Data Analysis & Visualization

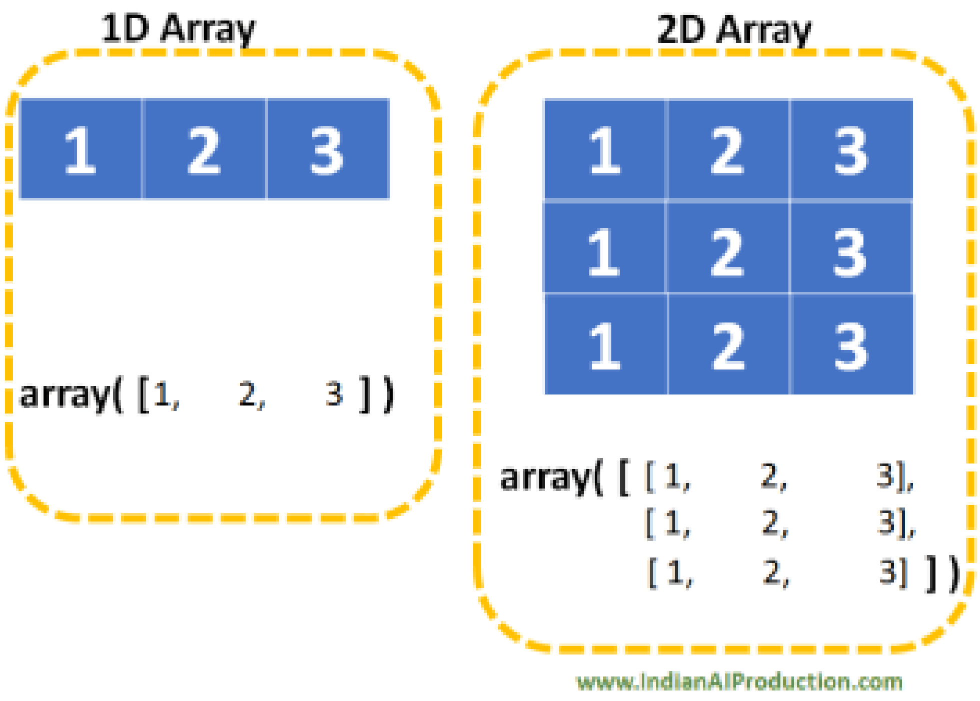
Eman Raslan

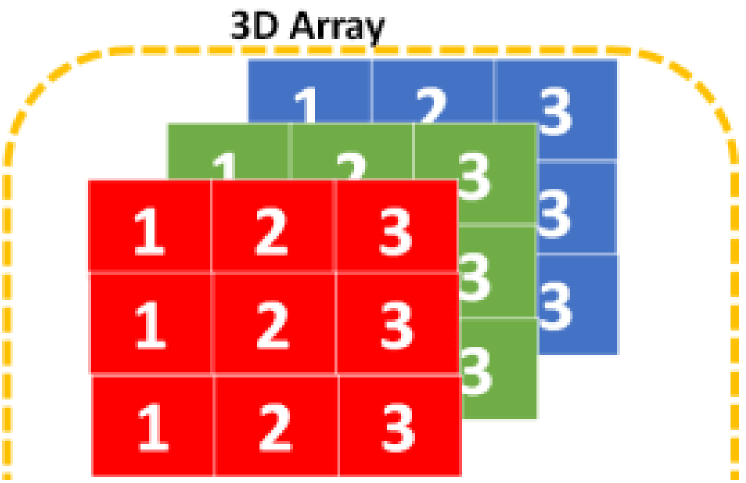
NumPy Basics

NumPy Agenda

- NumPy Intro
- Creating Arrays
- NumPy Array Indexing
- NumPy Array Slicing
- NumPy Data Types
- NumPy Copy vs View
- NumPy Array Shape
- NumPy Array Reshape
- NumPy Array Join
- NumPy Array Sort
- NumPy Array Filter

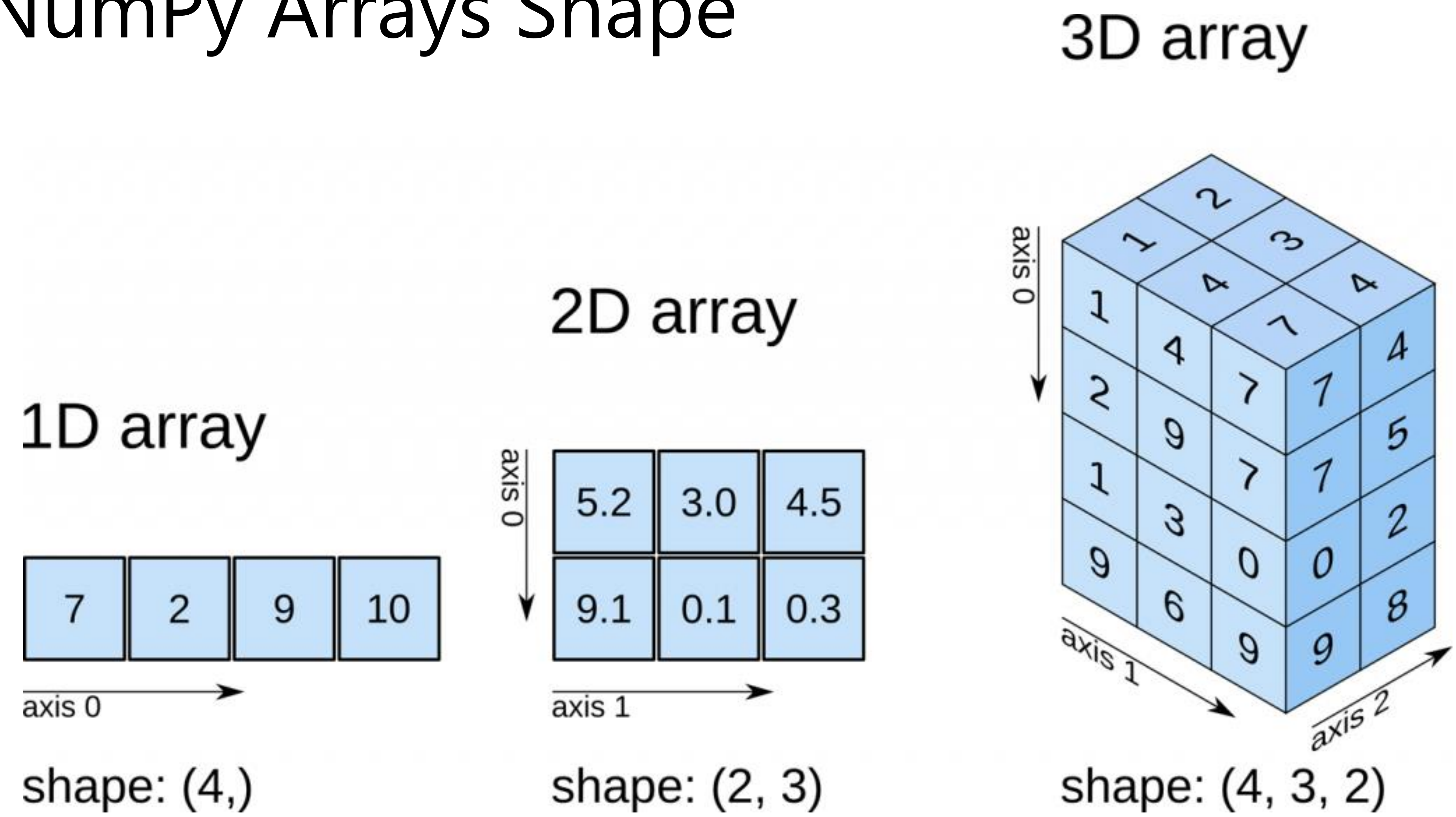
Creating NumPy Arrays



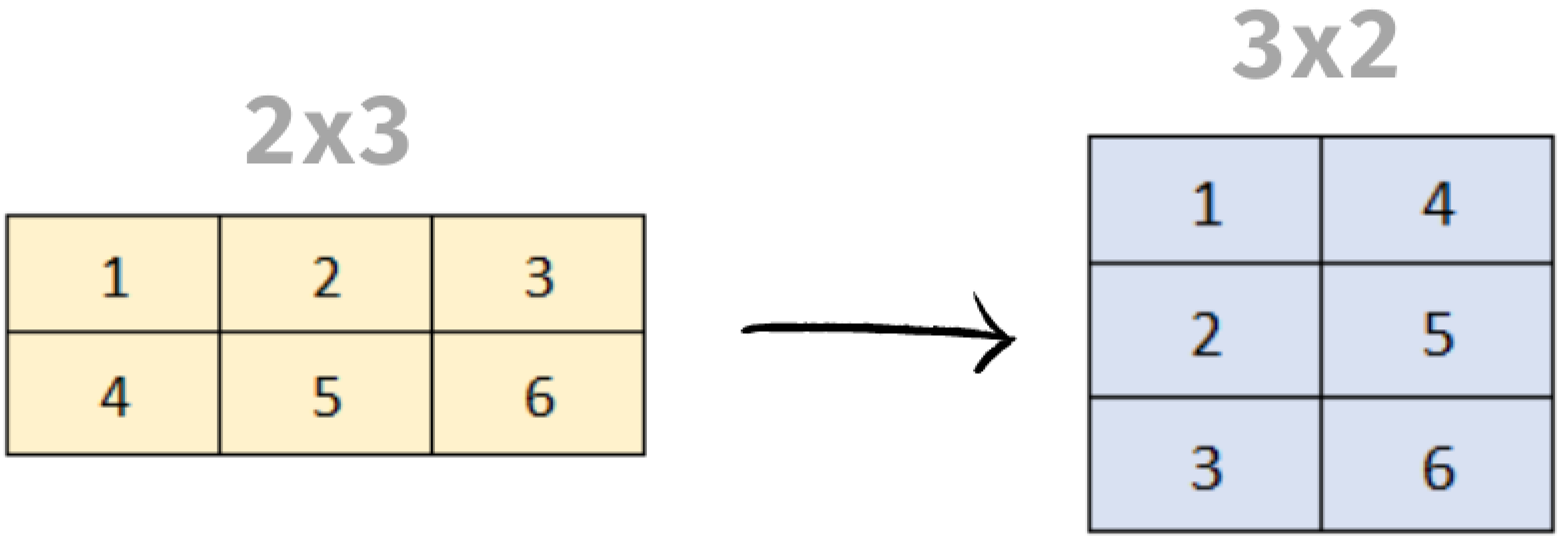



```
array([[[1, 2, 3],
[1, 2, 3],
[1, 2, 3]],
[[1, 2, 3],
[1, 2, 3],
[1, 2, 3]],
[[1, 2, 3],
[1, 2, 3],
[1, 2, 3]]])
```

NumPy Arrays Shape

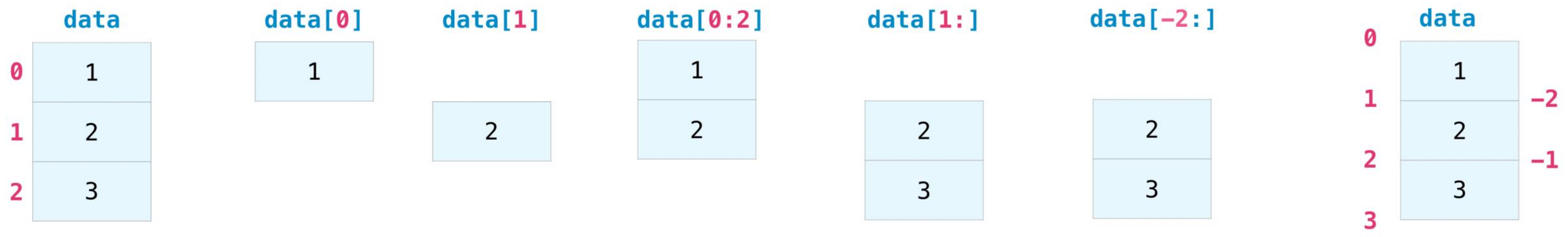


NumPy Arrays Transpose

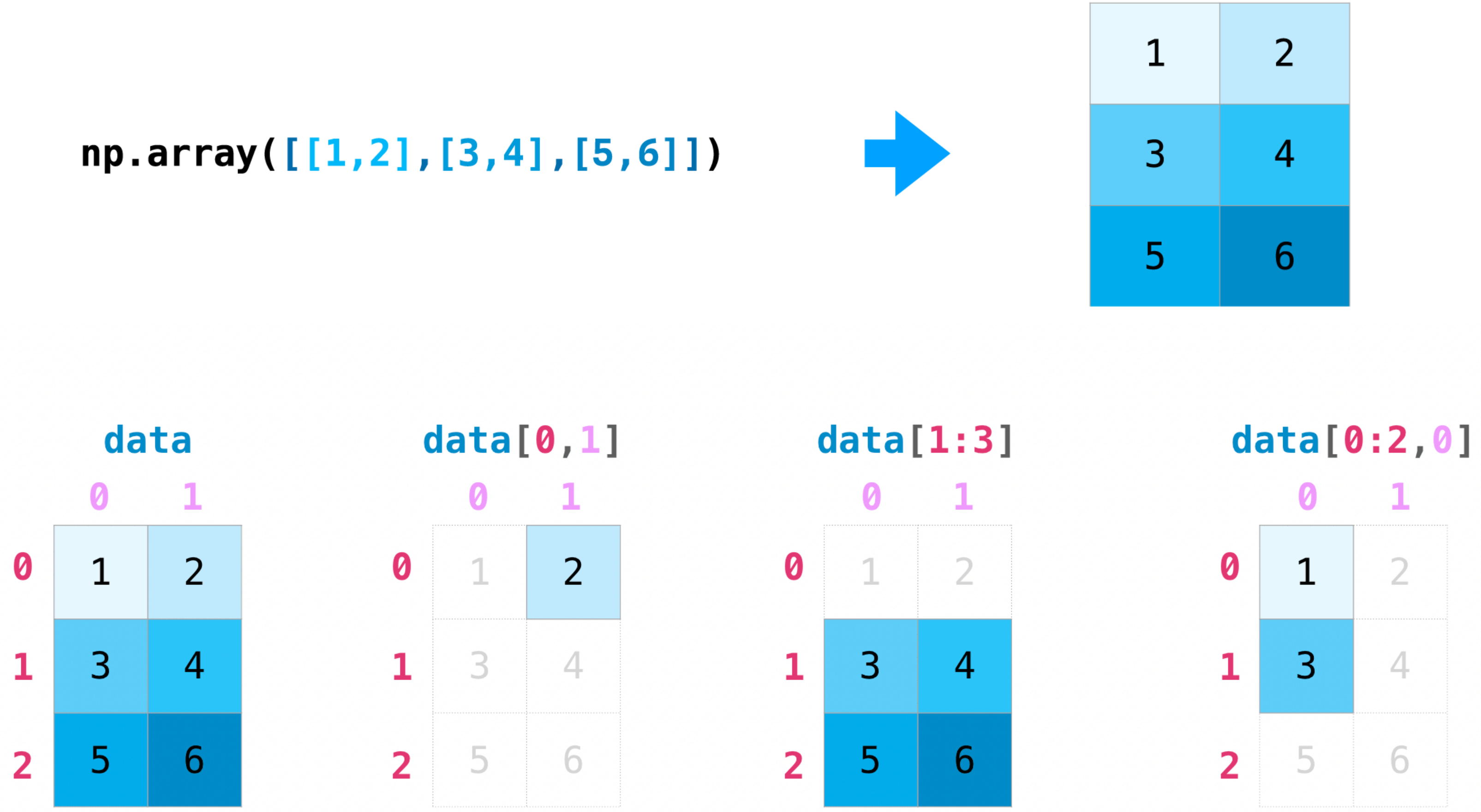


Transpose array

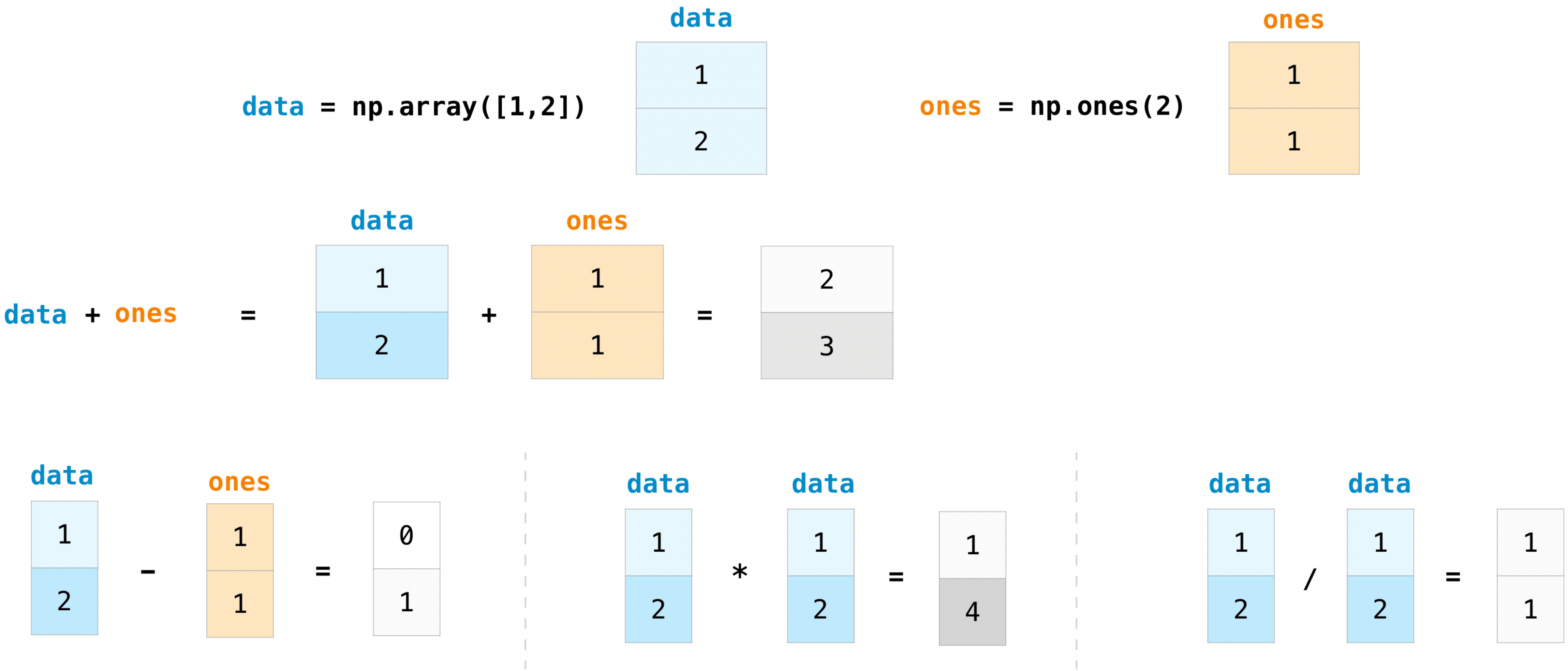
NumPy Array Indexing



NumPy Array Indexing



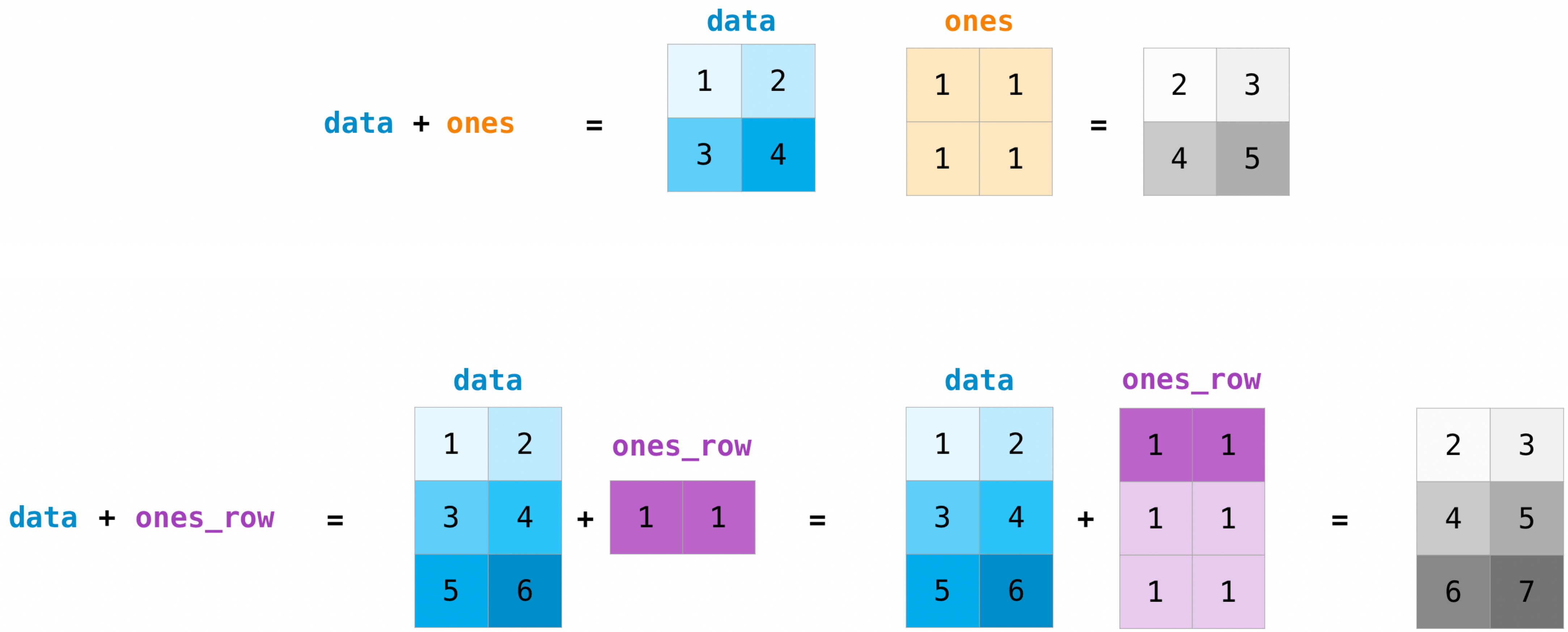
Basic array operations



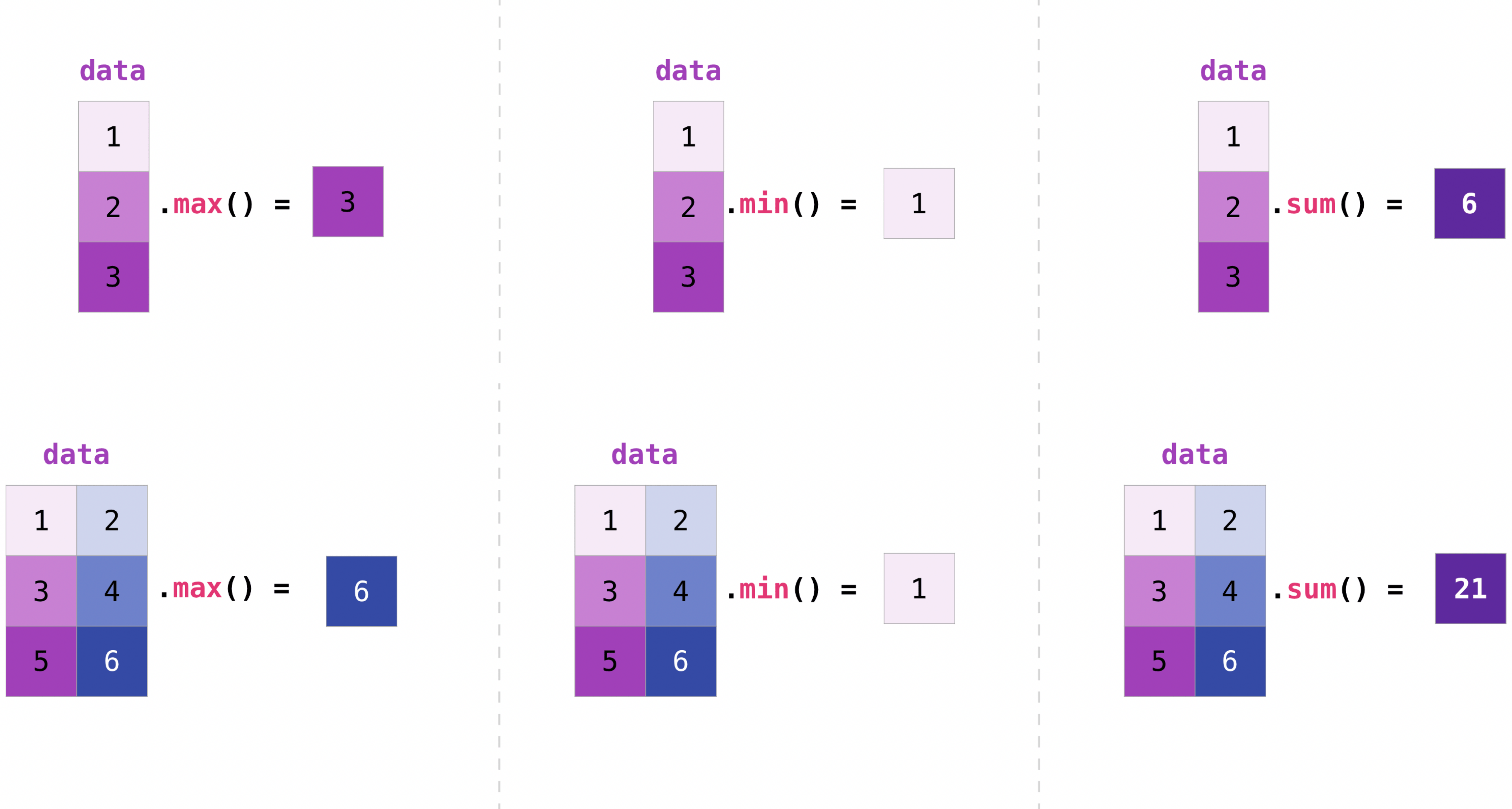
Broadcasting

$$\begin{bmatrix}1\\2\end{bmatrix} * 1.6 = \begin{bmatrix}1\\2\end{bmatrix} * \begin{bmatrix}1.6\\1.6\end{bmatrix} = \begin{bmatrix}1.6\\3.2\end{bmatrix}$$

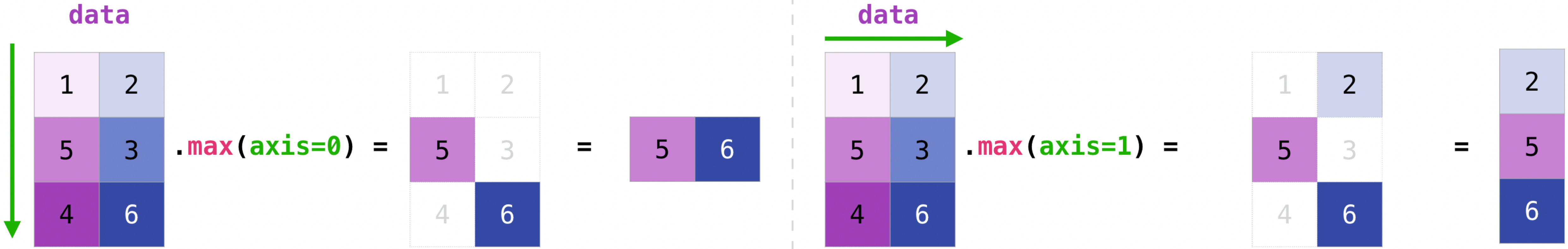
Basic array operations



More useful array operations



More useful array operations



Creating NumPy Array

`np.ones(3)`

1
1
1

`np.zeros(3)`

0
0
0

`np.random.random(3)`

0.5967
0.0606
0.2223

`np.ones((3,2))`

3

1	1
1	1
1	1

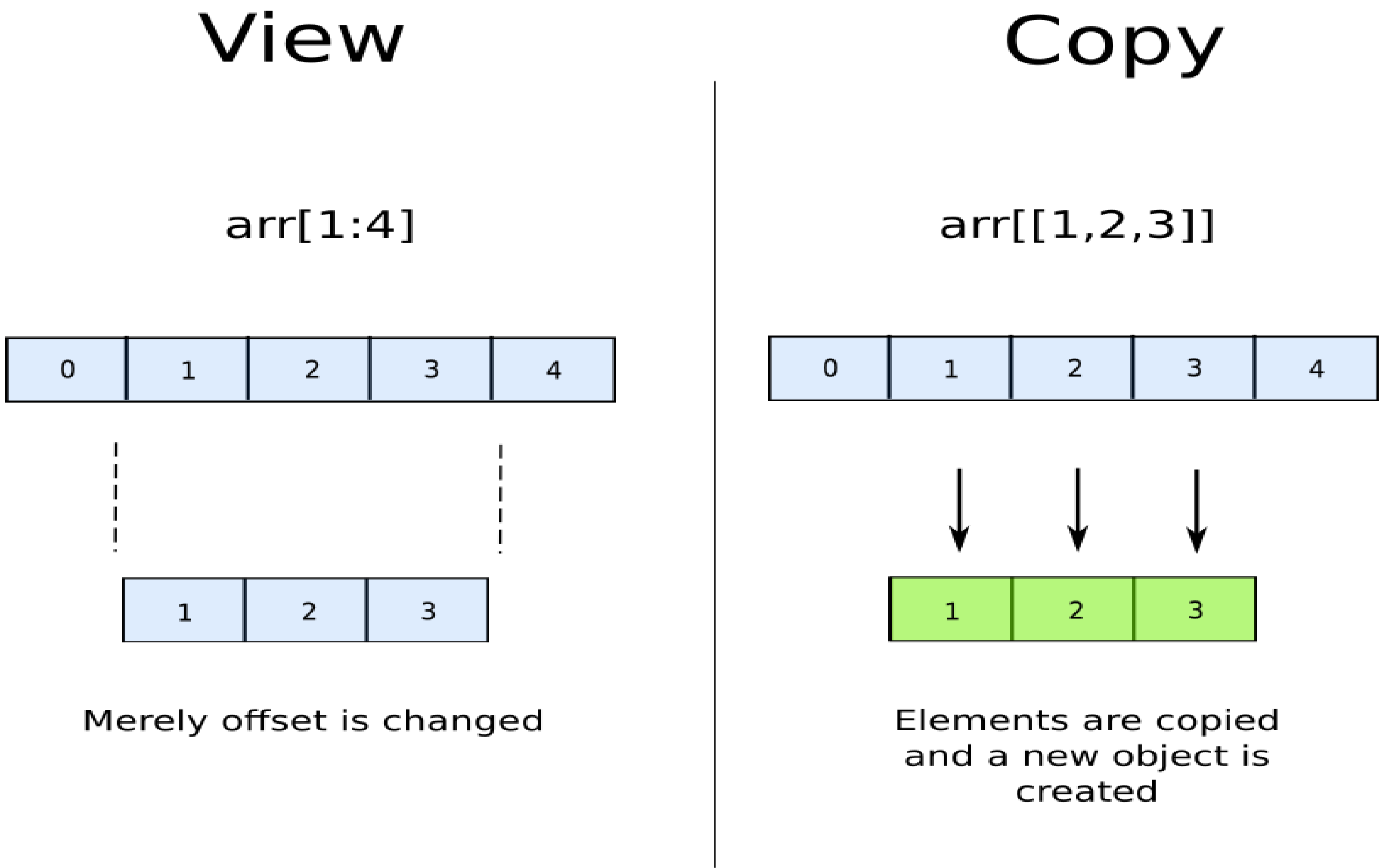
`np.zeros((3,2))`

0	0
0	0
0	0

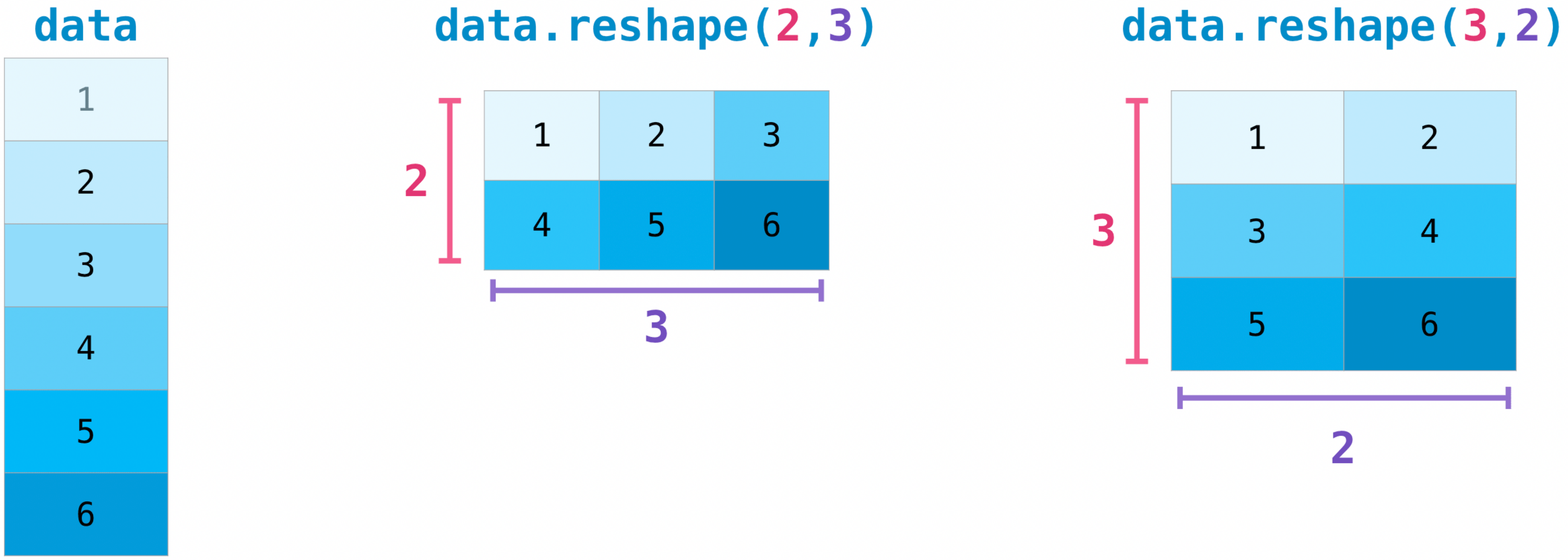
`np.random.random((3,2))`

0.37	0.88
0.75	0.79
0.63	0.16

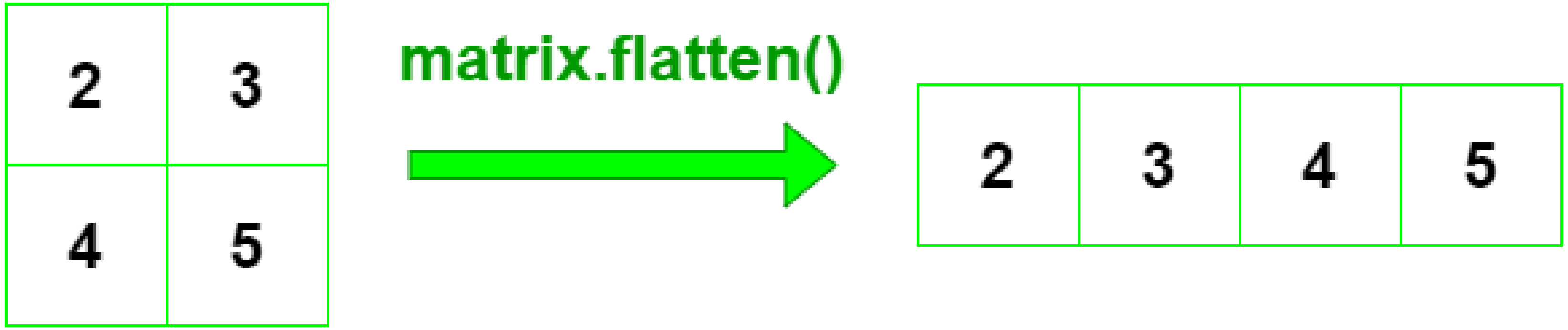
NumPy Copy vs View



NumPy Array Reshape



flattening multidimensional arrays



Working with mathematical formulas

$$MeanSquareError = \frac{1}{n}\sum_{i=1}^{n}(Y\_prediction_i - Y_i)^2$$

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

$$\texttt{error = (1/3) * np.sum(np.square(} \begin{array}{c}\texttt{predictions}\\ \begin{bmatrix}1\\1\\1\end{bmatrix}\end{array} - \begin{array}{c}\texttt{labels}\\ \begin{bmatrix}1\\2\\3\end{bmatrix}\end{array} \texttt{))}$$

Working with mathematical formulas

```
error = (1/3) * np.sum(np.square(    ))
```

0
-1
-2

```
error = (1/3) * np.sum(    )
```

0
1
4

```
error = (1/3) *    
```

5

Pandas Basics

Pandas

- Pandas is an open source library built on top of NumPy
- It allows for fast analysis and data cleaning and preparation
- It excels in performance and productivity.
- It also has built-in visualization features.
- It can work with data from a wide variety of sources.

Pandas Agenda

- Pandas Intro
- Pandas Series
- Pandas DataFrames
- Pandas Read CSV
- Pandas Analyzing Data
- Cleaning Data
- Cleaning Empty Cells
- Cleaning Wrong Format
- Cleaning Wrong Data
- Removing Duplicates
- Pandas Correlations
- Pandas Plotting
- Merging, joining, and concatenating
- Operations
- Apply function
- Data input and output

Pandas Agenda

Basic

- Introduction
- Getting Started
- Pandas Series
- DataFrames
- Read CSV
- Read JSON
- Analyze Data

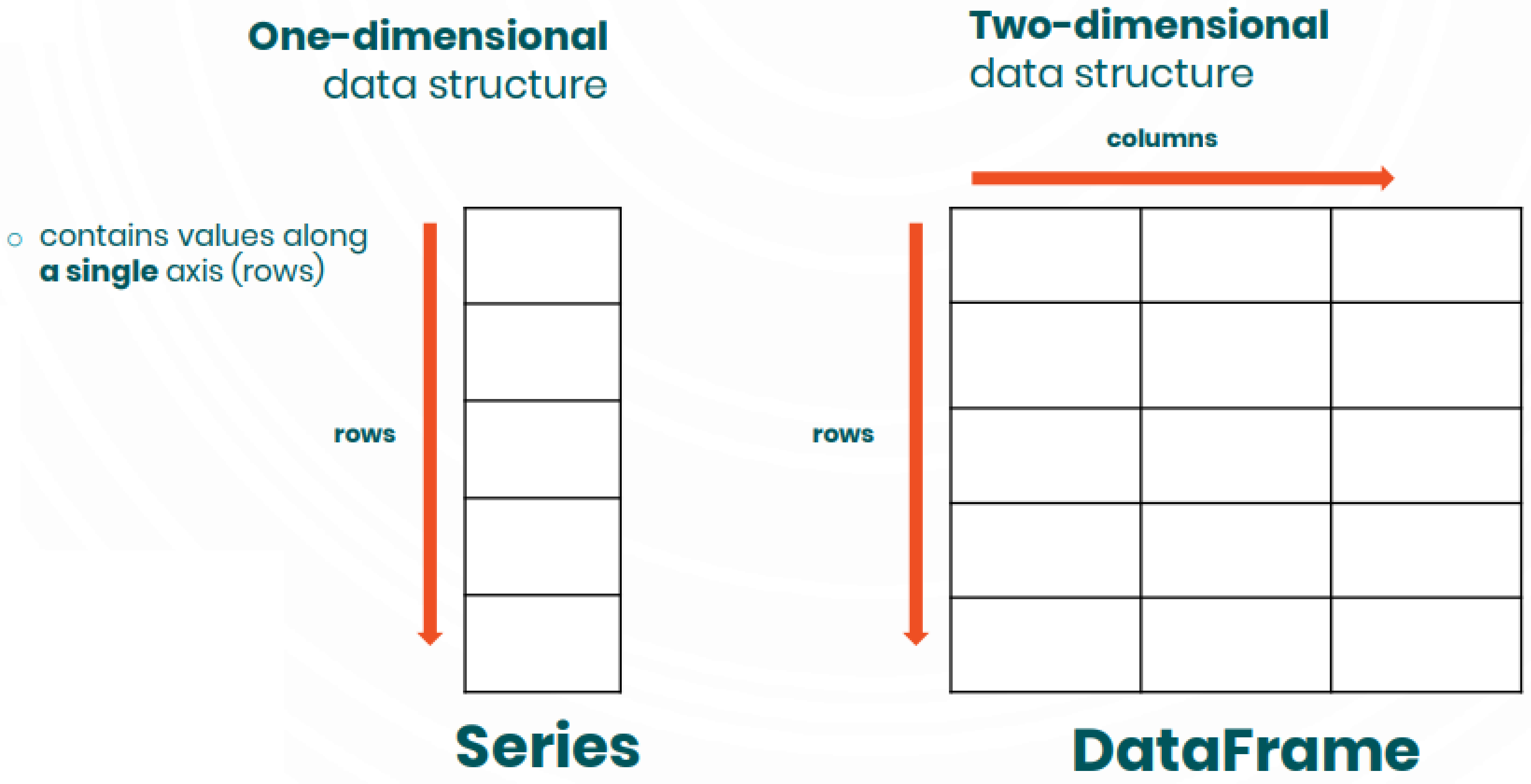
Cleaning Data

- Clean Data
- Clean Empty Cells
- Clean Wrong Format
- Clean Wrong Data
- Remove Duplicates

Advanced

- Correlations
- Plotting

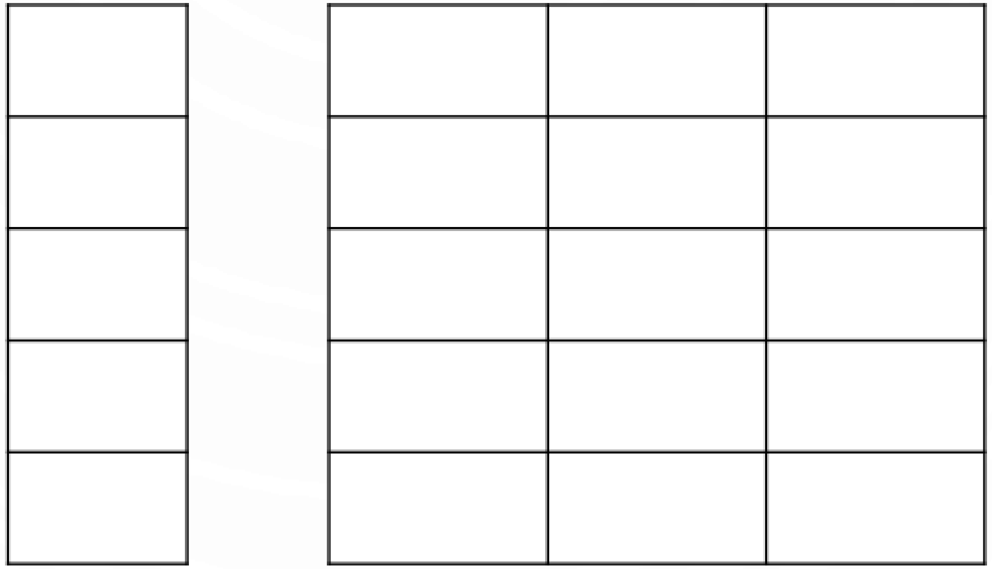
Pandas Data Structure



Pandas Data Structure

Single column data

- corresponds to a **single variable**
- information of a **single type**



Series **DataFrame**

Multi-column data

- each column represents a **different** variable
- every column contains data of **its own type**
- the information can potentially **be heterogeneous**

Pandas Data Structure

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0
6	Evan Turner	Boston Celtics	11.0

```
ser = pd.Series (df [ 'Name'])
ser = pd.Series (df [ 'Team'])
ser = pd.Series (df [ 'Number'])
```

Pandas Data Structure

Column names

Columns
axis=1

Index label

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

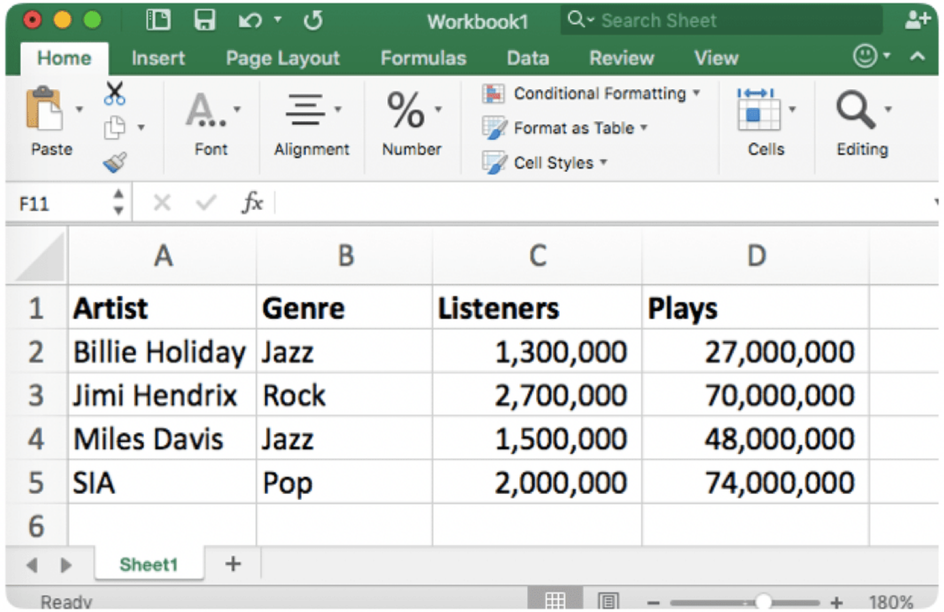
Index
axis=0

Missing value

Data

Reading files

`music.csv`

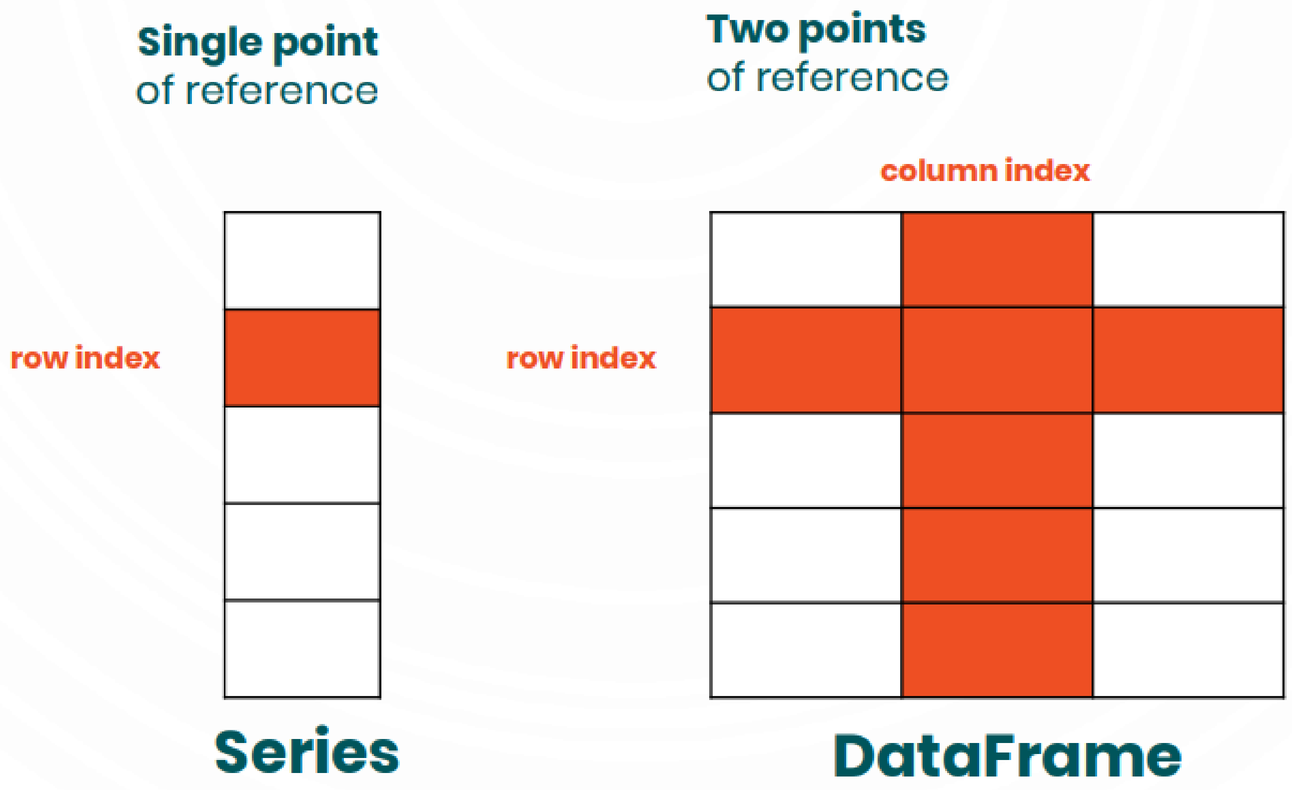


	A	B	C	D
1	**Artist**	**Genre**	**Listeners**	**Plays**
2	Billie Holiday	Jazz	1,300,000	27,000,000
3	Jimi Hendrix	Rock	2,700,000	70,000,000
4	Miles Davis	Jazz	1,500,000	48,000,000
5	SIA	Pop	2,000,000	74,000,000
6				

`pandas.read_csv('music.csv')`

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

Indexing



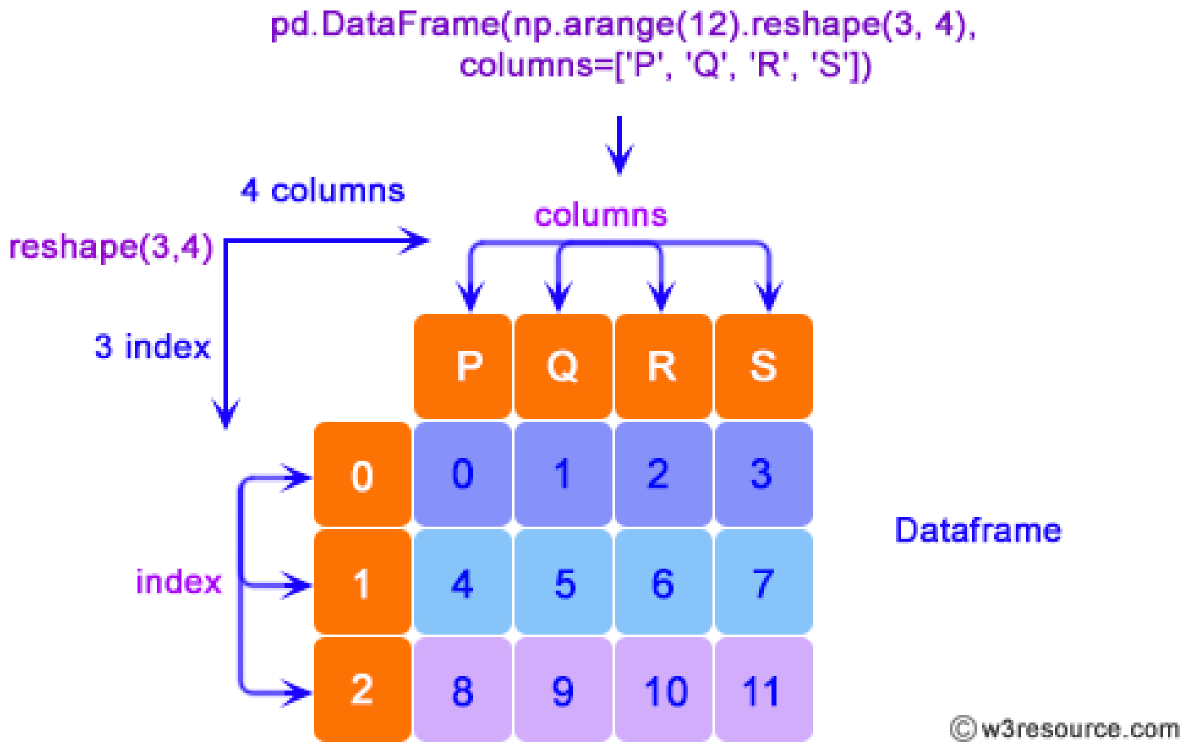
Single point
of reference
Two points
of reference
column index
row index
row index
Series
DataFrame

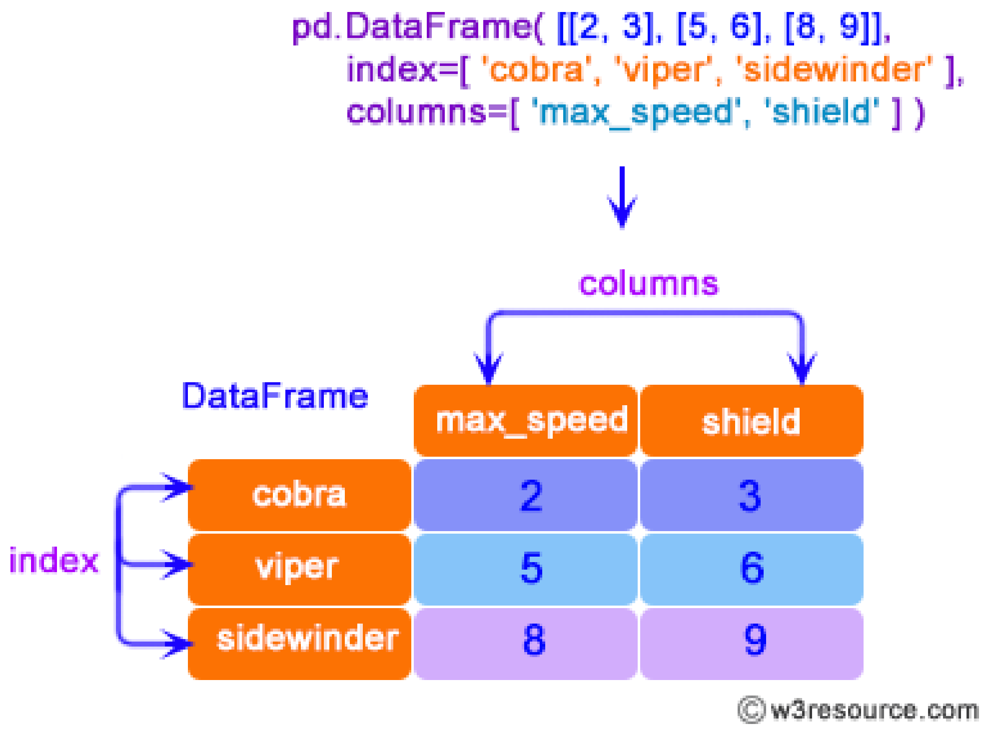
Pandas Data Structure

	index	Column-1	Column-2	...	Column-n
Row-1	0			...	
Row-2	1			...	
	...	...	...	...	...
				...	
Row-L	L			...	

DataFrame

Creating a DataFrame





Creating a DataFrame

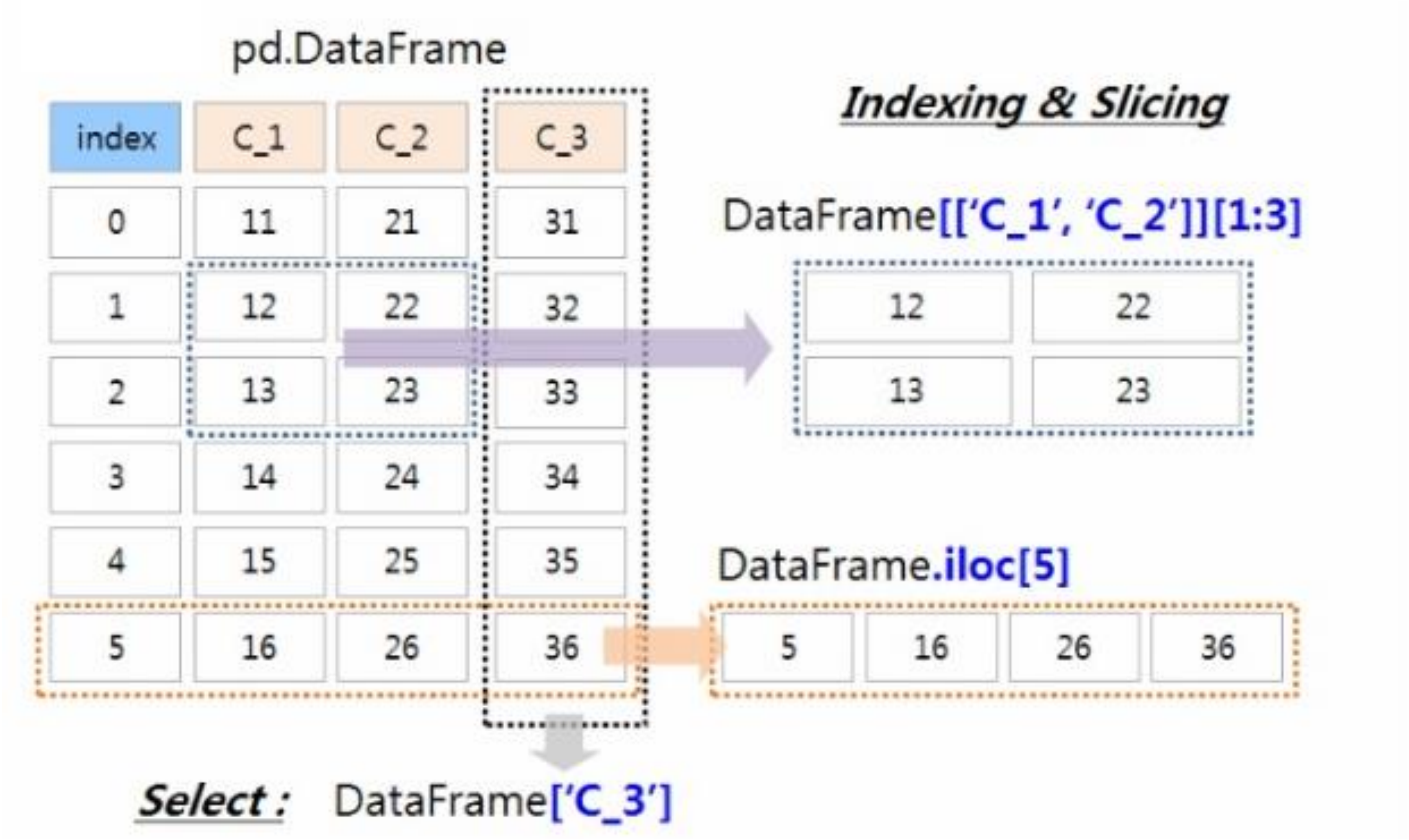
```
# Named Index
fruit = {
    'oranges' : [3,2,0,1],
    'apples' : [0,3,7,2],
    'grapes' : [5,6,9,0],
    'pear'   : [1,23,45,1]
}
df = pd.DataFrame(fruit ,index = ['June','July','August','September'])
df
```

	oranges	apples	grapes	pear
June	3	0	5	1
July	2	3	6	23
August	0	7	9	45
September	1	2	0	1

Indexing and slicing

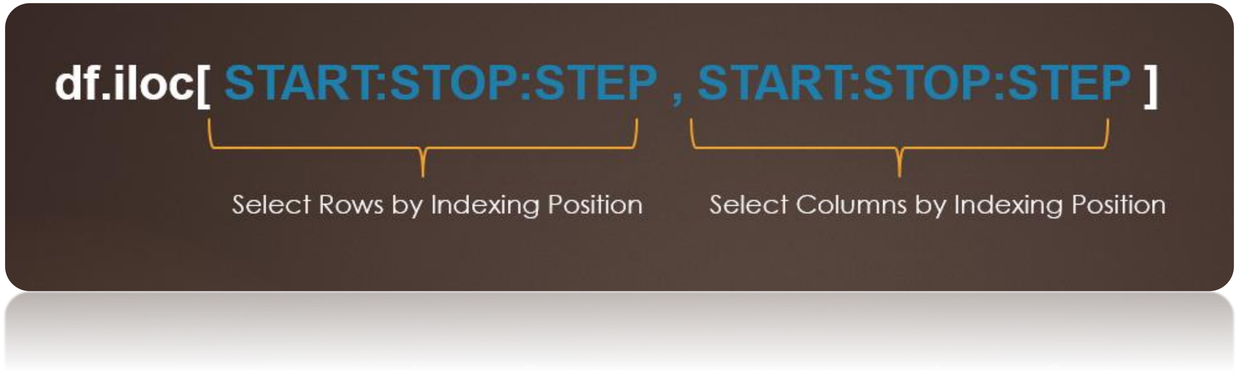
Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

Indexing and slicing

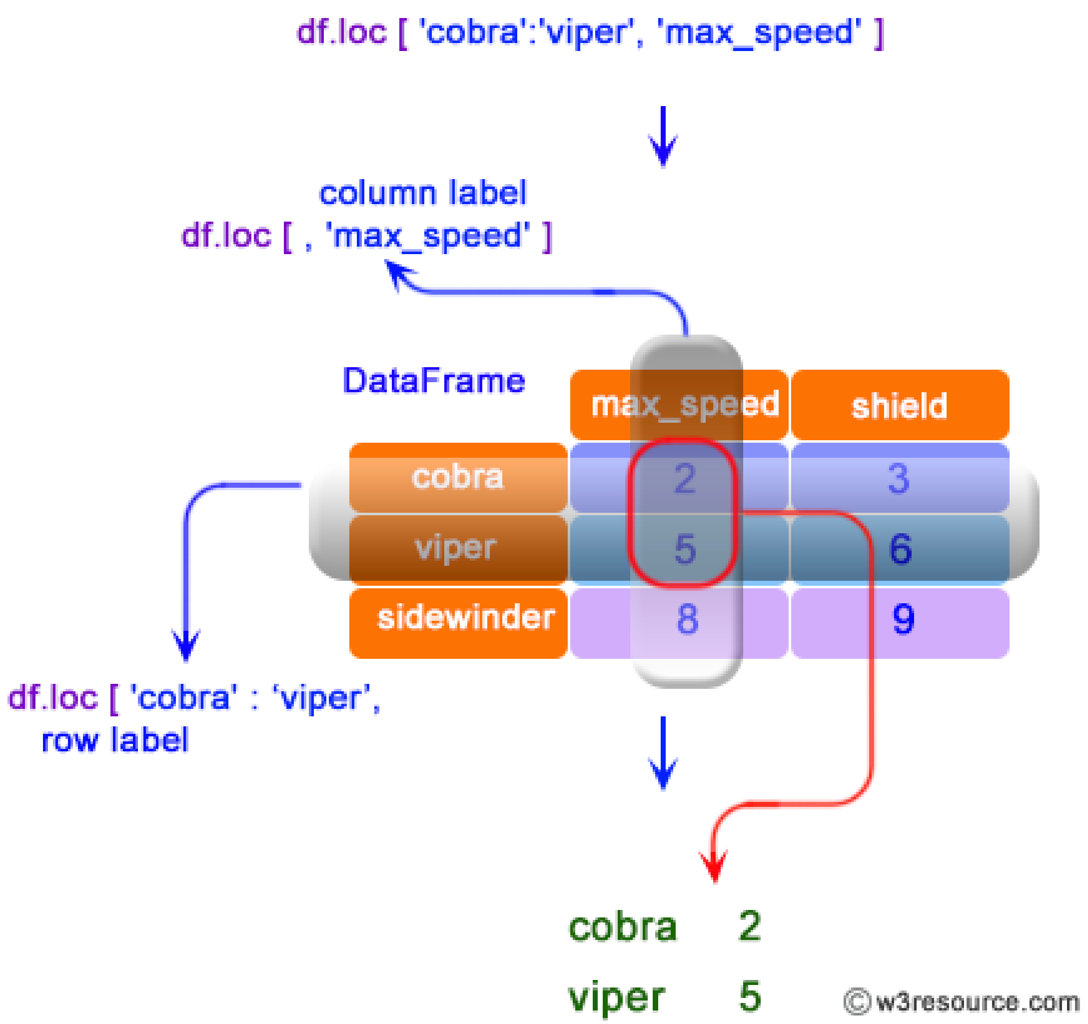


loc Vs. iloc

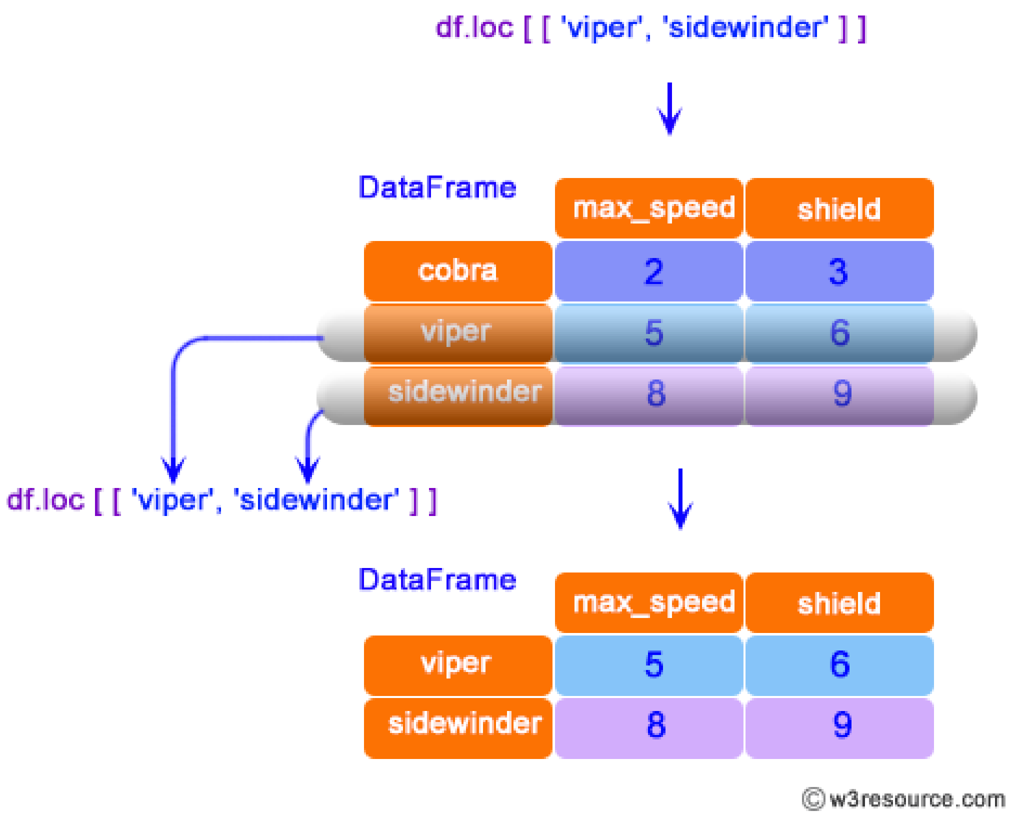

df.loc[START:STOP:STEP , START:STOP:STEP]
Select Rows by Names/Labels
Select Columns by Names/Labels


df.iloc[START:STOP:STEP , START:STOP:STEP]
Select Rows by Indexing Position
Select Columns by Indexing Position

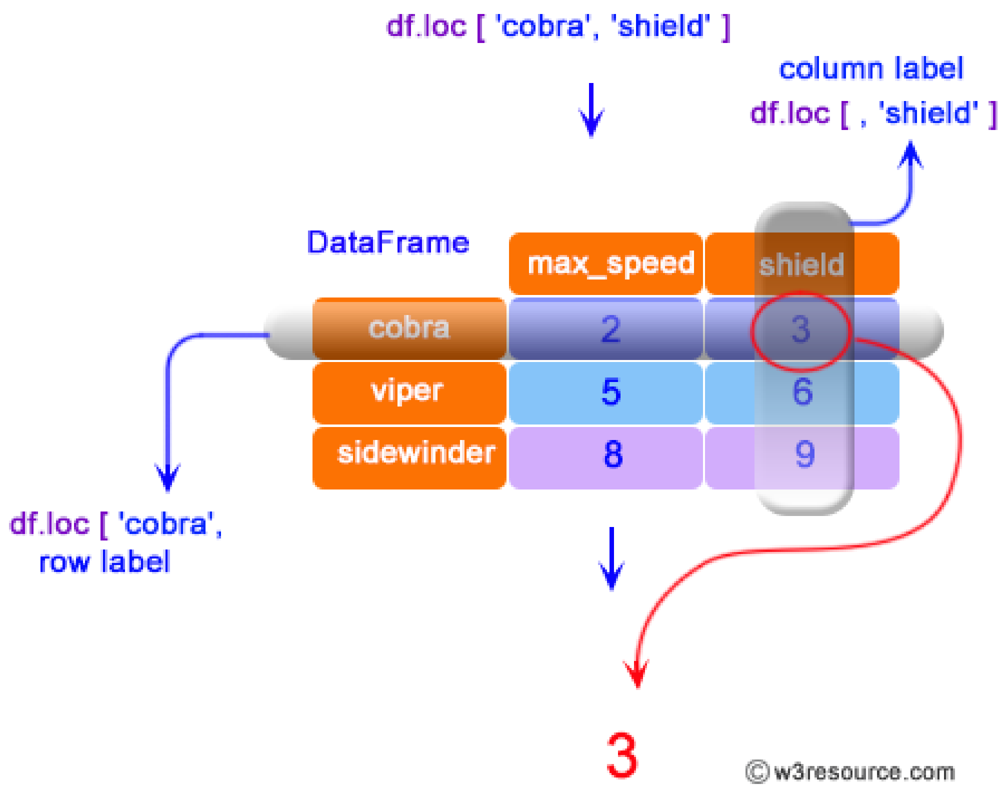
loc Vs. iloc


df.loc ['cobra':'viper', 'max_speed']
column label
df.loc [, 'max_speed']
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
df.loc ['cobra' : 'viper',
row label
cobra 2
viper 5
© w3resource.com

loc Vs. iloc

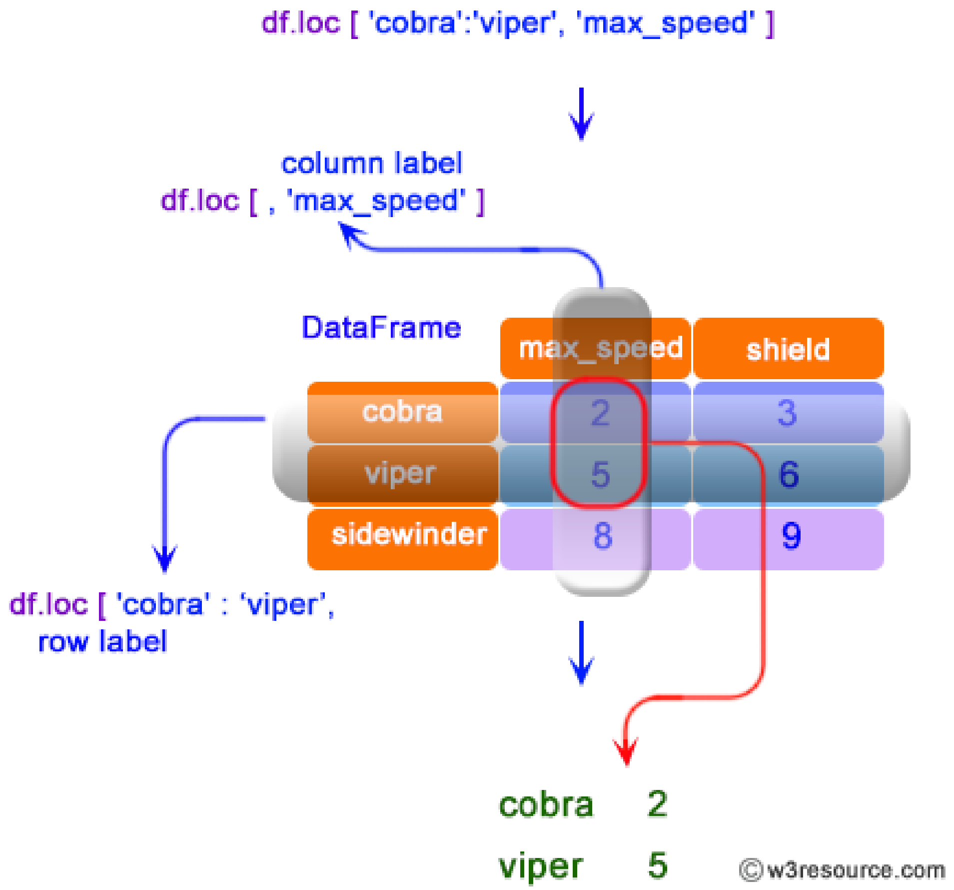


df.loc [['viper', 'sidewinder']]
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
df.loc [['viper', 'sidewinder']]
DataFrame
max_speed
shield
viper
5
6
sidewinder
8
9
© w3resource.com

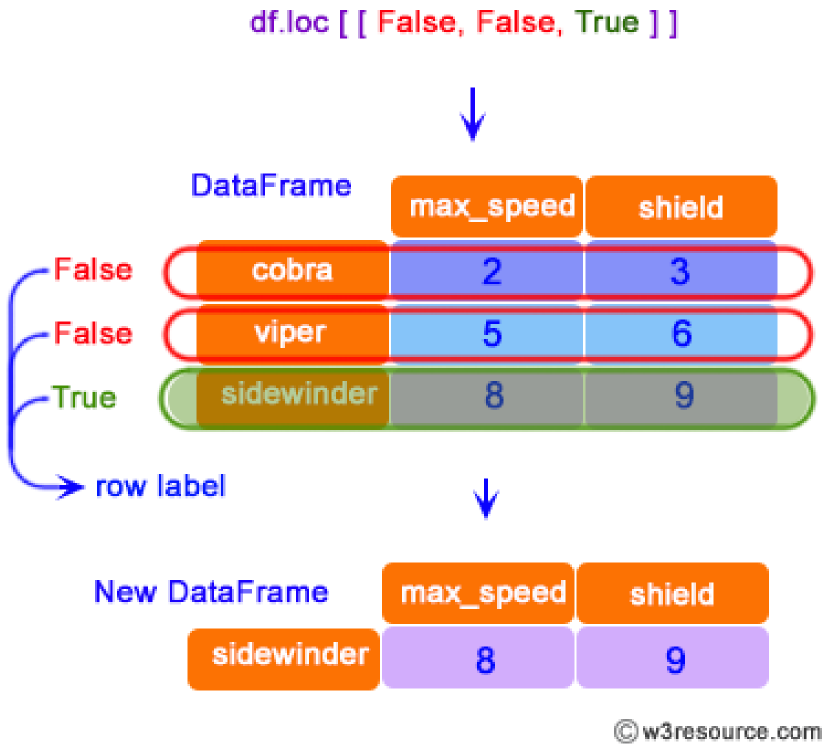


df.loc ['cobra', 'shield']
column label
df.loc [, 'shield']
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
df.loc ['cobra',
row label
3
© w3resource.com

loc Vs. iloc

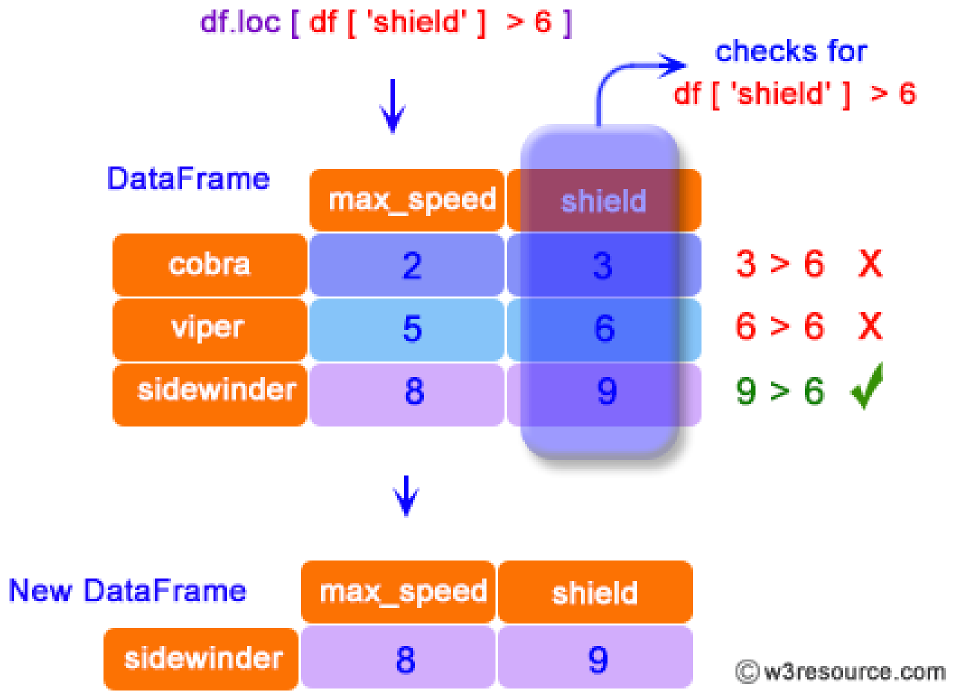


df.loc ['cobra':'viper', 'max_speed']
column label
df.loc [, 'max_speed']
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
df.loc ['cobra' : 'viper',
row label
cobra 2
viper 5
©w3resource.com

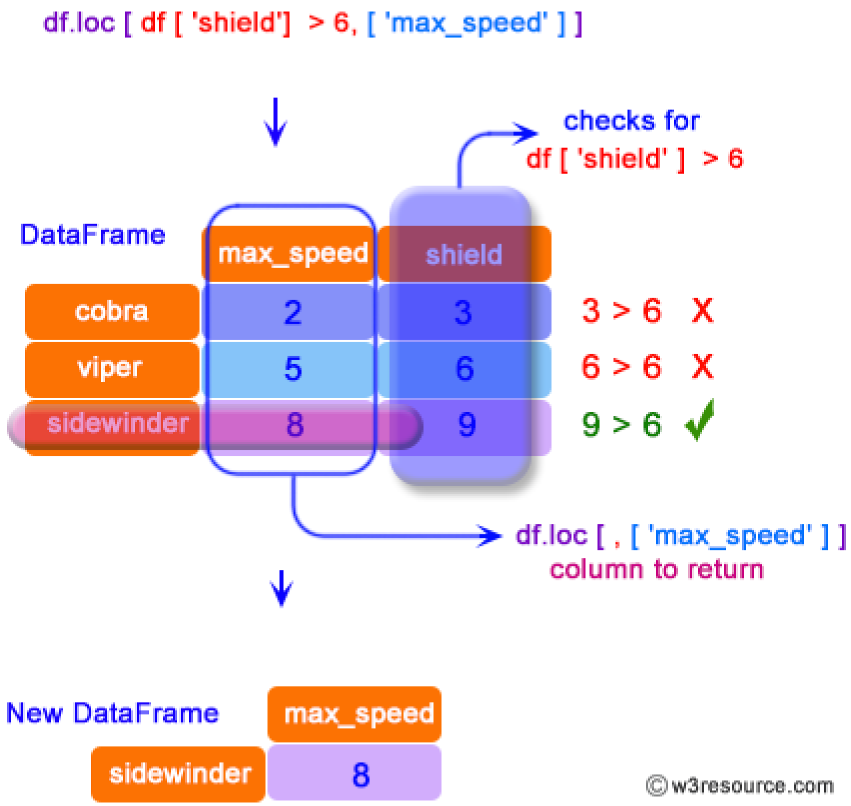


df.loc [[False, False, True]]
DataFrame
max_speed
shield
False
cobra
2
3
False
viper
5
6
True
sidewinder
8
9
row label
New DataFrame
max_speed
shield
sidewinder
8
9
©w3resource.com

loc Vs. iloc

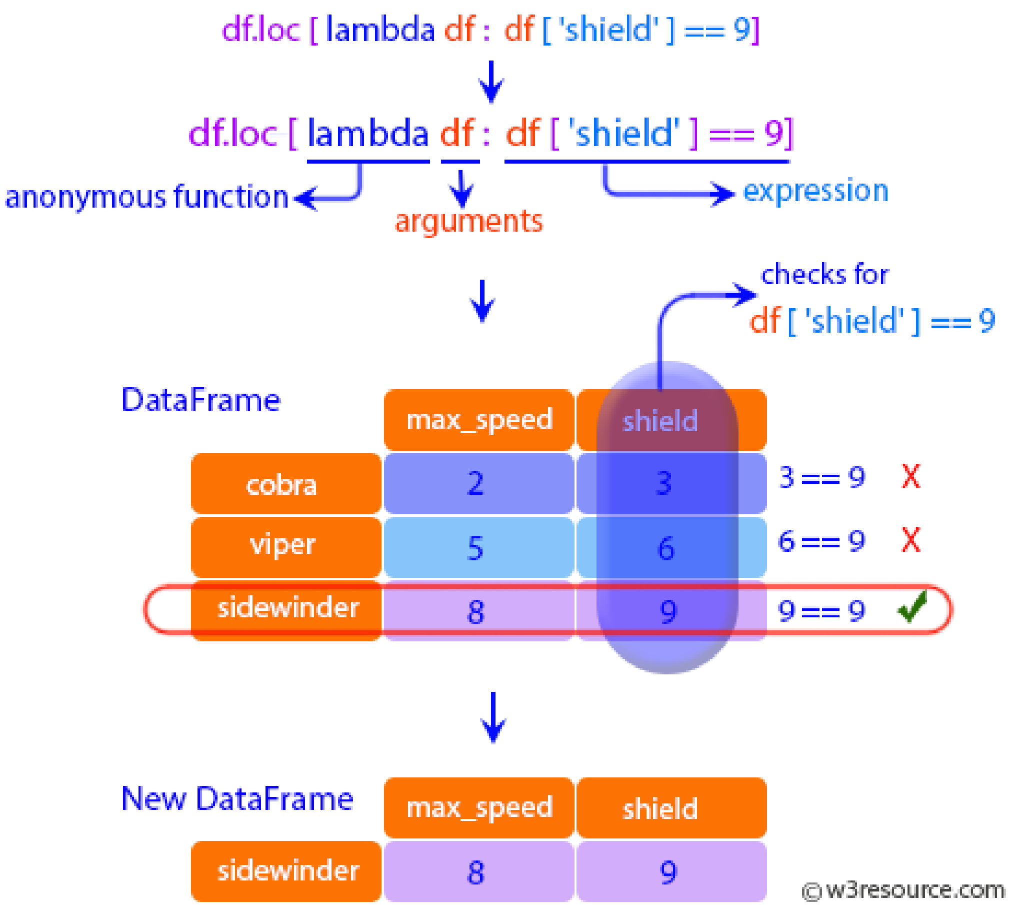


df.loc [df ['shield'] > 6]
checks for
df ['shield'] > 6
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
3 > 6 X
6 > 6 X
9 > 6
New DataFrame
max_speed
shield
sidewinder
8
9
©w3resource.com



df.loc [df ['shield'] > 6, ['max_speed']]
checks for
df ['shield'] > 6
DataFrame
max_speed
shield
cobra
2
3
viper
5
6
sidewinder
8
9
3 > 6 X
6 > 6 X
9 > 6
df.loc [, ['max_speed']]
column to return
New DataFrame
max_speed
sidewinder
8
©w3resource.com

loc Vs. iloc



df.loc [lambda df : df ['shield'] == 9]
df.loc [lambda df : df ['shield'] == 9]
anonymous function
arguments
expression
checks for
df ['shield'] == 9
DataFrame
max_speed
shield
cobra
2
3
3 == 9
X
viper
5
6
6 == 9
X
sidewinder
8
9
9 == 9
New DataFrame
max_speed
shield
sidewinder
8
9
© w3resource.com

Analyzing DataFrames

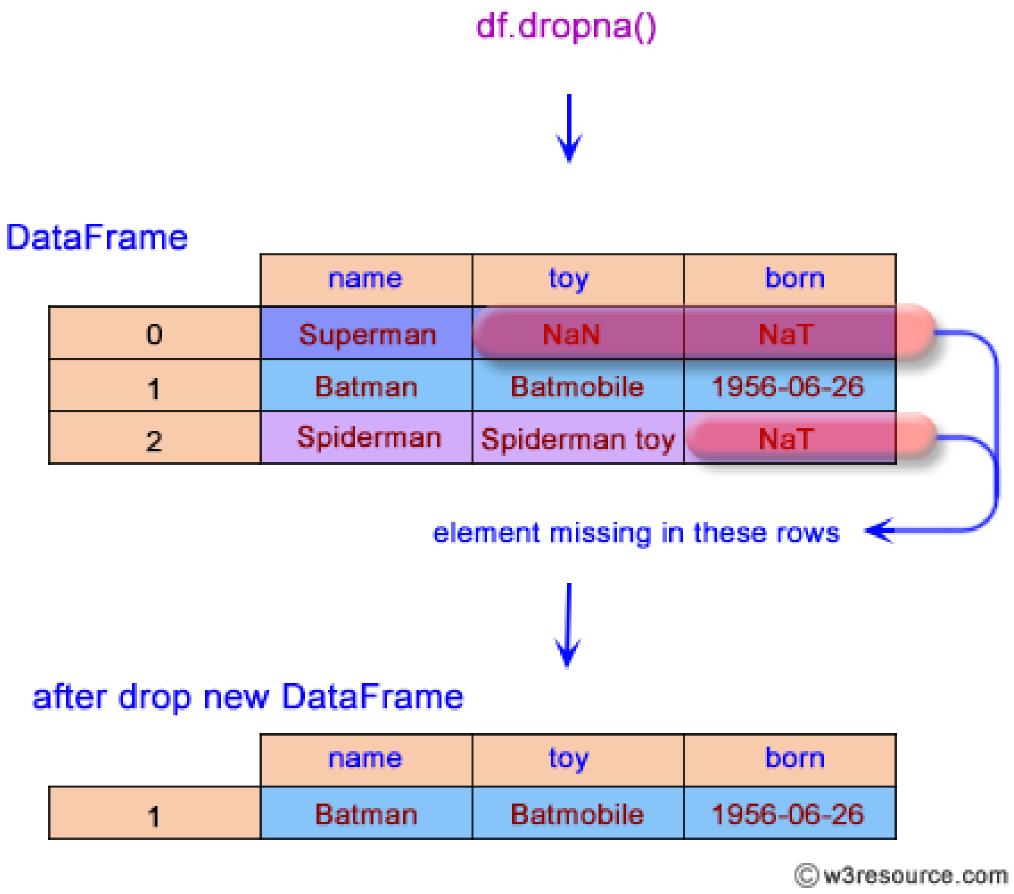
- head()
- tail()
- info()
- describe()

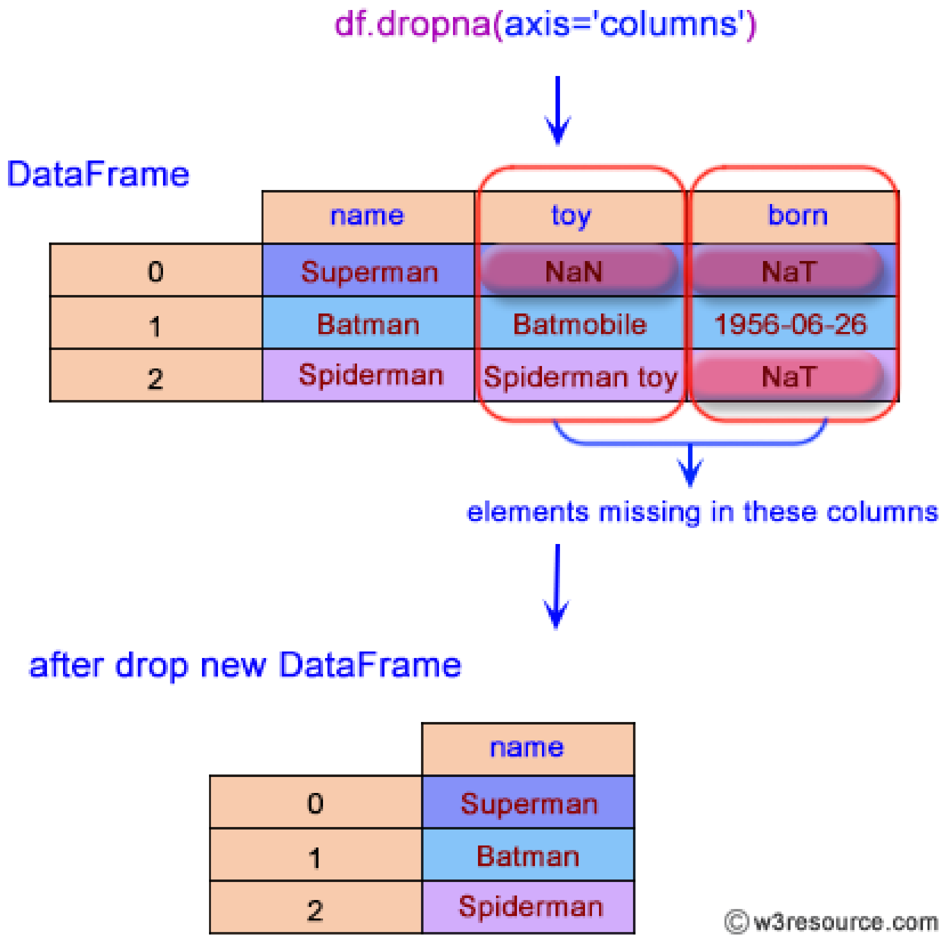
Cleaning Empty Cells

```
dropna(self, axis=0, how="any", thresh=None, subset=None,
inplace=False)
```

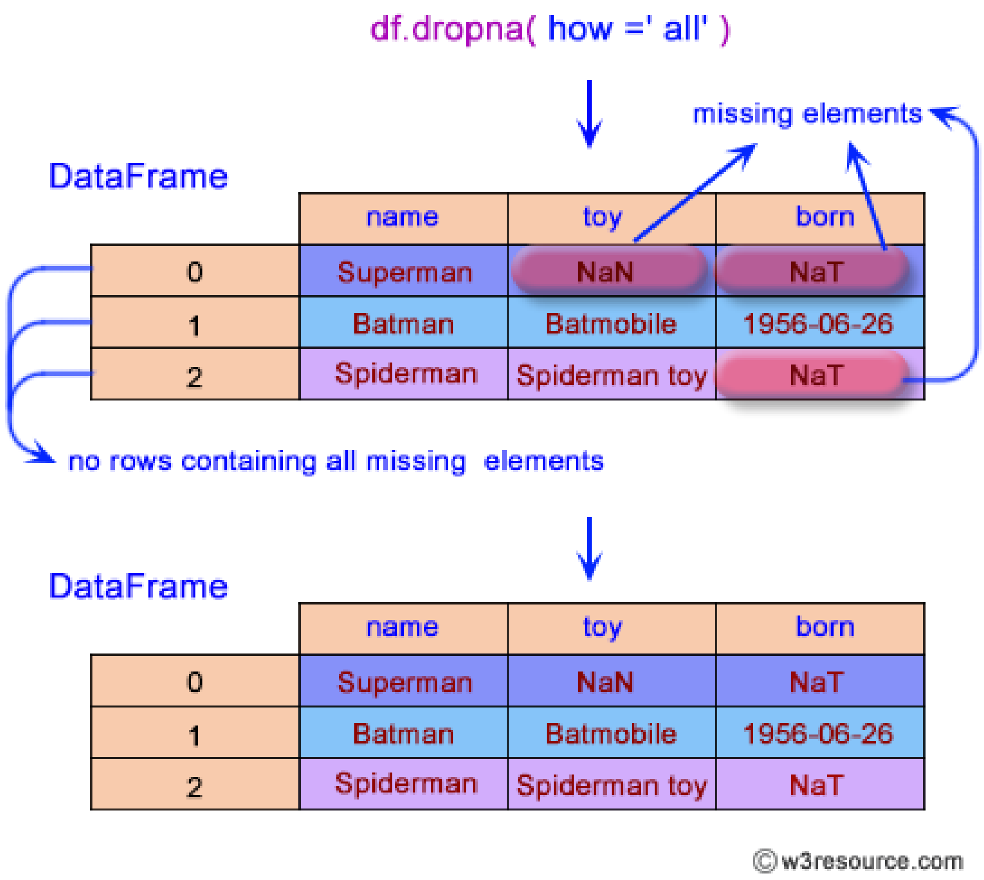
- **axis**: possible values are {0 or 'index', 1 or 'columns'}, default 0. If 0, drop rows with null values. If 1, drop columns with missing values.
- **how**: possible values are {'any', 'all'}, default 'any'. If 'any', drop the row/column if any of the values is null. If 'all', drop the row/column if all the values are missing.
- **thresh**: an int value to specify the threshold for the drop operation.
- **subset**: specifies the rows/columns to look for null values.
- **inplace**: a boolean value. If True, the source DataFrame is changed and None is returned.

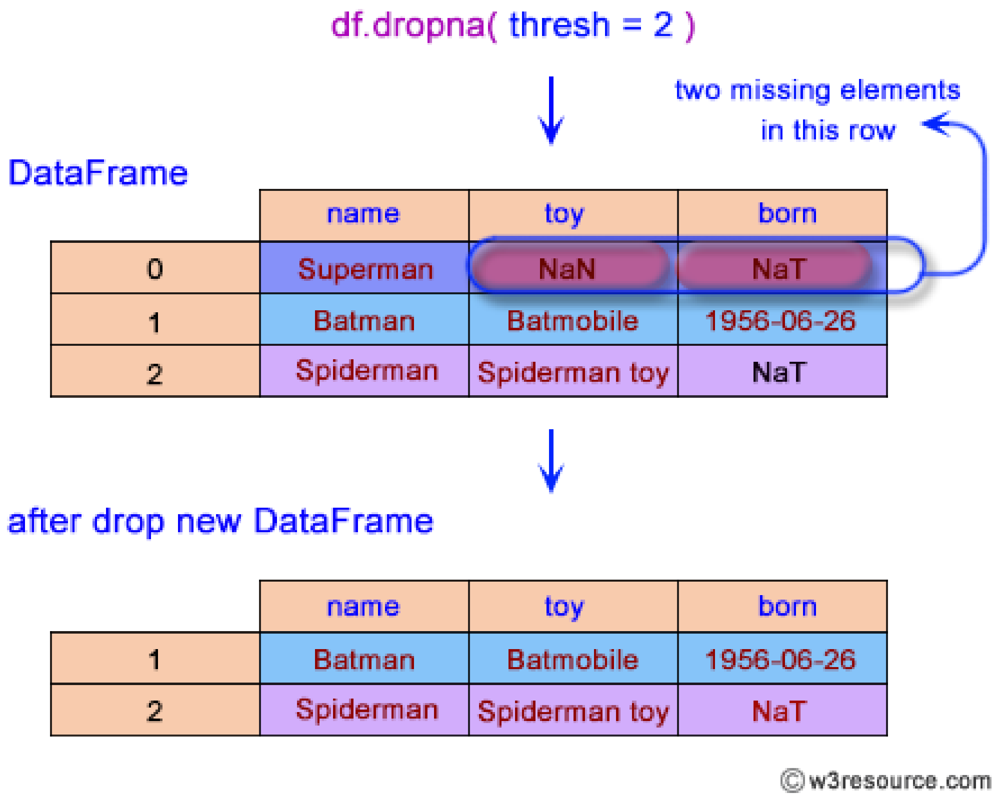
Cleaning Empty Cells



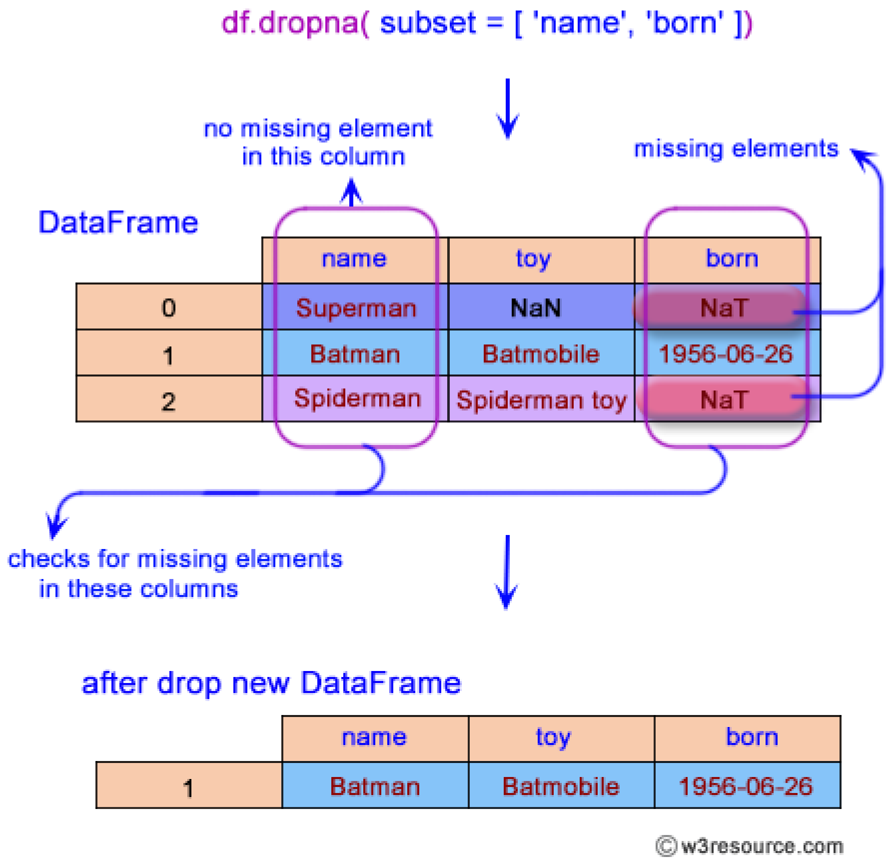


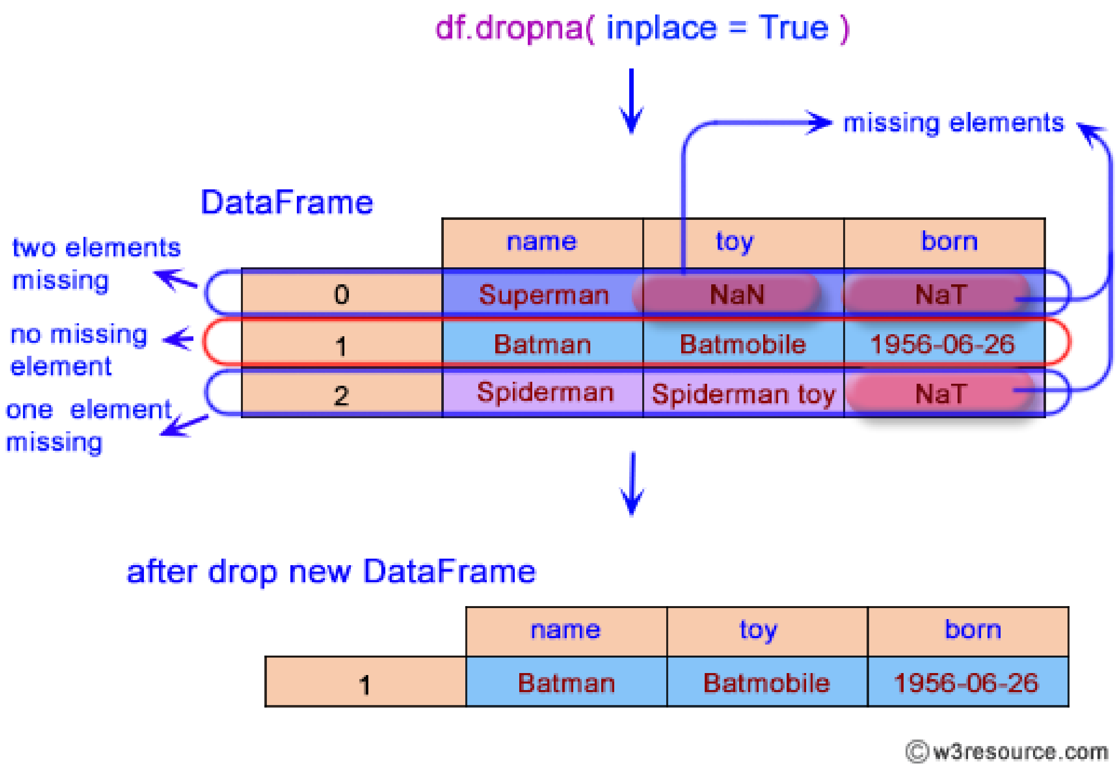
Cleaning Empty Cells





Cleaning Empty Cells





Removing Duplicates

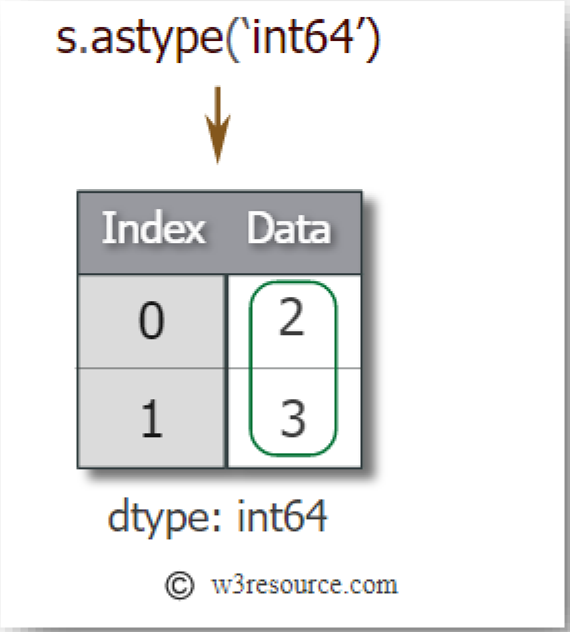
```
drop_duplicates(self, subset=None, keep="first", inplace=False)
```

name	region	sales	expense
William	East	50000	42000
~~William~~	~~East~~	~~50000~~	~~42000~~
Emma	North	52000	43000
Emma	West	52000	43000
Anika	East	65000	44000
Anika	East	72000	53000

Data Format

pd.to_datetime( )

"Given a format, convert a string to a datetime object"



Set DataFrame Index

df.set_index ('month')

to be replaced

set index by month column

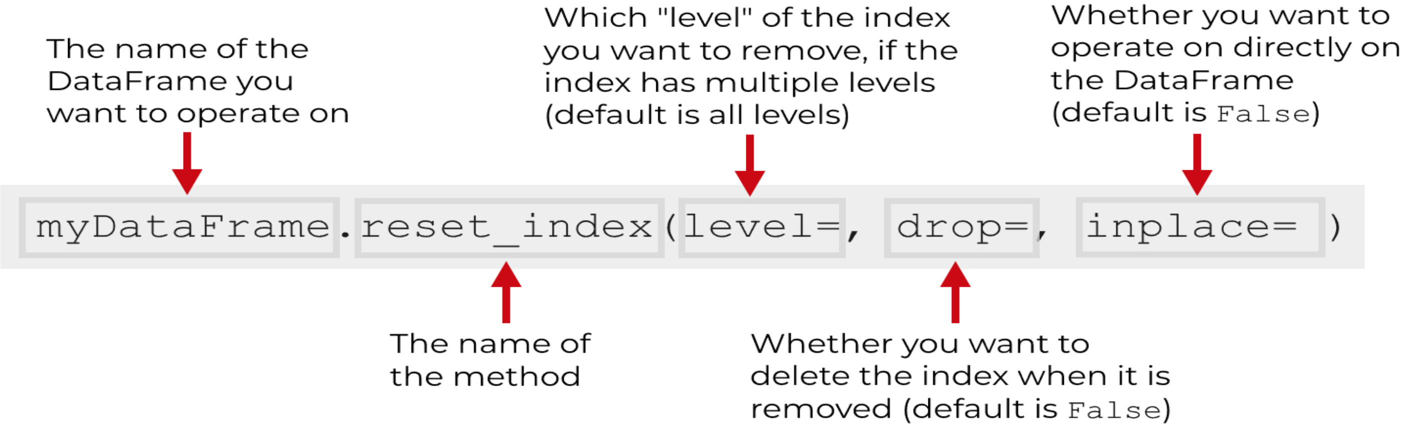
	month	year	sale
0	2	2017	60
1	5	2019	45
2	8	2018	90
3	10	2019	36

	year	sale
month		
2	2017	60
5	2019	45
8	2018	90
10	2019	36

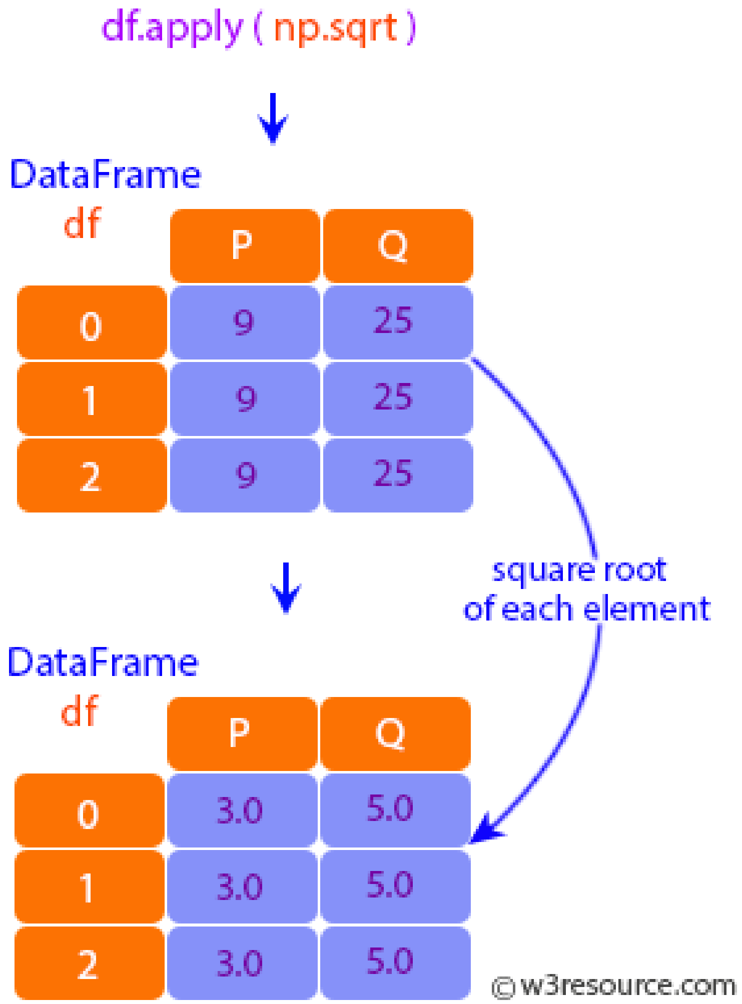
Reset DataFrame Index

	A	B	C
1	a1	1	c1
3	a2	2	c2
5	a3	3	c3
7	a4	4	c4

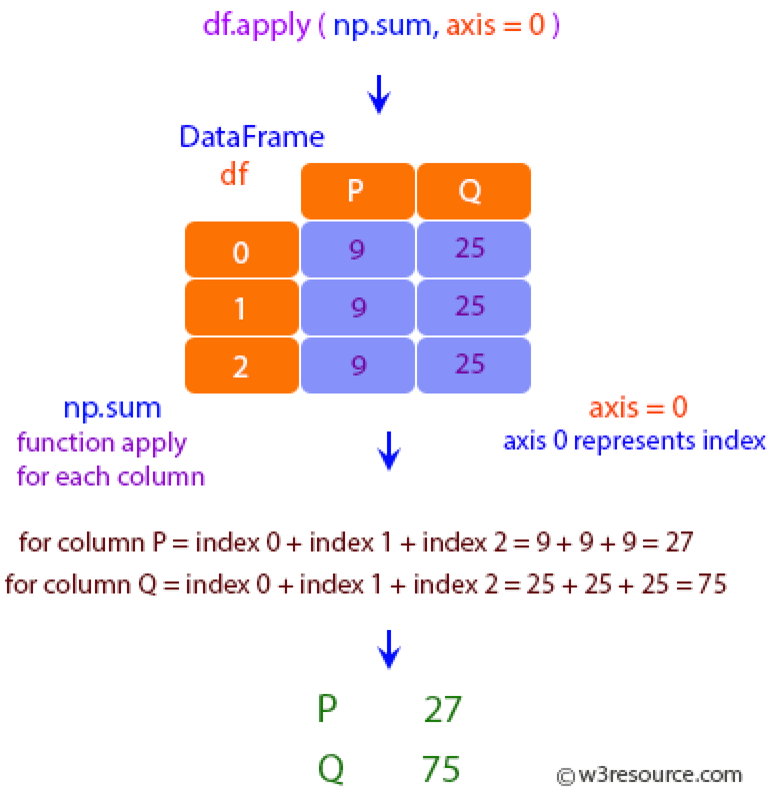
	A	B	C
0	a1	1	c1
1	a2	2	c2
2	a3	3	c3
3	a4	4	c4

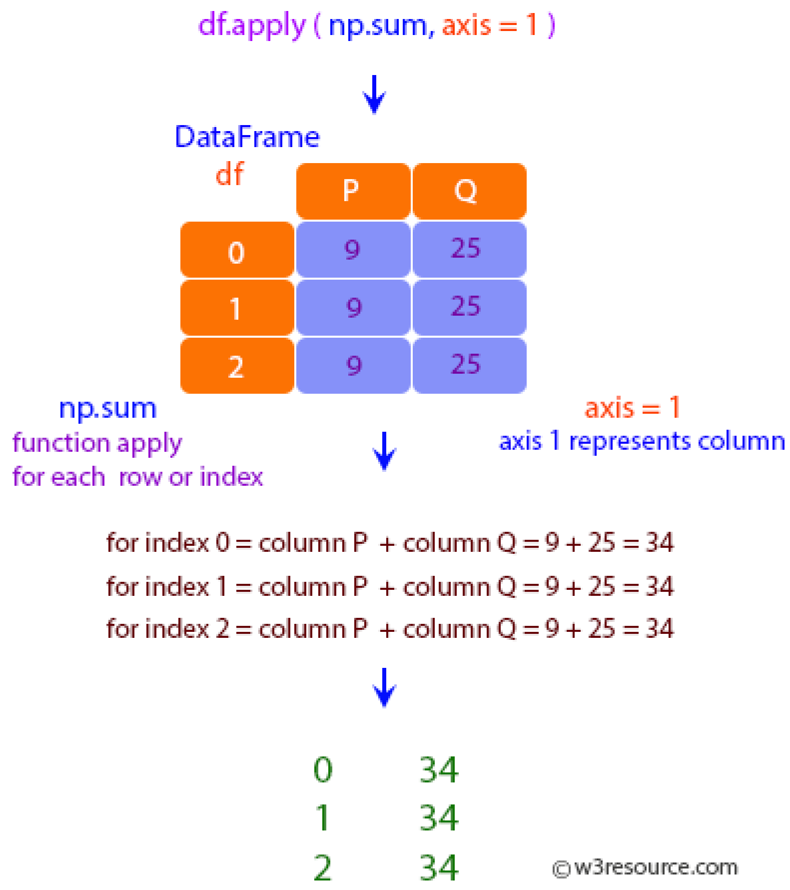


Apply Function

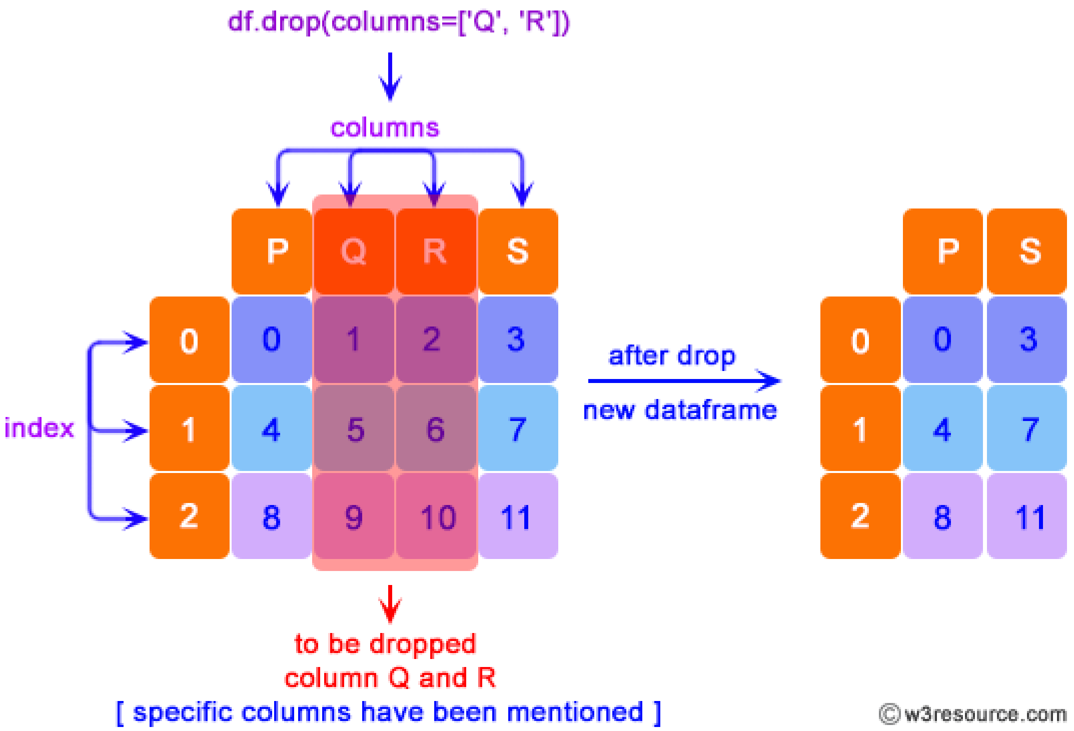


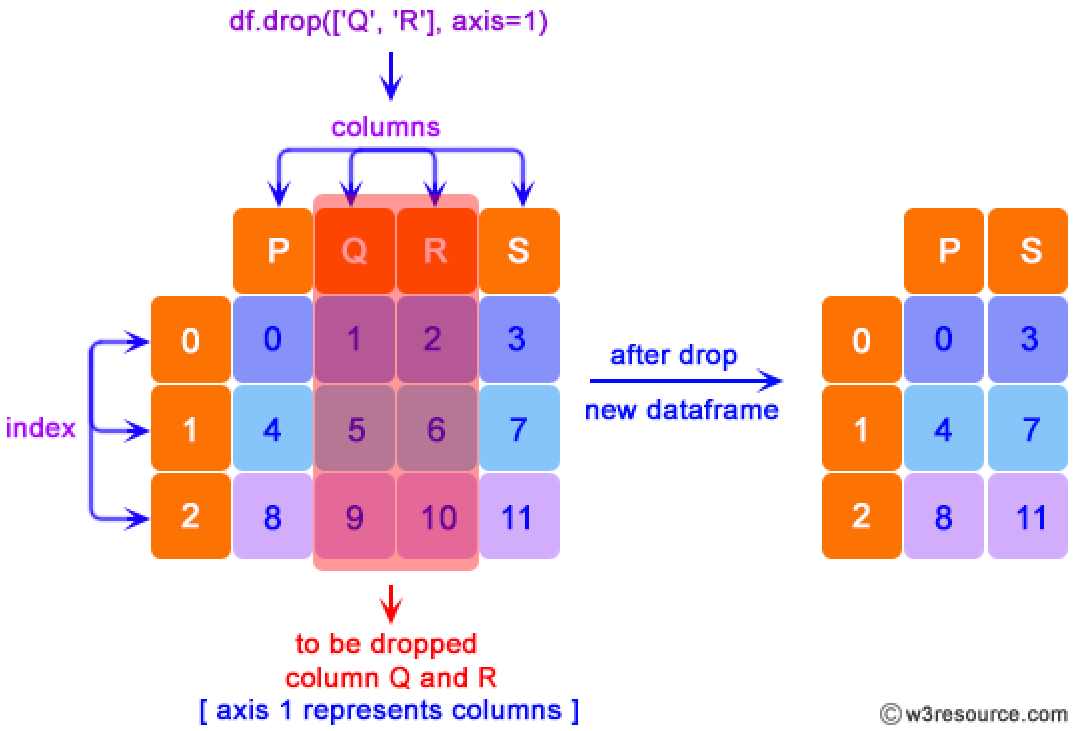
Apply Function



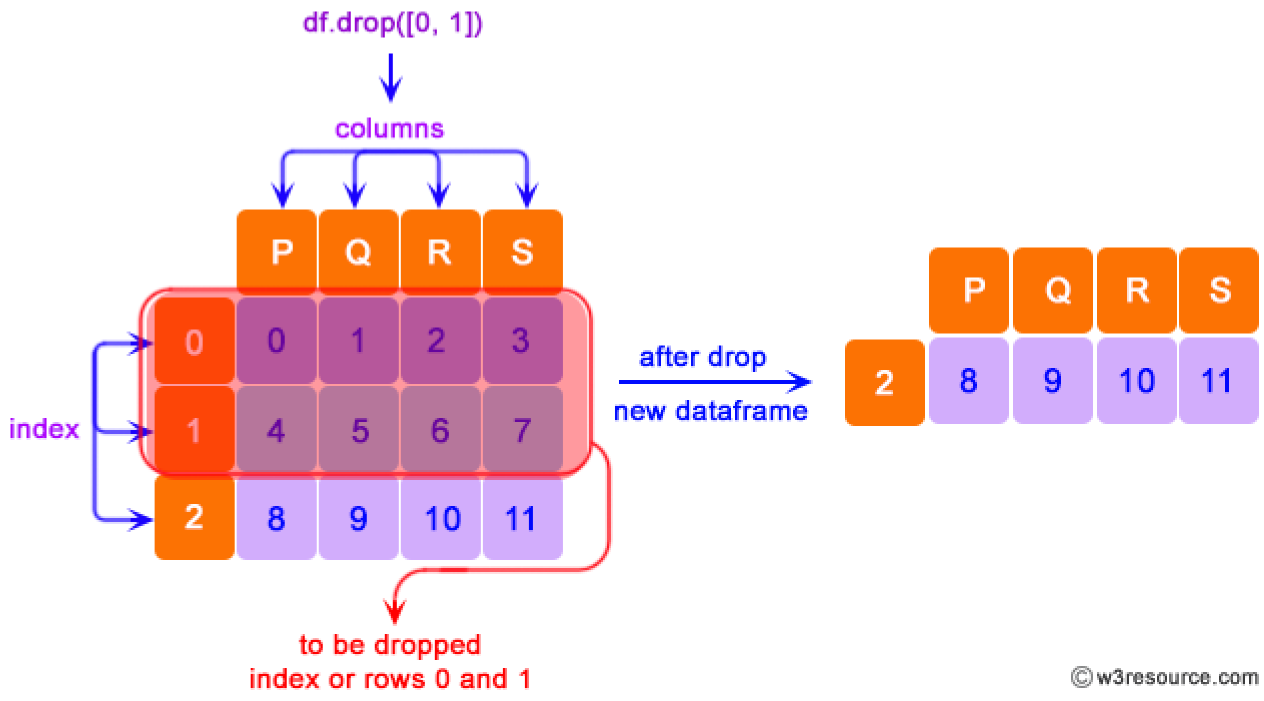


Drop Function

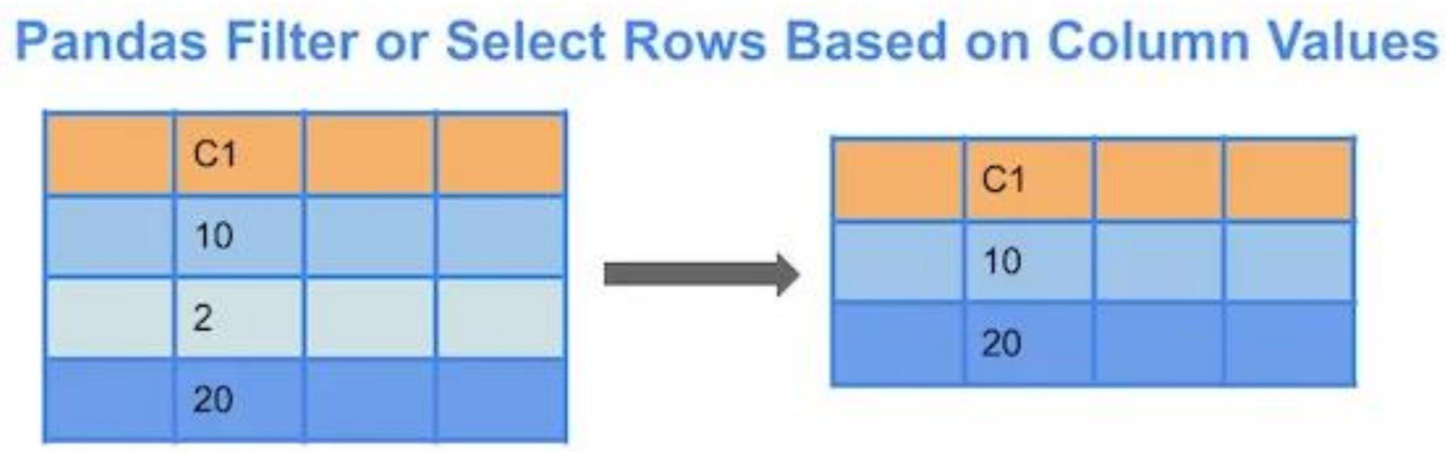




Drop Function



filter

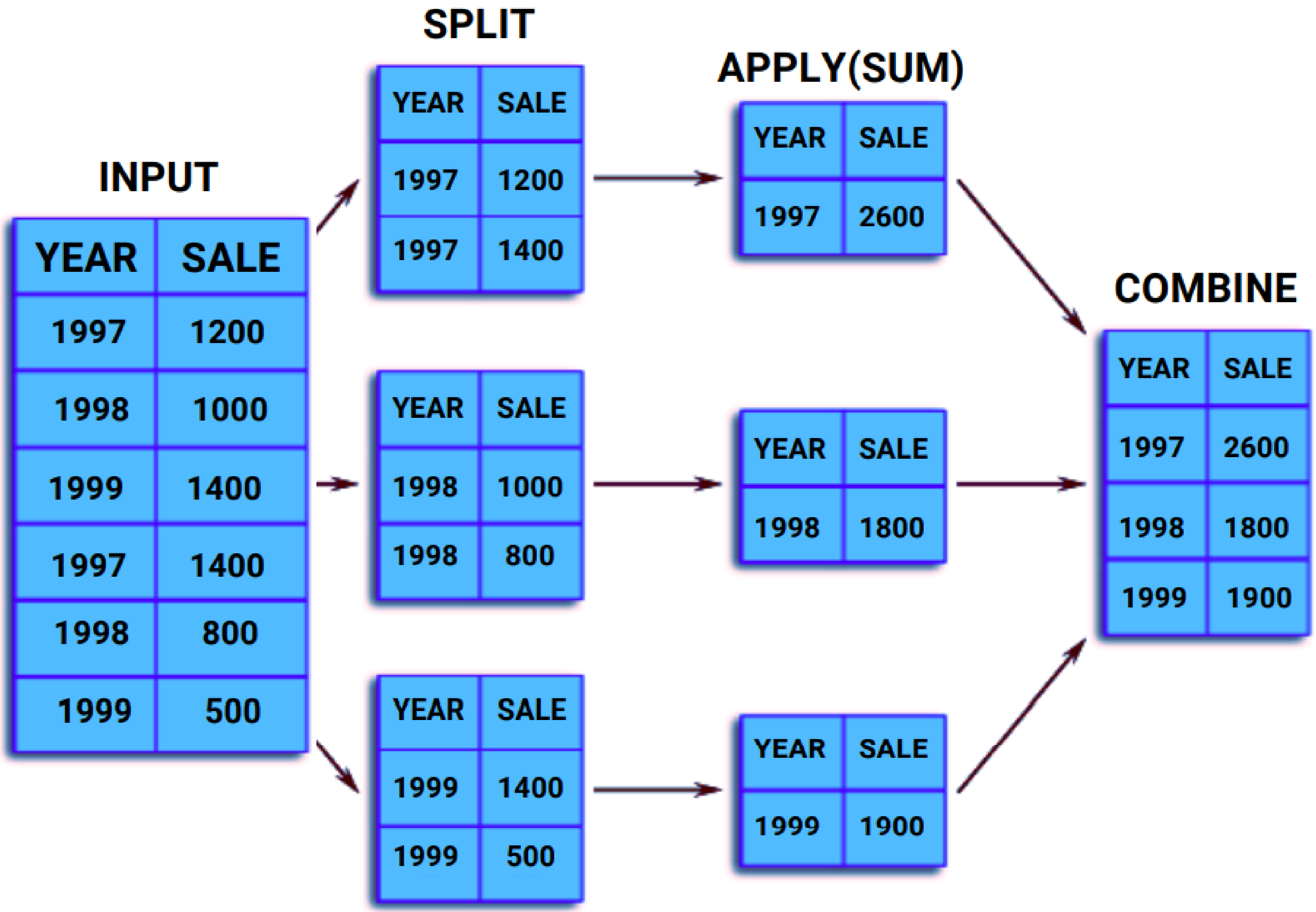


Pandas Filter/Select Rows Based on Column Values

filter

```
# filter Rows Based on condition
df[df["Courses"] == 'Spark']
df.loc[df['Courses'] == value]
df.query("Courses == 'Spark'")
df.loc[df['Courses'] != 'Spark']
df.loc[df['Courses'].isin(values)]
df.loc[~df['Courses'].isin(values)]

# filter Multiple Conditions using Multiple Columns
df.loc[(df['Discount'] >= 1000) & (df['Discount'] <= 2000)]
df.loc[(df['Discount'] >= 1200) & (df['Fee'] >= 23000 )]

# Using lambda function
df.apply(lambda row: row[df['Courses'].isin(['Spark','PySpark'])])

# filter columns that have no None & nana values
df.dropna()

# Other examples
df[df['Courses'].str.contains("Spark")]
df[df['Courses'].str.lower().str.contains("spark")]
df[df['Courses'].str.startswith("P")]
```

Group by


SPLIT
APPLY(SUM)
INPUT
COMBINE
YEAR SALE
1997 1200
1998 1000
1999 1400
1997 1400
1998 800
1999 500
YEAR SALE
1997 1200
1997 1400
YEAR SALE
1998 1000
1998 800
YEAR SALE
1999 1400
1999 500
YEAR SALE
1997 2600
YEAR SALE
1998 1800
YEAR SALE
1999 1900
YEAR SALE
1997 2600
1998 1800
1999 1900

Group by

df.groupby('column_to_group')['column_to_agg'].agg_function()

df.groupby('Name')['AvgBill'].sum()

Index	Name	Type	AvgBill
0	Liho Liho	Restaurant	$45.32
1	Chambers	Restaurant	$65.33
2	The Square	Bar	$12.45
3	Tosca Cafe	Restaurant	$180.34
4	Liho Liho	Restaurant	$145.42
5	Chambers	Restaurant	$25.35

→

Liho Liho: $190.74
Chambers: $90.68
The Square: $12.45
Tosca Cafe: $180.34

Aggregation Methods

Aggregation Method	Description
.count()	The number of non-null records
.sum()	The sum of the values
.mean()	The arithmetic mean of the values
.median()	The median of the values
.min()	The minimum value of the group
.max()	The maximum value of the group
.mode()	The most frequent value in the group
.std()	The standard deviation of the group
.var()	The variance of the group

Group by Example

```
daily_spend_count = df.groupby('Day')['Debit'].count()
daily_spend_sum = df.groupby('Day')['Debit'].sum()
```

1. Split the data by using values in the "Day" column

```
daily_spend = df.groupby('Day').agg({'Debit':['sum','count']})
```

2. Perform both "sum" and "count" operations on the "Debit" column of the grouped data

```
df.groupby(['Category','Month'])['Debit'].sum()
```

Sort

The name of the DataFrame you want to operate on

The column or list of columns you want to sort by

Whether you want to operate on directly on the DataFrame (default is `False`)

```
myDataFrame.sort_values(by, ascending =, inplace = )
```

The name of the method

Whether you want to sort in ascending order (default is `True`)

Correlations

	Maths	Physics	History
0	78	81	53
1	85	77	65
2	67	63	95
3	69	74	87
4	53	46	63
5	81	72	58
6	93	88	73
7	74	76	42

df.corr()

	Maths	Physics	History
Maths	1.000000	0.906340	-0.159063
Physics	0.906340	1.000000	-0.158783
History	-0.159063	-0.158783	1.000000

Concatenate

	Country	Currency
0	Japan	Yen
1	China	chin
2	India	rupees

	capital	population
0	Jaya'pura	456750
1	Toranto	3456545
2	Aukland	89752

Pandas concat( )

Default axis = 0

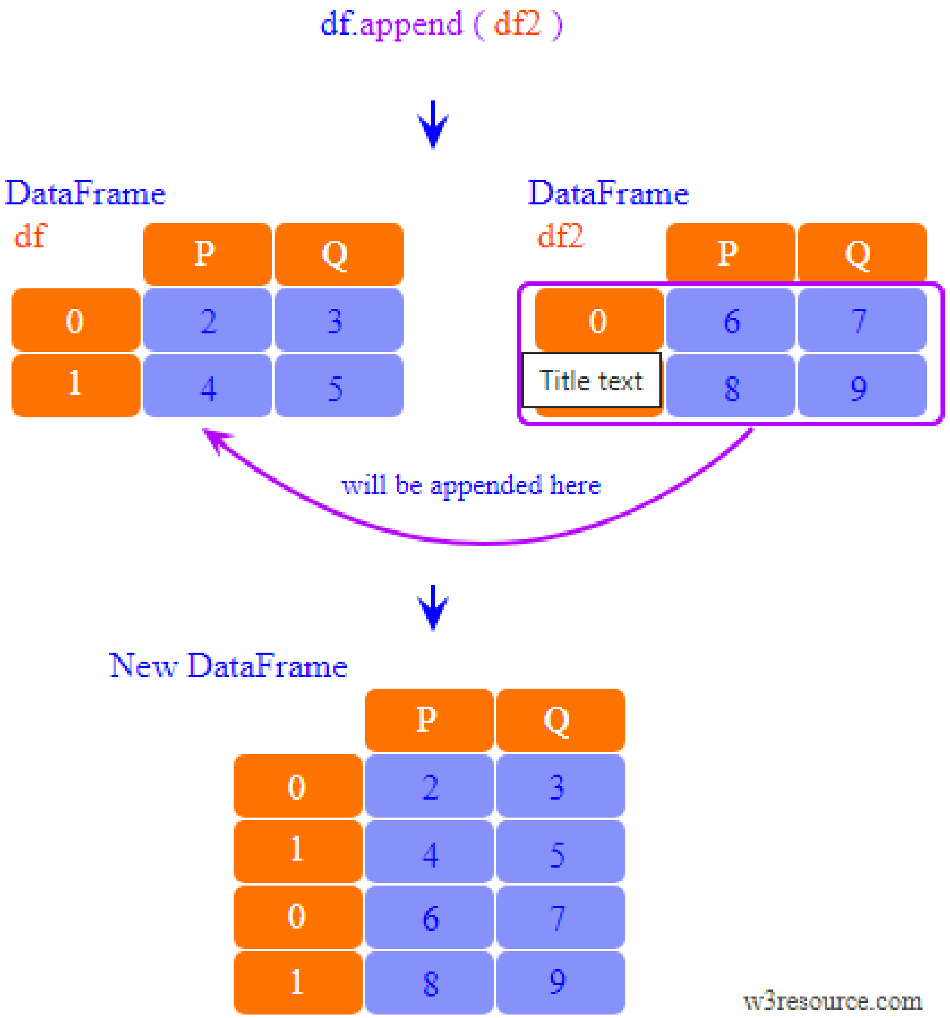
	Country	Currency	capital	population
0	Japan	Yen	NaN	NaN
1	China	chin	NaN	NaN
2	India	rupees	NaN	NaN
0	NaN	NaN	Jaya'pura	456750.0
1	NaN	NaN	Toranto	3456545.0
2	NaN	NaN	Aukland	89752.0

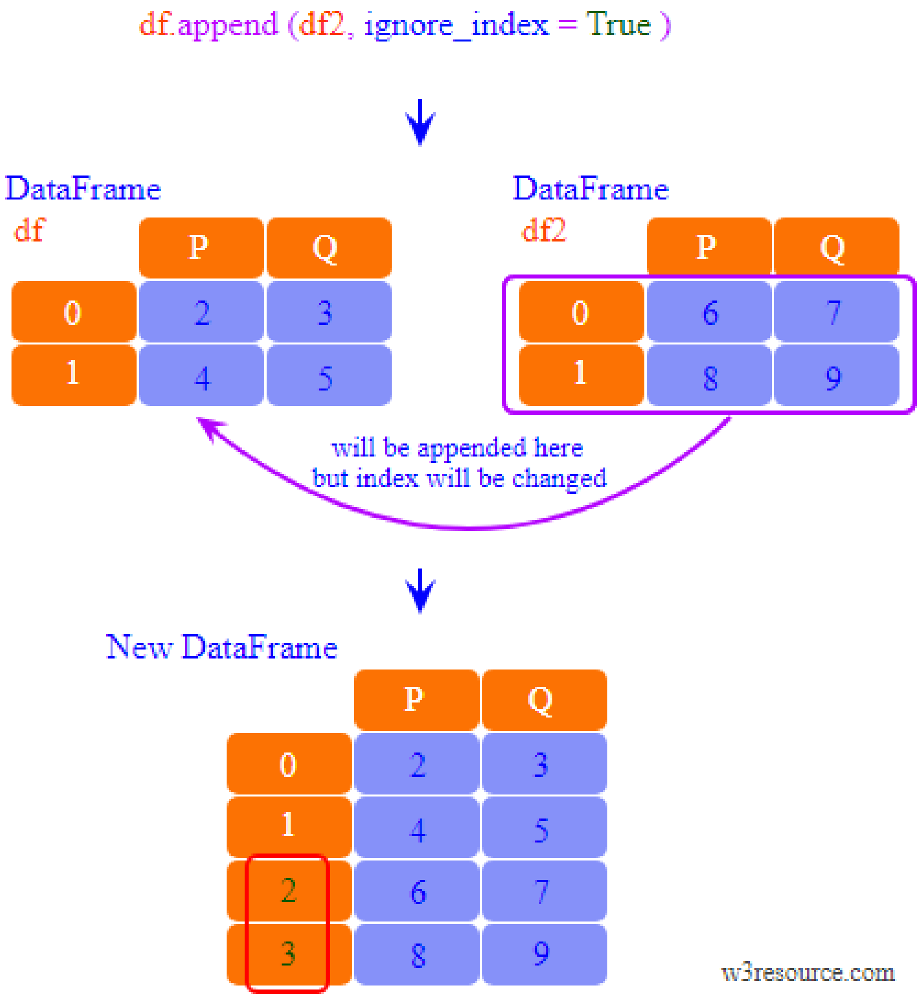
axis = 1

```
pd.concat([data_3, my_df_drop], axis = 1)
```

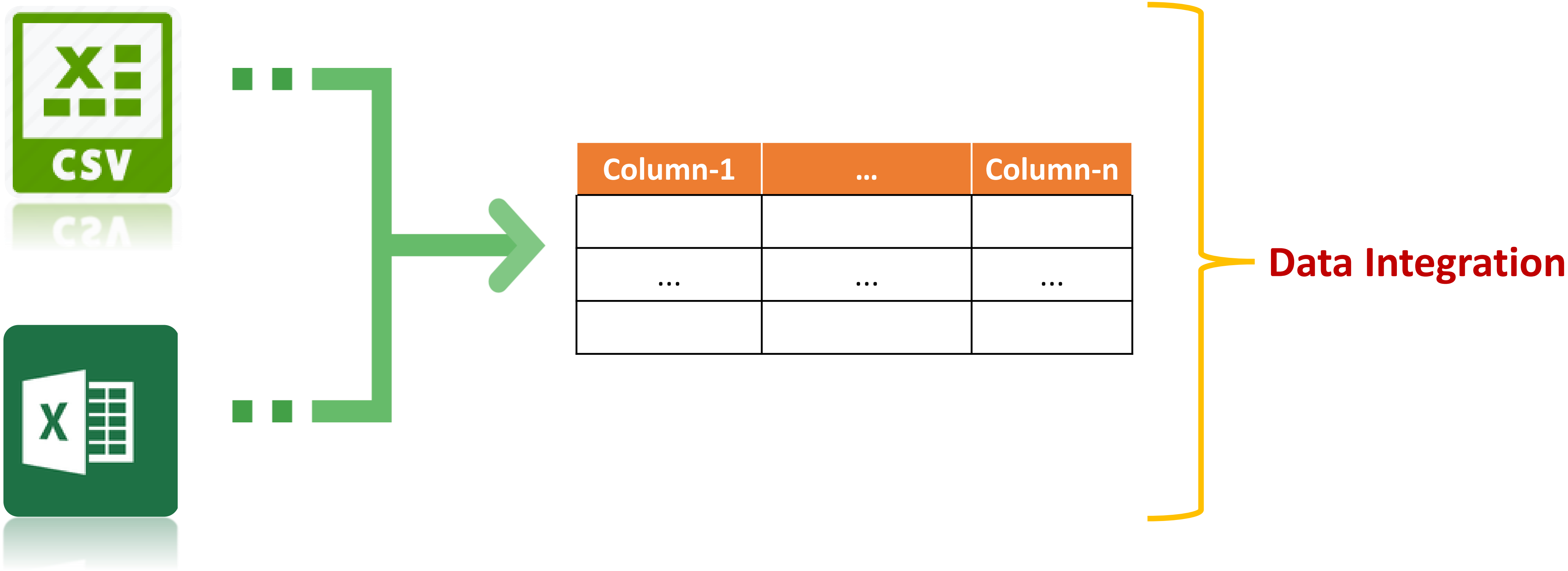
	Country	Currency	capital	population
0	Japan	Yen	Jaya'pura	456750
1	China	chin	Toranto	3456545
2	India	rupees	Aukland	89752

append


df.append (df2)
DataFrame
df
P
Q
0
2
3
1
4
5
DataFrame
df2
P
Q
0
6
7
Title text
8
9
will be appended here
New DataFrame
P
Q
0
2
3
1
4
5
0
6
7
1
8
9
w3resource.com


df.append (df2, ignore_index = True)
DataFrame
df
P
Q
0
2
3
1
4
5
DataFrame
df2
P
Q
0
6
7
1
8
9
will be appended here
but index will be changed
New DataFrame
P
Q
0
2
3
1
4
5
2
6
7
3
8
9
w3resource.com

Merge Function



Merge Function

index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29
6	1962	8.77
7	1963	26.88
8	1964	15.04

index	Year	Rainfall
0	1956	1.01
1	1957	1.66
2	1958	3.5
3	1959	3.31
4	1960	3.61
5	1961	4.71
6	1962	2
7	1963	3.12
8	1964	1.96

Column-1	...	Column-n
...	...	...

Pandas Merge

df.merge(right=other_df, on='common_column' , how='how_to_join')

df

index	Year	Rainfall
0	1956	1.01
1	1957	1.66
2	1958	3.5
3	1959	3.31
4	1960	3.61
5	1961	4.71

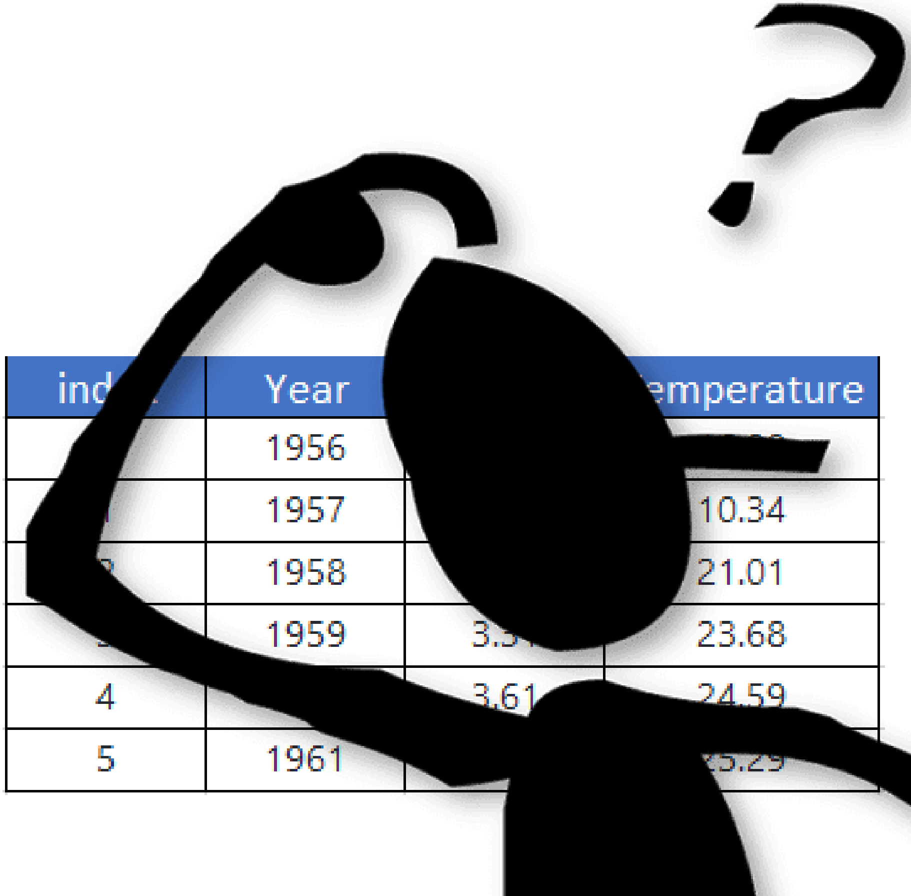
+

Other_df

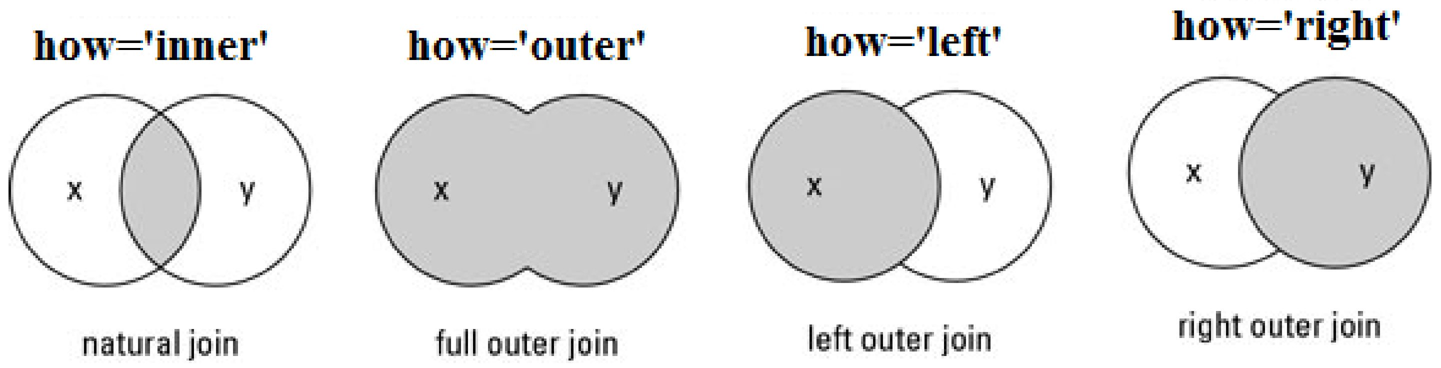
index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29

=

ind[illegible]	Year	[illegible]	[illegible]emperature
[illegible]	1956	[illegible]	[illegible]
[illegible]	1957	[illegible]	10.34
[illegible]	1958	[illegible]	21.01
[illegible]	1959	[illegible]	23.68
4	[illegible]	3.61	24.59
5	1961	[illegible]	[illegible]



Pandas Merge



index	Year	Temperature
0	1956	16.99
1	1957	10.34
2	1958	21.01
3	1959	23.68
4	1960	24.59
5	1961	25.29

index	Year	Rainfall
0	1956	1.01
1	1958	3.5
2	1959	3.31
3	1960	3.61
4	1962	2
5	1963	3.12

index	Year	Temperature	Rainfall
0	1956	16.99	1.01
1	1957	10.34	Nan
2	1958	21.01	3.5
3	1959	23.68	3.31
4	1960	24.59	3.61
5	1961	25.29	Nan
6	1962	Nan	2
7	1963	Nan	3.12

Pandas concat Vs append Vs join Vs merge

- **Concat** gives the flexibility to join based on the axis(all rows or all columns)
- **Append** is the specific case(axis=0, join='outer') of concat
- **Merge** is based on any particular column each of the two dataframes, this columns are variables on like 'left_on', 'right_on', 'on’.
- **Join** is based on the indexes (set by set_index) on how variable =['left','right','inner','outer']

THANK YOU