Pandas

March 1, 2024

- index -> axis=0
- columns -> axis=1

1 Initialisation

[1]:

```
import pandas as pd
import numpy  as np
from IPython.display import HTML, Markdown, Latex



def display_df(tp_df=None, index=False):
    # tp_df = tp_df if isinstance(tp_df,pd.DataFrame) else df
    tp_df = tp_df if tp_df is not None else df
    display(Markdown(tp_df.to_markdown(index=index)))

# def display_df(tp_df=None, index=False):
#     tp_df = tp_df if isinstance(tp_df,pd.DataFrame) else df
#     display(HTML(tp_df.to_html(index=index)))
```

2 Creating a dataframe

[2]:

```
data = {
    'age':             [10,22,13,21,12,11,17],
    'section':         ['A','B','C','B','B','A','A'],
    'city':           ␣
  ↪['Gurgaon','Delhi','Mumbai','Delhi','Mumbai','Delhi','Mumbai'],
    'gender':          ['M','F','F','M','M','M','F'],
    'favorite_color': ['red','black','yellow','pink','black','green','red']
}

data_csv = [
    ['age', 'section', 'city',    'gender', 'favorite_color'],
    [10,    'A',       'Gurgaon', 'M',      'red'            ],
    [22,    'B',       'Delhi',   'F',      'black'          ],
    [13,    'C',       'Mumbai',  'F',      'yellow'         ],
    [21,    'B',       'Delhi',   'M',      'pink'           ],
    [12,    'B',       'Mumbai',  'M',      'black'          ],
    [11,    'A',       'Delhi',   'M',      'green'         ],
    [17,    'A',       'Mumbai',  'F',      'red'           ]
]

data_dict = [
    {'age': 10, 'section': 'A', 'city': 'Gurgaon', 'gender': 'M',␣
 ↪'favorite_color': 'red'},
    {'age': 22, 'section': 'B', 'city': 'Delhi',   'gender': 'F',␣
 ↪'favorite_color': 'black'},
    {'age': 13, 'section': 'C', 'city': 'Mumbai',  'gender': 'F',␣
 ↪'favorite_color': 'yellow'},
    {'age': 21, 'section': 'B', 'city': 'Delhi',   'gender': 'M',␣
 ↪'favorite_color': 'pink'},
    {'age': 12, 'section': 'B', 'city': 'Mumbai',  'gender': 'M',␣
 ↪'favorite_color': 'black'},
    {'age': 11, 'section': 'A', 'city': 'Delhi',   'gender': 'M',␣
 ↪'favorite_color': 'green'},
    {'age': 17, 'section': 'A', 'city': 'Mumbai',  'gender': 'F',␣
 ↪'favorite_color': 'red'}
]

df = pd.DataFrame(data)
display_df()

df = pd.DataFrame(data_csv[1:], columns=data_csv[0])
display_df()

df = pd.DataFrame(data_dict)
display_df()
```

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink

age	section	city	gender	favorite_color
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

3 Meta stuff

[3]:
```
data = {
    'age':             [10,22,13,21,12,11,17],
    'section':         ['A','B','C','B','B','A','A'],
    'city':           ␣
  ↪['Gurgaon','Delhi','Mumbai','Delhi','Mumbai','Delhi','Mumbai'],
    'gender':          ['M','F','F','M','M','M','F'],
    'favorite_color': ['red','black','yellow','pink','black','green','red']
}

df = pd.DataFrame(df)

print(f'df.empty:   {df.empty}')
print(f'df.shape:   {df.shape}')
print(f'df.index:   {df.index}')
print(f'df.columns: {df.columns}')

print('\ndf.describe():')
display_df(df.describe(), index=True)

print('\ndf.info():')
df.info() # This automatically prints stuff to stdout
```

```
df.empty:   False
df.shape:   (7, 5)
df.index:   RangeIndex(start=0, stop=7, step=1)
df.columns: Index(['age', 'section', 'city', 'gender', 'favorite_color'],
dtype='object')

df.describe():
```

	age
count	7
mean	15.1429
std	4.8795
min	10
25%	11.5
50%	13
75%	19
max	22

```
df.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             7 non-null      int64 
 1   section         7 non-null      object
 2   city            7 non-null      object
 3   gender          7 non-null      object
 4   favorite_color  7 non-null      object
dtypes: int64(1), object(4)
memory usage: 412.0+ bytes
```

4 .head() and .tail()

Both take a single optional argument, ie `n`, which is an integer representing the number of records to show. By default, it is 5. `.head()` shows the n top most records and `.tail()` shows the n bottom most records

[4]:

```
display_df( df.head(3) , index=True )
display_df( df.tail(3) , index=True )
```

	age	section	city	gender	favorite_color
0	10	A	Gurgaon	M	red
1	22	B	Delhi	F	black
2	13	C	Mumbai	F	yellow

	age	section	city	gender	favorite_color
4	12	B	Mumbai	M	black
5	11	A	Delhi	M	green
6	17	A	Mumbai	F	red

5 `.iloc[]`

Format is something like `DataFrame.iloc[row_indexer, column_indexer]`. Here `column_indexer` is optional. If it is not given, all columns will be printed

Remember, here rows start with index 1 rather than index 0

- `row_indexer`: This can be a slice (like `1:9:2` or `:`), or a list of the indexes, like `[1,4,5]`
- `column_indexer`: This can be a slice (like `1:9:2` or `:`), or a list of the indexes of the columns, like `[1,4,5]`

[5]:
```python
data = {
    'age':              [10,22,13,21,12,11,17],
    'section':          ['A','B','C','B','B','A','A'],
    'city':            ␣
  ↪['Gurgaon','Delhi','Mumbai','Delhi','Mumbai','Delhi','Mumbai'],
    'gender':           ['M','F','F','M','M','M','F'],
    'favorite_color': ['red','black','yellow','pink','black','green','red']
}
df = pd.DataFrame(data)

print('\nOriginal data:')
display_df()

print('\nRecords of index 1 & 3')
display_df( df.iloc[ [1,3] , : ] )
```

```
Original data:
```

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

```
Records of index 1 & 3
```

age	section	city	gender	favorite_color
22	B	Delhi	F	black
21	B	Delhi	M	pink

6 `.loc[]`

Just like `.iloc[]`, but uses names (strings) rather than indexes, and the slicing in end-inclusive (unlike the slicing we have seen till now)

[ ]:

```

```

7 `.rename()`

It by default doesn't do the renaming inplace, and instead returns a copy

(Here the `mapper`, `index`, and `columns` arguments accept a dict-like object or a function)

Arguments:

- `mapper`: Dict-like or function transformations to apply to that axis' values
- `axis: int | str = 0`: Axis to target with mapper. Can be `0`/`index` or `1`/`columns`
- `index`: Alternative to specifying axis (`mapper,axis=0` is equivalent to `index=mapper`)
- `columns`: Alternative to specifying axis (`mapper,axis=1` is equivalent to `columns=mapper`)
- `inplace: bool = False`: Whether to modify the DataFrame rather than creating a new one
- `errors: str`: Can be "raise" or "ignore". If "raise", then raise a KeyError when a dict-like mapper, index, or columns contains labels that are not present in the Index being transformed. If "ignore", existing keys will be renamed and extra keys will be ignored

[6]:

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

df.rename(columns={'A':'New A', 'B':'New B'}, inplace=True)
df.rename(index=lambda x: f'Row {x}', inplace=True)
display_df(index=True)
```

	New A	New B
Row 0	1	4
Row 1	2	5
Row 2	3	6

8 `.query()`

Can do SQL-python like queries. Returns a dataframe

The format is `DataFrame.query(expr:str, inplace:bool)`. If `inplace` is `True`, `None` is returned and the original df is replaced by the df which would had been returned if `inplace` was `False`

[7]:

```
data = {
    'age':              [10,22,13,21,12,11,17],
    'section':          ['A','B','C','B','B','A','A'],
    'city':            ␣
 ↪['Gurgaon','Delhi','Mumbai','Delhi','Mumbai','Delhi','Mumbai'],
    'gender':           ['M','F','F','M','M','M','F'],
```

```python
    'favorite_color': ['red','black','yellow','pink','black','green','red']
}
df = pd.DataFrame(data)

print('\nOriginal data:')
display_df()

print('\nRecords where age >= 15:')
display_df( df.query('age >= 15') )

print('\nRecords where age >= 12 and gender = Male:')
display_df( df.query('age >= 12 and gender == "M"') )

print('\nCity and gender of people with age >= 12:')
display_df( df.query('age >= 12')[['city','gender']] )

# Use of `@` and ```
```

```
Original data:
```

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

```
Records where age >= 15:
```

age	section	city	gender	favorite_color
22	B	Delhi	F	black
21	B	Delhi	M	pink
17	A	Mumbai	F	red

```
Records where age >= 12 and gender = Male:
```

age	section	city	gender	favorite_color
21	B	Delhi	M	pink
12	B	Mumbai	M	black

