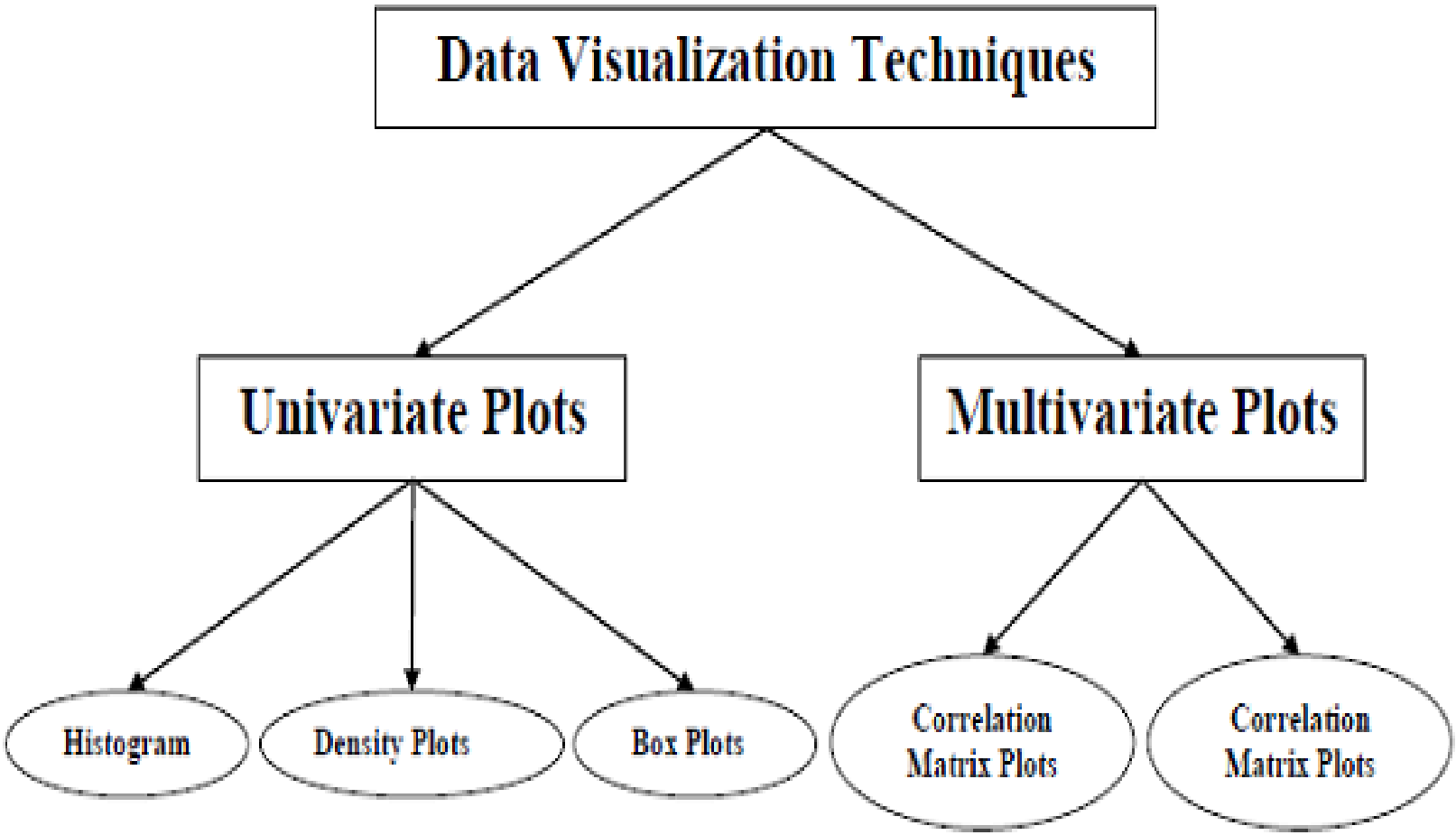
Machine Learning

ML - Understanding Data with Visualization

Computer & Health Informatics Department

Introduction

- Visualization is a way to understand the data. With the help of data visualization, we can see how the data looks like and what kind of correlation is held by the attributes of data.
- Visualization is the fastest way to see if the features correspond to the output.
- With the help of following Python recipes, we can understand ML data with Visualization .


