Basic ML Project (Data Visualization, Evaluating Algorithms, Making Predictions)

July 18, 2018

```
In [8]: #Load 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
```

```
In [9]: %matplotlib inline
```

```
In [10]: #Load dataset
         url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
         names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
         dataset = pandas.read_csv(url, names=names)
```

Now that the data has been imported using pandas, it is time to start exploring the data.

```
In [12]: #check what the dataset looks like in terms of columns and records
         dataset.shape
```

```
Out[12]: (150, 5)
```

```
In [13]: #check if there is are any missing data values
         dataset.isnull().values.any()
```

```
Out[13]: False
```

```
In [14]: #get the first ten records of the dataset
         dataset.head(10)
```

Out[14]:

	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 [15]:
```
#check the most common statistical descriptors, such as mean and standard deviation
dataset.describe()
```

Out[15]:

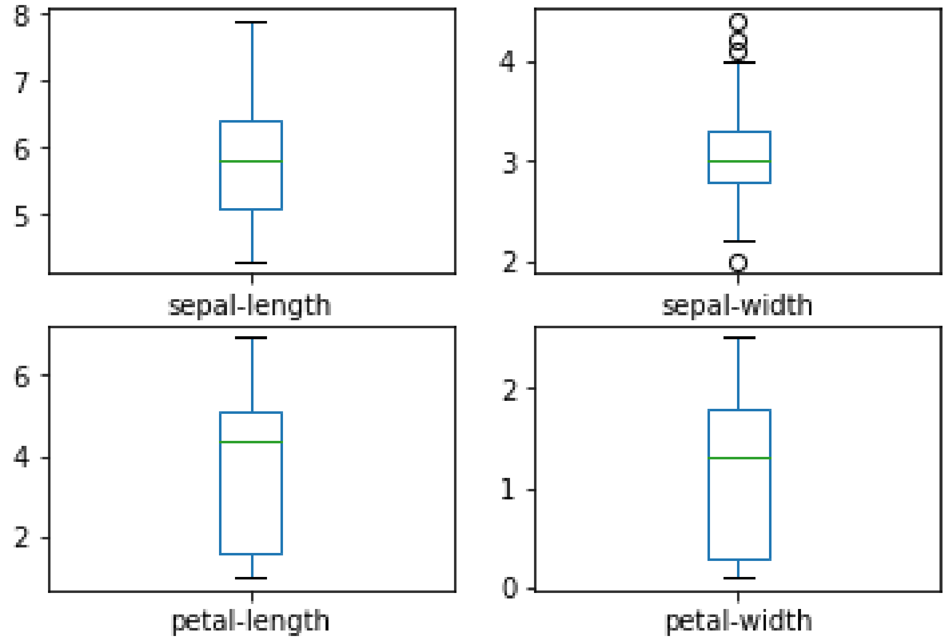
	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000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

In [16]:
```
#check how the data is distributed within the class column since this is what we want
dataset.groupby('class').size()
```

