Deep Learning for Mental Health Screening

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degrees of Bachelor of Science in Computer Science and Data Science.

Authored by:
Lillian Garfinkel (CS)
Madeline Halley (CS)
Nick Jurovich (CS/DS)
Mairéad O'Neill (CS)
Brian Phillips (DS)
Jyalu Wu (CS/DS)

Advised by:
Professor Elke Rundensteiner
PhD Candidate ML Tlachac

Abstract

Almost 300 million people across the world suffer from depression. This is a widespread issue in which a majority of those suffering do not receive treatment. Screening for depression using smartphone data could result in less biased, more proactive results and could be accomplished by recognizing text patterns in people's messages. To explore this potential, we analyzed previously collected text message data through the application of lexical category featurization and a deep learning model, BERT. While we experimented with existing data, we also researched methods for further data collection. We concluded that the Bag of Words method of categorizing words outperformed the Empath method. We also determined that the methods of lexical category featurization produced higher accuracies than the BERT models, and more advanced BERT models were prone to overfitting. By visualizing the results, we discovered that those with depression discussed less about personal topics than asymptomatic people.

Acknowledgements

This project would not have been possible without the support of many people. We would like to first thank our adviser, Professor Elke Rundensteiner. Over the course of this year, she has helped us shape this project into something we are proud of, encouraged us to make our own decisions and to explore new ideas, and shared with us her infinite wisdom.

We would like to thank our mentor, ML Tlachac, for always being willing to sit down with us to teach, troubleshoot, or just to check in. ML's oversight, management, and enthusiasm for making a difference with this project were invaluable. This team would not have been able to accomplish nearly as much without them.

Next, we would like to thank Dr. Ermal Toto and Dr. James Kingsley from the WPI Academic Technology Center. Dr. Ermal Toto helped us up the WPI development server for the Beiwe backend, and took the time to help us figure out what the best way forward would be. James Kingsley helped us create accounts for Turing, WPI's high performance research cluster for computational science, and for troubleshooting any problems that we encountered.

Endless thanks to Andrea Mignone for letting us use the code for his flower plots, and for being so enthusiastic about us doing so.

Finally, we would like to thank the previous MQP and research teams who have contributed to this project.

Contents

List of Figures

List of Tables

1 Introduction

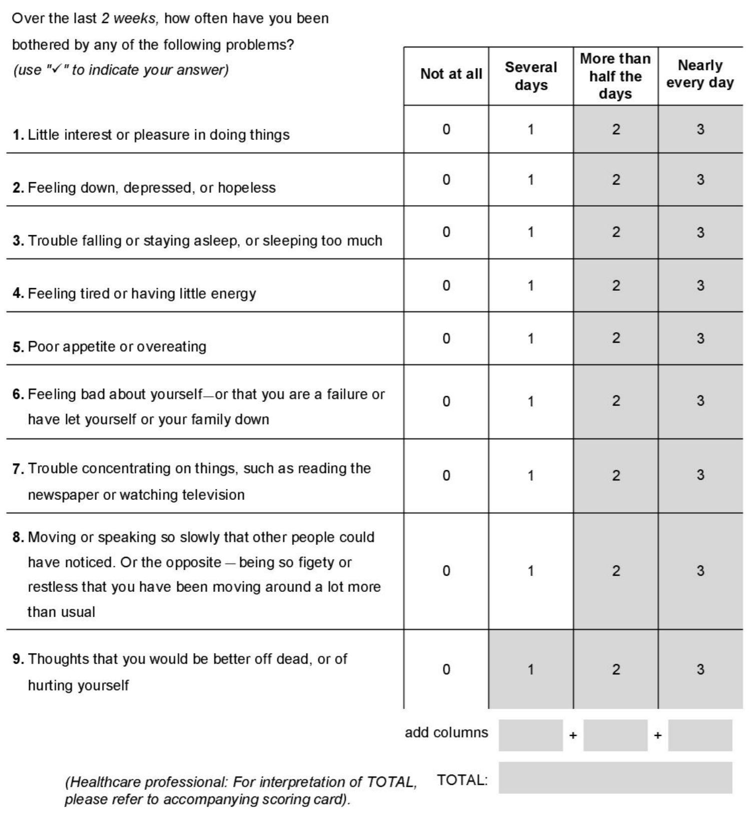
Depression is a "common and serious medical illness that negatively affects how you feel, the way you think and how you act" (Torres, 2020). This pervasive mental disorder affects over 264 million people worldwide [3]. Untreated depression can lead to an array of other problems, such as a weakened immune system, heart disease, physical pain, and, in the worst case scenario, suicide [4].

Thankfully, treatment for depression is highly effective, as up to 80% of those treated show signs of improvement within around four to six weeks [5]. However, "nearly two out of three people suffering with depression do not actively seek nor receive proper treatment" [5], and it takes an average of 11 years for someone with a mental illness to get treatment [6]. Much of this is due to the stigma surrounding depression and mental health – an individual suffering from depression may feel unworthy of help or not normal for suffering from depression. Since depression can worsen when left untreated, this is extremely detrimental and dangerous. In the modern and digital age, technology has increased the capacity for human connection. You can reach virtually anyone through a few clicks, and it seems that people are constantly interacting with one another through text messaging or social media. However, despite this constant connection, people are more lonely than ever. There is an alarming linear correlation between depression and smartphone use, with more time spent on smartphones being associated with a higher risk of depression [7]. Therefore, detecting depression with the culprit itself - mobile devices - could unobtrusively monitor for signs of depression.

1.1 Current Screening Methods

Screening methods - such as the EQ5D, the Child Behavior Checklist (CBCL), and the Patient Health Questionnaire (PHQ-9) - can be applied to detecting depression. These methods are not intended to diagnose depression, but to indicate that the patient should be evaluated further. For example, the EQ5D is a questionnaire designed to evaluate general quality of life in five different areas: mobility, self-care, usual activities, pain/discomfort and anxiety/depression [8]. Since depression can often lead to a decrease in overall quality of life, this screening method is useful as it can provide context to other areas of a person's life if it indicates they are depressed. The EQ5D is widely available and takes just under five minutes to complete. Furthermore, the CBCL is a questionnaire/checklist designed to measure behavioral and social problems for youth aged 6-18 [8]. The checklist can be administered to the child's parents, the child's teachers, or the child.

Lastly, the PHQ-9 contains a range of questions that aim to gauge the level of depressive symptoms that the patient has been experiencing [8]. The PHQ-9 is the technique that this project focuses on, as it is the most common screening method in the United States [9], and it has been proven to be effective when integrated with mobile devices [10]. PHQ-9 scores are divided into five categories: asymptomatic (scores 0-4), mild (5-9), moderate (10-14), moderate-severe (15-19), and severe (20+). This survey is pictured below.



Over the last *2 weeks,* how often have you been bothered by any of the following problems? *(use "✓" to indicate your answer)*	Not at all	Several days	More than half the days	Nearly every day
1. Little interest or pleasure in doing things	0	1	2	3
2. Feeling down, depressed, or hopeless	0	1	2	3
3. Trouble falling or staying asleep, or sleeping too much	0	1	2	3
4. Feeling tired or having little energy	0	1	2	3
5. Poor appetite or overeating	0	1	2	3
6. Feeling bad about yourself—or that you are a failure or have let yourself or your family down	0	1	2	3
7. Trouble concentrating on things, such as reading the newspaper or watching television	0	1	2	3
8. Moving or speaking so slowly that other people could have noticed. Or the opposite — being so figety or restless that you have been moving around a lot more than usual	0	1	2	3
9. Thoughts that you would be better off dead, or of hurting yourself	0	1	2	3
	add columns		+	+
(Healthcare professional: For interpretation of TOTAL, please refer to accompanying scoring card).	TOTAL:			

Figure 1: PHQ-9 Questionnaire
[11]

Unfortunately, surveys can be unreliable as they require honest self-reflection and a great deal of self awareness. Not only that, but merely 4.2% of adults were screened for depression during routine doctor visits in 2012 and 2013 [11]. Overall, current screening methods fail to provide the reliability nor the level of outreach necessary to address this global mental health crisis. A more proactive, unbiased method is a vital step towards helping people get the treatment they need. Hence, we aim to develop a system in which smartphones can unobtrusively indicate whether its user has depression, by advancing machine learning and deep learning models on text message data.

1.2 Problem Statement and Challenges

The undertaking of this project is accompanied by a few challenges, including a small data set and problems associated with applying text-speak to NLP models. Primarily, NLP models are typically complex and require a significant amount of data to avoid overfitting. Since this project works with a small data set of just approximately 300 participants (over 300 for Moodable, 60 for EMU), data must be methodically pre-processed and managed to yield the desired results. Moreover, applying text-speak data to NLP models has its own set of challenges due to the casual and often grammatically-inaccurate manner of text messages. For instance, people often use abbreviation, slang, or misspellings in their text messages. This can cause a model to categorize words or phrases separately when they really have the same meaning. Also, text message inboxes can be flooded with insignificant messages from bots, such as verification codes, appointment reminders, spam messages, etc. With an already limited dataset, these sorts of obstructions are a huge roadblock in developing an NLP model with high accuracy.

1.3 Summary of Prior Related Work and Research

The concept that deep learning models could have the capability to detect depression stems from prior research that correlates certain linguistic patterns with depression [12]. In the domain of implementing deep learning methods to detect depression, research has been conducted on audio-based depression screening [13]. BERT models have been explored for purposes of Twitter text analysis [14] and classifying tweets into different categories in the field of disaster management [15]. Our project bridges this gap between applying BERT towards text analysis and detecting depression with deep learning, by applying BERT towards text message analysis for classifying participants as depressed or not depressed. Furthermore, research has been conducted regarding visualizing patterns that represent what BERT is learning [16], and visualizing BERT's attention mechanism, or the mechanism that forms connections between elements, such as words [16].

Additionally, past MQP teams have spearheaded this project by experimenting with detecting depression and anxiety from audio data, collecting text message data, and experimenting with applying text and call log data to detecting depression. Researchers in the summer of 2021 implemented an Empath tool to create new lexical categories and developed machine learning models to determine if these new features could aid in detecting depression in text messages. Additionally, the researchers developed BERT models for the text message dataset, and began experimenting with preprocessing methods to improve the results of these BERT models.

1.4 Our Approach and Contributions

Our approach to addressing the aforementioned problems is to build off the prior work in machine learning with Empath features, and to develop an optimized BERT model for detecting depression. We aim to complete the preprocessing that the summer researchers began experimenting with, by applying techniques such as a sliding window to create chunked data, enabling the model to accomodate special characters, and generating new lexical categories with Empath. We also seek to strengthen this model by investigating the potential use of the data collection platform Beiwe to gather more data. Our overarching goal is to achieve a highly accurate model that will help advance research on mobile depression screening.

2 Background

2.1 Prior Work

Summer Work

During the summer of 2021, a Research Experience for Undergraduates (REU) team, consisting of Benjamin Litterer, Nicholas Jurovich, Mahum Shah, and Sai Thatigotla, investigated two ways of using machine learning to screen for depression by conducting preliminary experiments. The first approach involved creating new lexical categories through the use of the Empath tool. Unlike the original Empath lexical categories, which were created based on fictional writing on the web, these new categories were created based on a Reddit-based model. Unfortunately, the results collected with these new categories did not indicate a significant improvement in the ability to screen for depression. The other approach taken by the REU team was to run BERT models on the text message datasets. They found that, linguistically, it is difficult to run BERT models with text message data, and the team determined that the results could be improved if preprocessing techniques were implemented. They began experimenting with these techniques, but were not finished before the completion of the REU. Our team aims to continue the development of these preprocessing methods, as well as add post-processing methods, by improving the previous attempts and by implementing fresh, new ideas. Through these augmentations, the MQP team hopes to increase the accuracy of the BERT models. The team will also be working to improve the lexical categories approach by implementing new, more domain-specific categories and by running various machine learning models.

MQP 20-21

The MQP Team during the 2020-2021 year worked on collecting new data. The team improved and increased survey methods and collected a new dataset from college students. They also conducted experiments to see if time series constructed from text and call logs can predict if an individual is depressed.

MQP 19-20

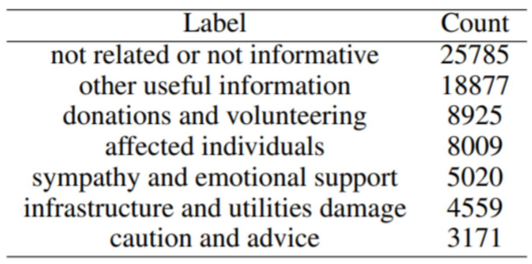
The MQP Team during the 2020-2021 year worked on collecting a new set of data. They developed an Android application and a website to distribute a survey, which was updated to incorporate questions from the PHQ-9 survey. The app also, with permission, offered audio prompts for participants to answer, and recorded call logs, text message data, and calendar events. This resulted in a new dataset from college students, in which they were able to analyze calculated PHQ-9 scores and compare them with past studies, and analyze the demographics of their data set. They also conducted experiments to see if time series constructed from text and call logs can reveal a pattern in PHQ-9 scores. Ultimately, they found some correlation between outgoing text logs and predicting depression, however more research needs to be done on the significance of this trend.

2.2 Prior Related Research

2.2.1 BERT

Tweets Classification with BERT in the Field of Disaster Management

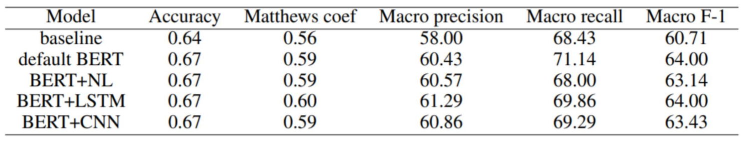
Data from social media can provide valuable real-time insight into local situations during times of disaster [15]. Therefore, the field of disaster management often leverages this data for tasks such as damage assessment, outbreak detection, and sentiment analysis. However, these data sets are enormous, noisy, and unstructured, which makes them difficult to work with. To address this, Stanford researchers decided to train a deep learning model to sort tweets into different categories (or labels), so that only relevant data can be analyzed. These labels include "Caution and Advice," "Not Related or Not Informative," etc., as shown on the table below.



Label	Count
not related or not informative	25785
other useful information	18877
donations and volunteering	8925
affected individuals	8009
sympathy and emotional support	5020
infrastructure and utilities damage	4559
caution and advice	3171

Figure 2: Classification labels for BERT in the Field of Disaster Relief [15]

This was accomplished by training various BERT models with a cumulative dataset of 75,800 tweets. The results are depicted below, and overall it was found that the BERT model outperformed the baseline model by about 3%. These results confirm that BERT is effective when applied to messy datasets of text-speak.



Model	Accuracy	Matthews coef	Macro precision	Macro recall	Macro F-1
baseline	0.64	0.56	58.00	68.43	60.71
default BERT	0.67	0.59	60.43	71.14	64.00
BERT+NL	0.67	0.59	60.57	68.00	63.14
BERT+LSTM	0.67	0.60	61.29	69.86	64.00
BERT+CNN	0.67	0.59	60.86	69.29	63.43

Figure 3: Results for BERT in the Field of Disaster Relief [15]

2.2.2 Visualizing BERT

Deconstructing BERT: Distilling 6 Patterns from 100 Million Parameters

This paper gives a surface level view of what BERT is and describes a technique to visualize the learned weights in the BERT model. It credits the importance of doing this to the fact that the process by which BERT fine-tunes these weights is hard to understand. The visualization of these weights aims to show exactly what BERT is learning. BERT can show what it's learned in many ways, but this paper summarizes the six most common learning patterns To see where the attention, or weighted average, in each of these patterns is, it will show a line from the left or selected token to the right or expected token with the opacity being how strong the attention is. The patterns were: attention to the next word, attention to the previous word, attention to identical/related words, attention to identical/related words in other sentences, attention to other words predictive of word, and attention to delimiter tokens. Figure 1 shows the attention to the next word visual graphic [16].

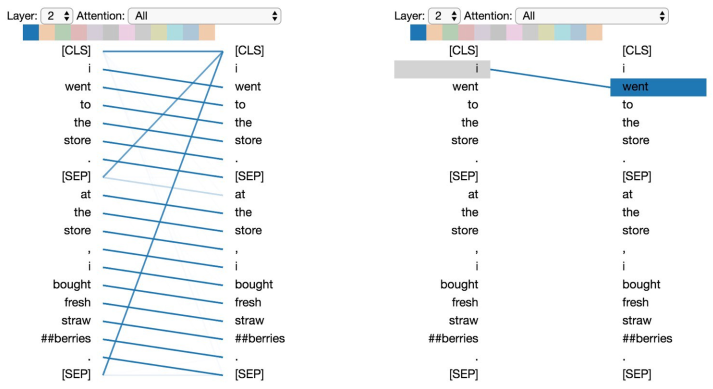


Figure 4: Next Word Attention
[16]

Deconstructing BERT, Part 2: Visualizing the Inner Workings of Attention

This paper focuses on BERT's attention mechanism. Attention is the importance of where you should focus in any given task and is another word for a weighted average. BERT makes use of attention by using multiple attention heads that work in parallel; it then adds layers of this that work off the previous layers, making for very rich representations at lower layers [17]. It goes into how attention weights are computed by a compatibility function, which gives a score to each pair of words and shows how much they should attend to each other. The model starts by assigning each word a query vector and a key vector, which are constructed for determining the compatibility of words. The compatibility score is found by taking the dot product of the query vector of one word and the key vector of the other. Taking these scores and making them attention weights, means normalizing them to positive and sum to one. This is done by using the softmax function over the scores for a given word. The final softmax value that is given is the final attention weight [17].

The paper also describes visualizing how attention weights are computed from the query and key vectors we talked about before, using neuron view. This view shows positive values as blue and negative values as orange, with the intensity of the color being its magnitude. The intensity of the lines also shows the strength of attention. These come together in this view to show how attention is computed from the chosen word to the end chosen words. The visual below shows a delimiter-focused attention pattern in the neuron view. The key vector for the two [SEP] tokens carry a distinctive look as they both have a few extremely positive (blue) or negative (orange) values and have a larger amount of spots that are closer to zero (light blue/orange or white). These spots then get further matched by the query vector, which causes the product to exemplify this with very positive (blue) spots.

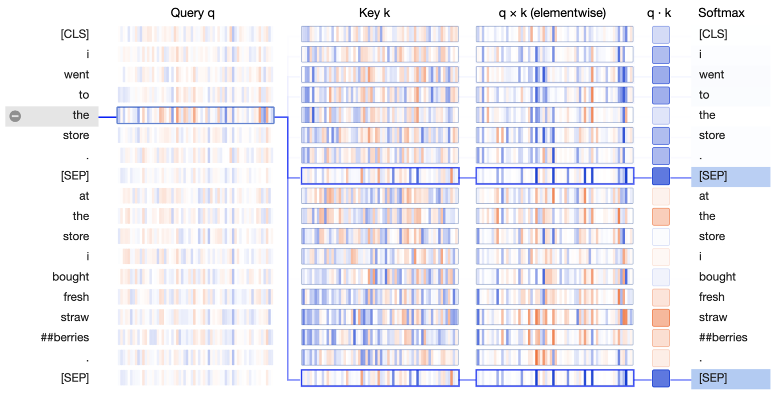
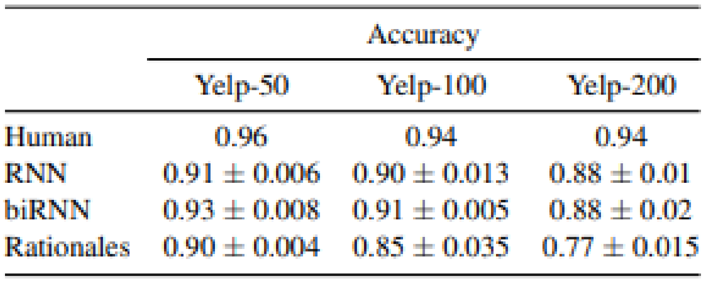


Figure 5: Neuron View on Delimiter Focused Attention [17]

2.2.3 Empath

Human Attention Maps for Text Classification: Do Humans and Neural Networks Focus on the Same Words?

In this paper the focus was on how close the attention maps created by humans and machines are. To do this there was a focus on the words focused on, the distribution over lexical categories and the context-dependency of sentiment polarity. The process of doing this was done three times with 50 word reviews, 100 word reviews and 200 word reviews and the groups of people determined sentiment and highlighted words that helped them make their decision. Then the machine was built and ran to make it's attention maps. The data showed that the amount of text there is decreases the accuracy of the machine as well as the similarity of attention between machine and human. It also showed that in general humans were slightly better at understanding sentiment from reviews with 50 words and the gap increased as the number of words increased [18].

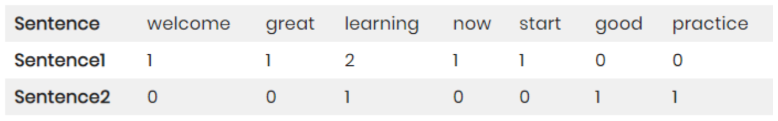


	Accuracy		
	Yelp-50	Yelp-100	Yelp-200
Human	0.96	0.94	0.94
RNN	0.91 ± 0.006	0.90 ± 0.013	0.88 ± 0.01
biRNN	0.93 ± 0.008	0.91 ± 0.005	0.88 ± 0.02
Rationales	0.90 ± 0.004	0.85 ± 0.035	0.77 ± 0.015

Figure 6: Test accuracy from three subsets of Yelp data.
[18]

An Introduction to Bag of Words (BoW) - What is Bag of Words?

This paper explains what the “Bag of Words” technique is, why it’s used, as well as its limitations. Bag of words is a NLP technique of text modelling that extracts features to show what words and how many times each word shows up in a sentence. Bag of Words takes messy and unstructured text and makes it structured in the form of numbers for machine learning algorithms to use [19]. One of the weaknesses of Bag of Words is the problem of using negative words to make a positive sentence or the opposite, this can be fixed with N-grams which groups words together and allows us to identify differences between sentences like “This is not good at all” versus “This is a good job. I will not miss it for anything” [19]. Bag of Words limitations include being unable to determine where words are in respect to each other and seeing words that mean the same thing as completely different [19].



Sentence	welcome	great	learning	now	start	good	practice
Sentence1	1	1	2	1	1	0	0
Sentence2	0	0	1	0	0	1	1

Figure 7: Bag of Words output for sentence 1: ”welcome great learning now start learning” and sentence 2: ”learning good practice”
[19]

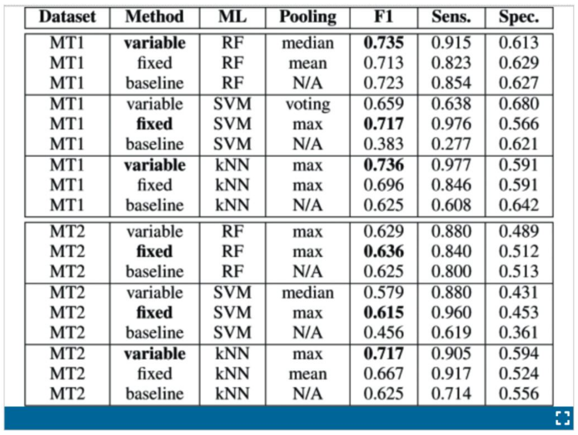
Understanding Depression from Psycholinguistic Patterns in Social Media Texts

In this paper, Alina Trifan et. al. from the University of Aveiro used Empath to detect how depression affects linguistic patterns by analyzing Reddit posts [12]. The posts were taken from 30,000 users who had either explicitly stated that they were diagnosed with depression or who had never made a post about mental health. After running different experiments with various classifiers and weighting schemes on the data – for example, using the TF-IDF technique with the Passive Aggressive classifier – they found that the language that depressed individuals used in their posts had significant differences from that of non-depressed individuals. Their conclusions were that depressed users typically use more negative emotion words, self-related speech, absolutist words, and use more words in posts [12].

2.2.4 Preprocessing Techniques

Audio-based Depression Screening using Sliding Window Sub-clip Pooling

Detecting signs of depression based on the human voice can be quite difficult, as the cues are subtle and voice datasets often have limited participants. To overcome these roadblocks, researchers applied a Sliding Window Sub-clip Pooling (SWUP) method [13]. Parent voice clips are divided into sub-clips, which are then pooled. Pooling is when the input size is decreased, effectively summarizing the input features. These pooled sub-clips enable depression detection as the signals occur in real-time, thus increasing the sensitivity to these signals. When evaluated on four datasets, it was found that SWUP outperforms the baseline models significantly, and that max pooling is the best version (as seen on the table below).



Dataset	Method	ML	Pooling	F1	Sens.	Spec.
MT1	**variable**	RF	median	**0.735**	0.915	0.613
MT1	fixed	RF	mean	0.713	0.823	0.629
MT1	baseline	RF	N/A	0.723	0.854	0.627
MT1	variable	SVM	voting	0.659	0.638	0.680
MT1	**fixed**	SVM	max	**0.717**	0.976	0.566
MT1	baseline	SVM	N/A	0.383	0.277	0.621
MT1	**variable**	kNN	max	**0.736**	0.977	0.591
MT1	fixed	kNN	max	0.696	0.846	0.591
MT1	baseline	kNN	N/A	0.625	0.608	0.642
MT2	variable	RF	max	0.629	0.880	0.489
MT2	**fixed**	RF	max	**0.636**	0.840	0.512
MT2	baseline	RF	N/A	0.625	0.800	0.513
MT2	variable	SVM	median	0.579	0.880	0.431
MT2	**fixed**	SVM	max	**0.615**	0.960	0.453
MT2	baseline	SVM	N/A	0.456	0.619	0.361
MT2	**variable**	kNN	max	**0.717**	0.905	0.594
MT2	fixed	kNN	mean	0.667	0.917	0.524
MT2	baseline	kNN	N/A	0.625	0.714	0.556

Figure 8: Results of Audio-based Depression Screening Model [13]

2.2.5 Beiwe

Use of Beiwe Smartphone App to Identify and Track Speech Decline in Amyotrophic Lateral Sclerosis (ALS)

In this paper, the researchers used the Beiwe application to identify the early symptoms of Amyotrophic Lateral Sclerosis (ALS). For each of the participants, the app would send them notifications each week that would ask them to complete a questionnaire and to record themselves reading a passage. The app would actively collect this data every week and then would passively collect a variety of other data to compare. Passively the app would collect the number of steps, battery life, GPS, among other data. This study showed how the "Beiwe platform provides a viable approach for [certain] symptoms and to track speech decline in ALS" [20].

Collecting and analyzing smartphone sensor data for health — Practice and Experience in Advanced Research Computing

In this study a team from the University of Texas used Beiwe in a study about pregnant women. The university used the application for data collection and created a dashboard to track the data's movement along the pipeline. The study used the pipeline to look at the participants' data over a span of time so they could see patterns or any changes in their routines. Evidence suggests that "signatures of health and disease, or digital biomarkers, exist within the heterogeneous, temporally-dense data gathered from smartphone sensors" [21]. From this, the study uses both physical and virtual sensors such as, accelerometer, GPS, power/screen state, call logs, among others.

2.3 Data Collection

Data collection is an essential part of the ongoing Emutivo project since the goal is to analyze private text messages to predict depression. This section will describe two existing datasets that will be used in this as well as a new data collection platform that will be used for future data collection and subsequent dataset generation.

2.3.1 Datasets

Amazon Mechanical Turk (MTurk) is a crowdsourcing marketplace that outsources posted jobs to a public distributed workforce. These jobs can be posted by anyone with an MTurk account and could include anything – from image recognition to survey participation. Previous MQP teams have used MTurk to create the Moodable and EMU datasets, which include the data types shown in the table below. Our team will used both the Moodable and Emu datasets for our research.

Active Data	
Surveys	Audio Recordings
Passive Data	
GPS	Text messages
Call logs	Calendar
Power state	Reachability
Social media content	

Table 1: Data Types in the Moodable/EMU Datasets [1]

Moodable

The 2017-2018 MQP team gathered data for this dataset by using the MTurk platform to deploy the Moodable application. This application then collected user data including PHQ-9 scores, texts, mobile files, voice samples, calendar, contacts, call logs, GPS locations, and social media content from over 300 crowd-sourced individuals. [1]

EMU

The dataset was collected by the 2018-2019 MQP team, using the same Amazon Mechanical Turk platform to deploy a different application called "EMU". The data collected by this app is similar to the Moodable dataset, but with one notable exception being the addition of GAD-7 screening scores from over 60 crowd-sourced individuals. [22]

2.3.2 Beiwe

Beiwe (pronounced bee-we) is an open-source research platform that was created for researchers to collect smartphone-based data from participants in their studies. The platform was developed by the Onnela Lab at the Harvard T.H. Chan School of Public Health with funding from the National Institutes of Health (NIH). The use of smartphone data allows researchers to study their participants' digital phenotypes – a term that was coined by the research team and refers to the continuous passive gathering of smartphone-based data on the individual level. The Beiwe platform is available as an open-source software to promote collaboration and reproducibility in studies. [2]

Parts of the Platform

The Beiwe platform is composed of two parts: frontend data collection applications for Android and iOS devices, and a backend system based on Amazon Web Services (AWS) cloud computing infrastructure for long-term data storage and analysis, as well as an online study management portal [20]. The frontend applications are written Java and Kotlin for the Android version, and Swift and Objective-C for the iOS version [2]. The backend code is written in Python 3.6, Django, and Flask, and uses the following AWS services: S3, EC2, Elastic

Beanstalk, and RDS [2]. The backend hosts an online study management portal, which allows researchers to create new studies and manage existing ones [21]. The following diagram shows a simplified version of the frontend and backend parts of the platform, as well as how they connect to each other.

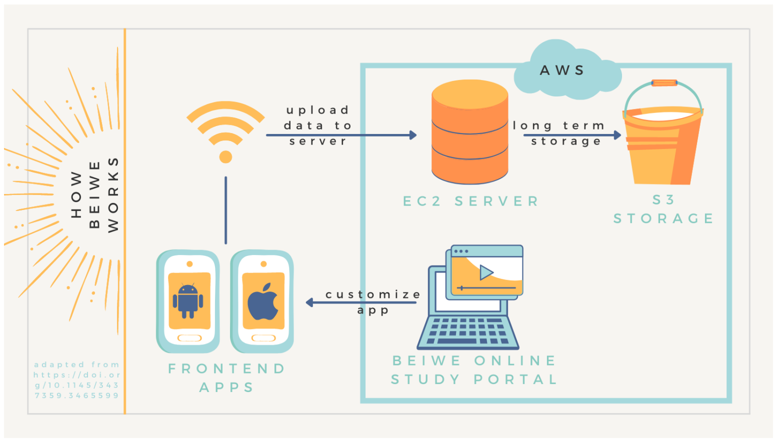


Figure 9: How Beiwe Works

Data Collection

Once a new study has been created, participants can download the frontend application on their personal smartphones and log in with their user ID, temporary password, and the URL address of the backend server – all of which are generated by the backend study management portal [2]. The Beiwe app allows participants to upload two types of data:

- Active data, which requires active involvement for its creation; and
- Passive data, which are created without any active involvement. [2]

The following table showcases what kinds of data Beiwe is able to collect.

Data Encryption and Storage

The Beiwe app will store all of the data collected from the participants in a CSV file that will then encrypt all the data with the public half of a 2048-bit RSA encryption key [21][2]. It will then upload the data to the AWS server whenever there is a Wi-Fi connection [21]. The server will re-encrypt the data with the study master key and ultimately route it to an S3 bucket [21][2].

Active Data	
Surveys	Audio Recordings
Passive Data	
GPS	Call logs - metadata
Message logs - metadata	Proximity
Power state	Reachability
Wi-Fi	Bluetooth
Device motion (accelerometer, gyroscope, magnetometer)	

Table 2: Data Types Collected by Beiwe [2]

Open-Source Materials

The code for Beiwe is all open-source and available on Github. There are a total of three Github repositories for different parts of the platform:

- Android app: `https://github.com/onnela-lab/beiwe-android`;
- iOS app: `https://github.com/onnela-lab/beiwe-ios`; and
- AWS backend: `https://github.com/onnela-lab/beiwe-backend`.

In addition to making the frontend code available on Github, the Onnela Lab also provides corresponding apps on the Apple App Store as well as the Google Play Store. This app is called "Beiwe2" and is meant to be downloaded by study participants in ongoing research studies.

2.3.3 AWS

Amazon Web Services (AWS) is a cloud-computing platform that offers over 200 services – from servers and data storage to artificial intelligence, and Internet of Things [23].

Simple Storage Service

Simple Storage Service (S3) allows you to store any amount of data and access it from anywhere. Data is stored in "buckets", which are secure and scalable.

Elastic Compute Cloud

Elastic Compute Cloud (EC2) allows you to build and run applications virtually. CPU, memory, storage, and networking capacity can be configured by choosing an instance type, such as the Beiwe AMI. Using EC2 allows your application to be secure and scalable.

CloudFormation

CloudFormation lets you model and provide other AWS services by creating templates to manage the configurations and infrastructure of the other services. The process of using CloudFormation is broken into three parts: creating a template, creating a stack, and finally building a running environment. CloudFormation allows you to set up a template, which is like a blueprint for AWS resources – it includes all the configurations for each resource. A stack is a subset of the AWS resources whose properties were outlined in the template – these are the resources you want to use to build your running environment. CloudFormation will then take the stack and build a running environment.

HIPPA Compliance

There is no formal HIPAA certification for AWS or other cloud service providers [24]. Instead, AWS meets requirements outlined by the Federal Risk and Authorization Management Program (FedRAMP) and National Institute of Standards and Technology (NIST) 800-53, which map to the HIPAA criterion [24]. However, though AWS can be naturally HIPAA-compliant, configuration mistakes can leave protected health information (PHI) unprotected and therefore accessible to unauthorized groups.

To solve this problem, AWS released their Quick Start program, from which you can choose pre-configured AWS services that align with HIPAA practices and use those services for your applications. The Quick Start program uses CloudFormation – you are given a template that includes various AWS resources that are configured to be HIPAA-compliant, and can then create stacks and run applications using those configured resources.

2.4 Machine Learning

Machine learning is a subset of artificial intelligence that helps computers learn how to make predictions from data using mathematical models and algorithms. It is extremely powerful as it allows the computer to learn without directly telling it how to make every decision. Machine learning algorithms are able to run on either labelled data (supervised learning) or unlabelled data (unsupervised learning), and can also replace a human agent (reinforcement learning). Algorithms can be used to solve either regression problems, in which the prediction is any numerical value – for example, 100 or 3.14 – or classification problems, in which the prediction is category – for example, "cat" or "dog." [25]

2.4.1 Classification Machine Learning Algorithms

There are four main kinds of machine learning algorithms: supervised, semi-supervised, unsupervised, and reinforcement machine learning algorithms [26]. Since the data that we used for this project was labelled with distinct PHQ-9 scores, we used supervised classification algorithms to build models to predict depression given some participant's text messages. This section describes the different types of supervised classification machine learning algorithms.

Logistic regression

Logistic regression algorithms fit a continuous sigmoid function – also known as a logistic curve – to the data. Logistic regression has categorical outcomes – for example, binary, multinomial, or ordinal – instead of continuous outcomes. [27] [28]

Naïve Bayes

The Naïve Bayes algorithm follows the principle of the Bayes Theorem, which calculates the probability of an event based on given true conditions, to calculate the probability that an event will occur. The algorithm is referred to as "naïve" because it believes that all variables are independent of each other, which is almost never the case in the real world. [29] [27]

KNN

The k-nearest neighbors (KNN) algorithm labels new data points based on the existing data points that are closest to the new point. Predictions are made based on a set number of nearest data points – this number is specified as the value k and is usually an odd number. The model will return the mode of the k nearest points. [29]

SVM

Support Vector Machines (SVMs) will draw an n-dimensional hyperplane between the two classes in the data by maximizing the distance between those classes. The vectors from those two classes are called support vectors, and the SVM will draw the hyperplane by maximizing the distance, also called the margin, between those support vectors (Figure 11). [29] [27]

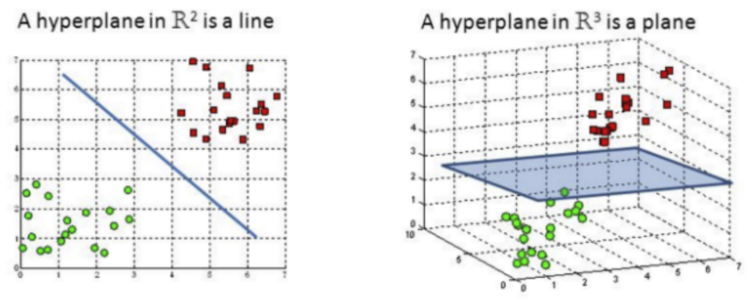


Figure 10: SVM hyperplanes drawn for different dimensions of data. [30]

Decision Trees and Random Forests

These algorithms model decision-making and contain nodes that represent questions about the data – for example, "Is the beetle pink?" Branches that stem from each node then split the data into homogeneous sets based on the answer to the question at the node. The data continues to be split and classified further until all data points have been classified. However, decision trees are vulnerable to overfitting. [28] [27] Random forests are built by combining hundreds of decision trees into one model. This "forest" of decision trees will aggregate votes from random formations of the trees in order to determine the final classification of the target (Figure 12). [29] [31]

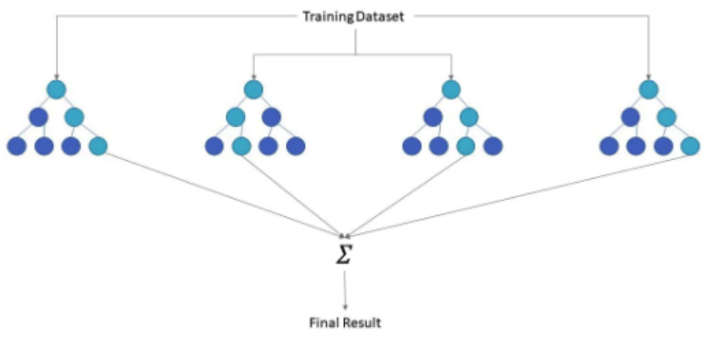


Figure 11: Random forest model containing four decision trees. [31]

Gradient Boosting

These are generic algorithms that aim to improve prediction performance by training a sequence of weak prediction models. The gradient boosting algorithm will train each subsequent weak model on what its predecessor did poorly on, thus compensating for all weaknesses. The result is a highly accurate predictive model. [29] [32]

Neural Networks

Neural network algorithms, also known as artificial neural networks (ANNs), make predictions by mimicking the way our brains work when processing new information. In the artificial neural network, new information is first passed through an input layer that is made up by features from the data, then various hidden layers, and finally an output layer that is made up by predictions. The figure below shows how the layers in an ANN are structured. The hidden layers contain interconnected neurons with weights and thresholds that are tweaked and fine-tuned as the neural network "learns" how to make accurate predictions from the data. The resulting model will pass the data along the layers and for each neuron it reaches, if the output from that neuron meets the threshold, the neuron will pass the data to the next layer. [28] [33]

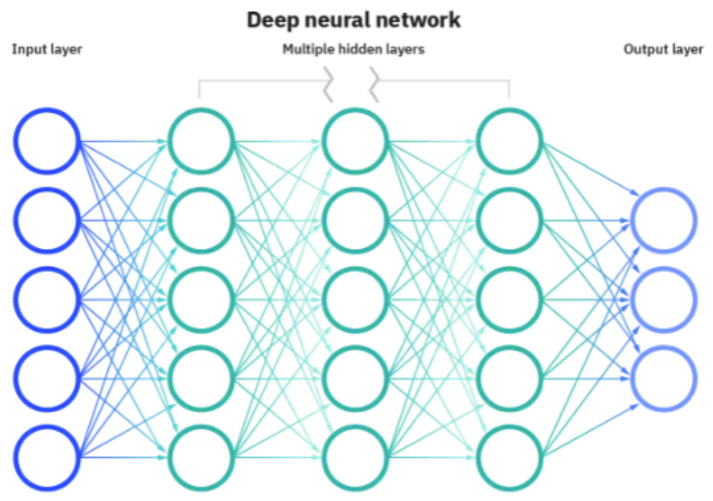


Figure 12: Layer structure in a neural network.
[33]

2.4.2 Classification Machine Learning Algorithms

In addition to refining machine learning models to predict depression from text messages, we also created visualizations to compare differences in texting topics among participants in different levels of depression. These texting topics were clustered into super-categories to make the visualization more intuitive. This section describes the most common types of clustering machine learning algorithms.