```
City and gender of people with age >= 12:
```

city	gender
Delhi	F
Mumbai	F
Delhi	M
Mumbai	M
Mumbai	F

9 `.sort_values()`

Arguments:

- `by: str | list[str]`: Name or list of names to sort by
- `axis: int | str = 0`: Axis to be sorted. Can be `0`/`index` or `1`/`columns`
- `ascending: bool | list[bool] = True`: Self explanatory. Specify list for multiple sort orders. If this is a list of bools, must match the length of the `by`
- `inplace: bool = False`: If `True`, perform operation in-place
- `na_position: str = "last"`: Puts NaNs at the beginning if `first`, and at the end if `last`
- `ignore_index: bool = False`: If `True`, the resulting axis will be labeled 0,1,…,n-1

[8]:

```
data = {
    'age':              [10,22,13,21,12,11,17],
    'section':          ['A','B','C','B','B','A','A'],
    'city':             ␣
 ↪['Gurgaon','Delhi','Mumbai','Delhi','Mumbai','Delhi','Mumbai'],
    'gender':           ['M','F','F','M','M','M','F'],
    'favorite_color': ['red','black','yellow','pink','black','green','red']
}
df = pd.DataFrame(data)

print('\nOriginal data:')
display_df()

print('\nSorted by age (descending):')
display_df( df.sort_values(by='age',ascending=False).head(3) , index=True )
```

```
Original data:
```

age	section	city	gender	favorite_color
10	A	Gurgaon	M	red
22	B	Delhi	F	black
13	C	Mumbai	F	yellow
21	B	Delhi	M	pink

age	section	city	gender	favorite_color
12	B	Mumbai	M	black
11	A	Delhi	M	green
17	A	Mumbai	F	red

```
Sorted by age (descending):
```

	age	section	city	gender	favorite_color
1	22	B	Delhi	F	black
3	21	B	Delhi	M	pink
6	17	A	Mumbai	F	red

10 `.count()`, `.sum()`, `min()`, `max()`, `.mean()`, `median()`, and `mode()`

All of these return a `pd.core.series.Series`, except `.mode()`. Mode returns a df cause there might be many values that are the mode, and different rows contains these different values

10.0.1 `.count()`

- `axis: int | str = 0`: If 0 or "index" counts are generated for each column. If 1 or "columns", counts are generated for each row
- `numeric_only: bool = False`: Include only float, int or boolean data'

10.0.2 `.sum()`

- `axis: int | str = 0`: Axis for the function to be applied on
- `numeric_only: bool = False`: Include only float, int or boolean data
- `skipna: bool = True`: Exclude NA/null values when computing the result
- `min_count: int = 0`: The required number of valid values to perform the operation. If fewer non-NA values are present, the result will be NA

10.0.3 `.min()`, `.max()`, `.mean()`, `.median()`, & `.mode()`

- `axis: int | str = 0`: Axis for the function to be applied on
- `numeric_only: bool = False`: Include only float, int or boolean data
- `skipna: bool = True`: Exclude NA/null values when computing the result

[9]:

```python
data = {
    "Person": ["John", "Myla", "Lewis", "John", "Myla"],
    "Age":    [24, np.nan, 21, 33, 26],
    "Single": [False, True, True, True, False]
}
df = pd.DataFrame(data)

print('Original df:')
display_df()
```

```
print('\n1) Count:')
display_df( df.count(), index=True)
display_df( df.count(numeric_only=True), index=True)

print('\n2) Sum:')
display_df( df.sum(), index=True)

print('\n3) Min:')
display_df( df.min(), index=True)

print('\n4) Max:')
display_df( df.max(), index=True)

print('\n5) Mean:')
display_df( df.mean(numeric_only=True), index=True)

print('\n6) Median:')
display_df( df.median(numeric_only=True), index=True)

print('\n7) Mode:')
display_df( df.mode(numeric_only=True), index=True)
```

```
Original df:
```

Person	Age	Single
John	24	False
Myla	nan	True
Lewis	21	True
John	33	True
Myla	26	False

```
1) Count:
```

	0
Person	5
Age	4
Single	5

	0
Age	4
Single	5

```
2) Sum:
```

	0
Person	JohnMylaLewisJohnMyla
Age	104.0
Single	3

3) Min:

	0
Person	John
Age	21.0
Single	False

4) Max:

	0
Person	Myla
Age	33.0
Single	True

5) Mean:

	0
Age	26
Single	0.6

6) Median:

	0
Age	25
Single	1

7) Mode:

	Age	Single
0	21	1
1	24	nan
2	26	nan

	Age	Single
3	33	nan

11 .groupby()

12 .select_dtypes()

13 .duplicated() and drop_duplicates()

14 pd.concat() and pd.append()

15 .pivot() and pivot_table()