|  |  |
| --- | --- |
| **Course Code: AI-2002** | **Course: Artificial Intelligence Lab** |
| **Instructor(s):** | **Sania Urooj** |

**National University of Computer & Emerging Sciences, Karachi Computer Science Department**

**Summer 2023, Lab Manual – 08**

# Objective

* 1. Introduction to basics of probability theory
  2. Computing probabilities of a single observation & across a range of observations
  3. Explore how to use python to solve basic probability and Bayesian Network problems and then apply these skills to a couple of challenge problems.
  4. Introduction to Markov model libraries of python.

# Probability

At the most basic level, probability seeks to answer the question, “What is the chance of an event happening?” An **event** is some outcome of interest. To calculate the chance of an event happening, we also need to consider all the other events that can occur. The quintessential representation of probability is the humble coin toss. In a coin toss the only events that can happen are:

* + - Flipping a heads
    - Flipping a tails

*These two events form the* ***sample space****, the set of all possible events that can happen. To calculate the probability of an event occurring, we count how many times are event of interest can occur (say flipping heads) and dividing it by the sample space. Thus, probability will tell us that an ideal coin will have a 1-in-2 chance of being heads or tails. By looking at the events that can occur, probability gives us a framework for making* predictions *about how often events will happen. However, even though it seems obvious, if we actually try to toss some coins, we’re likely to get an abnormally high or low counts of heads every once in a while. If we don’t want to make the assumption that the coin is fair, what can we do? We can gather data! We can use statistics to calculate probabilities based on observations from the real world and check how it compares to the ideal.*

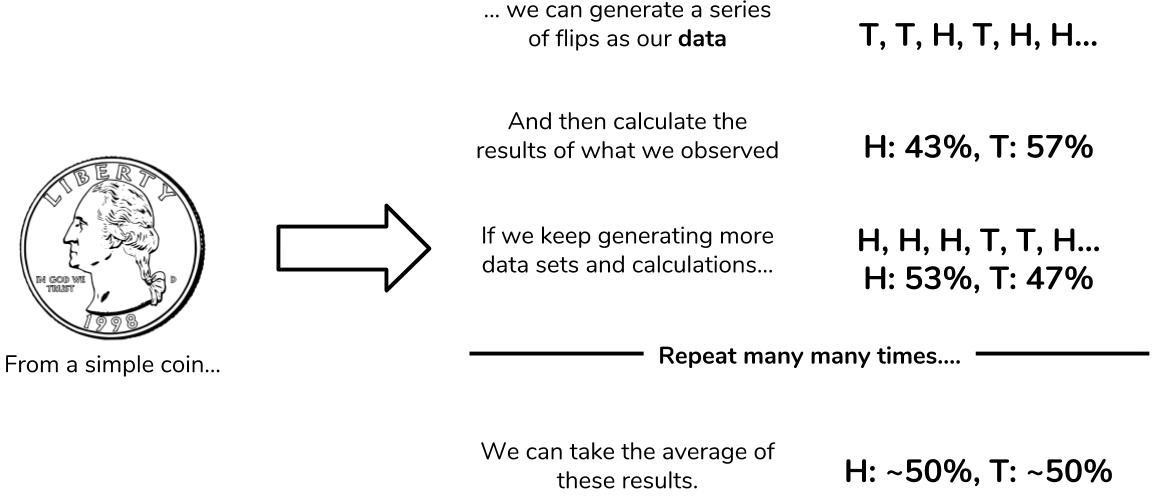
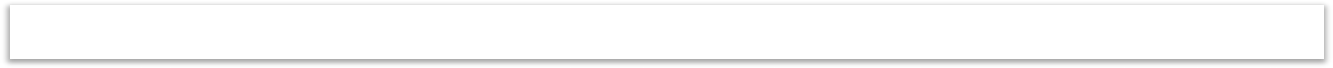
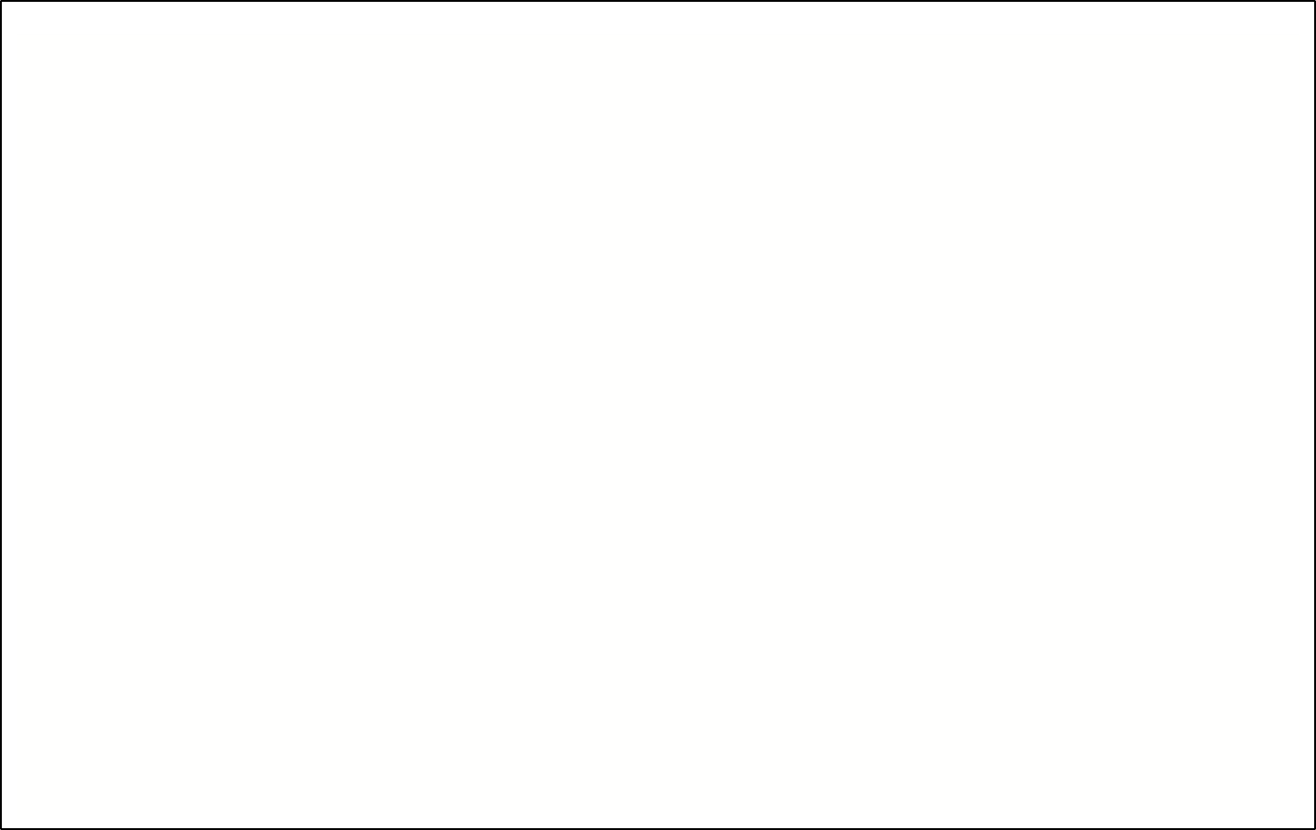
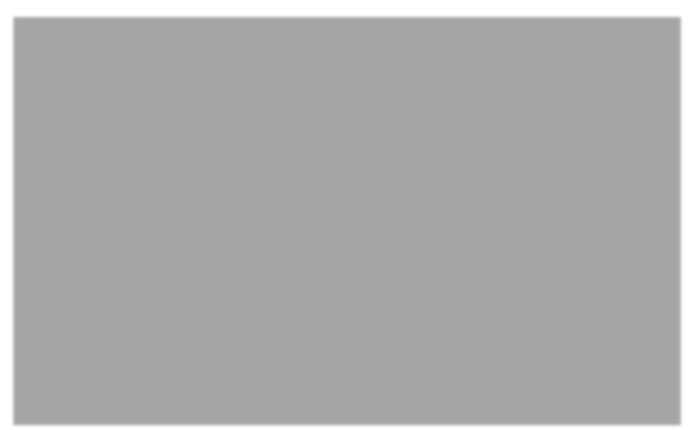
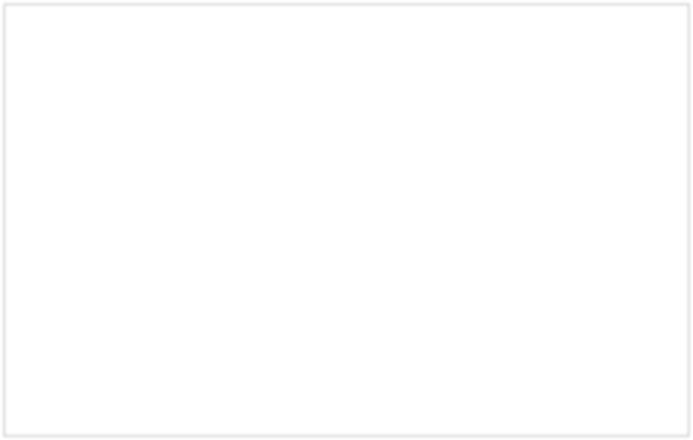
***From statistics to probability***

*Our data will be generated by flipping a coin 10 times and counting how many times we get heads. We will call a set of 10 coin tosses a* trial*. Our data point will be the number of heads we observe. We may not get the “ideal” 5 heads, but we won’t worry too much since one trial is only one data point. If we perform many, many trials, we expect the* average *number of heads over all of our trials to approach the 50%. The code below simulates 10, 100, 1000, and 1000000 trials, and then calculates the average proportion of heads observed. Our process is summarized in the image below as well.*

# Conditional Probability

# Conditional probability is the probability of an event A occurring given that event B has already occurred. It is denoted as P(A|B), where P represents probability. The formula for conditional probability is:

# P(A|B) = P(A ∩ B) / P(B)



**Code # 01**

Three bags contain 6 red, 4 black; 4 red, 6 black, and 5 red, 5 black balls respectively.

One of the bags is selected at random and a ball is drawn from it. If the ball drawn is red,

find the probability that it is drawn from the first bags.

# Given probabilities

p\_red\_given\_u1 = 3/5

p\_red\_given\_u2 = 2/5

p\_red\_given\_u3 = 1/2

p\_u1 = 1/3

p\_u2 = 1/3

p\_u3 = 1/3

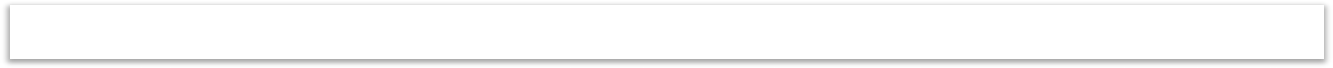
# Calculate P(R)

p\_red = p\_red\_given\_u1 \* p\_u1 + p\_red\_given\_u2 \* p\_u2 + p\_red\_given\_u3 \* p\_u3

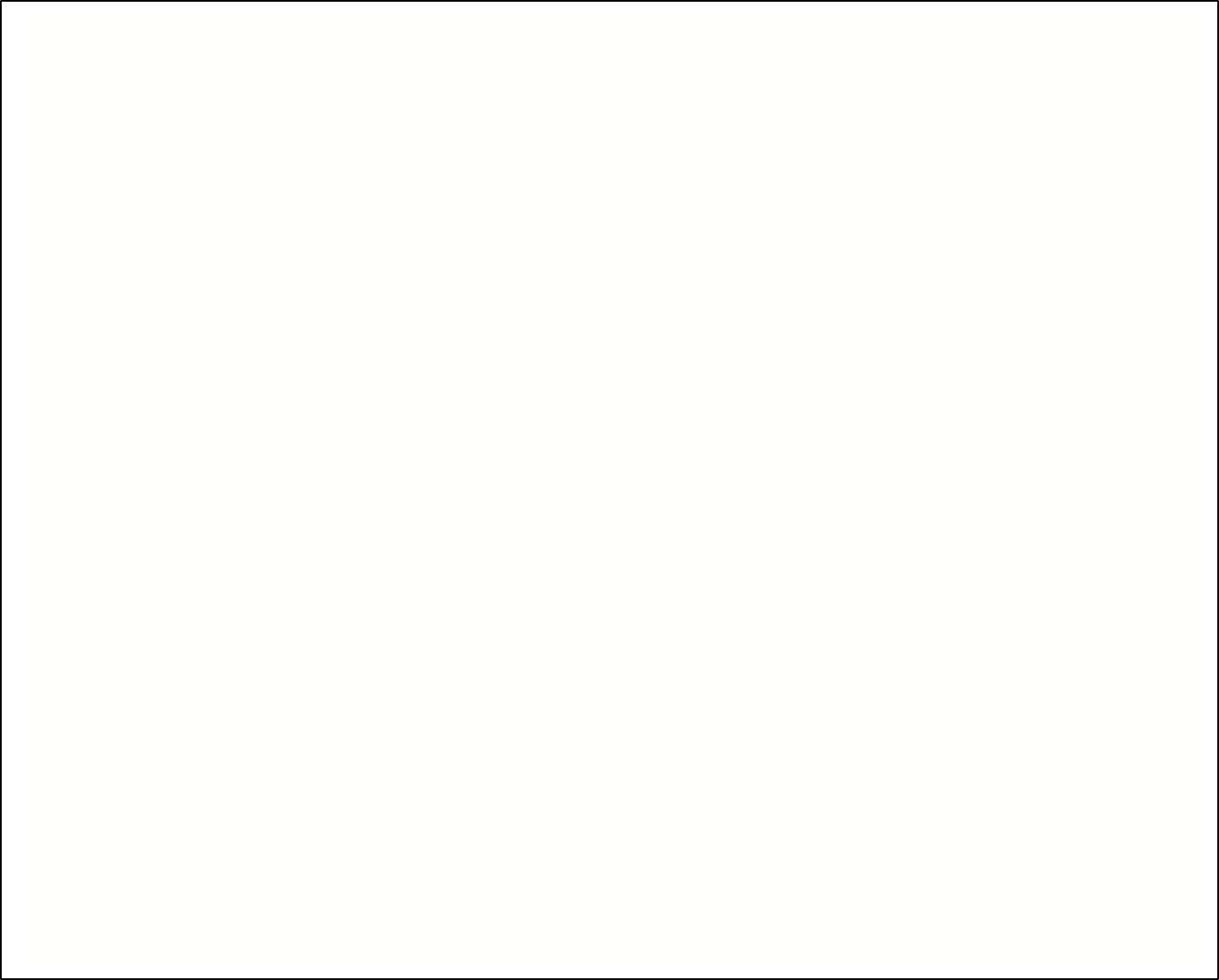
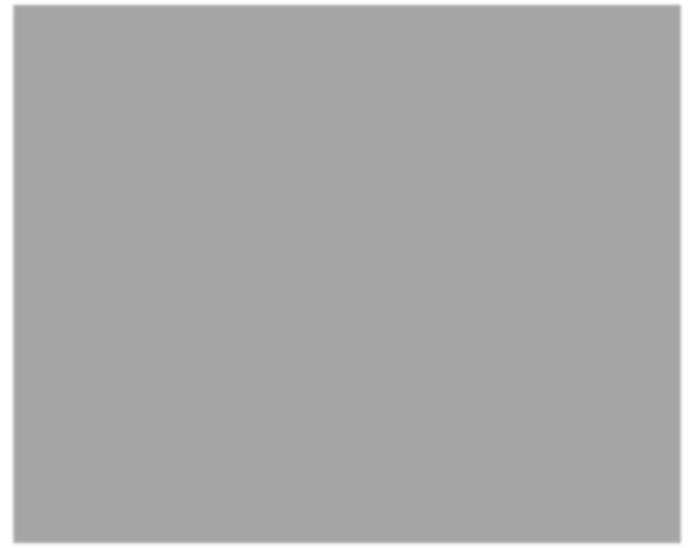
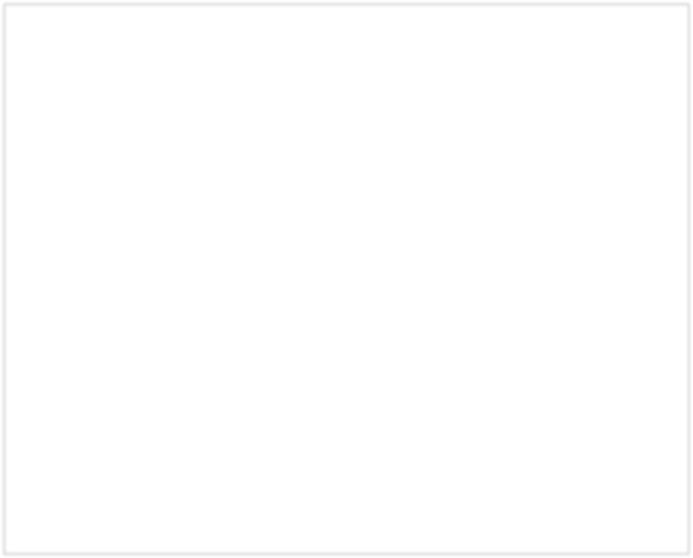
# Using Bayes' theorem to find the probability of selecting the first bag given drawing a red ball

p\_u1\_given\_red = (p\_red\_given\_u1 \* p\_u1) / p\_red

print("Probability of selecting the first bag given drawing a red ball:", p\_u1\_given\_red)



**Code # 02**



Let's consider an example of rolling two fair six-sided dice. We want to calculate the conditional probability of getting a sum of 7 on the two dice given that the first die shows a 3 (event A).

Then, we will use Bayes' Theorem to calculate the probability of rolling a 3 on the first die

given that the sum of the two dice is 7 (event B).

# Define the sample space of rolling two dice

sample\_space = [(i, j) for i in range(1, 7) for j in range(1, 7)]

# Event A: Getting a 3 on the first die

event\_A = [(3, j) for j in range(1, 7)]

# Event B: The sum of the two dice is 7

event\_B = [(i, j) for i, j in sample\_space if i + j == 7]

# Calculate P(B|A) - Conditional probability of getting a sum of 7 given the first die shows a 3

P\_B\_given\_A = len(set(event\_A) & set(event\_B)) / len(event\_A)

# Calculate P(A) - Probability of getting a 3 on the first die

P\_A = 1 / 6

# Calculate P(B) - Probability of getting a sum of 7 on the two dice

P\_B = len(event\_B) / len(sample\_space)

# Use Bayes' Theorem to calculate P(A|B) - Probability of getting a 3 given the sum of the two dice is 7

P\_A\_given\_B = (P\_B\_given\_A \* P\_A) / P\_B

print("Conditional probability P(B|A): {:.4f}".format(P\_B\_given\_A))

print("Probability P(A): {:.4f}".format(P\_A))

print("Probability P(B): {:.4f}".format(P\_B))

print("Probability P(A|B): {:.4f}".format(P\_A\_given\_B))

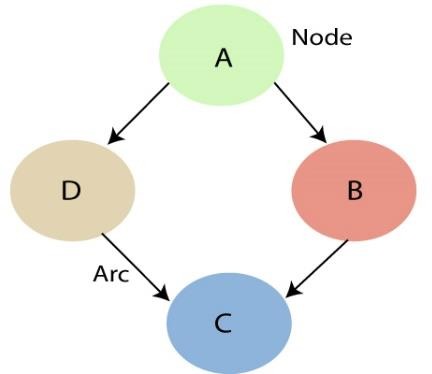
**Bayesian Network**

A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph. It is also called a Bayes network, belief network, decision network, or Bayesian model. Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection. Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

* Directed Acyclic Graph
* Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram. A Bayesian network graph is made up of nodes and Arcs (directed links), where:



* Each node corresponds to the random variables, and a variable can be continuous or discrete.
* Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.
* These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
* In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
* If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
* Node C is independent of node A.
* Each node in the Bayesian network has condition probability distribution **P(Xi |Parent(Xi) )**, which determines the effect of the parent on that node.

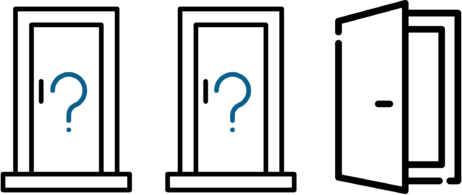
Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

**Bayesian Networks Python**

In this demo, we’ll be using Bayesian Networks to solve the famous Monty Hall Problem. For those of you who don’t know what the Monty Hall problem is, let me explain:

The Monty Hall problem named after the host of the TV series, ‘Let’s Make A Deal’, is a paradoxical probability puzzle that has been confusing people for over a decade.

So this is how it works. The game involves three doors, given that behind one of these doors is a car and the remaining two have goats behind them. So you start by picking a random door, say #2. On the other hand, the host knows where the car is hidden and he opens another door, say #1 (behind which there is a goat). Here’s the catch, you’re now given a choice, the host will ask you if you want to pick door #3 instead of your first choice i.e. #2.

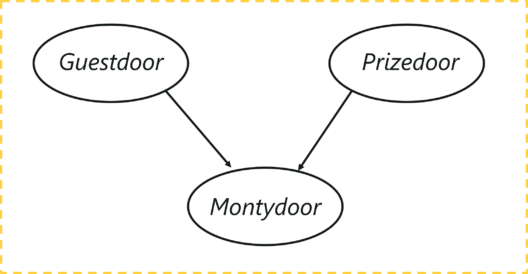


Is it better if you switch your choice or should you stick to your first choice?

This is exactly what we’re going to model. We’ll be creating a Bayesian Network to understand the probability of winning if the participant decides to switch his choice.

Now let’s get started.

* The first step is to build a Directed Acyclic Graph.
* The graph has three nodes, each representing the door chosen by:



* The door selected by the Guest
* The door containing the prize (car)
* The door Monty chooses to open

Let’s understand the dependencies here, the door selected by the guest and the door containing the car are completely random processes. However, the door Monty chooses to open is dependent on both the doors; the door selected by the guest, and the door the prize is behind. Monty has to choose in such a way that the door does not contain the prize and it cannot be the one chosen by the guest.

**Example 1**

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Create a Bayesian network for the Monty Hall problem

model = BayesianModel([('Door 1', 'Door 2'), ('Door 1', 'Door 3')])

# Define the conditional probability distributions (CPDs)

cpd\_door1 = TabularCPD(variable='Door 1', variable\_card=3, values=[[1/3], [1/3], [1/3]])

# Monty's actions depend on the contestant's choice

cpd\_door2 = TabularCPD(variable='Door 2', variable\_card=3,

                       values=[[0, 1/2, 1], [1/2, 0, 0], [1/2, 1/2, 0]],

                       evidence=['Door 1'], evidence\_card=[3])

cpd\_door3 = TabularCPD(variable='Door 3', variable\_card=3,

                       values=[[0, 1, 1/2], [1, 0, 1/2], [0, 0, 0]],

                       evidence=['Door 1'], evidence\_card=[3])

# Add the CPDs to the Bayesian network

model.add\_cpds(cpd\_door1, cpd\_door2, cpd\_door3)

# Check if the model is valid

assert model.check\_model(), "Invalid model."

# Perform inference using Variable Elimination algorithm

inference = VariableElimination(model)

# Probability of winning the prize when sticking to the initial choice (Door 1)

result\_stick = inference.query(variables=['Door 1'], evidence={})

prob\_win\_stick = result\_stick.values[0]

# Probability of winning the prize when switching doors (not choosing Door 1)

result\_switch = inference.query(variables=['Door 2', 'Door 3'], evidence={'Door 1': 0})

prob\_win\_switch = sum(result\_switch.values)

# Print the probabilities of winning with stick and switch strategies

print("Probability of winning by sticking with initial choice (Door 1):", prob\_win\_stick)

print("Probability of winning by switching doors:", prob\_win\_switch)

**Dynamic Bayesian Networks**

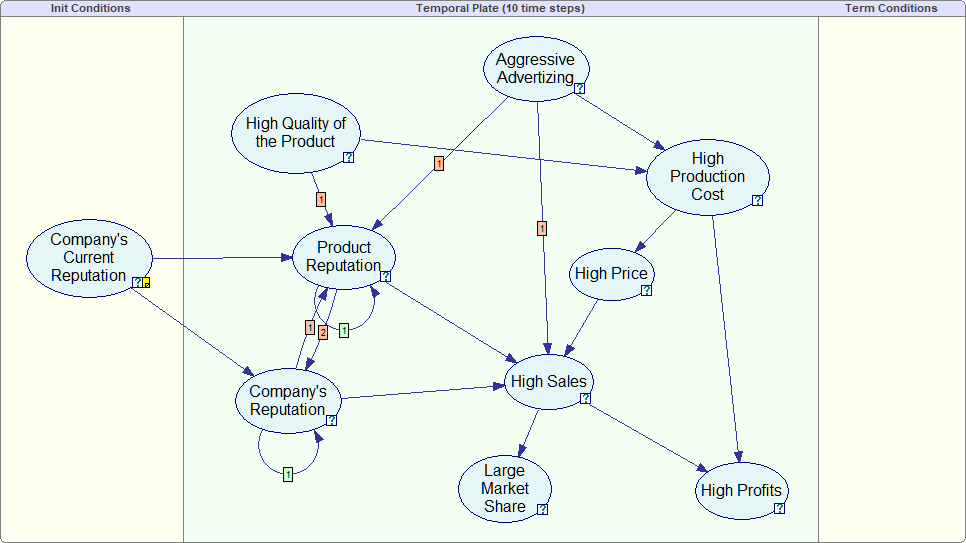
A [Bayesian network](https://www.bayesfusion.com/bayesian-networks/) is a snapshot of the system at a given time and is used to model systems that are in some kind of equilibrium state. Unfortunately, most systems in the world change over time and sometimes we are interested in how these systems evolve over time more than we are interested in their equilibrium states. Whenever the focus of our reasoning is change of a system over time, we need a tool that is capable of modeling dynamic systems.

A dynamic Bayesian network (DBN) is a Bayesian network extended with additional mechanisms that are capable of modeling influences over time. The temporal extension of Bayesian networks does not mean that the network structure or parameters change dynamically, but that a dynamic system is modeled. In other words, the underlying process, modeled by a DBN, is stationary. A DBN is a model of a stochastic process.

**The structure of a dynamic Bayesian network and its interpretation**

Consider a decision problem faced by a manufacturer, who is conscious of the fact that the quality of a new product will come at a price (high production cost), which will drive the product price up and, effectively, lead to decrease of sales, profits, and market share. On the other hand, high product quality will positively impact the product reputation over time and the product reputation will, again over time, impact the reputation of the company. These impact directly sales and profits. The problem is complex, as reputation (both products and companies) is to some degree a self-propelling process, i.e., reputation at a given point in time, in addition to other factors, impacts reputation in the immediate future. Company’s reputation will also impact the reputation of its products, including the current product. The decision is not a one-shot decision, whose effects we will know immediately but rather a decision whose effects will unfold over time. In order to assess the consequences of this decision, the manufacturer needs to model time explicitly and use dynamic Bayesian networks.

The following GeNIe screen shows the structure of a model representing this problem. The way that GeNIe represents dynamic models is different from other representations in that it does not focus on time slices but rather on variables. This representation allows for a compact modeling of higher order temporal influences. One thing that we can notice right away in the screen shot below is that there are special arcs in the model that are labeled by numbers inside little squares. These arcs represent temporal influences and the numbers denote their order. An influence of the first order, represents an influence spanning over one time step. Influences of higher order, represent slower influences that span over multiple time steps. The number in the square expresses the number of time steps over which the influence spans. We can see that both Product Reputation and Company’s Reputation impact themselves in the near future. There is an influence of Company’s Reputation on Product Reputation and a slower, two-step influence of Product Reputation on Company’s Reputation. Product Reputation will thus slowly impact Company’s Reputation. Aggressive Advertising will impact both Product Reputation and High Sales over time.



Dynamic Bayesian network model allows us to calculate how probabilities of interest change over time. This is of vital interest to decision who deal with consequences of their decisions over time. The following plot shows the same model with nodes viewed as bar charts and

High Quality of the Product set to False. We can see the marginal probabilities changing over the time horizon of 10 steps (this is a model parameter that can be easily changed).

**Dynamic Bayesian Networks Python**

* PGMPY implementation
* PyAgrum implementation

Run the code Files PGMPY and PyAgrum

**Example 2:**

# Import required libraries

from pgmpy.models import DynamicBayesianNetwork as DBN

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Define the variables and their relationships

dbn = DBN()

# Define the variables and their conditional probabilities at each time step

cpd1 = TabularCPD(variable='A', variable\_card=2,

values=[[0.7, 0.3], [0.3, 0.7]])

cpd2 = TabularCPD(variable='B', variable\_card=2,

values=[[0.9, 0.1], [0.2, 0.8]], evidence=['A'], evidence\_card=[2])

cpd3 = TabularCPD(variable='C', variable\_card=2,

values=[[0.8, 0.2], [0.1, 0.9]], evidence=['B'], evidence\_card=[2])

cpd4 = TabularCPD(variable='D', variable\_card=2,

values=[[0.5, 0.5], [0.7, 0.3]], evidence=['C'], evidence\_card=[2])

# Add the variables and their conditional probabilities to the DBN

dbn.add\_cpds(cpd1, cpd2, cpd3, cpd4)

# Perform inference using the VariableElimination algorithm

inference = VariableElimination(dbn)

# Query the network for the probability of D at time step 3, given evidence of A at time step 1 and B at time step 2

query = inference.query(variables=['D'], evidence={'A': 0, 'B\_1': 1}, joint=False, show\_progress=False)

print(query)

## Problems and Solutions

**Example 1: Two dies are thrown simultaneously, and the sum of the numbers obtained is found to be 7. What is the probability that the number 3 has appeared at least once?**

Solution: The sample space S would consist of all the numbers possible by the combination of two dies. Therefore S consists of 6 × 6, i.e. 36 events.

Event A indicates the combination in which 3 has appeared at least once.

Event B indicates the combination of the numbers which sum up to 7.

A = {(3, 1), (3, 2), (3, 3)(3, 4)(3, 5)(3, 6)(1, 3)(2, 3)(4, 3)(5, 3)(6, 3)}

B = {(1, 6)(2, 5)(3, 4)(4, 3)(5, 2)(6, 1)}

P(A) = 11/36

P(B) = 6/36

A ∩ B = 2

P(A ∩ B) = 2/36

Applying the conditional probability formula we get,

P(A|B) = P(A∩B)/P(B) = (2/36)/(6/36) = ⅓

**Example 2: In a group of 100 computer buyers, 40 bought CPU, 30 purchased monitor, and 20 purchased CPU and monitors. If a computer buyer chose at random and bought a CPU, what is the probability they also bought a Monitor?**

**Solution:**As per the first event, 40 out of 100 bought CPU,

So, P(A) = 40% or 0.4

Now, according to the question, 20 buyers purchased both CPU and monitors. So, this is the intersection of the happening of two events. Hence,

P(A∩B) = 20% or 0.2

By the formula of conditional probability we know;

P(B|A) = P(A∩B)/P(B)

P(B|A) = 0.2/0.4 = 2/4 = ½ = 0.5

The probability that a buyer bought a monitor, given that they purchased a CPU, is 50%

Three urns contain 6 red, 4 black; 4 red, 6 black, and 5 red, 5 black balls respectively. One of the urns is selected at random and a ball is drawn from it. If the ball drawn is red, find the probability that it is drawn from the first urn.

# Given probabilities

p\_red\_given\_u1 = 3/5

p\_red\_given\_u2 = 2/5

p\_red\_given\_u3 = 1/2

p\_u1 = 1/3

p\_u2 = 1/3

p\_u3 = 1/3

# Calculate P(R)

p\_red = p\_red\_given\_u1 \* p\_u1 + p\_red\_given\_u2 \* p\_u2 + p\_red\_given\_u3 \* p\_u3

# Using Bayes' theorem to find the probability of selecting the first urn given drawing a red ball

p\_u1\_given\_red = (p\_red\_given\_u1 \* p\_u1) / p\_red

print("Probability of selecting the first urn given drawing a red ball:", p\_u1\_given\_red)