Read data frame ¶

In [1]:
```
import pandas as pd
# read data frame
df=pd.read_csv('C:/Users/omkan/Desktop/Iris.csv')
df
```

Out[1]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

understanding_analysing the data Set

> Methods

- here v stands for variable
- v.head(), v.tail(), v.info(), v.describe(), v.corr()
- v.isnull().sum(), v.notnull().sum(), v.count()
- v['column_name'].value_counts().sort_value()

- v['column_name'].unique()
- v['column_name'].nunique()
- v.sample(no.)

> Attributes

- v.size
- v.shape
- v.columns
- v.ndim

```
# importent method to extract big amount of data sets
df.groupby(), df.query(), df.iloc[],df.loc[], extraction v[]_operators
```

In [ ]:

In [2]:
```
df.iloc[[98],:]
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
98	99	5.1	2.5	3.0	1.1	Iris-versicolor

In [3]:
```
df['Species']=df['Species'].replace({'Iris-setosa':0,'Iris-virginica':1,'Iris-versicolor':2})
```

In [4]:
```
df.tail()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	1
146	147	6.3	2.5	5.0	1.9	1
147	148	6.5	3.0	5.2	2.0	1
148	149	6.2	3.4	5.4	2.3	1
149	150	5.9	3.0	5.1	1.8	1

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    int64
dtypes: float64(4), int64(2)
memory usage: 7.2 KB
```

In [6]:
```
df.describe()
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667	1.000000
std	43.445368	0.828066	0.433594	1.764420	0.763161	0.819232
min	1.000000	4.300000	2.000000	1.000000	0.100000	0.000000
25%	38.250000	5.100000	2.800000	1.600000	0.300000	0.000000
50%	75.500000	5.800000	3.000000	4.350000	1.300000	1.000000
75%	112.750000	6.400000	3.300000	5.100000	1.800000	2.000000
max	150.000000	7.900000	4.400000	6.900000	2.500000	2.000000

In [7]:
```
df.corr()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
Id	1.000000	0.716676	-0.397729	0.882747	0.899759	0.471415
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954	0.460039
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544	-0.612165
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757	0.649101
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000	0.580749
Species	0.471415	0.460039	-0.612165	0.649101	0.580749	1.000000

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [9]:
```
df.count()
```

Out[9]:
```
Id               150
SepalLengthCm    150
SepalWidthCm     150
PetalLengthCm    150
PetalWidthCm     150
Species          150
dtype: int64
```

In [10]:
```
df['Species'].value_counts()
```

Out[10]:
```
0    50
2    50
1    50
Name: Species, dtype: int64
```

In [11]:
```
df['Species'].value_counts().sort_values()
```

Out[11]:
```
0    50
2    50
1    50
Name: Species, dtype: int64
```

In [12]:
```
df['Species'].sort_values()
```

Out[12]:
```
0     0
27    0
28    0
29    0
30    0
     ..
78    2
77    2
76    2
86    2
74    2
Name: Species, Length: 150, dtype: int64
```

In [13]:
```
df['Species'].unique()
```

Out[13]:
```
array([0, 2, 1], dtype=int64)
```

In [14]:
```
df['Species'].nunique()
```

Out[14]:
```
3
```

In [15]:
```
df.sample(7) # randomly provide values
```

Out[15]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
100	101	6.3	3.3	6.0	2.5	1
94	95	5.6	2.7	4.2	1.3	2
59	60	5.2	2.7	3.9	1.4	2
117	118	7.7	3.8	6.7	2.2	1
61	62	5.9	3.0	4.2	1.5	2
50	51	7.0	3.2	4.7	1.4	2
137	138	6.4	3.1	5.5	1.8	1

Attributes

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [17]:
```
df.dtypes
```

Out[17]:
```
Id                int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species            int64
dtype: object
```

In [18]:
```
df.shape,df.size,type(df)
```

Out[18]:
```
((150, 6), 900, pandas.core.frame.DataFrame)
```

In [ ]:
```

```

In [19]:
```
# creat series from list
import pandas as pd
l=[1,2,3,4]
s=pd.Series(l,index=('a','b','c','d'),dtype='float')
s,type(s),s.shape,s.size
```

Out[19]:
```
(a    1.0
 b    2.0
 c    3.0
 d    4.0
 dtype: float64,
 pandas.core.series.Series,
 (4,),
 4)
```

In [ ]:
```

```

In [20]:
```
# statastical operations with Series
s.mean(),s.mode(),s.median()
```

Out[20]:
```
(2.5,
 0    1.0
 1    2.0
 2    3.0
 3    4.0
 dtype: float64,
 2.5)
```

In [21]:
```
# create csv from series
s.to_csv('series_first.csv',index=False)
```

In [22]:
```
a=[1,2,3,4,5]            # create Series from list
a
import pandas as pd
a=pd.Series(a,index=('a','b','c','d','e'),dtype='str')
a,type(a)
```

Out[22]:
```
(a    1
 b    2
 c    3
 d    4
 e    5
 dtype: object,
 pandas.core.series.Series)
```

In [23]:
```
# create series form dict
s_d={'a':1,'b':2,'c':3}
s_d=pd.Series(s_d)
s_d
```

Out[23]:
```
a    1
b    2
c    3
dtype: int64
```

In [24]:
```
# create a series from tuple
s_t=(1,2,3,4,5)
pd.Series(s_t)
```

Out[24]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

DataFrame

In [25]:
```
l=[['om',100,'Indore'],        # Create from list
   ['kant',98,'Ujjain'],
   ['sharma',99,'Bhopal']]
d_l=pd.DataFrame(l,columns=('name','marks','place'),index=('r1','r2','r3'))
d_l
```

Out[25]:

	name	marks	place
r1	om	100	Indore
r2	kant	98	Ujjain
r3	sharma	99	Bhopal

In [26]:
```
import pandas as pd
df=pd.read_csv("C:\\Users\\omkan\\Desktop\\Iris.csv")
df.ndim
```

Out[26]: 2

In [27]:
```
d=pd.DataFrame({'Name':['om','kant','sharma'],
                'Marks':[100,98,99]
                })
d
```

Out[27]:

	Name	Marks
0	om	100
1	kant	98
2	sharma	99

In [28]:
```
# creating Dataframe from dictionary
d={'name':['om','kant','sharma'],'Marks':[10,20,30]}
d=pd.DataFrame(d)
d
```

Out[28]:

	name	Marks
0	om	10
1	kant	20
2	sharma	30

create csv file from dataframe

- v.to_csv('file_name.csv',index=False)

In [29]:
```
# create csv file from dataframe
d.to_csv('DataFrame_first.csv',index=False)
```

In [30]:
```
d=pd.read_csv('DataFrame_first.csv')
d
```

Out[30]:

	name	Marks
0	om	10
1	kant	20
2	sharma	30

In [31]:
```
d['name']                  # extract columns as Series
```

Out[31]:
```
0        om
1      kant
2    sharma
Name: name, dtype: object
```

In [32]:
```
d['Marks']
```

Out[32]:
```
0    10
1    20
2    30
Name: Marks, dtype: int64
```

In [33]:
```
e_a=d['name']
e_a
```

Out[33]:
```
0        om
1      kant
2    sharma
Name: name, dtype: object
```

```
# if you use duble square Bracket then it will become dic
```

In [34]:
```
#e_a[['name']]
w=[[e_a]] # it will become list
type(w)
```

Out[34]:
```
list
```

In [35]:
```
d[['name']]
```

Out[35]:

	name
0	om
1	kant
2	sharma

In [36]:
```
# extracting Series from DataFrame -
s=pd.Series(d['name'])
s1=d['name']
s,s1
```

Out[36]:
```
(0        om
 1      kant
 2    sharma
 Name: name, dtype: object,
 0        om
 1      kant
 2    sharma
 Name: name, dtype: object)
```

Casting in a list & numpy

- v.tolist() # sheries case
- v.index.tolist()
- v.values.tolist())
- v.columns.tolist()
- v.values as attribute convert array
- v.to_numpy()

In [37]:
```
# most Importent  in matplotlinb and seaborn
# value, index & columns
v_s=s.values   # casting in array
i_s=s.index
c_d=d.columns
v_s,i_s,c_d
```

Out[37]:
```
(array(['om', 'kant', 'sharma'], dtype=object),
 RangeIndex(start=0, stop=3, step=1),
 Index(['name', 'Marks'], dtype='object'))
```

In [38]:
```
pd.DataFrame(v_s)
```

Out[38]:

	0
0	om
1	kant
2	sharma

In [39]:
```
# convert value and index into list
s_v_l=s.values.tolist()  # array then list - v.values.tolist()
s_i_l=s.index.tolist()
s_v_l,s_i_l
```

Out[39]:
```
(['om', 'kant', 'sharma'], [0, 1, 2])
```

Use numpys to create DataFrame

In [40]:
```
import numpy as np
import random
arr=np.random.rand(10).reshape(2,5)
pd.DataFrame(arr)
```

Out[40]:

	0	1	2	3	4
0	0.313592	0.970164	0.117637	0.437993	0.123912
1	0.734477	0.749969	0.169344	0.270876	0.938221

In [41]:
```
# extracting elements from series  through indexing
s[2],s[0],s[1]
```

Out[41]:
```
('sharma', 'om', 'kant')
```

In [42]:
```
# extracting elements from dictionary through slicing
s[:2],s[-5:-1]
```

Out[42]:
```
(0      om
 1    kant
 Name: name, dtype: object,
 0      om
 1    kant
 Name: name, dtype: object)
```

In [43]:
```
s=pd.Series([1,2,3,4,4,6,5])
s.mean(),s.mode(),s.median(),s.count(),s.cumsum(),s.cumprod() # cumulative sum & product
```

Out[43]:
```
(3.5714285714285716,
 0    4
 dtype: int64,
 4.0,
 7,
 0     1
 1     3
 2     6
 3    10
 4    14
 5    20
 6    25
 dtype: int64,
 0       1
 1       2
 2       6
 3      24
 4      96
 5     576
 6    2880
 dtype: int64)
```

DataFrame - most importent creating dataframe from Series & add Series

In [44]:
```
# DataFrame - most imporetent creating dataframe from Series
l=[1,2,3,4,5]
l2=['a','b','c','d','e']
s=pd.Series(l)
s1=pd.Series(l2)
s,s1
```

Out[44]:
```
(0    1
 1    2
 2    3
 3    4
 4    5
 dtype: int64,
 0    a
 1    b
 2    c
 3    d
 4    e
 dtype: object)
```

In [45]:
```
d_s=pd.DataFrame({'int':s,'str':s1})
d_s
```

Out[45]:

	int	str
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

In [46]:
```
d_s.shape
```

Out[46]:
```
(5, 2)
```

In [47]:
```
# adding sering in dataframe
l='A','B','C','D','E'
s_a_d=pd.Series(l)
s_a_d,s_a_d.shape
```

Out[47]:
```
(0    A
 1    B
 2    C
 3    D
 4    E
 dtype: object,
 (5,))
```

In [48]:
```
d_s_a=d_s['U_str']=s_a_d   # v[key(given_name)]=value(series as value)
d_s_a # here single value assign to multiple variable
```

Out[48]:
```
0    A
1    B
2    C
3    D
4    E
dtype: object
```

In [49]:
```
l=[1.1,1.2,1.3,1.4,1.5]
f_s=pd.Series(l)
d_s['float']=f_s    # variable['key'(a key give name)]=value(pass Series)
```

In [50]:
```
d_s
```

Out[50]:

	int	str	U_str	float
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

set_index() - this function for set_index

- allow to assign a column name as a index of a dataframe

In [51]:
```
d_s.set_index('float')
```

Out[51]:

float	int	str	U_str
1.1	1	a	A
1.2	2	b	B
1.3	3	c	C
1.4	4	d	D
1.5	5	e	E

sort_index()

sort_values()

- list items
- list items

In [52]:
```
d_l.sort_index(ascending=False)
```

Out[52]:

	name	marks	place
r3	sharma	99	Bhopal
r2	kant	98	Ujjain
r1	om	100	Indore

In [53]:
```
d_l.sort_index(axis=1) # OR to print column names in reverse alphabetical order
# v.sort_index(axis=1,ascending=False)
```

Out[53]:

	marks	name	place
r1	100	om	Indore
r2	98	kant	Ujjain
r3	99	sharma	Bhopal

In [54]:
```
d_l.sort_index(axis=0,ascending=False) # alphabetical order
```

Out[54]:

	name	marks	place
r3	sharma	99	Bhopal
r2	kant	98	Ujjain
r1	om	100	Indore

In [55]:
```
d_l.sort_index(axis=0)
```

Out[55]:

	name	marks	place
r1	om	100	Indore
r2	kant	98	Ujjain
r3	sharma	99	Bhopal

In [56]:
```
#v.sort_value(by='col', ascending=fasle)  # colums wise asceding true or false
```

In [57]:
```
d=pd.Series([33,32,31],index=('r1','r2','r3'))
d_l['ages']=d
d_l
```

Out[57]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [58]:
```
d_l.sort_values(by='place', ascending=False)
```

Out[58]:

	name	marks	place	ages
r2	kant	98	Ujjain	32
r1	om	100	Indore	33
r3	sharma	99	Bhopal	31

In [59]:
```
d_l.sort_values(by=['ages','marks'],ascending=False)
```

Out[59]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [ ]:
```

```

In [60]:
```
# access columns values in series
# method  1 in Seies
d_s['str']
```

Out[60]:
```
0    a
1    b
2    c
3    d
4    e
Name: str, dtype: object
```

In [61]:
```
# method 2 in Series
d_s.str
```

Out[61]:
```
0    a
1    b
2    c
3    d
4    e
Name: str, dtype: object
```

Accessing one/multiple colums in DataFrame

- v['col_1'] extract as Series(single square brecket)
- v[['col_1']]extract as Data Frame (duble squre brecket)
- v[['col_1,col_2,col_n']]
- v.loc[[r_index_no1,r_index_no2],[col_names]]
- v.loc[slicing :,[col_names]]
- v.loc[[row_no./names], slicing :]
- v.loc[:,:]
- v.iloc[1:2 (sliceing),[cl_index_n1,cl_index_n2]]
- v.iloc[[1,2],[0]]
- v.iloc[[r1,r3(row_indexing)],[c0,c4(column_indexing)]]
- v.iloc[:,:]
- v.iloc[start:end:step(row),start:end:step(columns)]
- v.dorp() - drop(['row_no./col_name'],axis=0(by defalt row)/1(column))
- inplace=True - in any func means permanent change in present col.

- v.drop_duplicates()

In [62]:
```
# accessing multiple colums values in DataFrame
# method 1
d_s[['str','U_str']]      # use double squre Brecket
```

Out[62]:

	str	U_str
0	a	A
1	b	B
2	c	C
3	d	D
4	e	E

In [63]:
```
# loc function in row can indexing  and slicing but column write a column name
# method - 2 with loc function
d_s.loc[[1,2,4,1],['str','int']]  # v.loc[[1,3,4(random_indexing)],['cl_name1','cl_name2']]
```

Out[63]:

	str	int
1	b	2
2	c	3
4	e	5
1	b	2

In [64]:
```
# loc function in row can indexing  and slicing but column write a column name
# method - 3  with loc function
d_s.loc[1:2,['int']] # v.loc[ 2:30 (slicing),['cl_name1,cl_name2']]
```

Out[64]:

	int
1	2
2	3

In [65]:
```
d_s
```

Out[65]:

	int	str	U_str	float
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

In [66]:
```
d_s.loc[:,['float']]
```

Out[66]:

	float
0	1.1
1	1.2
2	1.3
3	1.4
4	1.5

In [67]:
```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
a[[1,2,0,2,1],[0,1,1,2,1]]
```

Out[67]:
```
array([4, 8, 2, 9, 5])
```

In [68]:
```
# method -1 in v.iloc[]
# v.iloc[ 1:2 (sliceing),[cl_index_n1,cl_index_n2]]
```

In [69]:
```
d_s.iloc[1:,[0,3]]      # index no. use for columns
```

Out[69]:

	int	float
1	2	1.2
2	3	1.3
3	4	1.4
4	5	1.5

In [70]:
```
d_s.iloc[[1,2],[0]]    # v.iloc[[r1,r3(row_indexing)],[c0,c4(column_indexing)]]
```

Out[70]:

	int
1	2
2	3

In [71]:
```
d_s.iloc[:,:]           # v.iloc[start:end:step(row),start:end:step(columns)]
```

Out[71]:

	int	str	U_str	float
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

In [72]:
```
df.loc[[10,130,80],['PetalLengthCm']]    # with indexing
```

Out[72]:

	PetalLengthCm
10	1.5
130	6.1
80	3.8

In [73]:
```
df.loc[:,['SepalLengthCm','PetalLengthCm']].head(2)
```

Out[73]:

	SepalLengthCm	PetalLengthCm
0	5.1	1.4
1	4.9	1.4

In [74]:
```
df.loc[(df['SepalWidthCm']>4)&(df['SepalLengthCm']>5)&(df['Species']=='Iris-setosa')]
```

Out[74]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
15	16	5.7	4.4	1.5	0.4	Iris-setosa
32	33	5.2	4.1	1.5	0.1	Iris-setosa
33	34	5.5	4.2	1.4	0.2	Iris-setosa

In [75]:
```
df.iloc[[1,125,93],[1,2,3,4,5]] # with indexing
```

Out[75]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	4.9	3.0	1.4	0.2	Iris-setosa
125	7.2	3.2	6.0	1.8	Iris-virginica
93	5.0	2.3	3.3	1.0	Iris-versicolor

In [76]:
```
df.iloc[3:10,1:]
```

Out[76]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [ ]:
```
```

Filtter DataFrame

query

- v.query('col_name < No.')
- complax queries

In [77]:
```
d_s
```

Out[77]:

	int	str	U_str	float
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

In [78]:
```
d_s.query('int > 4') # extract quanitative values
```

Out[78]:

	int	str	U_str	float
4	5	e	E	1.5

In [79]:
```
d_s[d_s['int']>4]
```

Out[79]:

	int	str	U_str	float
4	5	e	E	1.5

In [80]:
```
d_s[['int','str','float']].query('int<6 & float>1.1')
```

Out[80]:

	int	str	float
1	2	b	1.2
2	3	c	1.3
3	4	d	1.4
4	5	e	1.5

In [81]:
```
d_s[['int','str','float']].query('int<6 and float>1.1')
```

Out[81]:

	int	str	float
1	2	b	1.2
2	3	c	1.3
3	4	d	1.4
4	5	e	1.5

In [82]:
```
d_s[['int','str','float']][(d_s['int']<6) & (d_s['float']>1.1)]
```

Out[82]:

	int	str	float
1	2	b	1.2
2	3	c	1.3
3	4	d	1.4
4	5	e	1.5

In [83]:
```
#d_s[['int','str','float']][(d_s['int']<6) and (d_s['float']>1.1)]
```

In [84]:
```
d_s.query('str=="c"')
```

Out[84]:

	int	str	U_str	float
2	3	c	C	1.3

In [85]:
```
d_s.query('int==3')
```

Out[85]:

	int	str	U_str	float
2	3	c	C	1.3

In [86]:
```
d_s
```

Out[86]:

	int	str	U_str	float
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

In [87]:
```
d_s[['int','str']][d_s['float']<1.9]
```

Out[87]:

	int	str
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

In [ ]:
```

```

In [88]:
```
d_l[['name','marks']][d_l['marks']>98]
```

Out[88]:

	name	marks
r1	om	100
r3	sharma	99

In [89]:
```
d_l[['name','marks']].query('marks>98')
```

Out[89]:

	name	marks
r1	om	100
r3	sharma	99

In [90]:
```
d_l[['name','marks']][d_l['marks']>97].sum()
```

Out[90]:
```
name     omkantsharma
marks             297
dtype: object
```

In [91]:
```
d_l[['name','marks']].query('marks>90').sum()
```

Out[91]:
```
name     omkantsharma
marks             297
dtype: object
```

In [92]:
```
d_l[['marks','name']][d_l['marks']==100]
```

Out[92]:

	marks	name
r1	100	om

In [93]:
```
d_l[['marks','name']].query('marks==100')
```

Out[93]:

	marks	name
r1	100	om

In [94]:
```
l=[33,32,31]
l_l=pd.Series(l,index=('r1','r2','r3'))
d_l['ages']=l_l
d_l
```

Out[94]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [95]:
```
d_l.query('marks>=100 and ages<=33')
```

Out[95]:

	name	marks	place	ages
r1	om	100	Indore	33

In [96]:
```
d_l.query('marks>=100 or ages<=33')
```

Out[96]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [97]:
```
d_l.query('(marks>=100 and ages>=33) and name=="om"')
```

Out[97]:

	name	marks	place	ages
r1	om	100	Indore	33

In [98]:
```
d_l.query('marks>=100 or ages<=33 or name=="om"')
```

Out[98]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [99]:
```
d_l[(d_l['marks']>=100) | (d_l['ages']<=33) | (d_l['name']=='om')]
```

Out[99]:

	name	marks	place	ages
r1	om	100	Indore	33
r2	kant	98	Ujjain	32
r3	sharma	99	Bhopal	31

In [100]:
```
df.query('PetalLengthCm>1 and PetalWidthCm>0.3 and Species=="Iris-setosa"')
```

Out[100]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5	6	5.4	3.9	1.7	0.4	Iris-setosa
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
21	22	5.1	3.7	1.5	0.4	Iris-setosa
23	24	5.1	3.3	1.7	0.5	Iris-setosa
26	27	5.0	3.4	1.6	0.4	Iris-setosa
31	32	5.4	3.4	1.5	0.4	Iris-setosa
43	44	5.0	3.5	1.6	0.6	Iris-setosa
44	45	5.1	3.8	1.9	0.4	Iris-setosa

In [101]:
```
df[(df['PetalLengthCm']>1) & (df['PetalWidthCm']>0.3) & (df['Species']=='Iris-setosa')]
```

Out[101]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5	6	5.4	3.9	1.7	0.4	Iris-setosa
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
21	22	5.1	3.7	1.5	0.4	Iris-setosa
23	24	5.1	3.3	1.7	0.5	Iris-setosa
26	27	5.0	3.4	1.6	0.4	Iris-setosa
31	32	5.4	3.4	1.5	0.4	Iris-setosa
43	44	5.0	3.5	1.6	0.6	Iris-setosa
44	45	5.1	3.8	1.9	0.4	Iris-setosa

In [102]:
```
print(df['Species'].value_counts())
df_cq=df[df['Species']!='Iris-setosa']
print(df_cq['Species'].value_counts())
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

v.groupy()

In [103]:
```
df.columns
```

Out[103]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [104]:
```
df
```

Out[104]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
df4=df.drop(['e',20,'201'],axis=1)
df4.head(2)
```

In [105]:
```
df4=df
```

In [106]:
```
df4.columns=['i','sl','sw','pl','pw','s']
```

In [107]:
```
df4.groupby(['s'])['sl'].sum()
```

Out[107]:
```
s
Iris-setosa        250.3
Iris-versicolor    296.8
Iris-virginica     329.4
Name: sl, dtype: float64
```

In [108]:
```
df4['sl'].groupby(df4['s']).sum()
```

Out[108]:
```
s
Iris-setosa        250.3
Iris-versicolor    296.8
Iris-virginica     329.4
Name: sl, dtype: float64
```

In [109]:
```
se=df4['sl'].groupby(df4['s']).sum()
```

In [110]:
```
print(se.index.tolist())
print(se.values.tolist())
```

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
[250.3, 296.8, 329.4]
```

In [ ]:
```

```

Data Cleaning - Rename, Drop Columns, dropna, Fillna.ipynb

- df.drop('R_no./name') by defalt is axis 0
- df.drop('R_no./name',axis=0) - drop rows
- df.drop('Col_no./name'axis=1) - drop column copy
- df.drop(['col_name'],axis=1,inplace=True)
- df=df.drop(['B',"C"],axis=1)
- df.columns=['one','Two','Three','four','five','six','seven','eight']
- df.index=['a','b','c','d','e']

In [111]:
```
d_s.drop([1,2,3]) # drop row by indexing
```

Out[111]:

	int	str	U_str	float
0	1	a	A	1.1
4	5	e	E	1.5

In [112]:
```
d_s.drop(['str','float','int'],axis=1)
```

Out[112]:

	U_str
0	A
1	B
2	C
3	D
4	E

In [113]:
```
df.head(2)
```

Out[113]:

	i	sl	sw	pl	pw	s
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

In [114]:
```
df=pd.read_csv('C:/Users/omkan/Desktop/Iris.csv')
```

In [115]:
```
df.drop(['SepalLengthCm','PetalLengthCm','PetalWidthCm'],axis=1).head()
```

Out[115]:

	Id	SepalWidthCm	Species
0	1	3.5	Iris-setosa
1	2	3.0	Iris-setosa
2	3	3.2	Iris-setosa
3	4	3.1	Iris-setosa
4	5	3.6	Iris-setosa

In [116]:
```
df.drop([1,3,5]).head(5)
```

Out[116]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa

In [117]:
```
# rename columns with list
d_s.columns=list('isuf')
d_s
```

Out[117]:

	i	s	u	f
0	1	a	A	1.1
1	2	b	B	1.2
2	3	c	C	1.3
3	4	d	D	1.4
4	5	e	E	1.5

In [118]:
```
d_s['f']=333  # add item in dataset as whole column
d_s.loc[0,'i']=123  # v.loc[row,'col']=value
d_s.loc[0,'s']=311
d_s
df['e']=100
df[20]=20
df.head()
df.loc[0,'201']=33
df.head()
```

Out[118]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	e	20	201
0	1	5.1	3.5	1.4	0.2	Iris-setosa	100	20	33.0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	100	20	NaN
2	3	4.7	3.2	1.3	0.2	Iris-setosa	100	20	NaN
3	4	4.6	3.1	1.5	0.2	Iris-setosa	100	20	NaN
4	5	5.0	3.6	1.4	0.2	Iris-setosa	100	20	NaN

In [119]:
```
df.head(2)
```

Out[119]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	e	20	201
0	1	5.1	3.5	1.4	0.2	Iris-setosa	100	20	33.0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	100	20	NaN

In [ ]:
```

```

In [120]:
```
df.columns
```

Out[120]:
```
Index([           'Id', 'SepalLengthCm',  'SepalWidthCm', 'PetalLengthCm',
       'PetalWidthCm',       'Species',              'e',              20,
                  '201'],
      dtype='object')
```

In [121]:
```
df1=df.drop(['e',20,'201'],axis=1)
df1.head(2)
```

Out[121]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

Rename columns

In [122]:
```
# rename all columns
df1.columns
```

Out[122]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [123]:
```
# rename columns
df1.columns = ['id','sepallength','sepalwidth','petallength','petalwidth','species']
df1.head(1)
```

Out[123]:

	id	sepallength	sepalwidth	petallength	petalwidth	species
0	1	5.1	3.5	1.4	0.2	Iris-setosa

In [124]:
```
# rename specific columns
df1.rename(columns={'sepallength':'sl','sepalwidth':'sw','petallength':'pl','petalwidth':'pw'},inplace=True)
df1.head(2)
```

Out[124]:

	id	sl	sw	pl	pw	species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

In [125]:
```
# rearrange columns
# method = 1
col=['sw','pl','pw','id','sl','species']
df1=pd.DataFrame(df1,columns=col)
df1.head(1)
```

Out[125]:

	sw	pl	pw	id	sl	species
0	3.5	1.4	0.2	1	5.1	Iris-setosa

In [126]:
```
# method -2
df1=df1[['id','sw','sl','pl','pw','species']]
df1.head(1)
```

Out[126]:

	id	sw	sl	pl	pw	species
0	1	3.5	5.1	1.4	0.2	Iris-setosa

drop the duplicate - df= drop_duplicate row

In [127]:
```
d_s[10]=20
d_s[10]=22
print(d_s.head(2))
```

```
     i    s  u    f  10
0  123  311  A  333  22
1    2    b  B  333  22
```

In [128]:
```
d_d=d_s.drop(['i','s','u','f'],axis=1)
d_d.head(2)
```

Out[128]:

	10
0	22
1	22

In [129]:
```
d_d=d_d.drop_duplicates() # row wise
d_d
```

Out[129]:

	10
0	22

In [130]:
```
d_s.drop_duplicates([10])  # drop value column_name wise
```

Out[130]:

	i	s	u	f	10
0	123	311	A	333	22

In [131]:
```
d_s.columns
```

Out[131]:
```
Index(['i', 's', 'u', 'f', 10], dtype='object')
```

In [132]:
```
d_s[5,10]=np.nan
print(d_s)
d_s.drop_duplicates([(5,10),10])
```

```
     i    s  u    f  10  (5, 10)
0  123  311  A  333  22       NaN
1    2    b  B  333  22       NaN
2    3    c  C  333  22       NaN
3    4    d  D  333  22       NaN
4    5    e  E  333  22       NaN
```

Out[132]:

	i	s	u	f	10	(5, 10)
0	123	311	A	333	22	NaN

In [133]:
```
# delete -
import sys
del d_s[(5,10)]
```

In [134]:
```
d_s.head(2)
```

Out[134]:

	i	s	u	f	10
0	123	311	A	333	22
1	2	b	B	333	22

v.dropna()

In [135]:
```
d_s[100]=np.nan
d_s.iloc[1,1]=np.nan
d_s.loc[3,'f']=np.nan
d_s[3,'u']=np.nan
d_s
```

Out[135]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN
1	2	NaN	B	333.0	22	NaN	NaN
2	3	c	C	333.0	22	NaN	NaN
3	4	d	D	NaN	22	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN

In [136]:
```
d_s.dropna()
```

Out[136]:

	i	s	u	f	10	100	(3, u)

In [137]:
```
d_s.dropna(axis=0, how='any')  # row wise
```

Out[137]:

	i	s	u	f	10	100	(3, u)

In [138]:
```
d_1=d_s.dropna(axis=1,how='all')  # drop column with all null
d_1
```

Out[138]:

	i	s	u	f	10
0	123	311	A	333.0	22
1	2	NaN	B	333.0	22
2	3	c	C	333.0	22
3	4	d	D	NaN	22
4	5	e	E	333.0	22

In [139]:
```
d_1.dropna(axis=0,how='any')       # drop row with any null
```

Out[139]:

	i	s	u	f	10
0	123	311	A	333.0	22
2	3	c	C	333.0	22
4	5	e	E	333.0	22

v.fillna()

In [140]:
```
d_s.head()
```

Out[140]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN
1	2	NaN	B	333.0	22	NaN	NaN
2	3	c	C	333.0	22	NaN	NaN
3	4	d	D	NaN	22	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN

In [141]:
```
d_s.columns
```

Out[141]:
```
Index(['i', 's', 'u', 'f', 10, 100, (3, 'u')], dtype='object')
```

In [142]:
```
d_s.rename( columns={(6,10):'e',(2,'u'):'g'},inplace=True)
d_s.head(1)
```

Out[142]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN

In [143]:
```
#d_s.columns=list('123456789')
#d_s
```

In [144]:
```
d_s.fillna(0).head(2)
```

Out[144]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	0.0	0.0
1	2	0	B	333.0	22	0.0	0.0

In [145]:
```
d_s.fillna(method='ffill') # forward fill
```

Out[145]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN
1	2	311	B	333.0	22	NaN	NaN
2	3	c	C	333.0	22	NaN	NaN
3	4	d	D	333.0	22	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN

In [146]:
```
d_s.fillna(method='bfill') # backword fill
```

Out[146]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN
1	2	c	B	333.0	22	NaN	NaN
2	3	c	C	333.0	22	NaN	NaN
3	4	d	D	333.0	22	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN

In [147]:
```
d_s
```

Out[147]:

	i	s	u	f	10	100	(3, u)
0	123	311	A	333.0	22	NaN	NaN
1	2	NaN	B	333.0	22	NaN	NaN
2	3	c	C	333.0	22	NaN	NaN
3	4	d	D	NaN	22	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN

fill mean very importent

In [162]:
```
d_s['int']=[11,12,13,np.nan,15]
d_s
```

Out[162]:

	i	s	u	f	10	100	(3, u)	int
0	123	311	A	333.0	22	NaN	NaN	11.0
1	2	NaN	B	333.0	22	NaN	NaN	12.0
2	3	c	C	333.0	22	NaN	NaN	13.0
3	4	d	D	NaN	22	NaN	NaN	NaN
4	5	e	E	333.0	22	NaN	NaN	15.0

In [163]:
```
m_1=d_s['int'].mean()
m_1
```

Out[163]: 12.75

In [164]:
```
#fill_values = {'A': df['A'].mean(), 'B': 34, 'C': 2, 'D': 3}
fill={'int':m_1,100:2,'f':3,(3,'u'):2}
d_f=d_s.fillna(value=fill)
d_f
```

Out[164]:

	i	s	u	f	10	100	(3, u)	int
0	123	311	A	333.0	22	2.0	2.0	11.00
1	2	NaN	B	333.0	22	2.0	2.0	12.00
2	3	c	C	333.0	22	2.0	2.0	13.00
3	4	d	D	3.0	22	2.0	2.0	12.75
4	5	e	E	333.0	22	2.0	2.0	15.00

In [165]:
```
d_f.columns
```

Out[165]: Index(['i', 's', 'u', 'f', 10, 100, (3, 'u'), 'int'], dtype='object')

In [ ]:

```
```

In [166]:

```
# v.apply(np.sum,axis=0)
# v.apply(np.sum,axis=1)
# v.apply(np.mean,axis=1)
```

Merge, Append, Concatinate

In [167]:

```
d1=pd.DataFrame({'a':[1,2,4,6],
                 'B':[12,13,14,15],
                 'C':[111,222,333,444]},index=('R1','R2','R3','R4'))
d2=pd.DataFrame({'a':[1,4,6,7],
                 'b':[1.2,1.3,1.4,1.5],
                 'c':[100,200,300,400]},index=('r1','r2','r3','r4'))
```

In [168]:

```
d1
```

Out[168]:

	a	B	C
R1	1	12	111
R2	2	13	222
R3	4	14	333
R4	6	15	444

In [169]:
```
d2
```

Out[169]:

	a	b	c
r1	1	1.2	100
r2	4	1.3	200
r3	6	1.4	300
r4	7	1.5	400

Merge

- inner : Return only the rows in which the left table has matching keys in the right table
- v=pd.merge(v1,v2,on='Based col_name', how=inner)

In [170]:
```
d=pd.merge(d1,d2,on='a',how='inner')
d
```

Out[170]:

	a	B	C	b	c
0	1	12	111	1.2	100
1	4	14	333	1.3	200
2	6	15	444	1.4	300

In [171]:
```
d=pd.merge(d1,d2,on='a',how='outer')
d
```

Out[171]:

	a	B	C	b	c
0	1	12.0	111.0	1.2	100.0
1	2	13.0	222.0	NaN	NaN
2	4	14.0	333.0	1.3	200.0
3	6	15.0	444.0	1.4	300.0
4	7	NaN	NaN	1.5	400.0

In [172]:
```
d=pd.merge(d1,d2,on='a',how='left')
d
```

Out[172]:

	a	B	C	b	c
0	1	12	111	1.2	100.0
1	2	13	222	NaN	NaN
2	4	14	333	1.3	200.0
3	6	15	444	1.4	300.0

In [173]:
```
d=pd.merge(d1,d2,on='a',how='right')
d
```

Out[173]:

	a	B	C	b	c
0	1	12.0	111.0	1.2	100
1	4	14.0	333.0	1.3	200
2	6	15.0	444.0	1.4	300
3	7	NaN	NaN	1.5	400

append

In [174]:
```
d=d1.append(d2)
d
```

Out[174]:

	a	B	C	b	c
R1	1	12.0	111.0	NaN	NaN
R2	2	13.0	222.0	NaN	NaN
R3	4	14.0	333.0	NaN	NaN
R4	6	15.0	444.0	NaN	NaN
r1	1	NaN	NaN	1.2	100.0
r2	4	NaN	NaN	1.3	200.0
r3	6	NaN	NaN	1.4	300.0
r4	7	NaN	NaN	1.5	400.0

In [175]:
```
d=d2.append(d1)
d
```

Out[175]:

	a	b	c	B	C
r1	1	1.2	100.0	NaN	NaN
r2	4	1.3	200.0	NaN	NaN
r3	6	1.4	300.0	NaN	NaN
r4	7	1.5	400.0	NaN	NaN
R1	1	NaN	NaN	12.0	111.0
R2	2	NaN	NaN	13.0	222.0
R3	4	NaN	NaN	14.0	333.0
R4	6	NaN	NaN	15.0	444.0

In [176]:
```
d.columns
d.index=list(range(0,8))
d
```

Out[176]:

	a	b	c	B	C
0	1	1.2	100.0	NaN	NaN
1	4	1.3	200.0	NaN	NaN
2	6	1.4	300.0	NaN	NaN
3	7	1.5	400.0	NaN	NaN
4	1	NaN	NaN	12.0	111.0
5	2	NaN	NaN	13.0	222.0
6	4	NaN	NaN	14.0	333.0
7	6	NaN	NaN	15.0	444.0

In [177]:
```
# concat
pd.concat([d1,d2])
```

Out[177]:

	a	B	C	b	c
R1	1	12.0	111.0	NaN	NaN
R2	2	13.0	222.0	NaN	NaN
R3	4	14.0	333.0	NaN	NaN
R4	6	15.0	444.0	NaN	NaN
r1	1	NaN	NaN	1.2	100.0
r2	4	NaN	NaN	1.3	200.0
r3	6	NaN	NaN	1.4	300.0
r4	7	NaN	NaN	1.5	400.0

In [178]:
```
pd.concat([d2,d1],axis=1)
```

Out[178]:

	a	b	c	a	B	C
r1	1.0	1.2	100.0	NaN	NaN	NaN
r2	4.0	1.3	200.0	NaN	NaN	NaN
r3	6.0	1.4	300.0	NaN	NaN	NaN
r4	7.0	1.5	400.0	NaN	NaN	NaN
R1	NaN	NaN	NaN	1.0	12.0	111.0
R2	NaN	NaN	NaN	2.0	13.0	222.0
R3	NaN	NaN	NaN	4.0	14.0	333.0
R4	NaN	NaN	NaN	6.0	15.0	444.0

Transpose

In [179]:
```
df.T
```

Out[179]:

	0	1	2	3	4	5	6	7	8	9	...	140	141	142	143	
Id	1	2	3	4	5	6	7	8	9	10	...	141	142	143	144	
SepalLengthCm	5.1	4.9	4.7	4.6	5.0	5.4	4.6	5.0	4.4	4.9	...	6.7	6.9	5.8	6.8	
SepalWidthCm	3.5	3.0	3.2	3.1	3.6	3.9	3.4	3.4	2.9	3.1	...	3.1	3.1	2.7	3.2	
PetalLengthCm	1.4	1.4	1.3	1.5	1.4	1.7	1.4	1.5	1.4	1.5	...	5.6	5.1	5.1	5.9	
PetalWidthCm	0.2	0.2	0.2	0.2	0.2	0.4	0.3	0.2	0.2	0.1	...	2.4	2.3	1.9	2.3	
Species	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa	...	Iris-virginica	Iris-virginica	Iris-virginica	Iris-virginica	v
e	100	100	100	100	100	100	100	100	100	100	...	100	100	100	100	
20	20	20	20	20	20	20	20	20	20	20	...	20	20	20	20	
201	33.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	

9 rows × 150 columns

In [ ]:
```

```

In [180]:
```
#rediction
df
d_l=df
d_l
d_l.mean(),d_l.median(),d_l.std(),d_l.var() # ,d_l.mode()
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_5216/1306969315.py:5: FutureWarning: Dropping of nuisance colum
ns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  d_l.mean(),d_l.median(),d_l.std(),d_l.var() # ,d_l.mode()
```

```
Out[180]: (Id                75.500000
           SepalLengthCm      5.843333
           SepalWidthCm       3.054000
           PetalLengthCm      3.758667
           PetalWidthCm       1.198667
           e                100.000000
           20                20.000000
           201               33.000000
           dtype: float64,
           Id                75.50
           SepalLengthCm      5.80
           SepalWidthCm       3.00
           PetalLengthCm      4.35
           PetalWidthCm       1.30
           e                100.00
           20                20.00
           201               33.00
           dtype: float64,
           Id                43.445368
           SepalLengthCm      0.828066
           SepalWidthCm       0.433594
           PetalLengthCm      1.764420
           PetalWidthCm       0.763161
           e                  0.000000
           20                 0.000000
           201                     NaN
           dtype: float64,
           Id                1887.500000
           SepalLengthCm        0.685694
           SepalWidthCm         0.188004
           PetalLengthCm        3.113179
           PetalWidthCm         0.582414
           e                    0.000000
           20                   0.000000
           201                       NaN
           dtype: float64)
```

DataFrame statistical functions -

- mean(),median(),mode(),std(),var()

In [183]:
```
df.mean()
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_5216/3698961737.py:1: FutureWarning: Dropping of nuisance colum
ns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.mean()
```

Out[183]:
```
Id               75.500000
SepalLengthCm     5.843333
SepalWidthCm      3.054000
PetalLengthCm     3.758667
PetalWidthCm      1.198667
e               100.000000
20               20.000000
201              33.000000
dtype: float64
```

In [184]:
```
df.median()
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_5216/530051474.py:1: FutureWarning: Dropping of nuisance column
s in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Typ
eError.  Select only valid columns before calling the reduction.
  df.median()
```

Out[184]:
```
Id               75.50
SepalLengthCm     5.80
SepalWidthCm      3.00
PetalLengthCm     4.35
PetalWidthCm      1.30
e               100.00
20               20.00
201              33.00
dtype: float64
```

In [186]:
```
#df.mode()
df.std()
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_5216/300518187.py:2: FutureWarning: Dropping of nuisance column
s in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Typ
eError.  Select only valid columns before calling the reduction.
  df.std()
```

Out[186]:
```
Id               43.445368
SepalLengthCm     0.828066
SepalWidthCm      0.433594
PetalLengthCm     1.764420
PetalWidthCm      0.763161
e                 0.000000
20                0.000000
201                    NaN
dtype: float64
```

In [188]:
```
df.var()  #var() function that calculates the variance along all columns
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_5216/2027085253.py:1: FutureWarning: Dropping of nuisance colum
ns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.var()  #var() function that calculates the variance along all columns
```

Out[188]:
```
Id               1887.500000
SepalLengthCm       0.685694
SepalWidthCm        0.188004
PetalLengthCm       3.113179
PetalWidthCm        0.582414
e                   0.000000
20                  0.000000
201                      NaN
dtype: float64
```

In [ ]:
```

```