3-21-2019

Social Network Threat Detection

Nathanael R. Beveridge

Social Network Threat Detection

THESIS

Nathanael R. Beveridge, 2nd Lt, USAF

AFIT-ENS-MS-19-M-101

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-19-M-101

SOCIAL NETWORK THREAT DETECTION

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Nathanael R. Beveridge, B.S.

2nd Lt, USAF

21 March 2019

AFIT-ENS-MS-19-M-101

SOCIAL NETWORK THREAT DETECTION

THESIS

Nathanael R. Beveridge, B.S.
2nd Lt, USAF

Committee Membership:

Lt Col Andrew J. Geyer
Chair

LTC Christopher M. Smith
Member

Abstract

Various government agencies have a stake in knowing when bad actors cross the United States' borders, or how bad actors may be involved in the flow of people across borders. Interviews conducted at border checkpoints with individuals who intend to cross the border can contain valuable information. The quantity of interviews is such that intelligence analysts could benefit greatly from an automation system that extracts the information they are looking for from within the interviews. This would allow them to focus more of their time on analyzing what is extracted as opposed to inspecting all interviews themselves. The information extracted can be written to an SQL database, allowing the information to then be easily and efficiently queried for valuable insight and analysis.

Acknowledgements

Thank you to my advisor, Lt Col Andrew J. Geyer, for all of his help, and to LTC Christopher Smith for being on the committee.

Nathanael R. Beveridge

Table of Contents

Page

I. Introduction

1.1 Background

The United States is currently immersed in a national immigration debate. Irrespective of political opinion on the matter, numerous government agencies have a stake in understanding who is moving in and out of the country, who may be helping people into the country, and how transnational criminal organizations may be involved. According to the U.S. Customs and Border Patrol (CBP) Border Security Report for fiscal year 2017, the CBP is concerned about the steady increase in the flow of people over the border since May 2017. Transnational criminal organizations are exploiting legal and policy loopholes to help illegal aliens gain entry and automatic release into the country [1]. U.S. Border Patrol agents arrested 20,031 criminal aliens as well as another 10,908 individuals who were wanted by law enforcement [1]. Furthermore, the Department of Homeland Security, in their 2018 Border Security Metric Report, cites a notable change in the use of smugglers for entry into the country. The use of a smuggler for first time entry into the country has increased from 40-50% in the 1970's to 80-95% in 2015 [2]. In addition, they cite an increase in the fee charged by smugglers. Interviews conducted by CBP found that with these higher fees have come a corresponding increase in alternate forms of payment including smuggling illegal substances across the border [2]. These reports indicate the importance of being able to track movement into the United States and uncover criminal networks that support this movement.

Various government agencies use border crossing intelligence and data in support of anti-terrorism efforts. They rely on information collected by CBP through face to face interviews conducted at various checkpoints along the United States southern border. Intelligence analysts for these agencies use the information in these interviews to continuously stay up to date on criminal networks that may pose threats to the country. This is a common approach for detecting threats and keeping track of criminal networks. Unfortunately, it is difficult to effectively glean information from the vast amount of interviews being generated with a small number of analysts. Even with a large staff, the nature of the task depends on the ability to make "needle in the haystack" type connections between people that span the entire world and relationships that could go back years. In this instance, the analyst themself holds the keys to all the relationships. This poses a problem. Information may be forgotten or lost even with the best analysts. Also, if an analyst leaves the job, they take with them years worth of knowledge gleaned from reading interviews. For instance, imagine in 2015 an analyst reads an interview in which Person A referenced Person B as a brother. Two years later, another interview is conducted with Person B and it is discovered he has strong ties to a terrorist organization. A year later, a new analyst conducts the interview analyses and encounters an interview in which Person A is picked up and interviewed while en route to a United States port of entry. This raised no flags to the new analyst because they were not aware of the relationship between the two brothers. A combination of all the aforementioned reasons could contribute to this bit of information falling through the cracks.

1.2 Problem Statement

Given the vast amounts of interviews being processed by federal and state agencies and the inherent challenges present in incorporating ever changing information into

existing criminal networks, this thesis seeks to develop an automated text mining system that processes the interviews and writes the relevant pieces of information into an SQL database. In addition, the thesis aims to provide examples of the type of analysis that can be performed when the information is stored in a relational database like SQL.

1.3 Approach

Create a rules based algorithm built on top of a conditional random field named entity classifier to extract relevant information from interviews. This information is then written to a relational database for storage and intelligence analysis.

1.4 Research Goals

This research demonstrates that natural language processing can be used to process intelligence reports, creating structured data from unstructured text which can improve the effectiveness and speed of human intelligence analysis.

II. Literature Review

2.1 Overview

Due to the massive amounts of unstructured data facing analysts today, manual inspection of documents to uncover relevant information is not feasible. Broadly speaking, unstructured data is information that is not stored in a row/column format; *i.e.* email files, text files, slideshows, pictures, videos, etc. [3]. The ultimate goal for this type of data is to process it to the point that the information can be examined using well established analytical techniques. Natural language processing provides a number of techniques that can be used for extracting information from text.

2.2 Natural Language Processing

Very simplistic natural language processing efforts began in the 1950's and produced the idea of using regular expressions to identify text by pattern. Modest improvements in the field were made throughout the next thirty years. It was not until the 1980's, when a fundamental reorientation in approach took place, that the field started to look as it does today. Much of the improvements were a result of employing machine learning methods to large annotated corpora [4].

The creation of a system capable of extracting information from human intelligence reports depends on being able to extract entities from the text. This is widely known as Named Entity Recognition (NER). Some commonly extracted entities are "PEOPLE", "LOCATIONS", "DATES", "ORGANIZATIONS", etc. For example, consider the sentence "John Smith lives in Miami." There are 2 named entities in this sentence, ("John Smith", "PERSON"), and ("Miami", "LOCATION").

In the next section, two NER model types will be discussed: feature-based sequence labeling models and rule based models. Bidirectional long-short term memory

neural networks are also known to be effective, but will not be discussed in this paper due to lack of a training set. Feature based models make use of feature functions which are derived from characteristics of words, sequences of words, and possibly entire documents or corpora in text. They leverage the idea of using context to more accurately tag named entities. Some examples are:

- Binary identifier of whether of not a word is capitalized [5]
- Length of word
- Part of speech of word
  - Whether a word is an adjective (JJ), noun (NN), verb (VB), etc.[6]
- Word shape
  - For a given word, capital letters are mapped to "X", lowercase to "x", numbers to "d", and punctuation is retained (eg."Washington, D.C." → "Xxxxxxxxxx, X.X.")
- Prefixes and suffixes of words
- Gazetteers
  - Lists of entities by type where the feature corresponds to a binary identifier of a word's presence within a list (eg. Boston, MA being found in a lexicon of locations would result in "1" for these words' presence in the list)
- Word embedding
  - Vector representation of a word in terms of a list of numbers
- Co-reference

– A feature(s) of the entire document or corpora that encodes some information about how entities that are the same be referenced in slightly different ways throughout text (eg. Lt John Smith, Lieutenant Smith, John Smith, Mr. Smith)

This is not an exhaustive list of features, but aims to provide a good representation of the kind of information used. Depending on which type of sequence labeling model is being used, features from other words in front of or behind the word being classified can be used to enrich the feature set further. For example, instead of merely trying to predict $word_i$ with the features listed above with respect to $word_i$, the feature set per word could be expanded to include things like the part-of-speech tag for the words before and after it ($word_{i-1}$ and $word_{i+1}$), as well as their shadows and word embedding vectors.

Two specific feature-based models will be discussed: Maximum Entropy Models (MEMs) and Conditional Random Fields (CRFs).

2.3 Models

MEMs, in general, are beneficial in situations where a sample space is present but there is no accompanying model [7]. To understand maximum entropy models in the context of NER, some exploration of their underlying principles will be conducted. First, note that in statistical modeling it is of great importance to avoid introducing bias. According to E.T. Jaynes in 1957, this is a problem for which information theory offers a compelling approach. The trade-off when coming up with probability assignments consists of avoiding bias while still utilizing all information available. Information theory gives rise to the fact that there exists a way to clearly quantify the "amount of uncertainty" that a discrete probability distribution models [8]. So entropy can be thought as "uncertainty." Jaynes argues that when dealing with only

partial information, it is necessary to use the probability distribution which has the maximum entropy subject to some distributional constraints that represent the information available [8]. Maximum entropy modeling then refers to finding a distribution for your data that starts out uniform, so as to assume nothing, and then is forced away from uniformity by information that is encoded into distributional constraints. Armed with a bit of intuition regarding the principles and motivation for MEM, we will now consider how these models work in practice.

The MEM applied to the task of prediction aims to come up with a way of estimating the conditional probability of some outcome y, a given member of some larger set Y, given some contextual information x; this is taken to be $p(y|x)$. To understand how this model is constructed, consider an example from NER. Imagine someone trying to identify how the phrase "Wall Street" is being used in a sentence. It can be referred to as either a "LOCATION" or an "ORGANIZATION". Collecting training data for this scenario would amount to collecting N samples where each sample contains the sentence with the phrase being used and the appropriate output. From the generalized definitions above, the sample would be N pairs of $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots (x_N, y_N)$. Empirically this training data is represented as $\hat{p}(x, y) \equiv \frac{1}{N} \cdot \#$ times the pair (x, y) appear together in training data [9].

From here, the goal is to come up with a model that represents what was observed in the sample. Feature functions bolster this model by allowing it to capture how contextual words affect the tagging of a phrase such as "Wall Street". For example, say it is observed that when the word "intersection" appears within a two word window of the phrase "Wall Street", there is probability 0.8 that the appropriate tag is "LOCATION". Utilizing feature functions allows more information to be leveraged in making predictions. Without feature functions, the tag predictions would be based solely on frequency as this is all that would be observed. According to maximum

entropy principles that is all that could be used. The following indicator function illustrates the example:

$$f_i(x,y) = \begin{cases} 1, & \text{if } y = \text{“LOCATION”} \text{ and the word “intersection” appears within} \\ & \text{a two word window} \\ 0, & \text{else} \end{cases} \tag{1}$$

Notice, it is a binary indicator function of the context x and outcome y. The feature function is incorporated into the model as a constraint. This is done by forcing $p(f)$, the expected value of f with respect to the model that is being built Equation (2), to be equal to $\hat{p}(f)$, the expected value of f with respect to the empirical distribution Equation (3).

$$p(f_i) = E[f_i] = \sum_{x,y} p(y|x)\hat{p}(x)f_i(x,y) \tag{2}$$

$$\hat{p}(f_i) = \hat{E}[f_i] = \sum_{x,y} \hat{p}(x,y)f_i(x,y) \tag{3}$$

The equality shown in Equation (4) represents the constraint. Intuitively this constraint is put in place to ensure that the distribution chosen accurately represents how often feature f_i is exhibited in the training data; the model is forced to mimic what is observed empirically in $\hat{p}(f_i)$ [9]. Constraints of this type are generated for each feature $f_i \; \forall \; i = 1, \ldots, N$.

$$\sum_{x,y} p(y|x)\hat{p}(x)f_i(x,y) = \sum_{x,y} \hat{p}(x,y)f_i(x,y) \tag{4}$$

Even with a set of N constraints in place, there are still an infinite number of

distributions, p_d, that could be fit. Let the set of all possible distributions that fit the constraints be D. The distribution chosen will be that which has the maximum entropy with respect to the constraints. In other words, the distribution which is maximally uncertain and most uniform with respect to the constraints [10]. The amount of entropy present in a distribution is given by $H(p_d)$.

$$H(p_d) \equiv -\sum_{x,y} p(y|x)\hat{p}(x)\log\left(p(y|x)\right) \tag{5}$$

The following argument represents the distribution that maximizes entropy

$$p_{maxent} = \underset{p_d \in D}{\operatorname{argmax}}\, H(p_d) \tag{6}$$

The optimization is done using Lagrange multipliers. For each f_i, a corresponding Lagrange multiplier λ_i is introduced as a weighting parameter for f_i. For further detail on the specific optimization methods used to compute the distribution, please consult Berger *et al.* [9]. Without getting more specific, it can be shown that there is a unique $p_d \in D$ such that p_d maximizes entropy. In general the entropy maximizing distribution is of the form [9]

$$p_{maxent}(y|x) = \frac{(\exp(\sum_i \lambda_i f_i(x,y))}{\sum_y (\exp(\sum_i \lambda_i f_i(x,y))} \tag{7}$$

The denominator is commonly referred to as $Z(x)$ as it normalizes the distribution [10].

Maximum entropy models alone are classifiers. To use them in sequence labeling problems, an adaption called the Maximum Entropy Markov model (MEMM) is used. This is because the Markov property, which says that state transition probabilities at a particular point in the sequence depend only on states within a small window, allows for tractable computations of the most likely sequence [11]. MEMMs rely on

the maximum entropy model to estimate the probabilities $p_{maxent}(y|x)$ from Equation (7), and then use a decoding algorithm, often the Viterbi [10] algorithm, to return the most likely sequence based on the probability estimates.

CRFs are a natural improvement on MEMMs for sequence labeling. CRFs take on almost the same form. However, they correct the *label bias* problem which can arise due to how MEMs are normalized. First, note the conditional distribution of $p(y|x)$ given by a general CRF [12].

$$p(y|x) = \frac{1}{Z(x)} \prod_{a=1}^{A} \exp\left(\sum_{k=1}^{K(A)} \theta_{ak} f_{ak}(\mathbf{y}_a, \mathbf{x}_a) \right) \tag{8}$$

The distribution takes a product over all A, which represent the set of factors present. This arises from the graphical representation this distribution can take and the fact that representing $p(y|x)$ as a factor graph manifests itself in this form as taking the product of all the factors present [12]. Just as with the maximum entropy formulation, the crux of this distribution is the feature functions f and their corresponding weights given in Equation (8) as $\boldsymbol{\theta}$. As previously mentioned, MEMs suffer from the label bias problem. It was termed the label bias problem by Lafferty *et al.* [13], and is described by them as follows:

Suppose the task at hand is to distinguish between the words "rib" and "rob", and that " r i b " is the sequence observed. From Figure 1 the same letter is observed from the starting state for both possibilities. So, each transition receives roughly equal weight. Since they cannot self transition and each have only one possible outgoing transition, their outgoing transition probabilities to states 2 and 5 are also roughly equal. Suppose that in the training data, state 1 observes "i" frequently, while state 4 has rarely seen it. Despite this fact, since there is only one outgoing transition for each, the observation of "i" is effectively irrelevant [13].

In essence, the label bias problem arises because states with few transition prob-

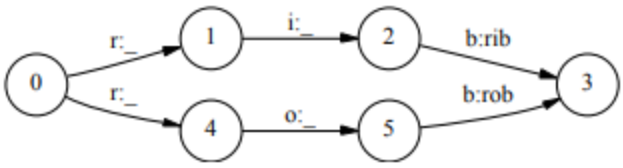


Figure 1. Label Bias Problem

abilities effectively ignore their observations. CRFs address this problem by using an observation dependent normalizing function $Z(x)$ that normalizes over the entire state sequence at once, allowing certain states to get more or less "vote" based on their actual observations [13]. The most likely tag sequence can then be decoded by Viterbi's algorithm based upon the state probability estimates from the CRF [13].

Rule based NER algorithms are exactly what one would expect. They are a set of rules constructed to identify named entities in text. They are often defined according to a domain specific problem and can consist of combinations of gazetteers, regular expressions, and any other rules or constraints that may apply [14]. Rule based systems, or a rule based/probabilistic hybrid are used far more often in industry than pure probabilistic models [15]. The costs associated with strictly defining the entity extraction problem in a mathematical sense, gathering vast amounts of annotated training data, engineering a useful feature set, computational time, and scalability contribute to the hesitancy of applying probabilistic models in industry [15]. In light of this, many entity extraction tools are built using a blend of probabilistic models such as MEMMs and CRFs and domain specific rules.

III. Data

3.1 Data Description

A total of 711 documents were provided for analysis. Each interview is in word document (.docx) form and contains the questions and responses asked by government officials at an immigration detention center. Each interview asks around 20 questions, with some containing slightly more and some slightly less. Ten questions were identified to be the questions whose responses the automation system would process. The specific questions chosen were those that seemed to consistently provide the most valuable information. Unfortunately, the questions are not always phrased in precisely the same way. So, some pre-processing was done to recover the most common form of the questions. Additionally, the word documents were not arranged in a consistent format in terms of questions and answers, with questions being numbered, answers being indented, etc. So, the meta structure of the documents was not a reliable way of locating specific responses. Also, once the interview moved passed the first couple questions, there was not a consistent ordering of questions. Therefore, the word documents had to be treated solely in terms of their text to identify which questions were being asked and identify their corresponding responses. Specific formatting throughout the document could not be leveraged to locate question responses. Uncovering the "common form" of each question was key to fixing this problem. This allowed the question text to serve as checkpoints throughout the documents, indicating where breaks are between questions and responses. It also allowed the information collected to be processed with respect to the appropriate question.

To identify the most common form of the questions, all entries in the documents matching the meta-data designation of "List Paragraph" were taken and then refined further to a collection of unique entries. This resulted in a set of 236 questions consist-

ing of numerous variations on the underlying basic questions. The cosine similarity between the strings was used to distill the list further resulting in a final set of 33 questions. Though only 20 or so questions were generally asked in each interview, towards the end of the interviews there were occasionally a couple of differing questions that were asked. This caused the total set to be 33. From this set, the ten questions around which the automation system was built were selected. The questions are given in Table 1, and the code used to extract them is shown in Appendix A.1 VI.1

Question Number	Question
1. (Bio)	What is your full name/date of birth/place of birth/nationality?
2. (Family)	What is your fathers/mothers full name; siblings?
3. (Imm. Family)	Are you married? What is the name of your spouse? Do you have any children, if so, how many?
4.(Journey)	Describe your journey to this point with dates/ countries visited/and others who joined you.
5. (Gang/mil.)	Have you ever served in the military in any country or been part of a gang in any country?
6.(Fam. gang/mil.)	Do you have relatives serving or having served in the military or a gang?
7.(Forger)	What identification documents did you depart your home country with? What identification documents do you have in your possession? If fraudulent, where/when/from who did you purchase them?
8. (Help to MX)	Describe the individuals who helped you reach Mexico.
9. (Smuggler)	Agent/Smuggler information?
10.(Destination)	What is your destination country/state/city? Why are you attempting to get to this country/state/city? Do you have relatives/friend in this country/state/city?

Table 1. Questions which automation system is built around.

Within the responses to these questions it was common for there to be additional information given than what is explicitly asked in the question. Common examples are Questions 1 and 2. Question 1 asks "What is your full name/date of birth/-place of birth/nationality?". The response to this question sometimes includes the interviewee's email address, passport #, height and weight, and telephone number in addition to the information asked. Likewise, while Question 2 only asks for the names of family members, their ages and locations are occasionally reported as well. The

specific variables that were chosen for extraction within each question, if available in the interview, are listed in Table 2.

Question Number	Variables
1.	facebook account, email, telephone # , height (in.)/weight, passport #, FINS #, full name, date of birth, place of birth
2.	names of father, mother, brother, sister, and siblings (generic) as well as whether they are deceased, age, location, and telephone #
3.	names of spouse, son, daughter, children (generic) as well as whether they are deceased, age, location, and telephone #
4.	locations the interviewee was along their journey to Mexico and corresponding dates
5.	whether or not interviewee served in the military (air force, navy, army), or police, or have gang ties (MS-13)
6.	same information as in 5 but in relation to relatives (mother, father, brother, daughter, son, sibling (generic), cousin, niece, nephew, relative (generic), spouse, children (generic))
7.	forger names, where and when the interviewee had contact with the forger, and the forger's nationality
8.	same information as in 7 but applied to any person who helped the interviewee reach Mexico
9.	same information as 7 and 8 applied to smuggler/agent
10.	location of final destination in journey

Table 2. Variables collected within each question.

Finally, the 711 documents provided were examined. Of those, 650 were retained for processing. The 61 were left out for a variety of reasons. Either they were duplicates, were not actually interviews, or were formatted differently enough from the rest that the system would not be able to process them. The 650 constitutes a very small dataset. A much larger dataset would make for more interesting analysis, however the current sample is enough for illustrative purposes.

3.2 Assumptions

It was assumed that dishonest answers may be given during interviews. As is discussed further in the next chapter, the database is designed to detect anomalies

that may arise because of dishonesty within the information extracted. Furthermore, it is assumed that the interviewer may misspell words within an interview. In light of this, the responses are processed exactly according to what is transcribed by the interviewer.

IV. Methodology

4.1 Introduction

The granular level of information that the system needs to be able to extract is not within the realm of what can be found in traditional NER software alone. From Table 2, notice that most of the variables are themselves named entities but they require context to give them meaning. For example, if the response from Question 2 was processed with only a named entity recognizer it would generate a list of names, but they would not have a corresponding relationship type. It would be unclear who is the interviewee's brother, as opposed to his father, etc. This prompted the construction of a rules based routine to be used in conjunction with traditional named entity recognition software. Very generally, the rules based routine takes the output from named entity recognition software and, using contextual words, maps it to the appropriate variable.

4.2 Software

The software used for the named entity recognition portion of the automation system was StanfordCoreNLP [16]. StanfordCoreNLP is a Java based software program that can be used in Python with a wrapper. Specifically, their "NERClassifierCombiner" was used, which is an annotator that applies several named entity recognizers in conjunction to identify named entities. According to Finkel *et al.* [11], the underlying statistical model for estimating the outcome probabilities is a CRF. However, unlike what was discussed in Chapter 2, the Markov assumption is relaxed. This was done so that non-local dependencies between states could be accounted for [11]. Because of this, the Viterbi algorithm was replaced with Gibbs sampling as the method for inferring the most likely tag sequence [11]. Even though Gibbs sampling does

not result in a deterministic solution for the most likely tag sequence, running it enough times can get to the same result [11]. Furthermore, Finkel *et al.* [11] noted a 1.3% increase in F_1 accuracy by allowing non-local dependencies between states to be modeled as compared to the previously used CRF. F_1 accuracy is a model performance metric that takes into account both recall, the completeness or accuracy of positive examples, and precision, how many are truly positive out of positively labeled examples [17].

Within the NERClassifierCombiner, the default annotator recognizes (PERSON, LOCATION, ORGANIZATION, MISC, MONEY, NUMBER, ORDINAL, PERCENT, DATE, TIME, DURATION, SET). When including a rule based annotator they call "regexner", functionality for the extraction of (EMAIL, URL, CITY, STATE_OR_PROVINCE, COUNTRY, NATIONALITY, RELIGION, (job) TITLE, IDEOLOGY, CRIMINAL_CHARGE, CAUSE_OF_DEATH) is added. In addition to named entity recognition, StanfordCoreNLP can do POS tagging, word tokenization, parsing, sentiment analysis, regex, and other natural language processing tasks. As was alluded to, Python was the programming language used for the creation of the entire automation system.

StanfordCoreNLP was chosen over other NER software primarily because of its ease of use within the Python programming language as well as its performance in preliminary investigations with the interviews. Furthermore, Stanford has a well known group of researchers focusing on cutting edge NLP techniques.

4.3 Rules Based Routine

There are many different approaches that could be taken to construct the rules. This system was coded to process one question's response at a time. To process the response for a specific question, the document is read line by line comparing each line

to the question of interest. Treating both the given line and the question as strings, if the cosine similarity between the two is above a certain threshold, each subsequent line is stored as the response for that question until there is another match with a question from the master set of 33 questions referenced in Chapter 3. This would indicate that all of the response is collected since a new question is encountered.

The code for any given response extraction differs based on what is being extracted, but the general process involves applying the NERClassifierCombiner, to the response and then passing that output through various rules and conditions that identify which variables from Table 2 the named entities belong to.

To give a general synopsis of the process, take Question 2's response. Question 2 asks "What is your fathers/mothers full name; siblings?" and the variables are name, whether or not they are alive, age, and location. To begin, there are empty lists created to house the information connected to each of the possible relationship types: father, mother, brother, sister, and siblings (generic). So for each relationship type there is a list for name, age, location and deceased (or not). A custom chunker is then created to break up the response at the mention of unique relationship types, or at the end of the sentence. The goal is to obtain chunks for which all of the entities in that chunk relate only to one relationship type. The chunks are then processed further so that the right information is attributed to the appropriate individual. For example, imagine a chunk containing the following phrase (with fictitious names): "brothers are John Smith (26 yoa, Lima, Peru) and Henry Smith (30 yoa).". There are 5 pieces of information that should be mapped to variables; **John Smith** (name), **26** (John's age), **Lima, Peru** (John's location), **Henry Smith** (name), and **30** (Henry's age). An assumption was made to subdivide the chunks even further by occurrence of names and periods and any entities encountered between names are attributed to the preceding name. Under this schema, all of the information is attributed to the

appropriate person.

Each specific response differs to some extent in the approach taken, presenting unique challenges in the logic that was employed. Creating custom chunkers, as mentioned above, was one of the common tactics used to isolate specific variables of interest. Care was also taken in trying to ensure a balance between the rigidity and flexibility of the code. The rules employed must be specific enough to discriminate between the variables it is searching for and other words that may not be of interest, while also being flexible to enough to ensure that information is actually extracted. This is a challenge inherent to rules based systems, especially applied to documents that exhibit significant variation in style of transcription. Differences in style, for example an interview referring to someones age by "yoa" when the code may be searching for the key word "age", were major factors in finding the specific vs. general code balance.

4.4 Extraction Code

For each question listed in Table 1, a brief overview of how the variables were extracted is outlined below, as well as references for where the corresponding code can be found in the Appendix. The code for response extraction is the same for every question. So, only the code for the first response extraction is provided. It can be found in Appendix A.2 VI.2. There were two main formats for the interviews. The most commonly encountered format, with questions and responses, was what the system was primarily coded for. However, there were a handful that did not follow the question/response format. Instead, they contained some of the variables from Table 2 at the top of the interview, and the rest was free form. The code was modified slightly to be able to handle that case by implementing a check at the beginning to determine if any of the lines in the interview matched a question in the question

bank. If there was a match, it proceeded as normal. If not, the system processed the interview slightly differently. Essentially, it treated the entire interview as one response. The code designed for the normal format case was modified slightly and joined together to extract whichever variables were present. Since it is very similar to the code for processing the normal interview format, it will not be discussed further or referenced in the Appendix. Additionally, at the end of the process the variables were reformatted slightly for input to the SQL database. The specifics of that code will not be discussed.

Question 1 (Bio): Code for the response is given in Appendix A.3 VI.3. First, find and replace operations are executed to replace various forms of words such as "Facebook", "FACEBOOK", "facebook", etc., with a common form, "facebook". This is routinely done throughout all of the processing since words like "facebook", "email", etc., are used as trigger words for the system so that it knows where to look for the variables being extracted. It is easier to have all possible variations of the trigger words converted to a common format. The entire response is then broken down into chunks based on what variables are being searched for; a passport chunk, FINS chunk, height/weight chunk, etc. Each chunk is then dealt with separately since the process for extracting something based on dates, such as date of birth, relies on locating "DATE" entities, whereas extracting country of birth requires "LOCATION" entities. Since entities often are composed of multiple words, "Houston, Texas" for example, there must be a way to ensure that "Houston, Texas" is counted as the same entity. This is also the case when extracting dates and names. To address this, counters were introduced to the processing of each chunk. The counter is initialized upon the first encounter of an entity. As long as consecutive iterations through the words in the chunk continue to be classified as that entity or appropriate

punctuation such as commas between "LOCATION" entities, the words were considered part of that entity. Once a word is encountered that does not fit that pattern, the counter is rewritten to zero, and the words are joined together to be stored as one entity. This approach is used throughout all question processing. During processing of the first question's response, a list is created that stores variations of the subject's name. This is done because throughout the rest of the interview, it is common for the person transcribing the interview to refer back to the subject using some variation of their full name. For example, if the subject's name is "John Fernando Garcia-Smith", he may be referred to throughout the rest of the interview as "Garcia-Smith", or "Smith". As names are extracted throughout the rest of the interview, they are compared to the name variations stored so that the subject's name is not accidentally stored when it should not be.

Question 2 (Family): Code for this response is given in Appendix A.3 VI.4. The response for this question was already briefly discussed in the previous section so further elaboration will be minimal. As with the processing in Question 1, there are some initial find and replaces done. Chunking is then performed to isolate the sections of the response that deal with the possible family relation types listed in Table 2. Within each family relation chunk the same process as was outlined for Question 1 was followed. The words in the chunk are iterated through and the entities are grouped based on counter value.

Question 3 (Imm. Family): The code for this response is not explicitly given since it is practically identical to Question 2's code. Note from Table 2 that the exact same things are extracted from Questions 2 and 3, it is just done for different family members.

