Intro to Data Analysis

Data Science with Python Bootcamp

BIG DATA UNIVERSITY

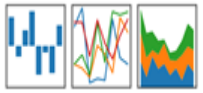
Outline

1. Data Science packages
2. NumPy Lib
3. Pandas
   - Series and DataFrames
4. Pandas DataFrame Operations

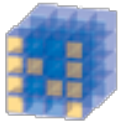
Python package for data science

- Scientifics Computing Libraries

1. **Scientifics Computing** Libraries



Pandas
(Data structures & tools)



NumPy
(Arrays & matrices)

SciPy
(Integrals, solving differential equations, optimization)

BIG DATA UNIVERSITY

Python package for data science

- Algorithmic Libraries

3. **Algorithmic** libraries

SciKit-Learn

(Machine Learning : regression, classification,…)

TensorFlow

(Artificial neural network)

StatsModels

(Explore data, estimate statistical models, and perform statistical tests.)

NumPy

BIG DATA UNIVERSITY

What is NumPy

- **NumPy**: NumPy is the fundamental package for scientific computing with Python
  - Fast
  - Multidimensional Arrays
  - Vectorized Computation

```
import numpy as np
```

```
data=np.array([[ 1.9526, -0.246 , -0.8856],
[ 0.5639, 0.2379, 0.9104]])
data
```

```
array([[ 1.9526, -0.246 , -0.8856],
       [ 0.5639,  0.2379,  0.9104]])
```

BIG DATA UNIVERSITY

Indexing arrays

- Use a tuple to index multi-dimensional arrays
- `a[2,3]`

- Slicing arrays

```
a[2]
a[2,:]
a[1,2:]
a[:1,2:]
```

Some useful ndarray methods

- ndarray. tolist ()
  - The contents of self as a nested list
- ndarray. copy ()
  - Return a copy of the array
- ndarray. fill (scalar)
  - Fill an array with the scalar value

Pandas: Series and DataFrames

BIG DATA UNIVERSITY

What is Pandas

- **Pandas** is a **Python** package for data analysis
- It is comprised of:
    1. Series
    2. Data frames

- You can download data
- Visualize data
- Analyze data

Pandas : Series

- Series are a list of numbers, each element has data and an index
- Default index are integers

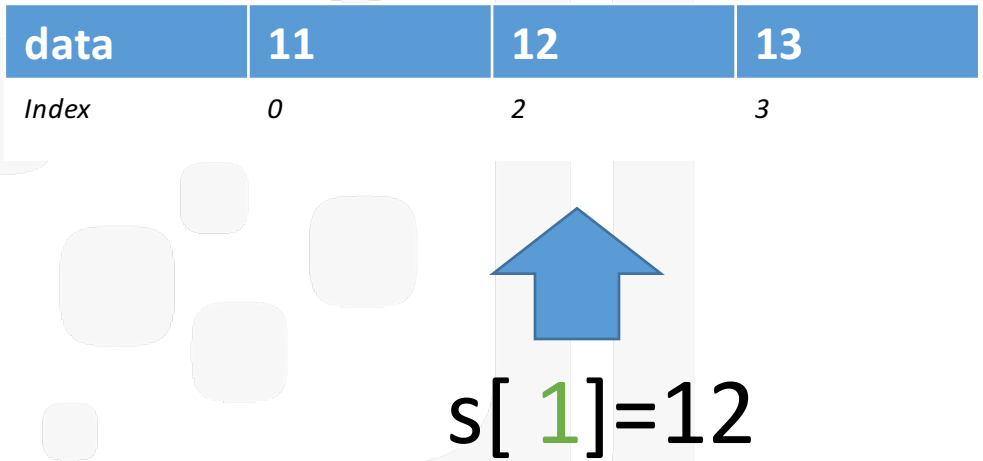
```
data=[11,23,23]
s = pd.Series(data)
```

```
s
```

```
0    11
1    23
2    23
dtype: int64
```

```
s[1]
```

```
23
```



