Probability distributions, random variables

This notebook illustrates the following concepts using simple scripts and functions from `Scipy` and `Numpy` packages.

- Random variables
- Law of the large number
- Expected value
- Discrete probability distributions
- Concitinuous probability distributions
- Moments, variance, and other properties of probability distributions

- What is Probability???

  likely something is to happen value ranges from 0 to 1, nearing to 0 less probability and nearing to 1 it is greater possibility of likely to happen
- Experiment – are the uncertain situations, which could have multiple outcomes. Whether it rains on a daily basis is an experiment.
- Outcome is the result of a single trial. So, if it rains today, the outcome of today’s trial from the experiment is “It rained”
- Event is one or more outcome from an experiment. “It rained” is one of the possible event for this experiment.
- Probability is a measure of how likely an event is. So, if it is 60% chance that it will rain tomorrow, the probability of Outcome “it rained” for tomorrow is 0.6
- Why it is needed ??
- What is Random Variable ?
- A random variable, usually written X, is a variable whose possible values are numerical outcomes of a random phenomenon. There are two types of random variables, discrete and continuous.

X = outcome of a coin toss / values that could be considered while tossing a coin

Possible Outcomes:

1 if heads 0 if tails

In [1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Throwing dice many times (illustrating the _Law of large numbers_)

When we throw dice a large number of times, the average reaches 3.5 which is the expected value.

In [2]:
```
dice = [x for x in range(1,7)]
```

In [3]:
```
print("A fair dice has 6 faces:",dice)
```

```
A fair dice has 6 faces: [1, 2, 3, 4, 5, 6]
```

In [4]:
```
def throw_dice(n=10):
    """
    Throw a (fair) die n number of times and returns the result in an array
    """
    r = []
    for _ in range(n):
        r.append(random.choice(dice))
    return np.array(r)
```

In [5]:
```
throw_dice(10)
```

Out[5]:
```
array([6, 6, 2, 6, 1, 1, 6, 1, 3, 3])
```

In [6]:
```
throw_dice(5)
```

Out[6]:
```
array([1, 6, 3, 1, 5])
```

In [7]:
```
for i in [1,5,10,50,100,500,1000,5000,10000]:
    print("Average of {} dice throws: {}".format(i,round(throw_dice(i).mean()
```

```
Average of 1 dice throws: 6.0
Average of 5 dice throws: 3.2
Average of 10 dice throws: 3.5
Average of 50 dice throws: 3.44
Average of 100 dice throws: 3.63
Average of 500 dice throws: 3.55
Average of 1000 dice throws: 3.51
Average of 5000 dice throws: 3.48
Average of 10000 dice throws: 3.48
```

In [8]:
```
E[x]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-6aa243b639ae> in <module>
----> 1 E[x]

NameError: name 'E' is not defined
```

Expected value of a function

Expected value or mean: the weighted average of the possible values, using their probabilities as their weights; or the continuous analog thereof.

Let X be a random variable with a finite number of finite outcomes $x_1,x_2,x_3,...$ occurring with probabilities $p_1,p_2,p_3,...$ respectively. The expectation of X is, then, defined as

$$E[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k$$

Since, all the probabilities p_1, p_2, p_3, add up to 1, $p_1 + p_2 + p_3 + ... = 1$, it is the **weighted average**.

For, continuous probability distributions, with a density function (PDF) of $f(x)$, the expected value is given by,

$$\mathrm{E}[X] = \int_{\mathbb{R}} x f(x)\, dx.$$

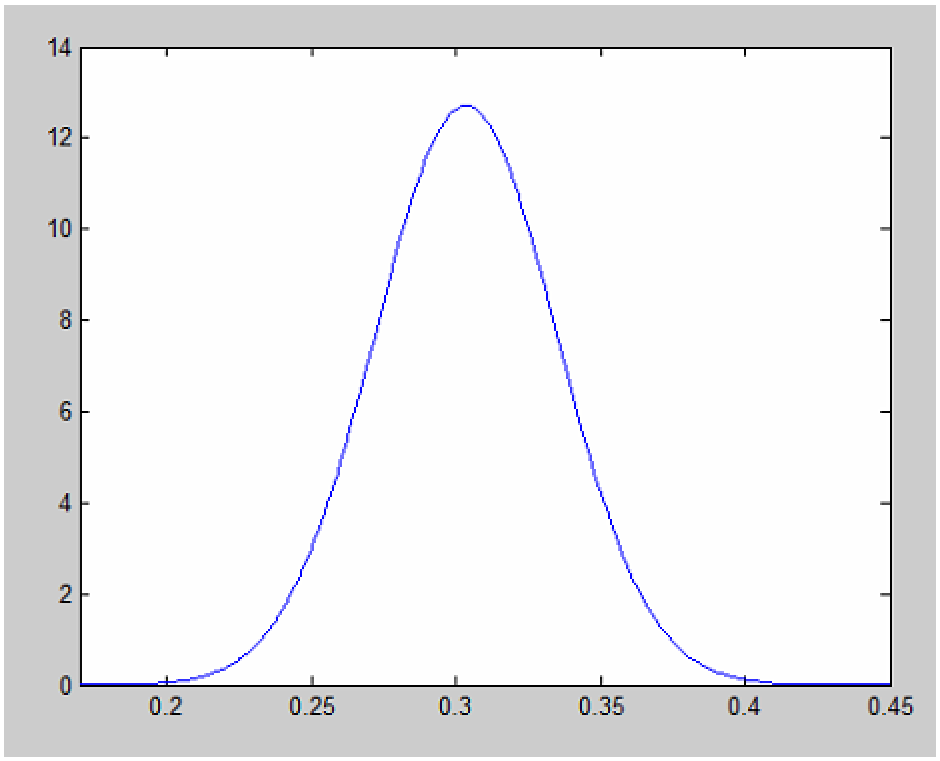
Discrete and Continuous Distributions

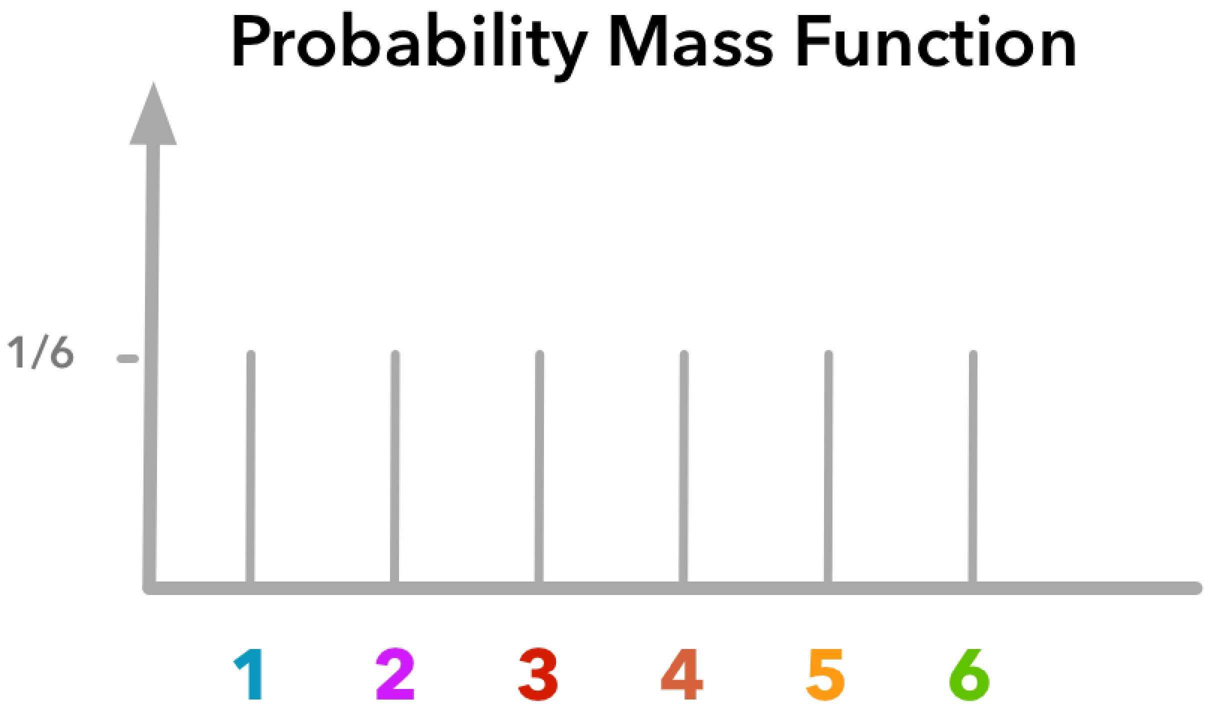
Probability distributions are generally divided into two classes.

A **discrete probability distribution** (applicable to the scenarios where the set of possible outcomes is discrete, such as a coin toss or a roll of dice) can be encoded by a discrete list of the probabilities of the outcomes, known as a probability mass function (https://en.wikipedia.org/wiki/Probability_mass_function).

On the other hand, a **continuous probability distribution** (applicable to the scenarios where the set of possible outcomes can take on values in a continuous range (e.g. real numbers), such as the temperature on a given day) is typically described by probability density functions (with the probability of any individual outcome actually being 0). Such distributions are generally described with the help of probability density functions (https://en.wikipedia.org/wiki/Probability_density_function).

- PDF

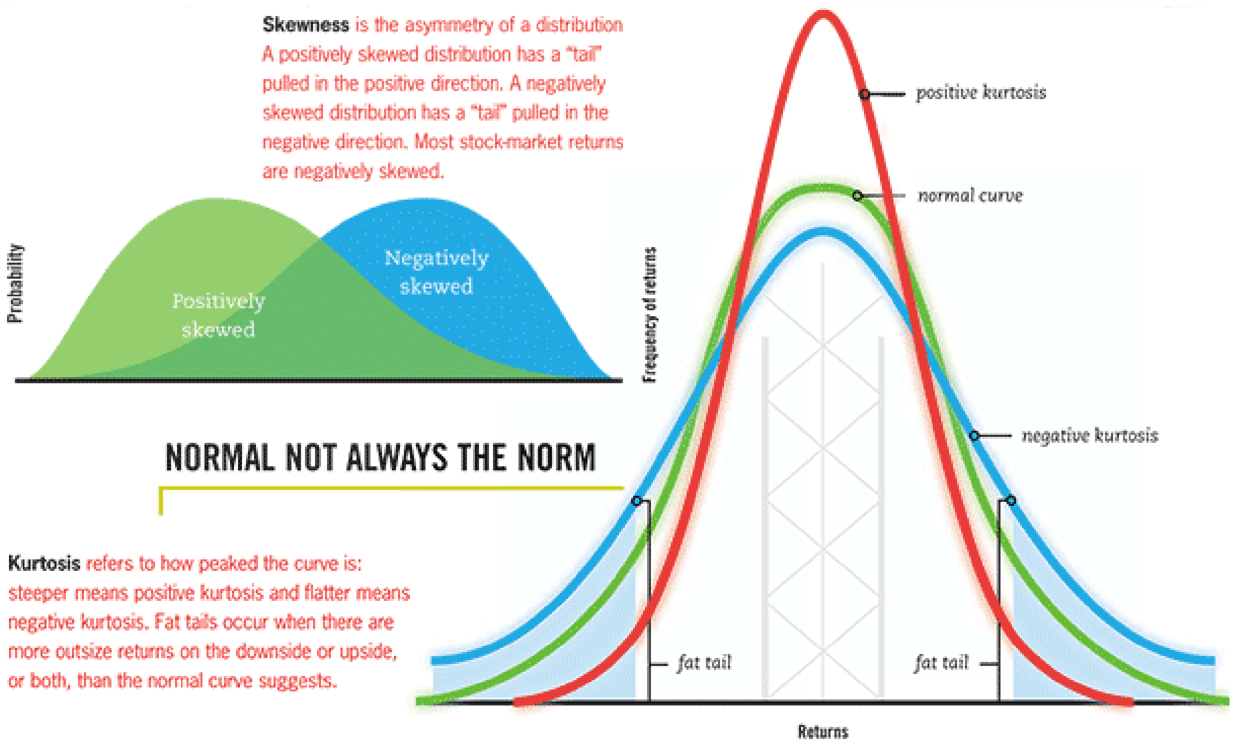




Some Essential Terminologies

- **Mode**: for a discrete random variable, the value with highest probability (the location at which the probability mass function has its peak); for a continuous random variable, a location at which the probability density function has a local peak.
- **Support**: the smallest closed set whose complement has probability zero.

- **Head**: the range of values where the pmf or pdf is relatively high.
- **Tail**: the complement of the head within the support; the large set of values where the pmf or pdf is relatively low.
- **Expected value or mean**: the weighted average of the possible values, using their probabilities as their weights; or the continuous analog thereof.
- **Median**: the value such that the set of values less than the median, and the set greater than the median, each have proba bilities no greater than one-half.
- **Variance**: the second moment of the pmf or pdf about the mean; an important measure of the dispersion of the distribution.
- **Standard deviation**: the square root of the variance, and hence another measure of dispersion.
- **Symmetry**: a property of some distributions in which the portion of the distribution to the left of a specific value is a mirror image of the portion to its right.
- **Skewness**: a measure of the extent to which a pmf or pdf "leans" to one side of its mean. The third standardized moment of the distribution.
- **Kurtosis**: a measure of the "fatness" of the tails of a pmf or pdf. The fourth standardized moment of the distribution.


Skewness is the asymmetry of a distribution A positively skewed distribution has a "tail" pulled in the positive direction. A negatively skewed distribution has a "tail" pulled in the negative direction. Most stock-market returns are negatively skewed.
Probability
Positively skewed
Negatively skewed
NORMAL NOT ALWAYS THE NORM
Kurtosis refers to how peaked the curve is: steeper means positive kurtosis and flatter means negative kurtosis. Fat tails occur when there are more outsize returns on the downside or upside, or both, than the normal curve suggests.
Frequency of returns
positive kurtosis
normal curve
negative kurtosis
fat tail
fat tail
Returns

Quick mathematical definitions of mean, variance, skewness, and kurtosis with respect to a PDF $P(x)$

$$\text{1st raw moment } \textbf{Mean (1st moment) :} \int x.\,P(x).\,dx$$

$$\text{Centralized 2nd moment } \textbf{Variance (2nd moment) :} \int (x-\mu)^2.\,P(x).\,dx$$

$$\text{Pearson's 3rd moment (Standardized) } \textbf{Skew (3rd moment) :} \int \left(\frac{x-\mu}{\sigma}\right)^3.\,P(x).\,dx$$

$$\text{Pearson's 4th moment (Standardized) } \textbf{Kurtosis (4th moment) :} \int \left(\frac{x-\mu}{\sigma}\right)^4 . P(x). dx$$

Central Tendancy

- Measure of Central Tendency - Mode, Mean and Median
- Measure of dispersion - SD, Variance
- Measure of Skewness

Bernoulii distribution

The Bernoulli distribution, named after Swiss mathematician Jacob Bernoulli (https://en.wikipedia.org/wiki/Jacob_Bernoulli), is the probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability $q = 1 - p$ — i.e., the probability distribution of any single experiment that asks a ***yes–no question***; the question results in a boolean-valued outcome, a single bit of information whose value is success/yes/true/one with probability p and failure/no/false/zero with probability q.

