Normal distribution

Normal Distribution (1/2)

- Most widely known and used of all distributions
- It is asymptotic to the horizontal axis. That is, it does not touch the x-axis and it goes on forever in each direction.

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\begin{aligned} \mu &= \text{ Mean} \\ \sigma &= \text{ Standard Deviation} \\ \pi &\approx 3.14159\cdots \\ e &\approx 2.71828\cdots \end{aligned}$$

Normal Distribution (2/2)

- It is unimodal. The normal curve is sometimes called a bell-shaped curve. All the values are "bunched up" in only one portion of the graph – the center of the curve.
- The area under the curve is 1. The area under the curve yields the probabilities
- The area of the distribution on each side of the mean is 0.5.

Normal Distribution in Python using SciPy package

scipy.stats.norm¶

scipy.stats.norm(**args, **kwds*) = *<scipy.stats._continuous_distns.norm_gen object>* [source]

A normal continuous random variable.

The location (`loc`) keyword specifies the mean. The scale (`scale`) keyword specifies the standard deviation.

As an instance of the rv_continuous class, norm object inherits from it a collection of generic methods (see below for the full list), and completes them with details specific for this particular distribution.

Notes

The probability density function for norm is:

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}}$$

for a real number x.

The probability density above is defined in the "standardized" form. To shift and/or scale the distribution use the `loc` and `scale` parameters. Specifically, `norm.pdf(x, loc, scale)` is identically equivalent to `norm.pdf(y) / scale` with `y = (x - loc) / scale`.

Methods available in binom module

Methods

rvs(loc=0, scale=1, size=1, random_state=None)	Random variates.
pdf(x, loc=0, scale=1)	Probability density function.
logpdf(x, loc=0, scale=1)	Log of the probability density function.
cdf(x, loc=0, scale=1)	Cumulative distribution function.
logcdf(x, loc=0, scale=1)	Log of the cumulative distribution function.
sf(x, loc=0, scale=1)	Survival function (also defined as `1 - cdf`, but *sf* is sometimes more accurate).
logsf(x, loc=0, scale=1)	Log of the survival function.
ppf(q, loc=0, scale=1)	Percent point function (inverse of `cdf` — percentiles).
isf(q, loc=0, scale=1)	Inverse survival function (inverse of `sf`).
moment(n, loc=0, scale=1)	Non-central moment of order n
stats(loc=0, scale=1, moments='mv')	Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
entropy(loc=0, scale=1)	(Differential) entropy of the RV.
fit(data, loc=0, scale=1)	Parameter estimates for generic data.
expect(func, args=(), loc=0, scale=1, lb=None, ub=None, conditional=False, **kwds)	Expected value of a function (of one argument) with respect to the distribution.
median(loc=0, scale=1)	Median of the distribution.
mean(loc=0, scale=1)	Mean of the distribution.
var(loc=0, scale=1)	Variance of the distribution.
std(loc=0, scale=1)	Standard deviation of the distribution.
interval(alpha, loc=0, scale=1)	Endpoints of the range that contains alpha percent of the distribution

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html

Poisson Distribution in Python

- Generating Number from Normal Distribution

```
from scipy.stats import norm
```

```
data=norm.rvs(size=1000, loc=2, scale=3)
data[:10]
```

```
array([ 6.36529124,  6.57837006,  7.85780416, -3.30033448,  1.32422146,
        0.96595595,  5.89286383,  4.14575392,  3.36006633,  4.7619259 ])
```

```
data.mean()
```

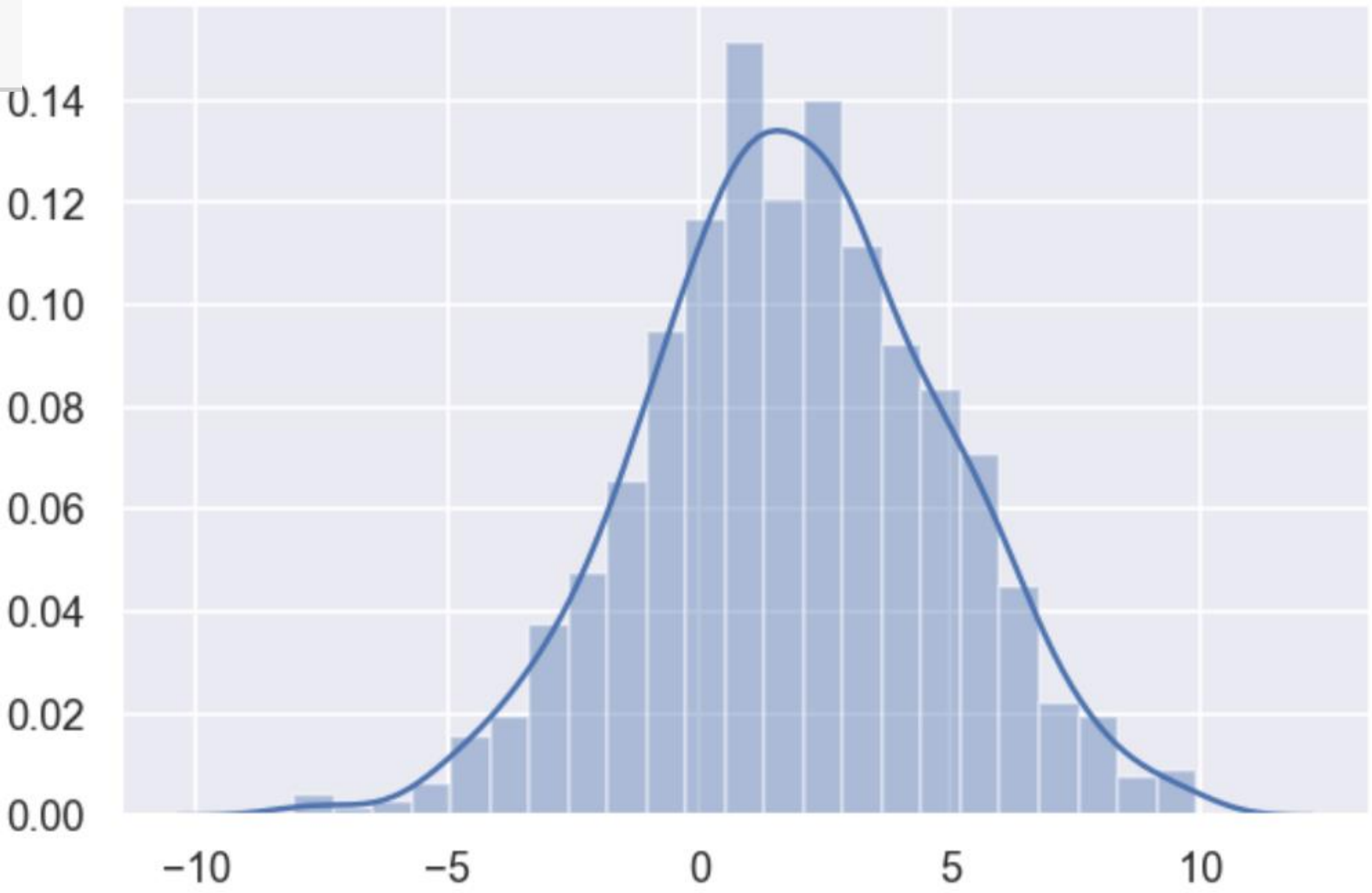
```
1.8668609300032817
```

```
data.std()
```

```
2.9450729999231617
```

Plotting the Poisson Distribution

```
plt.figure(dpi=120)
sns.distplot(data)
plt.show()
```



Estimation of CDF and its inverse

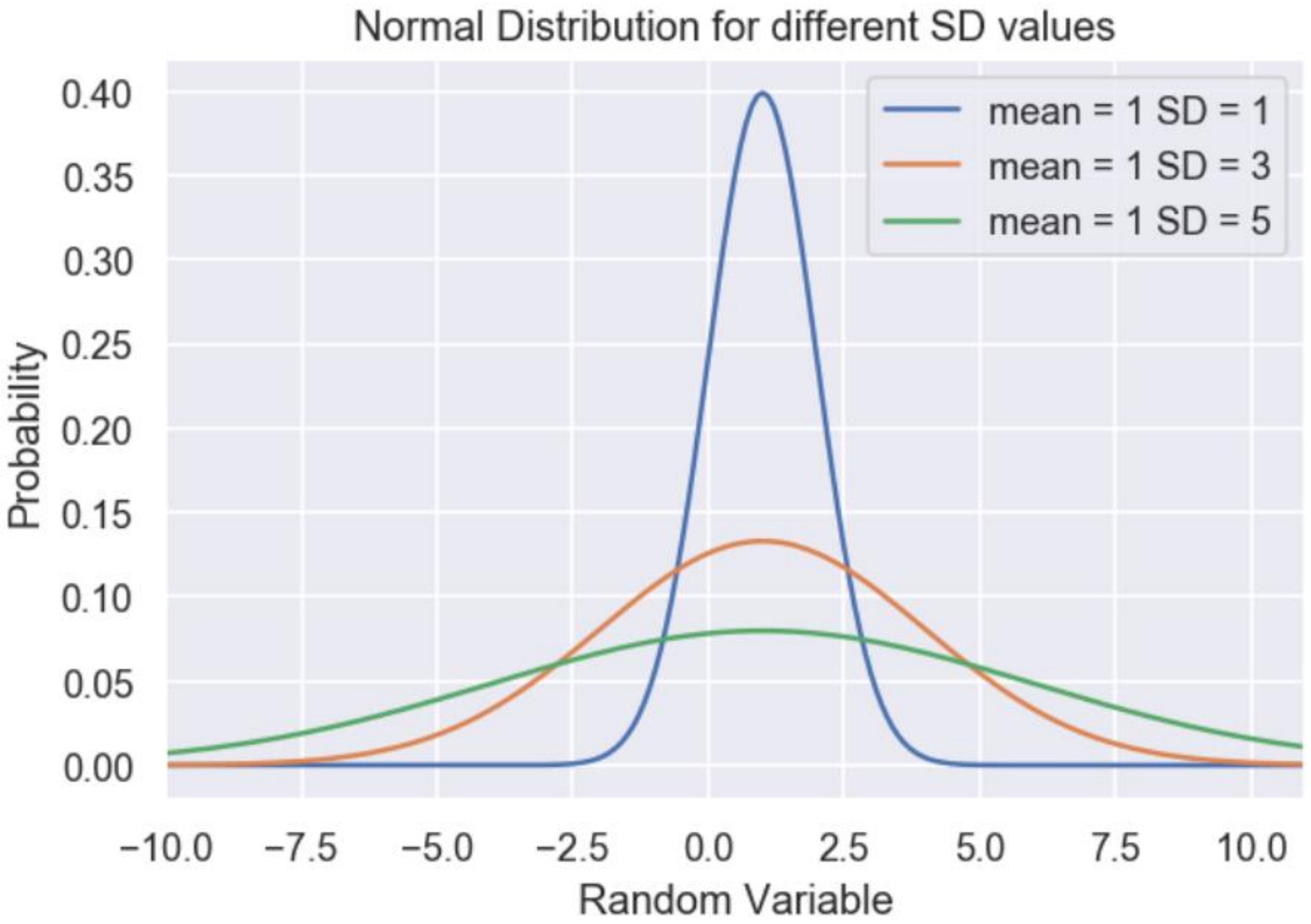
```
norm.cdf(x=5,loc=2,scale=3)
```

0.8413447460685429

```
norm.ppf(q=0.84,loc=2,scale=3)
```

4.983373649629259

Normal Distribution for different Means and SD



Calculation of Skewness and Kurtosis

For a standard normal distribution

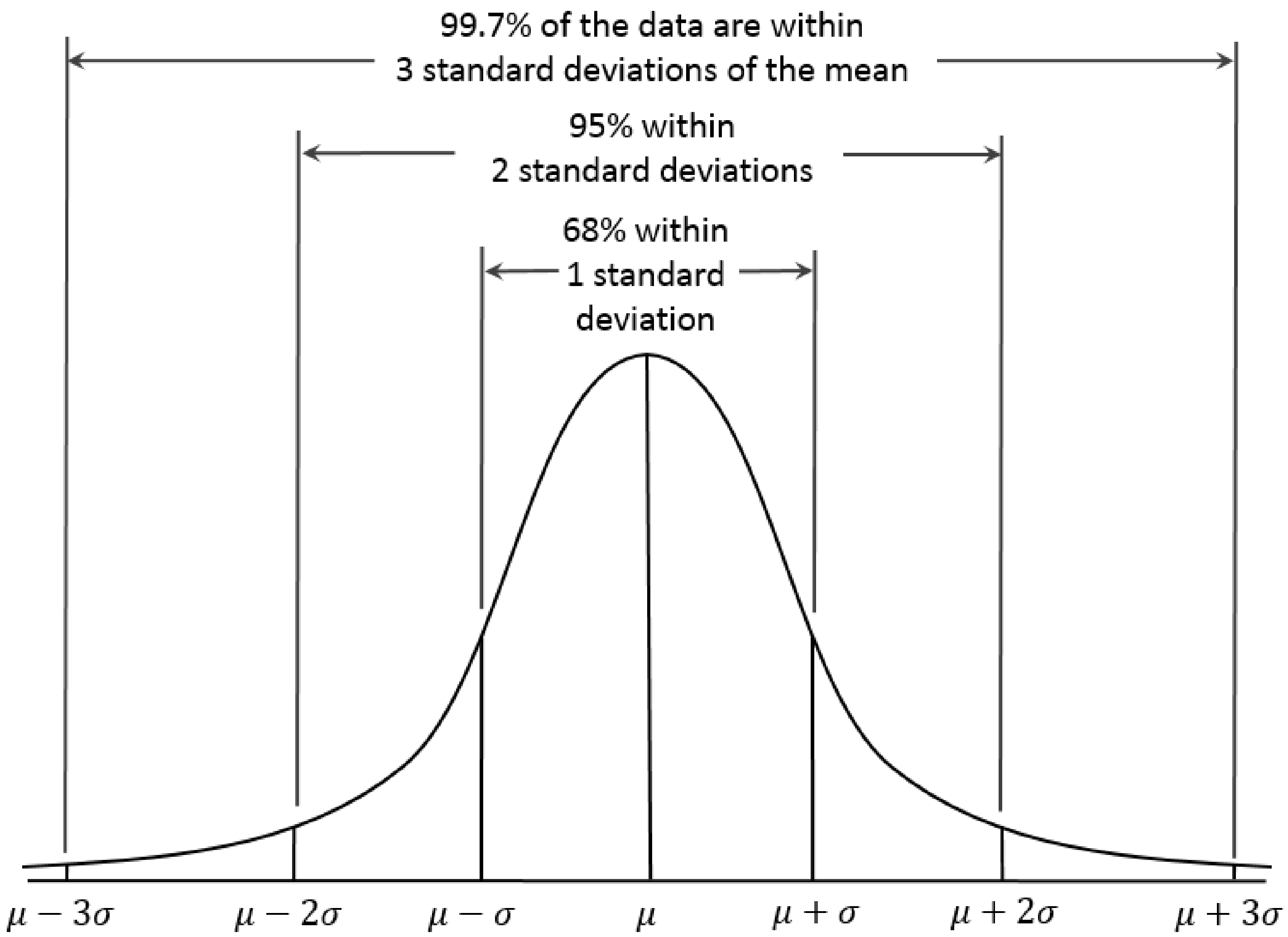
- Skewness is 0
- Kurtosis is 3

[Note: in Scipy package it is corrected to 0 as per Fisher's definition]

```
from scipy.stats import skew,kurtosis
print("Skewness",skew(data))
print("kurtosis",kurtosis(data))
```

```
Skewness -0.05999410614444226
kurtosis 0.012500309715253177
```

3 Sigma Rule


99.7% of the data are within
3 standard deviations of the mean
95% within
2 standard deviations
68% within
1 standard
deviation
$\mu - 3\sigma$
$\mu - 2\sigma$
$\mu - \sigma$
μ
$\mu + \sigma$
$\mu + 2\sigma$
$\mu + 3\sigma$

Checking 3 Sigma Rule in Python

```
mean=0
sd=1
```

```
# 1 sigma
norm.cdf(mean+sd,mean,sd)-norm.cdf(mean-sd,mean,sd)
```

```
0.6826894921370859
```

```
# 2 sigma
norm.cdf(mean+2*sd,mean,sd)-norm.cdf(mean-2*sd,mean,sd)
```

```
0.9544997361036416
```

```
# 3 sigma
norm.cdf(mean+3*sd,mean,sd)-norm.cdf(mean-3*sd,mean,sd)
```

```
0.9973002039367398
```

Central Limit Theorem

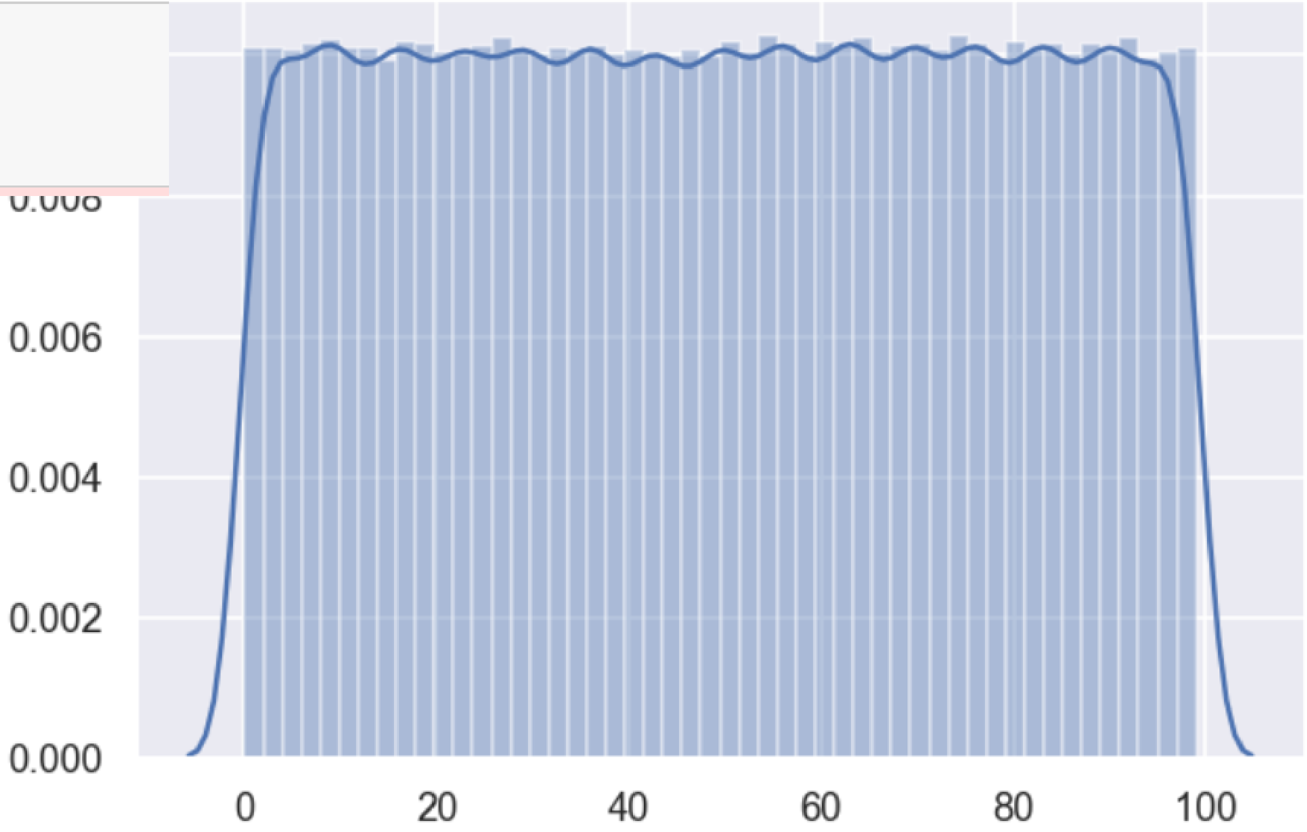
- All the samples will follow an approximate normal distribution pattern, with all variances being approximately equal to the variance of the population, divided by each sample's size.
- In other words, the Distribution of the sample estimates approaches Normal distribution irrespective of Population's distribution if the sample size is large enough {say >30}

Generation of Random Population

```
data=np.random.randint(100,size=10_00_000)
data[:10]
```

```
array([43, 35, 38, 84, 24, 54,  8, 66, 43, 23])
```

```
plt.figure(dpi=120)
sns.distplot(data)
plt.show()
```



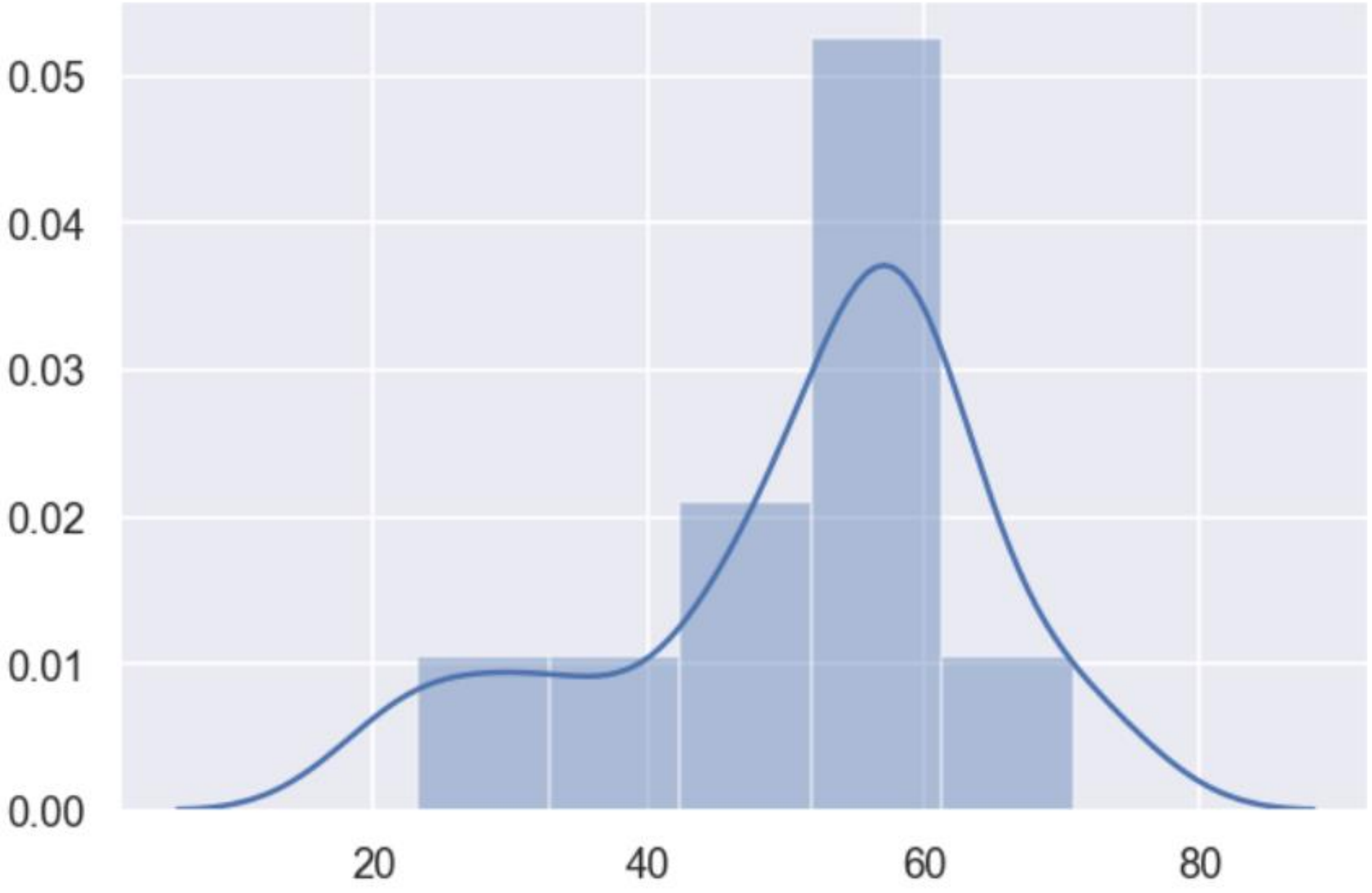
Sample Estimate

- A sample of size 10 is extracted and Mean is calculated

```
np.random.choice(data,size=10).mean()
```

```
67.9
```

By Extracting 10 Samples



By extracting more sample

- By extracting more samples the distribution tends to be Normal

```
print("Skewness",skew(data_mean))
print("kurtosis",kurtosis(data_mean))
```

```
Skewness 0.04475811772203599
kurtosis 0.06408090385800147
```