Statistical Functions in Python

In [25]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
import statistics as stat
```

Central Tendency

In [28]:
```
n= np.random.randint(7,10,20)
print(n)
```

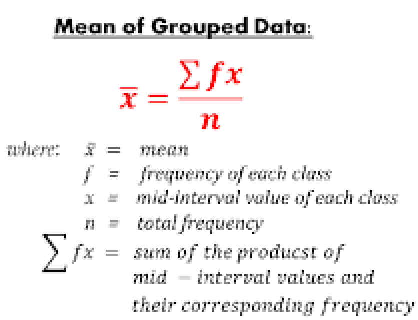
```
[8 7 9 8 9 8 7 8 7 7 7 7 7 9 8 7 7 8 9 8]
```

Mean Function

Mean is the sum of the value of each observations in data set divided by the number of observations

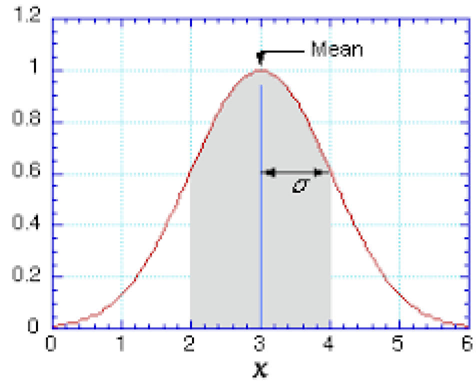
In [42]:
```
from IPython.display import Image
Image(filename='E:/pyimages/mean0.png')
```

Out[42]:



In [37]:
```
from IPython.display import Image
Image(filename='E:/pyimages/mean.png')
```

Out[37]:



In [38]:
```
np.mean(n)
```

Out[38]: 7.75

In [3]:
```
stat.mean([1,2,3,4])
```

Out[3]: 2.5

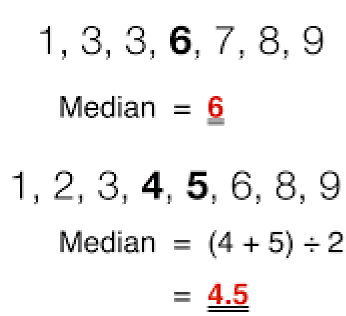
In [4]:
```
stat.mean([1,2,3,4,50])
```

Out[4]: 12

Median

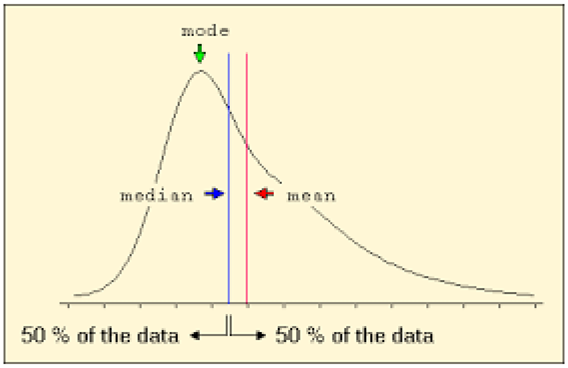
In [46]:
```
from IPython.display import Image
Image(filename='E:/pyimages/median0.png')
```

Out[46]:



In [47]:
```
from IPython.display import Image
Image(filename='E:/pyimages/median.png')
```

Out[47]:



In [48]:
```
np.median(n)
```

Out[48]: 8.0

In [5]:
```
stat.median({1,2,3,4,50})
```

Out[5]: 3

In [6]:
```
stat.median([1,3,50,4,2])
```

Out[6]: 3

In [7]:
```
stat.median([1,2,30,50,60,55,40,50]) # in this case finds avg of two middle nu
mber
```

Out[7]: 45.0

In [8]:
```
stat.median_low([1,3,50,4,2])
```

Out[8]: 3

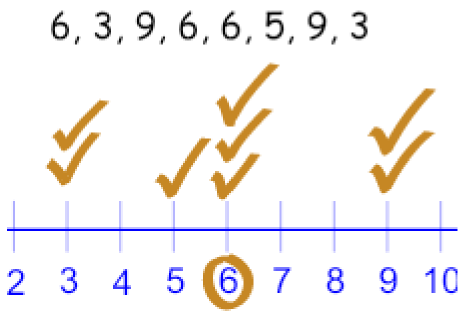
In [9]:
```
stat.median_high([1,6,3,50,4,2])
```

Out[9]: 4

Mode

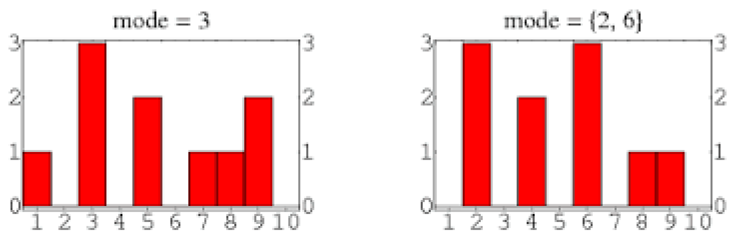
In [49]:
```
from IPython.display import Image
Image(filename='E:/pyimages/mode0.png')
```

Out[49]:



In [50]:
```
from IPython.display import Image
Image(filename='E:/pyimages/mode.png')
```

Out[50]:



In [10]:
```
stat.mode([1,2,1,3,3,4,4,4,5])
```

Out[10]: 4

In [11]:
```
stat.mode([1,2,3,4])
```

