Prepared by Asif Bhat

Pandas - Series & Dataframes

In [55]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
import re
import math
```

In [54]:

```
import warnings
warnings.filterwarnings("ignore")
```

Series

Create Series

In [61]:

```
# Create series from Nump Array
v = np.array([1,2,3,4,5,6,7])
s1 = pd.Series(v)
s1
```

Out[61]:

```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
dtype: int32
```

In [62]:

```
#Datatype of Series
s1.dtype
```

Out[62]:

```
dtype('int32')
```

In [431]:

```
# number of bytes allocated to each item
s1.itemsize
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWar
ning: Series.itemsize is deprecated and will be removed in a future version
```

Out[431]:

```
4
```

In [64]:

```
# Number of bytes consumed by Series
s1.nbytes
```

Out[64]:

```
28
```

In [65]:

```
# Shape of the Series
s1.shape
```

Out[65]:

```
(7,)
```

In [66]:

```
# number of dimensions
s1.ndim
```

Out[66]:

```
1
```

In [67]:

```
# Length of Series
len(s1)
```

Out[67]:

```
7
```

In [68]:

```
s1.count()
```

Out[68]:

```
7
```

In [69]:

```
s1.size
```

Out[69]:

```
7
```

In [70]:

```
# Create series from List
s0 = pd.Series([1,2,3],index = ['a','b','c'])
s0
```

Out[70]:

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

In [71]:

```
# Modifying index in Series
s1.index = ['a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g']
s1
```

Out[71]:

```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
dtype: int32
```

In [432]:

```
# Create Series using Random and Range function
v2 = np.random.random(10)
ind2 = np.arange(0,10)
s = pd.Series(v2,ind2)
v2 , ind2 , s
```

Out[432]:

```
(array([0.87790351, 0.21256923, 0.2833476 , 0.84976498, 0.17274437,
        0.36953613, 0.92661933, 0.13005525, 0.25394528, 0.43563311]),
 array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 0    0.877904
 1    0.212569
 2    0.283348
 3    0.849765
 4    0.172744
 5    0.369536
 6    0.926619
 7    0.130055
 8    0.253945
 9    0.435633
 dtype: float64)
```

In [433]:

```
# Creating Series from Dictionary
dict1 = {'a1' :10 , 'a2' :20 , 'a3':30 , 'a4':40}
s3 = pd.Series(dict1)
s3
```

Out[433]:

```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [434]:

```
pd.Series(99, index=[0, 1, 2, 3, 4, 5])
```

Out[434]:

```
0    99
1    99
2    99
3    99
4    99
5    99
dtype: int64
```

Slicing Series

In [435]:

```
s
```

Out[435]:

```
0    0.877904
1    0.212569
2    0.283348
3    0.849765
4    0.172744
5    0.369536
6    0.926619
7    0.130055
8    0.253945
9    0.435633
dtype: float64
```

In [436]:

```
# Return all elements of the series
s[:]
```

Out[436]:

```
0    0.877904
1    0.212569
2    0.283348
3    0.849765
4    0.172744
5    0.369536
6    0.926619
7    0.130055
8    0.253945
9    0.435633
dtype: float64
```

In [439]:

```
# First three element of the Series
s[0:3]
```

Out[439]:

```
0    0.877904
1    0.212569
2    0.283348
dtype: float64
```

In [440]:

```
# Last element of the Series
s[-1:]
```

Out[440]:

```
9    0.435633
dtype: float64
```

In [441]:

```
# Fetch first 4 elements in a series
s[:4]
```

Out[441]:

```
0    0.877904
1    0.212569
2    0.283348
3    0.849765
dtype: float64
```

In [442]:

```
# Return all elements of the series except last two elements.
s[:-2]
```

Out[442]:

```
0    0.877904
1    0.212569
2    0.283348
3    0.849765
4    0.172744
5    0.369536
6    0.926619
7    0.130055
dtype: float64
```

In [443]:

```
# Return all elements of the series except last element.
s[:-1]
```

Out[443]:

```
0    0.877904
1    0.212569
2    0.283348
3    0.849765
4    0.172744
5    0.369536
6    0.926619
7    0.130055
8    0.253945
dtype: float64
```

In [444]:

```
# Return last two elements of the series
s[-2:]
```

Out[444]:

```
8    0.253945
9    0.435633
dtype: float64
```

In [445]:

```
# # Return last element of the series
s[-1:]
```

Out[445]:

```
9    0.435633
dtype: float64
```

In [446]:

```
s[-3:-1]
```

Out[446]:

```
7    0.130055
8    0.253945
dtype: float64
```

Append Series

In [477]:

```
s2 = s1.copy()
s2
```

Out[477]:

```
0    10
1    20
2    30
dtype: int32
```

In [478]:

```
s3
```

Out[478]:

```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [479]:

```
# Append S2 & S3 Series
s4 = s2.append(s3)
s4
```

Out[479]:

```
0     10
1     20
2     30
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [480]:

```
# When "inplace=False" it will return a new copy of data with the operation performed
s4.drop('a4' , inplace=False)
```

Out[480]:

```
0     10
1     20
2     30
a1    10
a2    20
a3    30
dtype: int64
```

In [481]:

```
s4
```

Out[481]:

```
0     10
1     20
2     30
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [482]:

```
# When we use "inplace=True" it will affect the dataframe
s4.drop('a4', inplace=True)
s4
```

Out[482]:

```
0     10
1     20
2     30
a1    10
a2    20
a3    30
dtype: int64
```

In [483]:

```
s4 = s4.append(pd.Series({'a4': 7}))
s4
```

Out[483]:

```
0     10
1     20
2     30
a1    10
a2    20
a3    30
a4     7
dtype: int64
```

Operation on Series

In [484]:

```
v1 = np.array([10,20,30])
v2 = np.array([1,2,3])
s1 = pd.Series(v1)
s2 = pd.Series(v2)
s1 , s2
```

Out[484]:

```
(0    10
 1    20
 2    30
 dtype: int32, 0    1
 1    2
 2    3
 dtype: int32)
```

In [485]:

```
# Addition of two series
s1.add(s2)
```

Out[485]:

```
0    11
1    22
2    33
dtype: int32
```

In [486]:

```
# Subtraction of two series
s1.sub(s2)
```

Out[486]:

```
0     9
1    18
2    27
dtype: int32
```

In [487]:

```
# Subtraction of two series
s1.subtract(s2)
```

Out[487]:

```
0     9
1    18
2    27
dtype: int32
```

In [488]:

```
# Increment all numbers in a series by 9
s1.add(9)
```

Out[488]:

```
0    19
1    29
2    39
dtype: int32
```

In [489]:

```
# Multiplication of two series
s1.mul(s2)
```

Out[489]:

```
0    10
1    40
2    90
dtype: int32
```

In [490]:

```
# Multiplication of two series
s1.multiply(s2)
```

Out[490]:

```
0    10
1    40
2    90
dtype: int32
```

In [491]:

```
# Multiply each element by 1000
s1.multiply(1000)
```

Out[491]:

```
0    10000
1    20000
2    30000
dtype: int32
```

In [492]:

```
# Division
s1.divide(s2)
```

Out[492]:

```
0    10.0
1    10.0
2    10.0
dtype: float64
```

In [493]:

```
# Division
s1.div(s2)
```

Out[493]:

```
0    10.0
1    10.0
2    10.0
dtype: float64
```

In [494]:

```
# MAX number in a series
s1.max()
```

Out[494]:

```
30
```

In [495]:

```
# Min number in a series
s1.min()
```

Out[495]:

```
10
```

In [496]:

```
# Average
s1.mean()
```

Out[496]:

```
20.0
```

In [497]:

```
# Median
s1.median()
```

Out[497]:

```
20.0
```

In [498]:

```
# Standard Deviation
s1.std()
```

Out[498]:

```
10.0
```

In [499]:

```
# Series comparison
s1.equals(s2)
```

Out[499]:

```
False
```

In [500]:

```
s4 =s1
```

In [501]:

```
# Series comparison
s1.equals(s4)
```

Out[501]:

```
True
```

In [502]:

```
s5 = pd.Series([1,1,2,2,3,3], index=[0, 1, 2, 3, 4, 5])
s5
```

Out[502]:

```
0    1
1    1
2    2
3    2
4    3
5    3
dtype: int64
```

In [503]:

```
s5.value_counts()
```

Out[503]:

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

DataFrame

Create DataFrame

In [2]:

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

Out[2]:

—

In [3]:

```
# Create Dataframe using List
lang = ['Java' , 'Python' , 'C' , 'C++']
df = pd.DataFrame(lang)
df
```

Out[3]:

	0
0	Java
1	Python
2	C
3	C++

In [4]:

```
# Add column in the Dataframe
rating = [1,2,3,4]
df[1] = rating
df
```

Out[4]:

	0	1
0	Java	1
1	Python	2
2	C	3
3	C++	4

In [5]:

```
df.columns = ['Language','Rating']
```

In [6]:

```
df
```

Out[6]:

	Language	Rating
0	Java	1
1	Python	2
2	C	3
3	C++	4

In [509]:

```
# Create Dataframe from Dictionary

data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df2 = pd.DataFrame(data)
df3 = pd.DataFrame(data, index=['row1', 'row2'], columns=['a', 'b'])
df4 = pd.DataFrame(data, index=['row1', 'row2'], columns=['a', 'b' ,'c'])
df5 = pd.DataFrame(data, index=['row1', 'row2'], columns=['a', 'b' ,'c' , 'd'])
```

In [510]:

```
df2
```

Out[510]:

	a	b	c
0	1	2	NaN
1	5	10	20.0

In [511]:

```
df3
```

Out[511]:

	a	b
row1	1	2
row2	5	10

In [512]:

```
df4
```

Out[512]:

	a	b	c
row1	1	2	NaN
row2	5	10	20.0

In [513]:

```
df5
```

Out[513]:

	a	b	c	d
row1	1	2	NaN	NaN
row2	5	10	20.0	NaN

In [514]:

```
# Create Dataframe from Dictionary
df0 = pd.DataFrame({'ID' :[1,2,3,4] , 'Name' :['Asif' , 'Basit' , 'Ross' , 'John']})
df0
```

Out[514]:

	ID	Name
0	1	Asif
1	2	Basit
2	3	Ross
3	4	John

In [515]:

```
# Create a DataFrame from Dictionary of Series
dict = {'A' : pd.Series([1, 2, 3],    index=['a', 'b', 'c']),
        'B' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df1 = pd.DataFrame(dict)
df1
```

Out[515]:

	A	B
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Dataframe of Random Numbers with Date Indices

In [516]:

```
dates = pd.date_range(start='2020-01-20', end='2020-01-26')
dates
```

Out[516]:

```
DatetimeIndex(['2020-01-20', '2020-01-21', '2020-01-22', '2020-01-23',
               '2020-01-24', '2020-01-25', '2020-01-26'],
              dtype='datetime64[ns]', freq='D')
```

In [517]:

```
dates = pd.date_range('today',periods= 7)
dates
```

Out[517]:

```
DatetimeIndex(['2020-03-26 21:12:28.054030', '2020-03-27 21:12:28.054030',
               '2020-03-28 21:12:28.054030', '2020-03-29 21:12:28.054030',
               '2020-03-30 21:12:28.054030', '2020-03-31 21:12:28.054030',
               '2020-04-01 21:12:28.054030'],
              dtype='datetime64[ns]', freq='D')
```

In [518]:

```
dates = pd.date_range(start='2020-01-20', periods=7)
dates
```

Out[518]:

```
DatetimeIndex(['2020-01-20', '2020-01-21', '2020-01-22', '2020-01-23',
               '2020-01-24', '2020-01-25', '2020-01-26'],
              dtype='datetime64[ns]', freq='D')
```

In [519]:

```
M = np.random.random((7,7))
M
```

Out[519]:

```
array([[0.4622746 , 0.89035943, 0.71642701, 0.84377142, 0.49755232,
        0.11045011, 0.58396628],
       [0.17482429, 0.87319772, 0.45684689, 0.84031995, 0.78331096,
        0.31403177, 0.47437109],
       [0.93515504, 0.54242672, 0.22759177, 0.96704986, 0.56430298,
        0.57963586, 0.65763753],
       [0.75272979, 0.96463094, 0.4276211 , 0.25767407, 0.55057963,
        0.32127381, 0.39603304],
       [0.0319823 , 0.05349957, 0.77580459, 0.03393895, 0.0837259 ,
        0.71941967, 0.61385342],
       [0.34172216, 0.4961929 , 0.06987849, 0.27205465, 0.66536559,
        0.44655804, 0.28030833],
       [0.05618655, 0.29012725, 0.12826893, 0.62608765, 0.79321883,
        0.22290462, 0.52250865]])
```

In [520]:

```
dframe = pd.DataFrame(M , index=dates)
dframe
```

Out[520]:

	0	1	2	3	4	5	6
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	0.031982	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	0.341722	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	0.056187	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [521]:

