Pandas

Key Features of Pandas

It contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python

Tools for loading data into in-memory data objects from different file formats.

provides efficient data manipulation

Pandas deals with the following three data structures −

i) Series ii) DataFrame iii) Panel

the powertool of Pandas is its Dataframe

These data structures are built on top of Numpy array, which means they are fast.

i) series

Series is a one-dimensional array like structure with homogeneous data.

Key Points

```
can contain anytype of data
Values of Data Mutable
```

pandas.Series : a series can be created by passing a sequence type object to the function pandas.Series()

pandas.Series(data, index, dtype, copy)

Empty series

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

sr = pd.Series()

print(type(sr))
```

```
<class 'pandas.core.series.Series'>
```

In [2]:
```
data = np.array(['a','b',7.7,'d'])
l=['aa','bb','cc','dd']
s = pd.Series(data ,index=l)
s
```

Out[2]:
```
aa      a
bb      b
cc    7.7
dd      d
dtype: object
```

Index values

In [3]:
```
data = np.array(['a','b','c','d'])

s = pd.Series(data,index=[100,101,102,103])
s
```

Out[3]:
```
100    a
101    b
102    c
103    d
dtype: object
```

Create a Series from dict

In [4]:
```
data = {'a' : 0., 'b' : 1., 'c' : "a"}
sr_1 = pd.Series(data)
sr_1
```

Out[4]:
```
a    0
b    1
c    a
dtype: object
```

In [5]:
```
pd.Series(data,index=['a','b','c','e'])
```

Out[5]:
```
a      0
b      1
c      a
e    NaN
dtype: object
```

Create a Series from Scalar

In [6]:
```
sr_3 = pd.Series(12, index=[0, 1, 2, 3])
sr_3
```

Out[6]:
```
0    12
1    12
2    12
3    12
dtype: int64
```

Accessing Data from Series with Position

In [10]:
```
sr_4 = pd.Series([1,2,3,"b",5],index = ['a','b','c','d','e'])
#print(sr_4)

#print("0 th index value : ",sr_4[0])

#print("slice here : ",sr_4[0:3])


#sr_4[["a","b","e"]]

sr_4.name = "first_series"
sr_4.name
sr_4



sr_4.index.name ="id"
sr_4
```

Out[10]:
```
id
a    1
b    2
c    3
d    b
e    5
Name: first_series, dtype: object
```

In [11]:
```
sr_4.index
```

Out[11]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object', name='id')
```

In [12]:
```
sr_4.values
```

Out[12]:
```
array([1, 2, 3, 'b', 5], dtype=object)
```

In [13]:
```
sr_4.index
```

Out[13]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object', name='id')
```

In [14]:
```
sr_4.isnull().sum()
```

Out[14]:
```
0
```

In [15]:
```
sr_4.name='pop'
sr_4.index.name='namess'
sr_4.name
```

Out[15]:
```
'pop'
```

A series index can be altered by passing list to the . object.index attribute

In [16]:
```
sr_4.index=['bro','ror','tom','prm','srm' ]
sr_4
```

Out[16]:
```
bro    1
ror    2
tom    3
prm    b
srm    5
Name: pop, dtype: object
```

a usefull series feature for many applications is that it automatically aligns by index label in arithmatic operations

In [18]:
```
sr_4 + sr_4
```

Out[18]:
```
bro     2
ror     4
tom     6
prm    bb
srm    10
Name: pop, dtype: object
```

Dataframe

A Data frame represents a rectangular table of data ad contains an ordered collection of columns .

Features of DataFrame

```
* Potentially columns are of different types(numeric ,str , boolean)
* both row and columns indexed .

* Size – Mutable
* Labeled axes (rows and columns)
*Can Perform Arithmetic operations on rows and columns
```

pandas.DataFrame

pandas.DataFrame(data, index, columns, dtype, copy)

Create DataFrame

A pandas DataFrame can be created using various inputs like –

i) Lists , list of lists
ii) dict
iii) list of dict
iv) list of tuples
v) Series
vi) dict of series
vii) Numpy ndarrays
viii) Another DataFrame

In [19]:
```
# creating Dataframe from list
import pandas as pd
import numpy as np
lst=[1,2,3,4,5,6,7,8,9]

df = pd.DataFrame(lst)
df
```

Out[19]:

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

In [25]:
```
lst_1=[["boy",35],["girls",40],["class", 75]]

#lst_1[2][1]= "first"    #values are mutable

pd.DataFrame(lst_1,columns=['students','no of std'],dtype=int)
```

Out[25]:

	students	no of std
0	boy	35
1	girls	40
2	class	75

In [57]:
```
#index=[] argument
df3=pd.DataFrame(lst_1,index=["a","b","c"],columns=['students','no of std'],dtype=float)
```

In [ ]:

In [50]:
```
# creating DF by using dictionaries


dic={"day":['sun','mon','tue','wed','fri'],
       "month":['may','jun','july','aug','sept'],
    'temp':[21,22,23,24,34]}

index=['delhi','ajmera','nyk','bom','ker']

df_1=pd.DataFrame(dic,index=index)
#type(df_1.month)

df_1
```

Out[50]:

	day	month	temp
delhi	sun	may	21
ajmera	mon	jun	22
nyk	tue	july	23
bom	wed	aug	24
ker	fri	sept	34

In [88]:
```
# creating DataFrame from list of dictionaries
dic_1=[{'a':20, 'b':32},{'c':21},{'d':33333}]
df1=pd.DataFrame(dic_1)
df1["stat"]= df1.a==20
df1
```

Out[88]:

	a	b	c	d	stat
0	20.0	32.0	NaN	NaN	True
1	NaN	NaN	21.0	NaN	False
2	NaN	NaN	NaN	33333.0	False

In [73]:
```
# Create a DataFrame from Dict of Series
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df2=pd.DataFrame(d)

df2
```

Out[73]:

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Accessing and Retrieving data

.head()

.tail()

.columns

df.loc["row_lable"] # to access the required row

data['column_names']

df.loc[row_index,col_index]

df.ix[row_index,col_index]

df.set_index= " " # will set the index of your choice

a new column can be added by using a series

df[new_column] = series

del data['column_name'] :: del method can be use to remove any column

In [89]:
```
#Column can be modified  by assignment
df2.two = 43
df2.two=  np.arange(4)
df2.two= pd.Series([1,2,3],index = ["a","b","c"])
df2
```

Out[89]:

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

In [ ]:
```
# adding columns in a dataframe
# appending a data frame
# deleating a column  # df.pop("column") , del df["column_name"]
# deleating the rows  # df.drop("row")
```

loading data from

```
>> csv files
>> excel files
## pd.read_csv('data.csv') # if data is in the directory file
## pd.read_csv('c:\\data\\data.csv')   # if data is in some other locaton

pd.read_excel("data.xlsx")
```

reading and writing csv and excel

df.to_csv('new.csv')

In [1]:
```
#df.to_csv('new.csv',columns=['',''],header=False) 4
```

In [55]:
```
# using del function
print("Deleting the first column using DEL function:")
del df2['one']
print(df2)

# using pop function
print("Deleting another column using POP function:")
df2.pop('two')
print(df2)
# Drop rows with label 0
df2 = df2.drop(1)
```

```
Deleting the first column using DEL function:

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\users\dharm\appdata\local\programs\python\python36\lib\site-packages\pandas\core\indexes\base.py in get
_loc(self, key, method, tolerance)
   3077             try:
-> 3078                 return self._engine.get_loc(key)
   3079             except KeyError:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'one'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-55-f7e3a2c4bbe2> in <module>
      1 # using del function
      2 print("Deleting the first column using DEL function:")
----> 3 del df2['one']
      4 print(df2)
      5

c:\users\dharm\appdata\local\programs\python\python36\lib\site-packages\pandas\core\generic.py in __delite
m__(self, key)
   2741             # there was no match, this call should raise the appropriate
   2742             # exception:
-> 2743             self._data.delete(key)
   2744
   2745         # delete from the caches

c:\users\dharm\appdata\local\programs\python\python36\lib\site-packages\pandas\core\internals.py in delete
(self, item)
   4172         Delete selected item (items if non-unique) in-place.
   4173         """
-> 4174         indexer = self.items.get_loc(item)
   4175
   4176         is_deleted = np.zeros(self.shape[0], dtype=np.bool_)

c:\users\dharm\appdata\local\programs\python\python36\lib\site-packages\pandas\core\indexes\base.py in get
_loc(self, key, method, tolerance)
   3078                 return self._engine.get_loc(key)
   3079             except KeyError:
-> 3080                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   3081
   3082         indexer = self.get_indexer([key], method=method, tolerance=tolerance)

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'one'
```

methods/ functions

```
1  axes    Returns a list of the row axis labels.
2  dtype   Returns the dtype of the object.
3  empty   Returns True if series is empty.
4  ndim    Returns the number of dimensions of the underlying data, by definition 1.
5  size    Returns the number of elements in the underlying data.
6  values  Returns the Series as ndarray.
7  head()  Returns the first n rows.
8  tail()  Returns the last n rows.
   sum()
   1   count() Number of non-null observations
   2   sum()   Sum of values
   3   mean()  Mean of Values
   4   median()    Median of Values
   5   mode()  Mode of values
   6   std()   Standard Deviation of the Values
   7   min()   Minimum Value
   8   max()   Maximum Value
   9   abs()   Absolute Value
   10  prod()  Product of Values
   11  cumsum()    Cumulative Sum
   12  cumprod()   Cumulative Product

   sorted_df = unsorted_df.sort_index(ascending=False)
   sorted_df=unsorted_df.sort_index(axis=1)
   frame.corr()
   cov()
   df.reindex([,,,,,,,,])

   df['one'].isnull() # checking for null values in a particular column
```

Calculations with Missing Data

```
  * When summing data, NA will be treated as Zero
  * If the data are all NA, then the result will be NA
```

Cleaning / Filling Missing Data

```
   Replace NaN with a Scalar Value
   df.fillna(0)
```

Drop Missing Values

```
   If you want to simply exclude the missing values, then use the dropna function along with the axis argument.
By            default, axis=0, i.e., along row, which means that if any value within a row is NA then the whole
row is excluded.

   df.dropna()
   df.dropna(axis=1)
```

Replace Missing

```
   df.replace({1000:10,2000:60}) # pass a dictionary to replace the values
```

In [ ]:
```
#fdd=pd.read_html()
#dat=pd.read_json()
```

Data transformation

Removing Duplicates

```
data.duplicated()

data.replace()
data.fillna()
data.dropna()
```

In [42]:
```
data_1= pd.DataFrame({"k1":["one","two"]*3+["two"],
                      "k2":[1,1,2,3,3,4,4]})
data_1.duplicated(["k1","k2"])
data_1
```

Out[42]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

In [38]:
```
data_1.duplicated()
```

Out[38]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [43]:
```
data_1.drop_duplicates(["k1","k2"])
```

Out[43]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

In [35]:
```
data_1.drop_duplicates("k1") # pass the column you want to drop duplicates
data_1.drop_duplicates(["k1","k2"])
```

Out[35]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

Discretization and binning

```
pd.cut(data,bins)     ,
pd.value_counts
```

In [37]:
```
ages =[20,22,22,23,24,24,26,22,28,29,30,32,38,40,42,45,50,51,51]
bins=[18,25,30,50,70,100]
cutss=pd.cut(ages,bins)
cutss
```

Out[37]:
```
[(18, 25], (18, 25], (18, 25], (18, 25], (18, 25], ..., (30, 50], (30, 50], (30, 50], (50, 70], (50, 70]]
Length: 19
Categories (5, interval[int64]): [(18, 25] < (25, 30] < (30, 50] < (50, 70] < (70, 100]]
```

Detecting and Filtering Outliers

```
pd.describe()
```

dealing with categorical data

```
*pd.get_dummies(data["column"])
* OneHotEncoder
```

Types of categorical Variables we could have

```
Nominal # we cant compare

Ordinal   # comparision can be there
```

In [ ]:
```
import pandas as pd
dummies =pd.get_dummies(df.Age)
df = pd.concat([df,dummies],axis ="columns")
df.drop([''])
```