Transforming Digital Phenotyping Raw Data into Actionable Biomarkers, Quality Metrics, and Data Visualizations: An Introduction to the Cortex Software Package

James Burns, Kelly Chen, Matthew Flathers, Danielle Currey, Natalia Macrynikola, Aditya Vaidyam, Carsten Langholm, Ian Barnett, Andrew (Jin Soo) Byun, Erlend Lane, John Torous

Submitted to: Journal of Medical Internet Research
on: March 17, 2024

Table of Contents

Transforming Digital Phenotyping Raw Data into Actionable Biomarkers, Quality Metrics, and Data Visualizations: An Introduction to the Cortex Software Package

James Burns[1] BA; Kelly Chen[1] BS; Matthew Flathers[1] BS; Danielle Currey[2, 1] BS; Natalia Macrynikola[1] PhD; Aditya Vaidyam[3, 1] MS; Carsten Langholm[1] BS; Ian Barnett[4] PhD; Andrew (Jin Soo) Byun[1] MS; Erlend Lane[1] MS; John Torous[1] MD

[1]Division of Digital Psychiatry Beth Israel Deaconess Medical Center Harvard Medical School Boston US
[2]Case Western Reserve University School of Medicine, Cleveland US
[3]Carle Illinois College of Medicine Urbana US
[4]Department of Biostatistics Epidemiology, and Informatics Perelman School of Medicine at the University of Pennsylvania Philadephia US

Corresponding Author:
John Torous MD
Division of Digital Psychiatry
Beth Israel Deaconess Medical Center
Harvard Medical School
330 Brookline Ave
Boston
US

Abstract

As digital phenotyping, the capture of active and passive data from consumer devices like smartphones, becomes more common the need to properly process the data and derive replicable features from it has become paramount. Cortex is an open-source data processing pipeline for digital phenotyping data, optimized for use with the mindLAMP apps which is used by nearly 100 research teams across the world. Cortex is designed to help teams 1) assess digital phenotyping data quality in real time, 2) derive replicable clinical features from the data, 3) and enable easy to share data visualizations.
Cortex offers many options to work with digital phenotyping data, although some common approaches are likely of value to all teams using it. This paper highlights the reasoning, code, and example steps necessary to fully work with digital phenotyping data in an streamlined manner. Covering how to work with the data, assess its quality, derives features, and visualize findings, this paper is designed to offer the reader the knowledge and skills to apply towards analyzing any digital phenotyping dataset. Towards highlighting clinical applications, this paper also provides three easy to implement examples of Cortex use in real world settings. Through understanding how to work with digital phenotyping data and providing ready to deploy code with Cortex, the paper aims to show the new field of digital phenotyping can be both accessible to all yet still rigorous in methodology.

(JMIR Preprints 17/03/2024:58502)
DOI: https://doi.org/10.2196/preprints.58502

Preprint Settings

1) Would you like to publish your submitted manuscript as preprint?
Please make my preprint PDF available to anyone at any time (recommended).
Please make my preprint PDF available only to logged-in users; I understand that my title and abstract will remain visible to all users.
✓ **Only make the preprint title and abstract visible.**
No, I do not wish to publish my submitted manuscript as a preprint.
2) If accepted for publication in a JMIR journal, would you like the PDF to be visible to the public?
✓ **Yes, please make my accepted manuscript PDF available to anyone at any time (Recommended).**
Yes, but please make my accepted manuscript PDF available only to logged-in users; I understand that the title and abstract will remain v
Yes, but only make the title and abstract visible (see Important note, above). I understand that if I later pay to participate in <a href="http

Original Manuscript

Transforming Digital Phenotyping Raw Data into Actionable Biomarkers, Quality Metrics, and Data Visualizations: An Introduction to the Cortex Software Package

James Burns[1], Matt Flathers[1], Kelly Chen[1], Danielle Currey[1,2], Erlend Lane[1], Carsten Langholm[1], Aditya Vaidyam[1,3], Ian Barnett[4], Andrew (Jin Soo) Byun[1], Natalia Macrynikola[1], John Torous[1*]

1. Division of Digital Psychiatry, Beth Israel Deaconess Medical Center, Harvard Medical School, Boston, MA
2. Case Western Reserve University School of Medicine, Cleveland, OH, United States.
3. Carle Illinois College of Medicine, Urbana, IL, United States
4. Department of Biostatistics, Epidemiology, and Informatics Perelman School of Medicine at the University of Pennsylvania, Philadelphia, PA, United States

*

Department of Psychiatry Beth Israel Deaconess Medical Center Harvard Medical School
330 Brookline Ave Boston, MA, 02446 United States
Phone: 1 6176676700
Email: jtorous@bidmc.harvard.edu

Keywords: Digital Phenotyping, Mental Health, Data Visualization, Data Analysis, Smartphones

Abstract

As digital phenotyping, the capture of active and passive data from consumer devices like smartphones, becomes more common, the need to properly process the data and derive replicable features from it has become paramount. Cortex is an open-source data processing pipeline for digital phenotyping data, optimized for use with the mindLAMP apps which is used by nearly 100 research teams across the world. Cortex is designed to help teams 1) assess digital phenotyping data quality in real time, 2) derive replicable clinical features from the data, 3) and enable easy to share data visualizations.

Cortex offers many options to work with digital phenotyping data, although some common approaches are likely of value to all teams using it. This paper highlights the reasoning, code, and example steps necessary to fully work with digital phenotyping data in an streamlined manner. Covering how to work with the data, assess its quality, derive features, and visualize findings, this paper is designed to offer the reader the knowledge and skills to apply towards analyzing any digital phenotyping dataset. More specifically, the paper will teach the reader the ins and outs of the Cortex Python package. This includes background information on its interaction with the mindLAMP platform, some basic commands to learn what data can be pulled and how, and more advanced usage of the package mixed with basic Python with the goal of creating a correlation matrix. After the tutorial, different use cases of Cortex are discussed, along with limitations.

Towards highlighting clinical applications, this paper also provides three easy to implement examples of Cortex use in real world settings. Through understanding how to work with digital phenotyping data and providing ready to deploy code with Cortex, the paper aims to show the new field of digital phenotyping can be both accessible to all yet still rigorous in methodology.

Introduction

The popularity of smartphone-based digital phenotyping for advancing health research has resulted in a plethora of platforms and tools. Many have lauded the potential to capture real time behavioral data in a scalable manner from patient's personal smartphones [1, 2]. But recent reviews and commentaries consistently highlight the need for improved methodological rigor, more return of results to participants, and better data sharing [3, 4, 5]. Solving these challenges is multifactorial but can be framed around issues of 1) data collection, 2) data processing, and 3) return of results.

The evolving landscape of consumer technology, and thus digital phenotyping apps, presents numerous challenges to their scientific study. These apps require constant efforts to ensure they remain secure and able to collect the data they claim to capture. The result of this dynamic environment is that many digital phenotyping apps are no longer supported and many past apps are already abandoned. Keeping pace with new updates from Apple and Google, security vulnerabilities/ patches, user interface / experience improvements, and new sensors accessible from the phone requires constant effort and financial resources. Further, the shifting tools and libraries utilized in this research has also created challenges for transparent data processing and return of results. Few digital phenotyping studies have ever been replicated because differences in feature processing procedures make the resulting outputs from these studies incomparable. Of note, in the rare cases where replication has been attempted, the results have been negative [6, 7].

Beyond supporting replicable science, better data collection and data processing can enable better data visualization, better sharing of results with participants, and meaningful use outside of research. A 2022 paper proposed the return of results as a necessary aspect of ethical digital psychiatry research [8] and a 2023 ethical framework was proposed that specially identified the need to return results in digital phenotyping in psychiatry [9]. However, neither proposal identified how this could be accomplished or provided resources to ensure data sharing is possible. The need for data visualizations extends beyond just research. A December 2023 systematic review and meta-analysis on the utility of mental health apps to augment care for psychiatric illnesses found that, while nearly two thirds of those apps reviewed captured digital phenotyping related data, less than half of these apps were even able to share data with clinicians [10]. A 2022 review on data visualization in mental health highlighted the numerous benefits of return of results for patients including enabling proactive self-management, more effective communication with clinicians, and better engagement with remote tools like apps [11].

Towards supporting more transparent and ethical research, the need for better data collection, data processing, and return of results is clear. Data processing pipelines offer a productive focus as they can support data from multiple apps and transform that data to support myriad visualizations. There are several pipelines specific to smartphone digital phenotyping data including RAPIDS [12], FOREST [13], and CORTEX [14]. While each of the platforms offers value, they also feature differences, especially around natively supporting data visualizations. Towards the goal of expanding the utility of digital phenotyping methods and enabling the return of results, we review the CORTEX data processing pipeline and provide a tutorial to encourage more replicable, ethical, and clinically relevant research.

Towards building towards the use cases of data visualization, this paper first reviews digital phenotyping data types, second explores preprocessing steps, third discusses basic use of Cortex for data quality assessments, fourth presents advanced Cortex features, and fifth provides several real-world examples of using all the steps together. Figure 1 below provides a visual overview of the paper and details of topics covered.

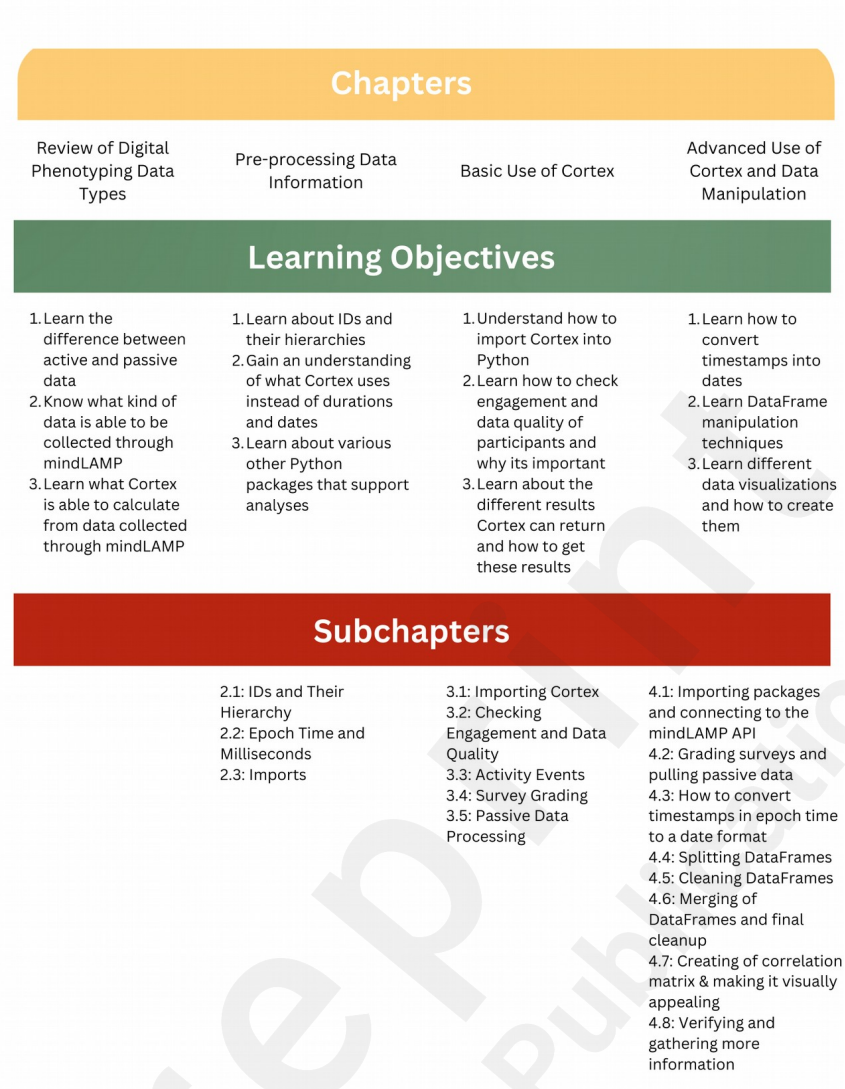

Figure 1: *Paper Schematic*

The goal of this paper is to facilitate an understanding of the steps, assumptions, and limitations around processing digital phenotyping data and transforming it into data visualizations. The steps outlined here correspond to interactive examples and we highlight in each step the ability to customize, expand, or alter the procedures to ensure the process can serve a range of outcomes from simple reporting of survey scores to more complex time series results. To facilitate these examples, we draw from the open-source mindLAMP application which has which is used by nearly 100 research teams across the world [15-17]

1) Review of Digital Phenotyping Data Types

Different types of digital phenotyping data require different approaches towards labeling, storing, and processing. In digital phenotyping and with mindLAMP, data is classified as either active or passive. Active data includes surveys and cognitive assessments. These data streams are customizable by the researchers/clinician and require the patient/user to actively engage (i.e. take a survey) for data to be collected. mindLAMP can be deployed solely as a survey tool, making it possible to use the platform for EMA research alone. All survey and cognitive data is stored within a secure server with no personal identifiable information, only the participant's user id is attached for identification [19].

