Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

NumPy, Pandas and Matplotlib

Numpy ,Pandas and Matplotlib

a. Pandas Module
   i. Series
   ii. DataFrames

b. Numpy Module
   i. Numpy arrays
   ii. Numpy operations

c. Matplotlib module
   i. Plotting information
   ii. Bar Charts and Histogram
   iii. Box and Whisker Plots
   iv. Heatmap
   v. Scatter Plots

NumPy Overview

NumPy is a package in python for scientific computing. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Features

- Multi dimensional array
- Methods for processing arrays
- Element by element operations
- Mathematical operations like linear algebra

Categories of NumPy Operations

- Mathematical and Logical
- Fourier transform and shape manipulation
- Linear algebra and random number generation

NumPy Array

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

```
import numpy as np
a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))          # Prints "<class 'numpy.ndarray'>"
print(a.shape)          # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                # Change an element of the array
print(a)                # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

NumPy
Functions to create Array

Numpy also provides many functions to create arrays:

```
import numpy as np
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #         [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #         [ 7.  7.]]"
```

```
d = np.eye(2)        # Create a 2x2 identity matrix
print(d)             # Prints "[[ 1.  0.]
                     #         [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with
                             random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

NumPy
Array Indexing

Numpy offers several ways to index into arrays:

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

NumPy
Integer array indexing

Integer array indexing: When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

NumPy
Boolean array indexing

Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

NumPy Operations
Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

NumPy Operations
Array math contd..

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

NumPy Operations
Transpose

Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest example of this type of operation is transposing a matrix; to transpose a matrix, simply use the **T** attribute of an array object:

```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x)    # Prints "[[1 2]
            #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]"
```

NumPy Operations
Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

Pandas Series

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

A pandas Series can be created using the following constructor −

```
pandas.Series( data, index, dtype, copy)
```

Parameters

data
data takes various forms like ndarray, list, constants

index
Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.

dtype
dtype is for data type. If None, data type will be inferred

copy
Copy data. Default False

A series can be created using various inputs like –

- Array
- Dict
- Scalar value or constant **(Matrix slide)**

Create an Empty Series

A basic series, which can be created is an Empty Series.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3…. **range(len(array))-1].**

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Its **output** is as follows –

```
0   a
1   b
2   c
3   d
dtype: object
```

Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

Its **output** is as follows –

```
a 0.0
b 1.0
c 2.0
dtype: float64
```

Observe – Dictionary keys are used to construct index.

Create a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

Its **output** is as follows –

```
0  5
1  5
2  5
3  5
dtype: int64
```

Pandas DataFrame

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Features of DataFrame

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

A pandas DataFrame can be created using the following constructor −
pandas.DataFrame(data, index, columns, dtype, copy)

Pandas DataFrame

Parameter & Description
data data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
index For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
columns For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
dtype Data type of each column.
copy This command (or whatever it is) is used for copying of data, if the default is False.

Create DataFrame

A pandas DataFrame can be created using various inputs like –

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

Create an Empty DataFrame

A basic DataFrame, which can be created is an Empty Dataframe.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

Its **output** is as follows –

```
Empty DataFrame
Columns: []
Index: []
```

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

Example 1

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
    0
0   1
1   2
2   3
3   4
4   5
```

Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.
If no index is passed, then by default, index will be range(n), where **n** is the array length.

Example 1

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
      Age       Name
0      28        Tom
1      34       Jack
2      29      Steve
3      42      Ricky
```

Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
    a    b      c
0   1    2    NaN
1   5   10   20.0
```

Note – Observe, NaN (Not a Number) is appended in missing areas.

Create a DataFrame from Dict of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df
```

Its **output** is as follows –

```
      one    two
a     1.0    1
b     2.0    2
c     3.0    3
d     NaN    4
```

Note – Observe, for the series one, there is no label **'d'** passed, but i result, for the **d** label, NaN is appended with NaN.

Column Selection

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df ['one']
```

Its **output** is as follows –

```
a     1.0
b     2.0
c     3.0
d     NaN
Name: one, dtype: float64
```

Pandas DataFrame

Column Addition

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with column label

print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print df

print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']

print df
```

Its **output** is as follows –

```
Adding a new column by passing as Series:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN

Adding a new column using the existing columns in DataFrame:
   one  two  three  four
a  1.0    1   10.0  11.0
b  2.0    2   20.0  22.0
c  3.0    3   30.0  33.0
d  NaN    4    NaN   NaN
```

Pandas DataFrame (Merginng, concat, group by)

Column Deletion

```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print df

# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print df

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print df
```

Its **output** is as follows –

```
Our dataframe is:
      one   three  two
a     1.0    10.0   1
b     2.0    20.0   2
c     3.0    30.0   3
d     NaN     NaN   4

Deleting the first column using DEL function:
      three    two
a     10.0     1
b     20.0     2
c     30.0     3
d     NaN      4

Deleting another column using POP function:
   three
a  10.0
b  20.0
c  30.0
d  NaN
```

Pandas DataFrame

Row Selection

Slice Rows

Multiple rows can be selected using ' : ' operator.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df[2:4]
```

Its **output** is as follows –

```
      one    two
c     3.0     3
d     NaN     4
```

Selection by Label

Rows can be selected by passing row label to a **loc** function.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df.loc['b']
```

Its **output** is as follows –

```
one 2.0
two 2.0
Name: b, dtype: float64
```

Row Addition

Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)
print df
```

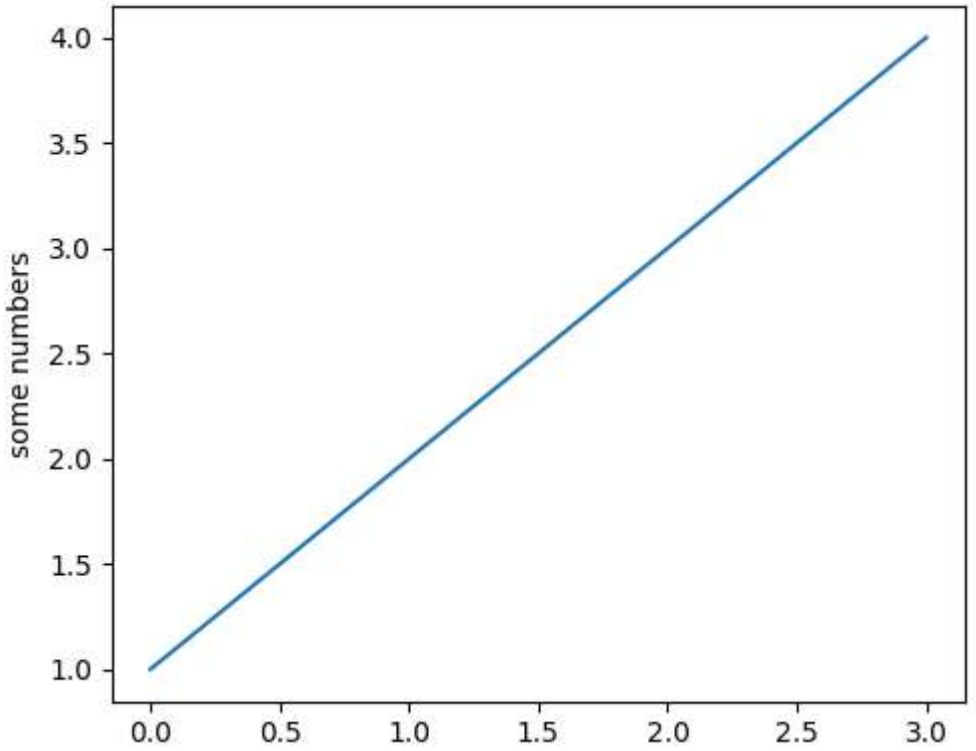
Its **output** is as follows –

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

Pandas
Matplotlib – Plotting Information

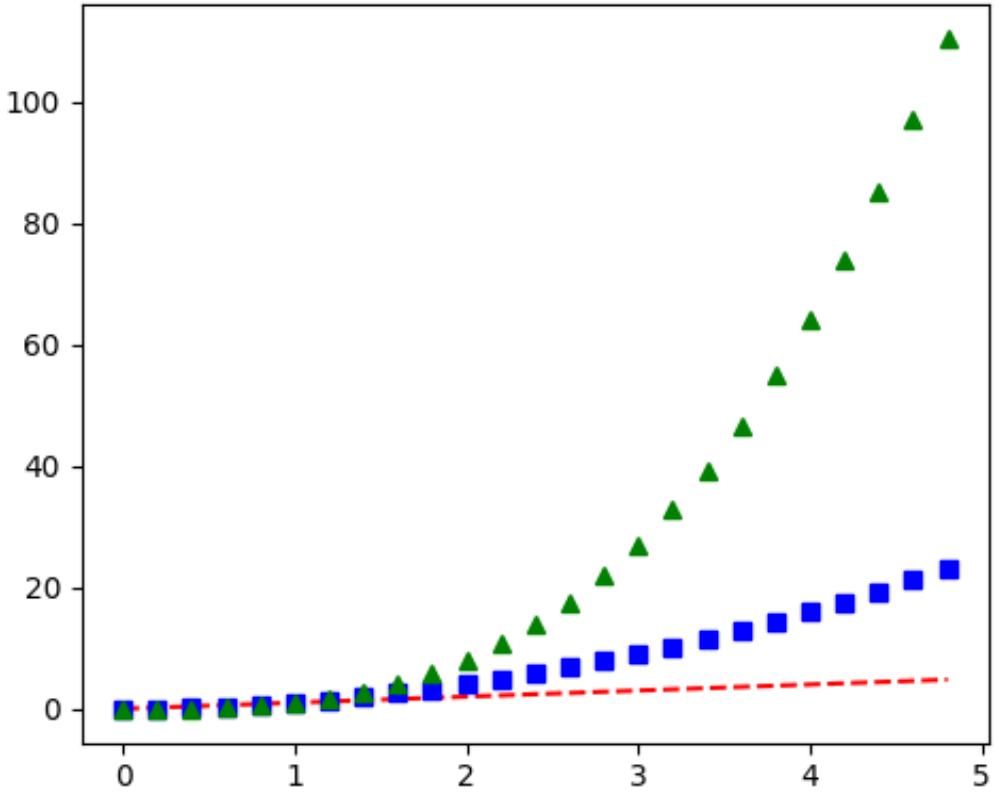
matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB.
Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes.

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```



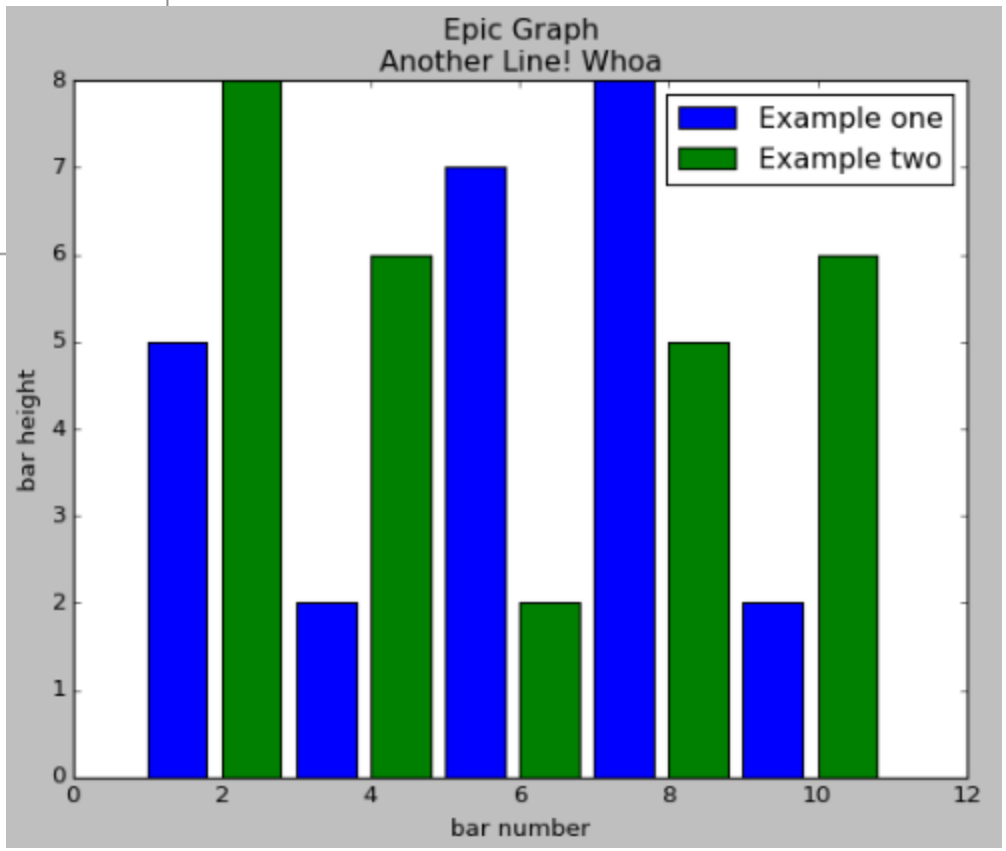
Pandas
Matplotlib – Plotting Information

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

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



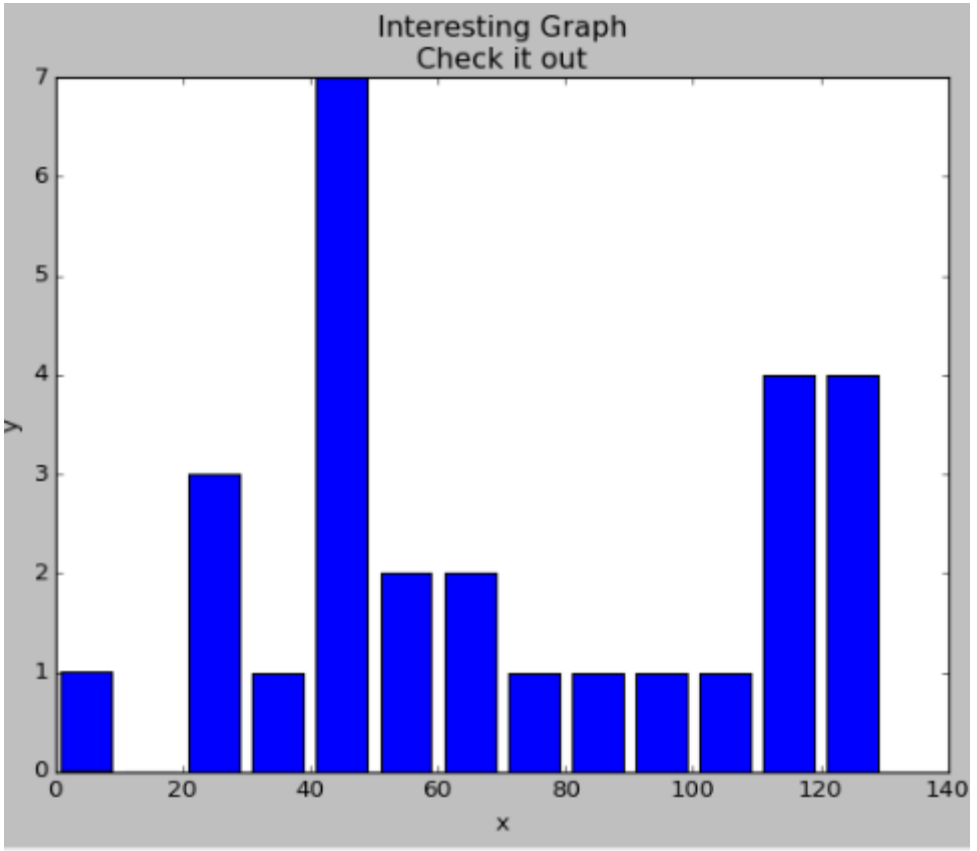
Pandas
Matplotlib – Bar Charts

```
import matplotlib.pyplot as plt
      plt.bar([1,3,5,7,9],[5,2,7,8,2], label="Example one")

plt.bar([2,4,6,8,10],[8,6,2,5,6], label="Example two", color='g')
plt.legend()
plt.xlabel('bar number')
plt.ylabel('bar height')

plt.title('Epic Graph\nAnother Line! Whoa')

plt.show()
```



Matplotlib – Histogram

```
import matplotlib.pyplot as plt

population_ages =
[22,55,62,45,21,22,34,42,42,4,99,102,110,120,121,122,1
30,111,115,112,80,75,65,54,44,43,42,48]

bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]

plt.hist(population_ages, bins, histtype='bar', rwidth=0.8)

plt.xlabel('x')

plt.ylabel('y')

plt.title('Interesting Graph\nCheck it out')

plt.legend()

plt.show()
```



Matplotlib – Box and Whisker Plots

matplotlib.pyplot.boxplot

```
matplotlib.pyplot.boxplot(x, notch=None, sym=None, vert=None, whis=None, positions=None, widths=None, patch_artist=None,
bootstrap=None, usermedians=None, conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None,
showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None, meanprops=None, capprops=None,
whiskerprops=None, manage_xticks=True, autorange=False, zorder=None, *, data=None)                                [s
```

Sample code

https://matplotlib.org/examples/pylab_examples/boxplot_demo.html

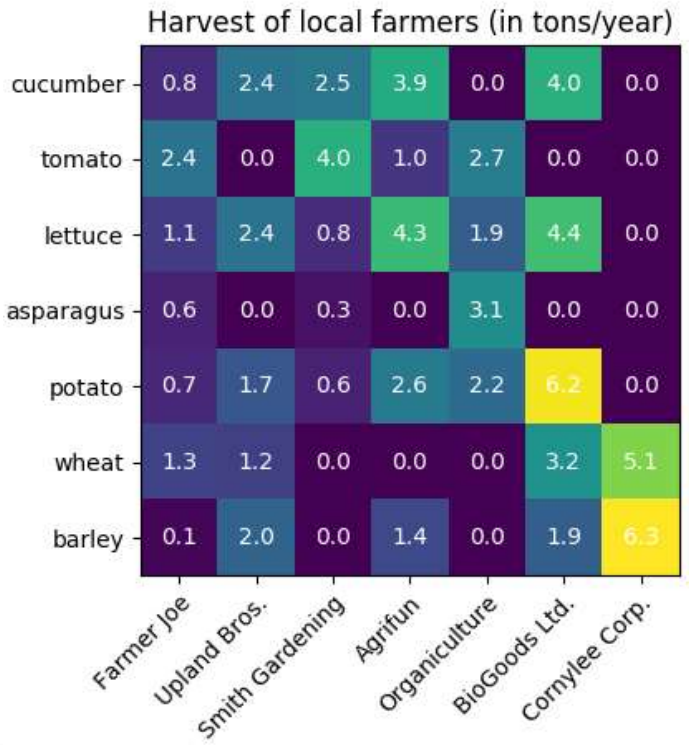
Pandas
Matplotlib – Heatmaps

Creating annotated heatmaps

It is often desirable to show data which depends on two independent variables as a color coded image plot. This is often referred to as a heatmap. If the data is categorical, this would be called a categorical heatmap. Matplotlib's imshow function makes production of such plots particularly easy.

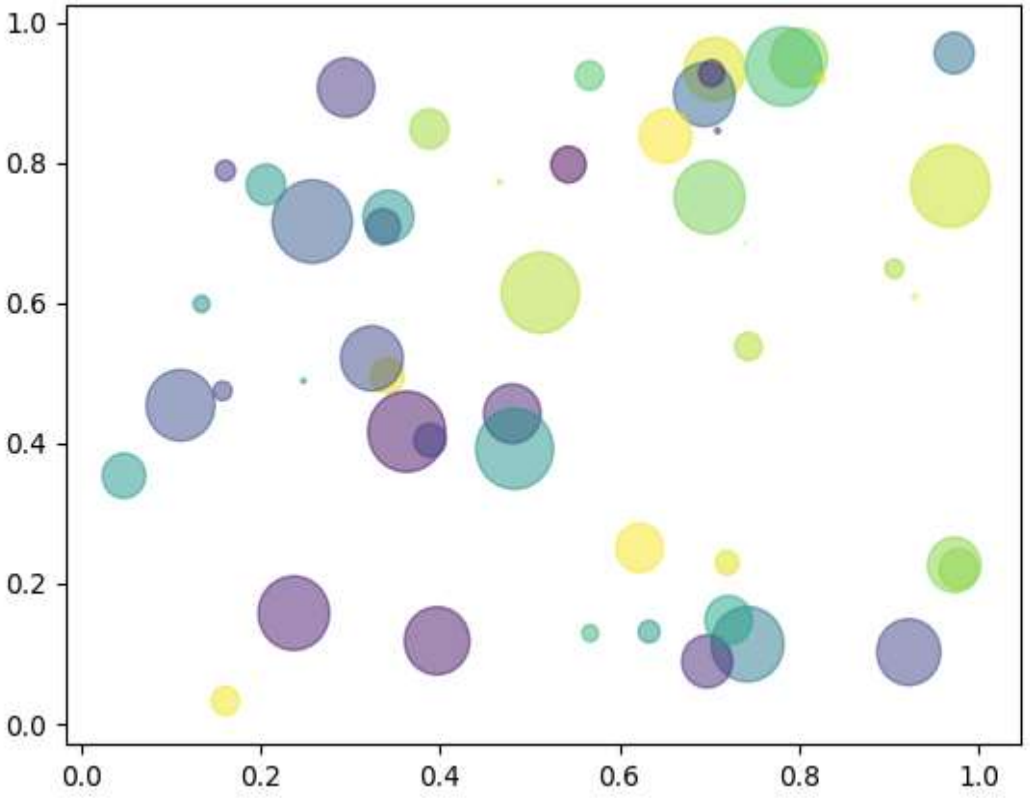
Sample code

https://matplotlib.org/gallery/images_contours_and_fields/image_annotated_heatmap.html



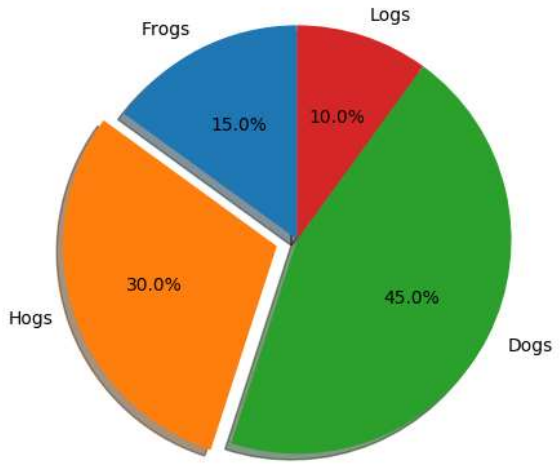
Pandas
Matplotlib – Scatter Plot

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

# Fixing random state for reproducibility
np.random.seed(19680801)


N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



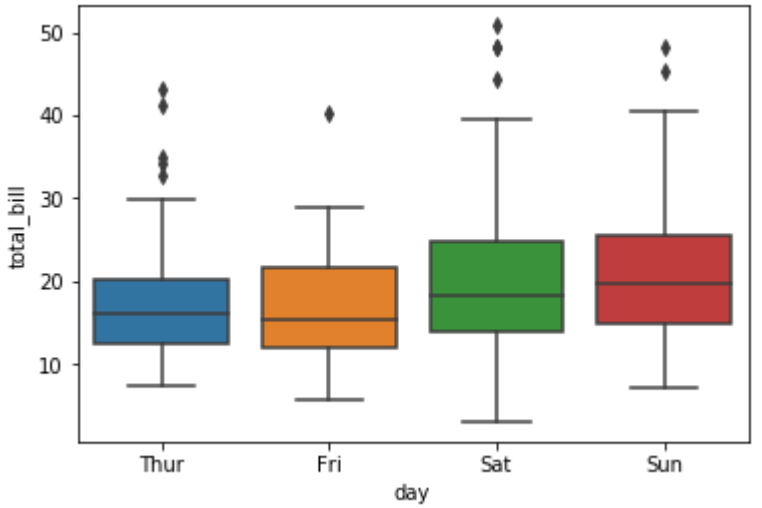
Matplotlib – Pie Chart

```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



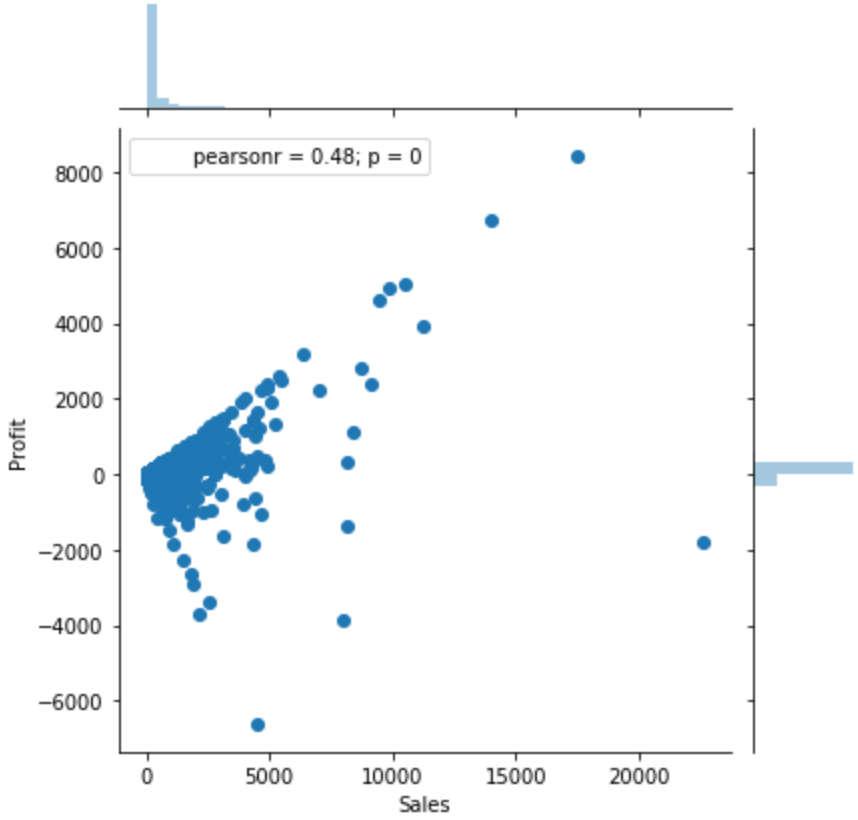
Matplotlib – Box and Whisker Plot

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

tips=sns.load_dataset("tips")
ax=sns.boxplot(x="day", y=tips["total_bill"], data=tips)
```



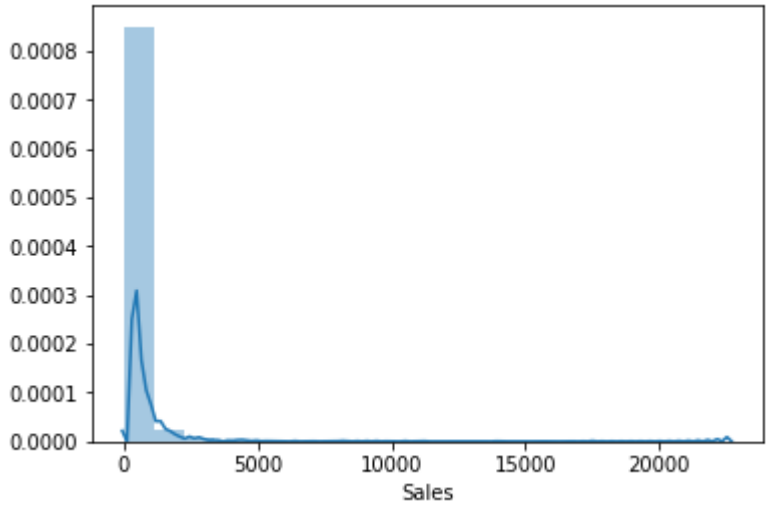
Matplotlib – Jointplot

```
import pandas as pd
import os
os.chdir('D:\\Python A-Z\\Module 1 - Python Basics\\')

sample_super_store=pd.read_excel('Sample - Superstore.xls')

from matplotlib import pyplot as plt
import seaborn as sns

jp1 = sns.jointplot(data=sample_super_store, x = 'Sales', y = 'Profit')
```



Matplotlib – Distplot

```
import pandas as pd
import os
os.chdir('D:\\Python A-Z\\Module 1 - Python Basics\\')

sample_super_store=pd.read_excel('Sample - Superstore.xls')

from matplotlib import pyplot as plt
import seaborn as sns

m1 = sns.distplot(sample_super_store.Sales, bins=20)
```




Cognixia
A Collabera LEARNING SOLUTIONS COMPANY
THANK YOU
45%