It can be used to represent a coin toss where 1 and 0 would represent "head" and "tail" (or vice versa), respectively. In particular, unfair coins would have $p \neq 0.5$.

The probability mass function f of this distribution, over possible outcomes k, is

$$f(k;p) = \begin{cases} p & \text{if } k = 1, \\ 1-p & \text{if } k = 0. \end{cases}$$

```
In [9]:  from scipy.stats import bernoulli
```

Generate random variates

```
In [10]: # p=0.5 i.e. fair coin
         bernoulli.rvs(p=0.5,size=20)

Out[10]: array([1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

Loaded coin towards tail, p=0.2 for head

In [11]:
```
# p=0.2 i.e. more tails (0) than heads(1)
bernoulli.rvs(p=0.2,size=20)
```

Out[11]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

Loaded coin towards head, p=0.8 for head

In [12]:
```
# p=0.8 i.e. more heads (1) than tails (0)
bernoulli.rvs(p=0.8,size=20)
```

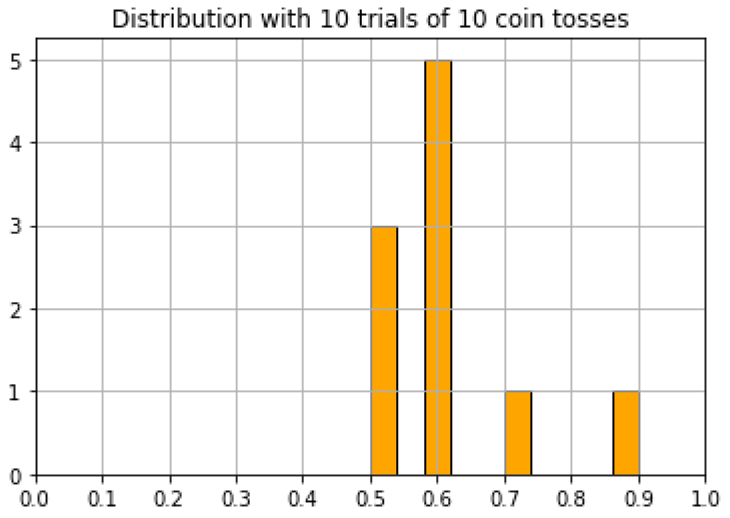
Out[12]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

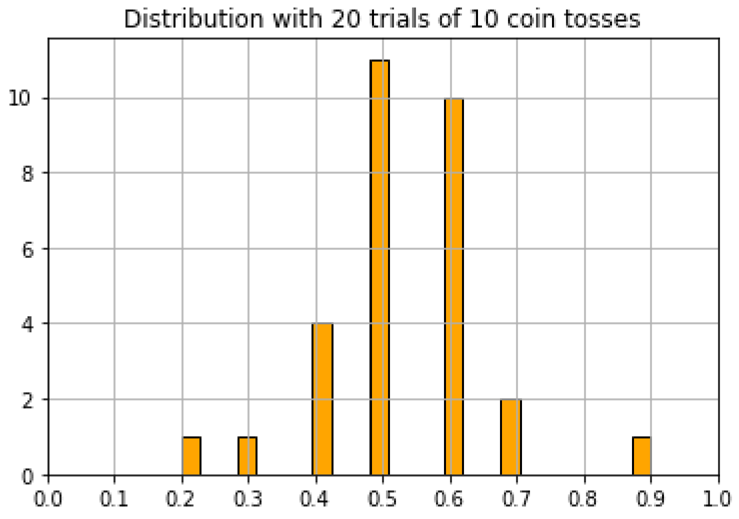
Note, a single run or even a small number of runs may not produce the expected distribution of 1's and 0's.

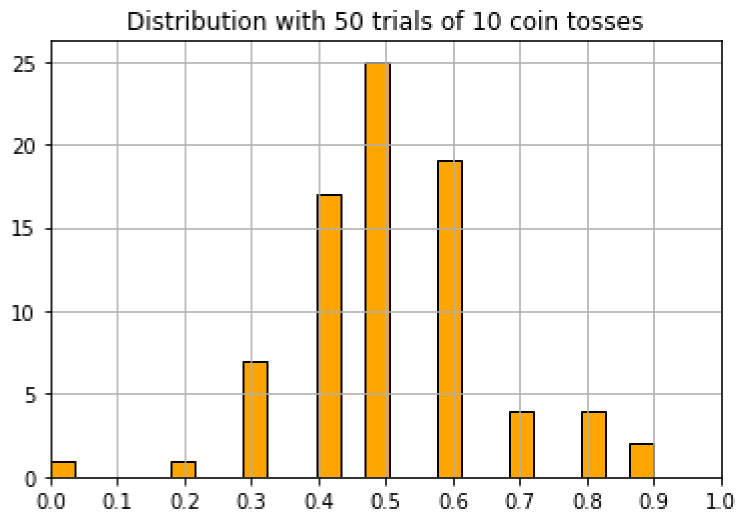
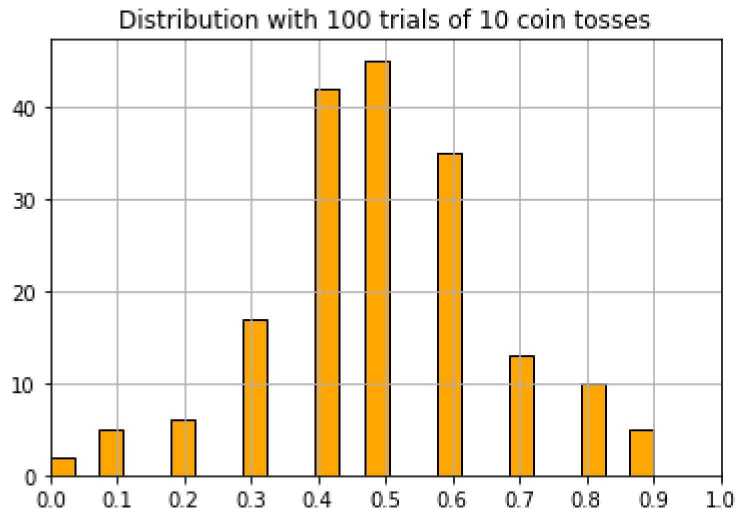
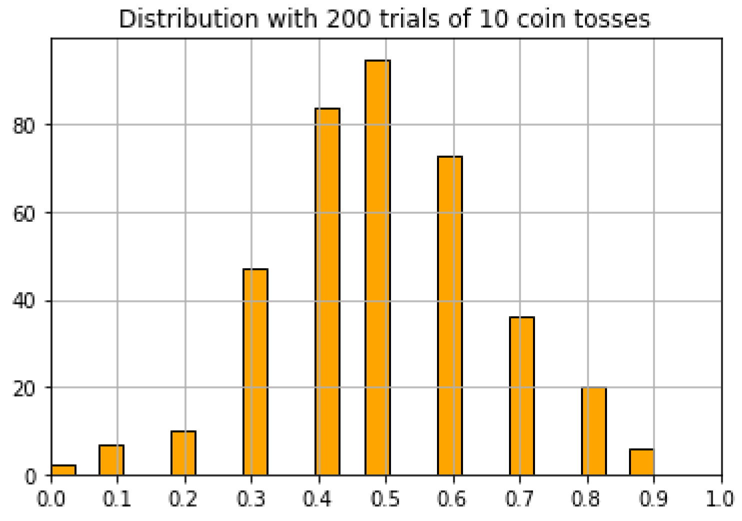
For example, if you assign $p = 0.5$, you may not get half 1's and half 0's every time you evaluate the function. Experiment with N number of trials to see how the probability distribution gradually centers around 0.5.

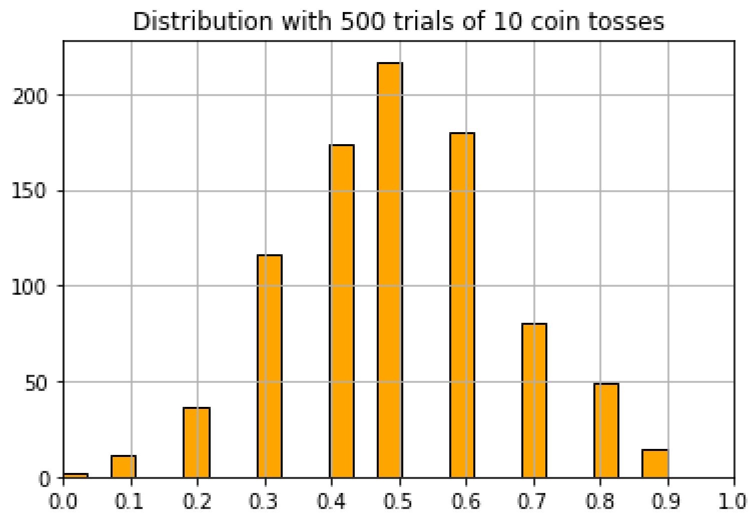
In [13]:

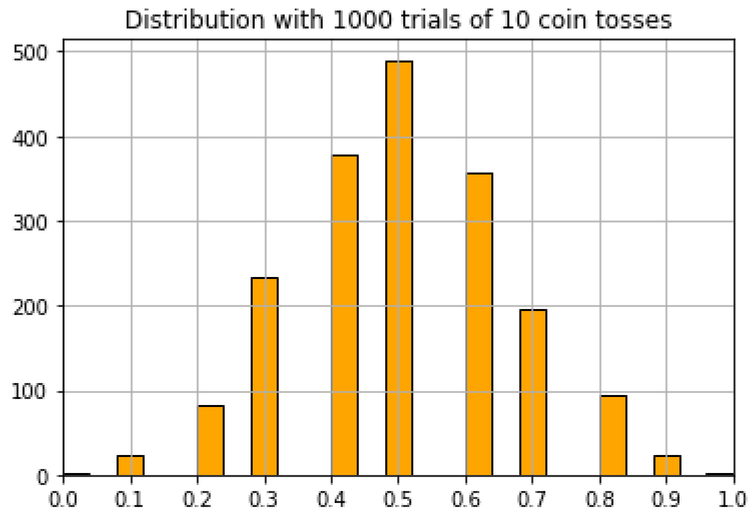
```python
N_trials = [10,20,50,100,200,500,1000,2000,5000] # Number of trials
pr=0.5 # Fair coin toss probability
av = [] # Empty list to store the average of the random variates

# Generate 10 variates every time and take the average. That should be # of 1
for i in N_trials:
    for n in range(1,i+1):
        av.append(np.mean(bernoulli.rvs(p=pr,size=10)))
    if (i==10):
        plt.title("Distribution with {} trials of 10 coin tosses".format(i))
        plt.hist(av,bins=10,edgecolor='k',color='orange')
        plt.xlim(0.0,1.0)
        plt.xticks([0.1*i for i in range(11)])
        plt.grid(True)
        plt.show()
    else:
        plt.title("Distribution with {} trials of 10 coin tosses".format(i))
        plt.hist(av,bins=25,edgecolor='k',color='orange')
        plt.xlim(0.0,1.0)
        plt.xticks([0.1*i for i in range(11)])
        plt.grid(True)
        plt.show()
```

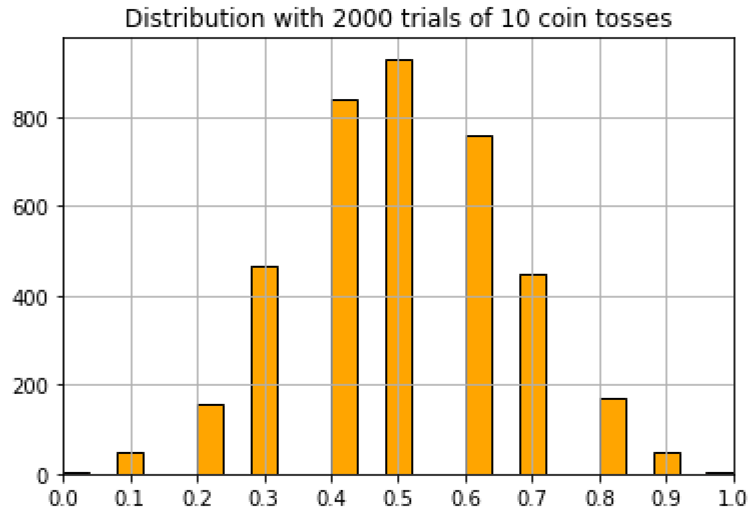


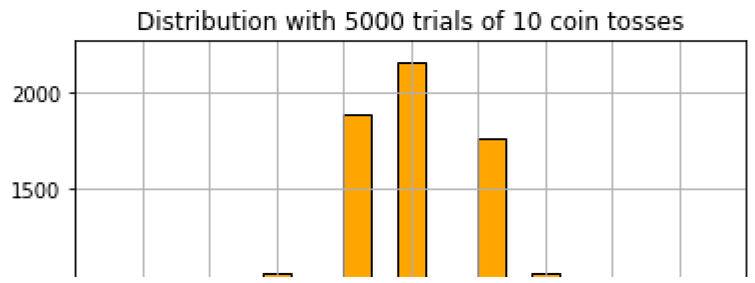



Distribution with 50 trials of 10 coin tosses
0
5
10
15
20
25
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Distribution with 100 trials of 10 coin tosses
0
10
20
30
40
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Distribution with 200 trials of 10 coin tosses
0
20
40
60
80
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Distribution with 500 trials of 10 coin tosses
0
50
100
150
200
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Distribution with 1000 trials of 10 coin tosses
0
100
200
300
400
500
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Distribution with 2000 trials of 10 coin tosses
0
200
400
600
800
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0



Mean, variance, skew, and kurtosis

Use `bernoulli.stats()` method

In [14]:
```python
print("A fair coin is spinning...\n"+"-"*30)
pr=0.5 # Fair coin toss probability
mean, var, skew, kurt = bernoulli.stats(p=pr, moments='mvsk')
print("Mean:",mean)
print("Variance:",var)
print("Skew:",skew)
print("Kurtosis:",kurt)
```

```
A fair coin is spinning...
------------------------------
Mean: 0.5
Variance: 0.25
Skew: 0.0
Kurtosis: -2.0
```

In [15]:
```python
print("\nNow a biased coin is spinning...\n"+"-"*35)
pr=0.2 # Biased coin toss probability
mean, var, skew, kurt = bernoulli.stats(p=pr, moments='mvsk')
print("Mean:",mean)
print("Variance:",var)
print("Skew:",skew)
print("Kurtosis:",kurt)
```

```

Now a biased coin is spinning...
-----------------------------------
Mean: 0.2
Variance: 0.16000000000000003
Skew: 1.4999999999999993
Kurtosis: 0.2499999999999991
```

Probability mass function (PMF) and cumulative distribution function (CDF)

In [16]:
```
rv = bernoulli(0.6)
x=0
print("Probability mass function for {}: {}".format(x,rv.pmf(x)))
x=0.5
print("Probability mass function for {}: {}".format(x,rv.pmf(x)))
x=1.0
print("Probability mass function for {}: {}".format(x,rv.pmf(x)))
x=1.2
print("Probability mass function for {}: {}".format(x,rv.pmf(x)))
```

```
Probability mass function for 0: 0.4
Probability mass function for 0.5: 0.0
Probability mass function for 1.0: 0.6
Probability mass function for 1.2: 0.0
```

In [17]:
```
print("CDF for x < 0:",rv.cdf(-2))
print("CDF for 0< x <1:",rv.cdf(0.75))
print("CDF for x >1:",rv.cdf(2))
```

```
CDF for x < 0: 0.0
CDF for 0< x <1: 0.4
CDF for x >1: 1.0
```

Binomial distribution

The binomial distribution with parameters n and p is the discrete probability distribution of the **number of successes in a sequence of n independent experiments, each asking a _yes–no question_**, and each with its own boolean-valued outcome: a random variable containing single bit of information: success/yes/true/one (with probability p) or failure/no/false/zero (with probability $q = 1 - p$). A single success/failure experiment is also called a *Bernoulli trial* or *Bernoulli experiment* and a sequence of outcomes is called a *Bernoulli process*.

For a single trial, i.e., n = 1, the binomial distribution is a **Bernoulli distribution**. The binomial distribution is the basis for the popular binomial test (https://en.wikipedia.org/wiki/Binomial_test) of statistical significance (https://en.wikipedia.org/wiki/Statistical_significance).

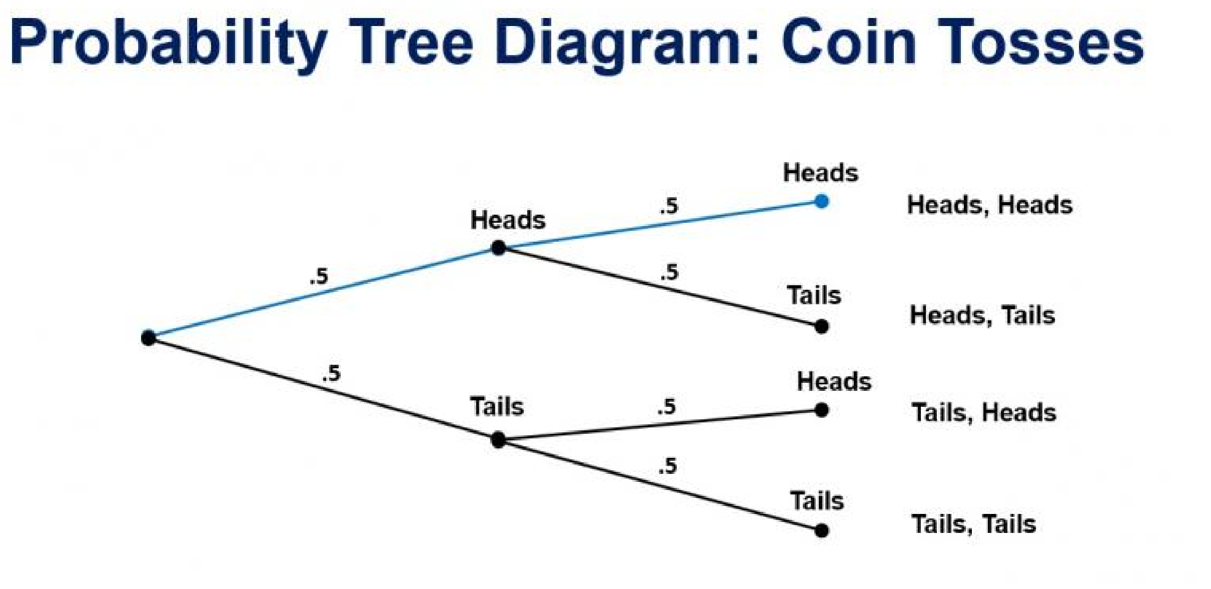
The binomial distribution is frequently used to model the number of successes in a sample of size n drawn with replacement from a population of size N. If the sampling is carried out without replacement, the draws are not independent and so the resulting distribution is a **hypergeometric distribution (https://en.wikipedia.org/wiki/Hypergeometric_distribution)**, not a binomial one. However, for N much larger than n, the binomial distribution remains a good approximation, and is widely used.

In general, if the random variable X follows the binomial distribution with parameters n ∈ $\mathbb{N}$ and p ∈ [0,1], we write X ~ B(n, p). The probability of getting exactly k successes in n trials is given by the probability mass function:

$$\Pr(k;n,p) = \Pr(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for k = 0, 1, 2, ..., n, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



Mean, variance, skew, and kurtosis

$$\textbf{Mean} = n.\,p,\ \ \textbf{Variance} = n.\,p(1-p), \textbf{skewness} = \frac{1-2p}{\sqrt{n.\,p(1-p)}},\ \textbf{kurtosis} = \frac{1-6p(1-}{n.\,p(1-p)}$$

Use binom.stats() method

Generate random variates

8 coins are flipped (or 1 coin is flipped 8 times), each with probability of success (1) of 0.25. This trial/experiment is repeated for 10 times.

In [18]:
```python
from scipy.stats import binom
import numpy as np
k=binom.rvs(8,0.25,size=10)
print("Number of success for each trial:",k)
print("Average of the success:", np.mean(k))
```

```
Number of success for each trial: [2 0 1 1 3 3 4 1 4 2]
Average of the success: 2.1
```

In [19]:

```
print("A fair coin (p=0.5) is spinning 5 times\n"+"-"*35)
pr=0.5 # Fair coin toss probability
n=5
mean, var, skew, kurt = binom.stats(n=n,p=pr, moments='mvsk')
print("Mean:",mean)
print("Variance:",var)
print("Skew:",skew)
print("Kurtosis:",kurt)
```

```
A fair coin (p=0.5) is spinning 5 times
-----------------------------------
Mean: 2.5
Variance: 1.25
Skew: 0.0
Kurtosis: -0.4
```

In [20]:

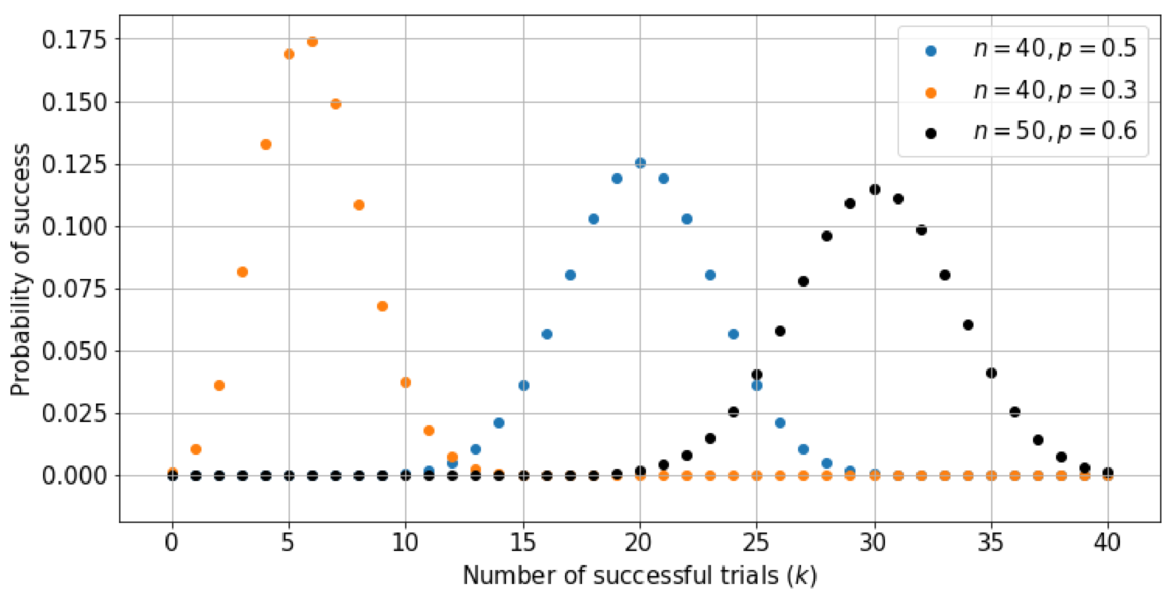
```
print("\nNow a biased coin (p=0.7) is spinning 5 times...\n"+"-"*45)
pr=0.7 # Biased coin toss probability
n=5
mean, var, skew, kurt = binom.stats(n=n,p=pr, moments='mvsk')
print("Mean:",mean)
print("Variance:",var)
print("Skew:",skew)
print("Kurtosis:",kurt)
```

```

Now a biased coin (p=0.7) is spinning 5 times...
---------------------------------------------
Mean: 3.5
Variance: 1.0500000000000003
Skew: -0.39036002917941315
Kurtosis: -0.24761904761904757
```

In [21]:

```
## Vizualizing PMF
import matplotlib.pyplot as plt
%matplotlib inline

n=40
pr=0.5
rv = binom(n,pr)
x=np.arange(0,41,1)
pmf1 = rv.pmf(x)

n=40
pr=0.15
rv = binom(n,pr)
x=np.arange(0,41,1)
pmf2 = rv.pmf(x)

n=50
pr=0.6
rv = binom(n,pr)
x=np.arange(0,41,1)
pmf3 = rv.pmf(x)

plt.figure(figsize=(12,6))
plt.title("Probability mass function: $\\binom{n}{k}\, p^k (1-p)^{n-k}$\n",fc
plt.scatter(x,pmf1)
plt.scatter(x,pmf2)
plt.scatter(x,pmf3,c='k')
plt.legend(["$n=40, p=0.5$","$n=40, p=0.3$","$n=50, p=0.6$"],fontsize=15)
plt.xlabel("Number of successful trials ($k$)",fontsize=15)
plt.ylabel("Probability of success",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.grid(True)
plt.show()
```

Probability mass function: $\binom{n}{k}\, p^k(1-p)^{n-k}$

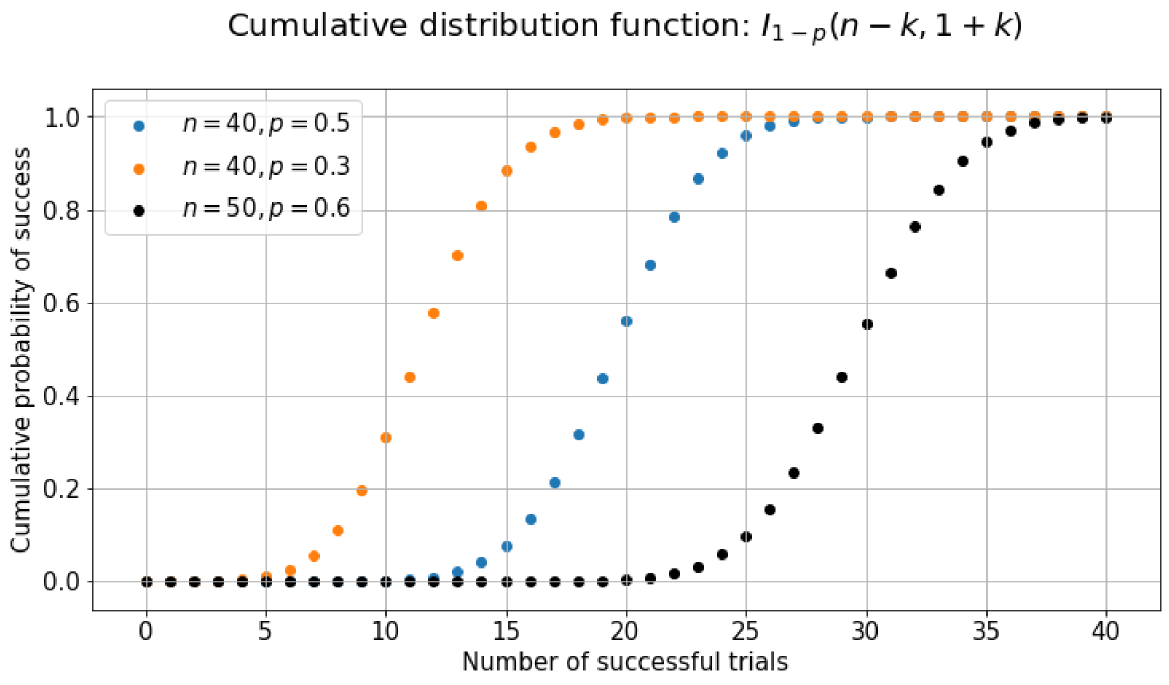


Visualize the cumulative distrubition function (cdf)

Cumulative distribution function for binomial distribution can also be represented in terms of the regularized incomplete beta function (https://en.wikipedia.org/wiki/Regularized_incomplete_beta_function), as follows

$$\begin{aligned} F(k; n, p) &= \Pr(X \leq k) \\ &= I_{1-p}(n-k, k+1) \\ &= (n-k)\binom{n}{k}\int_0^{1-p} t^{n-k-1}(1-t)^k \, dt. \end{aligned}$$

```
In [22]:  n=40
          pr=0.5
          rv = binom(n,pr)
          x=np.arange(0,41,1)
          cdf1 = rv.cdf(x)

          n=40
          pr=0.3
          rv = binom(n,pr)
          x=np.arange(0,41,1)
          cdf2 = rv.cdf(x)

          n=50
          pr=0.6
          rv = binom(n,pr)
          x=np.arange(0,41,1)
          cdf3 = rv.cdf(x)

          plt.figure(figsize=(12,6))
          plt.title("Cumulative distribution function: $I_{1-p}(n - k, 1 + k)$\n",fonts
          plt.scatter(x,cdf1)
          plt.scatter(x,cdf2)
          plt.scatter(x,cdf3,c='k')
          plt.legend(["$n=40, p=0.5$","$n=40, p=0.3$","$n=50, p=0.6$"],fontsize=15)
          plt.xlabel("Number of successful trials",fontsize=15)
          plt.ylabel("Cumulative probability of success",fontsize=15)
          plt.xticks(fontsize=15)
          plt.yticks(fontsize=15)
          plt.grid(True)
          plt.show()
```



Interval that contains a specific percentage of distribution

Use `binom.interval` method

In [23]:
```
n=40
pr=0.3
percent=25
interval = binom.interval(percent/100,n,pr,loc=0)
print("Interval that contains {} percent of distribution with an experiment w
      .format(percent,n,pr,interval))
```

```
Interval that contains 25 percent of distribution with an experiment with 4
0 trials and 0.3 success probability  is: (11.0, 13.0)
```

Poisson Distribution

The Poisson distribution (named after French mathematician Siméon Denis Poisson), is a discrete probability distribution that **expresses the probability of a given number of events occurring in a fixed interval of time or space if these events occur with a known constant rate and independently of the time since the last event.** The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.

For instance, an individual keeping track of the amount of mail they receive each day may notice that they receive an average number of 4 letters per day. If receiving any particular piece of mail does not affect the arrival times of future pieces of mail, i.e., if pieces of mail from a wide range of sources arrive independently of one another, then a reasonable assumption is that the number of pieces of mail received in a day obeys a Poisson distribution. Other examples, that may follow a Poisson distribution, include

- number of phone calls received by a call center per hour
- number of decay events per second from a radioactive source
- The number of meteors greater than 1 meter diameter that strike Earth in a year
- The number of patients arriving in an emergency room between 10 and 11 pm

Poisson distribution is a limiting case of a Binomial Distribution where the number of trials is sufficiently bigger than the number of successes one is asking about i.e. $n >> 1 >> p$

An event can occur 0, 1, 2, … times in an interval. The average number of events in an interval is designated λ. This is the event rate, also called the rate parameter. The probability of observing k events in an interval is given by the equation

$P(k \text{ events in interval}) = e^{-\lambda} \frac{\lambda^k}{k!}$

where,

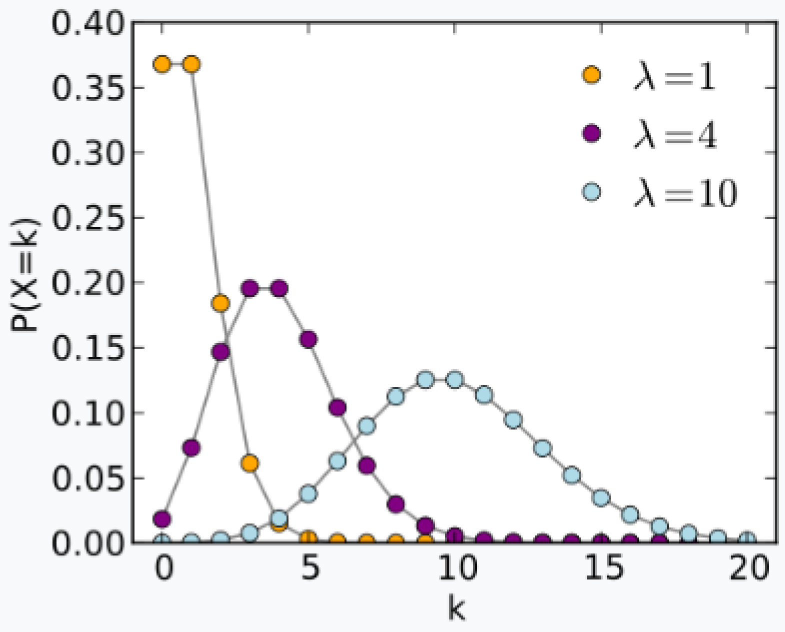
λ is the average number of events per interval

e is the number 2.71828... (Euler's number) the base of the natural logarithms

k takes values 0, 1, 2, … k! = k × (k − 1) × (k − 2) × … × 2 × 1 is the factorial of k.

Poisson Distribution

Probability mass function



The horizontal axis is the index *k*, the number of occurrences. *λ* is the expected number of occurrences, which need not be an integer. The vertical axis is the probability of *k* occurrences given *λ*. The function is defined only at integer values of *k*. The connecting lines are only guides for the eye.

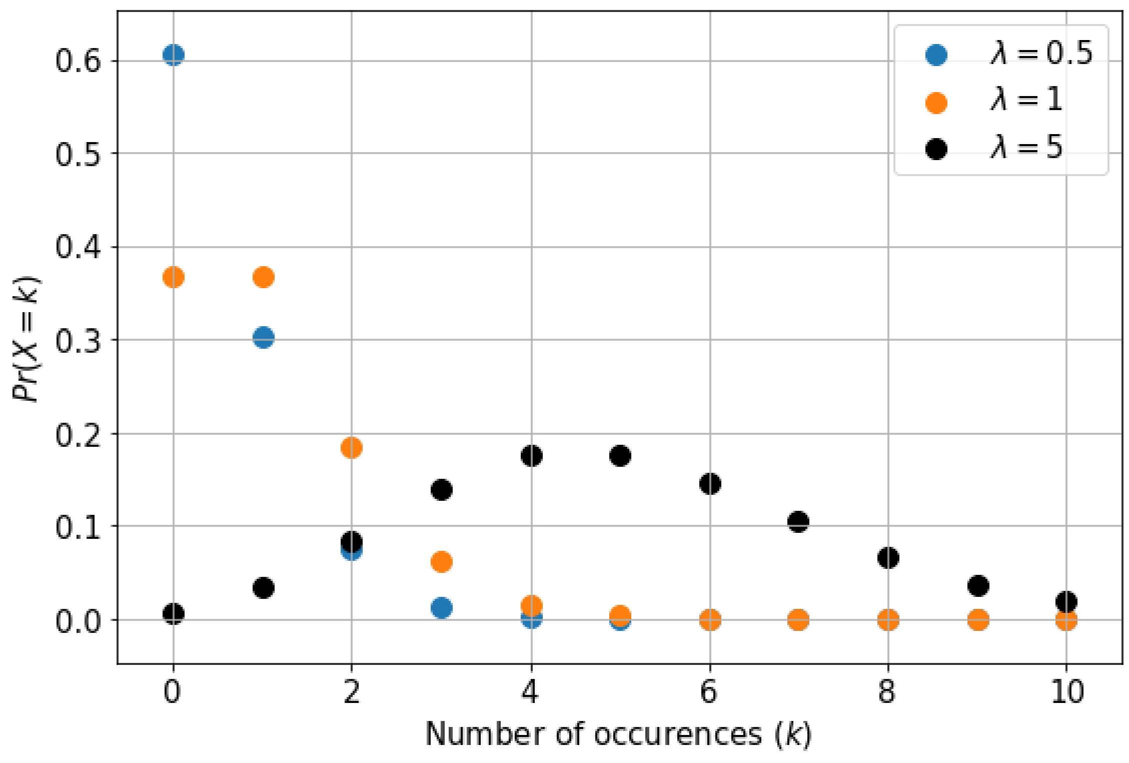
- The average number of major storms in your city is 2 per year. What is the probability that exactly 3 storms will hit your city next year
- λ = 2
- k = 3

$P(k \text{ events in interval}) = e^{-2}\,\frac{2^3}{3!}$ = 0.180

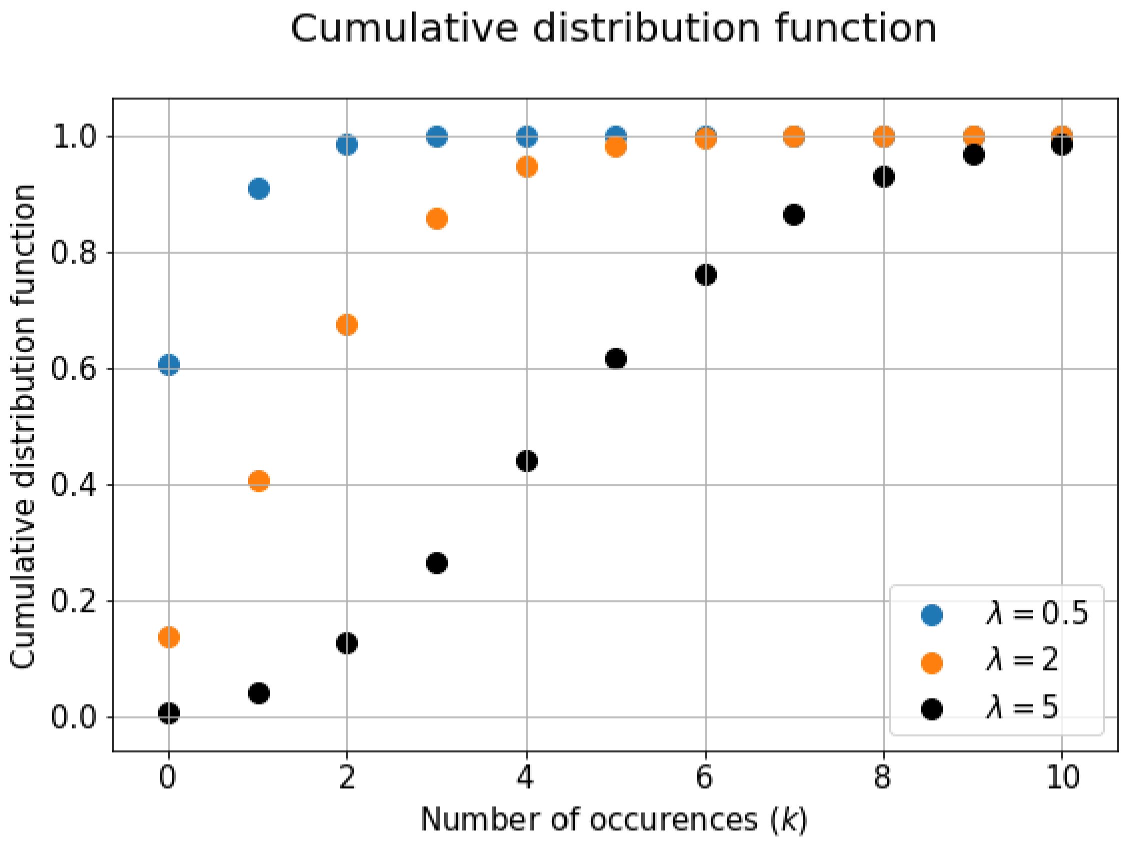
- The probability of 3 storms happening next year is 0.180, or 18%

```
In [24]:  from scipy.stats import poisson
          la=0.5
          rv = poisson(la)
          x=np.arange(0,11,1)
          pmf1 = rv.pmf(x)

          la=1
          rv = poisson(la)
          x=np.arange(0,11,1)
          pmf2 = rv.pmf(x)

          la=5
          rv = poisson(la)
          x=np.arange(0,11,1)
          pmf3 = rv.pmf(x)

          plt.figure(figsize=(9,6))
          plt.title("Probability mass function: $e^{-\lambda}{(\lambda^k/k!)}$\n",fonts
          plt.scatter(x,pmf1,s=100)
          plt.scatter(x,pmf2,s=100)
          plt.scatter(x,pmf3,c='k',s=100)
          plt.legend(["$\lambda=0.5$","$\lambda=1$","$\lambda=5$"],fontsize=15)
          plt.xlabel("Number of occurences ($k$)",fontsize=15)
          plt.ylabel("$Pr(X=k)$",fontsize=15)
          plt.xticks(fontsize=15)
          plt.yticks(fontsize=15)
          plt.grid(True)
          plt.show()
```

Probability mass function: $e^{-\lambda}(\lambda^k/k!)$



```
In [25]:  la=0.5
          rv = poisson(la)
          x=np.arange(0,11,1)
          cdf1 = rv.cdf(x)

          la=2
          rv = poisson(la)
          x=np.arange(0,11,1)
          cdf2 = rv.cdf(x)

          la=5
          rv = poisson(la)
          x=np.arange(0,11,1)
          cdf3 = rv.cdf(x)