On the other hand, passive data is collected using the phone's native sensors enabled by the participant and also specified by the researcher. These sensors commonly include an accelerometer and GPS which are both assigned to collect at a frequency of up to 5Hz and 1Hz respectively. The passive data collected through sensors can be customized by study to utilize as few or as many sensors as researchers desire with options ranging from screen time to Bluetooth connections. Table 1 offers a more comprehensive list of options at the time of this writing. It is also important to note that these sensor outputs are also referred to as raw data. They are what Cortex uses to calculate more actionable metrics.

Name	SensorSpec	Sampling Type	Requires watch / other device
Analytics	lamp.analytics	Continuous	No
Location	lamp.gps	Discrete	No
Accelerometer	lamp.accelerometer	Discrete	No
Device Motion	lamp.device_motion	Discrete	No
Screen	lamp.device_state	Continuous	No
Pedometer	lamp.steps	Interval	No
Bluetooth & WiFi	lamp.nearby_device	Discrete	No
Calls & Texts	lamp.telephony	Continuous	No
Sleep	lamp.sleep	Continuous	Yes
Workouts	lamp.segment	Continuous	Yes
Activity Recognition	lamp.activity_recognition	Continuous	Yes
Nutrition	lamp.nutrition	Continuous	Yes

Blood Glucose	lamp.blood_glucose	Continuous	Yes
Oxygen Saturation	lamp.oxygen_saturation	Continuous	Yes
Body Temperature	lamp.body_temperature	Continuous	Yes
Blood Pressure	lamp.blood_pressure	Continuous	Yes
Heart Rate	lamp.heart_rate	Continuous	Yes
Heart Rate Variability	lamp.heartratevariability_sdnn	Continuous	Yes
Respiratory Rate	lamp.respiratory_rate	Continuous	Yes

Table 1 *(Sensor Types): This table lists all of the sensors LAMP currently has the ability to collect and their name within mindLAMP. Sampling type represents how much data is collected within a certain time frame. If the sampling type is continuous, all data is collected the moment the sensor is activated. If the sampling type is discrete, the data is recorded every n seconds in which n is defined by a frequency parameter. If the sampling type is interval, which only relates to step count, the data describes how many steps were taken within n minutes, which is described by a frequency parameter.*

Unlike active data, passive data is captured at a high volume and velocity that is not conducive for most EMA data analysis methods. It must be processed into actionable metrics through a pipeline such as Cortex. For example, Cortex can process thousands of coordinates from the GPS phone sensor and transform them into a metric that reflects how much time a participant spent at home, appropriately named "hometime". Table 2 offers a more comprehensive list of calculated data features offered by Cortex at the time of this writing.

Secondary Feature Name	Link to documentation	Requires additional parameters for Cortex?
Call Duration	https://docs.lamp.digital/data_science/cortex/features/secondary/call_duration [24]	Yes
Call Number	https://docs.lamp.digital/data_science/cortex/features/secondary/call_number [25]	Yes
Data Quality	https://docs.lamp.digital/data_science/cortex/features/secondary/data_quality [26]	Yes
Entropy	https://docs.lamp.digital/data_science/cortex/features/secondary/entropy [27]	No
Cognitive Assessment Results	https://docs.lamp.digital/data_science/cortex/features/secondary/game_results [28]	Yes

HealthKit Sleep Duration	https://docs.lamp.digital/data_science/cortex/features/secondary/healthkit_sleep_duration [29]	Yes
Hometime	https://docs.lamp.digital/data_science/cortex/features/secondary/hometime [30]	No
Inactive Duration	https://docs.lamp.digital/data_science/cortex/features/secondary/inactive_duration [31]	No
Nearby Device Count	https://docs.lamp.digital/data_science/cortex/features/secondary/nearby_device_count [32]	No
Screen Duration	https://docs.lamp.digital/data_science/cortex/features/secondary/screen_duration [33]	No
Step Count	https://docs.lamp.digital/data_science/cortex/features/secondary/step_count [34]	Yes
Survey Results	https://docs.lamp.digital/data_science/cortex/features/secondary/survey_results [35]	Yes
Trip Distance	https://docs.lamp.digital/data_science/cortex/features/secondary/trip_distance [36]	No
Trip Duration	https://docs.lamp.digital/data_science/cortex/features/secondary/trip_duration [37]	No
Call Degree	https://docs.lamp.digital/data_science/cortex/features/secondary/call_degree [38]	No

Table 2 *(Secondary Features): This table lists all of the secondary features currently offered by Cortex within the "Secondary Feature Name" column. It also provides a link to the documentation which offers a description and an example (per link) on how to pull values using cortex.secondary.feature_name.feature_name(). More information regarding the command can be found in chapter 3.5. The final column, "Requires additional parameters for cortex?" has a boolean (True or False) outcome that represents if Cortex needs any additional parameters beyond start, end, and resolution to run either cortex.secondary.feature_name.feature_name() or cortex.run().*

2) Pre-Data Processing Information

The design and function of Cortex is motivated by the volume of data, particularly passive data, that a single user can generate in two weeks. Assuming a user captures GPS and accelerometer data from the smartphone, each at a rate of 5Hz, there will be just over 1.2 million data points gathered within 14 days. While lower sampling frequencies are possible, the ability to capture additional sensors and data over months or even years illustrates the size of the dataset in question. Cortex is designed to work flexibly with this data. This section reviews its foundations and assumptions to provide key background information to enable easy and effective use of Cortex for broad applications including data processing and data visualization.

2.1) IDs and Their Hierarchy:

Working with the data requires being able to identify users and studies. For every participant that downloads the mindLAMP app, a researcher needs to create a user profile for them in the mindLAMP dashboard. Upon creation of a user profile, mindLAMP will automatically generate a unique ID specific to the patient and the investigator account (researcher ID). The user ID is given in the format of a U followed by 10 digits (e.g. ”U0123456789”). These IDs are not to be confused with the researcher ID, which is provided in a 20 character string format of numbers and letters (e.g. ”123456abcdef78g912hi”). Within the investigator mindLAMP account, there can also exist multiple studies (groups). The studies themselves also have a unique ID, following the same format as the researcher ID.

The utility of having multiple studies under a singular researcher ID is to allow for multiple versions (e.g. A/B testing, all sensors active / no sensors active, feedback shared in the app / no feedback shared in the app) of a larger investigation (see Figure 2). This can be useful when assessing the impact of features or when seeking to offer participants multiple ways to partake, especially around how much passive data they wish to share. It can also enable different schedules of assessments or app reminders for participants as set by the research team.

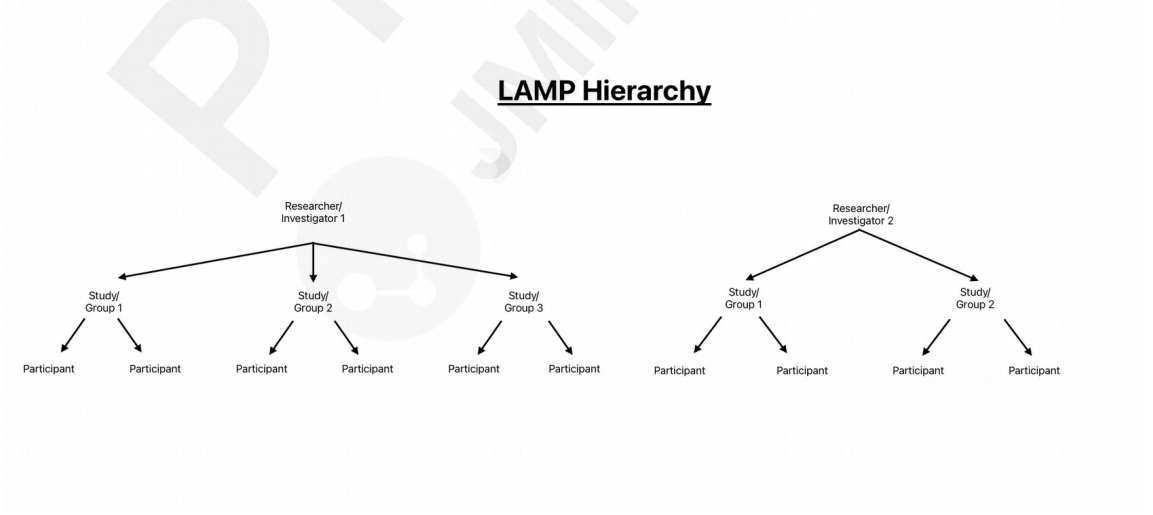

Figure 2 *(Lamp Hierarchy): This figure shows the hierarchy within LAMP. Researchers, also known as investigators, are at the top. Each researcher/investigator can have multiple different studies, and within each of those studies are individual participants.*

2.2) Epoch Time and Milliseconds:

Instead of dates, which can have multiple formats, Cortex uses a modified version of Epoch time. Epoch time is the number of seconds that have passed since January 1st, 1970 12:00 a.m. Coordinated Universal Time (UTC) to a specified date and time. The modification Cortex uses is that it uses the number of milliseconds instead of seconds. To provide an example, December 1st, 2023, 12:00 a.m. is 1701450000000 in the modified Epoch time. In addition to representing date values in modified Epoch time, all other time-related fields (e.g. time duration) will also use this format. For the remainder of the paper, when Epoch time is mentioned, it will be the modified version.

For the sake of convenience to the user, this paper will provide resources to turn dates into Epoch time and helpful variables to change milliseconds to more comprehensive units, like seconds or days, below. There also exist many different Epoch time converters on the internet that can assist with conversions. One such converter is: https://www.epochconverter.com/. [51]

Dates can be converted into Epoch time using a Python function combined with various Python packages, such as “pytz” and “datetime”.

```
import pytz
from datetime import datetime, timezone

def timestamp(dt):
    local = pytz.timezone("America/New_York")
    date = datetime.strptime(dt, '%m/%d/%Y')
    local_dt = local.localize(date, is_dst=None)
    utc_dt = local_dt.astimezone(pytz.utc)
    utc_dt.replace(tzinfo=timezone.utc).timestamp() * 1000
  return int(utc_dt.replace(tzinfo=timezone.utc).timestamp() * 1000)
```

This function takes a date in the form of MM/DD/YYYY in a string and converts it into Epoch time. An example of this function in use can be seen below.

```
timestamp('01/01/2001')
978325200000
```

Cortex has a useful command that gives the epoch time, when the command is run. This command will print the Epoch time if it is not saved into a variable.

```
cortex.now()
```

Finally, it is good practice to keep variable names that allow for easy conversions of milliseconds such as the variables below.

```
MS_IN_DAY = 24 * 3600 * 1000
MS_IN_HOUR = 3600 * 1000
```

2.3) Imports:

Cortex is designed to provide users with meaningful and understandable metrics. When combined with various Python packages, it is even more powerful for data processing. Below in table 3 are common Python packages that are often useful to run with Cortex. These are the packages that will be used in the rest of the paper and are useful to explore if planning to run Cortex. There are many more Python packages that can be used and the table above should not be viewed as a limiting factor on what can be done with data processed by Cortex.

