In [1]:
```
from string import maketrans
str="good"
str1="oo"
str2="ee"
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-1-703ecb8d01e2> in <module>
----> 1 from string import maketrans
      2 str="good"
      3 str1="oo"
      4 str2="ee"

ImportError: cannot import name 'maketrans' from 'string' (C:\Users\Preethi\A
naconda3\lib\string.py)
```

In [ ]:
```
txt = "Hi Sam!"
x = "mSa"
y = "eJo"
mytable = txt.maketrans(x, y)
print(txt.translate(mytable))
```

In [ ]:
```
str1="wy"
str2="gf"
str3="u"
txt="weeksyourweeks"
table=txt.maketrans(str1,str2,str3)
print("the string before translating is:", end= "")
print(txt)
print("the string after translating is:", end= "")
print(txt.translate(table))
```

In [ ]:
```
str="geeksforgeeks"
print("The minimum value chatacter is:" +min(str))
```

In [ ]:
```
print("The max value chatacter is:" +max(str))
```

In [ ]:
```
import sys
class Customer:
    '''customer class with bank operations'''
    bankname='Durgabank'

    def _init_(self,name,amt):

        self.amt=amt
        self.name=name
        self.balance=0

    def deposit(self,amt):
        print(self.amt)
        self.balance=self.balance+self.amt
        print('After deposit the balance:',self.balance)

    def withdraw(self,amt):
        print(self.balance)
        if self.amt>self.balance:
            print('Insufficent funds..cannot perform this operation')
            sys.exit()
        self.balance=self.balance-amt
        print('After withdraw the balance:',self.balance)
print('Welcome to', Customer.bankname)
name=input('Enter your Name:')
c=Customer()
while True:
    print('d-Deposit\nw-Withdraw\ne-exit')
    option=input('choose your option:')
    if option =='d'or option =='D':
        amt=float(input('Enter the amount to deposit:'))
        c.deposit(amt)

    elif option =='w'or option =='W':
        amt=float(input('Enter Amount to withdraw:'))
        c.withdraw(amt)

    elif option =='e'or option =='E':
        print('Thanks for Banking')
        sys.exit()
    else:
        print('Invalid option..plz choose valid option')
```

In [ ]:
```
import sys
class Customer:
    '''customer class with bank operations'''
    bankname='Durgabank'

    def __init__(self,name):
        self.name=name
        print(self.name)
        self.balance=0

    def deposit(self,amt):
        print(amt)
        self.balance=self.balance + amt
        print('After deposit the balance:',self.balance)

    def withdraw(self,amt):
        print(self.balance)
        if amt>self.balance:
            print('Insufficent funds..cannot perform this operation')
            sys.exit()
        self.balance=self.balance-amt
        print('After withdraw the balance:',self.balance)

print('Welcome to', Customer.bankname)
name=input('Enter your Name:')

c=Customer(name)


while True:
    print('d-Deposit\nw-Withdraw\ne-exit')
    option=input('choose your option:')
    if option =='d'or option =='D':
        amt=float(input('Enter the amount to deposit:'))
        c.deposit(amt)

    elif option =='w'or option =='W':
        amt=float(input('Enter Amount to withdraw:'))
        c.withdraw(amt)

    elif option =='e'or option =='E':
        print('Thanks for Banking')
        sys.exit()
    else:
        print('Invalid option..plz choose valid option')
```

NUMPY

In [ ]:
```
import numpy as np
a=np.array([1,2,3])
print(a)
```

In [ ]:
```
import numpy as np
a=np.array([(1,2,3),(4,5,6,)])
print(a)
```

In [ ]:
```
import numpy as np
import time
import sys

s=range(1000)

print(sys.getsizeof(999)*len(s))

d=np.arange(1000)

print(d.size*d.itemsize)
```

NUMPY VS LIST

In [ ]:
```
import numpy as np
import time
import sys

size=10000000

L1=range(size)
L2= range(size)

A1=np.arange(size)
A2=np.arange(size)

start =time.time()

#List
result=[(x,y) for x,y in zip(L1,L2)]
print((time.time()-start)*1000)

#Numpy
start = time.time()
result= A1+A2
print((time.time()-start)*1000)
```

NUMPY OPERATIONS

DIMENSION OF THE ARRAY

In [ ]:
```
import numpy as np

a=np.array([(1,2,3),(2,3,4)])
print(a.ndim)
```

In [ ]:
```
import numpy as np

a=np.array([1,2,3])
print(a.ndim)
```

BYTE SIZE OF EACH OF THE ITEM

In [ ]:
```
import numpy as np

a=np.array([1,2,3])
print(a.itemsize)
```

DATATYPE STORED IN THE ARRAY

In [ ]:
```
import numpy as np

a=np.array([1,2,3])
print(a.dtype)
```

FIND SIZE OF AN ARRRAY

In [ ]:
```
import numpy as np

a=np.array([1,2,3,4,5,6,7])
print(a.size)
```

In [ ]:
```
import numpy as np

a=np.array([1,2,3,4,5,6,7])
print(a.shape)
```

SHAPE OF AN ARRAY

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4,5,6,7),(8,9,10,11,12,13,14)])
print(a.shape)
```

RESHAPE AND SLICING OPERATOR

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4),(3,4,5,6)])
print(a)
a=a.reshape(4,2)
print(a)
```

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4),(3,4,5,6)])
print(a[0,1])
```

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4),(3,4,5,6)])
print(a[0:,3])
```

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4),(3,4,5,6),(7,8,9,10)])
print(a[0:,3])
```

In [ ]:
```
import numpy as np

a=np.array([(1,2,3,4),(3,4,5,6),(7,8,9,10)])
print(a[0:2,3])
```

In [ ]:
```
import numpy as np

a = np.linspace(1,3,5)

print(a)
```

In [ ]:
```
import numpy as np

a = np.linspace(1,3,10)

print(a)
```

MIN/MAX/SUM

In [ ]:
```
import numpy as np

a = np.array([1,3,10])

print(a.min())
print(a.max())
print(a.sum())
```

AXIS

In [ ]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])

print(a.sum(axis=0))
print(a.sum(axis=1))
```

FINDING SQUARE ROOT

In [ ]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])

print(np.sqrt(a))
```

STANDARD DEVIATION

In [2]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])

print(np.std(a))
```

```
1.2909944487358056
```

MATRIX ADDITON/SUBTRACTION/MULTIPLICATION AND DIVISION

In [3]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])
b = np.array([(1,2,3),(3,4,5)])
print(a+b)
```

```
[[ 2  4  6]
 [ 6  8 10]]
```

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

a = np.array([(1,2,3),(3,4,5)])
b = np.array([(1,2,3),(3,4,5)])
print(a-b)
```

```
[[0 0 0]
 [0 0 0]]
```

