Implementing Conditional Statements in Python

In [1]:
```
x = 7
if(x%2 == 0):
    print('x is even')
else:
    print('x is odd')
```

```
x is odd
```

In [2]:
```
y = 91
if(y > 90):
    print('Grade A')
elif(y > 60):
    print('Grade B')
else:
    print('Grade F')
```

```
Grade A
```

Looping Constructs

In [3]:
```
for i in range(5):
    print('Python is the best')
```

```
Python is the best
Python is the best
Python is the best
Python is the best
Python is the best
```

In [4]:
```
for i in range(11,20,2):
    print(i)
```

```
11
13
15
17
19
```

Functions

In [5]:
```
def area_circle(r):
    area = 3.14 * r * r
    return area
```

In [6]:
```
area_circle(1)
```

Out[6]:
```
3.14
```

In [7]:
```
def compare(a,b):
    if a > b :
        greater = a
    else:
        greater = b
    return greater
```

In [8]:
```
compare(10,20)
```

Out[8]:
```
20
```

In [9]:
```
compare(6,5)
```

Out[9]:
```
6
```

In [10]:
```
# Lambda function
a = lambda x,y : x+y
a(2,3)
```

Out[10]:
```
5
```

In [11]:
```
(lambda e : e * 2)(5)
```

Out[11]:
```
10
```

In [12]:
```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x % 2 == 0), my_list))
new_list
```

Out[12]:
```
[4, 6, 8, 12]
```

Data Structures

In [13]:
```
# Lists

# A list is an ordered data structure with elements separated by comma and enclos

list1 = [1,2,3,4,5,6]
list2 = ['Python','is','Awesome']

list1 # Integer type
```

Out[13]:
```
[1, 2, 3, 4, 5, 6]
```

In [14]:
```
list2 # String type
```

Out[14]:
```
['Python', 'is', 'Awesome']
```

In [15]:
```
list3 = [1,'Python',2,'is',3,'Awesome']
list3 # Mixed type
```

Out[15]:
```
[1, 'Python', 2, 'is', 3, 'Awesome']
```

In [16]:
```
list3[1] # To extract a single element
```

Out[16]:
```
'Python'
```

In [17]:
```
list3[1:4] # To extract a sequence of elements
```

Out[17]:
```
['Python', 2, 'is']
```

In [18]:
```
list3[-1] # To extract a element using negative index
```

Out[18]:
```
'Awesome'
```

In [20]:
```
# Adding single element to an existing list
list3.append(4)
list3
```

Out[20]:
```
[1, 'Python', 2, 'is', 3, 'Awesome', 4, 4]
```

In [21]:
```
# Adding multiple elements to an existing list
list3.extend([5,6])
list3
```

Out[21]:
```
[1, 'Python', 2, 'is', 3, 'Awesome', 4, 4, 5, 6]
```

In [22]:
```
# Adding list to a list
list3.append([7,8])
list3
```

Out[22]:
```
[1, 'Python', 2, 'is', 3, 'Awesome', 4, 4, 5, 6, [7, 8]]
```

In [23]:
```
list3[9]
```

Out[23]:
```
6
```

In [24]:
```
# Deleting an element by a value
list3.remove(4)
list3
```

Out[24]:
```
[1, 'Python', 2, 'is', 3, 'Awesome', 4, 5, 6, [7, 8]]
```

In [25]:
```
# Deleting an element by index
del list3[1]
list3
```

Out[25]:
```
[1, 2, 'is', 3, 'Awesome', 4, 5, 6, [7, 8]]
```

In [26]:
```
# Printing each element of a list
for i in list3:
    print (i)
```

```
1
2
is
3
Awesome
4
5
6
[7, 8]
```

In [28]:
```
# Dictionary

# A dictionary is an unordered data structure with elements separated by comma an
# A dictionary is enclosed in curly brackets.

dict1 = {'Ramesh' : 150, 'Suresh' : 146, 'Sudesh' : 160}
dict2 = {'Ramesh' : [150,46], 'Suresh' : [146,58], 'Sudesh' : [160,50]}
```

In [29]:
```
dict1
```

Out[29]:
```
{'Ramesh': 150, 'Sudesh': 160, 'Suresh': 146}
```

In [30]:
```
dict2
```

Out[30]:
```
{'Ramesh': [150, 46], 'Sudesh': [160, 50], 'Suresh': [146, 58]}
```

In [31]:
```
# Elements are accessed by keys rather than index
dict2['Suresh']
```

Out[31]:
```
[146, 58]
```

In [32]:
```
# Adding single element to a dictionary
dict2['Amit'] = [160,51]
dict2
```

Out[32]:
```
{'Amit': [160, 51],
 'Ramesh': [150, 46],
 'Sudesh': [160, 50],
 'Suresh': [146, 58]}
```

In [37]:
```
# Adding multiple elements at once
dict2.update({'Sunil' : [160,56], 'Disha' : [150,78]})
dict2
```

Out[37]:
```
{'Amit': [160, 51],
 'Disha': [150, 78],
 'Ramesh': [150, 46],
 'Sudesh': [160, 50],
 'Sunil': [160, 56],
 'Suresh': [146, 58]}
```

In [38]:
```
# Deleting an element
del dict2['Sunil']
dict2
```

Out[38]:
```
{'Amit': [160, 51],
 'Disha': [150, 78],
 'Ramesh': [150, 46],
 'Sudesh': [160, 50],
 'Suresh': [146, 58]}
```

In [44]:
```
importers = {'El Salvador' : 1234,
             'Nicaragua' : 152,
             'Spain' : 252
            }

exporters = {'Spain' : 252,
             'Germany' : 251,
             'Italy' : 1563
             }
```

In [45]:
```
importers.keys() & exporters.keys()
```

Out[45]:
```
{'Spain'}
```

In [46]:
```
importers.items() & exporters.items()
```

Out[46]:
```
{('Spain', 252)}
```

In [47]:
```
importers.keys() - exporters.keys()
```

Out[47]:
```
{'El Salvador', 'Nicaragua'}
```

Begining of Pandas

Abc of Series

In [40]:
```
import pandas as pd
import numpy as np

obj = pd.Series([4,7,-5,3])
obj
```

Out[40]:
```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [2]:
```
obj.values
```

Out[2]:
```
array([ 4,  7, -5,  3], dtype=int64)
```

In [3]:
```
obj.index
```

Out[3]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [4]:
```
# change the default index
obj2 = pd.Series([4,7,-5,3], index = ['d','b','a','c'])
obj2
```

Out[4]:
```
d    4
b    7
a   -5
c    3
dtype: int64
```

In [5]:
```
# Accessing single series elements based on index
obj2['b']
```

Out[5]:
```
7
```

In [7]:
```
# Insert single element on existing series
obj2['d'] = 6
obj2
```

Out[7]:
```
d    6
b    7
a   -5
c    3
dtype: int64
```

In [ ]:
```
# Accessing multiple series elements based on index
obj2[['d','b','a']]
```

In [10]:
```
# Multiplying series elements
obj2 * 2
```

Out[10]:
```
d    12
b    14
a   -10
c     6
dtype: int64
```

In [11]:
```
'b' in obj2
```

Out[11]:
```
True
```

In [12]:
```
'e' in obj2
```

Out[12]:
```
False
```

In [13]:
```
# We can create series by passing a dictionary
sdata = {'Ohio' : 35000, 'Texas' : 71000, 'Oregon' : 16000, 'Utah' : 5000}

obj3 = pd.Series(sdata)
obj3
```

Out[13]:
```
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
```

In [14]:
```
states = ['California','Ohio','Oregon','Texas']
```

In [15]:
```
obj4 = pd.Series(sdata,index = states)
obj4
```

Out[15]:
```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

In [16]:
```
pd.isnull(obj4)
```

Out[16]:
```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

In [17]:
```
pd.notnull(obj4)
```

Out[17]:
```
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

In [18]:
```
obj3 + obj4
```

Out[18]:
```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

In [19]:
```
obj4[obj4 > 16000]
```

Out[19]:
```
Ohio     35000.0
Texas    71000.0
dtype: float64
```

In [20]:
```
# To access series element based on index location
obj3.iloc[3]
```

Out[20]:
```
5000
```

In [21]:
```
obj3[3]
```

Out[21]:
```
5000
```

In [22]:
```
# To access series element based on index label
obj3.loc['Utah']
```

Out[22]:
```
5000
```

In [23]:
```
obj3['Utah']
```

Out[23]:
```
5000
```

In [24]:
```
sports = {99 : 'Bhutan', 100 : 'Scotland', 101 : 'Japan', 102 : 'South Korea'}
s = pd.Series(sports)
s
```

Out[24]:
```
99          Bhutan
100       Scotland
101          Japan
102    South Korea
dtype: object
```

In [25]:
```
# What happens if your index is a list of integers
# Pandas can't determine automatically whether you are intending to query by inde
# have to use iloc or loc attributes directly.