Package Name	Package Usability	Documentation Link
Pandas	Allows for data manipulation and simple analysis within DataFrames	https://pandas.pydata.org/about/ [39]
Numpy	Allows for scientific computations in python	https://numpy.org/doc/stable/user/whatisnumpy.html [40]
Datetime	Allows for manipulation of dates and times	https://docs.python.org/3/library/datetime.html [41]
Pytz	Allows cross platform timezone calculations	https://pypi.org/project/pytz/ [42]
Seaborn	Allows for data visualizations	https://seaborn.pydata.org/ [43]
Matplotlib	Library for creating visualizations in python	https://pypi.org/project/matplotlib/ [44]
Calplot	Used to create heatmaps per calendar year by plotting time series data	https://pypi.org/project/calplot/ [45]
Altair	Visualization Library for python	https://altair-viz.github.io/ [46]
Plotly	Python Graphing Library that makes interactive visualizations	https://plotly.com/python/ [47]

Table 3 *(Commonly Used Python Packages): Common Python Packages used to help manage and support data manipulation and analysis.*

3) Basic Use of Cortex:

Building on the previous sections, this section explores how to import Cortex, score active data surveys, pull active data, pull passive data, and use several miscellaneous Cortex commands. This

section discusses issues related to poor data quality (i.e. low engagement from the user, errors in data collection) and how to use these commands to easily assess and troubleshoot both active and passive digital phenotyping data quality issues.

3.1) Importing Cortex:

Cortex is a Python package that uses the mindLAMP API to collect the data from various users. In order to use its API, the user must have an access key provided by the mindLAMP consortium.

```
!pip install LAMP-core

import LAMP
api = 'exampleapi.api'
email = 'exampleemail@gmail.com'
password = 'password'
lamp.connect(email, password, api)
import cortex
```

3.2) Checking Engagement and Data Quality:

Participant engagement is critical to any study and especially for digital phenotyping where the goal is to collect active and passive data from their smartphones. A lack of active participant engagement limits the amount of data that can be pulled from LAMP and missing data will reduce the quality of analysis that can be performed [20]. Engagement thus encapsulates: 1) A participant's active participation in taking surveys and cognitive assessments and 2) A participant's passive data quality, a ratio of actual vs expected smartphone signal (passive) data captured each day. Due to this importance, it is best practice to check engagement and data quality metrics before proceeding with other analyses. Prior research suggests that active engagement is linked to passive data quality as if the app is not actively opened and used for several days, the phone may ignore the app's request to capture high frequency passive data [20].

To assess participant's engagement with active data, it is useful to evaluate how many surveys or cognitive assessments have been completed in a certain time frame. Cortex supports such assessments with the command below.

```
cortex.visualizations.participant.active(id)
```

This command displays a graph of the activities that the participant completed with the date of completion as the x-axis (see below in Figure 3). Activities seen can vary from mindfulness exercises to the surveys that the participants complete.

Given that a study can be customized to capture different active and passive data streams at different frequencies, the parameters within the command to check data quality can be changed to suit different purposes. For the scope of the paper, every parameter for every command will not be explained in detail. Instead, the most important parameters will be explained and a link to the documentation will be provided. The documentation will contain the other parameters not covered in the paper. If a documentation link is not provided, all parameters are covered in detail.

The important parameters of cortex.visualizations.participant.active are explained below:

- id_list: This parameter represents which user the researcher is generating the graph for. To generate for a single participant, one should make this parameter a python string with the user ID. This parameter also accepts a list of participant IDs, study IDs, and researcher IDs. If more than one participant ID is provided or represented within a study or researcher ID, multiple graphs will be printed.
- sample_length: This parameter represents the maximum amount of days that can be represented in the generated graph. It is possible to use an arbitrarily large number to pull every single day that a participant has completed a task. The graph will start with the first event completed.
- attach_graphs: This parameter is a boolean value (True or False) that, if true, will attach the graphs to the participant's Portal Tab in the mindLAMP app that the graph is generated for. This enables real time sharing of results with participants directly via mindLAMP and provides a simple means for return of results.

To learn about related parameters, visit the documentation link provided.

https://docs.lamp.digital/data_science/cortex/visualizations/participant_level [48]

An example of this command in use can be seen below:

```
cortex.visualizations.participant.active("U0929038949",
sample_length=100,
attach_graphs = True)
```

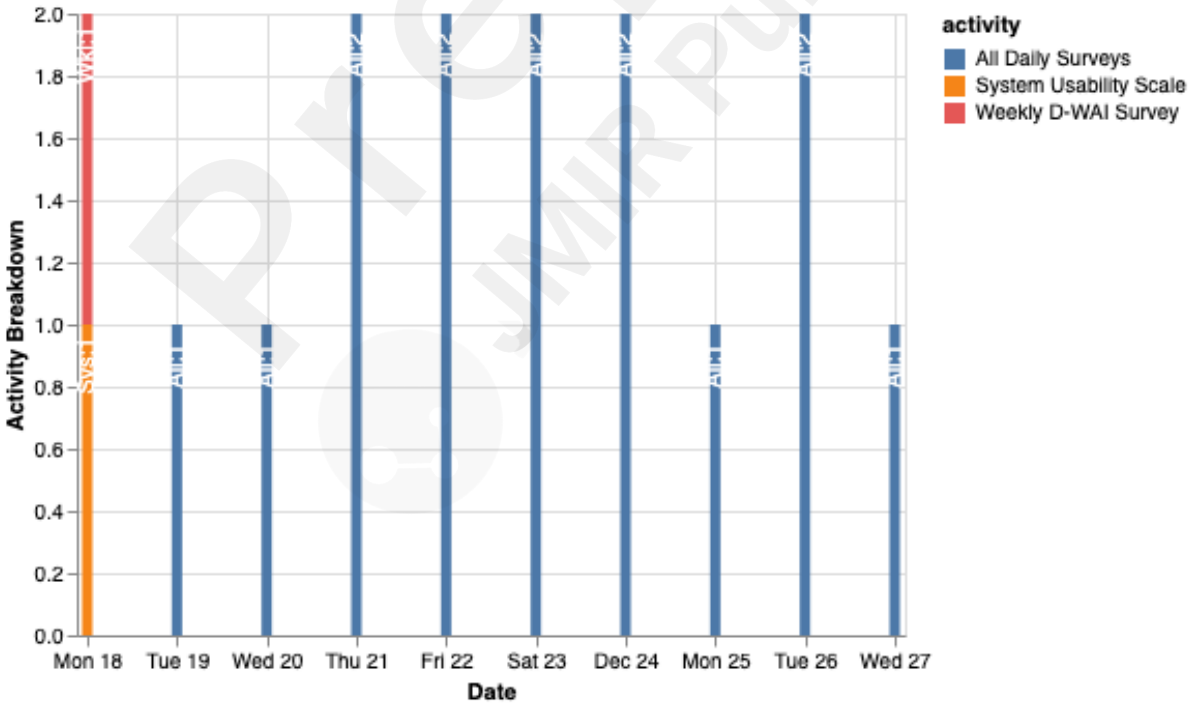

Figure 3 *(Cortex Active Data Visualization): This figure shows the activities a participant has completed through their time using mindLAMP. The x-axis shows the date of the activities completed, while the y-axis shows how many activities were completed. The color of the bar represents which activity was completed. The blue bars represent a created survey that the participant was supposed*

to take twice a day. The orange bar represents a survey that the participant completed in regards to the app itself and its ease of use. The red bar is another survey that represents responses to questions on the digital working alliance.

This participant was asked to complete two of the same daily surveys per day. These surveys were named “All Daily Surveys” in mindLAMP and are represented as blue bars as seen in the legend to the right. Looking at the graph, the participant has been active in terms of their engagement with surveys and has completed at least one survey between December 19th and December 27th. The participant completed the task of taking the survey twice on December 21st, 22nd, 23rd, 24th, and 26th.

To assess passive data quality, the command used is similar to that of assessing active data, which is shown below:

```
cortex.visualizations.participant.passive(id)
```

This command generates four graphs per participant. Two of the graphs represent the amount of data coming from the GPS sensor. The other two graphs will represent the amount of data coming from the accelerometer sensor. The graphs generated will be either a scatterplot or a heat map. Each of these two graphs will have a representation of the “ideal” amount of data being received. For the scatterplot, a horizontal red line is placed. For the ideal amount of data collected, the scatter plot should be above the line. In terms of the heat map, a blue star will represent the ideal amount of data collected. These “ideal” amounts should only be used as a reference point. The true ideal amount of data will always be up to the researchers and set by the study.

The important parameters of cortex.visualizations.participant.passive are explained below:

- id_list: This parameter represents which user the researcher is generating the graph for. To generate for a single participant, make this parameter a python string with the user ID. This parameter also accepts a list of participant IDs, study IDs, and researcher IDs. If more than one participant ID is provided or represented within a study or researcher ID, multiple graphs will be printed.
- sample_length: This parameter represents how many days the user wants to query for. Unlike the sample_length parameter in cortex.visualizations.participant.active, the user cannot use an arbitrarily large number to query for all of the data from a participant. If a user does take this approach, they will end up querying for days that a participant was not collecting data, causing the scale of the axes to be off and creating unnecessary blank space on the heat map.
- days_ago: This parameter represents where the user wants the end of the x-axis to be.
- reset: This parameter is a boolean value (True or False) that will reset any previous graphs generated for the participant. If a graph has already been generated, any new information being queried for will be added to the existing graph. If the user does not want this, use reset = True.

To find the rest of the parameters, visit the documentation link provided.

https://docs.lamp.digital/data_science/cortex/visualizations/participant_level [48]

An example of this command in use can be found below:

```
cortex.visualizations.participant.passive("U0929038949", sample_length = 7,
days_ago=27, reset=True)
```

In this example, the researcher wanted to query for the first week of the participant's time during a study. The researcher knows that the participant consented to data collection one month ago (31 days). The user also had generated a previous graph and wanted to reset the existing graph.

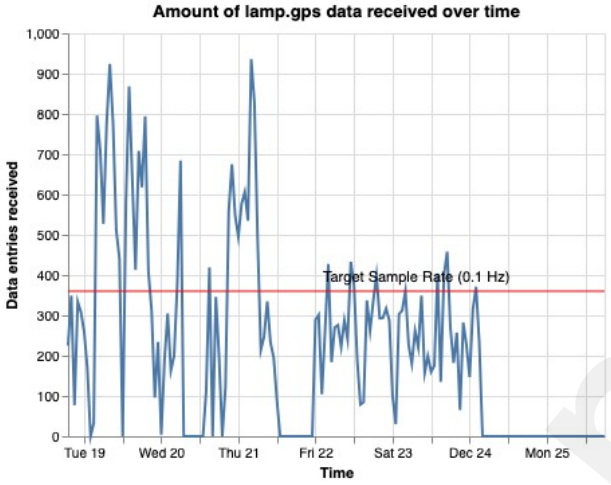

Figure 4: *This figure shows how many GPS data points are being collected over time. The x-axis represents the time of data point collection and the y-axis represents the amount. Ideally, the user would want to see the blue line never drop below the red.*

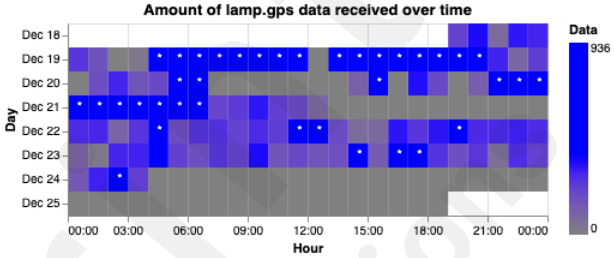

Figure 5: *This figure shows how many GPS data points are being collected over time. The beginning of the x-axis represents the very start of the day, 12:00 a.m., or 00:00 in 24 hour time. Each hour is represented as a square and given a gradient of blue with the deeper blues representing more data points collected. Each row represents one day and starts at the top. Ideally, the user would want each square to be blue with a little star within.*

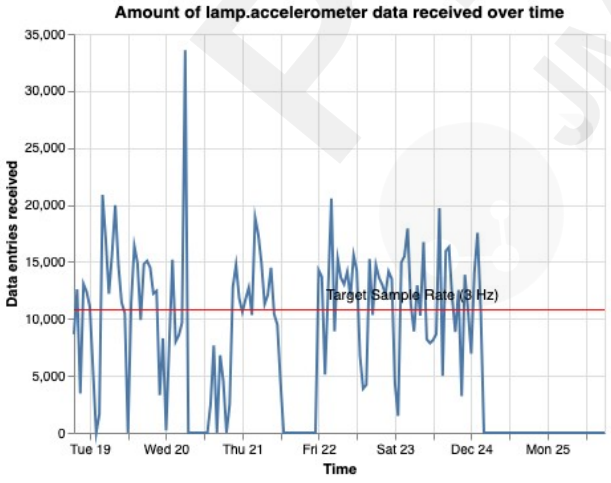

Figure 6: *This figure shows how many accelerometer data points are being collected over time. The x-axis represents the time of data point collection and the y-axis represents the amount. Ideally, the user would want to*

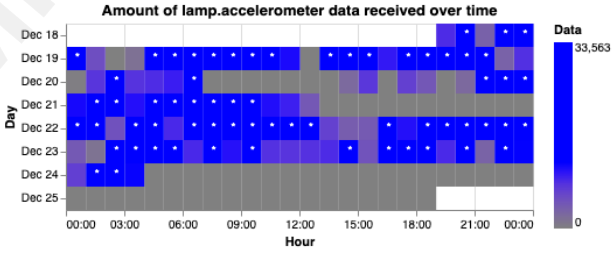

Figure 7: *This figure shows how many accelerometer data points are being collected over time. The beginning of the x-axis represents the very start of the day, 12:00 a.m., or 00:00 in 24 hour time. Each hour is represented as a square and given a gradient of blue with the deeper blues representing more data points collected. Each row represents one day and starts at the top. Ideally, the user would want each square to be blue with a little star within.*

see the blue line never drop below the red.

