



1. Introduction

Pandas library of python is very useful for the manipulation of mathematical, data and is widely used in the field of machine learning for data analysis.

Why Pandas

- Intrinsic Data alignment.
- Data Operation Functions
- Functions for handling missing data
- Data standardization functions
- Data Structures handling major use cases.

Pandas Features

- Powerful data Structure
- Fast and efficient data wrangling
- Easy data aggregation and transformation
- Tools for reading/ Writing data
- Intelligent and automated data alignment
- High performance merging and joining of data sets

2. Technical Setup

I). Install Anaconda (<https://www.anaconda.com/>)

II). Install Pandas Package

`!pip install pandas`

Import pandas as pd

III). Use Jupyter notebook or Google Colab

IV). Downloading the dataset (e.g. Kaggle,...)

3. Series and DataFrames

Series:

A series is a sequence of data. A series is a one-dimensional array of indexed data. However, a Series does not have a column name, it only has one overall name. Use Series () function.

- One-dimensional labeled array.
- Support multiple data types

Syntax:

```
S = pd.Series(data, index = [index])
```

Accessing a single Series:

```
DataFrame['SeriesName']
```

```
DataFrame["SeriesName"]
```

DataFrame.SeriesName - It does not work if space in SeriesName

Accessing Multiple Series:

```
DataFrame[['SeriesName1','SeriesName2']]
```

```
#pd.Series ([1, 2, 3, 4, 5])
```

```
#pd.Series ([3000, 3500, 4000], index=['2021 price', '2020 price', '2019 price'], name='Index A')
```

DataFrames:

Two-Dimensional data structure, like two-dimensional array, or a table with rows and columns.

Use DataFrame () function.

- Two-dimensional labeled array.

- Support multiple data types
- Input can be a Series
- Input can be another DataFrame.

type (DataFrame) : pandas.core.frame. DataFrame (check dataframe object)

```
#df = pd.DataFrame({'quantity': [10,12], 'price': [1200,1400]})
```

Note: DataFrame entries are not for only integers also dataframe whose values are strings.

```
#df=pd.DataFrame({'Nepal': ['nepal is', 'beautiful', 'country.'], 'Kathmandu': ['Kathmandu is', 'capital', 'of nepal']})
```

Index in DataFrame: The list of row labels used in a DataFrame is known as an Index.

```
#df=pd.DataFrame({'Nepal': ['nepal is', 'beautiful', 'country.'], 'Kathmandu': ['Kathmandu is', 'capital', 'of nepal']},  
index=['A', 'B', 'C'])
```

```
df = pd.read_csv("file location")
```

Parameters	Descriptions	Data Type
Filepath	File location	str
skiprows	For skip rows	Int, list, callable
Usecols	List of column no or name, if callable uses columns where the name passed to the callable result in TRUE.	Callable or list
Index	Indexing	Int, str
Skip_blank_lines	TRUE to skip blank lines rather than reading NAN values. Default is TRUE	bool

<i>Sep</i>	<i>This sep parameter tells the interpreter, which delimiter is used in our dataset or in Layman's term, how the data items are separated in our CSV file.</i>	<i>str</i>
<i>Delimiter</i>	<i>Alish for sep</i>	<i>str</i>
<i>Name</i>	<i>List of columns name to use</i>	<i>Array like</i>

4. Data Input and Validation

Data Input

Functions	Description
read_csv ()	Read CSV file
read_json()	Read JSON file
read_html()	Read HTML file
read_xml()	Read XML file
read_sql()	Read SQL file
read_excel()	Read Excel file
to_csv("file name")	Save DataFrame in CSV file format.

Shape:

The Shape attribute returns a tuple. Representing rows and columns the dimensionality of the DataFrame.

#DataFrame.shape

E.g. df.shape

Out: (rows, columns)

#df.shape[0]

Out: display rows

```
#df.shape[1]
```

Out: display columns

head () and tail() :

#DataFrame.head(n)

Return first n rows of Dataframe.

note: if you not pass any number, display first five rows.

#DataFrame.tail(n)

Return last N rows of Dataframe.

note: if you not pass any number, display last five rows.

info()

info() provides a summary of the data frame, including the number of entries, the data type and the number of non-null entries for each series in the data frame.

#DataFrame.info ()

5. Basic Analysis

value_counts ():

value_counts () method is very useful in pandas. It returns a series object, counting all the unique values in DataFrame. Returns a object containing counts of unique values.

By default, results are in descending order so first element is most frequently occurring element.

```
#Series.value_counts (normalize = False, sort=True, ascending=False, bins=None,  
drope=True)
```

→ you can use above parameters as your needs.

sort_values():

```
#Series.sort_values(axis=0, ascending=True, inplace=False, kind='quicksort',na_position='last')
```

➔ sort values along either axis.

```
#DataFrame.sort_values(by, axis=0, ascending= True, inplace=False,  
kind='quicksort',na_position='last')
```

Boolean Indexing:

➔ Boolean vectors can be used to filter data.

Operator	Symbol
AND	&
OR	
NOT	~
EQUAL-TO	==

➔ Multiple Boolean conditions must be grouped using brackets.

Eg. df[(df.DataFrame.value == 'value') & (df.DataFrame.value == 'value')]

String Handling:

➔ Available to every Series using the str attribute.

➔ Series.str – access values of series as strings and apply several methods to it

Series.str.contains('string value')

#Series.str.startswith()

Series.str.isnumeric()

Indexing:

➔ The index object is an immutable array.

➔ Indexing allows you to access a row or column using a label

#type(DataFrame.index)

#DataFrame.index[20]

set_index():

#DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False)

➔ Set the Dataframe using one or more columns

#set_index(keys, inplace=True)

reset_index():

#DataFrame.reset_index(level=None, drop=False, inplace=False,...)

➔ Returns a DataFrame with default (integer-based) index.

#DataFrame.reset_index(inplace =True)

reset_index():

#DataFrame.sort_index(axis=0, level=None, ascending=True, inplace=False, ... by=None)

➔ Sort objects by a label along the axis.

loc[]:

- `DataFrame.loc [] / Dataframe.Series.loc []`
- A label-based indexer for selection by label
- `loc[]` will raise a `KeyError` when the items are not found

`iloc[]`:

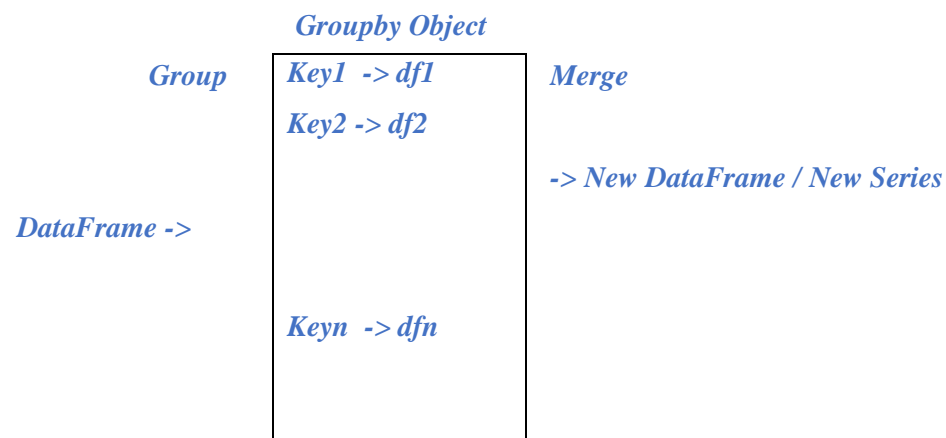
- `DataFrame.iloc[]`
- `iloc[]` is primarily integer position based (from 0 to length-1 of the axis)
- Allows traditional Pythonic slicing.

6. GroupBy

Groupby is one of the most important functionalities available in Pandas. Groupby does three things.

- Split DataFrame into group Based on some criteria.
- Apply a function to each group independently
- Combine the results into a DataFrame.
- Return a groupby object

*#pandas.DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)*



Iterate through a group:

- for key,group in DataFrame.groupby():
 print(key)
 print(group)
 type(group_value)

Groupby Computations:

```
# GroupBy.size()
# GroupBy.count()
# GroupBy.first(),GroupBy.last()
# GroupBy.head(), GroupBy.tail()
# GroupBy.mean()
# GroupBy.max(), GroupBy.min()
agg() -> multiple statistics in one calculation per group
# DataFrame.groupby(agg([...]))
```

7. Reshaping

stack():

```
#DataFrame.stack(level= -1, dropna=True)
```

➔ *Returns a DataFrame or Series.*

Pivot a level of the column labels, returning a DataFrame or Series, with a new innermost level of row labels.

Unstack():

```
#DataFrame.unstack(level=-1, full_value=None)
```


- *Pivot a level of the index labels, returning a DataFrame having a new level of columns labels.*
- *If the index is not a multi-Index, the output will be a Series-the level involved will automatically get sorted*