ML Pandas basic functions, P23CS005

In [2]:
```
#importing libraries
import numpy as np
import pandas as pd
```

Basic data structures in pandas

Pandas provides two types of classes for handling data:

Series: a one-dimensional labeled array holding data of any type such as integers, strings, Python objects etc.

DataFrame: a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

In [3]:
```
arr = [1,2,3,4,5]
pd.Series(arr)
```

Out[3]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

A customized index can be assigned to the array using code below

In [4]:
```
arr = pd.Series([1,2,3,4,5], index = [1,2,3,4,5])
arr
```

Out[4]:
```
1    1
2    2
3    3
4    4
5    5
dtype: int64
```

In [5]:
```
arr = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e'])
arr
```

Out[5]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

name parameter is usedd to give specific name to array

In [6]:
```
arr = pd.Series([1,2,3,4,5], index = [1,2,3,4,5], name = 'array1')
arr
```

Out[6]:
```
1    1
2    2
3    3
4    4
5    5
Name: array1, dtype: int64
```

Series() on dictionaries

In [7]:
```
d = {'a': 1, 'b': 2, 'c': 3, 'd' : 4, 'e' : 5}
s = pd.Series(data=d)
s
```

Out[7]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Series() stores index as key value pair so unknown index will give non exist output

In [8]:
```
ser = pd.Series(data=d, index=['x', 'y', 'z'])
ser
```

Out[8]:
```
x   NaN
y   NaN
z   NaN
dtype: float64
```

In [9]:
```
s[0:3]
```

Out[9]:
```
a    1
b    2
c    3
dtype: int64
```

In [10]:
```
s[2:5]
```

Out[10]:
```
c    3
d    4
e    5
dtype: int64
```

In [11]:
```
s.shape
```

Out[11]:
```
(5,)
```

In [12]:
```
pd.Series(s).values
```

Out[12]:
```
array([1, 2, 3, 4, 5], dtype=int64)
```

In [13]:
```
pd.Series(s).astype('category').values
```

Out[13]:
```
[1, 2, 3, 4, 5]
Categories (5, int64): [1, 2, 3, 4, 5]
```

.dtype returns data type of series elements

In [14]:
```
pd.Series(s).dtype
```

Out[14]:
```
dtype('int64')
```

In [15]:
```
pd.Series(ser).dtype
```

Out[15]:
```
dtype('float64')
```

DATAFRAMES a two-dimensional data structure that holds data like a two-dimension array or a table with rows and columns.

In [3]:
```
arr = [1,2,3,4,5,6]
d = pd.DataFrame(arr)
d
```

Out[3]:

	0
0	1
1	2
2	3
3	4
4	5
5	6

In [10]:
```
arr = [[1,2,3] ,[4,5,6], [7,8,9]]
d1 = pd.DataFrame(arr)
d1
```

Out[10]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

DataFrame() always fill null values to make 2D array

In [11]:
```
arr = [[1,2,3] ,[4,5,6, 10], [7,8,9]]
d2 = pd.DataFrame(arr)
d2
```

Out[11]:

	0	1	2	3
0	1	2	3	NaN
1	4	5	6	10.0
2	7	8	9	NaN

using distionary, key will represent column

In [17]:
```
new1 = pd.DataFrame([{'a':1, 'b':2, 'c':3}, {'a':8, 'b':7, 'c':6}])
new1
```

Out[17]:

	a	b	c
0	1	2	3
1	8	7	6

creating dataframe using series

In [18]:
```
new2 = pd.DataFrame({'Rollno':pd.Series([1,2,3,4,5]),'Mark':([98,89,99,79,80])})
new2
```

Out[18]:

	Rollno	Mark
0	1	98
1	2	89
2	3	99
3	4	79
4	5	80

Creating a DataFrame by passing a NumPy array with a datetime index using date_range() and labeled columns:

In [23]:
```
dates = pd.date_range("20130101", periods=5)
dates
df = pd.DataFrame(np.random.randn(5, 4), index=dates, columns=list("ABCD"))
df
```

Out[23]:

	A	B	C	D
2013-01-01	0.571676	0.295624	0.660957	-0.433244
2013-01-02	0.624724	-0.655125	0.220652	0.958882
2013-01-03	1.965876	1.545330	0.211698	-0.221696
2013-01-04	1.781682	1.645593	0.260492	0.430243
2013-01-05	-0.828340	0.279997	0.408621	0.487101

Creating a DataFrame by passing a dictionary of objects where the keys are the column labels and the values are the column values.

In [25]:
```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
    }
```

```
)

df2
```

Out[25]:

	A	B	C	D	E
0	1.0	2013-01-02	1.0	3	test
1	1.0	2013-01-02	1.0	3	train
2	1.0	2013-01-02	1.0	3	test
3	1.0	2013-01-02	1.0	3	train

In [30]:

```
arr = np.arange(1,101)
arr = arr.reshape(10,10)
df3 = pd.DataFrame(arr)
df3
```

Out[30]:

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame

In [31]:

```
df3.head()
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50

df3.tail()

In [33]:
```
df3.head(2)
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20

In [34]:
```
df3.tail(1)
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9
9	91	92	93	94	95	96	97	98	99	100

In [35]:
```
df3.index
```

Out[35]:
```
RangeIndex(start=0, stop=10, step=1)
```

In [36]:
```
df3.columns
```

Out[36]:
```
RangeIndex(start=0, stop=10, step=1)
```

describe() shows a quick statistic summary of your numerical data:

In [38]:
```
df3.describe()
```

Out[38]:

	0	1	2	3	4	5	6	
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.
mean	46.000000	47.000000	48.000000	49.000000	50.000000	51.000000	52.000000	53.
std	30.276504	30.276504	30.276504	30.276504	30.276504	30.276504	30.276504	30.
min	1.000000	2.000000	3.000000	4.000000	5.000000	6.000000	7.000000	8.
25%	23.500000	24.500000	25.500000	26.500000	27.500000	28.500000	29.500000	30.
50%	46.000000	47.000000	48.000000	49.000000	50.000000	51.000000	52.000000	53.
75%	68.500000	69.500000	70.500000	71.500000	72.500000	73.500000	74.500000	75.
max	91.000000	92.000000	93.000000	94.000000	95.000000	96.000000	97.000000	98.

info() returns basic information about dataframe

In [39]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       10 non-null     int32
 1   1       10 non-null     int32
 2   2       10 non-null     int32
 3   3       10 non-null     int32
 4   4       10 non-null     int32
 5   5       10 non-null     int32
 6   6       10 non-null     int32
 7   7       10 non-null     int32
 8   8       10 non-null     int32
 9   9       10 non-null     int32
dtypes: int32(10)
memory usage: 532.0 bytes
```

In [41]:
```
df3.T
```

Out[41]:

	0	1	2	3	4	5	6	7	8	9
0	1	11	21	31	41	51	61	71	81	91
1	2	12	22	32	42	52	62	72	82	92
2	3	13	23	33	43	53	63	73	83	93
3	4	14	24	34	44	54	64	74	84	94
4	5	15	25	35	45	55	65	75	85	95
5	6	16	26	36	46	56	66	76	86	96
6	7	17	27	37	47	57	67	77	87	97
7	8	18	28	38	48	58	68	78	88	98
8	9	19	29	39	49	59	69	79	89	99
9	10	20	30	40	50	60	70	80	90	100

isnull() returns true if data contains null value

In [42]:
```
df3.isnull()
```

Out[42]:

	0	1	2	3	4	5	6	7	8	9
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False

In [43]:
```
df3.size
```

Out[43]: 100

In [44]:
```
df3.shape
```

Out[44]: (10, 10)

In [46]:
```
df3.replace(100,1000)
```

Out[46]:

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	1000

Calculate the mean value for each col:

In [47]:
```
df3.mean()
```

Out[47]:
```
0    46.0
1    47.0
2    48.0
3    49.0
4    50.0
5    51.0
6    52.0
7    53.0
8    54.0
9    55.0
dtype: float64
```

Calculate the mean value for each row:

In [50]:
```
df3.mean(axis=1)
```

Out[50]:
```
0     5.5
1    15.5
2    25.5
3    35.5
4    45.5
5    55.5
6    65.5
7    75.5
8    85.5
9    95.5
dtype: float64
```

In [51]:
```
s = pd.Series(np.random.randint(0, 7, size=10))
s
```

Out[51]:
```
0    1
1    6
2    2
3    6
4    4
5    4
6    2
7    0
8    3
9    0
dtype: int32
```

In [52]:
```
s.value_counts()
```

Out[52]:
```
6    2
2    2
4    2
0    2
1    1
3    1
Name: count, dtype: int64
```

In [55]:
```
s = pd.Series(["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"])

s.str.lower()
```

Out[55]:
```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

In [56]:
```
s.str.upper()
```

Out[56]:
```
0       A
1       B
2       C
3    AABA
4    BACA
5     NaN
6    CABA
7     DOG
8     CAT
dtype: object
```

In [ ]: