Pandas: Panel Data is abbreviated to pandas

Why pandas:

- **Powerful data structure-fast and efficient data wrangling item**
- **Data aggregation and transformation**
- **easy input and output data with writing to files and reading from files**
- **List item**

Series: One Dimention Dataframe: Two Dimension Panel: 3 dimension

In [1]:

```
# Creating a pandas series

import pandas as pd

# creating series from list

list1=[1,2,3]
series1=pd.Series(list1)

print(series1)
```

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

In [5]:

```
# creating series from numpy

import numpy as np

ind=['a', 'b', 'c']
arr1=np.array(list1)

pd.Series(data=arr1, index=ind)
```

Out[5]:

```
a    1
b    2
c    3
dtype: int64
```

In [6]:

```
# creatubg series from dict

dict={'Shivam':'Teacher', 'Mala':'Student', 'Laptop':'Notes'}
pd.Series(dict)
```

Out[6]:

```
Shivam     Teacher
Sukanya    Student
Laptop       Notes
dtype: object
```

In [7]:

```
dict={'Shivam':'100', 'Sukanya':'90', 'Krishna':'99'}
pd.Series(dict)
```

Out[7]:

```
Shivam     100
```

```
Sukanya     90
Krishna     99
dtype: object
```

Indexes

Pandas use index to for eazy searching (similar to dictionary or hashing)

In [8]:

```
ser_1=pd.Series([1,2,3,4], index=['Shivam', 'Singh', 'Charles', 'Potter']) # custom index

print(ser_1['Charles']) #similar to calling by key value pairs in a dictionary
```

```
3
```

In [ ]:

```

```

Accessing elements

In [11]:

```
ser1= pd.Series(['S','h','i','v','a','m'])
ser1
```

Out[11]:

```
0    S
1    h
2    i
3    v
4    a
5    m
dtype: object
```

In [13]:

```
ser1[2]
```

Out[13]:

```
'i'
```

In [15]:

```
ser1[0:4]
```

Out[15]:

```
0    S
1    h
2    i
3    v
dtype: object
```

In [16]:

```
ser1.values # values is an attribute and not a function
```

Out[16]:

```
array(['S', 'h', 'i', 'v', 'a', 'm'], dtype=object)
```

In [19]:

```
ser2=pd.Series(['God', 'is', 'good', 'and', 'great'], [1,2,3,4,5])
```

```
ser2
```

Out[19]:

```
1      God
2       is
3     good
4      and
5    great
dtype: object
```

In [20]:

```
ser2[:3]
```

Out[20]:

```
1     God
2      is
3    good
dtype: object
```

In [21]:

```
ser2.values
```

Out[21]:

```
array(['God', 'is', 'good', 'and', 'great'], dtype=object)
```

In [22]:

```
ser2.index
```

Out[22]:

```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

In [38]:

```
ser3=pd.Series(['amar', 'abdul', 'ajay'], ['a', 'b', 'c'])
ser3.index
```

Out[38]:

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

In [37]:

```
ser3.drop('a', inplace=True)
ser3
```

Out[37]:

```
b    abdul
c     ajay
dtype: object
```

In [42]:

```
ser3['c']='Charles'
ser3
```

Out[42]:

```
a       amar
b      abdul
c    Charles
dtype: object
```

Dataframes

- **included in pandas**

- **row-->measures**
- **column-->data for the variable under study**

2D data

Data frames consists of

1. **Data**
2. **Index**
3. **Columns**

In [44]:

```
# Creating dataframes

import pandas as pd
import numpy as np

df=pd.DataFrame([[11,12,13], [14,15,16], [17,18,19]])
df
```

Out[44]:

	0	1	2
0	**11**	**12**	**13**
1	**14**	**15**	**16**
2	**17**	**18**	**19**

In [45]:

```
# Shape-->attribute which gives rows and columns

df.shape
```

Out[45]:

```
(3, 3)
```

In [46]:

```
df.index
```

Out[46]:

