the hidden Div component that has ID intermediate–value.
  The first output is is_open property of alert_map display an error to the user if such an errors exist. The last two outputs are the figure and style properties of the dcc.Graph and html.Div components respectively.

- gen_map_function:
  This function gets the results as an input. First, it checks if the results are empty it returns None and also displays an error to the users.
  We also have to check if there is more than one column in the results. If there is and also we have latitudes and longitudes in the data, the "plot_map_function" will be called which has latitudes and longitudes as input.
  If we don't have latitudes and longitudes, the "plot_map_function" will be

called again but its inputs are the first and second columns. So, the user can change the latitudes and longitudes manually by dropdowns if they exist but the "get_data_function" failed to distinguish them.

- plot_map_function:
  We use Plotly graph_objs to generate the map and update menus to modify the hover labels, latitudes, and longitudes.

5 Github Repository

All the code files and documentations have been uploaded to the github repository. The link for the repository is `https://github.com/avasharafi/Semantic-Data-Web` There are two branches in the repository.

- Main:
  This is the main branch containing both the python and jupyter notebook versions of the code. The structure of the branch is as follows:
  - data folder: contains some query samples and two sample files to upload.
  - Notebook folder: contains .ipynb version of code.
  - asset: contains style sheet.
  - src : contains python source codes of dashboard.
- Python_files:
  The web application is deployed on this branch.

6 Issues

- RdfLib Limitations:
  There are some limitations to rdflib parsing and function functionalities. A file above 300 mb takes about 15 minutes to parse and 5 minutes for querying.
  Also another query limitation noted is that the group by clause does not accept variables inside a bracket. For example groupby ?name is acceptable. groupby (?name) is not acceptable. This is a bug in rdflib. This case is for local querying as only local querying is done using rdflib.

- Logging File Upload Time:
  File upload time was initially logged when dash native upload component was used. Unfortunately that component did not allow upload of large size files so another component dash-uploader was used. This component allowed large file sizes to be uploaded and also showed progress of file upload on user interface. Only issue faced was that it had no attribute for time taken for upload or a function that was called when file upload began. The only callback function was for when file upload was completed hence the total time taken could not be measured or logged corrrectly.

7 Improvment

- Responsive layout

- Statistics

- Modify the functionality of the dashboard based on datatype. For example, in a case that there are no numerical values in the results, disable the Chart and Maps tab.