Centroid-Based Clustering

Centroid-based clustering is a technique that relies on the Euclidean distances between points in the datasets. The algorithm groups a data point to a cluster that results in the smallest square distance from the cluster centroid [34]. The most common centroid-based clustering algorithm is k-means, which groups data points into k clusters [35].

Density-Based Clustering

Density-based clustering groups the points that are in areas of high density [35]. This algorithm leaves out outliers [34]. A famous density-based clustering algorithm is density-based spatial clustering of applications with noise (DBSCAN) [34].

Distribution-Based Clustering

Distribution-based clustering assumes that the data is from a distribution such as the Gaussian distribution [35]. One example of a distribution-based clustering algorithm is the Gaussian Mixture Model algorithm [34].

Hierarchical Clustering

The use of hierarchical clustering implies that the data is hierarchical – for example, taxonomy or family tree data. This technique builds a tree of clusters that looks similar to a family tree, called a dendrogram [34]. In the beginning, the algorithm assigns each point to its own cluster. Clusters that are close together are grouped into larger clusters, and this process repeats until there is only one giant cluster [36]. There are many ways to compute the distance between two clusters – these methods are called linkage methods and are listed below [36].
Single

$$d(u, v) = min(dist(u[i], v[j]))$$

Complete

$$d(u, v) = max(dist(u[i], v[j]))$$

Average

$$d(u, v) = \sum_{ij} \frac{d(u[i], v[j])}{(|u| * |v|)}$$

Weighted

$$d(u, v) = (dist(s, v) + dist(t, v))/2$$

Centroid

$$dist(s, t) = ||c_s - c_t||_2$$

Ward

$$d(u, v) = \sqrt{\frac{|v| + |s|}{T} d(v, s)^2 + \frac{|v| + |t|}{T} d(v, t)^2 - \frac{|v|}{T} d(s, t)^2}$$

2.4.3 Evaluation Metrics

In machine learning, there are many different evaluation metrics that can be used to represent the effectiveness and accuracy of the model.

Confusion Matrix

A confusion matrix is used to describe the performance of a machine learning model on a set of test data. It is best represented by an NxN matrix of different combinations of predicted and actual values. The matrix is extremely useful for measuring precision, specificity, accuracy and AUC-ROC curves [37].

Accuracy

Accuracy is defined as the number of correct predictions divided by the total number of predictions made [38].

$$Accuracy = \frac{NumberOfCorrectPredictions}{TotalNumberOfPredictionsMade}$$

Precision

Precision is an indicator of a machine learning model's performance, it's the quality of a positive prediction made by the model. It measures the quality and the recall measure of quantity of the model.

Sensitivity

The sensitivity of a model is the accuracy which reports the presence or absence of a condition. It is also known as the true positive rate since it "corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points" [38]. The metric evaluates the ability to predict true positives for each of the available categories.

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

Specificity

Specificity, also known as the true negative rate, is when there is a portion "of negative data points that are correctly considered as negative, with respect to all negative data points" [37].

$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

AUC

Area under the curve (AUC) measures the entire two-dimensional area under the curve. The AUC is the area under the curve of a False Positive Rate vs. True Positive Rate at different points [38]. The greater the value of the AUC, the better the performance of the machine learning model.

F1 Score

An F1 Score is used to measure a model's test accuracy. It informs the viewer about "how precise [the] classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances" [38]. The F1 score is a combination of precision and recall which allows for higher precision but lower recall. As a result, it often "misses a larger number of instances that are difficult to classify" [38]. Since the F1 Score often misses pieces of data, it is often overlooked for more precise methods.

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$

MAE

Mean absolute error (MAE) is used to evaluate a Regression model. It finds the magnitude of the difference between the prediction of an observation and the true value of that observation. The metric gives the measure of how far the predictions were from the actual output of the model [38].

MSE

Mean squared error (MSE) finds the squared difference between the actual and predicted value. MSE is one of the simplest and most common loss functions used. MSE is very similar to MAE, the difference is "that MSE takes the average of the square of the difference between the original values and the predicted values" [38].

RMSE

Root mean squared error (RMSE) is used to evaluate the quality of predictions. RMSE is mostly used with supervised learning applications since it needs true measurements at each predicted data point.

R-Squared

R-Squared is a statistical measure which represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model [39]. The statistic indicates the percentage of the variance in the dependent variable that the independent variables explain correctly.

$$R^2 = \frac{var(mean) - var(line)}{var(mean)}$$

2.5 Natural Language Processing

Natural Language Processing, or NLP, is an interdisciplinary field between computer science, artificial intelligence, and linguistics that aims to equip computers with the tools to analyze and understand human language. NLP is a broad field that includes feature engineering techniques for analyzing text – such as lexical categories – and fully formed models – such as BERT.

2.5.1 NLP and Computational Linguistics

Before we begin to use different machine learning models to analyze text, we must first understand how to extract attributes and characteristics of data from the text. Feature engineering, which is the process of extracting those attributes and characteristics – which are called features – from data, is an extremely important technique used in both natural language processing (NLP) and computational linguistics. NLP is the use of computers to analyze natural language and is a term widely used by computer and data scientists, whereas computational linguistics is the study of linguistics through a computational lens and is a term widely used by linguists [40]. Though there are subtle differences between the two fields, the two terms can be used interchangeably [40].

2.5.2 Lexical Categories

In NLP, one way to create features from a document is through the use of lexical categories, also known as word categories [41]. Every word in a body of text has subtle meanings associated with it and the recognition of these signals in language is a growing interest in NLP [42]. For example, the previous sentence included words associated with communication ("word", "text", "language", "signal"), cleverness ("subtle", "recognition"), and health ("body"). These three categories – communication, cleverness, and health – are examples of lexical categories.

Empath

Lexical categories and their associated words are stored in vast collections called lexicons. When combined with machine learning models that are available to the public in the form of APIs, these lexicons allow researchers to quickly analyze a body of text to find hidden psycholinguistic patterns.

Unlike the LIWC lexicon, the Empath lexicon boasts more than 200 prebuilt categories that the model learned by analyzing modern fiction on the web [42]. Additionally, Fast et. al. have also released an Empath tool that allows researchers to create their own categories by using one of three models [43]. The descriptions of each model and the colloquial names we have decided to call them by are listed below:

- Empath model trained on modern web fiction — FanPATH
- Empath model trained on Reddit — ReddiPATH
- Empath model trained on the New York Times — NewsPATH

2.5.3 BERT

BERT, or Bidirectional Encoder Representations from Transformers, was developed by Google researchers in 2018 [44]. It is a semi-supervised model that is known to produce state-of-the-art results when applied to Natural Language Understanding (NLU) and Natural Language Processing (NLP) tasks.

Transformers are the basis of the BERT model, and it consists of an encoder and a decoder. A transformer consists of an encoder and a decoder. An encoder takes in all words, or tokens, in a piece of text simultaneously, which enables it to learn context [45]. It produces embeddings for each token, or vectors that represent the meaning of each token. Similar words will have closer numbers in their embeddings.

These embeddings are then processed through the decoder, which uses them to predict the next word. This process is repeated until the end of the sentence is reached. BERT is essentially stacked encoders, as depicted below.

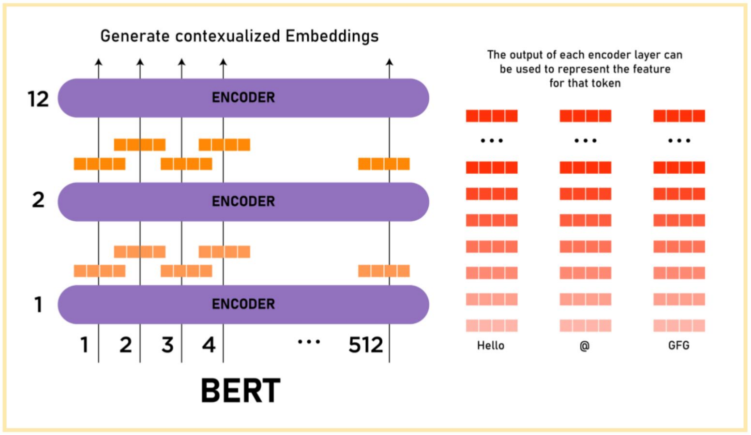

Generate contexualized Embeddings
12
ENCODER
2
ENCODER
1
ENCODER
1
2
3
4
...
512
BERT
The output of each encoder layer can be used to represent the feature for that token
Hello
@
GFG

Figure 13: BERT Model Encoder
[44]

BERT can perform a variety of NLP tasks, such as neural machine translation, question answering, sentiment analysis, and text summarization [46]. BERT is pre-trained to understand language, and then can be fine tuned to learn the specific task you are working for. Pre-training involves Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). With MLM, random words are "masked," or hidden, to train BERT to learn how to fill in the blanks. NSP is when BERT is given two sentences, and must predict whether the first sentence follows the first, or vice versa. Essentially, MLM helps BERT learn the context of words and NSP helps BERT learn the context of sentences. BERT is fine-tuned when the model is trained with an additional specific data set that aligns with the goal at hand.

2.5.4 spaCy

An open-source library, spaCy provides advanced NLP capabilities in Python [47]. The library provides many features including tokenization, lemmatization, part-of-speech tagging, named entity recognition, and similarity [47]. In this project, we used spaCy's similarity feature to calculate similarity scores between Empath category names. The similarity score is calculated by comparing the word vectors of two given words and results in a number between 0 and 1 [47].

2.6 Importance of Visualizations

Accompanying machine learning models with data visualizations is vital, as it can enhance one's ability to notice trends and outliers in the data. For example, if there is data with an upward trend, one will recognize this trend sooner by looking at a line graph than a spreadsheet of data. Further, one can identify outliers in this upward trend much more quickly and accurately by referencing this line graph. Data visualizations present the knowledge learned from the data in a story-like manner. They can be introduced in a way that is memorable, personable, and relatable. Statistics alone typically do not have the same extensive capabilities in informing a general audience as a visualization. It is also much easier to recall an image than it is to recall a number or statistic that was generated from a machine learning algorithm. Whether the purpose of a machine learning model is to prove, learn, or discover something about the data, the results should be presented in an easily digestible way so it can be shared with and understood by others.

3 Methodology

3.1 Lexical Categories

3.1.1 Empath

In order to create lexical categories to apply as features to the text messages, we first experimented with the built-in categories provided by the Empath tool developed by Ethan Fast [43]. The categories created by Empath, along with our own categories, were able to produce the results our team was looking for. However, we still wanted to find a way to make the code even more accurate. Very early on in the project we decided that we wanted to add new features and work on pre-processing the data.

URLs

When the text messages are analyzed, a URL shows up as characters unidentifiable by Empath. As we analyzed the messages with Empath to create lexical category features, the characters of the URL caused an error in the accuracy of the prediction in the message. Our team wanted to add in a function that will be able to identify a URL and remove it from the message before the analysis takes place. By doing this, we hoped that the predictions of the messages would be more accurate.

Emojis

When communicating with others over text, emojis are used to help convey emotions. Emojis tend to show people's emotions better than the words in the message themselves. We decided that adding a category to identify emojis would help to increase the accuracy of the predictions.

New Categories

In the Empath tool there are currently over 150 categories of words. As our team went through each of the different categories we thought of a few that could be added. The categories that we decided we wanted to add were drugs, environment, conflict, choice, absolutist, self-related, entertainment, and queer. To create these categories we plan on using the different Empath models which we named FanPath, NewsPath, and ReddiPath.

Each of the models were trained on different sites and have their own strengths and weaknesses. Our team plans on generating categories using each of the models and reviewing what words are generated. Once the categories are generated, we plan on going through and cleaning them so that the words in the categories are the ones that relate the most to the topic.

Improving the Efficiency of the Code

Existing code from the work the REU Emutivo team did over the summer of 2021 on lexical categories was not efficient [48]. Analyzing the text messages from only 5 participants with Empath took about 30 seconds to run. Since there are hundreds of participants, this would have resulted in a computationally expensive algorithm. Thus the team worked on improving the efficiency of the code.

Pre-processing

Pre-processing is required to get the data into the specific format Empath needs to run. Because Empath does lexical analysis on words, the characters need to be presented in a way that Empath will be able to identify. We tested many pre-processing techniques to see the effect they would have on Empath's classification code. We began by removing punctuation from the text and transforming the text into all lower case, for the purpose of finding exact word matches. In some experiments we then removed abbreviations and other text slang. In others we removed non-English entries using pyEnchant, which is a spell checking package for python. Each word was split into a list of individual characters. The list of characters of each word was compared against the extensive PyEnchant dictionary. If a word had more than four differences from the closest English word, it was deemed to be non-English [49]. The conversations with no remaining English words were then removed from the data set. We also tested word stemming and lemmatization techniques, as these could be useful for identifying different tenses or forms of words. We used the Wordnet extension of Python's natural language toolkit to perform these operations [50].

Some transformations had to be done to the data set of text messages after the physical texts themselves were cleaned in order to run a machine learning analysis. The team from the summer of 2021 had begun to transform the data, however it did not seem to be complete. The messages were imported and identified by a unique participant ID, and the features of each id included the body of the text, the number it was sent to, and the PHQ-9 score for the body of text. However, instead of looking at each individual message from each participant for analysis, we decided to group them by conversation, making Empath experiments run on Bag of Words experiments were run on **conversation level data**. Conversations were identified by the sender's ID and the recipient's phone number. The initial experiments only consisted of messages sent by the participant. We experimented with grouping the conversations by the ID of the sender as well to ensure that all conversations from one send end up in either the train or test split. Lastly, the set of sent messages was combined with the set of received messages to create a data set of the complete conversation. One problem that quickly arose was the high number of spam messages within the received data set. Spam messages are unsolicited or unwanted texts, and are usually advertisements from a company or automated reminders from an unknown phone number. After pre-processing the received data set in the same way as the sent, all of the messages were combined into one set of entire conversations.

3.1.2 Bag of Words

Bag of Words is a model that extracts features from text by disregarding everything besides what words show up and how often they do [51]. Bag of Words was implemented in separate experiments to compare results against Empath experiments. Empath cuts out words and focuses on categorizing words for further analysis. Bag of Words contritely shows us what keeping all of the words in the data set with no categorization does. Bag of Words experiments were run on **participant level data**.

Spell Checking

Similar to the Empath experiments, two different spell checkers were run, textblob and pyspellchecker. Textblob is a library for pre-processing textual data, and was used to fix typos and abbreviations that show up in text messages [52]. Pysypellchecker was used similarly. It spellchecks text and compares words with more than 2 errors with the closest word within the English dictionary [53]. This was done to fix any words that may not have been detected before. However, it did not account for non-english text and did not remove any entries.

3.1.3 Machine Learning

The conversation data set with the lexical features and the Bag of Words code was then run through the machine learning code from the 2019-2020 MQP using F1 scores to predict PHQ-9 scores of participants [41]. Adjustments had to be made to the train/test split to support the conversation-level data, as well as the conversations grouped by sender ID. Experiments were also run using the train/test split created for the BERT models described in the next section. The machine learning code calculates F1 scores to measure the most accurate model and feature set used to predict PHQ-9 scores from many different machine learning models. From there, the results of each experiment were compared using a box plot analysis code also adapted from the 2019-2020 MQP team's code.

3.2 BERT

Emoji Experiment

Our first experiments involved testing whether the inclusion of emojis in the text messages would impact the model's ability to determine whether the participant had depression. We decided to perform these experiments after finding prior research papers that modified BERT to accommodate emojis, which raised their accuracy by almost five percent [54]. We implemented the Python package Emoji 1.5.0, which can be used to transform emojis into text that is readable for the BERT model. Before giving the data to the model, we used the "demojize" function on the text messages. The text generated for each emoji was formatted with colons on either side of descriptive text of the given emoji. For instance, a winking emoji may be replaced with ":winkingface:". This allows the model to differentiate emojis from words because it can recognize the characters being used to represent them.

We added the emoji code to existing BERT code that had been produced over the summer by the REU team. This was an initial test to evaluate the potential of using emojis. The code uses the base BERT model that, alongside other BERT variants, is available through HuggingFace - a reputable startup that provides open source software, especially in the realm of NLP [55]. We chose to stick with the base BERT model at the moment but plan to investigate the potential of using other BERT models, such as BERTweet, in the future. For the initial emoji experiments, we ran five models with emojis and five models without emojis. We compared the two models by examining the average accuracy for each of them.

Increasing Code Efficiency

Following the conclusion of our emoji experiment, we decided to take the time to go over the previously written code. While the code from the summer was functional, we did not know whether it was the optimal way to approach the problem. Over the course of two days, we went through the code line by line until we understood every part of it. During this process, we eliminated unhelpful or unused parts of the code and commented on the purpose of each section. The newly cleaned code was moved into a new file that could be modified so we could conduct future experiments.

Test Set

A major challenge we encountered was the presence of participants with large amounts of data in the dataset. If these participants, which we referred to as "whales", were included in the test set, we risked having too little data to accurately train the model. Our solution was to remove the "whales" from the list of participants before creating the test set. We defined "whales" as participants with more than five chunks of data, where one chunk of data was 200 words. These chunks were created by adding messages from a participant together until the maximum size was reached. By dividing the participants by their number of chunks, we could ensure that only participants with a small amount of next, which was defined as five or less chunks of data, could be included in the test set.

Another challenge we faced was that the dataset with only the small participants had a different ratio of depressed to not depressed participants when compared to the overall dataset and the training dataset. Additionally, we had to ensure that no participant was in both the train set and test set. We chose to loop through random seeds until we found one that had a distribution of depressed and not depressed participants in the test set that reflected the ratio in the overall dataset.

After successfully solving these problems, we created a testing dataset that consisted of approximately 20 percent of the overall data. We recorded the participants' IDs in both the train set and the test set so the same test set could be used for all future experiments. Finally, by modifying the function that outputs the validation statistics, we were able to output statistics, such as the confusion matrix and the accuracy, for the test set.

Child IDs

For the training set, we wanted to implement cross validation by dividing the data into folds. These folds needed to have a consistent ratio of depressed and not depressed chunks, and needed to be approximately equal in size. We accomplished this by splitting the large participants, or “whales”, into individual participants with five chunks each. We assigned each group of five chunks a child ID that was appended onto the end of the parent ID. For example, “e1528” could become “e1528-0” and “e1528-1”. A visual example is included in the graphic below.

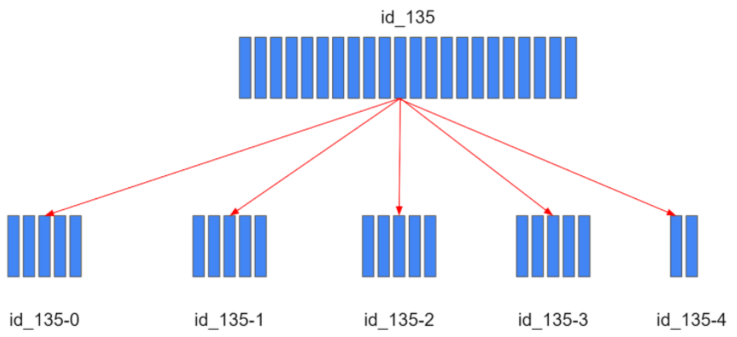


Figure 14: Visual Example of Splitting A Whale ID Into Smaller Child IDs

By dividing the “whales” into smaller participants, we were able to more easily create the even folds for cross validation. This was achieved by allowing different child IDs to be in different folds, even if they shared a parent ID.

Hyperparameter Tuning

Next, we ran experiments to determine the optimal hyperparameters to yield the highest accuracy, focusing on the number of epochs, the dropout rate, the batch size, and the number of folds. We tested these hyperparameters using the uncased version of the base BERT model.

For the epoch experiments, we experimented with a range of options that included 5, 10, 15, and 20 epochs. Each option was run with five random seeds (1-5), and the accuracies were calculated using the average of those five runs. We also varied the dropout rate for these experiments and ran each number of epochs with a dropout rate of 0.1, 0.3, and 0.5. The batch size experiments were run with sizes of 12, 16, 20, 25, and 32. Each batch size was run with five random seeds (1-5), and the accuracies were calculated using the average of those five runs. For the number of folds, we tried 3 folds, 4 folds, and 5 folds.

Each number of folds was run with ten random seeds (1-10), and again the accuracies were calculated using the average of those ten runs. Each of these hyperparameters were tuned using the validation accuracy.

BERT Models

Our initial experiments were all run with the uncased base BERT model, but we decided to test other additional BERT models to determine whether a different model could give us better test accuracies. The second model we chose to focus on was the cased base BERT model because it would allow us to learn how capitalization within the text would impact BERT's ability to accurately label the data as depressed or not depressed. We also added the ALBERT model to act as an example of a smaller version of BERT. By choosing a model with far fewer parameters than base BERT, we could learn how varying the number of parameters could negatively or positively affect our statistics. The final model that we decided to implement was the RoBERTa model, which was pretrained on more text than the base BERT model, which was only pretrained on BooksCorpus and English Wikipedia. We aimed to see if the increased text knowledge could help RoBERTa to outperform base BERT when classifying texts.

Data Input Types

Our team ran experiments with a total of thirteen different data input types. Each set of experiments was run with the optimized hyperparameters and five (1-5) random seeds for each of the four models we decided to test (base BERT, uncased and cased, ALBERT, and RoBERTa). We also kept the participants in the test set consistent to enable accurate comparisons across data types.

The first data type that we experimented with was chunks of 200 words. This dataset was made prior to the creation of our train and test sets. We referred to this dataset as "sent14chunked200." We then ran the models with the individual messages for the participants. We marked this dataset as "messageLevelTrain." These experiments would help us to see how the statistics from the chunked data compared to those from the individual messages.

Our next experiments involved using a sliding window to create overlapping chunks. The sliding window started with the first message for a participant and added text messages until the goal of "X" characters was reached or until no messages for that participant were left. We created datasets of 250, 500, 1000, and 2000 character chunks. We named the files for each of these datasets according to their character count - "chunkedDataFrame2000," "chunkedDataFrame1000," "chunkedDataFrame500," and "chunkedDataFrame250." We ran each of the models with each of the chunked datasets created using the sliding window approach.

As a comparison to the character-based chunks, we also used a sliding window to create chunks based on a number of messages. These chunks were created by adding messages from a participant until either the participant ran out of messages or the desired amount of messages was reached. We specifically chose five messages so we could still have enough data to accurately train the models. Like the character-based chunks, these message-based chunks also had overlapping. We referred to this dataset as “chunkedBy5Messages”.

Our next data input type was a conversation-level dataset that included both sent texts as well as received texts. Previously, our experiments had focused solely on sent text data so this was a new set of data to test with. Each chunk of text in this dataset was a conversation, where one conversation contained all the texts sent and received between two phone numbers. This dataset was referred to as “conversationLevelTrain”.

Following these experiments, we realized that the 200 word chunk training set had some overlapping IDs with the testing set. This situation occurred because those chunks were created prior to the development of the train and test splits. To fix this issue, we remade the 200 word non-overlapping chunks and also made chunks of 50, 100, 250, and 300 words as well. We added the number of words in each chunk to the names of the files - “chunked50word,” “chunked100word,” “chunked200word,” “chunked250word,” and “chunked300word.” We ran each of the models with the newly created non-overlapping datasets.

Datasets

The table below presents each of the previously mentioned datasets that were utilized in our experimentation, along with the size, whether or not the messages overlap between chunks, the number of data entries (or rows), and the percent of positively labeled data entries.

Name	Size	Overlapping	Count	% Positive
messageLevelTrain	One text message	No	4051	31.6218
chunkedBy5Messages	5 messages	Yes	4051	31.6218
chunkedDataFrame250	250 characters	Yes	4051	31.6218
chunkedDataFrame500	500 characters	Yes	4051	31.6218
chunkedDataFrame1000	1000 characters	Yes	4051	31.6218
chunkedDataFrame2000	2000 characters	Yes	4501	31.6218
conversationLevelTrain	Various amount of messages	No	176	48.8636
sent14chunked200	200 words (IDs in both train and test)	No	321	37.3832
chunked50word	50 words	No	657	34.8554
chunked100word	100 words	No	390	35.3846
chunked200word	200 words	No	217	35.9447
chunked250word	250 words	No	183	37.1585
chunked300word	300 words	No	158	36.7089

Documentation

Throughout the duration of the MQP, our team updated the README document in the MQPBert branch of the GitLab repository to reflect the optimized hyperparameters alongside the IDs that were included in the train and test sets.

Additionally, we created a series of visualizations that reflect how our code works. These illustrations solidified our understanding and will allow for easier replication of our work by future members of the project.

3.3 Lexical Categories Visualization

In addition to predicting depression from private text messages using basic machine learning models with lexical categories, or by using BERT, the team also wanted to create data visualizations. The goal of building these visualizations would allow for a better understanding of the results from our experiments and to bring a more humanistic touch to the data. Additionally, we want to create art pieces associated with our project that leave a long-lasting impression on the audience.

The Data

The Moodable dataset was used to build this visualization. First, the text messages and PHQ-9 scores were taken from the Moodable dataset, analyzed with Empath to create lexical category features, and then we split the featured dataset into subsets where each participant had the same PHQ-9 category – asymptomatic, mild, moderate, moderate-severe, and severe. We then ran statistics on each PHQ-9 subset to compute the mean score of each lexical category, and normalized the mean values. Finally, we sought to build a visualization for each PHQ-9 category using the normalized means and the steps described in the following sections.

Gathering Inspiration

Gathering inspiration and planning are essential requisites to building or creating an important project. This is why artists will use pencil on canvas before painting, why architects create blueprints, and why writers create outlines and character sheets. In the beginning, a Google Doc was used as a storage place for any found data visualizations that were relevant to the field of computational linguistics. However, this quickly became hard to navigate as we realized that scrolling up and down a document when attempting to find a certain visualization was inefficient, and the format did not invoke any feelings of inspiration.

To answer this problem, a Pinterest board was created to store all of the data visualizations, data art, artwork, and various other creations that caught our eye. This enabled us to not only see a minimized and organized array of awe-inspiring visuals, but also to find patterns and subgroups within the collection.

We grew our board through three main ways:

1. Adding projects that we were already aware of prior to this project;
2. Adding data visualizations from research conducted during this project; and
3. Adding recommendations from the Pinterest algorithm.

Adding projects that we were already aware of prior to this project

The captivating works of Giorgia Lupi, Nadieh Bremer, and Gabrielle Mérite, among others, were added to the Pinterest board. These data artists, information designers, and visualization experts have been an inspiration in the past and continue to inspire in the present. Additionally, data visualizations about mental health that were already known – for example, A View on Despair by Sonja Kuijpers – were added to the board.

Adding data visualizations from research conducted during this project

Visualizations such as the EmotionWatch created by Swiss Federal Institute of Technology Lausanne researchers were found after scouring the web for relevant linguistic data visualizations [56]. This, among others, was also added to the Pinterest collection.

Adding recommendations from the Pinterest algorithm

One of the most unique features of the Pinterest website is its ability to recommend visually similar suggestions to the "pins", or posts, that are already saved to your personal boards. By using this algorithm, the team was able to find beautiful creations that we may not have thought to look for – for example, a calendar containing data visualizations about the environment, various infographics, and data journalism.

3.3.1 Planning and Drafting

Once we had an ample amount of inspiration from our Pinterest board, we continued on brainstorming and drafting ideas for the lexical categories data visualizations. These ideas were both inspired by and honored the following goals:

- Visualizing how depression affects linguistic differences in texts;
- Creating art that induces an emotional response to hard data about depression; and
- Understanding the output from our lexical categories machine learning pipeline in a more intuitive way.

After the brainstorming stage, paper and pencil was used to make the abstract ideas more concrete and to draw out possible projects. Final ideas were drawn in a graphics editor app for a more professional finish.

3.3.2 Creating the Visualizations

D3.js was used to develop the data visualizations. Andrea Mignone kindly gave us permission to reuse the code for his flower plots, which were found on the data visualization platform Observable [57]. Though Mignone's original flower plots were intended to be a marriage between a pie chart and a bar graph, for the purpose of this project we treated the flower plots as pie charts. We chose these flower plots to highlight the big picture instead of numerical details, as well as to create something beautiful from data about depression.

The Big Picture

Our team was more interested in the contribution of each Empath category score contributed to the whole picture, so we looked into using radial visualizations like pie charts. We were less interested in details like the exact size of each Empath category score than an estimated relative size of the category to other categories. We also wanted a visualization that was compact so that comparisons could be drawn without using too much eye movement and brain power.

Creating Something Beautiful

Depression can be a hard topic to bring up in conversation because of the stigma, triggers, and unpleasant feelings around it. Our team wanted to give our researchers, study participants, and the general public some semblance of satisfaction when they stop to smell the roses – or in this case, when they stop to view the flowers.

3.3.3 Creating Super-Categories

We quickly realized that the flower plots could be improved by color-coding by Empath categories instead of by PHQ-9 categories. Some categories – like "banking", "wealthy", "payment", and "business" – are very similar to each other. The Empath categories can be summarized into groups – for example, "work and wealth" for the previous words. We created these "super-categories" by experimenting with machine learning techniques and by doing it manually.

Machine Learning Techniques

Natural language processing (NLP) and machine learning clustering techniques can be used to group the Empath categories into clusters based on similarity. Similarity between two words can be calculated using the spaCy package, which turns a given pair of words into respective word vectors and calculates the distance between the two vectors [58]. The similarity score was calculated for each pair of Empath category names, and the scores were then normalized so that the machine learning algorithms did not group all of the scores into one giant cluster.

There are four common types of clustering algorithms: centroid-based clustering, density-based clustering, distribution-based clustering, and hierarchical clustering [35]. The data generated by calculating the similarity scores between Empath category names resulted in a distance matrix. Since centroid-based clustering relies on Euclidean distance, we could not use that method since our distance matrix was not Euclidean [35]. Similarly, the distance matrix did not imply that the data fit a distribution, so we could not use distribution-based clustering [35]. Thus, we experimented with density-based clustering and hierarchical clustering to group the Empath categories into super-categories.

Manual Clustering

We used draw.io, an online diagram maker, to manually cluster the Empath category names. After inputting all the Empath category names as bubbles in draw.io, we clustered the category names using our knowledge of their semantic meanings.

3.3.4 Color Coding Using Empath Super-Categories

The flower plots were then color-coded based on the super-category of each Empath category petal. The color palette for the super-categories was tested for efficacy with different color blindness filters. By color-coding the super-categories, we could easily discover trends in what participants in each PHQ-9 category tended to text about.

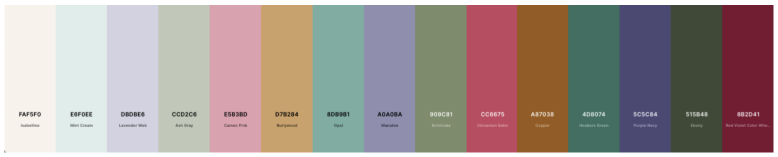


Figure 15: Color Palette for the Super-Categories

3.4 Beiwe

The open-source data collection platform Beiwe is complex and can be customized extensively to collect and analyze participants' smartphone phenotypes. The team's existing knowledge of this state-of-the-art platform was limited at the start of the project, but its open-source status meant that deep exploration could be done on its capabilities.

However, although Beiwe is an open-source platform, this does not mean that its implementation will be completely free of charge. The platform uses various Amazon Web Services (AWS) platforms including the EC2 Server, which is estimated to cost about $62 a month, and S3, which has variable costs depending on how much data will be stored in it [59].

Based on this information, our first goal was to *explore the logistics of using Beiwe for data collection.* This included obtaining a general overview of how the platform works, as well as what kinds of data it could collect.

Our second goal was to *test the viability of Beiwe to run without using AWS* so this project could continue without additional funding or costs. There were two main solutions to this problem that were proposed:

- Storing the data collected by the frontend before it uploads to the backend; and
- Running the backend on a VM instead of using AWS services.

3.4.1 Researching Beiwe's Capabilities

We began looking into Beiwe by doing research on its official website (Beiwe site). The site gave us the background we needed to understand Beiwe's capabilities. Beiwe is an open-source application so we started by analyzing the Github pages and created an analysis document which can be found in Appendix A. Once we understood a bit about the code our team wanted to look into studies that used Beiwe to collect data for various studies. Through these studies, we began to understand how Beiwe worked and how we could use Beiwe for our own study.

3.4.2 Storing The Data Collected By The Frontend

In order to test the viability of Beiwe to run without using AWS services, one solution was to re-route the data collected by the frontend to a secure data management platform other than AWS, such as REDCap. Beiwe has the compatibility to run on both iOS and Android devices. The application can be downloaded for free from the app store – "Beiwe2" from the Apple App Store and the Google Play Store. The code is also open-source and available for download from Github. Even though the app can be downloaded from the app store, by working with the open-source code, our team had the ability to control the different functions of Beiwe.

iOS

When we began working with the iOS frontend code, we downloaded it from the Github repository and made sure to save it on our computer. We worked with Xcode to debug, compile, and run the code. As we began to work with the code, there was not much documentation on the Github so we had to figure things out as we went. Our team documented everything we worked on and the errors we encountered which can be found in Appendix B.

Android

Similar to the iOS frontend code, we downloaded the Android frontend code from the Github repository and saved it to our computer. Compared to the iOS frontend code, the Android code was significantly more documented. As we began our debugging process, we started by following the instructions that we laid out to us. We used Android Studio to debug, compile, and run our code for the frontend. Similar to when our team was working on the iOS code we made sure to document all the errors we encountered and we compiled them into a document which can be found in Appendix C.

Testing the Viability of Using REDCap

The Emutivo team has an existing account with REDCap, a secure web platform for creating and managing both databases and surveys. Thus, the team wanted to test the viability of connecting the backend to a REDCap database for long-term data storage and to avoid using AWS S3 buckets, which could be costly. Our goal was to discover if the frontend code could be modified to upload data to REDCap instead of an S3 bucket.

3.4.3 Running The Backend Locally

To test the viability of Beiwe to run without using AWS, another solution was to run the backend on a VM. Running the backend on a VM would have provided the following benefits:

- Immediate implementation of Beiwe without applying for funding for AWS services; and
- Using a frontend app to test if participants can log into a study.

Immediate Implementation of Beiwe

To run the Beiwe backend open-source code found on Github, a virtual machine (VM) was set up to serve the backend. We first tried an Ubuntu VM; however, this made the backend run locally and the study management portal could only be accessed inside the VM. To make the portal accessible outside the VM, a development VM was set up on the WPI servers and given a unique URL that could be accessed by any device, even outside of the WPI wifi network. Documentation for setting up an Ubuntu VM can be found in Appendix D and documentation for setting up Emutivo's development VM can be found in Appendix E.

Running the code on a VM allowed the team to immediately be able to create new studies with the Beiwe study management portal. Additionally, running the backend on the VM allowed for further testing with frontend integration.

Testing With Frontend Apps

The team wanted to determine whether running the backend on a VM could be a long-term alternative to using AWS. To do this, ready-made frontend apps – "Beiwe2" from the Apple App Store and the Google Play Store – were downloaded. To test whether the studies created by the local server could be used in a real study, the team attempted to log into the Beiwe2 apps using information generated by the backend for a given study. In addition to using the Beiwe2 app, the team modified the open-source frontend code to connect to the backend, then attempted to log into the frontend app.

3.4.4 Making AWS Services HIPAA-Compliant

In order to run Beiwe on AWS services, the team first made an AWS Educate account, which offers up to $100 in credits for WPI students [60]. We then researched how to make the EC2 and S3 services HIPAA-compliant.

4 Results

4.1 Lexical Categories

4.1.1 Empath

The dataset created using Empath resulted in 200 feature sets. When running this through machine learning code the resulting F1 score was 0.72.

New Categories

In training the eight categories that we added to the original Empath categories, we used a different combination of hard coding, as well as the three Empath tools to achieve the appropriate words for each category. For the categories absolutist and self-related, we hard coded the words that fall under each category because of the small number of words, and the repetition that occurred among words when we ran the Empath tool. For the category queer, we ran reddiPath first to get a foundation of words, then hard coded ones we felt were missing, and removed more repetition. For choice and entertainment, we ran newsPath and fanPath, as news sources and fiction had more obvious expressions of words in each category than ReddiPath did. Conflict was run on fanPath and reddiPath, as news sources often gave a one-sided, unspecified description on perceived conflict. Lastly, the categories of drugs and environment were trained on all three Empath tools.

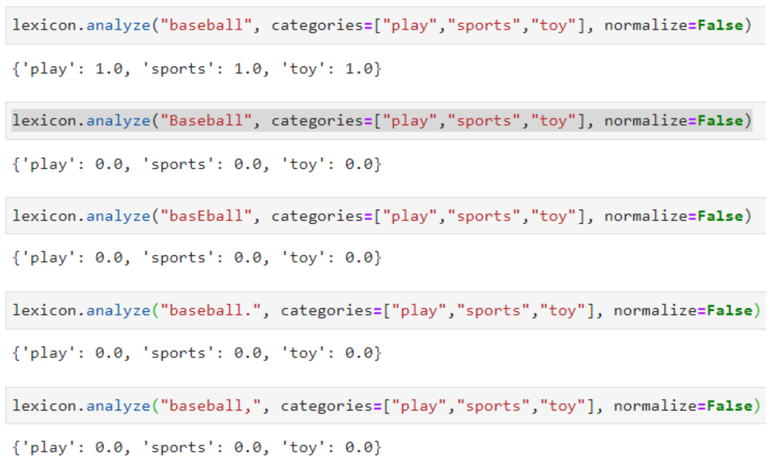
Besides the hard-coded categories, we initially had 300+ words in some of the categories. To fix this, we went through and removed words that had been repeated, words that were unrelated, and phrases that contained underscores. Doing this enabled us to cut down the categories to 100-200 words each.

Improving Efficiency of the Code

Previously, using Empath to analyze the text messages of 5 participants took about 30 seconds to run. The team worked on improving the efficiency of the Empath analysis algorithm, which resulted in an algorithm that took less than a second to run on 5 participants.

