# Ebola Open Linked Data Validation

**Team Members:** Jonathan Downs
Yash Pant

**Instructor:** Dr. Edward Fox

**Client:** S.M.Shamimul Hasan

CS 4624
Virginia Tech
Blacksburg, VA 24061
5/9/2016

## Table of Contents

## 1. Table of Tables

# 2. Table of Figures

## 3. Abstract

In 2014, the Ebola virus, a deadly disease with a fatality rate of about 50 percent, spread throughout several countries. This marked the largest outbreak of the Ebola virus ever recorded. Our client is gathering data on the Ebola virus to create the largest database of information ever made for this disease. The purpose of our project is to verify our client's data, to ensure the integrity and accuracy of the database.

The main requirements for this project are to locate multiple sources of data to be used for verification, to parse and standardize multiple types of sources to verify the data in our client's database, and to deliver the results to our client in an easy-to-interpret manner. Additionally, a key requirement is to provide the client with a generic script that can be run to validate any data in the database, given a CSV file. This will allow our client to continue validating data in the future.

The design for this project revolves around two major elements: the structure of the existing database of Ebola information and the structure of the incoming validation files. The existing database is structured in RDF format with Turtle7 syntax, meaning it uses relational syntax to connect various data values. The incoming data format is in CSV format, which is what most Ebola data is stored as. Our design revolves around normalizing the incoming validation source data with the database content, so that the two datasets can be properly compared.

After standardizing the datasets, data can be compared directly. The project encountered several challenges in this domain, ranging from data incompatibility to inconsistent formatting on the side of the database. Data incompatibility can be seen clearly when the validation data matches the date range of the database data, but the exact days of data collection vary slightly. Inconsistent formatting is often seen in naming conventions for the data and the way that dates are stored in the database (i.e., 9/8/2014 vs. 2014-09-08). These issues were the main hindrance in our project. Each was addressed before the project could be considered complete.

After all data was converted, standardized, and compared, the results were produced and formatted in a CSV file to be given to our client. The results are given individually, for each time the script is run, so if the user runs the script for 4 different datasets over 4 different sessions, there will be 4 different result files.

The second main goal of our project, to produce a generic script that allows the user to validate data on his own, uses all previously mentioned design elements such as parsing RDF and CSV files, standardization of data, and printing results to a CSV file. This script

builds a GUI interface on top of these design elements, providing a validation tool that the users can employ on their own.

# 4. Introduction

In 2014, the world faced the largest outbreak of the Ebola virus ever recorded in the history of mankind. The Ebola virus is a deadly disease with an average fatality rate of about 50 percent.[1] The disease can spread easily through human fluids. If a person has contact with any contaminated fluids such as contaminated blood, saliva, urine, and feces, the disease can spread. The disease can also spread through contaminated needles and syringes and through diseased animals such as diseased fruit bats or primates.[2]

It is due to the ease at which Ebola spreads and due to its high fatality rate that makes Ebola so dangerous. For this reason, we want to have an easily accessible, homogeneous database of knowledge for information surrounding this disease.

Epidemiologists and Ebola researchers have published a significant amount of data related to the disease following the recent outbreak. Much of this data is scattered throughout the internet, disorganized but inherently connected. If online Ebola datasets were linked, this would create a common dataset that could provide epidemiologists with insight into the mysteries of Ebola that are yet to be resolved.

Our client, Mr. Shamimul Hasan, is a member of the Network Dynamics and Simulation Science Laboratory. Mr. Hasan is currently working on an academic project that involves a database of linked data about Ebola. He is working on connecting various datasets on Ebola to compile the data into a single easy-to-access database. However, it is not easy to be sure that the data gathered for these datasets is accurate. Therefore, our job is to validate this data by finding several other sources and cross-referencing them with the current data in the datasets.

## 4.1. Purpose

Although a database of linked Ebola database has been prepared, the integrity of the data in the database has not yet been verified. Our task contains two major elements: to identify new sources of Ebola data that can verify the data in the database, then programmatically validate the data and to create a generic script that can let the user input his or her own CSV file to verify data. When validating data, our script recognizes if the database contains a field that says there were 1000 deaths caused by Ebola in November of 2014, then compares this value to another data source that mentions the number of deaths caused by Ebola in November of 2014, and determines if it is the same as the value in the database. If the values matched, the data is valid, otherwise the data is invalid. Our goal is to provide our client a script which allows him to enter validation sources and receive a report that gives the comparison of data fields in the Ebola data with the data fields of the validation source.

### 4.2. Scope

The scope of this project is to identify data sources for validation, then creating a program that validates the Ebola Linked Open Data (LOD) using these data sources. Our program is generic, meaning that it works on the entire database and is able to compare data fields from any CSV file as a source of validation, with assistance from a user entering some information as an input to the script. Our script does not modify or reconfigure the existing Ebola database in the process of validating Ebola data.

# 5. Requirements

The goals of this project are to validate the data in the Ebola database in its current state and to provide a generic script for user to perform self-validation. Both goals involve identifying data sources that can sufficiently validate the data and programmatically compare data values between these external sources and the Ebola database.

## 5.1. Functional Requirements

**A. Parse CSV and RDF files**

The script must be able to properly read in .csv[3] files and compare to a file formatted in RDF[6] format. CSV files are the required format for validation sources, and our program assumes that the CSV the user uploads is a well formatted CSV. This means that the entered CSV must be formatted by the standards for CSV's.

**B. Standardization of Data**

The script should format source data values so that they can be compared to data in the Ebola database. The generic script attempts to standardize generic pieces of data by prompting the user for input to help. For fields such as dates, the script asks the user to provide the format of the date so that it can then standardize different date formats (i.e., 1/2/2014 vs. 2014-01-02).

**C. Verification of Data**

The script returns the values in the database that are validated. To do this, the script will look for pieces of data that completely match up, compare the values, and output the results. The metrics that we have validated are shown in Appendix C. All other datasets will need to be validated by the user through the generic validation script.

## 5.2. Nonfunctional Requirements

### A. Identify Data Sources for Validation

A data source must be in CSV format for our script to properly run. One goal is to identify data sources on Ebola that are not currently in the database, so that the values in these sources can be compared to database values to check data integrity.

### B. Documentation

We need to provide documents containing information on the data sources we used for validation as well as documents that describe the algorithms used to validate the data.

### C. Extensibility

One important aspect to our project is extensibility. It is likely that we will not be the last people working on this project so we want to make our script as extensible as possible. Since our script may not address all future problems and datasets, we want our script to be edited easily to allow for the new data to be evaluated.

### D. Timeline

Our project follows a timeline that we've composed in collaboration with our client. The tasks we must complete include understanding the current database, identifying data sources, and creating and testing script prototypes. The detailed timeline can be found in Table 1, under Appendix A.

### E. Hardware Requirement

To run this script, a PC with at least 4GB of RAM is required. An Intel i5 processor or equivalent is also recommended. Most modern PC's will meet this requirement.

# 6. User's Manual

## 6.1. Intended User

The intended user for this program is someone with access to the Ebola database that is able to identify and locate sources of validation for the database. The user would need to know how the RDF database is organized in order to properly give inputs into our program so that the script can run with no error.

## 6.2. Requirements

To run the script to generate its output file (the result of the project), the user must have validation sources, the Ebola database, as well as Python 2.7 and the associated libraries installed (listed below). The program has been tested on Windows 8.1 and is guaranteed to work on this OS, but has not been tested on other operating systems. The user must also have the database in either Turtle (.ttl) or RDF (.rdf) file format and validation files in CSV format. These files must follow the syntax and standards for their respective file types.

The following libraries re necessary for the script to run:

- Pandas[12] - A library used for parsing and manipulating CSV data. To get this, *run the command "pip install pandas" in the command line.*
- Numpy[5] - Used for data homogenization between the CSVs and the RDF database. No additional install is needed for this.
- RDFlib[9] - A library used to read and search through the RDF database. To get this, to get this, *run the command "pip install rdflib" in the command line.*
- Python Image Library – A library used to display images in a GUI setting. To get this, *run the command "pip install image" in the command line.*

## 6.3. Running the Program

To run the script, you can simply run the *EbolaScript.py* file. This will open a GUI interface for the user to use to validate the file. The GUI requires that the user has the following:

- A database file (.ttl or .rdf file)
- A validation file (.csv file)
- Python 2 and all libraries properly installed on the machine

Once the GUI is loaded, it will prompt the user for the database and validation files. Figure 1, depicts this page of the GUI. If the user loads a large database file, the program may stall for a while before it will continue. After the database is loaded, the datasets should populate under an option menu on the page. The user should choose which dataset corresponds to the data they are validating and the data in the validation file. At this point, the user can press the button that says “Submit.”

