Module 3: Barplot Bootcamp

Introduction

Welcome to the **Barplot** bootcamp! In this module we are going to learn how to load data from a `.csv` file or as it is more commonly known, an excel spreadsheet. This will also be one of your first opportunities to load packages and use them to analyze and plot your data. **Barplots** are a useful plot that can be used to compare many variables, typically at the very least the relatioship between a **categorical variable** and a **numerical variable**. In this module we will learn about loading `.csv` files and analyzing them using a **Python** package called `pandas`. We will also be utilizing two plotting packages: `matplotlib` and `seaborn`. `Matplotlib` creates a framework and rules for creating figures. `Seaborn` uses the `matplotlib` rules and makes them loom more aesthetically pleasing and also makes it easier to use with data loaded using `pandas`. Lastly, we will utilize one of the most important packages for using data matrices: `Numpy` (Note: don't be a noob and pronounce it wrong it. It reads like Numb - pie. Think of Number-python.).

Loading Packages

Let's get started and dive in! First we are going to load in these packages. Anytime you load in a **package** into **Python** this should be done at the beginning of the script. To load a package you have to use the `import` function followed by the name of the package. You can choose to follow this with `as "insert your name of choice"` to give it a shorter name. This is convenient when you have to call a function from this package repeatedly and want to type less. For example instead of calling `numpy.mean(array)` I can just call `np.mean(array)`.

In [1]:
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import seaborn as sns
```

Loading Data

Alrighty, let's go ahead and load in some data. On **D2L** there should be some example data saved as a `.csv` file that you can download. If you have access to the excel version of the file save it as a `.csv` file. Important: you need to know the hierarchy of folders to get to the `.csv` file so that you can load it into your script. For example, if you saved it on your **Desktop** on *MacOS* the location will be something like this (using mine as an example) `/Users/JakeCanfield/Desktop/data.csv`.

The function that we are going to use to load our data is `pd.read_csv()`. When we load the data we need to assign it to a variable of our choosing so that we can interact with it in the future. We are also going to make a variable with a `string` that defines the path to our data and a place to save things later. When we do this we can just add the `string` for the filepath (**fp**) and the name of our file or anything else in that folder. Go ahead and try it!

In [13]:
```
fp = '/Users/JakeCanfield/Documents/PSL_475L_Capstone_Physiology_Lab_MSU/Modu
df = pd.read_csv(fp + 'data.csv')
```

Now that the data is loaded, let's print it out to see what it looks like. Use the `print()` or `display()` function to show the data (display looks nicer in **Jupyter Notebooks**.

In [15]:
```
display(df)
```

	Name	Age	Gender	BMI	Smoker	Fasted	Caffeine Free	Included?
0	John	19	m	22.0	n	y	y	y
1	Susie	20	f	18.0	n	y	y	y
2	Alex	21	m	24.0	n	y	y	y
3	Zoey	21	f	25.0	n	y	y	y
4	Joe	22	m	19.0	n	y	y	y
5	Chad	23	m	20.5	n	y	y	y
6	Sam	19	m	21.5	n	y	y	y
7	Josie	23	f	22.0	n	y	y	y
8	Henry	24	m	23.0	n	y	y	y
9	Elise	27	f	30.0	n	n	y	n

One of our experimental exclusion criteria was coming to the experiment fasted. As we can see from the table, **Elise** did not qualify to be counted in our data because she did not come fasted. We need to remove her from the data so that she doesn't contribute with her confounding variable. We are going to overwrite our `DataFrame` (name of the object that our data is stored as right now) and removing all of the rows where there is not a **y** in the **Fasted** column.

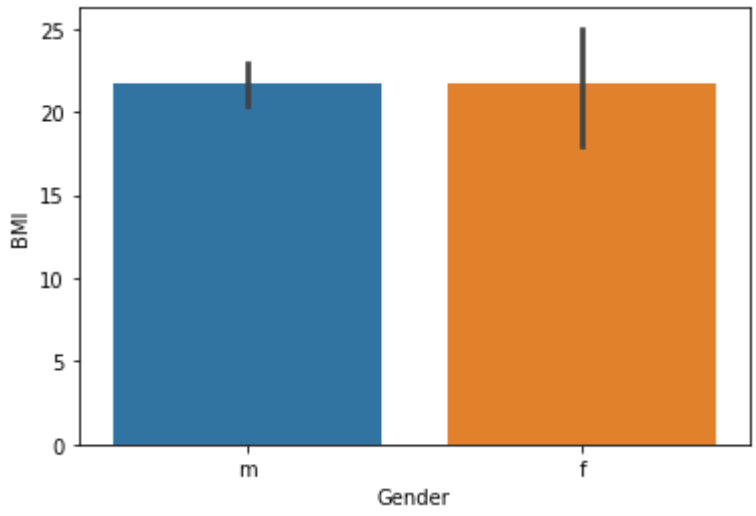
In [16]:
```
df = df.loc[df['Fasted'] == 'y']
display(df)
```

	Name	Age	Gender	BMI	Smoker	Fasted	Caffeine Free	Included?
0	John	19	m	22.0	n	y	y	y
1	Susie	20	f	18.0	n	y	y	y
2	Alex	21	m	24.0	n	y	y	y
3	Zoey	21	f	25.0	n	y	y	y
4	Joe	22	m	19.0	n	y	y	y
