Python for Data Analysis

Author, Mayank Tripathi

A Data Science Foundation White Paper

April 2020

---

www.datascience.foundation

Hello Friends,
Nice to see you once again. Writing this article on using Python for Data Analysis.

Before we explore Python for Data Analysis, we should know what Data Analysis and Python are, why we should use Python and what are other options we have. Don't worry we will see and explore these options next.

So first things first, an understanding of Data Analysis; what options are open to us when performing Data Analytics, why Python is useful and getting started with Python.

If you need to know what Data Analysis is, please refer to my previous article: https://datascience.foundation/datatalk/understanding-analytics before continuing here.But in-short we can say that the Data analysis involves a broad set of activities to **clean**, **process** and **transform** a data set in order to learn from it.

Before selecting Data Analysis tools, one has to consider the following:

1. How we should think about our data and what we are looking to learn?
2. Think about categorization and transforming a data set into one that is easier to analyze
3. Show how to visualize the results of your data exploration.

Among various Data Analysis resources or tools or languages name a few **R; Python; Scala; Java; C; C#; SQL; PHP; Tableau; GO; Ruby; etc.**

Among these I feel Python & R have such a large collection of libraries, they are bound to meet most of yours need for Data Analysis when compared to the others. Just to mention that Tableau is mainly for Data Visualization; SQL is mainly Data Cleaning; the others I am not so familiar with but these are also specialized in one or two areas which are part of Data Analysis, whereas Python provides us with everything, and we just have to import the required libraries. There are many other benefits as well, which we will discuss in another article.

In short, Python is a valuable resource / tool / language needed for Data Analysis.

The good thing about Python is that it is Open Source; and is easy to install. On top of this, if we want to get rid of the local installation and work with Python on the cloud then that is also possible. Please refer to article https://datascience.foundation/datatalk/setting-up-a-python-jupyter-notebook-online-working-with-python-on-the-cloud. This is where I have explained how to use Python without the hassle of installing software.

I know many of you are struggling with how to start Machine Learning / Data Science / Python etc so please refer to https://datascience.foundation/articles/mayanktripathi@dsfdatatalk, I hope this will help you.

Python

To start with Python, below are the important things we need to focus on.

- Learn Python Fundamentals

- Learn Python Libraries which are useful for Machine Learning / Data Science / Data Analysis
- Practice Mini Python Data Analysis projects
- Apply Advanced Techniques

Learn Python Fundamentals →

The best resource for learning Python is python docs https://docs.python.org/3/tutorial/index.html It has full details of each and every object or fundamental of Python. One may not be interested in learning full python, so we will skip the python docs for now, and will directly jump onto the things we need to get started with Data Analysis.

Python Data Structures

Below are some data structures, which are used in Python while performing Data Analysis. We should be familiar with these in order to use them as required.

Lists –

- Lists are just like the arrays, declared in other languages.
- Lists might contain items of different types, which makes it a most powerful tool in Python.
- A single list may contain DataTypes like Integers, Strings, as well as Objects.
- Lists are mutable, and hence, they can be altered even after their creation, and inpidual elements of a list can be changed.
- Lists are one of the most versatile data structures in Python.
- A list can simply be defined by writing a list of comma separated values in square brackets.
- Indexing of a list is done with 0 being the first index.
- Lists can be nested with other lists.
- Lists in Python can be created by just placing the sequence inside the square brackets[].

Here is a quick example to define a list and then access it:
Note: I am using Jupyter Notebook. I will share the entire Jupyter notebook at the end, so that you can have the code handy and refer back to it in future.

Creating a List is easy. Below we have created an empty List with a single line of code as **empty_List** = [ ] . Note empty_List is just a variable name given to the list. Any name could be used, but a recommendation is to use a meaningful name. This will make list management easier for you later on.

```
# Creating an Empty List
empty_List = []
print("Creating an Empty List.")
print(empty_List)
```

```
Creating an Empty List.
[]
```

Next will see a list with some values in it. We can have either one value or multiple values in the list.

We can have String; Numerical / Integer ; Float; boolean etc values in the List.

```
# Creating an List with value in it.
mix_List = ['Python', 'Data', 'Structure', "Lists", "123", 456, 78.56]
print("Lists with the use of String ; Number; Float ")
print(mix_List)
```

```
Lists with the use of String ; Number; Float
['Python', 'Data', 'Structure', 'Lists', '123', 456, 78.56]
```

Next will create a List-of-List or nested list.
If we look carefully, then the nested_List is having 3 different List in it.

```
# Creating an List within List (nested list)
nested_List = [['Python', 'Data', "Structure"] , ['Lists'], [123, '456', 78.6]]
print("Nested List or List of List ")
print(nested_List)
```

```
Nested List or List of List
[['Python', 'Data', 'Structure'], ['Lists'], [123, '456', 78.6]]
```

Now we know how to create a list, let's have a look on how we can add more elements to a list.
Elements can be added to the List by using built-in append() function.
Only one element at a time can be added to the list by using append() method.

To practise this, first will create an empty list, and then will add 3 elements into the list one-by-one.

```
# Creating an Empty List
empty_List = []
print("Creating an Empty List.")
print(empty_List)

# Add a single elements in the List
empty_List.append("Python Data Structure")
empty_List.append('Learning about LIST')
empty_List.append(123)
print("*"*50)
print('List After adding 3 elements one-by-one')
print(empty_List)
```

```
Creating an Empty List.
[]
**************************************************
List After adding 3 elements one-by-one
['Python Data Structure', 'Learning about LIST', 123]
```

It's not necessary to have an empty list, we can add the element to any list which has some elements into it. Check below example.

```
# Creating an List with value in it.
mix_List = ['Python', 'Data', 'Structure', "Lists", "123", 456, 78.56]
print("Lists with the use of String ; Number; Float ")
print(mix_List)

# Add a single elements in the List
mix_List.append("Python Data Structure")
mix_List.append('Learning about LIST')
mix_List.append(123)
print("*"*50)
print('List After adding 3 elements one-by-one')
print(mix_List)
```

```
Lists with the use of String ; Number; Float
['Python', 'Data', 'Structure', 'Lists', '123', 456, 78.56]
**************************************************
List After adding 3 elements one-by-one
['Python', 'Data', 'Structure', 'Lists', '123', 456, 78.56, 'Python Data Structure', 'Learning about LIST', 123]
```

If we need to add multiple elements, we will use a loop and iterate over those multiple elements and add the element using the **append()** method.

Here I am using the **range()** function to generate 4 values in sequence, and iterating the loop over it and adding each element to the list.

```
# Creating an Empty List
empty_List = []
print("Creating an Empty List.")
print(empty_List)

# Add a multiple elements in the List using loop.
for i in range(1, 5):
    empty_List.append(i)

print("*"*50)
print('List After adding 4 elements one-by-one using loop')
print(empty_List)
```

```
Creating an Empty List.
[]
**************************************************
List After adding 4 elements one-by-one using loop
[1, 2, 3, 4]
```

After creating a list and adding new elements to the existing list, it's time to access the elements from the List.

In order to access the list items refer to the index number.
Use the index operator [ ] to access an item in a list.
The index must be an integer.
Index will start from 0.
Nested list is accessed using nested indexing.

```
# Creating an List with value in it.
mix_List = ['Python', 'Data', 'Structure', "Lists", "123", 456, 78.56, True]
print("Lists with the use of String ; Number; Float; Boolean ")
print(mix_List)

print("*"*50)

print("Accessing a element from the list")
print(mix_List[0]) # To access the Very First element
print(mix_List[2]) # To access the 3rd element.
print(mix_List[-1]) # To access the last element, when we do not know the count of elements in the list.
print(mix_List[7]) # To access the last element
```

```
Lists with the use of String ; Number; Float; Boolean
['Python', 'Data', 'Structure', 'Lists', '123', 456, 78.56, True]
**************************************************
Accessing a element from the list
Python
Structure
True
True
```

Please make a note that the Indexing in Python List starts with zero (0). So to access the very first element in the list, we have to provide an index as 0.
Similar to accessing the 3rd element from the list, pass the index value as 2.

To access the very last element either we can pass index as -1 or one less of the count of elements (8-1), which is 7. Just have a look the list is having 8 elements such as **['Python', 'Data', 'Structure', 'Lists', '123', 456, 78.56, True]**.

Next will check how to access the element if we have a list-of-list.
Consider the elements in the form of Matrix.

```
[
        ['Python'       , 'Data'                        , 'Structure'],
        ['Lists'                , Data Frames'                  , 'Tuples'],
        [123            , '456'                 , 78.6]
]
```

So to access the 2nd Element of the 1st List which is 'Data'. Let's see how it traverses.

- 1st list (index is 0) is ['Python' , 'Data' , 'Structure'],
- 2nd element from this list (with the Index as 1) is 'Data'

So to access the 1st Element of the 2nd List which is 'Lists'. Let's see how it traverses.

- 2nd list (index is 1) is ['Lists' , 'Data Frames' , 'Tuples'],
- the 1st element from this list (with the Index as 0) is 'Lists'.

```
# Creating an List within List (nested list)
nested_List = [['Python', 'Data', "Structure"] , ['Lists', 'Data Frames', 'Tuples'], [123, '456', 78.6]]
print("Nested List or List of List ")
print(nested_List)

print("*"*50)

print("Acessing a element from a List-of-list")
print(nested_List[0][1]) # To access 2nd Element of the 1st List.
print(nested_List[1][0]) # To access 1st Element of the 2nd List
```

```
Nested List or List of List
[['Python', 'Data', 'Structure'], ['Lists', 'Data Frames', 'Tuples'], [123, '456', 78.6]]
**************************************************
Acessing a element from a List-of-list
Data
Lists
```

Hope this helps to kick start your learning on Lists. Please do more practise, and play around the Lists.

Dictionary –

Dictionary is an unordered set of **key: value** pairs.

The keys are unique (within one dictionary).
A pair of braces creates an empty dictionary: {}.
Dictionary keys are case sensitive, same name but different cases of Key will be treated differently.

Jump into practicality.

To create an empty dictionary, we can just use **dict()** function or directly with { } . Recommended to use **dict()** method.

```
# Creating a empty Dictionary
empty_dict = dict()
print(empty_dict)
```

```
{}
```

```
# Creating a empty Dictionary
empty_dict = {}
print(empty_dict)
```

```
{}
```

Let's create a dictionary with Integer or number as a Key. Value can be of any data-type (such as Integer ; string; float; boolean etc).

```
# Creating a Dictionary with number or integer as a key.
integer_dict = {1: 'Python', 2: 'For', 3: 'Data', 4:"Analysis"}
print("Dictionary with the use of Number or Integer Keys ")
print(integer_dict)
```

```
Dictionary with the use of Number or Integer Keys
{1: 'Python', 2: 'For', 3: 'Data', 4: 'Analysis'}
```

Let's create a dictionary with string or a single character as a Key. Value can be of any data-type (such as Integer ; string; float; boolean etc).

```
# Creating a Dictionary with String as a key.
string_dict = {'a': 'Python', 'b': 'For', 'c': 'Data', 'd':"Analysis", 'e': 4758.66}
print("Dictionary with the use of string as a Keys ")
print(string_dict)
```

```
Dictionary with the use of string as a Keys
{'a': 'Python', 'b': 'For', 'c': 'Data', 'd': 'Analysis', 'e': 4758.66}
```

Let's create a dictionary with Integer or number and string as a Key i.e. mixed keys. Value can be of any data-type (such as Integer ; string; float; boolean etc).

```
# Creating a Dictionary with mixed keys.
mixed_dict = {'a': 'Python', 2: 'For', 'c': 'Data', 4:"Analysis", 'e': 4758.66}
print("Dictionary with the use of mixed Keys ")
print(mixed_dict)
```

```
Dictionary with the use of mixed Keys
{'a': 'Python', 2: 'For', 'c': 'Data', 4: 'Analysis', 'e': 4758.66}
```

Let's create a nested dictionary. Similar to this we can have a list as well in the dictionary.
Key 3 is another set of dictionaries which is nested in nested_dict.

```
# Creating a Nested Dictionary
nested_dict = {1: 'Python', 2: 'For', 3:{'A' : 'Data', 'B' : 'Analysis', 'C' : '4758.66'}}

print(nested_dict)
```

```
{1: 'Python', 2: 'For', 3: {'A': 'Data', 'B': 'Analysis', 'C': '4758.66'}}
```

Now we know how to create a dictionary, let's do some practical on adding new keys-pair.

Note- While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

First will create a dictionary (either it can be empty or it can have some data / elements).
Then will add new elements or key-pair to the dictionary.
In below example [0], [2], [3] are the Keys.

```python
# Creating a empty Dictionary
empty_dict = dict()
print(empty_dict)

print("*"*50)

# Adding elements one at a time
empty_dict[0] = 'Python'
empty_dict[2] = 'For'
empty_dict[3] = "Data Analysis"
print("Dictionary after adding 3 elements into the empty_dict ")
print(empty_dict)
```

```
{}
**************************************************
Dictionary after adding 3 elements into the empty_dict
{0: 'Python', 2: 'For', 3: 'Data Analysis'}
```

Accessing elements from a Dictionary →
In order to access the items of a dictionary refer to its key name.
Key can be used inside square brackets.
If the key does not exist then it will return the error. So make sure to access the value which has a key.

```python
# Creating a Dictionary with mixed keys.
mixed_dict = {'abc': 'Python', 2: 'For', 'c': 'Data', 4:"Analysis", 'e': 4758.66}
print("Dictionary with the use of mixed Keys ")
print(mixed_dict)

print("*"*50)

# accessing a element using key
print("Accessing a element using key")
print(mixed_dict['abc']) # To access the element using string key.
print(mixed_dict['c']) # To access the element using string key.
print(mixed_dict[2])     # to access the element using integer key.
print(mixed_dict[4])     # to access the element using integer key.
```

```
Dictionary with the use of mixed Keys
{'abc': 'Python', 2: 'For', 'c': 'Data', 4: 'Analysis', 'e': 4758.66}
**************************************************
Accessing a element using key
Python
Data
For
Analysis
```

In order to access the value of any key in a nested dictionary, use indexing [ ] syntax.

```
# Creating a Nested Dictionary
nested_dict = {"Languages": {1:'Python', 2:'R', 3:"Java", 4:"Scala"}, 2: 'Some Random Value', 'Packages':{'lib1' : 'Pandas',

print(nested_dict)

print("*"*50)

# Accessing element using key
print(nested_dict["Packages"]) # To access the Dictionary with key as 'Packages'.
print(nested_dict['Packages']['lib2'])  # To access the element with the Key as "lib2" of the Dictionary with key as "Package
print(nested_dict["Languages"][1])  # To access the element with the Key as integer 1 of the Dictionary with key as "Language
```

```
{'Languages': {1: 'Python', 2: 'R', 3: 'Java', 4: 'Scala'}, 2: 'Some Random Value', 'Packages': {'lib1': 'Pandas', 'lib2':
'Numpy', 3: 'Seaborn'}}
**************************************************
{'lib1': 'Pandas', 'lib2': 'Numpy', 3: 'Seaborn'}
Numpy
Python
```

After the List and Dictionary there are Tuples; Series; Set etc.. which are also useful, but to kick start, these two are sufficient. But please do learn others as well along with Basics of Python (will try to have articles on these in future).

Learn Python Libraries →

Let's take one step ahead in our journey to learn Python by getting acquainted with some useful libraries. There are numerous libraries provided and supported by Python. To name a few →

NumPy stands for Numerical Python. The most powerful feature of NumPy is the n-dimensional array. This library also contains basic linear algebra functions.

SciPy stands for Scientific Python. SciPy is built on NumPy. It is one of the most useful libraries for a variety of high level science and engineering modules.

Matplotlib for plotting a vast variety of graphs, starting from histograms to line plots to heat plots.

Pandas for structured data operations and manipulations. It is extensively used for data munging and preparation.

Scikit Learn for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Seaborn for statistical data visualization. Seaborn is a library for making attractive and informative statistical graphics in Python. It is based on matplotlib.

There are many more but these are often for advanced users, so this is enough for now. Practice with the libraries and the code presented here and lookout for more of my articles.

The first thing to learn about libraries, is how to import them. There are several ways of doing this in Python, but below are the two most common.

- **import numpy as np**
- **from numpy import ***

In the above screenshot, in the first cell we have imported the library and alias it with np. And then in cell 2 we have created an array using that alias np, and then printed the variable.

In the next cell # 4 which is an alternate way for importing the library. We do not need any alias for this, also here I am using * to include all the functions from numpy, if one is looking for any specific function then we can import that specific function only.

Example
# Importing specific function instead of all (*)
from math import exp
xp(3) # Calculates Exponential

```
# Importing specific function instead of all (*)
from math import exp
exp(3) # Calculates Exponential
```

```
20.085536923187668
```

Covering each and every library in one article would be difficult. At Least with this you will have gained a basic understanding and are now able to get started. I will create separate articles for each library soon.

Practice Mini Python Data Analysis projects →

Please refer to https://datascience.foundation/sciencewhitepaper/hands-on-with-first-ml-model
This will give you an easy way to start Data Analysis and Machine Learning. But again this is just a basic model and which can be enhanced for better results.

Apply Advanced Techniques →

That's it for now, I will present a few more advanced techniques in my next article.

Reference : Python Fundamental Jupyter Notebook

https://drive.google.com/file/d/1rYlHWKNQAa7WWoKkzDjyLIEO_JkALAep/view?usp=sharing
https://drive.google.com/file/d/1mT5IOgEmezkwbCkdhjl-Rag4afZV-Xau/view?usp=sharing

About the Data Science Foundation

The Data Science Foundation is a professional body representing the interests of the Data Science Industry. Its membership consists of suppliers who offer a range of big data analytical and technical services and companies and individuals with an interest in the commercial advantages that can be gained from big data. The organisation aims to raise the profile of this developing industry, to educate people about the benefits of knowledge based decision making and to encourage firms to start using big data techniques.

Contact Data Science Foundation

Email:admin@datascience.foundation
Telephone: 0161 926 3641
Atlantic Business Centre
Atlantic Street
Altrincham
WA14 5NQ
web: www.datascience.foundation