Text Cleaning

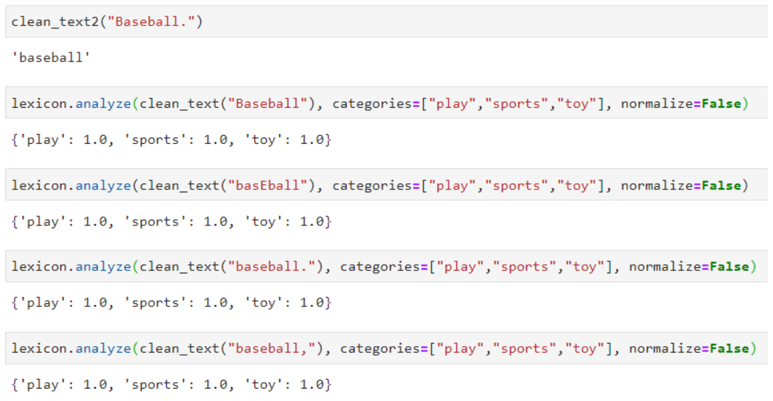
The effect of pre-processing is immense on the text as words that were previously not recognized due to syntactical errors such as capitalization and punctuation can now be analyzed using Empath. The effectiveness of fixing these problems can be easily tested with the existing Empath model and sample words and sentences.



```
lexicon.analyze("baseball", categories=["play","sports","toy"], normalize=False)

{'play': 1.0, 'sports': 1.0, 'toy': 1.0}

lexicon.analyze("Baseball", categories=["play","sports","toy"], normalize=False)

{'play': 0.0, 'sports': 0.0, 'toy': 0.0}

lexicon.analyze("basEball", categories=["play","sports","toy"], normalize=False)

{'play': 0.0, 'sports': 0.0, 'toy': 0.0}

lexicon.analyze("baseball.", categories=["play","sports","toy"], normalize=False)

{'play': 0.0, 'sports': 0.0, 'toy': 0.0}

lexicon.analyze("baseball,", categories=["play","sports","toy"], normalize=False)

{'play': 0.0, 'sports': 0.0, 'toy': 0.0}
```

Figure 16: Running Uppercase and Punctuation Through Empath

As seen in the figure above, "baseball" is a word contained in Empath, but as soon as you add any punctuation or capitalization it can no longer be identified. The answer for these problems is as simple as a cleaning function that will result in outcomes shown in the figure below.



```
clean_text2("Baseball.")

'baseball'

lexicon.analyze(clean_text("Baseball"), categories=["play","sports","toy"], normalize=False)

{'play': 1.0, 'sports': 1.0, 'toy': 1.0}

lexicon.analyze(clean_text("basEball"), categories=["play","sports","toy"], normalize=False)

{'play': 1.0, 'sports': 1.0, 'toy': 1.0}

lexicon.analyze(clean_text("baseball."), categories=["play","sports","toy"], normalize=False)

{'play': 1.0, 'sports': 1.0, 'toy': 1.0}

lexicon.analyze(clean_text("baseball,"), categories=["play","sports","toy"], normalize=False)

{'play': 1.0, 'sports': 1.0, 'toy': 1.0}
```

Figure 17: Running Same Problematic Text Through Pre-processing Function

However, harder problems arise when tackling issues like typos, spam messages, languages other than english, and text abbreviations. The steps we took to solve these problems are highlighted in the following sections.

Spell Checking

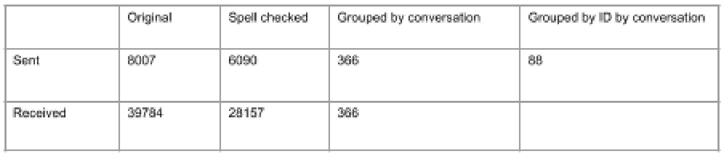
Using the pyEnchant spell checker to remove non-English entries refined the number of sent messages from 8,007 to 6,090, and the received data set from

39,784 messages to 28,157 messages. This also proved useful in detecting spam messages in the received data set. Upon further investigation, we found this is because spam messages often contain random sets of characters as a coupon, only a URL, or other content that would not be detected as English. Spam messages ultimately accounted for so many more entries being dropped from the received data set than the sent.

Grouping the Data set

After the data was cleaned, it was grouped by conversation. The sent message set was grouped into 366 different conversations. We also experimented with grouping the conversations by the sender's ID, but that proved to be a bit restrictive on the data set, as the index was refined to only 88 ID's, each with multiple conversations associated. When combining the sent messages with the received, the data was grouped only by conversation.

Ultimately, the message data set was refined to these sizes at each respective step:



	Original	Spell checked	Grouped by conversation	Grouped by ID by conversation
Sent	8007	6090	366	88
Received	39784	28157	366	

Figure 18: Number of Entries in the sent and recieved data sets after each preprocessing step.

4.1.2 Bag of Words

Running Bag of Words on the dataset created a new feature set that had 5,349 words in it compared to Empath's 198 categories. These features were also measured differently as rather than being a measurement between zero to one like Empath, Bag of Words counted how many times each word showed up. The F1 score from Bag of Words was 0.75.

Spellchecking

Running textblob on Bag of Words showed an increase from 5,349 words to 6,122 words in the dataset. On the other hand running pyspellchecker on Bag of Words showed no change in the amount of feature sets. The F1 scores for textblob with Empath were 0.72 for both running the spell checker before and after cleaning the text. For running text blob on Bag of Words the F1 scores were 0.74 for running it before cleaning the text and 0.75 for running it after cleaning the text. The F1 scores for pyspellchecker on Empath were 0.70 for running it before cleaning and 0.72 for running it after cleaning. For pyspellchecker with Bag of Words it was 0.74 for both before and after cleaning.

Minimum Text Messages

Testing for the best amount of minimum texts was done on the previously determined best parameters which was not running spell checker, running only on the sent texts database, running for Bag of Words and Chi for the features. Increasing the requirements on minimum text messages slowly decreased the amount of participants as it went from 129, 114, 105, 97, 88, 82, 78, and 75 for the amount of participants as the minimum increased from one text to eight texts. Running all these text messages on Bag of Words resulted in F1 scores 0.75, 0.75, 0.77, 0.78, 0.74, 0.74, 0.74, 0.72 for one text message minimum to eight text messages minimum respectively.

Different Data sets

Three different data sets were observed, those being sent text messages, received text messages and the combination of sent and received text messages. All three data sets were tested with bag of words and 4 minimum text messages being sent. The F1 scores were 0.78 for just sent text messages, 0.73 for received text messages and 0.74 for the combination of sent and received text messages.

4.1.3 Machine Learning Box Plots

Conversation Level Data

The following box plots demonstrate the accuracy of the best models for predicting PHQ-9 scores of different variations of the text message data at the conversation level.

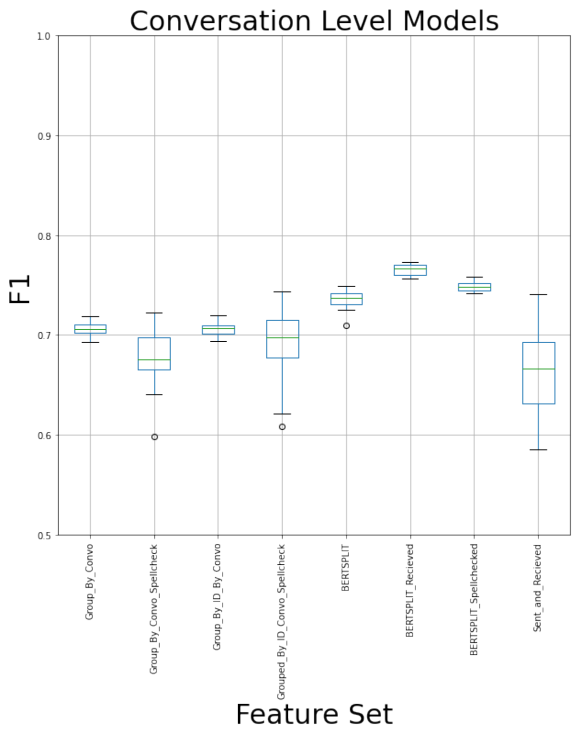


Figure 19: Best Models for Predicting PHQ-9 Scores on Conversation-Level Data

The box plots highlight significant results. First, the BERT train/test split out performed the existing train/test split methods from the summer team. Most surprisingly, the sent and received set (BERTSPLIT-Received) had the highest average prediction accuracy at .76, whereas the original splitting method averages consistently hovered around .7.

4.1.4 Participant Level Data

The following box plots demonstrate the accuracy of the best models for predicting PHQ-9 scores of different variations of the text message data at the participant level.

The following box plots use the described naming conventions. "Default" or not mentioning "BOW" means that it used the original Empath code found on github. "BOW" means that it used Bag of Words. "blob" refers to the use of Textblob. "pyspell" refers to the use of pyspellchecker. "After" and "Before" refer to when the spell checker was used in comparison to cleaning the text (Before means spell checker was before text cleaning and after means it was done after text cleaning). And "clean" is slightly redundant as they were all cleaned, but it was used with the spell checkers name and ”before” and ”after” to help naming.

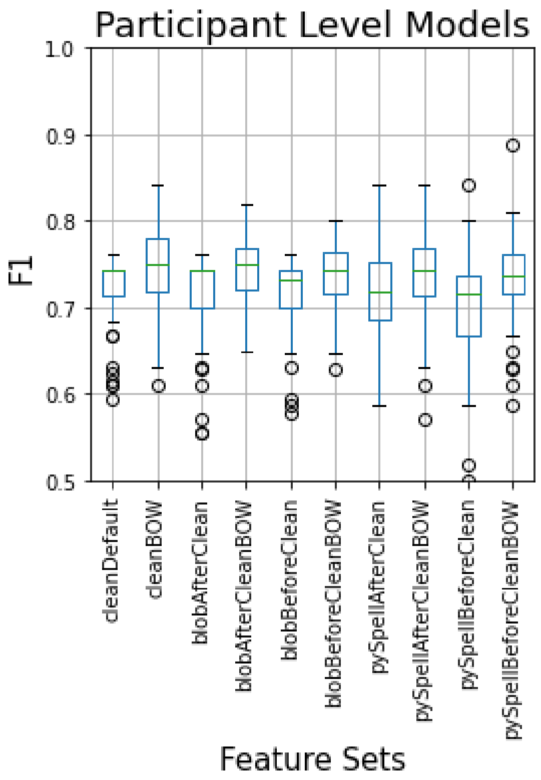


Figure 20: Best Models for Predicting PHQ-9 Scores on Participant-Level Data For Chi Squared

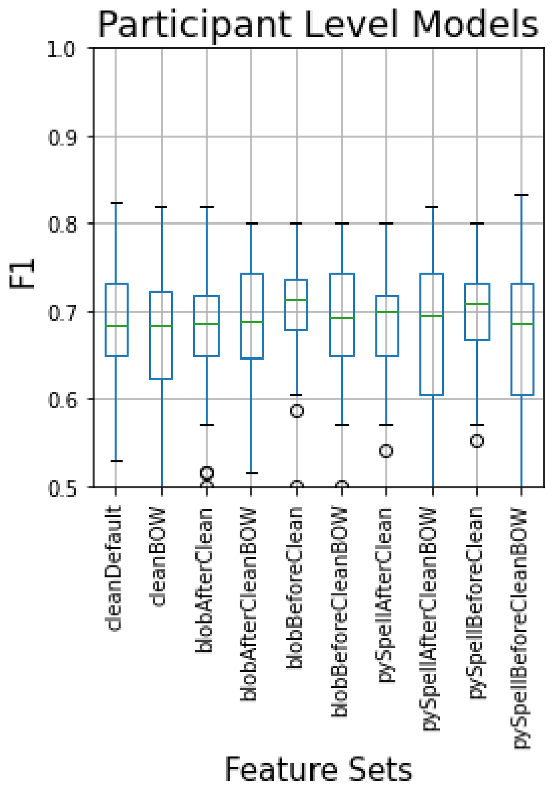


Figure 21: Best Models for Predicting PHQ-9 Scores on Participant-Level Data For PCA

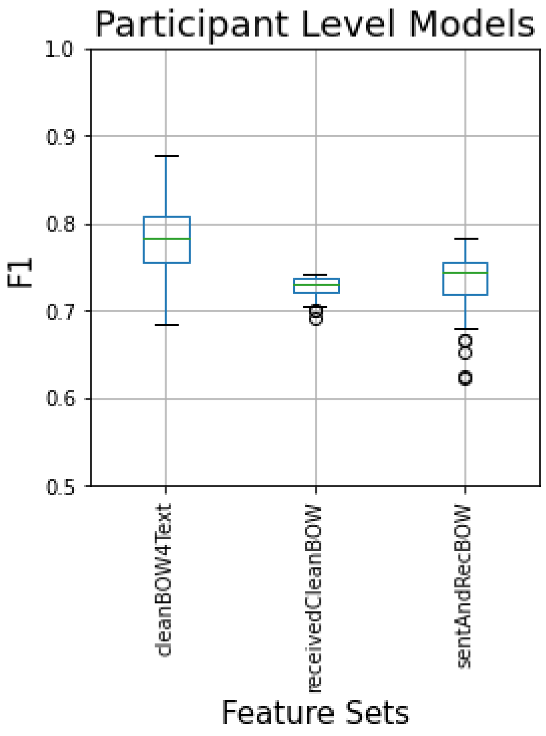


Figure 22: Best Models for Predicting PHQ-9 Scores on Participant-Level Data For Sent, Received and Combination of Those Datasets

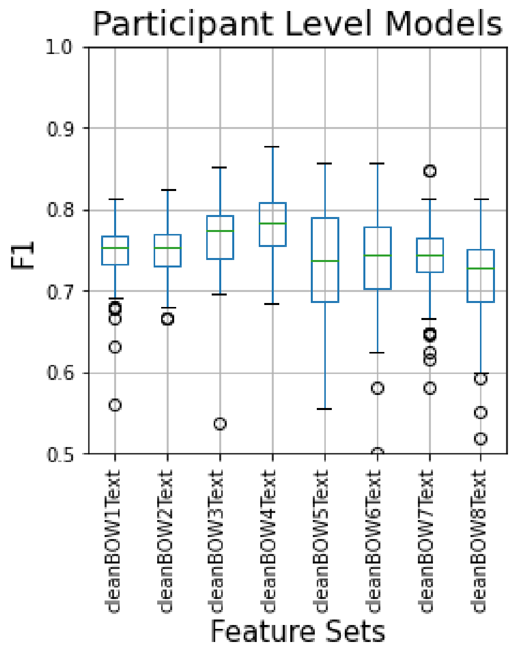


Figure 23: Best Models for Predicting PHQ-9 Scores on Participant-Level Data For Minimum Text Messages

4.2 BERT

The results from our work with the BERT model includes increasing code efficiency, running an experiment with emojis, and optimizing the hyperparameters.

Increasing Code Efficiency

Upon initially starting this project, we looked over the code we would be working with from the REU Team. After going through the code thoroughly to increase efficiency, we came across many areas where code could be reduced. As a result, the initial code was reduced from 535 to 455 lines. This decrease was despite writing comments throughout and adding whitespace for readability. Moreover, we documented instructions about how to run the code, wrote a shell script that first time users can execute to install all necessary technical requirements, and added system arguments so that experiments can be run more seamlessly. All of these steps resulted in cleaner, more legible code so that future collaborators will be able to effectively understand and work with the code.

Emoji Experiment

Although the emoji experiment was initially promising based on the research, ultimately incorporating emojis had no effect on the accuracy. The table below shows the accuracy of the base model with the emojis incorporated and without the emojis incorporated.

With Emojis and Without Emojis	
0.725	0.8
0.775	0.7
0.625	0.55
0.6	0.725
0.7	0.775

Table 3: Emoji Experiment Results

The following table displays the average accuracy and standard deviation for each method as well.

	With Emojis	Without Emojis
Average	0.685	0.71
STDEV	0.06442	0.087464
AVG+STDEV	0.74942	0.797464
AVG-STDEV	0.62058	0.622536

Table 4: Averages and Standard Deviations of Emoji Experiment Results

Hyperparameters Experiment

As discussed in the methodology, we conducted experiments to optimize four different hyperparameters - the batch size, number of folds for cross validation, number of epochs, and dropout rate. The optimal value for each of those parameters is displayed below. These are the parameters that were used for all succeeding experiments.

Hyperparameter	# Folds	Batch Size	Epochs	Dropout Rate
Optimal Value	5	16	15	0.1

Table 5: Hyperparameter Experiment Results

We experimented with batch sizes of 12, 16, 20, 25, and 32, and the results are outlined below. A batch size of 16 yielded the most accurate results, so that will be the standard moving forward.

Batch Size	12	16	20	25	32
Mean Accuracy	0.689	0.786	0.701	0.645	0.642

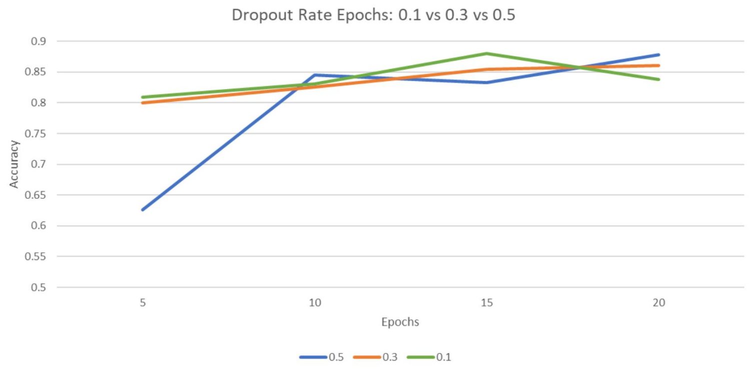
Table 6: Batch Size Experiment Results

After experimenting with 3, 4, and 5 folds, we decided to implement 5-fold cross validation based on the results below.

Folds	3	4	5
Mean Accuracy	0.705	0.747	0.755

Table 7: Folds Experiment Results

The dropout rates and epochs were tested together since the two parameters can impact each other. Based on the results below, we decided that 15 epochs with a dropout rate of 0.1 is optimal.



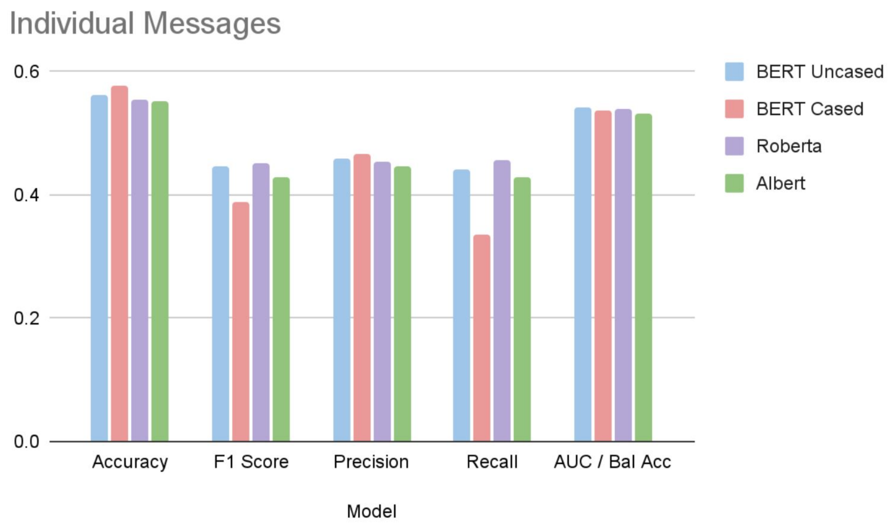
4.2.1 BERT Models and Data Input Types

As discussed in the methodology, we applied various data input types to the optimized BERT code to determine which preprocessing methods yield the best results. These data input types include individual text messages, chunks based on number of words, chunks created by sliding over messages, chunks created by sliding over chunks of characters, and conversation-level data. For each of these inputs, four different BERT models were experimented with - Base Cased, Base Uncased, ALBERT, and RoBERTa.

For each combination of models and data input types, we found the accuracy, F1 score, precision, recall, and AUC/balanced accuracy. AUC and balanced accuracy are the same for binary predictors like our model (https://arxiv.org/pdf/2103.11357.pdf), so for the sake of simplicity the results will refer to both with the title "AUC/Bal Acc." The tables below display each of these statistics for each of the combinations.

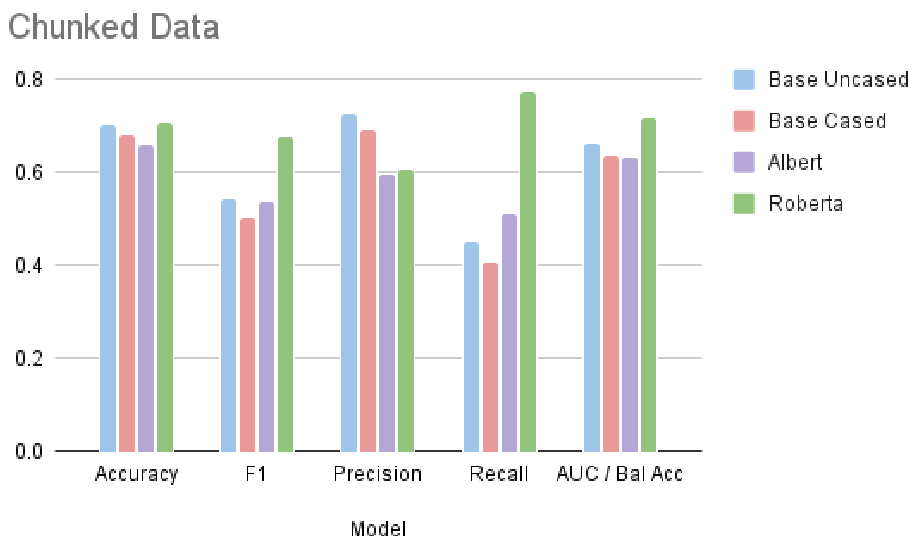
Individual Text Messages

The results of running the BERT code on each of the four BERT models on the individual messages data are displayed below. As you can see, the BERT Cased model has the highest accuracy but a small margin, but the lowest recall, or the ability to correctly identify true positives.

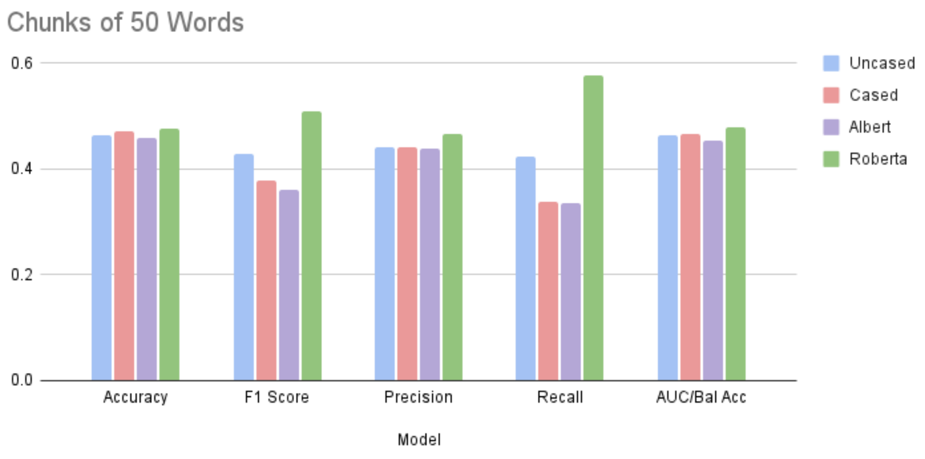


Chunks By Number of Words

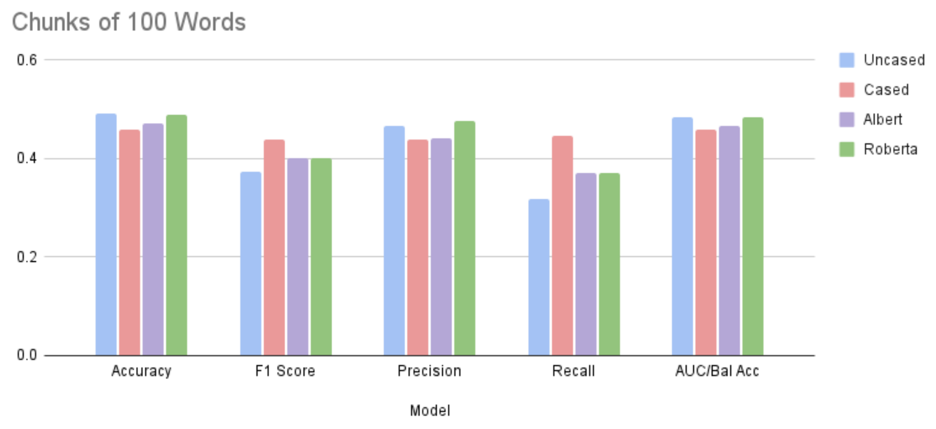
The graph below depicts the results that we found when running the BERT code with the original 200 word chunks. These are the experiments we had to rerun with the newly developed chunked datasets after we created the train and test splits.



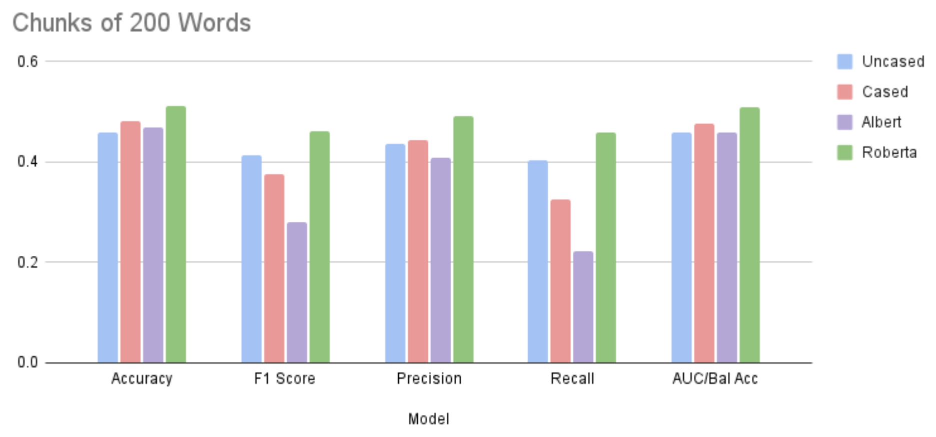
The following graph shows the results when we run the BERT models with chunks of 50 words. The RoBERTa model performs the best, with the highest value for each of the statistics. One notable area it particularly does well in is recall.



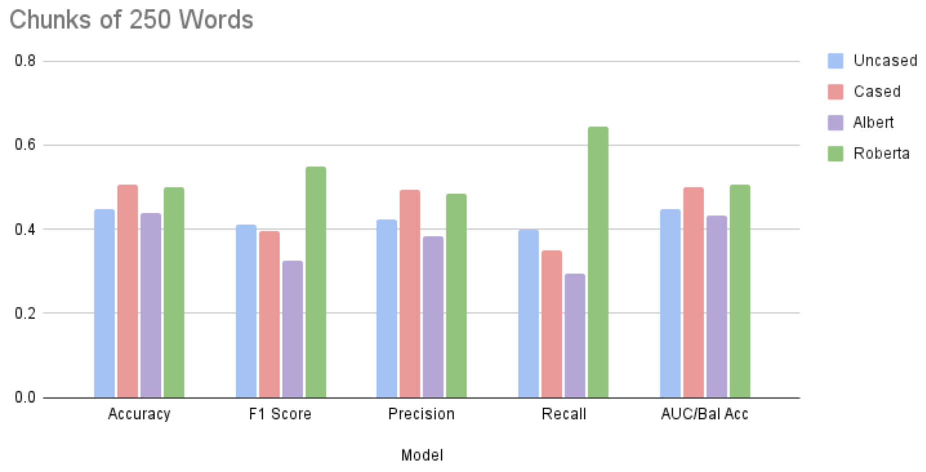
This graph displays the results for the chunks of 100 words. For this experiment, the BERT Uncased model has the highest accuracy and the highest balanced accuracy. Another model that performs well is the RoBERTa model, which has accuracies that are not far behind those of the BERT Uncased model.



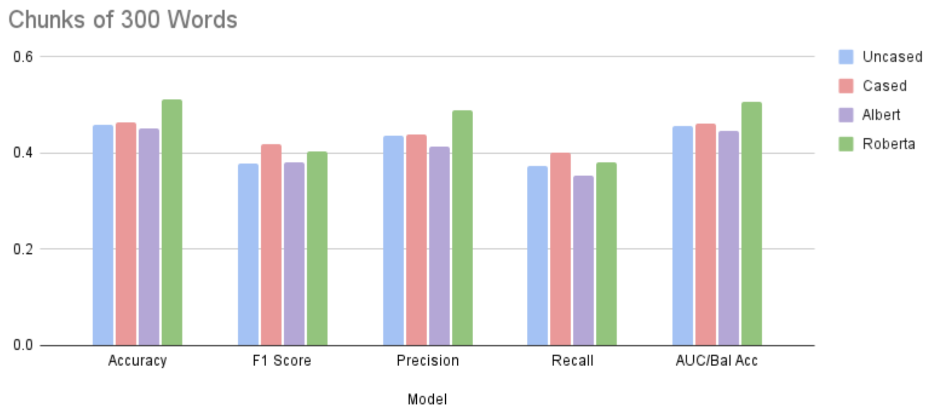
For the new 200 word chunks that were created with the new train and test splits, the RoBERTa model performs the best for all of the statistics. The ALBERT model performs poorly, with an F1 score and recall that are completely dwarfed by the RoBERTa model.



The RoBERTa model once again has a very high recall when it comes to the 250 word chunks. In terms of accuracy and balanced accuracy, however, the BERT Cased model performs equal or better when compared to the RoBERTa model.

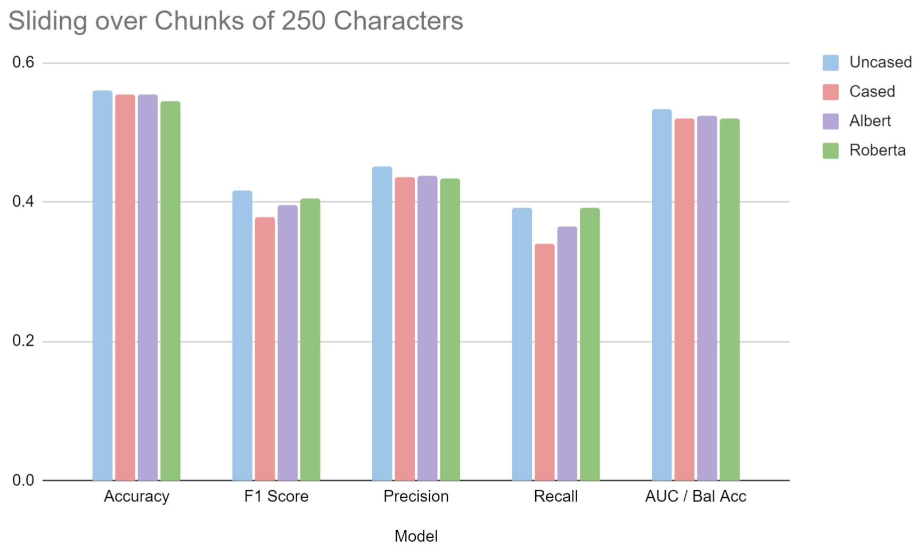


Similarly to the previous results, the 300 word chunks produce results that also see the RoBERTa model perform the best in terms of accuracy and balanced accuracy. Notably, it is beaten by BERT Cased when it comes to both F1 score and recall.

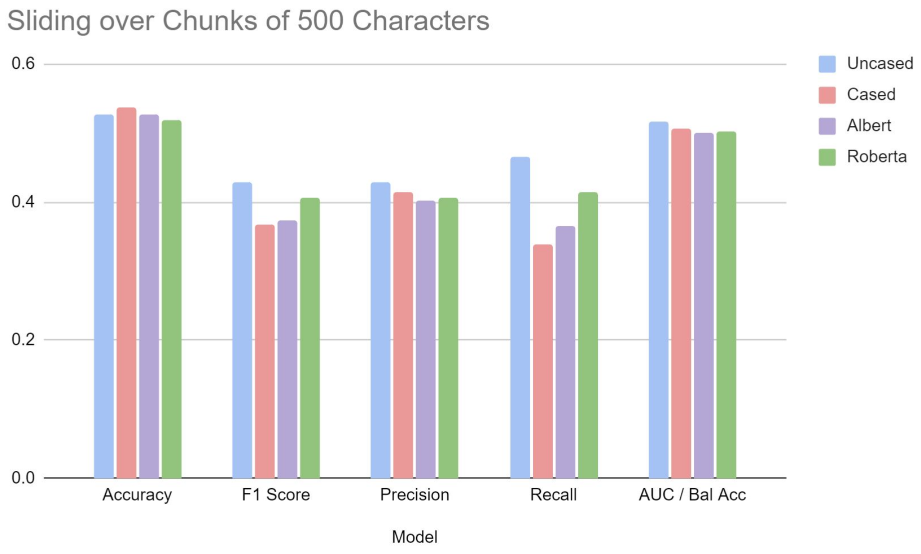


Chunks Created by Sliding over Chunks of Characters

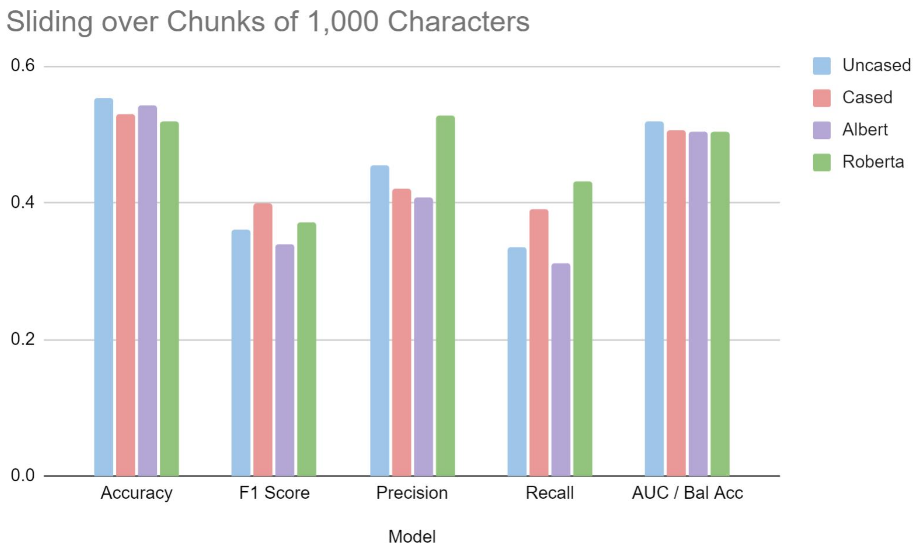
Below are the results of running each of the four BERT models with data generated by sliding over chunks of 250 characters. Once again, the BERT Uncased model slightly outperforms the other models, but has a far lower recall. This likely implies that the model can identify participants that do not have depression more accurately than participants that do have depression.



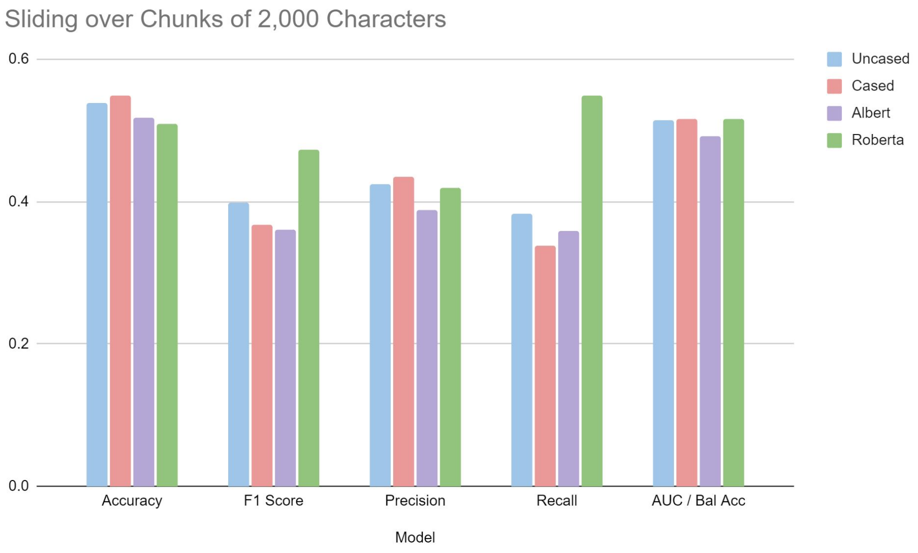
Each of the four BERT models were also run with data generated by sliding over chunks of 500 characters. The results below imply that the BERT Cased Model has once again outperformed other models in accuracy but underperformed in recall.



The graph below illustrates the results of running each BERT model with data generated by sliding over chunks of 1,000 characters. BERT Uncased proves to have the highest accuracy and AUC, yet the results vary for the F1 score, precision, and recall.

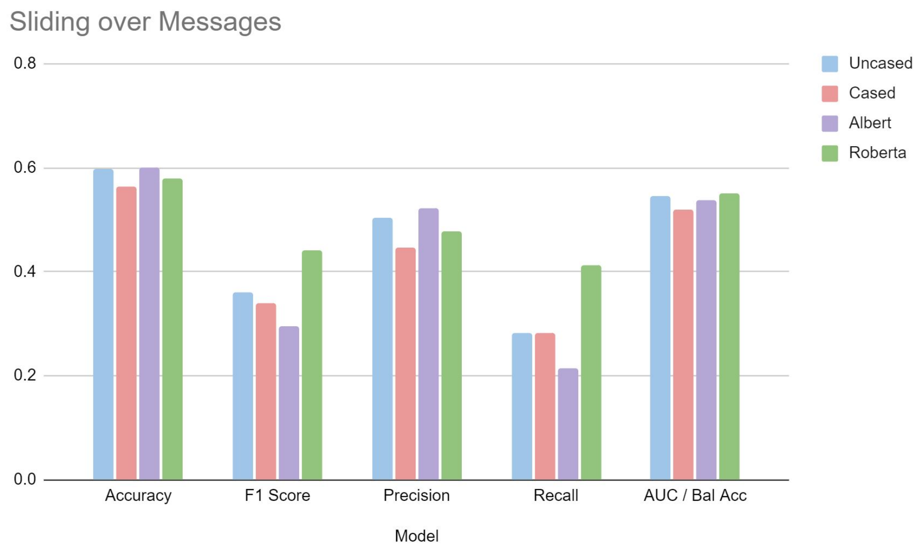


For the results of running each BERT model with data generated by sliding over chunks of 2,000 characters, BERT Cased again exhibits the highest accuracy yet the lowest recall. Roberta performs exceedingly well in terms of recall and F1 score.



