MOBILE FORENSICS:
GENERATING A USER DASHBOARD FROM EXTRACTED CELL PHONE DATA

Krista Balint

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Congdon School of Supply Chain, Business Analytics, and Information Systems

University of North Carolina Wilmington

2023

Approved by

Advisory Committee

_____	_____
Geoff Stoker	Jeff Cummings

_____
Ron Vetter, Chair

TABLE OF CONTENTS

Tables

Figures

ABSTRACT

Mobile Forensics: Generating a User Dashboard from Extracted Cell Phone Data. Balint, Krista 2023. Capstone Paper, University of North Carolina Wilmington.

Mobile forensics has become an increasingly prevalent branch of forensic science since the early 2000s as smartphone capabilities have advanced and begun to store a gold mine of user information. This collected data can be extracted and analyzed using forensic tools to tell a story about the user. In this paper, a proof-of-concept tool is developed to show what can be found through analysis of data found solely on a user's mobile device. Phone data obtained from a Cellebrite advanced logical extraction is uploaded to Physical Analyzer to generate HTML reports for the device. These reports are then uploaded to a dashboard that populates to display personal and demographic information (such as name, age, and city of residence), as well as a few usage statistics selected from collected user stories. The dashboard's accuracy was tested using extracted cell phone data from eight volunteers and the user interface was analyzed by students taking UNCW's digital forensics class.

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1: INTRODUCTION

The mobile forensics field has experienced tremendous growth since the early 2000s when smartphones began to gain popularity. As mobile devices have become increasingly common, their capabilities have become more advanced. The primary purpose of a mobile device was communication via a phone call or text message; but now, with abilities like downloading apps and searching the Internet, communication has become just one of the many tasks that users perform on their phones. Over 85% of the population has a smartphone with 6.92 billion users in the world (Turner, *How Many People*). The average time an adult spends on their smartphone per day is three hours and 43 minutes (Turner, *Average Screen Time*), with a daily average of 2,617 interactions (Turner, *50+ Smartphone Statistics*). Because our lives now revolve around our smartphones, the personal data that lies within them can be overwhelming. But what exactly can all this information tell us about the user?

In this paper, eight mobile devices underwent an advanced logical extraction using Cellebrite's Universal Forensic Extraction Device (UFED) and were then analyzed with Cellebrite's Physical Analyzer. The resulting HTML reports were uploaded to a proof-of-concept dashboard that extracted and displayed the user's personal and demographic information, as well as some usage statistics. The dashboard was written in Python, using an array of available libraries. The personal information section displayed data like the user's name, birthday, gender, and city of residence. User stories were collected to determine what additional usage statistics would be of interest to users, and three were selected for implementation: most frequent contacts communicated with, most common words sent in messages, and most frequent locations visited. The dashboard also contains additional information options and access to the raw reports.

CHAPTER 2: REVIEW OF LITERATURE REVIEW AND ANALYSIS

Digital & Mobile Forensics

Digital forensics, an area of forensic science focused on digital devices, became a recognized field in the 1990s with publications and policies emerging in the early 2000s ("Digital Forensics"). As technology has advanced, digital forensics has branched into many new subdivisions, from network forensics to vehicle forensics to mobile forensics. As shown in Figure 1, the annual smartphone sales have increased dramatically from 122 million in 2007 to 1.43 billion in 2021 ("Smartphone Sales").

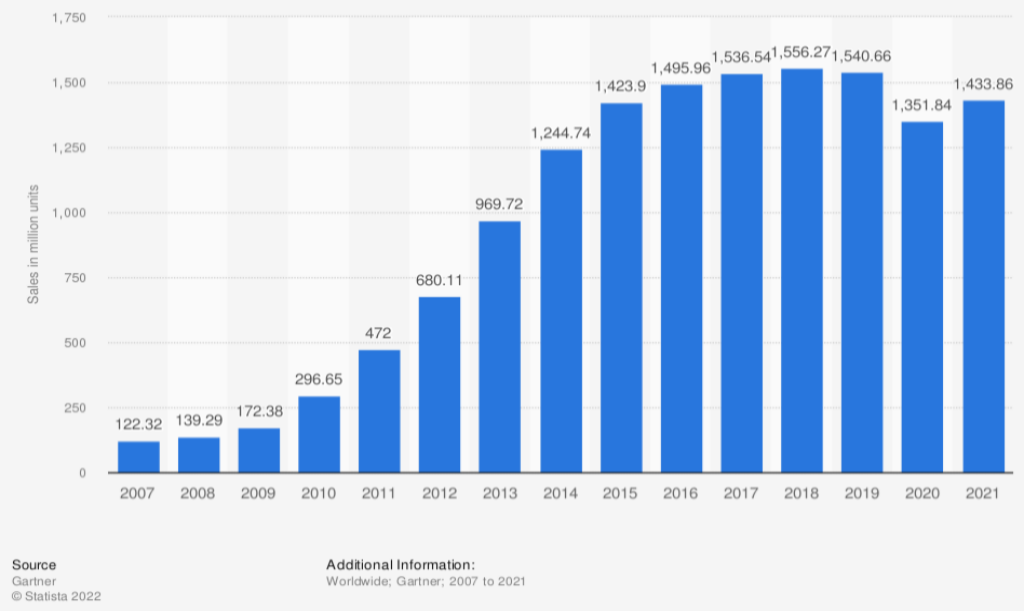


Figure 1. Number of smartphones sold to end users worldwide from 2007 to 2021 (in million units).

Because of the dramatic increase in smartphone popularity, the mobile forensics field has become increasingly vital. Mobile forensics is most simply defined as "the process of searching the contents of cell phones" (Easttom, 2022). As smartphone capabilities have advanced, so has the mobile data traffic per smartphone, averaging 1 GB/month in 2014 and 14.42 GB/month in 2022 ("Mobile data traffic"). Mobile devices contain information from a multitude of sources and have effectively come to replace

technologies such as “digital cameras, camcorders, book readers, newspapers, communication and navigation devices, portable game consoles, and even TV” (Afonin, 2016). Due to this expansion of smartphone capabilities, they often hold large amounts of user data. However, people are often unaware of how much data is collected and not adequately protected. A recent survey found that most respondents have inadequate security measures in place and follow poor security practices, with 14% of them commenting on their lack of security knowledge (Breitinger, 2020).

Available Software Tools

As the mobile forensics field has grown, so have the number of tools available to assist in data extraction and analysis. There are a wide variety of both open-source and commercial products available for the extraction and analysis of digital data. Open-source tools like Autopsy provide basic functionality for general digital forensics analyses. They strive to gather information and open it in a readable format to support standard analyses of digital devices (“About”). Commercial tools often have more advanced capabilities specific to mobile devices, such as MSAB’s XRY and Oxygen Forensics’ Detective. XRY supports more than 39,000 device profiles including non-standard mobile devices and provides full access to protected and deleted data (“XRY”). Oxygen’s Detective has unique features like image and facial categorization, as well as optical character recognition that converts words in images to text (“Mobile Forensic Solutions”).

This project utilizes the tool suite from Cellebrite, a leader in digital intelligence and investigative analytics whose products have been used in over five million investigations worldwide (“About – Cellebrite”). These tools allow for the collection, review, analysis, and management of sensitive data while ensuring it is accessible and secure. While Cellebrite offers over a dozen products, this project utilizes two: Physical

Analyzer and UFED 4PC.

Cellebrite's Physical Analyzer "is the market-leading solution for recovering and examining digital data." It includes an extensive set of tools to allow for a thorough examination of digital evidence and easy locating of key evidence. The Physical Analyzer's key capabilities include access to the widest range of apps, devices, and file formats, with application insights and a visual dashboard widget to help focus an initial examination. Additionally, a graphical timeline of events is generated, deleted data is reassembled into readable formats using advanced decoding, Android apps can be emulated using Virtual Analyzer, content can be reviewed in over 40 languages using Smart Translator, and it can create customized reports ("Cellebrite Physical Analyzer").

Cellebrite's Universal Forensic Extraction Device (UFED) is the market-leading solution for mobile device extractions and is "the industry standard for lawfully accessing and collecting digital data." UFED removes common barriers from investigations such as encryption and deleted content, allowing for the fullest possible picture and potential critical evidence to be produced from a device. Its key benefits include the efficient unlocking of devices, unsurpassed recovery methods, and support for the more than 31,000 device profiles, from mobile phones to drones to GPS devices. UFED 4PC, specifically, is a flexible and convenient software format allowing UFED access and extraction capabilities on any user's existing PC ("Cellebrite UFED").

With this software, users can perform several different types of common extractions including logical, file system, and physical. According to NIST guidelines, a logical extraction "involves capturing a copy of logical storage objects (e.g., directories and files) that reside on a logical store" (Ayers). File system extractions are when files embedded in memory are retrieved, along with hidden files and/or databases that may not

be visible in a logical extraction ("File System Extraction"). A physical extraction is more invasive as it "produces a low-level bit-by-bit copy of the phone's storage device" ("Physical Extraction"). Cellebrite also has an advanced logical option, which combines the capabilities of a file system and logical extraction and is often used as an alternative to a physical extraction ("Advanced Logical Extraction").

Literature Review

There is a substantial amount of literature published surrounding current digital forensics tools and their applications. Aljahdali et al. (2021) compared and analyzed several tools that are often used for mobile device forensics, including the open-source tool Autopsy, and commercial products Oxygen and UFED. Cellebrite's UFED is described as having almost all ideal functionality, checking the boxes for device identification, data extraction, messenger application analysis, data reporting, and data recovery. Oxygen meets all the ideal criteria as it has the same abilities as UFED while also being able to decode extracted data. Autopsy, however, only checked three boxes: extraction, messenger application analysis, and data recovery. It became clear that open-source tools are good, but paid tools have notable advantages. However, a similar analysis of digital forensics tools performed by Fernando (2021) noted some major issues currently facing commercial mobile forensics tools; most notably that these tools are rather expensive to obtain. Luckily, open-source tools are available for organizations that may not require all the bells and whistles of a commercial product. While both open-source and commercial tools have their respective benefits and drawbacks, mobile forensic tools in general are far from perfect. There is a lack of standardized documentation formatting for hex-dumping, or "the physical acquisition process of file systems in a mobile phone," as well as a lack of resource-sharing procedures in cloud

environments. Also, Fernando mentioned the difficulty of extracting deleted files in Apple iPhones because of their encryption key techniques but stated that tools are being experimented with to help recover such files.

Regarding Cellebrite tools specifically, Tajuddin and Manaf (2015) analyzed a Samsung Galaxy Note III using Cellebrite's UFED to "demonstrate the smartphone as a goldmine for investigators and as sources of digital evidence." The authors performed a physical extraction of the device resulting in 98,127 artifacts. They provided an in-depth description of what information was found including general device information, location data, call logs, contacts, emails, and more. Further, the authors included how the data could be analyzed to assist in investigations and reveal substantial personal information on the user.

Another study performed by Dragos and Schmeelk (2021) discussed the importance of location data, specifically Geographic Information Systems (GIS) data, in the digital forensics field and analyzed its role in current forensics curriculums. It was quickly discovered that there is little importance placed on GIS's role in the academic setting. The authors used Cellebrite's UFED and Physical Analyzer and had 140 students take the Cellebrite Certified Operator (CCO) certification exam to better understand how GIS data is currently represented in mobile forensics tools and training programs. After completing both the course and exam, 121 of the students noted that they felt satisfied with what they had learned from the experience, but it is suggested that further material be developed supporting the analysis of location information.

Additionally, there is a wide range of articles discussing the generation of user information using cell phone data. Auliya et al. (2021) stated that with the advancements being made in technology, we could "transform smartphone data into personal data by

generating user identification and user profiling." They performed a comprehensive review of the current literature regarding both topics, focusing on smartphone usage data, and concluded that "the current studies on this field generated a high accuracy and precision for user identification and user profiling." Another study was able to extract user demographic information including age and gender based solely on phone brand, phone model, and app usage (Yalcin, 2017). Similarly, Zhao et al. (2019) analyzed how user information can be derived from application use, stating that "apps on smartphones are a reflection of what users need, what they look like, what they are interested in, what activities they perform, how they live, etc." As smartphone data is often centered around one user, the authors found that user profiles can be effectively generated from app usage in areas including demographics, personality, psychological state, interests, and lifestyle.

Fartukov et al. (2021) discussed approaches to mobile user profiling using the plethora of personal data that can be extracted from mobile devices. The authors suggested analyzing data such as call logs, SMS logs, and application usage when predicting demographic information and analyzing web data with natural language processing to identify interests. Using this technique, the authors were able to obtain "demographic prediction accuracies on gender, marital status, and age as high as 97%, 94%, and 76% respectively." Further, O'Day et al. (2013) discussed how Python's Natural Language Toolkit (NLTK) could be used to analyze message data and identify the most frequently used words. This technique allows for the identification of specific words, in this case, drug-related or neutral, testing against a specified set of stop words.

While there have been a significant number of studies done on user profiling through extracted mobile device data, there is a lack of discussion around developing a comprehensive user profile specific to the owner of the device. Most studies tend to focus

on identifying one or a few characteristics like age, gender, and even emotional characteristics from large sets of phone data. The goal of this project is to develop a prototype of a tool that will generate a user profile that includes personal information and demographic data, as well as a few usage statistics, from a single device's data.

CHAPTER 3: METHODOLOGY

The mobile data dashboard allows iPhone users to see a picture of themselves painted by the data stored on their phones. Most people are unaware of how much data is kept on their mobile devices and how that data can be used. This dashboard will hopefully demonstrate how revealing the data on their devices can be and encourage users to follow better cybersecurity practices with their personal devices. User stories were collected to identify what usage statistics would be of interest to someone using the dashboard and three were chosen for implementation. These three include the contacts with the most interactions, the most common words used, and where the device has been.

Analysis

To gain a deeper understanding of what information can be gathered with digital forensic tools, an advanced logical extraction was performed on my iPhone using Cellebrite's UFED 4PC. To perform the extraction, the mobile device needed to be connected to the UFED Device Adapter using an iPhone-compatible cable. At the time of extraction, there was 67.3 GB of data on the device and 87 applications downloaded. A summary of the extraction process is depicted in Appendix A. The extraction took approximately 28 minutes to complete and Table 1 shows a few of the data types that were extracted from the device.

Further, once the UFED extraction was complete, there were detailed HTML reports generated using Cellebrite's Physical Analyzer. The home page of the HTML report produced by Physical Analyzer is shown in Appendix B.

Dashboard Development

After reviewing the data provided in the HTML reports, there were some noticeable gaps in the analysis provided. Most notably, no section explicitly listed

information about the user, such as name, age, gender, etc. If an extraction was conducted on an unknown mobile device, there would need to be significant analyses conducted to figure out the user's personal information. The mobile data dashboard is a prototype of a tool, written in Python, which can generate a dashboard showing user information using the extracted data's HTML reports. Python was selected for this project as it has a wide variety of libraries that are used to assist in the creation of the dashboard, analysis of the data, and development of the necessary graphics. The user profile includes basic information such as the user's name, email, birthday, gender, and city of residence. Further, there are visuals depicting the contacts that are most frequently communicated with, the most common words used, as well as an interactive map showing every location stored on the device. The dashboard also contains several optional features that display more in-depth information about specific aspects such as contacts and word usage. A wireframe of this dashboard prototype was developed using Balsamiq and is shown in Figure 2.

Table 1.

Data Types and Number of Records Found from Extraction

Data Type	Records Found
Contacts	2567
Chats	765
Calendar	1238
Locations	3351
Notes	71
IM	777
Call Logs	406
Data Files	84282

To generate the dashboard data, a wide variety of tools and techniques needed to

be utilized to analyze and collect data from the previously mentioned HTML reports. The import statements and global variables required for the dashboard are shown in Appendix C. The dashboard itself is constructed with the Streamlit library. The dashboard requires a CSV file upload that has the data set names and file paths for the HTML report, like the one shown in Figure 3. Once the CSV is uploaded, a drop-down box populates allowing the user to select which data set to analyze. There are also three checkboxes indicating whether the user wants to see additional data including a detailed contact report, word counts, and additional device information. Once the user clicks submit, the dashboard executes. The parsing of the HTML report that corresponds to the selected data set is done using the html2text library and this text is then analyzed to identify and display the data set owner's information. The Python functions that complete the CSV and HTML parsing are shown in Appendix D.

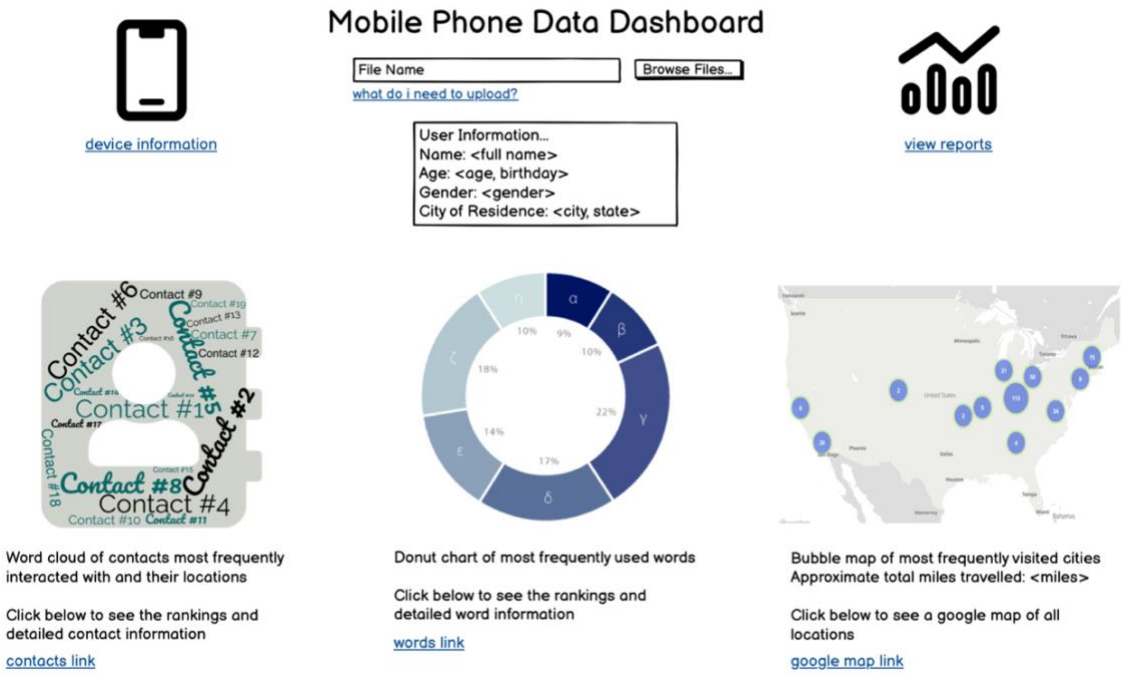


Figure 2. Mobile Phone Data Dashboard Wireframe made using Balsamiq.

To identify the user's full name, the user accounts associated with the device are analyzed to identify the most common name used. If no name is identified through user account data, the device name is checked as it is often renamed to be '<*user's name*>'s iPhone.' If this does not provide a name, the contacts are checked as a last resort, finding the name associated with the device's number. For birthday information, the contacts are searched to see if a birthday is associated with the user. If unsuccessful, an analysis is conducted on chat data to identify the date when the user receives the most messages containing the word 'birthday.' For gender analysis, a name categorization library, gender-guesser, is used with the first name identified earlier. The number of contacts is taken directly from the HTML reports using regex. To identify the user's email, the Apple ID associated with the device is taken from the HTML report. Lastly, the city of residence is determined by identifying the most common city visited within the past year. The area code of the phone number was going to be used for this, but they are often not representative of the user's current residence. The location data is analyzed using GeoPy's reverse geocoding to convert coordinates to their corresponding city and state value. The code for and result of the above analyses are shown in Appendices E and F, respectively.

Name	HTML filepath
Krista	D:\krista\2023-03-10.21-18-51\Apple_A2484 iPhone 13 Pro Max\Apple_A2484
Dataset #1	D:\Dataset #1\UFED Apple iPhone X 2023_02_23 (002)\AdvancedLogical File Sy
Dataset #2	D:\Dataset #2\2023-02-18.20-31-52\Apple_A2484 iPhone 13 Pro Max\Apple_A
Dataset #3	D:\Dataset #3\UFED Apple A2341 MGLW3LL A iPhone 12 Pro 2023_02_20 (001)
Dataset #4	D:\Dataset #4\2023-03-03.18-43-18\Apple_iPhone XR (A1984)\Apple_iPhone X
Dataset #5	D:\Dataset #5\2023-03-03.15-11-45\Apple_iPhone 7 (A1778)\Apple_iPhone 7 (
Dataset #6	D:\Dataset #6\2023-03-08.23-29-34\Apple_A2651 iPhone 14 Pro Max\Apple_A
Dataset #7	D:\Dataset #7\2023-03-12.21-07-03\Apple_iPhone 12 (A2172)\Apple_iPhone 1
Dataset #8	D:\Dataset #8\2023-03-12.23-38-19\Apple_iPhone 12 (A2172) _2\Apple_iPhone

Figure 3. CSV file containing data set names and the file paths to the HTML report.

To depict the most frequent contacts, a word cloud is generated using the Python wordcloud library. The largest name in the word cloud represents the most frequently communicated with contact and the color of the name indicates their estimated location. The contact names are associated with their phone number, and their location is estimated using the phonenumbers library. The word cloud, however, will only differentiate the color for the top 3 states for enhanced readability. Additionally, the dashboard user may select additional contact information to be displayed. The additional information will display a data frame containing the name, phone number, location, and the number of interactions for the top 20 contacts. These functions are provided in Appendix G and the result is shown in Appendix H.

Next, the most common words sent in chats are analyzed using Python's Natural Language Toolkit (NLTK) package. These words are cleaned by removing stop words and undesirable words and then visualized using a donut chart through Matplotlib's Pyplot library. There is also an option to display detailed word data, which would show a data frame with the top 20 words and how many times they have been used in chats. The code for and result of the word analysis are shown in Appendices I and J. Further, there is an interactive map that displays every location stored on the device using Streamlit's map feature. Additionally, the approximate total distance traveled will be shown below the map, which is found using the haversine library to calculate the approximate total mileage between all stored coordinates.

If the user wishes, they can also display additional device information. This includes the user's phone number, phone number origin state, manufacturer, device name, OS version, and time zone. This data is found in the HTML reports using regex, except for the origin state which analyzes the area code with GeoPy to get the origin

location. The Python functions and these dashboard features are shown in Appendices K and L, respectively.

The additional device information feature can also generate a table that displays all accounts associated with the device. These accounts were identified directly from the HTML reports using regex. Lastly, there is a button at the bottom of the dashboard that allows the user to view and inspect the HTML reports developed by the Physical Analyzer. This is depicted in Appendix M. The Python code using Streamlit to develop the dashboard with the analyzed information is provided in Appendix N. Also, a comprehensive list of the libraries used throughout the project is provided in Appendix O.

Dashboard Testing

To test the accuracy of this tool, eight individuals volunteered to have their phone data extracted using Cellebrite tools and then uploaded to the dashboard. Their device's data will remain private and be used solely to test the accuracy of the dashboard. Upon completion of the project, their data will be deleted. Further, to analyze the dashboard's user experience, students currently taking UNCW's digital forensics class have been asked to view a demonstration of the dashboard and respond to a short survey.

The accuracy of each dashboard feature was calculated by comparing the results produced from the user information section of the dashboard to the known information about the user. Among the eight devices that were used to test the dashboard's accuracy, there were only two issues identified. For one data set, the dashboard calculated the wrong city of residence, as the user recently moved. Another data set had no birthday saved in the contacts and had minimal chat data, so the birthday was unable to be identified. The accuracy for the three usage statistics is calculated assuming the data

provided by Cellebrite is representative of the user's behavior. The full accuracy report is shown in Table 2.

Table 2.

Dashboard Feature Accuracy

Dashboard Aspect	% Accurate
Name	100
Email	100
Number	100
Gender	100
Birthday	87.5
City of Residence	87.5
Phone Origin State	100
Top 20 Contacts	100*
Top 20 Words	100*
Locations Visited	100*

*Accuracy based on device information extracted by Cellebrite software

To test user experience, seventeen individuals were given a demonstration of the dashboard and asked to answer a short survey. The complete survey is shown in Appendix P. The first question asked the respondents to rate the ease of use of the dashboard using the following Likert scale: Very Easy, Easy, Just OK, Hard, Very Hard. All responses were either "Very Easy" (10) or "Easy" (7). Next, the survey asked if they would be interested in using the dashboard if available to them— all responses were "Yes." The survey ended with an optional comments section and some responses included "Cool that you could get the birthday from text messages" and "Overall, I think it's pretty interesting how thorough of a profile you can build on someone purely based on information stored on their phone."

CHAPTER 4: RESULTS AND DISCUSSION

Results & Analysis

The result of the mobile data dashboard was a successful proof-of-concept tool for developing user profiles based on cell phone data. All nine phones were able to be run through the dashboard and generate a basic user profile. The average time to run the dashboard with all features included is 416.3 seconds or about seven minutes. However, this time is greatly reduced when the city of residence feature is removed, dropping to 15.7 seconds, on average. Further, the time needed is dependent on how many locations the device has stored and the program needs to process. In this case, 239 device locations are being analyzed to identify the city of residence.

Lessons Learned

Mobile forensics takes a lot of time and patience from the analyst. Moreover, it requires significant memory and processing power from the device used to conduct the analysis. To complete this project, several external memory devices needed to be acquired, and weeks were spent extracting phone data and generating reports. All the data collected for this project was stored on a 1TB external hard drive that had 100 GB of storage space remaining. While Cellebrite's tools provide easy data extractions and generate detailed reports, they do not always work as expected. The tools crashed several times and failed to generate the needed reports, causing work to be delayed by days. Further, this project emphasized the importance of keeping backups. Towards the end of the project, the hard drive that held all nine phone extractions stopped working and all data was lost. The nature of this project, however, made it difficult to keep backups as one extraction contained almost 100 GB worth of data. Further, flash drives could not be used for storage as Cellebrite is not compatible with FAT 32 file systems.

The most interesting aspect of this project, however, was the interactions with different people throughout the process. To collect device data for testing, family, friends, and peers were asked if they would be willing to have their phone data extracted. Most individuals who were asked if they wanted to volunteer their phone data quickly denied after the details of the extraction process were explained. It was shocking to see how many people did not feel comfortable with other people seeing what they had on their mobile devices. Further, these individuals were then scared by the idea that their data could potentially be collected and used to generate a profile about them. Despite ensuring everyone that their data would only be uploaded to the dashboard for accuracy testing and then deleted upon completion of the project, only eight devices were able to be collected for testing. The advanced logical extraction performed on these eight devices required the device password to be entered several times throughout the process, and surprisingly all eight people simply told me their device password when asked to enter it the first time. Overall, this project allowed people to be educated on what type of data is being stored on their mobile devices and to consider what kinds of security practices they want to follow with their personal data.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Conclusions

Overall, the mobile data dashboard was an effective tool for developing a profile of a specific device's user. Also, it raised awareness about the kind of data that is being stored on these mobile devices.

Limitations

While this concept can be applied to any mobile device, this project was limited to iOS devices 7th generation or higher with users located in North America. Further, the dashboard was built assuming the input is a CSV file containing the file path to an HTML report generated for an advanced logical extraction by Cellebrite's Physical Analyzer. Additionally, it was hard to find more than eight devices for accuracy testing, as most people were unwilling to allow their device's data to be extracted. Lastly, as stated in the lessons learned section, there were time and storage restrictions present that limited the scope of the project.

Future Work

Several enhancements could be added to the mobile data dashboard. One issue that can be resolved is the length of time needed to calculate the city of residence—which takes, on average, six minutes and 40 seconds. This is because every location within the past year needs to be reverse-geolocated using an API that limits the number of calls per second. This issue could be resolved by using a more advanced API. Ideally, the visited locations feature can be made to reflect the true visited locations of the user rather than the device. As of now, the locations visited feature displays every location stored on the device, including locations from images that may have been sent to the user. For instance, one device's map indicated it had been to California, even though the user

hadn’t but knew a friend had and sent a picture to their device. This improvement could potentially be accomplished by using source information to detect whether images were taken with the device or sent to it.

Additionally, there were many good ideas suggested throughout this project for additional features. Some of these include determining relationships between the user and their contacts, listing which applications had passwords stored insecurely, and analyzing location data to determine hobbies or frequent activities. Further, the associated accounts feature could be expanded upon to then analyze data from the user’s social media accounts. Moreover, this dashboard could be expanded to support data from mobile devices other than Apple iPhones.

REFERENCES

1. “Advanced Logical Extraction - Mobile Device Forensics Archives.” Cellebrite, https://cellebrite.com/en/glossary/advanced-logical-extraction-mobile-device-forensics/. Accessed 30 Jan. 2023.
2. Afonin, Oleg. *Mobile Forensics - Advanced Investigative Strategies: Master Powerful Strategies to Acquire and Analyze Evidence from Real-Life Scenarios*. Packt Publishing, 2016.
3. Aljahdali, Asia, et al. “Mobile Device Forensics.” *Revista Română de Informatică Și Automatică*, vol. 31, no. 3, Sept. 2021, pp. 81–96. *DOI.org (Crossref)*, https://doi.org/10.33436/v31i3y202107.
4. Auliya, Syafira, et al. “A Review on Smartphone Usage Data for User Identification and User Profiling.” *Communications in Science and Technology*, vol. 6, no. 1, July 2021, pp. 25–34. *DOI.org (Crossref)*, https://doi.org/10.21924/cst.6.1.2021.363.
5. Ayers, Rick, et al. *Guidelines on Mobile Device Forensics*. NIST SP 800-101r1, National Institute of Standards and Technology, May 2014, p. NIST SP 800-101r1. *DOI.org (Crossref)*, https://doi.org/10.6028/NIST.SP.800-101r1.
6. Breitinger, Frank, et al. “A Survey on Smartphone User’s Security Choices, Awareness and Education.” *Computers & Security*, vol. 88, Jan. 2020, p. 101647. *DOI.org (Crossref)*, https://doi.org/10.1016/j.cose.2019.101647.
7. “Cellebrite Physical Analyzer.” Cellebrite, 10 Nov. 2022, https://cellebrite.com/en/physical-analyzer/. Accessed 30 Jan. 2023.
8. “Cellebrite UFED.” Cellebrite, 29 Nov. 2022, https://cellebrite.com/en/ufed. Accessed 30 Jan. 2023.