Question 4 (Journey): The code for this is response is given in Appendix A.3 VI.5. This response is first broken down by sentence. The reason is that the system attempts to extract places and dates for where the subject was and when they were there, along their journey to Mexico. The assumption is made that information relating to a certain place and date are contained within one sentence. For each chunk/sentence it is determined whether a location or a date is stated first. Then, as was done in all previous processing, the location and date entities are grouped together. Depending on which was stated first, location or date, as well as how many of each entity type are observed, the chunk is processed according to a particular pattern. Differing patterns are established because of the assumptions that go along with how the locations/dates were reported, and whether they have corresponding entries. For example, if the entities are reported in the order location-date-location and there is not a second date reported, it was assumed that the date given corresponds to the first location. An entry of "no corresponding date found" is then added to the date list so that as further sentences are processed, and the list grows, there can be a clear accounting of which locations and dates are connected. There are a number of different patterns that can be observed so a majority of the code for processing Question 4 deals with appropriately extracting paired locations and dates based on the patterns observed.

Question 5 (Gang/mil.): The code for this response is given in Appendix A.3 VI.6. This response is relatively simple to process. The response is first broken down by word. The assumption is made that if the words "no", "not", "N/A", "none", or "None" are found, it is determined that the subject is not in the military, police, or a gang. If none of those words are found, then the words "military" (and specific branch names), "police", "MS-13" (and variations), and "gang"

are sought out. If the response contains one of those words then it is assumed the interviewee belongs to the corresponding category. If not, they are identified as not having ties.

Question 6 (Fam. gang/mil.): The code for this response is given in Appendix A.3 VI.7. The specifics of the code are not discussed since it is exactly the same logic as for Question 5, just applied to multiple family members.

Question 7 (Forger): The code for this response is given in Appendix A.3 VI.8. The approach for processing Question 7, as well as Questions 8 and 9, is rather similar to Question 4's code. To begin, the entire response is broken down by sentence since the assumption is made that the information relating to a forger will be primarily given within a sentence. Additionally, within each chunk/sentence the assumption is made that once a name is encountered, any nationality, location, or date that comes after that name and before either the end of the sentence, or another name, is attributed to the first name. Also similar to Question 4, placeholder values are inserted for when forger names are extracted but the other variables for that person (nationality, location, date) cannot be found. This is done so that each list is of equal length and the correct location, date, and nationality, or placeholder for those values, is attributed to the correct person. Since the code for Questions 8 and 9 is practically identical to that of Question 7, it is not discussed explicitly and not referenced in the Appendix.

Question 10 (Destination): The code for this response is given in Appendix A.3 VI.9. The processing of this question is very straightforward. The only task is extraction of the subject's destination. This amounts to simply using a counter to group together location entities.

4.5 SQL

As the autonomous system processes the variables listed in Table 2, the information is stored in an SQL database. In the words of I.R. Mansuri and S. Sarawagi in *Integrating Unstructured Data into Relational Databases*, "Database systems are islands of structure in a sea of unstructured data sources" [18]. Without something to store the information processed over time, the system would be ineffective. In addition to providing efficient storage of data, SQL also has a powerful built-in querying language and the ability to be used in conjunction with general programming languages, such as Python [19].

The database was built with two goals in mind: ability to easily identify circumstances when interviewees may not be telling the truth, and efficient storage of data. For these reasons many of the variables being processed are stored in tables designed for one-to-many relationships. For example, the passport variable is accounted for with the use of three tables. First, there is a table titled "passport_table", which has one column that is a unique ID for each passport and another that contains the passport number. The unique ID, called "passportID", is the primary key of the table and there is a unique constraint put on the passport number. This structure forces each row in the table to be unique. The second table is the "personID_table". Similar to the "passport_table", this table has a unique ID, called "personID", associated with each person in the database. The primary key for this table is "personID". To tie these together, there is a third table called "passport_link_table" that tracks who the passports belong to. This table is composed of all foreign keys. The first foreign key is the "personID" column from the "personID_table", and the second foreign key is the "passportID" from the "passport_table". Together, these columns form a composite primary key for the "passport_link_table" to ensure that each row is unique. By recording each passport number only once and then using foreign keys as links

between people and passports, the database remains efficient.

Assuming some interviewees may give false information, this database structure made the most sense. For example, under the devised database schema, if multiple bad actors are interviewed and all give the same fictitious passport number, each person would be linked to the same "passportID" via an arc. By contrast, if the database simply had one table to track passports, the same passport number would be repeatedly entered. This type of storage is less efficient. In addition, the devised schema would allow such inconsistencies to be identified since anytime the same "passportID" appears more than once in the "passport_link_table", it raises red flags.

A number of other variables listed in Table 2 including Facebook, telephone, FINS, email, locations, gang/military connections, and relationships are all structured in roughly the same way. The database schema, showing how all the tables are connected via foreign key arcs, is given in Figure 2.

The database was created using SQLite since SQLite can be used entirely within the Python programming language [20]. This allows both the document processing and variable entry into SQL all from one place. In practice, each interview is both processed and integrated into the database one interview at a time.

Some individuals could appear in more than one interview. Ideally, they would be recognized as the same person. However, under the described method of processing and integrating the interviews, the individual would be counted twice. It was initially thought that this issue could be resolved entirely autonomously based on whether a name appeared exactly the same way in multiple documents where contextual information, such as mutual family members, agreed between documents. It was noted however, that if an individual is the subject of an interview, they normally provide their full name. However, if they are providing the names of their family members, they may only give first and last names. Such differences make name matches hard to

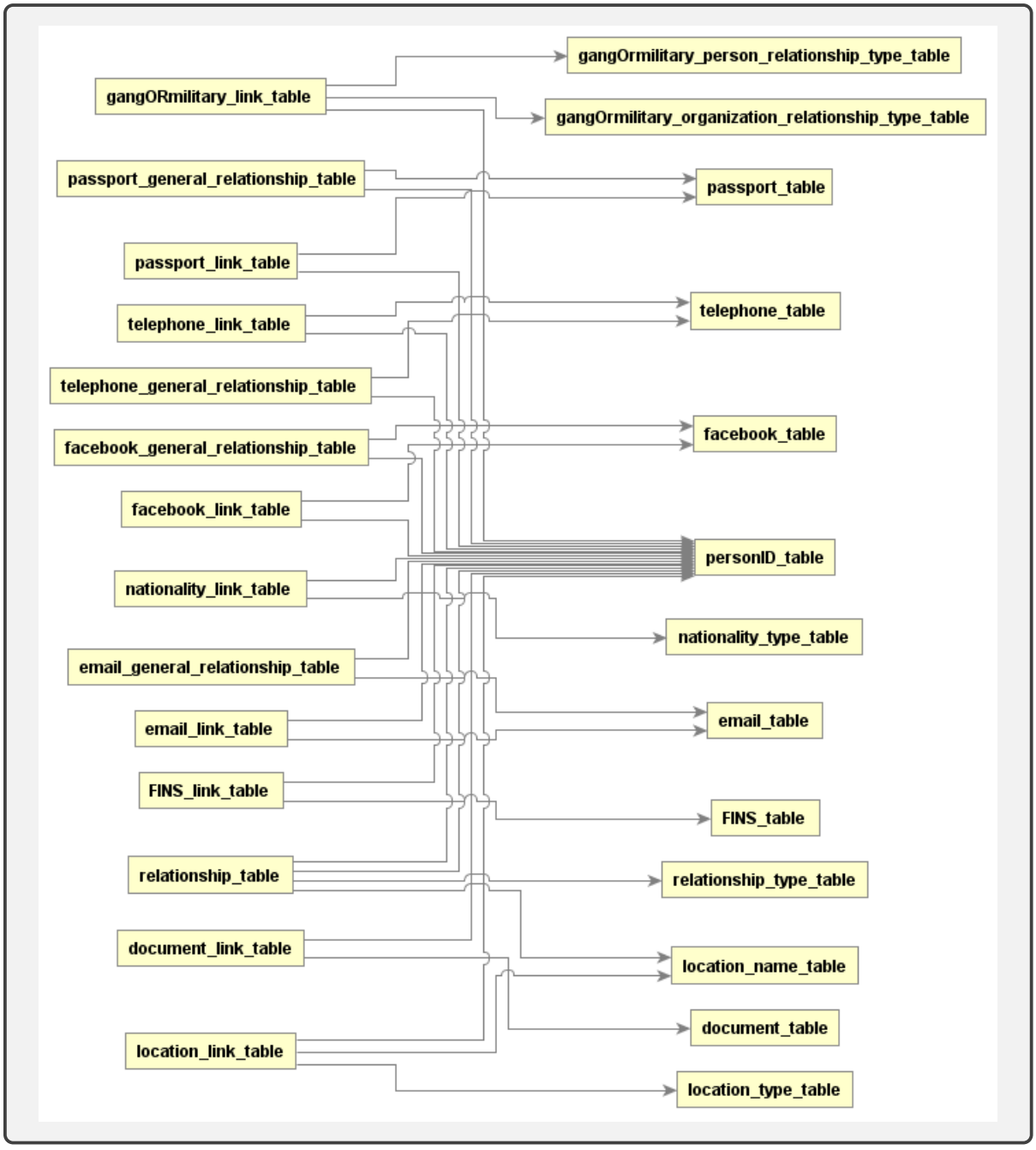


Figure 2. Database Schema

find. If the criteria is lowered to accommodate the fact that the same individual may be referred to in different documents with varying lengths of their name, it is highly likely that an autonomous system would identify two different individuals as the same person. For example, imagine an interviewee named "John Michael Smith", and in his interview he provides his full name. John's brother is also interviewed and mentions that he has a brother who he refers to as "John Smith". It is not unreasonable to assume that there are many other people who also have the name "John Smith", and are referred to as such in interviews. So, if it is decided that two individuals will be counted as the same person if their names are the same to a certain degree, there will almost certainly be errors made.

To remedy this, a graphical user interface (GUI) was built using Python's "tkinter" package to allow for a human-in-the-loop type approach [21]. The initial approach described above forms the basis for how the GUI was constructed. Each time a name is ready to be written to the database, the database is queried to determine if the name in consideration is similar enough to any names already in the database. If it is, a window is displayed to the user so that a human can determine if the individuals in question truly are the same person. To assist the user, the window displays the name that is currently about to be inserted into the database as well as a link to pull up the document it is mentioned in, as well as the other names already in the database that it is similar to and corresponding links to those documents. The names appear in descending order of similarity so that if there happens to be a name for which there are many similar existing names, the user can quickly identify which documents and names mostly likely contain a match. This opportunity was also taken to allow the user to add comments about the person in question, make corrections to the name extracted, as well as decide that the current name should not be saved in case the system has extracted something that is not a name. Figure 3 provides an example of

the GUI, with actual names blotted out in white. The code for the GUI construction is given in Appendix A.4 VI.10, and the SQL database entry code is in Appendix A.5 VI.11.

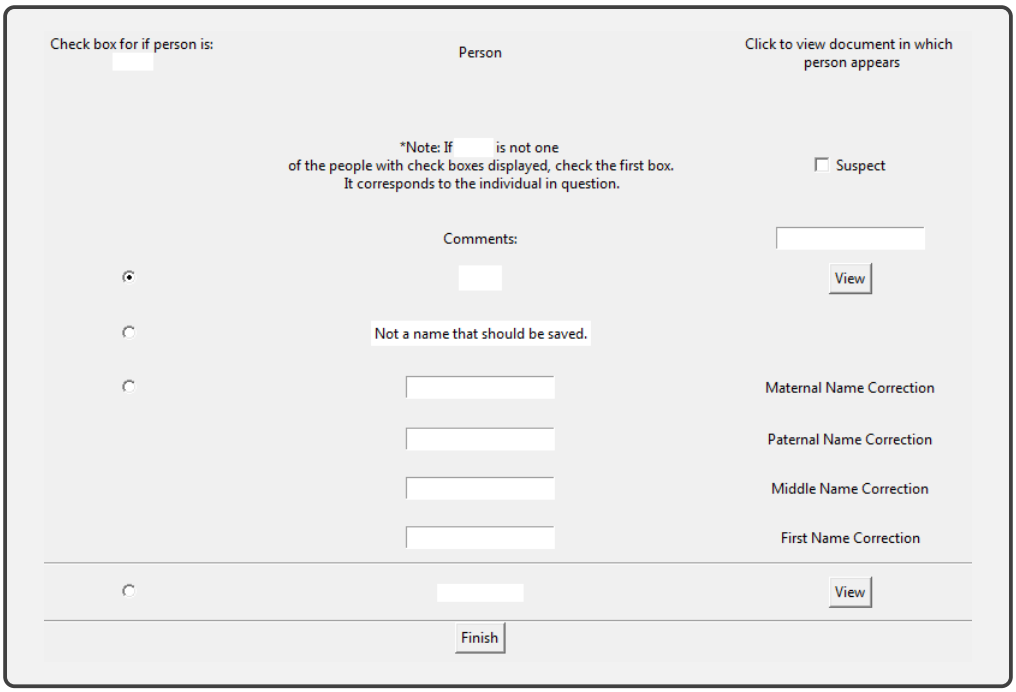


Figure 3. GUI Example

V. Analysis

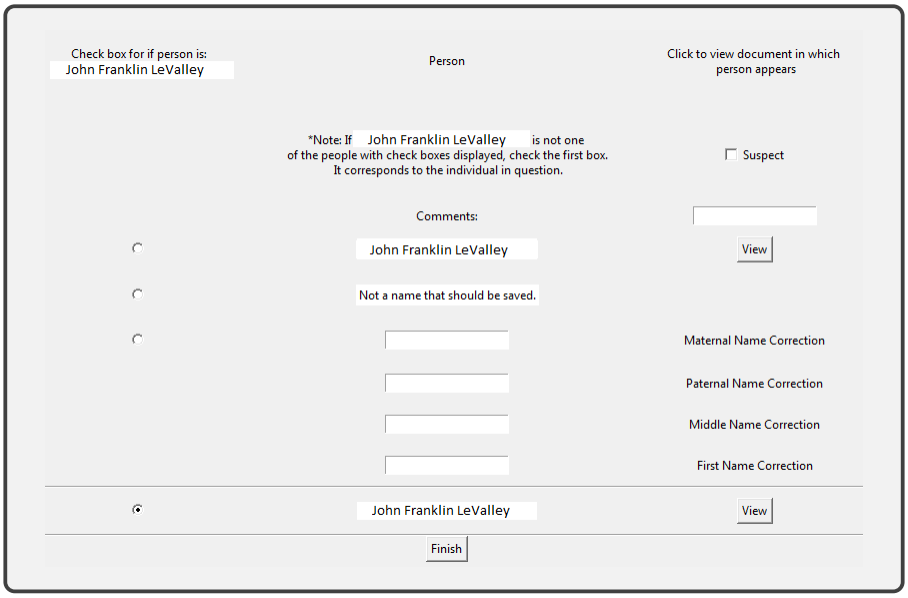
5.1 Introduction

Since this research is primarily a proof of concept, there are not results in the traditional sense. Instead this chapter will highlight the successes of the GUI's human in the loop approach, as well as some analysis of the final database to demonstrate what may be of interest to intelligence analysts.

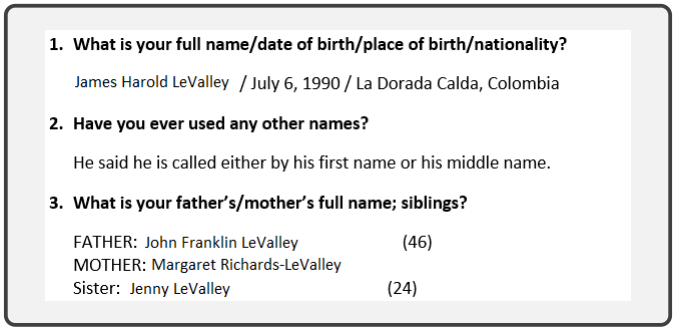
5.2 Analysis

The GUI was effective in prompting the user to discriminate between people. A couple of instances in which it was used were recorded to present as examples. The first example occurred when a father and son were both interviewed. The GUI for this specific instance, as well as the two documents it had links to are given in Figure 4. The true names have been replaced with fictitious ones.

The GUI indicates that the system was about to input the name *John Franklin LeValley*, but that the name already appears in the database. The button titled **View** towards the top right provides a link to the document that is currently being processed, an interview with *James Harold LeValley* (Figure 4b) . The **View** button at the bottom provides a link to the document in which *John Franklin LeValley* already appeared, which was *John Franklin LeValley*'s interview (Figure 4c). Generally, context is the only way to make the determination of whether or not the name mentioned in one interview is the same as the name mentioned in another. In this case, since from Figure 4b it is seen that *John Franklin LeValley* is *James Harold LeValley*'s father and *Margaret Richards-LeValley* is his mother, and in Figure 4c *John Franklin LeValley* identifies one of his sons as *James Harold LeValley* and his wife as *Margaret LeValley*, it can be determined that the same *John Franklin LeValley* is present in



(a) GUI showing person under consideration.



1. **What is your full name/date of birth/place of birth/nationality?**

   James Harold LeValley / July 6, 1990 / La Dorada Calda, Colombia

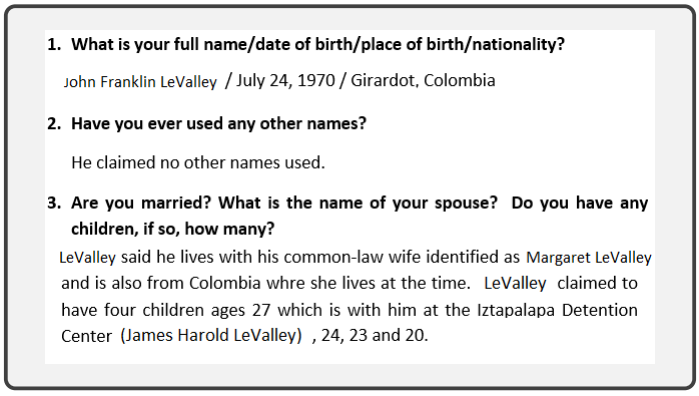
2. **Have you ever used any other names?**

   He said he is called either by his first name or his middle name.

3. **What is your father's/mother's full name; siblings?**

   FATHER: John Franklin LeValley (46)
   MOTHER: Margaret Richards-LeValley
   Sister: Jenny LeValley (24)

(b) Interview of LeValley's son.



1. **What is your full name/date of birth/place of birth/nationality?**

   John Franklin LeValley / July 24, 1970 / Girardot, Colombia

2. **Have you ever used any other names?**

   He claimed no other names used.

3. **Are you married? What is the name of your spouse? Do you have any children, if so, how many?**

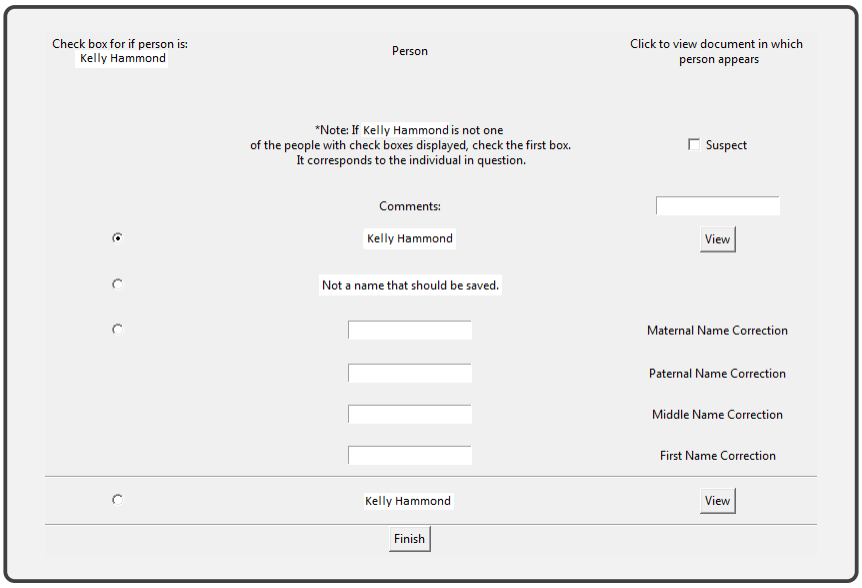
   LeValley said he lives with his common-law wife identified as Margaret LeValley and is also from Colombia whre she lives at the time. LeValley claimed to have four children ages 27 which is with him at the Iztapalapa Detention Center (James Harold LeValley) , 24, 23 and 20.

(c) Interview of LeValley.

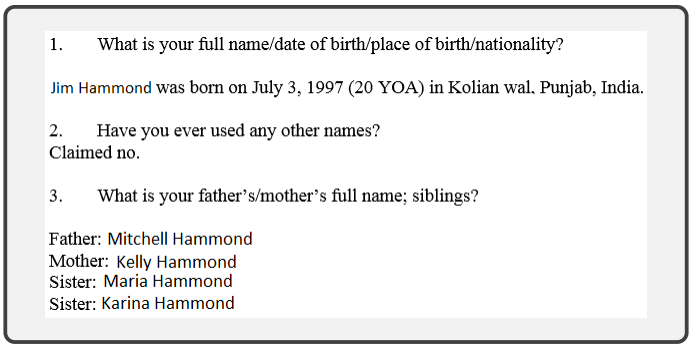
Figure 4. Demonstration of what the GUI (a) presents to the user and what information decisions are based on, (b) and (c).

both cases.

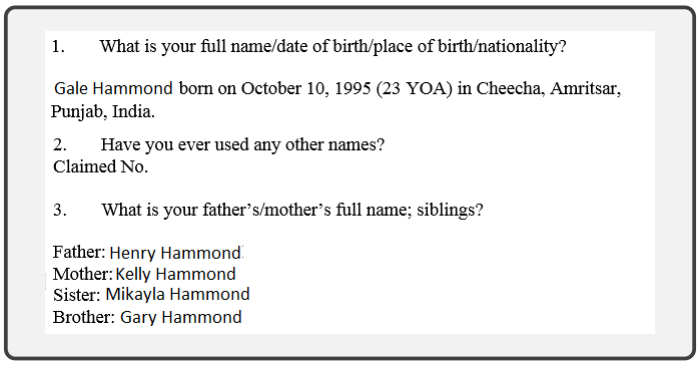
Figure 5, also with fictitious names, depicts an instance where two people have the same exact name but it is determined they are not the same person. Figures 5b and 5c contain portions of interviews that were conducted by sons of *Kelly Hammond*. For it to be the same *Kelly Hammond* in both interviews, one would expect there to be overlap in the other siblings mentioned as well as the same father being mentioned in both places. The interviewees' places of birth could also be compared. These comparisons show that there is no overlap in the family names other than the mother, *Kelly Hammond*. Additionally, while both interviewees are from India, the specific locations are different. It is concluded that mere coincidence is responsible for the two individuals sharing the name *Kelly Hammond*. It is also worth noting that there tended to be many Indian individuals with the same name. Specifically the names *Kaur* and *Singh*, which seem to be very commonly included in female and male names respectively.



(a) GUI showing person under consideration.



(b) First interview containing name *Kelly Hammond*.



(c) Second interview containing name *Kelly Hammond*.

Figure 5. GUI demonstration of instance where person being considered is not the same person in the interviews from (b) and (c).

Ultimately this system would be used to aggregate data over time, allowing unstructured text to be stored in an efficient database for easier analysis. The analysis becomes more interesting and presumably more useful as data is accumulated and processed over time. Due to the relatively small dataset used for building and testing the system, there are not any specific "results". Instead, a couple of examples showing what intelligence analysts could do with the database are provided. The first example provides a visualization of the network that represents individuals who are connected with either an agent/smuggler, someone who was involved in providing them with fraudulent paperwork, or someone who helped them reach Mexico. The second example provides an accounting of all the people who appear in more than one interview and who mentioned them.

Example 1.

Drawing upon what was discussed in Chapter 1 of this paper, various government agencies are interested in tracking the unauthorized entry of people into the United States especially as it relates to smuggling. This information is explored in two different ways to demonstrate the effect of different GUI parameter settings. First, the data is looked at exactly as it was processed by the system. A query is executed directly on the database's relationships table. Figure 6 depicts the schema for this table. It is comprised of all foreign keys: two **personID**'s, one **locationNameID** and one **relationshipTypeID**. The goal is to retrieve all unique pairings of two people where the relationship between them is either agent/smuggler, forger, or someone who helped them get to Mexico. The columns of interest for this query are **personID1**, **personID2** and a **relationshipTypeID**. The **locationTypeID** will not be used in this query.

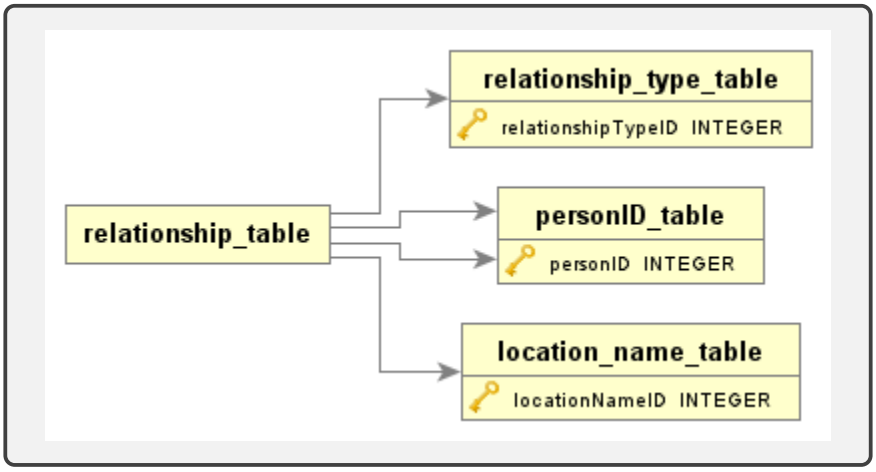


Figure 6. Relationships table schema.

The results of the query can be visualized as a network via Python's built-in "Networkx" package [22]. It is shown in Figure 7. This network is very unconnected. The reasons for this are two-fold. First, the dataset is extremely small. With only 650 interviews being processed, bad actors are not very prevalent. The second reason

is a result the settings chosen for the GUI during processing. The GUI was set to detect matches in at least two of the four names a person could possess: paternal last name, maternal last name, middle name, and first name. For most names that an interviewee mentions, such as family member names, this matching criteria resulted in the GUI being activated. However, names that were mentioned in response to Questions 7, 8, and 9, relating to suspected criminals, were often only one word long so they never met the match criteria. Therefore, many of the names pertaining to suspected criminals got their own person ID's irrespective of that name appearing in a different interview. So, the network shown in Figure 7 is not very connected since it is based on person ID's. From an intelligence perspective, storing criminal names in this way is not necessarily a bad thing. For example, intelligence analysts are aware of multiple instances where different criminals acting in a smuggler capacity are referred to by the same alias. In this case, it could be beneficial to account for these individuals with different person ID's so that all information pertaining to them is also kept separate.