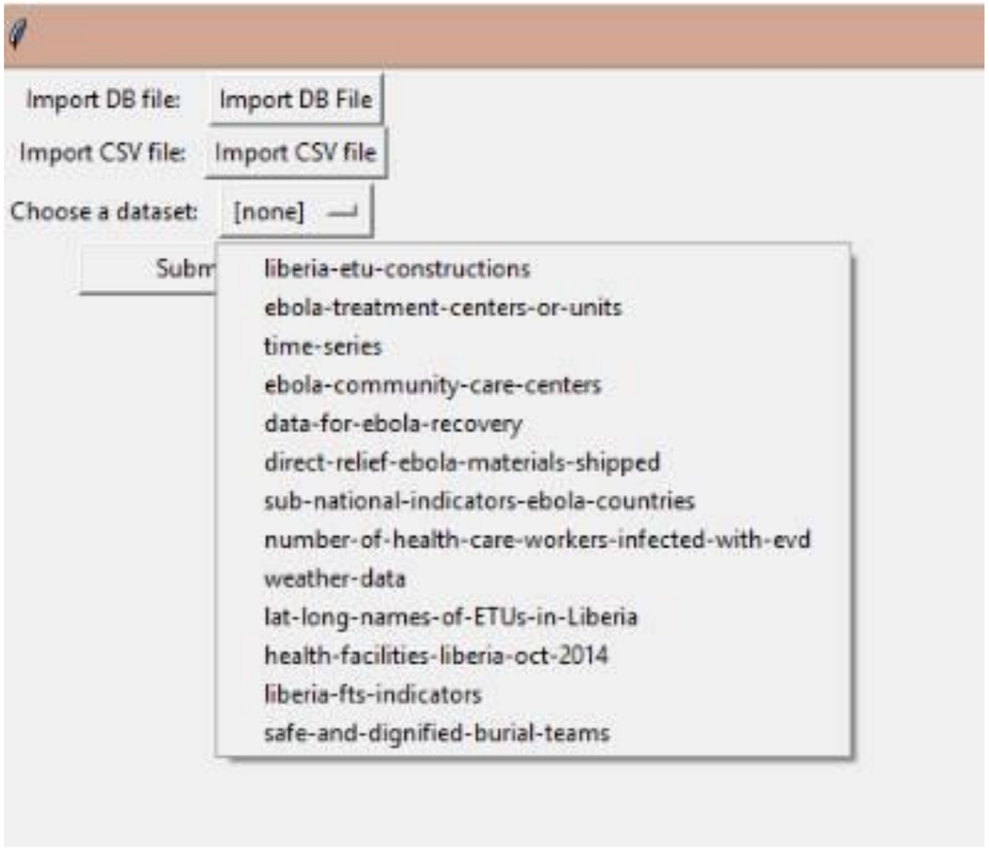
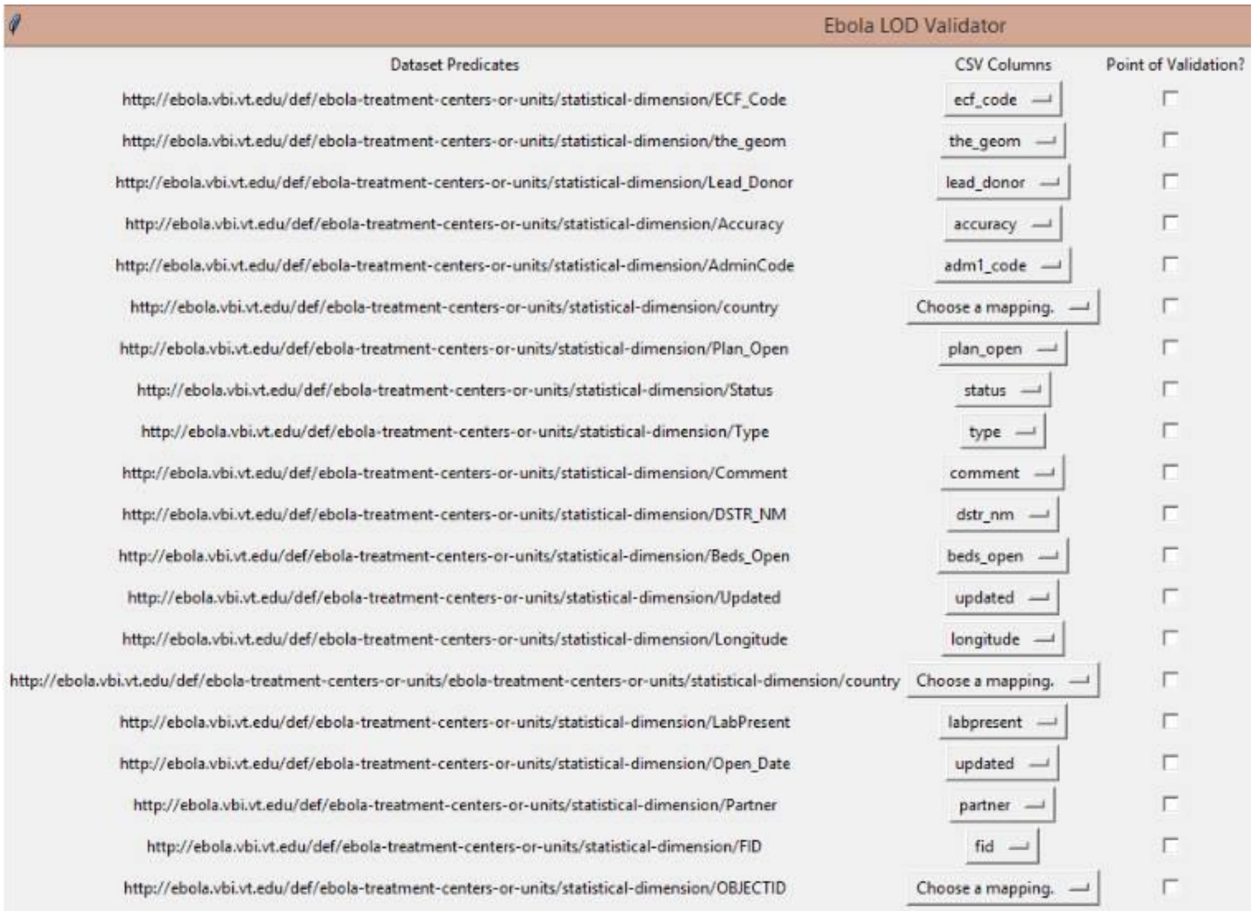

Import DB file:
Import DB File
Import CSV file:
Import CSV file
Choose a dataset:
[none]
Subm
liberia-etu-constructions
ebola-treatment-centers-or-units
time-series
ebola-community-care-centers
data-for-ebola-recovery
direct-relief-ebola-materials-shipped
sub-national-indicators-ebola-countries
number-of-health-care-workers-infected-with-evd
weather-data
lat-long-names-of-ETUs-in-Liberia
health-facilities-liberia-oct-2014
liberia-fts-indicators
safe-and-dignified-burial-teams

*Figure 1: First page of GUI*

At this point, the user is brought to the second page of the GUI, shown in Figure 2, on the next page. This page displays the predicates extracted from the database’s dataset and the CSV column names. On each line are two checkboxes, one asking if the row is a *point of validation* and one asking if the row requires an *object mapping*. The user should check the “Point of Validation” checkbox if the predicate is the one that they want to validate (this is usually a value of some kind). For example, if the user wants to validate the number 1000 as the number of deaths in Liberia on a given month, they would select its column as the point of validation. The user should check the “Object Mapping” checkbox if the values under a column in the CSV are different than how they appear in the RDF database.



Ebola LOD Validator

| Dataset Predicates | CSV Columns | Point of Validation? |
|---|---|---|
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/ECF_Code | ecf_code | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/the_geom | the_geom | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Lead_Donor | lead_donor | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Accuracy | accuracy | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/AdminCode | adm1_code | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/country | Choose a mapping. | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Plan_Open | plan_open | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Status | status | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Type | type | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Comment | comment | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/DSTR_NM | dstr_nm | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Beds_Open | beds_open | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Updated | updated | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Longitude | longitude | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/ebola-treatment-centers-or-units/statistical-dimension/country | Choose a mapping. | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/LabPresent | labpresent | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Open_Date | updated | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/Partner | partner | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/FID | fid | ☐ |
| http://ebola.vbi.vt.edu/def/ebola-treatment-centers-or-units/statistical-dimension/OBJECTID | Choose a mapping. | ☐ |

*Figure 2: Second page of GUI*

On the second page, the user also needs to select the CSV column that corresponds to the predicate in the database. If the database has a predicate like "http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE", we can infer a few things from this: the user most likely chose the time series dataset to verify data on and the corresponding CSV column will likely have dates and be titled something like "Date". For dates, specifically, the GUI will ask the user for a date format. This format is given in text format and is required for the program to understand the incoming dates so that it can properly convert these to the format of the database. After this is done, the user can move on.

After the submit button is pressed on the second page, there are two possibilities. In the first case, the program may begin its validation process and may produce a result file immediately. In this case, the user is done and the results can be viewed after computations have been finished. In the second case, the user will be led to a third screen. This screen is brought up if the user indicated that there were object mappings required. On this page, the user will map object names in the database to names in the CSV file. This will be the final step for the program to understand how to interpret the data. After this page is submitted, the program will run and produce a result file.

# 7. Design

Our overall approach to designing the data validation program is to have a single script that can be easily run and is able to work with any dataset in the database with any validation source in CSV format. The final design was centered on a way to have a user who knows little about coding to be able to validate the Ebola database, so long as they understand what the data in the Ebola database is. This is why the final design is graphical and has simple options for the user to input.

The underlying code of the program works by parsing through the database file, validation file, standardizing the two files, and then comparing and outputting the results. The results are generated to a CSV file with columns containing several different fields such as: the values of the database data, the values of the validation data, the difference between the two values, and whether or not they are close or not. This CSV file can be easily read by the user and is easy to make into a graph that can give the user a better understanding of the data over time.

For our initial prototype, we focused on validating data in the database with the objective of validating all data in the database. Since the amount of data was large and the number of viable, easy-to-locate sources was small, we decided to focus on building a general script that allows a user to continue to validate the data himself, even as the database changes. This makes it possible to validate the script even if the database continues to grow.

If there are multiple data sources for a dataset with disagreeable values, the user must decide what to do in this situation. Our program will provide a different result file for each data source, allowing the user to display each set of results separately. If these end up being very different results, the user may decide to choose the more reliable source. The scope of our project does not include making decisions for this scenario, therefore it is essential that the user is someone who thoroughly understands the Ebola database and the sources surrounding it.

## 7.1. Technologies Used

### A. Resource Description Framework (RDF)

The database with which we are working uses the RDF data model, which is an efficient way to store interlinked data. The database creates a graph data structure in its backend which connects any linked information that is entered. The data can then be queried to get connected information about the data set, which may not have before been intuitive to extract manually, but is made possible through the connected graph elements were produced as a result of information entering and being processed by the RDF database.[6]

The RDF model utilizes the subject-predicate-object notation to express the data. This is the general format for all facts that are in the Ebola database. For example, it would be valid to enter the sentence “Ebola is dangerous” because here the subject is “Ebola”, the predicate is “is” and the object is “dangerous.”[6]

For our project, we need to be able to parse through this RDF data to retrieve the data for validation. Specifically, we need to be able to pick out individual data fields in the database and match them up to sourced data for validation. For example, if we find source data to validate the number of deaths in Liberia on a specific date, we need to be able to parse the RDF data for the corresponding data entry so the values can be compared.

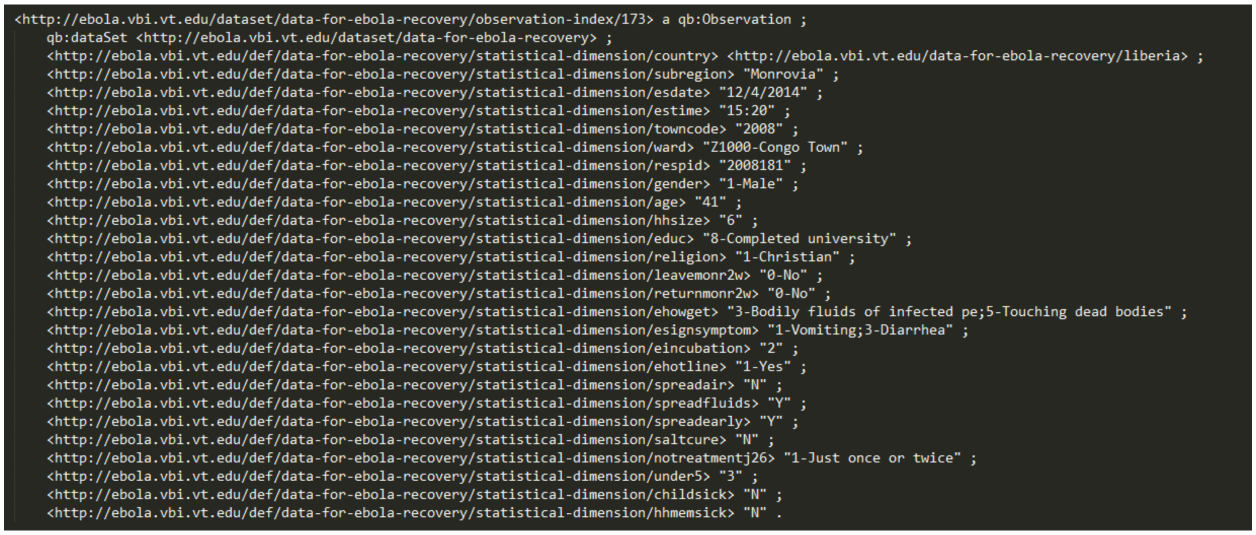
## B. Turtle

Turtle[7], or Terse RDF Triple Language, is the syntax that is used for our database. Turtle is a syntax for RDF that expresses data in 3-tuples, or triples, which is a statement that relates three different objects to each other. These triples represent the general subject-predicate-object structure of RDF. In turtle, subjects are declared in an unindented line of text. Underneath each subject is an indented list that contains pairs of predicate and objects that are associated with each subject.[7]

Figure 3, on the next page, is a snippet of our database, showing the response to a survey on Ebola taken in Liberia. The snippet highlights this Turtle syntax as the first line identifies a subject in the database. Underneath the subject, there are indented lines, each signifying a predicate and an object for this subject. For example, look at the following line:

<http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistcal-dimension/age> “41”

This line connects the predicate “age” and object “41” to its subject, telling us that he is 41 years old. Furthermore, these attributes of the subject are inserted into the graph that the database maintains, connecting the subject with his age. In this fashion, all of the turtle triples get linked and form a graph that the database can query to extract connected information.



```
<http://ebola.vbi.vt.edu/dataset/data-for-ebola-recovery/observation-index/173> a qb:Observation ;
    qb:dataSet <http://ebola.vbi.vt.edu/dataset/data-for-ebola-recovery> ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/country> <http://ebola.vbi.vt.edu/data-for-ebola-recovery/liberia> ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/subregion> "Monrovia" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/esdate> "12/4/2014" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/estime> "15:20" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/towncode> "2008" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/ward> "Z1000-Congo Town" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/respid> "2008181" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/gender> "1-Male" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/age> "41" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/hhsize> "6" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/educ> "8-Completed university" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/religion> "1-Christian" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/leavemonr2w> "0-No" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/returnmonr2w> "0-No" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/ehowget> "3-Bodily fluids of infected pe;5-Touching dead bodies" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/esignsymptom> "1-Vomiting;3-Diarrhea" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/eincubation> "2" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/ehotline> "1-Yes" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/spreadair> "N" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/spreadfluids> "Y" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/spreadearly> "Y" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/saltcure> "N" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/notreatmentj26> "1-Just once or twice" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/under5> "3" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/childsick> "N" ;
    <http://ebola.vbi.vt.edu/def/data-for-ebola-recovery/statistical-dimension/hhmemsick> "N" .
```

*Figure 3: Example of Ebola LOD in Turtle syntax*

### C. Python

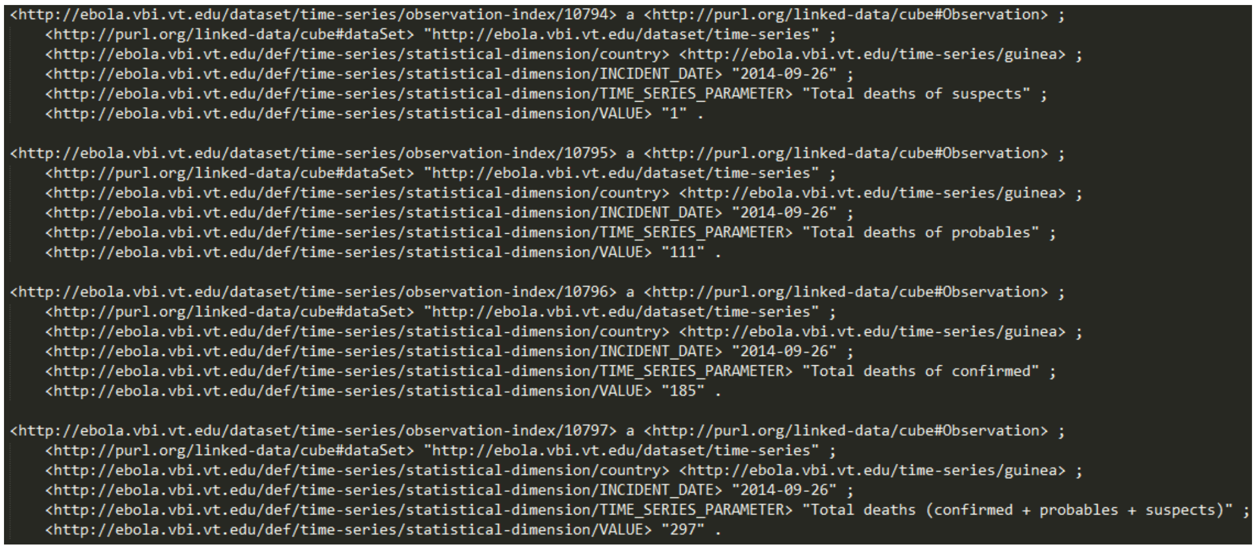
The programming language that most fit this specific problem is Python[8] . Python makes it easy to scrape information from files, process data from varying data sources, and compare pieces of data. It also features an extensive collection of libraries, such as RDFLib[9], a library which is able to parse through various RDF syntaxes. This will help us extract data from the Ebola database. Another relevant library for our script is BeautifulSoup[10], which specializes in scraping data from websites.

Python is also good for standardizing data so that data from the database and external sources can be compared correctly. We will utilize python's dictionary data structure[11], which is in essence a Hashtable data structure, to hold the values parsed and standardized from our validation sources. This will allow is to have efficient (constant time) insertion and lookup into our validated data. Once we lookup the value, we will utilize the RDFLib library functionality to pick up corresponding data values from the RDF database and compare them.