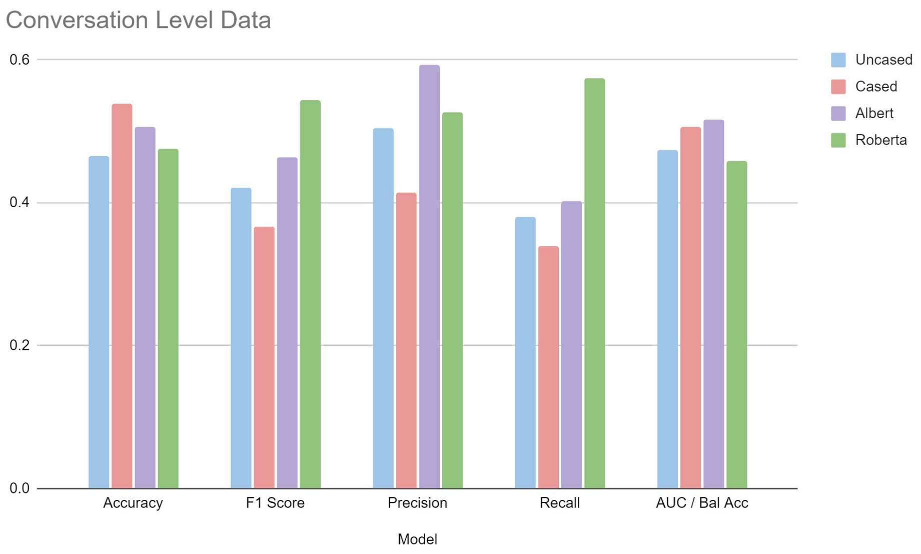
Chunks Created by Sliding over Messages

Applying all four BERT models to the data generated by sliding over five messages, it is found that BERT Uncased and Albert have the highest accuracy, and BERT Uncased outperforms Albert with most other performance measures.



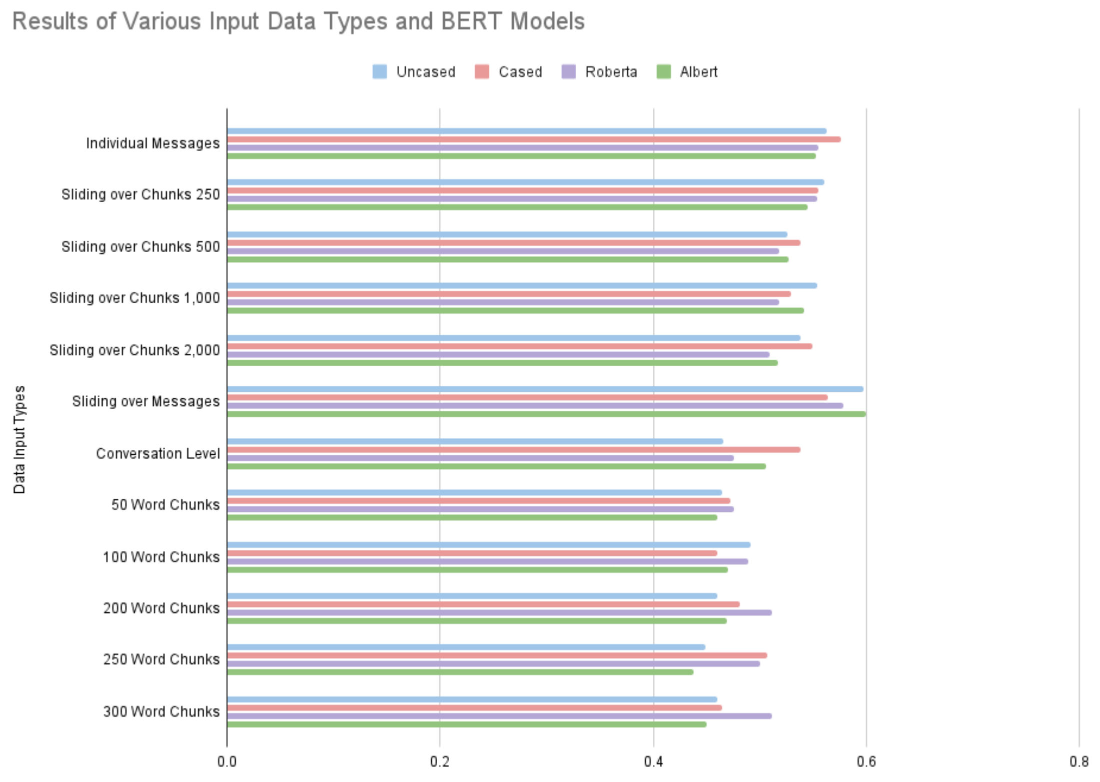
Conversation Level Data

For the conversation level data, there is another case of Albert having the highest accuracy yet the lowest recall. Albert has the second highest accuracy and significantly outperforms the other models in precision.



Total Data Input Types

The graph below displays every previously-mentioned resulting accuracy from the various data input types. Overall, running the sliding over messages data with the ALBERT model has the highest accuracy, with a 60.011% test accuracy for predicting whether a participant has depression or not. This result is closely followed by the BERT Uncased model with the same data, which scores a test accuracy of 59.740%.



4.3 Lexical Categories Visualization

Gathering Inspiration

The work put into building the Pinterest board was able to inspire new ideas and perspectives on visualizing the lexical categories, and the team was able to gain a better understanding of what factors make a data visualization intuitive, memorable, and beautiful. A snapshot of the board can be seen in Figure 24.

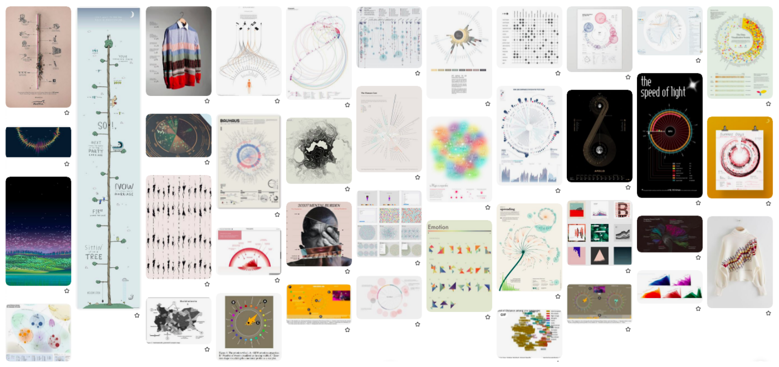


Figure 24: Snapshot of the lexical categories visualization Pinterest board.

4.3.1 Planning and Drafting

After creating the Pinterest board, possible data visualizations were drawn with pencil and paper. The team was inspired by elements found in nature, since nature has long been known to have positive benefits in improving mental health [61]. The goal of the visualizations was to visualize the output from the lexical categories machine learning pipeline. The output is the estimated value for each lexical category based on the given PHQ-9 score. In the visualizations, the PHQ-9 score is shown along with corresponding lexical categories, which are sized proportional to the value predicted by the machine learning algorithm. Rough sketches for the ideas can be found in Figures 25 and 26, and digitized versions of the sketches can be found in Figures 27 through 31.

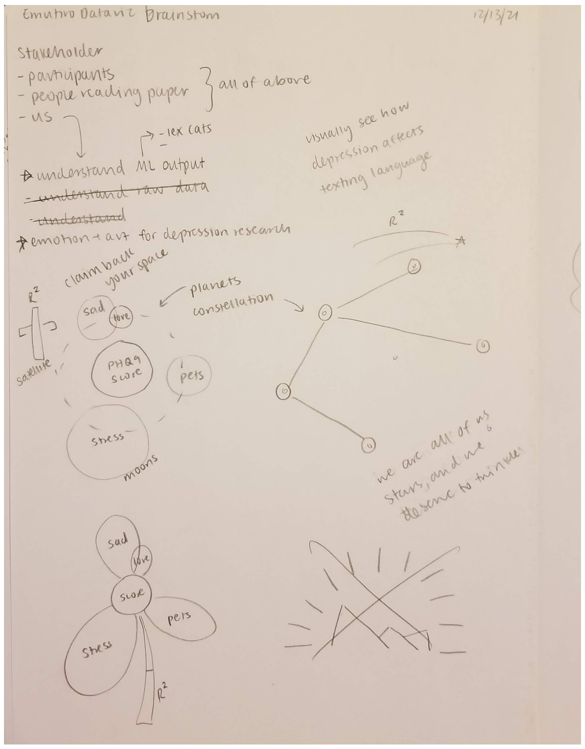


Figure 25: Brainstormed sketches

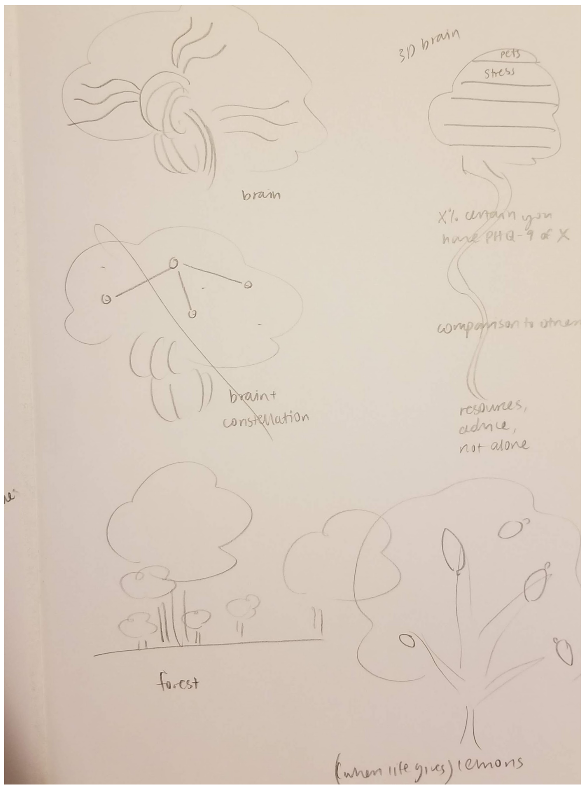


Figure 26: Brainstormed sketches part 2

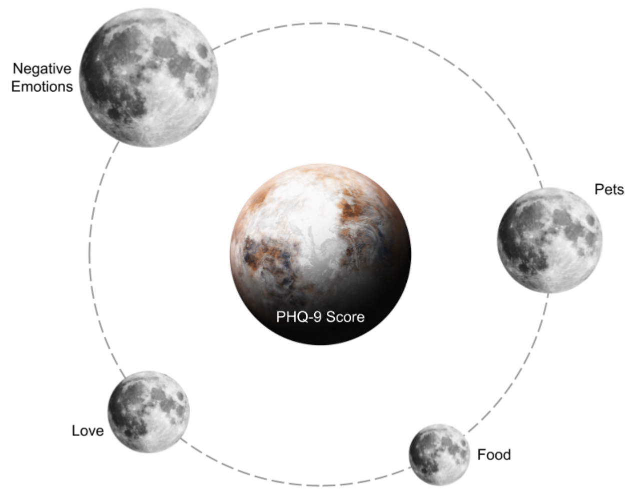


Figure 27: Visualization inspired by planets

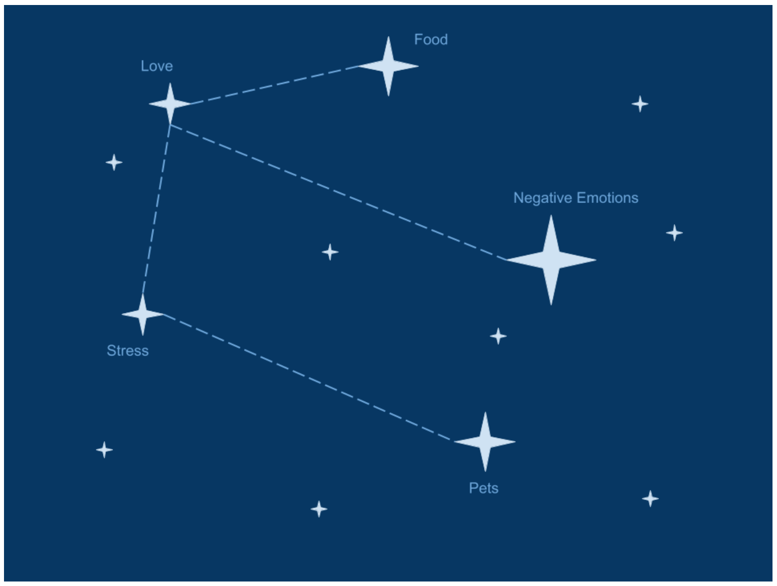


Figure 28: Visualization inspired by constellations

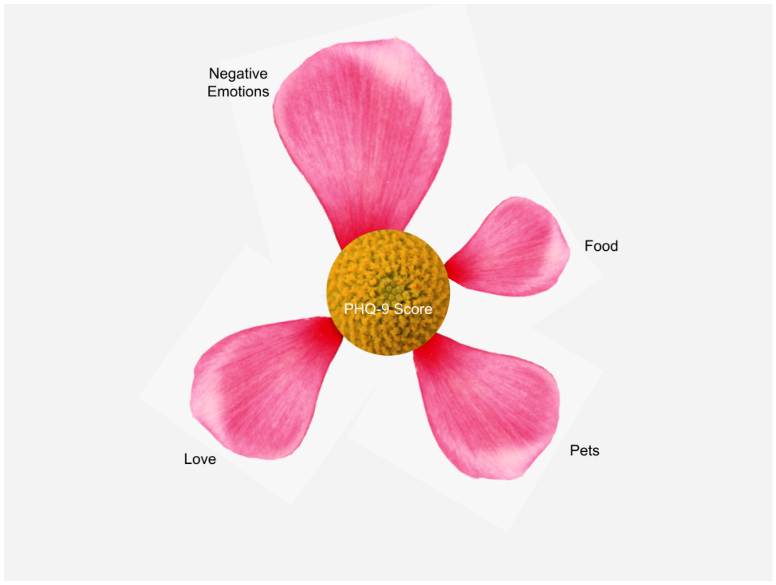


Figure 29: Visualization inspired by flowers

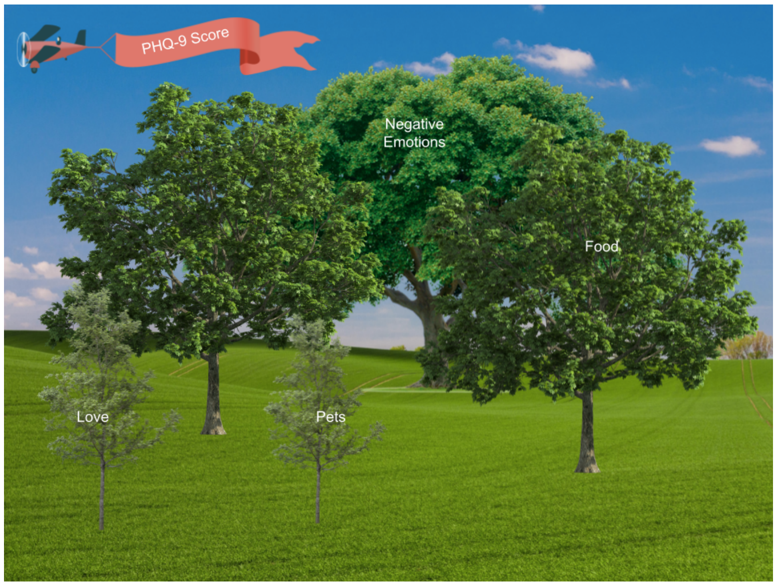


Figure 30: Visualization inspired by forests

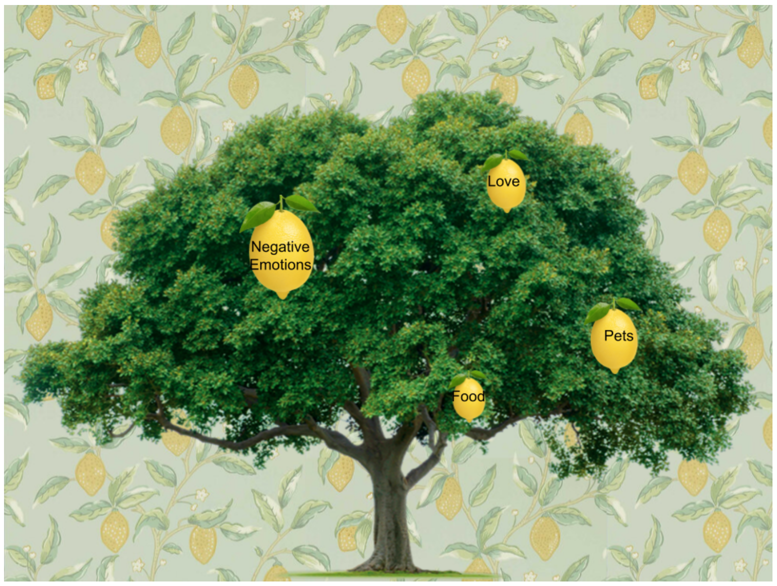


Figure 31: Visualization inspired by lemon trees

In addition, the team designed possible logos relating to the lexical categories aspect of this project. These logos may be used to brand the data visualizations in the future so that credit can be traced back to the Emutivo project. Three variations were created and can be seen in Figure 32.

Figure 32: Possible logo designs for the lexical categories subteam.

4.3.2 Creating the Visualizations

Andrea Mignone kindly gave us permission to reuse his flower plots, which were found on the data visualization platform Observable. The following visualizations were created and colored by PHQ-9 category.

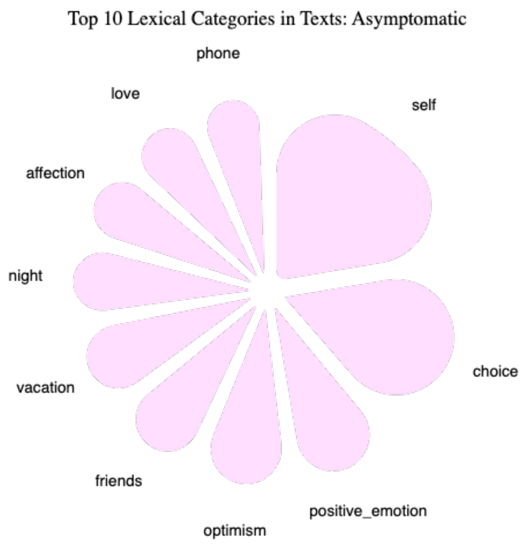


Figure 33: Top 10 Lexical Categories in Texts: Asymptomatic

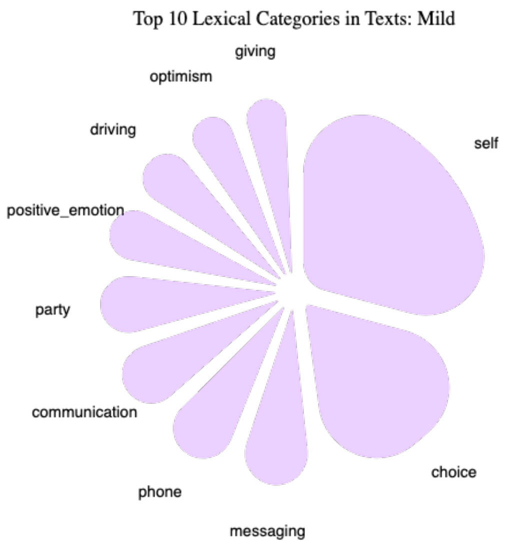


Figure 34: Top 10 Lexical Categories in Texts: Mild

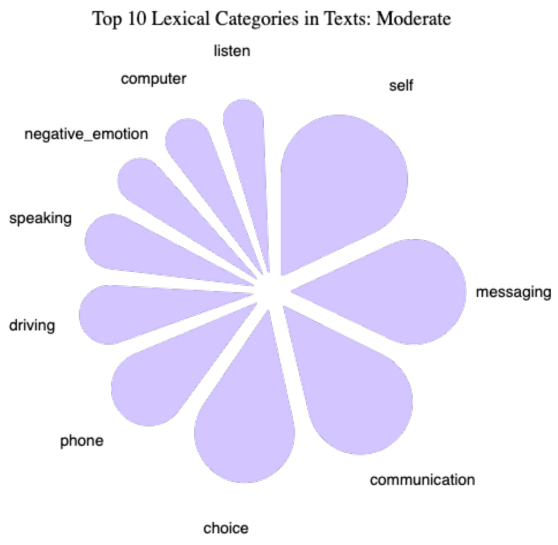


Figure 35: Top 10 Lexical Categories in Texts: Moderate

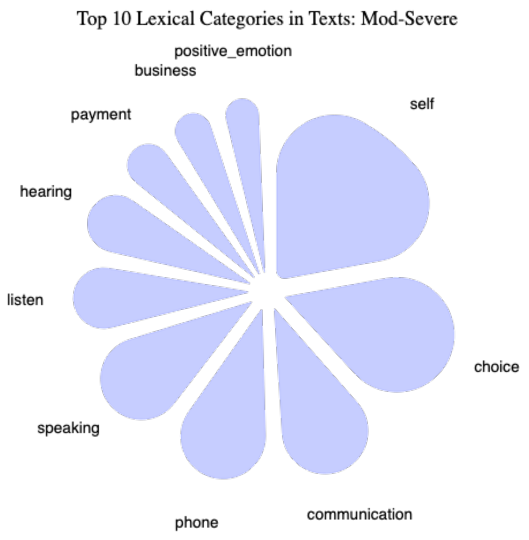


Figure 36: Top 10 Lexical Categories in Texts: Moderate-Severe

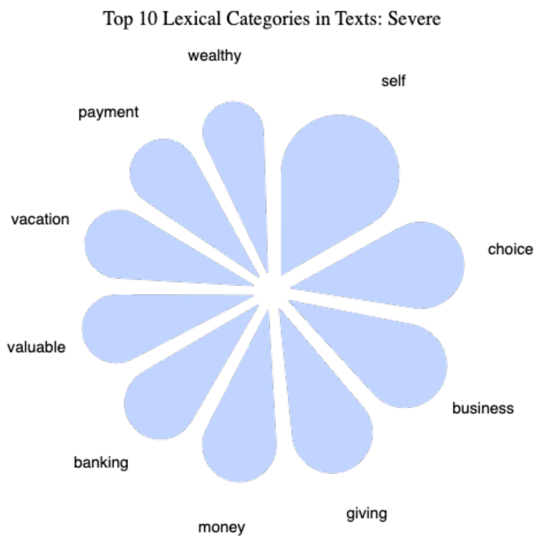


Figure 37: Top 10 Lexical Categories in Texts: Severe

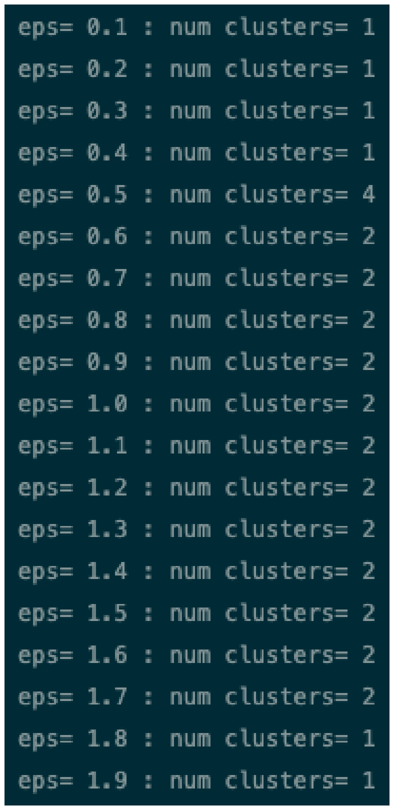
From the visualizations, it can be determined that asymptomatic and mild patients tend to talk about more light-hearted and personal topics, as evidenced by the presence of the "friends", "positive_emotions", "affection", "love", and "optimism" categories. The "night" category is present only for the asymptomatic category, perhaps because those patients keep talking to loved ones at night. Mild, moderate, and moderately-severe patients talk a lot about methods of communication such as "communication", "messaging", "transportation", "speaking", "listen", and "hearing". Moderately-severe and severe patients tend to talk about impersonal categories like money and work – "banking", "money", "business", "payment", and "wealthy".

4.3.3 Creating Super-Categories

We experimented with creating super-categories with machine learning as well as manually. We used spaCy to compute similarity scores between each pair of Empath category names and created a distance matrix using those scores. We then used two clustering algorithms to create clusters: density-based spatial clustering of applications with noise (DBSCAN), the most common density-clustering algorithm, and hierarchical clustering. We then compared the results of using machine learning to clusters created manually.

DBSCAN

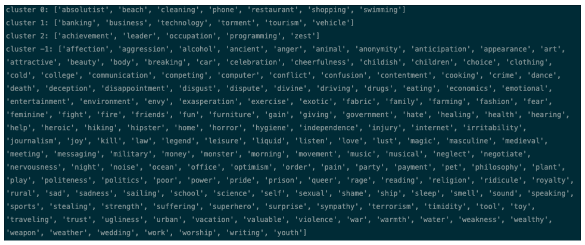
The scikit-learn DBSCAN package includes two important parameters: eps, the maximum distance between two samples in one neighborhood, and min_samples, the minimum number of samples in a neighborhood. When skimming the Empath category names, we hypothesized that most super-category clusters would be at least 5, so we chose a min_samples value of 5. We ran several experiments to determine the eps value that would result in the largest number of clusters, which turned out to be eps=0.4 and a cluster size of 4. The results of the experiment are shown in Figure 38



```
eps= 0.1 : num clusters= 1
eps= 0.2 : num clusters= 1
eps= 0.3 : num clusters= 1
eps= 0.4 : num clusters= 1
eps= 0.5 : num clusters= 4
eps= 0.6 : num clusters= 2
eps= 0.7 : num clusters= 2
eps= 0.8 : num clusters= 2
eps= 0.9 : num clusters= 2
eps= 1.0 : num clusters= 2
eps= 1.1 : num clusters= 2
eps= 1.2 : num clusters= 2
eps= 1.3 : num clusters= 2
eps= 1.4 : num clusters= 2
eps= 1.5 : num clusters= 2
eps= 1.6 : num clusters= 2
eps= 1.7 : num clusters= 2
eps= 1.8 : num clusters= 1
eps= 1.9 : num clusters= 1
```

Figure 38: Number of Clusters for Different DBSCAN eps Values

When analyzing the results of running DBSCAN with min_samples=5 and eps=0.4, we realized that the algorithm gave 3 small clusters and 1 large primary cluster with more than half of the Empath categories, as seen in Figure 39. Thus, we moved on to experimenting with hierarchical clustering.



```
cluster 0: ['absolutist', 'beach', 'cleaning', 'phone', 'restaurant', 'shopping', 'swimming']
cluster 1: ['banking', 'business', 'technology', 'torment', 'tourism', 'vehicle']
cluster 2: ['achievement', 'leader', 'occupation', 'programming', 'zest']
cluster -1: ['affection', 'aggression', 'alcohol', 'ancient', 'anger', 'animal', 'anonymity', 'anticipation', 'appearance', 'art',
'attractive', 'beauty', 'body', 'breaking', 'car', 'celebration', 'cheerfulness', 'childish', 'children', 'choice', 'clothing',
'cold', 'college', 'communication', 'competing', 'computer', 'conflict', 'confusion', 'contentment', 'cooking', 'crime', 'dance',
'death', 'deception', 'disappointment', 'disgust', 'dispute', 'divine', 'driving', 'drugs', 'eating', 'economics', 'emotional',
'entertainment', 'environment', 'envy', 'exasperation', 'exercise', 'exotic', 'fabric', 'family', 'farming', 'fashion', 'fear',
'feminine', 'fight', 'fire', 'friends', 'fun', 'furniture', 'gain', 'giving', 'government', 'hate', 'healing', 'health', 'hearing',
'help', 'heroic', 'hiking', 'hipster', 'home', 'horror', 'hygiene', 'independence', 'injury', 'internet', 'irritability',
'journalism', 'joy', 'kill', 'law', 'legend', 'leisure', 'liquid', 'listen', 'love', 'lust', 'magic', 'masculine', 'medieval',
'meeting', 'messaging', 'military', 'money', 'monster', 'morning', 'movement', 'music', 'musical', 'neglect', 'negotiate',
'nervousness', 'night', 'noise', 'ocean', 'office', 'optimism', 'order', 'pain', 'party', 'payment', 'pet', 'philosophy', 'plant',
'play', 'politeness', 'politics', 'poor', 'power', 'pride', 'prison', 'queer', 'rage', 'reading', 'religion', 'ridicule', 'royalty',
'rural', 'sad', 'sadness', 'sailing', 'school', 'science', 'self', 'sexual', 'shame', 'ship', 'sleep', 'smell', 'sound', 'speaking',
'sports', 'stealing', 'strength', 'suffering', 'superhero', 'surprise', 'sympathy', 'terrorism', 'timidity', 'tool', 'toy',
'traveling', 'trust', 'ugliness', 'urban', 'vacation', 'valuable', 'violence', 'war', 'warmth', 'water', 'weakness', 'wealthy',
'weapon', 'weather', 'wedding', 'work', 'worship', 'writing', 'youth']
```

Figure 39: Resulting Clusters Using DBSCAN

Hierarchical Clustering

We tested three linkage techniques: average linkage, centroid linkage, and Ward linkage (Figures 40, 41, and 42). Ward linkage resulted in the most balanced dendrogram so we used that linkage technique for further experimentation. We first directed the SciPy hierarchical clustering package to create clusters no larger than 20 words – the results are shown in Figure 43.

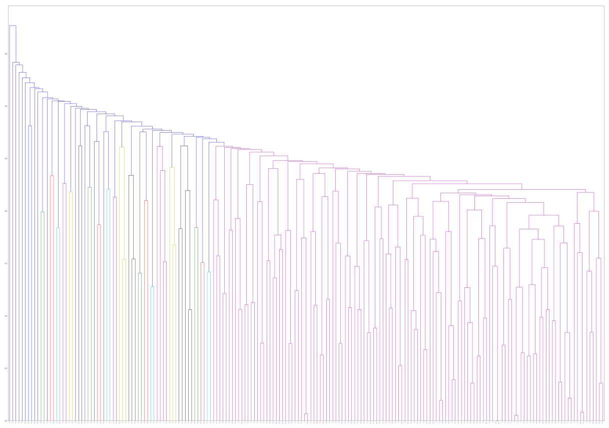


Figure 40: Dendrogram Using Average Linkage

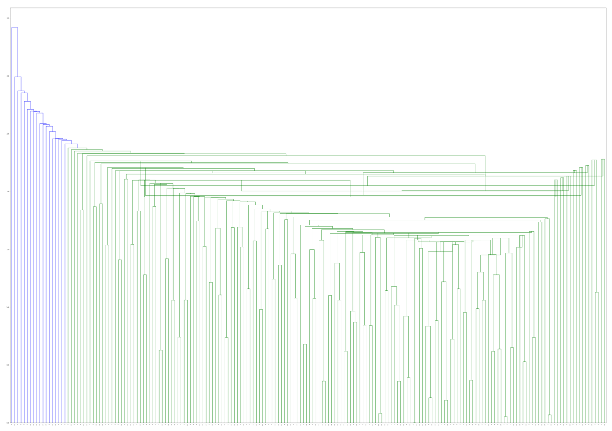


Figure 41: Dendrogram Using Centroid Linkage

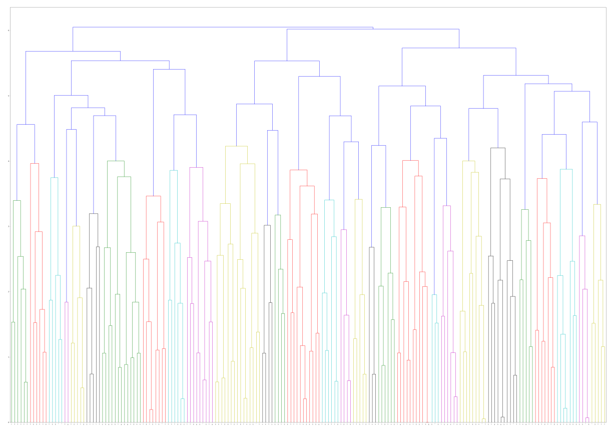
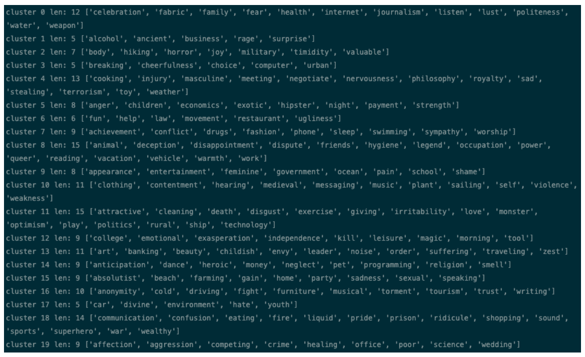


Figure 42: Dendrogram Using Ward Linkage



```
cluster 0 len: 12 ['celebration', 'fabric', 'family', 'fear', 'health', 'internet', 'journalism', 'listen', 'lust', 'politeness',
'water', 'weapon']
cluster 1 len: 5 ['alcohol', 'ancient', 'business', 'rage', 'surprise']
cluster 2 len: 7 ['body', 'hiking', 'horror', 'joy', 'military', 'timidity', 'valuable']
cluster 3 len: 5 ['breaking', 'cheerfulness', 'choice', 'computer', 'urban']
cluster 4 len: 13 ['cooking', 'injury', 'masculine', 'meeting', 'negotiate', 'nervousness', 'philosophy', 'royalty', 'sad',
'stealing', 'terrorism', 'toy', 'weather']
cluster 5 len: 8 ['anger', 'children', 'economics', 'exotic', 'hipster', 'night', 'payment', 'strength']
cluster 6 len: 6 ['fun', 'help', 'law', 'movement', 'restaurant', 'ugliness']
cluster 7 len: 9 ['achievement', 'conflict', 'drugs', 'fashion', 'phone', 'sleep', 'swimming', 'sympathy', 'worship']
cluster 8 len: 15 ['animal', 'deception', 'disappointment', 'dispute', 'friends', 'hygiene', 'legend', 'occupation', 'power',
'queer', 'reading', 'vacation', 'vehicle', 'warmth', 'work']
cluster 9 len: 8 ['appearance', 'entertainment', 'feminine', 'government', 'ocean', 'pain', 'school', 'shame']
cluster 10 len: 11 ['clothing', 'contentment', 'hearing', 'medieval', 'messaging', 'music', 'plant', 'sailing', 'self', 'violence',
'weakness']
cluster 11 len: 15 ['attractive', 'cleaning', 'death', 'disgust', 'exercise', 'giving', 'irritability', 'love', 'monster',
'optimism', 'play', 'politics', 'rural', 'ship', 'technology']
cluster 12 len: 9 ['college', 'emotional', 'exasperation', 'independence', 'kill', 'leisure', 'magic', 'morning', 'tool']
cluster 13 len: 11 ['art', 'banking', 'beauty', 'childish', 'envy', 'leader', 'noise', 'order', 'suffering', 'traveling', 'zest']
cluster 14 len: 9 ['anticipation', 'dance', 'heroic', 'money', 'neglect', 'pet', 'programming', 'religion', 'smell']
cluster 15 len: 9 ['absolutist', 'beach', 'farming', 'gain', 'home', 'party', 'sadness', 'sexual', 'speaking']
cluster 16 len: 10 ['anonymity', 'cold', 'driving', 'fight', 'furniture', 'musical', 'torment', 'tourism', 'trust', 'writing']
cluster 17 len: 5 ['car', 'divine', 'environment', 'hate', 'youth']
cluster 18 len: 14 ['communication', 'confusion', 'eating', 'fire', 'liquid', 'pride', 'prison', 'ridicule', 'shopping', 'sound',
'sports', 'superhero', 'war', 'wealthy']
cluster 19 len: 9 ['affection', 'aggression', 'competing', 'crime', 'healing', 'office', 'poor', 'science', 'wedding']
```

Figure 43: Resulting Clusters Using Hierarchical Clustering

Manual Clustering

Clustering manually in draw.io resulted in 15 clusters, which can be seen in Table 4.3.3. An overview of the clusters in draw.io can also be seen in Figure 44.

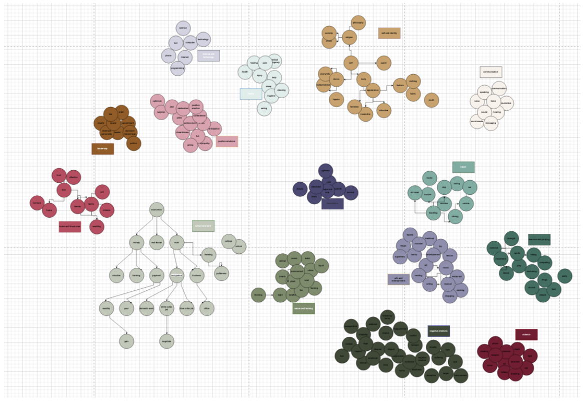


Figure 44: Overview of Manual Clustering

Science and technology	['science', 'tool', 'technology', 'internet', 'computer', 'phone', 'programming']
Health	['health', 'healing', 'pain', 'medical emergency', 'injury', 'help', 'sleep', 'hygiene', 'cleaning', 'eating']
Positive emotions	['optimism', 'surprise', 'zest', 'celebration', 'pride', 'positive emotion', 'contentment', 'achievement', 'joy', 'anticipation', 'fun', 'cheerfulness', 'giving', 'sympathy', 'trust', 'affection', 'love']
Negative emotions	['exasperation', 'childish', 'fear', 'neglect', 'ridicule', 'emotional', 'rage', 'irritability', 'sadness', 'shame', 'dispute', 'swearing terms', 'disappointment', 'disgust', 'aggression', 'envy', 'horror', 'confusion', 'weakness', 'negative emotion', 'torment', 'conflict', 'timidity', 'suffering', 'hate', 'sad', 'anger', 'deception', 'nervousness']
Violence	['prison', 'stealing', 'weapon', 'violence', 'crime', 'kill', 'death', 'terrorism', 'breaking', 'military', 'war', 'fight']
Self and identity	['philosophy', 'religion', 'worship', 'divine', 'self', 'queer', 'body', 'anonymity', 'choice', 'independence', 'hipster', 'appearance', 'feminine', 'masculine', 'attractive', 'fashion', 'clothing', 'fabric', 'youth']
Communication	['communication', 'speaking', 'noise', 'listen', 'journalism', 'hearing', 'sound', 'social media', 'messaging']
Travel	['exotic', 'tourism', 'air travel', 'traveling', 'vacation', 'ship', 'sailing', 'car', 'vehicle', 'driving']
Descriptions	['ugliness', 'cold', 'warmth', 'ancient', 'shape and size', 'absolutist', 'smell', 'beauty']
Arts and entertainment	['legend', 'magic', 'monster', 'heroic', 'superhero', 'medieval', 'toy', 'entertainment', 'leisure', 'art', 'music', 'reading', 'writing', 'musical', 'restaurant', 'cooking', 'shopping']
Exercise and partying	['strength', 'exercise', 'movement', 'sports', 'play', 'dance', 'hiking', 'swimming', 'competing', 'drugs', 'lust', 'party', 'sexual', 'alcohol']
Nature and environment	['animal', 'ocean', 'water', 'liquid', 'urban', 'environment', 'beach', 'plant', 'rural', 'fire', 'weather', 'night', 'morning']
School and work	['economics', 'money', 'real estate', 'work', 'meeting', 'politeness', 'valuable', 'banking', 'payment', 'occupation', 'business', 'wealthy', 'poor', 'domestic work', 'white collar job', 'blue collar job', 'office', 'gain', 'negotiate', 'college','farming','school']
Home and loved ones	['furniture', 'home', 'friends', 'family', 'pet', 'children', 'wedding']
Leadership	['law', 'order', 'royalty', 'power', 'government', 'dominant personality', 'leader', 'dominant heirarchical', 'politics']

Resulting Clusters Using Manual Clustering

4.3.4 Color Coding Using Empath Super-Categories

After creating the super-categories, we altered the flower plots so that each petal displayed the color that corresponded to its super-category. The super-categories are also roughly ordered from most personal to least personal: lighter colors are more personal topics like "self and identity" and darker colors are more impersonal topics like "science and technology". The visualizations are shown in Figures 45, 45, 46, 47, 48, and 49.

