Pandas

Pandas is a Python library.

Pandas is used to analyze data.

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

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

In [1]:
```
pip install pandas
```

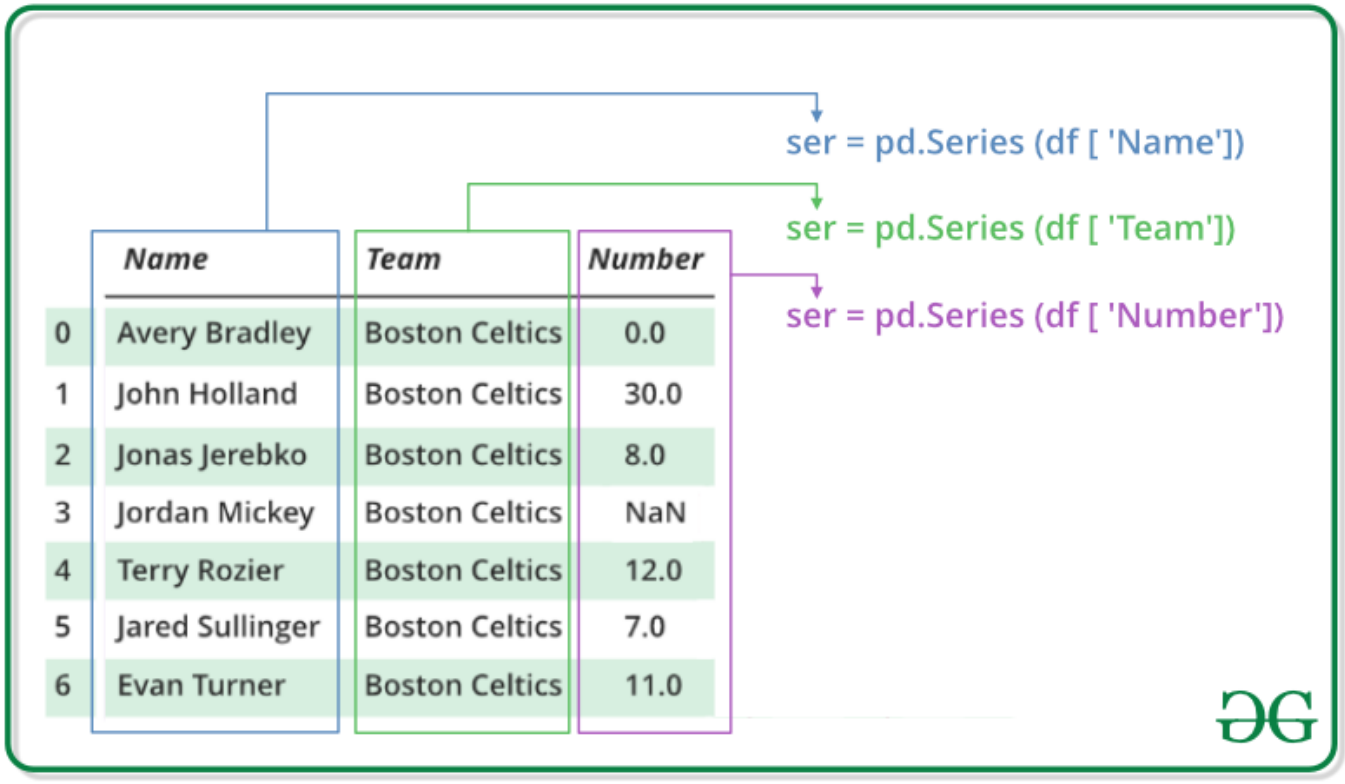
```
Requirement already satisfied: pandas in d:\archana\lib\site-packages (1.3.4)Note: you may
need to restart the kernel to use updated packages.
Requirement already satisfied: python-dateutil>=2.7.3 in d:\archana\lib\site-packages (fro
m pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in d:\archana\lib\site-packages (from pandas)
(1.20.3)
Requirement already satisfied: pytz>=2017.3 in d:\archana\lib\site-packages (from pandas)
(2021.3)
Requirement already satisfied: six>=1.5 in d:\archana\lib\site-packages (from python-dateu
til>=2.7.3->pandas) (1.16.0)
```

In [9]:
```
import pandas as pd
import numpy as np
```

Pandas Series

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.



In [4]:
```
# creating series
s1=pd.Series([10,12,14,16,18])
s1
```

Out[4]:
```
0    10
1    12
2    14
3    16
4    18
dtype: int64
```

In [4]:
```
# creating series with index values
s2=pd.Series([10,12,14,16,18],index=['a','b','c','d','e'])
s2
```

Out[4]:
```
a    10
b    12
c    14
d    16
e    18
dtype: int64
```

In [7]:
```
# with index we can name own labels
s3['b']
```

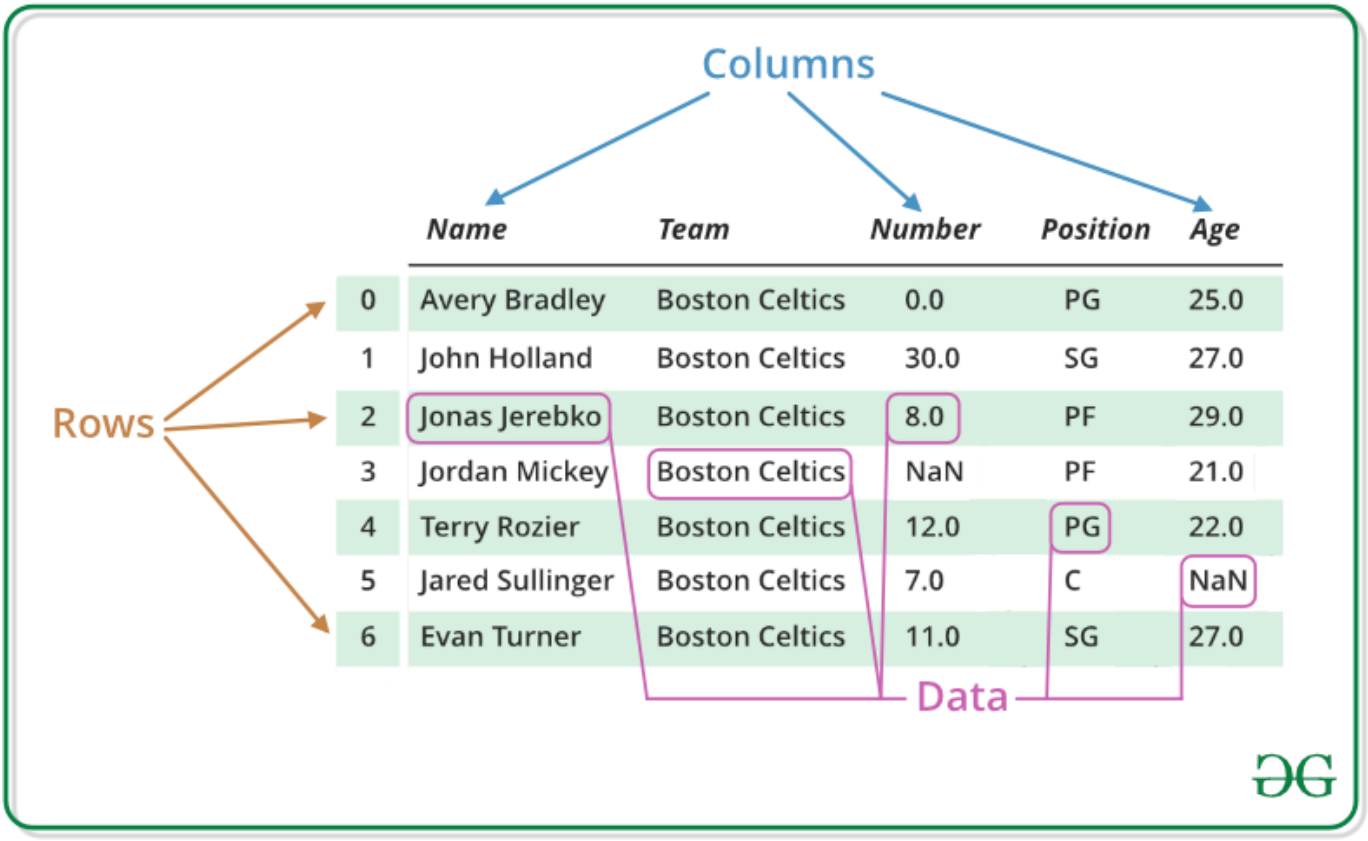
Out[7]:
```
12.0
```

In [10]:
```
#create series using dictionary
s4=pd.Series({'e':65,'f':45,'c':43})
s4
```

Out[10]:
```
e    65
f    45
c    43
dtype: int64
```

DataFrame

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.



In [11]:
```
d1=pd.DataFrame([43,54,65,76])
d1
```

Out[11]:

	0
0	43
1	54
2	65
3	76

In [12]:
```
d2=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]])
d2
```

Out[12]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

In [5]:
```
#converting series to dataframe
d2=pd.DataFrame(s2)
d2
```

Out[5]:

	0
a	10
b	12
c	14
d	16
e	18

In [7]:
```
d3=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],columns=['a','b','c'])
d3
```

Out[7]:

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

In [10]:
```
#creating dataframe with column names and labels
d3=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],columns=['a','b','c'],index=['x','y','z'])
d3
```

Out[10]:

	a	b	c
x	1	2	3
y	4	5	6
z	7	8	9

In [13]:
```
#creating dataframe from list of dictionaries
dic=[{'ram':1,'sam':2},{'john':5,'kim':10,'sim':20}]
pd.DataFrame(dic,index=['a','b'])
```

Out[13]:

	ram	sam	john	kim	sim
a	1.0	2.0	NaN	NaN	NaN
b	NaN	NaN	5.0	10.0	20.0

DataFrame operations

In [36]:
```
d4=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],columns=['a','b','c'],index=['x','y','z'])
d4
```

Out[36]:

	a	b	c
x	1	2	3
y	4	5	6
z	7	8	9

In [37]:
```
d4['a']
```

Out[37]:
```
x    1
y    4
z    7
Name: a, dtype: int64
```

In [39]:
```
d4['d']=d4['a']*d4['b']
d4['e']=d4['a']+d4['b']
d4
```

Out[39]:

	a	b	c	d	e
x	1	2	3	2	3
y	4	5	6	20	9
z	7	8	9	56	15

In [50]:
```
d4.insert(1,'new1',d4['c'])
d4
```

Out[50]:

	a	new1	c	d	e
x	1	3	3	2	3
y	4	6	6	20	9
z	7	9	9	56	15

In [53]:
```
import numpy as np
d5=pd.DataFrame({'abc':np.random.randint(2,6,size=(10)),'bcd':np.random.randint(4,10,size=
d5
```

Out[53]:

	abc	bcd	cde
0	3	4	5
1	3	8	6
2	5	4	4
3	2	7	6
4	2	5	6
5	2	9	5
6	3	8	6
7	2	9	3
8	3	5	3
9	4	8	5

In [54]:
```
d5.head()#returns top 5 rows of data
```

Out[54]:

	abc	bcd	cde
0	3	4	5

	abc	bcd	cde
1	3	8	6
2	5	4	4
3	2	7	6
4	2	5	6

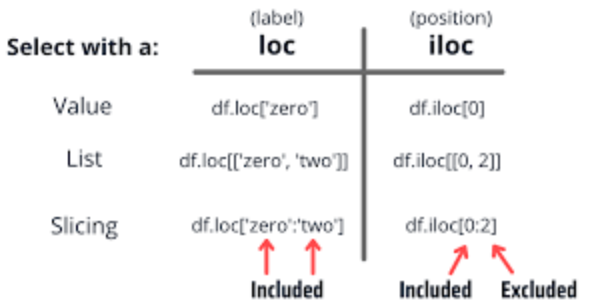
In [55]:
```
d5.tail()# last 5 rows of data
```

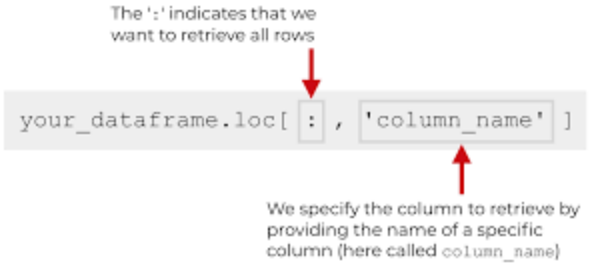
Out[55]:

	abc	bcd	cde
5	2	9	5
6	3	8	6
7	2	9	3
8	3	5	3
9	4	8	5

In [56]:
```
d5.info()# complete information of given data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   abc     10 non-null     int32
 1   bcd     10 non-null     int32
 2   cde     10 non-null     int32
dtypes: int32(3)
memory usage: 248.0 bytes
```





Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

In [57]:
```
d5
```

Out[57]:

	abc	bcd	cde
0	3	4	5
1	3	8	6
2	5	4	4
3	2	7	6
4	2	5	6
5	2	9	5
6	3	8	6
7	2	9	3
8	3	5	3
9	4	8	5

In [58]:
```
d5.loc[9,'cde']#loc[row name,col name]
```

Out[58]:
```
5
```

In [59]:
```
d5.loc[4:9,['abc','cde']]
```

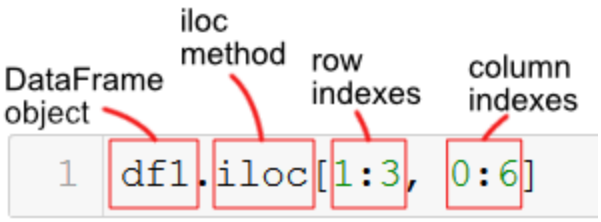
Out[59]:

	abc	cde
4	2	6
5	2	5
6	3	6
7	2	3
8	3	3
9	4	5

In [61]:
```
d5.loc[[3,4,7],['abc','cde']]
```

Out[61]:

	abc	cde
3	2	6
4	2	6
7	2	3



In [64]:
```
d5
```

Out[64]:

	abc	bcd	cde
0	3	4	5
1	3	8	6
2	5	4	4
3	2	7	6
4	2	5	6
5	2	9	5
6	3	8	6
7	2	9	3
8	3	5	3
9	4	8	5

In [65]:
```
d5.iloc[9,2]#[row index,column index]
```

Out[65]:
```
5
```

In [66]:
```
d5.iloc[2:7,[0,2]]
```

Out[66]:

	abc	cde
2	5	4
3	2	6
4	2	6
5	2	5
6	3	6

In [67]:
```
d5.abc
```

Out[67]:
```
0    3
1    3
2    5
3    2
4    2
5    2
6    3
7    2
8    3
```

```
9    4
Name: abc, dtype: int32
```

In [68]:
```
d5.abc.values #dataframe.column,values
```

Out[68]:
```
array([3, 3, 5, 2, 2, 2, 3, 2, 3, 4])
```

In [71]:
```
d5['sum']=d5.abc.values+d5.bcd.values+d5.cde.values
d5
```

Out[71]:

	abc	bcd	cde	sum
0	3	4	5	12
1	3	8	6	17
2	5	4	4	13
3	2	7	6	15
4	2	5	6	13
5	2	9	5	16
6	3	8	6	17
7	2	9	3	14
8	3	5	3	11
9	4	8	5	17

In [3]:
```
a=[['sam',425,20000],['ram',426,25000],['kim',427,30000]]
df1=pd.DataFrame(a,columns=['name','id','salary'])
df1
```

Out[3]:

	name	id	salary
0	sam	425	20000
1	ram	426	25000
2	kim	427	30000

In [5]:
```
y=df1[df1.salary>=20000]
print(y)
y[['id','salary']]
```

```
  name   id  salary
0  sam  425   20000
1  ram  426   25000
2  kim  427   30000
```

Out[5]:

	id	salary
0	425	20000
1	426	25000
2	427	30000

In [7]:
```
df1.append({'name':'john','id':428,'salary':35000},ignore_index=True)
```

Out[7]:

	name	id	salary
0	sam	425	20000
1	ram	426	25000
2	kim	427	30000
3	john	428	35000

In [10]:
```
df1=df1.append({'name':np.nan,'id':428,'salary':35000},ignore_index=True)
df1
```

Out[10]:

	name	id	salary
0	sam	425.0	20000.0
1	ram	426.0	25000.0
2	kim	427.0	30000.0
3	NaN	428.0	35000.0

In [11]:
```
df1.isnull()
```

Out[11]:

	name	id	salary
0	False	False	False
1	False	False	False
2	False	False	False
3	True	False	False

In [12]:
```
df1.isnull().sum()
```

Out[12]:
```
name      1
id        0
salary    0
dtype: int64
```

The dropna() method returns a new DataFrame, and will not change the original.

The result from the converting in the example above gave us a Nan value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

In [13]:
```
df1.dropna()
```

Out[13]:

	name	id	salary
0	sam	425.0	20000.0
1	ram	426.0	25000.0
2	kim	427.0	30000.0

In [14]:
```
df1
```

Out[14]:

	name	id	salary
0	sam	425.0	20000.0
1	ram	426.0	25000.0
2	kim	427.0	30000.0
3	NaN	428.0	35000.0

In [17]:
```
df1.fillna(value='abc')
```

Out[17]:

	name	id	salary
0	sam	425.0	20000.0
1	ram	426.0	25000.0
2	kim	427.0	30000.0
3	abc	428.0	35000.0

Groupby

. We can create a grouping of categories and apply a function to the categories. It's a simple concept but it's an extremely valuable technique that's widely used in data science

Groupby mainly refers to a process involving one or more of the following steps they are:

Splitting : It is a process in which we split data into group by applying some conditions on datasets.

Applying : It is a process in which we apply a function to each group independently

Combining : It is a process in which we combine different datasets after applying groupby and results into a data structure

In [18]:
```
df=pd.DataFrame({'animal':['Falcon','Falcon','parrot','parrot'],'Maxspeed':[380.,370.,24.,
df
```

Out[18]:

	animal	Maxspeed
0	Falcon	380.0
1	Falcon	370.0
2	parrot	24.0
3	parrot	26.0

In [19]:
```
df.groupby(['animal']).mean()
```

Out[19]:

	Maxspeed
animal	
Falcon	375.0
parrot	25.0

In [44]:

```
import pandas as pd
df_csv=pd.read_csv("C:\\Users\\Dell\\Downloads\\archive\\russia_losses_personnel.csv")
df_csv
```

Out[44]:

	date	day	personnel	personnel*	POW
0	2022-02-25	2	2800	about	0
1	2022-02-26	3	4300	about	0
2	2022-02-27	4	4500	about	0
3	2022-02-28	5	5300	about	0
4	2022-03-01	6	5710	about	200
5	2022-03-02	7	5840	about	200
6	2022-03-03	8	9000	about	200
7	2022-03-04	9	9166	about	200
8	2022-03-05	10	10000	about	216
9	2022-03-06	11	11000	about	232
10	2022-03-07	12	11000	more	259
11	2022-03-08	13	12000	about	284
12	2022-03-09	14	12000	about	360
13	2022-03-10	15	12000	more	371
14	2022-03-11	16	12000	more	389
15	2022-03-12	17	12000	more	389
16	2022-03-13	18	12000	more	389
17	2022-03-14	19	12000	more	389
18	2022-03-15	20	13500	about	389
19	2022-03-16	21	13800	about	389
20	2022-03-17	22	14000	about	405
21	2022-03-18	23	14200	about	405
22	2022-03-19	24	14400	about	405
23	2022-03-20	25	14700	about	405
24	2022-03-21	26	15000	about	405
25	2022-03-22	27	15300	about	411
26	2022-03-23	28	15600	about	412
27	2022-03-24	29	15800	about	412
28	2022-03-25	30	16100	about	412
29	2022-03-26	31	16400	about	412
30	2022-03-27	32	16600	about	421
31	2022-03-28	33	17000	about	421
32	2022-03-29	34	17200	about	430

	date	day	personnel	personnel*	POW
33	2022-03-30	35	17300	about	430
34	2022-03-31	36	17500	about	459
35	2022-04-01	37	17700	about	459
36	2022-04-02	38	17700	about	460
37	2022-04-03	39	18000	about	460
38	2022-04-04	40	18300	about	460
39	2022-04-05	41	18500	about	467
40	2022-04-06	42	18600	about	467
41	2022-04-07	43	18900	about	467
42	2022-04-08	44	19000	about	467
43	2022-04-09	45	19100	about	467
44	2022-04-10	46	19300	about	467
45	2022-04-11	47	19500	about	467
46	2022-04-12	48	19600	about	477
47	2022-04-13	49	19800	about	477
48	2022-04-14	50	19900	about	477
49	2022-04-15	51	20000	about	477
50	2022-04-16	52	20100	about	477
51	2022-04-17	53	20300	about	477
52	2022-04-18	54	20600	about	477
53	2022-04-19	55	20800	about	489
54	2022-04-20	56	20900	about	489

In [45]:
```
df_csv.columns
df_csv.isnull().sum()
```

Out[45]:
```
date          0
day           0
personnel     0
personnel*    0
POW           0
dtype: int64
```

In [ ]:
```
```