```
#Changing Column Names
dframe.columns = ['C1' , 'C2' , 'C3', 'C4', 'C5', 'C6', 'C7']
dframe
```

Out[521]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	0.031982	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	0.341722	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	0.056187	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [522]:

```
# List Index
dframe.index
```

Out[522]:

```
DatetimeIndex(['2020-01-20', '2020-01-21', '2020-01-22', '2020-01-23',
               '2020-01-24', '2020-01-25', '2020-01-26'],
              dtype='datetime64[ns]', freq='D')
```

In [523]:

```
# List Column Names
dframe.columns
```

Out[523]:

```
Index(['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7'], dtype='object')
```

In [524]:

```
# Datatype of each column
dframe.dtypes
```

Out[524]:

```
C1    float64
C2    float64
C3    float64
C4    float64
C5    float64
C6    float64
C7    float64
dtype: object
```

In [525]:

```
# Sort Dataframe by Column 'C1' in Ascending Order
dframe.sort_values(by='C1')
```

Out[525]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-24	0.031982	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-26	0.056187	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-25	0.341722	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638

In [526]:

```
# Sort Dataframe by Column 'C1' in Descending Order
dframe.sort_values(by='C1' , ascending=False)
```

Out[526]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-25	0.341722	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-26	0.056187	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509
2020-01-24	0.031982	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853

Delete Column in DataFrame

In [527]:

```
df1
```

Out[527]:

	A	B
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

In [528]:

```
# Delete Column using "del" function
del df1['B']
```

In [529]:

```
df1
```

Out[529]:

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [530]:

```
df5
```

Out[530]:

	a	b	c	d
row1	1	2	NaN	NaN
row2	5	10	20.0	NaN

In [531]:

```
# Delete Column using pop()
df5.pop('c')
```

Out[531]:

```
row1     NaN
row2    20.0
Name: c, dtype: float64
```

In [532]:

```
df5
```

Out[532]:

	a	b	d
row1	1	2	NaN
row2	5	10	NaN

In [87]:

```
dict = {'A' : pd.Series([1, 2, 3,11],    index=['a', 'b', 'c','d']),
        'B' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df12 = pd.DataFrame(dict)
df12
```

Out[87]:

	A	B
a	1	1
b	2	2
c	3	3
d	11	4

In [89]:

```
df12.drop(['A'], axis=1,inplace=True)
df12
```

Out[89]:

	B
a	1
b	2
c	3
d	4

Delete Rows in DataFrame

In [171]:

```
col1 = np.linspace(10, 100, 30)
col2 = np.random.randint(10,100,30)
df10 = pd.DataFrame({"C1" : col1 , "C2" :col2})
df10
```

Out[171]:

	C1	C2
0	10.000000	63
1	13.103448	24
2	16.206897	62
3	19.310345	48
4	22.413793	42
5	25.517241	28
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
16	59.655172	28
17	62.758621	48
18	65.862069	95
19	68.965517	57
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11
26	90.689655	31
27	93.793103	75
28	96.896552	80
29	100.000000	17

In [172]:

```
# Delete rows with index values 17,18,19
df10 = df10.drop([17,18,19], axis=0)
df10
```

Out[172]:

	C1	C2
0	10.000000	63
1	13.103448	24
2	16.206897	62
3	19.310345	48
4	22.413793	42
5	25.517241	28
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
16	59.655172	28
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11
26	90.689655	31
27	93.793103	75
28	96.896552	80
29	100.000000	17

In [173]:

```
# Delete rows with index values 16 without using assignment operation
df10.drop([16], axis=0,inplace=True)
df10
```

Out[173]:

	C1	C2
0	10.000000	63
1	13.103448	24
2	16.206897	62
3	19.310345	48
4	22.413793	42
5	25.517241	28
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11
26	90.689655	31
27	93.793103	75
28	96.896552	80
29	100.000000	17

In [174]:

```
df10.drop(df10.index[5] , inplace=True)
df10
```

Out[174]:

	C1	C2
0	10.000000	63
1	13.103448	24
2	16.206897	62
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11
26	90.689655	31
27	93.793103	75
28	96.896552	80
29	100.000000	17

In [175]:

```
#Delete first three rows
df10 = df10.iloc[3:,]
df10
```

Out[175]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11
26	90.689655	31
27	93.793103	75
28	96.896552	80
29	100.000000	17

In [176]:

```
#Delete last four rows
df10 = df10.iloc[:-4,]
df10
```

Out[176]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79
14	53.448276	20
15	56.551724	42
20	72.068966	64
21	75.172414	20
22	78.275862	74
23	81.379310	45
24	84.482759	92
25	87.586207	11

In [177]:

```
#Keep top 10 rows
df10 = df10.iloc[:10,]
df10
```

Out[177]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79

In [178]:

```
df10
```

Out[178]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
12	47.241379	56
13	50.344828	79

In [179]:

```
df10.index[df10['C2'] == 56].tolist()
```

Out[179]:

```
[12]
```

In [180]:

```
# Delete row based on Column value
df10.drop(df10.index[df10['C2'] == 56].tolist() , axis=0,inplace=True)
df10
```

Out[180]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21
13	50.344828	79

In [181]:

```
# Delete row based on Column value
df10 = df10.drop(df10[df10["C2"]==79].index)
df10
```

Out[181]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
7	31.724138	44
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21

In [182]:

```
# Delete all rows with column C2 value 14
df10 = df10[df10.C2 != 44]
df10
```

Out[182]:

	C1	C2
3	19.310345	48
4	22.413793	42
6	28.620690	55
8	34.827586	21
9	37.931034	52
10	41.034483	76
11	44.137931	21

In [183]:

```
# Delete all rows with column C2 value 88 & 55 using isin operator
df10 = df10[~(df10.C2.isin ([21,48]))]
df10
```

Out[183]:

	C1	C2
4	22.413793	42
6	28.620690	55
9	37.931034	52
10	41.034483	76

In [184]:

```
# Keep all rows with column C2 value 10,89,31 & 64 using isin operator
df10 = df10[df10.C2.isin ([42,76])]
df10
```

Out[184]:

	C1	C2
4	22.413793	42
10	41.034483	76

In [186]:

```
dict = {'A' : pd.Series([1, 2, 3,11],    index=['a', 'b', 'c','d']),
        'B' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df11 = pd.DataFrame(dict)
df11
```

Out[186]:

	A	B
a	1	1
b	2	2
c	3	3
d	11	4

In [187]:

```
#Delete all rows with label "d"
df11.drop("d", axis=0,inplace=True)
df11
```

Out[187]:

	A	B
a	1	1
b	2	2
c	3	3

In [188]:

```
df13 = pd.DataFrame({ 'ID' :[1,2,3,4] ,
                      'Name' :['Asif' , 'Basit' , 'Ross' , 'John'] ,
                      'location' : ['India' , 'Australia','UK' , 'US'] })
df13
```

Out[188]:

	ID	Name	location
0	1	Asif	India
1	2	Basit	Australia
2	3	Ross	UK
3	4	John	US

In [193]:

```
ind = df13[((df13.Name == 'Ross') &(df13.ID == 3) & (df13.location == 'UK'))].index
df13.drop(ind,inplace=True)
df13
```

Out[193]:

	ID	Name	location
0	1	Asif	India
1	2	Basit	Australia
3	4	John	US

Data Selection in Dataframe

In [533]:

```
df
```

Out[533]:

	Language	Rating
0	Java	1
1	Python	2
2	C	3
3	C++	4

In [534]:

```
df.index = [1,2,3,4]
df
```

Out[534]:

	Language	Rating
1	Java	1
2	Python	2
3	C	3
4	C++	4

In [535]:

```
# Data selection using row label
df.loc[1]
```

Out[535]:

```
Language    Java
Rating         1
Name: 1, dtype: object
```

In [536]:

```
# Data selection using position (Integer Index based)
df.iloc[1]
```

Out[536]:

```
Language    Python
Rating           2
Name: 2, dtype: object
```

In [537]:

```
df.loc[1:2]
```

Out[537]:

	Language	Rating
1	Java	1
2	Python	2

In [538]:

```
df.iloc[1:2]
```

Out[538]:

	Language	Rating
2	Python	2

In [539]:

```
# Data selection based on Condition
df.loc[df.Rating > 2]
```

Out[539]:

	Language	Rating
3	C	3
4	C++	4

In [540]:

```
df1
```

Out[540]:

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [541]:

```
# Row & Column label based selection
df1.loc['a']
```

Out[541]:

```
A    1.0
Name: a, dtype: float64
```

In [542]:

```
df1.iloc['a'] # This will throw error because iloc will not work on labels
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-542-808cc54286b0> in <module>()
----> 1 df1.iloc['a'] # This will throw error because iloc will not work on
 labels

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, k
ey)
   1476 
   1477             maybe_callable = com._apply_if_callable(key, self.obj)
-> 1478             return self._getitem_axis(maybe_callable, axis=axis)
   1479 
   1480     def _is_scalar_access(self, key):

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self,
key, axis)
   2093         # a single integer
   2094         else:
-> 2095             key = self._convert_scalar_indexer(key, axis)
   2096 
   2097             if not is_integer(key):

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in _convert_scalar_ind
exer(self, key, axis)
    259         ax = self.obj._get_axis(min(axis, self.ndim - 1))
    260         # a scalar
--> 261         return ax._convert_scalar_indexer(key, kind=self.name)
    262 
    263     def _convert_slice_indexer(self, key, axis):

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _convert_scalar
_indexer(self, key, kind)
   1650 
   1651         if kind == 'iloc':
-> 1652             return self._validate_indexer('positional', key, kind)
   1653 
   1654         if len(self) and not isinstance(self, ABCMultiIndex,):

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _validate_index
er(self, form, key, kind)
   4143             pass
   4144         elif kind in ['iloc', 'getitem']:
-> 4145             self._invalid_indexer(form, key)
   4146         return key
   4147 

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _invalid_indexe
r(self, form, key)
   1861                         "indexers [{key}] of {kind}".format(
   1862                             form=form, klass=type(self), key=key,
-> 1863                             kind=type(key)))
   1864 
   1865     def get_duplicates(self):

TypeError: cannot do positional indexing on <class 'pandas.core.indexes.bas
e.Index'> with these indexers [a] of <class 'str'>
```

In [544]:

```
dframe
```

Out[544]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	0.031982	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	0.341722	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	0.056187	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [545]:

```
# Data selection using Row Label
dframe['2020-01-20' : '2020-01-22' ]
```

Out[545]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.462275	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-21	0.174824	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638

In [546]:

```
# Selecting all rows & selected columns
dframe.loc[:,['C1' , 'C7']]
```

Out[546]:

	C1	C7
2020-01-20	0.462275	0.583966
2020-01-21	0.174824	0.474371
2020-01-22	0.935155	0.657638
2020-01-23	0.752730	0.396033
2020-01-24	0.031982	0.613853
2020-01-25	0.341722	0.280308
2020-01-26	0.056187	0.522509

In [547]:

```
#row & column label based selection
dframe.loc['2020-01-20' : '2020-01-22',['C1' , 'C7']]
```

Out[547]:

	C1	C7
2020-01-20	0.462275	0.583966
2020-01-21	0.174824	0.474371
2020-01-22	0.935155	0.657638

In [548]:

```
# Data selection based on Condition
dframe[dframe['C1'] > 0.5]
```

Out[548]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-22	0.935155	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	0.752730	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033

In [549]:

```
# Data selection based on Condition
dframe[(dframe['C1'] > 0.5) & (dframe['C4'] > 0.5)]
```

Out[549]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-22	0.935155	0.542427	0.227592	0.96705	0.564303	0.579636	0.657638

In [550]:

```
# Data selection using position (Integer Index based)
dframe.iloc[0][0]
```

Out[550]:

```
0.46227460203265247
```

In [551]:

```
# Select all rows & first three columns
dframe.iloc[:,0:3]
```

Out[551]:

	C1	C2	C3
2020-01-20	0.462275	0.890359	0.716427
2020-01-21	0.174824	0.873198	0.456847
2020-01-22	0.935155	0.542427	0.227592
2020-01-23	0.752730	0.964631	0.427621
2020-01-24	0.031982	0.053500	0.775805
2020-01-25	0.341722	0.496193	0.069878
2020-01-26	0.056187	0.290127	0.128269

In [552]:

```
dframe.iloc[0][0] = 10
```

In [554]:

```
# Display all rows where C1 has value of 10 or 20
dframe[dframe['C1'].isin([10,20])]
```

Out[554]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	10.0	0.890359	0.716427	0.843771	0.497552	0.11045	0.583966

Set Value

In [555]:

```
# Set value of 888 for all elements in column 'C1'
dframe['C1'] = 888
dframe
```

Out[555]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	0.716427	0.843771	0.497552	0.110450	0.583966
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	0.314032	0.474371
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	0.579636	0.657638
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [556]:

```
# Set value of 777 for first three rows in Column 'C6'
dframe.at[0:3,'C6'] = 777
```

In [557]:

```
dframe
```

Out[557]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	0.716427	0.843771	0.497552	777.000000	0.583966
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	777.000000	0.474371
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	0.657638
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [564]:

```
# Set value of 333 in first row and third column
dframe.iat[0,2] = 333
```

In [565]:

```
dframe
```

Out[565]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	333.000000	0.843771	0.497552	777.000000	0.583966
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	777.000000	0.474371
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	0.657638
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [566]:

```
dframe.iloc[0,2] = 555
dframe
```

Out[566]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	777.000000	0.583966
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	777.000000	0.474371
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	0.657638
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [567]:

```
# Create Copy of the calling objects data along with indices.
# Modifications to the data or indices of the copy will not be reflected in the original ob
dframe1 = dframe.copy(deep=True)
```

In [568]:

```
dframe1[(dframe1['C1'] > 0.5) & (dframe1['C4'] > 0.5)] = 0
```

In [569]:

```
dframe1[dframe1['C1'] == 0]
```

Out[569]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0	0.0	0.0	0.0	0.0	0.0	0.0
2020-01-21	0	0.0	0.0	0.0	0.0	0.0	0.0
2020-01-22	0	0.0	0.0	0.0	0.0	0.0	0.0
2020-01-26	0	0.0	0.0	0.0	0.0	0.0	0.0

In [570]:

```
# Replace zeros in Column C1 with 99
dframe1[dframe1['C1'].isin([0])] = 99
dframe1
```

Out[570]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2020-01-21	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2020-01-22	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	99	99.000000	99.000000	99.000000	99.000000	99.000000	99.000000

In [571]:

```
dframe
```

Out[571]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	777.000000	0.583966
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	777.000000	0.474371
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	0.657638
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	0.396033
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	0.613853
2020-01-25	888	0.496193	0.069878	0.272055	0.665366	0.446558	0.280308
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	0.522509

In [572]:

```
# Display all rows where value of C1 is 99
dframe1[dframe1['C1'] == 99]
```

Out[572]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	99	99.0	99.0	99.0	99.0	99.0	99.0
2020-01-21	99	99.0	99.0	99.0	99.0	99.0	99.0
2020-01-22	99	99.0	99.0	99.0	99.0	99.0	99.0
2020-01-26	99	99.0	99.0	99.0	99.0	99.0	99.0

Dealing with NULL Values

In [573]:

```
dframe.at[0:8 , 'C7'] = np.NaN
dframe.at[0:2 , 'C6'] = np.NaN
dframe.at[5:6 , 'C5'] = np.NaN
dframe
```

Out[573]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	NaN	NaN
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	NaN	NaN
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-25	888	0.496193	0.069878	0.272055	NaN	0.446558	NaN
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	NaN

In [574]:

```
# Detect Non-Missing Values
# It will return True for NOT-NULL values and False for NULL values
dframe.notna()
```

Out[574]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	True	True	True	True	True	False	False
2020-01-21	True	True	True	True	True	False	False
2020-01-22	True	True	True	True	True	True	False
2020-01-23	True	True	True	True	True	True	False
2020-01-24	True	True	True	True	True	True	False
2020-01-25	True	True	True	True	False	True	False
2020-01-26	True	True	True	True	True	True	False

In [575]:

```
# Detect Missing or NULL Values
# It will return True for NULL values and False for NOT-NULL values
dframe.isna()
```

Out[575]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	False	False	False	False	False	True	True
2020-01-21	False	False	False	False	False	True	True
2020-01-22	False	False	False	False	False	False	True
2020-01-23	False	False	False	False	False	False	True
2020-01-24	False	False	False	False	False	False	True
2020-01-25	False	False	False	False	True	False	True
2020-01-26	False	False	False	False	False	False	True

In [576]:

```
# Fill all NULL values with 1020
dframe = dframe.fillna(1020)
dframe
```

Out[576]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	1020.000000	1020.0
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	1020.000000	1020.0
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	1020.0
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	1020.0
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	1020.0
2020-01-25	888	0.496193	0.069878	0.272055	1020.000000	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [577]:

```
dframe.at[0:5 , 'C7'] = np.NaN
dframe.at[0:2 , 'C6'] = np.NaN
dframe.at[5:6 , 'C5'] = np.NaN
dframe
```

Out[577]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	NaN	NaN
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	NaN	NaN
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-25	888	0.496193	0.069878	0.272055	NaN	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [578]:

```
# Replace Null values in Column 'C5' with number 123
# Replace Null values in Column 'C6' with number 789
dframe.fillna(value={'C5' : 123 , 'C6' : 789})
```

Out[578]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	789.000000	NaN
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	789.000000	NaN
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-25	888	0.496193	0.069878	0.272055	123.000000	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [579]:

```
#Replace first NULL value in Column C7 with 789
dframe.fillna(value={'C7' : 789} , limit=1)
```

Out[579]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	NaN	789.0
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	NaN	NaN
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-25	888	0.496193	0.069878	0.272055	NaN	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [580]:

```
# Drop Rows with NULL values
dframe.dropna()
```

Out[580]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [581]:

```
# Drop Columns with NULL values
dframe.dropna(axis='columns')
```

Out[581]:

	C1	C2	C3	C4
2020-01-20	888	0.890359	555.000000	0.843771
2020-01-21	888	0.873198	0.456847	0.840320
2020-01-22	888	0.542427	0.227592	0.967050
2020-01-23	888	0.964631	0.427621	0.257674
2020-01-24	888	0.053500	0.775805	0.033939
2020-01-25	888	0.496193	0.069878	0.272055
2020-01-26	888	0.290127	0.128269	0.626088

In [582]:

```
dframe
```

Out[582]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	NaN	NaN
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	NaN	NaN
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-25	888	0.496193	0.069878	0.272055	NaN	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [583]:

```
# Drop Rows with NULL values present in C5 or C6
dframe.dropna(subset=['C5' ,'C6'])
```

Out[583]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	NaN
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	NaN
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	NaN
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

Descriptive Statistics

In [584]:

```
# Fill NULL values with 55
dframe.fillna(55 , inplace=True)
dframe
```

Out[584]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	55.000000	55.0
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	55.000000	55.0
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	55.0
2020-01-23	888	0.964631	0.427621	0.257674	0.550580	0.321274	55.0
2020-01-24	888	0.053500	0.775805	0.033939	0.083726	0.719420	55.0
2020-01-25	888	0.496193	0.069878	0.272055	55.000000	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [585]:

```
# Mean of all Columns
dframe.mean()
```

Out[585]:

```
C1    888.000000
C2      0.587205
C3     79.583716
C4      0.548700
C5      8.324670
C6    126.958594
C7    330.714286
dtype: float64
```

In [586]:

```
# Max value per column
dframe.max()
```

Out[586]:

```
C1     888.000000
C2       0.964631
C3     555.000000
C4       0.967050
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [587]:

```
# Min value per column
dframe.min()
```

Out[587]:

```
C1    888.000000
C2      0.053500
C3      0.069878
C4      0.033939
C5      0.083726
C6      0.222905
C7     55.000000
dtype: float64
```

In [588]:

```
# Median
dframe.median()
```

Out[588]:

```
C1    888.000000
C2      0.542427
C3      0.427621
C4      0.626088
C5      0.564303
C6      0.719420
C7     55.000000
dtype: float64
```

In [589]:

```
dframe.std() #Standard Deviation
```

Out[589]:

```
C1      0.000000
C2      0.341450
C3    209.639012
C4      0.360398
C5     20.583237
C6    287.793467
C7    470.871785
dtype: float64
```

In [590]:

```
dframe.var()  #Variance
```

Out[590]:

```
C1         0.000000
C2         0.116588
C3     43948.515274
C4         0.129887
C5       423.669665
C6     82825.079704
C7    221720.238095
dtype: float64
```

In [591]:

```
#Lower Quartile / First Quartile
dframe.quantile(0.25)
```

Out[591]:

```
C1    888.000000
C2      0.393160
C3      0.177930
C4      0.264864
C5      0.524066
C6      0.383916
C7     55.000000
Name: 0.25, dtype: float64
```

In [592]:

```
#Second Quartile / Median
dframe.quantile(0.50)
```

Out[592]:

```
C1    888.000000
C2      0.542427
C3      0.427621
C4      0.626088
C5      0.564303
C6      0.719420
C7     55.000000
Name: 0.5, dtype: float64
```

In [593]:

```
# Upper Quartile
dframe.quantile(0.75)
```

Out[593]:

```
C1    888.000000
C2      0.881779
C3      0.616326
C4      0.842046
C5      0.788265
C6     55.000000
C7    537.500000
Name: 0.75, dtype: float64
```

In [594]:

```
 #IQR (Interquartile Range)
dframe.quantile(0.75) - dframe.quantile(0.25)
```

Out[594]:

```
C1      0.000000
C2      0.488618
C3      0.438395
C4      0.577181
C5      0.264199
C6     54.616084
C7    482.500000
dtype: float64
```

In [595]:

```
# SUM of column values
dframe.sum()
```

Out[595]:

```
C1    6216.000000
C2       4.110435
C3     557.086012
C4       3.840897
C5      58.272691
C6     888.710156
C7    2315.000000
dtype: float64
```

In [596]:

```
# GENERATES DESCRIPTIVE STATS
dframe.describe()
```

Out[596]:

	C1	C2	C3	C4	C5	C6	C7
count	7.0	7.000000	7.000000	7.000000	7.000000	7.000000	7.000000
mean	888.0	0.587205	79.583716	0.548700	8.324670	126.958594	330.714286
std	0.0	0.341450	209.639012	0.360398	20.583237	287.793467	470.871785
min	888.0	0.053500	0.069878	0.033939	0.083726	0.222905	55.000000
25%	888.0	0.393160	0.177930	0.264864	0.524066	0.383916	55.000000
50%	888.0	0.542427	0.427621	0.626088	0.564303	0.719420	55.000000
75%	888.0	0.881779	0.616326	0.842046	0.788265	55.000000	537.500000
max	888.0	0.964631	555.000000	0.967050	55.000000	777.000000	1020.000000

In [597]:

```
#Return unbiased skew
# https://www.youtube.com/watch?v=HnMGKsupF8Q
dframe.skew()
```

Out[597]:

```
C1    0.000000
C2   -0.434127
C3    2.645744
C4   -0.289345
C5    2.645020
C6    2.602175
C7    1.229634
dtype: float64
```

In [598]:

```
# Return unbiased kurtosis using Fisher’s definition of kurtosis
# https://www.youtube.com/watch?v=HnMGKsupF8Q
dframe.kurt()
```

Out[598]:

```
C1    0.000000
C2   -1.128285
C3    6.999971
C4   -1.839310
C5    6.997064
C6    6.819785
C7   -0.840000
dtype: float64
```

In [599]:

```
#Correlation
# https://www.youtube.com/watch?v=qtaqvPAeEJY&list=PLblh5JKOoLUK0FLuzwntyYI10UQFUhsY9&index
# https://www.youtube.com/watch?v=xZ_z8KWkhXE&list=PLblh5JKOoLUK0FLuzwntyYI10UQFUhsY9&index
dframe.corr()
```

Out[599]:

	C1	C2	C3	C4	C5	C6	C7
C1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
C2	NaN	1.000000	0.391336	0.435022	-0.112135	-0.004680	-0.388220
C3	NaN	0.391336	1.000000	0.360583	-0.168275	-0.110496	-0.259009
C4	NaN	0.435022	0.360583	1.000000	-0.331306	0.573266	-0.188844
C5	NaN	-0.112135	-0.168275	-0.331306	1.000000	-0.193131	0.649566
C6	NaN	-0.004680	-0.110496	0.573266	-0.193131	1.000000	-0.300565
C7	NaN	-0.388220	-0.259009	-0.188844	0.649566	-0.300565	1.000000

In [600]:

```
#Covariance
# https://www.youtube.com/watch?v=qtaqvPAeEJY&list=PLblh5JKOoLUK0FLuzwntyYI10UQFUhsY9&index
# https://www.youtube.com/watch?v=xZ_z8KWkhXE&list=PLblh5JKOoLUK0FLuzwntyYI10UQFUhsY9&index
dframe.cov()
```

Out[600]:

	C1	C2	C3	C4	C5	C6	C7
C1	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
C2	0.0	0.116588	28.012348	0.053533	-0.788101	-0.459855	-62.417763
C3	0.0	28.012348	43948.515274	27.243357	-726.113997	-6666.522658	-25567.559926
C4	0.0	0.053533	27.243357	0.129887	-2.457682	59.459365	-32.047122
C5	0.0	-0.788101	-726.113997	-2.457682	423.669665	-1144.055556	6295.640483
C6	0.0	-0.459855	-6666.522658	59.459365	-1144.055556	82825.079704	-40730.675740
C7	0.0	-62.417763	-25567.559926	-32.047122	6295.640483	-40730.675740	221720.238095

