Pandas

By Nirav Prajapati
LinkedIn: (https://www.linkedin.com/in/nirav07/)

Table of Contents

In [275]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import glob
import re
import math
```

In [2]:
```
# For apply Theme
    # ! pip install --upgrade jupyterthemes
    #jt -t oceans16
# For reset theme
    # !jt -r
```

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

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

In [4]:
```
#Datatype of Series
s1.dtype
```

Out[4]:
```
dtype('int32')
```

In [5]:
```
# number of bytes allocated to each item
s1.itemsize
```

```
D:\Software_installed\python 3.8.3\lib\site-packages\ipykernel_launcher.py:2: F
utureWarning: Series.itemsize is deprecated and will be removed in a future ver
sion
```

Out[5]:
```
4
```

In [6]:
```
# number of bytes consumed by series
s1.nbytes
```

Out[6]:
```
28
```

In [7]:
```
# shape of series
s1.shape
```

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

In [8]:
```
# number of dimensions
s1.ndim
```

Out[8]:
```
1
```

In [9]:
```
# Length of Series
len(s1)
```

Out[9]:
```
7
```

In [10]:
```
s1.count()
```

Out[10]:
```
7
```

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

Out[11]:
```
7
```

In [12]:
```
# Create Series from list
s0 = pd.Series([1,2,3], index = ['a','b','c'])
s0
```

Out[12]:
```
a    1
b    2
c    3
dtype: int64
```

In [13]:
```
# Modifying index in Series
X= np.array(['a','b','c','d','e','f','g'])
s1.index = X
s1
```

Out[13]:
```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
dtype: int32
```

In [14]:
```
# Creating series using Random and Range function
v2 = np.random.random(10)
ind2 = np.arange(0,10)
s = pd.Series(v2,ind2)
v2
```

Out[14]:
```
array([0.2557229 , 0.33307615, 0.52462706, 0.95613077, 0.93835988,
       0.28024331, 0.32528214, 0.66792633, 0.92824013, 0.22945246])
```

In [15]:
```
ind2,v2
```

Out[15]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([0.2557229 , 0.33307615, 0.52462706, 0.95613077, 0.93835988,
        0.28024331, 0.32528214, 0.66792633, 0.92824013, 0.22945246]))
```

In [16]:
```
dict = {'a1':10, 'a2':20, 'a3':30, 'a4':40}
s3 = pd.Series(dict)
s3
```

Out[16]:
```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [17]:
```
pd.Series(99, index = [0,1,2,3,4,5])
```

Out[17]:
```
0    99
1    99
2    99
3    99
4    99
5    99
dtype: int64
```

1 Different Things on Series with Pandas

1.1 Slicing Series

In [18]:
```
s
```

Out[18]:
```
0    0.255723
1    0.333076
2    0.524627
3    0.956131
4    0.938360
5    0.280243
6    0.325282
7    0.667926
8    0.928240
9    0.229452
dtype: float64
```

In [19]:
```
# Return all elements of the series
s[:]
```

Out[19]:
```
0    0.255723
1    0.333076
2    0.524627
3    0.956131
4    0.938360
5    0.280243
6    0.325282
7    0.667926
8    0.928240
9    0.229452
dtype: float64
```

In [20]:
```
# First three element of the Series
s[0:3]
```

Out[20]:
```
0    0.255723
1    0.333076
2    0.524627
dtype: float64
```

In [21]:
```
# Last element of the Series
s[-1:]
```

Out[21]:
```
9    0.229452
dtype: float64
```

In [22]:
```
# Fetch first 4 elements in a series
s[:4]
```

Out[22]:
```
0    0.255723
1    0.333076
2    0.524627
3    0.956131
dtype: float64
```

In [23]:
```
# Return all elements of the series except last two elements.
s[:-2]
```

Out[23]:
```
0    0.255723
1    0.333076
2    0.524627
3    0.956131
4    0.938360
5    0.280243
6    0.325282
7    0.667926
dtype: float64
```

In [24]:
```
# Return all elements of the series except last element.
s[:-1]
```

Out[24]:
```
0    0.255723
1    0.333076
2    0.524627
3    0.956131
4    0.938360
5    0.280243
6    0.325282
7    0.667926
8    0.928240
dtype: float64
```

In [25]:
```
# Return last two elements of the series
s[-2:]
```

Out[25]:
```
8    0.928240
9    0.229452
dtype: float64
```

In [26]:
```
# Return last element of the series
s[-1:]
```

Out[26]:
```
9    0.229452
dtype: float64
```

In [27]:
```
s[-3:-1]
```

Out[27]:
```
7    0.667926
8    0.928240
dtype: float64
```

1.2 Append Series

In [28]:
```
s2 = s1.copy()
s2
```

Out[28]:
```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
dtype: int32
```

In [29]:
```
s3
```

Out[29]:
```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [30]:
```
# Append S2 & S3 Series
s4 = s2.append(s3)
s4
```

Out[30]:
```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [31]:
```
# When "inplace=False" it will return a new copy of data with the operation per
s4.drop('a4', inplace=False)
```

Out[31]:
```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
dtype: int64
```

In [32]:
```
s4
```

Out[32]:
```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

In [33]:
```
# When we use "inplace=True" it will affect the dataframe
s4.drop('a4', inplace=True)
s4
```

Out[33]:
```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
dtype: int64
```

In [34]:
```
s4 = s4.append(pd.Series({'a4':7}))
s4
```

Out[34]:
```
a      1
b      2
c      3
d      4
e      5
f      6
g      7
a1    10
a2    20
a3    30
a4     7
dtype: int64
```

1.3 Operation on Series

In [35]:
```
v1 = np.array([10,20,30])
v2 = np.array([1,2,3])
s1 = pd.Series(v1)
s2 = pd.Series(v2)
s1, s2
```

Out[35]:
```
(0    10
 1    20
 2    30
 dtype: int32, 0    1
 1    2
 2    3
 dtype: int32)
```

In [36]:
```
# Addition of two series
s1.add(s2)
```

Out[36]:
```
0    11
1    22
2    33
dtype: int32
```

In [37]:
```
# Subtraction of two series
s1.sub(s2)
```

Out[37]:
```
0     9
1    18
2    27
dtype: int32
```

In [38]:
```
# Subtraction of two series
s1.subtract(s2)
```

Out[38]:
```
0     9
1    18
2    27
dtype: int32
```

In [39]:
```
# Increment all numbers in a series by 9
s1.add(9)
```

Out[39]:
```
0    19
1    29
2    39
dtype: int32
```

In [40]:
```
# Multiplication of two series
s1.mul(s2)
```

Out[40]:
```
0    10
1    40
2    90
dtype: int32
```

In [41]:
```
# Multiplication of two series
s1.multiply(s2)
```

Out[41]:
```
0    10
1    40
2    90
dtype: int32
```

In [42]:
```
# Multiply each element by 1000
s1.mul(1000)
```

Out[42]:
```
0    10000
1    20000
2    30000
dtype: int32
```

In [43]:
```
# Division
s1.divide(s2)
```

Out[43]:
```
0    10.0
1    10.0
2    10.0
dtype: float64
```

In [44]:
```
# Division
s1.div(s2)
```

Out[44]:
```
0    10.0
1    10.0
2    10.0
dtype: float64
```

In [45]:
```
# MAX number in a series
s1.max()
```

Out[45]: 30

In [46]:
```
# Min number in a series
s1.min()
```

Out[46]: 10

In [47]:
```
# Average
s1.mean()
```

Out[47]: 20.0

In [48]:
```
# median
s1.median()
```

Out[48]: 20.0

In [49]:
```
# Standard Deviation
s1.std()
```

Out[49]: 10.0

In [50]:
```
# Series comparison
s1.equals(s2)
```

Out[50]: False

In [51]:
```
s4 =s1
```

In [52]:
```
# Series comparison
s1.equals(s4)
```

Out[52]: True

In [53]:
```
s5 = pd.Series([1,1,2,2,3,3], index = [0,1,2,3,4,5])
s5
```

Out[53]:
```
0    1
1    1
2    2
3    2
4    3
5    3
dtype: int64
```

In [54]:
```
# afind frequency
s5.value_counts()
```

Out[54]:
```
3    2
2    2
1    2
dtype: int64
```

2 DataFrame

2.1 Create DataFrame

In [55]:
```
df = pd.DataFrame()
df
```

Out[55]:

In [56]:
```
# Create Dataframe using List
lang = ['Java', 'Python', 'C', 'C++' ]
df = pd.DataFrame(lang)
df
```

Out[56]:

