Probability

In [2]:
```
import numpy as np
import pandas as pd
import os
```

Example1: There are 52 cards In a standard deck of cards and of those 52 cards, 4 are Aces. What's the probability of drawing an Ace.

In [3]:
```
# Sample Space
cards = 52

# Outcomes
aces = 4

# Divide possible outcomes by the sample set
ace_probability = aces / cards

# Print probability rounded to two decimal places
print(round(ace_probability, 2))

# Ace Probability Percent Code
ace_probability_percent = ace_probability * 100

# Print probability percent rounded to one decimal place
print(str(round(ace_probability_percent, 0)) + '%')
```

```
0.08
8.0%
```

Example2: What is the probability of drawing a card that is a Heart, a face card (such as Jacks, Queens, or Kings), or a combination of both, such as a Queen of Hearts.

In [4]:
```
# Create function that returns probability percent rounded to one decimal place
def event_probability(event_outcomes, sample_space):
    probability = (event_outcomes / sample_space) * 100
    return round(probability, 1)

# Sample Space
cards = 52

# Determine the probability of drawing a heart
hearts = 13
heart_probability = event_probability(hearts, cards)

# Determine the probability of drawing a face card
face_cards = 12
face_card_probability = event_probability(face_cards, cards)

# Determine the probability of drawing the queen of hearts
queen_of_hearts = 1
queen_of_hearts_probability = event_probability(queen_of_hearts, cards)

# Print each probability
print("Probability of Heart :- ",str(heart_probability) + '%')
print("Probability of Face Card :- ",str(face_card_probability) + '%')
print("Probability of Queen of Hearts :- ",str(queen_of_hearts_probability) + '%')
```

```
Probability of Heart :-  25.0%
Probability of Face Card :-  23.1%
Probability of Queen of Hearts :-  1.9%
```

Permutations

Consider another example with Aces. There are four Aces in a deck of cards, and these are all the different combinations of pocket Aces;

a). Ace Hearts / Ace Diamonds

b). Ace Hearts / Ace Clubs

c). Ace Hearts / Ace Spades

d). Ace Diamonds / Ace Clubs

e). Ace Diamonds / Ace Spades

f). Ace Clubs / Ace Spades

There are six combinations of pocket Aces. To find the number of combinations, you first must find the number of permutations:

In [5]:
```
# Permutations Code
import math
n = 4
k = 2

# Determine permutations and print result
Permutations = math.factorial(n) / math.factorial(k)
print(Permutations)
```

```
12.0
```

Find out Permutations

In [7]:
```
# A Python program to print all
# permutations using library function
from itertools import permutations

perm = permutations([1, 2, 3])

# Print the obtained permutations
for i in list(perm):
    print (i)
```

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

In [8]:
```
# A Python program to print all
# permutations of given length
from itertools import permutations

# Get all permutations of length 2
# and length 2
perm = permutations([1, 2, 3], 2)

# Print the obtained permutations
for i in list(perm):
    print (i)
```

```
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

Combinations

In [6]:
```
# Combinations Code
n = 52
k = 2

# Determine Permutations
Permutations = math.factorial(n) / math.factorial(n - k)

# Determine Combinations and print result
Combinations = Permutations / math.factorial(k)
print(Combinations)
```

```
1326.0
```

Find out all Combinations

In [9]:
```
# A Python program to print all
# combinations of given length
from itertools import combinations

# Get all combinations of [1, 2, 3]
# and length 2
comb = combinations([1, 2, 3], 2)

# Print the obtained combinations
for i in list(comb):
    print (i)
```

```
(1, 2)
(1, 3)
(2, 3)
```

In [10]:
```
# A Python program to print all
# combinations of a given length
from itertools import combinations

# Get all combinations of [1, 2, 3]
# and length 2
comb = combinations([1, 2, 3], 2)

# Print the obtained combinations
for i in list(comb):
    print (i)
```

```
(1, 2)
(1, 3)
(2, 3)
```

In [11]:
```
# A Python program to print all combinations
# with an element-to-itself combination is
# also included
from itertools import combinations_with_replacement

