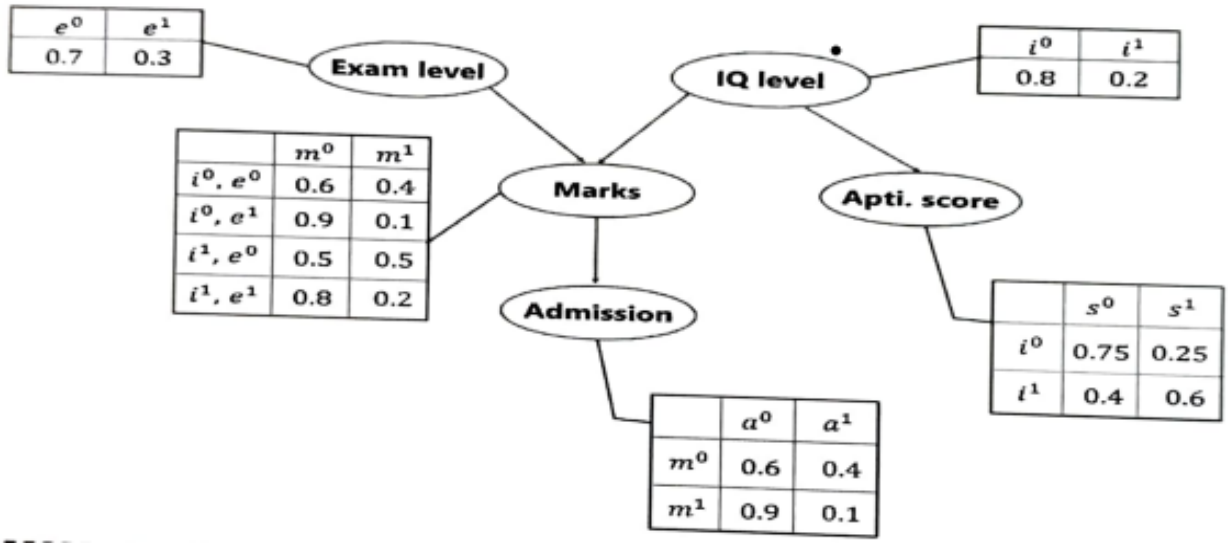
Name: Kaushik Kotian Roll No:29 Div : D20B Batch:B

EXPERIMENT NO. 1

Aim: 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, factorised representation of the joint probability distribution by taking advantage of conditional independence.



e^0	e^1
0.7	0.3

i^0	i^1
0.8	0.2

	m^0	m^1
i^0, e^0	0.6	0.4
i^0, e^1	0.9	0.1
i^1, e^0	0.5	0.5
i^1, e^1	0.8	0.2

	s^0	s^1
i^0	0.75	0.25
i^1	0.4	0.6

	a^0	a^1
m^0	0.6	0.4
m^1	0.9	0.1

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. Let's assume that we're creating a Bayesian Network that will model the marks (m) of a student on his examination. The marks will depend on:
1. Exam level (e): This is a discrete variable that can take two values, (difficult, easy)
2. IQ of the student (i): A discrete variable that can take two values (high, low)
The marks will intern predict whether or not he/she will get admitted (a) to a university. The IQ will also predict the aptitude score (s) of the student.
With this information, we can build a Bayesian Network that will model the performance of a student on an exam. The Bayesian Network can be represented as a DAG where each node denotes a variable that predicts the performance of the student. We can now calculate the Joint Probability Distribution of these 5 variables, i.e. the product of conditional probabilities:

p(a, m, i, e, s) = p(a | m) p(m | i, e) p(i) p(e) p(s | i)

Bayesian networks satisfy the local Markov property, which states that a node is conditionally independent of its non-descendants given its parents,the joint distribution for a Bayesian network is equal to the product of P(nodelparents(node)) for all nodes, stated below:

P(X, ... ,X.)=[P(x, |X,, ... ,X,)=[[P(X,\Parents(X,))

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 marginalise 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.

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.

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

```
!pip install pgmpy
!pip install --upgrade networkx
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).

```
model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])

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])
```

P(C):

C	0	1	2
	0.33	0.33	0.33

P(P):

P	0	1	2
	0.33	0.33	0.33

P(H | P, C):

C	0			1			2		
P	0	1	2	0	1	2	0	1	2
H=0	0	0	0	0	0.5	1	0	1	0.5
H=1	0.5	0	1	0	0	0	1	0	0.5
H=2	0.5	1	0	1	0.5	0	0	0	0

Step 3: Add the CPDs to the model.

```
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.

```
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 |
+---------+-------------+
```

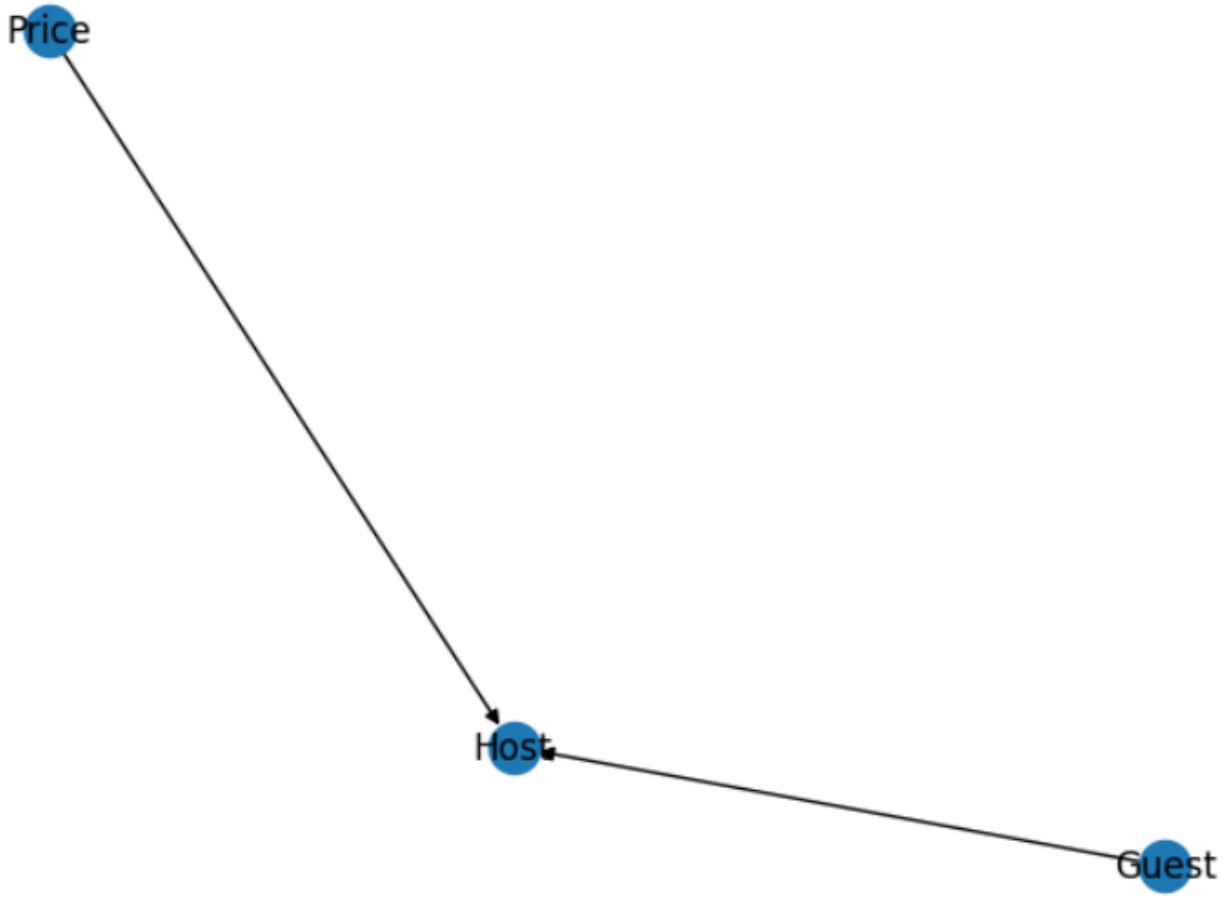
```
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 1, 'Price': 2})
print(posterior_p)
```

```
+---------+-------------+
| 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.

```
nxgraph = nx.DiGraph(model.edges())
nx.draw(nxgraph, with_labels=True)
plt.show()
```



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