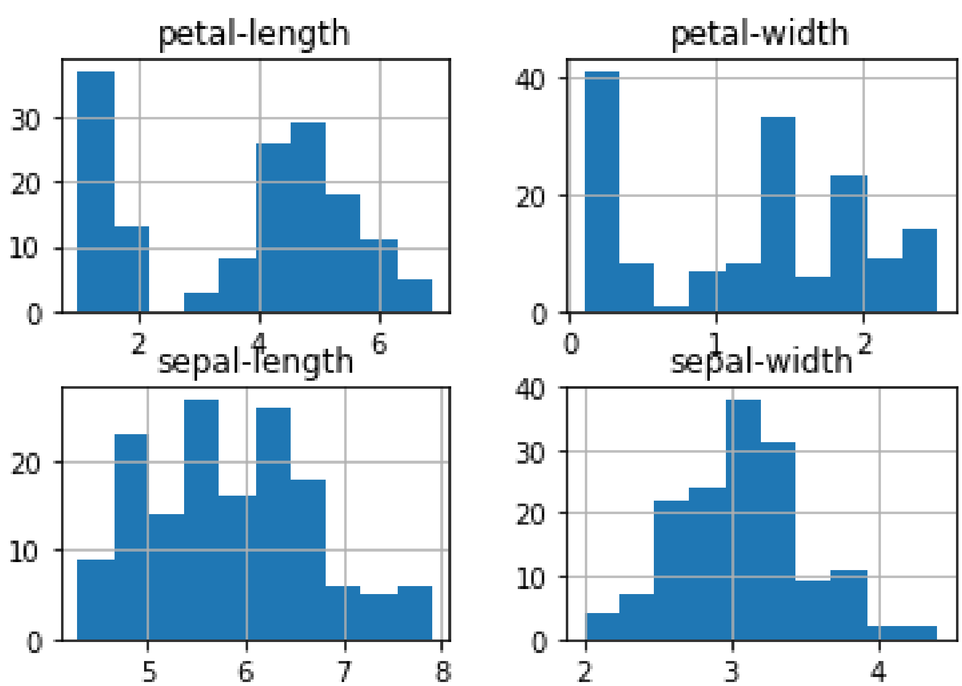
Out[16]:
```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Having looked at the data, found no null values, and gotten a better idea of what the data is, it is now time to start visualizing the data. In particular, this can help to see distributions and relationships better.

In [17]:
```
#box and whisker plots to show the distribution of the input attributes
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```

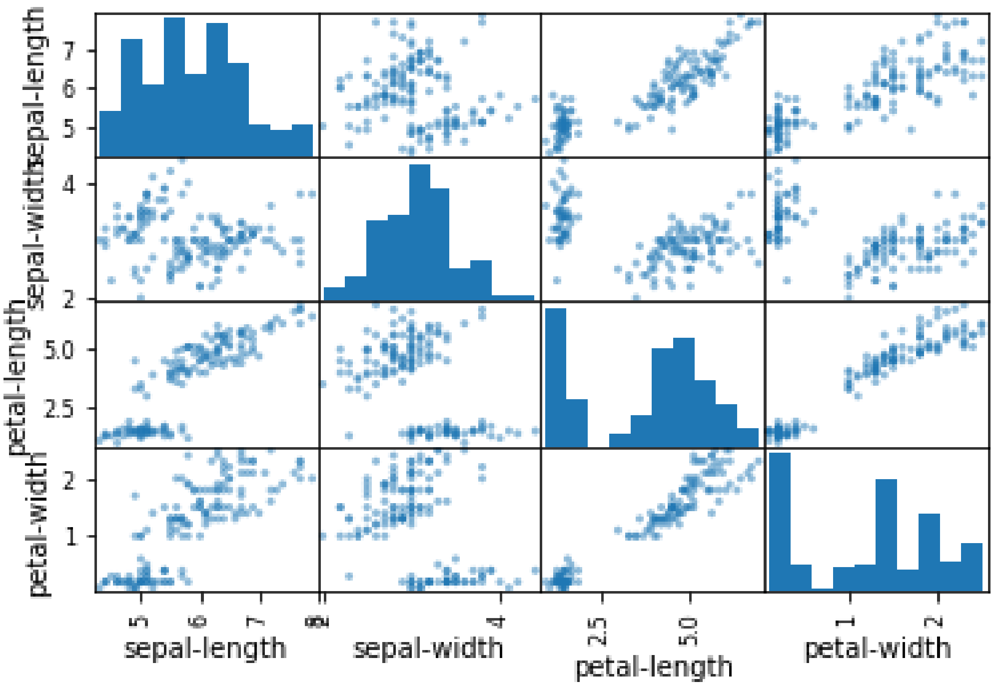



```
In [18]: #histograms also show the distribution
         dataset.hist()
         plt.show()
```



From the histograms it seems like sepal-length and sepal-width might be somewhat similar to the Normal Distributions, so this helps to determine what algorithms may be worth testing.

```
In [19]: #scatter plot matrix to see if there are relationships between the different attribut
         scatter_matrix(dataset)
         plt.show()
```



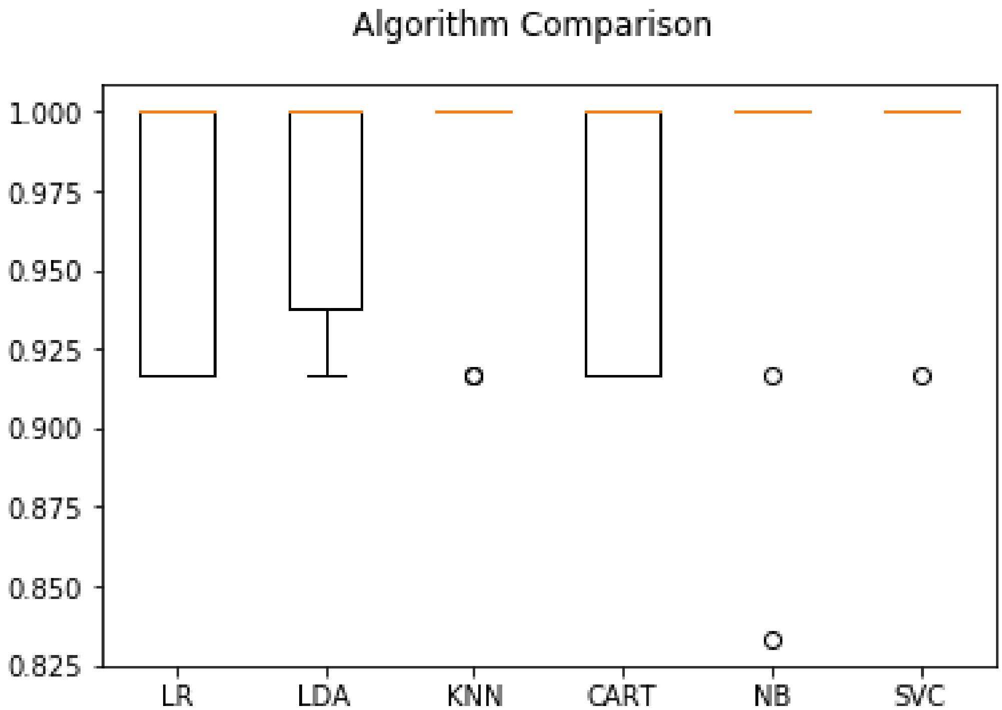
Some of the relationships seem to be positively correlated. In particular, petal-length and petal-width are highly correlated. Now that some of the distributions and relationships are more evident, it is time to split the dataset into training and validation data so that the accuracy of different ML models can be predicted, then the chosen model can be trained and validated.

```
In [20]: # Split-out validation dataset
         array = dataset.values
         X = array[:,0:4]
         Y = array[:,4]
         validation_size = 0.20
         seed = 7
         X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y,

In [21]: # Test options and evaluation metric
         seed = 3 #pseudorandom number for randomly selected the way the data is used for each
         scoring = 'accuracy'

In [22]: # 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(('SVC', SVC()))
# 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=kfold, sc
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

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

In [23]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```




```
In [24]: # Make predictions on validation dataset
         svc = SVC()
         svc.fit(X_train, Y_train)
         predictions = svc.predict(X_validation)
         print(accuracy_score(Y_validation, predictions))
         print(confusion_matrix(Y_validation, predictions))
         print(classification_report(Y_validation, predictions))
```

```
0.933333333333
[[ 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
```

The best algorithm was the support vector machine. Once selected, the next step was to train and test the algorithm and display the results. As seen in the above confusion matrix, it is evident that SVC has very high precision and high recall. This is likely in part to the small nature of the dataset, as SVC tends to be stronger and more powerful for smaller datasets.