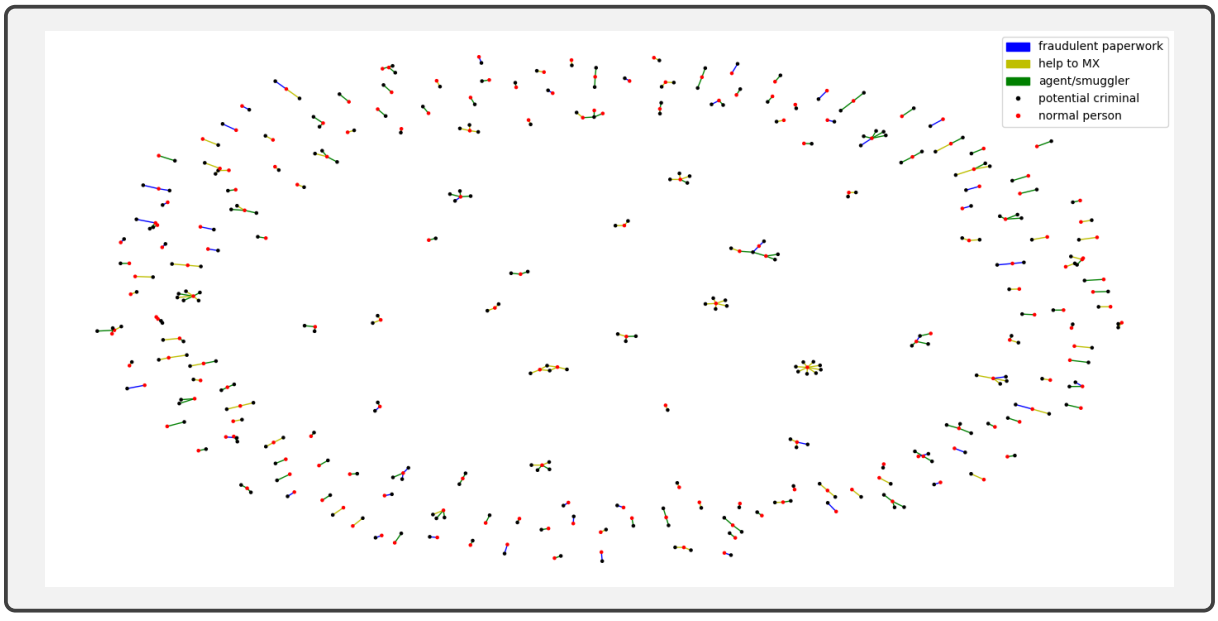
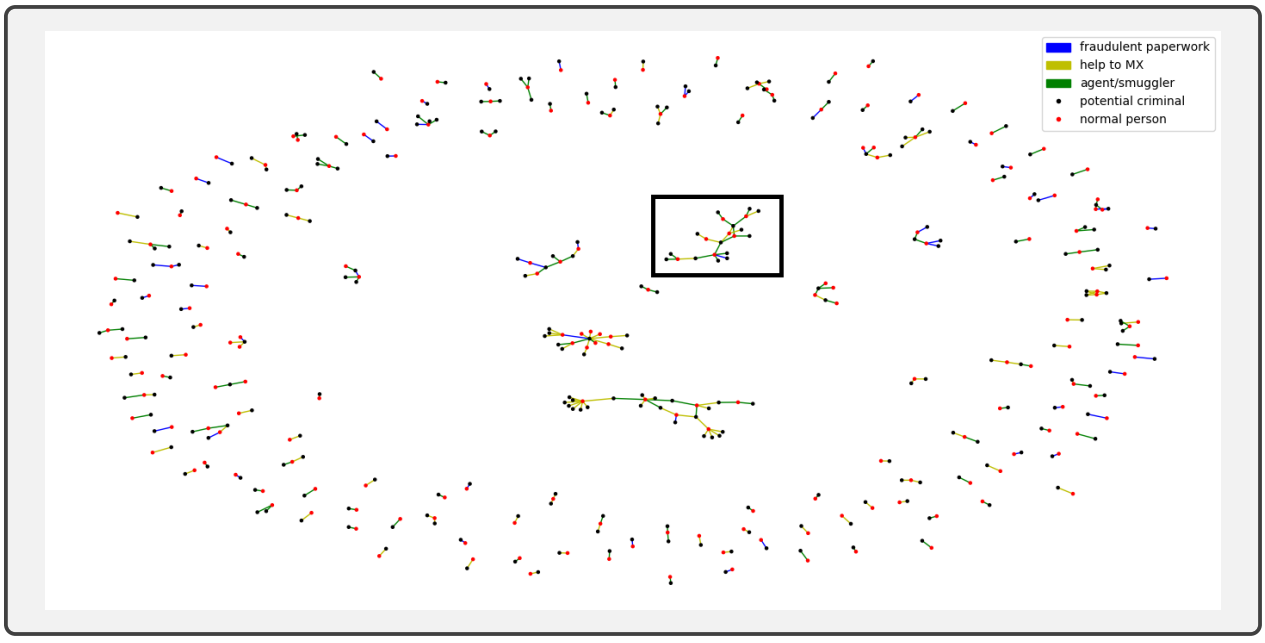


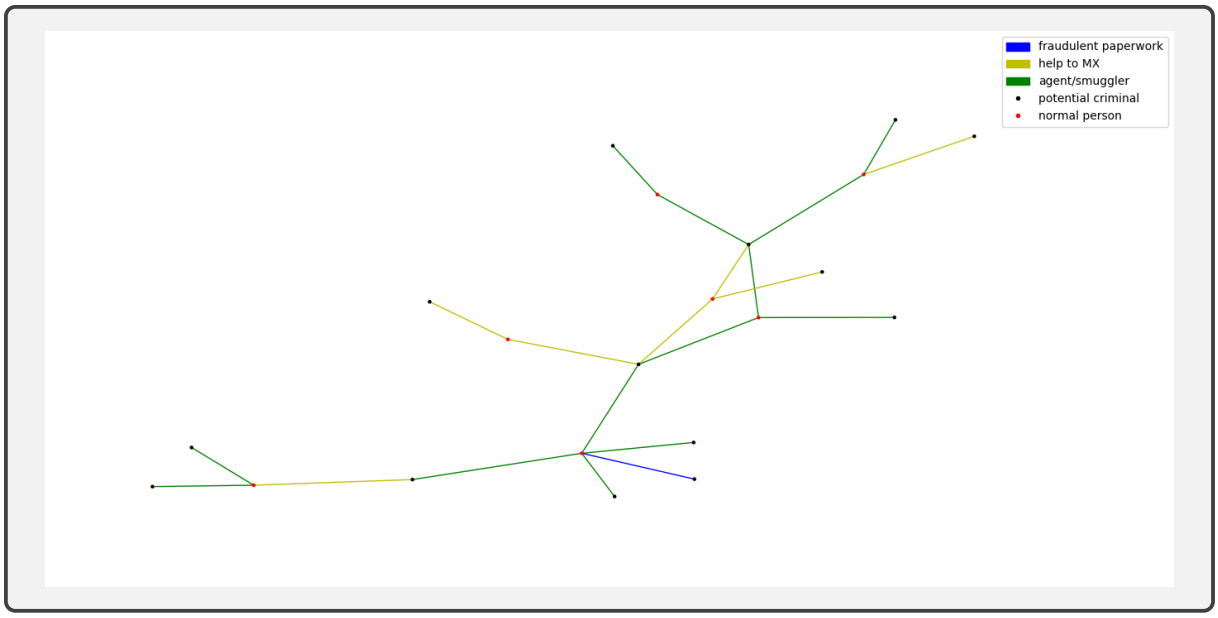
Figure 7. Relationships table based on agent/smuggler, forger, etc. person ID's.

This information can be examined a second way to demonstrate what the network may show under different parameter settings. Specifically, how may the network look if the GUI parameters are changed so that people with the same one word name are considered the same person? This effect was achieved by re-querying the person ID table for the names associated with the agents/smugglers, forgers, and individuals who helped people to Mexico. The network was then recreated based on names instead of person ID's.

The adjustment made in basing the network off of names yields interesting results. Specifically, portions of the network become far more connected. In order to protect the anonymity of the individuals from the database, the network nodes do not have associated names. Instead, as indicated in the legend, they are categorized as either a traveler (red), or agent, smuggler, forger, etc. (black). Likewise, the nature of the relationship between two people is represented by the three different edge colors also shown in the legend.



(a) Relationships table based on agent/smuggler, forger, etc. names.



(b) Boxed portion of network.

Figure 8. Relationships table based on agent/smuggler, forger, etc. names.

Figure 8b shows two specific bad actors who have four connections each. These two nodes each have more connections than anything from Figure 7. In addition to being more connected, this network can help intelligence analysts further inspect the nature of relationships between people. For example, there are a number of instances where a red node and black node are connected via an agent/smuggler edge and the same black node is connected to a different red node via a different type of edge. This hints at the fact that maybe both red nodes were relying on the black node for the same type of service, or that the black node is acting in multiple capacities.

Ultimately, the differences in Figures 7 and 8a highlight the flexibility this system provides. The analyst can change the GUI parameters to fit their needs as well as use SQLite on the back end to achieve similar effects.

Example 2.

The second example provides a visual of what was accomplished through the GUI. Figure 9 shows the schema of the primary table of interest for this query. The database was queried to return all the individuals who appear in more than more than one document. The data returned was made into a network shown in Figure 11. The nodes in this network represent individuals in the database and are color coded similarly to Figures 7 and 8. Red nodes represent someone not presumed to be criminal, and black nodes indicate the person is a smuggler/agent, forger, or someone who is helping an individual travel to Mexico. The edges represent the type of relationship between people. Green edges correspond to a general relationship, meaning that one of the individuals mentioned the other in an interview but the exact nature of their relationship could not be extracted. Blue edges correspond to an agent/smuggler, forger, or someone aiding in travel to Mexico relationship. Yellow edges indicate the two people are family members.

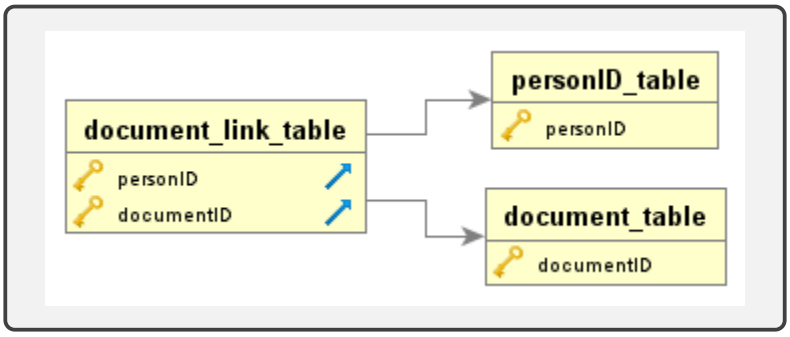
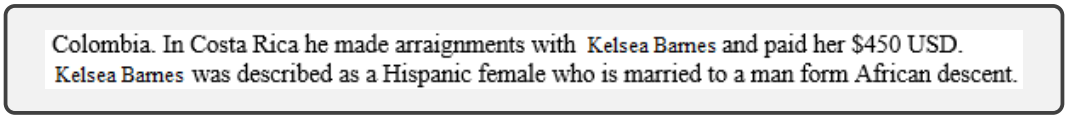


Figure 9. Schema for the document link table.

As was the case with the network from Example 1, the network in Figure 11 is not very connected. This is again most likely due to the small number of documents processed. Despite this, there were still a number of clusters that could be of interest. Two such clusters, those from Figures 11b and 11c, will be examined further.

Figure 11b contains three nodes, two corresponding to unsuspicious people and

one corresponding to a suspected criminal. For the sake of deeper exploration, the interviews having to do with the people involved in this cluster were examined. The relevant portions are shown in Figure 10. The true name of the smuggler being mentioned was replaced with the ficititious name *Kelsea Barnes*. The top left red node from Figure 11b corresponds to the interview snippet for *Person B* from Figure 10b and the bottom right red node corresponds to *Person A*'s interview snippet shown in Figure 10a. In truth, both individuals should be connected to *Kelsea Barnes* with blue edges since she acted as a smuggler for both people, but *Person B* did not mention her in one of the responses dedicated to smuggler/agent, forger, etc. So, she was designated as general relationship.


Colombia. In Costa Rica he made arraignments with Kelsea Barnes and paid her $450 USD. Kelsea Barnes was described as a Hispanic female who is married to a man form African descent.

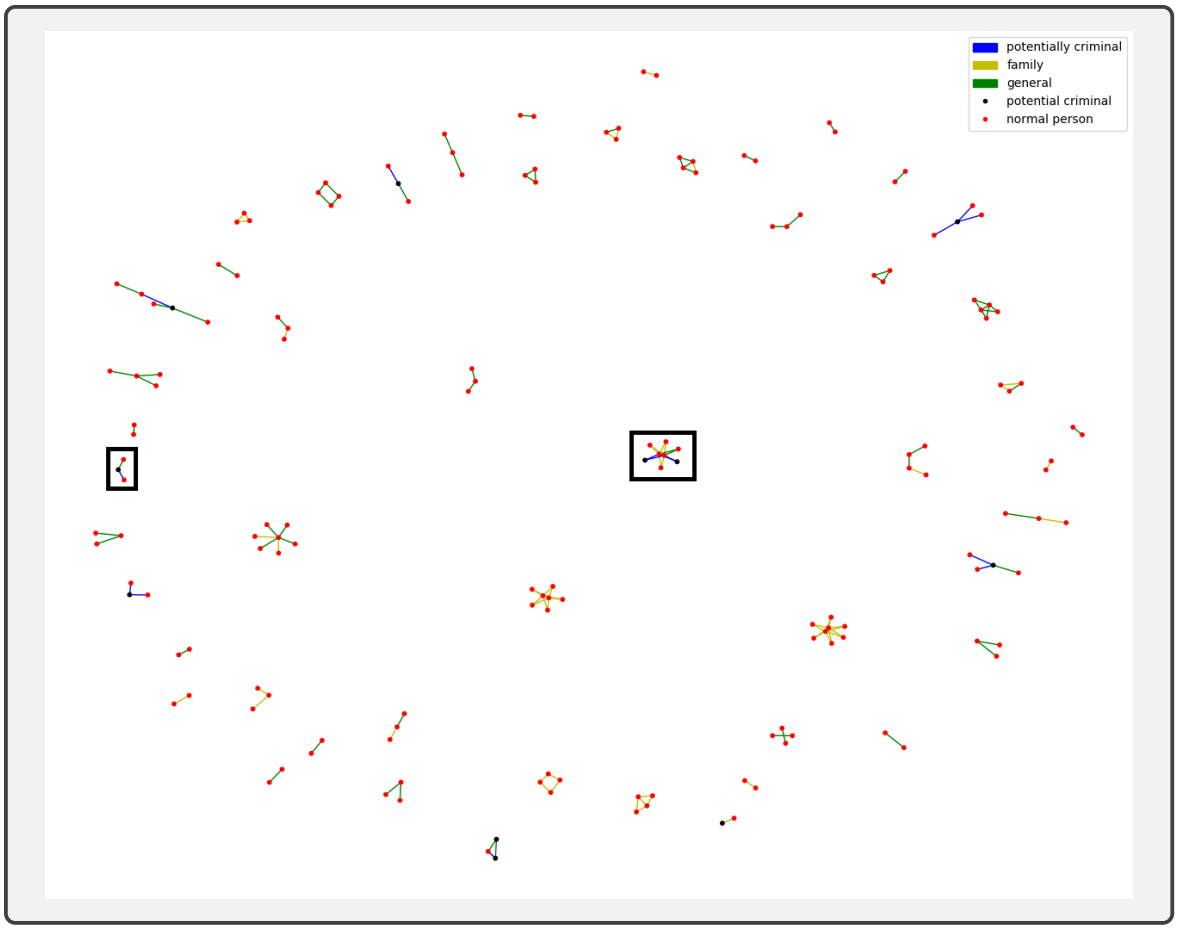
(a) Person A interview mentioning *Kelsea Barnes* as a smuggler.


Rica and processed. He then moved on to La Cruz and found out about Kelsea Barnes who he paid 500.00 USD to cross into Nicaragua in car. They

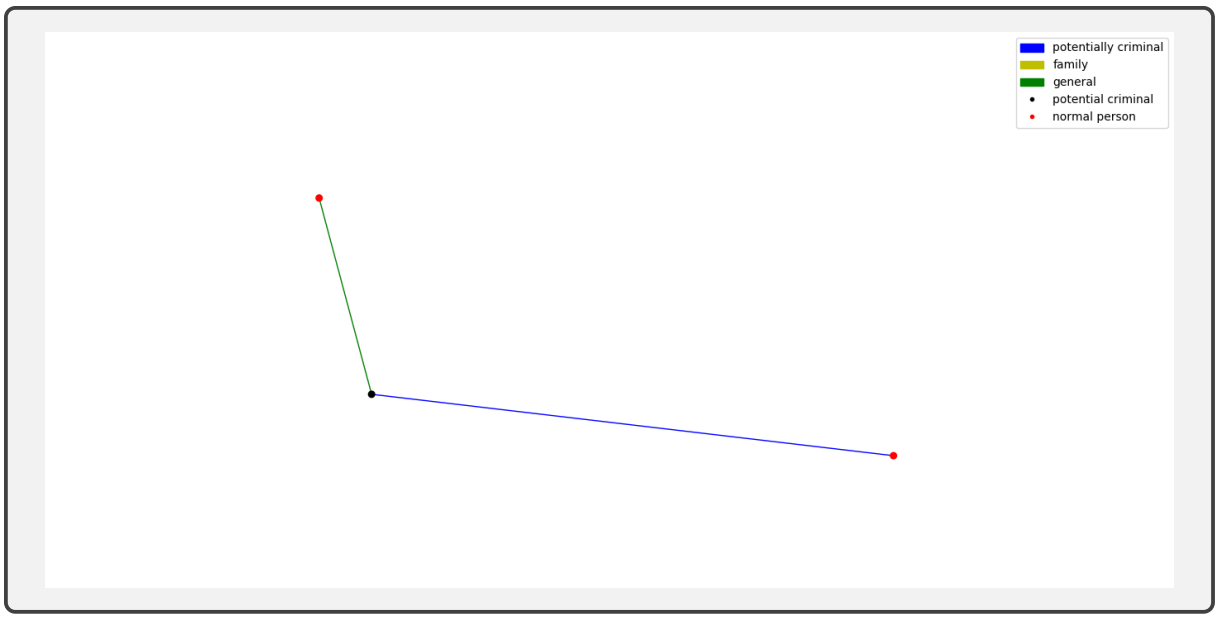
(b) Person B interview mentioning *Kelsea Barnes* as a general relationship.

Figure 10. Interviews relating to Figure 11b.

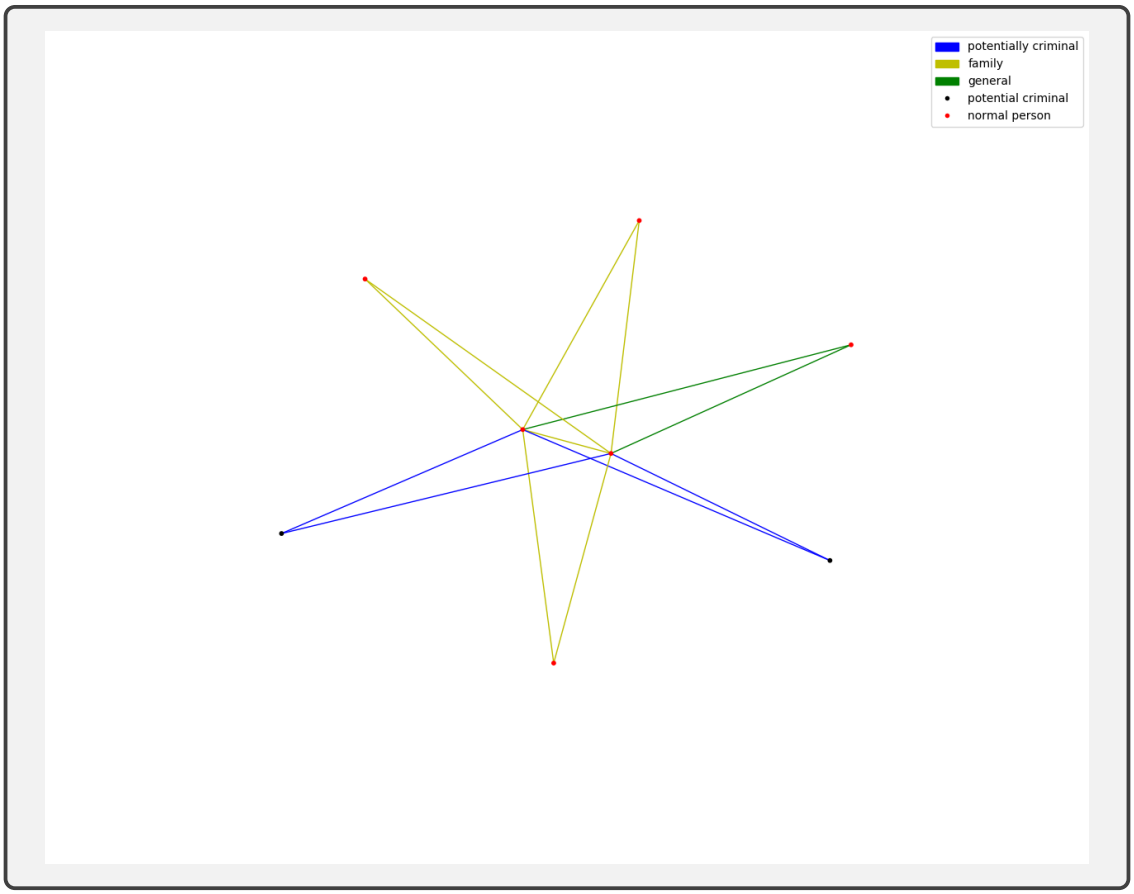
Figure 11c is a cluster containing two potential criminals and six unsuspicious individuals. The two red nodes in the middle are a husband and wife who both mentioned the same two potential criminals. The interview response from which these names were taken is shown in Figure 12. Both husband and wife provided identical responses. So, the response is only shown once. The response is to Question 8 from Table 1, asking "Describe the individuals who helped you reach Mexico." Sometimes the response in this section contains names of people who are actually smugglers. This often gets verified by the fact that the names will be referenced again in the response to Questions 9 and 10 from Table 1. In this case, the names did not have



(a) Full network.



(b) L-shaped boxed portion of network.



(c) Star shaped boxed portion of network.

Figure 11. Network of connections by document.

the additional agent/smuggler, or forger relationship to the husband and wife. For the sake of being thorough, this was checked manually. The full interviews were examined. Neither one explicitly indicated that *Alan Green* or *Brian Schultz* were agents/smugglers or forgers. Nonetheless, their inclusion in the database as people who were involved in helping travelers reach Mexico is valuable and could be updated in the future if someone else mentions them in a different capacity. Returning to the interview snippet in Figure 12, there is a third name at the bottom. For one reason or another *Marley* was only extracted from the husbands interview and not the wife's. For that reason, *Marley* did not show up as a node in the network from Figure 11 since it was not a name that appeared in both documents.

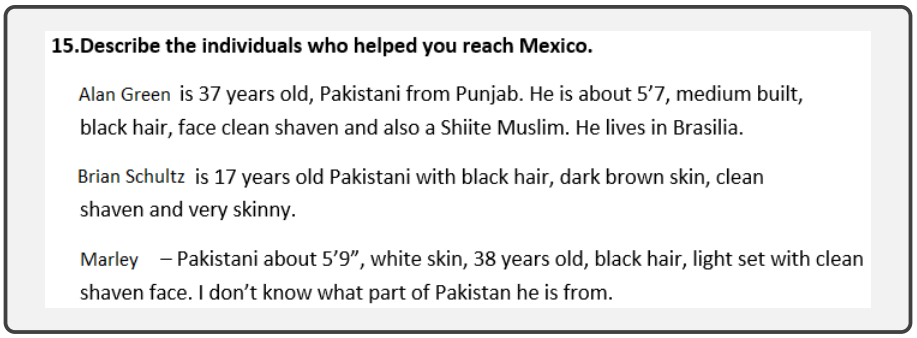


Figure 12. Interview relating to Figure 11c.

The other red nodes from Figure 11c are a combination of three family members of the husband and wife and one person with whom they both have a general relationship.

VI. Conclusions and Future Research

6.1 Conclusion

For a human to complete the type of analysis done in Examples 1 and 2 from Chapter 5, they would have to read all 650 documents and perfectly recall the information in each of them. Even though 650 documents is a very small data set, this would take an analyst over 16 hours assuming each document takes an average of 90 seconds to read. Then, even if the analyst did have perfect recall, that information is stored only in their brain. The automation system allows the analyst to let a computer process and store the information. This approach allows for a more appropriate division of labor by freeing up the analyst to spend their time working with a processed dataset and putting the burden of information processing and storage on the computer.

In light of this, the automated natural language processing system has shown itself capable of processing interviews and writing information to an SQL database. Examples 1 and 2 from Chapter 5 highlight the ease of analysis and the capabilities available when the interviews are processed into a structured format. As more interviews are input, the database will continually add new connections and the types of networks visualized in Figures 7 and 11 will grow. Furthermore, since Chapter 5 was only meant as a proof of concept, only two examples were given of what could be analyzed using the database. However, storage in a database provides analysts the ability to easily write many other queries that allow them to explore other areas they see fit.

6.2 Future Research

There are many ways that research on this topic could be furthered. First, with only 650 documents for initial processing, the networks that were generated in Figures 7 and 11 were not very connected. It is assumed that with enough interviews these networks would be far bigger and more connected. At that point, a number of network analysis metrics for understanding the centrality of the network could be employed such as closeness, betweenness, and degree of the nodes (people). Degree indicates how many nodes a given node is connected to. Betweenness is a measure of how often a node is traversed as the shortest path between nodes. Closeness is a measure of the distance from a node to all other nodes [23]. These kind of metrics would be very valuable when analyzing networks involving smugglers and other bad actors.

Another thing that could be done is the inclusion of more variables in the database. There are a number of other variables, such as aliases/nicknames for people, occupation, religion, etc., that could prove useful. Additionally, due to the inconsistent nature of how interview write-ups are formatted, there is a constant balance that must be struck within the code. The rules employed must be specific enough to discriminate between the variables it is searching for and other words that are not of interest, while also being flexible enough to ensure that information is actually extracted. This thesis does not claim to have found the perfect balance. There is most likely some refinement that could be done to improve the existing system.

In terms of the GUI, similarity detection using name spelling could be explored. Instead of determining name similarity based upon a certain number of words within two names matching, the similarity could be based upon letters within the words matching. Testing for name similarity in this way would address the assumption that there are misspellings within the data.

Finally, since it is so beneficial to have the interview's information stored inside a relational database, it seems that the idea of creating a front-end data entry system to replace transcription of the interviews should be explored. A GUI, similar to the one shown in Figure 3, but more advanced, could be constructed and used as the primary tool for storing what is said in interviews. The information could be input to an SQL database in real time, as the interviews are being conducted. This eliminates the errors in information extraction associated with using imperfect statistical models to extract entities, as well as the the inherent difficulty of defining an appropriate rules based system to handle irregular interview formats.

Appendix

A

A.1.

Listing VI.1. Code for extracting questions from documents.

```
import os,glob
from docx import Document
#import xlsxwriter
# extracting paragraph entries from docs
path="C:/Users/Nathanael Beveridge/Documents/INTERVIEWS"
fullList=[]
paragraphList=[]
for file in glob.glob(os.path.join(path, "*.docx" or "*.DOCX")):
    try:
        doc=open(file, "rb")
        document=Document(doc)
        doc.close()
        #paragraphList.clear()
        paragraphs = document.paragraphs
        for paragraph in paragraphs:
            if paragraph.style.name == "List Paragraph":
                paragraphList.append(paragraph.text)
        fullList.append(paragraphList)
    except:
        pass
# do lists in for loops need to be in enumerate(list)?
# getting unique entries
k=0
newlist=[]
for i in fullList:
    for j in i:
        x=0
        if k == 0:
            question=fullList[0][0]
            newlist.append(question)
            k=k+1
        else:
            for l in newlist:
                if l == j:
                    x=x+1
                if x>0:
                    break
            if x==0:
                newlist.append(j)
# more cleaning to keep those that begin with describe and those that end with question marks
cleanedList1=[]
for entry in newlist:
    if "Describe" in entry:
        cleanedList1.append(entry)
    elif entry.endswith("?"):
        cleanedList1.append(entry)
# write new list to a file for viewing
# from open source
def listToTxtFile(list, path, fileName):
    theFile=open(path+"/"+fileName, "w")
    for entry in list:
        theFile.write("%s\n" % entry)
#CODE FOR EXTRACTING UNIQUE QUESTIONS BASED ON COSINE DISTANCE
# https://stackoverflow.com/questions/15173225/
#calculate-cosine-similarity-given-2-sentence-strings
import re, math
from collections import Counter
WORD = re.compile(r'\w+')
def get_cosine(vec1, vec2):
     intersection = set(vec1.keys()) & set(vec2.keys())
     numerator = sum([vec1[x] * vec2[x] for x in intersection])
     sum1 = sum([vec1[x]**2 for x in vec1.keys()])
     sum2 = sum([vec2[x]**2 for x in vec2.keys()])
     denominator = math.sqrt(sum1) * math.sqrt(sum2)
     if not denominator:
        return 0.0
     else:
        return float(numerator) / denominator
def text_to_vector(text):
     words = WORD.findall(text)
     return Counter(words)
# MY CODE FOR SEARCHING BELOW
i=0
questionList=[]
questionList.append(cleanedList1[0])
for entry in cleanedList1:
    for k in range(i, len(cleanedList1)):
        text1=entry
        text2=cleanedList1[k]
        vector1=text_to_vector(text1)
        vector2=text_to_vector(text2)
        cosine=get_cosine(vector1,vector2)
        if cosine<0.85:
```

```
            n=0
            for j in questionList:
                text3=j
                vector3=text_to_vector(text3)
                cosine=get_cosine(vector2,vector3)
                if cosine>0.85:
                    n=n+1
                if n>0:
                    break
            if n==0:
                questionList.append(text2)
listToTxtFile(questionList,"C:/Users/Nathanael Beveridge/Documents/Interviews (.txt)/Interview Questions", "newQuestionList(0.85).txt" )
```

A.2.

Listing VI.2. Code for extracting the response for a question.

```
########## Extraction of first question line by line into list ##########################
        personalInformation = questionMasterList[0]
        personalInformationVec = text_to_vector(personalInformation)
        personalInformationAnalysisList = []
        x = 0
        for entry in textTry2:
            if x < 1:
                entryVec = text_to_vector(entry)
                cosine = get_cosine(personalInformationVec, entryVec)
                if cosine > 0.95:
                    x = x + 1
            if x == 1:
                x = x + 1
                continue
            if x == 2:
                for question in questionMasterList:
                    entryVecPt2 = text_to_vector(entry)
                    questionVec = text_to_vector(question)
                    cosinePt2 = get_cosine(entryVecPt2, questionVec)
                    if cosinePt2 < 0.95:
                        personalInformationAnalysisList.append(entry)
                    if cosinePt2 >= 0.95:
                        x = x + 1
                        break
            if x > 2:
                break
        from more_itertools import unique_everseen
        personalInformationAnalysisListCleaned = list(unique_everseen(personalInformationAnalysisList))
        personalInformationAnalysisList2 = []
        for entry in personalInformationAnalysisListCleaned:
            x = 0
            for question in questionMasterList:
                entryVecPt3 = text_to_vector(entry)
                questionVecPt2 = text_to_vector(question)
                cosinePt3 = get_cosine(entryVecPt3, questionVecPt2)
                if cosinePt3 < 0.95:
                    x = x + 1
            if x == len(questionMasterList):
                personalInformationAnalysisList2.append(entry)
        personalInformationAnalysisListFinal = [x.replace("\t", " ").replace("\n", "") for x in personalInformationAnalysisList2]
```

A.3.

Listing VI.3. Question 1 processing.

```
            groupingsTemp = []
            groupings = []
            x6 = 0
            personalQuestionsChunks = []
            for line in personalInformationAnalysisListFinal:
                line = line.replace("Tel", "tel").replace("tel", "telephone").replace("#", "").replace("/", " ").replace(
                    "Email", "email").replace("Facebook", "facebook").replace("Height", "height").replace("Weight",
                                                                                                          "weight").replace(
                    "PASSPORT", "passport").replace("Passport", "passport")
                personalQuestionsChunks.append(sNLP.word_tokenize(line))
            for line in personalQuestionsChunks:
                x6 = 0
                for entry in line:
                    if x6 == 0 and (
                                                entry == "telephone" or entry == "FINS" or entry == "height" or entry == "weight" or entry ==
                                                        "passport" or entry == "facebook"): # or entry[0] == "email"
                        x6 = x6 + 1
                        groupingsTemp.append(entry)
                        continue
                    if x6 > 0 and (
                                                entry != "telephone" and entry != "FINS" and entry != "height" and entry != "weight" and entry
                                                        != "passport" and entry != "facebook"): # and entry[0] != "email"
                        groupingsTemp.append(entry)
                    if x6 > 0 and (
```

```
                                entry == "telephone" or entry == "FINS" or entry == "height" or entry == "weight" or entry ==
                                    "passport" or entry == "facebook"): # or entry[0] == "email"
            temp = " ".join(groupingsTemp)
            groupings.append(temp)
            groupingsTemp.clear()
            groupingsTemp.append(entry)
    if len(groupingsTemp) != 0:
        temp = " ".join(groupingsTemp)
        groupings.append(temp)
        groupingsTemp.clear()
subjectFacebookTemp = []
subjectFacebook = []
subjectHeightTemp = []
subjectHeightFeet = []
subjectHeightInches = []
subjectHeight = []
subjectWeightTemp = []
subjectWeight = []
subjectPassportTemp = []
subjectPassport = []
subjectFINSTemp = []
subjectFINS = []
subjectTelephoneTemp = []
subjectTelephone = []
groupingsNER = []
subjectInfoTemp = []
emailChunk = []
emailChunkNER = []
facebookChunk = []
facebookChunkNER = []
weightChunk = []
weightChunkNER = []
heightChunk = []
heightChunkNER = []
FINSChunk = []
FINSChunkNER = []
passportChunk = []
passportChunkNER = []
telephoneChunk = []
telephoneChunkNER = []
groupingsChunks = []
personalQuestionsChunksNER = []
personalQuestionsJoin = []
if len(groupings) != 0:
    for entry in groupings:
        groupingsChunks.append(sNLP.word_tokenize(entry))
    for i in groupingsChunks:
        for word in i:
            if word == "height":
                heightChunk.append(i)
                temp = " ".join(heightChunk[0])
                heightChunk.clear()
                heightChunk.append(temp)
            if word == "weight":
                weightChunk.append(i)
                temp = " ".join(weightChunk[0])
                weightChunk.clear()
                weightChunk.append(temp)
            if word == "FINS":
                FINSChunk.append(i)
                temp = " ".join(FINSChunk[0])
                FINSChunk.clear()
                FINSChunk.append(temp)
            if word == "passport":
                passportChunk.append(i)
                temp = " ".join(passportChunk[0])
                passportChunk.clear()
                passportChunk.append(temp)
            if word == "telephone":
                telephoneChunk.append(i)
                temp = " ".join(telephoneChunk[0])
                telephoneChunk.clear()
                telephoneChunk.append(temp)
            if word == "facebook":
                facebookChunk.append(i)
                temp = " ".join(facebookChunk[0])
                facebookChunk.clear()
                facebookChunk.append(temp)
################################################## HEIGHT EXTRACTION ##################################################
if len(heightChunk) != 0:
    for i in heightChunk:
        heightChunkNER.append(sNLP.ner(i))
    inches = 0
    feet = 0
    counter1 = 0
    for i in heightChunkNER[0]:
        if i[1] == "NUMBER" and feet == 0 and counter1 == 0:
            subjectHeightFeet.append(i[0])
            counter1 = counter1 + 1
            continue
        if i[1] == "NUMBER" and counter1 > 0:
            subjectHeightInches.append(i[0])
    if len(subjectHeightInches)==0 and len(subjectHeightFeet)==0:
        subjectHeight.append('subject height not given')
    if len(subjectHeightInches)>0 and len(subjectHeightFeet)==0:
        subjectHeight.append(int(subjectHeightInches[0]))
    if len(subjectHeightInches)==0 and len(subjectHeightFeet)>0:
        subjectHeight.append(int(subjectHeightFeet[0])*12)
    if len(subjectHeightInches)>0 and len(subjectHeightFeet)>0:
        subjectHeight.append(int(subjectHeightInches[0]) + int(subjectHeightFeet[0]) * 12)
else:
    counter6 = 0
    j1 = 1
    j2 = 0
    for line in personalQuestionsChunks:
```

```
            temp = " ".join(line)
            personalQuestionsJoin.append(temp)
        for line in personalQuestionsJoin:
            personalQuestionsChunksNER.append(sNLP.ner(line))
        print(personalQuestionsChunksNER)
        for line in personalQuestionsChunksNER:
            for entry in line:
                j1 = j1 + 1
                if entry[1] == "NUMBER" and j1 != 1:
                    subjectHeightFeet.append(entry[0])
                    j1 = 0
                    continue
                if entry[0] == "'" and j1 == 1:
                    j1 = 0
                    continue
                if entry[1] == "NUMBER" and j1 == 1:
                    subjectHeightInches.append(entry[0])
                    break
                if entry[1] != "NUMBER" and j1 == 1:
                    subjectHeightInches.clear()
                    subjectHeightFeet.clear()
        if len(subjectHeightFeet) != len(subjectHeightInches) or (
                len(subjectHeightFeet) == 0 and len(subjectHeightInches) == 0):
            subjectHeightInches.clear()
            subjectHeightFeet.clear()
        else:
            subjectHeight.append(int(subjectHeightInches[0]) + int(subjectHeightFeet[0]) * 12)
    if len(subjectHeight) == 0:
        subjectHeight.append("subject height not given")
        ############################################## WEIGHT EXTRACTION ##################################################
    if len(weightChunk) != 0:
        for i in weightChunk:
            weightChunkNER.append(sNLP.ner(i))
        counter2 = 0
        for i in weightChunkNER[0]:
            if i[1] == "NUMBER" and counter2 == 0:
                subjectWeight.append(i[0])
                counter2 = counter2 + 1
        counter2 = 0
    else:
        counter5 = 0
        j = 1
        for line in personalQuestionsChunksNER:
            for entry in line:
                j = j + 1
                if entry[1] == "NUMBER" and counter5 == 0:
                    counter5 = counter5 + 1
                    subjectWeightTemp.append(entry[0])
                    j = 0
                    continue
                if (entry[0] == "LBS" or entry[0] == "lbs." or entry[0] == "pounds" or entry[
                    0] == "lbs") and j == 1 and counter5 > 0:
                    temp = " ".join(subjectWeightTemp)
                    subjectWeight.append(temp)
                    subjectWeightTemp.clear()
                    counter5 = 0
                else:
                    subjectWeightTemp.clear()
                    counter5 = 0
    if len(subjectWeight) == 0:
        subjectWeight.append("subject weight not given")
        ################################### FINS EXTRACTION ######################################################
    if len(FINSChunk) != 0:
        for i in FINSChunk:
            FINSChunkNER.append(sNLP.ner(i))
        counter3 = 0
        for i in FINSChunkNER[0]:
            if i[1] == "NUMBER" and counter3 == 0:
                subjectFINS.append(i[0])
                counter3 = counter3 + 1
        counter3 = 0
    else:
        subjectFINS.append("subject FINS # not given")
    ################################### PASSPORT EXTRACTION ##################################################
    if len(passportChunk) != 0:
        for i in passportChunk:
            passportChunkNER.append(sNLP.ner(i))
        counter4 = 0
        for i in passportChunkNER[0]:
            if i[1] == "NUMBER" or i[1] == "MONEY" and i[0] != "#" and counter4 == 0:
                subjectPassport.append(i[0])
                counter4 = counter4 + 1
        counter4 = 0
    else:
        subjectPassport.append("subject passport # not given")
    ################################### FACEBOOK EXTRACTION ##################################################
    if len(facebookChunk) != 0:
        for i in facebookChunk:
            facebookChunkNER.append(sNLP.word_tokenize(i))
        print(facebookChunkNER)
        for i in facebookChunkNER[0]:
            if i != "facebook" and i != ":" and i != "-" and i != "NONE" and i != "None" and i != "none" and i != "claims" and
                    i!="Claims" and i != "N/A":
                subjectFacebookTemp.append(i)
        if len(subjectFacebookTemp) != 0:
            temp = " ".join(subjectFacebookTemp)
            subjectFacebook.append(temp)
            subjectFacebookTemp.clear()
        else:
            subjectFacebookTemp.clear()
    else:
        subjectFacebook.append("subject facebook not given")
    ################################### TELEPHONE EXTRACTION ##################################################
    if len(telephoneChunk) != 0:
        for i in telephoneChunk:
```

```
        telephoneChunkNER.append(sNLP.ner(i))
    print(telephoneChunkNER)
    for i in telephoneChunkNER[0]:
        if i[1] == "NUMBER" or i[1] == "MONEY":
            subjectTelephoneTemp.append(i[0])
    if len(subjectTelephoneTemp) != 0:
        temp = " ".join(subjectTelephoneTemp)
        subjectTelephone.append(temp)
        subjectTelephoneTemp.clear()
if len(subjectTelephone)==0:
    subjectTelephone.append("subject telephone not given")
############################################## now extract from first question, name, DOB, COB
      ################################
#########################################################################################################################################################
t1 = 0
x1 = 0
x2 = 0
x5 = 0
x4 = 0
for line in personalInformationAnalysisListFinal:
    text = line
    textCleaned = text.replace("'", "").replace("/", " / ").replace("MAY", " May ").replace("may",
                                                                                            " May ").replace(
        "JUNE", " June ").replace("june", " June ").replace("JULY", " July ").replace("july", " July ").replace(
        "AUGUST", " August ").replace("AUG", " August ").replace("aug", " August ").replace("SEPTEMBER",
                                                                                            " September ").replace(
        "SEPT", " September ").replace("sept", " September ").replace("OCTOBER", " October ").replace("OCT",
                                                                                                      " October ").replace(
        "oct", " October ").replace("NOVEMBER", " November ").replace("NOV", " November ").replace("nov",
                                                                                                   " November ").replace(
        "DECEMBER", " December ").replace("DEC", " December ").replace("dec", " December ").replace("JANUARY",
                                                                                                    " January ").replace(
        "JAN", " January ").replace("jan", " January ").replace("FEBRUARY", " February ").replace("FEB",
                                                                                                  " February ").replace(
        "feb", " February ").replace("MARCH", " March ").replace("MAR", " March ").replace(
        "APRIL", " April ").replace("APR", " April ").replace("apr", " April ").replace("years old",
                                                                                        "").replace("-",
                                                                                                    " ").replace(
        "Cedula", "cedula").replace("Father", "father").replace("Fathers", "father").replace("fathers",
                                                                                             "father").replace(
        "FATHER", "father").replace("Brother", "brother").replace("Brothers", "brother").replace("BROTHER",
                                                                                                 "brother").replace(
        "BROTHERS", "brother").replace("Half-", "").replace("half-brother", "brother").replace("Half-Brother",
                                                                                               "brother").replace(
        "brothers", "brother").replace("Mother", "mother").replace("mothers", "mother").replace("Mothers",
                                                                                                "mother").replace(
        "MOTHER", "mother").replace("half-sister", "sister").replace("Sister", "sister").replace("sisters",
                                                                                                 "sister").replace(
        "Sisters", "sister").replace("SISTER", "sister").replace("SISTERS", "sister").replace("&",
                                                                                              "and").replace(
        "bother", "brother").replace("(DECEASED)", "deceased").replace("(deceased)", "deceased").replace(
        "DECEASED", "deceased").replace("Deceased", "deceased").replace("yrs.", "").replace("yrs", "").replace(
        "YOA", "").replace("Step-Father", "father").replace("Step-Fathers", "father").replace("STEP-FATHER",
                                                                                              "father").replace(
        "Step Father", "father").replace("Step Fathers", "father").replace("STEP FATHER", "father").replace(
        "Stepfather", "father").replace("stepfathers", "father").replace("stepfather", "father").replace(
        "Siblings", "sibling").replace("Cousin", "cousin").replace("Cousins", "cousin").replace("Sibling",
                                                                                                "sibling").replace(
        "siblings", "sibling")
    nerTagsPersonalInformationForSubjectName = sNLP.ner(textCleaned)
    if t1 == 0:
        subjectNameTemp = []
        subjectName = []
    c = 0
    for i in nerTagsPersonalInformationForSubjectName:
        if (i[0] != "Name" and i[0] != "NAME" and i[0] != "Name:" and i[0]!='ANSWER:' and i[0] != "NAME:" and i[0] != ":" and
            i[
            0] != "was" and i[0] != "born" and i[0] != "," and i[0] != ";" and i[0] != "DOB" and i[
            0] != "Date" and i[0] != "DATE" and i[1] != "DATE" and i[1] != "NUMBER" and
                   i[0] != "/") and c == 0 and x4 == 0:
            subjectNameTemp.append(i[0])
            c = c + 1
            continue
        if (i[0] != "Name" and i[0] != "NAME" and i[0] != "Name:" and i[0]!='ANSWER:' and i[0] != "NAME:" and i[0] != ":" and
                (
                                        i[0] != "was" and i[0] != "born" and i[0] != "," and i[
                                0] != ";" and i[0] != "DOB" and i[0] != "Date" and i[0] != "DATE" and i[
                    1] != "DATE" and i[1] != "NUMBER" and i[0] != "/")) and c > 0 and x4 == 0:
            subjectNameTemp.append(i[0])
            c = c + 1
        if (i[0] == "was" or i[0] == "born" or i[0] == "," or i[0] == ";" or i[0] == "DOB" or i[0] == "Date" or
                    i[0] == "DATE" or i[1] == "DATE" or i[1] == "NUMBER" or i[
            0] == "/") and x4 == 0:
            temp = " ".join(subjectNameTemp)
            subjectName.append(temp)
            subjectNameTemp.clear()
            x4 = x4 + 1
    t1 = t1 + 1
c = 0
x4 = 0
t1 = 0
for line in personalInformationAnalysisListFinal:
    text = line
    textCleaned = text.replace("'", "").replace("/", " ").replace("MAY", " May ").replace("may",
                                                                                           " May ").replace(
        "JUNE", " June ").replace("june", " June ").replace("JULY", " July ").replace("july", " July ").replace(
        "AUGUST", " August ").replace("AUG", " August ").replace("aug", " August ").replace("SEPTEMBER",
                                                                                            " September ").replace(
        "SEPT", " September ").replace("sept", " September ").replace("OCTOBER", " October ").replace("OCT",
                                                                                                      " October ").replace(
        "oct", " October ").replace("NOVEMBER", " November ").replace("NOV", " November ").replace("nov",
                                                                                                   " November ").replace(
        "DECEMBER", " December ").replace("DEC", " December ").replace("dec", " December ").replace("JANUARY",
                                                                                                    " January ").replace(
        "JAN", " January ").replace("jan", " January ").replace("FEBRUARY", " February ").replace("FEB",
```

```
                                                                                                    " February ").replace(
        "feb", " February ").replace("MARCH", " March ").replace("MAR", " March ").replace(
        "APRIL", " April ").replace("APR", " April ").replace("apr", " April ").replace("years old",
                                                                                        "").replace("-",
                                                                                                    " ").replace(
        "Cedula", "cedula").replace("Father", "father").replace("Fathers", "father").replace("fathers",
                                                                                             "father").replace(
        "FATHER", "father").replace("Brother", "brother").replace("Brothers", "brother").replace("BROTHER",
                                                                                                 "brother").replace(
        "BROTHERS", "brother").replace("Half-", "").replace("half-brother", "brother").replace("Half-Brother",
                                                                                               "brother").replace(
        "brothers", "brother").replace("Mother", "mother").replace("mothers", "mother").replace("Mothers",
                                                                                               "mother").replace(
        "MOTHER", "mother").replace("half-sister", "sister").replace("Sister", "sister").replace("sisters",
                                                                                                 "sister").replace(
        "Sisters", "sister").replace("SISTER", "sister").replace("SISTERS", "sister").replace("&",
                                                                                              "and").replace(
        "bother", "brother").replace("(DECEASED)", "deceased").replace("(deceased)", "deceased").replace(
        "DECEASED", "deceased").replace("Deceased", "deceased").replace("yrs.", "").replace("yrs", "").replace(
        "YOA", "").replace("Step-Father", "father").replace("Step-Fathers", "father").replace("STEP-FATHER",
                                                                                              "father").replace(
        "Step Father", "father").replace("Step Fathers", "father").replace("STEP FATHER", "father").replace(
        "Stepfather", "father").replace("stepfathers", "father").replace("stepfather", "father").replace(
        "Siblings", "sibling").replace("Cousin", "cousin").replace("Cousins", "cousin").replace("Sibling",
                                                                                               "sibling").replace(
        "siblings", "sibling")

    nerTagsPersonalInformation = sNLP.ner(textCleaned)
    if t1 == 0:
        subjectDOB = []
        subjectDOBTemp = []
        subjectEmail = []
        subjectCOBTemp = []
        subjectCOB = []
        nameCheckerInitial = []
        nameCheckerTemp = []
        nameChecker = []
    x = 0
    x3 = 0
    x4 = 0
    c = 0
    for i in nerTagsPersonalInformation:
        c = 0
        if i[1] == "DATE" or i[1] == "NUMBER" and x2 == 0:
            subjectDOBTemp.append(i[0])
            x3 = x3 + 1
        if i[1] != "DATE" and i[1] != "NUMBER" and x3 > 0:
            temp = " ".join(subjectDOBTemp)
            subjectDOB.append(temp)
            subjectDOBTemp.clear()
            x3 = 0
            x2 = x2 + 1
        if i[1] == "EMAIL":
            subjectEmail.append(i[0])
        if (i[1] == "LOCATION" or i[1] == "ORGANIZATION" or i[1] == "COUNTRY" or i[1] == "CITY" or i[
            1] == "STATE_OR_PROVINCE") and i[0] != "POB" and i[0] != "Place" and i[
            0] != "place" and x5 == 0 and x4 == 0 and x2>0:
            subjectCOBTemp.append(i[0])
            x4 = x4 + 1
            continue
        if (i[1] == "LOCATION" or i[1] == "ORGANIZATION" or i[1] == "COUNTRY" or i[1] == "STATE_OR_PROVINCE" or
                i[1] == "CITY" or i[0] == ",") and x4 > 0 and x5 == 0:
            if i[0] == ",":
                c = c + 1
            subjectCOBTemp.append(i[0])
        if i[1] != "LOCATION" and i[1] != "ORGANIZATION" and i[1] != "COUNTRY" and i[1] != "CITY" and i[
            1] != "STATE_OR_PROVINCE" and x4 > 0 and c == 0:
            temp = " ".join(subjectCOBTemp)
            subjectCOB.append(temp)
            subjectCOBTemp.clear()
            x4 = 0
            x5 = x5 + 1
    subjectNamePresent = 1
    if len(subjectNameTemp) != 0:
        temp = " ".join(subjectNameTemp)
        subjectName.append(temp)
        subjectNameTemp.clear()
        x1 = x1 + 1
        x = 0
    if len(subjectDOBTemp) != 0:
        temp = " ".join(subjectDOBTemp)
        subjectDOB.append(temp)
        subjectDOBTemp.clear()
        x3 = 0
        x2 = x2 + 1
    if len(subjectCOBTemp) != 0:
        temp = " ".join(subjectCOBTemp)
        subjectCOB.append(temp)
        subjectCOBTemp.clear()
        x4 = 0
        x5 = x5 + 1
if len(subjectName) == 0:
    subjectName.append("subject name not recovered")
if len(subjectEmail) != len(subjectName):
    subjectEmail.append("no email provided")
if len(subjectDOB) != len(subjectName):
    subjectDOB.append("no DOB provided")
if len(subjectCOB) != len(subjectName):
    subjectCOB.append("no COB provided")
for i in subjectName:
    nameCheckerInitial.append(sNLP.word_tokenize(i))
temp = " ".join(nameCheckerInitial[0])
i = 0
while i < len(nameCheckerInitial[0]) + 1:
    nameChecker.append(temp)
```

```
            del nameCheckerInitial[0][0]
            temp = " ".join(nameCheckerInitial[0])
            i = i + 1
        if len(nameCheckerInitial[0]) != 0:
            temp = " ".join(nameCheckerInitial[0])
            nameChecker.append(temp)
        upperCaseChecker = []
        wordLength=0
        for i in subjectName:
            upperCaseChecker.append(sNLP.word_tokenize(i))
        x9 = 0
        for i in upperCaseChecker[0]:
            for x in i:
                if x.isupper():
                    wordLength=wordLength+1
            if wordLength==len(i):
                nameChecker.append(i)
            wordLength=0
        x9 = 0
        nameCheckerInitial.clear()
        nameChecker.append(sNLP.word_tokenize(subjectName[0])[0])
```

Listing VI.4. Question 2 processing.

```
    t = 0
    if len(analysisListFinal)!=0:
        for line in analysisListFinal:
            text = line
            textCleaned = text.replace("'", "").replace("/", " and ").replace("bother", "brother").replace("phone",
                                                                                "telephone").replace(
                "tel", "telephone").replace("Tel", "telephone").replace("years old", "").replace("-", " ").replace(
                "Cedula", "cedula").replace("Father", "father").replace("Fathers", "father").replace("fathers",
                                                                                "father").replace(
                "FATHER", "father").replace("Brother", "brother").replace("Brothers", "brother").replace("BROTHER",
                                                                                "brother").replace(
                "BROTHERS", "brother").replace("Half-", "").replace("half-brother", "brother").replace("Half-Brother",
                                                                                "brother").replace(
                "brothers", "brother").replace("Mother", "mother").replace("mothers", "mother").replace("Mothers",
                                                                                "mother").replace(
                "MOTHER", "mother").replace("half-sister", "sister").replace("Sister", "sister").replace("sisters",
                                                                                "sister").replace(
                "Sisters", "sister").replace("SISTER", "sister").replace("SISTERS", "sister").replace("&",
                                                                                "and").replace(
                "bother", "brother").replace("(DECEASED)", "deceased").replace("(deceased)", "deceased").replace(
                "DECEASED", "deceased").replace("Deceased", "deceased").replace("yrs.", "").replace("yrs", "").replace(
                "YOA", "").replace("Step-Father", "father").replace("Step-Fathers", "father").replace("STEP-FATHER",
                                                                                "father").replace(
                "Step Father", "father").replace("Step Fathers", "father").replace("STEP FATHER", "father").replace(
                "Stepfather", "father").replace("stepfathers", "father").replace("stepfather", "father").replace(
                "Siblings", "sibling").replace("Cousin", "cousin").replace("Cousins", "cousin").replace("Sibling",
                                                                                "sibling").replace(
                "siblings", "sibling")
            nerTags = sNLP.ner(textCleaned)
            nerTagsTemp = []
            nerTagsAll = []
            b = 0
            for i in nerTags:
                if b == 0 and (i[0] == "brother" or i[0] == "brothers" or i[0] == "father" or i[0] == "fathers" or i[
                    0] == "mother" or i[0] == "mothers" or i[0] == "sister" or i[0] == "sisters" or i[0] == "sibling" or
                               i[0] == "cousin"): # or i[0] == "."):
                    b = b + 1
                    nerTagsTemp.append(i[0])
                    continue
                if b > 0 and (i[0] != "brother" and i[0] != "brothers" and i[0] != "father" and i[0] != "fathers" and i[
                    0] != "mother" and i[0] != "mothers" and i[0] != "sister" and i[0] != "sisters" and i[
                    0] != "sibling" and i[0] != "cousin"): # and i[0] != "."):
                    nerTagsTemp.append(i[0])
                if b > 0 and (i[0] == "brother" or i[0] == "brothers" or i[0] == "father" or i[0] == "fathers" or i[
                    0] == "mother" or i[0] == "mothers" or i[0] == "sister" or i[0] == "sisters" or i[0] == "sibling" or
                              i[0] == "cousin"): # or i[0] == "."):
                    temp = " ".join(nerTagsTemp)
                    nerTagsAll.append(temp)
                    nerTagsTemp.clear()
                    nerTagsTemp.append(i[0])
            if len(nerTagsTemp) != 0:
                temp = " ".join(nerTagsTemp)
                nerTagsAll.append(temp)
                nerTagsTemp.clear()
            if len(nerTagsAll) == 0:
                temp = " ".join(nerTagsTemp)
                nerTagsAll.append(temp)
                nerTagsTemp.clear()
            if t == 0:
                fatherName = []
                fatherDeceased = []
                fatherAge = []
                fatherLocation = []
                fatherLocationOtherName = []
                fatherNickname = []
                fatherTelephone = []
                motherName = []
                motherDeceased = []
                motherAge = []
                motherLocation = []
                motherLocationOtherName = []
                motherNickname = []
                motherTelephone = []
                brotherName = []
                brotherDeceased = []
                brotherAge = []
                brotherLocation = []
                brotherLocationOtherName = []
```

```
    brotherNickname = []
    brotherTelephone = []
    sisterName = []
    sisterDeceased = []
    sisterAge = []
    sisterLocation = []
    sisterLocationOtherName = []
    sisterNickname = []
    sisterTelephone = []
    cousinName = []
    cousinDeceased = []
    cousinAge = []
    cousinLocation = []
    cousinLocationOtherName = []
    cousinNickname = []
    cousinTelephone = []
    siblingAge = []
    siblingDeceased = []
    siblingName = []
    siblingLocation = []
    siblingLocationOtherName = []
    siblingNickname = []
    siblingTelephone = []
    personName = []
    personAge = []
    personLocationTemp = []
    personLocation = []
    personLocationOtherName = []
    personNicknameTemp = []
    personNickname = []
    personTelephone = []
    personTemp = []
    nerChunksCheck1 = []
    nerChunksCheck2 = []
    nerChunks = []
    personDeceased = []
    personProfile = []
for i in nerTagsAll:
    nerChunksCheck1.append(sNLP.ner(i))
for i in nerChunksCheck1:
    c = 0
    for o in i:
        if o[0] == 'mother':
            c = c + 1
        if o[0] == 'father':
            c = c + 1
        if o[0] == 'brother':
            c = c + 1
        if o[0] == 'cousin':
            c = c + 1
        if o[0] == 'sister':
            c = c + 1
        if o[0] == 'sibling':
            c = c + 1
    if c > 0:
        nerChunksCheck2.append(i)
nerChunksCheck2NameCheckerTemp = []
nerChunksCheck2NameChecker = []
nerChunksCheck2OtherWT = []
nerChunksNameDuplicate = []
nerChunksNameDuplicateChunks = []
nerChunksCheck2Other = []
nerChunksCheck2DuplicateB4Duplicate = []
nerChunksCheck2DuplicateName = []
nerChunksCheck2DuplicateAfterDuplicate = []
nerChunksCheck3=[]
counter = 0
x9 = 0
for i in nerChunksCheck2:
    c = 0
    for o in i:
        if o[1] == "PERSON":
            c = c + 1
    if c > 0:
        for o in i:
            if o[1] == "PERSON" and counter == 0:
                nerChunksCheck2NameCheckerTemp.append(o[0])
                counter = counter + 1
                continue
            if counter > 0 and o[1] == "PERSON":
                nerChunksCheck2NameCheckerTemp.append(o[0])
                counter = counter + 1
            if counter > 0 and o[1] != "PERSON":
                temp = " ".join(nerChunksCheck2NameCheckerTemp)
                nerChunksCheck2NameChecker.append(temp)
                nerChunksCheck2NameCheckerTemp.clear()
                counter = 0
        if len(nerChunksCheck2NameCheckerTemp) != 0:
            temp = " ".join(nerChunksCheck2NameCheckerTemp)
            nerChunksCheck2NameChecker.append(temp)
            nerChunksCheck2NameCheckerTemp.clear()
            counter = 0
        counter = 0
        for j in nameChecker:
            for k in nerChunksCheck2NameChecker:
                if j == k:
                    counter = counter + 1
                    nerChunksNameDuplicate.append(k)
                    for i in nerChunksCheck2[x9]:
                        nerChunksCheck2Other.append(i[0])
                    temp1 = " ".join(nerChunksCheck2Other)
                    if len(nerChunksNameDuplicate) == 1:
                        temp2 = temp1.replace(str(nerChunksNameDuplicate[0]), "")
                        nerChunksCheck3.append(sNLP.ner(temp2))
        if counter == 0:
```

```
            nerChunksCheck3.append(i)
        x9 = x9 + 1
    nerChunksCheck2NameChecker.clear()
for i in nerChunksCheck3:
    counter=0
    for o in i:
        if o[1]=="PERSON":
            counter=counter+1
    if counter>0:
        nerChunks.append(i)
################################################################################
if len(nerChunks) == 0:
    # do the whole check in here, then construct set of continue's to escape to the next iteration of the loop as normal
    nerTagsTemp.clear()
    nerTagsAll.clear()
    nerChunksCheck1.clear()
    nerChunksCheck2.clear()
    nerChunks.clear()
    b = 0
    for i in nerTags:
        if i[1] == "PERSON":
            b = b + 1
            nerTagsTemp.append(i[0])
        if i[0] == "father" or i[0] == "mother" or i[0] == "brother" or i[0] == "sister" or i[
            0] == "cousin" or i[0] == "sibling" and b > 0:
            nerTagsTemp.append(i[0])
            temp = " ".join(nerTagsTemp)
            nerTagsAll.append(temp)
            nerTagsTemp.clear()
    if len(nerTagsAll) == 0:
        temp = " ".join(nerTagsTemp)
        nerTagsAll.append(temp)
        nerTagsTemp.clear()
    for i in nerTagsAll:
        nerChunksCheck1.append(sNLP.ner(i))
    print(nerChunksCheck1)
    for i in nerChunksCheck1:
        # print(i)
        c = 0
        for o in i:
            if o[0] == 'mother':
                c = c + 1
            if o[0] == 'father':
                c = c + 1
            if o[0] == 'brother':
                c = c + 1
            if o[0] == 'cousin':
                c = c + 1
            if o[0] == 'sister':
                c = c + 1
            if o[0] == 'sibling':
                c = c + 1
        if c > 0:
            nerChunksCheck2.append(i)
    print(nerChunksCheck2)
    nerChunksCheck2NameCheckerTemp = []
    nerChunksCheck2NameChecker = []
    counter = 0
    counter1 = 0
    # below checks for if they mention sibling but then also mentions more granular sister or brother, it will take
        delete item list that just has sibling
    for i in nerChunksCheck2:
        c = 0
        for o in i:
            if o[1] == "PERSON":
                c = c + 1
        for o in i:
            if o[1] == "PERSON" and counter == 0:
                nerChunksCheck2NameCheckerTemp.append(o[0])
                counter = counter + 1
                continue
            if counter > 0 and o[1] == "PERSON":
                nerChunksCheck2NameCheckerTemp.append(o[0])
                counter = counter + 1
            if counter > 0 and o[1] == "PERSON":
                temp = " ".join(nerChunksCheck2NameCheckerTemp)
                nerChunksCheck2NameChecker.append(temp)
                nerChunksCheck2NameCheckerTemp.clear()
                counter = 0
        for k in nameChecker:
            for j in nerChunksCheck2NameChecker:
                if k == j:
                    counter = counter + 1
        if c > 0 and counter == 0:
            nerChunks.append(i)
    counter = 0
################################################################################
for i in nerChunks:
    x = 0
    y = 0
    bro = 0
    sis = 0
    pops = 0
    ma = 0
    cuz = 0
    sib = 0
    l = 0
    p = 0
    n = 0
    s = 0
    r = 0
    tel = 0
    match = 0
    for k in i:
        if k[0] == "mother":
```

```
            ma = ma + 1
        if k[0] == "brother":
            bro = bro + 1
        if k[0] == "sister":
            sis = sis + 1
        if k[0] == "father":
            pops = pops + 1
        if k[0] == "cousin":
            cuz = cuz + 1
        if k[0] == "sibling":
            sib = sib + 1
        if k[0] == "telephone" or k[0] == "phone" and tel == 0:
            tel = tel + 1
        if (k[1] == "NUMBER" or k[1] == "MONEY") and len(k[0]) > 2 and tel > 0:
            personTelephone.append(k[0])
            tel = 0
        if (k[1] == "CITY" or k[1] == "COUNTRY" or k[1] == "LOCATION" or k[
            1] == "STATE_OR_PROVINCE") and r == 0: # and c==0:
            personLocationTemp.append(k[0])
            r = r + 1
            continue
        if k[1] != "CITY" and k[1] != "COUNTRY" and k[1] != "LOCATION" and k[0] == "or" and k[
            1] != "STATE_OR_PROVINCE" and r > 0: # and c>0:
            temp = " ".join(personLocationTemp)
            personLocationOtherName.append(temp)
            personLocationTemp.clear()
        if r > 0 and (k[0] == "," or k[1] == "CITY" or k[1] == "COUNTRY" or k[1] == "LOCATION" or k[
            1] == "STATE_OR_PROVINCE"):
            personLocationTemp.append(k[0])
        if (k[1] != "CITY" and k[1] != "COUNTRY" and k[1] != "LOCATION" and k[1] != "STATE_OR_PROVINCE") and \
                    k[0] != "or" and k[0] != "," and r > 0: # and c>0:
            temp = " ".join(personLocationTemp)
            personLocation.append(temp)
            personLocationTemp.clear()
            r = 0
            if len(personLocationOtherName) != len(personName):
                personLocationOtherName.append("no alternate location name given")
        if k[1] == "NUMBER" and len(k[0]) < 3 and p == 0: # and y>0:
            personAge.append(int(2018)-int(k[0]))
            p = p + 1
        if k[0] == "deceased" or k[0] == "killed" or k[0] == "murdered" and n == 0: # and y>0:
            personDeceased.append("deceased")
            n = n + 1
        if k[1] == "PERSON":
            if len(personName) != len(personDeceased):
                personDeceased.append("assumed alive")
            if len(personName) != len(personAge):
                personAge.append("age not given")
            if len(personName) != len(personLocation):
                personLocation.append("location not given")
            if len(personName) != len(personTelephone):
                personTelephone.append("no telephone number given")
            if len(personName) != len(personLocationOtherName):
                personLocationOtherName.append("no alternate location name given")
            x = x + 1
            tel = 0
            p = 0
            personTemp.append(k[0])
        if k[1] != "PERSON" and k[0] == "or" and x > 0: # p==0:
            temp = " ".join(personTemp)
            personNickname.append(temp)
            personTemp.clear()
            x = 0
            n = 0
        if k[1] != "PERSON" and x > 0 and k[0] != "or": # p==0:
            temp = " ".join(personTemp)
            personName.append(temp)
            personTemp.clear()
            x = 0
            if len(personNickname) != len(personName):
                personNickname.append("no nickname given")
    if len(personTemp) != 0:
        temp = " ".join(personTemp)
        personName.append(temp)
        personTemp.clear()
        x = 0
    if len(personLocationTemp) != 0:
        temp = " ".join(personLocationTemp)
        personLocation.append(temp)
        personLocationTemp.clear()
        r = 0
    if len(personName) == 0:
        temp = " ".join(personTemp)
        personName.append(temp)
        personTemp.clear()
        x = 0
    if len(personName) != len(personDeceased):
        personDeceased.append("assumed alive")
    if len(personName) != len(personAge):
        personAge.append("age not given")
    if len(personName) != len(personLocation):
        personLocation.append("location not given")
    if len(personName) != len(personNickname):
        personNickname.append("no nickname given")
    if len(personLocationOtherName) != len(personName):
        personLocationOtherName.append("no alternate location name given")
    if len(personName) != len(personTelephone):
        personTelephone.append("no telephone number given")
    if ma > 0:
        for i in personName:
            motherName.append(i)
        for i in personDeceased:
            motherDeceased.append(i)
        for i in personAge:
            motherAge.append(i)
```

```
                    for i in personLocation:
                        motherLocation.append(i)
                    for i in personLocationOtherName:
                        motherLocationOtherName.append(i)
                    for i in personNickname:
                        motherNickname.append(i)
                    for i in personTelephone:
                        motherTelephone.append(i)
                if bro > 0:
                    for i in personName:
                        brotherName.append(i)
                    for i in personDeceased:
                        brotherDeceased.append(i)
                    for i in personAge:
                        brotherAge.append(i)
                    for i in personLocation:
                        brotherLocation.append(i)
                    for i in personLocationOtherName:
                        brotherLocationOtherName.append(i)
                    for i in personNickname:
                        brotherNickname.append(i)
                    for i in personTelephone:
                        brotherTelephone.append(i)
                if pops > 0:
                    for i in personName:
                        fatherName.append(i)
                    for i in personDeceased:
                        fatherDeceased.append(i)
                    for i in personAge:
                        fatherAge.append(i)
                    for i in personLocation:
                        fatherLocation.append(i)
                    for i in personLocationOtherName:
                        fatherLocationOtherName.append(i)
                    for i in personNickname:
                        fatherNickname.append(i)
                    for i in personTelephone:
                        fatherTelephone.append(i)
                if sis > 0:
                    for i in personName:
                        sisterName.append(i)
                    for i in personDeceased:
                        sisterDeceased.append(i)
                    for i in personAge:
                        sisterAge.append(i)
                    for i in personLocation:
                        sisterLocation.append(i)
                    for i in personLocationOtherName:
                        sisterLocationOtherName.append(i)
                    for i in personNickname:
                        sisterNickname.append(i)
                    for i in personTelephone:
                        sisterTelephone.append(i)
                if cuz > 0:
                    for i in personName:
                        cousinName.append(i)
                    for i in personDeceased:
                        cousinDeceased.append(i)
                    for i in personAge:
                        cousinAge.append(i)
                    for i in personLocation:
                        cousinLocation.append(i)
                    for i in personLocationOtherName:
                        cousinLocationOtherName.append(i)
                    for i in personNickname:
                        cousinNickname.append(i)
                    for i in personTelephone:
                        cousinTelephone.append(i)
                if sib > 0:
                    for i in personName:
                        siblingName.append(i)
                    for i in personDeceased:
                        siblingDeceased.append(i)
                    for i in personAge:
                        siblingAge.append(i)
                    for i in personLocation:
                        siblingLocation.append(i)
                    for i in personLocationOtherName:
                        siblingLocationOtherName.append(i)
                    for i in personNickname:
                        siblingNickname.append(i)
                    for i in personTelephone:
                        siblingTelephone.append(i)
                personName.clear()
                personDeceased.clear()
                personAge.clear()
                personLocation.clear()
                personLocationOtherName.clear()
                personNickname.clear()
                personTelephone.clear()

            nerChunks.clear()
            nerChunksCheck1.clear()
            nerChunksCheck2.clear()
            t = t + 1
```

Listing VI.5. Question 4 processing.

```
for entry in journeyAnalysisListCleaned:
    x = 0
    for question in questionMasterList:
        entryVecPt3 = text_to_vector(entry)
```

```
        questionVecPt2 = text_to_vector(question)
        cosinePt3 = get_cosine(entryVecPt3, questionVecPt2)
        if cosinePt3 < 0.95:
            x = x + 1
    if x == len(questionMasterList):
        journeyAnalysisList2.append(entry)
journeyAnalysisListFinal = [x.replace("/", " ").replace("\t", " ").replace("\n", "") for x in journeyAnalysisList2]
journeySentenceChunks = []
journeySentenceChunksTemp = []
journeyAnalysisListFinalString = " ".join(journeyAnalysisListFinal)
journeyWordTokens = sNLP.word_tokenize(journeyAnalysisListFinalString)
for i in journeyWordTokens:
    if i != ".":
        journeySentenceChunksTemp.append(i)
    else:
        journeySentenceChunksTemp.append(i)
        temp = " ".join(journeySentenceChunksTemp)
        journeySentenceChunks.append(temp)
        journeySentenceChunksTemp.clear()
journeySentenceChunksNER = []
for i in journeySentenceChunks:
    journeySentenceChunksNER.append(sNLP.ner(i))
date = 0
location = 0
journeyLocations = []
journeyLocationsTemp = []
journeyDates = []
journeyDatesTemp = []
locationHasDate = 0
dateHasLocation = 0
location1 = 1
date1 = 1
journeyChunksTemp = []
journeyChunks = []
journeyChunksSequentialTemp = []
journeyChunksSequential = []
x1 = 0
current = ""
trigger = 0
for sentence in journeySentenceChunksNER:
    location = 0
    location1 = 0
    location2 = 0
    date2 = 0
    date = 0
    date1 = 0
    firstIsChecker = 0
    journeyLocationsTemp.clear()
    journeyDatesTemp.clear()
    current = ""
    firstIs = ""
    for word in sentence:
        if (word[1] == "LOCATION" or word[1] == "COUNTRY" or word[1] == "CITY" or word[
            1] == "STATE_OR_PROVINCE") and location == 0:
            if current == "location":
                location1 = 0
                date1 = 0
                location2 = 0
                date2 = 0
                for i in journeyChunksSequential:
                    if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                        1] == "STATE_OR_PROVINCE") and firstIsChecker == 0: # checks whether the first thing in the
                        # list is a location or a date
                        firstIs = "location"
                        firstIsChecker = firstIsChecker + 1
                    if i[
                        1] == "DATE" and firstIsChecker == 0: # checks whether the first thing in the list is a
                        # location or a date
                        firstIs = "date"
                        firstIsChecker = firstIsChecker + 1
                    if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                        1] == "STATE_OR_PROVINCE") and location1 == 0:
                        location1 = location1 + 1
                        continue
                    if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                        1] == "STATE_OR_PROVINCE" or i[0] == ","):
                        location1 = location1 + 1
                    if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                        1] != "STATE_OR_PROVINCE" and i[0] != ",":
                        location2 = location2 + 1
                        location1 = 0
                    if i[1] == "DATE" and date1 == 0:
                        date1 = date1 + 1
                        continue
                    if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                        date1 = date1 + 1
                    if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                        date2 = date2 + 1
                        date1 = 0
                firstIsChecker = 0
                if date1 > 0:
                    date2 = date2 + 1
                    date1 = 0
                if location1 > 0:
                    location2 = location2 + 1
                    location1 = 0
                difference = abs(date2 - location2)
                if difference % 2 == 0: # this means difference is even
                    if firstIs == "location": # condition for if the first thing mentioned was a location then it
                        # will go " location, date, location, date ..... "
                        for i in journeyChunksSequential:
                            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                                1] == "STATE_OR_PROVINCE") and location1 == 0:
                                location1 = location1 + 1
                                journeyLocationsTemp.append(i[0])
```

```
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyLocationsTemp) != 0:
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
    else:
        for i in journeyChunksSequential:
            print(i)
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and (i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY"):
                print("HERE")
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
            date1 = 0
        if len(journeyLocationsTemp) != 0:
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
else: # this else case will present some strange cases potentially, if date is first that means it
    # goes "date, location, date" WEIRD
    if firstIs == "location":
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
```

```
                journeyDatesTemp.clear()
            if len(
                    journeyLocationsTemp) != 0: # i suspect this will be the case, since it is the case
                # of even pairs and it begins with location it will end with dates but wont be triggered
                # to end
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
            journeyDates.append(
                "no corresponding date found") # this is being done because we want equal length list
            # for dates and locations, since this iteration is dealing with odd lengths we have to add an
            # entry
        else:
            for i in journeyChunksSequential:
                if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                    1] == "STATE_OR_PROVINCE") and location1 == 0:
                    location1 = location1 + 1
                    journeyLocationsTemp.append(i[0])
                    continue
                if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                    1] == "STATE_OR_PROVINCE" or i[0] == ","):
                    location1 = location1 + 1
                    journeyLocationsTemp.append(i[0])
                if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                    1] != "STATE_OR_PROVINCE" and i[0] != ",":
                    temp = " ".join(journeyLocationsTemp)
                    journeyLocations.append(temp)
                    journeyLocationsTemp.clear()
                    location1 = 0
                if i[1] == "DATE" and date1 == 0:
                    date1 = date1 + 1
                    journeyDatesTemp.append(i[0])
                    continue
                if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                    date1 = date1 + 1
                    journeyDatesTemp.append(i[0])
                if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                    temp = " ".join(journeyDatesTemp)
                    journeyDates.append(temp)
                    journeyDatesTemp.clear()
                    date1 = 0
            if len(journeyLocationsTemp) != 0:
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
            if len(journeyDatesTemp) != 0:
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
            journeyLocations.append(
                "no corresponding location found") # this is being done because we want equal length
            # list for dates and locations, since this iteration is dealing with odd lengths we have to
            # add an entry
        journeyChunksSequential.clear()
        journeyLocationsTemp.clear()
        journeyDatesTemp.clear()
        firstIs = ""
    journeyLocationsTemp.append(word)
    location = location + 1
    current = "location"
    continue
if (word[1] == "LOCATION" or word[1] == "COUNTRY" or word[1] == "CITY" or word[1] == "STATE_OR_PROVINCE" or
        word[0] == ",") and location > 0:
    journeyLocationsTemp.append(word)
if (word[1] != "LOCATION" and word[1] != "COUNTRY" and word[1] != "CITY" and word[1] != "STATE_OR_PROVINCE" and
        word[0] != ",") and location > 0:
    for i in journeyLocationsTemp:
        journeyChunksSequential.append(i)
    journeyLocationsTemp.clear()
    location = 0
if word[1] == "DATE" and date == 0:
    if current == "date":
        location1 = 0
        date1 = 0
        location2 = 0
        date2 = 0
        for i in journeyChunksSequential: ########## THIS SHOULD GO DOWN BELOW (MODIFIED MAYBE) AT THE
            # CORRESPONDING LOCATION FOR CURRENT==" DATE " AND AT THE END OF THE INNER LOOP
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and firstIsChecker == 0: # checks whether the first thing in the
                # list is a location or a date
                firstIs = "location"
                firstIsChecker = firstIsChecker + 1
            if i[
                1] == "DATE" and firstIsChecker == 0: # checks whether the first thing in the list is a
                # location or a date
                firstIs = "date"
                firstIsChecker = firstIsChecker + 1
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                location2 = location2 + 1
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
```

```
        date2 = date2 + 1
        date1 = 0
firstIsChecker = 0
if date1 > 0:
    date2 = date2 + 1
    date1 = 0
if location1 > 0:
    location2 = location2 + 1
    location1 = 0
difference = abs(date2 - location2)
if difference % 2 == 0: # this means difference is even
    if firstIs == "location": # condition for if the first thing mentioned was a location then it
        # will go " location, date, location, date ..... "
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyLocationsTemp) != 0:
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
        if len(
                journeyDatesTemp) != 0: # i suspect this will be the case, since it is the case of
            # even pairs and it begins with location it will end with dates but wont be triggered to end
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
    else:
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
        if len(
                journeyLocationsTemp) != 0: # as with the case above, since this is the else case,
            # it implies it starts with dates so will end with locations
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
else: # this else case will present some strange cases potentially, if date is first that means it
    # goes "date, location, date" WEIRD
    if firstIs == "location":
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
```

```
                        temp = " ".join(journeyLocationsTemp)
                        journeyLocations.append(temp)
                        journeyLocationsTemp.clear()
                        location1 = 0
                    if i[1] == "DATE" and date1 == 0:
                        date1 = date1 + 1
                        journeyDatesTemp.append(i[0])
                        continue
                    if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                        date1 = date1 + 1
                        journeyDatesTemp.append(i[0])
                    if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                        temp = " ".join(journeyDatesTemp)
                        journeyDates.append(temp)
                        journeyDatesTemp.clear()
                        date1 = 0
                if len(journeyDatesTemp) != 0:
                    temp = " ".join(journeyDatesTemp)
                    journeyDates.append(temp)
                    journeyDatesTemp.clear()
                if len(
                        journeyLocationsTemp) != 0: # i suspect this will be the case, since it is the case
                    # of even pairs and it begins with location it will end with dates but wont be triggered
                    # to end
                    temp = " ".join(journeyLocationsTemp)
                    journeyLocations.append(temp)
                    journeyLocationsTemp.clear()
                journeyDates.append(
                    "no corresponding date found") # this is being done because we want equal length list
                # for dates and locations, since this iteration is dealing with odd lengths we have to add an
                # entry
            else:
                for i in journeyChunksSequential:
                    if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                        1] == "STATE_OR_PROVINCE") and location1 == 0:
                        location1 = location1 + 1
                        journeyLocationsTemp.append(i[0])
                        continue
                    if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                        1] == "STATE_OR_PROVINCE" or i[0] == ","):
                        location1 = location1 + 1
                        journeyLocationsTemp.append(i[0])
                    if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                        1] != "STATE_OR_PROVINCE" and i[0] != ",":
                        temp = " ".join(journeyLocationsTemp)
                        journeyLocations.append(temp)
                        journeyLocationsTemp.clear()
                        location1 = 0
                    if i[1] == "DATE" and date1 == 0:
                        date1 = date1 + 1
                        journeyDatesTemp.append(i[0])
                        continue
                    if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                        date1 = date1 + 1
                        journeyDatesTemp.append(i[0])
                    if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                        temp = " ".join(journeyDatesTemp)
                        journeyDates.append(temp)
                        journeyDatesTemp.clear()
                        date1 = 0
                if len(journeyLocationsTemp) != 0:
                    temp = " ".join(journeyLocationsTemp)
                    journeyLocations.append(temp)
                    journeyLocationsTemp.clear()
                if len(journeyDatesTemp) != 0:
                    temp = " ".join(journeyDatesTemp)
                    journeyDates.append(temp)
                    journeyDatesTemp.clear()
                journeyLocations.append(
                    "no corresponding location found") # this is being done because we want equal length
                # list for dates and locations, since this iteration is dealing with odd lengths we have to
                # add an entry
            journeyChunksSequential.clear()
            firstIs = ""
        journeyDatesTemp.append(word)
        date = date + 1
        current = "date"
        continue
    if (word[1] == "DATE" or word[1] == "NUMBER" or word[1] == "MONEY") and date > 0:
        journeyDatesTemp.append(word)
    if word[1] != "DATE" and word[1] != "MONEY" and word[1] != "NUMBER" and date > 0:
        for i in journeyDatesTemp:
            journeyChunksSequential.append(i)
        journeyDatesTemp.clear()
        date = 0
if len(
        journeyDatesTemp) != 0: # this if statement is for if the sentence ends and there is date entries that
    # couldnt close out yet
    for i in journeyDatesTemp:
        journeyChunksSequential.append(i)
    journeyDatesTemp.clear()
    date = 0
if len(
        journeyLocationsTemp) != 0: # this if statement is for if the sentence ends and there is location
    # entries that couldnt close out yet
    for i in journeyLocationsTemp:
        journeyChunksSequential.append(i)
    journeyLocationsTemp.clear()
    location = 0
if len(
        journeyChunksSequential) != 0: # this will be the case if either of the above two entries is the case,
    # but only one of the above could be true
    location1 = 0
    date1 = 0
    location2 = 0
```

```
date2 = 0
for i in journeyChunksSequential: ########## THIS SHOULD GO DOWN BELOW (MODIFIED MAYBE) AT THE CORRESPONDING
    # LOCATION FOR CURRENT==" DATE " AND AT THE END OF THE INNER LOOP
    if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
        1] == "STATE_OR_PROVINCE") and firstIsChecker == 0: # checks whether the first thing in the list is
        # a location or a date
        firstIs = "location"
        firstIsChecker = firstIsChecker + 1
    if i[
        1] == "DATE" and firstIsChecker == 0: # checks whether the first thing in the list is a location or
        # a date
        firstIs = "date"
        firstIsChecker = firstIsChecker + 1
    if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
        1] == "STATE_OR_PROVINCE") and location1 == 0:
        location1 = location1 + 1
        continue
    if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
        1] == "STATE_OR_PROVINCE" or i[0] == ","):
        location1 = location1 + 1
    if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
        1] != "STATE_OR_PROVINCE" and i[0] != ",":
        location2 = location2 + 1
        location1 = 0
    if i[1] == "DATE" and date1 == 0:
        date1 = date1 + 1
        continue
    if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
        date1 = date1 + 1
    if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
        date2 = date2 + 1
        date1 = 0
if date1 > 0:
    date2 = date2 + 1
    date1 = 0
if location1 > 0:
    location2 = location2 + 1
    location1 = 0
difference = abs(date2 - location2)
if difference % 2 == 0: # this means difference is even
    if firstIs == "location": # condition for if the first thing mentioned was a location then it will go "
        # location, date, location, date ....... "
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                current_place = i[0]
                location1 = location1 + 1
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ",") and i[0] != current_place:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                print('in')
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyLocationsTemp) != 0:
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
        if len(
                journeyDatesTemp) != 0: # i suspect this will be the case, since it is the case of even
            # pairs and it begins with location it will end with dates but wont be triggered to end
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
    else:
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
```

```
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
        if len(
                journeyLocationsTemp) != 0: # as with the case above, since this is the else case,
            # it implies it starts with dates so will end with locations
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
    if len(journeyDates) < len(journeyLocations):
        while len(journeyDates) < len(journeyLocations):
            journeyDates.append('no corresponding date found')
    if len(journeyLocations) < len(journeyDates):
        while len(journeyLocations) < len(journeyDates):
            journeyLocations.append('no corresponding location found')
else: # this else case will present some strange cases potentially, if date is first that means it goes
    # "date, location, date" WEIRD
    if firstIs == "location":
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
        if len(
                journeyLocationsTemp) != 0: # i suspect this will be the case, since it is the case of even
            # pairs and it begins with location it will end with dates but wont be triggered to end
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
        journeyDates.append(
            "no corresponding date found") # this is being done because we want equal length list for dates
        # and locations, since this iteration is dealing with odd lengths we have to add an entry
    else:
        for i in journeyChunksSequential:
            if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE") and location1 == 0:
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
                continue
            if location1 > 0 and (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[
                1] == "STATE_OR_PROVINCE" or i[0] == ","):
                location1 = location1 + 1
                journeyLocationsTemp.append(i[0])
            if location1 > 0 and i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[
                1] != "STATE_OR_PROVINCE" and i[0] != ",":
                temp = " ".join(journeyLocationsTemp)
                journeyLocations.append(temp)
                journeyLocationsTemp.clear()
                location1 = 0
            if i[1] == "DATE" and date1 == 0:
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
                continue
            if date1 > 0 and (i[1] == "DATE" or i[1] == "NUMBER" or i[1] == "MONEY"):
                date1 = date1 + 1
                journeyDatesTemp.append(i[0])
            if date1 > 0 and i[1] != "DATE" and i[1] != "NUMBER" and i[1] != "MONEY":
                temp = " ".join(journeyDatesTemp)
                journeyDates.append(temp)
                journeyDatesTemp.clear()
                date1 = 0
        if len(journeyLocationsTemp) != 0:
            temp = " ".join(journeyLocationsTemp)
            journeyLocations.append(temp)
            journeyLocationsTemp.clear()
        if len(journeyDatesTemp) != 0:
            temp = " ".join(journeyDatesTemp)
            journeyDates.append(temp)
            journeyDatesTemp.clear()
        journeyLocations.append(
            "no corresponding location found") # this is being done because we want equal length list for
        # dates and locations, since this iteration is dealing with odd lengths we have to add an entry
journeyChunksSequential.clear()
```

```
        firstIs = ""
        journeyDates = [x.replace("th", "") for x in journeyDates]
        if len(journeyDates) < len(journeyLocations):
            while len(journeyDates) < len(journeyLocations):
                journeyDates.append('no corresponding date found')
        if len(journeyLocations) < len(journeyDates):
            while len(journeyLocations) < len(journeyDates):
                journeyLocations.append('no corresponding location found')
    journeyDatesTemp.clear()
    journeyLocationsTemp.clear()


# from stackoverflow.com
def check_int(s):
    s = str(s)
    if s[0] in ('-', '+'):
        return s[1:].isdigit()
    return s.isdigit()


journeyDatesTokenizedTemp = []
yearTracker = []
i = -1
yearChecker = 0
for entry in journeyDates:
    i = i + 1
    journeyDatesTokenizedTemp.append(sNLP.word_tokenize(entry))
    for word in journeyDatesTokenizedTemp[0]:
        intChecker = check_int(word)
        if (intChecker == False) and (word != "no" and word != "corresponding" and word != "date" and word != "found" and
            word != "," and word != "/" and word != "July" and word != "JULY" and word != "JUL" and word != "June" and
            word != "JUNE" and word != "JUN" and word != "August" and word != "AUGUST" and word != "AUG" and
            word != "Aug" and word != "September" and word != "SEPTEMBER" and word != "SEPT" and word != "Sept" and
            word != "October" and word != "OCTOBER" and word != "OCT" and word != "Oct" and word != "November" and
            word != "NOVEMBER" and word != "NOV" and word != "Nov" and word != "December" and word != "DECEMBER" and
            word != "DEC" and word != "Dec" and word != "January" and word != "JANUARY" and word != "JAN" and
            word != "Jan" and word != "February" and word != "FEBRUARY" and word != "FEB" and word != "Feb" and
            word != "March" and word != "MARCH" and word != "MAR" and word != "Mar" and word != "April" and
            word != "APRIL" and word != "APR" and word != "Apr" and word != "May" and word != "MAY" and word != "June" and
            word != "JUNE" and word != "JUN" and word != "Jun"):
            yearChecker = yearChecker + 1
    if yearChecker == len(journeyDatesTokenizedTemp[0]):
        yearTracker.append(i)
    yearChecker = 0
    journeyDatesTokenizedTemp.clear()
journeyDatesTokenized = [sNLP.word_tokenize(x) for x in journeyDates]
for i in yearTracker:
    for j in range(len(journeyDates)):
        if i == j:
            journeyDates[i] = "no corresponding date found"
j = -1
locationTracker = []
for entry in journeyLocations:
    j = j + 1
    if entry == "no corresponding location found":
        locationTracker.append(j)
match_decreaser = 0
for entry in locationTracker:
    if journeyDates[entry - match_decreaser] == "no corresponding date found":
        del journeyDates[entry - match_decreaser]
        del journeyLocations[entry - match_decreaser]
        match_decreaser += 1
journeyDates.clear()
journeyDatesTemp.clear()
for entry in journeyDatesTokenized:
    print(len(entry))
    for word in entry:
        if (check_int(word) == True) or (word == "no" or word == "corresponding" or word == "date" or word == "found" or
            word == "," or word == "/" or word == "July" or word == "JULY" or word == "JUL" or word == "June" or
            word == "JUNE" or word == "JUN" or word == "August" or word == "AUGUST" or word == "AUG" or word == "Aug" or
            word == "September" or word == "SEPTEMBER" or word == "SEPT" or word == "Sept" or word == "October" or
            word == "OCTOBER" or word == "OCT" or word == "Oct" or word == "November" or word == "NOVEMBER" or
            word == "NOV" or word == "Nov" or word == "December" or word == "DECEMBER" or word == "DEC" or word == "Dec" or
            word == "January" or word == "JANUARY" or word == "JAN" or word == "Jan" or word == "February" or
            word == "FEBRUARY" or word == "FEB" or word == "Feb" or word == "March" or word == "MARCH" or word == "MAR" or
            word == "Mar" or word == "April" or word == "APRIL" or word == "APR" or word == "Apr" or word == "May" or
            word == "MAY" or word == "June" or word == "JUNE" or word == "JUN" or word == "Jun"):
            journeyDatesTemp.append(word)
    temp = " ".join(journeyDatesTemp)
    journeyDates.append(temp)
    journeyDatesTemp.clear()
journeyDatesTokenized.clear()
journeyDatesTokenized = [sNLP.word_tokenize(x) for x in journeyDates]
x = -1
for entry in journeyDatesTokenized:
    x = x + 1
    if len(entry) == 0:
        journeyDates[x] = "no corresponding date found"
        continue
    if len(entry) == 1 and check_int(entry[0]) == True:
        journeyDates[x] = "no corresponding date found"
```

Listing VI.6. Question 5 processing.

```
subjectGang = []
subjectMilitary = []
subjectGangMilitaryTokenized = [sNLP.word_tokenize(x) for x in subjectGangMilitaryAnalysisListFinal]
no = 0
for chunk in subjectGangMilitaryTokenized:
    for word in chunk:
        if word == "no" or word == "not" or word == "N/A" or word == "none" or word == "None":
```

```
            no = no + 1
        if (word == "military" or word == "MILITARY" or word == "Military") and no == 0:
            subjectMilitary.append("military")
        if (word == "police" or word == "Police" or word == "POLICE") and no == 0:
            subjectMilitary.append("police")
        if (word == "army" or word == "AMRY" or word == "Army") and no == 0:
            subjectMilitary.append("army")
        if (word == "navy" or word == "NAVY" or word == "Navy") and no == 0:
            subjectMilitary.append("navy")
        if (word == "air force" or word == "AIR FORCE" or word == "Air Force") and no == 0:
            subjectMilitary.append("air force")
        if (word == "Ms-13" or word == "MS13" or word == "MS-13" or word == "ms13" or word == "ms-13") and no == 0:
            subjectGang.append("MS-13")
        if (word == "gang" or word == "gangs") and no == 0:
            subjectGang.append("gang ties")
if len(subjectMilitary) == 0:
    subjectMilitary.append("no military/gov't service")
if len(subjectGang) == 0:
    subjectGang.append("no gang ties")
```

Listing VI.7. Question 6 processing.

```
relativeGang = []
relativeMilitary = []
cousinGang = []
cousinMilitary = []
fatherGang = []
fatherMilitary = []
motherGang = []
motherMilitary = []
brotherGang = []
brotherMilitary = []
sisterGang = []
sisterMilitary = []
sonGang = []
sonMilitary = []
daughterGang = []
daughterMilitary = []
nieceGang = []
nieceMilitary = []
nephewGang = []
nephewMilitary = []
siblingGang = []
siblingMilitary = []
spouseGang = []
spouseMilitary = []
childrenGang = []
childrenMilitary = []
relativeGangMilitaryTokenized = [x.replace("Father", "father").replace("Fathers", "father").replace("fathers",
                                                                       "father").replace(
    "FATHER", "father").replace("Brother", "brother").replace("Brothers", "brother").replace("BROTHER",
                                                                         "brother").replace(
    "BROTHERS", "brother").replace("Half-", "").replace("half-brother", "brother").replace("Half-Brother",
                                                                           "brother").replace(
    "brothers", "brother").replace("Mother", "mother").replace("mothers", "mother").replace("Mothers",
                                                                           "mother").replace(
    "MOTHER", "mother").replace("half-sister", "sister").replace("Sister", "sister").replace("sisters",
                                                                           "sister").replace(
    "Sisters", "sister").replace("SISTER", "sister").replace("SISTERS", "sister").replace("&",
                                                                         "and").replace(
    "bother", "brother").replace("(DECEASED)", "deceased").replace("(deceased)", "deceased").replace(
    "DECEASED", "deceased").replace("Deceased", "deceased").replace("yrs.", "").replace("yrs", "").replace(
    "YOA", "").replace("Step-Father", "father").replace("Step-Fathers", "father").replace("STEP-FATHER",
                                                                         "father").replace(
    "Step Father", "father").replace("Step Fathers", "father").replace("STEP FATHER", "father").replace(
    "Stepfather", "father").replace("stepfathers", "father").replace("stepfather", "father").replace("Siblings",
                                                                           "sibling").replace(
    "Cousin", "cousin").replace("Cousins", "cousin").replace("Sibling", "sibling").replace("siblings",
                                                                         "sibling").replace("Boy",
                                                                                            "son").replace(
    "boy", "son").replace("Girl", "daughter").replace("girl", "daughter").replace("Married", "married").replace(
    "Children", "children").replace("Husband", "husband").replace("Wife", "wife").replace("husband",
                                                                         "spouse").replace("wife",
                                                                                           "spouse").replace(
    "kids", "kid").replace("kid", "children").replace("married", "spouse").replace("", "").replace("Common law",
                                                                           "spouse").replace(
    "common law", "spouse").replace("Common Law", "spouse").replace("Spouse", "spouse") for x in
                               relativeGangMilitaryAnalysisListFinal]
relativeGangMilitaryTokenized = [sNLP.word_tokenize(x) for x in relativeGangMilitaryTokenized]
no = 0
for chunk in subjectGangMilitaryTokenized:
    cousin = 0
    father = 0
    mother = 0
    brother = 0
    sister = 0
    spouse = 0
    children = 0
    son = 0
    daughter = 0
    niece = 0
    nephew = 0
    sibling = 0
    for word in chunk:
        if word == "cousin":
            cousin = cousin + 1
        if word == "father":
            father = father + 1
        if word == "brother":
            brother = brother + 1
        if word == "sister":
```

```
        sister = sister + 1
    if word == "spouse":
        spouse = spouse + 1
    if word == "children":
        children = children + 1
    if word == "son":
        son = son + 1
    if word == "daughter":
        daughter = daughter + 1
    if word == "niece":
        niece = niece + 1
    if word == "nephew":
        nephew = nephew + 1
    if word == "sibling":
        sibling = sibling + 1
    if word == "no" or word == "not" or word == "N/A" or word == "none" or word == "None":
        no = no + 1
    if (word == "military" or word == "MILITARY" or word == "Military") and no == 0:
        relativeMilitary.append("military")
    if (word == "police" or word == "Police" or word == "POLICE") and no == 0:
        relativeMilitary.append("police")
    if (word == "army" or word == "AMRY" or word == "Army") and no == 0:
        relativeMilitary.append("army")
    if (word == "navy" or word == "NAVY" or word == "Navy") and no == 0:
        relativeMilitary.append("navy")
    if (word == "air force" or word == "AIR FORCE" or word == "Air Force") and no == 0:
        relativeMilitary.append("air force")
    if (word == "Ms-13" or word == "MS13" or word == "MS-13" or word == "ms13" or word == "ms-13") and no == 0:
        relativeGang.append("MS-13")
    if (word == "gang" or word == "gangs") and no == 0:
        relativeGang.append("gang ties")
if len(relativeMilitary) == 0:
    relativeMilitary.append("no military/gov't service")
if len(relativeGang) == 0:
    relativeGang.append("no gang ties")
if cousin > 0:
    for i in relativeMilitary:
        cousinMilitary.append(i)
    for i in relativeGang:
        cousinGang.append(i)
if father > 0:
    for i in relativeMilitary:
        fatherMilitary.append(i)
    for i in relativeGang:
        fatherGang.append(i)
if mother > 0:
    for i in relativeMilitary:
        motherMilitary.append(i)
    for i in relativeGang:
        motherGang.append(i)
if brother > 0:
    for i in relativeMilitary:
        brotherMilitary.append(i)
    for i in relativeGang:
        brotherGang.append(i)
if sister > 0:
    for i in relativeMilitary:
        sisterMilitary.append(i)
    for i in relativeGang:
        sisterGang.append(i)
if spouse > 0:
    for i in relativeMilitary:
        spouseMilitary.append(i)
    for i in relativeGang:
        spouseGang.append(i)
if children > 0:
    for i in relativeMilitary:
        childrenMilitary.append(i)
    for i in relativeGang:
        childrenGang.append(i)
if son > 0:
    for i in relativeMilitary:
        sonMilitary.append(i)
    for i in relativeGang:
        sonGang.append(i)
if daughter > 0:
    for i in relativeMilitary:
        daughterMilitary.append(i)
    for i in relativeGang:
        daughterGang.append(i)
if niece > 0:
    for i in relativeMilitary:
        nieceMilitary.append(i)
    for i in relativeGang:
        nieceGang.append(i)
if nephew > 0:
    for i in relativeMilitary:
        nephewMilitary.append(i)
    for i in relativeGang:
        nephewGang.append(i)
if sibling > 0:
    for i in relativeMilitary:
        siblingMilitary.append(i)
    for i in relativeGang:
        siblingGang.append(i)
relativeGang.clear()
relativeMilitary.clear()
```

Listing VI.8. Question 7, 8, and 9 processing. Form is identical in all three so only Question 7 is displayed.

```
fraudulentPPWPerson = []
fraudulentPPWPersonTemp = []
fraudulentPPWPersonNationality = []
fraudulentPPWLocation = []
fraudulentPPWLocationTemp = []
fraudulentPPWWhen = []
fraudulentPPWWhenTemp = []
fraudulentListNER = []
fraudulentPPWAnalysisListFinalTokenizedBySentence = []
fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp = []
fraudulentPPWAnalysisListFinalTokenized = []
fraudulentPPWAnalysisListFinalTokenized = [sNLP.word_tokenize(x) for x in fraudulentPPWAnalysisListFinal]
for i in fraudulentPPWAnalysisListFinalTokenized:
    for j in i:
        if j != ".":
            fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp.append(j)
        else:
            temp = " ".join(fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp)
            fraudulentPPWAnalysisListFinalTokenizedBySentence.append(temp)
            fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp.clear()
if len(fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp) != 0:
    temp = " ".join(fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp)
    fraudulentPPWAnalysisListFinalTokenizedBySentence.append(temp)
    fraudulentPPWAnalysisListFinalTokenizedBySentenceTemp.clear()
fraudulentListNER = [sNLP.ner(x) for x in fraudulentPPWAnalysisListFinalTokenizedBySentence]
x = 0
person = 0
location = 0
date = 0
x1 = 0
x2 = 0
for j in fraudulentListNER:
    location = 0
    date = 0
    person = 0
    for i in j:
        if i[1] == "PERSON" and x == 0:
            if len(fraudulentPPWLocation) != len(fraudulentPPWPerson):
                fraudulentPPWLocation.append("location not given")
            if len(fraudulentPPWWhen) != len(fraudulentPPWPerson):
                fraudulentPPWWhen.append("date for fraud ppw not given")
            if len(fraudulentPPWPersonNationality) != len(fraudulentPPWPerson):
                fraudulentPPWPersonNationality.append("nationality not given")
            location = 0
            date = 0
            fraudulentPPWPersonTemp.append(i[0])
            x = x + 1
            continue
        if i[1] == "PERSON" and x > 0:
            fraudulentPPWPersonTemp.append(i[0])
            x = x + 1
        if i[1] != "PERSON" and x > 0:
            temp = " ".join(fraudulentPPWPersonTemp)
            fraudulentPPWPerson.append(temp)
            fraudulentPPWPersonTemp.clear()
            x = 0
            person = person + 1
        if i[1] == "NATIONALITY" and person > 0:
            fraudulentPPWPersonNationality.append(i[0])
        if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[1] == "STATE_OR_PROVINCE") and len(
                fraudulentPPWLocation) < len(fraudulentPPWPerson) and x1 == 0:
            fraudulentPPWLocationTemp.append(i[0])
            x1 = x1 + 1
            continue
        if (i[1] == "COUNTRY" or i[1] == "LOCATION" or i[1] == "CITY" or i[1] == "STATE_OR_PROVINCE" or i[
            0] == ",") and x1 > 0:
            fraudulentPPWLocationTemp.append(i[0])
            x1 = x1 + 1
        if i[1] != "COUNTRY" and i[1] != "LOCATION" and i[1] != "CITY" and i[1] != "STATE_OR_PROVINCE" and i[
            0] != "," and x1 > 0:
            temp = " ".join(fraudulentPPWLocationTemp)
            fraudulentPPWLocation.append(temp)
            fraudulentPPWLocationTemp.clear()
            x1 = 0
            location = location + 1
        if i[1] == "DATE" and len(fraudulentPPWLocation) < len(fraudulentPPWPerson) and (check_int(i[0]) == True or i[
            0] == "July" or i[0] == "JULY" or i[0] == "JUL" or i[0] == "June" or i[0] == "JUNE" or i[0] == "JUN" or i[
            0] == "August" or i[0] == "AUGUST" or i[0] == "AUG" or i[0] == "Aug" or i[0] == "September" or i[
            0] == "SEPTEMBER" or i[0] == "SEPT" or i[0] == "Sept" or i[0] == "October" or i[0] == "OCTOBER" or i[
            0] == "OCT" or i[0] == "Oct" or i[0] == "November" or i[0] == "NOVEMBER" or i[0] == "NOV" or i[
            0] == "Nov" or i[0] == "December" or i[0] == "DECEMBER" or i[0] == "DEC" or i[0] == "Dec" or i[
            0] == "January" or i[0] == "JANUARY" or i[0] == "JAN" or i[0] == "Jan" or i[0] == "February" or i[
            0] == "FEBRUARY" or i[0] == "FEB" or i[0] == "Feb" or i[0] == "March" or i[0] == "MARCH" or i[0] == "MAR" or
                                                                                                        i[0] == "Mar" or i[
            0] == "April" or i[0] == "APRIL" or i[0] == "APR" or i[0] == "Apr" or i[0] == "May" or i[0] == "MAY" or i[
            0] == "June" or i[0] == "JUNE" or i[0] == "JUN" or i[0] == "Jun") and x2 == 0:
            fraudulentPPWWhenTemp.append(i[0])
            x2 = x2 + 1
            continue
        if (i[1] == "DATE" or i[1] == "NUMBER" or i[0] == "," or i[1] == "MONEY") and (check_int(i[0]) == True or i[
            0] == "/" or i[0] == "," or i[0] == "July" or i[0] == "JULY" or i[0] == "JUL" or i[0] == "June" or i[
            0] == "JUNE" or i[0] == "JUN" or i[0] == "August" or i[0] == "AUGUST" or i[0] == "AUG" or i[0] == "Aug" or
                                                                                                        i[0] == "September" or i[
            0] == "SEPTEMBER" or i[0] == "SEPT" or i[0] == "Sept" or i[0] == "October" or i[0] == "OCTOBER" or i[
            0] == "OCT" or i[0] == "Oct" or i[0] == "November" or i[0] == "NOVEMBER" or i[0] == "NOV" or i[
            0] == "Nov" or i[0] == "December" or i[0] == "DECEMBER" or i[0] == "DEC" or i[0] == "Dec" or i[
            0] == "January" or i[0] == "JANUARY" or i[0] == "JAN" or i[0] == "Jan" or i[0] == "February" or i[
            0] == "FEBRUARY" or i[0] == "FEB" or i[0] == "Feb" or i[0] == "March" or i[0] == "MARCH" or i[0] == "MAR" or
                                                                                                        i[0] == "Mar" or i[
            0] == "April" or i[0] == "APRIL" or i[0] == "APR" or i[0] == "Apr" or i[0] == "May" or i[0] == "MAY" or i[
            0] == "June" or i[0] == "JUNE" or i[0] == "JUN" or i[0] == "Jun") and x2 > 0:
            fraudulentPPWWhenTemp.append(i[0])
            x2 = x2 + 1
```

```
        if i[1] != "DATE" and i[1] != "NUMBER" and i[0] != "," and i[1] != "MONEY" and x2 > 0:
            temp = " ".join(fraudulentPPWWhenTemp)
            fraudulentPPWWhen.append(temp)
            fraudulentPPWWhenTemp.clear()
            x2 = 0
            date = date + 1
    if len(fraudulentPPWPersonTemp) != 0:
        temp = " ".join(fraudulentPPWPersonTemp)
        fraudulentPPWPerson.append(temp)
        fraudulentPPWPersonTemp.clear()
        x = 0
    if len(fraudulentPPWLocationTemp) != 0:
        temp = " ".join(fraudulentPPWLocationTemp)
        fraudulentPPWLocation.append(temp)
        fraudulentPPWLocationTemp.clear()
        x1 = 0
    if len(fraudulentPPWWhenTemp) != 0:
        temp = " ".join(fraudulentPPWWhenTemp)
        fraudulentPPWWhen.append(temp)
        fraudulentPPWWhenTemp.clear()
        x2 = 0
    if len(fraudulentPPWLocation) != len(fraudulentPPWPerson):
        fraudulentPPWLocation.append("location not given")
    if len(fraudulentPPWWhen) != len(fraudulentPPWPerson):
        fraudulentPPWWhen.append("date for fraud ppw not given")
    if len(fraudulentPPWPersonNationality) != len(fraudulentPPWPerson):
        fraudulentPPWPersonNationality.append("nationality not given")
    # this part deletes all entries associated with the subject himself
    i = -1
    for j in fraudulentPPWPerson:
        i += 1
        for t in nameChecker:
            if t == j:
                del fraudulentPPWPerson[i]
                del fraudulentPPWWhen[i]
                del fraudulentPPWLocation[i]
                del fraudulentPPWPersonNationality[i]
                break
    i = 0
    fraudulentPPWDateTokenizer = [sNLP.word_tokenize(x) for x in fraudulentPPWWhen]
    i = -1
    for j in fraudulentPPWDateTokenizer:
        i = i + 1
        if len(j) < 2:
            fraudulentPPWWhen[i] = "date for fraud ppw not given"
    i = 0
```

Listing VI.9. Question 10 processing.

```
if len(destinationAnalysisListFinal) != 0:
    r = 0
    subjectDestinationTemp = []
    subjectDestination = []
    destinationResponseNER = []
    for i in destinationAnalysisListFinal:
        destinationResponseNER.append(sNLP.ner(i))
    for i in destinationResponseNER:
        for k in i:
            if (k[1] == "CITY" or k[1] == "COUNTRY" or k[1] == "LOCATION" or k[
                1] == "STATE_OR_PROVINCE") and r == 0: # and r1 == 0: # and c==0:
                subjectDestinationTemp.append(k[0])
                r = r + 1
                continue
            if r > 0 and (k[0] == "," or k[1] == "CITY" or k[1] == "COUNTRY" or k[1] == "LOCATION" or k[
                1] == "STATE_OR_PROVINCE"): # and r1 == 0:
                subjectDestinationTemp.append(k[0])
            if (k[1] != "CITY" and k[1] != "COUNTRY" and k[1] != "LOCATION" and k[1] != "STATE_OR_PROVINCE") and k[
                0] != "or" and k[0] != "," and r > 0: # and r1 == 0: # and c>0:
                temp = " ".join(subjectDestinationTemp)
                subjectDestination.append(temp)
                subjectDestinationTemp.clear()
                r = 0
        if len(subjectDestinationTemp) != 0:
            temp = " ".join(subjectDestinationTemp)
            subjectDestination.append(temp)
            subjectDestinationTemp.clear()
            r = 0
    if len(subjectDestination) == 0:
        subjectDestination.append("subject did not specify a destination")
    subjectDestination = list(unique_everseen(subjectDestination))
    subjectDestinationFinal = []
    for i in subjectDestination:
        tempList = sNLP.word_tokenize(i)
        if tempList[len(tempList) - 1] == ",":
            del tempList[len(tempList) - 1]
            temp = " ".join(tempList)
            subjectDestinationFinal.append(temp)
        else:
            temp = " ".join(tempList)
            subjectDestinationFinal.append(temp)
    subjectDestination.clear()
    for i in subjectDestinationFinal:
        subjectDestination.append(i)
    subjectDestinationFinal.clear()
    subjectDestination = list(unique_everseen(subjectDestination))
```

A.4.

Listing VI.10. GUI Code.

```
def collapse_entries(maternal_last_name, paternal_last_name, middle_name, first_name, documentName):
    import itertools
    from more_itertools import unique_everseen
    collapse_entries.pID_list = []
    collapse_entries.pID_list.clear()
    # maternal AND paternal AND middle AND first (1)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE (maternalLastName=?) AND (paternalLastName=?) AND (middleName=?) '
        'AND (firstName=?)',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # maternal AND paternal AND middle AND first (1)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE (maternalLastName=? OR maternalLastName IS NULL) AND ('
        'paternalLastName=?) AND (middleName=?) AND (firstName=?)',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # this is meant to capture instance in a document where someone references single part name as first name of a
    # person they have already talked about

    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE firstName=?',
    # (paternal_last_name, )).fetchall())
    # maternal AND paternal AND (middle OR first) (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE (maternalLastName=?) AND (paternalLastName=?) AND ((middleName IS '
        'NULL OR middleName=?) OR (firstName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (maternal OR paternal) AND middle AND first (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (paternalLastName=?)) AND (middleName IS '
        'NULL OR middleName=?) AND (firstName=?)',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # maternal AND first AND (paternal OR middle) (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE (maternalLastName=?) AND (firstName=?) AND ((middleName IS NULL OR '
        'middleName=?) OR (paternalLastName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (maternal OR first) AND paternal AND middle (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (firstName=?)) AND (middleName IS NULL OR '
        'middleName=?) AND (paternalLastName=?)',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # maternal AND middle AND (paternal OR first) (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE (maternalLastName=?) AND (middleName IS NULL OR middleName=?) AND '
        '((paternalLastName=?) OR (firstName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (maternal OR middle) AND paternal AND first (2)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (middleName IS NULL OR middleName=?)) AND '
        '(paternalLastName=?) AND (firstName=?)',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (maternal OR middle) AND (paternal OR first) (3)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (middleName=?)) AND ((paternalLastName=?) '
        'OR (firstName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (middle OR paternal) AND (middle OR first) (3)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (paternalLastName=?)) AND ((middleName=?) '
        'OR (firstName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # (maternal OR first) AND (paternal OR middle) (3)
    collapse_entries.pID_list.append(c.execute(
        'SELECT personID FROM personID_table WHERE ((maternalLastName=?) OR (firstName=?)) AND ((middleName=?) OR ('
        'paternalLastName=?))',
        (maternal_last_name, paternal_last_name, middle_name, first_name)).fetchall())
    # stuff that will be commented out because the inclusion of null values picks up too many names
    # # maternal AND paternal AND middle AND first (1)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE (maternalLastName IS NULL
    # OR maternalLastName=?) AND (paternalLastName IS NULL OR paternalLastName=?) AND (middleName IS NULL OR
    # middleName=?) AND (firstName IS NULL OR firstName=?)', (maternal_last_name, paternal_last_name, middle_name,
    # first_name)).fetchall())
    # # maternal AND paternal AND (middle OR first) (2)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE (maternalLastName IS NULL
    # OR maternalLastName=?) AND (paternalLastName IS NULL OR paternalLastName=?) AND ((middleName IS NULL OR
    # middleName=?) OR (firstName IS NULL OR firstName=?))', (maternal_last_name, paternal_last_name, middle_name,
    # first_name)).fetchall())
    # # (maternal OR paternal) AND middle AND first (2)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
    # NULL OR maternalLastName=?) OR (paternalLastName IS NULL OR paternalLastName=?)) AND (middleName IS NULL OR
    # middleName=?) AND (firstName IS NULL OR firstName=?)', (maternal_last_name, paternal_last_name, middle_name,
    # first_name)).fetchall())
    # # maternal AND first AND (paternal OR middle) (2)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE (maternalLastName IS NULL
    # OR maternalLastName=?) AND (firstName IS NULL OR firstName=?) AND ((middleName IS NULL OR middleName=?) OR (
    # paternalLastName IS NULL OR paternalLastName=?))', (maternal_last_name, paternal_last_name, middle_name,
    # first_name)).fetchall())
    # # (maternal OR first) AND paternal AND middle (2)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
    # NULL OR maternalLastName=?) OR (firstName IS NULL OR firstName=?)) AND (middleName IS NULL OR middleName=?) AND
    # (paternalLastName IS NULL OR paternalLastName=?)', (maternal_last_name, paternal_last_name, middle_name,
    # first_name)).fetchall())
    # # maternal AND middle AND (paternal OR first) (2)
    # collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE (maternalLastName IS NULL
    # OR maternalLastName=?) AND (middleName IS NULL OR middleName=?) AND ((paternalLastName IS NULL OR
    # paternalLastName=?) OR (firstName IS NULL OR firstName=?))', (maternal_last_name, paternal_last_name,
    # middle_name, first_name)).fetchall())
    # # (maternal OR middle) AND paternal AND first (2)
```

```
# collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
# NULL OR maternalLastName=?) OR (middleName IS NULL OR middleName=?)) AND (paternalLastName IS NULL OR
# paternalLastName=?) AND (firstName IS NULL OR firstName=?)', (maternal_last_name, paternal_last_name,
# middle_name, first_name)).fetchall())
# # (maternal OR middle) AND (paternal OR first) (3)
# collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
# NULL OR maternalLastName=?) OR (middleName IS NULL OR middleName=?)) AND ((paternalLastName IS NULL OR
# paternalLastName=?) OR (firstName IS NULL OR firstName=?))', (maternal_last_name, paternal_last_name,
# middle_name, first_name)).fetchall())
# # (middle OR paternal) AND (middle OR first) (3)
# collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
# NULL OR maternalLastName=?) OR (paternalLastName IS NULL OR paternalLastName=?)) AND ((middleName IS NULL OR
# middleName=?) OR (firstName IS NULL OR firstName=?))',(maternal_last_name, paternal_last_name, middle_name,
# first_name)).fetchall())
# # (maternal OR first) AND (paternal OR middle) (3)
# collapse_entries.pID_list.append(c.execute('SELECT personID FROM personID_table WHERE ((maternalLastName IS
# NULL OR maternalLastName=?) OR (firstName IS NULL OR firstName=?)) AND ((middleName IS NULL OR middleName=?) OR
# (paternalLastName IS NULL OR paternalLastName=?))',(maternal_last_name, paternal_last_name, middle_name,
# first_name)).fetchall())
collapse_entries.pID_list = list(itertools.chain(*list(itertools.chain(*collapse_entries.pID_list))))
collapse_entries.pID_list = list(unique_everseen(collapse_entries.pID_list))
if len(collapse_entries.pID_list) == 0:
    collapse_entries.radio_button_selection = -1
    collapse_entries.suspect_selection = 0
    return collapse_entries.radio_button_selection, collapse_entries.suspect_selection
doc_list = []
doc_list_temp = []
doc_list.clear()
for entry in collapse_entries.pID_list:
    doc_list_temp.clear()
    corresponding_doc_Ids = c.execute('SELECT documentID FROM document_link_table WHERE (personID=?)',
                                      (entry,)).fetchall()
    corresponding_doc_Ids = [list(x) for x in corresponding_doc_Ids]
    corresponding_doc_Ids = list(itertools.chain(*corresponding_doc_Ids))
    corresponding_doc_Ids = list(unique_everseen(corresponding_doc_Ids))
    current_documentID = \
        c.execute('SELECT documentID FROM document_table WHERE (documentName=?)', (documentName,)).fetchone()[0]
    if current_documentID != None:
        while current_documentID in corresponding_doc_Ids:
            corresponding_doc_Ids.remove(current_documentID)
    doc_list.append(corresponding_doc_Ids)
counter = -1
counter_list = []
counter_list.clear()
for entry in doc_list[:]:
    counter += 1
    if len(entry) == 0:
        counter_list.append(counter)
        doc_list.remove(entry)
counter = 0
for entry in counter_list:
    del collapse_entries.pID_list[entry - counter]
    counter += 1
if len(doc_list) == 0 and len(collapse_entries.pID_list) == 0:
    collapse_entries.radio_button_selection = -1
    collapse_entries.suspect_selection = 0
    return collapse_entries.radio_button_selection, collapse_entries.suspect_selection
import os
master = Tk()
var = IntVar()

def callback_view_iterable(docID):
    document_name = c.execute('SELECT documentName FROM document_table WHERE documentID=?', (docID,)).fetchone()[0]
    os.startfile('C:/Users/Nathanael Beveridge/Documents/INTERVIEWS/' + document_name, 'open')

def callback_view(documentName):
    os.startfile(path + '/' + documentName, 'open')

current_name_list = [first_name, middle_name, paternal_last_name, maternal_last_name]
current_name = []
current_name.clear()
for name in current_name_list:
    if name != None:
        current_name.append(name)
current_name = ' '.join(current_name)
master.title('Are any of these the same person as ' + current_name)
master.minsize(300, 300)
master.geometry('800x800')
Label(master, text='Check box for if person is: \n' + current_name).grid(row=0, column=0, padx=25, pady=15)
Label(master, text='Person').grid(row=0, column=1, padx=25, pady=15)
Label(master, text='Click to view document in which \n person appears').grid(row=0, column=2, padx=25, pady=15)
Label(master,
     text='*Note: If ' + current_name + ' is not one \n of the people with check boxes displayed, '
                                        'check the first box. \n It corresponds to the individual in '
                                        'question.').grid(
    row=1, column=1, padx=25, pady=15)
suspectVar = IntVar()
Checkbutton(master, text='Suspect', variable=suspectVar, offvalue=0).grid(row=1, column=2, padx=25, pady=15)
suspectText = StringVar()
Entry(master, textvariable=suspectText).grid(row=2, column=2)
Label(master, text='Comments:').grid(row=2, column=1)
Radiobutton(master, variable=var, value=-1).grid(row=3, column=0, padx=10, pady=5) # button for current Name
Button(master, text='View', command=lambda: callback_view(documentName)).grid(row=3, column=2, padx=10,
                                                                               pady=5) # button for current name
Label(master, text=current_name, bg='white').grid(row=3, column=1, padx=5, pady=5) # label for current Name
Radiobutton(master, variable=var, value=-2).grid(row=4, column=0, padx=10, pady=5)
Label(master, text='Not a name that should be saved.', bg='white').grid(row=4, column=1, padx=10,
                                                                        pady=5) # label for current Name
maternal_text = StringVar()
Entry(master, textvariable=maternal_text).grid(row=5, column=1, padx=10, pady=5)
Label(master, text='Maternal Name Correction').grid(row=5, column=2, padx=10, pady=5)
paternal_text = StringVar()
Entry(master, textvariable=paternal_text).grid(row=6, column=1, padx=10, pady=5)
Label(master, text='Paternal Name Correction').grid(row=6, column=2, padx=10, pady=5)
middle_text = StringVar()
```

```
    Entry(master, textvariable=middle_text).grid(row=7, column=1, padx=10, pady=5)
    Label(master, text='Middle Name Correction').grid(row=7, column=2, padx=10, pady=5)
    first_text = StringVar()
    Entry(master, textvariable=first_text).grid(row=8, column=1, padx=10, pady=5)
    Label(master, text='First Name Correction').grid(row=8, column=2, padx=10, pady=5)
    Radiobutton(master, variable=var, value=-3).grid(row=5, column=0, padx=10,
                                            pady=5) # button for if new name entries have been added
    import tkinter.ttk
    tkinter.ttk.Separator(master, orient=HORIZONTAL).grid(row=9, columnspan=3, sticky='ew')
    total_length = 10 # start at 2 so that the first docID gets placed on the second third row
    iterator_for_pid = 0 # to keep track of which person is being referenced for documents
    for pid in collapse_entries.pID_list:
        related_name = []
        related_name.clear()
        number_of_docs_per_current_pid = 0
        maternal = c.execute('SELECT maternalLastName FROM personID_table WHERE personID=?', (pid,)).fetchone()[0]
        paternal = c.execute('SELECT paternalLastName FROM personID_table WHERE personID=?', (pid,)).fetchone()[0]
        middle = c.execute('SELECT middleName FROM personID_table WHERE personID=?', (pid,)).fetchone()[0]
        first = c.execute('SELECT firstName FROM personID_table WHERE personID=?', (pid,)).fetchone()[0]
        # making string of the name of the current person
        related_name_list = [first, middle, paternal, maternal]
        for name in related_name_list:
            if name != None:
                related_name.append(name)
        related_name = ' '.join(related_name)
        for did in doc_list[iterator_for_pid]:
            Button(master, text='View', command=lambda did=did: callback_view_iterable(did)).grid(
                row=total_length + number_of_docs_per_current_pid, column=2,
                pady=5) # button_name_iterator[button_name_counter]=
            number_of_docs_per_current_pid += 1
            interview_name = c.execute('SELECT documentName FROM document_table WHERE documentID=?', (did,)).fetchone()[
                0]
        Label(master, text=related_name).grid(row=total_length, column=1, pady=5)
        Radiobutton(master, variable=var, value=iterator_for_pid).grid(row=total_length, column=0, pady=5)
        iterator_for_pid += 1
        total_length = total_length + number_of_docs_per_current_pid
        tkinter.ttk.Separator(master, orient=HORIZONTAL).grid(row=total_length, columnspan=3, sticky='ew')
        total_length += 1
    # collapse_entries.radio_button_selection=var.get() # this assignes the selection to be an attribute of the function
    Button(master, text='Finish', command=lambda: master.destroy()).grid(row=1, column=0) # 'QUIT' button
    master.mainloop()
    collapse_entries.radio_button_selection = var.get()
    collapse_entries.radio_button_selection = var.get()
    collapse_entries.maternal_name_fn = maternal_text.get()
    collapse_entries.paternal_name_fn = paternal_text.get()
    collapse_entries.middle_name_fn = middle_text.get()
    collapse_entries.first_name_fn = first_text.get()
    collapse_entries.suspect_selection = suspectVar.get()
    collapse_entries.suspect_text = suspectText.get()
```

A.5.

