NumPy

CONTENTS

- Introduction
- NumPy Objects
- Creating arrays
- Indexing and slicing
- Iterators
- Manipulation of array
- Statistical operations
- Algorithms
- Working with matrices

Introduction

Numpy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

NumPy – Numerical Python

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.

Introduction

Numpy Objects

Creating arrays

Indexing and slicing

Iterators

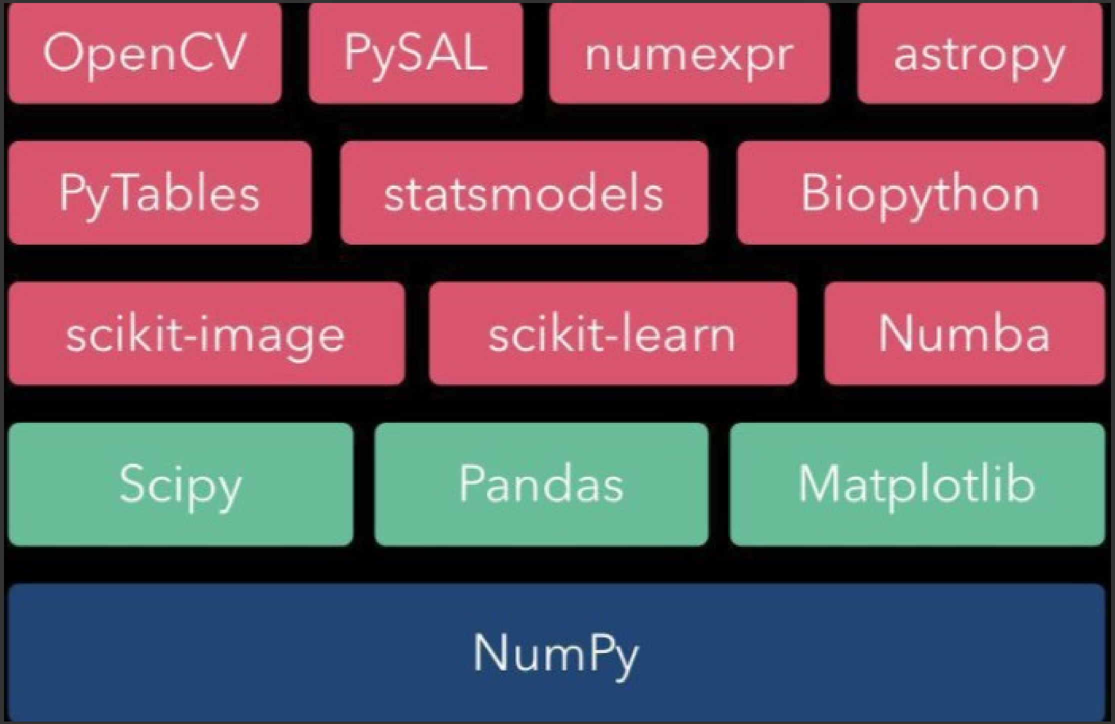
Manipulation of array

Statistical operations

Algorithms

Working with matrices

NumPy - Echo system



OpenCV PySAL numexpr astropy

PyTables statsmodels Biopython

scikit-image scikit-learn Numba

Scipy Pandas Matplotlib

NumPy

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

NumPy Objects:

The most important object defined in NumPy is an N-dimensional array type called “ndarray”.

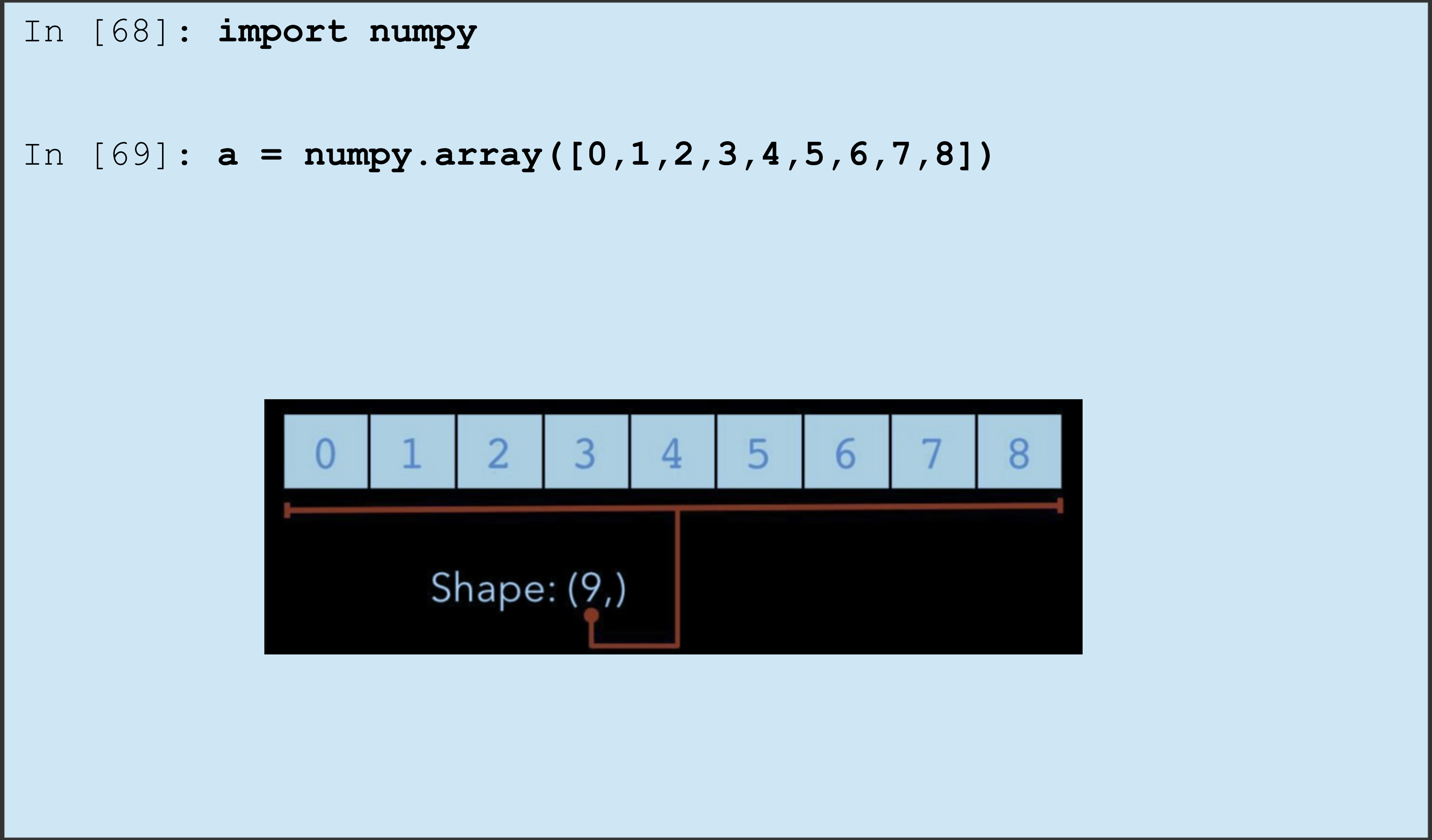
It describes the collection of items of the same type.

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).

Let us start a simple program:



```
In [68]: import numpy


In [69]: a = numpy.array([0,1,2,3,4,5,6,7,8])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

```
A= numpy.array(object, dtype=None, copy=True, order='K',
subok=False, ndmin=0)

Parameters
----------
object : array_like
    An array, any object exposing the array interface, an
object whose

dtype : data-type, optional
    The desired data-type for the array.

copy : bool, optional

order : {'C', 'F'}, optional

subok : bool, optional

ndmin : int, optional
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

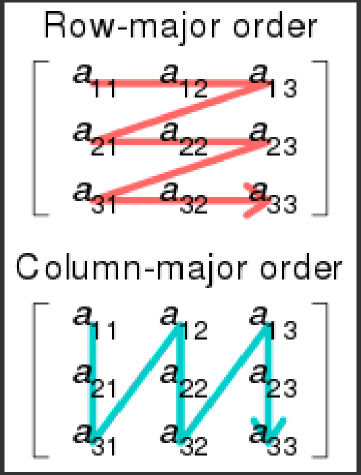
Statistical operations

Algorithms

Working with matrices

order:

In computing, **row-major order** and **column-major order** are methods for storing multidimensional arrays in linear storage such as random access memory.


Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

dtype:

- NumPy arrays comprise elements of a single data type
- The type object is accessible through the .dtype attribute Here are a few of the most important attributes of dtype objects.
- Array dtypes are usually inferred automatically, but can also be specified explicitly

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

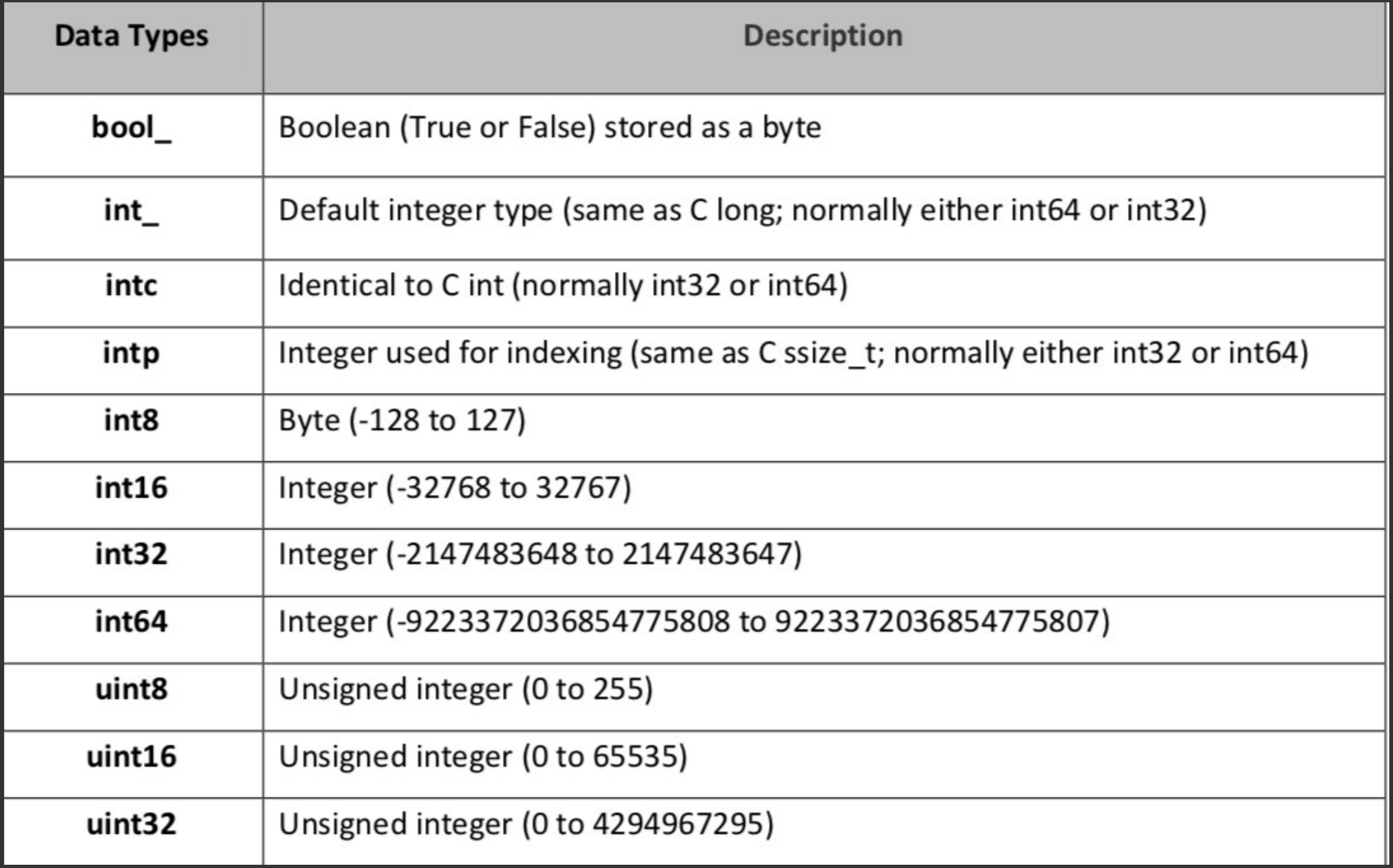
Manipulation of array

Statistical operations

Algorithms

Working with matrices

Mentioning data types is very important:



Data Types	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

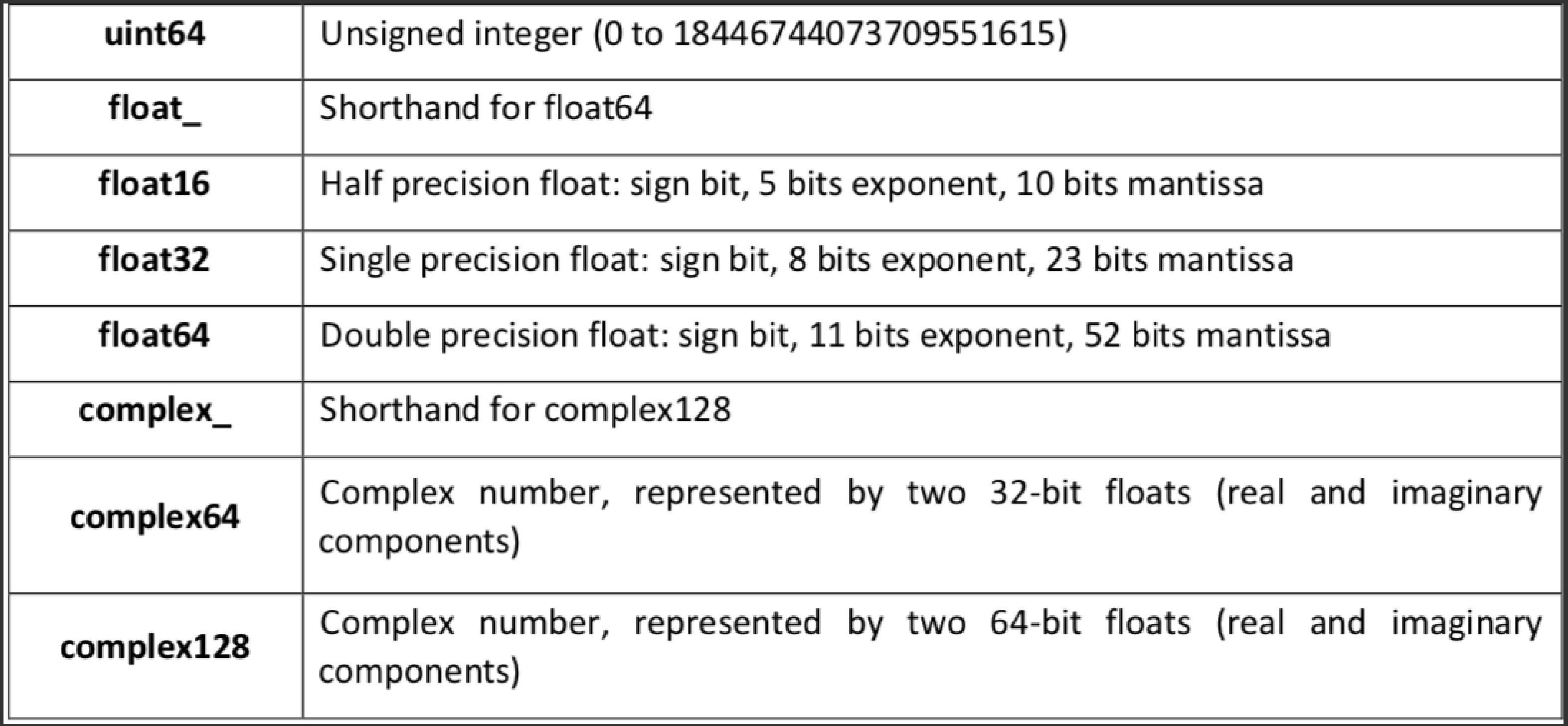
Manipulation of array

Statistical operations

Algorithms

Working with matrices

Mentioning data types is very important: Cont…



uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

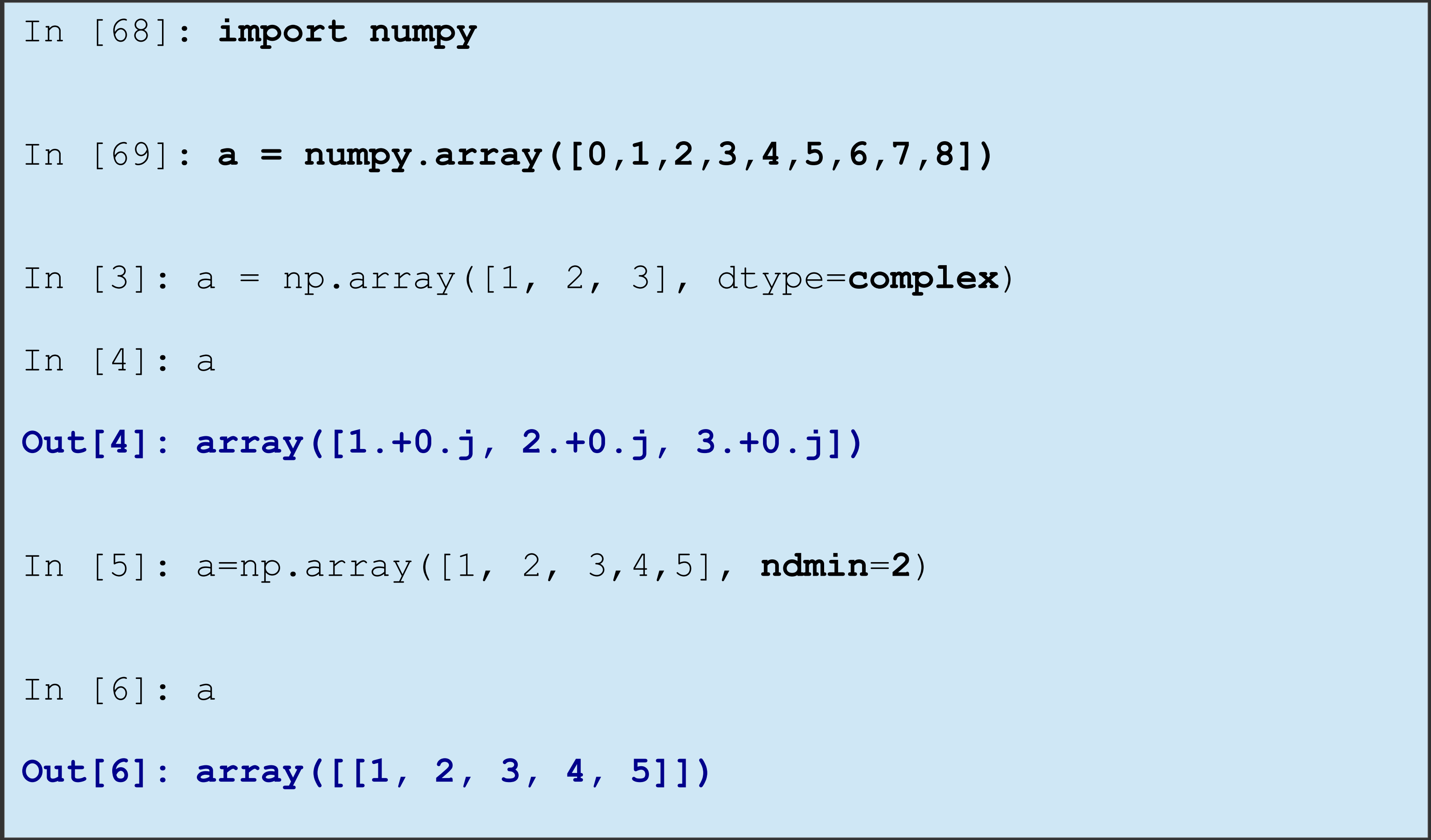
Manipulation of array

Statistical operations

Algorithms

Working with matrices

Let us start a simple program:



```
In [68]: import numpy

In [69]: a = numpy.array([0,1,2,3,4,5,6,7,8])

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

In [4]: a

Out[4]: array([1.+0.j, 2.+0.j, 3.+0.j])

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

In [6]: a

Out[6]: array([[1, 2, 3, 4, 5]])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Numpy operations are faster than lists:

