

# Pandas

In Python.



# AGENDA

01



## Introduction

What is Pandas ? What are Series and Data Frames ?, Retrieving Series / Data Frames Information

02



## Phase – 1

I/O with Pandas, Selection, Selecting , Boolean Indexing & Setting, Applying Functions

03



## Phase – 2

Dropping, Sorting, Resetting Index, Grouping Data, Missing Data, Combining Data ,Dates

04



## Phase – 3

Pivot Table, Iterations, Multi Indexing, Duplicates Data





## Introduction

What is Pandas ? What are Series and Data Frames ?, Retrieving Series / Data Frames Information

# What is Pandas ?

---

Pandas is used for **data manipulation, analysis and cleaning**. Its is built on NumPy and provides easy-to-use data structures and data analysis tools for the python.



## Installation & Import

```
>>> pip install pandas
```

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



## Pandas: kinds of data

---

Python pandas is well suited for different kinds of data, such as:

- Tabular data with heterogeneously-typed columns
- Ordered and unordered time series data
- Arbitrary matrix data with row & column labels
- Unlabelled data
- Any other form of observational or statistical data sets

# Pandas : Series

A **one – dimensional labelled array** capable of holding any type of data. In terms of data frame, each column is a **Series**

Series are one-dimensional ndarray with axis labels (including time series).

Syntax

```
1 pd.Series(data=None, index=None, dtype=None, name=None)
```

C	O
D	E

```
1 pd.Series(data=[1, 2, 3, 4], index=['A', 'B', 'C', 'D'],  
2          dtype= np.int32 , name='Sample Series')
```

 OUTPUT

```
A    1  
B    2  
C    3  
D    4  
Name: Sample Series, dtype: int32
```

# Pandas : Series Example

Extracting the column of **GDP** as a Series form the Dataset

	Country	Year	Life expectancy	GDP	Population
0	Afghanistan	2015	65.0	584.259210	33736494.0
1	Afghanistan	2014	59.9	612.696514	327582.0
2	Afghanistan	2013	59.9	631.744976	31731688.0
3	Afghanistan	2012	59.5	669.959000	3696958.0
4	Afghanistan	2011	59.2	63.537231	2978599.0



Data Set

GDP
584.259210
612.696514
631.744976
669.959000
63.537231

 **OUTPUT**



```
df.GDP[:5]
```

```
0    584.259210
1    612.696514
2    631.744976
3    669.959000
4     63.537231
Name: GDP, dtype: float64
```

# Pandas : Data Frames

A **Bunch of Series** put together to share the **same index**. Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Syntax

```
pd.DataFrame(data=None, index = None, columns = None, dtype = None)
```

C	O
D	E

```
pd.DataFrame(data=np.random.randn(3,3), index = [11, 22, 33],  
             columns = ['A', 'B', 'C'], dtype = None)
```

 OUTPUT

	A	B	C
11	1.198996	-1.069480	1.855216
22	-0.028789	-0.542810	0.165172
33	-1.513444	-1.917747	-1.380437



# Pandas : Data Frames Example

Slicing the Data Frames from the Data set

	Country	Year	Life expectancy	GDP	Population
0	Afghanistan	2015	65.0	584.259210	33736494.0
1	Afghanistan	2014	59.9	612.696514	327582.0
2	Afghanistan	2013	59.9	631.744976	31731688.0
3	Afghanistan	2012	59.5	669.959000	3696958.0
4	Afghanistan	2011	59.2	63.537231	2978599.0

	Country	Year	Life expectancy	GDP	Population
0	Afghanistan	2015	65.0	584.259210	33736494.0
1	Afghanistan	2014	59.9	612.696514	327582.0
2	Afghanistan	2013	59.9	631.744976	31731688.0

3	Afghanistan	2012	59.5	669.959000	3696958.0
4	Afghanistan	2011	59.2	63.537231	2978599.0

# Pandas : Retrieving Information

---

Basic Information	
df.shape	returns (rows, columns)
df.index	describe index of the data
df.columns	returns the columns of the dataframe
df.info()	info on dataframe
df.count()	number of non – na values.

# Pandas : Retrieving Information

---

Summary	
df.sum()	sum of values
df.cumsum()	cumulative sum of values
df.min() / df.max()	minimum / maximum values
df.idxmin() / df.idxmax()	minimum / maximum index values
df.describe()	summary statistics
df.mean()	mean of the values
df.median()	median of the values



## Phase – 1

I/O with Pandas, Selection, Selecting,  
Boolean Indexing & Setting,  
Applying Functions

# Pandas : I/O

---

## Read and Write to CSV

<code>pd.read_csv('filename.csv', nrows = 5)</code>	Reads a CSV File with 5 rows
<code>Df.to_csv('name_of_dataframe.csv', index = False)</code>	Output the file in csv format
<code>pd.read_excel('filename.xlsx', sheetname = 'sheet1')</code>	Reads a Excel File with sheet1
<code>Df.to_excel('name_of_file.excel', sheet_name = 'Sheet1')</code>	Output in excel format in sheet1
<code>pd.ExcelFile('filename.xls')</code>	Read Multiple sheets from the file.

## Pandas : Selecting, Boolean Indexing, Setting

---

By position	
<code>df.iloc[ [0],[0] ]</code>	Select single value by row & column
<code>df.iat( [0], [0] )</code>	Select single value by row & column
By Label	
<code>df.loc[ [0], ['gdp'] ]</code>	Select single value by row & column labels
<code>df.at( [0], ['gdp'] )</code>	Select single value by row & column labels
By Label / Position	
<code>df.ix[ ]</code>	Select single rows of subset of rows
<code>df.ix[ : , 'gdp' ]</code>	Select single column of subset of columns
<code>df.ix[1 , 'gdp' ]</code>	Select rows and columns

# Pandas : Boolean Indexing, Setting, Apply Functions

---

Boolean Indexing	
<code>s[~ (s &gt; 1)]</code>	Series s where values is not > 1
<code>s[ (s &gt; -1)   (s &gt; 2) ]</code>	S where value is <-1 or > 2
<code>df [ df ['gdp'] &gt; 1893]</code>	Use filter to adjust Data Frame
Setting	
<code>s['a'] = 6</code>	Set index a of Series s to 6
<code>df.at( [0], ['gdp'] )</code>	Select single value by row & column labels
Applying Functions	
<code>f = lambda x: x * 2</code>	
<code>df.apply(f)</code>	Apply function
<code>df.applymap(f)</code>	Apply function element wise



## Phase – 2

Dropping, Sorting, Resetting Index,  
Grouping Data, Missing Data,  
Combining Data ,Dates



# Pandas : Dropping & Sorting

---

## Dropping

`df.drop( ['a','c'] )`

drop values from rows

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

drop values from columns

## Sorting

`df.sort_index()`

sort by labels along an axis

`df.sort_values(by = 'gdp')`

sort by the values along an axis

`df.rank()`

assign ranks to the entries

# Pandas : Resetting Index, Grouping, Missing Data

---

Setting / Resetting Index	
<code>df.set_index('gdp')</code>	Set the Index
<code>df1 = df.reset_index()</code>	Reset the Index
Grouping	
<code>df.groupby(by = ['gdp', 'population']).mean()</code>	Groups the data according to mean values
<code>df.groupby(by = level = 0)</code>	If df is a Multi Index
Missing Data	
<code>df.dropna()</code>	Drop NaN values
<code>df.fillna(df.mean())</code>	Fill NaN values with a predetermined val
<code>df.replace('a', 'b')</code>	Replace values with others
<code>df.any()</code>	Returns bool if any null values exists

# Pandas : Combining Data, dates

---

## Combining Data

`df.join(df2, how = 'right')`

Joining

`pd.merge(df1, df2, how= 'left', on = 'param')`

Merging

`s.append(s2)`

Append the data to series or to df

## Date range

`pd.date_range('2021-06-26', periods = 6)`

Creating a range of dates

`index = pd.datetimeindex(dates)`



## **Phase – 3**

Pivot Table, Iterations, Multi  
Indexing, Duplicates Data

## Pandas : Pivot Table,

---

Pivot	
<code>df = df1.pivot(index = 'date' , columns = 'type', values = 'value')</code>	Spread rows into columns
<code>stacked – df1.stack()</code>	Pivot a label of columns labels
<code>stacked.unstack()</code>	Pivot a label of Index labels
Duplicate Data	
<code>ser1.unique() / df.column.unique()</code>	Return unique elements
<code>ser1.nunique()</code>	Return no of unique elements
<code>df1.duplicated('gdp')</code>	Check duplicates
<code>df1.drop_duplicates('gdp', keep = 'last')</code>	Drop duplicates
<code>df1.index.duplicated()</code>	Check index duplicates

## Pandas : Iteration,

---

Iteration	
df.iteritems()	(Column – Index, Series) pairs
df.iterrows()	(Row – Index, Series) pairs

# THANKS

Do you have any questions?

contact [edufabricaclassroom@gmail.com](mailto:edufabricaclassroom@gmail.com)

Instagram: [@sumitkumar.9912](https://www.instagram.com/sumitkumar.9912)

LinkedIn : [@sumitkumarshukla](https://www.linkedin.com/in/sumitkumarshukla)