```
---------------------------------------------------------------------------
StatisticsError                           Traceback (most recent call last)
<ipython-input-11-b9b6425c424e> in <module>
----> 1 stat.mode([1,2,3,4])

C:\anaconda3\lib\statistics.py in mode(data)
    504     elif table:
    505         raise StatisticsError(
--> 506                 'no unique mode; found %d equally common values' % le
n(table)
    507                 )
    508     else:

StatisticsError: no unique mode; found 4 equally common values
```

Harmonic Mean

In [ ]:
```
stat.harmonic_mean([1,2,3,4])
```

Standard Deviation Function

[1,2,3,4,5] # consicutive difference is 1 no lot of disperssion among number

[1,2,500,600,-2] # dispession varies alot in number

In [21]:
```
stat.stdev([1,2,3])
```
Out[21]:
```
1.0
```

In [22]:
```
stat.stdev([1,2,500,600,-2])
```
Out[22]:
```
303.13726263856114
```

In [23]:
```
stat.variance([1,2,3,4,5,600])
```
Out[23]:
```
59403.5
```

Creation of Population dataset

In [56]:
```
population = np.random.randint(10,20,100)
population
```
Out[56]:
```
array([18, 11, 12, 18, 15, 13, 19, 18, 11, 19, 10, 19, 16, 11, 13, 15, 15,
       11, 14, 16, 18, 11, 19, 16, 19, 17, 13, 17, 12, 19, 11, 12, 17, 17,
       14, 14, 17, 19, 17, 16, 13, 10, 19, 14, 16, 16, 12, 18, 18, 16, 13,
       15, 14, 12, 19, 16, 14, 17, 18, 19, 10, 19, 11, 11, 15, 19, 11, 17,
       12, 15, 16, 16, 19, 19, 15, 10, 16, 11, 14, 18, 11, 15, 17, 14, 13,
       10, 10, 14, 11, 14, 12, 11, 12, 12, 14, 18, 12, 13, 14, 13])
```

In [57]:
```
np.mean(population)
```
Out[57]:
```
14.72
```

In [59]:
```
np.median(population)
```
Out[59]:
```
15.0
```

In [61]:
```
from statistics import mode
mode(population)
```
Out[61]:
```
19
```

In [62]:
```
sample=np.random.choice(population,20)
sample
```
Out[62]:
```
array([15, 13, 11, 16, 12, 17, 16, 17, 13, 19, 15, 10, 16, 19, 10, 12, 14,
       14, 11, 12])
```

In [63]:
```
np.mean(sample)
```
Out[63]: 14.1

In [64]:
```
np.median(sample)
```
Out[64]: 14.0

In [68]:
```
sample1=np.random.choice(population,10)
sample1
```
Out[68]: array([19, 11, 17, 14, 19, 18, 19, 19, 14, 11])

In [69]:
```
np.mean(sample1)
```
Out[69]: 16.1

In [71]:
```
np.median(sample1)
```
Out[71]: 17.5

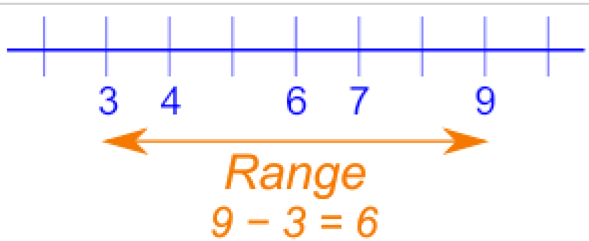
In [72]:
```
sample=np.random.choice(population,20)
sample1=np.random.choice(population,15)
sample2=np.random.choice(population,10)
sample3=np.random.choice(population,5)
```

In [75]:
```
all_samples=[sample,sample1,sample2,sample3]
sample_mean=[]
for i in all_samples:
    sample_mean.append(np.mean(1))
sample_mean
```
Out[75]: [1.0, 1.0, 1.0, 1.0]

In [76]:
```
np.mean(sample_mean)
```
Out[76]: 1.0

Range

In [2]:
```
from IPython.display import Image
Image(filename='E:/pyimages/range.png')
```
Out[2]:



In [7]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

In [6]:
```
n=np.random.randn(9)
n
```

Out[6]:
```
array([ 0.64835067, -0.39925531,  0.64320203,  0.24283355,  0.78986282,
       -0.96110071,  0.99804129,  0.22722202, -0.77016285])
```

In [8]:
```
# Range:

