GeoData Visualization & Analysis Framework

guoqianz / jiajunzh

Framework Domain

It is not uncommon to have data associated with regions, e.g. population distribution across a country, temperature changes over different states. This framework interacts with geo-data by extracting geo-data from various sources, processing and visualizing it. Users can perform geocoding, filtering or sorting function on the import dataset before visualizing it.

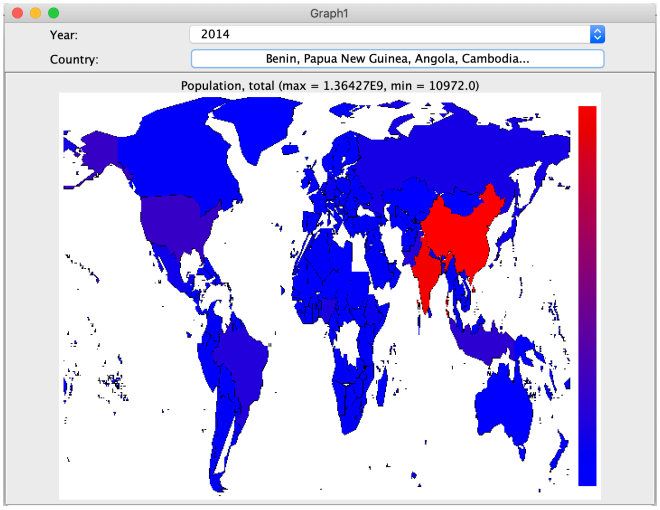

Figure 1 - choropleth map

Running Framework

Effort needed to run our framework is trivial. Users should first add dependencies of our framework into their project. First, users should add below code into build.gradle file of their project.

```
apply plugin: 'java'
apply plugin: 'maven'
apply plugin: 'application'

repositories {
    mavenCentral()
}

dependencies {
    compile project(':framework')
}
```

```
mainClassName = 'edu.cmu.cs.cs214.hw5.Main'
```

Then users should add below code into settings.gradle file of their project.

```
include ':framework'
project(':framework').projectDir = file('../framework')
```

Once dependency is added, users can simply run the framework by using command below.

```
$ gradle run
```

Usage Guide

In this section, we will provide a framework usage guide to help users navigate through the framework GUI.

- **Import Data**

To import data, user should click the top left “File” menu and choose any one data plugin adaptable to their data source in the “Import Data” option.

For example, “File Reader” example data plugin we provide can be used to extract data from a file source. Once the plugin is selected, a window will pop up to let user specify usage-specific configurations or information. In this example, user should provide the file path, file delimiter and name of the imported dataset.

After user clicks “OK”, the newly imported dataset entity will be displayed in panel. User can further display or transform the dataset by clicking the dataset entity.

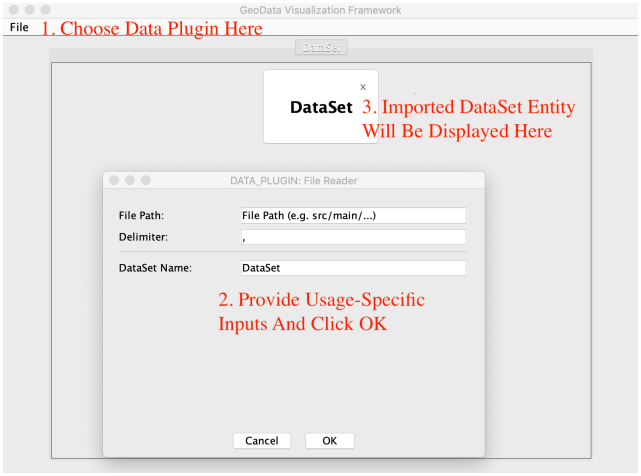

Figure 2 - dataset import instructions

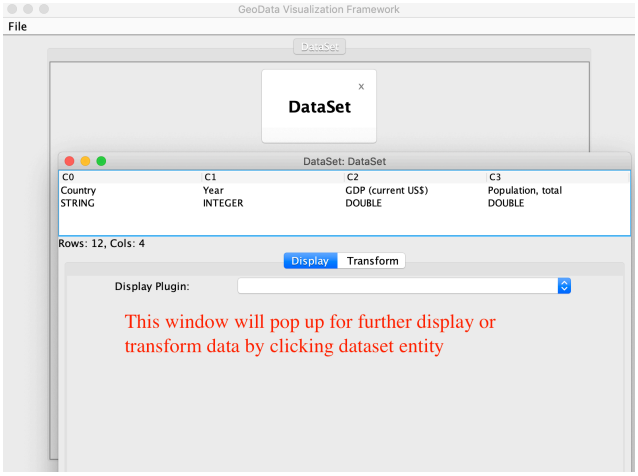

Figure 3 - control panel for display or transform

● Data Transformation

Our framework provides three kinds of transformation: geocoding, filtering and sorting.

✓ GeoCoding

Geocoding converts address representation to geo-coordinates and boundary information of a place. Applying the geocoding transformation will generate a new dataset containing longitude, latitude and area columns apart from the columns in original dataset.

In order to geocode dataset, user needs to navigate to geocoding panel by clicking “Transform” and “GeoCode” tab and specify which column(s) is the address-relevant column(s) that will be geocoded and new label name for newly generated columns and name of new dataset.

Two kinds of forms are provided here. Rigid Form let users specify rigid-format address field (e.g. country, state, city, county, street), users need to provide at least one fields column. Free Form, however, lets user provide a column as required free-format address.

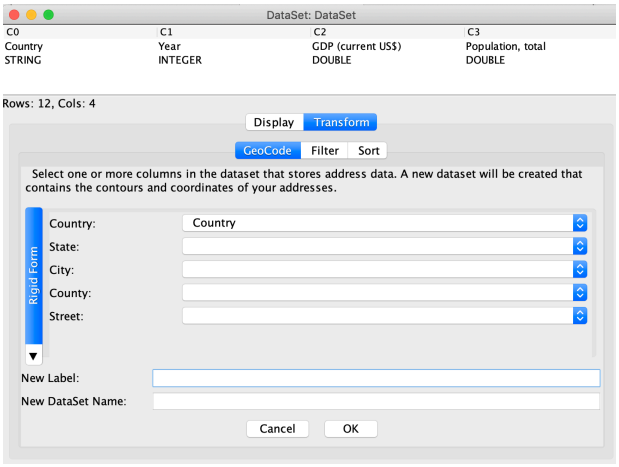

Figure 4 - rigid form for geocoding

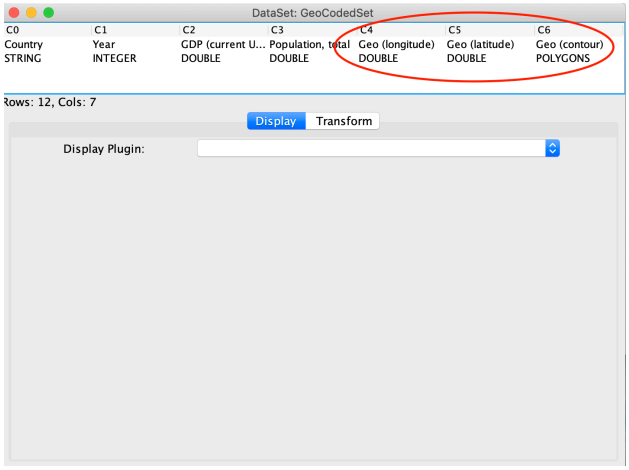

Figure 5 - new dataset info after geocoding

✓ Filtering

Filtering generates a new dataset with only data entries satisfying filtering constraints in the original dataset.

In order to filter dataset, user needs to navigate to filtering panel by clicking “Transform” and “Filter” tab, specify name of new dataset, filtering constraints and which column(s) will be filtered on.

Similarly, two kinds of forms provided here to allow user specify filtering constraints. Numeric Form only filters columns of Integer or Double type (user will not be able to choose column

of String type for filtering in this form) and requires user to provide operator and constraint value as filtering rule (e.g. “>= 2.0”). String form however, only filters columns of String type, and once a column is selected, all possible values in this column will show distinctly in the “Values” menu which can be selected by users. The dataset rows containing the values selected will remain in the filtered dataset.