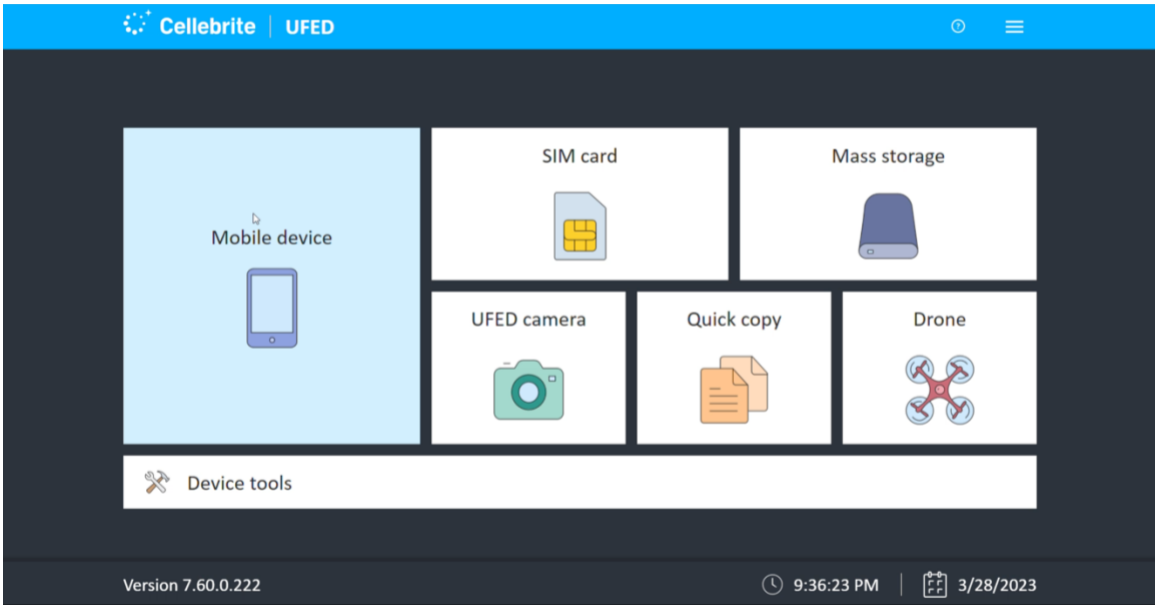
9. “Digital Forensics.” EC-Council, https://www.eccouncil.org/what-is-digital-forensics/. Accessed 30 Jan. 2023.
10. Dragos, Denise, and Suzanna Schmeelk. “Locating the Perpetrator: Industry Perspectives of Cellebrite Education and Roles of GIS Data in Cybersecurity and Digital Forensics.” *Intelligent Computing*, edited by Kohei Arai, vol. 285, Springer International Publishing, 2021, pp. 1041–50. *DOI.org (Crossref)*, https://doi.org/10.1007/978-3-030-80129-8_68.
11. Easttom, Chuck. *Digital Forensics, Investigation, and Response*. Fourth edition, Jones & Bartlett Learning, 2022.
12. Fartukov, Alexey M., et al. “Mobile User Profiling.” *Smart Algorithms for Multimedia and Imaging*, edited by Michael N. Rychagov et al., Springer International Publishing, 2021, pp. 259–75. *DOI.org (Crossref)*, https://doi.org/10.1007/978-3-030-66741-2_10.
13. Fernando, Vihara. “Cyber Forensics Tools: A Review on Mechanism and Emerging Challenges.” *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2021, pp. 1–7. *DOI.org (Crossref)*, https://doi.org/10.1109/NTMS49979.2021.9432641.
14. “File System Extraction - Mobile Device Forensics Archives.” Cellebrite, https://cellebrite.com/en/glossary/file-system-extraction-mobile-device-forensics/. Accessed 30 Jan. 2023.
15. “Mobile Data Traffic per Smartphone Worldwide 2014-2027.” *Statista*, https://www.statista.com/statistics/738977/worldwide-monthly-data-traffic-per-smartphone/. Accessed 30 Jan. 2023.

16. O'Day, Daniel R., and Ricardo A. Calix. "Text Message Corpus: Applying Natural Language Processing to Mobile Device Forensics." *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 2013, pp. 1–6. *IEEE Xplore*, https://doi.org/10.1109/ICMEW.2013.6618380.
17. *"*Mobile Forensic Solutions: Software and Hardware." Oxygen Forensics, https://www.oxygen-forensic.com/en/products/oxygen-forensic-detective. Accessed 30 Jan. 2023.
18. "Physical Extraction - Mobile Device Forensics Archives." Cellebrite, https://cellebrite.com/en/glossary/physical-extraction-mobile-device-forensics/. Accessed 30 Jan. 2023.
19. "Smartphone Sales Worldwide 2007-2021." Statista, https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. Accessed 30 Jan. 2023.
20. Tajuddin, Taniza Binti, and Azizah Abd Manaf. "Forensic Investigation and Analysis on Digital Evidence Discovery through Physical Acquisition on Smartphone." *2015 World Congress on Internet Security (WorldCIS)*, IEEE, 2015, pp. 132–38. *DOI.org (Crossref)*, https://doi.org/10.1109/WorldCIS.2015.7359429.
21. Turner, Ash. *50+ Smartphone Addiction Statistics & Phone Usage (2023)*. 4 June 2018, https://www.bankmycell.com/blog/smartphone-addiction/.
22. ---. Average Screen Time On IPhone & Android (Feb 2023). 13 Jan. 2022, https://www.bankmycell.com/blog/average-screen-time-on-iphone-android.
23. ---. *How Many People Have Smartphones Worldwide (Feb 2023)*. 10 July 2018, https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

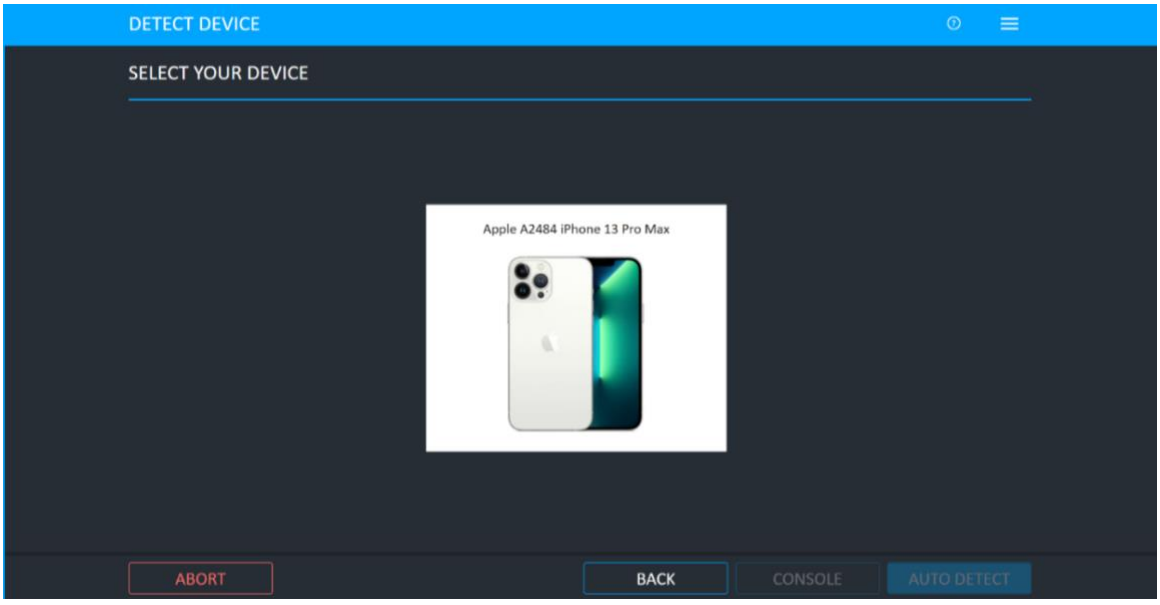
24. “About.” The Sleuth Kit and Autopsy, https://www.sleuthkit.org/about.php. Accessed 30 Jan. 2023.
25. “About – Cellebrite.” Cellebrite, https://www.cellebrite.com/en/about/. Accessed 30 Jan. 2023.
26. “XRY - Mobile Data Extraction Software - Phone Data Recovery Software.” MSAB, https://www.msab.com/product/xry-extract/. Accessed 30 Jan. 2023.
27. Yalcin, Hulya. “Extracting User Profiles from Mobile Data.” *2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2017, pp. 1–5. *IEEE Xplore*, https://doi.org/10.1109/BlackSeaCom.2017.8277701.
28. Zhao, Sha, et al. “User Profiling from Their Use of Smartphone Applications: A Survey.” *Pervasive and Mobile Computing*, vol. 59, Oct. 2019, p. 101052. *DOI.org (Crossref)*, https://doi.org/10.1016/j.pmcj.2019.101052.

APPENDIX A

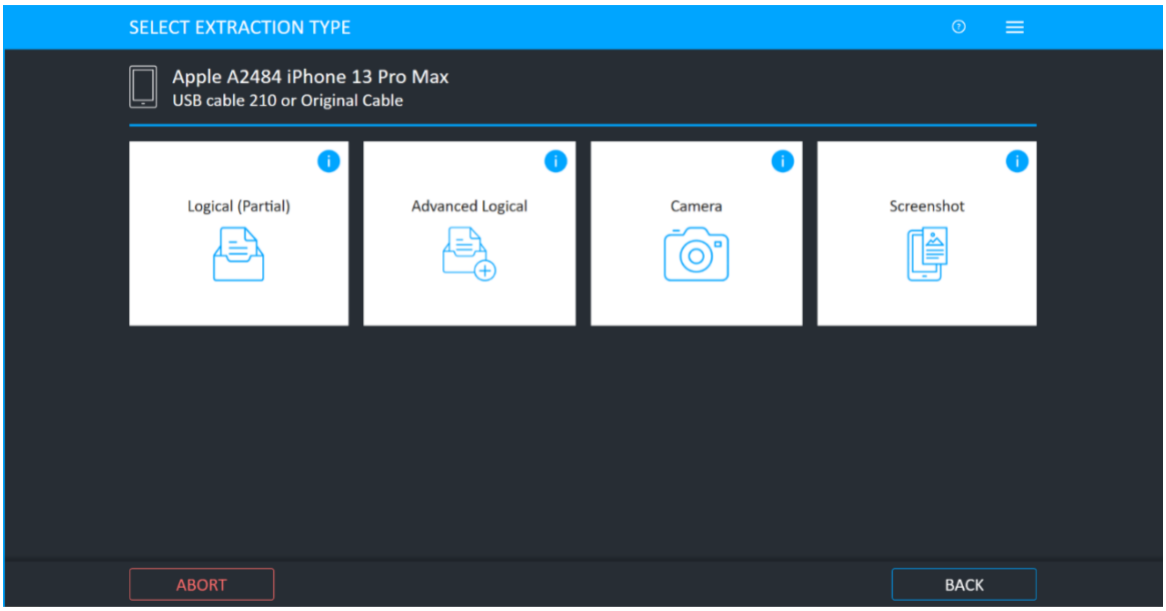
Advanced logical extraction steps using Cellebrite's UFED 4PC.



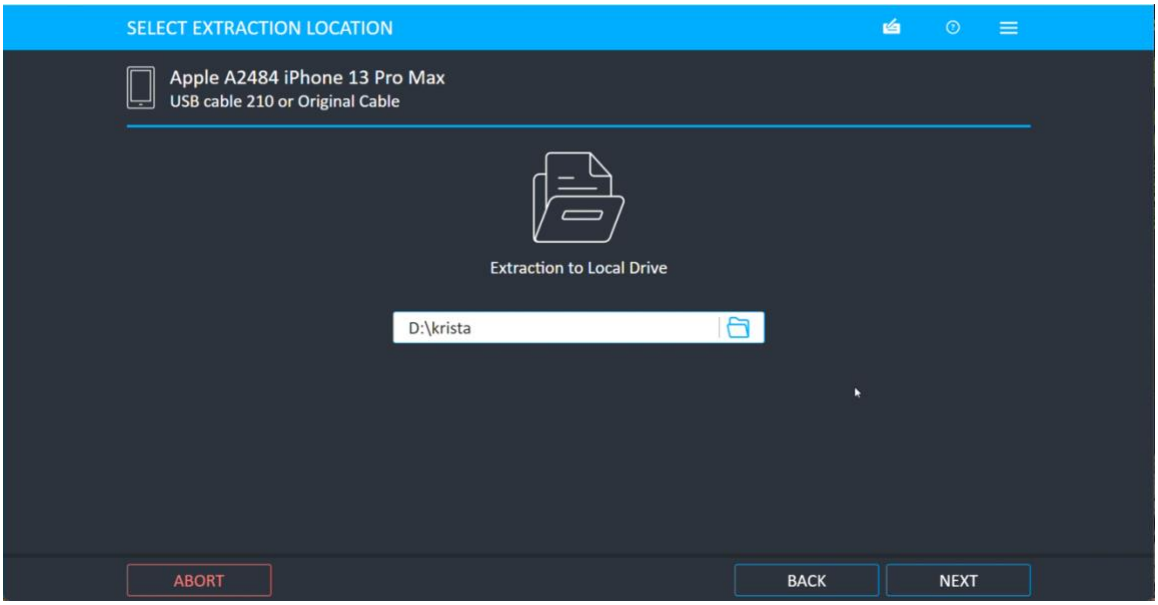
Home screen of UFED 4PC. Click 'Mobile device' and then 'Auto Detect.'



Select the device.

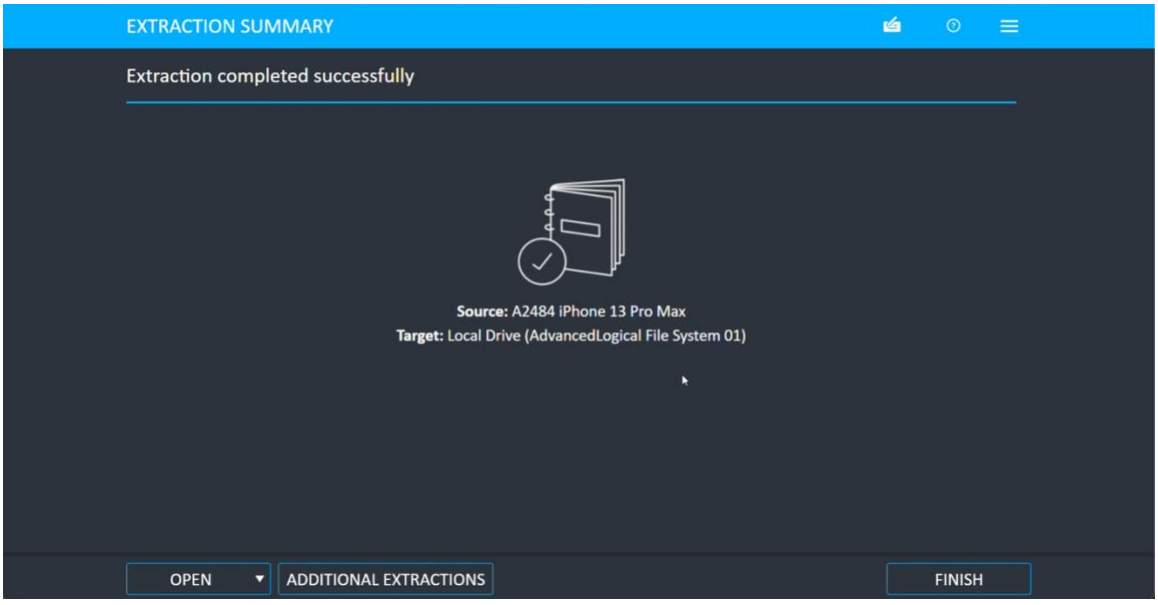


Select desired extraction type (Advanced Logical), then click File System.



Select the extraction location.

Follow the prompts and click continue. Wait for extraction to complete— the device password may need to be entered a few times throughout the extraction process.



Review extraction summary and click Finish.

APPENDIX B

HTML report generated by Physical Analyzer after completion of the extraction.

Summary

Cellebrite Physical Analyzer version	7.61.0.12
Report creation time	4/6/2023 5:24:36 PM -04:00
Time zone settings (UTC)	Original UTC value
Examiner name	krista

Source Extraction

Advanced Logical	
Extraction start date/time	4/5/2023 7:44:52 PM(UTC-4)
Extraction end date/time	4/5/2023 8:12:01 PM(UTC-4)
Unit identifier	818942658
UFED version	7.60.0.222
Internal version	7.60.0.222
Selected manufacturer	Apple
Selected device name	A2484 iPhone 13 Pro Max
Machine name	DESKTOP-3JFN640
Connection type	Cable No. 210
Is encrypted	Encrypted by Physical/Logical Analyzer during the extraction process for user credentials information
Backup password	1234
Extraction type	Advanced Logical
Extraction ID	D1A65A8E-D05B-49DD-8FBB-1C75B8A3BAFC
Extraction (UFD) file data integrity	Intact

Device Information

Name	Value
Advanced Logical	
Model	iPhone14,3
OS	16.3.1
Vendor	Apple
Model number	D64AP
AirDrop ID	[redacted]

Last Cloud Backup Date	4/5/2023 4:02:33 AM(UTC+0)
Apple ID	[redacted]
Apple ID	[redacted]
iCloud account present	True
Advertising Id (IDFA) #1	[redacted]
Time Zone	(UTC-05:00) New_York (America)
Last restore from backup	1/1/2022 12:10:22 AM(UTC+0)
Source of last restored backup	iCloud Backup
Last user ICCID	[redacted]
ICCID	[redacted]
MSISDN	[redacted]
IMSI	[redacted]
Bluetooth device address	[redacted]
WiFi address	[redacted]
Serial	[redacted]
IMEI	[redacted]
Unique ID	[redacted]
Detected model	iPhone (D64AP)
Phone date/time	4/5/2023 11:45:03 PM(UTC+0)
Detected Phone Model Identifier	iPhone14,3
OS Version	16.3.1
Phone date/time	4/5/2023 11:45:09 PM(UTC+0)
Krista's iPhone	
OS Version	16.3.1
ICCID	[redacted]
IMEI	[redacted]
Unique ID	[redacted]
Detected Phone Model Identifier	iPhone14,3
Owner Name	Krista's iPhone
Serial	[redacted]
Is encrypted	True
MSISDN	[redacted]
Phone Settings	
Message Retention Duration	Forever
Location Services Enabled	True
Locale language	en_US
Cloud Backup Enabled	True
Find my iPhone enabled	True

Tethering	
Last Hotspot Activity	4/5/2023 5:19:38 AM(UTC+0)
Physical SIM	
Last used MSISDN	[redacted]
Last user ICCID	[redacted]

Image Hash Details (1)

⚠ Hash data is available for this project.

#	Name	Info	
1	**FileDump**	**Path**	Apple_A2484 iPhone 13 Pro Max.zip
		Size (bytes)	31316241681
		SHA256	C4266442A018F6E876645917ED0BFF470C95B5A4BB3CB0C82B81E18A0CBC4510

Plugins

#	Name	Author	Version
1	**PreProject**	Cellebrite	2.0
2	**iPhone Backup Parser** Parses all iPhone Backup/Logical/FS dumps, including decryption and/or FileSystem creation when necessary	Cellebrite	2.0
3	**ContactsCrossReference** Cross references the phone numbers in a device's contacts with the numbers in SMS messages and Calls. Will fill in the Name field of calls and SMS if there's a match.	Cellebrite	2.0
4	**ProjectProcessorFinisher**	Cellebrite	2.0
5	**PostProject**	Cellebrite	2.0

Contents

Type	Included in report	Total
Activity Sensor Data	27342	27342
Applications Usage Log	10	10
Calendar	1238	1238
Call Log	406	406
Chats	765	765
Instagram	56	56
• [redacted]	15	15
• [redacted]	20	20
• [redacted]	21	21
Life360	2	2
• [redacted]	2	2
Native Messages	323	323
• [redacted]	153	153
• [redacted]	157	157
Recents	384	384
Contacts	2567	2567

Cookies	4803	4803
Credit Cards	2	2
Device Connectivity	1143	1143
Device Events	1	1
Emails	20	20
Installed Applications	263	263
Instant Messages	777	777
Journeys	13	13
Locations	3351	3351
Log Entries	26046	26046
Notes	71	71
Passwords	1336	1336
Searched Items	690	690
Transfers	245	245
User Accounts	33	33
Voicemails	197	197
Web Bookmarks	47	47
Web History	8338	8338
Wireless Networks	646	646
Timeline	258520	258520
Data Files	84282	84282
Archives	73	73
Audio	301	301
Configurations	44676	44676
Databases	708	708
Documents	431	431
Images	20707	20707
Text	13660	13660
Videos	3726	3726

APPENDIX C

Import statements and global variable declarations.

```
import nltk
import pandas as pd
from nltk.tokenize import RegexpTokenizer
import os
import html2text
import re
import streamlit as st
import phonenumbers
from phonenumbers import geocoder
import gender_guesser.detector as gender
import datetime
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter
from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import us
from haversine import haversine
import webbrowser

ROOT_DIR = ''
USER_NAME = ''
USER_EMAIL = ''
USER_NUMBER = ''
NUMBER_LOCATION = ''
NUM_OF_CONTACTS = 0
USER_BIRTHDAY = ''
USER_GENDER = ''
CITY_OF_RESIDENCE = ''
DETAILED_CONTACTS = {'Contact Name': [], 'Contact Number': [], 'Location': [], 'Number of
Interactions': []}
DETAILED_WORDS = {'Word': [], 'Count': []}
LOCS = []
ALL_LOCS = []  # all locations for top 20 contacts
COLORS = ["#003366", "#006666", "#FFD600", "#F00000"]  # teal, blue, yellow, red
ALL_CORDS = {}
COORD_COUNT = 1
TOT_DIST = 0
MANUFACTURER = ''
DEVICE_NAME = ''
OS_VERSION = ''
TIME_ZONE = ''

geolocator = Nominatim(user_agent="my-app", timeout=10)
reverse_geocode = RateLimiter(geolocator.reverse, min_delay_seconds=0.001)
```

APPENDIX D

Python functions to parse the CVS and HTML files.

```
def parse_csv_file(file: str):
    """
    opens and parses csv to find dataset names and index file paths
    :param file: CSV file with name, file path values
    :return: dictionary with dataset names as keys and file paths as values
    """
    text = file.read().decode('utf-8').split('\r\n')[1:-1]
    datasets = {}
    for entry in text:
        name, path = entry.split(',')
        datasets[name] = path
    return datasets


def parse_file(file: str):
    """
    opens and parses html file
    :param file: file path to HTML report
    :return: HTML file as text
    """

    html = open(file, encoding='utf-8').read()
    text = html2text.html2text(html)
    return text
```

APPENDIX E

Python functions for getting basic user information.

```
def get_basic_info(text: str):
    """
    extract name, phone number, number of contacts from HTML file
    :param text: text of HTML file
    :return: N/A
    """

    global USER_NAME
    global USER_NUMBER
    global USER_EMAIL
    global NUMBER_LOCATION
    global NUM_OF_CONTACTS
    global USER_GENDER

    apple_id_list = re.findall(".*Apple ID.*", text)
    USER_EMAIL = str(apple_id_list[0].split('|')[1])

    get_name_from_accounts()
    if USER_NAME == '':
        owner_name_list = re.findall(".*Owner Name.*", text)
        USER_NAME = re.findall('[a-zA-Z]*', str(owner_name_list[0].split('|')[1]))[1]
        if USER_NAME == 'iPhone' or USER_NAME == '':
            get_name_from_contacts()

    contacts_list = re.findall(".*Contacts.*", text)
    NUM_OF_CONTACTS = int(re.findall('^| \d*|$ ', contacts_list[1])[3])

    number_list = re.findall(".*MSISDN.*", text)
    USER_NUMBER = str(number_list[0].split('|')[1])

    detector = gender.Detector()
    first_name = USER_NAME.split(" ")[0]
    USER_GENDER = detector.get_gender(first_name)
    if USER_GENDER == 'male' or USER_GENDER == 'mostly_male':
        USER_GENDER = 'Male'
    if USER_GENDER == 'female' or USER_GENDER == 'mostly_female':
        USER_GENDER = 'Female'

    contacts = parse_file(ROOT_DIR + '\\pages\\contacts.html')
    get_birthday_from_contacts(contacts)
    if USER_BIRTHDAY == '':
        chat_data = analyze_chats()
        get_birthday_from_chats(chat_data)

def get_name_from_accounts():
    """
    get user's full name from associated account names
```

```
    :return: N/A
    """

    global USER_NAME

    path = ROOT_DIR + "\\pages\\user_accounts.html"
    accts = parse_file(path)
    accounts = accts.split("\n\n")
    names = {}
    for acc in accounts:
        if acc.__contains__("**Account Name**"):
            name = acc.split("**Account Name**")[1].split('\n')[1][0:-2]
            if re.match('[a-zA-Z]* [a-zA-Z]*', name):
                full_name = re.findall('[a-zA-Z]* [a-zA-Z]*', name)[0]
                if full_name in names.keys():
                    names[full_name] += 1
                else:
                    names[full_name] = 1
    if names:
        USER_NAME = str(max(names, key=names.get))
    else:
        USER_NAME = ''

def get_name_from_contacts():
    """
    get the user's name from the user's number's contact entry
    :return: N/A
    """

    global USER_NAME

    number = phonenumbers.parse(USER_NUMBER, "US")
    formatted_number = phonenumbers.format_number(number,
phonenumbers.PhoneNumberFormat.NATIONAL)
    ext = '\pages\contacts.html'
    new_filepath = ROOT_DIR + ext
    contacts = parse_file(new_filepath)
    if contacts.__contains__(formatted_number):
        owner_name = contacts.index(formatted_number)
        start = owner_name-290
        text_area = contacts[start:owner_name]
        name = re.findall('\*\*Name:\*\*.*', text_area)
        name_list = name[0].rsplit('*')
        USER_NAME = name_list[len(name_list)-1][1:]

def get_birthday_from_contacts(contacts):
    global USER_BIRTHDAY
    if contacts.__contains__(USER_NAME):
        user_entry = contacts.split(USER_NAME)[1]
        if user_entry.__contains__("Birthday|"):
            birthday = re.findall("^Birthday| [0-9]{1,2}/[0-9]{1,2}/[0-9]{4}|$", user_entry)[0][1:]
```

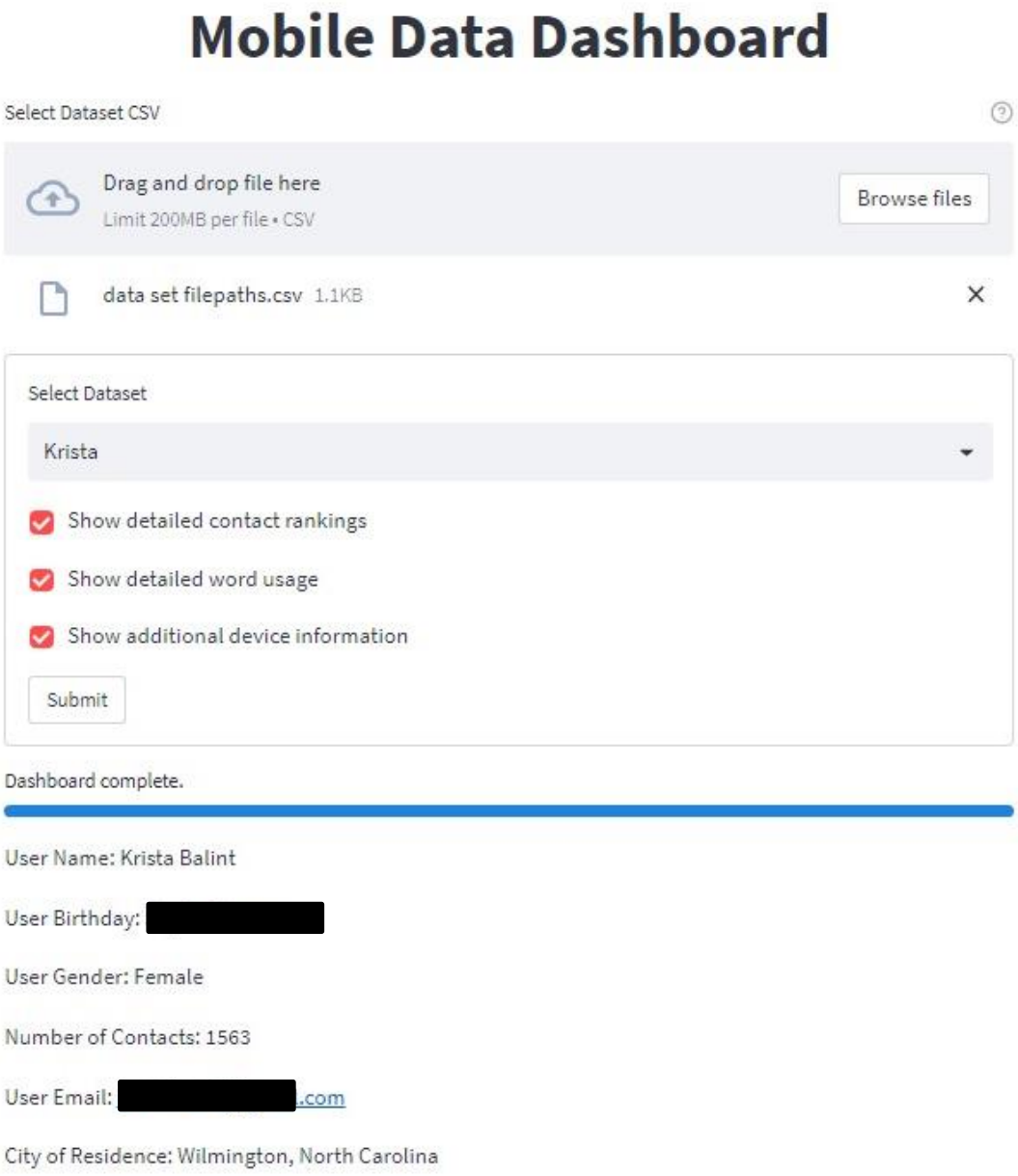
```
            month, day, year = birthday.split('/')
            USER_BIRTHDAY = str(datetime.date(day=int(day), month=int(month),
year=int(year)).strftime('%B %d %Y'))
    else:
        USER_BIRTHDAY = ''

def analyze_chats():
    path = ROOT_DIR + '\\chats'
    paths = os.listdir(path)
    chats = {}
    for file in paths:
        new_path = path + "\\" + file
        files = os.listdir(new_path)
        for txt in files:
            final_path = new_path + "\\" + txt
            if os.path.exists(final_path) and final_path.endswith('.txt'):
                chats[file + '\\' + txt] = open(final_path, encoding='utf-8').read()
    return chats

def get_birthday_from_chats(chats):
    """
    finds the user's birthday by finding the most common day other people send the user a message
            with the word ‘birthday' in it
    :param chats: a dictionary containing dates as keys and messages as values
    :return: N/A
    """

    global USER_BIRTHDAY

    dates = {}
    for key in chats:
        message = chats[key]
        messages = message.split('------------------------------')
        for chat in messages:
            if not chat.__contains__('(owner)'):
                if chat.lower().__contains__('birthday'):
                    date = re.findall('Timestamp: [0-9]{1,2}/[0-9]{1,2}/[0-9]{4}', chat)[0].split(': ')[1]
                    month, day, year = date.split('/')
                    day = month + '/' + day
                    if day in dates.keys():
                        dates[day] += 1
                    else:
                        dates[day] = 1
    if dates:
        birthday = max(dates, key=dates.get)
        month, day = birthday.split('/')
        USER_BIRTHDAY = str(datetime.date(day=int(day), month=int(month),
year=2000).strftime('%B %d'))
    else:
        USER_BIRTHDAY = 'Unknown'
```

```
def analyze_location():
    """
    parse the location HTML report and extract coordinate, date pairs
    :return: dictionary with dates as keys and coordinates as values
    """

    global ALL_CORDS
    global COORD_COUNT

    location = {}
    ext = '\\pages\\locations.html'
    new_filepath = ROOT_DIR + ext
    if os.path.exists(new_filepath):
        locations = parse_file(new_filepath)
    coordinates = locations.split('|\n\n| | ')[1:]
    for loc in coordinates:
        # print(loc)
        try:
            coords = re.findall('\(.*\)\|', loc)[0][1:-2]
        except:
            continue
        if coords.count(",") > 1:
            temp = coords.rsplit(',')
            coords = temp[0] + ',' + temp[1]
        try:
            time = re.findall('\*\*Time\*\*\|.*', loc)[0]
        except:
            continue
        date = re.findall('[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}', time)[0].split('/')
        year = int(date[2])
        month = int(date[0])
        day = int(date[1])
        formatted_date = datetime.datetime(year, month, day)
        location[formatted_date.date()] = coords
        ALL_CORDS[COORD_COUNT] = coords
        COORD_COUNT += 1
    return location

def get_city_of_residence(locations: dict):
    """
    determine the user's city of residence by finding most frequent city visited within the past year
    :param locations: dictionary of visited coordinates and dates
    :return: N/A
    """

    global CITY_OF_RESIDENCE

    cities_visited = {}
    for key in locations:
        year = int(str(key)[0:4])
        if year >= 2022:
```

```
        # location = geolocator.reverse(locations[key])
        location = reverse_geocode(locations[key])
        if 'city' in location.raw['address'].keys():
            city_or_town = location.raw['address']['city']
        elif 'town' in location.raw['address'].keys():
            city_or_town = location.raw['address']['town']
        else:
            continue
        # try:
        #     county = location.raw['address']['county']
        # except:
        #     continue
        if 'state' in location.raw['address'].keys():
            state = location.raw['address']['state']
        else:
            continue
        area = city_or_town + ', ' + state
        if area in cities_visited.keys():
            cities_visited[area] += 1
        else:
            cities_visited[area] = 1
# print(cities_visited)
CITY_OF_RESIDENCE = str(max(cities_visited, key=cities_visited.get))
```

APPENDIX F

Mobile data dashboard user information section.