# 8. Implementation

## 8.1. System Organization

Our program has three major components to it. The first component is the Turtle database parser. The goal of this component is to parse the data fields in the Ebola database that was given to us in the .ttl file format, and load them into memory so that they can be validated. This processes is going to be done by the python library RDFLib, which leverages the structure of .ttl files to extract the data values contained in them. The RDFLib library parses through the .ttl file and returns a graph object that can be searched through using RDFLib's own API. This graph object is about 33MB, which can easily be stored in any machine with over the required 4GB of ram.  Once the graph is stored and ready to use, we search through it.



```
<http://ebola.vbi.vt.edu/dataset/time-series/observation-index/10794> a <http://purl.org/linked-data/cube#Observation> ;
    <http://purl.org/linked-data/cube#dataSet> "http://ebola.vbi.vt.edu/dataset/time-series" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/country> <http://ebola.vbi.vt.edu/time-series/guinea> ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE> "2014-09-26" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/TIME_SERIES_PARAMETER> "Total deaths of suspects" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/VALUE> "1" .

<http://ebola.vbi.vt.edu/dataset/time-series/observation-index/10795> a <http://purl.org/linked-data/cube#Observation> ;
    <http://purl.org/linked-data/cube#dataSet> "http://ebola.vbi.vt.edu/dataset/time-series" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/country> <http://ebola.vbi.vt.edu/time-series/guinea> ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE> "2014-09-26" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/TIME_SERIES_PARAMETER> "Total deaths of probables" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/VALUE> "111" .

<http://ebola.vbi.vt.edu/dataset/time-series/observation-index/10796> a <http://purl.org/linked-data/cube#Observation> ;
    <http://purl.org/linked-data/cube#dataSet> "http://ebola.vbi.vt.edu/dataset/time-series" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/country> <http://ebola.vbi.vt.edu/time-series/guinea> ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE> "2014-09-26" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/TIME_SERIES_PARAMETER> "Total deaths of confirmed" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/VALUE> "185" .

<http://ebola.vbi.vt.edu/dataset/time-series/observation-index/10797> a <http://purl.org/linked-data/cube#Observation> ;
    <http://purl.org/linked-data/cube#dataSet> "http://ebola.vbi.vt.edu/dataset/time-series" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/country> <http://ebola.vbi.vt.edu/time-series/guinea> ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE> "2014-09-26" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/TIME_SERIES_PARAMETER> "Total deaths (confirmed + probables + suspects)" ;
    <http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/VALUE> "297" .
```

*Figure 4: Example of Ebola LOD subject-predicate-object form*

In Figure 4, the subject-predicate-object form can be seen clearly. The subject is the first line of each section, indicated by the lack of indent. Each subject represents a data point in the Ebola database. The predicate is at the beginning of each line after the subject and is always indented. The predicate associates values to its associated subject. In the example above, you can see that a common predicate is "INCIDENT_DATE," associates a date to each subject (data point). The object is at the end of each line containing a predicate. It holds the value associated with a given predicate. In the figure, you can see that "2014-09-2016" is an object next to the date predicate. The combination of this object, its predicate, and its subject describes the data point, saying that it has the date 2014-09-2016 associated with it.

When we load an RDF file in this format, RDFLib produces a graph object, and we are able to search through all subject, predicates, and objects in this graph and filter out the ones we want. Since the subjects in this database do not provide much useful information (a subject by itself may be "Data point 1", for example), we are only focused on searching using predicates and objects. One such search may look for a specific country, incident date, and time-series parameter to help us find the subject associated with that data point. The search parameters of the search are determined by what value the user wants to validate. If a user wants to validate a certain data column, it will not be considered in the search. For example, if the user wanted to validate the number of Ebola cases on a given date as 1000, the number 1000 wouldn't be used to search for that data point in the Ebola database. This is because this value could be different, and our goal is to determine if it is different.  However, we would use ever other piece of information about that data point as a composite key to search for it in the database. In this example we know that we are looking for the number of cases on a certain date. Here, we would use the date and the fact that we are searching for cases as keys to search the database on. Once we find a data point that has the date we are looking and is about the number of cases, we can look at its value, and check to see if it is equal to 1000.

The next major component is the dataset parser. This component is responsible for scraping data from the various validation sources that we have identified and loading them into memory so that they can be compared to the Turtle database. For our prototype and our final user-friendly program, we only accept comma separated value files, or CSV files. Using the Pandas library detailed earlier in this report, we are able to parse these CSV files line by line to extract and compare data. To do this, extract each row of the entered CSV, and treat each row as a data point in the Ebola database. This is because a row in the database will contain all the information that describes a subject in the RDF database. The column headings in the CSV represent the predicates of the RDF database, and the values in the columns themselves represent objects in the RDF database. Therefore, we can have each row of a csv parsed as a data point of the RDF database.  Since CSV files are the standard format for storing Ebola data, this decision should not limit the number of sources too greatly.

The final component of our system is the validation engine itself. This is the component of the program that will compare values from the Turtle database to the values from the data sources. The validation engine contains a mapping from the parsed data sources to the objects they validate, which will be created through user input into our generic Ebola script. This mapping will allow comparison of the correct dataset values to corresponding database values. Before datasets are compared, they will be pre-processed so that values being compared will have the same format. For example, if a data field from a dataset contains the value “3,230”, and it is being compared to “3230” (without the comma) in the Ebola database, we convert it to an integer so that we can run a comparison operation between both numbers.

Our script then outputs the results in CSV format. It displays the rows if the input CSV, with two additional column. The first additional column appear next to the column that holds the input source’s value that was being validated, and holds the value of the corresponding field in the database. The second column appears to the right of this column, and holds either the value “TRUE” or “FALSE”, depending on if the validation source and the RDF database had data fields that matched exactly.

## 8.2. Major Tasks

### 8.2.1. Locate Sources

The first major task for this project is to locate sources for validation. The process of locating sources is an ongoing process. Although we have found new data sources for most of the existing data already, it is possible that those sources may not be viable or the dates of that data does not match up with the dates in the database. This is the main reason that we created a generic script. A generic script allows users to continue to add sources, even as new sources surface, or the data in the database changes.

We have located 27 different data sources, located by searching through Google, that are being tested for validation against 22 existing datasets. Table 2, in Appendix B, shows the 22 existing datasets and the 27 possible data sources that can be used for validation. As seen in the table, there are three datasets that we have been unable to find alternate sources for and there may be new datasets introduced in later iterations of the Ebola database. The users of our script will be responsible for locating these validation sources and use our program to validate the data. This will allow the data to be constantly updated and validated, even as the database grows larger.

### 8.2.2. Generic Script

Our ultimate task for this project was to write a generic script that can take any input CSV and match its data points up with the data points in the Ebola database, and return the compassion of them. The script contains a graphical user interface so that it can be used easily by users who do not have little knowledge with scripting or command line. The GUI requests inputs from the user until it has enough information and is able to run the underlying Ebola script. The underlying code is what does the validation for the user and provides an output file.

### 8.2.3. Parsing Existing Database

A major task during the running of the script is to parse the existing data in the database and organize it to be easily accessed. As previously mentioned, we are using dictionaries to store the existing datasets since dictionaries can be quickly accessed and easily searched through Python. Using RDFLib, we will be able to parse through the Turtle file and create an RDFLib Graph Object. This is an object in python, that, when queried, can return the subjects, predicate, and objects in the RDF database.

### 8.2.4. Scraping Sources for New Data

Another major task in the script is to scrape the sources that we located for relevant data. The data from sources will be taken from CSV files and stored and organized inside our program. After all data is stored in memory, the data is ready to be used for comparison.

### 8.2.5. Compare and Validate Data

Now that both the existing datasets and the new data sources have been stored in memory, the data can be compared and validated. The process of validation will involve homogenizing all the different sets of data and then comparing them. The homogenization process is necessary since much of the data is stored in different formats. For instance, there is data that is stored as monthly reports and other data that is stored as weekly or daily reports. Some data stores the dates in the format "2014-09-07" while others store it as "9/7/2014." The validation process needs to be able to compare these two sets of data and therefore needs to make the data comparable first. The Ebola database is then searched to find the data point in the database that corresponds to the data point in the validation source that we wish to validate. We will repeat this process for all the data points in the validation source, thus validating the entire CSV.

After the data is compared, the results are ready to be interpreted. Since giving our client a listing of numbers and comparisons would not be useful to him, our last step is to report the results in a useful manner.

### 8.2.6. Report Results

In order to report the results, we need to organize the data into a file that can be easily distributed and read by the client. We decided on reporting the results in a CSV file since CSV files can be easily read on a number of devices and are a very good way of organizing the data we are delivering. The CSV will contain information such as the values from the validation source, the values from the database, and whether or not those values were close. It will also tell the user the specific parameters that the value is related to. This makes it easy for the user to generate a graph of the results for easier viewing or just view the data directly to see specific data points.

## 8.3. Schedule

See Table 1 in Appendix A for the complete Implementation Schedule. This schedule was followed partially, but some of the deadlines were shifted to meet project requirements.

# 9. Prototype

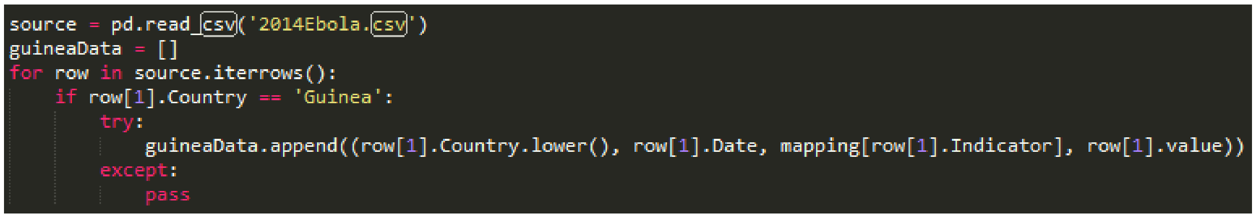
## 9.1. Program Structure

Our prototype is structured into three distinct parts: the database, the validation source, and comparison of the two sets of data. For the prototype, we focused on a specific set of data for our script, the time-series data for the country Guinea. This allowed us to have a smaller portion of the data to test the prototype over, while still allowing the script to be easily extended to a much larger portion of the data.

The data sources for this script are manually found and matched up to a specific dataset in the database. Specifically, we looked at the time series data, comparing data points describing the number of cases, deaths, and Ebola related values on given dates between August 2014 and January 2015. Our prototype successfully returned the comparison of these data points between our validation sources and the RDF database.

## 9.2. Functionality

The step of the script deals with parsing the database. The database is parsed through the RDFLib library into a RDFLib Graph object, which is an object the library uses for managing the Ebola database in memory and searching thought the database. After the database is parsed, the source data is parsed. To do this, our script parses through the source CSV file and stores the values in a Python list, where each element of the list contains 4 data values, corresponding to the four columns in our source file. This functionality is demonstrated in Figure 5.

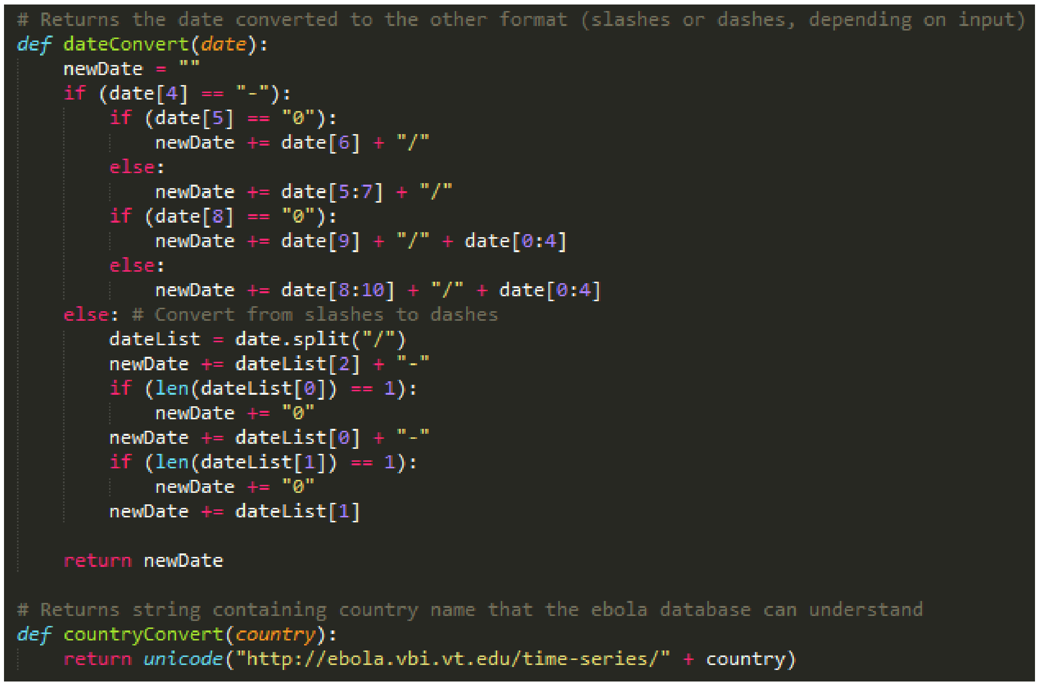


```
source = pd.read_csv('2014Ebola.csv')
guineaData = []
for row in source.iterrows():
    if row[1].Country == 'Guinea':
        try:
            guineaData.append((row[1].Country.lower(), row[1].Date, mapping[row[1].Indicator], row[1].value))
        except:
            pass
```

*Figure 5: Parsing of CSV file into a Python list*

After all data had been parsed, the prototype needed to homogenize the data to a common format. To do this, we decided to have the source data match the database format. Homogenization for the Guinea prototype involved a wide range of different techniques. One such technique we used was getting fields to match between the database and the source. For instance, the validation data had the country stored as "Guinea" but the database stores the country name at the end of a URL, such as "http://ebola.vbi.vt.edu/time-series/guinea." To match the 2 data points up, we converted the first string "Guinea" to the URL string used in the database.

The dates had similar homogenization issues. In the validation data, the dates were usually stored in dash format (i.e., 2014-09-08) as opposed to slash format (9/8/2014). Unfortunately, the database did not adhere to a single format. Some data is formatted in dash format while other data is in slash format. Because of this, we checked for both formats in the database and used the one that existed, assuming the date existed at all. These two examples of homogenization can be seen in simple functions in Figure 6.



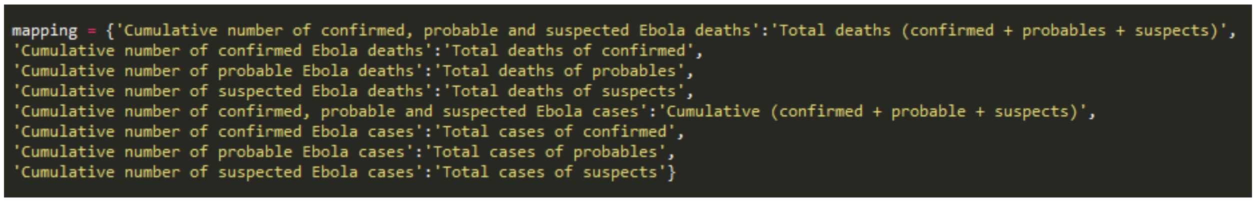
```python
# Returns the date converted to the other format (slashes or dashes, depending on input)
def dateConvert(date):
    newDate = ""
    if (date[4] == "-"):
        if (date[5] == "0"):
            newDate += date[6] + "/"
        else:
            newDate += date[5:7] + "/"
        if (date[8] == "0"):
            newDate += date[9] + "/" + date[0:4]
        else:
            newDate += date[8:10] + "/" + date[0:4]
    else: # Convert from slashes to dashes
        dateList = date.split("/")
        newDate += dateList[2] + "-"
        if (len(dateList[0]) == 1):
            newDate += "0"
        newDate += dateList[0] + "-"
        if (len(dateList[1]) == 1):
            newDate += "0"
        newDate += dateList[1]

    return newDate

# Returns string containing country name that the ebola database can understand
def countryConvert(country):
    return unicode("http://ebola.vbi.vt.edu/time-series/" + country)
```

Figure 6: Date and country homogenization

The last type of homogenization we implemented was for the parameter names. For instance, a data source may list a data as “Cumulative number of confirmed, probable, and suspected Ebola cases” while our database may list it as “Cumulative (confirmed + probable + suspects).” To solve this, we created a mapping that mapped the strings from source to destination. The mapping is displayed in Figure 7.



```python
mapping = {'Cumulative number of confirmed, probable and suspected Ebola deaths':'Total deaths (confirmed + probables + suspects)',
'Cumulative number of confirmed Ebola deaths':'Total deaths of confirmed',
'Cumulative number of probable Ebola deaths':'Total deaths of probables',
'Cumulative number of suspected Ebola deaths':'Total deaths of suspects',
'Cumulative number of confirmed, probable and suspected Ebola cases':'Cumulative (confirmed + probable + suspects)',
'Cumulative number of confirmed Ebola cases':'Total cases of confirmed',
'Cumulative number of probable Ebola cases':'Total cases of probables',
'Cumulative number of suspected Ebola cases':'Total cases of suspects'}
```

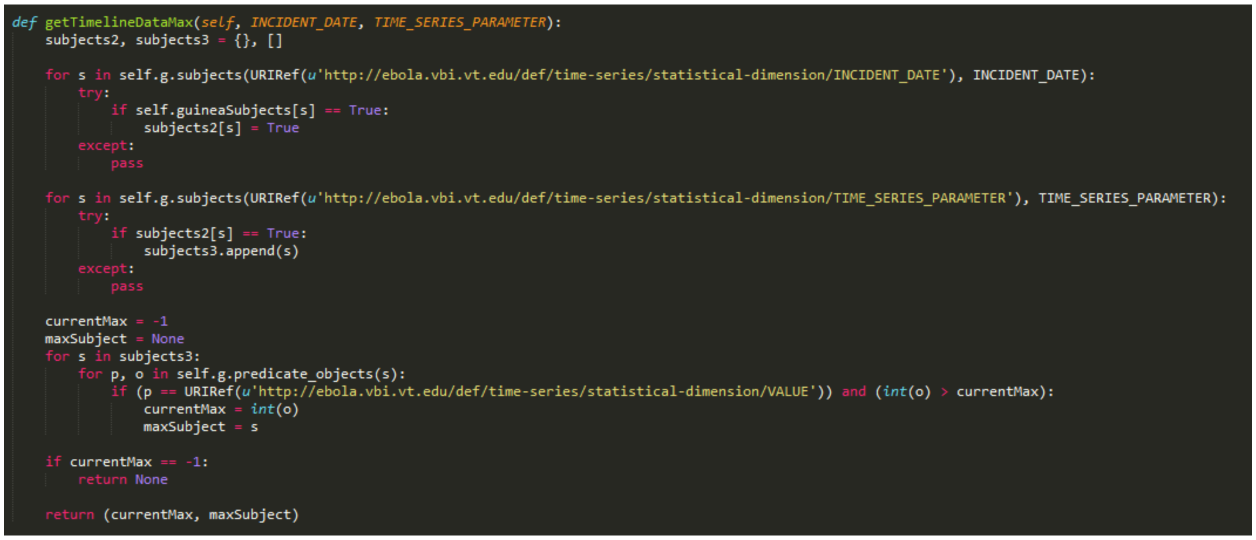
Figure 7: Mapping for parameter names

One other type of complex homogenization involves matching up the dates of the source data and the dates in the database. For a large amount of the data, the dates we have to verify are off by one or two days from the database. For instance the source data may have dates 9/6/2014, 9/9/2014, and 9/14/2014 while the database may have dates

9/7/2014, 9/10/2014, and 9/12/2014. We addressed this issue by comparing a date in the validation source only if it existed in the Ebola database.

The final part of the prototype compares the external data source to the database and outputs the results to the console. The first part of this comparison is to get the correct data point from our database.

The database is organized in such a way that there are multiple values for the same date, country, and parameter. This makes it difficult when comparing external data which usually has only one value for the same date, country, and parameter pairing. We found out that the data in the database kept records for individual cities inside Guinea, even though it did not note this fact anywhere inside the database. Since the source data kept records on Guinea national data, we decided to test those specific values. To find the value corresponding to Guinea national data (as opposed to a city inside Guinea), we created a function in the prototype that found the max value for a given search. The max value found is always the value of the country, as opposed to a city within the country. This is because the database had values for cities as well as the entire country when we searched for the value based on the information within our validation source. The function for this is depicted on the next page in Figure 8.



```python
def getTimelineDataMax(self, INCIDENT_DATE, TIME_SERIES_PARAMETER):
    subjects2, subjects3 = {}, []

    for s in self.g.subjects(URIRef(u'http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/INCIDENT_DATE'), INCIDENT_DATE):
        try:
            if self.guineaSubjects[s] == True:
                subjects2[s] = True
        except:
            pass

    for s in self.g.subjects(URIRef(u'http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/TIME_SERIES_PARAMETER'), TIME_SERIES_PARAMETER):
        try:
            if subjects2[s] == True:
                subjects3.append(s)
        except:
            pass

    currentMax = -1
    maxSubject = None
    for s in subjects3:
        for p, o in self.g.predicate_objects(s):
            if (p == URIRef(u'http://ebola.vbi.vt.edu/def/time-series/statistical-dimension/VALUE')) and (int(o) > currentMax):
                currentMax = int(o)
                maxSubject = s

    if currentMax == -1:
        return None

    return (currentMax, maxSubject)
```

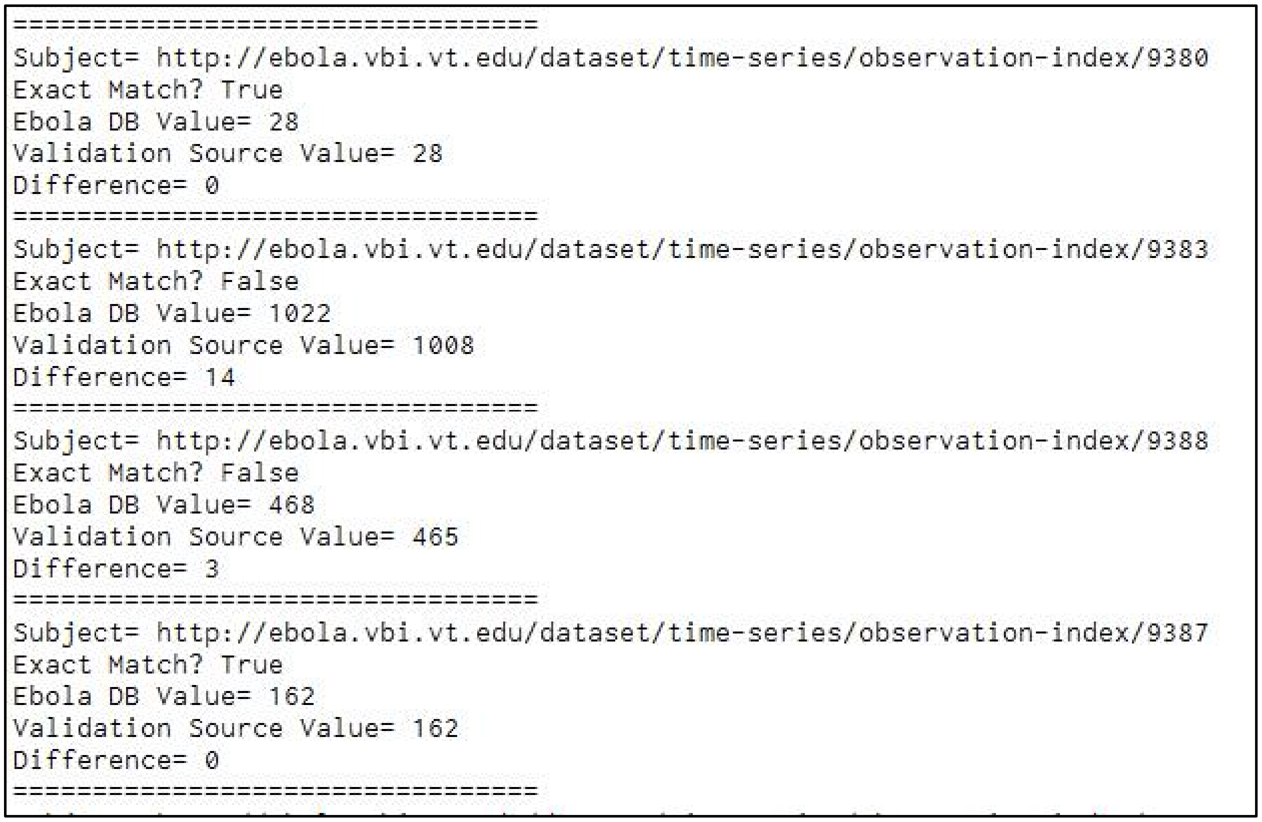
*Figure 8: Code for finding max value among a list of data values*

After the completion of this prototype, we were given an updated database file that contained sub-national data, including the values for each individual city. This is a major reason that a generic script is so valuable. The values in the database are being updated even throughout the timeline of this project, meaning that creating a program that can be applied to generic data is a valuable tool.

After figuring out how to find the overall country data value, we looped through the CSV file to get the date, country, and parameter information. We used this information to find the specific piece of matching data in the database. Searching the database involved finding the RDF subject that corresponded to a row of data in our source CSV file. We were able to search the Ebola database by searching for RDF predicates that corresponded to the column headings of our CSV and RDF objects corresponding the value of the cells underneath the corresponding column heading in the database. We retrieved the max data value for each search so that the data corresponded to the country Guinea instead of a city inside it, since searching for these values resulted in getting the data points for both the country and cities within that country, and the maximum value represented the value at the country level. An array of values from the database is then returned and compared to the matching values in the CSV file. At this point, the only thing left is to format and output the results.

## 9.3. Results

The prototype outputs the results from comparing the two sets of data. For every data point in the Ebola database that had a matching data point in the validation source (based on the country, data, and parameter) the prototype printed the value corresponding to that data point in both the RDF database and the validation source, and printed a Boolean, either True or False, if the two values were exactly the same. The results are depicted in Figure 9.



```
==================================
Subject= http://ebola.vbi.vt.edu/dataset/time-series/observation-index/9380
Exact Match? True
Ebola DB Value= 28
Validation Source Value= 28
Difference= 0
==================================
Subject= http://ebola.vbi.vt.edu/dataset/time-series/observation-index/9383
Exact Match? False
Ebola DB Value= 1022
Validation Source Value= 1008
Difference= 14
==================================
Subject= http://ebola.vbi.vt.edu/dataset/time-series/observation-index/9388
Exact Match? False
Ebola DB Value= 468
Validation Source Value= 465
Difference= 3
==================================
Subject= http://ebola.vbi.vt.edu/dataset/time-series/observation-index/9387
Exact Match? True
Ebola DB Value= 162
Validation Source Value= 162
Difference= 0
==================================
```

*Figure 9: Output from Guinea prototype*

It is important to note that although the “Subject” line gave us no useful information pertaining to the data points that was compared, it does tell us where in the database the data can be found and is therefore used in a similar fashion to an ID.

# 10. Refinement

## 10.1. Efficiency Modifications

Obtaining results from the first version of the prototype was a very time consuming process, so the script was modified to give results faster. Originally, running the script to validate time series data on Guinea was a process that repeated searches over the entire RDF database. We modified the script to save the result of certain searches in directories, which improved the performance time of the script, but required us to store more data values in memory. This enhancement made the script run in about 4 to 5 minutes instead of the previous version, which took approximately 1 hour to run, about an 11x speed improvement. Specifically, for this improvement, we searched for all the time series data values in the RDF database that have the country Guinea as the object of the country predicate, and saved them in a Python dictionary. This eliminated the process of finding time series data values about Guinea every time we wanted to compare a value from our data source with Guinea data.

Another modification we made to speed up the process of obtaining results is by serializing and saving selected intermediate data and data structures that are produced as our script executes. We did this by leveraging Pickle[13], a library built into Python that can serialize the value of any data used by a Python script and save it in disk, encoded in hexadecimal. This file can then be loaded into any Python variable when its content needs to be read and stored in memory. We used Pickle in our prototype to save the list of tuples that comes from parsing the CSV data source we used for validation. This allowed us to run the part of the script that parses and extracts data from the CSV only once, and any subsequent time we need this data, we can load the Pickled version of the data instead of having to recompute and reparse the data. Here, we are eliminating computation time of the overall script by saving and loading intermediate data values used by the script from disk, instead of having code that loads this data into memory.

## 10.2. Extending Prototype to General Script

Our prototype was modified to create the general validation script, which takes any CSV as a validation source, and is able to compare its values to the corresponding values in the Ebola database. To do this, we first had to change our code which parses CSVs to parse any CSV in general, rather than the CSVs we used as validation sources in the prototype. To do this we stored the column headings of the input CSV in a list, and then looped through each row of the CSV, storing the values in each column of the CSV. Since a row in a CSV represents a data point that needs to be compared, we stored the values in each row into its own list, and mapped this list to the list which holds the column headings. This way we were able to load any CSV row by row, storing the values in each row along with what the value represents.

We then need to modify how the data points are searched for and compared. To do this, we needed to ask the user for some input. We modified our script to ask the user to map CSV column headings to RDF predicates. For example, an input CSV could have a column entitled “Sub Region” but in the RDF database, this corresponds to an RDF predicate that is entitled “County.” Here, we require the user to user to tell the script that the Sub Region column in their validation source corresponds to the County predicate in the RDF database. Since we have access to the predicates in the RDF Database, and our script now reads the column titles for any CSV, we were able to provide the user all the options for CSV column headings, and all the options for RDF predicates, so that they have an easy time matching up these values.

We also asked the user to create an Object mapping, which is based on the same principle as the predicate mapping. The values of the CSV file could potentially be different than the objects in the RDF database. For example, one value in the CSV file could be “Under Construction,” referring to the state of an Ebola care center. In the database, the corresponding RDF object could be “Construction Ongoing.” In this case, the user would tell the script to map the “Under Construction” in the CSV to the “Construction Ongoing” object in the database. This object mapping is optional, and will not be required in most cases, because CSV values will mainly be numbers (such as 1000, under a column called “Cases”), so the user will often not be required to enter this. Since the script parses all the unique values in any column of any CSV, and we have all the objects for the predicate mapping the user has already created, we can again give the user all options for RDF objects and CSV values, to make it easy to match.

To make the search functionality generic, we ran the search on the object and predicates that the user mapped in the previous step of the script, attempting to find the subject that corresponds to the mapped object and predicates. For instance, if a row in the CSV file says “Guinea, Cases, 45” corresponding the to the column headings “Country, Parameter, and Value”, and we want to validate the “45”, the search will find RDF subjects based on two pairs of predicate and object. The pairs will be (Country, Guinea) and (Parameter, Cases), mapped based on the object and predicate mappings that were entered. These pairs form a composite key to search the database on, which give the RDF subject we are looking for. We get the missing pair from the RDF subject, which will be (Value, 45) if the database holds the same number as the CSV for the value. One important distinction here between the generic script and the prototype is that the prototype only considered three parameters (country, date, and parameter) as they key to the search. With the generic script, the number of parameters to consider in each will be equal to the number of predicates in the source we are validating minus one. This is because each predicate is part of the key used to search for the subject, except the predicate we are validating. If we want to validate all predicates, then we just simply have to run the search once for every predicate, changing the predicate that is being validated every time.

In the generic script, we decided to change the output format to a CSV file. This is because the data we are looking at originally came from a CSV, and the result of the comparison can be well structured in a CSV, increasing its readability. Just like the prototype, the RDF database value and the input source value is generated in our CSV, along with the Boolean that says “True” or “False” depending on if the two numbers were an exact match.

## 10.3. Graphical User Interface

For our general script, we decided that a GUI interface would be important since the goal of this script was for any user with knowledge of the Ebola database to be able to use it. Due to the generic nature of our script, combined with the large amount of data displayed from the database, a GUI was the best way to display information to the user and also receive necessary input from the user to run the generic script. To best display the information, we separated the GUI into three different pages, each page with its own specific function.

### 10.3.1. Content of Pages

The first page of the GUI has 4 main elements to it: importing the database, importing the CSV file, choosing the dataset, and the submit button. When the user hits the first button to import the database, it brings up a file dialog which takes the user's file choice and extracts all the datasets that are found in the database. From there, it populates the third element, the drop down which prompts the user to choose a dataset, with all datasets found in the database. The user also import a CSV file which is chosen in the same manner that database files are chosen (via file dialog). From there, the user knows every dataset in the database and the validation file they are using. This will lead the user to open the drop down menu and choose the dataset that matches up with the CSV data. At this point, the program has all necessary pieces of information so the user will hit the submit button, taking the program to the second page.

The second page of the program displays a list of all unique predicates found in the specified dataset. They are displayed one row after another. On the same row of each predicate are three other elements: a drop down menu and two checkboxes. The drop down menu is used for the user to map CSV column names to the corresponding predicate from the database. These are pre-populated if the program finds names that are "close enough." Specifically what this means is that the script assigns each column name and predicate a value saying how close the two strings are. We find the closeness of two strings through a Python library called difflib.[14] This library uses algorithms that look at the order of a word and the letters in each word and assigns each pair of words a value between 0 and 1 based on their closeness (where 1 is a perfect match and 0 is extremely different). If the strings are "close enough" (assigned a value of 0.6 or greater) the script will set the mapping by default. This mapping can be changed manually by the user as well.

To the right of the drop down on each row are two checkboxes. The left checkbox asks the user if the predicate is a point of validation. If the user checks this box, the information will be sent to the script upon submission of the page. The final checkbox lies on the rightmost part of the GUI. This checkbox asks the user if the predicate requires object mapping. Like the other checkbox, the user's choice is recorded and sent to the script to be used for the final algorithm.

The final page of the GUI has the function of mapping objects. All predicates that the user specified will be printed to the screen, and under each predicate, a list of all unique objects will be listed, one row at a time. To the right of each object is a drop down menu. Like the last page, these drop down menus are pre-loaded with the script's "best guess" for what the mapping will be. The user can once again manually change each of these to fix any wrong mappings. On this page is a submit button as well. Once the user hits this button, the script will begin the algorithm based on the specific inputs of the user.

#### 10.3.2. Underlying Structure

The overall layout of each page is set up in a grid format. Each element on the page is placed inside a certain cell of the grid, giving the GUI a more structured look to it. The GUI also has a scrollbar on the right side which is useful when there are more predicates to display to the user than the screen can fit. The structure of the GUI itself is structured around this scrollbar.

To make the scrollbar work correctly, it was necessary to create an inner frame inside of a canvas inside of an outer frame. The inner frame is what holds the grid structure which displays all the GUI elements to the user. The canvas is only there so that the inner frame can be scrolled through when needed. Frames cannot be scrolled in Tkinter, so putting a frame inside a scrollable canvas was the best available option. Outside of the canvas is the outer frame which holds the canvas and the scrollbar. The scrollbar controls the inner frame, but is displayed using the outer frame so that it does not need to adhere to the grid structure (which every other element adheres to). Although these design decisions are invisible to the user, they are important to understand when developing the GUI.

# 11. Testing

## 11.1. Unit Testing

Unit testing refers to the testing performed on specific functionality in our script. Each function we made was individually tested to ensure that it ran properly before being deployed to the final script. In short, each function we made was tested to ensure it ran properly before being deployed.

One such function we tested was involved in our prototype, called "getTimelineDataMax". This function only applies to the time-series data. It finds the maximum value in a set of data with the same exact parameters. This is necessary because the database we were given has no field for regions inside a country, so there are multiple, inequivalent values for the same country/data/parameter combination. In order to test this function, we checked that the value we grabbed was indeed the largest value out of a set of values. We applied this function to the database and checked that it worked for multiple sets of data. Once we found that the value was consistently correct, we concluded that the function worked.

Another function from our prototype that we tested was called “dateConvert”. This function is part of our code that homogenizes the data and is part of the homogenization testing talked about more in-depth in the next section. Before homogenization testing, though, we tested the method directly by feeding it a date with one of the two possible formats and then checking that the data correctly converted to the other. Our tests targeted the possible edge cases that relate to the dateConvert function. For example, there may be only one or two digits for the month and day in the date. We checked all possible combinations of these formats and, after seeing that our function worked with each edge case, we concluded that the function worked.

Another function we tested is called “loopThroughCSV”. This function is important since it is what traverses through the CSV validation files and parses the data to be compared to the database data. To test this function, we gave it a couple different CSVs and verified that it correctly parsed the data. We gave it simple CSV files at first just to see that it was transferring and storing the data correctly. We gave it more complex CSV files until we finally gave it the validation files in CSV format. Once we verified that the function worked with all of these files to store the data correctly, we concluded that the function worked.

Finally, we tested the “printResults” function. This function prints the data from both validation source and database and puts them together in a CSV file with other information such as the difference between the values and the source of the validation. To test this, we checked that the number of entries in the CSV file is what we expected. If the data has 50 pieces of information that can be verified, then the CSV file should have 50 lines. After verifying that the number of lines was correct, we concluded that the function worked correctly.

In the generic script we recycled most of these function and tested them using the same methodology. The one exception to this is the getTimelineMax function, which was replaced with our generic search function.

To test our generic search function, we entered lists of rows from a CSV file, and made sure that the subject that was returned from the search corresponded to the data set that we intended to find in the RDF database. So if we entered a row of a CSV file that represented the number of cases on 12/4/2014, we checked to make sure the RDF subject returned by the search corresponded to the same data point, by looking through the database and finding that subject, and making sure it had the same predicates and objects as the column headers and values in the CSV row we entered. We then scaled this test to handle entire CSV files, rather than individual rows.

Another function we needed to test in the generic script was the readCSV function. This is the function that allows us to read any CSV. To test this, we ran the function on many CSVs, and ensured that it gave use a list of the column headers in the CSV and a list for each row in the CSV, holding the values of columns on that row. We then checked to see if the values in those lists correspond the values that were in the CSVs that we entered to the function.

At this point, all functions had been individually tested. However, we still needed to test our results to ensure the integrity of the data that we deliver to our client. This testing is detailed in the systems integration testing section.

## 9.2 Systems Integration Testing

Systems integration testing refers to the testing done on the results and output of our script. These include accuracy testing, homogenization testing, and flexibility testing. Accuracy testing is testing the accuracy of the data in our results, homogenization testing is testing that the data on both sides (database and validation side) are normalized to the same format so they can be compared, and flexibility testing is the testing of our code with multiple files.

Accuracy testing, specifically, is the comparison of the values in our script's output to the values in the database and validation files. To test this, we took a random sample of values in our output and manually compared them to the source files. For example, if you look back at Figure 8, the results had two values, 1022 and 1008. To test that these values were accurate, we check that these two values, 1022 and 1008, matched the data in the database and validation file, respectively. After about 10 values with different fields (i.e., different country or parameter) were compared and each one produced the same value as the source files, we concluded that the results were accurate.

Homogenization testing involves checking that the varying field names in the source file are mapped to the correct field in the database. The database itself has varying field names among different countries or different datasets, meaning that the cumulative number of deaths due to Ebola may go under differing names such as "Deaths", "cum deaths", "Cumulative deaths (suspected, probable, and confirmed)" as well as others. In our testing, we checked that we correctly mapped the names in the source files to the aforementioned names in the database. To do this, we simply checked that the correct values were mapped into the correct field names in our output. This is similar to accuracy testing since we were checking the correctness of values, but in this case, we checked that values were placed in the correct fields (i.e., country, date) instead of just checking that values matched the source files. We also used the random sampling testing method for this type of testing. After checking about 10 values and ensuring that they matched up, we concluded that the script correctly homogenized the data, since the values are not there if the data had not been properly homogenized.

