Fitbit SIS

Matt Barren

December 2016

1 Introduction

The Fitbit SIS is a slow intelligence system utilizing data collection from the Fitbit wristband sensor to perform activity classification. Fitbit software is capable of making a few automated classifications, sedentary, walking, and running. The motivation of Fitbit SIS is to expand the domain of potential activity classifications by utilizing a classification learning model and leveraging the user through active learning. So far, the model classifies varying degrees of walking and running, lifting, and sitting.

The following sections will describe the particular Fitbit wristband used (section 2), implementation of Fitbit data with the SIS platform (section 3), and future works and concluding remarks (section 4).

2 Fitbit Wristband

A Fitbit Charge HR wristband was used to collect sensor data. This particular wristband is capable of providing several different data types, step, distance, heart rate, elevation and calorie data. Fitbit has two classes of data, interday and intraday data. Interday can be seen as daily summative data that provides averages and sums of the data collection. On the other hand, intraday data can be specified to time intervals throughout a given day.For this project, intraday data was collected at the minute by minute level from Fitbit. The following section Fitbit Data describes how the various data types are derived.

2.1 Fitbit Data

Fitbit has several internal sensors, PurePulse, an accelerometer, and an altimeter. PurePulse LED lights are used shine a green flash onto the wrist of the wearer. The reflection of these lights are used to determine the blood coursing through an individual's capilaries at a given moment.

For steps, Fitbit uses a 3-axis accelerometer to detect motion. The algorithm for step counting is split into walking and running steps. For both of these step types, there is a prescribed stride length given the user's height and gender. Similarly, to calculate distance the step quantity is multiplied by its particular stride type.

Fitbit wristbands have an altimeter to calculate the elevation change. When Fitbit software examines the altimeter it classifies users elevation change only if it is increasing, which is added to the floor count. Altitude decreases are not considered by the software towards floor values.

Calories are determined by two components the Basal Metabolic Rate (BMR) and physical activity recorded or manually logged. BMR is the number of calories that an individual burns from

involuntary activities, such as breathing. This is determined by the gender, age, height, and weight of the individual. Additionally, activity detection, such as walking, and manually logged in activities comprise a calories burned value. The activity, duration, and heart rate are used as parameters for calculating burned calories.

There are some limitations to the accuracy of these sensors. The PurePulse sensor is biased based on the placement relative to an individuals wrist. Varying the distance between the lights and skin will result in an inconsistent pulse reading. User stride length depends on fixed physical characteristics of the user. For the Fitbit Charge HR 2 and newer models, it is possible to update the stride. As an additional note about Fitbit data, all cumulative values are reset at midnight.

In regards to the data used for the Fitbit SIS platform, heart rate, steps, and elevation were utilized. Since calories and distance calculations are dependent on other data points, these two data sets were disregarded.

3 Implementation

The implementation section details the implementation for each of the components and the communication that occurs between them. For all components where coding was necessary, the scripts are provided in a separate file that will be submitted with the report. All coding languages are in Java with the exception of the Google Scripts code for the "Raw Data" Google Sheet.

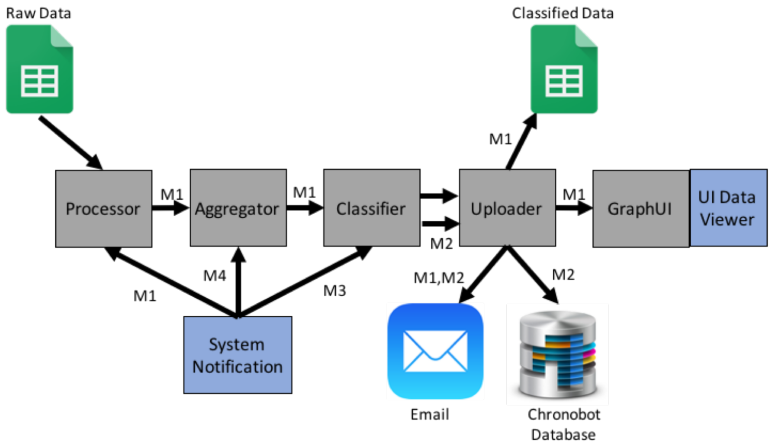


Figure 1: SIS Fitbit Implementation Diagram

3.1 Message Passing

There are four types of messages in the system:

- M1: New reading being passed from the prior component
- M2: Alert of a particular activity. This will occur if there has been 30 consecutive minutes of either strenuous running or sitting. The messages are listed below.
  - Strenuous Running: "Good run!"

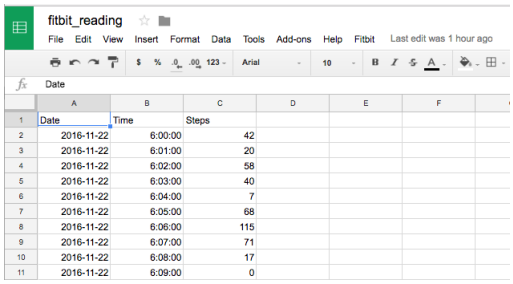
  - Sitting: " Get up and walk around! You've been sitting too long."

- M3: New iteration should be performed to retrain the supervised classifier on new data.
- M4: Initial clustering of training data using expectation maximization.

3.2 Components

3.2.1 Data Requesting Script and Raw Data Google Sheet

Data requests are handled by a Google Script that populates a Google Sheet called "fitbit_reading". The script has a trigger set to request data every hour from Fitbit with an hour offset. Thus, at 8:00 am, data is collected from that day for the hours of 6:00 am to 6:59 am. The script trigger was chosen to be less fine-grained than say five minutes because Fitbit limits the number of requests that can be performed per a day. Additionally, an hour offset was employed because updating Fitbit data drains the battery of both the wristband and the cell phone, which is a 5S iPhone model. Also, the phone used does not have unlimited data services.



	A	B	C	D	E	F
1	Date	Time	Steps			
2	2016-11-22	6:00:00	42			
3	2016-11-22	6:01:00	20			
4	2016-11-22	6:02:00	58			
5	2016-11-22	6:03:00	40			
6	2016-11-22	6:04:00	7			
7	2016-11-22	6:05:00	68			
8	2016-11-22	6:06:00	115			
9	2016-11-22	6:07:00	71			
10	2016-11-22	6:08:00	17			
11	2016-11-22	6:09:00	0			

Figure 2: Fitbit Raw Readings

The raw data is listed as minute by minute data with a date, time, and quantity. The steps are the number of steps for that particular minute interval, the heart rate is the average rate during that minute, and the elevation is the cumulative elevation change.

3.2.2 Fitbit System Interface

The Fitbit System Interface is used to display the messages passing between each of the components. When a message is passed or an action occurs, the interface receives a message to update its display for that component.

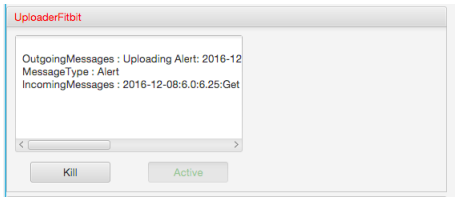


Figure 3: Example of Upload Alert

3.2.3 Processor

The processor is the start point for remote data being passed to the local machine. The processor contains a Time Notification class that passes a message to the processor to begin extracting data from the raw data Google Sheet. The time notification always notifies the Processor to extract data upon startup. After, the time notification component will wait for a period of time before it sends its next notification.

Next, the processor retrieves the raw data, formats the date, and writes the data into a "temp.xlsx" file. When this is finished, the processor component sends a new message notification to the aggregator that the "temp.xlsx" file has been written. The processor has the option to delete the values from the Google Sheet. This option was used for testing purposes.

3.2.4 Aggregator

Upon receiving a new reading notification from the processor, the aggregator will open the temp.xlsx file and extract the data. The aggregator aggregates the data into 15 minute time intervals. Each interval contains the cumulative steps, step variance, cumulative elevation, elevation variance, average heart rate, and heart rate variance.

When constructing the time windows, the data has to be cleaned due to the possibility that there are gaps in the readings. This is possible when the Fitbit wristband is worn. For steps and elevation, if there are no readings, Fitbit assumes the values to be 0. Conversely, Fitbit does not infer or assume heart rate readings. Therefore, if there is not a reading at 12:25 PM for example, then this reading is missing in the data set. If there is at least one reading in a given time interval, the average is calculated based on those values. If the interval has no heart rate readings, then the interval is not added to the aggregated data set.

Upon completion, the aggregated intervals are written into a "temp.arff" file, which is a file format used by WEKA. After this file is composed, the "temp.xlsx" file is deleted. Upon writing the WEKA file, the classifier is sent a notification that new readings are available.

3.2.5 Classifier

The classifier can receive one of three different messages.

- New Reading: notifies the classifier that a new reading is available in the "temp.arff" file to be classified
- Initial Iteration: an unsupervised clustering iteration using expectation maximization (EM) on the "training_classified.arff" file.
- New Iteration: a supervised training iteration using a bayes classifier on the "training_classified.arff" file.

Deed 1

The **new reading** message occurs when the aggregator has finished producing the "temp.arff" file. The classifier component then uses the "training_classified.arff" file to perform an naive bayes classification on the new data. The possible classifications are as follows: *sitting, light walk, moderate walk, uphill walk, strenuous walk, running, strenuous running, lifting, sitting stressed.* Sitting stressed is a classification I am planning on exploring further. It appeared a few times, but essentially, it was a high heart rate while sitting. It is possible that this reading is the result of a misreading of the Fitbit heart rate sensor.

When this is complete the "temp.arff" file is updated and the name is changed to "temp_classified.arff". While performing the classification process, if it is observed that the user has been either strenusously running or sitting for a 30 minute interval, an alert is sent with a message to the uploader. The alert will either read *"Good Run!"* or *"Get up and walk around! You've been sitting too long"*, which are sent to the SIS Chronobot Database. Upon completing classification process, the uploader is notified of a new set of classified readings to be uploaded to the Google Sheet classified data file.

In the Fitbit User Interface, there is detail about how the user can update incorrect classifications, which are then added to the training set to allow for active learning. Periodically, the time notification class will send a **new iteration** message to the classifier, which will result in the model being retrained with the updated classifications included.

Prior to receiving the first new reading and supervised classification, initial training had to be performed to generate the classes. The **first iteration** message uses the initial raw training data to perform clustering using expectation maximization (EM). For this, several clustering size were considered along with the integration of other features, such as minimum heart rate, maximum heart rate. After exploring the feature sets and k values, a k value of 14 clusters and using the feature set cumulative steps, cumulative elevation, mean heart rate, and their respective variances.

Next the clusters were examined relative to the ground truth, and classification names were assigned along with some aggregation of clusters. With these classification names, the supervised ensemble model is trained, and then used for incoming readings.

3.2.6 Uploader

The uploader component receives new reading messages from the classifier. When an incoming message is received, the uploader reads the "temp_classified.arff" and uploads the data to two locations. First, a Google Sheet named "classified data" gets the new values appended to the end of the file. If these values match previous date and time values, then the old values are overwritten with updated values. This is performed by using the Google Sheets class in the Utilities Library.

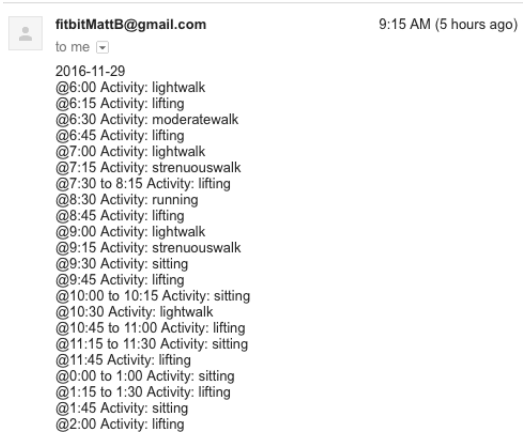


Figure 4: Email Summary

Deed 2

Next, an email is sent to and from fitbitMattB@gmail and chronobot@ksiresearch.org.com with a summary of the classifications from the new readings using the Google Email class in the Utilities

Library. The email is titled:
Activity from <Start_Date >@ <Start_Time >to <End_Date >@ <End_Time >

The activities are listed with the start time for each interval,

@ <Start_Time >Activity : <Activity >

For activities that are the same on time contiguous intervals, the time is aggregated to a single line,

@ <Start_Time >to <End_Time >Activity : <Activity >

Additionally, if an alert is sent from the classifier, an email is sent to the previously mentioned addresses as a notification with the following format:
Subject FitbitAlert: <Date >: <Message >
Body FitbitAlert: <Date >From: <Start_Time >to <End_Time >Message: <Message >

1684476 376896 2016-11-29 18:45:00 Fitbit message Good Run!

Figure 5: Chronobot Entry

The alert is also posted to the Chronobot database containing the message. The alerts are the aforementioned messages related to cumulative sitting and running.

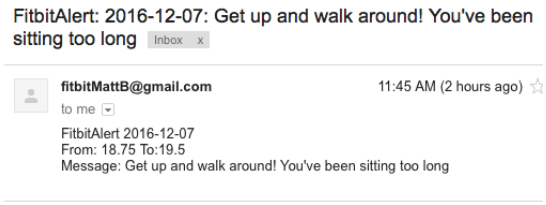


Figure 6: Alert Email

After, completing an upload for a reading, the uploader sends a message to the GraphUI component to notify that new readings have been added to Google Sheets classified data file.

3.2.7 GraphUI and Fitbit User Interface

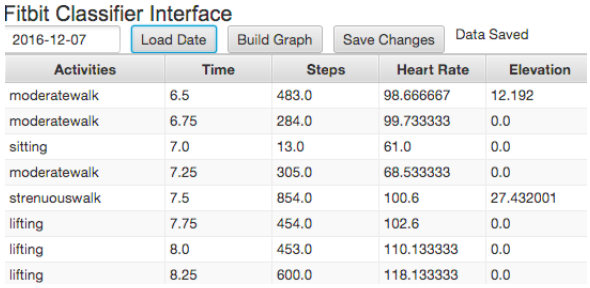
The GraphUI is a basic component that waits for a new reading message from the uploader. When this message is received the GraphUI starts a new Fitbit User Interface process. When starting by the GraphUI a command line message is passed to tell the Fitbit User Interface to load the most recent days data from the Google Sheets classified data file. The Fitbit User Interface provides a platform for users to visualize classified data by day and change classifications.

Deed 3

In the Fitbit User Interface, there is a table that displays all of the readings for a particular day of activities. The first two columns display the activity and time. The remaining columns display the aggregated data for each interval. The activities column contains the classifications for each time interval. These classifications can be edited, and are the only column that can be manipulated. When activities are altered, the graphs reflect the new classification by changing the data point to the appropriate marker on the legend. After changing a classification, a notification is posted to notify the user that these classifications have not been saved. Upon saving the updated classifications, the Google sheet file is update and the training file is updated. If the training file already contains this

Figure 7: Data Viewer User Interface

time interval, the old interval will be updated with the new classification. Updating the training file facilitates active learning by allowing the user to adjust the model by editing classifications.



Activities	Time	Steps	Heart Rate	Elevation
moderatewalk	6.5	483.0	98.666667	12.192
moderatewalk	6.75	284.0	99.733333	0.0
sitting	7.0	13.0	61.0	0.0
moderatewalk	7.25	305.0	68.533333	0.0
strenuouswalk	7.5	854.0	100.6	27.432001
lifting	7.75	454.0	102.6	0.0
lifting	8.0	453.0	110.133333	0.0
lifting	8.25	600.0	118.133333	0.0

Figure 8: Close Up of Data Viewer User Interface Functions

Deed 4

In terms of visualization three graphs are provided. Two of the graphs are steps and heart rate respective to time. The third graph, is steps respective to heart rate. A graph including elevation was considered, but the elevation change is predominantly 0, which is not particularly revealing. The axis intervals and scale are adjustable to allow for the data view to be changed. The graphs are particularly useful to understand causal relationships between the values and the activity classifications.

Deed 5

3.3 Utilities Library

The Utilities Library can be considered the backbone for many of the actions that occur across the Fitbit SIS system. This library contains original SIS classes used, such as MsgDecoder, MsgEncoder, and KeyValueList. Additionally, it contains the following classes:

- GoogleEmail: The Google Email class allows for the uploader to send an email to the fitbitMattB@gmail.com username. This email address can be changed, but the sender is fitbit-

MattB@gmail.com unless the hardcoding is changed. Note: This would require getting the appropriate scopes and authorization too.

- GoogleSheet: The Google Sheet class provides a class to interact with the two Google Sheets "fitbit_reading" and "Classified_Data". The class allows for both reading and writing.
- Excel: The Excel class allows for creating, reading, and writing to Excel(xlsx) sheets locally.
- Data_Set: This is a supporting class for Google Sheets to allow for data to be encapsulated in a concise class.

For many of these classes, there are a variety of dependencies. These dependencies are generated by using Gradle, which results in a jar file containing the classes and their dependencies.

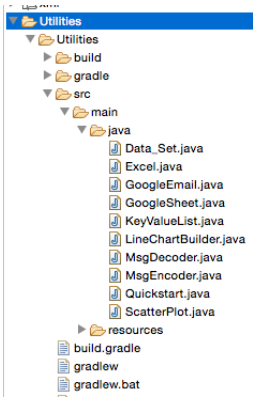


Figure 9: Utilities Library Classes

The Utilities Library provides a consistent place for all of the additional classes that are used among the various components. In order to use these classes, the class path should be included for both compilation and run.

4 Future Works and Conclusion

The SIS platform with the integration of Fitbit readings and a classifier component has the potential to expand its ability to classify by using ensemble learning models and semi-supervised learning. In the future, the target is to integrating these two features to the system to increase the efficacy of predicting values. Additionally, there exists the potential to expand the number of users, and to determine if this model, can adapt to the user rather than a generalization of potential users. Thus the main target for future works is to provide user specific classifications and increase the accuracy of detecting activities.

5 Deeds

1. Classification using Naive Bayes Classifier and Initial Clustering using Expectation Maximization

2. Uploads several different message, alerts of cumulative readings, summary, and classified readings
3. User interface that facilitates active learning by updating the training and classified readings file
4. Visualization of activity classifications relative to two key features
5. Utilities library that allows for classes to be stored in one location. This makes it easier to edit classes and keep the versions consistent across all components.

6 Appendix

The below code segments are the scripts that I developed, or code that I altered heavily from a base set of code. Code segments that are left out are those that may have been edited by myself, but were not altered enough to warrant being added to the file.

```
\subsection{Google Script for Raw Data}
\begin{lstlisting}
var d = new Date(), lastSavedTime, lastSavedDate;
var date, startTime, endTime;
// function setup accepts and stores the Consumer Key,
Consumer Secret, Project Key, firstDate, and list of Data Elements

//Trigger Function to Activate New Reading
function setup_trigger(){
    if (d.getHours()>7 && d.getHours()<22){ //checking between the hours of 6 am and 8 pm
      var h = d.getHours()-1;
      Logger.log(h);
      date = Utilities.formatDate(d, "EST", "yyyy-MM-dd");
      endTime = (h-1)+":"+"59";
      startTime = (h-1)+":"+"00";


      Logger.log(date);
      Logger.log(startTime);
      Logger.log(endTime);
      refreshTimeSeries();
   }
}


//Function for triggering the refresh of the series
function refreshTimeSeries() {
    var ss = SpreadsheetApp.getActiveSpreadsheet();
  if (!isConfigured()) {
    setup();
    return;
  }

    var options =
        {headers:{
        "Authorization": 'Bearer ' + getFitbitService().getAccessToken(),
        "method": "GET"
        }};
     // var activities = ["activities/steps","activities/calories","activities/elevation","activities/distance","activities/
```

```
     // var intradays = ["activities-steps-intraday","activities-calories-intraday","activities-elevation-intraday","activit
     var activities = ["activities/steps","activities/heart", "activities/elevation"];
     var intradays = ["activities-steps-intraday","activities-heart-intraday", "activities-elevation-intraday"];

      //GET https://api.fitbit.com/1/user/-/activities/heart/date/[date]/[end-date]/[detail-level].json

  var lastIndex = 0;
  var i=0;
    for (var activity in activities) {
      Logger.log(activities[i].split("/")[1]);
      var doc = ss.getSheetByName(activities[i].split("/")[1]);

      var index = 0;
    //var dateString = getFirstDate();
    //date = parseDate(dateString);

        var table = new Array();
          var currentActivity = activities[i];
          try {
            if (currentActivity == "activities/heart"){
                //var result = UrlFetchApp.fetch("https://api.fitbit.com/1/user/-/"+currentActivity+"/date/" + dateString+ "
                var result = UrlFetchApp.fetch("https://api.fitbit.com/1/user/-/"+currentActivity+"/date/" + date+ "/" + dat
              //Logger.log("Heart"+result);
            }else{
                //var result = UrlFetchApp.fetch("https://api.fitbit.com/1/user/-/"+currentActivity+"/date/" + dateString+ "
                var result = UrlFetchApp.fetch("https://api.fitbit.com/1/user/-/"+currentActivity+"/date/" + date+ "/" + dat
            }
          } catch(exception) {
              Logger.log(exception);
          }
          var o = JSON.parse(result.getContentText());


//          var titleCell = doc.getRange("a1");
//          titleCell.setValue("Date");
//          var title = currentActivity.split("/");
//          title = title[title.length - 1];

          //titleCell.offset(0, 2 + activity * 1.0).setValue(title);

            var row = o[intradays[i]]["dataset"];
          for (var j in row) {
              var val = row[j];
                var arr = new Array(3);
                arr[0] = date;
                arr[1] = val["time"];
                arr[2] = val["value"];
                table.push(arr);
              // set the value index index
               index++;
           }
          Logger.log("MADE IT HERE!");

          if(table.length!=0){
            var lastRow = doc.getLastRow();
            Logger.log("A"+(lastRow+1)+":C"+(table.length+lastRow));
            Logger.log(table.length);
            doc.getRange("A"+(lastRow+1)+":C"+(table.length+lastRow)).setValues(table);
          }
//          date.setDate(date.getDate()+1);
```

```
//          dateString = Utilities.formatDate(date, "GMT", "yyyy-MM-dd");

      //getRange(row, column, numRows, numColumns)
      // Batch set values of table, much faster than doing each time per loop run, this wouldn't work as is if there were mu

      i=i+1;
  }
}
\end{lstlisting}


\subsection{Processor Component}


\subsection{Aggregator Component}

\subsubsection{Process Message Additions}
\begin{lstlisting}
case "Reading":
double[] heart_raw;
String[] heart_time;
String[] step_dates;
String[] heart_dates;
String[] step_time;
double[] steps_raw;
double[] ele_raw;
double[][] aggregate;
switch (sender) {
case "Processor":
System.out.println("Received a message from the processor");
//SCRIPT for google is called here
System.out.println("New Readings!");
xls.openFile("../../Data/temp.xlsx");
xls.setSheet("heart");
heart_time = xls.readString(0,1);
heart_raw = xls.readDouble(0, 2);
heart_dates = xls.readString(0,0);

xls.setSheet("steps");
step_dates = xls.readString(0, 0);
step_time = xls.readString(0,1);
steps_raw = xls.readDouble(0,2);


xls.setSheet("elevation");
ele_raw = xls.readDouble(0,2);

xls.close();

Files.deleteIfExists((new File("../../Data/temp.xlsx")).toPath());

aggregate = reformat_rows(steps_raw, ele_raw, heart_raw, toMinutes(step_time), toMinutes(heart_time), step_dates, heart_date

aggregate = aggregate(aggregate, step_dates, 15.0);
step_dates = reformat_date_rows(step_dates);

Files.deleteIfExists((new File("../../Data/temp.arff").toPath()));

writeArff("../../Data/temp.arff",aggregate,step_dates);
```

```
//xls.createSheet("Aggregate");
//xls.setSheet("Aggregate");
//Aggregates all training data into sets of 15 minute time windows

//xls.write(step_dates,0,0);
//xls.write(aggregate, 0,1);
//xls.close();
System.out.println("Temporary Aggregate .arff Created");

record.putPair("Purpose",purpose);
record.putPair("IncomingMessages","New Raw Data");
record.putPair("OutgoingMessages","Aggregation is Complete");
encoder.sendMsg(record);
kvList.putPair("Sender", NAME);
kvList.putPair("Receiver", "Classifier");
encoder.sendMsg(kvList);
break;

case "SystemNotification":
xls.openFile("../../Data/TrainingData_Sept-1-30-2016.xlsx");
System.out.println("New Iteration!");

xls.setSheet("heart");
heart_time = xls.readString(1,1);
heart_raw = xls.readDouble(1, 2);
heart_dates = xls.readString(1,0);
xls.setSheet("steps");
step_dates = xls.readString(1, 0);
step_time = xls.readString(1,1);
steps_raw = xls.readDouble(1,2);
xls.setSheet("elevation");
ele_raw = xls.readDouble(1,2);

xls.close();

aggregate = reformat_rows(steps_raw, ele_raw, heart_raw, toMinutes(step_time), toMinutes(heart_time), step_dates, heart_date


//xls.setSheet("Aggregate");
//Aggregates all training data into sets of 15 minute time windows
aggregate = aggregate(aggregate, step_dates, 15.0);
step_dates = reformat_date_rows(step_dates);

writeArff("../../Data/training.arff",aggregate,step_dates);
//xls.write(step_dates,1,0);
//xls.write(aggregate, 1,1);
System.out.println("Training Aggregate .arff Created");
break;
}
break;
\end{lstlisting}

\subsubsection{Aggregation Related Code}
\begin{lstlisting}
private static double[][] simple_aggregate(double[][] raw,String[] dates, int window){
ArrayList<double[]> aggregate = new ArrayList<double[]>();
for (int row=0; row<raw.length; row=row+window){
double[] array = new double[3];
array[0] = raw[row][0]/60.0;
int heart_index=0;
```

```
String date = dates[row];
for(int i=row; i<row+window && i<raw.length; i++){
array[1]+=raw[i][1];
dates[i] = "del";
if(raw[i][2]>-1){
array[2] +=raw[i][2];
heart_index+=1;
}
}
if(heart_index>0){
dates[row] = date;
array[2]=array[2]/heart_index;
aggregate.add(array);
}
}//end row for=loop
return aggregate.stream().toArray(double[][]::new);
}//end of simple_aggregate

private static double[][] aggregate(double[][] raw,String[] dates, double window){
ArrayList<double[]> aggregate = new ArrayList<double[]>();
for (int row=0; row<raw.length; row=row+(int)window){
double[] array = new double[7];
array[0] = raw[row][0]/60.0;//time
int heart_index=0;
String date = dates[row];// adding date
for(int i=row; i<row+(int)window && i<raw.length; i++){
array[1]+=raw[i][1]; //steps
array[3]+=raw[i][2]; //elevation sum
dates[i] = "del"; // rows to remove

if(raw[i][3]>-1){
array[5] +=raw[i][3]; //summing heart value
heart_index+=1;
}
}
if(heart_index>0){ // if the heart index is used then keep it
dates[row] = date;
array[5]=array[5]/heart_index; //heart mean
//CALCULATING VARIANCE
double step_mean = array[1]/window;
double ele_mean = array[3]/window;

for(int i=row; i<row+(int)window && i<raw.length; i++){
array[2]+=Math.pow(raw[i][1]-step_mean,2); //step variance
array[4]+=Math.pow(raw[i][2]-ele_mean,2); //ele variance
if(raw[i][3]>-1)
array[6]+=Math.pow(raw[i][3]-array[5],2);//heart variance
}
array[2] = array[2]/window;
array[4] = array[4]/window;
array[6] = array[6]/heart_index;

aggregate.add(array);
}
}//end row for=loop
return aggregate.stream().toArray(double[][]::new);
}//end of simple_aggregate
```

```
private static String[] reformat_date_rows(String[] dates){
ArrayList<String> array = new ArrayList<String>();
for (int row=0; row<dates.length; row++){
if(!dates[row].equals("del")){
array.add(dates[row]);
}
}
return array.stream().toArray(String[] :: new);
}

private static double[][] reformat_rows(double[] steps, double[] elev, double[] heart, double[] sTime, double[]hTime, String
ArrayList<double[]> formatted_array = new ArrayList<double[]>();
int heart_index = 0;

for (int i=0; i<steps.length; i++){
if(heart_index<heart.length &&sTime[i]==hTime[heart_index] && step_dates[i].equals(heart_dates[heart_index])){
double[] array = {sTime[i],steps[i], elev[i],heart[heart_index]};
formatted_array.add(array);
heart_index+=1;
}else{
double[] array = {sTime[i],steps[i],elev[i],-1.0};
formatted_array.add(array);
}
}

return formatted_array.stream().toArray(double[][]::new);
}

private static double[] toMinutes(String[] array){
double[] time = new double[array.length];

for (int i=0; i<array.length;i++){
if(array[i].length()>8){
String hours = array[i].split(":")[0];
hours = hours.substring(hours.length()-2, hours.length());
time[i] = Double.parseDouble(hours)*60.0 +
Double.parseDouble(array[i].split(":")[1]);
}else{
time[i] = Double.parseDouble(array[i].split(":")[0])*60.0 +
Double.parseDouble(array[i].split(":")[1]);
}
}

return time;
}

private static double[] toHours(double[] array){
for (int i=0; i<array.length;i++){
array[i] = array[i]/60.0;
}
return array;
}
/*
 * process a certain message, execute corresponding actions
 */

\end{lstlisting}

\subsubsection{WEKA Writing Related Code}
```

```
\begin{lstlisting}
static void writeArff(String fileName, double[][] data, String[] dates){
try{
ArffSaver saver = new ArffSaver();

FastVector fv = new FastVector();
fv.addElement(dates[0]);
for (int i=1; i<dates.length; i++){
if(!fv.lastElement().equals(dates[i])){
fv.addElement(dates[i]);
}
}
FastVector atts = new FastVector();
atts.addElement(new Attribute("Date",fv));
atts.addElement(new Attribute("Time"));
atts.addElement(new Attribute("S-Sum"));
atts.addElement(new Attribute("S-Var"));
atts.addElement(new Attribute("E-Sum"));
atts.addElement(new Attribute("E-Var"));
atts.addElement(new Attribute("H-Mean"));
atts.addElement(new Attribute("H-Var"));



Instances instances = new Instances("Data",atts,data.length);
System.out.println(atts.toString());
System.out.println(atts.size());
System.out.println(data[0].length);
for(int i=0; i<data.length; i++){
System.out.println(Arrays.toString(data[i]));
instances.add(new Instance(8));
instances.instance(i).setValue(0,dates[i]);
instances.instance(i).setValue(1,data[i][0]);
instances.instance(i).setValue(2,data[i][1]);
instances.instance(i).setValue(3,data[i][2]);
instances.instance(i).setValue(4,data[i][3]);
instances.instance(i).setValue(5,data[i][4]);
instances.instance(i).setValue(6,data[i][5]);
instances.instance(i).setValue(7,data[i][6]);
}



saver.setInstances(instances);
saver.setFile(new File(fileName));
saver.writeBatch();
}catch(Exception e){
e.printStackTrace();
}
}

\end{lstlisting}


\subsection{Classifier Component}

\subsubsection{Process Message Additions}
\begin{lstlisting}
case "Aggregator":
switch (purpose){
```

```
case "NewReading":
WekaML.supervised_training_iteration();
WekaML.classify();
record.putPair("Purpose","Reading");
record.putPair("IncomingMessages","New Aggregated Data");
record.putPair("OutgoingMessages","Classification is Complete");
encoder.sendMsg(record);
kvList.putPair("Sender", NAME);
kvList.putPair("Receiver", "UploaderFitbit");
encoder.sendMsg(kvList);
break;
case "Iteration":
WekaML.supervised_training_iteration();
System.out.println("Supervised Training Iteration Complete");
break;
case "InitialIteration":
WekaML.unsupervised_training_iteration();
System.out.println("Unsupervised Training Complete");
break;
}
}
\end{lstlisting}

\subsubsection{WEKA Classification}
\begin{lstlisting}
public static class WekaML{
static EM em;
static NaiveBayes nbayes;
//static J48 dectree;
//static Logistic logistic_reg;
static Vote vote_ensemble;
static Instances training_data_EM, training_data_supervised;
public WekaML(){

}
public static void unsupervised_training_iteration(){
try{
System.out.println("Training Unsupervised Iteration!");
DataSource source = new DataSource("../../Data/training.arff");
training_data_EM = source.getDataSet();
int[] colsToUse = {2, 3, 4, 5, 6, 7};
Remove remove = new Remove();
remove.setAttributeIndicesArray(colsToUse);
remove.setInvertSelection(true);
remove.setInputFormat(training_data_EM);
Instances newData = Filter.useFilter(training_data_EM, remove);

em = new EM();
em.setNumClusters(14);
System.out.println(Arrays.toString(em.getOptions()));
em.buildClusterer(newData);
FastVector classes = new FastVector();
for (int i=0; i<em.numberOfClusters(); i++){
classes.addElement("act"+i);
}

training_data_EM.insertAttributeAt(new Attribute("class",classes), training_data_EM.numAttributes());
for (int i=0; i<newData.numInstances(); i++){
int cluster = em.clusterInstance(newData.instance(i));
```

```
training_data_EM.instance(i).setValue(training_data_EM.numAttributes()-1, cluster);
}
writeArff(training_data_EM,"../../Data/training_classified.arff");

System.out.println("Finished Training!");



}catch(Exception e){
e.printStackTrace();
}

}// end unsupervised_training iteration

public static void supervised_training_iteration(){
try{
DataSource source = new DataSource("../../Data/training_classified.arff");
training_data_supervised = source.getDataSet();
int[] colsToUse = {2, 3, 4, 5, 6, 7, 8};
Remove remove = new Remove();
remove.setAttributeIndicesArray(colsToUse);
remove.setInvertSelection(true);
remove.setInputFormat(training_data_supervised);
Instances newData = Filter.useFilter(training_data_supervised, remove);
newData.setClass(newData.attribute("class"));
newData.setClassIndex(newData.attribute("class").index());

nbayes = new NaiveBayes();
nbayes.buildClassifier(newData);
//dectree = new J48();

//dectree.buildClassifier(newData);

//logistic_reg = new Logistic();
//logistic_reg.buildClassifier(newData);
//Classifier[] classifiers = {nbayes,dectree,logistic_reg};

// vote_ensemble = new Vote();
// vote_ensemble.setClassifiers(classifiers);
// vote_ensemble.buildClassifier(newData);
}catch(Exception e){
e.printStackTrace();
}
}// j48 classifying tree

public static void classify(){
try{
System.out.println("Classifying!");
DataSource source = new DataSource("../../Data/temp.arff");
Instances data = source.getDataSet();
int[] colsToUse = {2, 3};
Remove remove = new Remove();
remove.setAttributeIndicesArray(colsToUse);
remove.setInvertSelection(true);
remove.setInputFormat(data);
Instances newData = Filter.useFilter(data, remove);
Attribute att_class = training_data_supervised.attribute("class");
newData.insertAttributeAt(att_class, newData.numAttributes());
data.insertAttributeAt(att_class, data.numAttributes());
newData.setClassIndex(newData.attribute("class").index());
```

```
String class_value = "";
int counter = 0;
double start_time=-1.0;
int index=-1;
String date = data.instance(0).stringValue(data.attribute("Date"));

index = (int)nbayes.classifyInstance(newData.instance(0));
data.instance(0).setValue(att_class, att_class.value(index));
if(att_class.value(index).equals("strenuousrunning")||att_class.value(index).equals("sitting")){
counter+=1;
start_time = data.instance(0).value(data.attribute("Time"));
class_value = att_class.value(index);
}

for (int i=1; i<newData.numInstances(); i++){
index = (int)nbayes.classifyInstance(newData.instance(i));
data.instance(i).setValue(att_class, att_class.value(index));



if(class_value.equals(att_class.value(index)) && date.equals(data.instance(i).stringValue(data.attribute("Date")))){
counter+=1;
}else{
class_value = att_class.value(index);
start_time = data.instance(i).value(data.attribute("Time"));
counter=0;
date = data.instance(i).stringValue(data.attribute("Date"));
}

if(counter>1){
switch(class_value){
case "strenuousrunning":
send_message(data.instance(i).stringValue(data.attribute("Date")),
start_time,
data.instance(i).value(data.attribute("Time")),
running_message);
break;
case "sitting":
System.out.println("Date: "+date+" Start time "+start_time+" End time"+data.instance(i).value(data.attribute("Time"))+" "+s
send_message(data.instance(i).stringValue(data.attribute("Date")),
start_time,
data.instance(i).value(data.attribute("Time")),
sitting_message);
break;
}
class_value = att_class.value(index);
start_time = data.instance(i).value(data.attribute("Time"));
if(class_value.equals(att_class.value(index))){
counter=1;
}else{
counter=0;
}
date = data.instance(i).stringValue(data.attribute("Date"));
}


}
writeArff(data, "../../Data/temp_classified.arff");
```

```
System.out.println("Finished Classifying!");


}catch(Exception e){
e.printStackTrace();
}
}//end classify

static void send_message(String date, double start_time, double end_time, String message){
alert.putPair("Message", date+":"+start_time+":"+end_time+":"+message);
alert.putPair("Receiver", "UploaderFitbit");
try{
encoder.sendMsg(alert);
}catch(IOException e){
e.printStackTrace();
}
}

static void writeArff(Instances instances, String fileName){
try{
ArffSaver saver = new ArffSaver();

saver.setInstances(instances);
saver.setFile(new File(fileName));
saver.writeBatch();
}catch(IOException e){
e.printStackTrace();
}
}


}//end WEKAML
\end{lstlisting}


\subsection{Uploader Component}

\subsubsection{Process Message Additions}
\begin{lstlisting}
case "Reading":
switch (sender) {
case "Classifier":
System.out.println("Uploading...");
GoogleSheet gs = new GoogleSheet("classified_data");
gs.appendData("temp_classified.arff");
System.out.println("Updated GSpreadsheet!");
try{
DataSource source = new DataSource("../../Data/temp_classified.arff");
Instances data = source.getDataSet();
GoogleEmail gm = new GoogleEmail();
gm.sendMsg(reading.recipients,
"fitbitMattB@gmail.com",
createSubject(data),
createBody(data));
upload_new_readings(data);
}catch(Exception e){
e.printStackTrace();
}
record.putPair("Purpose",purpose);
record.putPair("IncomingMessages","New Classified Data");
```

```
record.putPair("OutgoingMessages","Upload to Google Sheet, Email Sent, Submitted to Chronobot is Complete");
encoder.sendMsg(record);
kvList.putPair("Sender", NAME);
kvList.putPair("Receiver", "GraphUI");
encoder.sendMsg(kvList);
}
break;
\end{lstlisting}

\begin{lstlisting}
        case "Alert":
            System.out.println("\n*** Alert from "+sender+" ***");
            switch (sender)
            {
            case "Classifier":
             alert.putPair("IncomingMessages", kvList.getValue("Message"));
             alert.putPair("OutgoingMessages", "Uploading Alert: "+kvList.getValue("Message"));

             update_alert(kvList.getValue("Message"));

             encoder.sendMsg(alert);
             break;
\end{lstlisting}

\subsubsection{Uploader Alert Specific Code}
\begin{lstlisting}

private static void update_alert(String content){
String[] data = content.split(":");
reading.date_string = data[0];
reading.end_time = Double.parseDouble(data[2])+0.25;
reading.convertDateToLong();
reading.start_time = Double.parseDouble(data[1]);
reading.message = data[3];
try{
upload_alert();
}catch(Exception e){
e.printStackTrace();
}

String subject = "FitbitAlert: "+data[0]+": "+ reading.message;
String message = "FitbitAlert "+data[0]+"\n"
+"From: "+reading.start_time + " To:"+reading.end_time+"\n"
+"Message: "+reading.message;


GoogleEmail gm = new GoogleEmail();
gm.sendMsg(reading.recipients,
"fitbitMattB@gmail.com",
subject,
message);


}//end update_alert

    private static void upload_alert() throws Exception
    {

        System.out.println(reading);
```

```
        System.out.println("Updating DB...");
        execute(formQuery(reading.date,"Fitbit","message",reading.message));


//        execute(formQuery(reading.dateSPO2,"SPO2","spo2",reading.spo2));
//        execute(formQuery(reading.dateBP,"BloodPressure","systolic",reading.systolic));
//        execute(formQuery(reading.dateBP,"BloodPressure","diastolic",reading.diastolic));
//        execute(formQuery(reading.dateBP,"BloodPressure","pulse",reading.pulse));
//        execute(formQuery(reading.dateEKG,"EKG","ekg",reading.ekg));
        //execute(formQuery(reading.dateTemp,"Temperature","temp",reading.temp));

        System.out.println("DB Updated");
        // sendSSLMessage(sendTo, emailSubjectTxt, emailMsgTxt,
        // emailFromAddress, lname, fname, spo2);
    }


\end{lstlisting}

\subsubsection{Uploader Reading Specific Code}
\begin{lstlisting}

public static void upload_new_readings(Instances data) throws Exception{

Attribute dateAtt = data.attribute("Date");
Attribute timeAtt = data.attribute("Time");
Attribute stepAtt = data.attribute("S-Sum");
Attribute heartAtt = data.attribute("H-Mean");
Attribute classAtt = data.attribute("class");


for (int i=0; i<data.numInstances(); i++){
reading.date_string = data.instance(i).stringValue(dateAtt);
reading.convertDateToLong();
reading.time = data.instance(i).value(timeAtt);
reading.steps = data.instance(i).value(stepAtt);
reading.heart_rate = data.instance(i).value(heartAtt);
reading.activity = data.instance(i).stringValue(classAtt);
}//normally update would need to be in the for loop by this is for test
update();

}

\end{lstlisting}

\subsubsection{Uploading Email Specific Code}
\begin{lstlisting}
public static String getTime(double time){
int mins = (int)((time*60.0)%60);
if(mins==0)
return (int)(time%12)+":00";
else if (mins<10)
return (int)(time%12)+":0"+mins;
else
return (int)(time%12)+":"+mins;
}

public static String createSubject(Instances data){

if (data.firstInstance().stringValue(0).equals(data.lastInstance().stringValue(0)))
```

```
return "Activity from "+data.firstInstance().stringValue(0)+" @ "+getTime(data.firstInstance().value(1))
 +" to "+ getTime(data.lastInstance().value(1));
else
return "Activity from "+data.firstInstance().stringValue(0)+" @ "+getTime(data.firstInstance().value(1))
+" to "+data.lastInstance().stringValue(0)+" @ "+ getTime(data.lastInstance().value(1));
}

public static String createBody(Instances data){
String body="";
Attribute classAtt = data.attribute("class");