5	Chad	23	m	20.5	n	y	y	y
6	Sam	19	m	21.5	n	y	y	y
7	Josie	23	f	22.0	n	y	y	y
8	Henry	24	m	23.0	n	y	y	y

As you can see we removed Elise. We used the `loc` function on the `df` DataFrame. This lets us search through the rows or **Index** and perform some task on all of them. Then inside the `[]` we called `df['Fasted']`. This selects the column **Fasted** out of the list of columns in the DatFrame `df`. Lastly, we say that we are only going to keep the rows that have the value **y** in the column **Fasted**. Aka, we filtered out anybody who responded **No** or **n**.

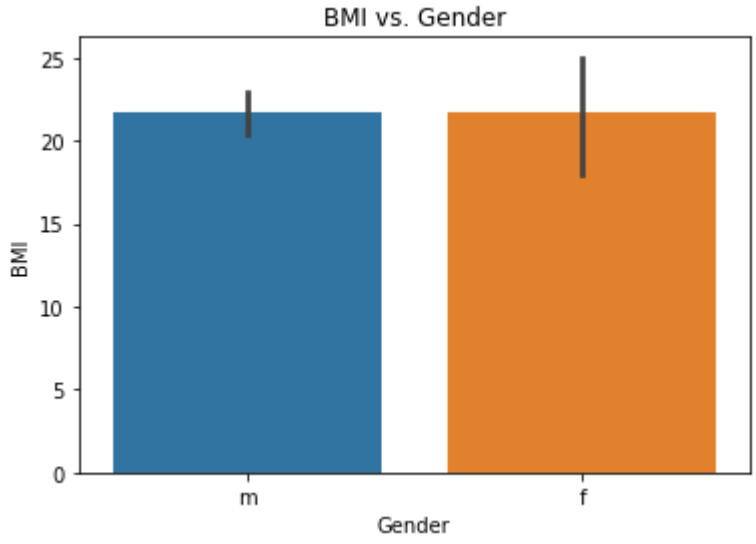
Now that we have our data cleaned up let's start making figures! We are going to use the function `sns.barplot()` to make our figure. Let's try it. We need to specify the `data` and the `x` and `y`. To actually see our figure we need to use the function `plt.show()`. We only have to call the name of the columns for the `x` and `y` because we loaded the data into the plotting function as a **pandas** `DataFrame` we are able to do this quite easily.

In [17]:
```
1 sns.barplot(data = df, x = 'Gender', y = 'BMI')
2 plt.show()
```



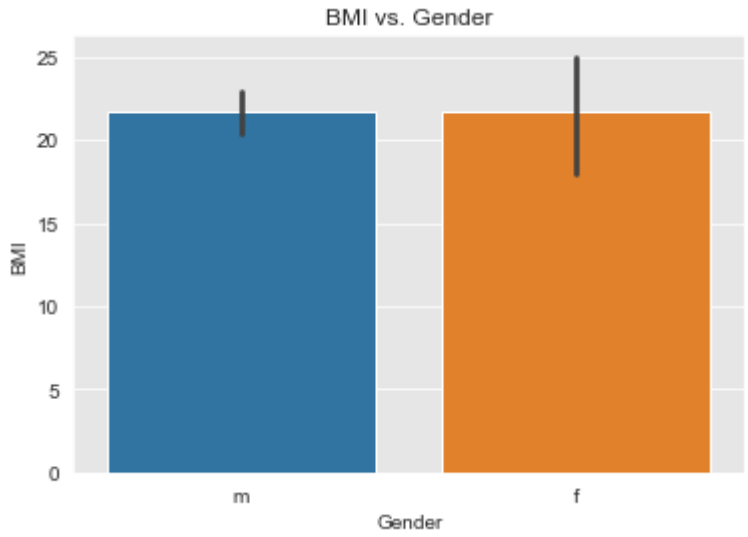
We can add a title with the function `plt.title()`.

In [19]:
```
1 sns.barplot(data = df, x = 'Gender', y = 'BMI')
2 plt.title('BMI vs. Gender')
3 plt.show()
```



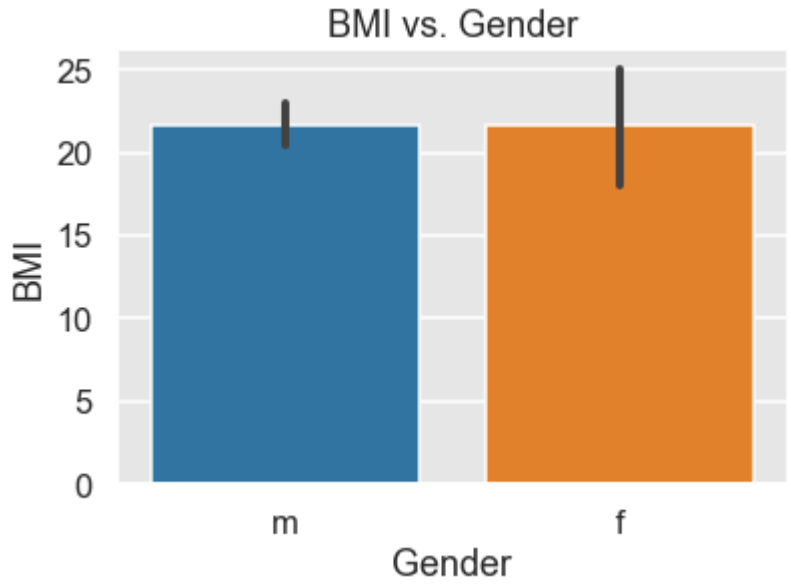
We can change the figure aesthetics using the function `sns.set_style()`. I'm going to use one called **darkgrid**. You can find more of these themes here (https://seaborn.pydata.org/tutorial/aesthetics.html).

In [23]:
```
1 sns.set_style("darkgrid", {"axes.facecolor": ".9"})
2 sns.barplot(data = df, x = 'Gender', y = 'BMI')
3 plt.title('BMI vs. Gender')
4 plt.show()
```



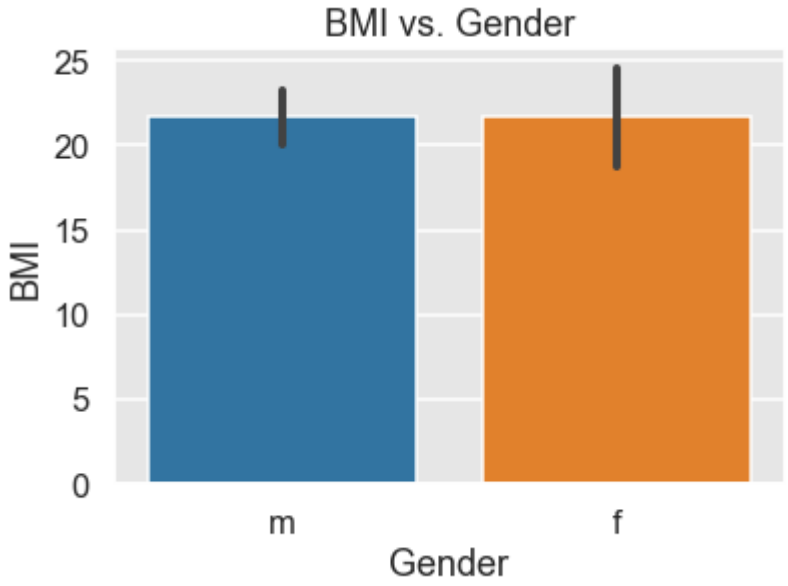
We can also change many features by declaring the figure context using `set_context()`. Check out some other options at the bottom of the page here (https://seaborn.pydata.org/tutorial/aesthetics.html).

In [26]:
```
1 sns.set_style("darkgrid", {"axes.facecolor": ".9"})
2 sns.set_context('talk')
3 sns.barplot(data = df, x = 'Gender', y = 'BMI')
4 plt.title('BMI vs. Gender')
5 plt.show()
```



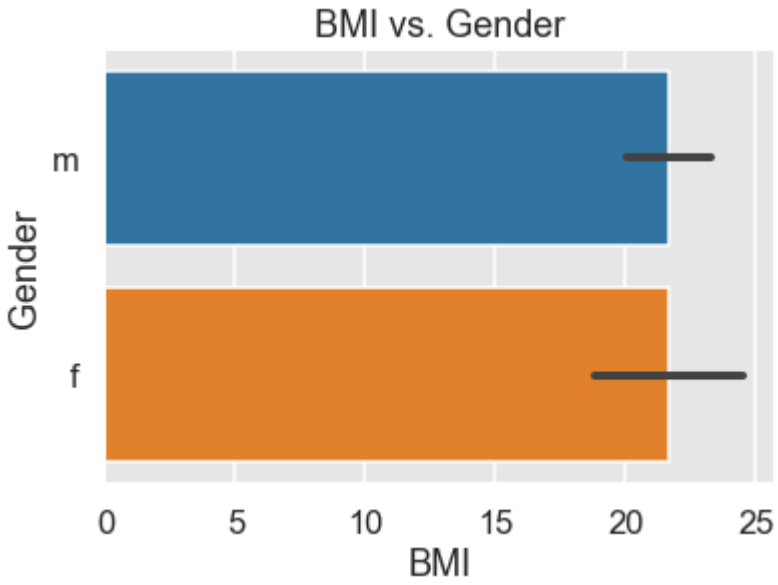
If we want our error bars to be **standard deviation** we need to add the argument `ci = 'sd'`.

In [27]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Gender', y = 'BMI', ci = 'sd')
plt.title('BMI vs. Gender')
plt.show()
```



You can change the orientation of the plot with the `orient` argument. Note: you need to change the `x` and `y` variables.

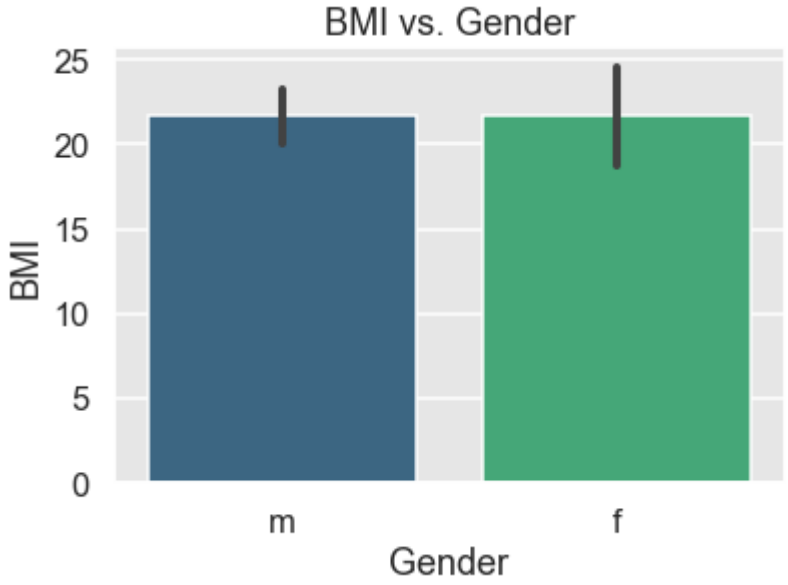
In [30]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'BMI', y = 'Gender', ci = 'sd', orient = 'h')
plt.title('BMI vs. Gender')
plt.show()
```



You can also change the color palete using the argument `palette` . Check out the different available color palettes here (http://seaborn.pydata.org/tutorial/color_palettes.html). I picked **viridis** which is a a color

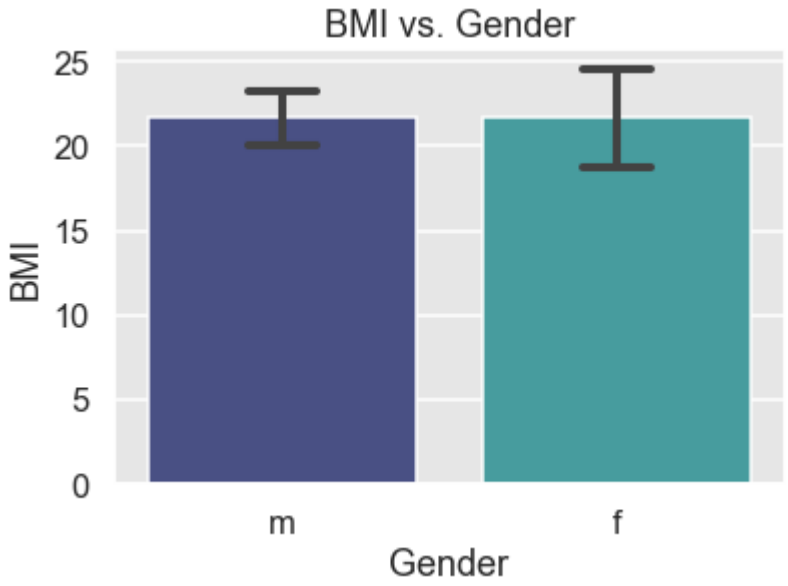
palette that maximizes contrast for color blind people.

In [32]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Gender', y = 'BMI', ci = 'sd', palette = 'viridis
plt.title('BMI vs. Gender')
plt.show()
```



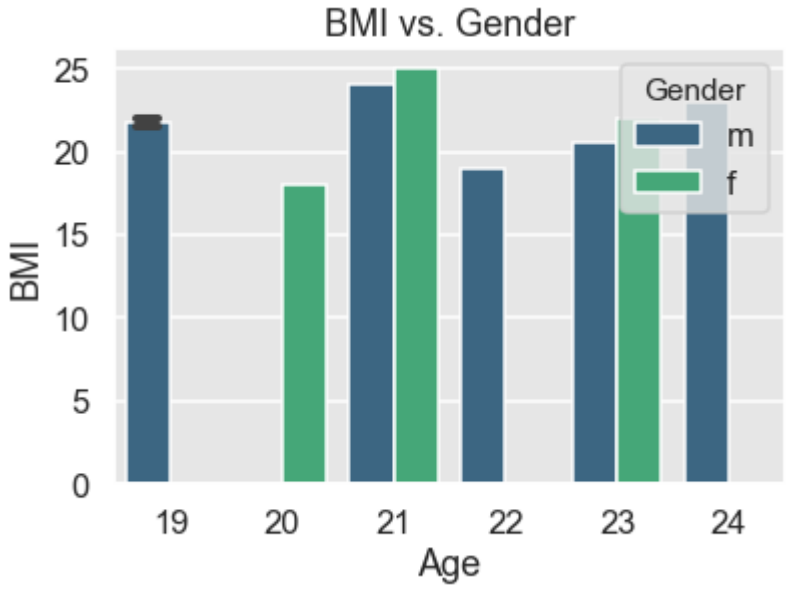
We can add some caps to the error bars with the argument `capsize` .

In [48]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Gender', y = 'BMI', ci = 'sd', palette = 'mako',
plt.title('BMI vs. Gender')
plt.show()
```



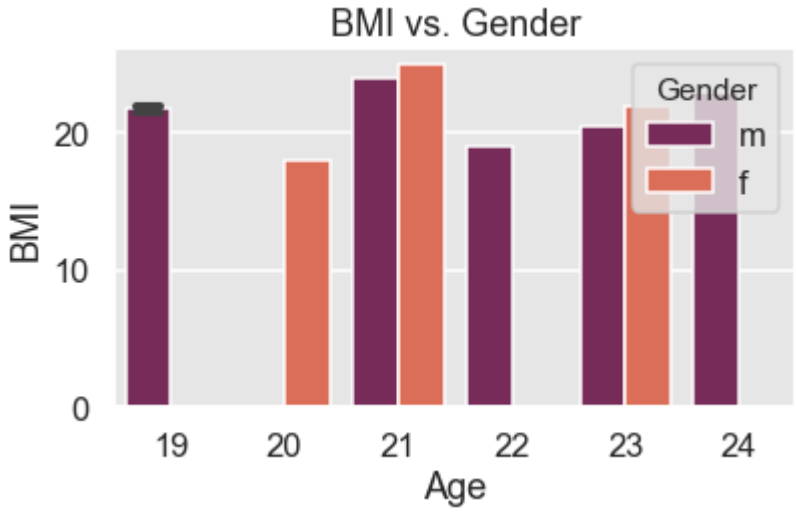
We could also plot another variable using the `hue` argument. Let's try it.

In [41]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Age', y = 'BMI', hue = 'Gender', ci = 'sd', palet
plt.title('BMI vs. Gender')
plt.show()
```



We can add the function `plt.tight_layout()` to make sure that the figure fits the screen dimensions better.

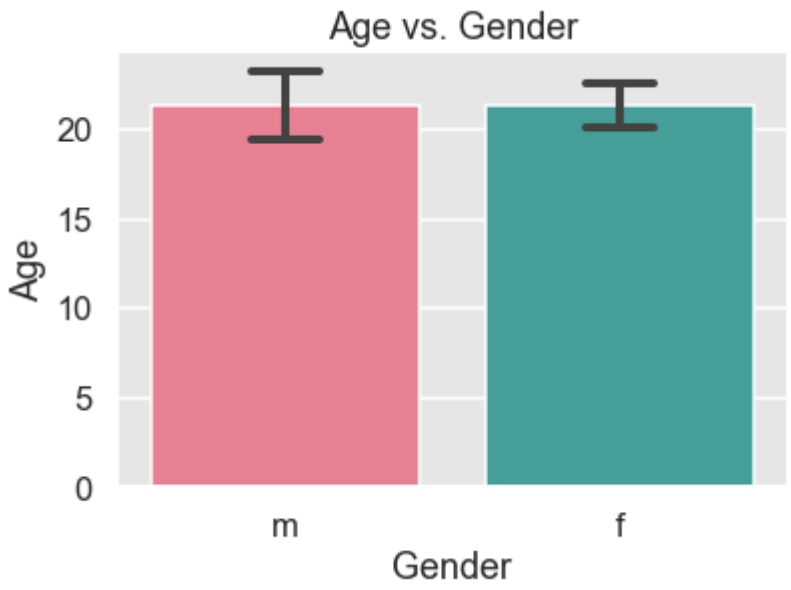
In [44]:
```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Age', y = 'BMI', hue = 'Gender', ci = 'sd', palet
plt.title('BMI vs. Gender')
plt.tight_layout()
plt.show()
```



Let's try age:

In [46]:

```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sns.set_context('talk')
sns.barplot(data = df, x = 'Gender', y = 'Age', ci = 'sd', palette = 'husl',
plt.title('Age vs. Gender')
plt.show()
```



Conclusion

I can go on for eternity showing you all of the different ways in which you can modify figures using this code. There isn't a single aspect that I can think of that can't be customized or modified to your heart's desire. If you get stuck an excellent resource is **Stack Overflow** where other people post questions and answers to their problems. Include python and the name of the package or error in your search for best results. Lastly, a great resource is the example gallery on the seaborn website, which also includes tutorial. Check it out here (https://seaborn.pydata.org/examples/index.html)! Thanks for doing this module and I look forward to doing the next one with you! Feel free to email me with questions @ canfie44@msu.edu (mailto:canfie44@msu.edu) !