Listing VI.11. SQL database entry code.

```
from tkinter import *
import sqlite3

conn = sqlite3.connect("C:/Users/Nathanael Beveridge/Documents/Interviews (.txt)/SQL Database Stuff/full_db_final.db")
conn.execute("PRAGMA foreign_keys = 1")
c = conn.cursor()
subjectNamePresent = 0
if len(subjectNameSQL) == 0 or len(subjectName) == 0:
    subjectNamePresent += 1
    subject_name_function(documentName)
    if not subject_name_function.subject_maternal_correction:
        maternalLastNameSQLEntry = None
    else:
        maternalLastNameSQLEntry = subject_name_function.subject_maternal_correction
    if not subject_name_function.subject_paternal_correction:
        paternalLastNameSQLEntry = None
    else:
        paternalLastNameSQLEntry = subject_name_function.subject_paternal_correction
    if not subject_name_function.subject_middle_correction:
        middleNameSQLEntry = None
    else:
        middleNameSQLEntry = subject_name_function.subject_middle_correction
    if not subject_name_function.subject_first_correction:
        firstNameSQLEntry = None
    else:
        firstNameSQLEntry = subject_name_function.subject_first_correction
    subjectNameSQL = [[]]
if len(subjectNameSQL[0]) > 4:
    subjectNamePresent += 1
    subject_name_function(documentName)
    if not subject_name_function.subject_maternal_correction:
        maternalLastNameSQLEntry = None
    else:
        maternalLastNameSQLEntry = subject_name_function.subject_maternal_correction
    if not subject_name_function.subject_paternal_correction:
        paternalLastNameSQLEntry = None
    else:
        paternalLastNameSQLEntry = subject_name_function.subject_paternal_correction
    if not subject_name_function.subject_middle_correction:
        middleNameSQLEntry = None
    else:
        middleNameSQLEntry = subject_name_function.subject_middle_correction
```

```
    if not subject_name_function.subject_first_correction:
        firstNameSQLEntry = None
    else:
        firstNameSQLEntry = subject_name_function.subject_first_correction
personID = None
# subjectName (MATERNAL, PATERNAL, MIDDLE, FIRST)
if subjectNamePresent == 0:
    if len(subjectNameSQL[0]) == 4:
        maternalLastNameSQLEntry = subjectNameSQL[0][3]
        paternalLastNameSQLEntry = subjectNameSQL[0][2]
        middleNameSQLEntry = subjectNameSQL[0][1]
        firstNameSQLEntry = subjectNameSQL[0][0]
    if len(subjectNameSQL[0]) == 3 and subjectNameSQL[0][1].isupper() == True:
        maternalLastNameSQLEntry = subjectNameSQL[0][2]
        paternalLastNameSQLEntry = subjectNameSQL[0][1]
        middleNameSQLEntry = None
        firstNameSQLEntry = subjectNameSQL[0][0]
    if len(subjectNameSQL[0]) == 3 and subjectNameSQL[0][1].isupper() == False:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = subjectNameSQL[0][2]
        middleNameSQLEntry = subjectNameSQL[0][1]
        firstNameSQLEntry = subjectNameSQL[0][0]
    if len(subjectNameSQL[0]) == 2:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = subjectNameSQL[0][1]
        middleNameSQLEntry = None
        firstNameSQLEntry = subjectNameSQL[0][0]
    if len(subjectNameSQL[0]) == 1:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = subjectNameSQL[0][0]
        middleNameSQLEntry = None
        firstNameSQLEntry = None
# interviewDate (input them as timestring in form "YYYY-MM-DD")
interviewDateSQLEntry = interviewDateSQL
# subjectDOBSQL
dateOfBirth = subjectDOBSQL
# subjectHeight
if len(subjectHeight) > 0:
    if subjectHeight[0] == "subject height not given":
        subjectHeightSQLEntry = None
    else:
        subjectHeightSQLEntry = subjectHeight[0]
# subjectWeight
if len(subjectWeight) > 0:
    if subjectWeight[0] == "subject weight not given":
        subjectWeightSQLEntry = None
    else:
        subjectWeightSQLEntry = subjectWeight[0]
subjectDeceased = 0
c.execute('INSERT OR IGNORE INTO document_table(documentID, documentName) VALUES(NULL, ?)', (documentName,))
conn.commit()
collapse_entries(maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
                 documentName)
suspect = collapse_entries.suspect_selection
if suspect == 1:
    suspectText = collapse_entries.suspect_text
if suspect == 0:
    suspectText = 'N/A'
if collapse_entries.radio_button_selection == -3: # updates what maternal, paternal, middle, and first names should
    # be according to input given
    if not collapse_entries.maternal_name_fn:
        maternalLastNameSQLEntry = None
    else:
        maternalLastNameSQLEntry = collapse_entries.maternal_name_fn
    if not collapse_entries.paternal_name_fn:
        paternalLastNameSQLEntry = None
    else:
        paternalLastNameSQLEntry = collapse_entries.paternal_name_fn
    if not collapse_entries.middle_name_fn:
        middleNameSQLEntry = None
    else:
        middleNameSQLEntry = collapse_entries.middle_name_fn
    if not collapse_entries.first_name_fn:
        firstNameSQLEntry = None
    else:
        firstNameSQLEntry = collapse_entries.first_name_fn
if collapse_entries.radio_button_selection == -1 or collapse_entries.radio_button_selection == -3 or \
            collapse_entries.radio_button_selection == -2: # or collapse_entries.original==-1:
    c.execute(
        "INSERT INTO personID_table(personID, maternalLastName, paternalLastName, middleName, firstName, "
        "interviewDate, subjectHeightInches, subjectWeightPounds, deceased, dateOfBirth, suspect, suspectText) "
        "VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
        (personID, maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
         interviewDateSQLEntry, subjectHeightSQLEntry, subjectWeightSQLEntry, subjectDeceased, dateOfBirth, suspect,
         suspectText))
    conn.commit()
    # subjectUserID
    cursor = c.execute('SELECT max(personID) FROM personID_table')
    subjectPersonID = cursor.fetchone()[0]
else:
    subjectPersonID = collapse_entries.pID_list[collapse_entries.radio_button_selection]
    name_length_check_current = [maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry,
                                 firstNameSQLEntry]
    maternal_last_name_check = \
        c.execute('SELECT maternalLastName FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    paternal_last_name_check = \
        c.execute('SELECT paternalLastName FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    middle_name_check = \
        c.execute('SELECT middleName FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    first_name_check = \
        c.execute('SELECT firstName FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    name_length_check_from_db = [maternal_last_name_check, paternal_last_name_check, middle_name_check,
                                 first_name_check]
    if maternalLastNameSQLEntry == None and maternal_last_name_check != None:
        maternalLastNameSQLEntry = maternal_last_name_check
```

```
    if paternalLastNameSQLEntry == None and paternal_last_name_check != None:
        paternalLastNameSQLEntry = paternal_last_name_check
    if middleNameSQLEntry == None and middle_name_check != None:
        middleNameSQLEntry = middle_name_check
    if firstNameSQLEntry == None and first_name_check != None:
        firstNameSQLEntry = first_name_check
    c.execute(
        'UPDATE personID_table SET maternalLastName=?, paternalLastName=?, middleName=?, firstName=? WHERE personID=?',
        (maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry, subjectPersonID))
    conn.commit()
    interview_date_check = \
        c.execute('SELECT interviewDate FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    if interview_date_check == None:
        c.execute('UPDATE personID_table SET interviewDate=? WHERE personID=?',
                  (interviewDateSQLEntry, subjectPersonID))
        conn.commit()
    subject_height_check = \
        c.execute('SELECT subjectHeightInches FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    if subject_height_check == None:
        c.execute('UPDATE personID_table SET subjectHeightInches=? WHERE personID=?',
                  (subjectHeightSQLEntry, subjectPersonID))
        conn.commit()
    subject_weight_check = \
        c.execute('SELECT subjectWeightPounds FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    if subject_weight_check == None:
        c.execute('UPDATE personID_table SET subjectWeightPounds=? WHERE personID=?',
                  (subjectWeightSQLEntry, subjectPersonID))
        conn.commit()
    subject_deceased_check = \
        c.execute('SELECT deceased FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    if subject_weight_check == None:
        c.execute('UPDATE personID_table SET deceased=? WHERE personID=?',
                  (subjectDeceased, subjectPersonID))
        conn.commit()
    subject_dob_check = \
        c.execute('SELECT dateOfBirth FROM personID_table WHERE personID=?', (subjectPersonID,)).fetchone()[0]
    if subject_dob_check == None:
        c.execute('UPDATE personID_table SET dateOfBirth=? WHERE personID=?',
                  (dateOfBirth, subjectPersonID))
        conn.commit()
    c.execute('UPDATE personID_table SET suspect=? WHERE personID=?', (suspect, subjectPersonID))
    conn.commit()
    c.execute('UPDATE personID_table SET suspectText=? WHERE personID=?', (suspectText, subjectPersonID))
    conn.commit()


# document_table and document relationship table function definition
def document_table_entry_for_subject(PERSONID, documentNAME):
    cursor = c.execute('SELECT documentID FROM document_table WHERE documentName=?', (documentNAME,))
    currentDocumentID = cursor.fetchone()[0]
    c.execute('INSERT OR IGNORE INTO document_link_table(personID, documentID) VALUES(?, ?)',
              (PERSONID, currentDocumentID))
    conn.commit()


def document_table_entry(PERSONID, documentNAME):
    c.execute('INSERT OR IGNORE INTO document_table(documentID, documentName) VALUES(NULL, ?)', (documentNAME,))
    conn.commit()
    cursor = c.execute('SELECT documentID FROM document_table WHERE documentName=?', (documentNAME,))
    currentDocumentID = cursor.fetchone()[0]
    c.execute('INSERT OR IGNORE INTO document_link_table(personID, documentID) VALUES(?, ?)',
              (PERSONID, currentDocumentID))
    conn.commit()


# subjectPersonID document tables
document_table_entry_for_subject(subjectPersonID, documentName)
COBLocationType = 1
destinationLocationType = 2
journeyLocationsLocationType = 3
relativeLocationType = 4
if len(subjectCOBSQL) > 0: # inserted so that if subjectCOB is not available it does not try to enter it
    if len(subjectCOBSQL[0]) != 0 and subjectCOB[0] != 'no COB provided':
        # establish locationNameID as a null variable to auto increment
        locationNameID = None
        numberOfLocations = len(subjectCOBSQL[0])
        subjectCOBLocationIDs = []
        subjectCOBLocationIDs.clear()
        # THIS RIGHT BELOW GENERATES THE LOCATION_NAME_TABLE and tracks the locationID's related to our subjectCOB
        for locationName in subjectCOBSQL[0]:
            c.execute('INSERT OR IGNORE INTO location_name_table(locationNameID, locationName) VALUES(?, ?)',
                      (locationNameID, locationName))
            conn.commit()
            cursor = c.execute('SELECT locationNameID FROM location_name_table WHERE locationName=?', (locationName,))
            maxLocationNameID = cursor.fetchone()[0]
            subjectCOBLocationIDs.append(maxLocationNameID)
        locationDateCOB = 'N/A'
        for locationNameID in subjectCOBLocationIDs:
            c.execute(
                'INSERT OR IGNORE INTO location_link_table(personID , locationNameID, locationTypeID, locationDate) '
                'VALUES(?, ?, ?, ?)',
                (subjectPersonID, locationNameID, COBLocationType, locationDateCOB))
            conn.commit()
# passport table stuff
if len(subjectPassport) > 0:
    if subjectPassport[0] != 'subject passport # not given':
        subjectPassportNumber = int(subjectPassport[0])
        passportID = None
        c.execute('INSERT OR IGNORE INTO passport_table(passportID, subjectPassport) VALUES(?, ?)',
                  (passportID, subjectPassportNumber))
        conn.commit()
        cursor = c.execute('SELECT passportID FROM passport_table WHERE subjectPassport=?', (subjectPassportNumber,))
        maxPassportID = cursor.fetchone()[0]
        c.execute('INSERT OR IGNORE INTO passport_link_table(personID, passportID) VALUES(?, ?)',
                  (subjectPersonID, maxPassportID))
```

```
        conn.commit()
# FINS table stuff
if len(subjectFINS) > 0:
    if subjectFINS[0] != 'subject FINS # not given':
        subjectFINSNumber = int(subjectFINS[0])
        FINSID = None
        c.execute('INSERT OR IGNORE INTO FINS_table(FINSID, subjectFINS) VALUES(?, ?)',
                  (FINSID, subjectFINSNumber))
        conn.commit()
        cursor = c.execute('SELECT FINSID FROM FINS_table WHERE subjectFINS=?', (subjectFINSNumber,))
        maxFINSID = cursor.fetchone()[0]
        c.execute('INSERT OR IGNORE INTO FINS_link_table(personID, FINSID) VALUES(?, ?)',
                  (subjectPersonID, maxFINSID))
        conn.commit()


def extra_phone_digit_removal(telephone_number):
    number = []
    for digit in telephone_number: # from stack overflow
        if check_int(digit) == True:
            number.append(digit)
    extra_phone_digit_removal.number = ''.join(number)
    return extra_phone_digit_removal.number


# telephone table stuff
if len(subjectTelephone) > 0:
    if subjectTelephone[0] != 'subject telephone not given':
        extra_phone_digit_removal(subjectTelephone[0])
        subjectTelephoneNumber = int(extra_phone_digit_removal.number)
        telephoneID = None
        c.execute('INSERT OR IGNORE INTO telephone_table(telephoneID, subjectTelephone) VALUES(?, ?)',
                  (telephoneID, subjectTelephoneNumber))
        conn.commit()
        cursor = c.execute('SELECT telephoneID FROM telephone_table WHERE subjectTelephone=?',
                           (subjectTelephoneNumber,))
        maxTelephoneID = cursor.fetchone()[0]
        c.execute("INSERT OR IGNORE INTO telephone_link_table(personID, telephoneID) VALUES(?, ?)",
                  (subjectPersonID, maxTelephoneID))
        conn.commit()
if len(subjectEmail) > 0:
    if subjectEmail[0] != 'no email provided':
        subjectEmailAcct = (subjectEmail[0])
        emailID = None
        c.execute('INSERT OR IGNORE INTO email_table(emailID, subjectEmail) VALUES(?, ?)',
                  (emailID, subjectEmailAcct))
        conn.commit()
        cursor = c.execute('SELECT emailID FROM email_table WHERE subjectEmail=?', (subjectEmailAcct,))
        maxEmailID = cursor.fetchone()[0]
        c.execute("INSERT OR IGNORE INTO email_link_table(personID, emailID) VALUES(?, ?)",
                  (subjectPersonID, maxEmailID))
        conn.commit()
if len(subjectFacebook) > 0:
    if subjectFacebook[0] != 'subject facebook not given':
        subjectFacebookAcct = (subjectFacebook[0])
        facebookID = None
        c.execute('INSERT OR IGNORE INTO facebook_table(facebookID, subjectFacebook) VALUES(?, ?)',
                  (facebookID, subjectFacebookAcct))
        conn.commit()
        cursor = c.execute('SELECT facebookID FROM facebook_table WHERE subjectFacebook=?', (subjectFacebookAcct,))
        maxFacebookID = cursor.fetchone()[0]
        c.execute("INSERT OR IGNORE INTO facebook_link_table(personID, facebookID) VALUES(?, ?)",
                  (subjectPersonID, maxFacebookID))
        conn.commit()
# family relation type stuff
# numbering below are the id numbers for each relation type from the relationship_type_table
fatherRelationType = 1
motherRelationType = 2
brotherRelationType = 3
sisterRelationType = 4
cousinRelationType = 5
siblingRelationType = 6
sonRelationType = 7
daughterRelationType = 8
childRelationType = 9
spouseRelationType = 10
helpToMXRelationType = 11
fraudulentPPWRelationType = 12
agentSmugglerRelationType = 13
generalRelationType = 14
familyRelationTypes = ['father', 'mother', 'brother', 'sister', 'cousin', 'sibling', 'son', 'daughter', 'child',
                       'spouse']
infoTypes = ['Name', 'Age', 'Deceased', 'Telephone', 'Location']
blank = 0 # the blank tracker below is used to continue on to the next family relation type if the
# relationType+'NameSQL' is empty
for relationType in familyRelationTypes:
    if len(eval(relationType + 'NameSQL')[0]) == 0:
        blank = blank + 1
    if blank > 0:
        blank = 0
        continue
    i = -1
    for entry in eval(
                    relationType + 'NameSQL'): # within each entry of the relationType+'NameSQL' collect all the
        # info that goes in the personID_table so it can be inserted and then the ID can be retrived to fill in the
        # relationship_table
        i = i + 1
        if len(entry) == 4:
            maternalLastNameSQLEntry = entry[3]
            paternalLastNameSQLEntry = entry[2]
            middleNameSQLEntry = entry[1]
            firstNameSQLEntry = entry[0]
        if len(entry) == 3 and entry[1].isupper() == True:
            maternalLastNameSQLEntry = entry[2]
            paternalLastNameSQLEntry = entry[1]
```

```
    middleNameSQLEntry = None
    firstNameSQLEntry = entry[0]
if len(entry) == 3 and entry[1].isupper() == False:
    maternalLastNameSQLEntry = None
    paternalLastNameSQLEntry = entry[2]
    middleNameSQLEntry = entry[1]
    firstNameSQLEntry = entry[0]
if len(entry) == 2:
    maternalLastNameSQLEntry = None
    paternalLastNameSQLEntry = entry[1]
    middleNameSQLEntry = None
    firstNameSQLEntry = entry[0]
if len(entry) == 1:
    maternalLastNameSQLEntry = None
    paternalLastNameSQLEntry = entry[0]
    middleNameSQLEntry = None
    firstNameSQLEntry = None
if eval(relationType + 'Age')[i] == 'age not given': # this gives the age of the relative
    relativeDOB = None
else:
    relativeDOB = eval(relationType + 'Age')[i]
if eval(relationType + 'Deceased')[
    i] == 'deceased': # gets binary response for whether or not the relative is deceased
    relativeDeceased = 1
else:
    relativeDeceased = 0
# now put this information into the personID_table, extract the person_ID, and then fill in
# relationships_table, telephone_table, and location_table
collapse_entries(maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
                 documentName)
suspect = collapse_entries.suspect_selection
if suspect == 1:
    suspectText = collapse_entries.suspect_text
if suspect == 0:
    suspectText = 'N/A'
if collapse_entries.radio_button_selection == -2: ######## so that if an entry is shown to the user that
    # should not be entered it can just be skipped
    continue
if collapse_entries.radio_button_selection == -3: # updates what maternal, paternal, middle, and first names
    # should be according to input given
    if not collapse_entries.maternal_name_fn:
        maternalLastNameSQLEntry = None
    else:
        maternalLastNameSQLEntry = collapse_entries.maternal_name_fn
    if not collapse_entries.paternal_name_fn:
        paternalLastNameSQLEntry = None
    else:
        paternalLastNameSQLEntry = collapse_entries.paternal_name_fn
    if not collapse_entries.middle_name_fn:
        middleNameSQLEntry = None
    else:
        middleNameSQLEntry = collapse_entries.middle_name_fn
    if not collapse_entries.first_name_fn:
        firstNameSQLEntry = None
    else:
        firstNameSQLEntry = collapse_entries.first_name_fn
if collapse_entries.radio_button_selection == -1 or collapse_entries.radio_button_selection == -3: # or
    # collapse_entries.original==-1:
    c.execute(
        'INSERT INTO personID_table(personID, maternalLastName, paternalLastName, middleName, firstName, '
        'interviewDate, subjectHeightInches, subjectWeightPounds, deceased, dateOfBirth, suspect, '
        'suspectText) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
        (None, maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry, None,
         None, None, relativeDeceased, relativeDOB, suspect, suspectText))
    conn.commit()
    cursor = c.execute('SELECT max(personID) FROM personID_table')
    currentRelativePersonID = cursor.fetchone()[
        0] # this is the personID of the current relative so we can fill in relationships_table,
    # telephone_table, and location_table
else:
    currentRelativePersonID = collapse_entries.pID_list[collapse_entries.radio_button_selection]
    name_length_check_current = [maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry,
                                 firstNameSQLEntry]
    maternal_last_name_check = c.execute('SELECT maternalLastName FROM personID_table WHERE personID=?',
                                         (currentRelativePersonID,)).fetchone()[0]
    paternal_last_name_check = c.execute('SELECT paternalLastName FROM personID_table WHERE personID=?',
                                         (currentRelativePersonID,)).fetchone()[0]
    middle_name_check = \
        c.execute('SELECT middleName FROM personID_table WHERE personID=?',
                  (currentRelativePersonID,)).fetchone()[
            0]
    first_name_check = \
        c.execute('SELECT firstName FROM personID_table WHERE personID=?',
                  (currentRelativePersonID,)).fetchone()[0]
    name_length_check_from_db = [maternal_last_name_check, paternal_last_name_check, middle_name_check,
                                 first_name_check]
    # if len(name_length_check_from_db)<len(name_length_check_current):
    if (maternalLastNameSQLEntry == None) and (maternal_last_name_check != None):
        maternalLastNameSQLEntry = maternal_last_name_check
    if (paternalLastNameSQLEntry == None) and (paternal_last_name_check != None):
        paternalLastNameSQLEntry = paternal_last_name_check
    if (middleNameSQLEntry == None) and (middle_name_check != None):
        middleNameSQLEntry = middle_name_check
    if (firstNameSQLEntry == None) and (first_name_check != None):
        firstNameSQLEntry = first_name_check
    c.execute(
        'UPDATE personID_table SET maternalLastName=?, paternalLastName=?, middleName=?, firstName=? WHERE '
        'personID=?',
        (maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
         currentRelativePersonID))
    conn.commit()
    current_relative_deceased_check = \
        c.execute('SELECT deceased FROM personID_table WHERE personID=?',
                  (currentRelativePersonID,)).fetchone()[0]
    if current_relative_deceased_check == None:
```

```
                c.execute('UPDATE personID_table SET deceased=? WHERE personID=?',
                          (relativeDeceased, currentRelativePersonID))
                conn.commit()
            current_relative_dob_check = \
                c.execute('SELECT dateOfBirth FROM personID_table WHERE personID=?',
                          (currentRelativePersonID,)).fetchone()[
                    0]
            if current_relative_dob_check == None:
                c.execute('UPDATE personID_table SET dateOfBirth=? WHERE personID=?',
                          (dateOfBirth, currentRelativePersonID))
                conn.commit()
            c.execute('UPDATE personID_table SET suspect=? WHERE personID=?', (suspect, currentRelativePersonID))
            conn.commit()
            c.execute('UPDATE personID_table SET suspectText=? WHERE personID=?',
                      (suspectText, currentRelativePersonID))
            conn.commit()
        # relationship_table (who 2 is to 1)
        c.execute(
            'INSERT OR IGNORE INTO relationship_table(personID1, personID2, relationshipTypeID, date, locationNameID) '
            'VALUES(?, ? , ?, "N/A", 1)',
            (subjectPersonID, currentRelativePersonID, eval(relationType + 'RelationType')))
        conn.commit()
        # current relativePerson document tables
        document_table_entry(currentRelativePersonID, documentName)
        # relative telephone table
        if eval(relationType + 'Telephone')[i] != 'no telephone number given':
            extra_phone_digit_removal(eval(relationType + 'Telephone')[i])
            relativeTelephone = int(extra_phone_digit_removal.number)
            c.execute('INSERT OR IGNORE INTO telephone_table(telephoneID, subjectTelephone) VALUES(?, ?)',
                      (None, relativeTelephone))
            conn.commit()
            cursor = c.execute('SELECT telephoneID FROM telephone_table WHERE subjectTelephone=?', (relativeTelephone,))
            currentRelativeTelephoneID = cursor.fetchone()[0]
            c.execute('INSERT OR IGNORE INTO telephone_link_table(personID, telephoneID) VALUES(?, ?)',
                      (currentRelativePersonID, currentRelativeTelephoneID))
            conn.commit()
        if len(eval(relationType + 'LocationSQL')[i]) != 0 and eval(relationType + 'LocationSQL')[i][
            0] != 'location not given':
            locationNameID = None
            relativeLocationIDs = []
            relativeLocationIDs.clear()
            # this should go through the list of the COB for the subject, insert it into the table if it is a new
            # place otherwise it will not be repeated, then we track what the locationNameID's were for our subjects
            # COB so we can enter them into the location link table
            # THIS RIGHT BELOW GENERATES THE LOCATION_NAME_TABLE and tracks the locationID's related to our subjectCOB
            for locationName in eval(relationType + 'LocationSQL')[
                i]: # done from the [0]th entry since all location SQL lists are lists of sublists to account for
                # when there are two brothers with multiple locations but here theres only one person,
                # so one location collection in the [0]th entry
                c.execute('INSERT OR IGNORE INTO location_name_table(locationNameID, locationName) VALUES(?, ?)',
                          (locationNameID, locationName))
                conn.commit()
                cursor = c.execute('SELECT locationNameID FROM location_name_table WHERE locationName=?',
                                   (locationName,))
                relativeLocationNameID = cursor.fetchone()[0]
                relativeLocationIDs.append(relativeLocationNameID)
            relativeLocationDate = interviewDateSQL
            for locationNameID in relativeLocationIDs:
                c.execute(
                    'INSERT OR IGNORE INTO location_link_table(personID , locationNameID, locationTypeID, '
                    'locationDate) VALUES(?, ?, ?, ?)',
                    (currentRelativePersonID, locationNameID, relativeLocationType, relativeLocationDate))
                conn.commit()
# i have it as an if statement so that it doesnt do anything if the subjectDestination is not collected
# insert the destination location into location_name_table
if len(subjectDestinationSQL) != 0:
    if len(subjectDestinationSQL[0]) != 0 and subjectDestination[0] != 'subject did not specify a destination':
        # establish locationNameID as a null variable to auto increment
        locationNameID = None
        numberOfLocations = len(subjectDestinationSQL[0])
        subjectDestinationLocationIDs = []
        subjectDestinationLocationIDs.clear()
        # this should go through the list of the COB for the subject, insert it into the table if it is a new place
        # otherwise it will not be repeated, then we track what the locationNameID's were for our subjects COB so we
        # can enter them into the location link table
        # THIS RIGHT BELOW GENERATES THE LOCATION_NAME_TABLE and tracks the locationID's related to our subjectCOB
        for locationName in subjectDestinationSQL[
            0]: # done from the [0]th entry since all location SQL lists are lists of sublists to account for when
            # there are two brothers with multiple locations but here theres only one person, so one location
            # collection in the [0]th entry
            c.execute('INSERT OR IGNORE INTO location_name_table(locationNameID, locationName) VALUES(?, ?)',
                      (locationNameID, locationName))
            conn.commit()
            cursor = c.execute('SELECT locationNameID FROM location_name_table where locationName=?', (locationName,))
            maxLocationNameID = cursor.fetchone()[0]
            subjectDestinationLocationIDs.append(maxLocationNameID)
        locationDateDestination = 'N/A'
        for locationNameID in subjectDestinationLocationIDs:
            c.execute(
                'INSERT OR IGNORE INTO location_link_table(personID, locationNameID, locationTypeID, locationDate) '
                'VALUES(?, ?, ?, ?)',
                (subjectPersonID, locationNameID, destinationLocationType, locationDateDestination))
            conn.commit()
# journeyLocations entry in locations_name_table and locations_link_table
i = -1
subjectJourneyLocationIDs = []
subjectJourneyLocationIDs.clear()
for entry in journeyLocationsSQL:
    i = i + 1
    if len(entry) == 1 and entry[
        0] == 'no corresponding location found': # we will keep track of times when no location is found but an
        # entry is input for date
        if journeyDatesSQL[i][0] == 'no month given':
            subjectJourneyDateCurrent = 'unknown'
        else:
```

```
            subjectJourneyDateCurrent = journeyDatesSQL[i][2] + '-' + journeyDatesSQL[i][0] + '-' + journeyDatesSQL[i][
                1]
        subjectJourneyLocationIDs.append(1)
    else:
        for location in entry:
            c.execute('INSERT OR IGNORE INTO location_name_table(locationNameID, locationName) VALUES(?, ?)',
                      (None, location))
            conn.commit()
            cursor = c.execute('SELECT locationNameID FROM location_name_table WHERE locationName=?', (location,))
            locationNameID = cursor.fetchone()[0]
            subjectJourneyLocationIDs.append(locationNameID)
        if journeyDatesSQL[i][0] == 'no month given':
            subjectJourneyDateCurrent = 'unknown'
        else:
            subjectJourneyDateCurrent = journeyDatesSQL[i][2] + '-' + journeyDatesSQL[i][0] + '-' + journeyDatesSQL[i][
                1]
    for locationNameID in subjectJourneyLocationIDs:
        c.execute(
            'INSERT OR IGNORE INTO location_link_table(personID, locationNameID, locationTypeID, locationDate) '
            'VALUES(?, ?, ?, ?)',
            (subjectPersonID, locationNameID, journeyLocationsLocationType, subjectJourneyDateCurrent))
        conn.commit()
outsidePeopleType = ['agentSmuggler', 'helpToMX', 'fraudulentPPW']
outsidePeopleLocationIDs = []
for relationType in outsidePeopleType:
    i = 0
    if len(eval(relationType + 'PersonSQL')[0]) == 0: # just changed this to
        continue
    for entry in eval(relationType + 'PersonSQL'):
        if len(entry) == 4:
            maternalLastNameSQLEntry = entry[3]
            paternalLastNameSQLEntry = entry[2]
            middleNameSQLEntry = entry[1]
            firstNameSQLEntry = entry[0]
        if len(entry) == 3 and entry[1].isupper() == True:
            maternalLastNameSQLEntry = entry[2]
            paternalLastNameSQLEntry = entry[1]
            middleNameSQLEntry = None
            firstNameSQLEntry = entry[0]
        if len(entry) == 3 and entry[1].isupper() == False:
            maternalLastNameSQLEntry = None
            paternalLastNameSQLEntry = entry[2]
            middleNameSQLEntry = entry[1]
            firstNameSQLEntry = entry[0]
        if len(entry) == 2:
            maternalLastNameSQLEntry = None
            paternalLastNameSQLEntry = entry[1]
            middleNameSQLEntry = None
            firstNameSQLEntry = entry[0]
        if len(entry) == 1:
            maternalLastNameSQLEntry = None
            paternalLastNameSQLEntry = entry[0]
            middleNameSQLEntry = None
            firstNameSQLEntry = None
        collapse_entries(maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
                         documentName)
        suspect = collapse_entries.suspect_selection
        if suspect == 1:
            suspectText = collapse_entries.suspect_text
        if suspect == 0:
            suspectText = 'N/A'
        if collapse_entries.radio_button_selection == -2: ######## so that if an entry is shown to the user that
            # should not be entered it can just be skipped
            continue
        if collapse_entries.radio_button_selection == -3: # updates what maternal, paternal, middle, and first names
            # should be according to input given
            if not collapse_entries.maternal_name_fn:
                maternalLastNameSQLEntry = None
            else:
                maternalLastNameSQLEntry = collapse_entries.maternal_name_fn
            if not collapse_entries.paternal_name_fn:
                paternalLastNameSQLEntry = None
            else:
                paternalLastNameSQLEntry = collapse_entries.paternal_name_fn
            if not collapse_entries.middle_name_fn:
                middleNameSQLEntry = None
            else:
                middleNameSQLEntry = collapse_entries.middle_name_fn
            if not collapse_entries.first_name_fn:
                firstNameSQLEntry = None
            else:
                firstNameSQLEntry = collapse_entries.first_name_fn
        if collapse_entries.radio_button_selection == -1 or collapse_entries.radio_button_selection == -3: # or
            # collapse_entries.original==-1:
            c.execute(
                'INSERT INTO personID_table(personID, maternalLastName, paternalLastName, middleName, firstName, '
                'interviewDate, subjectHeightInches, subjectWeightPounds, deceased, dateOfBirth, suspect, '
                'suspectText) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
                (None, maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry, None,
                 None, None, None, None, suspect, suspectText))
            conn.commit()
            cursor = c.execute('SELECT max(personID) FROM personID_table')
            currentOutsidePersonID = cursor.fetchone()[0]
        else:
            currentOutsidePersonID = collapse_entries.pID_list[collapse_entries.radio_button_selection]
            name_length_check_current = [maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry,
                                         firstNameSQLEntry]
            maternal_last_name_check = c.execute('SELECT maternalLastName FROM personID_table WHERE personID=?',
                                                 (currentOutsidePersonID,)).fetchone()[0]
            paternal_last_name_check = c.execute('SELECT paternalLastName FROM personID_table WHERE personID=?',
                                                 (currentOutsidePersonID,)).fetchone()[0]
            middle_name_check = c.execute('SELECT middleName FROM personID_table WHERE personID=?',
                                          (currentOutsidePersonID,)).fetchone()[0]
            first_name_check = c.execute('SELECT firstName FROM personID_table WHERE personID=?',
                                         (currentOutsidePersonID,)).fetchone()[0]
```

```
            name_length_check_from_db = [maternal_last_name_check, paternal_last_name_check, middle_name_check,
                                         first_name_check]
            if maternalLastNameSQLEntry == None and maternal_last_name_check != None:
                maternalLastNameSQLEntry = maternal_last_name_check
            if paternalLastNameSQLEntry == None and paternal_last_name_check != None:
                paternalLastNameSQLEntry = paternal_last_name_check
            if middleNameSQLEntry == None and middle_name_check != None:
                middleNameSQLEntry = middle_name_check
            if firstNameSQLEntry == None and first_name_check != None:
                firstNameSQLEntry = first_name_check
            c.execute(
                'UPDATE personID_table SET maternalLastName=?, paternalLastName=?, middleName=?, firstName=? WHERE '
                'personID=?',
                (maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
                 currentOutsidePersonID))
            conn.commit()
            c.execute('UPDATE personID_table SET suspect=? WHERE personID=?', (suspect, currentOutsidePersonID))
            conn.commit()
            c.execute('UPDATE personID_table SET suspectText=? WHERE personID=?', (suspectText, currentOutsidePersonID))
            conn.commit()
        # enter outsidePerson document info
        document_table_entry(currentOutsidePersonID, documentName)
        # collect info on date
        if eval(relationType + 'WhenSQL')[i][0] == 'no month given':
            outsidePersonDateCurrent = 'unknown'
        else:
            outsidePersonDateCurrent = eval(relationType + 'WhenSQL')[i][2] + '-' + eval(relationType + 'WhenSQL')[i][
                0] + '-' + eval(relationType + 'WhenSQL')[i][1]
        # collect info on location
        outsidePeopleLocationIDs.clear() # clear old stuff
        for locationNAME in eval(relationType + 'LocationSQL')[i]:
            if locationNAME == "location not given":
                outsidePeopleLocationIDs.append(1)
            else:
                c.execute('INSERT OR IGNORE INTO location_name_table(locationNameID, locationName) VALUES(NULL, ?)',
                          (locationNAME,))
                conn.commit()
                cursor = c.execute('SELECT locationNameID FROM location_name_table WHERE locationName=?',
                                   (locationName,))
                locationNameID = cursor.fetchone()[0]
                outsidePeopleLocationIDs.append(locationNameID)
        # relationship table filling the subject, the current outside personID (smuggler, forger, etc.) the
        # relationshipTypeID, date and locationNameID
        for outsidePPLLocationID in outsidePeopleLocationIDs:
            c.execute(
                'INSERT OR IGNORE INTO relationship_table(personID1, personID2, relationshipTypeID, date, '
                'locationNameID) VALUES(?, ?, ?, ?, ?)',
                (subjectPersonID, currentOutsidePersonID, eval(relationType + 'RelationType'), outsidePersonDateCurrent,
                 outsidePPLLocationID))
            conn.commit()
        if eval(relationType + 'PersonNationality')[i] != "nationality not given":
            c.execute(
                'INSERT OR IGNORE INTO nationality_type_table(nationalityTypeID, nationalityType) VALUES(NULL, ?)',
                (eval(relationType + 'PersonNationality')[i],))
            conn.commit()
            cursor = c.execute('SELECT nationalityTypeID FROM nationality_type_table WHERE nationalityType=?',
                               (eval(relationType + 'PersonNationality')[i],))
            nationalityID = cursor.fetchone()[0]
            c.execute('INSERT OR IGNORE INTO nationality_link_table(personID, nationalityTypeID) VALUES(?, ?)',
                      (currentOutsidePersonID, nationalityID))
            conn.commit()
        i = i + 1
subjectMilitaryGangRelationType = 1
relativeMilitaryGangRelationType = 2
cousinMilitaryGangRelationType = 3
fatherMilitaryGangRelationType = 4
brotherMilitaryGangRelationType = 5
motherMilitaryGangRelationType = 6
sisterMilitaryGangRelationType = 7
sonMilitaryGangRelationType = 8
daughterMilitaryGangRelationType = 10
nieceMilitaryGangRelationType = 11
nephewMilitaryGangRelationType = 12
siblingMilitaryGangRelationType = 13
spouseMilitaryGangRelationType = 14
childrenMilitaryGangRelationType = 15
airForceMilitaryGangOrgType = 1
navyMilitaryGangOrgType = 2
armyMilitaryGangOrgType = 3
policeMilitaryGangOrgType = 4
militaryMilitaryGangOrgType = 5
gangTiesMilitaryGangOrgType = 6
MS13MilitaryGangOrgType = 7
gangOrMilitaryRelationshipTypes = ['subject', 'relative', 'cousin', 'father', 'brother', 'mother', 'sister', 'son',
                                   'daughter', 'niece', 'nephew', 'sibling', 'spouse', 'children']
gangOrMilitaryTypes = ['air force', 'navy', 'army', 'police', 'military', 'gang ties', 'MS-13']
gangOrMilitaryTypesforSQLEntry = ['airForce', 'navy', 'army', 'police', 'military', 'gangTies', 'MS13']
for relationType in gangOrMilitaryRelationshipTypes:
    i = -1
    for organizationType in gangOrMilitaryTypes:
        i = i + 1
        if len(eval(relationType + 'Military')) == 1 and eval(relationType + 'Military') != "no military/gov't service":
            if eval(relationType + 'Military')[0] == organizationType:
                c.execute(
                    'INSERT OR IGNORE INTO gangORmilitary_link_table(personID, '
                    'gangOrMilitaryOrganizationRelationshipTypeID, gangOrMilitaryPersonRelationshipTypeID) VALUES(?, '
                    '?, ?)',
                    (subjectPersonID, eval(gangOrMilitaryTypesforSQLEntry[i] + 'MilitaryGangOrgType'),
                     eval(relationType + 'MilitaryGangRelationType')))
                conn.commit()
        if len(eval(relationType + 'Gang')) == 1 and eval(relationType + 'Gang') != "no gang ties":
            if eval(relationType + 'Gang')[0] == organizationType:
                c.execute(
                    'INSERT OR IGNORE INTO gangORmilitary_link_table(personID, '
```

```
                'gangOrMilitaryOrganizationRelationshipTypeID, gangOrMilitaryPersonRelationshipTypeID) VALUES(?, '
                '?, ?)',
                (subjectPersonID, eval(gangOrMilitaryTypesforSQLEntry[i] + 'MilitaryGangOrgType'),
                 eval(relationType + 'MilitaryGangRelationType')))
            conn.commit()
# names of people with just a general relationship to the subject
for person in relatedNamesSQL:
    if len(person) == 4:
        maternalLastNameSQLEntry = person[3]
        paternalLastNameSQLEntry = person[2]
        middleNameSQLEntry = person[1]
        firstNameSQLEntry = person[0]
    if len(person) == 3 and person[1].isupper() == True:
        maternalLastNameSQLEntry = person[2]
        paternalLastNameSQLEntry = person[1]
        middleNameSQLEntry = None
        firstNameSQLEntry = person[0]
    if len(person) == 3 and person[1].isupper() == False:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = person[2]
        middleNameSQLEntry = person[1]
        firstNameSQLEntry = person[0]
    if len(person) == 2:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = person[1]
        middleNameSQLEntry = None
        firstNameSQLEntry = person[0]
    if len(person) == 1:
        maternalLastNameSQLEntry = None
        paternalLastNameSQLEntry = person[0]
        middleNameSQLEntry = None
        firstNameSQLEntry = None
    # fill in the personID with the current outside person
    collapse_entries(maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
                     documentName)
    suspect = collapse_entries.suspect_selection
    if suspect == 1:
        suspectText = collapse_entries.suspect_text
    if suspect == 0:
        suspectText = 'N/A'
    if collapse_entries.radio_button_selection == -2:
        continue
    if collapse_entries.radio_button_selection == -3: # updates what maternal, paternal, middle, and first names
        # should be according to input given
        if not collapse_entries.maternal_name_fn:
            maternalLastNameSQLEntry = None
        else:
            maternalLastNameSQLEntry = collapse_entries.maternal_name_fn
        if not collapse_entries.paternal_name_fn:
            paternalLastNameSQLEntry = None
        else:
            paternalLastNameSQLEntry = collapse_entries.paternal_name_fn
        if not collapse_entries.middle_name_fn:
            middleNameSQLEntry = None
        else:
            middleNameSQLEntry = collapse_entries.middle_name_fn
        if not collapse_entries.first_name_fn:
            firstNameSQLEntry = None
        else:
            firstNameSQLEntry = collapse_entries.first_name_fn
    if collapse_entries.radio_button_selection == -1 or collapse_entries.radio_button_selection == -3: # or
        # collapse_entries.original==-1:
        c.execute(
            'INSERT INTO personID_table(personID, maternalLastName, paternalLastName, middleName, firstName, '
            'interviewDate, subjectHeightInches, subjectWeightPounds, deceased, dateOfBirth, suspect, suspectText) '
            'VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
            (
                None, maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry, None,
                None,
                None, None, None, suspect, suspectText))
        conn.commit()
        cursor = c.execute('SELECT max(personID) FROM personID_table')
        currentGeneralPersonID = cursor.fetchone()[0]
    else:
        currentGeneralPersonID = collapse_entries.pID_list[collapse_entries.radio_button_selection]
        name_length_check_current = [maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry,
                                     firstNameSQLEntry]
        maternal_last_name_check = c.execute('SELECT maternalLastName FROM personID_table WHERE personID=?',
                                             (currentGeneralPersonID,)).fetchone()[0]
        paternal_last_name_check = c.execute('SELECT paternalLastName FROM personID_table WHERE personID=?',
                                             (currentGeneralPersonID,)).fetchone()[0]
        middle_name_check = c.execute('SELECT middleName FROM personID_table WHERE personID=?',
                                      (currentGeneralPersonID,)).fetchone()[0]
        first_name_check = c.execute('SELECT firstName FROM personID_table WHERE personID=?',
                                     (currentGeneralPersonID,)).fetchone()[0]
        name_length_check_from_db = [maternal_last_name_check, paternal_last_name_check, middle_name_check,
                                     first_name_check]
        # if len(name_length_check_from_db)<len(name_length_check_current):
        if maternalLastNameSQLEntry == None and maternal_last_name_check != None:
            maternalLastNameSQLEntry = maternal_last_name_check
        if paternalLastNameSQLEntry == None and paternal_last_name_check != None:
            paternalLastNameSQLEntry = paternal_last_name_check
        if middleNameSQLEntry == None and middle_name_check != None:
            middleNameSQLEntry = middle_name_check
        if firstNameSQLEntry == None and first_name_check != None:
            firstNameSQLEntry = first_name_check
        c.execute(
            'UPDATE personID_table SET maternalLastName=?, paternalLastName=?, middleName=?, firstName=? WHERE '
            'personID=?',
            (maternalLastNameSQLEntry, paternalLastNameSQLEntry, middleNameSQLEntry, firstNameSQLEntry,
             currentGeneralPersonID))
        conn.commit()
        c.execute('UPDATE personID_table SET suspect=? WHERE personID=?', (suspect, currentGeneralPersonID))
        conn.commit()
        c.execute('UPDATE personID_table SET suspectText=? WHERE personID=?', (suspectText, currentGeneralPersonID))
```

```
    # general person document table
    document_table_entry(currentGeneralPersonID, documentName)
    c.execute(
        'INSERT OR IGNORE INTO relationship_table(personID1, personID2, relationshipTypeID, date, locationNameID) '
        'VALUES(?,?, 14, "N/A", 2)',
        (subjectPersonID, currentGeneralPersonID)) # date is "N/A" but not a fk, locaitonNameID has fk id 2
    conn.commit()
# facebook, email, passport, telephone entries with a general relationship to the subject
if len(relatedFacebook) != 0:
    for facebook in relatedFacebook:
        c.execute('INSERT OR IGNORE INTO facebook_table(facebookID, subjectFacebook) VALUES(?, ?)', (None, facebook))
        conn.commit()
        cursor = c.execute('SELECT facebookID FROM facebook_table where subjectFacebook=?', (facebook,))
        facebookGeneralID = cursor.fetchone()[0]
        c.execute('INSERT OR IGNORE INTO facebook_general_relationship_table(subjectPersonID, facebookID) VALUES(?, ?)',
                  (subjectPersonID, facebookGeneralID))
        conn.commit()
if len(relatedEmail) != 0:
    for email in relatedEmail:
        c.execute('INSERT OR IGNORE INTO email_table(emailID, subjectEmail) VALUES(?, ?)', (None, email))
        conn.commit()
        cursor = c.execute('SELECT emailID FROM email_table where subjectEmail=?', (email,))
        emailGeneralID = cursor.fetchone()[0]
        c.execute('INSERT OR IGNORE INTO email_general_relationship_table(subjectPersonID, emailID) VALUES(?, ?)',
                  (subjectPersonID, emailGeneralID))
        conn.commit()
if len(relatedPassport) != 0:
    for passport in relatedPassport:
        c.execute('INSERT OR IGNORE INTO passport_table(passportID, subjectPassport) VALUES(?, ?)', (None, passport))
        conn.commit()
        cursor = c.execute('SELECT passportID FROM passport_table where subjectPassport=?', (passport,))
        passportGeneralID = cursor.fetchone()[0]
        c.execute('INSERT OR IGNORE INTO passport_general_relationship_table(subjectPersonID, passportID) VALUES(?, ?)',
                  (subjectPersonID, passportGeneralID))
        conn.commit()
if len(relatedTelephone) != 0:
    for telephone in relatedTelephone:
        extra_phone_digit_removal(telephone)
        telephone_related = int(extra_phone_digit_removal.number)
        c.execute('INSERT OR IGNORE INTO telephone_table(telephoneID, subjectTelephone) VALUES(?, ?)',
                  (None, telephone_related))
        conn.commit()
        cursor = c.execute('SELECT telephoneID FROM telephone_table where subjectTelephone=?', (telephone_related,))
        telephoneGeneralID = cursor.fetchone()[0]
        c.execute(
            'INSERT OR IGNORE INTO telephone_general_relationship_table(subjectPersonID, telephoneID) VALUES(?, ?)',
            (subjectPersonID, telephoneGeneralID))
        conn.commit()
c.close()
conn.close()
```

Bibliography

1. O. Budget Division, "CBP Border Security Report," tech. rep., Department of Homeland Security, 2017.

2. U. Department of Homeland Security, "Department of Homeland Security Border Security Metrics Report May 1 2018," tech. rep., U.S. Department of Homeland Security, 2018.

3. R. Blumberg and S. Atre, "The Problem with Unstructured Data," *DM Review*, 2003.

4. P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction.," *Journal of the American Medical Informatics Association : JAMIA*, vol. 18, no. 5, pp. 544–51, 2011.

5. D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvistic Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

6. "Penn Treebank P.O.S. Tags." *Available at* ***https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html***. Accessed: 2018-12-02.

7. E. Jaynes, *Probability Theory the Logic of Science*. Cambridge University Press, 2003.

8. E. Jaynes, "Information Theory and Statistical Mechanics," *The Physical Review*, vol. 106, no. 4, pp. 620–630, 1957.

9. A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra, "A maximum entropy approach to natural language processing," *Comput. Linguist.*, vol. 22, pp. 39–71, Mar. 1996.

10. A. E. Borthwick, *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, New York University, New York, NY, USA, 1999.

11. J. R. Finkel, T. Grenager, and C. D. Manning, "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling," tech. rep., Stanford University, Stanford, CA.

12. C. Sutton and A. McCallum, "An introduction to conditional random fields," *Foundations and Trends in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2011.

13. J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.

14. A. Mykowiecka, M. Marciniak, and A. Kup, "Rule-based information extraction from patients clinical data," *Journal of Biomedical Informatics*, vol. 42, no. 5, pp. 923 – 936, 2009. Biomedical Natural Language Processing.

15. L. Chiticariu, Y. Li, and F. Reiss, "Rule-based information extraction is dead! long live rule-based information extraction systems!," in *EMNLP*, 2013.

16. C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.

17. M. Bekkar, D. Kheliouane Djemaa, and D. Akrouf Alitouche, "Evaluation Measures for Models Assessment over Imbalanced Data Sets," *Journal of Information Engineering and Applications*, vol. 3, no. 10, 2013.

18. I. R. Mansuri and S. Sarawagi, "Integrating unstructured data into relational databases," in *22nd International Conference on Data Engineering (ICDE'06)*, pp. 29–29, April 2006.

19. J. Cunha, J. Saraiva, and J. Visser, "From spreadsheets to relational databases and back," in *Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM '09, (New York, NY, USA), pp. 179–188, ACM, 2009.

20. D. R. Hipp, "SQLite." *Available at* `https://www.sqlite.org/`. Accessed: 2018-11-12.

21. F. Lundh, "Tkinter." *Available at* `https://wiki.python.org/moin/TkInter`. Accessed: 2019-01-17.

22. A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

23. L. C. Freeman, "Centrality in Social Networks Conceptual Clarification," *Social Networks*, vol. 179, pp. 215–239, 1978.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
21-03-2019	Master's Thesis	October 2017 – March 2019

4. TITLE AND SUBTITLE
Social Network Threat Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Beveridge, Nathanael R, 2nd Lt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
(AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-19-M-101

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USSOCOM SOCNORTH/Peterson AFB, Colorado.

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the
U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
Various government agencies have a stake in knowing when bad actors cross the United States' borders, or how bad actors may be involved in the flow of people across borders. Interviews conducted at border checkpoints with individuals who intend to cross the border can contain valuable information. The quantity of interviews is such that intelligence analysts could benefit greatly from an automation system that extracts the information they are looking for from within the interviews. This would allow them to focus more of their time on analyzing what is extracted as opposed to inspecting all interviews themselves. The information extracted can be written to an SQL database, allowing the information to then be easily and efficiently queried for valuable insight and analysis.

15. SUBJECT TERMS
Named entity recognition, human intelligence, SQL, networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
Unclassified					Lt Col Andrew Geyer (ENC)
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	90	19b. TELEPHONE NUMBER *(include area code)*
U	U	U			(937) 785-6565 x4584

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18