Numpy

- Numerical Python
- Fastest
- Vectorised Operation
- Contigous Memory

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

In [3]:
```
# creating an array unsing a python list
x = np.array([22,33,44,55])
x
```

Out[3]:
```
array([22, 33, 44, 55])
```

In [4]:
```
x.size
```

Out[4]:
```
4
```

In [5]:
```
# 1d= Vector
# 2d = Matrix
# 3d = Cube
# nd = ....
x.shape
```

Out[5]:
```
(4,)
```

In [6]:
```
x.ndim
```

Out[6]:
```
1
```

In [8]:
```
list(range(10))
```

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

In [9]:
```
arr1d = np.arange(10)
arr1d
```

Out[9]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [10]:
```
arr2d = np.array([[1,2,3],[11,22,33],[111,222,333],[1111,2222,3333]])
```

In [11]:
```
arr2d
```

Out[11]:
```
array([[   1,    2,    3],
       [  11,   22,   33],
       [ 111,  222,  333],
       [1111, 2222, 3333]])
```

In [12]:
```
arr2d.shape
```

Out[12]:
```
(4, 3)
```

In [44]:
```
#arr2d = np.arange(20).reshape(1,20)
#arr2d = np.arange(20).reshape(20,1)
#arr2d = np.arange(20).reshape(4,5)
arr2d = np.arange(20).reshape(5,4)
# arr2d = np.arange(20).reshape(10,2)
# arr2d = np.arange(20).reshape(2,10)
arr2d
```

Out[44]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

In [15]:
```
arr2d.shape
```

Out[15]:
```
(1, 20)
```

In [16]:
```
arr2d.ndim
```

Out[16]:
```
2
```

In [17]:
```
arr2d.reshape(20)
```

Out[17]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [18]:
```
arr2d.flatten()
```

Out[18]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [19]:
```
arr2d.ravel()
```

Out[19]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [30]:
```
# Scaler
arr = np.array(10)
arr
```

Out[30]:
```
array(10)
```

In [31]:
```
arr.ndim
```

Out[31]:
```
0
```

In [33]:
```
#  1dVector
arr = np.array([10])
arr
```

Out[33]:
```
array([10])
```

In [34]:
```
#2dmatrix
arr = np.array([[10]])
arr
```

Out[34]:
```
array([[10]])
```

In [40]:
```
arr3d = np.arange(64).reshape(4,4,4)# (depth, row, colums)
                                    # (matrixnum, row, col)
arr3d
```

Out[40]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]],

       [[16, 17, 18, 19],
        [20, 21, 22, 23],
        [24, 25, 26, 27],
        [28, 29, 30, 31]],

       [[32, 33, 34, 35],
        [36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59],
        [60, 61, 62, 63]]])
```

In [45]:
```
arr2d
```

Out[45]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

In [46]:
```
arr2d[3,2]
```

Out[46]:
```
14
```

In [47]:
```
arr2d[3][2]
```

Out[47]:
```
14
```

In [51]:
```
arr3d[2]
```

Out[51]:
```
array([[32, 33, 34, 35],
       [36, 37, 38, 39],
       [40, 41, 42, 43],
       [44, 45, 46, 47]])
```

In [52]:
```
arr3d
```

Out[52]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]],

       [[16, 17, 18, 19],
        [20, 21, 22, 23],
        [24, 25, 26, 27],
        [28, 29, 30, 31]],

       [[32, 33, 34, 35],
        [36, 37, 38, 39],
        [40, 41, 42, 43],
        [44, 45, 46, 47]],

       [[48, 49, 50, 51],
        [52, 53, 54, 55],
        [56, 57, 58, 59],
        [60, 61, 62, 63]]])
```

In [53]:
```
arr3d[2,1,2]
```

Out[53]:
```
38
```

In [54]:
```
x
```

Out[54]:
```
array([22, 33, 44, 55])
```

In [56]:
```
z = []
for n in x:
    z.append(n*2)
z
```

Out[56]:
```
[44, 66, 88, 110]
```

In [57]:
```
[n**2 for n in x]
```

Out[57]:
```
[484, 1089, 1936, 3025]
```

In [58]:
```
x *2
```

Out[58]:
```
array([ 44,  66,  88, 110])
```

In [59]:
```
arr2d /10
```

Out[59]:
```
array([[0. , 0.1, 0.2, 0.3],
       [0.4, 0.5, 0.6, 0.7],
       [0.8, 0.9, 1. , 1.1],
       [1.2, 1.3, 1.4, 1.5],
       [1.6, 1.7, 1.8, 1.9]])
```

In [61]:
```
arr3d*10
```

Out[61]:
```
array([[[  0,  10,  20,  30],
        [ 40,  50,  60,  70],
        [ 80,  90, 100, 110],
        [120, 130, 140, 150]],

       [[160, 170, 180, 190],
        [200, 210, 220, 230],
        [240, 250, 260, 270],
        [280, 290, 300, 310]],

       [[320, 330, 340, 350],
        [360, 370, 380, 390],
        [400, 410, 420, 430],
        [440, 450, 460, 470]],

       [[480, 490, 500, 510],
        [520, 530, 540, 550],
        [560, 570, 580, 590],
        [600, 610, 620, 630]]])
```

In [62]:
```
arr3d[2,1,2] = arr3d[2,1,2] * 10
```

In [63]:
```
arr3d[2,1,2]
```

Out[63]:
```
380
```

In [64]:
```
x
```

Out[64]:
```
array([22, 33, 44, 55])
```

In [65]:
```
y= np.arange(4)
```

In [66]:
```
y
```

Out[66]:
```
array([0, 1, 2, 3])
```

In [68]:
```
x+y
```

Out[68]:
```
array([22, 34, 46, 58])
```

In [72]:
```
# mirroring = elementwise operation
a = np.arange(25).reshape(5,5)
b = np.arange(25,50).reshape(5,5)
```

In [73]:
```
a
```

Out[73]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [74]:
```
b
```

Out[74]:
```
array([[25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34],
       [35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44],
       [45, 46, 47, 48, 49]])
```

In [75]:
```
# mirror or element wise
a + b
```

Out[75]:
```
array([[25, 27, 29, 31, 33],
       [35, 37, 39, 41, 43],
       [45, 47, 49, 51, 53],
       [55, 57, 59, 61, 63],
       [65, 67, 69, 71, 73]])
```

In [76]:
```
a * b
```

Out[76]:
```
array([[   0,   26,   54,   84,  116],
       [ 150,  186,  224,  264,  306],
       [ 350,  396,  444,  494,  546],
       [ 600,  656,  714,  774,  836],
       [ 900,  966, 1034, 1104, 1176]])
```

In [77]:
```
a@b
# a.dot(b)
```

Out[77]:
```
array([[ 400,  410,  420,  430,  440],
       [1275, 1310, 1345, 1380, 1415],
       [2150, 2210, 2270, 2330, 2390],
       [3025, 3110, 3195, 3280, 3365],
       [3900, 4010, 4120, 4230, 4340]])
```

In [78]:
```
c = np.arange(20).reshape(5,4)
```

In [79]:
```
c
```

Out[79]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

In [80]:
```
#(5,5) and (5,4) #
b@c
```

Out[80]:
```
array([[1120, 1255, 1390, 1525],
       [1320, 1480, 1640, 1800],
       [1520, 1705, 1890, 2075],
       [1720, 1930, 2140, 2350],
       [1920, 2155, 2390, 2625]])
```

In [81]:
```
c@b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-81-98148850ede7> in <module>
----> 1 c@b

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, w
ith gufunc signature (n?,k),(k,m?)->(n?,m?) (size 5 is different from 4)
```

Array Slicing

In [82]:
```
a
```

Out[82]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [87]:
```
(a[2,3])
```

Out[87]:
```
13
```

In [86]:
```
# array [row, col]
#       [start:end   ,   start:end]
```

In [88]:
```
a[2:3, 3:4]
```

Out[88]:
```
array([[13]])
```

In [89]:
```
a[1:4,3]
```

Out[89]:
```
array([ 8, 13, 18])
```

In [91]:
```
a[1:4,3:4]
```

Out[91]:
```
array([[ 8],
       [13],
       [18]])
```

In [93]:
```
a[4, :4]
```

Out[93]:
```
array([20, 21, 22, 23])
```

Querrying Array

In [94]:
```
a
```

Out[94]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [97]:
```
mask= a>5   # Mask
mask
```

Out[97]:
```
array([[False, False, False, False, False],
       [False,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

In [98]:
```
a[mask]
```

Out[98]:
```
array([ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
       23, 24])
```

In [99]:
```
a[a>10]
```

Out[99]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])
```

In [100]:
```
a[a>7] = 100
```

In [101]:
```
a
```

Out[101]:
```
array([[  0,   1,   2,   3,   4],
       [  5,   6,   7, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100]])
```

In [102]:
```
np.where(a>10, 0, a)
```

Out[102]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

In [104]:
```
a[(a%2==0) & (a%7==0)]
```

Out[104]:
```
array([0])
```

In [105]:
```
a
```

Out[105]:
```
array([[  0,   1,   2,   3,   4],
       [  5,   6,   7, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100]])
```

In [106]:
```
a[(a%2==0) | (a%7==0)]
```

Out[106]:
```
array([  0,   2,   4,   6,   7, 100, 100, 100, 100, 100, 100, 100, 100,
       100, 100, 100, 100, 100, 100, 100, 100, 100])
```

In [107]:
```
np.where((a%2==0) | (a%7==0), "Wow", "Oh")
```

Out[107]:
```
array([['Wow', 'Oh', 'Wow', 'Oh', 'Wow'],
       ['Oh', 'Wow', 'Wow', 'Wow', 'Wow'],
       ['Wow', 'Wow', 'Wow', 'Wow', 'Wow'],
       ['Wow', 'Wow', 'Wow', 'Wow', 'Wow'],
       ['Wow', 'Wow', 'Wow', 'Wow', 'Wow']], dtype='<U3')
```

Combinig Arrays

In [111]:
```
print(a)
a.shape
```

```
[[  0   1   2   3   4]
 [  5   6   7 100 100]
 [100 100 100 100 100]
 [100 100 100 100 100]
 [100 100 100 100 100]]
```

Out[111]: (5, 5)

In [112]:
```
print(b)
b.shape
```

```
[[25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]
```

Out[112]: (5, 5)

In [115]:
```
np.concatenate((a,b),axis=1)
```

Out[115]:
```
array([[  0,   1,   2,   3,   4,  25,  26,  27,  28,  29],
       [  5,   6,   7, 100, 100,  30,  31,  32,  33,  34],
       [100, 100, 100, 100, 100,  35,  36,  37,  38,  39],
       [100, 100, 100, 100, 100,  40,  41,  42,  43,  44],
       [100, 100, 100, 100, 100,  45,  46,  47,  48,  49]])
```

In [116]:
```
np.concatenate((a,b),axis=0)
```

Out[116]:
```
array([[  0,   1,   2,   3,   4],
       [  5,   6,   7, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [ 25,  26,  27,  28,  29],
       [ 30,  31,  32,  33,  34],
       [ 35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44],
       [ 45,  46,  47,  48,  49]])
```

In [117]:
```
np.stack((a,b))
```

Out[117]:
```
array([[[  0,   1,   2,   3,   4],
        [  5,   6,   7, 100, 100],
        [100, 100, 100, 100, 100],
        [100, 100, 100, 100, 100],
        [100, 100, 100, 100, 100]],

       [[ 25,  26,  27,  28,  29],
        [ 30,  31,  32,  33,  34],
        [ 35,  36,  37,  38,  39],
        [ 40,  41,  42,  43,  44],
        [ 45,  46,  47,  48,  49]]])
```

In [118]:
```
np.hstack((a,b))
```

Out[118]:
```
array([[  0,   1,   2,   3,   4,  25,  26,  27,  28,  29],
       [  5,   6,   7, 100, 100,  30,  31,  32,  33,  34],
       [100, 100, 100, 100, 100,  35,  36,  37,  38,  39],
       [100, 100, 100, 100, 100,  40,  41,  42,  43,  44],
       [100, 100, 100, 100, 100,  45,  46,  47,  48,  49]])
```

In [119]:
```
np.vstack((a,b))
```

Out[119]:
```
array([[  0,   1,   2,   3,   4],
       [  5,   6,   7, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100],
       [ 25,  26,  27,  28,  29],
       [ 30,  31,  32,  33,  34],
       [ 35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44],
       [ 45,  46,  47,  48,  49]])
```

In [ ]:

Pandas

- Data analysis library
- Series Single Dimension
- Dataframe Multidimensional

In [5]:
```
import pandas as pd
```

Series

In [7]:
```
s1 = pd.Series([1,2,3,4])
```

In [8]:
```
type(s1)
```

Out[8]:
```
pandas.core.series.Series
```

In [9]:
```
print(s1)
```

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

In [10]:
```
s1.index
```

Out[10]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [11]:
```
s1.values
```

Out[11]:
```
array([1, 2, 3, 4], dtype=int64)
```

In [12]:
```
s1.index = ["one", 'two','three','four']
```

In [13]:
```
s1
```

Out[13]:
```
one      1
two      2
three    3
four     4
dtype: int64
```

In [14]:
```
s1['one']
```

Out[14]:
```
1
```

In [15]:
```
s1.index
```

Out[15]:
```
Index(['one', 'two', 'three', 'four'], dtype='object')
```

In [16]:
```
s1[0]
```

Out[16]:
```
1
```

In [27]:
```
s2= pd.Series([22,33,44,55,100], index=['apples', 'oranges','bananas','gauvas'
s2
```

Out[27]:
```
apples      22
oranges     33
bananas     44
gauvas      55
two        100
dtype: int64
```

In [28]:
```
s1  = s1*10
```

In [29]:
```
s1
```

Out[29]:
```
one      100
two      200
three    300
four     400
dtype: int64
```

In [30]:
```
s2
```

Out[30]:
```
apples      22
oranges     33
bananas     44
gauvas      55
two        100
dtype: int64
```

In [31]:
```
s1  + s2
```

Out[31]:
```
apples       NaN
bananas      NaN
four         NaN
gauvas       NaN
one          NaN
oranges      NaN
three        NaN
two        300.0
dtype: float64
```

In [32]:
```
s3 = pd.Series(range(10))
s3
```

Out[32]:
```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
dtype: int64
```

In [35]:
```
s4 = pd.Series(range(10,21))
s4
```

Out[35]:
```
0     10
1     11
2     12
3     13
4     14
5     15
6     16
7     17
8     18
9     19
10    20
dtype: int64
```

In [36]:
```
s3+ s4
```

Out[36]:
```
0     10.0
1     12.0
2     14.0
3     16.0
4     18.0
5     20.0
6     22.0
7     24.0
8     26.0
9     28.0
10     NaN
dtype: float64
```

In [38]:
```
import numpy as np
arr1 = np.arange(5)
arr1
```

Out[38]:
```
array([0, 1, 2, 3, 4])
```

In [39]:
```
arr2 = np.arange(6)
```

In [40]:
```
arr1+arr2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-e489ba1ad4d1> in <module>
----> 1 arr1+arr2

ValueError: operands could not be broadcast together with shapes (5,) (6,)
```

In [44]:
```
s2
```

Out[44]:
```
apples      22
oranges     33
bananas     44
gauvas      55
two        100
dtype: int64
```

In [45]:
```
s2[2:4]
```

Out[45]:
```
bananas    44
gauvas     55
dtype: int64
```

In [46]:
```
s2['oranges':'gauvas']
```

Out[46]:
```
oranges    33
bananas    44
gauvas     55
dtype: int64
```

In [47]:
```
s2[2:4] = 23
```

In [48]:
```
s2
```

Out[48]:
```
apples      22
oranges     33
bananas     23
gauvas      23
two        100
dtype: int64
```

Lambda Functions : Anonymous Functions (IIF)

In [75]:
```
new = pd.Series([99, 100, 89, 50 ,45,30], index=['Ahmed', 'Ali', 'Faisal', 'Ha
```

In [76]:
```
new
```

Out[76]:
```
Ahmed      99
Ali       100
Faisal     89
Hassan     50
Aqsa       45
Waji       30
dtype: int64
```

In [53]:
```
def myfunc(marks):
    return marks/100
```

In [72]:
```
new = new.apply(myfunc)
```

In [81]:
```
new = new.apply(lambda x: np.where(x>40, "Pass", "Fail"))
```

In [58]:
```
s1
```

Out[58]:
```
one      100
two      200
three    300
four     400
dtype: int64
```

In [59]:
```
s2
```

Out[59]:
```
apples      22
oranges     33
bananas     23
gauvas      23
two        100
dtype: int64
```

In [60]:
```
s1 + s2
```

Out[60]:
```
apples       NaN
bananas      NaN
four         NaN
gauvas       NaN
one          NaN
oranges      NaN
three        NaN
two        300.0
dtype: float64
```

In [61]:
```
s2.index
```

Out[61]:
```
Index(['apples', 'oranges', 'bananas', 'gauvas', 'two'], dtype='object')
```

In [64]:
```
s5 = pd.Series([1,2,3,4,5],index=s2.index)
s5
```

Out[64]:
```
apples     1
oranges    2
bananas    3
gauvas     4
two        5
dtype: int64
```

In [67]:
```
s6 = pd.Series(100,index=s2.index)
s6
```

Out[67]:
```
apples     100
oranges    100
bananas    100
gauvas     100
two        100
dtype: int64
```

In [70]:
```
np.linspace(0,10,10)
```

Out[70]:
```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

In [78]:
```
new.count()
```

Out[78]:
```
6
```

In [82]:
```
new.value_counts()
```

Out[82]:
```
Pass    5
Fail    1
dtype: int64
```

In [85]:
```
new[['Ahmed','Ali' ]]
```

Out[85]:
```
Ahmed    Pass
Ali      Pass
dtype: object
```

In [88]:
```
s7 = pd.Series([1,2,3,np.nan, np.nan])
s7
```

Out[88]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
dtype: float64
```

In [107]:
```
np.random.seed(1)
s1 = pd.Series(np.random.randn(3))
s2 = pd.Series(np.random.randn(3))
```

In [108]:
```
s1
```

Out[108]:
```
0    1.624345
1   -0.611756
2   -0.528172
dtype: float64
```

In [91]:
```
s2
```

Out[91]:
```
0   -1.135632
1    1.212112
2   -0.173215
dtype: float64
```

In [109]:
```
combined = pd.concat([s1, s2])
combined
```

Out[109]:
```
0    1.624345
1   -0.611756
2   -0.528172
0   -1.072969
1    0.865408
2   -2.301539
dtype: float64
```

In [115]:
```
combined.reset_index(drop=True,inplace=True)
```

In [116]:
```
combined
```

Out[116]:
```
0    1.624345
1   -0.611756
2   -0.528172
3   -1.072969
4    0.865408
5   -2.301539
dtype: float64
```

In [117]:
```
combined[4] = 2345
combined
```

Out[117]:
```
0       1.624345
1      -0.611756
2      -0.528172
3      -1.072969
4    2345.000000
5      -2.301539
dtype: float64
```

In [118]:
```
del combined[4]
```

In [119]:
```
combined
```

Out[119]:
```
0    1.624345
1   -0.611756
2   -0.528172
3   -1.072969
5   -2.301539
dtype: float64
```

In [125]:
```
del combined[3]
```

In [128]:
```
combined[1:5] =100
combined
```

Out[128]:
```
0      1.624345
1    100.000000
2    100.000000
5    100.000000
dtype: float64
```

In [130]:
```
newCom = combined.copy()   # copy by value
newCom = combined[:,:]
```

In [132]:
```
oldCom  = newCom   # copy by refernce
```

In [134]:
```
oldCom
```

Out[134]:
```
0      1.624345
1    100.000000
2    100.000000
5    100.000000
dtype: float64
```

Pandas Data Frame

In [140]:
```
students = {'rolls':[1,2,3,4,5,6],
            'names':['Ali','Amjad', 'Saima', 'Qaiser','Hamid','Nasir'],
            'courses':['Python', 'Pandas', 'Numpy','Pandas', 'Numpy', "Python"
            'mode':['Online', 'Onsite','Online', 'Onsite','Online', 'Onsite']}
```

In [145]:
```
# Creating a data frame
df  = pd.DataFrame(students)
df
```

Out[145]:

	rolls	names	courses	mode
0	1	Ali	Python	Online
1	2	Amjad	Pandas	Onsite
2	3	Saima	Numpy	Online
3	4	Qaiser	Pandas	Onsite
4	5	Hamid	Numpy	Online
5	6	Nasir	Python	Onsite

In [148]:
```
type(df[['rolls', 'names']])
```

Out[148]:
```
pandas.core.frame.DataFrame
```

In [150]:
```
type(df['rolls'])
```

Out[150]:
```
pandas.core.series.Series
```

In [151]:
```
type(df[['rolls']])
```

Out[151]:
```
pandas.core.frame.DataFrame
```

In [155]:
```
np.array([[9]]).ndim
```

Out[155]:
```
2
```

In [166]:
```
lst=[[55,66,77],[2,4,5],[4,5,6]]
```

In [167]:
```
lst
```

Out[167]:
```
[[55, 66, 77], [2, 4, 5], [4, 5, 6]]
```

In [168]:
```
pd.DataFrame(lst,index=[1,2,3],columns=["a","b","c"] )
```

Out[168]:

	a	b	c
1	55	66	77
2	2	4	5
3	4	5	6

In [169]:
```
#Accessing data
# Columns
```

In [171]:
```
df['courses']
```

Out[171]:
```
0    Python
1    Pandas
2     Numpy
3    Pandas
4     Numpy
5    Python
Name: courses, dtype: object
```

In [173]:
```
df[['courses', 'names']]
```

Out[173]:

	courses	names
0	Python	Ali
1	Pandas	Amjad
2	Numpy	Saima
3	Pandas	Qaiser
4	Numpy	Hamid
5	Python	Nasir

In [174]:
```
df['Country'] = "Pakistan"
df
```

Out[174]:

	rolls	names	courses	mode	Country
0	1	Ali	Python	Online	Pakistan
1	2	Amjad	Pandas	Onsite	Pakistan
2	3	Saima	Numpy	Online	Pakistan
3	4	Qaiser	Pandas	Onsite	Pakistan
4	5	Hamid	Numpy	Online	Pakistan
5	6	Nasir	Python	Onsite	Pakistan

In [175]:
```
df['Ages'] = [23,34,23,21,20,19]
df
```

Out[175]:

	rolls	names	courses	mode	Country	Ages
0	1	Ali	Python	Online	Pakistan	23
1	2	Amjad	Pandas	Onsite	Pakistan	34
2	3	Saima	Numpy	Online	Pakistan	23
3	4	Qaiser	Pandas	Onsite	Pakistan	21
4	5	Hamid	Numpy	Online	Pakistan	20
5	6	Nasir	Python	Onsite	Pakistan	19

In [177]:
```
dic = {'Ali':34, 'Amjad':45, "Saima":23, "Hamid":34, "Nasir":20,'Qaiser':45}
```

In [208]:
```
df['newAges'] = df.names.map(dic)
```

In [183]:
```
df['Status'] = ["Pass" if age>25 else "Fails" for age in df['Ages']]
```

In [209]:
```
df
```

Out[209]:

	rolls	names	courses	mode	Country	Ages	Status	City	Generation	Generation Next	newA
0	1	Ali	Python	Online	Pakistan	23	Fails	Outside Khi	Young	Young	
1	2	Amjad	Pandas	Onsite	Pakistan	34	Pass	Karachi	Young	Young	
2	3	Saima	Numpy	Online	Pakistan	23	Fails	Outside Khi	Young	Young	
3	4	Qaiser	Pandas	Onsite	Pakistan	21	Fails	Karachi	Young	Young	
4	5	Hamid	Numpy	Online	Pakistan	20	Fails	Outside Khi	Young	Young	
5	6	Nasir	Python	Onsite	Pakistan	19	Fails	Karachi	Teen Ager	Teen Ager	

In [190]:
```
df['City'] = df['mode'].apply(lambda x: 'Karachi' if x=="Onsite" else "Outside
```

In [191]:
```
df
```

Out[191]:

	rolls	names	courses	mode	Country	Ages	Status	City
0	1	Ali	Python	Online	Pakistan	23	Fails	Outside Khi
1	2	Amjad	Pandas	Onsite	Pakistan	34	Pass	Karachi
2	3	Saima	Numpy	Online	Pakistan	23	Fails	Outside Khi
3	4	Qaiser	Pandas	Onsite	Pakistan	21	Fails	Karachi
4	5	Hamid	Numpy	Online	Pakistan	20	Fails	Outside Khi
5	6	Nasir	Python	Onsite	Pakistan	19	Fails	Karachi

