```
import pandas as pd
import numpy as np

d = np.random.randint(10,50,(10,4))
c = ['col1','col2','col3','col4']
i = ['A','B','C','D','E','F','G','H','I','J']

df = pd.DataFrame(data = d, columns = c, index = i)
```

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
# Selection of rows and columns
ser1 = df['col1']
print(ser1)
```

```
A    18
B    43
C    44
D    49
E    21
F    31
G    34
H    49
I    35
J    10
Name: col1, dtype: int64
```

```
type(df)
```

pandas.core.frame.DataFrame
```
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype |
None=None, copy: bool | None=None) -> None
```

```
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.
```

```
type(ser1)
```

pandas.core.series.Series
```
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None,
fastpath: bool | lib.NoDefault=lib.no_default) -> None
```

```
One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
methods from ndarray have been overridden to automatically exclude
```

```
df[['col3','col2']]
```

	col3	col2
A	27	44
B	21	17
C	45	44
D	47	44
E	28	18
F	43	39
G	29	48
H	18	46
I	25	43
J	25	27

```
df.shape
```

```
(10, 4)
```

```
list(df.columns)
```

```
['col1', 'col2', 'col3', 'col4']
```

```
df.index
```

```
Index(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'], dtype='object')
```

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
df.iloc[6]  #with index; extract values of 5th row(0-4)
```

	G
col1	34
col2	48
col3	29
col4	47

dtype: int64

```
df.loc['J']
```

	J
col1	16
col2	27
col3	25
col4	29

dtype: int64

```
np.array([10,3.14,'ABC'])
```

```
array(['10', '3.14', 'ABC'], dtype='<U32')
```

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
df.loc[['E','H','I']]
```

	col1	col2	col3	col4
E	21	18	28	10
H	49	46	18	27
I	35	43	25	29

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
df.loc[['C','D','E','F']][['col1','col2']]
```

	col1	col2
C	44	44
D	49	44
E	21	18
F	31	39

```
df.loc[['C','D','E','F'],['col1','col2']]
```

	col1	col2
C	44	44
D	49	44
E	21	18
F	31	39

```
df[['col1','col2']].loc[['C','D','E','F']]
```

	col1	col2
C	44	44
D	49	44
E	21	18
F	31	39

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	16	27	25	29

```
df.iloc[::3] #extract rows starting from 1st and then skip 1
#and so on...
```

	col1	col2	col3	col4
A	18	44	27	47
D	49	44	47	32
G	34	48	29	47
J	10	20	30	40

```
# Last 5 rows
df.iloc[-5:]
```

	col1	col2	col3	col4
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
# except last 5 rows
df.iloc[:-5]
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	16	27	25	29

```
df.iloc[4:7,0:3] #cut rows from 4 to 6 and extarct colms from 1,23
```

	col1	col2	col3
E	42	25	35
F	49	28	28
G	43	38	17

```
df.iloc[::3,::2]
```

	col1	col3
A	18	27
D	49	47
G	34	29
J	16	25

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

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31

```
df
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40

```
# Create new cols/rows , deletion
df['total'] = range(10)
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	0
B	43	17	21	37	1
C	44	44	45	31	2
D	49	44	47	32	3
E	21	18	28	10	4
F	31	39	43	48	5
G	34	48	29	47	6
H	49	46	18	27	7
I	35	43	25	29	8
J	16	27	25	29	9

```
df['total'] = df['col1'] + df['col2']
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30

```
df.loc['J'] = [10,20,30,40,50]
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	0
B	43	17	21	37	1
C	44	44	45	31	2
D	49	44	47	32	3
E	21	18	28	10	4
F	31	39	43	48	5
G	34	48	29	47	6
H	49	46	18	27	7
I	35	43	25	29	8
J	10	20	30	40	50

```
df.loc['K'] = [100,200,300,400,500]
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30
K	100	200	300	400	500

```
df.iloc[10]
```

	K
col1	100
col2	200
col3	300
col4	400
total	500

dtype: int64

```
df.iloc[11] = [4,3,2,1,7] #will gv error as no recorsd
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-76-fbbe2ff514a3> in <cell line: 1>()
----> 1 df.iloc[11] = [4,3,2,1,7] #will gv error as no recorsd

                              1 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in _has_valid_setitem_indexer(self, indexer)
   1644             elif is_integer(i):
   1645                 if i >= len(ax):
-> 1646                     raise IndexError("iloc cannot enlarge its target object")
   1647             elif isinstance(i, dict):
   1648                 raise IndexError("iloc cannot enlarge its target object")

IndexError: iloc cannot enlarge its target object
```

```
df.sum()
```

	0
col1	434
col2	563
col3	613
col4	748
total	1217

dtype: int64

```
df.loc['TOTAL'] = df.sum()
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30
K	100	200	300	400	500
TOTAL	434	563	613	748	1197

```
# Deletion
df.drop('total', axis=1)
```

	col1	col2	col3	col4
A	18	44	27	47
B	43	17	21	37
C	44	44	45	31
D	49	44	47	32
E	21	18	28	10
F	31	39	43	48
G	34	48	29	47
H	49	46	18	27
I	35	43	25	29
J	10	20	30	40
K	100	200	300	400
TOTAL	434	563	613	748

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	50
K	100	200	300	400	500
TOTAL	434	563	613	748	1217

```
df.drop('total', axis=1, inplace=True) # permanent deletion
```

```
df
```

	col1	col2	col3	col4
A	10	46	39	30
B	11	37	48	19
C	14	48	42	37
D	41	22	19	29
E	23	21	10	29
F	16	11	48	41
G	36	11	39	45
H	38	20	49	41
I	33	44	18	10
J	10	20	30	40
K	100	200	300	400
TOTAL	664	960	1284	1442

```
df.drop('TOTAL', axis=0)
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30
K	100	200	300	400	500

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30
K	100	200	300	400	500
TOTAL	434	563	613	748	1197

```
df.drop('TOTAL',axis=0,inplace=True)
```

```
df
```

	col1	col2	col3	col4	total
A	18	44	27	47	62
B	43	17	21	37	60
C	44	44	45	31	88
D	49	44	47	32	93
E	21	18	28	10	39
F	31	39	43	48	70
G	34	48	29	47	82
H	49	46	18	27	95
I	35	43	25	29	78
J	10	20	30	40	30
K	100	200	300	400	500

```
# Opertions
```

```
df
```

	col1	col2	col3	col4
A	19	45	14	18
B	45	20	30	46
C	16	38	36	10
D	37	16	19	33
E	35	33	33	22
F	47	16	10	28
G	37	34	14	42
H	10	28	19	42
I	43	29	20	42
J	10	20	30	40
K	100	200	300	400

```
df['col1'] + df['col2'] - df['col3'] # df.col1
```

	0
A	17
B	0
C	20
D	44
E	34
F	-21
G	8
H	9
I	59
J	0
K	0

dtype: int64

```
df['col1'] < 30
```

	col1
A	True
B	True
C	True
D	False
E	True
F	True
G	False
H	False
I	False
J	True
K	False

dtype: bool

```
df['col1'] < df['col2']
```

	0
A	True
B	True
C	True
D	False
E	False
F	False
G	False
H	False
I	True
J	True
K	True

dtype: bool

```
b1 = df['col1'] < 60
b2 = df['col1'] > 30
b1 & b2
```

```
A    False
B     True
C    False
```

```
D     True
E     True
F     True
G     True
H    False
I     True
J    False
K    False
Name: col1, dtype: bool
```

```
b3 = (df['col1'] < 60) & (df['col1'] > 30)
```

```
b3
```

	col1
A	False
B	False
C	False
D	True
E	False
F	False
G	True
H	True
I	True
J	False
K	False

dtype: bool

```
df[b3]
```

	col1	col2	col3	col4
B	45	20	30	46
D	37	16	19	33
E	35	33	33	22
F	47	16	10	28
G	37	34	14	42
I	43	29	20	42

```
# Select rows from df where col1 value between 30 and 60
df[(df['col1'] < 60) & (df['col1'] > 30)]
```

	col1	col2	col3	col4
D	41	22	19	29
G	36	11	39	45
H	38	20	49	41
I	33	44	18	10

```
# Select only those rows where col1 and col2 values are even
df[(df['col1']%2 == 0)  &  (df['col2']%2 == 0)]
```

	col1	col2	col3	col4
A	10	46	39	30
C	14	48	42	37
H	38	20	49	41
J	10	20	30	40
K	100	200	300	400

```
# Aggregation functions (Series)
# .sum, .max, .min, .median .........
df['col1'].sum()
```

```
332
```

```
df['col1'].median()
```

```
23.0
```

Reading external files

```
#create a file
# upload it in the left pan and then use the following code
df_student = pd.read_excel('/content/sample_data/stud.xlsx', sheet_name='Sheet1')
```

```
df_student
```

	roll	name	marks	address	dept
0	1	amol	7	thane	CS
1	2	vijaya	8	mulund	IT
2	3	vrushali	6	Kopar	IT
3	4	amey	7	kalyan	CS
4	5	Manish	5	kurla	IT
5	6	rajat	6	dahisar	CS
6	7	tejal	6	mulund	CS

```
# Avg marks of all students
df_student['marks'].mean()
```

```
6.428571428571429
```

```
# Select students from Pune
df_student[df_student['address'] == "mulund"]
```

	roll	name	marks	address	dept
1	2	vijaya	8	mulund	IT
6	7	tejal	6	mulund	CS

```
df_student[df_student['address'] == "thane"]['marks'].mean()
```

```
7.0
```

```
df_student[df_student['address'] == "vasai"]['marks'].mean()
```

```
nan
```

```
df_student[df_student['address'] == 'mulund']['marks'].mean()
```

```
7.0
```

```
df_student
```

	roll	name	marks	address	dept
0	1	bhuvan	76	mulund	1
1	2	sonali	45	thane	2
2	3	vijaya	66	nahur	1
3	4	vrushali	78	vasai	2
4	5	amey	89	andheri	1
5	6	amol	78	thane	2

```
# List of Names of students whoose marks are more than avg marks
list(df_student[df_student['marks'] > df_student['marks'].mean()]['name'])
```

```
['amol', 'vijaya', 'amey']
```

```
len(df_student['address'].unique())
```

```
6
```

```
df_student['address'].value_counts()
```

address	count
mulund	2
thane	1
Kopar	1
kalyan	1
kurla	1
dahisar	1

dtype: int64

```
df_student.groupby('dept').max()['marks']   # #####
```

dept	marks
CS	7
IT	8

dtype: int64

```
df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-00cf07b74dcd> in <cell line: 1>()
----> 1 df

NameError: name 'df' is not defined
```

```
df.sum()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-7e5fdb616c56> in <cell line: 1>()
----> 1 df.sum()

NameError: name 'df' is not defined
```

```
df.sum(axis=1)
```

Series functions -> unique,nunique, value_counts

```
df_student = pd.read_excel('stud_db.xlsx', sheet_name='XII')
df_dept = pd.read_excel('stud_db.xlsx', sheet_name='Depts')
```

```
df_student
```

	Roll	Name	Marks	Address	Dept
0	1	qw	76	Pune	1
1	2	df	45	Mumbai	1
2	3	vc	56	Nagpur	1
3	4	bn	78	Pune	2
4	5	hj	74	Mumbai	2
5	6	yu	49	Nagpur	2
6	7	fg	68	Pune	2
7	8	tr	84	Mumbai	1

```
df_student['address'].unique()
```

```
array(['mulund', 'thane', 'nahur', 'vasai', 'andheri'], dtype=object)
```

```
len(df_student['address'].unique())
```

```
5
```

```
df_student['address'].nunique()
```

```
5
```

```
df_student['address'].value_counts()
```

```
thane      2
mulund     1
nahur      1
vasai      1
andheri    1
Name: address, dtype: int64
```

```
df
```

	col1	col2	col3	col4	total
A	38	44	11	19	82
B	44	41	49	43	85
C	35	42	16	10	77
D	20	41	38	15	61
E	42	25	35	22	67
F	49	28	28	44	77
G	43	38	17	12	81
H	39	29	32	34	68
I	40	30	21	33	70
J	10	20	30	40	50
K	100	200	300	400	500
TOTAL	460	538	577	672	1218

```
def get_code(city):
    return city[:2]
```

```
get_code('thane')
```

```
'th'
```

```
df_student['address'].apply(get_code)
```

```
0    mu
1    th
2    na
3    va
4    an
5    th
Name: address, dtype: object
```

```
df_student['address'].apply(lambda X:X[:2])
```

```
0    mu
1    th
2    na
3    va
4    an
5    th
Name: address, dtype: object
```

```
def get_marks(m,d):
    return m+5 if d==1 else m+2
```

```
get_marks(50,1)
```

```
55
```

```
get_marks(50,2)
```

```
52
```

```
df_student[['marks','dept']].apply(lambda X:get_marks(X[0], X[1]),axis=1)
```

```
0    81
1    47
2    71
3    80
4    94
5    80
dtype: int64
```

```
df_student[['marks','dept']].apply(get_marks,axis=1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-74-4d52fd5081cb> in <cell line: 1>()
----> 1 df_student[['marks','dept']].apply(get_marks,axis=1)

                              3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/apply.py in apply_series_generator(self)
    905             for i, v in enumerate(series_gen):
    906                 # ignore SettingWithCopy here in case the user mutates
--> 907                 results[i] = self.f(v)
    908                 if isinstance(results[i], ABCSeries):
    909                     # If we have a view on v, we need to make a copy because

TypeError: get_marks() missing 1 required positional argument: 'd'
```

```
df_student['Grace_M']=df_student[['Marks','Dept']].apply(lambda X: X[0]+5 if X[1]==1 else X[0]+
```

```
df_student
```

	Roll	Name	Marks	Address	Dept	Grace_M
0	1	qw	76	Pune	1	81
1	2	df	45	Mumbai	1	50
2	3	vc	56	Nagpur	1	61
3	4	bn	78	Pune	2	80
4	5	hj	74	Mumbai	2	76
5	6	yu	49	Nagpur	2	51
6	7	fg	68	Pune	2	70
7	8	tr	84	Mumbai	1	89

```
df_student['Address'].value_counts()
```

```
Mumbai    3
Pune      3
Nagpur    2
Name: Address, dtype: int64
```

```
df_student.groupby('Address').mean()['Marks']
```

```
Address
Mumbai    67.666667
Nagpur    52.500000
Pune      74.000000
Name: Marks, dtype: float64
```

```
df_student.groupby('Dept').max()['Marks']
```

```
Dept
1    84
2    78
Name: Marks, dtype: int64
```

```
# Merging of Df's
df1 = pd.DataFrame({'col1':[1,2,3,4],
                    'col2':[5,6,7,8],
                    'col3':[9,9,8,7]})
df2 = pd.DataFrame({'col1':[1,2],
                    'col2':[5,6],
                    'col3':[9,9]})
df3 = pd.DataFrame({'col4':[11,12,13,14],
                    'col5':[21,22,23,24]})
```

```
df1
```

	col1	col2	col3
0	1	5	9
1	2	6	9
2	3	7	8
3	4	8	7

```
df2
```

	col1	col2	col3
0	1	5	9
1	2	6	9

```
df3
```

	col4	col5
0	11	21
1	12	22
2	13	23
3	14	24

```
pd.concat([df1,df2],axis=0)
```

	col1	col2	col3
0	1	5	9
1	2	6	9
2	3	7	8
3	4	8	7
0	1	5	9
1	2	6	9

```
pd.concat([df1,df3],axis=1)
```

	col1	col2	col3	col4	col5
0	1	5	9	11	21
1	2	6	9	12	22
2	3	7	8	13	23
3	4	8	7	14	24

```
pd.concat([df2,df3],axis=0)
```

	col1	col2	col3	col4	col5
0	1.0	5.0	9.0	NaN	NaN
1	2.0	6.0	9.0	NaN	NaN
0	NaN	NaN	NaN	11.0	21.0
1	NaN	NaN	NaN	12.0	22.0
2	NaN	NaN	NaN	13.0	23.0
3	NaN	NaN	NaN	14.0	24.0

```
def get_corners2D(X):
    return X[::X.shape[0]-1,::X.shape[1]-1]

X = A = np.random.randint(1,10,(5,7))
A
```

```
array([[9, 7, 6, 1, 6, 9, 7],
       [9, 2, 5, 8, 1, 4, 4],
       [5, 1, 1, 8, 7, 1, 9],
       [8, 7, 2, 2, 3, 4, 4],
       [8, 1, 2, 1, 5, 1, 9]])
```

```
get_corners2D(A)
```

```
array([[3, 9],
       [6, 5]])
```

```
X[0:len(X):len(X)-1 , 0:len(X[0]):len(X[0])-1]
```

```
array([[9, 7],
       [8, 9]])
```

```
np.linspace()
```

```
df_student = pd.read_excel('stud_db.xlsx',sheet_name='XII')
df_depts = pd.read_excel('stud_db.xlsx',sheet_name='Depts')

print(df_student, df_depts, sep='\n')
```

```
   Roll Name  Marks Address  Dept
0     1   qw     76    Pune     1
1     2   df     45  Mumbai     1
2     3   vc     56  Nagpur     1
3     4   bn     78    Pune     2
4     5   hj     74  Mumbai     2
5     6   yu     49  Nagpur     2
6     7   fg     68    Pune     2
7     8   tr     84  Mumbai     1
   dept_id          Name  HOD
0        1  Comp Science  XYZ
1        2    Statistics  PQR
2        3         Maths  MNO
```

```
df = pd.merge(df_student,df_depts,left_on='Dept',right_on='dept_id', suffixes=('_stud','_dept')

df
```

	Roll	Name_stud	Marks	Address	Dept	dept_id	Name_dept	HOD
0	1	qw	76	Pune	1	1	Comp Science	XYZ
1	2	df	45	Mumbai	1	1	Comp Science	XYZ
2	3	vc	56	Nagpur	1	1	Comp Science	XYZ
3	8	tr	84	Mumbai	1	1	Comp Science	XYZ
4	4	bn	78	Pune	2	2	Statistics	PQR
5	5	hj	74	Mumbai	2	2	Statistics	PQR
6	6	yu	49	Nagpur	2	2	Statistics	PQR

Pandas Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

Need of pandas to Play wit data and find answers of :

Is there a correlation between two or more columns? What is average value? Max value? Min value?

```
#numpy and pandas to work on..
import pandas as pd
import io
from google.colab import files
uploaded=files.upload()
df=pd.read_csv(io.BytesIO(uploaded['marks.csv']))

print(df)
```

Choose Files marks.csv

- **marks.csv**(text/csv) - 48 bytes, last modified: 22/8/2023 - 100% done

```
Saving marks.csv to marks.csv
  name  m1  m2
0    a  44  54
1    b  55  56
2    c  66  65
3    d  55  65
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dr
```

Start coding or generate with AI.

```
import numpy as np
col1=df['name']
print(col1)
m1=df['m1']
m2=df['m2']
print('marks are ', m1,m2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-0fd58140e74b> in <cell line: 2>()
      1 import numpy as np
----> 2 col1=df['name']
      3 print(col1)
      4 m1=df['m1']
      5 m2=df['m2']

NameError: name 'df' is not defined
```

```
total=m1+m2
print('\n sum of m1 and m2 \n',total)
```

```
 sum of m1 and m2
 0    132
1    144
2    151
3    154
4    133
dtype: int64
```

```
print('mean is ', total.mean())
```

```
mean is  142.8
```

```
print('maxi marks of m1', m1.max())
print('mini score of m2 ',m2.min())
print('\n extracting marks if more than 5', m1>85)
```

```
maxi marks of m1 89
mini score of m2  45

 extracting marks if more than 5 0    False
1    False
2    False
3     True
4     True
Name: m1, dtype: bool
```

```
print('only 1st 2 row \n ',df[:3])
```

```
only 1st 2 row
    name  m1  m2
0    a  55  77
1    b  66  78
2    c  56  95
```

```
print('extract data of name \n ' ,df[1:])
```

```
extract data of name
    name  m1  m2
1    b  66  78
2    c  56  95
3    d  89  65
4    e  88  45
```

```
print(m1)
print('count the occurance of element in the column',np.bincount(m1))
```

```
0    44
1    55
2    66
3    55
Name: m1, dtype: int64
count the occurance of element in the column [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 1]
```

```
#create an array of 3X3 and perform
a=np.random.randint(1,15,(3,3))
print(a)
```

```
[[ 1  2  8]
 [ 7 13  9]
 [ 2  3  1]]
```

```
#create identity matirx
b=np.eye(3)
print(b)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
c=a+b
print (c)
```

```
[[ 2.  2.  8.]
 [ 7. 14.  9.]
 [ 2.  3.  2.]]
```

```
#stack numpy arrays k=jining of 2 arrays
#use a and c
st=np.vstack((a,c)) #vertical joining
print (st)
st1=np.hstack((a,c)) #horizontal joining
print(st1)
```

```
[[ 2.  3.  8.]
 [ 6.  2. 14.]
 [ 6. 12.  8.]
 [ 3.  3.  8.]
 [ 6.  3. 14.]
 [ 6. 12.  9.]]
[[ 2.  3.  8.  3.  3.  8.]
 [ 6.  2. 14.  6.  3. 14.]
 [ 6. 12.  8.  6. 12.  9.]]
```

```
print(st[st%3==0])
```

```
[ 3.  6.  6. 12.  3.  3.  6.  3.  6. 12.  9.]
```

create dataset with attributes (Emp_no,name,basic_sal,TA,DA,IT,other_deductions,gross_sal,Take_away)

1. enter 10 records with Emp_no,name and Basci_sal.
2. Compute TA,DA,other_deductions (use your own formula)
3. compute gross sal and take_away based on computations

```
#to read a file
#create a file in notepad with some data

file = open("/content/sample_data/sample.txt")
```

```
for i in file:
    print(i)
```

```
10

20

30
```

Start coding or generate with AI.

```
#read contents from a csv
#importing required library
from pandas import read_csv

data = read_csv('/content/sample_data/data3.csv')

df = data.values
print(df)
```

```
[[1. 6. 4.]
 [2. 4. 8.]
 [3. 9. 1.]]
```

```
#create an array and save it in csv
# import numpy library
import numpy as np

# create an array
a = np.array([[3,3,3] ,
              [1, 6, 4] ,
                 [2, 4, 8] ,
                 [3, 9, 1]],dtype=np.int32)

# save array into csv file
print(a)
np.savetxt("/content/sample_data/data3.csv", a,delimiter = ",")
#download the file and chck contents
```

```
[[3 3 3]
 [1 6 4]
 [2 4 8]
 [3 9 1]]
```