Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

COMPARATIVE ANALYSIS OF TREE BASED AND HILL CLIMB SEARCH BASED CPDS FOR EVIDENTIAL INFERENCING

Vineet Chotaliya - 60009210051
Harsh Shetye - 60009210068
Bhuvi Ghosh - 600092100191
Satvam Thakkar - 60009210196
Nivedi Kondlekar - 60009220177

Guide:
Prof. Dr. Kriti Srivastava

TABLE OF CONTENTS

Colab Link:
https://colab.research.google.com/drive/1M_6vncYnVu9E1qRP5bpaWKzO-a1eK44D?usp=sharing

Introduction

Postnatal depression, a pervasive mental health concern affecting mothers during the early postpartum period, stands as a critical issue with profound implications for maternal and infant well-being. Its multifaceted nature demands a comprehensive exploration of the underlying factors contributing to its occurrence and, more importantly, its outcomes. In this research endeavor, our focus lies in the in-depth analysis of a medical dataset specifically curated to investigate postnatal depression. The overarching goal is to discern the risk factors associated with this condition, model the conditional dependencies between various variables, and, crucially, gain insights into the factors influencing outcomes such as heightened anxiety and the risk of suicide.

The complexity of postnatal depression necessitates sophisticated analytical tools, and probabilistic graphical models (PGMs) present themselves as a promising methodology for navigating this intricate landscape. Among PGMs, Bayesian Networks offer a particularly robust framework for capturing the conditional relationships that exist within the dataset. By encapsulating the interplay between demographic information, symptomatic indicators, and key risk factors, Bayesian Networks provide a powerful mechanism for probabilistic inference. This introduction establishes the motivation behind employing PGMs, highlighting the unique capabilities of Bayesian Networks to unravel the complexities inherent in postnatal depression. It sets the stage for a detailed exploration into how these models enhance our understanding of the condition and its associated risk factors, ultimately contributing to more effective interventions and support systems for mothers during this critical phase of their lives.

Problem Definition

This project centers around the predictive analysis of postnatal depression using a carefully curated dataset encompassing ten variables. These variables include demographic details such as age, symptomatic indicators like feelings of sadness and irritability, and potential risk factors including anxiety and bonding issues. The ultimate objective is to develop a robust predictive model that can effectively anticipate the likelihood of postnatal depression based on these variables.

To achieve this, we leverage probabilistic graphical models, specifically Bayesian Networks, as a powerful tool for capturing the conditional dependencies between the diverse set of variables. Bayesian Networks offer a unique advantage in modeling complex relationships through a directed acyclic graph, allowing us to unveil how different factors interact and contribute to the risk of postnatal depression. The project involves not only the construction of this Bayesian Network but also the utilization of inference techniques to make accurate predictions and gain valuable insights into the factors influencing postnatal depression outcomes.

Through this project, we seek to provide a practical and impactful tool for healthcare professionals and support systems to identify and address postnatal depression in a timely and targeted manner. By integrating data-driven insights into the maternal care framework, we aim to enhance the well-being of new mothers during the critical postpartum period.

Dataset Description:

The dataset comprises records collected from a medical hospital related to postpartum depression where mothers were asked to answer a questionnaire on various parameters.

The selected attributes are as follows:

1. **Timestamp:** The date and time when the questionnaire was administered.
2. **Age:** The age of the respondents.
3. **Feeling sad or Tearful:** Indicating if respondents experience feelings of sadness or tearfulness.
4. **Irritable towards baby & partner:** Reflecting whether respondents feel irritable towards their baby and partner.
5. **Trouble sleeping at night:** Capturing any reported difficulties in sleeping at night.

6. **Problems concentrating or making decisions:** Indicating challenges in concentration or decision-making.
7. **Overeating or loss of appetite:** Identifying if respondents experience changes in eating habits.
8. **Feeling anxious:** The target attribute, indicating the presence or absence of feelings of anxiety.
9. **Feeling of guilt:** Capturing any reported feelings of guilt.
10. **Problems of bonding with baby:** Reflecting any challenges reported in bonding with the baby.

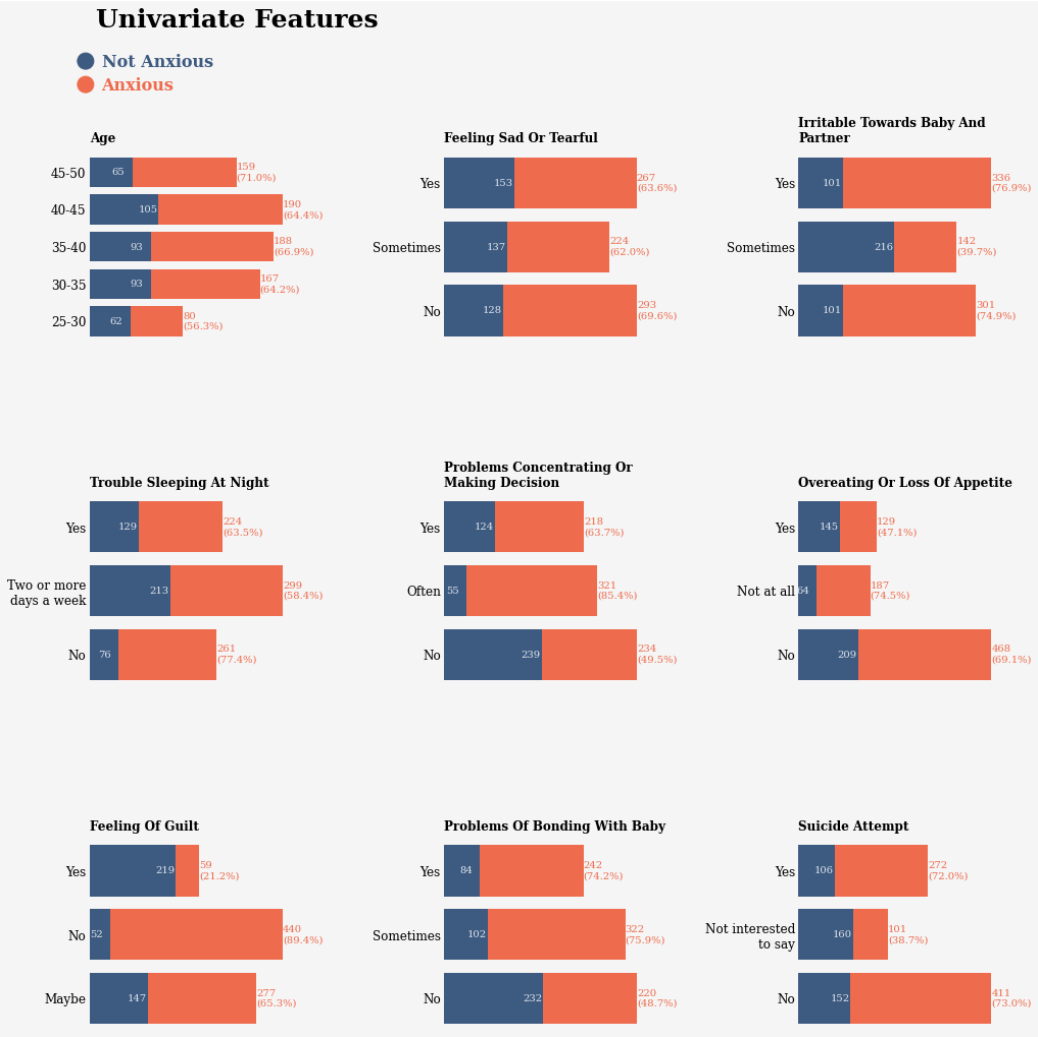


Fig 1 - Analysis of features from the dataset

Need of PGM

Probabilistic Graphical Models (PGMs) are powerful tools for modeling complex relationships among variables in a dataset, providing a structured framework for probabilistic reasoning. In the context of our project on predicting postnatal depression, it's essential to understand the significance of PGMs in uncovering the intricate dynamics within the dataset.

A Probabilistic Graphical Model is a graphical representation of a probability distribution that captures the conditional dependencies and independencies among a set of variables. Specifically, Bayesian Networks, a type of PGM, use a directed acyclic graph (DAG) to depict these relationships. In our project, the variables include demographic details, symptoms, and risk factors associated with postnatal depression.

The need for PGMs arises from the complexity of the postnatal depression dataset, where variables are interrelated, and their joint distribution influences the likelihood of adverse outcomes. Bayesian Networks, chosen for their efficiency, offer a structured approach to model conditional dependencies. By encoding relationships in a DAG, these models facilitate both interpretation and probabilistic inference, enabling us to predict and understand the factors contributing to postnatal depression.

In summary, the project leverages PGMs, particularly Bayesian Networks, to navigate the intricacies of the dataset. The structured representation allows for a more nuanced understanding of conditional dependencies, crucial for accurate prediction and insightful interpretation of postnatal depression risk factors.
The dataset represents a Directed Acyclic Graph (DAG) that encapsulates relationships between key attributes relevant to postnatal depression, each delineated by an associated weight. This DAG serves as a visual framework, capturing the directed influences among variables. For instance, the link from "Age" to "Overeating or loss of appetite" implies a potential correlation between age and changes in eating habits.

The weights assigned to each edge in the DAG indicate the strength or significance of the relationship between the source and target attributes. These weights are instrumental in quantifying the impact of one attribute on another, providing valuable insights into the conditional dependencies within the dataset.

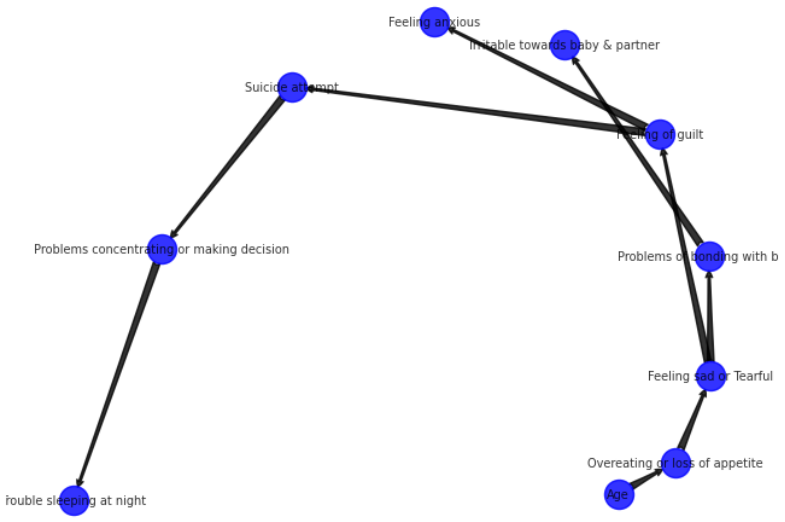


Fig 3 - Directed Acyclic Graph using 'Age' as the start node

This structured DAG is crucial for constructing a probabilistic graphical model, particularly a Bayesian Network, as it illuminates the intricate web of dependencies in postnatal mental health factors. Analyzing this graph enables us to understand not only the direct connections but also the cascading effects through the network, allowing for a comprehensive exploration of the interplay between demographic details, symptoms, and risk factors associated with postnatal depression.

	source	target
0	Age	Overeating or loss of appetite
1	Overeating or loss of appetite	Feeling sad or Tearful
2	Feeling sad or Tearful	Problems of bonding with baby
3	Feeling sad or Tearful	Feeling of guilt
4	Problems of bonding with baby	Irritable towards baby & partner
5	Feeling of guilt	Feeling anxious
6	Feeling of guilt	Suicide attempt
7	Suicide attempt	Problems concentrating or making decision
8	Problems concentrating or making decision	Trouble sleeping at night

Fig 4 - Table describing the DAG in Fig3

Representation

In our project for predicting postnatal depression, we have chosen to represent the relationships among variables using Bayesian Networks. This selection is driven by several key factors that align with the nature of our dataset and the objectives of our analysis.

Representation using a simple Bayesian Network:

1. **Conditional Dependency Modeling:** Bayesian Networks excel in modeling conditional dependencies between variables. Given the diverse set of variables in our dataset, including demographic details, symptoms, and risk factors, it is crucial to capture how these factors influence each other. The directed acyclic graph (DAG) structure of Bayesian Networks allows us to represent these dependencies explicitly, providing a clear understanding of the relationships at play.

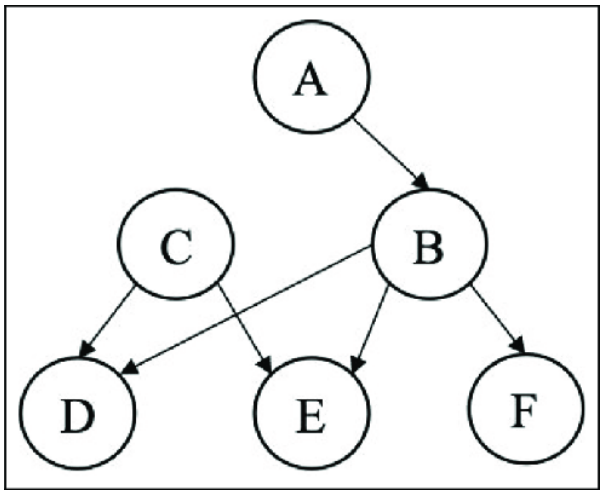


Fig 2 - A simple Bayesian Network with 6 variables

2. **Interpretability:**Interpretability is paramount in a project focused on maternal mental health. The graphical nature of Bayesian Networks makes it easier to communicate and understand the relationships among variables. This transparency is particularly valuable for healthcare professionals and support systems seeking actionable insights from the model to aid in timely interventions.

3. **Efficient Inference:**Bayesian Networks facilitate efficient inference, enabling us to compute probabilities and make predictions with relative ease. The efficient algorithms associated with Bayesian Networks are well-suited for handling the complexities inherent in postnatal depression, ensuring that the model can provide timely and accurate assessments.
4. **Graphical Structure:**The graphical structure of Bayesian Networks not only aids in understanding conditional dependencies but also allows for intuitive visualization of the complex relationships within the dataset. This visual representation enhances our ability to communicate findings and insights effectively.
5. **Applicability to Probabilistic Reasoning:**Given that our project involves predicting postnatal depression, a condition with inherent uncertainty, Bayesian Networks are particularly apt for probabilistic reasoning. They allow us to model and quantify uncertainties associated with predictions, providing a more realistic and nuanced understanding of the outcomes.

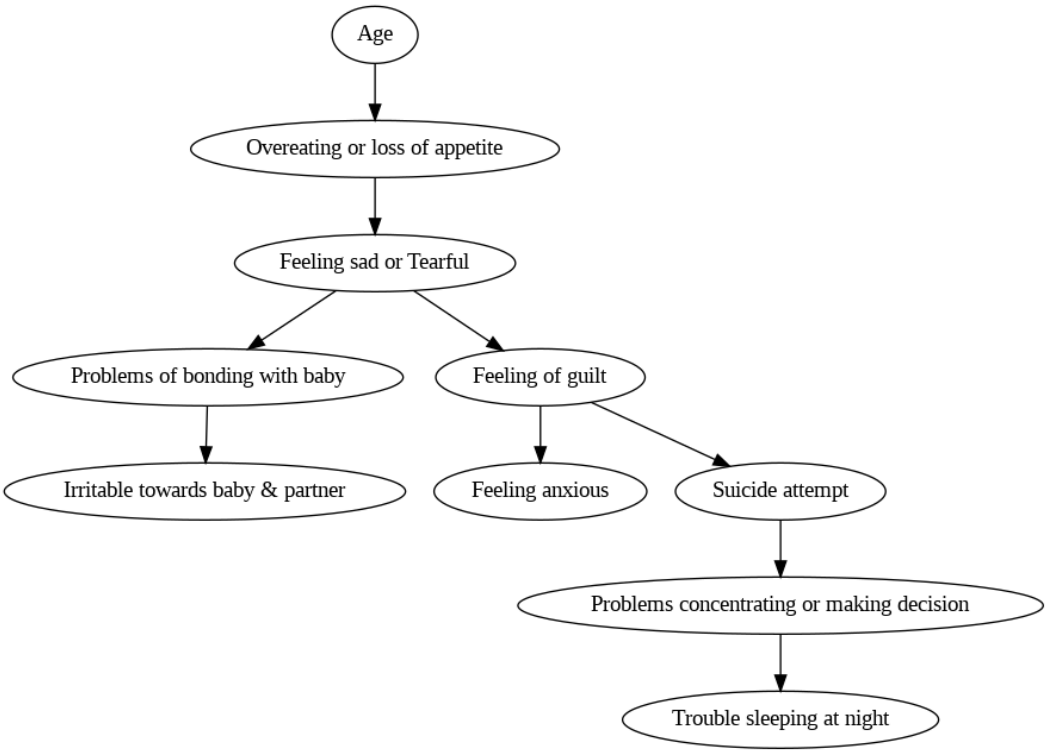


Fig 5 - Bayesian Network with Age as the start node and 'Feeling anxious' as the target node

Representation Using Hill Climb Search:

The graph generated through the Hill Climb Search algorithm encapsulates a Hill Climbing Bayesian Network, providing a visual representation of inferred conditional dependencies among variables relevant to postnatal depression. Employing an iterative approach, the algorithm explores different network structures, refining and climbing towards the most probable configuration based on the provided dataset.

This Bayesian Network graph effectively captures the directed relationships identified by the Hill Climb Search, showcasing the dependencies between attributes such as age, symptoms, and risk factors. Each edge in the graph signifies a potential influence or dependency, with the structure reflecting the most likely connections based on the observed data.

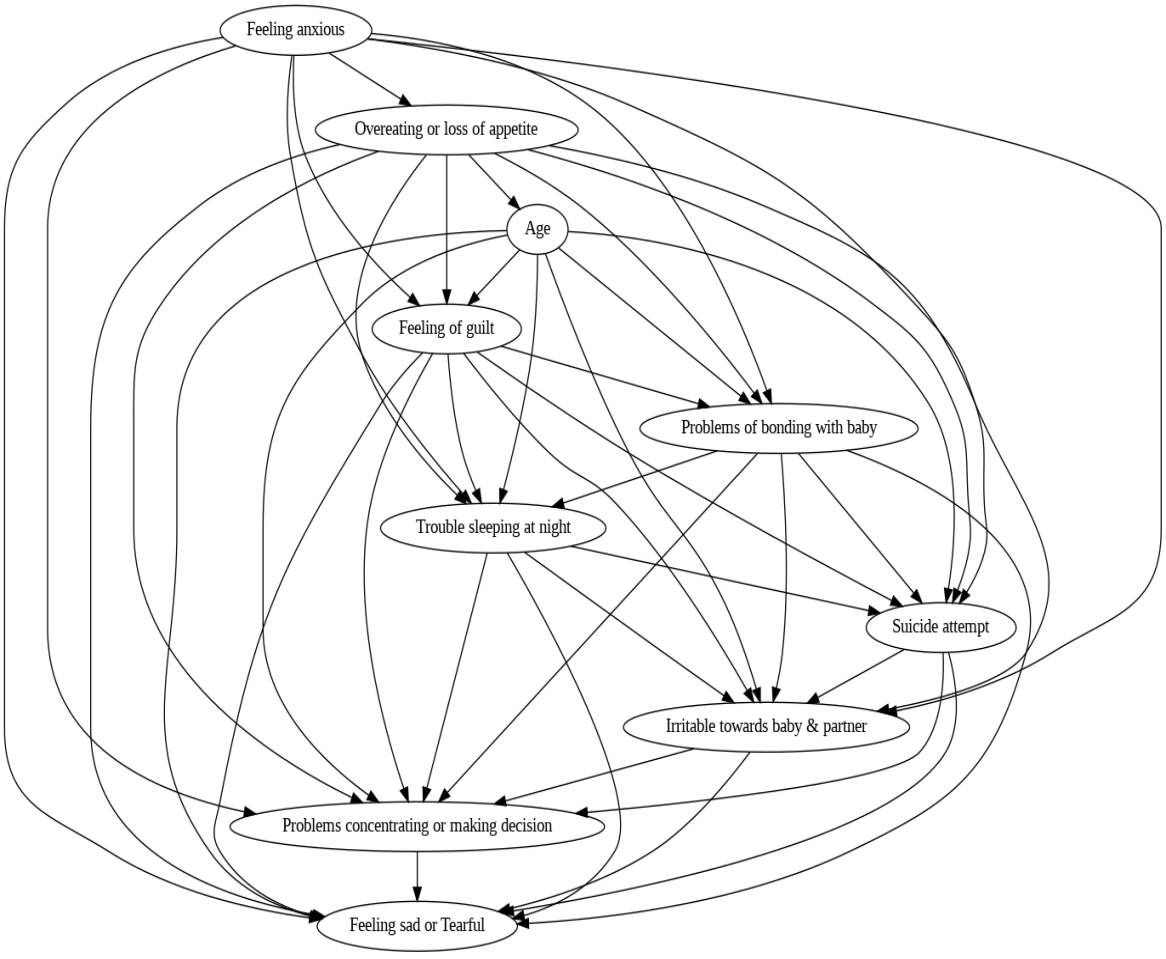


Fig 6 - Representation using a heuristic hill climb search method

The Hill Climb Search representation offers a valuable insight into the complex web of interactions within the dataset. It serves as a graphical model that not only provides a visual overview of the identified relationships but also facilitates probabilistic inference, enabling a deeper understanding of how variables interconnect in the context of postnatal mental health.

Subsequently, using the Hill Climb Search object (hc1), we deduce the structure of a Bayesian Network. This entails refining our comprehension of how different factors within the dataset may be interlinked. The algorithm systematically adjusts the network structure based on patterns observed in the data, resulting in the derived structure denoted as est_model1.

To enhance the accessibility of our findings, we opt to visually represent the estimated Bayesian Network. This graphical depiction provides an intuitive overview of potential connections and dependencies between different elements associated with postnatal depression. In summary, these steps guide our exploration and representation of likely relationships within the dataset without delving into technical details.

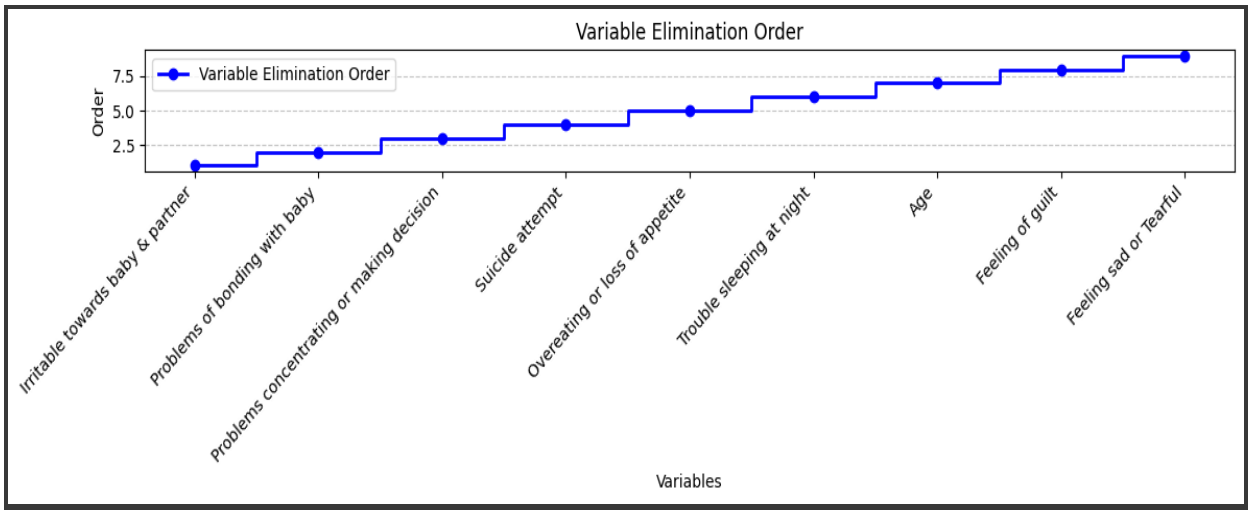
Inference

In the process of drawing insights and predictions from our Bayesian Network model, we employ a powerful technique known as variable elimination. Variable elimination is a systematic method for computing marginal and conditional probabilities efficiently. It allows us to answer questions about specific variables or events given observed evidence.

The Bayesian Network structure, which we derived through Hill Climb Search, contains information about how different variables are connected. Variable elimination leverages this structure to streamline the computation of probabilities. By eliminating irrelevant variables and focusing on the relevant ones, we can expedite the inference process while maintaining accuracy.

For example, we can use variable elimination to calculate the probability of experiencing anxiety given evidence of specific symptoms and demographic information. This technique not only enhances the efficiency of our probabilistic reasoning but also aligns with the goal of providing meaningful insights into postnatal depression.

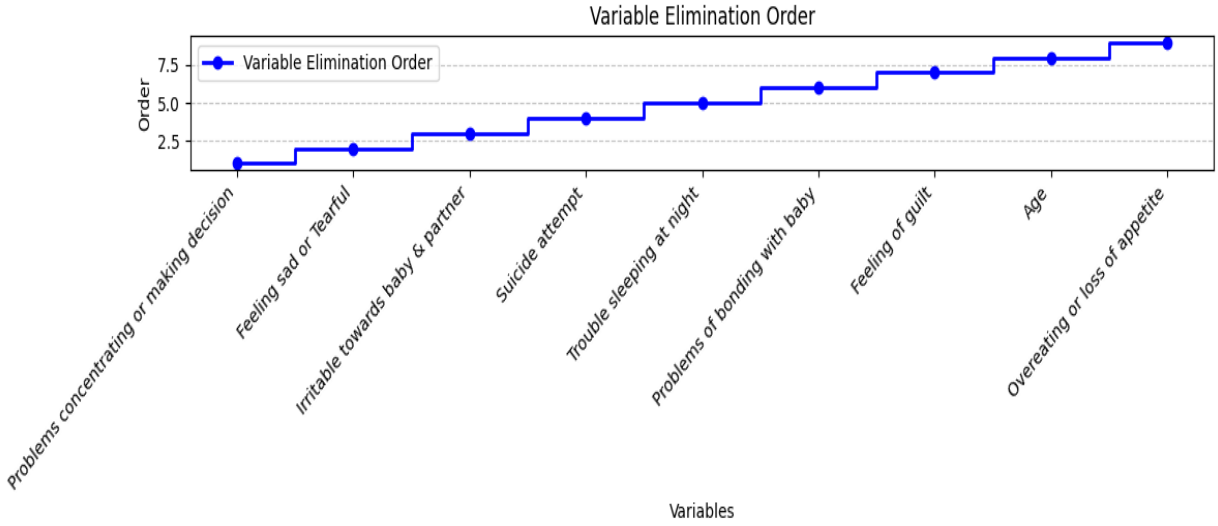
Variable Elimination Order through the Bayesian Network:



The Variable Elimination Order generated using the Bayesian Model reflects a thoughtful progression from emotional factors to cognitive and physiological aspects. Starting with irritability and bonding issues, the order sequentially explores concentration problems, severe

outcomes like suicide attempts, physiological indicators such as eating patterns and sleep troubles, and concludes with demographic factors and deep emotional states. This systematic approach ensures a holistic examination of variables, contributing to a nuanced understanding of postnatal mental health dynamics.

Variable Elimination Order through the Heuristic Hill Search Algorithm



Variable Elimination Order derived through Hill Climb Search takes a different route. It prioritizes concentration problems, followed by emotional states, irritability, and severe outcomes. The order then explores sleep troubles, bonding issues, feelings of guilt, demographic factors, and physiological indicators. This approach, influenced by the Hill Climb Search algorithm, unveils a distinct perspective on the interconnectedness of variables, emphasizing concentration problems as an early focal point.

Comparative study of the Conditional Probability Distribution using Bayesian model and the Hill Climb Search.

A Conditional Probability Distribution (CPD) specifies the likelihood of a variable's values based on observed evidence. It shows how the probability of an outcome changes when certain conditions are known, providing a key tool in probabilistic modeling.

For instance, consider a Bayesian network representing weather conditions. The CPD for the variable "Rain" would describe the probability of rain based on observed evidence such as the humidity level or the presence of dark clouds. If we observe high humidity, the CPD might indicate a higher probability of rain compared to a scenario with low humidity.

1. **Inference for Feeling Anxious given Age:**

This probabilistic inference examines the likelihood of an individual feeling anxious based on the observed evidence of their age. Specifically, it seeks to understand how the probability distribution of feeling anxious is influenced when we know the person's age is in category 1 (30-35 years). The inference process allows us to quantify the impact of age as evidence on the probability of experiencing feelings of anxiety.

Bayesian Model Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.3514
Feeling anxious(1)	0.6486

In our Bayesian model, the Conditional Probability Distribution (CPD) for "Feeling anxious" reveals a probability of 0.3514 for not feeling anxious and 0.6486 for feeling anxious. This signifies the likelihood of experiencing different emotional states given the model's learned relationships.

Hill Climb Search Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.3659
Feeling anxious(1)	0.6341

Conversely, employing the Hill Climb algorithm results in a slightly adjusted CPD. The probabilities shift to 0.3659 for not feeling anxious and 0.6341 for feeling anxious. This nuanced alteration indicates the algorithm's influence on the learned relationships within the Bayesian network.

Notably, the probabilities for feeling anxious or not feeling anxious given age evidence are closely aligned between the Bayesian Model and the Hill Climb Algorithm. The marginal differences underscore the consistency in their estimations, emphasizing the robustness of these results across distinct learning methodologies.

2. **Inference for "Feeling Anxious" Given Overeating or Loss of Appetite:**

Utilizing probabilistic inference, we estimate the probability distribution of feeling anxious based on observed evidence of overeating or loss of appetite. This analysis provides insights into the likelihood of experiencing anxiety in the context of eating behaviors.

Bayesian Model Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.3530
Feeling anxious(1)	0.6470

The Bayesian model suggests a probability of 0.6470 for feeling anxious when there is evidence of Overeating or Loss of Appetite. This probability provides insights into the potential correlation between these eating behaviors and an increased likelihood of experiencing feelings of anxiety according to the Bayesian inference.

Hill Climb Search Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.2627
Feeling anxious(1)	0.7373

The probabilities indicate a notable likelihood of feeling anxious (0.7373) when there's evidence of Overeating or Loss of Appetite, emphasizing a potential correlation between these eating behaviors and heightened feelings of anxiety according to the Hill Climb inference.
Both the Bayesian model and Hill Climb algorithm yield similar outcomes for "Feeling Anxious" given Overeating or Loss of Appetite. With probabilities around 0.64 to 0.74, both methods indicate a potential correlation between these eating behaviors and an increased likelihood of anxiety. The consistency in results across different inference approaches enhances the reliability of the findings.

3. **Inference for "Feeling Anxious" Given Age and Feeling of Guilt:**

This query delves into the estimation of the probability distribution for "Feeling Anxious" based on two specified conditions: "Age=0" (0 represents the class 25-30 years) and "Feeling of guilt=1." The objective is to uncover how these specific circumstances influence the likelihood of experiencing feelings of anxiety.

Bayesian Model Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.1079
Feeling anxious(1)	0.8921

The probabilities indicate that, given the observed conditions of age being 0 and the presence of feelings of guilt, there is a substantial likelihood of feeling anxious. The probability of 0.8921 for

feeling anxious underscores the strong association between these specific conditions and an increased probability of experiencing feelings of anxiety, as inferred by the Bayesian model.

Hill Climb Search Results:

Feeling anxious	phi(Feeling anxious)
Feeling anxious(0)	0.1759
Feeling anxious(1)	0.8241

Considering the observed conditions of age being 0 and the presence of feelings of guilt, the Hill Climb algorithm indicates a notable likelihood of feeling anxious. The probability of 0.8241 for feeling anxious highlights a discernible association between these specific conditions and an increased probability of experiencing feelings of anxiety, as inferred by the Hill Climb algorithm. The results align with the Bayesian model, albeit with slight differences, reinforcing the consistency of findings across distinct inference methodologies.

Comparison of the Inference results:

Query	Bayesian Model	Heurestic Hill Climb Search
Feeling Anxious given Age [30-35 years]	P(Anxious)(0) = 0.3514 P(Anxious)(1) = 0.6486	P(Anxious)(0) = 0.3659 P(Anxious)(1) = 0.6341
Feeling Anxious given overeating or loss of appetite [1]	P(Anxious)(0) = 0.3530 P(Anxious)(1) = 0.6470	P(Anxious)(0) = 0.2627 P(Anxious)(1) = 0.7373
Feeling Anxious given Age [25-30 years] and Feeling of guilt [1]	P(Anxious)(0) = 0.1079 P(Anxious)(1) = 0.8921	P(Anxious)(0) = 0.1759 P(Anxious)(1) = 0.8241

Conclusion

The comparative analysis between tree-based and hill-climb search-based Bayesian Networks in this study offers valuable insights into the probabilistic modeling of postnatal depression. While both methods demonstrate a robust capacity to model and predict the likelihood of postnatal depression based on various demographic and symptomatic variables, slight differences in their conditional probability distributions highlight the nuances in their inferential capabilities. The consistency of results across both methodologies, particularly in the context of key variables like age, feelings of guilt, and eating behaviors, underscores the reliability and applicability of probabilistic graphical models in understanding complex health conditions. This study not only contributes to the predictive analysis of postnatal depression but also reinforces the significance of employing sophisticated analytical tools like Bayesian Networks for in-depth exploration and interpretation of medical datasets, ultimately aiding in the development of targeted interventions and support systems.

Colab Link:
https://colab.research.google.com/drive/1M_6vncYnVu9E1qRP5bpaWKzO-a1eK44D?usp=sharing

pgm-miniproject-51-68-191-196-177

December 9, 2023

0.1 Bayesian Network

```
<p style="padding: 5px;color:white;"><b> What is Bayesian Network? </b></p>
```

0.1.1 Bayesian Network

Bayesian Network is a kind of Graphical model called a Directed Acyclic Graph or DAG. It means all the edges (arcs) in the graph are directed, and there are no cycles. Each edge in the Bayesian Network encodes a joint probability factorization which would ne the the joint distribution of all nodes (variables/vertices).

$p(A, B, C) = P(A|B).P(C|B).P(C)$

0.1.2 Inference

The inference would be the definition of a task that is responsible to compute the probability of each node in the Bayesian Network, assuming the known values of the other nodes. There are two different types of inference; exact and approximate.

Exact Inference Different exact approaches to finding the node's probability have been applied in this notebook, such as variable elimination and junction tree algorithm.

Importing libraries

```
<p style="padding: 5px;color:white;"><b> Used libraries for the notebook </b></p>
```

[ ]:
```
!pip install pgmpy
```

```
Requirement already satisfied: pgmpy in /usr/local/lib/python3.10/dist-packages
(0.1.24)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from pgmpy) (3.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-
```

```
packages (from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (2.1.0+cu118)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (from pgmpy) (0.14.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (4.66.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-
packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->pgmpy) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmpy) (3.2.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-
packages (from statsmodels->pgmpy) (0.5.3)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch->pgmpy) (3.13.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch->pgmpy) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch->pgmpy) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch->pgmpy) (2.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.2->statsmodels->pgmpy) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch->pgmpy) (1.3.0)
```

[ ]:
```
from pgmpy.models import BayesianNetwork
```

[ ]:
```
import os
import random
import warnings
import numpy as np
import pandas as pd
```

```
import networkx as nx
from scipy.io import arff
import matplotlib.pyplot as plt
import pgmpy.estimators as ests
from pgmpy.estimators import TreeSearch,BicScore
from pgmpy.models import BayesianNetwork
from pgmpy.metrics import structure_score
from sklearn.metrics import accuracy_score
from pgmpy.inference import BeliefPropagation
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from pgmpy.inference import VariableElimination
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
random_state = 123
np.random.seed(random_state)
warnings.simplefilter('ignore')
np.set_printoptions(precision=2, suppress=True)
```

1 Importing dataset

[ ]:
```
df=pd.read_csv('/content/post natal data.csv')
```

[ ]:
```
df.isnull().sum(axis=0)
```

[ ]:
```
Timestamp                                  0
Age                                        0
Feeling sad or Tearful                     0
Irritable towards baby & partner           6
Trouble sleeping at night                  0
Problems concentrating or making decision    12
Overeating or loss of appetite             0
Feeling anxious                            0
Feeling of guilt                           9
Problems of bonding with baby              0
Suicide attempt                            0
dtype: int64
```

[ ]:
```
df = df.dropna()
```

[ ]:
```
df = df.drop('Timestamp', axis=1)
```

[ ]:
```
df.head()
```

[ ]:
```
     Age Feeling sad or Tearful Irritable towards baby & partner  \
0  35-40                    Yes                                Yes
1  40-45                    Yes                                 No
2  35-40                    Yes                                 No
3  35-40                    Yes                                Yes
4  40-45                    Yes                                 No

  Trouble sleeping at night Problems concentrating or making decision  \
0   Two or more days a week                                        Yes
1                        No                                        Yes
2                       Yes                                        Yes
3                       Yes                                        Yes
4   Two or more days a week                                        Yes

  Overeating or loss of appetite Feeling anxious Feeling of guilt  \
0                            Yes             Yes               No
1                            Yes              No              Yes
2                            Yes             Yes               No
3                             No             Yes            Maybe
4                             No             Yes               No

  Problems of bonding with baby Suicide attempt
0                           Yes             Yes
1                           Yes              No
2                     Sometimes              No
3                            No              No
4                           Yes              No
```

[ ]:
```
for column in df.columns:
    unique_values = df[column].unique()
    print(f"Unique values for {column}:\n{unique_values}\n")
```

```
Unique values for Age:
['35-40' '40-45' '30-35' '45-50' '25-30']

Unique values for Feeling sad or Tearful:
['Yes' 'No' 'Sometimes']

Unique values for Irritable towards baby & partner:
['Yes' 'No' 'Sometimes']

Unique values for Trouble sleeping at night:
['Two or more days a week' 'No' 'Yes']

Unique values for Problems concentrating or making decision:
```

```
['Yes' 'No' 'Often']

Unique values for Overeating or loss of appetite:
['Yes' 'No' 'Not at all']

Unique values for Feeling anxious:
['Yes' 'No']

Unique values for Feeling of guilt:
['No' 'Yes' 'Maybe']

Unique values for Problems of bonding with baby:
['Yes' 'Sometimes' 'No']

Unique values for Suicide attempt:
['Yes' 'No' 'Not interested to say']
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Feeling sad or Tearful', 'Irritable towards baby & partner',
       'Trouble sleeping at night',
       'Problems concentrating or making decision',
       'Overeating or loss of appetite', 'Feeling anxious', 'Feeling of guilt',
       'Problems of bonding with baby', 'Suicide attempt'],
      dtype='object')
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Loop through each column and label encode if it's categorical
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = label_encoder.fit_transform(df[column])
```

[ ]:
```
df.head()
```

[ ]:
```
   Age  Feeling sad or Tearful  Irritable towards baby & partner  \
0    2                       2                                 2
1    3                       2                                 0
2    2                       2                                 0
3    2                       2                                 2
4    3                       2                                 0

   Trouble sleeping at night  Problems concentrating or making decision  \
0                          1                                          2
1                          0                                          2
```

```
2                          2                                       2
3                          2                                       2
4                          1                                       2

   Overeating or loss of appetite  Feeling anxious  Feeling of guilt  \
0                               2                1                 1
1                               2                0                 2
2                               2                1                 1
3                               0                1                 0
4                               0                1                 1

   Problems of bonding with baby  Suicide attempt
0                              2                2
1                              2                0
2                              1                0
3                              0                0
4                              2                0
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Feeling sad or Tearful', 'Irritable towards baby & partner',
       'Trouble sleeping at night',
       'Problems concentrating or making decision',
       'Overeating or loss of appetite', 'Feeling anxious', 'Feeling of guilt',
       'Problems of bonding with baby', 'Suicide attempt'],
      dtype='object')
```

[ ]:
```
for column in df.columns:
    unique_values = df[column].unique()
    print(f"Unique values for {column}:\n{unique_values}\n")
```

```
Unique values for Age:
[2 3 1 4 0]

Unique values for Feeling sad or Tearful:
[2 0 1]

Unique values for Irritable towards baby & partner:
[2 0 1]

Unique values for Trouble sleeping at night:
[1 0 2]

Unique values for Problems concentrating or making decision:
[2 0 1]

Unique values for Overeating or loss of appetite:
```

```
[2 0 1]

Unique values for Feeling anxious:
[1 0]

Unique values for Feeling of guilt:
[1 2 0]

Unique values for Problems of bonding with baby:
[2 1 0]

Unique values for Suicide attempt:
[2 0 1]
```

1.1 Defining the input data

```
<p style="padding: 5px;color:white;"><b> return X, y </b></p>
```

[ ]:
```
X = df.drop(columns=['Feeling anxious'],axis=1)
y = df['Feeling anxious']
```

1.2 Feature engineering

```
<p style="padding: 5px;color:white;"><b> Variable elimination due to selecting the highest sco
```

[ ]:
```
fs = SelectKBest(score_func=chi2, k=8)
f_best = fs.fit_transform(X, y)
top_features = sorted(zip(list(X.columns), fs.scores_), key=lambda x: x[1],␣
 ↪reverse=True)
top_features
```

[ ]:
```
[('Feeling of guilt', 72.95242988938031),
 ('Problems of bonding with baby', 53.2862442777557),
 ('Overeating or loss of appetite', 35.41955077749806),
 ('Problems concentrating or making decision', 29.562012746898066),
 ('Trouble sleeping at night', 12.849097175926513),
 ('Age', 3.4713124038612424),
 ('Suicide attempt', 2.3357702885200227),
 ('Feeling sad or Tearful', 1.966983680554224),
 ('Irritable towards baby & partner', 0.8765237770901146)]
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Feeling sad or Tearful', 'Irritable towards baby & partner',
       'Trouble sleeping at night',
       'Problems concentrating or making decision',
       'Overeating or loss of appetite', 'Feeling anxious', 'Feeling of guilt',
```

```
       'Problems of bonding with baby', 'Suicide attempt'],
      dtype='object')
```

```
<p style="padding: 5px;color:white;"><b> Every variable seems of good importanc in the chi squa
```

1.3 Joint probability

```
<p style="padding: 5px;color:white;"><b> Simple example </b></p>
```

[ ]:
```
data = df
```

[ ]:
```
joint_probability = data.groupby([data.columns], axis=1).size()/data.shape[0]
pd.DataFrame(joint_probability)
```

[ ]:

	0
Age	0.000671
Feeling anxious	0.000671
Feeling of guilt	0.000671
Feeling sad or Tearful	0.000671
Irritable towards baby & partner	0.000671
Overeating or loss of appetite	0.000671
Problems concentrating or making decision	0.000671
Problems of bonding with baby	0.000671
Suicide attempt	0.000671
Trouble sleeping at night	0.000671

This could be an indication that the data is either highly uniform or that the specific groupings of values in each variable are evenly distributed.

2 Bayesian Models

2.1 DAG

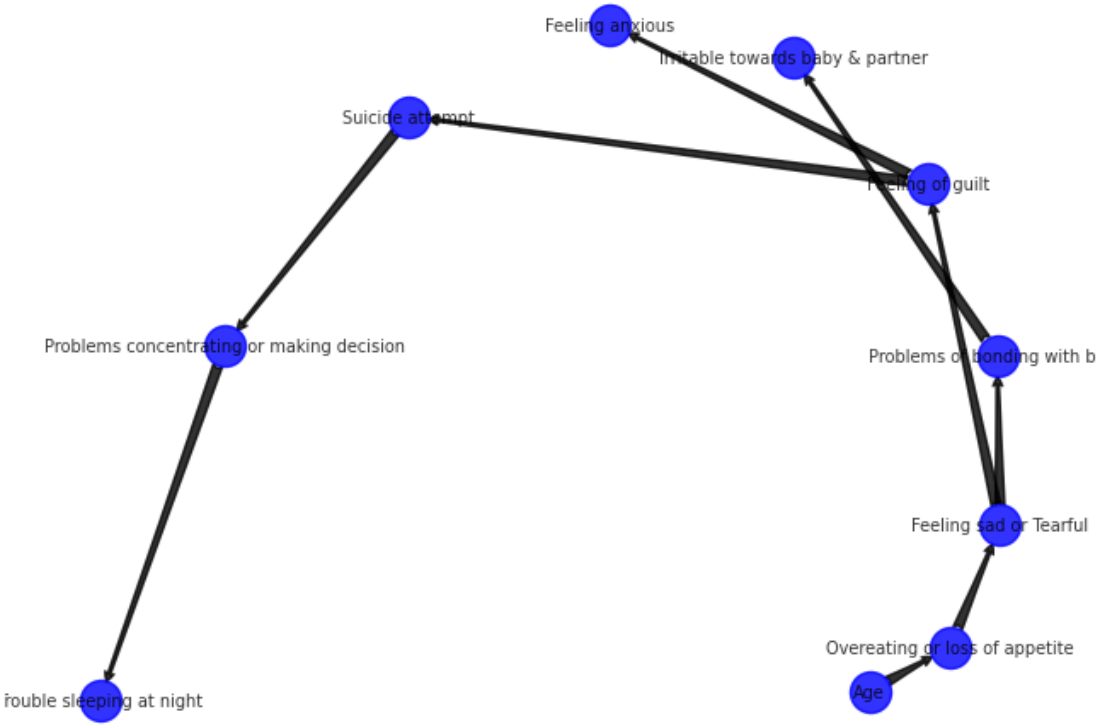
```
<p style="padding: 5px;color:white;"><b> Probabilistic Directed Acyclic Graphical - pDAG </b></
```

To have the Bayesian Network over this example, we need to define dependencies between pairwise variables.

For the structure learning techniques, There are various standards such as Hill Climb search, Structure score, Tree search, Exhaustive search, etc. The one I used for this illustration was the Tree search.

[ ]:
```
est = TreeSearch(data, root_node='Age')
dag = est.estimate(estimator_type='chow-liu')
pos = nx.spiral_layout(dag)
nx.draw(dag, pos=pos, with_labels=True,node_color='b', font_size=7,␣
 ↪arrowstyle='fancy', alpha=0.8)
```

```
Building tree:   0%|          | 0/45.0 [00:00<?, ?it/s]
```



1. **Compute Mutual Information:**
   - Calculate mutual information (I(X_i; X_j)) for each pair of variables (X_i) and (X_j) in the dataset. This quantifies the strength of the relationship between variables.
2. **Build Fully Connected Graph:**
   - Form a fully connected undirected graph, where nodes represent variables and edges denote the mutual information between corresponding variable pairs.
3. **Compute Maximum Spanning Tree:**
   - Apply a maximum spanning tree algorithm (e.g., Kruskal's or Prim's) to identify the tree structure that connects all nodes while minimizing edge weights. Edges in this tree represent the strongest pairwise dependencies.
4. **Convert to DAG:**
   - Designate a root node and orient edges to create a directed acyclic graph (DAG). The root becomes the parent, and other nodes are its children in the directed structure.
5. **Resulting Bayesian Network:**
   - The DAG signifies the Bayesian Network structure. Nodes represent variables, and directed edges denote conditional dependencies between variables. The algorithm assumes relationships can be approximated by a tree structure.
6. **Parameter Estimation:**
   - Estimate Bayesian Network parameters based on the dataset.
   - For each DAG node, estimate the conditional probability distribution given its parent(s).
7. **Bayesian Network Construction:**

- Combine the learned DAG structure with estimated parameters to construct the complete Bayesian Network.

2.2 DAG INFOs

[ ]:
```
print("Nodes: ", dag.nodes())
print("---------------")
print("Edges: ", dag.edges())

print("--------" "\n")

nx.to_pandas_edgelist(dag)
```

```
Nodes:  ['Age', 'Overeating or loss of appetite', 'Feeling sad or Tearful',
'Problems of bonding with baby', 'Feeling of guilt', 'Irritable towards baby &
partner', 'Feeling anxious', 'Suicide attempt', 'Problems concentrating or
making decision', 'Trouble sleeping at night']
---------------
Edges:  [('Age', 'Overeating or loss of appetite'), ('Overeating or loss of
appetite', 'Feeling sad or Tearful'), ('Feeling sad or Tearful', 'Problems of
bonding with baby'), ('Feeling sad or Tearful', 'Feeling of guilt'), ('Problems
of bonding with baby', 'Irritable towards baby & partner'), ('Feeling of guilt',
'Feeling anxious'), ('Feeling of guilt', 'Suicide attempt'), ('Suicide attempt',
'Problems concentrating or making decision'), ('Problems concentrating or making
decision', 'Trouble sleeping at night')]
--------
```

[ ]:

	source	target	weight
0	Age	Overeating or loss of appetite	None
1	Overeating or loss of appetite	Feeling sad or Tearful	None
2	Feeling sad or Tearful	Problems of bonding with baby	None
3	Feeling sad or Tearful	Feeling of guilt	None
4	Problems of bonding with baby	Irritable towards baby & partner	None
5	Feeling of guilt	Feeling anxious	None
6	Feeling of guilt	Suicide attempt	None
7	Suicide attempt	Problems concentrating or making decision	None

```
8                  Trouble sleeping at night   None
```

