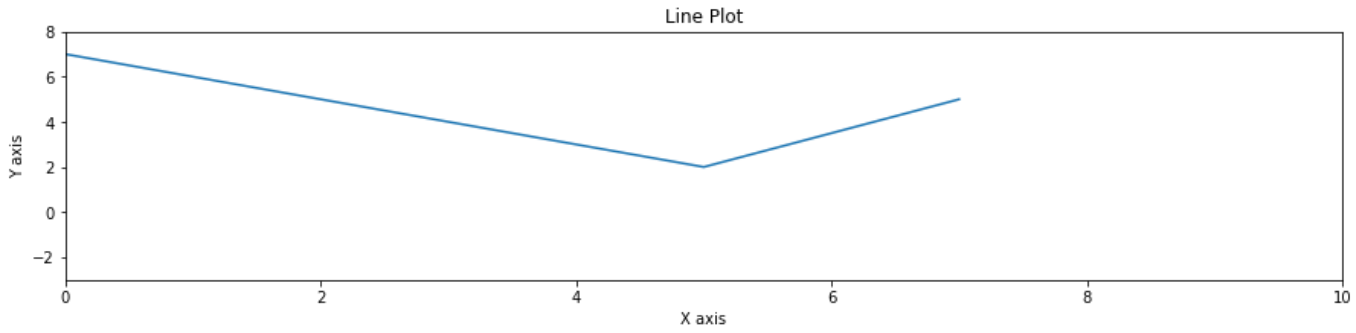
Python Data Visualization:Matplotlib

```
# Matplotlib is a 2D plotting library in Python. It has two interfaces for users
to develop plots with : Stateful and stateless
```

Stateful Visualization

```
#Visualizaing data with matplotlib - stateful approach with python
import matplotlib.pyplot as plt

x=[-3,5,7]
y=[10,2,5]

plt.figure(figsize=(15,3))
plt.plot(x,y)
plt.xlim(0,10)
plt.ylim(-3,8)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('Line Plot')

plt.show()
```

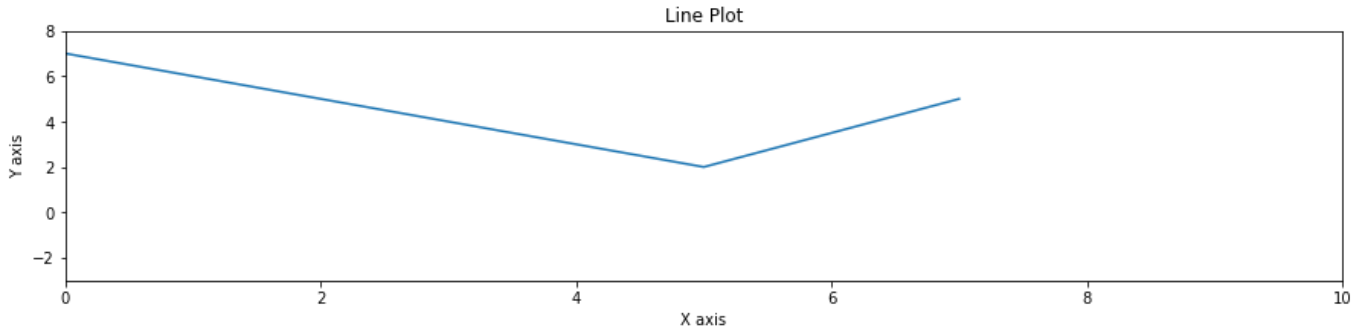



```
# Description :
# In the above code we are plotting a figure using the stateful approach, we have
deifined x axis and y axis .
# A line plot is created with limits for both axis defined
```

Stateless Visualization

Visualization with class Axes: Stateless (OO Approach)

```
import matplotlib.pyplot as plt

x=[-3,5,7]
y=[10,2,5]

fig, ax = plt.subplots(figsize=(15,3))
ax.plot(x,y)
ax.set_xlim(0,10)
ax.set_ylim(-3,8)
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_title('Line Plot')

plt.show()
```




```
# Description
# In the above code we are visualizing a figure using stateless approach with
class axes
# ax refers to axes
# we have set limits for x and y both
# ax.plot will plot the data given to it
#xlim, ylim are used to adjust the limits of both axes
```

Import and load Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('C:/Users/sohai/Iris.csv')
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

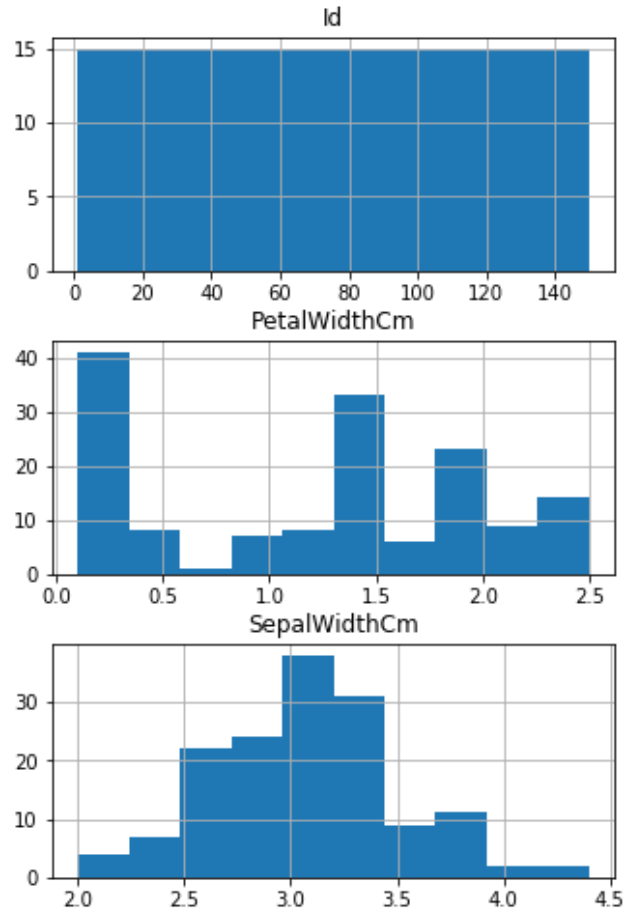
```
}

.dataframe thead th {
    text-align: right;
}
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
# Description: above mentioned code
# Fist all desired libraries are imported
# the dataset was loaded into the pandas dataframe
#  we then read the csv file
# df.head() This function returns the first n rows for the object based on
position. It is useful for quickly testing if your object has the right type of
data in it.
```