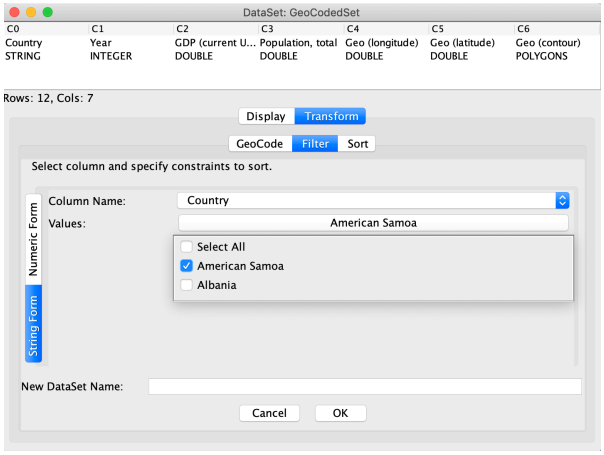

Figure 6 – string form for filtering

✓ Sorting

Sorting generates a new dataset which is sorted by the column specified by user. In order to sort dataset, user needs to navigate to sorting panel by clicking “Transform” and “Sort” tab, specify name of new dataset and the column by which the dataset will be sorted.

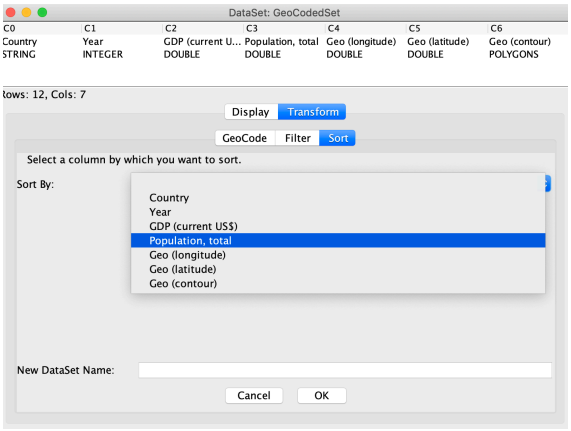

Figure 7 - control panel for sorting

● Visualize Data

To visualize specific dataset, click on the dataset displayed in the dataset panel and a display control panel will pop up. Users should navigate to display panel by clicking “Display” and choose a display plugin they want to display dataset.

Once users select a display, a bunch of plugin-specific configuration input component will pop up. In the choropleth map example we provide here, the inputs that users need to specify

include name of the graph, “Area Name” (column representing the name of an area), “Area” (a column containing boundary information of this area), “Value” (a column whose values will be shown by color in the map) and an optional field “Time” (a column representing time, user can select only to show the data of some years in the dataset later). After clicking “OK” button, dataset will be displayed according to selected display plugin. User can also select to only show data at some timestamps and some areas (circled as red below) in real time.

Notice that choropleth map requires “Area” field, thus is constrained to dataset containing the area boundary info (if dataset do not have “Area” field, the respective drop-down menu will be empty, thus it cannot be displayed by this plugin). However, to visualize geo-dataset with no “Area” field by choropleth map, users can first use geocoding transformation functionality (geo-dataset should at least have information about area name or address) our framework provides and then display dataset respectively.