BIG DATA UNIVERSITY

Pandas : Series

- Series are a list of numbers, each element has data and an index
- Find element in array using index

```
data=[11,23,23]
index=["a","b","c"]
s = pd.Series(data, index=index)
```

```
s
```

```
a    11
b    23
c    23
dtype: int64
```

```
s["a"]
```

```
11
```

data	11	12	13
Index	a	b	c

s[“a”]=11

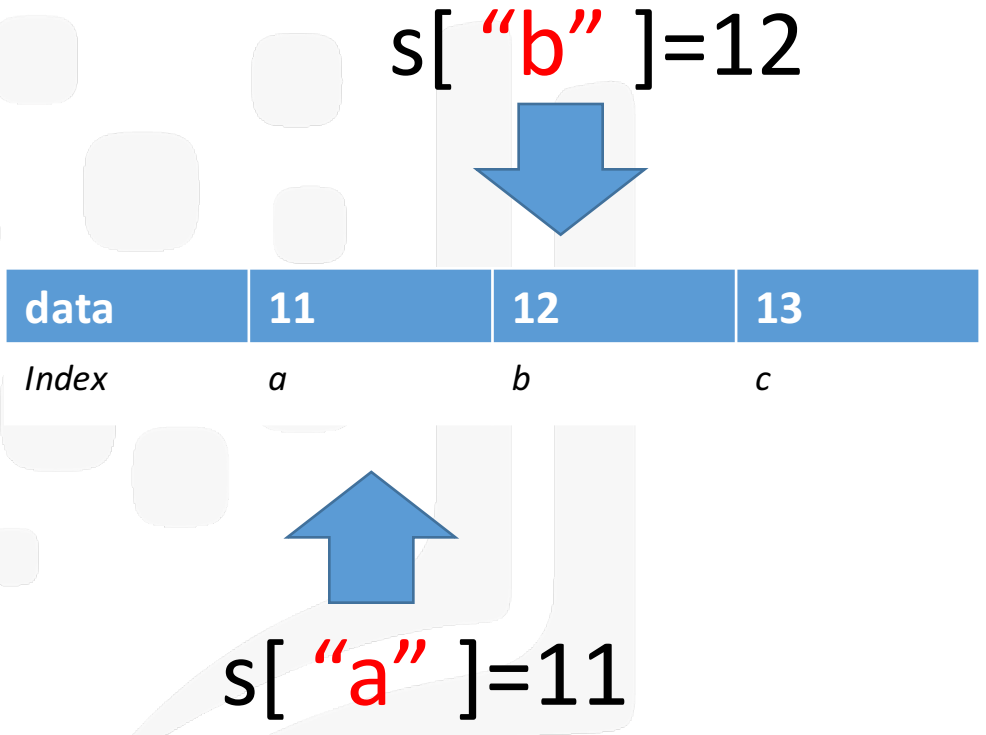
BIG DATA UNIVERSITY

Pandas : Series

- Series are a list of numbers, each element has data and an index
- Find element in array using index

```
import pandas as pd

data = [11,12,13]

ind = ["a","b","c"]

s = pd.Series(data, index=ind)

s[["a", "b"]]
```

```
a    11
b    12
dtype: int64
```



BIG DATA UNIVERSITY

Pandas: Dataframe

BIG DATA UNIVERSITY

Data Acquisition

- **Read Data**

we use `pandas.read_csv()` method to read csv file into Pandas DataFrame.

```
pandas.read_csv(filepath,  header = None)
```

- `header` is an argument that allow you set header of the dataset. Default value is “infer”, means set the first row as header, None here means do not set headers.

```
# import pandas library
import pandas as pd
# read the online file by the URL provides above, and assign it to variable "df"
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data",  header = None)
```

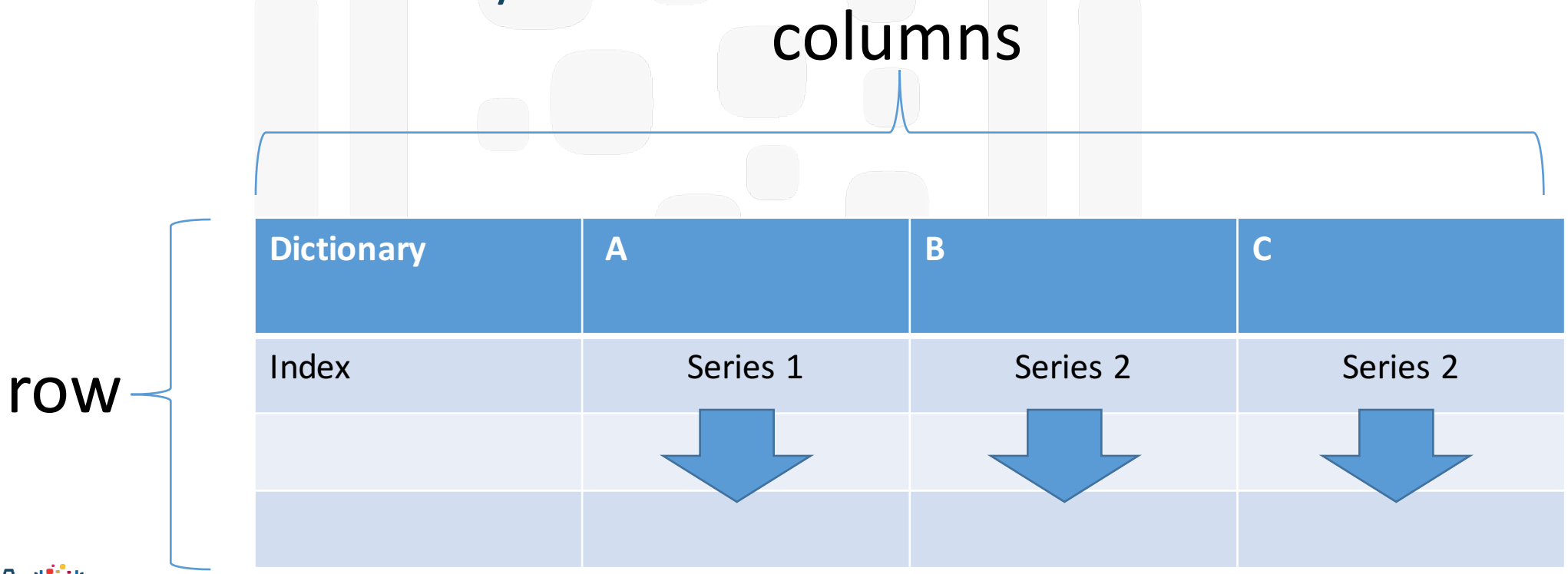
Data Acquisition - Other formats

- **Read/Save Other Data Formats**
  - Same as read/save csv file, we use similar methods to read/save other dataset formats:

Data Format	Read	Save
csv	pd.read_csv()	df.to_csv()
json	pd.read_json()	df.to_json()
excel	pd.read_excel()	df.to_excel()
hdf	pd.read_hdf()	df.to_hdf()
sql	pd.read_sql()	df.to_sql()
...	...	...

Pandas: “DataFrame”

- Spreadsheet-like containing an order collection of columns
- Has both a rows and columns index
- Consider as dict Dictionary of Series



BIG DATA UNIVERSITY

Pandas: “DataFrame”

```
import pandas as pd
```

```
data = {'name': ['Joe', 'Cat', 'Mike', 'Kim', 'Amy'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'Points': [4, 24, 31, 2, 3]}
df = pd.DataFrame(data, index = ['Day1', 'Day2', 'Day3', 'Day4', 'Day5'])
df
```

	Points	name	year
Day1	4	Joe	2012
Day2	24	Cat	2012
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014

Pandas: “DataFrame”

	Points	name	year
Day1	4	Joe	2012
Day2	24	Cat	2012
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014

```
df['Points']
```

```
Day1      4
Day2     24
Day3     31
Day4      2
Day5      3
Name: Points, dtype: int64
```

```
df['name']
```

```
Day1      Joe
Day2      Cat
Day3     Mike
Day4      Kim
Day5      Amy
Name: name, dtype: object
```

- Can select columns:

```
df['Points']
```

	Points	name	year
Day1	4	Joe	2012
Day2	24	Cat	2012
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014

```
df['name']
```

BIG DATA UNIVERSITY

Pandas Dataframe Operations

BIG DATA UNIVERSITY

Basic Insights of Dataset – Describe()

- Check the **statistical summary** of each column, such as:
  - records count,
  - column mean value,
  - column standard deviation, etc.
- `dataframe.describe()` generates various summary statistics, excluding NaN (Not a Number) values.

Dataframe Operations

1. Describe
2. Unique
3. Group-by
4. Merging Data
5. Data Cleaning

BIG DATA UNIVERSITY

Basic Insights of Dataset – Describe()

In [10]:
```
# use .describe method to get the stat summary of "df"
df.describe()
```

Out[10]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

- Here it shows the statistical summary of all numeric-typed (int and float) columns.

List Unique Values

- “Unique” is a method that List Unique Values In A Pandas Colum

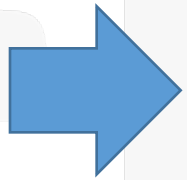
BIG DATA UNIVERSITY

Example

- Select the column you would like the unique values for
- There are multiple ways to do this

```
df['Start_Date']
```

	Start_Date	name
0	2002	Jason
1	2012	Molly
2	2012	Tina
3	2014	Jake
4	2014	Amy
5	1999	John
6	1999	Bob



Example

- The unique values are obtained as follows

```
df['Start_Date'].unique()
```

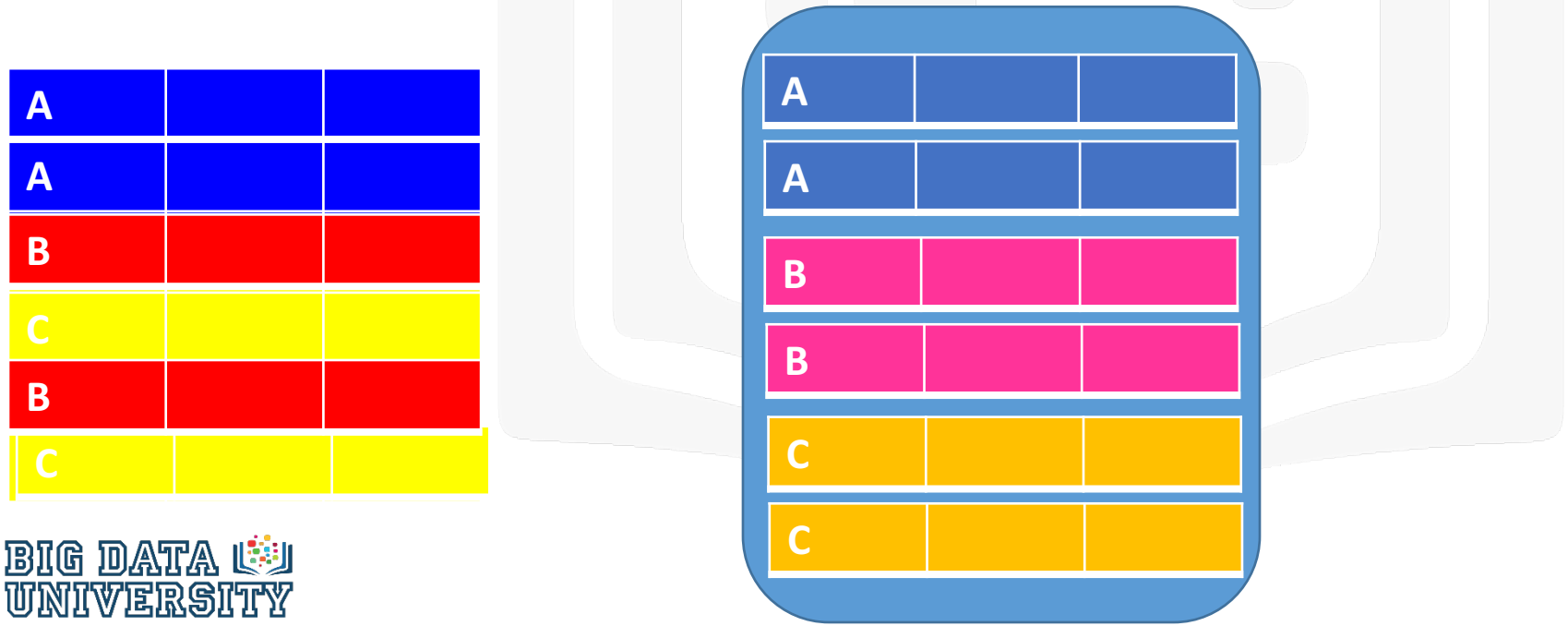
Start_Date
2002
2012
2012
2014
2014
1999
1999

BIG DATA UNIVERSITY

Group By

- By “group by” involving one or more of the following steps

1. Splitting the data into groups based on some criteria
2. Applying a function to each group independently
3. Combining the results into a data structure



BIG DATA UNIVERSITY

Group By

- Analyzing Pandas Series by Category
- Example Finding the average rain fall of a country using the rain fall in cities

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

Create a Data Frame and Display

```
CountryList=['Iraq', 'China', 'USA', 'China','USA','Iraq','USA','USA']
CityList=['Baghdad', 'Beijing', 'New York', 'Hong Kong','Los Angeles','Karbala','Chicago','Seattle']
RainNumpyArray=np.array([2.24,25,47.2,87,15,3.5,34,40])
Blaa=np.array([2.24,25,47.2,87,15,3.5,34,40])
df = pd.DataFrame({'City':CityList ,'Country':CountryList,'RainFall ':RainNumpyArray})

df.head(8)
```

	City	Country	RainFall
0	Baghdad	Iraq	2.24
1	Beijing	China	25.00
2	New York	USA	47.20
3	Hong Kong	China	87.00
4	Los Angeles	USA	15.00
5	Karbala	Iraq	3.50
6	Chicago	USA	34.00
7	Seattle	USA	40.00

Group By

- Lest see how the this grope by works
- Consider the following Example:

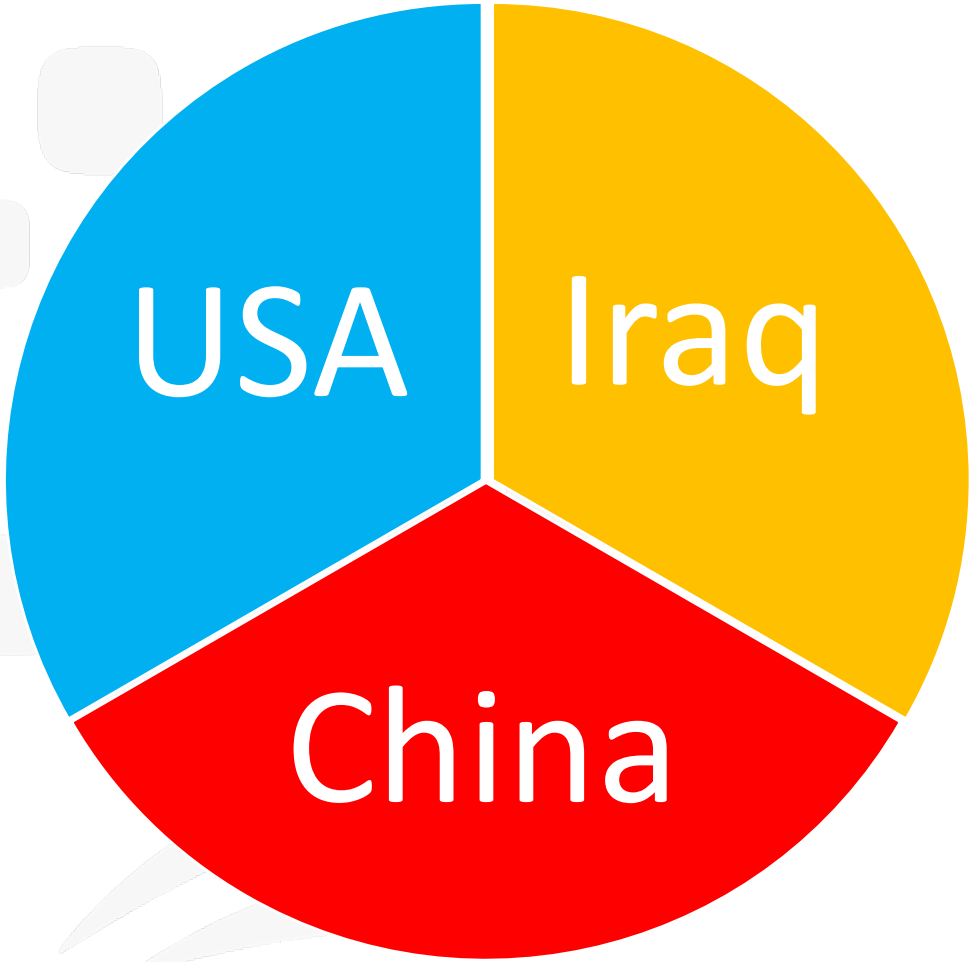
```
df.groupby(['Country'],as_index= False).mean()
```

	Country	RainFall
0	China	56.00
1	Iraq	2.87
2	USA	34.05

Group By

```
df.groupby(['Country'],as_index= False)
```

City	Country	Rain Fall (inch)
Baghdad	Iraq	2.24
Beijing	China	25
New York	USA	47.2
Hong Kong	China	87
Los Angeles	USA	15
Karbala	Iraq	3.5
Chicago	USA	34
Seattle	USA	40



BIG DATA UNIVERSITY

Sorce:www.worldclimate.com/

Group By

```
df.groupby(['Country'],as_index= False).mean()
```

Country	Mean Rain Fall
China	56

City	Country	Rain Fall (inch)
Baghdad	Iraq	2.24
Beijing	China	25
New York	USA	47.2
Hong Kong	China	87
Los Angeles	USA	15
Karbala	Iraq	3.5
Chicago	USA	34
Seattle	USA	40

$$Mean = \frac{1}{2}(25 + 87) = 56$$

BIG DATA UNIVERSITY

Group By

City	Country	Rain Fall (inch)
Baghdad	Iraq	2.24
Beijing	China	25
New York	USA	47.2
Hong Kong	China	87
Los Angeles	USA	15
Karbala	Iraq	3.5
Chicago	USA	34
Seattle	USA	40

Country	Mean Rain Fall
China	56
Iraq	2.87
USA	34.5

Merge

- Merging method involves merging the columns of different data frame together based on a column

A	B	C
1		
2		
3		

A	D	E
1		
2		
3		

A	B	C	D	E
1				
2				
3				

Example

- We have two Data frames Student ID numbers, first and last Names
- The second Data frame has the Student ID and there Grades for English and Math
- We would like to create a new data frame that contains the student ID number, first name, last name and grades for math and English

Example

- Data frame one

1. Student ID
2. First name
3. Last Name

In [1]:
```
import pandas as pd
```

In [2]:
```
NameData= {'ID': ['1', '2', '3'],'first_name':['John', 'Sara', 'Bob'], 'last_name': ['Smith', 'Eli','Aby']}
dfA=pd.DataFrame(NameData, columns = ['ID', 'first_name', 'last_name'])
dfA
```

Out[2]:

	ID	first_name	last_name
0	1	John	Smith
1	2	Sara	Eli
2	3	Bob	Aby

BIG DATA UNIVERSITY

Example

- Data frame two

1. Student ID
2. Math Grade
3. English Grade

In [3]:
```
GradeData= {'ID': ['1', '2', '3'],'Math': ['A', 'B', 'C'], 'English': ['B', 'A','B']}
dfB=pd.DataFrame(GradeData, columns = ['ID', 'Math', 'English'])
dfB
```

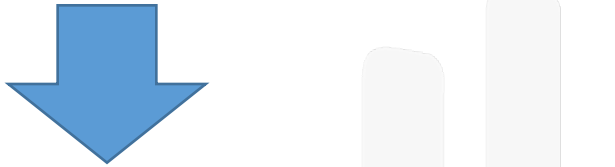
Out[3]:

	ID	Math	English
0	1	A	B
1	2	B	A
2	3	C	B

BIG DATA UNIVERSITY

Example

- We would like to Merge the data set on the id Column



	ID	first_name	last_name
0	1	John	Smith
1	2	Sara	Eli
2	3	Bob	Aby

	ID	Math	English
0	1	A	B
1	2	B	A
2	3	C	B

Example

- We use the following command to merge the data frames

```
In [4]: dfNew=pd.merge(dfA, dfB, on='ID')
        dfNew
```

	ID	Math	English
0	1	A	B
1	2	B	A
2	3	C	B

	ID	first_name	last_name
0	1	John	Smith
1	2	Sara	Eli
2	3	Bob	Aby

	ID	first_name	last_name	Math	English
0	1	John	Smith	A	B
1	2	Sara	Eli	B	A
2	3	Bob	Aby	C	B

BIG DATA UNIVERSITY

Missing values - Methodology

How to deal with missing data?

1. Drop data
   a. drop the variable
   b. drop the data entry

2. Replace data
   a. replace it by mean
   b. replace it by frequency
   c. replace it based on other functions

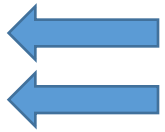
	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheel
0	3	?	alfa-romero	gas	std	two	convertible	rwd
1	3	?	alfa-romero	gas	std	two	convertible	rwd
2	1	?	alfa-romero	gas	std	two	hatchback	rwd

BIG DATA UNIVERSITY

Dropping Rows

```
data = {'name': ['Joe', 'Cat', 'Mike', 'Kim', 'Amy'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'Points': [4, 24, 31, 2, 3]}
df = pd.DataFrame(data, index = ['Day1', 'Day2', 'Day3', 'Day4', 'Day5'])
df
```

	Points	name	year
Day1	4	Joe	2012
Day2	24	Cat	2012
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014



```
df.drop(['Day1','Day2'])
```

	Points	name	year
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014

Dropping Columns

```
df.drop(['Day1','Day2'])
df
```

	Points	name	year
Day1	4	Joe	2012
Day2	24	Cat	2012
Day3	31	Mike	2013
Day4	2	Kim	2014
Day5	3	Amy	2014

```
df.drop('year', axis=1)
```

	Points	name
Day1	4	Joe
Day2	24	Cat
Day3	31	Mike
Day4	2	Kim
Day5	3	Amy

Missing Data :dropna()

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

```
raw_data = {'name': ['Joe', np.nan, 'Tina', 'Mike', 'Amy'],
        'last_name': ['Miller', np.nan, 'Ali', 'Bob', 'Das'],
        'age': [42, np.nan, 36, 24, 73]}
df =pd.DataFrame(raw_data)
df
```

	age	last_name	name
0	42.0	Miller	Joe
1	NaN	NaN	NaN
2	36.0	Ali	Tina
3	24.0	Bob	Mike
4	73.0	Das	Amy

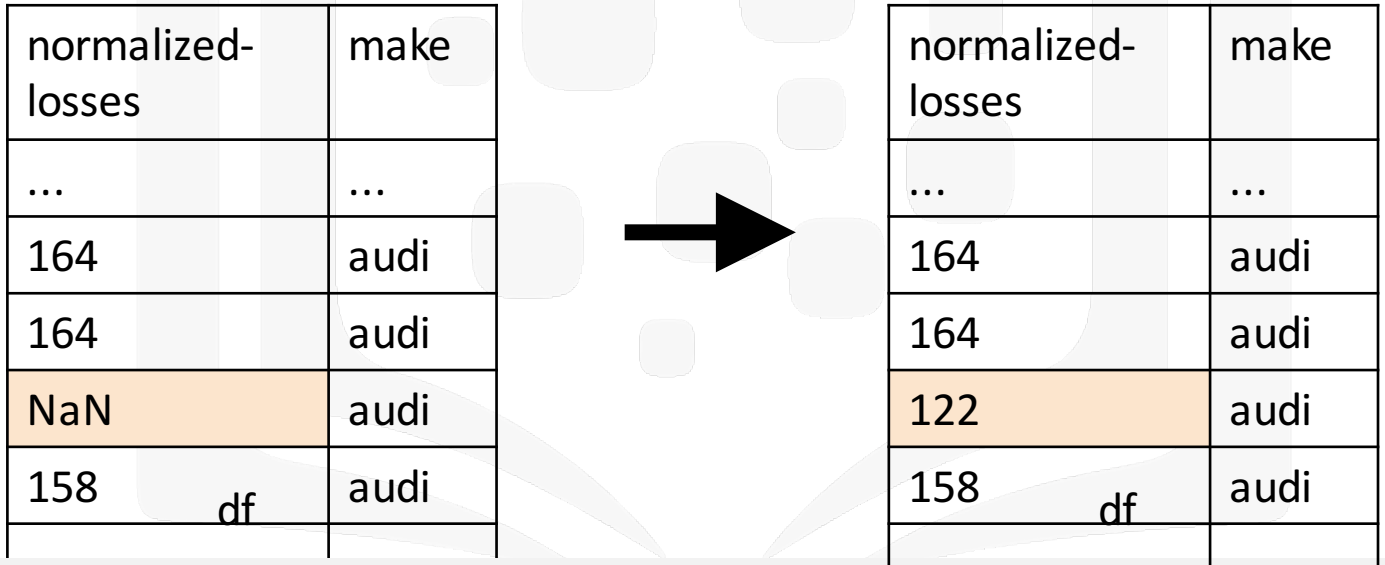
```
df.dropna()
```

	age	last_name	name
0	42.0	Miller	Joe
2	36.0	Ali	Tina
3	24.0	Bob	Mike
4	73.0	Das	Amy

Missing values - Example 2

Replace data

- First, calculate the new value that would be used in replacement,
- Then, use `dataframe.replace(missing value, new value)` method, replace (missing value) by (new value)




```
avg = df["normalized-losses"].astype("int").mean(axis = 0)
df["normalized-losses"].replace(np.nan, avg, inplace = True)
```

Data Normalization

- **Normalization utilization?**

Not-normalized:

- values of “income” is about 1000 times than “age”
- “age” [0-100] and “income” [20000, 500000] are in different range.
- hard to compare “age” and “income”.
- when we do further analysis (regression for example), “income” will influence the result more, compared to “age”, due to its larger range scale.

age	income
20	100000
30	20000
40	500000

before

Normalized:

- similar value range.
- both variables have similar intrinsic influence on analytical model.

age	income
0.2	0.2
0.3	0.04
0.4	1

after

BIG DATA UNIVERSITY

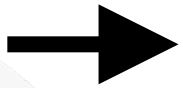
Data Normalization - Implementation

- **Implementation:**
  - Normalize column “length” into the range of [0,1]
  - several ways of normalization

① $x_{new} = \frac{x_{old}}{x_{max}}$ ② $x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$ ③ $x_{new} = \frac{x_{old} - \mu}{\sigma}$

here we use ① and `lambda x: x/x.max()` method

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
...	...df	...



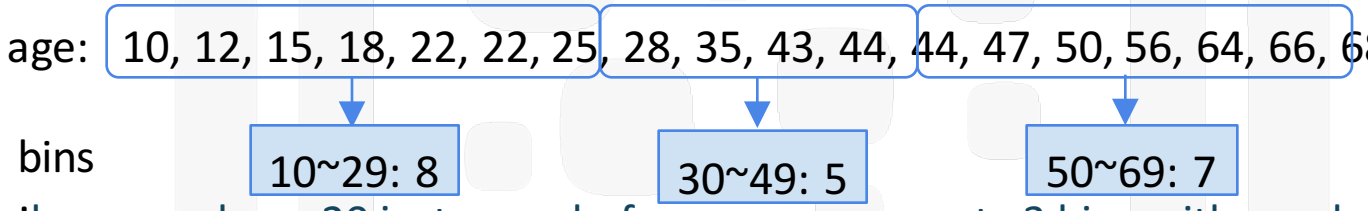
length	width	height
0.811148	64.1	48.8
0.811148	64.1	48.8
0.822681	65.5	52.4
...	...df	...

```
df["length"] = df["length"]/df["length"].max()
```

BIG DATA UNIVERSITY

Binning

- **what is binning?**
  - A process of transforming numerical variables into categorical variables.
  - A process of grouping continuous values into a smaller number of "bins".



  - As shown above, we have 20 instances before, now we create 3 bins with number of values in each bin.

- **Why binning?**
  - Improve accuracy of the predictive models by reducing the noise or non-linearity
  - Easy to identify the outliers (e.g. only one value falls in one certain bin)
  - Numerical variables are usually discretized in the modeling methods based on frequency tables (e.g., decision trees)