In [1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
df = pd.DataFrame({'A':[1,2,np.nan],'B':[5,np.nan,np.nan],'C':[1,2,3]})
df['States']="CA NV AZ".split()
df.set_index('States',inplace=True)
print(df)
```

```
          A    B  C
States
CA      1.0  5.0  1
NV      2.0  NaN  2
AZ      NaN  NaN  3
```

In [4]:
```
df = pd.DataFrame({'A':[1,2,np.nan],'B':[5,np.nan,np.nan],'C':[1,2,3]})
df
```

Out[4]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [3]:
```
print("\nDropping any rows with a NaN value\n",'-'*35, sep='')
print(df.dropna(axis=0))
```

```
Dropping any rows with a NaN value
-----------------------------------
          A    B  C
States
CA      1.0  5.0  1
```

In [4]:
```
print("\nDropping any column with a NaN value\n",'-'*35, sep='')
print(df.dropna(axis=1))
```

```
Dropping any column with a NaN value
-----------------------------------
        C
States
CA      1
NV      2
AZ      3
```

In [5]:
```
print("\nDropping a row with a minimum 2 NaN value using 'thresh' parameter\n",'-
print(df.dropna(axis=0, thresh=2))
```

```
Dropping a row with a minimum 2 NaN value using 'thresh' parameter
------------------------------------------------------------------------
          A    B  C
States
CA      1.0  5.0  1
NV      2.0  NaN  2
```

In [6]:
```
print("\nFilling values with a default value\n",'-'*35, sep='')
print(df.fillna(value='FILL VALUE'))
```

```
Filling values with a default value
-----------------------------------
                A           B  C
States
CA              1           5  1
NV              2  FILL VALUE  2
AZ      FILL VALUE  FILL VALUE  3
```

In [7]:
```
print("\nFilling values with a computed value (mean of column A here)\n",'-'*60,
print(df.fillna(value=df['A'].mean()))
```

```
Filling values with a computed value (mean of column A here)
------------------------------------------------------------
          A    B  C
States
CA      1.0  5.0  1
NV      2.0  1.5  2
AZ      1.5  1.5  3
```

In [5]:
```
# Create dataframe
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Sales':[200,120,340,124,243,350]}
df = pd.DataFrame(data)
df
```

Out[5]:

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

In [7]:
```
byComp = df.groupby('Company')
print(byComp)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F04B2F5448>
```

In [9]:
```
#avg saless of google
byComp = df.groupby('Company')
print("\nGrouping by 'Company' column and listing mean sales\n",'-'*55, sep='')
print(byComp.mean())
```

```
Grouping by 'Company' column and listing mean sales
-------------------------------------------------------
         Sales
Company
FB       296.5
GOOG     160.0
MSFT     232.0
```

In [ ]:
```

```

In [10]:
```
print("\nGrouping by 'Company' column and listing sum of sales\n",'-'*55, sep='')
print(byComp.sum())
```

```
Grouping by 'Company' column and listing sum of sales
-------------------------------------------------------
         Sales
Company
FB         593
GOOG       320
MSFT       464
```

In [11]:
```
print("\nAll in one line of command (Stats for 'FB')\n",'-'*65, sep='')
print(pd.DataFrame(df.groupby('Company').describe().loc['FB']).transpose())
```

```
All in one line of command (Stats for 'FB')
-----------------------------------------------------------------
   Sales
   count   mean        std    min     25%    50%     75%    max
FB   2.0  296.5  75.660426  243.0  269.75  296.5  323.25  350.0
```

In [6]:
```
df.groupby('Company').describe()
```

Out[6]:

	Sales							
	count	mean	std	min	25%	50%	75%	max
Company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

In [12]:
```
(pd.DataFrame(df.groupby('Company').describe().loc['FB'])).transpose()
```

Out[12]:

	Sales							
	count	mean	std	min	25%	50%	75%	max
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0

In [ ]:

In [13]:
```
(pd.DataFrame(df.groupby('Company').describe().loc['FB'])).transpose()
```

Out[13]:

	Sales							
	count	mean	std	min	25%	50%	75%	max
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0

In [ ]:

In [ ]:

In [14]:
```
print("\nSame type of extraction with little different command\n",'-'*68, sep='')
print(df.groupby('Company').describe().loc[['GOOG', 'MSFT']])
```

