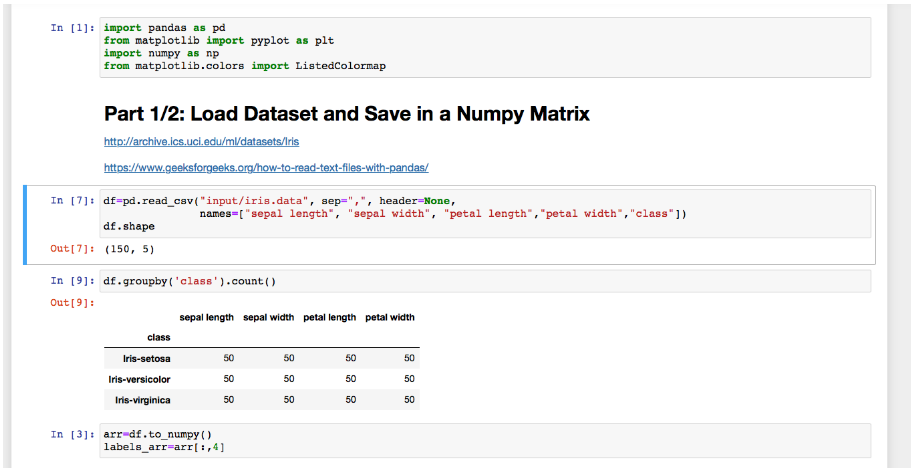
**Applied Machine Learning: CS5785** **Name:** *Rhia Singh*
**Assignment 0**
**September 1, 2022**

1. There are 4 attributes per sample and 3 species - Iris-setosa, Iris-versicolor, Iris-virginica - included in the Iris data set. The data set is balanced as there is a total of 150 samples, with each species having 50 instances in the data set (Figure 1).



```
In [1]: import pandas as pd
        from matplotlib import pyplot as plt
        import numpy as np
        from matplotlib.colors import ListedColormap
```

**Part 1/2: Load Dataset and Save in a Numpy Matrix**

http://archive.ics.uci.edu/ml/datasets/Iris

https://www.geeksforgeeks.org/how-to-read-text-files-with-pandas/

```
In [7]: df=pd.read_csv("input/iris.data", sep=",", header=None,
                       names=["sepal length", "sepal width", "petal length","petal width","class"])
        df.shape

Out[7]: (150, 5)

In [9]: df.groupby('class').count()

Out[9]:
```

| | sepal length | sepal width | petal length | petal width |
|---|---|---|---|---|
| **class** | | | | |
| **Iris-setosa** | 50 | 50 | 50 | 50 |
| **Iris-versicolor** | 50 | 50 | 50 | 50 |
| **Iris-virginica** | 50 | 50 | 50 | 50 |

```
In [3]: arr=df.to_numpy()
        labels_arr=arr[:,4]
```

Figure 1: Code for loading, parsing, and overview of the data set (part 1 and 2)

2. The data set was downloaded as a text file with comma delimiters

   (http://archive.ics.uci.edu/ml/datasets/Iris). After looking online, I found that I could parse the text file with the 'read csv' function in the pandas library (https://www.geeksforgeeks.org/how-to-read-text-files-with-pandas/). This one line of code is clean and efficiently handles the small data set.

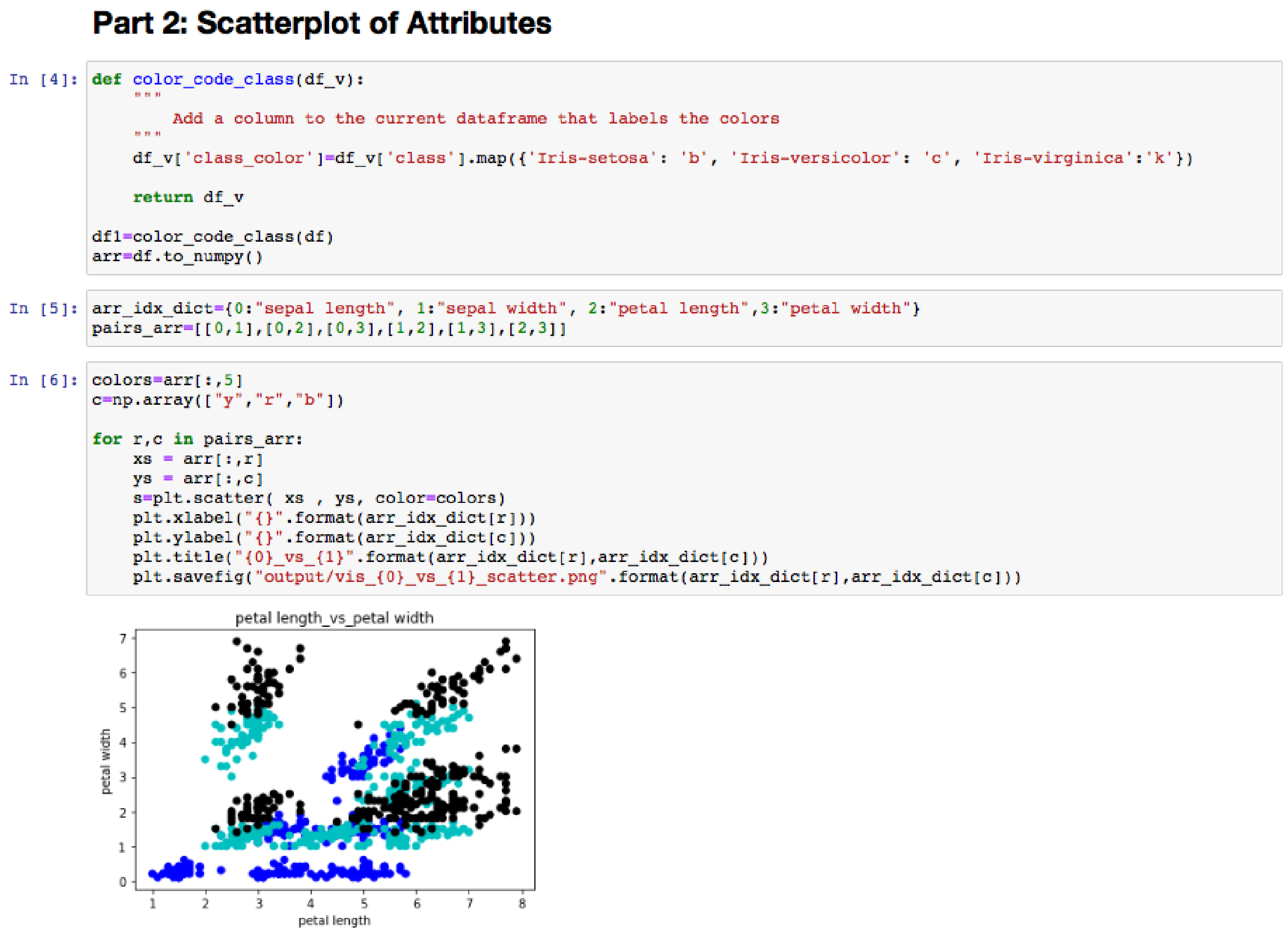
   The function ignores headers, parses with commas, and included the column titles that I specified as a parameter. I expected the data set to include the columns I labeled because as described in the data overview on the website (http://archive.ics.uci.edu/ml/datasets/Iris). In a situation where I did not have this prior information, then I would have to tinker around a bit and possibly try a bit of trial and error. In this case, this worked well.

   As requested in the assignment instructions, I cast the data frame object to a Numpy array of $N * p$, where "N is the number of samples and p is the number of attributes per sample." More specifically, N is 150 and p is 4 (technically the 4th column is a label that could be potentially be predicted with the first 3 attributes). The label column is further isolated in the 'labels arr' variable (Fig. 1). To conclude, the data set was loaded and parsed with Pandas and transformed from a data frame to an array with the Numpy library.

3. **Methodology Overview** The following details my method for creating scatter plots of the 3 attributes within the 3 classes. The function 'color code class' takes a data frame as an input and then adds a column called, class color, to label each record with a color based on the class label (Fig. 3).

   The new data frame is then converted to an $N * p$ array, where N is the number of samples and p is the attributes (Fig. 3). After, I create a dictionary that maps the column number to the attribute name. For example, column 0 in the array details the 'sepal length' of the sample. An array with each permutation of the attributes is created, to generalize and streamline the creation of the scatter plots. There are only 4 attributes in this data, so hard-coding this array works, but ideally, a permutation function should be written that can work for larger sets of attributes.

   A for loop then iterates through the array of attribute permutations to create the 6 scatter plots of the attributes and saves the graphs to an output folder as 'png' files (Fig. 3). This for loop runs in O(n) time, which is efficient and will scale well for future scatter-plot creation.



## Part 2: Scatterplot of Attributes

```
In [4]: def color_code_class(df_v):
            """
                Add a column to the current dataframe that labels the colors
            """
            df_v['class_color']=df_v['class'].map({'Iris-setosa': 'b', 'Iris-versicolor': 'c', 'Iris-virginica':'k'})

            return df_v

        df1=color_code_class(df)
        arr=df.to_numpy()
```

```
In [5]: arr_idx_dict={0:"sepal length", 1:"sepal width", 2:"petal length",3:"petal width"}
        pairs_arr=[[0,1],[0,2],[0,3],[1,2],[1,3],[2,3]]
```

```
In [6]: colors=arr[:,5]
        c=np.array(["y","r","b"])

        for r,c in pairs_arr:
            xs = arr[:,r]
            ys = arr[:,c]
            s=plt.scatter( xs , ys, color=colors)
            plt.xlabel("{}".format(arr_idx_dict[r]))
            plt.ylabel("{}".format(arr_idx_dict[c]))
            plt.title("{0}_vs_{1}".format(arr_idx_dict[r],arr_idx_dict[c]))
            plt.savefig("output/vis_{0}_vs_{1}_scatter.png".format(arr_idx_dict[r],arr_idx_dict[c]))
```

Figure 2: Code for creating scatterplots

**Scatter Plots** (Iris-setosa - royal blue, Iris-versicolor - cyan, Iris-virginica - black)

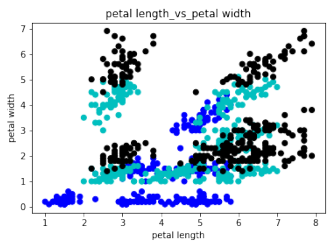


Figure 3: Petal Length v. Petal Width

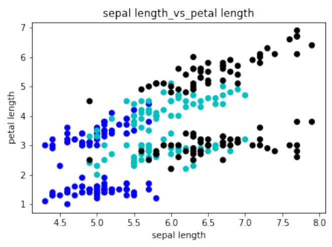


Figure 4: Sepal Length v. Petal Length

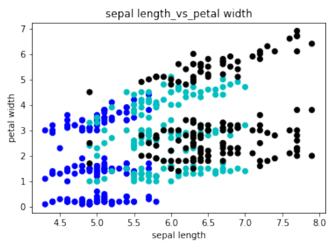


Figure 5: Sepal Length v. Petal Width

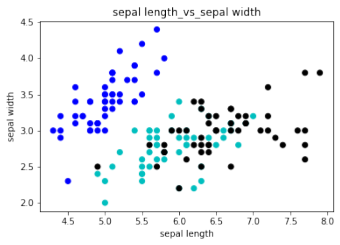


Figure 6: Sepal Length v. Sepal Width

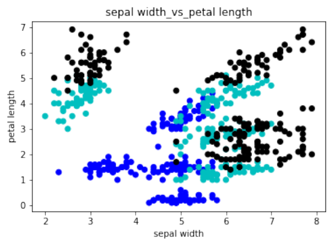


Figure 7: Sepal Width v. Petal Length

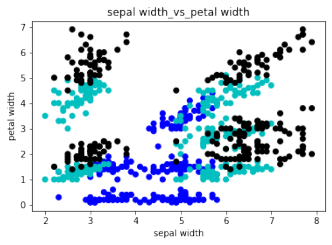


Figure 8: Sepal Width v. Petal Width

**Discussion** Overall, there is no clear strong correlation in the scatter plots. It may be worth checking for outliers before proceeding further, to ensure that the trends are not being hidden by irregular iris data. There is a slight direct relationship between Sepal Length and Petal Length, which seems a bit surprising and I'm not sure why those 2 plant features would be related (Fig. 4). Also, it seems like the Setosa's tend to have shorter Sepal length than the other 2 species. This may be beneficial in identifying the species with a clustering algorithm.