Data visualization with Pandas and Matplotlib

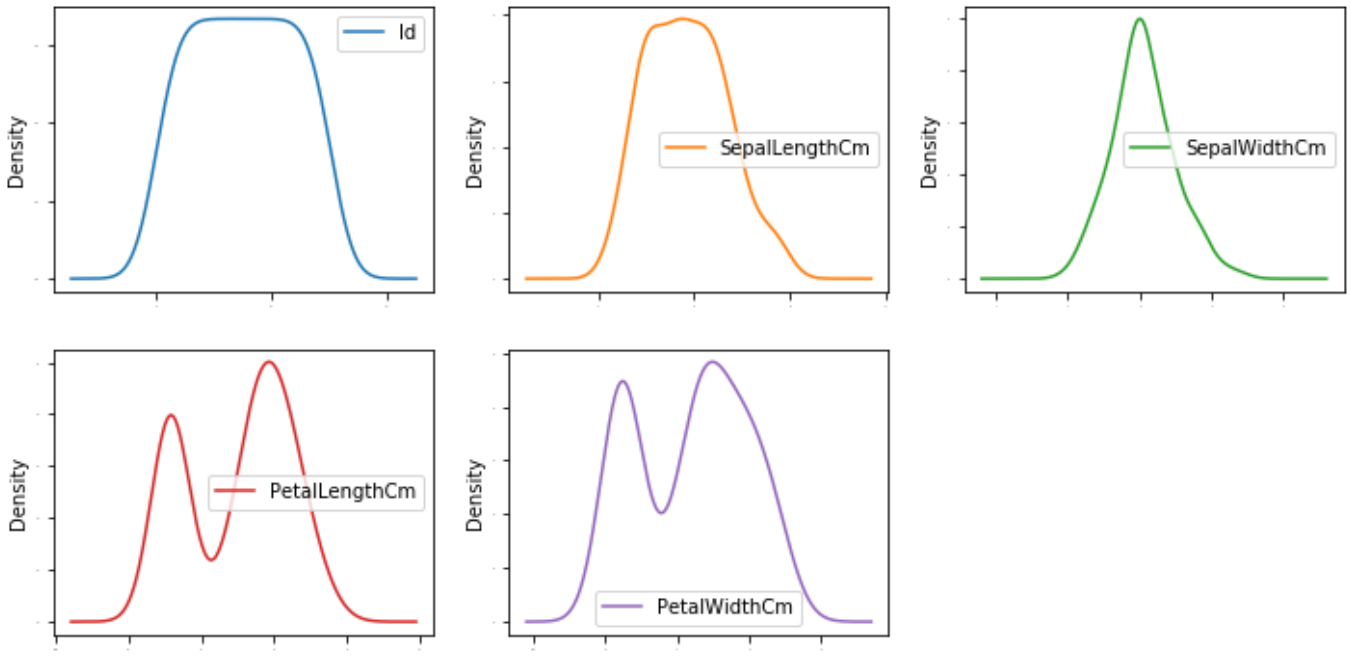
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('C:/Users/sohai/Iris.csv')
df.hist(figsize=(12,8))
plt.show()
```




```
# Description
# In the above mentioned code we are creating histograms of the data provided to
us
# Histogram-The purpose of a histogram (Chambers) is to graphically summarize the
distribution of a univariate data set.
# we have provided what file we want to use for dataset by providing teh location
and read command
# a representation of distribution of data is created in the form of a histogram
# histogram for each numerical value is created
```

Density Plots

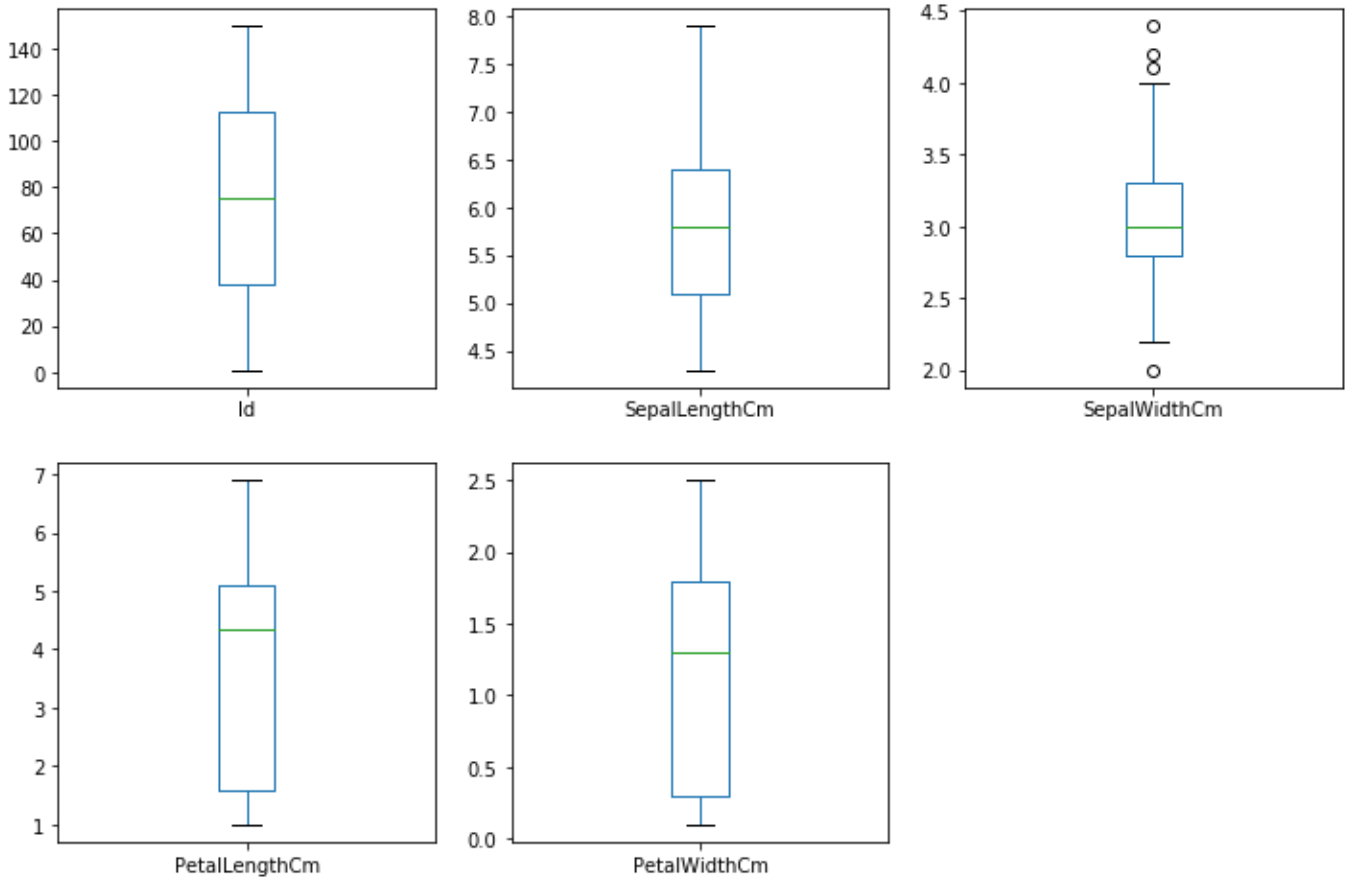
```
df.plot(kind='density', subplots=True,
        layout=(2,3),sharex=False, legend=True, fontsize=1,
        figsize=(12,6))


plt.show()
```




```
# Description
# Density plot for the data set is created
# It takes numerical values as input and plot it. we have created plots for the
different species of plant by showing lengths and widths
```

Boxplots (Box & Whisker Plots)