```
RangeIndex(start=0, stop=3, step=1)
```

To access specific columns we use loc and iloc

In [48]:

```
df.loc[1] # gives the value of the row corresponding to the specified index
```

Out[48]:

```
0    14
1    15
2    16
Name: 1, dtype: int64
```

In [49]:

```
type(df.loc[1])
```

Out[49]:

```
pandas.core.series.Series
```

In [52]:

```
# For multiple rows

df.loc[[0,2]]  #Note: only square bracket. for 2D, use 2 square brackets
```

Out[52]:

	0	1	2
0	11	12	13
2	17	18	19

In [53]:

```
# select rows and columns

df.loc[1,0]  #[Row, columnn]
```

Out[53]:

14

In [69]:

```
# selecting  multiple row/columns
df1=pd.DataFrame([['a', 'b','c', 'd', 'e'],['aa', 'bb', 'cc', 'dd', 'ee'],['aaa', 'bbb',
'ccc', 'ddd', 'eee']])
df1
```

Out[69]:

	0	1	2	3	4
0	a	b	c	d	e
1	aa	bb	cc	dd	ee
2	aaa	bbb	ccc	ddd	eee

In [74]:

```
df1.loc[[1,2],[0,2]] # multiple rows and columns can be selected
```

Out[74]:

	0	2
1	aa	cc
2	aaa	ccc

In [76]:

```
df2=pd.DataFrame(np.random.rand(5,4)) # create a dataframe from numpy
df2
```

Out[76]:

	0	1	2	3
0	0.757979	0.702727	0.642190	0.395020
1	0.040179	0.405281	0.637647	0.025732
2	0.548982	0.159618	0.570354	0.265411
3	0.269617	0.433987	0.950473	0.699003
4	0.893415	0.689420	0.940556	0.797780

In [5]:

```
# can set the columns and index
import pandas as pd
import numpy as np

df3=pd.DataFrame(np.random.rand(5,5), index=['1', '2','3','4','5'], columns=['A', 'B', '
C', 'D', 'E'])
df3
```

Out[5]:

	A	B	C	D	E
1	**0.759529**	**0.951694**	**0.420392**	**0.416662**	**0.804216**
2	**0.796179**	**0.267372**	**0.020443**	**0.635148**	**0.902215**
3	**0.486212**	**0.925857**	**0.384783**	**0.907503**	**0.672541**
4	**0.845540**	**0.538245**	**0.785797**	**0.296111**	**0.468723**
5	**0.481530**	**0.045254**	**0.992635**	**0.368201**	**0.779094**

In [6]:

```
print(df3.columns)
```

```
Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

In [7]:

```
print(df3.index)
```

```
Index(['1', '2', '3', '4', '5'], dtype='object')
```

In [9]:

```
df3.loc['3'] # .loc uses [] and within that we use index value or the columns value
```

Out[9]:

```
A    0.486212
B    0.925857
C    0.384783
D    0.907503
E    0.672541
Name: 3, dtype: float64
```

In [12]:

```
df3.loc[['2','4'], ['A', 'B']] # to choose specific rows and columns
```

Out[12]:

	A	B
2	**0.796179**	**0.267372**
4	**0.845540**	**0.538245**

In [13]:

```
df3.iloc[0] # similar to loc, here we use only the index
```

Out[13]:

```
A    0.759529
B    0.951694
C    0.420392
D    0.416662
E    0.804216
Name: 1, dtype: float64
```

In [14]:

```
df3.iloc[[0,2,4],[1]] #Here column is B, but we access with the index for the columns as 0,1,2...
```

Out[14]:

	B
1	0.951694
3	0.925857
5	0.045254

Access based on conditions

In [15]:

```
df3
```

Out[15]:

	A	B	C	D	E
1	0.759529	0.951694	0.420392	0.416662	0.804216
2	0.796179	0.267372	0.020443	0.635148	0.902215
3	0.486212	0.925857	0.384783	0.907503	0.672541
4	0.845540	0.538245	0.785797	0.296111	0.468723
5	0.481530	0.045254	0.992635	0.368201	0.779094