In [601]:

```
import statistics as st
dframe.at[3:6,'C1'] = 22
dframe
```

Out[601]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	55.000000	55.0
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	55.000000	55.0
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	55.0
2020-01-23	22	0.964631	0.427621	0.257674	0.550580	0.321274	55.0
2020-01-24	22	0.053500	0.775805	0.033939	0.083726	0.719420	55.0
2020-01-25	22	0.496193	0.069878	0.272055	55.000000	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [602]:

```
# Average
st.mean(dframe['C1'])
```

Out[602]:

```
516.8571428571429
```

In [603]:

```
# Hormonic Mean
st.harmonic_mean(dframe['C1'])
```

Out[603]:

```
49.69186046511628
```

In [604]:

```
#Returns average of the two middle numbers when length is EVEN
arr = np.array([1,2,3,4,5,6,7,8])
st.median(arr)
```

Out[604]:

```
4.5
```

In [605]:

```
# low median of the data with EVEN length
st.median_low(arr)
```

Out[605]:

```
4
```

In [606]:

```
# High median of the data with EVEN length
st.median_high(arr)
```

Out[606]:

```
5
```

In [607]:

```
# Mode of Dataset
st.mode(dframe['C7'])
```

Out[607]:

```
55.0
```

In [608]:

```
# Sample Variance
st.variance(dframe['C1'])
```

Out[608]:

```
214273.14285714287
```

In [609]:

```
#Population Variance
st.pvariance(dframe['C1'])
```

Out[609]:

```
183662.69387755104
```

In [610]:

```
#Sample  Standard Deviation
st.stdev(dframe['C1'])
```

Out[610]:

```
462.89647099231905
```

In [611]:

```
#Population Standard Deviation
st.pstdev(dframe['C1'])
```

Out[611]:

```
428.5588569584708
```

Apply function on Dataframe

In [612]:

```
dframe
```

Out[612]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.890359	555.000000	0.843771	0.497552	55.000000	55.0
2020-01-21	888	0.873198	0.456847	0.840320	0.783311	55.000000	55.0
2020-01-22	888	0.542427	0.227592	0.967050	0.564303	777.000000	55.0
2020-01-23	22	0.964631	0.427621	0.257674	0.550580	0.321274	55.0
2020-01-24	22	0.053500	0.775805	0.033939	0.083726	0.719420	55.0
2020-01-25	22	0.496193	0.069878	0.272055	55.000000	0.446558	1020.0
2020-01-26	888	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [613]:

```
# Finding MAX value in Columns
dframe.apply(max)
```

Out[613]:

```
C1     888.000000
C2       0.964631
C3     555.000000
C4       0.967050
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [614]:

```
# Finding minimum value in Columns
dframe.apply(min)
```

Out[614]:

```
C1    22.000000
C2     0.053500
C3     0.069878
C4     0.033939
C5     0.083726
C6     0.222905
C7    55.000000
dtype: float64
```

In [615]:

```
#Sum of Column Values
dframe.apply(sum)
```

Out[615]:

```
C1    3618.000000
C2       4.110435
C3     557.086012
C4       3.840897
C5      58.272691
C6     888.710156
C7    2315.000000
dtype: float64
```

In [616]:

```
#Sum of Column Values
dframe.apply(np.sum)
```

Out[616]:

```
C1    3618.000000
C2       4.110435
C3     557.086012
C4       3.840897
C5      58.272691
C6     888.710156
C7    2315.000000
dtype: float64
```

In [617]:

```
# Sum of rows
dframe.apply(np.sum ,axis=1)
```

Out[617]:

```
2020-01-20    1555.231683
2020-01-21    1000.953676
2020-01-22    1722.301371
2020-01-23      79.521780
2020-01-24      78.666389
2020-01-25    1098.284684
2020-01-26    1910.060607
Freq: D, dtype: float64
```

In [618]:

```
# Square root of all values in a DataFrame
dframe.applymap(np.sqrt)
```

Out[618]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	29.799329	0.943589	23.558438	0.918570	0.705374	7.416198	7.416198
2020-01-21	29.799329	0.934450	0.675904	0.916690	0.885049	7.416198	7.416198
2020-01-22	29.799329	0.736496	0.477066	0.983387	0.751201	27.874720	7.416198
2020-01-23	4.690416	0.982156	0.653927	0.507616	0.742011	0.566810	7.416198
2020-01-24	4.690416	0.231300	0.880798	0.184225	0.289354	0.848186	7.416198
2020-01-25	4.690416	0.704410	0.264345	0.521589	7.416198	0.668250	31.937439
2020-01-26	29.799329	0.538635	0.358147	0.791257	0.890628	0.472128	31.937439

In [619]:

```
# Square root of all values in a DataFrame
dframe.applymap(math.sqrt)
```

Out[619]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	29.799329	0.943589	23.558438	0.918570	0.705374	7.416198	7.416198
2020-01-21	29.799329	0.934450	0.675904	0.916690	0.885049	7.416198	7.416198
2020-01-22	29.799329	0.736496	0.477066	0.983387	0.751201	27.874720	7.416198
2020-01-23	4.690416	0.982156	0.653927	0.507616	0.742011	0.566810	7.416198
2020-01-24	4.690416	0.231300	0.880798	0.184225	0.289354	0.848186	7.416198
2020-01-25	4.690416	0.704410	0.264345	0.521589	7.416198	0.668250	31.937439
2020-01-26	29.799329	0.538635	0.358147	0.791257	0.890628	0.472128	31.937439

In [620]:

```
dframe.applymap(float)
```

Out[620]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888.0	0.890359	555.000000	0.843771	0.497552	55.000000	55.0
2020-01-21	888.0	0.873198	0.456847	0.840320	0.783311	55.000000	55.0
2020-01-22	888.0	0.542427	0.227592	0.967050	0.564303	777.000000	55.0
2020-01-23	22.0	0.964631	0.427621	0.257674	0.550580	0.321274	55.0
2020-01-24	22.0	0.053500	0.775805	0.033939	0.083726	0.719420	55.0
2020-01-25	22.0	0.496193	0.069878	0.272055	55.000000	0.446558	1020.0
2020-01-26	888.0	0.290127	0.128269	0.626088	0.793219	0.222905	1020.0

In [621]:

```
# Using Lambda function in Dataframes
dframe.apply(lambda x: min(x))
```

Out[621]:

```
C1    22.000000
C2     0.053500
C3     0.069878
C4     0.033939
C5     0.083726
C6     0.222905
C7    55.000000
dtype: float64
```

In [622]:

```python
# Using Lambda function in Dataframes
dframe.apply(lambda x: x*x)
```

Out[622]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	788544	0.792740	308025.000000	0.711950	0.247558	3025.000000	3025.0
2020-01-21	788544	0.762474	0.208709	0.706138	0.613576	3025.000000	3025.0
2020-01-22	788544	0.294227	0.051798	0.935185	0.318438	603729.000000	3025.0
2020-01-23	484	0.930513	0.182860	0.066396	0.303138	0.103217	3025.0
2020-01-24	484	0.002862	0.601873	0.001152	0.007010	0.517565	3025.0
2020-01-25	484	0.246207	0.004883	0.074014	3025.000000	0.199414	1040400.0
2020-01-26	788544	0.084174	0.016453	0.391986	0.629196	0.049686	1040400.0

Merge Dataframes

In [623]:

```python
daf1 =  pd.DataFrame ({'id': ['1', '2', '3', '4', '5'], 'Name': ['Asif', 'Basit', 'Bran', '
daf1
```

Out[623]:

	id	Name
0	1	Asif
1	2	Basit
2	3	Bran
3	4	John
4	5	David

In [624]:

```
daf2 =  pd.DataFrame ({'id': ['1', '2', '6', '7', '8'], 'Score': [40 , 60 , 80 , 90 , 70]})
daf2
```

Out[624]:

	id	Score
0	1	40
1	2	60
2	6	80
3	7	90
4	8	70

In [625]:

```
# Inner Join
pd.merge(daf1, daf2, on='id', how='inner')
```

Out[625]:

	id	Name	Score
0	1	Asif	40
1	2	Basit	60

In [626]:

```
# Full Outer Join
pd.merge(daf1, daf2, on='id', how='outer')
```

Out[626]:

	id	Name	Score
0	1	Asif	40.0
1	2	Basit	60.0
2	3	Bran	NaN
3	4	John	NaN
4	5	David	NaN
5	6	NaN	80.0
6	7	NaN	90.0
7	8	NaN	70.0

In [627]:

```
# Left Outer Join
pd.merge(daf1, daf2, on='id', how='left')
```

Out[627]:

	id	Name	Score
0	1	Asif	40.0
1	2	Basit	60.0
2	3	Bran	NaN
3	4	John	NaN
4	5	David	NaN

In [628]:

```
#Right Outer Join
pd.merge(daf1, daf2, on='id', how='right')
```

Out[628]:

	id	Name	Score
0	1	Asif	40
1	2	Basit	60
2	6	NaN	80
3	7	NaN	90
4	8	NaN	70

Importing multiple CSV files in DataFrame

In [5]:

```
# Append all CSV files
path =r'C:\Users\DELL\Documents\GitHub\Public\COVID-19\COVID-19\csse_covid_19_data\csse_cov
filenames = glob.glob(path + "/*.csv")
covid = pd.DataFrame()
for f in filenames:
    df = pd.read_csv(f)
    covid = covid.append(df,ignore_index=True,sort=True)
```

In [6]:

```
# Top 10 rows of the Dataframe
covid.head(10)
```

Out[6]:

	Confirmed	Country/Region	Deaths	Last Update	Latitude	Longitude	Province/State	Recovere
0	1.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Anhui	Nal
1	14.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Beijing	Nal
2	6.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Chongqing	Nal
3	1.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Fujian	Nal
4	NaN	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Gansu	Nal
5	26.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Guangdong	Nal
6	2.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Guangxi	Nal
7	1.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Guizhou	Nal
8	4.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Hainan	Nal
9	1.0	Mainland China	NaN	1/22/2020 17:00	NaN	NaN	Hebei	Nal

In [635]:

```
# Bottom 10 rows of the Dataframe
covid.tail(10)
```

Out[635]:

	Confirmed	Country/Region	Deaths	Last Update	Latitude	Longitude	Province/State	Rec
6428	1.0	The Gambia	0.0	2020-03-17T23:33:02	13.4667	-16.6000	NaN	
6429	1.0	Togo	0.0	2020-03-13T22:22:02	8.6195	0.8248	NaN	
6430	1.0	US	0.0	2020-03-17T23:33:02	38.4912	-80.9545	West Virginia	
6431	1.0	United Kingdom	1.0	2020-03-16T14:53:04	19.3133	-81.2546	Cayman Islands	
6432	0.0	Australia	0.0	2020-03-14T02:33:04	35.4437	139.6380	From Diamond Princess	
6433	0.0	Guernsey	0.0	2020-03-17T18:33:03	49.4500	-2.5800	NaN	
6434	0.0	Jersey	0.0	2020-03-17T18:33:03	49.1900	-2.1100	NaN	
6435	0.0	Puerto Rico	0.0	2020-03-17T16:13:14	18.2000	-66.5000	NaN	
6436	0.0	Republic of the Congo	0.0	2020-03-17T21:33:03	-1.4400	15.5560	NaN	
6437	0.0	occupied Palestinian territory	0.0	2020-03-11T20:53:02	31.9522	35.2332	NaN	

In [12]:

```
# Unique values in Country column
covid['Country/Region'].unique()
```

Out[12]:

