```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as pltfruits =
pd.read_table('fruit_data_with_colors.txt')
fruits.head()
```

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

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())
```

```
fruit_name
apple       19
lemon       16
mandarin     5
orange      19
dtype: int64
```

Figure 2

```
import seaborn as sns
sns.countplot(fruits['fruit_name'],label="Count")
plt.show()
```

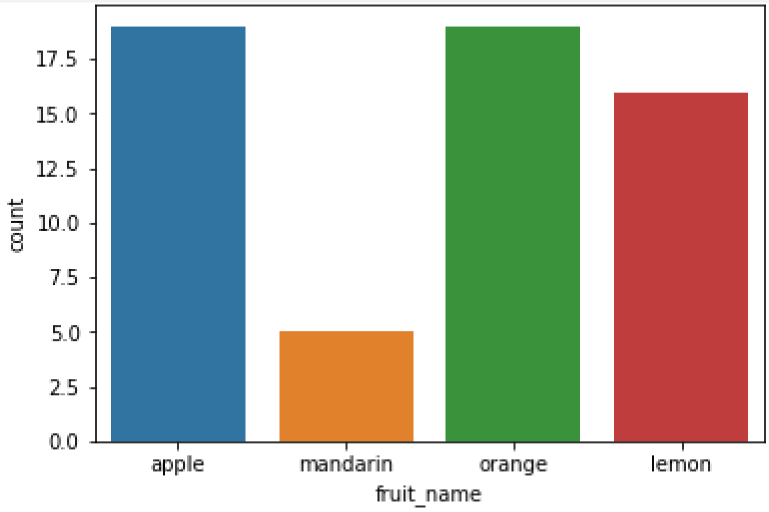

Figure 3

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()
```

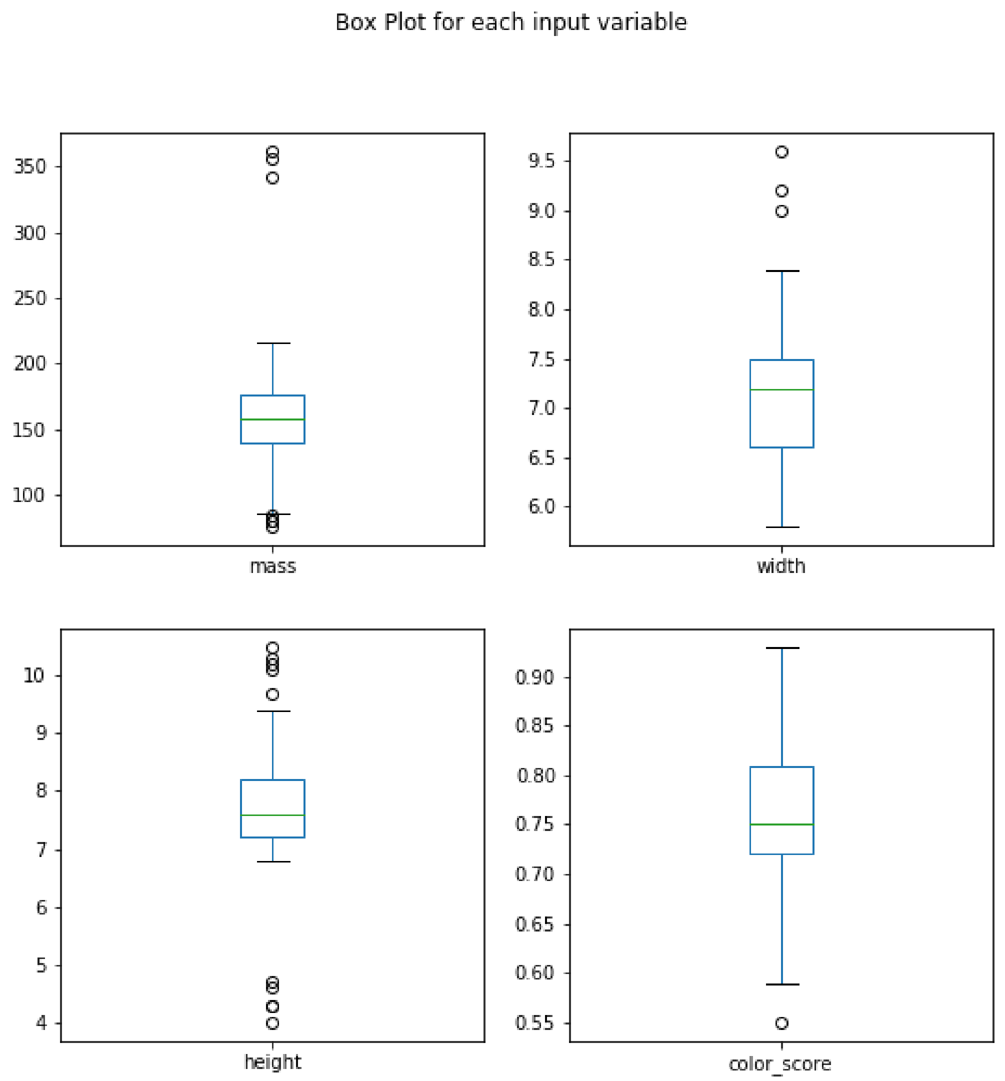

Figure 4

- 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()
```

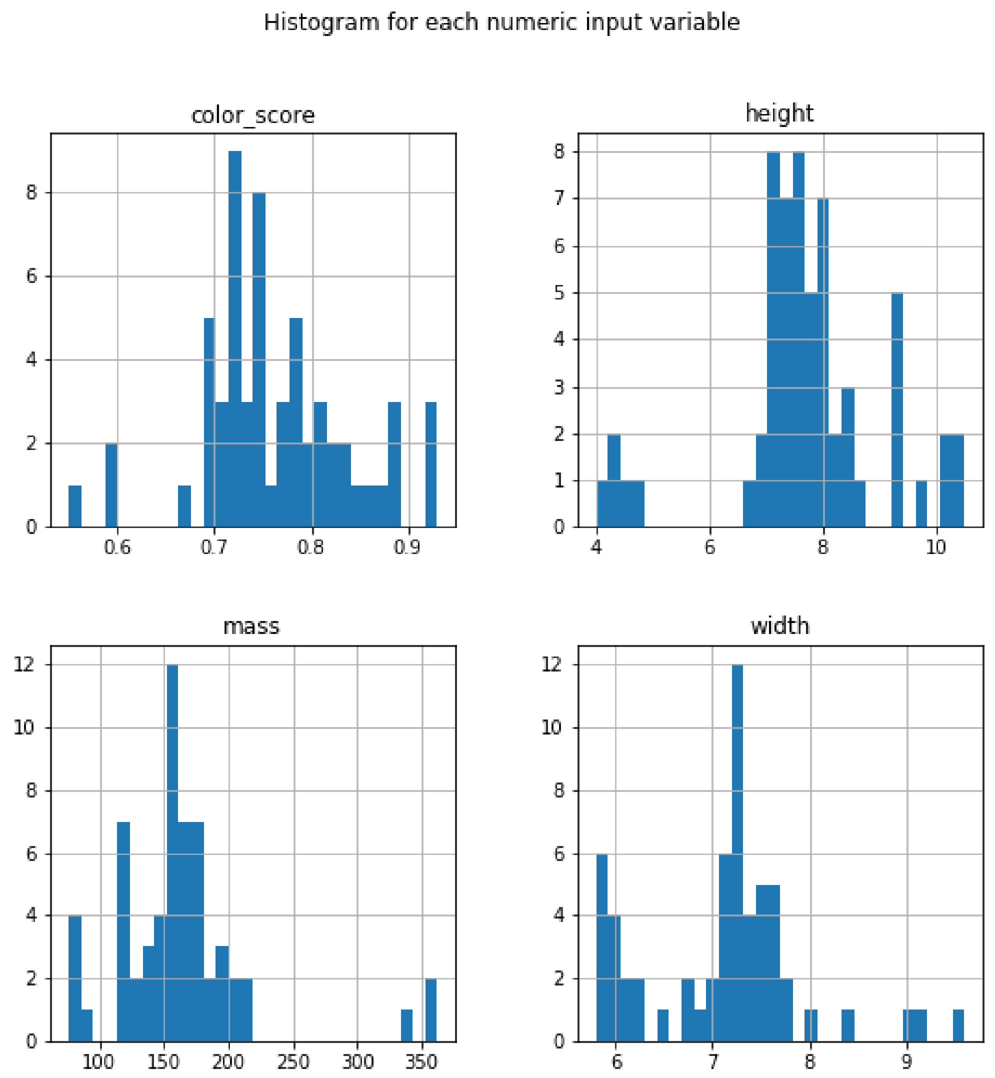

Figure 5

- 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')
```

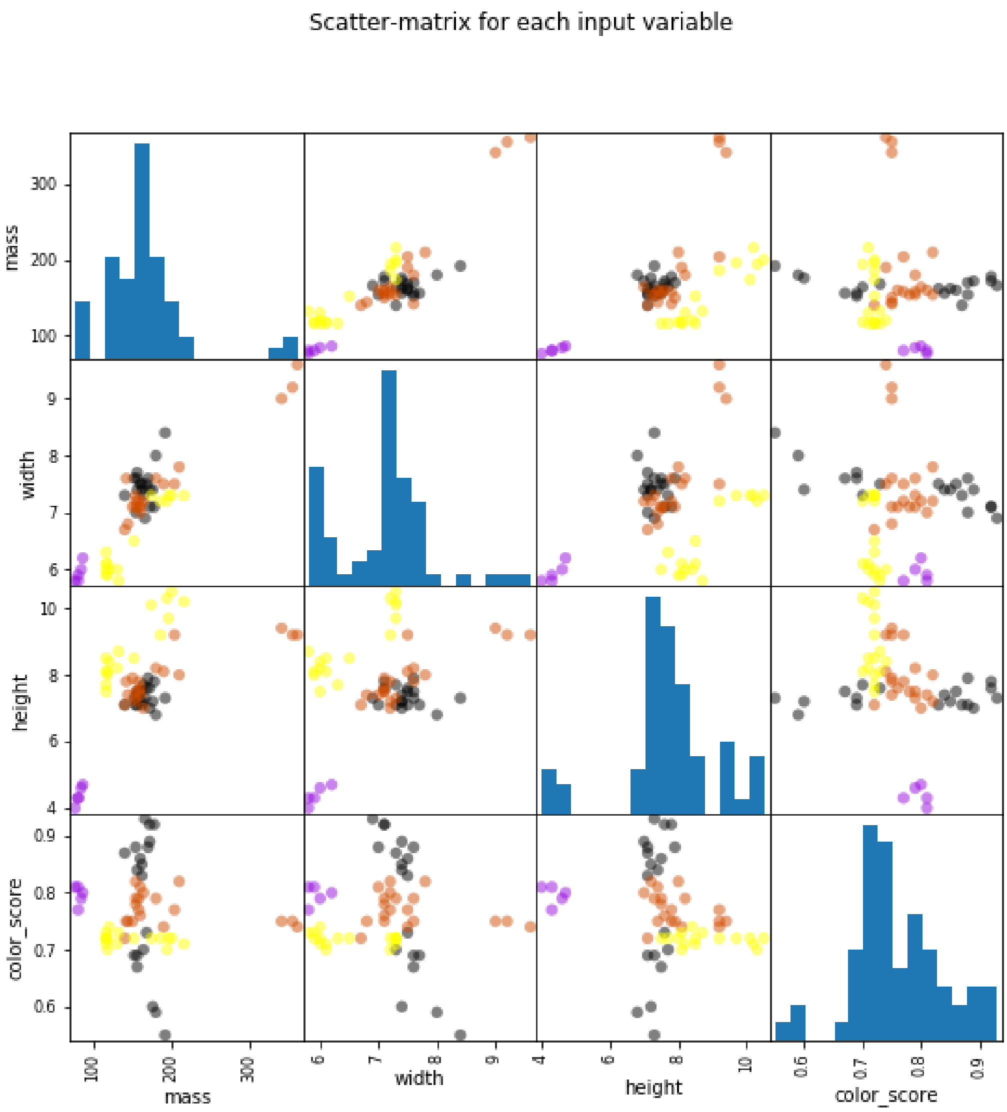

Figure 6

Statistical Summary

	fruit_label	mass	width	height	color_score
count	59.000000	59.000000	59.000000	59.000000	59.000000
mean	2.542373	163.118644	7.105085	7.693220	0.762881
std	1.208048	55.018832	0.816938	1.361017	0.076857
min	1.000000	76.000000	5.800000	4.000000	0.550000
25%	1.000000	140.000000	6.600000	7.200000	0.720000
50%	3.000000	158.000000	7.200000	7.600000	0.750000
75%	4.000000	177.000000	7.500000	8.200000	0.810000
max	4.000000	362.000000	9.600000	10.500000	0.930000

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))
```

```
[[4 0 0 0]
 [0 1 0 0]
 [0 0 8 0]
 [0 0 0 2]]
             precision    recall  f1-score   support

          1       1.00      1.00      1.00         4
          2       1.00      1.00      1.00         1
          3       1.00      1.00      1.00         8
          4       1.00      1.00      1.00         2

avg / total       1.00      1.00      1.00        15
```

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')
```

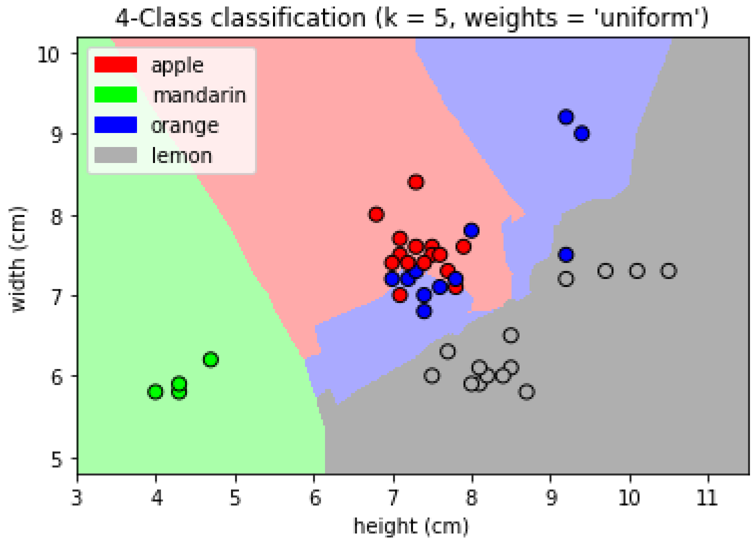

Figure 8

```
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])
```

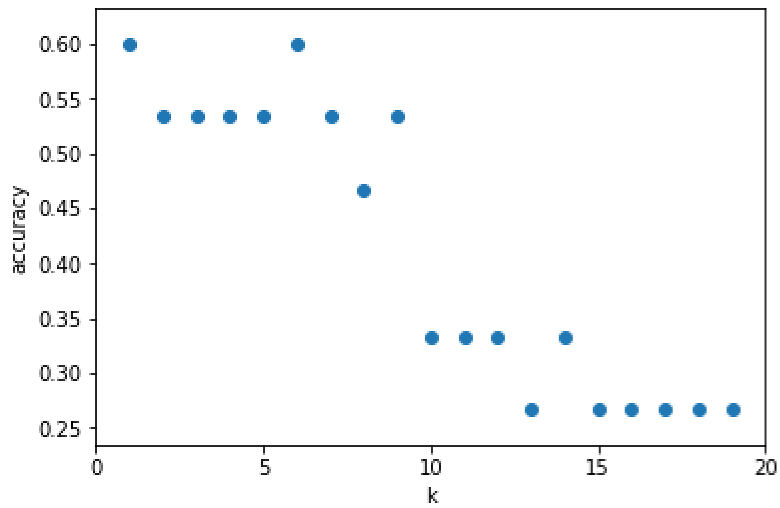

Figure 9

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