The examples shown in Figures 4,5,6,7 highlight that LAMP is collecting passive data of accelerometer and GPS for this single user, although on some days the data quality captured is below the target. While beyond the immediate scope of this paper, possible reasons include the phone being in low power or airplane mode, lack of active engagement (which was assessed above), or technical errors, one common error being the participant's device being on low power mode. Monitoring passive data quality is easy and can be done on a weekly or even daily basis to rapidly spot issues and help a study or patient troubleshoot.

3.3) Activity Events:

Surveys and sensors are not the only type of data that can be captured. mindLAMP also offers functionality beyond active and passive data collection to enable users to engage with psychoeducation (the L in LAMP) and complete therapeutic exercises (the M in LAMP). For some studies, this can be important or even the focus of the research question. For example, "Does the efficacy of completing mindfulness on mindLAMP vary by location or does cognitive therapy work better on nights with more sleep duration?" To accomplish this from a data perspective, the user can access information regarding everything a participant engages in within mindLAMP. Every time a participant completes an activity within mindLAMP, details about the event are recorded. These details include when the activity was started, how long it took, and specific details relating to the activity. The specific details can include the answer to a question within a survey, the results of a cognitive assessment, or the timestamp when a mindfulness exercise was completed.

A researcher or clinician can create their own activities, including custom surveys, custom psychoeducation, and custom therapeutic activities. To ensure these custom activities can be easily scored and work well with cortex, there is a standard data format that is important to understand if seeking data from any activity event. In order to view these activity events for any study, this Cortex command, which can be seen below, can be used.

```
LAMP.ActivityEvent.all_by_participant(id)["data"]
```

The command will print a Python dictionary that contains every activity a participant has completed. The dictionary will also contain information related to the activity being completed, which will be contained within ‘’temporal_slices’, a key within the dictionary (or column within a DataFrame). For example, if a participant took a survey, every question within the survey would be contained within ‘temporal_slices’’.

An example of the command in use can be found below:

```
LAMP.ActivityEvent.all_by_participant(part)["data"][0]
```

Since this command will print every activity completed, it can have a very long output. In the example, only the most recent activity completed will be printed.

```
{'temporal_slices': [{'item': 'Overall, how would I rate my anxiety today?',
                      'value': '2',
                      'duration': 3750,
                      'type': None,
                      'level': None},
                     {'item': 'I am able to manage my day-to-day life',
                      'value': '3',
                      'duration': 7116,
                      'type': None,
                      'level': None},
                     {'item': 'Overall, how would you rate your mood today?',
                      'value': '7',
                      'duration': 11154,
                      'type': None,
                      'level': None},
                     {'item': 'Approximately what time did you fall asleep last night?',
                      'value': '04:00AM',
                      'duration': 56773,
                      'type': None,
                      'level': None},
                     {'item': 'Approximately what time did you wake up this morning?',
                      'value': '07:00AM',
                      'duration': 75563,
                      'type': None,
                      'level': None},
                     {'item': 'How much physical exercise did you engage in today? ',
                      'value': '30-45 minutes',
                      'duration': 23957,
                      'type': None,
                      'level': None}],
 'duration': 178313,
 'static_data': {},
 'activity': 'ywse81hs411cdwvgcr5n',
 'timestamp': 1703646117283,
 '_parent': 'U0929038949'}
```


Going through this output, it can be seen that there are six keys within the Python dictionary, each describing something specific about the activity that was completed. Starting at the top in the ‘temporal_slices’ key, the information contained within is specific details about the activity event and actions within. In this example, since the activity event was a survey, each answer to the questions is recorded, along with how long it took to answer the question. The next key in the dictionary is ‘duration’. This is simply how long the entire activity event took. The ‘static_data’ key is empty in this activity. Static data refers to details about certain activity events, specifically the games offered within mindLAMP. These details include points scored, correct answers, attempts, etc. Moving to the next key, ‘activity’, represents the activity event that was completed. Each activity event is given a string for identification regardless of activity type e.g. one survey will have a different identification string than another. Within this example, the activity completed was ‘All Daily Surveys’. The final two keys are ‘timestamp’ and ‘_parent’, which are the time the activity was completed and who

completed the activity respectively.

3.4) Survey Grading:

Surveys, which are a common form of active data collection, can be completed on the LAMP platform and scored with Cortex. Scoring with Cortex is an efficient way of analyzing the results of the surveys and prevents mistakes caused by calculating scores by hand.

To score a survey using Cortex a Python dictionary needs to be created. These dictionaries provide Cortex with instructions on how the user wants the survey to be scored. For example, a participant may have completed a daily survey that asked them three questions. The three questions being “Overall, how would you rate your mood today?”, “Overall, how would I rate my anxiety today?”, and “How much physical exercise did you engage in today?” relating to the participants mood, anxiety, and exercise level respectively.

This scoring dictionary can be seen below with scoring parameters easily adjusted by the researcher:

```
score_dict = {'category_list': ['Daily Mood Survey', 'Daily Anxiety Survey',
'Daily                                Exercise                                Survey'],
                                                                'questions':     {
                'Overall, how would you rate your mood today?': {'category':
'Daily          Mood          Survey',          'scoring':          'M_map'},
              'Overall, how would I rate my anxiety today?': {'category': 'Daily
Anxiety                Survey',                'scoring':                'A_map'},
                  'How much physical exercise did you engage in today? ':
{'category':    'Daily    Exercise    Survey',    'scoring':    'Exercise_map'},
                                                                            },
            'A_map': {'10': 10, '9': 9, '8': 8, '7': 7, '6': 6, '5': 5, '4': 4, '3': 3, '2':
2,                '1':                1,                '0':                0},
            'M_map': {'10': 10, '9': 9, '8': 8, '7': 7, '6': 6, '5': 5, '4': 4, '3': 3, '2':
2,                '1':                1,                '0':                0},
             'Exercise_map': {'None': 0, '< 15 minutes': 1, '15-30 minutes': 2,
'30-45 minutes': 3, '45-60 minutes': 4, '60+': 5}}
```

This dictionary can be updated with any survey that has been taken, but the dictionary can only take numeric outputs, whether that be boolean (True or False) or an integer.

With a start time, end time, and scoring dictionary the user can query for scores. The scores will be labeled to match how categories are named, in this case they will be labeled “Daily Mood Survey”, “Daily Anxiety Survey”, and “Daily Exercise Survey”

The command to grade the survey can be seen below:

```
cortex.primary.survey_scores.survey_scores(id=, start=, end=,
                                           scoring_dict=score_dict)['data']
```

The important parameters of cortex.primary.survey_scores.survey_scores are explained below:

- id: This parameter represents the participant’s survey that is being graded. Unlike the cortex

visualizations, ID is limited to one participant at a time. If the user wants to grade multiple participants at once, consider a loop. Additionally, only putting the user ID without the parameter title (id=) will cause an error.

- start: This parameter represents the start of the time period the user wants to query for. (epoch time)
- end: This parameter represents the end of the time period the user wants to query for. (epoch time)
- scoring_dict: This parameter represents the scoring dictionary that was created above.
- return_ind_ques (optional): This parameter is a boolean value (True or False) that, if True, will return the individual question along with the results of the survey. Since this parameter is optional, if it is not included within the command, it is assumed by Cortex to be False.

An example of this command can be found below:

```
cortex.primary.survey_scores.survey_scores(id="U0929038949",
                                           start=1703005200000,
                                           end=1703091600000,
                                  scoring_dict=score_dict)['data']
```

In this example, the researcher wanted to query every survey taken from December 19th, 2023 to December 20th, 2023. The researcher also did not want Cortex to return the individual questions within the surveys.

```
[{'start':                                          1703021202227,
                                  'end':            1703022012667,
                  'category':      'Daily      Anxiety      Survey',
                  'question':      'Daily      Anxiety      Survey',
                                            'score':               1},
                         {'start':                      1703021202227,
                                  'end':                1703022012667,
                   'category':        'Daily        Mood        Survey',
                   'question':        'Daily        Mood        Survey',
                                              'score':              10},
                         {'start':                      1703021202227,
                                  'end':                1703022012667,
                  'category':       'Daily      Exercise      Survey',
                  'question':       'Daily      Exercise      Survey',
  'score': 5}]
```

3.5) Passive Data Processing:

Passive data streams, (eg accelerometer) are often most useful when that data can be transformed into behavioral features. Cortex has two ways of processing the raw passive data collected from the LAMP platform. The first way a user can process passive data is individually, though the format of the command below:

```
cortex.secondary.feature_name.feature_name(id=,      start=,      end=,
```

```
resolution=)
```

The important parameters of cortex.secondary.feature_name.feature_name are explained below:

- feature_name: This parameter is in the name of the command. It represents what passive data feature the user wants to query for. Reference table 2 in chapter 1 for the list of feature_names that are currently available.
- id: This parameter represents the participant that the user wants to query for.
- start: This parameter represents the start of the time period the user wants to query for. (epoch time)
- end: This parameter represents the end of the time period the user wants to query for. (epoch time)
- resolution: This parameter represents the time frame of which the passive data feature is calculated in MS. For example, if the user wanted to calculate hometime in A DAY the user would have resolution equal to the amount of milliseconds in a day.
- The parameters above are required for ALL features, but some features require more parameters. Cortex will produce an error if the required parameters are not included while executing the command. The features that require more than the parameters above can be seen in table 2, which is located within chapter 1.

An example of this command can be found below:

```
hometime = cortex.secondary.hometime.hometime(id=part,
                                              start=1702962000000,
                                              end=1703048400001,
                                              resolution=MS_IN_DAY)
```

With this command the user is looking for how much time the participant spent at home from December 19th, 2023 to December 20th, 2023, specifically 12:00 a.m. for both dates. Since the resolution is binned by the number of milliseconds in a day, Cortex will return the amount of time at home per day. If the user changed the resolution to be the amount of milliseconds in a week Cortex would return the amount of time at home that week. It is important to note that the difference between the start and end time needs to be greater than the resolution by at least one millisecond. This is why in the example the end time ends with a one.

```
{'timestamp': 1702962000000,
 'duration': 86400001,
 'resolution': 86400000,
 'data': [{'timestamp': 1702962000000, 'value': 85599870}]}
```

Looking at the results, it can be seen that this participant spent 85599870 milliseconds at home, but for the sake of understanding this value should be changed into something more digestible. This can be seen in the example continuation below:

```
pd.DataFrame(hometime["data"])["value"] / (3600 * 1000)
```

This command uses the Pandas package to take the “value”, which is the amount of home time in milliseconds and converts it into hours.

```
0                                                    23.777742
Name: value, dtype: float64
```

Within this example, it can be seen that this participant has spent nearly the entire day at home.