In [5]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])
b = np.array([(1,2,3),(3,4,5)])
print(a*b)
```

```
[[ 1  4  9]
 [ 9 16 25]]
```

In [6]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])
b = np.array([(1,2,3),(3,4,5)])
print(a/b)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

In [7]:
```
import numpy as np

a = np.array([(2,2,2),(3,4,5)])
b = np.array([(1,2,3),(3,4,5)])

print(np.vstack((a,b)))

print(np.hstack((a,b)))
```

```
[[2 2 2]
 [3 4 5]
 [1 2 3]
 [3 4 5]]
[[2 2 2 1 2 3]
 [3 4 5 3 4 5]]
```

In [8]:
```
import numpy as np

a = np.array([(1,2,3),(3,4,5)])
print(a.ravel())
```

```
[1 2 3 3 4 5]
```

NUMPY SPECIAL FUNCTIONS(COSINE AND SINE FUNCTION)

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

x=np.arange(0,3*np.pi,0.1)

y = np.sin(x)

plt.plot(x,y)

plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

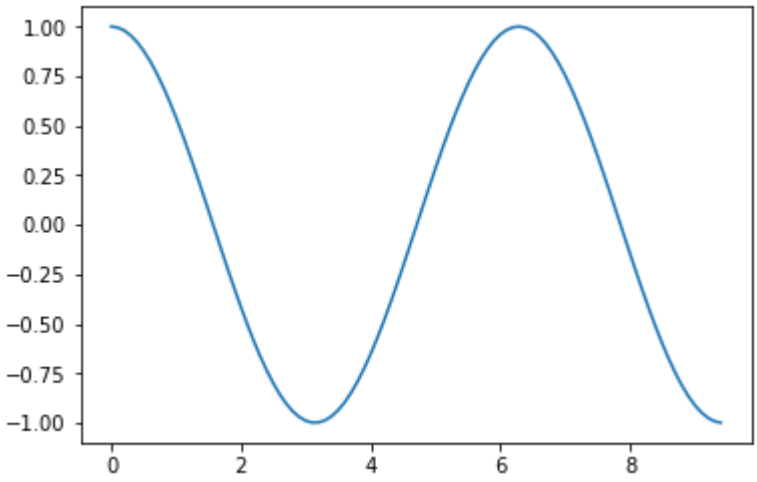
In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

x=np.arange(0,3*np.pi,0.1)

y = np.cos(x)

plt.plot(x,y)

plt.show()
```



EXPONENTIAL AND LOGARITHMIC FUNCTION

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt

ar=np.array([1,2,3])

print(np.exp(ar))
```

```
[ 2.71828183  7.3890561  20.08553692]
```

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt

ar=np.array([1,2,3])

print(np.log10(ar))
```

```
[0.         0.30103    0.47712125]
```

PANDAS PACKAGE - 9/3//2021

In [13]:
```
import pandas as pd
import numpy as np
```

In [14]:
```
s=pd.Series([1,2,3,4,5,6,np.nan,8,9,10])
```

In [15]:
```
s
```

Out[15]:
```
0     1.0
1     2.0
2     3.0
3     4.0
4     5.0
5     6.0
6     NaN
7     8.0
8     9.0
9    10.0
dtype: float64
```

In [16]:
```
d = pd.date_range('20210301',periods=10)
```

In [17]:
```
d
```

Out[17]:
```
DatetimeIndex(['2021-03-01', '2021-03-02', '2021-03-03', '2021-03-04',
               '2021-03-05', '2021-03-06', '2021-03-07', '2021-03-08',
               '2021-03-09', '2021-03-10'],
              dtype='datetime64[ns]', freq='D')
```

In [18]:
```
df= pd.DataFrame(np.random.randn(10,4),index=d,columns=['A','B','C','D'])
```

In [19]:
```
df
```

Out[19]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963
2021-03-03	-1.152579	0.417639	-1.055187	0.900075
2021-03-04	0.659322	1.073032	0.740063	-0.596763
2021-03-05	-1.137206	-0.292519	0.841183	1.113691
2021-03-06	2.012901	0.193830	1.117842	-1.847076
2021-03-07	1.036199	0.354417	0.512969	-0.178448
2021-03-08	-1.231841	-0.108971	-0.436260	1.513314
2021-03-09	-0.338333	-2.742900	-0.218041	1.054895
2021-03-10	-0.873903	-0.692282	0.505610	1.338890

In [20]:
```
df1 = pd.DataFrame({'A':[1,2,3,4],
                    'B':pd.Timestamp('20210301'),
                    'C':pd.Series(1,index=list(range(4)), dtype='float32'),
                    'D':np.array([5]*4, dtype='int32'),
                    'E':pd.Categorical(['true','false','True','False']),
                    'F':'Christuniversity'})
```

In [21]:
```
df1
```

Out[21]:

	A	B	C	D	E	F
0	1	2021-03-01	1.0	5	true	Christuniversity
1	2	2021-03-01	1.0	5	false	Christuniversity
2	3	2021-03-01	1.0	5	True	Christuniversity
3	4	2021-03-01	1.0	5	False	Christuniversity

In [22]:
```
df1.dtypes
```

Out[22]:
```
A             int64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

How to View Data?

Its very important to understand the various aspects to view the data in order to work on it

In [23]:
```
df.dtypes
```

Out[23]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

In [24]:
```
df.head()
```

Out[24]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963
2021-03-03	-1.152579	0.417639	-1.055187	0.900075
2021-03-04	0.659322	1.073032	0.740063	-0.596763
2021-03-05	-1.137206	-0.292519	0.841183	1.113691

In [25]:
```
df.tail()
```

Out[25]:

	A	B	C	D
2021-03-06	2.012901	0.193830	1.117842	-1.847076
2021-03-07	1.036199	0.354417	0.512969	-0.178448
2021-03-08	-1.231841	-0.108971	-0.436260	1.513314
2021-03-09	-0.338333	-2.742900	-0.218041	1.054895
2021-03-10	-0.873903	-0.692282	0.505610	1.338890

In [26]:
```
df.index
```

Out[26]:
```
DatetimeIndex(['2021-03-01', '2021-03-02', '2021-03-03', '2021-03-04',
               '2021-03-05', '2021-03-06', '2021-03-07', '2021-03-08',
               '2021-03-09', '2021-03-10'],
              dtype='datetime64[ns]', freq='D')
```

In [27]:
```
df.columns
```

Out[27]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [28]:
```
df.to_numpy()
```

Out[28]:
```
array([[ 3.16700329, -1.31961811, -1.84064933, -0.66644279],
       [-0.68837814, -0.4572313 ,  0.55066334, -0.33896346],
       [-1.15257898,  0.4176389 , -1.05518682,  0.90007501],
       [ 0.65932219,  1.0730318 ,  0.7400634 , -0.59676334],
       [-1.13720563, -0.29251871,  0.84118254,  1.11369074],
       [ 2.01290062,  0.1938304 ,  1.11784217, -1.84707646],
       [ 1.03619931,  0.35441715,  0.51296939, -0.17844768],
       [-1.23184113, -0.10897115, -0.4362605 ,  1.51331422],
       [-0.33833343, -2.7429001 , -0.21804054,  1.05489501],
       [-0.87390317, -0.6922822 ,  0.50561014,  1.33888967]])
```

In [29]:
```
df.describe()
```

Out[29]:

	A	B	C	D
count	10.000000	10.000000	10.000000	10.000000
mean	0.145318	-0.357460	0.071819	0.229317
std	1.522792	1.067805	0.943666	1.109496
min	-1.231841	-2.742900	-1.840649	-1.847076
25%	-1.071380	-0.633519	-0.381706	-0.532313
50%	-0.513356	-0.200745	0.509290	0.360814
75%	0.941980	0.314270	0.692713	1.098992
max	3.167003	1.073032	1.117842	1.513314

In [30]:
```
df.sort_index(axis=1,ascending=False)
```

Out[30]:

	D	C	B	A
2021-03-01	-0.666443	-1.840649	-1.319618	3.167003
2021-03-02	-0.338963	0.550663	-0.457231	-0.688378
2021-03-03	0.900075	-1.055187	0.417639	-1.152579
2021-03-04	-0.596763	0.740063	1.073032	0.659322
2021-03-05	1.113691	0.841183	-0.292519	-1.137206
2021-03-06	-1.847076	1.117842	0.193830	2.012901
2021-03-07	-0.178448	0.512969	0.354417	1.036199
2021-03-08	1.513314	-0.436260	-0.108971	-1.231841
2021-03-09	1.054895	-0.218041	-2.742900	-0.338333
2021-03-10	1.338890	0.505610	-0.692282	-0.873903

In [31]:
```
df.sort_values(by ='C')
```

Out[31]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-03	-1.152579	0.417639	-1.055187	0.900075
2021-03-08	-1.231841	-0.108971	-0.436260	1.513314
2021-03-09	-0.338333	-2.742900	-0.218041	1.054895
2021-03-10	-0.873903	-0.692282	0.505610	1.338890
2021-03-07	1.036199	0.354417	0.512969	-0.178448
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963
2021-03-04	0.659322	1.073032	0.740063	-0.596763
2021-03-05	-1.137206	-0.292519	0.841183	1.113691
2021-03-06	2.012901	0.193830	1.117842	-1.847076

In [32]:
```
df['C']
```

Out[32]:
```
2021-03-01   -1.840649
2021-03-02    0.550663
2021-03-03   -1.055187
2021-03-04    0.740063
2021-03-05    0.841183
2021-03-06    1.117842
2021-03-07    0.512969
2021-03-08   -0.436260
2021-03-09   -0.218041
2021-03-10    0.505610
Freq: D, Name: C, dtype: float64
```

In [33]:
```
df[0:3]
```

Out[33]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963
2021-03-03	-1.152579	0.417639	-1.055187	0.900075

In [34]:
```
 df.loc[d[0]]
```

Out[34]:
```
A    3.167003
B   -1.319618
C   -1.840649
D   -0.666443
Name: 2021-03-01 00:00:00, dtype: float64
```

In [35]:
```
df.loc[ :,['A','C']]
```

Out[35]:

	A	C
2021-03-01	3.167003	-1.840649
2021-03-02	-0.688378	0.550663
2021-03-03	-1.152579	-1.055187
2021-03-04	0.659322	0.740063
2021-03-05	-1.137206	0.841183
2021-03-06	2.012901	1.117842
2021-03-07	1.036199	0.512969
2021-03-08	-1.231841	-0.436260
2021-03-09	-0.338333	-0.218041
2021-03-10	-0.873903	0.505610

In [36]:
```
df.loc['20210301':'20210305',['A','C']]
```

Out[36]:

	A	C
2021-03-01	3.167003	-1.840649
2021-03-02	-0.688378	0.550663
2021-03-03	-1.152579	-1.055187
2021-03-04	0.659322	0.740063
2021-03-05	-1.137206	0.841183

In [37]:
```
df.loc['20210305',['A','C']]
```

Out[37]:
```
A   -1.137206
C    0.841183
Name: 2021-03-05 00:00:00, dtype: float64
```

In [38]:
```
df.loc[d[0],['D','C']]
```

Out[38]:
```
D   -0.666443
C   -1.840649
Name: 2021-03-01 00:00:00, dtype: float64
```

In [39]:
```
df.at[d[0],'C']
```

Out[39]:
```
-1.8406493340259262
```

In [40]:
```
df.iloc[3]
```

Out[40]:
```
A    0.659322
B    1.073032
C    0.740063
D   -0.596763
Name: 2021-03-04 00:00:00, dtype: float64
```

In [41]:
```
df.iloc[3:5]
```

Out[41]:

	A	B	C	D
2021-03-04	0.659322	1.073032	0.740063	-0.596763
2021-03-05	-1.137206	-0.292519	0.841183	1.113691

In [42]:
```
df.iloc[3:5,0:2]
```

Out[42]:

	A	B
2021-03-04	0.659322	1.073032
2021-03-05	-1.137206	-0.292519

df[df['A']>0]

In [43]:
```
df[df['A']> 2]
```

Out[43]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-06	2.012901	0.193830	1.117842	-1.847076

HANDLING THE MISSING DATA

How to handle Missing Data?

Missing data or null values in a data can create a lot of disoeder in another stage of data science life cycle.It is very important to deal withe missing data in an effective manner.

In [62]:
```
df2 = df.reindex(index=d[0:4],columns=list(df.columns)+['E'])
```

In [63]:
```
df2.loc[d[0]:d[1],'E']=1
```

In [64]:
```
df2
```

Out[64]:

	A	B	C	D	D	E
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443	-0.666443	1.0
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963	-0.338963	1.0
2021-03-03	-1.152579	0.417639	-1.055187	0.900075	0.900075	NaN
2021-03-04	0.659322	1.073032	0.740063	-0.596763	-0.596763	NaN

In [65]:
```
df2.isnull()
```

Out[65]:

	A	B	C	D	D	E
2021-03-01	False	False	False	False	False	False
2021-03-02	False	False	False	False	False	False
2021-03-03	False	False	False	False	False	True
2021-03-04	False	False	False	False	False	True

In [48]:
```
df2.isnull().count()
```

Out[48]:
```
A    4
B    4
C    4
D    4
E    4
dtype: int64
```

In [49]:
```
df2.dropna()
```

Out[49]:

	A	B	C	D	E
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443	1.0
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963	1.0

In [50]:
```
df2.isnull()
```

Out[50]:

	A	B	C	D	E
2021-03-01	False	False	False	False	False
2021-03-02	False	False	False	False	False
2021-03-03	False	False	False	False	True
2021-03-04	False	False	False	False	True

In [51]:
```
df2.fillna(value=2)
```

Out[51]:

	A	B	C	D	E
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443	1.0
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963	1.0
2021-03-03	-1.152579	0.417639	-1.055187	0.900075	2.0
2021-03-04	0.659322	1.073032	0.740063	-0.596763	2.0

In [52]:
```
pd.isna(df2)
```

Out[52]:

	A	B	C	D	E
2021-03-01	False	False	False	False	False
2021-03-02	False	False	False	False	False
2021-03-03	False	False	False	False	True
2021-03-04	False	False	False	False	True

In [53]:
```
df2.dropna()
```

Out[53]:

	A	B	C	D	E
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443	1.0
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963	1.0

PANDAS OPERATIONS

Different operations can help ecplore the data to reach conclusions.

In [54]:
```
df.mean()
```

Out[54]:
```
A    0.145318
B   -0.357460
C    0.071819
D    0.229317
dtype: float64
```

In [55]:
```
df.mean(1)
```

Out[55]:
```
2021-03-01   -0.164927
2021-03-02   -0.233477
2021-03-03   -0.222513
2021-03-04    0.468914
2021-03-05    0.131287
2021-03-06    0.369374
2021-03-07    0.431285
2021-03-08   -0.065940
2021-03-09   -0.561095
2021-03-10    0.069579
Freq: D, dtype: float64
```

In [56]:
```
s = pd.Series([1,2,3,np.nan,4,5,6,7,8,9,],index= d).shift(2)
```

In [57]:
```
s
```

Out[57]:
```
2021-03-01    NaN
2021-03-02    NaN
2021-03-03    1.0
2021-03-04    2.0
2021-03-05    3.0
2021-03-06    NaN
2021-03-07    4.0
2021-03-08    5.0
2021-03-09    6.0
2021-03-10    7.0
Freq: D, dtype: float64
```

In [58]:
```
df.sub(s, axis='index')
```

Out[58]:

	A	B	C	D
2021-03-01	NaN	NaN	NaN	NaN
2021-03-02	NaN	NaN	NaN	NaN
2021-03-03	-2.152579	-0.582361	-2.055187	-0.099925
2021-03-04	-1.340678	-0.926968	-1.259937	-2.596763
2021-03-05	-4.137206	-3.292519	-2.158817	-1.886309
2021-03-06	NaN	NaN	NaN	NaN
2021-03-07	-2.963801	-3.645583	-3.487031	-4.178448
2021-03-08	-6.231841	-5.108971	-5.436260	-3.486686
2021-03-09	-6.338333	-8.742900	-6.218041	-4.945105
2021-03-10	-7.873903	-7.692282	-6.494390	-5.661110

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

Out[59]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-02	-0.688378	-0.457231	0.550663	-0.338963
2021-03-03	-1.152579	0.417639	-1.055187	0.900075
2021-03-04	0.659322	1.073032	0.740063	-0.596763
2021-03-05	-1.137206	-0.292519	0.841183	1.113691
2021-03-06	2.012901	0.193830	1.117842	-1.847076
2021-03-07	1.036199	0.354417	0.512969	-0.178448
2021-03-08	-1.231841	-0.108971	-0.436260	1.513314
2021-03-09	-0.338333	-2.742900	-0.218041	1.054895
2021-03-10	-0.873903	-0.692282	0.505610	1.338890

In [60]:
```
df.apply(np.cumsum)
```

Out[60]:

	A	B	C	D
2021-03-01	3.167003	-1.319618	-1.840649	-0.666443
2021-03-02	2.478625	-1.776849	-1.289986	-1.005406
2021-03-03	1.326046	-1.359211	-2.345173	-0.105331
2021-03-04	1.985368	-0.286179	-1.605109	-0.702095
2021-03-05	0.848163	-0.578697	-0.763927	0.411596
2021-03-06	2.861063	-0.384867	0.353915	-1.435480
2021-03-07	3.897263	-0.030450	0.866885	-1.613928
2021-03-08	2.665422	-0.139421	0.430624	-0.100614
2021-03-09	2.327088	-2.882321	0.212584	0.954281
2021-03-10	1.453185	-3.574603	0.718194	2.293171

In [61]:
```
df.apply(lambda x: x.max() - x.min())
```

Out[61]:
```
A    4.398844
B    3.815932
C    2.958491
D    3.360391
dtype: float64
```

In [ ]:
```

```

Loading data into pandas

In [103]:
```
import pandas as pd

df3=pd.read_csv('pokemon_data.csv')

#print(df3.head(5))

#df _xlsx =pd.read_excel('pokemon_data.xlsx')
#print(df.xlsx.head(3)



#Read a specific function
#print(df3.iloc[2,1])


#for index, row in df3.iterrows():
    #print(index,row['Name'])

#df3.loc[df3['Type 1']== "Fire"]
```

Out[103]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generatio
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	
42	37	Vulpix	Fire	NaN	38	41	40	50	65	65	
43	38	Ninetales	Fire	NaN	73	76	75	81	100	100	
63	58	Growlithe	Fire	NaN	55	70	45	70	50	60	
64	59	Arcanine	Fire	NaN	90	110	80	100	80	95	
83	77	Ponyta	Fire	NaN	50	85	55	65	65	90	
84	78	Rapidash	Fire	NaN	65	100	70	80	80	105	
135	126	Magmar	Fire	NaN	65	95	57	100	85	93	
147	136	Flareon	Fire	NaN	65	130	60	95	110	65	
158	146	Moltres	Fire	Flying	90	100	90	125	85	90	
169	155	Cyndaquil	Fire	NaN	39	52	43	60	50	65	2
170	156	Quilava	Fire	NaN	58	64	58	80	65	80	2
171	157	Typhlosion	Fire	NaN	78	84	78	109	85	100	2
236	218	Slugma	Fire	NaN	40	40	40	70	40	20	2
237	219	Magcargo	Fire	Rock	50	50	120	80	80	30	2
259	240	Magby	Fire	NaN	45	75	37	70	55	83	2
263	244	Entei	Fire	NaN	115	115	85	90	75	100	2
270	250	Ho-oh	Fire	Flying	106	130	90	110	154	90	2
276	255	Torchic	Fire	NaN	45	60	40	70	50	45	3
277	256	Combusken	Fire	Fighting	60	85	60	85	60	55	3
278	257	Blaziken	Fire	Fighting	80	120	70	110	70	80	3
279	257	BlazikenMega Blaziken	Fire	Fighting	80	160	80	130	80	100	3
352	322	Numel	Fire	Ground	60	60	40	65	45	35	3
353	323	Camerupt	Fire	Ground	70	100	70	105	75	40	3
354	323	CameruptMega Camerupt	Fire	Ground	70	120	100	145	105	20	3
355	324	Torkoal	Fire	NaN	70	85	140	85	70	20	3
435	390	Chimchar	Fire	NaN	44	58	44	58	44	61	4
436	391	Monferno	Fire	Fighting	64	78	52	78	52	81	4
437	392	Infernape	Fire	Fighting	76	104	71	104	71	108	4

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generatio
518	467	Magmortar	Fire	NaN	75	95	67	125	95	83	
542	485	Heatran	Fire	Steel	91	90	106	130	106	77	
557	498	Tepig	Fire	NaN	65	63	45	45	45	45	
558	499	Pignite	Fire	Fighting	90	93	55	70	55	55	
559	500	Emboar	Fire	Fighting	110	123	65	100	65	65	
572	513	Pansear	Fire	NaN	50	53	48	53	48	64	
573	514	Simisear	Fire	NaN	75	98	63	98	63	101	
614	554	Darumaka	Fire	NaN	70	90	45	15	45	50	
615	555	DarmanitanStandard Mode	Fire	NaN	105	140	55	30	55	95	
616	555	DarmanitanZen Mode	Fire	Psychic	105	30	105	140	105	55	
692	631	Heatmor	Fire	NaN	85	97	66	105	66	65	
721	653	Fennekin	Fire	NaN	40	45	40	62	60	60	
722	654	Braixen	Fire	NaN	59	59	58	90	70	73	
723	655	Delphox	Fire	Psychic	75	69	72	114	100	104	
730	662	Fletchinder	Fire	Flying	62	73	55	56	52	84	
731	663	Talonflame	Fire	Flying	78	81	71	74	69	126	
735	667	Litleo	Fire	Normal	62	50	58	73	54	72	
736	668	Pyroar	Fire	Normal	86	68	72	109	66	106	
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	

Reading Data in Pandas

In [96]:
```
#Read Headers
print(df3.columns)

#Read each Column
#print(df3['Name'][0:5])
#print(df3.Name)
#print(df3['Name'][0:5])
#print(df3[['Name', 'Type 1', 'HP']])

#print each row
#print(df3.head(4))

print(df3.iloc[1:4])
```

```
Index(['#', 'Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Sp. Atk',
       'Sp. Def', 'Speed', 'Generation', 'Legendary'],
      dtype='object')
   #                   Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  \
1  2                Ivysaur  Grass  Poison  60      62       63       80
2  3              Venusaur  Grass  Poison  80      82       83      100
3  3  VenusaurMega Venusaur  Grass  Poison  80     100      123      122

   Sp. Def  Speed  Generation  Legendary
1       80     60           1      False
2      100     80           1      False
3      120     80           1      False
```

Sorting and Describing the data

In [104]:
```
df3.describe()
```

Out[104]:

	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
count	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	80
mean	362.813750	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	
std	208.343798	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	
min	1.000000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	
25%	184.750000	50.000000	55.000000	50.000000	49.750000	50.000000	45.000000	
50%	364.500000	65.000000	75.000000	70.000000	65.000000	70.000000	65.000000	
75%	539.250000	80.000000	100.000000	90.000000	95.000000	90.000000	90.000000	
max	721.000000	255.000000	190.000000	230.000000	194.000000	230.000000	180.000000	

In [116]:
```
df3.sort_values(['Name'],ascending=False)
```

Out[116]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
794	718	Zygarde50% Forme	Dragon	Ground	108	100	121	81	95	95	6
695	634	Zweilous	Dark	Dragon	72	85	70	65	70	58	5
46	41	Zubat	Poison	Flying	40	45	35	30	40	55	1
631	570	Zorua	Dark	NaN	40	65	40	80	40	65	5
632	571	Zoroark	Dark	NaN	60	105	60	120	60	105	5
286	263	Zigzagoon	Normal	NaN	38	30	41	30	41	60	3
707	644	Zekrom	Dragon	Electric	100	150	120	120	100	90	5
582	523	Zebstrika	Electric	NaN	75	100	63	80	63	116	5
157	145	Zapdos	Electric	Flying	90	90	85	125	90	100	1
367	335	Zangoose	Normal	NaN	73	115	60	60	60	90	3
793	717	Yveltal	Dark	Flying	126	131	95	131	98	99	6
520	469	Yanmega	Bug	Flying	86	76	86	116	56	95	4
208	193	Yanma	Bug	Flying	65	65	45	75	45	95	2
623	562	Yamask	Ghost	NaN	38	30	85	55	65	30	5
792	716	Xerneas	Fairy	NaN	126	131	95	131	98	99	6
192	178	Xatu	Psychic	Flying	65	75	70	95	70	95	2
394	360	Wynaut	Psychic	NaN	95	23	48	23	48	23	3
288	265	Wurmple	Bug	NaN	45	45	35	20	30	20	3
460	413	WormadamTrash Cloak	Bug	Steel	60	69	95	69	95	36	4
459	413	WormadamSandy Cloak	Bug	Ground	60	79	105	59	85	36	4
458	413	WormadamPlant Cloak	Bug	Grass	60	59	85	79	105	36	4
209	194	Wooper	Water	Ground	55	45	45	25	25	15	2
586	527	Woobat	Psychic	Flying	55	45	43	55	43	72	5
217	202	Wobbuffet	Psychic	NaN	190	33	58	33	58	33	2
301	278	Wingull	Water	Flying	40	30	30	55	30	85	3
45	40	Wigglytuff	Normal	Fairy	140	70	45	85	50	45	1
317	293	Whismur	Normal	NaN	64	51	23	51	23	28	3
372	340	Whiscash	Water	Ground	110	78	73	76	71	60	3
604	544	Whirlipede	Bug	Poison	40	55	99	40	79	47	5
607	547	Whimsicott	Grass	Fairy	60	67	85	77	75	116	5
...	...	...	...	...	...	...	...	...	...	...	...
182	168	Ariados	Bug	Poison	70	90	70	60	60	40	2
628	567	Archeops	Rock	Flying	75	140	65	112	65	110	5

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
627	566	Archen	Rock	Flying	55	112	45	74	45	70	5
552	493	Arceus	Normal	NaN	120	120	120	120	120	120	4
64	59	Arcanine	Fire	NaN	90	110	80	100	80	95	1
29	24	Arbok	Poison	NaN	60	85	69	65	79	80	1
379	347	Anorith	Rock	Bug	45	95	50	40	50	75	3
196	181	AmpharosMega Ampharos	Electric	Dragon	90	95	105	165	110	45	2
195	181	Ampharos	Electric	NaN	90	75	85	115	90	55	2
652	591	Amoonguss	Grass	Poison	114	85	70	85	80	30	5
471	424	Ambipom	Normal	NaN	75	100	66	60	66	115	4
768	698	Amaura	Rock	Ice	77	59	50	67	63	46	6
366	334	AltariaMega Altaria	Dragon	Fairy	75	110	110	110	105	80	3
365	334	Altaria	Dragon	Flying	75	70	90	70	105	80	3
655	594	Alomomola	Water	NaN	165	75	80	40	45	65	5
71	65	AlakazamMega Alakazam	Psychic	NaN	55	50	65	175	95	150	1
70	65	Alakazam	Psychic	NaN	55	50	45	135	95	120	1
205	190	Aipom	Normal	NaN	55	70	55	40	55	85	2
333	306	AggronMega Aggron	Steel	NaN	70	140	230	60	80	50	3
332	306	Aggron	Steel	Rock	70	110	180	60	60	50	3
154	142	AerodactylMega Aerodactyl	Rock	Flying	80	135	85	70	95	150	1
153	142	Aerodactyl	Rock	Flying	80	105	65	60	75	130	1
751	681	AegislashShield Forme	Steel	Ghost	60	50	150	50	150	60	6
750	681	AegislashBlade Forme	Steel	Ghost	60	150	50	150	50	60	6
678	617	Accelgor	Bug	NaN	80	70	40	100	60	145	5
393	359	AbsolMega Absol	Dark	NaN	65	150	60	115	60	115	3
392	359	Absol	Dark	NaN	65	130	60	75	60	75	3
68	63	Abra	Psychic	NaN	25	20	15	105	55	90	1
511	460	AbomasnowMega Abomasnow	Grass	Ice	90	132	105	132	105	30	4
510	460	Abomasnow	Grass	Ice	90	92	75	92	85	60	4

800 rows × 12 columns

In [117]:
```
df3.sort_values(['Type 1','HP'],ascending=[1,0])
```

Out[117]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
520	469	Yanmega	Bug	Flying	86	76	86	116	56	95	4	
698	637	Volcarona	Bug	Fire	85	60	65	135	105	100	5	
231	214	Heracross	Bug	Fighting	80	125	75	40	95	85	2	
232	214	HeracrossMega Heracross	Bug	Fighting	80	185	115	40	105	75	2	
678	617	Accelgor	Bug	NaN	80	70	40	100	60	145	5	
734	666	Vivillon	Bug	Flying	80	52	50	90	50	89	6	
447	402	Kricketune	Bug	NaN	77	85	51	55	51	65	4	
220	205	Forretress	Bug	Steel	75	90	140	60	60	40	2	
602	542	Leavanny	Bug	Grass	75	103	80	70	80	92	5	
717	649	Genesect	Bug	Steel	71	120	95	120	95	99	5	
54	49	Venomoth	Bug	Poison	70	65	60	90	75	90	1	
132	123	Scyther	Bug	Flying	70	110	80	55	80	105	1	
182	168	Ariados	Bug	Poison	70	90	70	60	60	40	2	
228	212	Scizor	Bug	Steel	70	130	100	55	80	65	2	
229	212	ScizorMega Scizor	Bug	Steel	70	150	140	65	100	75	2	
308	284	Masquerain	Bug	Flying	70	60	62	80	82	60	3	
461	414	Mothim	Bug	Flying	70	94	50	94	50	66	4	
463	416	Vespiquen	Bug	Flying	70	80	102	80	102	40	4	
619	558	Crustle	Bug	Rock	70	95	125	65	75	45	5	
650	589	Escavalier	Bug	Steel	70	135	105	60	105	20	5	
657	596	Galvantula	Bug	Electric	70	77	60	97	60	108	5	
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	
136	127	Pinsir	Bug	NaN	65	125	100	55	70	85	1	
137	127	PinsirMega Pinsir	Bug	Flying	65	155	120	65	90	105	1	
208	193	Yanma	Bug	Flying	65	65	45	75	45	95	2	
342	313	Volbeat	Bug	NaN	65	73	55	47	75	85	3	
343	314	Illumise	Bug	NaN	65	47	55	73	75	85	3	
315	291	Ninjask	Bug	Flying	61	90	45	50	50	160	3	
15	12	Butterfree	Bug	Flying	60	45	50	90	80	70	1	
...	...	...	...	...	...	...	...	...	...	...	...	
725	657	Frogadier	Water	NaN	54	63	52	83	56	97	6	
438	393	Piplup	Water	NaN	53	51	53	61	56	40	4	

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	L
59	54	Psyduck	Water	NaN	50	52	48	65	50	55	1	
98	91	Cloyster	Water	Ice	50	95	180	85	45	70	1	
172	158	Totodile	Water	NaN	50	65	64	44	48	43	2	
280	258	Mudkip	Water	NaN	50	70	50	50	50	40	3	
371	339	Barboach	Water	Ground	50	48	43	46	41	60	3	
574	515	Panpour	Water	NaN	50	53	48	53	48	64	5	
595	535	Tympole	Water	NaN	50	50	40	50	40	64	5	
762	692	Clauncher	Water	NaN	50	53	62	58	63	44	6	
506	456	Finneon	Water	NaN	49	49	56	49	61	66	4	
127	118	Goldeen	Water	NaN	45	67	60	35	50	63	1	
347	318	Carvanha	Water	Dark	45	90	20	65	20	65	3	
508	458	Mantyke	Water	Flying	45	20	50	60	120	50	4	
9	7	Squirtle	Water	NaN	44	48	65	50	64	43	1	
373	341	Corphish	Water	NaN	43	80	65	50	35	35	3	
405	370	Luvdisc	Water	NaN	43	30	55	40	65	97	3	
724	656	Froakie	Water	NaN	41	56	40	62	44	71	6	
65	60	Poliwag	Water	NaN	40	50	40	40	40	90	1	
78	72	Tentacool	Water	Poison	40	40	35	50	100	70	1	
293	270	Lotad	Water	Grass	40	30	30	40	50	30	3	
301	278	Wingull	Water	Flying	40	30	30	55	30	85	3	
241	223	Remoraid	Water	NaN	35	65	35	65	35	65	2	
401	366	Clamperl	Water	NaN	35	64	85	74	55	32	3	
97	90	Shellder	Water	NaN	30	65	100	45	25	40	1	
106	98	Krabby	Water	NaN	30	105	90	25	25	50	1	
125	116	Horsea	Water	NaN	30	40	70	70	25	60	1	
129	120	Staryu	Water	NaN	30	45	55	70	55	85	1	
139	129	Magikarp	Water	NaN	20	10	55	15	20	80	1	
381	349	Feebas	Water	NaN	20	15	20	10	55	80	3	

800 rows × 12 columns

MAking changes to the data

In [119]:
```
df3.head(5)
```

Out[119]:

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

In [141]:
```
df3['Total']= df3['HP']+ df3['Attack']+ df3['Defense'] + df3['Sp. Atk']+ df3[
'Sp. Def']+ df3['Speed']

