25th July

Python- tutorial 13

- from array to dataframe:

```
from numpy.random import randn as rn   (for random data)
np.random.seed(101)   (anything ← random but fixed every time we execute)
matrix_data = rn(5,4)   ← random array 5x4
matrix_data
array([[0.2, -0.4, 0.6, 1.  ],
       [2.  , 9.  , 8.  , 4.  ],
       [11.  , 2.  , 6.  , 4.  ],
       [2.  , 9.  , 5.  , 2.  ],
       [1.  , 2.  , 9.  , 5.  ]])
pd.DataFrame(matrix_data)
```

	0	1	2	3
0	0.2	-0.4	0.6	1
1	2	9	8	4
2	11	2	6	4
3	2	9	5	2
4	1	2	9	5

(default heading & index)

if we wish to change heading, indexes:-

```
row_labels = ['A','B','C','D','E']
column_headings = ['W','X','Y','Z']
df = pd.DataFrame(matrix_data, row_labels, column_headings)
df → same above Dataframe with changed head & index
```

[iloc → default index location.
  loc → named by us indexes.
  ix → combination of loc & iloc

- extracting some section of dataframe:

① some columns:

df['x'], df[['x','z']]
↑ single column ↑ multiple columns

② some rows:

same [df.loc['C'] & type = series (single row or column)
       df.iloc[2] ← by using default index

if I change my index with integer data
ces, ['A','B','C','D','E'] → [56,23,42,8,80]

→ [df.loc[23] → 2nd row
    df.iloc[23] → error (no data found)

Note: iloc → understands only default integer location & not the integer indexes given by us.

[df.loc[8] → 4th & 5th rows
  df.iloc[3] → 4th row only.
  df.iloc[[4,1,2]] → multiple rows.

③ intersection of some rows / columns:

df.iloc[[2,3,4], [0,1]]
rows column

	W	X
42 or C	11	9
8 or D	9	2
8 or D	1	2

df.loc[[42,8], ['W','X']]
rows column

* We can drop row/column from df, ~~not a~~ can't drop data (yes we can replace)

④ using range function extracting section:
df.loc[56:42, ['w':'x']]

- Dropping some rows & columns:

① drop column:
df.drop('X')
↳ error (∵ by default axis=0)

axis=0 → x-axis (row)
axis=1 → y-axis (column)
(by-default axis=0)

df.drop('x', axis=1) (∵ doesn't drops permanently)
↳ same previous dataframe with 'X' removed

- if you want to make changes permanently (∵ inplace=True
by default, inplace=False)

df.drop('x', axis=1, inplace=True)
↳ drops 'X' column permanently.

② drop row: ← named indexes not able to understand default indexes.
df.drop(56)
↳ row-1 will be droped

df = df.drop(56)
↳ another way of permanent deletion

- filter data based on some conditions:
print(df.loc[['A','B','C']]>0)
↳

	W	X	Y	Z
A	True	False	True	True
B	True	True	True	True
C	True	True	True	True

if you want values, then put condition in df:

```
df[df.loc[['A','B','C']]>0]
```

	W	X	Y	Z
A	0.2	NaN	0.6	1
B	2	9	8	4
C	11	2	6	4
D	NaN	NaN	NaN	NaN
D	NaN	NaN	NaN	NaN

- multiple conditions ("and"):

```
df[(df['Age']>30)&(df['Height']>65
        & (df['Weight']<70)]
```

- Resetting index to default index:

no permanent changes
- `df.reset_index()` → default index + old index
- `df.reset_index(drop=True)` → only default index
- `df.rest_index(drop=True, inplace=True)` → ~~only df~~ [permanently changed to default index.

[zip() → creates list of zip based on indexes]

- "My name is MADHU".split()
  → ['My', 'name', 'is', 'MADHU'] a list

adding new column inside df:

df['xyz'] = "My name is MADHU".split()

df

→ data frame with new 'xyz' column.

use this column as index-column:

df.set_index('xyz')

- Multiple index (hierarchy)

outside = ['G1', 'G1', 'G1', 'G2', 'G2', 'G2']

inside = [1, 2, 3, 1, 2, 3]

hier_index = list(zip(outside, inside))

hier_index = pd.MultiIndex.from_tuples(hier_index)

df1 = pd.DataFrame(data = np.random.randn(6,3), index = hier_index, columns = ['A', 'B', 'C'])

print(df1)

		A	B	C
G1	1	1.0	1.0	-1.0
	2	-1.0	-0.0	2.0
	3	1.0	0.0	-1.0
G2	1	1.0	-0.0	-1.0
	2	-2.0	-2.0	1.0
	3	-1.0	2.0	-2.0

now check whether 'G1' is working as a row:

```
df1.loc['G1']
```

	A	B	C
1	1.0	1.0	-1.0
2	-1.0	-0.0	2.0
3	1.0	0.0	-1.0

```
df1.loc['G1'].loc[3,['B','C']]
```

B 0.0
C -1.0

(we may also use combination of loc & iloc)

- Drop row/column having NaN-values:

```
df = pd.DataFrame({'A':[1,2,np.nan],
    'B':[5,np.nan,np.nan],'C':[1,2,3]})
df['States'] = "MA DH UP".split()
df.set_index('States', inplace=True)
df
```

States	A	B	C
MA	1.0	5.0	1
DH	2.0	NaN	2
UP	NaN	NaN	3

```
print(df.dropna())
```

← dropped rows with NaN because by-default axis=0 (rows)

States	A	B	C
MA	1.0	5.0	1

print (df.dropna(axis=1))

States	C
MA	1
DH	2
UP	3

- With conditions dropping: (read carefully)

df.dropna(thresh=2) (with 2 or more NaN non) thersold=2

states	A	B	C
MA	1.0	5.0	1
DH	2.0	NaN	2

(axis=0 default)

Note: 'thresh' is applied on non-NaN.

Note: (for permant changes use, inplace = True)