The second way Cortex can process sensor data is with the cortex.run() command. The functionality of the parameters are largely the same as cortex.secondary.

```
cortex.run(id_or_set, features=[], feature_params={}, start=None,
end=None,
                                                    resolution=MS_PER_DAY)
```

Cortex.run is meant to be customizable. Every parameter, from ID to the start and end times is meant to give you all the information you need in one go. Going through the parameters, examples of what can be customized are given.

- id_or_set: This parameter is who the user is querying for. The user can run this command with one participant, many participants in a contained list, or a study/researcher ID.
- features: This parameter describes what to query for. Reference Table 2 for what features are available to be queried.
- start: This parameter represents the start time in epoch time
- end: This parameter represents the end time in epoch time
- resolution: This parameter represents the time frame of which the passive data feature is calculated in MS. For example, if the user wanted to calculate hometime in A DAY the user would have resolution equal to the amount of milliseconds in a day.

To find the rest of the parameters, visit the documentation link provided.

https://docs.lamp.digital/data_science/cortex/running_cortex [49]

An example of this command in use can be found below:

```
cortex.run(part,
                                        ['screen_duration', 'data_quality'],
                          feature_params={'data_quality': {"feature":"gps",
"bin_size":3600000}},
                                                    start=1702962000000,
        end=1702962000000 + 7 * MS_IN_DAY)
```

In this example, the researcher wanted to query for both screen_duration and data_quality for one week. Within the feature_params parameter it is specified that data quality will be calculated using the GPS sensor and not the accelerometer. The bin size is set at 3600000, which means that Cortex will split the time between the start and end time into “chunks” of 3600000 milliseconds, which is the equivalent of an hour. The value calculated under data_quality is the percentage of bins or “chunks” that have a point of either GPS or accelerometer data. In this case it is GPS data. The results below are cleaned up from the normal result using the “tabulate” Python package and provide a visually appealing table to look at:

```
screen_duration:
+------------+--------------+-----------+--------------------+
|       id         |     timestamp     |     value      |         datetime          |
+------------+--------------+-----------+--------------------+
| U0929038949 | 1702962000000 | 36803989.0 | 2023-12-19 05:00:00 |
| U0929038949 | 1703048400000 | 16234234.0 | 2023-12-20 05:00:00 |
| U0929038949 | 1703134800000 | 26260277.0 | 2023-12-21 05:00:00 |
| U0929038949 | 1703221200000 | 48899441.0 | 2023-12-22 05:00:00 |
| U0929038949 | 1703307600000 | 38569785.0 | 2023-12-23 05:00:00 |
| U0929038949 | 1703394000000 | 11286251.0 | 2023-12-24 05:00:00 |
| U0929038949 | 1703480400000 |     0.0      | 2023-12-25 05:00:00 |
+------------+--------------+-----------+--------------------+
data_quality:
+------------+--------------+--------------------+--------------------+
|      id        |    timestamp    |         value          |      datetime       |
+------------+--------------+--------------------+--------------------+
| U0929038949 | 1702962000000 | 0.9166666666666666  | 2023-12-19
05:00:00                                                                |
| U0929038949 | 1703048400000 |      0.625      | 2023-12-20 05:00:00 |
| U0929038949 | 1703134800000 | 0.5416666666666666  | 2023-12-21
05:00:00                                                                |
| U0929038949 | 1703221200000 |       1.0       | 2023-12-22 05:00:00 |
| U0929038949 | 1703307600000 |       1.0       | 2023-12-23 05:00:00 |
| U0929038949 | 1703394000000 | 0.16666666666666666 | 2023-12-24
05:00:00                                                                |
| U0929038949 | 1703480400000 |       0.0       | 2023-12-25 05:00:00 |
+------------+--------------+--------------------+--------------------+
```

The actual code chunk will print a data dictionary, which can be confusing to read, thus, it is recommended to the user that when using cortex.run, to save it as a variable. Once it is saved as a variable, the user can call individual keys from the dictionary. An example of this can be seen below:

```
variable                                =                                cortex.run(part,
                                                    ['screen_duration',     'data_quality'],
                                feature_params={'data_quality':    {"feature":"gps",
"bin_size":3600000}},
                                                                    start=1702962000000,
        end=1702962000001 + 7 * MS_IN_DAY)
```

```
variable['screen_duration']
```

	id	timestamp	value	datetime
0	U0929038949	1702962000000	36803989.0	2023-12-19 05:00:00
1	U0929038949	1703048400000	16234234.0	2023-12-20 05:00:00
2	U0929038949	1703134800000	26260277.0	2023-12-21 05:00:00
3	U0929038949	1703221200000	48899441.0	2023-12-22 05:00:00
4	U0929038949	1703307600000	38569785.0	2023-12-23 05:00:00
5	U0929038949	1703394000000	11286251.0	2023-12-24 05:00:00
6	U0929038949	1703480400000	0.0	2023-12-25 05:00:00

Researchers also have the ability to pull raw sensor data. The functionality of this command is much like pulling activity events, which can be referred to in section 3.3. The command can be run without parameters, but will return an incomprehensible amount of data.

```
LAMP.SensorEvent.all_by_participant(id)
```

Parameters can be used within this command to narrow how much data is received and what sensor spec to return:

- id: This parameter represents the participant that the user wants to query for.
- origin: This parameter represents what sensor spec the user wants to query for. Reference table 1 in chapter 1 for what features are available to be queried.
- _from: This parameter represents the start time in epoch time.
- to: This parameter represents the end time in epoch time.
- _limit: This parameter represents the maximum amount of data points to pull.

An example of this command in use can be seen below:

```
LAMP.SensorEvent.all_by_participant(part, _limit=5,
                                    origin="lamp.accelerometer")["data"]
```

In this example, the researcher wanted to query the first five accelerometer data points for the participant.

```
[{'data':                    {'x':                  0.31465616822242737,
                                'y':                   4.24367094039917,
                                'z':                   8.520841598510742},
                             'sensor':                 'lamp.accelerometer',
                             'timestamp':              1703406991186},
              {'data':                   {'x':        0.19621145725250244,
                                          'y':         4.146761417388916,
                                          'z':         9.060422897338867},
                                         'sensor':     'lamp.accelerometer',
                                         'timestamp':  1703406990985},
               {'data':                  {'x':        0.3026920557022095,
                                          'y':         4.328616142272949,
                                          'z':         9.351150512695312},
                                         'sensor':     'lamp.accelerometer',
                                         'timestamp':  1703406990786},
               {'data':                  {'x':        0.2069791555404663,
                                          'y':         3.957728624343872,
                                          'z':         8.622536659240723},
                                         'sensor':     'lamp.accelerometer',
                                         'timestamp':  1703406990584},
               {'data':                  {'x':        0.33379876613616943,
                                          'y':         4.285545349121094,
                                          'z':         9.06401252746582},
                                         'sensor':     'lamp.accelerometer',
 'timestamp': 1703406990381}]
```


Looking at the example, it can be seen what raw data looks like and highlights the necessity for data pipelines like Cortex.

With the information present thus far, the reader is able to pull both active and passive data. With this background, the paper will now shift into data manipulation and how to create data visualizations that can give the user insight into participant's behavior in relation to data gathered.

4) Advanced Participant Level Example and Data Manipulation:

Cortex makes pulling data simple, but most times the data format given is not compatible with various different Python packages to create charts, tables, and different machine learning techniques. Due to this difficulty, it is worthwhile to consider an example where various levels of data manipulation are used to create a pandas DataFrame that will allow for Python packages to be used. The coming example will show:

1. Grading surveys and pulling passive data for a participant
2. How to convert timestamps in epoch time into a date format.
3. Splitting DataFrames
4. Cleaning DataFrames
5. Merging of DataFrames and Final Cleanup Using Pandas
6. Creating the Correlation Matrix & Making it Visually Appealing
7. Verifying and Gathering More Information

While many types of data visualization as possible, one compelling data visualization that is often used is a correlation matrix. These matrices can point the user into the right direction in terms of understanding the relationships between active and passive data. It is, however, important to understand that although variables may correlate with each other, there may be confounding variables that are not represented. To combat this, extra examples will be given to implement scientific processes and give weight to what the data is telling the user.

To create the correlation matrix this example will go through each step. These steps will also be labeled in reference to the enumerated list above for easy access.

Within the correlation matrix, the variables that will be included are:

- Anxiety (Measured daily through surveys on a scale of 1 to 10)
- Mood (Measured daily through surveys on a scale of 1 to 10)
- Exercise Length (Measured daily through surveys, scale is a range of durations up to 60 minutes)
- Screen Duration (Collected from passive data sensors)
- Step Count (Collected from passive data sensors)
- Data Quality (Calculated using passive data sensors)

4.1) Grading surveys and pulling passive data for a participant:

After Importing packages and connecting to the API, the user should specify the user ID and save it within a variable. In this example, ‘Newpart’. The user should then pull the active and passive data using the respective Cortex commands and save the results within properly named variables.

```
Newpart = 'U3971360469'

active = pd.DataFrame(cortex.primary.survey_scores.survey_scores(id=Newpart,
                    start=1702314000000, end=1702314000000 + 14 * MS_IN_DAY,
                                    scoring_dict=score_dict)['data'])
```

```
passive = cortex.run(Newpart, ['screen_duration', 'data_quality', 'step_count'], feature_params={'data_quality': {"feature":"gps", "bin_size":3600000}}, start=1702314000000, end=1702314000000 + 14 * MS_IN_DAY)
```

The dictionary used to grade the surveys is the same dictionary used within the survey grading subsection.

4.2) Converting timestamps:

Taking a look at the ‘active’ variable, the start timestamp needs to be converted from epoch time, which is in UTC, to a date in EST. Doing this will allow the user to merge the passive and active DataFrames together on the formatted date. Knowing that the surveys started and ended on the same date, only the start timestamp needs to be converted. The end timestamp will be deleted. Below shows the current state of the active variable before timestamp conversion.

start	end	category	question	score
1703377320153	1703379135290	Daily Anxiety Survey	Daily Anxiety Survey	5
1703377320153	1703379135290	Daily Mood Survey	Daily Mood Survey	3
1703377320153	1703379135290	Daily Exercise Survey	Daily Exercise Survey	2
1703206840167	1703256334979	Daily Anxiety Survey	Daily Anxiety Survey	6
1703206840167	1703256334979	Daily Mood Survey	Daily Mood Survey	5
1703206840167	1703256334979	Daily Exercise Survey	Daily Exercise Survey	3
1703189284252	1703191154192	Daily Anxiety Survey	Daily Anxiety Survey	7
1703189284252	1703191154192	Daily Mood Survey	Daily Mood Survey	3
1703189284252	1703191154192	Daily Exercise Survey	Daily Exercise Survey	2

```
# Convert epoch time in milliseconds to datetime
active['start'] = pd.to_datetime(active['start'], unit='ms')

# Set the timezone to NYC
nyc_timezone = pytz.timezone('America/New_York')
active['start'] = active['start'].dt.tz_localize(pytz.utc).dt.tz_convert(nyc_timezone)

# Extract date without time
active['start'] = active['start'].dt.date

#Dropping end timestamp
active = active.drop('end', axis=1)
```

start	category	question	score
2023-12-23	Daily Anxiety Survey	Daily Anxiety Survey	5
2023-12-23	Daily Mood Survey	Daily Mood Survey	3
2023-12-23	Daily Exercise Survey	Daily Exercise Survey	2
2023-12-21	Daily Anxiety Survey	Daily Anxiety Survey	6
2023-12-21	Daily Mood Survey	Daily Mood Survey	5
2023-12-21	Daily Exercise Survey	Daily Exercise Survey	3
2023-12-21	Daily Anxiety Survey	Daily Anxiety Survey	7
2023-12-21	Daily Mood Survey	Daily Mood Survey	3
2023-12-21	Daily Exercise Survey	Daily Exercise Survey	2

The passive data does not need as many steps due to the fact that cortex.run produces a datetime value. The user does need to be wary that this datetime value is in the UTC timezone and needs to be converted.

```
#Changing screen_duration to date instead of datetime
nyc_timezone = pytz.timezone('America/New_York')
passive['screen_duration']['datetime'] = passive['screen_duration']['datetime'].dt.tz_localize(pytz.utc).dt.tz_convert(nyc_timezone)
passive['screen_duration']['datetime'] = passive['screen_duration']['datetime'].dt.date
#Changing data_quality to date instead of datetime
nyc_timezone = pytz.timezone('America/New_York')
passive['data_quality']['datetime'] = passive['data_quality']['datetime'].dt.tz_localize(pytz.utc).dt.tz_convert(nyc_timezone)
passive['data_quality']['datetime'] = passive['data_quality']['datetime'].dt.date
#Changing step_count to date instead of datetime
nyc_timezone = pytz.timezone('America/New_York')
passive['step_count']['datetime'] = passive['step_count']['datetime'].dt.tz_localize(pytz.utc).dt.tz_convert(nyc_timezone)
passive['step_count']['datetime'] = passive['step_count']
```

```
['datetime'].dt.date
```

id	timestamp	value	datetime
U3971360469	1702314000000	18776209.0	2023-12-11
U3971360469	1702400400000	18280763.0	2023-12-12
U3971360469	1702486800000	33523357.0	2023-12-13
U3971360469	1702573200000	26536546.0	2023-12-14
U3971360469	1702659600000	18876016.0	2023-12-15
U3971360469	1702746000000	17029980.0	2023-12-16
U3971360469	1702832400000	26569784.0	2023-12-17
U3971360469	1702918800000	15253011.0	2023-12-18
U3971360469	1703005200000	0.0	2023-12-19

Screen_duration is currently contained within the ‘passive’ dictionary. Though only screen_duration is shown, both data_quality and step_count are also contained within ‘passive’ and look identical to screen_duration in terms of format. The table above shows the first nine rows of screen_duration within the passive dictionary.

4.3) Splitting DataFrames:

For the sake of simplicity in the merging and reduction of redundancy process, separation of each variable allows for precise changes to be made.

```
screen_duration = passive['screen_duration']
step_count = passive['step_count']
data_quality = passive['data_quality']
mood = active[active['category'] == 'Daily Mood Survey']
anxiety = active[active['category'] == 'Daily Anxiety Survey']
exercise = active[active['category'] == 'Daily Exercise Survey']
```

4.4) Cleaning DataFrames Using Pandas:

When cleaning data and getting it ready for merging, it makes sense to have a plan regarding what the user wants the DataFrame to look like. This can help to prevent unnecessary commands. In the case of this example, the final DataFrame will have a column for the UserID, date that will be merged on, and one column per data feature. Currently, the passive DataFrames have the most information in regards to the columns, but only one of the three need more than just the date and their respective data. Knowing this, the data_quality DataFrame will be established as the base and will be the only DataFrame to keep its UserID column. It is also worth noting that each of the ‘value’ columns, which are within each of the DataFrames, will need to be renamed into what the value represents. For instance, for the step_count DataFrame, the ‘value’ column needs to be renamed into ‘stepcount’ or some variation of that.

```
#Renaming value columns to what the value represents for the passive features
screen_duration = screen_duration.rename(columns = {'value':'Screen Duration'})
data_quality = data_quality.rename(columns = {'value':'Data Quality'})
step_count = step_count.rename(columns = {'value':'Step Count'})

#Dropping unnecessary columns for the passive features
data_quality = data_quality.drop(columns=['timestamp'])
screen_duration = screen_duration.drop(columns=['timestamp', 'id'])
step_count = step_count.drop(columns=['timestamp', 'id'])
```

id	Data Quality	datetime
U3971360469	0.8333333333333334	2023-12-11
U3971360469	0.9166666666666666	2023-12-12
U3971360469	1.0	2023-12-13
U3971360469	0.9166666666666666	2023-12-14
U3971360469	0.75	2023-12-15
U3971360469	0.5833333333333334	2023-12-16
U3971360469	0.9166666666666666	2023-12-17
U3971360469	0.625	2023-12-18
U3971360469	0.0	2023-12-19
U3971360469	0.0	2023-12-20

Step Count	datetime
747631.0	2023-12-11
594845.0	2023-12-12
3718114.0	2023-12-13
431599.0	2023-12-14
206163.0	2023-12-15
1043189.0	2023-12-16
726736.0	2023-12-17
1205445.0	2023-12-18
nan	2023-12-19
nan	2023-12-20

Screen Duration	datetime
18776209.0	2023-12-11
18280763.0	2023-12-12
33523357.0	2023-12-13
26536546.0	2023-12-14
18876016.0	2023-12-15
17029980.0	2023-12-16
26569784.0	2023-12-17
15253011.0	2023-12-18
0.0	2023-12-19

```
#Dropping unnecessary columns for the active features
anxiety = anxiety.drop(columns=['category', 'question'])
exercise = exercise.drop(columns=['category', 'question'])
mood = mood.drop(columns=['category', 'question'])
```

```
#Renaming value columns to what the value represents for the active features
anxiety = anxiety.rename(columns = {'score':'Anxiety Score'})
exercise = exercise.rename(columns = {'score':'Exercise Score'})
mood = mood.rename(columns = {'score':'Mood Score'})

#Renaming date columns from start to datetime in for the active features
anxiety = anxiety.rename(columns = {'start':'datetime'})
exercise = exercise.rename(columns = {'start':'datetime'})
mood = mood.rename(columns = {'start':'datetime'})
```

datetime	Mood Score
2023-12-23	3
2023-12-21	5
2023-12-21	3
2023-12-19	3
2023-12-18	2
2023-12-18	2
2023-12-17	3
2023-12-17	3

Notice how in the DataFrames above there are multiple of the same date that have a score. This is due to the fact that this participant took more than one survey in a day. There are many different solutions to remedy this, but for the example, the average between the scores will be taken.

```
#Taking the average of two survey scores if scores share a date
mood = mood.groupby('datetime')['Mood Score'].mean().reset_index()
exercise = exercise.groupby('datetime')['Exercise Score'].mean().reset_index()
anxiety = anxiety.groupby('datetime')['Anxiety Score'].mean().reset_index()
```

4.5) Merging of DataFrames and Final Cleanup:

Now that every data point is attached to its formatted date, each of the DataFrames are ready to be merged together.

```
#Merging all DataFrames into one
final_df = pd.merge(data_quality, screen_duration, how = 'left', on = 'datetime')
final_df = pd.merge(final_df, step_count, how = 'left', on = 'datetime')
final_df = pd.merge(final_df, anxiety, how = 'left', on = 'datetime')
final_df = pd.merge(final_df, mood, how = 'left', on = 'datetime')
final_df = pd.merge(final_df, exercise, how = 'left', on = 'datetime')
```

	id	Data Quality	datetime	Screen Duration	Step Count	Anxiety Score	Mood Score	Exercise Score
0	U3971360469	0.833333	2023-12-11	18776209.0	747631.0	7.0	3.0	1.0
1	U3971360469	0.916667	2023-12-12	18280763.0	594845.0	6.0	3.0	0.5
2	U3971360469	1.000000	2023-12-13	33523357.0	3718114.0	6.0	3.5	2.5
3	U3971360469	0.916667	2023-12-14	26536546.0	431599.0	6.0	3.0	1.0
4	U3971360469	0.750000	2023-12-15	18876016.0	206163.0	6.0	3.0	1.0
5	U3971360469	0.583333	2023-12-16	17029980.0	1043189.0	5.0	3.5	1.0
6	U3971360469	0.916667	2023-12-17	26569784.0	726736.0	5.0	3.0	1.0
7	U3971360469	0.625000	2023-12-18	15253011.0	1205445.0	9.0	2.0	1.0
8	U3971360469	0.000000	2023-12-19	0.0	NaN	9.0	3.0	0.0
9	U3971360469	0.000000	2023-12-20	0.0	NaN	NaN	NaN	NaN
10	U3971360469	0.791667	2023-12-21	23245933.0	842118.0	6.5	4.0	2.5
11	U3971360469	0.750000	2023-12-22	24242672.0	432991.0	NaN	NaN	NaN
12	U3971360469	1.000000	2023-12-23	21260291.0	913142.0	5.0	3.0	2.0

Now that the DataFrames are merged together, the user can look for discrepancies. Two glaring edits that need to be implemented into this DataFrame is switching the locations of ‘Data Quality’ and ‘datetime’ and changing the units of ‘Screen Duration’ from milliseconds to hours.

A keen eye will also notice strange results within the ‘Step Count’ column. It is known that the values within ‘Step Count’ are supposed to represent how many steps a participant has taken within a day, but the values obtained are way too high to be accurate. It is unknown what went wrong while collecting this participant’s data, but it goes to show that researchers should keep a sharp eye out for discrepancies. For this example, the step count correlations will be taken with a grain of salt.

Finally, on December 19th and 20th screen duration is zero for both. Since zero can and will be used within correlations, the result can be skewed by outliers. Due to this fact, it is reasonable to turn these zeros into NaN values using the Numpy Python package.

```
#Switching the column locations of datetime and data_quality
final_df['Data Quality'], final_df['datetime'] = final_df['datetime'].copy(), final_df['Data Quality'].copy()
```

```
final_df = final_df.rename(columns = {'datetime': 'Data Quality', 'Data Quality': 'datetime'})

#Converting Screen Duration in MS to hours
final_df['Screen Duration'] = final_df['Screen Duration']/3600000

final_df['Screen Duration'][8] = np.nan
final_df['Screen Duration'][9] = np.nan
```

	id	datetime	Data Quality	Screen Duration	Step Count	Anxiety Score	Mood Score	Exercise Score
0	U3971360469	2023-12-11	0.833333	5.215614	747631.0	7.0	3.0	1.0
1	U3971360469	2023-12-12	0.916667	5.077990	594845.0	6.0	3.0	0.5
2	U3971360469	2023-12-13	1.000000	9.312044	3718114.0	6.0	3.5	2.5
3	U3971360469	2023-12-14	0.916667	7.371263	431599.0	6.0	3.0	1.0
4	U3971360469	2023-12-15	0.750000	5.243338	206163.0	6.0	3.0	1.0
5	U3971360469	2023-12-16	0.583333	4.730550	1043189.0	5.0	3.5	1.0
6	U3971360469	2023-12-17	0.916667	7.380496	726736.0	5.0	3.0	1.0
7	U3971360469	2023-12-18	0.625000	4.236948	1205445.0	9.0	2.0	1.0
8	U3971360469	2023-12-19	0.000000	NaN	NaN	9.0	3.0	0.0
9	U3971360469	2023-12-20	0.000000	NaN	NaN	NaN	NaN	NaN
10	U3971360469	2023-12-21	0.791667	6.457204	842118.0	6.5	4.0	2.5
11	U3971360469	2023-12-22	0.750000	6.734076	432991.0	NaN	NaN	NaN
12	U3971360469	2023-12-23	1.000000	5.905636	913142.0	5.0	3.0	2.0

4.6) Creating the Correlation Matrix & Making it Visually Appealing

The correlation matrix is created with a simple command through the “pandas” Python package. To make it visually appealing, the “seaborn” package will be used. The result will be shown in figure 8.

```
#Creation of correlation matrix
correlation_matrix = final_df.corr()

#Eliminating top half of Correlation matrix including correlations of 1
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
correlation_matrix = correlation_matrix.mask(mask)
#Dropping rows with no values
correlation_matrix = correlation_matrix.drop(index='Data Quality', columns='Exercise Score')
#Figure creation
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1);
```

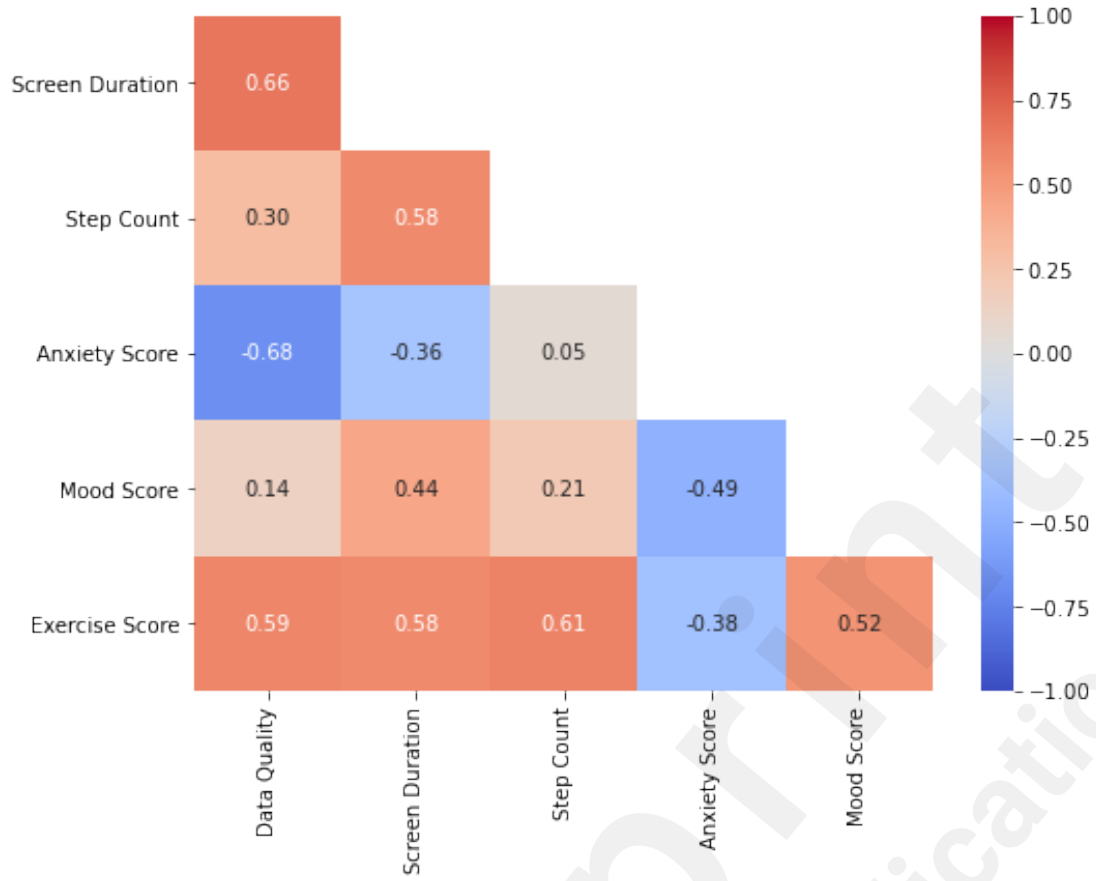

Figure 8 (Correlation Matrix): *This graph shows each variable and its relationship to the other variables within the final DataFrame. Correlations closer to one represent a positive linear correlation whereas correlations close to negative one represent a negative linear correlation. Positive in this context means that both variables are going in the same direction. Negative means that the variables are heading in opposite directions.*

Looking at the graph, it is possible to see at an overview how the variables may correlate with each other. It is important to note that correlation does not equal causation, but also how this graph can help share back both active and passive data with participants. To obtain and alternative image of what this data may represent, it is also helpful to create scatterplots.

4.7) Verifying and Gathering More Information:

In the correlation matrix above, there are some interesting correlations to look at. Perhaps the most interesting is the .52 correlation between exercise score and mood. The assumption that most people would make is that if the exercise score is higher the participant's mood would also be higher, but to make sure this is the case, a scatter plot should be made. The scatter plot results will be shown in figure 9.

```
plt.scatter(final_df['Exercise Score'], final_df['Mood Score'])
plt.xlabel('Exercise Score')
plt.ylabel('Mood Score')
plt.title('Basic Scatterplot')
```

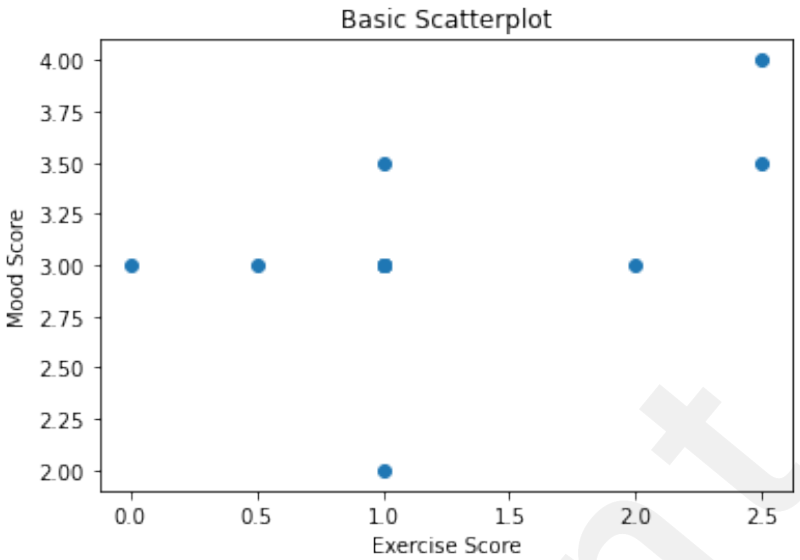

Figure 9 (Scatterplot of Exercise Score v. Mood Score): *This graph shows the mood score with respect to the exercise score. The x-axis represents the exercise score, with higher scores meaning the participant completed more exercise that day. The y-axis represents the mood score, with higher scores representing a more positive mood. Each dot represents one day.*

Looking at the graph it can be seen that the initial assumption that as the exercise score increases so does mood looks to be correct. For the truly diligent user, more information can be gathered by running a simple linear regression. Especially due to the linear look of the data points. To do this, a Python package named “statsmodels.api” will be used.

```
#Linear Regression
import statsmodels.api as sm
X = final_df[['Exercise Score']]
Y = final_df['Mood Score']
X = sm.add_constant(X)
X = X.dropna()
Y = Y.dropna()

model = sm.OLS(Y, X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Mood Score   R-squared:                       0.275
Model:                            OLS   Adj. R-squared:                  0.194
Method:                 Least Squares   F-statistic:                     3.409
Date:                Thu, 29 Feb 2024   Prob (F-statistic):             0.0979
Time:                        16:59:16   Log-Likelihood:                -5.4891
No. Observations:                  11   AIC:                             14.98
Df Residuals:                       9   BIC:                             15.77
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.6893      0.255     10.552      0.000       2.113       3.266
Exercise Score 0.3272      0.177      1.846      0.098      -0.074       0.728
==============================================================================
Omnibus:                        7.947   Durbin-Watson:                   2.315
Prob(Omnibus):                  0.019   Jarque-Bera (JB):                3.401
Skew:                          -1.185   Prob(JB):                        0.183
Kurtosis:                       4.342   Cond. No.                         3.83
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Looking at the p-value in red, it can be seen that this relationship is approaching the point where it would be considered significant. (under .05) This is not to say that this is not a valuable result. There absolutely is a case to be made that this participant's mood can benefit from increased exercise.

Discussion

Thus far, this paper has illustrated the technical process of utilizing Cortex to process, manipulate, visualize, and analyze digital phenotyping data. To more clearly illustrate the utility of these

functionalities, we will discuss below the significance of the steps outlined in this paper, as well as use cases. Figure 10 below highlights a common work flow with Cortex that is applicable to many types of studies and use case.

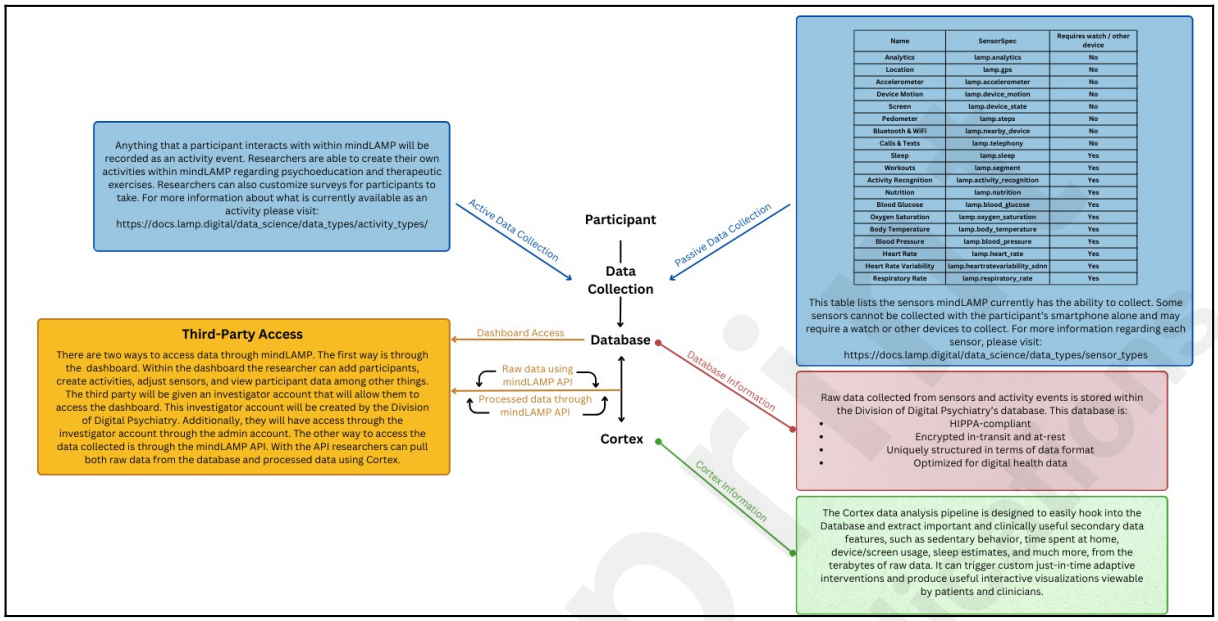

Figure 10: A schematic of a common deployment of mindLAMP, hosted by the BIDMC team, and Cortex

Basic features of Cortex can be immediately translated into analysis of digital phenotyping data. For example, Cortex collects timestamps not only of start and end time of active data activities (eg: a survey), but the time of specific item responses for a given survey, and of any discrete passive data points. Prior research has utilized analytical methods such as this to predict self reported thoughts of harm based on latency [21] in responding to items assessing such. Furthermore, epoch time provides foundational support for pivotal features that can be derived from cortex, such as sleep duration, and screen usage data, which allow the transformation of passive data into clinically relevant variables. While discrete data points can be informative, even without incorporating time (for example, an individual's home may be approximated for most by the gps point most frequently visited), the formatting of data points with timestamps allows for more nuanced analysis and deeper insight. By combining basic Cortex features it is also possible to derive novel digital biomarkers as well as replicate those created by other teams.

The nature of digital phenotyping data when combined with Cortex is the flexibility in using derived features or even creating novel digital biomarkers. For example, Cortex derived features, such as sleep duration, time spent at home, screen time, and data quality, have been demonstrated when utilized in research applying anomaly detection. Specifically, anomalies within personal baselines of active data and cortex derived features of data for individuals with schizophrenia have been used to predict clinical relapse, an example of advanced data processing [22]. Using the methods outlined above in this paper, any digital phenotyping data can be processed in features and DataFrames that will enable easy use of the code (https://www.notion.so/digitalpsychiatry/Anomaly-Detection-in-R-177a40b5120343fdad1bff6db7632118) [50] for anomaly detection. Related applications may include

using the digital phenotyping data features and anomaly detection to detect early warning of risk of self harm, relapse in bipolar disorder, or medication adherence as three broad examples.

The nature of digital phenotyping is also well suited when combined with Cortex for machine learning research. Linear regression models are a common tool for analyzing digital phenotyping data, and the readily accessible Cortex features, particularly those derived from passive data, make Cortex a productive API for analyzing the relationships between digital phenotyping derived features. In addition, the abundance of time and duration columns in the dataset makes Cortex a useful tool to prepare data for time series analysis. Recent research has leveraged Cortex to pull data, split it using a windowing approach, and train a variety of Recurrent Neural Network (RNN) models using a diverse architecture including Long Short Term Memory (LSTM). Cortex's unique strengths in over-time data allow researchers to use the library for a variety of predictive tasks ranging from prognosis prediction to quantitative score prediction.

The utility features such as visualizations for monitoring data quality and providing data back to users have been illustrated in our BIDMC-based clinic utilizing mindLAMP [23]. As illustrated above, patients can be offered heatmaps of passive data volume generated daily (see section 3.2 B), in addition to visualizations of active data. With staff allocated to assess patient data quality multiple times per week, they can reach out to patients about their data quality, with specific strategies for ameliorating poor data quality [20]. For example, low passive data quality across sensors may be a result of low weekly app usage, as many devices limit app data collection based on use. However poor GPS quality with high accelerometer quality may simply indicate a misconfiguration of phone settings. The root cause is easy to detect with data visualization. Similarly, patients provided with visualizations of their own active data overlaid with passive data streams, can gain insight by observing and even interacting with their own data, as shown below in Figure 11.

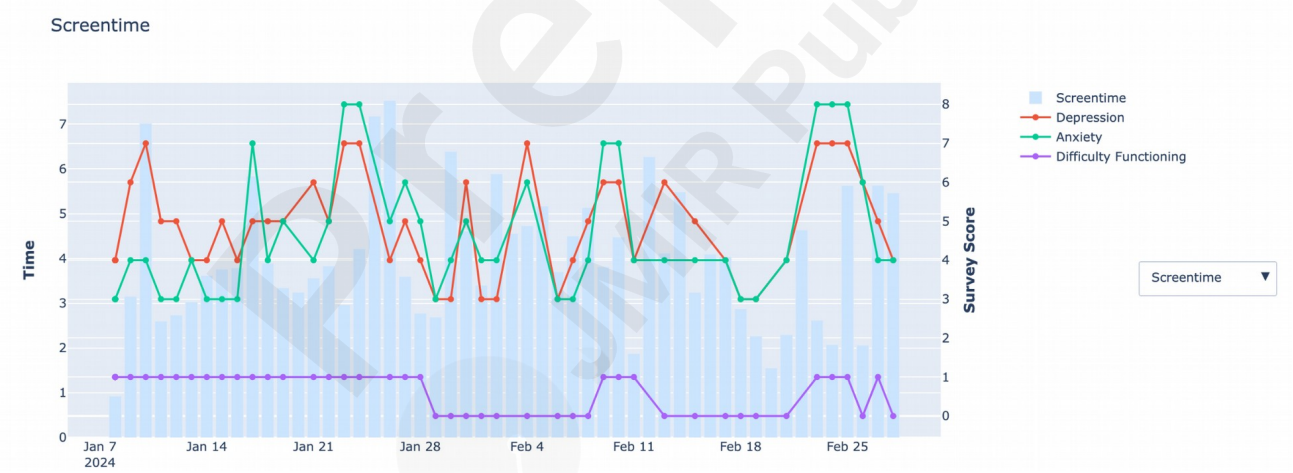

Figure 11: An example of a test account data used to illustrate how one passive data feature screen time can be visualized in light of different symptoms and functioning metrics.

Conclusion

Digital phenotyping presents researchers, clinicians, and patients with a wealth of new data. Packages like Cortex ensure this data and resulting insights are accessible to all. Through reviewing how to pull active and passive data, check participant's data quality, manipulate DataFrames, and create basic data visualizations, this paper serves as an interactive guide for advancing reproducible

and replicable digital phenotyping research.

Like many other tools, Cortex has its limits. While Cortex has been optimized for mindLAMP, it will still work on any dataset that meets the formatting requirements outlined in section 1. Cortex will attempt to provide meaningful insights and features derived from low data quality, but the quality of such features is questionable. Fortunately, low data quality can be prevented with regular checkups with participants using Cortex for quality checks, as discussed in section 3.2. Clinical judgment is always necessary when interpreting any output and at the time of this writing there are no digital biomarkers for psychiatry recognized by the FDA or other regulatory bodies.

Cortex is an open-source package. As such, it is always evolving, both internally within the team and externally from users worldwide. Due to this evolving nature, this paper is not meant to be rigid, but more of a living document to be updated periodically. Updates are posted at docs.lamp.digital

Acknowledgments: We would like to thank Jenn Sabbagh MBA for her ongoing support of digital psychiatry research over the years and advocacy for this work through her research administration role.

Data Availability: The code is available on our GitHub. This paper did not use or generate any data and all examples provided are simulated.

Conflicts of Interest: None declared

References

1. Huckvale K, Venkatesh S, Christensen H. Toward clinical digital phenotyping: a timely opportunity to consider purpose, quality, and safety. NPJ digital medicine. 2019 Sep 6;2(1):1-1
2. Montag C, Quintana DS. Digital phenotyping in molecular psychiatry—a missed opportunity?. Molecular Psychiatry. 2023 Jan;28(1):6-9.
3. 1-Moura I, Teles A, Viana D, Marques J, Coutinho L, Silva F. Digital phenotyping of mental health using multimodal sensing of multiple situations of interest: A systematic literature review. Journal of Biomedical Informatics. 2023 Feb 1;138:104278.
4. Ebner-Priemer U, Santangelo P. Digital phenotyping: hype or hope?. The Lancet Psychiatry. 2020 Apr 1;7(4):297-9.
5. Onnela JP. Opportunities and challenges in the collection and analysis of digital phenotyping data. Neuropsychopharmacology. 2021 Jan;46(1):45-54.
6. Currey D, Torous J. (2023). Increasing the value of digital phenotyping through reducing missingness: a retrospective review and analysis of prior studies. BMJ Ment Health, 26:e300718.
7. Asselbergs J, Ruwaard J, Ejdys M, Schrader N, Sijbrandij M, Riper H. Mobile phone-based unobtrusive ecological momentary assessment of day-to-day mood: an explorative study. Journal of medical Internet research. 2016 Mar 29;18(3):e5505.
8. FX, Silverman BC, Monette P, Kimble S, Rauch SL, Baker JT. An ethics checklist for digital health research in psychiatry. Journal of medical Internet research. 2022 Feb 9;24(2):e31146.
9. Shen FX, Baum ML, Martinez-Martin N, Miner AS, Abraham M, Brownstein CA, Cortez N, Evans BJ, Germine LT, Glahn DC, Grady C. Returning Individual Research Results from

Digital Phenotyping in Psychiatry. The American Journal of Bioethics. 2023 May 5:1-22.

10. Fuhrmann LM, Weisel KK, Harrer M, Kulke JK, Baumeister H, Cuijpers P, Ebert DD, Berking M. Additive effects of adjunctive app-based interventions for mental disorders-A systematic review and meta-analysis of randomised controlled trials. Internet Interventions. 2023 Dec 18:100703.
11. Polhemus A, Novak J, Majid S, Simblett S, Morris D, Bruce S, Burke P, Dockendorf MF, Temesi G, Wykes T. Data visualization for chronic neurological and mental health condition self-management: Systematic review of user perspectives. JMIR mental health. 2022 Apr 28;9(4):e25249.
12. https://github.com/carissalow/rapids
13. https://github.com/onnela-lab/forest
14. https://github.com/BIDMCDigitalPsychiatry/LAMP-cortex
15. Breitinger S, Gardea-Resendez M, Langholm C, Xiong A, Laivell J, Stoppel C, Harper L, Volety R, Walker A, D'Mello R, Byun AJS, Zandi P, Goes FS, Frye M, Torous J. Digital Phenotyping for Mood Disorders: Methodology-Oriented Pilot Feasibility Study. J Med Internet Res. 2023 Dec 29;25:e47006. doi: 10.2196/47006. PMID: 38157233; PMCID: PMC10787337.
16. Gansner M, Nisenson M, Carson N, Torous J. A pilot study using ecological momentary assessment via smartphone application to identify adolescent problematic internet use. Psychiatry Res. 2020 Nov;293:113428. doi: 10.1016/j.psychres.2020.113428. Epub 2020 Aug 23. PMID: 32889344.
17. Bilden R, Torous J. Global collaboration around digital mental health: the LAMP consortium. Journal of Technology in Behavioral Science. 2022 Jun;7(2):227-33.
18. Rodriguez-Villa E, Rozatkar AR, Kumar M, Patel V, Bondre A, Naik SS, Dutt S, Mehta UM, Nagendra S, Tugnawat D, Shrivastava R, Raghuram H, Khan A, Naslund JA, Gupta S, Bhan A, Thirthall J, Chand PK, Lakhtakia T, Keshavan M, Torous J. Cross cultural and global uses of a digital mental health app: results of focus groups with clinicians, patients and family members in India and the United States. Glob Ment Health (Camb). 2021 Aug 24;8:e30. doi: 10.1017/gmh.2021.28. PMID: 34512999; PMCID: PMC8392688.
19. Division of Digital Psychiatry. Privacy. Available from: https://docs.lamp.digital/privacy/ . Accessed June 17, 2024.
20. Currey D, Torous JIncreasing the value of digital phenotyping through reducing missingness: a retrospective review and analysis of prior studies *BMJ Ment Health* 2023;**26:**e300718.

21. Henson P, Torous J. Feasibility and correlations of smartphone meta-data toward dynamic understanding of depression and suicide risk in schizophrenia. International journal of methods in psychiatric research. 2020 Jun;29(2):e1825.
22. Cohen A, Naslund JA, Chang S, Nagendra S, Bhan A, Rozatkar A, Thirthalli J, Bondre A, Tugnawat D, Reddy PV, Dutt S. Relapse prediction in schizophrenia with smartphone digital phenotyping during COVID-19: a prospective, three-site, two-country, longitudinal study. Schizophrenia. 2023 Jan 27;9(1):6.
23. Macrynikola N, Nguyen N, Lane E, Yen S, Torous J. The Digital Clinic: An innovative mental health care delivery model utilizing hybrid synchronous and asynchronous treatment. NEJM Catalyst Innovations in Care Delivery. 2023 Aug 16;4(9):CAT-23.
24. Division of Digital Psychiatry. Call duration. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/call_duration. Accessed June 17, 2024.
25. Division of Digital Psychiatry. Call number. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/call_number. Accessed June 17, 2024.
26. Division of Digital Psychiatry. Data quality. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/data_quality. Accessed June 17, 2024.
27. Division of Digital Psychiatry. Entropy. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/entropy. Accessed June 17, 2024.
28. Division of Digital Psychiatry. Game results. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/game_results. Accessed June 17, 2024.
29. Division of Digital Psychiatry. HealthKit sleep duration. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/healthkit_sleep_duration. Accessed June 17, 2024.
30. Division of Digital Psychiatry. Hometime. Available from:

https://docs.lamp.digital/data_science/cortex/features/secondary/hometime. Accessed June 17, 2024.

31. Division of Digital Psychiatry. Inactive duration. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/inactive_duration. Accessed June 17, 2024.

32. Division of Digital Psychiatry. Nearby device count. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/nearby_device_count. Accessed June 17, 2024.

33. Division of Digital Psychiatry. Screen duration. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/screen_duration. Accessed June 17, 2024.

34. Division of Digital Psychiatry. Step count. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/step_count. Accessed June 17, 2024.

35. Division of Digital Psychiatry. Survey results. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/survey_results. Accessed June 17, 2024.

36. Division of Digital Psychiatry. Trip distance. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/trip_distance. Accessed June 17, 2024.

37. Division of Digital Psychiatry. Trip duration. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/trip_duration. Accessed June 17, 2024.

38. Division of Digital Psychiatry. Call degree. Available from: https://docs.lamp.digital/data_science/cortex/features/secondary/call_degree. Accessed June 17, 2024.

39. https://pandas.pydata.org/about/

40. https://numpy.org/doc/stable/user/whatisnumpy.html

41. https://docs.python.org/3/library/datetime.html

42. https://pypi.org/project/pytz/

43. https://seaborn.pydata.org/

44. https://pypi.org/project/matplotlib/

45. https://pypi.org/project/calplot/

46. https://altair-viz.github.io/

47. https://plotly.com/python/

48. Division of Digital Psychiatry. Participant Level Visualizations. Available from: https://docs.lamp.digital/data_science/cortex/visualizations/participant_level. Accessed June 17, 2024.

49. Division of Digital Psychiatry. Running Cortex. Available from: https://docs.lamp.digital/data_science/cortex/running_cortex. Accessed June 17, 2024.

50. Division of Digital Psychiatry. Anomaly Detection in R. Available from: https://www.notion.so/digitalpsychiatry/Anomaly-Detection-in-R-177a40b5120343fdad1bff6db7632118. Accessed June 17, 2024.

51. https://www.epochconverter.com/