4-2017

Saving Babies Using Big Data

Evan Dienstman
College of William and Mary

Saving Babies Using Big Data

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science in Mathematics from
The College of William and Mary

by

Evan Dienstman

Accepted for _____
(Honors, High Honors, Highest Honors)

_____
Prof. John Delos, Director

_____
Prof. Leah Shaw

_____
Prof. Junping Shi

_____
Prof. Daniel Vasiliu

Williamsburg, VA
April 21, 2017

William & Mary Mathematics Honors Project

Saving Babies Using Big Data

Student

Evan Dienstman

eddienstman@email.wm.edu

Academic Advisers

John Delos, jbdelo@wm.edu
Daniel Vasiliu, dvasiliu@wm.edu

May 10, 2017

This project was jointly supported by NSF Grant 1331021.

Contents

List of Figures

Abstract

Because of their underdeveloped immune systems, premature babies are at an increased risk to contract many illnesses. Thus, early detection of a disease is vital to saving a premature baby's life. Current methods of detecting illnesses, however, have been inadequate, providing many false positives and insufficient amount of warning time. However, patterns in the heart rate of babies have shown signs of predicting the onset of sepsis in premature infants. Research conducted by Prof. John Delos and others suggest that low variability and clusters of decelerations in an infant's heart rate may indicate an impending septic event. Additionally, there is weak evidence that low variability may be linked to gram-positive bacteria and clusters of decelerations may be linked to gram-negative bacteria. If this statement is true, then not only will the heart rate of an infant predict the onset of sepsis, but also provide a partial diagnosis and thereby indicate the preferred treatment for the baby. However, much more work needs to be done to prove this hypothesis. Over twelve terabytes of data has been collected on premature babies' heart rate and breathing. To search through this data, one first needs to know what to look for. Unfortunately, only looking for low variability and clusters of decelerations would be inadequate since most babies experience some low variability and decelerations in their heart rate at some point. Therefore, sophisticated statistical analysis is necessary to quantify this data. The general idea of this analysis includes creating many different heart rate characteristics (HRCs) and measuring their predictive power through multiple methods. The results of our research indicate that the HRCs of variance, sample entropy, and asymmetry are strong predictors of illness. However, no HRC shows strong signs of indicating the type of invading organism that caused the illness.

Acknowledgments

I would like to thank my academic advisors at William & Mary, Prof. John Delos and Prof. Daniel Vasiliu, for their instruction and support during this entire process. Various Matlab programs used in the research are original/modified programs by Prof. Douglas Lake and Abigail Flower. I would also like to thank our UVA friends, Prof. Douglas Lake and Dr. Randall Moorman, for their guidance on the medical side of this project. The data calculations in this report were performed on computational resources supported by William & Mary's high-performance computers (SciClone). This project is an extension of work started as part of my participation in EXTREEM-QED program at William & Mary. The EXTREEM-QED program is supported by NSF Grant 1331021.

Chapter 1

Introduction

Premature babies are at an increased risk of having a septic event due to their immature immune systems. Therefore, nurses must frequently take blood samples of the infants in order to determine a patient's health status. Blood work, however, is a slow process, so doctors might not be performing the best treatment for the infant while waiting for the results of a test. Furthermore, drawing blood, as with any invasive measure, could cause complications. Our goal is to try to use the baby's heart rate as a quick, noninvasive way to decide if an infant is unhealthy. Research has already shown some signs that reduced variability and transient decelerations in the heart rate could predict sepsis [4]. We also hypothesize that low variability may be linked to gram-positive bacteria and clusters of decelerations may be linked to gram-negative bacteria, giving insight into the preferred treatment for the baby. In order to test our two hypotheses, we will develop heart rate characteristics (HRCs) to better understand the data. We will then test the usefulness of the HRCs at predicting sepsis. In the end, the ultimate goal will be to test if our HRCs are predictive of sepsis and the hypothesis that low variability is associated with gram-positive diseases and clusters of decelerations are associated with gram-negative diseases.

1.1 Medical Background

1.1.1 Sepsis

Sepsis refers to an inflammatory response caused by the body fighting off an infection [8]. In order to detect sepsis, a nurse will look for changes in a baby's body temperature, digestion, and other vital signs. Unfortunately, these symptoms may not appear until long after the baby has become infected. Another challenge is that even though early treatment with antibiotics is very effective in treating sepsis, without the proper diagnosis of the invading organism, a doctor will not be able to administer the best drug to the patient. A nurse, consequently, must take a blood sample in order to properly identify the pathogen. However, this procedure is time consuming and not ideal for very small infants. In our report, we will focus on five classifications of invading organisms: coagulase-negative staphylococci (CONS), gram-positive bacteria, gram-negative bacteria, fungus, and other. The hope from the heart rate analysis is two-fold: 1) give earlier warning than current methods about a septic event and 2) provide information about the type of organism that caused the septic event. If we accomplish both our goals, then not only will we be able to provide doctors with an inexpensive, noninvasive way of detecting sepsis in its early stages, but also provide a partial diagnosis, and consequently a treatment, for the pathogen.

1.1.2 RR Intervals

An electrocardiogram (EKG) displays a time series of electric impulses created by the heart. In this time series, the largest peaks occur at the beginning ventricular contraction. This peak is defined as the R-peak. Physicians use the R-peak to represent the time a heart beat occurs. Therefore, the RR interval is the time between one R-peak to the next, or rather, the time between heart beats. When analyzing heart rate throughout this report, we will use the RR intervals. Note that even though the the RR intervals are not technically a rate, one can easily obtain a heart rate by finding the number of intervals in a certain unit of time. Another way to relate the RR intervals to a heart rate is that large RR intervals imply a slower heart rate and small RR intervals imply a faster heart rate. [1] Figure 1.1 shows an example time-series of an EKG with one RR interval.

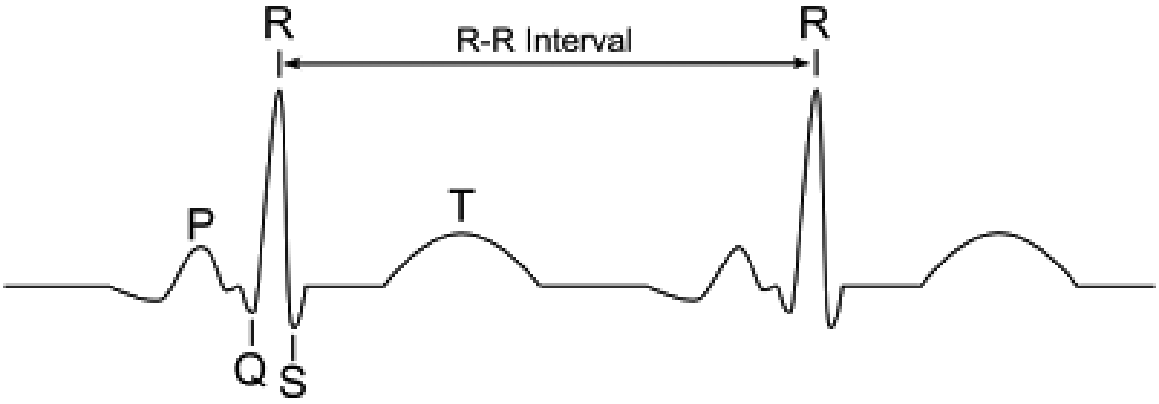


Figure 1.1: `RR Interval`[9]

1.2 Prior Work

As reported in *Mortality Reduction by Heart Rate Characteristic Monitoring in Very Low Birth Weight Neonates: A Randomized Trial* by Dr. J. Randall Moorman and Prof. Douglas E. Lake, et al., four heart rate characteristics were found to provide early warnings of illness: variance, asymmetry 1, asymmetry 2, and sample entropy. We will explain the definitions of these measures in Chapter 2. A Heart Rate Observation score, or HeRO score, which was previously derived using a sample of a few hundred sepsis events, was also used in the randomized clinical trial to provide warning of illness. The HeRO score is a measure that combines the four heart rate characteristics. We will explain more about the HeRO score in Chapter 10. Specifically, the trial showed that as a consequence of these early warnings, mortality in neonatal intensive care units was reduced by between $10-40\%$ amongst various cohorts.

In this paper, we reanalyze the data from the clinical trial and develop a new HeRO score based off this data. We also incorporate additional measures of heart rate variability into the score. We hope that these two changes will improve the statistical predications of the HeRO score.

1.3 Data Collection

All data was collected from electrocardiograms at eight neonatal intensive care units

(NICUs) during the randomized trial led by Dr. J. Randall Moorman [6]. The locations of the NICUs have been withheld to protect the identity of the patients. In total, the trial included around 3,000 babies with each baby having roughly one month's worth of data. From the 3,000 babies, about 1,000 septic events were recorded. Note that is is possible for one baby to experience multiple septic events.

Each infant has a file containing a vector of RR intervals as well as the time each RR interval ends. Times are calculated from time 0 corresponding to when the patient was first connected to the EKG. Because the RR intervals are times themselves, the time vector is simply a cumulative sum of the RR interval vector assuming the monitor is never disconnected. However, the time vector is necessary since babies may be disconnected from the monitor at times. We, therefore, need the time vector to indicate at what time each RR interval occurred in the patient's history. An example of these two vector are shown below:

$$\text{rr\_intervals} = [420.25,\ 421.75,\ 420.25,\ 418.25,\ 422.00,\ 423.75\ ...]$$
$$\text{tt\_intervals} = [420.25,\ 842.00,\ 1262.3,\ 370470,\ 370880,\ ...]$$

In the example above, `rr_intervals` refers to the RR intervals and `tt_intervals` refers to the times of the RR peaks. Note that `tt_intervals` is a cumulative sum of `rr_intervals` until the large jump from 1,262.3 to 370,470. A large jump like this corresponds to a time when a baby was disconnected from the monitor. Smaller jumps may also occur when the analyzing software fails to find a beat. In this example, both `rr_intervals` and `tt_intervals` are measured in milliseconds. Throughout the rest of the report, we measure the RR intervals in milliseconds and the times of the RR intervals in days.

The RR intervals and the times of the RR intervals constitute the bulk of the data for a patient. Other data for a patient includes a site number corresponding to the patient's NICU, an ID number, demographic information, and a file indicating the times when the patient was ventilated. If a patient is having difficulty breathing, the nurses will ventilate the infant. We keep a record of this information because we believe ventilation reduces heart rate variability. Lastly, we have another file containing all the patients that had a septic event, the time of the event, and the invading organism that caused the event. Note that a single patient can experience multiple septic events while at a NICU.

In order to better understand the data, we break up all the `rr_interval` vectors into half hour pieces. Later, when we calculate statistical measures on the RR intervals, we calculate them for each half hour vector instead of over the entire vector. Doing our calculations this way helps us determine change in the various measurements over time. The choice to divide our vectors into half hours is an empirical decision aided by opinions of Prof. Lake and Dr. Moormon at UVA. Note that changing the length of the vector pieces will have an effect on the statistical measurements, and similar work in the field uses 5 or 10 minute intervals. However, for the purpose of this report, we use 30 minute intervals.

1.4 Notes on the Matlab Code

Throughout this thesis, we will reference Matlab code used to create data files and figures. To help relate the code to the report, each section using code will contain the names of the files involved at the beginning. Consider the following example below:

Bivariate Risk Figures: `multiple_bivariate_risk_figures` → `one_bivariate_risk_figure` → `one_bivariate_risk_plot` → `one_risk_matrix` → `one_prob_matrix`

The text above shows the code needed to create the bivariate risk figures. Note that

`multiple_bivariate_risk_figures` → `one_bivariate_risk_figure`

means that `multiple_bivariate_risk_figures` calls `one_bivariate_risk_figure`. However, sometimes an arrow does not mean one function calls another, but rather the second function is the next function in the procedural order. While the report reviews important aspects of the code, all code will not be discussed in detail. However, Appendix B contains all the Matlab code for reference written by Evan Dienstman. Code written by other people has been omitted from this report.

Chapter 2

Heart Rate Characteristics

Now that we have RR intervals broken up into half hour sections, we need to develop statistical measures for each half hour that may be predictive of illness. We will focus on seven statistical measures, or heart rate characteristics (HRCs), throughout this report: mean, variance, sample entropy, three measures of asymmetry, and decelerations. Additionally, we will calculate five subcategories for each HRC: the raw HRC value, the 10th percentile of the HRC over the past 12 hours, the 50th percentile of the HRC over the past 12 hours, the 90th percentile of the HRC over the past 12 hours, and the slope of the HRC over the past 2 days. Accordingly, with seven HRCs and five subcategories for each one, we get a total of 35 HRCs we will analyze. The remainder of the chapter gives the definitions for each of these 35 HRCs.

2.1 Mean

The first HRC is the mean. For each half hour, we simply calculate the average RR interval length. The mean for each half hour typically ranges from 420 ms - 440 ms.

2.2 Variance

Like the mean, we also calculate the variance for each half hour. However, when doing our calculation, we take the natural logarithm of the variance so the scale is easier to visualize. These log-variance values range from 0 - 5.

2.3 Asymmetry

Asymmetry is a measure of how skewed the data looks. To calculate asymmetry, consider a vector with N RR intervals with median m where B is the set of intervals below m and A is the set of intervals above m. We can then define the following quantities:

$$r_1 = \frac{1}{N}\sum_{i\in B}(RR_i - m)^2 \tag{2.1}$$

$$r_2 = \frac{1}{N}\sum_{i\in A}(RR_i - m)^2 \tag{2.2}$$

Notice that the calculations for r_1 and r_2 are similar to the calculation of variance except with the median instead of the mean. Finally, we define three measures of asymmetry:

$$asymmetry\_1 = \ln(r_1) \tag{2.3}$$

$$asymmetry\_2 = \ln(r_2) \tag{2.4}$$

$$asymmetry\_ratio = \frac{asymmetry\_2}{asymmetry\_1} \tag{2.5}$$

For each half hour, typical asymmetry_1 and asymmetry_2 values range from 0 - 10 and asymmetry_ratio values range from 0 - 3. Any asymmetry_ratio greater than 1 indicates that there are more intervals greater than the median than are less than the median, i.e., the data is skewed towards large intervals, indicating a slow heart rate [5].

2.4 Sample Entropy

The exact definition of sample entropy is beyond the scope of this report. However, the general idea is that sample entropy is a measurement of how random the numbers in the RR interval vector occur. A low sample entropy means the RR interval vector is fairly regular while a high sample entropy means the intervals appear to be random. Values of sample entropy range from 0 - 1 with 0 indicating the signal is completely periodic and 1 indicating the signal is completely random. Note that a horizontal line would be considered perfectly periodic and would have a sample entropy of 0. A sine or cosine function would also have a very low sample entropy. For a complete explanation of sample entropy, please refer to the paper *Physiological time-series analysis using approximate entropy and sample entropy* by J.S. Richman and J.R. Moorman [7].

2.5 Decelerations

The final raw HRC is number of decelerations. We define a deceleration as a sharp increase in the RR intervals followed by a sudden return to a baseline. Recall that an RR interval is the time between heart beats; therefore, an increase in the RR intervals represents a decrease in the heart rate. Figure 2.1 shows an example of a deceleration highlighted in red over time series of RR intervals.

Figure 2.1 contains one half hour worth of RR intervals. Within this half hour, there are many more decelerations than one highlighted in red. The computer algorithm for detecting decelerations was developed by Abigail Flower [2]. The algorithm uses a template deceleration, such as the one highlighted in red, and sweeps the template through the signal. The algorithm then makes a decision if the part of the signal in question is a significant deceleration based on the height of the peak and how well the signal matches the template. The algorithm then records the number of decelerations found. Typical values for the number of decelerations in a half hour range from 0 - 10 with rare cases going up to 100.

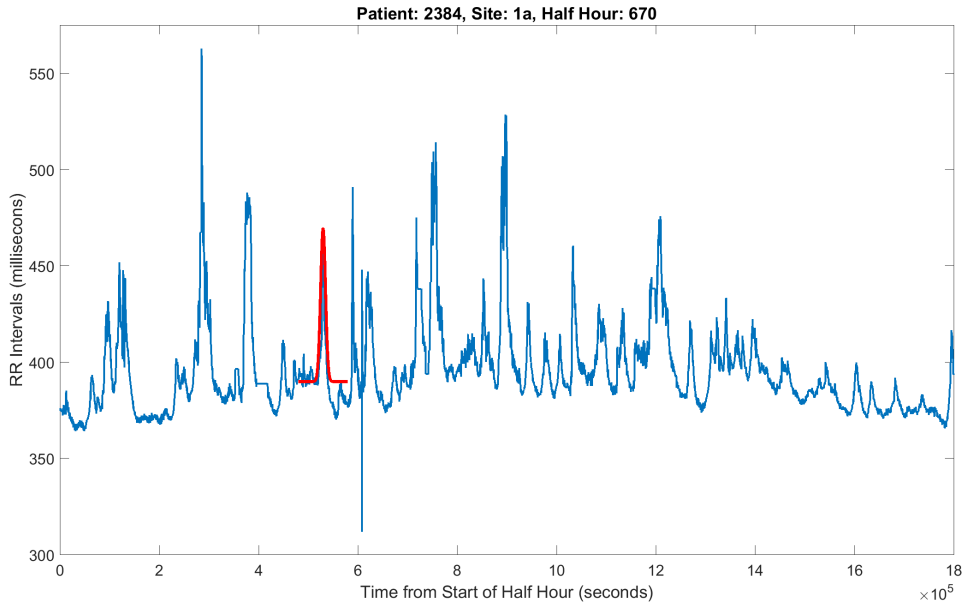


Figure 2.1: Deceleration

2.6 HRC Subcategories

In addition to all seven raw HRC measurements described above, we also calculate four more measurements for each HRC. The first three are the 10th, 50th, and 90th percentiles of the HRC values over the past 12 hours. To better understand the procedure for calculating these subcategories, consider the example of calculating the 10th percentile of variance. If all half hours are perfectly sequential in terms of their start times, *i.e* no missing data, a window of 12 hours in the past will correspond to a vector of 24 half hours. We then calculate the 10th percentile for these 24 variance values and record this value as the 10th percentile of variance for the current half hour. The logic behind taking the 10th percentile is that one very low variance over a 12 hour window might be a stronger predictor of sepsis than looking at all the values for each half hour. However, because babies are frequently disturbed by the nurses, we take the 10th percentile instead of the minimum to avoid taking outliers caused by outside influences. Note that for a vector of 24 values, the 10th percentile is about the second lowest value.

We then repeat this process for next half hour which will result in shifting our 12 hour window up one half hour. Consequently, this new 24 half hour vector will be identical to the previous one with the exception of the oldest half hour being removed from the end and the current half hour being added to the beginning. We, therefore, should expect to see similar 10th percentile values for consecutive half hours most of the time and sudden changes sometimes once a new very low variance is captured by the window. The 50th and 90th percentiles follow a similar procedure.

The last subcategory is slope. In order to calculate the slope, we first index the raw HRC values over the past 2 days. We then take a linear fit of the points in this vector and map the slope of this fit to the HRC slope for the current half hour. Our thought process behind recording the slope is that the actual value of the HRC might not be as strong of a predictor as the rate of change of the values.

For any HRC subcategory, if there is insufficient data as a result of a baby being newly connected to a monitor or from mechanical errors, we mark the HRC as "not a number" or NaN in Matlab. For exact thresholds of how much data we need, please see the documentation of the relevant code.

Chapter 3

Data Organization and Preprocessing

In this section, we will focus on where we get the data for our HRC calculations and how we store the results of those calculations.

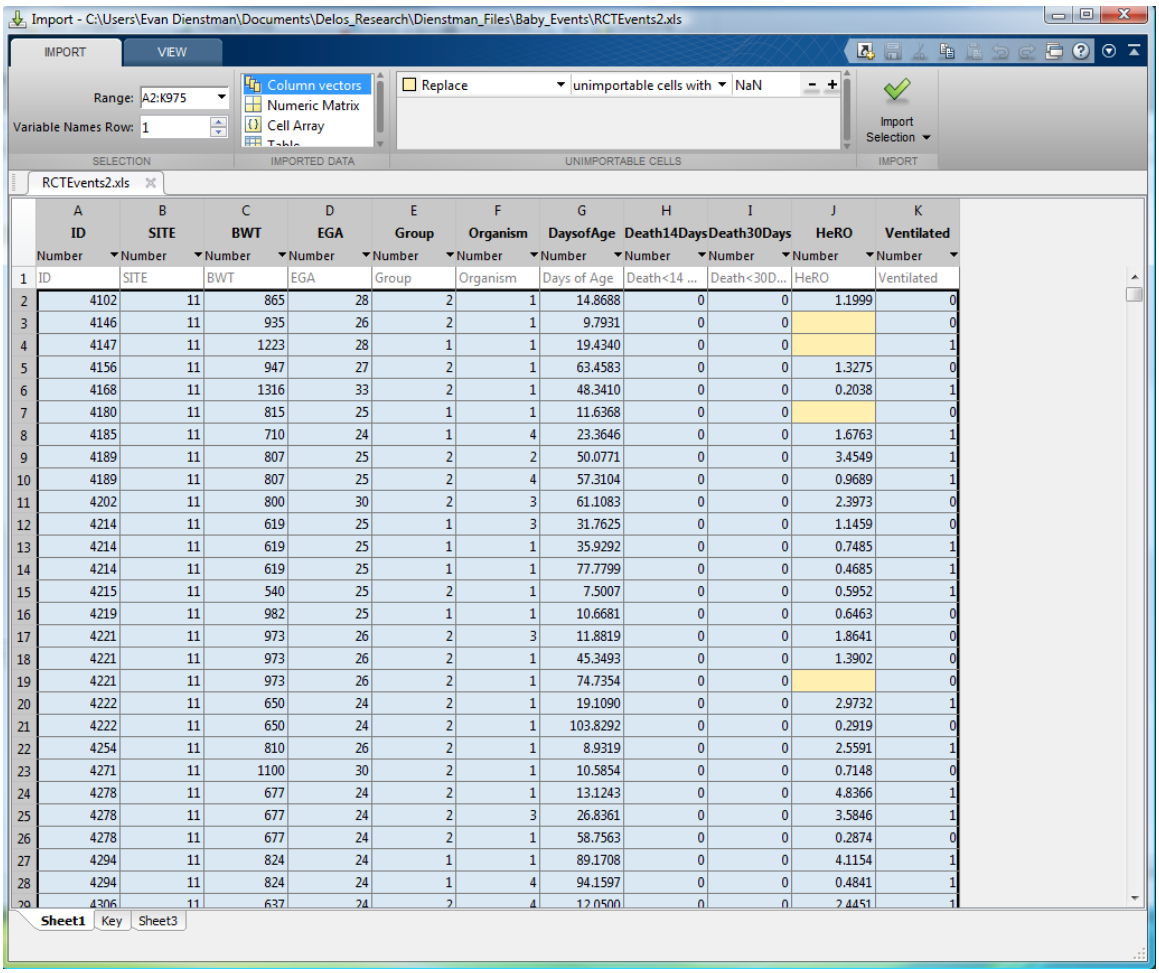
3.1 HRB Files

All the information from the EKG monitors comes from `.hrb` files. Each baby has one HRB file for his/her entire stay in the NICU. The HRB files contain the RR intervals as well as the time of each RR interval. Additionally, the monitor also reports a 0 or 1 for each interval to indicate whether the interval was good (0) or bad (1). Bad intervals may be the result of the monitor missing a heartbeat or a mechanical error. Lastly, each file is labeled with the four digit ID of the baby and contained in a folder with a site number of his/her NICU. The eight site numbers are 11, 13, 15, 23, 24, 26, 27, and 30. We will discuss what we do with this information in Section 3.3.

3.2 Excel Files

Apart from the HRB files, all the other information we have is contained in Excel files. An example of what one of these Excel files looks like is shown in Figure 3.1. This file, called `RCTEvents2.xls` contains information for every baby who had a septic event. Moving from left to right, the columns in the file are ID, site, birth weight, gestational age, group, type of organism that caused the septic event, days of age at the time of the event, death within 14 days after the event, death within 30 days after the event, HeRO score at the time of the event, and ventilation status at the time of the event. For the purpose of this report, we will ignore the group column. The organism column can contain five different classifications of organisms: 1 - coagulase-negative staphylococci (CONS), 2 - gram-positive bacteria, 3 - gram-negative bacteria, 4 - fungus, and 5 - other. The HeRO score is a type of logistic regression that we will discuss in further detail in Chapter 10. Concerning ventilation status, we have another Excel file containing specific times when each patient was ventilated. Later, when we need to determine the ventilation status of each individual half hour, we will use this file for the information. Again, we keep a record of the ventilation status because we hypothesize that babies have lower heart variability

while ventilated. Finally, one other Excel file contains demographic information about all babies that we will use later for separating babies into categories.



	A	B	C	D	E	F	G	H	I	J	K
	ID	SITE	BWT	EGA	Group	Organism	DaysofAge	Death14Days	Death30Days	HeRO	Ventilated
	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number
1	ID	SITE	BWT	EGA	Group	Organism	Days of Age	Death<14 ...	Death<30D...	HeRO	Ventilated
2	4102	11	865	28	2	1	14.8688	0	0	1.1999	0
3	4146	11	935	26	2	1	9.7931	0	0		0
4	4147	11	1223	28	1	1	19.4340	0	0		1
5	4156	11	947	27	2	1	63.4583	0	0	1.3275	0
6	4168	11	1316	33	2	1	48.3410	0	0	0.2038	1
7	4180	11	815	25	1	1	11.6368	0	0		0
8	4185	11	710	24	1	4	23.3646	0	0	1.6763	1
9	4189	11	807	25	2	2	50.0771	0	0	3.4549	1
10	4189	11	807	25	2	4	57.3104	0	0	0.9689	1
11	4202	11	800	30	2	3	61.1083	0	0	2.3973	0
12	4214	11	619	25	1	3	31.7625	0	0	1.1459	0
13	4214	11	619	25	1	1	35.9292	0	0	0.7485	1
14	4214	11	619	25	1	1	77.7799	0	0	0.4685	1
15	4215	11	540	25	2	1	7.5007	0	0	0.5952	1
16	4219	11	982	25	1	1	10.6681	0	0	0.6463	0
17	4221	11	973	26	2	3	11.8819	0	0	1.8641	0
18	4221	11	973	26	2	1	45.3493	0	0	1.3902	0
19	4221	11	973	26	2	1	74.7354	0	0		0
20	4222	11	650	24	2	1	19.1090	0	0	2.9732	1
21	4222	11	650	24	2	1	103.8292	0	0	0.2919	0
22	4254	11	810	26	2	1	8.9319	0	0	2.5591	1
23	4271	11	1100	30	2	1	10.5854	0	0	0.7148	0
24	4278	11	677	24	2	1	13.1243	0	0	4.8366	1
25	4278	11	677	24	2	3	26.8361	0	0	3.5846	1
26	4278	11	677	24	2	1	58.7563	0	0	0.2874	0
27	4294	11	824	24	1	1	89.1708	0	0	4.1154	1
28	4294	11	824	24	1	4	94.1597	0	0	0.4841	1
29	4306	11	637	24	2	4	12.0500	0	0	2.4451	1

Figure 3.1: `RCTEvents2.xls`

3.3 Dienstman Result Files

Dienstman Result Files: `Dienstman_submit_parallel` → `multiple_result_files` → `one_result_file`

Now that we have discussed all the raw data files, we can discuss how we process and organize the data. For each HRB file, we need to break up the information into half hours and calculate the HRCs for each half hour. The result of this procedure will be files similar to the one in Figure 3.2. The file contains a structured array with many fields. The first field is the start time of the half hour. The next 35 fields are the HRCs discussed in Section 2. The `Good_Frac` field refers to the percent of intervals in the half hour the EKG monitor marked as "good". Because of mechanical errors or disturbances from the baby, the monitor might miss a heartbeat. Therefore, if an interval is recognized to be "bad", an algorithm in the machine will mark it. Ideally, we want the fraction of good beats to be as close to 1 as possible. However, we currently do not remove any half hours for being below a

minimum threshold. We do however, do some preprocessing with the data. The results of the preprocessing can be found in the final field `Extra_Info`. This field contains the raw RR interval vector as well as the processed RR interval vector. The purpose of the preprocessing is to remove the "bad" beats before calculating the HRCs. There are two methods we use for preprocessing the data. For details on these methods, please refer to `one_result_file.m`.

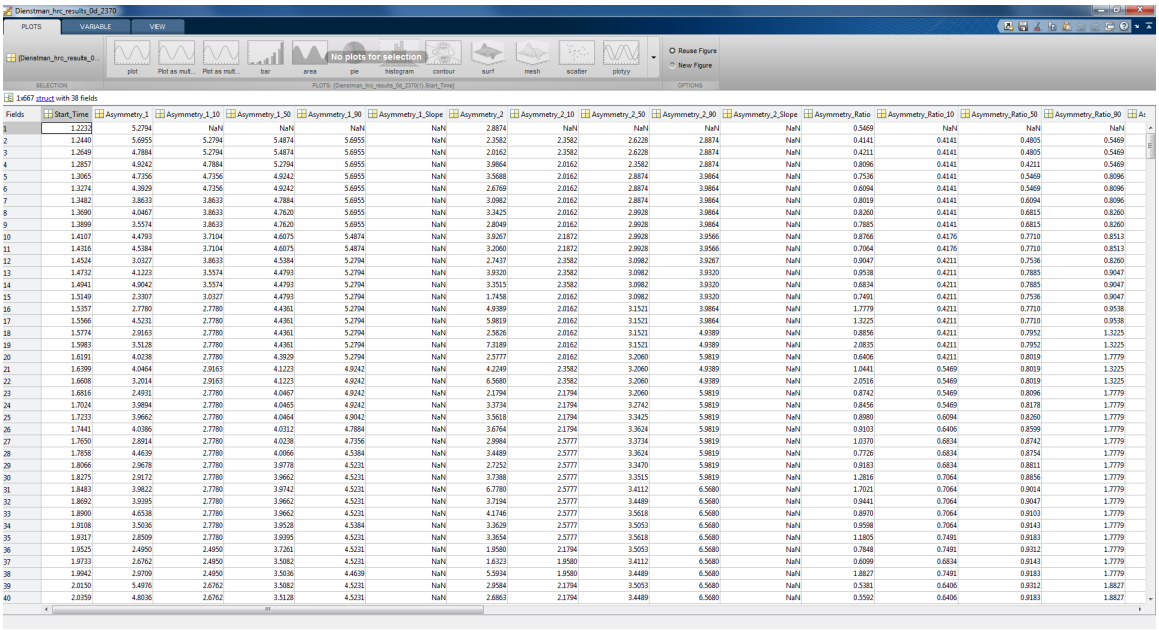


Figure 3.2: Dienstman Result File Example

We will refer to the files like the one in Figure 3.2 as Dienstman result files or just result files. Each baby has a result file containing its HRCs for each half hour. Using William & Mary's SciClone computers, one can create these files using the following command:

Listing 3.1: Usage for `Dienstman_submit_parallel`

```
1 >> chmod u+x Dienstman_submit_parallel.txt
2 >> dos2unix Dienstman_submit_parallel.txt
3 >> ./Dienstman_submit_parallel.txt 1 500
```

This example will create result files for all babies. Since we have about 3000 babies, we submit 500 batches of jobs indicated on the third line. Each batch will run at most 20 result files in serial. All the batches run in parallel on SciClone's computers so we can calculate all the result files as fast as possible. When executing the command above, make sure all the programs it calls are in the appropriate directories. For more details about how to create the result files, please see the appropriate code in Appendix B.

3.4 CSV Files

Dienstman CSV Files: `Dienstman_submit_batch` → `csv_files` → `csv_splitter` → `csv_avg_hrcs` → `csv_bin_widths` → `csv_indices` → `csv_logistic_coeffs`

After calculating the HRCs for each half hour, we want to group like half hours into CSV files for easier access. Such groups include sick vs. healthy, ventilated vs. not ventilated, and the type of organism that went on to cause the illness for babies who experienced a septic event. For this report, our definition of sick is any half hour that occurs 12 hours before a recognized septic event. The definition of healthy is any half hour that is 7 days or more before an event or 3 days or more after an event. Concerning ventilated vs. not ventilated, the ventilation status of each half hour can be found within the Excel file `RCT_Mechanical_Ventilation_Times.xlsx`. Additionally, the file `RCTEvents2.xls` contains the organism that caused the illness if a baby had an event, and the file `RCT_Demographics_ALL.xls` contains the birth weight, gestational age, and gender of each baby.

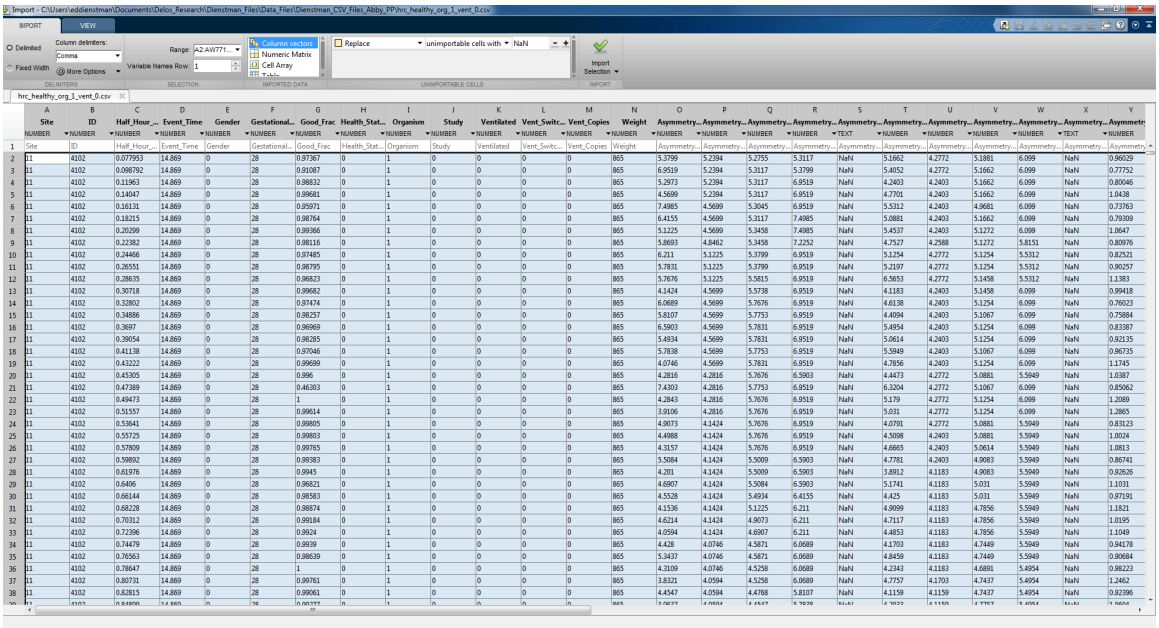


Figure 3.3: CSV File Example

Figure 3.3 depicts what one of these CSV files looks like. Every row in the CSV file corresponds to one half hour. Half hours are first selected from the Dienstman result files as either being sick or healthy using the file `RCTEvents2.xls` which contains the event times. It is important to note that the CSV files do not contain every half hour in the data set since some half hours might not be classified as either sick or healthy. The Dienstman result files contain a thorough record of this information organized by infant. Once we select a half hour, we record the site, ID, start time, and HRCs for that half hour. If the baby had a septic event, we also record the organism, event time, birth weight, and gestational age. Lastly, for both healthy and sick, we record the ventilation status for that half hour. (The last two columns in the CSV files are markers which can be ignored for the sake of this report.) In order to create the CSV files, a user calls the following commands on William & Mary's SciClone computers:

Listing 3.2: Usage for `Dienstman_submit_batch`

```
>> chmod u+x Dienstman_submit_batch.txt
>> dos2unix Dienstman_submit_batch.txt
>> qsub ./Dienstman_submit_batch
```

These commands call `Dienstman_submit_batch.txt` which in turn calls `csv_files.m`. The programs make many CSV files. The first CSV file is an extensive file containing all the sick and healthy half hours. The remainder of the files only contain specific types of half hours. For example, one CSV file contains ventilated sick half hours of babies who had a septic event caused by organism 1. Breaking the CSV files down into these smaller subcategories helps with organization and loading in data when we do our analysis later. In general, the CSV files help facilitate manipulation of large data sets that would be more difficult with the result files. Thus, a majority of the statistical techniques later in the report will use the CSV files. The remaining Matlab functions in the CSV series calculate meta information from the CSV files that we will use later. For more information about how to make the CSV files, please see Appendix B.

Chapter 4

Time Series

We are finally ready to analyze the predictive power of the HRCs. Before we use the CSV files to do a more sophisticated analysis, we will first use the result files to create a time series graph of the HRCs for each infant who had a septic event. In this manner, we can gain some intuition about how the HRCs behave, especially leading up to an event for the babies that experienced one. We will also create a time series of the average HRC values relative to the time of an event. The hope is that we will see significant changes in the HRC values before an event, predicting the onset of sepsis.

4.1 Time Series Figures

Time Series Figures: `multiple_event_figures_caller` → `multiple_event_figures` → `one_event_hrc` → `one_event_plot`

Figure 4.1 shows what a time series figure looks like for a single baby. The seven subplots represent the seven HRC. On each subplot, there are five time series for the five HRC subcategories. Note that the slope HRC uses the right y-axis. The black vertical line indicates the time of the event, and the x-axis shows the half hour index relative to the time of the event. The set of time series in a dark color depict times a baby was ventilated and light colors indicate when the baby was not ventilated. Lastly, the black horizontal line gives the average HRC value across all half hours of all babies for comparison purposes. For further examples of individual time series figures, see Appendix A where we have included examples where the baby was always unventilated, examples where the baby was always ventilated, and examples where the baby is missing data at certain times. The documentation of `multiple_event_figures_caller.m` provides more information about these figures.

After we create the time series graphs for each baby who had an event, we can then average these time series across all babies. Figure 4.2 shows the results of this averaging. To better understand the average time series, consider the data point right before the event on the variance subplot of Figure 4.2. This data point was the result of averaging all the variances at the half hour right before the event for each baby. However, since not all babies have data for this half hour, this data point was calculated using only the subset of the babies who had data at this half hour. We then repeat this process for every half hour relative to the event, noting that each average was calculated using a different amount of half hours depending on how much data we have. To get a sense about how many babies

were used for the averages, we average the number of babies used to calculate each half hour across all the average time series. This number is reported in the legend of Figure 4.2.

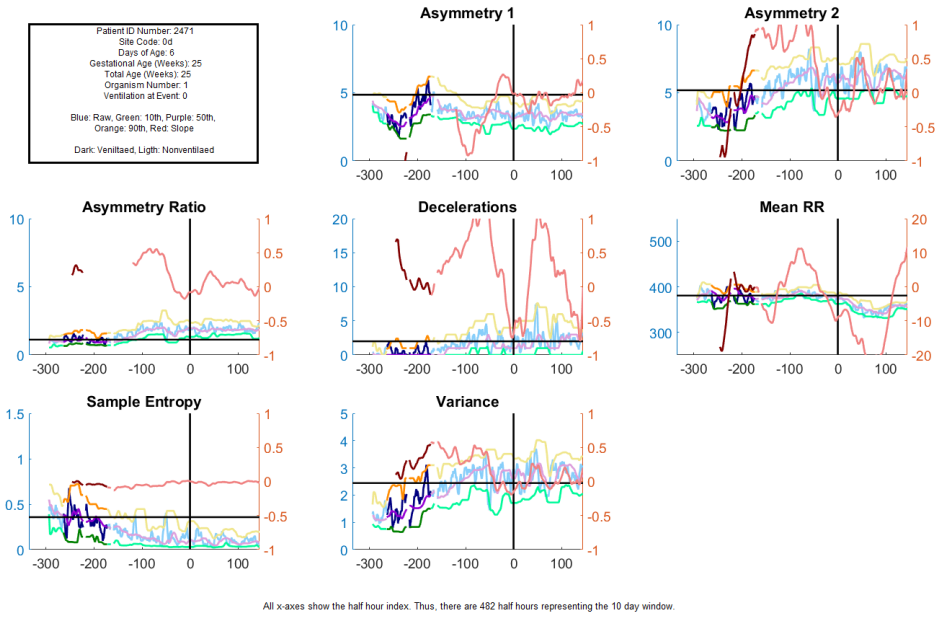


Figure 4.1: Individual Time Series Figure Example

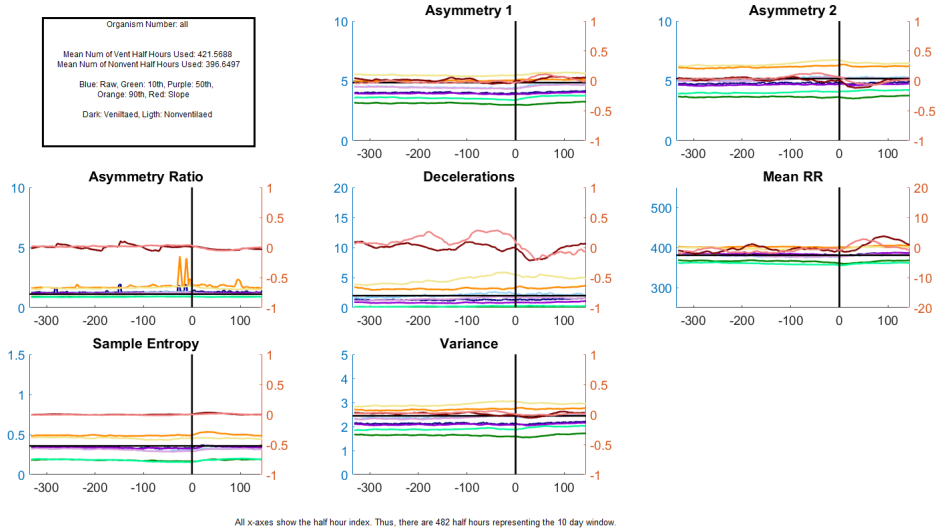


Figure 4.2: Average Time Series Figure

4.2 Time Series Discussion

The major result from Figure 4.2 is that all of the average time series are relatively flat. We notice slight changes in the HRCs before an event, but nothing that gives us a strong indication that an HRC is very predictive. The individual plots can help explain this

result. Most seem to fluctuate quite a bit, with some babies having very low HRCs and some babies having very high HRCs.

While the time series plots are useful references to visualize the data for a particular baby, they do not provide an adequate method for predicting sepsis. We therefore must resort to more sophisticated techniques, which we will discuss in the next section.

Chapter 5

Univariate Probability Density Functions

We can now use the CSV files to make probability density functions (PDFs) of the HRCs. By comparing PDFs from different groups of half hours, we can gain insight into which HRCs are more predictive. For example, recall that we have now separated the half hours into a sick category and healthy category. Using methods we will describe below, we can create a PDF for a particular HRC using only the sick half hours and a PDF for the same HRC using only the healthy half hours. If the two PDFs look significantly different, we can conclude that the HRC in question is a predictor of sepsis. We then repeat this process for every other HRC.

The naive approach for analyzing any data set is to first make a histogram. The most basic form of this approach is a histogram for one variable. However, since all our HRCs are continuous (with the exception of decelerations), we really want PDFs and not histograms. One approach for making PDFs is to assume an underlying distribution and then calculate point estimators for the parameters of that distribution using the data set. Unfortunately, we do not believe we can model our HRCs with any known distribution. Therefore, we resort to nonparametric methods for creating the PDFs. The method we use is called kernel density estimation. We will explain how this method works, the results of this method, and the conclusions from our PDFs in the following sections.

5.1 Kernel Density Estimation Definition

All our PDFs in this report are generated using kernel density estimation. In this section, we will explain how kernel density estimation works. Accordingly, first assume we have an independent and identically distributed sample $x_1, x_2, \ldots, x_n$. The kernel density estimator $\hat{f}_X(x)$ for the PDF of X is then

$$\hat{f}_X(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{5.1}$$

The function $K(\cdot)$ in Equation 5.1 is the kernel. For our report, we use the Epanechnikov kernel which is given below:

$$K(u) = \begin{cases} \frac{3}{4}(1 - u^2) \ , & |u| \leq 1 \\ 0 \qquad\quad , & otherwise \end{cases} \tag{5.2}$$

One can think of the kernel density estimator (KDE) as a smoothed histogram. Consider Figure 5.1: the left panel shows a histogram for a data set, and the right panel shows the KDE for that same data set. In this data set, there are six data points represented by the ticks on the x-axis. The kernels in red are then calculate for each data point. Summing all the kernels together yields the solid blue KDE curve. Note that the author of this figure uses a normal kernel [10].

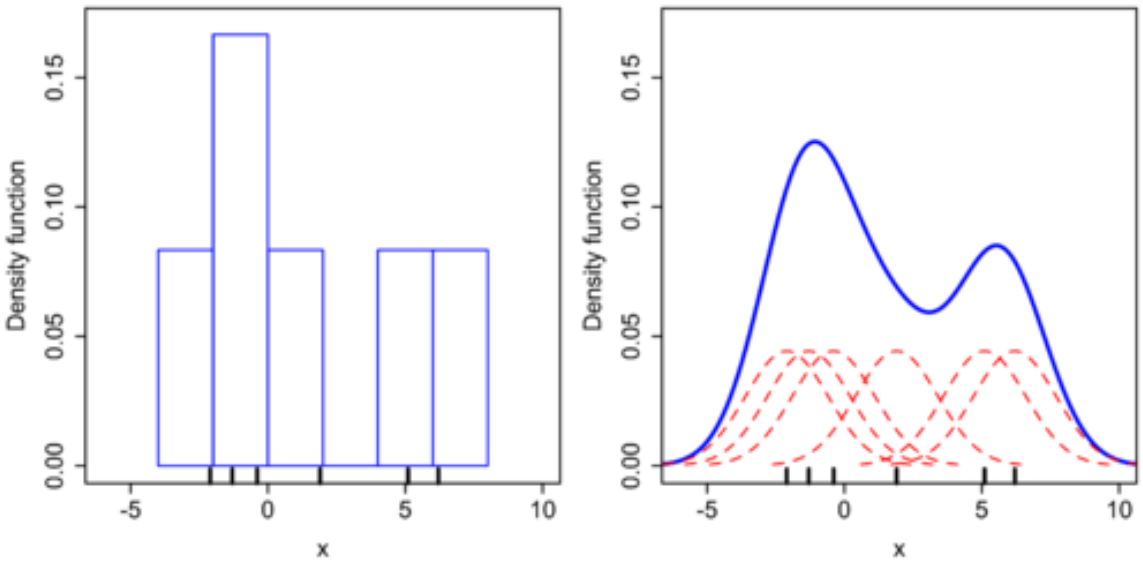


Figure 5.1: Comparison of Histogram and Corresponding KDE [10]

The kernel density estimation method is implemented in Matlab using the function `ksdensity()`. The two important choices we have to make are the kernel $K(\cdot)$ and the bandwidth h. We chose the Epanechnikov kernel because it is "optimal in the mean square error sense" [10]. The parameter h acts a smoothing parameter. Typically, we use Matlab's default bandwidth. However, for reasons we will explain later, sometimes we must set the bandwidth ourselves. We use the Freedman-Diaconis method for choosing the bin width of a histogram to generate the bandwidth of our KDEs [3]. This method is given below:

$$bin\ width = 2\ IQR\ n^{-\frac{1}{3}} \tag{5.3}$$

Here, IQR stands for interquartile range and n is the number of observations. While Matlab uses a similar procedure to calculate the bandwidth, we may want to use a uniform bin width across various PDFs where the number of observations differ. Typically, we use the n and IQR from the unventilated organism 3 group because this is the smallest subgroup we analyze. Therefore, whenever we compare subgroups, we are limited by the observations of our smallest category.

5.2 Univariate PDF Figures

Univariate PDF Figures: `multiple_univariate_pdf_figures` → `one_univariate_pdf_figure`

We can now generate univariate PDF figures using the kernel density estimation method. Figure 5.2 depicts an example of one of these figures. In Figure 5.2, we plot the raw HRCs for the group of sick half hours across all babies and the group of healthy half hours across all babies. Note that the healthy PDFs were generated using more half hours than the sick PDFs as indicated by the legend in the top left corner.

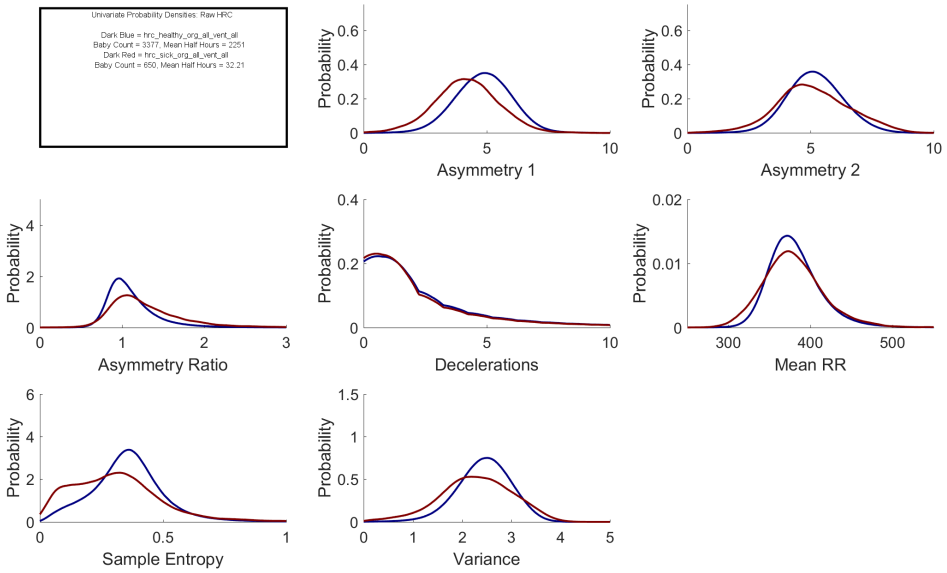


Figure 5.2: Univariate PDF Example

Appendix A contains more univariate PDF figures. Figure A.4 shows the PDFs for the raw HRCs from all the unventilated sick half hours separated by each organism. We also plot the PDF for the raw HRCs across all half hours for comparison. The next five figures correspond to the group of half hours from babies who experienced a septic event caused by organism 1 while they were in the NICU. There are five figures for this group for the five HRC subcategories (raw HRC, 10th percentile, 50th percentile, 90th percentile, and slope). Thus, we collectively have 35 subplots amongst the five figures for our 35 HRCs. On each subplot of each figure, we plot a PDF for the subgroups within the main group of organism 1 of sick ventilated, sick unventilated, healthy ventilated, and healthy unventilated. We can then repeat this process for every other organism. We can also repeat this process for any groups of half hours we wish to compare. However, the litany of additional PDF figures have been omitted from this report. For more information on the creation of these figures, see the documentation of `multiple_univariate_pdf_figures.m`.

5.3 Univariate PDF Discussion

When comparing healthy PDFs to sick PDFs using all relevant figures, we notice significant difference across all HRCs and HRC subcategories with the exception of decelerations. Consequently, we have our first major piece of support that the majority of the HRCs can distinguish between healthy and sick half hours. We also see significant differences when

comparing ventilated PDFs with unventilated PDFs for figures that display this information. This result leads us to separate the analysis of ventilated and unventilated half hours in the later chapters. Another result we find is that there is little difference in the PDFs of the five organisms. We thus conclude that it is much more difficult to determine the type of organism that caused the illness if we know a baby is ill when compared to determining if a baby is sick or healthy. At this point in the report, we will stop separating half hours by organism. Rather, we will focus on separating groups by healthy or sick and ventilated or unventilated.

Chapter 6

Univariate Risks

In the previous section, we plotted the PDFs of two groups in order to compare them. While this method provides a good visual way of comparing two groups, we cannot take away any quantitative information. Ideally, we want to define a measure, which we will call the risk, that relates the two PDFs. In this section, we will define and analyze the risk.

6.1 Risk Definition

The risk gives us a number indicating if a half hour is more likely to be in the sick group or healthy group. Accordingly, let us first define the probability of a half hour being sick or $P(sick)$:

$$P(sick) = \frac{number\ of\ sick\ half\ hours}{number\ of\ sick\ half\ hours + number\ of\ healthy\ half\ hours} \tag{6.1}$$

Note that the number of sick/healthy half hours changes depending on the subgroup we are concerned with. For example, if we are calculating a risk for unventilated half hours, the number of sick/healthy half hours must be calculated using only this group. Next, we will define the probability of a half hour being sick given an HRC signal or $P(sick|signal)$:

$$P(sick|signal) = \frac{P(signal|sick)P(sick)}{P(signal|healthy)P(healthy) + P(signal|sick)P(sick)} \tag{6.2}$$

Since we are dealing with univariate risks, the variable "signal" will refer to one HRC (e.g. $P(sick|variance = x)$). When we discuss bivariate risks in Chapter 8, signal will refer to HRC 1 and HRC 2 (e.g. $P(sick|variance = x \quad and \quad sample \quad entropy = y)$). Next, we can calculate the terms $P(signal|sick)$ and $P(signal|healthy)$ using the PDFs generated in Chapter 5. Because those terms are probabilities, we integrate the PDFs over the signal's respective bin width centered around the value of the signal to get those two terms. Moreover, integrating the PDF allows us to find the probability between two points. We therefore estimate to probability terms by integrating over a small area defined by the bin width. Finally, our definition of risk is

$$Risk = \ln\left(\frac{P(sick|signal)}{P(sick)}\right) \tag{6.3}$$

Since we take the natural logarithm of this fraction, any number above 0 indicates that the half hour is more likely to be sick given the signal. In the next section, we will create figures that plot the risk over typical values of each HRC signal.

6.2 Univariate Risk Figures

Univariate Risk Figures: `multiple_univariate_risk_figures` → `one_univariate_risk_figure` → `one_risk_matrix` → `one_prob_matrix`

Figure 6.1 gives an example of what one of the univariate risk figures looks like. This figure illustrates the risk of a half hour being sick across all half hours (ventilated and unventilated). The figure shows seven subplots for the seven HRCs. On each subplot, we have five curves for the five HRC subcategories. Accordingly, this figure gives the risk for all 35 HRCs. If the probability of a certain HRC is very small, we do not plot the risk. For example, a variance value of 4.5 is so rare that we cut off the curve before this point.

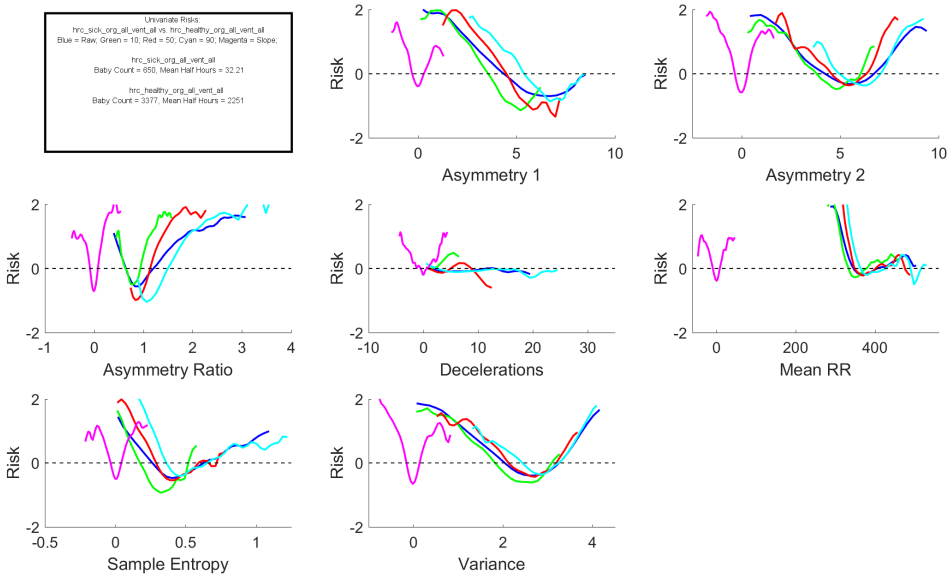


Figure 6.1: Univariate Risk Example

Appendix A contains two more risk figures: one for only unventilated half hours and another for only ventilated half hours. Again, we are not separating the half hours by organism anymore. However, if we were to separate half hours this way, the risk of organism x compared to organism y would be very flat and around 0 across all HRCs, indicating that the HRCs do not do a very good job at distinguishing between two organisms. For more information about how we create the univariate risk figures, please see the documentation of `multiple_univariate_risk_figures.m`.

6.3 Univariate Risk Discussion

When looking at the univariate risk figures, we hope to find curves that are significantly

above the dotted 0 line in each subplot. Effectively, such a scenario would indicate that these HRC values would be very predictive of sepsis. However, the figures can be misleading. For example, consider a variance value of 4 in Figure 6.1. From the figure, a half hour with a variance of 4 would indicate that the half hour is e^2 times more likely than the typical half hour to be sick. We would thus conclude that variance is a very strong predictor of sepsis. The problem with this analysis is that a variance of 4 is very unlikely, so such a value holds little predictive power. What one really must do is look at the univariate risk figures with the univariate PDFs of each HRC in mind. The most useful HRCs would then be ones whose risk is relatively large in areas where the sick PDF is also large. To reiterate this point one more time, if the $P(sick|variance = 4) = 0.002$ and $P(healthy|variance = 4) = 0.001$, we would conclude a baby is e^2 times as more likely to be sick than healthy when the variance is 4. However, since the sick probability is so small, the risk is not extremely useful.

If we ignore the distributions of the HRCs for a moment, Figure 6.1 demonstrates that the following values are predictive of sepsis: low asymmetry 1 values, low and high asymmetry 2 values, high asymmetry ratio values, low mean RR values, low sample entropy values, and low and high variance values. Furthermore, low and high values for all slopes seem to predict sepsis as well.

Ideally, we would plot the sick and healthy PDFs of each HRC on the risk plots. However, with five risks on each subplot, adding ten more PDFs would become quite cumbersome. Fortunately, the figures in Chapter 9 will give a similar illustration of such a figure.

Chapter 7

Bivariate Probability Density Functions

After developing methods to calculate the probability of illness given one HRC, we can extend these methods to two HRCs. We will still use the kernel density estimation method to generate PDFs. However, now we will sum over bivariate Epanechnikov kernels instead of univariate ones. Because Matlab's default bandwidth smooths the bivariate PDFs too much, we set our own bandwidths for the bivariate PDFs. We use bandwidths that are half of the bin widths calculated by Equation 5.3. While our choice for halving the bandwidths is empirical, the bandwidths of the HRCs for a bivariate PDF must be smaller than the bandwidths for a univariate PDF so that the rectangle we smooth over in the bivariate case is somewhat proportional to the line segment we smooth over in the univariate case.

Again, we use Matlab's `ksdensity()` function to generate the bivariate KDEs. The rest of the chapter will describe the resulting bivariate figures and analyze their meaning.

7.1 Bivariate PDF Figures

Bivariate PDF Figures: `multiple_bivariate_pdf_figures` → `one_bivariate_pdf_figure`

Similar to the univariate case, we will create bivariate PDFs for different combinations of HRCs. However, with 35 HRCs, we would end up with 595 bivariate PDFs. Therefore, we will only analyze the six bivariate PDFs which we empirically flagged as the most useful: raw variance vs. raw sample entropy, raw variance vs. raw asymmetry ratio, raw variance vs. raw decelerations, raw sample entropy vs. raw asymmetry ratio, raw sample entropy vs. raw decelerations, and raw asymmetry ratio vs. raw deceleration.

Figure 7.1 provides an example of a figure with these six bivariate PDFs. The figure compares the sick half hours across all babies against the healthy half hours across all babies. Appendix A provides two more examples of these figures: one comparing unventilated sick and healthy half hours and one comparing ventilated sick and healthy half hours. The documentation of `multiple_bivariate_pdf_figures.m` contains more info on these figures.

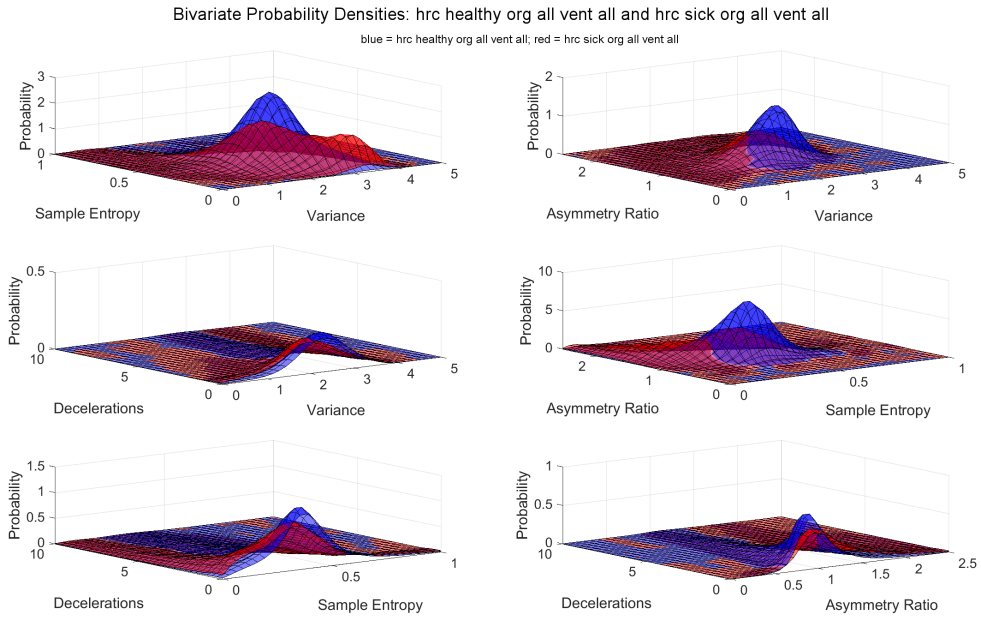


Figure 7.1: Bivariate PDF Example

7.2 Bivariate PDF Discussion

Parallel to the analysis for the univariate PDFs, we are looking for disparate sick and healthy bivariate PDFs for one set of HRCs. Figure 7.1 illustrates that the combination of variance and sample entropy is a very good predictor of sepsis. Any set with deceleration, alternatively, produces very similar sick and healthy PDFs. Overall, Figure 7.1 shows strong support for the combinations of raw variance with raw sample entropy, raw variance with raw asymmetry ratio, and raw sample entropy with raw asymmetry as good predictors. Other HRCs, however, might be more useful for different groups and needs to be looked into further.

Theoretically, with an infinite number of observations, we could calculate the probability of illness given n HRCs using an n-variate PDF. However, we believe with our data set, the best we will be able to calculate is probabilities using two HRCs. Visualizing the probabilities given more than two HRCs would also be quite a challenge. Therefore, we will have to resort to different statistical techniques in the later chapters which will allow us to incorporate more than two HRCs. However, first we will analyze the bivariate risks in the next chapter.

Chapter 8

Bivariate Risks

The bivariate risks mirror Equation 6.3 except this time *signal* refers to two HRC signals. Furthermore, we calculate the terms $P(signal|healthy)$ and $P(signal|sick)$ by integrating the bivariate PDFs over a small rectangle around the value in question. The dimensions of the rectangle are given by the bin widths of the two HRCs. The next step is to make bivariate risk figures similar to the univariate ones.

8.1 Bivariate Risk Figures

Bivariate Risk Figures: `multiple_bivariate_risk_figures` → `one_bivariate_risk_figure` → `one_bivariate_risk_plot` → `one_risk_matrix` → `one_prob_matrix`

Figure 8.1 depicts the bivariate risks corresponding to the bivariate PDFs in Figure 7.1. Furthermore, Appendix A gives the bivariate risk figures for the bivariate PDF figures in the appendix. The function `multiple_bivariate_risk_figures.m` creates these figures.

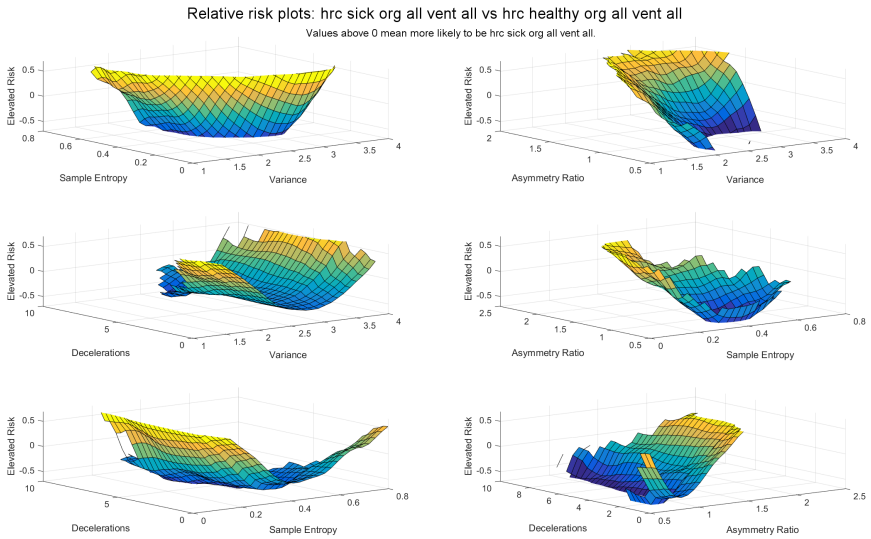


Figure 8.1: Bivariate Risk Example

8.2 Bivariate Risk Discussion

As with the univariate risks, one must keep the corresponding bivariate PDFs in mind when analyzing the bivariate risks. While high risks show the HRC set is useful at predicting sepsis, the set's predictive power is diminished if there is limited data in the high risk area. Any risk above the $z = 0$ plane in Figure 8.1 indicates the half hour is more likely to be sick given that set of HRCs. The analysis of Figure 8.1 provides the same conclusions that we found in Chapter 7, *i.e*, low and high variance are useful, low sample entropy is useful, etc.

Chapter 9

Single Variable Logistic Regression

As stated at the end of Chapter 7, we cannot simply continue calculating n-variate PDFs. Instead, we will resort to logistic regression in order to calculate the probability of illness given n HRCs. Before we can build up to logistic regression with n HRCs, we will start with a single HRC. The rest of this section details the mathematics of single variable logistic regression as well as the resulting figures and analysis.

9.1 Single Variable Logistic Regression Definition

Consider a response vector $[y_1, y_2, \ldots, y_n]^T$ drawn from a binomial distribution with support $y_i = \{0, 1\}$. Now consider a set of explanatory variables $[x_1, x_2, \ldots, x_q]^T$. Using a logistic regression model, the probability that $Y_i = 1$ is then

$$P(Y_i = 1) = \frac{e^{\boldsymbol{\beta}\mathbf{x}}}{e^{\boldsymbol{\beta}\mathbf{x}} + 1} \tag{9.1}$$

where $\mathbf{x}$ is the vector of explanatory variables and $\boldsymbol{\beta}$ is the vector of coefficients for the explanatory variables. The coefficients are calculated using a least squares method which requires the response vector and the matrix of associate explanatory variables. Moreover, each response variable y_i has an associated $\mathbf{x_i}$ for a total of n $\mathbf{x}$-vectors. The i^{th} $\mathbf{x}$-vector becomes one row in the matrix of explanatory variables [11].

For our single variable logistic regression models, $\mathbf{x}$ will have two or three terms depending on whether we use a linear or quadratic model. In order to explain the terms in the vector, consider the example of predicting sepsis using the single HRC of raw variance. The two possibilities for $\mathbf{x}$ are given below:

$$Linear : \mathbf{x} = [1 \;\; variance]^T$$

$$Quadratic : \mathbf{x} = [1 \;\; variance \;\; variance^2]^T$$

Note that the 1 in each vector represents the $y-intercept$. The term $variance$ is simply the raw variance for that half hour and the term $variance^2$ is the square of the raw variance.

For this report, our response vector is 1 if the half hour is sick and 0 if the half hour is healthy. We can then use the Matlab function `fitglm()` to calculate the coefficients for our model using the response vector and the matrix of explanatory variables, i.e., the HRCs. Next, we will construct figures for the probability of illness given various values of a single HRC.

9.2 Single Variable Logistic Regression Figures

Single Variable Logistic Regression Figures: `multiple_univariate_logistic_figures` → `one_univariate_logistic_figure` → `one_risk_matrix` → `one_prob_matrix`

We can now run our single variable logistic regression on all 35 HRCs. Figure 9.1 gives the results of the regression for the raw HRCs using all the healthy and all the sick half hours. The logistic regression probabilities are given by the blue curve. We use a quadratic model for each of these figures. The red curve on each subplot, which we call the Bayesian probability, is simply the $P(sick|signal)$ term from the univariate risks. Finally, the red and blue areas are the distributions of the HRCs where the blue region represents healthy half hours and the red region represents the sick half hours. Note that the shaded regions are stacked on top of one another, i.e., no region is being hidden. One can interpret the entire shaded regions as the joint distribution of healthy and sick half hours. The scale for the joint distribution is given by the right y-axis.

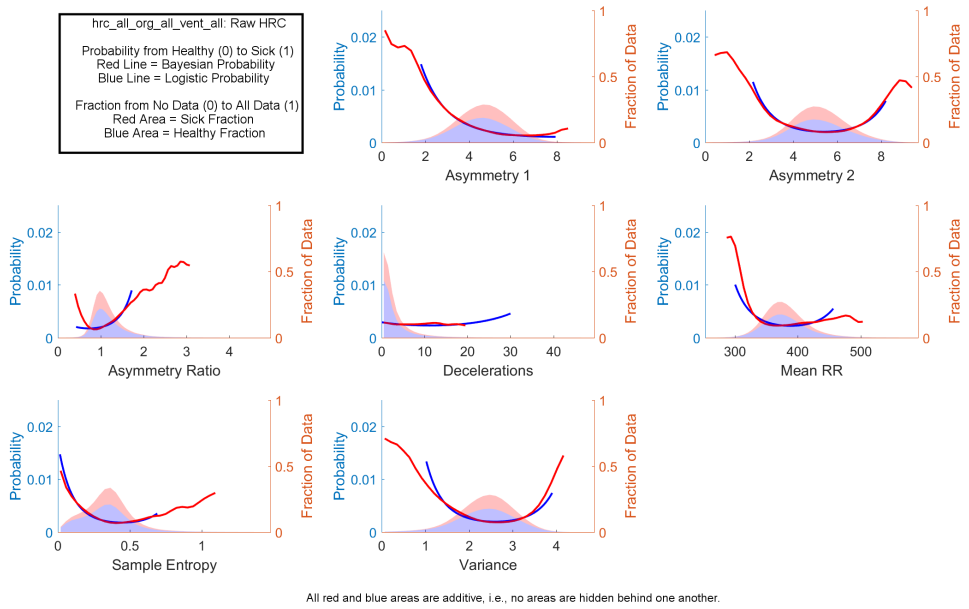


Figure 9.1: Single Variable Logistic Regression Example

Appendix A gives the corresponding figure for the 10th percentile, 50th percentile, 90th percentile, and Slope HRCs. Furthermore, one could calculate the logistic probabilities for all 35 HRCs using a different group of healthy and sick half hours such as only the ventilated or only the unventilated half hours. For more information on these figures, please see the documentation of `multiple_univariate_logistic_figures.m`.

9.3 Single Variable Logistic Regression Discussion

The analysis of the single variable logistic regression follows closely to the analysis the univariate risks. First, recall that the red curves are the $P(sick|signal)$ terms from the univariate risk calculations. One can then think of the red curves as the empirical results

and the blue curves as our fitted model. Likewise, as we addressed for the univariate risks, the most predictive HRCs are the ones who have large sick probabilities/risks in areas with large sick PDFs. The PDFs on the logistic figures, therefore, serve a very useful purpose.

We start with single variable logistic regression to first check our empirical results. We also start with one variable to try to cut down on the number of HRCs before we move to the n-variable logistic regression. If we use all 35 HRCs for the logistic regression, we would have 666 terms in our $\mathbf{x}$-vector for a quadratic model. However, a model with so many terms would severely over-fit our data. The single variable logistic regression can, therefore, provide one method to hand pick a few HRCs we think are the most predictive.

Looking at Figure 9.1, we see a lot of HRCs with high probabilities for low values and high values of the HRCs. This phenomenon gives us strong evidence that we should use a quadratic model over a linear model. For example, the variance subplot in Figure 9.1 shows high probabilities on both tails. A linear fit would only capture one extreme. Therefore, we need a reaction term, which gives us a quadratic model, to capture both sides of the distribution.

Not shown in this report are the single variable logistic figure for only the ventilated group and only the unventilated group. The important result from these sets of figures is that the ventilated probabilities whether sick or healthy are about six times greater than the unventilated probabilities. From this finding, one might want to include ventilation status as an additional HRC. However, we do not include ventilation status because different NICUs might have different standards for ventilating babies. Despite this concern, babies who are ventilated do seem to be more likely to be sick. Therefore, we continue to separate half hours into ventilated and unventilated because there may be different optimal parameters and thresholds for each group when determining if a baby is sick or healthy.

Chapter 10

HeRO Score

The groundwork has now been set to conduct n-variable logistic regression which we call the HeRO score. This section will describe various HeRO score models we have developed and the findings from these models.

10.1 HeRO Score Definition

In general, the HeRO score is an n-variable logistic regression model. The formula for the HeRO score is given below:

$$HeRO\ \ Score = \frac{e^{\boldsymbol{\beta}\mathbf{x}}}{e^{\boldsymbol{\beta}\mathbf{x}}+1} \cdot \frac{1}{\mu_0} \tag{10.1}$$

In this formula, we multiply the logistic regression model by the parameter μ_0^{-1} where μ_0 represents the probability of a half hour being sick. Multiplying by this coefficient means that a HeRO score of 1 indicates that the half hour is just as likely to be sick as any half hour. A HeRO score of 2 would mean the half hour is twice as likely to be sick compared to the typical half hour.

Our explanatory variables vector $\mathbf{x}$ can take on many forms. Below gives three models we will analyze in this section:

$$Hrch : \mathbf{x} = [1\ \ var_{10}\ \ (asym2 - asym1)_{10}]^T$$
$$Hrcg : \mathbf{x} = [1\ \ var_{50}\ \ sampen_{10}\ \ asym1_{50}\ \ asym2_{50}]^T$$

$$\begin{aligned} Dienstman : \mathbf{x} = [1\ \ & asym1_{10}\ \ asym1_{slope}\ \ asym2_{90}\ \ asym2_{slope}\ \ asym\_ratio_{50}\ \ asym\_ratio_{slope} \\ & decels_{90}\ \ sampen_{10}\ \ sampen_{slope}\ \ var_{10}\ \ var_{90}\ \ var_{slope} \\ & asym1_{10} * asym1_{slope}\ \ asym1_{10} * asym2_{90} \ldots asym1_{10} * var_{slope} \ldots \\ & {asym1_{10}}^2 \ldots {var_{slope}}^2]^T \end{aligned}$$

Note that the $Hrch$ and $Hrcg$ models are linear models while the $Dienstman$ model is a quadratic model. The $Hrch$ and $Hrcg$ model were developed by Prof. Douglas Lake while the $Dienstman$ model was developed for this report. As a result, the coefficients for the $Hrch$ and $Hrcg$ model were calculated using a different data set and with a slightly different implementation than the coefficients for the $Dienstman$ model. In total, the $Dienstman$ model contains 91 coefficients and contains every reaction term of the 12 initial HRCs. We

picked the HRCs for the *Dienstman* model based on the single variable logistic probabilities that we thought to be the most predictive. We tried not to include too many HRCs for fear of over-fitting. For more information about how to calculate the coefficients for the *Dienstman* model, please see the documentation of `csv_logistic_coeffs.m`.

All HeRO scores use percentiles and slopes instead of the raw HRCs in order to reduce noise when we plot a time series of the HeRO scores in the next section. If HeRO scores only used information from the current half hour, then one outlier would cause the HeRO score to fluctuate considerably, and because the babies are frequently disturbed by the nurses, we can get a significant number of outliers. Therefore, if we look at var_{10} for example, the second lowest variance value over the previous 24 half hours would go into the HeRO score and not the lowest, reducing outliers and noise (10th percentile usually translates to the second lowest value for a 24 half hour window).

10.2 HeRO Score Figures

HeRO Score Figures: `multiple_hero_score_figures` → `one_hero_score_figure`

After we calculate the coefficients for all the HeRO score models, we can make time series of the HeRO scores for every baby who had an event as we did for the time series in Chapter 4. Figure 10.2 provides an example of one of these figures. The vertical black line represents the time of the septic event and the four curves represent four different HeRO scores. The green line, which we call the legacy HeRO score or just HeRO score, is the maximum between the Hrch and Hrcg models, which are indicated on the figure by the plus makers and circle markers, respectively. The gray line is the *Dienstman* HeRO score discussed previously. Finally, the *Dienstman Vent* and *Dienstman Nonvent* HeRO scores are the *Dienstman* model, but instead of using all the half hours to calculate the coefficients, we only use the ventilated or unventilated half hours. We plot *Dienstman Vent* on the figure only when the baby is ventilated, and otherwise we plot plot *Dienstman Nonvent*. Appendix A provides more examples of these figures for other events. All HeRO score figures were developed using `multiple_hero_score_figures.m`.

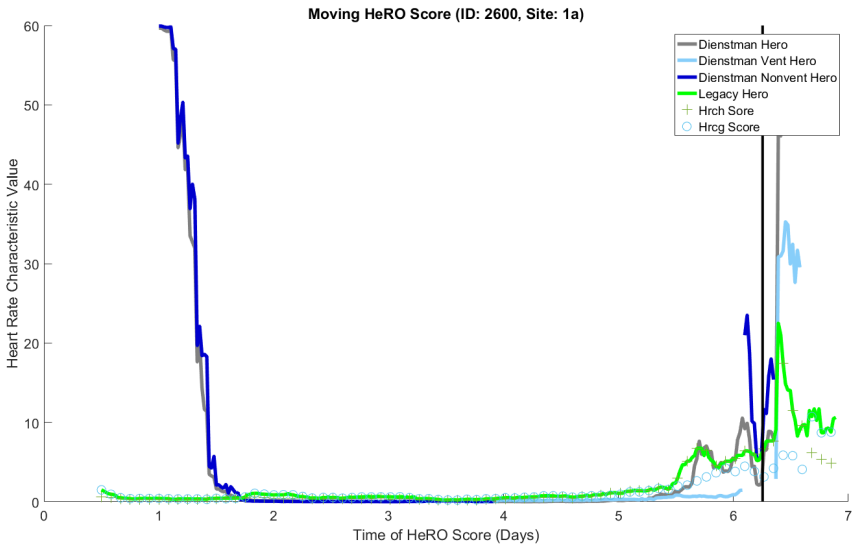


Figure 10.1: HeRO Score Figure Example

After we have created these figures for all events, we can take the average HeRO score leading up the the event in the same manner as we took the average HRC for the time series figures. Figure 10.2 presents the resulting figure for the average HeRO score. Recall that not every baby will have data for every half hour leading up to an event. Subsequently, we record how many half hours we use for each half hour relative to the event and record the average of these numbers in the title of the figure. It should also be noted that *Dienstman Vent* and *Dienstman Nonvent* use about half as many observations as the number noted in the title since a baby can never be ventilated and unventilated at the same time.

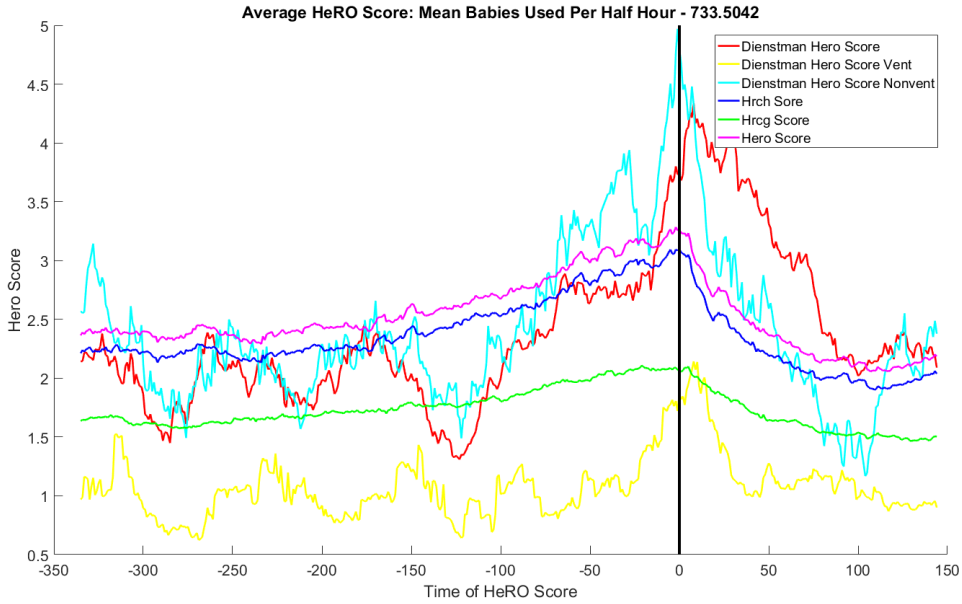


Figure 10.2: Average HeRO Score Figure

10.3 HeRO Score Discussion

The first major feature of the individual HeRO figures is that the *Dienstman* model seems to over fit the data. HeRO scores are either extremely high or extremely low, which is a classic sign of over fitting. The individual figures are also very noisy and do not always provide a clear indication that a high HeRO score will indicate the baby is sick.

The average figure, on the other hand, does show us that the HeRO scores are predicting sepsis. Unlike the time series figures, the HeRO scores increase right before an event on average. Thus, while no individual HRC showed significant predictive power in the average time series, a combination of multiple HRCs certainly provides warning for the onset of sepsis. The legacy HeRO score also provides a very smooth average. This feature is extremely useful because it gives medical professionals more confidence that a high HeRO score is a true indication of illness and not a false positive. However, the individual legacy HeRO scores are still noisy, so the average in this case may be deceiving. We hypothesize that the noise for the average *Dienstman* models comes from over-fitting as one large score at half hour x will skew the average. However, one interesting feature is that the *Dienstman Vent* and *Dienstman Nonvent* scores are significantly different, giving us more support that we should treat the ventilated and unventilated half hours separately.

In general, we hope to see an increase in the HeRO leading up to the event. However, at present, we have not implemented any method for ranking the HeRO scores apart from just looking to see if they increase smoothly before the event. A gradual increase might be useful as it could give early warning that the baby is ill. Conversely, a sharp spike right before the event also has merit as it provides a clear threshold for the moment the baby requires attention. Accordingly, more work needs to be done in order to determine which model performs the best. This will be especially important for testing the performance of any future HeRO score against current ones.

Lastly, note that we used all the half hours to calculate the coefficients for the *Dienstman* scores. Ideally, we should separate the half hours into a learning set and testing set. We would then calculate the coefficients using only the learning set and produce figures for the testing set based off these coefficients. Since we did not take this step, this area could be another cause for noise and over-fitting. Also note that we have many more healthy half hours than sick half hours. Thus, it might also be necessary to randomly select a subgroup of the healthy half hours so we use the same number of healthy and sick half hours to calculate the coefficients. Accordingly, we need to look into this issue more carefully.

Chapter 11

Conclusion

After rigorously analyzing the predictive power of our 35 HRCs, there are many important results to summarize. Most of these findings also lead to more questions and further areas to investigate. We will address both topics in the following sections.

11.1 Discussion

Recall that one of our early goals was to try to distinguish between the types of invading organisms that cause illness. Our original hypothesis was that low variability, or variance, is predictive of gram-positive bacteria and decelerations are predictive of gram-negative bacteria. By analyzing the univariate PDFs, we saw that no HRC performed particularly well in identifying the invading organism. Therefore, we would either need to develop new HRCs to accomplish this goal or analyze the current HRCs using a different method.

We were, however, able to show substantial difference in the PDFs of sick and healthy half hours. The HRCs of variance, sample entropy, and asymmetry ratio were particularly useful in this respect. These three HRCs had univariate and bivariate PDFs that differed significantly between their respective sick and healthy PDFs. The univariate and bivariate risks also demonstrated this discovery.

We also repeatedly saw that ventilated and unventilated half hours behave very differently. The HRCs for these two groups gave very distinct distributions, risks, and HeRO scores. Subsequently, we have given strong evidence for the use of separate parameters, thresholds, and baselines for these two groups in any future work.

Lastly, the HeRO scores showed promising signs for the ability to predict illness using many HRCs. We are limited in the analysis we could do through Bayesian methods by the size of our data. Thus, a logistic regression based model proved to be a viable way to predict illness using multiple HRCs. However, the HeRO scores were very noisy for the individual HeRO score figures, and the *Dienstman* scores seemed to over-fit that data. Thus, we need to address these issues in future work.

11.2 Future Work

A significant portion of the future work involves improving the HeRO score. As mentioned at the end of Chapter 10, we need to reduce the noise in some of the scores. One possible solution is to reduce the number terms in the explanatory vector. We could remove some reaction terms so we have a mix of a pure linear and pure quadratic model. We could

also remove some HRCs entirely. We also need to separate data into a learning set and testing set as well as investigate whether we need to use the same number of healthy half hours as sick half hours for calculating the coefficients.

We also want to try to create the best HeRO score possible for predicting sepsis. Such a goal might require adding more HRCs to a model. We could also have HeRO scores for other groups beyond just ventilated and unventilated. These groups may be based on gestational age, post-menstrual age, gender, and race. The HeRO scores for these groups might also require different HRCs as opposed to just different coefficients for the same HRCs. We would also need a reasonable way to compare the performance of all these different HeRO scores in order to determine which one we would prefer the most. Such a test should consider how many false positives and negatives the score gives and how early of a warning does the score give.

Finally, we could also try to look towards other statistical techniques for predicting sepsis beyond logistic regression. Possibly models include support vector machines or neural networks.

Appendix A

Additional Figures

This appendix contains additional figures not in the chapters above. Note that this appendix is not a complete list of figures produced from the research. Because of the large number of figures, we have elected to only give a few more examples of the types of figures we discussed. However, we can produce more figures upon request.

