Tutorial

Programme GFM MSc 2015, Module 13
Spatio-Temporal Analytics and Modelling

DATA EXPLORATION WITH PYTHON

some parts of this document are a translation of chapter 3 'R and Data Mining: Examples and Case Studies' by Y.Zhao

C. Piccinini, June 2015, version 2

The tutorial

This document shows examples on data exploration in Python. It starts with inspecting the dimensionality,structure and data of a Pandas DataFrame object, followed by basic statistics and various charts like pie charts and histograms. Exploration of multiple variables are then demonstrated, including grouped distribution, grouped boxplots, scattered plot. After that, examples are given on 3D plots, heatmaps, Andrew curves, and parallel coordinates. It also shows how to save charts to graphics files.

Students should have installed python 3.6.x and libraries (refer to the installation instructions available on Blackboard)

Students should have a basic knowledge of:

- python basics
- differences between python 2 and python 3
- matplotlib
- numpy

GFM students have learnt Python programming during the module 5-10 ‘programming skills’.

Other students are invited to review the material available on blackboard.

For this tutorial we advise using *IPython notebook*. (refer to the installation instructions available on Blackboard)

Pandas

For this tutorial you are going to use the Pandas library.
Complete Pandas documentation available at:
http://pandas.pydata.org/pandas-docs/stable/index.html

- Pandas is a Python library wich provides high-performance and easy-to-use data structures.
- Pandas is used for data modelling and data analysis

Main components of Pandas:

- A set of labeled array data structures, the primary of which are Series/TimeSeries and DataFrame
- Index objects enabling both simple axis indexing and multi-level / hierarchical axis indexing
- An integrated group by engine for aggregating and transforming data sets
- Date range generation (date_range) and custom date offsets enabling the implementation of customized frequencies
- Input/Output tools: loading tabular data from flat files (CSV, delimited, Excel 2003), and saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
- Memory-efficent “sparse” versions of the standard data structures for storing data that is mostly missing or mostly constant (some fixed value)
- Moving window statistics (rolling mean, rolling standard deviation, etc.)
- Static and moving window linear and panel regression

Pandas data structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
1	TimeSeries	Series with index containing datetimes
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed columns
3	Panel	General 3D labeled, also size-mutable array

To know more about Pandas refer to the short tutorial available on Blackboard. You can also broaden your knowledge with the following material:

- Intro to Data Structures http://pandas.pydata.org/pandas-docs/stable/dsintro.html

- Essential Basic Functionality: http://pandas.pydata.org/pandas-docs/stable/basics.html

- Indexing and Selecting Data: http://pandas.pydata.org/pandas-docs/stable/indexing.html

1) Have a look at data

The iris data is used in this chapter for demonstration of data exploration in Python.
Find the iris file on blackboard.
We first check the size and structure of data.

- A coincise summary of a DataFrame can be obtained with **DataFrame.info()**
- **DataFrame.describe()** returns various summary statistics, excluding NaN values
- **DataFrame.shape** returns the number of rows and columns

First we set the inline plotting
In [ ]: `%matplotlib inline`

We import the necessary modules and we set a nice plotting style

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
pd.options.display.mpl_style='default'
```

We open the csv file using **Pandas.read_csv()** (more info at http://pandas.pydata.org/pandas-docs/stable/io.html)

In [ ]:
```
#read a csv file to a dataframe; here we need to assign column names; we use species name for the row index
df=pd.read_csv(r'[path]\iris.dataComma.txt', index_col=4,header=None,
names=['sepal length','sepal width','petal length','petal width','species'])
df   #in IPython this prints a formatted table, in a program you would write print(df)
```

	sepal length	sepal width	petal length	petal width
species				
Iris-setosa	5.1	3.5	1.4	0.2
Iris-setosa	4.9	3.0	1.4	0.2
Iris-setosa	4.7	3.2	1.3	0.2
Iris-setosa	4.6	3.1	1.5	0.2
Iris-setosa	5.0	3.6	1.4	0.2
Iris-setosa	5.4	3.9	1.7	0.4
Iris-setosa	4.6	3.4	1.4	0.3
Iris-setosa	5.0	3.4	1.5	0.2
Iris-setosa	4.4	2.9	1.4	0.2
Iris-setosa	4.9	3.1	1.5	0.1
Iris-setosa	5.4	3.7	1.5	0.2
Iris-setosa	4.8	3.4	1.6	0.2
Iris-setosa	4.8	3.0	1.4	0.1
Iris-setosa	4.3	3.0	1.1	0.1

In [ ]: df.info()

Out [ ]:
```
<class 'pandas.core.frame.DataFrame'>
Index: 150 entries, Iris-setosa to Iris-virginica
Data columns (total 4 columns):
sepal length    150 non-null float64
sepal width     150 non-null float64
petal length    150 non-null float64
petal width     150 non-null float64
dtypes: float64(4)None
memory usage: 5.3+ KB
```

In [ ]: df.describe()

	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 [ ]:
```
# get only the shape of the dataframe
df.shape
```

Out [ ]: (150, 4)

We can get a list of columns and unique rows with **DataFrame.columns** and **DataFrame.index**. Use **.unique()** to get unique names.

In[ ]: `list(df.columns) #get a list with the names of the columns`

Out[ ]: ['sepal length', 'sepal width', 'petal length', 'petal width']

In[ ]: `df.dtypes #get the type of each column`

Out[ ]:
```
sepal length    float64
sepal width     float64
petal length    float64
petal width     float64
dtype: object
```

In[ ]: `list(df.index.unique()) #get a list with the unique names of the rows`

Out[ ]: ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

We have a look at the first three rows of data using slicing. The first or last rows of data can also be retrieved with **DataFrame.head()**or **DataFrame.tail()**

In[ ]: df[:3] #Get only the first 3 rows, this is the same as df[0:3] (last index is not comprised in the slice)

	sepal length	sepal width	petal length	petal width
species				
Iris-setosa	5.1	3.5	1.4	0.2
Iris-setosa	4.9	3.0	1.4	0.2
Iris-setosa	4.7	3.2	1.3	0.2

In[ ]: df.head(3) # by default it will return 5 rows

	sepal length	sepal width	petal length	petal width
species				
Iris-setosa	5.1	3.5	1.4	0.2
Iris-setosa	4.9	3.0	1.4	0.2
Iris-setosa	4.7	3.2	1.3	0.2

In[ ]: df[-3:] #Get only the last 3 rows, this is the same as df[-3:150]

	sepal length	sepal width	petal length	petal width
species				
Iris-virginica	6.5	3.0	5.2	2.0
Iris-virginica	6.2	3.4	5.4	2.3
Iris-virginica	5.9	3.0	5.1	1.8

In[ ]: df.tail(3) #by default it will return 5 rows

	sepal length	sepal width	petal length	petal width
species				
Iris-virginica	6.5	3.0	5.2	2.0
Iris-virginica	6.2	3.4	5.4	2.3
Iris-virginica	5.9	3.0	5.1	1.8

We can also retrieve the values of a single column (which is a Pandas Series). For example, the first 4 values of Sepal.Length can be fetched with the code below:

In[ ]: `df['sepal length'][:4]` #find the sepal lenght of the first 4 rows ; this will return a Pandas Series

Out[ ]:
```
species
Iris-setosa    5.1
Iris-setosa    4.9
Iris-setosa    4.7
Iris-setosa    4.6
Name: sepal length, dtype: float64
```

2) Explore individual variables

We have already learnt that **DataFrame.describe()** returns various summary statistics for each column including:

- number of rows
- mean value
- std value
- min value
- max value
- first quartile
- second quartile
- third quartile

In[ ]: `df.describe()`

	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

The mean, median, min, max, median, mode, std, variance, and quantile can also be obtained with functions:

- **DataFrame.mean()**
- **DataFrame.median()**
- **DataFrame.min()**
- **DataFrame.max()**
- **DataFrame.var()**
- **DataFrame.quantile()**

```
In[ ]:   df.mean(0)  #pass 0 to calculate the columns mean; pass 1 to calculate the rows mean; 0 is the default

Out[ ]:  sepal length    5.843333
         sepal width     3.054000
         petal length    3.758667
         petal width     1.198667
         dtype: float64

In[ ]:   df.median(0)

Out[ ]:  sepal length    5.80
         sepal width     3.00
         petal length    4.35
         petal width     1.30
         dtype: float64

In[ ]:   df.mode(0)

Out[ ]:  sepal length sepal width petal length  petal width
         5            3           1.5           0.2

In[ ]:   df.std(0)

Out[ ]:  sepal length    0.828066
         sepal width     0.433594
         petal length    1.764420
         petal width     0.763161
         dtype: float64

In[ ]:   df.min(0)

Out[ ]:  sepal length    4.3
         sepal width     2.0
         petal length    1.0
         petal width     0.1
         dtype: float64

In[ ]:  df.max(0)

Out[ ]:  sepal length    7.9
         sepal width     4.4
         petal length    6.9
         petal width     2.5
         dtype: float64

In[ ]: df.var(0)

Out[ ]:  sepal length    0.685694
         sepal width     0.188004
         petal length    3.113179
         petal width     0.582414
         dtype: float64
```

In[ ]: df.quantile([0,0.25,0.5,0.75,1])

	sepal length	sepal width	petal length	petal width
0.00	4.3	2.0	1.00	0.1
0.25	5.1	2.8	1.60	0.3
0.50	5.8	3.0	4.35	1.3
0.75	6.4	3.3	5.10	1.8
1.00	7.9	4.4	6.90	2.5

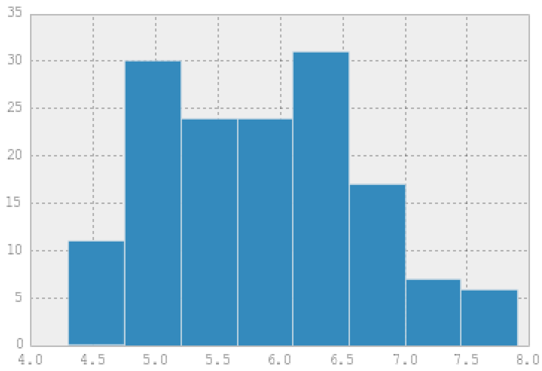
To get values for a single column we use the index

In[ ]: df['sepal length'].var()
Out[ ]: 0.68569351230421827

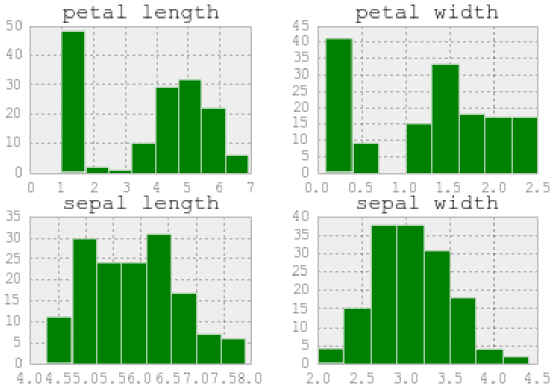
We check a distribution with histogram and density using **Series/DataFrame.hist()** and **Series/DataFrame.plot()**.

These functions are just a wrapper around **matplotlib.pyplot.plot()**

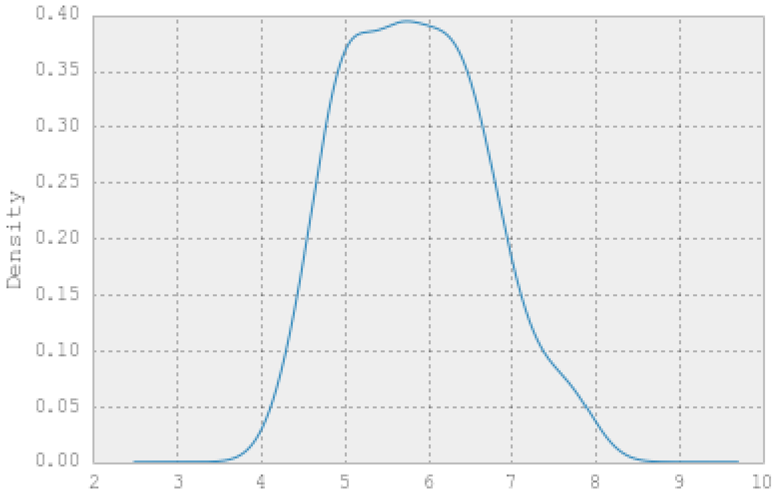
In[ ]:
```
df['sepal length'].hist(bins=8)
plt.show()   # .show() is not necessary in IPython
```



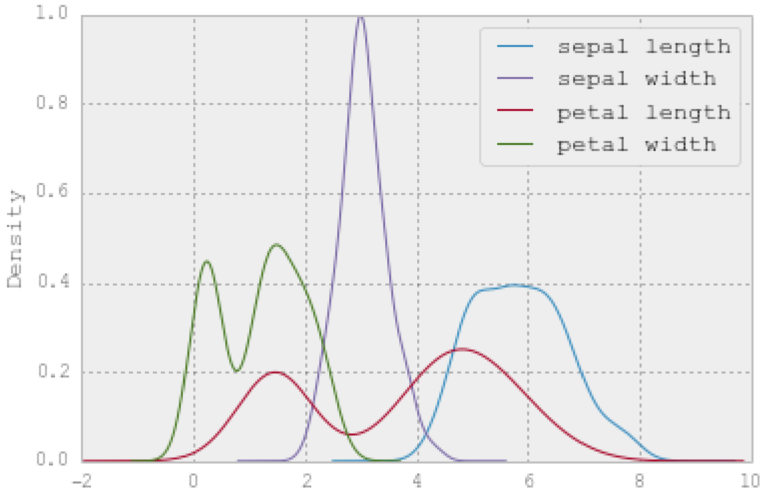
In[ ]:
```
df.hist(bins=8,color='g')
plt.show()
```



In[ ]:
```
df['sepal length'].plot(kind='kde')
plt.show()
```



In[ ]:
```
df.plot(kind='kde')
plt.show()
```

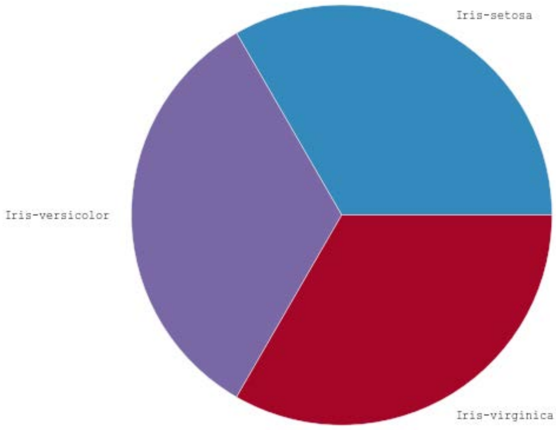


The frequency of factors can be calculated after grouping data with **DataFrame.groupby()**, and then plotted as a pie chart or a bar chart with **Series.plot()**

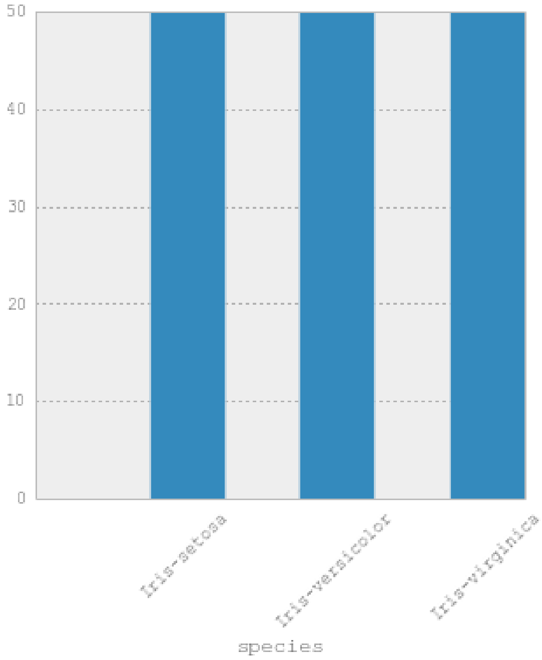
In[ ]:
```
#here we get a Pandas.Series with the size for each species
a=df['sepal length'].groupby(level='species').size() #if you use count you would get the number of non-NAN
a
```

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

In[ ]:
```
a.plot(kind='pie',figsize=(8,8) )
plt.show()
```



In[ ]:
```
a.plot(kind='bar',figsize=(5,5), rot=45)
plt.show()
```



3) Explore multiple variables

After checking the distributions of individual variables, we then investigate the relationships between two variables.

Below we calculate covariance and correlation between variables with **DataFrame.cov()** and **DataFrame.cor()**.

In[ ]: df.cov()

	sepal length	sepal width	petal length	petal width
sepal length	0.685694	-0.039268	1.273682	0.516904
sepal width	-0.039268	0.188004	-0.321713	-0.117981
petal length	1.273682	-0.321713	3.113179	1.296387
petal width	0.516904	-0.117981	1.296387	0.582414

In[ ]: df.corr()

	sepal length	sepal width	petal length	petal width
sepal length	1.000000	-0.109369	0.871754	0.817954
sepal width	-0.109369	1.000000	-0.420516	-0.356544
petal length	0.871754	-0.420516	1.000000	0.962757
petal width	0.817954	-0.356544	0.962757	1.000000

We compute the stats of 'sepal length' of every species with **Series.groupby()** and **SeriesGroupBy.describe()**

In[ ]:
```
a=df['sepal length'].groupby(level='species').describe()
print (a)
```

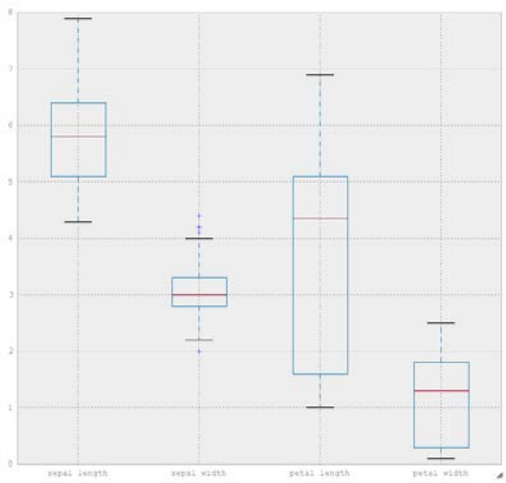
Out[]:
```
species
Iris-setosa   count    50.000000
              mean      5.006000
              std       0.352490
              min       4.300000
              25%       4.800000
              50%       5.000000
              75%       5.200000
              max       5.800000

Iris-versicolor  count    50.000000
                 mean      5.936000
                 std       0.516171
                 min       4.900000
                 25%       5.600000
                 50%       5.900000
                 75%       6.300000
                 max       7.000000
Iris-virginica   count    50.000000
                 mean      6.588000
                 std       0.635880
                 min       4.900000
                 25%       6.225000
                 50%       6.500000
```

```
75%      6.900000
max      7.900000
dtype: float64
```

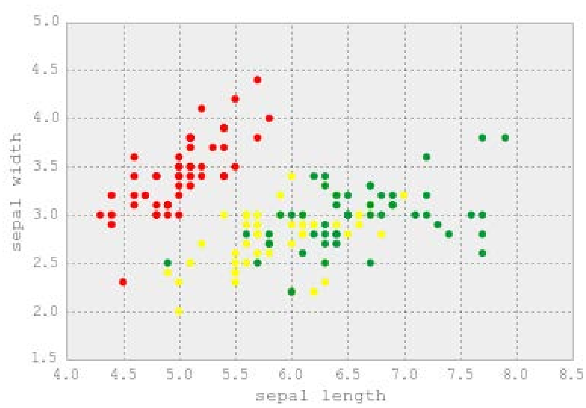
We use the function **DataFrame.boxplot()** to plot a box plot, also known as box-and-whisker plot, to show the median, first and third quartile of a distribution (i.e., the 50%, 25% and 75% points in cumulative distribution), and outliers. The bar in the middle is the median. The box shows the interquartile range (IQR), which is the range between the 75% and 25% observation.

In[ ]:
```
df.boxplot()
plt.show()
```



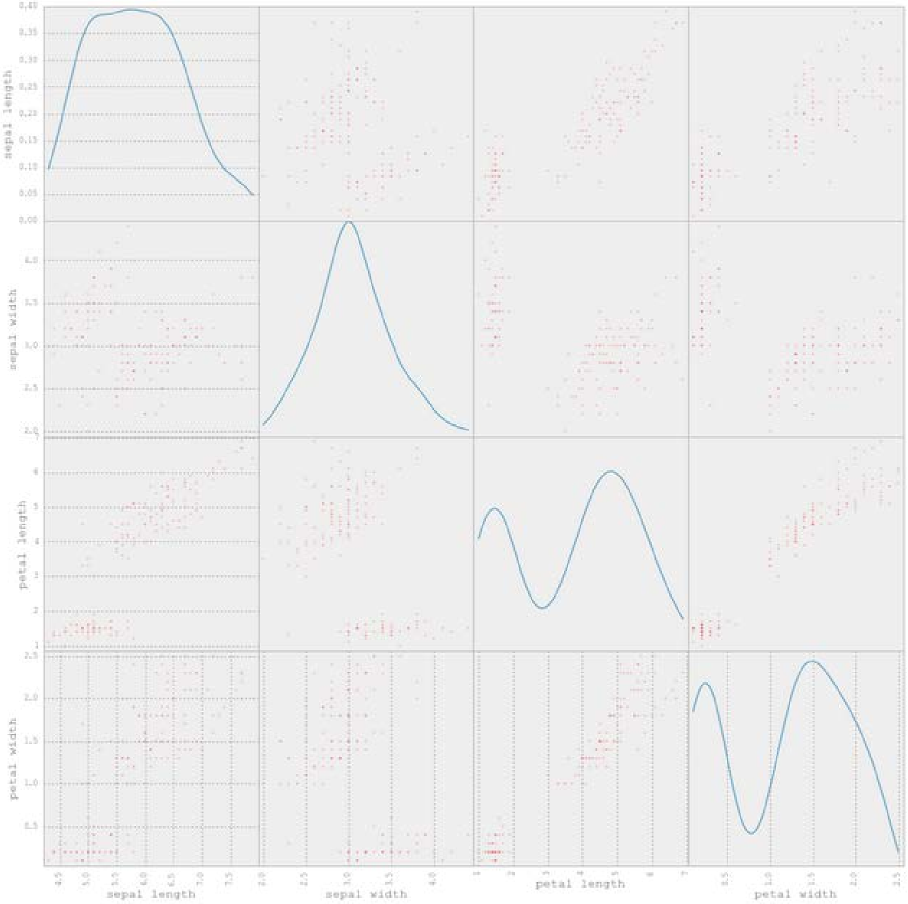
A scatter plot can be drawn for two numeric variables with **DataFrame.plot()**

In[ ]:
```
indexes=df.index
species=list(indexes.unique())
color_set = ('#FF0000', '#FFFF00', '#009933')
color_list = [color_set[species.index(label)] for label in indexes]
df.plot(kind='scatter',x='sepal length',y='sepal width', c = color_list)
plt.show()
```



A matrix of scatter plots can be produced using the **scatter_matrix** method in **pandas.tools.plotting**

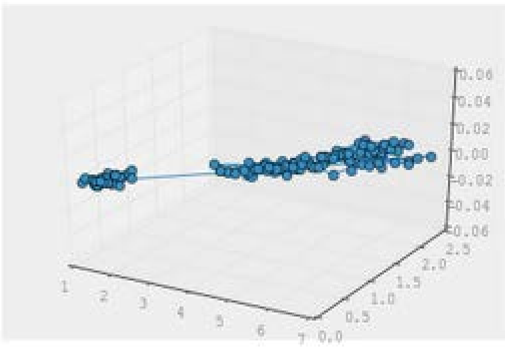
In[ ]:
```
from pandas.tools.plotting import scatter_matrix
scatter_matrix(df,alpha=0.2, figsize=(15,15),diagonal='kde',c='r')
plt.show()
```



4) More exploration

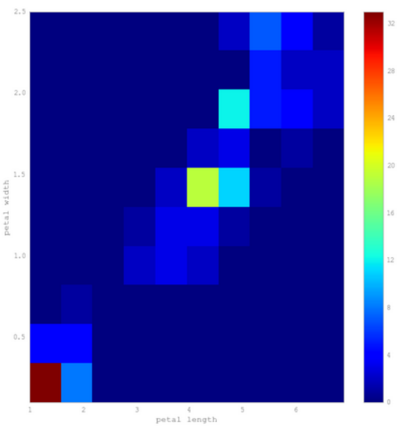
This section presents some fancy graphs, including 3D plots, heatmaps,hexagonal bin plots, andrews curves, and parallel coordinates.

In[ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = list(df['petal length'])
y=  list(df['petal width'])
ax.plot(x, y, zs=0, zdir='z',marker='o' )
plt.show()
```



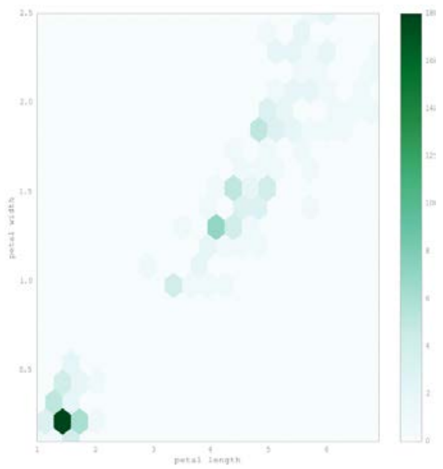
A heatmap can be created with **matplotlib.pyplot.hist2d()**

In[ ]:
```
plt.figure(figsize=(10, 10))
x = list(df['petal length'])
y= list(df['petal width'])
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.grid(False)
plt.hist2d(x, y)
plt.colorbar()
plt.show()
```



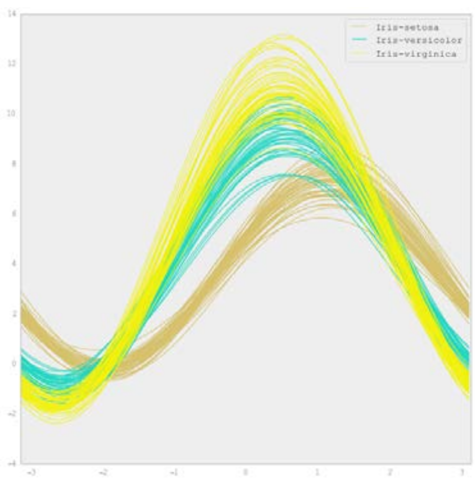
Another type of heatmap called hexagonal bin plot can be created with **DataFrame.plot()**

In[ ]:
```
df.plot(kind='hexbin', x='petal length', y='petal width',gridsize=20,figsize=(10, 10))
plt.show()
```



Andrews curves allow one to plot multivariate data as a large number of curves that are created using the attributes of samples as coefficients for Fourier series. By coloring these curves differently for each class it is possible to visualize data clustering. Curves belonging to samples of the same class will usually be closer together and form larger structures.

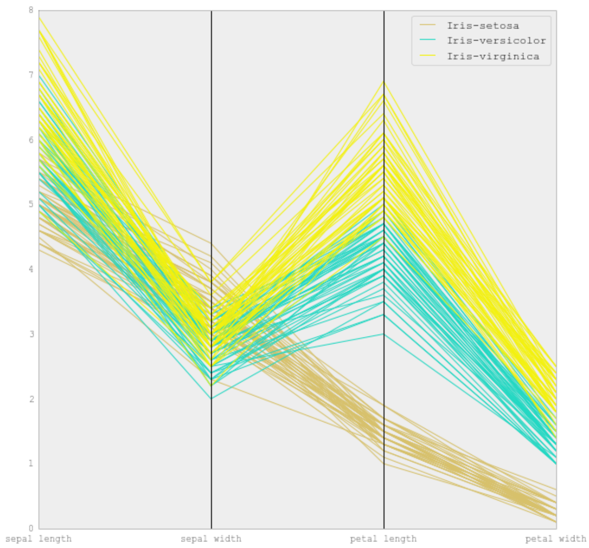
In[ ]:
```
from pandas.plotting import andrews_curves
#here we load again the data because we need a column with the spacies name
data=pd.read_csv(r'[path]\iris.dataComma.txt',header=None,names=['sepal length','sepal width','petal length','petal width','species'])
plt.figure(figsize=(10, 10))
andrews_curves(data,'species')
plt.show()
```



Parallel coordinates is a plotting technique for plotting multivariate data. It allows one to see clusters in data and to estimate other statistics visually. Using parallel coordinates points are represented as connected line segments. Each vertical line represents one attribute. One set of connected line segments represents one data point. Points that tend to cluster will appear closer together.

To save a plot use **matplotlib.pyplot.savefig()**

In[ ]:
```
from pandas.plotting import parallel_coordinates
plt.figure(figsize=(10, 10))
parallel_coordinates(data, 'species')
plt.savefig(r'[path]\parallel.png', format='png')
plt.show()
```



More info about visualization are available at http://pandas.pydata.org/pandas-docs/dev/visualization.html

INTERESTING LIBRARIES TO COMBINE WITH PANDAS

Statsmodels: allows users to explore data, estimate statistical models, and perform statistical tests
http://statsmodels.sourceforge.net/

sklearn-pandas: provides a bridge between Scikit-Learn's machine learning methods and pandas-style Data Frames.
https://github.com/paulgb/sklearn-pandas

Geopandas: add support to geographic data with Geoseries and Geodataframe
https://github.com/kjordahl/geopandas

VISUALIZATION

ggplot: mimics the R ggplot2 plottong library
https://github.com/yhat/ggplot

Seaborn: visualization library based on matplotlib and pandas
http://stanford.edu/~mwaskom/software/seaborn/

Bokeh: a Python interactive visualization library that targets modern web browsers for presentation
http://bokeh.pydata.org/en/latest/