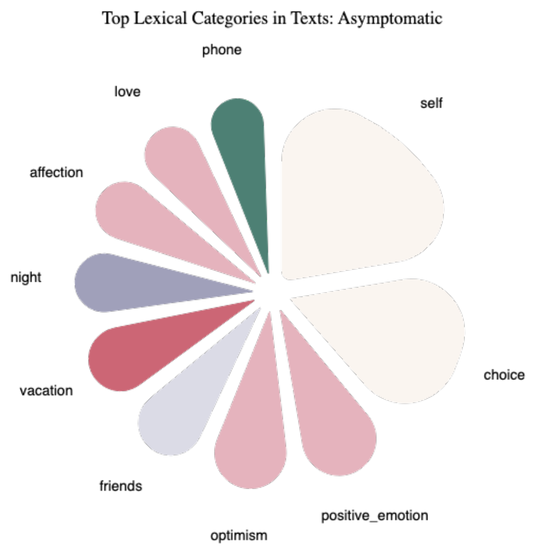


Figure 45: Top 10 Lexical Categories in Texts: Asymptomatic (Color-Coded)

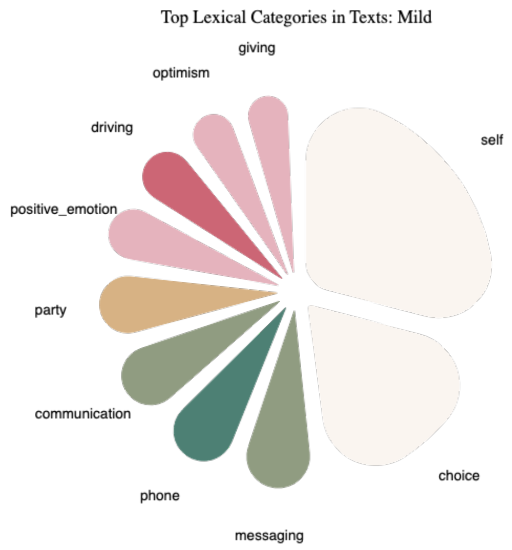


Figure 46: Top 10 Lexical Categories in Texts: Mild (Color-Coded)

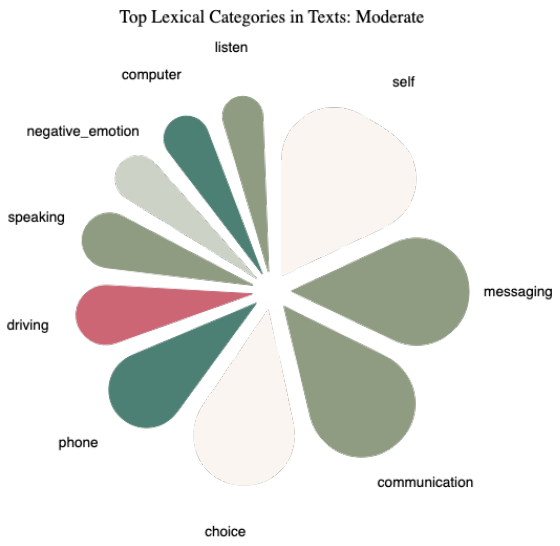


Figure 47: Top 10 Lexical Categories in Texts: Moderate (Color-Coded)

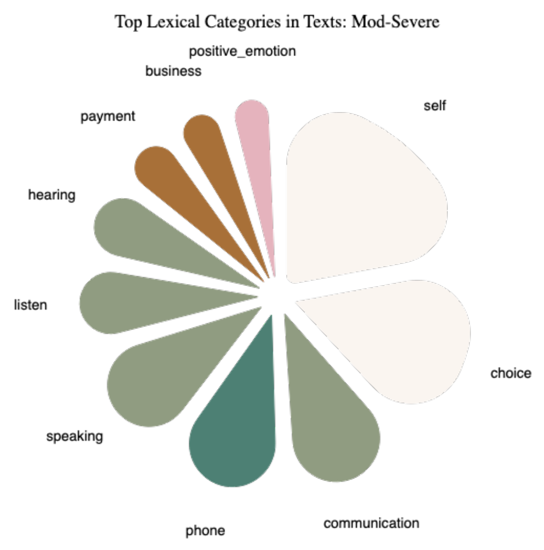


Figure 48: Top 10 Lexical Categories in Texts: Moderate-Severe (Color-Coded)

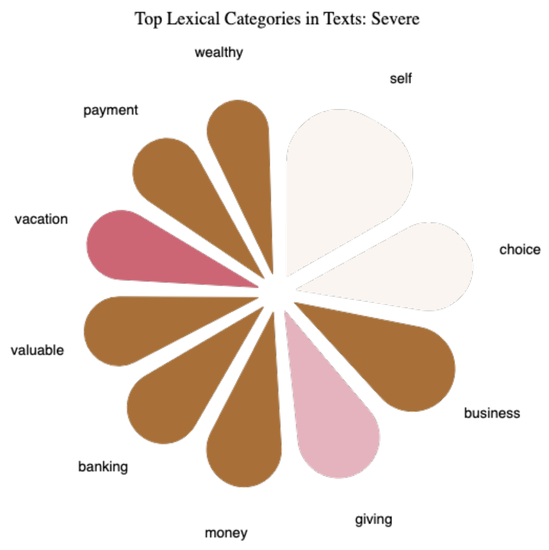


Figure 49: Top 10 Lexical Categories in Texts: Severe (Color-Coded)

From the updated visualizations, it can be determined that asymptomatic and mild patients tend to talk about more light-hearted and personal topics, as evidenced by the lighter colors and presence of the pink “positive emotions” category. Mild, moderate, and moderately-severe patients talk a lot about methods of communication as seen by the presence of the dark green “communication” category. Moderately-severe and severe patients tend to talk about impersonal categories like money and work – as seen by the dark brown “school and work” category. The shift from lighter colors to darker colors is also more prevalent.

4.4 Beiwe

4.4.1 Storing The Data Collected By The Frontend

Our team worked hard debugging both the iOS and Android frontend code. We could not integrate the frontend with REDCap; however, we were able to successfully compile the Android code and integrate it with the backend code. In terms of the iOS code, we worked on debugging the code but ran into an error that can only be fixed by the authors of the code. This put a halt to working with the iOS code so we then began to focus on getting the Android code to compile successfully.

iOS

When our team began working with the iOS frontend, we started by going through Github and understanding the structures. The repository for iOS had no readme file and was not well documented. When trying to get the application up and running on our personal devices we ran into many issues. Within the repository we were able to use some of the posts in the issues section to solve errors along with Google searches. The team learned early on that we needed to document our steps so that if another team wanted to run Beiwe, they would be able to set it up more easily. All the information we gathered went into a tutorial document which can be found in Appendix B.

We spent most of our time debugging the iOS frontend code to have the ability to run locally on our own devices without having to use the application from the app store. As we began the debugging process, we updated all of the packages used in the project and updated our computers operating system in order to ensure the versions we were using would work. Our team also encountered challenges with integrating the files properly with the project. The Beiwe project used cocoapods to import code from various Github repositories. We quickly learned that we needed to update versions of the repositories and change how some variables were declared in the project. Our debugging process and solutions for the integration problems can be found in Appendix B.

Once the cocoapods were integrated properly with the project, we were presented with a new set of issues. We found that the Hakuba package which was imported via the cocoapods was outdated and would not integrate with the project. In order to fix this error we downgraded our version of Xcode and ran various pod commands to try and integrate the package. This error was a constant throughout our debugging process and there were many times when we thought we had fixed it but we were mistaken. This process finally fixed the error and it did not come up again as we continued to debug.

We also had errors involving objective-c headers and not being able to find headers within the scope of the project. These were easily fixed with a google search and found solutions through Stack Overflow and the Apple developer website, these fixes can be found in Appendix B. The final error we were left with was "'CellModel' is ambiguous for type lookup in this context." When working on this error it was very hard to find any specific information on how to solve it. The only constant in my research was that our team needed to be more specific when declaring the variable. We decided to reach out on the Github repository to get help from others. When we reached out we were contacted by one of the authors of the repository and they informed our team that they would fix the issue.

Android

Similar to how we started with the iOS frontend, we did the same with the Android frontend. The Android Github repository had significantly more documentation and direction than the iOS one provided. We began downloading the code and using Android Studio to run our version of the Beiwe Android frontend. As we began trying to get the frontend application up and running the app automatically compiled with no errors, but when we went to build we were met with a multitude of errors.

The first set of errors we worked on were updating versions within the SDK files. Some of them were out of date since they were uploaded. We then went through and updated the Android Manifest along with making sure the firebase was set up properly. There were instructions in the Github on how to set up a firebase so we made sure to go back through each step to make sure we set it up properly. A package that was implemented within FirebaseInstanceId was deprecated meaning that it was out of date and was no longer compatible with the program. There was an updated version of the package so our team went through and updated the program. Once this was done we ran into some new errors.

We had immense trouble with the Android Studio APK/Bundles. In order to run a clean build of the program we needed to generate a signed APK. The generated APK went hand in hand with a file called keystore.properties which held the username and password. When we read through the Github instructions it led us to believe that we needed to create this file ourselves. By creating this file ourselves we cause a multitude of errors. As we began looking for solutions to this issue we learned that this file was automatically generated when the signed APK was made. Once we figured this out we deleted the keystore.properties file that our team created and re-made the signed APK. After the signed APK was made the keystore.properties file was automatically generated and the error was resolved.

We were able to have a successful build of the Android app once these errors were resolved. We documented all of our steps in Appendix C so that if others ran into similar problems they would be able to solve them easier.

Testing the Viability of Using REDCap

Unfortunately, the team could not successfully find where the frontend app stored data locally or sent data to an S3 bucket, in both the iOS and Android versions. This meant that we could not determine how to re-route the data collected by the frontend to REDCap instead of AWS.

4.4.2 Running The Backend on a VM

To test the viability of Beiwe to run without using AWS services, one solution was to run the backend completely locally. A virtual machine (VM) was set up to serve the backend, which allowed the team to immediately be able to create new studies with the Beiwe study management portal. While undergoing research, two different VMs were set up:

- A local Ubuntu VM; and
- A development VM for the Emutivo website.

The team tested the integration of the frontend with the backend and determined that using a VM instead of AWS would not work.

Implementation on Ubuntu VM

Initially the team tried running the Beiwe backend on a local MacOS host. However, this was quickly determined to be an ineffective solution since the Beiwe backend is designed to run on an Ubuntu OS. Thus, an Ubuntu VM was set up on the host MacOS machine and the Beiwe backend code was downloaded onto the VM. The team was able to run the Flask app included in the backend code to host the study management portal on a local port. Once the portal was opened, the team was able to log in, create new studies, researchers, and participants. Documentation on implementing the backend on the Ubuntu VM can be found in Appendix D.

To test whether the studies created by the Ubuntu local server could be used in a real study, the team attempted to log into the Beiwe2 apps from the Apple Store and the Google Play Store using information generated by the backend for a given study. However, the Beiwe2 app required a URL for the server. Since the study management portal was run on a local port in the Ubuntu server, a persistent URL did not exist and it was not possible to log into the app. To solve this issue, a development VM was set up for the Emutivo team's home website.

Implementation on Emutivo's Development VM

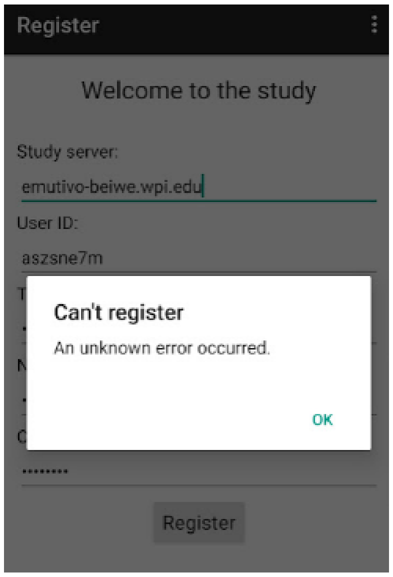
The solution to the inability to log into the Beiwe2 app was to host the backend on a persistent URL. Initially, the team thought to incorporate the study portal directly on the Emutivo team's home website, which is hosted on Wordpress (emutivo.wpi.edu). Dr. Ermal Toto, the Assistant Director of Academic and Research Computing at WPI, proposed a better solution: create a development VM that acts as a mirror version of the Emutivo home website (emutivo-dev.wpi.edu).

Thus, a development VM was set up and the team was able to upload the Beiwe backend code to the VM using Cyberduck, an open-source client that is used for transferring files over the internet. The backend was reconfigured to run on the emutivo-dev.wpi.edu server and was set up by following the same steps as those outlined in the previous section.

However, the team ran into an issue. The backend is composed of a Flask app, which needs both a server and a port to be specified in order to run properly. Thus, the study portal was hosted on emutivo-dev.wpi.edu:8080 instead of the desired emutivo-dev.wpi.edu. Dr. Toto helped us solve this issue by setting up Flask as a service on the Ubuntu-based development VM. Running Flask as a service allowed the Beiwe Flask app to remain uninterrupted so that progress wouldn't be lost. Additionally, he helped us certify a new website URL with a public IP address – emutivo-beiwe.wpi.edu – to signify the change to a production server, run the service on a URL that doesn't include a port number, and make the URL accessible outside of the WPI wifi network. Documentation on setting up the backend on the Emutivo development VM can be found in Appendix E.

Testing With Frontend Apps

After successfully setting up the Beiwe backend on the development VM, we began testing integration with the frontend. We discovered that the "Beiwe2" app from the app store expected to connect to the studies.beiwe.org server and would fail to connect to any custom server URLs. Instead of using Beiwe2, we used our version of the Android open-source code and paired the code on Android Studio with a personal Android device for testing. After modifying the frontend to connect to the emutivo-beiwe.wpi.edu study server, we began to test if we could log into a study created by the backend. However, the team could not register and the app showed the error message in the figure below.


Register

Welcome to the study

Study server:

emutivo-beiwe.wpi.edu

User ID:

aszsne7m

Can't register

An unknown error occurred.

Figure 50: Error message on frontend app

After tracing the error back to the code, the team discovered that a 500 error was hard-coded for any error that was not known, as seen in Table 8. By looking at the other error messages, the team knew that the frontend could connect to the emutivo-beiwe.wpi.edu study server, the login username and password were correct, and the device was connected to the internet. We determined that the backend error is likely caused by an AWS EC2 error – the Android frontend is expecting to connect to a Beiwe AMI but detects the WPI Emutivo development VM instead. Ultimately, the team discovered that the Beiwe is too integrated with AWS for the platform to run properly without those services. Documentation on this error can be found in Appendix E.

Error Code	Error Message
1	Someone misconfigured the server! Please contact research staff.
2	Received an invalid encryption key, please contact the Beiwe administrator.
400	This device could not be registered under the provided patient ID; please contact research staff.
405	Sorry, this User ID already has a phone registered. Please contact the research staff and ask them to clear your registration so that you can register this new phone.
502	It looks like you're not connected to the internet- please connect to the internet and try again.
502 and 404	Are you sure the URL for the study is correct? Beiwe can't connect to the server.
403	Sorry: either your Study URL, your User ID, or your Temporary Password is incorrect.
403	Sorry, your Current Password is incorrect.
403	Sorry, your Temporary Password is incorrect.
500	An unknown error occurred.

Table 8: Error messages in the frontend

4.4.3 Making AWS Services HIPAA-Compliant

In order to run Beiwe on AWS services, the team first made an AWS Educate account, which offers up to $100 in credits for WPI students [60]. Through our research on how to make the EC2 and S3 services HIPAA-compliant, we discovered that though AWS is naturally HIPAA-compliant, configuration mistakes can leave protected health information (PHI) open to unauthorized groups and therefore no longer HIPAA-compliant.

To solve this problem, AWS released their Quick Start program, from which you can choose pre-configured AWS services that align with HIPAA practices and use those services for your applications. The team did not have enough time to fully set up the Beiwe platform with the credits from AWS Educate and use the Quick Start program, so we created a tutorial for future teams (Appendix F).

5 Conclusions

5.1 Lexical Categories

Evaluating F1 Prediction accuracy

Grouping the data by conversation and leaving all of the text entry content as is had the highest prediction accuracy when using the train/test split used by the BERT team. Grouping them by ID after grouping by conversation ultimately was too restrictive on the train/test split with this specific data set, as there were only 88 unique ID's.

The Bag of Words Experiments also tested PCA against Chi squared as the machine learning model. Chi squared was way better than PCA as the F1 scores were always higher for the Chi squared than PCA when looking at the same feature set.

Spell Checker

Three different spell checkers were run on the data to try and fix typos, language and abbreviations to make the data formatted Empath to read. The results showed no increase in F1 score for either pyEnchant, Textblob or pyspellchecker. This is likely because neither of the spell checkers make any relevant fixes to words that would be indicative of depression.

Minimum Text Messages

When running the best results so far being, Bag of Words, Chi squared, the sent messages dataset, and no spell checker, we tested the different amounts of minimum text messages. The results showed an increase until four minimum text messages and then decrease after four, meaning that the best amount of minimum text messages was four. This makes sense because if the minimum is too low you get too much noise and if the minimum is too high you lose too much data.

Future Work

The experiments in this project can be built upon in many ways, especially in regards to the set of sent and received messages. The new Empath code could be run with different numbers of messages considered per conversation similar to the Bag of Words experiments run. Also, if further data collection is done, a larger set of participants would increase the diversity in the train/test splits.

5.2 BERT

Individual Messages VS Chunks

When comparing the individual message data to the chunked data, we found that the individual message data generally had slightly higher accuracies than the chunked data. The only exception to this statement was the chunks of messages that were created using the sliding window. These findings show that dividing by messages may produce better results than dividing by characters or words.

No Overlap VS Overlap

We found that the chunks that were generated using the sliding window, either based on the number of characters or number of messages, did noticably better than the chunks that contained no overlap. The overlapping chunks had accuracies that were around 5-10 percent higher than the non-overlapping chunks. From these findings, we can determine that using a sliding window will likely lead to better BERT results.

Sliding Over Messages vs Sliding Over Characters

The five message chunks that we generated performed slightly better than the chunks that were generated using a specific number of characters, with the closest results being between the "chunkedDataFrame250" and "chunkedBy5Messages" datasets. These two datasets had accuracies that were only around 1 to 2 percent apart. This discovery makes sense because we found that our messages have an average of 54 characters in them. This statistic shows that five messages and 250 characters are roughly equivalent for our dataset. It also shows that smaller chunks may be better at producing more accurate BERT results. For comparison, the 2000 character chunks performed about 2 percent worse than the 250 character chunks when it came to test accuracy.

Conversation

Our findings showed that the conversation-level data performed worse than the datasets of overlapping chunks of data, but it performed better than the non-overlapping chunks. This finding shows that using both sent and received texts could allow BERT to produce better results.

Future Work Suggestions

Future work should look into whether it could be possible to combine conversation-level data with the sliding window. Additionally, a sliding window could be used to create more overlapping chunks based on a varying number of messages. Finally, new data collected through Beiwe could be used to further train and test the models.

5.3 Lexical Categories Visualization

The lexical categories visualizations showed that there are differences between topics commonly talked about among mild and asymptomatic participants, compared to the most common topics among moderate-severe and severe participants. Those who had lower PHQ-9 scores talked much more about personal super-categories like "self and identity" and "positive emotions". Those who had higher PHQ-9 scores talked much more about "school and work". Future work includes continuing work on visualizing super-categories and testing their efficacy.

Alternative flower visualizations in which each petal is an Empath super-category instead of an Empath category could be created. This would allow a more intuitive understanding of which super-categories are more prevalent in different PHQ-9 categories.

Also, in order to ensure that the data visualizations are intuitive, memorable, and effective, we will survey a group of people to gather feedback. Then, we will implement any changes that need to be made, and repeat the process.

5.4 Beiwe

The team focused mainly on research – whether it be for implementing new technologies, methods for pre-processing data, or the best hyperparameters to use for a model. After about three and a half months of exploring the dataset, different technologies, and learning about what other research teams have done in the past, next steps are to apply our learnings by running experiments with different models on our cleaned datasets. These experiments include running various classification algorithms on the dataset with features created by Empath and running BERT with various aggregation and pre-processing methods. In addition, we will experiment with setting up the Beiwe backend Flask app with a custom URL.

5.4.1 Front-End

By working with the frontend code, our team would hopefully have the ability to design our own studies with the capability to collect a wide variety of data that we could customize. We were unable to run sample studies with Beiwe but our hope is that another team will be able to customize a study and collect more data over a long period of time. Having the ability to edit the code freely allows the director of a study to pick and choose what kind of data they would like to have collected. The ease of using Beiwe we hope will allow future groups to have an easier time customizing their studies and collect a variety of different data.

iOS

The iOS frontend code has potential to run a study that could be designed to collect almost any data that the directors of the study want. Our team was unable to get a sample up and running due to a problem with the code that needs to be fixed by the owners of the Github repository. Once this error is fixed, our hope is that another team will be able to customize a sample study to get more diverse data.

Android

Our team was able to get the Android code up and running and connect it to the backend. We were unable to run a sample study due to some errors in the connection to the backend. We hope that another team will be able to take the information we have documented and run a sample study using AWS and then eventually reconfigure the code to work on the WPI server.

5.4.2 Back-End

Using a development VM to run the Beiwe backend was determined to not be ideal. Future teams will have to use AWS services to run the platform and may sign up for either AWS Educate – which offers $100 to students and $200 to faculty – or AWS Cloud Credit for Research, through which faculty and staff can receive any amount of credits they put in their research proposal to AWS [60]. Finally, the Quick Start program must be used to ensure that the AWS resources used are HIPAA-compliant.

6 Appendix A: Analysis of Githubs

Beiwe Android Github

Link to Github: https://github.com/onnela-lab/beiwe-android

Overview: Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe Android app code. The Beiwe2 app is also available on the Google Play store to use with open source builds of the Beiwe backend.

What is this Repository?
This repository is the front end of the Android Beiwe application.

Compiling

1. To compile and sign the app, you must add a directory called private (beiwe-android's .gitignore file keeps the private directory out of Git), and a file called private/keystore.properties, containing these lines(no quotes around the values that you fill in):

```
storePassword=KEYSTORE_PASSWORD
keyPassword=KEY_PASSWORD
keyAlias=KEY_ALIAS
storeFile=KEYSTORE_FILEPATH
```

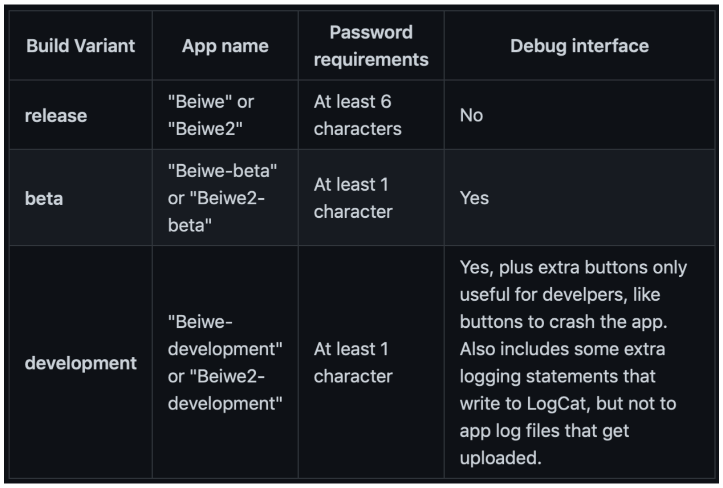
2. If it's your first time running the project, open android Studio and click "Sync Project with Gradle Files". If you run into errors, open Android Studio's "SDK Manager" and make sure you have the correct SDK Platform installed (the "API level" should match the one specified in app/build.gradle's compileSdkVersion).
3. You'll need to generate a key file b y running Build -> Generate Signed Bundle/APK, and then, inside private/keystore.properties, update the four values with the information for your newly-generated key and keystore.
4. (Optional) You can also configure a Sentry DSN for each build type inside your private/keystore.properties file.

```
releaseDSN=https://publicKey:secretKey@host:port/1?options
betaDSN=https://publicKey:secretKey@host:port/1?options
developmentDSN=https://publicKey:secretKey@host:port/1?options
```

Build Variants and Product Flavors

When building the Android app, we can choose one of the Build Variants and one of the Product Flavors. There are three of each, specified in the buildTypes section of app/build.gradle. To select which Build Variant the app compiles as, go to Build > Select Build Variant in the menu bar (documentation).

Three Build Variants:



Build Variant	App name	Password requirements	Debug interface
release	"Beiwe" or "Beiwe2"	At least 6 characters	No
beta	"Beiwe-beta" or "Beiwe2-beta"	At least 1 character	Yes
development	"Beiwe-development" or "Beiwe2-development"	At least 1 character	Yes, plus extra buttons only useful for develpers, like buttons to crash the app. Also includes some extra logging statements that write to LogCat, but not to app log files that get uploaded.

Three Product Flavors:

Product Flavor	App Name	Intended for	Server URL	Record text message and call log stats	Request background location permission
googlePlayStore	Beiwe2	Distribution via Google Play Store	Customizable at registration	No	No
onnelaLabServer	Beiwe	Download APK from studies.beiwe.org /download	Hardcoded to studies.beiwe.org	Yes	Yes
commStatesCustomUrl	Beiwe	Download APK from /download link on other Beiwe deployments	Customizable at registration	Yes	Yes

Beiwe Backend Github

Link to Github: https://github.com/onnela-lab/beiwe-backend

Overview: Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe backend code.

What is this Repository?
This repository is home to the backend code of the Beiwe application.

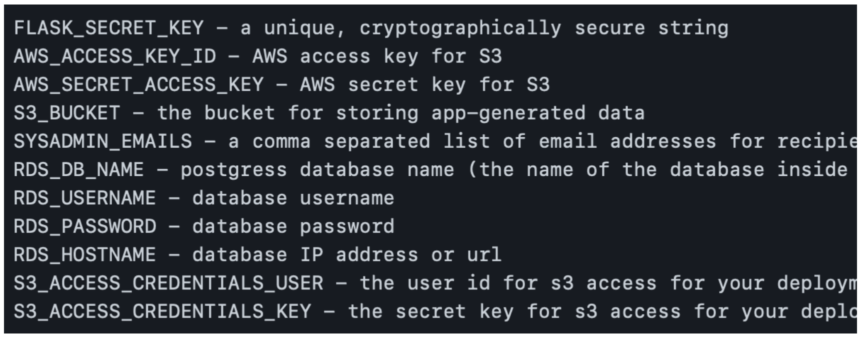
Setup Instructions

Configuring SSL
Beiwe deals with sensitive data that is covered under HIPAA, so we need to make sure to set up SSL certificates so that the web traffic is encrypted with HTTPS.

The instructions in this Github uses AWS Certificate Manager to generate an SSL certificate. AWS checks that we control the domain by sending verification emails to the email addresses in the domain's WHOIS listing.

Configuring Firebase
To configure the firebase SDK we generate a private key file and place it in the project root.

Configuration Settings
Empty strings and None are considered invalid

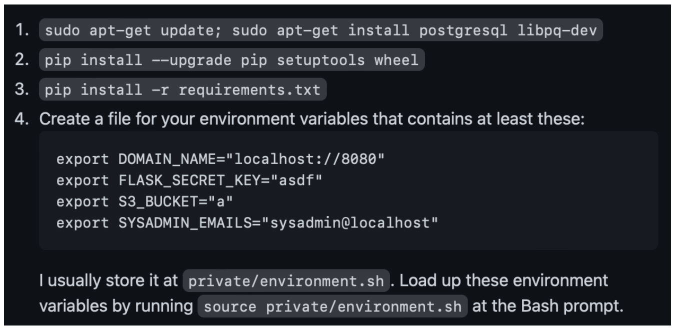


```
FLASK_SECRET_KEY - a unique, cryptographically secure string
AWS_ACCESS_KEY_ID - AWS access key for S3
AWS_SECRET_ACCESS_KEY - AWS secret key for S3
S3_BUCKET - the bucket for storing app-generated data
SYSADMIN_EMAILS - a comma separated list of email addresses for recipie
RDS_DB_NAME - postgress database name (the name of the database inside
RDS_USERNAME - database username
RDS_PASSWORD - database password
RDS_HOSTNAME - database IP address or url
S3_ACCESS_CREDENTIALS_USER - the user id for s3 access for your deploym
S3_ACCESS_CREDENTIALS_KEY - the secret key for s3 access for your deplo
```

Development setup
Set-up for development NOT production

- Strongly discouraged to set up the development environment inside of a Python system

  Before starting:



1. `sudo apt-get update; sudo apt-get install postgresql libpq-dev`
2. `pip install --upgrade pip setuptools wheel`
3. `pip install -r requirements.txt`
4. Create a file for your environment variables that contains at least these:

```
export DOMAIN_NAME="localhost://8080"
export FLASK_SECRET_KEY="asdf"
export S3_BUCKET="a"
export SYSADMIN_EMAILS="sysadmin@localhost"
```

   I usually store it at `private/environment.sh`. Load up these environment variables by running `source private/environment.sh` at the Bash prompt.

Local Celery Setup

Don’t need to use Celery for local testing

Beiwe IOS Github

Link to Github: https://github.com/onnela-lab/beiwe-ios

Overview: Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe iOS app code. The Beiwe2 app is also available on the Apple app store to use with open source builds of the Beiwe backend.

Initial Setup

There is no Readme, in another document there will be an explanation of the initial setup and the steps taken.

General Notes

How does the front end connect to the back end?

7 Appendix B: Running the Beiwe IOS Front-end

Running the Beiwe IOS Front-end

Madeline Halley (mehalley@wpi.edu) and Jyalu Wu (jwu7@wpi.edu)

Setting up Xcode and downloading the Beiwe code

We will be downloading Beiwe for IOS, so you will need to be using an apple computer or mac operating system.

Xcode

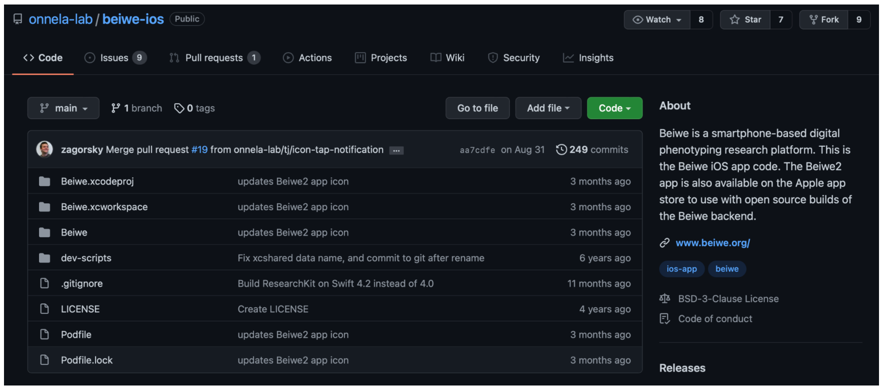
Can download Xcode from the apple app store, make sure it is updated to the most recent version.

Figure 1: Screenshot of Xcode on the Apple App Store

Download Beiwe Code

Beiwe IOS front end code can be found on Github.
Link: https://github.com/onnela-lab/beiwe-ios

In order to download the code you will need to access the Git Repository. Once you have opened the Repository, click on the green button that reads **Code**.



onnela-lab / beiwe-ios Public

Watch 8 · Star 7 · Fork 9

Code · Issues 9 · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

main · 1 branch · 0 tags · Go to file · Add file · Code

zagorsky Merge pull request #19 from onnela-lab/tj/icon-tap-notification ... aa7cdfe on Aug 31 · 249 commits

Beiwe.xcodeproj	updates Beiwe2 app icon	3 months ago
Beiwe.xcworkspace	updates Beiwe2 app icon	3 months ago
Beiwe	updates Beiwe2 app icon	3 months ago
dev-scripts	Fix xcshared data name, and commit to git after rename	6 years ago
.gitignore	Build ResearchKit on Swift 4.2 instead of 4.0	11 months ago
LICENSE	Create LICENSE	4 years ago
Podfile	updates Beiwe2 app icon	3 months ago
Podfile.lock	updates Beiwe2 app icon	3 months ago

About

Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe iOS app code. The Beiwe2 app is also available on the Apple app store to use with open source builds of the Beiwe backend.

www.beiwe.org/

ios-app · beiwe

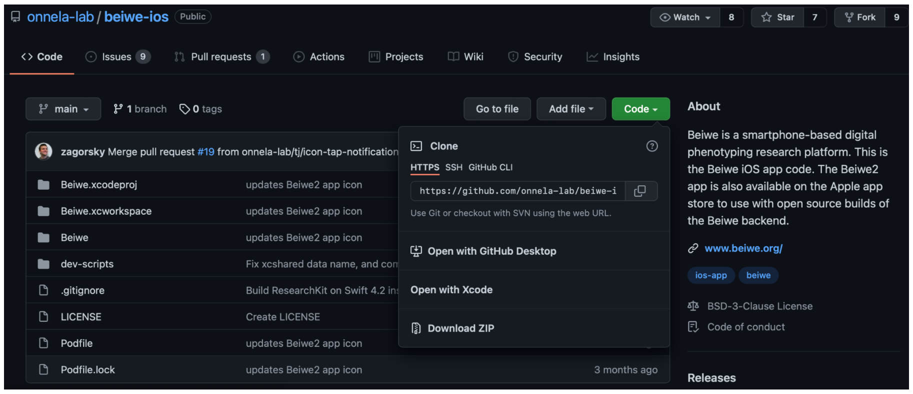
BSD-3-Clause License

Code of conduct

Releases

Figure 2: Main Page of the IOS Beiwe Repository

Once you've clicked the button, a window will pop up. From that window you will want to select **Download ZIP**.



zagorsky Merge pull request #19 from onnela-lab/tj/icon-tap-notification

Beiwe.xcodeproj	updates Beiwe2 app icon
Beiwe.xcworkspace	updates Beiwe2 app icon
Beiwe	updates Beiwe2 app icon
dev-scripts	Fix xcshared data name, and con
.gitignore	Build ResearchKit on Swift 4.2 in
LICENSE	Create LICENSE
Podfile	updates Beiwe2 app icon
Podfile.lock	updates Beiwe2 app icon 3 months ago

https://github.com/onnela-lab/beiwe-i

Use Git or checkout with SVN using the web URL.

Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe iOS app code. The Beiwe2 app is also available on the Apple app store to use with open source builds of the Beiwe backend.

www.beiwe.org/

Figure 3: Main Page of the IOS Beiwe Repository once ***Code*** *is selected.*

After the code is downloaded, save it somewhere it can easily be found. I like to save my things to my Desktop or to my Documents.

Cocoapods

The pods file for Beiwe is already created when the files are downloaded from Github.

Install on Computer

Open terminal, you use sudo or pip

```
In [ ]: sudo gem install cocoapods

        pip install cocoapods
```

Integrating with the Beiwe Project

Once installed, you will want to right click on your project folder and open terminal.

```
-MacBook-Pro-2 beiwe-ios-main %
```

Then we want to integrate the cocoapods with the current Beiwe project.

```
-MacBook-Pro-2 beiwe-ios-main % pod install
```

Once **pod install** is run, the cocoapods should be integrated into the project. Go into your Beiwe project folder and open the xcworkspace project file. If the cocoapods don't integrate into the project, try running **pod update** to make sure everything is up to date. If you are still struggling, try **pod deintegrate** and then **pod install** again.

Building the Project

While working on the project, these are the errors I encountered. As you work through the errors when building the project I hope these articles help solve the error.

Objective-c Headers

https://coderedirect.com/questions/425245/error-could-not-build-objective-c-module-firebase-with-swift-5

- Followed these steps and restarted xCode and the Objective-c headers error was gone
  Command used in terminal: rm -rf some_dir
  Error in terminal: “Could not find 'CFPropertyList' (>= 2.3.3, < 4.0) among 78 total gem(s) (Gem::MissingSpecError)”
    - Solved by doing “gem install CFPropertyList”

Work on “no such module” error, messed around with some of the fixes found in this stackoverflow message

- https://stackoverflow.com/questions/29500227/getting-error-no-such-module-using-xcode-but-the-framework-is-there

“Cannot find type ‘SectionHeader’” in Scope

https://developer.apple.com/forums/thread/691672

- Following the directions to change the build settings from yes -> no fixed this error

“No such model” Hakuba Error

Error is noted in the Github repository, just need to mess around with pod install, pod update, and the different branches of Hakuba. After downgrading the software and then updating it again, along with running the various pod commands, our team was able to integrate the Hakuba package into our project.

Downgrading Xcode Software

- Need to download old versions of Xcode from the apple development website
    - First trying version 13.0 (10.58GB)
    - Then version 12.3 (11.56GB)
- Xcode 12 not compatible with Monterey
    - https://ifnotnil.com/t/xcode-12-is-not-compatible-with-monterey/2093

“‘CellModel’ is ambiguous for type lookup in this context”

Swift class documentation

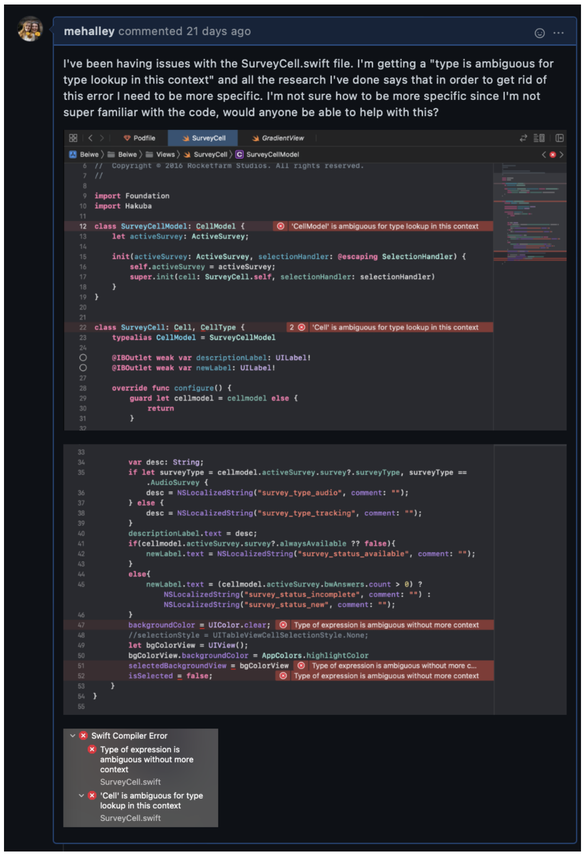
- https://docs.swift.org/swift-book/LanguageGuide/Protocols.html

Ambiguous without more context

- https://codesource.io/solved-type-of-expression-is-ambiguous-without-more-context/

Conclusion

I was unable to complete a successful build of the iOS frontend of Beiwe due to the error “‘CellModel’ is ambiguous for type lookup in this context.” Through my own research I was struggling to find an answer other than needing to be more specific with the variable I was declaring. I posted on the Beiwe iOS Github with this issue and it is currently being looked at by the original author’s of the Beiwe code.

