Methodology and Algorithm Development for Historical Addresses

Keagan McNew

Master of Applied Geography in GIS

August 06, 2019

Abstract

Historical GIS is a unique relationship between historical studies, geography, and geographic information science (GIS). This area of study combines several components of various fields and the use of new techniques and technologies to yield new questions, answers, and perspectives on datasets. In this paper, a generalized methodology is created and discussed to process a large historical data set. The dataset consisted of a record of Jewish people who were placed in Budapest which was created shortly after the end of World War II. Several pieces of information were found within the dataset, which included personal information along with addresses for where they lived. The addresses were normalized and joined to another dataset that was manually created for the same area and time from a different dataset. The whole dataset was also geocoded and processed to snap to a streets layer. A program was developed to fully encompass the methodology and would allow the use of any of the created functions to be used at any time.

Introduction

Historical GIS is a nexus for both historians and geographers. Knowles describes the field as a "meeting ground for historians, geographers, [and] geographic information scientists" (Knowles 2002). The field is a unique blend of both time and space, merging datasets of the past with modern technologies. It is a unique field as it requires several specialties from both sides and can provide insights or ask questions that were previously overlooked. With Historical GIS, datasets are often required to give location data. There are a variety of methods to create the data; one is implementing a geocoding algorithm or service.

Geocoding is the process by which locational data in the form a piece of text such as an address is converted into geographic coordinates (e.g. latitude and longitude). There are a variety of methods and processes that go into geocoding, and it remains an ever-developing field. Relatively recently it has grown to house vast datasets to compare inputs and complex interpolation processes in response to the advancement of new technologies. A major component of this paper details the creation of code through python and the utilization of several functions that manipulate the historical dataset. Several robust algorithms were developed to facilitate the required cleaning and normalization of the dataset as well as grant access to the geocoder functions.

This paper aims to provide an overview of the methodology used in creating various GIS layers for historical GIS data. Various processes and software are used to manipulate the dataset and produce the desired GIS layers. Much of the process is created using python and a final application is developed to allow ease of access to each step of the process.

Background and Literature Review

Historical GIS

There are similar and unique problems when dealing with historical GIS datasets versus typical GIS datasets. As with most historical data, the dataset is often found within a medium that is incompatible with more modern techniques and technologies. It would often be required for data to be digitized into a format that can interact with modern technologies. This is a long and tedious process, which typically demands the manual transcription of the dataset into a more advanced format and “takes up the lion’s share of the time that goes into creating a historical GIS” (Knowles 2002). However, as more advanced techniques and tools are developed, there

may be ways to facilitate this stage. Another issue stems from errors found within the record. Unfortunately, if an error is found within the original dataset, it may prove difficult or impossible to determine original intention. However, mistakes in the record may provide insights as well. During the digitization process, errors may also be introduced to the dataset. It is expected that errors will be introduced since as it is a long and tedious process that mainly must be completed by hand. If there is an opportunity to include the assistance from some piece of software, there may still be errors introduced as it is difficult for a computer to process ambiguous data. The result from these errors may stem from the quality of the dataset to being with or with the parameters or modeling algorithm within the software itself.

Geocoding

Geocoding is the process by which a real-world location described in a textual form (e.g. address) is mapped and given coordinates. There is also the concept of reverse-geocoding which is a similar process that takes a set of coordinates and produces an address. There are several methods used in the geocoding process. One method that is heavily used is the comparison of location-related text to a database. Relatively recently, this was a conceptual idea. however, with the "development of true digital geographies" and advancement of technology, software, and the need for spatial information (e.g. GPS, tracking packages, etc.) systems have been developed to accommodate for this (Goldberg, Wilson, and Knoblock 2007). Depending on the service and the algorithm, the "most common data to be geocoded are postal addresses…postal address data are among the most prevalent forms of information" (Goldberg, Wilson, and Knoblock 2007). Several large entities with the resources required have been able to develop vast databases and highly advanced algorithms to provide accurate location data. However, the world is a

disorganized and every-changing place, and no entity can provide a fully complete database. To facilitate this, another method is implemented which interpolates for data that may be missing.

With Geocoding, a major issue is determining where a point would be placed to represent a real-world location. This is a problem that may be quite prominent when utilizing multiple data sources. Especially if different entities who create the data have different rules deciding the placement of a point. Do you place a point at the entrance of a location, a corner of a building, on the street, or center it to the center of a building? As we see in this project, utilizing multiple methods of creating locational data results in several different locations, albeit close to one another. This may not prove to be a prominent issue considering the usage of the data, and often enough maintaining a distance within a certain vicinity to the true location can prove to be acceptable. One issue that often occurs is the assumption of equally spaced out entities. This is especially prevalent when interpolation techniques are utilized to find locations. Unfortunately, streets and addresses are ambiguous and often cause problems during geocoding. For example, perhaps a geocoding service creates a set of addresses along a road and places them equidistant from one another. In the real-world, buildings the addresses belong to may vary in size. Another assumption that is difficult to overcome is that every address is allotted a building. In reality, there are often apartments that may house multiple addresses. However, there are still conventions that street address ecosystems follow (e.g. evens of odds on different sides) which can be used in the creation of a geocoding system. Another issue in the geocoding process is the incompleteness of the database used. This often depends on the entity itself, and the resources they may have at hand. There may also be several issues that can restrict the collection of data such as political ties, difficulty in getting to the location, or the lack of desire for that location.

This can restrict the process that compares a location related piece of text to data that is similarly found within the database.

With geocoding historical address data there are several issues to consider. Unfortunately, geocoding practices do not typically “take the temporal aspect of the query,” but rather present data locations, or the remains of a historic location still around to this day (Cura et al. 2018). Those services typically utilize a type of "hierarchical database" from which the "ground truth" of the location could be extracted (Cura et al. 2018). However, this may not be the case for historical locations, as locations tend to change over time; this is even more prominent depending on the temporal scale of the data. Finally, another issue with the geocoding process and historical location data is that "historical sources for the weaving of a gazetteer are sparse" (Cura et al. 2018). Although modern geocoding services may not initially be developed to accept historical location data, they may still be of use in predicting the locations of the data.

Technology and Software

Excel was used to facilitate the digitization process and to house the digitize data. Excel allowed transcribers an easy way of converting data to a simple table like format that would be used for future projects. The major platform used in this project is the programming language Python. Python is an abstract language that is easy to learn and develop code. There are already a variety of packages (completed code) that offer a wide range of geometric and geographic functionality. There are several main packages, along with some minor packages used throughout the project:

- Shapley is designed specifically to handle geometry objects and perform geometric operations. It is built specifically for these operations and is a cornerstone in other geographic packages.
- PANDAS is a data processing package. It has a vast range of operations and functions that allow a user to easily read datasets from multiple formats. Datasets are read into a data-frame, which can be iterated upon. This allows for functions or other operations to be applied to the whole dataset or certain columns and rows.
- The re module found in the standard library (python's default library) allows for the retrieval of certain patterns found within a body of the text. Several functions can remove or change a desired pattern found in the text.
- Geopy is a package that allows user access to various geocoding services. Functions and methods are developed to allow easy access to the desired service and the returns an output received from the geocoding process.
- Finally, the Tkinter package found in the standard library was used to create a small program. Tkinter allows users to create widgets (e.g. buttons) and connect them to functions or methods.

The tools above offer a unique way of handling historical GIS datasets. If the original intention is non-spatial, "[exploiting] the technology's ability to integrate text, images, maps…" can still provide many benefits (Knowles 2002). Especially with python, as it is an easy language to learn and easy to develop scripts or programs. The utilization of such a tool allows for a degree of flexibility not found in any pre-created program or system. Albeit, the initial process requires the manual transcription of data, once the dataset has been digitized, several technologies can be used to process and analyze the data in quickly. This is especially prevalent

in the cleaning and normalization process, which may require the most work and will ultimately determine the quality of the data.

Sites and Data

The main study area is centered around the city Budapest in 1945, within the districts that were designated at the time as seen in Figure 1. The data itself included personal information of a vast number of individuals (approximately 70,000 individuals). This information included: title, last name, first name, maiden name, gender, birthplace, birth year, the title for mother, mother's name, address, district, and page number as shown in Table 1. The columns this paper centers around is the address and district columns.

title_last_first	last_name	first_name	maiden_name	gender	birth_place	birth_year	title_mother	mother	address	district	page_number
	Weisz	Hugó György		Male	Budapest	1940		Gottselig Sarolta	Dembinszky utca 20	VII	1259
Özv.	Weisz	Hugóné	Goldberger Klára	Female	Budapest	1905		Engler Etel	Akácfa utca 11	VII	1259
Özv. Dr.	Weisz	Hugóné	Dorner Amália	Female	Budapest	1881		Sternlicht Etelka	Katona József utca 25	V	1259
	Weisz	Hugóné	Bronner Éva	Female	Budapest	1924		Kukuk Jolán	Kazinczy utca 55	VII	1259
	Weisz	Hugóné	Weisz Rózsi	Female	Budapest	1896		Weisz Regina	Rákóczi út 47	VIII	1259

Table 1. An example of rows and columns created from the digitization process.

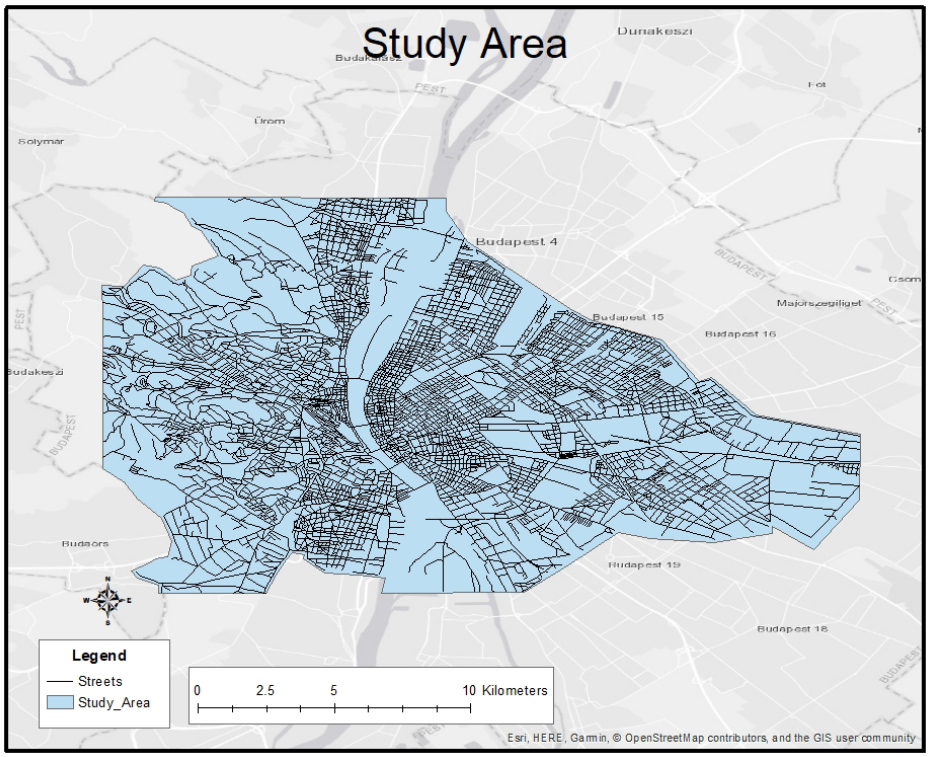


Figure 1. Study area found in Budapest, Hungary.

Methodology

The process that went into the creation of the GIS layers resulted from several processes. These processes consisted of digitizing items, manual cleaning, normalization, joining datasets, geocoding, separation of relevant data, snapping, database creation, and the development of a platform. Python was mainly employed along with several python libraries that provided data management and geographic functionality. The code created for this project was done in the integrated development environment (IDE) Pycharm. This IDE provided and easy to use interface for development of the code. The software known as Anaconda was used to facilitate the organization of the python packages and insure that the proper versions were installed.

Normalization

The dataset was first digitized by entering the records into Excel, and from there the data was integrated to form a large dataset (book dataset). Other datasets that contained streets data (street dataset) or locations with geometry data (placed-locations dataset) would also be included. A data-frame was created for the datasets utilizing the Python module, PANDAS. The creation of a data-frame allowed the program to easily access the Excel files. The data-frame also facilitated the need to iterate over the dataset which would be required for several functions that were used later. Data-frame management functions were created to create the data-frames and find the appropriate column that housed location data.

Although the dataset was manually inspected for error, common mistakes may continue to plague the dataset. Aside from manual cleaning, several automated cleaning functions, and normalization functions were created to facilitate the preparation of the dataset. Several of these functions included regular expressions. Regular expressions are patterns that can be developed to select a piece of textual information from a larger piece of text. These patterns were used to select whitespace, certain characters, and characters used to signify unknown values as seen in Table 2. Undesired characters were either changed or removed from all three datasets. Once values were cleaned, the columns that housed location data were normalized. Normalization of address data required the parsing of addresses. For the book dataset and placed-locations dataset,

regular expressions were created to separate street information from address number information. The district column was already separated, a city and country column were added. Finally, a full address column was created in the book dataset that contained the street, address number, district, city, and country.

	title_last_first	last_name	first_name	...	district	page_number	notes
0	Özv.	Aczél	Gyuláné	...	II	2	NaN
1	Özv.	Adler	Dezsőné	...	VII	3	NaN
2	NaN	Ádler	Ármin	...	VII	3	NaN
3	NaN	Adler	Hilda	...	VII	4	NaN
4	NaN	Adler	Miklós	...	XII	5	NaN
5	Özv.	Adler	Pálné	...	VII	6	NaN
6	Özv.	Adler	Samuné	...	VII	6	NaN
7	NaN	Agulár	Györgyné	...	VII	7	NaN
8	NaN	Alapi	Béláné	...	II	7	NaN
9	NaN	Allpár	Tiborné	...	V	9	NaN

Figure 2. A PANDAS data-frame as see in the python output terminal. Pandas data-frames are like tables found in excel, however, they contain an index as seen to the left by default.

Expression	Description
\s(u|U)?\.	This expression was used with the streets dataset to change an "u." to "utca".
\b[uU](?=\b|\d)	This expression was used with the address dataset to change an "u." to "utca". This one is different from the street version as there was a whole address to consider.
[\.\?\!,]	This expression was used to remove punctuation.
[-—]|[\(-\)]	This expression was used to remove values that represent unknown or null.
\d+\-\d+|\d+\/*\w*|\(\-\)	This expression was used to highlight the number from the address

Table 2. This table shows the various regular expressions used to process the data.

Joining

Once the datasets were cleaned and normalized, an attribute join was used to find values from the book dataset that matched with the value found in the placed-locations dataset. This was done using PANDAS merge function and a left join was used to join the two datasets on the original address column. The original address column was used since both the book dataset and placed-locations dataset had the same data. A left join was used to keep all values from the book dataset, but append the values found in the placed-locations dataset (geometry values) to matching rows. A final merged dataset was created which result in approximately 20,000 rows being matched and given geometry data.

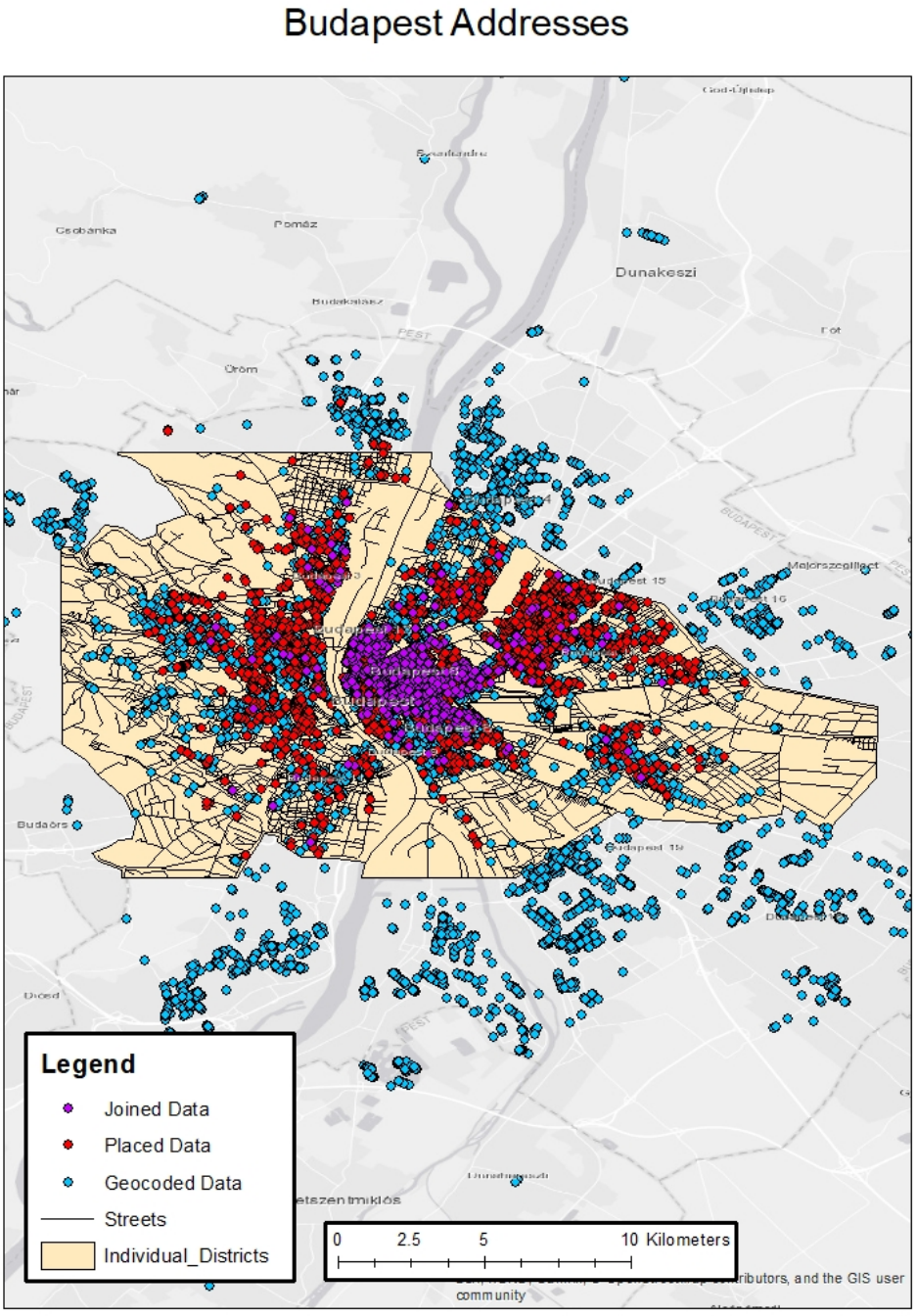
Geocoding

The merging of the dataset left approximately 50,000 address that still required locational data. Several geocoding functions were created to process the points. The Python library geopy was used to facilitate the connection between geocoders and the data. Through geopy Googles geocoding service (GoogleV3) and Open Street Maps geocoding service (Nomintim) can be accessed. A user agent or API key is then required to make use of one of the two geocoding services. The API key or user agent are required based on the design of the geocoding services. Accounts are required to place restrictions on usage of these services and potentially charge account owners a fee for utilizing those services. Google provides a trial period for 12 months for new accounts which allows for certain benefits as well as allocates a monthly amount of $200 along with a one-time allotment of $300 for the entire trial period. For Open Street Maps, the datasets and services are open-sourced, however certain restrictions are still placed (often stricter in terms of how much access is granted per day). A user agent parameter is required to link accounts to enforce restrictions. The function also accepts a point of interest and a geocoder

name. The function was utilized with the PANDAS data frame to properly loop through each record. Finally, a latitude and longitude column were created to store the data. As of now the geocoder functionality only supports two sets of geocoding services. Due to funding and access to certain resources, ArcGIS geocoding service was used in place of the mentioned geocoder services. For the scale of this project, resources provided by Texas State University in the form of ArcGIS Online credits were provided to allow geocoding. Unfortunately, due to the unique way of accessing the school's account to utilize the allocated credits, this functionality was not added to the final program. Geocoding was completed on all points with the new latitude and longitude data and appended to datasets that already had location information. Figure 3 depicts the points that were created through manual placement, geocoding, or joining of datasets. Figure 4 is set to a larger scale to depict the same datasets, while Figure 5 is scaled to show the extent of the dataset. Several addresses were located far outside the study area, which may be a result of several different factors.

Separation

Datasets were then processed through a separation algorithm that was designed to separate the addresses and streets into pairs. The parsed addresses and streets that matched to each other by name of the street. This was first done by creating a reference address list function that would accept the merged data frame. This function would parse the data frame and pull out all the addresses and create a unique address list. Next two functions were created to parse the merged data frame and the streets data frame. The separate addresses function would loop through all the addresses in the merged data frame appending data to a "points of interest" list should the addresses be found in the first location of the unique address list.



in Figure 3. A basic map detailing some of the points from the dataset. The point shown in red is from a dataset that was manually placed. Points in blue are those that were geocoded through the ArcGIS geocoder. Points in Purple are locations from the book dataset that matched up with locations from the placed dataset.

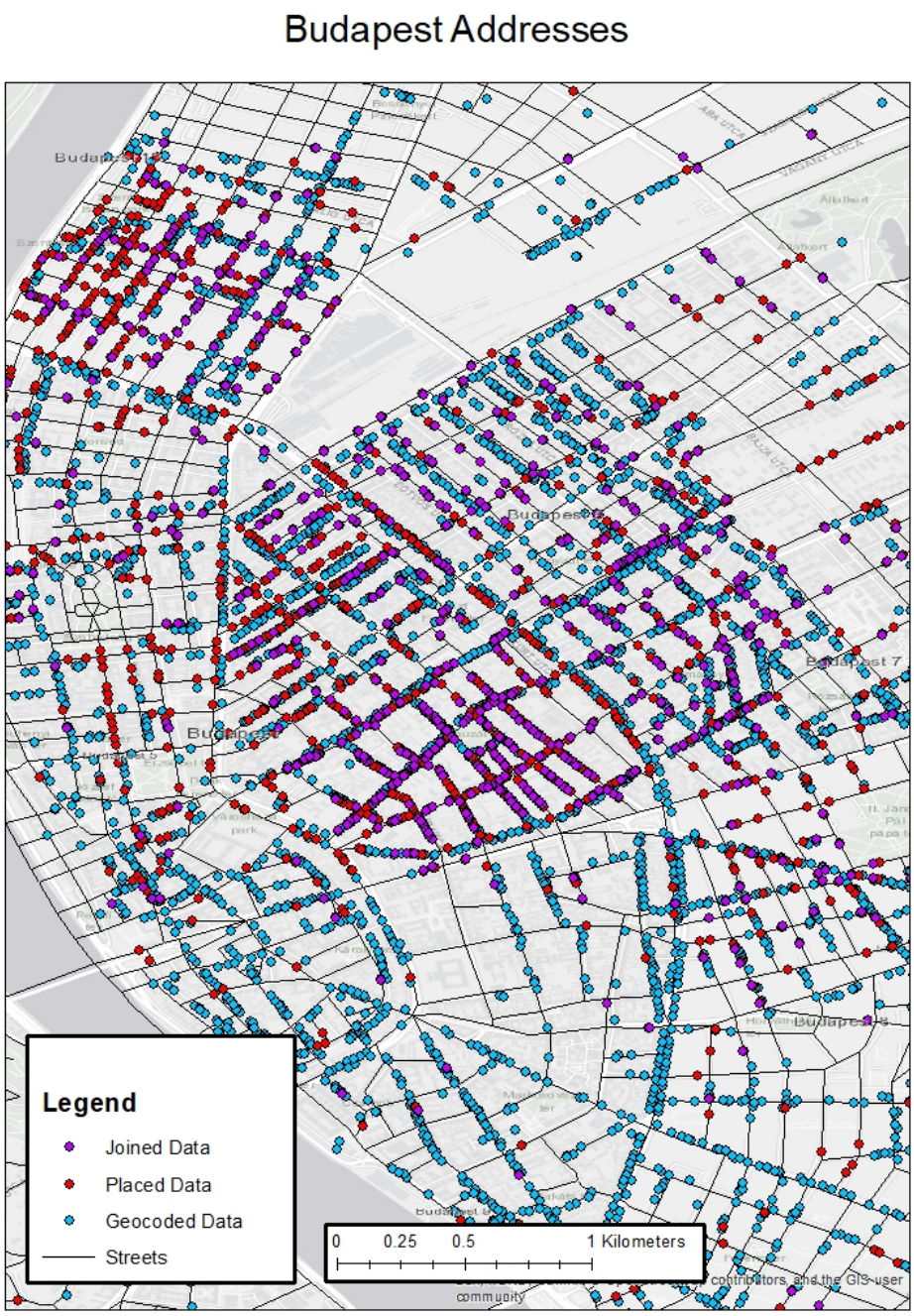


Figure 4. Close up version of Figure 3 detailing the points and various sources of location data

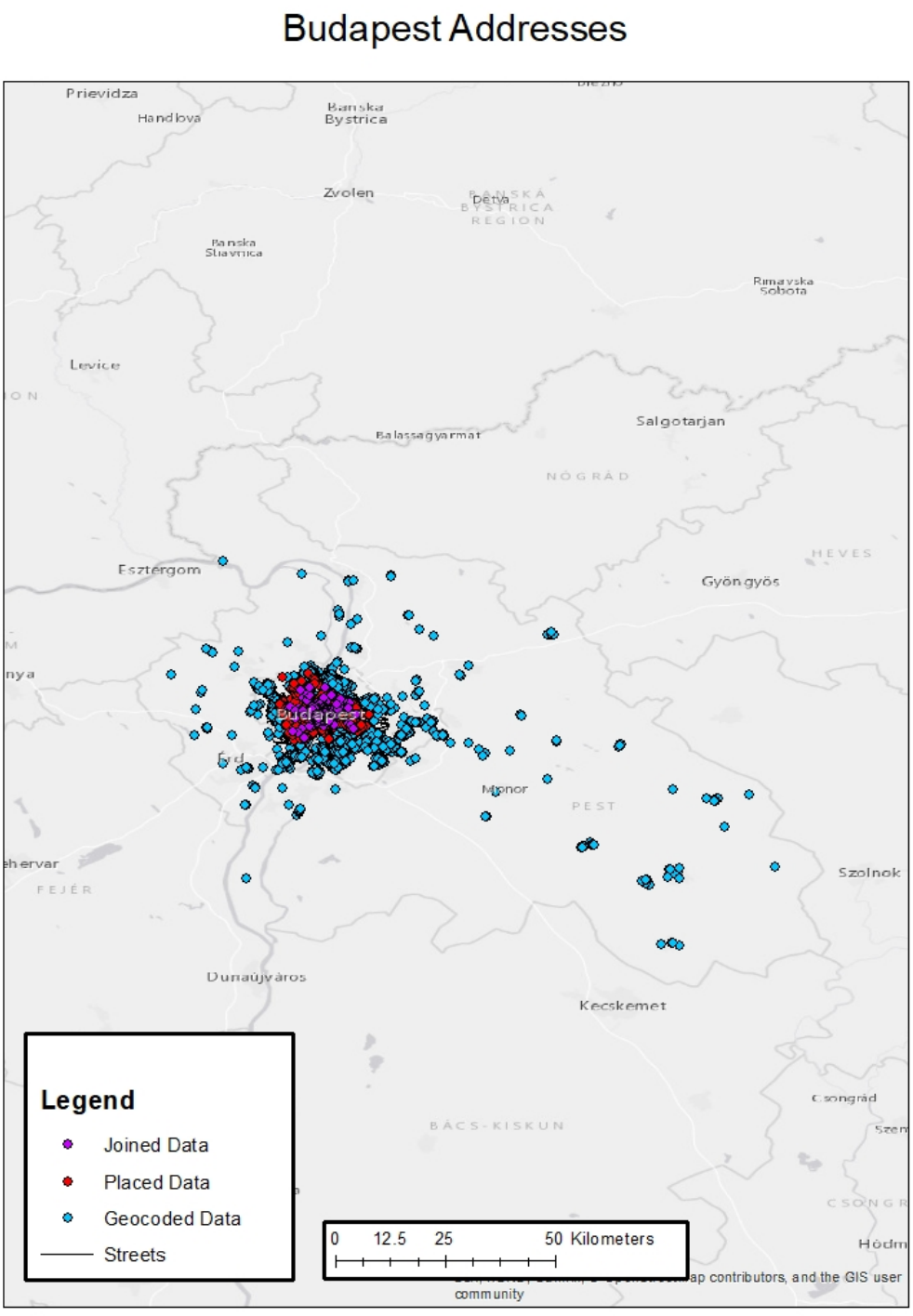


Figure 5. A full map that showcases the data. Several addresses were located outside of the study area, and this may result from them being located correctly or a misunderstanding between the address and the geocoder.

The separate streets function would provide similar functionality as the previous function but would append street information should the street name be found in the first location of the unique address list. The first location of the unique address list was used with both datasets to create a bridge and pull relevant data that would correspond to each other. Finally, the unique address list would have the address found in the first location removed from the list, thus "iterating" over the list and removing any chance of an accidental reoccurrence of a street name. The main purpose of the separation functions would help snap points to the appropriate streets.

Transformation

The next step of the process involved the creation of several transformation functions. Geocoding the addresses provides latitude and longitude location data in WGS84, however, the datasets were transformed to UTM34N. PROJ4 and pyproj were used to facilitate the transformations, and a function was created to accept both line and point objects. These objects were first transformed into GEOJSON, which allowed for the object data (e.g. geometry) to be accessed. This was done by creating a conversion function that would take a data frame (either the merged data frame or streets data frame) and build the GEOJSON, placing attributes such as latitude and longitude into the appropriate keys as seen in Figure 4. For line objects, the geometry column was iterated over for every x and y pair, which were then transformed to the desired projection. Point objects were simpler and a similar process (excluding the iteration) was performed.

```
{“type”: “Feature”,
  “properties”: {},
  “geometry”:
        {“type”:
              “Point”,
              “coordinates”: [longitude][latitude]}}
```

Figure 6. An example of a point geojson object. This would later be used to pass into the shapely interpolate function for snapping the data.

Snapping

Snapping was the final processing step for the datasets. Snapping functions were created to snap the points to streets that had the same street name found in the points. This was done mainly through the Python package shapely. New line and point objects were created (with a projected coordinate system) and were ran through the interpolate function. This function makes use of linear referencing, which finds the most straightforward path between two points. The boundaries of each new point were then updated with the boundaries found from the interpolating function. Finally, a new data frame called snapped points were created with the updated point locations.

Database

Extra functionality was added to the program to facilitate the creation of a PostgreSQL database. A simple script was created that utilized PANDAS, psycopg2, and sqlalchemy to create a connection between the dataset and the database. PANDAS contain methods designed to save to a SQL, while psycopg2 and sqlalchemy were required to create the connection to the database.

The result is a database with the same basic format that the dataset was originally found in. Next, an ontology was created to reference and help facilitate the normalization of the database. The ontology is centered around a Person class as depicted in Figure 8. which acts as a primary key to classes that consist of personal information of the individual and their respective attributes. A separate Address class as depicted in Figure 7. is created and acts as a secondary key to various geometry and geography related classes. This ontology was created in order to guide the creation of tables that would follow this scheme. Finally, the dataset was normalized by separating classes and removing repeating values. This would allow changes to be made and affect the dataset. The creation of a database will allow for further analysis and provide a method of querying of the dataset. PostGIS can be utilized to perform more advanced spatial queries or functions on the dataset.

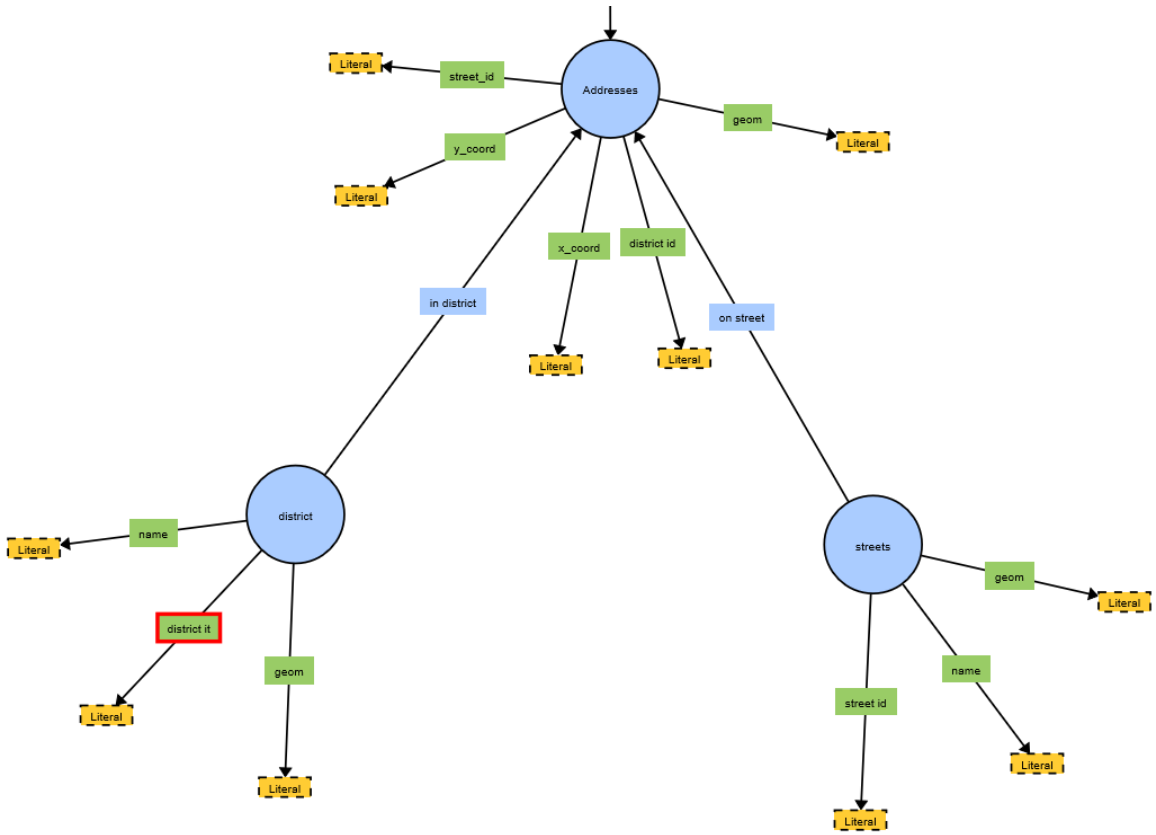


Figure 7. Part of the ontology that would be used in the creation of the database. Here the address class along with the streets and district classes is shown along with their respective attributes and relationships.

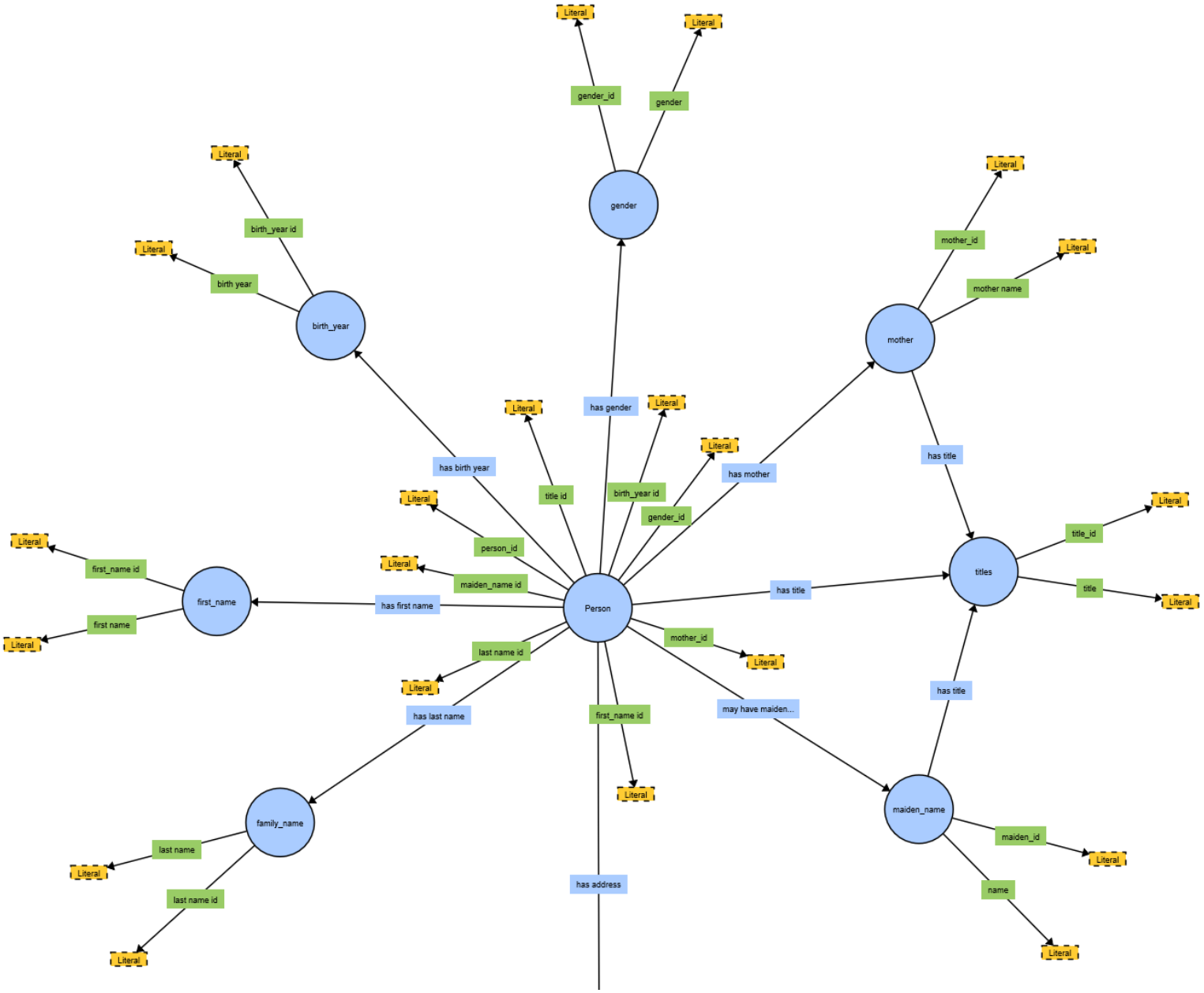


Figure 8. This is another part of the ontology that would be used in the creation of a database. The classes here consist of all the personal information within the dataset such as first name, last name, gender, family name, maiden name, title, and mother. Attributes for all the classes are also shown here.

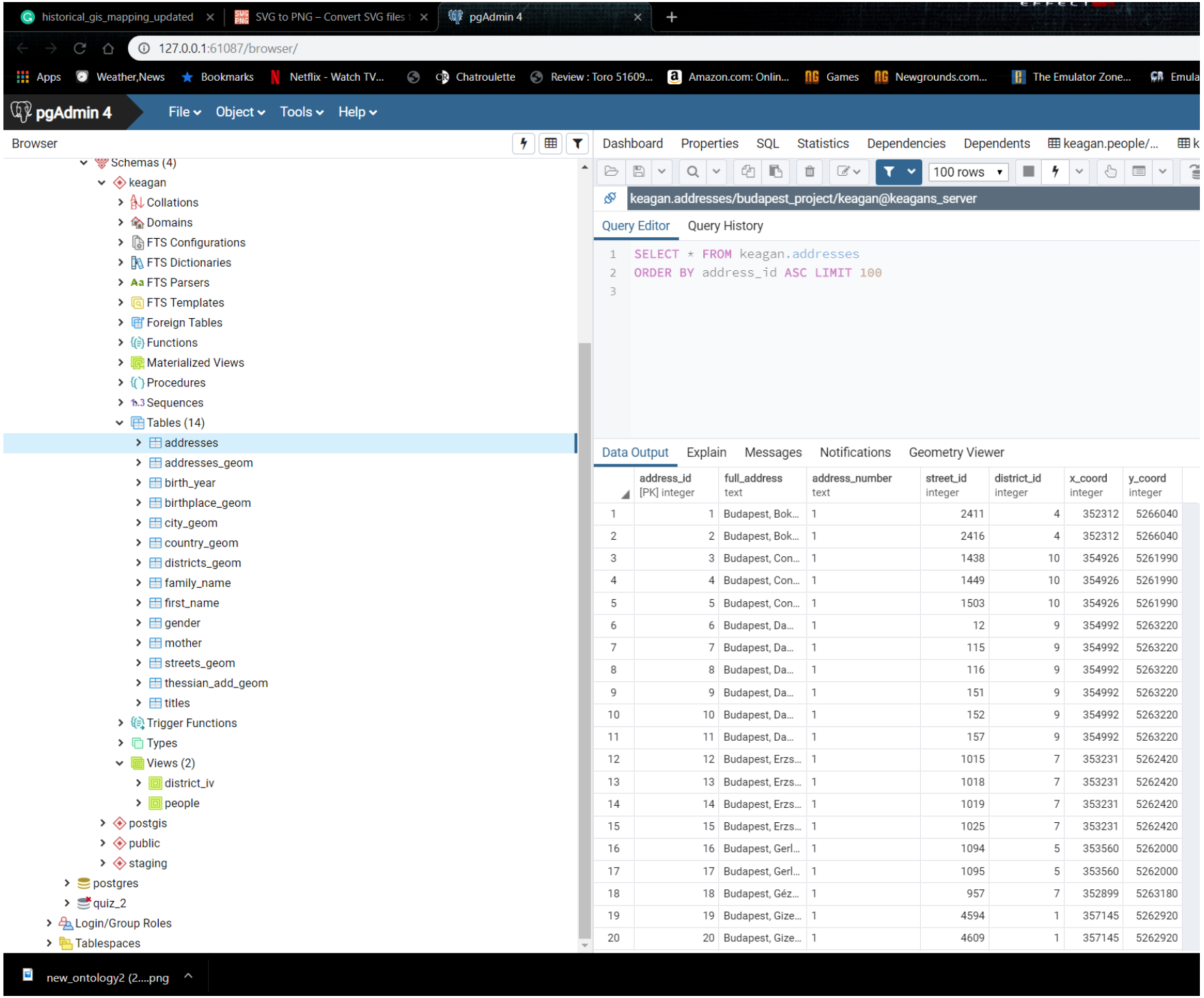


Figure 9. PgAdmin4 is a platform that can be used to work on a relational database. In this instance, the address table is viewed. The dataset is normalized which is indicated by the ids (street_id,district_id,etc.)

Platform – Tkinter Interface

With the creation of the main functionality, the final step was to connect and wrap up all the functions mentioned above. This was done by creating an object-oriented based graphical user interface (GUI). Several classes were created to house the respective functions along with the necessary components to create the various parts and widgets for the GUI. Three main classes were created that would act as the main body of the program. The Main class is the container that would house the rest of the classes and functionalities; generalized variables were created and would act as global variables for the entire program (e.g. input data-frame, data text box). These global variables could be changed by the various functions and provide connection between the classes. The Toolbar class is a robust class that can contain several basic tools such as saving and opening files, along with the ability to accept several other classes or functions in the form of a list of dictionaries. The Table class is a simple class that is designed to contain a textbox object; this object allows for the visualization of text-based datasets. The InputContainer class is a general class that is used to help structure the creation of other, separate classes. Essentially this class is used to build and house the various tools created for this project. Creating separate classes and base classes allowed for a high degree of flexibility and organization within the program, and new tools or functions can easily be inserted or removed. It was also important to maintain a high degree of the organization due to the creation of many various widgets for each class. A full diagram showing the relationships between the classes can be found in Figure 10. A full explanation of the functions used in this project is found in Table 3, that depicts the name, parameters, and what the function should return along with a description. Figure 11 showcases

what the GUI would look like, with this particular instance encompassing the normalization process.

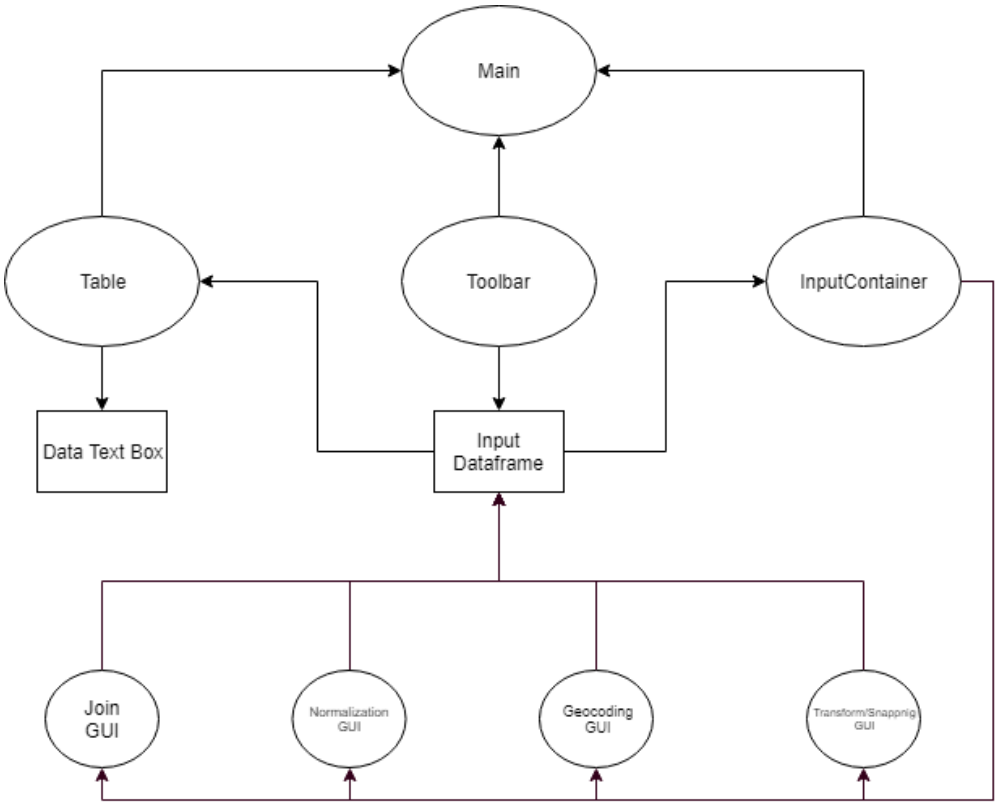
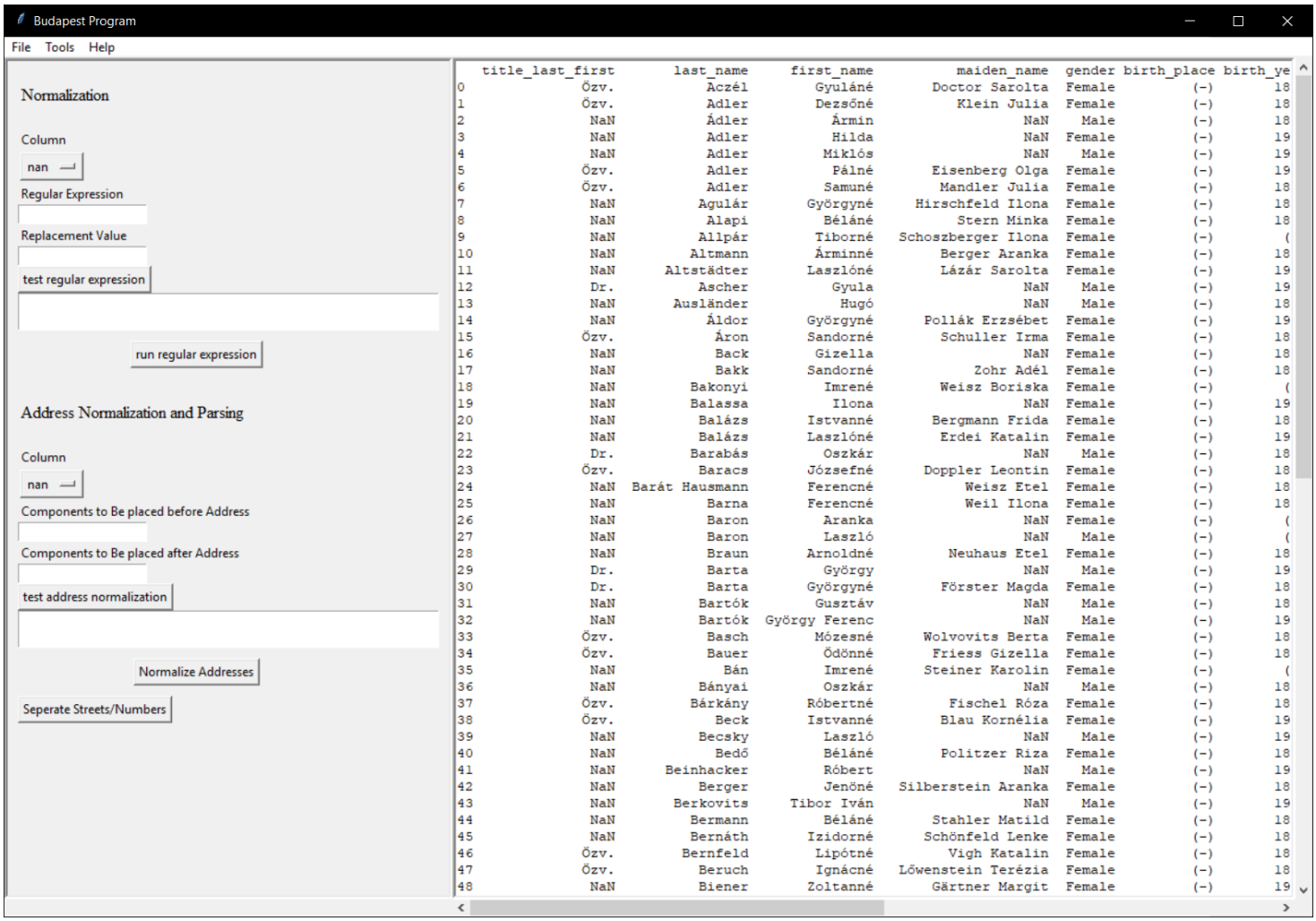


Figure 10. The chart depicts the various relationships between the classes and major variables. All the classes meet and connect through the main class, but the input data frame is a true nexus. All classes share the same input data-frame variable. When one class or function changes the state of the input data-frame, the rest of the classes and functions will use the updated input data-frame. When the input data-frame is altered, the data text box is updated.

Normalization				
Function	Parameters	Returns	Packages	Description
normalize_column	a data frame, column name, regex pattern, replacement text	pandas data frame	pandas, re	This function accepts a data frame and the column in question with data in the form of text and replaces a part of the text with the desired replacement text through the use of a regular expression.
create_full_address	a data frame, column name, before items, after items	pandas data frame	pandas, re	this function accepts a data frame and the column in question with data in the form of an address and appends data before it and after it. This allows for the creation of a full, in-depth address that can be passed to geocoding services.
separate_street_and_number	a data frame, column name	pandas data frame	pandas, re	This function accepts a data frame and column in question with data in the form of an address and splits the address between the street and address number. This is done through the use of two very similar regular expressions, however, one method in the function extracts the number, while another method replaces the number with "(essentially removing the number).
Join				
merge_dataframes	an input data frame, join data frame, join column, join method	pandas data frame	pandas	This function accepts two data frames, the input data frame that is to be joined to the join data frame. A join column is decided which is a similar column between the two datasets, in this case, the address column. A join method is also used to set the type of join, by default it is set to a left inner join.
Geocoding				
geocoder	geocoder service, user agent or API key, data frame, column name	latitude list, longitude list	pandas, geopy	This is a function that allows access to various geocoder services. At the time, only two services can be used, which include Google and Open Street Maps. A data frame along with the column of interest is passed, and a user agent or API key is required to unlock access to the service. The user agent or API key is one variable but may vary with what it is. A user agent is most likely a label of sorts (an email) while an API key is a long string of assorted letters and numbers that connect an account to the program.

Transformation				
create_reference_address_list	a data frame, streets column	address list	pandas	This function simply creates a unique list of addresses based on a street's column found within a data frame.
convert_to_streets_and_addresses_to_geojson	streets list, address list	street geojson, address geojson	n/a	This function takes in a streets list and an address list and iterates through them. While iterating, the function places the components of the lists into the appropriate positions usually found in a geojson file.
transform_geom	geojson, input coordinate reference system, output coordinate reference system	geojson	pyproj	This function iterates there a geojson and transforms the current coordinate system to a new coordinate system. It can accept line objects or point objects.
Separation				
separate_streets	a data frame, reference list	street list	pandas	This is one of two functions that will iterate through a data frame, compare it to a reference list, and output a list dataset that is similar to one another found in the data frame. This particular function iterates through a streets data frame and pulls out streets that contain the same name. The reference list is a list of addresses and is meant to keep track of which set of streets/addresses the function is on.
separate_addresses	a data frame, reference list	address list	pandas	This is one of two functions that will iterate through a data frame, compare it to a reference list, and output a list dataset that is similar to one another found in the data frame. This particular function iterates through an addresses data frame and pulls out addresses that contain the same street name. The reference list is a list of addresses and is meant to keep track of which set of streets/addresses the function is on.
Snapping				
snapping_function	an address data frame, streets data frame, input coordinates, output coordinates, reference list	pandas data frame	pandas, Proj, shapely	This is a function that makes use of several other functions during a large iteration cycle. The functions found within the transformation and separation processes are used during each iteration. This enables only addresses and streets that contain the same street name to be considered and thus snapped. The snapping process creates a new geometric object and utilizes shapely interpolate method. Utilizing linear referencing, the shortest path between the point and line can be found, and the original points have their coordinates changed to the position. The process is then repeated for a new set of address-street pair.

Table 3. Various functions and their parameters, returns, and description used in the methodological process.



	title_last_first	last_name	first_name	maiden_name	gender	birth_place	birth_ye
0	Özv.	Aczél	Gyuláné	Doctor Sarolta	Female	(-)	18
1	Özv.	Adler	Dezsőné	Klein Julia	Female	(-)	18
2	NaN	Ádler	Ármin	NaN	Male	(-)	18
3	NaN	Adler	Hilda	NaN	Female	(-)	19
4	NaN	Adler	Miklós	NaN	Male	(-)	19
5	Özv.	Adler	Pálné	Eisenberg Olga	Female	(-)	19
6	Özv.	Adler	Samuné	Mandler Julia	Female	(-)	18
7	NaN	Agulár	Györgyné	Hirschfeld Ilona	Female	(-)	18
8	NaN	Alapi	Béláné	Stern Minka	Female	(-)	18
9	NaN	Allpár	Tiborné	Schoszberger Ilona	Female	(-)	(
10	NaN	Altmann	Árminné	Berger Aranka	Female	(-)	18
11	NaN	Altstädter	Laszlóné	Lázár Sarolta	Female	(-)	19
12	Dr.	Ascher	Gyula	NaN	Male	(-)	19
13	NaN	Ausländer	Hugó	NaN	Male	(-)	18
14	NaN	Áldor	Györgyné	Pollák Erzsébet	Female	(-)	19
15	Özv.	Áron	Sandorné	Schuller Irma	Female	(-)	18
16	NaN	Back	Gizella	NaN	Female	(-)	18
17	NaN	Bakk	Sandorné	Zohr Adél	Female	(-)	18
18	NaN	Bakonyi	Imrené	Weisz Boriska	Female	(-)	(
19	NaN	Balassa	Ilona	NaN	Female	(-)	19
20	NaN	Balázs	Istvanné	Bergmann Frida	Female	(-)	18
21	NaN	Balázs	Laszlóné	Erdei Katalin	Female	(-)	19
22	Dr.	Barabás	Oszkár	NaN	Male	(-)	18
23	Özv.	Baracs	Józsefné	Doppler Leontin	Female	(-)	18
24	NaN	Barát Hausmann	Ferencné	Weisz Etel	Female	(-)	18
25	NaN	Barna	Ferencné	Weil Ilona	Female	(-)	18
26	NaN	Baron	Aranka	NaN	Female	(-)	(
27	NaN	Baron	Laszló	NaN	Male	(-)	(
28	NaN	Braun	Arnoldné	Neuhaus Etel	Female	(-)	18
29	Dr.	Barta	György	NaN	Male	(-)	19
30	Dr.	Barta	Györgyné	Förster Magda	Female	(-)	18
31	NaN	Bartók	Gusztáv	NaN	Male	(-)	18
32	NaN	Bartók	György Ferenc	NaN	Male	(-)	19
33	Özv.	Basch	Mózesné	Wolvovits Berta	Female	(-)	18
34	Özv.	Bauer	Ödönné	Friess Gizella	Female	(-)	18
35	NaN	Bán	Imrené	Steiner Karolin	Female	(-)	(
36	NaN	Bányai	Oszkár	NaN	Male	(-)	18
37	Özv.	Bárkány	Róbertné	Fischel Róza	Female	(-)	18
38	Özv.	Beck	Istvanné	Blau Kornélia	Female	(-)	19
39	NaN	Becsky	Laszló	NaN	Male	(-)	19
40	NaN	Bedő	Béláné	Politzer Riza	Female	(-)	18
41	NaN	Beinhacker	Róbert	NaN	Male	(-)	19
42	NaN	Berger	Jenőné	Silberstein Aranka	Female	(-)	18
43	NaN	Berkovits	Tibor Iván	NaN	Male	(-)	19
44	NaN	Bermann	Béláné	Stahler Matild	Female	(-)	18
45	NaN	Bernáth	Izidorné	Schönfeld Lenke	Female	(-)	18
46	Özv.	Bernfeld	Lipótné	Vigh Katalin	Female	(-)	18
47	Özv.	Beruch	Ignácné	Lőwenstein Terézia	Female	(-)	18
48	NaN	Biener	Zoltanné	Gärtner Margit	Female	(-)	19

Figure 11. An example of the program created utilizing tkinter. Inputs are typically found on the left while the dataset is found on the right. In this instance, the normalization GUI is activated and can accept a wide range of inputs and provide areas for testing.

Discussion

Several issues common to historical GIS arose during the processing of the data. Historical datasets often require a long and tedious process of digitization. Digitizing them can be an extensive and time-consuming process. Due to the characteristics and quality of the original of the data source, a manual approach was taken. However, the initial transference from the book to digital media is only the beginning, and further work is needed to clean and prepare the data for analysis. Once the dataset is in a digital format, Python was utilized to provide a platform for developing the necessary functions required to process the datasets. Several functions could then be created to filter the data, clean up text, joining and removing rows, finding and creating location data, and snapping points onto a network.

Concerning the manual entry of the book, mistakes were common throughout the dataset. Mistakes made by previous transcribers may negatively affect the dataset. One method to improve the dataset was to manually make corrections while going through the dataset. Manually making corrections may still cause human prone errors to remain unchanged or could create new errors. Another method is to normalize and clean the dataset using an automatic method such as an algorithm. Depending on the design of the algorithm, the consideration of mistakes, and the extent of mistakes, many common problems and be eliminated. For example, removing punctuation is a simple mistake that can fixed easily because there are only a few characters used for punctuation. Extra white space is another simple mistake that can be fixed as it is a character that represents space. However, there are some problems when special characters or complicated groups of characters are used. For example, some rows did not have a location, thus characters such as: -, (-), (-) were used to signify the lack of a location. Another issue came from the normalization of addresses, especially with the removal of abbreviations. This was especially

prevalent with the “u.” which was an abbreviation for “utca." A complex regular expression was created to find all instances of "u." regardless of location within the address and while skipping over the character "u" when it was part of a different word. However, there are still some instances that will escape the normalization process. For example, there was one instance where "u." was attached to the street name without space. Further development and consideration of the types of errors present may help provide better solutions in utilizing this process in the future.

It may be difficult to accurately place locations in a GIS as references may not be of the highest resolution, locations change, mistakes in the record. Often enough, manually placing historical locations in a GIS requires the use of historical records and maps. For this project a historical map was used to guide the placement of the first 3000 points. However, when compared to base maps, the locations and streets do not match up at certain locations. However, relative to the historical map, the original 3000 points line up close to what was deemed their true locations. For this project, whole datasets were created during the geocoding process to maintain a degree of individuality between datasets. This way, the dataset will still undergo a merging process to eliminate several rows (with the 3000 points already placed on the map); while also being geocoded by different geocoders regardless if locational data may exist for the record or not. However, when the new datasets need to be appended to this final dataset, it may be prudent to consistently use a single data source (manual, geocoder, etc.) which may provide more consistency.

Several things could help alleviate some of the discussed issues and uncertainties. One concept that may help would be to include a fuzzy-based approach that takes in various outputs of geocoded points and creates an output that represents the degree of certainty of where that address would be located. There may be an opportunity to use fuzzy techniques that include the

use of the Levenshtein distance to compare textual data. In the context of historical datasets, it may be prudent to accept a fuzzy product. In the case with using modern geocoders, comparing their results to one another could yield an area that would contain that location of interest. Other techniques and algorithms that utilize machine learning could also be used to train datasets and predict the location of new addresses. For this particular project, the placed address data could be used to train a machine learning program, from which the book dataset could be processed through and interpolated.

Conclusion

Historical GIS a unique blend of fields that focus on both time and space. It requires knowledge and skillsets from history, geography, and GIS, and can yield new answers and questions. The advancement of more modern analysis techniques and technologies can help facilitate and further advance the field. However, the procedures required for the processing of historical GIS datasets is long and tedious, mainly due to the unique characteristics of the datasets. This is especially prevalent in finding a major component of historical GIS datasets: the location of various features.

In this paper, a methodology was developed to process and locate various historical addresses across Budapest, Hungary. Much of the processing and data manipulation was done using Python and a variety of Python packages. These addresses were first digitized by hand into an excel file and manually inspected for mistakes. Another round of cleaning and processing involved the creation of regular expressions to pull out certain groups of characters from the text, and either remove or replace them. The data set was then joined to another dataset that already contained location data. Although location data was appended to some locations, the entire

dataset was geocoded utilizing ArcGIS's geocoding service. These points were then separated by similar address street name. From there data was projected and snapped to a polyline layer. A full program was developed to facilitate the need to perform any of the steps above in any order. During the creation of the GIS layers, there are several considerations to deliberate on. Many issues reflect those that often arise when working with historical datasets, such as the tedious process of digitization or human-induced errors during the digitization process. With the creation of the locational dataset, the joining process would allow the work that went into the placing of the addresses on the map from a separate dataset, to transfer over to this dataset. The geocoding process ended up geocoding several of the locations, although the accuracy of those locations greatly depends on the service and the construction of the address text. Some several more techniques and technologies were considered for this project, and future implementation of those may yield new and interesting results.

References

Anaconda Software Distribution, Vers. 2-2.40, Anaconda

Cura R., Bumenieu B., Abadie N., Costes B., Perret J., Gribaudi M. 2018. Historical Collaborative Geocoding. *International Journal of Geo-Information* 7(262)

Diener, M. 2015. *Python Geospatial Analysis Cookbook*. Birmingham, UK: Packt Publishing Ltd.

geopy, Vers. 1.20.0, GeoPy Contributors

Goldberg, D., Wilson, J., and Knoblock, C. 2007. From text to geographic coordinates: the current state of geocoding. *URISA Journal* 1(19):33-46

Google 2019. Developer Guide. https://developers.google.com/maps/documentation/geocoding/intro (accessed August 06, 2019)

Knowles, A., 2002. *Past Time, Past Place GIS for History.* Redlands, California: ESRI

matplotlib, 3.1.0, John D. Hunter and Michael Droettboom

Open Street Maps. 2019. Nominatim. https://wiki.openstreetmap.org/wiki/Nominatim (accessed August 06, 2019)

pandas, Vers. 0.25.0, The PyData Development Team,

psycopg2, Vers. 2.8.3, Federico Di Gregorio

Pycharm Community Edition, Vers. 192.5728.105, JetBrains

pyproj, Vers. 1.9.6, Jeff Whitaker

seaborn, Vers. 0.90, Michael Waskom

shapely, Vers. 1.6.4post2, Sean Gillies

sqlalchemy, Vers. 1.3.6, Mike Bayer