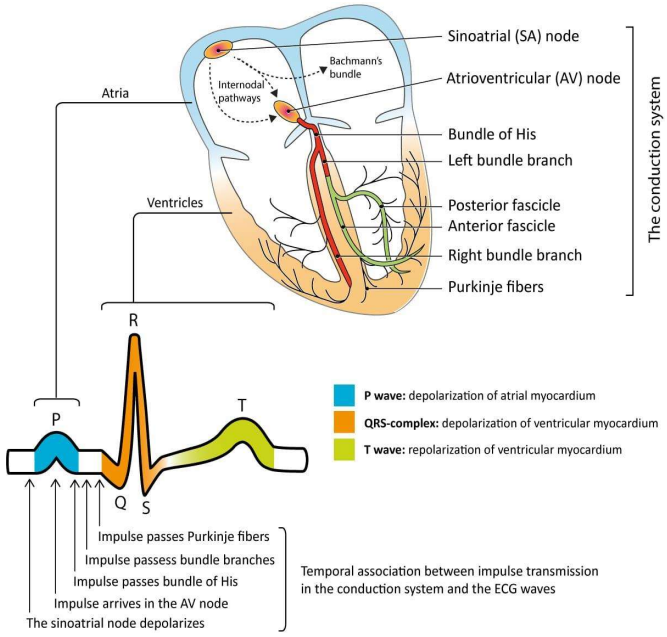
EK122: MATLAB Course Project
ECG Data Analysis

Please complete in groups of two or three.
Due Tuesday, December 10, 11:59 PM

One of the great things about learning to code is being able to use your newfound skills to solve real-world problems. In this project, we're going to ask you to process some electrocardiogram (ECG) data taken from an actual person (Christine [Mulvey]!). An ECG measures the electrical signal generated by the cells in the heart during a cardiac cycle. These specific data were taken using a three-lead ECG system under three conditions: resting, after exercise, and during *box breathing*. The objective of the experiment was to see how changing conditions changed the subject's heart rate variability (HRV). Wait, you ask, isn't heart rate just a number? Not quite. Our heart rate actually changes subtly from beat to beat, and we can measure it by finding each interval between heartbeats. You'll learn more about how to extract information like this from a continuous recording by completing this project.

First, you'll need to know a little bit about what you expect to see. The cardiac cycle involves the sequential contractions of the atria and the ventricles in the heart which are triggered by action potentials in the myocardial heart cells. The combined electrical activity of the myocardial cells produces electrical currents that spread through ions in the body's fluids. These currents are large and detectable by recording through electrodes placed on the skin. The regular pattern of signals produced by the heart is called the electrocardiogram or ECG.



The components of the ECG are correlated with electrical activity in the atria and ventricles such that:

- Atrial depolarization produces the P wave.
- Atrial repolarization and ventricular depolarization produce the QRS complex.
- Ventricular repolarization produces the T wave.

The depolarization of the myocardial cells in the ventricle causes the ventricles to contract and force blood into the major arteries of the circulatory system in a pulsatile manner.

Create a script in MATLAB to do the following:

Hint: If you want to create sections for each of these headings or for individual plots, create a section break with a double % like this

`%% Section Break` (then you can run only a single section of code at a time, just like with Colab!)

Step 1. Load our data!

In a the Course Project folder on Blackboard you will find three data files: **`Resting.mat`**, **`Exercise.mat`**, and **`BoxBreathing.mat`**. Each filename corresponds to the condition that it was collected under. The data itself consists of a three column array. The **first column is the time vector in s**, the **second column is the recorded ECG in mV**, and the last column is a calculation of the instantaneous heart rate based on the ECG. NOTE: This calculation is done by the software and may not be accurate, so we're not going to use it!

1) Load the file using the **`load()`** command. Make sure that your data files are either in your working directory, or you include the complete filepath.

2) The exported **`.mat`** files contain a variable called **`b1`** containing your time, ECG, and HR data in columns 1 through 3, respectively. Use `help` followed by the function name if you need assistance with any MATLAB function. You'll only need the first two columns.

3) Repeat this process for the other two files. If you try loading them before extracting your data from **`b1`**, they will overwrite your previous files because they contain the same variable names. Oops!

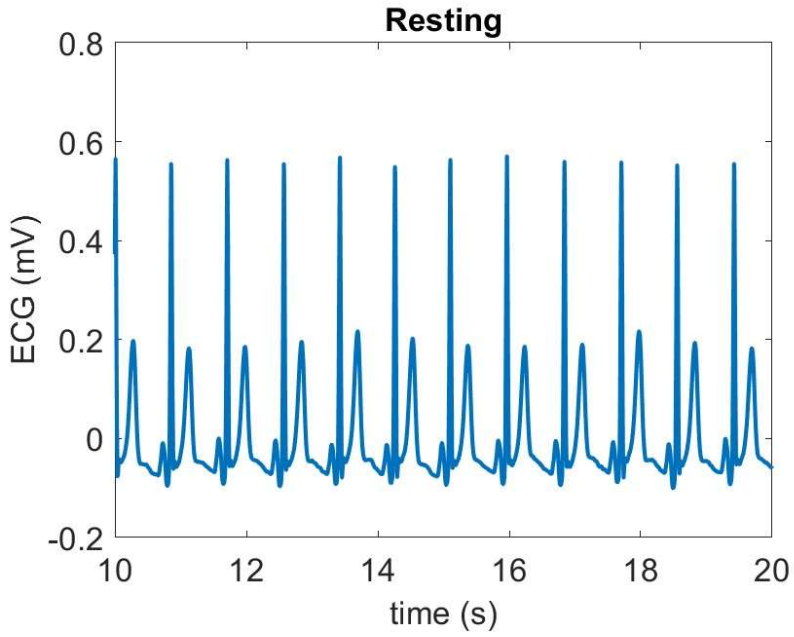
Step 2: Plot our Signals!

Before we analyze anything, we first want to visualize our data to get an idea of what it looks like. This allows us to make sure that the quality is good enough to analyze (or if we should go back and collect more before we waste time on analysis) and to see what features of our data we might exploit for that analysis.

4) Plot 10 seconds of the resting ECG. Look for 10 seconds with a stable baseline. Set the y-scale so you can see the shape of your ECG waveforms easily.

5) Assume you're going to want this figure later and you're going to want to put it in something you're going to put in another document. Let's make sure it's formatted appropriately to put in another document. That is, let's make sure that it has axis labels(use **`xlim()`** and **`ylim()`**), the lines are thick (the **`'LineWidth'`** parameter in **`plot()`**), and the fonts are large enough that if we shrink it down and put it in a report, people can still read it (**`fontsize()`** of 16). (You can't believe how often that isn't the case!) Then we'll save it.

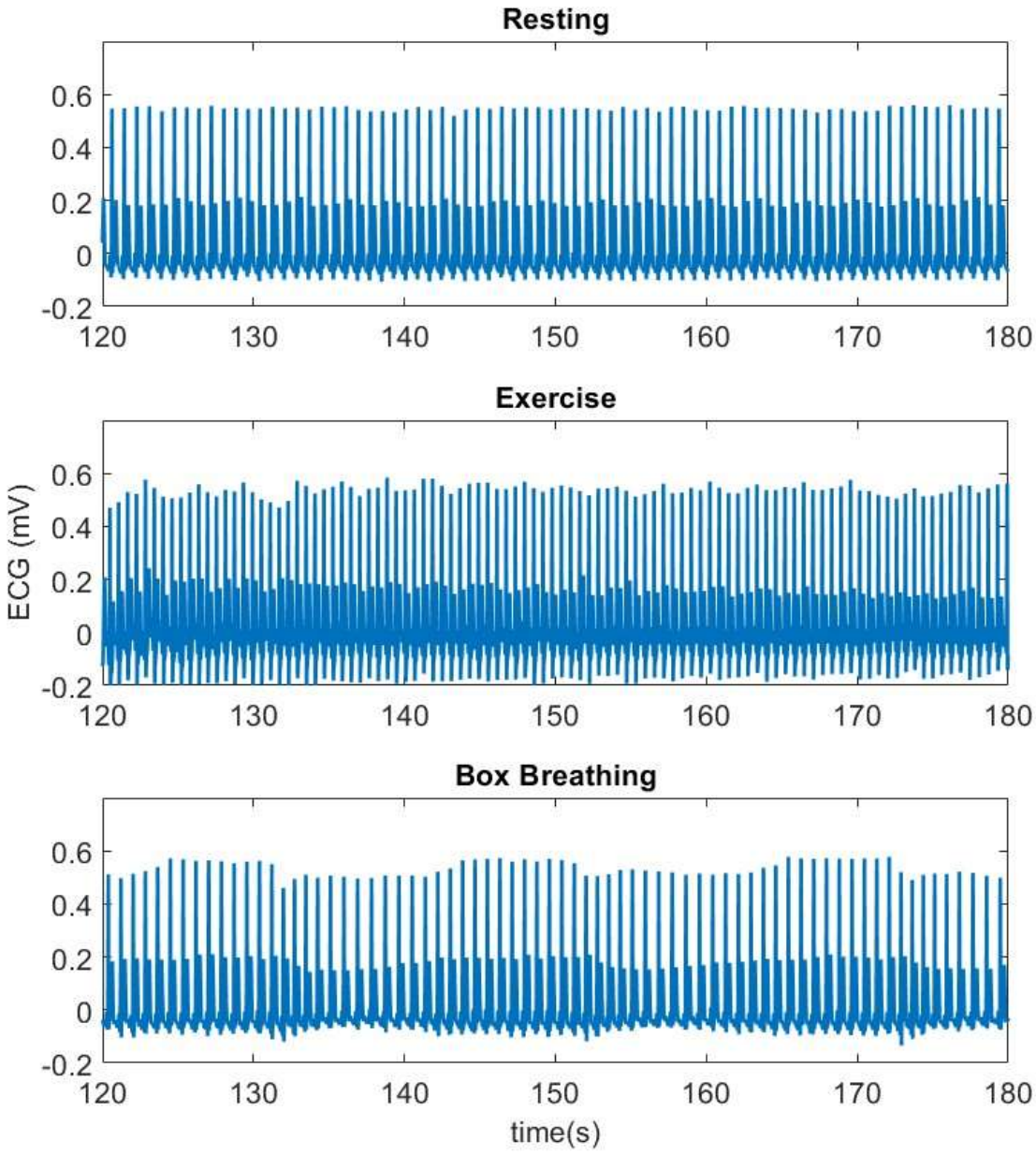
You should get something that looks like this:



Yes, that might have looked silly on your screen when you generated it, but you can still make that a little smaller and it will still look fine when printed.

6) Plot your other two files worth of data, too.

7) We can even put all three sets of data together in the same figure using **`subplot()`**. **`Subplot(m,n,k)`** makes a grid of `m x n` grid of plots where `k` is the current plot. Try plotting 60 seconds of all three conditions on a 3 x 1 subplot. See if you can find data that has a relatively even baseline for each condition. You'll see why in a minute.



Ok. Just by plotting our data there are some things we can tell about it right away. First, looking at the 10-second plot, there is a clear pattern to each cardiac cycle, so we can use that to our advantage. The QRS-complex (that sharp peak that you see) is a nice distinct feature that is easy to pick out.

Also, looking at our three conditions, it is clear that heart rate speeds up with exercise. Variability is harder to assess at this scale, but if you look closely you can see some variation in the intervals during box breathing that you can't see in the resting and exercise cases.

3. Step 3: Feature Extraction

So, we're going to try to quantify the duration of each cardiac cycle and the peak of the QRS complex is the peak we're going to try to locate. How are we going to go about that? While visually it's quite straightforward, from a signal processing perspective, peak detection isn't quite as simple as you might think. But this is one of the great things about MATLAB. Yeah, there's a function for that. You'll find the `findpeaks()` function in the Signal Processing Toolbox. If you are using a local Matlab install and haven't installed the Signal Processing Toolbox, do that now. I'll wait.

Home tab → Add-Ons button → Search for Signal Processing Toolbox

8) Look at the documentation for **`findpeaks()`**. There's some essential information in there. First, there are a few different ways that you can use it. There are definitely instances in which you would want the actual values of the peaks themselves when they occur depending on the data, or you might want the locations where they happen, so you can get both pieces of information. Which of these values do we want?

   ```
   [PKS,LOCS] = findpeaks(Y) also returns the indices LOCS at which
   the peaks occur.
   ```

   Looking further down, there are also a set of parameters that you can pass to findpeaks in addition to your dataset, Y.

   ```
   [...] = findpeaks(...,'MinPeakHeight',MPH) finds only those peaks
   that are greater than the minimum peak height, MPH.

   [...] = findpeaks(...,'MinPeakProminence',MPP) finds peaks
   guaranteed to have a vertical drop of more than MPP from the peak
   on both sides without encountering either the end of the signal
   or a larger intervening peak.

   [...] = findpeaks(...,'MinPeakDistance',MPD) finds peaks
   separated by more than the minimum peak distance, MPD.
   ```

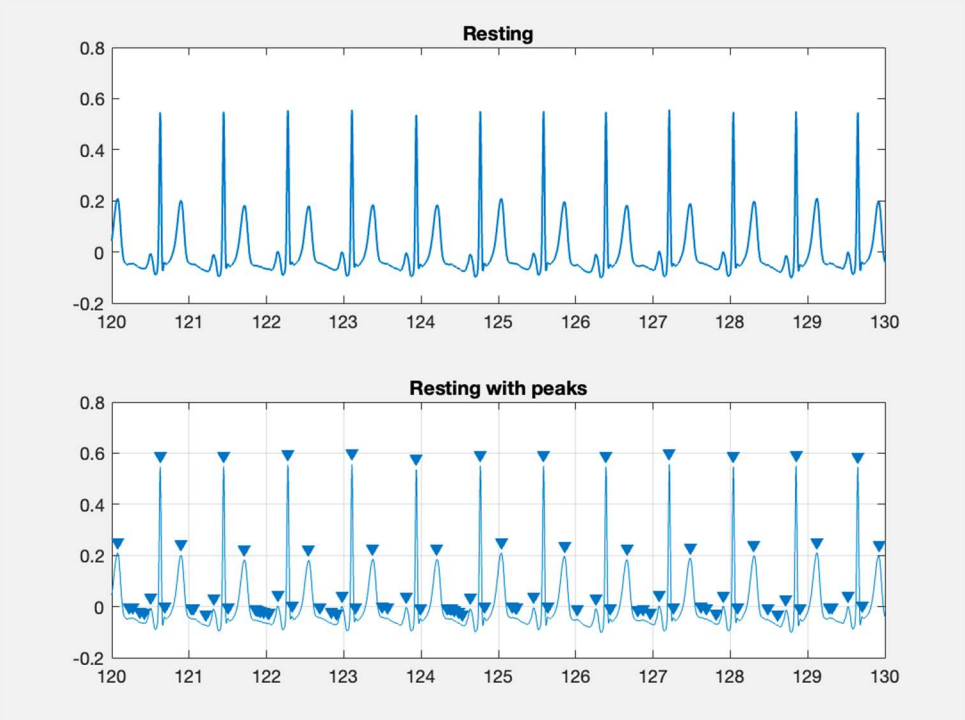
   These all seem like parameters that could be relevant to our data set based on what we know about it. If the baseline of our ECG signal doesn't vary too much we might be able to choose a threshold voltage with `MinPeakHeight` that is below the top of R but above the T wave. Even if the baseline varies, we might still get away with using `MinPeakProminence` since it measures a drop not an absolute amplitude. In this dataset, there is a pretty big gap between R and T, and the baseline is pretty even, so MinPeakHeight will likely work pretty well.

   But some people have very large T-waves compared to their QRS-complex. What happens for them? Does this analysis fall apart completely? `MinPeakDistance` to the rescue. We can give `findpeaks()` more than one parameter. So, I could say, find peaks that are greater than 0.5 mV but I know that a heart rate can't be above, say 200 bpm, so the minimum interval between beats must be 300 ms. (`MinPeakDistance` is in units of indices, not time, so you have to multiply the time

interval by the sampling rate.) This way, while the T wave may cross the threshold, because it occurs too close to the R wave, it won't be counted.

Step 4: Use `findpeaks()` to plot the peaks, using default settings

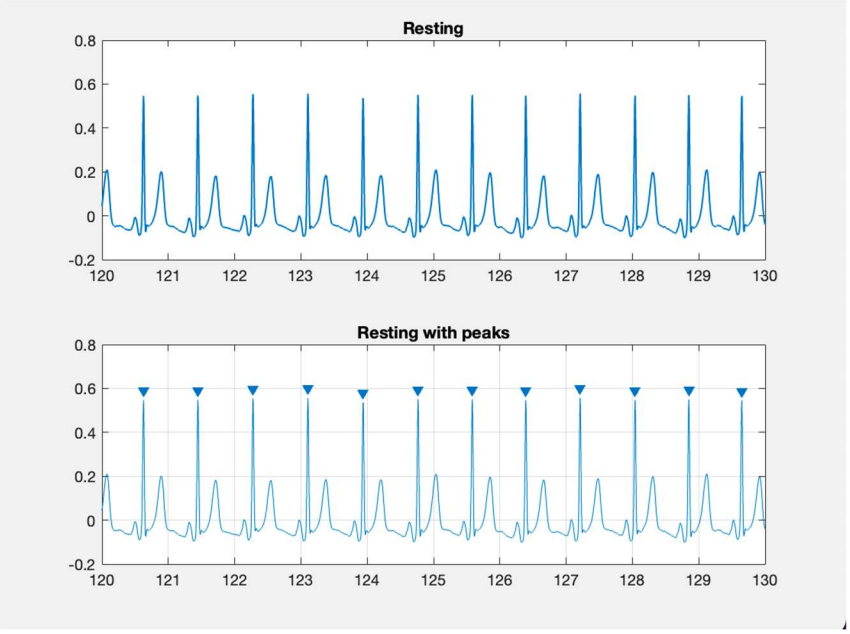
Using the `findpeaks()` (and the associated documentation as necessary) plot the resting and the exercise data on two separate plots, but use the default settings of `findpeaks()`, which will label the R, T, and P waves (oops!). On each plot, include a subplot of the original data, plotted over an appropriate interval so you can see each cardiac cycle and you can also see a few of them in a row, and also a subplot of that same time interval with the peaks indicated. Your plots should look something like this:



Do this for both resting and exercise data. Using this plot, pick an appropriate value for `'MinPeakHeight'` so that we only capture the R peak.

Step 5: Redo Step 4 with the appropriate settings of `findpeaks`

Redo the above analysis, and generate plots with only the R peak labeled. Your plots should look something like this:



Step 6: A more advanced view

Now that we've found the way forward to do signal analysis, and can easily identify individual heartbeats, we are interested in *qualitatively* estimating how regular the heartbeat is. For this, you will want to plot the heartbeat such that it is always centered about the *trigger*.

Trigger is a word you might be familiar with in case you've spent some time with oscilloscopes. (If you haven't yet, you will later on in your engineering studies.) It means the point of time at which we consider the signal to have the sought-after shape. Typically, it's when the signal goes above a certain threshold, e.g. a horse's nose crossing the finish line, although you can certainly have far more complex triggers, e.g. you only win an escape room when all puzzles have been solved.

In our case, the trigger is the peak that we've detected in the prior examples. So this is easy, because we've already found it!

With the trigger in hand, we'll want to plot each heartbeat **on the same graph**. But because what we're looking for is the similarity between heartbeats, we don't want to plot with a wide, heavy line. It's much better to use a thin, semi-transparent line.

Transparency in Matlab is straightforward to do when plotting lines. It only requires setting the 4th value in an RGBA color spec. This looks like

```
x = 0:.1:2*pi;
y = sin(x);
plot(x, y, 'Color', [1 0 0 .5])
```

The above code will plot a sine wave in web red (because the first value is 1), and with 50% transparency.

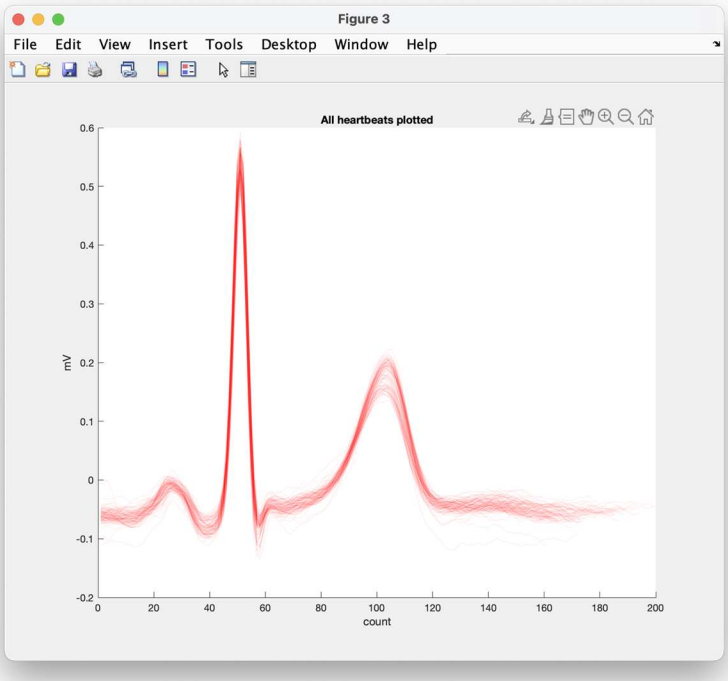
The below code would do web green (because it's 1 in the green channel):

```
plot(x, y, 'Color', [0 1 0 .5])
```

Lastly, the below code would do web blue (because it's 1 in the blue channel):

```
plot(x, y, 'Color', [0 0 1 .5])
```

So if we use the correct series of **`plot`** commands, we can plot all the heartbeats and it looks something like the following



Notice how the amount of blurriness indicates how regular the ECG reading was at that point of the cycle. In the above example, we can see that the ECG is extremely consistent around the R wave, but less consistent around the T wave.

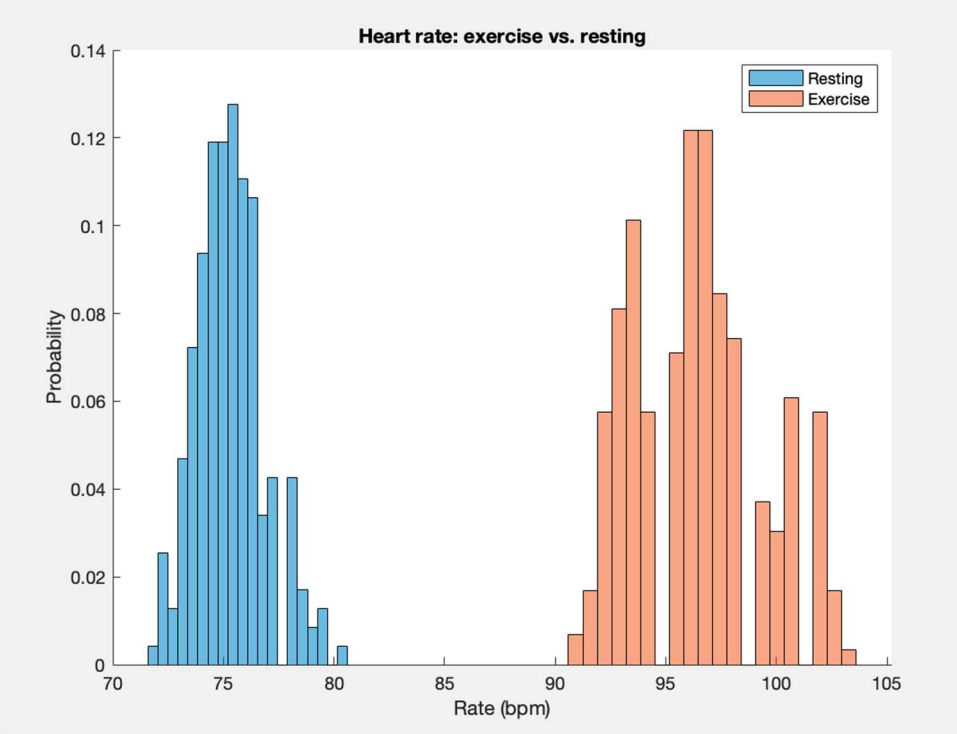
Your job is to **plot something like this for each of the three datasets**.