In [193]:
```
def generationAssign(age):
    if age <20:
        return "Teen Ager"
    elif age<40:
        return "Young"
    elif age <60:
        return "Matured"
    else:
        return 'Senior'
```

In [201]:
```
df['Generation Next'] = df.Ages.apply(generationAssign)
```

In [202]:
```
df
```

Out[202]:

	rolls	names	courses	mode	Country	Ages	Status	City	Generation	Generation Next
0	1	Ali	Python	Online	Pakistan	23	Fails	Outside Khi	Young	Young
1	2	Amjad	Pandas	Onsite	Pakistan	34	Pass	Karachi	Young	Young
2	3	Saima	Numpy	Online	Pakistan	23	Fails	Outside Khi	Young	Young
3	4	Qaiser	Pandas	Onsite	Pakistan	21	Fails	Karachi	Young	Young
4	5	Hamid	Numpy	Online	Pakistan	20	Fails	Outside Khi	Young	Young
5	6	Nasir	Python	Onsite	Pakistan	19	Fails	Karachi	Teen Ager	Teen Ager

In [196]:
```
df.to_excel("student.xlsx")
```

In [197]:
```
a =np.array([1,2,3])
```

In [198]:
```
a
```

Out[198]:
```
array([1, 2, 3])
```

In [199]:
```
a = a*10
```

Out[199]:
```
array([10, 20, 30])
```

In [200]:
```
a
```

Out[200]:
```
array([1, 2, 3])
```

In [205]:
```
df['Generation Next']
```

Out[205]:
```
0        Young
1        Young
2        Young
3        Young
4        Young
5    Teen Ager
Name: Generation Next, dtype: object
```

In [210]:
```
data = pd.read_csv("astronauts.csv")
```

In [211]:
```
data
```

Out[211]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Undergraduat Majc
0	Joseph M. Acaba	2004.0	19.0	Active	5/17/1967	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	Geolog
1	Loren W. Acton	NaN	NaN	Retired	3/7/1936	Lewiston, MT	Male	Montana State University; University of Colorado	Engineerir Physic
2	James C. Adamson	1984.0	10.0	Retired	3/3/1946	Warsaw, NY	Male	US Military Academy; Princeton University	Engineerir
3	Thomas D. Akers	1987.0	12.0	Retired	5/20/1951	St. Louis, MO	Male	University of Missouri-Rolla	Applie Mathematic
4	Buzz Aldrin	1963.0	3.0	Retired	1/20/1930	Montclair, NJ	Male	US Military Academy; MIT	Mechanic Engineerir
...	...	...	...	...	...	...	...	...	
352	David A. Wolf	1990.0	13.0	Retired	8/23/1956	Indianapolis, IN	Male	Purdue University; Indiana University	Electric Engineerir
353	Neil W. Woodward III	1998.0	17.0	Retired	7/26/1962	Chicago, IL	Male	MIT; University of Texas-Austin; George Washin...	Physic
354	Alfred M. Worden	1966.0	5.0	Retired	2/7/1932	Jackson, MI	Male	US Military Academy; University of Michigan	Military Scienc
355	John W. Young	1962.0	2.0	Retired	9/24/1930	San Francisco, CA	Male	Georgia Institute of Technology	Aeronautic Engineerir
356	George D. Zamka	1998.0	17.0	Retired	6/29/1962	Jersey City, NJ	Male	US Naval Academy; Florida Institute of Technology	Mathematic

357 rows × 19 columns

In [ ]:

In [4]:
```
import pandas as pd
import numpy as np
```

In [151]:
```
try:
    data = pd.read_csv("astronauts.csv")
    print("dataloaded")
except:
    print("File missing")
```

```
dataloaded
```

In [51]:
```
data.shape
```

Out[51]:
```
(357, 19)
```

In [52]:
```
data.ndim
```

Out[52]:
```
2
```

In [53]:
```
data.head(1)
```

Out[53]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Undergraduate Major	Grad M
0	Joseph M. Acaba	2004.0	19.0	Active	5/17/1967	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	Geology	Geo

In [54]:
```
data.tail(3)
```

Out[54]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Undergraduate Major	
354	Alfred M. Worden	1966.0	5.0	Retired	2/7/1932	Jackson, MI	Male	US Military Academy; University of Michigan	Military Science	A A
355	John W. Young	1962.0	2.0	Retired	9/24/1930	San Francisco, CA	Male	Georgia Institute of Technology	Aeronautical Engineering	
356	George D. Zamka	1998.0	17.0	Retired	6/29/1962	Jersey City, NJ	Male	US Naval Academy; Florida Institute of Technology	Mathematics	M

In [138]:
```
data.sample(10)
```

Out[138]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Underg
230	Leland D. Melvin	1998.0	17.0	Management	2/15/1964	Lynchburg, VA	Male	University of Richmond; University of Virginia	C
169	Douglas G. Hurley	2000.0	18.0	Active	10/21/1966	Endicott, NY	Male	Tulane University	Eng
352	David A. Wolf	1990.0	13.0	Retired	8/23/1956	Indianapolis, IN	Male	Purdue University; Indiana University	Eng
320	William E. Thornton	1967.0	6.0	Retired	4/14/1929	Faison, NC	Male	University of North Carolina	
76	Eileen M. Collins	1990.0	13.0	Retired	11/19/1959	Elmira, NY	Female	Syracuse University; Stanford University; Webs...	Mathe Ec
222	William C. McCool	1996.0	16.0	Deceased	9/23/1961	San Diego, CA	Male	US Naval Academy; University of Maryland; US N...	Naval
162	Kathryn P. Hire	1995.0	15.0	Management	8/26/1959	Mobile, AL	Female	US Naval Academy; Florida State Institute of T...	Eng Man
21	Michael R. Barratt	2000.0	18.0	Active	4/16/1959	Vancouver, WA	Male	University of Washington; Northwestern Univers...	
183	Scott J. Kelly	1996.0	16.0	Active	2/21/1964	Orange, NJ	Male	State University of New York Maritime College;...	Eng
155	Thomas J. Hennen	NaN	NaN	Retired	8/17/1952	Albany, GA	Male	NaN	

In [139]:
```
data.columns
```

Out[139]:
```
Index(['Name', 'Year', 'Group', 'Status', 'Birth Date', 'Birth Place',
       'Gender', 'Alma Mater', 'Undergraduate Major', 'Graduate Major',
       'Military Rank', 'Military Branch', 'Space Flights',
       'Space Flight (hr)', 'Space Walks', 'Space Walks (hr)', 'Missions',
       'Death Date', 'Death Mission'],
      dtype='object')
```

In [161]:
```
# changing column name

data.columns = [11,22,33,44,55,66,77,88,99,100,111,222,333,444,555,666,777,888
data.columns = ['Name', 'Year', 'Group', 'Status', 'Birth Date', 'Birth Place'
       'Gender', 'Alma Mater', 'Undergraduate Major', 'Graduate Major',
       'Military Rank', 'Military Branch', 'Space Flights',
       'Space Flight (hr)', 'Space Walks', 'Space Walks (hr)', 'Missions',
       'Death Date', 'Death Mission']
```

In [162]:
```
data.head(1)
```

Out[162]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Undergraduate Major	Grad M
0	Joseph M. Acaba	2004.0	19.0	Active	5/17/1967	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	Geology	Geo

In [163]:
```
data.rename(columns={'Birth Date':'birth_date', 'Birth Place':'birth_place',
                     'Alma Mater':'alma_mater', 'Undergraduate Major':'undergrad
       'Military Rank':'military_rank', 'Military Branch':'military_branch', '
       'Space Flight (hr)':'space_flight(hr)', 'Space Walks':'space_walks', 'S
       'Death Date':"death_date",'Space Walks (hr)':'space_walks(hr)', 'Death
```

In [164]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 357 entries, 0 to 356
Data columns (total 19 columns):
Name                    357 non-null object
Year                    331 non-null float64
Group                   330 non-null float64
Status                  357 non-null object
birth_date              357 non-null object
birth_place             357 non-null object
Gender                  357 non-null object
alma_mater              356 non-null object
undergraduate_major     335 non-null object
graduate_major          298 non-null object
military_rank           207 non-null object
military_branch         211 non-null object
space_fights            357 non-null int64
space_flight(hr)        357 non-null int64
space_walks             357 non-null int64
space_walks(hr)         357 non-null float64
Missions                334 non-null object
death_date              52 non-null object
death_mission           16 non-null object
dtypes: float64(3), int64(3), object(13)
memory usage: 53.1+ KB
```

In [165]:
```
data.notnull().sum()
data.isnull().sum()
```

Out[165]:
```
Name                    0
Year                   26
Group                  27
Status                  0
birth_date              0
birth_place             0
Gender                  0
alma_mater              1
undergraduate_major    22
graduate_major         59
military_rank         150
military_branch       146
space_fights            0
space_flight(hr)        0
space_walks             0
space_walks(hr)         0
Missions               23
death_date            305
death_mission         341
dtype: int64
```

In [166]:
```
data.birth_date = pd.to_datetime(data.birth_date)
```

In [167]:
```
data.birth_date
```

Out[167]:
```
0     1967-05-17
1     1936-03-07
2     1946-03-03
3     1951-05-20
4     1930-01-20
         ...
352   1956-08-23
353   1962-07-26
354   1932-02-07
355   1930-09-24
356   1962-06-29
Name: birth_date, Length: 357, dtype: datetime64[ns]
```

In [168]:
```
data.death_date = pd.to_datetime(data.death_date)
```

In [169]:
```
data.head(1)
```

Out[169]:

	Name	Year	Group	Status	birth_date	birth_place	Gender	alma_mater	undergraduate_ma
0	Joseph M. Acaba	2004.0	19.0	Active	1967-05-17	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	Geol

In [170]:
```
# we will deal you later
#data.Year.astype('int32')
# data.Group.astype('int32')
```

In [171]:
```
data.Status.value_counts()
```

Out[171]:
```
Retired       220
Deceased       51
Active         50
Management     36
Name: Status, dtype: int64
```

In [172]:
```
data.alma_mater.value_counts().sum()
```

Out[172]:
```
356
```

In [173]:
```
#data.alma_mater.unique()
data.alma_mater.nunique()
```

Out[173]:
```
280
```

In [174]:
```
pd.Series([11,2,np.nan, 33,33,55,6,55,np.nan]).unique()
```

Out[174]:
```
array([11.,  2., nan, 33., 55.,  6.])
```

In [175]:
```
pd.Series([11,2,np.nan, 33,33,55,6,55,np.nan]).nunique()
```

Out[175]:
```
5
```

In [176]:
```
data.Year
```

Out[176]:
```
0      2004.0
1      1996.0
2      1984.0
3      1987.0
4      1963.0
        ...
352    1990.0
353    1998.0
354    1966.0
355    1962.0
356    1998.0
Name: Year, Length: 357, dtype: float64
```

In [177]:
```
data[data.Year.notnull()]
```

Out[177]:

	Name	Year	Group	Status	birth_date	birth_place	Gender	alma_mater	undergradu
0	Joseph M. Acaba	2004.0	19.0	Active	1967-05-17	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	
1	Loren W. Acton	1996.0	NaN	Retired	1936-03-07	Lewiston, MT	Male	Montana State University; University of Colorado	Engineeri
2	James C. Adamson	1984.0	10.0	Retired	1946-03-03	Warsaw, NY	Male	US Military Academy; Princeton University	E
3	Thomas D. Akers	1987.0	12.0	Retired	1951-05-20	St. Louis, MO	Male	University of Missouri-Rolla	Applied M
4	Buzz Aldrin	1963.0	3.0	Retired	1930-01-20	Montclair, NJ	Male	US Military Academy; MIT	M E
...	...	...	...	...	...	...	...	...	
352	David A. Wolf	1990.0	13.0	Retired	1956-08-23	Indianapolis, IN	Male	Purdue University; Indiana University	Electrical E
353	Neil W. Woodward III	1998.0	17.0	Retired	1962-07-26	Chicago, IL	Male	MIT; University of Texas-Austin; George Washin...	
354	Alfred M. Worden	1966.0	5.0	Retired	1932-02-07	Jackson, MI	Male	US Military Academy; University of Michigan	Milita
355	John W. Young	1962.0	2.0	Retired	1930-09-24	San Francisco, CA	Male	Georgia Institute of Technology	Ae E
356	George D. Zamka	1998.0	17.0	Retired	1962-06-29	Jersey City, NJ	Male	US Naval Academy; Florida Institute of Technology	M

331 rows × 19 columns

Handling Missing Values

- filling missing values
- dropping missing values

In [178]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 357 entries, 0 to 356
Data columns (total 19 columns):
Name                   357 non-null object
Year                   331 non-null float64
Group                  330 non-null float64
Status                 357 non-null object
birth_date             357 non-null datetime64[ns]
birth_place            357 non-null object
Gender                 357 non-null object
alma_mater             356 non-null object
undergraduate_major    335 non-null object
graduate_major         298 non-null object
military_rank          207 non-null object
military_branch        211 non-null object
space_fights           357 non-null int64
space_flight(hr)       357 non-null int64
space_walks            357 non-null int64
space_walks(hr)        357 non-null float64
Missions               334 non-null object
death_date             52 non-null datetime64[ns]
death_mission          16 non-null object
dtypes: datetime64[ns](2), float64(3), int64(3), object(11)
memory usage: 53.1+ KB
```

In [179]:
```
data.Year.isnull().sum()
```

Out[179]:
```
26
```

In [180]:
```
data.Year.fillna(data.Year.mode(),inplace=True)
```

In [181]:
```
data.Year
```

Out[181]:
```
0      2004.0
1      1996.0
2      1984.0
3      1987.0
4      1963.0
        ...
352    1990.0
353    1998.0
354    1966.0
355    1962.0
356    1998.0
Name: Year, Length: 357, dtype: float64
```

In [182]:
```
data.Year.isnull().sum()
```

Out[182]: 26

In [183]:
```
data.fillna({
            'Group':data.Group.mode(),
            'alma_mater':data.alma_mater.mode(),
            'undergraduate_major':data.undergraduate_major.mode(),
            'graduate_major':data.graduate_major.mode(),
            })
```

Out[183]:

	Name	Year	Group	Status	birth_date	birth_place	Gender	alma_mater	undergradu
0	Joseph M. Acaba	2004.0	19.0	Active	1967-05-17	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	
1	Loren W. Acton	1996.0	16.0	Retired	1936-03-07	Lewiston, MT	Male	Montana State University; University of Colorado	Engineeri
2	James C. Adamson	1984.0	10.0	Retired	1946-03-03	Warsaw, NY	Male	US Military Academy; Princeton University	E
3	Thomas D. Akers	1987.0	12.0	Retired	1951-05-20	St. Louis, MO	Male	University of Missouri-Rolla	Applied M
4	Buzz Aldrin	1963.0	3.0	Retired	1930-01-20	Montclair, NJ	Male	US Military Academy; MIT	M E
...	...	...	...	...	...	...	...	...	
352	David A. Wolf	1990.0	13.0	Retired	1956-08-23	Indianapolis, IN	Male	Purdue University; Indiana University	Electrical E
353	Neil W. Woodward III	1998.0	17.0	Retired	1962-07-26	Chicago, IL	Male	MIT; University of Texas-Austin; George Washin...	
354	Alfred M. Worden	1966.0	5.0	Retired	1932-02-07	Jackson, MI	Male	US Military Academy; University of Michigan	Milita
355	John W. Young	1962.0	2.0	Retired	1930-09-24	San Francisco, CA	Male	Georgia Institute of Technology	A E
356	George D. Zamka	1998.0	17.0	Retired	1962-06-29	Jersey City, NJ	Male	US Naval Academy; Florida Institute of Technology	M

357 rows × 19 columns

In [160]:
```
data
```

Out[160]:

	Name	Year	Group	Status	Birth Date	Birth Place	Gender	Alma Mater	Undergraduat Majc
0	Joseph M. Acaba	2004.0	19.0	Active	5/17/1967	Inglewood, CA	Male	University of California-Santa Barbara; Univer...	Geolog
1	Loren W. Acton	1996.0	NaN	Retired	3/7/1936	Lewiston, MT	Male	Montana State University; University of Colorado	Engineerir Physic
2	James C. Adamson	1984.0	10.0	Retired	3/3/1946	Warsaw, NY	Male	US Military Academy; Princeton University	Engineerir
3	Thomas D. Akers	1987.0	12.0	Retired	5/20/1951	St. Louis, MO	Male	University of Missouri-Rolla	Applie Mathematic
4	Buzz Aldrin	1963.0	3.0	Retired	1/20/1930	Montclair, NJ	Male	US Military Academy; MIT	Mechanic Engineerir
...	...	...	...	...	...	...	...	...	
352	David A. Wolf	1990.0	13.0	Retired	8/23/1956	Indianapolis, IN	Male	Purdue University; Indiana University	Electric Engineerir
353	Neil W. Woodward III	1998.0	17.0	Retired	7/26/1962	Chicago, IL	Male	MIT; University of Texas-Austin; George Washin...	Physic
354	Alfred M. Worden	1966.0	5.0	Retired	2/7/1932	Jackson, MI	Male	US Military Academy; University of Michigan	Military Scienc
355	John W. Young	1962.0	2.0	Retired	9/24/1930	San Francisco, CA	Male	Georgia Institute of Technology	Aeronautic Engineerir
356	George D. Zamka	1998.0	17.0	Retired	6/29/1962	Jersey City, NJ	Male	US Naval Academy; Florida Institute of Technology	Mathematic

357 rows × 19 columns

In [ ]:
```
data.fillna()
```

In [190]:
```
df = pd.read_excel("grouping.xlsx")
df
```

Out[190]:

	ID	Names	Grades	Depts	Salaries
0	1	Asad	16	Accounts	NaN
1	2	Fahad	17	Taxation	123.0
2	3	NaN	18	Accounts	121.0
3	4	Afzal	19	NaN	NaN
4	5	Yasir	16	Information	99.0
5	6	Nasir	19	NaN	100.0
6	7	NaN	14	Accounts	123.0
7	8	Hassan	17	Information	122.0
8	9	Jami	17	Defence	156.0
9	10	Haseb	18	NaN	160.0

In [192]:
```
df.dropna(axis=0,how='any')
```

Out[192]:

	ID	Names	Grades	Depts	Salaries
1	2	Fahad	17	Taxation	123.0
4	5	Yasir	16	Information	99.0
7	8	Hassan	17	Information	122.0
8	9	Jami	17	Defence	156.0

In [193]:
```
df.dropna(axis=1,how='any')
```

Out[193]:

	ID	Grades
0	1	16
1	2	17
2	3	18
3	4	19
4	5	16
5	6	19
6	7	14
7	8	17
8	9	17
9	10	18

In [194]:
```
df.dropna(axis=0,how="all")
```

Out[194]:

	ID	Names	Grades	Depts	Salaries
0	1	Asad	16	Accounts	NaN
1	2	Fahad	17	Taxation	123.0
2	3	NaN	18	Accounts	121.0
3	4	Afzal	19	NaN	NaN
4	5	Yasir	16	Information	99.0
5	6	Nasir	19	NaN	100.0
6	7	NaN	14	Accounts	123.0
7	8	Hassan	17	Information	122.0
8	9	Jami	17	Defence	156.0
9	10	Haseb	18	NaN	160.0

In [195]:
```
df.dropna(axis=1,how="all")
```

Out[195]:

	ID	Names	Grades	Depts	Salaries
0	1	Asad	16	Accounts	NaN
1	2	Fahad	17	Taxation	123.0
2	3	NaN	18	Accounts	121.0
3	4	Afzal	19	NaN	NaN
4	5	Yasir	16	Information	99.0
5	6	Nasir	19	NaN	100.0
6	7	NaN	14	Accounts	123.0
7	8	Hassan	17	Information	122.0
8	9	Jami	17	Defence	156.0
9	10	Haseb	18	NaN	160.0

In [196]:
```
df['Na_Vals'] = np.nan
```

In [199]:
```
df.iloc[9:,:]=np.nan
```

In [200]:
```
df
```

Out[200]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
0	1.0	Asad	16.0	Accounts	NaN	NaN
1	2.0	Fahad	17.0	Taxation	123.0	NaN
2	3.0	NaN	18.0	Accounts	121.0	NaN
3	4.0	Afzal	19.0	NaN	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN
5	6.0	Nasir	19.0	NaN	100.0	NaN
6	7.0	NaN	14.0	Accounts	123.0	NaN
7	8.0	Hassan	17.0	Information	122.0	NaN
8	9.0	Jami	17.0	Defence	156.0	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN

In [201]:
```
df.dropna(axis=0, how='all')
```

Out[201]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
0	1.0	Asad	16.0	Accounts	NaN	NaN
1	2.0	Fahad	17.0	Taxation	123.0	NaN
2	3.0	NaN	18.0	Accounts	121.0	NaN
3	4.0	Afzal	19.0	NaN	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN
5	6.0	Nasir	19.0	NaN	100.0	NaN
6	7.0	NaN	14.0	Accounts	123.0	NaN
7	8.0	Hassan	17.0	Information	122.0	NaN
8	9.0	Jami	17.0	Defence	156.0	NaN

In [203]:
```
df.dropna(axis=1, how='all')
```

Out[203]:

	ID	Names	Grades	Depts	Salaries
0	1.0	Asad	16.0	Accounts	NaN
1	2.0	Fahad	17.0	Taxation	123.0
2	3.0	NaN	18.0	Accounts	121.0
3	4.0	Afzal	19.0	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0
5	6.0	Nasir	19.0	NaN	100.0
6	7.0	NaN	14.0	Accounts	123.0
7	8.0	Hassan	17.0	Information	122.0
8	9.0	Jami	17.0	Defence	156.0
9	NaN	NaN	NaN	NaN	NaN

In [204]:
```
df
```

Out[204]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
0	1.0	Asad	16.0	Accounts	NaN	NaN
1	2.0	Fahad	17.0	Taxation	123.0	NaN
2	3.0	NaN	18.0	Accounts	121.0	NaN
3	4.0	Afzal	19.0	NaN	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN
5	6.0	Nasir	19.0	NaN	100.0	NaN
6	7.0	NaN	14.0	Accounts	123.0	NaN
7	8.0	Hassan	17.0	Information	122.0	NaN
8	9.0	Jami	17.0	Defence	156.0	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN

In [206]:
```
df.dropna(axis=0,thresh=5)
```

Out[206]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
1	2.0	Fahad	17.0	Taxation	123.0	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN
7	8.0	Hassan	17.0	Information	122.0	NaN
8	9.0	Jami	17.0	Defence	156.0	NaN

In [208]:
```
df.dropna(axis=1,thresh=4)
```

Out[208]:

	ID	Names	Grades	Depts	Salaries
0	1.0	Asad	16.0	Accounts	NaN
1	2.0	Fahad	17.0	Taxation	123.0
2	3.0	NaN	18.0	Accounts	121.0
3	4.0	Afzal	19.0	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0
5	6.0	Nasir	19.0	NaN	100.0
6	7.0	NaN	14.0	Accounts	123.0
7	8.0	Hassan	17.0	Information	122.0
8	9.0	Jami	17.0	Defence	156.0
9	NaN	NaN	NaN	NaN	NaN

Handling Duplicates Records

In [213]:
```
df.iloc[10:,:]=np.nan
```

In [215]:
```
df.iloc[8,:] = np.nan
```

In [216]:
```
df
```

Out[216]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
0	1.0	Asad	16.0	Accounts	NaN	NaN
1	2.0	Fahad	17.0	Taxation	123.0	NaN
2	3.0	NaN	18.0	Accounts	121.0	NaN
3	4.0	Afzal	19.0	NaN	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN
5	6.0	Nasir	19.0	NaN	100.0	NaN
6	7.0	NaN	14.0	Accounts	123.0	NaN
7	8.0	Hassan	17.0	Information	122.0	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN

In [219]:
```
df.duplicated()
```

Out[219]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9     True
dtype: bool
```

In [220]:
```
df.duplicated().sum()
```

Out[220]:
```
1
```

In [223]:
```
df.drop_duplicates(subset=['Depts', 'Na_Vals'])
```

Out[223]:

	ID	Names	Grades	Depts	Salaries	Na_Vals
0	1.0	Asad	16.0	Accounts	NaN	NaN
1	2.0	Fahad	17.0	Taxation	123.0	NaN
3	4.0	Afzal	19.0	NaN	NaN	NaN
4	5.0	Yasir	16.0	Information	99.0	NaN

In [ ]:

Combining and Reshaping Data

Merging and Joining Data Frame

In [1]:
```
import pandas as pd
import numpy as np
import datetime
customers  = {'customer_id':[1234,1235,1111,2222,3334,3333,4444,5555],
              'Name':['Mike', 'Maria', 'Nasir', 'Ali',"Ahmed", 'Saad',"Hamid",
              'Address':['abc23-1', 'xyz-3','abc23-2', 'xyz-5','abc23-6', 'xyz-
              }

customers = pd.DataFrame(customers)
customers
```

Out[1]:

	customer_id	Name	Address
0	1234	Mike	abc23-1
1	1235	Maria	xyz-3
2	1111	Nasir	abc23-2
3	2222	Ali	xyz-5
4	3334	Ahmed	abc23-6
5	3333	Saad	xyz-0
6	4444	Hamid	abc23-11
7	5555	Kami	xyz-301

In [2]:
```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 3 columns):
customer_id    8 non-null int64
Name           8 non-null object
Address        8 non-null object
dtypes: int64(1), object(2)
memory usage: 320.0+ bytes
```

In [3]:
```
# customers.set_index('customer_id')
```

In [4]:
```
orders = {'customer_id':[1234,1235,1234,1234,1235,1267,1237,1890],
          'order_id':[11,22,33,44,55,66,77,88],
         'order_date':[datetime.date(2023,2,1),datetime.date(2023,1,1),
                       datetime.date(2023,2,2),datetime.date(2023,1,1),
                       datetime.date(2023,2,3),datetime.date(2023,1,2),
                       datetime.date(2023,2,4),datetime.date(2023,1,2)],
         }

orders = pd.DataFrame(orders)
orders
```

Out[4]:

	customer_id	order_id	order_date
0	1234	11	2023-02-01
1	1235	22	2023-01-01
2	1234	33	2023-02-02
3	1234	44	2023-01-01
4	1235	55	2023-02-03
5	1267	66	2023-01-02
6	1237	77	2023-02-04
7	1890	88	2023-01-02

In [5]:
```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 3 columns):
customer_id    8 non-null int64
order_id       8 non-null int64
order_date     8 non-null object
dtypes: int64(2), object(1)
memory usage: 320.0+ bytes
```

Merge

- Inner
- Outer
- Left
- Right

In [6]:
```
merged_data = pd.merge(customers,orders)
# inner >> intersection
```

In [7]:
```
merged_data.sort_values(by='customer_id',ascending=False)
```

Out[7]:

	customer_id	Name	Address	order_id	order_date
3	1235	Maria	xyz-3	22	2023-01-01
4	1235	Maria	xyz-3	55	2023-02-03
0	1234	Mike	abc23-1	11	2023-02-01
1	1234	Mike	abc23-1	33	2023-02-02
2	1234	Mike	abc23-1	44	2023-01-01

In [8]:
```
merged_data = pd.merge(orders,customers)
```

In [9]:
```
merged_data
```

Out[9]:

	customer_id	order_id	order_date	Name	Address
0	1234	11	2023-02-01	Mike	abc23-1
1	1234	33	2023-02-02	Mike	abc23-1
2	1234	44	2023-01-01	Mike	abc23-1
3	1235	22	2023-01-01	Maria	xyz-3
4	1235	55	2023-02-03	Maria	xyz-3

In [10]:
```
merged_data = pd.merge(orders,customers,how='inner')
merged_data
```

Out[10]:

	customer_id	order_id	order_date	Name	Address
0	1234	11	2023-02-01	Mike	abc23-1
1	1234	33	2023-02-02	Mike	abc23-1
2	1234	44	2023-01-01	Mike	abc23-1
3	1235	22	2023-01-01	Maria	xyz-3
4	1235	55	2023-02-03	Maria	xyz-3

In [11]:
```
merged_data = pd.merge(customers,orders, how='outer')# outer merger >>> union
merged_data
```

Out[11]:

	customer_id	Name	Address	order_id	order_date
0	1234	Mike	abc23-1	11.0	2023-02-01
1	1234	Mike	abc23-1	33.0	2023-02-02
2	1234	Mike	abc23-1	44.0	2023-01-01
3	1235	Maria	xyz-3	22.0	2023-01-01
4	1235	Maria	xyz-3	55.0	2023-02-03
5	1111	Nasir	abc23-2	NaN	NaN
6	2222	Ali	xyz-5	NaN	NaN
7	3334	Ahmed	abc23-6	NaN	NaN
8	3333	Saad	xyz-0	NaN	NaN
9	4444	Hamid	abc23-11	NaN	NaN
10	5555	Kami	xyz-301	NaN	NaN
11	1267	NaN	NaN	66.0	2023-01-02
12	1237	NaN	NaN	77.0	2023-02-04
13	1890	NaN	NaN	88.0	2023-01-02

In [12]:
```
merged_data = pd.merge(customers,orders, how='left')
merged_data
```

Out[12]:

	customer_id	Name	Address	order_id	order_date
0	1234	Mike	abc23-1	11.0	2023-02-01
1	1234	Mike	abc23-1	33.0	2023-02-02
2	1234	Mike	abc23-1	44.0	2023-01-01
3	1235	Maria	xyz-3	22.0	2023-01-01
4	1235	Maria	xyz-3	55.0	2023-02-03
5	1111	Nasir	abc23-2	NaN	NaN
6	2222	Ali	xyz-5	NaN	NaN
7	3334	Ahmed	abc23-6	NaN	NaN
8	3333	Saad	xyz-0	NaN	NaN
9	4444	Hamid	abc23-11	NaN	NaN
10	5555	Kami	xyz-301	NaN	NaN

In [13]:
```
merged_data = pd.merge(customers,orders, how='right')
merged_data
```

Out[13]:

	customer_id	Name	Address	order_id	order_date
0	1234	Mike	abc23-1	11	2023-02-01
1	1234	Mike	abc23-1	33	2023-02-02
2	1234	Mike	abc23-1	44	2023-01-01
3	1235	Maria	xyz-3	22	2023-01-01
4	1235	Maria	xyz-3	55	2023-02-03
5	1267	NaN	NaN	66	2023-01-02
6	1237	NaN	NaN	77	2023-02-04
7	1890	NaN	NaN	88	2023-01-02

In [15]:
```
df1 = pd.DataFrame({'left': ['foo', 'bar']})
df2 = pd.DataFrame({'right': [7, 8]})
df = pd.merge(df1,df2, how='cross')
df
```

```
---------------------------------------------------------------------------
MergeError                                Traceback (most recent call last)
<ipython-input-15-0ddd13bea12b> in <module>
      1 df1 = pd.DataFrame({'left': ['foo', 'bar']})
      2 df2 = pd.DataFrame({'right': [7, 8]})
----> 3 df = pd.merge(df1,df2, how='cross')
      4 df

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in merge(left, right, how, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator, validate)
     79         copy=copy,
     80         indicator=indicator,
---> 81         validate=validate,
     82     )
     83     return op.get_result()

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in __init__(self, left, right, how, on, left_on, right_on, axis, left_index, right_index, sort, suffixes, copy, indicator, validate)
    617             warnings.warn(msg, UserWarning)
    618
--> 619         self._validate_specification()
    620
    621         # note this function has side effects

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _validate_specification(self)
   1188                                 ron=self.right_on,
   1189                                 lidx=self.left_index,
-> 1190                                 ridx=self.right_index,
   1191                             )
   1192                     )

MergeError: No common columns to perform merge on. Merge options: left_on=None, right_on=None, left_index=False, right_index=False
```

In [16]:
```
df1 = pd.DataFrame({'key':[1,1],'left': ['foo', 'bar']})
df2 = pd.DataFrame({'key':[1,1],'right': [7, 8]})
df = pd.merge(df1,df2, how='cross')[['left','right']]
df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-16-6e69deda8a5c> in <module>
      1 df1 = pd.DataFrame({'key':[1,1],'left': ['foo', 'bar']})
      2 df2 = pd.DataFrame({'key':[1,1],'right': [7, 8]})
----> 3 df = pd.merge(df1,df2, how='cross')[['left','right']]
      4 df

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in merge(left, rig
ht, how, on, left_on, right_on, left_index, right_index, sort, suffixes, cop
y, indicator, validate)
     81         validate=validate,
     82     )
---> 83     return op.get_result()
     84
     85

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in get_result(sel
f)
    640             self.left, self.right = self._indicator_pre_merge(self.le
ft, self.right)
    641
--> 642         join_index, left_indexer, right_indexer = self._get_join_info
()
    643
    644         ldata, rdata = self.left._data, self.right._data

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _get_join_info
(self)
    857             )
    858         else:
--> 859             (left_indexer, right_indexer) = self._get_join_indexers()
    860
    861             if self.right_index:

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _get_join_index
ers(self)
    836         """ return the join indexers """
    837         return _get_join_indexers(
--> 838             self.left_join_keys, self.right_join_keys, sort=self.sor
t, how=self.how
    839         )
    840

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _get_join_index
ers(left_keys, right_keys, sort, how, **kwargs)
   1316     if how == "left":
   1317         kwargs["sort"] = sort
-> 1318     join_func = _join_functions[how]
   1319
   1320     return join_func(lkey, rkey, count, **kwargs)

KeyError: 'cross'
```

In [18]:
```
pd.__version__
```

Out[18]:
```
'0.25.1'
```

In [25]:
```
customers  = {'customerID':[1234,1235,1111,2222,3334,3333,4444,5555],
              'Name':['Mike', 'Maria', 'Nasir', 'Ali',"Ahmed", 'Saad',"Hamid",
              'Address':['abc23-1', 'xyz-3','abc23-2', 'xyz-5','abc23-6', 'xyz-
              }

customers = pd.DataFrame(customers)

orders = {'customer_id':[1234,1235,1234,1234,1235,1267,1237,1890],
          'order_id':[11,22,33,44,55,66,77,88],
         'order_date':[datetime.date(2023,2,1),datetime.date(2023,1,1),
                       datetime.date(2023,2,2),datetime.date(2023,1,1),
                       datetime.date(2023,2,3),datetime.date(2023,1,2),
                       datetime.date(2023,2,4),datetime.date(2023,1,2)],
         }

orders = pd.DataFrame(orders)
```

In [26]:
```
customers
```

Out[26]:

	customerID	Name	Address
0	1234	Mike	abc23-1
1	1235	Maria	xyz-3
2	1111	Nasir	abc23-2
3	2222	Ali	xyz-5
4	3334	Ahmed	abc23-6
5	3333	Saad	xyz-0
6	4444	Hamid	abc23-11
7	5555	Kami	xyz-301

In [27]:
```
orders
```

Out[27]:

	customer_id	order_id	order_date
0	1234	11	2023-02-01
1	1235	22	2023-01-01
2	1234	33	2023-02-02
3	1234	44	2023-01-01
4	1235	55	2023-02-03
5	1267	66	2023-01-02
6	1237	77	2023-02-04
7	1890	88	2023-01-02

In [28]:
```
merged = pd.merge(customers,orders)
```

```
---------------------------------------------------------------------------
MergeError                                Traceback (most recent call last)
<ipython-input-28-06b68d5c504b> in <module>
----> 1 merged = pd.merge(customers,orders)

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in merge(left, rig
ht, how, on, left_on, right_on, left_index, right_index, sort, suffixes, cop
y, indicator, validate)
     79         copy=copy,
     80         indicator=indicator,
---> 81         validate=validate,
     82     )
     83     return op.get_result()

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in __init__(self,
left, right, how, on, left_on, right_on, axis, left_index, right_index, sort,
suffixes, copy, indicator, validate)
    617             warnings.warn(msg, UserWarning)
    618
--> 619         self._validate_specification()
    620
    621         # note this function has side effects

~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _validate_speci
fication(self)
   1188                                 ron=self.right_on,
   1189                                 lidx=self.left_index,
-> 1190                                 ridx=self.right_index,
   1191                             )
   1192                         )

MergeError: No common columns to perform merge on. Merge options: left_on=Non
e, right_on=None, left_index=False, right_index=False
```

In [31]:
```
merged = pd.merge(customers,orders,left_on='customerID', right_on='customer_id
```

In [32]:
```
merged
```

Out[32]:

	customerID	Name	Address	customer_id	order_id	order_date
0	1234	Mike	abc23-1	1234	11	2023-02-01
1	1234	Mike	abc23-1	1234	33	2023-02-02
2	1234	Mike	abc23-1	1234	44	2023-01-01
3	1235	Maria	xyz-3	1235	22	2023-01-01
4	1235	Maria	xyz-3	1235	55	2023-02-03

In [34]:
```
customers  = {'customerID':[1234,1235,1111,2222,3334,3333,4444,5555],
              'Name':['Mike', 'Maria', 'Nasir', 'Ali',"Ahmed", 'Saad',"Hamid",
              'Address':['abc23-1', 'xyz-3','abc23-2', 'xyz-5','abc23-6', 'xyz-
               'order_date':[datetime.date(2023,2,1),datetime.date(2023,1,1),
                        datetime.date(2023,2,2),datetime.date(2023,1,1),
                        datetime.date(2023,2,3),datetime.date(2023,1,2),
                        datetime.date(2023,2,4),datetime.date(2023,1,2)]
              }

customers = pd.DataFrame(customers)

orders = {'customerID':[1234,1235,1234,1234,1235,1267,1237,1890],
          'order_id':[11,22,33,44,55,66,77,88],
         'order_date':[datetime.date(2023,2,1),datetime.date(2023,1,1),
                        datetime.date(2023,2,2),datetime.date(2023,1,1),
                        datetime.date(2023,2,3),datetime.date(2023,1,2),
                        datetime.date(2023,2,4),datetime.date(2023,1,2)],
         }

orders = pd.DataFrame(orders)
```

In [35]:
```
customers
```

Out[35]:

	customerID	Name	Address	order_date
0	1234	Mike	abc23-1	2023-02-01
1	1235	Maria	xyz-3	2023-01-01
2	1111	Nasir	abc23-2	2023-02-02
3	2222	Ali	xyz-5	2023-01-01
4	3334	Ahmed	abc23-6	2023-02-03
5	3333	Saad	xyz-0	2023-01-02
6	4444	Hamid	abc23-11	2023-02-04
7	5555	Kami	xyz-301	2023-01-02

In [36]:
```
orders
```

Out[36]:

	customerID	order_id	order_date
0	1234	11	2023-02-01
1	1235	22	2023-01-01
2	1234	33	2023-02-02
3	1234	44	2023-01-01
4	1235	55	2023-02-03
5	1267	66	2023-01-02
6	1237	77	2023-02-04
7	1890	88	2023-01-02

In [37]:
```
df_merged = pd.merge(customers,orders)
df_merged
```

Out[37]:

	customerID	Name	Address	order_date	order_id
0	1234	Mike	abc23-1	2023-02-01	11
1	1235	Maria	xyz-3	2023-01-01	22

In [40]:
```
df_merged = pd.merge(customers,orders, on='order_date',suffixes=("_left", "_ri
df_merged
```

Out[40]:

	customerID_left	Name	Address	order_date	customerID_right	order_id
0	1234	Mike	abc23-1	2023-02-01	1234	11
1	1235	Maria	xyz-3	2023-01-01	1235	22
2	1235	Maria	xyz-3	2023-01-01	1234	44
3	2222	Ali	xyz-5	2023-01-01	1235	22
4	2222	Ali	xyz-5	2023-01-01	1234	44
5	1111	Nasir	abc23-2	2023-02-02	1234	33
6	3334	Ahmed	abc23-6	2023-02-03	1235	55
7	3333	Saad	xyz-0	2023-01-02	1267	66
8	3333	Saad	xyz-0	2023-01-02	1890	88
9	5555	Kami	xyz-301	2023-01-02	1267	66
10	5555	Kami	xyz-301	2023-01-02	1890	88
11	4444	Hamid	abc23-11	2023-02-04	1237	77

In [41]:
```
df_merged = pd.merge(customers,orders, on=['order_date', 'customerID'],suffixe
df_merged
```

Out[41]:

	customerID	Name	Address	order_date	order_id
0	1234	Mike	abc23-1	2023-02-01	11
1	1235	Maria	xyz-3	2023-01-01	22

In [45]:
```
df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two',
                           'two'],
                   'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'baz': [1, 2, 3, 4, 5, 6],
                   'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
df
```

Out[45]:

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

In [52]:
```
df.pivot(index='foo', columns='bar')
```

Out[52]:

	baz			zoo		
bar	**A**	**B**	**C**	**A**	**B**	**C**
foo						
one	1	2	3	x	y	z
two	4	5	6	q	w	t

In [48]:
```
df.T
```

Out[48]:

	0	1	2	3	4	5
foo	one	one	one	two	two	two
bar	A	B	C	A	B	C
baz	1	2	3	4	5	6
zoo	x	y	z	q	w	t

In [56]:
```
sales = pd.read_excel("SaleData.xlsx")
```

In [57]:
```
sales
```

Out[57]:

	OrderDate	Region	Manager	SalesMan	Item	Units	Unit_price	Sale_amt
0	2018-01-06	East	Martha	Alexander	Television	95.00	1198.000	113810.00
1	2018-01-23	Central	Hermann	Shelli	Home Theater	50.00	500.000	25000.00
2	2018-02-09	Central	Hermann	Luis	Television	36.00	1198.000	43128.00
3	2018-02-26	Central	Timothy	David	Cell Phone	27.00	225.000	6075.00
4	2018-03-15	West	Timothy	Stephen	Television	56.00	1198.000	67088.00
5	2018-04-01	East	Martha	Alexander	Home Theater	60.00	500.000	30000.00
6	2018-04-18	Central	Martha	Steven	Television	75.00	1198.000	89850.00
7	2018-05-05	Central	Hermann	Luis	Television	90.00	1198.000	107820.00
8	2018-05-22	West	Douglas	Michael	Television	32.00	1198.000	38336.00
9	2018-06-08	East	Martha	Alexander	Home Theater	60.00	500.000	30000.00
10	2018-06-25	Central	Hermann	Sigal	Television	90.00	1198.000	107820.00
11	2018-07-12	East	Martha	Diana	Home Theater	29.00	500.000	14500.00
12	2018-07-29	East	Douglas	Karen	Home Theater	81.00	500.000	40500.00
13	2018-08-15	East	Martha	Alexander	Television	35.00	1198.000	41930.00
14	2018-09-01	Central	Douglas	John	Desk	2.00	125.000	250.00
15	2018-09-18	East	Martha	Alexander	Video Games	16.00	58.500	936.00
16	2018-10-05	Central	Hermann	Sigal	Home Theater	28.00	500.000	14000.00
17	2018-10-22	East	Martha	Alexander	Cell Phone	64.00	225.000	14400.00
18	2018-11-08	East	Douglas	Karen	Cell Phone	15.00	225.000	3375.00
19	2018-11-25	Central	Hermann	Shelli	Video Games	96.00	58.500	5616.00
20	2018-12-12	Central	Douglas	John	Television	67.00	1198.000	80266.00
21	2018-12-29	East	Douglas	Karen	Video Games	74.00	58.500	4329.00
22	2019-01-15	Central	Timothy	David	Home Theater	46.00	500.000	23000.00
23	2019-02-01	Central	Douglas	John	Home Theater	87.00	500.000	43500.00
24	2019-02-18	East	Martha	Alexander	Home Theater	4.00	500.000	2000.00
25	2019-03-07	West	Timothy	Stephen	Home Theater	7.00	500.000	3500.00
26	2019-03-24	Central	Hermann	Luis	Video Games	50.00	58.500	2925.00
27	2019-04-10	Central	Martha	Steven	Television	66.00	1198.000	79068.00
28	2019-04-27	East	Martha	Diana	Cell Phone	96.00	225.000	21600.00
29	2019-05-14	Central	Timothy	David	Television	53.00	1198.000	63494.00
30	2019-05-31	Central	Timothy	David	Home Theater	80.00	500.000	40000.00
31	2019-06-17	Central	Hermann	Shelli	Desk	5.00	125.000	625.00
32	2019-07-04	East	Martha	Alexander	Video Games	62.00	58.500	3627.00
33	2019-07-21	Central	Hermann	Sigal	Video Games	55.00	58.500	3217.50
34	2019-08-07	Central	Hermann	Shelli	Video Games	42.00	58.500	2457.00
35	2019-08-24	West	Timothy	Stephen	Desk	3.00	125.000	375.00

	OrderDate	Region	Manager	SalesMan	Item	Units	Unit_price	Sale_amt
36	2019-09-10	Central	Timothy	David	Television	7.00	1198.000	8386.00
37	2019-09-27	West	Timothy	Stephen	Cell Phone	76.00	225.000	17100.00
38	2019-10-14	West	Douglas	Michael	Home Theater	57.00	500.000	28500.00
39	2019-10-31	Central	Martha	Steven	Television	14.00	1198.000	16772.00
40	2019-11-17	Central	Hermann	Luis	Home Theater	11.00	500.000	5500.00
41	2019-12-04	Central	Hermann	Luis	Home Theater	94.00	500.000	47000.00
42	2019-12-21	Central	Martha	Steven	Home Theater	28.00	500.000	14000.00
43	NaT	NaN	NaN	NaN	NaN	278.00	1125.000	62550.00
44	NaT	NaN	NaN	NaN	NaN	34.75	140.625	7818.75

In [ ]:

In [534]:
```
import pandas as pd
import numpy as np
from IPython.display import Image
```

In [535]:
```
pd.__version__
```

Out[535]:
```
'0.25.1'
```

Intro

Pandas' groupby is undoubtly one of the most powerful functionalities that Pandas brings to the table. However, most users only utilize a fraction of the capabilities of `groupby`.

`Groupby` allows to adopt a split-apply-combine approach to your data set. This is comparable to slicing and dicing your data such that it serves your specific need.

On a high level this means:

1. split the data based on column(s)/condition(s) into groups
2. apply a function/transformation to all the groups and combine the results into an output

0. Load the data

We are going to use data from a hypothetical sales division where we have, among other columns ficticious sales reps, order leads, order values, the company the deal might happen with and the date of the order lead.

In [536]:
```
order_leads = pd.read_csv(
    'https://raw.githubusercontent.com/FBosler/Medium-Data-Exploration/master/
    parse_dates = [3]
)
sales_team = pd.read_csv(
    'https://raw.githubusercontent.com/FBosler/Medium-Data-Exploration/master/
    parse_dates = [3]
)
df = pd.merge(order_leads,sales_team,on=['Company Id','Company Name'])
df = df.rename(columns={'Order Value':'Val','Converted':'Sale'})
df
```

Out[536]:

	Order Id	Company Id	Company Name	Date	Val	Sale	Sales Rep
0	HZSXLI1IS9RGABZW	D0AUXPP07H6AVSGD	Melancholy Social-Role	2017-10-13	6952	0	William Taylor
1	582WPS3OW8T6YT0R	D0AUXPP07H6AVSGD	Melancholy Social-Role	2017-09-02	7930	0	William Taylor
2	KRF65MQZBOYG4Y9T	D0AUXPP07H6AVSGD	Melancholy Social-Role	2016-12-21	5538	1	William Taylor
3	N3EDZ5V1WGSWW828	D0AUXPP07H6AVSGD	Melancholy Social-Role	2018-06-03	1113	0	William Taylor
4	QXBC8COXEXGFSPLP	D0AUXPP07H6AVSGD	Melancholy Social-Role	2014-07-26	4596	0	William Taylor
...	...	...	...	...	...	...	...
99995	HKZFX556ZQRZJZWR	APH243SK72T90MPS	Trade-Preparatory Quarterbacks	2017-11-06	7516	0	Ida Woodward
99996	962CSDMAJ49E0CRK	APH243SK72T90MPS	Trade-Preparatory Quarterbacks	2018-08-02	442	1	Ida Woodward
99997	ZW7RO9TLL6EVVJEC	APH243SK72T90MPS	Trade-Preparatory Quarterbacks	2014-11-02	8544	0	Ida Woodward
99998	LNKGIWMZ9RT49IE9	APH243SK72T90MPS	Trade-Preparatory Quarterbacks	2017-04-01	6650	0	Ida Woodward
99999	X9Y21H4JWX6OGC2Z	APH243SK72T90MPS	Trade-Preparatory Quarterbacks	2016-07-27	953	0	Ida Woodward

100000 rows × 8 columns

1. Groupby: Split the data based on condition/column into groups

The default approach of calling groupby, is by explicitly providing a column name to split the dataset by. However, and this is less known, you can also pass a Series (has to have the same length as the dataframe) to groupby. This means that you can group by a processed version of

a column, without having to create a new helper column for that.

groupby sales rep

First let's create a grouped DataFrame, i.e. split the dataset up.

In [537]:
```
grouped = df.groupby('Sales Rep')
grouped
```

Out[537]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x1155e19e8>
```

In [538]:
```
type(grouped)
```

Out[538]:
```
pandas.core.groupby.generic.DataFrameGroupBy
```

show all groups

calling `groups` on the grouped object returns the list of indices for every group (as every row can be uniquely identified via it's index)

In [539]:
```
grouped.groups
```

Out[539]:
```
{'Aaron Hendrickson': Int64Index([25612, 25613, 25614, 25615, 25616, 25617,
25618, 25619, 25620,
             25621,
             ...
             25894, 25895, 25896, 25897, 25898, 25899, 25900, 25901, 25902,
             25903],
            dtype='int64', length=292),
 'Adam Sawyer': Int64Index([67140, 67141, 67142, 67143, 67144, 67145, 6714
6, 67147, 67148,
             67149,
             ...
             67454, 67455, 67456, 67457, 67458, 67459, 67460, 67461, 67462,
             67463],
            dtype='int64', length=324),
 'Adele Kimmel': Int64Index([90915, 90916, 90917, 90918, 90919, 90920, 9092
1, 90922, 90923,
             90924,
             ...
             91020, 91021, 91022, 91023, 91024, 91025, 91026, 91027, 91028,
```

select a specific group

In [540]:
```
grouped.get_group('Aaron Hendrickson')
```

Out[540]:

	Order Id	Company Id	Company Name	Date	Val	Sale	Sales Rep	
25612	3BJY12LWBN7D0GJL	CE4544HJOFMONMH2	Follow-Up Boundary	2014-09-04	1940	0	Aaron Hendrickson	A
25613	W3HHOSC1H6A1PW37	CE4544HJOFMONMH2	Follow-Up Boundary	2015-09-24	2109	0	Aaron Hendrickson	A
25614	G9JKIZO4WD945GBH	CE4544HJOFMONMH2	Follow-Up Boundary	2014-12-06	4300	1	Aaron Hendrickson	A
25615	BKIJVKZ7REVN6P8B	CE4544HJOFMONMH2	Follow-Up Boundary	2017-05-07	3026	0	Aaron Hendrickson	A
25616	WFHGWR4PAD04A2GJ	CE4544HJOFMONMH2	Follow-Up Boundary	2016-01-20	5033	0	Aaron Hendrickson	A
...	...	...	...	...	...	...	...	
25899	NATK7K3TZUH32BBE	CGDGXAW6GNU6JIEG	Fiftieth Art'S	2015-01-27	6095	1	Aaron Hendrickson	A
25900	EGD6IRB0UML62XB0	CGDGXAW6GNU6JIEG	Fiftieth Art'S	2018-11-04	7652	1	Aaron Hendrickson	A
25901	9Z18A7D1T8EUH58D	CGDGXAW6GNU6JIEG	Fiftieth Art'S	2016-05-08	4746	0	Aaron Hendrickson	A
25902	R0LUW64V2F3O2HSD	CGDGXAW6GNU6JIEG	Fiftieth Art'S	2017-02-16	6158	0	Aaron Hendrickson	A
25903	UMHMBM5M179IHX6D	CGDGXAW6GNU6JIEG	Fiftieth Art'S	2017-07-28	2164	0	Aaron Hendrickson	A

292 rows × 8 columns

For the following examples we will use the simplest version of the apply step (and just count the rows in each group) via the `size` method. We do this, so that we can focus on the groupby operations.

We will go into much more detail regarding the apply methods in section 2 of the article.

Basic Example: Count rows in each group

In [541]:
```
grouped.size()
```

Out[541]:
```
Sales Rep
Aaron Hendrickson    292
Adam Sawyer          324
Adele Kimmel         115
Adrian Daugherty     369
Adrianna Shelton      37
                    ...
Willie Lin            44
Willie Rau            95
Willie Sanchez       309
Yvonne Jones          74
Yvonne Lindsey        67
Length: 499, dtype: int64
```

Advanced Example: Group by first name of sales rep

In [542]:
```
# Grouping by first name of our Sales Rep without creating a column
df.groupby(
    df['Sales Rep'].str.split(' ').str[0]
).size()
```

Out[542]:
```
Sales Rep
Aaron        292
Adam         324
Adele        115
Adrian       369
Adrianna      37
            ...
Wesley       144
Wilbert      213
William     1393
Willie       448
Yvonne       141
Length: 318, dtype: int64
```

Advanced Example: Grouping by whether or not there is a "William" in the name of the rep

In [543]:
```
df.groupby(
    df['Sales Rep'].apply(lambda x: 'William' in x)
).size()
```

Out[543]:
```
Sales Rep
False    97111
True      2889
dtype: int64
```

Advanced Example: Group by random series (for illustrative purposes only)

In [544]:
```
# Grouping random letters (obviously does not make sense)
df.groupby(
    pd.Series(np.random.choice(list('ABCDG'),len(df)))
).size()
```

Out[544]:
```
A    19852
B    19949
C    20184
D    19912
G    20103
dtype: int64
```

Advanced Example: Grouping by 3 evenly cut "Order Value" buckets

In [545]:
```
# qcut bins the passed Series into q evenly sized parts (and labels the bins)
df.groupby(
    pd.qcut(x=df['Val'],q=3,labels=['low','mid','high'])
).size()
```

Out[545]:
```
Val
low     33339
mid     33336
high    33325
dtype: int64
```

Advanced Example: Grouping by cutom "Order Value" buckets

In [547]:
```
# cut bins the passed Series into defined bins
df.groupby(
    pd.cut(df['Val'],[0,3000,5000,7000,10000])
).size()
```

Out[547]:
```
Val
(0, 3000]        29220
(3000, 5000]     19892
(5000, 7000]     20359
(7000, 10000]    30529
dtype: int64
```

pd.Grouper

The biggest "gotcha" in that area for me was understanding pd.Grouper, which allows seamless aggregation on different date/time granularities.

Advanced Example: Grouping by year

In [548]:
```
df.groupby(pd.Grouper(key='Date',freq='Y')).size()
```

Out[548]:
```
Date
2014-12-31    19956
2015-12-31    20054
2016-12-31    20133
2017-12-31    20079
2018-12-31    19778
Freq: A-DEC, dtype: int64
```

Advanced Example: Grouping by quarter

In [552]:
```
# grouping by year
df.groupby(pd.Grouper(key='Date',freq='SM')).size()
```

Out[552]:
```
Date
2013-12-31    761
2014-01-15    837
2014-01-31    820
2014-02-15    740
2014-02-28    817
              ...
2018-10-31    810
2018-11-15    805
2018-11-30    824
2018-12-15    837
2018-12-31     50
Freq: SM-15, Length: 121, dtype: int64
```

Advanced Example: Grouping by multiple columns

In [553]:
```
df.groupby(['Sales Rep','Company Name']).size()
```

Out[553]:
```
Sales Rep          Company Name
Aaron Hendrickson  6-Foot Homosexuals             20
                   63D House'S                    27
                   Angular Liberalism             28
                   Boon Blish'S                   18
                   Business-Like Structures       21
                                                  ..
Yvonne Jones       Entry-Limiting Westinghouse    20
                   Intractable Fairgoers          18
                   Smarter Java                   17
Yvonne Lindsey     Meretricious Fabrication       28
                   Shrill Co-Op                   39
Length: 4619, dtype: int64
```

2. Apply and Combine: Apply a function/transformation to all groups and combine the results into an output

In the previous section we discussed how to group the data based on various conditions. This section deals with available functions that we can apply to the groups before combining them to a final result.

The section is structured along how to use

1. apply,
2. agg(regate),
3. transform, and
4. filter, on a grouped object.

If you are anything like me when I started using groupby, you are probably using a combination of 1. and 2. along the lines of:

`group = df.groupby('GROUP')` and then:

- `group.apply(mean)`
- `group.agg(mean)`
- `group['INTERSTING COLUMN'].apply(mean)`
- `group.agg({'INTERSTING COLUMN':mean})`
- `group.mean()`

Where `mean` could also be another function.

All of them work. And most of the time, the result is going to be roughly what you expected it to be. However, there are nuances to apply and agg that are worthwhile pointing out.

Additionally, but much more importantly there are two lesser-known extremly powerful functions that can be used on a grouped object, `filter` and `transform` .

`Apply` : Let's get `apply` out of the way

This is somewhat confusing, as we often talk about applying functions while there also is an apply function. But bear with me. The `apply` function applies a function along an axis of the DataFrame. This could be either column-wise or row-wise. `apply` is not strictly speaking a function that can only be used in the context of groupby. It could also used on an entire dataframe, like in the following example.

In [554]:
```
_ = pd.DataFrame(
    np.random.random((2,6)),
    columns=list('ABCDEF')
)
_
```

Out[554]:

	A	B	C	D	E	F
0	0.400871	0.964452	0.305504	0.252755	0.542108	0.911006
1	0.035872	0.729973	0.453693	0.330489	0.967016	0.590400

In [555]:
```
_.apply(sum, axis=0) # axis=0 is default, so you can drop that
```

Out[555]:
```
A    0.436743
B    1.694425
C    0.759197
D    0.583244
E    1.509123
F    1.501406
dtype: float64
```

In [556]:
```
_.apply(sum, axis=1)
```

Out[556]:
```
0    3.376695
1    3.107443
dtype: float64
```

But it can also be used in a groupby context. Which makes sense, considering the fact that each group is a smaller DataFrame on its own. Keep in mind that the function will be applied to the entire DataFrame. This means typically you want to select the columns you are `applying` a function to. We will leave it at these examples and instead focus on `agg(regation)` which is the "intended" way of aggregating groups.

In [557]:
```
df.groupby(
    pd.Grouper(key='Date',freq='Y')
)['Sale'].apply(sum)
```

Out[557]:
```
Date
2014-12-31    3681
2015-12-31    3800
2016-12-31    3881
2017-12-31    3068
2018-12-31    2478
Freq: A-DEC, Name: Sale, dtype: int64
```

In [558]:
```
df.groupby(
    pd.Grouper(key='Date',freq='Y')
)['Val','Sale'].apply(sum)
```

Out[558]:

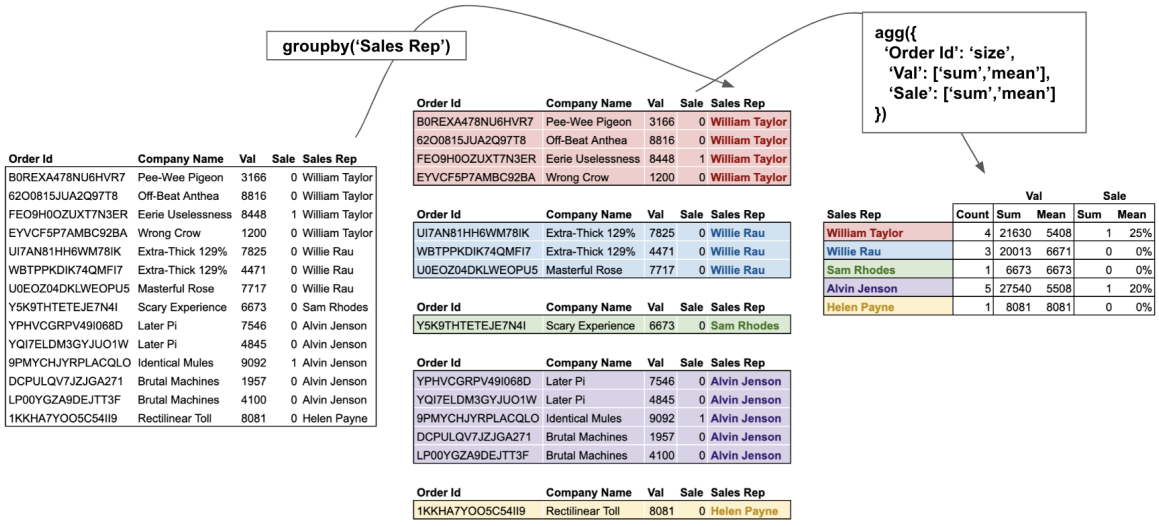
Date	Val	Sale
2014-12-31	100422394	3681
2015-12-31	101724648	3800
2016-12-31	101789642	3881
2017-12-31	101957784	3068
2018-12-31	100399962	2478

In [559]:
```
df.groupby(
    pd.Grouper(key='Date',freq='Y')
)['Val','Sale'].apply(sum).to_clipboard(sep=' ')
```

agg(regate)

In [561]:
```
Image(filename='groupby-agg.png')
```

Out[561]:



In [562]:
```
# agg(regation)
df.groupby('Sales Rep')\
\
.agg({
    'Order Id':'size',
    'Val':['sum','mean'],
    'Sale':['sum','mean']
})
```

Out[562]:

	Order Id	Val		Sale
	size	sum	mean	sum	mean
Sales Rep					
Aaron Hendrickson	292	1550608	5310.301370	46	0.157534
Adam Sawyer	324	1587828	4900.703704	48	0.148148
Adele Kimmel	115	527339	4585.556522	20	0.173913
Adrian Daugherty	369	1841274	4989.902439	51	0.138211
Adrianna Shelton	37	186651	5044.621622	8	0.216216
...	...	...	...	...	...
Willie Lin	44	254128	5775.636364	6	0.136364
Willie Rau	95	434918	4578.084211	19	0.200000
Willie Sanchez	309	1525229	4936.016181	52	0.168285
Yvonne Jones	74	416388	5626.864865	12	0.162162
Yvonne Lindsey	67	324334	4840.805970	18	0.268657

499 rows × 5 columns

In [563]:
```
def cr(x):
    return round(np.mean(x),2)

aggregation = {
    'Potential Sales':('Val','size'),
    'Sales':('Sale','sum'),
    'Conversion Rate':('Sale',cr)
}

df.groupby('Sales Rep').agg(**aggregation)
```

Out[563]:

Sales Rep	Potential Sales	Sales	Conversion Rate
Aaron Hendrickson	292	46	0.16
Adam Sawyer	324	48	0.15
Adele Kimmel	115	20	0.17
Adrian Daugherty	369	51	0.14
Adrianna Shelton	37	8	0.22
...	...	...	...
Willie Lin	44	6	0.14
Willie Rau	95	19	0.20
Willie Sanchez	309	52	0.17
Yvonne Jones	74	12	0.16
Yvonne Lindsey	67	18	0.27

499 rows × 3 columns

In [564]:
```
def cr(x):
    return round(np.mean(x),2)

_ = df.groupby('Sales Rep').agg({
    'Val':'size',
    'Sale':['sum',cr]
})

_.columns = ['Potential Sales','Sales','Conversion Rate']
_
```

Out[564]:

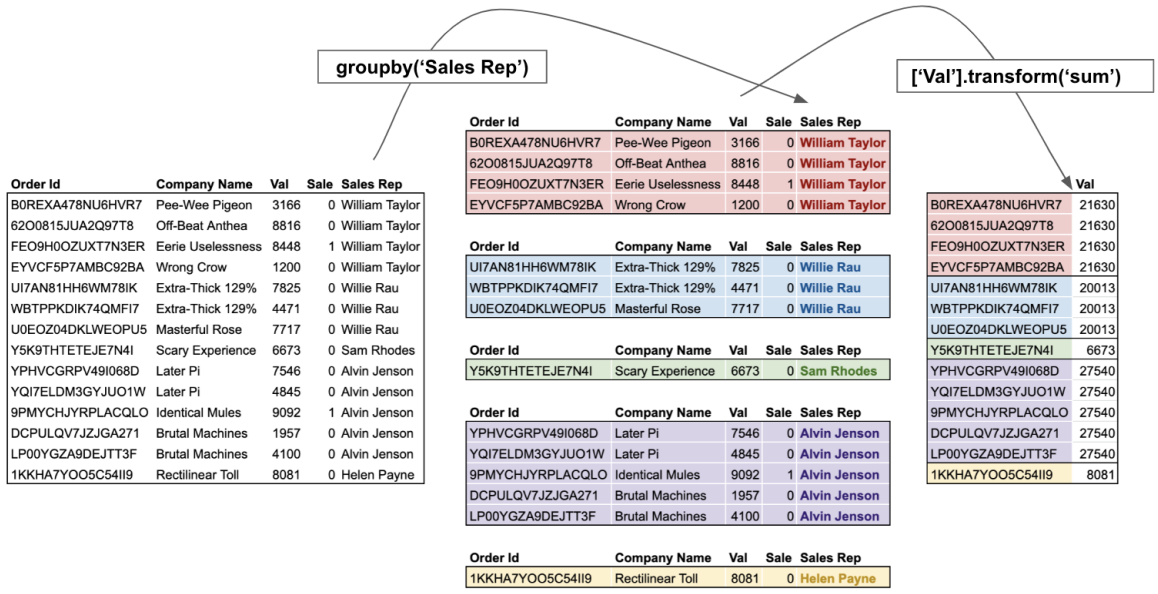
Sales Rep	Potential Sales	Sales	Conversion Rate
Aaron Hendrickson	292	46	0.16
Adam Sawyer	324	48	0.15
Adele Kimmel	115	20	0.17
Adrian Daugherty	369	51	0.14
Adrianna Shelton	37	8	0.22
...	...	...	...
Willie Lin	44	6	0.14
Willie Rau	95	19	0.20
Willie Sanchez	309	52	0.17
Yvonne Jones	74	12	0.16
Yvonne Lindsey	67	18	0.27

499 rows × 3 columns

Transform

In [567]:
```
Image(filename='groupby-transform.png')
```

Out[567]:



In [568]:
```
df.groupby('Sales Rep')['Val'].transform(lambda x: x/sum(x))
```

Out[568]:
```
0        0.004991
1        0.005693
2        0.003976
3        0.000799
4        0.003300
             ...
99995    0.012088
99996    0.000711
99997    0.013741
99998    0.010695
99999    0.001533
Name: Val, Length: 100000, dtype: float64
```

In [569]:
```
df['%'] = df.groupby('Sales Rep')['Val'].transform(
  lambda x: x/sum(x)
)
```

In [570]:
```
del df['%']
```

In [571]:
```
# filter (at least 200k in sales)
df.groupby('Sales Rep').filter(lambda x: (x['Val'] * x['Sale']).sum() > 200000
```

Out[571]:

	Order Id	Company Id	Company Name	Date	Val	Sale	Sales Rep	
0	HZSXLI1IS9RGABZW	D0AUXPP07H6AVSGD	Melancholy Social-Role	2017-10-13	6952	0	William Taylor	ZTZ
1	582WPS3OW8T6YT0R	D0AUXPP07H6AVSGD	Melancholy Social-Role	2017-09-02	7930	0	William Taylor	ZTZ
2	KRF65MQZBOYG4Y9T	D0AUXPP07H6AVSGD	Melancholy Social-Role	2016-12-21	5538	1	William Taylor	ZTZ
3	N3EDZ5V1WGSWW828	D0AUXPP07H6AVSGD	Melancholy Social-Role	2018-06-03	1113	0	William Taylor	ZTZ
4	QXBC8COXEXGFSPLP	D0AUXPP07H6AVSGD	Melancholy Social-Role	2014-07-26	4596	0	William Taylor	ZTZ
...	...	...	...	...	...	...	...	
99129	GVRNNUAPFE0IUBHW	BLURX3WQK51RI6P7	Baptismal Tensions	2014-01-11	2677	0	Richard Dugas	Y43H
99130	GX4S2LXPU3VZOS4Z	BLURX3WQK51RI6P7	Baptismal Tensions	2018-05-12	6877	0	Richard Dugas	Y43H
99131	FU4ELGDQEGSGOHFZ	BLURX3WQK51RI6P7	Baptismal Tensions	2014-10-16	9189	0	Richard Dugas	Y43H
99132	S553MU5UDAUID8TG	BLURX3WQK51RI6P7	Baptismal Tensions	2016-05-01	2013	0	Richard Dugas	Y43H
99133	CV3ANPEYZAGVDGGT	BLURX3WQK51RI6P7	Baptismal Tensions	2016-06-20	4543	0	Richard Dugas	Y43H

68313 rows × 8 columns

In [572]:
```
# Let's add this for verification
df['cr'] = df.groupby('Sales Rep')['Sale'].transform('mean')

df.groupby('Sales Rep').filter(lambda x: x['Sale'].mean() > .3)
```

Out[572]:

	Order Id	Company Id	Company Name	Date	Val	Sale	Sales Rep	
3897	4MWBSVADRWSNLBA0	4D9PJORE7YYNDV2E	Qualitative Asimov'S	2014-03-17	2637	0	Teddy Cook	36I1
3898	8C13U50FF5ZKU1TJ	4D9PJORE7YYNDV2E	Qualitative Asimov'S	2015-10-26	9495	0	Teddy Cook	36I1
3899	D6Y3HVKNT480ADL1	4D9PJORE7YYNDV2E	Qualitative Asimov'S	2017-12-09	9156	0	Teddy Cook	36I1
3900	JXO5XMLWEDZKLGCG	4D9PJORE7YYNDV2E	Qualitative Asimov'S	2016-07-05	1219	0	Teddy Cook	36I1
3901	37PVLZLXMXRCZLNK	4D9PJORE7YYNDV2E	Qualitative Asimov'S	2018-03-07	4374	0	Teddy Cook	36I1
...	...	...	...	...	...	...	...	
94627	H72B7571AMUFAH2A	JQ7DP9EX0HY1OKRU	Road-Shy Small-Town	2015-08-07	2764	0	Jennifer Peck	UTJ4
94628	SFQVIH3PNXVLR7PM	JQ7DP9EX0HY1OKRU	Road-Shy Small-Town	2014-11-28	1008	0	Jennifer Peck	UTJ4
94629	B4GFG6BPT1HOHJOU	JQ7DP9EX0HY1OKRU	Road-Shy Small-Town	2018-06-27	8999	0	Jennifer Peck	UTJ4
94630	7ZO3XQ1C3U3BOM6T	JQ7DP9EX0HY1OKRU	Road-Shy Small-Town	2015-02-19	2079	0	Jennifer Peck	UTJ4
94631	CN1LKURFMOKKK15D	JQ7DP9EX0HY1OKRU	Road-Shy Small-Town	2018-09-06	1681	0	Jennifer Peck	UTJ4

366 rows × 9 columns

In [573]:
```
del df['cr']
```

Advanced Examples Agg

In [576]:
```
# Grouping by 3 evenly cut "Order Value" buckets
df.groupby(
    pd.qcut(df['Val'],3,['low','mid','high'])
).agg({'Val':['mean','std'],'Sale':['sum','size']})
```

Out[576]:

	Val		Sale	
	mean	std	sum	size
Val				
low	1759.218063	955.198008	5653	33339
mid	5078.456234	957.488032	5577	33336
high	8352.541395	945.261300	5678	33325

In [577]:
```
# Grouping by cutom "Order Value" buckets
df.groupby(
    pd.cut(df['Val'],[0,3000,5000,7000,10000])
).agg({'Val':['mean','std'],'Sale':['sum','size']})
```

Out[577]:

	Val		Sale	
	mean	std	sum	size
Val				
(0, 3000]	1555.837474	839.194392	4913	29220
(3000, 5000]	3998.367283	579.842580	3389	19892
(5000, 7000]	5999.759369	582.543076	3399	20359
(7000, 10000]	8488.592355	868.442376	5207	30529

In [578]:
```
df.groupby(
    pd.cut(df['Val'],[0,5000,10000],labels=['low','high'])
).agg(
    **{'Conversion Rate':pd.NamedAgg(column='Sale',aggfunc=lambda x: sum(x)/le
)
```

Out[578]:

	Conversion Rate
Val	
low	0.169042
high	0.169116

In [579]:
```
# grouping by year and a cut oder value column
df.groupby(
    [pd.Grouper(key='Date',freq='Y'),pd.qcut(df['Val'],3,['low','mid','high'])
).agg(
    **{'Conversion Rate':pd.NamedAgg(column='Sale',aggfunc=lambda x: sum(x)/le
).unstack()
```

Out[579]:

	Conversion Rate		
Val	low	mid	high
Date			
2014-12-31	0.185964	0.185123	0.182244
2015-12-31	0.192487	0.184618	0.191403
2016-12-31	0.193346	0.189693	0.195297
2017-12-31	0.151350	0.151591	0.155423
2018-12-31	0.123520	0.124867	0.127470

In [580]:
```
!open .
```

In [ ]:
```

```