Data Visualization Techniques
Univariate Plots
Multivariate Plots
Histogram
Density Plots
Box Plots
Correlation Matrix Plots
Correlation Matrix Plots

Univariate Plots: Understanding Attributes Independently

- The simplest type of visualization is single-variable or "univariate" visualization.
- With the help of univariate visualization, we can understand each attribute of our dataset independently.
- The following are some techniques in Python to implement univariate visualization –

Histograms

- Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in dataset.
- The following are some of the characteristics of histograms :
  - It provides us a count of the number of observations in each bin created for visualization.
  - From the shape of the bin, we can easily observe the distribution.
  - Histograms also help us to see possible outliers.

Example

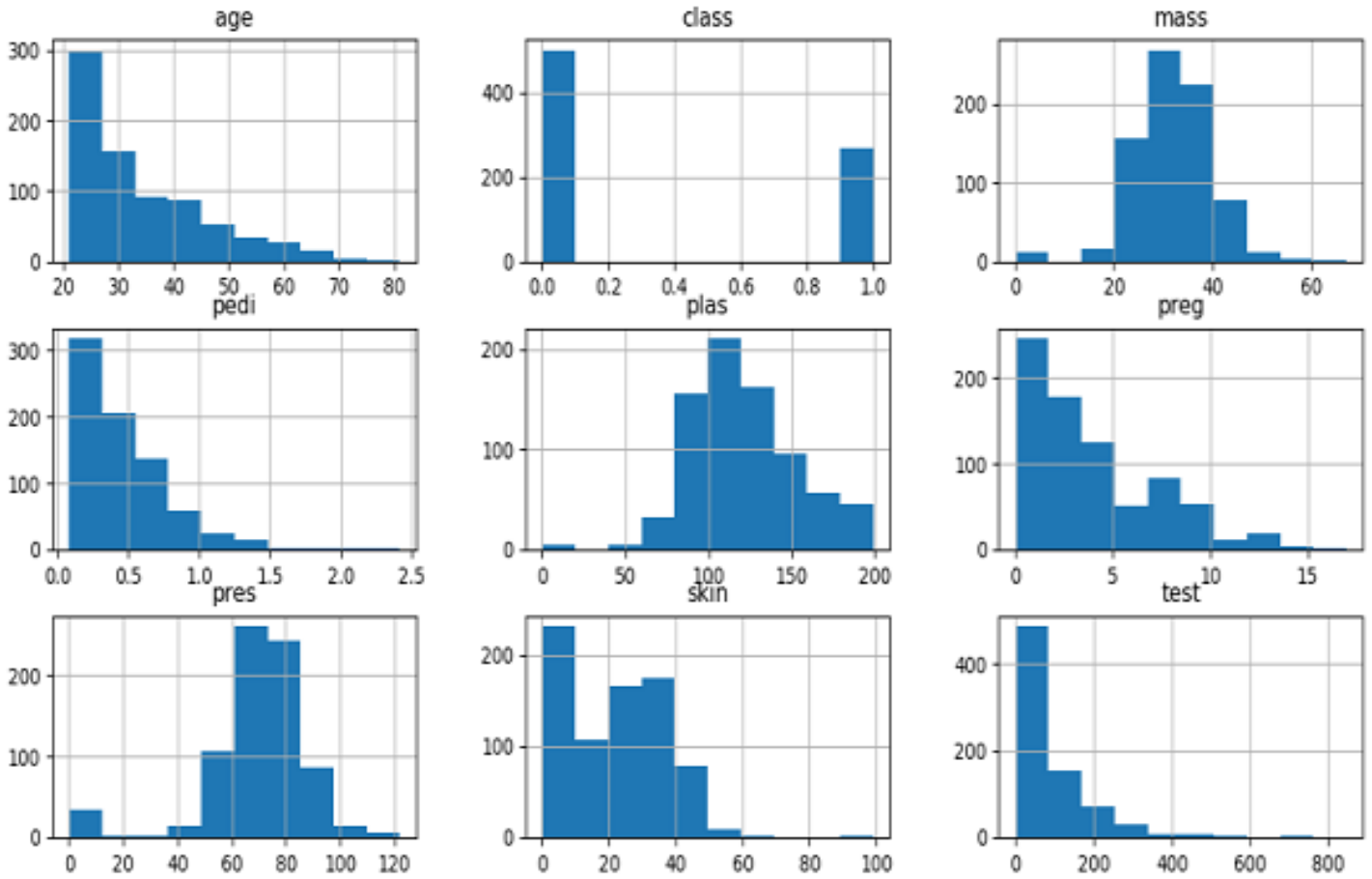
- The code shown below is an example of Python script creating the histogram of the attributes of Pima Indian Diabetes dataset.

- Here, we will be using hist() function on Pandas DataFrame to generate histograms and matplotlib for ploting them.

Example

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names=names)
data.hist()
pyplot.show()
```

Example



Density Plots

- Another quick and easy technique for getting each attributes distribution is Density plots.
- It is also like histogram but having a smooth curve drawn through the top of each bin.
- We can call them as abstracted histograms.

Example

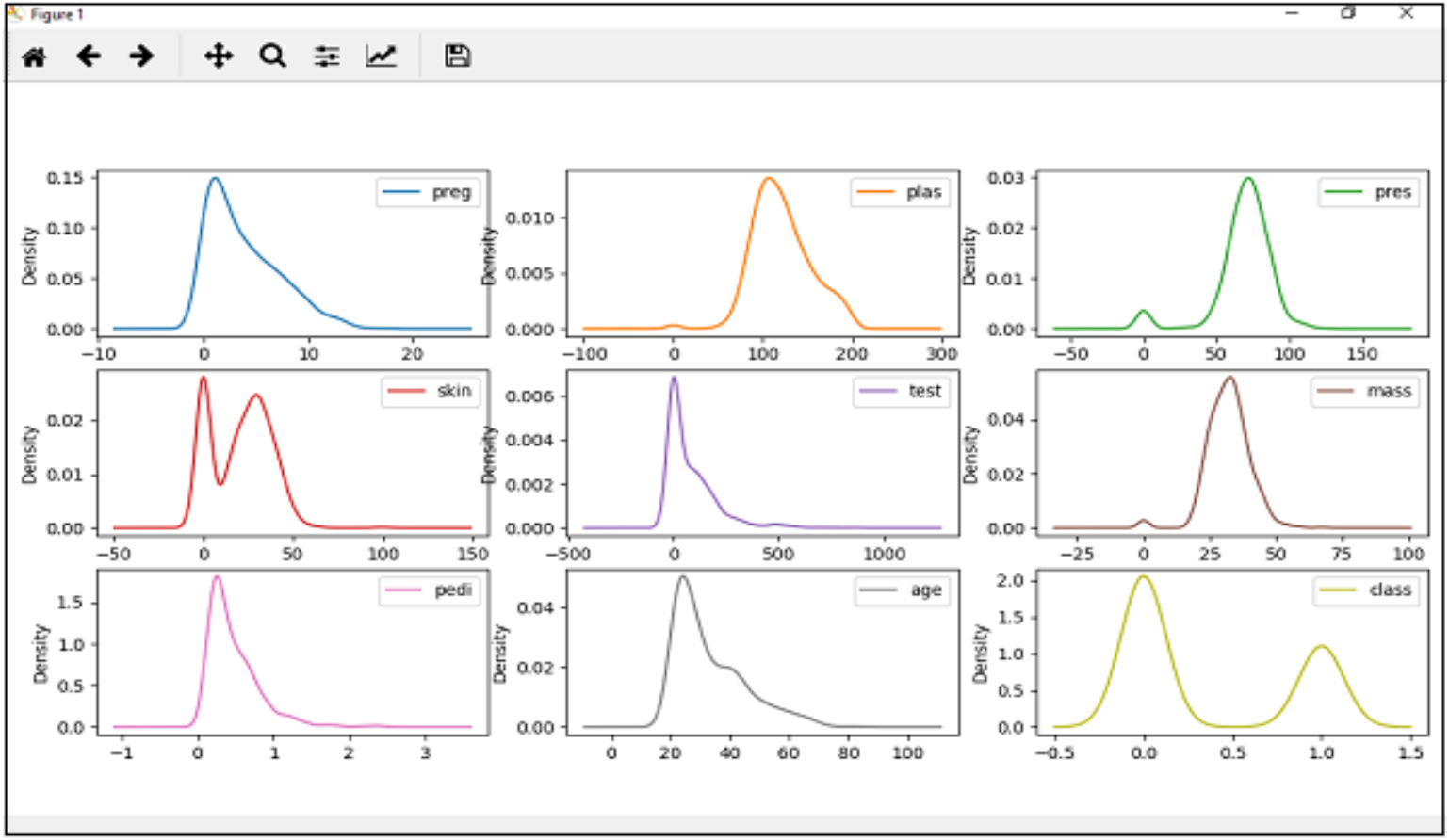
- In the following example, Python script will generate Density Plots for the distribution of attributes of Pima Indian Diabetes dataset.

Example

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names=names)
 data.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
pyplot.show()
```