#df3=df3.drop(columns=['Total'])

df['Total'] = df.iloc[:, 4:10].sum(axis=1)


df3.head(5)
```

Out[141]:

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

Saving our data (Exporting into Desired Format)

In [143]:
```
df3
```

Out[143]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generati
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	
15	12	Butterfree	Bug	Flying	60	45	50	90	80	70	
16	13	Weedle	Bug	Poison	40	35	30	20	20	50	
17	14	Kakuna	Bug	Poison	45	25	50	25	25	35	
18	15	Beedrill	Bug	Poison	65	90	40	45	80	75	
19	15	BeedrillMega Beedrill	Bug	Poison	65	150	40	15	80	145	
20	16	Pidgey	Normal	Flying	40	45	40	35	35	56	
21	17	Pidgeotto	Normal	Flying	63	60	55	50	50	71	
22	18	Pidgeot	Normal	Flying	83	80	75	70	70	101	
23	18	PidgeotMega Pidgeot	Normal	Flying	83	80	80	135	80	121	
24	19	Rattata	Normal	NaN	30	56	35	25	35	72	
25	20	Raticate	Normal	NaN	55	81	60	50	70	97	
26	21	Spearow	Normal	Flying	40	60	30	31	31	70	
27	22	Fearow	Normal	Flying	65	90	65	61	61	100	
28	23	Ekans	Poison	NaN	35	60	44	40	54	55	
29	24	Arbok	Poison	NaN	60	85	69	65	79	80	
...	...	...	...	...	...	...	...	...	...	...	...
770	700	Sylveon	Fairy	NaN	95	65	65	110	130	60	

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generati
771	701	Hawlucha	Fighting	Flying	78	92	75	74	63	118	
772	702	Dedenne	Electric	Fairy	67	58	57	81	67	101	
773	703	Carbink	Rock	Fairy	50	50	150	50	150	50	
774	704	Goomy	Dragon	NaN	45	50	35	55	75	40	
775	705	Sliggoo	Dragon	NaN	68	75	53	83	113	60	
776	706	Goodra	Dragon	NaN	90	100	70	110	150	80	
777	707	Klefki	Steel	Fairy	57	80	91	80	87	75	
778	708	Phantump	Ghost	Grass	43	70	48	50	60	38	
779	709	Trevenant	Ghost	Grass	85	110	76	65	82	56	
780	710	PumpkabooAverage Size	Ghost	Grass	49	66	70	44	55	51	
781	710	PumpkabooSmall Size	Ghost	Grass	44	66	70	44	55	56	
782	710	PumpkabooLarge Size	Ghost	Grass	54	66	70	44	55	46	
783	710	PumpkabooSuper Size	Ghost	Grass	59	66	70	44	55	41	
784	711	GourgeistAverage Size	Ghost	Grass	65	90	122	58	75	84	
785	711	GourgeistSmall Size	Ghost	Grass	55	85	122	58	75	99	
786	711	GourgeistLarge Size	Ghost	Grass	75	95	122	58	75	69	
787	711	GourgeistSuper Size	Ghost	Grass	85	100	122	58	75	54	
788	712	Bergmite	Ice	NaN	55	69	85	32	35	28	
789	713	Avalugg	Ice	NaN	95	117	184	44	46	28	
790	714	Noibat	Flying	Dragon	40	30	35	45	40	55	
791	715	Noivern	Flying	Dragon	85	70	80	97	80	123	
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	
795	719	Diancie	Rock	Fairy	50	100	150	100	150	50	
796	719	DiancieMega Diancie	Rock	Fairy	50	160	110	160	110	110	
797	720	HoopaHoopa Confined	Psychic	Ghost	80	110	60	150	130	70	
798	720	HoopaHoopa Unbound	Psychic	Dark	80	160	60	170	130	80	
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	

800 rows × 13 columns

In [151]:
```
#df3.to_csv('modified.csv',index=False)

#df3.to_excel('modified.xlsx',index = False)

df3.to_csv('modified1.csv',index=False,sep = '\t')
```

Filtering Data

In [169]:
```
new_df3 = df3.loc[(df3['Type 1'] == 'Grass') & (df3['Type 2'] == 'Poison') & (
df3['HP'] > 70)]

new_df3 = new_df3.reset_index(drop=True)
new_df3
```

Out[169]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Lege
0	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	
1	3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	
2	45	Vileplume	Grass	Poison	75	80	85	110	90	50	1	
3	71	Victreebel	Grass	Poison	80	105	65	100	70	70	1	
4	591	Amoonguss	Grass	Poison	114	85	70	85	80	30	5	

In [177]:
```
#df3.loc[df3['Name'].str.contains('Mega')]

import re
df3.loc[df3['Type 1'].str.contains('Fire|Grass', regex=True)]
```

Out[177]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generatic
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	
42	37	Vulpix	Fire	NaN	38	41	40	50	65	65	
43	38	Ninetales	Fire	NaN	73	76	75	81	100	100	
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	
63	58	Growlithe	Fire	NaN	55	70	45	70	50	60	
64	59	Arcanine	Fire	NaN	90	110	80	100	80	95	
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	
83	77	Ponyta	Fire	NaN	50	85	55	65	65	90	
84	78	Rapidash	Fire	NaN	65	100	70	80	80	105	
110	102	Exeggcute	Grass	Psychic	60	40	80	60	45	40	
111	103	Exeggutor	Grass	Psychic	95	95	85	125	65	55	
122	114	Tangela	Grass	NaN	65	55	115	100	40	60	
135	126	Magmar	Fire	NaN	65	95	57	100	85	93	
147	136	Flareon	Fire	NaN	65	130	60	95	110	65	
158	146	Moltres	Fire	Flying	90	100	90	125	85	90	
166	152	Chikorita	Grass	NaN	45	49	65	49	65	45	
167	153	Bayleef	Grass	NaN	60	62	80	63	80	60	
168	154	Meganium	Grass	NaN	80	82	100	83	100	80	
...	...	...	...	...	...	...	...	...	...	...	...
571	512	Simisage	Grass	NaN	75	98	63	98	63	101	
572	513	Pansear	Fire	NaN	50	53	48	53	48	64	

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generatic
573	514	Simisear	Fire	NaN	75	98	63	98	63	101	
606	546	Cottonee	Grass	Fairy	40	27	60	37	50	66	
607	547	Whimsicott	Grass	Fairy	60	67	85	77	75	116	
608	548	Petilil	Grass	NaN	45	35	50	70	50	30	
609	549	Lilligant	Grass	NaN	70	60	75	110	75	90	
614	554	Darumaka	Fire	NaN	70	90	45	15	45	50	
615	555	DarmanitanStandard Mode	Fire	NaN	105	140	55	30	55	95	
616	555	DarmanitanZen Mode	Fire	Psychic	105	30	105	140	105	55	
617	556	Maractus	Grass	NaN	75	86	67	106	67	60	
651	590	Foongus	Grass	Poison	69	55	45	55	55	15	
652	591	Amoonguss	Grass	Poison	114	85	70	85	80	30	
658	597	Ferroseed	Grass	Steel	44	50	91	24	86	10	
659	598	Ferrothorn	Grass	Steel	74	94	131	54	116	20	
692	631	Heatmor	Fire	NaN	85	97	66	105	66	65	
701	640	Virizion	Grass	Fighting	91	90	72	90	129	108	
718	650	Chespin	Grass	NaN	56	61	65	48	45	38	
719	651	Quilladin	Grass	NaN	61	78	95	56	58	57	
720	652	Chesnaught	Grass	Fighting	88	107	122	74	75	64	
721	653	Fennekin	Fire	NaN	40	45	40	62	60	60	
722	654	Braixen	Fire	NaN	59	59	58	90	70	73	
723	655	Delphox	Fire	Psychic	75	69	72	114	100	104	
730	662	Fletchinder	Fire	Flying	62	73	55	56	52	84	
731	663	Talonflame	Fire	Flying	78	81	71	74	69	126	
735	667	Litleo	Fire	Normal	62	50	58	73	54	72	
736	668	Pyroar	Fire	Normal	86	68	72	109	66	106	
740	672	Skiddo	Grass	NaN	66	65	48	62	57	52	
741	673	Gogoat	Grass	NaN	123	100	62	97	81	68	
799	721	Volcanion	Fire	Water	80	110	120	130	90	70	

122 rows × 13 columns

Aggregate Statistics(Groupby)

In [187]:
```
df3 =pd.read_csv('modified.csv')

#df3.groupby(['Type 1']).mean().sort_values('Defense', ascending=False)

#df3.groupby(['Type 1']).sum()

#df3.groupby(['Type 1']).count()

df3['count'] = 1

df3.groupby(['Type 1', 'Type 2']).count()['count']
```

Out[187]:

Type 1	Type 2	
Bug	Electric	2
	Fighting	2
	Fire	2
	Flying	14
	Ghost	1
	Grass	6
	Ground	2
	Poison	12
	Rock	3
	Steel	7
	Water	1
Dark	Dragon	3
	Fighting	2
	Fire	3
	Flying	5
	Ghost	2
	Ice	2
	Psychic	2
	Steel	2
Dragon	Electric	1
	Fairy	1
	Fire	1
	Flying	6
	Ground	5
	Ice	3
	Psychic	4
Electric	Dragon	1
	Fairy	1
	Fire	1
	Flying	5
		..
Rock	Fighting	1
	Flying	4
	Grass	2
	Ground	6
	Ice	2
	Psychic	2
	Steel	3
	Water	6
Steel	Dragon	1
	Fairy	3
	Fighting	1
	Flying	1
	Ghost	4
	Ground	2
	Psychic	7
	Rock	3
Water	Dark	6
	Dragon	2
	Electric	2
	Fairy	2
	Fighting	3
	Flying	7
	Ghost	2
	Grass	3
	Ground	10

```
Ice          3
Poison       3
Psychic      5
Rock         4
Steel        1
Name: count, Length: 136, dtype: int64
```

Handling Missing Value(Filling up NA Values)

In [188]:
```
import pandas as pd
import numpy as np
```

In [192]:
```
null_values= pd.Series([1,np.NaN,2,np.NaN,3,4,np.NaN,np.NaN,8])
```

In [193]:
```
null_values
```

Out[193]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    3.0
5    4.0
6    NaN
7    NaN
8    8.0
dtype: float64
```

In [194]:
```
null_values.fillna(method='ffill')
```

Out[194]:
```
0    1.0
1    1.0
2    2.0
3    2.0
4    3.0
5    4.0
6    4.0
7    4.0
8    8.0
dtype: float64
```

In [195]:
```
null_values.fillna(method='bfill')
```

Out[195]:
```
0    1.0
1    2.0
2    2.0
3    3.0
4    3.0
5    4.0
6    8.0
7    8.0
8    8.0
dtype: float64
```

In [196]:
```
null_values.fillna(0)
```

Out[196]:
```
0    1.0
1    0.0
2    2.0
3    0.0
4    3.0
5    4.0
6    0.0
7    0.0
8    8.0
dtype: float64
```

In [199]:
```
null_values.fillna(np.mean(null_values))
```

Out[199]:
```
0    1.0
1    3.6
2    2.0
3    3.6
4    3.0
5    4.0
6    3.6
7    3.6
8    8.0
dtype: float64
```

In [207]:
```
null_values.fillna(method='ffill',limit = 1)
```

Out[207]:
```
0    1.0
1    1.0
2    2.0
3    2.0
4    3.0
5    4.0
6    4.0
7    NaN
8    8.0
dtype: float64
```

In [201]:
```
null_values.interpolate()
```

Out[201]:
```
0    1.000000
1    1.500000
2    2.000000
3    2.500000
4    3.000000
5    4.000000
6    5.333333
7    6.666667
8    8.000000
dtype: float64
```

In [204]:
```
interpolate_series= pd.Series([1,np.NaN,5,np.NaN,9,np.NaN,13])
```

In [205]:
```
interpolate_series.interpolate()
```

Out[205]:
```
0     1.0
1     3.0
2     5.0
3     7.0
4     9.0
5    11.0
6    13.0
dtype: float64
```

In [ ]:
```

```