Attribute dateAtt = data.attribute("Date");
double currentClass = data.instance(0).value(classAtt);
double time = data.instance(0).value(data.attribute("Time"));
double currentDate = data.instance(0).value(dateAtt);
int start_index=0;
body+= dateAtt.value((int)data.instance(0).value(dateAtt))+" \n";

for (int i=1; i<data.numInstances(); i++){
if (data.instance(i).value(classAtt)!=currentClass || currentDate!=data.instance(i).value(dateAtt)){
if(i-1==start_index){
body+="@"+getTime(time)+" Activity: "+classAtt.value((int)currentClass)+" \n";
}else{
body+="@"+getTime(time)+" to "+getTime(data.instance(i-1).value(data.attribute("Time")))+" Activity: "+classAtt.value((int)c
}
if(currentDate!=data.instance(i).value(dateAtt)){
body+= dateAtt.value((int)data.instance(i).value(dateAtt))+" \n";
currentDate = data.instance(i).value(dateAtt);
}

start_index=i;
time=data.instance(i).value(data.attribute("Time"));
currentClass= data.instance(i).value(data.attribute("class"));
}
}
if(start_index==data.numInstances()-1){
body+="@"+getTime(time)+" Activity: "+classAtt.value((int)currentClass)+" \n";
}else{
body+="@"+getTime(time)+" to "+getTime(data.instance(data.numInstances()-1).value(data.attribute("Time")))+" Activity: "+cla
}
System.out.println(body);
return body;
}

\end{lstlisting}


\subsection{GraphUI Component}

\subsubsection{Process Message Additions}
\begin{lstlisting}
case "Reading":
switch (sender) {
case "UploaderFitbit"
+ "":
System.out.println("Starting up the graph ui");
Process process = Runtime.getRuntime().exec("java -cp .:../../Utilities/Utilities/Utilities-1.0.jar UIDataViewer new");
InputStreamReader is =new InputStreamReader(process.getErrorStream());
char[] b;
do{
b = new char[128];
```

```
is.read(b);
System.out.println(String.valueOf(b));

}while(process.isAlive());

System.out.println("Exited Graph");
record.putPair("IncomingMessages","New Classified Data");
record.putPair("OutgoingMessages","Classification Graph");
encoder.sendMsg(record);

}
\end{lstlisting}


\subsection{User Interface Data Viewer Component}

\subsubsection{Graph Initialization and Listening Handlers}
\begin{lstlisting}
    public static void main(String[] args) {
     if(args.length>0){
     newReading = true;
     }else{
     newReading = false;
     }

        launch(args);
    }

    @Override
    public void start(Stage s) {
        scene = new Scene(new Group());
        stage = s;
        stage.setTitle("Fitbit Classifier Interface");
        stage.setWidth(1100);
        stage.setHeight(550);

        tabgraph = new HBox();
        buttons = new HBox();
        graphs = new VBox();

        saved = new Label("Data Saved");

        dimensions = new Dimensions[3];
        if(newReading){
         getrecentdate();
         update_table(date);
        }else{
         data = FXCollections.observableArrayList(
                    new DataSet(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, "<activity>"));
        }


        svtXAxis = new NumberAxis("Time", 5,20,0.25);
        svtYAxis = new NumberAxis("Steps", -10,1500,100);
        stepVtime =
                new ScatterChart<Number,Number>(svtXAxis, svtYAxis);
        stepVtime.setMaxWidth(550);


        hvtXAxis = new NumberAxis("Time", 5,20,0.25);
```

```
hvtYAxis = new NumberAxis("Heart", 0,170,40);
heartVtime =
        new ScatterChart<Number,Number>(hvtXAxis, hvtYAxis);
heartVtime.setMaxWidth(550);

svhXAxis = new NumberAxis("Heart", 0,170,40);
svhYAxis = new NumberAxis("Steps", -10,1500,100);
stepVheart =
        new ScatterChart<Number,Number>(svhXAxis, svhYAxis);
stepVheart.setMaxWidth(550);


stepVtime.getData().addAll(series_builderSVT(data.iterator(), 1, 2));
heartVtime.getData().addAll(series_builderHVT(data.iterator(), 1, 3));

stepVheart.getData().addAll(series_builderSVH(data.iterator(), 3, 2));

dimensions[0] = (new Dimensions(stepVtime, "Steps vs. Time"));
dimensions[1] = (new Dimensions(heartVtime, "Heart Rate vs. Time"));
dimensions[2] = (new Dimensions(stepVheart, "Steps vs. Heart Rate"));

scroll_table = new ScrollPane();
scroll_table.setVbarPolicy(ScrollBarPolicy.ALWAYS);
scroll_table.setMaxWidth(600);
scroll_table.setMaxHeight(400);


scroll_graph = new ScrollPane();
scroll_graph.setVbarPolicy(ScrollBarPolicy.ALWAYS);
scroll_graph.setMaxWidth(600);
scroll_graph.setMaxHeight(400);
label.setFont(new Font("Arial", 20));

table.setEditable(true);

TableColumn timeCol = new TableColumn("Time");
timeCol.setMinWidth(100);


timeCol.setCellValueFactory(
    new PropertyValueFactory<DataSet,Number>("time"));
timeCol.setEditable(false);



TableColumn stepsCol = new TableColumn("Steps");
stepsCol.setMinWidth(100);
stepsCol.setCellValueFactory(
    new PropertyValueFactory("steps"));
stepsCol.setEditable(false);

TableColumn stepsVarCol = new TableColumn("StepsVar");
stepsVarCol.setMinWidth(100);
stepsVarCol.setCellValueFactory(
    new PropertyValueFactory("stepsVar"));
stepsVarCol.setEditable(false);


TableColumn eleCol = new TableColumn("Elevation");
eleCol.setMinWidth(100);
```

```
  eleCol.setCellValueFactory(
      new PropertyValueFactory("elevation"));
  eleCol.setEditable(false);

  TableColumn eleVarCol = new TableColumn("ElevationVar");
  eleVarCol.setMinWidth(100);
  eleVarCol.setCellValueFactory(
      new PropertyValueFactory("elevationVar"));
  eleVarCol.setEditable(false);

  TableColumn heartRateCol = new TableColumn("Heart Rate");
  heartRateCol.setMinWidth(100);
  heartRateCol.setCellValueFactory(
      new PropertyValueFactory("heartRate"));
  timeCol.setEditable(false);

  TableColumn heartVarCol = new TableColumn("HeartVar");
  heartVarCol.setMinWidth(100);
  heartVarCol.setCellValueFactory(
      new PropertyValueFactory("heartVar"));
  heartVarCol.setEditable(false);


  TableColumn activityCol = new TableColumn("Activities");
  activityCol.setMinWidth(150);
  activityCol.setCellValueFactory(
      new PropertyValueFactory("activity"));
  activityCol.setCellFactory(TextFieldTableCell.forTableColumn());
  activityCol.setOnEditCommit(
      new EventHandler<CellEditEvent<DataSet, String>>() {
          @Override
          public void handle(CellEditEvent<DataSet, String> t) {
              ((DataSet) t.getTableView().getItems().get(
                  t.getTablePosition().getRow())
                  ).setActivity(t.getNewValue());
              saved.setText("Unsaved Data");
              update_graphs();
              update();
          }
      }
  );

  table.setItems(data);
  table.getColumns().addAll(activityCol, timeCol, stepsCol, heartRateCol, eleCol,
   stepsVarCol, heartVarCol, eleVarCol);


  addDate = new TextField();
  addDate.setMaxWidth(activityCol.getPrefWidth());
  if(date!=null){
   addDate.setPromptText(date);
  }else{
   addDate.setPromptText("YYYY-MM-DD");
  }
  addDate.setMinWidth(110);


dateButton = new Button("Load Date");
dateButton.setOnAction(new EventHandler<ActionEvent>() {
 @Override
```

```
        public void handle(ActionEvent e) {
        date = addDate.getText();
        addDate.setText(date);
        update_table(date);
        update_graphs();
        update();
      }
       });

       saveButton = new Button ("Save Changes");
       saveButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
        saved.setText("Saving...");
        update();
        update_spreadsheet();
        saved.setText("Data Saved");
        update();
      }
       });

       graphButton = new Button ("Build Graph");
       graphButton.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
        update_graphformat(dimensions[0],svtXAxis, svtYAxis);
        update_graphformat(dimensions[1],hvtXAxis, hvtYAxis);
        update_graphformat(dimensions[2],svhXAxis, svhYAxis);
        update();
      }
       });

       graphs.getChildren().addAll(dimensions[0].vbox,stepVtime,
        dimensions[1].vbox, heartVtime,
        dimensions[2].vbox, stepVheart);
       scroll_graph.setContent(graphs);

       scroll_table.setContent(table);

       tabgraph.getChildren().addAll(scroll_table, scroll_graph);
       tabgraph.setMinHeight(450);

       buttons.setSpacing(10);
       buttons.getChildren().addAll(addDate, dateButton, graphButton, saveButton, saved);
     vbox = new VBox();
     vbox.setPadding(new Insets(10, 0, 0, 10));
     vbox.getChildren().addAll(label, buttons,tabgraph);
         vbox.setMinHeight(700);
     ((Group) scene.getRoot()).getChildren().addAll(vbox);



         stage.setScene(scene);
         stage.show();
     }
\end{lstlisting}

\subsubsection{Graph, Table, and File Updating}
\begin{lstlisting}
    public void update(){
```

```
    table.setItems(data);
    stage.setScene(scene);
    stage.show();

   // ((Group) scene.getRoot()).getChildren().addAll(vbox);
    stage.setScene(scene);
    stage.show();

}


public void update_table(String date){
 System.out.println("Updating Table with "+date);

 gs = new GoogleSheet("classified_data");
    Data_Set[] data_vals = null;
    ArrayList<DataSet> dataset = new ArrayList<DataSet>();

    try{
     data_vals = gs.getClassifiedData(date);
    }catch(IOException e){
     e.printStackTrace();
    }
    //(Double t, Double s,Double sv, Double e, Double ev,
    // Double hr, Double hrv, String activity)
    for(int i=0; i<data_vals.length; i++){
     if(date.equals(data_vals[i].date)){
     dataset.add(new DataSet(data_vals[i].time,
     data_vals[i].steps, data_vals[i].stepsv,
     data_vals[i].elevation, data_vals[i].elevationv,
     data_vals[i].heartrate, data_vals[i].heartratev,
     data_vals[i].activity));
     }
    }
 data = FXCollections.observableArrayList(dataset);
}

public void update_spreadsheet(){
 Iterator<DataSet> iter = data.iterator();
 ArrayList<DataSet> changedList = new ArrayList<DataSet>();
 int index=0;
 while(iter.hasNext()){
 DataSet val = iter.next();
 if(val.changed){
 Data_Set value = new Data_Set(date,
 val.getTime(),
 val.getSteps(), val.getStepsVar(),
 val.getElevation(), val.getElevationVar(),
 val.getHeartRate(), val.getHeartVar(),
 val.getActivity());
    try{
     gs.writeClassifiedData(value);

    }catch(IOException e){
     e.printStackTrace();
    }
    data.get(index).changed = false;
    changedList.add(val);
 }
 index+=1;
```

```
     }
     writeToArff(changedList);
    }

    public void writeToArff(ArrayList<DataSet> new_vals){
try{
DataSource source = new DataSource("../../Data/training_classified.arff");
Instances data = source.getDataSet();

Instance inst;
Attribute timeAtt = data.attribute("Time");
Attribute dateAtt = data.attribute("Date");
Attribute stepsAtt = data.attribute("S-Sum");
Attribute stepsVAtt = data.attribute("S-Var");
Attribute eleAtt = data.attribute("E-Sum");
Attribute eleVAtt = data.attribute("E-Var");
Attribute heartAtt = data.attribute("H-Mean");
Attribute heartVAtt = data.attribute("H-Var");

Attribute classAtt = data.attribute("class");

FastVector fvclass = new FastVector();
for(int i=0; i<classAtt.numValues(); i++)
fvclass.addElement(classAtt.value(i));

for (int i=0; i<new_vals.size(); i++){
if(!fvclass.contains(new_vals.get(i).getActivity())){
fvclass.addElement(new_vals.get(i).getActivity());
}
}

FastVector fvdate = new FastVector();
for(int i=0; i<dateAtt.numValues(); i++)
fvdate.addElement(dateAtt.value(i));

if(!fvdate.contains(date)){
fvdate.addElement(date);
}



FastVector atts = new FastVector();
atts.addElement(new Attribute("Date",fvdate));
atts.addElement(new Attribute("Time"));

atts.addElement(new Attribute("S-Sum"));
atts.addElement(new Attribute("S-Var"));

atts.addElement(new Attribute("E-Sum"));
atts.addElement(new Attribute("E-Var"));

atts.addElement(new Attribute("H-Mean"));
atts.addElement(new Attribute("H-Var"));

atts.addElement(new Attribute("class",fvclass));



Instances instances = new Instances("Data",atts,data.numInstances()+new_vals.size());
```

```
Attribute ntimeAtt = instances.attribute("Time");
Attribute ndateAtt = instances.attribute("Date");
Attribute nstepsAtt = instances.attribute("S-Sum");
Attribute nstepsVAtt = instances.attribute("S-Var");

Attribute neleAtt = instances.attribute("E-Sum");
Attribute neleVAtt = instances.attribute("E-Var");

Attribute nheartAtt = instances.attribute("H-Mean");
Attribute nheartVAtt = instances.attribute("H-Var");

Attribute nclassAtt = instances.attribute("class");
int replace=0;
for(int i=0; i<data.numInstances(); i++){
inst = new Instance(atts.size());
if((replace = check_inst(data.instance(i),ndateAtt, ntimeAtt, new_vals))==-1){
inst.setValue(ntimeAtt, data.instance(i).value(timeAtt));
inst.setValue(ndateAtt, data.instance(i).stringValue(dateAtt));

inst.setValue(nstepsAtt, data.instance(i).value(stepsAtt));
inst.setValue(nstepsVAtt, data.instance(i).value(stepsVAtt));

inst.setValue(neleAtt, data.instance(i).value(eleAtt));
inst.setValue(neleVAtt, data.instance(i).value(eleVAtt));

inst.setValue(nheartAtt, data.instance(i).value(heartAtt));
inst.setValue(nheartVAtt, data.instance(i).value(heartVAtt));

inst.setValue(nclassAtt, data.instance(i).stringValue(nclassAtt));
}else{
inst.setValue(ndateAtt, date);
inst.setValue(ntimeAtt, new_vals.get(replace).getTime());

inst.setValue(nstepsAtt, new_vals.get(replace).getSteps());
inst.setValue(nstepsVAtt, new_vals.get(replace).getStepsVar());

inst.setValue(neleAtt, new_vals.get(replace).getElevation());
inst.setValue(neleVAtt, new_vals.get(replace).getElevationVar());

inst.setValue(nheartAtt, new_vals.get(replace).getHeartRate());
inst.setValue(nheartVAtt, new_vals.get(replace).getHeartVar());

inst.setValue(nclassAtt, new_vals.get(replace).getActivity());
new_vals.remove(replace);
}
instances.add(inst);
}


for(DataSet val : new_vals){
inst = new Instance(data.numAttributes());

inst.setValue(ndateAtt, date);
inst.setValue(ntimeAtt, val.getTime());

inst.setValue(nstepsAtt, val.getSteps());
inst.setValue(nstepsVAtt, val.getStepsVar());

inst.setValue(neleAtt, val.getElevation());
```

```
inst.setValue(neleVAtt, val.getElevationVar());


inst.setValue(nheartAtt, val.getHeartRate());
inst.setValue(nheartVAtt, val.getHeartVar());


inst.setValue(nclassAtt, val.getActivity());
instances.add(inst);


}
ArffSaver saver = new ArffSaver();

saver.setInstances(instances);
saver.setFile(new File("../../Data/training_classified.arff"));
saver.writeBatch();

}catch(Exception e){
e.printStackTrace();
}

    }

    public int check_inst(Instance inst, Attribute d, Attribute time, ArrayList<DataSet> new_vals){

     for(int i=0; i<new_vals.size(); i++){
     if(inst.value(time)==new_vals.get(i).getTime() && inst.stringValue(d).equals(date)){
     return i;
     }
     }

     return -1;
    }

   public void getrecentdate(){
   try{
DataSource source = new DataSource("../../Data/temp_classified.arff");
Instances data = source.getDataSet();
Attribute dateatt = data.attribute("Date");
date = data.lastInstance().stringValue(dateatt);
   }catch(Exception e){

   }
    }

    public Collection <XYChart.Series<Number, Number>> series_builderSVT(Iterator<DataSet> data, int time, int y){
     HashMap<String,XYChart.Series<Number, Number>> series_set = new HashMap<String,XYChart.Series<Number, Number>>();
     while(data.hasNext()){
     DataSet vals = data.next();
         if(!series_set.containsKey(vals.getActivity())){
         series_set.put(vals.getActivity(), new XYChart.Series());
         series_set.get(vals.getActivity()).setName(vals.getActivity());
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getTime(),vals.getSteps()));
         }else{
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getTime(),vals.getSteps()));
         }
        }

        return series_set.values();
```

```
    }

    public Collection <XYChart.Series<Number, Number>> series_builderHVT(Iterator<DataSet> data, int time, int y){
     HashMap<String,XYChart.Series<Number, Number>> series_set = new HashMap<String,XYChart.Series<Number, Number>>();
     while(data.hasNext()){
     DataSet vals = data.next();
         if(!series_set.containsKey(vals.getActivity())){
         series_set.put(vals.getActivity(), new XYChart.Series());
         series_set.get(vals.getActivity()).setName(vals.getActivity());
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getTime(),vals.getHeartRate()));
         }else{
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getTime(),vals.getHeartRate()));
         }
        }

        return series_set.values();
    }

    public Collection <XYChart.Series<Number, Number>> series_builderSVH(Iterator<DataSet> data, int time, int y){
     HashMap<String,XYChart.Series<Number, Number>> series_set = new HashMap<String,XYChart.Series<Number, Number>>();
     while(data.hasNext()){
     DataSet vals = data.next();
         if(!series_set.containsKey(vals.getActivity())){
         series_set.put(vals.getActivity(), new XYChart.Series());
         series_set.get(vals.getActivity()).setName(vals.getActivity());
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getHeartRate(),vals.getSteps()));
         }else{
         series_set.get(vals.getActivity()).getData().add(new XYChart.Data(vals.getHeartRate(),vals.getSteps()));
         }
        }

        return series_set.values();
    }

    public void update_graphs(){

     stepVtime.getData().clear();
        stepVtime.getData().addAll(series_builderSVT(data.iterator(), 1, 2));
        dimensions[0].resetFields(stepVtime);

     heartVtime.getData().clear();
        heartVtime.getData().addAll(series_builderHVT(data.iterator(), 1, 2));
        dimensions[1].resetFields(heartVtime);

     stepVheart.getData().clear();
        stepVheart.getData().addAll(series_builderSVH(data.iterator(), 3, 2));
        dimensions[2].resetFields(stepVheart);

    }

    public void update_graphformat(Dimensions dims, NumberAxis xaxis, NumberAxis yaxis){

xaxis.upperBoundProperty().set(Double.parseDouble(dims.max_x.getText()));
xaxis.lowerBoundProperty().set(Double.parseDouble(dims.min_x.getText()));

yaxis.upperBoundProperty().set(Double.parseDouble(dims.max_y.getText()));
yaxis.lowerBoundProperty().set(Double.parseDouble(dims.min_y.getText()));

xaxis.setTickUnit(Double.parseDouble(dims.x_intvl.getText()));
yaxis.setTickUnit(Double.parseDouble(dims.y_intvl.getText()));
```

```
}
\end{lstlisting}


\subsubsection{Dimensions Class}
\begin{lstlisting}
    public class Dimensions{
     HBox hbox;
     VBox[] dims;
     VBox vbox;
     public Label chart_title;

     public Label label_max_x;
     public Label label_min_x;

     public Label label_max_y;
     public Label label_min_y;

     public Label label_x_intvl;
     public Label label_y_intvl;

     public TextField max_x;
     public TextField min_x;

     public TextField max_y;
     public TextField min_y;

     public TextField x_intvl;
     public TextField y_intvl;
     private static final double SIZE = 50;

     public Dimensions(ScatterChart<Number,Number> chart, String title){
     dims = new VBox[6];
     hbox = new HBox();
     vbox = new VBox();

     chart_title = new Label();
     chart_title.setText(title);
            chart_title.setFont(new Font("Arial", 20));


     label_max_x = new Label("Max X");
     label_min_x = new Label("Min X");

     label_max_y = new Label("Max Y");
     label_min_y = new Label("Min Y");

     label_x_intvl = new Label("X Scale");
     label_y_intvl = new Label("Y Scale");




     max_x = new TextField();
     min_x = new TextField();
     max_y = new TextField();
     min_y = new TextField();
     x_intvl = new TextField();
     y_intvl = new TextField();
```

```
        max_x.setMaxWidth(SIZE);
        min_x.setMaxWidth(SIZE);
        max_y.setMaxWidth(SIZE);
        min_y.setMaxWidth(SIZE);
        x_intvl.setMaxWidth(SIZE);
        y_intvl.setMaxWidth(SIZE);

        NumberAxis xaxis = (NumberAxis)chart.getXAxis();
        NumberAxis yaxis = (NumberAxis)chart.getYAxis();

        max_x.setText(String.valueOf(xaxis.upperBoundProperty().doubleValue()));
        min_x.setText(String.valueOf(xaxis.lowerBoundProperty().doubleValue()));
        max_y.setText(String.valueOf(yaxis.upperBoundProperty().doubleValue()));
        min_y.setText(String.valueOf(yaxis.lowerBoundProperty().doubleValue()));
        x_intvl.setText(String.valueOf(xaxis.getTickUnit()));
        y_intvl.setText(String.valueOf(yaxis.getTickUnit()));

        dims[0] = (new VBox());
        dims[0].getChildren().addAll(label_max_x, max_x);

        dims[1] = (new VBox());
        dims[1].getChildren().addAll(label_min_x, min_x);

        dims[2] = (new VBox());
        dims[2].getChildren().addAll(label_max_y, max_y);

        dims[3] = (new VBox());
        dims[3].getChildren().addAll(label_min_y, min_y);

        dims[4] = (new VBox());
        dims[4].getChildren().addAll(label_x_intvl, x_intvl);

        dims[5] = (new VBox());
        dims[5].getChildren().addAll(label_y_intvl, y_intvl);

        hbox.setSpacing(20);
        hbox.setAlignment(Pos.CENTER);
        hbox.getChildren().addAll(dims);

        vbox.setAlignment(Pos.CENTER);
        vbox.setSpacing(20);
        vbox.getChildren().addAll(chart_title,hbox);

        }

        public void resetFields(ScatterChart<Number,Number> chart){
        NumberAxis xaxis = (NumberAxis)chart.getXAxis();
        NumberAxis yaxis = (NumberAxis)chart.getYAxis();

        max_x.setText(String.valueOf(xaxis.upperBoundProperty().doubleValue()));
        min_x.setText(String.valueOf(xaxis.lowerBoundProperty().doubleValue()));
        max_y.setText(String.valueOf(yaxis.upperBoundProperty().doubleValue()));
        min_y.setText(String.valueOf(yaxis.lowerBoundProperty().doubleValue()));
        x_intvl.setText(String.valueOf(xaxis.getTickUnit()));
        y_intvl.setText(String.valueOf(yaxis.getTickUnit()));
        }
       }
\end{lstlisting}
```

```
\subsubsection{Data Set Class}
\begin{lstlisting}
public static class DataSet {

        private final SimpleDoubleProperty time;
        private final SimpleDoubleProperty steps;
        private final SimpleDoubleProperty heartRate;
        private final SimpleDoubleProperty elevation;
        private final SimpleDoubleProperty elevationVar;
        private final SimpleDoubleProperty stepsVar;
        private final SimpleDoubleProperty heartVar;
        private final SimpleStringProperty activity;
        public boolean changed;

        private DataSet(Double t, Double s,Double sv, Double e, Double ev,
         Double hr, Double hrv, String act) {

         this.time = new SimpleDoubleProperty(t);

         this.steps = new SimpleDoubleProperty(s);
         this.stepsVar = new SimpleDoubleProperty(sv);

         this.elevation = new SimpleDoubleProperty(e);
         this.elevationVar = new SimpleDoubleProperty(ev);

            this.heartRate = new SimpleDoubleProperty(hr);
            this.heartVar = new SimpleDoubleProperty(hrv);

            this.activity = new SimpleStringProperty(act);
            this.changed = false;
        }


        public Double getSteps() {
            return steps.get();
        }

        public void setSteps(Double val) {
            steps.set(val);
        }

        public Double getStepsVar() {
            return stepsVar.get();
        }

        public void setStepsVar(Double val) {
            stepsVar.set(val);
        }


        public Double getElevation() {
            return elevation.get();
        }

        public void setElevation(Double val) {
            elevation.set(val);
        }

        public Double getElevationVar() {
            return elevationVar.get();
```

```
        }

        public void setElevationVar(Double val) {
            elevationVar.set(val);
        }



        public Double getHeartRate() {
            return heartRate.get();
        }

        public void setHeartRate(Double val) {
            heartRate.set(val);
        }

        public Double getHeartVar() {
            return heartVar.get();
        }

        public void setHeartVar(Double val) {
            heartVar.set(val);
        }


        public Double getTime() {
            return time.get();
        }

        public void setTime(Double val) {
            time.set(val);
        }


        public String getActivity() {
            return activity.get();
        }

        public void setActivity(String val) {
         if(!val.equals(activity.getValue())){
         this.changed = true;
         System.out.println(this.changed);
         }
            activity.set(val);
        }
    }
\end{lstlisting}


\subsection{Utilities Library}

\subsubsection{Google Email Class}
\begin{lstlisting}

    public static MimeMessage createEmail(String[] to,
     String from,
     String subject,
     String bodyText)
     throws MessagingException {
```

```
     Properties props = new Properties();
Session session = Session.getDefaultInstance(props, null);

MimeMessage email = new MimeMessage(session);

email.setFrom(new InternetAddress(from));

for(int i=0; i<to.length ; i++)
email.addRecipient(javax.mail.Message.RecipientType.TO,
new InternetAddress(to[i]));

// email.addRecipient(javax.mail.Message.RecipientType.TO,
// new InternetAddress(to));
email.setSubject(subject);
email.setText(bodyText);
return email;
    }
    public static Message sendMessage(Gmail service,
            String userId,
            MimeMessage emailContent)
            throws MessagingException, IOException {
Message message = createMessageWithEmail(emailContent);
message = service.users().messages().send(userId, message).execute();

System.out.println("Message id: " + message.getId());
System.out.println(message.toPrettyString());
return message;
}

    public static Message createMessageWithEmail(MimeMessage emailContent)
            throws MessagingException, IOException {
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        emailContent.writeTo(buffer);
        byte[] bytes = buffer.toByteArray();
        String encodedEmail = Base64.encodeBase64URLSafeString(bytes);
        Message message = new Message();
        message.setRaw(encodedEmail);
        return message;
    }

    public void sendMsg(String[] to, String from, String subject, String bodyText){
        // Build a new authorized API client service.
     try{
        Gmail service = getGmailService();

        // Print the labels in the user's account.
        try{
        sendMessage(service,"me",createEmail(to,from,subject,bodyText));
        }catch(MessagingException e){
         e.printStackTrace();
        }

//         String user = "me";
//         ListLabelsResponse listResponse =
//             service.users().labels().list(user).execute();
//         List<Label> labels = listResponse.getLabels();
//         if (labels.size() == 0) {
//             System.out.println("No labels found.");
//         } else {
//             System.out.println("Labels:");
```

```
//             for (Label label : labels) {
//                 System.out.printf("- %s\n", label.getName());
//             }
//         }
     }catch(IOException e){
     e.printStackTrace();
     }
    }

\end{lstlisting}

\subsubsection{Google Sheet Class}
\begin{lstlisting}
    public int findLastRow(Sheet sheet){
     return sheet.getProperties().getGridProperties().getRowCount();
    }

    public int findAppendPoint(Instance inst) throws IOException{
        Spreadsheet spreadsheet = service.spreadsheets().get(spreadsheetId).execute();
        List<Sheet> sheets = spreadsheet.getSheets();

     String range = "data!A1:E";
        ValueRange response = service.spreadsheets().values()
            .get(spreadsheetId, range)
            .execute();

        List<List<Object>> values = response.getValues();

        ValueRange new_vals;

        int index=1;
        String updateRange="";
        if (values == null || values.size() == 0) {
            System.out.println("No data found.");
        } else {
          for (List row : values) {
            // Print columns A and E, which correspond to indices 0 and 4.
         if (row!=null){
         if(String.valueOf(row.get(0)).equals(inst.stringValue(0)) && (Double.parseDouble(String.valueOf(row.get(1)))==inst
         return index;
         }
         }
         index+=1;
          }//end of for
        }//end else

     return index;
    }

    public void appendData(String fileName) throws IOException{
        // Build a new authorized API client service.
        try{
        // Prints the names and majors of students in a sample spreadsheet:
        // https://docs.google.com/spreadsheets/d/1BxiMVs0XRA5nFMdKvBdBZjgmUUqptlbs74OgvE2upms/edit
        Spreadsheet spreadsheet = service.spreadsheets().get(spreadsheetId).execute();

        List<Sheet> sheets = spreadsheet.getSheets();

        DataSource source = new DataSource("../../Data/"+fileName);
Instances data = source.getDataSet();
```

```
List<List<Object>> vals = new ArrayList<List<Object>>();
for (int i=0; i<data.numInstances();i++){
List<Object> array = new ArrayList<Object>();
for (int j=0; j<data.numAttributes(); j++){
if(data.instance(i).attribute(j).isNominal())
array.add(data.instance(i).stringValue(j));
else
array.add(data.instance(i).value(j));
}
vals.add(array);
}

int start_cols = findAppendPoint(data.firstInstance());
int last_row= findLastRow(sheets.get(0));
String range = "data!A"+start_cols+":I"+start_cols+data.numInstances();
System.out.println("RANGE: "+range);

ValueRange val_range = new ValueRange();
val_range.setRange(range);
val_range.setValues(vals);
List<ValueRange> list = new ArrayList<ValueRange>();
list.add(val_range);
Request req;
if(start_cols+data.numInstances()-last_row>0){
req = new Request().setAppendDimension(new AppendDimensionRequest());
req.getAppendDimension().setLength(start_cols+data.numInstances()-last_row);
req.getAppendDimension().setDimension("Rows");
}else{
req = new Request().setUpdateCells(new UpdateCellsRequest());
req.getUpdateCells().setRange(new GridRange());
req.getUpdateCells().setFields("*");
req.getUpdateCells().getRange().setStartRowIndex(start_cols);
req.getUpdateCells().getRange().setEndRowIndex(start_cols+data.numInstances());
req.getUpdateCells().getRange().setStartColumnIndex(0);
req.getUpdateCells().getRange().setEndColumnIndex(data.numAttributes()-1);

}

List<Request> requests = new ArrayList<Request>();
requests.add(req);
BatchUpdateSpreadsheetRequest body = new BatchUpdateSpreadsheetRequest().setRequests(requests);
service.spreadsheets().batchUpdate(spreadsheetId, body).execute();

BatchUpdateValuesRequest request = new BatchUpdateValuesRequest();
request.setValueInputOption("USER_ENTERED");
request.setData(list);

BatchUpdateValuesResponse resp = service.spreadsheets().values().batchUpdate(spreadsheetId, request).execute();
        }catch(Exception e){
         e.printStackTrace();
        }
    }//end writeData

    public void writeClassifiedData(Data_Set data) throws IOException{
        Spreadsheet spreadsheet = service.spreadsheets().get(spreadsheetId).execute();
        List<Sheet> sheets = spreadsheet.getSheets();


     String range = "data!A1:I";
        ValueRange response = service.spreadsheets().values()
```

```
        .get(spreadsheetId, range)
        .execute();

    List<List<Object>> values = response.getValues();

    ValueRange new_vals;

    int index=1;
    String updateRange="";
    if (values == null || values.size() == 0) {
        System.out.println("No data found.");
    } else {
      for (List row : values) {
        // Print columns A and E, which correspond to indices 0 and 4.
     if (row!=null){

     if(String.valueOf(row.get(0)).equals(data.date) && (Double.parseDouble(String.valueOf(row.get(1)))==data.time)){
     updateRange = "data!I"+index;
     List<List<Object>> sheet_data = new ArrayList<List<Object>>();
     ArrayList<Object> row_data = new ArrayList<Object>();
     sheet_data.add(row_data);
     row_data.add(data.activity);

     new_vals = new ValueRange();
             new_vals.setRange(updateRange);
             new_vals.setValues(sheet_data);

     List<ValueRange> list = new ArrayList<>();
     list.add(new_vals);


             BatchUpdateValuesRequest oRequest = new BatchUpdateValuesRequest();
             oRequest.setValueInputOption("RAW");
             oRequest.setData(list);

             BatchUpdateValuesResponse oResp1 = service.spreadsheets().values().batchUpdate(spreadsheetId, oRequest).ex
     }
     }else{
     break;
     }
     index+=1;
      }//end of for
    }
}

public boolean getRawData(boolean opts) throws IOException{
    // Build a new authorized API client service.


    Excel xls = new Excel();
    xls.createFile("../../Data/temp.xlsx");
    xls.openFile("../../Data/temp.xlsx");
    // Prints the names and majors of students in a sample spreadsheet:
    // https://docs.google.com/spreadsheets/d/1BxiMVs0XRA5nFMdKvBdBZjgmUUqptlbs74OgvE2upms/edit
    String spreadsheetId = "1vynf_DD3VkDXIdzptz4CMNhxGYYjpihWseF8SAxG4Bg";
    String [] cats = {"steps","heart", "elevation"};
    Spreadsheet spreadsheet = service.spreadsheets().get(spreadsheetId).execute();
    List<Sheet> sheets = spreadsheet.getSheets();
    boolean new_readings=true;
    //List<GridData> data = spreadsheet.getSheets().get(0).getData();
```

```
    for(int i=0; i<sheets.size(); i++){
     System.out.println(sheets.get(i).getProperties().getTitle());
     SheetProperties props = sheets.get(i).getProperties();
    xls.createSheet(props.getTitle());
    xls.setSheet(props.getTitle());
 String range = props.getTitle()+"!A2:C";
    ValueRange response = service.spreadsheets().values()
        .get(spreadsheetId, range)
        .execute();

    List<List<Object>> values = response.getValues();
    int lastRow = 0;
    ArrayList<String> date = new ArrayList<String>();
    ArrayList<String> time = new ArrayList<String>();


    if (values == null || values.size() == 0) {
        System.out.println("No data found.");
    } else {
      for (List row : values) {
        // Print columns A and E, which correspond to indices 0 and 4.
     if (row!=null){
     System.out.printf("%s, %s, %s\n", row.get(0), row.get(1), row.get(2));
     date.add(String.valueOf(row.get(0)));
     time.add(String.valueOf(row.get(1)));
     }else{
     if (date.isEmpty()){
     new_readings = false;
     }
     break;
     }
      }//end of for
      xls.write(date.stream().toArray(String[]::new), xls.getLastRow(0)+1, 0);
      xls.write(time.stream().toArray(String[]::new), xls.getLastRow(1)+1, 1);
      double[] vals = new double[date.size()];
      for (int j=0; j<date.size();j++){
       vals[j] = Double.parseDouble(String.valueOf(values.get(j).get(2)));
      }
      xls.write(vals, xls.getLastRow(2)+1, 2);
    }//end of else
    }//end of category for
xls.close();


if (opts){
 for (int i=0;i<sheets.size();i++){
     SheetProperties props = sheets.get(i).getProperties();
 String range = props.getTitle()+"!A2:C";
 service.spreadsheets().values().clear(spreadsheetId, range, new ClearValuesRequest()).execute();
 }
}
return new_readings;
}//end getrawData

public Data_Set[] getClassifiedData(String date) throws IOException{
    // Build a new authorized API client service.
```

```
        // Prints the names and majors of students in a sample spreadsheet:
        // https://docs.google.com/spreadsheets/d/1BxiMVs0XRA5nFMdKvBdBZjgmUUqptlbs74OgvE2upms/edit
        Spreadsheet spreadsheet = service.spreadsheets().get(spreadsheetId).execute();
        List<Sheet> sheets = spreadsheet.getSheets();

        //List<GridData> data = spreadsheet.getSheets().get(0).getData();
        System.out.println("Getting data for date " + date + "...");

     String range = "data!A1:I";
        ValueRange response = service.spreadsheets().values()
            .get(spreadsheetId, range)
            .execute();

        List<List<Object>> values = response.getValues();
        ArrayList<Data_Set> data_set = new ArrayList<Data_Set>();

        if (values == null || values.size() == 0) {
            System.out.println("No data found.");
        } else {
          for (List row : values) {
            // Print columns A and E, which correspond to indices 0 and 4.
         if (row!=null){
         if(row.get(0).equals(date)){
         data_set.add(new Data_Set(String.valueOf(row.get(0)), Double.parseDouble(String.valueOf(row.get(1))),
         Double.parseDouble(String.valueOf(row.get(2))),
         Double.parseDouble(String.valueOf(row.get(3))),
         Double.parseDouble(String.valueOf(row.get(4))),
         Double.parseDouble(String.valueOf(row.get(5))),
         Double.parseDouble(String.valueOf(row.get(6))),
         Double.parseDouble(String.valueOf(row.get(7))),
         String.valueOf(String.valueOf(row.get(8)))));
         }
         }else{
         break;
         }
          }//end of for
        }

    return data_set.stream().toArray(Data_Set[] :: new);
    }//end getData

\end{lstlisting}

\subsubsection{Data Set Class}
\begin{lstlisting}

public class Data_Set{
String date;
double time;
double heartrate;
double heartratev;
double steps;
double stepsv;
double elevation;
double elevationv;
String activity;


public Data_Set(String d, double t, double s, double sv, double e, double ev, double h, double hv, String a){
this.date = d;
```

```
this.time = t;
this.steps = s;
this.stepsv = sv;
this.elevation = e;
this.elevationv = ev;
this.heartrate = h;
this.heartratev = hv;
this.activity = a;
}
}
\end{lstlisting}

\subsubsection{Excel Class}
\begin{lstlisting}

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.Array;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;

import org.apache.poi.hssf.usermodel.HSSFDateUtil;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.DateUtil;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import sun.util.calendar.BaseCalendar.Date;

public class Excel{
public String fileName;
FileInputStream inputStream;
Workbook workbook;
Sheet currentSheet;
public Excel(){

}
public Excel(String fn){
try{
fileName = fn;
inputStream = new FileInputStream(new File(fileName));
workbook = new XSSFWorkbook(inputStream);
}catch(IOException e){
e.printStackTrace();
}
}
public void createFile(String fn){
if(!Files.exists(Paths.get(fn))){
try{
XSSFWorkbook wb = new XSSFWorkbook();
FileOutputStream fos = new FileOutputStream(fn);
wb.write(fos);
```

```
fos.close();
wb.close();
}catch(IOException e){
e.printStackTrace();
}
}
}//end createFile

public void openFile(String fn){
try{
fileName = fn;
inputStream = new FileInputStream(new File(fileName));
workbook = new XSSFWorkbook(inputStream);
}catch(IOException e){
e.printStackTrace();
}
}//end openFile

public void createSheet(String name){
if (workbook.getSheet(name)!=null){
return;
}else{
workbook.createSheet(name);
}
}

public void setSheet(String sheetName){
currentSheet = workbook.getSheet(sheetName);
}//end setSheet

public void close(){
try{
workbook.close();
inputStream.close();
}catch(IOException e){
e.printStackTrace();
}
}//end close

public double[][] readDouble(int startRow, int startCol, int numCol, String timeFormat){
        ArrayList<double[]> result = new ArrayList<double[]>();

        Iterator<Row> iterator = currentSheet.iterator();
        int rowLength=currentSheet.getLastRowNum();

        for (int i=startRow; i<rowLength+1; i++){
         Row row = currentSheet.getRow(i);
         double[] array = new double[numCol];
         for (int j=0; j<numCol; j++){
         Cell cell = row.getCell(j+startCol);
         System.out.println(cell);
     if (DateUtil.isCellDateFormatted(cell)){
     switch(timeFormat){
     case "H+M=H":
     array[j] = (cell.getDateCellValue().getMinutes()/60.0)+cell.getDateCellValue().getHours();
     break;
     case "H+M=M":
     array[j] = cell.getDateCellValue().getMinutes()+(cell.getDateCellValue().getHours()*60.0);
     break;
     }
```

```
     }else{
             array[j] = cell.getNumericCellValue();
     }
         }
         result.add(array);
        }

return result.stream().toArray(double[][]::new);
}

public double[][] readDouble(int startRow, int startCol, int numCol){
        ArrayList<double[]> result = new ArrayList<double[]>();

        Iterator<Row> iterator = currentSheet.iterator();
        int rowLength=currentSheet.getLastRowNum();

        for (int i=startRow; i<rowLength+1; i++){
         Row row = currentSheet.getRow(i);
         double[] array = new double[numCol];
         for (int j=0; j<numCol; j++){
         Cell cell = row.getCell(j+startCol);
             array[j] = cell.getNumericCellValue();
         }
         result.add(array);
        }

return result.stream().toArray(double[][]::new);
}

public String[][] readString(int startRow, int startCol, int numCol){
        ArrayList<String[]> result = new ArrayList<String[]>();

        int rowLength=currentSheet.getLastRowNum();

        for (int i=startRow; i<rowLength+1; i++){
         Row row = currentSheet.getRow(i);
         String[] array = new String[numCol];
         for (int j=0; j<numCol; j++){
         Cell cell = row.getCell(j+startCol);
         switch (cell.getCellType()){
         case Cell.CELL_TYPE_STRING:
             array[j] = cell.getStringCellValue();
             break;
         case Cell.CELL_TYPE_NUMERIC:
         if (HSSFDateUtil.isCellDateFormatted(cell))
         array[j] = cell.getDateCellValue().toString();
         else
         array[j] = String.valueOf(cell.getNumericCellValue());
         break;
         }
         }
         result.add(array);
        }

return result.stream().toArray(String[][]::new);
}

public String[] readString(int startRow, int col){
        ArrayList<String> result = new ArrayList<String>();
```

```
        int rowLength=currentSheet.getLastRowNum();

        for (int i=startRow; i<rowLength+1; i++){
         Row row = currentSheet.getRow(i);
     Cell cell = row.getCell(col);
     String val="";

     switch (cell.getCellType()){
     case Cell.CELL_TYPE_STRING:
         val = cell.getStringCellValue();
         break;
     case Cell.CELL_TYPE_NUMERIC:
     if (HSSFDateUtil.isCellDateFormatted(cell))
     val = cell.getDateCellValue().toString();
     else
     val = String.valueOf(cell.getNumericCellValue());
     break;
     }
         result.add(val);
        }

return result.stream().toArray(String[]::new);
}


public double[] readDouble(int startRow, int col){
        ArrayList<Double> result = new ArrayList<Double>();

        int rowLength=currentSheet.getLastRowNum();

        for (int i=startRow; i<rowLength+1; i++){
         Row row = currentSheet.getRow(i);
     result.add(row.getCell(col).getNumericCellValue());
        }
double[] array = new double[result.size()];
for (int i=0; i<result.size();i++)
array[i] = result.get(i);

return array;
}


public int getLastRow(int col){
int index=-1;
Iterator<Row> rowIterator;
try{
rowIterator = currentSheet.iterator();
}catch(NullPointerException e){
return index-1;
}
while(rowIterator.hasNext()){
Row row = rowIterator.next();
index+=1;
if(row.getCell(col)==null){
return index-1;
}
}
return index;
}
```

```
public void write(double[][] text, int startRow, int startCol){
int lastRow= currentSheet.getLastRowNum();

        for (int i=startRow; i<text.length+startRow; i++){
           Row row = currentSheet.getRow(i);
         if (row==null){
         row = currentSheet.createRow(i);
         }
         for (int j=0; j<text[0].length; j++){
         Cell cell = row.createCell(j+startCol);
         cell.setCellValue(text[i-startRow][j]);
         }
        }
        try{
         FileOutputStream outputStream = new FileOutputStream(new File(fileName));
workbook.write(outputStream);
        }catch(IOException e){
         e.printStackTrace();
        }
}//end write

public void write(String[][] text, int startRow, int startCol){
int lastRow = currentSheet.getLastRowNum();

for (int i=startRow; i<text.length+startRow; i++){
           Row row = currentSheet.getRow(i);
         if (row==null){
         row = currentSheet.createRow(i);
         }
         for (int j=0; j<text[0].length; j++){
         Cell cell = row.createCell(j+startCol);
         cell.setCellValue(text[i-startRow][j]);
         }
        }
        try{
         FileOutputStream outputStream = new FileOutputStream(new File(fileName));
workbook.write(outputStream);
        }catch(IOException e){
         e.printStackTrace();
        }
}//end write

public void write(String[] text, int startRow, int startCol){
int lastRow = currentSheet.getLastRowNum();

        for (int i=startRow; i<text.length+startRow; i++){
           Row row = currentSheet.getRow(i);
         if (row==null){
         row = currentSheet.createRow(i);
         }
         Cell cell = row.createCell(startCol);
         cell.setCellValue(text[i-startRow]);
        }
        try{
         FileOutputStream outputStream = new FileOutputStream(new File(fileName));
workbook.write(outputStream);
        }catch(IOException e){
         e.printStackTrace();
        }
}//end write
```

```
public void write(double[] text, int startRow, int startCol){
int lastRow = currentSheet.getLastRowNum();

for (int i=startRow; i<text.length+startRow; i++){
           Row row = currentSheet.getRow(i);
         if (row==null){
         row = currentSheet.createRow(i);
         }
         Cell cell = row.createCell(startCol);

         cell.setCellValue(text[i-startRow]);

        }
        try{
         FileOutputStream outputStream = new FileOutputStream(new File(fileName));
workbook.write(outputStream);
        }catch(IOException e){
         e.printStackTrace();
        }
}//end write


public void build(){
try{
        String message="";

        Iterator<Row> iterator = currentSheet.iterator();
        while (iterator.hasNext()) {
            Row nextRow = iterator.next();
            Iterator<Cell> cellIterator = nextRow.cellIterator();
            while (cellIterator.hasNext()) {
                Cell cell = cellIterator.next();
                switch(cell.getCellType()){
                case Cell.CELL_TYPE_STRING:
                message += cell.getStringCellValue()+"-";
                 break;
                case Cell.CELL_TYPE_NUMERIC:
                message += String.valueOf(cell.getNumericCellValue())+"-";
                break;
                }
            }
                message = message.substring(0,message.length()-1)+":";
        }

//  Process program = Runtime.getRuntime().exec("java LineChartBuilder "+message);
//  InputStream is = program.getInputStream();
//  byte[] b = new byte[128];
//  do{
//  is.read(b);
//  System.out.println(String.valueOf(Arrays.toString(b)));
//  }while(program.isAlive());

workbook.close();
        inputStream.close();
}catch(IOException e){
         e.printStackTrace();
        }
}//end build
}
```

```
\end{lstlisting}
```