```
df.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False,
figsize=(12,8))
plt.show()
```

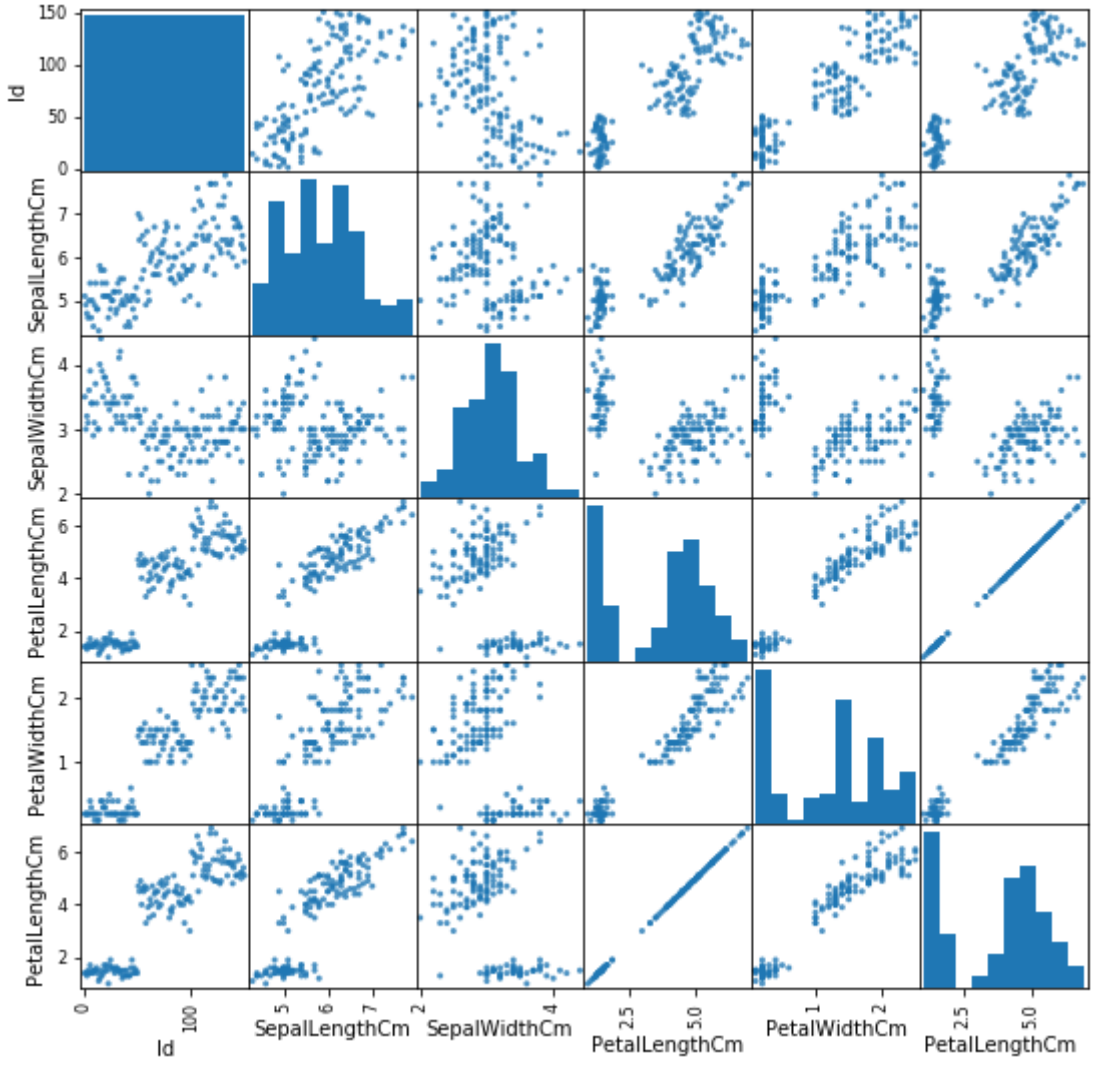



```
# Description
# Now for the same dataset we have created the boxplot
# Box Plots are a measure of how well distributed the data in a data set is. It
divides the data set into three quartiles. This graph represents the minimum,
maximum, median, first quartile and third quartile in the data set.
```

MultiVariate Data Visualization

Scatter Matrix Plot

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




```
# Description:
# A scatter plot is created which includes the plots of all pairs of attributes
```

Machine Learning and Supervised Linear Regression

Load Data - Import Python Libraries and Module

```
# Import Python Libraries: Numpy and Pandas
import pandas as pd
import numpy as py

# Import Libraries & modules for data visualization
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
# Import scikit-Learn module for the algorithm/modeL: Linear Regression
from sklearn.linear_model import LinearRegression
# Import scikit-Learn module to split the dataset into train/ test sub-datasets
from sklearn.model_selection import train_test_split
# Import scikit-Learn module for K-fold cross-validation - algorithm/modeL
```

```
evaluation & validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# Specify location of the dataset
filename = 'C:/Users/sohai/housing_boston.csv'
# Specify the fields with their names
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT', 'MEDV']
# Load the data into a Pandas DataFrame
df = pd.read_csv(filename, names=names)
# VIP NOTES:
# Extract a sub-dataset from the original one -- > dataframe: df2
df2 = df[['RM', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'MEDV']]
```

```
# Description
# First we import Python libraries
# Then we load the dataset by proviidng which file to use by providing a path of
that file
# we then specify the fields to use
# load the data into the pandas dataframe
# we then define a second dataframe and give it values
```

Pre-process Dataset

Clean Data: Find & Mark Missing Values

```
# mark zero values as missing or NaN
df[['RM', 'PTRATIO', 'MEDV']] = df[['RM', 'PTRATIO', 'MEDV']].replace(0, np.NaN)
# count the number of NaN values in each
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
```

```
# Description
# Here we are clening the data as it can not have 0 values
# in each column if there is an invalid value we mark it as missig value of NaN -
not a number
```

Perform the Exploratory Data Analysis (EDA) on the dataset

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

#DESCRIPTION
# we will be getting the shape of data set which will tell us the exact dimensions
e.g. rows by columns and printing it
```

```
(452, 6)
```

```
# Get the data types of all variabLes/attributes of the data set
# The results show
print(df2.dtypes)

#DESCRIPTION
# here we are printing the type of the variables in the dataset e.g. if it is an
integer or float (has decimals)
```

```
RM         float64
AGE        float64
DIS        float64
RAD          int64
PTRATIO    float64
```

```
MEDV       float64
dtype: object
```

```
# Get several records/rows at the top of the dataset
# Get the first five records
print(df2.head(5))

#DESCRIPTION
#here we have specified that we need to display data from the first 5 rows
therefore starting from 0 to 4
```

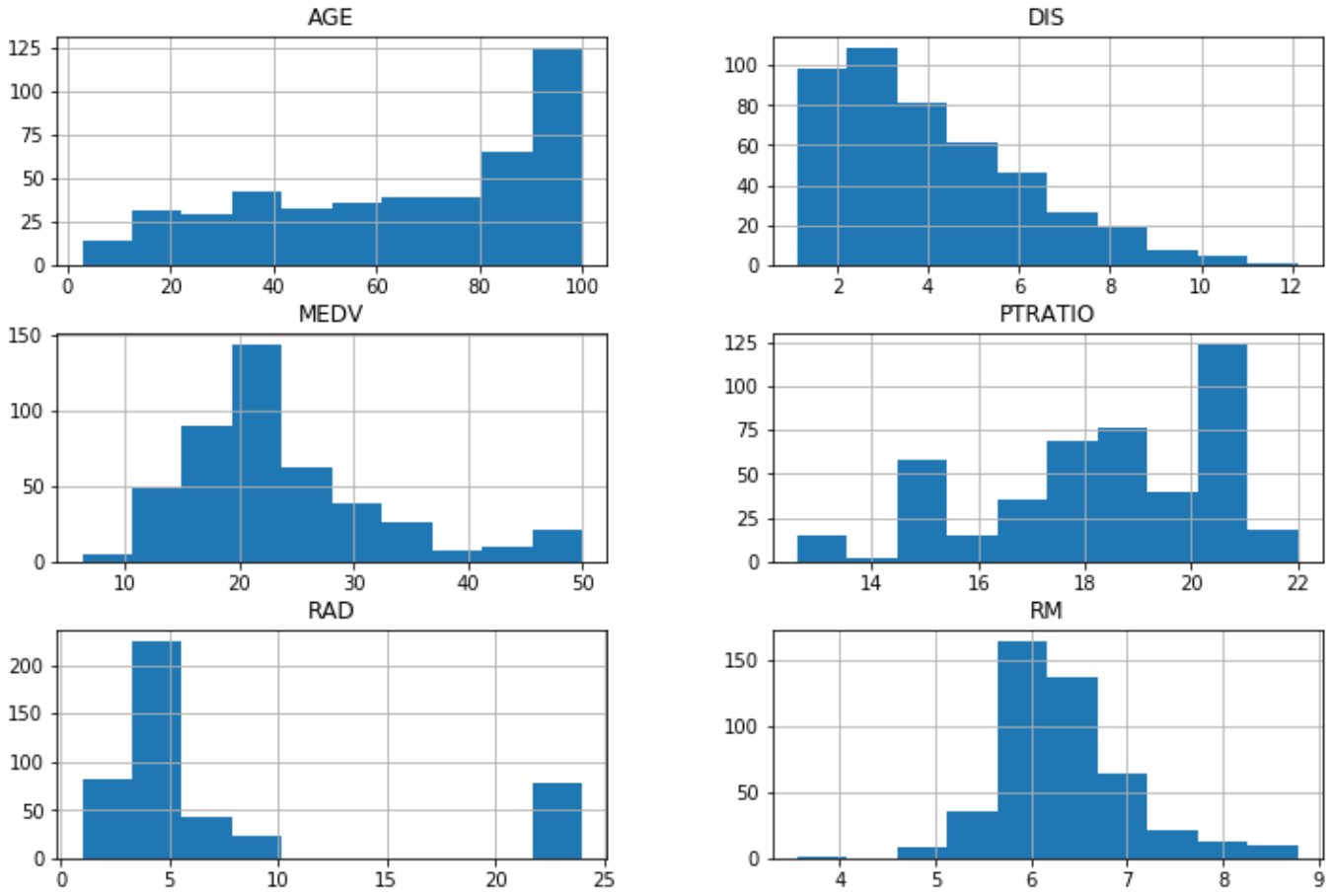
```
      RM   AGE     DIS  RAD  PTRATIO  MEDV
0  6.575  65.2  4.0900    1     15.3  24.0
1  6.421  78.9  4.9671    2     17.8  21.6
2  7.185  61.1  4.9671    2     17.8  34.7
3  6.998  45.8  6.0622    3     18.7  33.4
4  7.147  54.2  6.0622    3     18.7  36.2
```

```
# Get the summary statistics of the numeric variables/attributes of the dataset
print(df2.describe())

# DESCRIPTION
# here we are printing the summary statistics of the data provided to us, which
includes the mean, median, std dwviation, min ans maximum values
```

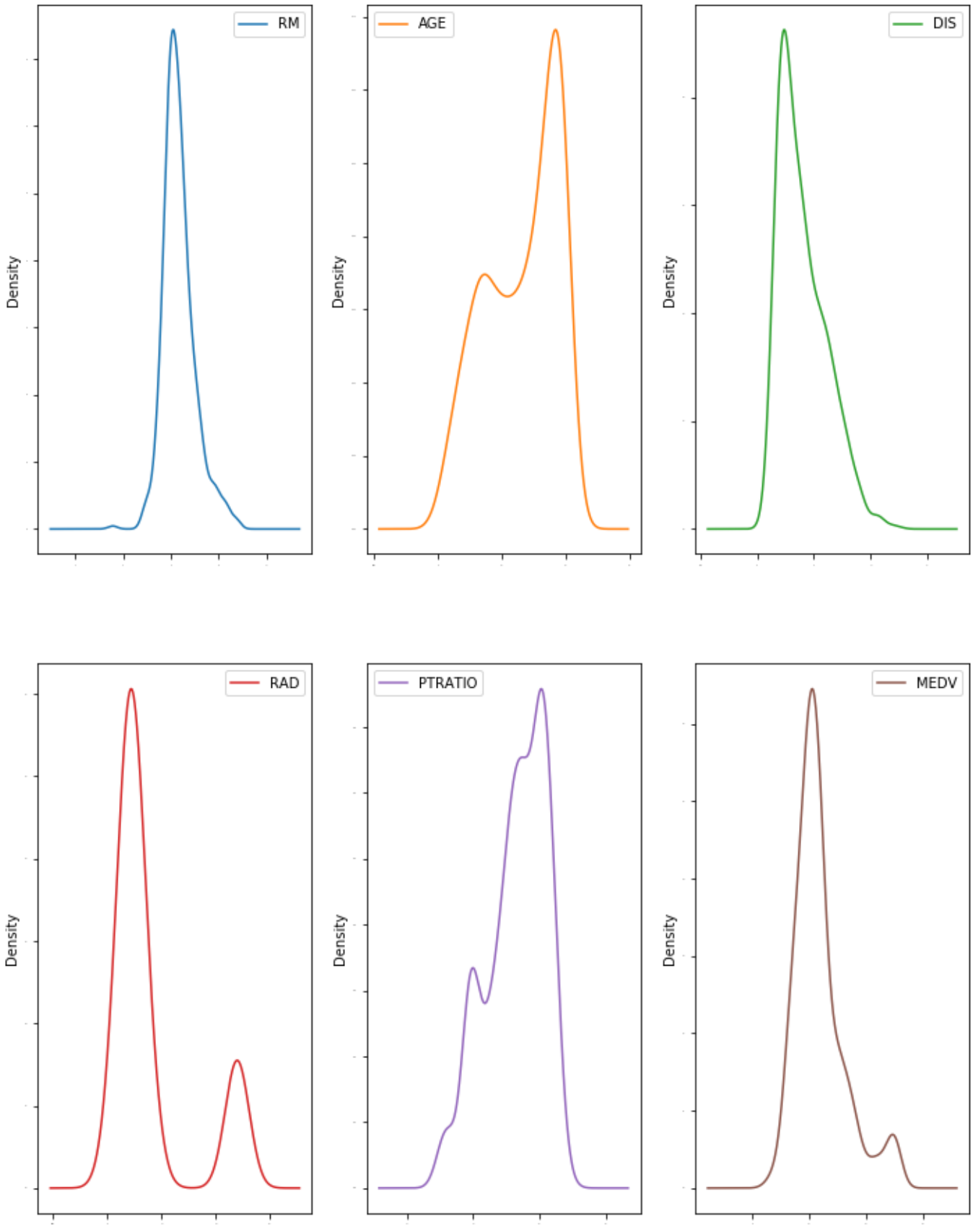
```
                RM         AGE         DIS         RAD     PTRATIO        MEDV
count  452.000000  452.000000  452.000000  452.000000  452.000000  452.000000
mean     6.343538   65.557965    4.043570    7.823009   18.247124   23.750442
std      0.666808   28.127025    2.090492    7.543494    2.200064    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
```

```
# Plot histrogram for each numeric
df2.hist(figsize=(12, 8))
pyplot.show()
```

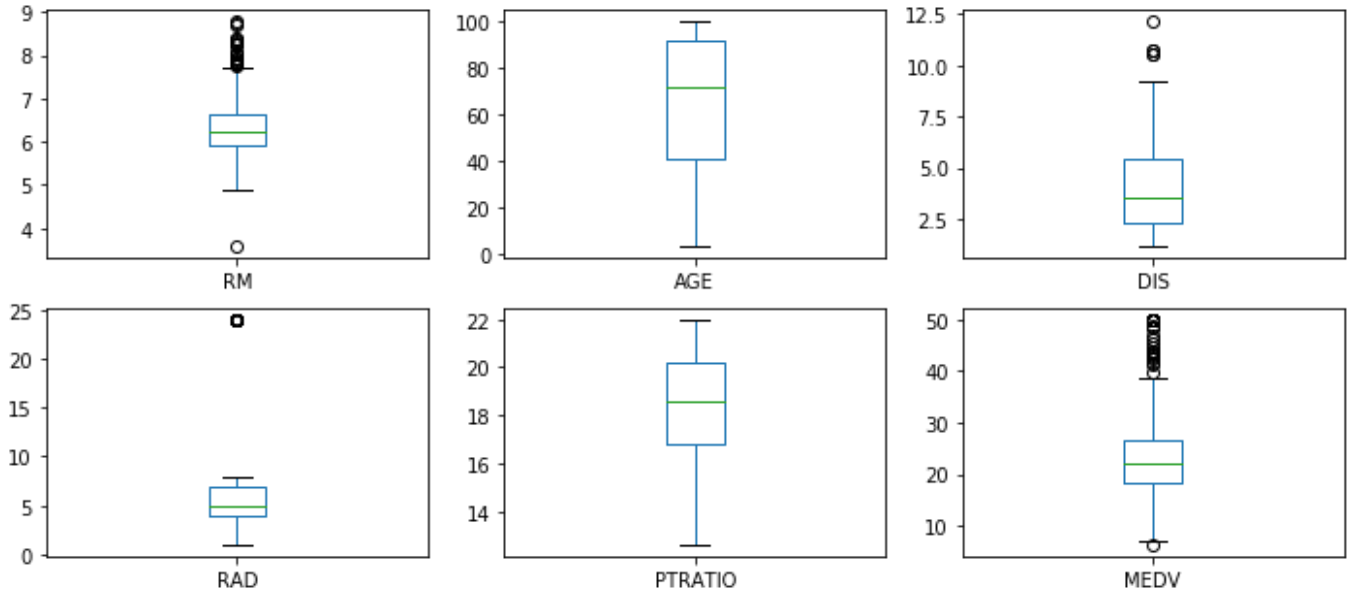
```
# DESCRIPTION
# here we are plotting the histogram for each numeric e.g.RM, RAD, MEDV etc
```



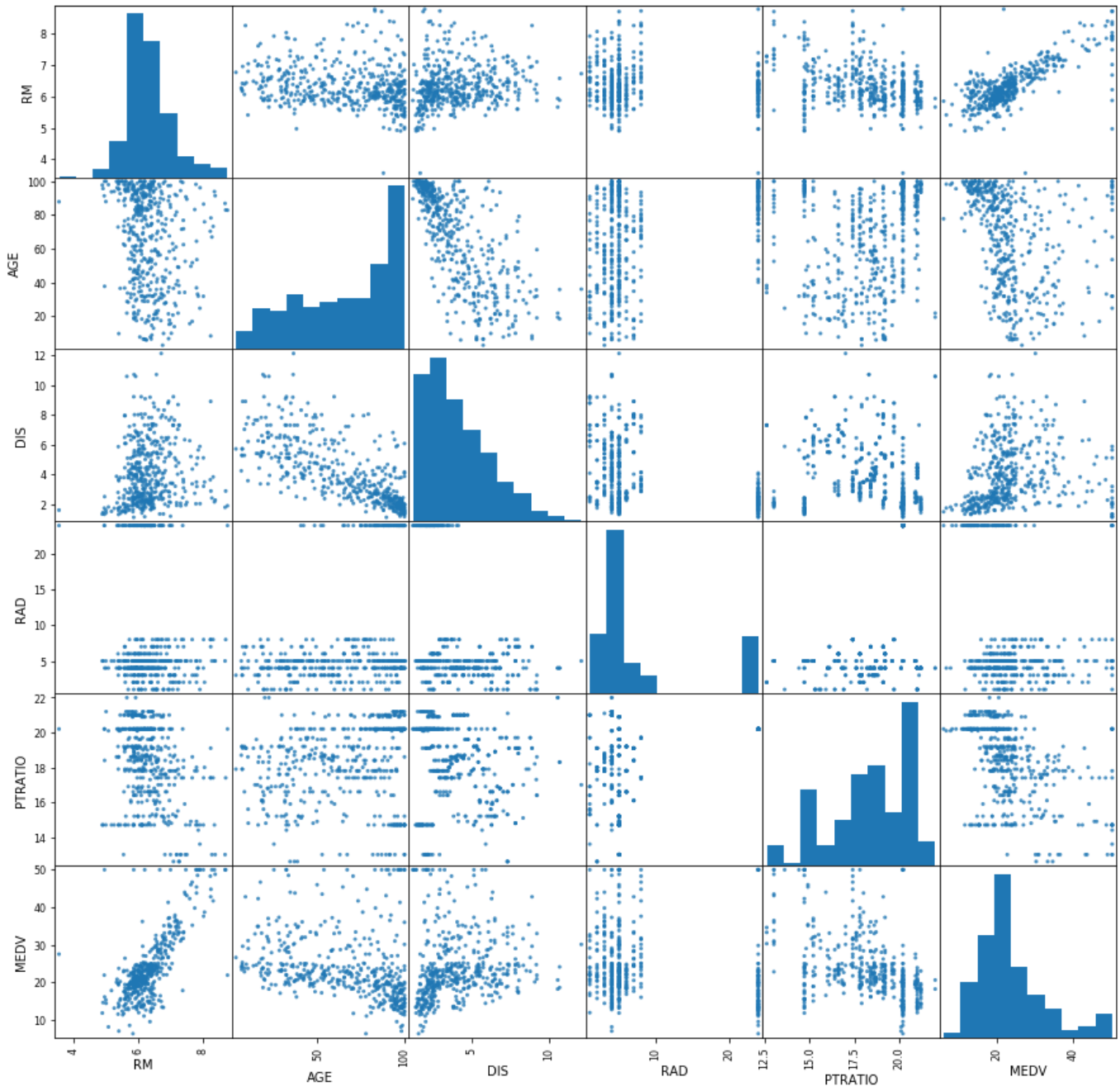

```
# Density plots
# IMPORTANT NOTES: 5 numeric variables -> at Least 5 plots -> Layout (2, 3): 2
rows, each row with 3 plots
df2.plot(kind='density', subplots=True, layout=(2, 3), sharex=False, legend=True,
fontsize=1,
figsize=(12, 16))
pyplot.show()

#Description
# this is the density plot for the same dataset  2,3 means 2 rows and each row
with  plots
```




```
df2.plot(kind='box', subplots=True, layout=(3,3), sharex=False, figsize=(12,8))
pyplot.show()


# Description
# bos and whiskers plot is now created which shows the quartile range and outliers
```




```
# scatter plot matrix
scatter_matrix(df2, alpha=0.8, figsize=(15, 15))
pyplot.show()

#A scatter plot is created
```



Separate Dataset into Input & Output NumPy Arrays

```
# 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]


# Description
# Here we are storing the dataset in the multidimensional array
# y here is a dependent variable
```

Split Input/Output Arrays into Training/Testing Datasets

```
# 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 randomly
# 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)

# here we are splitting dataset, 1/3rd of the data will be used for test ad the
rest will be used for training
```

Build and Train the Model

```
# Build the model
model = LinearRegression()
# Train the model using the training sub-dataset
model.fit(X_train, Y_train)
# Print out the coefficients and the intercept
# print intercept and coefficients
print (model.intercept_)
print (model.coef_)


#Description
# here we are using the function linear regression to print the coefficients and
intercept
```

```
-4.53602172360959
[ 8.45701265 -0.08160589 -0.80038663 -0.1426911  -0.86413873]
```

```
# If we want to print out the list of the coefficients with their correspondent
variable name
# pair the feature names with the coefficients
names_2 = ['RM', 'AGE','DIS','RAD', 'PTRATIO']
coeffs_zip = zip(names_2, model.coef_)
# Convert iterator into set
coeffs = set(coeffs_zip)
#print (coeffs)
for coef in coeffs:
    print(coef, "\n")

#Description: Here we have listed a variable names_2 and we are printing the
```

```
coefficients
# zip() The function takes in iterables as arguments and returns an iterator.
# set() set() method is used to convert any of the iterable to the distinct
element and sorted sequence of iterable elements, commonly called Set
```

```
('DIS', -0.800386625412262)

('RAD', -0.14269109757375334)

('RM', 8.457012650599342)

('PTRATIO', -0.8641387261012642)

('AGE', -0.08160589380401152)
```

Calculate R-Squared

```
R_squared = model.score(X_test, Y_test)
print(R_squared)

#Desciption
# Here we are calculating the value of R square value ; R-squared is a statistical
measure of how close the data are to the fitted regression line
# the higher the Rsquare, the better it is
# best models score above 83% - this tells us how good the independent cariables
predict the dependent
```

```
0.92
```

Prediction

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

# 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
# the model predict that the median value of owner-occupied homes in 1000 dollars
```

```
in the above suburb should be around 23,000.
```

```
array([23.60419508])
```

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

```
# 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 the 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())



# Description:
# afte rwe train, we do the evaluation
# Use K-Fold to determine if the model is acceptable
# We pass the whole set because the system will divide for us -31 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 5 and 6
```

```
-31.17777676938244


C:\Users\sohai\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:296:
FutureWarning: Setting a random_state has no effect since shuffle is False. This
will raise an error in 0.24. You should leave random_state to its default (None),
or set shuffle=True.
  FutureWarning
```

Machine Learning: Supervised – Logistic Regression

Import Python Libraries and Modules

```
# 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
```

```
# Import scikit-Learn module for the algorithm/modeL: Logistic Regression
from sklearn.linear_model import LogisticRegression
```

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

```
# Import scikit-Learn module for K-fold cross-validation - algorithm/modeL
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 information
about how the system try to classify / lable each record
from sklearn.metrics import classification_report
```

Load the data - Dataset: Iris.scv

```
# Specify location of the dataset
filename = 'C:/Users/sohai/iris.csv'
# Load the data into a Pandas DataFrame
df = pd.read_csv(filename)

# Description: we are uploading the iris.csv file by providing file path and the
reading the file. A datafram using pandas is created
```

```
## Preprocess Dataset - Clean Data: Find & Mark Missing Values
```

```
# mark zero values as missing or NaN
df[[ 'SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm ' ,'PetalWidthCm' ]] \
= df[['SepalLengthCm' , 'SepalWidthCm' ,'PetalLengthCm' , 'PetalWidthCm'
]].replace(0,np.NaN)
# count the number of NaN values in each column
print (df.isnull().sum())

# Description: data cleaning - we look at all values and any invalid value is
marked as not a number or zero
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
PetalLengthCm    0
dtype: int64
```

Perform the exploratory data analysis (EDA) on the dataset

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

# here the shape is being defined
```

```
(150, 7)
```

```
#get the data types of all the variables / attributes in the data set
print(df.dtypes)

#Description - the data type of variables is printed
```

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
PetalLengthCm    float64
dtype: object
```

```
#return the first five records / rows of the data set
print(df.head(5))

#DEscription : first 5 rows are displayed
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

   PetalLengthCm
0            1.4
1            1.4
2            1.3
3            1.5
4            1.4
```

```
#return the summary statistics of the numeric variables / attributes in the data
set
print(df.describe())


#Description: calculating summary statistics
```

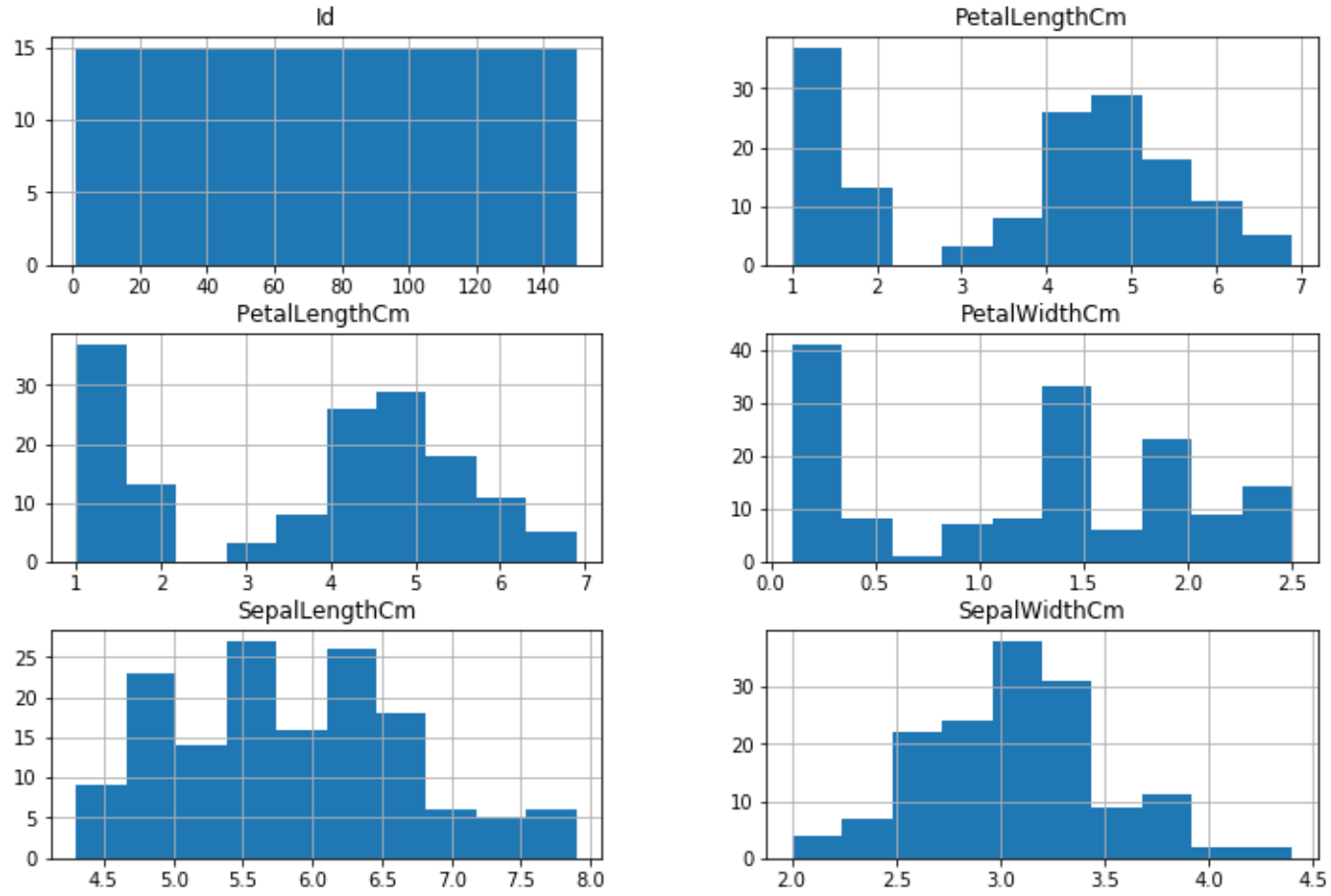
```
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
count  150.000000     150.000000    150.000000     150.000000    150.000000
mean    75.500000       5.843333      3.054000       3.758667      1.198667
std     43.445368       0.828066      0.433594       1.764420      0.763161
min      1.000000       4.300000      2.000000       1.000000      0.100000
```

```
25%     38.250000       5.100000      2.800000       1.600000      0.300000
50%     75.500000       5.800000      3.000000       4.350000      1.300000
75%    112.750000       6.400000      3.300000       5.100000      1.800000
max    150.000000       7.900000      4.400000       6.900000      2.500000

       PetalLengthCm
count      150.000000
mean         3.758667
std          1.764420
min          1.000000
25%          1.600000
50%          4.350000
75%          5.100000
max          6.900000
```

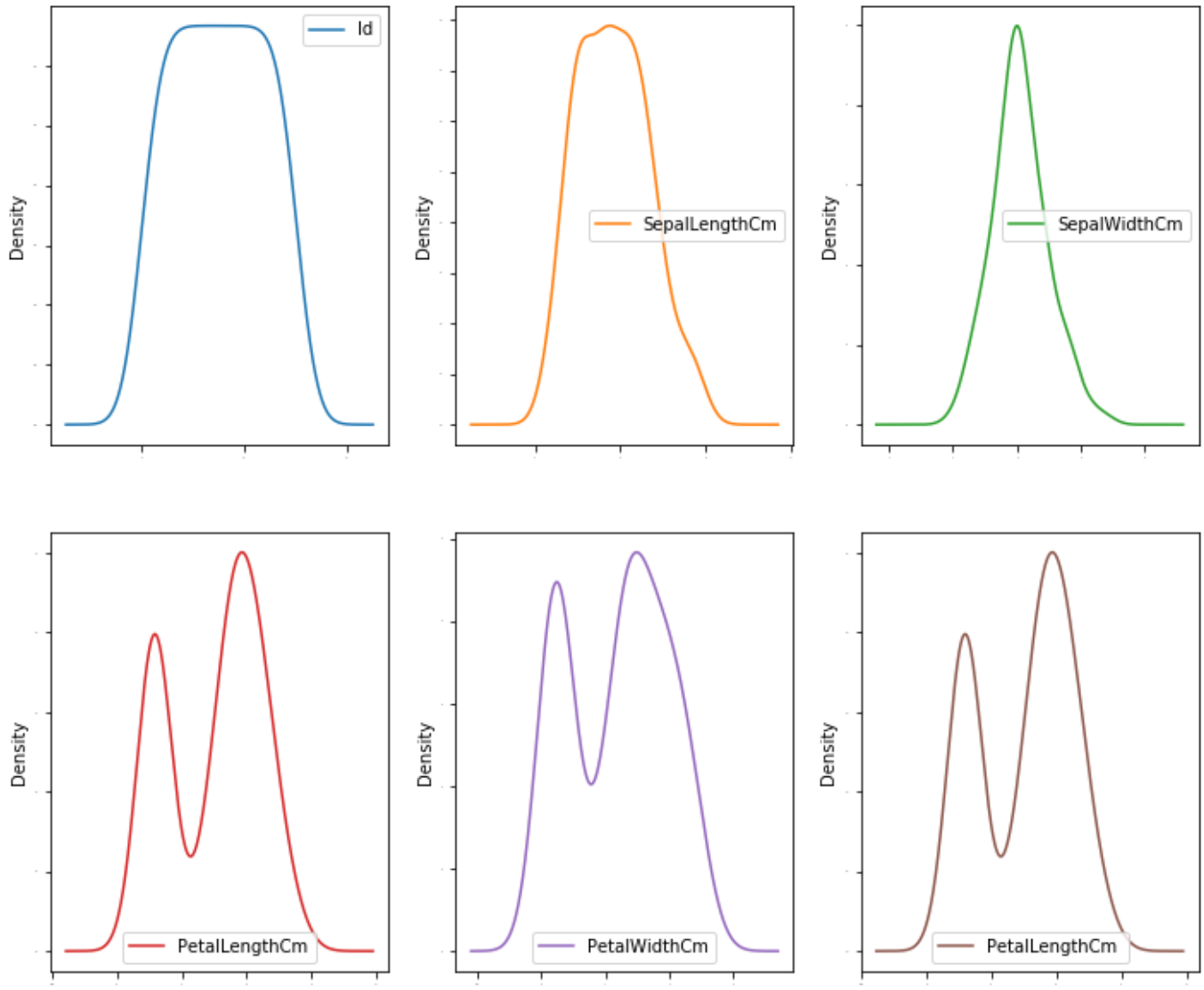
```
#class distribution i.e. how many records are in each class

print(df.groupby('Species').size())

#Description - here we are grouping by type species and showing the number of
record in each group, also mentioning that data type is integer
```

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

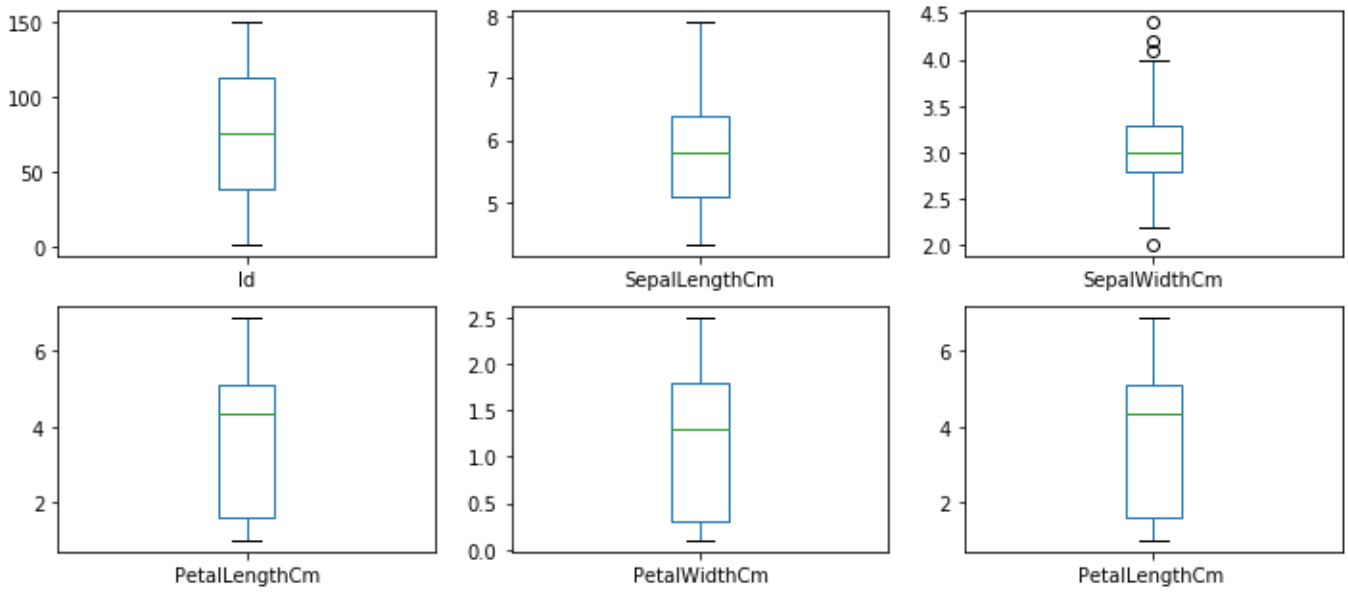
```
#plot histogram of each numeric variable / attribute in the data set
df.hist(figsize=(12, 8))
pyplot.show()

# Description - plotting histogram of the variables in the data set e.g. petal
length, petal width etc.
```



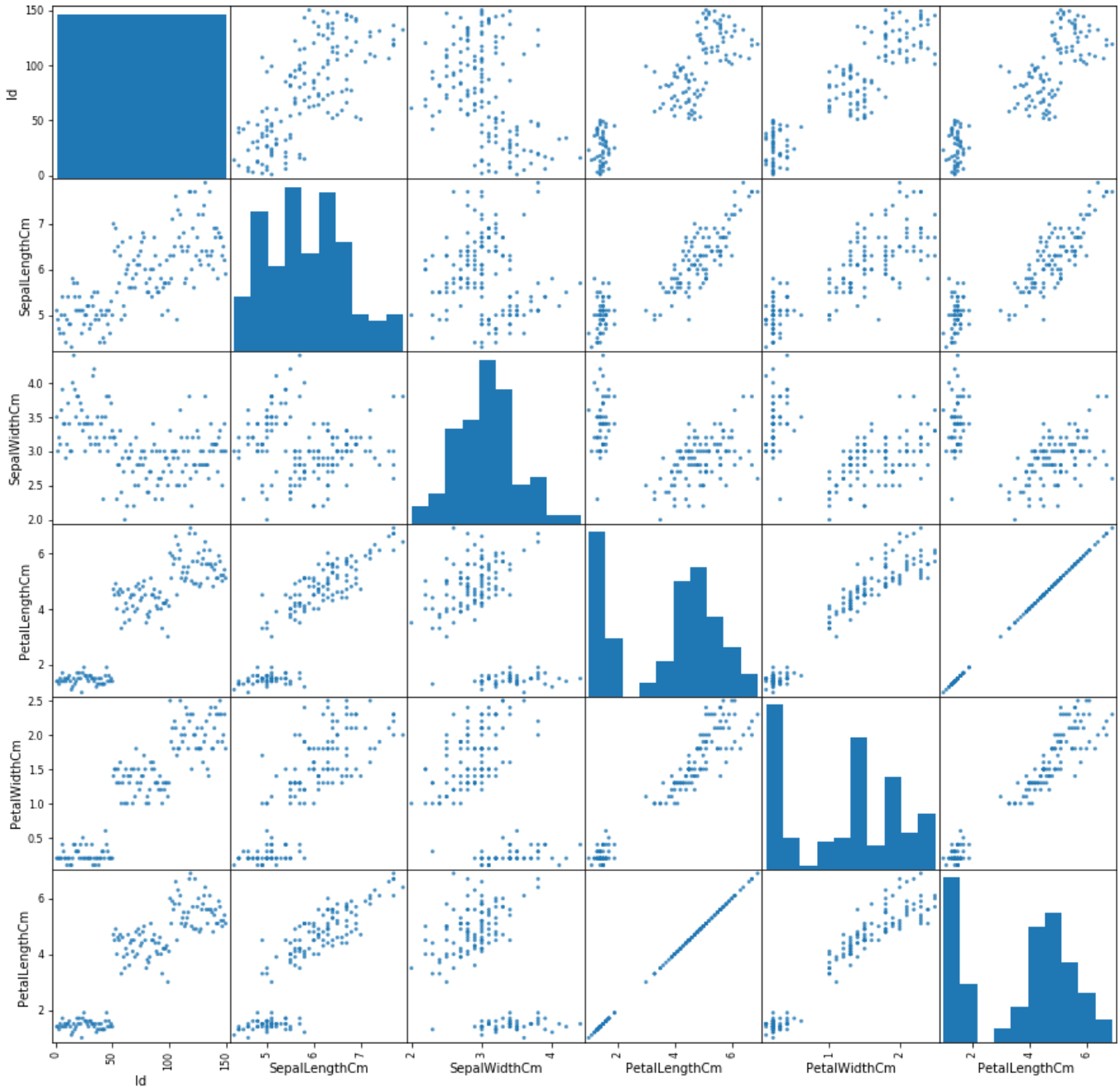

```
# generate density plots of each numeric variable / attribute in the data set
df.plot(kind='density', subplots=True, layout=(3, 3), sharex=False, legend=True,
fontsize=1,
figsize=(12, 16))
pyplot.show()

# Description - plotting the density plot for the same variables as above
```




```
# generate box plots of each numeric variable / attribute in the data set
df.plot(kind='box', subplots=True, layout=(3,3), sharex=False, figsize=(12,8))
pyplot.show()

# Description - plotting the box plot for the variables - it shows the outliers
and quartile range
```




```
# generate scatter plot matrix of each numeric variable / attribute in the data
set
scatter_matrix(df, alpha=0.8, figsize=(15, 15))
pyplot.show()



# Description - Scatter plot is also constructed for the data set. here we have
specified the size as well
```



Separate Dataset into Input & Output NumPy arrays

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

# Description - Here we are seperating datset as trainign data (X) consists of
independent variables
# Also beacuse the desired output which is (Y) consists of dependent variable or
the ones we are trying to predict
```

Split Input/Output Arrays into Training/Testing Datasets

```
# 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 randomly
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)
```

Build and Train the Model

```
#build the model
model = LogisticRegression()
# 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)

# Description - 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
# 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
# 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
# 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
# The support is the number of occurrences of each class in y_true
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.89      0.89      0.89        18
 Iris-virginica       0.89      0.89      0.89        18
```

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

Score the accuracy of the model

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

# Description: here we are calculating the accuracy
```

```
Accuracy: 92.000%
```

Classify/Predict Model

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

#Description - we are predicting the flower type here
```

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

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

```
# evaluate the algorythm
# 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 logistic regression, we can use the accuracy level to evaluate the model /
algorithm
scoring = 'accuracy'

# Description:
#specify the number of time of repeated splitting, Fix the random seed, Must use
the same see 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,
we can use accuracy level to evaluate the model/algorithm, train the model and run
k-fold cross-validation to validate/evaluate the model, Print out the evaluation
results
```

```
# 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()))

##using the 10-fold cross-validation to evaluate the model / algorithm, the
accuracy of this logistic regression model is 94.7%
# There is 94.7% chance that this new record is an Iris-versicolor
```

```
C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)


Accuracy: 0.947 (0.058)


C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\sohai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```