MathWorks Hackathon

Fitness Tracker Challenge

Instructions for Getting Started

This guide has detailed steps to help your team use MATLAB Mobile and MATLAB Online to create your fitness tracker. We have also included a simple example that you can use to start with or draw inspiration from as you create your own solution to this challenge.

Good luck!

Table of Contents

Getting Started

Challenge Statement

Fitness trackers are a relatively new and exciting technology that are used to track fitness data as you live your everyday life. The technology used in these types of devices are low tech and can be recreated from home. In this challenge you will use MATLAB and MATLAB mobile to make your own fitness tracker.

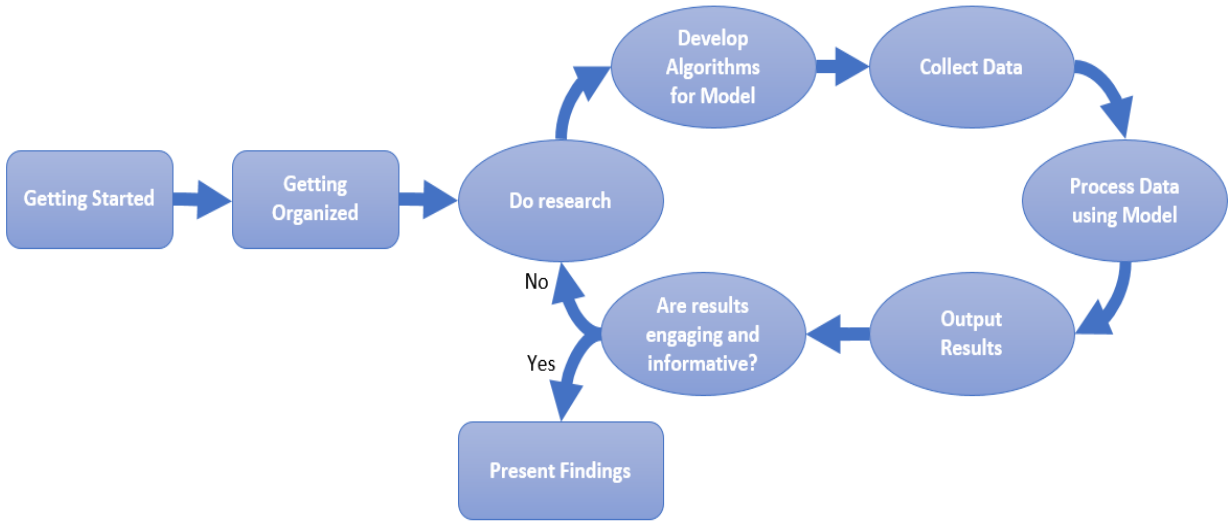
Using sensor data retrieved from your phone, the goal is to create a model to turn this data into useable results to inform someone about how effective their workout was. This output data could include any sort of fitness data such as calories burned, steps taken, or flights climbed. Your task is not only to figure out what information you want to output but also how to make a model to output this information. Participants who incorporate machine or deep learning techniques into their models will score higher, but this is not required. Once you have a model and results, you will need to present your findings in an easy-to-understand manner.

The Process

Make sure to read though this entire documentation to fully understand the competition. Once you finish reading this document as well as getting set up, you will enter a cycle of gathering and processing data. It is very important to remember that this is a process that will be iterative, and you will have to go back many times until you have the model that you want. Below is a flowchart of how you should proceed with this competition.

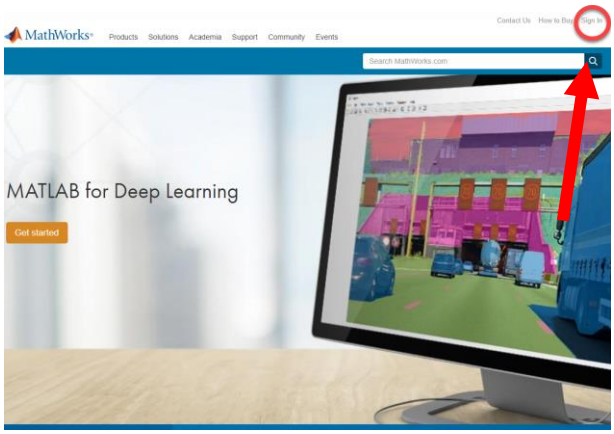
Now that you have a sense of the challenge, let's dive into it!

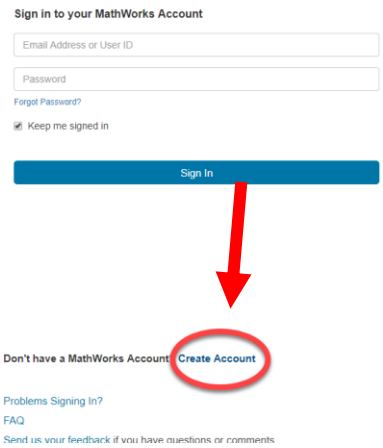
Competition Process



Getting Started with MATLAB

1. Create a MathWorks account
    - Go to https://www.mathworks.com/
    - Click 'Sign In' in the upper right corner of the page
    - Click 'Create Account '
    - Fill out page and then press 'Create'





2. Access MATLAB Online
    - Go to https://matlab.mathworks.com/
    - Log in Using MathWorks account created earlier
3. Download MATLAB Mobile
    - App is available for both Android and IOS
    - Go to Google Play Store or App Store and search "MATLAB Mobile"
    - Install app on phone
    - Open app and press 'Connect to MathWorks Cloud'
    - Log in using MathWorks account created earlier

Note: You cannot be signed into both MATLAB Mobile and MATLAB Online at the same time.

Getting Organized

The Roles

Your team does not need to be divided into roles, but we suggest that this is done to be more organized. It is also important to note that while we define each role below, that does not mean that each person should do one job alone. When working on this challenge, it is vital that everyone works together and that the work is spread out across the team.

Programmer

As the programmer, your role will be to make and test the model. It is important that the programmer is someone who feels more comfortable using computers and new software. It will be your job to convert all the sensor data and ensure that the results that you obtain are meaningful.

Marketing

As the marketer, your job will be to create the visuals for the results because data isn't useful unless someone can interpret them. You will determine how you want to create these visuals and work with the programmer to choose and modify graphics to present the data. Your job will also be to decide how you want to present your findings and how to make this look unique to your team.

Athlete

As the athlete, your job will be to be the voice of the consumer. Since you will be the person collecting all the data, it will be your job to determine what the most useful results are. Do you want steps taken, calories burned, or even flights climbed? Don't think that your job is done when you collect the data because your voice is vital at every step of the process. Without the consumer, there is no product!

Available Sensors

Sensors included in MATLAB mobile:

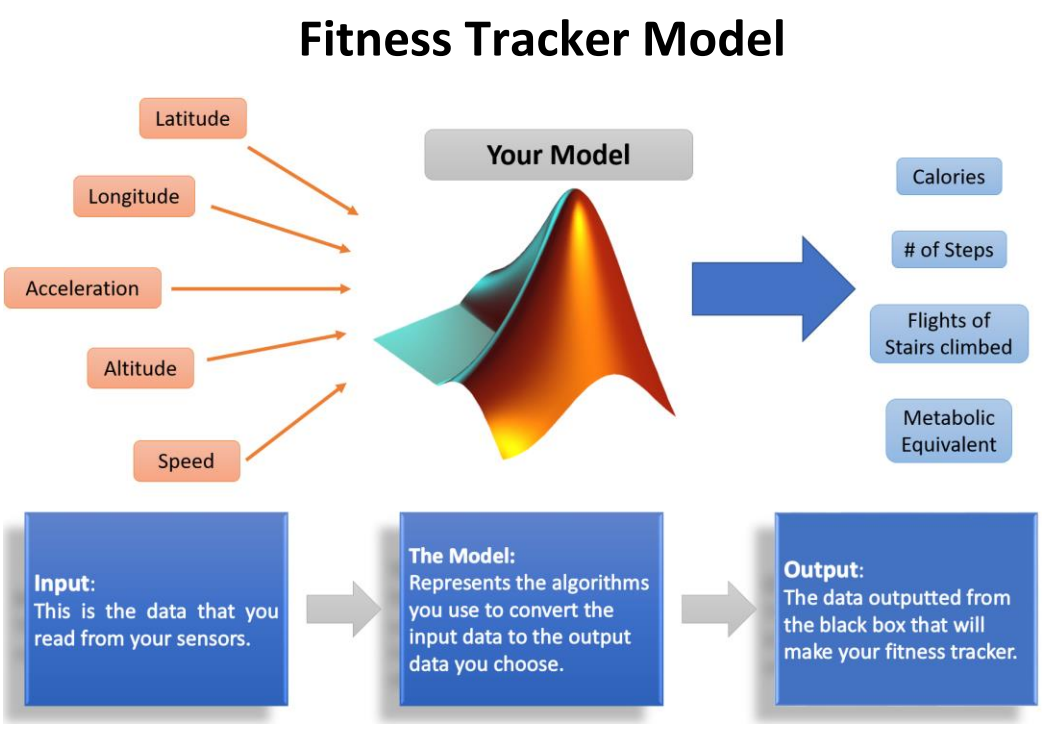
- Acceleration (m/s^2)
- GPS Position
    - Latitude (degrees)
    - Longitude (degrees)
- Speed (m/s)
- Altitude (m)
- Course/Heading (degrees)
- Horizontal Accuracy (m)
- Orientation (degrees)
- Angular Velocity (rad/s)

To find out more about these sensors check out this documentation page!

Researching Fitness Models

To turn your sensor data into meaningful results, you will need to create a model. You will input data into your model and some sort of calculations happen within it to output your desired results. The first step to creating a model is determining what kind of data you want to output. Steps? Distance traveled? Intensity of an activity?

Once you have decided what data you want to find, the next step is to figure out how to convert the data from the sensors to this output. It is important to find reasonable data to output because while something may be a better way of representing fitness data, if you can't calculate it from the sensors provided you may be wasting time.



Example Model:

As part of the documents provided for this competition, we have provided a MATLAB example that can be used as reference when developing your own fitness tracking algorithms. The example MATLAB program first processes previously acquired data in order to calculate an estimate of the number of steps taken by a person and then uses a machine learning model to classify what type of exercise a person was doing. It also shows how to access acceleration data and visualize it with some common MATLAB plots. Here is a list of the files included as part of the example:

ExampleModel.mlx

This is a MATLAB Live Script that goes through examples on how to process data acquired from a MATLAB Mobile session in order to make meaningful conclusions.

ExampleData.mat

This is a MATLAB data file containing a set of sensor data previously acquired using the workflow demonstrated in the “Collecting Data” section of this document.

ActivityLogs.mat

This is a MATLAB data file containing 4 sets of sensor data previously acquired using the workflow demonstrated in the “Collecting Data” section of this document.

timeElapsed.m

This is a helper MATLAB function used in the example model to transform an array containing the date and time of the data points collected to an array of the time elapsed since the acquisition was started.

Collecting Data

This section will be completed using the MATLAB Mobile application.

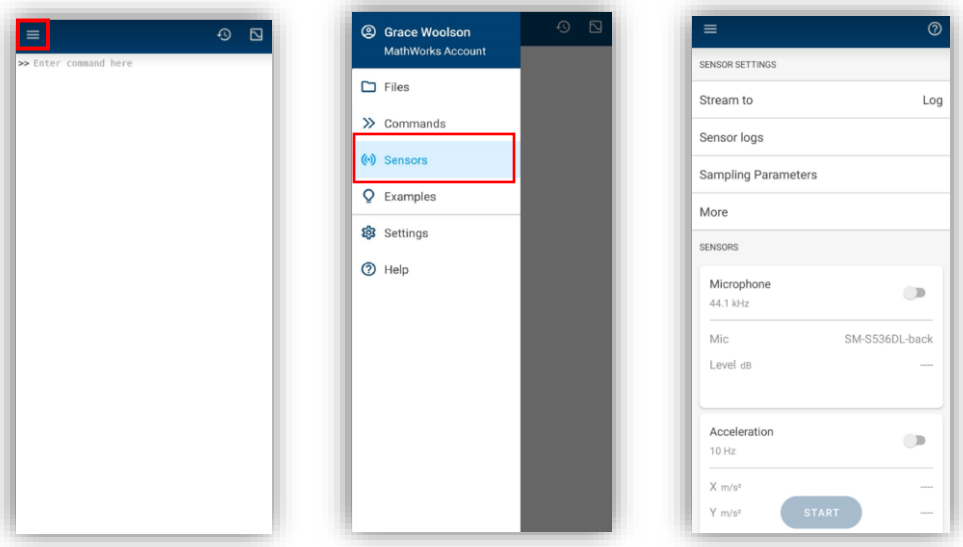
Methods of Collecting Data

Fitness trackers are used to track data everywhere someone goes. Because of this, you should be able to use your fitness model to track data in various settings. When you collect data, you may have someone running laps, but will everyone be running laps when they use your tracker? Probably not! You will want to do various activities such as walking, running, and even climbing stairs. This will ensure that you have the most diverse fitness tracker. As a team, make sure to discuss what activities you want to do to collect data and what is the most important.

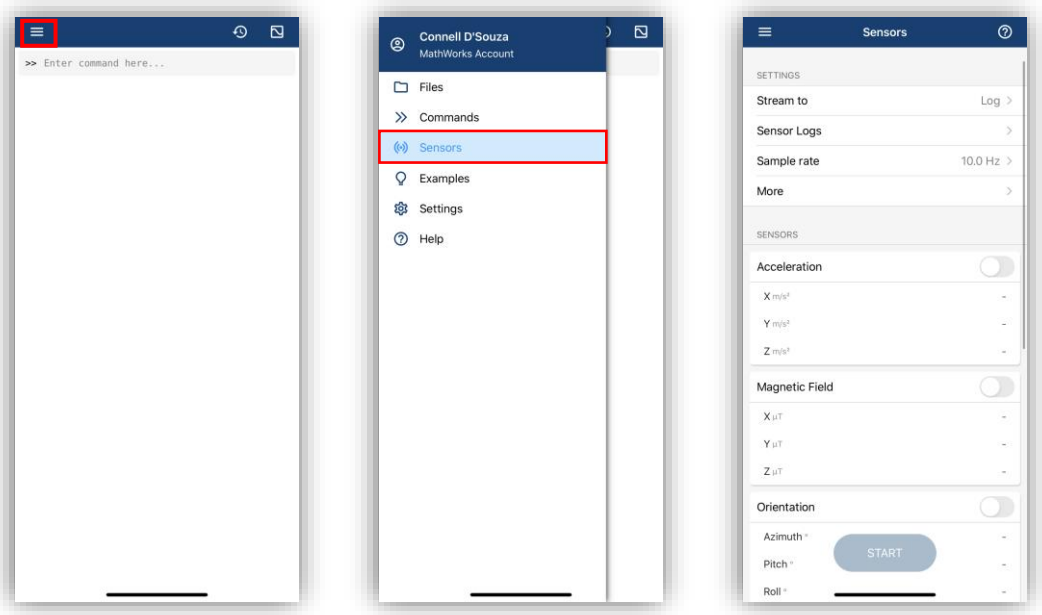
Recording Data

Once you have the MATLAB Mobile app installed on your phone, you are now ready to start recording your data. Start by opening the side menu and then press 'Sensors'. This will bring you to a screen that shows all the sensors that you will be able to use to calculate your results.

Android:



iPhone:



The next step is to make sure you record the values – this is called logging data. Before we start logging data, switch the mode from 'Stream to MATLAB' to 'Log'. Switching this option will make the app save the data to your phone and automatically upload it to MATLAB drive. To check what is stored in your drive you can check it here: https://drive.matlab.com.

To start recording data, first enable the sensors you wish to use by tapping on the toggle next to the sensor's name, then press the 'Start' button from the bottom of the sensor page. Once you are done recording data, just press the 'Stop' button. The data will be saved to a folder named 'MobileSensorData' by default, but this can be changed within the settings. This data will be saved into a .mat file, which is a file type specifically for storing MATLAB data.

Note: If you turn on the 'Acquire Data in Background' option within the Sensor Settings, you will be able to log data while your phone is locked.

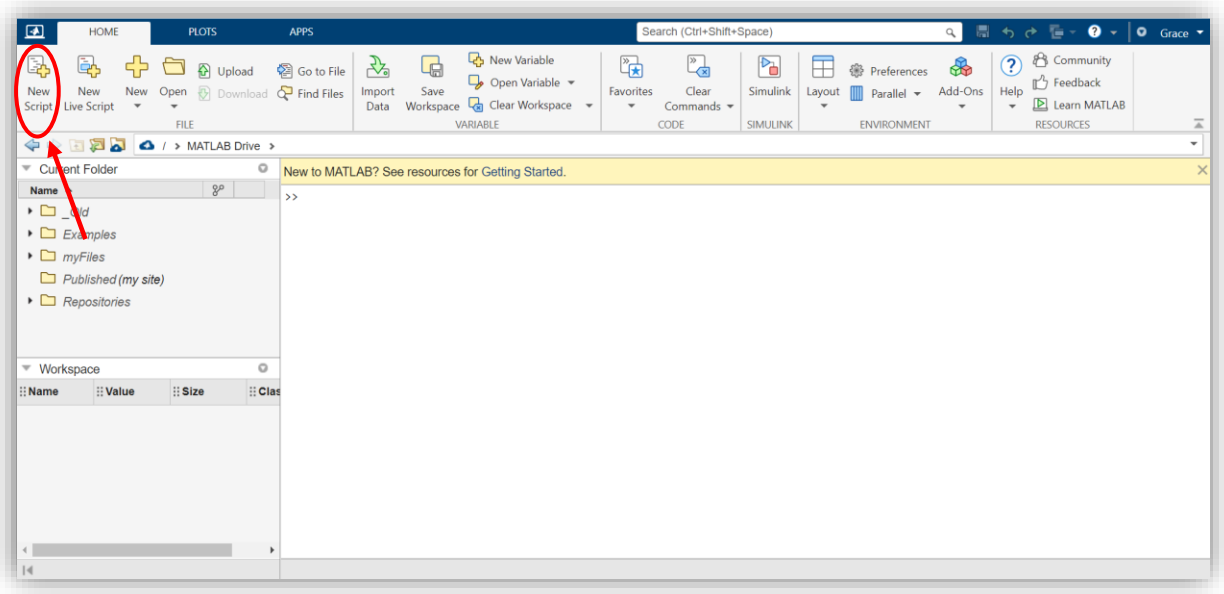
Processing Data

This part will be completed using MATLAB Online.

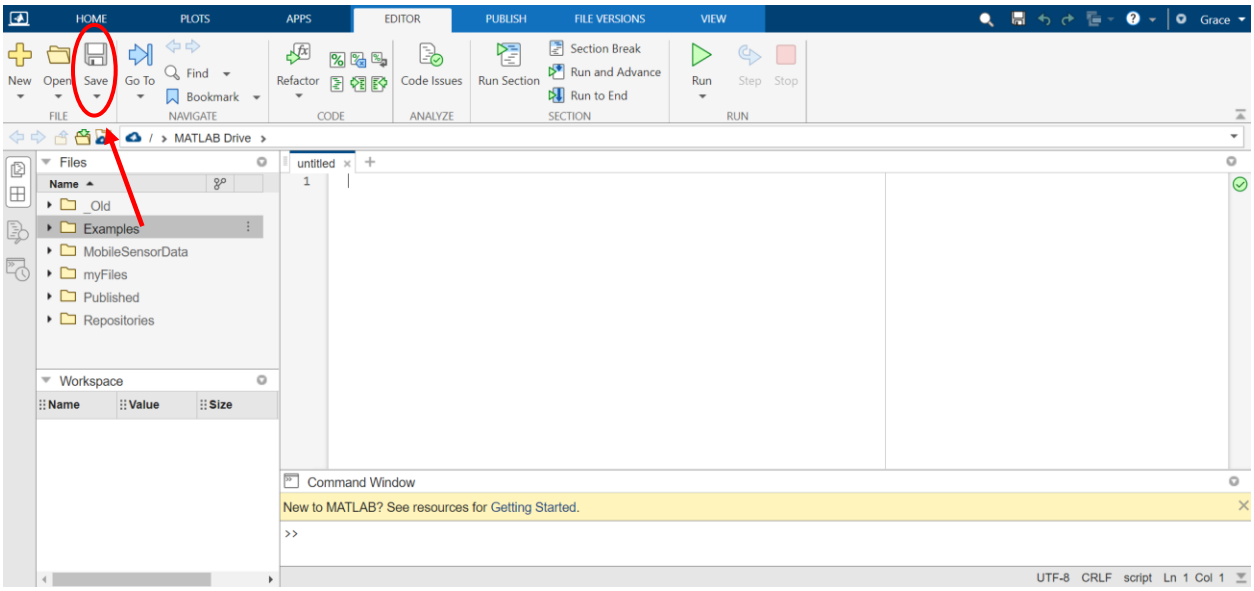
Creating a Script

When you first open MATLAB Online, you will see the window shown below. One way to execute commands is by typing them in the window with the ">>" – this is called the Command Window. The Command Window is a great tool for quickly testing individual functions, but the commands you type in here will not be saved.

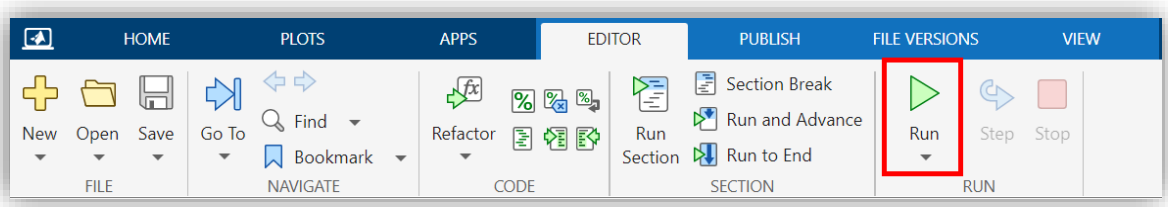
To save your model, you will need to create a script. Think of a script like a text document for code. In the script, you will be able to write and edit all your commands, and instead of running one line at a time you will be able to run the whole script at once. To create a new script, click the 'New Script' button in the top left of the screen.



This will also open a new 'Editor' tab. To save any changes you make to the script, click on the 'Save' button in the top left of the screen. A pop-up box will appear asking you to name this script.

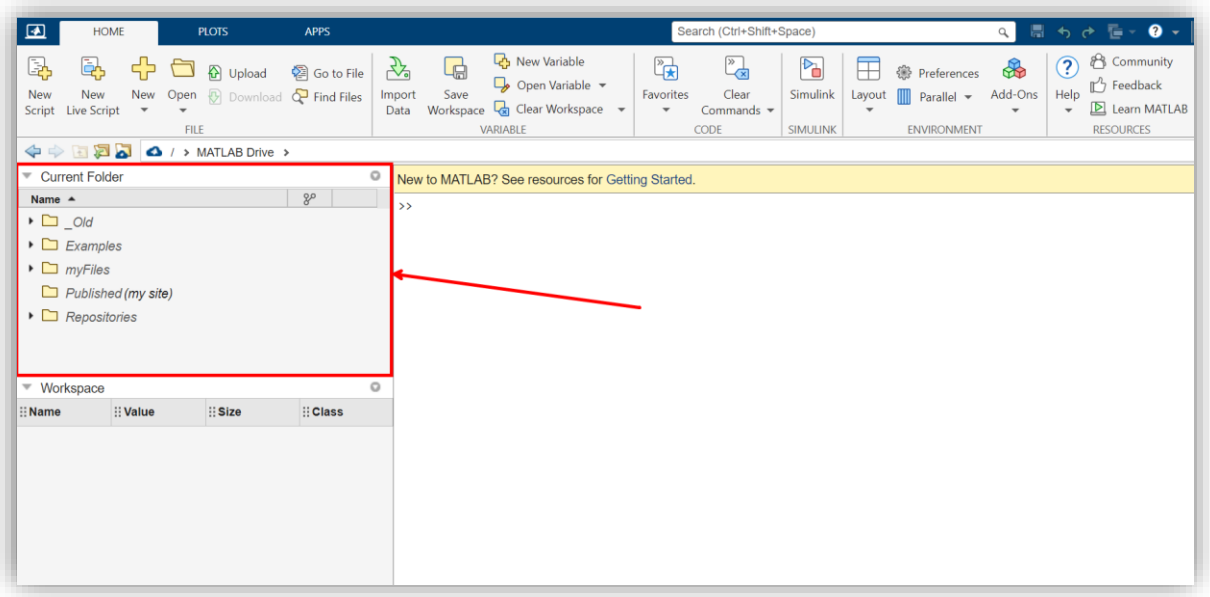


This script will be saved into your drive along with all the data that you saved. Once you have some code that you want to run, you simply press the green play button on the taskbar.

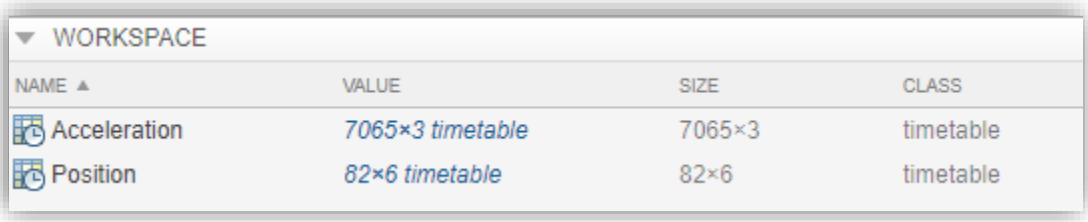


Accessing Data

When working with MATLAB Online, all of the files that are in your MATLAB Drive can be accessed from the left-hand plane labeled 'CURRENT FOLDER.'



To load the data that you have collected, double click on the .MAT file that the data is saved in (ex: 'ExampleData.mat'). In doing this, you will now see the sensor logs in the workspace.



The data is stored into a type of table called a timetable. This table is special because it allows users to be able to store data with relation to time. To see what is held within a table, we can check the 'Properties' value of the table using dot notation:

```
>> Position.Properties

ans =

  struct with fields:

           Description: ''
              UserData: []
        DimensionNames: {'Timestamp'  'Variables'}
         VariableNames: {'latitude'  'longitude'  'altitude'  'speed'  'course'  'hacc'}
   VariableDescriptions: {}
          VariableUnits: {}
     VariableContinuity: []
              RowTimes: [82×1 datetime]
```

From this, we see two important qualities held in the table. The first being the VariableNames, which are the names of the data that we collected. If we want to extract all these variables and store them into a 2D array, we can use the same dot notation.

```
posVariables = Position.Variables;
```

This might not be the most efficient way to extract the data however, because we may only want one of these variables. We can specify a specific variable in the same way just used.

```
latitudeData = Position.latitude;
```

To specify a specific range within this data we can use the same notation that we would with arrays (parenthesis) as seen below. The problem with doing this is that this will create a timetable of this specific range.

```
>> first6 = Position(1:6,:)

first6 =

  6×6 timetable

          Timestamp           latitude    longitude    altitude    speed    course    hacc
    ______________________    ________    _________    ________    _____    ______    ______

    15-Aug-2018 11:41:18.445      42.3       -71.349        35       0.74     300.1    13.936
    15-Aug-2018 11:41:19.446      42.3       -71.349        41       1.67     232.6    12.864
    15-Aug-2018 11:41:20.446      42.3       -71.349        15       0.63     232.6    11.792
    15-Aug-2018 11:41:21.445      42.3       -71.349        37       1.14      22.3     10.72
    15-Aug-2018 11:41:22.445      42.3       -71.349        28       2.05       106     9.648
    15-Aug-2018 11:41:23.445      42.3       -71.349        46        0.4       106     9.648
```

To extract data as an array we must use the curly braces:

```
>> first6 = Position{1:6,:}

first6 =

   42.2998  -71.3491   35.0000    0.7400  300.1000   13.9360
   42.2997  -71.3491   41.0000    1.6700  232.6000   12.8640
   42.2997  -71.3491   15.0000    0.6300  232.6000   11.7920
   42.2997  -71.3491   37.0000    1.1400   22.3000   10.7200
   42.2997  -71.3491   28.0000    2.0500  106.0000    9.6480
   42.2997  -71.3490   46.0000    0.4000  106.0000    9.6480
```

Note: Sometimes the data given can be long ugly numbers. The best way to adjust all the numbers in your script is by using the format command.

As mentioned earlier, the timetable has another important piece of information that is indicated in the name: the time at which each one of these data points is taken. These values are stored within a data type which stores the year, month, day, hour all the way down to the milliseconds. To extract this information, we pull the timestamp information.

```
positionTime = Position.Timestamp;
```

To plot the data against time, we will need to convert this data type. The time information that we just stored contains complete date and time information, but it may be better to plot this data against time in **seconds**. To convert the datetime array *positionTime*, we have provided a function *timeElapsed* within the competition files that will return an array of elapsed time since the first reading was taken. To use it, make sure the function file is within the same folder (or path) that your script is in. You can call this function in the following way:

```
positionTime = timeElapsed(positionTime)
```

Developing Your Model

Now that we have data, we need to do something with it. This section will take you through making a model to convert GPS data to steps taken. Remember **this is an example** and should only be used to help guide you to making your own model. And while this is how we calculate steps taken, there are many other ways to do the same thing and we encourage that you try other ways to solve this same problem.

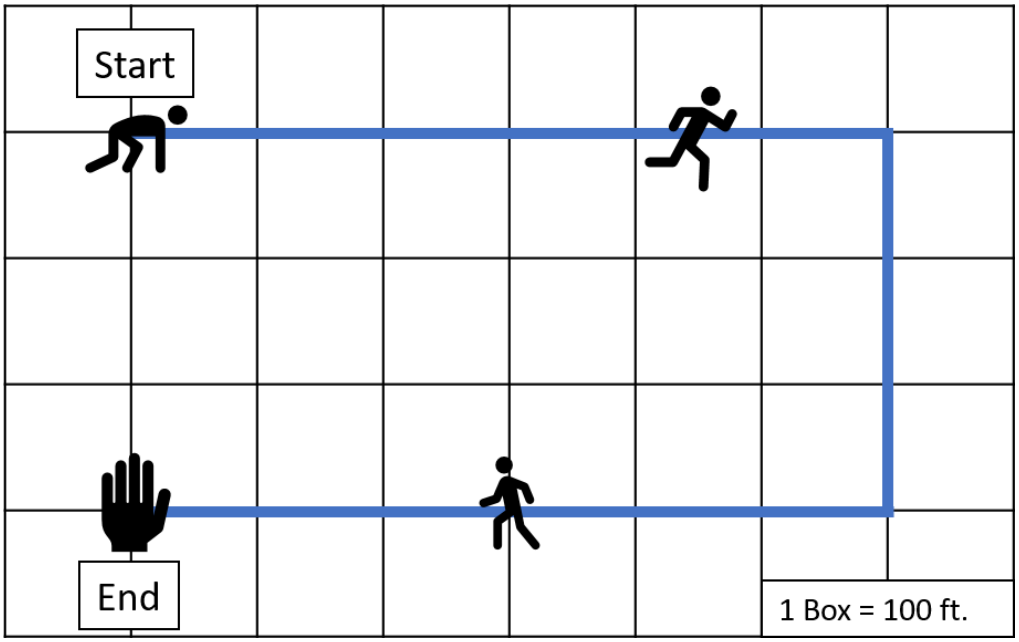
Example: Counting Steps

Let's say that you wanted to track the number of steps taken. One way that this can be done is by calculating the total distance traveled and then dividing it by your stride length. This

works because stride is the distance taken for one step, so if we look at the units they cancel out as seen below.

$$\frac{totalDistance(ft)}{stride\ (\frac{ft}{step})} = \frac{ft}{ft/step} = ft * \frac{step}{ft} = steps$$

Now that we have an equation, the next step is to solve it in MATLAB. The first step is to get the total distance traveled, which can be done using data from your GPS location or the latitude and longitude sensor data. One way that we can solve this is by looking at your start location and end location and calculating the distance between those points. If we did that with the figure below, however, the distance traveled would be 300ft because the start and end position is 3 squares apart. We can see that the person traveled much further than this however, so we must calculate the distance along the path using each data point. In doing this, we can see that the person below actually traveled 1500ft, which is a large difference from the 300ft that we originally calculated.



Now it's time to solve this in MATLAB. In this example, latitude and longitude have been stored into variable names *lat* and *lon* respectively. The first step is to declare a couple of variables. The first will be the circumference of the earth in miles. The next variable is the total distance traveled which will be set to zero for now and explained why later.

```
earthCirc = 24901;

totaldis = 0;
```

To solve this problem, calculations will have to be repeated several times in a loop. Because of this, we use a function called a for loop which will repeat **for** a specified number of

times. In this case, we want to have it run until the end of our vector of data. The *length* function can be used to determine the number of elements in a vector and therefore how many times we need to run the loop. In this method, both one position and the next position are used at the same time, so we need to stop one before the end of the vector as seen below.

```
for i = 1: (length(lat)-1)
```

Now we save the first position and the second position into variables. We will do this using the variable *i* from above. This variable is a vector from 1 to the end of our latitude vector which can be used to index the latitude and longitude vectors.

```
lat1 = lat(i);     %This represents the first latitude

lat2 = lat(i+1);   %This will be the second latitude

lon1 = lon(i);

lon2 = lon(i+1);
```

Thankfully for us MATLAB has a command to solve for distance given latitude and longitude. This will give us the difference in degrees which we will then turn into length by creating equivalent fractions as such.

$$\frac{traveledDistance}{Earth'sCircumference} = \frac{degreesTraveled}{earth'sDegrees}$$

From this we solve for *traveledDistance*. We will use 360 for *earth'sDegrees* because that is the total number of degrees in a circle. While the earth is not a perfect circle, this will still be a good estimation

```
diff = distance(lat1, lon1, lat2, lon2);

dis = (diff/360)*earthCirc;
```

Now we add up the distance and store it into the total distance variable. What this code will do is take the previous total distance and add the distance just calculated onto it. We declare the variable before the loop so that we could use it on the first iteration of the loop. With this we've calculated everything that we needed to within the for loop and must close the loop by using the *end* command.

```
totaldis = totaldis +dis;

end
```

From here, we need to divide the total distance traveled by the stride to get the steps taken. For this example, we will use the average stride of 2.5ft. Since the total distance traveled is in miles, we will have to convert this into feet.

```
stride = 2.5;

totaldis_ft = totaldis*5280;

steps = totaldis_ft/stride;
```

With that we have created a fitness tracker model that counts steps! From here, you should have all the information you need to create your own model!

Example: Classifying Activity using Machine Learning

In addition to counting steps, let's say that you want to know what kind of activity you were doing. Let's train a machine learning model that takes our acceleration data and shows if we were sitting, walking, or running.

First, you'll need to collect sensor data for each of these activities. For simplicity, I collected this data in three different sessions: one where I was sitting the whole time, one where I was walking, and one where I was running. Let's load the logs for these sessions and see what they look like:

```
load ActivityLogs

head(sitAcceleration)
```

Now, let's add the activity labels to each table. I'll do this by creating a label, and then using the 'repmat' function to multiply that label, creating a column that has as many rows as the table we want to add the label to. Then we add the column to the data table.

```
sitLabel = 'sitting';

sitLabel = repmat(sitLabel, size(sitAcceleration, 1), 1);

sitAcceleration.Activity = sitLabel

% Repeat for walking data

walkLabel = 'walking';

walkLabel = repmat(walkLabel, size(walkAcceleration, 1), 1);

walkAcceleration.Activity = walkLabel;

% Repeat for running data

runLabel = 'running';

runLabel = repmat(runLabel, size(runAcceleration, 1), 1);

runAcceleration.Activity = runLabel;
```

Now we can combine all of our data into one table, using the brackets to combine the data and using semicolons to indicate we want to stack the data vertically.

```
allAcceleration = [sitAcceleration; walkAcceleration;
runAcceleration];
```

Last, we can remove the 'Timestamp' variable, since this variable does not help determine what type of activity we are doing. I use the ‘timetable2table’ function to convert the data to a regular table, and set ‘ConvertRowTimes’ to false to indicate that we don’t need the timestamps.

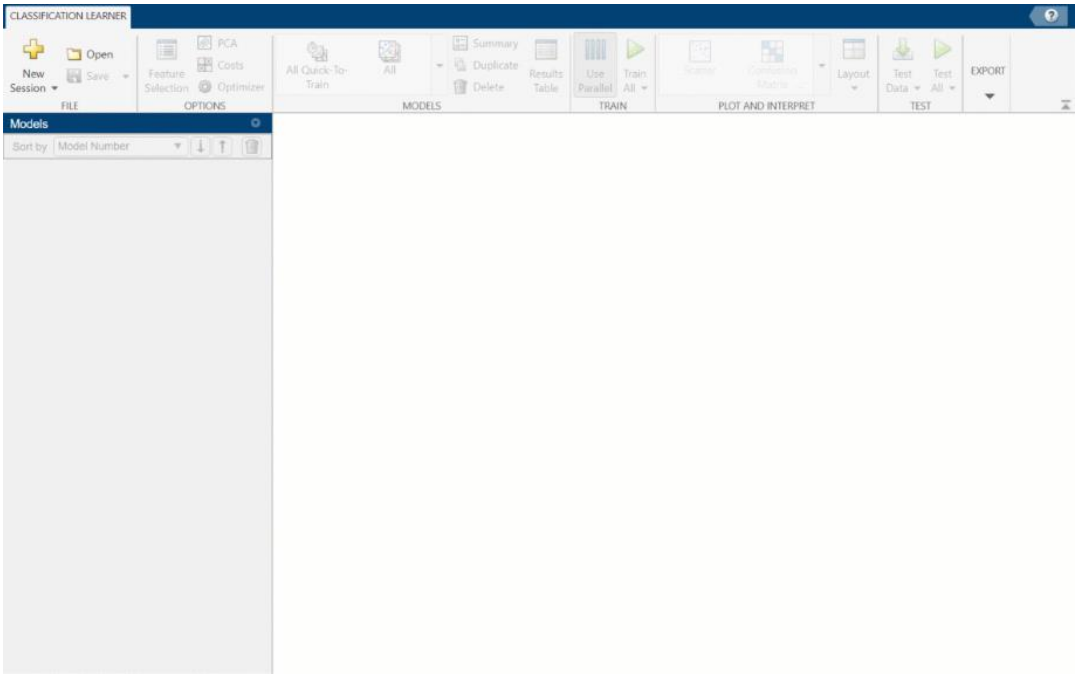
```
allAcceleration = timetable2table(allAcceleration,
"ConvertRowTimes", false)
```

Now, your data is ready to be used for training! One way to train several different machine learning algorithms at once is to use the Classification Learner App, which allows you to rapidly prototype and fine tune your models until you have one you are happy with. You have a lot of options on how you want to use your data and train your models in this app, so while this will be a pretty basic workflow you can learn more about the different options using this documentation.

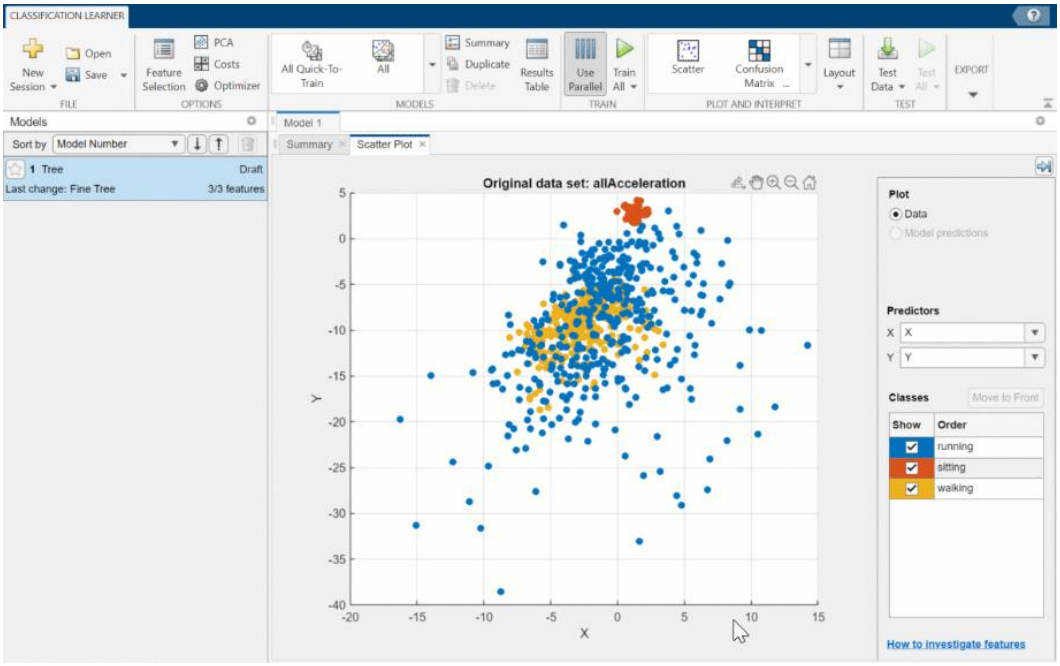
First, open the app:

```
classificationLearner
```

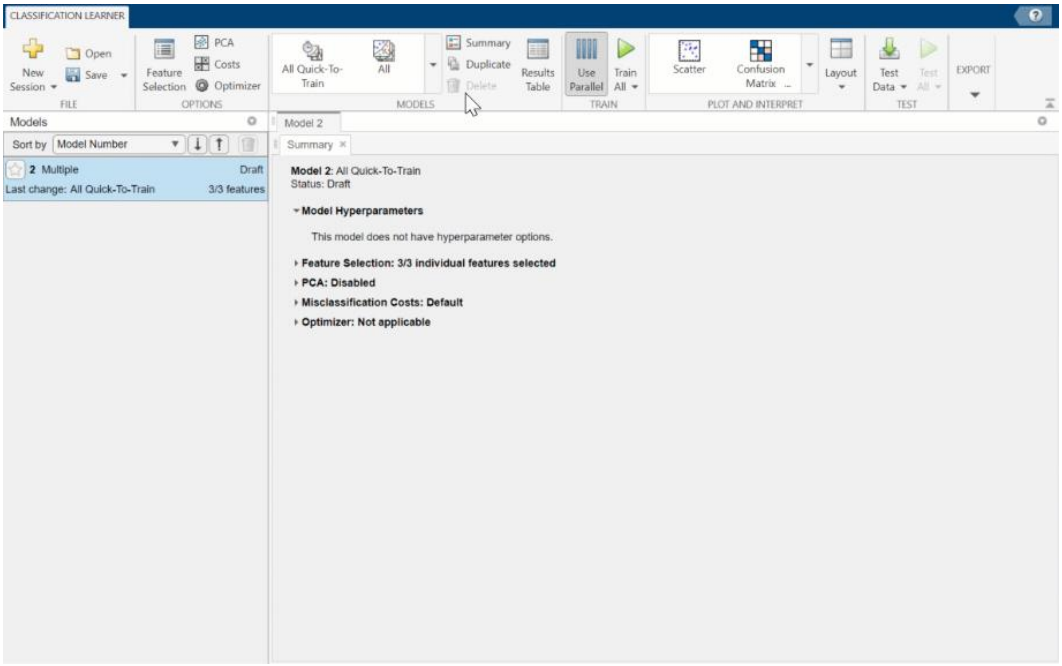
Start a "New Session" and select "From Workspace". In the window that opens, select `allAcceleration` as the 'Data Set Variable', then start the session.



From the 'Models' section of the toolstrip, select 'All Quick-To-Train'. This will add a model labeled 'Multiple' to the 'Models' tab. Then delete the model labeled 'Tree'.



Click Train All, which will start training several models. Once the training has finished, each model will have an ‘RMSE (Validation)’ value. This represents the *root mean squared error* of that model when used on a subset of the training data. The lower the RMSE, the better the model performed. Select the model with the lowest RMSE, then export the model to the workplace.



Now we can use that model to make predictions on new data!

```
justAcc = timetable2table(unknownAcceleration, "ConvertRowTimes", false)

yfit = trainedModel.predictFcn(justAcc)
```

Other useful functions

Function	Use
While loop	Loop through code while a certain condition is true
If statement	Execute certain code when a condition is true
Switch statement	Execute certain code depending on the value of a variable

Present Findings

Creating Graphics

There are many ways in which you can visualize data in MATLAB. Say we wanted to graph the acceleration data that we took earlier. We can do this using the plot command. The only problem with this command is if you wanted to plot the X, Y and Z data on the same graph the code can be messy in one line. Because of this, we use the *hold on* command to continuously plot on the same graph.

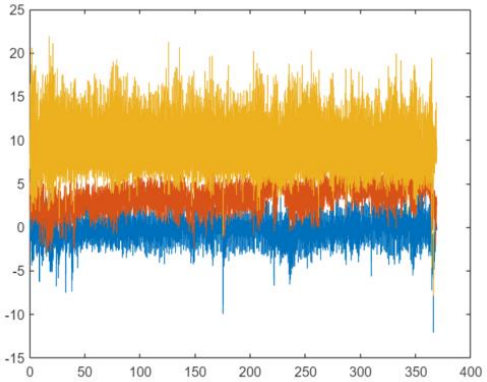
```
plot(t,Xacc);

hold on;

plot(t,Yacc);

plot(t,Zacc);
```

The graph to the right shows the graph that we just made. While there is nothing wrong with this graph, it is very hard to read if you don't know what this data is for. We need to make this graph more detailed by zooming in, differentiating the lines, adding axis labels, and adding a title. This is done with the following commands.




```
xlim([0 50])
```

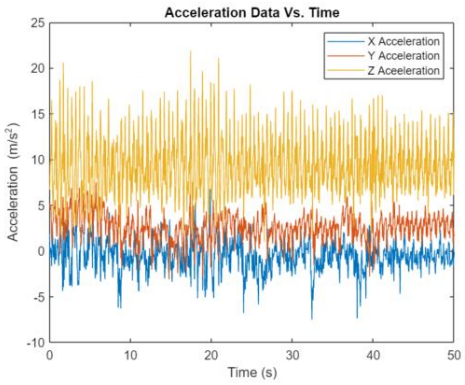
```
legend('X Acceleration','Y Acceleration','Z Aceeleration');

xlabel('Time (s)')

ylabel('Acceleration (m/s^2)');

title('Acceleration Data Vs. Time');
```

Now, as you can see on the right, the graph is presentable. There is just one more thing that we must add to the code and that is *hold off*. If we forget this step, everything we try to plot after this will appear on this same graph.



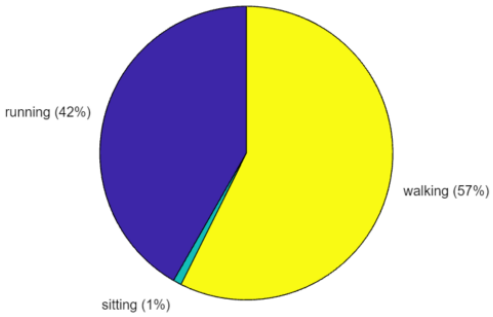

```
hold off
```

Note: Each sensor takes readings at different intervals. For example, if you were to log data with both the position and acceleration sensors, the acceleration log will have a lot more values since acceleration changes quickly. Because of this, there will be more time values for this log. <u>When plotting, be sure to use the right time log.</u>

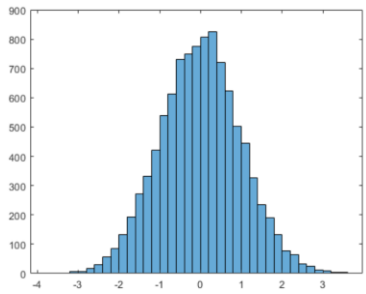
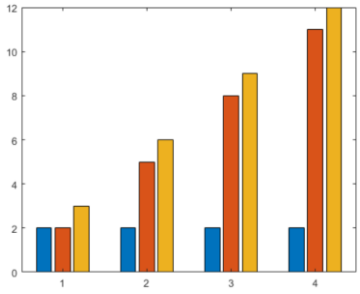
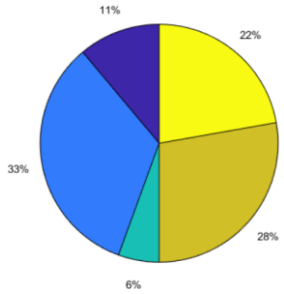
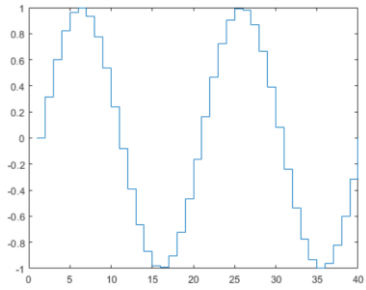
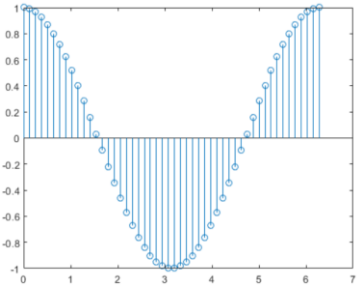
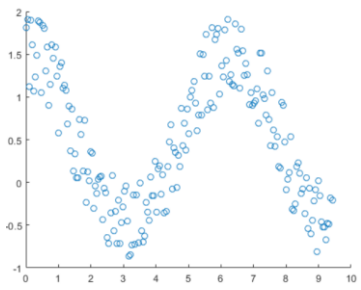
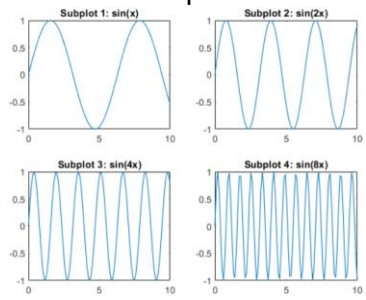
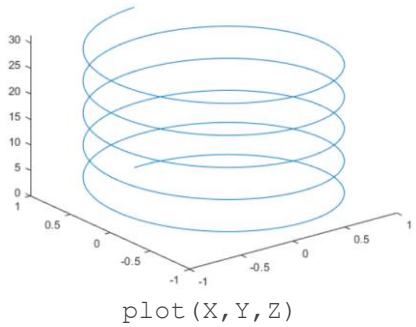
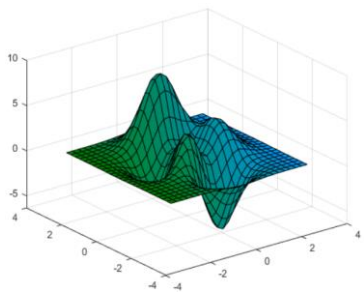
You can also visualize the results of the machine learning model to show a summary of the person's workout. If we want to know how much of each activity a person did during their workout, we can create a histogram:

```
yfitcat = categorical(cellstr(yfit));

pie(yfitcat)
```



Other Graphic tools available in MATLAB

Histogram  `histogram(X)`	Bar  `bar(X)`	Pie  `pie(X)`
Stairs  `stairs(X,Y)`	Stem  `stem(X,Y)`	Scatter  `scatter(X,Y)`
Subplot  `subplot(m,n,p)`	3D Plot  `plot(X,Y,Z)`	Surface  `surf(X,Y,Z)`

To learn how to use these commands, you can use the doc command followed by the specific graph. For example, if you wanted to learn how to use subplot, you could type the following:

```
>> doc subplot
```

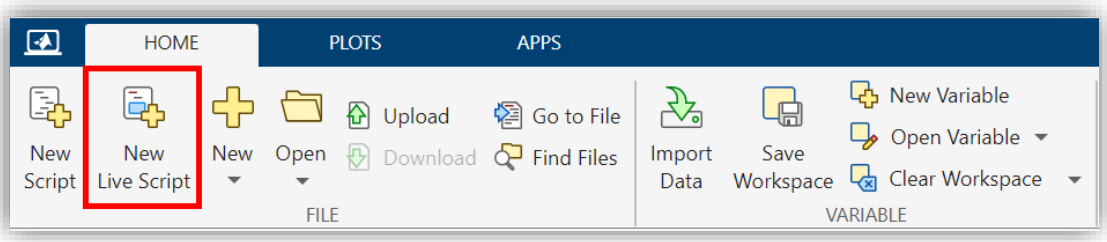
Ways to present results

When you are all ready to present your data, it will be up to your team to choose how to present to the judges. Feel free to be creative with this step because this will really set your group apart. You can choose to do a report, presentation, or even a video.

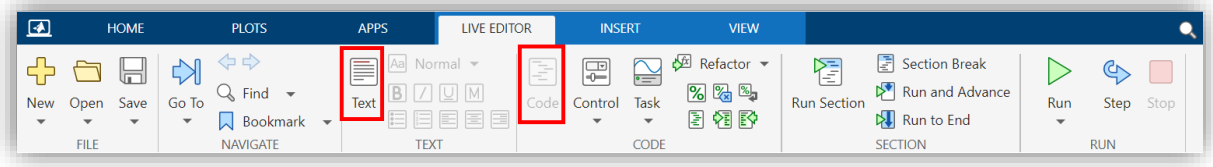
When presenting your findings, remember to do it in such a way that anyone could understand. How would you tell your little brother about how many calories he's burned? Possibly by telling him he has burned half a burger in is workout. Get creative and use some comparison such as this one to convey the results.

Live Scripts

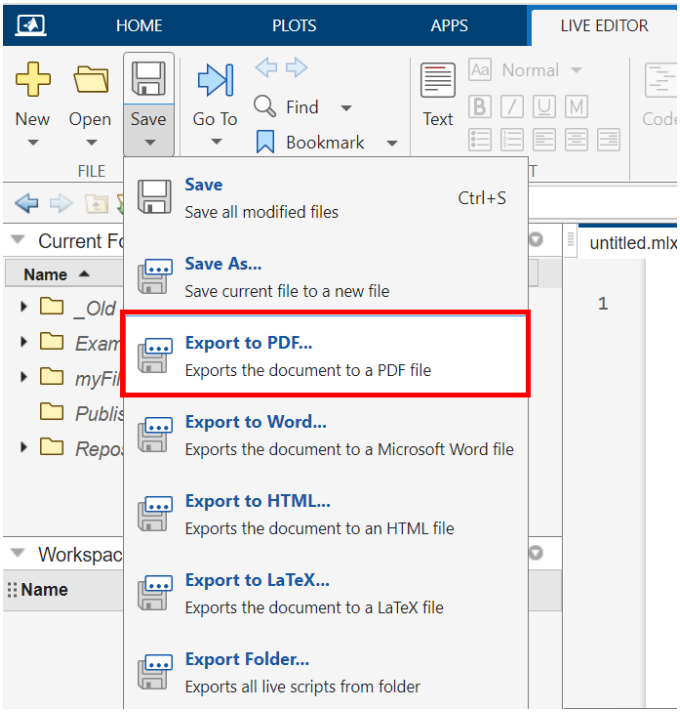
MATLAB has a really great way of presenting code called Live Scripts, which will likely be the best way to present your code. This is a form of a script that we mentioned earlier that also includes formatted text and images. The way to open a live script is the same way that you would create a script, except you would click the 'New Live Script' Button under the 'Home' tab.



As mentioned earlier, you can write text as well as code within a live script. If you want to switch between the two there are two buttons that you can use. The 'Text' button switches to text mode and anything that you type while in this mode will not be executed when you run the code. To switch back to writing code you can click the 'Code' button.



Another useful feature about live scripts is the ability to publish your scripts. First click the 'Save' drop down in the task bar, then click 'Export to PDF.' A pop-up will ask you to name the file and then you are done! When submitting live scripts, this is how you should present them.



Submitting your Results

Once you have finished your model, you will have to create a submission for your team. The submission must be a public repository that contains all the MATLAB files developed and an explanation of your fitness model and results in the form of a report, presentation, or video. Teams are encouraged to be creative when demonstrating their efforts and follow the grading rubric available in the provided documents when developing their submission. Further instructions on how to deliver the submission will be shared with teams at the event.