In [1]:

```
# Load needed libraries
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.externals import joblib
```

load the data

here we will load the data and interigate the content to understand the kind of data we are dealing with. This would normally be done by domain experts. For this example we will use the existing iris dataset to get facts about Iris Classification

In [2]:

```
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(url, names=names)

# print 10 rows to see the structure and dataset
#print(dataset.head(10))
dataset.head(10)
```

Out[2]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [3]:

```
# shape of the dataset tells use the number of rows vs columns
print(dataset.shape)
```

```
(150, 5)
```

In [4]:

```
# the .describe function will return some stats on the data in the array
print(dataset.describe())
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

In [5]:

```
# class distribution look at the number of instances (rows) that belong to each
 "class" (a particular column)
print(dataset.groupby('class').size())
```

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

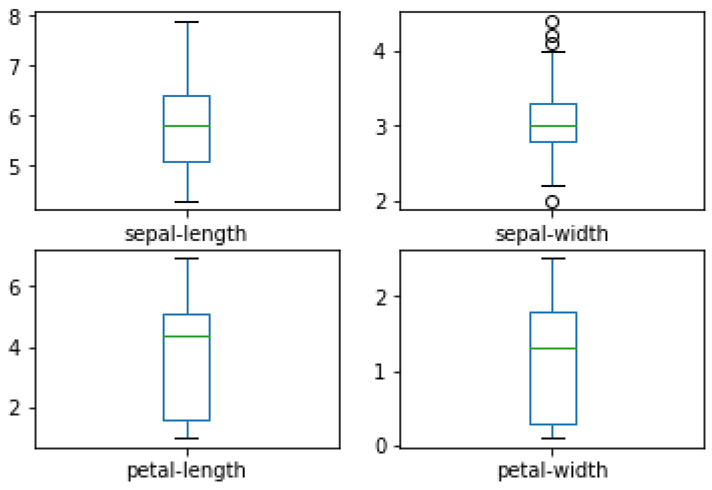
Univariate Plots

We start with some univariate plots, that is, plots of each individual variable. Given that the input variables are numeric, we can create box and whisker plots of each.

Then we can also create a histogram of each input variable to get an idea of the distribution.

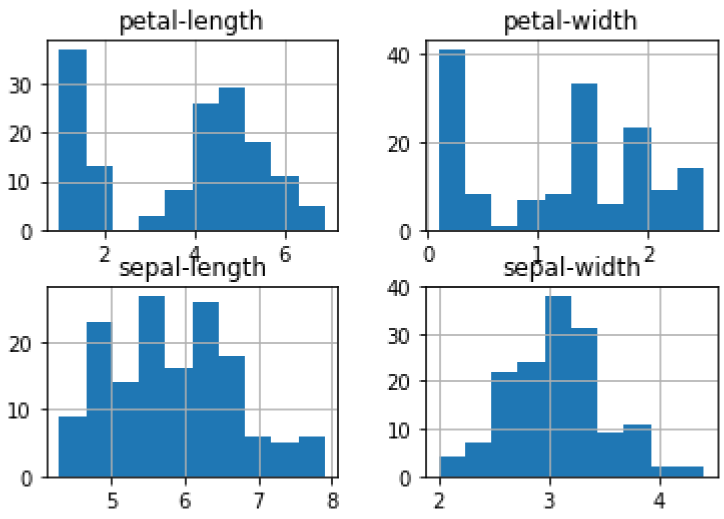
In [6]:

```
# box and whisker plots on the numeric colomns only
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False
)
plt.show()
```



In [7]:

```
# histograms
# It looks like perhaps two of the input variables have a Gaussian distribution.

# This is useful to note as we can use algorithms that can exploit this assumpti
on.

dataset.hist()
plt.show()
```



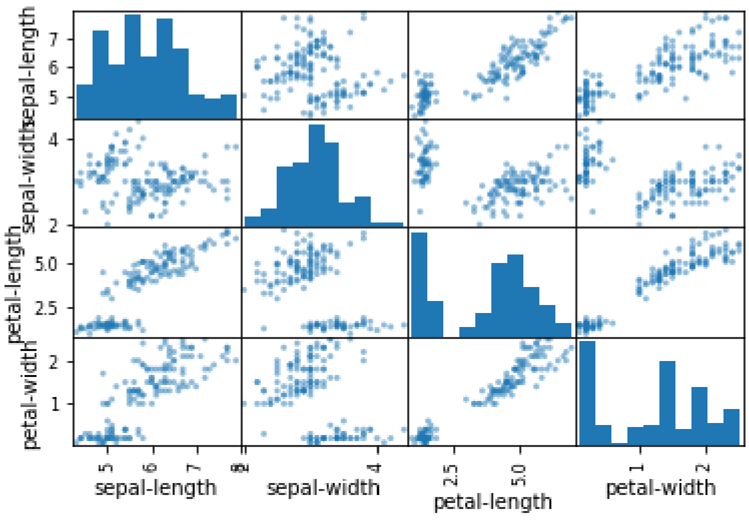
Multivariate Plots

Now we can look at the interactions between the variables.

First, let's look at scatterplots of all pairs of attributes. This can be helpful to spot structured relationships between input variables.

In [8]:

```
# scatter plot matrix
# Note the diagonal grouping of some pairs of attributes.
# This suggests a high correlation and a predictable relationship.
scatter_matrix(dataset)
plt.show()
```



Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data.

Here is what we are going to cover in this step:

Separate out a validation dataset. Set-up the test harness to use 10-fold cross validation. Build 5 different models to predict species from flower measurements. Select the best model.

In [9]:

```
# Split-out validation dataset for testing the ML
# We will split the loaded dataset into two, 80% of which we will use to train o
ur models
# and 20% that we will hold back as a validation dataset.


# Test options and evaluation metric
seed = 7 #randomiser can be any number
scoring = 'accuracy'

array = dataset.values #copy the whole data set to a new array
#print(array)
X = array[:,0:4] #remove last column
#print(X)
Y = array[:,4] #only pass the 5 colum, then answers
#print(Y)
validation_size = 0.20 #only use 20% of the whole data set

#get training data
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(
X, Y, test_size=validation_size, random_state=seed)
```

In [10]:

```
#X_train is the data of the iris without the classification from the original
#data set in a random row selection of the remainder of 20%
print(X_train.shape)
#Y_train is the matching classification row of the above array
print(Y_train.shape)


#X_validation & Y_validation are the same as above but the 20% randomly sellecte
d rows
print(X_validation.shape)
print(Y_validation.shape)

#print(X_validation)
#print(Y_validation)
```

```
(120, 4)
(120,)
(30, 4)
(30,)
```

Build Models

We don’t know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results.

Let’s evaluate 6 different algorithms:

Logistic Regression (LR) Linear Discriminant Analysis (LDA) K-Nearest Neighbors (KNN). Classification and Regression Trees (CART). Gaussian Naive Bayes (NB). Support Vector Machines (SVM). This is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

Let’s build and evaluate our five models:

In [11]:

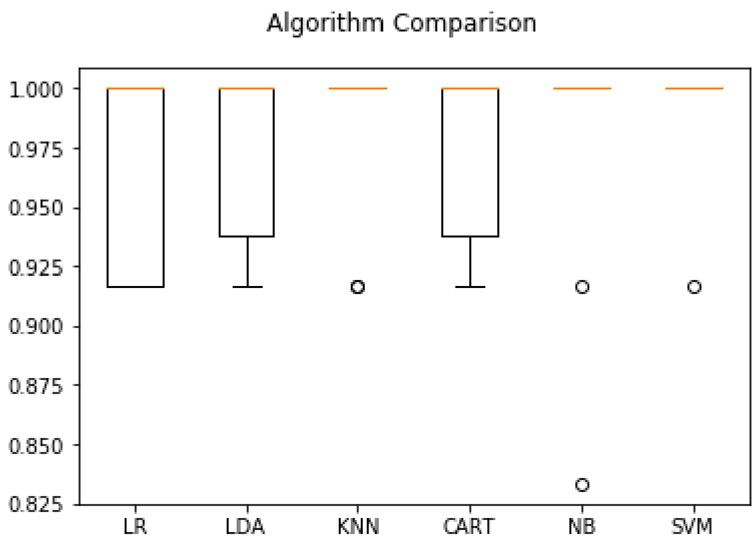
```
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
#print(models)

# evaluate each model in turn
results = []
names = []

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfo
ld, scoring=scoring)
    results.append(cv_results)
    #print(results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

#print(results)
```

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.975000 (0.038188)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (10 fold cross validation).

In [12]:

```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Make Predictions

The SVM algorithm was the most accurate model that we tested. Now we want to get an idea of the accuracy of the model on our validation set.

In [13]:

```
# Make predictions on validation dataset, not the testing set to see how accurat
e the model is
svm = SVC()
svm.fit(X_train, Y_train)
predictions = svm.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9333333333333333
[[ 7  0  0]
 [ 0 10  2]
 [ 0  0 11]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      0.83      0.91        12
 Iris-virginica       0.85      1.00      0.92        11

    avg / total       0.94      0.93      0.93        30
```

In [14]:

```
# Make predictions using next best model to see if initial prediction
# on the test set (20% of data) holds true with 80% of data
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

    avg / total       0.90      0.90      0.90        30
```

Save the MODEL

In [15]:

```
# Write the model to a file called "IRIS_Classifier_Model"
joblib.dump(svm, "IRIS_Classifier_Model", compress=9)
```

Out[15]:

```
['IRIS_Classifier_Model']
```