```
array(['Mainland China', 'Hong Kong', 'Macau', 'Taiwan', 'US', 'Japan',
       'Thailand', 'South Korea', 'Singapore', 'Philippines', 'Malaysia',
       'Vietnam', 'Australia', 'Mexico', 'Brazil', 'Colombia', 'France',
       'Nepal', 'Canada', 'Cambodia', 'Sri Lanka', 'Ivory Coast',
       'Germany', 'Finland', 'United Arab Emirates', 'India', 'Italy',
       'UK', 'Russia', 'Sweden', 'Spain', 'Belgium', 'Others', 'Egypt',
       'Iran', 'Israel', 'Lebanon', 'Iraq', 'Oman', 'Afghanistan',
       'Bahrain', 'Kuwait', 'Austria', 'Algeria', 'Croatia',
       'Switzerland', 'Pakistan', 'Georgia', 'Greece', 'North Macedonia',
       'Norway', 'Romania', 'Denmark', 'Estonia', 'Netherlands',
       'San Marino', ' Azerbaijan', 'Belarus', 'Iceland', 'Lithuania',
       'New Zealand', 'Nigeria', 'North Ireland', 'Ireland', 'Luxembourg',
       'Monaco', 'Qatar', 'Ecuador', 'Azerbaijan', 'Czech Republic',
       'Armenia', 'Dominican Republic', 'Indonesia', 'Portugal',
       'Andorra', 'Latvia', 'Morocco', 'Saudi Arabia', 'Senegal',
       'Argentina', 'Chile', 'Jordan', 'Ukraine', 'Saint Barthelemy',
       'Hungary', 'Faroe Islands', 'Gibraltar', 'Liechtenstein', 'Poland',
       'Tunisia', 'Palestine', 'Bosnia and Herzegovina', 'Slovenia',
       'South Africa', 'Bhutan', 'Cameroon', 'Costa Rica', 'Peru',
       'Serbia', 'Slovakia', 'Togo', 'Vatican City', 'French Guiana',
       'Malta', 'Martinique', 'Republic of Ireland', 'Bulgaria',
       'Maldives', 'Bangladesh', 'Moldova', 'Paraguay', 'Albania',
       'Cyprus', 'St. Martin', 'Brunei', 'Iran (Islamic Republic of)',
       'Republic of Korea', 'Hong Kong SAR', 'Taipei and environs',
       'Viet Nam', 'occupied Palestinian territory', 'Macao SAR',
       'Russian Federation', 'Republic of Moldova', 'Saint Martin',
       'Burkina Faso', 'Channel Islands', 'Holy See', 'Mongolia',
       'Panama', 'China', 'Korea, South', 'Cruise Ship', 'United Kingdom',
       'Czechia', 'Taiwan*', 'Bolivia', 'Honduras', 'Congo (Kinshasa)',
       "Cote d'Ivoire", 'Jamaica', 'Reunion', 'Turkey', 'Cuba', 'Guyana',
       'Kazakhstan', 'Cayman Islands', 'Guadeloupe', 'Ethiopia', 'Sudan',
       'Guinea', 'Antigua and Barbuda', 'Aruba', 'Kenya', 'Uruguay',
       'Ghana', 'Jersey', 'Namibia', 'Seychelles', 'Trinidad and Tobago',
       'Venezuela', 'Curacao', 'Eswatini', 'Gabon', 'Guatemala',
       'Guernsey', 'Mauritania', 'Rwanda', 'Saint Lucia',
       'Saint Vincent and the Grenadines', 'Suriname', 'Kosovo',
       'Central African Republic', 'Congo (Brazzaville)',
       'Equatorial Guinea', 'Uzbekistan', 'Guam', 'Puerto Rico', 'Benin',
       'Greenland', 'Liberia', 'Mayotte', 'Republic of the Congo',
       'Somalia', 'Tanzania', 'The Bahamas', 'Barbados', 'Montenegro',
       'The Gambia', 'Kyrgyzstan', 'Mauritius', 'Zambia', 'Djibouti',
       'Gambia, The', 'Bahamas, The', 'Chad', 'El Salvador', 'Fiji',
       'Nicaragua', 'Madagascar', 'Haiti', 'Angola', 'Cabo Verde',
       'Niger', 'Papua New Guinea', 'Zimbabwe'], dtype=object)
```

In [7]:

```
# Number of Unique values in Country column
covid['Country/Region'].nunique()
```

Out[7]:

```
206
```

In [9]:

```
#Dataframe information
covid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7313 entries, 0 to 7312
Data columns (total 8 columns):
Confirmed         7294 non-null float64
Country/Region    7313 non-null object
Deaths            6872 non-null float64
Last Update       7313 non-null object
Latitude          4495 non-null float64
Longitude         4495 non-null float64
Province/State    4223 non-null object
Recovered         6925 non-null float64
dtypes: float64(5), object(3)
memory usage: 457.1+ KB
```

In [636]:

```
# Reading columns
covid['Country/Region'].head(10)
```

Out[636]:

```
0    Mainland China
1    Mainland China
2    Mainland China
3    Mainland China
4    Mainland China
5    Mainland China
6    Mainland China
7    Mainland China
8    Mainland China
9    Mainland China
Name: Country/Region, dtype: object
```

In [637]:

```
# Reading columns
df1 = covid[['Country/Region' ,'Province/State','Confirmed' , 'Last Update']]
df1.head(10)
```

Out[637]:

	Country/Region	Province/State	Confirmed	Last Update
0	Mainland China	Anhui	1.0	1/22/2020 17:00
1	Mainland China	Beijing	14.0	1/22/2020 17:00
2	Mainland China	Chongqing	6.0	1/22/2020 17:00
3	Mainland China	Fujian	1.0	1/22/2020 17:00
4	Mainland China	Gansu	NaN	1/22/2020 17:00
5	Mainland China	Guangdong	26.0	1/22/2020 17:00
6	Mainland China	Guangxi	2.0	1/22/2020 17:00
7	Mainland China	Guizhou	1.0	1/22/2020 17:00
8	Mainland China	Hainan	4.0	1/22/2020 17:00
9	Mainland China	Hebei	1.0	1/22/2020 17:00

In [638]:

```
#Read specific rows
df1.iloc[1:4]
```

Out[638]:

	Country/Region	Province/State	Confirmed	Last Update
1	Mainland China	Beijing	14.0	1/22/2020 17:00
2	Mainland China	Chongqing	6.0	1/22/2020 17:00
3	Mainland China	Fujian	1.0	1/22/2020 17:00

In [639]:

```
#Filter data
df1.loc[df1['Country/Region']== 'India']
```

Out[639]:

	Country/Region	Province/State	Confirmed	Last Update
430	India	NaN	1.0	1/30/20 16:00
491	India	NaN	1.0	1/31/2020 23:59
547	India	NaN	1.0	1/31/2020 8:15
607	India	NaN	2.0	2020-02-02T06:03:08
672	India	NaN	3.0	2020-02-03T21:43:02
740	India	NaN	3.0	2020-02-03T21:43:02
810	India	NaN	3.0	2020-02-03T21:43:02
881	India	NaN	3.0	2020-02-03T21:43:02
954	India	NaN	3.0	2020-02-03T21:43:02
1026	India	NaN	3.0	2020-02-03T21:43:02
1098	India	NaN	3.0	2020-02-03T21:43:02
1171	India	NaN	3.0	2020-02-03T21:43:02
1243	India	NaN	3.0	2020-02-03T21:43:02
1316	India	NaN	3.0	2020-02-03T21:43:02
1389	India	NaN	3.0	2020-02-13T18:53:02
1463	India	NaN	3.0	2020-02-13T18:53:02
1538	India	NaN	3.0	2020-02-13T18:53:02
1613	India	NaN	3.0	2020-02-16T07:43:02
1688	India	NaN	3.0	2020-02-16T07:43:02
1763	India	NaN	3.0	2020-02-16T07:43:02
1838	India	NaN	3.0	2020-02-16T07:43:02
1915	India	NaN	3.0	2020-02-16T07:43:02
1995	India	NaN	3.0	2020-02-16T07:43:02
2079	India	NaN	3.0	2020-02-16T07:43:02
2163	India	NaN	3.0	2020-02-16T07:43:02
2248	India	NaN	3.0	2020-02-16T07:43:02
2341	India	NaN	3.0	2020-02-16T07:43:02
2438	India	NaN	3.0	2020-02-16T07:43:02
2543	India	NaN	3.0	2020-02-16T07:43:02
2650	India	NaN	3.0	2020-02-16T07:43:02
2771	India	NaN	3.0	2020-02-16T07:43:02
2895	India	NaN	3.0	2020-02-16T07:43:02
3016	India	NaN	5.0	2020-03-02T22:33:09
3162	India	NaN	5.0	2020-03-02T22:33:09

	Country/Region	Province/State	Confirmed	Last Update
3288	India	NaN	28.0	2020-03-04T12:33:03
3451	India	NaN	30.0	2020-03-05T13:53:03
3624	India	NaN	31.0	2020-03-06T13:03:12
3825	India	NaN	34.0	2020-03-07T18:13:27
4051	India	NaN	39.0	2020-03-08T09:23:05
4308	India	NaN	43.0	2020-03-09T09:13:17
4573	India	NaN	56.0	2020-03-10T10:13:28
4781	India	NaN	62.0	2020-03-11T22:13:12
4996	India	NaN	73.0	2020-03-12T08:33:13
5277	India	NaN	82.0	2020-03-11T20:00:00
5451	India	NaN	102.0	2020-03-14T20:33:03
5699	India	NaN	113.0	2020-03-15T18:20:18
5967	India	NaN	119.0	2020-03-16T14:38:45
6236	India	NaN	142.0	2020-03-17T15:33:06

In [641]:

```
#Sort Data Frame
display('Sorted Data Frame', df1.sort_values(['Country/Region'], ascending=True).head(5))
```

```
'Sorted Data Frame'
```

	Country/Region	Province/State	Confirmed	Last Update
2663	Azerbaijan	NaN	1.0	2020-02-28T15:03:26
2556	Afghanistan	NaN	1.0	2020-02-24T23:33:02
3919	Afghanistan	NaN	1.0	2020-02-24T23:33:02
3189	Afghanistan	NaN	1.0	2020-02-24T23:33:02
3705	Afghanistan	NaN	1.0	2020-02-24T23:33:02

In [642]:

```
#Sort Data Frame
display('Sorted Data Frame', df1.sort_values(['Country/Region'], ascending=False).head(5))
```

```
'Sorted Data Frame'
```

	Country/Region	Province/State	Confirmed	Last Update
6437	occupied Palestinian territory	NaN	0.0	2020-03-11T20:53:02
5152	occupied Palestinian territory	NaN	0.0	2020-03-11T20:53:02
5889	occupied Palestinian territory	NaN	0.0	2020-03-11T20:53:02
5631	occupied Palestinian territory	NaN	0.0	2020-03-11T20:53:02
4592	occupied Palestinian territory	NaN	25.0	2020-03-10T19:13:21

In [643]:

```
#Sort Data Frame - Ascending on "Country" & descending on "Last update"
display('Sorted Data Frame', df1.sort_values(['Country/Region', 'Last Update'], ascending=[
```

```
'Sorted Data Frame'
```

	Country/Region	Province/State	Confirmed	Last Update
2663	Azerbaijan	NaN	1.0	2020-02-28T15:03:26
6324	Afghanistan	NaN	22.0	2020-03-17T11:53:10
6037	Afghanistan	NaN	21.0	2020-03-16T14:38:45
5779	Afghanistan	NaN	16.0	2020-03-15T18:20:18
5534	Afghanistan	NaN	11.0	2020-03-14T14:53:04

In [644]:

```
#Iterating through the dataset
for index , row in df1.iterrows():
    if (row['Country/Region'] == 'Indonesia' ):
        display(row[['Country/Region' ,'Confirmed']])
```

```
Country/Region    Indonesia
Confirmed                 2
Name: 3034, dtype: object

Country/Region    Indonesia
Confirmed                 2
Name: 3176, dtype: object

Country/Region    Indonesia
Confirmed                 2
Name: 3337, dtype: object

Country/Region    Indonesia
Confirmed                 2
Name: 3504, dtype: object

Country/Region    Indonesia
Confirmed                 4
Name: 3672, dtype: object

Country/Region    Indonesia
Confirmed                 4
Name: 3878, dtype: object

Country/Region    Indonesia
Confirmed                 6
Name: 4102, dtype: object

Country/Region    Indonesia
Confirmed                19
Name: 4326, dtype: object

Country/Region    Indonesia
Confirmed                27
Name: 4590, dtype: object

Country/Region    Indonesia
Confirmed                34
Name: 4798, dtype: object

Country/Region    Indonesia
Confirmed                34
Name: 5018, dtype: object

Country/Region    Indonesia
Confirmed                69
Name: 5265, dtype: object

Country/Region    Indonesia
Confirmed                96
Name: 5453, dtype: object

Country/Region    Indonesia
Confirmed               117
Name: 5696, dtype: object

Country/Region    Indonesia
```

```
Confirmed                 134
Name: 5960, dtype: object

Country/Region    Indonesia
Confirmed               172
Name: 6227, dtype: object
```

In [645]:

```
#Unique Values
covid['Country/Region'].drop_duplicates(keep='first').head(10)
```

Out[645]:

```
0      Mainland China
12          Hong Kong
20              Macau
28             Taiwan
31                 US
35              Japan
36           Thailand
37        South Korea
76          Singapore
77        Philippines
Name: Country/Region, dtype: object
```

In [646]:

```
# Countries impacted with Coronavirus
countries = covid['Country/Region'].unique()
type(countries) , countries
```

Out[646]:

