Data scientists use many programming tools such as Python, R, SAS, Java, Perl, and C/C++ to extract knowledge from data.
To extract this information, various fit-to-purpose models based on machine leaning, algorithms, statistics, and mathematical methods are used
Data scientists have to put

- In-depth knowledge of programming tools such as Python, R, SAS, Hadoop platforms, and SQL databases.
- Good knowledge of semi structured formats such as JSON, XML, HTML.
- Knowledge of how to work with unstructured data.

Python Lambdas Library

Pandas is an open-source Python library provides
High-performance data manipulation &
Analysis tools
Using Panda's powerful data structures
The name
Pandas is derived from "panel data" a term meaning multidimensional data
"Pandas" <= "panel data" <= multidimensional data
Key features of the Pandas library:

- Provides a mechanism to load data objects from different formats
- Creates efficient data frame objects with default and customized indexing
- Reshapes and pivots date sets
- Provides efficient mechanisms to handle missing data
- Merges, groups by, aggregates, and transforms data
- Manipulates large data sets by implementing various functionalities such as
    - Slicing,
    - Indexing,
    - Sub setting,
    - Deletion, &
    - Insertion
- Provides efficient time series functionality

Importing Pandas Package

Pandas is not standard with Python,
Hence Pandas doesn't come bundled with Python
Pandas has to be installed and imported

Installing Pandas

Option 1 installed python from https://www.python.org.

Use pip to install pandas (package installer for python)

Pip is built in from python 3 or later,

For older versions pip has to be installed first from https://www.python.org

Open the command prompt

Type the following command:

```
pip install pandas
```

Option 2 Install Anaconda software

Anaconda comes bundled with Panda and NumPy

Installing Numpy

Option 1 installed python from https://www.python.org.

Use pip to install NumPy(package installer for python)

Pip is built in from python 3 or later,

For older versions pip has to be installed first from https://www.python.org

Open the command prompt

Type the following command:

```
pip install numpy
```

Option 2 Install Anaconda software

Anaconda comes bundled with Panda and NumPy

importing pandas

Once Pandas is installed

It can be imported with commands

```
import pandas
import numpy
        or
import pandas as pd
import numpy as np
```

Note: Pandas and NumPy are almost always used in Tandem(together)

Data sets in Panda

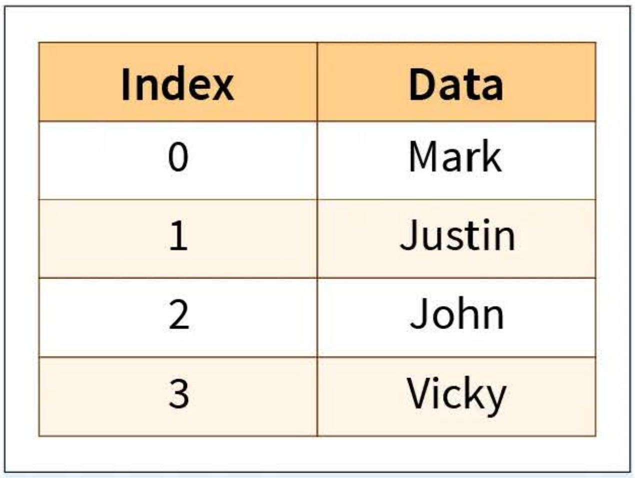
The Pandas library is used for creating and processing

Series

data frames &

panels.

A Pandas Series
A series is a one-dimensional labelled array
Series can hold data of any type
(integer, string, float, Python objects, etc.)



Index	Data
0	Mark
1	Justin
2	John
3	Vicky

The row labels of series are called the index.
A Series cannot contain multiple columns.
It has the following parameter:

- **data:** It can be any list, dictionary, or scalar value.
- **index:** The value of the index should be unique and hashable.
  It must be of the same length as data.
  If we do not pass any index, default np.arrange(n) will be used.
- **dtype:** It refers to the data type of series.
- **copy:** It is used for copying the data.

Creating a Series:
We can create a Series in two ways:

- **Create an empty Series**
- **Create a Series using inputs.**

Creating an Empty Series:
Creating an empty series in Pandas => series will not have any value.
The syntax for creating an Empty Series:

<series object> = pandas.Series()

Example

```
import pandas as pd
x = pd.Series()
print (x)
```

Output

```
Series([], dtype: float64)
```

Creating a Series using inputs:
Creating a Series by using various inputs:

- **Array**
- **Dict**
- **Scalar value**

Creating Series from Array:
Before creating a Series,
import the **numpy** module
use array() function of numpy
If the data is ndarray,
    then the passed index must be of the same length.
If an index is not passed,
    then by default index of **range(n)** is passed where n defines the length of an array,
    i.e., [0,1,2,....**range(len(array))-1**].

Example (without index)
```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])
a = pd.Series(info)
print(a)
```
Output
```
0   P
1   a
2   n
3   d
4   a
5   s
dtype: object
```

Example (with Index)
```
import numpy as np
data = np.array([90,75,50,66])
s = pd.Series(data,index=['A','B','C','D'])
print (s)
```
Output
```
A 90
B 75
C 50
D 66
dtype: int64
```

Create a Series from dictionary

Creating a Series from dict.

If the dictionary object is being passed as an input

index is not specified,

then the dictionary keys are taken in a sorted order to construct the index.

If index is passed,

then values correspond to a particular label in the index will be extracted from the dictionary.

Example (without Index)

```
import pandas as pd
import numpy as np
info = {'x' : 0., 'y' : 1., 'z' : 2.}
a = pd.Series(info)
print (a)
```

Output

```
x    0.0
y    1.0
z    2.0
dtype: float64
```

Example (with Index)

```
import numpy as np
data = {'B' : 92, 'A' : 55, 'C' : 83}
s = pd.Series(data, index=['A','B','C'])
print (s)
```

Output

```
A 55
B 92
C 83
dtype: int64
```

Create a Series using Scalar:

If scalar value is used,

then the index must be provided

The scalar value will be repeated for matching the length of the index.

Example

```
import pandas as pd
import numpy as np
x = pd.Series(4, index=[0, 1, 2, 3])
print (x)
```

Output

```
0    4
1    4
2    4
3    4
Dtype: Int64
```

Series Object Attributes

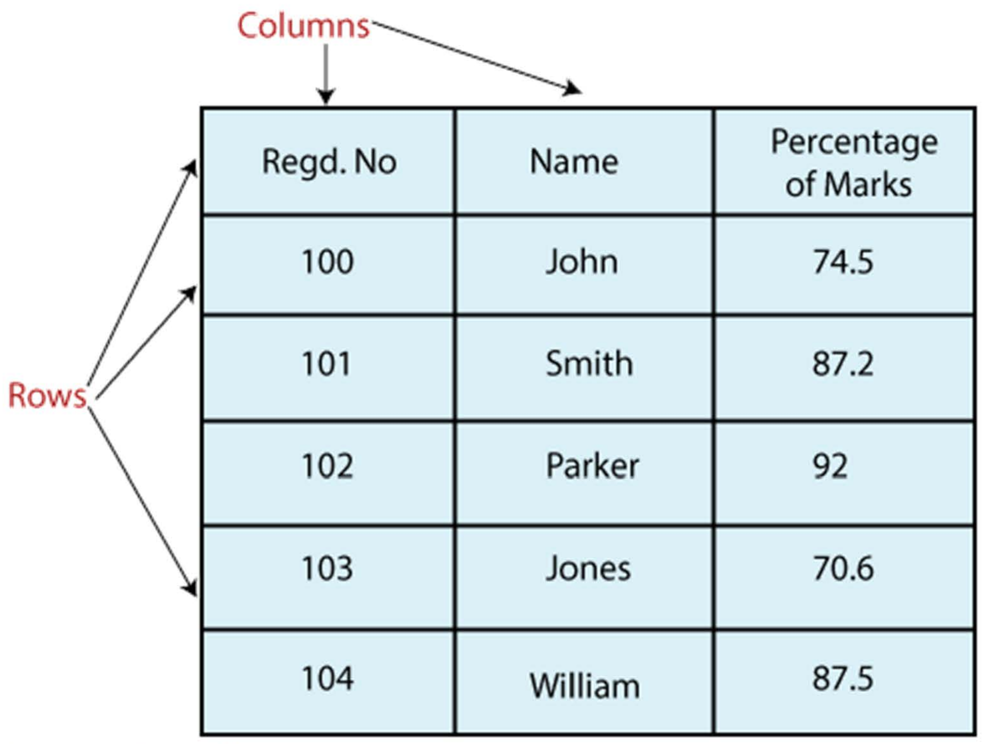
Attributes	**Description**
Series.index	Defines the index of the Series.
Series.shape	It returns a tuple of shape of the data.
Series.dtype	It returns the data type of the data.
Series.size	It returns the size of the data.
Series.empty	It returns True if Series object is empty, otherwise returns false.
Series.hasnans	It returns True if there are any NaN values, otherwise returns false
Series.nbytes	It returns the number of bytes in the data.
Series.ndim	It returns the number of dimensions in the data.
Series.itemsize	It returns the size of the datatype of item.

Pandas DataFrame

Pandas DataFrame data structure is a two-dimensional array with labeled axes (rows and columns).
DataFrame is defined as a standard way to store data that has two different indexes
i.e., row index and column index

Parameters of Pandas DataFrame:

data: It consists of different forms like ndarray, series, map, constants, lists, array.
index: The Default np.arrange(n) index is used for the row labels if no index is passed.
columns: The default syntax is np.arrange(n) for the column labels. It shows only true if no index is passed.
dtype: It refers to the data type of each column.
copy(): It is used for copying the data.



Regd. No	Name	Percentage of Marks
100	John	74.5
101	Smith	87.2
102	Parker	92
103	Jones	70.6
104	William	87.5

Create a DataFrame

DataFrame can be created from

- **Empty data**
- **dict**
- **Lists**
- **Series**

Creating an empty DataFrame

Example

```
import pandas as pd
df = pd.DataFrame()
print (df)
```

Output

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

Create a DataFrame using List:

Example

```
import pandas as pd
 # a list of strings
x = ['Python', 'Pandas']
df = pd.DataFrame(x)
print(df)
```

Output

```
     0
0   Python
1   Pandas
```

Creating a Data Frame Using the Pandas Library
Example

```
import pandas as pd
data = [['Ram',35],['Shyam',17],['Ajay',25]]
DataFrame1 = pd.DataFrame(data,columns=['Name','Age'])
print (DataFrame1)
```

Output

```
      Name     Age
0     Ram      35
1     Shyam    17
2     Ajay     25
```

Create a DataFrame from Dict of arrays/ Lists
Example

```
import pandas as pd
info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}
df = pd.DataFrame(info)
print (df)
    ID    Department
0   101      B.Sc
1   102      B.Tech
2   103      M.Tech
```

Create a DataFrame from Dict of Series:
Example

```
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),
'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g',
'h'])}
d1 = pd.DataFrame(info)
print (d1)
```

Output

```
     one    two
a    1.0     1
b    2.0     2
c    3.0     3
d    4.0     4
e    5.0     5
f    6.0     6
g    NaN     7
h    NaN     8
```

DataFrame Functions

Functions	Description
Pandas DataFrame.append()	Add the rows of other dataframe to the end of the given dataframe.
Pandas DataFrame.apply()	Allows the user to pass a function and apply it to every single value of the Pandas series.
Pandas DataFrame.assign()	Add new column into a dataframe.
Pandas DataFrame.astype()	Cast the Pandas object to a specified dtype.astype() function.
Pandas DataFrame.concat()	Perform concatenation operation along an axis in the DataFrame.
Pandas DataFrame.count()	Count the number of non-NA cells for each column or row.
Pandas DataFrame.describe()	Calculate some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame.
Pandas DataFrame.drop_duplicates()	Remove duplicate values from the DataFrame.
Pandas DataFrame.groupby()	Split the data into various groups.
Pandas DataFrame.head()	Returns the first n rows for the object based on position.
Pandas DataFrame.hist()	Divide the values within a numerical variable into "bins".
Pandas DataFrame.iterrows()	Iterate over the rows as (index, series) pairs.
Pandas DataFrame.mean()	Return the mean of the values for the requested axis.
Pandas DataFrame.melt()	Unpivots the DataFrame from a wide format to a long format.
Pandas DataFrame.merge()	Merge the two datasets together into one.
Pandas DataFrame.pivot_table()	Aggregate data with calculations such as Sum, Count, Average, Max, and Min.
Pandas DataFrame.query()	Filter the dataframe.
Pandas DataFrame.sample()	Select the rows and columns from the dataframe randomly.
Pandas DataFrame.shift()	Shift column or subtract the column value with the previous row value from the dataframe.
Pandas DataFrame.sort()	Sort the dataframe.
Pandas DataFrame.sum()	Return the sum of the values for the requested axis by the user.
Pandas DataFrame.to_excel()	Export the dataframe to the excel file.
Pandas DataFrame.transpose()	Transpose the index and columns of the dataframe.
Pandas DataFrame.where()	Check the dataframe for one or more conditions.

NumPy Library

Python Streams
Python stream is a programming paradigm (programming style) for data processing

Involves

Sequential processing of data items as they pass through a pipeline of processes.

Streams allow data processing to be continuous, effective, and memory-friendly
Dose not overload the entire dataset into memory at once.
Lambda functions are used in combination with the functions

functional programming
Functional programming is a programming paradigm (programming style)

that treats computation as evaluating mathematical functions and avoids changing state and mutable data.

In functional programming,

functions can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.

Functional programming is possible in Python using a number of features and tools, such as:

- **Higher-order functions**
- **Lambda Functions**
- **map, filter, and reduce**
- **List Comprehension**

Higher-order functions
Python enables the assignment of functions to variables,
passing of functions as arguments, and the return of functions as values.

Higher-order functions are the functions that accept other functions as arguments or return them as results and can be used as a result of this function.
Example
```
# Function that adds 1 to the passed value (x)
    def add(x):
        return x + 1
```
Output
```
    result = apply(add, 3)
                # Result: 4
```

Example

# Function that multiplies the passed value (x) by 2

```
def multiply(x):
        return x * 2
```

Output

```
result = apply(multiply, 3)
                # Result: 6
```

Lambda Functions

Lambda functions, also known as anonymous functions, can be defined inline without requiring a formal function declaration

Short and one-time-use functions

Helpful for the performance of a single task that only requires one line of code

Example

# Lambda function that adds 1 to x

```
add = lambda x: x + 1
```

Output

```
result = add(3)
                # Result: 4
```

Example

# Lambda function that multiplies x by 2

```
multiply = lambda x: x * 2
```

Output

```
result = multiply(3)
                # Result: 6
```

map, filter, and reduce

Map, Filter, and Reduce are built-in Python functions that can be used for functional programming tasks.

map() function

The map() function is used to apply a specific function on a sequence of data

map() syntax is : map(function, iterable)

first argument is a function

second argument is an iterable (sequence of elements) such as a list, tuple, set, string, etc.

map() function has two arguments.

Eg..

```
r = map(func, seq)
```

Here,

func is the name of a function to apply
seq is the sequence (e.g., a list)
func applies the function to all the elements of the sequence seq

Example map()

# Using the map to apply a function to each element, Applying Lambda function returns the square of x

```
data = [1, 2, 3, 4, 5]
result = map(lambda x: x * 2, data)
```

Output

```
# Result: [2, 4, 6, 8, 10]
```

filter() Function

The filter() function is used to filter out all elements of a list for which the applied function returns false.

filter() syntax is : filter(function, iterable)

first argument is a function
second argument is an iterable (sequence of elements)
such as a list, tuple, set, string, etc.

Eg..

r = filter(func, list)
func is a function that returns a Boolean value
list is a sequence of elements

Only if func returns true for the element it will be included in the result list.

Example filter()

# Using the filter-to-filter elements based on a condition, Using Lambda function to return True for even numbers

```
data = [1, 2, 3, 4, 5]
result = filter(lambda x: x % 2 == 0, data)
```

Output

```
# Result: [2, 4]
```

The reduce () Function

reduce() is a built-in function that applies a given function to the elements of an iterable,
reducing them to a single value.
The reduce() function continually applies the function func to a sequence seq and returns a single value.

reduce() syntax is : reduce(function, iterable[, initializer])

Reduce() receives function as an argument
This function takes two arguments and returns a single value
first argument is accumulated value
second argument is the current value from the iterable.
iterable argument => sequence of values to be reduced.
initializer argument (optional)
=> an initial value for the accumulated result
Note: If no initializer is specified, the first element of the iterable is used as the initial value.

Example reduce()

```
# Using reduce to aggregate elements of the list, Applying Lambda function that returns product of x and y
    data = [1, 2, 3, 4, 5]
    result = reduce(lambda x, y: x * y, data)
```

Output

```
            # Result: 120
```

List Comprehension

List comprehensions, are simple and expressive techniques to build new lists from older ones
Functional programming techniques such as mapping, filtering, and aggregating can be implemented using list comprehensions.

Building New List using map() function
Example map()

```
    # Using the map to create new list Squares
    data = [1, 2, 3, 4, 5]
    squares = list(map(lambda x: x*x, data))
    print("Original list: {data}")
    print("Squares: {squares}")
```

Output

```
Original list: [1, 2, 3, 4, 5]
Squares: [1, 4, 9, 16, 25]
```

Building New List using map() function with multiple variables
Example map()

```
# Using the map to create new list answers ans =base^^power
base = [2, 4, 6, 8, 10]
power = [1, 2, 3, 4, 5]
answer = list(map(pow, base, power))
print("Answer:", answer)
```

Output

```
Answer: [2, 16, 216, 4096, 100000]
```

Building New List using filter() function

Example filter()

```
# Using the filter() to create new list Squares
data = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x % 2 == 0, data))
print("Original list: {data}")
print("Evens = {evens}")
```

Output

```
Original list: [1, 2, 3, 4, 5]
Evens = [2, 4]
```

Building New List using filter() function
Example filter()

```
# Using the filter() to create new list of perfect Squares from
random list
from math import sqrt
data = [0, 1, 4, 6, 8, 9, 10, 12, 16, 81, 23, 36]
def isPerfectSqr(i):
    return sqrt(i).is_integer()
answer = list(filter(isPerfectSqr, data))
print("Original list: {data}")
print ("Answer: ", answer)
```

Output

```
Original list: [0, 1, 4, 6, 8, 9, 10, 12, 16, 81, 23, 36]
Evens = [0, 1, 4, 9, 16, 81, 36]
```

Building New List using filter() function
Example filter()

```
# Using filter() to filter names starting with letter H
# A list containing names
names = ["Arun", "Sonu", "Harsh", "Harry", "Anu", "Hassi"]
H = list(filter(lambda x: x[0] == 'H', names))
print("Original list: {names}")
print ("Result List: ", H)
```

Output

```
Original list: ["Arun", "Sonu", "Harsh", "Harry", "Anu", "Hassi"]
Result List = ['Harsh', 'Harry', 'Hassi']
```

Aggerating List using reduce() function

Example reduce()

```
# sum of list using reduce()
from functools import reduce
# define add function
def add(a, b):
        return a + b
num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = reduce(add, num_list)
print("Original list: { num_list }")
print("Sum of the integers of num_list : {sum}")
```

Output

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of the integers of num_list : 55
```

#
Example reduce()

```
# reduce function with 10 as an initial value
from functools import reduce
# define add function
def add(a, b):
return a + b
num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = reduce(add, num_list,10)
print("Original list: { num_list }")
print("Sum of the integers of num_list : {sum}")
```

Output

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of the integers of num_list : 65
```

Example reduce()

```
# operator functions with reduce function
from functools import reduce
import operator
list1 = [1, 2, 3, 4, 5]
list2 = ["I", "Love", "Bharath"]
sum = reduce(operator.add, list1)
product = reduce(operator.mul, list1)
str1 = reduce(operator.concat, list2)
str2 = reduce(operator.add, list2)
print("list1: {list1}")
print("list2: { list2}")
print("Sum of all elements in list1 : {sum}")
print("Product of all elements in list1 : {product}")
print("Concatenated string by using operator.concat : {str1}")
print("Concatenated string by using operator.add : {str2}")
```

Output

```
list1: [1, 2, 3, 4, 5]
list2: ["I", "Love", "Bharath"]
Sum of all elements in list1 : 15
Product of all elements in list1 : 120
Concatenated string by using operator.concat : ILove Bharath
Concatenated string by using operator.add : ILove Bharath
```