Pandas

Pandas is defined as an open-source library that provides high-performance data manipulation in Python.

The name of Pandas is derived from the word Panel Data, which means an Econometrics from Multidimensional data.

It is used for data analysis in Python and developed by Wes McKinney in 2008.

we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools.

Pandas is built on top of the Numpy package, means Numpy is required for operating the Pandas.

Benefits of Pandas

Data Representation: It represents the data in a form that is suited for data analysis through its DataFrame and Series.

Clear code: The clear API of the Pandas allows you to focus on the core part of the code. So, it provides clear and concise code for the user.

Python Pandas Data Structure

#series - 1d

#Dataframe -2d

#panel - 3d

Series - It is defined as a one-dimensional array that is capable of storing various data types.

The row labels of series are called the index. We can easily convert the list, tuple, and dictionary into series using "series' method.

A Series cannot contain multiple columns. It has one parameter:

In [1]:

```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])   #ndarray
a = pd.Series(info)
print(a)
print(info)
```

```
0    P
1    a
2    n
3    d
4    a
5    s
dtype: object
['P' 'a' 'n' 'd' 'a' 's']
```

In [6]:

```
d = {'a' : 0., 'b' : 1., 'c' : 2.} #dict
a=pd.Series(d)
```

In [7]:

```
a.index
```

Out[7]:

```
Index(['a', 'b', 'c'], dtype='object')
```

In [15]:

```
print(a.values)
```

```
[0. 1. 2.]
```

In [8]:

```
pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])  #scalar
```

Out[8]:

```
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

In [13]:

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
s
```

Out[13]:

```
a   -1.404596
b    0.570304
c   -0.631735
d   -0.615060
e    0.875518
dtype: float64
```

In [14]:

```
print("type of data",type(s))
```

```
type of data <class 'pandas.core.series.Series'>
```

In [17]:

```
print(s.shape)
print(s.ndim)
print(s.size)
print(s.nbytes)  #5*8
```

```
(5,)
1
5
40
```

In [20]:

```
# Indexing and slicing
s[0]
```

Out[20]:

```
-1.4045956788263276
```

In [21]:

```
s[:3]
```

Out[21]:

```
a   -1.404596
b    0.570304
c   -0.631735
dtype: float64
```

In [22]:

```
s[s > s.median()]
```

Out[22]:

```
b    0.570304
e    0.875518
dtype: float64
```

In [23]:

```
s[[4, 3, 1]]
```

Out[23]:

```
e    0.875518
d   -0.615060
b    0.570304
dtype: float64
```

In [24]:

```
s + s
```

Out[24]:

```
a   -2.809191
b    1.140608
c   -1.263471
d   -1.230120
e    1.751036
dtype: float64
```

In [25]:

```
s * 2
```

Out[25]:

```
a   -2.809191
b    1.140608
c   -1.263471
d   -1.230120
e    1.751036
dtype: float64
```

In [26]:

```
np.exp(s)
```

Out[26]:

```
a    0.245466
b    1.768805
c    0.531668
d    0.540608
e    2.400118
dtype: float64
```

In [27]:

```
s[1:] + s[:-1]
```

Out[27]:

```
a         NaN
b    1.140608
c   -1.263471
d   -1.230120
e         NaN
dtype: float64
```

In [30]:

```
s['e'] = 12 #update
s
```

Out[30]:

```
a    -1.404596
b     0.570304
c    -0.631735
d    -0.615060
e    12.000000
dtype: float64
```

In [33]:

```
'e' in s
```

Out[33]:

```
True
```

In [34]:

```
'f' in s
```

Out[34]:

```
False
```

In [2]:

```
s = pd.Series(np.random.randn(5), name='random series')
s
```

Out[2]:

```
0   -0.025157
1   -1.009122
2   -0.357368
3   -0.889173
4   -0.049994
Name: random series, dtype: float64
```

In [36]:

```
s.name
```

Out[36]:

```
'random series'
```

In [38]:

```
# Checking Emptiness and Presence of NaNs
import numpy as np
import pandas as pd
a=pd.Series(data=[1,2,3,np.NaN])
b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
c=pd.Series()
print(a.empty,b.empty,c.empty)
print(a.hasnans,b.hasnans,c.hasnans)
print(len(a),len(b))
print(a.count( ),b.count( ))
```

```
False False True
True False False
4 3
3 3

C:\Users\91920\anaconda3\lib\site-packages\ipykernel_launcher.py:6: Deprecat
ionWarning: The default dtype for empty Series will be 'object' instead of
'float64' in a future version. Specify a dtype explicitly to silence this wa
rning.
```

In [ ]:

In [ ]:

DataFrame - It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns).

DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index.

We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

In [10]:

```
import pandas as pd
# a list of strings
x = ['Python', 'Pandas']

# Calling DataFrame constructor on list
df = pd.DataFrame(x)
print(df)
```

```
        0
0  Python
1  Pandas
```

In [52]:

```
d = {'one' : [1., 2., 3., 4.], 'two' : [4., 3., 2., 1.]}
pd.DataFrame(d)
```

Out[52]:

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

In [53]:

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

Out[53]:

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0
d	4.0	1.0

In [43]:

```
d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
      'two': pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}    #ndarray,dict and
df = pd.DataFrame(d)
df
```

Out[43]:

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

In [80]:

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

```
     Name Qualification
0     Jai           Msc
1  Princi            MA
2  Gaurav           MCA
3    Anuj           Phd
```

In [81]:

```
print ("Delete the first column:")
del df['Name']
print (df)
```

```
Delete the first column:
   Age    Address Qualification
0   27      Delhi           Msc
1   24     Kanpur            MA
2   22  Allahabad           MCA
3   32    Kannauj           Phd
```

In [83]:

```
#dataframe[col][row]=value  ----------- MODIFY VALUE
df['Age'][1]=700
print(df)
```

```
   Age    Address Qualification
0   27      Delhi           Msc
1  700     Kanpur            MA
2   22  Allahabad           MCA
3   32    Kannauj           Phd

C:\Users\91920\anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
```

In [57]:

```
df_stockprice=pd.DataFrame([[100,200,300,400,500],[10,20,30,40,50]],index=['SBI','HDFC'],co
print(df_stockprice)
```

```
      12-may  13-may  14-may  15-may  16-may
SBI      100     200     300     400     500
HDFC      10      20      30      40      50
```

In [85]:

```
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20130102'),
                    'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                    'D': np.array([3] * 4, dtype='int32'),
                    'E': pd.Categorical(["test", "train", "test", "train"]),
                    'F': 'foo'})
df2
```

Out[85]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [86]:

```
df2.dtypes
```

Out[86]:

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

In [4]:

```
dates = pd.date_range('20201201', periods=6)
dates
```

Out[4]:

```
DatetimeIndex(['2020-12-01', '2020-12-02', '2020-12-03', '2020-12-04',
               '2020-12-05', '2020-12-06'],
              dtype='datetime64[ns]', freq='D')
```

In [5]:

```
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
df
```

Out[5]:

	A	B	C	D
2020-12-01	2.271449	0.747441	0.365299	0.818065
2020-12-02	-0.560790	1.120833	-1.151483	0.558297
2020-12-03	-1.106388	0.059874	-2.457928	-0.682377
2020-12-04	-0.275468	-0.729243	1.815373	1.246609
2020-12-05	1.313735	-0.188765	-0.561172	-1.126471
2020-12-06	0.142360	-0.647427	-0.766369	-1.042490

In [8]:

```
df.head(2) #by defalut 5
```

Out[8]:

	A	B	C	D
2020-12-01	2.271449	0.747441	0.365299	0.818065
2020-12-02	-0.560790	1.120833	-1.151483	0.558297

In [11]:

```
df.tail(1)
```

Out[11]:

	A	B	C	D
2020-12-06	0.14236	-0.647427	-0.766369	-1.04249

In [104]:

```
df.to_numpy() #does not include the index or column labels in the output.
```

Out[104]:

```
array([[-1.78958646e+00, -1.43231953e+00, -8.61920960e-01,
         2.33602248e+00],
       [ 8.86876420e-01, -8.42748086e-01,  3.62692721e-02,
        -5.34087004e-01],
       [ 1.16900241e+00, -4.00982362e-01,  3.56919180e-01,
        -1.00918671e+00],
       [-1.42824495e+00,  4.64848361e-01, -1.00996008e+00,
         1.15615644e+00],
       [ 1.53763852e+00, -3.72703647e-01, -9.90326352e-01,
         8.12168237e-01],
       [-1.48193943e+00, -5.52078365e-01, -2.70121589e-01,
         1.75557690e-03]])
```

In [105]:

```
df.describe()
```

Out[105]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.184376	-0.522664	-0.456523	0.460472
std	1.533110	0.623421	0.582214	1.223505
min	-1.789586	-1.432320	-1.009960	-1.009187
25%	-1.468516	-0.770081	-0.958225	-0.400126
50%	-0.270684	-0.476530	-0.566021	0.406962
75%	1.098471	-0.379773	-0.040328	1.070159
max	1.537639	0.464848	0.356919	2.336022

In [106]:

```
df.T #Transposing your data:
```

Out[106]:

	2020-12-01	2020-12-02	2020-12-03	2020-12-04	2020-12-05	2020-12-06
A	-1.789586	0.886876	1.169002	-1.428245	1.537639	-1.481939
B	-1.432320	-0.842748	-0.400982	0.464848	-0.372704	-0.552078
C	-0.861921	0.036269	0.356919	-1.009960	-0.990326	-0.270122
D	2.336022	-0.534087	-1.009187	1.156156	0.812168	0.001756

In [15]:

```
df.sort_index(axis=1, ascending=False)  #0 index , 1 coloum
```

Out[15]:

	D	C	B	A
2020-12-01	0.818065	0.365299	0.747441	2.271449
2020-12-02	0.558297	-1.151483	1.120833	-0.560790
2020-12-03	-0.682377	-2.457928	0.059874	-1.106388
2020-12-04	1.246609	1.815373	-0.729243	-0.275468
2020-12-05	-1.126471	-0.561172	-0.188765	1.313735
2020-12-06	-1.042490	-0.766369	-0.647427	0.142360

Operation Syntax Result

Select column df[col] Series

Select row by label df.loc[label] Series

Select row by integer location df.iloc[loc] Series

Slice rows df[5:10] DataFrame

In [108]:

```
df['A'] #Selecting a single column
```

Out[108]:

```
2020-12-01   -1.789586
2020-12-02    0.886876
2020-12-03    1.169002
2020-12-04   -1.428245
2020-12-05    1.537639
2020-12-06   -1.481939
Freq: D, Name: A, dtype: float64
```

In [109]:

```
df[0:3]
```

Out[109]:

	A	B	C	D
2020-12-01	-1.789586	-1.432320	-0.861921	2.336022
2020-12-02	0.886876	-0.842748	0.036269	-0.534087
2020-12-03	1.169002	-0.400982	0.356919	-1.009187

In [16]:

```
df.loc[dates[1]] #Selection by label
```

Out[16]:

```
A   -0.560790
B    1.120833
C   -1.151483
D    0.558297
Name: 2020-12-02 00:00:00, dtype: float64
```

In [113]:

```
df.loc['20201202', ['A', 'B']]
```

Out[113]:

```
A    0.886876
B   -0.842748
Name: 2020-12-02 00:00:00, dtype: float64
```

In [111]:

```
df.iloc[3]
```

Out[111]:

```
A   -1.428245
B    0.464848
C   -1.009960
D    1.156156
Name: 2020-12-04 00:00:00, dtype: float64
```

In [114]:

```
df.iloc[3:5, 0:2]
```

Out[114]:

	A	B
2020-12-04	-1.428245	0.464848
2020-12-05	1.537639	-0.372704

In [126]:

```
df['E'] = [1,2,np.nan,np.nan,5,6]
df
```

Out[126]:

	A	B	C	D	E
2020-12-01	-1.789586	-1.432320	-0.861921	2.336022	1.0
2020-12-02	0.886876	-0.842748	0.036269	-0.534087	2.0
2020-12-03	1.169002	-0.400982	0.356919	-1.009187	NaN
2020-12-04	-1.428245	0.464848	-1.009960	1.156156	NaN
2020-12-05	1.537639	-0.372704	-0.990326	0.812168	5.0
2020-12-06	-1.481939	-0.552078	-0.270122	0.001756	6.0

In [131]:

```
df.dropna(how='any') #axis=0,1
```

Out[131]:

	A	B	C	D	E
2020-12-01	-1.789586	-1.432320	-0.861921	2.336022	1.0
2020-12-02	0.886876	-0.842748	0.036269	-0.534087	2.0
2020-12-05	1.537639	-0.372704	-0.990326	0.812168	5.0
2020-12-06	-1.481939	-0.552078	-0.270122	0.001756	6.0

In [132]:

```
df.fillna(value=50)
```

Out[132]:

	A	B	C	D	E
2020-12-01	-1.789586	-1.432320	-0.861921	2.336022	1.0
2020-12-02	0.886876	-0.842748	0.036269	-0.534087	2.0
2020-12-03	1.169002	-0.400982	0.356919	-1.009187	50.0
2020-12-04	-1.428245	0.464848	-1.009960	1.156156	50.0
2020-12-05	1.537639	-0.372704	-0.990326	0.812168	5.0
2020-12-06	-1.481939	-0.552078	-0.270122	0.001756	6.0

In [133]:

```
dir(df)
```

```
 '_try_aggregate_string_function',
 '_typ',
 '_update_inplace',
 '_validate_dtype',
 '_values',
 '_where',
 '_xs',
 'abs',
 'add',
 'add_prefix',
 'add_suffix',
 'agg',
 'aggregate',
 'align',
 'all',
 'any',
 'append',
 'apply',
 'applymap',
 'asfreq',
```

In [ ]: