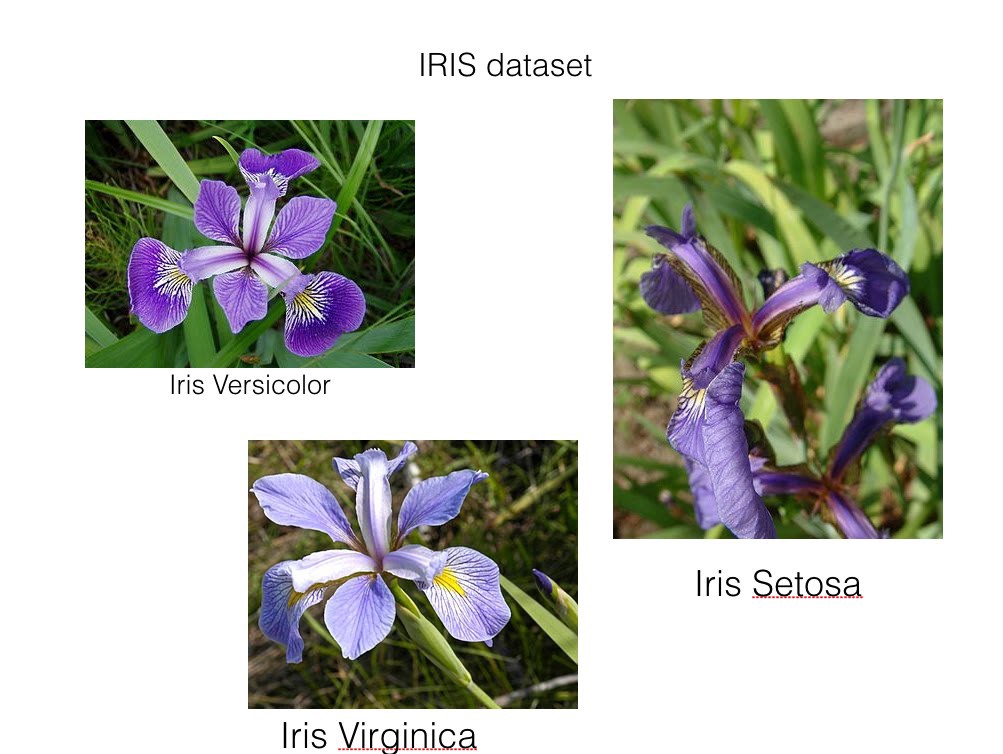
# Analysis of the Iris Data set Using Python

The Iris flower data is a multivariate [d](https://en.wikipedia.org/wiki/Data_set)ata set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis . It is sometimes called Anderson’s Iris data set because [Edgar Anderson](https://en.wikipedia.org/wiki/Edgar_Anderson) collected the data to quantify the morphologic variation of [Iris](https://en.wikipedia.org/wiki/Iris_(plant)) flowers of three related species. Two of the three species were collected in the [Gaspé Peninsula](https://en.wikipedia.org/wiki/Gaspé_Peninsula) “all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus”.



The data set consists of 50 samples from each of three species of Iris ([Iris setosa](https://en.wikipedia.org/wiki/Iris_setosa), [Iris virginica](https://en.wikipedia.org/wiki/Iris_virginica) and [Iris versicolor](https://en.wikipedia.org/wiki/Iris_versicolor)). Four [features](https://en.wikipedia.org/wiki/Features_(pattern_recognition)) were measured from each sample: the length and the width of the [sepals](https://en.wikipedia.org/wiki/Sepal) and [petals](https://en.wikipedia.org/wiki/Petal), in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other. (for more information ont the iris data set visit;

<https://en.wikipedia.org/wiki/Iris_flower_data_set>)

## The data set:

The data set contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

## Process:

import all of the modules, functions and objects:

import pandas

from pandas.tools.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

We are using pandas to load the data. We will also use pandas next to explore the data both with descriptive statistics and data visualization.

Note that we are specifying the names of each column when loading the data. This will help later when we explore the data.

dataset = pandas.read\_csv('iris\_dataset.csv')

After loading the data via pandas, we should checkout what the content is, description andvia the following:

dataset.head() #to check the first 10 rows of the data set

dataset.tail() #to check out last 10 row of the data set

dataset.describe() #to give a statistical summary about the dataset

dataframe.sample(5) #pops up 5 random rows from the data set

dataframe.isnull().sum() #checks out how many null info are on the dataset

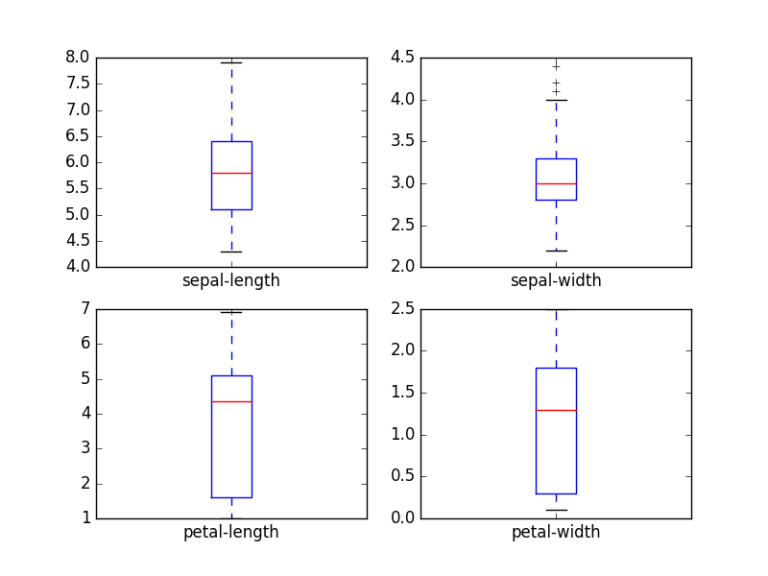
Now we visualize our data;

first with a boxplot which is going to be in the univariate form for each measurement.

# box and whisker plots

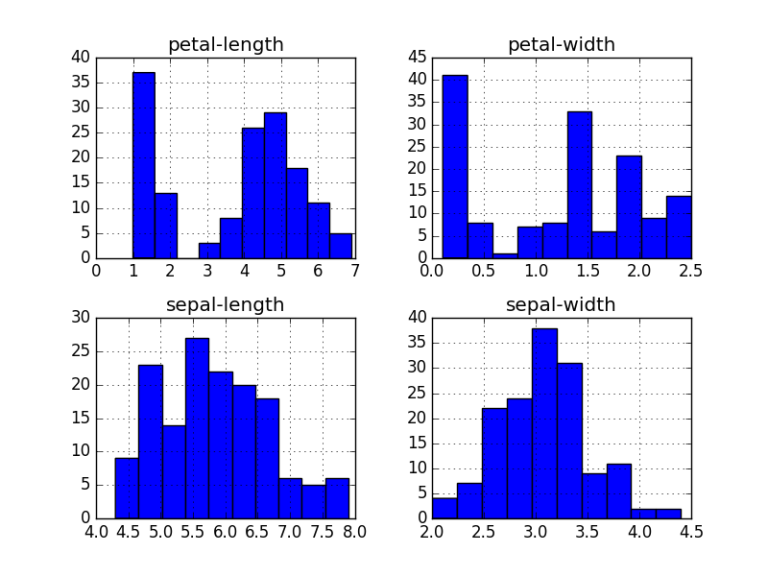
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)

plt.show()



We can also use histogram to analysis

# histogramsdataset.hist()plt.show()



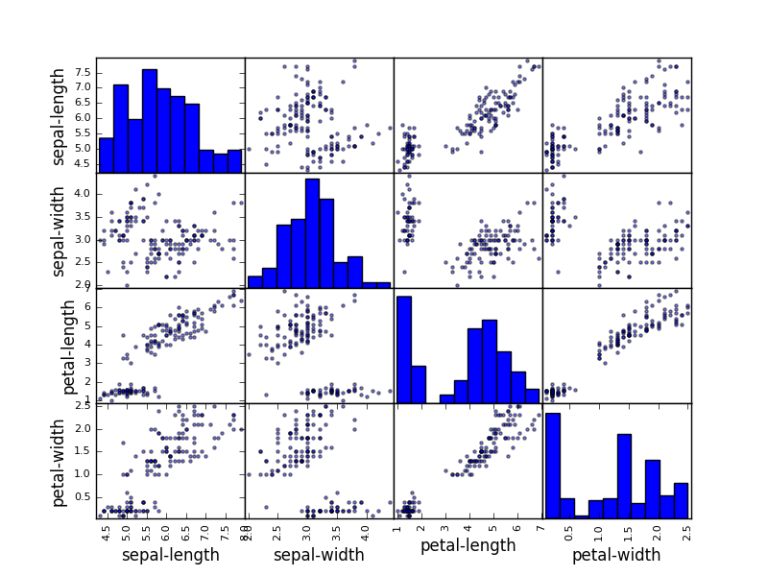
Now we can also look at the interactions between the variables.

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

# scatter plot matrix

scatter\_matrix(dataset)

plt.show()



From here we can create a validation set for our dataset:

# 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, test\_size=validation\_size, random\_state=seed)

We have splited the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

## Test Harness:

We will use 10-fold cross validation to estimate accuracy. This will split our dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

# Test options and evaluation metric

seed = 7

scoring = 'accuracy'

We are using the metric of ‘accuracy‘ to evaluate models. This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the scoring variable when we run build and evaluate each model next.

## Building Models:

We are going to test the following algorithms to know which one is the best to to take care of our data set:

* 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.

# 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()))

# 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, scoring=scoring)

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

We will have this output:

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: 0.981667 (0.025000)

Then we’ll choose the best algorithm: KNN seems to be the best with the value 0.983

## Make Predictions:

# Make predictions on validation dataset

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))

The accuracy is 0.9 or 90%. The confusion matrix provides an indication of the three errors made. Finally, the classification report provides a breakdown of each class by precision, recall, f1-score and support showing excellent results (granted the validation dataset was small).

#output

0.9[[ 7 0 0]

[ 0 11 1]

[ 0 2 9]]precision recall f1-score supportIris-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 11avg / total 0.90 0.90 0.90 30

We have been able to analyze and make predictions with these few basic steps. T