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

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

In [2]:
```
import numpy as np
q=[1,2,3,4,5]
w=np.array([1,2,3,4,5])
w
```

Out[2]:
```
array([1, 2, 3, 4, 5])
```

In [4]:
```
import pandas as pd
pd.Series(1)      #data point is given a lot of importance
#int64 means this data has an ability to express itself maximum by 64 bits of
 data
```

Out[4]:
```
0    1
dtype: int64
```

In [5]:
```
pd.Series(1+6j)   #why complex128?- because 64 and 64 to represent both real a
nd imaginary part .
```

Out[5]:
```
0    1.000000+6.000000j
dtype: complex128
```

In [7]:
```
pd.Series('python')
#whenever we write alphabet , special character or space ,data type is an obje
ct.
```

Out[7]:
```
0    python
dtype: object
```

In [ ]:
```
#heterogenous list - half of the list is numbers and the other half is not num
bers
```

In [8]:
```
a=[1,2,3,'apple','banana','&*#@!']
pd.Series(a)
#there are some changes in the output
# 1)data becomes more vulnerable , it is not protected as the data comes out o
f the list and is not packed.
# 2)Horizontally placed data is changed into a vertical data .
# 3)Data is given specific indices after the package has been removed.
```

Out[8]:
```
0         1
1         2
2         3
3     apple
4    banana
5     &*#@!
dtype: object
```

In [9]:
```
pd.Series(None)
```

```
<ipython-input-9-410dc3eaa75a>:1: DeprecationWarning: The default dtype for e
mpty Series will be 'object' instead of 'float64' in a future version. Specif
y a dtype explicitly to silence this warning.
  pd.Series(None)
```

Out[9]:
```
Series([], dtype: float64)
```

In [15]:
```
plm1 = np.arange(11,21)
plm1
```

Out[15]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

In [17]:
```
plm1=pd.Series(plm1)
plm1
```

Out[17]:
```
0    11
1    12
2    13
3    14
4    15
5    16
6    17
7    18
8    19
9    20
dtype: int32
```

In [20]:
```
plm1[4]
```

Out[20]:
```
15
```

In [21]:
```
plm1[:5]
```

Out[21]:
```
0    11
1    12
2    13
3    14
4    15
dtype: int32
```

In [22]:
```
plm1[5:]
```

Out[22]:
```
5    16
6    17
7    18
8    19
9    20
dtype: int32
```

In [24]:
```
p=pd.Series(plm1)
p.sum()
```

Out[24]:
```
155
```

In [ ]:
```
#In many machine learning models numpy and pandas are imported together .
```

In [25]:
```
p.cumsum()
```

Out[25]:
```
0     11
1     23
2     36
3     50
4     65
5     81
6     98
7    116
8    135
9    155
dtype: int32
```

In [26]:
```
np.sqrt(p)
```

Out[26]:
```
0    3.316625
1    3.464102
2    3.605551
3    3.741657
4    3.872983
5    4.000000
6    4.123106
7    4.242641
8    4.358899
9    4.472136
dtype: float64
```

In [27]:
```
np.mean(p)
```

Out[27]:
```
15.5
```

In [28]:
```
np.cos(p)
```

Out[28]:
```
0    0.004426
1    0.843854
2    0.907447
3    0.136737
4   -0.759688
5   -0.957659
6   -0.275163
7    0.660317
8    0.988705
9    0.408082
dtype: float64
```

In [29]:
```
p[0:5]
```

Out[29]:
```
0    11
1    12
2    13
3    14
4    15
dtype: int32
```

In [ ]:
```
#broadcasting in python means changing the data at one point of time
```

In [31]:
```
p[0:5]=100
p
```

Out[31]:
```
0    100
1    100
2    100
3    100
4    100
5     16
6     17
7     18
8     19
9     20
dtype: int32
```

In [32]:
```
p[0::2]=120
p
```

Out[32]:
```
0    120
1    100
2    120
3    100
4    120
5     16
6    120
7     18
8    120
9     20
dtype: int32
```

In [34]:
```
p.add(345,axis=0)
```

Out[34]:
```
0    465
1    445
2    465
3    445
4    465
5    361
6    465
7    363
8    465
9    365
dtype: int32
```

In [37]:
```
data=[1,2,3,4,5,6]
labels=('a','b','c','d','e','f')
columns=['corona cases','covid death rate','recovery rate','non traced','tota
l','travel history']
pd.DataFrame(data)
```

Out[37]:

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

In [38]:
```
pd.DataFrame(data,index=labels)
```

Out[38]:

	0
a	1
b	2
c	3
d	4
e	5
f	6

In [49]:
```
data=[1,2,3,4,5,6]
labels=('a','b','c','d','e','f')
column=['corona cases']
corona_cases=pd.DataFrame(data,labels,column)
#this is a pandas data frame
corona_cases
```

Out[49]:

	corona cases
a	1
b	2
c	3
d	4
e	5
f	6

In [51]:
```
corona_cases.loc['e']
```

Out[51]:
```
corona cases    5
Name: e, dtype: int64
```

In [ ]:
```
import pandas as pd
pd.concat()  #to concatenate two things
```

In [ ]:
```
#axis = 0 means we are joining along the row
#rows are given more preference in numpy and pandas
#axis = 1 means we are joining along the column(two columns will be created)
```

creating dataframes and operating upon them

In [ ]:
```
#operations like .loc , .iloc , sorting , duplicating , dropping the duplicate
s , appending something on the dataframes etc.
```

In [7]:
```
import pandas as pd
import numpy as np

sales_a = [101,102,103,104,105,106]
sales_b = [107,108,109,110,111,112]
a=pd.Series(sales_a)
a
```

Out[7]:
```
0    101
1    102
2    103
3    104
4    105
5    106
dtype: int64
```

In [8]:
```
b=pd.Series(sales_b)
b
```

Out[8]:
```
0    107
1    108
2    109
3    110
4    111
5    112
dtype: int64
```

if we want to concatenate both the series

In [ ]:
```
# the elements in both the series must be equal to concatenate .
```

In [10]:
```
pd.concat([a,b],axis=0) #axis=0 is default as it is joined alond the row
```

Out[10]:
```
0    101
1    102
2    103
3    104
4    105
5    106
0    107
1    108
2    109
3    110
4    111
5    112
dtype: int64
```

In [11]:
```
pd.concat([a,b],axis=1)  #if axis = 1 we are joining along the columns
```

Out[11]:

	0	1
0	101	107
1	102	108
2	103	109
3	104	110
4	105	111
5	106	112

In [14]:
```
pd.concat([a,b],axis=0,keys=['a','b'])
#here a belongs to column a and b belongs to column b.
#we are bifurcating the data here
```

Out[14]:
```
a  0    101
   1    102
   2    103
   3    104
   4    105
   5    106
b  0    107
   1    108
   2    109
   3    110
   4    111
   5    112
dtype: int64
```

In [15]:
```
pd.concat([a,b],axis=1,keys=['a','b'])
```

Out[15]:

	a	b
0	101	107
1	102	108
2	103	109
3	104	110
4	105	111
5	106	112

In [17]:
```
d={'Sales':[1,2,3,4],'sales_person':['a','b','c','d']}
pd.DataFrame(d)
#here the first key becomes the first column and the second key becomes the se
cond column.
```

Out[17]:

	Sales	sales_person
0	1	a
1	2	b
2	3	c
3	4	d

In [18]:
```
pd.DataFrame(d,index=['a','b','c','d'])
```

Out[18]:

	Sales	sales_person
a	1	a
b	2	b
c	3	c
d	4	d

In [22]:
```
df_prof_info_A = pd.DataFrame({
   'Emp_ID': [1001,1002,1003,1004,1005],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Stephan'],
   'Gender':['M', 'F', 'M', 'F', 'M'],
   'Company':['Apple','Walmart','Intel','Cummins','Ford'],
   'Salary':[67000,90000,87000,69000,78000]},
   index=[101,102,103,104,105])
```

In [23]:
```
df_prof_info_A
```

Out[23]:

	Emp_ID	Name	Gender	Company	Salary
101	1001	Alex	M	Apple	67000
102	1002	Amy	F	Walmart	90000
103	1003	Allen	M	Intel	87000
104	1004	Alice	F	Cummins	69000
105	1005	Stephan	M	Ford	78000

In [24]:
```
df_prof_info_A.ndim  #number of dimensions
```

Out[24]:
```
2
```

In [25]:
```
df_prof_info_A.shape  #number of rows and columns
```

Out[25]:
```
(5, 5)
```

In [27]:
```
df_prof_info_A.info()    #whole information about the data frame
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 101 to 105
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Emp_ID   5 non-null      int64 
 1   Name     5 non-null      object
 2   Gender   5 non-null      object
 3   Company  5 non-null      object
 4   Salary   5 non-null      int64 
dtypes: int64(2), object(3)
memory usage: 240.0+ bytes
```

In [28]:
```
df_prof_info_A.describe()   #describe is one type of descriptive statistics
#it works only on numerical data because it is a type of descriptive statistic
s
#mean - average of all the salaries
#std - standard deviation
#25% - 1st quartile
#50% - median
#75% - 3rd quartile
```

Out[28]:

	Emp_ID	Salary
count	5.000000	5.000000
mean	1003.000000	78200.000000
std	1.581139	10329.569207
min	1001.000000	67000.000000
25%	1002.000000	69000.000000
50%	1003.000000	78000.000000
75%	1004.000000	87000.000000
max	1005.000000	90000.000000

In [29]:
```
df_prof_info_A['Gender']
#we are slicing out the gender column
```

Out[29]:
```
101    M
102    F
103    M
104    F
105    M
Name: Gender, dtype: object
```

In [30]:
```
df_prof_info_A[df_prof_info_A['Gender']=='F']
#we are extracting the females here
```

Out[30]:

	Emp_ID	Name	Gender	Company	Salary
102	1002	Amy	F	Walmart	90000
104	1004	Alice	F	Cummins	69000

In [31]:
```
df_prof_info_A[df_prof_info_A['Salary']>69000]
```

Out[31]:

	Emp_ID	Name	Gender	Company	Salary
102	1002	Amy	F	Walmart	90000
103	1003	Allen	M	Intel	87000
105	1005	Stephan	M	Ford	78000

In [34]:
```
df_prof_info_A[df_prof_info_A['Salary']>69000]['Company']
```

Out[34]:
```
102    Walmart
103      Intel
105       Ford
Name: Company, dtype: object
```

In [37]:
```
df_prof_info_A[df_prof_info_A['Salary']>69000][['Name','Company']]
```

Out[37]:

	Name	Company
102	Amy	Walmart
103	Allen	Intel
105	Stephan	Ford

In [38]:
```
df_prof_info_A[ (df_prof_info_A['Salary']>=69000) &  (df_prof_info_A['Salary']
<=78000)]
```

Out[38]:

	Emp_ID	Name	Gender	Company	Salary
104	1004	Alice	F	Cummins	69000
105	1005	Stephan	M	Ford	78000

In [39]:
```
df_prof_info_A['Salary'].sum()  #sum of all the salaries
#.max and .min to find the maximum and minimum values
```

Out[39]:
```
391000
```

In [41]:
```
df_prof_info_A['Salary'].apply(float)
#converted the data into float
```

Out[41]:
```
101    67000.0
102    90000.0
103    87000.0
104    69000.0
105    78000.0
Name: Salary, dtype: float64
```

In [42]:
```
df_prof_info_A['Salary'].apply(lambda num:num**2)
```

Out[42]:
```
101    4489000000
102    8100000000
103    7569000000
104    4761000000
105    6084000000
Name: Salary, dtype: int64
```

In [46]:
```
df_prof_info_A
```

Out[46]:

	Emp_ID	Name	Gender	Company	Salary
101	1001	Alex	M	Apple	67000
102	1002	Amy	F	Walmart	90000
103	1003	Allen	M	Intel	87000
104	1004	Alice	F	Cummins	69000
105	1005	Stephan	M	Ford	78000

In [49]:
```
df_prof_info_A.loc[104]     #iloc is for index
```

Out[49]:
```
Emp_ID        1004
Name         Alice
Gender           F
Company    Cummins
Salary       69000
Name: 104, dtype: object
```

In [53]:
```
qwe=df_prof_info_A.loc[101:103]
qwe
```

Out[53]:

	Emp_ID	Name	Gender	Company	Salary
101	1001	Alex	M	Apple	67000
102	1002	Amy	F	Walmart	90000
103	1003	Allen	M	Intel	87000

In [70]:
```
df_prof_info_A
```

Out[70]:

	Emp_ID	Name	Gender	Company	Salary
106	1006	Billie	M	Cognizant	89000
107	1007	Brian	M	Apple	80000
108	1008	Bran	M	Intel	79000
109	1009	Bryce	F	Cummins	97000
110	1010	Betty	F	Walmart	88000
111	1011	James	M	Intel	89000

In [71]:
```
df_prof_info_A = pd.DataFrame({
   'Emp_ID': [1001,1002,1003,1004,1005],
   'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Stephan'],
   'Gender':['M', 'F', 'M', 'F', 'M'],
   'Company':['Apple','Walmart','Intel','Cummins','Ford'],
   'Salary':[67000,90000,87000,69000,78000]},
   index=[101,102,103,104,105])
df_prof_info_A
```

Out[71]:

	Emp_ID	Name	Gender	Company	Salary
101	1001	Alex	M	Apple	67000
102	1002	Amy	F	Walmart	90000
103	1003	Allen	M	Intel	87000
104	1004	Alice	F	Cummins	69000
105	1005	Stephan	M	Ford	78000

In [63]:
```
df_pers_info_A = pd.DataFrame({
     'Emp_ID': [1001,1002,1003,1004,1005],
   'Hometown': ['New York', 'London', 'San Francisco', 'Seattle', 'Madrid'],
     'Gender':['M', 'F', 'M', 'F', 'M'],
    'Marital':['Married','Divorsed','Single','Married','Single'],
   'Dependents':[1,1,3,2,1]},
   index=[101,102,103,104,105])
df_pers_info_A
```

Out[63]:

	Emp_ID	Hometown	Gender	Marital	Dependents
101	1001	New York	M	Married	1
102	1002	London	F	Divorsed	1
103	1003	San Francisco	M	Single	3
104	1004	Seattle	F	Married	2
105	1005	Madrid	M	Single	1

In [ ]:
```
#we will try to concatenate the above two dataframes
```

In [72]:
```
abc=pd.concat([df_prof_info_A,df_pers_info_A],axis=1)
abc
```

Out[72]:

	Emp_ID	Name	Gender	Company	Salary	Emp_ID	Hometown	Gender	Marital	Depen
101	1001	Alex	M	Apple	67000	1001	New York	M	Married	
102	1002	Amy	F	Walmart	90000	1002	London	F	Divorsed	
103	1003	Allen	M	Intel	87000	1003	San Francisco	M	Single	
104	1004	Alice	F	Cummins	69000	1004	Seattle	F	Married	
105	1005	Stephan	M	Ford	78000	1005	Madrid	M	Single	

In [73]:
```
abc.T.drop_duplicates().T
```

Out[73]:

	Emp_ID	Name	Gender	Company	Salary	Hometown	Marital	Dependents
101	1001	Alex	M	Apple	67000	New York	Married	1
102	1002	Amy	F	Walmart	90000	London	Divorsed	1
103	1003	Allen	M	Intel	87000	San Francisco	Single	3
104	1004	Alice	F	Cummins	69000	Seattle	Married	2
105	1005	Stephan	M	Ford	78000	Madrid	Single	1

In [74]:
```
abc.drop('Emp_ID',axis=1,inplace=True)
```

In [ ]:
```
# .append (two data frames are added together)
```

multi index data frame

In [75]:
```
labels = pd.MultiIndex.from_product([['Before Course','After Course'],['Python','ML']])


marks = [[82,95,12,90],[78,89,67,76],[78,87,89,90],[76,89,56,65],[66,89,56,87]]

df_marks = pd.DataFrame(data = marks, index = ['Alisa','Bobby','Cathrine','Jack','Mia'], columns = labels)


df_marks
```

Out[75]:

	Before Course		After Course	
	Python	ML	Python	ML
Alisa	82	95	12	90
Bobby	78	89	67	76
Cathrine	78	87	89	90
Jack	76	89	56	65
Mia	66	89	56	87

In [77]:
```
df_marks.stack().stack()
```

Out[77]:
```
Alisa     ML      After Course     90
                  Before Course    95
          Python  After Course     12
                  Before Course    82
Bobby     ML      After Course     76
                  Before Course    89
          Python  After Course     67
                  Before Course    78
Cathrine  ML      After Course     90
                  Before Course    87
          Python  After Course     89
                  Before Course    78
Jack      ML      After Course     65
                  Before Course    89
          Python  After Course     56
                  Before Course    76
Mia       ML      After Course     87
                  Before Course    89
          Python  After Course     56
                  Before Course    66
dtype: int64
```

In [76]:
```
df_marks.unstack()
```

Out[76]:
```
Before Course  Python  Alisa       82
                       Bobby       78
                       Cathrine    78
                       Jack        76
                       Mia         66
               ML      Alisa       95
                       Bobby       89
                       Cathrine    87
                       Jack        89
                       Mia         89
After Course   Python  Alisa       12
                       Bobby       67
                       Cathrine    89
                       Jack        56
                       Mia         56
               ML      Alisa       90
                       Bobby       76
                       Cathrine    90
                       Jack        65
                       Mia         87
dtype: int64
```

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

case study :

In [ ]:
```
#seaborn-python visaulization library
#matplotlib
#ordinal data - the data is in some kind of order
```

In [1]:
```
sales = {'Months': ['Jan','Feb','Mar','Apr','May','June','July','Aug','Sep','Oct','Nov','Dec'],
        'Sales': [22000,27000,25000,29000,35000,67000,78000,67000,56000,56000,89000,60000],
         'Seasons': ['Winter','Winter','Spring','Spring','Spring','Summer','Summer','Summer','Fall','Fall','Fall','Winter']
        }
```

In [6]:
```
df_sales = pd.DataFrame(sales)
df_sales    #  months(ordinal) , Sales(numerical), Seasons(categorical) columns
```

Out[6]:

	Months	Sales	Seasons
0	Jan	22000	Winter
1	Feb	27000	Winter
2	Mar	25000	Spring
3	Apr	29000	Spring
4	May	35000	Spring
5	June	67000	Summer
6	July	78000	Summer
7	Aug	67000	Summer
8	Sep	56000	Fall
9	Oct	56000	Fall
10	Nov	89000	Fall
11	Dec	60000	Winter

In [7]:
```
pd.pivot_table(df_sales,index = ['Seasons'],values = ['Sales'])
```

Out[7]:

	Sales
Seasons	
Fall	67000.000000
Spring	29666.666667
Summer	70666.666667
Winter	36333.333333

In [8]:
```
pd.pivot_table(df_sales,index = ['Months'],values = ['Sales'])
```

Out[8]:

Months	Sales
Apr	29000
Aug	67000
Dec	60000
Feb	27000
Jan	22000
July	78000
June	67000
Mar	25000
May	35000
Nov	89000
Oct	56000
Sep	56000

In [9]:
```
pd.pivot_table(df_sales,index = ['Seasons'],values = ['Sales'],aggfunc=np.mean
) #np.sum will calculate the total sales
```

Out[9]:

Seasons	Sales
Fall	67000.000000
Spring	29666.666667
Summer	70666.666667
Winter	36333.333333

In [ ]:
```
 #groupby means studying the characteristics of a population group
```

In [ ]:
```
#two methods in dataframes - 1)Creating the data sets 2)Reading of the dataset
s
```

In [13]:
```
data1=pd.read_csv('C:\\Users\\Vishal Venkata\\Downloads\\people.csv')  #to rea
d the file in some other location
```

In [ ]:
```
# data=pd.read_csv('people.csv') - if the jupyter notebook and the file we are
working on are in the same location.
```

In [15]:
```
pwd
```

Out[15]:
```
'C:\\Users\\Vishal Venkata\\Desktop\\Data Science\\Python Labs\\Self Prepared
Jupyter Notes'
```

In [17]:
```
data1.head()   #we get the first 5 rows
```

Out[17]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	34	Male	Delhi	155	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes

In [18]:
```
data1.tail()    #we get the last 5 rows
```

Out[18]:

	Age	Gender	Hometown	Height	Qualification	Smoker
16	56	Female	Mumbai	151	Graduate	No
17	41	Male	Bangalore	180	Diploma	No
18	56	Male	Delhi	182	Graduate	Yes
19	53	Male	Bangalore	156	Post-graduate	Yes
20	21	Female	Bangalore	158	Post-graduate	No

In [19]:
```
data1.info()    #information of the data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Age            21 non-null     int64 
 1   Gender         21 non-null     object
 2   Hometown       21 non-null     object
 3   Height         21 non-null     int64 
 4   Qualification  21 non-null     object
 5   Smoker         21 non-null     object
dtypes: int64(2), object(4)
memory usage: 1.1+ KB
```

In [20]:
```
data1.describe()   # .describe will work only on the numerical data
#descriptive statistics
```

Out[20]:

	Age	Height
count	21.000000	21.000000
mean	42.333333	166.190476
std	11.858893	8.919748
min	21.000000	151.000000
25%	34.000000	158.000000
50%	43.000000	167.000000
75%	54.000000	175.000000
max	59.000000	182.000000

In [21]:
```
data1.describe(include='all')
```

Out[21]:

	Age	Gender	Hometown	Height	Qualification	Smoker
count	21.000000	21	21	21.000000	21	21
unique	NaN	2	3	NaN	3	2
top	NaN	Male	Mumbai	NaN	Graduate	No
freq	NaN	11	9	NaN	8	11
mean	42.333333	NaN	NaN	166.190476	NaN	NaN
std	11.858893	NaN	NaN	8.919748	NaN	NaN
min	21.000000	NaN	NaN	151.000000	NaN	NaN
25%	34.000000	NaN	NaN	158.000000	NaN	NaN
50%	43.000000	NaN	NaN	167.000000	NaN	NaN
75%	54.000000	NaN	NaN	175.000000	NaN	NaN
max	59.000000	NaN	NaN	182.000000	NaN	NaN

In [24]:
```
data1.ndim
```

Out[24]: 2

In [25]:
```
data1.shape
```

Out[25]: (21, 6)

In [27]:
```
data1['Age'].dtype    #to know the data type
```

Out[27]: dtype('int64')

In [28]:
```
data1['Gender']=='Female'
```

Out[28]:
```
0     False
1      True
2     False
3     False
4      True
5     False
6      True
7      True
8     False
9      True
10    False
11     True
12    False
13     True
14    False
15     True
16     True
17    False
18    False
19    False
20     True
Name: Gender, dtype: bool
```

In [30]:
```
data1[data1['Gender']=='Female']
```

Out[30]:

	Age	Gender	Hometown	Height	Qualification	Smoker
1	23	Female	Mumbai	170	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes
6	44	Female	Mumbai	165	Graduate	Yes
7	56	Female	Delhi	170	Diploma	Yes
9	49	Female	Bangalore	162	Post-graduate	Yes
11	28	Female	Mumbai	165	Diploma	No
13	59	Female	Mumbai	175	Graduate	No
15	43	Female	Mumbai	155	Post-graduate	No
16	56	Female	Mumbai	151	Graduate	No
20	21	Female	Bangalore	158	Post-graduate	No

In [32]:
```
female_data=data1[data1['Gender']=='Female']
female_data['Age'].mean()
```

Out[32]:
```
42.2
```

In [37]:
```
female_data[female_data['Height']>162]
```

Out[37]:

	Age	Gender	Hometown	Height	Qualification	Smoker
1	23	Female	Mumbai	170	Graduate	No
6	44	Female	Mumbai	165	Graduate	Yes
7	56	Female	Delhi	170	Diploma	Yes
11	28	Female	Mumbai	165	Diploma	No
13	59	Female	Mumbai	175	Graduate	No

In [38]:
```
female_data[female_data['Height']>162]['Hometown']
```

Out[38]:
```
1     Mumbai
6     Mumbai
7      Delhi
11    Mumbai
13    Mumbai
Name: Hometown, dtype: object
```

In [40]:
```
data1.head()
```

Out[40]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	34	Male	Delhi	155	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes

In [41]:
```
data1.isna()
```

Out[41]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	False	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False

In [42]:
```
data1.head()
```

Out[42]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	34	Male	Delhi	155	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes

In [43]:
```
data1.isna()   # NaN represents missing data in pandas
# this command is to check whether there are any missing values
```

Out[43]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	False	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False

In [44]:
```
data1.isnull()
```

Out[44]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False
10	False	False	False	False	False	False
11	False	False	False	False	False	False
12	False	False	False	False	False	False
13	False	False	False	False	False	False
14	False	False	False	False	False	False
15	False	False	False	False	False	False
16	False	False	False	False	False	False
17	False	False	False	False	False	False
18	False	False	False	False	False	False
19	False	False	False	False	False	False
20	False	False	False	False	False	False

In [48]:
```
[data1['Age'].sort_values]
```

Out[48]:
```
[<bound method Series.sort_values of 0     45
 1     23
 2     27
 3     34
 4     43
 5     34
 6     44
 7     56
 8     34
 9     49
 10    35
 11    28
 12    54
 13    59
 14    54
 15    43
 16    56
 17    41
 18    56
 19    53
 20    21
 Name: Age, dtype: int64>]
```

In [49]:
```
data1.sort_values('Age',ascending=False) #it will arrange in descending order
```

Out[49]:

	Age	Gender	Hometown	Height	Qualification	Smoker
13	59	Female	Mumbai	175	Graduate	No
18	56	Male	Delhi	182	Graduate	Yes
7	56	Female	Delhi	170	Diploma	Yes
16	56	Female	Mumbai	151	Graduate	No
14	54	Male	Bangalore	175	Diploma	Yes
12	54	Male	Delhi	170	Diploma	Yes
19	53	Male	Bangalore	156	Post-graduate	Yes
9	49	Female	Bangalore	162	Post-graduate	Yes
0	45	Male	Mumbai	167	Graduate	Yes
6	44	Female	Mumbai	165	Graduate	Yes
4	43	Female	Mumbai	157	Post-graduate	Yes
15	43	Female	Mumbai	155	Post-graduate	No
17	41	Male	Bangalore	180	Diploma	No
10	35	Male	Bangalore	160	Graduate	No
8	34	Male	Delhi	175	Post-graduate	No
5	34	Male	Delhi	167	Diploma	Yes
3	34	Male	Delhi	155	Graduate	No
11	28	Female	Mumbai	165	Diploma	No
2	27	Male	Mumbai	175	Post-graduate	No
1	23	Female	Mumbai	170	Graduate	No
20	21	Female	Bangalore	158	Post-graduate	No

In [50]:
```
data1.sort_values('Age',ascending=True) #data arranged in ascending order
```

Out[50]:

	Age	Gender	Hometown	Height	Qualification	Smoker
20	21	Female	Bangalore	158	Post-graduate	No
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
11	28	Female	Mumbai	165	Diploma	No
3	34	Male	Delhi	155	Graduate	No
5	34	Male	Delhi	167	Diploma	Yes
8	34	Male	Delhi	175	Post-graduate	No
10	35	Male	Bangalore	160	Graduate	No
17	41	Male	Bangalore	180	Diploma	No
4	43	Female	Mumbai	157	Post-graduate	Yes
15	43	Female	Mumbai	155	Post-graduate	No
6	44	Female	Mumbai	165	Graduate	Yes
0	45	Male	Mumbai	167	Graduate	Yes
9	49	Female	Bangalore	162	Post-graduate	Yes
19	53	Male	Bangalore	156	Post-graduate	Yes
12	54	Male	Delhi	170	Diploma	Yes
14	54	Male	Bangalore	175	Diploma	Yes
16	56	Female	Mumbai	151	Graduate	No
18	56	Male	Delhi	182	Graduate	Yes
7	56	Female	Delhi	170	Diploma	Yes
13	59	Female	Mumbai	175	Graduate	No

In [51]:
```
data1.sort_values('Age',ascending=True,ignore_index = True)
```

Out[51]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	21	Female	Bangalore	158	Post-graduate	No
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	28	Female	Mumbai	165	Diploma	No
4	34	Male	Delhi	155	Graduate	No
5	34	Male	Delhi	167	Diploma	Yes
6	34	Male	Delhi	175	Post-graduate	No
7	35	Male	Bangalore	160	Graduate	No
8	41	Male	Bangalore	180	Diploma	No
9	43	Female	Mumbai	157	Post-graduate	Yes
10	43	Female	Mumbai	155	Post-graduate	No
11	44	Female	Mumbai	165	Graduate	Yes
12	45	Male	Mumbai	167	Graduate	Yes
13	49	Female	Bangalore	162	Post-graduate	Yes
14	53	Male	Bangalore	156	Post-graduate	Yes
15	54	Male	Delhi	170	Diploma	Yes
16	54	Male	Bangalore	175	Diploma	Yes
17	56	Female	Mumbai	151	Graduate	No
18	56	Male	Delhi	182	Graduate	Yes
19	56	Female	Delhi	170	Diploma	Yes
20	59	Female	Mumbai	175	Graduate	No

In [52]:
```
data1.sort_values(['Height','Age'],ascending=[True,False],ignore_index=True)
```

Out[52]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	56	Female	Mumbai	151	Graduate	No
1	43	Female	Mumbai	155	Post-graduate	No
2	34	Male	Delhi	155	Graduate	No
3	53	Male	Bangalore	156	Post-graduate	Yes
4	43	Female	Mumbai	157	Post-graduate	Yes
5	21	Female	Bangalore	158	Post-graduate	No
6	35	Male	Bangalore	160	Graduate	No
7	49	Female	Bangalore	162	Post-graduate	Yes
8	44	Female	Mumbai	165	Graduate	Yes
9	28	Female	Mumbai	165	Diploma	No
10	45	Male	Mumbai	167	Graduate	Yes
11	34	Male	Delhi	167	Diploma	Yes
12	56	Female	Delhi	170	Diploma	Yes
13	54	Male	Delhi	170	Diploma	Yes
14	23	Female	Mumbai	170	Graduate	No
15	59	Female	Mumbai	175	Graduate	No
16	54	Male	Bangalore	175	Diploma	Yes
17	34	Male	Delhi	175	Post-graduate	No
18	27	Male	Mumbai	175	Post-graduate	No
19	41	Male	Bangalore	180	Diploma	No
20	56	Male	Delhi	182	Graduate	Yes

In [53]:
```
data1.head()
```

Out[53]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	34	Male	Delhi	155	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes

Find out the gender of the person with maximum height .

In [56]:
```
data1['Height'].max()
```

Out[56]:
```
182
```

In [57]:
```
data1[data1['Height']==182]
```

Out[57]:

	Age	Gender	Hometown	Height	Qualification	Smoker
18	56	Male	Delhi	182	Graduate	Yes

In [58]:
```
data1[data1['Height']==182]['Gender']
```

Out[58]:
```
18    Male
Name: Gender, dtype: object
```

In [60]:
```
pd.pivot_table(data1,index=['Qualification'],values=['Age'])
```

Out[60]:

	Age
Qualification	
Diploma	44.500000
Graduate	44.000000
Post-graduate	38.571429

In [61]:
```
pd.pivot_table(data1,index=['Smoker'],values=['Age'])
```

Out[61]:

	Age
Smoker	
No	36.454545
Yes	48.800000

In [65]:
```
data1.groupby('Hometown').mean()
```

Out[65]:

	Age	Height
Hometown		
Bangalore	42.166667	165.166667
Delhi	44.666667	169.833333
Mumbai	40.888889	164.444444

we can pull out data from pivot table and groupby methods.

In [67]:
```
data1.head(2)
```

Out[67]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No

In [68]:
```
pd.crosstab(data1['Gender'],data1['Qualification'])
```

Out[68]:

Qualification	Diploma	Graduate	Post-graduate
Gender			
Female	2	4	4
Male	4	4	3

How to fill missing values

In [70]:
```
d={1:[1,2,np.nan],2:[2,3,np.nan]}
d
#we wantedly created a dictionary with NaN values
```

Out[70]:
```
{1: [1, 2, nan], 2: [2, 3, nan]}
```

In [72]:
```
abc=pd.DataFrame(d)
abc
```

Out[72]:

	1	2
0	1.0	2.0
1	2.0	3.0
2	NaN	NaN

In [73]:
```
abc.fillna('Vishal')  # method for filling the missing values
```

Out[73]:

	1	2
0	1	2
1	2	3
2	Vishal	Vishal

In [74]:
```
abc[1].fillna('shambhavi') # if we want to fill only the first coloumn
```

Out[74]:
```
0           1
1           2
2    shambhavi
Name: 1, dtype: object
```

In [80]:
```
data1.head()
```

Out[80]:

	Age	Gender	Hometown	Height	Qualification	Smoker
0	45	Male	Mumbai	167	Graduate	Yes
1	23	Female	Mumbai	170	Graduate	No
2	27	Male	Mumbai	175	Post-graduate	No
3	34	Male	Delhi	155	Graduate	No
4	43	Female	Mumbai	157	Post-graduate	Yes

What is the average age and height of postgraduates who smoke?

In [84]:
```
abc=data1[(data1['Smoker']=='Yes') & (data1['Qualification']=='Post-graduate'
)]
abc
```

Out[84]:

	Age	Gender	Hometown	Height	Qualification	Smoker
4	43	Female	Mumbai	157	Post-graduate	Yes
9	49	Female	Bangalore	162	Post-graduate	Yes
19	53	Male	Bangalore	156	Post-graduate	Yes

In [86]:
```
pd.pivot_table(abc,index=['Gender'],values=['Age','Height'])
```

Out[86]:

	Age	Height
Gender		
Female	46	159.5
Male	53	156.0

different types of files to read the data

In [ ]:
```
#csv format , xlsx file , txt(text) file , json file , xml file , html
```

In [ ]:
```

```