# Get all combinations of [1, 2, 3] and length 2
comb = combinations_with_replacement([1, 2, 3], 2)

# Print the obtained combinations
for i in list(comb):
    print (i)
```

```
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)
```

Discrete and continuous probabilities

Discrete Probabilityies

By a discrete random variable, it is meant a function (or a mapping), say X, from a sample space Ω, into the set of real numbers. Symbolically, if ω ∈Ω, then X (ω) = x, where x is a real number.

A random variable X is a **discrete random variable** if:

- there are a finite number of possible outcomes of X, or
- there are a countably infinite number of possible outcomes of X.

A countably infinite number of possible outcomes means that there is a one-to-one correspondence between the outcomes and the set of integers.

No such one-to-one correspondence exists for an uncountably infinite number of possible outcomes.

For a value x of the set of possible outcomes of the random variable X , i.e., x ∈ T , p(x) denotes the probability that random variable X has the outcome x.

For discrete random variables, this is written as P (X = x), which is known as the probability mass function. The pmf is often referred to as the distribution". For continuous variables, p(x) is called the probability density function (often referred to as a density).

```
Suppose we want to calculate the probability of getting heads or tails when flipping a fair coin. We can represent this experiment with a Bernoulli distribution,
which is a type of discrete probability distribution. Here is an example code to calculate the probability
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import random

def get_frequencies(values):
    frequencies = {}
    for v in values:
        if v in frequencies:
            frequencies[v] += 1
        else:
            frequencies[v] = 1
    return frequencies
```

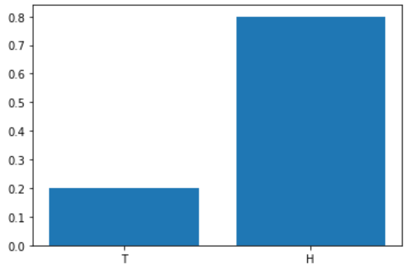
In [3]:
```
# Suppose we want to calculate the probability of getting heads or tails when flipping a fair coin.
# We can represent this experiment with a Bernoulli distribution, which is a type of discrete probability distribution.
# Here is an example code to calculate the probability

def get_probabilities(sampledata, freqs):
    probabilities = []
    for k, v in freqs.items():
        probabilities.append(round(v / len(sampledata), 1))
    return probabilities

sample = ["H"] * 1 + ['T'] * 5
print('sample', sample)
calculated_frequencies = get_frequencies(sample)
print(calculated_frequencies)
calculate_probabilities = get_probabilities(sample, calculated_frequencies)
print("prob", calculate_probabilities)
x_axis = list(set(sample))

plt.bar(x_axis, calculate_probabilities)
plt.show()
```

```
sample ['H', 'T', 'T', 'T', 'T', 'T']
{'H': 1, 'T': 5}
prob [0.2, 0.8]
```



Continuous Probabilities

Continuous probability distribution: A probability distribution in which the random variable X can take on any value (is continuous). Because there are infinite values that X could assume, the probability of X taking on any one specific value is zero. Therefore we often speak in ranges of values (p(X>0) = .50). The normal distribution is one example of a continuous distribution. The probability that X falls between two values (a and b) equals the integral (area under the curve) from a to b:

Probability Density Function

$$F(x) = P(a \leq x \leq b) = \int_a^b f(x)dx \geq 0$$

Normal Distribution

In statistics, a **normal distribution** or **Gaussian distribution** is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is

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

In [8]:

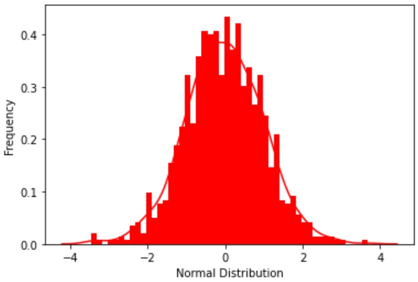
```
# import packages
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt


# generate data
data =stats.norm(scale=1, loc=0).rvs(1000)

# plotting a histogram
ax = sns.distplot(data,
                  bins=50,
                  kde=True,
                  color='red',
                  hist_kws={"linewidth": 15,'alpha':1})
ax.set(xlabel='Normal Distribution', ylabel='Frequency')

plt.show()
```

```
D:\anacondasucks\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Pleas
e adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Binomial Distribution

Under a given set of factors or assumptions, the binomial distribution expresses the likelihood that a variable will take one of two outcomes or independent values. ex: if an experiment is successful or a failure. if the answer for a question is “yes” or “no” etc… . np.random.binomial() is used to generate binomial data. n refers to a number of trails and prefers the probability of each trail.

In [10]:

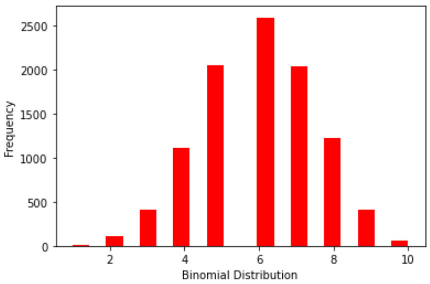
```
# import packages
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# generate data
# n== number of trials,p== probability of each trial
n, p = 10, .6
data = np.random.binomial(n, p, 10000)

# plotting a histogram
ax = sns.distplot(data,
                  bins=20,
                  kde=False,
                  color='red',
                  hist_kws={"linewidth": 15, 'alpha': 1})
ax.set(xlabel='Binomial Distribution', ylabel='Frequency')

plt.show()
```

```
D:\anacondasucks\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Pleas
e adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



poisson Distribution

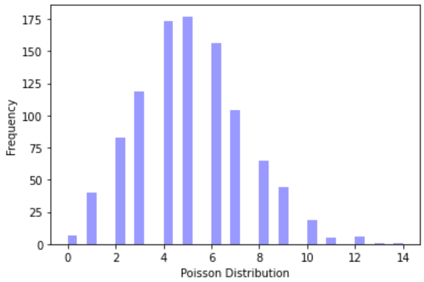
A Poisson distribution is a kind of probability distribution used in statistics to illustrate how many times an event is expected to happen over a certain amount of time. It’ s also called count distribution. np.random.poisson function() is used to create data for poisson distribution. lam refers to The number of occurrences that are expected to occur in a given time frame.

In [12]:
```
# import packages
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# generate poisson data
poisson_data = np.random.poisson(lam=5, size=1000)

# plotting a histogram
ax = sns.distplot(poisson_data,
                  kde=False,
                  color='blue')
ax.set(xlabel='Poisson Distribution', ylabel='Frequency')

plt.show()
```



In [ ]:

Sum rule, product rule, and Bayes' theorem

Sum Rule

The addition rule states the probability of two events is the sum of the probability that either will happen minus the probability that both will happen.

The addition rule is: $P(A \cup B)=P(A)+P(B)-P(A \cap B)$

Example: If we have two events A and B, with probabilities P(A) = 0.4 and P(B) = 0.3, and the probability of their intersection is P(A and B) = 0.1, we can calculate the probability of their union as follows:

In [14]:
```
def sum_rule(p_a, p_b, p_a_and_b):
    p_a_or_b = p_a + p_b - p_a_and_b
    return p_a_or_b

# Example usage
p_a = 0.4
p_b = 0.3
p_a_and_b = 0.1
p_a_or_b = sum_rule(p_a, p_b, p_a_and_b)
print("P(A or B) =", p_a_or_b)
```

```
P(A or B) = 0.6
```

Product rule:

The product rule states that the probability of the intersection of two events is equal to the product of their individual probabilities, given that they are independent events.

Example: If we have two independent events A and B, with probabilities P(A) = 0.4 and P(B) = 0.3, we can calculate the probability of their intersection as follows:

In [15]:
```
def product_rule(p_a, p_b):
    p_a_and_b = p_a * p_b
    return p_a_and_b

# Example usage
p_a = 0.4
p_b = 0.3
p_a_and_b = product_rule(p_a, p_b)
print("P(A and B) =", p_a_and_b)
```

```
P(A and B) = 0.12
```

Bayes' theorem:

Bayes theorem helps us find conditional probability. It simply derived from the product rule.

If we rewrite the product rule in terms of P(X|Y) we would have:

$$P(Y|X) = \frac{P(X,Y)}{P(X)}$$

Now we can use the symmetry property from the product rule to replace the numerator. The we have:

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)}$$

This is the legendary Bayes' theorem!

Bayes' theorem is a way of calculating the probability of a hypothesis based on new evidence. It is based on the prior probability of the hypothesis and the likelihood of the evidence, given the hypothesis.

Example: suppose the probability of the weather being cloudy is 40%.

Also suppose the probability of rain on a given day is 20%.

Also suppose the probability of clouds on a rainy day is 85%.

If it's cloudy outside on a given day, what is the probability that it will rain that day?

P(cloudy) = 0.40 P(rain) = 0.20 P(cloudy | rain) = 0.85

In [18]:
```
def bayesTheorem(pA, pB, pBA):
    return pA * pBA / pB

#define probabilities
pRain = 0.2
pCloudy = 0.4
pCloudyRain = 0.85

#use function to calculate conditional probability
bayesTheorem(pRain, pCloudy, pCloudyRain)
```

Out[18]:
```
0.425
```

Basic Statistics with Python

Mean

In [19]:
```
# importing statistics to handle statisticaloperations
import statistics

# initializing list
li = [1, 2, 3, 3, 2, 2, 2, 1]

# using mean() to calculate average of list
# elements
print ("The average of list values is : ",end="")
print (statistics.mean(li))
```

```
The average of list values is : 2
```

Median

In [20]:
```
# importing the statistics module
from statistics import median

# Importing fractions module as fr
from fractions import Fraction as fr

# tuple of positive integer numbers
data1 = (2, 3, 4, 5, 7, 9, 11)

# tuple of floating point values
data2 = (2.4, 5.1, 6.7, 8.9)

# tuple of fractional numbers
data3 = (fr(1, 2), fr(44, 12),
        fr(10, 3), fr(2, 3))

# tuple of a set of negative integers
data4 = (-5, -1, -12, -19, -3)

# tuple of set of positive
# and negative integers
data5 = (-1, -2, -3, -4, 4, 3, 2, 1)

# Printing the median of above datasets
print("Median of data-set 1 is % s" % (median(data1)))
print("Median of data-set 2 is % s" % (median(data2)))
print("Median of data-set 3 is % s" % (median(data3)))
print("Median of data-set 4 is % s" % (median(data4)))
print("Median of data-set 5 is % s" % (median(data5)))
```

```
Median of data-set 1 is 5
Median of data-set 2 is 5.9
Median of data-set 3 is 2
Median of data-set 4 is -5
Median of data-set 5 is 0.0
```

Median Low

median_low() function returns the median of data in case of odd number of elements, but in case of even number of elements, returns the lower of two middle elements. If the passed argument is empty, StatisticsError is raised

In [22]:
```
# simple list of a set of integers
set1 = [1, 3, 3, 4, 5, 7]

# Print median of the data-set

# Median value may or may not
# lie within the data-set
print("Median of the set is % s"
    % (statistics.median(set1)))

# Print low median of the data-set
print("Low Median of the set is % s "
    % (statistics.median_low(set1)))
```

```
Median of the set is 3.5
Low Median of the set is 3
```

Median High

median_high() function returns the median of data in case of odd number of elements, but in case of even number of elements, returns the higher of two middle elements. If passed argument is empty, StatisticsError is raised.

In [23]:
```
# simple list of a set of integers
set1 = [1, 3, 3, 4, 5, 7]

# Print median of the data-set

# Median value may or may not
# lie within the data-set
print("Median of the set is %s"
    % (statistics.median(set1)))

# Print high median of the data-set
print("High Median of the set is %s "
    % (statistics.median_high(set1)))
```

```
Median of the set is 3.5
High Median of the set is 4
```

Mode

In [27]:
```
 from statistics import mode
# Importing fractions module as fr
# Enables to calculate harmonic_mean of a
# set in Fraction
from fractions import Fraction as fr

# tuple of positive integer numbers
data1 = (2, 3, 3, 4, 5, 5, 5, 5, 6, 6, 6, 7)

# tuple of a set of floating point values
data2 = (2.4, 1.3, 1.3, 1.3, 2.4, 4.6)

# tuple of a set of fractional numbers
data3 = (fr(1, 2), fr(1, 2), fr(10, 3), fr(2, 3))

# tuple of a set of negative integers
data4 = (-1, -2, -2, -2, -7, -7, -9)

# tuple of strings
data5 = ("red", "blue", "black", "blue", "black", "black", "brown")


print("Mode of data set 1 is % s" % (mode(data1)))
print("Mode of data set 2 is % s" % (mode(data2)))
print("Mode of data set 3 is % s" % (mode(data3)))
print("Mode of data set 4 is % s" % (mode(data4)))
print("Mode of data set 5 is % s" % (mode(data5)))
```

```
Mode of data set 1 is 5
Mode of data set 2 is 1.3
Mode of data set 3 is 1/2
Mode of data set 4 is -2
Mode of data set 5 is black
```

Measure of variability

Range

In [29]:
```

# Sample Data
arr = [1, 2, 3, 4, 5]

#Finding Max
Maximum = max(arr)
# Finding Min
Minimum = min(arr)

# Difference Of Max and Min
Range = Maximum-Minimum
print("Maximum = {}, Minimum = {} and Range = {}".format(
    Maximum, Minimum, Range))
```

```
Maximum = 5, Minimum = 1 and Range = 4
```

Variance

In [31]:
```
from statistics import variance

# importing fractions as parameter values
from fractions import Fraction as fr

# tuple of a set of positive integers
# numbers are spread apart but not very much
sample1 = (1, 2, 5, 4, 8, 9, 12)

# tuple of a set of negative integers
sample2 = (-2, -4, -3, -1, -5, -6)

# tuple of a set of positive and negative numbers
# data-points are spread apart considerably
sample3 = (-9, -1, -0, 2, 1, 3, 4, 19)

# tuple of a set of fractional numbers
sample4 = (fr(1, 2), fr(2, 3), fr(3, 4),
           fr(5, 6), fr(7, 8))

# tuple of a set of floating point values
sample5 = (1.23, 1.45, 2.1, 2.2, 1.9)

# Print the variance of each samples
print("Variance of Sample1 is % s " % (variance(sample1)))
print("Variance of Sample2 is % s " % (variance(sample2)))
print("Variance of Sample3 is % s " % (variance(sample3)))
print("Variance of Sample4 is % s " % (variance(sample4)))
print("Variance of Sample5 is % s " % (variance(sample5)))
```

```
Variance of Sample1 is 15.80952380952381
Variance of Sample2 is 3.5
Variance of Sample3 is 61.125
Variance of Sample4 is 1/45
Variance of Sample5 is 0.17613000000000006
```

Standard Deviation

In [32]:

```
# importing the statistics module
from statistics import stdev

# importing fractions as parameter values
from fractions import Fraction as fr

# creating a varying range of sample sets
# numbers are spread apart but not very much
sample1 = (1, 2, 5, 4, 8, 9, 12)

# tuple of a set of negative integers
sample2 = (-2, -4, -3, -1, -5, -6)

# tuple of a set of positive and negative numbers
# data-points are spread apart considerably
sample3 = (-9, -1, -0, 2, 1, 3, 4, 19)

# tuple of a set of floating point values
sample4 = (1.23, 1.45, 2.1, 2.2, 1.9)

# Print the standard deviation of
# following sample sets of observations
print("The Standard Deviation of Sample1 is % s"
      % (stdev(sample1)))

print("The Standard Deviation of Sample2 is % s"
      % (stdev(sample2)))

print("The Standard Deviation of Sample3 is % s"
      % (stdev(sample3)))


print("The Standard Deviation of Sample4 is % s"
      % (stdev(sample4)))
```

```
The Standard Deviation of Sample1 is 3.9761191895520196
The Standard Deviation of Sample2 is 1.8708286933869707
The Standard Deviation of Sample3 is 7.8182478855559445
The Standard Deviation of Sample4 is 0.41967844833872525
```

In [ ]: