Machine Learning: k-nearest neighbors and CART

k-Nearest Neighbors

The k-nearest neighbors algorithm is among the simplest of all machine learning algorithms.

An object is classified by a majority vote of its neighbors with the object being assigned to the class most common among its k nearest neighbors.

K is a positive integer, typically small: k = 3, 5, or 10.

KNN can do multiple (more than two) class prediction.

In binary (two-class) classification problems, it is helpful to choose k to be an odd number as this can avoid a tie vote.

In [23]:
```
# Import Python Libraries: NumPy and Pandas
import pandas as pd
import numpy as np

# Import Libraries & modules for data visualization
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt

# Import scikit-Learn module for the algorithm/modeL: Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

# Import scikit-Learn module to split the dataset into train/ test sub-d
atasets
from sklearn.model_selection import train_test_split

# Import scikit-Learn module for K-fold cross-validation - algorithm/mod
eL evaluation & validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

# Import scikit-Learn module classification report to later use for info
rmation about how the system try to classify / lable each record
from sklearn.metrics import classification_report
```

Load the data

- Data Set: Iris.csv
- Title: Iris Plants Database
- Updated Sept 21 by C. Blake - Added discrepancy information
- Sources:

```
- Creator: RA_ Fisher
- Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
```

- Date: 1988

This is perhaps the best known database to be found in the pattern recognition literature. Fishers paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

- Predicted attribute: class of iris plant.
- Number of Instances: 150 (50 in each of three classes)
- Number of predictors: 4 numeric, predictive attributes and the class
- Attribute Information:
    1. sepal length in cm
    2. sepal width in cm
    3. petal length in cm
    4. petal width in cm
    5. class: Iris Setosa, Iris Versicolour, Iris Virginica

In [24]:
```
# Specify location of the dataset
dirpath = "/Users/jacobdenny/Desktop/Babloo/Masters/Semester3/ADTA5340/DataFiles/"
df = pd.read_csv(dirpath+"Iris.csv")
```

Preprocess Dataset

Clean Data: Find & Mark Missing Values

NOTES: The following columns cannot contain 0 (zero) values. i.e., 0 (zero) values are invalid in these columns SepalLengthCm float64 SepalWidthCm float64 PetalLengthCm float64 PetalWidthCm float64 It they exist, we need to mark them as missing value or numpy.NaN

The code below marks zero values as missing or NaN, count the number of NaN values in each column, get the dimensions or Shape of the dataset (i.e. number of records/rows x number of variables/columns), get the data types of all the variables/attributes of the data set. The results shows:

1) get several records/rows at the top of the dataset

2) get the first five records

3) get the summary statistics of the numeric variables/attributes of the dataset

In [25]:
```
# mark zero values as missing or NaN
df[[ 'SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm' ,'PetalWidthCm'
]] = df[['SepalLengthCm' , 'SepalWidthCm' ,'PetalLengthCm' , 'PetalWidth
Cm' ]].replace(0,np.NaN)
# count the number of NaN values in each column
print (df.isnull().sum())
#get familiarized with the data
print(df.head(5))
#Drop id column
df=df.drop(['Id'], axis=1)
print("\nAfter droping column Id \n",df.head(5))
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Sp
ecies
0   1            5.1           3.5            1.4           0.2  Iris-s
etosa
1   2            4.9           3.0            1.4           0.2  Iris-s
etosa
2   3            4.7           3.2            1.3           0.2  Iris-s
etosa
3   4            4.6           3.1            1.5           0.2  Iris-s
etosa
4   5            5.0           3.6            1.4           0.2  Iris-s
etosa

After droping column Id
    SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Speci
es
0            5.1           3.5            1.4           0.2  Iris-setos
a
1            4.9           3.0            1.4           0.2  Iris-setos
a
2            4.7           3.2            1.3           0.2  Iris-setos
a
3            4.6           3.1            1.5           0.2  Iris-setos
a
4            5.0           3.6            1.4           0.2  Iris-setos
a
```

Exploratory Data Analysis

In this step we are getting to know our data. The number of columns and rows, what each column represent etc. The following are explored in this section:

- Shape of data set
- Type of each value in the dataset
- View a snippet of the dataset
- Descriptive statistics of each column
- Univariate plots
- Multivariate plots

In [26]:
```
# get the dimensions or shape of the dataset
# i.e. number of records / rows X number of variables / columns
print(df.shape)
```

```
(150, 5)
```

We have 5 variables and 150 rows as we have deleted the column labeled Id.

In [27]:
```
#get the data types of all the variables / attributes in the data set
print(df.dtypes)
#return the first five records / rows of the data set
print(df.head(5))
```

```
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Specie
s
0            5.1           3.5            1.4           0.2  Iris-setos
a
1            4.9           3.0            1.4           0.2  Iris-setos
a
2            4.7           3.2            1.3           0.2  Iris-setos
a
3            4.6           3.1            1.5           0.2  Iris-setos
a
4            5.0           3.6            1.4           0.2  Iris-setos
a
```

All the features are of type floating point except species which is object type.

In [31]:
```
#return the summary statistics of the numeric variables / attributes in
 the data set
print(df.describe())
#class distribution i.e. how many records are in each class
print(df.groupby('Species').size())
```

```
       SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
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
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

The above table gives the summary statistics for each of the numeric columns.

This includes number of records, mean, standard deviation, minimum 25 percentile 50 percentile 75 percentile and the maximum values of each column namely SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm

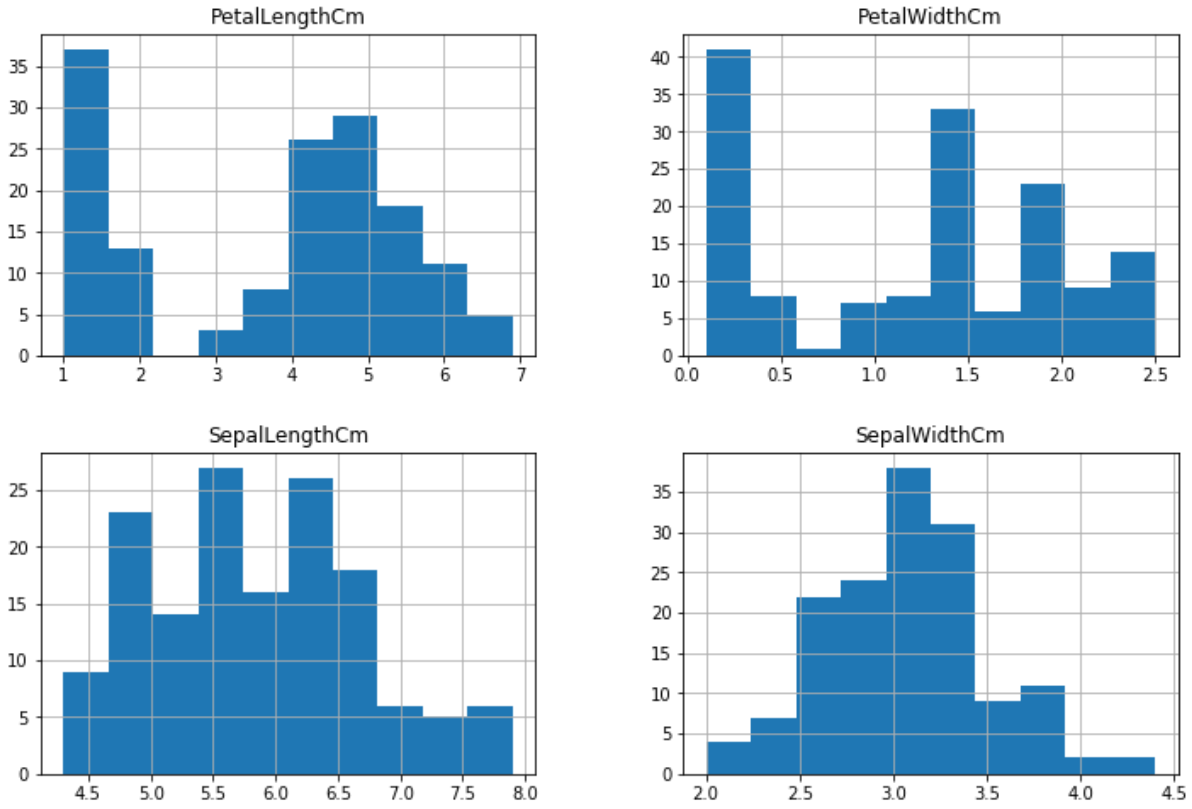
Univariate Plots

Plot each column to get an idea of the distribution. We will use the following

- Histogram
- Densityplot
- Boxplot

In [29]:
```
#Histogram
df.hist(figsize=(12,8))
plt.show
```

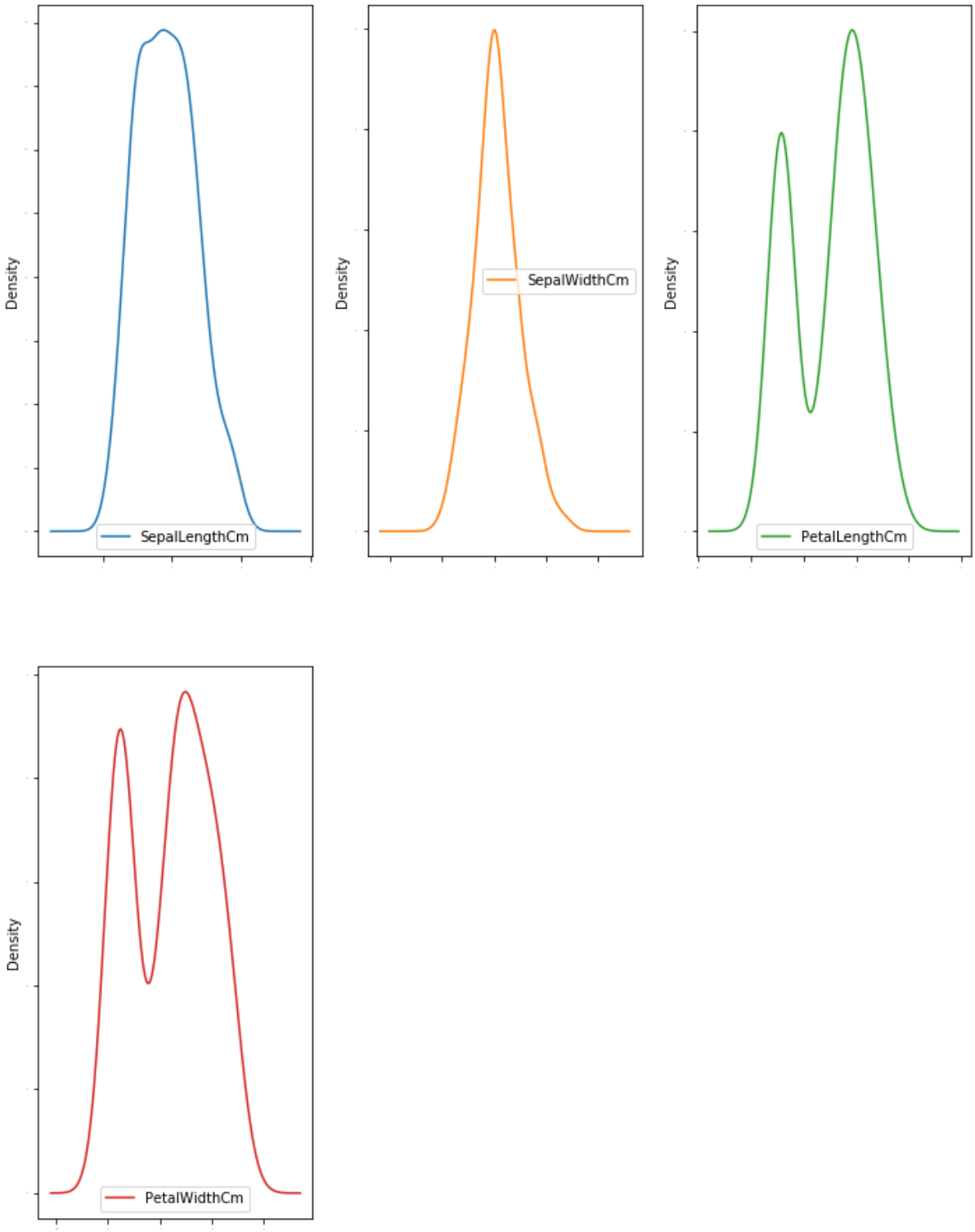
Out[29]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



What Histogram Tells About Data

The histograms are displayed for each column of the data. Sepallength and sepalwidth shows almost normal distribution, petal lenght and width seems to be bimodal

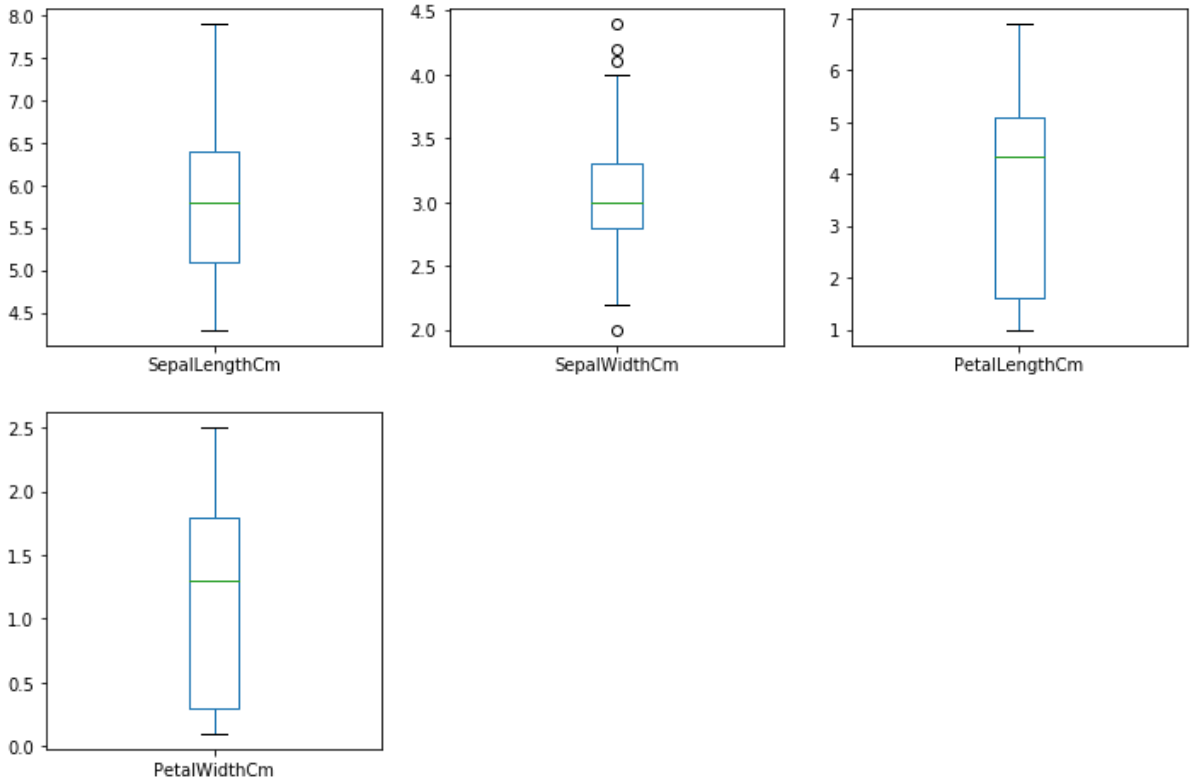
In [30]:
```
#Density plot
df.plot(kind="density", subplots = True, layout = (2,3), sharex = False,
legend = True, fontsize=1, figsize=(12,16) )
plt.show()
```



What Densityplots Tells About Data

The density plot is consistant with histogram and are displayed for each column of the data. Sepallength and sepalwidth shows almost normal distribution, petal lenght and width seems to be bimodal

In [32]:
```
#Boxplot
df.plot(kind="box", subplots = True, layout = (2,3), sharex = False, sha
rey = False, figsize=(12,8) )
plt.show()
```



What Boxplots Tells About Data

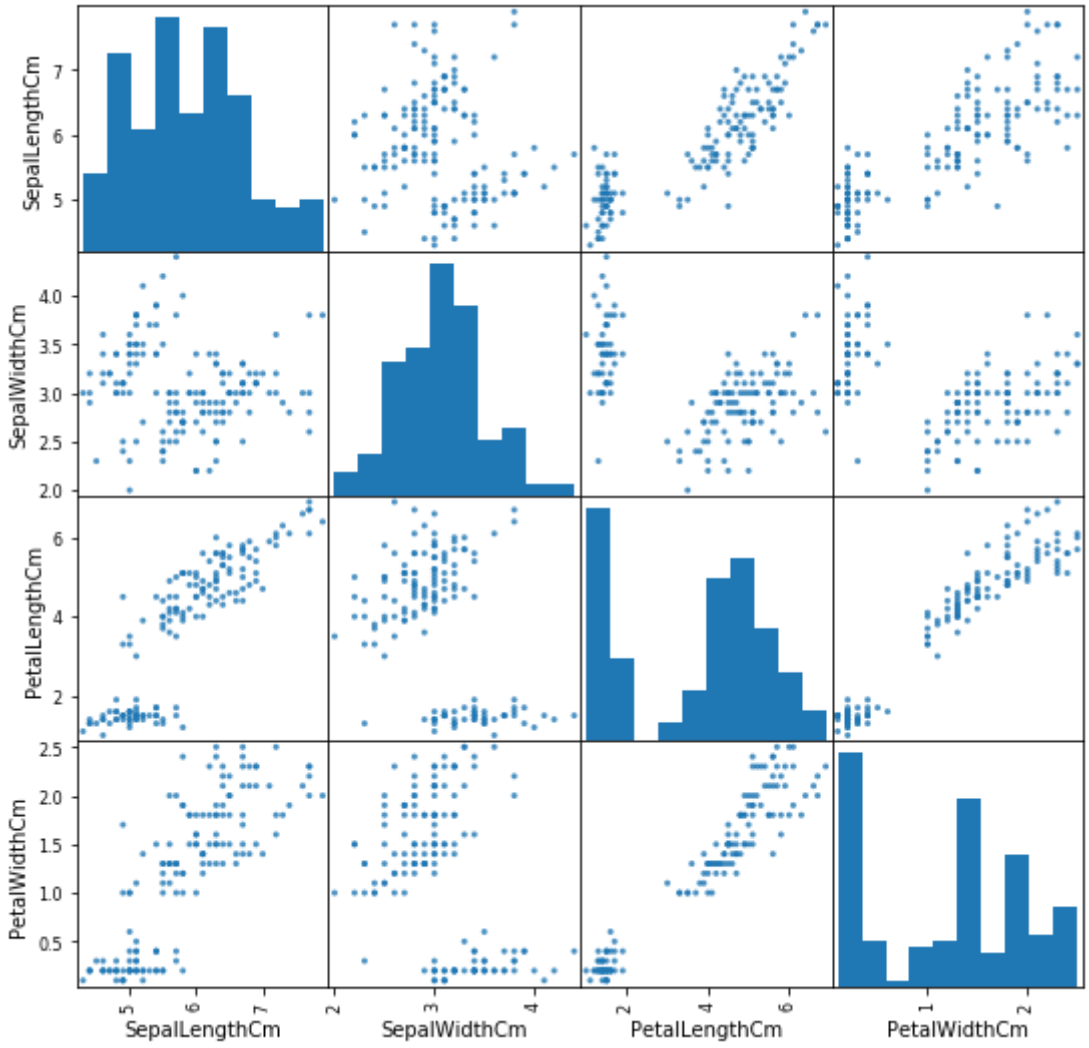
- For sepal length the values range from 4.5cm to 8cm
- For sepal width the values range from 2cm to 4.5cm
- For petal length the values range from 1cm to 7cm
- For petal widht the values range from 0 to 2.5cm
- Sepalwidth has evident outliers

Multivariate Analysis

The above plots shows characteristics of each column like distribution of data, range, dispersion or independent features of each column. Now we would like to see how each feature is affected by other features and we use multivariate analysis for this. Scatter matrix is the best technique for this analysis.

In [34]:
```
#Scatter matrix

scatter_matrix(df, alpha=0.8, figsize=(9,9))
plt.show()
```



What scatter matrix tells us

As sepal length increase petal length and width increase Sepal width exhibits almost null corelation with other features and hence has data points scattered all over Petal lenght and petal width also shows a positive linear corelation

Separate Dataset into Input & Output NumPy arrays

Training data Set (X) consist of the independent variables or predictors Desired Output (Y) consist of the dependent variable or that which we are trying to predict

In [35]:
```
# store dataframe values into a numpy array
array = df.values
# separate array into input and output by slicing
# for X(input) [:, 1:4] --> all the rows, columns from 1 - 5 (6 - 1)
# these are the independent variables or predictors
X = array[:,0:4]
# for Y(input) [:, 4] --> all the rows, column 5
# this is the value we are trying to predict
Y = array[:,4]
```

Split Input/Output Arrays into Training/Testing Datasets

In [36]:
```
# split the dataset --> training sub-dataset: 67%; test sub-dataset: 33%
test_size = 0.33
#selection of records to include in each data sub-dataset must be done r
andomly
seed = 7
#split the dataset (input and output) into training / test datasets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test
_size,
random_state=seed)
```

- 33% of the data is test data
- 67% of the data is train data
- Split is based on random sampling

Build and Train the Model

The code below build the Model, Train the model using the training sub-dataset, Print out the coefficients

and the intercept, Print intercept and coefficients, if we want to print out the list of coefficients with their correspondent variable name, pair the feature names with the coefficients and convert iterator in to set

In [37]:
```
#build the model
model = KNeighborsClassifier()
# train the model using the training sub-dataset
model.fit(X_train, Y_train)
#print the classification report
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.85      0.94      0.89        18
 Iris-virginica       0.94      0.83      0.88        18

       accuracy                           0.92        50
      macro avg       0.93      0.93      0.93        50
   weighted avg       0.92      0.92      0.92        50
```

A note on classification report - precision, recall, F1-score and support

Precision

The precision is the ratio tp / (tp + fp) --> where tp is the number of true positives and fp the number of false positives The precision represents the ability of the classifier not to label a positive sample as negative

Recall

The recall is the ratio tp / (tp + fn) --> where tp is the number of true positives and fn the number of false negatives The recall represents the ability of the classifier to find all the positive samples

F-beta

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall --> where an F-beta score reaches its best value at 1 and worst score at 0 The F-beta score weights recall more than precision by a factor of beta beta == 1.0 means recall and precision are equally important

Support

The support is the number of occurrences of each class in y_true

We have perfect prediction of the Iris-setosa

For Iris-versicolor, 94% Iris-versicolor were classified correctly but 6% were classified as Iris-versicolor wrongly(from recall). Also, out of total Iris-versicolor 85% were classified correctly but 15 were missclassified(from precision)

For Iris-virginica , 83% Iris-virginica were classified correctly but 17% were classified as Iris-virginica wrongly(from recall). Also, out of total Iris-virginica 94% were classified correctly but 6 were missclassified(from precision)

Total accuracy of model predictions= 92 % meaning the prediction was correct 92% of times

Score the accuracy of the model

In [39]:
```
#score the accuracy leve
result = model.score(X_test, Y_test)
#print out the results
print(("Accuracy: %.3f%%") % (result*100.0))
```

```
Accuracy: 92.000%
```

The model correctly classifies 90% of the times

Classify/Predict Model

Now use the model to predict the group of a given flower with given measurents for different features The new record has the following predictors:

- sepal length in cm = 5.3
- sepal Width in cm = 3.0
- petal length in cm = 4.5
- petal width in cm 1.5

```
In [41]: model.predict([[5.3, 3.0, 4.5, 1.5]])
```

```
Out[41]: array(['Iris-versicolor'], dtype=object)
```

The model prediction for the given feaures is : Iris-versicolor

Based on the model's accuracy score: there is 92% chance that this new record is a Iris-versicolor.

Evaluate the model using the 10-fold cross-validation technique.

- Specify the number of time of repeated splitting,
- Fix the random seed, Must use the same seed value so that the same subset can be obtained for each time the process is repeated,
- Split the whole data set into folds, for logistic regression
- Use accuracy level to evaluate the model/algorithm, train the model
- Run k-fold cross-validation to validate/evaluate the model,
- Print out the evaluation results

In [42]:
```
# evaluate the algorithm
# specify the number of time of repeated splitting, in this case 10 folds
n_splits = 10
# fix the random seed
# must use the same seed value so that the same subsets can be obtained
# for each time the process is repeated
seed = 7
# split the whole dataset into folds
'''In k-fold cross-validation, the original sample is randomly partitioned into k equal sized
subsamples. Of the k subsamples, a single subsample is retained as the validation data for
testing the model, and the remaining k - 1 subsamples are used as training data. The crossvalidation
process is then repeated k times, with each of the k subsamples used exactly once as
the validation data. The k results can then be averaged to produce a single estimation. The
advantage of this method over repeated random sub-sampling is that all observations are used for
both training and validation, and each observation is used for validation exactly once
'''
kfold = KFold(n_splits, random_state=seed)
# for kNN, we can use the accuracy level to evaluate the model / algorithm
scoring = 'accuracy'
# train the model and run K-fold cross validation to validate / evaluate the model
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
# print the evaluationm results
# result: the average of all the results obtained from the K-fold cross validation
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
Accuracy: 0.933 (0.084)
```

- using the 10-fold cross-validation to evaluate the model / algorithm, the accuracy of this logistic regression model is 93.3%. It increased the model accuracy from 92 for a single sample to 93.3% using k-fold validation

Above, the model predicted the flower type of the new record as Iris-versicolor. According to the model's accuracy score obtained from the model evaluation using 10-told cross-validation there is 93.3% chance that this new record is an Iris-versicolor as opposed to 92% obtained by single sampling done earlier.

Comparison to Last weeks Logistic Regression Model

The logistic regression model accuracy was only 88% after k-fold validation and hence the chance of prediction being true was only 88%. But the K-nearest neighbour classification has an accuracy of 93% hence the chance of a prediction being true is more compared to logistic regression.

Hence the prediction for the new record that has the following predictors:

sepal length in cm = 5.3 sepal Width in cm = 3.0 petal length in cm = 4.5 petal width in cm 1.5

being Iris-versicolor (predicted through kNN model) is 5% more than it being Iris-virginica(predicted by logistic regression model)

CART Regression with Scikit-Learn

Classification and Regression Tree (CART): CART Regression

Decision tree builds regression or classification models in the form of a tree structure. Regression trees are needed when the response variable is numeric or continuous.

- For example, the predicted price of a consumer good.

Thus regression trees are applicable for prediction type of problems as opposed to classification

- In either case, the predictors or independent variables may be categorical or numeric
- It is the target variable that determines the type of decision tree needed

Data Set

We will investigate the Boston House Price dataset

Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970.

The attributes are defined as follows:

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 tract bounds river; 0 otherwise)
5. NOX nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS weighted distances to five Boston employment centers
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per 10,000 dollars
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0:63)^2 where Bk is the proportion of blocks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in 1000 dollars

For this study: we use a subset of the original dataset.

1. RM: average number of rooms per dwelling
2. AGE: proportion of owner-occupied units built prior to 1940
3. DIS weighted distances to five Boston employment centers
4. RAD: index of accessibility to radial highways
5. PTRATIO: pupil-teacher ratio by town
6. MEDV: Median value of owner-occupied homes in 1000 dollars

Import Python Libraries and Modules

In [43]:
```
# Import scit-Learn module for the algorithm/model: Linear Regression
from sklearn. tree import DecisionTreeRegressor
```

Load Dataset

In [44]:
```
#Load data into a pandas dataframe
dirpath = "/Users/jacobdenny/Desktop/Babloo/Masters/Semester3/ADTA5340/DataFiles/"
df = pd.read_csv(dirpath+"housing_boston_w_hdrs.csv")

# Not assigning column headers, as this dataset has headers
#get familiarized with the data
df.head(5)
```

Out[44]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

Preprocessing Data

We are trying to predict the median value of owner occupied houses in 1000 dollors(MEDV) using CART(Decision Tree) regression with 'RM', 'AGE', 'DIS', 'RAD', 'PTRATIO' as predictors. Hence the subset is extracted from dataframe and is stored into another data frame. Also, from domain knowledge we know that the average value of a home, the pupil teacher ratio and average number of rooms per dwelling cannot be zero. If any row cell contain zero for these three columns that needs to be marked as mising. So preprocessing includes:

- Extract the columns under consideration
- Data cleaning
- Check for invalid Values

In [45]:
```
# Extract a sub-dataset from the original one -- > dataframe: df2
df2 = df[['RM', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'MEDV']]
#Data cleaning
#Replace 0 value with NaN for columns 'RM', 'PTRATIO', 'MEDV'
df[['RM', 'PTRATIO', 'MEDV']] = df[['RM', 'PTRATIO', 'MEDV']].replace(0,
np.NaN)
#Verify there are no missing values in the dataset
print(df.isnull().sum())
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

There are no missing values in the dataset. Now we are ready to move to Exploratory data analysis

Exploratory Data Analysis

In this step we are getting to know our data. The number of columns and rows, what each column represent etc. The following are explored in this section:

- Shape of data set
- Type of each value in the dataset
- View a snippet of the dataset
- Descriptive statistics of each column
- Univariate plots
- Multivariate plots

In [46]:
```
#SHAPE
# Get the dimensions or Shape of the dataset
# i.e. number of records/rows x number of variables/columns
print(df2.shape)
```

```
(452, 6)
```

We have 6 variables and 452 rows.

In [47]:
```
#TYPE
# Get the data types of all variabLes/attributes of the data set
# The results show
print(df2.dtypes)
```

```
RM          float64
AGE         float64
DIS         float64
RAD           int64
PTRATIO     float64
MEDV        float64
dtype: object
```

All the features and the dependent variable(MEDV) are of type floating point except RAD which is integer type.

In [49]:
```
#View of a snippet of Dataset
# Get the first five records
print(df2.head(5))
```

```
      RM   AGE     DIS  RAD  PTRATIO  MEDV
0  6.575  65.2  4.0900    1     15.3  24.0
1  6.421  78.9  4.9671    2     17.8  21.6
2  7.185  61.1  4.9671    2     17.8  34.7
3  6.998  45.8  6.0622    3     18.7  33.4
4  7.147  54.2  6.0622    3     18.7  36.2
```

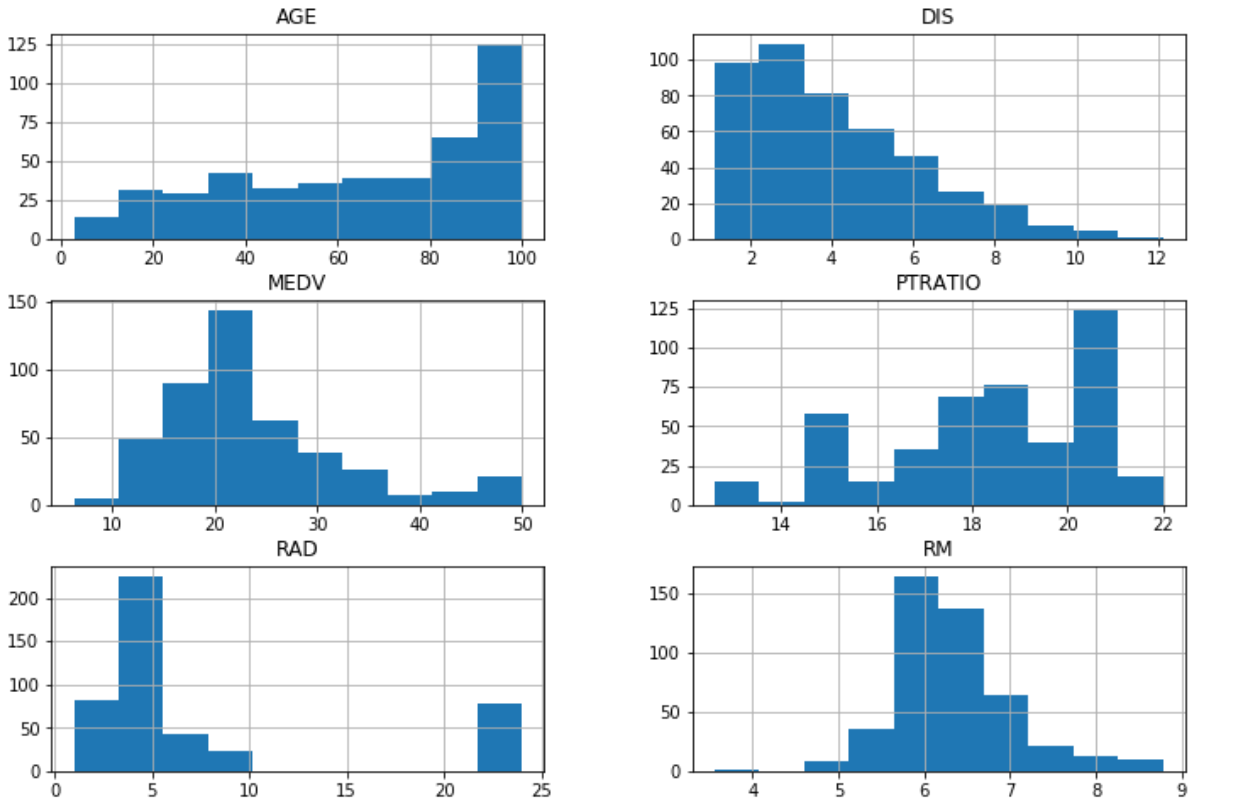
In [50]:
```
#Summary Statistics
# Get the summary statistics of the numeric variables/attributes of the
 dataset
print(df2.describe())
```

```
               RM         AGE         DIS         RAD     PTRATIO
MEDV
count  452.000000  452.000000  452.000000  452.000000  452.000000  452.
000000
mean     6.343538   65.557965    4.043570    7.823009   18.247124   23.
750442
std      0.666808   28.127025    2.090492    7.543494    2.200064    8.
808602
min      3.561000    2.900000    1.129600    1.000000   12.600000    6.
300000
25%      5.926750   40.950000    2.354750    4.000000   16.800000   18.
500000
50%      6.229000   71.800000    3.550400    5.000000   18.600000   21.
950000
75%      6.635000   91.625000    5.401100    7.000000   20.200000   26.
600000
max      8.780000  100.000000   12.126500   24.000000   22.000000   50.
000000
```

The above table gives the summary statistics for each of the columns. This includes number of records, mean, standard deviation, minimum 25 percentile 50 percentile 75 percentile and the maximum values of each column namely RM, AGE, DIS, RAD, PTRATIIO and MEDV

Histograms

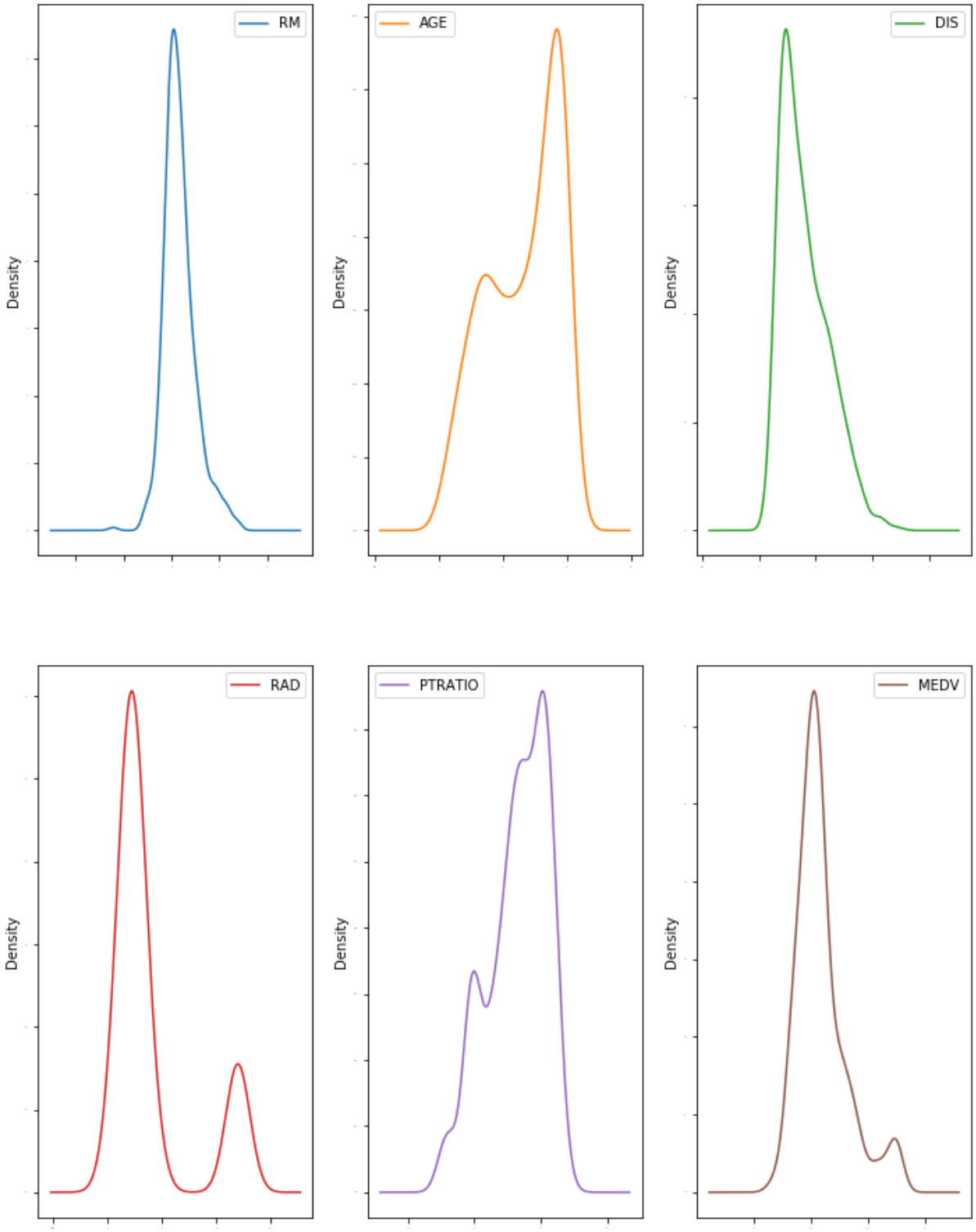
In [51]:
```
# Plot histrogram for each numeric
df2.hist(figsize=(12, 8))
plt.show()
```



MEDV,PTRATIO and RM has a distribution that is almost normal but the rest are not.

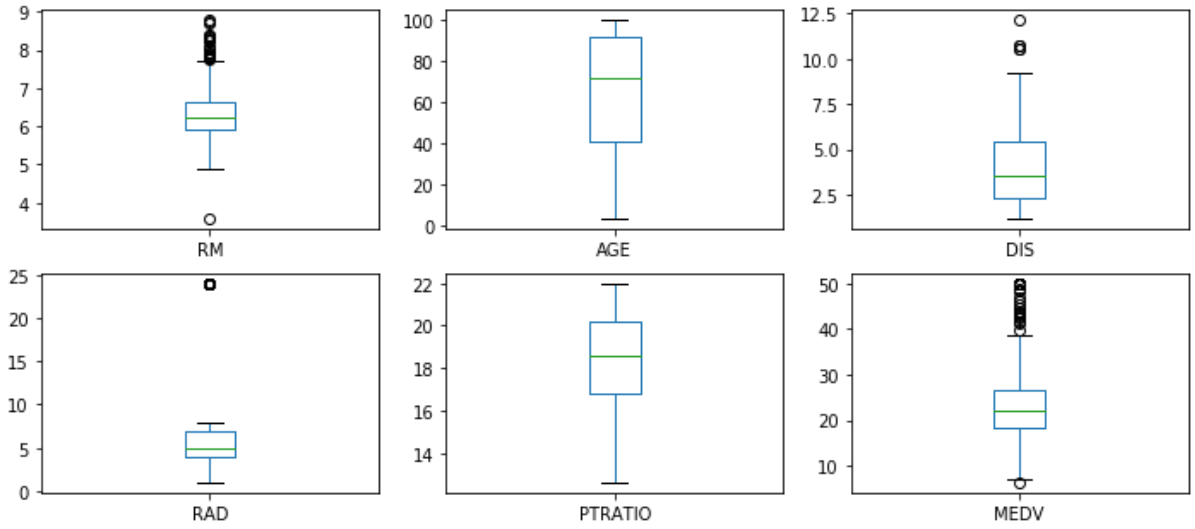
Density Plots

In [53]:
```
# Density plots
# IMPORTANT NOTES: 5 numeric variables -> at Least 5 plots -> Layout (2, 3): 2 rows, each row with 3 plots
df2.plot(kind='density', subplots=True, layout=(2, 3), sharex=False, legend=True, fontsize=1,
figsize=(12, 16))
plt.show()
```



Consistent with the histogram, density plots also gives an almost normal distribution for MEDV,PTRATIO and RM. but DIS is also fairly normal in density plot. But RAD and AGE are bimodal.

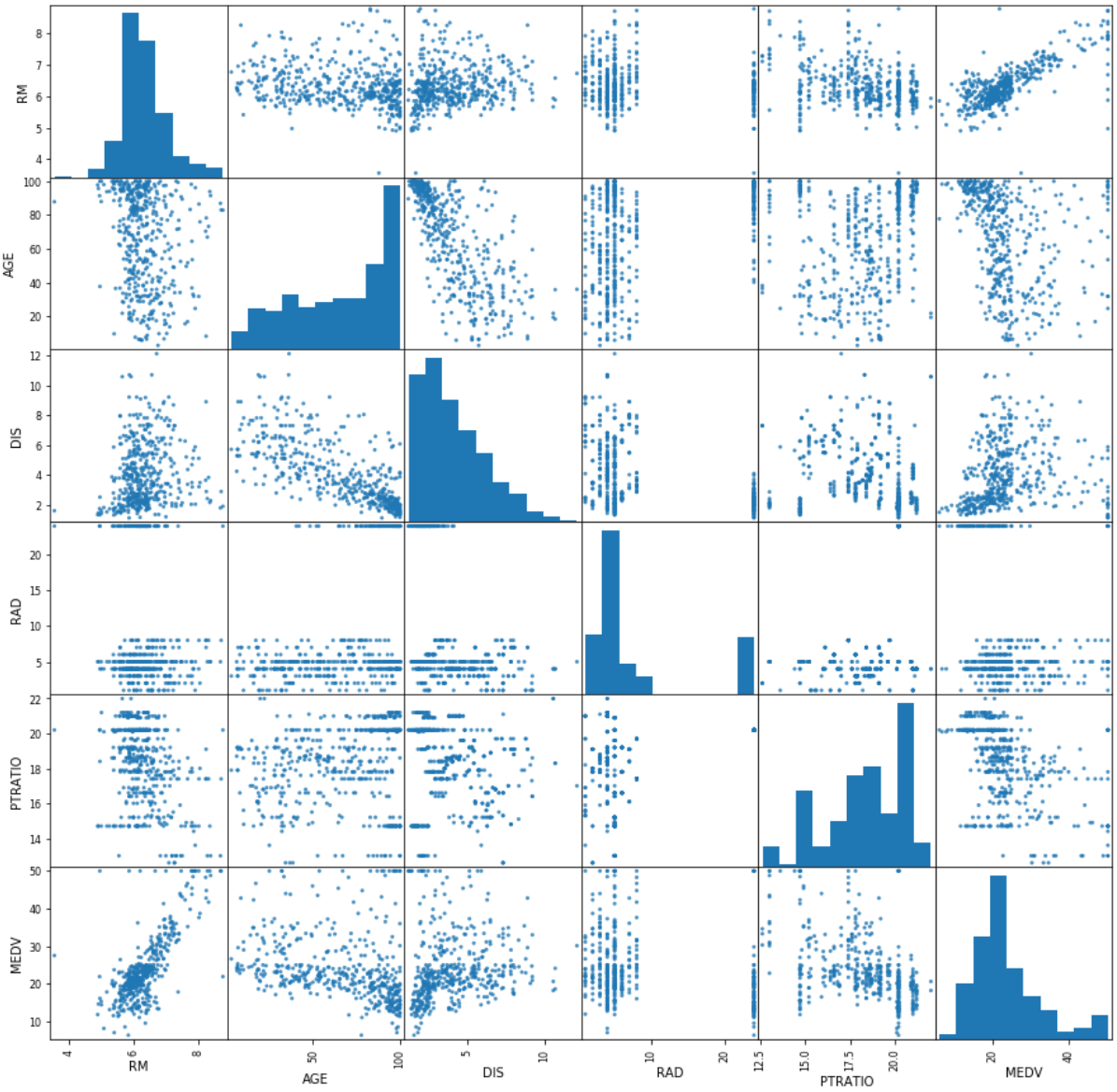
In [54]:
```
df2.plot(kind='box', subplots=True, layout=(3,3), sharex=False, figsize=
(12,8))
plt.show()
```



- RAD has a single outlier
- MEDV, RM and DIS has outliers at both ends
- Min value of RM is 1 and max at 9
- Min value od AGE is 4 and max just below 100
- Min value of DIS is 1 and max just below 12.5
- Min value of RAD is 1 and max just below 10 with an outlier at 25
- Min value of PTRATIO is 5 and max just below 22
- Min value of MEDV is 41 and max just above 50

Multivariate Analysis : Scatter Matrix

In [55]:
```
# scatter plot matrix
scatter_matrix(df2, alpha=0.8, figsize=(15, 15))
plt.show()
```



- There is an evident linear relation between MEDV and RM
- MEDV and AGE is showing an inverse linear relation
- DIS and MEDV on the other hand has a direct linear relation

Separate Dataset into Input & Output NumPy Arrays

In [90]:
```
# Store dataframe values into a numpy array
array = df2.values
# separate array into input and output components by slicing
# For X (input)[:, 5] --> all the rows, columns from 0 - 4 (5 - 1)
X = array[:,0:5]
# For Y (output)[:, 5] --> all the rows, column index 5 (Last column)
Y = array[:,5]
```

- InDependent variables are 'RM', 'AGE', 'DIS', 'RAD' and 'PTRATIO' columns 0 to 4
- Dependent variable is 'MEDV' column 5

Split Input/Output Arrays into Training/Testing Datasets

In [91]:
```
# Split the dataset --> training sub-dataset: 67%; test sub-dataset:
test_size = 0.33
# Selection of records to include in which sub-dataset must be done rand
omly
# use this seed for randomization
seed = 7
# Split the dataset (both input & outout) into training/testing datasets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test
_size,
random_state=seed)
```

- 33% of the data is test data
- 67% of the data is train data
- Split is based on random sampling

Build and Train the Model

In [92]:
```
# Build the model
model = DecisionTreeRegressor()
# Train the model using the training sub-dataset
model.fit(X_train,Y_train)
# Non-Linear --> NO coefficients and the intercept
DecisionTreeRegressor (criterion='mse', max_depth=None, max_features=Non
e, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0
,presort=False, random_state=None,
                       splitter='best')
```

Out[92]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=Non
e,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.
0,
                      presort=False, random_state=None, splitter='bes
t')
```

In [93]:
```
R_squared = model.score(X_test, Y_test)
print(R_squared)
```

```
0.5342088995226266
```

The model has successfully explained 53 % of variance in MEDV using variables 'RM', 'AGE','DIS','RAD' and 'PTRATIO'

Prediction

We have trained the model. Let's use the trained model to predict the "Median value of owner-occupied homes in 1000 dollars" (MEDV)_

The suburb area has the following predictors:

- RM: average number of rooms per dwelling = 6.0
- AGE: proportion of owner-occupied units built prior to 1940 = 55
- DIS: weighted distances to five Boston employment centers = 5
- RAD: index of accessibility to radial highways = 2
- PTRATIO: pupil-teacher ratio by town = 16

In [94]:
```
model.predict([[6.0, 55, 5, 2, 16]])
```

Out[94]:
```
array([20.4])
```

Predicted Median value of owner-occupied homes in 1000 dollars for the above house is : 23.60419508093679 Thousand

Evaluate/Validate Algorithm/Model Using K-Fold Cross-Validation

In [96]:
```
# Evaluate the algorithm
# Specify the K-size
num_folds = 10
# Fix the random seed
# must use the same seed value so that the same subsets can be obtained
# for each time the process is repeated
seed = 7
# Split the whole data set into folds
kfold = KFold(n_splits=num_folds, random_state=seed)
# For Linear regression, we can use MSE (mean squared error) value
# to evaluate the model/algorithm
scoring = 'neg_mean_squared_error'
# Train the model and run K-foLd cross-validation to validate/evaluate t
he model
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
# Print out the evaluation results
# Result: the average of all the results obtained from the k-foLd cross-
validation
print(results.mean())
```

```
-39.41912898550724
```

After we train we evaluate Use K-Fold to determine if the model is acceptable. We pass the whole set because the system will divide for us. -39 avg of all error (mean of square errors) this value would traditionally be positive value, but scikit reports as neg Square root would be between 6 and 7

With the test train split we calculated an R square of 53 but that is applicable to only that sample. Where as in K-fold validation with number of folds set to 10, we have 10 different accuracies which averages to 39.4. K-fold validation describes the population better than a predicting on single sample from population and that deceases the accuracy from 53 to 39.4.

In [ ]: