import data from downloads (or can say where file/csv/excel and so on is saved)

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv(r'C:\Users\USER\Downloads\Planets.xls')
```

In [3]:
```
print(df)
```

```
    Planet       Mass Diameter  DayLength  SunDistance OrbitPeriod  \
0  MERCURY     0.3300     4879     4222.6         57.9          88
1    VENUS     4.8700   12,104     2802.0        108.2       224.7
2    EARTH     5.9700   12,756       24.0        149.6       365.2
3     MOON     0.0730     3475      708.7          NaN        27.3
4     MARS     0.6420     6792       24.7        227.9         687
5  JUPITER  1898.0000  142,984        9.9        778.6        4331
6   SATURN   568.0000  120,536       10.7       1433.5      10,747
7   URANUS    86.8000   51,118       17.2       2872.5      30,589
8  NEPTUNE   102.0000   49,528       16.1       4495.1      59,800
9    PLUTO     0.0146     2370      153.3       5906.4      90,560

   OrbitVelocity  MeanTemperature  SurfacePressure  Moons Rings MagneticField  \
0           47.4              167          0.00000      0    No           Yes
1           35.0              464         92.00000      0    No            No
2           29.8               15          1.00000      1    No           Yes
3            1.0              -20          0.00000      0    No            No
4           24.1              -65          0.01000      2    No            No
5           13.1             -110              NaN     67   Yes           Yes
6            9.7             -140              NaN     62   Yes           Yes
7            6.8             -195              NaN     27   Yes           Yes
8            5.4             -200              NaN     14   Yes           Yes
9            4.7             -225          0.00001      5    No           NaN

  FirstVisited  FirstMission
0   1974-03-29    Mariner 10
1   1962-08-27     Mariner 2
2          NaN           NaN
3   1959-09-12        Luna 2
4   1965-07-15     Mariner 4
5   1973-12-04    Pioneer 10
6   1979-09-01    Pioneer 11
7   1986-01-24     Voyager 2
8   1989-08-25     Voyager 2
9   2015-07-14  New Horizons
```

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

Out[4]:

	Planet	Mass	Diameter	DayLength	SunDistance	OrbitPeriod	OrbitVelocity	MeanTemperature	SurfacePressure	Moons	Rings	Magn
0	MERCURY	0.330	4879	4222.6	57.9	88	47.4	167	0.00	0	No	
1	VENUS	4.870	12,104	2802.0	108.2	224.7	35.0	464	92.00	0	No	
2	EARTH	5.970	12,756	24.0	149.6	365.2	29.8	15	1.00	1	No	
3	MOON	0.073	3475	708.7	NaN	27.3	1.0	-20	0.00	0	No	
4	MARS	0.642	6792	24.7	227.9	687	24.1	-65	0.01	2	No	

Suppose if we need planet, Mass, Diameter column only

In [5]:
```
df[['Planet','Mass','Diameter']].head()
```

Out[5]:

	Planet	Mass	Diameter
0	MERCURY	0.330	4879
1	VENUS	4.870	12,104
2	EARTH	5.970	12,756
3	MOON	0.073	3475
4	MARS	0.642	6792

We can use directly . notation or index too - loc

In [6]:
```
df.Planet      #dot notation
```

Out[6]:
```
0    MERCURY
1      VENUS
2      EARTH
3       MOON
4       MARS
5    JUPITER
6     SATURN
7     URANUS
8    NEPTUNE
9      PLUTO
Name: Planet, dtype: object
```

df.loc[row_labels, column_labels] - Select specific rows and columns

In [7]:
```
#loc is for index/positional selection
df.loc[0:3]  #zero rows to third column
```

Out[7]:

	Planet	Mass	Diameter	DayLength	SunDistance	OrbitPeriod	OrbitVelocity	MeanTemperature	SurfacePressure	Moons	Rings	Magn
0	MERCURY	0.330	4879	4222.6	57.9	88	47.4	167	0.0	0	No	
1	VENUS	4.870	12,104	2802.0	108.2	224.7	35.0	464	92.0	0	No	
2	EARTH	5.970	12,756	24.0	149.6	365.2	29.8	15	1.0	1	No	
3	MOON	0.073	3475	708.7	NaN	27.3	1.0	-20	0.0	0	No	

In [8]:
```
#suppose we need 3 column
df.iloc[:,0:3]
```

Out[8]:

	Planet	Mass	Diameter
0	MERCURY	0.3300	4879
1	VENUS	4.8700	12,104
2	EARTH	5.9700	12,756
3	MOON	0.0730	3475
4	MARS	0.6420	6792
5	JUPITER	1898.0000	142,984
6	SATURN	568.0000	120,536
7	URANUS	86.8000	51,118
8	NEPTUNE	102.0000	49,528
9	PLUTO	0.0146	2370

Information about data

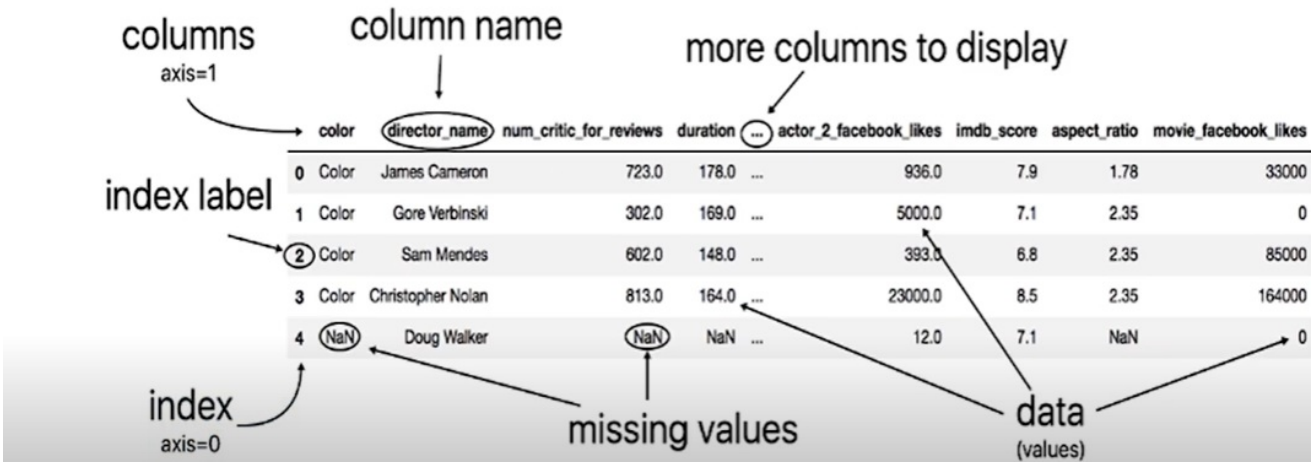
In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Planet           10 non-null     object
 1   Mass             10 non-null     float64
 2   Diameter         10 non-null     object
 3   DayLength        10 non-null     float64
 4   SunDistance      9 non-null      float64
 5   OrbitPeriod      10 non-null     object
 6   OrbitVelocity    10 non-null     float64
 7   MeanTemperature  10 non-null     int64
 8   SurfacePressure  6 non-null      float64
 9   Moons            10 non-null     int64
 10  Rings            10 non-null     object
 11  MagneticField    9 non-null      object
 12  FirstVisited     9 non-null      object
 13  FirstMission     9 non-null      object
dtypes: float64(5), int64(2), object(7)
memory usage: 1.2+ KB
```

NOTE: loc mainly used for string and iloc is for integer.

DataFrame

DataFrame



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Similarly, there is series which is similar to dataframe but dataframe is multi dimensional, series is single.

In [11]:
```
import pandas as pd
import numpy as np
s=pd.Series([1,2,3,4,np.NaN,4.5])
```

In [12]:
```
print(s)
```

```
0    1.0
1    2.0
2    3.0
3    4.0
4    NaN
5    4.5
dtype: float64
```

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

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

Now, lets create series of dates and lets use that dates to create dataframe from random.

In [15]:
```
date=pd.date_range('20231125', periods=6)
print(date)
```

```
DatetimeIndex(['2023-11-25', '2023-11-26', '2023-11-27', '2023-11-28',
               '2023-11-29', '2023-11-30'],
              dtype='datetime64[ns]', freq='D')
```

In [16]:
```
type(date)
```

Out[16]:
```
pandas.core.indexes.datetimes.DatetimeIndex
```

In [22]:
```
df=pd.DataFrame(np.random.randn(6,4),index=date,columns=('A','B','C','D'))
```

In [23]:
```
print(df)
```

```
                   A         B         C         D
2023-11-25  0.339085  0.438277 -0.851866  1.642472
2023-11-26 -0.569652  0.260210  0.419846 -0.220805
2023-11-27 -0.769831 -1.575531  1.312005  1.304006
2023-11-28 -1.060944 -1.919002 -0.711696  0.756773
2023-11-29 -1.038965 -1.048602 -0.785355 -0.115891
2023-11-30  1.944065  0.516361 -0.921907  0.336267
```

In [24]:
```
#Note: In above we have use random but have not imported random because it is NumPy random.
```

In [27]:
```
df[0:2]        #showing two rows
```

Out[27]:

	A	B	C	D
2023-11-25	0.339085	0.438277	-0.851866	1.642472
2023-11-26	-0.569652	0.260210	0.419846	-0.220805

In [33]:
```
#Lets create series and some list

labels=["A","B","C","D"]
l1=[1,2,3,4]
l2=np.array(l1)
```

In [34]:
```
#What is difference here in l1 and l2, as l2 is array, we can do many mathematical like:
l1.mean()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[34], line 2
      1 #What is difference here in l1 and l2, as l2 is array, we can do many mathematical like:
----> 2 l1.mean()

AttributeError: 'list' object has no attribute 'mean'
```

In [35]:
```
#we get error but l2
l2.mean()
```

Out[35]:
```
2.5
```

In [36]:
```
#This is power of numpy array, but here we are to create series with array and list - labels
```

In [37]:
```
df1=pd.Series(l2, index=labels)
```

In [38]:
```
print(df1)
```

```
A    1
B    2
C    3
D    4
dtype: int32
```

In [43]:
```
#Adding two series
a=pd.Series([1,2,3,4],index=("cricket","football","basketball","golf"))
```

In [44]:
```
b=pd.Series([5,6,7,8],index=("cricket","football","basetball","golf"))
```

In [45]:
```
print(a+b)
```

```
basetball      NaN
basketball     NaN
cricket        6.0
football       8.0
golf          12.0
dtype: float64
```

In [46]:
```
#Why we are getting NaN for two values - Because index does not matched.
```

Create DataFrame

In [47]:
```
l1=[1,2,3,4]
l2=[4,5,6,7]
l3=[8,9,10,11]
```

In [49]:
```
df=pd.DataFrame((l1,l2,l3),index='A B C'.split(),columns='F G H I'.split())
```

In [50]:
```
print(df)
```

```
   F  G   H   I
A  1  2   3   4
B  4  5   6   7
C  8  9  10  11
```

In [51]:
```
type(df)
```

Out[51]:
```
pandas.core.frame.DataFrame
```

In [52]:
```
#Here above in index and columns, instead of creating list, I just insert the name and use split
```

Now before moving forward lets see one concept random.seeds

In [55]:
```
import random
a=random.randint(1,20)
print(a)
```

```
19
```

In [56]:
```
#But how many times we run above code, random value gets changed, suppose we want to lock random value
random.seed(1)
a=random.randint(1,20)
print(a)
```

```
5
```

In [57]:
```
#Now, see running above code again. Our first random value is changed but not second one with seed.
```

In [1]:
```
import pandas as pd

# initialize data of lists.
data = {'Name': ['Tom', 'nick', 'krish', 'jack'],
```

```
        'Age': [20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
df
```

Out[1]:

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

In [2]:
```
type(data)
```

Out[2]:
```
dict
```

In [3]:
```
#yes we first created dictionary and converted it into pandas dataframe, so if we want different index?
df = pd.DataFrame(data, index='A B C D'.split())
```

In [4]:
```
print(df)
```

```
    Name  Age
A    Tom   20
B   nick   21
C  krish   19
D   jack   18
```

In [5]:
```
#if we want age
df['Age']
```

Out[5]:
```
A    20
B    21
C    19
D    18
Name: Age, dtype: int64
```

In [6]:
```
#Adding new column in dataframe
df['Income']=[2000,3000,2200,6000]
```

In [7]:
```
print(df)
```

```
    Name  Age  Income
A    Tom   20    2000
B   nick   21    3000
C  krish   19    2200
D   jack   18    6000
```

In [14]:
```
#suppose we want output Income and age, Income first and age
df[['Income','Age']]
```

Out[14]:

	Income	Age
A	2000	20
B	3000	21
C	2200	19
D	6000	18

In [25]:
```
#Removing Column from dataframe
df.drop('Income',axis=1)      #axis 1 means column
```

Out[25]:

	Name	Age	Name_Income
A	Tom	20	NameIncome
B	nick	21	NameIncome
C	krish	19	NameIncome
D	jack	18	NameIncome

In [26]:
```
#In previous I mistakely created Name_Income column, lets drop that too
df.drop('Name_Income',axis=1)
```

Out[26]:

	Name	Age	Income
A	Tom	20	2000
B	nick	21	3000
C	krish	19	2200
D	jack	18	6000

In [28]:
```
#But, it does not remove see, income again comes back so we have to use inplace=True for permanent setup
df.drop('Name_Income',axis=1,inplace=True)
df.drop('Income',axis=1,inplace=True)
```

In [29]:
```
df
```

Out[29]:

	Name	Age
A	Tom	20
B	nick	21
C	krish	19
D	jack	18

In [31]:
```
#if we want to drop row
df.drop('C',axis=0)
```

Out[31]:

	Name	Age
A	Tom	20
B	nick	21
D	jack	18

Selecting rows and column

In [32]:
```
#selecting column
df['Name']
```

Out[32]:
```
A      Tom
B     nick
C    krish
D     jack
Name: Name, dtype: object
```

In [33]:
```
#next way
df.Age
```

Out[33]:
```
A    20
B    21
C    19
D    18
Name: Age, dtype: int64
```

In [34]:
```
#But we cannot get row in that way, for row we have to use loc or iloc function
```

In [36]:
```
#loc= location
df.loc['B']
```

Out[36]:
```
Name    nick
Age       21
Name: B, dtype: object
```

In [37]:
```
#iloc=index location
df.iloc[2]
```

Out[37]:
```
Name    krish
Age        19
Name: C, dtype: object
```

Conditional statement

In [38]:
```
#lets see df first
df
```

Out[38]:

	Name	Age
A	Tom	20
B	nick	21
C	krish	19
D	jack	18

In [40]:
```
#need all value greater than 19
df.Age>19
```

Out[40]:
```
A     True
B     True
C    False
D    False
Name: Age, dtype: bool
```

In [43]:
```
df[df['Age']>19]
```

Out[43]:

	Name	Age
A	Tom	20
B	nick	21

In [45]:
```
#if we want to change index?
# have to add new column and need to set_index
df["Country"]=["Nepal","USA","London","AUS"]
```

In [47]:
```
print(df)
```

```
    Name  Age Country
A    Tom   20   Nepal
B   nick   21     USA
C  krish   19  London
D   jack   18     AUS
```

In [48]:
```
df.set_index('Country',inplace=True)
print(df)
```

```
          Name  Age
Country
Nepal      Tom   20
USA       nick   21
London   krish   19
AUS       jack   18
```

Handling Missing values - dropna, fillna

In [50]:
```
import numpy as np
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)
```

In [51]:
```
df
```

Out[51]:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

In [52]:
```
df.isnull()
```

Out[52]:

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

In [53]:
```
#dropping missing values with rows
df.dropna()
```

Out[53]:

	First Score	Second Score	Third Score
1	90.0	45.0	40.0

In [54]:
```
df
```

Out[54]:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

In [55]:
```
#filling
df.fillna(value=0)
```

Out[55]:

	First Score	Second Score	Third Score
0	100.0	30.0	0.0
1	90.0	45.0	40.0
2	0.0	56.0	80.0
3	95.0	0.0	98.0

But, mainly in real world, we keep mean, lets try that

In [56]:
```
df
```

Out[56]:

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

In [57]:
```
df['First Score'].fillna(value=df['First Score'].mean())     #mean =(100+90+95/3)
```

Out[57]:
```
0    100.0
1     90.0
2     95.0
3     95.0
Name: First Score, dtype: float64
```

Aggregation Operation

In [58]:
```
df=pd.read_csv(r'C:\Users\USER\Downloads\nba.csv',encoding='latin1')
```

In [59]:
```
df.head()
```

Out[59]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

In [62]:
```
team_groupby=df.groupby('Team')  #for now lets see aggregation only, later in practice we will use above datset
```

In [65]:
```
team_groupby.first()
```

Out[65]:

Team	Name	Number	Position	Age	Height	Weight	College	Salary
Atlanta Hawks	Kent Bazemore	24.0	SF	26.0	6-5	201.0	Old Dominion	2000000.0
Boston Celtics	Avery Bradley	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Brooklyn Nets	Bojan Bogdanovic	44.0	SG	27.0	6-8	216.0	Oklahoma State	3425510.0
Charlotte Hornets	Nicolas Batum	5.0	SG	27.0	6-8	200.0	Virginia Commonwealth	13125306.0
Chicago Bulls	Cameron Bairstow	41.0	PF	25.0	6-9	250.0	New Mexico	845059.0
Cleveland Cavaliers	Matthew Dellavedova	8.0	PG	25.0	6-4	198.0	Saint Mary's	1147276.0
Dallas Mavericks	Justin Anderson	1.0	SG	22.0	6-6	228.0	Virginia	1449000.0
Denver Nuggets	Darrell Arthur	0.0	PF	28.0	6-9	235.0	Kansas	2814000.0
Detroit Pistons	Joel Anthony	50.0	C	33.0	6-9	245.0	UNLV	2500000.0
Golden State Warriors	Leandro Barbosa	19.0	SG	33.0	6-3	194.0	North Carolina	2500000.0
Houston Rockets	Trevor Ariza	1.0	SF	30.0	6-8	215.0	UCLA	8193030.0
Indiana Pacers	Lavoy Allen	5.0	PF	27.0	6-9	255.0	Temple	4050000.0
Los Angeles Clippers	Cole Aldrich	45.0	C	27.0	6-11	250.0	Kansas	1100602.0
Los Angeles Lakers	Brandon Bass	2.0	PF	31.0	6-8	250.0	LSU	3000000.0
Memphis Grizzlies	Jordan Adams	3.0	SG	21.0	6-5	209.0	UCLA	1404600.0
Miami Heat	Chris Bosh	1.0	PF	32.0	6-11	235.0	Georgia Tech	22192730.0
Milwaukee Bucks	Giannis Antetokounmpo	34.0	SF	21.0	6-11	222.0	Arizona	1953960.0
Minnesota Timberwolves	Nemanja Bjelica	88.0	PF	28.0	6-10	240.0	Louisville	3950001.0
New Orleans Pelicans	Alexis Ajinca	42.0	C	28.0	7-2	248.0	California	4389607.0
New York Knicks	Arron Afflalo	4.0	SG	30.0	6-5	210.0	UCLA	8000000.0
Oklahoma City Thunder	Steven Adams	12.0	C	22.0	7-0	255.0	Pittsburgh	2279040.0
Orlando Magic	Dewayne Dedmon	3.0	C	26.0	7-0	245.0	USC	947276.0
Philadelphia 76ers	Elton Brand	42.0	PF	37.0	6-9	254.0	Duke	947276.0
Phoenix Suns	Eric Bledsoe	2.0	PG	26.0	6-1	190.0	Kentucky	13500000.0
Portland Trail Blazers	Cliff Alexander	34.0	PF	20.0	6-8	240.0	Kansas	525093.0
Sacramento Kings	Quincy Acy	13.0	SF	25.0	6-7	240.0	Baylor	981348.0
San Antonio Spurs	LaMarcus Aldridge	12.0	PF	30.0	6-11	240.0	Texas	19689000.0
Toronto Raptors	Bismack Biyombo	8.0	C	23.0	6-9	245.0	Missouri	2814000.0
Utah Jazz	Trevor Booker	33.0	PF	28.0	6-8	228.0	Clemson	4775000.0
Washington Wizards	Alan Anderson	6.0	SG	33.0	6-6	220.0	Michigan State	4000000.0

In [66]:
```
team_groupby.get_group('Boston Celtics')
```

Out[66]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0

In [67]:
```
team_groupby.describe()
```

Out[67]:

	Number								Age		...	Weight				
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	
Team																
Atlanta Hawks	15.0	19.000000	11.476684	0.0	11.50	17.0	25.50	43.0	15.0	28.200000	...	242.50	260.0	15.0	4.860197e+06	5.194508
Boston Celtics	15.0	31.866667	30.300558	0.0	9.50	28.0	42.50	99.0	15.0	24.733333	...	236.50	260.0	14.0	4.181505e+06	3.146033
Brooklyn Nets	15.0	18.266667	14.104035	0.0	8.00	15.0	27.00	44.0	15.0	25.600000	...	220.50	275.0	15.0	3.501898e+06	5.317817
Charlotte Hornets	15.0	17.133333	16.672761	0.0	4.00	12.0	27.50	50.0	15.0	26.133333	...	240.00	289.0	15.0	5.222728e+06	4.538601
Chicago Bulls	15.0	19.200000	17.193022	0.0	5.50	16.0	28.00	55.0	15.0	27.400000	...	231.00	275.0	15.0	5.785559e+06	6.251088
Cleveland Cavaliers	15.0	14.466667	13.809245	0.0	4.50	12.0	21.50	52.0	15.0	29.533333	...	250.50	275.0	14.0	7.642049e+06	7.730329
Dallas Mavericks	15.0	20.000000	16.252472	1.0	6.00	21.0	30.50	50.0	15.0	29.733333	...	245.00	275.0	15.0	4.746582e+06	5.030279
Denver Nuggets	15.0	15.266667	19.655849	0.0	4.00	9.0	18.00	77.0	15.0	25.733333	...	226.50	280.0	14.0	4.294424e+06	4.320214
Detroit Pistons	15.0	17.266667	15.303906	0.0	5.50	13.0	23.50	50.0	15.0	26.200000	...	242.50	279.0	15.0	4.477884e+06	4.668478
Golden State Warriors	15.0	20.866667	11.413442	4.0	11.50	20.0	30.50	40.0	15.0	27.666667	...	247.50	273.0	15.0	5.924600e+06	5.664282
Houston Rockets	15.0	14.666667	12.505237	0.0	5.50	12.0	25.50	35.0	15.0	26.866667	...	237.50	265.0	15.0	5.018868e+06	6.414749
Indiana Pacers	15.0	18.933333	15.988686	0.0	4.00	13.0	30.50	44.0	15.0	26.400000	...	246.50	255.0	15.0	4.450122e+06	4.584514
Los Angeles Clippers	15.0	19.533333	13.125040	3.0	8.50	19.0	31.00	45.0	15.0	29.466667	...	242.50	265.0	15.0	6.323643e+06	7.600225
Los Angeles Lakers	15.0	16.066667	15.285225	0.0	3.50	9.0	26.00	50.0	15.0	27.533333	...	250.00	270.0	15.0	4.784695e+06	6.835688
Memphis Grizzlies	18.0	15.555556	14.030313	0.0	5.50	10.5	21.25	50.0	18.0	28.388889	...	236.75	270.0	14.0	5.467920e+06	5.201676
Miami Heat	15.0	10.466667	10.377632	0.0	3.50	8.0	13.00	40.0	15.0	28.933333	...	237.50	265.0	13.0	6.347359e+06	7.848628
Milwaukee Bucks	16.0	20.000000	17.485232	3.0	10.50	17.5	21.25	77.0	16.0	24.562500	...	246.75	265.0	16.0	4.350220e+06	4.875071
Minnesota Timberwolves	14.0	19.571429	21.964007	1.0	8.25	13.0	21.75	88.0	14.0	26.357143	...	240.75	307.0	13.0	4.593054e+06	4.139625
New Orleans Pelicans	19.0	17.000000	14.011900	0.0	4.50	15.0	27.50	44.0	19.0	26.894737	...	235.00	270.0	19.0	4.355304e+06	4.537874
New York Knicks	16.0	13.250000	12.964053	1.0	4.75	8.5	17.25	43.0	16.0	27.000000	...	240.00	278.0	16.0	4.581494e+06	5.952487
Oklahoma City Thunder	15.0	14.000000	12.130246	0.0	4.50	11.0	21.50	35.0	15.0	27.066667	...	247.50	255.0	15.0	6.251020e+06	6.632400
Orlando Magic	14.0	16.428571	16.411601	0.0	5.50	10.5	20.75	55.0	14.0	25.071429	...	238.75	260.0	14.0	4.297248e+06	3.068412
Philadelphia 76ers	15.0	18.066667	14.660280	0.0	6.00	12.0	32.00	42.0	15.0	24.600000	...	246.50	275.0	14.0	2.213778e+06	1.900402
Phoenix Suns	15.0	15.466667	10.405127	1.0	7.00	15.0	22.00	35.0	15.0	25.866667	...	241.00	260.0	15.0	4.229676e+06	5.022561
Portland Trail Blazers	15.0	16.000000	13.711309	0.0	4.50	11.0	23.50	44.0	15.0	25.066667	...	240.00	265.0	15.0	3.220121e+06	2.392741
Sacramento Kings	15.0	16.933333	12.002777	0.0	7.50	15.0	25.50	41.0	15.0	26.800000	...	239.00	270.0	15.0	4.778911e+06	4.701792
San Antonio Spurs	15.0	17.933333	11.067757	1.0	10.50	17.0	23.50	40.0	15.0	31.600000	...	245.00	290.0	15.0	5.629516e+06	6.396804
Toronto Raptors	15.0	22.466667	25.856380	1.0	5.50	10.0	27.50	92.0	15.0	26.133333	...	242.50	255.0	15.0	4.741174e+06	4.195943
Utah Jazz	15.0	17.866667	11.432202	2.0	9.00	20.0	24.50	41.0	15.0	24.466667	...	232.50	265.0	15.0	4.204006e+06	4.467878
Washington Wizards	15.0	17.600000	22.610996	1.0	5.50	12.0	19.00	90.0	15.0	27.866667	...	241.00	250.0	15.0	5.088576e+06	4.869388

30 rows × 32 columns

Concatenation and Merging

In [69]:
```
#lets create two new dataframe from dictionary
dist1={"custid":[1,2,3],"q1":[2000,3000,1000],"q2":[2500,1500,500]}
```

In [70]:
```
dist2={"custid":[4,5,6],"q3":[9000,8000,100],"q4":[250,150,5000]}
```

In [71]:
```
df1=pd.DataFrame(dist1)
df2=pd.DataFrame(dist2)
```

In [72]:
```
df1
```

Out[72]:

	custid	q1	q2
0	1	2000	2500
1	2	3000	1500
2	3	1000	500

In [73]:
```
df2
```

Out[73]:

	custid	q3	q4
0	4	9000	250
1	5	8000	150
2	6	100	5000

In [75]:
```
#lets use concatenation
pd.concat([df1,df2])
```

Out[75]:

	custid	q1	q2	q3	q4
0	1	2000.0	2500.0	NaN	NaN
1	2	3000.0	1500.0	NaN	NaN
2	3	1000.0	500.0	NaN	NaN
0	4	NaN	NaN	9000.0	250.0
1	5	NaN	NaN	8000.0	150.0
2	6	NaN	NaN	100.0	5000.0

In [76]:
```
#it defaultly join horizantally, if want to join according to column?
pd.concat([df1,df2], axis=1)
```

Out[76]:

	custid	q1	q2	custid	q3	q4
0	1	2000	2500	4	9000	250
1	2	3000	1500	5	8000	150
2	3	1000	500	6	100	5000

In [77]:
```
#Its not a good way, lets merge - outer & inner
```

In [79]:
```
pd.merge(df1,df2,how='outer',on='custid')
```

Out[79]:

	custid	q1	q2	q3	q4
0	1	2000.0	2500.0	NaN	NaN
1	2	3000.0	1500.0	NaN	NaN
2	3	1000.0	500.0	NaN	NaN
3	4	NaN	NaN	9000.0	250.0
4	5	NaN	NaN	8000.0	150.0
5	6	NaN	NaN	100.0	5000.0

In [80]:
```
pd.merge(df1,df2,how='inner',on='custid')
```

Out[80]:

	custid	q1	q2	q3	q4

In [81]:
```
#in inner we did not get any output because there is no matching id. inner will give only those record with mat
```

PRACTICE

Which function is used to assign names to the index in a Pandas Series?

In [93]:
```
import pandas as pd

s = pd.Series([5, 10, 15], index=['a', 'b', 'c'])
print(s)
```

```
s.index.name = 'Letters'
print(s)
```

```
a     5
b    10
c    15
dtype: int64
Letters
a     5
b    10
c    15
dtype: int64
```

How do you print common elements that are present in 2 series?

first_series[first_series.isin(second_series)]

Which method is used to extract items from index locations using a series?

Answer choices Select only one option take()

extract()

position()

All of the options

In [94]:
```
import pandas as pd

s = pd.Series([1, 2, 3, 4, 5])

s.take([0, 2, 4])
```

Out[94]:
```
0    1
2    3
4    5
dtype: int64
```

Is it possible to create a series using dictionaries in Python?

In [95]:
```
dict1={"ram":1,"shyam":2}
```

In [96]:
```
series1=pd.Series(dict1)
```

In [97]:
```
print(series1)
```

```
ram      1
shyam    2
dtype: int64
```

In [98]:
```
type(dict1)
```

Out[98]:
```
dict
```

In [99]:
```
#yes it is possible.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js