Example

Output



Box and Whisker Plots

- Box and Whisker plots, also called boxplots in short, is another useful technique to review the distribution of each attribute's distribution.
- The following are the characteristics of this technique:
  - It is univariate in nature and summarizes the distribution of each attribute.

Box and Whisker Plots

- It draws a box around the 25% and 75%.
- It also draws whiskers which will give us an idea about the spread of the data.
- The dots outside the whiskers signifies the outlier values. Outlier values would be 1.5 times greater than the size of the spread of the middle data.

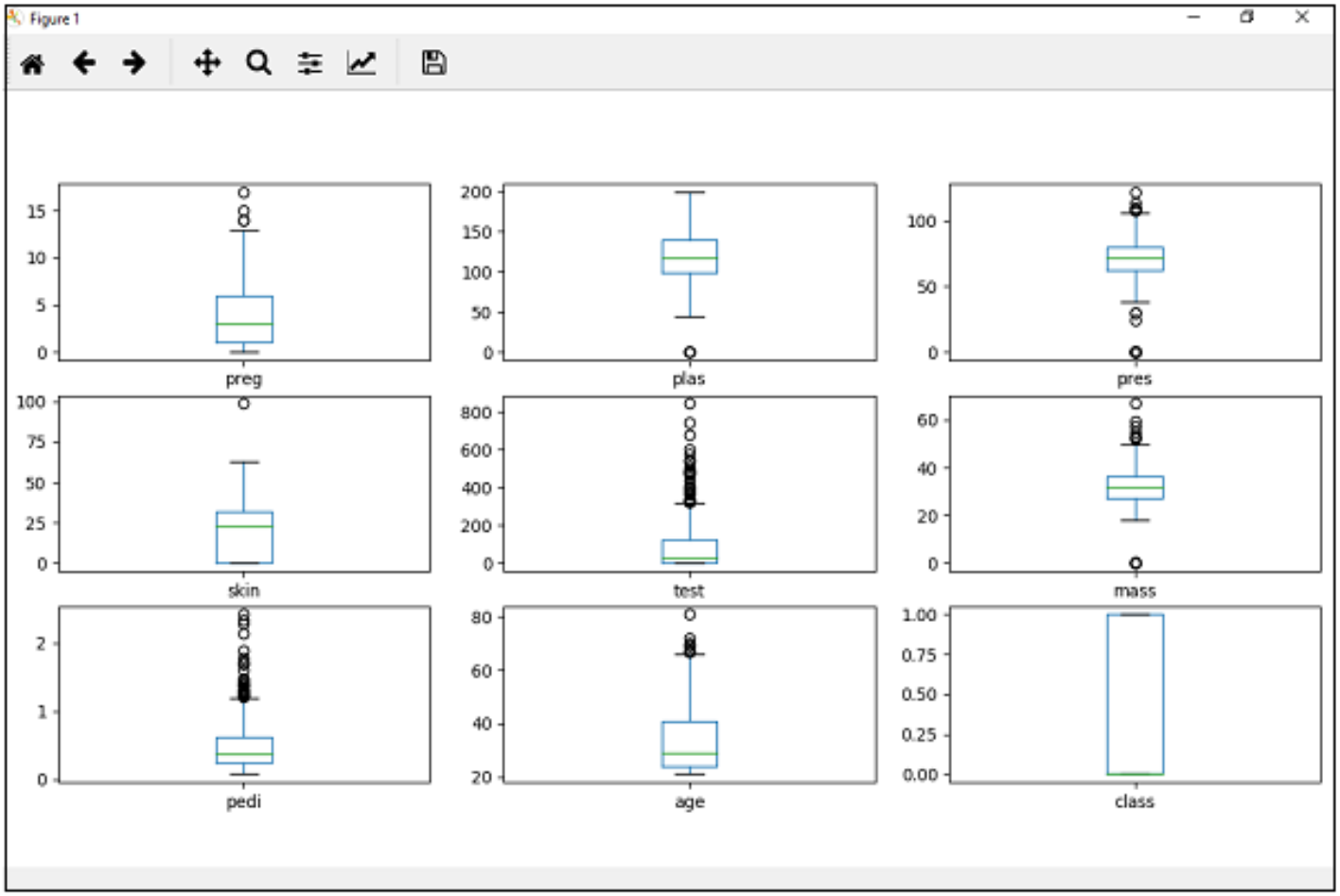
Example

- In the following example, Python script will generate Box Plots for the distribution of attributes of Pima Indian Diabetes dataset.

Example

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(path, names = names)
data.plot(kind = 'box', subplots = True, layout = (3,3), sharex = False,sharey = False)
pyplot.show()
```

Output



Multivariate Plots: Interaction Among Multiple Variables

- Another type of visualization is multi-variable or "multivariate" visualization.
- With the help of multivariate visualization, we can understand interaction between multiple attributes of our dataset.
- The following are some techniques in Python to implement multivariate visualization −

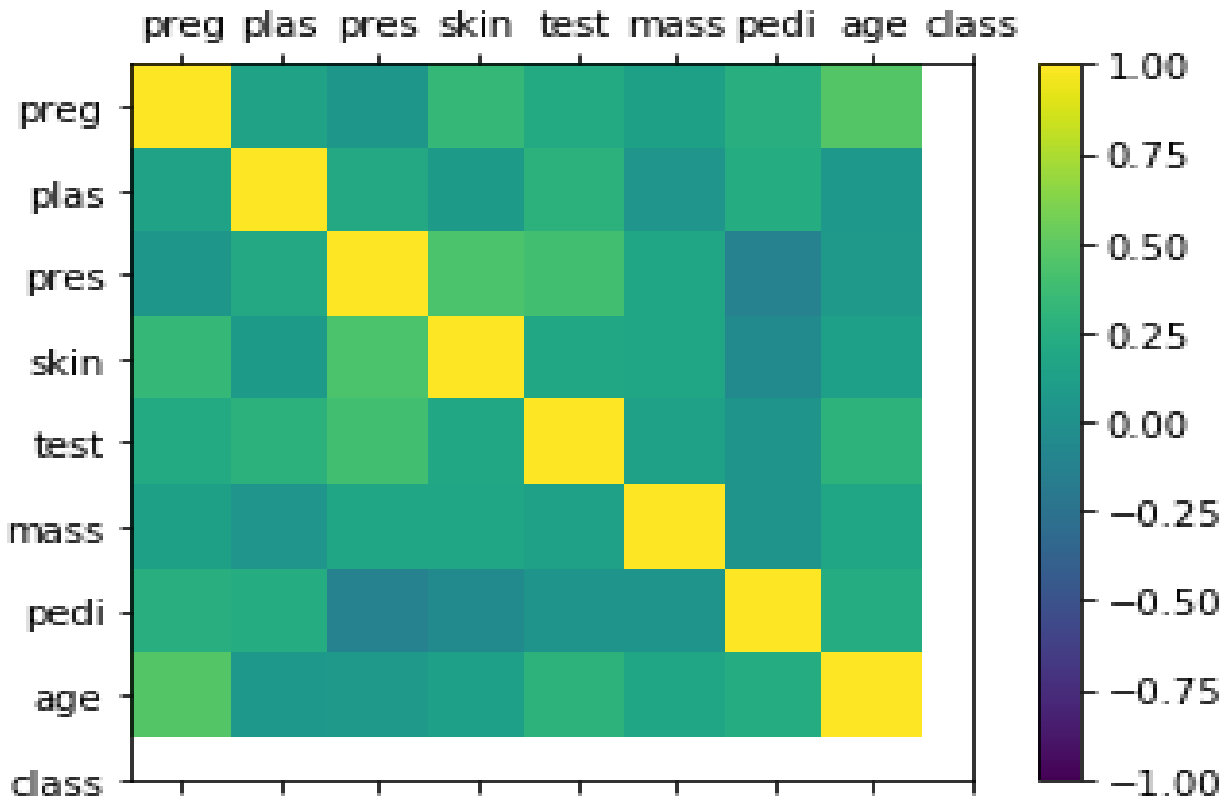
Correlation Matrix Plot

- Correlation is an indication about the changes between two variables.
- In our previous chapters, we have discussed Pearson's Correlation coefficients and the importance of Correlation too.
- We can plot correlation matrix to show which variable is having a high or low correlation in respect to another variable.

Example

- In the following example, Python script will generate and plot correlation matrix for the Pima Indian Diabetes dataset.
- It can be generated with the help of corr() function on Pandas DataFrame and plotted with the help of *pyplot*.

```
from matplotlib import pyplot
from pandas import read_csv
import numpy
Path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(Path, names = names)
correlations = data.corr()
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```



From the above output of correlation matrix, we can see that it is symmetrical i.e. the bottom left is same as the top right. It is also observed that each variable is positively correlated with each other.

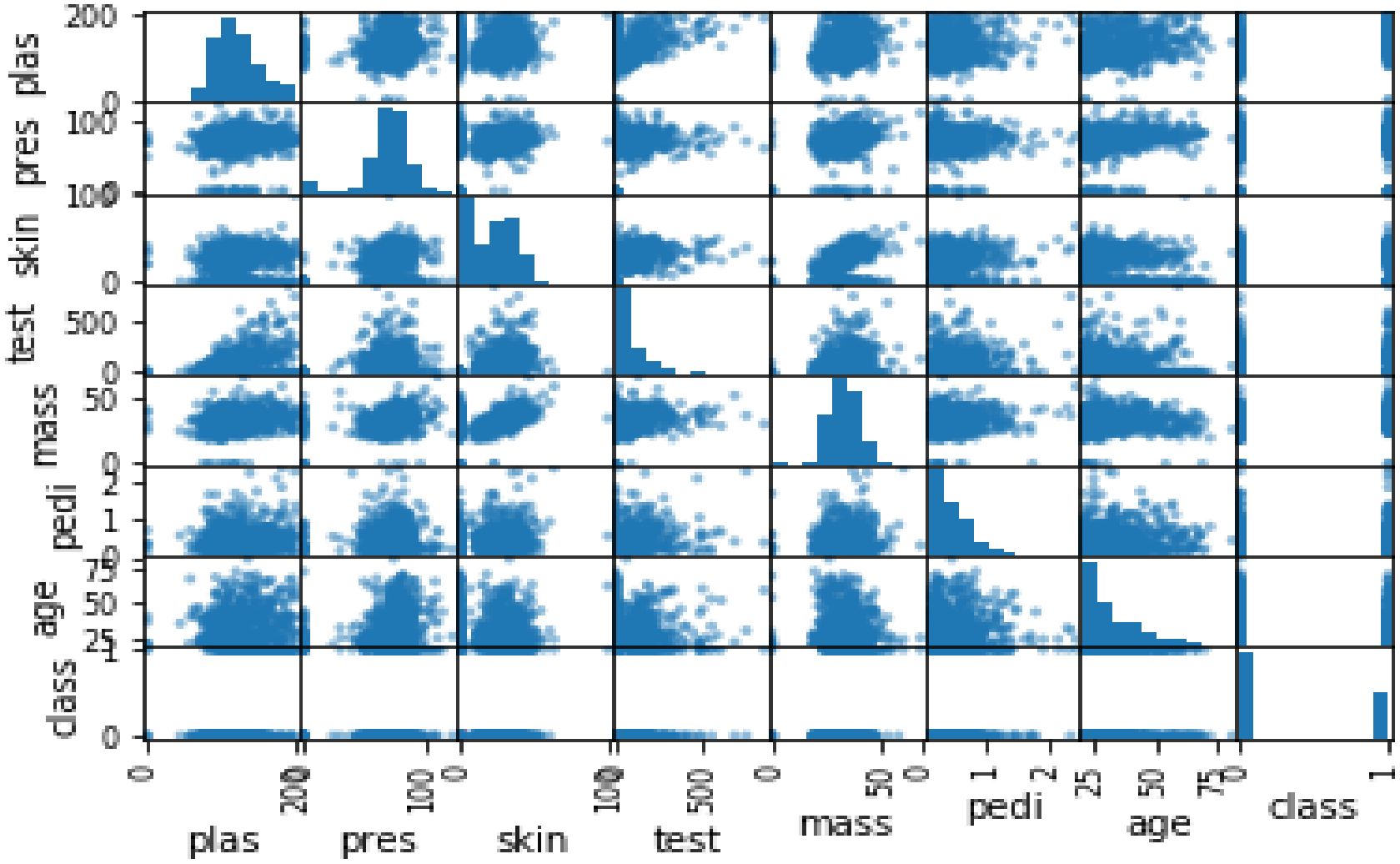
Scatter Matrix Plot

- Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions.
- Scatter plots are very much like line graphs in the concept that they use horizontal and vertical axes to plot data points.

Example

- In the following example, Python script will generate and plot Scatter matrix for the Pima Indian Diabetes dataset.
- It can be generated with the help of *scatter_matrix()* function on *Pandas* DataFrame and plotted with the help of *pyplot*.

```
from matplotlib import pyplot
from pandas import read_csv
from pandas.tools.plotting import scatter_matrix
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']
data = read_csv(path, names = names)
scatter_matrix(data)
pyplot.show()
```


plas
pres
skin
test
mass
pedi
age
class

Preparing Data

- Machine Learning algorithms are completely dependent on data because it is the most crucial aspect that makes model training possible.
- In simple words, we always need to feed right data i.e. the data in correct scale, format and containing meaningful features, for the problem we want machine to solve.
- This makes data preparation the most important step in ML process.
- Data preparation may be defined as the procedure that makes our dataset more appropriate for ML process.

Why Data Pre-processing?

- After selecting the raw data for ML training, the most important task is data pre-processing.
- In broad sense, data preprocessing will convert the selected data into a form we can work with or can feed to ML algorithms.
- We always need to preprocess our data so that it can be as per the expectation of machine learning algorithm.

Data Pre-processing Techniques

- We have the following data preprocessing techniques that can be applied on data set to produce data for ML algorithms:
- **Scaling**

Most probably our dataset comprises of the attributes with varying scale, but we cannot provide such data to ML algorithm hence it requires rescaling.

Data Pre-processing Techniques

- **Scaling**

Data rescaling makes sure that attributes are at same scale.

Generally, attributes are rescaled into the range of 0 and 1. ML algorithms like gradient descent and k-Nearest Neighbors requires scaled data. We can rescale the data with the help of *MinMaxScaler* class of *scikit-learn* Python library.

Data Pre-processing Techniques

- **Scaling - Example**

In this example we will rescale the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded (as done in the previous chapters) and then with the help of *MinMaxScaler* class, it will be rescaled in the range of 0 and 1.

The first few lines of the following script are same as we have written in previous chapters while loading CSV data.

Data Pre-processing Techniques

- **Scaling - Example**

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn import preprocessing
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Data Pre-processing Techniques

- **Scaling - Example**

Now, we can use *MinMaxScaler* class to rescale the data in the range of 0 and 1.

```
data_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
data_rescaled = data_scaler.fit_transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 1 and showing the first 10 rows in the output.

Data Pre-processing Techniques

- **Scaling – Example**

```
set_printoptions(precision=1)
print ("\nScaled data:\n", data_rescaled[0:10])
```

From the bellow output, all the data got rescaled into the range of 0 and 1.

Data Pre-processing Techniques

- **Scaling – Example**

Output

```
Scaled data:
[[0.4 0.7 0.6 0.4 0.  0.5 0.2 0.5 1. ]
[0.1  0.4 0.5 0.3 0.  0.4 0.1 0.2 0. ]
[0.5  0.9 0.5 0.  0.  0.3 0.3 0.2 1. ]
[0.1  0.4 0.5 0.2 0.1 0.4 0.  0.  0. ]
[0.   0.7 0.3 0.4 0.2 0.6 0.9 0.2 1. ]
[0.3  0.6 0.6 0.  0.  0.4 0.1 0.2 0. ]
[0.2  0.4 0.4 0.3 0.1 0.5 0.1 0.1 1. ]
[0.6  0.6 0.  0.  0.  0.5 0.  0.1 0. ]
[0.1  1.  0.6 0.5 0.6 0.5 0.  0.5 1. ]
[0.5  0.6 0.8 0.  0.  0.  0.1 0.6 1. ]]
```

Data Pre-processing Techniques

- **Normalization**

Another useful data preprocessing technique is Normalization.

This is used to rescale each row of data to have a length of 1. It is mainly useful in Sparse dataset where we have lots of zeros.

We can rescale the data with the help of *Normalizer* class of *scikit-learn* Python library.

Data Pre-processing Techniques

- Types of Normalization
- In machine learning, there are two types of normalization preprocessing techniques as follows :
  - L1 Normalization
  - L2 Normalization

Data Pre-processing Techniques

- **Binarization**

As the name suggests, this is the technique with the help of which we can make our data binary.

We can use a binary threshold for making our data binary.

The values above that threshold value will be converted to 1 and below that threshold will be converted to 0.

Data Pre-processing Techniques

- **Binarization**

For example, if we choose threshold value = 0.5, then the dataset value above it will become 1 and below this will become 0.

That is why we can call it **binarizing** the data or **thresholding** the data.

Data Pre-processing Techniques

- **Binarization**

This technique is useful when we have probabilities in our dataset and want to convert them into crisp values.

We can binarize the data with the help of *Binarizer* class of *scikit-learn* Python library

Data Pre-processing Techniques

- **Binarization - Example**

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded and then with the help of *Binarizer* class it will be converted into binary values i.e. 0 and 1 depending upon the threshold value. We are taking 0.5 as threshold value.

Data Pre-processing Techniques

- **Binarization - Example**

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

Data Pre-processing Techniques

- **Binarization – Example**

```
from pandas import read_csv
from sklearn.preprocessing import Binarizer
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Now, we can use *Binarize* class to convert the data into binary values.

Data Pre-processing Techniques

- **Binarization – Example**

```
binarizer = inarizer(threshold=0.5).fit(array)
Data_binarized = binarizer.transform(array)
```

- Here, we are showing the first 5 rows in the output.

```
print ("\nBinary data:\n", Data_binarized [0:5])
```

Data Pre-processing Techniques

Output

```
Binary data:
[[1. 1. 1. 1. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 0. 1. 0. 1. 0.]
[1.  1. 1. 0. 0. 1. 1. 1. 1.]
[1.  1. 1. 1. 1. 1. 0. 1. 0.]
[0.  1. 1. 1. 1. 1. 1. 1. 1.]]
```

Data Pre-processing Techniques

- **Data Labeling**

We discussed the importance of good data for ML algorithms as well as some techniques to pre-process the data before sending it to ML algorithms.

One more aspect in this regard is data labeling. It is also very important to send the data to ML algorithms having proper labeling.

For example, in case of classification problems, lot of labels in the form of words, numbers etc. are there on the data.

Data Pre-processing Techniques

- **What is Label Encoding?**

Most of the sklearn functions expect that the data with number labels rather than word labels. Hence, we need to convert such labels into number labels. This process is called label encoding.

We can perform label encoding of data with the help of *LabelEncoder()* function of *scikit-learn* Python library.

Data Pre-processing Techniques

- **Example**

In the following example, Python script will perform the label encoding.

First, import the required Python libraries as follows

```
import numpy as np
from sklearn import preprocessing
```

Data Pre-processing Techniques

- **Example**

Now, we need to provide the input labels as follows

```
input_labels =
['red','black','red','green','black','yellow','white']
```

The next line of code will create the label encoder and train it.

```
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

Data Pre-processing Techniques

- **Example**

The next lines of script will check the performance by encoding the random ordered list

```
test_labels = ['green','red','black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
encoded_values = [3,0,4,1]
decoded_list = encoder.inverse_transform(encoded_values)
```

Data Pre-processing Techniques

- **Example**

We can get the list of encoded values with the help of following python script :

```
print("\nEncoded values =", encoded_values)
print("\nDecoded labels =", list(decoded_list))
```

Data Pre-processing Techniques

- **Example**

Output

```
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]
Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
```

Thank you