```
Same type of extraction with little different command
--------------------------------------------------------------------
        Sales
        count   mean         std    min    25%    50%    75%    max
Company
GOOG      2.0  160.0   56.568542  120.0  140.0  160.0  180.0  200.0
MSFT      2.0  232.0  152.735065  124.0  178.0  232.0  286.0  340.0
```

In [7]:
```
# Merging two data frames
# Creating data frames
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])
```

In [16]:
```
df1
```

Out[16]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [17]:
```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[0, 1, 2, 3])
```

In [18]:
```
df2
```

Out[18]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [19]:
```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8,9,10,11])
```

In [20]:
```
df3
```

Out[20]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [21]:
```
print("\nThe DataFrame number 1\n",'-'*30, sep='')
print(df1)
```

```
The DataFrame number 1
------------------------------
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
```

In [22]:
```
print("\nThe DataFrame number 2\n",'-'*30, sep='')
print(df2)
```

```
The DataFrame number 2
------------------------------
    A   B   C   D
0  A4  B4  C4  D4
1  A5  B5  C5  D5
2  A6  B6  C6  D6
3  A7  B7  C7  D7
```

In [23]:
```
print("\nThe DataFrame number 3\n",'-'*30, sep='')
print(df3)
```

```
The DataFrame number 3
------------------------------
      A    B    C    D
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

In [24]:
```
#concatenation
df_cat1 = pd.concat([df1,df2,df3], axis=0)
print("\nAfter concatenation along row\n",'-'*30, sep='')
print(df_cat1)
df_cat1.loc[2]
```

```
After concatenation along row
------------------------------
      A    B    C    D
0    A0   B0   C0   D0
1    A1   B1   C1   D1
2    A2   B2   C2   D2
3    A3   B3   C3   D3
0    A4   B4   C4   D4
1    A5   B5   C5   D5
2    A6   B6   C6   D6
3    A7   B7   C7   D7
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

Out[24]:

	A	B	C	D
2	A2	B2	C2	D2
2	A6	B6	C6	D6

In [25]:
```
df_cat1.iloc[4]
```

Out[25]:
```
A    A4
B    B4
C    C4
D    D4
Name: 0, dtype: object
```

In [ ]:
```

```

In [26]:
```
df_cat2 = pd.concat([df1,df2,df3], axis=1)
print("\nAfter concatenation along column\n",'-'*60, sep='')
print(df_cat2)
```

```
After concatenation along column
------------------------------------------------------------
      A    B    C    D    A    B    C    D    A    B    C    D
0    A0   B0   C0   D0   A4   B4   C4   D4  NaN  NaN  NaN  NaN
1    A1   B1   C1   D1   A5   B5   C5   D5  NaN  NaN  NaN  NaN
2    A2   B2   C2   D2   A6   B6   C6   D6  NaN  NaN  NaN  NaN
3    A3   B3   C3   D3   A7   B7   C7   D7  NaN  NaN  NaN  NaN
8   NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN   A8   B8   C8   D8
9   NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN   A9   B9   C9   D9
10  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A10  B10  C10  D10
11  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A11  B11  C11  D11
```

In [27]:
```
df_cat2.fillna(value=0, inplace=True)
print("\nAfter filling missing values with zero\n",'-'*60, sep='')
print(df_cat2)
```

```
After filling missing values with zero
------------------------------------------------------------
     A   B   C   D   A   B   C   D    A    B    C    D
0   A0  B0  C0  D0  A4  B4  C4  D4    0    0    0    0
1   A1  B1  C1  D1  A5  B5  C5  D5    0    0    0    0
2   A2  B2  C2  D2  A6  B6  C6  D6    0    0    0    0
3   A3  B3  C3  D3  A7  B7  C7  D7    0    0    0    0
8    0   0   0   0   0   0   0   0   A8   B8   C8   D8
9    0   0   0   0   0   0   0   0   A9   B9   C9   D9
10   0   0   0   0   0   0   0   0  A10  B10  C10  D10
11   0   0   0   0   0   0   0   0  A11  B11  C11  D11
```

In [28]:
```
# merging by a common key
```

In [2]:
```
left = pd.DataFrame({'key': ['K0', 'K8', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                          'C': ['C0', 'C1', 'C2', 'C3'],
                          'D': ['D0', 'D1', 'D2', 'D3']})
```

In [3]:
```
left
```

Out[3]:

	key	A	B
0	K0	A0	B0
1	K8	A1	B1
2	K2	A2	B2
3	K3	A3	B3

In [4]:
```
right
```

Out[4]:

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

In [5]:
```
print("\nThe DataFrame 'left'\n",'-'*30, sep='')
print(left)
```

```
The DataFrame 'left'
------------------------------
  key   A   B
0  K0  A0  B0
1  K8  A1  B1
2  K2  A2  B2
3  K3  A3  B3
```

In [6]:
```
print("\nThe DataFrame 'right'\n",'-'*30, sep='')
print(right)
```

```
The DataFrame 'right'
------------------------------
  key   C   D
0  K0  C0  D0
1  K1  C1  D1
2  K2  C2  D2
3  K3  C3  D3
```

In [7]:
```
merge1= pd.merge(left,right,how='inner',on='key')
print("\nAfter simple merging with 'inner' method\n",'-'*50, sep='')
print(merge1)
```

```
After simple merging with 'inner' method
--------------------------------------------------
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K2  A2  B2  C2  D2
2  K3  A3  B3  C3  D3
```

In [35]:
```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                                'key2': ['K0', 'K0', 'K0', 'K0'],
                                   'C': ['C0', 'C1', 'C2', 'C3'],
                                   'D': ['D0', 'D1', 'D2', 'D3']})
```

In [36]:
```
left
```

Out[36]:

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

In [37]:
```
right
```

Out[37]:

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

In [38]:
```
pd.merge(left, right, on=['key1', 'key2'])
```

Out[38]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

In [39]:
```
pd.merge(left, right, how='left',on=['key1', 'key2'])
```

Out[39]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN

In [40]:
```
pd.merge(left, right, how='right',on=['key1', 'key2'])
```

Out[40]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

In [41]:
```
#join operators
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                    'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

In [42]:
```
left
```

Out[42]:

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

In [43]:
```
right
```

Out[43]:

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

In [44]:
```
left.join(right)
```

Out[44]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

In [45]:
```
left.join(right, how='outer')
```

Out[45]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

In [46]:
```
# use of apply functions
```

In [9]:
```
# Define a function
def testfunc(x):
    if (x> 500):
        return (10*np.log10(x))
    else:
        return (x/10)
```

In [10]:
```
df = pd.DataFrame({'col1':[1,2,3,4,5,6,7,8,9,10],
                   'col2':[444,555,666,444,333,222,666,777,666,555],
                   'col3':'aaa bb c dd eeee fff gg h iii j'.split()})
df
```

Out[10]:

	col1	col2	col3
0	1	444	aaa
1	2	555	bb
2	3	666	c
3	4	444	dd
4	5	333	eeee
5	6	222	fff
6	7	666	gg
7	8	777	h
8	9	666	iii
9	10	555	j

In [11]:
```
df['FuncApplied'] = df['col2'].apply(lambda x : np.log(x))
print(df)
```

```
   col1  col2  col3  FuncApplied
0     1   444   aaa     6.095825
1     2   555    bb     6.318968
2     3   666     c     6.501290
3     4   444    dd     6.095825
4     5   333  eeee     5.808142
5     6   222   fff     5.402677
6     7   666    gg     6.501290
7     8   777     h     6.655440
8     9   666   iii     6.501290
9    10   555     j     6.318968
```

In [12]:
```
df['func'] = df['col2'].apply(testfunc)
print(df)
```

```
   col1  col2  col3  FuncApplied       func
0     1   444   aaa     6.095825  44.400000
1     2   555    bb     6.318968  27.442930
2     3   666     c     6.501290  28.234742
3     4   444    dd     6.095825  44.400000
4     5   333  eeee     5.808142  33.300000
5     6   222   fff     5.402677  22.200000
6     7   666    gg     6.501290  28.234742
7     8   777     h     6.655440  28.904210
8     9   666   iii     6.501290  28.234742
9    10   555     j     6.318968  27.442930
```

In [50]:
```
df['col3length']= df['col3'].apply(len)
print(df)
```

```
   col1  col2  col3  FuncApplied  col3length
0     1   444   aaa     6.095825           3
1     2   555    bb     6.318968           2
2     3   666     c     6.501290           1
3     4   444    dd     6.095825           2
4     5   333  eeee     5.808142           4
5     6   222   fff     5.402677           3
6     7   666    gg     6.501290           2
7     8   777     h     6.655440           1
8     9   666   iii     6.501290           3
9    10   555     j     6.318968           1
```

In [51]:
```
df['FuncApplied'].apply(lambda x: np.sqrt(x))
```

Out[51]:
```
0    2.468972
1    2.513756
2    2.549763
3    2.468972
4    2.410009
5    2.324366
6    2.549763
7    2.579814
8    2.549763
9    2.513756
Name: FuncApplied, dtype: float64
```

In [52]:
```
print("\nSum of the column 'FuncApplied' is: ",df['FuncApplied'].sum())
```

```
Sum of the column 'FuncApplied' is:  62.19971458619886
```

In [53]:
```
print("Mean of the column 'FuncApplied' is: ",df['FuncApplied'].mean())
```

```
Mean of the column 'FuncApplied' is:  6.219971458619886
```

In [54]:
```
print("Std dev of the column 'FuncApplied' is: ",df['FuncApplied'].std())
```

```
Std dev of the column 'FuncApplied' is:  0.3822522801574853
```

In [55]:
```
print("Min and max of the column 'FuncApplied' are: ",df['FuncApplied'].min(),"ar
```

```
Min and max of the column 'FuncApplied' are:  5.402677381872279 and 6.655440350
367647
```

In [56]:
```
### Deletion, sorting, list of column and row names
```

In [57]:
```
print("\nName of columns\n",'-'*20, sep='')
print(df.columns)
```

```
Name of columns
--------------------
Index(['col1', 'col2', 'col3', 'FuncApplied', 'col3length'], dtype='object')
```

In [58]:
```
l = list(df.columns)
print("\nColumn names in a list of strings for later manipulation:",l)
```

```
Column names in a list of strings for later manipulation: ['col1', 'col2', 'col
3', 'FuncApplied', 'col3length']
```

In [59]:
```
print("\nDeleting last column by 'del' command\n",'-'*50, sep='')
del df['col3length']
print(df)
df['col3length']= df['col3'].apply(len)
```

```
Deleting last column by 'del' command
--------------------------------------------------
   col1  col2  col3  FuncApplied
0     1   444   aaa     6.095825
1     2   555    bb     6.318968
2     3   666     c     6.501290
3     4   444    dd     6.095825
4     5   333  eeee     5.808142
5     6   222   fff     5.402677
6     7   666    gg     6.501290
7     8   777     h     6.655440
8     9   666   iii     6.501290
9    10   555     j     6.318968
```

In [60]:
```
df.sort_values(by='col2') #inplace=False by default
```

Out[60]:

	col1	col2	col3	FuncApplied	col3length
5	6	222	fff	5.402677	3
4	5	333	eeee	5.808142	4
0	1	444	aaa	6.095825	3
3	4	444	dd	6.095825	2
1	2	555	bb	6.318968	2
9	10	555	j	6.318968	1
2	3	666	c	6.501290	1
6	7	666	gg	6.501290	2
8	9	666	iii	6.501290	3
7	8	777	h	6.655440	1

In [61]:
```
df.sort_values(by='FuncApplied',ascending=False) #inplace=False by default
```

Out[61]:

	col1	col2	col3	FuncApplied	col3length
7	8	777	h	6.655440	1
2	3	666	c	6.501290	1
6	7	666	gg	6.501290	2
8	9	666	iii	6.501290	3
1	2	555	bb	6.318968	2
9	10	555	j	6.318968	1
0	1	444	aaa	6.095825	3
3	4	444	dd	6.095825	2
4	5	333	eeee	5.808142	4
5	6	222	fff	5.402677	3

In [62]:
```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[None,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

Out[62]:

	col1	col2	col3
0	1.0	NaN	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	NaN	444.0	xyz

In [63]:
```
df.isnull()
```

Out[63]:

	col1	col2	col3
0	False	True	False
1	False	False	False
2	False	False	False
3	True	False	False

In [64]:
```
df.fillna('FILL')
```

Out[64]:

	col1	col2	col3
0	1	FILL	abc
1	2	555	def
2	3	666	ghi
3	FILL	444	xyz

In [65]:
```
df1
```

Out[65]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [66]:
```
df2
```

Out[66]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [67]:
```
df3
```

Out[67]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In [68]:
```
pd.merge(df1, df2, how='inner')
```

Out[68]:

A	B	C	D

In [69]:
```
pd.merge(df1, df2, how='outer')
```

Out[69]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

In [70]:
```
pd.merge(df1, df2, how='left')
```

Out[70]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [71]:
```
pd.merge(df1, df2, how='right')
```

Out[71]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [ ]:
```

```

In [ ]:
```
#911 dataset
```