



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Report

on

Python Pandas Library

(Python Data Analysis Library)



Most Frequently Used Functions

Name of the Student: Rithvika T

Registration Number: 21BCE5554

Programme: B.Tech CSE in I Sem

School: School of Computer Science & Engineering (SCOPE)

Subject / Paper: BCSE101E - Computer Programming in Python

Contents

Section	Page Num
01. What is Python Pandas?	3
02. Key Features of Pandas	3
03. Padas Data Structures	4
04. Functions for Data Science & Data Analysis	4
05. Other set of Functions for Data Science & Data Analysis	4
06. Combining Data Sets	6
07. Reshaping the Data	9
08. Indexing the Data	9
09. Summarizing Data	10
10. Handling Missing Data	11
11. To Make New Columns	12
12. Group Data	12
13. Functions on Window Operation	13
14. Functions for Plotting	13
15. Creating Data Frames	14



Python Pandas Library

Most Frequently Used Functions

01. So, What is Python Pandas ?

“ Pandas- is a Python Data Analysis Library “ (Extensively used for Data Science)

“Pandas” is a fast, powerful, flexible and easy to use open source Data Analysis and Manipulation Library built on top of the Python programming language.

Pandas is defined as an open-source library that provides **high-performance data manipulation** in **Python**. The name of Pandas is derived from the word Panel Data, which means an Econometrics from Multidimensional data.

It is used for data analysis in Python and developed by **Wes McKinney** in **2008**.

Data Analysis requires lots of processing, such as **restructuring, cleaning or merging**, etc. There are different tools are available for fast data processing, such as **Numpy, Scipy, Cython**, and **Panda**. But we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools.

Pandas is built on top of the Numpy package, means **Numpy** is required for operating the **Pandas**.

Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., **load, manipulate, prepare, model, and analyze**.

02. Key Features of Pandas

- It has a fast and efficient **DataFrame** object with the default and customized indexing.
- Used for reshaping and pivoting of the data sets.
- Group by data for aggregations and transformations.
- It is used for data alignment and integration of the missing data.
- Provide the functionality of Time Series.

- Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
- Handle multiple operations of the data sets such as **sub-setting, slicing, filtering, group-by, re-ordering, and re-shaping**.
- It integrates with the other libraries such as SciPy, and scikit-learn.
- Provides fast performance, and If you want to speed it, even more, you can use the **Cython**.

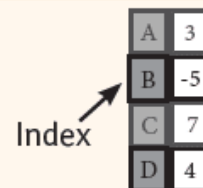
03. Padas Data Structures

Series	A one-dimensional labelled array capable of holding any data types
DataFrame	A two-dimensional labelled data structure with columns of potentially different types

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

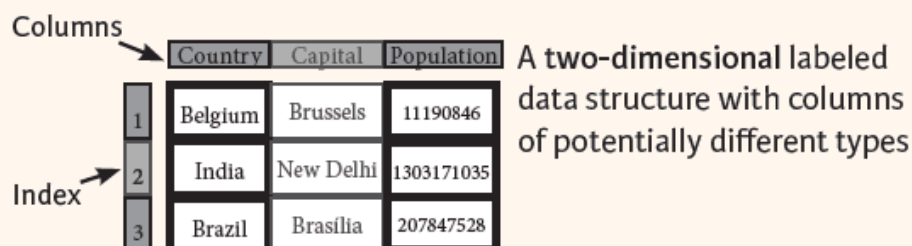


A	3
B	-5
C	7
D	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns



	Country	Capital	Population
1	Belgium	Brussels	11190846
2	India	New Delhi	1303171035
3	Brazil	Brasília	207847528

A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                       columns=['Country', 'Capital', 'Population'])
```

04. Functions for Data Science & Data Analysis

read_csv()	Helps read a comma-separated values (csv) file into a Pandas DataFrame
head(n)	Is used to return the first n rows of a dataset
describe()	is used to generate descriptive statistics of the data in a Pandas DataFrame or Series

memory_usage()	Returns a Pandas Series having the memory usage of each column (in bytes) in a Pandas DataFrame
astype()	Is used to cast a Python object to a particular data type
Loc()	Helps to access a group of rows and columns in a dataset, a slice of the dataset, as per our requirement
Iloc()	iloc are used to select rows and columns ; iloc: select by positions
to_datetime()	converts a Python object to datetime format. It can take an integer, floating point number, list, Pandas Series, or Pandas DataFrame as argument
value_counts()	Returns a Pandas Series containing the counts of unique values
drop_duplicates()	returns a Pandas DataFrame with duplicate rows removed
groupby()	Is used to group a Pandas DataFrame by 1 or more columns, and perform some mathematical operation on it. groupby() can be used to summarize data in a simple manner
sort_values()	Is used to sort column in a Pandas DataFrame (or a Pandas Series) by values in ascending or descending order
fillna()	Helps to replace all NaN values in a DataFrame or Series by imputing these missing values with more appropriate values

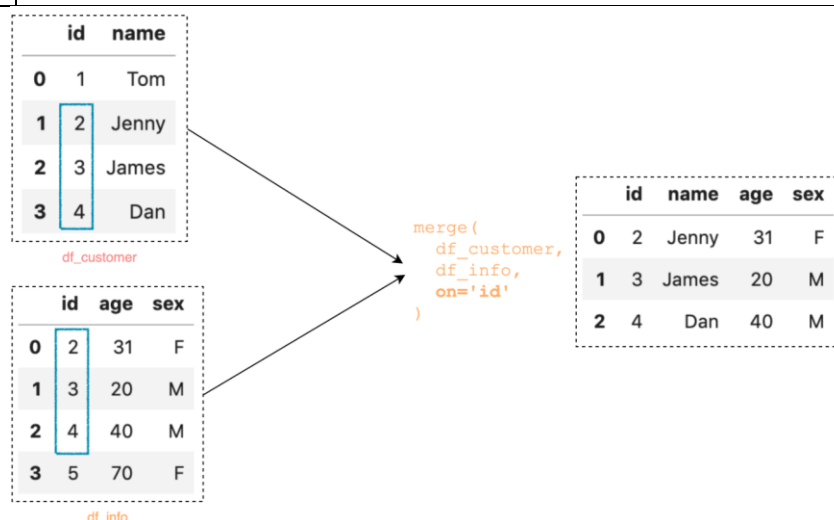
05. Other set of Functions for Data Science & Data Analysis

Query()	We sometimes need to filter a dataframe based on a condition or apply a mask to get certain values. One easy way to filter a dataframe is query function
Insert()	When we want to add a new column to a dataframe, it is added at the end by default. However, pandas offers the option to add the new column in any position using insert function
Cumsum()	The dataframe contains some yearly values of 3 different groups. We may only be interested in yearly values but there are some cases in which we also need a cumulative sum. Pandas provides an easy-to-use function to calculate cumulative sum which is cumsum
Sample()	Sample method allows you to select values randomly from a Series or DataFrame. It is useful when we want to select a random sample from a distribution.
Where()	“Where” is used to replace values in rows or columns based on a condition. The default replacement values is NaN but we can also specify the value to be put as a replacement.
Isin()	We use filtering or selecting methods a lot when working with dataframes. Isin method is kind of an advanced filtering. For example, we can filter values based on a list of selections

Pct_change()	This function is used to calculate the percent change through the values in a series.
Rank()	Rank function assigns rank to the values.
Melt()	Melt() is used to convert wide dataframes to narrow ones.
Explode()	Assume your data set includes multiple entries of a feature on a single observation (row) but you want to analyze them on separate rows.
Nunique()	Nunique counts the number of unique entries over columns or rows. It is very useful in categorical features especially in cases where we do not know the number of categories beforehand.
Lookup()	It can be used to look up values in the DataFrame based on the values on other row, column pairs.
Infer_objects()	Pandas supports a wide range of data types, one of which is object. Object covers text or mixed (numeric and non-numeric) values. Pandas supports a wide range of data types, one of which is object. Object covers text or mixed (numeric and non-numeric) values.
Select_dtypes()	This function returns a subset of the DataFrame's columns based on the condition set on data types
Replace()	As the name suggests, it allows to replace values in a dataframe.
Applymap()	Function is used to apply a function to a dataframe elementwise.

06. Combining Data Sets

merge()	Combining Data on Common Columns or Indices
append()	This function concatenate along axis=0, namely the index
join()	Combining Data on a Column or Index
concat()	Combining Data Across Rows or Columns ; Merge or Append Tables using concat() by row or column



merge(): Combining Data on Common Columns or Indices

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd

	Name	Age	Address	Qualification
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons



	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

append(): function this function concatenate along axis=0, namely the index

pd.concat(['tabel1,table2])

Item_id	Item
11	Sweet Pickle
22	Salty Pickle
33	Japanese Pickle
44	German Pickle
55	Korean Pickle



Item_id	Item
88	Chocalate Ice Cream
99	Vanilla Ice Cream
66	Strawberry Ice Cream
77	Coffe Ice Cream



Item_id	Item
88	Chocalate Ice Cream
99	Vanilla Ice Cream
66	Strawberry Ice Cream
77	Coffe Ice Cream
11	Sweet Pickle
22	Salty Pickle
33	Japanese Pickle
44	German Pickle
55	Korean Pickle

Concat(): Combining Data Across Rows or Columns ; Merge / Append Tables using concat() by row or column

Combine Data Sets

adf		bdf	
x1	x2	x1	x3
A	1	A	T
B	2	B	F
C	3	D	T



Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

```
pd.merge(adf, bdf,
         how='left', on='x1')
```

Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

```
pd.merge(adf, bdf,
         how='right', on='x1')
```

Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

```
pd.merge(adf, bdf,
         how='inner', on='x1')
```

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

```
pd.merge(adf, bdf,
         how='outer', on='x1')
```

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

```
adf[adf.x1.isin(bdf.x1)]
```

All rows in adf that have a match in bdf.

x1	x2
C	3

```
adf[~adf.x1.isin(bdf.x1)]
```

All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3



zdf

x1	x2
B	2
C	3
D	4



Set-like Operations

x1	x2
B	2
C	3

```
pd.merge(ydf, zdf)
```

Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

```
pd.merge(ydf, zdf, how='outer')
```

Rows that appear in either or both ydf and zdf (Union).

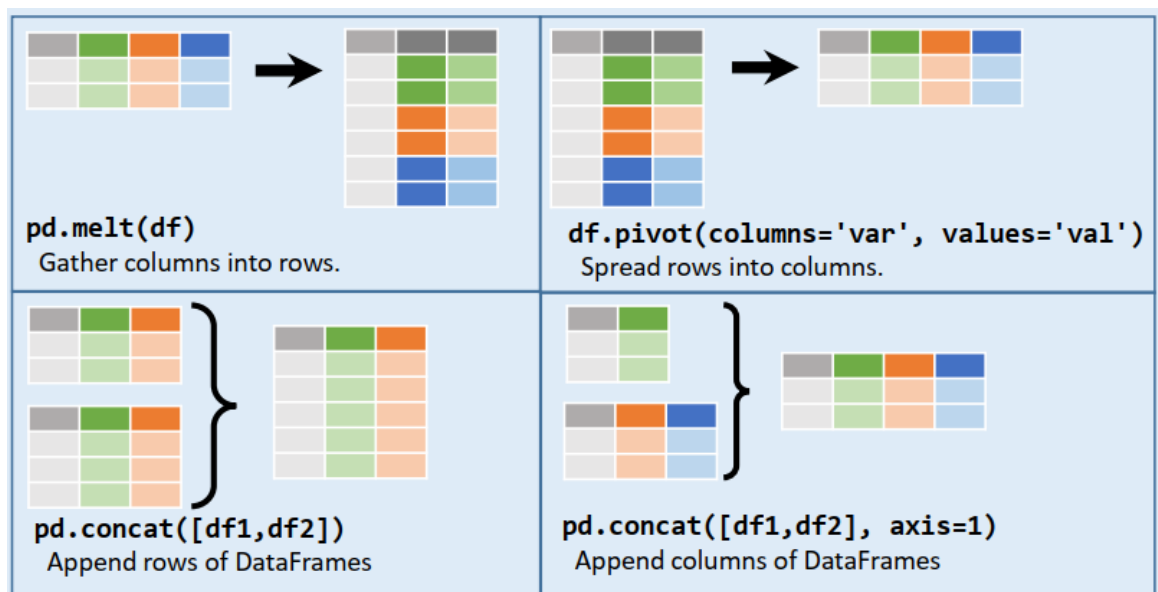
x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```

Rows that appear in ydf but not zdf (Setdiff).

07. Reshaping the Data

melt(df)	Gather columns into rows
pivot()	<code>pivot(columns='var', values='val')</code> : Spread rows into columns
sort_values()	Order rows by values of a column (low to high); <code>sort_values('mpg',ascending=False)</code> : Order rows by values of a column (high to low)
rename()	<code>rename(columns = {'y':'year'})</code> : Rename the columns of a DataFrame
sort_index()	Sort the index of a DataFrame
reset_index()	Reset index of DataFrame to row numbers, moving index to columns.
drop()	<code>drop(columns=['Length','Height'])</code> : Drop columns from DataFrame



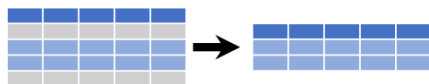
Difference between `melt()`, `pivot()`, `concat()` functions

08. Indexing the Data

length()	<code>df[Length > 7]</code> : Extract rows that meet logical criteria.
drop_duplicates()	Remove duplicate rows (only considers columns).
head(n)	Select first n rows.
tail(n)	Select last n rows.
sample(frac=0.5)	Randomly select fraction of rows; sample(n=10) - Randomly

	select n rows.
nlargest()	nlargest(n, 'value'): Select and order top n entries.
nsmallest()	nsmallest(n, 'value') : Select and order bottom n entries
width()	df['width'] or df.width : Select single column with specific name.
length(), species()	df[['width','length','species']] : Select multiple columns with specific names.
filter()	df.filter(regex='regex') : Select columns whose name matches regular expression regex

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Subset Variables (Columns)



df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] or df.width
Select single column with specific name.

df.filter(regex='regex')

09. Summarizing Data

value_counts()	df['w'].value_counts() : Count number of rows with each unique value of variable
len(df)	len(df) # of rows in DataFrame.
nunique()	df['w'].nunique() : # of distinct values in a column
describe()	df.describe() : Basic descriptive statistics for each column (or GroupBy)
sum()	Sum values of each object.
count()	Count non-NA/null values of each object.
median()	Median value of each object.
quantile()	quantile([0.25,0.75]) Quantiles of each object.
apply()	apply(function) Apply function to each object
min()	Minimum value in each object.
max()	Maximum value in each object.
mean()	Mean value of each object.

var()	Variance of each object.
std()	Standard deviation of each object

Summarize Data

df['w'].value_counts()

Count number of rows with each unique value of variable

len(df)

of rows in DataFrame.

df['w'].nunique()

of distinct values in a column.

df.describe()

Basic descriptive statistics for each column (or GroupBy)



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

sum()

Sum values of each object.

count()

Count non-NA/null values of each object.

median()

Median value of each object.

quantile([0.25,0.75])

Quantiles of each object.

apply(function)

Apply function to each object.

min()

Minimum value in each object.

max()

Maximum value in each object.

mean()

Mean value of each object.

var()

Variance of each object.

std()

Standard deviation of each object.

10. Handling Missing Data

dropna()	Drop rows with any column having NA/null data.
fillna(value)	Replace all NA/null data with value.

Handling Missing Data

df.dropna()

Drop rows with any column having NA/null data.

df.fillna(value)

Replace all NA/null data with value.

11. To Make New Columns

assign()	df.assign(Area=lambda df: df.Length*df.Height) Compute and append one or more new columns
Volume : Length(),Height(),Depth()	df['Volume'] = df.Length*df.Height*df.Depth Add single column
qcut()	pd.qcut(df.col, n, labels=False) Bin column into n buckets
clip()	clip(lower=-10,upper=10) Trim values at input thresholds

Make New Columns



df.assign(Area=lambda df: df.Length*df.Height)

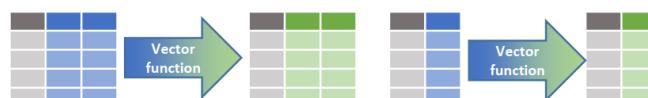
Compute and append one or more new columns.

df['Volume'] = df.Length*df.Height*df.Depth

Add single column.

pd.qcut(df.col, n, labels=False)

Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

max(axis=1)

Element-wise max.

min(axis=1)

Element-wise min.

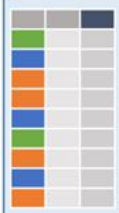
clip(lower=-10,upper=10) abs()

Trim values at input thresholds Absolute value.

12. Group Data

size()	Size of each group