	0
0	Java
1	Python
2	C
3	C++

In [57]:
```
# Add column in the Dataframe
rating = [1,2,3,4]
df[1] = rating
df
```

Out[57]:

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

In [58]:
```
df.columns = ['Language', 'Rating']
```

In [59]:
```
df
```

Out[59]:

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

In [60]:
```
# 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 [61]:
```
df2
```

Out[61]:

	a	b	c
0	1	2	NaN
1	5	10	20.0

In [62]:
```
df3
```

Out[62]:

	a	b
row1	1	2
row2	5	10

In [63]:
```
df4
```

Out[63]:

	a	b	c
row1	1	2	NaN
row2	5	10	20.0

In [64]:
```
df5
```

Out[64]:

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

In [65]:
```
# Create Dataframe from Dictionary
df0 = pd.DataFrame({'ID' :[1,2,3,4], 'Name' :['Aryan','Nayan','John','Rose']})
df0
```

Out[65]:

	ID	Name
0	1	Aryan
1	2	Nayan
2	3	John
3	4	Rose

In [66]:
```
# 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[66]:

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

2.2 Dataframe of Random Numbers with Date Indices

In [67]:
```
dates = pd.date_range(start='2020-01-20', end='2020-01-26')
dates
```

Out[67]:
```
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 [68]:
```
dates = pd.date_range('today',periods= 7)
dates
```

Out[68]:
```
DatetimeIndex(['2020-07-11 17:23:40.629449', '2020-07-12 17:23:40.629449',
               '2020-07-13 17:23:40.629449', '2020-07-14 17:23:40.629449',
               '2020-07-15 17:23:40.629449', '2020-07-16 17:23:40.629449',
               '2020-07-17 17:23:40.629449'],
              dtype='datetime64[ns]', freq='D')
```

In [69]:
```
dates = pd.date_range(start='2020-01-20',periods= 7)
dates
```

Out[69]:
```
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 [70]:
```
M = np.random.random((7,7))
M
```

Out[70]:
```
array([[0.68096777, 0.6898421 , 0.33241401, 0.8300275 , 0.13412361,
        0.6369128 , 0.01566236],
       [0.09504884, 0.32464077, 0.85838129, 0.8566076 , 0.78199635,
        0.45819325, 0.77872014],
       [0.66575749, 0.91854776, 0.61560233, 0.49874779, 0.76408254,
        0.1064303 , 0.8557539 ],
       [0.67773746, 0.51461509, 0.37812294, 0.82915894, 0.13519876,
        0.27856374, 0.37872909],
       [0.3849658 , 0.20786193, 0.89258391, 0.89102031, 0.92297737,
        0.7354544 , 0.51148957],
       [0.77159037, 0.16750849, 0.21146916, 0.96990117, 0.70372491,
        0.7523114 , 0.59170043],
       [0.03378018, 0.94361344, 0.74223457, 0.66373513, 0.43151117,
        0.55326425, 0.23994092]])
```

In [71]:
```
dframe = pd.DataFrame(M, index= dates)
dframe
```

Out[71]:

	0	1	2	3	4	5	6
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	0.384966	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	0.033780	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [72]:
```
#Changing Column Names
dframe.columns = ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7']
dframe
```

Out[72]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	0.384966	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	0.033780	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [73]:
```
# List Index
dframe.index
```

Out[73]:
```
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 [74]:
```
# List Column Names
dframe.columns
```

Out[74]:
```
Index(['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7'], dtype='object')
```

In [75]:
```
# Datatype of each column
dframe.dtypes
```

Out[75]:
```
C1    float64
C2    float64
C3    float64
C4    float64
C5    float64
C6    float64
C7    float64
dtype: object
```

In [76]:
```
# Sort Dataframe by Column 'C1' in Ascending Order
dframe.sort_values(by='C1')
```

Out[76]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-26	0.033780	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-24	0.384966	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700

In [77]:
```
# Sort Dataframe by Column 'C1' in Descending Order
dframe.sort_values(by='C1' , ascending=False)
```

Out[77]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-24	0.384966	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-26	0.033780	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

2.3 Delete Column in DataFrame

In [78]:
```
df1
```

Out[78]:

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

In [79]:
```
del df1['B']
```

In [80]:
```
df1
```

Out[80]:

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [81]:
```
df5
```

Out[81]:

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

In [82]:
```
# Delete Column using pop()
df5.pop('c')
```

Out[82]:
```
row1     NaN
row2    20.0
Name: c, dtype: float64
```

In [83]:
```
df5
```

Out[83]:

	a	b	d
row1	1	2	NaN
row2	5	10	NaN

2.4 Data Selection in Dataframe

In [84]:
```
df
```

Out[84]:

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

In [85]:
```
df.index = [1,2,3,4]
df
```

Out[85]:

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

In [86]:
```
# Data selection using row label
df.loc[1]
```

Out[86]:
```
Language    Java
Rating         1
Name: 1, dtype: object
```

In [87]:
```
df.iloc[1]
```

Out[87]:
```
Language    Python
Rating           2
Name: 2, dtype: object
```

In [88]:
```
df.loc[1:2]
```

Out[88]:

	Language	Rating
1	Java	1
2	Python	2

In [89]:
```
df.iloc[1:2]
```

Out[89]:

	Language	Rating
2	Python	2

In [90]:
```
# Data selection based on Condition
df.loc[df.Rating>2]
```

Out[90]:

	Language	Rating
3	C	3
4	C++	4

In [91]:
```
df1
```

Out[91]:

	A
a	1.0
b	2.0
c	3.0
d	NaN

In [92]:
```
# Row & Column label based selection
df1.loc['a']
```

Out[92]:
```
A    1.0
Name: a, dtype: float64
```

In [93]:
```
# df1.iloc['a']# This will throw error because iloc will not work on labels
```

In [94]:
```
dframe
```

Out[94]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	0.384966	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	0.033780	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [95]:
```
# Data selection using Row Label
dframe['2020-01-20' : '2020-01-22' ]
```

Out[95]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-21	0.095049	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754

In [96]:
```
# Selecting all rows & selected columns
dframe.loc[:, ['C1','C7']]
```

Out[96]:

	C1	C7
2020-01-20	0.680968	0.015662
2020-01-21	0.095049	0.778720
2020-01-22	0.665757	0.855754
2020-01-23	0.677737	0.378729
2020-01-24	0.384966	0.511490
2020-01-25	0.771590	0.591700
2020-01-26	0.033780	0.239941

In [97]:
```
#row & column label based selection
dframe.loc['2020-01-20':'2020-01-22',['C1','C7']]
```

Out[97]:

	C1	C7
2020-01-20	0.680968	0.015662
2020-01-21	0.095049	0.778720
2020-01-22	0.665757	0.855754

In [98]:
```
# Data selection based on Condition
dframe[dframe['C1']>0.5]
```

Out[98]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-22	0.665757	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700

In [99]:
```
# Data selection based on Condition
dframe[(dframe['C1'] > 0.5) & (dframe['C4']>0.5)]
```

Out[99]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	0.680968	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-23	0.677737	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-25	0.771590	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700

In [100]:
```
# Data selection using position (Integer Index based)
dframe.iloc[0][0]
```

Out[100]: 0.6809677686418316

In [101]:
```
# # Select all rows & first three columns
dframe.iloc[:,0:3]
```

Out[101]:

	C1	C2	C3
2020-01-20	0.680968	0.689842	0.332414
2020-01-21	0.095049	0.324641	0.858381
2020-01-22	0.665757	0.918548	0.615602
2020-01-23	0.677737	0.514615	0.378123
2020-01-24	0.384966	0.207862	0.892584
2020-01-25	0.771590	0.167508	0.211469
2020-01-26	0.033780	0.943613	0.742235

In [102]:
```
dframe.iloc[0][0] = 10
```

In [103]:
```
# Display all rows where C1 has value of 10 or 20
dframe[dframe['C1'].isin([10,20])]
```

Out[103]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	10.0	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662

2.5 Set Value

In [104]:
```
# Set value of 888 for all elements in column 'C1'
dframe['C1'] = 888
dframe
```

Out[104]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	0.332414	0.830028	0.134124	0.636913	0.015662
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	0.458193	0.778720
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	0.106430	0.855754
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	888	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [105]:
```
# Set value of 777 for first three rows in Column 'C6'
dframe.at[0:3,'C6']=777
dframe
```

Out[105]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	0.332414	0.830028	0.134124	777.000000	0.015662
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	777.000000	0.778720
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	0.855754
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	888	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [106]:
```
# Set value of 333 in first row and third column
dframe.iat[0,2] = 333
dframe
```

Out[106]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	333.000000	0.830028	0.134124	777.000000	0.015662
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	777.000000	0.778720
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	0.855754
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	888	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [107]:
```
dframe.iloc[0,2] = 555
dframe
```

Out[107]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	777.000000	0.015662
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	777.000000	0.778720
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	0.855754
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	888	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [108]:
```
# 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
dframe1 = dframe.copy(deep=True)
```

In [109]:
```
dframe1[(dframe1['C1']>0.5) & (dframe1['C4']>0.5)]=0
```

In [110]:
```
dframe1[dframe1['C1']==0]
```

Out[110]:

	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
2020-01-23	0	0.0	0.0	0.0	0.0	0.0	0.0
2020-01-24	0	0.0	0.0	0.0	0.0	0.0	0.0
2020-01-25	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 [111]:
```
# Replace zeros in Column C1 with 99
dframe1[dframe1['C1'].isin([0])]=99
dframe1
```

Out[111]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
2020-01-21	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.0	0.855754
2020-01-23	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
2020-01-24	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
2020-01-25	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
2020-01-26	99	99.000000	99.000000	99.000000	99.000000	99.0	99.000000

In [112]:
```
dframe
```

Out[112]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	777.000000	0.015662
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	777.000000	0.778720
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	0.855754
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	0.378729
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	0.511490
2020-01-25	888	0.167508	0.211469	0.969901	0.703725	0.752311	0.591700
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	0.239941

In [113]:
```
# Display all rows where value of C1 is 99
dframe1[dframe1['C1']==99]
```

Out[113]:

	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
2020-01-23	99	99.0	99.0	99.0	99.0	99.0	99.0
2020-01-24	99	99.0	99.0	99.0	99.0	99.0	99.0
2020-01-25	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

2.6 Dealing with NULL Values

In [114]:
```
dframe.at[0:8,'C7']=np.NaN
dframe.at[0:2,'C6']=np.NaN
dframe.at[5:6,'C5']=np.NaN
dframe
```

Out[114]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	NaN	NaN
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	NaN	NaN
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-25	888	0.167508	0.211469	0.969901	NaN	0.752311	NaN
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	NaN

In [115]:
```
# Detect Non-Missing Values
# It will return True for NOT-NULL values and False for NULL values
dframe.notna()
```

Out[115]:

	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 [116]:
```
# Detect Missing or NULL Values
# It will return True for NULL values and False for NOT-NULL values
dframe.isna()
```

Out[116]:

	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 [117]:
```
# Fill all NULL values with 1020
dframe= dframe.fillna(1020)
dframe
```

Out[117]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	1020.000000	1020.0
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	1020.000000	1020.0
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	1020.0
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	1020.0
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	1020.0
2020-01-25	888	0.167508	0.211469	0.969901	1020.000000	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [118]:
```
dframe.at[0:5 , 'C7'] = np.NaN
dframe.at[0:2 , 'C6'] = np.NaN
dframe.at[5:6 , 'C5'] = np.NaN
dframe
```

Out[118]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	NaN	NaN
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	NaN	NaN
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-25	888	0.167508	0.211469	0.969901	NaN	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [119]:
```
# 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[119]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	789.000000	NaN
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	789.000000	NaN
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-25	888	0.167508	0.211469	0.969901	123.000000	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [120]:
```
#Replace first NULL value in Column C7 with 789
dframe.fillna(value={'C7':789}, limit=1)
```

Out[120]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	NaN	789.0
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	NaN	NaN
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-25	888	0.167508	0.211469	0.969901	NaN	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [121]:
```
# Drop Rows with NULL values
dframe.dropna()
```

Out[121]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [122]:
```
# Drop Columns with NULL values
dframe.dropna(axis='columns')
```

Out[122]:

	C1	C2	C3	C4
2020-01-20	888	0.689842	555.000000	0.830028
2020-01-21	888	0.324641	0.858381	0.856608
2020-01-22	888	0.918548	0.615602	0.498748
2020-01-23	888	0.514615	0.378123	0.829159
2020-01-24	888	0.207862	0.892584	0.891020
2020-01-25	888	0.167508	0.211469	0.969901
2020-01-26	888	0.943613	0.742235	0.663735

In [123]:
```
dframe
```

Out[123]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	NaN	NaN
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	NaN	NaN
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-25	888	0.167508	0.211469	0.969901	NaN	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [124]:
```
# Drop Rows with NULL values present in C5 or C6
dframe.dropna(subset=['C5' ,'C6'])
```

Out[124]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	NaN
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	NaN
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	NaN
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

2.7 Descriptive Statistics

In [125]:
```
# Fill NULL values with 55
dframe.fillna(55, inplace=True)
dframe
```

Out[125]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	55.000000	55.0
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	55.000000	55.0
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	55.0
2020-01-23	888	0.514615	0.378123	0.829159	0.135199	0.278564	55.0
2020-01-24	888	0.207862	0.892584	0.891020	0.922977	0.735454	55.0
2020-01-25	888	0.167508	0.211469	0.969901	55.000000	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [126]:
```
# Mean of all Columns
dframe.mean()
```

Out[126]:
```
C1    888.000000
C2      0.538090
C3     79.814056
C4      0.791314
C5      8.309984
C6    127.045656
C7    330.714286
dtype: float64
```

In [127]:
```
# Max value per column
dframe.max()
```

Out[127]:
```
C1     888.000000
C2       0.943613
C3     555.000000
C4       0.969901
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [128]:
```
# Min value per column
dframe.min()
```

Out[128]:
```
C1    888.000000
C2      0.167508
C3      0.211469
C4      0.498748
C5      0.134124
C6      0.278564
C7     55.000000
dtype: float64
```

In [129]:
```
# Median
dframe.median()
```

Out[129]:
```
C1    888.000000
C2      0.514615
C3      0.742235
C4      0.830028
C5      0.764083
C6      0.752311
C7     55.000000
dtype: float64
```

In [130]:
```
#Standard Deviation
dframe.std()
```

Out[130]:
```
C1      0.000000
C2      0.322676
C3    209.537453
C4      0.158588
C5     20.590770
C6    287.748820
C7    470.871785
dtype: float64
```

In [131]:
```
# Variance
dframe.var()
```

Out[131]:
```
C1         0.000000
C2         0.104120
C3     43905.944333
C4         0.025150
C5       423.979805
C6     82799.383192
C7    221720.238095
dtype: float64
```

In [132]:
```
#Lower Quartile / First Quartile
dframe.quantile(0.25)
```

Out[132]:
```
C1    888.000000
C2      0.266251
C3      0.496863
C4      0.746447
C5      0.283355
C6      0.644359
C7     55.000000
Name: 0.25, dtype: float64
```

In [133]:
```
#Second Quartile / Median
dframe.quantile(0.5)
```

Out[133]:
```
C1    888.000000
C2      0.514615
C3      0.742235
C4      0.830028
C5      0.764083
C6      0.752311
C7     55.000000
Name: 0.5, dtype: float64
```

In [134]:
```
# Upper Quartile
dframe.quantile(0.75)
```

Out[134]:
```
C1    888.000000
C2      0.804195
C3      0.875483
C4      0.873814
C5      0.852487
C6     55.000000
C7    537.500000
Name: 0.75, dtype: float64
```

In [135]:
```
# IQR (Inter Quartile range)
dframe.quantile(0.75)-dframe.quantile(0.25)
```

Out[135]:
```
C1      0.000000
C2      0.537944
C3      0.378620
C4      0.127367
C5      0.569132
C6     54.355641
C7    482.500000
dtype: float64
```

In [136]:
```
# SUM of column values
dframe.sum()
```

Out[136]:
```
C1    6216.000000
C2       3.766630
C3     558.698394
C4       5.539198
C5      58.169890
C6     889.319594
C7    2315.000000
dtype: float64
```

In [137]:
```
# GENERATES DESCRIPTIVE STATS
dframe.describe()
```

Out[137]:

	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	0.538090	79.814056	0.791314	8.309984	127.045656	330.714286
std	0.0	0.322676	209.537453	0.158588	20.590770	287.748820	470.871785
min	888.0	0.167508	0.211469	0.498748	0.134124	0.278564	55.000000
25%	888.0	0.266251	0.496863	0.746447	0.283355	0.644359	55.000000
50%	888.0	0.514615	0.742235	0.830028	0.764083	0.752311	55.000000
75%	888.0	0.804195	0.875483	0.873814	0.852487	55.000000	537.500000
max	888.0	0.943613	555.000000	0.969901	55.000000	777.000000	1020.000000

In [138]:
```
#Return unbiased skew
#   https://www.youtube.com/watch?v=HnMGKsupF8Q
dframe.skew()
```

Out[138]:
```
C1    0.000000
C2    0.198711
C3    2.645743
C4   -1.172444
C5    2.644451
C6    2.602402
C7    1.229634
dtype: float64
```

In [139]:
```
# Return unbiased kurtosis using Fisher’s definition of kurtosis
# https://www.youtube.com/watch?v=HnMGKsupF8Q
dframe.kurt()
```

Out[139]:
```
C1    0.000000
C2   -1.897991
C3    6.999968
C4    1.040791
C5    6.994764
C6    6.820734
C7   -0.840000
dtype: float64
```

In [140]:
```
# Correlation
# https://www.youtube.com/watch?v=qtaqvPAeEJY&list=PLblh5JKOoLUK0FLuzwntyYI10UQ
dframe.corr()
```

Out[140]:

	C1	C2	C3	C4	C5	C6	C7
C1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
C2	NaN	1.000000	0.207536	-0.882971	-0.510997	0.523999	0.036988
C3	NaN	0.207536	1.000000	0.107374	-0.175931	-0.110371	-0.258654
C4	NaN	-0.882971	0.107374	1.000000	0.494935	-0.808822	0.109861
C5	NaN	-0.510997	-0.175931	0.494935	1.000000	-0.188575	0.643816
C6	NaN	0.523999	-0.110371	-0.808822	-0.188575	1.000000	-0.300063
C7	NaN	0.036988	-0.258654	0.109861	0.643816	-0.300063	1.000000

In [141]:
```
# Covariance
# https://www.youtube.com/watch?v=xZ_z8KWkhXE&list=PLblh5JKOoLUK0FLuzwntyYI10UQ
dframe.cov()
```

Out[141]:

	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	0.104120	14.032072	-0.045184	-3.395137	48.653092	5.619847
C3	0.0	14.032072	43905.944333	3.568046	-759.061432	-6654.723105	-25520.134097
C4	0.0	-0.045184	3.568046	0.025150	1.616182	-36.909328	8.203815
C5	0.0	-3.395137	-759.061432	1.616182	423.979805	-1117.303460	6242.189778
C6	0.0	48.653092	-6654.723105	-36.909328	-1117.303460	82799.383192	-40656.372679
C7	0.0	5.619847	-25520.134097	8.203815	6242.189778	-40656.372679	221720.238095

In [142]:
```
import statistics as st
dframe.at[3:6,'C1'] = 22
dframe
```

Out[142]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	55.000000	55.0
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	55.000000	55.0
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	55.0
2020-01-23	22	0.514615	0.378123	0.829159	0.135199	0.278564	55.0
2020-01-24	22	0.207862	0.892584	0.891020	0.922977	0.735454	55.0
2020-01-25	22	0.167508	0.211469	0.969901	55.000000	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [143]:
```
# Average
st.mean(dframe['C1'])
# dframe['C1'].mean()
```

Out[143]: 516.8571428571429

In [144]:
```
# Hormonic Mean
st.harmonic_mean(dframe['C1'])
```

Out[144]: 49.69186046511628

In [145]:
```
#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[145]: 4.5

In [146]:
```
# low median of the data with EVEN length
st.median_low(arr)
```

Out[146]: 4

In [147]:
```
# High median of the data with EVEN length
st.median_high(arr)
```

Out[147]: 5

In [148]:
```
# Mode of Dataset
st.mode(dframe['C7'])
```

Out[148]: 55.0

In [149]:
```
# Sample Variance
st.variance(dframe['C1'])
```

Out[149]: 214273.14285714287

In [150]:
```
#Population Variance
st.pvariance(dframe['C1'])
```

Out[150]: 183662.69387755104

In [151]:
```
#Sample Standard Deviation
st.stdev(dframe['C1'])
```

Out[151]: 462.89647099231905

In [152]:
```
#Population Standard Deviation
st.pstdev(dframe['C1'])
```

Out[152]: 428.5588569584708

3 Apply function on Dataframe

In [153]:
```
dframe
```

Out[153]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888	0.689842	555.000000	0.830028	0.134124	55.000000	55.0
2020-01-21	888	0.324641	0.858381	0.856608	0.781996	55.000000	55.0
2020-01-22	888	0.918548	0.615602	0.498748	0.764083	777.000000	55.0
2020-01-23	22	0.514615	0.378123	0.829159	0.135199	0.278564	55.0
2020-01-24	22	0.207862	0.892584	0.891020	0.922977	0.735454	55.0
2020-01-25	22	0.167508	0.211469	0.969901	55.000000	0.752311	1020.0
2020-01-26	888	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [154]:
```
# Finding MAX value in Columns
dframe.apply(max)
```

Out[154]:
```
C1     888.000000
C2       0.943613
C3     555.000000
C4       0.969901
C5      55.000000
C6     777.000000
C7    1020.000000
dtype: float64
```

In [155]:
```
# Finding minimum value in Columns
dframe.apply(min)
```

Out[155]:
```
C1    22.000000
C2     0.167508
C3     0.211469
C4     0.498748
C5     0.134124
C6     0.278564
C7    55.000000
dtype: float64
```

In [156]:
```
#Sum of Column Values
dframe.apply(sum)
```

Out[156]:
```
C1    3618.000000
C2       3.766630
C3     558.698394
C4       5.539198
C5      58.169890
C6     889.319594
C7    2315.000000
dtype: float64
```

In [157]:
```
#Sum of Column Values
dframe.apply(np.sum)
```

Out[157]:
```
C1    3618.000000
C2       3.766630
C3     558.698394
C4       5.539198
C5      58.169890
C6     889.319594
C7    2315.000000
dtype: float64
```

In [158]:
```
# sum of row
dframe.apply(np.sum, axis=1)
```

Out[158]:
```
2020-01-20    1554.653993
2020-01-21    1000.821626
2020-01-22    1722.796980
2020-01-23      79.135659
2020-01-24      80.649898
2020-01-25    1099.101190
2020-01-26    1911.334359
Freq: D, dtype: float64
```

In [159]:
```
# Square root of all values in dataframe
dframe.applymap(np.sqrt)
```

Out[159]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	29.799329	0.830567	23.558438	0.911058	0.366229	7.416198	7.416198
2020-01-21	29.799329	0.569773	0.926489	0.925531	0.884306	7.416198	7.416198
2020-01-22	29.799329	0.958409	0.784603	0.706221	0.874118	27.874720	7.416198
2020-01-23	4.690416	0.717367	0.614917	0.910582	0.367694	0.527791	7.416198
2020-01-24	4.690416	0.455919	0.944767	0.943939	0.960717	0.857586	7.416198
2020-01-25	4.690416	0.409278	0.459858	0.984836	7.416198	0.867359	31.937439
2020-01-26	29.799329	0.971398	0.861530	0.814699	0.656895	0.743817	31.937439

In [160]:
```
# Square root of all values in a DataFrame
dframe.applymap(math.sqrt)
```

Out[160]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	29.799329	0.830567	23.558438	0.911058	0.366229	7.416198	7.416198
2020-01-21	29.799329	0.569773	0.926489	0.925531	0.884306	7.416198	7.416198
2020-01-22	29.799329	0.958409	0.784603	0.706221	0.874118	27.874720	7.416198
2020-01-23	4.690416	0.717367	0.614917	0.910582	0.367694	0.527791	7.416198
2020-01-24	4.690416	0.455919	0.944767	0.943939	0.960717	0.857586	7.416198
2020-01-25	4.690416	0.409278	0.459858	0.984836	7.416198	0.867359	31.937439
2020-01-26	29.799329	0.971398	0.861530	0.814699	0.656895	0.743817	31.937439

In [161]:
```
dframe.applymap(float)
```

Out[161]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	888.0	0.689842	555.000000	0.830028	0.134124	55.000000	55.0
2020-01-21	888.0	0.324641	0.858381	0.856608	0.781996	55.000000	55.0
2020-01-22	888.0	0.918548	0.615602	0.498748	0.764083	777.000000	55.0
2020-01-23	22.0	0.514615	0.378123	0.829159	0.135199	0.278564	55.0
2020-01-24	22.0	0.207862	0.892584	0.891020	0.922977	0.735454	55.0
2020-01-25	22.0	0.167508	0.211469	0.969901	55.000000	0.752311	1020.0
2020-01-26	888.0	0.943613	0.742235	0.663735	0.431511	0.553264	1020.0

In [162]:
```
# Using Lambda function in Dataframes for find min value
dframe.apply(lambda x: min(x))
```

Out[162]:
```
C1    22.000000
C2     0.167508
C3     0.211469
C4     0.498748
C5     0.134124
C6     0.278564
C7    55.000000
dtype: float64
```

In [163]:
```
# Using Lambda function in Dataframes for square all values
dframe.apply(lambda x: x*x)
```

Out[163]:

	C1	C2	C3	C4	C5	C6	C7
2020-01-20	788544	0.475882	308025.000000	0.688946	0.017989	3025.000000	3025.0
2020-01-21	788544	0.105392	0.736818	0.733777	0.611518	3025.000000	3025.0
2020-01-22	788544	0.843730	0.378966	0.248749	0.583822	603729.000000	3025.0
2020-01-23	484	0.264829	0.142977	0.687505	0.018279	0.077598	3025.0
2020-01-24	484	0.043207	0.796706	0.793917	0.851887	0.540893	3025.0
2020-01-25	484	0.028059	0.044719	0.940708	3025.000000	0.565972	1040400.0
2020-01-26	788544	0.890406	0.550912	0.440544	0.186202	0.306101	1040400.0

3.1 Merge Dataframes

In [164]:
```
daf1 = pd.DataFrame({'Id': ['1','2','3','4','5'], 'Name': ['Aryan','Rose','Brar
daf1
```

Out[164]:

	Id	Name
0	1	Aryan
1	2	Rose
2	3	Bran
3	4	Shiv
4	5	Joy

In [165]:
```
daf2 = pd.DataFrame({'Id':['1','2','6','7','8'], 'Score':[40, 60, 80, 90, 70]})
daf2
```

Out[165]:

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

In [166]:
```
# Inner Join
pd.merge(daf1,daf2, on = 'Id',how ='inner')
```

Out[166]:

	Id	Name	Score
0	1	Aryan	40
1	2	Rose	60

In [167]:
```
# Full Outer Join
pd.merge(daf1,daf2, on= 'Id', how = 'outer')
```

Out[167]:

	Id	Name	Score
0	1	Aryan	40.0
1	2	Rose	60.0
2	3	Bran	NaN
3	4	Shiv	NaN
4	5	Joy	NaN
5	6	NaN	80.0
6	7	NaN	90.0
7	8	NaN	70.0

In [168]:
```
# Left Outer Join
pd.merge(daf1,daf2, on= 'Id', how='left')
```

Out[168]:

	Id	Name	Score
0	1	Aryan	40.0
1	2	Rose	60.0
2	3	Bran	NaN
3	4	Shiv	NaN
4	5	Joy	NaN

In [169]:
```
# Left Outer Join
pd.merge(daf1,daf2, on= 'Id', how='right')
```

Out[169]:

	Id	Name	Score
0	1	Aryan	40
1	2	Rose	60
2	6	NaN	80
3	7	NaN	90
4	8	NaN	70

3.2 Importing multiple CSV files in DataFrame

In [170]:
```
# Append all CSV files
csv1 = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/m
csv2 = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/m
csv3 = pd.read_csv('https://raw.githubusercontent.com/CSSEGISandData/COVID-19/m

# csv1.to_csv('/Data_File1.csv')
# csv2.to_csv('/Data_File2.csv')
# csv3.to_csv('/Data_File3.csv')
```

In [171]:
```
# Append all CSV files
path =r'D:\\MObile\\'
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 [172]:
```
# Top 10 rows of the Dataframe
covid.head(3)
```

Out[172]:

	#	Active	Admin2	Attack	Case-Fatality_Ratio	Combined_Key	Confirmed	Country_R
0	NaN	133.0	Abbeville	NaN	0.746269	Abbeville, South Carolina, US	134.0	
1	NaN	1025.0	Acadia	NaN	4.026217	Acadia, Louisiana, US	1068.0	
2	NaN	1028.0	Accomack	NaN	1.343570	Accomack, Virginia, US	1042.0	

3 rows × 28 columns

In [173]:
```
# Bottom 10 rows of the Dataframe
covid.tail(3)
```

Out[173]:

	#	Active	Admin2	Attack	Case-Fatality_Ratio	Combined_Key	Confirmed	Count
13801	720.0	NaN	NaN	110.0	NaN	NaN	NaN	
13802	720.0	NaN	NaN	160.0	NaN	NaN	NaN	
13803	721.0	NaN	NaN	110.0	NaN	NaN	NaN	

3 rows × 28 columns

In [174]:
```
# Reading columns
covid['Country_Region'].head(3)
```

Out[174]:
```
0    US
1    US
2    US
Name: Country_Region, dtype: object
```

In [175]:
```
# Reading columns
df1 = covid[['Country_Region' ,'Province_State','Confirmed' , 'Last_Update']]
df1.head(5)
```

Out[175]:

	Country_Region	Province_State	Confirmed	Last_Update
0	US	South Carolina	134.0	2020-07-08 05:33:48
1	US	Louisiana	1068.0	2020-07-08 05:33:48
2	US	Virginia	1042.0	2020-07-08 05:33:48
3	US	Idaho	3252.0	2020-07-08 05:33:48
4	US	Iowa	16.0	2020-07-08 05:33:48

In [176]:
```
#Read specific rows
df1.iloc[1:4]
```

Out[176]:

	Country_Region	Province_State	Confirmed	Last_Update
1	US	Louisiana	1068.0	2020-07-08 05:33:48
2	US	Virginia	1042.0	2020-07-08 05:33:48
3	US	Idaho	3252.0	2020-07-08 05:33:48

In [177]:
```
#Filter data
df1.loc[df1['Country_Region']=='India']
```

Out[177]:

	Country_Region	Province_State	Confirmed	Last_Update
3141	India	Andaman and Nicobar Islands	147.0	2020-07-08 05:33:48
3142	India	Andhra Pradesh	21197.0	2020-07-08 05:33:48
3156	India	Arunachal Pradesh	276.0	2020-07-08 05:33:48
3157	India	Assam	12522.0	2020-07-08 05:33:48
3179	India	Bihar	12570.0	2020-07-08 05:33:48
...	...	...	...	...
11174	India	Tripura	1568.0	2020-07-06 04:33:57
11187	India	Unknown	4913.0	2020-07-06 04:33:57
11193	India	Uttar Pradesh	27707.0	2020-07-06 04:33:57
11194	India	Uttarakhand	3124.0	2020-07-06 04:33:57
11217	India	West Bengal	22126.0	2020-07-06 04:33:57

108 rows × 4 columns

In [178]:
```
#Sort Data Frame
display('Sorted Data Frame', df1.sort_values(['Country_Region'], ascending=True
```

'Sorted Data Frame'

	Country_Region	Province_State	Confirmed	Last_Update
7434	Afghanistan	NaN	33190.0	2020-07-07 04:34:00
11234	Afghanistan	NaN	32951.0	2020-07-06 04:33:57
3632	Afghanistan	NaN	33384.0	2020-07-08 05:33:48
11235	Albania	NaN	2893.0	2020-07-06 04:33:57
3633	Albania	NaN	3038.0	2020-07-08 05:33:48

In [179]:
```
#Sort Data Frame
display('Sorted Data Frame', df1.sort_values(['Country_Region'], ascending=Fals
```

'Sorted Data Frame'

	Country_Region	Province_State	Confirmed	Last_Update
7602	Zimbabwe	NaN	734.0	2020-07-07 04:34:00
11402	Zimbabwe	NaN	716.0	2020-07-06 04:33:57
3800	Zimbabwe	NaN	787.0	2020-07-08 05:33:48
3799	Zambia	NaN	1895.0	2020-07-08 05:33:48
7601	Zambia	NaN	1632.0	2020-07-07 04:34:00

In [180]:
```
#Sort Data Frame - Ascending on "Country" & descending on "Last update"
display('Sorted data Frame', df1.sort_values(['Country_Region', 'Last_Update'],
```

'Sorted data Frame'

	Country_Region	Province_State	Confirmed	Last_Update
3632	Afghanistan	NaN	33384.0	2020-07-08 05:33:48
7434	Afghanistan	NaN	33190.0	2020-07-07 04:34:00
11234	Afghanistan	NaN	32951.0	2020-07-06 04:33:57
3633	Albania	NaN	3038.0	2020-07-08 05:33:48
7435	Albania	NaN	2964.0	2020-07-07 04:34:00
11235	Albania	NaN	2893.0	2020-07-06 04:33:57

In [181]:
```
#Iterating through the dataset
for index, row in df1.iterrows():
    if(row['Country_Region']=='Indonesia'):
        display(row[['Country_Region','Confirmed']])
```

```
Country_Region    Indonesia
Confirmed             66226
Name: 3704, dtype: object

Country_Region    Indonesia
Confirmed             64958
Name: 7506, dtype: object

Country_Region    Indonesia
Confirmed             63749
Name: 11306, dtype: object
```

In [182]:
```
#Unique Values
covid['Country_Region'].drop_duplicates(keep='first').head(10)
```

Out[182]:
```
0            US
3124      Italy
3125     Brazil
3126     Russia
3127     Mexico
3128      Japan
3131     Canada
3136   Colombia
3137       Peru
3140      Spain
Name: Country_Region, dtype: object
```

In [183]:
```
# Countries impacted with Coronavirus
countries= covid['Country_Region'].unique()
type(countries), countries
```

Out[183]:
```
(numpy.ndarray,
 array(['US', 'Italy', 'Brazil', 'Russia', 'Mexico', 'Japan', 'Canada',
        'Colombia', 'Peru', 'Spain', 'India', 'United Kingdom', 'China',
        'Chile', 'Netherlands', 'Australia', 'Pakistan', 'Germany',
        'Sweden', 'Ukraine', 'Denmark', 'France', 'Afghanistan', 'Albania',
        'Algeria', 'Andorra', 'Angola', 'Antigua and Barbuda', 'Argentina',
        'Armenia', 'Austria', 'Azerbaijan', 'Bahamas', 'Bahrain',
        'Bangladesh', 'Barbados', 'Belarus', 'Belgium', 'Belize', 'Benin',
        'Bhutan', 'Bolivia', 'Bosnia and Herzegovina', 'Botswana',
        'Brunei', 'Bulgaria', 'Burkina Faso', 'Burma', 'Burundi',
        'Cabo Verde', 'Cambodia', 'Cameroon', 'Central African Republic',
        'Chad', 'Comoros', 'Congo (Brazzaville)', 'Congo (Kinshasa)',
        'Costa Rica', "Cote d'Ivoire", 'Croatia', 'Cuba', 'Cyprus',
        'Czechia', 'Diamond Princess', 'Djibouti', 'Dominica',
        'Dominican Republic', 'Ecuador', 'Egypt', 'El Salvador',
        'Equatorial Guinea', 'Eritrea', 'Estonia', 'Eswatini', 'Ethiopia',
        'Fiji', 'Finland', 'Gabon', 'Gambia', 'Georgia', 'Ghana', 'Greece',
        'Grenada', 'Guatemala', 'Guinea', 'Guinea-Bissau', 'Guyana',
        'Haiti', 'Holy See', 'Honduras', 'Hungary', 'Iceland', 'Indonesia',
        'Iran', 'Iraq', 'Ireland', 'Israel', 'Jamaica', 'Jordan',
        'Kazakhstan', 'Kenya', 'Korea, South', 'Kosovo', 'Kuwait',
        'Kyrgyzstan', 'Laos', 'Latvia', 'Lebanon', 'Lesotho', 'Liberia',
        'Libya', 'Liechtenstein', 'Lithuania', 'Luxembourg', 'MS Zaandam',
        'Madagascar', 'Malawi', 'Malaysia', 'Maldives', 'Mali', 'Malta',
        'Mauritania', 'Mauritius', 'Moldova', 'Monaco', 'Mongolia',
        'Montenegro', 'Morocco', 'Mozambique', 'Namibia', 'Nepal',
        'New Zealand', 'Nicaragua', 'Niger', 'Nigeria', 'North Macedonia',
        'Norway', 'Oman', 'Panama', 'Papua New Guinea', 'Paraguay',
        'Philippines', 'Poland', 'Portugal', 'Qatar', 'Romania', 'Rwanda',
        'Saint Kitts and Nevis', 'Saint Lucia',
        'Saint Vincent and the Grenadines', 'San Marino',
        'Sao Tome and Principe', 'Saudi Arabia', 'Senegal', 'Serbia',
        'Seychelles', 'Sierra Leone', 'Singapore', 'Slovakia', 'Slovenia',
        'Somalia', 'South Africa', 'South Sudan', 'Sri Lanka', 'Sudan',
        'Suriname', 'Switzerland', 'Syria', 'Taiwan*', 'Tajikistan',
        'Tanzania', 'Thailand', 'Timor-Leste', 'Togo',
        'Trinidad and Tobago', 'Tunisia', 'Turkey', 'Uganda',
        'United Arab Emirates', 'Uruguay', 'Uzbekistan', 'Venezuela',
        'Vietnam', 'West Bank and Gaza', 'Western Sahara', 'Yemen',
        'Zambia', 'Zimbabwe', nan], dtype=object))
```

In [184]:
```
#https://data.world/data-society/pokemon-with-stats

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

Out[184]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	4
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	6
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	8
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	8
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	6

In [185]:
```
# Sum of Columns
df2['Total'] = df2['HP'] + df2['Attack']
df2.head(5)
```

Out[185]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	1	Bulbasaur	Grass	Poison	94	45	49	49	65	65	4
1	2	Ivysaur	Grass	Poison	122	60	62	63	80	80	6
2	3	Venusaur	Grass	Poison	162	80	82	83	100	100	8
3	3	VenusaurMega Venusaur	Grass	Poison	180	80	100	123	122	120	8
4	4	Charmander	Fire	NaN	91	39	52	43	60	50	6

In [186]:
```
# Sum of Columns
df2['Total'] = df2.iloc[:,5:11].sum(axis=1)
df2.head()
```

Out[186]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	4
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	6
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	8
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	8
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	6

In [187]:
```
#Shifting "Total" column

cols = list(df2.columns)

df2 = df2[cols[:4] + cols[5:11] + [cols[4]] + cols[11:]]
df2.head()
```

Out[187]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Total
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	3
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	4
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	5
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	6
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	3

In [188]:
```
#Shifting "Legendary" column - Its Index location -1 or 12

cols = list(df2.columns)

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

Out[188]:

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

In [189]:
```
#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[189]:

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

In [190]:
```
#Save to CSV file
df2.to_csv('poke_updated.csv')
```

In [191]:
```
#Save to CSV file without index column
df2.to_csv('poke_updated1.csv', index=False)
```

In [192]:
```
df2.head(7)
```

Out[192]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	

In [193]:
```
# Save Dataframe as text file
df2.to_csv('poke.txt', sep = '\t', index= False)
```

In [194]:
```
# Save Dataframe as xlsx file
df2.to_excel('poke.xlsx')
```

In [195]:
```
# Save Dataframe as xlsx file without row names
df2.to_excel('poke.xlsx',index = 0)
```

In [196]:
```
#Filtering using loc
df2.loc[df2['Type 2']=='Dragon']
```

Out[196]:

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

In [197]:
```
#Filtering using loc
df3 = df2.loc[(df2['Type 2']=='Dragon') & (df2['Type 1']=='Dark')]
df3
```

Out[197]:

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

In [198]:
```
#Reset index for Dataframe df3 keeping old index column
df4 = df3.reset_index()
df4
```

Out[198]:

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

In [199]:
```
#Reset index for Dataframe df3 removing old index column
df3.reset_index(drop=True, inplace=True)
df3
```

Out[199]:

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

In [200]:
```
df2.head(5)
```

Out[200]:

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

4 LIKE OPERATION IN PANDAS

In [201]:
```
df2.Name.str.contains('rill').head(7)
```

Out[201]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
Name: Name, dtype: bool
```

In [202]:
```
# Display all rows containing Name "rill"
df2.loc[df2.Name.str.contains("rill")]
```

Out[202]:

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

In [203]:
```
# Exclude all rows containing "rill"
df2.loc[~df2.Name.str.contains("rill")].head(7)
```

Out[203]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	

In [204]:
```
#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("Poiso
```

Out[204]:

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

In [205]:
```
df2.loc[df2['Type 1'].str.contains('Grass|Water',regex = True)].head(7)
```

Out[205]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	

In [206]:
```
# Due to Case-sensitive it will not return any data
df2.loc[df2['Type 1'].str.contains("grass|water",regex=True)].head(7)
```

Out[206]:

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

In [207]:
```
# To ignore case we can use "case = False"
df2.loc[df2['Type 1'].str.contains("grass|water",case=False,regex=True)].head(7
```

Out[207]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	

In [208]:
```
# To ignore case we can use "Flags = re.I"
df2.loc[df2['Type 1'].str.contains("grass|water", flags = re.I, regex=True)].he
```

Out[208]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	
10	8	Wartortle	Water	NaN	59	63	80	65	80	58	
11	9	Blastoise	Water	NaN	79	83	100	85	105	78	

5 Regex in Pandas dataframe

In [209]:
```
#Get all rows with name starting with "wa"
df2.loc[df2.Name.str.contains('^Wa', flags =re.I, regex = True)].head(7)
```

Out[209]:

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

In [210]:
```
#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(7)
```

Out[210]:

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

In [211]:
```
#Get all rows with name starting with x , y, z
df2.loc[df2.Name.str.contains('^[x-z]', flags =re.I,regex=True)]
```

Out[211]:

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

In [212]:
```
# Extracting first 3 characters from "Name" column
df2['Name2'] = df2.Name.str.extract(r'(^\w{3})')
df2.head(5)
```

Out[212]:

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

In [213]:
```
# Return all rows with "Name" starting with character 'B or b'
df2.loc[df2.Name.str.match(r'(^[B|b].*)')].head(5)
```

Out[213]:

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

6 Replace values in dataframe

In [214]:
```
df2.head(7)
```

Out[214]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	
4	4	Charmander	Fire	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Fire	NaN	58	64	58	80	65	80	
6	6	Charizard	Fire	Flying	78	84	78	109	85	100	

In [215]:
```
df2['Type 1'] = df2['Type 1'].replace({"Grass": "Medow", "Fire":"Blaze"})
df2.head(7)
```

Out[215]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gener
0	1	Bulbasaur	Medow	Poison	45	49	49	65	65	45	
1	2	Ivysaur	Medow	Poison	60	62	63	80	80	60	
2	3	Venusaur	Medow	Poison	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Medow	Poison	80	100	123	122	120	80	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	

In [216]:
```
df2['Type 2'] = df2['Type 2'].replace({"Poison": "Venom"})
df2.head()
```

Out[216]:

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

In [217]:
```
df2["Type 2"] = df2['Type 2'].replace(['Venom','Dragon'],'DANGER')
df2.head(7)
```

Out[217]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
0	1	Bulbasaur	Medow	DANGER	45	49	49	65	65	45	
1	2	Ivysaur	Medow	DANGER	60	62	63	80	80	60	
2	3	Venusaur	Medow	DANGER	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Medow	DANGER	80	100	123	122	120	80	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	

In [218]:
```
df2.loc[df2['Type 2'] == 'DANGER', 'Name2'] =np.NaN
df2.head(7)
```

Out[218]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
0	1	Bulbasaur	Medow	DANGER	45	49	49	65	65	45	
1	2	Ivysaur	Medow	DANGER	60	62	63	80	80	60	
2	3	Venusaur	Medow	DANGER	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Medow	DANGER	80	100	123	122	120	80	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	

In [219]:
```
df2.loc[df2['Total'] > 400, ['Name2', 'Legendary']]='ALERT'
df2.head(7)
```

Out[219]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
0	1	Bulbasaur	Medow	DANGER	45	49	49	65	65	45	
1	2	Ivysaur	Medow	DANGER	60	62	63	80	80	60	
2	3	Venusaur	Medow	DANGER	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Medow	DANGER	80	100	123	122	120	80	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	

In [220]:
```
df2.loc[df2['Total'] > 100, ['Legendary','Name2']] = ['Aleart-1','Aleart-2']
df2.head(7)
```

Out[220]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
0	1	Bulbasaur	Medow	DANGER	45	49	49	65	65	45	
1	2	Ivysaur	Medow	DANGER	60	62	63	80	80	60	
2	3	Venusaur	Medow	DANGER	80	82	83	100	100	80	
3	3	VenusaurMega Venusaur	Medow	DANGER	80	100	123	122	120	80	
4	4	Charmander	Blaze	NaN	39	52	43	60	50	65	
5	5	Charmeleon	Blaze	NaN	58	64	58	80	65	80	
6	6	Charizard	Blaze	Flying	78	84	78	109	85	100	

7 Group By

In [221]:
```
df = pd.read_csv('poke_updated.csv')
df.head()
```

Out[221]:

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

In [222]:
```
df.groupby(['Type 1']).mean().head(7)
```

Out[222]:

Type 1	Unnamed: 0	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Bug	368.072464	334.492754	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159
Dark	507.387097	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290
Dragon	521.843750	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250
Electric	400.590909	363.500000	59.795455	69.090909	66.295455	90.022727	73.704545	84.500000
Fairy	494.529412	449.529412	74.117647	61.529412	65.705882	78.529412	84.705882	48.588235
Fighting	400.444444	363.851852	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074
Fire	360.942308	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308

In [223]:
```
df.groupby(['Type 1']).mean().sort_values('Attack', ascending = False).head(7)
```

Out[223]:

Type 1	Unnamed: 0	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Dragon	521.843750	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250
Fighting	400.444444	363.851852	69.851852	96.777778	65.925926	53.111111	64.703704	66.074074
Ground	392.312500	356.281250	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250
Rock	431.840909	392.727273	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091
Steel	486.296296	442.851852	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259
Dark	507.387097	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290
Fire	360.942308	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.442308

In [224]:
```
df.groupby(['Type 1']).mean().sort_values('Defense' , ascending = False).head(1
```

Out[224]:

Type 1	Unnamed: 0	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Steel	486.296296	442.851852	65.222222	92.703704	126.370370	67.518519	80.629630	55.259259
Rock	431.840909	392.727273	65.363636	92.863636	100.795455	63.340909	75.477273	55.909091
Dragon	521.843750	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.031250
Ground	392.312500	356.281250	73.781250	95.750000	84.843750	56.468750	62.750000	63.906250
Ghost	536.281250	486.500000	64.437500	73.781250	81.187500	79.343750	76.468750	64.343750
Water	333.312500	303.089286	72.062500	74.151786	72.946429	74.812500	70.517857	65.964286
Ice	465.666667	423.541667	72.000000	72.750000	71.416667	77.541667	76.291667	63.458333
Grass	380.414286	344.871429	67.271429	73.214286	70.800000	77.500000	70.428571	61.928571
Bug	368.072464	334.492754	56.884058	70.971014	70.724638	53.869565	64.797101	61.681159
Dark	507.387097	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.161290

In [225]:
```
df.groupby(['Type 1']).mean().sort_values('Speed', ascending= False).head(7)
```

Out[225]:

Type 1	Unnamed: 0	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
Flying	746.500000	677.750000	70.750000	78.750000	66.250000	94.250000	72.500000	102.50000
Electric	400.590909	363.500000	59.795455	69.090909	66.295455	90.022727	73.704545	84.50000
Dragon	521.843750	474.375000	83.312500	112.125000	86.375000	96.843750	88.843750	83.03125
Psychic	420.105263	380.807018	70.631579	71.456140	67.684211	98.403509	86.280702	81.49122
Dark	507.387097	461.354839	66.806452	88.387097	70.225806	74.645161	69.516129	76.16129
Fire	360.942308	327.403846	69.903846	84.769231	67.769231	88.980769	72.211538	74.44230
Normal	351.581633	319.173469	77.275510	73.469388	59.846939	55.816327	63.724490	71.55102

In [226]:
```
df.sum()
```

Out[226]:
```
Unnamed: 0                                                    319600
#                                                             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 [227]:
```
df.groupby(['Type 2']).sum().head()
```

Out[227]:

Type 2	Unnamed: 0	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
Bug	1260	1146	160	270	240	140	185	185	10	
Dark	9112	8277	1511	2196	1441	1636	1397	1507	75	
Dragon	9578	8686	1479	1700	1567	1773	1502	1450	75	
Electric	3068	2794	529	436	410	487	441	429	24	
Fairy	9572	8718	1479	1417	1699	1725	1885	1408	82	

In [228]:
```
df.count()
```

Out[228]:
```
Unnamed: 0    800
#             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 [229]:
```
df['count1'] = 0
df.groupby(['Type 2']).count()['count1']
```

Out[229]:
```
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 [230]:
```
df['count1'] = 0
df.groupby(['Type 1']).count()['count1']
```

Out[230]:
```
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 [231]:
```
df['count1'] = 0
df.groupby(['Type 1','Type 2','Legendary']).count()['count1']
```

Out[231]:
```
Type 1  Type 2    Legendary
Bug     Electric  False         2
        Fighting  False         2
        Fire      False         2
        Flying    False        14
        Ghost     False         1
                                ..
Water   Ice       False         3
        Poison    False         3
        Psychic   False         5
        Rock      False         4
        Steel     False         1
Name: count1, Length: 150, dtype: int64
```

8 Loading Data in Chunks

In [232]:
```
for df in pd.read_csv('poke_updated1.csv', chunksize=3):
    print(df)
```

```
   #       Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  Sp. Def  Speed
\
0  1  Bulbasaur  Grass  Poison  45      49       49       65       65     45
1  2    Ivysaur  Grass  Poison  60      62       63       80       80     60
2  3   Venusaur  Grass  Poison  80      82       83      100      100     80

   Generation  Legendary  Total
0           1      False    318
1           1      False    405
2           1      False    525
   #                  Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  \
3  3  VenusaurMega Venusaur  Grass  Poison  80     100      123      122
4  4             Charmander   Fire     NaN  39      52       43       60
5  5             Charmeleon   Fire     NaN  58      64       58       80

   Sp. Def  Speed  Generation  Legendary  Total
3      120     80           1      False    625
4       50     65           1      False    309
5       65     80           1      False    405
```

In [233]:
```
df
```

Out[233]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Genera
798	720	HoopaHoopa Unbound	Psychic	Dark	80	160	60	170	130	80	
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	

In [234]:
```
df1 = pd.DataFrame()
for df in pd.read_csv('poke_updated1.csv',chunksize=10):
    df1 = pd.concat([df1,df])
df1.head(15)
```

Out[234]:

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

9 Stack & unstack in Pandas

In [235]:
```
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=cc
df6
```

Out[235]:

	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 [236]:
```
# Stack() Function stacks the columns to rows.
st_df = df6.stack()
st_df
```

Out[236]:

		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 [237]:
```
#Unstacks the row to columns
unst_df = st_df.unstack()
unst_df
```

Out[237]:

	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 [238]:
```
unst_df = unst_df.unstack()
unst_df
```

Out[238]:
```
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 [239]:
```
unst_df = unst_df.unstack()
unst_df
```

Out[239]:

		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

10 PIVOT Tables

In [240]:
```
data = {
'Country':['India','USA','Russia' , 'China','India','USA' , 'Russia','China','I
'Year':['2010','2010','2010' , '2010','2010','2010' , '2015','2015','2015' , '2
'Literacy/GDP':['GDP' , 'GDP' , 'GDP' , 'GDP','Literacy' , 'Literacy', 'Literac
'Value':[7,8,7,6,80,90,89,87,6,7]}
df7 = pd.DataFrame(data,columns=['Country','Year','Literacy/GDP','Value'])
df7
```

Out[240]:

	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	2015	Literacy	89
7	China	2015	Literacy	87
8	India	2015	GDP	6
9	USA	2015	GDP	7

In [241]:
```
pd.pivot_table(df7, index = ['Year', 'Literacy/GDP'], aggfunc='sum')
```

Out[241]:

		Value
Year	Literacy/GDP	
2010	GDP	28
	Literacy	170
2015	GDP	13
	Literacy	176

In [242]:
```
#Pivot table with MEAN aggregation
pd.pivot_table(df7 , index= ['Year' , 'Literacy/GDP'] , aggfunc='mean')
```

Out[242]:

		Value
Year	**Literacy/GDP**	
2010	GDP	7.0
	Literacy	85.0
2015	GDP	6.5
	Literacy	88.0

11 Hierarchical indexing

In [243]:
```
df7.head()
```

Out[243]:

	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

In [244]:
```
df8 = df7.set_index(['Year', 'Literacy/GDP'])
df8
```

Out[244]:

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

In [245]:
```
df8.index
```

Out[245]:
```
MultiIndex([('2010',      'GDP'),
            ('2010',      'GDP'),
            ('2010',      'GDP'),
            ('2010',      'GDP'),
            ('2010', 'Literacy'),
            ('2010', 'Literacy'),
            ('2015', 'Literacy'),
            ('2015', 'Literacy'),
            ('2015',      'GDP'),
            ('2015',      'GDP')],
           names=['Year', 'Literacy/GDP'])
```

In [246]:
```
df8.loc['2010']
```

Out[246]:

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

In [247]:
```
df8.loc[['2010']]
```

Out[247]:

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

In [248]:
```
df8.loc['2015','Literacy']
```

```
D:\Software_installed\python 3.8.3\lib\site-packages\ipykernel_launcher.py:1: P
erformanceWarning: indexing past lexsort depth may impact performance.
  """Entry point for launching an IPython kernel.
```

Out[248]:

			Country	Value
Year	**Literacy/GDP**			
2015		Literacy	Russia	89
		Literacy	China	87

In [249]:
```
df8 = df7.set_index(['Year', 'Literacy/GDP', 'Country'])
df8
```

Out[249]:

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

12 SWAP Columns in Hierarchical indexing

In [250]:
```
df7.head()
```

Out[250]:

	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

In [251]:
```
df8=df7.set_index(['Year', 'Literacy/GDP'])
df8
```

Out[251]:

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

In [252]:
```
# Swaping the columns in Hierarchical index
df9 = df8.swaplevel('Year','Literacy/GDP')
df9
```

Out[252]:

		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
	2015	Russia	89
	2015	China	87
GDP	2015	India	6
	2015	USA	7

In [253]:
```
# Swaping the columns in Hierarchical index
df9 = df9.swaplevel('Year', 'Literacy/GDP')
df9
```

Out[253]:

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

13 Crosstab in Pandas

In [254]:
```
df7.head()
```

Out[254]:

	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

In [255]:
```
pd.crosstab(df7['Literacy/GDP'],df7.Value, margins =True)
```

Out[255]:

Value	6	7	8	80	87	89	90	All
Literacy/GDP								
GDP	2	3	1	0	0	0	0	6
Literacy	0	0	0	1	1	1	1	4
All	2	3	1	1	1	1	1	10

In [256]:
```
# 2 way cross table
pd.crosstab(df7.Year, df7['Literacy/GDP'], margins = True)
```

Out[256]:

Literacy/GDP	GDP	Literacy	All
Year			
2010	4	2	6
2015	2	2	4
All	6	4	10

In [257]:
```
# 3 way cross table
pd.crosstab([df7.Year, df7['Literacy/GDP']],df7.Country, margins = True)
```

Out[257]:

	Country	China	India	Russia	USA	All
Year	**Literacy/GDP**					
2010	GDP	1	1	1	1	4
	Literacy	0	1	0	1	2
2015	GDP	0	1	0	1	2
	Literacy	1	0	1	0	2
All		2	3	2	3	10

14 Row & Column Bind

14.1 Row Bind

In [258]:
```
df8 = pd.DataFrame({'ID' :[1,2,3,4], 'Name':['Aryan' , 'Basitro' , 'Rose' , 'Jo
df8
```

Out[258]:

	ID	Name	Score
0	1	Aryan	99
1	2	Basitro	66
2	3	Rose	44
3	4	John	33

In [259]:
```
df9 = pd.DataFrame({'ID' :[5,6,7,8] , 'Name' :['Michelle' , 'Ramiro' , 'Vignesh
df9
```

Out[259]:

	ID	Name	Score
0	5	Michelle	78
1	6	Ramiro	55
2	7	Vignesh	77
3	8	Damon	87

In [260]:
```
# Row Bind with concat() function
pd.concat([df8,df9])
```

Out[260]:

	ID	Name	Score
0	1	Aryan	99
1	2	Basitro	66
2	3	Rose	44
3	4	John	33
0	5	Michelle	78
1	6	Ramiro	55
2	7	Vignesh	77
3	8	Damon	87

In [261]:
```
# Row Bind with append() function
df8.append(df9)
```

Out[261]:

	ID	Name	Score
0	1	Aryan	99
1	2	Basitro	66
2	3	Rose	44
3	4	John	33
0	5	Michelle	78
1	6	Ramiro	55
2	7	Vignesh	77
3	8	Damon	87

14.2 Column Bind

In [262]:
```
df10 = pd.DataFrame({'ID' :[1,2,3,4] , 'Name' :['Aryan' , 'Basitro' , 'Rose' ,
df10
```

Out[262]:

	ID	Name
0	1	Aryan
1	2	Basitro
2	3	Rose
3	4	John

In [263]:
```
df11 = pd.DataFrame({'Age' :[20,30,35,40] , 'Score' :[99 , 66 , 44 , 33]})
df11
```

Out[263]:

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

In [264]:
```
pd.concat([df10,df11] , axis = 1)
```

Out[264]:

	ID	Name	Age	Score
0	1	Aryan	20	99
1	2	Basitro	30	66
2	3	Rose	35	44
3	4	John	40	33