APPENDIX G

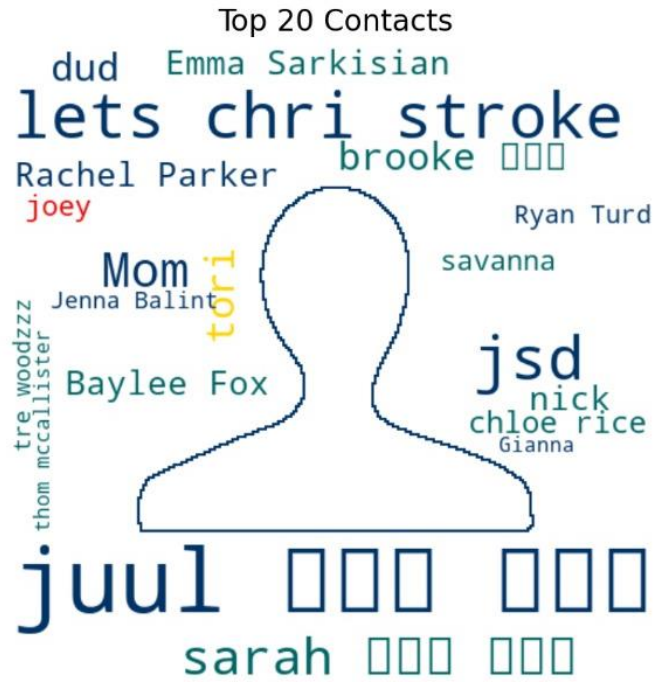
Python code for development of the word cloud.

```
def contacts_analysis(chats):
    """
    identifies the 20 contacts most often interacted with and their locations
    :param chats: analyzed and parsed chat data
    :return: word cloud of top 20 contacts
    """

    global LOCS
    global ALL_LOCS
    global DETAILED_CONTACTS

    def get_name_color(word, font_size, position, orientation, font_path, random_state):
        state = number_locations[word]
        if state in LOCS:
            return COLORS[LOCS.index(state)]
        else:
            return COLORS[len(LOCS)]

    contacts = {}
    contact_numbers = {} #contact name keys w corresponding phone number values
    for key in chats:
        message = chats[key]
        messages = message.split('------------------------------')
        for chat in messages:
            # print(chat)
            if not chat.__contains__('(owner)'):
                contact = re.findall('From: .*\n', chat)
                if contact:
                    contact_name = contact[0].split('From: ')[1]
                    entry = re.findall('^\+[0-9]{11} .*', contact_name)
                    if entry:
                        entry_split = entry[0].split(' ', 1)
                        if len(entry_split) > 1:
                            number, name = entry_split
                            if name in contacts.keys():
                                contacts[name] += 1
                            else:
                                contacts[name] = 1
                            if name not in contact_numbers.keys():
                                contact_numbers[name] = number
    #find state values
    number_locations = {} #state value corresponding to contact name key
    most_common_locations = {} #state value and count for top 20 contacts
    contacts_counter = Counter(contacts)
    top_20_contacts = dict(contacts_counter.most_common(20))
    for key in contacts_counter.most_common(20):
        num = contact_numbers[key[0]]
```

```
            number = phonenumbers.parse(num, "US")
            NUMBER_LOCATION = geocoder.description_for_number(number, "en")
            if re.match('[a-zA-Z ]*, [A-Z]{2}', NUMBER_LOCATION):
                _, state_abbr = NUMBER_LOCATION.split(", ")
                NUMBER_LOCATION = str(us.states.lookup(state_abbr))
            number_locations[key[0]] = NUMBER_LOCATION
            if NUMBER_LOCATION in most_common_locations.keys():
                most_common_locations[NUMBER_LOCATION] += 1
            else:
                most_common_locations[NUMBER_LOCATION] = 1
        most_common_locations = Counter(most_common_locations)
        for loc in most_common_locations:
            ALL_LOCS.append(loc[0])
        for loc in most_common_locations.most_common(3):
            LOCS.append(loc[0])

        for key in top_20_contacts.keys():
            DETAILED_CONTACTS['Contact Name'].append(key)
            DETAILED_CONTACTS['Contact Number'].append(contact_numbers[key])
            DETAILED_CONTACTS['Location'].append(number_locations[key])
            DETAILED_CONTACTS['Number of Interactions'].append(top_20_contacts[key])

        shape = np.array(Image.open(r'D:\Contact.png'))
        wordcloud = WordCloud(background_color='white', mask=shape, height=shape.shape[0],
width=shape.shape[1], color_func=get_name_color, contour_color='#003366', contour_width=1,
scale=2, max_font_size=40, font_step=1).generate_from_frequencies(top_20_contacts)

        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title('Top 20 Contacts')

        return plt
```

APPENDIX H

Mobile data dashboard contact word cloud and detailed contact information.



	Contact Name	Contact Number	Location	Number of Interactions
0	juul	[redacted]	New York	20758
1	lets chri stroke	[redacted]	New York	9851
2	jsd	[redacted]	New York	9245
3	sarah	[redacted]	North Carolina	6493
4	Mom	[redacted]	New York	5979
5	tori	[redacted]	Virginia	4390
6	brooke	[redacted]	North Carolina	4080
7	dud	[redacted]	New York	4076
8	Emma Sarkisian	[redacted]	North Carolina	2987
9	Rachel Parker	[redacted]	New York	2927
10	nick	[redacted]	North Carolina	2743
11	Baylee Fox	[redacted]	North Carolina	2734
12	chloe rice	[redacted]	North Carolina	2291
13	savanna	[redacted]	North Carolina	1987
14	joey	[redacted]	New Jersey	1819
15	Ryan Turd	[redacted]	New York	1557
16	tre woodzzz	[redacted]	North Carolina	1395
17	Jenna Balint	[redacted]	New York	1275
18	thom mccallister	[redacted]	North Carolina	863
19	Gianna	[redacted]	New York	836

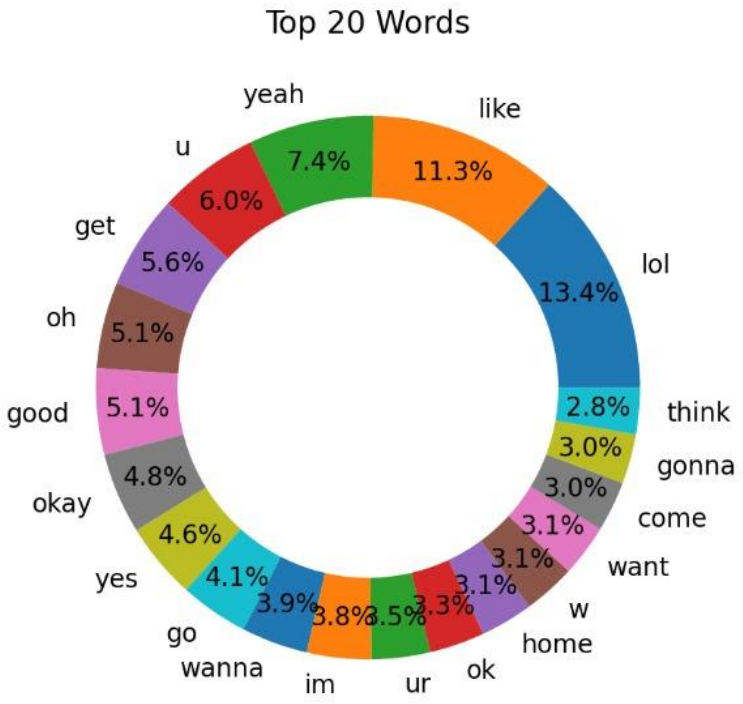
APPENDIX I

Python code for development of the word donut chart.

```
def word_analysis(chats):
    """
    identifies 20 most common words sent in chats
    :param chats: analyzed and parsed chat data
    :return: donut chart of top 20 words
    """
    global DETAILED_WORDS
    stopwords = nltk.corpus.stopwords.words('english')
    iphone_stopwords = ['8', 'ball', 'cup', 'pong', 'emphasized', 'loved', 'questioned', 'liked', 'disliked',
'laughed']
    stopwords.extend(iphone_stopwords)
    all_messages = {}
    message_bodies = ""
    for key in chats:
        message = chats[key]
        messages = message.split('------------------------------')
        for chat in messages:
            if chat.__contains__('(owner)'):
                _, message_body = chat.split('Body:\n')
                if message_body != '\n':
                    all_messages[key] = message_body.strip()
                    message_bodies += " " + message_body.strip().lower()
                    # print(message_body.strip().lower())

    # removes non-alphanumeric characters
    tokenizer = RegexpTokenizer(r'\w+')
    words = tokenizer.tokenize(message_bodies)
    tokens_without_sw = [word for word in words if not word in stopwords]
    top_words = nltk.FreqDist(tokens_without_sw)
    top_20_words = []
    top_words_freq = []
    for word in top_words.most_common(20):
        top_20_words.append(word[0])
        DETAILED_WORDS['Word'].append(word[0])
        top_words_freq.append(word[1])
        DETAILED_WORDS['Count'].append(word[1])

    plt.pie(top_words_freq, labels=top_20_words,
            autopct='%1.1f%%', pctdistance=0.85)

    centre_circle = plt.Circle((0, 0), 0.70, fc='white')
    fig = plt.gcf()

    fig.gca().add_artist(centre_circle)

    plt.title('Top 20 Words')
    return plt
```

APPENDIX J

Mobile data dashboard word donut chart and detailed word information.



	Word	Count
0	lol	3920
1	like	3292
2	yeah	2177
3	u	1765
4	get	1631
5	oh	1503
6	good	1499
7	okay	1410
8	yes	1332
9	go	1193
10	wanna	1154
11	im	1119
12	ur	1015
13	ok	961
14	home	919
15	w	908
16	want	900
17	come	877
18	gonna	863
19	think	808

APPENDIX K

Python functions for map generation and additional device information.

```
def location_analysis(locations):
    """
    generate data frame from location data
    :param locations: dictionary of coordinates visited by user
    :return: data frame with lat and lon
    """
    coordinates = {'lat': [], 'lon': []}
    for key in locations.keys():
        coords = locations[key]
        lat, lon = coords.split(", ")
        coordinates['lat'].append(float(lat))
        coordinates['lon'].append(float(lon))
    return pd.DataFrame(coordinates)

def get_total_distance():

    global TOT_DIST
    global ALL_CORDS
    global COORD_COUNT
    # print(ALL_CORDS)
    for i in range(1, COORD_COUNT-1):
        loc1_lat, loc1_lon = ALL_CORDS[i].split(", ")
        loc1 = (float(loc1_lat), float(loc1_lon))
        loc2_lat, loc2_lon = ALL_CORDS[i+1].split(", ")
        loc2 = (float(loc2_lat), float(loc2_lon))
        TOT_DIST += haversine(loc1, loc2, unit='mi')

def get_device_info(text):
    global NUMBER_LOCATION
    global MANUFACTURER
    global DEVICE_NAME
    global OS_VERSION
    global TIME_ZONE

    number = phonenumbers.parse(USER_NUMBER, "US")
    NUMBER_LOCATION = geocoder.description_for_number(number, "en")

    manufacturer = re.findall(".*Selected manufacturer\|\n\n.*", text)
    MANUFACTURER = manufacturer[0].split("\n\n")[1].strip()

    device_name = re.findall(".*Selected device name\|\n\n.*", text)
    DEVICE_NAME = device_name[0].split("\n\n")[1].strip()

    os_version = re.findall(".*OS Version\|.*", text)
    OS_VERSION = os_version[0].split("| ")[1].strip()

    time_zone = re.findall(".*Time Zone\|.*", text)
```

```
TIME_ZONE = time_zone[0].split("|")[1].strip()

associated_accounts = {'Username': [], 'Service/Source': [], 'Account Name': []}

path = ROOT_DIR + "\\pages\\user_accounts.html"
accts = parse_file(path)
acct_source = False
accounts = accts.split("|")
for acc in accounts:
    if acc.__contains__("Username"):
        _, info = acc.split("**Username**  \n")
        username, info = info.split('\n', 1)
        associated_accounts['Username'].append(username.strip())
        if info.__contains__("**Service Type**"):
            _, info = info.split("**Service Type**  \n")
            type, info = info.split('\n', 1)
            associated_accounts['Service/Source'].append(type.strip())
            acct_source = True
        if info.__contains__("**Account Name**"):
            _, info = info.split("**Account Name**  \n")
            name, info = info.split('\n', 1)
            associated_accounts['Account Name'].append(name.strip())
        else:
            associated_accounts['Account Name'].append('N/A')
        if info.__contains__("**Source**"):
            _, info = info.split("**Source**  \n")
            source, info = info.split('\n', 1)
            if source.strip() != 'Accounts' and source.strip() != 'Chrome':
                associated_accounts['Service/Source'].append(source.strip())
                acct_source = True
        if not acct_source:
            associated_accounts['Service/Source'].append('N/A')
return associated_accounts
```

APPENDIX L

Mobile data dashboard map of visited locations and additional device information.

Map of Visited Locations

Approximate total distance travelled: 175315.64 mi.

Additional Device Information

User Number: [redacted]

Phone Number Origin State: New York

Maufacturer: Apple

Device Name: A2484 iPhone 13 Pro Max

OS Version: 16.3.1

Time Zone: (UTC-05:00) New_York (America)

APPENDIX M

Mobile data dashboard associated accounts and button to view reports.

Associated Accounts

	Username	Service/Source	Account Name
0	[redacted]	Device Locator	N/A
1	[redacted]	CloudKit	N/A
2	[redacted]	Find My Friends	N/A
3	[redacted]	IDMS	N/A
4	[redacted]	Apple ID	N/A
5	[redacted]	Apple ID	N/A
6	[redacted]	iTunes Store	N/A
7	[redacted]	Gmail	N/A
8	local	iTunes Store	N/A
9	[redacted]	iTunes Store	N/A
10	[redacted]	Game Center	N/A
11	local	iTunes Store (Sandbox)	N/A
12	[redacted]	Messages	N/A
13	[redacted]	iCloud	N/A
14	[redacted]	Snapchat	N/A
15	[redacted]	Instagram	[redacted]
16	[redacted]	Life360	[redacted]
17	[redacted]	Instagram	[redacted]
18	[redacted]	Gmail	[redacted]
19	[redacted]	LinkedIn	[redacted]
20	[redacted]	Venmo	[redacted]
21	[redacted]	Pinterest	[redacted]
22	[redacted]	Instagram	[redacted]

View HTML Reports

APPENDIX N

Python code using Streamlit to generate the dashboard.

```
# run in terminal "streamlit run main.py"
st.markdown("<div style='text-align: center'> <h1> Mobile Data Dashboard </h1> </div>",
unsafe_allow_html=True)

file = st.file_uploader('Select Dataset CSV', type='csv', help='Enter a CSV with each dataset
name and HTML index filepath on its own line', key='file_upload')
if file:
   datasets = parse_csv_file(file)
   with st.form("data_selection"):
      selection = st.selectbox("Select Dataset", datasets, key='dataset_selection')
      detailed_contacts = st.checkbox('Show detailed contact rankings', value=False)
      detailed_words = st.checkbox('Show detailed word usage', value=False)
      additional_data = st.checkbox('Show additional device information', value=False)
      submitted = st.form_submit_button("Submit")
   if submitted:
      start = time.time()
      progress_bar = st.progress(0, "Getting basic info...")
      filepath = datasets[selection]
      start = 0
      end = filepath.rfind('\\')
      ROOT_DIR = filepath[start:end]
      text = parse_file(filepath)
      get_basic_info(text)
      number = phonenumbers.parse(USER_NUMBER, "US")
      formatted_number = phonenumbers.format_number(number,
phonenumbers.PhoneNumberFormat.NATIONAL)
      st.write("User Name: " + USER_NAME)
      st.write("User Birthday: " + USER_BIRTHDAY)
      st.write("User Gender: " + USER_GENDER)
      st.write("Number of Contacts: " + str(NUM_OF_CONTACTS))
      st.write("User Email: " + USER_EMAIL)
      progress_bar.progress(15, "Calculating city of residence...")
      location_data = analyze_location()
      get_city_of_residence(location_data)
      st.write("City of Residence: " + CITY_OF_RESIDENCE)
      st.write()
      progress_bar.progress(30, "Analyzing contacts...")
      chat_data = analyze_chats()

      #Contacts Analysis: Top 20 Contacts
      contact_fig = contacts_analysis(chat_data)
      st.set_option('deprecation.showPyplotGlobalUse', False)
      progress_bar.progress(45, "Plotting contacts...")
      st.pyplot(contact_fig)
      col1, col2, col3, col4 = st.columns(4)
      for i in range(len(LOCS)):
         if i == 0:
```

```
            col1.color_picker(label=LOCS[i], value=COLORS[i])
        if i == 1:
            col2.color_picker(label=LOCS[i], value=COLORS[i])
        if i == 2:
            col3.color_picker(label=LOCS[i], value=COLORS[i])
    if len(ALL_LOCS) > 3:
        col4.color_picker(label='Other', value=COLORS[3])
    if detailed_contacts:
        st.table(pd.DataFrame(DETAILED_CONTACTS))

    progress_bar.progress(60, "Analyzing word usage...")
    #Word Analysis: Top 20 Words
    word_fig = word_analysis(chat_data)
    st.set_option('deprecation.showPyplotGlobalUse', False)

    progress_bar.progress(75, "Plotting word usage...")
    st.pyplot(word_fig)
    if detailed_words:
        st.table(pd.DataFrame(DETAILED_WORDS))

    progress_bar.progress(90, "Plotting location data...")
    #Location Analysis: Stored locations and total distance travelled
    st.markdown("<div style='text-align: center'> <h3> Map of Visited Locations </h3>
</div>", unsafe_allow_html=True)
    loc_df = location_analysis(location_data)
    st.map(loc_df, zoom=1)
    get_total_distance()
    st.write("Approximate total distance travelled: " + '{:.2f}'.format(TOT_DIST) + ' mi.')

    if additional_data:
        st.markdown("<div style='text-align: center'> <h3> Additional Device Information </h3>
</div>",
                    unsafe_allow_html=True)
        df = get_device_info(text)
        st.write("User Number: " + formatted_number)
        st.write("Phone Number Origin State: " + NUMBER_LOCATION)
        st.write("Maufacturer: " + MANUFACTURER)
        st.write("Device Name: " + DEVICE_NAME)
        st.write("OS Version: " + OS_VERSION)
        st.write("Time Zone: " + TIME_ZONE)
        st.markdown("<div style='text-align: center'> <h4> Associated Accounts </h4> </div>",
                    unsafe_allow_html=True)
        st.table(pd.DataFrame(df))
    progress_bar.progress(100, "Dashboard complete.")
    st.write("\n\n")
    url = str(datasets[selection])
    button = st.button('View HTML Reports', on_click=None)
    end = time.time()
    print(end-start)
    if button:
        webbrowser.open_new_tab(url)
```

APPENDIX O

Complete list of libraries used.

Tool	How it is used
Streamlit	Construction of the dashboard
html2text	Parsing of the HTML reports
os	Access to filenames on the OS
re	Generation of regular expressions for pattern searching
NumPy	Assisting with plot creation
datetime	Formatting of date and time values
pandas	Data frame generation & analysis
WordCloud	Development of the word cloud
Pyplot	Graphing of the word cloud & donut chart
Pillow	Shaping of the word cloud
gender-guesser	Gender estimation
phonenumbers	Identification of phone number information
Natural Language Toolkit (NLTK)	Parsing and analysis of chat data in HTML reports
GeoPy	Finding addresses from coordinates
US	Finding state from zip code
haversine	Calculating the approximate total distance travelled
webbrowser	Opening of HTML reports in the browser
collections	Creating counter objects

APPENDIX P

Survey given to those who attended the dashboard demonstration.

Mobile Data Dashboard Survey - 3/23/2023

Was the mobile data dashboard easy to use? Circle one.

Very Easy - Easy - Just Ok - Hard - Very Hard
5 4 3 2 1

Would you be interested in the mobile dashboard if it was available to you? Choose one.

Yes
No
Not Sure

Is there any other information you want to see on the dashboard? Please describe below.

Is there anything specific date item or feature that you liked or disliked? Please describe below.

Any additional comments: