PracticalML_Tutorial

July 13, 2021

1 Practical Machine Learning using Python

1.1 Objectives

This tutorial plans to give an introduction to Python based tools and packages for big data analytics. This will cover * Why Python * Setting up the environment * Anaconda and Jupyter Notebook * Python Basic data structures and programming constructs * Packages for Data Analytics * Numpy * Matplotlib * Pandas * Scikitlearn * Case Study- Semantic Analysis

1.2 Why learn Python for Machine Learning?

Here are some reasons which go in favour of learning Python: * Open Source – free to install * Awesome online community * Very easy to learn * Can become a common language for data science and production of web based analytics products.

Needless to say, it still has few drawbacks too: It is an interpreted language rather than compiled language – hence might take up more CPU time. **However, given the savings in programmer time (due to ease of learning), it might still be a good choice.**

1.3 Setting up the environment

For analysing your data with Python, you have to first set up your programming environment. There are multiple ways to do this. One of the recommended method is iPython notebook. It provides a lot of good features for documenting while writing the code itself and you can choose to run the code in blocks (rather than the line by line execution)

We will use iPython environment **(Jupyter Notebook)** for this complete tutorial.

ANACONDA is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution of Python and R and includes over 100 of the most popular Python, R and Scala packages for data science. Additionally, you'll have access to over 720 packages that can easily be installed with conda, our renowned package, dependency and environment manager, that is included in Anaconda. Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.

You can also use Google colab, a cloud based notebook environment.

1.4 Python Basics

```
In [1]: print "Hello World"
```

```
Hello World
```

1.4.1 To execute the cell press ctl+Enter or shift+Enter

```
In [2]: #Run each instuction and see the outpu
        print (2+3)
        a=20
        print (a)
        print type(a)
        a="hello"
        print (len(a))
```

```
5
20
<type 'int'>
5
```

1.4.2 Basic Operations

```
In [3]: x = 3
        print (type(x)) # Prints "<type 'int'>"
        print (x)       # Prints "3"
        print (x + 1 )  # Addition; prints "4"
        print (x - 1 )  # Subtraction; prints "2"
        print (x * 2)   # Multiplication; prints "6"
        print (x ** 2)  # Exponentiation; prints "9"
        x += 1
        print (x)  # Prints "4"
        x *= 2
        print (x ) # Prints "8"
        y = 2.5
        print (type(y)) # Prints "<type 'float'>"
        print (y, y + 1, y * 2, y ** 2)
```

```
<type 'int'>
3
4
2
6
9
4
8
<type 'float'>
2.5 3.5 5.0 6.25
```

1.4.3 Strings have some more useful operations

```
In [4]: s = "hello"
        print (s.capitalize())  # Capitalize a string; prints "Hello"
```

```
        print (s.upper())       # Convert a string to uppercase; prints "HELLO"
        print (s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello
        print (s.center(7))     # Center a string, padding with spaces; prints " hello "
        print (s.replace('l', '(ell)'))  # Replace all instances of one substring with another
                                       # prints "he(ell)(ell)o"
        print( '  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

```
Hello
HELLO
  hello
 hello
he(ell)(ell)o
world
```

```
In [5]: #Concatenating, loop
        a="JKLM"
        for i in a:
            print (i+"ack")
```

```
Jack
Kack
Lack
Mack
```

1.5 Reading input from user

```
In [6]: a=input("Enter a number: ") #Read input from user
        #Condition
        if a>10:
            print ("Number is greater than 10")
        else:
            print ("Number is less than 10")
```

```
Enter a number: 10
Number is less than 10
```

```
In [7]: name=raw_input("Enter name")
        print ("Hello "+name)
```

```
Enter namehii
Hello hii
```

1.5.1 Python: Iterations and Conditions

```
In [8]: a=10
        if a>5:
```

```
    print ("a>5")
elif a>4:
    print "a>4"
else:
    print ("Not matching")

range(10)

for i in range(10):
    print (i)
```

```
a>5
0
1
2
3
4
5
6
7
8
9
```

1.6 Functions

```
In [9]: def my_fun(x,y):
            print (x+y)

        my_fun(10,20)
        my_fun("10","20")
```

```
30
1020
```

1.6.1 Python Data structures

Following are some data structures, which are used in Python. You should be familiar with them in order to use them as appropriate. * **Lists**:Lists are one of the most versatile data structure in Python. A list can simply be defined by writing a list of comma separated values in square brackets. Lists might contain items of different types, but usually the items all have the same type. Python lists are mutable and individual elements of a list can be changed. * **String**: Strings can simply be defined by use of single ('), double (") or triple ("') inverted commas. Strings enclosed in tripe quotes ("') can span over multiple lines and are used frequently in docstrings (Python's way of documenting functions). is used as an escape character. Please note that Python strings are immutable, so you can not change part of strings. * **Tuples**: A tuple is represented by a number of values separated by commas. Tuples are immutable and the output is surrounded by parentheses so that nested tuples are processed correctly. Additionally, even though tuples are immutable, they can hold mutable data if needed. * **Dictionary**: Dictionary is an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}.

1.6.2 Lists

```
In [10]: a=[] #Null list
         a=[1,2,3,4.5,"Hello","50"]#is a list
         print (type(a))
         print (len(a))
         print( a[0])
         print (a[-1])
         print (a[2:5])

         a.append(100) #add new element at the end of list
         print (a)
         a.remove(2)#Remove second element
```

```
<type 'list'>
6
1
50
[3, 4.5, 'Hello']
[1, 2, 3, 4.5, 'Hello', '50', 100]
```

```
In [11]: a.append("append value")
         print (a)
         a.insert(3,"insert value")
         print (a)
```

```
[1, 3, 4.5, 'Hello', '50', 100, 'append value']
[1, 3, 4.5, 'insert value', 'Hello', '50', 100, 'append value']
```

List Compression

```
In [12]: nums = [0, 1, 2, 3, 4]
         even_squares=[]
         for x in nums:
             if x%2==0:
                 even_squares.append(x**2)
         print( even_squares)
```

```
[0, 4, 16]
```

```
In [13]: ## The above code can be compressed as
         even_squares = [x ** 2 for x in nums if x % 2 == 0]
         print (even_squares ) # Prints "[0, 4, 16]"
```

```
[0, 4, 16]
```

Slicing

```
In [14]: a=range(1,25)
         a[2:5]
         a[-1]
         a[1:5:2]
         a[-3:-1]
         a[:]
         a[::]
         a[::2]
         a[::-1]
```

```
Out[14]: [24,
          23,
          22,
          21,
          20,
          19,
          18,
          17,
          16,
          15,
          14,
          13,
          12,
          11,
          10,
          9,
          8,
          7,
          6,
          5,
          4,
          3,
          2,
          1]
```

1.7 Set

```
In [15]: animals = {'cat', 'dog'}
         print ('cat' in animals)   # Check if an element is in a set; prints "True"
         print ('fish' in animals ) # prints "False"
         animals.add('fish')      # Add an element to a set
         print ('fish' in animals)  # Prints "True"
         print (len(animals))       # Number of elements in a set; prints "3"
```

```
animals.add('cat')       # Adding an element that is already in the set does nothing
print (len(animals) )      # Prints "3"
animals.remove('cat')    # Remove an element from a set
print (len(animals) )      # Prints "2"
```

```
True
False
True
3
3
2
```

1.7.1 Dictionary

```
In [21]: pos={}#empty dictionary
         print (type(dict))
         pos["Man"]="Noun"
         pos["Play"]="Verb"
         pos["Good"]="Adjective"
         pos["Apple"]="Noun"
         pos["Cricket"]="Noun"
         pos["Cry"]="Verb"
         print (pos)
         #Print all the nouns in the dictionary
         for w in pos:
             if pos[w]=="Noun":
                 print (w)
```

```
<type 'type'>
{'Play': 'Verb', 'Good': 'Adjective', 'Apple': 'Noun', 'Cry': 'Verb', 'Cricket': 'Noun', 'Man'
Apple
Cricket
Man
```

1.8 Tuple

A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a trivial example:

```
In [22]: d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
         t = (5, 6)        # Create a tuple
         print (type(t))    # Prints "<type 'tuple'>"
         print (d[t] )      # Prints "5"
         print (d[(1, 2)])  # Prints "1"
```

```
<type 'tuple'>
5
1
```

1.9 File

```
In [1]: file=open("newfile.txt","w")
        print (type(file))
        a="This will be written to the file."
        file.write(a)
        file.close()
```

```
<type 'file'>
```

```
In [24]: fread=open("newfile.txt")
         text=fread.read()
         words=text.split()
         print( words)
         print (sorted(words))
```

```
['This', 'will', 'be', 'written', 'to', 'the', 'file.']
['This', 'be', 'file.', 'the', 'to', 'will', 'written']
```

More examples on Python programming and basic data structures can be find here

2 Python for Machine Learning

The following are the important packages for machine learning. * **NumPy** stands for Numerical Python. The most powerful feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran, C and C++ * **SciPy** stands for Scientific Python. SciPy is built on NumPy. It is one of the most useful library for variety of high level science and engineering modules like discrete Fourier transform, Linear Algebra, Optimization and Sparse matrices. * **Matplotlib** for plotting vast variety of graphs, starting from histograms to line plots to heat plots.. You can use Pylab feature in ipython notebook (ipython notebook –pylab = inline) to use these plotting features inline. If you ignore the inline option, then pylab converts ipython environment to an environment, very similar to Matlab. You can also use Latex commands to add math to your plot. * **Pandas** for structured data operations and manipulations. It is extensively used for data munging and preparation. Pandas were added relatively recently to Python and have been instrumental in boosting Python's usage in data scientist community. * **Scikit Learn** for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of effiecient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

2.1 Numpy- Numerical Python

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.

2.1.1 Arrays

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.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

```
In [27]: import numpy as np

         a = np.array([1, 2, 3])  # Create a rank 1 array
         print (type(a))            # Prints "<type '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"
```

```
<type 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

Numpy also provides many functions to create arrays:

```
In [28]: 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)
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.914601   0.16575799]
 [0.8793792  0.37226457]]
```

Array 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:

```
In [29]: 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"
```

```
2
77
```

2.2 Array math

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

In [30]:
```python
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))

# 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))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

You can explore the following Numpy functions from Doumentation * array() * type() * ndim * [start:end:step] * dtype * copy() * view() * reshape() * concatenate() * array_split() * where() * searchsorted()
* sort() * randint() * rand() * shuffle() * permutation() * binomial() * multinomial() * exponential() * add() * prod() * cumprod() * diff() * unique() * lcm() * linspace() * gcd() * sin() * sinh()

2.3 Pandas

One of the most popular data science libraries is Pandas. Developed by data scientists familiar with R and Python, it has grown to support a large community of scientists and analysts. It has many built-in features, such as the ability to read data from many sources, create large dataframes (or matrixes / tables) from these sources and compute aggregate analytics based on what questions you'd like to answer. It has some built-in visualizations which can be used to chart and graph your results as well as several export functions to turn your completed analysis into an Excel Spreadsheet.

2.3.1 Data Frames

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input: * Dict of 1D ndarrays, lists, dicts, or Series * 2-D numpy.ndarray * Structured or record ndarray * A Series * Another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

```
In [35]: import pandas as pd
         import numpy as np

         df1 = pd.DataFrame({ 'A' : range(1,5),
                             'B' : pd.Timestamp('20130102'),
                             'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
                             'D' : np.array([3] * 4,dtype='int32'),
                             'E' : pd.Categorical(["test","train","test","train"]),
                             'F' : 'foo' })
         df1
```

```
Out[35]:    A          B    C  D      E    F
         0  1 2013-01-02  1.0  3   test  foo
         1  2 2013-01-02  1.0  3  train  foo
```

```
2   3 2013-01-02  1.0  3   test  foo
3   4 2013-01-02  1.0  3  train  foo
```

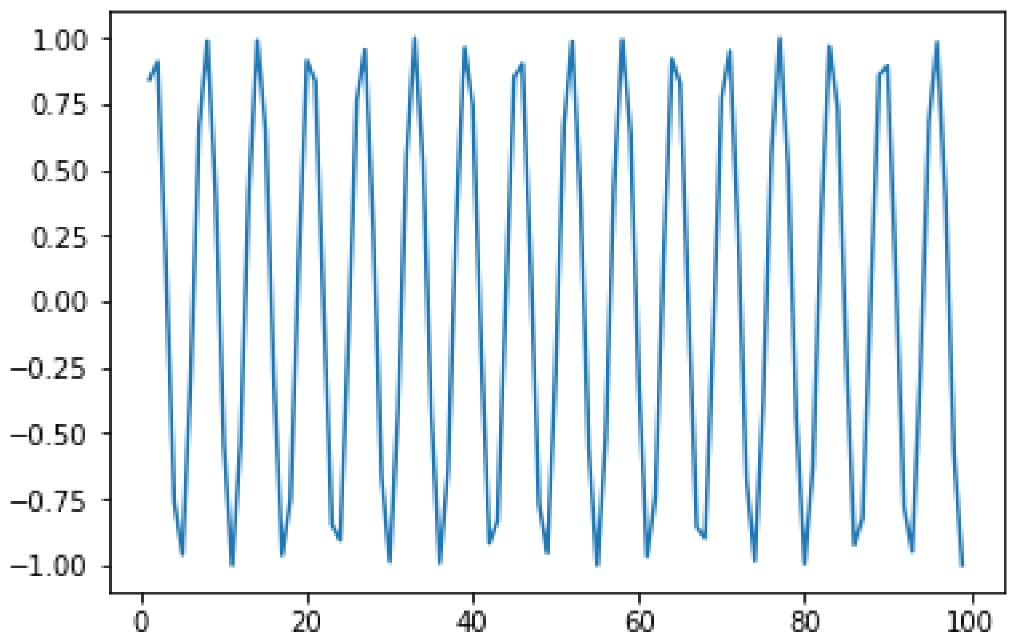
In [36]:
```
df1.dtypes
```

Out[36]:
```
A              int64
B     datetime64[ns]
C            float32
D              int32
E           category
F             object
dtype: object
```

Please try the following Pandas functions from Pandas documentation. * DataFrame() * Series() * Loc[] * read_csv() * to_string() * head() * tail() * info() * dropna() * fillna() * mean() * median() * mode() * to_datetime() * duplicated() * drop_duplicates() * corr() * plot() * show()

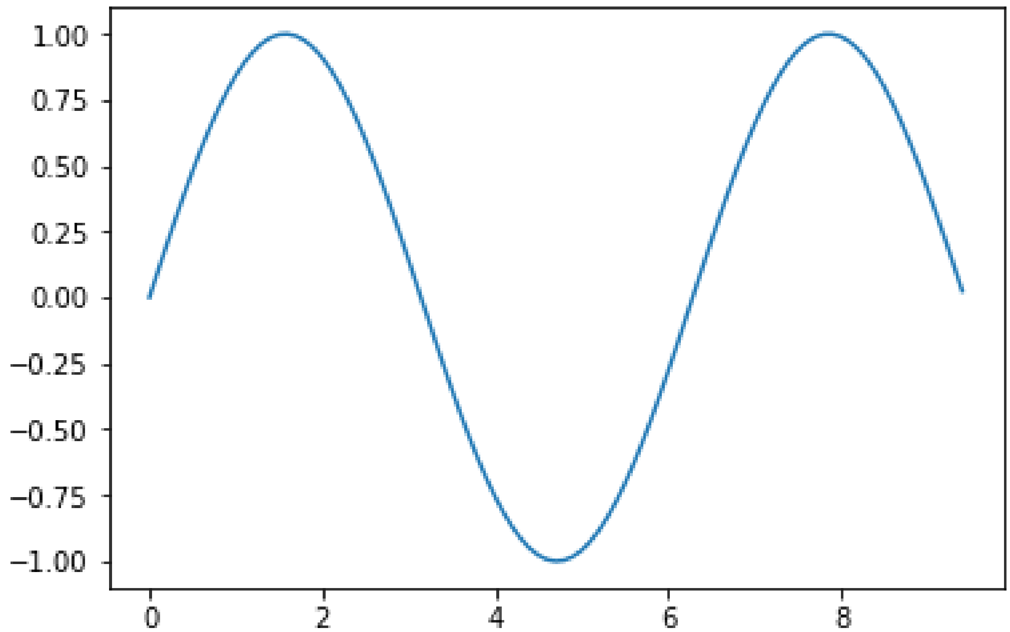
2.4 Matplotlib

In [26]:
```
%matplotlib inline
import math
import matplotlib.pyplot as plt
x=range(1,100)
y=[math.sin(i) for i in x]
plt.plot(x,y)
plt.show()
```



Ploting simple graphs

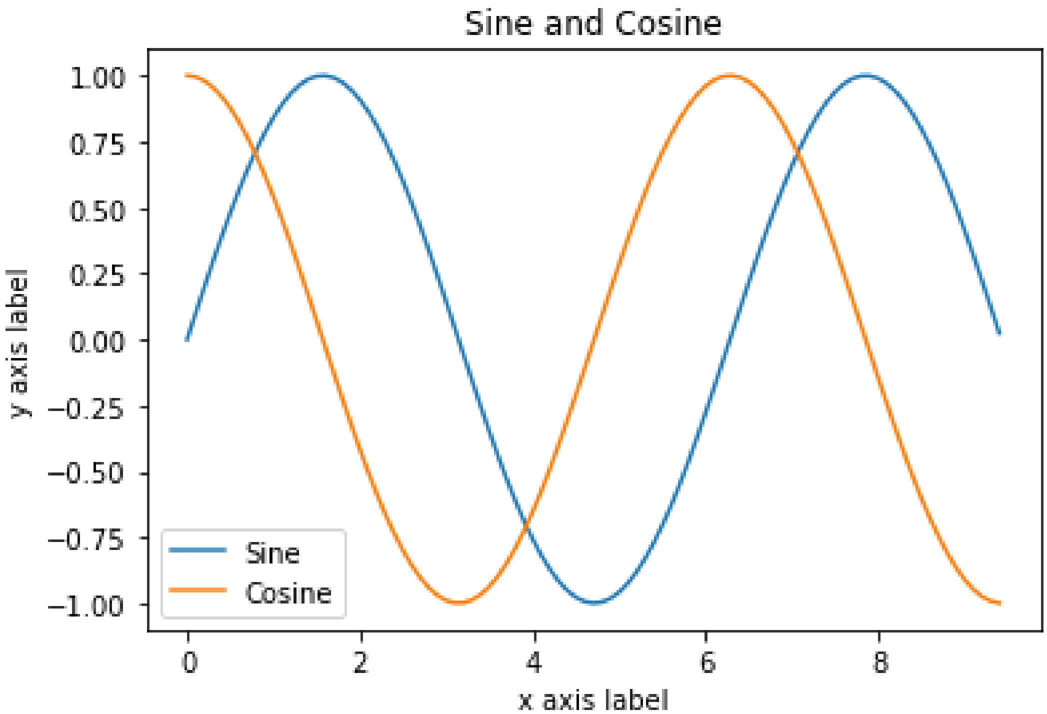
In [31]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



More than one data in same plot

In [32]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
```

```
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

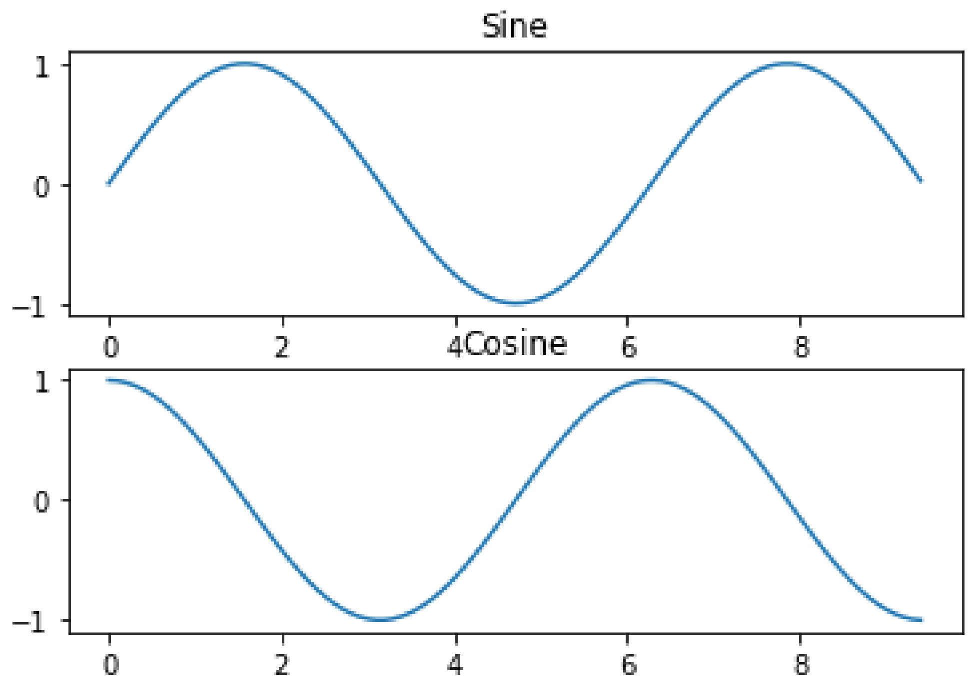


2.4.1 Subplots

You can plot different things in the same figure using the subplot function. Here is an example:

```
In [33]: import numpy as np
         import matplotlib.pyplot as plt

         # Compute the x and y coordinates for points on sine and cosine curves
         x = np.arange(0, 3 * np.pi, 0.1)
         y_sin = np.sin(x)
         y_cos = np.cos(x)

         # Set up a subplot grid that has height 2 and width 1,
         # and set the first such subplot as active.
         plt.subplot(2, 1, 1)

         # Make the first plot
         plt.plot(x, y_sin)
         plt.title('Sine')
```

```
# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



2.5 Exploring Data with Python

2.5.1 Importing Data to Pandas dataframe

The data may stored in sqllite database or csv/tsv format. Here, we will explore a csv file using Pandas dataframe. ** Read data from csv file ** * Note: data is stored in ./salaries/Salaries.csv * This is salary details of employess of different organization of four years

```
In [37]: %matplotlib inline
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from pandas import DataFrame
         from pandas import Series
```

```
In [38]: pd.set_option('display.max_columns', None)
         df=pd.read_csv('./salaries/Salaries.csv',error_bad_lines=False)
```

```
/home/manu/anaconda2/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2723: DtypeWa
  interactivity=interactivity, compiler=compiler, result=result)
```

** View the data ** * len * head * tail

```
In [39]: #df
         print "\nToral Rows=",len(df),"\n\n"
         print df.head(3)
         #print df.tail(3)
```

```
Toral Rows= 148654


   Id    EmployeeName                                         JobTitle BasePay  \
0   1  NATHANIEL FORD  GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY  167411
1   2    GARY JIMENEZ                 CAPTAIN III (POLICE DEPARTMENT)  155966
2   3  ALBERT PARDINI                 CAPTAIN III (POLICE DEPARTMENT)  212739

  OvertimePay OtherPay Benefits   TotalPay  TotalPayBenefits  Year  Notes  \
0           0   400184      NaN  567595.43         567595.43  2011    NaN
1      245132   137811      NaN  538909.28         538909.28  2011    NaN
2      106088  16452.6      NaN  335279.91         335279.91  2011    NaN

          Agency Status
0  San Francisco    NaN
1  San Francisco    NaN
2  San Francisco    NaN
```

** To get a description of data set**

```
In [40]: df.describe()
```

```
Out[40]:                   Id       TotalPay  TotalPayBenefits           Year  Notes
         count  148654.000000  148654.000000     148654.000000  148654.000000    0.0
         mean    74327.500000   74768.321972      93692.554811    2012.522643    NaN
         std     42912.857795   50517.005274      62793.533483       1.117538    NaN
         min         1.000000    -618.130000       -618.130000    2011.000000    NaN
         25%     37164.250000   36168.995000      44065.650000    2012.000000    NaN
         50%     74327.500000   71426.610000      92404.090000    2013.000000    NaN
         75%    111490.750000  105839.135000     132876.450000    2014.000000    NaN
         max    148654.000000  567595.430000     567595.430000    2014.000000    NaN
```

```
In [41]: df.dtypes
```

```
Out[41]: Id                    int64
         EmployeeName         object
```

```
JobTitle             object
BasePay              object
OvertimePay          object
OtherPay             object
Benefits             object
TotalPay            float64
TotalPayBenefits    float64
Year                  int64
Notes               float64
Agency               object
Status               object
dtype: object
```

2.5.2 Selecting Data

```
In [42]: df['Year']
```

```
Out[42]: 0         2011
         1         2011
         2         2011
         3         2011
         4         2011
         5         2011
         6         2011
         7         2011
         8         2011
         9         2011
         10        2011
         11        2011
         12        2011
         13        2011
         14        2011
         15        2011
         16        2011
         17        2011
         18        2011
         19        2011
         20        2011
         21        2011
         22        2011
         23        2011
         24        2011
         25        2011
         26        2011
         27        2011
         28        2011
         29        2011
                   ...
```

```
148624    2014
148625    2014
148626    2014
148627    2014
148628    2014
148629    2014
148630    2014
148631    2014
148632    2014
148633    2014
148634    2014
148635    2014
148636    2014
148637    2014
148638    2014
148639    2014
148640    2014
148641    2014
148642    2014
148643    2014
148644    2014
148645    2014
148646    2014
148647    2014
148648    2014
148649    2014
148650    2014
148651    2014
148652    2014
148653    2014
Name: Year, Length: 148654, dtype: int64
```

In [43]:
```
set(df['Year']) # Unique items
```

Out[43]:
```
{2011, 2012, 2013, 2014}
```

** Project the data with some conditions** Eg: Show the details of employees with Total pay above 500000.

In [44]:
```
df[df.TotalPay>500000]
```

Out[44]:
```
   Id     EmployeeName                                        JobTitle BasePay  \
0   1  NATHANIEL FORD  GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY  167411
1   2    GARY JIMENEZ                 CAPTAIN III (POLICE DEPARTMENT)  155966

  OvertimePay OtherPay Benefits   TotalPay  TotalPayBenefits  Year  Notes  \
0           0   400184      NaN  567595.43         567595.43  2011    NaN
1      245132   137811      NaN  538909.28         538909.28  2011    NaN
```

```
           Agency Status
0  San Francisco    NaN
1  San Francisco    NaN
```

** Gruop by some column **

```
In [45]: df.groupby('Year').sum()
```

```
Out[45]:               Id      TotalPay  TotalPayBenefits  Notes
         Year
         2011   653754720  2.594195e+09      2.594195e+09    0.0
         2012  2005309555  2.724848e+09      3.696940e+09    0.0
         2013  3449541971  2.918656e+09      3.814772e+09    0.0
         2014  4940473939  2.876911e+09      3.821866e+09    0.0
```

** Project selected columns ** Create a new data frame with specific coulmns

```
In [46]: df.loc[:,['EmployeeName','BasePay']]
```

```
Out[46]:                 EmployeeName       BasePay
         0             NATHANIEL FORD        167411
         1               GARY JIMENEZ        155966
         2             ALBERT PARDINI        212739
         3          CHRISTOPHER CHONG         77916
         4            PATRICK GARDNER        134402
         5             DAVID SULLIVAN        118602
         6                  ALSON LEE         92492
         7              DAVID KUSHNER        256577
         8             MICHAEL MORRIS        176933
         9         JOANNE HAYES-WHITE        285262
         10             ARTHUR KENNEY        194999
         11          PATRICIA JACKSON         99722
         12         EDWARD HARRINGTON        294580
         13               JOHN MARTIN        271329
         14            DAVID FRANKLIN        174873
         15           RICHARD CORRIEA        198778
         16                  AMY HART        268605
         17           SEBASTIAN WONG         140547
         18                MARTY ROSS        168693
         19             ELLEN MOFFATT        257511
         20                VENUS AZAR        257510
         21              JUDY MELINEK        257510
         22             GEORGE GARCIA        140547
         23             VICTOR WYRSCH        168693
         24           JOSEPH DRISCOLL        140547
         25              GREGORY SUHR        256470
         26               JOHN HANLEY       92080.8
         27            RAYMOND GUZMAN        168693
         28            DENISE SCHMITT        261718
```

```
29              MONICA FIELDS        246226
...                       ...           ...
148624      Lorraine Rosenthal         0.00
148625         Renato C Gurion         0.00
148626           Paulet Gaines         0.00
148627        Brett A Lundberg         0.00
148628          Mark W Mcclure         0.00
148629       Elizabeth Iniguez         0.00
148630           Randy J Keys          0.00
148631        Andre M Johnson          0.00
148632  Sharon D Owens-Webster         0.00
148633        Edward Ferdinand         0.00
148634          David M Turner         0.00
148635     James S Kibblewhite         0.00
148636          Andrew J Enzi          0.00
148637        Kadeshra D Green         0.00
148638     Lennard B Hutchinson        0.00
148639       Richard A Talbert         0.00
148640      Charlene D Mccully         0.00
148641   Raphael Marquis Goins         0.00
148642       Dominic C Marquez         0.00
148643              Kim Brewer         0.00
148644           Randy D Winn          0.00
148645        Carolyn A Wilson         0.00
148646            Not provided  Not Provided
148647          Joann Anderson         0.00
148648             Leon Walker         0.00
148649           Roy I Tillery         0.00
148650            Not provided  Not Provided
148651            Not provided  Not Provided
148652            Not provided  Not Provided
148653               Joe Lopez         0.00

[148654 rows x 2 columns]
```

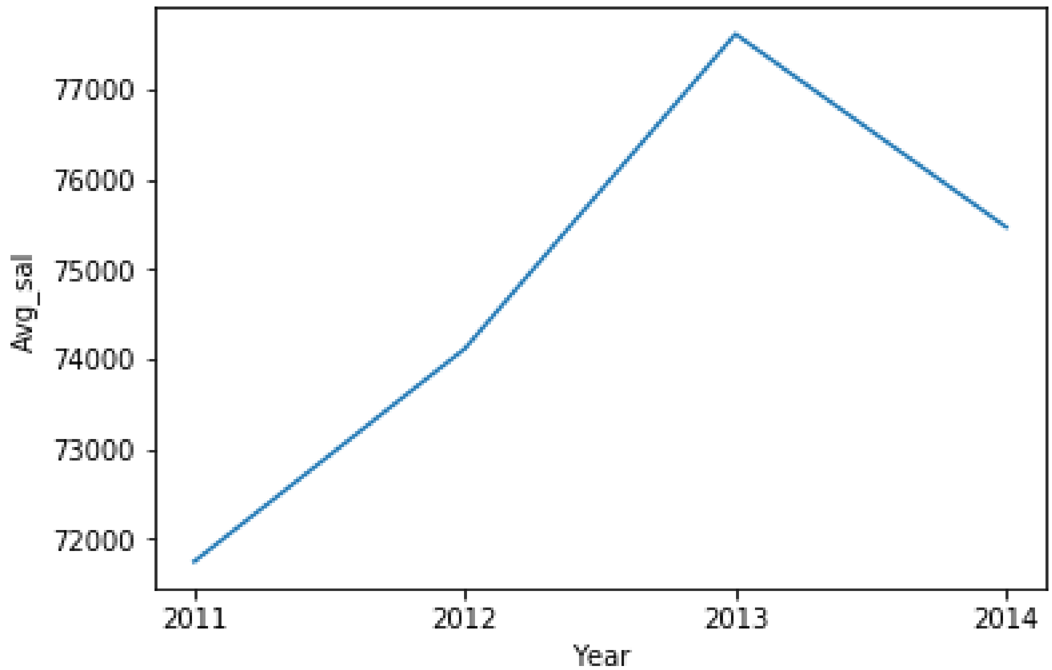
In [47]:
```python
#Use slicing
df.iloc[:3,1:3]
```

Out[47]:
```
      EmployeeName                                         JobTitle
0  NATHANIEL FORD  GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY
1    GARY JIMENEZ                 CAPTAIN III (POLICE DEPARTMENT)
2  ALBERT PARDINI                 CAPTAIN III (POLICE DEPARTMENT)
```

** Plot the Data** Plot the average salary *vs.* year

In [48]:
```python
avg_sal=df.groupby('Year').mean()['TotalPay']
year=range(2011,2015)

my_xticks=range(2011,2015)
```

```
plt.xticks(year,my_xticks)
plt.plot(year,avg_sal)
plt.xlabel('Year')
plt.ylabel('Avg_sal')
plt.show()
```



3 Reference

- Wes McKinney, **Python for Data Analysis** . O'Reilly Media.
- A Complete Tutorial to Learn Data Science with Python from Scratch, Analytic Vidya
- Anaconda and Jupyter installation
- PANDAS: Python Data Analysis Library
- Numpy Tutorial
- Han, Datamining Concepts and Techniques, MK Publishers, 2000