%matplotlib inline  
import pandas as pd  
import matplotlib.pyplot as pltfruits = pd.read\_table('fruit\_data\_with\_colors.txt')  
fruits.head()

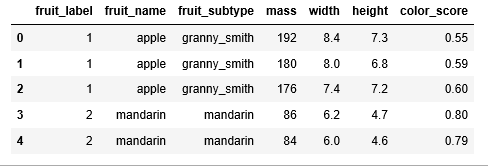


Figure 1

Each row of the dataset represents one piece of the fruit as represented by several features that are in the table’s columns.

We have 59 pieces of fruits and 7 features in the dataset:

print(fruits.shape)

***(59, 7)***

We have four types of fruits in the dataset:

print(fruits['fruit\_name'].unique())

***[‘apple’ ‘mandarin’ ‘orange’ ‘lemon’]***

The data is pretty balanced except mandarin. We will just have to go with it.

print(fruits.groupby('fruit\_name').size())

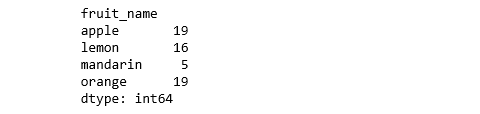


Figure 2

import seaborn as sns  
sns.countplot(fruits['fruit\_name'],label="Count")  
plt.show()

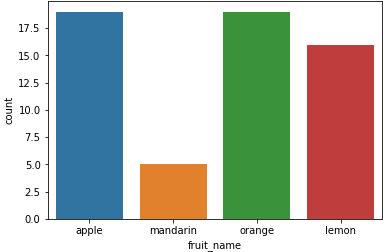


Figure 3

**Visualization**

* Box plot for each numeric variable will give us a clearer idea of the distribution of the input variables:

fruits.drop('fruit\_label', axis=1).plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False, figsize=(9,9),   
 title='Box Plot for each input variable')  
plt.savefig('fruits\_box')  
plt.show()

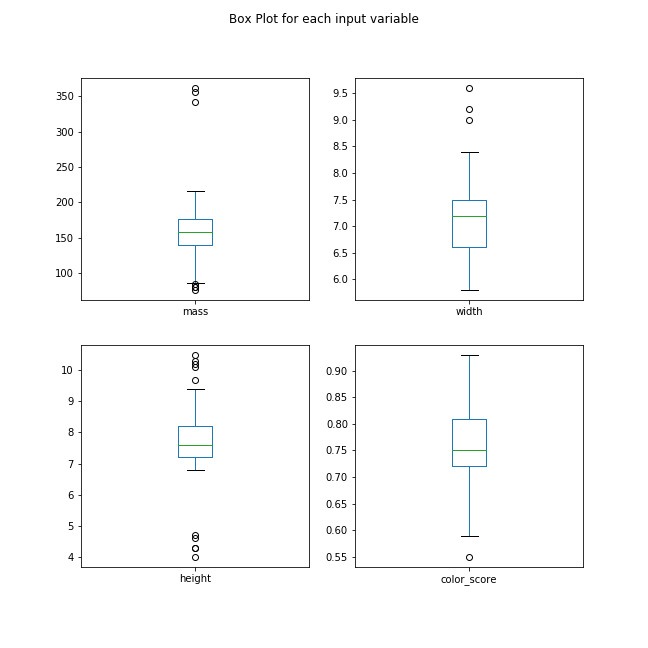


Figure 4

* It looks like perhaps color score has a near Gaussian distribution.

import pylab as pl  
fruits.drop('fruit\_label' ,axis=1).hist(bins=30, figsize=(9,9))  
pl.suptitle("Histogram for each numeric input variable")  
plt.savefig('fruits\_hist')  
plt.show()

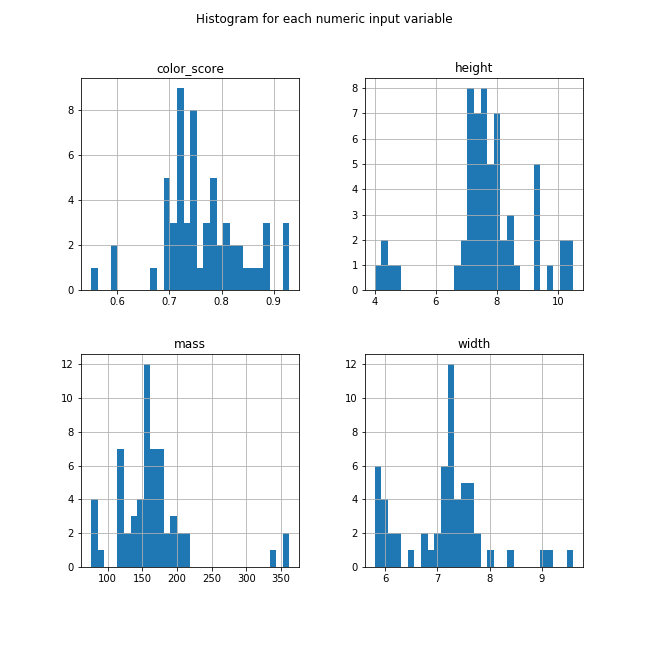


Figure 5

* Some pairs of attributes are correlated (mass and width). This suggests a high correlation and a predictable relationship.

from pandas.tools.plotting import scatter\_matrix  
from matplotlib import cmfeature\_names = ['mass', 'width', 'height', 'color\_score']  
X = fruits[feature\_names]  
y = fruits['fruit\_label']cmap = cm.get\_cmap('gnuplot')  
scatter = pd.scatter\_matrix(X, c = y, marker = 'o', s=40, hist\_kwds={'bins':15}, figsize=(9,9), cmap = cmap)  
plt.suptitle('Scatter-matrix for each input variable')  
plt.savefig('fruits\_scatter\_matrix')

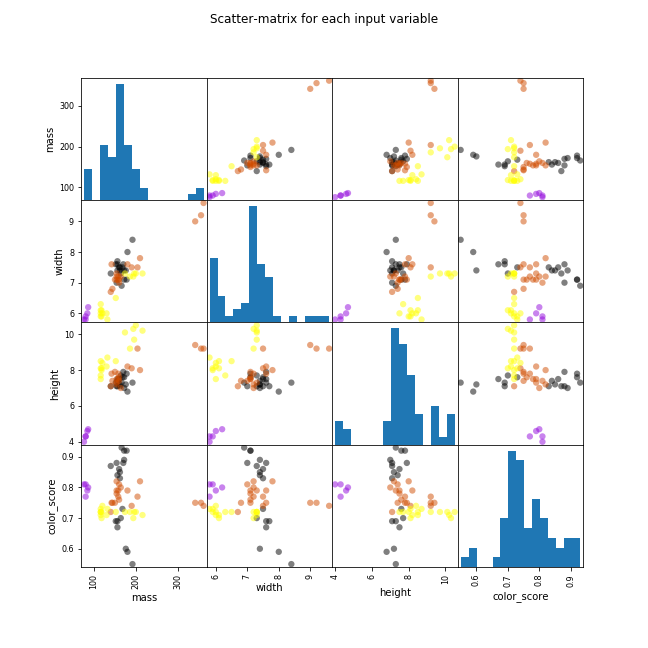


Figure 6

**Statistical Summary**

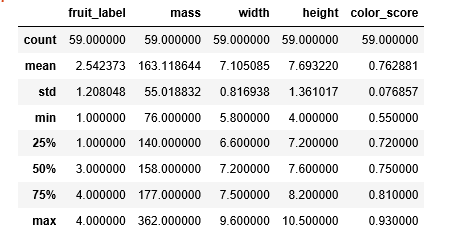


Figure 7

We can see that the numerical values do not have the same scale. We will need to apply scaling to the test set that we computed for the training set.

**Create Training and Test Sets and Apply Scaling**

from sklearn.model\_selection import train\_test\_splitX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)from sklearn.preprocessing import MinMaxScaler  
scaler = MinMaxScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)

**Build Models**

**Logistic Regression**

from sklearn.linear\_model import LogisticRegressionlogreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)print('Accuracy of Logistic regression classifier on training set: {:.2f}'  
 .format(logreg.score(X\_train, y\_train)))  
print('Accuracy of Logistic regression classifier on test set: {:.2f}'  
 .format(logreg.score(X\_test, y\_test)))

***Accuracy of Logistic regression classifier on training set: 0.70  
Accuracy of Logistic regression classifier on test set: 0.40***

**Decision Tree**

from sklearn.tree import DecisionTreeClassifierclf = DecisionTreeClassifier().fit(X\_train, y\_train)print('Accuracy of Decision Tree classifier on training set: {:.2f}'  
 .format(clf.score(X\_train, y\_train)))  
print('Accuracy of Decision Tree classifier on test set: {:.2f}'  
 .format(clf.score(X\_test, y\_test)))

***Accuracy of Decision Tree classifier on training set: 1.00  
Accuracy of Decision Tree classifier on test set: 0.73***

**K-Nearest Neighbors**

from sklearn.neighbors import KNeighborsClassifierknn = KNeighborsClassifier()  
knn.fit(X\_train, y\_train)  
print('Accuracy of K-NN classifier on training set: {:.2f}'  
 .format(knn.score(X\_train, y\_train)))  
print('Accuracy of K-NN classifier on test set: {:.2f}'  
 .format(knn.score(X\_test, y\_test)))

***Accuracy of K-NN classifier on training set: 0.95  
Accuracy of K-NN classifier on test set: 1.00***

**Linear Discriminant Analysis**

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysislda = LinearDiscriminantAnalysis()  
lda.fit(X\_train, y\_train)  
print('Accuracy of LDA classifier on training set: {:.2f}'  
 .format(lda.score(X\_train, y\_train)))  
print('Accuracy of LDA classifier on test set: {:.2f}'  
 .format(lda.score(X\_test, y\_test)))

***Accuracy of LDA classifier on training set: 0.86  
Accuracy of LDA classifier on test set: 0.67***

**Gaussian Naive Bayes**

from sklearn.naive\_bayes import GaussianNBgnb = GaussianNB()  
gnb.fit(X\_train, y\_train)  
print('Accuracy of GNB classifier on training set: {:.2f}'  
 .format(gnb.score(X\_train, y\_train)))  
print('Accuracy of GNB classifier on test set: {:.2f}'  
 .format(gnb.score(X\_test, y\_test)))

***Accuracy of GNB classifier on training set: 0.86  
Accuracy of GNB classifier on test set: 0.67***

**Support Vector Machine**

from sklearn.svm import SVCsvm = SVC()  
svm.fit(X\_train, y\_train)  
print('Accuracy of SVM classifier on training set: {:.2f}'  
 .format(svm.score(X\_train, y\_train)))  
print('Accuracy of SVM classifier on test set: {:.2f}'  
 .format(svm.score(X\_test, y\_test)))

***Accuracy of SVM classifier on training set: 0.61  
Accuracy of SVM classifier on test set: 0.33***

The KNN algorithm was the most accurate model that we tried. The confusion matrix provides an indication of no error made on the test set. However, the test set was very small.

from sklearn.metrics import classification\_report  
from sklearn.metrics import confusion\_matrix  
pred = knn.predict(X\_test)  
print(confusion\_matrix(y\_test, pred))  
print(classification\_report(y\_test, pred))

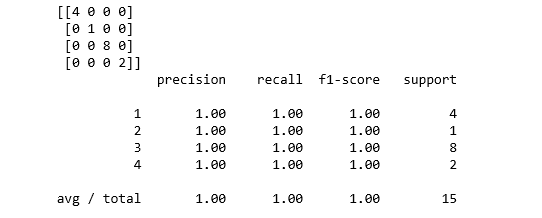


Figure 7

**Plot the Decision Boundary of the k-NN Classifier**

import matplotlib.cm as cm  
from matplotlib.colors import ListedColormap, BoundaryNorm  
import matplotlib.patches as mpatches  
import matplotlib.patches as mpatchesX = fruits[['mass', 'width', 'height', 'color\_score']]  
y = fruits['fruit\_label']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)def plot\_fruit\_knn(X, y, n\_neighbors, weights):  
 X\_mat = X[['height', 'width']].as\_matrix()  
 y\_mat = y.as\_matrix()# Create color maps  
 cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF','#AFAFAF'])  
 cmap\_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#AFAFAF'])clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=weights)  
 clf.fit(X\_mat, y\_mat)# Plot the decision boundary by assigning a color in the color map  
 # to each mesh point.  
   
 mesh\_step\_size = .01 # step size in the mesh  
 plot\_symbol\_size = 50  
   
 x\_min, x\_max = X\_mat[:, 0].min() - 1, X\_mat[:, 0].max() + 1  
 y\_min, y\_max = X\_mat[:, 1].min() - 1, X\_mat[:, 1].max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, mesh\_step\_size),  
 np.arange(y\_min, y\_max, mesh\_step\_size))  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])# Put the result into a color plot  
 Z = Z.reshape(xx.shape)  
 plt.figure()  
 plt.pcolormesh(xx, yy, Z, cmap=cmap\_light)# Plot training points  
 plt.scatter(X\_mat[:, 0], X\_mat[:, 1], s=plot\_symbol\_size, c=y, cmap=cmap\_bold, edgecolor = 'black')  
 plt.xlim(xx.min(), xx.max())  
 plt.ylim(yy.min(), yy.max())patch0 = mpatches.Patch(color='#FF0000', label='apple')  
 patch1 = mpatches.Patch(color='#00FF00', label='mandarin')  
 patch2 = mpatches.Patch(color='#0000FF', label='orange')  
 patch3 = mpatches.Patch(color='#AFAFAF', label='lemon')  
 plt.legend(handles=[patch0, patch1, patch2, patch3])plt.xlabel('height (cm)')  
plt.ylabel('width (cm)')  
plt.title("4-Class classification (k = %i, weights = '%s')"  
 % (n\_neighbors, weights))   
plt.show()plot\_fruit\_knn(X\_train, y\_train, 5, 'uniform')

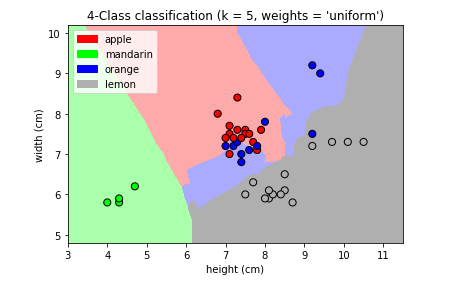


Figure 8

k\_range = range(1, 20)  
scores = []for k in k\_range:  
 knn = KNeighborsClassifier(n\_neighbors = k)  
 knn.fit(X\_train, y\_train)  
 scores.append(knn.score(X\_test, y\_test))  
plt.figure()  
plt.xlabel('k')  
plt.ylabel('accuracy')  
plt.scatter(k\_range, scores)  
plt.xticks([0,5,10,15,20])

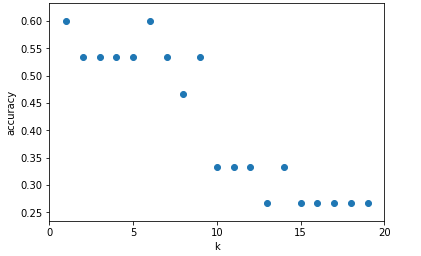


Figure 9

For this particular dateset, we obtain the highest accuracy when k=5.