
Pandas

Data Structures in Pandas

1. Series

   → 1D labeled array.

2. DataFrame

   → 2D labeled array that stores data like a table with rows and column.

Series

→ Can hold any datatype.

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

→ data can be Python dict,
ndarray,
scalar value

→ Index should be list of indexes

→ For dict data, Indexes are taken from dict.

s.dtype → Returns data type

s.array → Returns in an array list

s.to_numpy() → Returns a ndarray.

s[" index_name"]

↳ To access or assign data to an index

"index_name" in s

↳ Returns boolean value

s.get (" index_name ",
0) #Optional

↳ Returns 0 if index is not found (By default np.nan)

```
s.iloc[0]          # Returns 0th index
s.iloc[:3]         # Returns 0, 1, 2 index
s.iloc[[4,3,1]]    # Returns 4, 3, 1 index
s[s > s.median()]  # Returns values greater than median value.
```

Operations

```
s + s
s * 2
np.exp(s)
```

→ Series automatically aligns data based on label.

```
s.iloc[1:] + s.iloc[:-1]
```

output :

a	NaN
b	-0.53
c	-0.42
d	-0.63
e	NaN

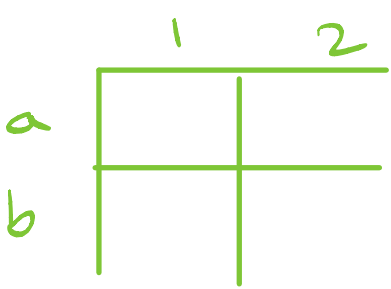
```
s.name    # Name a series
s.rename  # Rename a series
```

Data Frame

→ Dict of 1D ndarrays,
list,
dict,
Series

→ 2D ndarray
→ Series

```
pd.DataFrame( data,        # data
              index=["a","b"],   # Row
              columns=["1","2"]) # Column
```



```
df.index      # Returns index names
df.columns    # Returns column names

df["column_name"]   # Returns column

df["three"] = df["one"] + df["two"]
                          *
                          /
                          -

df["flag"] = df["one"] > 2
```

Boolean Column

```
del   df["column_name"]
col = df.pop("three")
```

Removes column

```
df["column_name"]["index"]
```

Ex:- `df["three"][:2]`

Insert Column

```
df.insert(1,          # Insert at
          "bar",      # Column number
          df["one"])  # Column data
```

Select column	`df[col]`
Select row by label	`df.loc[label]`
Select row by index	`df.iloc[loc]`
Slice rows	`df[5:10]`
slice rows by boolean vector	`df[bool_vector]`

```
df1 + df2

df1 - df2

df1 * df2

df1 / df2

df1 & df2

df1 | df2

df1 ^ df2

-df1

df.T    # Transpose
```

Head() & Tail()

→ head(n) returns top n rows of the dataframe

→ tail(n) returns the bottom n rows of dataframe.

Default value of n: 5

df.shape

↳ Returns the shape of data Frame.

```
pd.date_range( "2000",  #Start date
              periods=2,  #no.of days
              tz = "CET" ) #Time zone
```

numexpr
bottleneck

} Libraries that speedup pandas.

```
pd.set_option("compute.use_bottleneck", False)
pd.set_option("compute.use_numexpr", False)
```

```
df1.sub( df2 ,          # Dataframe
         axis = 0)      # Axis
```

- .add( )
- .mul( )
- .div( )
- .radd( )
- .rsub( )

} For binary operations

axis can be 0 / "index"
1 / "column"

```
df.copy()    # Copy to new dataFrame
df.empty     # To check if df is empty
```

```
df1.eq(df2)  # Equals
df1.ne(df2)  # Not equals
df1.lt(df2)  # less than
df1.gt(df2)  # greater than
df1.le(df2)  # less than or equals
df1.ge(df2)  # greater than or equals
```

Note: `np.nan == np.nan #False`

→ Series or DataFrame index needs to be in same order

Ex: df1.equals(df2) # False

df1.equals(df2.sort_index()) # True

→ While comparing, Make sure that both are of same length.

df1.combine_first(df2)

→ Combines 2 dataframes

→ Replaces all nan values in df1 with df2 values.

df1.combine(df2, combiner)

Custom combiner function.

Descriptive Statistics

```
df.mean( axis=0,  # Axis
         skipna=False) # Default=True
```

count()	Number of non Nan values
sum()	Sum
mean()	Mean
median()	Median
min()	Minimum value
max()	Maximum value
mode()	Mode
abs()	Absolute
prod()	Product
std()	Standard deviation
var()	Variance
sem()	Standard error of mean

skew()	Skewness (3rd moment)
kurt()	kurtosis (4th moment)
quantile()	Sample quantile
cumsum()	Cumulative sum
cumprod()	Cumulative product
cummax()	Cumulative maximum
cummin()	Cumulative minimum

Describe

→ describe() computes variety of summary statistics about series or dataFrame (Excludes NA values)

```
df.describe(percentiles = [0.05, 0.25, 0.75],
            include = ["objects", "number"])
```

→ Set percentiles and type of objects to describe.

```
df["A"].idxmin() # Returns min value index
df["B"].idxmax() # Returns max value index
```

```
S.value_counts()
# Returns the frequency of each value.
```

`pd.cut()`
`pd.qcut()`
→ To make categories for numerical data.

Ex: `[(0,1], (-1, 0], (0,1], (0,1], ...]`

```
df.appy( funcl )
```

→ To appy a custom function to the dataframe.

→ Can also apply lambda functions

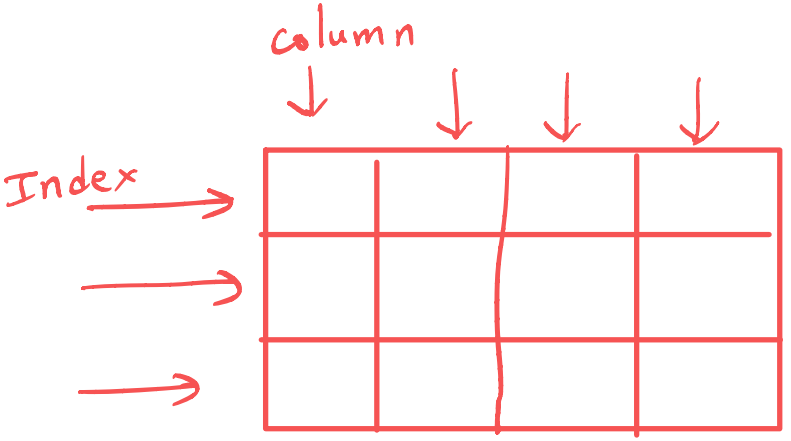
Ex:- df.apply(lambda x: np.mean(x))

df.apply(np.exp)

df.apply(np.cumsum, axis=0)

df.apply("mean", axis=1)

```
df.iloc[3:7] = np.nan
# Assign np.nan values to indexes 3 to 7
```



```
df["A"].agg(["sum", "mean"])
    # Returns both sum & mean values
```

→ Can also make custom describe using agg

```
df.agg(["sum", "mean", "min", ...
    ..., lambda x: x.max()])
```

```
df.transform([fn1, fn2])
```

→ Similar to agg

```
df.map(f)
```

→ Similar to agg

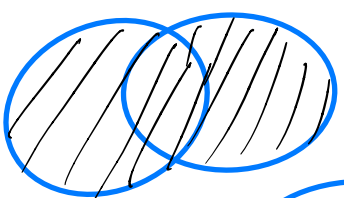
```
s.reindex(index=["e", "b", "f", "d"],
          columns=["three", "one", "two"])
```

→ To change the order of columns

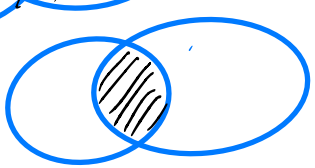
→ To change the order of indexes

Joins

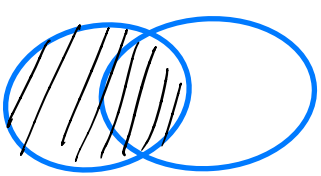
```
s1.align(s2)
```



```
s1.align(s2, join="inner")
```



```
s1.align(s2, join="left")
```



```
s1.align(s2, join="inner",
         axis=0) # Optional
```

Reindex filling

```
df2.reindex(df1,
            method = "ffill",
            limit = 1,
            tolerance = "1 day")
```

→ Method

ffill	forward fill
bfill	backward fill
nearest	nearest fill

→ limit

limits the no.of continuous fills

→ Tolerance

Max distance between current index and indexer value (filler index)

Drop

```
df.drop( ["a", "b"],
         axis = 0)
```

```
df.rename( columns= { "old" : "new" },
           index= { "old1" :"new1", ... },
           axis = 0) # optional
```

```
df.iterrows()
df.itertuples()
df.items
```

} Useful for for-loop iterations

Sorting

```
df.sort_index( ascending= False, #Optional
               axis= 0) #Optional
```

↳ Sort by index

```
df.sort_values (by=["one", "two"])
```

↳ sort by values (by column names)

```
s.nsmallest(3) # Returns 3 smallest values
s.nlargest(4) #Returns 3 largest values
```

Object Conversion

```
to_numeric()
to_datetime()
to_timedelta()
```

I/O

.read_*( ) .to_*( )

Ex:

```
df = pd.read_csv("filename",
                 dtype = object,
                 names = ["foo"],
                 header = 0
                 usecols = ["b", "d"],
                 comment = "#",
                 skip_blank_lines = False,
                 skiprows = 4,
                 encoding = "latin-1")
```

names specify column names

header indicates whether to use

headers in csv file.

usecols specify to choose subset of column names

skiprows skips top n rows

encoding specifies type of encoding

Concatenate

```
pd.concat ( [df1, df2, df3],
            axis = 0,
            sort = False,
            ignore_index = False,
            keys = ["x", "y", "z"])
```

keys

→ Overrides the index/column names creating new dataframe

Merge

```
pd.merge( df1, df2,
          how = "left"
          on = ["key1", "key2"])
```

how
→ left , right , outer , inner , cross

```
pd.merge_ordered( df1, df2,
                  fill_method = "ffill",
                  left_by = "s")
```

```
pd.merge_asof ( df1, df2 ,
                on = "time" ,
                by = "col1 ")
```

Compare

→ compares 2 dataframes / Series

```
df1.compare(df2,
            align_axis=0,
            keep_shape=True,
            keep_equal=True)
```