A.1 Time Series Figures

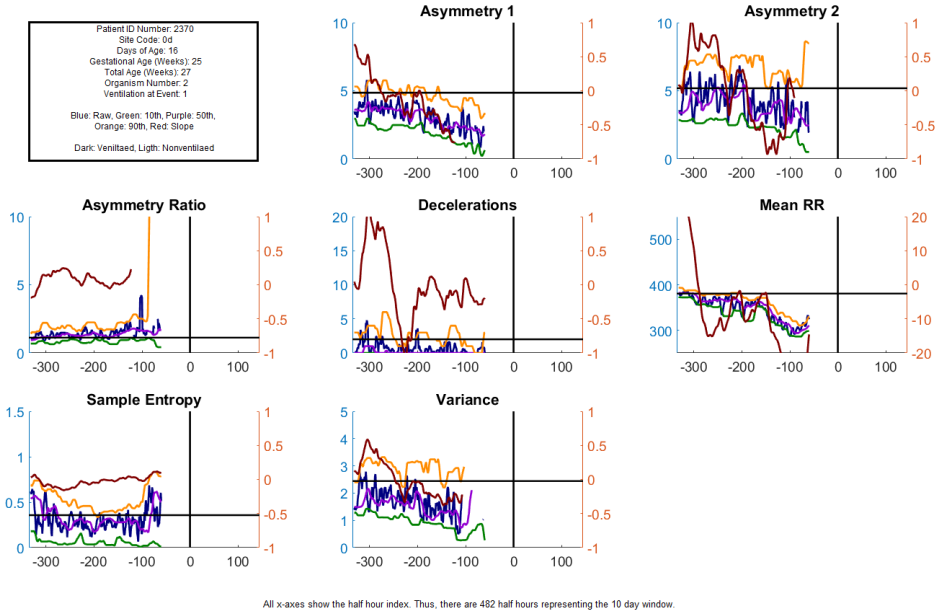


Figure A.1

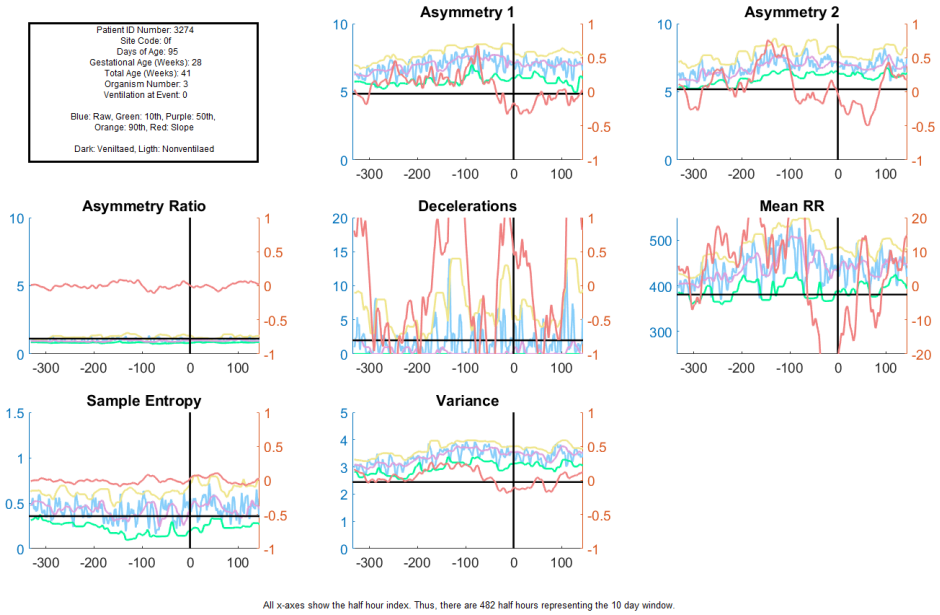


Figure A.2

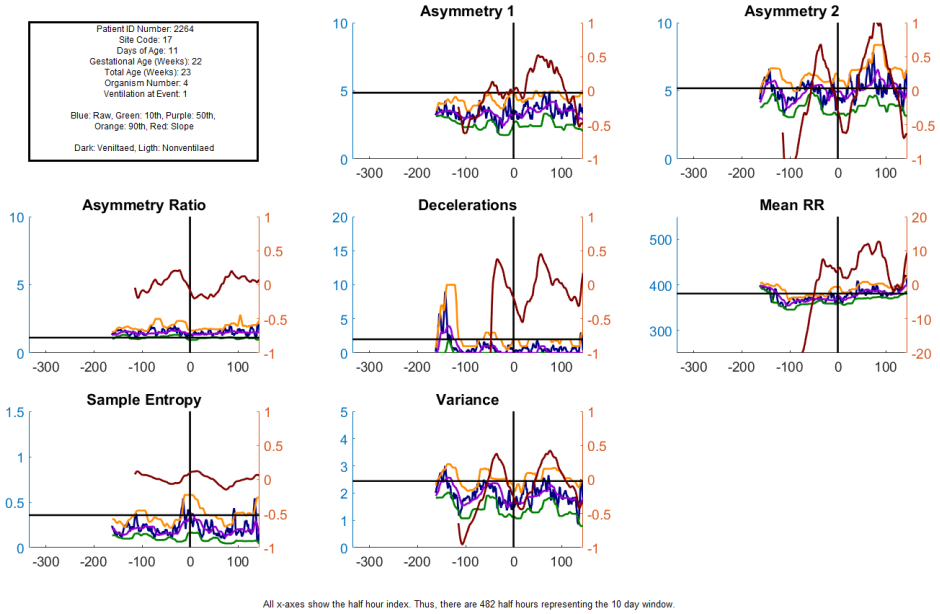


Figure A.3

A.2 Univariate PDF Figures

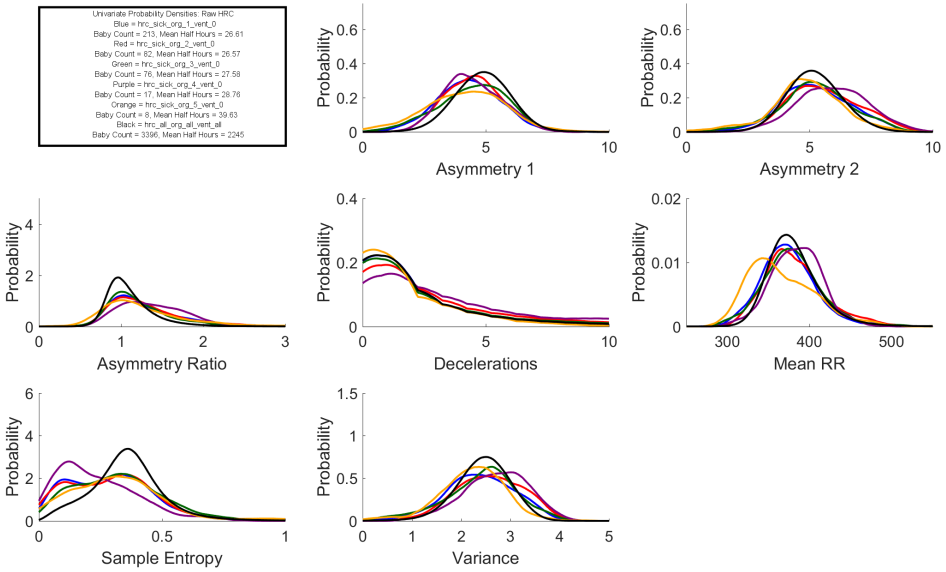


Figure A.4

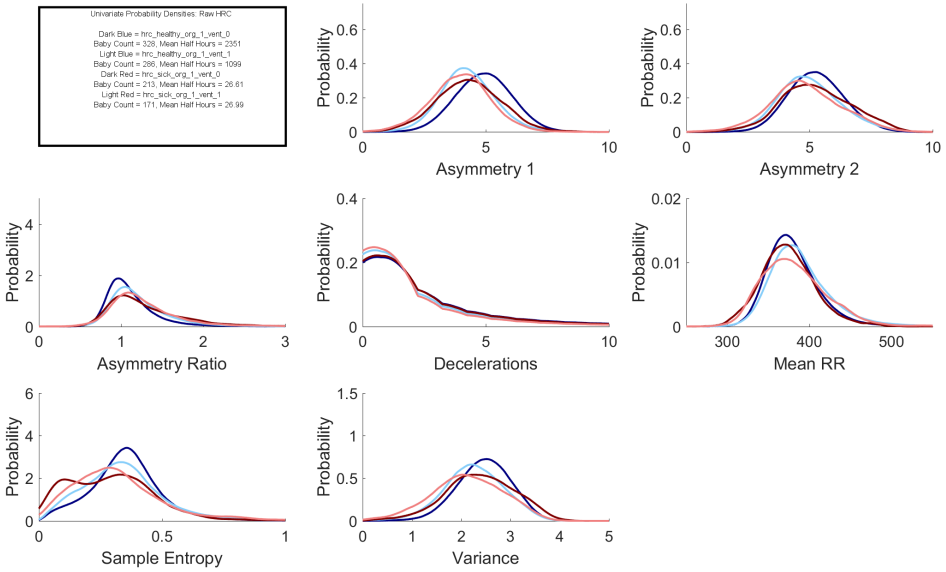


Figure A.5

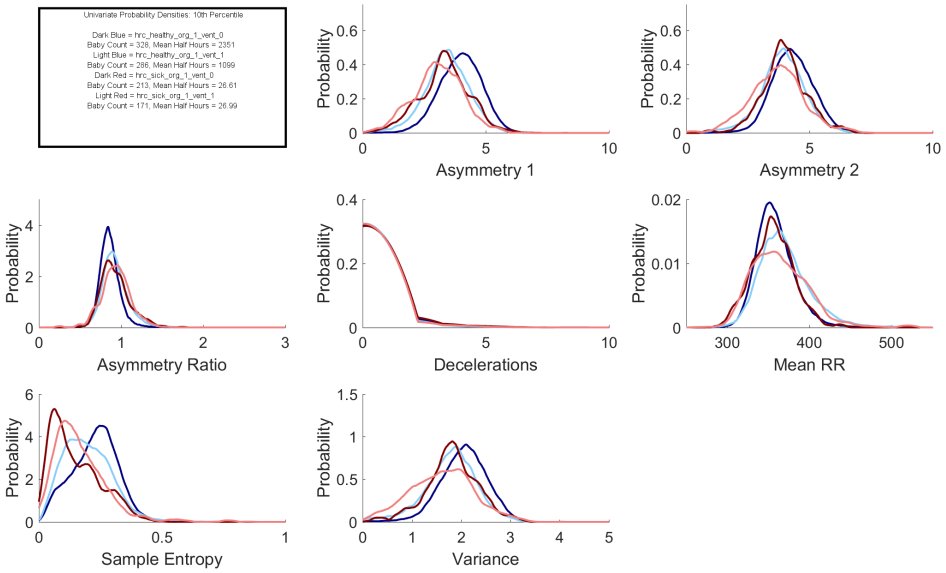


Figure A.6

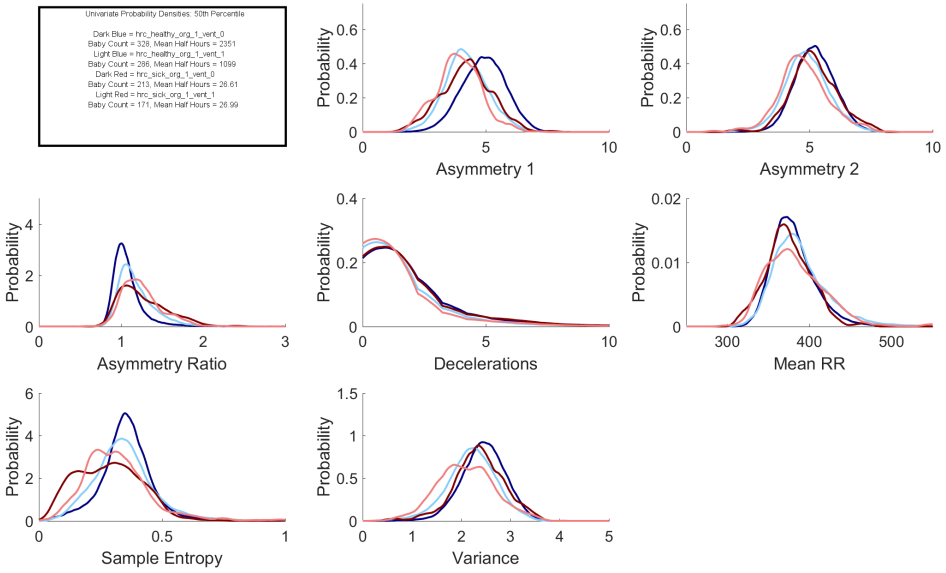


Figure A.7

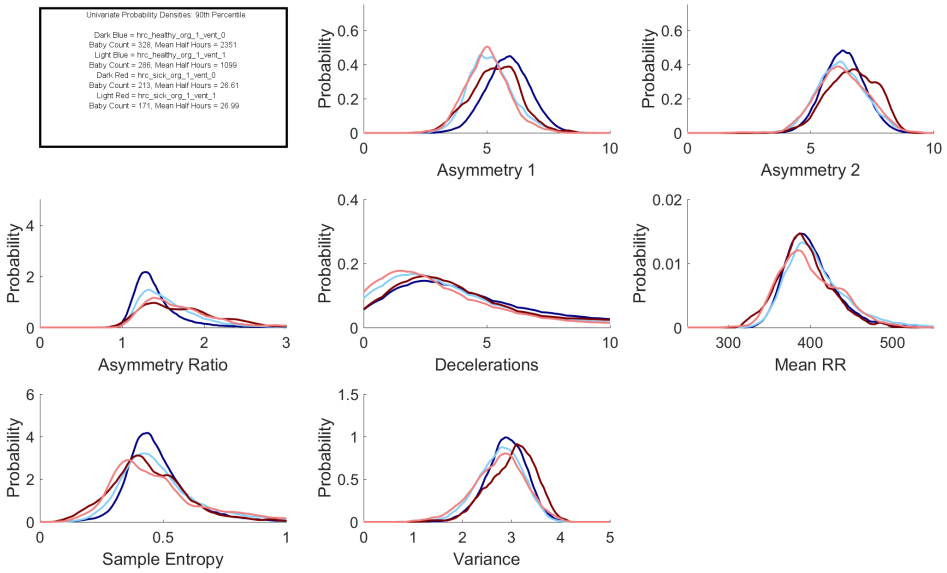


Figure A.8: Univariate PDF 5

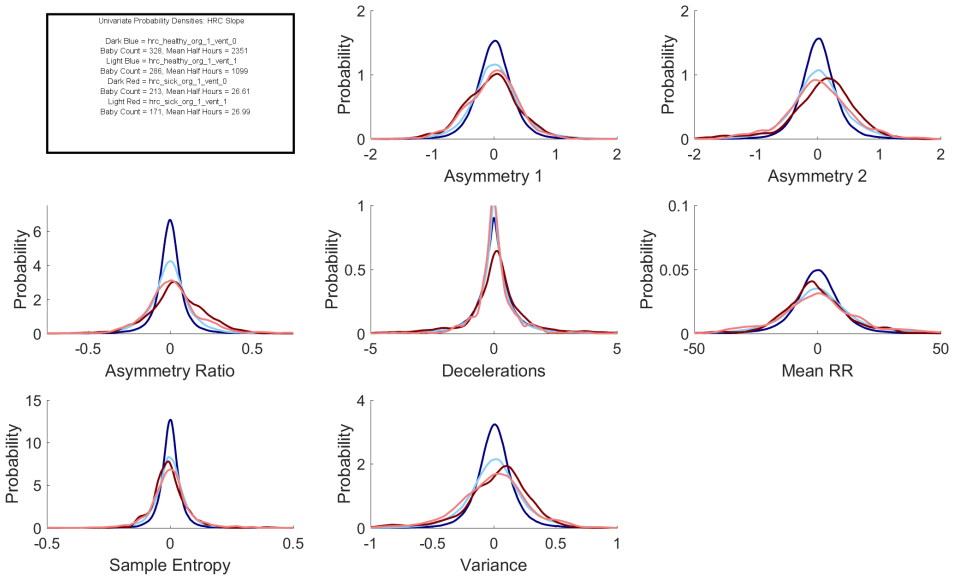


Figure A.9

A.3 Univariate Risk Figures

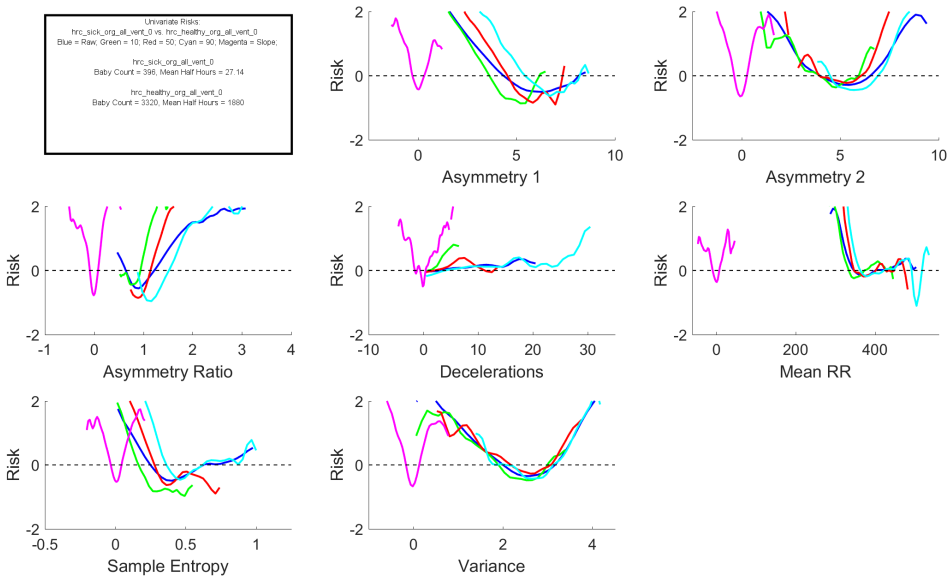


Figure A.10

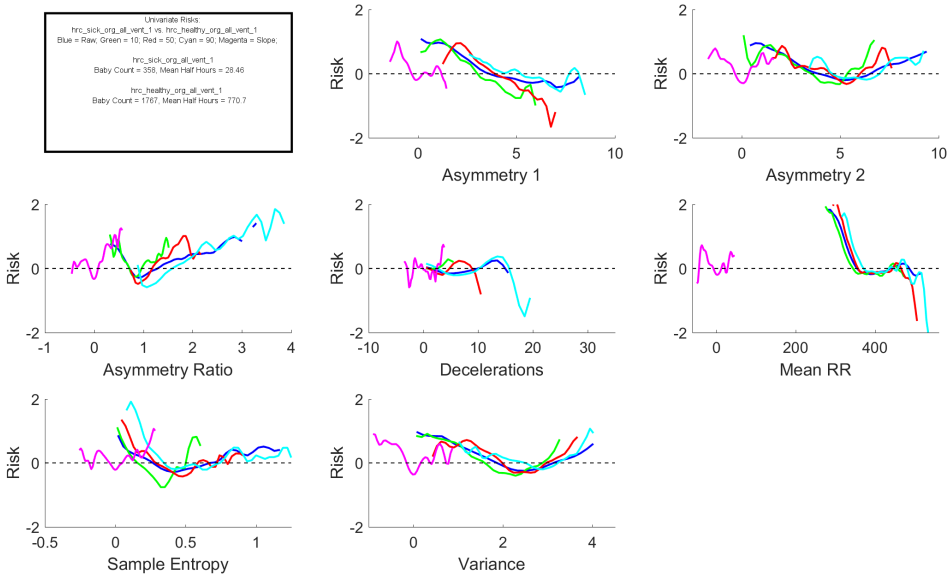


Figure A.11

A.4 Bivariate PDF Figures

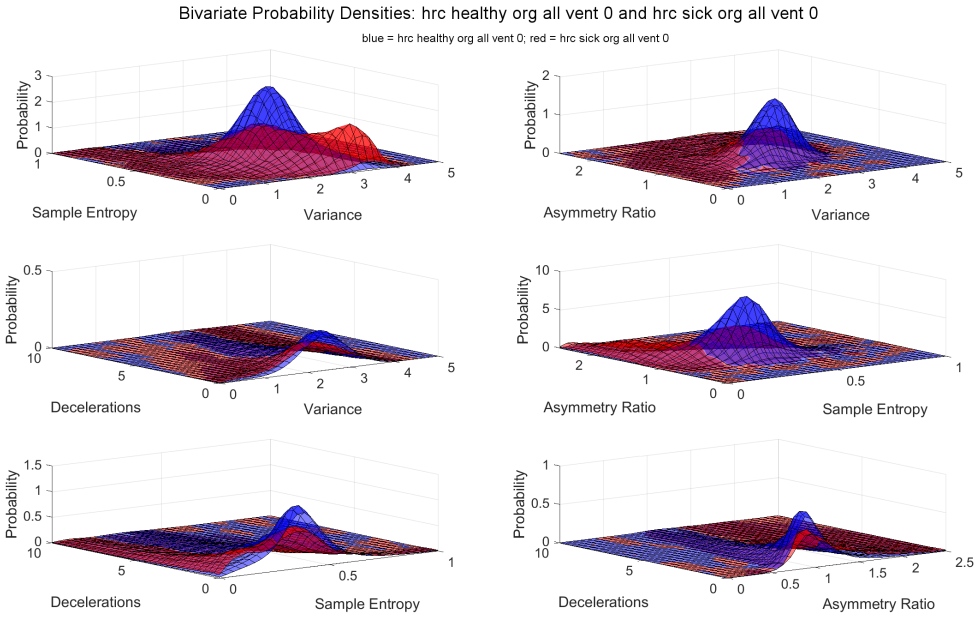


Figure A.12

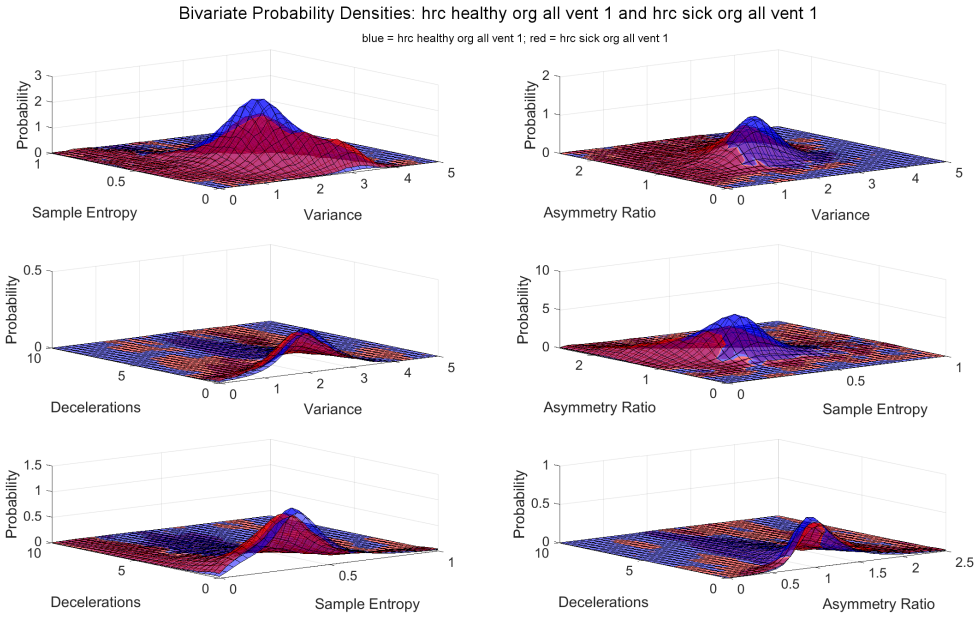


Figure A.13

A.5 Bivariate Risk Figures

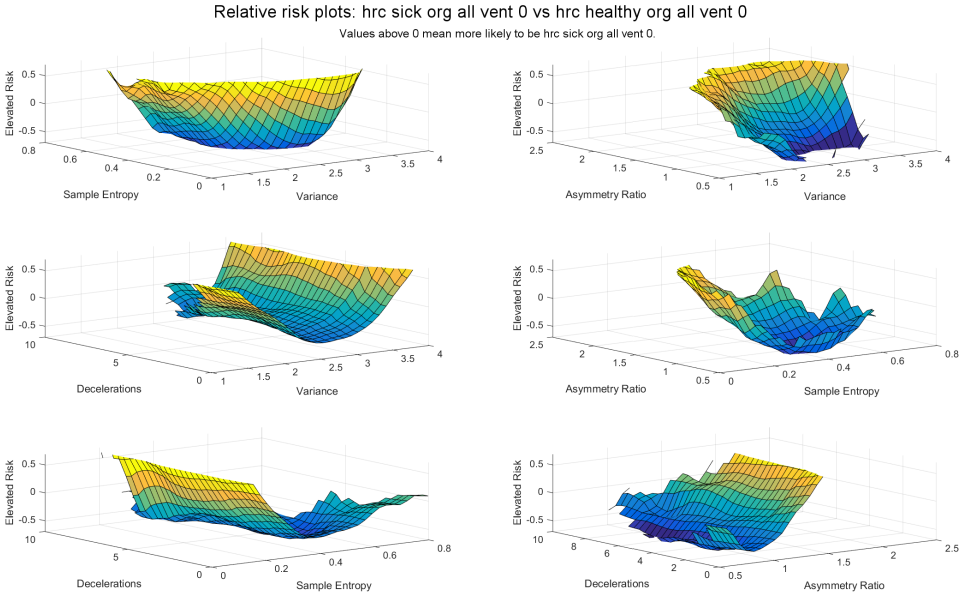


Figure A.14

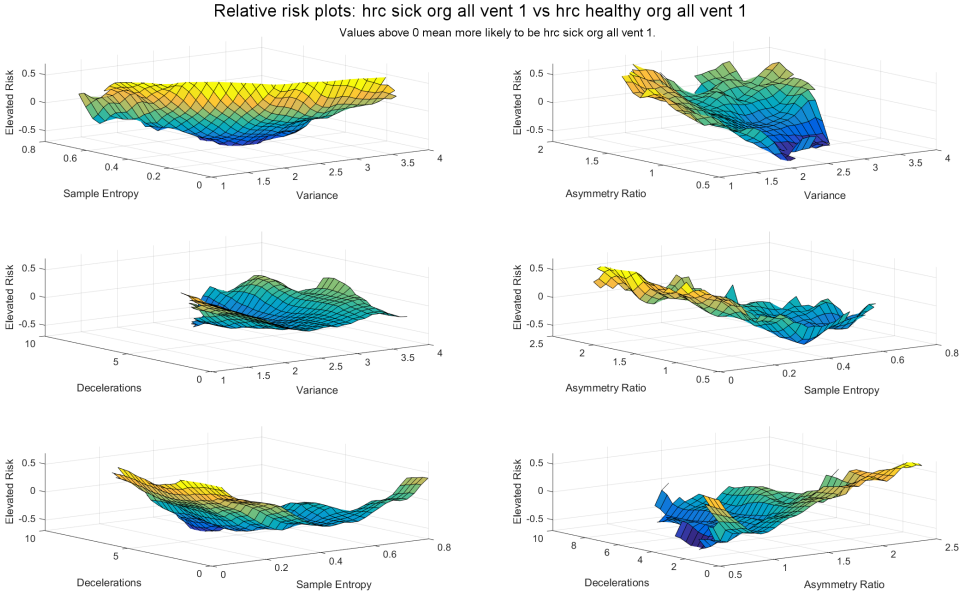


Figure A.15

A.6 Single Variable Logistic Figures

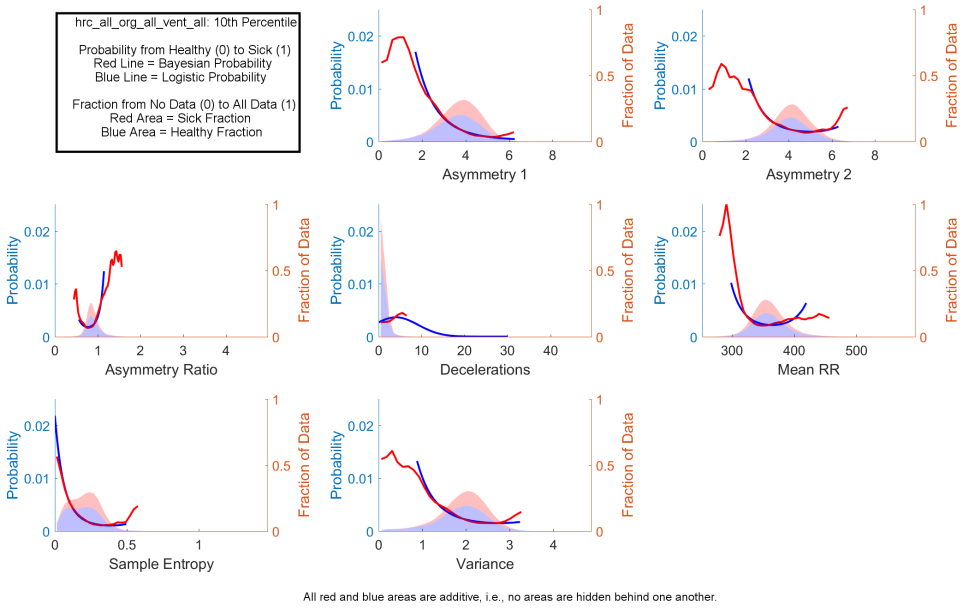


Figure A.16

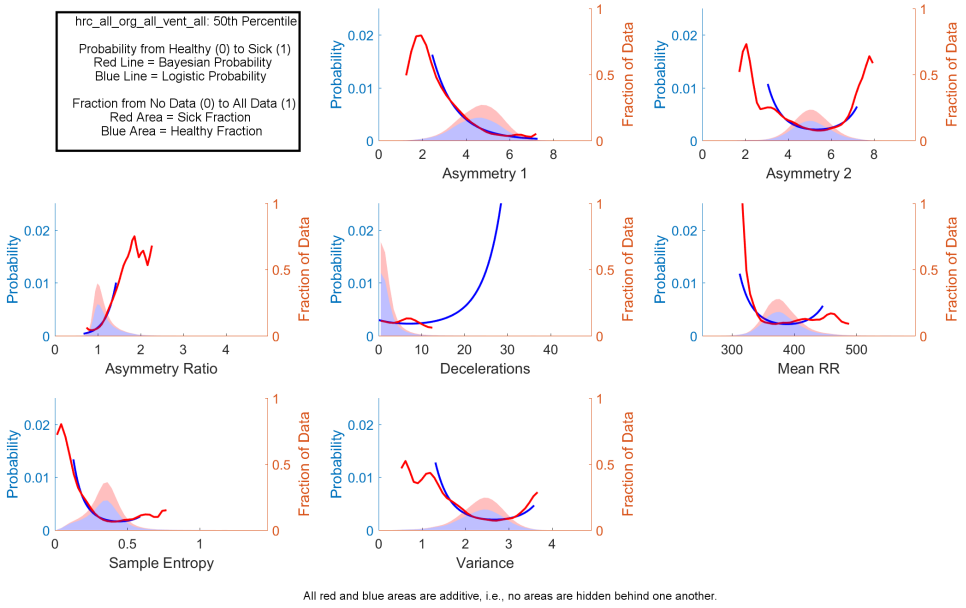


Figure A.17

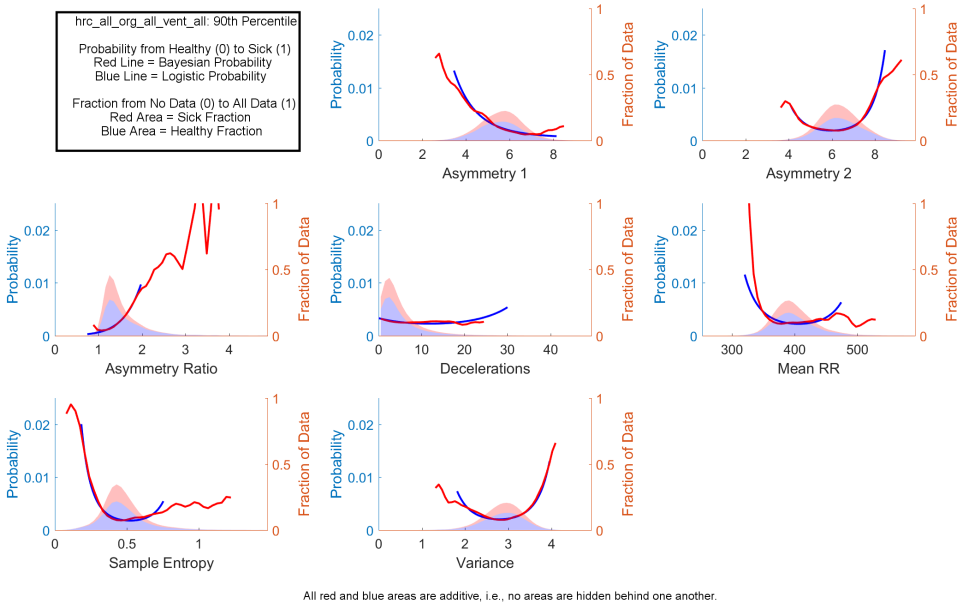


Figure A.18

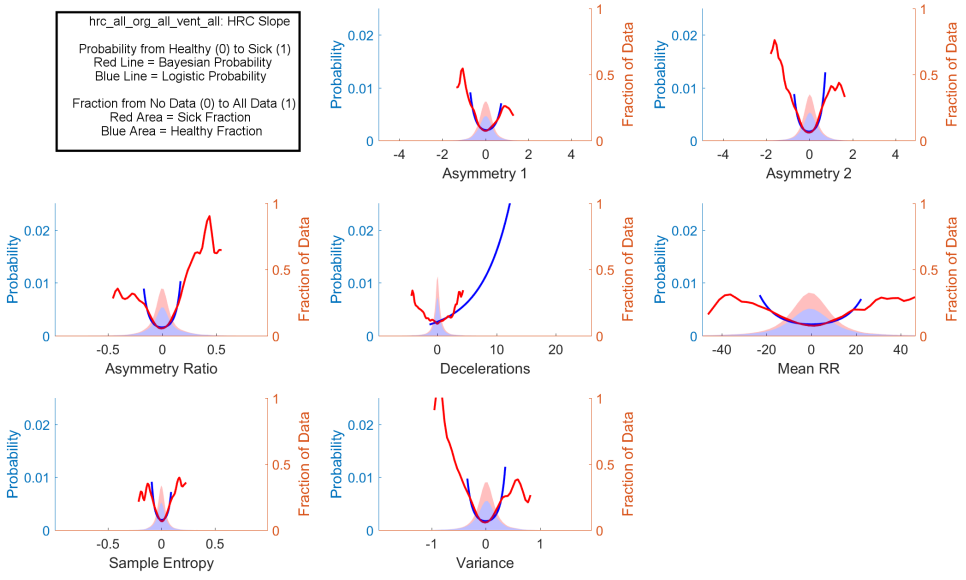


Figure A.19

A.7 HeRO Score Figures

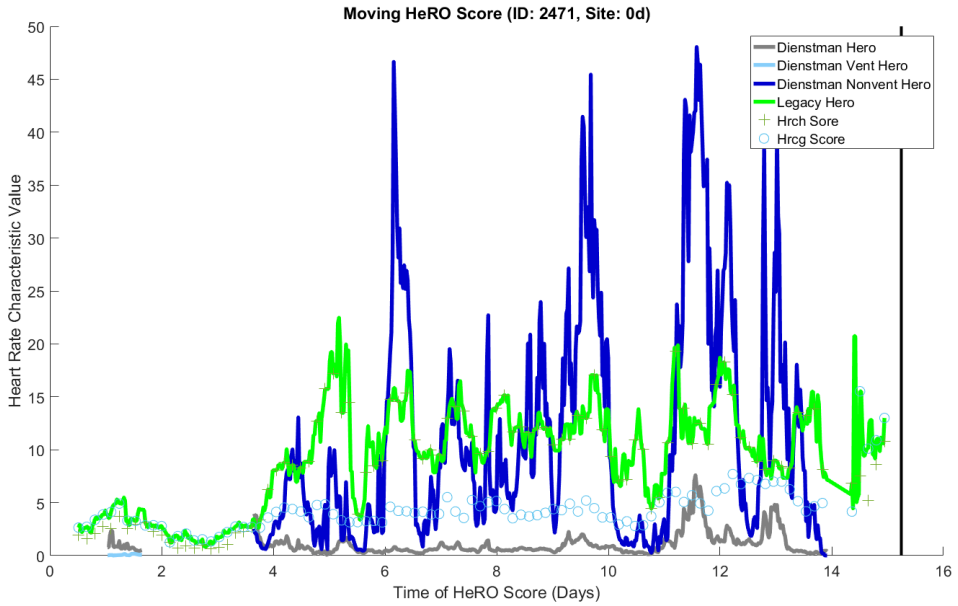


Figure A.20

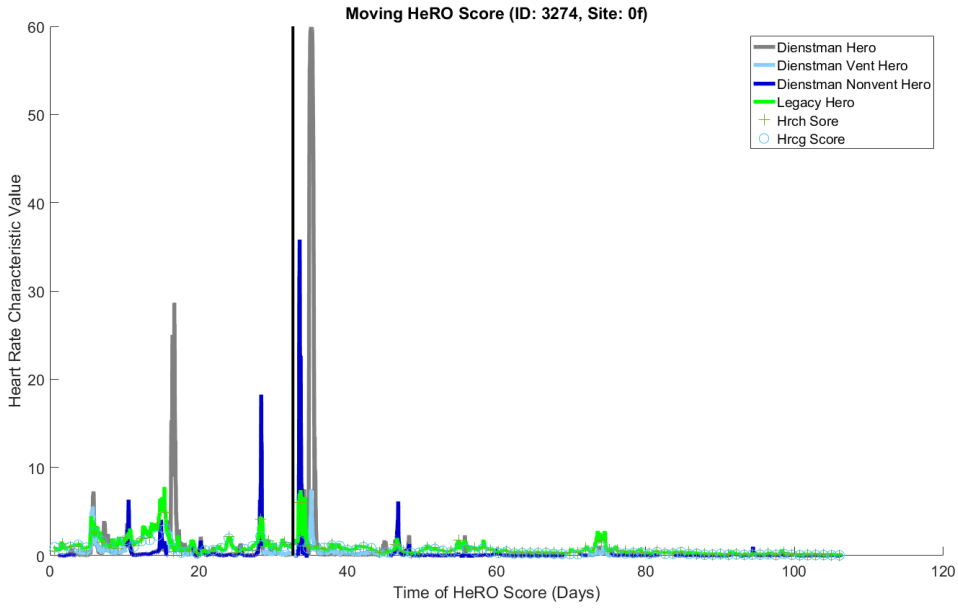


Figure A.21

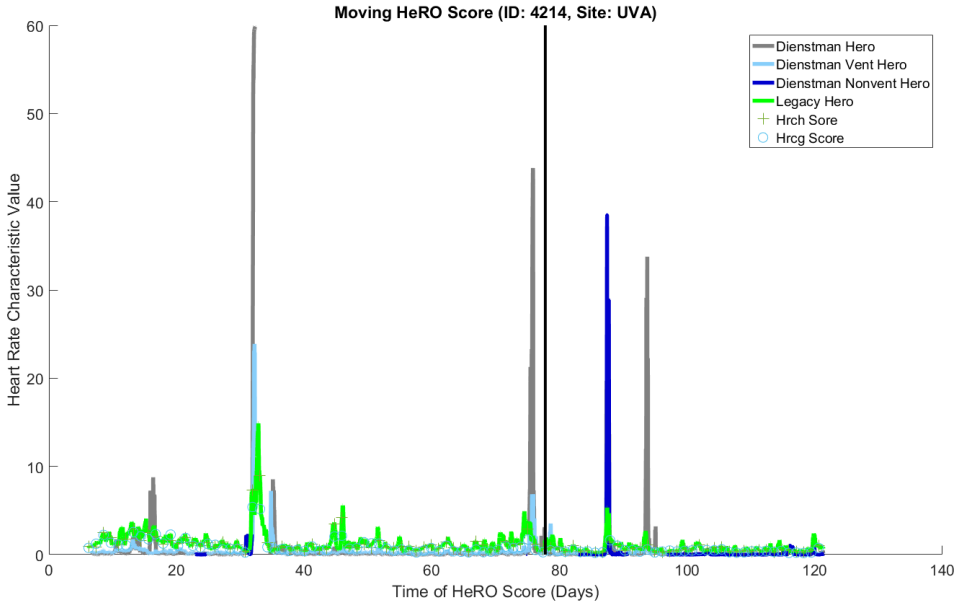

Moving HeRO Score (ID: 4214, Site: UVA)
Dienstman Hero
Dienstman Vent Hero
Dienstman Nonvent Hero
Legacy Hero
Hrch Sore
Hrcg Score
Heart Rate Characteristic Value
Time of HeRO Score (Days)
0
10
20
30
40
50
60
0
20
40
60
80
100
120
140

Figure A.22

Appendix B

Matlab Programs

B.1 Result Files

Listing B.1: `Dienstman_submit_paralle.txt`

```
#!/bin/tcsh

# Check for right number of arguments
if ($# != 2) then
  echo "Usage: Dienstman_submit_parallel begin end"
  exit 1
endif

if ($1 > $2) then
  echo "Error: Beginning value is larger than ending value."
  exit 1
endif

# Generate an array of jobs
qsub -N Dienstman_parallel -t $1-$2 <<EOF
#!/bin/tcsh
#PBS -l nodes=1:c9:ppn=1
#PBS -l walltime=180:00:00
#PBS -j oe
#PBS -q matlab
cd /sciclone/home00/eddienst/data10/Dienstman_Files
matlab -nojvm -nodisplay -r "multiple_result_files(\$PBS_ARRAYID)" >\$PBS_ARRAYID.out
EOF

exit
```

Listing B.2: `multiple_result_files.m`

```
function multiple_result_files(num)
% Author: Evan Dienstman
% Last Update: 2/24/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This program takes one batch number and computes the heart
% rate characteristics (HRCs) for each half hour of each ID number in
% that batch. Batches consits of 20 ID numbers. This architecture was
% created in order to run jobs in parallel most efficently that also
% comply with the rules for W&M's SciClone HPCs. If an ID does not
% exist, another function will print "Failed: ID" and continue to the
% next ID number. Half hours associated with one ID are saved in a file
% corresponding to the site and ID. For more info about what each file
% looks like, see the documentation for one_result_file.m. All files
% with the same site are saved in a folder corresponding to that site.
% If the folder has already been created, the patient files will
% automatically be saved there. The information used to calculate the
% HRCs comes from the HRC and storm files corresponding to the same
% site and ID.
%
% Arguments:
%     1. num - the batch number where the IDs in each batch are defined
%        within the code below
%
% Precondtions:
%     1. Make sure "preprocessing" is set correctly in this function.
%     2. Make sure the directories in this function are set correctly.
```

```
%     3. Make sure the file one_result_file.m is in the working
%        directory.
%
% Returns:
%     1. This function returns all files for existing IDs in the batch
%        saved to the appropriate folder.

% Change this with the propoer preprocessing and site.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% Here, we define which IDs go into which batch.
sites = {};
step = 20;

indices_0d = 2153:step:3287;
indices_0f = 2810:step:3981;
indices_17 = 2261:step:2512;
indices_18 = 2309:step:2486;
indices_1a = 2044:step:2637;
indices_1b = 2220:step:3078;
indices_1e = 2231:step:3315;
indices_UVA = 4102:step:7651;
start_indices = [indices_0d indices_0f indices_17 indices_18 indices_1a indices_1b indices_1e
    indices_UVA];

for j = 1:length(indices_0d)
    sites = [sites '0d']; %#ok<*AGROW>
end

for j = 1:length(indices_0f)
    sites = [sites '0f'];
end

for j = 1:length(indices_17)
    sites = [sites '17'];
end

for j = 1:length(indices_18)
    sites = [sites '18'];
end

for j = 1:length(indices_1a)
    sites = [sites '1a'];
end

for j = 1:length(indices_1b)
    sites = [sites '1b'];
end

for j = 1:length(indices_1e)
    sites = [sites '1e'];
end

for j = 1:length(indices_UVA)
    sites = [sites 'UVA'];
end

N = length(start_indices);

% After defining the batches, we call the function that will calculate
% the HRCs for each ID number in the batch.
if num <= N
    site = sites{num};
    start_index = start_indices(num);

    % Now we start looping through the IDs and call one_result_file()
    % to calculate the HRCs for each ID. The HRB and storm file
    % contain the information needed to calculate the HRCs.
    for id = start_index:start_index+step-1
        hrc_directory = [pwd '/Data_Files/Dienstman_Results_' preprocessing '_PP/' site];

        if ~exist(hrc_directory, 'dir')
            mkdir(hrc_directory);
        end

        if strcmp(site, 'UVA')
            hrb_file = [pwd '/Data_Files/HRB_Files/' site '//UVA_id' num2str(id) '_vch1.hrb'];
        else
            hrb_file = [pwd '/Data_Files/HRB_Files/' site '//_id' num2str(id) '_vch1.hrb'];
        end

        storm_file = [pwd '/Data_Files/Coleman_Results/' site '//storm_results_' site '_id'
            num2str(id) '.mat'];
        save_file = [hrc_directory '//Dienstman_hrc_results_' site '_' num2str(id) '.mat'];
        one_result_file(hrb_file, storm_file, save_file, id, site, preprocessing)
    end
end


end
```

Listing B.3: `one_result_file.m`

```
function one_result_file(hrb_file, storm_file, save_file, id, site, preprocessing)
% Author: Evan Dienstman
% Last Update: 2/24/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function computes the heart rate characteristics (HRCs) for each
% half hour of one patient. Patients are indentified by their site and
% ID number. The half hours are saved in one MATLAB file. Each file is
% a struct where each field correspond to one HRC. Each row contains
% the HRCs for one half hour. More information about the individual
% HRCs can be found throughout the code. Parts of this function were
% taken from Abigail Flower and Douglas Lake.
%
% Arguments:
%     1. hrb_file - the name of the HRB file which contains the RR
%        intervals used for calulating the HRCs. More info about the HRB
%        files and RR intervals can be found within the code.
%     2. storm_file - the name of the storm file which contains the
%        deceleration HRC. More info on the decelerations can be found
%        within the code.
%     3. save_file - the name of file the struct is saved to where the
%        name also contains the complete pathway to the file
%     4. id - the ID number of the patient
%     5. site - the string of the site for use in the saven file
%     6. preprocessing - a string specifying the type of processing the
%        raw data goes through. Descriptions of the processing can be
%        found below.
%
% Preconditions:
%     1. The complete pathway to the HRB, storm, and save file must be
%        included in respective input variables.
%     2. Make sure all the functions this program calls are in the same
%        directory.
%
% Returns:
%     1. This function returns a MATLAB file containing the HRCs for
%        one baby.

% We first check multiple exception cases before making the save file.
if ~exist(hrb_file, 'file');
    disp(['Failed: ', num2str(id), ' hrb file does not exist.' char(10)])

else
    % Here, we extract the RR intervals from the HRB file. An RR
    % interval is the time between each heart beat. RR interval time
    % is measure in miliseconds. All HRCs are calculated from the
    % RR intervals. For example, the variance is the variance of the
    % RR intervals.
    [rr,rrt,drop,info] = gethrb(hrb_file,inf,1);
    start_ind_pre = find(rr < 1000);

    if isempty(start_ind_pre)
        disp(['Failed: ', num2str(id), ' hrb file is empty.' char(10)])

    else

        if ~exist('rrt', 'var')
            disp(['Failed: ', num2str(id), ' hrb file has no time variable rrt.' char(10)])

        else
            % Here we create a blank struct where we will store the
            % HRCs. Note thathe variable max_num_intervals is ued to
            % preallocate the struct. If there are big jumps in the
            % times of the file, then the number of intervals
            % (half hours) containing data will be less than the
            % maximum number of intervals. The actual number of
            % intervals will only be equal to the max number of
            % intervals if the time is fairly continuos (no big jumps).
            % The way this function deals with jumps in the time is by
            % taking the start time and end time of the HRB file and
            % creating enough half hours to fill that entite period of
            % time. However, since there are usually times with no data,
            % there are usually rows in the struct left unfilled. This
            % way is a bit unnecessary but keeps the format coherant
            % with other files already created and provides an easy way
            % to preallocate the struct. It also provides a good measure
            % of how many jumps in the time there are by comparing
            % intervals with data to the total number of intervals in
            % the structure.
            start_ind = start_ind_pre(1);
            k = start_ind;
            end_file_time = rrt(end)*24*60;
            start_file_time = rrt(start_ind)*24*60;
            length_file = end_file_time-start_file_time;
            max_num_intervals = ceil(length_file/30);
            save_variable = ['Dienstman_hrc_results_' site '_' num2str(id)];

            % The purpose of these eval statements is so we can
            % personalize the name of the variable for each patient.
            % This way, we can pull up multiple variables from
            % different files and not get confused as to which variable
            % belongs to which patient.
```

```
95          eval([save_variable '(1:' num2str(max_num_intervals) ') = struct( ''Start_Time'', [], ' ...
96              ' ''Asymmetry_1'', [], ''Asymmetry_1_10'', [], ''Asymmetry_1_50'', [], ''Asymmetry_1_90'', [], ''Asymmetry_1_Slope'', [], ' ...
97              ' ''Asymmetry_2'', [], ''Asymmetry_2_10'', [], ''Asymmetry_2_50'', [], ''Asymmetry_2_90'', [], ''Asymmetry_2_Slope'', [], ' ...
98              ' ''Asymmetry_Ratio'', [], ''Asymmetry_Ratio_10'', [], ''Asymmetry_Ratio_50'', [], ''Asymmetry_Ratio_90'', [], ''Asymmetry_Ratio_Slope'', [], ' ...
99              ' ''Decelerations'', [], ''Decelerations_10'', [], ''Decelerations_50'', [], ''Decelerations_90'', [], ''Decelerations_Slope'', [], ' ...
100             ' ''Mean_RR'', [], ''Mean_RR_10'', [], ''Mean_RR_50'', [], ''Mean_RR_90'', [], ''Mean_RR_Slope'', [], ' ...
101             ' ''Sample_Entropy'', [], ''Sample_Entropy_10'', [], ''Sample_Entropy_50'', [], ''Sample_Entropy_90'', [], ''Sample_Entropy_Slope'', [], ' ...
102             ' ''Variance'', [], ''Variance_10'', [], ''Variance_50'', [], ''Variance_90'', [], ''Variance_Slope'', [], ' ...
103             ' ''Good_Frac'', [], ' ...
104             ' ''Extra_Info'', []);'])
105
106         % Now we loop through every half hour and calculate the
107         % HRCs for that half hour. We use k to index the RR
108         % interval vector and then find the index closest to
109         % one half hour from k. This gives us the RR intervals
110         % within the current half hour. We then use these RR
111         % intervals to calclulate the HRCs for the current half
112         % hour. Note that each half hour will have a different
113         % number of intervals for many reasons.
114         for halfhour = 1:max_num_intervals
115
116             if k < length(rrt)
117                 start_time = rrt(k);
118                 end_time = rrt(k)+(30*(1/60)*(1/24));
119
120                 % Here, there are two types of preprocessing (for
121                 % the RR intervals) to choose from. The the first
122                 % one (preprocess) was used by Abigail Flower and
123                 % the second one was used by Prof. Lake. Abigail's
124                 % replaces bad intervals with interpolated ones and
125                 % Lake's simply removes them and concatenates the
126                 % vector.
127                 if strcmp(preprocessing, 'Abby')
128                     end_ind_pre = find(abs(end_time-rrt) == min(abs(end_time-rrt)));
129                     end_ind = end_ind_pre(1);
130                     half_hour_indices = k:min(end_ind,length(rrt));
131
132                 else
133                     half_hour_indices = find(rrt > start_time & rrt <= end_time);
134                     end_ind = half_hour_indices(end);
135                 end
136
137                 raw_rr_interval_times = rrt(half_hour_indices);
138                 raw_rr_intervals = rr(half_hour_indices);
139                 raw_drop_rr_intervals = drop(half_hour_indices);
140                 good_frac = 1 - sum(raw_drop_rr_intervals)/length(raw_rr_intervals);
141
142                 if strcmp(preprocessing, 'Abby')
143                     [processed_rr_intervals, ignore1, ignore2, processed_rr_interval_times] =
                            preprocess(raw_rr_intervals, raw_drop_rr_intervals); %#ok<*ASGLU>
144                     processed_rr_interval_times = processed_rr_interval_times ./
                            (1000*60*60*24) + start_time;
145
146                 else
147                     processed_rr_intervals = raw_rr_intervals(raw_drop_rr_intervals == 0);
148                     processed_rr_interval_times = raw_rr_interval_times(raw_drop_rr_intervals
                            == 0);
149                 end
150
151                 extra_info = struct('info', info, 'raw_rr_intervals', raw_rr_intervals, '
                        raw_rr_interval_times', raw_rr_interval_times, 'processed_rr_intervals',
                        processed_rr_intervals, 'processed_rr_interval_times',
                        processed_rr_interval_times);
152
153                 % This is where the HRC's for good half hours are
154                 % created. Note note that we have already completed
155                 % all the preprocessing so the variable
156                 % processed_drop_rr_interval is all zero indicating
157                 % nothing will be dropped in the call to
158                 % calchrcx. For more info about each HRCs
159                 % calculated in calchrcx(), see the documentation
160                 % for calchrcx().
161                 if length(processed_rr_intervals)>300
162                     cflag = [1,1,1,1,0];
163                     filter = 1;
164                     processed_drop_rr_intervals = zeros(1,length(processed_rr_intervals));
165                     hrc_values = calchrcx(processed_rr_intervals, processed_drop_rr_intervals,
                            cflag, filter);
166
167                     % Here we extract the HRCs calculated in
168                     % calchrxc(). Note that varaince and sample
169                     % asymmetry measurements are recorded on a
170                     % natural log scale.
171                     variance = num_check(hrc_values(1));
172                     sampen = num_check(hrc_values(2));
173                     asym1 = num_check(hrc_values(3));
```

```
                    asym2 = num_check(hrc_values(4));
                    asym_ratio = asym2/asym1;
                    mean_rr = num_check(mean(processed_rr_intervals));

                    % Here we get the deceleration HRC from the
                    % storm file.
                    if exist(storm_file, 'file')
                        load_variable = load(storm_file);
                        storm_results = load_variable.storm_results;
                        height_list = storm_results(halfhour).final_height;
                        decels = 0;

                        % The variable r1 is the sample asymmetry
                        % measurement for accelerations. In
                        % symbols, r1 = sum[(beat < mediean -
                        % median )^2] / total_beats.
                        r1 = exp(hrc_values(3));

                        for height = height_list.'
                            % We only include decelerations above a
                            % 5*r1 in height. In other words, we
                            % only include delerations that are 5
                            % times greater than the average
                            % acceleration.
                            if height > 5 * sqrt(r1)
                                decels = decels + 1;
                            end
                        end

                    else
                        decels = NaN;
                    end

                    % The next set of HRCs to calculate are the
                    % slopes and percentiles of the HRCs over a
                    % window in the past for the HRCS already
                    % calculated above. Note that we first need to
                    % enter the current HRCs into the struct so
                    % the current values are used when calculating
                    % the slopes and percentiles.
                    hrc_fields = {'Start_Time', 'Good_Frac', 'Extra_Info', 'Asymmetry_1', '
                        Asymmetry_2', 'Asymmetry_Ratio', 'Decelerations', 'Mean_RR', '
                        Sample_Entropy', 'Variance'};
                    hrc_vector = {start_time good_frac extra_info asym1 asym2 asym_ratio
                        decels mean_rr sampen variance};

                    for i = 1:length(hrc_fields)
                        if i == 3
                            eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} ' =
                                extra_info;'])
                        else
                            eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} ' = '
                                num2str(hrc_vector{i}) ';'])
                        end
                    end

                    eval(['time_vector = [' save_variable '(:).Start_Time];'])

                    % Now that we have entered the current HRCs
                    % into the struct, we can now calculate the
                    % slopes and percentiles of the HRCs. Once we
                    % calculate these additional measurements,
                    % we end by adding these measurements into the
                    % struct as yet another HRC.
                    for i = 4:length(hrc_fields)
                        eval([hrc_fields{i} '_Vector = [' save_variable '(:).' hrc_fields{i} '
                            ];'])
                        slope_end_time = start_time;
                        slope_start_time = slope_end_time - 2;
                        percentile_end_time = start_time;
                        percentile_start_time = percentile_end_time - 0.5;

                        eval(['slope_vector = ' hrc_fields{i} '_Vector(time_vector >=
                            slope_start_time & time_vector <= slope_end_time);'])
                        slope_time_vector = time_vector(time_vector >= slope_start_time &
                            time_vector <= slope_end_time);
                        eval(['percentile_vector = ' hrc_fields{i} '_Vector(time_vector >=
                            percentile_start_time & time_vector <= percentile_end_time);'])
                        percentile_time_vector = time_vector(time_vector >=
                            percentile_start_time & time_vector <= percentile_end_time); %#ok
                            <*NASGU>

                        if ~isempty(slope_vector) && ~isempty(percentile_vector)
                            slope = slope_calculator(slope_time_vector.', slope_vector.');
                            percentile10 = num_check(prctile1(percentile_vector,10));
                            percentile50 = num_check(prctile1(percentile_vector,50));
                            percentile90 = num_check(prctile1(percentile_vector,90));

                        else
                            slope = NaN;
                            percentile10 = NaN;
                            percentile50 = NaN;
                            percentile90 = NaN;
                        end
```

```
                                eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} '_Slope =
                                    ' num2str(slope) ';'])
                                eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} '_10 ='
                                    num2str(percentile10) ';'])
                                eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} '_50 ='
                                    num2str(percentile50) ';'])
                                eval([save_variable '(' num2str(halfhour) ').' hrc_fields{i} '_90 ='
                                    num2str(percentile90) ';'])
                            end

                        else
                            % This is where blank values are created for
                            % bad half hours. Note NaN indicates bad
                            % HRC's as opposed to [] which indicates empty
                            % HRC slots created from preallocating the
                            % struct that we never ended up using.
                            eval([save_variable '(' num2str(halfhour) ') = struct( ''Start_Time'',
                                start_time, ' ...
                                ' ''Asymmetry_1'', NaN, ''Asymmetry_1_10'', NaN, ''Asymmetry_1_50'',
                                    NaN, ''Asymmetry_1_90'', NaN, ''Asymmetry_1_Slope'', NaN, ' ...
                                ' ''Asymmetry_2'', NaN, ''Asymmetry_2_10'', NaN, ''Asymmetry_2_50'',
                                    NaN, ''Asymmetry_2_90'', NaN, ''Asymmetry_2_Slope'', NaN, ' ...
                                ' ''Asymmetry_Ratio'', NaN, ''Asymmetry_Ratio_10'', NaN, ''
                                    Asymmetry_Ratio_50'', NaN, ''Asymmetry_Ratio_90'', NaN, ''
                                    Asymmetry_Ratio_Slope'', NaN, ' ...
                                ' ''Decelerations'', NaN, ''Decelerations_10'', NaN, ''
                                    Decelerations_50'', NaN, ''Decelerations_90'', NaN, ''
                                    Decelerations_Slope'', NaN, ' ...
                                ' ''Mean_RR'', NaN, ''Mean_RR_10'', NaN, ''Mean_RR_50'', NaN, ''
                                    Mean_RR_90'', NaN, ''Mean_RR_Slope'', NaN, ' ...
                                ' ''Sample_Entropy'', NaN, ''Sample_Entropy_10'', NaN, ''
                                    Sample_Entropy_50'', NaN, ''Sample_Entropy_90'', NaN, ''
                                    Sample_Entropy_Slope'', NaN, ' ...
                                ' ''Variance'', NaN, ''Variance_10'', NaN, ''Variance_50'', NaN, ''
                                    Variance_90'', NaN, ''Variance_Slope'', NaN, ' ...
                                ' ''Good_Frac'', NaN, ' ...
                                ' ''Extra_Info'', extra_info);'])
                        end
                    end

                    k = end_ind + 1;
                end

                % Finally, we save the struct using the site and id in
                % the file name.
                eval(['save ' save_file ' ' save_variable ';'])
            end
        end
    end
end
end




function xnew = num_check(x)
% This function turns any HRC that is negative, -inf, or inf to NaN
% because an HRC should never be any of these values. Occurances like
% those may happen due to bad data, so we simply mark them as NaN.

xnew = x;

if isempty(xnew)
    xnew = NaN;
elseif xnew < 0 || xnew == inf || xnew == -inf
    xnew = NaN;
end

end




function slope = slope_calculator(x,y)
% This function calculates the slope for any HRC. Our method of
% calculating the slope is to find a linear fit for the HRC data over
% a window in the past and then map the slope of that linear fit as
% the HRC slope for the current half hour.

if length(x) < 48
    slope = NaN;

else
    X = [ones(length(x),1) x];
    coeff = X\y;
    slope = coeff(2);
end

end
```

B.2 CSV Files

Listing B.4: `Dienstman_submit_batch.txt`

```
#!/bin/tcsh
#PBS -l nodes=1:c16:ppn=1
#PBS -l walltime=48:00:00
#PBS -j oe
#PBS -N Dienstman_batch
#PBS -q matlab
cd /sciclone/home2/eddienst/data10/Dienstman_Files
module load matlab/R2016b
matlab -nodisplay <multiple_event_figures_caller.m >output_file.out
```

Listing B.5: `csv_files.m`

```
% Author: Evan Dienstman
% Last Update: 3/3/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script takes all the information from the Dienstman_result
% files and organizes them into csv files. Each row in the csv file
% corresponds to one half hour from one baby. For the information
% that goes into each column, see the header variable in the script
% below. The script first creates a CSV file containg all the sick and
% healthy half hours from every baby. Note that a healthy half hour is
% any half hours that occurs 7 days before an event or 3 days after an
% event, and a sick half hour is any half hour that occurs within 12
% hours before an event. Additional CSV files are then created for
% specific groups. For example, one CSV file may contain only
% nonventilated sick half hours from oragnism 1. The CSV files are
% then saved to the appropriate directory. Meta info about the CSV
% files are also saved at the end.
%
% Preconditions:
%     1. Make sure the Dienstman_result files are in the approriate
%        directory.
%     2. Makes sure event_matrix.m, demographic_matrix.m,
%        vent_matrix.m, csv_master_file.m, csv_splitter.m,
%        csv_avg_hrcs.m, csv_bin_widths.m, csv_indices.m, and
%        csv_logistic_coeffs.m are in the current working directory.
%
% Returns:
%     1. This script creates many CSV files correspondsing to
%        different groups of half hours. CSV files are saved to the
%        Data_Files/Dienstman_CSV_Files directory.
%     2. This script also saves meta info about the CSV files
%        (averages, bin widths, indices, and logicitic coefficients)
%        to the current working directory.

clear
clc

% Change this with the propoer preprocessing and site.
preprocessing = 'Abby';
%preprocessing = 'Doug';

% Here, we create the header for the CSV file. You can use the header
% to see what info goes into the CSV file.
header = {'Site' 'ID' 'Half_Hour_Time' 'Event_Time' 'Gender' 'Gestational_Age' 'Good_Frac' '
    Health_Status' 'Organism' 'Study' 'Ventilated' 'Vent_Switched' 'Vent_Copies' 'Weight'};
hrcs = {'Asymmetry_1' 'Asymmetry_2' 'Asymmetry_Ratio' 'Decelerations' 'Mean_RR' 'Sample_Entropy' '
    Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};

for i = 1:length(hrcs)
    for j = 1:length(hrc_types)
        header = [header, [hrcs{i} hrc_types{j}]]; %#ok<AGROW>
    end
end

% Now we create the directory where we will save the CSV file.
save_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];

if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% Here, we create the master CSV file containing all the sick and
% healthy half hours.
master_csv_matrix = csv_master_file(preprocessing, save_dir, header, hrcs, hrc_types);

% Below, we index the master matrix by different qualifiers and write
% the various matrices to CSV files. Thus, we will have many CSV
% files for different groups of half hours. Creating these smaller
% CSV files helps future scripts because we won't have to read in the
% master CSV file every time.
```

```
csv_splitter(header, save_dir, master_csv_matrix)

% Finally, we call some functions that create some meta info for the
% CSV files that we will use later in other scripts.
csv_avg_hrcs(preprocessing)
csv_bin_widths(preprocessing)
csv_indices(preprocessing)
csv_logistic_coeffs(preprocessing, 'vent')
csv_logistic_coeffs(preprocessing, 'nonvent')
csv_logistic_coeffs(preprocessing, 'all')
```

Listing B.6: `csv_splitter.m`

```
function csv_splitter(header, save_dir, master_csv_matrix)
% Author: Evan Dienstman
% Last Update: 3/24/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function function takes the master csv matrix which contains
% all the healthy and sick half hours and creates multiple CSV files
% from that matrix. Each CSV corresponds to one particular group.
% For example, one CSV file may contain only sick, unventilated,
% organism 3 half hours. Seperating half hours in mutliple files
% helps run future programs faster because it takes a very long
% time each time we have to load the master CSV file.
%
% Arguments:
%     1. header - the header of the CSV files
%     2. save_dir - the directory to save all the SCV files
%     3. master_csv_matrix - the matrix containing all the healthy
%        and sick half hours that we want to split into many CSV files
%
% Precondtions:
%     1. Make sure the variable master_csv_matrix is a matrix and not
%        a CSV file.
%
% Returns:
%     1. This function saves many CSV files where each file contains
%        half hours for one specific category. CSV files are saved to
%        the driectory specified by the variable save_dir.

% First, we define some variables.
health_col = find(strcmp(header, 'Health_Status'));
org_col = find(strcmp(header, 'Organism'));
vent_col = find(strcmp(header, 'Ventilated'));
sick_strs = {'healthy', 'sick'};

% Now we will loop through every category we want to divided the
% master CSV matrix into. For every category, we index the appropraite
% half hours from the master CSV matrix and save those half hours as
% one CSV file.
for sick = 0:1
    sick_str = sick_strs{sick+1};
    file_name = [save_dir '//hrc_' sick_str '_org_all_vent_all.csv'];
    file_id = fopen(file_name, 'w') ;
    fprintf(file_id, '%s,', header{1,1:end-1}) ;
    fprintf(file_id, '%s\n', header{1,end}) ;
    fclose(file_id);

    indices = find(master_csv_matrix(:,health_col) == sick);
    one_csv_matrix  = master_csv_matrix(indices,:); %#ok<*FNDSB>
    dlmwrite(file_name, one_csv_matrix, '-append')

    for ventilated = 0:1
        file_name = [save_dir '//hrc_' sick_str '_org_all_vent_' num2str(ventilated) '.csv'];
        file_id = fopen(file_name, 'w') ;
        fprintf(file_id, '%s,', header{1,1:end-1}) ;
        fprintf(file_id, '%s\n', header{1,end}) ;
        fclose(file_id);

        indices = find(master_csv_matrix(:,health_col) == sick & master_csv_matrix(:,vent_col) ==
            ventilated);
        one_csv_matrix  = master_csv_matrix(indices,:);
        dlmwrite(file_name, one_csv_matrix, '-append')

        for organism = 1:5
            file_name = [save_dir '//hrc_' sick_str '_org_' num2str(organism) '_vent_' num2str(
                ventilated) '.csv'];
            file_id = fopen(file_name, 'w') ;
            fprintf(file_id, '%s,', header{1,1:end-1}) ;
            fprintf(file_id, '%s\n', header{1,end}) ;
            fclose(file_id);

            indices = find(master_csv_matrix(:,health_col) == sick & master_csv_matrix(:,org_col)
                == organism & master_csv_matrix(:,vent_col) == ventilated);
            one_csv_matrix  = master_csv_matrix(indices,:);
            dlmwrite(file_name, one_csv_matrix, '-append')
        end
    end
end
```

```
end
```

Listing B.7: csv_avg_hrcs.m

```
function csv_avg_hrcs(preprocessing)
% Author: Evan Dienstman
% Last Update: 3/3/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function takes the average of each column in the master CSV file
% hrc_all_org_all_vent_all.csv and saves the info to the current
% working directory. Becuase the master CSV file is relatively large,
% it is helpful to calculate the averages once and save the info
% instead of having to load the CSV file each time we need the average.
%
% Arguments:
%     1. preprocessing - the preprocessing method used when determining
%        which csv file to use
%
% Precondtions:
%     1. Make sure the file hrc_all_org_all_vent_all.csv is in the
%        approriate directory.
%
% Returns:
%     1. This function saves the column averages of the master CSV
%        file as a Matlab file called avg_hrc_values to the current
%        working directory.

csv_file = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP//hrc_all_org_all_vent_all.
    csv'];
raw_data = dlmread(csv_file,',',1,0);
avg_hrc_values = nanmean(raw_data); %#ok<*NASGU>
save([pwd '//avg_hrc_values_' preprocessing '_PP'], 'avg_hrc_values')
end
```

Listing B.8: csv_bin_widths.m

```
function csv_bin_widths(preprocessing)
% Author: Evan Dienstman
% Last Update: 3/24/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates the bin widths used for smoothing and
% integrating our probability density functions (PDFs) of each heart
% rate characteristic (HRC). We use the Freedman-Diaconis method to get
% a bin width for each HRC. The Freedman-Diaconis method is bin_width =
% 2*IQR*n^(-1/3), where IQR stands for inter-quartile range and n is
% the number of observations. When calculating our bin widths for each
% HRC, we use the file hrc_sick_org_3_vent_0.csv. We then use these bin
% widths to smooth or integrate a PDF regardless of what CSV file the
% PDF came from. Our thought process is that we want to be consistent
% with our choice of bin widths when making calculations amongst various
% groups. We use the file hrc_sick_org_3_vent_0.csv because it is our
% smallest sizeable category. However, if the size of the groups are
% relatively close, it might sometimes be better to use a bin widths
% calculated from the szie of the group we are looking at. In short,
% sometimes we will use these bin widths and sometimes we won't. The
% same bin widths can also be used for bivariate PDFs. Originally, we
% thought we needed to change the IQR (which goes from the 25th
% percentile to the 75th percentile) to the 15th percentile and
% 85th percentile for bivariate PDFs. Our thought process was that in
% two-dimensions, we are smoothing and intergrating over a box and we
% need to keep the area of that box proportional to the entire area.
% However, the Matlab function ksdensity already handles this issue if
% you simply pass the univariate bin widths as arguments.
%
% Arguments:
%     1. preprocessing - the preprocessing method used when determining
%        which csv file to use
%
% Preconditions:
%     1. Make sure the directories and file names used in the scripts
%        are the right ones for the computer you are using.
%     2. Make sure the csv file for sick_her_org_3_vent_0 exists.
%
% Returns:
%     1. This function creartes a file called bin_widths.mat that saves
%        all the HRC bin widths. Note that the width for decels will be
%        1 and is not saved in the file.

% Makes sure the name of the csv file used to make the bin widths
% matched the csv file you'll use on your computer.
load('hrc_indices')
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
```

```
csv_file = [csv_dir '/hrc_sick_org_3_vent_0.csv'];
csv_matrix = dlmread(csv_file, ',', 1, 0);

% Here, we determine how many HRCs we have.
[~, N] = size(csv_matrix);
bin_widths = zeros(1,N);

% Next, we iterate through every HRC and make the bin width for that
% HRC. Note that some HRCs are category info (e.g. id number) and
% will never be used for integrating or smoothing.
for i = 1:N

    % All deceleration widths are 1.
    if i >= decelerations_index && i <= decelerations_90_index
        bin_widths(i) = 1;

    else
        % Here, we extract the HRC vector from the csv file.
        csv_vector = csv_matrix(:,i);

        % Using the Freedman-Diaconis method, we now calculate the bin
        % width using the vector of HRC values for the specific
        % HRC we are on.
        n = length(csv_vector);
        q1 = quantile(csv_vector, .25);
        q3 = quantile(csv_vector, .75);
        bin_IQR = q3 - q1;
        bin_widths(i) = 2*bin_IQR*(n^(-1/3));
    end
end

% Finally, we save the bin width vector.
save([pwd '/bin_widths_' preprocessing '_PP'], 'bin_widths')
end
```

Listing B.9: `csv_indices.m`

```
function csv_indices(preprocessing)
% Author: Evan Dienstman
% Last Update: 3/2/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function saves the column index of every HRC in the CSV file.
% Saving the column indices helps future functions because we only
% have to load the index variables instead of having to load a CSV file
% and search the header.
%
% Arguments:
%     1. preprocessing - the preprocessing method used when determining
%        which csv file to use
%
% Precondtions:
%     1. Make sure the variable csv_file is the appopriate directory
%        for the computer.
%
% Returns:
%     1. This function saves the column index of every HRC in a file
%        called hrc_indices.mat. Every index has a unique variable
%        name taken from the header of the CSV files.

% First, we load a CSV file that we can use to look up the column
% indices.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
csv_file = [csv_dir '//hrc_all_org_all_vent_all.csv'];

[~, col_labels] = xlsread(csv_file, '1:1');

% Next, we loop through every column and save the index number with
% a unique variable name corresponding to the HRC of that column.
for i = 1:length(col_labels)
    save_variable = [lower(col_labels{i}) '_index'];
    eval([save_variable ' = i;'])

    if i == 1
        save('hrc_indices', save_variable)

    else
        save('hrc_indices', save_variable, '-append')
    end
end

end
```

Listing B.10: `csv_logistic_coeffs.m`

```
function csv_logistic_coeffs(preprocessing, type_str)
% Author: Evan Dienstman
```

```
% Last Update: 4/12/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates the coefficients used in the logistic HeRO
% score model. We have three sets of coefficients: all coeffs, vent
% coeffs, and nonvent coeffs. Since we use a quadratic model, we would
% have 666 coefficients from our 35 HRCs. However, we have hand picked
% a subset of the HRCs for the HeRO score.
%
% Arguments:
%     1. preprocessing - the preprocessing method used when determining
%        which csv file to use
%     2. type_str - a string indicating if we are calculating the
%        coefficients for vent, nonvent, or all half hours
%
% Precondtions:
%     1. Make sure the file hrc_indices.mat is in the current working
%        directory.
%
% Returns:
%     1. This functions saves all the coefficients in the file
%        Dienstman_coeffs.mat. The file contains three vectors for the
%        three sets of coefficents, three u0 values, and three
%        coefficient name cells.

% First, we define some constants and variables used later. Note we are
% only picking 12 HRCs.
load('hrc_indices.mat')
variables = [asymmetry_1_10_index asymmetry_1_slope_index...
    asymmetry_2_90_index asymmetry_2_slope_index...
    asymmetry_ratio_50_index asymmetry_ratio_slope_index...
    decelerations_90_index...
    sample_entropy_10_index sample_entropy_slope_index...
    variance_10_index variance_90_index variance_slope_index];
model = 'quadratic';

% Here, load the appropriate CSV file.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
csv_file = [csv_dir '//hrc_all_org_all_vent_all.csv'];
csv_matrix = dlmread(csv_file, ',', 1, 0);

% Here, we load the header and define some variables. We use the
% smallest file since all the headers are the same.
[~, variable_names] = xlsread([csv_dir '//hrc_sick_org_5_vent_0.csv'], '1:1');

% Now we create the coefficients for nonventilated, ventilated, or all
% half hours.
if strcmp(type_str, 'vent')
    csv_matrix = csv_matrix(csv_matrix(:,ventilated_index) == 1,:);
elseif strcmp(type_str, 'nonvent')
    csv_matrix = csv_matrix(csv_matrix(:,ventilated_index) == 0,:);
end

% Here, we calculate some variables we will use later. The order of
% these lines are very important because I reuse variable names. I
% do this to save memory since the matrices are very large.
N = length(variables);
sick_half_hours = csv_matrix(csv_matrix(:,health_status_index) == 1,:);
[num_sick, ~] = size(sick_half_hours);
[num_all, ~] = size(csv_matrix);
u0 = num_sick/num_all;
healthy_half_hours = csv_matrix(csv_matrix(:,health_status_index) == 0,:);
csv_matrix = [sick_half_hours; healthy_half_hours];
response_vector = csv_matrix(:,health_status_index);
csv_matrix = csv_matrix(:,variables);

% Before we calculate the probabilities, we remove outliers from the
% data. For the Bayesian method, removing outliers will not affect
% the results becuase outliers will have very low probabilities.
% However, we want to remove outliers for the logistic probability
% becuase we don't want to over fit the data at the tails. For
% decelerations, we define the high_fence as 30 becuase the outlier
% method removes too much data. This procedure is strictly empircal
% and needs to be analyzied further.
for j = 1:N
    variable = variables(j);
    temp_data = csv_matrix(:,j);
    q1 = quantile(temp_data, .25);
    q3 = quantile(temp_data, .75);
    IQR = q3 - q1;
    low_fence = q1 - 1.5*IQR;
    high_fence = q3 + 1.5*IQR;

    if variable >= decelerations_index && variable <= decelerations_90_index
        temp_data(temp_data < low_fence | temp_data > 30) = NaN;
        response_vector(temp_data < low_fence | temp_data > 30) = NaN;
    else
        temp_data(temp_data < low_fence | temp_data > high_fence) = NaN;
        response_vector(temp_data < low_fence | temp_data > high_fence) = NaN;
    end

    csv_matrix(:,j) = temp_data;
```

```
end

% Now we calculate the probability of illness using logistic
% regression. We use the Matlab function fitglm to calculate the
% probability. For more information, please see the Matlab
% documentation of this function.
fit = fitglm(csv_matrix, response_vector, model, 'distribution', 'binomial', 'VarNames', [
    variable_names(variables), 'Health_Status']);
coeffs = fit.Coefficients.Estimate;
coeff_names = fit.CoefficientNames;

% Lastly, we save all the variables we need.
if strcmp(type_str, 'vent')
    vent_coeffs = coeffs; %#ok<*NASGU>
    vent_u0 = u0;
    vent_coeff_names = coeff_names;
    save('Dienstman_coeffs_vent.mat', 'vent_coeffs', 'vent_u0', 'vent_coeff_names')
elseif strcmp(type_str, 'nonvent')
    nonvent_coeffs = coeffs;
    nonvent_u0 = u0;
    nonvent_coeff_names = coeff_names;
    save('Dienstman_coeffs_nonvent.mat', 'nonvent_coeffs', 'nonvent_u0', 'nonvent_coeff_names')
elseif strcmp(type_str, 'all')
    all_coeffs = coeffs;
    all_u0 = u0;
    all_coeff_names = coeff_names;
    save('Dienstman_coeffs_all.mat', 'all_coeffs', 'all_u0', 'all_coeff_names')
end
end
```

B.3 Time Series Figures

Listing B.11: `multiple_event_fiugres_caller.m`

```
% Author: Evan Dienstman
% Date: 3/30/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script calls the function multiple_event_figures for various
% combinations of input arguments. For each call to the function, the
% function creates a figure for the average HRC at each half hour seven
% days before an event and three days after. The figure also contains
% the average of the moving slope as well as the average 10th, 50th,
% and 90th percentiles of the HRCs. If the argument plot_str is 'yes',
% the function will also produce the same figure for each individual
% event. Only events that match the arguments target_org are used in
% the average. See the documentation of multiple_event_figures
% for more detail.
%
% Precondtions:
%     1. Make sure the Dienstman files and storm files are
%        all in their proper directories.
%     2. Make sure the files avg_hrc_values.m, event_matrix.mat,
%        vent_matrix.m, one_event_hrc.m, hrc_indices.m,
%        one_event_plot.m, and multiple_event_figures.m are in the
%        working directory.
%
% Returns:
%     1. The script returns a figure plotting the average of each HRC
%        for each half hour. The half hours plotted are ones 7 days
%        before an event and 3 days after.
%     2. If plot_str is 'yes', the function will return the same figure
%        for each indiviudal event.

clear
clc

% Change the preprocessing to the one you want to use.
preprocessing = 'Abby';
%preprocessing = 'Doug';

% Now we cell multiple_event_figures for different combinations of
% paramters.
multiple_event_figures(1, 'yes', preprocessing)
multiple_event_figures(2, 'yes', preprocessing)
multiple_event_figures(3, 'yes', preprocessing)
multiple_event_figures(4, 'yes', preprocessing)
multiple_event_figures(5, 'yes', preprocessing)
multiple_event_figures('all', 'yes', preprocessing)
```

Listing B.12: `multiple_event_fiugres.m`

```
function multiple_event_figures(target_org, plot_str, preprocessing)
% Author: Evan Dienstman
% Date: 3/30/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates a figure of the average HRC at each half hour
% seven days before an event and three days after.  The figure also
% contains the average of the moving slope as well as the average 10th,
% 50th, and 90th percentiles of the HRCs. If plot_str is 'yes', the
% function will also produce the same figure for each individual event.
% Only events that match the target_org are used in the average.
%
% Arguments:
%     1. target_org - the organism number the user wants to find the
%        average of (can be 'ALL' for all orgnaisms)
%     2. plot_str - a string indicating if the user want to plot the
%        indiviudal event plots
%     3. preprocessing - the preprocessing method used when determining
%        which result files to use
%
% Precondtions:
%     1. Make sure the Dienstman files and storm files are
%        all in their proper directories.
%     2. Make sure the files avg_hrc_vaules.m, event_matrix.mat,
%        vent_matrix.m, one_event_hrc.m, hrc_indices.m, and
%        one_event_plot.m are in the working directory.
%
% Returns:
%     1. The function returns a figure plotting the average of each HRC
%        for each half hour. The half hours plotted are ones 7 days
%        before an event and 3 days after.
%     2. If plot_str is 'yes', the function will return the same figure
%        for each indiviudal event as well.

% We first check that the save file for the average figure doesn't
% already exist.
avg_figure_directory = [pwd '/Figure_Files/Dienstman_Event_Figures_' preprocessing '_PP/Averages'
    ];

if ~exist(avg_figure_directory, 'dir')
    mkdir(avg_figure_directory)
end

avg_save_file_str = [avg_figure_directory '//Dienstman_figure_org_' num2str(target_org) '.fig'];

if exist(avg_save_file_str, 'file')
    disp('Error: A file already exists with the save file name. The program stopped because
        running the program would overwrite the existing file.')
    return
end

% Next, we load some files and define some variables.
load('event_matrix.mat');
load('vent_matrix.mat');
site_map_keys = {11, 13, 15, 23, 24, 26, 27, 30};
site_map_values = {'UVA', '0d', '0f', '17', '18', '1a', '1b', '1e'};
site_map = containers.Map(site_map_keys, site_map_values);
load(['avg_hrc_values_' preprocessing '_PP'])
field_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio',...
    'Decelerations', 'Mean_RR', 'Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};
vent_strs = {'vent', 'nonvent'};
reverse_vent_strs = {'nonvent', 'vent'}; %#ok<*NASGU>

% Next, we preallocate some empty structures which we will use to store
% the average HRCs.
avg_hrc_struct_vent(1:482) = struct('Ventilated', NaN);
avg_hrc_struct_nonvent(1:482) = struct('Ventilated', NaN);
count_hrc_struct_vent(1:482) = struct('Ventilated', NaN);
count_hrc_struct_nonvent(1:482) = struct('Ventilated', NaN);

for i = 1:length(field_names)

    for j = 1:length(vent_strs)

        for k = 1:length(hrc_types)
            eval([ '[avg_hrc_struct_' vent_strs{j} '(:).' field_names{i} hrc_types{k} '] = deal(
                NaN);' ])
            eval([ '[count_hrc_struct_' vent_strs{j} '(:).' field_names{i} hrc_types{k} '] = deal
                (0);' ])
        end
    end
end

% Next, we iterate through every event in the event_matrix file.
for i = 1:length(event_matrix)
    id = event_matrix(i,1);
    site_num = event_matrix(i,2);
    site = site_map(site_num);
    gest_age = event_matrix(i,4);
    event_time = event_matrix(i,7);
    total_age = floor(event_time/7) + gest_age;
```

```
    organism = event_matrix(i,6);
    ventilated = event_matrix(i,11);
    birth_weight = event_matrix(i,3);
    baby_info = struct('ID', id, 'Site', site, 'Event_Time', event_time, 'Gest_Age', gest_age, '
        Total_Age', total_age, 'Organism', organism, 'Ventilated', ventilated, 'Birth_Weight',
        birth_weight);

    vent_indices = find(vent_matrix(:,1) == site_num & vent_matrix(:,2) == id); %#ok<NODEF>
    baby_vent_info = vent_matrix(vent_indices,3:4); %#ok<*FNDSB>

    % Here, we continue only if the event matches our target organism.
    if strcmp(num2str(target_org), num2str(organism)) || strcmp(target_org, 'all')
        Dienstman_file = [pwd '/Data_Files/Dienstman_Results_' preprocessing '_PP/' site '//
            Dienstman_hrc_results_' site '_' num2str(id) '.mat'];

        % Here, we stop the entire function if a file is missing.
        if ~exist(Dienstman_file, 'file')
            disp(['Failed: ', site, ' ', num2str(id), ' files or directories do not exist.'])
            continue
        end

        % Otherwise, we extract the info for this event.
        one_hrc_struct = one_event_hrc(baby_info, baby_vent_info, Dienstman_file);

        % If plot_str is 'yes', we create a figure for this individual
        % event.
        if strcmp(plot_str, 'yes')
            one_event_figure = one_event_plot(avg_hrc_values, baby_info, one_hrc_struct);
            figure_directory = [pwd '/Figure_Files/Dienstman_Event_Figures_' preprocessing '_PP/'
                site];

            if ~exist(figure_directory, 'dir')
                mkdir(figure_directory)
            end

            save_file_str = [figure_directory '//Dienstman_figure_' site '_' num2str(id) '_'
                num2str(round(event_time)) '.fig'];
            hgsave(one_event_figure, save_file_str, '-v7.3')
        end

        % Here, we update the avg_structs. This is very dense code,
        % so I apologize that it's hard to read.
        one_nonvent_indices = find([one_hrc_struct(:).Ventilated] == 0);
        one_vent_indices = find([one_hrc_struct(:).Ventilated] == 1);

        for x = 1:length(field_names)

            for y = 1:length(vent_strs)

                for z = 1:length(hrc_types)
                    eval(['temp_vector = [one_hrc_struct(:).' field_names{x} hrc_types{z} '];'])
                    eval(['temp_vector(one_' reverse_vent_strs{y} '_indices) = NaN;'])
                    eval(['temp_vector = num2cell(nansum([[avg_hrc_struct_' vent_strs{y} '(:).'
                        field_names{x} hrc_types{z} ']; temp_vector]));'])
                    eval(['[avg_hrc_struct_' vent_strs{y} '(:).' field_names{x} hrc_types{z} '] =
                        deal(temp_vector{:});'])

                    eval(['temp_vector = +~isnan([one_hrc_struct(:).' field_names{x} hrc_types{z}
                        ']);'])
                    eval(['temp_vector(one_' reverse_vent_strs{y} '_indices) = NaN;'])
                    eval(['temp_vector = num2cell(nansum([[count_hrc_struct_' vent_strs{y} '(:).'
                        field_names{x} hrc_types{z} ']; temp_vector]));'])
                    eval(['[count_hrc_struct_' vent_strs{y} '(:).' field_names{x} hrc_types{z} ']
                        = deal(temp_vector{:});'])
                end
            end
        end
    end
end

% Here, we divide by the total number of half hours used for each
% index to get the average.
for i = 1:length(field_names)

    for j = 1:length(vent_strs)

        for k = 1:length(hrc_types)
            eval(['temp_avg_vector = num2cell([avg_hrc_struct_' vent_strs{j} '(:).' field_names{i}
                hrc_types{k} '] ./ [count_hrc_struct_' vent_strs{j} '(:).' field_names{i}
                hrc_types{k} ']);'])
            eval(['[avg_hrc_struct_' vent_strs{j} '(:).' field_names{i} hrc_types{k} '] = deal(
                temp_avg_vector{:});'])
        end
    end
end

% Lastly, we plot and save the average figure.
avg_event_figure = one_event_plot(avg_hrc_values, count_hrc_struct_vent, avg_hrc_struct_vent,
    count_hrc_struct_nonvent, avg_hrc_struct_nonvent, target_org);
hgsave(avg_event_figure, avg_save_file_str, '-v7.3')
end
```