Note that you don't have to get exactly the same result, and it doesn't have to be the same transparency, but it should be obvious when we're looking at a regular vs. an irregular heartbeat. And as you might imagine, the three different data sets will have different regularities, with irregularity increasing from resting to box breathing to exercise.

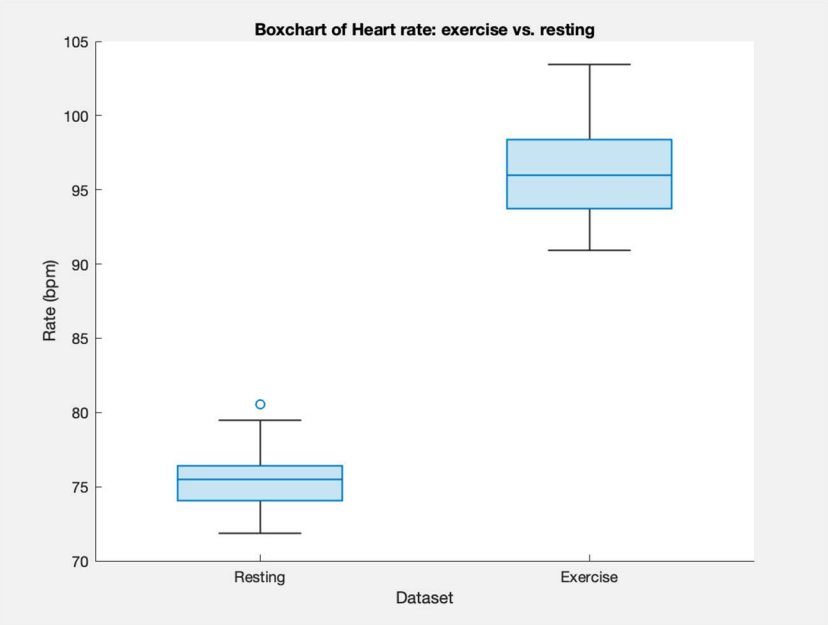
Step 7: Plot a histogram using `histogram` and a boxplot using `boxchart`

Now that we have our peaks identified, we are in a position to do some analysis! First we have to do a bit of additional processing

- Using logical indexing, extract the times of the R peaks within a limited time window, say from 120 seconds to 130 seconds (the time window we plotted above) or perhaps a bit longer than that. We just want to limit the analysis we are doing so we don't analyze the entire recording session. Show your code for doing this!
- Using both a numerical operation on the array (i.e. by hand!) and using the built in matlab function `diff` (use documentation as necessary!) compute the interval between R peaks.
- Using this interval compute the frequency of these events, i.e. our subject's heart rate!
- Using `histogram` plot a histogram of this frequency for all the events in the limited window you are analyzing. Label your axes and each histogram. Use the documentation about `histogram` as necessary. Your plot should look like this:



Another very handy plot is called a *boxchart*. Using the `boxchart` function, generate a plot like the following:

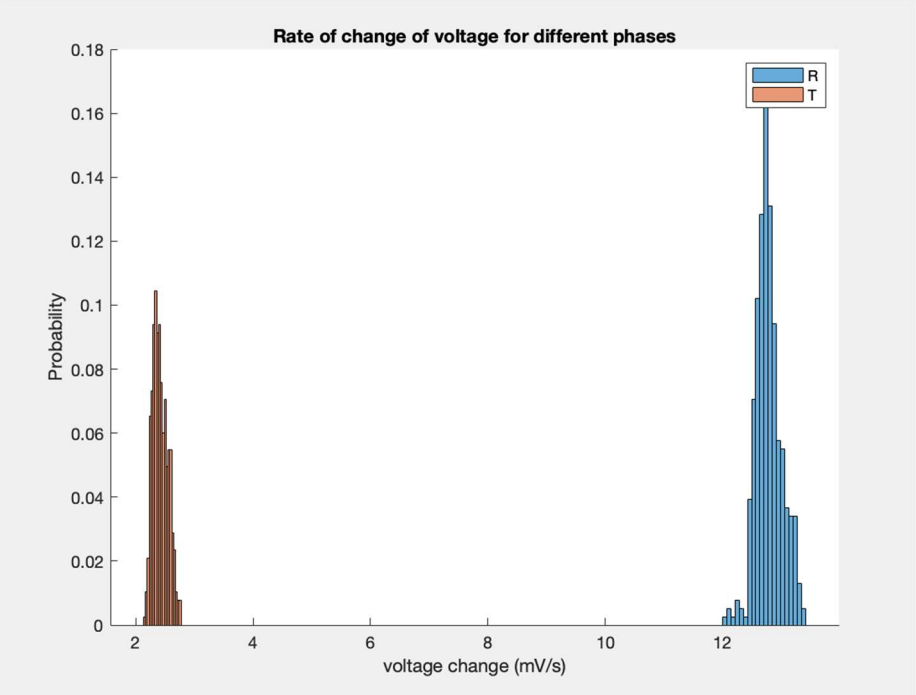


Use documentation as needed. One thing to recognize is that you will need to combine the data you want to plot into one long vector, and associate with it another vector that labels each data point as being from dataset 1 (resting) or dataset 2 (exercise), i.e. your plot call will look something like this `boxchart(group, combinedData)` where `group` is a vector the same length as your combined data and labels each datapoint.

It should be quite clear from these plots that when you are resting and exercising, your heart rate is different! (Hopefully no surprises there!)

Step 8: Compute the derivative of different phases of the signal

Visually examining the R and T components of the signal in the resting data during Step 4, it was clear that the R pulse changes much more quickly than the T pulse. Using a `for` loop, examine data around each R pulse and each T pulse in the resting data. Compute the absolute value of the change in voltage for 10 samples before and after each R and T pulse, and use that to compute the speed of each pulse. Plot a histogram of the speed of all the pulses. Your plot should look like this



No surprise – the speed for the R pulse is quite a bit faster than for the T pulse!

Step 9: Run a two-sided t-test using `ttest2`

A typical last step when performing data analysis like we are doing is to do some statistics on the data. Matlab has lots of built-in functions for that! One of the most basic analyses we can do is called a 't-test'. (The reasons for this name are somewhat complicated, but for the curious see here: LINK). In brief, and somewhat simplifying the notion, a t-test will compare a dataset against a specified value (an average) and determine if that dataset is consistent with that average value. Alternatively, we can compare two datasets and compare them against each other to see if they are similar (statistically). This latter test is called a 'two-sample t-test' (because we have two datasets, or samples).

Matlab has a nice built-in function to run a two-sample t-test, called `ttest2`. Take a peak at this function's documentation to see how it works.

Understanding a t-test in its entirety is way beyond the scope of this course, but at its core a t-test boils down to a simple decision: is there enough evidence to support the idea that our two data samples are different (in technical terms this is called 'rejecting the null hypothesis'), or must we instead claim that there is simple not enough evidence to make such a claim (technically, lacking any evidence that they are different we adopt a 'null hypothesis' that they are *not different*). (Notice, in the second case, we are not saying they are the same, we are just saying that we cannot claim they are different. This tiresome legalese has a long history, but is important to underscore).

From our above plots, it should be pretty clear that our heart rate during rest and exercise seem pretty different, and there should be strong evidence to suggest that they are different. Let's run a t-test to make this claim statistically!

The output of the `ttest2` function is a 1 or a 0, 1 if there is enough evidence to claim the two datasets are different ('reject the null hypothesis') and 0 if we cannot claim this ('cannot reject the null hypothesis').

Using a for-loop and any conditional statements (e.g. if else, etc.), execute 2 t-tests, as described below:

- Run a t-test on the heart rate frequency data during rest and exercise and compute the output of the test, if we can or cannot reject the null hypothesis. Print the output!
- Run a t-test on the **heart rate frequency during two separate periods of the rest data** and compute the output of the test, if we can or cannot reject the null hypothesis. Print the output!

CONGRATULATIONS! YOU'VE JUST RUN YOUR FULL DATA ANALYSIS PIPELINE, SOUP TO NUTS!