          plt.figure(figsize=(9,6))
          plt.title("Cumulative distribution function\n",fontsize=20)
          plt.scatter(x,cdf1,s=100)
          plt.scatter(x,cdf2,s=100)
          plt.scatter(x,cdf3,c='k',s=100)
          plt.legend(["$\lambda=0.5$","$\lambda=2$","$\lambda=5$"],fontsize=15)
          plt.xlabel("Number of occurences ($k$)",fontsize=15)
          plt.ylabel("Cumulative distribution function",fontsize=15)
          plt.xticks(fontsize=15)
          plt.yticks(fontsize=15)
          plt.grid(True)
          plt.show()
```



Moments - mean, variance, skew, and kurtosis

Various moments of a Poisson distributed random variable X are as follows:

$$\textbf{Mean} = \lambda,\ \textbf{Variance} = \lambda,\ \textbf{skewness} = \frac{1}{\sqrt{\lambda}},\ \textbf{kurtosis} = \frac{1}{\lambda}$$

Uniform (continuous) distribution

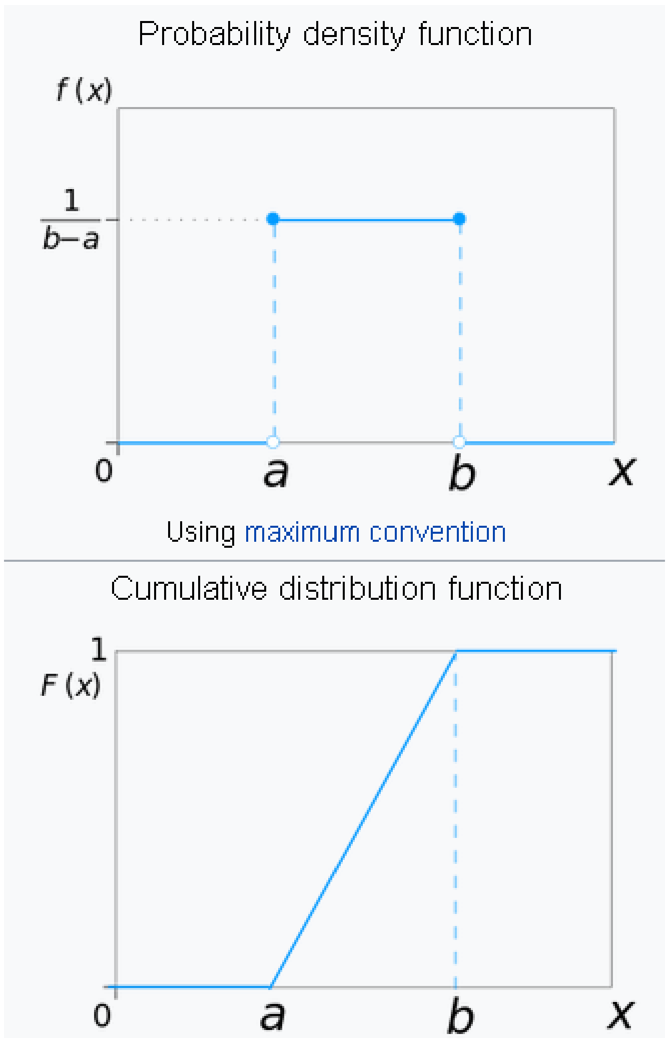
This is the distribution of the likelihood of uniformly randomly selecting an item out of a finite collection.

We are mostly familiar with the discontinuous version of this distribution. For example, in case of throwing a fair dice, the probability distribution of a single throw is given by:

$$\left\{\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right\}$$

For the continuous case, the PDF looks deceptively simple, but the concept is subtle,

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$



Normal (Gaussian) distribution

In probability theory, the normal (or Gaussian or Gauss or Laplace–Gauss) distribution is a very common continuous probability distribution. Normal distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known. A random variable with a Gaussian distribution is said to be normally distributed and is called a normal deviate.

Physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have distributions that are nearly normal. Moreover, many results and methods (such as propagation of uncertainty and least squares parameter fitting) can be derived analytically in explicit form when the relevant variables are normally distributed.

PDF and CDF

The probability density function (PDF) is given by,

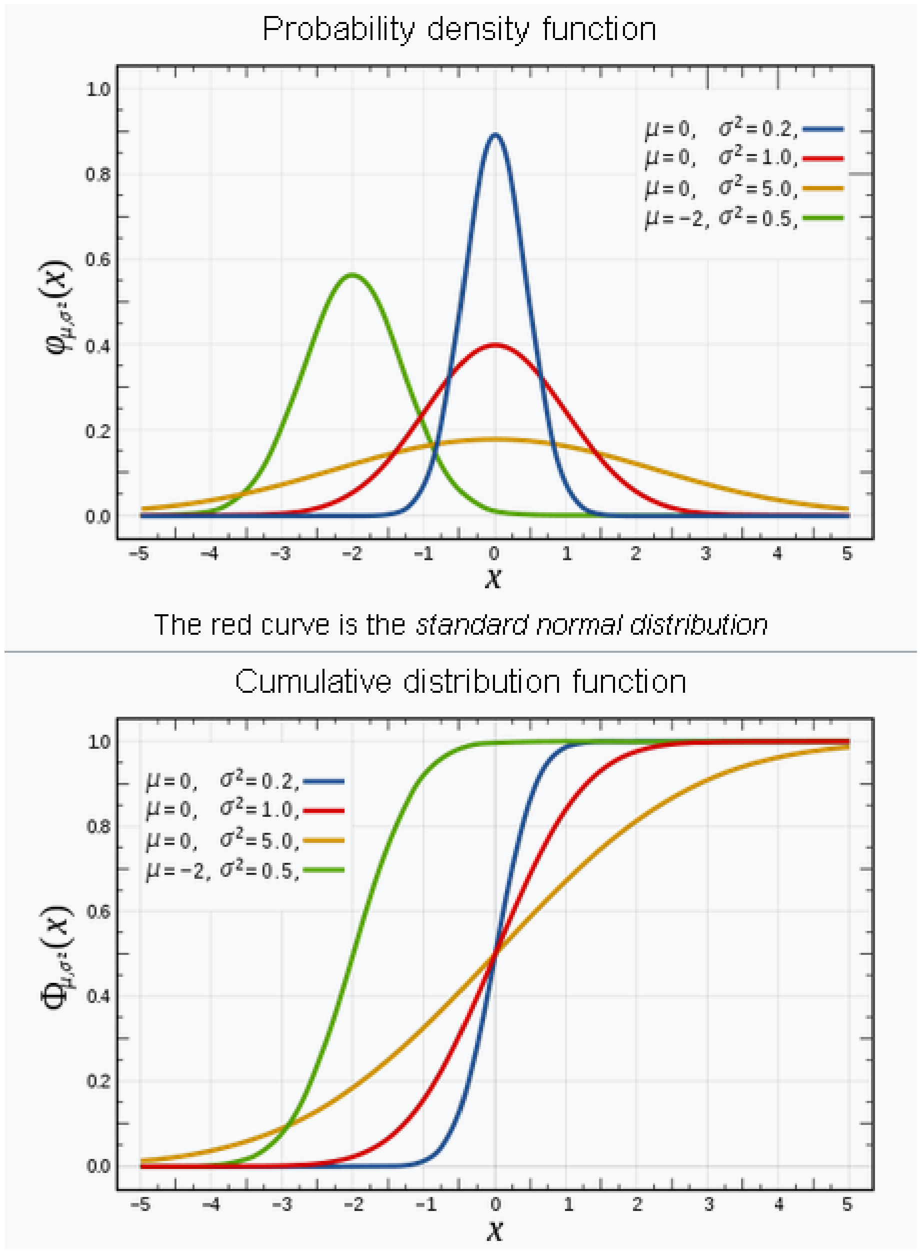
$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where,

- μ is the mean or expectation of the distribution (and also its median and mode),
- σ is the standard deviation, and σ^2 is the variance.

Cumulative distribution function (CDF) is given by,

$$\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

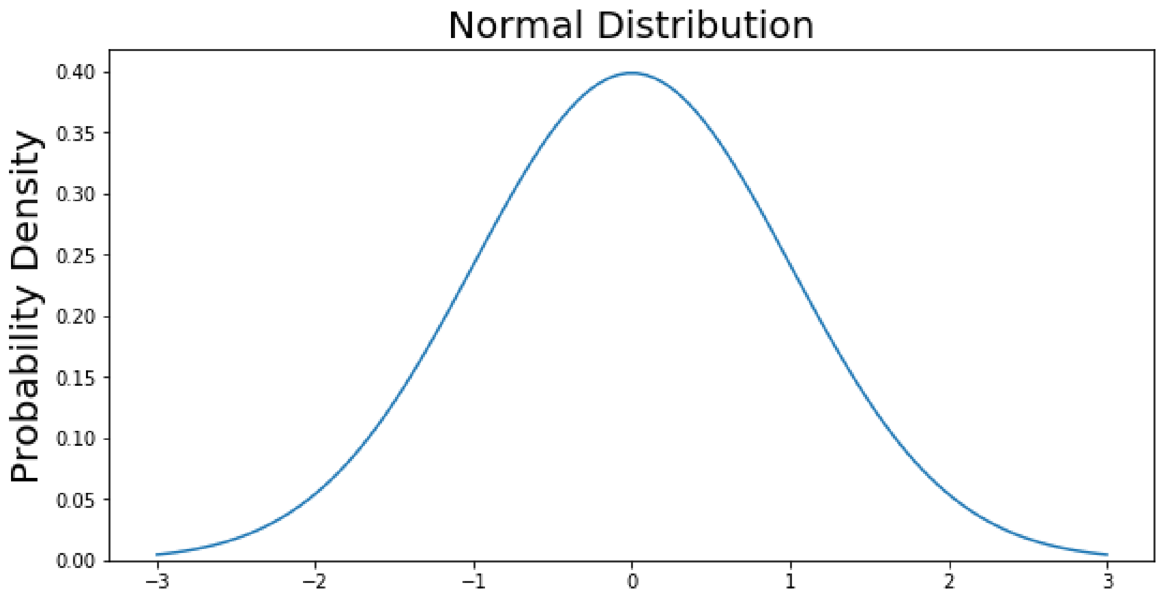


Scipy Stats page:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html#scipy.stats.norm (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html#scipy.stats.norm)

```
In [26]:  from scipy.stats import norm
In [27]:  import numpy as np
          import matplotlib.pyplot as plt
          %matplotlib inline
          x = np.linspace(-3, 3, num = 100)
          print(x)
          constant = 1.0 / np.sqrt(2*np.pi)
          pdf_normal_distribution = constant * np.exp((-x**2) / 2.0)

          fig, ax = plt.subplots(figsize=(10, 5));
          ax.plot(x, pdf_normal_distribution);
          ax.set_ylim(0);
          ax.set_title('Normal Distribution', size = 20);
          ax.set_ylabel('Probability Density', size = 20)
```

```
[-3.         -2.93939394 -2.87878788 -2.81818182 -2.75757576 -2.6969697
 -2.63636364 -2.57575758 -2.51515152 -2.45454545 -2.39393939 -2.33333333
 -2.27272727 -2.21212121 -2.15151515 -2.09090909 -2.03030303 -1.96969697
 -1.90909091 -1.84848485 -1.78787879 -1.72727273 -1.66666667 -1.60606061
 -1.54545455 -1.48484848 -1.42424242 -1.36363636 -1.3030303  -1.24242424
 -1.18181818 -1.12121212 -1.06060606 -1.         -0.93939394 -0.87878788
 -0.81818182 -0.75757576 -0.6969697  -0.63636364 -0.57575758 -0.51515152
 -0.45454545 -0.39393939 -0.33333333 -0.27272727 -0.21212121 -0.15151515
 -0.09090909 -0.03030303  0.03030303  0.09090909  0.15151515  0.21212121
  0.27272727  0.33333333  0.39393939  0.45454545  0.51515152  0.57575758
  0.63636364  0.6969697   0.75757576  0.81818182  0.87878788  0.93939394
  1.          1.06060606  1.12121212  1.18181818  1.24242424  1.3030303
  1.36363636  1.42424242  1.48484848  1.54545455  1.60606061  1.66666667
  1.72727273  1.78787879  1.84848485  1.90909091  1.96969697  2.03030303
  2.09090909  2.15151515  2.21212121  2.27272727  2.33333333  2.39393939
  2.45454545  2.51515152  2.57575758  2.63636364  2.6969697   2.75757576
  2.81818182  2.87878788  2.93939394  3.        ]
```

```
Out[27]: Text(0, 0.5, 'Probability Density')
```



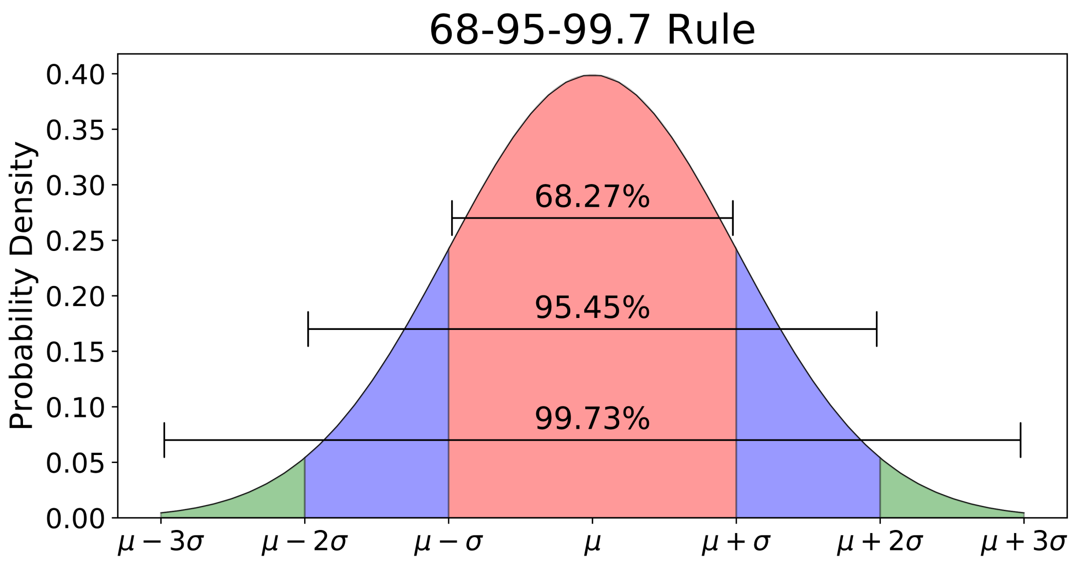
Derive the familiar 68-95-99.7 rule from the basic definition

In [28]:
```python
import scipy
import numpy as np
```

In [29]:
```python
def normalProbabilityDensity(x):
    constant = 1.0 / np.sqrt(2*np.pi)
    return(constant * np.exp((-x**2) / 2.0) )# Integrate PDF from -1 to 1

def integrate_normal(num_sigma):
    result, _ = scipy.integrate.quad(normalProbabilityDensity, -num_sigma, nu
    return round(result,3)
```

In [30]:
```python
print("The percentage of data present within 1 standard deviation:",integrate
print("The percentage of data present within 2 standard deviations:",integrat
print("The percentage of data present within 3 standard deviations:",integrat
```

```
The percentage of data present within 1 standard deviation: 0.683
The percentage of data present within 2 standard deviations: 0.954
The percentage of data present within 3 standard deviations: 0.997
```



Random variable generation using `Numpy.random` module

Numpy offers an amazing module called `Numpy.random` , which has all the important probability distributions built-in for generation. We will check it out for,

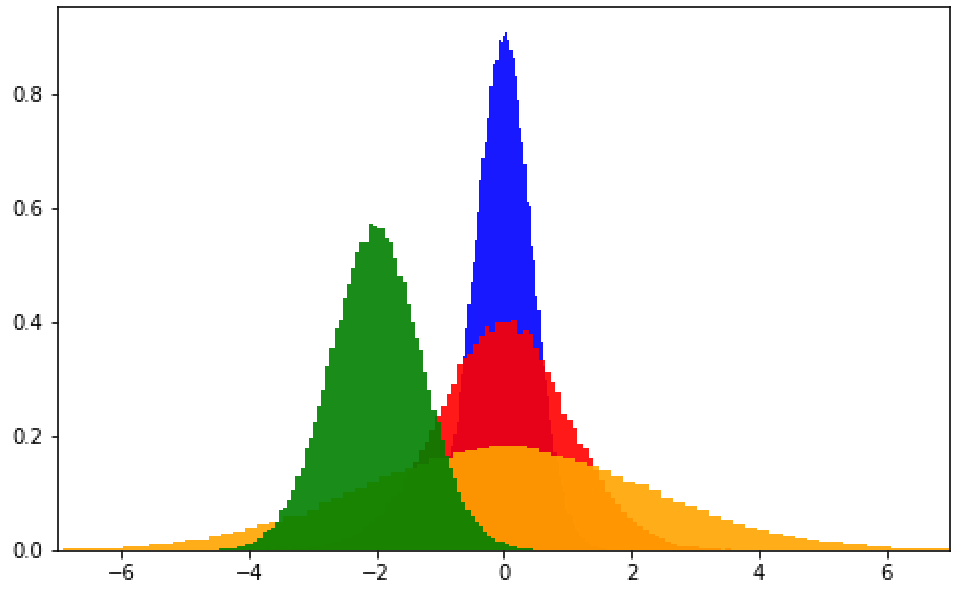
- Normal
- Uniform
- Binomial
- Chi-square
- Poisson
- F-distribution and Student's t-distribution

Generate normally distributed numbers with various mean and std.dev

In `numpy.random.normal` method, the `loc` argument is the mean, adnd the `scale` argument is the std.dev

In [31]:
```python
import numpy as np
a1 = np.random.normal(loc=0,scale=np.sqrt(0.2),size=100000)
a2 = np.random.normal(loc=0,scale=1.0,size=100000)
a3 = np.random.normal(loc=0,scale=np.sqrt(5),size=100000)
a4 = np.random.normal(loc=-2,scale=np.sqrt(0.5),size=100000)
```

In [32]:
```python
import matplotlib.pyplot as plt
# %matplotlib inline
plt.figure(figsize=(8,5))
plt.hist(a1,density=True,bins=100,color='blue',alpha=0.9)
plt.hist(a2,density=True,bins=100,color='red',alpha=0.9)
plt.hist(a3,density=True,bins=100,color='orange',alpha=0.9)
plt.hist(a4,density=True,bins=100,color='green',alpha=0.9)
plt.xlim(-7,7)
plt.show()
```



Generate dice throws and average them to show the emergence of Normality as per the Central Limit Theorem

We can use either `np.random.uniform` or `np.random.randint` to generate dice throws uniformly randomly

In [33]:
```python
np.random.uniform(low=1.0,high=7.0,size=10)
```

Out[33]:
```
array([4.59197533, 6.86577467, 1.96279044, 1.17193181, 3.11299373,
       5.9646086 , 3.61531241, 1.22690631, 3.8700939 , 6.14118701])
```

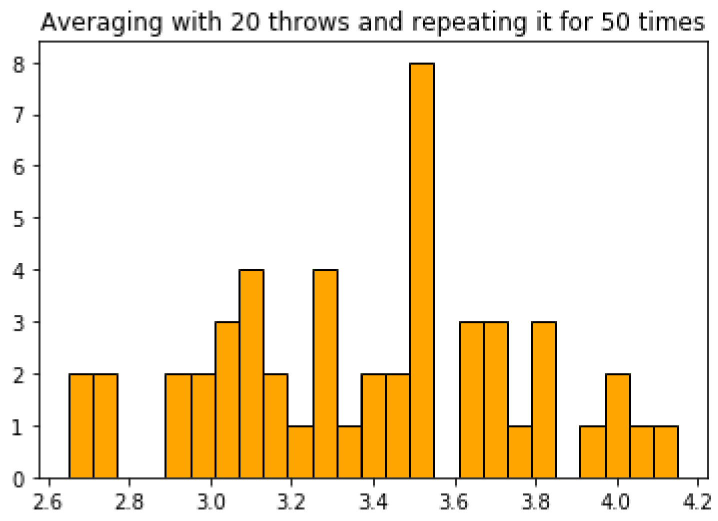
```
In [34]: def dice_throws(num_sample):
             int_throws = np.vectorize(int)
             throws = int_throws(np.random.uniform(low=1.0,high=7.0,size=num_sample))
             return throws
```

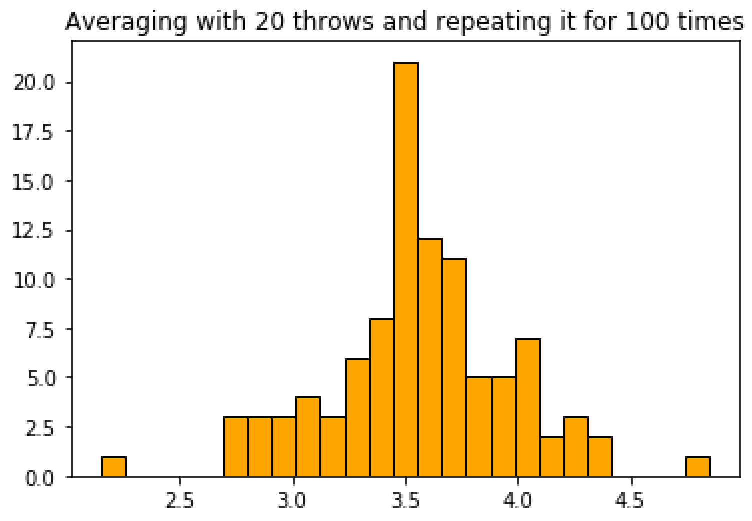
```
In [35]: dice_throws(5)

Out[35]: array([6, 2, 3, 4, 5])
```

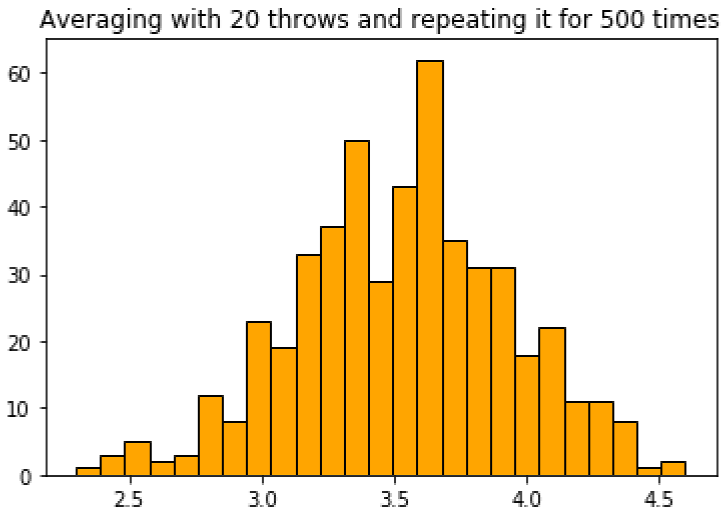
```
In [36]: np.random.randint(1,7,5)

Out[36]: array([6, 6, 4, 6, 4])
```

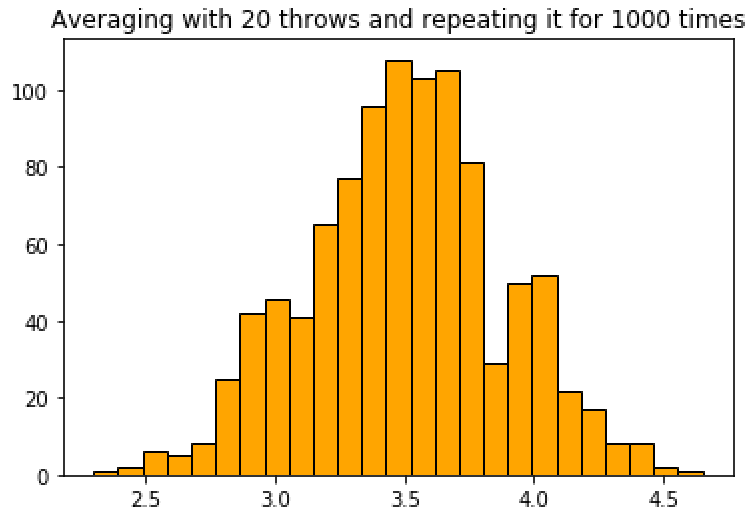
```
In [37]: def average_throws(num_throws=5,num_experiment=100):
             averages = []
             for i in range(num_experiment):
                 a = dice_throws(num_throws)
                 av = a.mean()
                 averages.append(av)
             return np.array(averages)
```

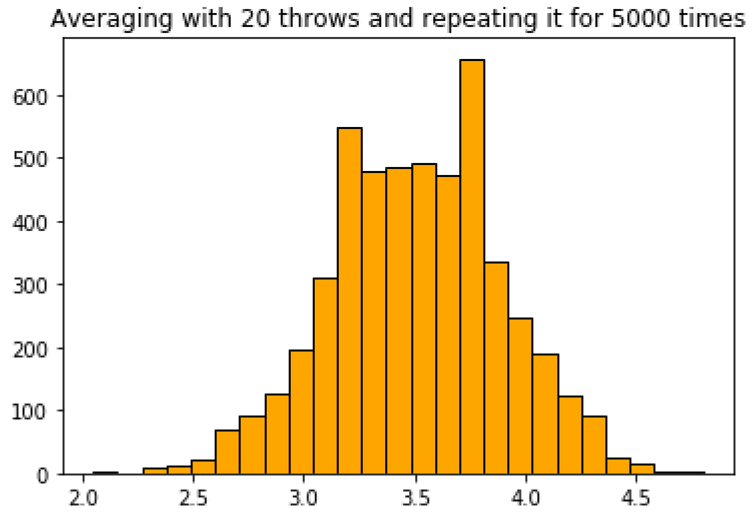
In [38]:
```
for i in [50,100,500,1000,5000,10000,50000,100000]:
    plt.hist(average_throws(num_throws=20,num_experiment=i),bins=25,edgecolor
    plt.title(f"Averaging with 20 throws and repeating it for {i} times")
    plt.show()
```

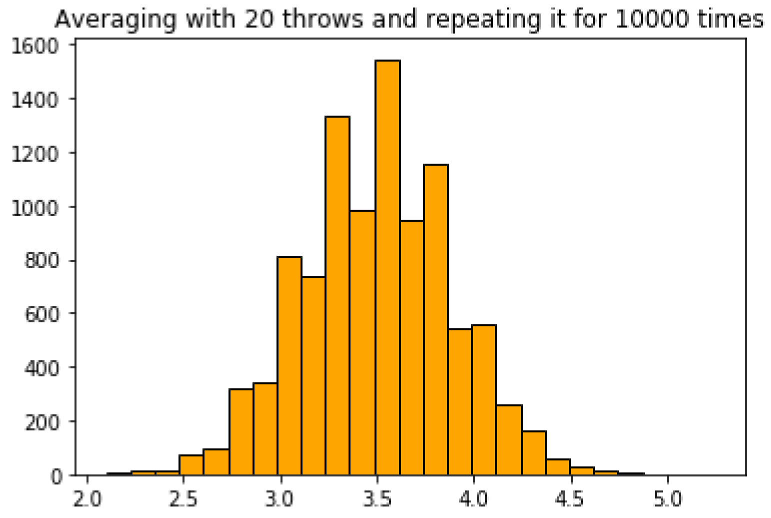


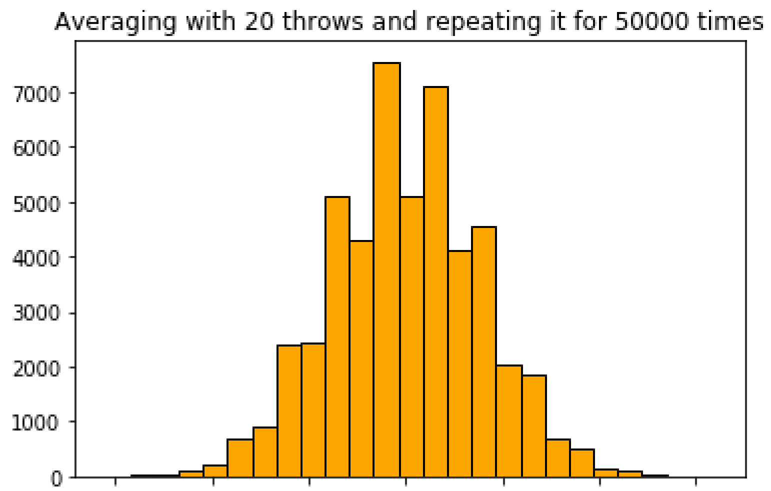


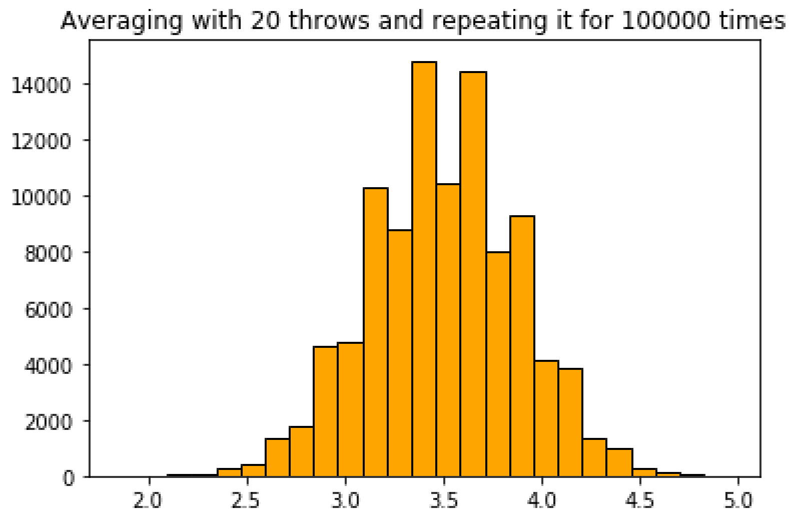



Averaging with 20 throws and repeating it for 1000 times
0
20
40
60
80
100
2.5
3.0
3.5
4.0
4.5


Averaging with 20 throws and repeating it for 5000 times
0
100
200
300
400
500
600
2.0
2.5
3.0
3.5
4.0
4.5


Averaging with 20 throws and repeating it for 10000 times
0
200
400
600
800
1000
1200
1400
1600
2.0
2.5
3.0
3.5
4.0
4.5
5.0





Central Limit Theorem

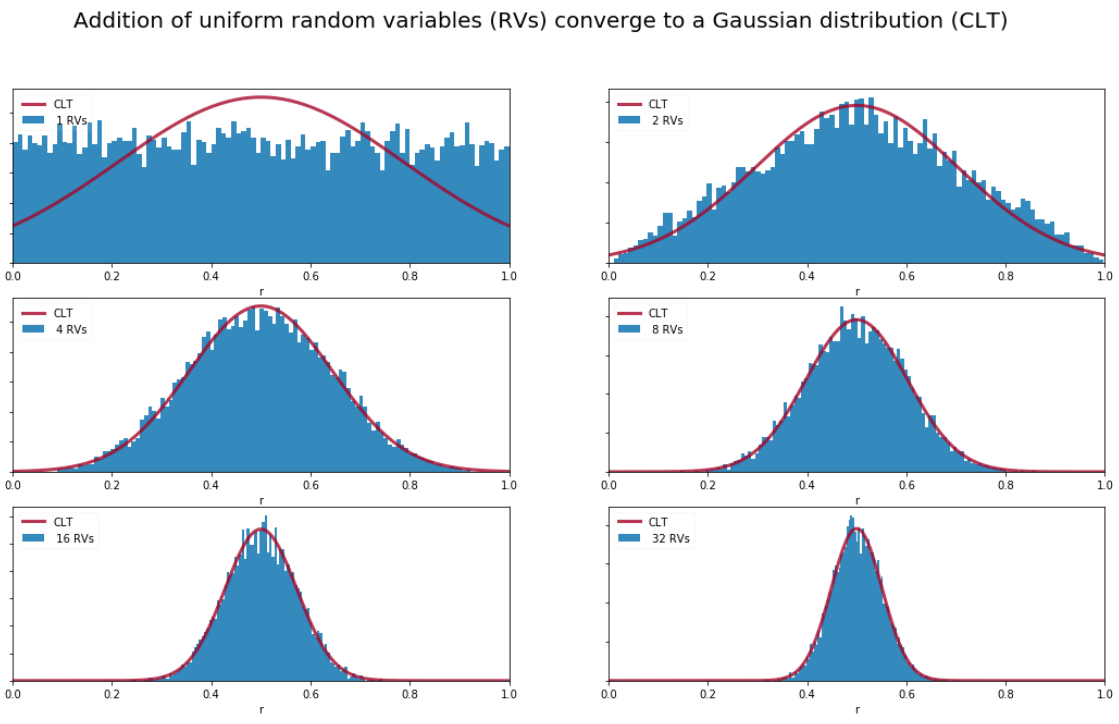
- CLT is a statistical theory stating that given a sufficiently large sample size from a population with a finite level of variance, the mean of all samples from the same population will be approximately equal to the mean of the population.
- The central limit theorem (CLT) states that the distribution of sample means approximates a normal distribution as the sample size gets larger.
  - Sample sizes equal to or greater than 30 are considered sufficient for the CLT to hold.
  - A key aspect of CLT is that the average of the sample means and standard deviations will equal the population mean and standard deviation.
  - A sufficiently large sample size can predict the characteristics of a population accurately.

** The Central Limit Theorem is used to help us understand the following facts regardless of whether the population distribution is normal or not:

1. the mean of the sample means is the same as the population mean
2. the standard deviation of the sample means is always equal to the standard error.
3. the distribution of sample means will become increasingly more normal as the sample size increases.**

```
In [39]:
f = plt.figure(figsize=(18, 10))

def plotHist(nr, N, n_, mean, var0, x0):
    ''' plots the RVs'''
    x = np.zeros((N))
    sp = f.add_subplot(3, 2, n_ )

    for i in range(N):
        for j in range(nr):
            x[i] += np.random.random()
        x[i] *= 1/nr
    plt.hist(x, 100, normed=True, color='#348ABD', label=" %d RVs"%(nr));
    plt.setp(sp.get_yticklabels(), visible=False)

    variance = var0/nr
    fac = 1/np.sqrt(2*np.pi*variance)
    dist = fac*np.exp(-(x0-mean)**2/(2*variance))
    plt.plot(x0,dist,color='#A60628',linewidth=3,label='CLT',alpha=0.8)
    plt.xlabel('r')
    plt.xlim([0, 1])
    leg = plt.legend(loc="upper left")
    leg.get_frame().set_alpha(0.1)


N = 10000   # number of samples taken
nr = ([1, 2, 4, 8, 16, 32])

mean, var0 = 0.5, 1.0/12  # mean and variance of uniform distribution in rang
x0 = np.linspace(0, 1, 128)

for i in range(np.size(nr)):
    plotHist(nr[i], N, i+1, mean, var0, x0)

plt.suptitle("Addition of uniform random variables (RVs) converge to a Gaussi
```

```
C:\Users\Meghaa\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: Matpl
otlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in
3.1. Use 'density' instead.
  if sys.path[0] == '':
```



Conditional Probability:

Conditional probability is the probability of one event occurring with some relationship to one or more other events. For example:

Event A is that it is raining outside, and it has a 0.3 (30%) chance of raining today. Event B is that you will need to go outside, and that has a probability of 0.5 (50%). A conditional probability would look at these two events in relationship with one another, such as the probability that it is both raining and you will need to go outside.

P(B|A) = P(A and B) / P(A) which you can also rewrite as: P(B|A) = P(A∩B) / P(A)

In a group of 100 sports car buyers, 40 bought alarm systems, 30 purchased bucket seats, and 20 purchased an alarm system and bucket seats. If a car buyer chosen at random bought an alarm system, what is the probability they also bought bucket seats?

Figure out P(A). It’s given in the question as 40%, or 0.4.

Figure out P(A∩B). This is the intersection of A and B: both happening together. It’s given in the question 20 out of 100 buyers, or 0.2.

Insert your answers into the formula: P(B|A) = P(A∩B) / P(A) = 0.2 / 0.4 = 0.5.

The probability that a buyer bought bucket seats, given that they purchased an alarm system, is 50%.

Inference and Descriptive Statistics

Descriptive statistics is the term given to the analysis of data that helps describe, show or summarize data in a meaningful way such that, for example, patterns might emerge from the data. Descriptive statistics do not, however, allow us to make conclusions beyond the data we have analysed or reach conclusions regarding any hypotheses we might have made. They are simply a way to describe our data.

- Measure of Central Tendency - Mode, Mean and Mode
- Measure of dispersion - SD, Variance
- Measure of Skewness

Inferential statistics makes inferences about the population based on random sample data taken from a population. We use a smaller set of random data compiled from a population to extend and make conclusions or generalize about the population.

- Population - Entire dataset
- Sample - portion of a dataset

The most common methodologies in inferential statistics are hypothesis tests, t and z statistics, confidence intervals, and regression analysis. Interestingly, these inferential methods can produce similar summary values as descriptive statistics, such as the mean and standard deviation.

- https://statisticsbyjim.com/basics/descriptive-inferential-statistics/ (https://statisticsbyjim.com/basics/descriptive-inferential-statistics/)

In [40]:
```
import pandas as pd
import matplotlib.pyplot
movies = pd.read_csv('http://bit.ly/imdbratings')
```

In [41]:
```
movies.head()
```

Out[41]:

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

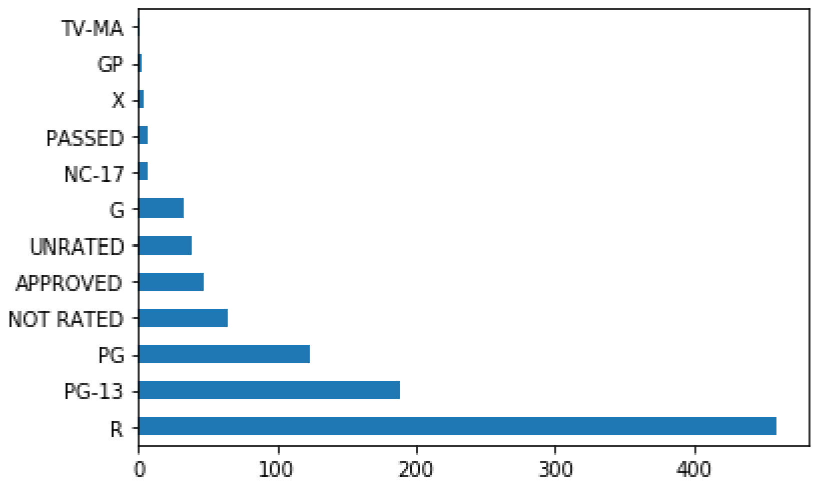
```
In [42]:  movies.describe()
```

Out[42]:

	star_rating	duration
count	979.000000	979.000000
mean	7.889785	120.979571
std	0.336069	26.218010
min	7.400000	64.000000
25%	7.600000	102.000000
50%	7.800000	117.000000
75%	8.100000	134.000000
max	9.300000	242.000000

```
In [43]:  movies.content_rating.value_counts().plot(kind='barh')
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1b426cf95c8>
```



In [44]:
```
movies.describe(include='all')
```

Out[44]:

	star_rating	title	content_rating	genre	duration	actors_list
count	979.000000	979	976	979	979.000000	979
unique	NaN	975	12	16	NaN	969
top	NaN	Dracula	R	Drama	NaN	[u'Daniel Radcliffe', u'Emma Watson', u'Rupert...
freq	NaN	2	460	278	NaN	6
mean	7.889785	NaN	NaN	NaN	120.979571	NaN
std	0.336069	NaN	NaN	NaN	26.218010	NaN
min	7.400000	NaN	NaN	NaN	64.000000	NaN
25%	7.600000	NaN	NaN	NaN	102.000000	NaN
50%	7.800000	NaN	NaN	NaN	117.000000	NaN
75%	8.100000	NaN	NaN	NaN	134.000000	NaN
max	9.300000	NaN	NaN	NaN	242.000000	NaN

What are the similarities between descriptive and inferential statistics? Both descriptive and inferential statistics rely on the same set of data. Descriptive statistics rely solely on this set of data, whilst inferential statistics also rely on this data in order to make generalisations about a larger population.

What are the strengths of using descriptive statistics to examine a distribution of scores? Other than the clarity with which descriptive statistics can clarify large volumes of data, there are no uncertainties about the values you get (other than only measurement error, etc.).

What are the limitations of descriptive statistics? Descriptive statistics are limited in so much that they only allow you to make summations about the people or objects that you have actually measured. You cannot use the data you have collected to generalize to other people or objects (i.e., using data from a sample to infer the properties/parameters of a population). For example, if you tested a drug to beat cancer and it worked in your patients, you cannot claim that it would work in other cancer patients only relying on descriptive statistics (but inferential statistics would give you this opportunity).

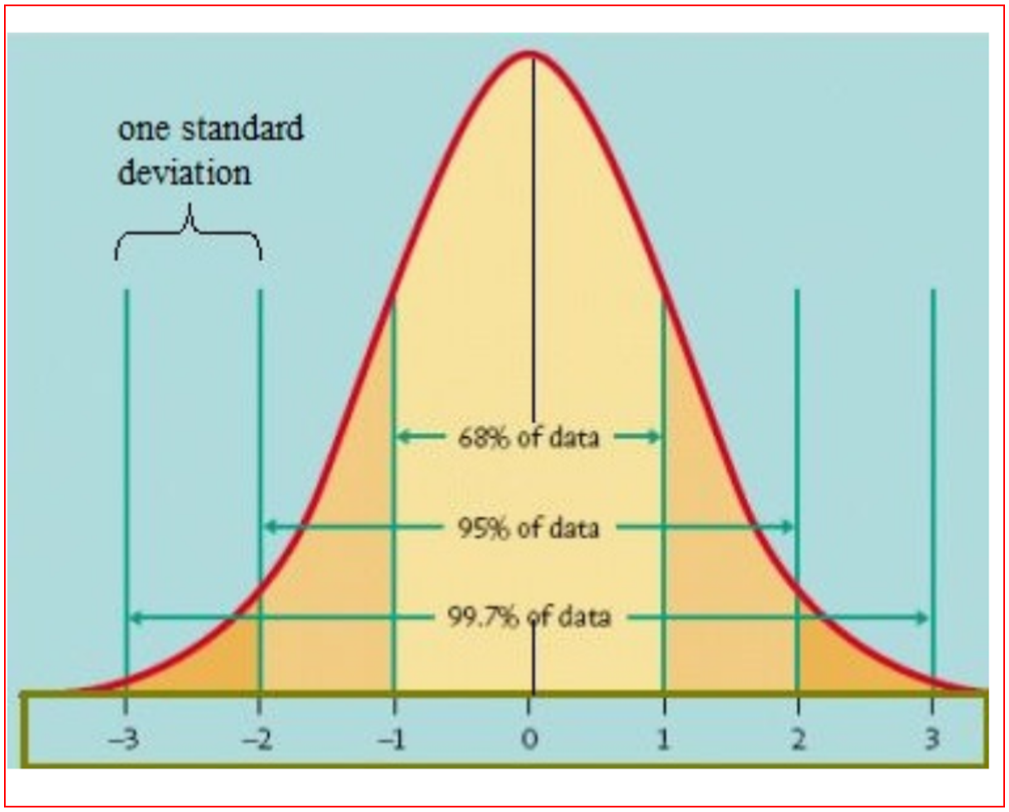
Differential inferential statistics techniques are

- Z Statistics
- Hypothesis testing
- T Statistics
- Normal distribution
- Central Limit Theorem
- Confidence Interval

- ANOVA
- R²
- Regression

Z Statistics

- Z statistics uses z score
- Z Score tells how many standard deviation above or below a datapoint is from the population mean.
- Z score is also known as standard score as it can be placed on a normal distribution curve.
- Z scores range from -3 standard deviation to +3 standard deviation. A negative value for Z score implies data point is below the mean and a positive value for Z score implies data point is above the mean
- Z score tells the probability of a data point occurring on normal distribution curve
- Helps compare two scores from different normal distribution
- Useful when comparing data points from two different sets of data



Formula for Z score = (Observation — Mean)/Standard Deviation

z = (X — μ) / σ - for entire population

z = [(x-μ)/ s/ sqrt (no. of samples)] - for sample population

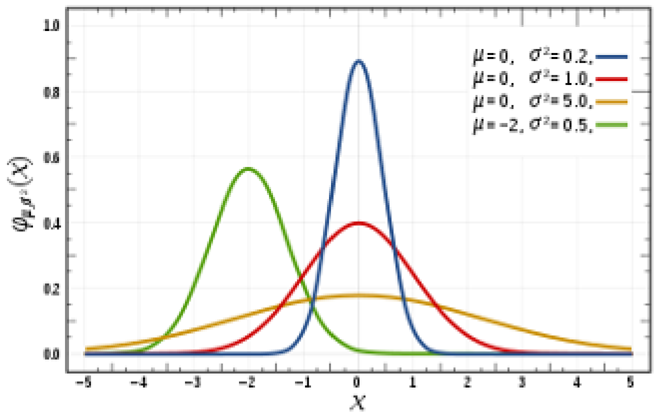
s - standard deviation of the samples

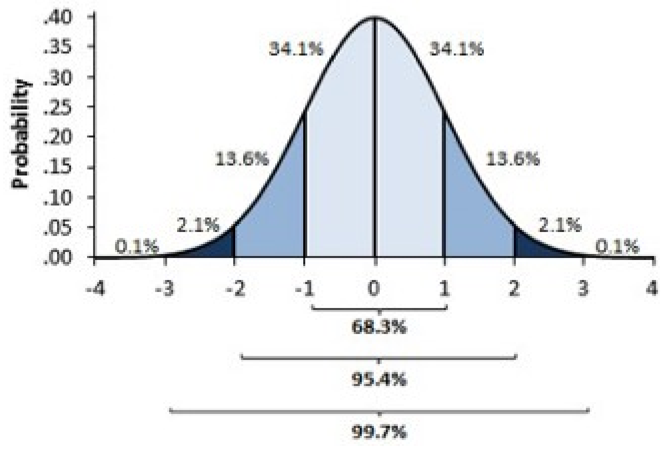
What is hypothesis testing ?

Hypothesis testing is a statistical method that is used in making statistical decisions using experimental data. Hypothesis Testing is basically an assumption that we make about the population parameter.

Hypothesis testing is an essential procedure in statistics. A hypothesis test evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data. When we say that a finding is statistically significant, it’s thanks to a hypothesis test.

The basic of hypothesis is normalisation and standard normalisation. all our hypothesis is revolve around basic of these 2 terms. let’s see these.





Normalization

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

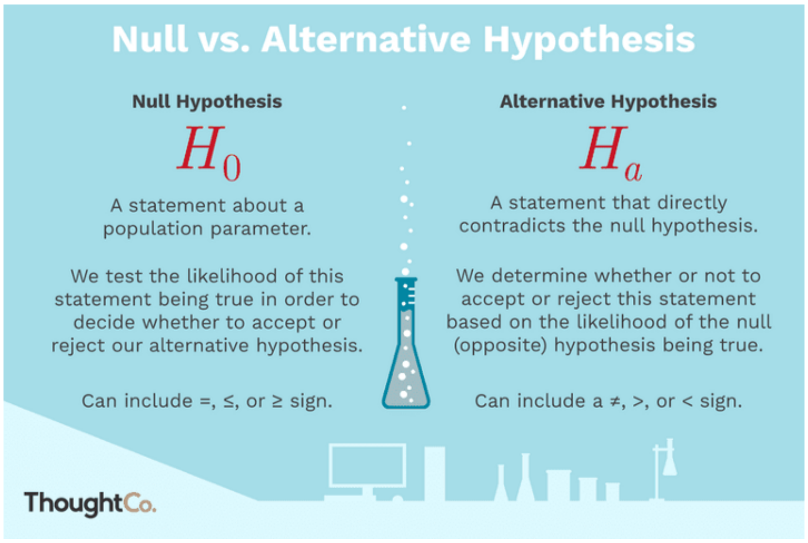
Standard Normalization

x_new = (X — μ) / σ

Steps for performing Hypothesis testing

- Step 1: we first state the null hypothesis, which is the accepted fact currently
- Step 2: we state the alternate hypothesis that we are trying to prove if it is right or wrong
- Step 3: select a significance level denoted by α, usually it is .05 but we can select .02 or .01 whichever seems right for our testing
- Step 4: Find the right test, could be z statistics, or t statistics, we will assign this to p value
- Step 5: Compare p value with α and decide if we need to reject the null hypothesis or support null hypothesis

Null Hypothesis Vs Alternative Hypotheis


Null vs. Alternative Hypothesis
Null Hypothesis
H_0
A statement about a population parameter.
We test the likelihood of this statement being true in order to decide whether to accept or reject our alternative hypothesis.
Can include =, ≤, or ≥ sign.
Alternative Hypothesis
H_a
A statement that directly contradicts the null hypothesis.
We determine whether or not to accept or reject this statement based on the likelihood of the null (opposite) hypothesis being true.
Can include a ≠, >, or < sign.
ThoughtCo.

Null Hypothesis is usually the accepted or know fact or a default position like there is no relationship between dependent and independent variable.

Null Hypothesis is usually the hypothesis that the experiment is trying to disapprove and is currently an accepted fact. It is the status quo and hence includes only equality operator.

Alternate hypothesis is the hypothesis that we are trying to prove with our experiment or survey. It can be stated with an inequality operator, less than or greater than operator Null and alternate hypothesis should always be mutually exclusive.

Step 1 & 2

- Media time has any impact on a student’s CAT score. Here we want to investigate if a student spending less media time will increase their CAT score.
- Assumption that media time has no impact on student’s CAT score and that becomes our null hypothesis.
- Alternate hypothesis will be, media time has an impact on the student’s CAT score.

Step 3

- **Confidence level (C)** is the percentage of time a statistical result would be correct if you took numerous random sample. It is represented by C
- **Level of significance(α)**: Refers to the degree of significance in which we accept or reject the null-hypothesis.

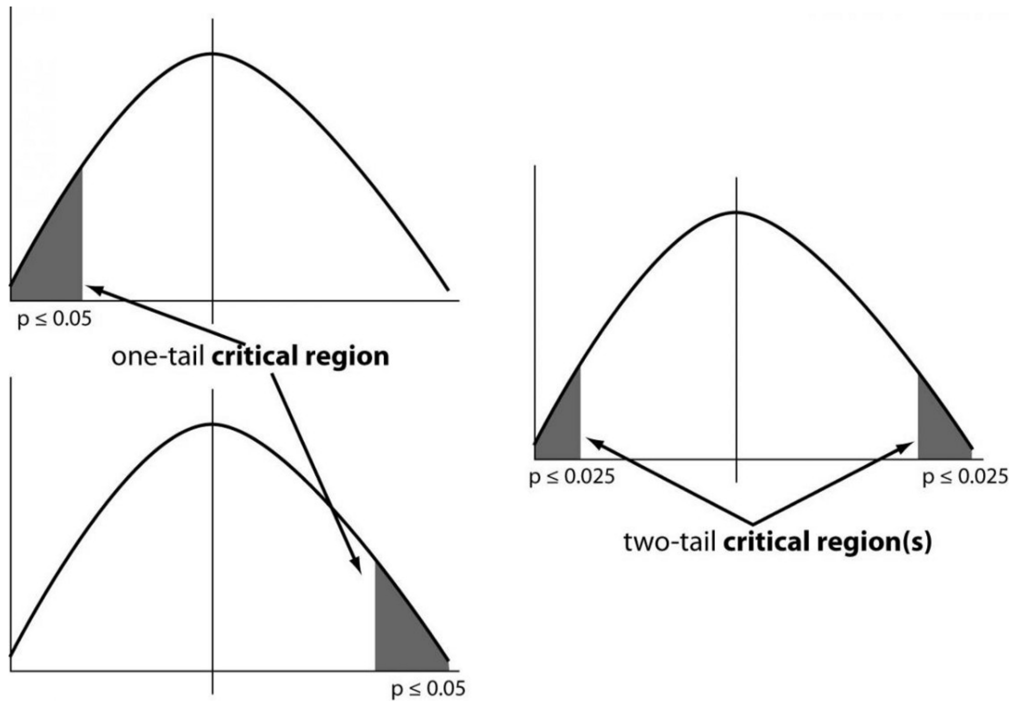
- **α=1-C**

Critical regions describe the entire area of values that indicate you reject the null hypothesis.

100% accuracy is not possible for accepting or rejecting a hypothesis, so we therefore select a level of significance that is usually 5%.

One tailed test :- A test of a statistical hypothesis , where the region of rejection is on only one side of the sampling distribution , is called a one-tailed test. Example :- a college has ≥ 4000 student

Two-tailed test :- A two-tailed test is a statistical test in which the critical area of a distribution is two-sided and tests whether a sample is greater than or less than a certain range of values. If the sample being tested falls into either of the critical areas, the alternative hypothesis is accepted instead of the null hypothesis. Example : a college != 4000 student



Step - 4

Z- Test This test is only appropriate when you know the true mean and standard deviation of the population.

z-Test

Formula to find the value of Z (z-test) Is:

$$Z = \frac{\overline{x} - \mu_0}{\sigma / \sqrt{n}}$$

$\bar{x}$ = mean of sample
μ_0 = mean of population
σ = standard deviation of population
n = no. of observations

- **Hypothesis Tests When You Don't Know Your Population Parameters**

The Student's t-distribution is similar to the normal distribution, except it is more spread out and wider in appearance, and has thicker tails. The differences between the t-distribution and the normal distribution are more exaggerated when there are fewer data points, and therefore fewer degrees of freedom.

Degrees of Freedom

Degrees of freedom is a combination of how much data you have and how many parameters you need to estimate. It indicates how much independent information goes into a parameter estimate. The mean of 1,2,3,4 and 5 is 3, and it is based on 5 values. So, we know that the values must sum to 15 based on the equation for the mean. So with 4 values and mean - we will be able to estimate the 5 value.. So degree of freedom in this case is 4

t-Test

$$t = \frac{\bar{x} - \mu_0}{\frac{s}{\sqrt{n}}}$$

where:

t is the test statistic and has $n - 1$ degrees of freedom.

$\bar{x}$ is the sample mean

μ_0 is the population mean under the null hypothesis.

s is the sample standard deviation

n is the sample size

$\frac{s}{\sqrt{n}}$ is the estimated standard error