In [16]:

```
df3>0.2 #checks for the condition and states boolean values
```

Out[16]:

	A	B	C	D	E
1	True	True	True	True	True
2	True	True	False	True	True
3	True	True	True	True	True
4	True	True	True	True	True
5	True	False	True	True	True

In [17]:

```
df3[df3>0.2] # only returns the values as per the condition given
```

Out[17]:

	A	B	C	D	E
1	0.759529	0.951694	0.420392	0.416662	0.804216
2	0.796179	0.267372	NaN	0.635148	0.902215
3	0.486212	0.925857	0.384783	0.907503	0.672541
4	0.845540	0.538245	0.785797	0.296111	0.468723
5	0.481530	NaN	0.992635	0.368201	0.779094

In [25]:

```
df3[(df3['A']>0.4)& (df3['C']<0.8)] #can choose multiple conditions
```

Out[25]:

	A	B	C	D	E
1	0.759529	0.951694	0.420392	0.416662	0.804216
2	0.796179	0.267372	0.020443	0.635148	0.902215
3	0.486212	0.925857	0.384783	0.907503	0.672541
4	0.845540	0.538245	0.785797	0.296111	0.468723

In [29]:

```
df3[(df3['A']>0.4) & (df3['C']>0.8)] # retuns the values only these two conditions are satisfied. (intersection)
```

Out[29]:

	A	B	C	D	E
5	0.48153	0.045254	0.992635	0.368201	0.779094

In [30]:

```
# To select specific column values
df3[df3>0.2]['A']
```

Out[30]:

```
1    0.759529
2    0.796179
3    0.486212
4    0.845540
5    0.481530
Name: A, dtype: float64
```

In [34]:

```
df3
```

Out[34]:

	A	B	C	D
1	0.759529	0.951694	0.420392	0.416662
2	0.796179	0.267372	0.020443	0.635148
3	0.486212	0.925857	0.384783	0.907503
4	0.845540	0.538245	0.785797	0.296111
5	0.481530	0.045254	0.992635	0.368201

In [35]:

```
df3=pd.DataFrame(np.random.rand(5,5), index=['1', '2','3','4','5'], columns=['A', 'B', 'C', 'D', 'E'])
df3
```

Out[35]:

	A	B	C	D	E
1	0.543345	0.842597	0.741067	0.030300	0.178024
2	0.492359	0.379912	0.727657	0.531254	0.113078
3	0.392997	0.442506	0.545522	0.163823	0.192328
4	0.623160	0.031885	0.775418	0.358421	0.430473
5	0.968686	0.434094	0.266399	0.016026	0.446078

In [36]:

```
df3.drop('E', axis=1, inplace=True)
df3
```

Out[36]:

	A	B	C	D
1	0.543345	0.842597	0.741067	0.030300
2	0.492359	0.379912	0.727657	0.531254
3	0.392997	0.442506	0.545522	0.163823
4	0.623160	0.031885	0.775418	0.358421
5	0.968686	0.434094	0.266399	0.016026

Adding rows to DataFrames

In [41]:

```
#Adding new rows to DF
df3.append(pd.Series([0.0001, 0.0002, 0.0003, 0.0004], name='6', index=['A','B','C','D']
))

# name is to add a name for the index-->adding index element for the rows
# Here index is for the column; column indexing
```

Out[41]:

	A	B	C	D
1	0.543345	0.842597	0.741067	0.030300
2	0.492359	0.379912	0.727657	0.531254
3	0.392997	0.442506	0.545522	0.163823
4	0.623160	0.031885	0.775418	0.358421
5	0.968686	0.434094	0.266399	0.016026
6	0.000100	0.000200	0.000300	0.000400

In [42]:

```
# to reset the index

df3.reset_index() # the original index statys after resetting
```

Out[42]:

	index	A	B	C	D
0	1	0.543345	0.842597	0.741067	0.030300
1	2	0.492359	0.379912	0.727657	0.531254
2	3	0.392997	0.442506	0.545522	0.163823
3	4	0.623160	0.031885	0.775418	0.358421
4	5	0.968686	0.434094	0.266399	0.016026

In [43]:

```
# to remove the original indexing after resetting index

df3.reset_index(drop=True) # drops all other manual indexing
```

Out[43]:

	A	B	C	D

	A	B	C	D
0	0.543345	0.842597	0.741067	0.030300
1	0.492359	0.379912	0.727657	0.531254
2	0.392997	0.442506	0.545522	0.163823
3	0.623160	0.031885	0.775418	0.358421
4	0.968686	0.434094	0.266399	0.016026

In [46]:

```
# sorting based on index
df3.sort_index() # This can be coupled with reset index as follows

df3.sort_index().reset_index(drop=True)
```

Out[46]:

	A	B	C	D
0	0.543345	0.842597	0.741067	0.030300
1	0.492359	0.379912	0.727657	0.531254
2	0.392997	0.442506	0.545522	0.163823
3	0.623160	0.031885	0.775418	0.358421
4	0.968686	0.434094	0.266399	0.016026

Reading CSV

In [4]:

```
import pandas as pd
df=pd.read_csv("hotel_bookings.csv")
df
```

Out[4]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_m
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	
...	...	...	...	...	...	...	
119385	City Hotel	0	23	2017	August	35	
119386	City Hotel	0	102	2017	August	35	
119387	City Hotel	0	34	2017	August	35	
119388	City Hotel	0	109	2017	August	35	
119389	City Hotel	0	205	2017	August	35	

119390 rows × 32 columns

In [5]:

```
print('Shape', df.shape)
print('Index', df.index)
```

```
Shape (119390, 32)
Index RangeIndex(start=0, stop=119390, step=1)
```

In [6]:

```
# to display the first 5 rows

df.head()
```

Out[6]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
0	Resort Hotel	0	342	2015	July	27	1
1	Resort Hotel	0	737	2015	July	27	1
2	Resort Hotel	0	7	2015	July	27	1
3	Resort Hotel	0	13	2015	July	27	1
4	Resort Hotel	0	14	2015	July	27	1

In [7]:

```
# last 5 rows

df.tail()
```

Out[7]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mo
119385	City Hotel	0	23	2017	August	35	
119386	City Hotel	0	102	2017	August	35	
119387	City Hotel	0	34	2017	August	35	
119388	City Hotel	0	109	2017	August	35	
119389	City Hotel	0	205	2017	August	35	

In [10]:

```
# display only specific columns

df[['hotel', 'is_canceled', 'arrival_date_month']]
```

Out[10]:

	hotel	is_canceled	arrival_date_month
0	Resort Hotel	0	July
1	Resort Hotel	0	July

	hotel	is_canceled	arrival_date_month
2	Resort Hotel	0	July
3	Resort Hotel	0	July
4	Resort Hotel	0	July
...	...	...	...
119385	City Hotel	0	August
119386	City Hotel	0	August
119387	City Hotel	0	August
119388	City Hotel	0	August
119389	City Hotel	0	August

119390 rows × 3 columns

In [11]:

```
# Analyze dataframes
df.info()

# info gives the summary such as rows, columns, data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119390 non-null  object 
 1   is_canceled                     119390 non-null  int64  
 2   lead_time                       119390 non-null  int64  
 3   arrival_date_year               119390 non-null  int64  
 4   arrival_date_month              119390 non-null  object 
 5   arrival_date_week_number        119390 non-null  int64  
 6   arrival_date_day_of_month       119390 non-null  int64  
 7   stays_in_weekend_nights         119390 non-null  int64  
 8   stays_in_week_nights            119390 non-null  int64  
 9   adults                          119390 non-null  int64  
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64  
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
 31  reservation_status_date         119390 non-null  object 
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

In [12]:

```
# To get unique values

df['hotel'].unique()
```

Out[12]:

```
array(['Resort Hotel', 'City Hotel'], dtype=object)
```

In [14]:

```
df['reservation_status'].unique()
```

Out[14]:

```
array(['Check-Out', 'Canceled', 'No-Show'], dtype=object)
```

In [15]:

```
df['customer_type'].unique()
```

Out[15]:

```
array(['Transient', 'Contract', 'Transient-Party', 'Group'], dtype=object)
```

In [16]:

```
# to count the total values of a column under specific categories
df['reservation_status'].value_counts()
```

Out[16]:

```
Check-Out    75166
Canceled     43017
No-Show       1207
Name: reservation_status, dtype: int64
```

In [17]:

```
df['hotel'].value_counts()
```

Out[17]:

```
City Hotel      79330
Resort Hotel    40060
Name: hotel, dtype: int64
```

In [18]:

```
df['customer_type'].value_counts()
```

Out[18]:

```
Transient          89613
Transient-Party    25124
Contract            4076
Group                577
Name: customer_type, dtype: int64
```

In [19]:

```
# to get the first 10 rows

df[:10]
```

Out[19]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
0	Resort Hotel	0	342	2015	July	27	1
1	Resort Hotel	0	737	2015	July	27	1
2	Resort Hotel	0	7	2015	July	27	1
3	Resort Hotel	0	13	2015	July	27	1
4	Resort	0	14	2015	July	27	1

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
5	Resort Hotel	0	14	2015	July	27	1
6	Resort Hotel	0	0	2015	July	27	1
7	Resort Hotel	0	9	2015	July	27	1
8	Resort Hotel	1	85	2015	July	27	1
9	Resort Hotel	1	75	2015	July	27	1

In [23]:

```
# Selecting based on multiple conditions

df[(df['hotel']=='Resort Hotel') & (df['arrival_date_month']=='July')]
```

Out[23]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mo
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	
...	...	...	...	...	...	...	
39195	Resort Hotel	0	301	2017	July	30	
39196	Resort Hotel	0	301	2017	July	30	
39230	Resort Hotel	0	187	2017	July	30	
39236	Resort Hotel	0	173	2017	July	31	
39249	Resort Hotel	0	269	2017	July	31	

4573 rows × 32 columns

In [24]:

```
# describe

df.describe() # summary of the statistical calculations such as dispersion, mean, median
and mode
```

Out[24]:

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weeken
count	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	11939
mean	0.370416	104.011416	2016.156554	27.165173	15.798241	
std	0.482918	106.863097	0.707476	13.605138	8.780829	

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weeken
min	0.000000	0.000000	2015.000000	1.000000	1.000000	
25%	0.000000	18.000000	2016.000000	16.000000	8.000000	
50%	0.000000	69.000000	2016.000000	28.000000	16.000000	
75%	1.000000	160.000000	2017.000000	38.000000	23.000000	
max	1.000000	737.000000	2017.000000	53.000000	31.000000	1

In [26]:

```
df['lead_time'].describe()
```

Out[26]:

```
count    119390.000000
mean        104.011416
std         106.863097
min           0.000000
25%          18.000000
50%          69.000000
75%         160.000000
max         737.000000
Name: lead_time, dtype: float64
```

In [27]:

```
# Rename columns

df.rename(columns={'arrival_date_month': 'arrival_month'}, inplace=True)
df.head()
```

Out[27]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_month	arrival_date_week_number	arrival_date_day_of_month	stays
0	Resort Hotel	0	342	2015	July	27	1	
1	Resort Hotel	0	737	2015	July	27	1	
2	Resort Hotel	0	7	2015	July	27	1	
3	Resort Hotel	0	13	2015	July	27	1	
4	Resort Hotel	0	14	2015	July	27	1	

In [32]:

```
# Drop columns

df.drop(labels=['arrival_date_week_number'], axis=1, inplace=True)
df
```

Out[32]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_month	arrival_date_day_of_month	stays_in_weekend_nights
0	Resort Hotel	0	342	2015	July	1	0
1	Resort Hotel	0	737	2015	July	1	0
2	Resort Hotel	0	7	2015	July	1	0
3	Resort Hotel	0	13	2015	July	1	0

	hotel	is_canceled	lead_time	arrival_date_year	arrival_month	arrival_date_day_of_month	stays_in_weekend_nights
4	Resort Hotel	0	14	2015	July	1	0
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	30	2
119386	City Hotel	0	102	2017	August	31	2
119387	City Hotel	0	34	2017	August	31	2
119388	City Hotel	0	109	2017	August	31	2
119389	City Hotel	0	205	2017	August	29	2

119390 rows × 31 columns

In [40]:

```
# Top 5 highest booked months

df['arrival_month'].value_counts().sort_values(ascending=False).head()
```

Out[40]:

```
August     13877
July       12661
May        11791
October    11160
April      11089
Name: arrival_month, dtype: int64
```

In [43]:

```
# Mean of the stays

df['lead_time'].mean()
```

Out[43]:

```
104.01141636652986
```

In [45]:

```
# Grouping by values based on categories
import numpy as np

df.groupby(df['hotel']).agg(np.mean)
```

Out[45]:

hotel	is_canceled	lead_time	arrival_date_year	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights
City Hotel	0.417270	109.735724	2016.174285	15.786625	0.795185	2.182957
Resort Hotel	0.277634	92.675686	2016.121443	15.821243	1.189815	3.128732

Concat Join and Merge

Concat: attaching two dataframes (the shape should match)

Join: Has several types: inner, outer, left, right. The indexes may be different

Merge: Combine all to 1 data frame

In [47]:

```
# Concat example

df_1=pd.DataFrame({'A': [1,2,3,4,5], 'B': [11,12,13,14,15], 'C': [21,22,23,24,25]})
df_1
```

Out[47]:

	A	B	C
0	1	11	21
1	2	12	22
2	3	13	23
3	4	14	24
4	5	15	25

In [52]:

```
df_2=pd.DataFrame({'A': [31, 32, 33,34,35], 'B':[41,42,43,44,45], 'C':[51,52,53,54,55]}
)
df_2
```

Out[52]:

	A	B	C
0	31	41	51
1	32	42	52
2	33	43	53
3	34	44	54
4	35	45	55

In [53]:

```
pd.concat([df_1, df_2]) # when axis not mentioned, it adds more rows
```

Out[53]:

	A	B	C
0	1	11	21
1	2	12	22
2	3	13	23
3	4	14	24
4	5	15	25
0	31	41	51
1	32	42	52
2	33	43	53
3	34	44	54
4	35	45	55

In [54]:

```
pd.concat([df_1, df_2], axis=1) # giving axis=1 addes to the columns
```

Out[54]:

	A	B	C	A	B	C
0	1	11	21	31	41	51
1	2	12	22	32	42	52
2	3	13	23	33	43	53
3	4	14	24	34	44	54
4	5	15	25	35	45	55

In [63]:

```
# Merging

one=pd.DataFrame({'col1': [1,2,3,4,5], 'col2': [11,12,13,14,15], 'shiv':[121,122,123,124
,125]})
two=pd.DataFrame({'col1': [31,32,33,34,35], 'col2': [41,42,43,44,45], 'suki': [221,222,2
23,224,225]})
```

In [59]:

```
pd.merge(one,two, how='inner', on=['col1', 'col2'])
```

Out[59]:

	col1	col2	shiv	suki

In [60]:

```
pd.merge(one,two, how='outer', on=['col1', 'col2'])
```

Out[60]:

	col1	col2	shiv	suki
0	1	11	121.0	NaN
1	2	12	122.0	NaN
2	3	13	123.0	NaN
3	4	14	124.0	NaN
4	5	15	125.0	NaN
5	31	41	NaN	221.0
6	32	42	NaN	222.0
7	33	43	NaN	223.0
8	34	44	NaN	224.0
9	35	45	NaN	225.0