Pandas

- is used to make dataframe
- is open source and produces high performance and analysis
- it gives us a way to acomploish 5 types of data processing

  dataload

  prepare

  manipulate

  model

  analyze

In [2]:

```

import pandas as pd
pd.__version__
```

Out[2]:

```
'0.25.1'
```

it has two types of DATA STRUCTURES

- series
- dataframes

Series

```
"""
well for starters when using [] <-- these brackets ,
        rember the first one refers to the row
                                                   1 2 3
                                                -> 4 5 6
                                                   7 8 9



 and the second one refers to the columns
                                                    |
                                                    V
                                                  1 2 3
                                                  4 5 6
                                                  7 8 9

"""
```

In [ ]:

```

```

In [7]:

```
pd.Series([8,1,2,0])
```

Out[7]:

```
1    8
2    1
3    2
4    0
dtype: int64
```

In [9]:

```
data = pd.Series([8,1,2,0],index=["a","b","c","d"])
data
```

Out[9]:

```
a    8
b    1
c    2
d    0
dtype: int64
```

In [10]:

```
data[0]       ## we can use both the index number and the user-defined index name to re
```

Out[10]:

```
8
```

In [11]:

```
data["a"]
```

Out[11]:

```
8
```

In [12]:

```
print(data[0],data[1],data["c"],data["d"])
```

```
8 1 2 0
```

In [13]:

```
data["b": "c"]   ##user-defined index name can also be used as normal index reference i
```

Out[13]:

```
b    1
c    2
dtype: int64
```

In [17]:

```
data[1:4:2]
```

Out[17]:

```
b    1
d    0
dtype: int64
```

In [18]:

```
###  python dictionery can be converted to pandas series
d = {"eng":50,"comp":51,"maths":52}
d
```

Out[18]:

```
{'eng': 50, 'comp': 51, 'maths': 52}
```

In [19]:

```
pd.Series(d)
```

Out[19]:

```
eng      50
comp     51
maths    52
dtype: int64
```

In [20]:

```
########################################################################################
```

In [18]:

```
from numpy import *
### pandas has its it own dating and timeing system unlike the timemodule and thios can
date = pd.date_range("1-1-2000","31-12-2000")
print(date)
```

```
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08',
               '2000-01-09', '2000-01-10',
               ...
               '2000-12-22', '2000-12-23', '2000-12-24', '2000-12-25',
               '2000-12-26', '2000-12-27', '2000-12-28', '2000-12-29',
               '2000-12-30', '2000-12-31'],
              dtype='datetime64[ns]', length=366, freq='D')
```

In [39]:

```
q= pd.Series(arange(14,50,3),pd.date_range("12-12-12","29-12-12",periods=12))
q
```

Out[39]:

```
2012-12-12 00:00:00.000000000    14
2012-12-13 13:05:27.272727272    17
2012-12-15 02:10:54.545454545    20
2012-12-16 15:16:21.818181818    23
2012-12-18 04:21:49.090909090    26
2012-12-19 17:27:16.363636363    29
2012-12-21 06:32:43.636363636    32
2012-12-22 19:38:10.909090909    35
2012-12-24 08:43:38.181818181    38
2012-12-25 21:49:05.454545454    41
2012-12-27 10:54:32.727272727    44
2012-12-29 00:00:00.000000000    47
dtype: int32
```

In [43]:

```
q[0]
```

Out[43]:

```
14
```

In [46]:

```
pd.Series(arange(1,23))
```

Out[46]:

```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
10    11
11    12
12    13
13    14
14    15
15    16
16    17
17    18
18    19
19    20
20    21
21    22
dtype: int32
```

```
In [9]: k = pd.Series(range(3,10))
        k
```

```
Out[9]: 0    3
        1    4
        2    5
        3    6
        4    7
        5    8
        6    9
        dtype: int64
```

```
In [10]: list(k)
```

```
Out[10]: [3, 4, 5, 6, 7, 8, 9]
```

```
In [11]: dict(k)
```

```
Out[11]: {0: 3, 1: 4, 2: 5, 3: 6, 4: 7, 5: 8, 6: 9}
```

```
In [12]: a = "nana sen nana hyaku nana jun nana chou, nana sen nana hyaku nana jun nana oku, nan
```

```
In [13]: ####################################################  DATAFRAME  ######################
```

```
In [16]: ## example

         studmark = {
                     "names":["a","b","c","d"],
                     "math":[12,13,14,15],
                     "eng":[11,11,11,11],
                     "science":[13,13,12,14]
                     }
         studmark
```

```
Out[16]: {'names': ['a', 'b', 'c', 'd'],
          'math': [12, 13, 14, 15],
          'eng': [11, 11, 11, 11],
          'science': [13, 13, 12, 14]}
```

```
In [17]: pd.DataFrame(studmark)
```

Out[17]:

	names	math	eng	science
0	a	12	11	13
1	b	13	11	13
2	c	14	11	12
3	d	15	11	14

```
In [27]: data1 = array([["a",12,13,14,15],["b",10,20,30,40]])
         data1
```

```
Out[27]: array([['a', '12', '13', '14', '15'],
                ['b', '10', '20', '30', '40']], dtype='<U2')
```

```
In [22]: pd.DataFrame(data1)
```

```
Out[22]:
      0   1   2   3   4
   0  a  12  13  14  15
   1  b  10  20  30  40
```

```
In [23]: pd.DataFrame(data1.T)
```

```
Out[23]:
       0   1
   0   a   b
   1  12  10
   2  13  20
   3  14  30
   4  15  40
```

```
In [36]: data2 = array([["a",12,13,14,15],["b",10,20,30,40]])
         data2
```

```
Out[36]: array([['a', '12', '13', '14', '15'],
                ['b', '10', '20', '30', '40']], dtype='<U2')
```

```
In [44]: col=['Country',  'Capital' , 'State' , 'City' , 'Street']
```

```
In [39]: pd.DataFrame(data2,index=["str1","str2"])
```

```
Out[39]:
         0   1   2   3   4
   str1  a  12  13  14  15
   str2  b  10  20  30  40
```

```
In [48]: q= pd.DataFrame(data2,index=["str1","str2"],columns=col)  #index and column for writing
         q
```

```
Out[48]:
         Country  Capital  State  City  Street
   str1        a       12     13    14      15
   str2        b       10     20    30      40
```

pandas.read_csv(" FILENAME . csv")

arguments :

- header = None
- nrows = {NUMBER OF LINES TO READ}

In [69]:

```
pd.read_csv('df.csv')  # # the data in csv file is saperated by spaces
```

Out[69]:

	name roll mark
0	qw 12 90
1	qwe 13 89
2	asd 14 78
3	zxc 15 98
4	qaz 16 88
5	lil 17 89

In [70]:

```
f = open("df.csv")
print(f.read())
f.close()
```

```
name roll mark
qw 12 90
qwe 13 89
asd 14 78
zxc 15 98
qaz 16 88
lil 17 89
```

In [59]:

```
pd.read_csv('df.csv',header = 1)
```

Out[59]:

	qw 12 90
0	qwe 13 89
1	asd 14 78
2	zxc 15 98
3	qaz 16 88
4	lil 17 87

In [61]:

```
pd.read_csv('df.csv',nrows = 4)
```

Out[61]:

	name roll mark
0	qw 12 90
1	qwe 13 89
2	asd 14 78
3	zxc 15 98

In [91]:

```
pd.read_csv('df2.csv')   # the data in csv file is saperated by commas unlike the first
```

Out[91]:

	name	roll	mark
0	qw	12	90
1	qwe	13	89
2	asd	14	78
3	zxc	15	98
4	qaz	16	88
5	lil	17	87
6	qw1	12	90
7	qwe1	13	89
8	a1sd	14	78
9	z1xc	15	98
10	q1az	16	88
11	li1l	17	87

In [83]:

```
f = open("df2.csv")
print(f.read())
f.close()
```

```
name,roll,mark
qw,12,90
qwe,13,89
asd,14,78
zxc,15,98
qaz,16,88
lil,17,87
```

In [72]:

```
f1 = pd.read_csv('df.csv')
f1.columns
```

Out[72]:

```
Index(['name roll mark '], dtype='object')
```

In [108]:

```
f2 = pd.read_csv('df2.csv')
f2.columns
```

Out[108]:

```
Index(['name', 'roll', 'mark'], dtype='object')
```

In [89]:
```
print(f2["name"],"\n\n")
print(f2["roll"],"\n\n")
print(f2["mark"],"\n\n")
```

```
0        qw
1       qwe
2       asd
3       zxc
4       qaz
5       lil
6       qw1
7      qwe1
8      a1sd
9      z1xc
10     q1az
11     li1l
Name: name, dtype: object


0     12
1     13
2     14
3     15
4     16
5     17
6     12
7     13
8     14
9     15
10    16
11    17
Name: roll, dtype: int64


0     90
1     89
2     78
3     98
4     88
5     87
6     90
7     89
8     78
9     98
10    88
11    87
Name: mark, dtype: int64
```

In [92]:
```
f2.head() # gives the first 5 values in the data
```

Out[92]:

	name	roll	mark
0	qw	12	90
1	qwe	13	89
2	asd	14	78
3	zxc	15	98
4	qaz	16	88

In [97]:
```
f2.sample()             # this gives one object as the return value randomly for each ru
```

Out[97]:

	name	roll	mark
4	qaz	16	88

In [98]:
```
f2.sample(4)
```

Out[98]:

	name	roll	mark
5	lil	17	87
4	qaz	16	88
2	asd	14	78
7	qwe1	13	89

In [100]:
```
f2.index            #it is an attribute of DataFrame that returns the indices of the data
                    #dataframe object just for the reference and operation of the user
```

Out[100]:
```
RangeIndex(start=0, stop=12, step=1)
```

In [104]:
```
f2.values    # returns the array of all the value content inside the dataframe
```

Out[104]:
```
array([['qw', 12, 90],
       ['qwe', 13, 89],
       ['asd', 14, 78],
       ['zxc', 15, 98],
       ['qaz', 16, 88],
       ['lil', 17, 87],
       ['qw1', 12, 90],
       ['qwe1', 13, 89],
       ['a1sd', 14, 78],
       ['z1xc', 15, 98],
       ['q1az', 16, 88],
       ['li1l', 17, 87]], dtype=object)
```

In [110]:
```
f2.items   # it returns the values in atabulated format ina row/col format
```

Out[110]:
```
<bound method DataFrame.items of     name  roll  mark
0     qw    12    90
1    qwe    13    89
2    asd    14    78
3    zxc    15    98
4    qaz    16    88
5    lil    17    87
6    qw1    12    90
7   qwe1    13    89
8   a1sd    14    78
9   z1xc    15    98
10  q1az    16    88
11  li1l    17    87>
```

In [111]:
```
f2.describe()   # this methord returns the raw estimation of data inside the DataFrame
```

Out[111]:

	roll	mark
count	12.000000	12.000000
mean	14.500000	88.333333
std	1.783765	6.110101
min	12.000000	78.000000
25%	13.000000	87.000000
50%	14.500000	88.500000
75%	16.000000	90.000000
max	17.000000	98.000000

In [113]:
```
f2.info()  # it estimates the Characterstics of objects of each object inside the DataF
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 3 columns):
name    12 non-null object
roll    12 non-null int64
mark    12 non-null int64
dtypes: int64(2), object(1)
memory usage: 416.0+ bytes
```

In [115]:
```
f2.count  ##works the same as 'items' attribute
```

Out[115]:
```
<bound method DataFrame.count of     name  roll  mark
0     qw    12    90
1    qwe    13    89
2    asd    14    78
3    zxc    15    98
4    qaz    16    88
5    lil    17    87
6    qw1    12    90
7   qwe1    13    89
8   a1sd    14    78
9   z1xc    15    98
10  q1az    16    88
11  li1l    17    87>
```

In [122]:
```
f2.dtypes.value_counts()
```

Out[122]:
```
int64     2
object    1
dtype: int64
```

iloc[]

In [124]:
```
f2.loc[0]
```

Out[124]:
```
name    qw
roll    12
mark    90
Name: 0, dtype: object
```

In [134]:
```
f2.iloc[0][1]  # returns Series with only the data of that position # it is a locator :
```

Out[134]:
```
12
```

In [131]:
```
f2.iloc[[0,8]]  # returns data as DF as the original DF had # it takes list of indices (
```

Out[131]:

	name	roll	mark
0	qw	12	90
8	a1sd	14	78

In [139]:
```
f2[2:]  # DF sliceing ::  works for most Data Structures
```

Out[139]:

	name	roll	mark
2	asd	14	78
3	zxc	15	98
4	qaz	16	88
5	lil	17	87
6	qw1	12	90
7	qwe1	13	89
8	a1sd	14	78
9	z1xc	15	98
10	q1az	16	88
11	li1l	17	87

In [140]:
```
f2.iloc[2:]
```

Out[140]:

	name	roll	mark
2	asd	14	78
3	zxc	15	98
4	qaz	16	88
5	lil	17	87
6	qw1	12	90
7	qwe1	13	89
8	a1sd	14	78
9	z1xc	15	98
10	q1az	16	88
11	li1l	17	87

In [146]:
```
f2.iloc[2:8,1:]  # [SLICE 1,SLICE 2]  allows 2d sliceing
```

Out[146]:

	roll	mark
2	14	78
3	15	98
4	16	88
5	17	87
6	12	90
7	13	89

In [148]:
```
f2.iloc[[6,2,9],[0,2]]    # [LIST 1,LIST 2]  allows selection, use  # works the same as
```

Out[148]:

	name	mark
6	qw1	90
2	asd	78
9	z1xc	98

In [160]:
```
# ############## ###difference btw loc[]  iloc[]  #####################


"""
loc[]    :  it is a locator #can call a column with heading "data_name"......
            ??  atleast the indexes must be named during creation to use the above attr

iloc[]   :  it is an index based locator ## cannot call using heading "data_name"......



"""
```

In [161]:
```
f2.loc[1]
```

Out[161]:
```
name     qwe
roll      13
mark      89
Name: 1, dtype: object
```

In [162]:
```
f2.loc[[1]]
```

Out[162]:

	name	roll	mark
1	qwe	13	89

In [163]:
```
f2.iloc[1]
```

Out[163]:
```
name     qwe
roll      13
mark      89
Name: 1, dtype: object
```

# #

# #

In [166]:
```
f2.loc[[1],["name"]]
```

Out[166]:

	name
1	qwe

In [167]:
```
f2.iloc[[1],["name"]]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-167-1565b2cb06dc> in <module>
----> 1 f2.iloc[[1],["name"]]

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
   1416             except (KeyError, IndexError, AttributeError):
   1417                 pass
-> 1418         return self._getitem_tuple(key)
   1419     else:
   1420         # we by definition only have the 0th axis

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self, tup)
   2090     def _getitem_tuple(self, tup):
   2091 
-> 2092         self._has_valid_tuple(tup)
   2093         try:
   2094             return self._getitem_lowerdim(tup)

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _has_valid_tuple(self, key)
    233                 raise IndexingError("Too many indexers")
    234             try:
--> 235                 self._validate_key(k, i)
    236             except ValueError:
    237                 raise ValueError(

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_key(self, key, axi
s)
   2024             if not is_numeric_dtype(arr.dtype):
   2025                 raise IndexError(
-> 2026                     ".iloc requires numeric indexers, got {arr}".format(arr=ar
r)
   2027                 )
   2028 

IndexError: .iloc requires numeric indexers, got ['name']
```

# #

# #

In [ ]: