**NAME**

**COLLEGE NUMBER**

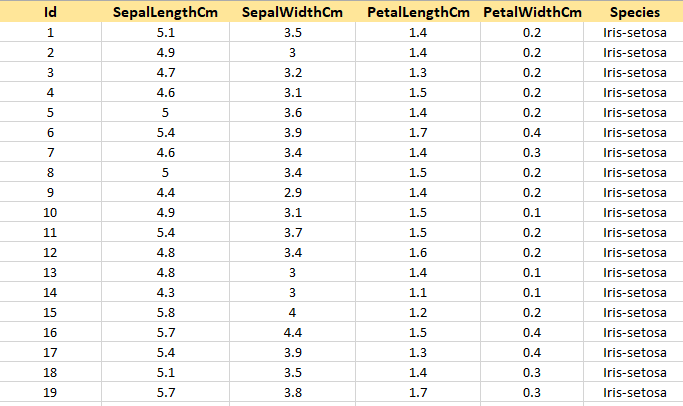
**Implementation of the KNN model based on the Iris dataset**

**Introduction:**

The KNN mode is an effective algorithm based and suitable for the classification and forecasting of certain class object properties within the dataset. Usually, a typical class has certain objects properties that can then be used to objectively classify the other objects within the plant family. The Iris dataset has a combination of three families that include the Setosa, Versicolour and the Virginica species. Further, the dataset consist of 150 instances of data. The classification of each of these species clearly depends on the characteristics of each of these species outlined below. Further, the dataset consists of For instance, in the case of the Iris flower set, the following are some of the objects identified:

* Petal length
* Petal width
* Sepal length
* Sepal width

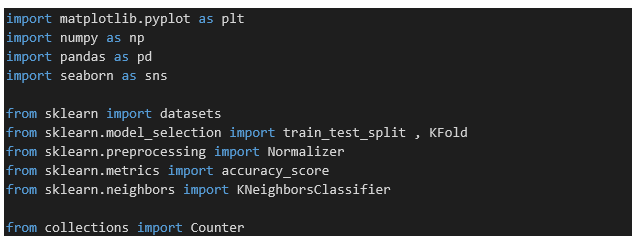
The characteristics of each of the Iris datasets is dependent on the above four attributes described above.



A quick preview of the dataset looks like above with the variable names of id, sepal length, sepal width, petal length, petal width and the species. The ids takes a variable type of integer, the sepal and petals take the variable types of Boolean and the species take the variable types of string.

**Analysis:**

The analysis of the iris dataset is based on the KNN model with Python and the Spyder IDE development environment. Then import the necessary packages into the datasets required for the analysis of each steps.



The next step in the analysis is to load the dataset into the IDE.

# importing the iris dataset

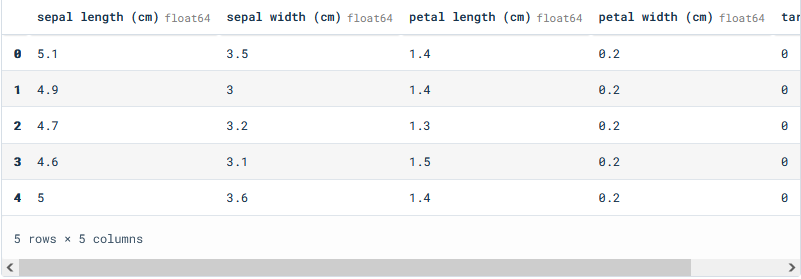
iris = datasets.load\_iris()

# np.c\_ is the numpy concatenate function

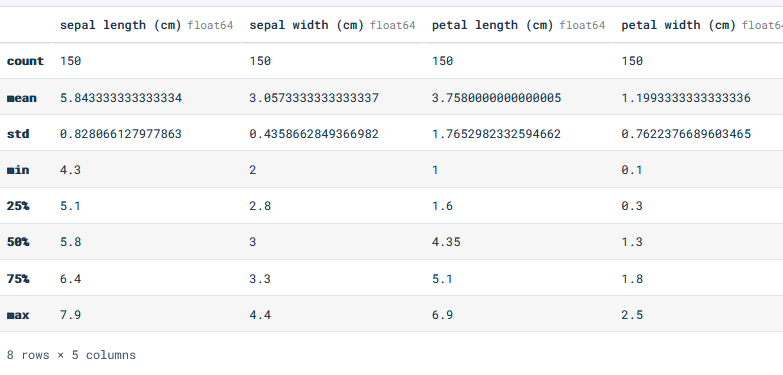
iris\_df = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],

                      columns= iris['feature\_names'] + ['target'])

iris\_df.head()



iris\_df.describe()



The next step is to split the dataset into training sets and tests data as below:

#Split the data into train and tests

x= iris\_df.iloc[:, :-1]

y= iris\_df.iloc[:, -1]

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y,test\_size= 0.2,shuffle= True)

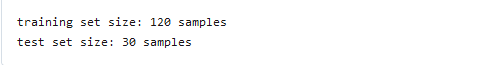
x\_train= np.asarray(x\_train)

y\_train= np.asarray(y\_train)

x\_test= np.asarray(x\_test)

y\_test= np.asarray(y\_test)

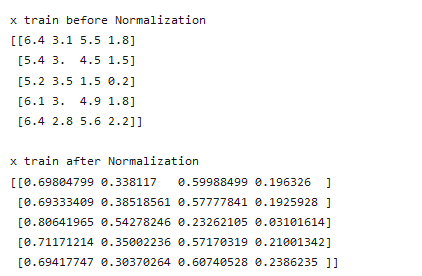
print(f'training set size: {x\_train.shape[0]} samples \ntest set size: {x\_test.shape[0]} samples')



scaler= Normalizer().fit(x\_train)

normalized\_x\_train= scaler.transform(x\_train)

normalized\_x\_test= scaler.transform(x\_test)



#Normalize the dataset

scaler= Normalizer().fit(x\_train)

normalized\_x\_train= scaler.transform(x\_train)

normalized\_x\_test= scaler.transform(x\_test)

#Visualize the Dataset before and after Normalization

## Before

# View the relationships between variables; color code by species type

di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0:'Virginica'} # dictionary

before= sns.pairplot(iris\_df.replace({'target': di}), hue= 'target')

before.fig.suptitle('Pair Plot of the dataset Before normalization', y=1.08)

## After

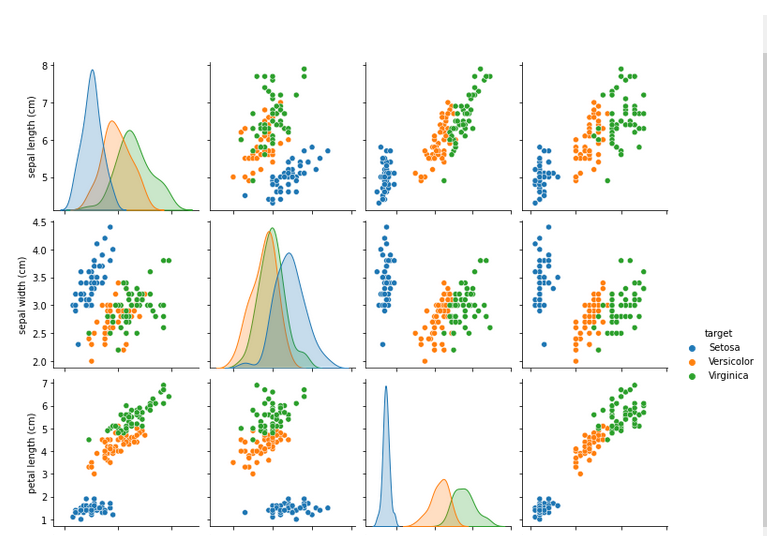
iris\_df\_2= pd.DataFrame(data= np.c\_[normalized\_x\_train, y\_train],

                        columns= iris['feature\_names'] + ['target'])

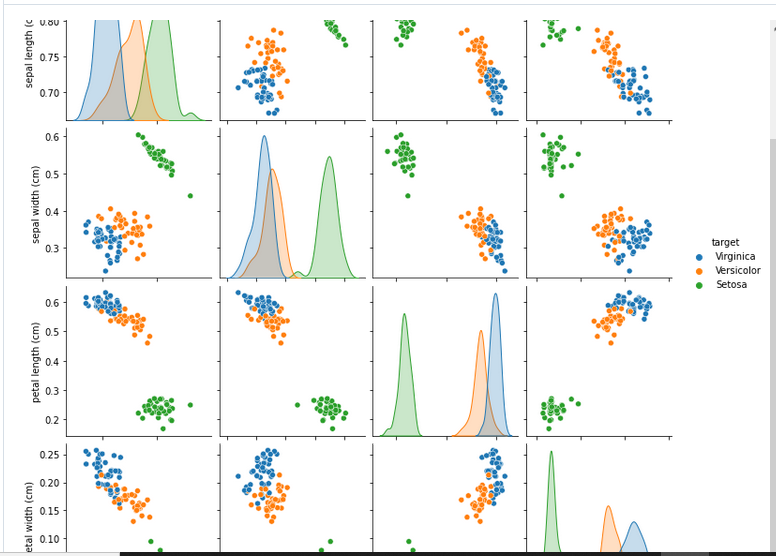
di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0: 'Virginica'}

after= sns.pairplot(iris\_df\_2.replace({'target':di}), hue= 'target')

after.fig.suptitle('Pair Plot of the dataset After normalization', y=1.08)



**After normalisation**



**Step one for the KN1 Modelling:**

#KNN STEP 1

def distance\_ecu(x\_train, x\_test\_point):

  """

  Input:

    - x\_train: corresponding to the training data

    - x\_test\_point: corresponding to the test point

  Output:

    -distances: The distances between the test point and each point in the training data.

  """

  distances= []  ## create empty list called distances

  for row in range(len(x\_train)): ## Loop over the rows of x\_train

      current\_train\_point= x\_train[row] #Get them point by point

      current\_distance= 0 ## initialize the distance by zero

      for col in range(len(current\_train\_point)): ## Loop over the columns of the row

          current\_distance += (current\_train\_point[col] - x\_test\_point[col]) \*\*2

          ## Or current\_distance = current\_distance + (x\_train[i] - x\_test\_point[i])\*\*2

      current\_distance= np.sqrt(current\_distance)

      distances.append(current\_distance) ## Append the distances

  # Store distances in a dataframe

  distances= pd.DataFrame(data=distances,columns=['dist'])

  return distances