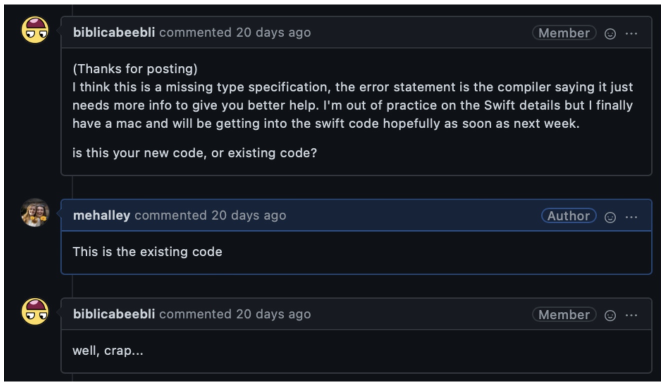


mehalley commented 21 days ago

I've been having issues with the SurveyCell.swift file. I'm getting a "type is ambiguous for type lookup in this context" and all the research I've done says that in order to get rid of this error I need to be more specific. I'm not sure how to be more specific since I'm not super familiar with the code, would anyone be able to help with this?

```
//  Copyright © 2016 Rocketfarm Studios. All rights reserved.
//

import Foundation
import Hakuba

class SurveyCellModel: CellModel {          'CellModel' is ambiguous for type lookup in this context
    let activeSurvey: ActiveSurvey;

    init(activeSurvey: ActiveSurvey, selectionHandler: @escaping SelectionHandler) {
        self.activeSurvey = activeSurvey;
        super.init(cell: SurveyCell.self, selectionHandler: selectionHandler)
    }
}


class SurveyCell: Cell, CellType {          2  'Cell' is ambiguous for type lookup in this context
    typealias CellModel = SurveyCellModel

    @IBOutlet weak var descriptionLabel: UILabel!
    @IBOutlet weak var newLabel: UILabel!

    override func configure() {
        guard let cellmodel = cellmodel else {
            return
        }


        var desc: String;
        if let surveyType = cellmodel.activeSurvey.survey?.surveyType, surveyType ==
            .AudioSurvey {
            desc = NSLocalizedString("survey_type_audio", comment: "");
        } else {
            desc = NSLocalizedString("survey_type_tracking", comment: "");
        }
        descriptionLabel.text = desc;
        if(cellmodel.activeSurvey.survey?.alwaysAvailable ?? false){
            newLabel.text = NSLocalizedString("survey_status_available", comment: "");
        }
        else{
            newLabel.text = (cellmodel.activeSurvey.bwAnswers.count > 0) ?
                NSLocalizedString("survey_status_incomplete", comment: "") :
                NSLocalizedString("survey_status_new", comment: "");
        }
        backgroundColor = UIColor.clear;    Type of expression is ambiguous without more context
        //selectionStyle = UITableViewCellSelectionStyle.None;
        let bgColorView = UIView();
        bgColorView.backgroundColor = AppColors.highlightColor
        selectedBackgroundView = bgColorView    Type of expression is ambiguous without more c...
        isSelected = false;                 Type of expression is ambiguous without more context
    }
}

```

```
Swift Compiler Error
    Type of expression is ambiguous without more context
    SurveyCell.swift
    'Cell' is ambiguous for type lookup in this context
    SurveyCell.swift
```



biblicabeebli commented 20 days ago Member

(Thanks for posting)
I think this is a missing type specification, the error statement is the compiler saying it just needs more info to give you better help. I'm out of practice on the Swift details but I finally have a mac and will be getting into the swift code hopefully as soon as next week.

is this your new code, or existing code?

mehalley commented 20 days ago Author

This is the existing code

biblicabeebli commented 20 days ago Member

well, crap...

8 Appendix C: Running the Beiwe Android Front-end

Running the Beiwe Android Front-end

Madeline Halley (mehalley@wpi.edu) and Jyalu Wu (jwu7@wpi.edu)

Setting up Android Studio and downloading the Beiwe code

We will be downloading Beiwe for Android, you do not need to have a Windows system to edit the code but in order to run the code you will need to have an Android device.

Android Studio

You will need to download Android studio from the internet, once searched you will want to click on the website that is **developer.android.tools**.

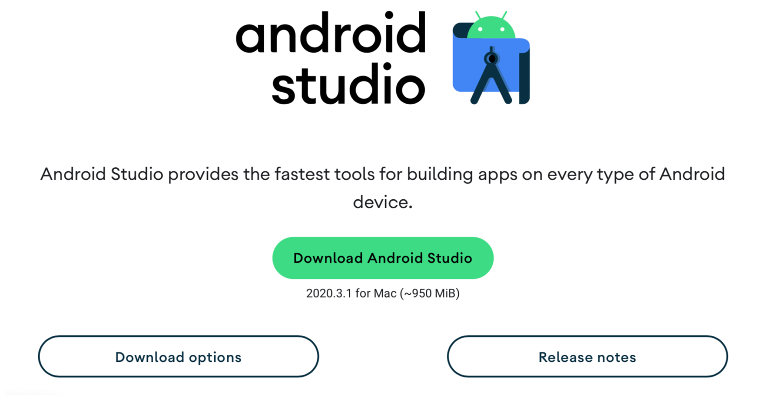

Android Studio provides the fastest tools for building apps on every type of Android device.

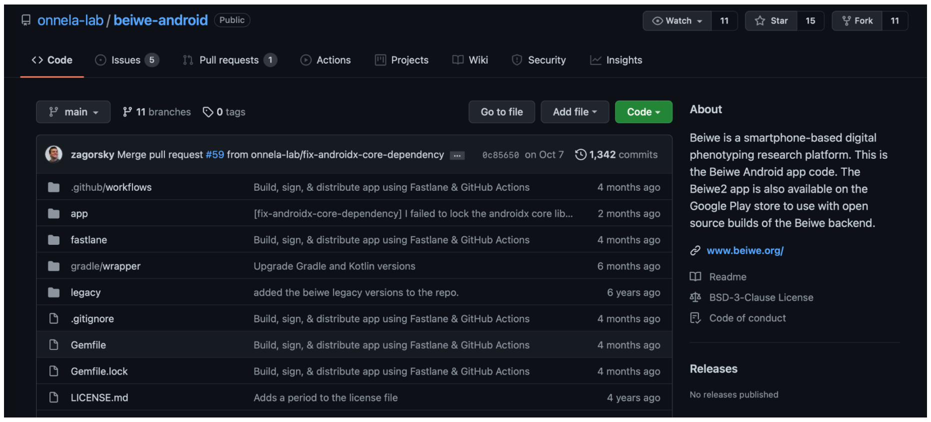
Figure 1: Website Once Opened

You will want to download the version for the computer you are using. I am using a mac computer so my suggested download is for Mac, but if you have a Windows or Linux system you will want to download that version.

Download Beiwe Code

Beiwe Android front end code can be found on Github.
Link: https://github.com/onnela-lab/beiwe-android

In order to download the code you will need to access the Git Repository. Once you have opened the Repository, click on the green button that reads **Code**.


onnela-lab / beiwe-android Public
Watch 11 Star 15 Fork 11
Code Issues 5 Pull requests 1 Actions Projects Wiki Security Insights
main 11 branches 0 tags
Go to file Add file Code

zagorsky Merge pull request #59 from onnela-lab/fix-androidx-core-dependency … 0c85650 on Oct 7 1,342 commits

.github/workflows	Build, sign, & distribute app using Fastlane & GitHub Actions	4 months ago
app	[fix-androidx-core-dependency] I failed to lock the androidx core lib…	2 months ago
fastlane	Build, sign, & distribute app using Fastlane & GitHub Actions	4 months ago
gradle/wrapper	Upgrade Gradle and Kotlin versions	6 months ago
legacy	added the beiwe legacy versions to the repo.	6 years ago
.gitignore	Build, sign, & distribute app using Fastlane & GitHub Actions	4 months ago
Gemfile	Build, sign, & distribute app using Fastlane & GitHub Actions	4 months ago
Gemfile.lock	Build, sign, & distribute app using Fastlane & GitHub Actions	4 months ago
LICENSE.md	Adds a period to the license file	4 years ago

About

Beiwe is a smartphone-based digital phenotyping research platform. This is the Beiwe Android app code. The Beiwe2 app is also available on the Google Play store to use with open source builds of the Beiwe backend.

www.beiwe.org/

Readme

BSD-3-Clause License

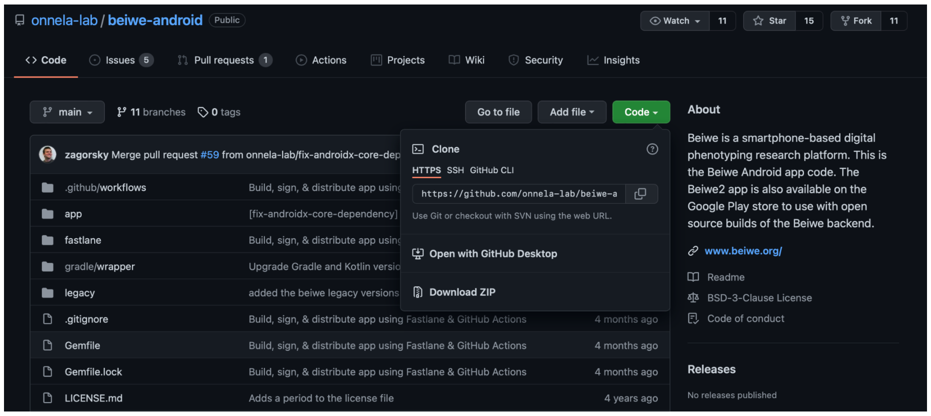
Code of conduct

Releases

No releases published

Figure 2: Main Page of the Android Beiwe Repository

Once you've clicked the button, a window will pop up. From that window you will want to select **Download ZIP**.



https://github.com/onnela-lab/beiwe-a

Use Git or checkout with SVN using the web URL.

*Figure 3: Main Page of the Android Beiwe Repository once **Code** is selected.*

After the code is downloaded, save it somewhere it can easily be found. I like to save my things to my Desktop or to my Documents.

Building the Project

When first opening the project in Android Studio, it will build with no errors but you will need to build the project again which is where errors will begin to emerge. Below I have listed the ones I encountered and how I fixed them.

SDK Version Issues

`compileSdkVersion` was 30 needed to be 31
`targetSdkVersion` was 29 needed to be 31
`minSdkVersion` was 16 needed to be 19

Android Manifest

```
<activity
   android:name="org.beiwe.app.ui.LoadingActivity"
  android:label="${appName}${appNameSuffix}"
   android:exported="true">
<receiver android:name=".listeners.BootListener"
   android:exported="true">
<receiver android:name=".listeners.SmsReceivedLogger"
   android:exported="true">
<receiver android:name=".listeners.PowerStateListener"
   android:exported="true">
```

Needed to add the android:exported line to each of this activities and receivers

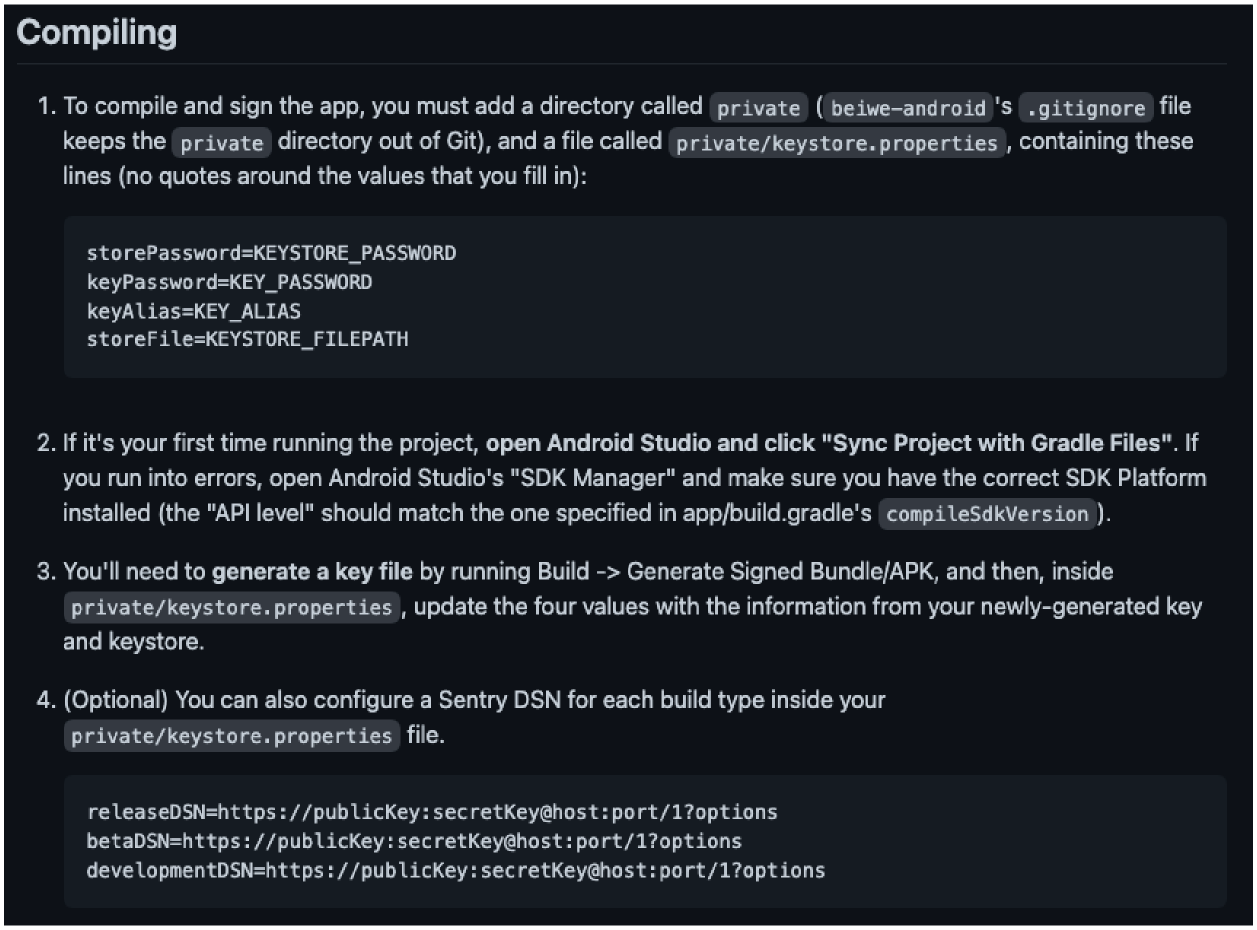
FirebaseInstanceId

- Make sure firebase is set up properly -> instructions in the github README
    - https://firebase.google.com/docs/android/setup
- Package is deprecated -> need to change toFirebaseMessaging
    - **Lines 269-308**: made changes in MainService.java
    - https://firebase.google.com/docs/reference/android/com/google/firebase/iid/FirebaseInstanceId

ComStatsCustomUrlBeta

Error: “com.android.ide.common.signing.KeytoolException: Failed to read key androidBeiwe from store”

- Use this tutorial to generate a key and keystore, and sign the app with it. Scroll down to where it says “Sign your app for release to Google Play” and follow all steps in that section.
    - This tutorial may also come in handy.
    - Use the values listed in the README from Beiwe github to generate the APK to connect the keystore.properties (screenshot below)


Compiling

1. To compile and sign the app, you must add a directory called `private` (`beiwe-android`'s `.gitignore` file keeps the `private` directory out of Git), and a file called `private/keystore.properties`, containing these lines (no quotes around the values that you fill in):

```
storePassword=KEYSTORE_PASSWORD
keyPassword=KEY_PASSWORD
keyAlias=KEY_ALIAS
storeFile=KEYSTORE_FILEPATH
```

2. If it's your first time running the project, **open Android Studio and click "Sync Project with Gradle Files"**. If you run into errors, open Android Studio's "SDK Manager" and make sure you have the correct SDK Platform installed (the "API level" should match the one specified in app/build.gradle's `compileSdkVersion`).
3. You'll need to **generate a key file** by running Build -> Generate Signed Bundle/APK, and then, inside `private/keystore.properties`, update the four values with the information from your newly-generated key and keystore.
4. (Optional) You can also configure a Sentry DSN for each build type inside your `private/keystore.properties` file.

```
releaseDSN=https://publicKey:secretKey@host:port/1?options
betaDSN=https://publicKey:secretKey@host:port/1?options
developmentDSN=https://publicKey:secretKey@host:port/1?options
```

Changing Server URL

Change the “production_website” string in the file res/values/strings.xml to https://emutivo-beiwe.wpi.edu.

```
<string name="production_website" translatable="false">https://emutivo-beiwe.wpi.edu </string>
```

Permission Errors

Went through and made sure ruby was properly downloaded (Mac)

- https://www.moncefbelyamani.com/how-to-install-xcode-homebrew-git-rvm-ruby-on-mac/?utm_source=stackoverflow

Original error: While executing gem ... (Gem::FilePermissionError) You don't have write permissions for the /Library/Ruby/Gems/2.6.0 directory.

- https://stackoverflow.com/questions/51126403/you-dont-have-write-permissions-for-the-library-ruby-gems-2-3-0-directory-ma

CommStatsCustomUrlBetaJavaWithJavac

BuildConfig error -> solved by generating a bundle/APK

APK, keystore.properties file

- Make sure to generate a signed APK - clean/build
- Check the SDK platform version
    - Must be the same as the SDK version in the gradle file

Fastlane -> Google Play Console

In order for the frontend to run it needs to be connected to the Google play console in order for the information collected to be uploaded to the cloud
- Need API access

Fastlane local setup from Android Beiwe Github

You shouldn't need to do this, other than for debugging the Fastlane setup. We intend Fastlane to be primarily run on the cloud, inside GitHub Actions. But if for some reason you decide to run it locally, here's how to do it:
1. Follow Fastlane's Android getting started documentation, including: install Ruby. Then gem install bundler, and then bundle install.
2. Make sure you have the following credentials, and that they're correct:
    a. The keystore inside this repo at private/signing_keystore.jks
    b. Your private/keystore.properties file, which should include releaseDSN
    c. Your Google Service Account credentials JSON file, which enables you to upload the Android App Bundle to Google Play. Its location is defined in fastlane/Appfile.
    d. AWS IAM credentials that allow you to upload an object to the bucket used for hosting APK files for download. Your credentials should be exported as two environment variables called AWS_ACCESS_KEY_IDand AWS_SECRET_ACCESS_KEY.
3. Run fastlane buildAAB
4. Run fastlane buildAPKs

Once I set up Google Play Console and fastlane, I opened up the application to many more errors in terms of RegistrationActivity and various build errors.

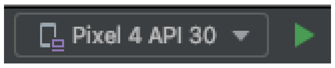
Signed APK Error

Solution for this was to re-download from the original github, which built with no errors initially
- Next steps to see if we can run on an android divide
- See what happens and evaluate from there

1.https://developer.android.com/studio/publish/app-signing#sign_release
2.https://stackoverflow.com/questions/57253143/keystoreexception-this-keystore-does-not-support-probing-and-must-be-loaded-wit

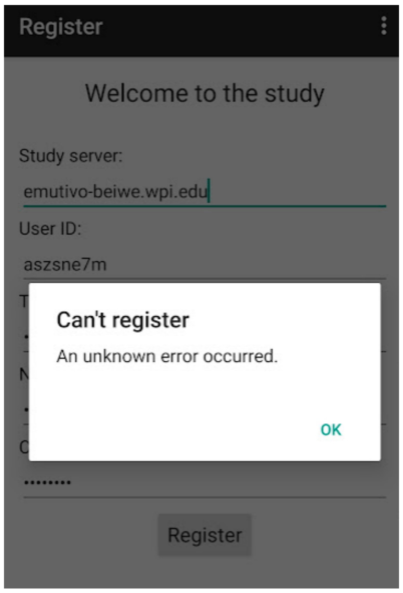
Running the Application With Emutivo Development VM

Note: We have only managed to run the application using an actual Android phone. We could not get it to work using an emulator.

1. First, create a new participant in the backend.
2. Open Android Studio and open the Beiwe Android frontend code.
3. Connect your computer to your Android device using a USB cord. The device selection button next to the “Run” button should automatically select your device to be run on.


4. Click the “Run” button and run the app on your device.
5. You should be taken to a login screen – log in using the participant information generated by the backend and choose your password.

Error: When following the steps in “Running The Beiwe Backend On The Emutivo Development VM” to set up the backend and create a new participant, the app will show the following error:



Can't register

An unknown error occurred.

OK

Tracing the error through logs shows us that this is caused by a 500 error:

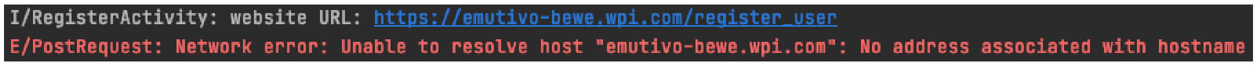
```
I/RegisterActivity: register button pressed
I/RegisterActivity: trying to register with server
I/RegisterActivity: website URL: https://emutivo-beiwe.wpi.edu/register_user
I/RegisterActivity: on post execute
E/RegisterActivity: 500can't register
```

The “can’t register” is something that we printed out, not the official error.

Figuring Out Why There's An Error

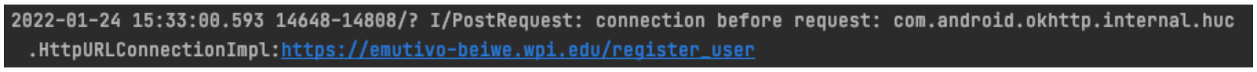
List of things the error **IS NOT** caused by:

1. Can't connect to the website / website doesn't exist
    a. Would give a 502 error and different log



```
I/RegisterActivity: website URL: https://emutivo-bewe.wpi.com/register_user
E/PostRequest: Network error: Unable to resolve host "emutivo-bewe.wpi.com": No address associated with hostname
```

2. Using a https website instead of http → we thought maybe this was the case because the app uses the function "HttpURLConnectionImp" instead of "HttpsURLConnectionImp"



```
2022-01-24 15:33:00.593 14648-14808/? I/PostRequest: connection before request: com.android.okhttp.internal.huc
  .HttpURLConnectionImpl:https://emutivo-beiwe.wpi.edu/register_user
```

    a. "HttpURLConnectionImp" actually works with both http and https websites
3. AWS S3 error
    a. We connected the application with an S3 bucket (see step II.III for directions on how to do that) but the error persisted

Conclusion

After working through these errors, our team had a successful build of the Android frontend of Beiwe. However, the frontend does not work perfectly with the backend. We determined that the backend error is likely caused by an AWS EC2 error – the Android frontend is expecting to connect to a Beiwe AMI but detects the WPI Emutivo development VM instead.

9 Appendix D: Running the Beiwe Backend on Linux

Running The Beiwe Backend Locally On Linux

[Deprecated - See "Running The Beiwe Backend On The Development VM" for updated instructions]

Madeline Halley (mehalley@wpi.edu) and Jyalu Wu (jwu7@wpi.edu)

Note: This was our first method for running the Beiwe backend without using the AWS EC2 server. We saw this through completion but determined that this method does not work since the study server is not an actual URL but a port on the VM.

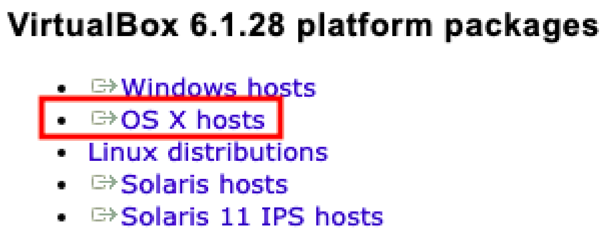
One way to run the Beiwe backend locally is through Ubuntu. This documentation will include steps on how to set up an Ubuntu virtual machine on a Mac, but the process should be similar for a Windows machine. The rest of the steps should be the same if run on any Ubuntu machine / Ubuntu VM.

I. Setting Up an Ubuntu VM (MacOS)

These steps are taken from this tutorial by dev2qa and from this tutorial by how2geek.

Download VirtualBox

1. First, head to this link to download VirtualBox.
2. Go to where it says VirtualBox *[version number]* platform packages and click the where it says OS X hosts. The download should happen automatically.



VirtualBox 6.1.28 platform packages

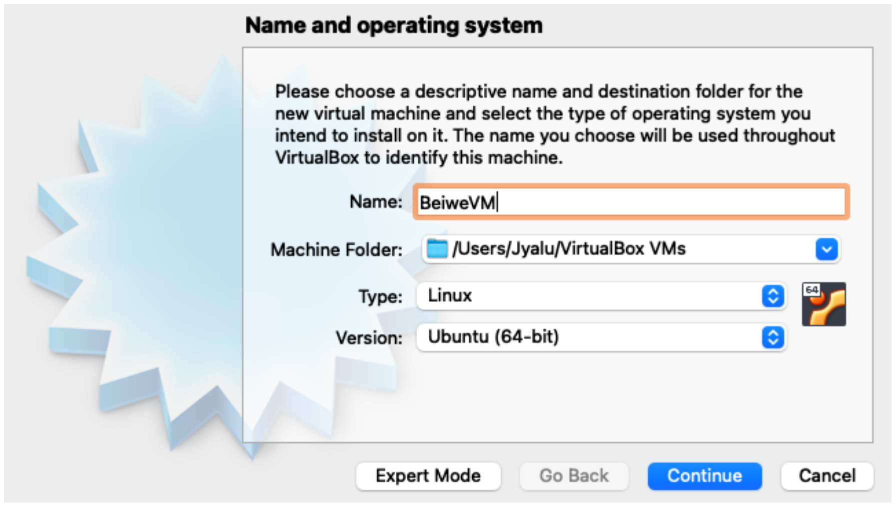
- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts
- Solaris 11 IPS hosts

3. After downloading it, open up the downloaded file and finish installing VirtualBox.

Set Up a New Ubuntu VM

1. Open up VirtualBox.

2. Click on the “New” button to create a new VM.
3. Input a custom name for your VM, choose “Linux” as the Type and “Ubuntu (64-bit)” as the Version.



Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

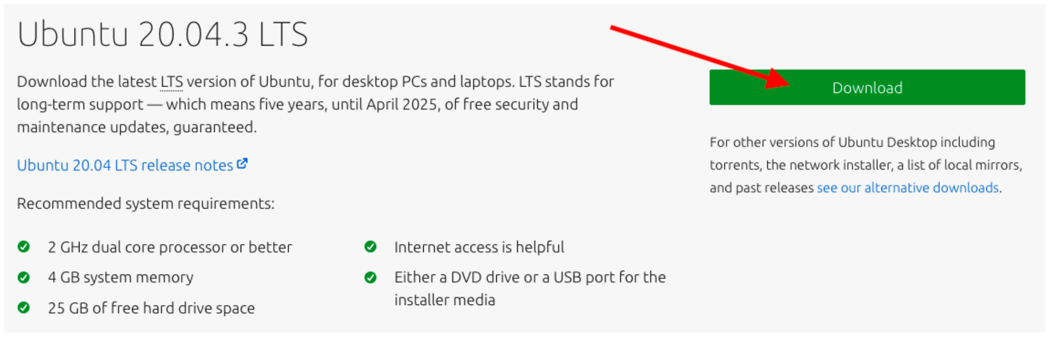
Name: BeiweVM

Machine Folder: /Users/Jyalu/VirtualBox VMs

Type: Linux

Version: Ubuntu (64-bit)

4. Click the “Continue” button and adjust the virtual machine’s hard disk size, memory size, and CPU processor number if necessary. After you’re done, click the “Create” button.
    a. The Beiwe server will need quite a lot of space in order to run smoothly since we will need to create a local sqlite database as well. 2048MB is not enough. I chose about 4000MB but this may be excessive.
5. Head to ubuntu.com and download the ubuntu iso file to a local folder.



Ubuntu 20.04.3 LTS

Download the latest LTS version of Ubuntu, for desktop PCs and laptops. LTS stands for long-term support — which means five years, until April 2025, of free security and maintenance updates, guaranteed.

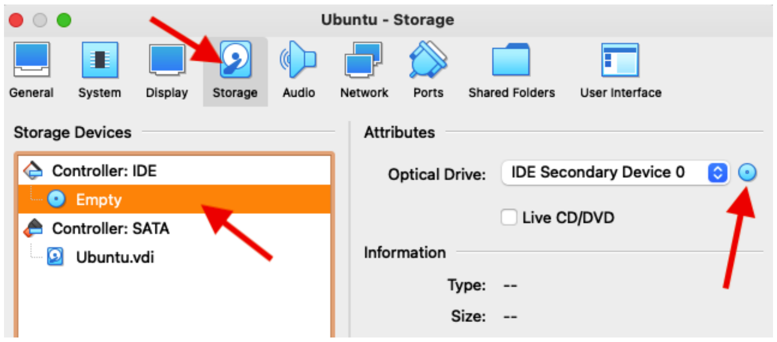
Ubuntu 20.04 LTS release notes

Recommended system requirements:

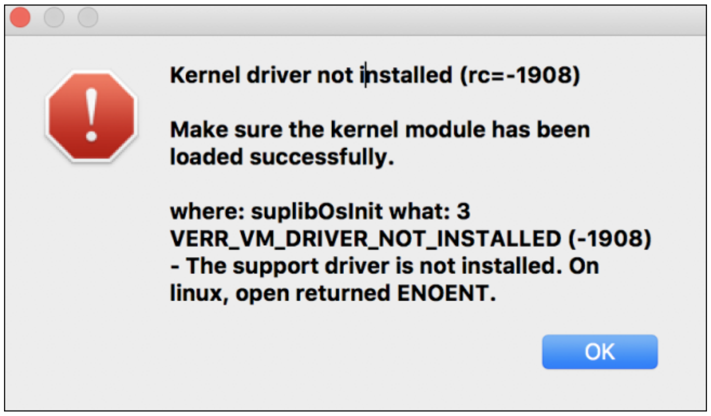
- 2 GHz dual core processor or better
- 4 GB system memory
- 25 GB of free hard drive space
- Internet access is helpful
- Either a DVD drive or a USB port for the installer media

For other versions of Ubuntu Desktop including torrents, the network installer, a list of local mirrors, and past releases see our alternative downloads.

6. Select the new VM in the left panel, click “Settings” on the top menu bar, then click “Storage”.
7. Select the blue laser disk icon (it should look like a blue CD) in the left Storage Devices panel, then click the laser disk icon again where it says “Optical Drive”. Click “Choose/Create Virtual Optical Disk File” and select the ubuntu iso file you just downloaded.



8. Select the VM on the left side and click the "Start" button on the top menu bar. If you get an error that looks like this, head to the next section.



Kernel driver not installed (rc=-1908)

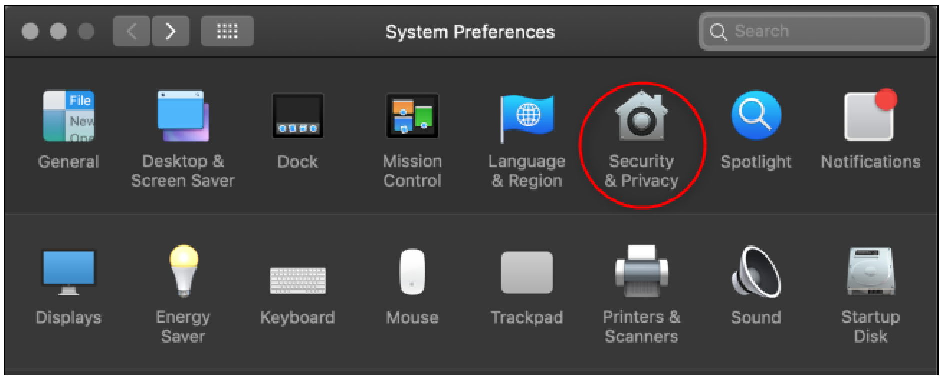
Make sure the kernel module has been loaded successfully.

where: suplibOsInit what: 3 VERR_VM_DRIVER_NOT_INSTALLED (-1908) - The support driver is not installed. On linux, open returned ENOENT.

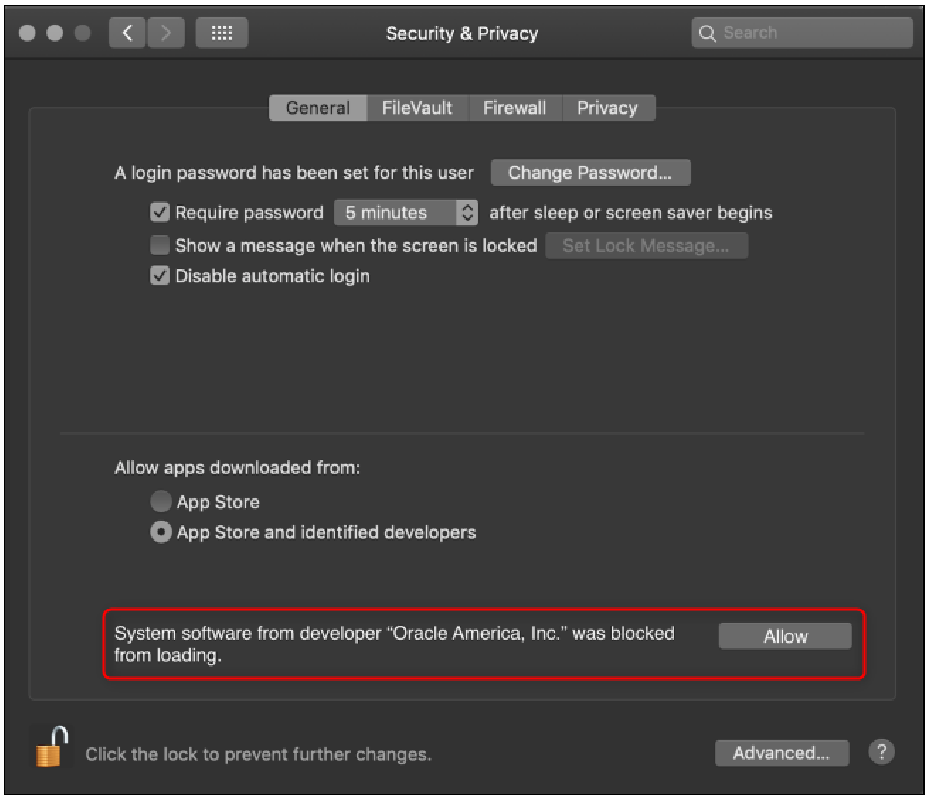
Fixing the "Kernel driver not installed" Error

Make sure that you do this within 30 minutes of installing VirtualBox. If the "System software from developer..." message doesn't appear in your System Preferences, uninstall VirtualBox and redo everything.

1. Go to the System Preferences app, then click "Security and Privacy".



2. There should be text near the bottom that says, "System Software from Developer 'Oracle America, Inc.' Was Blocked from Loading." Click the "Allow" button and click the lock to prevent further changes.



A login password has been set for this user Change Password...

[✓] Require password 5 minutes after sleep or screen saver begins

[ ] Show a message when the screen is locked Set Lock Message...

[✓] Disable automatic login

Allow apps downloaded from:

○ App Store

● App Store and identified developers

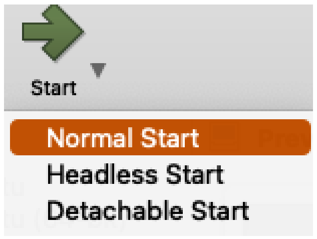
System software from developer "Oracle America, Inc." was blocked from loading. Allow

Click the lock to prevent further changes. Advanced... ?

II. Running the VM

Installing Ubuntu in the VM

1. Select the VM on the left panel and click **Start —> Normal Start** in the top menu.



2. After the VM gets up and running, it will show the Ubuntu installation wizard.
3. It will then show you two options: "Try Ubuntu" and "Install Ubuntu". I chose "Install Ubuntu".
4. Then choose what software will be installed. I chose the "Normal installation" option.
5. Then, you can choose the "Erase disk and install Ubuntu" option if you want to go with default settings for the Ubuntu OS. Otherwise, feel free to choose the ""Something else" option to customize the OS.
6. Finally, create a user account so you can log into Ubuntu. Make sure you save this information!

Fixing Screen Resolution

If your VM was cutting off the desktop during the installation process, you can change the screen resolution to fix this problem.

1. Go to “Show Applications” on the dock in your VM, search “Settings”, click “Settings”.
2. Scroll and click on “Display” on the left menu.
3. Change the resolution to whatever makes sense for your screen size.

III. Running Beiwe Locally on the VM

These steps are copied from the Beiwe backend github and this tutorial by Liquid Web.

Download the Code

1. In your VM, head to https://github.com/onnela-lab/beiwe-backend and download the code. Save this in your Documents folder.

Setting Up a Virtual Environment

You should create a virtual environment for running the Beiwe codebase - it expects at least Python 3.6.

1. Install some necessary prerequisites:
   a. `sudo apt-get update; sudo apt-get install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev`
   b. `sudo apt install git-all`
   c. `sudo apt install curl`
2. Install the latest version of pyenv:
   a. `git clone https://github.com/pyenv/pyenv.git ~/.pyenv`
3. Set up some important environment variables and pyenv autocompletion:
   a. `echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc`
   b. `echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc`
   c. `echo -e 'if command -v pyenv 1>/dev/null 2>&1; then\n eval "$(pyenv init -)"\nfi' >> ~/.bashrc`
4. To start using pyenv, restart the shell by running:
   a. `exec "$SHELL"`

Setting Up the Codebase

1. Open the folder in Terminal and run these commands:
   a. `sudo apt-get update; sudo apt-get install postgresql libpq-dev`

   b. sudo apt install python3-pip
   c. pip install --upgrade pip setuptools wheel
   d. pip install -r requirements.txt
   e. Create a file for your environment variables that contains at least these:
      i. export DOMAIN_NAME="localhost://8080"
      ii. export FLASK_SECRET_KEY="asdf"
      iii. export S3_BUCKET="a"
      iv. Export SYSADMIN_EMAILS="sysadmin@localhost"
   f. I usually store it at private/environment.sh.
2. There's no need to set up a local database since the Beiwe Backend automatically recognizes when it's running locally and creates a sqlite database in the private folder.
3. Set up the database schema:
   a. python3 manage.py migrate

Running Beiwe

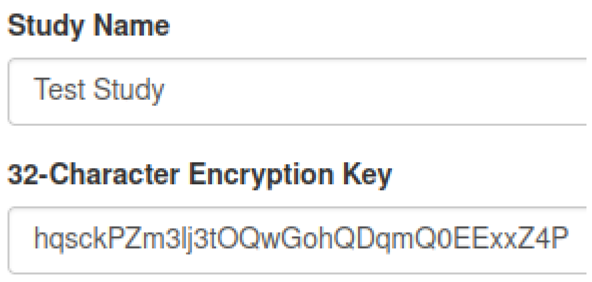
1. To run your local codebase run source private/environment.sh and then python3 app.py from the project root.
   a. When you run python3 app.py Beiwe checks for any missing required variables and tells you if anything is missing.
2. To get the python shell for Beiwe, run python3 manage.py shell_plus from the project root.

IV. Using the Study Management Portal

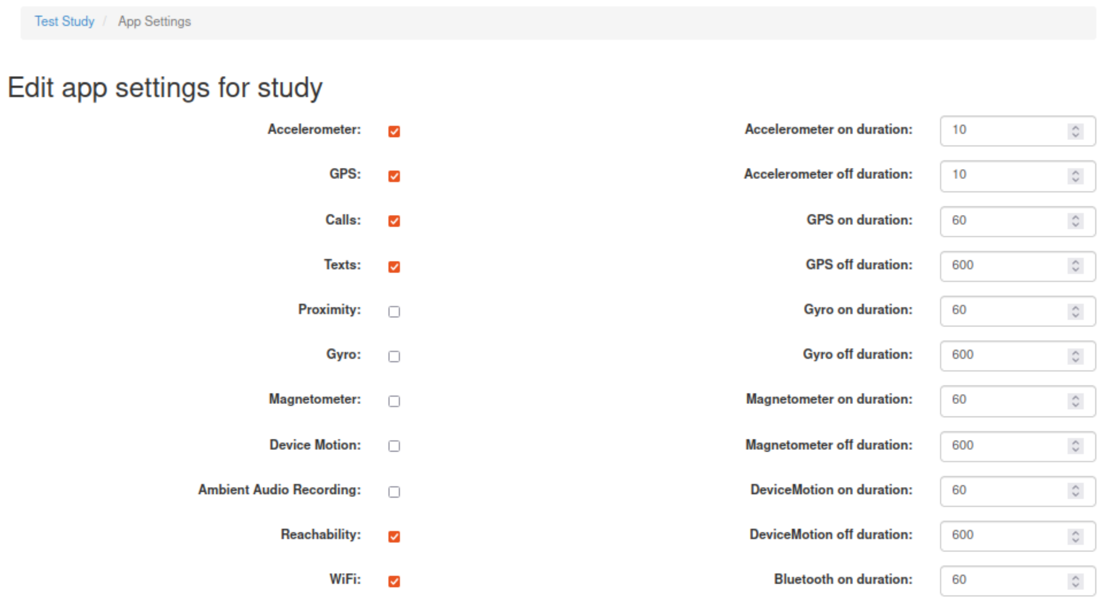
1. Once you start up the Beiwe server, head to the port address shown in the terminal. You will be redirected to a login page. Log in with the following credentials:
   a. Username: default_admin
   b. Password: abc123
2. For safety, change the login password since this is a default password.

Creating a New Study

1. Head to the "Manage Studies" tab and click on the "Create New Study" button. Give a name to your study and encrypt it with a 32-character key.


Study Name
Test Study
32-Character Encryption Key
hqsckPZm3lj3tOQwGohQDqmQ0EExxZ4P

2. After creating the study, you will be taken to a page that allows you to customize the types of data your study will collect.

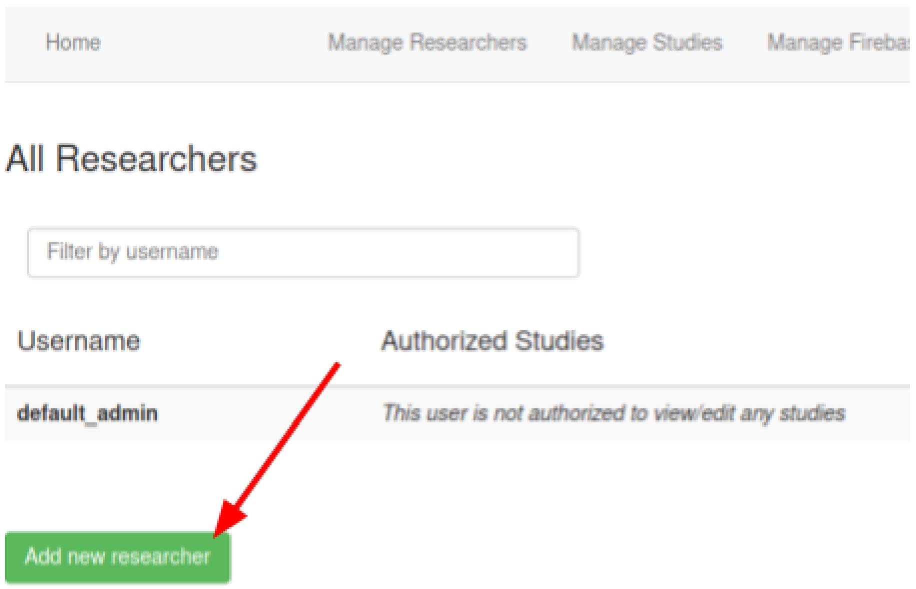

Test Study / App Settings

Edit app settings for study

Accelerometer:	✓	Accelerometer on duration:	10
GPS:	✓	Accelerometer off duration:	10
Calls:	✓	GPS on duration:	60
Texts:	✓	GPS off duration:	600
Proximity:	☐	Gyro on duration:	60
Gyro:	☐	Gyro off duration:	600
Magnetometer:	☐	Magnetometer on duration:	60
Device Motion:	☐	Magnetometer off duration:	600
Ambient Audio Recording:	☐	DeviceMotion on duration:	60
Reachability:	✓	DeviceMotion off duration:	600
WiFi:	✓	Bluetooth on duration:	60

Creating a New Researcher for the Study

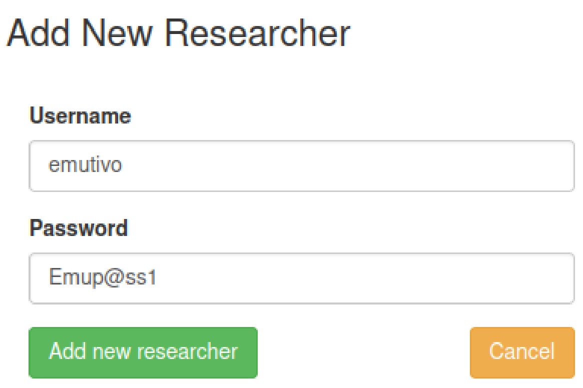
1. Head to the "Manage Researchers" tab and click on the "Add new researcher" button.



All Researchers

Username	Authorized Studies
default_admin	*This user is not authorized to view/edit any studies*

2. Choose a username and password for the new researcher, then click "Add new researcher". An example username and pass is shown below.


Add New Researcher

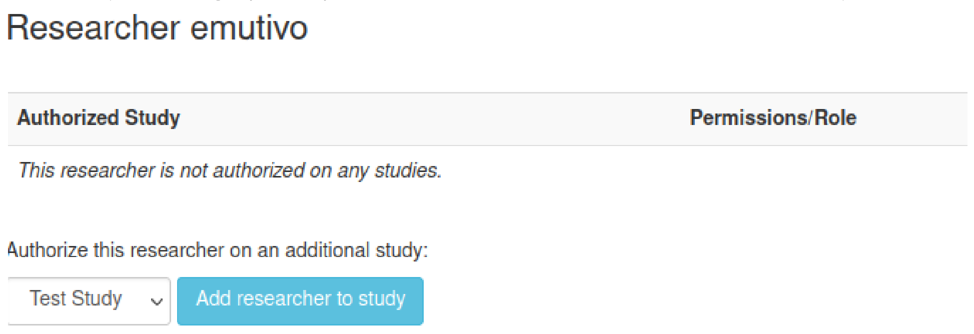
Username

emutivo

Password

Emup@ss1

3. Authorize the new researcher to keep track of the study you just created by adding them to it - select a study then click “Add researcher to study”. This allows the researcher to edit the study, create new surveys, manage participants, and download the data from the study.


Researcher emutivo

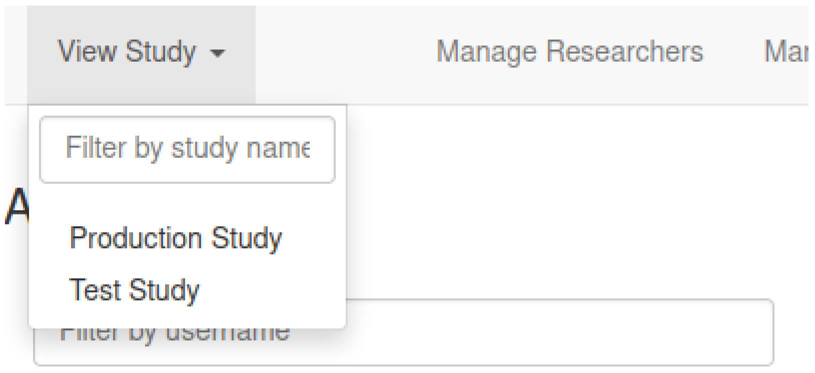
Authorized Study	Permissions/Role
This researcher is not authorized on any studies.	

Authorize this researcher on an additional study:

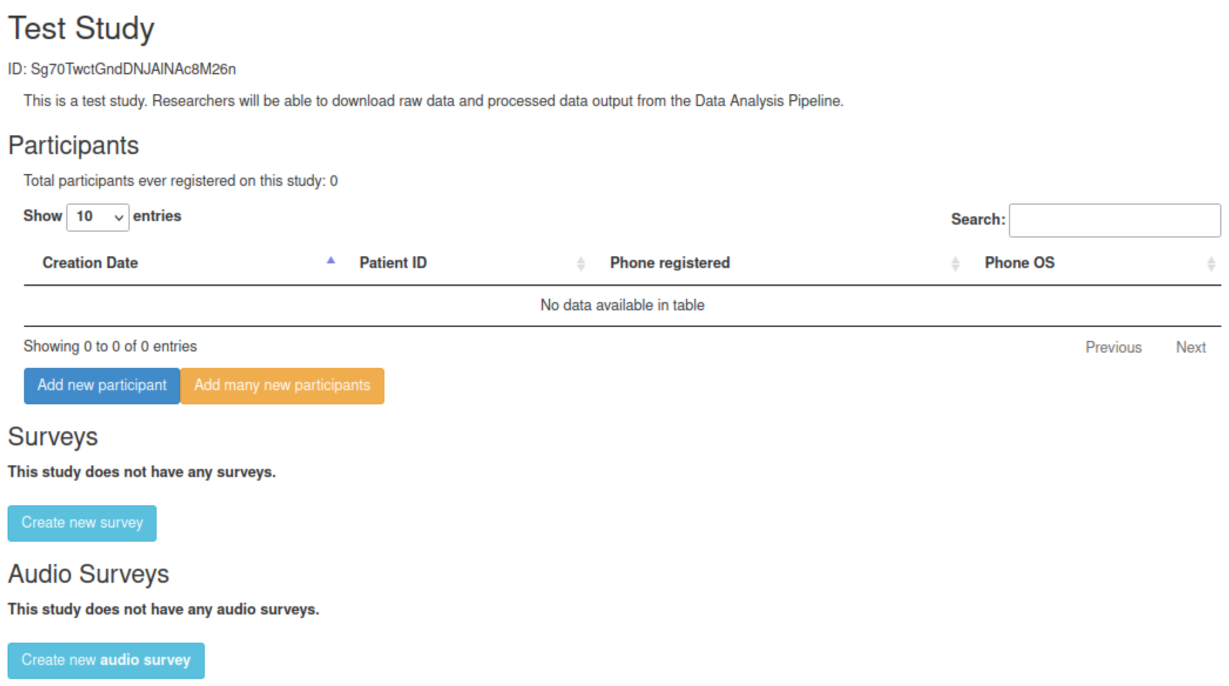
Test Study

Adding Participants to the Study

1. Head to the “View Study” dropdown list on the top menu and select a study.



2. Click on “Add new participant” to generate a unique ID for a new participant.


Test Study

ID: Sg70TwctGndDNJAlNAc8M26n

This is a test study. Researchers will be able to download raw data and processed data output from the Data Analysis Pipeline.

Participants

Total participants ever registered on this study: 0

Show 10 entries Search:

Creation Date	Patient ID	Phone registered	Phone OS
No data available in table			

Showing 0 to 0 of 0 entries Previous Next

Surveys

This study does not have any surveys.

Audio Surveys

This study does not have any audio surveys.

3. Ask the new participant to download the “Beiwe2” app on their phone and give them their unique participant ID, as well as the study server. The participant can then log into the app and start using it for data collection.

Note: Running the study management portal locally means that the study server is not an actual URL but is a port on the VM. This means that the participant won’t be able to log into the app. Updated instructions on solving this problem are in the tutorial titled “Running The Beiwe Backend On The Emutivo Development VM”.

10 Appendix E: Running the Beiwe Backend on Emutivo Development VM

Running The Beiwe Backend On The Emutivo Development VM

[Deprecated - See "Running The Beiwe Backend On AWS" for updated instructions]

Madeline Halley (mehalley@wpi.edu) and Jyalu Wu (jwu7@wpi.edu)

Note: This was our second method for running the Beiwe backend without using the AWS EC2 server. We saw this through completion but determined that this method does not work since integration with the Android frontend gave errors.

Note: The VM will only run on WPI machines or when connected to the GlobalProtect VPN. Login information for the development playground was set up by Dr. Ermal Toto. The following instructions are Mac-specific. Running the VM on a Windows machine may be different.

I. Connecting to the Development VM

1. On your personal device, connect to the GlobalProtect VPN and open Terminal.
2. Generate an ssh key if you don't have one already: `ssh-keygen -t rsa`
3. Run the VM:
    a. `ssh-copy-id [username]@emutivo-dev.wpi.edu`
    b. `ssh [username]@emutivo-dev.wpi.edu`
4. Enter the root environment
    a. `sudo su -`
    b. Enter in your password
5. Download Cyberduck for transfering files. Make sure to ask Dr. Toto to give you sudo access when using Cyberduck.
6. Connect to Cyberduck using the following information:
    a. SFTP
    b. Server: emutivo-dev.wpi.edu
    c. Username: your username
    d. SSH Private Key: the ssh key you just made
7. Redirect to `/var/www/emutivo-dev` - this is where the Beiwe backend code will live.

II. Setting Up the Beiwe Backend

It is not necessary to follow these steps – this only needs to be done once.

Download the Code

1. In your VM, head to https://github.com/onnela-lab/beiwe-backend and download the code. Save this in your Documents folder.
2. Using Cyberduck, transfer the files you just downloaded to the development VM in `/var/www/emutivo-dev.`

Setting Up a Virtual Environment

You should create a virtual environment for running the Beiwe codebase - it expects at least Python 3.6.

1. Install some necessary prerequisites:
    a. `sudo apt-get update; sudo apt-get install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev`
    b. `sudo apt install git-all`
    c. `sudo apt install curl`
2. Install the latest version of pyenv:
    a. `git clone https://github.com/pyenv/pyenv.git ~/.pyenv`
3. Set up some important environment variables and pyenv autocompletion:
    a. `echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc`
    b. `echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc`
    c. `echo -e 'if command -v pyenv 1>/dev/null 2>&1; then\n eval "$(pyenv init -)"\nfi' >> ~/.bashrc`
4. To start using pyenv, restart the shell by running:
    a. `exec "$SHELL"`
5. Install pyenv 3.8.3 and run a virtual environment on python 3.8.3.
    a. `pyenv install --list`
    b. `pyenv install 3.8.3`
    c. `pyenv global 3.8.3`

Setting Up the Codebase

1. Open the folder in Terminal and run these commands:
    a. `sudo apt-get update; sudo apt-get install postgresql libpq-dev`
    b. `sudo apt install python3-pip`
    c. `pip install --upgrade pip setuptools wheel`
    d. `pip install -r requirements.txt`
    e. Create a file for your environment variables that contains at least these:

i. export DOMAIN_NAME="emutivo-dev.wpi.edu"
ii. export FLASK_SECRET_KEY="asdf"
iii. export S3_BUCKET="a"
iv. Export SYSADMIN_EMAILS="sysadmin@localhost"

f. I usually store it at private/environment.sh.

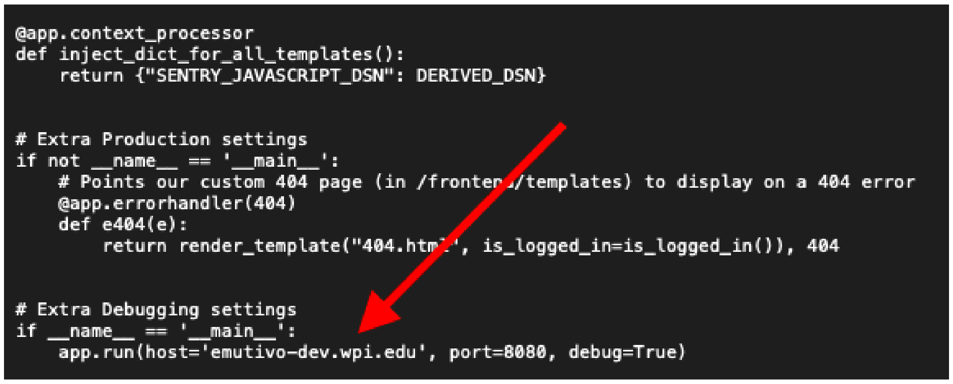
3. There's no need to set up a local database since the Beiwe Backend automatically recognizes when it's running locally and creates a sqlite database in the private folder.
4. Set up the database schema:
   a. python3 manage.py migrate

II. Running the Beiwe Backend

Method I: Running Beiwe With Flask 'Run' Command

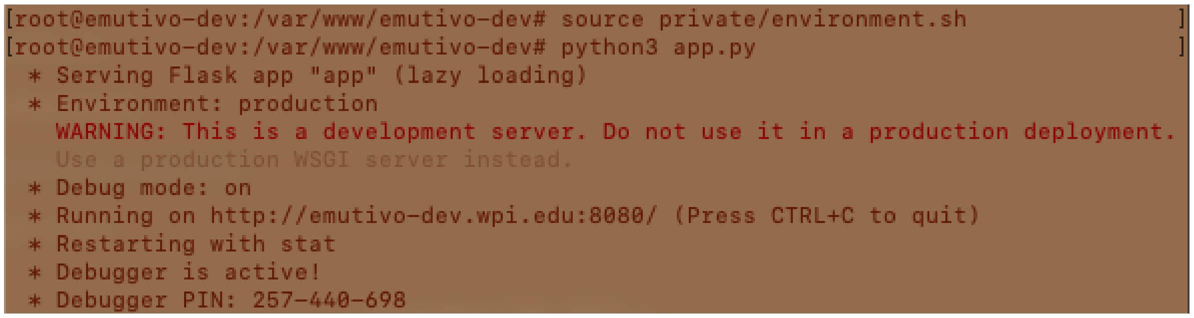
We first tried to run Beiwe using the Flask 'run' command directly in the virtual machine. However, **this method is not recommended** – see the 'Method II' section for a better method.

1. Change the last line of app.py so it runs on emutivo-dev.wpi.edu instead of localhost.



```
@app.context_processor
def inject_dict_for_all_templates():
    return {"SENTRY_JAVASCRIPT_DSN": DERIVED_DSN}


# Extra Production settings
if not __name__ == '__main__':
    # Points our custom 404 page (in /frontend/templates) to display on a 404 error
    @app.errorhandler(404)
    def e404(e):
        return render_template("404.html", is_logged_in=is_logged_in()), 404


# Extra Debugging settings
if __name__ == '__main__':
    app.run(host='emutivo-dev.wpi.edu', port=8080, debug=True)
```

2. To run your local codebase run source private/environment.sh and then python3 app.py from the project root.
   a. When you run python3 app.py Beiwe checks for any missing required variables and tells you if anything is missing.
3. Head to a browser and open emutivo-dev.wpi.edu:8080 to see your changes.



```
[root@emutivo-dev:/var/www/emutivo-dev# source private/environment.sh          ]
[root@emutivo-dev:/var/www/emutivo-dev# python3 app.py                         ]
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://emutivo-dev.wpi.edu:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 257-440-698
```

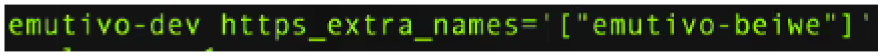
This method of running Beiwe results in the following problems:

4. **Problem I:** since the Beiwe website is a Flask app, the 'run' command must include a host (e.g. emutivo-dev.wpi.edu) and a port (e.g. 8080) as parameters. Thus, the development vm will run the study management portal on emutivo-dev.wpi.edu:8080 instead of the desired emutivo-dev.wpi.edu.
5. **Problem II:** since the Beiwe website runs locally on the emutivo-dev VM, any changes made on the website are lost once the user exits the Flask app (CTRL+C). These changes include creating new researchers, studies, users, etc. The same thing happens once the user exits the VM on Terminal.

Method II: Running Beiwe as a Service

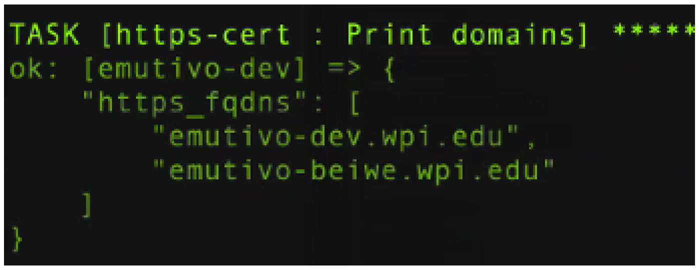
To address the problems highlighted in the previous section, Dr. Ermal Toto helped us to set up Flask as a service on the Ubuntu-based emutivo-dev virtual machine. Running Flask as a service allows the Beiwe Flask app to remain uninterrupted so progress won't be lost (Problem II). Additionally, he certified a new website URL – emutivo-beiwe.wpi.edu – to both signify the change to a production server and also to run the service on a URL that doesn't include a port number (Problem I). In order to accomplish all of this, Dr. Toto followed this tutorial by Digital Ocean and this tutorial by Miguel Grinberg.

1. In Terminal, log into your virtual machine manager account (arc-manager) and add emutivo-beiwe.wpi.edu as a certified URL under the emutivo-dev VM.
   a. cd /etc/ansible
   b. Open the https.yml file
   c. Head to where it lists 'emutivo-dev' as a VM and add 'emutivo-beiwe' as an extra URL



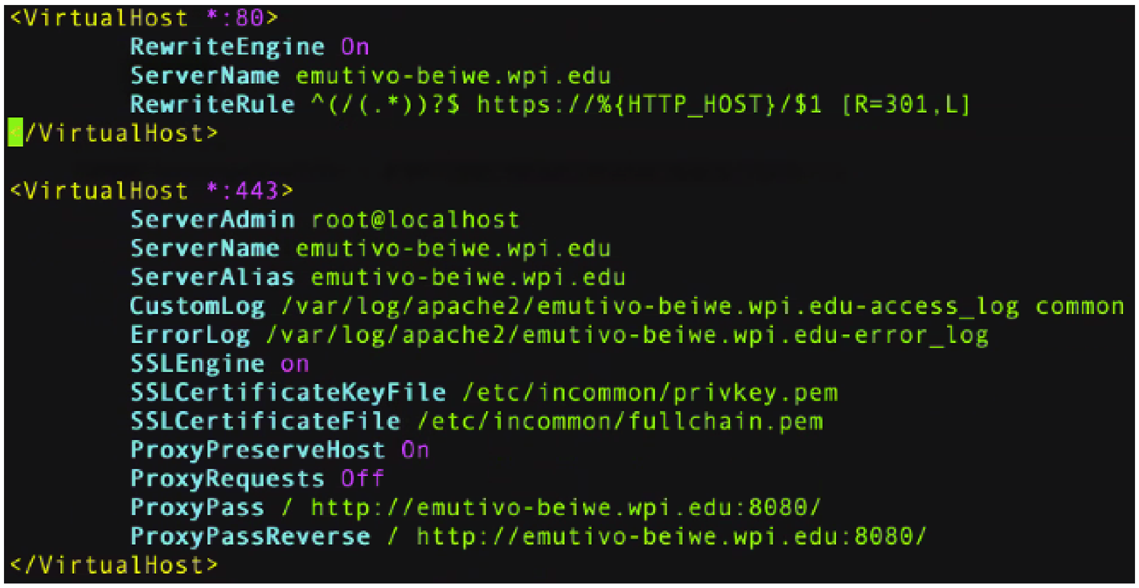
```
emutivo-dev https_extra_names='["emutivo-beiwe"]'
```

2. Create a certification for the emutivo-beiwe URL
   a. ansible-playbook https:yml -l emutivo-dev
   b. The output from that command should show something that looks like this:



```
TASK [https-cert : Print domains] *****
ok: [emutivo-dev] => {
    "https_fqdns": [
        "emutivo-dev.wpi.edu",
        "emutivo-beiwe.wpi.edu"
    ]
}
```

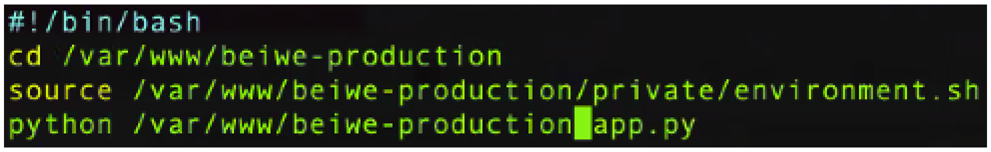
3. Next, enable the emutivo-dev Apache server to run on emutivo-beiwe.wpi.edu.
   a. Log into your account on the emutivo-dev VM (see 'I. Connecting to the Development VM' for a refresher).
   b. cd /etc/apache2/sites-enabled/
   c. Edit the 000-default.conf file so that it contains the following:



```
<VirtualHost *:80>
        RewriteEngine On
        ServerName emutivo-beiwe.wpi.edu
        RewriteRule ^(/(.*))?$ https://%{HTTP_HOST}/$1 [R=301,L]
</VirtualHost>

<VirtualHost *:443>
        ServerAdmin root@localhost
        ServerName emutivo-beiwe.wpi.edu
        ServerAlias emutivo-beiwe.wpi.edu
        CustomLog /var/log/apache2/emutivo-beiwe.wpi.edu-access_log common
        ErrorLog /var/log/apache2/emutivo-beiwe.wpi.edu-error_log
        SSLEngine on
        SSLCertificateKeyFile /etc/incommon/privkey.pem
        SSLCertificateFile /etc/incommon/fullchain.pem
        ProxyPreserveHost On
        ProxyRequests Off
        ProxyPass / http://emutivo-beiwe.wpi.edu:8080/
        ProxyPassReverse / http://emutivo-beiwe.wpi.edu:8080/
</VirtualHost>
```

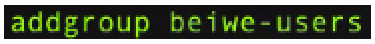
   d. Enable Apache to use mod_proxy on the VM by following this tutorial by Digital Ocean.
   e. Head to emutivo-beiwe.wpi.edu. It should be hosting the study management portal!

4. Do some housekeeping.
   a. Rename the /var/www/emutivo-dev folder as /var/www/beiwe-production.
   b. Update the run.sh file in /var/www/beiwe-production



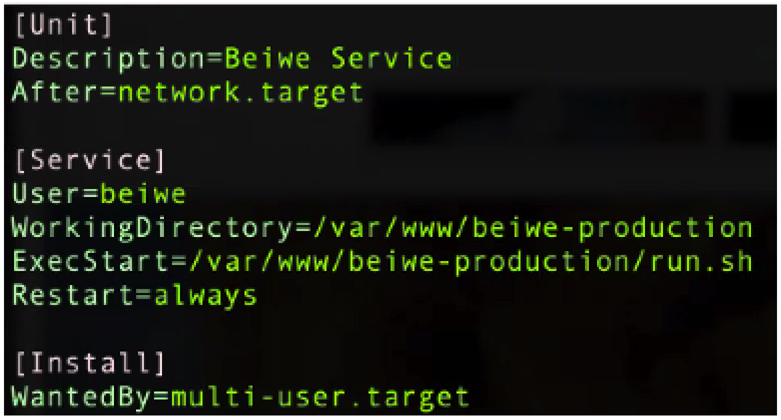
```
#!/bin/bash
cd /var/www/beiwe-production
source /var/www/beiwe-production/private/environment.sh
python /var/www/beiwe-production/app.py
```

   c. Add a new group called beiwe-users then add any users working on the Beiwe project to that group



```
addgroup beiwe-users
```

5. Now, set up the Flask app as a service so that it can run uninterrupted.
   a. Head to this tutorial by Miguel Grinberg for running Flask as a Debian service.
   b. cd /etc/systemd/system/
   c. Edit the beiwe.service file



```
[Unit]
Description=Beiwe Service
After=network.target

[Service]
User=beiwe
WorkingDirectory=/var/www/beiwe-production
ExecStart=/var/www/beiwe-production/run.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

d. Start and stop the service to make sure that it works by starting and crashing `emutivo-beiwe.wpi.edu`
   i. `systemctl daemon-reload`
   ii. `systemctl start beiwe`
   iii. `systemctl stop beiwe`

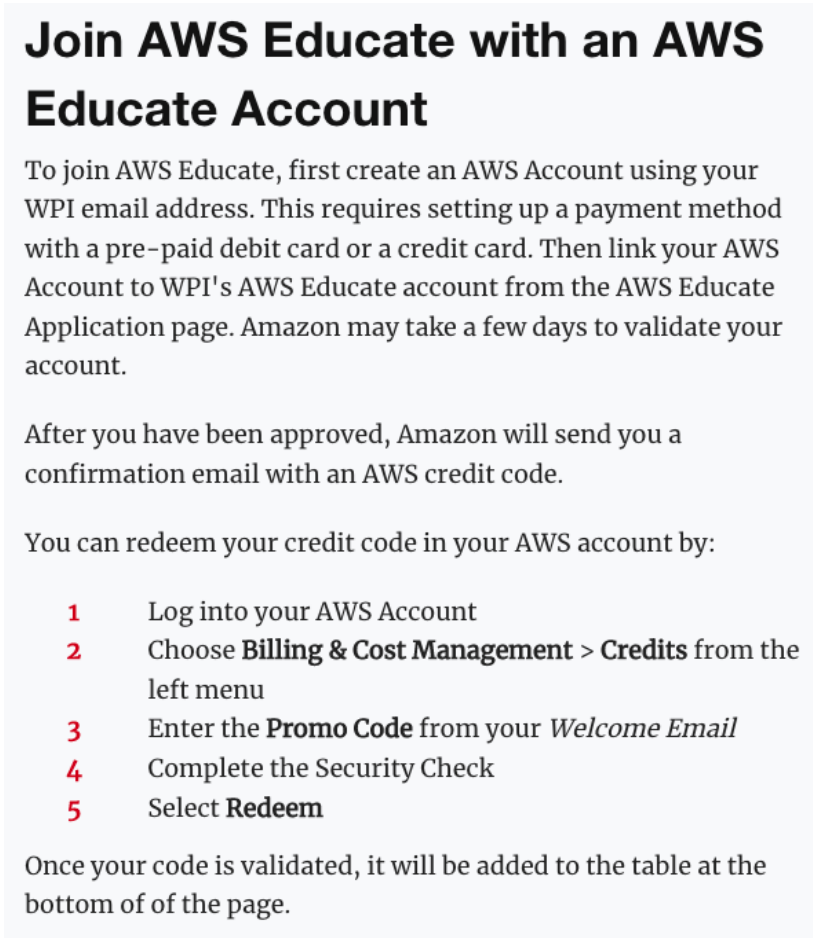
e. Enable the service and reboot the system. Head to `emutivo-beiwe.wpi.edu`. It should still be hosting the study management portal!
   i. `systemctl enable beiwe.service`
   ii. `reboot`

6. Finally, make `emutivo-bewe.wpi.edu` to a public IP so that it is accessible outside of WPI. Now, users won't have to turn on GlobalProtect in order to access it!

III. Connecting to AWS S3

Create an AWS Educate Account

1. Follow this article by WPI Hub on how to set up an AWS Educate account and redeem credits.



Join AWS Educate with an AWS Educate Account

To join AWS Educate, first create an AWS Account using your WPI email address. This requires setting up a payment method with a pre-paid debit card or a credit card. Then link your AWS Account to WPI's AWS Educate account from the AWS Educate Application page. Amazon may take a few days to validate your account.

After you have been approved, Amazon will send you a confirmation email with an AWS credit code.

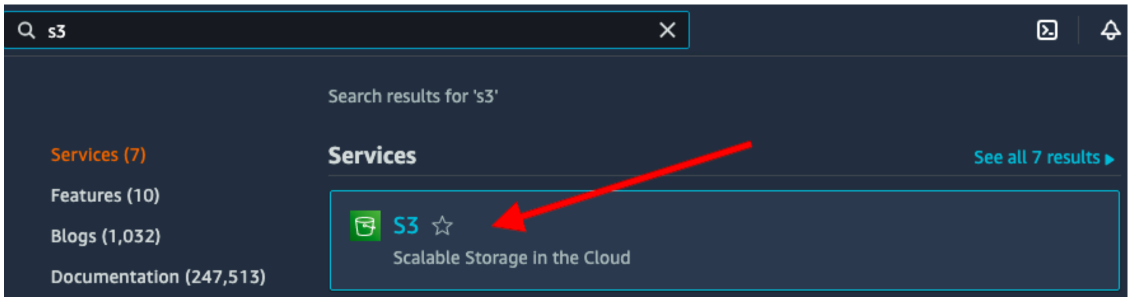
You can redeem your credit code in your AWS account by:

1. Log into your AWS Account
2. Choose **Billing & Cost Management** > **Credits** from the left menu
3. Enter the **Promo Code** from your *Welcome Email*
4. Complete the Security Check
5. Select **Redeem**

Once your code is validated, it will be added to the table at the bottom of of the page.

Create an S3 Bucket

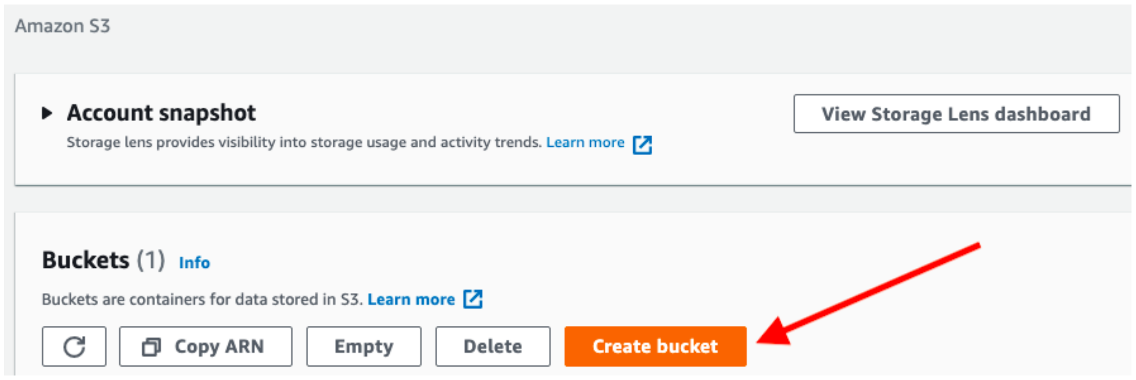
1. Head to https://aws.amazon.com/ and log into the account you created with your WPI email.
2. Search for “S3” and click on the “S3” service.



s3

Search results for 's3'

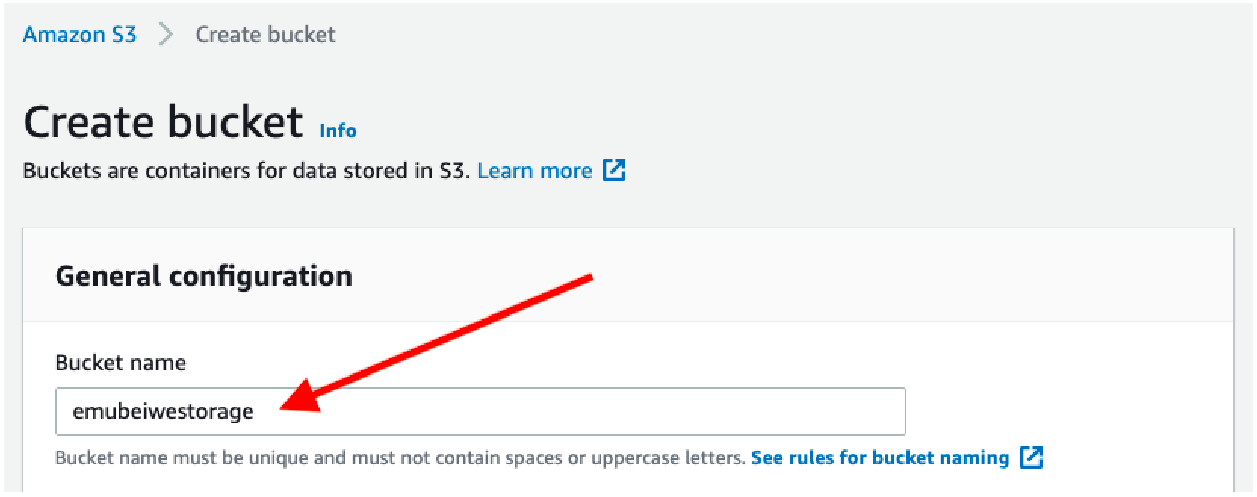
3. Click “Create bucket”



Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets are containers for data stored in S3. Learn more

4. Choose a name for your bucket, scroll down to the end of the page, and click “Create bucket”.



Buckets are containers for data stored in S3. Learn more

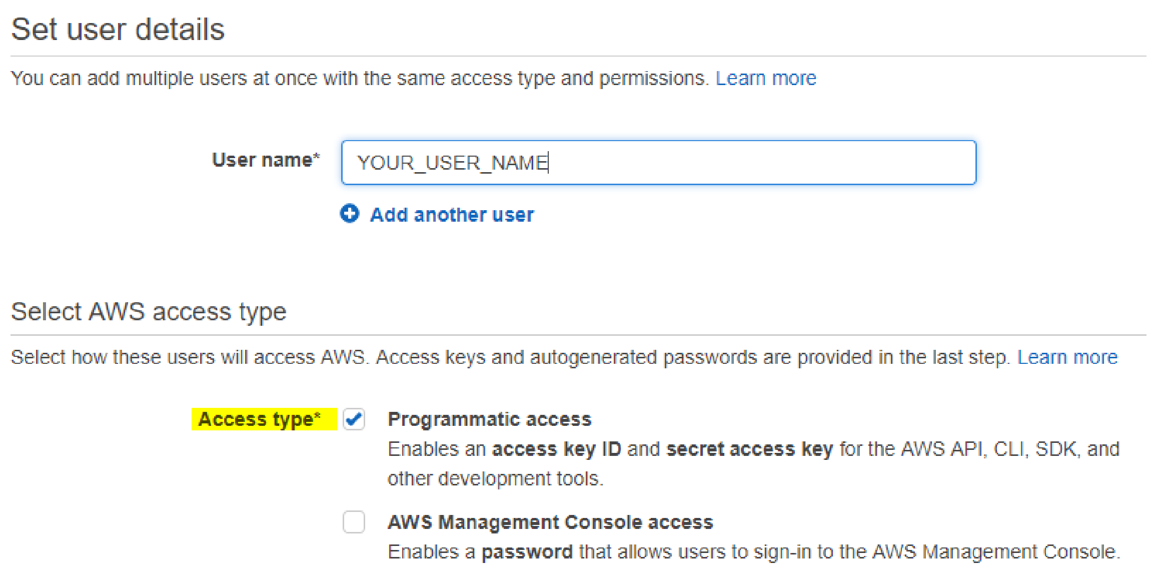
Bucket name: emubeiwestorage

Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming

Create an IAM User

Steps taken from the Beiwe Wiki.

1. Go to the AWS Users page, fill out your desired username, and check “Programmatic access”.


Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* YOUR_USER_NAME

Add another user

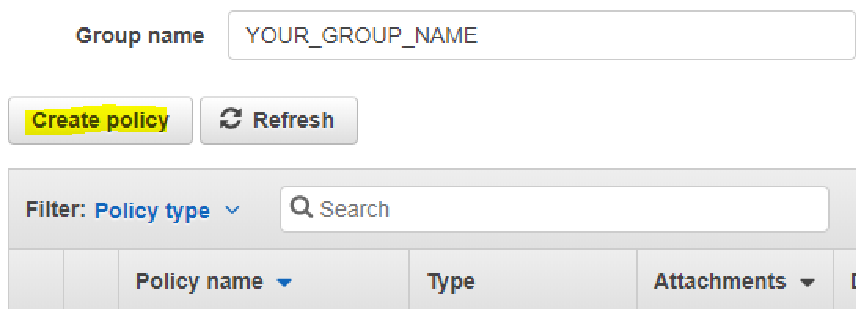
Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

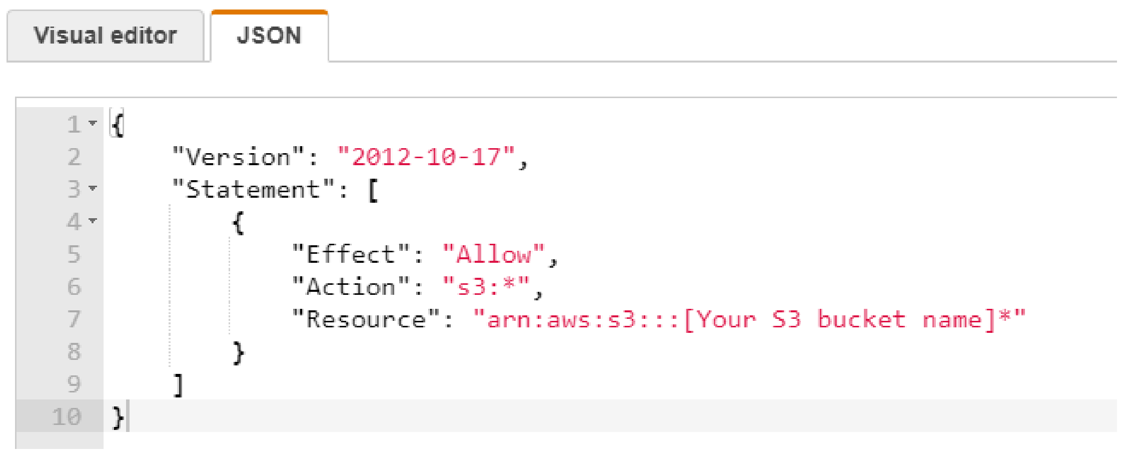
Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

2. Click “Next: Permissions” then click “Create Group”. Fill out your desired group name, then click “Create policy”.


Group name YOUR_GROUP_NAME

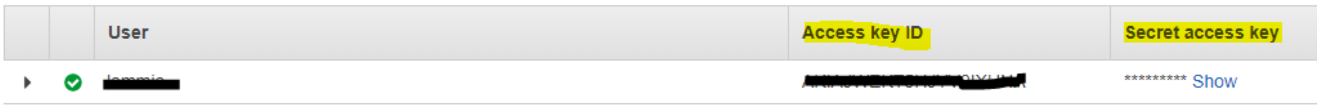
3. Click “JSON” and fill the text box with the following JSON.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::[Your S3 bucket name]*"
        }
    ]
}
```

4.

5. Finish creating the policy and group, then finish creating the user using the new group. Once you have created the user, save the “Access key ID” and “Secret access key”.

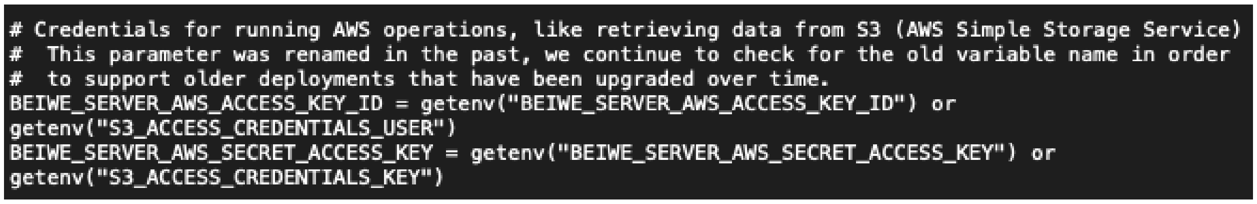


User	Access key ID	Secret access key
[redacted]	[redacted]	********* Show

Change The Config Files In The Beiwe Backend

1. Open the file /var/www/beiwe-production/config/remote_db_env.py.
2. Change the name of the S3_BUCKET variable to the name of the S3 bucket you just created.

```
os.environ['S3_BUCKET'] = 'emubeiwestorage'
```

3. Change the variable BEIWE_SERVER_AWS_ACCESS_KEY_ID to S3_ACCESS_CREDENTIALS_USER and the variable BEIWE_SERVER_AWS_SECRET_ACCESS_KEY to S3_ACCESS_CREDIENTALS_KEY.
    a. We assumed that this can be done because in the file /var/www/beiwe-production/config/settings.py, the program looks for either the BEIWE_SERVER_AWS_ACCESS_KEY_ID or the S3_ACCESS_CREDENTIALS_USER, and same for the other two variables
4. 

```
# Credentials for running AWS operations, like retrieving data from S3 (AWS Simple Storage Service)
#  This parameter was renamed in the past, we continue to check for the old variable name in order
#  to support older deployments that have been upgraded over time.
BEIWE_SERVER_AWS_ACCESS_KEY_ID = getenv("BEIWE_SERVER_AWS_ACCESS_KEY_ID") or
getenv("S3_ACCESS_CREDENTIALS_USER")
BEIWE_SERVER_AWS_SECRET_ACCESS_KEY = getenv("BEIWE_SERVER_AWS_SECRET_ACCESS_KEY") or
getenv("S3_ACCESS_CREDENTIALS_KEY")
```

5. Fill out the S3_ACCESS_CREDENTIALS_USER and S3_ACCESS_CREDIENTALS_KEY variables using the "Access key ID" and "Secret access key", respectively.
6. Restart the server using sudo service apache2 restart.

III. Using the Study Management Portal

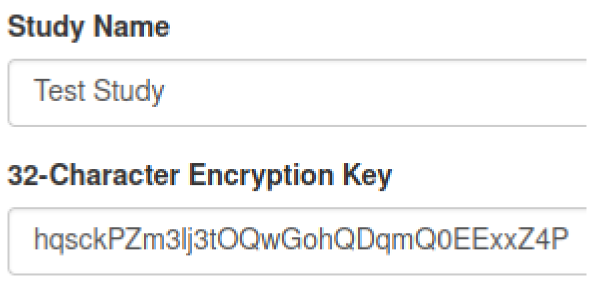
1. Get the password for logging in as admin

```
[root@arcubuntu18template:~# cat passwords
```

2. Log into the server as username "default_admin" and fill with the password

Creating a New Study

1. Head to the "Manage Studies" tab and click on the "Create New Study" button. Give a name to your study and encrypt it with a 32-character key.



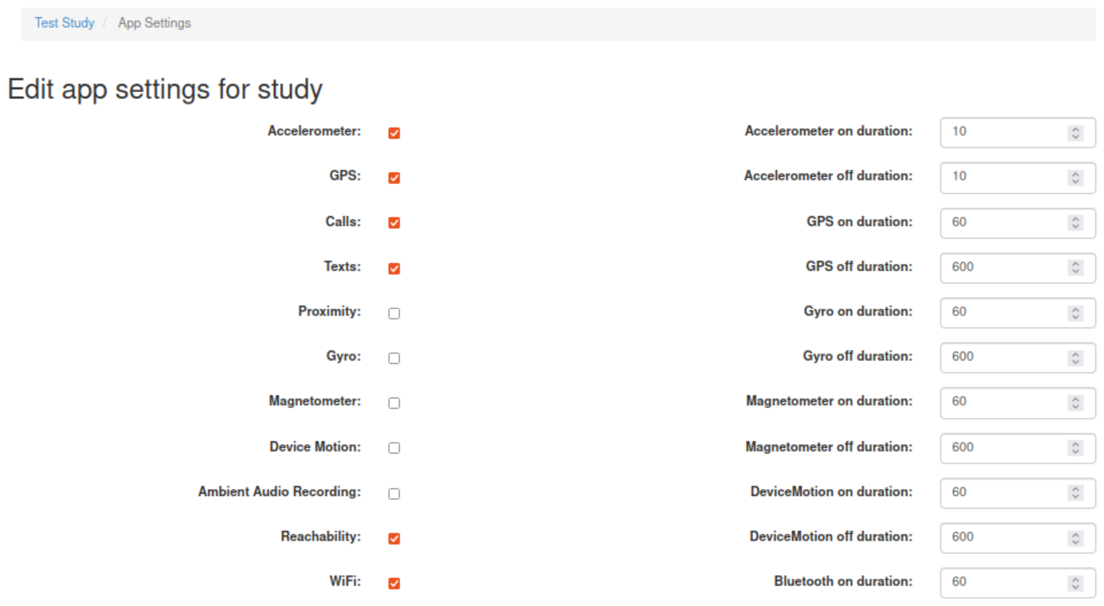
Study Name

Test Study

32-Character Encryption Key

hqsckPZm3lj3tOQwGohQDqmQ0EExxZ4P

2. After creating the study, you will be taken to a page that allows you to customize the types of data your study will collect.



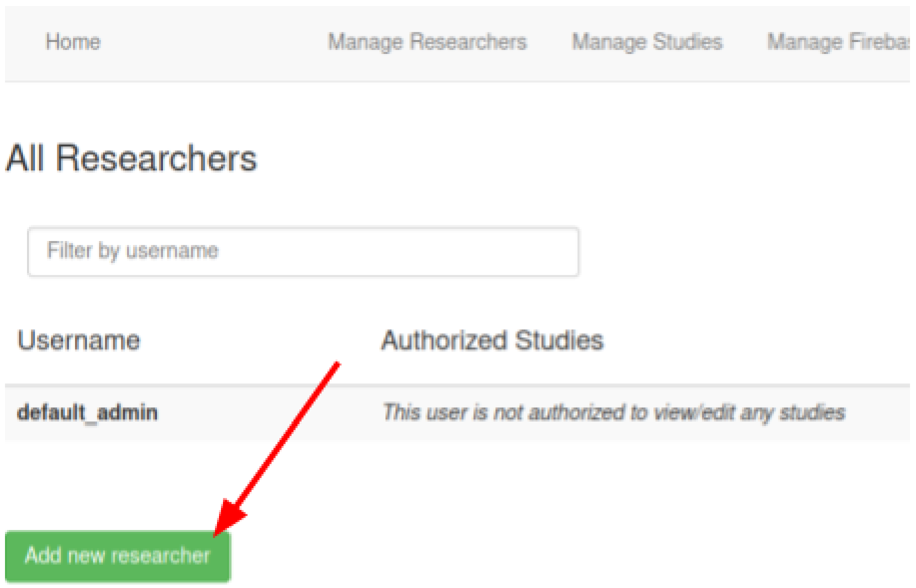
Test Study / App Settings

Edit app settings for study

Accelerometer:	✓	Accelerometer on duration:	10
GPS:	✓	Accelerometer off duration:	10
Calls:	✓	GPS on duration:	60
Texts:	✓	GPS off duration:	600
Proximity:	☐	Gyro on duration:	60
Gyro:	☐	Gyro off duration:	600
Magnetometer:	☐	Magnetometer on duration:	60
Device Motion:	☐	Magnetometer off duration:	600
Ambient Audio Recording:	☐	DeviceMotion on duration:	60
Reachability:	✓	DeviceMotion off duration:	600
WiFi:	✓	Bluetooth on duration:	60

Creating a New Researcher for the Study

1. Head to the “Manage Researchers” tab and click on the “Add new researcher” button.

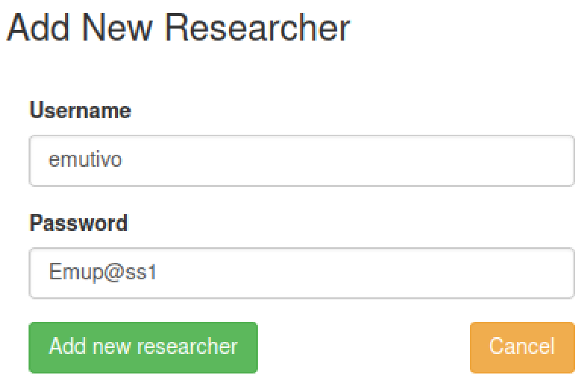


All Researchers

Filter by username

Username	Authorized Studies
default_admin	*This user is not authorized to view/edit any studies*

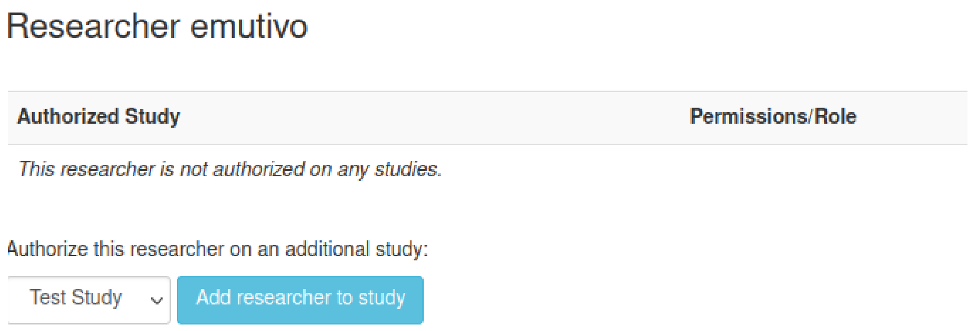
2. Choose a username and password for the new researcher, then click “Add new researcher”. An example username and pass is shown below.



Add New Researcher

Username	emutivo
Password	Emup@ss1

3. Authorize the new researcher to keep track of the study you just created by adding them to it - select a study then click “Add researcher to study”. This allows the researcher to edit the study, create new surveys, manage participants, and download the data from the study.



Researcher emutivo

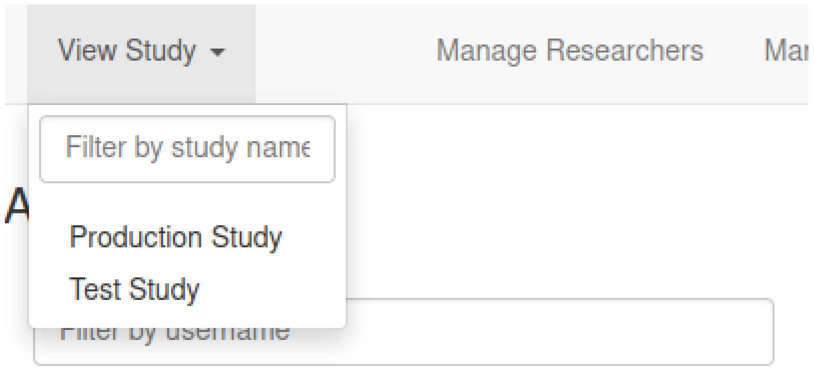
Authorized Study	Permissions/Role
This researcher is not authorized on any studies.	

Authorize this researcher on an additional study:

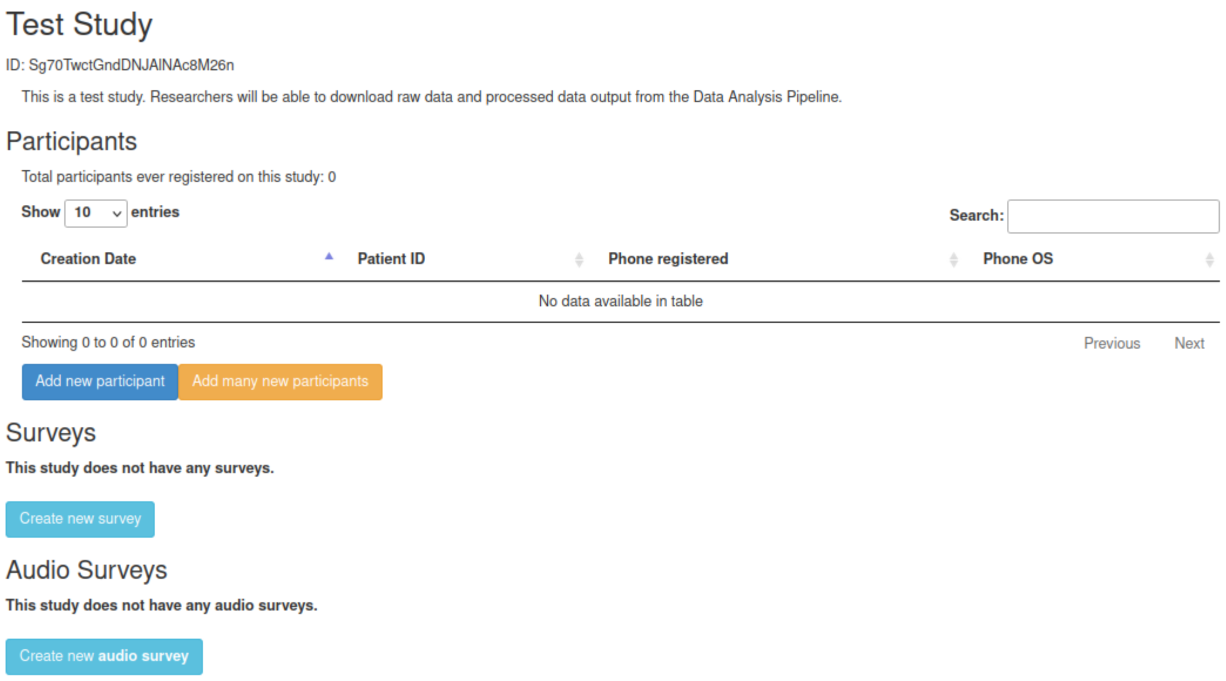
Test Study

Adding Participants to the Study

1. Head to the "View Study" dropdown list on the top menu and select a study.



2. Click on "Add new participant" to generate a unique ID for a new participant.



Test Study

ID: Sg70TwctGndDNJAlNAc8M26n

This is a test study. Researchers will be able to download raw data and processed data output from the Data Analysis Pipeline.

Participants

Total participants ever registered on this study: 0

Show 10 entries Search:

Creation Date	Patient ID	Phone registered	Phone OS
No data available in table			

Showing 0 to 0 of 0 entries Previous Next

Surveys

This study does not have any surveys.

Audio Surveys

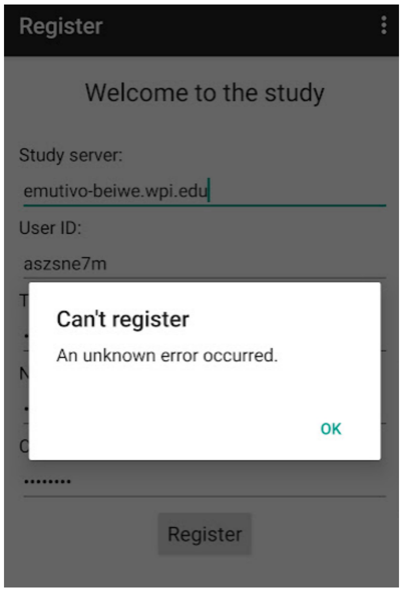
This study does not have any audio surveys.

1. Ask the new participant to download the "Beiwe2" app on their phone and give them their unique participant ID, as well as the study server. The participant can then log into the app and start using it for data collection.

IV. Running With Android Frontend

1. Follow the "Running the Beiwe Android Front-end" documentation to set up the frontend. Follow all steps before "Permission Errors" – the last step should be "Changing Server URL."
2. Scroll down to "Running the Application With Emutivo Development VM" and follow those steps.

Error: When following the steps in "Running The Beiwe Backend On The Emutivo Development VM" to set up the backend and create a new participant, the app will show the following error:



Register

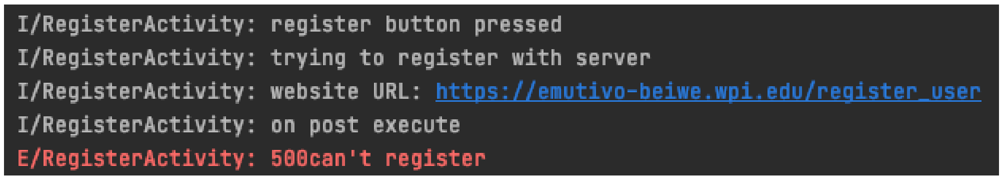
Welcome to the study

Study server:
emutivo-beiwe.wpi.edu

User ID:
aszsne7m

Can't register
An unknown error occurred.

Tracing the error through logs shows us that this is caused by a 500 error:



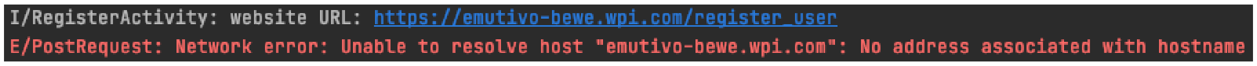
```
I/RegisterActivity: register button pressed
I/RegisterActivity: trying to register with server
I/RegisterActivity: website URL: https://emutivo-beiwe.wpi.edu/register_user
I/RegisterActivity: on post execute
E/RegisterActivity: 500can't register
```

The "can't register" is something that we printed out, not the official error.

Figuring Out Why There's An Error

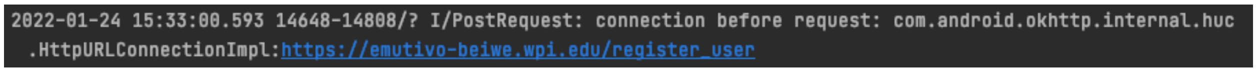
List of things the error **IS NOT** caused by:

1. Can't connect to the website / website doesn't exist
    a. Would give a 502 error and different log



```
I/RegisterActivity: website URL: https://emutivo-bewe.wpi.com/register_user
E/PostRequest: Network error: Unable to resolve host "emutivo-bewe.wpi.com": No address associated with hostname
```

2. Using a https website instead of http → we thought maybe this was the case because the app uses the function "HttpURLConnectionImp" instead of "HttpsURLConnectionImp"



```
2022-01-24 15:33:00.593 14648-14808/? I/PostRequest: connection before request: com.android.okhttp.internal.huc
  .HttpURLConnectionImpl:https://emutivo-beiwe.wpi.edu/register_user
```

    a. "HttpURLConnectionImp" actually works with both http and https websites
3. AWS S3 error
    a. We connected the application with an S3 bucket (see step II.III for directions on how to do that) but the error persisted

In conclusion: We determined that the error is likely caused by an AWS EC2 error – the Android frontend is expecting to connect to a Beiwe AMI but detects the WPI Emutivo development VM instead.

11 Appendix F: Running the Beiwe Backend on AWS

Running The Beiwe Backend On AWS

Madeline Halley (mehalley@wpi.edu) and Jyalu Wu (jwu7@wpi.edu)

Note: This is the **best** way to run the Beiwe backend. We have tried running it locally (see "Running The Beiwe Backend Locally On Linux") and running it on a development VM (see "Running The Beiwe Backend On The Emutivo Development VM") but both methods were not ideal.

I. Making an AWS Account

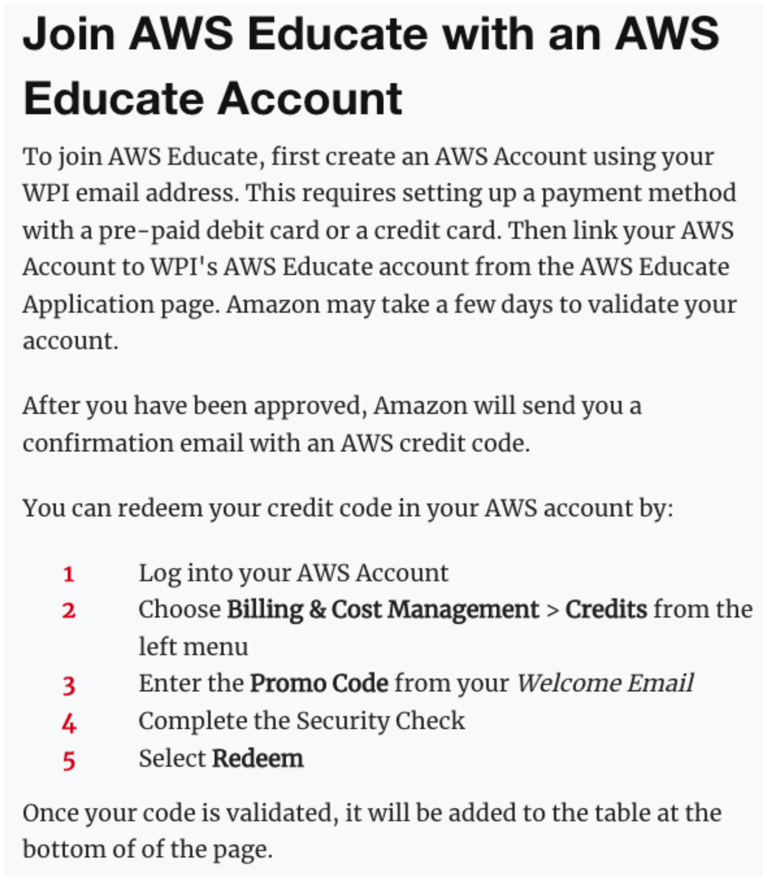
Just AWS

Head to the AWS website to create an account with your WPI email. An EC2 server that can handle no more than 20 participants a month will cost about $62/month, according to the Beiwe Github. Paying on your own will not be ideal - try to use AWS Educate or AWS Cloud Credit for Research.

Using AWS Educate

AWS Educate is a program that gives credits to faculty and students, as well as online courses for learning how to use AWS. WPI is an institutional member of AWS Educate so students will be able to receive $75-$100 worth of credits when signing up with their WPI email.

1. Follow the steps in this article from WPI Hub to set up AWS educate and redeem your credits.
    a. IMPORTANT: make sure you create an AWS account with your WPI email BEFORE registering with AWS Educate! If you don't create an AWS account first you won't be able to access the credits - they simply won't send you an email with a promo code.
    b. Our team had trouble setting up this account so maybe ask WPI IT to help you.



Join AWS Educate with an AWS Educate Account

To join AWS Educate, first create an AWS Account using your WPI email address. This requires setting up a payment method with a pre-paid debit card or a credit card. Then link your AWS Account to WPI's AWS Educate account from the AWS Educate Application page. Amazon may take a few days to validate your account.

After you have been approved, Amazon will send you a confirmation email with an AWS credit code.

You can redeem your credit code in your AWS account by:

1. Log into your AWS Account
2. Choose **Billing & Cost Management** > **Credits** from the left menu
3. Enter the **Promo Code** from your *Welcome Email*
4. Complete the Security Check
5. Select **Redeem**

Once your code is validated, it will be added to the table at the bottom of of the page.

AWS Cloud Credit for Research

Full-time faculty, research staff, and postgraduate students are eligible to apply for AWS Cloud Credit for Research. You will have to send an application to get research credits – postgraduate awards have a maximum of $5000 but faculty and staff awards don't have a maximum.

II. Deploying the Beiwe Backend

Head to the Github wiki and follow the steps on "Deployment Instructions Scalable Deployment". For a more in-depth tutorial, head to this wiki page for non-IT users – we found this to be pretty helpful!

III. Making AWS HIPAA Compliant

Since we are dealing with PHI, we will need to make AWS compliant with HIPAA.

1. Head to this article to see the FAQ page for definitions about related terms (e.g. HIPAA, HITECH, business associate addendum) and how AWS can become HIPAA-compliant.

    a. If the “Is AWS HIPAA certified?” answer leaves you confused, we were too. Head to this article by the HIPAA Journal for a no-nonsense, blunt explanation.
2. The best way to protect your AWS resources and make them HIPAA-compliant is to use Quick Start, which is based on AWS CloudFormation.

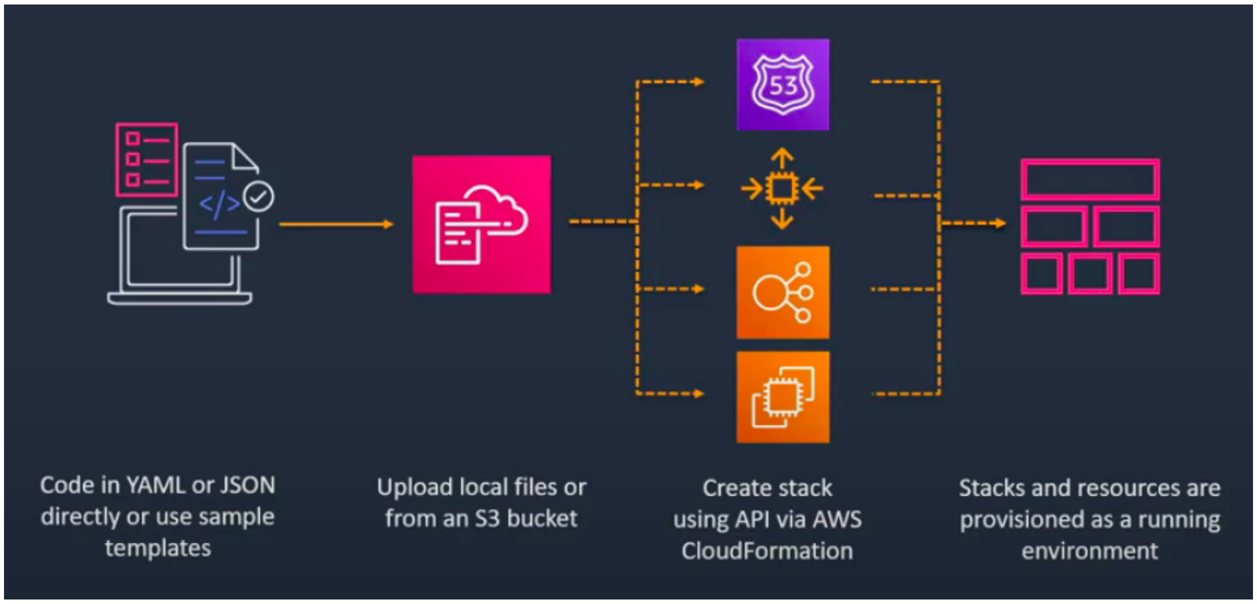
How CloudFormation Works

This is how we (the Emutivo team) think of how CloudFormation works, along with a very loose restaurant metaphor:

- CloudFormation (chef): you model your AWS resources, CloudFormation sets them up
- Template (recipe book): blueprint for AWS resources
- Stack (order): collection of resources, properties specified by template

Quick Start gives you predefined (HIPAA-compliant) templates and lets you create stacks. In this case, we would use the Quick Start templates for EC2 and S3 to deploy the Beiwe backend.

For a more in-depth explanation of Quick Start, see the Deployment Guide.


53
Code in YAML or JSON directly or use sample templates
Upload local files or from an S3 bucket
Create stack using API via AWS CloudFormation
Stacks and resources are provisioned as a running environment

References

[1] Y. M. Taye, N. F. Pingal, Y. G. Seifu, J. P. Caltabiano, M. M. Thant, and A. L. Sargent, "Mental health sensing using machine learning," 100 Institute Road, Worcester MA 01609-2280 USA, Tech. Rep., 2020.

[2] "Home," Mar 2021. [Online]. Available: https://www.beiwe.org/

[3] "Depression," 2021.

[4] "The long-term effects of not seeking treatment for depression," 2020.

[5] "Depression statistics," 2019.

[6] "Mental health by the numbers," 2021.

[7] A. A. Alhassan, E. M. Alqadhib, N. W. Taha, R. A. Alahmari, M. Salam, and A. F. Almutairi, "The relationship between addiction to smartphone usage and depression among adults: a cross sectional study," *BMC psychiatry*, vol. 18, no. 1, pp. 1–8, 2018.

[8] "Depression assessment instruments," 2019.

[9] E. O'Connor, R. Rossom, M. Henninger, H. Groom, B. Burda, J. Henderson, and E. Whitlock, "Screening for depression in adults: an updated systematic evidence review for the us preventive services task force [internet]. rockville, md: Agency for healthcare research and quality, 2016. evidence syntheses, no. 128," 2019.

[10] S. Kim and K. Lee, "Screening for depression in mobile devices using patient health questionnaire-9 (phq-9) data: A diagnostic meta-analysis via machine learning methods," *Neuropsychiatric Disease and Treatment*, vol. 17, p. 3415, 2021.

[11] "Patient health questionnaire (phq-9 phq-2)," *American Psychology Association*, 2011.

[12] A. Trifan, R. Antunes, S. Matos, and J. L. Oliveira, "Understanding depression from psycholinguistic patterns in social media texts," *Advances in Information Retrieval*, vol. 12036, p. 402, 2020.

[13] E. Toto, M. Tlachac, F. L. Stevens, and E. A. Rundensteiner, "Audio-based depression screening using sliding window sub-clip pooling," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 791–796.

[14] M. M. A. Qudar and V. Mago, "Tweetbert: A pretrained language representation model for twitter text analysis," *arXiv preprint arXiv:2010.11091*, 2020.

[15] G. Ma, "Tweets classification with bert in the field of disaster management," *Dept. Civil Eng., Stanford Univ., Stanford, CA, USA, Tech. Rep*, vol. 15785631, 2019.

[16] J. Vig, "Deconstructing bert: Distilling 6 patterns from 100 million parameters," 2018.

[17] ——, "Deconstructing bert: Part 2: Visualizing the inner workings of attention," 2019.

[18] C. Sen, T. Hartvigsen, B. Yin, X. Kong, and E. Rundensteiner, "Human attention maps for text classification: Do humans and neural networks focus on the same words?" in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* Online: Association for Computational Linguistics, July 2020, pp. 4596–4608. [Online]. Available: https://aclanthology.org/2020.acl-main.419

[19] G. L. Team. (2020) An introduction to bag of words (bow) — what is bag of words? [Online]. Available: https://www.mygreatlearning.com/blog/bag-of-words/

[20] K. P. Connaghan, J. R. Green, S. Paganoni, J. Chan, H. Weber, E. Collins, B. Richburg, M. Eshghi, J.-P. Onnela, and J. D. Berry, "Use of beiwe smartphone app to identify and track speech decline in amyotrophic lateral sclerosis (als)." in *INTERSPEECH*, 2019, pp. 4504–4508.

[21] J. Drake, K. Schulz, R. Bukowski, and K. Gaither, "Collecting and analyzing smartphone sensor data for health," in *Practice and Experience in Advanced Research Computing*, 2021, pp. 1–4.

[22] J. Assan, M. Flannery, Y. Gao, A. Resom, and Y. Wu, "Machine learning for mental health detection," 2019.

[23] (2021) What is aws?

[24] (2021) Hipaa.

[25] (2021) What is machine learning?

[26] W. by: HUSPI, "Quick introduction to machine learning algorithms for beginners," Jun 2021. [Online]. Available: https://huspi.com/blog-open/guide-to-machine-learning-algorithms/

[27] P. Pedamkar, "Machine learning algorithms: Know top 8 machine learning algorithms," Aug 2021. [Online]. Available: https://www.educba.com/machine-learning-algorithms/

[28] N. Castle, "An introduction to machine learning algorithms," Jun 2017. [Online]. Available: https://blogs.oracle.com/ai-and-datascience/post/an-introduction-to-machine-learning-algorithms

[29] "Machine learning algorithms: Microsoft azure." [Online]. Available: https://azure.microsoft.com/en-us/overview/machine-learning-algorithms/#overview

[30] R. Gandhi, "Support vector machine - introduction to machine learning algorithms," Jul 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[31] B. I. C. Education, "What is random forest?" [Online]. Available: https://www.ibm.com/cloud/learn/random-forest

[32] Z. Zhang, "Boosting algorithms explained," Aug 2019. [Online]. Available: https://towardsdatascience.com/boosting-algorithms-explained-d38f56ef3f30

[33] B. I. C. Education, "What are neural networks?" [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks

[34] M. McGregor. (2020) 8 clustering algorithms in machine learning that all data scientists should know. [Online]. Available: https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/

[35] (2020) Clustering algorithms. [Online]. Available: https://developers.google.com/machine-learning/clustering/clustering-algorithms

[36] (2022) scipy.cluster.hierarchy.linkage. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

[37] S. Narkhede. (2018) Understanding confusion matrix. [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[38] A. Mihra. (2018) Metrics to evaluate your machine learning algorithm. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

[39] C. Maklin. (2019) R squared interpretation — r squared linear regression. [Online]. Available: https://towardsdatascience.com/statistics-for-machine-learning-r-squared-explained-425ddfebf667

[40] "What is the acl and what is computational linguistics?" [Online]. Available: https://www.aclweb.org/portal/what-is-cl

[41] M. Tlachac and E. Rundensteiner, "Screening for depression with retrospectively harvested private versus public text," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 11, p. 3326–3332, 2020.

[42] E. Fast, B. Chen, and M. S. Bernstein, "Empath," *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016.

[43] E. Fast, "Empath client github." [Online]. Available: https://github.com/Ejhfast/empath-client

[44] "Explanation of bert model – nlp," 2020.

[45] R. Horev, "Bert explained: State of the art language model for nlp," 2019.

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[47] (2022) spacy 101: Everything you need to know. [Online]. Available: https://spacy.io/usage/spacy-101

[48] M. Tlachac, "Text screening github." [Online]. Available: https://github.com/mltlachac/IEEEBHI2021

[49] R. Kelly, "Pyenchant github." [Online]. Available: https://github.com/pyenchant/pyenchant

[50] K. Bird, Loper, "Wordnet github." [Online]. Available: https://github.com/nltk/nltk

[51] J. Brownlee. (2019) A gentle introduction to the bag-of-words model. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[52] S. Loria, "Textblob: Simplified text processing." [Online]. Available: https://textblob.readthedocs.io/en/dev/

[53] P. Norvig, "pyspellchecker." [Online]. Available: https://pyspellchecker.readthedocs.io/en/latest/

[54] P. Delobelle and B. Berendt, "Time to take emoji seriously: They vastly improve casual conversational models," *arXiv preprint arXiv:1910.13793*, 2019.

[55] P. Apte, "How hugging face is tackling bias in nlp," 2021.

[56] R. Kempter, V. Sintsova, C. Musat, and P. Pu, "Emotionwatch: Visualizing fine-grained emotions in event-related tweets," *Proceedings of the International AAAI Conference on Web and Social Media*. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14556

[57] A. Mignone, "Reusable flower plot." [Online]. Available: https://observablehq.com/@floatingpurr/reusable-flower-plot

[58] (2022) Linguistic features. [Online]. Available: https://spacy.io/usage/linguistic-features#vectors-similarity

[59] Onnela-Lab, "Beiwe backend wiki." [Online]. Available: https://github.com/onnela-lab/beiwe-backend/wiki

[60] (2020) Amazon web services (aws) educate.

[61] “How nature benefits mental health.” [Online]. Available: https://www.mind.org.uk/information-support/tips-for-everyday-living/nature-and-mental-health/how-nature-benefits-mental-health/