```
(numpy.ndarray,
 array(['Mainland China', 'Hong Kong', 'Macau', 'Taiwan', 'US', 'Japan',
        'Thailand', 'South Korea', 'Singapore', 'Philippines', 'Malaysia',
        'Vietnam', 'Australia', 'Mexico', 'Brazil', 'Colombia', 'France',
        'Nepal', 'Canada', 'Cambodia', 'Sri Lanka', 'Ivory Coast',
        'Germany', 'Finland', 'United Arab Emirates', 'India', 'Italy',
        'UK', 'Russia', 'Sweden', 'Spain', 'Belgium', 'Others', 'Egypt',
        'Iran', 'Israel', 'Lebanon', 'Iraq', 'Oman', 'Afghanistan',
        'Bahrain', 'Kuwait', 'Austria', 'Algeria', 'Croatia',
        'Switzerland', 'Pakistan', 'Georgia', 'Greece', 'North Macedonia',
        'Norway', 'Romania', 'Denmark', 'Estonia', 'Netherlands',
        'San Marino', ' Azerbaijan', 'Belarus', 'Iceland', 'Lithuania',
        'New Zealand', 'Nigeria', 'North Ireland', 'Ireland', 'Luxembourg',
        'Monaco', 'Qatar', 'Ecuador', 'Azerbaijan', 'Czech Republic',
        'Armenia', 'Dominican Republic', 'Indonesia', 'Portugal',
        'Andorra', 'Latvia', 'Morocco', 'Saudi Arabia', 'Senegal',
        'Argentina', 'Chile', 'Jordan', 'Ukraine', 'Saint Barthelemy',
        'Hungary', 'Faroe Islands', 'Gibraltar', 'Liechtenstein', 'Poland',
        'Tunisia', 'Palestine', 'Bosnia and Herzegovina', 'Slovenia',
        'South Africa', 'Bhutan', 'Cameroon', 'Costa Rica', 'Peru',
        'Serbia', 'Slovakia', 'Togo', 'Vatican City', 'French Guiana',
        'Malta', 'Martinique', 'Republic of Ireland', 'Bulgaria',
        'Maldives', 'Bangladesh', 'Moldova', 'Paraguay', 'Albania',
        'Cyprus', 'St. Martin', 'Brunei', 'Iran (Islamic Republic of)',
        'Republic of Korea', 'Hong Kong SAR', 'Taipei and environs',
        'Viet Nam', 'occupied Palestinian territory', 'Macao SAR',
        'Russian Federation', 'Republic of Moldova', 'Saint Martin',
        'Burkina Faso', 'Channel Islands', 'Holy See', 'Mongolia',
        'Panama', 'China', 'Korea, South', 'Cruise Ship', 'United Kingdom',
        'Czechia', 'Taiwan*', 'Bolivia', 'Honduras', 'Congo (Kinshasa)',
        "Cote d'Ivoire", 'Jamaica', 'Reunion', 'Turkey', 'Cuba', 'Guyana',
        'Kazakhstan', 'Cayman Islands', 'Guadeloupe', 'Ethiopia', 'Sudan',
        'Guinea', 'Antigua and Barbuda', 'Aruba', 'Kenya', 'Uruguay',
        'Ghana', 'Jersey', 'Namibia', 'Seychelles', 'Trinidad and Tobago',
        'Venezuela', 'Curacao', 'Eswatini', 'Gabon', 'Guatemala',
        'Guernsey', 'Mauritania', 'Rwanda', 'Saint Lucia',
        'Saint Vincent and the Grenadines', 'Suriname', 'Kosovo',
        'Central African Republic', 'Congo (Brazzaville)',
        'Equatorial Guinea', 'Uzbekistan', 'Guam', 'Puerto Rico', 'Benin',
        'Greenland', 'Liberia', 'Mayotte', 'Republic of the Congo',
        'Somalia', 'Tanzania', 'The Bahamas', 'Barbados', 'Montenegro',
        'The Gambia'], dtype=object))
```

In [648]:

```
df2 = pd.read_csv('Pokemon.csv')
df2.head(5)
```

Out[648]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	

In [649]:

```
# Sum of Columns
df2['Total'] = df2['HP'] + df2['Attack']
df2.head(5)
```

Out[649]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	F
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	F
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	F
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	F
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	F

In [651]:

```
# Sum of Columns
df2['Total'] = df2.iloc[:,4:10].sum(axis=1)
df2.head(5)
```

Out[651]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	F
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	F
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	F
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	F
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	F

In [652]:

```
#Shifting "Total" column

cols = list(df2.columns)

df2 = df2[cols[0:10] + [cols[-1]] + cols[10:12]]
df2.head(5)
```

Out[652]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Total	Generation
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	318	1
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	405	1
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	525	1
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	625	1
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	309	1

In [653]:

```
#Shifting "Legendary" column -  Index location -1 or 12

cols = list(df2.columns)

df2 = df2[cols[0:10] + [cols[-1]] + cols[10:12]]
df2.head(5)
```

Out[653]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Legendary	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	False	318
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	False	405
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	False	525
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	False	625
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	False	309

In [654]:

```
#Shifting "Generation" column - Index location -1 or 12

cols = list(df2.columns)

df2 = df2[cols[0:10] + [cols[12]] + cols[10:12]]
df2.head(5)
```

Out[654]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	F
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	F
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	F
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	F
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	F

In [655]:

```
#Save to CSV file

df2.to_csv('poke_updated.csv')
```

In [656]:

```
#Save to CSV file without index column

df2.to_csv('poke_updated1.csv', index=False)
```

In [657]:

```
df2.head(10)
```

Out[657]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legen
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [664]:

```
# Save Dataframe as text file
df2.to_csv('poke.txt' , sep='\t' , index=False)
```

In [658]:

```
# Save Dataframe as xlsx file
df2.to_excel('poke.xlsx')
```

In [659]:

```
# Save Dataframe as xlsx file without row names
df2.to_excel('poke.xlsx', index=0)
```

In [665]:

```
#Filtering using loc

df2.loc[df2['Type 2'] == 'Dragon']
```

Out[665]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1
196	181	AmpharosMega Ampharos	Electric	Dragon	90	95	105	165	110	45	2
249	230	Kingdra	Water	Dragon	75	95	95	95	95	85	2
275	254	SceptileMega Sceptile	Grass	Dragon	70	110	75	145	85	145	3
360	329	Vibrava	Ground	Dragon	50	70	50	50	50	70	3
361	330	Flygon	Ground	Dragon	80	100	80	80	80	100	3
540	483	Dialga	Steel	Dragon	100	120	120	150	100	90	4
541	484	Palkia	Water	Dragon	90	120	100	150	120	100	4
544	487	GiratinaAltered Forme	Ghost	Dragon	150	100	120	100	120	90	4
545	487	GiratinaOrigin Forme	Ghost	Dragon	150	120	100	120	100	90	4
694	633	Deino	Dark	Dragon	52	65	50	45	50	38	5
695	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5
696	635	Hydreigon	Dark	Dragon	92	105	90	125	90	98	5
761	691	Dragalge	Poison	Dragon	65	75	90	97	123	44	6
766	696	Tyrunt	Rock	Dragon	58	89	77	45	45	48	6
767	697	Tyrantrum	Rock	Dragon	82	121	119	69	59	71	6
790	714	Noibat	Flying	Dragon	40	30	35	45	40	55	6
791	715	Noivern	Flying	Dragon	85	70	80	97	80	123	6

In [666]:

```
#Filtering using loc
df3 = df2.loc[(df2['Type 2'] == 'Dragon') & (df2['Type 1'] == 'Dark')]
df3
```

Out[666]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
694	633	Deino	Dark	Dragon	52	65	50	45	50	38	5	F
695	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5	F
696	635	Hydreigon	Dark	Dragon	92	105	90	125	90	98	5	F

In [667]:

```
#Reset index for Dataframe df3 keeping old index column

df4 = df3.reset_index()
df4
```

Out[667]:

	index	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Le
0	694	633	Deino	Dark	Dragon	52	65	50	45	50	38	5	
1	695	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5	
2	696	635	Hydreigon	Dark	Dragon	92	105	90	125	90	98	5	

In [668]:

```
#Reset index for Dataframe df3 removing old index column

df3.reset_index(drop=True , inplace=True)
df3
```

Out[668]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendar
0	633	Deino	Dark	Dragon	52	65	50	45	50	38	5	Fals
1	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5	Fals
2	635	Hydreigon	Dark	Dragon	92	105	90	125	90	98	5	Fals

In [669]:

```
df2.head(10)
```

Out[669]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legen
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	I
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	I
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	I
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	I
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	I
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	I
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	I
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	I
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	I
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	I

LIKE OPERATION IN PANDAS

In [670]:

```
df2.Name.str.contains("rill").head(10)
```

Out[670]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
Name: Name, dtype: bool
```

In [671]:

```
# Display all rows containing Name "rill"
df2.loc[df2.Name.str.contains("rill")]
```

Out[671]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Le
18	15	Beedrill	Bug	Poison	65	90	40	45	80	75	1	
19	15	BeedrillMega Beedrill	Bug	Poison	65	150	40	15	80	145	1	
198	183	Marill	Water	Fairy	70	20	50	20	50	40	2	
199	184	Azumarill	Water	Fairy	100	50	80	60	80	50	2	
322	298	Azurill	Normal	Fairy	50	20	40	20	40	20	3	
589	530	Excadrill	Ground	Steel	110	135	60	50	65	88	5	
653	592	Frillish	Water	Ghost	55	40	50	65	85	40	5	

In [672]:

```
# Exclude all rows containing "rill"
df2.loc[~df2.Name.str.contains("rill")].head(10)
```

Out[672]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legen
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	I
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	I
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	I
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	I
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	I
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	I
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	I
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	I
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	I
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	I

In [673]:

```
#Display all rows with Type-1 as "Grass" and Type-2 as "Poison"

df2.loc[df2['Type 1'].str.contains("Grass") & df2['Type 2'].str.contains("Poison")]
```

Out[673]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
48	43	Oddish	Grass	Poison	45	50	55	75	65	30	1	
49	44	Gloom	Grass	Poison	60	65	70	85	75	40	1	
50	45	Vileplume	Grass	Poison	75	80	85	110	90	50	1	
75	69	Bellsprout	Grass	Poison	50	75	35	70	30	40	1	
76	70	Weepinbell	Grass	Poison	65	90	50	85	45	55	1	
77	71	Victreebel	Grass	Poison	80	105	65	100	70	70	1	
344	315	Roselia	Grass	Poison	50	60	45	100	80	65	3	
451	406	Budew	Grass	Poison	40	30	35	50	70	55	4	
452	407	Roserade	Grass	Poison	60	70	65	125	105	90	4	
651	590	Foongus	Grass	Poison	69	55	45	55	55	15	5	
652	591	Amoonguss	Grass	Poison	114	85	70	85	80	30	5	

In [674]:

```
df2.loc[df2['Type 1'].str.contains('Grass|Water',regex = True)].head(10)
```

Out[674]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	1	
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120	135	115	78	1	
48	43	Oddish	Grass	Poison	45	50	55	75	65	30	1	
49	44	Gloom	Grass	Poison	60	65	70	85	75	40	1	

In [675]:

```
# Due to Case-sensitive it will not return any data

df2.loc[df2['Type 1'].str.contains('grass|water',regex = True)].head(10)
```

Out[675]:

#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total

In [676]:

```
# To ignore case we can use "case = False"

df2.loc[df2['Type 1'].str.contains('grass|water', case = False ,regex = True)].head(10)
```

Out[676]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	1	
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120	135	115	78	1	
48	43	Oddish	Grass	Poison	45	50	55	75	65	30	1	
49	44	Gloom	Grass	Poison	60	65	70	85	75	40	1	

In [677]:

```
# To ignore case we can use "Flags = re.I"

df2.loc[df2['Type 1'].str.contains('grass|water',flags = re.I ,regex = True)].head(10)
```

Out[677]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	1	
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120	135	115	78	1	
48	43	Oddish	Grass	Poison	45	50	55	75	65	30	1	
49	44	Gloom	Grass	Poison	60	65	70	85	75	40	1	

Regex in Pandas dataframe

In [678]:

```
#Get all rows with name starting with "wa"

df2.loc[df2.Name.str.contains('^Wa',flags = re.I ,regex = True)].head(10)
```

Out[678]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	F
350	320	Wailmer	Water	NaN	130	70	35	70	35	60	3	F
351	321	Wailord	Water	NaN	170	90	45	90	45	60	3	F
400	365	Walrein	Ice	Water	110	80	90	95	90	65	3	F
564	505	Watchog	Normal	NaN	60	85	69	60	69	77	5	F

In [679]:

```
#Get all rows with name starting with "wa" followed by any letter between a-l

df2.loc[df2.Name.str.contains('^Wa[a-l]+',flags = re.I ,regex = True)].head(10)
```

Out[679]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendai
350	320	Wailmer	Water	NaN	130	70	35	70	35	60	3	Fals
351	321	Wailord	Water	NaN	170	90	45	90	45	60	3	Fals
400	365	Walrein	Ice	Water	110	80	90	95	90	65	3	Fals

In [680]:

```
#Get all rows with name starting with x , y, z

df2.loc[df2.Name.str.contains('^[x-z]',flags = re.I ,regex = True)]
```

Out[680]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
46	41	Zubat	Poison	Flying	40	45	35	30	40	55	1	
157	145	Zapdos	Electric	Flying	90	90	85	125	90	100	1	
192	178	Xatu	Psychic	Flying	65	75	70	95	70	95	2	
208	193	Yanma	Bug	Flying	65	65	45	75	45	95	2	
286	263	Zigzagoon	Normal	NaN	38	30	41	30	41	60	3	
367	335	Zangoose	Normal	NaN	73	115	60	60	60	90	3	
520	469	Yanmega	Bug	Flying	86	76	86	116	56	95	4	
582	523	Zebstrika	Electric	NaN	75	100	63	80	63	116	5	
623	562	Yamask	Ghost	NaN	38	30	85	55	65	30	5	
631	570	Zorua	Dark	NaN	40	65	40	80	40	65	5	
632	571	Zoroark	Dark	NaN	60	105	60	120	60	105	5	
695	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5	
707	644	Zekrom	Dragon	Electric	100	150	120	120	100	90	5	
792	716	Xerneas	Fairy	NaN	126	131	95	131	98	99	6	
793	717	Yveltal	Dark	Flying	126	131	95	131	98	99	6	
794	718	Zygarde50% Forme	Dragon	Ground	108	100	121	81	95	95	6	

In [681]:

```
# Extracting first 3 characters from "Name" column
df2['Name2'] = df2.Name.str.extract(r'(^\w{3})')
```

In [682]:

```
df2.head(5)
```

Out[682]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	F
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	F
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	F
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	F
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	F

In [683]:

```
# Return all rows with "Name" starting with character 'B or b'
df2.loc[df2.Name.str.match(r'(^[B|b].*)')].head(5)
```

Out[683]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	1	
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120	135	115	78	1	
15	12	Butterfree	Bug	Flying	60	45	50	90	80	70	1	
18	15	Beedrill	Bug	Poison	65	90	40	45	80	75	1	

Replace values in dataframe

In [684]:

```
df2.head(10)
```

Out[684]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legen
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [685]:

```
df2['Type 1'] = df2['Type 1'].replace({"Grass" : "Meadow" , "Fire" :"Blaze"})
```

In [686]:

```
df2.head(10)
```

Out[686]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
0	1	Bulbasaur	Meadow	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	Poison	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Blaze	Dragon	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Blaze	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [687]:

```
df2['Type 2'] = df2['Type 2'].replace({"Poison" : "Venom"})
```

In [688]:

```
df2.head(5)
```

Out[688]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	1	Bulbasaur	Meadow	Venom	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	Venom	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	Venom	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	Venom	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	

In [689]:

```
df2['Type 2'] = df2['Type 2'].replace(['Venom' , 'Dragon'] , 'DANGER')
```

In [690]:

```
df2.head(10)
```

Out[690]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
0	1	Bulbasaur	Meadow	DANGER	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	DANGER	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	DANGER	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	DANGER	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Blaze	DANGER	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Blaze	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [691]:

```
df2.loc[df2['Type 2'] == 'DANGER' , 'Name2'] = np.NaN
```

In [692]:

```
df2.head(10)
```

Out[692]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
0	1	Bulbasaur	Meadow	DANGER	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	DANGER	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	DANGER	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	DANGER	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Blaze	DANGER	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Blaze	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [693]:

```
df2.loc[df2['Total'] > 400 , ['Name2' , 'Legendary']] = 'ALERT'
df2.head(10)
```

Out[693]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
0	1	Bulbasaur	Meadow	DANGER	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	DANGER	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	DANGER	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	DANGER	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Blaze	DANGER	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Blaze	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

In [694]:

```
df2.loc[df2['Total'] > 400 , ['Legendary' , 'Name2']] = ['ALERT-1' , 'ALERT-2']
df2.head(10)
```

Out[694]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
0	1	Bulbasaur	Meadow	DANGER	45	49	49	65	65	45	1	
1	2	Ivysaur	Meadow	DANGER	60	62	63	80	80	60	1	
2	3	Venusaur	Meadow	DANGER	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Meadow	DANGER	80	100	123	122	120	80	1	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Blaze	DANGER	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Blaze	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	

Group By

In [695]:

```
df = pd.read_csv('poke_updated1.csv')
df.head(5)
```

Out[695]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legenc
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	F
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	F
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	F
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	F
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	F

In [696]:

```
df.groupby(['Type 1']).mean().head(10)
```

Out[696]:

Type 1	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gene
Bug	334.492754	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159	3.2
Dark	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290	4.0
Dragon	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250	3.8
Electric	363.500000	59.795455	69.090909	66.295455	90.022727	73.704545	84.500000	3.2
Fairy	449.529412	74.117647	61.529412	65.705882	78.529412	84.705882	48.588235	4.1
Fighting	363.851852	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074	3.3
Fire	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308	3.2
Flying	677.750000	70.750000	78.750000	66.250000	94.250000	72.500000	102.500000	5.5
Ghost	486.500000	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750	4.1
Grass	344.871429	67.271429	73.214286	70.800000	77.500000	70.428571	61.928571	3.3

In [697]:

```
df.groupby(['Type 1']).mean().sort_values('Attack' , ascending = False).head(10)
```

Out[697]:

Type 1	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
Dragon	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250	3
Fighting	363.851852	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074	3
Ground	356.281250	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250	3
Rock	392.727273	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091	3
Steel	442.851852	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259	3
Dark	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290	4
Fire	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308	3
Flying	677.750000	70.750000	78.750000	66.250000	94.250000	72.500000	102.500000	5
Poison	251.785714	67.250000	74.678571	68.821429	60.428571	64.392857	63.571429	2
Water	303.089286	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286	2

In [698]:

```
df.groupby(['Type 1']).mean().sort_values('Defense' , ascending = False).head(10)
```

Out[698]:

	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
Type 1								
Steel	442.851852	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259	3.8
Rock	392.727273	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091	3.4
Dragon	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250	3.8
Ground	356.281250	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250	3.1
Ghost	486.500000	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750	4.1
Water	303.089286	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286	2.8
Ice	423.541667	72.000000	72.750000	71.416667	77.541667	76.291667	63.458333	3.5
Grass	344.871429	67.271429	73.214286	70.800000	77.500000	70.428571	61.928571	3.3
Bug	334.492754	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159	3.2
Dark	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290	4.0

In [699]:

```
df.groupby(['Type 1']).mean().sort_values('Speed' , ascending = False).head(10)
```

Out[699]:

	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gene
Type 1								
Flying	677.750000	70.750000	78.750000	66.250000	94.250000	72.500000	102.500000	5.5
Electric	363.500000	59.795455	69.090909	66.295455	90.022727	73.704545	84.500000	3.2
Dragon	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250	3.8
Psychic	380.807018	70.631579	71.456140	67.684211	98.403509	86.280702	81.491228	3.3
Dark	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290	4.0
Fire	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308	3.2
Normal	319.173469	77.275510	73.469388	59.846939	55.816327	63.724490	71.551020	3.0
Fighting	363.851852	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074	3.3
Water	303.089286	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286	2.8
Ghost	486.500000	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750	4.1

In [700]:

```
df.sum()
```

Out[700]:

```
#                                                        290251
Name          BulbasaurIvysaurVenusaurVenusaurMega VenusaurC...
Type 1        GrassGrassGrassGrassFireFireFireFireFireWaterW...
HP                                                        55407
Attack                                                    63201
Defense                                                   59074
Sp. Atk                                                   58256
Sp. Def                                                   57522
Speed                                                     54622
Generation                                                 2659
Legendary                                                    65
Total                                                    348082
dtype: object
```

In [701]:

```
df.groupby(['Type 2']).sum().head(5)
```

Out[701]:

	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total
Type 2										
Bug	1146	160	270	240	140	185	185	10	0.0	1180
Dark	8277	1511	2196	1441	1636	1397	1507	75	1.0	9688
Dragon	8686	1479	1700	1567	1773	1502	1450	75	4.0	9471
Electric	2794	529	436	410	487	441	429	24	1.0	2732
Fairy	8718	1479	1417	1699	1725	1885	1408	82	2.0	9613

In [702]:

```
df.count()
```

Out[702]:

```
#             800
Name          800
Type 1        800
Type 2        414
HP            800
Attack        800
Defense       800
Sp. Atk       800
Sp. Def       800
Speed         800
Generation    800
Legendary     800
Total         800
dtype: int64
```

In [703]:

```
df['count1'] = 0
df.groupby(['Type 2']).count()['count1']
```

Out[703]:

```
Type 2
Bug          3
Dark        20
Dragon      18
Electric     6
Fairy       23
Fighting    26
Fire        12
Flying      97
Ghost       14
Grass       25
Ground      35
Ice         14
Normal       4
Poison      34
Psychic     33
Rock        14
Steel       22
Water       14
Name: count1, dtype: int64
```

In [704]:

```
df['count1'] = 0
df.groupby(['Type 1']).count()['count1']
```

Out[704]:

```
Type 1
Bug          69
Dark         31
Dragon       32
Electric     44
Fairy        17
Fighting     27
Fire         52
Flying        4
Ghost        32
Grass        70
Ground       32
Ice          24
Normal       98
Poison       28
Psychic      57
Rock         44
Steel        27
Water       112
Name: count1, dtype: int64
```

In [705]:

```
df['count1'] = 0
df.groupby(['Type 1' , 'Type 2' , 'Legendary']).count()['count1']
```

Out[705]:

```
Type 1    Type 2    Legendary
Bug       Electric  False         2
          Fighting  False         2
          Fire      False         2
          Flying    False        14
          Ghost     False         1
          Grass     False         6
          Ground    False         2
          Poison    False        12
          Rock      False         3
          Steel     False         7
          Water     False         1
Dark      Dragon    False         3
          Fighting  False         2
          Fire      False         3
          Flying    False         4
                    True          1
          Ghost     False         2
          Ice       False         2
          Psychic   False         2
          Steel     False         2
Dragon    Electric  True          1
          Fairy     False         1
          Fire      True          1
          Flying    False         4
                    True          2
          Ground    False         4
                    True          1
          Ice       True          3
          Psychic   True          4
Electric  Dragon    False         1
                                 ..
Rock      Grass     False         2
          Ground    False         6
          Ice       False         2
          Psychic   False         2
          Steel     False         3
          Water     False         6
Steel     Dragon    True          1
          Fairy     False         3
          Fighting  True          1
          Flying    False         1
          Ghost     False         4
          Ground    False         2
          Psychic   False         6
                    True          1
          Rock      False         3
Water     Dark      False         6
          Dragon    False         1
                    True          1
          Electric  False         2
          Fairy     False         2
          Fighting  False         3
          Flying    False         7
          Ghost     False         2
          Grass     False         3
          Ground    False        10
          Ice       False         3
          Poison    False         3
          Psychic   False         5
          Rock      False         4
          Steel     False         1
Name: count1, Length: 150, dtype: int64
```

Loading Data in Chunks

In [706]:

```
for df in pd.read_csv('poke_updated1.csv', chunksize=10):
    print(df)
```

```
   #                     Name Type 1  Type 2  HP  Attack  Defense  Sp. A
tk  \
0  1                Bulbasaur  Grass  Poison  45      49       49
65
1  2                  Ivysaur  Grass  Poison  60      62       63
80
2  3                 Venusaur  Grass  Poison  80      82       83      1
00
3  3      VenusaurMega Venusaur  Grass  Poison  80     100      123      1
22
4  4               Charmander   Fire     NaN  39      52       43
60
5  5               Charmeleon   Fire     NaN  58      64       58
80
6  6                Charizard   Fire  Flying  78      84       78      1
09
7  6  CharizardMega Charizard X   Fire  Dragon  78     130      111      1
30
8  6  CharizardMega Charizard Y   Fire  Flying  78     104       78      1
```

In [707]:

```
df
```

Out[707]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
790	714	Noibat	Flying	Dragon	40	30	35	45	40	55	6
791	715	Noivern	Flying	Dragon	85	70	80	97	80	123	6
792	716	Xerneas	Fairy	NaN	126	131	95	131	98	99	6
793	717	Yveltal	Dark	Flying	126	131	95	131	98	99	6
794	718	Zygarde50% Forme	Dragon	Ground	108	100	121	81	95	95	6
795	719	Diancie	Rock	Fairy	50	100	150	100	150	50	6
796	719	DiancieMega Diancie	Rock	Fairy	50	160	110	160	110	110	6
797	720	HoopaHoopa Confined	Psychic	Ghost	80	110	60	150	130	70	6
798	720	HoopaHoopa Unbound	Psychic	Dark	80	160	60	170	130	80	6
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	6

In [708]:

```
df1 = pd.DataFrame()
for df in pd.read_csv('poke_updated1.csv', chunksize=10):
    df1 = pd.concat([df1 ,df])
df1.head(15)
```

Out[708]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Leg
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	1	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	1	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	1	
7	6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	
8	6	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	1	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	1	
12	9	BlastoiseMega Blastoise	Water	NaN	79	103	120	135	115	78	1	
13	10	Caterpie	Bug	NaN	45	30	35	20	20	45	1	
14	11	Metapod	Bug	NaN	50	20	55	25	25	30	1	

Stack & unstack in Pandas

In [709]:

```
col = pd.MultiIndex.from_product([['2010','2015'],['Literacy' , 'GDP']])

data =([[80,7,88,6],[90,8,92,7],[89,7,91,8],[87,6,93,8]])

df6 = pd.DataFrame(data, index=['India','USA' , 'Russia' , 'China'], columns=col)
df6
```

Out[709]:

	2010		**2015**	
	Literacy	**GDP**	**Literacy**	**GDP**
India	80	7	88	6
USA	90	8	92	7
Russia	89	7	91	8
China	87	6	93	8

In [710]:

```
# Stack() Function stacks the columns to rows.
st_df = df6.stack()
st_df
```

Out[710]:

		2010	**2015**
India	**GDP**	7	6
	Literacy	80	88
USA	**GDP**	8	7
	Literacy	90	92
Russia	**GDP**	7	8
	Literacy	89	91
China	**GDP**	6	8
	Literacy	87	93

In [711]:

```
#Unstacks the row to columns
unst_df = st_df.unstack()
unst_df
```

Out[711]:

	2010		2015	
	GDP	Literacy	GDP	Literacy
India	7	80	6	88
USA	8	90	7	92
Russia	7	89	8	91
China	6	87	8	93

In [712]:

```
unst_df = unst_df.unstack()
unst_df
```

Out[712]:

```
2010  GDP       India      7
                USA        8
                Russia     7
                China      6
      Literacy  India     80
                USA       90
                Russia    89
                China     87
2015  GDP       India      6
                USA        7
                Russia     8
                China      8
      Literacy  India     88
                USA       92
                Russia    91
                China     93
dtype: int64
```

In [713]:

```
unst_df = unst_df.unstack()
unst_df
```

Out[713]:

		India	USA	Russia	China
2010	**GDP**	7	8	7	6
	Literacy	80	90	89	87
2015	**GDP**	6	7	8	8
	Literacy	88	92	91	93

PIVOT Tables

In [714]:

```
data = {
    'Country':['India','USA' , 'Russia' , 'China','India','USA' , 'Russia' , 'China','India
    'Year':['2010','2010','2010','2010' , '2010','2010','2010','2010','2015','2015','2015',

    'Literacy/GDP':['GDP' , 'GDP' , 'GDP' , 'GDP','Literacy' , 'Literacy', 'Literacy' , 'Li
   'Value':[7,8,7,6,80,90,89,87,6,7,8, 8, 88 , 92 , 91 ,93]}

df7 = pd.DataFrame(data,columns=['Country','Year','Literacy/GDP','Value'])
df7
```

Out[714]:

	Country	Year	Literacy/GDP	Value
0	India	2010	GDP	7
1	USA	2010	GDP	8
2	Russia	2010	GDP	7
3	China	2010	GDP	6
4	India	2010	Literacy	80
5	USA	2010	Literacy	90
6	Russia	2010	Literacy	89
7	China	2010	Literacy	87
8	India	2015	GDP	6
9	USA	2015	GDP	7
10	Russia	2015	GDP	8
11	China	2015	GDP	8
12	India	2015	Literacy	88
13	USA	2015	Literacy	92
14	Russia	2015	Literacy	91
15	China	2015	Literacy	93

In [715]:

```
# Pivot table with SUM aggregation
pd.pivot_table(df7 , index= ['Year' , 'Literacy/GDP'] , aggfunc='sum')
```

Out[715]:

		Value
Year	**Literacy/GDP**	
2010	**GDP**	28
	Literacy	346
2015	**GDP**	29
	Literacy	364

In [716]:

```
# Pivot table with MEAN aggregation
pd.pivot_table(df7 , index= ['Year' , 'Literacy/GDP'] , aggfunc='mean')
```

Out[716]:

		Value
Year	**Literacy/GDP**	
2010	**GDP**	7.00
	Literacy	86.50
2015	**GDP**	7.25
	Literacy	91.00

Hierarchical indexing

In [717]:

```
df7
```

Out[717]:

	Country	Year	Literacy/GDP	Value
0	India	2010	GDP	7
1	USA	2010	GDP	8
2	Russia	2010	GDP	7
3	China	2010	GDP	6
4	India	2010	Literacy	80
5	USA	2010	Literacy	90
6	Russia	2010	Literacy	89
7	China	2010	Literacy	87
8	India	2015	GDP	6
9	USA	2015	GDP	7
10	Russia	2015	GDP	8
11	China	2015	GDP	8
12	India	2015	Literacy	88
13	USA	2015	Literacy	92
14	Russia	2015	Literacy	91
15	China	2015	Literacy	93

In [718]:

```
df8=df7.set_index(['Year', 'Literacy/GDP'])
df8
```

Out[718]:

		Country	Value
Year	**Literacy/GDP**		
2010	**GDP**	India	7
	GDP	USA	8
	GDP	Russia	7
	GDP	China	6
	Literacy	India	80
	Literacy	USA	90
	Literacy	Russia	89
	Literacy	China	87
2015	**GDP**	India	6
	GDP	USA	7
	GDP	Russia	8
	GDP	China	8
	Literacy	India	88
	Literacy	USA	92
	Literacy	Russia	91
	Literacy	China	93

In [719]:

```
df8.index
```

Out[719]:

```
MultiIndex(levels=[['2010', '2015'], ['GDP', 'Literacy']],
           labels=[[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1], [0, 0,
0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]],
           names=['Year', 'Literacy/GDP'])
```

In [720]:

```
df8.loc['2010']
```

Out[720]:

	Country	Value
Literacy/GDP		
GDP	India	7
GDP	USA	8
GDP	Russia	7
GDP	China	6
Literacy	India	80
Literacy	USA	90
Literacy	Russia	89
Literacy	China	87

In [721]:

```
df8.loc[['2010']]
```

Out[721]:

		Country	Value
Year	**Literacy/GDP**		
2010	**GDP**	India	7
	GDP	USA	8
	GDP	Russia	7
	GDP	China	6
	Literacy	India	80
	Literacy	USA	90
	Literacy	Russia	89
	Literacy	China	87

In [722]:

```
df8.loc['2015','Literacy']
```

Out[722]:

		Country	Value
Year	**Literacy/GDP**		
2015	**Literacy**	India	88
	Literacy	USA	92
	Literacy	Russia	91
	Literacy	China	93

In [723]:

```
df8.loc['2015','Literacy']
```

Out[723]:

		Country	Value
Year	**Literacy/GDP**		
2015	**Literacy**	India	88
	Literacy	USA	92
	Literacy	Russia	91
	Literacy	China	93

In [724]:

```
df8=df7.set_index(['Year', 'Literacy/GDP' , 'Country'])
df8
```

Out[724]:

```
                              Value
Year  Literacy/GDP  Country
2010           GDP    India       7
                        USA       8
                     Russia       7
                      China       6
          Literacy    India      80
                        USA      90
                     Russia      89
                      China      87
2015           GDP    India       6
                        USA       7
                     Russia       8
                      China       8
          Literacy    India      88
                        USA      92
                     Russia      91
                      China      93
```

SWAP Columns in Hierarchical indexing

In [725]:

```
df7
```

Out[725]:

	Country	Year	Literacy/GDP	Value
0	India	2010	GDP	7
1	USA	2010	GDP	8
2	Russia	2010	GDP	7
3	China	2010	GDP	6
4	India	2010	Literacy	80
5	USA	2010	Literacy	90
6	Russia	2010	Literacy	89
7	China	2010	Literacy	87
8	India	2015	GDP	6
9	USA	2015	GDP	7
10	Russia	2015	GDP	8
11	China	2015	GDP	8
12	India	2015	Literacy	88
13	USA	2015	Literacy	92
14	Russia	2015	Literacy	91
15	China	2015	Literacy	93

In [726]:

```
df8=df7.set_index(['Year', 'Literacy/GDP'])
df8
```

Out[726]:

```
                       Country  Value
Year  Literacy/GDP
2010           GDP       India      7
               GDP         USA      8
               GDP      Russia      7
               GDP       China      6
          Literacy       India     80
          Literacy         USA     90
          Literacy      Russia     89
          Literacy       China     87
2015           GDP       India      6
               GDP         USA      7
               GDP      Russia      8
               GDP       China      8
          Literacy       India     88
          Literacy         USA     92
          Literacy      Russia     91
          Literacy       China     93
```

In [727]:

```
# Swaping the columns in Hierarchical index
df9 = df8.swaplevel('Year', 'Literacy/GDP')
df9
```

Out[727]:

		Country	Value
Literacy/GDP	**Year**		
GDP	**2010**	India	7
	2010	USA	8
	2010	Russia	7
	2010	China	6
Literacy	**2010**	India	80
	2010	USA	90
	2010	Russia	89
	2010	China	87
GDP	**2015**	India	6
	2015	USA	7
	2015	Russia	8
	2015	China	8
Literacy	**2015**	India	88
	2015	USA	92
	2015	Russia	91
	2015	China	93

In [728]:

```
# Swaping the columns in Hierarchical index
df9 = df9.swaplevel('Year', 'Literacy/GDP')
df9
```

Out[728]:

		Country	Value
Year	**Literacy/GDP**		
2010	**GDP**	India	7
	GDP	USA	8
	GDP	Russia	7
	GDP	China	6
	Literacy	India	80
	Literacy	USA	90
	Literacy	Russia	89
	Literacy	China	87
2015	**GDP**	India	6
	GDP	USA	7
	GDP	Russia	8
	GDP	China	8
	Literacy	India	88
	Literacy	USA	92
	Literacy	Russia	91
	Literacy	China	93

Crosstab in Pandas

In [729]:

```
df7
```

Out[729]:

	Country	Year	Literacy/GDP	Value
0	India	2010	GDP	7
1	USA	2010	GDP	8
2	Russia	2010	GDP	7
3	China	2010	GDP	6
4	India	2010	Literacy	80
5	USA	2010	Literacy	90
6	Russia	2010	Literacy	89
7	China	2010	Literacy	87
8	India	2015	GDP	6
9	USA	2015	GDP	7
10	Russia	2015	GDP	8
11	China	2015	GDP	8
12	India	2015	Literacy	88
13	USA	2015	Literacy	92
14	Russia	2015	Literacy	91
15	China	2015	Literacy	93

In [732]:

```
pd.crosstab(df7['Literacy/GDP'] , df7.Value , margins=True)
```

Out[732]:

Value	6	7	8	80	87	88	89	90	91	92	93	All
Literacy/GDP												
GDP	2	3	3	0	0	0	0	0	0	0	0	8
Literacy	0	0	0	1	1	1	1	1	1	1	1	8
All	2	3	3	1	1	1	1	1	1	1	1	16

In [733]:

```
# 2 way cross table
pd.crosstab(df7.Year , df7['Literacy/GDP'] , margins=True)
```

Out[733]:

Literacy/GDP	GDP	Literacy	All
Year			
2010	4	4	8
2015	4	4	8
All	8	8	16

In [734]:

```
# 3 way cross table
pd.crosstab([df7.Year , df7['Literacy/GDP']] , df7.Country, margins=True)
```

Out[734]:

	Country	China	India	Russia	USA	All
Year	**Literacy/GDP**					
2010	**GDP**	1	1	1	1	4
	Literacy	1	1	1	1	4
2015	**GDP**	1	1	1	1	4
	Literacy	1	1	1	1	4
All		4	4	4	4	16

Row & Column Bind

Row Bind

In [735]:

```
df8 = pd.DataFrame({'ID' :[1,2,3,4] ,  'Name' :['Asif' , 'Basit' , 'Ross' , 'John'] , 'Scor
df8
```

Out[735]:

	ID	Name	Score
0	1	Asif	99
1	2	Basit	66
2	3	Ross	44
3	4	John	33

In [736]:

```
df9 = pd.DataFrame({'ID' :[5,6,7,8] ,  'Name' :['Michelle' , 'Ramiro' , 'Vignesh' , 'Damon'
df9
```

Out[736]:

	ID	Name	Score
0	5	Michelle	78
1	6	Ramiro	54
2	7	Vignesh	77
3	8	Damon	87

In [737]:

```
# Row Bind with concat() function
pd.concat([df8 , df9])
```

Out[737]:

	ID	Name	Score
0	1	Asif	99
1	2	Basit	66
2	3	Ross	44
3	4	John	33
0	5	Michelle	78
1	6	Ramiro	54
2	7	Vignesh	77
3	8	Damon	87

In [738]:

```
# Row Bind with append() function
df8.append(df9)
```

Out[738]:

	ID	Name	Score
0	1	Asif	99
1	2	Basit	66
2	3	Ross	44
3	4	John	33
0	5	Michelle	78
1	6	Ramiro	54
2	7	Vignesh	77
3	8	Damon	87

Column Bind

In [739]:

```
df10 = pd.DataFrame({'ID' :[1,2,3,4] , 'Name' :['Asif' , 'Basit' , 'Ross' , 'John']})
df10
```

Out[739]:

	ID	Name
0	1	Asif
1	2	Basit
2	3	Ross
3	4	John

In [740]:

```
df11 = pd.DataFrame({'Age' :[20,30,35,40] , 'Score' :[99 , 66 , 44 , 33]})
df11
```

Out[740]:

	Age	Score
0	20	99
1	30	66
2	35	44
3	40	33

In [741]:

```
pd.concat([df10,df11] , axis = 1)
```

Out[741]:

	ID	Name	Age	Score
0	1	Asif	20	99
1	2	Basit	30	66
2	3	Ross	35	44
3	4	John	40	33