```
it is a tool use to analyze data and data preprocessing
2 data types - series and data frames
```

In [ ]:
```
#series data type like 1 d array
# can't grab multiple values at same time
```

In [2]:
```
import pandas as pd
```

In [59]:
```
dir(pd)
```

Out[59]:
```
['Categorical',
 'CategoricalIndex',
 'DataFrame',
 'DateOffset',
 'DatetimeIndex',
 'ExcelFile',
 'ExcelWriter',
 'Expr',
 'Float64Index',
 'Grouper',
 'HDFStore',
 'Index',
 'IndexSlice',
 'Int64Index',
 'Interval',
 'IntervalIndex',
 'MultiIndex',
 'NaT',
 'Panel',
```

In [8]:
```
marks=[80,90,100,40]
labels=['maths','science','physics','chemistry']
```

In [9]:
```
ser=pd.Series(marks,labels)
```

In [10]:
```
ser
```

Out[10]:
```
maths         80
science       90
physics      100
chemistry     40
dtype: int64
```

In [11]:
```
ser=pd.Series(data=marks,index=labels)
```

In [12]:
```
ser
```

Out[12]:
```
maths         80
science       90
physics      100
chemistry     40
dtype: int64
```

In [13]:
```
type(ser)
```

Out[13]:
```
pandas.core.series.Series
```

In [14]:
```
ser['maths']
```

Out[14]:
```
80
```

```
# data frmaes like 2 d array

pd.DataFrame(data,index in list format,col in list format)
```

In [3]:
```
import numpy as np
```

In [17]:
```
p=np.arange(16).reshape(4,4)
label=['a','b','c','d']
label2=['A','B','C','D']
```

In [18]:
```
p
```

Out[18]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [19]:
```
df=pd.DataFrame(p,index=label,columns=label2)
```

In [20]:
```
df
```

Out[20]:

	A	B	C	D
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [21]:
```
df1=pd.DataFrame([[1,2,3,4],[4,5,6,7],[8,9,10,11],[12,13,14,15]],index=" A B C D".split(),columns="a b c d".split())
```

In [22]:
```
df1
```

Out[22]:

	a	b	c	d
A	1	2	3	4
B	4	5	6	7
C	8	9	10	11
D	12	13	14	15

In [23]:
```
df1[['b','d']]
```

Out[23]:

	b	d
A	2	4
B	5	7
C	9	11
D	13	15

In [24]:
```
# to access rows with lables

df1.loc['A']
```

Out[24]:
```
a    1
b    2
c    3
d    4
Name: A, dtype: int64
```

In [25]:
```
df1.loc[['A','B']]
```

Out[25]:

	a	b	c	d
A	1	2	3	4
B	4	5	6	7

In [26]:
```
# to access row with index location

df1.iloc[0]
```

Out[26]:
```
a    1
b    2
c    3
d    4
Name: A, dtype: int64
```

In [27]:
```
# to add column

df1['new']=df1['a']+df1['b'] # adding value of a col and b col
```

In [28]:
```
df1
```

Out[28]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17
D	12	13	14	15	25

In [29]:
```
df1['new2']=[1,2,3,4]
```

In [30]:
```
df1
```

Out[30]:

	a	b	c	d	new	new2
A	1	2	3	4	3	1
B	4	5	6	7	9	2
C	8	9	10	11	17	3
D	12	13	14	15	25	4

In [31]:
```
#drop a col
# 'drop' function will try to delete row wise so put axis=1 for col delete

df1.drop('new2',axis=1) # not delete properly
```

Out[31]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17
D	12	13	14	15	25

In [32]:
```
df1
```

Out[32]:

	a	b	c	d	new	new2
A	1	2	3	4	3	1
B	4	5	6	7	9	2
C	8	9	10	11	17	3
D	12	13	14	15	25	4

In [33]:
```
# to delete col permenantly use 'inplace = True'

df1.drop('new2',axis=1,inplace = True)
```

In [34]:
```
df1
```

Out[34]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17
D	12	13	14	15	25

In [29]:
```
# delete rows

df1.drop('D')
```

Out[29]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17

In [36]:
```
df1
```

Out[36]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17

In [31]:
```
# to delete row permenantly use 'inplace = True'

df1.drop('D',inplace=True)
```

In [32]:
```
df1
```

Out[32]:

	a	b	c	d	new
A	1	2	3	4	3
B	4	5	6	7	9
C	8	9	10	11	17

In [35]:
```
#reset index

df1.reset_index()
```

Out[35]:

	index	a	b	c	d	new
0	A	1	2	3	4	3
1	B	4	5	6	7	9
2	C	8	9	10	11	17
3	D	12	13	14	15	25

In [39]:
```
newind="CA NY WY DR".split()
```

In [40]:
```
newind
```

Out[40]:
```
['CA', 'NY', 'WY', 'DR']
```

In [41]:
```
#set a new index

df1['newindex']=newind
```

In [43]:
```
df1
```

Out[43]:

	a	b	c	d	new	newindex
A	1	2	3	4	3	CA
B	4	5	6	7	9	NY
C	8	9	10	11	17	WY
D	12	13	14	15	25	DR

In [44]:
```
df1.set_index('newindex',inplace=True)
```

In [45]:
```
df1
```

Out[45]:

	a	b	c	d	new
newindex					
CA	1	2	3	4	3
NY	4	5	6	7	9
WY	8	9	10	11	17
DR	12	13	14	15	25

In [48]:
```
#will return 1st 5 rows of dataset

df1.head()
```

Out[48]:

newindex	a	b	c	d	new
CA	1	2	3	4	3
NY	4	5	6	7	9
WY	8	9	10	11	17
DR	12	13	14	15	25

In [49]:
```
#will return 1st 2 rows of dataset
#head will return 1st 5 rows
df1.head(2)
```

Out[49]:

newindex	a	b	c	d	new
CA	1	2	3	4	3
NY	4	5	6	7	9

In [50]:
```
# will show last 5 rows of dataset

df1.tail()
```

Out[50]:

newindex	a	b	c	d	new
CA	1	2	3	4	3
NY	4	5	6	7	9
WY	8	9	10	11	17
DR	12	13	14	15	25

In [51]:
```
#will return last 2 rows of dataset

df1.tail(2)
```

Out[51]:

newindex	a	b	c	d	new
WY	8	9	10	11	17
DR	12	13	14	15	25

In [55]:
```
#unique elements in a col

df1['a'].unique()
```

Out[55]:
```
array([ 1,  4,  8, 12], dtype=int64)
```

In [57]:
```
# frequency of a no in a col

df1['a'].value_counts()
```

Out[57]:
```
12    1
4     1
1     1
8     1
Name: a, dtype: int64
```

In [61]:
```
#statitical info of a data means count is total no of values in col a b c.., mean is mean of col a b..., std means[(mean-values)

df1.describe()
```

Out[61]:

	a	b	c	d	new
count	4.000000	4.000000	4.000000	4.000000	4.000000
mean	6.250000	7.250000	8.250000	9.250000	13.500000
std	4.787136	4.787136	4.787136	4.787136	9.574271
min	1.000000	2.000000	3.000000	4.000000	3.000000
25%	3.250000	4.250000	5.250000	6.250000	7.500000
50%	6.000000	7.000000	8.000000	9.000000	13.000000
75%	9.000000	10.000000	11.000000	12.000000	19.000000
max	12.000000	13.000000	14.000000	15.000000	25.000000

In [64]:
```
#info of data types in c col

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, CA to DR
Data columns (total 5 columns):
a      4 non-null int64
b      4 non-null int64
c      4 non-null int64
d      4 non-null int64
new    4 non-null int64
dtypes: int64(5)
memory usage: 192.0+ bytes
```

```
if there any missing value

1-replace missing value with mean
2-replace with mode
3-replace it with median
```

```
4-your own value
```

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

In [5]:
```
df
```

Out[5]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [6]:
```
#showing all null or not

df.isnull()
```

Out[6]:

	A	B	C
0	False	False	False
1	False	True	False
2	True	True	False

In [7]:
```
#only those col having null values

df.isnull().any()
```

Out[7]:
```
A     True
B     True
C    False
dtype: bool
```

In [71]:
```
#how many null are there in particualr col

df.isnull().sum()
```

Out[71]:
```
A    1
B    2
C    0
dtype: int64
```

In [72]:
```
#delete those rows having null

df.dropna()
```

Out[72]:

	A	B	C
0	1.0	5.0	1

In [74]:
```
#delete those col having null

df.dropna(axis=1)
```

Out[74]:

	C
0	1
1	2
2	3

In [8]:
```
#Keep only the rows with at least 2 non-NA values

df.dropna(thresh=2)
```

Out[8]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

In [76]:
```
#checking those col haing more than one null and deleting that

df.dropna(thresh=2,axis=1)
```

Out[76]:

	A	C
0	1.0	1
1	2.0	2
2	NaN	3

In [77]:
```
#replace missing values by ur own value

df.fillna(value=60)
```

Out[77]:

	A	B	C
0	1.0	5.0	1
1	2.0	60.0	2
2	60.0	60.0	3

In [101]:
```
#replacing with means

df["A"].fillna(df['A'].mean())
```

Out[101]:
```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

In [112]:
```
df["B"].fillna(df['B'].mean(),inplace=True) # inplace=True for permenant sol
```

In [113]:
```
#replacing with median

df["A"].fillna(df['A'].median(),inplace=True)
```

In [106]:
```
#showing series of values with index

p=df['A'].mode()
```

In [82]:
```
p
```

Out[82]:
```
0    1.0
1    2.0
dtype: float64
```

In [12]:
```
#need to fill wither 0 index or 1 index, optimal is 1st index of series like 0

df["A"].fillna(df['A'].mode()[0],inplace=True)
```

In [95]:
```
df1=pd.DataFrame(['Aindia',np.nan,'Aindia','Australia','Australia','us','canada'],columns=['country'])
```

In [96]:
```
df1
```

Out[96]:

	country
0	Aindia
1	NaN
2	Aindia
3	Australia
4	Australia
5	us
6	canada

In [97]:
```
#no mean and median with characters but mode is oh and it gives priority to Capital alphabatical order

q=df1['country'].mode()
```

In [98]:
```
q
```

Out[98]:
```
0       Aindia
1    Australia
dtype: object
```

In [99]:
```
q[0]
```

Out[99]:
```
'Aindia'
```

In [100]:
```
df1['country'].fillna(df1['country'].mode()[0])
```

Out[100]:
```
0       Aindia
1       Aindia
2       Aindia
3    Australia
4    Australia
5           us
6       canada
Name: country, dtype: object
```

In [114]:
```
df
```

Out[114]:

	A	B	C
0	1.0	5.0	1
1	2.0	5.0	2
2	1.5	5.0	3

```
mean-ideal condition is if col data is in a range - age(0,80)
median()-lot of variations - salaray(100 to 100000)
mode-for textual values
```

```
create a dataframe with 10 rows any sven cols two cols should be textual
in that make some missing values in four of the colums one in textual colum
apply all the functions
```

In [ ]:
```

```

```
#reading dataset, r is raw path means reading files
```

```
dataset=pd.read_csv(r"path\filename.format")
```

In [ ]: