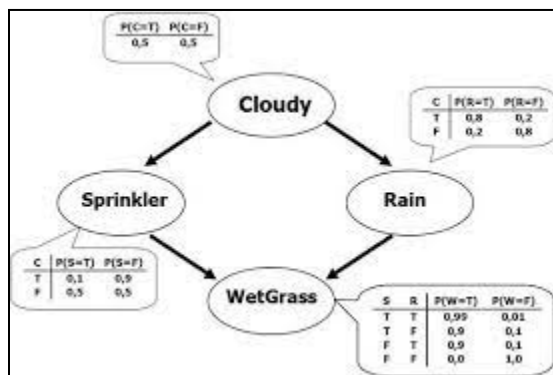


Aim: To implement Inferencing with Bayesian Network in Python.

Theory:

Bayesian networks are a type of probabilistic graphical model that uses Bayesian inference for therefore causation, by representing conditional dependence by edges in a directed graph. Using the relationships specified by our Bayesian network, we can obtain a compact, factorized representation of the joint probability distribution by taking advantage of conditional independence.



A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. Formally, if an edge (A, B) exists in the graph connecting random variables A and B , it means that $P(B|A)$ is a factor in the joint probability distribution, so we must know $P(B|A)$ for all values of B and A in order to conduct inference. In the above example, since Rain has an edge going into WetGrass, it means that $P(WetGrass|Rain)$ will be a factor, whose probability values are specified next to the WetGrass node in a conditional probability table.

Bayesian networks satisfy the local Markov property, which states that a node is conditionally independent of its non-descendants given its parents. In the above example, this means that $P(Sprinkler|Cloudy, Rain) = P(Sprinkler|Cloudy)$ since Sprinkler is conditionally independent of its non-descendant, Rain, given Cloudy. This property allows us to simplify the joint distribution, obtained in the previous section using the chain rule, to a smaller form. After simplification, the joint distribution for a Bayesian network is equal to the product of $P(\text{node}|\text{parents}(\text{node}))$ for all nodes, stated below:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) = \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

In larger networks, this property allows us to greatly reduce the amount of required computation, since generally, most nodes will have few parents relative to the overall size of the network.

Inference over a Bayesian network can come in two forms.

The first is simply evaluating the joint probability of a particular assignment of values for each variable (or a subset) in the network. For this, we already have a factored form of the joint distribution, so we simply evaluate that product using the provided conditional probabilities. If we only care about a subset of variables, we will need to marginalize out the ones we are not interested in. In many cases, this may result in underflow, so it is common to take the logarithm of that product, which is equivalent to adding up the individual logarithms of each term in the product.

The second, more interesting inference task, is to find $P(x|e)$, or, to find the probability of some assignment of a subset of the variables (x) given assignments of other variables (our evidence, e). In the above example, an example of this could be to find $P(\text{Sprinkler}, \text{WetGrass} | \text{Cloudy})$, where $\{\text{Sprinkler}, \text{WetGrass}\}$ is our x , and $\{\text{Cloudy}\}$ is our e . In order to calculate this, we use the fact that $P(x|e) = P(x, e) / P(e) = \alpha P(x, e)$, where α is a normalization constant that we will calculate at the end such that $P(x|e) + P(\neg x | e) = 1$. In order to calculate $P(x, e)$, we must marginalize the joint probability distribution over the variables that do not appear in x or e , which we will denote as Y .

$$P(x|e) = \alpha \sum_{\forall y \in Y} P(x, e, Y)$$


Code and Output:

In this demonstration, we'll use Bayesian Networks to solve the well-known Monty Hall Problem. Let me explain the Monty Hall problem to those of you who are unfamiliar with it:

This problem entails a competition in which a contestant must choose one of three doors, one of which conceals a price. The show's host (Monty) unlocks an empty door and asks the contestant if he wants to swap to the other door after the contestant has chosen one.


The decision is whether to keep the current door or replace it with a new one. It is preferable to enter by the other door because the price is more likely to be higher. To come out from this ambiguity let's model this with a Bayesian network.

For this demonstration, we are using a python-based package pgmpy is a Bayesian Networks implementation written entirely in Python with a focus on modularity and flexibility. Structure Learning, Parameter Estimation, Approximate (Sampling-Based) and Exact inference, and Causal Inference are all available as implementations.



```
!pip install pgmpy
```

Step 1: To initialize the BayesianModel by passing a list of edges in the model structure.



```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
import networkx as nx
import pylab as plt
```

Step 2: Define the CPDs(Conditional Probability Distribution).

```
# Defining Bayesian Structure
model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])
# Defining the CPDs:
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])
cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])
cpd_host = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5], [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5], [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]], evidence=['Guest', 'Price'], evidence_card=[3, 3])
```

Step 3: Add the CPDs to the model.

```
# Associating the CPDs with the network structure.
model.add_cpds(cpd_guest, cpd_price, cpd_host)
model.check_model()

True
```

Now let's infer the network, if we want to check at the next step which door will the host open now. For that, we need access to the posterior probability from the network and while accessing we need to pass the evidence to the function. Evidence is needed to be given when we are evaluating posterior probability, here in our task evidence is nothing but which door is Guest selected and where is the Price.

```
# Inferring the posterior probability
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price': 2})
print(posterior_p)
```

Host	phi(Host)
Host(0)	0.5000
Host(1)	0.5000
Host(2)	0.0000

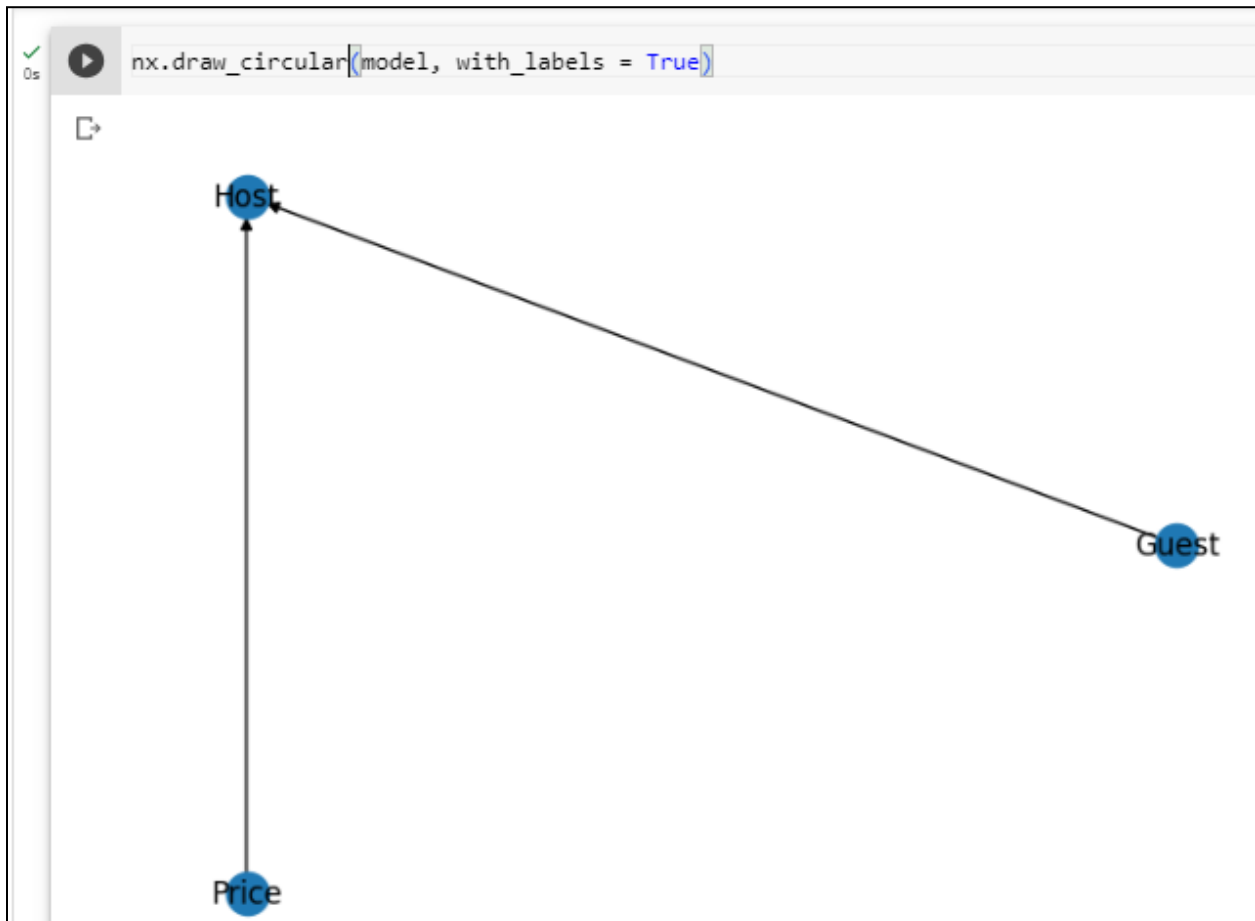


```
# Inferring the posterior probability
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 1, 'Price': 2})
print(posterior_p)
```

Host	phi(Host)
Host(0)	1.0000
Host(1)	0.0000
Host(2)	0.0000

The probability distribution of the Host is clearly satisfying the theme of the contest. In reality also, in this situation the host is definitely not going to open the second door; he will open either of the first two and that's what the above simulation tells us.

Now, let's plot our above model. This can be done with the help of Network and PyLab. NetworkX is a Python-based software package for constructing, altering, and researching the structure, dynamics, and function of complex networks. PyLab is a procedural interface to the object-oriented charting toolkit Matplotlib, and it is used to examine large complex networks represented as graphs with nodes and edges.



Conclusion:

Thus we studied an overview of Bayesian networks and implemented Inferencing with Bayesian Network in Python.