groupby()	df.groupby(by="col") Return a GroupBy object, grouped by values in column named "col". df.groupby(level="ind") Return a GroupBy object, grouped by values in index level named "ind".
agg()	agg(function) Aggregate group using function
shift()	shift(1) Copy with values shifted by 1 shift(-1) Copy with values lagged by 1
rank()	rank(method='dense') Ranks with no gaps. rank(method='min') Ranks. Ties get min rank. rank(pct=True) Ranks rescaled to interval [0, 1] rank(method='first') Ranks. Ties go to first value
cumsum()	Cumulative sum
cummax()	Cumulative max
cummin()	Cumulative min
cumprod()	Cumulative product

Group Data



df.groupby(by="col")
Return a GroupBy object, grouped by values in column named "col".

df.groupby(level="ind")
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.

Additional GroupBy functions:

size()
Size of each group.

agg(function)
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

shift(1)
Copy with values shifted by 1.

rank(method='dense')
Ranks with no gaps.

rank(method='min')
Ranks. Ties get min rank.

rank(pct=True)
Ranks rescaled to interval [0, 1].

rank(method='first')
Ranks. Ties go to first value.

shift(-1)
Copy with values lagged by 1.

cumsum()
Cumulative sum.

cummax()
Cumulative max.

cummin()
Cumulative min.

cumprod()
Cumulative product.

13. Functions on Window Operation

expanding()	Return an Expanding object allowing summary functions to be applied cumulatively
rolling(n)	Return a Rolling object allowing summary functions to be applied to windows of length n

Windows

df.expanding()

Return an Expanding object allowing summary functions to be applied cumulatively.

df.rolling(n)

Return a Rolling object allowing summary functions to be applied to windows of length n.

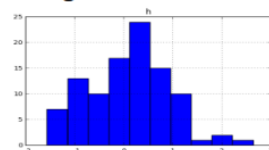
14. Functions for Plotting

hist()	df.plot.hist() Histogram for each column
scatter()	df.plot.scatter(x='w',y='h') Scatter chart using pairs of points

Plotting

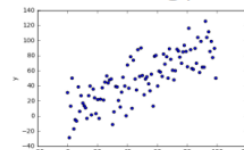
df.plot.hist()

Histogram for each column



df.plot.scatter(x='w',y='h')

Scatter chart using pairs of points



15. Creating Data Frames

DataFrame() with index	Specify values for each column
DataFrame() with index & columns	Specify values for each row
DataFrame() with MultiIndex.from_tuples()	Create DataFrame with a MultiIndex

Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(  
    [[4, 7, 10],  
     [5, 8, 11],  
     [6, 9, 12]],  
    index=[1, 2, 3],  
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = pd.MultiIndex.from_tuples(  
        [('d',1),('d',2),('e',2)],  
        names=['n','v']))
```

Create DataFrame with a MultiIndex

END

Thank You