## Pandas

import pandas as pd

## Creating Data

There are 2 core objects in pandas.

- L> Data Frame
- L> Series

## Data Frame

=> pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]})

| | Yes | No |
|---|---|---|
| 0 | 50 | 131 |
| 1 | 21 | 2 |

=> pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],

'Sue': ['Pretty good.', 'Bland.']})

| | Bob | Sue |
|---|---|---|
| 0 | I liked it | Pretty good |
| 1 | It was awful | Bland |

Up until this point index was by default starting with 0 and was increasing value.

## Assigning Customized index

⇒ pd.DataFrame({'Bob': ['I liked it.', 'It was awful'], 'Sue': ['Pretty good', 'Bland.']}, index=['Product A', 'Product B'])

| | Bob | Sue |
|---|---|---|
| Product A | I liked it ~~Pretty good~~ | Pretty good |
| Product B | It was awful ~~Pretty good~~ | bland |

# Index

## Series

=> pd.Series([1, 2, 3, 4, 5])

series consists of only one column

```
0  1
1  2
2  3
3  4
4  5
dtype: int64
```

Also part of output

## Assigning Customized Index & an overall name

=> pd.Series([2100, 2200, 2300], index = ["2021 sales", "2015 sales", "2027 sales"], name = "Sales")

## Reading data files

⇒ wine_reviews = pd.read_csv("... address list.csv")

only assigning part is done

## shape

⇒ wine_reviews.shape

(129971, 14)

↓ row number of dataset

↳ column number

## Examining the DataFrame

⇒ wine_reviews.head()

(first 5 ta row ashbe)

Index define না করলে automatic ভাবে 1

index বসে।

A Pointing specific column of the table as Index.

wine_reviews = pd.read_csv("...address link.csv"
, index_col=0 )

0th column will be index.

Saving DataFrame as CSV file in harddisk

⇒ animals.to_csv("animals_in_Rangpur.csv")

DF

(এই নামে save হবে।

## Native Accessors

→ suppose reveiws is a DataFrame and country is one of it's objects (columns)

=> reviews.country

Native Accessor

O/P:

| | |
|---|---|
| 0 | Italy |
| 1 | Portugal |
| ... | ... |
| 129969 | china |
| 129970 | Nepal |

Turned a DF into a series

→ columns can also be treated as keys of dictionary as in Python.

=>reviews['country']

O/P:

same

2025.01.16 08:56

=> reviews['country'][0]

O/P

'Italy'

This can't be done by reviews.country

## Indexing in pandas

Pandas indexing works in one of two paradigms.

→ Index-based selection
→ Label-based "

## Index-based selection

→ selecting data based on it's numerical position. iloc operator is used.

=> reviews.iloc[0]
↓ [ → location
index

0th row of the data

row first column second

O/P':

country Italy

Description blah blah

...

Variety White

Flag ~~@~~ RWR

Name: 0, Length: number-of-rows, dtype: object

=> reviews.iloc[:,0]

0th column ની બધા element બતાવશે.

0 Italy

1 France

...

129969 Spain

129970 Portugal

Name: Length dtype

=> reviews.iloc[:3,0]

↓

~~પહેલા બધા 3 ની element~~

0 Italy

1 France

2 USA

0 element થી (3-1) element

= 2

=> reviews.iloc[1:3, 0]

OP:

1 France

2 USA

1st row (3-1)th element = 2nd

=> reviews.iloc[[0,1,2], 0]

0th element থেকে show করবে

=> reviews.iloc[-5:]

129966 [illegible] v

129967 pw

129968 [illegible]

129969 y

129970 z

[illegible] row থেকে 5 element দেখাবে।

## Label-based selection

loc operator is used

=> reviews.loc[0, 'country']

O/P :

'Italy'

0 row ~~column~~ ~ ની country column ને select

કરવા

=> reviews.loc[:, ['A', 'B', 'C']]

O/P :

(0-n) all row and specific 'A', 'B', 'C' column ની

data show કરવા

size = ~~n x 3~~ (n+1) x 3

⇒ reviews.iloc[0:10]

O/P

0
1
2
3
4
5
6
7
8
9

end

⇒ reviews.loc[0:10]

O/P

0
1
2
3
4
5
6
7
8
9
10

## Manipulating the index :

=> reviews.set_index("title")

O/P

title name এর একটি column আসে। যেটি এখন index হিসেবে কাজ করছে। সুন্দর table এর মত দেখাচ্ছে।

## Conditional selection :

=> reviews.country == 'Italy'

O/P :

সবগুলো 'Italy' আছে এমন row এর country column এ এমনভাবে ~~show করছে~~ যদি True & অন্যগুলো যদি False show করছে।

=> reviews.~~loc~~ loc[reviews.country == 'Italy']

=> reviews.loc[0]

2025.01.16 08:57

'Italy' এর সব column দেখাবে।

=> reviews.loc[(reviews.country == 'Italy') &
(reviews.points >= 90)]

=> reviews.loc[(revelws.country == 'Italy') |
(revlws.points >= 90)]

## Selecting data in a list of values:

=> reviews.loc[reviews.country.isin(['Italy',
'France'])]

## Selecting Null or not Null values

=> reviews.loc[reviews.price.notnull()]

=> reviews.loc[reviews.price.isnull()]

## Assigning data

⇒ reviews['critic'] = 'everyone'

O/P:

critic name ↳ একটা column create হবে

সেই column এ আগে যতগুলো row ছিল

সবগুলোতে everyone বসে যাবে।

| | critic |
|---|---|
| 0 | everyone |
| 1 | everyone |
| 2 | everyone |
| 3 | everyone |
| ... | ... |
| n | everyone |

⇒ reviews['blew'] = range(len(reviews), 0, -1)

↓ n থেকে শুরু ↓ -1 করে যোগ

| | |
|---|---|
| 0 | n |
| 1 | n-1 |
| ... | ... |
| n-1 | 2 |
| n | 1 |

↓ 0 এর আগপর্যন্ত যাবে।

2025.01.16 08:58

## Summary Functions

### .describe()

⇒ reviews.points.describe()

points ↓ column of reviews table

O/P

```
count    129971.00
mean        88.447...
 - - - .
75%         91.00
max        100.00
Name: points, Length: 8, dtype: float.
```

⇒ reviews.taster_name.describe

O/P

```
count     103727
unique        19
top       Roger...
freq       25514
```

.describe() is data-type aware, that's why we got different summary

for ~~different~~ numerical & string column.

=> reviews.points.mean()

O/P

88.447

=> reviews.taster_name.unique()

O/P

taster_name column ও unique value

গুলো আসবে

=> reviews.taster_name.value_counts()

O/P

Unique name ~~frequency~~ how many times

Roger Voss 25514

Michael schachner 15134

for ~~different~~ numerical & string column.

=> reviews.points.mean()

O/P

88.447

=> reviews.taster_name.unique()

O/P

taster_name column ~~ও~~ unique value

~~খুঁজে~~ বের করে

=> reviews.taster_name.value_counts()

O/P

Unique name ~~frequency~~ how many times

Roger Voss 25514

Michael schachner 15134

## Maps

Takes one set of values and "maps" them to another set of values.

There are two mapping methods.

-> map() [slightly simpler]

-> apply()

### map()

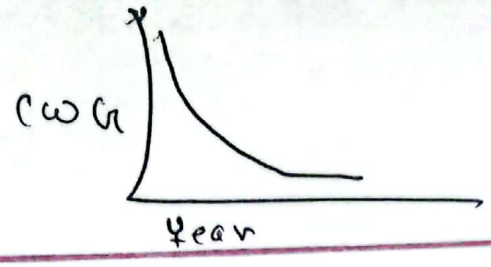
=> review_points_mean = reviews.points.mean()

reviews.points.map(lambda p: p - review_points_mean)


maps ~~are~~ return a new series.

### apply()

To return a whole dataFrame apply() is used.