```
import time
import numpy as np
num = 10000000
count = num
l1=list(range(count))
l2=list(range(count))

start=time.time()
list3 = [l1[i]+l2[i] for i in range(num)]
end=time.time()
elapsed = 1000*(end-start)
print("List time:",elapsed)

a1 = np.arange(count)
a2 = np.arange(count)
start=time.time()
a3=a1+a2
end=time.time()
elapsed = 1000*(end-start)
print("Numpy time:",elapsed)
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Numpy arrays are memory efficient:

```
import sys
import numpy as np
list1 = [1,3,7,100,4,6,10,45,3,6,8,99,87]
print("Size taken by list = ", sys.getsizeof(list1))

a = np.array([1,3,7,100,4,6,10,45,3,6,8,99,87],dtype=np.int8)
print("Size taken by numpy array = ", sys.getsizeof(a))
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

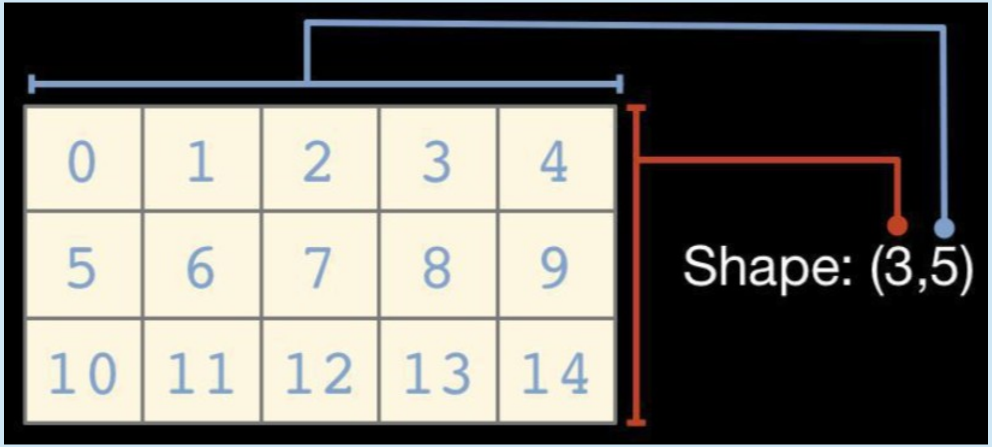
Statistical operations

Algorithms

Working with matrices

Creating Two dimensional array:

```
In [91]: m1 = numpy.array([[0,1,2,3,4],[5,6,7,8,9],
[10,11,12,13,14]], dtype="int64")

In [92]: m1.ndim

Out[92]: 2

In [93]: m1.size

Out[93]: 15

In [94]: m1.nbytes

Out[94]: 120

In [95]: m1.shape

Out[95]: (3, 5)
```



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Creating using arange(init, final, increment)

```
array1 = np.arange(21,32)
print("array1 = ",array1)



array1 =  [21 22 23 24 25 26 27 28 29 30 31]
```

Creating using linspace(init, final, number of elements)

```
array2 = np.linspace(12,22,11)


array2 =  [12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22.]
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Creating using “random()”:

```
ar = np.random.rand(10)
print(ar)


[0.76165674 0.290724   0.0366434  0.56961439 0.55207746 0.14025679
 0.01142086 0.28403208 0.13511246 0.49535445]
```

Creating using linspace(init, final, number of elements)

```
a = numpy.linspace(0,1, 100)
import matplotlib.pyplot as plt
plt.plot(a,numpy.sin(a*np.pi*2))
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Creating using “zero()”:

```
In  [4]: np.zeros((2,2))

Out[4]:
array([[ 0., 0.],
[ 0., 0.]])
```

Creating using “ones()”:

```
In  [13]: np.ones((1,5))


Out[13]: array([[1., 1., 1., 1., 1.]])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

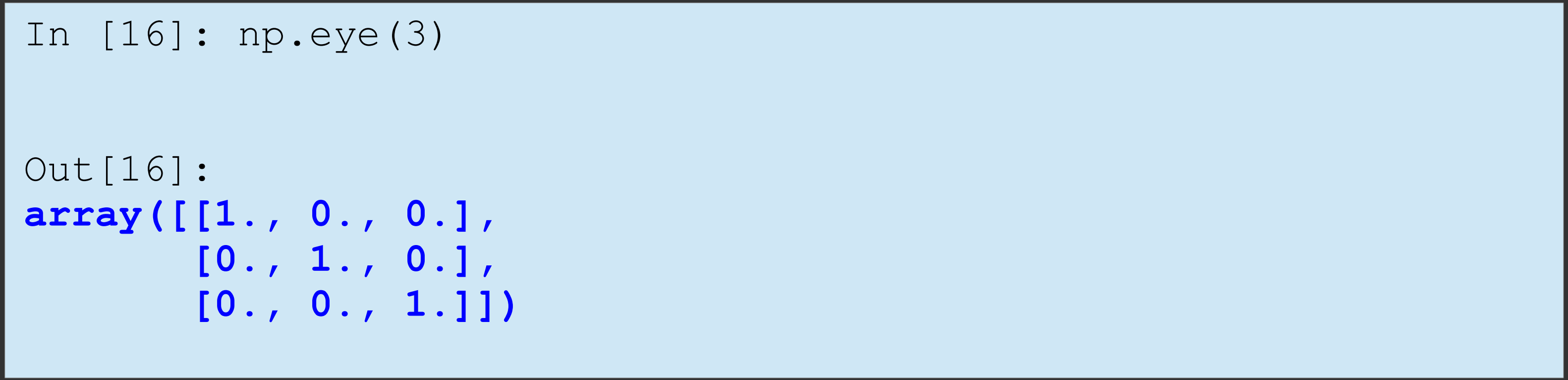
Creating using “empty()”:

```
In [14]: np.empty((1,5))


Out[14]: array([[1., 1., 1., 1., 1.]])
```

Creating using “ones()”:

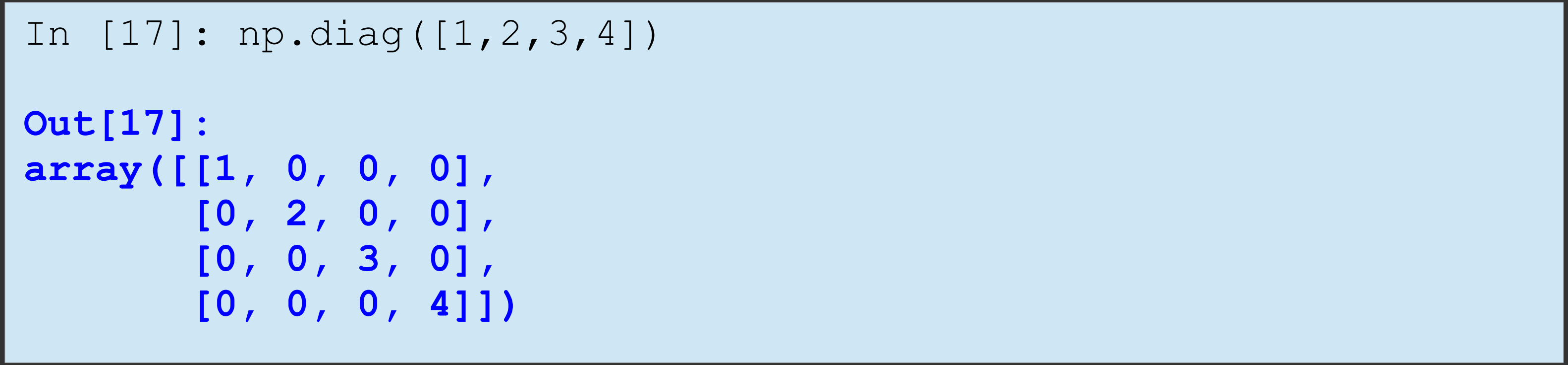
```
In [14]: np.empty((1,5))


Out[14]: array([[1., 1., 1., 1., 1.]])
```

Creating using “eye()”:



```
In [16]: np.eye(3)


Out[16]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Creating using “diag()”:



```
In [17]: np.diag([1,2,3,4])

Out[17]:
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

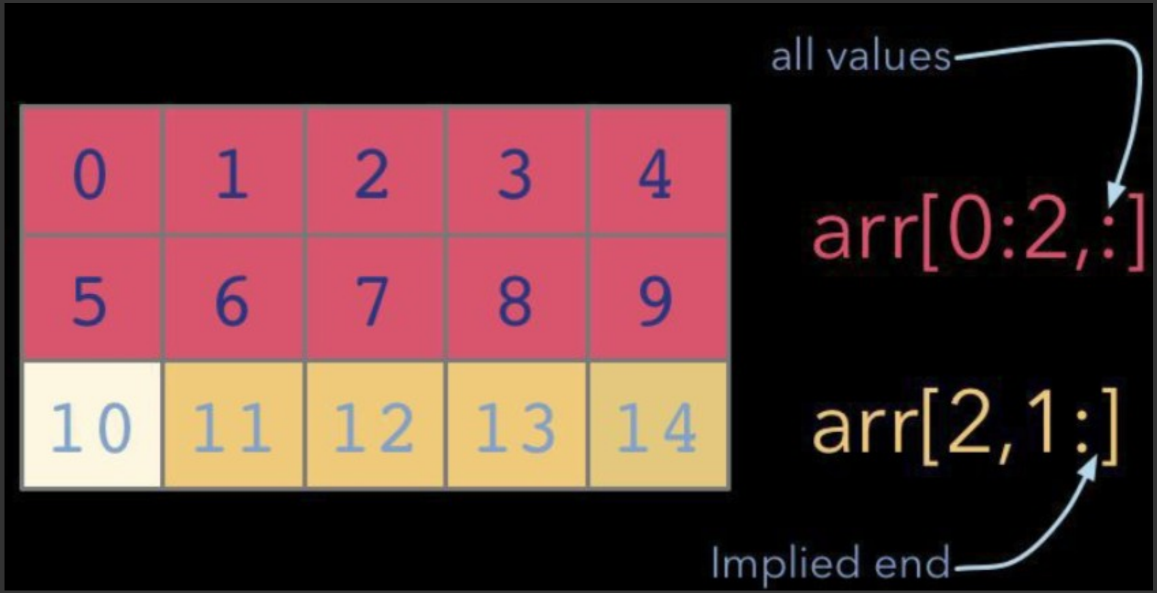
Statistical operations

Algorithms

Working with matrices

Indexing:

- Same as lists/tuples/string:



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

```
arr[0:2,:]
arr[2,1:]
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

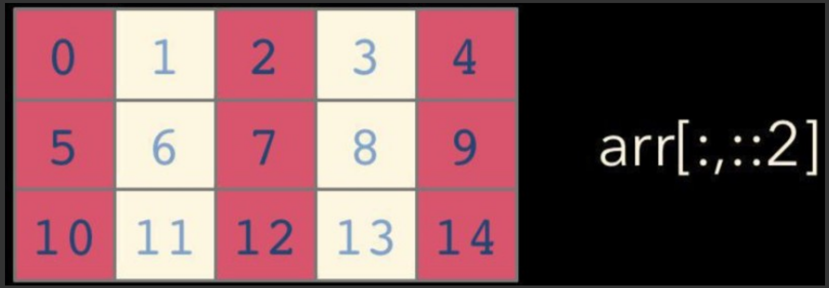
Iterators

Manipulation of array

Statistical operations

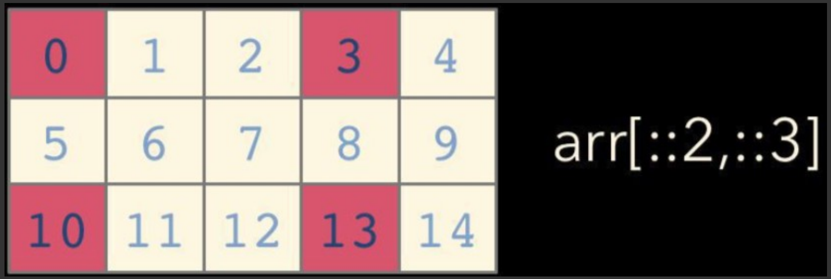
Algorithms

Working with matrices



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[:,::2]



0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

arr[::2,::3]

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Operations using vectors:

```
#Vector addition, subtraction, multiplication, division,
modulus,
x = np.array([2,4,6])
y = np.array([1,3,5])
add = x+y
sub = x-y
div = x/y
mul = x*y
mod = x%y
matmul = np.matmul(x,y) or x@y or np.dot(x,y)
a**2+b**2+2*a*b (Where a and b are two arrays)
print("matmul", matmul)
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Comparison : <, <=, ==, !=, >=, > arithmetic: +, -, *, /, reciprocal, square

Exponential : exp, expm1, exp2, log, log10, log1p, log2, power, sqrt

Trigonometric : sin, cos, tan, acsin, arccos, atctan hyperbolic: sinh, cosh, tanh, acsinh, arccosh, atctanh

Bitwise operations : &, |, ~, ^, left_shift, right_shift logical

Operations : and, logical_xor, not, or

Predicates : isfinite, isinf, isnan, signbit

Other : abs, ceil, floor, mod, modf, round, sinc, sign, trunc

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Broadcasting:

```
array = np.ones(30)

Array[:] = 100

[100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
100. 100.
 100. 100.]
```

reshape():

```
a = np.arange(30)
array_matrix = a.reshape(5,6)

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

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

:

```
array = np.ones(30)

Array[:] = 100

[100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
100. 100.
 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100. 100.
100. 100.
 100. 100.]
```

reshape():

```
a = np.arange(30)
array_matrix = a.reshape(5,6)

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

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Relations operators:

```
arr = np.eye(3)

arr[arr == 0] = 21

array([[ 1., 21., 21.],
       [21.,  1., 21.],
       [21., 21.,  1.]])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Getting statistics about an array:

```
In [51]: reviews = numpy.array([[3.3,5,4,2.8,5,4],[1,2,3,4,5,6]])


In [52]: reviews.sum()

Out[52]: 45.1

In [53]: reviews.sum(axis=0)

Out[53]: array([ 4.3,  7. ,  7. ,  6.8, 10. , 10. ])

In [54]: reviews.sum(axis=1)

Out[54]: array([24.1, 21. ])

In [55]: reviews.mean()

Out[55]: 3.7583333333333333
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

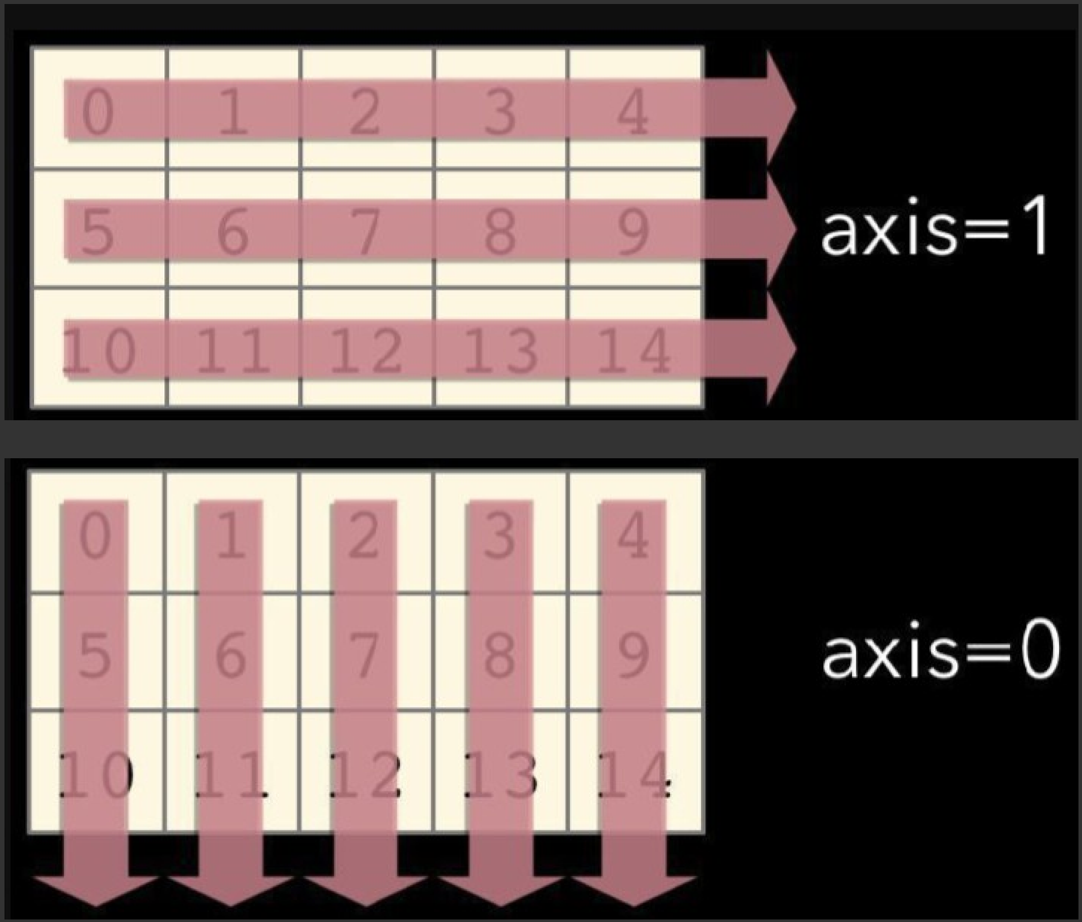
Manipulation of array

Statistical operations

Algorithms

Working with matrices

axis =0, axis = 1



Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Getting statistics about an array:

```
In [56]: reviews.max()

Out[56]: 6.0

In [57]: reviews.std()

Out[57]: 1.361040247588423

In [58]: reviews.var()

Out[58]: 1.852430555555556

In [59]: reviews.var(axis=0)

Out[59]: array([1.3225, 2.25  , 0.25  , 0.36  , 0.    , 1.    ])

In [60]: reviews.var(axis=1)

Out[60]: array([0.65472222, 2.91666667])
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

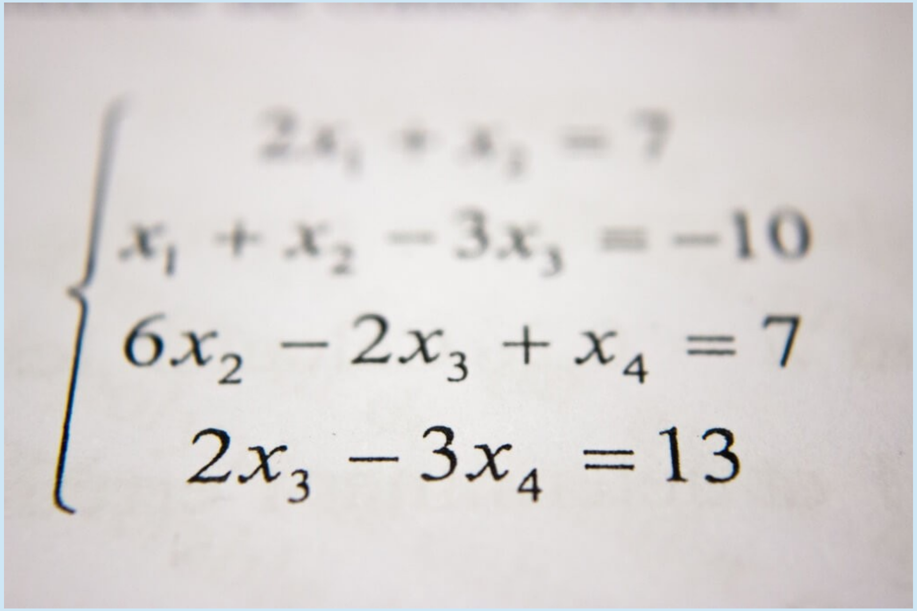
Statistical operations

Algorithms

Working with matrices

Linear Algebra: Solving equation

```
In [130]: a = numpy.array([[1,2],[3,4]],ndmin=2)


In [131]: b = numpy.array([1,2])


In [132]: numpy.linalg.solve(a,b)

Out[132]: array([0. , 0.5])
```



Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

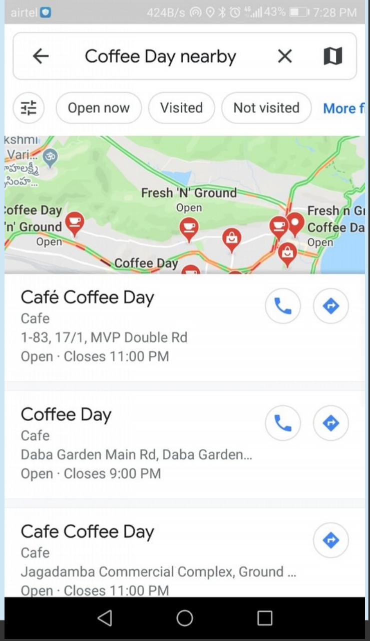
Statistical operations

Algorithms

Working with matrices

Linear Algebra: Finding euclidean distance

```
In [133]: point = numpy.array([1,3])


In [134]: points = numpy.array([[1,2],[3,4],[6,7]])


In [135]: index = numpy.argmin(numpy.linalg.norm(points-
point,axis=1))

In [136]: print(index)

0
```



Café Coffee Day
Cafe
1-83, 17/1, MVP Double Rd
Open · Closes 11:00 PM

Coffee Day
Cafe
Daba Garden Main Rd, Daba Garden...
Open · Closes 9:00 PM

Cafe Coffee Day
Cafe
Jagadamba Commercial Complex, Ground ...
Open · Closes 11:00 PM

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Difference between “matrix” and “array”

```
In [65]: a = np.array([1,2,3])


In [66]: print("Ndim = ", a.ndim)

Ndim =  1

In [67]: print("Shape = ", a.shape)

Shape =  (3,)

In [68]: a = np.matrix([1,2,3])


In [69]: print("Ndim = ", a.ndim)

Ndim =  2

In [70]: print("Shape = ", a.shape)

Shape =  (1, 3)
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Operations using matrices:

```
#Matrix operations
x = np.matrix( ((2,3), (3, 5)) )
y = np.matrix( ((1,2), (5, -1)) )
print(x+y)
print(x*y)
print(x**2)
print(x**-1)
Indexing: x[1,1]
```

Introduction

NumPy Objects

Creating arrays

Indexing and slicing

Iterators

Manipulation of array

Statistical operations

Algorithms

Working with matrices

Linear Algebra: Finding euclidean distance

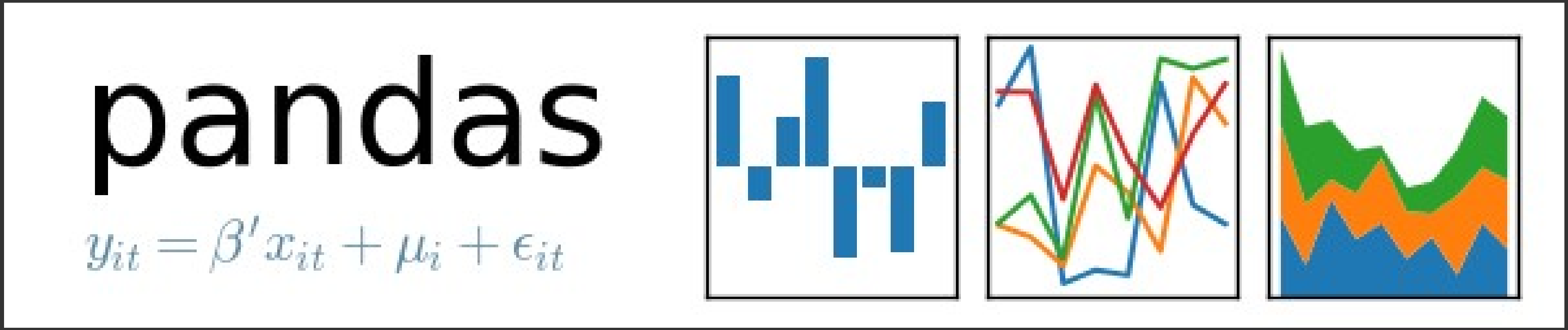
```
a = np.matrix([1,2,3,4]).reshape((2,2))
b = np.matrix([1,2,3,4]).reshape([2,2])

print(numpy.concatenate([a,b], axis=0))
```

```
[[1, 2],
       [3, 4],
       [1, 2],
       [3, 4]]
```

```
print(numpy.concatenate([a,b], axis=1))
```

```
[[1 2 1 2]
 [3 4 3 4]]
```

Thank You......


pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

CONTENTS

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting data

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

PanDaS - Panel Data Sets

- **Pandas provides** set of functions to process various types of data.
- Panda is fast, easy and more expressive than other tools

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Series:

- Series is a one-dimensional array.
- The row labels in a Series are called the index.
- Any list, tuple and dictionary can be converted in to Series using ‘series’ method.

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Creating Series using dictionary:

```
In [6]: dict1 = {"One":1, "Two":2}


In [7]: s1 = pd.Series(dict1)


In [8]: s1

Out[8]:
One     1
Two     2
dtype: int64

In [9]: s1.index

Out[9]: Index(['One', 'Two'], dtype='object')

In [10]: s1.values

Out[10]: array([1, 2])
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Creating Series using Tuple:

```
In [21]: b = (8,1,25,1,0,0,3.13)


In [22]: s = pd.Series(b)
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Adding “Index” using list:

```
In [35]: s =
pd.Series(b,index=["overs","M","runs",'wickets',"nb","wd","avg"])


In [36]: s

Out[36]:
overs        8.00
M            1.00
runs        25.00
wickets      1.00
nb           0.00
wd           0.00
avg          3.13
dtype: float64
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Accessing data in Series:

```
In [44]: s['M']

Out[44]: 1.0

In [45]: s['overs']

Out[45]: 8.0

In [46]: s['wickets']

Out[46]: 1.0
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Using “Describe()”:

```
In [52]: Bumrah = (3,4,2,1,6,3,0)


In [53]: s = pd.Series(Bumrah)


In [54]: s.describe()

Out[54]:
count    7.000000
mean     2.714286
std      1.976047
min      0.000000
25%      1.500000
50%      3.000000
75%      3.500000
max      6.000000
dtype: float64
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Using “Describe()”:

```
In [59]: s = pd.Series(Bumrah)


In [60]: s[0] = 3#Modifying the data


In [61]: s[7] = 7#Adding new data

In [62]: s

Out[62]:
0     3
1     4
2     2
3     1
4     6
5     3
6     0
7     7
dtype: int64
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Dropping duplicates:

```
In [71]: s1

Out[71]:
0     3
1     3
2     2
3     1
4     6

dtype: int64

In [72]: s1.drop_duplicates()

Out[72]:
0     3
2     2
3     1
4     6
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Other operations”:

```
In [64]: s+10    #Adding 10 to all the values


In [63]: s-1     #Subtracting 1 from all the values

In [68]: s1+s2   #Adding two series
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Data Frame:

DataFrame is the widely used data structure of pandas.
DataFrame has two different index i.e. column-index and row-index.

The most common way to create a DataFrame is by using the dictionary of equal-length list
ll the spreadsheets and text files are read as DataFrame

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Creating data frames:

```
In [86]: bowlers = {'Bhuvneshwar Kumar':
[8.1,1,30,1,0,0,3.67],'Jasprit Bumrah':
[8,1,25,1,0,0,3.13],'Hardik Pandya':[10,0,55,1,0,5,
    ...: 5.50]}


In [87]: df1 = pd.DataFrame(data=bowlers)


In [88]: df1

Out[88]:
   Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya
0               8.10            8.00           10.0
1               1.00            1.00            0.0
2              30.00           25.00           55.0
3               1.00            1.00            1.0
4               0.00            0.00            0.0
5               0.00            0.00            5.0
6               3.67            3.13            5.5
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

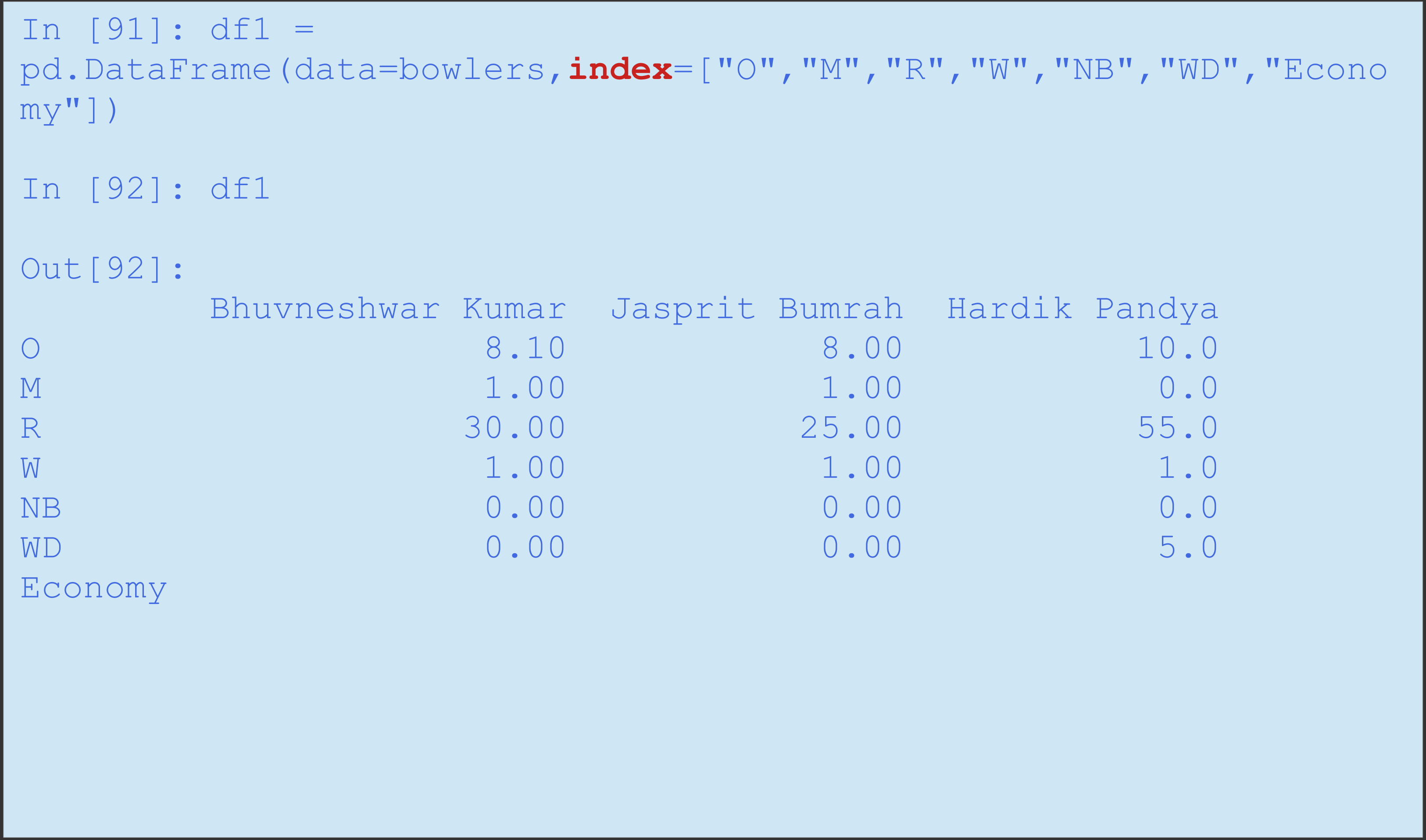
DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Creating data frames: Adding “index”:



```
In [91]: df1 =
pd.DataFrame(data=bowlers,index=["O","M","R","W","NB","WD","Econo
my"])

In [92]: df1

Out[92]:
         Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya
O                     8.10            8.00           10.0
M                     1.00            1.00            0.0
R                    30.00           25.00           55.0
W                     1.00            1.00            1.0
NB                    0.00            0.00            0.0
WD                    0.00            0.00            5.0
Economy
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

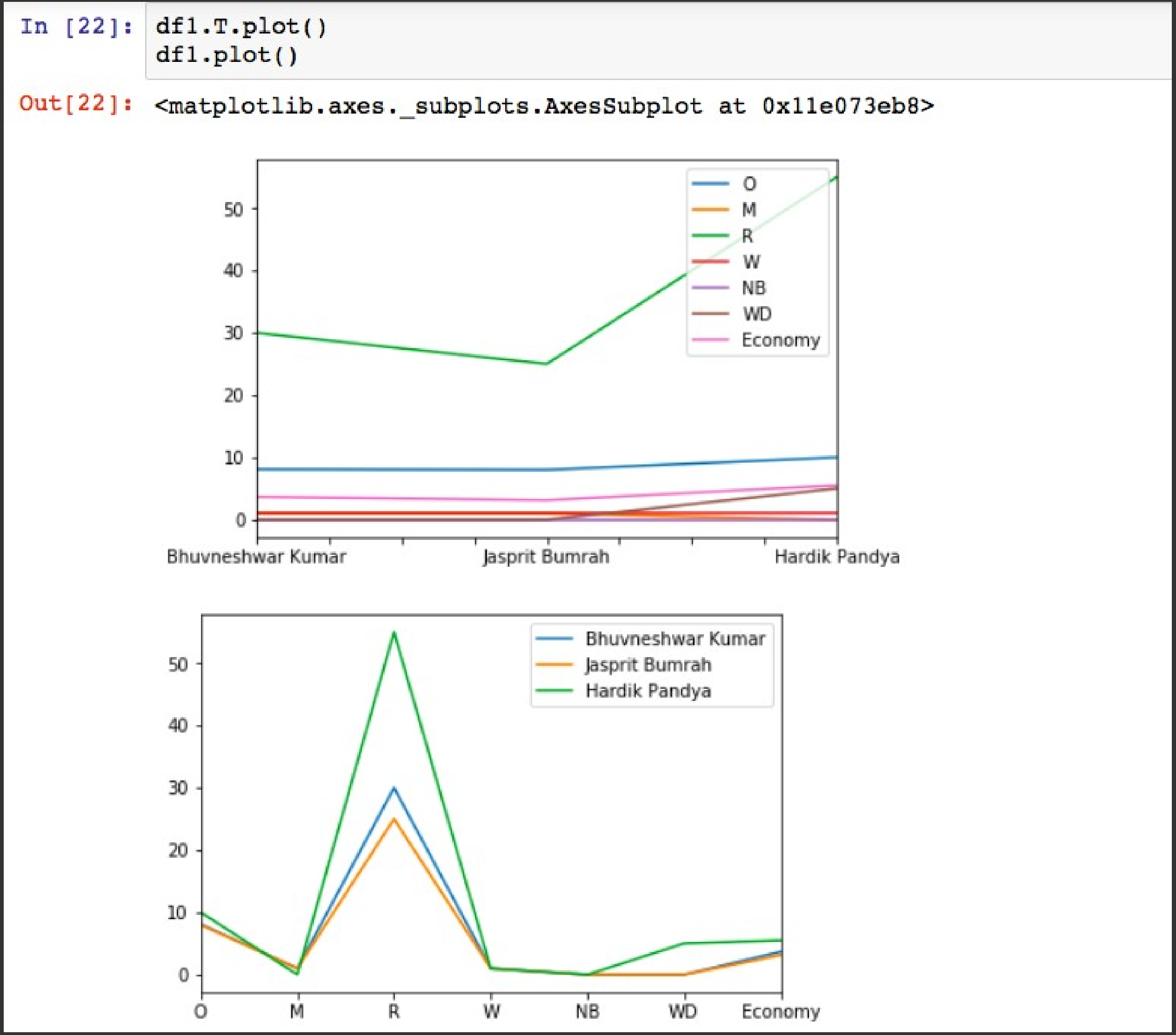
DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping



```
In [22]: df1.T.plot()
         df1.plot()
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11e073eb8>
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

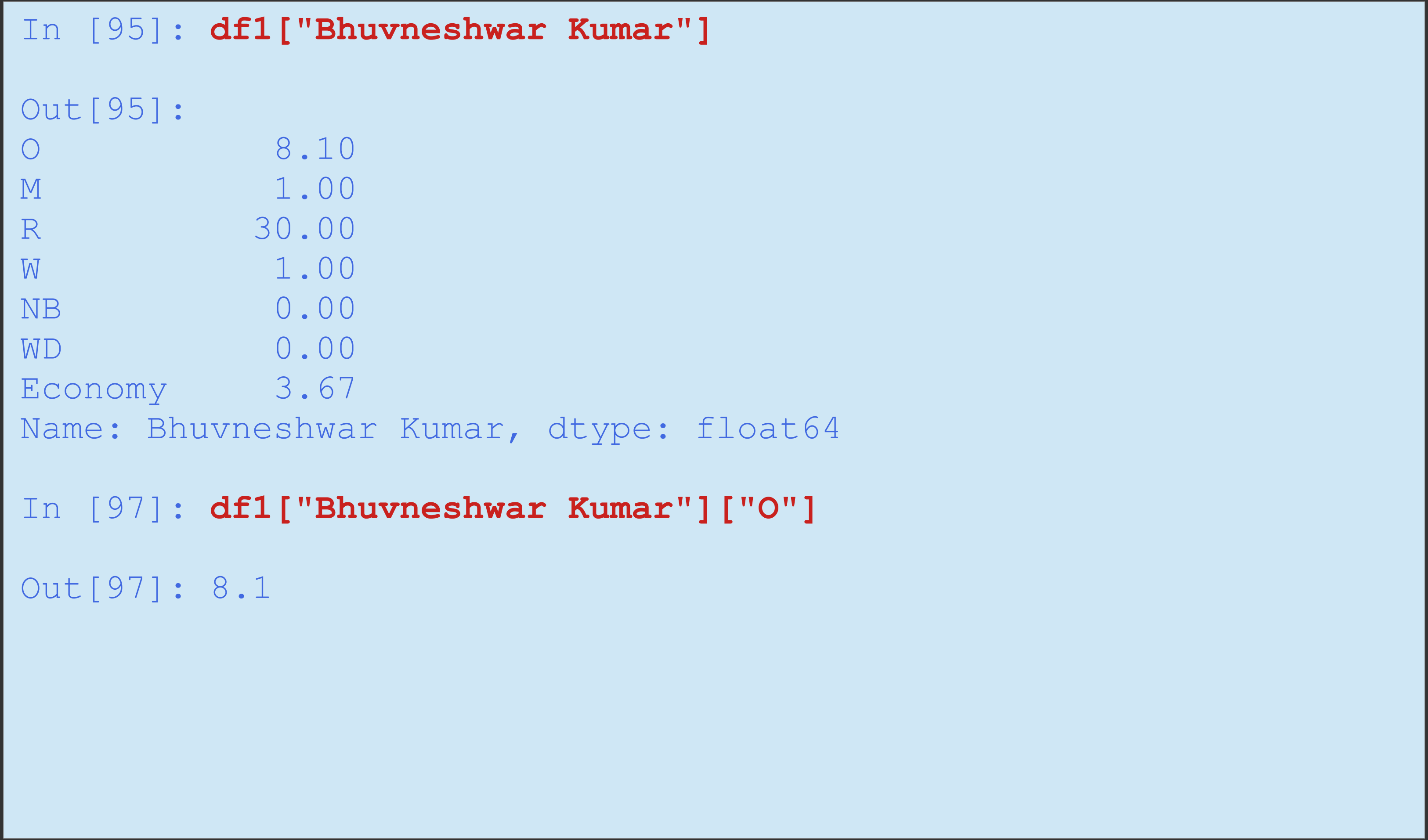
DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Accessing the data:



```
In [95]: df1["Bhuvneshwar Kumar"]

Out[95]:
O              8.10
M              1.00
R             30.00
W              1.00
NB             0.00
WD             0.00
Economy        3.67
Name: Bhuvneshwar Kumar, dtype: float64

In [97]: df1["Bhuvneshwar Kumar"]["O"]

Out[97]: 8.1
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

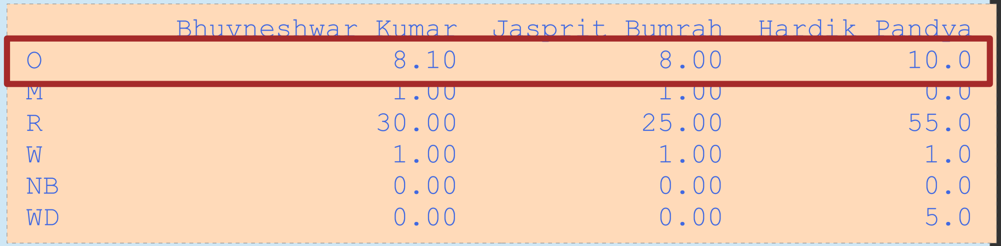
DF: Sorting

DF: Selecting data and copy

DF: Grouping

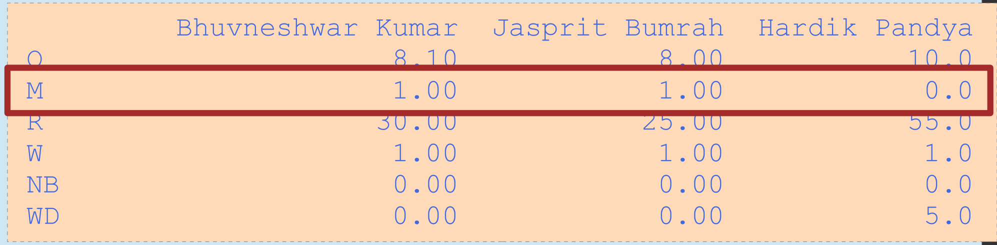
Accessing the data using “loc” and “iloc”:

```
In [98]: df1.loc["O"]

Out[98]:
Bhuvneshwar Kumar     8.1
Jasprit Bumrah        8.0
Hardik Pandya        10.0
Name: O, dtype: float64
```



```
      Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya
O                  8.10            8.00           10.0
M                  1.00            1.00            0.0
R                 30.00           25.00           55.0
W                  1.00            1.00            1.0
NB                 0.00            0.00            0.0
WD                 0.00            0.00            5.0
```

```
In [101]: df1.iloc[1]      #Index location

Out[101]:
Bhuvneshwar Kumar    1.0
Jasprit Bumrah       1.0
Hardik Pandya        0.0
Name: M, dtype: float64
```



```
      Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya
O                  8.10            8.00           10.0
M                  1.00            1.00            0.0
R                 30.00           25.00           55.0
W                  1.00            1.00            1.0
NB                 0.00            0.00            0.0
WD                 0.00            0.00            5.0
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

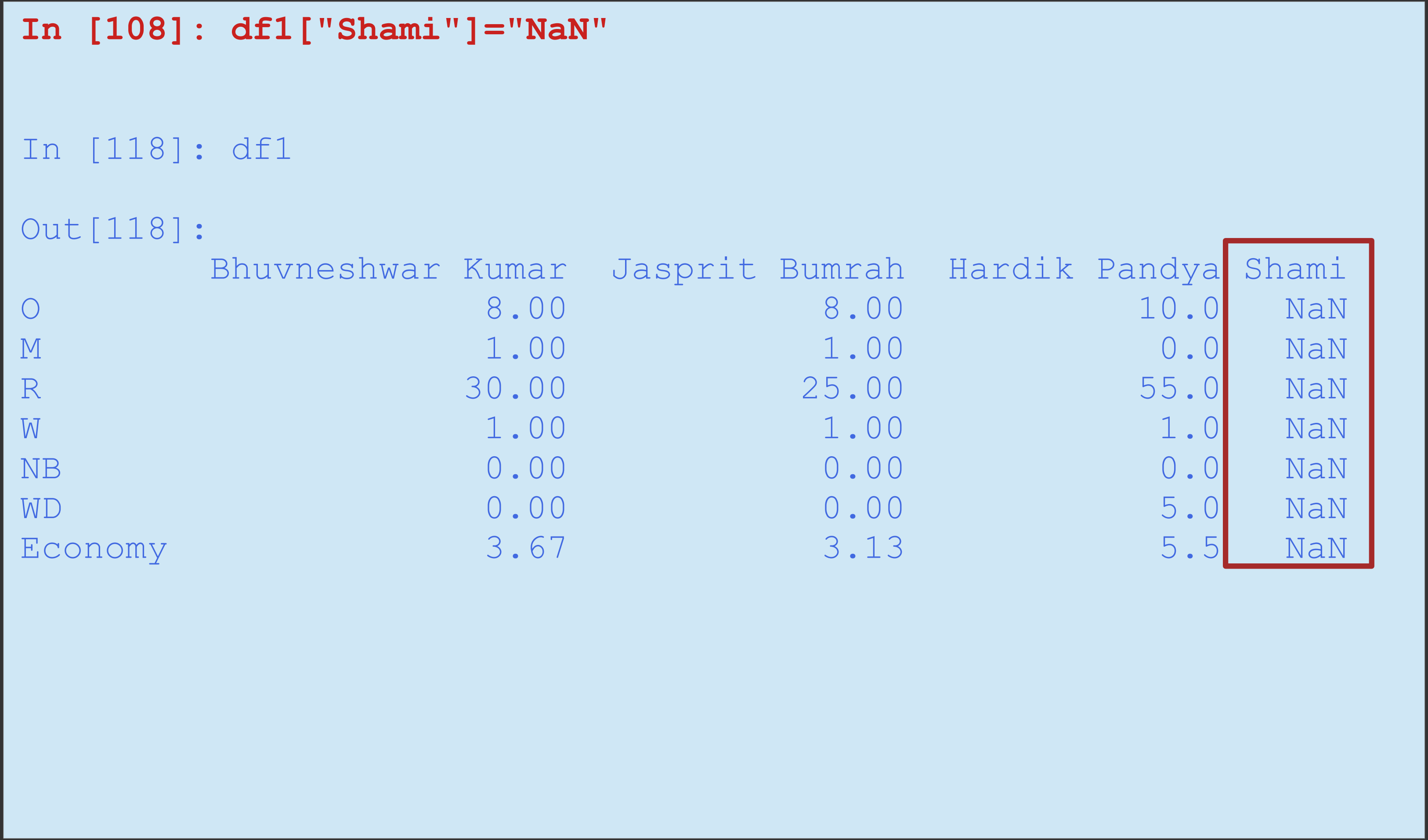
DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Adding the data:



```
In [108]: df1["Shami"]="NaN"


In [118]: df1

Out[118]:
         Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya Shami
O                     8.00            8.00           10.0   NaN
M                     1.00            1.00            0.0   NaN
R                    30.00           25.00           55.0   NaN
W                     1.00            1.00            1.0   NaN
NB                    0.00            0.00            0.0   NaN
WD                    0.00            0.00            5.0   NaN
Economy               3.67            3.13            5.5   NaN
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

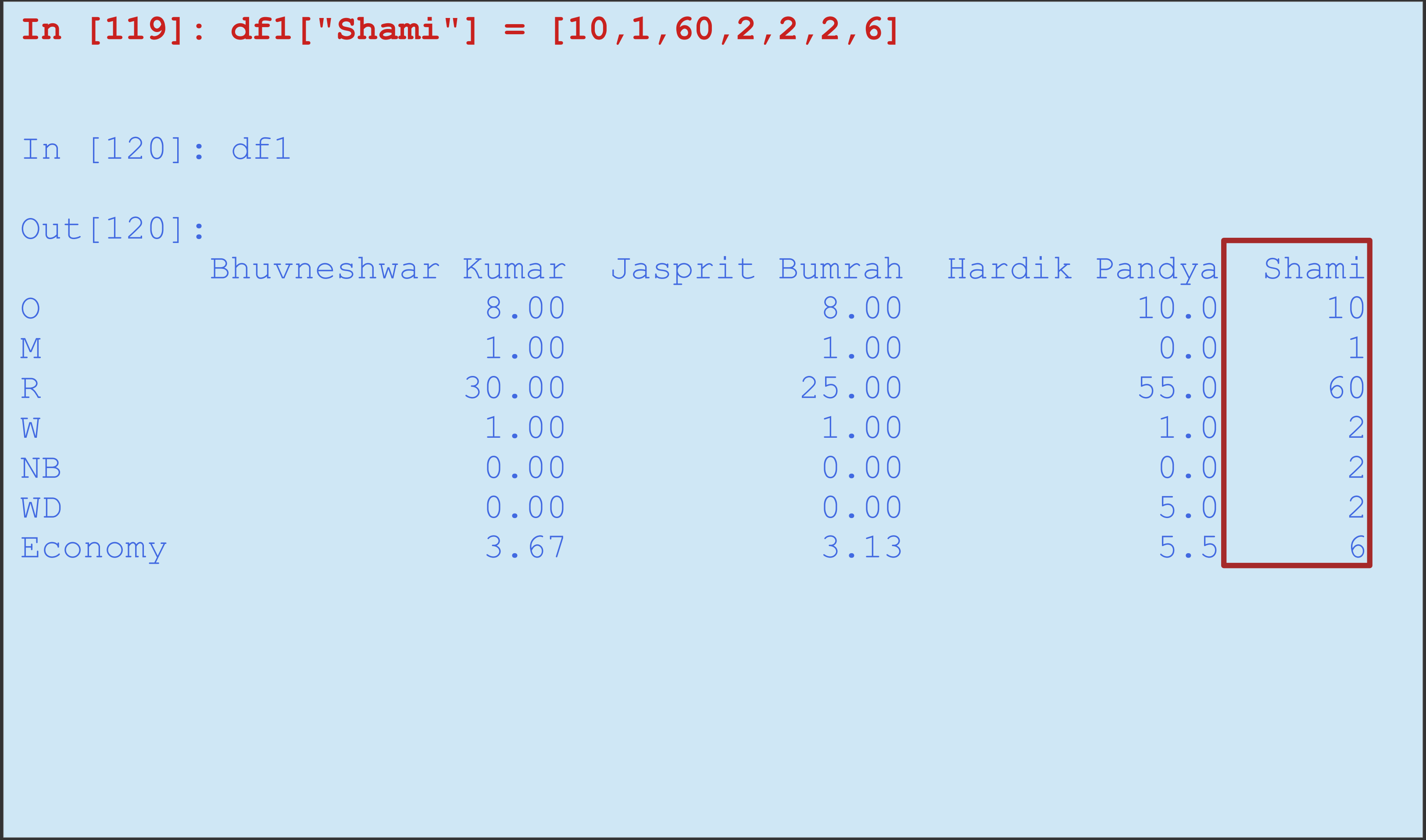
DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Adding the data:



```
In [119]: df1["Shami"] = [10,1,60,2,2,2,6]


In [120]: df1

Out[120]:
         Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya  Shami
O                     8.00            8.00           10.0     10
M                     1.00            1.00            0.0      1
R                    30.00           25.00           55.0     60
W                     1.00            1.00            1.0      2
NB                    0.00            0.00            0.0      2
WD                    0.00            0.00            5.0      2
Economy               3.67            3.13            5.5      6
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

In previous data frames, find the rank and assign in column “Rank”

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Transposing data frame:

```
In [123]: df2 = df1.T


In [124]: df2

Out[124]:
                      O    M     R    W   NB   WD  Economy
Bhuvneshwar Kumar   8.0  1.0  30.0  1.0  0.0  0.0     3.67
Jasprit Bumrah      8.0  1.0  25.0  1.0  0.0  0.0     3.13
Hardik Pandya      10.0  0.0  55.0  1.0  0.0  5.0     5.50
Shami              10.0  1.0  60.0  2.0  2.0  2.0     6.00
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Deleting a column:

```
In [131]: del df3["Bhuvneshwar Kumar"]


In [132]: df3

Out[132]:
         Jasprit Bumrah  Hardik Pandya  Shami
O                  8.00           10.0     10
M                  1.00            0.0      1
R                 25.00           55.0     60
W                  1.00            1.0      2
NB                 0.00            0.0      2
WD                 0.00            5.0      2
Economy            3.13            5.5      6
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Dropping a column:

```
In [134]: df1.drop("O")

Out[134]:
         Bhuvneshwar Kumar  Jasprit Bumrah  Hardik Pandya  Shami
M                     1.00            1.00            0.0      1
R                    30.00           25.00           55.0     60
W                     1.00            1.00            1.0      2
NB                    0.00            0.00            0.0      2
WD                    0.00            0.00            5.0      2
Economy               3.67            3.13            5.5      6



df2.drop(columns="Bhuvneshwar Kumar")
df2.drop(columns="Bhuvneshwar Kumar", inplace=True)
```

Other Operations

```
df3 = df2.T
df2.iloc[3,2] # Accessing index locations
df2.iloc[2:5,]
df2.iloc[:,[0,3]]
df2.loc[:,["Shami"]]

df2.reindex(index=["O","W"])
```

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Merging methods:

- Merge ---- It merges based on column name: To merge, at least one column name must match
- Join ---- Joins based on index
- Concat ---- It appends data frames one after another: No need to match column or index

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

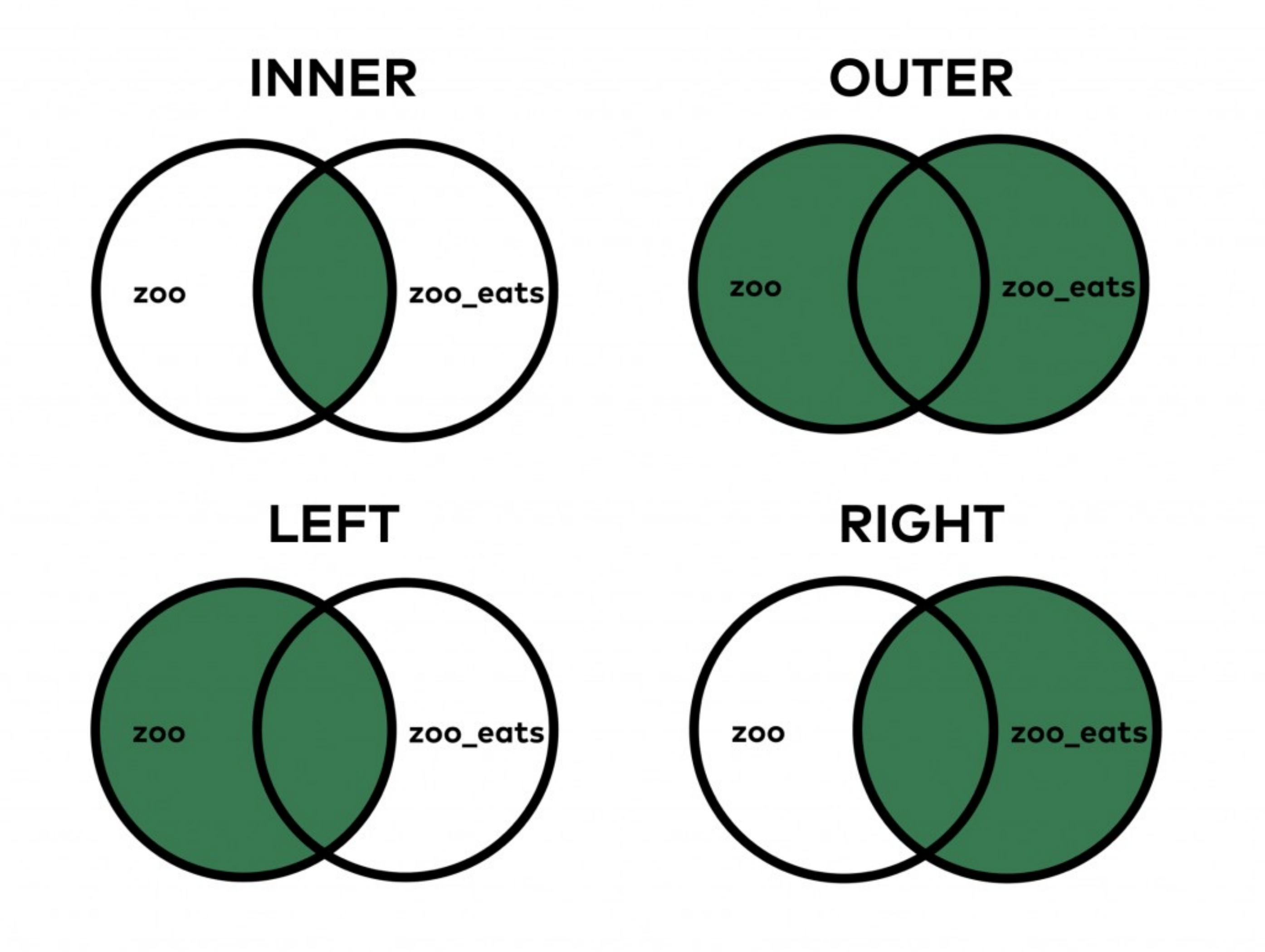
DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping



Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Merging

- Merged based on column name: At least one common column name should be common in both data frames.
- **pd.merge(left_frame,right_frame, how='right', on=’key’)**
    - **how -> ‘left’, ‘right’, ‘in’, ‘out’**

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Joining

- Joins table based on index value:
  - `Table1.join(Table2)`
  - `Table1.join([Table2,Table3])    #Joining multiple tables`

Concatenating

- Joins table based on index value:
  - `pd.concat([left_frame, right_frame])`
  - `pd.concat([left_frame, right_frame], axis=1)`

Introduction

Pandas Series

Pandas Data Frames

DF: Accessing data

DF: Adding data

DF: Dropping/Deleting

DF: Sorting

DF: Selecting data and copy

DF: Grouping

Joining

- Joins table based on index value:
  - `Table1.join(Table2)`
  - `Table1.join([Table2,Table3])    #Joining multiple tables`

Concatinating

- Joins table based on index value:
  - `pd.concat([left_frame, right_frame])`
  - `pd.concat([left_frame, right_frame], axis=1)`