# s[0] # It Won't walk

s.iloc[0]
```

Out[25]:
```
'Bhutan'
```

In [41]:
```
s = pd.Series([100, 120, 101, 3])
s
```

Out[41]:
```
0    100
1    120
2    101
3      3
dtype: int64
```

In [42]:
```
# Selects the smallest two numbers from the series elements
s = s.nsmallest(2)
s
```

Out[42]:
```
3      3
0    100
dtype: int64
```

In [43]:
```
# Selects the largest two numbers from the series elements
s = s.nlargest(2)
s
```

Out[43]:
```
0    100
3      3
dtype: int64
```

In [62]:
```
# Broadcasting
s += 2
s
```

Out[62]:
```
0    102
1    122
2    103
3      5
dtype: int64
```

In [27]:
```
total = np.sum(s)
print(total)
```

```
324
```

In [32]:
```
total = 0
for item in s:
    total += item
print(total)
```

```
324
```

In [35]:
```
# Return random integers from low (inclusive) to high (exclusive)

# Syntax : numpy.random.randint(low, high=None, size=None)

np.random.randint(100,200,10)
```

Out[35]:
```
array([121, 155, 192, 182, 191, 179, 143, 108, 193, 148])
```

In [36]:
```
np.random.randint(100,200,(2,3))
```

Out[36]:
```
array([[134, 168, 130],
       [155, 117, 170]])
```

In [44]:
```
np.random.randn(2)
```

Out[44]:
```
array([0.80519789, 2.20451544])
```

In [45]:
```
np.random.randn(2,3)
```

Out[45]:
```
array([[ 0.10226113, -0.86301055,  1.0540317 ],
       [ 0.21307125,  0.71891457, -1.558937  ]])
```

In [49]:
```
np.random.randn(3,2,3)
```

Out[49]:
```
array([[[ 1.08978778,  1.66977438, -2.95072125],
        [-0.24715961,  2.07791046,  1.18162848]],

       [[ 1.72897628, -0.38747807,  0.05249034],
        [ 0.34344909, -0.97524719, -1.81228638]],

       [[-1.30924845,  0.37208512, -0.55962682],
        [ 0.08174392,  0.78824688,  1.23139077]]])
```

In [63]:
```
s1 = pd.Series(np.arange(3))
s1
```

Out[63]:
```
0    0
1    1
2    2
dtype: int32
```

In [64]:
```
# Copying through variables

data = pd.Series(['a', 'b', 'c', 'd'])
new = data
new[1]='Changed value'

print(new)
print(data)

# the changes made in new data are also reflected in the old data since the new v
```

```
0               a
1    Changed value
2               c
3               d
dtype: object
0               a
1    Changed value
2               c
3               d
dtype: object
```

In [65]:
```
# Using Pandas.copy() method

data = pd.Series(['a', 'b', 'c', 'd'])
new = data.copy()
new[1]='Changed value'

print(new)
print(data)
```

```
0               a
1    Changed value
2               c
3               d
dtype: object
0    a
1    b
2    c
3    d
dtype: object
```

In [112]:
```
# In case of deep copy, a copy of object is copied in other object. It means that
# object do not reflect in the original object.

s2 = [1, 2, [3,5], 4]

# using deepcopy to deep copy

import copy
s3 = copy.deepcopy(s2)

# original elements of list

print ("The original elements before deep copying :")

for i in range(0,len(s3)):
    print (s3[i],end=" ")
```

```
The original elements before deep copying :
1 2 [3, 5] 4
```

In [113]:
```
# adding and element to new list
s3[2][0]= 7
```

In [114]:
```
print ("The new list of elements after deep copying :")

for i in range (0,len(s3)):
    print(s3[i],end = " ")
```

```
The new list of elements after deep copying :
1 2 [7, 5] 4
```

In [115]:
```
print ("The original elements after deep copying :")

for i in range (0,len(s2)):
    print(s2[i],end = " ")
```

```
The original elements after deep copying :
1 2 [3, 5] 4
```

In [106]:
```
# In case of shallow copy, a reference of object is copied in other object. It me
# to a copy of object do reflect in the original object.

s2_1 = [1, 2, [3,5], 4]

# using copy to shallow copy

import copy
s3_1 = copy.copy(s2_1)

# original elements of list

print ("The original elements before shallow copying :")

for i in range(0,len(s3_1)):
    print (s3_1[i],end=" ")
```

```
The original elements before shallow copying :
1 2 [3, 5] 4
```

In [107]:
```
# adding and element to new list
s3_1[2][0] = 7
```

In [108]:
```
print ("The original elements after shallow copying :")
for i in range (0,len(s2_1)):
    print(s2_1[i],end = " ")
```

```
The original elements after shallow copying :
1 2 [7, 5] 4
```

In [ ]:
```
# The difference between shallow and deep copying is only relevant for compound o
# (objects that contain other objects, like lists or class instances).
```

Abc of Dataframes

In [80]:
```
import pandas as pd

p1 = ({'Name' : 'Shampa','Age' : 27, 'City' : 'Mumbai'})
p2 = ({'Name' : 'Ankush','Age' : 28, 'City' :'Mumbai'})
p3 = ({'Name' : 'Priyanka', 'Age' : 18, 'City' : 'Kolkata'})
p4 = ({'Name' : 'Shreyank', 'Age' : 1, 'City' : 'Bangalore'})
p5 = ({'Name' : 'Rajdip','Age' : 24,'City' : 'Jaisalmer'})
p6 = ({'Name' : 'Amit','Age' : 31,'City' : 'Kolkata'})
p7 = ({'Name': 'Aman','Age' : 24,'City' : 'Jaisalmer'})

df = pd.DataFrame([p1,p2,p3,p4,p5,p6,p7], index = ['A1','B1','C1','D1','E1','F1',
df
```

Out[80]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip
F1	31	Kolkata	Amit
G1	24	Jaisalmer	Aman

In [2]:
```
# To get the dimension of the dataframe
df.ndim
```

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

In [3]:
```
# To get the size of the dataframe
df.size
```

Out[3]:
```
21
```

In [4]:
```
# To get the index of the dataframe
df.index
```

Out[4]:
```
Index(['A1', 'B1', 'C1', 'D1', 'E1', 'F1', 'G1'], dtype='object')
```

In [5]:
```
# To get the columns of the dataframe
df.columns
```

Out[5]:
```
Index(['Age', 'City', 'Name'], dtype='object')
```

In [6]:
```
# To get the axes of the dataframes
df.axes
```

Out[6]:
```
[Index(['A1', 'B1', 'C1', 'D1', 'E1', 'F1', 'G1'], dtype='object'),
 Index(['Age', 'City', 'Name'], dtype='object')]
```

In [7]:
```
# To get the values of the dataframes
df.values
```

Out[7]:
```
array([[27, 'Mumbai', 'Shampa'],
       [28, 'Mumbai', 'Ankush'],
       [18, 'Kolkata', 'Priyanka'],
       [1, 'Bangalore', 'Shreyank'],
       [24, 'Jaisalmer', 'Rajdip'],
       [31, 'Kolkata', 'Amit'],
       [24, 'Jaisalmer', 'Aman']], dtype=object)
```

In [8]:
```
# To get the datatypes of every columns in the dataframes
df.dtypes
```

Out[8]:
```
Age      int64
City    object
Name    object
dtype: object
```

In [9]:
```
# To know whether the dataframe is empty or not
df.empty
```

Out[9]:
```
False
```

In [10]:
```
# To know the no. of rows and columns in the dataframe
df.shape
```

Out[10]:
```
(7, 3)
```

In [11]:
```
# To get the basic statistics measures of all the numeric variables
df.describe()
```

Out[11]:

	Age
count	7.000000
mean	21.857143
std	10.056981
min	1.000000
25%	21.000000
50%	24.000000
75%	27.500000
max	31.000000

In [12]:
```
# To get the basic statistics measures of all the categorical variables
df.describe(include = ['object'])
```

Out[12]:

	City	Name
count	7	7
unique	4	7
top	Kolkata	Amit
freq	2	1

In [16]:
```
type(df)
```

Out[16]:
```
pandas.core.frame.DataFrame
```

Integer location based indexing or selection by position (iloc)

In [17]:
```
# To select first row of data frame
df.iloc[0]
```

Out[17]:
```
Age         27
City    Mumbai
Name    Shampa
Name: A1, dtype: object
```

In [18]:
```
# To select second row of data frame
df.iloc[1]
```

Out[18]:
```
Age         28
City    Mumbai
Name    Ankush
Name: B1, dtype: object
```

In [19]:
```
# To select last row of data frame
df.iloc[-1]
```

Out[19]:
```
Age            24
City    Jaisalmer
Name         Aman
Name: G1, dtype: object
```

In [33]:
```
# To select 1st 3 rows and all columns
df.iloc[0:3,:]
```

Out[33]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka

In [21]:
```
#To select first column of data frame
df.iloc[:,0]
```

Out[21]:
```
A1    27
B1    28
C1    18
D1     1
E1    24
F1    31
G1    24
Name: Age, dtype: int64
```

In [22]:
```
# To select second column of data frame
df.iloc[:,1]
```

Out[22]:
```
A1       Mumbai
B1       Mumbai
C1      Kolkata
D1    Bangalore
E1    Jaisalmer
F1      Kolkata
G1    Jaisalmer
Name: City, dtype: object
```

In [23]:
```
# To select last column of data frame
df.iloc[:,-1]
```

Out[23]:
```
A1      Shampa
B1      Ankush
C1    Priyanka
D1    Shreyank
E1      Rajdip
F1        Amit
G1        Aman
Name: Name, dtype: object
```

In [24]:
```
# Multiple columns and rows can be selected together using the .iloc indexer

# To select first five rows of dataframe
df.iloc[0:5]
```

Out[24]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip

In [30]:
```
# To select first two columns of data frame with all rows
df.iloc[:, 0:2]
```

Out[30]:

	Age	City
A1	27	Mumbai
B1	28	Mumbai
C1	18	Kolkata
D1	1	Bangalore
E1	24	Jaisalmer
F1	31	Kolkata
G1	24	Jaisalmer

In [32]:
```
# To select arbitary rows and columns
df.iloc[[0,1,3], [0,2]]
```

Out[32]:

	Age	Name
A1	27	Shampa
B1	28	Ankush
D1	1	Shreyank

In [ ]:
```
# Note that .iloc returns a Pandas Series when one row is selected, and a Pandas
# or if any column in full is selected.

# When selecting multiple columns or multiple rows in this manner, remember that
# selected will run from the first number to one minus the second number. e.g. [1
```

Selecting pandas data using loc

In [34]:
```
# Selecting rows by label/index

df.loc['A1']
```

Out[34]:
```
Age          27
City     Mumbai
Name     Shampa
Name: A1, dtype: object
```

In [35]:
```
df.loc[['A1','C1']]
```

Out[35]:

	Age	City	Name
A1	27	Mumbai	Shampa
C1	18	Kolkata	Priyanka

In [37]:
```
# Note that the first example returns a series, and the second returns a DataFram
```

Out[37]:

	City	Name
A1	Mumbai	Shampa
B1	Mumbai	Ankush
D1	Bangalore	Shreyank

In [38]:
```
df.loc['A1':'C1',['City','Name']]
```

Out[38]:

	City	Name
A1	Mumbai	Shampa
B1	Mumbai	Ankush
C1	Kolkata	Priyanka

In [ ]:
```
# Boolean / Logical indexing using .loc
```

In [39]:
```
# the statement df[‘Name’] == ‘Ankush’] produces a Pandas Series with a True/Fals
# where there are “True” values for the rows where the Name is “Ankush”.
df.loc[df['Name'] == 'Ankush']
```

Out[39]:

	Age	City	Name
B1	28	Mumbai	Ankush

In [40]:
```
df.loc[df['Name'] == 'Ankush',['Age','Name']]
```

Out[40]:

	Age	Name
B1	28	Ankush

In [70]:
```
# Note that when selecting columns, if one column only is selected, the .loc oper
# DataFrame, use a one-element list to keep the DataFrame format.
df.loc[df['Name'] == 'Ankush',['Age']]
```

Out[70]:

	Age
B1	28

In [47]:
```
# Select rows with City equal to some values
df.loc[df['City'].isin(['Mumbai','Bangalore'])]
```

Out[47]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
D1	1	Bangalore	Shreyank

In [48]:
```
# Preview DataFrames with head() and tail()
df.head(2)
```

Out[48]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush

In [50]:
```
# Preview DataFrames with head() and tail()
df.tail(2)
```

Out[50]:

	Age	City	Name
F1	31	Kolkata	Amit
G1	24	Jaisalmer	Aman

In [61]:
```
# How many rows the dataset
df['Age'].count()
```

Out[61]: 7

In [62]:
```
# What was the max value in age variable?
df['Age'].max()
```

Out[62]: 31

In [63]:
```
# Frequency distribution of City variable
df['City'].value_counts()
```

Out[63]:
```
Kolkata      2
Jaisalmer    2
Mumbai       2
Bangalore    1
Name: City, dtype: int64
```

In [64]:
```
# Number of unique city entries
df['City'].nunique()
```

Out[64]: 4

In [65]:
```
# Sum of age variable
df['Age'].sum()
```

Out[65]: 153

In [66]:
```
# Std deviation of age variable
df['Age'].std(ddof = 0)
```

Out[66]: 9.310954706743042

In [67]:
```
# variance of age variable
df['Age'].var(ddof = 0)
```

Out[67]:
```
86.69387755102039
```

In [68]:
```
# Mean of age variable
df['Age'].mean()
```

Out[68]:
```
21.857142857142858
```

In [69]:
```
# Mode of age variable
df['Age'].mode()
```

Out[69]:
```
0    24
dtype: int64
```

In [71]:
```
# Median of age variable
df['Age'].median()
```

Out[71]:
```
24.0
```

In [82]:
```
#Quartiles
Q1 = df['Age'].quantile(0.25)
Q2 = df['Age'].quantile(0.50)
Q3 = df['Age'].quantile(0.75)
Q4 = df['Age'].quantile(1)

print('1st Quartile', Q1)
print('2nd Quartile', Q2)
print('3rd Quartile', Q3)
print('4th Quartile', Q4)
```

```
1st Quartile 21.0
2nd Quartile 24.0
3rd Quartile 27.5
4th Quartile 31.0
```

In [84]:
```
# Range
max_data = df['Age'].max()
min_data = df['Age'].min()

range_data = max_data - min_data
print('Range of the data is ',range_data)
```

```
Range of the data is  30
```

In [85]:
```
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)

IQR = Q3 - Q1

print('IQR is ', IQR)
```

```
IQR is  6.5
```

In [74]:
```
# Cumulative sum of age variable

# cumsum() -> cumulative sum
# cumprod() -> cumulative product
# cummax() -> cumulative maximum
# cummin() -> cumulative minimum

df['Age'].cumsum()
```

Out[74]:
```
A1     27
B1     55
C1     73
D1     74
E1     98
F1    129
G1    153
Name: Age, dtype: int64
```

In [75]:
```
#Transpose a dataframe
df.T
```

Out[75]:

	A1	B1	C1	D1	E1	F1	G1
Age	27	28	18	1	24	31	24
City	Mumbai	Mumbai	Kolkata	Bangalore	Jaisalmer	Kolkata	Jaisalmer
Name	Shampa	Ankush	Priyanka	Shreyank	Rajdip	Amit	Aman

In [76]:
```
df.T.loc['Age']
```

Out[76]:
```
A1    27
B1    28
C1    18
D1     1
E1    24
F1    31
G1    24
Name: Age, dtype: object
```

In [79]:
```
df['Age']
```

Out[79]:
```
A1    27
B1    28
C1    18
D1     1
E1    24
F1    31
G1    24
Name: Age, dtype: int64
```

In [80]:
```
df.T.iloc[0]
```

Out[80]:
```
A1    27
B1    28
C1    18
D1     1
E1    24
F1    31
G1    24
Name: Age, dtype: object
```

In [11]:
```
# Deleting Single Row
df.drop('G1', axis = 0)
```

Out[11]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip
F1	31	Kolkata	Amit

In [10]:
```
# Deleting Multiple Rows
df.drop(['G1','F1'], axis = 0)
```

Out[10]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip

In [12]:
```
# Deleting Single column
df.drop('City', axis = 1)
```

Out[12]:

	Age	Name
A1	27	Shampa
B1	28	Ankush
C1	18	Priyanka
D1	1	Shreyank
E1	24	Rajdip
F1	31	Amit
G1	24	Aman

In [13]:
```
# Deleting Multiple columns
df.drop(['City','Name'], axis = 1)
```

Out[13]:

	Age
A1	27
B1	28
C1	18
D1	1
E1	24
F1	31
G1	24

In [8]:
```
df.drop(columns = ['Age','City'])
```

Out[8]:

	Name
A1	Shampa
B1	Ankush
C1	Priyanka
D1	Shreyank
E1	Rajdip
F1	Amit
G1	Aman

In [83]:
```
# Temporarily deleted, to delete permanently use the option inplace = True
df
```

Out[83]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip
F1	31	Kolkata	Amit
G1	24	Jaisalmer	Aman

In [115]:
```
df.drop('G1', axis = 0, inplace = True)
```

In [116]:
```
df
```

Out[116]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip
F1	31	Kolkata	Amit

In [120]:
```
 # Another way of deleting
del df['Age']
```

In [121]:
```
df
```

Out[121]:

	City	Name
A1	Mumbai	Shampa
B1	Mumbai	Ankush
C1	Kolkata	Priyanka
D1	Bangalore	Shreyank
E1	Jaisalmer	Rajdip
F1	Kolkata	Amit
G1	Jaisalmer	Aman

In [124]:
```
# Selecting rows based on a condition
df[df['Age'] > 24]
```

Out[124]:

	Age	City	Name
A1	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
F1	31	Kolkata	Amit

In [30]:
```
# Renaming a column and index
df.rename(columns = {'City' : 'Place'}, index = {'A1' : 'A2'})
```

Out[30]:

	Age	Place	Name
A2	27	Mumbai	Shampa
B1	28	Mumbai	Ankush
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank
E1	24	Jaisalmer	Rajdip
F1	31	Kolkata	Amit
G1	24	Jaisalmer	Aman

In [3]:
```
# Sorting by an axis:
df.sort_index(axis=1, ascending=False)
```

Out[3]:

	Name	City	Age
A1	Shampa	Mumbai	27
B1	Ankush	Mumbai	28
C1	Priyanka	Kolkata	18
D1	Shreyank	Bangalore	1
E1	Rajdip	Jaisalmer	24
F1	Amit	Kolkata	31
G1	Aman	Jaisalmer	24

In [4]:
```
#Sorting by an axis:
df.sort_index(axis=0, ascending=False)
```

Out[4]:

	Age	City	Name
G1	24	Jaisalmer	Aman
F1	31	Kolkata	Amit
E1	24	Jaisalmer	Rajdip
D1	1	Bangalore	Shreyank
C1	18	Kolkata	Priyanka
B1	28	Mumbai	Ankush
A1	27	Mumbai	Shampa

In [6]:
```
#Sorting by values:
df.sort_values(by='Age', ascending = False)
```

Out[6]:

	Age	City	Name
F1	31	Kolkata	Amit
B1	28	Mumbai	Ankush
A1	27	Mumbai	Shampa
E1	24	Jaisalmer	Rajdip
G1	24	Jaisalmer	Aman
C1	18	Kolkata	Priyanka
D1	1	Bangalore	Shreyank

In [17]:
```
# Use of set_index
df = df.set_index('Name')
df
```

Out[17]:

	Age	City
Name		
Shampa	27	Mumbai
Ankush	28	Mumbai
Priyanka	18	Kolkata
Shreyank	1	Bangalore
Rajdip	24	Jaisalmer
Amit	31	Kolkata
Aman	24	Jaisalmer

In [18]:
```
# Name of the index changed
df.index
```

Out[18]:
```
Index(['Shampa', 'Ankush', 'Priyanka', 'Shreyank', 'Rajdip', 'Amit', 'Aman'], d
type='object', name='Name')
```

In [19]:
```
df.loc['Shampa']
```

Out[19]:
```
Age          27
City     Mumbai
Name: Shampa, dtype: object
```

In [21]:
```
# To reset the index
df = df.reset_index()
df
```

Out[21]:

	Name	Age	City
0	Shampa	27	Mumbai
1	Ankush	28	Mumbai
2	Priyanka	18	Kolkata
3	Shreyank	1	Bangalore
4	Rajdip	24	Jaisalmer
5	Amit	31	Kolkata
6	Aman	24	Jaisalmer

In [22]:
```
df.index
```

Out[22]:
```
RangeIndex(start=0, stop=7, step=1)
```

In [23]:
```
# Use of reindex to change the order of columns
df = df.reindex(columns=['City', 'Age', 'Name'])
df
```

Out[23]:

	City	Age	Name
0	Mumbai	27	Shampa
1	Mumbai	28	Ankush
2	Kolkata	18	Priyanka
3	Bangalore	1	Shreyank
4	Jaisalmer	24	Rajdip
5	Kolkata	31	Amit
6	Jaisalmer	24	Aman

In [26]:
```
# Use of reindex to change the order of rows
df.reindex([0,6,5,4,3,2,1])
```

Out[26]:

	City	Age	Name
0	Mumbai	27	Shampa
6	Jaisalmer	24	Aman
5	Kolkata	31	Amit
4	Jaisalmer	24	Rajdip
3	Bangalore	1	Shreyank
2	Kolkata	18	Priyanka
1	Mumbai	28	Ankush

In [27]:
```
df.reindex([0,6,5,4,3,2,1], columns=['Name', 'Age', 'City'])
```

Out[27]:

	Name	Age	City
0	Shampa	27	Mumbai
6	Aman	24	Jaisalmer
5	Amit	31	Kolkata
4	Rajdip	24	Jaisalmer
3	Shreyank	1	Bangalore
2	Priyanka	18	Kolkata
1	Ankush	28	Mumbai

In [63]:
```
# Update the rows and columns in existing dataframe
df['Remarks'] = 'Good'
df.loc['E1', 'Age'] = np.nan
df.loc['E1', 'City'] = np.nan
df.loc['E1', 'Name'] = np.nan
df
```

Out[63]:

	Age	City	Name	Remarks
A1	27.0	Mumbai	Shampa	Good
B1	28.0	Mumbai	Ankush	Good
C1	18.0	Kolkata	Priyanka	Good
D1	1.0	Bangalore	Shreyank	Good
E1	NaN	NaN	NaN	Good
F1	31.0	Kolkata	Amit	Good
G1	24.0	Jaisalmer	Aman	Good

In [64]:
```
# Update values based on a condition
df.loc[df.index == 'E1', 'Remarks'] = np.nan
df.loc[df.index == 'G1', 'Remarks'] = np.nan
df
```

Out[64]:

	Age	City	Name	Remarks
A1	27.0	Mumbai	Shampa	Good
B1	28.0	Mumbai	Ankush	Good
C1	18.0	Kolkata	Priyanka	Good
D1	1.0	Bangalore	Shreyank	Good
E1	NaN	NaN	NaN	NaN
F1	31.0	Kolkata	Amit	Good
G1	24.0	Jaisalmer	Aman	NaN

In [65]:
```
# Dropping rows where all values are NA
df.dropna(how = 'all')
# df.dropna(axis = 1,how = 'all')
```

Out[65]:

	Age	City	Name	Remarks
A1	27.0	Mumbai	Shampa	Good
B1	28.0	Mumbai	Ankush	Good
C1	18.0	Kolkata	Priyanka	Good
D1	1.0	Bangalore	Shreyank	Good
F1	31.0	Kolkata	Amit	Good
G1	24.0	Jaisalmer	Aman	NaN

In [66]:
```
df.dropna(how = 'any')
```

Out[66]:

	Age	City	Name	Remarks
A1	27.0	Mumbai	Shampa	Good
B1	28.0	Mumbai	Ankush	Good
C1	18.0	Kolkata	Priyanka	Good
D1	1.0	Bangalore	Shreyank	Good
F1	31.0	Kolkata	Amit	Good

In [59]:
```
# Fill missing places with 0
df['Score'] = np.nan
df.fillna(0)
```

Out[59]:

	Age	City	Name	Score
A1	27	Mumbai	Shampa	0.0
B1	28	Mumbai	Ankush	0.0
C1	18	Kolkata	Priyanka	0.0
D1	1	Bangalore	Shreyank	0.0
E1	24	Jaisalmer	Rajdip	0.0
F1	31	Kolkata	Amit	0.0
G1	24	Jaisalmer	Aman	0.0

In [69]:
```
# Drop rows that contain less than two observations
df.dropna(thresh = 2)
```

Out[69]:

	Age	City	Name	Remarks
A1	27.0	Mumbai	Shampa	Good
B1	28.0	Mumbai	Ankush	Good
C1	18.0	Kolkata	Priyanka	Good
D1	1.0	Bangalore	Shreyank	Good
F1	31.0	Kolkata	Amit	Good
G1	24.0	Jaisalmer	Aman	NaN

In [48]:
```
import numpy as np
df2 = pd.DataFrame(np.arange(20).reshape(5,4))
df2
```

Out[48]:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

In [3]:
```
df = pd.DataFrame(np.random.randn(10, 4))
df
```

Out[3]:

	0	1	2	3
0	0.471152	0.532805	-0.172532	0.258010
1	1.435113	-1.202906	1.306918	-1.891925
2	-0.583750	0.585046	-1.473438	-0.760760
3	-0.305981	-0.678233	-2.260818	-1.687572
4	0.291623	-0.885667	-1.742831	0.015437
5	-1.549860	1.079697	1.185648	-0.162538
6	-0.959903	0.953907	-0.048494	0.765768
7	-0.590647	-0.149416	-0.287368	0.315150
8	-0.407303	-0.036361	0.851904	-0.911676
9	0.523317	-0.000795	-0.762681	0.902694

In [4]:
```
pieces = [df[:3], df[3:7], df[7:]]
pieces
```

Out[4]:
```
[          0         1         2         3
 0  0.471152  0.532805 -0.172532  0.258010
 1  1.435113 -1.202906  1.306918 -1.891925
 2 -0.583750  0.585046 -1.473438 -0.760760,
           0         1         2         3
 3 -0.305981 -0.678233 -2.260818 -1.687572
 4  0.291623 -0.885667 -1.742831  0.015437
 5 -1.549860  1.079697  1.185648 -0.162538
 6 -0.959903  0.953907 -0.048494  0.765768,
           0         1         2         3
 7 -0.590647 -0.149416 -0.287368  0.315150
 8 -0.407303 -0.036361  0.851904 -0.911676
 9  0.523317 -0.000795 -0.762681  0.902694]
```

In [7]:
```
pd.concat(pieces)
```

Out[7]:

	0	1	2	3
0	0.471152	0.532805	-0.172532	0.258010
1	1.435113	-1.202906	1.306918	-1.891925
2	-0.583750	0.585046	-1.473438	-0.760760
3	-0.305981	-0.678233	-2.260818	-1.687572
4	0.291623	-0.885667	-1.742831	0.015437
5	-1.549860	1.079697	1.185648	-0.162538
6	-0.959903	0.953907	-0.048494	0.765768
7	-0.590647	-0.149416	-0.287368	0.315150
8	-0.407303	-0.036361	0.851904	-0.911676
9	0.523317	-0.000795	-0.762681	0.902694

In [85]:
```
left = pd.DataFrame({
         'id':[1,2,3,4,5],
         'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
         'subject_id':['sub1','sub2','sub4','sub6','sub5']})
right = pd.DataFrame(
         {'id':[1,2,3,4,5],
         'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
         'subject_id':['sub2','sub4','sub3','sub6','sub5']})
```

In [86]:
```
left
```

Out[86]:

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5

In [87]:
```
right
```

Out[87]:

	Name	id	subject_id
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

In [89]:
```
pd.concat([left, right])
```

Out[89]:

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5
0	Billy	1	sub2
1	Brian	2	sub4
2	Bran	3	sub3
3	Bryce	4	sub6
4	Betty	5	sub5

In [90]:
```
pd.concat([left,right],keys=['x','y'])
```

Out[90]:

		Name	id	subject_id
x	0	Alex	1	sub1
	1	Amy	2	sub2
	2	Allen	3	sub4
	3	Alice	4	sub6
	4	Ayoung	5	sub5
y	0	Billy	1	sub2
	1	Brian	2	sub4
	2	Bran	3	sub3
	3	Bryce	4	sub6
	4	Betty	5	sub5

In [92]:
```
pd.concat([left,right],ignore_index = 'True')
```

Out[92]:

	Name	id	subject_id
0	Alex	1	sub1
1	Amy	2	sub2
2	Allen	3	sub4
3	Alice	4	sub6
4	Ayoung	5	sub5
5	Billy	1	sub2
6	Brian	2	sub4
7	Bran	3	sub3
8	Bryce	4	sub6
9	Betty	5	sub5

In [93]:
```
pd.concat([left,right],axis=1)
```

Out[93]:

	Name	id	subject_id	Name	id	subject_id
0	Alex	1	sub1	Billy	1	sub2
1	Amy	2	sub2	Brian	2	sub4
2	Allen	3	sub4	Bran	3	sub3
3	Alice	4	sub6	Bryce	4	sub6
4	Ayoung	5	sub5	Betty	5	sub5

In [6]:
```
df = pd.DataFrame(np.random.randn(8, 4), columns=['A','B','C','D'])
df
```

Out[6]:

	A	B	C	D
0	0.820153	-0.538027	0.333031	-0.544162
1	-0.775901	-0.550577	1.933888	-1.629364
2	-0.895730	-0.735608	-2.038051	-0.887756
3	-2.357170	0.105006	-0.769001	0.767905
4	0.518690	-0.283077	0.297053	1.034273
5	-0.739437	-0.146746	-0.172616	0.074032
6	-0.048359	-0.179831	2.251649	1.789412
7	0.371958	0.577648	-0.898418	-0.342568

In [7]:
```
s = df.iloc[3]
s
```

Out[7]:
```
A   -2.357170
B    0.105006
C   -0.769001
D    0.767905
Name: 3, dtype: float64
```

In [8]:
```
df.append(s,ignore_index=True)
```

Out[8]:

	A	B	C	D
0	0.820153	-0.538027	0.333031	-0.544162
1	-0.775901	-0.550577	1.933888	-1.629364
2	-0.895730	-0.735608	-2.038051	-0.887756
3	-2.357170	0.105006	-0.769001	0.767905
4	0.518690	-0.283077	0.297053	1.034273
5	-0.739437	-0.146746	-0.172616	0.074032
6	-0.048359	-0.179831	2.251649	1.789412
7	0.371958	0.577648	-0.898418	-0.342568
8	-2.357170	0.105006	-0.769001	0.767905

In [94]:
```
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
                     'Kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Ri
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017]
          'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}

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

Out[94]:

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014
5	812	4	Kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014
10	804	1	Royals	2015
11	690	2	Riders	2017

In [95]:
```
# Get the first entry for each Year
df.groupby('Year').first()
```

Out[95]:

	Points	Rank	Team
Year			
2014	876	1	Riders
2015	789	2	Riders
2016	756	1	Kings
2017	788	1	Kings

In [96]:
```
#Get the last entry for each Year
df.groupby('Year').last()
```

Out[96]:

	Points	Rank	Team
Year			
2014	701	4	Royals
2015	804	1	Royals
2016	694	2	Riders
2017	690	2	Riders

In [33]:
```
df.groupby('Year')['Points'].sum()
```

Out[33]:
```
Year
2014    3181
2015    3078
2016    1450
2017    1478
Name: Points, dtype: int64
```

In [14]:
```
df.groupby('Team').groups
```

Out[14]:
```
{'Devils': Int64Index([2, 3], dtype='int64'),
 'Kings': Int64Index([4, 5, 6, 7], dtype='int64'),
 'Riders': Int64Index([0, 1, 8, 11], dtype='int64'),
 'Royals': Int64Index([9, 10], dtype='int64')}
```

In [9]:
```
grouped = df.groupby('Year')
for name, group in grouped:
    print (name)
    print (group)
```

```
2014
   Points  Rank    Team  Year
0     876     1  Riders  2014
2     863     2  Devils  2014
4     741     3   Kings  2014
9     701     4  Royals  2014
2015
    Points  Rank    Team  Year
1      789     2  Riders  2015
3      673     3  Devils  2015
5      812     4   Kings  2015
10     804     1  Royals  2015
2016
   Points  Rank    Team  Year
6     756     1   Kings  2016
8     694     2  Riders  2016
2017
    Points  Rank    Team  Year
7      788     1   Kings  2017
11     690     2  Riders  2017
```

In [10]:
```
# Select a group
grouped.get_group(2014)
```

Out[10]:

	Points	Rank	Team	Year
0	876	1	Riders	2014
2	863	2	Devils	2014
4	741	3	Kings	2014
9	701	4	Royals	2014

In [49]:
```
# Create a groupby variable that groups Team by Year
grp_team = df['Team'].groupby(df['Year'])
list(grp_team)
```

Out[49]:
```
[(2014, 0    Riders
  2    Devils
  4     Kings
  9    Royals
  Name: Team, dtype: object), (2015, 1     Riders
  3     Devils
  5      Kings
  10    Royals
  Name: Team, dtype: object), (2016, 6     Kings
  8    Riders
  Name: Team, dtype: object), (2017, 7      Kings
  11    Riders
  Name: Team, dtype: object)]
```

In [51]:
```
grp_team.count()
```

Out[51]:
```
Year
2014    4
2015    4
2016    2
2017    2
Name: Team, dtype: int64
```

In [52]:
```
# Descriptive statistics by group
df['Team'].groupby(df['Year']).describe()
```

Out[52]:

Year	count	unique	top	freq
2014	4	4	Riders	1
2015	4	4	Riders	1
2016	2	2	Riders	1
2017	2	2	Riders	1

In [11]:
```
# Aggregations
import numpy as np
grouped.agg(np.mean)
```

Out[11]:

Year	Points	Rank
2014	795.25	2.5
2015	769.50	2.5
2016	725.00	1.5
2017	739.00	1.5

In [12]:
```
grouped['Points'].agg(np.mean)
```

Out[12]:
```
Year
2014    795.25
2015    769.50
2016    725.00
2017    739.00
Name: Points, dtype: float64
```

In [15]:
```
# Filteration
df.groupby('Team').filter(lambda x: len(x) >= 3)
```

Out[15]:

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
4	741	3	Kings	2014
5	812	4	Kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
11	690	2	Riders	2017

In [34]:
```
for key,value in df.iteritems():
   print(key,value)
```

```
Points 0     876
1     789
2     863
3     673
4     741
5     812
6     756
7     788
8     694
9     701
10    804
11    690
Name: Points, dtype: int64
Rank 0     1
1     2
2     2
3     3
4     3
5     4
6     1
7     1
8     2
9     4
10    1
11    2
Name: Rank, dtype: int64
Team 0     Riders
1     Riders
2     Devils
3     Devils
4      Kings
5      Kings
6      Kings
7      Kings
8     Riders
9     Royals
10    Royals
11    Riders
Name: Team, dtype: object
Year 0     2014
1     2015
2     2014
3     2015
4     2014
5     2015
6     2016
7     2017
8     2016
9     2014
10    2015
11    2017
Name: Year, dtype: int64
```

In [35]:
```
for row in df.itertuples():
    print (row)
```

```
Pandas(Index=0, Points=876, Rank=1, Team='Riders', Year=2014)
Pandas(Index=1, Points=789, Rank=2, Team='Riders', Year=2015)
Pandas(Index=2, Points=863, Rank=2, Team='Devils', Year=2014)
Pandas(Index=3, Points=673, Rank=3, Team='Devils', Year=2015)
Pandas(Index=4, Points=741, Rank=3, Team='Kings', Year=2014)
Pandas(Index=5, Points=812, Rank=4, Team='Kings', Year=2015)
Pandas(Index=6, Points=756, Rank=1, Team='Kings', Year=2016)
Pandas(Index=7, Points=788, Rank=1, Team='Kings', Year=2017)
Pandas(Index=8, Points=694, Rank=2, Team='Riders', Year=2016)
Pandas(Index=9, Points=701, Rank=4, Team='Royals', Year=2014)
Pandas(Index=10, Points=804, Rank=1, Team='Royals', Year=2015)
Pandas(Index=11, Points=690, Rank=2, Team='Riders', Year=2017)
```

In [36]:
```
for row_index,row in df.iterrows():
   print (row_index,row)
```

```
0 Points       876
Rank           1
Team      Riders
Year        2014
Name: 0, dtype: object
1 Points       789
Rank           2
Team      Riders
Year        2015
Name: 1, dtype: object
2 Points       863
Rank           2
Team      Devils
Year        2014
Name: 2, dtype: object
3 Points       673
Rank           3
Team      Devils
Year        2015
Name: 3, dtype: object
4 Points      741
Rank          3
Team      Kings
Year       2014
Name: 4, dtype: object
5 Points      812
Rank          4
Team      Kings
Year       2015
Name: 5, dtype: object
6 Points      756
Rank          1
Team      Kings
Year       2016
Name: 6, dtype: object
7 Points      788
Rank          1
Team      Kings
Year       2017
Name: 7, dtype: object
8 Points       694
Rank           2
Team      Riders
Year        2016
Name: 8, dtype: object
9 Points       701
Rank           4
Team      Royals
Year        2014
Name: 9, dtype: object
10 Points       804
Rank           1
Team      Royals
Year        2015
```

```
Name: 10, dtype: object
11 Points       690
Rank              2
Team         Riders
Year           2017
Name: 11, dtype: object
```

In [66]:
```
# Groupby in details

raw_data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks',
                         'Dragoons', 'Scouts', 'Scouts', 'Scouts', 'Scouts'],
        'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd','1st',
        'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner
        'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
        'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['regiment', 'company', 'name', 'preTestSco
df
```

Out[66]:

	regiment	company	name	preTestScore	postTestScore
0	Nighthawks	1st	Miller	4	25
1	Nighthawks	1st	Jacobson	24	94
2	Nighthawks	2nd	Ali	31	57
3	Nighthawks	2nd	Milner	2	62
4	Dragoons	1st	Cooze	3	70
5	Dragoons	1st	Jacon	4	25
6	Dragoons	2nd	Ryaner	24	94
7	Dragoons	2nd	Sone	31	57
8	Scouts	1st	Sloan	2	62
9	Scouts	1st	Piger	3	70
10	Scouts	2nd	Riani	2	62
11	Scouts	2nd	Ali	3	70

In [54]:
```
# Create a groupby variable that groups preTestScores by regiment
groupby_regiment = df['preTestScore'].groupby(df['regiment'])
list(groupby_regiment)
```

Out[54]:
```
[('Dragoons', 4     3
  5     4
  6    24
  7    31
  Name: preTestScore, dtype: int64), ('Nighthawks', 0     4
  1    24
  2    31
  3     2
  Name: preTestScore, dtype: int64), ('Scouts', 8     2
  9     3
  10    2
  11    3
  Name: preTestScore, dtype: int64)]
```

In [55]:
```
# Descriptive statistics by group
df['preTestScore'].groupby(df['regiment']).describe()
```

Out[55]:

regiment	count	mean	std	min	25%	50%	75%	max
Dragoons	4.0	15.50	14.153916	3.0	3.75	14.0	25.75	31.0
Nighthawks	4.0	15.25	14.453950	2.0	3.50	14.0	25.75	31.0
Scouts	4.0	2.50	0.577350	2.0	2.00	2.5	3.00	3.0

In [56]:
```
# Mean of each regiment's preTestScore
groupby_regiment.mean()
```

Out[56]:
```
regiment
Dragoons      15.50
Nighthawks    15.25
Scouts         2.50
Name: preTestScore, dtype: float64
```

In [57]:
```
# Mean preTestScores grouped by regiment and company
df['preTestScore'].groupby([df['regiment'], df['company']]).mean()
```

Out[57]:
```
regiment    company
Dragoons    1st         3.5
            2nd        27.5
Nighthawks  1st        14.0
            2nd        16.5
Scouts      1st         2.5
            2nd         2.5
Name: preTestScore, dtype: float64
```

In [58]:
```
# Mean preTestScores grouped by regiment and company without heirarchical indexin
df['preTestScore'].groupby([df['regiment'], df['company']]).mean().unstack()
```

Out[58]:

regiment	company 1st	2nd
Dragoons	3.5	27.5
Nighthawks	14.0	16.5
Scouts	2.5	2.5

In [59]:
```
# Group the entire dataframe by regiment and company
df.groupby(['regiment', 'company']).mean()
```

Out[59]:

regiment	company	preTestScore	postTestScore
Dragoons	1st	3.5	47.5
	2nd	27.5	75.5
Nighthawks	1st	14.0	59.5
	2nd	16.5	59.5
Scouts	1st	2.5	66.0
	2nd	2.5	66.0

In [60]:
```
# Number of observations in each regiment and company
df.groupby(['regiment', 'company']).size()
```

Out[60]:
```
regiment    company
Dragoons    1st        2
            2nd        2
Nighthawks  1st        2
            2nd        2
Scouts      1st        2
            2nd        2
dtype: int64
```

In [62]:
```
df.groupby('regiment').mean().add_prefix('mean_')
```

Out[62]:

regiment	mean_preTestScore	mean_postTestScore
Dragoons	15.50	61.5
Nighthawks	15.25	59.5
Scouts	2.50	66.0

In [69]:
```
# Pivots

pd.pivot_table(df, index=['regiment','company'], aggfunc='mean')
```

Out[69]:

regiment	company	postTestScore	preTestScore
Dragoons	1st	47.5	3.5
	2nd	75.5	27.5
Nighthawks	1st	59.5	14.0
	2nd	59.5	16.5
Scouts	1st	66.0	2.5
	2nd	66.0	2.5

In [82]:
```
pd.pivot_table(df,index=['regiment'],values=["preTestScore","postTestScore"],
               columns=["company"],aggfunc=[np.sum, np.mean], margins = True).sta
```

Out[82]:

		sum		mean	
		postTestScore	preTestScore	postTestScore	preTestScore
regiment	company				
	1st	95	7	47.500000	3.500000
Dragoons	2nd	151	55	75.500000	27.500000
	All	246	62	61.500000	15.500000
	1st	119	28	59.500000	14.000000
Nighthawks	2nd	119	33	59.500000	16.500000
	All	238	61	59.500000	15.250000
	1st	132	5	66.000000	2.500000
Scouts	2nd	132	5	66.000000	2.500000
	All	264	10	66.000000	2.500000
	1st	346	40	57.666667	6.666667
All	2nd	402	93	67.000000	15.500000
	All	748	133	62.333333	11.083333

In [76]:
```
raw_data = {'first_name': ['Jason', 'Jason', 'Jason','Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Miller', 'Miller','Ali', 'Milner', 'Cooze'],
        'age': [42, 42, 1111111, 36, 24, 73],
        'preTestScore': [4, 4, 4, 31, 2, 3],
        'postTestScore': [25, 25, 25, 57, 62, 70]}
d_2 = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'preTes
d_2
```

Out[76]:

	first_name	last_name	age	preTestScore	postTestScore
0	Jason	Miller	42	4	25
1	Jason	Miller	42	4	25
2	Jason	Miller	1111111	4	25
3	Tina	Ali	36	31	57
4	Jake	Milner	24	2	62
5	Amy	Cooze	73	3	70

In [38]:
```
d_2.duplicated()
```

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

In [39]:
```
d_2.drop_duplicates()
```

Out[39]:

	first_name	last_name	age	preTestScore	postTestScore
0	Jason	Miller	42	4	25
2	Jason	Miller	1111111	4	25
3	Tina	Ali	36	31	57
4	Jake	Milner	24	2	62
5	Amy	Cooze	73	3	70

In [40]:
```
d_2.drop_duplicates(['first_name'], keep='last')
```

Out[40]:

	first_name	last_name	age	preTestScore	postTestScore
2	Jason	Miller	1111111	4	25
3	Tina	Ali	36	31	57
4	Jake	Milner	24	2	62
5	Amy	Cooze	73	3	70

In [44]:
```
d_2['preTestScore'].idxmax()
```

Out[44]: 3

In [45]:
```
d_2['preTestScore'].idxmin()
```

Out[45]: 4

Merge in Pandas

In [10]:
```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A'  : ['A0', 'A1', 'A2', 'A3'],
                     'B'  : ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K3', 'K4'],
                      'C'  : ['C0', 'C1', 'C2', 'C3'],
                      'D'  : ['D0', 'D1', 'D2', 'D3']})
```

In [11]:
```
left
```

Out[11]:

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

In [12]:
```
right
```

Out[12]:

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K3
3	C3	D3	K4

In [13]:
```
# Inner Join
result = pd.merge(left, right, on=['key'])
result
```

Out[13]:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A3	B3	K3	C2	D2

In [15]:
```
# Left Join
result = pd.merge(left, right, how = 'left', on=['key'])
result
```

Out[15]:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	NaN	NaN
3	A3	B3	K3	C2	D2

In [16]:
```
# Right Join
result = pd.merge(left, right, how = 'right', on=['key'])
result
```

Out[16]:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A3	B3	K3	C2	D2
3	NaN	NaN	K4	C3	D3

In [17]:
```
# Outer Join
result = pd.merge(left, right, how = 'outer', on=['key'])
result
```

Out[17]:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	NaN	NaN
3	A3	B3	K3	C2	D2
4	NaN	NaN	K4	C3	D3

In [24]:
```
# merge indicator to track merges
result = pd.merge(left, right, how = 'outer', on=['key'], indicator = True)
result
```

Out[24]:

	A	B	key	C	D	_merge
0	A0	B0	K0	C0	D0	both
1	A1	B1	K1	C1	D1	both
2	A2	B2	K2	NaN	NaN	left_only
3	A3	B3	K3	C2	D2	both
4	NaN	NaN	K4	C3	D3	right_only

Joining on Index

In [25]:
```
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 [26]:
```
left
```

Out[26]:

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

In [27]:
```
right
```

Out[27]:

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

In [30]:
```
result = left.join(right, how = 'left')
result
```

Out[30]:

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

In [29]:
```
result = left.join(right, how='right')
result
```

Out[29]:

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

In [31]:
```
# Creating a DataFrame by passing a NumPy array, with a datetime index and labele
import pandas as pd
import numpy as np
dates = pd.date_range('20130101', periods=6)
dates
```

Out[31]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [32]:
```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

Out[32]:

	A	B	C	D
2013-01-01	-1.467781	0.686160	-1.123775	-0.477091
2013-01-02	0.042029	-0.236656	-2.078726	-0.746387
2013-01-03	0.782668	-0.789674	0.460949	-1.523262
2013-01-04	0.041406	-0.287302	0.944371	-0.439100
2013-01-05	0.115625	0.522209	-0.673775	0.868145
2013-01-06	-2.496386	0.029785	-0.942788	-1.376378

In [33]:
```
# Creating a DataFrame by passing a dict of objects that can be converted to seri

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

Out[33]:

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

Apply, Map, Applymap in Pandas

In [86]:
```
import pandas as pd
import numpy as np

data = {'Name' : ['Jack', 'Joe','Jinny','Sinchan','Nobita','Doremon'],
        'Maths': [67,56,75,89,87,99],
        'Science' : [89,67,54,78,71,97],
        'City' : ['Mumbai','Boston','Mumbai','Sydney','Paris','Auckland'],
        'Gender' : ['M','F','F','M','M','M']}
df = pd.DataFrame(data)
df
```

Out[86]:

	City	Gender	Maths	Name	Science
0	Mumbai	M	67	Jack	89
1	Boston	F	56	Joe	67
2	Mumbai	F	75	Jinny	54
3	Sydney	M	89	Sinchan	78
4	Paris	M	87	Nobita	71
5	Auckland	M	99	Doremon	97

In [46]:
```
# Apply : As the name suggests, applies a function along any axis of the DataFram

caps = lambda x : x.upper()
df['City_U'] = df['City'].apply(caps)
df
```

Out[46]:

	City	Gender	Maths	Name	Science	City_U
0	Mumbai	M	67	Jack	89	MUMBAI
1	Boston	F	56	Joe	67	BOSTON
2	Mumbai	F	75	Jinny	54	MUMBAI
3	Sydney	M	89	Sinchan	78	SYDNEY
4	Paris	M	87	Nobita	71	PARIS
5	Auckland	M	99	Doremon	97	AUCKLAND

In [47]:
```
# Performing row wise sum
df[['Maths','Science']].apply(np.sum)
```

Out[47]:
```
Maths      473
Science    456
dtype: int64
```

In [88]:
```
# Performing column wise sum
df['Total'] = df[['Maths','Science']].apply(np.sum,axis = 1)
df
```

Out[88]:

	City	Gender	Maths	Name	Science	Total
0	Mumbai	M	67	Jack	89	156
1	Boston	F	56	Joe	67	123
2	Mumbai	F	75	Jinny	54	129
3	Sydney	M	89	Sinchan	78	167
4	Paris	M	87	Nobita	71	158
5	Auckland	M	99	Doremon	97	196

In [93]:
```
df['Ranking'] = df['Total'].rank(ascending = 0)
df
```

Out[93]:

	City	Gender	Maths	Name	Science	Total	Ranking
0	Mumbai	M	67	Jack	89	156	4.0
1	Boston	F	56	Joe	67	123	6.0
2	Mumbai	F	75	Jinny	54	129	5.0
3	Sydney	M	89	Sinchan	78	167	2.0
4	Paris	M	87	Nobita	71	158	3.0
5	Auckland	M	99	Doremon	97	196	1.0

In [62]:
```
# Map() : map applies a function on any column or index
df['Sex_num'] = df['Gender'].map({'F':0, 'M':1})
df
```

Out[62]:

	City	Gender	Maths	Name	Science	City_U	Total	Sex_num
0	Mumbai	M	67	Jack	89	MUMBAI	156	1
1	Boston	F	56	Joe	67	BOSTON	123	0
2	Mumbai	F	75	Jinny	54	MUMBAI	129	0
3	Sydney	M	89	Sinchan	78	SYDNEY	167	1
4	Paris	M	87	Nobita	71	PARIS	158	1
5	Auckland	M	99	Doremon	97	AUCKLAND	196	1

In [64]:
```
# Applymap : applies a function to every single element in the entire dataframe.
df[['Maths','Science']].applymap(lambda x : x + 2)
```

Out[64]:

	Maths	Science
0	69	91
1	58	69
2	77	56
3	91	80
4	89	73
5	101	99

In [70]:
```
# Working with text data
s = pd.Series(['Tom', 'William Rick', 'John', 'Alber@t', np.nan, '1234','SteveSmi
s
```

Out[70]:
```
0            Tom
1    William Rick
2           John
3         Alber@t
4            NaN
5           1234
6     SteveSmith
dtype: object
```

In [56]:
```
print (s.str.lower())
```

```
0            tom
1    william rick
2           john
3         alber@t
4            NaN
5           1234
6     stevesmith
dtype: object
```

In [57]:
```
print (s.str.upper())
```

```
0             TOM
1    WILLIAM RICK
2            JOHN
3         ALBER@T
4             NaN
5            1234
6      STEVESMITH
dtype: object
```

In [58]:
```
print (s.str.len())
```

```
0     3.0
1    12.0
2     4.0
3     7.0
4     NaN
5     4.0
6    10.0
dtype: float64
```

In [59]:
```
print (s.str.cat(sep='_'))
```

```
Tom_William Rick_John_Alber@t_1234_SteveSmith
```

In [61]:
```
print (s.str.contains('Rick'))
```

```
0    False
1     True
2    False
3    False
4      NaN
5    False
6    False
dtype: object
```

In [62]:
```
print ("After replacing @ with $:")
print (s.str.replace('@','$'))
```

```
After replacing @ with $:
0             Tom
1    William Rick
2            John
3         Alber$t
4             NaN
5            1234
6      SteveSmith
dtype: object
```

In [63]:
```
print (s.str.repeat(2))
```

```
0                       TomTom
1    William RickWilliam Rick
2                      JohnJohn
3                Alber@tAlber@t
4                          NaN
5                     12341234
6          SteveSmithSteveSmith
dtype: object
```

In [64]:
```
print ("The number of 'm's in each string:")
print (s.str.count('m'))
```

```
The number of 'm's in each string:
0    1.0
1    1.0
2    0.0
3    0.0
4    NaN
5    0.0
6    1.0
dtype: float64
```

In [71]:
```
print ("Strings that start with 'T':")
print (s.str.startswith ('T'))
```

```
Strings that start with 'T':
0     True
1    False
2    False
3    False
4      NaN
5    False
6    False
dtype: object
```

In [72]:
```
print ("Strings that end with 't':")
print (s.str.endswith('t'))
```

```
Strings that end with 't':
0    False
1    False
2    False
3     True
4      NaN
5    False
6    False
dtype: object
```

In [73]:
```
print (s.str.find('e')) # Returns the first position of the first occurrence of t
```

```
0   -1.0
1   -1.0
2   -1.0
3    3.0
4    NaN
5   -1.0
6    2.0
dtype: float64
```

In [74]:
```
print (s.str.swapcase())
```

```
0             tOM
1    wILLIAM rICK
2            jOHN
3         aLBER@T
4             NaN
5            1234
6      sTEVEsMITH
dtype: object
```

In [75]:
```
print (s.str.islower())
```

```
0    False
1    False
2    False
3    False
4      NaN
5    False
6    False
dtype: object
```

In [76]:
```
print (s.str.isupper())
```

```
0    False
1    False
2    False
3    False
4      NaN
5    False
6    False
dtype: object
```

In [77]:
```
print (s.str.isnumeric())
```

```
0    False
1    False
2    False
3    False
4      NaN
5     True
6    False
dtype: object
```

In [78]:
```
print (pd.datetime.now()) # Get Current Time
```

```
2018-08-10 18:20:42.377585
```

In [79]:
```
print (pd.date_range('1/1/2011', periods=5)) # Create a Range of Dates
```

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-04',
               '2011-01-05'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:
```

```