Listing B.13: one_event_hrc.m

```
function hrc_fig_struct = one_event_hrc(baby_info, baby_vent_info, Dienstman_file)
% Author: Evan Dienstman
% Date: 3/30/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function takes a Dienstman_hrc_results file and find the
% HRCs in each half hour seven days before and three days after a
% septic event. This function also determines if each half hour is
% ventilated or not ventilated. All this information is then stored
% in a struct.
%
% Arguments:
%     1. baby_info - a struct containing the id, site, event time,
%        gestational age, total age, organism, and ventilation of the
%        patient
%     2. baby_vent_info - a matrix containg the start and end times
%        of each period the baby was ventilated
%     3. Dienstman_file - the Dienstman_hrc_results file containing the
%        HRCs for each half hour
%
% Returns:
%     1. hrc_fig_struct - a structure containing the HRCs and
%        ventilation status of each half hour seven days before the
%        event and three days after the event.

% First, we define some variables.
id = baby_info.ID;
site = baby_info.Site;
event_time = baby_info.Event_Time;
num_half_hours = 482;
field_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio',...
    'Decelerations', 'Mean_RR', 'Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};

% Next, we preallocate a struct where we wil store all the HRC info
% for seven days before the event and 3 days after the event.
hrc_fig_struct(1:num_half_hours) = struct('Ventilated', NaN);

for i = 1:length(field_names)
    for j = 1:length(hrc_types)
        eval([ '[hrc_fig_struct(:).' field_names{i} hrc_types{j} '] = deal(NaN);' ])
    end
end

% Here, we calculate the time window and load in the corresponding
% result file.
event_window = (event_time-7):(1/48):(event_time+2/48+3);
load_variable = load(Dienstman_file); %#ok<NASGU>
eval_string = ['hrc_results_struct = load_variable.Dienstman_hrc_results_' site '_' num2str(id) ';
    '];
eval(eval_string);

% We now loop through every half hour in the time window and record the
% HRCs for each half hour in a struct.
for i = 1:length(hrc_results_struct)
    hrc_entry = hrc_results_struct(i);

    % If the half hour falls within the time window, we look up the
    % corresoonding half hour index and record the HRCs associated
    % with that index.
    if ~isempty(hrc_entry.Start_Time) && event_window(1) < hrc_entry.Start_Time && hrc_entry.
        Start_Time < event_window(end)
        index = find(event_window <= hrc_entry.Start_Time, 1, 'last');

        for j = 1:length(field_names)
            for k = 1:length(hrc_types)
                variable = [field_names{j} hrc_types{k}];
                eval(['hrc_fig_struct(index).' variable ' = hrc_entry.' variable ';'])
            end
        end

        % Lastly, we find the ventilation status for the half hour.
        if ~isempty(find(hrc_entry.Start_Time > baby_vent_info(:,1) & hrc_entry.Start_Time <
            baby_vent_info(:,2), 1))
            hrc_fig_struct(index).Ventilated = 1;
        else
            hrc_fig_struct(index).Ventilated = 0;
        end
    end
end
end
```

Listing B.14: one_event_plot.m

```
function [baby_figure] = one_event_plot(avg_hrc_values, baby_info_1, hrc_struct_1, baby_info_2,
    hrc_struct_2, organism) %#ok<*INUSL>
% Author: Evan Dienstman
% Date: 3/30/2016
```

```
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates a nice plot of the HRCs seven days before and
% three days after a septic event for. The plot contains seven subplots,
% one for each HRC. Each subplot contains five curves, one for each HRC
% subcategory. If the figure is for a single event, the color of the
% curve changes from dark to light if the baby is ventilated or not
% ventilated, respectively. If the figure is for an average, each plot
% contains a light and dark curve for each HRC, giving a total of ten
% curves per subplot. Each subplot also contains a horizontal line
% representing the average HRC for all healthy and sick half hours from
% all babies for comparison. Lastly, each subplot contains a vertical
% line representing the time of the event. Note that the left y axis is
% used for the slope.
%
% Arguments:
%     1. avg_hrc_values - the average value of the seven HRCs for all
%        sick and healthy half hours
%     2. baby_info_1 - a structure containing the id, site, event time,
%        gestational age, total age, organism, and ventilation of the
%        patient (for one event) or a structure containing the
%        ventilated counts (for an average)
%     3. hrc_strcut_1 - a structure containing the HRCs seven days
%        before the event and three days after the event (for one event)
%        or a strucute containing the ventilated average HRCs (for an
%        average)
%     4. baby_info_2 -  a structure containing the non-ventilated
%        counts (for an average)
%     5. hrc_struct_2 - a strucute containing the non-ventilated
%        average HRCs (for an average)
%     6. organism - the number of the organism (for an average)
%
% Returns:
%     1. baby_figure - a Matlab figure containing the seven subplots
%        for each HRC seven days before and three days after a septic
%        event

% First, we load the hrc indices and define some variables.
load('hrc_indices.mat')
field_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio',...
    'Decelerations', 'Mean_RR', 'Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};
left_ylim_vector = {[0 10], [0 10], [0 10], [0 20], [250 550], [0 1.5], [0 5]};
right_ylim_vector = {[-1 1], [-1 1], [-1 1], [-1 1], [-20 20], [-1 1], [-1 1]};
nonvent_color_vector = {[135/255, 206/255, 250/255], [0/255, 250/255, 154/255],...
    [221/255, 160/255, 221/255], [240/255, 230/255, 140/255], [240/255, 128/255, 128/255]};
vent_color_vector = {[0, 0, 128/255], [0, 128/255, 0], [148/255, 0, 211/255],...
    [255/255, 140/255, 0], [128/255, 0, 0]};
window = 5; %#ok<*NASGU>
avg_type = 's';
index_vector = -336:145;

% Next, we get some information for the legend in the top left of the
% figure. For one event, we get info on the baby. For averages, we
% get info on the average number of half hours used.
if isfield(baby_info_1, 'ID')
    id = baby_info_1.ID;
    site = baby_info_1.Site;
    event_time = baby_info_1.Event_Time;
    gest_age = baby_info_1.Gest_Age;
    total_age = baby_info_1.Total_Age;
    organism = baby_info_1.Organism;
    ventilated = baby_info_1.Ventilated;
    vent = [hrc_struct_1(:).Ventilated];

else
    temp_vent_count_1 = 0;
    temp_vent_count_2 = 0;
    temp_nonvent_count_1 = 0;
    temp_nonvent_count_2 = 0;

    % This loop calculates the average number of half hours used when
    % calculating the average HRC for each HRC index relative to the event.
    % In short, we add together the half hours used for each index
    % across all HRCs and then divide by the total number of indices
    % across all HRRCs. This number just gives us a rough understanding
    % of how many half hours were used for the average so don't worry
    % if you don't completely undertand this part.
    for i = 1:length(field_names)

        for j = 1:length(hrc_types)
            variable = [field_names{i} hrc_types{j}];
            eval(['vent_half_hour_vector = [baby_info_1(:).' variable '];'])
            temp_vent_count_1 = temp_vent_count_1 + sum(vent_half_hour_vector);
            temp_vent_count_2 = temp_vent_count_2 + length(vent_half_hour_vector);

            eval(['nonvent_half_hour_vector = [baby_info_2(:).' variable '];'])
            temp_nonvent_count_1 = temp_nonvent_count_1 + sum(nonvent_half_hour_vector);
            temp_nonvent_count_2 = temp_nonvent_count_2 + length(nonvent_half_hour_vector);
        end
    end

```

```
    vent_mean_half_hours = temp_vent_count_1 / temp_vent_count_2;
    nonvent_mean_half_hours = temp_nonvent_count_1 / temp_nonvent_count_2;
end

% We now start creating the figure.
baby_figure = figure('Position', [50,50,1600,900]);
set(baby_figure,'color','w');

% Here, we loop through all the subplots. Each subplot corresponds to
% one HRC and contains fives curves for each of the HRC subcategories.
% For one event, the curves will change from dark to light to
% represent ventilated and nonventilated, respectively. For an
% average, we plot ventilated and nonventilated separately for a
% total of 10 curves per subplot.
for i = 1:length(field_names)
    subplot(3,3,i+1)
    hold on

    for j = 1:length(hrc_types)
        variable = [field_names{i} hrc_types{j}];

        % We extract the vectors for plotting here. Note that we
        % smooth the vectors for plotting purposes only.
        if isfield(baby_info_1, 'ID')
            eval(['vent_plot_vector = tsmovavg([hrc_struct_1(:).' variable '], avg_type, window);'])
            eval(['nonvent_plot_vector = tsmovavg([hrc_struct_1(:).' variable '], avg_type, window);'])
            vent_plot_vector(vent == 0) = NaN; %#ok<*AGROW>
            nonvent_plot_vector(vent == 1) = NaN;

        else
            eval(['vent_plot_vector = tsmovavg([hrc_struct_1(:).' variable '], avg_type, window);'])
            eval(['nonvent_plot_vector = tsmovavg([hrc_struct_2(:).' variable '], avg_type, window);'])
        end

        % If we are ploting the slope subcategory, we use a different
        % y axis.
        if j ~= 5
            yyaxis('left')
            plot(index_vector, vent_plot_vector, '-', 'LineWidth', 2, 'Color', vent_color_vector{j})
            plot(index_vector, nonvent_plot_vector, '-', 'LineWidth', 2, 'Color', nonvent_color_vector{j})

        % Else, we plot all the other subcategories on the right y axis.
        else
            yyaxis('right')
            plot(index_vector, vent_plot_vector, '-', 'LineWidth', 2, 'Color', vent_color_vector{j})
            plot(index_vector, nonvent_plot_vector, '-', 'LineWidth', 2, 'Color', nonvent_color_vector{j})
            ylim(right_ylim_vector{i})
        end
    end

    % Finally, we add more info to the subplots including a horizontal
    % line for the average HRC across all half hours from all babies and
    % a vertical line for the time of the event.
    yyaxis('left')
    ylim(left_ylim_vector{i})
    hrc_index = lower([field_names{i} '_index']);
    eval(['plot(index_vector, ones(1,length(index_vector))*avg_hrc_values(' hrc_index '), ''-'', ''LineWidth'', 2, ''Color'', [0, 0, 0])'])
    line([0 0], left_ylim_vector{i}, 'Color', [0,0,0], 'LineWidth', 2);
    title(regexprep(field_names{i}, '_', ' '), 'FontSize', 13)
    xlim([-336 145])
    set(gca, 'FontSize', 14)
    hold off
end

% Finally, we create the legend at the top left of the figure.
title_frame = uicontrol('style', 'frame');
set(title_frame, 'Position', [208, 635, 320, 195], 'BackgroundColor', [0 0 0])

if isfield(baby_info_1, 'ID')
    round_time = round(event_time);
    baby_title_string = ['         Patient ID Number: ' num2str(id) char(10)...
        '              Site Code: ' site char(10)...
        '              Days of Age: ' num2str(round_time) char(10)...
        '       Gestational Age (Weeks): ' num2str(gest_age) char(10)...
        '           Total Age (Weeks): ' num2str(total_age) char(10)...
        '            Organism Number: ' num2str(organism) char(10)...
        '         Ventilation at Event: ' num2str(ventilated) char(10) char(10)...
        'Blue: Raw, Green: 10th, Purple: 50th,' char(10)...
        '       Orange: 90th, Red: Slope' char(10) char(10)...
        ' Dark: Veniltaed, Ligth: Nonventilaed'];

else
    baby_title_string = ['            Organism Number: ' num2str(organism) char(10) char(10) char(10)...
        ' Mean Num of Vent Half Hours Used: ' num2str(vent_mean_half_hours) char(10)...
```

```
        'Mean Num of Nonvent Half Hours Used: ' num2str(nonvent_mean_half_hours) char(10) char(10)...
        '    Blue: Raw, Green: 10th, Purple: 50th,' char(10)...
        '         Orange: 90th, Red: Slope' char(10) char(10)...
        '     Dark: Veniltaed, Ligth: Nonventilaed'];
end

baby_title = uicontrol('style', 'text');
set(baby_title, 'String', baby_title_string, 'Position', [211, 638, 314, 189], 'FontSize', 9, 'BackgroundColor', [1 1 1])

axes_note_string = 'All x-axes show the half hour index. Thus, there are 482 half hours representing the 10 day window.';
axes_note = uicontrol('style', 'text');
set(axes_note, 'String', axes_note_string, 'Position', [520,-20,600, 50], 'FontSize', 10, 'BackgroundColor', [1 1 1])
end
```

B.4 Univariate PDF Figures

Listing B.15: `multiple_univariate_pdf_figures.m`

```
% Author: Evan Dienstman
% Last Update: 3/23/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script creates figures for the univariate PDFs of HRCs from
% various categories. For more information about the figures, please
% see the documentation for one_univariate_pdf_figure.m.
%
% Preconditions:
%     1. Make sure the files in file_list have the correct name.
%     2. Make sure the color_list, color_name_list, and file_list
%        used in one call to one_univariate_pdf_figure have the same
%        length.
%     3. Make sure the file one_univariate_pdf_figure.m is in the
%        current working directory.
%
% Returns:
%     1. This script will create five figures for each call to
%        one_bivariate_pdf_figure. For info about where the files are
%        saved, see the documentation for one_bivariate_pdf_figure.

clear
clc

% Change the preprocessing to the one you want to use.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% First, we define the colors of the PDFs in our figure.
color_list_1 = {[0, 0, 128/255], [135/255, 206/255, 250/255], [128/255, 0, 0], [240/255, 128/255, 128/255]};
color_list_2 = {[0, 0, 1], [1, 0, 0], [0, 100/255, 0], [128/255, 0, 128/255], [255/255, 165/255, 0], [0, 0, 0]};
color_list_3 = {[0, 0, 128/255], [128/255, 0, 0]};

color_name_list_1 = {'Dark Blue', 'Light Blue', 'Dark Red', 'Light Red'};
color_name_list_2 = {'    Blue', '    Red', '    Green', '    Purple', '    Orange', '    Black'};
color_name_list_3 = {'Dark Blue', 'Dark Red'};

% Next, we create lists of CSV files that we want to plot to the
% same figure.
file_list_1 = {'hrc_healthy_org_1_vent_0.csv', 'hrc_healthy_org_1_vent_1.csv', 'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_1_vent_1.csv'};
file_list_2 = {'hrc_healthy_org_2_vent_0.csv', 'hrc_healthy_org_2_vent_1.csv', 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_2_vent_1.csv'};
file_list_3 = {'hrc_healthy_org_3_vent_0.csv', 'hrc_healthy_org_3_vent_1.csv', 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_3_vent_1.csv'};
file_list_4 = {'hrc_healthy_org_4_vent_0.csv', 'hrc_healthy_org_4_vent_1.csv', 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_4_vent_1.csv'};
file_list_5 = {'hrc_healthy_org_5_vent_0.csv', 'hrc_healthy_org_5_vent_1.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_sick_org_5_vent_1.csv'};
file_list_6 = {'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_all_org_all_vent_all.csv'};
file_list_7 = {'hrc_healthy_org_all_vent_0.csv', 'hrc_sick_org_all_vent_0.csv'};
file_list_8 = {'hrc_healthy_org_all_vent_1.csv', 'hrc_sick_org_all_vent_1.csv'};
file_list_9 = {'hrc_healthy_org_all_vent_all.csv', 'hrc_sick_org_all_vent_all.csv'};

% Finally, we call one_univariate_pdf_figure for each file_list.
% Every call to one_univatiate_pdf_figure will create five figures
% (one for each HRC subcategory). Each of the five figures will
% contain PDFs from the files in file_list.
```

```
one_univariate_pdf_figure('hrc_all_org_1_vent_all', file_list_1, color_list_1, color_name_list_1,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_2_vent_all', file_list_2, color_list_1, color_name_list_1,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_3_vent_all', file_list_3, color_list_1, color_name_list_1,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_4_vent_all', file_list_4, color_list_1, color_name_list_1,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_5_vent_all', file_list_5, color_list_1, color_name_list_1,
    preprocessing)
one_univariate_pdf_figure('hrc_sick_org_all_vent_0', file_list_6, color_list_2, color_name_list_2,
     preprocessing)
one_univariate_pdf_figure('hrc_all_org_all_vent_0', file_list_7, color_list_3, color_name_list_3,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_all_vent_1', file_list_8, color_list_3, color_name_list_3,
    preprocessing)
one_univariate_pdf_figure('hrc_all_org_all_vent_all', file_list_9, color_list_3, color_name_list_3
    , preprocessing)
```

Listing B.16: one_univariate_pdf_figure.m

```
function one_univariate_pdf_figure(save_file_str, file_list, color_list, color_name_list,
    preprocessing)
% Author: Evan Dienstman
% Last Update: 3/23/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates univariate probability density functions (PDFs)
% for each heart rate characteristics (HRC) of each csv file given in
% the variable file_list. The PDFs are then all plotted onto one
% of five figures. Each of the five figures contains the PDFs for one
% HRC subtype (raw, 10th, 50th, 90th, and slope). One each figure,
% there are seven subplots that corresponds to the seven HRCs. Each
% subplot contains a PDF for every CSV in the variable file_list.
% There is also a text box at the top of the figrue that acts as a
% legend for each subplot. This text box also contains information
% on how many babies and half hours were used for each PDF. Note that
% for any sick categroy, we take any half hour within 24 half hours
% of an event. For any healthy cateogry, we take any half hour
% 7 days before an event or 3 days after event. If a baby never had an
% event, then all half hours are healthy. The PDFs were created using
% kernel density estimation. For more details about this method, see
% the comments in the code. Each row in the csv files corresponds to
% one half hour. Each column contains an HRC or info about the half
% hour. Different csv files contain half hours that fall into different
% cateogries from the informaiton in the category columns. If the user
% wants to graph the PDFs from differnt categories of half hours, they
% can do so by adding, removing, or changing the corresponding csv
% files in the varibale file_list.
%
% Arguments:
%     1. save_file_str - the name of the save files
%     2. file_list - the CSV files used to make the PFDs
%     3. color_list - the RGB colors of the PDFs
%     4. color_name_list - the color names of the PDFs
%     5. preprocessing - a string indicating which preprocessing method
%        to use
%
% Preconditions:
%     1. Make sure the directories and file names used in the scripts
%        are the right ones for the computer you are using.
%     2. Make sure file_list, color_list, and color_name_list contain
%        the same number of elements.
%     3. This script will not overwrite any existing files with the
%        same name. Change the variable save_file to a name that does
%        not alreadcy exist or delete the existing file with the same
%        name before running this script.
%
% Returns:
%     1. This function creates five figures saved with the name given
%        by the variable save_file and with the appropriate subcategory
%        string at the end. The figures are saved to the directory
%        given by the variable save_dir below.

% Change the directory names to match the directories on your
% computer.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
save_dir = [pwd '/Figure_Files/Dienstman_Univariate_PDFs_' preprocessing '_PP'];
save_file = [save_dir '//' save_file_str '.png'];

% If the save_dir doesn't already exist, we make the save_dir here.
if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% If the save file already exists, we stop the program so we don't
% overwrite the file.
if exist(save_file, 'file')
    disp('Error: A file already exists with the save file name. The program stopped because
        running the program would overwrite the existing file.')
```

```
    return
end

% Here, we define the edges used for plotting.
N = 1e3;
asym1_edges = linspace(0,10,N);
asym2_edges = linspace(0,10,N);
asym_ratio_edges = linspace(0,5,N);
decel_edges = linspace(0,50,N);
mean_rr_edges = linspace(250,550,N);
sampen_edges = linspace(0,1.25,N);
variance_edges = linspace(0,5,N);
edge_vector = {asym1_edges, asym2_edges, asym_ratio_edges, decel_edges, mean_rr_edges,
    sampen_edges, variance_edges};

asym1_edges_slope = linspace(-5,5,N);
asym2_edges_slope = linspace(-5,5,N);
asym_ratio_edges_slope = linspace(-1,1,N);
decel_edges_slope = linspace(-10,10,N);
mean_rr_edges_slope = linspace(-50,50,N);
sampen_edges_slope = linspace(-1,1,N);
variance_edges_slope = linspace(-2,2,N);
edge_vector_slope = {asym1_edges_slope, asym2_edges_slope, asym_ratio_edges_slope,
    decel_edges_slope, mean_rr_edges_slope, sampen_edges_slope, variance_edges_slope};

% These vectors are used to create the information in the boxes at
% the top of the figures.
baby_count_vector = zeros(1,length(file_list));
mean_half_hour_vector = zeros(1,length(file_list));

% These cells are used to create the HRC names.
hrc_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio', 'Decelerations', 'Mean_RR', '
    Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};

% Here, we load the header. We use the smallest file since all the
% headers are the same.
[~, header_names] = xlsread([csv_dir '//hrc_sick_org_5_vent_0.csv'], '1:1');

% Next, we loop through every file in file_list and make the PDFs for
% each HRC of that file.
for i = 1:length(file_list)
    csv_file = [csv_dir '//' file_list{i}];
    raw_data = dlmread(csv_file,',',1,0);

    % Next, we calculate the info that goes into the box at the top
    % of the figure for this csv file.
    site_index = find(strcmp(header_names, 'Site'));
    id_index = find(strcmp(header_names, 'ID'));
    unique_matrix = unique(raw_data(:,[site_index id_index]), 'rows');
    baby_count_vector(i) = length(unique_matrix);
    mean_half_hour_vector(i) = length(raw_data)/length(unique_matrix);

    % Now we can loop through every HRC and make the PDF for that
    % HRC with the half hours of the current CSV file.
    for j = 1:length(hrc_names)

        for k = 1:length(hrc_types)
            variable_name = [hrc_names{j} hrc_types{k}];

            if k == 5
                edge = edge_vector_slope{j}; %#ok<*NASGU>
            else
                edge = edge_vector{j};
            end

            % Next, we extrct the HRC information from the csv file.
            column_data = raw_data(:, strcmp(header_names, variable_name));

            % Here, we create the PDFs for each HRC of this file.
            % We create PDFs using the Matlab function ksdensity,
            % which uses the kernel density estimation method to
            % create the PDFs. For the kernel density estimation,
            % we use an Epanechnikov keernel. We also allow the
            % function to calculate the optimal bandwidth except for
            % decelerations where we use a width of 1. At first,
            % we used the width the Freeman-Diaconis method produced
            % from the organism 3 file so all the PDFs would have a
            % consistent width. However, we believe this isn't a
            % propblem if all our n's are relatively close. The
            % input for ksdensity is a vector of one spefic type of
            % HRC values corresponding to every half hour in the
            % csv file. Note that since we are creating PDFs, the
            % area underneath each curve is 1. Consequently, all
            % the PDFs are normalized and values of the PDF can be
            % greater than 1. For more information about the kernel
            % density estimation, Epanechnikov kernel function, and
            % PDFs, see their respective wikipedia pages.
            if ~strcmp(hrc_names{j}, 'Decelerations') || k == 5
                eval([variable_name '_Prob(:,i) = ksdensity(column_data, edge, ''kernel'', ''
                    epanechnikov'');'])
            else
                eval([variable_name '_Prob(:,i) = ksdensity(column_data, edge, ''kernel'', ''
                    epanechnikov'', ''width'', 1);'])
```

```
            end
        end
    end
end

% Here, we define some variables used for plotting.
x_axis_list = {[0 10], [0 10], [0 3], [0 10], [250 550], [0 1], [0 5]};
y_axis_list = {[0 0.75], [0 0.75], [0 5], [0 0.4], [0 0.02], [0 6], [0 1.5]};
x_axis_list_slope = {[-2 2], [-2 2], [-.75 .75], [-5 5], [-50 50], [-.5 .5], [-1 1]};
y_axis_list_slope = {[0 2], [0 2], [0 7.5], [0 1], [0 0.1], [0 15], [0 4]};
title_list = {'Raw HRC  ', '10th Percentile ', '50th Percentile', '90th Percentile', 'HRC Slope  '
    };

% Finally, we plot the PDFs. We will create five figures. Each figure
% contains the PDFs of one subcategory. The subcategories are raw HRC,
% 10th percentile, 50th percentile, 90th percentile, and slope. Each
% figure contains seven subplots that corresponds to one specific HRC.
% On each subplot, there is a PDF for each CSV file in the variable
% file_list.
for i = 1:length(hrc_types)
    probability_figure = figure('Position', [50,50,1600,900]);
    set(probability_figure,'color','w');

    for j = 1:length(hrc_names)
        subplot(3,3,j+1)
        hold on

        if  i == 5
            x = edge_vector_slope{j}.';
        else
            x = edge_vector{j}.';
        end

        variable_name = [hrc_names{j} hrc_types{i}];

        % Here, we plot the PDFs.
        for k = 1:length(file_list)
            eval(['y = ' variable_name '_Prob(:,k);'])
            plot(x, y, 'LineWidth', 2, 'Color', color_list{k})
        end

        if i == 5
            xlim(x_axis_list_slope{j})
            ylim(y_axis_list_slope{j})
        else
            xlim(x_axis_list{j})
            ylim(y_axis_list{j})
        end

        xlabel(regexprep(hrc_names{j},'_',' '), 'FontSize', 16)
        ylabel('Probability', 'FontSize', 16)
        set(gca, 'FontSize', 16)
        hold off
    end

    % Here, we plot the box at the top of the figure.
    if length(file_list) == 6
        baby_title_string = ['    Univariate Probability Densities: ' title_list{i}];
    else
        baby_title_string = ['   Univariate Probability Densities: ' title_list{i} char(10)];
    end

    for n = 1:length(file_list)
        file_str = file_list{n};
        baby_title_string = [baby_title_string char(10) '        ' color_name_list{n} ' = '
            file_str(1:end-4) char(10) '      Baby Count = ' num2str(baby_count_vector(n)) ', Mean
             Half Hours = ' num2str(mean_half_hour_vector(n), 4)]; %#ok<AGROW>
    end

    title_frame = uicontrol('style', 'frame');
    set(title_frame, 'Position', [208, 635, 345, 195], 'BackgroundColor', [0 0 0])
    baby_title = uicontrol('style', 'text');
    set(baby_title, 'String', baby_title_string, 'Position', [211, 638, 339, 189], 'FontSize', 8,
        'BackgroundColor', [1 1 1])

    % Lastly, we save the figure.
    print(probability_figure, [save_file(1:end-4) lower(hrc_types{i}) '.png'], '-dpng')
end
end
```

B.5 Univariate Risk Figures

Listing B.17: `multiple_univariate_risk_figures.m`

```
% Author: Evan Dienstman
% Last Update: 3/31/2017
% Email: eddienstman@email.wm.edu
```

```
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script creates figures for the univariate risks of HRCs from
% various categories. For more information about the figures, please
% see the documentation for one_univariate_risk_figure.m.
%
% Preconditions:
%     1. Make sure the files in file list have the correct names.
%     2. Make sure the file one_univariate_risk_figure.m is in the
%        current working directory.
%     3. Make sure to select the appropriate preprocessing below.
%
% Returns:
%     1. This script will create one figure for each call to
%        one_bivariate_pdf_figure. For info about where the files are
%        saved, see the documentation for one_univariate_risk_figure.

clear
clc

% Change the preprocessing to the one you want to use.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% First, we define different lists of CSV files.
file_list_0 = {'hrc_sick_org_all_vent_all.csv', 'hrc_healthy_org_all_vent_all.csv'};
file_list_1 = {'hrc_sick_org_all_vent_0.csv', 'hrc_healthy_org_all_vent_0.csv'};
file_list_2 = {'hrc_sick_org_all_vent_1.csv', 'hrc_healthy_org_all_vent_1.csv'};
file_list_3 = {'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_4 = {'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_5 = {'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_6 = {'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_7 = {'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_8 = {'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_9 = {'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_sick_org_ALL_vent_0.
    csv'};
file_list_10 = {'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_ALL_vent_0
    .csv'};
file_list_11 = {'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_sick_org_ALL_vent_0
    .csv'};
file_list_12 = {'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', 'hrc_sick_org_ALL_vent_0
    .csv'};

% Next, we call one_univariate_risk_figure for eacg file_list above.
one_univariate_risk_figure(file_list_0, preprocessing)
one_univariate_risk_figure(file_list_1, preprocessing)
one_univariate_risk_figure(file_list_2, preprocessing)
one_univariate_risk_figure(file_list_3, preprocessing)
one_univariate_risk_figure(file_list_4, preprocessing)
one_univariate_risk_figure(file_list_5, preprocessing)
one_univariate_risk_figure(file_list_6, preprocessing)
one_univariate_risk_figure(file_list_7, preprocessing)
one_univariate_risk_figure(file_list_8, preprocessing)
one_univariate_risk_figure(file_list_9, preprocessing)
one_univariate_risk_figure(file_list_10, preprocessing)
one_univariate_risk_figure(file_list_11, preprocessing)
one_univariate_risk_figure(file_list_12, preprocessing)
```

Listing B.18: one_univariate_risk_figure.m

```
function one_univariate_risk_figure(file_list, preprocessing)
% Author: Evan Dienstman
% Last Update: 3/31/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script creates univariate risk figures for each heart rate
% characteristics (HRC) using the CSV files in file_list. If there are
% two files in file_list, then the risk is defined as risk of file_1
% relative to file_2. If there are three files in file_list, then the
% risk is defined as file_1 relative to file_3 divided by file_2
% relative to file_3. For a further explanation of how we define risk,
% see the documentation of one_risk_matrix. The risks are then all
% plotted onto one figure. There are seven subplots in the figure,
% one for each HRC category, and each subplot has a curve that
% corresponds to one of the risks for that HRC category. For example,
% one subplot will have the risk curves for variance, variance_slope,
% variance_10, etc. There is also a text box at the top of the figrue
% that acts as a legend for each subplot. This text box also contains
% information on how many babies and half hours were used when
% calculating the risks. Note that for any sick categroy, we take any
% half hour within 24 half hours of an event. For any healthy cateogry,
% we take any half hour 7 days before an event or 3 days after event.
% If a baby never had an event, then all half hours are healthy.
```

```
%
% Arguments:
%     1. file_list - the list of csv_files for calculating the risk
%     2. preprocessing - a string indicating which preprocessing method
%        to use
%
% Preconditions:
%     1. Make sure the directories and file names used in the scripts
%        are the right ones for the computer you are using.
%     2. Make sure the files one_risk_matrix.m, one_porb_matrix.m,
%        and bin_width_maker.m are in the working directory.
%     3. The variable file_list can only contain 2 or 3 files.
%     4. This script will not overwrite any existing files with the
%        same name. Change the variable save_file to a name that does
%        not alreadcy exist, or delete the existing file with the same
%        name before running this script.
%
% Returns:
%     1. This function returns a figure saved with the name given by
%        the variable save_file that shows the univariate risks
%        relative to the files in variable file_list for each HRC.

% Change the directory names to match the directories on your
% computer. Furthermore, change the save file name to the name you want
% the save file to be.
save_file_str = [file_list{1}(1:end-4) '_vs_' file_list{2}(1:end-4)];
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
save_dir = [pwd '/Figure_Files/Dienstman_Univariate_Risks_' preprocessing '_PP'];
save_file = [save_dir '//' save_file_str '.png'];

% If the save_dir doesn't already exist, we make the save_dir here.
if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% If the save file already exists, we stop the program so we don't
% overwrite the file.
if exist(save_file, 'file')
    disp('Error: A file already exists with the save file name. The program stopped because
        running the program would overwrite the existing file.')
    return
end

% These vectors are used to create the information in the boxes at
% the top of the figures.
N = length(file_list);
baby_count_vector = zeros(1, N);
mean_half_hour_vector = zeros(1, N);

% Here, we define the HRC names.
hrc_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio', 'Decelerations', 'Mean_RR', '
    Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};
x_axis_list = {[0 10], [0 10], [0 5], [0 50], [250 600], [0 1.5], [0 5]};
x_axis_list_slope = {[-5 5], [-5 5], [-1 1], [-10 10], [-50 50], [-1 1], [-2 2]};
x_axis_plot = {[-2.5 10], [-2.5 10], [-1 4], [-10 35], [-60 560], [-0.5 1.25], [-1 4.5]};

% We take the header of one CSV file so we can look up the index of
% each HRC based on the string name.
[~, header_names] = xlsread([csv_dir '//' file_list{1}], '1:1');
site_index = find(strcmp(header_names, 'Site'));
id_index = find(strcmp(header_names, 'ID'));

% Next, we loop through every file in file_list and extract the
% necessary info.
for i = 1:N
    csv_file = [csv_dir '//' file_list{i}];
    raw_data = dlmread(csv_file,',',1,0);
    unique_matrix = unique(raw_data(:,[site_index id_index]), 'rows');
    baby_count_vector(i) = length(unique_matrix);
    mean_half_hour_vector(i) = length(raw_data)/length(unique_matrix);
    eval(['raw_data_' num2str(i) ' = raw_data;'])
end

% Next, for each HRC, we calculate the risk. For a complete explanation
% on row to calculate the risk, see the documetation for
% one_risk_matrix.m.
for i = 1:length(hrc_names)

    for j = 1:length(hrc_types)
        variable_name = [hrc_names{i} hrc_types{j}];
        variable_index = find(strcmp(header_names, variable_name)); %#ok<*NASGU>

        if j ~= 5
            xmin = x_axis_list{i}(1);
            xmax = x_axis_list{i}(2);
        else
            xmin = x_axis_list_slope{i}(1);
            xmax = x_axis_list_slope{i}(2);
        end

        if N == 2
            eval(['[' variable_name '_Risk ' variable_name '_Points] = one_risk_matrix(
                preprocessing, variable_index, xmin, xmax, raw_data_1, raw_data_2);'])
```

```
        else
            eval(['[' variable_name '_Risk ' variable_name '_Points] = one_risk_matrix(
                preprocessing, variable_index, xmin, xmax, raw_data_1, raw_data_2, raw_data_3);'])
        end
    end
end

% Lastly, we graph the risks.
color_list = {[0 0 1], [0 1 0], [1 0 0], [0 1 1], [1 0 1]};
color_name_list = {'Blue', 'Green', 'Red', 'Cyan', 'Magenta'};
title_str = {'Raw', '10', '50', '90', 'Slope'};
probability_figure = figure('Position', [50,50,1600,900]);
set(probability_figure,'color','w');

% Each subplot corresponds to one specific HRC type. On each subplot,
% there are five curves representing the ricsk for each type of HRC
% (raw, slope, 10, 50, and 90).
for i = 1:length(hrc_names)
    subplot(3,3,i+1)
    hold on

    for j = 1:length(hrc_types)
        variable_name = [hrc_names{i} hrc_types{j}];
        eval(['x = ' variable_name '_Points;'])
        eval(['y = ' variable_name '_Risk;'])
        plot(x, y, 'LineWidth', 2, 'Color', color_list{j})
    end

    xlim(x_axis_plot{i})
    ylim([-2 2])
    plot(linspace(x_axis_plot{i}(1), x_axis_plot{i}(2), 100), zeros(1,100), '--', 'Color',
        [0,0,0], 'LineWidth', 1);
    xlabel(regexprep(hrc_names{i},'_',' '), 'FontSize', 16)
    ylabel('Risk', 'FontSize', 16)
    set(gca, 'FontSize', 16)
    hold off
end

% Here, we plot the box at the top of the figure.
if  strcmp(file_list{1}, 'hrc_sick_org_all_vent_all.csv')
    baby_title_string = ['                                    Univariate Risks: ' char(10)...
        '      ' file_list{1}(1:end-4) ' vs. ' file_list{2}(1:end-4) char(10) ' '];

    for n = 1:length(hrc_types)
        baby_title_string = [baby_title_string color_name_list{n} ' = ' title_str{n} '; '];
    end

    for n = 1:length(file_list)
        file_str = file_list{n};
        baby_title_string = [baby_title_string char(10) char(10)...
            '                          ' file_str(1:end-4) char(10)...
            '            Baby Count = ' num2str(baby_count_vector(n)) ', Mean Half Hours = '
                num2str(mean_half_hour_vector(n), 4)];
    end

else
    baby_title_string = ['                          Univariate Risks: ' char(10)...
        '      ' file_list{1}(1:end-4) ' vs. ' file_list{2}(1:end-4) char(10) ' '];

    for n = 1:length(hrc_types)
        baby_title_string = [baby_title_string color_name_list{n} ' = ' title_str{n} '; '];
    end

    for n = 1:length(file_list)
        file_str = file_list{n};
        baby_title_string = [baby_title_string char(10) char(10)...
            '                          ' file_str(1:end-4) char(10)...
            '            Baby Count = ' num2str(baby_count_vector(n)) ', Mean Half Hours = '
                num2str(mean_half_hour_vector(n), 4)];
    end
end

title_frame = uicontrol('style', 'frame');
set(title_frame, 'Position', [208, 635, 345, 195], 'BackgroundColor', [0 0 0])
baby_title = uicontrol('style', 'text');
set(baby_title, 'String', baby_title_string, 'Position', [211, 638, 339, 189], 'FontSize', 8, '
    BackgroundColor', [1 1 1])

% Finally, we save the figure.
print(probability_figure, save_file, '-dpng')
end
```

Listing B.19: `one_risk_matrix.m`

```
function [risk, points, P_sigs_given_group_1, P_sigs_given_group_2] = one_risk_matrix(
    preprocessing, variables, xmin, xmax, half_hours_group_1, half_hours_group_2,
    half_hours_group_3)
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
```

```
% make sense, it might be because I haven't updated the code yet.
%
% This function calculates the risk of being in different groups given
% one or two HRC signals. The function calcualtes the risk over many
% differnt values of the signals. The exact definition of risk is
% defined in the function. We calculate the risk using the data
% provided in the "half_hours_group" arguments. The specific HRC values
% in question are determined by the argument "variables". The function
% determines which HRC values used for the risk calculation.
% Thus, the user cannot query specific values to get their risks.
%
% Arguments:
%     1. preprocessing - a string indicating which preprocessing method
%        to use
%     2. variables - a vector of number(s) that indicate which HRC
%        signal(s) to use when calculating the risk
%     3. xmin - the minimum HRC value to evaluate the risk at
%     4. xmax - the maximum HRC value to evaluate the risk at
%     5. half_hours_group_1 - the HRC data for group 1 (most likely the
%        sick hlaf hours or sick organism x half hours)
%     6. half_hours_group_2 - the HRC data for group 2 (most likely the
%        healthy half hour group)
%     7. half_hours_group_3 - the HRC data for group 3 (most likely the
%        sick organism y group)
%
% Precondtions:
%     1. Make sure the files one_porb_matrix.m hrc_indices.m, and
%        bin_width_ma.m are in the working directory.
%
% Returns:
%     1. risk - the risk at each value in the meshgrid (or vector for
%        for one signal) of the vairbale "points"
%     2. points - the matrix (or vector for one signal) of various
%        HRC values for the signal(s) specified where the meshgrid of
%        the rows in "points" correspond to the location of each "risk"
%        value
%     3. P_sigs_given_group_1 - the probability at each value in the
%        meshgrid of the vairbale "points" for the signal(s) given
%        group 1
%     4. P_sigs_given_group_2 - the probability at each value in the
%        meshgrid of the vairbale "points" for the signal(s) given
%        group 2

% Frist, we calculate the number of half hours we have.
num_half_hours_group_1 = length(half_hours_group_1);
num_half_hours_group_2 = length(half_hours_group_2);
total_half_hours = num_half_hours_group_1 + num_half_hours_group_2;

% Next, we find the probability of being in group 1, probability of
% being in group 2, probability of getting various signals given
% group 1, probability of getting various signals given group 2, and
% probability of getting various signals. Notice that P_group_1 and
% P_group_2 are numbers while the other probabilities are matrices.
% The mesh grid of the rows in "points" indicate the signal values(s)
% of the probability matrices.
P_group_1 = num_half_hours_group_1/total_half_hours;
P_group_2 = num_half_hours_group_2/total_half_hours;
[P_sigs_given_group_1, points] = one_prob_matrix(variables, half_hours_group_1, xmin, xmax,
    preprocessing);
P_sigs_given_group_2 = one_prob_matrix(variables, half_hours_group_2, xmin, xmax, preprocessing);
P_sigs = P_sigs_given_group_1 .* P_group_1 + P_sigs_given_group_2 .* P_group_2;

% We define the risk in two different ways depending on if we are
% comparing sick to healthy half hours or sick organism x to sick
% organism y half hours.
if ~exist('half_hours_group_3', 'var')

    % Here, we define the risk for comparing sick half hours to
    % healthy half hours. Note that the sick half hours can be either
    % group 1 or group 2, but generally we will use group 1 for the
    % sick half hours. Therefore, values above 0 indicates the patient
    % is more likely to be sick. We also remove any risks at points
    % where the probabilites used to calculate the risk is very small
    % (corresponding to little data).
    risk = log(P_sigs_given_group_1 ./ P_sigs);
    risk(P_sigs_given_group_1 < 0.001 & P_sigs_given_group_2 < 0.001) = NaN;

else

    % Here, we define the risk for comparing sick organism x to sick
    % organism y. In short, we first calcualte the risk of sick
    % organism x to healthy and sick organism y to healthy. Afterwards,
    % we take the ratio of these two risks to get a new risk. Thus, we
    % can interpret values above 0 as more likely to be sick from
    % organism x and values below zero as more likely to be sick from
    % organism y. Again, we remove risks at points where the
    % probabilites are very small.
    num_half_hours_group_3 = length(half_hours_group_3);
    total_half_hours_new = num_half_hours_group_3 + num_half_hours_group_2;
    P_group_3 = num_half_hours_group_3/total_half_hours_new;
    P_group_2_new = num_half_hours_group_2/total_half_hours_new;
    P_sigs_given_group_3 = one_prob_matrix(variables, half_hours_group_3, xmin, xmax,
        preprocessing);
    P_sigs_given_group_2_new = one_prob_matrix(variables, half_hours_group_2, xmin, xmax,
```

```
        preprocessing);
    P_sigs_new = P_sigs_given_group_3 * P_group_3 + P_sigs_given_group_2_new * P_group_2_new;

    risk1 = P_sigs_given_group_1./P_sigs;
    risk3 = P_sigs_given_group_3./P_sigs_new;
    risk = log(risk1./risk3);
    risk( (P_sigs_given_group_1 < 0.001 & P_sigs_given_group_2 < 0.001) | (P_sigs_given_group_3 <
        0.001 & P_sigs_given_group_2_new < 0.001) ) = NaN;
end
end
```

Listing B.20: `one_prob_matrix.m`

```
function [prob, points] = one_prob_matrix(variables, data_matrix, xmin, xmax, preprocessing)
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions!
%
% This function calculates the probability of getting various HRC
% values using the data provided in "data_matrix". The function
% determines which HRC values used for the probability calculation.
% Thus, the user cannot query specific values to get their probability.
% The specific HRCs in question are determined by the argument
% "variables". The argument "variables" can only contain one or two
% numbers corresponding to the probability of HRC X or the bivariate
% probability of HRC X and HRC Y. To calculate the probability, we use
% kernel density esimation to get a PDF, and then we integrate the PDF
% over a binwidth to get the probability.
%
% Arguments:
%     1. variables - a vector of number(s) that indicate which HRC
%        signal(s) to use from data_matrix
%     2. data_matrix - the data used to calculate the probabiliy of
%        various values of the HRC signal(s) specified
%     3. xmin - the minimum HRC value to evaluate the probability at
%     4. xmax - the maximum HRC value to evaluate the probability at
%     5. preprocessing - a string indicating which preprocessing method
%        to use
%
% Precondtions:
%     1. The Matlab function ksdensity for bivaraite PDFs only works
%        on Matlab 2016 or later.
%     2. Make sure the file hrc_indices.m and bin_widths.m are in the
%        working directory.
%
% Returns:
%     1. prob - the probability at each value in vairbale "points"
%     2. points - the matrix (or vector from one signal) of various HRC
%        values for the signal(s) specified where the meshgrid of the
%        rows in "points" correspond to the location of each "prob"
%        value

% First, we load the binwidths we use for the bandwidth of ksdensity and
% when intergrating the PDF to calculate the probabilitty. For more
% information of how we calculate the binwidth, please see the
% documentation for csv_bin_widths.m.
load(['bin_widths_' preprocessing '_PP.mat']);
load('hrc_indices.mat')

% We must split up the function into two cases: 1) when we want the
% probability given two signals and 2) when we want the probability
% given one signal.
if length(variables) == 2

    % Here, we find the probability for a matrix of (x,y)-points given
    % two HRC signals. We use N = 32 because a 32x32 matrix gives us a
    % good plot for not too much computation time.
    N = 32;

    % Next, we prepare the data for the ksdensity function.
    var1 = variables(1);
    var2 = variables(2);

    x_points = linspace(xmin(1), xmax(1), N);
    y_points = linspace(xmin(2), xmax(2), N);
    points = [x_points; y_points];
    [X, Y] = meshgrid(x_points, y_points);

    data1 = data_matrix(:,var1);
    data2 = data_matrix(:,var2);

    prob = zeros(N,N);

    % Here, we visit each point the the matrix one by one and the
    % calculate the probability around that point.
    for i = 1:N

        for j = 1:N
            x = X(i,j);
            y = Y(i,j);
            local_x_points = [x-0.5*bin_widths(var1) x+0.5*bin_widths(var1)];
```

```
            local_y_points = [y-0.5*bin_widths(var2) y+0.5*bin_widths(var2)];
            [local_X, local_Y] = meshgrid(local_x_points, local_y_points);

            % Next, we calculate the bivariate PDF using the Matlab
            % function ksdensity. We do not let ksdensity pick an
            % optimal bandwidth because the bandwith for
            % decelerations should be 1. Since there is no way to set
            % the bandwidth for decelerations as 1 and generate the
            % other bandwith automatically, we simply use our binwidths
            % as the bandwidth. We also use the Epanechnikov kernel.
            % For more information on kernel density estimation, please
            % refer to its Wikipedia page.
            p = ksdensity([data1 data2], [local_X(:) local_Y(:)], 'kernel', 'epanechnikov', 'width
                ', [0.5*bin_widths(var1) 0.5*bin_widths(var2)]);
            p = reshape(p, size(local_X));

            % Finally, we integrat the PDF to get the probability in a
            % certain area. Notice that we use our binwidth for the
            % local_x_points and local_y_points. Thus, when we
            % integrate, we expect a value close to the true
            % probability in that area.
            prob(i,j) = trapz( local_y_points, trapz(local_x_points,p,2) );
        end
    end

else

    % Here, we find the probability for a vector of x-points given one
    % HRC signal. The x-point vector is defined below. Note that each
    % x-point is one "width" apart. The importance of this feature
    % is explained below.
    data = data_matrix(:,variables);
    x_points = (xmin:bin_widths(variables):xmax).';
    points = x_points(2:end) - 0.5*bin_widths(variables);
    N = (length(x_points)-1);
    prob = zeros(N,1);

    % Next, we calculate the PDF using the Matlab function ksdensity.
    % Here, we let ksdensity chose the optimal bandwidth except for the
    % case of decelerartions where the width should be 1. We also use
    % the Epanechnikov kernel. For more information on kernel
    % density estimation, please refer to its Wikipedia page.
    if variables >= decelerations_index && variables <= decelerations_90_index
        p = ksdensity(data, x_points, 'kernel', 'epanechnikov', 'width', 1);
    else
        p = ksdensity(data, x_points, 'kernel', 'epanechnikov');
    end

    % Next, we intergrate the PDF over a standrad bandwidth to get the
    % probabilities around certain points. We use a slightly
    % different width then what ksdensity chose above so all the
    % vectors for like HRC's contain the same number of points.
    % In this manner, we can add these vectors later in
    % one_risk_matrix(). Also, since we step by a "width" in the
    % x-points vector, integrating between consecutive points will give
    % us a probability around the points close to the true value.
    for i = 1:N
        prob(i) = trapz([x_points(i) x_points(i+1)], [p(i) p(i+1)]);
    end
end
end
```

B.6 Bivariate PDF Figures

Listing B.21: `multiple_bivariate_pdf_figures.m`

```
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script calls the function one_bivariate_pdf_maker using
% different combinations of csv files. Each call to the function is
% responsible for one figure found in the folder
% Dienstman_Bivariate_PDFs. For more information on what is in these
% figures and how they were created, see the documentations for
% one_bivariate_pdf_maker.
%
% Preconditions:
%     1. Make sure the files one_bivariate_pdf_figure.m, hrc_indces,
%        and bin_width.m are in the working directory.
%     2. Make sure to select the proper preprocessing method below.
%
% Returns:
%     1. This function creates one figure for every call to
%        one_bivariate_pdf_figure saved in the folder
```

```
%         Dienstman_bivariate_PDFs.

clear
clc

% Change the variable preprocessing to match the preprocessing method
% used to make the CSV files that you will use below to create the
% bivariate PDFs.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% These are the column numbers of the CSV files we will use when
% making different bivariate PDFs. Each column corresponds to a
% different HRC. Change these numbers if you wish to use other HRCs.
% Note that we only use the raw HRCs because including all subcategories
% would create too many bivariate PDFs. We also don't use mean RR,
% asymmetry 1, and asymmetry 2 for the same reason.
load('hrc_indices.mat')
variables = [variance_index sample_entropy_index asymmetry_ratio_index decelerations_index];

% Here, we call one_bivariate_pdf_maker with different combinaitons
% of CSV files. The function one_bivariate_pdf_maker will create the
% figures saved in the folder Dienstman_Bivariate_PDFs using the two
% CSV files you pass into the function.
one_bivariate_pdf_figure(variables, 'hrc_healthy_org_all_vent_all.csv', 'hrc_sick_org_all_vent_all.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_healthy_org_all_vent_1.csv', 'hrc_sick_org_all_vent_1.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_healthy_org_all_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_2_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_3_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_3_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_4_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', preprocessing)
one_bivariate_pdf_figure(variables, 'hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_5_vent_0.csv', preprocessing)
```

Listing B.22: one_bivariate_pdf_figure.m

```
function one_bivariate_pdf_figure(variables, file1, file2, preprocessing)
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates bivariate probability density functions (PDFs)
% for certain combinations of two heart rate characteristics (HRCs).
% The PDFs are then all plotted onto one figure. There are six subplots
% in the figure, one for each combination of HRCs in the input argument
% "variables", and each subplot has two PDFs that corresponds to file1
% and file2. Note that the argument "variables" must contain exactly
% four indices (making six combinations) becuase that the figure is
% only set up to make six subplots. There is also a subtitle at the top
% of the figrue that acts as a legend for each subplot. Note that for
% any sick categroy, we take a period of 24 half hours from an event
% and for any healthy cateogry, we take a preriod of 7 days before and
% event or 3 days after an event. The PDFs are created using kernel
% density estimation. For more details about this method, see the
% comments in the code.
%
% Arguments:
%     1. variables - the six column numbers from the CSV files to use
%        when making the bivariate PDFs
%     2. file1 - the first file to calculate the bivariate PDFs of for
%        each combination of the indices in variables
%     3. file2 - the second file to calculate the bivariate PDFs of for
%        each combination of the indices in variables
%     4. preprocessing - the preprocessing method used when determining
%        which CSV files to use
%
% Preconditions:
%     1. The Matlab function ksdensity for bivaraite PDFs only works
%        on Matlab 2016 or later.
%     2. Make sure the directories and file names used in the scripts
%        are the right ones for the computer you are using.
%     3. This script will not overwrite any existing files with the
%        same name. Change the variable save_file to a name that does
```

```
%         not already exist, or delete the existing file with the same
%         name before running this script.
%     4. Make sure the file bin_widths.m is in the working
%         directory.
%     5. The input argument "variables" must contain exactyl four
%         indices.
%
% Returns:
%     1. This function returns a figure saved with the name given by
%         the variable save_file that shows the bivariate PDFs for each
%         combination of HRCs in variables for file1 and file2.

% Change the directory names to match the directories on your
% computer. Furthermore, change the bin width file name to the name of
% the file already on your computer and change the save file name to
% the name you want the save file to be.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
save_dir = [pwd '/Figure_Files/Dienstman_Bivariate_PDFs_' preprocessing '_PP'];
save_file = [save_dir '//' file1(1:end-4) '_vs_' file2(1:end-4) '.png'];
bin_width_file = ['bin_widths_' preprocessing '_PP.mat'];

% If the save_dir doesn't already exist, we make the save_dir here.
if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% If the save file already exists, we stop the program so we don't
% overwrite the file.
if exist(save_file, 'file')
    disp('Error: A file already exists with the save file name. The program stopped because
        running the program would overwrite the existing file.')
    return
end

% Here, we prepare the title for our figure by using the names from
% the csv files given.
title_name_1 = file1(1:end-4);
title_name_1 = regexprep(title_name_1, '_', ' ');

title_name_2 = file2(1:end-4);
title_name_2 = regexprep(title_name_2, '_', ' ');

% Here, we open up the CSV files. Change the file name to match the
% location of the file on your computer.
raw_data_1 = dlmread([csv_dir '//' file1], ',', 1, 0);
raw_data_2 = dlmread([csv_dir '//' file2], ',', 1, 0);

% Next, we load in the bivariate bin widths used for smoothing and
% creating the edges for the PDFs.
load(bin_width_file)

% Here, we load the header. We use the smallest file since all the
% headers are the same.
[~, variable_names] = xlsread([csv_dir '//hrc_sick_org_5_vent_0.csv'], '1:1');

% We use N = 32 because a 32x32 matrix gives us a good plot for not too
% much computation time.
N = 32;

% Next, we loop through each plot for each file (six plots for two
% files for a total of 12 plots) and create the bivariate PDFs for each
% plot.
for i = 1:12

    % If i <= 6, we are plotting the first file's PDFs. Else, we are
    % plottting the second file's PDFs.
    if i <= 6
        raw_data = raw_data_1;
        plot_num = i;
    else
        raw_data = raw_data_2;
        plot_num = i - 6;
    end

    % If i == 1, we create the figure and the title for the figure.
    if i == 1
        hist_fig = figure('Position', [50,50,1600,900]);
        set(hist_fig,'color','w');
        title_string = ['Bivariate Probability Densities: ' title_name_1 ' and ' title_name_2];
        sub_title_string = ['blue = ' title_name_1 '; red = ' title_name_2];
        figure_title = uicontrol('style', 'text');
        figure_sub_title = uicontrol('style', 'text');
        set(figure_title, 'String', title_string, 'Position', [325, 860, 1000, 40], 'FontSize',
            18, 'BackgroundColor', [1 1 1])
        set(figure_sub_title, 'String', sub_title_string, 'Position', [590, 830, 500, 30], '
            FontSize', 12, 'BackgroundColor', [1 1 1])
    end

    % Below, we check which iteration we are on to determine which
    % subplot we are at. For each subplot, we must determine the HRC
    % for the x_edge and y_dge of the bivariate PDF. Looking at the
    % code below, the variables x_edge and y_edge can be any number in
    % "variables" corresponding to an HRC column in the data file.
    if plot_num <= 3
```

```
        x_edge = variables(1);
        x_plot_edge = linspace(0,5,N);
        y_edge = variables(plot_num + 1);

        % Change the axes for the HRCs you are using.
        if y_edge == variables(2)
            y_plot_edge = linspace(0,1,N);
            z_axis = [0 3];
        elseif y_edge == variables(3)
            y_plot_edge = linspace(0,2.5,N);
            z_axis = [0 2];
        elseif y_edge == variables(4)
            y_plot_edge = linspace(0,10,N);
            z_axis = [0 0.5];
        end

    elseif plot_num <= 5
        x_edge = variables(2);
        x_plot_edge = linspace(0,1,N);
        y_edge = variables(plot_num - 1);

        % Change the axes for the HRCs you are using.
        if y_edge == variables(3)
            y_plot_edge = linspace(0,2.5,N);
            z_axis = [0 10];
        elseif y_edge == variables(4)
            y_plot_edge = linspace(0,10,N);
            z_axis = [0 1.5];
        end

    elseif plot_num == 6
        x_edge = variables(3);
        x_plot_edge = linspace(0,2.5,N);
        y_edge = variables(4);

        % Change the axes for the HRCs you are using.
        y_plot_edge = linspace(0,10,N);
        z_axis = [0 1];
    end

    % The variables x_data and y_data are the two data vecotrs used to
    % make the PDFs. Once we have selected the two data vectors, we can
    % create the bivariate PDFs. We create PDFs using the Matlab
    % function ksdensity, which uses the kernel density estimation
    % method to create the PDFs. For the kernel density estimation, we
    % use an Epanechnikov kernel and band widths equal to the bin widths
    % of each HRC. We use the bin widths we calculated earlir so all
    % the PDFs are consistent with the band widths used. We feel we can
    % more justly compare PDFs in this manner. For more information
    % about our choice of bin widths, see the documenation for
    % bin_width_maker. We also evaluate the PDFs at the edges
    % calculated earlier. The input for ksdensity is a two column
    % matrix with each column corresponding to one spefic type of HRC
    % values. Each row in the matrix corresponds to every half hour in
    % the csv file in question. Note that since we are creating PDFs,
    % the volume underneath each surface is 1. Consequently, all the
    % PDFs are normalized and values of the PDF can be greater than 1.
    % For more information about the kernel density estimation,
    % Epanechnikov kernel function, and PDFs, see their respective
    % wikipedia pages.
    x_data = raw_data(:,x_edge);
    y_data = raw_data(:,y_edge);
    [X, Y] = meshgrid(x_plot_edge, y_plot_edge);
    Z = ksdensity([x_data y_data], [X(:) Y(:)], 'kernel', 'epanechnikov', 'width', [0.5*bin_widths
        (x_edge) 0.5*bin_widths(y_edge)]);
    Z = reshape(Z, size(X));

    % Lastly, we can plot the bivariate PDFs. We plot the first
    % file's PDFs in blue and the second file's PDFs in red.
    hold on
    subplot(3,2,plot_num)

    if i <= 6
        surf(X, Y, Z, 'facealpha', .5, 'Facecolor', 'b')
    else
        surf(X, Y, Z, 'facealpha', .5, 'Facecolor', 'r')
    end

    xlabel(regexprep(variable_names{x_edge}, '_', ' '), 'FontSize', 14)
    ylabel(regexprep(variable_names{y_edge}, '_', ' '), 'FontSize', 14)
    zlabel('Probability', 'FontSize', 14)
    set(gca, 'FontSize', 14)
    xlim([x_plot_edge(1) x_plot_edge(end)])
    ylim([y_plot_edge(1) y_plot_edge(end)])
    zlim(z_axis)
    caxis(z_axis)
    hold off
end

% Finally, we save the figure.
print(hist_fig, save_file, '-dpng')
end
```

B.7 Bivariate Risk Figures

Listing B.23: `multiple_bivariate_risk_figures.m`

```
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script creates multiple bivariate risk figures. Each figure
% calculates the risk associated with different half hour groups for
% many different combination of two HRCs. For more information about
% the figures, individual plots, and how we define the risk, please see
% the documentation for one_bivariate_risk_figure.m,
% one_bivariate_risk_plot.m, and one_risk_matrix.m.
%
% Precondtions:
%     1. Make sure the files one_bivariate_risk_figure.m,
%        one_bivariate_risk_plot.m, one_risk_matrix.m,
%        one_porb_matrix.m, bin_widths.m, and hrc_indices.m are in the
%        working directory.
%     2. Make sure to select the proper preprocessing method below.
%
% Returns:
%     1. This function returns one bivariate risk figure for every call
%        to one_bivariate_risk_figure.m. For info on where the figures
%        are saved, see the documentation of one_bivariate_risk.m.

clear
clc

% Here, we select the preprocessing method.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% These are the column numbers of the CSV files we will use when
% making different bivariate risks. Each column corresponds to a
% different HRC. Change these numbers if you wish to use other HRCs.
% Note that we only use the raw HRCs because including all subcategories
% would create too many bivariate risks. We also don't use mean
% RR, asymmetry 1, and asymmetry 2 for the same reason.
load('hrc_indices.mat')
variables = [variance_index sample_entropy_index asymmetry_ratio_index decelerations_index];

% Here we define the xmins and xmaxs for plotting.
xmin_vector = {[1 0], [1 0.5], [1 0], [0 0.5], [0 0], [0.5 0]};
xmax_vector = {[4 0.8], [4 2.5], [4 10], [0.8 2.5], [0.8 10], [2.5 10]};

% Next, we call the procedure to create each figure. Each figure
% uses a different combination of half hour groups.
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector, '
    hrc_sick_org_all_vent_all.csv', 'hrc_healthy_org_all_vent_all.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_all_vent_1.csv', 'hrc_healthy_org_all_vent_1.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_all_vent_0.csv', 'hrc_healthy_org_all_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_2_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_3_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_4_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_1_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_5_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_3_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_4_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_2_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_5_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_4_vent_0.csv')
% one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_3_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_5_vent_0.csv')
one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector,'
    hrc_sick_org_4_vent_0.csv', 'hrc_sick_org_all_vent_0.csv', 'hrc_sick_org_5_vent_0.csv')
```

Listing B.24: `one_bivariate_risk_figure.m`

```
function one_bivariate_risk_figure(preprocessing, variables, xmin_vector, xmax_vector, file1,
    file2, file3)
% Author: Evan Dienstman
% Last Update: 4/5/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function creates a figure containing multiple risk plots.
```

```
% Each figure contins six subplots corresponding to six differnt
% combinations of the four input HRCs. Note that the input argument
% "variables" must contain exactly four HRCs becuase the figure is only
% set up for four. Each subplot then contains the appropriate risk plot
% for those two HRCs. The subplots are created by
% one_bivariate_risk_plot.m. For more information about the subplots
% and how we define the risk, please see the documentation for
% one_bivariate_risk_plot.m.
%
% Arguments:
%     1. preprocessing - a string indicating which preprocessing method
%        to use
%     2. variables - the column indices from the CSV files of the four
%        HRCS we want to use for the bivariate risks
%     3. xmin_vector - the minimum x-values for the subplots
%     4. xmax_vector - the maximum x-values for the subplots
%     5. file1 - the file containing the half hours for group 1
%     6. file2 - the file containing the half hours for group 2
%     7. file3 - the file containing the half hours for group 3
%
% Precondtions:
%     1. Make sure the files one_bivariate_risk_plot.m,
%        one_risk_matrix.m., one_porb_matrix.m, and bin_width_maker.m
%        are in the working directory.
%     2. Make sure there are exactly four indices in the input argument
%        "variables".
%     3. Make sure xmin_vector and xmax_vector are of the form
%        {[x1 y1], ..., [x4 y4]}.
%     4. This script will not overwrite any existing files with the
%        same name. Change the variable save_file_str to a name that
%        does not already exist, or delete the existing file with the
%        same name before running this script.
%
% Returns:
%     1. This function returns a figure containing the six subplots
%        produced by one_bivariate_risk_plot.m. Each plot represents
%        the risk associate with a different combination of two HRCs
%        relative to the two groups in question.

% First, we create the save directory.
save_dir = [pwd '/Figure_Files/Dienstman_Bivariate_Risks_' preprocessing '_PP'];

% If the save_dir doesn't already exist, we make the save_dir here.
if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% Next, we create the save file.
if exist('file3', 'var')
    save_file_str_short = [file1(1:end-4) '_vs_' file3(1:end-4)];
    save_file_str = [save_dir '//' file1(1:end-4) '_vs_' file3(1:end-4) '.png'];
else
    save_file_str_short = [file1(1:end-4) '_vs_' file2(1:end-4)];
    save_file_str = [save_dir '//' file1(1:end-4) '_vs_' file2(1:end-4) '.png'];
end

% If the save file already exists, we stop the program so we don't
% overwrite the file.
if exist(save_file_str, 'file')
    disp('Error: A file already exists with the save file name. The program stopped because
        running the program would overwrite the existing file.')
    return
end

% Next, we do all the formatting for the figure.
title_str = ['Relative risk plots: ' regexprep(save_file_str_short, '_', ' ')];
sub_title_str = ['Values above 0 mean more likely to be ' regexprep(file1(1:end-4), '_', ' ') '.'
    ];
plot_fig = figure('Position', [50,50,1600,900]);
set(plot_fig,'color','w');
figure_title = uicontrol('style', 'text');
figure_sub_title = uicontrol('style', 'text');
set(figure_title, 'String', title_str, 'Position', [410, 845, 800, 40], 'FontSize', 18, '
    BackgroundColor', [1 1 1])
set(figure_sub_title, 'String', sub_title_str, 'Position', [540, 830, 575, 20], 'FontSize', 12, '
    BackgroundColor', [1 1 1])

% Here, we call one_bivariate_risk_plot.m with each combination of
% the HRCs to make the six subplots.
if exist('file3', 'var')
    subplot(3,2,1)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(2), xmin_vector{1}, xmax_vector
        {1}, file1, file2, file3)
    subplot(3,2,2)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(3), xmin_vector{2}, xmax_vector
        {2}, file1, file2, file3)
    subplot(3,2,3)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(4), xmin_vector{3}, xmax_vector
        {3}, file1, file2, file3)
    subplot(3,2,4)
    one_bivariate_risk_plot(preprocessing, variables(2), variables(3), xmin_vector{4}, xmax_vector
        {4}, file1, file2, file3)
    subplot(3,2,5)
    one_bivariate_risk_plot(preprocessing, variables(2), variables(4), xmin_vector{5}, xmax_vector
```

```
        {5}, file1, file2, file3)
    subplot(3,2,6)
    one_bivariate_risk_plot(preprocessing, variables(3), variables(4), xmin_vector{6}, xmax_vector
        {6}, file1, file2, file3)

else
    subplot(3,2,1)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(2), xmin_vector{1}, xmax_vector
        {1}, file1, file2)
    subplot(3,2,2)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(3), xmin_vector{2}, xmax_vector
        {2}, file1, file2)
    subplot(3,2,3)
    one_bivariate_risk_plot(preprocessing, variables(1), variables(4), xmin_vector{3}, xmax_vector
        {3}, file1, file2)
    subplot(3,2,4)
    one_bivariate_risk_plot(preprocessing, variables(2), variables(3), xmin_vector{4}, xmax_vector
        {4}, file1, file2)
    subplot(3,2,5)
    one_bivariate_risk_plot(preprocessing, variables(2), variables(4), xmin_vector{5}, xmax_vector
        {5}, file1, file2)
    subplot(3,2,6)
    one_bivariate_risk_plot(preprocessing, variables(3), variables(4), xmin_vector{6}, xmax_vector
        {6}, file1, file2)
end

% Finally, we save the figure.
print(plot_fig, save_file_str, '-dpng')
end
```

Listing B.25: one_bivariate_risk_plot.m

```
function one_bivariate_risk_plot( preprocessing, var1, var2, xmin, xmax, file1, file2, file3)
% Author: Evan Dienstman
% Last Update: 3/31/2016
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This function plots the risk of group 1 compared to group 2/3 for
% for many different values of two HRC's indicated by var1 and var2.
% The data for the groups are located in file1, file2, and file3. If
% file3 does not exist, the risk is defined as group 1 compared to
% group 2. If file3 exists, the risk is defined as the ratio of group 1
% compared to group 2 divided by group 3 comapred to group 2. For more
% information about how we calculate the risk, please see the
% documnetation for one_risk_matrix.m.
%
% Arguments:
%     1. preprocessing - a string indicating which preprocessing method
%        to use
%     2. var1 - a number indicating the first HRC
%     3. var2 - a number indicating the second HRC
%     4. xmin - the minimum value for both HRCs
%     5. xmax - the maximum value for both HRCs
%     6. file1 - the file containing the half hours for group 1
%     7. file2 - the file containing the half hours for group 2
%     8. file3 - the file containing the half hours for group 3
%
% Precondtions:
%     1. Make sure the files one_risk_matrix.m., one_porb_matrix.m,
%        hrc_indices.m, and bin_widths.m are in the working directory.
%
% Returns:
%     1. This function returns a 3D plot of the risks of group 1
%        compared to group 2/3 for many differenct values of the two
%        HRCs.

% First, we read in the data.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP//'];

half_hours_group_1 = dlmread([csv_dir file1], ',', 1, 0);
half_hours_group_2 = dlmread([csv_dir file2], ',', 1, 0);

if exist('file3', 'var')
    organism = str2double(file1(15));
    half_hours_group_2 = half_hours_group_2(half_hours_group_2(:,10) ~= organism, :);
    half_hours_group_3 = dlmread([csv_dir file3], ',', 1, 0);
end

% Next, we calculte the risk for different HRC values.
if exist('file3', 'var')
    [risk_matrix, points] = one_risk_matrix(preprocessing, [var1 var2], xmin, xmax,
        half_hours_group_1, half_hours_group_2, half_hours_group_3);
else
    [risk_matrix, points] = one_risk_matrix(preprocessing, [var1 var2], xmin, xmax,
        half_hours_group_1, half_hours_group_2);
end

% Finally, we plot the risks. Note that we use a log scale. Thus, any
% risk over 0 indicates the HRC values are more likely to be in
% group 1. Any risk below 0 indicates the HRC values are more likely to
```

```
% be in group 2/3.
[~, axes_labels] = xlsread([csv_dir file1], '1:1');
[X, Y] = meshgrid(points(1,:), points(2,:));
surf(X, Y, risk_matrix)
xlabel(regexprep(axes_labels{var1}, '_', ' '))
ylabel(regexprep(axes_labels{var2}, '_', ' '))
zlabel('Elevated Risk')
zlim([log(.5) log(2)])
caxis([log(.5) log(2)])
end
```

B.8 Single Variable Logistic Figures

Listing B.26: `multiple_univariate_logistic_figures.m`

```
% Author: Evan Dienstman
% Last Update: 3/31/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script makes multiple figures that plot the probaility of
% illness for various values of each HRC. We first start off by making
% figures for on specific category of half hours. For example, we might
% only use the nonventilated half hours. Next, we make five figures for
% each of these half hour categories corresponding to the five HRC
% subcategories. Each figure then contains seven subplots corresponding
% to the seven HRCs. On each subplot, we graph the probability of
% illness calculated from single variable logistic regression and from
% bayesian methods. We also graph the univariate PDFs for the HRCs on
% the subplot to give a sense of where most of the data lies. For
% example, the PDF of variance from 0-1 is very small. Therefore,
% even though a variance between 0-1 indicates an increased
% probability of being ill, we also want to note that these values are
% very rare to begin with. For extremely rare values, we do not even
% plot the probability of illness. For more info about the figures,
% see the documentation of one_univariate_logistic_plot.m.
%
% Precondtions:
%     1. Make sure the files one_univariate_logistic_figure,
%        one_risk_matrix.m, one_porb_matix.m, hrc_indices.m, and
%        hrc_bin_widths.m are in the current working directory.
%     2. Make sure all the CSV files are in the appropriate directory.
%     3. Make sure to select the proper preprocessing method below.
%
% Returns:
%     1. This script returns multiple single variable logistic figures
%        saved to the directory indicated in
%        one_univariate_logistic_figure.m with the file name given
%        in the call to one_univariate_logistic_figure.m.

clear
clc

% Here, we select the preprocessing method.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% Next, we define the CSV files we want to use for the figures. Note
% that we don't consider and specific organism categories because the
% PDF and risk figures told us that the HRCs are not useful for
% distinguishing amongst different organisms.
file_list_1 = {'hrc_sick_org_all_vent_all.csv', 'hrc_healthy_org_all_vent_all.csv'};
file_list_2 = {'hrc_sick_org_all_vent_0.csv', 'hrc_healthy_org_all_vent_0.csv'};
file_list_3 = {'hrc_sick_org_all_vent_1.csv', 'hrc_healthy_org_all_vent_1.csv'};

% Finally, we call one_univariate_logistic_figure to make the
% figures for each group of CSV files.
one_univariate_logisitc_figure(preprocessing, file_list_1, 'hrc_all_org_all_vent_all')
one_univariate_logisitc_figure(preprocessing, file_list_2, 'hrc_all_org_all_vent_0')
one_univariate_logisitc_figure(preprocessing, file_list_3, 'hrc_all_org_all_vent_1')
```

Listing B.27: `one_univariate_logistic_figure.m`

```
function one_univariate_logisitc_figure(preprocessing, file_list, save_str)
% Author: Evan Dienstman
% Last Update: 3/31/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script makes fives figures that plot the probaility of
% illness for various values of each HRC using the CSV files in
% file_list. Each of the five figures corresponds to one of the five
```

```
% HRC subcategories (raw, 10th, 50th, 90th, and slope). Each figure
% then contains seven subplots corresponding to the seven HRCs. On each
% subplot, we graph the probability of illness calculated from single
% variable logistic regression and from bayesian methods. We also
% graph the univariate PDFs for the HRCs on the subplot to give a sense
% of where most of the data lies. For example, the PDF of variance
% from 0-1 is very small. Therefore, even though a variance between 0-1
% indicates an increased probability of being ill, we also want to note
% that these values are very rare to begin with. For extremely rare
% values, we do not even plot the probability of illness. For more info
% about the figures, see the comments below.
%
% Arguments:
%     1. preprocessing - the type of preprocessing used to create the
%        CSV files
%     2. file_list - the list of CSV files used to make the figure
%     3. save_file - the name of the file the figure is saved to
%
% Precondtions:
%     1. Make sure the files one_risk_matrix.m, one_porb_matix.m,
%        hrc_indices.m, and hrc_bin_widths.m are in the current working
%        directory.
%     2. Make sure all the CSV files are in the appropriate directory.
%     3. This script will not overwrite any existing files with the
%        same name. Change the variable save_str to a name that does
%        not alreadcy exist, or delete the existing file with the same
%        name before running this script.
%
% Returns:
%     1. This function returns five figures corresponding to the five
%        HRC subcategories. The figures are saved to the directory
%        indicated by the variable save_dir with the file name given
%        by the variable save_file.

% Frist, load hrc_indices.mat and define some constants and variables
% used later.
load('hrc_indices.mat')
model = 'quadratic';
hrc_names = {'Asymmetry_1', 'Asymmetry_2', 'Asymmetry_Ratio', 'Decelerations', 'Mean_RR', '
    Sample_Entropy', 'Variance'};
hrc_types = {'', '_10', '_50', '_90', '_Slope'};
title_strs = {'Raw HRC', '10th Percentile', '50th Percentile', '90th Percentile', 'HRC Slope'};
x_axis_list = {[0 10], [0 10], [0 5], [0 50], [250 600], [0 1.5], [0 5]};
x_axis_list_slope = {[-5 5], [-5 5], [-1 1], [-10 10], [-50 50], [-1 1], [-2 2]};

% Next, we read in the data from the CSV files.
csv_dir = [pwd '/Data_Files/Dienstman_CSV_Files_' preprocessing '_PP'];
csv_matrix_1 = dlmread([csv_dir '//' file_list{1}], ',', 1, 0);
csv_matrix_2 = dlmread([csv_dir '//' file_list{2}], ',', 1, 0);
csv_matrix = [csv_matrix_1; csv_matrix_2];
healthy_half_hours = csv_matrix(csv_matrix(:,health_status_index) == 0,:); %#ok<*NASGU>
sick_half_hours = csv_matrix(csv_matrix(:,health_status_index) == 1,:);
p_sick = length(sick_half_hours)/length(csv_matrix);

% We now loop through all five HRC subcategories and create a figure
% for each one.
for i = 1:length(hrc_types)

    % Here, we create the save directory and save file for the figure.
    save_dir = [pwd '/Figure_Files/Dienstman_Logistic_Figures_' preprocessing '_PP'];
    save_file = [save_dir '//' save_str hrc_types{i} '.png'];

    % If the save_dir doesn't already exist, we make the save_dir here.
    if ~exist(save_dir, 'dir')
        mkdir(save_dir)
    end

    % If the save file already exists, we stop the program so we don't
    % overwrite the file.
    if exist(save_file, 'file')
        disp('Error: A file already exists with the save file name. The program stopped because
            running the program would overwrite the existing file.')
        continue
    end

    % Here, we create the figure, info box in the top left corner, and
    % the annotation at the bottom of the figure.
    probability_figure = figure('Position', [50,50,1600,900]);
    set(probability_figure,'color','w');
    title_frame = uicontrol('style', 'frame');
    if i >= 2 && i <=4
        baby_title_string = [save_str ': ' title_strs{i} char(10) char(10)];
    else
        baby_title_string = ['    ' save_str ': ' title_strs{i} char(10) char(10)];
    end
    baby_title_string = [baby_title_string ...
        ' Probability from Healthy (0) to Sick (1)' char(10)...
        '     Red Line = Bayesian Probability' char(10)...
        '    Blue Line = Logistic Probability' char(10) char(10)...
        'Fraction from No Data (0) to All Data (1)' char(10)...
        '        Red Area = Sick Fraction' char(10)...
        '      Blue Area = Healthy Fraction']; %#ok<*AGROW>
    set(title_frame, 'Position', [210, 635, 340, 195], 'BackgroundColor', [0 0 0])
    baby_title = uicontrol('style', 'text');
```

```
    set(baby_title, 'String', baby_title_string, 'Position', [213, 638, 334, 189], 'FontSize', 13,
        'BackgroundColor', [1 1 1])
    axes_note_string = 'All red and blue areas are additive, i.e., no areas are hidden behind one
        another.';
    axes_note = uicontrol('style', 'text');
    set(axes_note, 'String', axes_note_string, 'Position', [530,-20,600, 50], 'FontSize', 12, '
        BackgroundColor', [1 1 1])

    % Here, we iterate through all seven HRCs and create the
    % corresponding probability plot for that HRC.
    for j = 1:length(hrc_names)

        % First, we specify the subplot.
        subplot(3,3,j+1)
        var_name = [hrc_names{j} hrc_types{i}];
        var_index = [lower(var_name) '_index'];

        % Next, we separate the data into the HRC values (data_vector)
        % and health status (response_vector).
        eval(['data_vector = csv_matrix(:,' var_index ');'])
        response_vector = csv_matrix(:,health_status_index);
        [data_vector, sorted_indices] = sort(data_vector);
        response_vector = response_vector(sorted_indices);

        % Before we calculate the probabilities, we remove outliers
        % from the data. For the Bayesian method, removing outliers
        % will not affectthe results becuase outliers will have very
        % low probabilities. However, we want to remove outliers for
        % the logistic probability becuase over fitting will
        % considerably change the tails if we keep the outliers. For
        % decelerations, we define the high_fence as 30 becuase the
        % outlier method removes too much data. This procedure is
        % strictly empircal and needs to be analyzed further.
        q1 = quantile(data_vector, .25);
        q3 = quantile(data_vector, .75);
        IQR = q3 - q1;
        low_fence = q1 - 1.5*IQR;
        high_fence = q3 + 1.5*IQR;

        if strcmp(hrc_names{j}, 'Decelerations')
            data_vector(data_vector < low_fence | data_vector > 30) = NaN;
            response_vector(data_vector < low_fence | data_vector > 30) = NaN;
        else
            data_vector(data_vector < low_fence | data_vector > high_fence) = NaN;
            response_vector(data_vector < low_fence | data_vector > high_fence) = NaN;
        end

        % Now we calculate the probability of illness using logistic
        % regression. We use the Matlab function fitglm to calculate the
        % probability. For more information, please see the Matlab
        % documentation of this function.
        fit = fitglm(data_vector, response_vector, model, 'distribution', 'binomial', 'VarNames',
            {var_name, 'Health_Status'});
        log_prob = fit.Fitted.Probability;

        % Now we calculate the probability of illness using the Bayesian
        % probability. We use the function one_rish_matrix to
        % calculate the probability. For more information, please see
        % the documentation of this function.
        if strcmp(title_strs{i}, 'HRC Slope')
            eval(['[risk, x_points, p_sigs_given_sick, p_sigs_given_healthy] = one_risk_matrix(
                preprocessing,' var_index ', x_axis_list_slope{j}(1), x_axis_list_slope{j}(2),
                sick_half_hours, healthy_half_hours);'])
        else
            eval(['[risk, x_points, p_sigs_given_sick, p_sigs_given_healthy] = one_risk_matrix(
                preprocessing,' var_index ', x_axis_list{j}(1), x_axis_list{j}(2), sick_half_hours
                , healthy_half_hours);'])
        end
        bayesian_prob = p_sick .* exp(risk);

        % In each subplot, we also include a PDF for the sick and
        % healthy half hours. With this information, we can visualize
        % how much data we have at each HRC value. We want this
        % information because we are more confident in probabilities at
        % values with more data.
        hold on
        yyaxis right
        myarea = area(x_points, [p_sigs_given_sick + p_sigs_given_healthy, p_sigs_given_healthy],
            'FaceAlpha', 0.25, 'LineStyle', 'none');
        myarea(1).FaceColor = 'B';
        myarea(2).FaceColor = 'R';
        ylim([0 1])
        ylabel('Fraction of Data')
        yyaxis left
        plot(data_vector, log_prob, 'LineWidth', 2, 'Color', 'B')
        plot(x_points, bayesian_prob, '-', 'LineWidth', 2, 'Color', 'R')
        ylabel('Probability')
        ylim([0 0.025])

        % Lastly, we set and label the axes for the subplot.
        if strcmp(hrc_names{j}, 'Mean_RR') || strcmp(title_strs{i}, 'HRC Slope')
            xmin = min([data_vector; x_points]);
        else
            xmin = 0;
```

```
        end

        xmax = max([data_vector; x_points]);
        xlim([xmin xmax])
        xlabel(regexprep(hrc_names{j}, '_', ' '))
        set(gca, 'FontSize', 14)
        hold off
    end

    % Finally, we save the figure.
    print(probability_figure, save_file, '-dpng')
end
end
```

B.9 HeRO Score Figures

Listing B.28: `multiple_hero_score_figures.m`

```
% Author: Evan Dienstman
% Last Update: 4/7/2017
% Email: eddienstman@email.wm.edu
% Note: Feel free to email me with questions! If something doesn't
% make sense, it might be because I haven't updated the code yet.
%
% This script creates a hero score figure for every septic events.
% Each hero score figure shows six hero scores at each half hour of
% the Dienstman_results file associated with the event in question.
% The six hero scores are Dienstman_all, Dienstman_vent,
% Dienstman_nonvent, Hrch, Hrcg, and Hero. Each hero score is calculted
% using the half hours in the Dienstman_results files. Starting from
% the half hour in question, the hero score looks at the prior 24 half
% hours when considering what heart rate characteristics (HRCs) to use
% in the score. Thus, the first hero score in the figure looks at half
% hours 1-24 in the Dienstman_results file, the second hero score looks
% at half hours 2-25, and so on. The verticle black line in the figure
% represents the time of the event. For more information about how to
% calculate the hero score, see the documentations for
% one_hero_score_figure. This function then creates a figure containg
% the average hero scores calculated from all the individual figures.
%
% Preconditions:
%     1. Make sure the directories and file names used in the script
%        are the right ones for the computer you are using.
%     2. Make sure the files event_matrix.mat, Doug_coeffs.mat,
%        Dienstman_coeffs.mat, prctile1.m, logistic.m, and
%        one_hero_score_figure.m are in the working directory.
%     3. This function will not overwrite the average figure that
%        already exists. Delete the old figure or change the name of
%        save_file below.
%
% Returns:
%     1. This script returns a hero score figures for every septic event
%        and an average figure for all the events.

clear
clc

% Change the preprocessing to the one you want to use.
preprocessing = 'Abby';
% preprocessing = 'Doug';

% First, we create the save_sir and save_file.
save_dir = [pwd '/Figure_Files/Dienstman_Hero_Scores_' preprocessing '_PP/Average'];
save_file = [save_dir '//average_hero_figure.fig'];

if ~exist(save_dir, 'dir')
    mkdir(save_dir)
end

% If the save file already exists, we stop the program so we don't
% overwrite the file.
if exist(save_file, 'file')
    disp(['Error: File ' save_file ' already exists. Did not execute because program would alter
        existing file.'])
    return
end

% Here, we create the container that let's us translate between the
% site codes and the site numbers. The site refers to the hospitals of
% the babies.
site_map_keys = {11, 13, 15, 23, 24, 26, 27, 30};
site_map_values = {'UVA', '0d', '0f', '17', '18', '1a', '1b', '1e'};
site_map = containers.Map(site_map_keys, site_map_values);
plot_str = 'yes';

% Here, we load in the variable event_matrix and hrc_indices. The
% variable event_info contains the site, ID, time, organism, and
```

```
% other demographic info for each spetic event. The variable
% hrc_indices contains the column indices of the CSV files for
% each HRC.
load('event_matrix.mat')
load('hrc_indices.mat')

% Next, we define some variables to use later.
index_vector = -335:144;
mean_dienstman_hero_vector_all = zeros(1,480);
mean_dienstman_hero_vector_vent = zeros(1,480);
mean_dienstman_hero_vector_nonvent = zeros(1,480);
mean_hrch_vector = zeros(1,480);
mean_hrcg_vector = zeros(1,480);
mean_hero_score_vector = zeros(1,480);
all_count_vector = zeros(1,480);
vent_count_vector = zeros(1,480);
nonvent_count_vector = zeros(1,480);
count = 1;

% Next, we interate through every index in rand_indices and create
% a figure for the event corresponding to that index in event_matrix.
for rand_index = 1:length(event_matrix)
    id = event_matrix(rand_index,1);
    site_num = event_matrix(rand_index,2);
    site = site_map(site_num);
    event_time = event_matrix(rand_index,7);

    save_dir = [pwd '/Figure_Files/Dienstman_Hero_Scores_' preprocessing '_PP/' site];
    Dienstman_dir = [pwd '/Data_Files/Dienstman_Results_' preprocessing '_PP/' site];
    Dienstman_file = [Dienstman_dir '//Dienstman_hrc_results_' site '_' num2str(id) '.mat'];

    % If the save directory doesn't exist, we make it here.
    if ~exist(save_dir, 'dir')
        mkdir(save_dir)
    end

    % Lastly, we call one_hero_score_figure with the information
    % for this event to create the hero score figure.
    [one_dienstman_hero_vector_all, one_dienstman_hero_vector_vent,
        one_dienstman_hero_vector_nonvent, one_hrch_vector, one_hrcg_vector, time_vector] =
        one_hero_score_figure(id, site_num, event_time, save_dir, Dienstman_file, plot_str);

    % After we create the individual figure, we update the average
    % vectors.
    for i = 1:480
        start_time = (event_time - 7) + i/48;
        end_time = (event_time - 7) + (i+1)/48;
        index = find(time_vector > start_time & time_vector < end_time);

        if length(index) == 1 && ~isnan(one_dienstman_hero_vector_all(index))
            mean_dienstman_hero_vector_all(i) = nansum([mean_dienstman_hero_vector_all(i),
                one_dienstman_hero_vector_all(index)]);
            mean_dienstman_hero_vector_vent(i) = nansum([mean_dienstman_hero_vector_vent(i),
                one_dienstman_hero_vector_vent(index)]);
            mean_dienstman_hero_vector_nonvent(i) = nansum([mean_dienstman_hero_vector_nonvent(i),
                one_dienstman_hero_vector_nonvent(index)]);
            mean_hrch_vector(i) = nansum([mean_hrch_vector(i), one_hrch_vector(index)]);
            mean_hrcg_vector(i) = nansum([mean_hrcg_vector(i),  one_hrcg_vector(index)]);
            mean_hero_score_vector(i) = nansum([mean_hero_score_vector(i), max(one_hrch_vector(
                index), one_hrcg_vector(index))]);
            all_count_vector(i) = all_count_vector(i) + 1;

            if ~isnan(one_dienstman_hero_vector_vent(index))
                vent_count_vector(i) = vent_count_vector(i) + 1;
            end

            if ~isnan(one_dienstman_hero_vector_nonvent(index))
                nonvent_count_vector(i) = nonvent_count_vector(i) + 1;
            end
        end
    end

    count = count + 1;
end

% Here, we calculate the average vectors.
mean_dienstman_hero_vector_all = mean_dienstman_hero_vector_all./all_count_vector;
mean_dienstman_hero_vector_vent = mean_dienstman_hero_vector_vent./vent_count_vector;
mean_dienstman_hero_vector_nonvent = mean_dienstman_hero_vector_nonvent./nonvent_count_vector;
mean_hrch_vector = mean_hrch_vector./all_count_vector;
mean_hrcg_vector = mean_hrcg_vector./all_count_vector;
mean_hero_score_vector = mean_hero_score_vector./all_count_vector;

% Finally, we plot the average vectors.
figure_handle = figure('Position', [50,50,1600,900]);
set(figure_handle,'color','w');
hold on
plot(index_vector, mean_dienstman_hero_vector_all, 'LineWidth', 2, 'Color', 'R')
plot(index_vector, mean_dienstman_hero_vector_vent, 'LineWidth', 2, 'Color', [1 1 0])
plot(index_vector, mean_dienstman_hero_vector_nonvent, 'LineWidth', 2, 'Color', [0 1 1])
plot(index_vector, mean_hrch_vector, 'LineWidth', 2, 'Color', 'B')
plot(index_vector, mean_hrcg_vector, 'LineWidth', 2, 'Color', 'G')
plot(index_vector, mean_hero_score_vector, 'LineWidth', 2, 'Color', [1 0 1])
line([0 0], ylim, 'Color', [0,0,0], 'LineWidth', 3);
```

```
157 legend('Dienstman Hero Score', 'Dienstman Hero Score Vent', 'Dienstman Hero Score Nonvent', 'Hrch
        Sore', 'Hrcg Score', 'Hero Score')
158 title(['Average HeRO Score: Mean Babies Used Per Half Hour - ' num2str(mean(all_count_vector))], '
        FontSize', 24)
159 xlabel('Time of HeRO Score', 'FontSize', 16)
160 ylabel('Hero Score', 'FontSize', 16)
161 set(gca,'fontsize',16)
162 hold off
163
164 % Lastly, we save the average figure.
165 hgsave(figure_handle, save_file, '-v7.3');
```

Listing B.29: one_hero_score_figure.m

```
1  function [dienstman_hero_vector_all, dienstman_hero_vector_vent, dienstman_hero_vector_nonvent,
       hrch_vector, hrcg_vector, plot_time_vector] = one_hero_score_figure(id, site_num, event_time,
       save_dir, Dienstman_file, plot_str)
2  % Author: Evan Dienstman
3  % Last Update: 4/7/2017
4  % Email: eddienstman@email.wm.edu
5  % Note: Feel free to email me with questions! If something doesn't
6  % make sense, it might be because I haven't updated the code yet.
7  %
8  % This function creates a HeRO score figure for one septic event. We
9  % save the figure to the directory Dienstman_Hero_Scores. Each figure
10 % contains the multiple HeRO scores calculated at each half hour The
11 % HeRO scores include Dienstman_Hero_All, Dienstman_Hero_Vent,
12 % Dienstman_Hero_Nonvent, Hero, Hrch, and Hrcg. For more info on these
13 % six HeRO scores, please see the code below. We calculate the HeRO
14 % scores using the heart rate characteristics (HRCs) for each half hour.
15 % The HRCs for each half hour are stored in the Dienstman_results files.
16 % The HeRO score looks over a window of half hours in the past given by
17 % the variable half_hour_window_length found in the code below. The
18 % HeRO score then takes a certain percentile value for each HRC to
19 % calculate the HeRO score. Since we are looking at a window into the
20 % past, the time of the first HeRO score in the figure is the time of
21 % the half hour in the Dienstman_file with an index equal to the number
22 % half_hour_window_length. Note that the percentiles of the HRCs have
23 % already been calculated in the Dienstman_result files. Thus, we do
24 % not need to calculate them in this file. The black verticle line
25 % represents the time of the event.
26 %
27 % Arguments:
28 %     1. id - the id number of the baby
29 %     2. site - the site of the baby
30 %     3. event_time - the time of the septic event for the baby
31 %     4. save_dir - the directory where we will save the figure
32 %     5. Dienstman_file - the Dienstman_file that corresponds to the
33 %        event that we will calculate the HeRO scores for
34 %
35 % Preconditions:
36 %     1. Make sure the directories and file names used in the scripts
37 %        are the right ones for the computer you are using.
38 %     2. Make sure the files Dienstman_coeffs.mat, Doug_coeffs.mat,
39 %        prctile1.m, and logistic.m, are in the working directory.
40 %     3. Make sure the variable half_hour_window_length matches the
41 %        time window you want the HeRO score to calculate.
42 %     4. This function will not overwrite any figure that already
43 %        exists. Delete the old figure or change the name of save_file
44 %        below.
45 %
46 % Returns:
47 %     1. dienstman_hero_vector_all - the vector containing the
48 %        Dienstman_Hero_All HeRO score
49 %     2. dienstman_hero_vector_vent - the vector containing the
50 %        Dienstman_Hero_Vent HeRO score
51 %     3. dienstman_hero_vector_nonvent - the vector containing the
52 %        Dienstman_Hero_Nonvent HeRO score
53 %     4. hrch_vector - the vector containing the Hrch HeRO score
54 %     5. hrcg_vector - the vector containing the Hrcg HeRO score
55 %     6. plot_time_vector - the vector containing the start times
56 %        of the half hours
57 %     7. This function also returns a figure with all six HeRO scores at
58 %        every half hour of the file Dienstman_file.
59
60 % First, we define some variables that we will use later.
61 site_map_keys = {11, 13, 15, 23, 24, 26, 27, 30};
62 site_map_values = {'UVA', '0d', '0f', '17', '18', '1a', '1b', '1e'};
63 site_map = containers.Map(site_map_keys, site_map_values);
64 site = site_map(site_num);
65 save_file = [save_dir '//hero_figure_' site '_' num2str(id) '_' num2str(round(event_time)) '.fig'
       ];
66
67 % If the save file already exists, we stop the program so we don't
68 % overwrite the file.
69 if exist(save_file, 'file')
70     disp(['Error: File ' save_file ' already exists. Did not execute because program would alter
          existing file.'])
71     hrch_vector = NaN;
72     hrcg_vector = NaN;
73     dienstman_hero_vector_all = NaN;
74     dienstman_hero_vector_vent = NaN;
```

```
75      dienstman_hero_vector_nonvent = NaN;
76      plot_time_vector = NaN;
77      return
78  end
79
80  % We then load in the Dienstman file containg the half hours needed
81  % to calculate the hero scores for this event. If the Dienstman_file
82  % doesn't exists, we return to the calling function.
83  if exist(Dienstman_file, 'file')
84      load_variable = load(Dienstman_file);
85      struct_name = fieldnames(load_variable);
86      struct_name = struct_name(1);
87      eval_str = ['Dienstman_struct = load_variable.' char(struct_name) ';'];
88      eval(eval_str)
89  else
90      hrch_vector = NaN;
91      hrcg_vector = NaN;
92      dienstman_hero_vector_all = NaN;
93      dienstman_hero_vector_vent = NaN;
94      dienstman_hero_vector_nonvent = NaN;
95      plot_time_vector = NaN;
96      return
97  end
98
99  % Here, we load in variables needed to calculate the hero score. Each
100 % one of the variables is a coefficient in the hero score model that
101 % has been optimized to give the best results.
102 load Doug_coeffs cg ch u0
103 load('Dienstman_coeffs_all.mat')
104 load('Dienstman_coeffs_vent.mat')
105 load('Dienstman_coeffs_nonvent.mat')
106 load('vent_matrix.mat')
107
108 % Change this number if you want the HeRO score to encompass more or
109 % less half hours in the past. Currently, it is set to 24 to encompass
110 % 24 half hours in the past.
111 half_hour_window = 24;
112
113 % Using the Dienstman_file we loaded, we create the time_vector each
114 % hero score will be calculated at. Note that we start the time
115 % starting at half_hour_window_length because the first score will
116 % look at that number of half hours in the past. We then prealocate
117 % the vectors below that we will plot later.
118 time_vector = [Dienstman_struct(:).Start_Time]; %#ok<*NODEF>
119 plot_time_vector = time_vector(half_hour_window:end);
120 num_of_hero_scores = length(time_vector) - half_hour_window + 1;
121 hero_score_vector = zeros(1, num_of_hero_scores);
122 hrch_vector = zeros(1, num_of_hero_scores);
123 hrcg_vector = zeros(1, num_of_hero_scores);
124 dienstman_hero_vector_all = zeros(1, num_of_hero_scores);
125 dienstman_hero_vector_vent = zeros(1, num_of_hero_scores).*NaN;
126 dienstman_hero_vector_nonvent = zeros(1, num_of_hero_scores).*NaN;
127 vent_indices = find(vent_matrix(:,1) == site_num & vent_matrix(:,2) == id); %#ok<*NODEF>
128 baby_vent_info = vent_matrix(vent_indices,3:4); %#ok<*FNDSB>
129
130 % Now we iterate through every half hour, calculating the hero score
131 % each time. Again, we start at the half hour with index
132 % half_hour_window_length becuase the hero score looks at that
133 % number of half hours in the past.
134 for i = half_hour_window:(num_of_hero_scores + half_hour_window - 1)
135     hrc_entry = Dienstman_struct(i);
136     indices = find(time_vector <= hrc_entry.Start_Time & time_vector >= (hrc_entry.Start_Time -
          half_hour_window/48));
137
138     if ~isempty(hrc_entry.Start_Time) && ~isempty(indices)
139         hrc_struct = Dienstman_struct(indices);
140
141         % First, we calculate the Hrch, Hrcg, and Hero scores. The
142         % components of these HeRO scores are contained in the vector
143         % hero_score_nums_1. These HeRO scores look at different
144         % percentiles of HRCs over the half_hour_window. The
145         % coefficients for these HeRO scores are contained in
146         % Doug_coeffs.mat
147         hero_score_nums_1 = zeros(1,6);
148         hero_score_nums_1(1) = hrc_entry.Variance_10;
149         hero_score_nums_1(2) = hrc_entry.Sample_Entropy_10;
150         hero_score_nums_1(3) = hrc_entry.Asymmetry_1_50;
151         hero_score_nums_1(4) = hrc_entry.Asymmetry_2_50;
152         hero_score_nums_1(5) = prctile1([hrc_struct(:).Asymmetry_2] - [hrc_struct(:).Asymmetry_1],
              50);
153         hero_score_nums_1(6) = hrc_entry.Variance_50;
154         hrch = logistic(ch, hero_score_nums_1([1 5])) / u0;
155         hrcg = logistic(cg, hero_score_nums_1([6 2 3 4])) / u0;
156         hero_score = max(hrch,hrcg);
157
158         % Next, we determine if the half hour is ventilated.
159         if ~isempty(find(hrc_entry.Start_Time > baby_vent_info(:,1) & hrc_entry.Start_Time <
              baby_vent_info(:,2), 1))
160             ventilated = 'yes';
161         else
162             ventilated = 'no';
163         end
164
165         % Now we calculate the Dienstman_Hero_Vent or
```

```
        % Dienstman_Hero_Nonvent HeRO score depending on the
        % ventilation status of the half hour as well as the
        % Dienstman_Hero_All score.
        variables = {'Asymmetry_1_10', 'Asymmetry_1_Slope',...
            'Asymmetry_2_90', 'Asymmetry_2_Slope',...
            'Asymmetry_Ratio_50', 'Asymmetry_Ratio_Slope',.....
            'Decelerations_90',...
            'Sample_Entropy_10', 'Sample_Entropy_Slope',...
            'Variance_10', 'Variance_90', 'Variance_Slope'};
        N = length(variables);
        num_of_coeffs = sum(1:N) + N;
        hero_score_nums_2 = zeros(1,N);
        hero_score_nums_3 = zeros(1,N);

        if strcmp(ventilated, 'no')
            coeffs2 = nonvent_coeffs;
        else
            coeffs2 = vent_coeffs;
        end

        % Here, we set the components of the Dienstman_Hero_Vent/Nonvent
        % and Dienstman_Hero_All HeRO scores.
        for j = 1:N
            eval(['hero_score_nums_2(j) = hrc_entry.' variables{j} ';'])
            eval(['hero_score_nums_3(j) = hrc_entry.' variables{j} ';'])
        end

        % Since we have a quadratic model, we now calculate all the
        % cross products from the HRC components. We save these new
        % components in Dienstman_hero_score_nums_2. Note that the
        % order we save these components is important because they
        % must match the coefficients in Dienstman_coeffs_vent/nonvent
        % and Dienstman_coeffs_all.
        Dienstman_hero_score_nums_2 = zeros(1, num_of_coeffs);
        Dienstman_hero_score_nums_3 = zeros(1, num_of_coeffs);
        count = 1;

        for j = 1:N
            Dienstman_hero_score_nums_2(count) = hero_score_nums_2(j);
            Dienstman_hero_score_nums_3(count) = hero_score_nums_3(j);
            count = count + 1;
        end

        for j = 1:N
            for k = j+1:N
                Dienstman_hero_score_nums_2(count) = hero_score_nums_2(j) * hero_score_nums_2(k);
                Dienstman_hero_score_nums_3(count) = hero_score_nums_3(j) * hero_score_nums_3(k);
                count = count + 1;
            end
        end

        for j = 1:N
            Dienstman_hero_score_nums_2(count) = hero_score_nums_2(j)^2;
            Dienstman_hero_score_nums_3(count) = hero_score_nums_3(j)^2;
            count = count + 1;
        end

        % Finally, we calculate the hero score for
        % Dienstman_Hero_Vent/Nonvent and Dienstman_Hero_all.
        if sum(isnan(Dienstman_hero_score_nums_2)) == 0 && sum(isnan(Dienstman_hero_score_nums_3))
            == 0
            Dienstman_hero_score_2 = logistic(coeffs2, Dienstman_hero_score_nums_2) / u0;
            Dienstman_hero_score_3 = logistic(all_coeffs, Dienstman_hero_score_nums_3) / u0;
        else
            Dienstman_hero_score_2 = NaN;
            Dienstman_hero_score_3 = NaN;
        end

        % We then save the all the HeRO scores for plotting later.
        hero_score_vector(i-half_hour_window+1) = hero_score;
        hrch_vector(i-half_hour_window+1) = hrch;
        hrcg_vector(i-half_hour_window+1) = hrcg;
        dienstman_hero_vector_all(i-half_hour_window+1) = Dienstman_hero_score_3;

        if strcmp(ventilated, 'no')
            dienstman_hero_vector_nonvent(i-half_hour_window+1) = Dienstman_hero_score_2;
        else
            dienstman_hero_vector_vent(i-half_hour_window+1) = Dienstman_hero_score_2;
        end
    end
end

% Finally, we plot the HeRO scores at the time of each half hour.
if strcmp(plot_str, 'yes')
    figure_handle = figure('Position', [50,50,1600,900]);
    set(figure_handle,'color','w');
    hold on
    plot_step = ceil(num_of_hero_scores * .01);
    plot(plot_time_vector, dienstman_hero_vector_all, 'LineWidth', 4, 'Color', [.5 .5 .5])
    plot(plot_time_vector, dienstman_hero_vector_vent, 'LineWidth', 4, 'Color', [135/255 206/255
        250/255])
    plot(plot_time_vector, dienstman_hero_vector_nonvent, 'LineWidth', 4, 'Color', [0 0 205/255])
    plot(plot_time_vector, hero_score_vector, 'LineWidth', 4, 'Color', 'G')
    plot(plot_time_vector(1:plot_step:end), hrch_vector(1:plot_step:end), '+', 'MarkerSize', 10)
```

```
258         plot(plot_time_vector(1:plot_step:end), hrcg_vector(1:plot_step:end), 'o', 'MarkerSize', 10)
259         line([event_time event_time], ylim, 'Color', [0,0,0], 'LineWidth', 3);
260         legend('Dienstman Hero', 'Dienstman Vent Hero', 'Dienstman Nonvent Hero', 'Legacy Hero', 'Hrch
                Sore', 'Hrcg Score')
261         title(['Moving HeRO Score (ID: ' num2str(id) ', Site: ' site ')'], 'FontSize', 24)
262         xlabel('Time of HeRO Score (Days)', 'FontSize', 16)
263         ylabel('HeRO Score Value', 'FontSize', 16)
264         set(gca,'fontsize',16)
265         hold off
266         hgsave(figure_handle, save_file, '-v7.3');
267     end
268 end
```

Bibliography

[1] M.-K. Encyclopedia, *electrocardiogram*, November 2003.

[2] A. A. Flower, J. R. Moorman, D. E. Lake, and J. B. Delos, *Periodic heart rate decelerations in premature infants*, Experimental Biology and Medicine, 235 (2010), pp. 531–538.

[3] D. Freedman and P. Diaconis, *On the histogram as a density estimator:l2 theory*, Wahrscheinlichkeitstheorie verw Gebiete, 57 (1980), pp. 453–476.

[4] M. P. e. a. Griffin, *Abnormal heart rate characteristics preceding neonatal sepsis and sepsis-like illness*, Pediatric Research, 53 (2003), pp. 920–926.

[5] B. P. e. a. Kovatchev, *Sample asymmetry analysis of heart rate characteristics with application to neonatal sepsis and systemic inflammatory response syndrome*, Pediatric Research, 54 (2003), pp. 892–898.

[6] J. R. e. a. Moorman, *Mortality reduction by heart rate characteristic monitoring in very low birth weight neonates: a randomized trial*, The Journal of pediatrics, 159 (2011), pp. 900–906.

[7] J. S. Richman and J. R. Moorman, *Physiological time-series analysis using approximate entropy and sample entropy*, American Journal of Physiology-Heart and Circulatory Physiology, 278 (2000), pp. H2039–H2049.

[8] M. C. Staff, *Diseases and conditions: Sepsis*, July 2014.

[9] A. Thompson, *The electrocardiogram - part v*, 2010.

[10] Wikipedia, *Kernel density estimation*, March 2017.

[11] ——, *Logistic regression*, April 2017.