```
In [45]:  import pandas as pd
          from scipy import stats
          from statsmodels.stats import weightstats as stests
          df = pd.read_csv('blood_pressure.csv')
          ztest ,pval = stests.ztest(df['bp_before'], x2=None, value=156)
          print(float(pval))
          if pval<0.05:
              print("reject null hypothesis")
          else:
              print("accept null hypothesis")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-45-e8e9dec43c1c> in <module>
      2 from scipy import stats
      3 from statsmodels.stats import weightstats as stests
----> 4 df = pd.read_csv('blood_pressure.csv')
      5 ztest ,pval = stests.ztest(df['bp_before'], x2=None, value=156)
      6 print(float(pval))

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(filepath_or_
buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix,
mangle_dupe_cols, dtype, engine, converters, true_values, false_values, ski
pinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_
filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep
_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compres
sion, thousands, decimal, lineterminator, quotechar, quoting, doublequote,
 escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, d
elim_whitespace, low_memory, memory_map, float_precision)
    683         )
    684 
--> 685         return _read(filepath_or_buffer, kwds)
    686 
    687     parser_f.__name__ = name

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buf
fer, kwds)
    455 
    456     # Create the parser.
--> 457     parser = TextFileReader(fp_or_buf, **kwds)
    458 
    459     if chunksize or iterator:

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, eng
ine, **kwds)
    893             self.options["has_index_names"] = kwds["has_index_name
s"]
    894 
--> 895         self._make_engine(self.engine)
    896 
    897     def close(self):

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, en
gine)
   1133     def _make_engine(self, engine="c"):
   1134         if engine == "c":
```

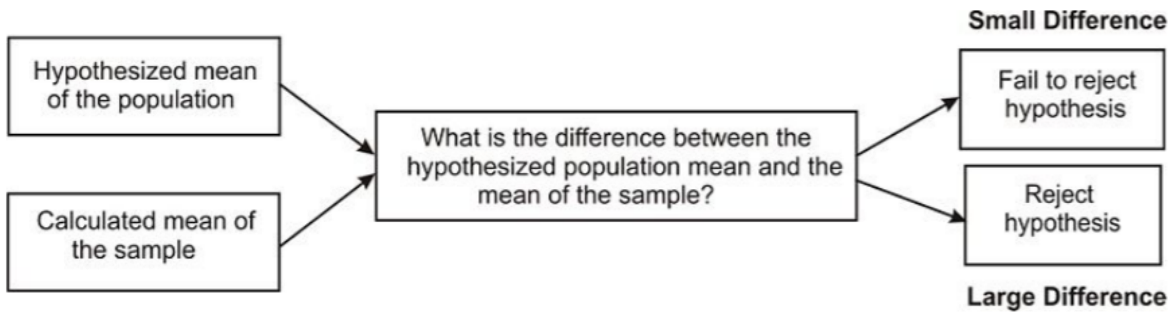
```
-> 1135                 self._engine = CParserWrapper(self.f, **self.options)
   1136             else:
   1137                 if engine == "python":

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, *
*kwds)
   1915             kwds["usecols"] = self.usecols
   1916 
-> 1917             self._reader = parsers.TextReader(src, **kwds)
   1918             self.unnamed_cols = self._reader.unnamed_cols
   1919 

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_s
ource()

FileNotFoundError: [Errno 2] File b'blood_pressure.csv' does not exist: b'b
lood_pressure.csv'
```

Step - 4 P- Value

- The P value, or calculated probability, is the probability of finding the observed
  - If your P value is less than the chosen significance level then you reject the null hypothesis i.e. accept that your sample gives reasonable evidence to support the alternative hypothesis. It does NOT imply a “meaningful” or “important” difference; that is for you to decide when considering the real-world relevance of your result.

if p <= α then reject the null hypothesis

if p > α then fail to reject the null hypothesis so we support the null hypothesis


Hypothesized mean of the population
Calculated mean of the sample
What is the difference between the hypothesized population mean and the mean of the sample?
Small Difference
Fail to reject hypothesis
Reject hypothesis
Large Difference

.

Hypothesis Testing for Categorical variables

Chi-square test is used for categorical data and it can be used to estimate how closely the distribution of a categorical variable matches an expected distribution (the goodness-of-fit test), or to estimate whether two categorical variables are independent of one another (the test of independence).

$$\chi^2 = \sum \frac{(observed - expected)^2}{expected}$$

degree of freedom (df) = (rows−1)(columns−1)

In [ ]:
```
df_chi = pd.read_csv('chi-test.csv')
df_chi
```

In [ ]:
```
contingency_table=pd.crosstab(df_chi["Gender"],df_chi["Like Shopping?"])
contingency_table
```

In [ ]:

```
contingency_table=pd.crosstab(df_chi["Gender"],df_chi["Like Shopping?"])
print('contingency_table :-\n',contingency_table)
#Observed Values
Observed_Values = contingency_table.values
print("Observed Values :-\n",Observed_Values)
b=stats.chi2_contingency(contingency_table)
Expected_Values = b[3]
print("Expected Values :-\n",Expected_Values)
no_of_rows=len(contingency_table.iloc[0:2,0])
no_of_columns=len(contingency_table.iloc[0,0:2])
ddof=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",ddof)
alpha = 0.05
from scipy.stats import chi2
chi_square=sum([(o-e)**2./e for o,e in zip(Observed_Values,Expected_Values)]
chi_square_statistic=chi_square[0]+chi_square[1]
print("chi-square statistic:-",chi_square_statistic)
critical_value=chi2.ppf(q=1-alpha,df=ddof)
print('critical_value:',critical_value)
#p-value
p_value=1-chi2.cdf(x=chi_square_statistic,df=ddof)
print('p-value:',p_value)
print('Significance level: ',alpha)
print('Degree of Freedom: ',ddof)
print('chi-square statistic:',chi_square_statistic)
print('critical_value:',critical_value)
print('p-value:',p_value)
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables'
else:
    print("Retain H0,There is no relationship between 2 categorical variables

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables'
else:
    print("Retain H0,There is no relationship between 2 categorical variables
```

In [ ]:

```
(4*4)/9
```

In [ ]:

```
contingency_table
```

In [ ]:

```
20/9
```

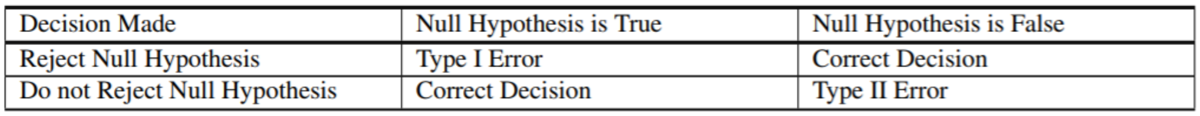
In [ ]:

```

```

In [ ]:

```
Row total = 2+3
Col total = 4
5*4 / = 20/9

Row total = 5
Col total = 5
25/9
```

Expected Cell Frequency = (Row Total * Column Total) / N



Decision Made	Null Hypothesis is True	Null Hypothesis is False
Reject Null Hypothesis	Type I Error	Correct Decision
Do not Reject Null Hypothesis	Correct Decision	Type II Error