=> def remean_points(row):

    row.points = row.points - review_points_mean

    return row

reviews.apply(remean_points, axis='columns')

O/P

reviews.apply func এর ভেতর যে আর্গুমেন্ট remean_points func call হচ্ছে।

সেজন্য আমরা row.points কে change করতে পারি।

Result হিসাবে point column modified হয়ে dataframe return হবে।

=> review_points_mean = reviews.points.mean()

reviews.points - review_points_mean

=> reviews.country + "-" + reviews.region_1

Grouping

=> reviws.groupby('points').price.min()

O/P

points অনুযায়ী group করে যার price সবচেয়ে কম ঐ group এর price দেখাবে।

price.min() না দিলে শুধু price দেখাবে।

=> reviews.groupby('winery').apply(lambda df:
df.title.iloc[0])

O/P

প্রত্যেক unique winery এর set এর first element দেখাবে।

## Multiple Groupby

=>reviews.groupby(['country','province']).

apply(lambda df: df.loc[df.points.idxmax()])

O/P:

Country অনুযায়ী group হয়ে সেখানে province অনুযায়ী group হবে। Then যে loc এর point max সেখানকার এর data দেখাবে।

## Aggregate Functions using Groupby

=>reviews.groupby(['country']).price.agg(

[len, min, max])

O/P

কোনো group এর len, min price, max price wise ~~sort~~ O/P 2021-01-13 08:59

## Multi-indexes

=> countries_reviewed = reviews.groupby
(['country', 'province']).description.agg([len])

O/P

len doesn't count the number of char instead how many descriptions there are.

## Converting back to regular index

=> countries.reviewed.reset_index()

## Sorting

=> countries_reviewed .sort_values(by='len')

O/P

groupby ব্যবহার করে sort করে 'len' এর ভিত্তিতে

Ascending Order ও . 1 → ∞

## Deseending Order

countries_reviewed. sort - values (by='len',

ascending= False)

## ~~Sorty~~ Sorting by index

countries_ reviewed . sort-index()

N.B: By default index ই sort থাকে। তাহলে
অন্য কোনো sorting technique use করলে তাহলে
ব্যবহার করে।

## Sorting by Multiple Columns

=> countries_reviewed , sort_values ( by = ['country',
'len'])

2025.01.16 08:59

# Data Types & Missing Values

## dtype('float64')

=> reviews.price.dtype

O/P

dtype('float64')

=> reviews.dtypes

O/P

| | |
|---|---|
| country | object |
| description | object |
| . . . | |
| variety | object |
| winery | object |

## Missing data

NaN → Not a Number

NaN is always float64 dtype

⇒ reviews[pd.isnull(reviews.country)]

reviews → DF

country → column

## Replacing missing values with fillna()

⇒ reviews.region_2.fillna("Unknown")

O/P

~~region_2~~

0 Unknown

1 Unknown

...

n-1 "

n "

## Replacing non-null value

DF ↑ column ↑

=> reviews.taster_twitter_handle.replace

("@balsal", "@bal2sal2")

## Changing dtype

=>reiews.points.astype('float64')

## Renaming

=> reviews.rename(columns = {'points': 'score'})

previous changed

## Renaming entries

which entry to change

=> revIews.rename(index = {0:'firstEntry',

1: 'secOnd.Entry'})

column

change to what

## Renaming index

⇐> reviews. rename_axis("wines", axis='rows')

.rename_axis("fields", axis='columns')

By Default

| columns / Rows | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 0 | - | - | - | - | - | |
| 1 | - | - | | - | - | |
| 2 | - | - | | - | - | |
| 3 | | | | | | |

This will change to:

| fields / wines | |
|---|---|
| | |

Rest is same

## Combining

### concat()

=> canadian_youtube = pd.read_csv("link.csv")

bangladeshi_youtube = pd.read_csv("link2.csv")

pd.concat([canadian_youtube, bangladeshi_youtube])

In this case the ~~csv file~~ columns of both dataFrames or series must be same.

left = can

### join()

=> left = can_yt.set_index(['title', 'trending_date'])

right = bd_yt.set_index['title', trending_date])

left.join(right, lsuffix = '_CAN', rsuffix='_BD')

left DF এর সাথে যোগ করবো — right DF যোগ করবো — left এর ডানপাশে '_CAN' যোগ হবে — right এর ডানপাশে '_BD' যোগ হবে

2025.01.16 08:00