

---

# MARS File Conversion

BIOT 670  
Dr. Rumpf  
Spring 2021

— Jeffrey Day, Maura Franz, Alex Mancera,  
Hilary Mehler, Stephen Panossian, Analia  
Treviño-Flitton —

---

# Background

# Project Overview

- The hospital's GE MARS software requires proprietary format .nat file as input
- The bedside ECG monitors output data in .csv and .hdr formats
- Currently, the hospital is required to run the ECG output files through several conversion steps before loading the data into the MARS software
- Our goal is to create a single program to convert the ECG files into a file format compatible with the MARS proprietary software using free Python interpreter

# Original Workflow

The bedside monitors output .hdr and .csv files.

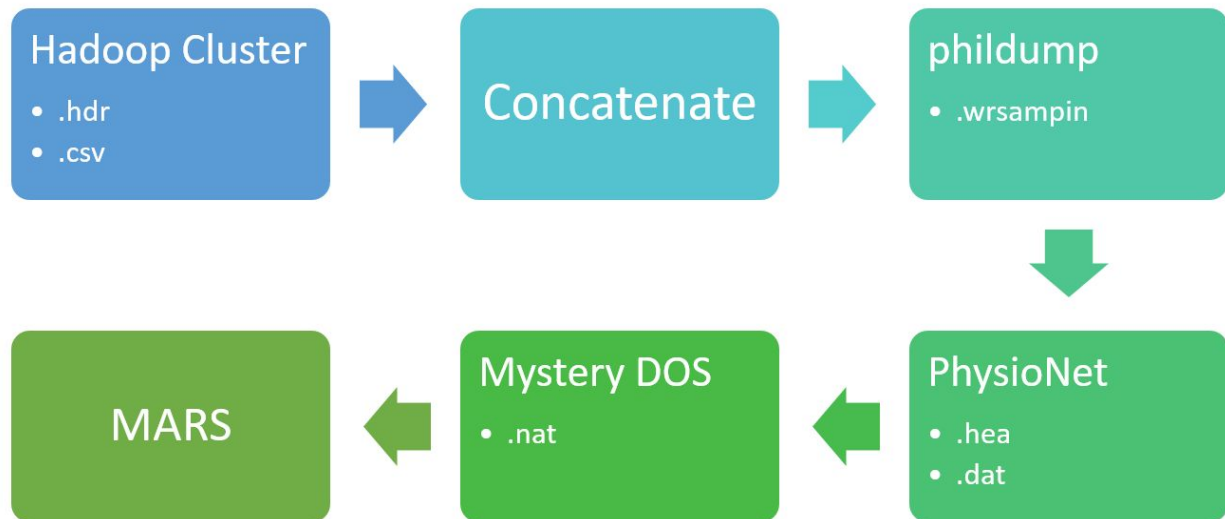
These are then concatenated using cmd and the resulting file is ran through a Python program

The Python program outputs a few bars of information and produces a .wrsampin file.

PhysioNet is used to produce .hea and .dat files

These files are then run through a mystery DOS converter to produce a .nat file to feed to the MARS software

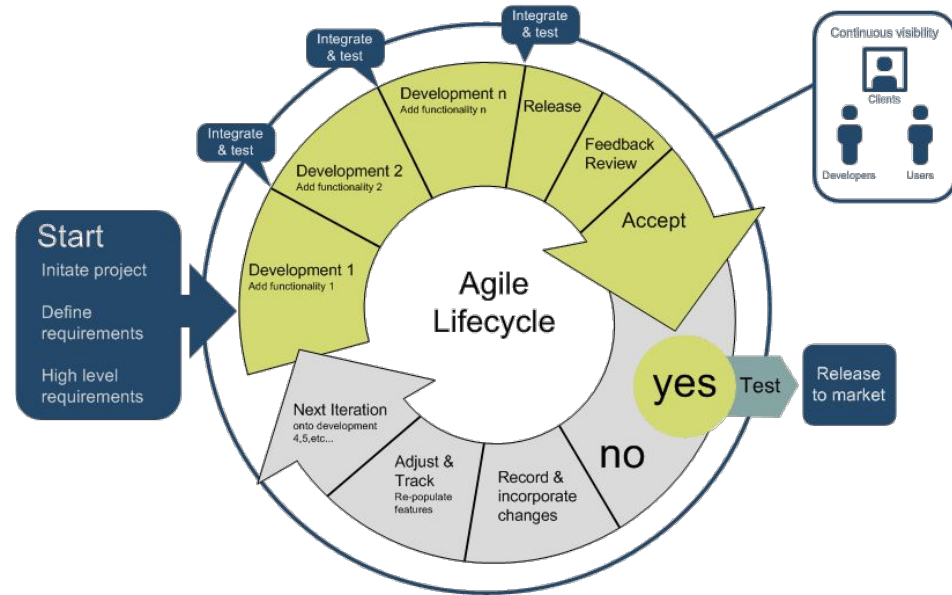
Figure 1. Original hospital ECG signal processing workflow.



# Agile Software Development

- Influenced by the project's research topic
- Prioritizes dynamic collaboration from developers instead of rigid structure
  - ◆ Accomplished through active Discord channel & weekly meetings
- Focuses on developing functional products first with documentation after
  - ◆ Consistent testing throughout

Figure 2. Visual depiction of the Agile Lifecycle.  
*SaigonTechnology.com*

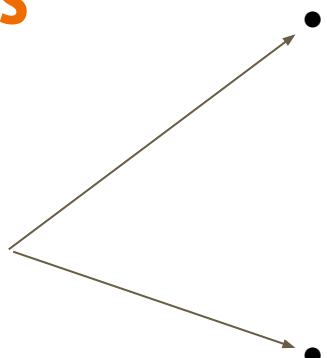


# **.nat File Research**

# .nat GE Proprietary Format

- Essentially no explanatory material
- 3 hex view/edit apps + 3 Linux dump commands + 1 online hex editor
- Binary more coherent viewed as hex
  - byte-offset
  - column titles
  - no numerical values observed
  - converter apparently inserted extra data
  - differencing .dat and .nat files unproductive
- Unable to determine transformation algorithm
- Programming bottleneck led to redefinition of project goals

# Refocused Project Goals

- Scrap DOS converter format reverse-engineering
  - Deliver 2 Python maintainable and expandable applications
  - Developed & tested using versions 3.9.1, 3.9.2, and 3.9.3
  - Environments:
    - Windows 10 Home 10.0.18363 Build 18363
    - macOS v11.2.3 Big Sur
- 
- HRPY: heart rate data processor
    - Accepts .csv and .hdr files
    - Produces .hea and .dat files
    - Option to save modified data
    - DOS converter still needed
  - HRVY: heart rate data viewer
    - 1 channel / output plot
    - Option to preprocess data
      - Baseline correction
      - Noise reduction
      - Flat/steep peak reduction
    - Option to save uncorrected/corrected data





# HRPY

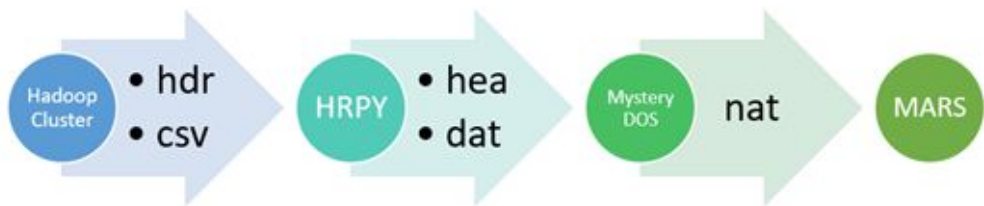
## Heart Rate Processor in Python

### The Preprocessing Program

---

# New Processing Workflow

Figure 3. MARS Group New ECG Processing Workflow.



- HRPY accepts .hdr and .csv files from the Hadoop Cluster and processes them into .hea and .dat files
- The .hea and .dat files are then input into the Mystery DOS Software

# How HRPY Works

.csv file is read and stored in a dataframe

- These are processed to make them readable by the final function:
  - `wfdb.io.csv2mit()`

.hdr file is read and parsed out to find:

- IDs -- used as column names
- Units
- Sample frequency
- This information is used in the final function

## How HRPY Works, cont.

Each column from the dataframe is cycled through functions that:

- Fill in any empty spaces:
  - No spaces are removed as it would not accurately correspond to the appropriate timestamp.
  - Average of last valid value and next value is computed and used.
  - Essentially creating a straight line between the two values
- Ensure the sample frequency is set to 8 ms
  - In order to maintain 8 ms time between samples, data rows are either removed or added via extrapolation (similar to above process)
  - Sample frequency pulled from .hdr file to calculate number of data rows to add/remove.

# Running HRPY

From HRPY directory

1. Ensure:
  - a. WFDB library is installed
    - i. Otherwise `>pip install wfdb`
  - b. `.hdr` and `.csv` files are in the same directory as HRPY
    - i. Otherwise modify PATH
  - c. *filename.hdr* and *filename.csv* have same *filename*
2. Enter: `> python hrpy_v1.py <hdr_filename.hdr> <csv_filename.csv>`
3. Output: `<hdr_filename_new.he> <hdr_filename_new.dat>`
  - a. Optional `<csv_filename_new.csv>`

# Sample HRPY Output

- The first line is the record line.
- Others are signal specification lines.
  - Two signal specification lines in this case

Data file (.dat) output:

- Binary file

New .csv file:

- This is the file that is read by the wfdb.io.csv2mit() function

Figure 4. Header file (.hea) output from file a.csv/a.hdr

```
a_new 2 500 8061440 19:31:14.582 11/09/2017
a_new.dat 16 200(0)/mV 16 0 -8092 53207 0 ID15
a_new.dat 16 200(0)/mV 16 0 -8092 2467 0 ID43
```



# HRVY

## Heart Rate Viewer in Python

### The Viewing Program

---

# HRVY Back-End: Development Considerations

- Memory accommodations for file size
  - ◆ Package Import
    - Import only functions called
  - ◆ Program Structure
    - One-time read of large input data file
  - ◆ Clearing Variables
    - Objects no longer needed are cleared
  - ◆ Variable Recycling
    - Variables commonly used between functions reused as needed



# Free, Open-Source Python Packages & Modules

## → HeartPy

- ◆ Heart rate analysis toolkit
- ◆ Processes both electrocardiogram (ECG) and photoplethysmogram (PPG) data
- ◆ Used in HRVY for raw input ECG signal data preprocessing

## → Plotly

- ◆ Graphing functions library

## → NumPy

- ◆ Module for scientific computing via multidimensional array object

## → SciPy

- ◆ Builds on NumPy with additional math functions

# HeartPy Functions

- Remove\_Baseline\_Wander
  - ◆ Corrects baseline wander

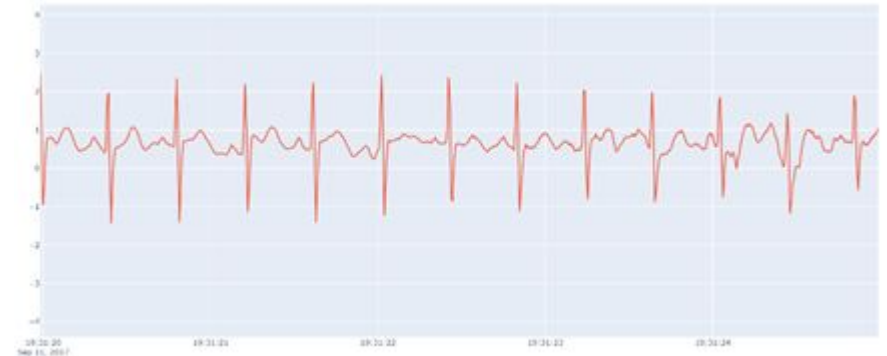
\*Corresponding *images zoomed in*

- Get\_SampleRate\_DateTime
  - ◆ Calculates the sample\_rate

Figure 5. Raw Data Before Remove\_Baseline\_Wander.



Figure 6. Raw Data After Remove\_Baseline\_Wander.



# HeartPy Functions

## → Smooth\_Signal

- ◆ Low-level background noise correction

\*Corresponding *images zoomed in*

## → Flip\_Signal

- ◆ Inverts the signal values

Figure 7. Raw Data Before Smooth\_Signal.

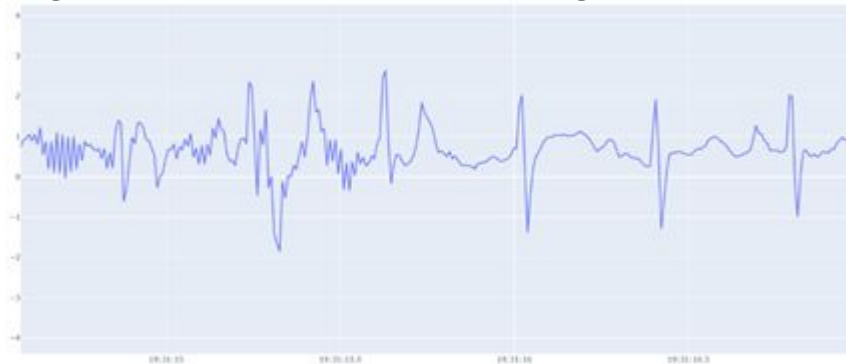
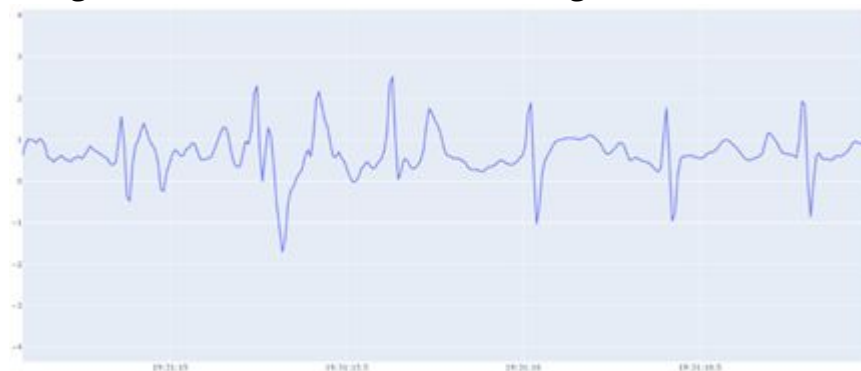


Figure 8. Raw Data After Smooth\_Signal.



# Pulling Data & Filling Gaps

Data from the .hdr file was pulled and saved into the `hdr_data` dictionary

- Additional data for nodes that had an interval  $> 2$  ms was recorded in the `nodes_not_2ms` dictionary

Data from the .csv file was pulled and saved into the `file_dat` dictionary

- Columns with blank values were filled with the baseline value of 0.5 mV
- Gaps of null values were also filled with the baseline value of 0.5 mV
- Missing datetime values were filled by adding 2 ms to the time from the previous line.
- Time gaps (inconsistencies) were filled by adding missing datetime values and a baseline value of 0.5 for each node.
- Time interval gaps (for nodes with an interval  $> 2$  ms) were filled by taking an average of the values before and after the gap

# Custom HRVY Filtering Functions

## → Flat\_Peak\_Reduct:

- ◆ Originally only designed to reduce long, flat “peaks” of identical values found at the beginning of datasets to avoid getting curved graphs after baseline correction. Also used to reduce tall peaks that had been cut off at the range maximum (~40 mVs) and left as tall, flat peaks.
- ◆ More drastic correction came from reducing excessively tall/steep peaks before baseline correction.

## → Tall\_Peak\_Reduct:

- ◆ Reduces excessively tall/steep peaks after baseline correction.
- ◆ Detects inverted signals and calls `negdata_flip()` if necessary.

Figure 9. Data After Flat Peak Reduction.

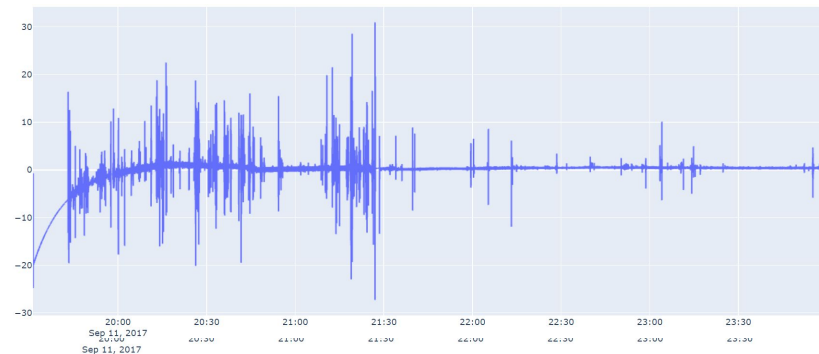
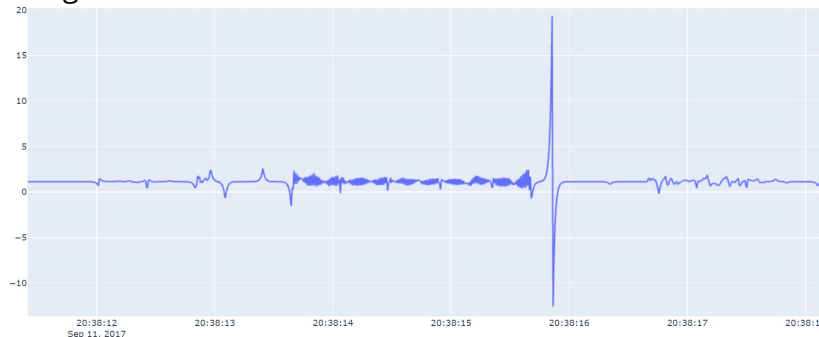


Figure 10. Data After Tall Peak Reduction.



# HRVY Complete Filtering Algorithm

- Flat\_Peak\_Reduct
  - ◆ Reduces flat peaks  $> 20$  ms and reduces steep peaks (MARS Group)
- Remove\_Baseline\_Wander
  - ◆ Accepts sample rate to yield flattened baseline (HeartPy)
- Tall\_Peak\_Reduct
  - ◆ Reduces (less) steep peaks and detects inverted peaks (MARS Group)
- Flip\_Signal
  - ◆ Inverts signal, with optional peak enhancement and/or keeping original range (HeartPy)
- Smooth\_Signal
  - ◆ Accepts sample rate, window length, and polynomial order to remove noise (HeartPy)

# HRVY Front-End: Running the HRVY Program

From HRVY directory

1. Ensure:
  - a. HeartPy, Plotly, Numpy, and SciPy installed and current
    - i. `>pip install module_name.py`
  - b. Header and data files in same directory as HRVY
    - i. modify PATH variable
  - c. Header and data files have same filename
2. Enter: `>python hrvy_v1.py`

# Running the HRVY Program, cont.

## 3. Output:

- New directory *filename*
  - *filename\_channel\_name.html* or
  - *filename\_channel\_name\_***corrected***.html*
- Optional dataset:
  - *filename\_concat.csv* or
  - *filename\_concat\_***corrected***.csv*

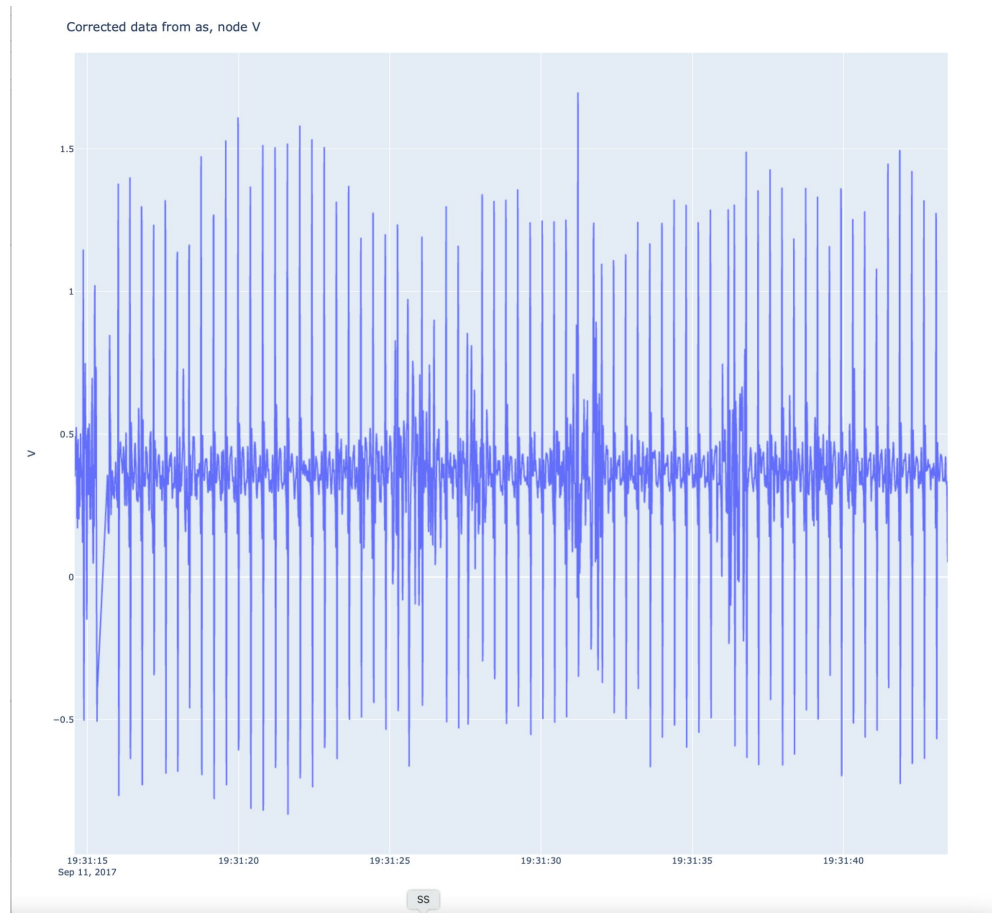


# Sample HRVY Plot

User can:

- Hover
  - Show closest data point/value
- Zoom
  - In/out via double click

Figure 11. Sample Preprocessed Data Plot for a given Channel.



# HRVY

Dash Application

---

# HRVY Dash App

## Overview:

- Same data processing functions
- User Interface (UI) built for web using Dash framework

## Some key differences:

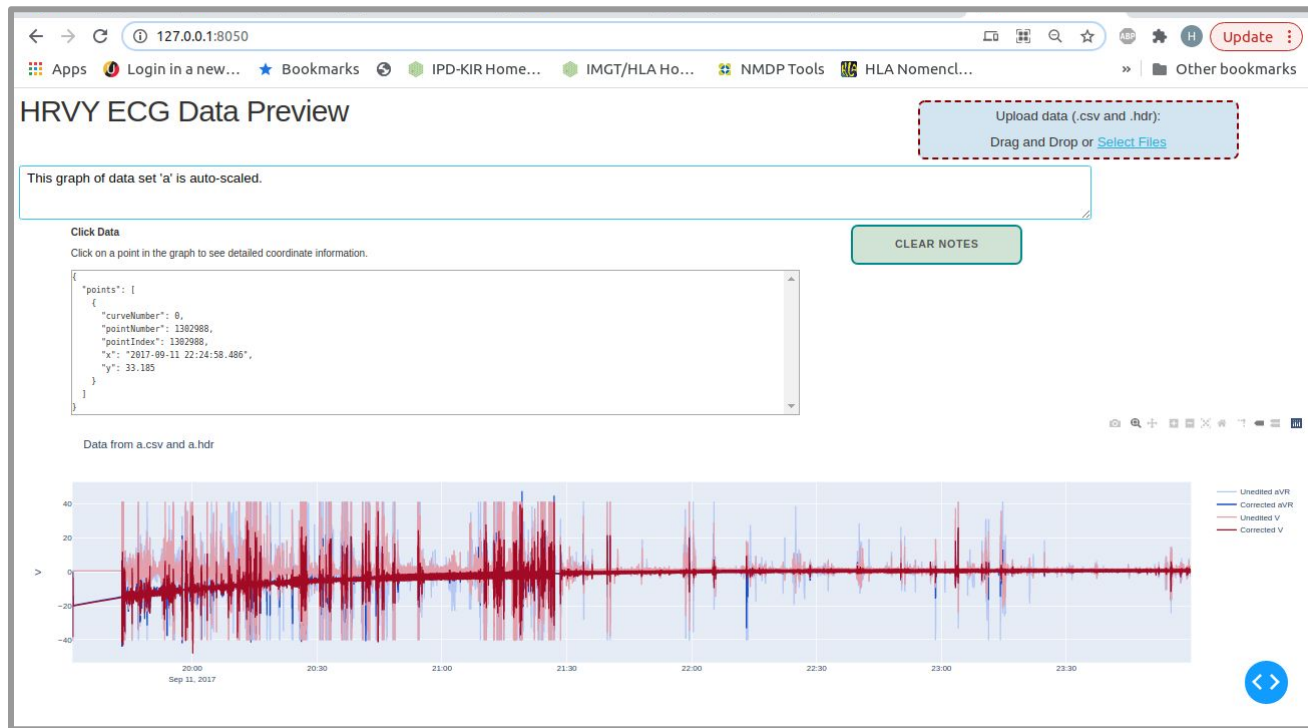
- Data files may be located in any directory
  - Dash temporarily stores uploaded files as base64 encoded strings
- Graphing function adjusted to plot all data, at the cost of memory
  - Users may hide or view traces using Plotly's interactive features

# Dash Overview

A Python interface to create interactive web components that normally require Javascript, CSS, and HTML.

Figure 12. Sample Dash Development.

- Open-Source & MIT licensed
- Written on top of Flask, React, and **Plotly**
  - Ideal for building web app for data visualization using Python



# Dash Code

Two main sections:

- 1) Layout - create web components & customize visual design (HTML), Figure 13.
- 2) Callbacks - provide interactivity by tying functions to the web components, Figure 14.

```
html.Div([
    html.Button('Clear Notes', id='clear-notes-button',
        style={
            'width': '100%',
            'height': '60px',
            'lineHeight': '20px',
            'borderWidth': '4px',
            'borderRadius': '10px',
            'borderColor': 'darkcyan',
            'backgroundColor': '#d1e3d5',
            'font-size': '16px'
        })
], className='two columns'),
```

```
@app.callback(Output('notes', 'value'),
               Input('clear-notes-button', 'n_clicks'),
               Input('upload-data', 'filename'))
def update_notes(n_clicks, list_of_names):
    count = 0
    value = 'Upload the .hdr and .csv files for a single data'
    ctx = dash.callback_context
    trigger = ctx.triggered[0]['prop_id'].split('.')[0]

    if trigger == 'clear-notes-button': #if clear button was clicked
        value = ''

    else: #read file names from upload component to provide up
        if list_of_names is not None:
            for filename in list_of_names:
```


Example shows creation of a 'Clear Notes' button (left) and the callback that provides function to the button

# Running the Dash App

Figure 15. Sample Dash Interaction.

```
In [1]: runfile('/home/hilary/PythonScripts/MARS/PythonScripts/MARS')
Dash is running on http://127.0.0.1:8050/

* Serving Flask app "dash_test" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not
  Use a production WSGI server instead.
* Debug mode: on
```



- Locally
  - Uses local machine's memory
  - User must install all required packages
    - Run Python script from terminal or IDE console
    - In browser, navigate to IP address & port shown
      - In Linux, may Ctrl+click on IP to open Dash browser session
    - In terminal or console, press Ctrl+C to quit
- On server: <http://hrvy.herokuapp.com/>
  - Virtual environment with all required packages installed
    - Package & Python versions have been tested & proven to run without error
  - Server caps memory usage at 512 MB - limited to small data files

# Dash App Features

- Upload .csv & .hdr to view data
- If the wrong number or type of files are uploaded, updates and instruction are provided to the user
- Large text area in which user may type notes
- User may clear the text area with button click
- 'Click Data' shows details when user clicks a point on the graph
  - Utility: User may copy/paste X coordinate (timestamp) into notes, in order to keep track of points to view in MARS software
- Plotly interactivity: zoom, hover data, hide/view data traces
- User may save .png of graph

# Future Enhancements

- HRPY
  - Cache data into smaller chunks to avoid memory limitations
- HRVY
  - Add machine learning to fill in data gaps
  - Develop unsupervised algorithm to detect anomalies
  - Incorporate HRPY
- Dash App
  - Button to download notes as .txt
  - Adjust to accept & process .cat files
  - Incorporate HRPY functions upon button click
    - requires caching .csv & .hdr to server, may be limited by file size



## Future Enhancements cont.

- Overcome memory limitations
  - Reduce data set size
  - Limit time range or bootstrap
  - Eliminate data points to plot using % difference
  - Use client side callbacks' to run functions with large overhead
    - Runs directly on server instead of making request to Dash
  - WSGI with higher [free] memory cap
- Integrate HRPY & HRVY into Jupyter notebook
- Further development in Anaconda Navigator
- Replace DOS Converter with GE assistance

# MARS Group GitHub

HRPY

<https://github.com/mars-group-2021/mars-hrpy>

HRVY

<https://github.com/mars-group-2021/mars-hrvy>

Dash

<https://github.com/mars-group-2021/mars-dash-app>

# MARS File Conversion

Thank You  
for your Time and Attention

Questions?