np.max(n)-np.min(n)
```

Out[8]:
```
1.959141997817723
```

In [9]:
```
n=np.random.randn(30)
n
```

Out[9]:
```
array([ 0.30660136,  0.18955531, -0.41204737,  0.4298422 ,  0.63080927,
       -1.64107963,  0.2620718 , -0.4127817 , -1.82698773,  0.0291693 ,
        0.69688716, -0.30101659, -0.7317028 ,  0.61273998,  0.411687  ,
        0.14236308, -0.90975577, -1.4865773 ,  0.30066822,  0.38880768,
       -1.48479099, -1.36498011, -0.8690589 ,  0.46676093,  0.88125612,
        1.64162044,  0.39106158,  1.39715284,  0.17389935,  0.30019409])
```

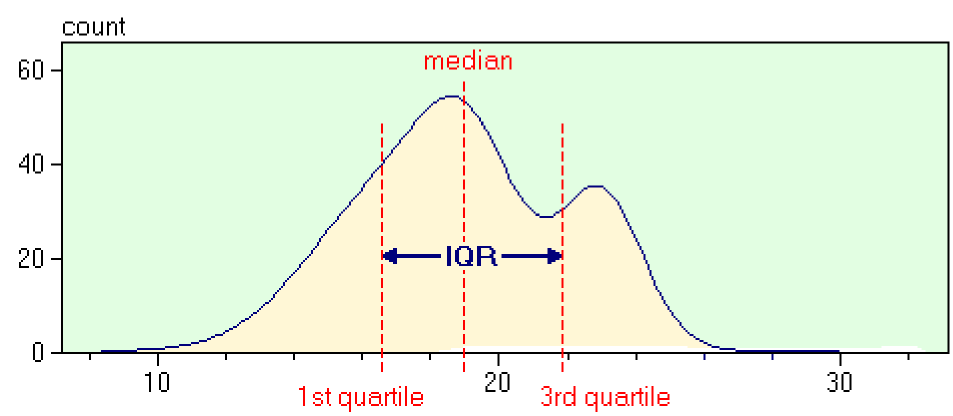
In [10]:
```
np.max(n)-np.min(n)
```

Out[10]:
```
3.4686081689554884
```

Interquatile Range

In [12]:
```
from IPython.display import Image
Image(filename='E:/pyimages/interquartile.png')
```

Out[12]:



In [14]:
```
# First Quartile

q1=np.percentile(n,75)
```

In [ ]:
```
# Second Quartile

q2=np.percentile(n,50)
```

In [15]:
```
# Third Quartile

q3=np.percentile(n,75)
```

Interquartile range

IQR= q3-q1

In [19]:
```
IQR= q3-q1
```

In [20]:
```
IQR
```

Out[20]:
```
0.0
```

variance

In [25]:
```
from IPython.display import Image
Image(filename='E:/pyimages/meanvariance.png')
```

Out[25]:

	Population	**Sample**
# of subjects	N	n
Mean	$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$	$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$
Variance	$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}$	$S^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$

Note: S^2 is the formula for unbiased sample variance, since we're dividing by $n-1$.

Standard deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}}$	$S = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$

Note: Finding S by taking $\sqrt{S^2}$ reintroduces bias.

In [23]:
```
population = np.random.randn(100)
population
```

Out[23]:
```
array([ 1.13364281,  0.20813156, -0.49039347,  0.47607173,  1.57715744,
        1.14729139, -0.51960049,  1.13747426, -1.28885488,  1.85517791,
        0.11698179, -0.86450702, -1.06594138, -1.79239126,  0.20973238,
        0.05388449,  0.58854362,  1.55263027,  0.68690154, -0.90941053,
       -1.88125526,  1.77251892, -0.36496575,  0.76970442, -0.38617487,
        0.99580853,  0.87559861,  0.91986788,  1.46623713, -1.41431437,
        0.52688283, -0.22368626, -0.32982899, -0.82980222,  0.14615214,
       -0.41513706, -0.13472317, -0.47941308, -0.29690663, -0.34841352,
        0.12709846,  0.09934264, -1.56349637, -2.39193288, -0.69650291,
        0.33057995,  0.54481002, -1.07170811, -0.24867409,  0.13860025,
        0.1149794 ,  0.75312523,  0.01454702,  1.15784119,  0.41790696,
        1.69612267,  1.935609  ,  0.3724933 , -1.26240029, -1.1671613 ,
       -0.59061535,  0.74263151,  0.97325808, -0.14645279, -0.51579546,
       -1.26982565, -1.03846874,  0.9101527 , -2.07570431, -2.06042914,
       -0.53444461,  0.62360466,  1.2091826 ,  0.44100514, -0.37055556,
       -1.27923513,  0.77171813, -0.47292255, -0.76284864,  0.84026693,
        0.64853318, -1.88610845,  0.71826119, -0.33981284,  1.09014468,
       -0.6925029 , -0.72124843, -1.79760064, -0.27191069, -1.12522377,
        0.74962558,  1.00079606,  0.29802669,  1.9686116 ,  0.23098387,
        0.39512942,  0.80799811, -0.18477792, -0.55540335,  0.23820809])
```

In [24]:
```
np.var(population)
```

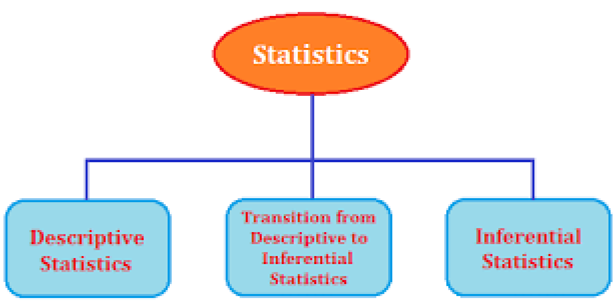
Out[24]:
```
0.9826764141230292
```

In [26]:
```
np.std(population)
```

Out[26]:
```
0.9913003652390274
```

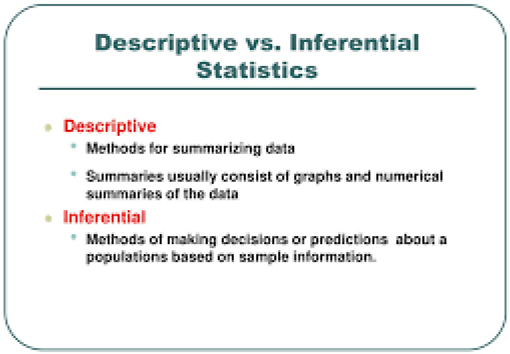
In [84]:
```
from IPython.display import Image
Image(filename='E:/pyimages/typesstat.png')
```

Out[84]:



In [86]:
```
from IPython.display import Image
Image(filename='E:/pyimages/typestat1.png')
```

Out[86]:



Descriptive statistics:

Deals with presentation of collections of data

In [36]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [41]:
```
df1 =pd.DataFrame(dict(id=range(6), age=np.random.randint(18,31,size=6)))

df1
```

Out[41]:

	id	age
0	0	27
1	1	25
2	2	20
3	3	27
4	4	18
5	5	19

In [46]:
```
df1.age.mean()

df1.mean()
```

Out[46]:
```
id       2.500000
age     22.666667
dtype: float64
```

In [68]:
```
df1["age"].mean()
```

Out[68]:
```
22.666666666666668
```

In [69]:
```
df1.age.median()
```
Out[69]:
```
22.5
```

In [70]:
```
df1.age.mode()
```
Out[70]:
```
0    27
dtype: int32
```

In [71]:
```
df1.age.var()
```
Out[71]:
```
17.066666666666667
```

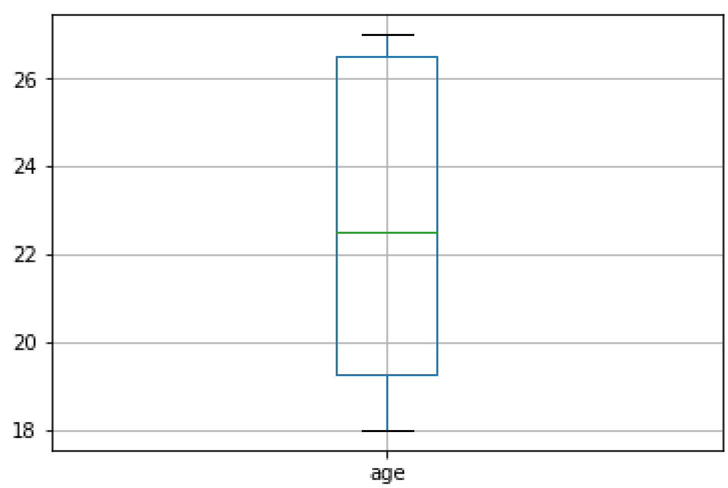
In [72]:
```
df1.age.std()
```
Out[72]:
```
4.1311822359545785
```

In [73]:
```
df1.age.max()
```
Out[73]:
```
27
```

In [74]:
```
df1.age.min()
```
Out[74]:
```
18
```

In [75]:
```
# Range
df1.age.max()-df1.age.min()
```
Out[75]:
```
9
```

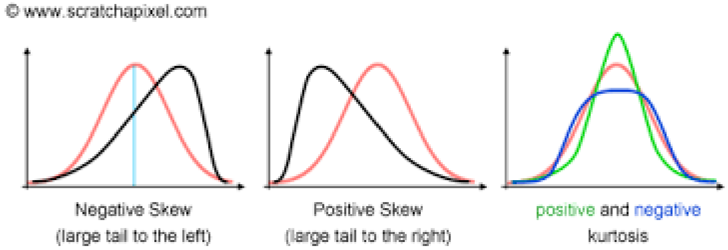
In [76]:
```
df1.boxplot(column='age',return_type='axes')
```
Out[76]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f1f367e7f0>
```



Skewness and Kurtosis

In [77]:
```
from IPython.display import Image
Image(filename='E:/pyimages/skew.png')
```

Out[77]:



In [80]:
```
df1["age"].skew()
```

Out[80]: 0.023638814368940428

In [81]:
```
df1["age"].kurt()
```

Out[81]: -2.788238525390625

Inferential Statistics

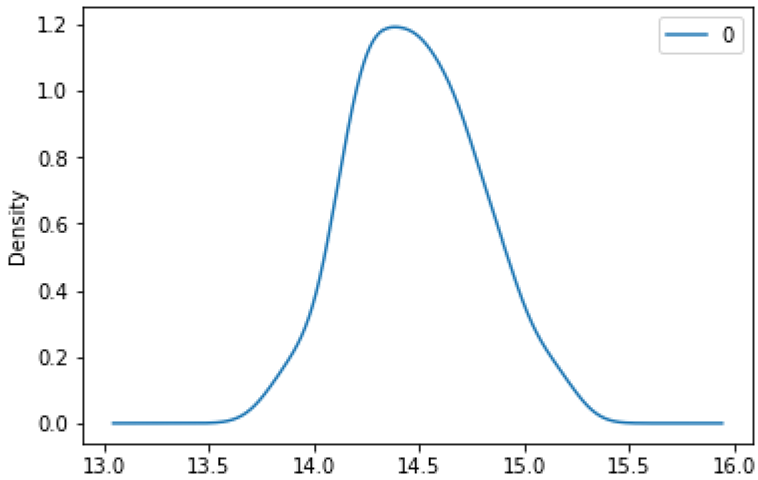
In [88]:
```
pop=np.random.randint(10,20,1000)

np.random.seed(10)

estimates =[]  # make empty list to hold point estimates

for x in range(200):  # genearte 200 samples,each with 500 sampled values
    sample = np.random.choice(a=pop,size=100)
    estimates.append(sample.mean()) # keep the sample mean in list
```

In [89]:
```
np.mean(pop)
```

Out[89]: 14.491

In [90]:
```
pd.DataFrame(estimates).plot(kind="density")
```

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x1f1f3a3a898>

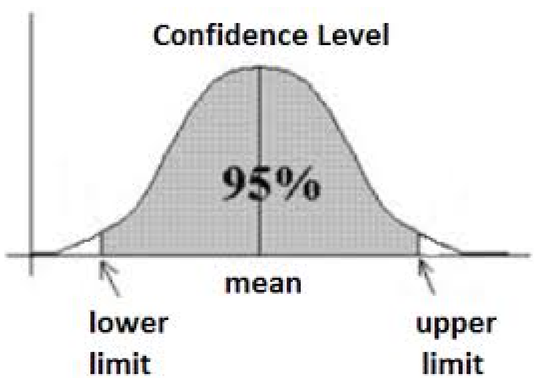


Point of Estimations

Confidence Interval

In [96]:
```
from IPython.display import Image
Image (filename=("E:/pyimages/cf.png"))
```

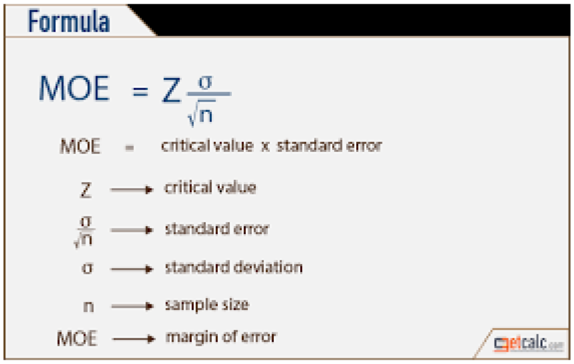
Out[96]:



Margin of error

In [97]:
```
from IPython.display import Image
Image (filename=("E:/pyimages/mirro.png"))
```

Out[97]:



In [99]:
```
# to find Critical value

import scipy.stats as stats

z_critical = stats.norm.ppf(q=0.975) ##percentage point function
```

In [100]:
```
t_critical = stats.t.ppf(q=0.975,df=24) ## df is degree of freedom (sample six
e minus 1)
```

In [101]:
```
margin_of_error = z_critical = (np.std(estimates)/np.sqrt(200))
```

In [103]:
```
## Lower : sample_mean_of_erroe

np.mean(estimates)- margin_of_error
```

Out[103]: 14.464465800050748

In [104]:
```
## upper : sample_mean_of_erroe

np.mean(estimates) + margin_of_error
```

Out[104]: 14.506634199949252

In [ ]:
```

```