Python - Data Science

Data is the new Oil. This statement shows how every modern IT system is driven by capturing, storing and analysing data for various needs. Be it about making decision for business, forecasting weather, studying protein structures in biology or designing a marketing campaign. All of these scenarios involve a multidisciplinary approach of using mathematical models, statistics, graphs, databases and of course the business or scientific logic behind the data analysis. So we need a programming language which can cater to all these diverse needs of data science. Python shines bright as one such language as it has numerous libraries and built in features which makes it easy to tackle the needs of Data science.

### Recommendation systems

As online shopping becomes more prevalent, the e-commerce platforms are able to capture users shopping preferences as well as the performance of various products in the market. This leads to creation of recommendation systems which create models predicting the shoppers needs and show the products the shopper is most likely to buy.

### Financial Risk management

The financial risk involving loans and credits are better analysed by using the customers past spend habits, past defaults, other financial commitments and many socio-economic indicators. These data is gathered from various sources in different formats. Organising them together and getting insight into customers profile needs the help of Data science. The outcome is minimizing loss for the financial organization by avoiding bad debt.

### Improvement in Health Care services

The health care industry deals with a variety of data which can be classified into technical data, financial data, patient information, drug information and legal rules. All this data need to be analysed in a coordinated manner to produce insights that will save cost both for the health care provider and care receiver while remaining legally compliant.

### Computer Vision

The advancement in recognizing an image by a computer involves processing large sets of image data from multiple objects of same category. For example, Face recognition. These data sets are modelled, and algorithms are created to apply the model to newer images to get a satisfactory result. Processing of these huge data sets and creation of models need various tools used in Data science.

### Efficient Management of Energy

As the demand for energy consumption soars, the energy producing companies need to manage the various phases of the energy production and distribution more efficiently. This involves optimizing the production methods, the storage and distribution mechanisms as well as studying the customers consumption patterns. Linking the data from all these sources and deriving insight seems a daunting task. This is made easier by using the tools of data science.

## Python in Data Science

The programming requirements of data science demands a very versatile yet flexible language which is simple to write the code but can handle highly complex mathematical processing. Python is most suited for such requirements as it has already established itself both as a language for general computing as well as scientific computing. More over it is being continuously upgraded in form of new addition to its plethora of libraries aimed at different programming requirements. Below we will discuss such features of python which makes it the preferred language for data science.

* A simple and easy to learn language which achieves result in fewer lines of code than other similar languages like R. Its simplicity also makes it robust to handle complex scenarios with minimal code and much less confusion on the general flow of the program.
* It is cross platform, so the same code works in multiple environments without needing any change. That makes it perfect to be used in a multi-environment setup easily.
* It executes faster than other similar languages used for data analysis like R and MATLAB.
* Its excellent memory management capability, especially garbage collection makes it versatile in gracefully managing very large volume of data transformation, slicing, dicing and visualization.
* Most importantly Python has got a very large collection of libraries which serve as special purpose analysis tools. For example – the NumPy package deals with scientific computing and its array needs much less memory than the conventional python list for managing numeric data. And the number of such packages is continuously growing.
* Python has packages which can directly use the code from other languages like Java or C. This helps in optimizing the code performance by using existing code of other languages, whenever it gives a better result.

In the subsequent chapters we will see how we can leverage these features of python to accomplish all the tasks needed in the different areas of Data Science.

# Python - Data Science Environment Setup

To successfully create and run the example code in this tutorial we will need an environment set up which will have both general-purpose python as well as the special packages required for Data science. We will first look as installing the general-purpose python which can be python 2 or python 3. But we will prefer python 2 for this tutorial mainly because of its maturity and wider support of external packages.

## Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python <https://www.python.org/>

You can download Python documentation from <https://www.python.org/doc/>. The documentation is available in HTML, PDF, and PostScript formats.

## Installing Python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

Here is a quick overview of installing Python on various platforms −

### Unix and Linux Installation

Here are the simple steps to install Python on Unix/Linux machine.

* Open a Web browser and go to <https://www.python.org/downloads/>.
* Follow the link to download zipped source code available for Unix/Linux.
* Download and extract files.
* Editing the *Modules/Setup* file if you want to customize some options.
* run ./configure script
* make
* make install

This installs Python at standard location */usr/local/bin* and its libraries at */usr/local/lib/pythonXX* where XX is the version of Python.

### Windows Installation

Here are the steps to install Python on Windows machine.

* Open a Web browser and go to <https://www.python.org/downloads/>.
* Follow the link for the Windows installer *python-XYZ.msi*file where XYZ is the version you need to install.
* To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
* Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

**Installation Step:-**

1 python -m pip install numpy

2 python -m pip install scipy

3 python -m pip install matplotlib

## Installing SciPy Pack

The best way to enable the required packs is to use an installable binary package specific to your operating system. These binaries contain full SciPy stack (inclusive of NumPy, SciPy, matplotlib, IPython, SymPy and nose packages along with core Python).

## Windows

Anaconda (from [www.continuum.io](https://www.continuum.io/)) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.

Canopy ([www.enthought.com/products/canopy/](https://www.enthought.com/products/canopy/)) is available as free as well as commercial distribution with full SciPy stack for Windows, Linux and Mac.

Python (x,y): It is a free Python distribution with SciPy stack and Spyder IDE for Windows OS. (Downloadable from [www.python-xy.github.io/](https://python-xy.github.io/))

# Python - Pandas

Pandas is an open-source Python Library used for high-performance data manipulation and data analysis using its powerful data structures. Python with pandas is in use in a variety of academic and commercial domains, including Finance, Economics, Statistics, Advertising, Web Analytics, and more. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, organize, manipulate, model, and analyse the data.

Below are the some of the important features of Pandas which is used specifically for Data processing and Data analysis work.

## Key Features of Pandas

* Fast and efficient DataFrame object with default and customized indexing.
* Tools for loading data into in-memory data objects from different file formats.
* Data alignment and integrated handling of missing data.
* Reshaping and pivoting of date sets.
* Label-based slicing, indexing and subsetting of large data sets.
* Columns from a data structure can be deleted or inserted.
* Group by data for aggregation and transformations.
* High performance merging and joining of data.
* Time Series functionality.

Pandas deals with the following three data structures −

* Series
* DataFrame

These data structures are built on top of Numpy array, making them fast and efficient.

## Dimension & Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Dimensions** | **Description** |
| Series | 1 | 1D labeled homogeneous array, size-immutable. |
| Data Frames | 2 | General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns. |

DataFrame is widely used and it is the most important data structures.

## Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 23 | 56 | 17 | 52 | 61 | 73 | 90 | 26 | 72 |

### Key Points of Series

* Homogeneous data
* Size Immutable
* Values of Data Mutable

## DataFrame

DataFrame is a two-dimensional array with heterogeneous data. For example,

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Age** | **Gender** | **Rating** |
| Steve | 32 | Male | 3.45 |
| Lia | 28 | Female | 4.6 |
| Vin | 45 | Male | 3.9 |
| Katie | 38 | Female | 2.78 |

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

## Data Type of Columns

The data types of the four columns are as follows −

|  |  |
| --- | --- |
| **Column** | **Type** |
| Name | String |
| Age | Integer |
| Gender | String |
| Rating | Float |

### Key Points of Data Frame

* Heterogeneous data
* Size Mutable
* Data Mutable

We will see lots of examples on using pandas library of python in Data science work in the next chapters.

# Python - Numpy

NumPy is a Python package which stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

## Operations using NumPy

Using NumPy, a developer can perform the following operations −

* Mathematical and logical operations on arrays.
* Fourier transforms and routines for shape manipulation.
* Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

## NumPy – A Replacement for MatLab

NumPy is often used along with packages like **SciPy** (Scientific Python) and **Mat−plotlib** (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.

It is open source, which is an added advantage of NumPy.

## ndarray Object

The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index. Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**). Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.

We will see lots of examples on using NumPy library of python in Data science work in the next chapters.

# Python - SciPy

The SciPy library of Python is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge. NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

## SciPy Sub-packages

SciPy is organized into sub-packages covering different scientific computing domains. These are summarized in the following table

## Data Structure

The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for Linear Algebra, Fourier Transforms and Random Number Generation, but not with the generality of the equivalent functions in SciPy.

# Python - Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

Conventionally, the package is imported into the Python script by adding the following statement

from matplotlib import pyplot as plt

## Matplotlib Example

The following script produces the **sine wave plot** using matplotlib.

### Example

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)

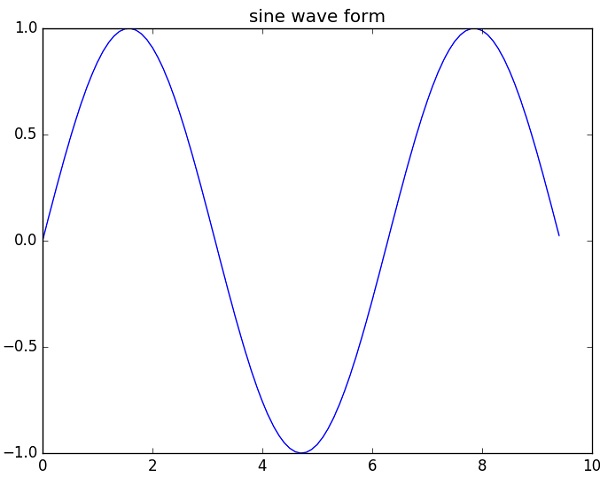
plt.title("sine wave form")

# Plot the points using matplotlib

plt.plot(x, y)

plt.show()

Its **output** is as follows −



We will see lots of examples on using Matplotlib library of python in Data science work in the next chapters.

Python handles data of various formats mainly through the two libraries, Pandas and Numpy. We have already seen the important features of these two libraries in the previous chapters. In this chapter we will see some basic examples from each of the libraries on how to operate on data.

## Data Operations in Numpy

The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index. An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows −

numpy.array

Following are some examples on Numpy Data handling.

### Example 1

# more than one dimensions

import numpy as np

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

print a

The output is as follows −

[[1, 2]

[3, 4]]

### Example 2

# minimum dimensions

import numpy as np

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

print a

The output is as follows −

[[1, 2, 3, 4, 5]]

## Example 3

# dtype parameter

import numpy as np

a = np.array([1, 2, 3], dtype = complex)

print a

The output is as follows −

[ 1.+0.j, 2.+0.j, 3.+0.j]

## Data Operations in Pandas

Pandas handles data through **Series**,**Data Frame**, and **Panel**. We will see some examples from each of these.

## 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)

### Example

Here we create a series from a Numpy Array.

#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

## Pandas DataFrame

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. A pandas DataFrame can be created using the following constructor −

pandas.DataFrame( data, index, columns, dtype, copy)

Let us now create an indexed DataFrame using arrays.

import pandas as pd

data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}

df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])

print df

Its **output** is as follows −

Age Name

rank1 28 Tom

rank2 34 Jack

rank3 29 Steve

rank4 42 Ricky

## Pandas Panel

A **panel** is a 3D container of data. The term **Panel data** is derived from econometrics and is partially responsible for the name pandas − **pan(el)-da(ta)**-s.

A Panel can be created using the following constructor −

pandas.Panel(data, items, major\_axis, minor\_axis, dtype, copy)

In the below example we create a panel from dict of DataFrame Objects

#creating an empty panel

import pandas as pd

import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),

'Item2' : pd.DataFrame(np.random.randn(4, 2))}

p = pd.Panel(data)

print p

Its **output** is as follows −

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 5 (minor\_axis)

Items axis: 0 to 1

Major\_axis axis: 0 to 3

Minor\_axis axis: 0 to 4

# Python - Data Cleansing

Missing data is always a problem in real life scenarios. Areas like machine learning and data mining face severe issues in the accuracy of their model predictions because of poor quality of data caused by missing values. In these areas, missing value treatment is a major point of focus to make their models more accurate and valid.

## When and Why Is Data Missed?

Let us consider an online survey for a product. Many a times, people do not share all the information related to them. Few people share their experience, but not how long they are using the product; few people share how long they are using the product, their experience but not their contact information. Thus, in some or the other way a part of data is always missing, and this is very common in real time.

Let us now see how we can handle missing values (say NA or NaN) using Pandas.

# import the pandas library

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df

Its **output** is as follows −

one two three

a 0.077988 0.476149 0.965836

b NaN NaN NaN

c -0.390208 -0.551605 -2.301950

d NaN NaN NaN

e -2.000303 -0.788201 1.510072

f -0.930230 -0.670473 1.146615

g NaN NaN NaN

h 0.085100 0.532791 0.887415

Using reindexing, we have created a DataFrame with missing values. In the output, **NaN** means **Not a Number.**

### Check for Missing Values

To make detecting missing values easier (and across different array dtypes), Pandas provides the **isnull()** and **notnull()**functions, which are also methods on Series and DataFrame objects −

### Example

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df['one'].isnull()

Its **output** is as follows −

a False

b True

c False

d True

e False

f False

g True

h False

Name: one, dtype: bool

## Cleaning / Filling Missing Data

Pandas provides various methods for cleaning the missing values. The fillna function can “fill in” NA values with non-null data in a couple of ways, which we have illustrated in the following sections.

## Replace NaN with a Scalar Value

The following program shows how you can replace "NaN" with "0".

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',

'two', 'three'])

df = df.reindex(['a', 'b', 'c'])

print df

print ("NaN replaced with '0':")

print df.fillna(0)

Its **output** is as follows −

one two three

a -0.576991 -0.741695 0.553172

b NaN NaN NaN

c 0.744328 -1.735166 1.749580

NaN replaced with '0':

one two three

a -0.576991 -0.741695 0.553172

b 0.000000 0.000000 0.000000

c 0.744328 -1.735166 1.749580

Here, we are filling with value zero; instead we can also fill with any other value.

## Fill NA Forward and Backward

Using the concepts of filling discussed in the ReIndexing Chapter we will fill the missing values.

|  |  |
| --- | --- |
| **Method** | **Action** |
| pad/fill | Fill methods Forward |
| bfill/backfill | Fill methods Backward |

### Example

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df.fillna(method='pad')

Its **output** is as follows −

one two three

a 0.077988 0.476149 0.965836

b 0.077988 0.476149 0.965836

c -0.390208 -0.551605 -2.301950

d -0.390208 -0.551605 -2.301950

e -2.000303 -0.788201 1.510072

f -0.930230 -0.670473 1.146615

g -0.930230 -0.670473 1.146615

h 0.085100 0.532791 0.887415

## Drop Missing Values

If you want to simply exclude the missing values, then use the **dropna** function along with the **axis** argument. By default, axis=0, i.e., along row, which means that if any value within a row is NA then the whole row is excluded.

### Example

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df.dropna()

Its **output** is as follows −

one two three

a 0.077988 0.476149 0.965836

c -0.390208 -0.551605 -2.301950

e -2.000303 -0.788201 1.510072

f -0.930230 -0.670473 1.146615

h 0.085100 0.532791 0.887415

## Replace Missing (or) Generic Values

Many times, we have to replace a generic value with some specific value. We can achieve this by applying the replace method.

Replacing NA with a scalar value is equivalent behavior of the **fillna()** function.

### Example

import pandas as pd

import numpy as np

df = pd.DataFrame({'one':[10,20,30,40,50,2000],

'two':[1000,0,30,40,50,60]})

print df.replace({1000:10,2000:60})

Its **output** is as follows −

one two

0 10 10

1 20 0

2 30 30

3 40 40

4 50 50

5 60 60

# Python - Processing JSON Data

JSON file stores data as text in human-readable format. JSON stands for JavaScript Object Notation. Pandas can read JSON files using the **read\_json** function.

## Input Data

Create a JSON file by copying the below data into a text editor like notepad. Save the file with **.json** extension and choosing the file type as **all files(\*.\*)**.

{

"ID":["1","2","3","4","5","6","7","8" ],

"Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru" ]

"Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5" ],

"StartDate":[ "1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013",

"7/30/2013","6/17/2014"],

"Dept":[ "IT","Operations","IT","HR","Finance","IT","Operations","Finance"]

}

## Read the JSON File

The **read\_json** function of the pandas library can be used to read the JSON file into a pandas DataFrame.

import pandas as pd

data = pd.read\_json('path/input.json')

print (data)

When we execute the above code, it produces the following result.

Dept ID Name Salary StartDate

0 IT 1 Rick 623.30 1/1/2012

1 Operations 2 Dan 515.20 9/23/2013

2 IT 3 Tusar 611.00 11/15/2014

3 HR 4 Ryan 729.00 5/11/2014

4 Finance 5 Gary 843.25 3/27/2015

5 IT 6 Rasmi 578.00 5/21/2013

6 Operations 7 Pranab 632.80 7/30/2013

7 Finance 8 Guru 722.50 6/17/2014

## Reading Specific Columns and Rows

Similar to what we have already seen in the previous chapter to read the CSV file, the **read\_json** function of the pandas library can also be used to read some specific columns and specific rows after the JSON file is read to a DataFrame. We use the multi-axes indexing method called **.loc()** for this purpose. We choose to display the Salary and Name column for some of the rows.

import pandas as pd

data = pd.read\_json('path/input.xlsx')

# Use the multi-axes indexing funtion

print (data.loc[[1,3,5],['salary','name']])

When we execute the above code, it produces the following result.

salary name

1 515.2 Dan

3 729.0 Ryan

5 578.0 Rasmi

## Reading JSON file as Records

We can also apply the **to\_json** function along with parameters to read the JSON file content into individual records.

import pandas as pd

data = pd.read\_json('path/input.xlsx')

print(data.to\_json(orient='records', lines=True))

When we execute the above code, it produces the following result.

{"Dept":"IT","ID":1,"Name":"Rick","Salary":623.3,"StartDate":"1\/1\/2012"}

{"Dept":"Operations","ID":2,"Name":"Dan","Salary":515.2,"StartDate":"9\/23\/2013"}

{"Dept":"IT","ID":3,"Name":"Tusar","Salary":611.0,"StartDate":"11\/15\/2014"}

{"Dept":"HR","ID":4,"Name":"Ryan","Salary":729.0,"StartDate":"5\/11\/2014"}

{"Dept":"Finance","ID":5,"Name":"Gary","Salary":843.25,"StartDate":"3\/27\/2015"}

{"Dept":"IT","ID":6,"Name":"Rasmi","Salary":578.0,"StartDate":"5\/21\/2013"}

{"Dept":"Operations","ID":7,"Name":"Pranab","Salary":632.8,"StartDate":"7\/30\/2013"}

{"Dept":"Finance","ID":8,"Name":"Guru","Salary":722.5,"StartDate":"6\/1