# Report

## Part 1

Libraries used for this problem: numpy, pandas, statistics, sklearn and matplotlib.pyplot.

Pandas (pd.read_csv) used to load the data into the program so it can be used later.

Matplotlib.pyplot imported as plt used to visually represent the data in order to better understand it.

.head() function can be found in the pandas library and it shows the head of the table and the first 5 rows of data found in it.

.test_train_split(), here used with 4 arguments. First two being x and y, one dimensional arrays that contain the coordinate values found in the data file. Next one is train_size that decides where the split in the data should be made. Here the value is 0.8 meaning that the X_train and y_train should get 80% of the data.

In order to see the different outcomes between different approaches to this problem I decided to do both a gradient descent version and a least squares method version.

For the gradient descent method the .sum() function prom pandas is used to calculate the sum between different arrays. This is valuable when calculating the mean and the y-intercept.

Once those values are found, the regression line is represented using plt.scatter() and plt.plot().

The first function shows the different points from our data and the second one shows the line we found the coordinates for.

plt.plot() takes (here) as arguments the coordinates of the points we want to plot and the color green.

plt.show() actually formats and prints on the screen the graphics we just described with the functions above.

Statistics were used in order to calculate the mean of the residual data we have in order to figure out the error margin and how effective the prediction is.

After this we plot the residuals.

For the least square method, first we use the numpy .size() function which returns the number of entries in the data given as an argument.

In order to calculate the mean and the y-intercept we use the np.mean() function, found in the numpy library. This returns the mean of the values found in the array given as the argument.

Other than this, we use the np.sum() function which returns the sum of all the values in the array given as argument.

After we calculate the prediction we use the same method as before in order to visually see the result.

## Part 2

The first thing we do is to use the .read_csv() function. It is found in pandas and although in part 1 we used it with a single argument, here, besides the name of the data file we want, we have a different argument, names, which gives each column a name in the order we wrote it in.

So that we can visualize the data better, we make a colors dictionary which attributes each class a different color.

Next, we use plt.subplots in order to map the scattered data. Here the .scatter function takes as arguments two of the columns found in the data table and the color dictionary in order to visually see the difference between each class.

.drop function is used to delete the class column so that we are left only with the numerical data.

We then use the functions .groupby() and .size() to see how many of each flower type we have.

Next we split the data into two arrays. X which is multidimensional and contains the numerical data and y which contains the class.

From sklearn.preprocessing we import the LabelEncoder, from this library we need the .fit_transform() function which turns the string data currently in y into a numerical encoding of it so that we can process it easier.

After this we use the same function as in the part 1 of this project, .train_test_split() in order to split 75% of our data into training data and the rest for testing.

From sklearn, we import the KNeighborsClassifier. This function will set up a classifier for us that we can later train using certain functions. The only argument used in this case is the number of neighbours it should look at before making a prediction.

.fit() function takes as argument the two coordinate arrays for our training data. The purpose of the fit function is so that our model can “learn” and store the data in order to make future predictions.

.predict() function predicts the class of a certain entry. It takes as an argument the array of the testing data.

In the end, in order to test the accuracy of our model we import metrics from sklearn. From here we use the metrics.accuracy_score() which takes as argument the testing data first and the predicted data second and measures the differences between the two.