Finally, the last part of systems integration testing was the flexibility testing. This testing involves testing our script on multiple files. This was done just by finding multiple sources for the data and then running them through our script. If each file produced results with correct values, we concluded that the script was flexible enough to accommodate that file.

We preformed systems integration testing for both the prototype and the generic script. We ran the script on multiple data sources, the output is complete and accurate, and the code works for all the circumstances we tested.

## 12. Inventory of Files

The zip folder containing our project contains the following files:

- **ebola.py** – This is the main ebola validation script.
- **questionMark.png** – This is an image used for tooltips in the GUI.
- **EbolaValidatorReport.docx** – The Word format of this report.
- **EbolaValidatorReport.pdf** – The PDF format of this report.
- **EbolaValidatorPresentation.pdf** – The PDF format of the presentation given on this project.
- **EbolaValidatorPresentation.pptx** – The Powerpoint format of the presentation given on this project.
- **readme.txt** – Readme file containing information for the user

## 13. Future Work

Although the majority of the functionality is present in the Ebola script, it can still be refined and improved a good amount. The first major piece of work to be done is to add mappings for the date. In the database, dates are formatted in several different forms and in a CSV, a date can have any number of formatting. Right now, we have a function in the script called “dateConvert” which is commented out. It requires user input to tell the program how the CSV dates are formatted and it will convert to another date format. This function did not make the final build of the program since it was difficult to implement without causing the user even more work. A way to homogenize the dates in the CSV file with the format in the database is the first essential piece of future work.

The second piece of functionality to add is a bit more complex. Right now, the script uses a library called difflib to find the closeness of strings. This functionality is found in the function attemptToMatch. This function is vital to the program as it provides the user with the suggestions for predicate mappings on the second page of the GUI and object mappings on the third page of the GUI. Our client, however, wants this third page of the GUI entirely gone since it is too much work for the user. Therefore, this function, attemptToMatch, will be trying to map objects with no user to correct it if it is wrong. The function right now is a bit too lenient and only looks for matches in strings, not in meaning. It will match words like "date" and "updated" which obviously have different meanings. This function can be vastly improved to search for meanings between words as opposed to only string matching. Improving this function will greatly improve the overall script.

Our script right now is made to be as generic as possible, but there are many improvements that can be made to improve this as well to make it work with verifying data in any database. The main portion that needs to change to make it more generic is to change how the script looks for datasets inside the database. Right now, it looks for a specific string that is specific to this Ebola database. This string can be found near the top of the code in quotes as "http://ebola.vbi.vt.edu/dataset/." If this is changed so that the script has a more verbose method of finding datasets, it will be generic for any RDF database.

One more step to making this script more generic would be to add more compatibility for database types and validation types. The script only works with database formats that are in RDF format right now and only works with validation files in CSV format. Adding compatibility with a broader range of database file types and validation file types will further generalize the script.

## 14. Acknowledgments

We would like to thank S.M. Shamimul Hasan for his guidance on this project and his help on understanding the Ebola database, Dr. Edward Fox for his help in connecting us with our client and his feedback on our project, and Yilong Jin for his help with crafting this report.

## 15. Appendix A – Implementation Schedule

| **Date** | **Task** |
|---|---|
| 2/15 | Study current version of Ebola LOD |
| 2/29 | Identify data sources for validation |
| 3/15 | Implement version 1 of prototype, which runs with at least 3 validation sources |
| 3/31 | Implement version 2 of prototype, which validates at least half of the 22 datasets |
| 4/15 | Implement final prototype, validating all datasets that contain at least one usable data source |
| 4 /20 | Complete Testing |
| 4/25 | Finalize Script |
| 4/30 | Project Report |
| 5/6 | Final Presentation |

*Table 1: Project due dates*

This appendix outlines our project schedule.

## 16. Appendix B – Data Validation Sources

| **Ebola LOD Dataset** | **Validation Sources** |
|---|---|
| Health Care Workers Infected | **http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6438a6.htm** |
| | **http://www.who.int/csr/resources/publications/ebola/health-worker-infections/en/** |
| | **http://ecdc.europa.eu/en/press/news/_layouts/forms/News_DispForm.aspx?List=8db7286c-fe2d-476c-9133-18ff4cb1b568&ID=1127** |
| Risk Factors for EVD and Marburg infection | **http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6438a6.htm** |
| Sub-national Indicators Ebola Countries | **http://www.eboladata.org/dataset** |
| Time Series Data | **http://currents.plos.org/outbreaks/article/estimating-the-reproduction-number-of-zaire-ebolavirus-ebov-during-the-2014-outbreak-in-west-africa/#ref5** |
| | **http://knoema.com/xujfoaf/ebola-sub-national-time-series-data-on-ebola-cases-and-deaths-2015** |
| | **https://data.hdx.rwlabs.org/dataset/ebola-cases-2014** |
| | **https://data.hdx.rwlabs.org/dataset/ebola-cases-2014/resource/c59b5722-ca4b-41ca-a446-472d6d824d01** |
| | **http://opendataforafrica.org/ipgyike/national-and-sub-national-data-on-ebola-outbreak-in-western-africa** |
| Weather Data | **https://www.ashrae.org/resources--publications/bookstore/climate-data-center** |
| Weekly EVD cases by country | **http://apps.who.int/gho/data/node.ebola-sitrep.quick-downloads?lang=en** |

| | |
|---|---|
| Ebola Data and Statistics | **http://apps.who.int/gho/data/node.ebola-sitrep.quick-downloads?lang=en** |
| Data for Ebola Recovery | **http://www.theigc.org/project/population-based-surveys-to-inform-ebola-recovery/** |
| Ebola Community Care Centers | **https://www.humanitarianresponse.info/system/files/documents/files/L1_Liberia_NetHope_ETU%2BStatus_22Oct.pdf** |
| Ebola Treatment Centers or Units | **http://www.cdc.gov/vhf/ebola/healthcare-us/preparing/current-treatment-centers.html** |
| | **http://www.cdc.gov/mmwr/preview/mmwrhtml/mm63e1114a4.htm** |
| Ebola Outbreaks before 2014 | **http://www.cdc.gov/vhf/ebola/outbreaks/history/chronology.html#modalIdString_outbreaks** |
| Global Food Prices Database (WFP) | **http://www.numbeo.com/cost-of-living/country_result.jsp?country=Liberia** |
| Health Facilities Liberia | **http://www.who.int/csr/disease/ebola/health-systems/health-systems-ppt1.pdf** |
| | **http://www.eboladata.org/dataset/liberia-health-care-facilities** |
| Human Outbreaks of Ebola (Zaire, Sudan, and Bundibugyo from 1976) | **http://www.cdc.gov/vhf/ebola/outbreaks/history/chronology.html** |
| Lat/Long/Names of ETUs in Liberia | **https://www.humanitarianresponse.info/system/files/documents/files/L1_Liberia_NetHope_ETU%2BStatus_22Oct.pdf** |
| Liberia ETU Constructions | **http://reliefweb.int/sites/reliefweb.int/files/resources/MA012_EbolaResponseUnits_v5-300dpi.pdf.pdf** |
| Liberia FTS Indicators | **https://github.com/luiscape/hdxscraper-fts-ebola/tree/master/data** |
| | **http://www.globalhumanitarianassistance.org/countryprofile/liberia** |

| | |
|---|---|
| Number of health Care Worker Deaths by EVD | **http://ecdc.europa.eu/en/press/news/_layouts/forms/News_DispForm.aspx?List=8db7286c-fe2d-476c-9133-18ff4cb1b568&ID=1127** |
| | **http://www.who.int/hrh/documents/21may2015_web_final.pdf** |

*Table 2: Data validation sources and corresponding sections of Ebola Database*

This appendix outlines the validation sources found online to verify the data in the Ebola database.

## 17. Appendix C – Completed Metrics of Validation

| **Validation Field** |
|---|
| Confirmed Cases |
| Confirmed Deaths |
| Probable Cases |
| Probable Deaths |
| Suspected Cases |
| Suspected Deaths |
| Cumulative Deaths to Date |
| Cumulative Cases to Death |

*Table 3: Metrics of Validation*

This appendix contains the values that were validated by our prototype.

# 18. Bibliography

1. World Health Organization (2016, January). *Ebola Virus Disease*. Retrieved from http://www.who.int/mediacentre/factsheets/fs103/en/
2. US Department of Health & Human Services (2016, February 18). *About Ebola Virus Disease*. Retrieved from http://www.cdc.gov/vhf/ebola/about.html
3. Uniblue Systems Ltd (2016). *.CSV File*. Retrieved from http://filext.com/file-extension/CSV
4. *Python Imaging Library Handbook* (2016). Retrieved from http://effbot.org/imagingbook/pil-index.htm
5. *NumPy* (2016). Retrieved from http://www.numpy.org/
6. *RDF* (2014, February 25). Retrieved from https://www.w3.org/RDF/
7. W3C (2011, March 28). *Turtle - Terse RDF Triple Language*. Retrieved from https://www.w3.org/TeamSubmission/turtle/
8. Python Software Foundation (2016). *Python*. Retrieved from https://www.python.org/about/
9. *RDFLib* (2016, February). Retrieved from https://github.com/RDFLib/rdflib
10. *beautifulsoup4 4.4.1* (2016). Retrieved from https://pypi.python.org/pypi/beautifulsoup4
11. Python Software Foundation (2016). *Data Structures*. Retrieved from https://docs.python.org/2/tutorial/datastructures.html#Dictionaries
12. *Python Data Analysis Library* (2016, February). Retrieved from http://pandas.pydata.org/
13. *Python Pickle Library* (2015). Retrieved from https://docs.python.org/3/library/pickle.html
14. *difflib – Helpers for computing deltas* (2016). Retrieved from https://docs.python.org/2/library/difflib.html