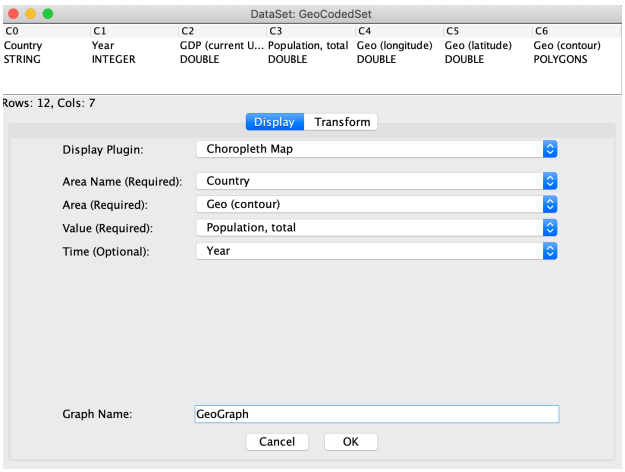

Figure 8 - control panel for display plugin

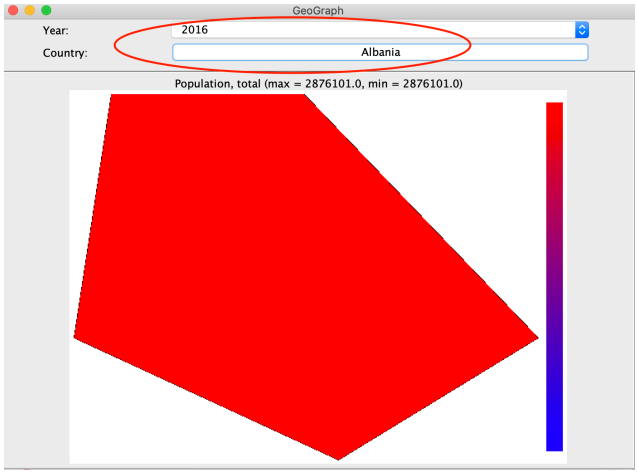

Figure 9 - graph window

Plugin Writing Guide

In this section, we will provide guide for writing plugins for this framework. Before we get into the plugins, client should first understand three classes that will be used in plugin writing: DataSet, UserInputConfig and DisplayFilterConfig.

- **DataSet**

DataSet is an immutable class that abstracts and stores the geo-dataset into the framework. It decouples the data plugins and display plugins by its abstraction. DataPlugin has to convert their data from their source into this class. DisplayPlugin will query the dataset for data visualization.

To create a DataSet, client has to provide a list of column names, a list of data types (it can be string, integer or double) of each column and data entity itself as well. Data entity is

represented by a list of data rows (represented by a list of Object in each row). Suppose we have a dataset below:

col1(String)	col2(Integer)	col3(Double)
A	1	2.9
B	2	4.1

An example code to create a DataSet object for this dataset will be:

```
List<String> labels = Arrays.asList("col1", "col2", "col3");
List<DataType> dataTypes = Arrays.asList(DataType.STRING, DataType.INTEGER,
DataType.DOUBLE);
List<List<Object>> data = new ArrayList<>();
List<Object> row1 = Arrays.asList("A", 1, 2.9);
List<Object> row2 = Arrays.asList("B", 2, 4.1);
data.add(row1);
data.add(row2);
DataSet dataSet = new DataSet(labels, dataTypes, data);
```

DataType represents the type which respective data will be stored as, it can be any one of STRING, INTEGER, DOUBLE, POLYGONS.

- **UserInputConfig & DisplayFilterConfig**

In our design, instead of making writing plugins a lot of work by keeping client responsible for creating GUI component (e.g. ComboBox) for plugin-specific configurations or usage-specific information, we provide these two defined classes which client can use aiming to tell our framework which configuration GUI components are needed by simply providing the component name, component type and a candidate list which may show in the drop-down menu.

✓ **UserInputConfig**

UserInputConfig lets users specify plugin-specific configuration or usage-specific information (e.g. file path, dataset column users are interested). There are three types of GUI components this UserInputConfig corresponds to. Plugins can use three types defined in UserInputType to tell framework which type of component should be displayed.

TEXT_FIELD tells the framework to display a text field component and the choice list parameters used to create UserInputConfig will be discarded by the framework. The code below tells framework to create a text field to allow user to enter the file path.

```
new UserInputConfig("File Path", UserInputType.TEXT_FIELD, null)
```

SINGLE_SELECTION tells the framework to display a drop-down menu where user can only select one entry in the menu. The code below tells the framework to create a single selection drop-down menu which displays elements in topics list in the menu.

```
List<String> selections = Arrays.asList("GDP", "Population", "CO2 Emissions");
new UserInputConfig("Topic", UserInputType.SINGLE_SELECTION, selections);
```

MULTISELECTION tells the framework to display a drop-down menu where user can select one or multiple entry in the menu. The code below tells the framework to create a multi-selection drop-down menu which displays elements in topics list in the menu.

```
List<String> selections = Arrays.asList("GDP", "Population", "CO2 Emissions");
new UserInputConfig("Topic", UserInputType.MULTI_SELECTION, selections);
```

✓ **DisplayFilterConfig**

DisplayPlugins are also allowed to provide a list of DisplayFilterConfig by which user can specify which value in a column to display in real time. Framework will create GUI component respectively depending on DisplayFilterConfig in the graph window to allow user to choose which rows of data should be displayed in the graph.

The drop-down menu in the figure 8 and 9 is an example of this. In the display plugin control panel, user specifies “year” column as Time. If the dataset “year” column has value [2010, 2011, 2012… 2019], then all values in the list will be displayed in the drop-down menu in figure 9, and user can select to decide which year of data should be displayed in the graph. Each time the user changes the selected values, the framework will call “draw” API in display plugin to redraw a new graph.

Differently, client only needs to provide label name and UserInputType to create a DisplayFilterConfig object. The framework will get the choice list by the label name specified afterwards to both save work of client as well as information hiding. The code below creates a DisplayFilterConfig object to tell framework to create a single selection drop-menu whose label name is YEAR in the graph window.

```
// Suppose "year" is a label name in the dataset
new DisplayFilterConfig("year", UserInputType.SINGLE_SELECTION);
```

● DataPlugin

In this section, we will use the code snippet (you can also reference for the whole code in example code we provide) in the example data plugin FileReaderPlugin to provide guide for

DataPlugin writing. This is just a simple plugin which extracts dataset from file source. The DataPlugin interface has three APIs shown as below.

```
public interface DataPlugin {
    String getName();
    List<UserInputConfig> getUserInputConfig();
    DataSet loadData(Map<String, List<String>> params);
}
```

In order to implement this plugin, firstly, plugin should return its name back to the framework which will be displayed later.

Then, plugin could provide a list of UserInputConfig to customize the plugin-specific usage configuration or information (e.g. file path, dataset name ...) that user needs to specify. The plugin needs to specify configuration name, input type (TEXT_FIELD, MULTI_SELECTION or SINGLE_SELECTION) and a list of selections if any. The concrete parameters that user specifies can be fetched later when loading data. In this example, the plugin allows users to specify the file path and file delimiter.

```
@Override
public List<UserInputConfig> getUserInputConfigs() {
    List<UserInputConfig> options = new ArrayList<>();
    options.add(new UserInputConfig(FILE_PATH, UserInputType.TEXT_FIELD, null));
    options.add(new UserInputConfig(DELIMITER, UserInputType.TEXT_FIELD, null));
    return options;
}
```

Once user specifies their parameter values and confirms, the plugin will be told the user's input specified by itself earlier.

```
@Override
public DataSet loadData(Map<String, List<String>> params) {
    String path = params.get(FILE_PATH).get(0);
    String delimiter = params.get(DELIMITER).get(0);
```

The plugin should load data as a DataSet object. When the plugin finished reading the data, It should create a DataSet with labels, data types and data entity.

```
List<String> labels = Arrays.asList(columnLine.split(delimiter));
```

```
List<DataType> types = Arrays.stream(typeLine.split(delimiter))
       .map(DataType::valueOf)
       .collect(Collectors.toList());

List<List<Object>> data = new ArrayList<>();
while(scanner.hasNextLine()) {
   List<Object> row = new ArrayList<>();
   String[] valuesInLine = scanner.nextLine().split(delimiter);
   for (int i = 0; i < columnNum; i++) {
      switch (types.get(i)) {
         case STRING:
            row.add(valuesInLine[i]);
            break;
         case INTEGER:
            row.add(Integer.parseInt(valuesInLine[i]));
            break;
         case DOUBLE:
            row.add(Double.parseDouble(valuesInLine[i]));
            break;
         default:
            break;
      }
   }
   data.add(row);
}

DataSet dataSet = new DataSet(labels, dataTypes, data);
return dataSet;
```

With the list of labels, list of data types as well as the data entity, now you are ready to create a DataSet and return it back to the framework!

- **DisplayPlugin**

In this section, we will use the code snippet in the example display plugin ChoroplethMap to provide guide for DisplayPlugin writing. The example plugin will display data as a choropleth map.

DisplayPlugin interface has four APIs shown as below:

```
public interface DisplayPlugin {
    String getName();
    List<UserInputConfig> getPluginConfigs(Map<DataType, List<String>> columnPreview);
    List<DisplayFilterConfig> getDisplayFilterConfig(Map<String, List<String>>
pluginParams);
    JPanel draw(DataSet dataSet, int width, int height, Map<String, List<String>>
pluginParams);
}
```

Similarly, plugin should provide its name to framework first and then a list of UserInputConfig to customize the plugin-specific usage configuration or information. The code snippet below notifies the framework to create four configuration components with labels named “Area Name”, “Area”, “Value” and “time”. For example, “Area Name” is a single selection drop-down menu with only column of string type will show up. Note the columnPreview passed in by the framework is a mapping from DataType to a List of labels of column of that DataType. Plugin can get a list of column labels whose column is of the required data type, which will be served as a choice list.

```
@Override
public List<UserInputConfig> getPluginConfigs(Map<DataType, List<String>> preview) {
    List<UserInputConfig> configs = new ArrayList<>();

    configs.add(new UserInputConfig("Area Name", SINGLE_SELECTION, preview.get(STRING)));
    configs.add(new UserInputConfig("Area", SINGLE_SELECTION, preview.get(POLYGONS)));
    List<String> numericLabels = new ArrayList<>(preview.get(DOUBLE));
    numericLabels.addAll(preview.get(INTEGER));
    configs.add(new UserInputConfig("Value", SINGLE_SELECTION, numericLabels));
    configs.add(new UserInputConfig("Time", SINGLE_SELECTION, preview.get(INTEGER)));

    return configs;
}
```

The plugin can add other non-column-specific configurations if needed, such as letting user choosing the paint color. The steps are the same with those of DataPlugin.

Different from the DataPlugin, DisplayPlugin also provides an API allowing plugin to provide a list of DisplayFilterConfig to let user control and filter the data which is being displayed in real-time. The framework will call this method after the user has completed plugin-specific usage configurations (user inputs can be fetched from the mapping pluginParams) and before graph is going to be drawn. The plugin only needs to specify the labels of controllable columns and how the user can control them (choosing a single value or multiple values), so that in

later times the framework will handle the user's actions and call the plugin to draw the plot with the data that is already processed. The code snippet below tells the framework to create a single selection drop-down menu for time column values and a multi-selection drop-down menu for area name column values.

```
@Override
public List<DisplayFilterConfig> getDisplayFilterConfig(Map<String, List<String>>
pluginParams) {
    checkParams(pluginParams);
    List<DisplayFilterConfig> configs = new ArrayList<>();
    if (!pluginParams.get("Time").isEmpty()) {
        String timeLabel = pluginParams.get("Time").get(0);
        configs.add(new DisplayFilterConfig(timeLabel, SINGLE_SELECTION));
    }
    String nameLabel = pluginParams.get("Area Name").get(0);
    configs.add(new DisplayFilterConfig(nameLabel, UserInputType.MULTI_SELECTION));
    return configs;
}
```

Finally, the framework will call plugins to get a JPanel of given width and height which contains the plots that display data based on the user configuration inputs (the user inputs can be fetched from the mapping pluginParams). Then framework will call this function and display this panel properly. Note that the loaded dataset is already filtered by framework based on filter configs it previously gave to framework and the user's current selection.

```
@Override
public JPanel draw(DataSet dataSet, int width, int height, Map<String, List<String>>
pluginParams) {
    // Create a JPanel to be displayed.
    return new JPanel();
}
```

Adding New Plugin

After finishing writing the data plugins or display plugins, clients can add their own customized plugins into our framework. It is trivial to add new plugins into the framework without modifying the framework. Only thing the clients should do is to provide full class names of new plugins in file

src/main/resources/META-INF/services/edu.cmu.cs.cs214.hw5.core.DataPlugin

src/main/resources/META-INF/services/edu.cmu.cs.cs214.hw5.core.DataPlugin

in the directory of clients' project. Then the framework will load the plugins for clients after running the framework.