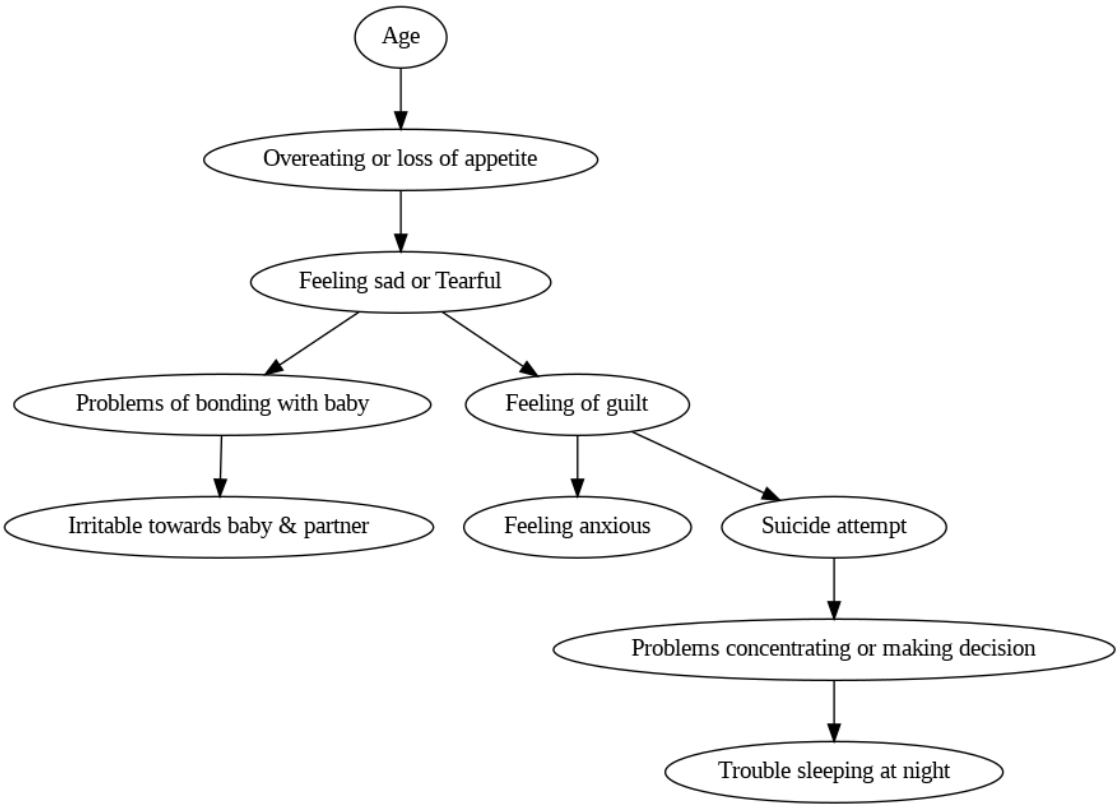
2.3 Bayesian Network

```
<p style="padding: 5px;color:white;"><b> Return the (Estimation of) Conditional probability bas
```

[ ]:
```
from IPython.display import Image, display
```

[ ]:
```
hc1 = TreeSearch(data, root_node='Age')
est_model1 = hc1.estimate()
display(Image((nx.drawing.nx_pydot.to_pydot(est_model1)).create_png()))
```

```
Building tree:   0%|          | 0/45.0 [00:00<?, ?it/s]
```



The code is designed to explore and visualize the hierarchical structure within a dataset. It uses a custom TreeSearch class to estimate the hierarchy, with the root node set to 'Age.' The resulting hierarchical model is represented as a directed graph, and the code converts it into a visual format for better understanding.

This visualization aids in comprehending the relationships and dependencies within the data, facilitating a more intuitive exploration of its organizational structure.

The combination of the TreeSearch class and NetworkX provides a powerful tool for hierarchical

analysis and visualization in a Python environment.

[ ]:
```
model = BayesianNetwork(dag)
model.fit(data)
model.get_cpds()
```

[ ]:
```
[<TabularCPD representing P(Age:5) at 0x795d51f6bd90>,
 <TabularCPD representing P(Overeating or loss of appetite:3 | Age:5) at
0x795d4f410250>,
 <TabularCPD representing P(Feeling sad or Tearful:3 | Overeating or loss of
appetite:3) at 0x795d4f410610>,
 <TabularCPD representing P(Problems of bonding with baby:3 | Feeling sad or
Tearful:3) at 0x795d4f410a60>,
 <TabularCPD representing P(Feeling of guilt:3 | Feeling sad or Tearful:3) at
0x795d4f4108b0>,
 <TabularCPD representing P(Irritable towards baby & partner:3 | Problems of
bonding with baby:3) at 0x795d4f410bb0>,
 <TabularCPD representing P(Feeling anxious:2 | Feeling of guilt:3) at
0x795d4f410a00>,
 <TabularCPD representing P(Suicide attempt:3 | Feeling of guilt:3) at
0x795d4f4103d0>,
 <TabularCPD representing P(Problems concentrating or making decision:3 |
Suicide attempt:3) at 0x795d4f410400>,
 <TabularCPD representing P(Trouble sleeping at night:3 | Problems concentrating
or making decision:3) at 0x795d4f410940>]
```

Retrieves the Conditional Probability Distributions (CPDs) for each variable in the Bayesian Network using the get_cpds method. CPDs provide the probability of a variable given its parents in the network.

2.4 Inference

2.4.1 Variable Elimination

```
<p style="padding: 5px;color:white;"><b> Initialize the inference by Variable Elimination metho
```

[ ]:
```
infer = VariableElimination(model)
q = infer.query(variables=['Feeling anxious'], evidence={"Age": 1}, joint=False)
print(q['Feeling anxious'])
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.3514 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.6486 |
+--------------------+------------------------+
```

[ ]:
```
infer = VariableElimination(model)
q1 = infer.query(variables=['Feeling anxious'], evidence={"Overeating or loss␣
 ↪of appetite": 1}, joint=False)
print(q1['Feeling anxious'])
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.3530 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.6470 |
+--------------------+------------------------+
```

[ ]:
```
infer = VariableElimination(model)
q2 = infer.query(variables=['Feeling anxious'], evidence={'Age':0,'Feeling of␣
 ↪guilt':1}, joint=False)
print(q2['Feeling anxious'])
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.1079 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.8921 |
+--------------------+------------------------+
```

[ ]:
```
from pgmpy.estimators import MaximumLikelihoodEstimator
for variable in model.nodes:
    cpd = MaximumLikelihoodEstimator(model, data).estimate_cpd(variable)
    model.add_cpds(cpd)

# Function to calculate Min-fill heuristic for a variable in a graph
def min_fill_heuristic(graph, variable):
    neighbors = set(graph.neighbors(variable))
    fill_edges = set()

    for neighbor1 in neighbors:
        for neighbor2 in neighbors:
            if neighbor1 != neighbor2 and not graph.has_edge(neighbor1,␣
 ↪neighbor2):
                fill_edges.add((neighbor1, neighbor2))

    return len(fill_edges)

# Find the variable elimination order using Min-fill heuristic
elimination_order_ = []
remaining_variables_ = set(model.nodes)
```

```
# Reverse the order to start eliminating from variables with fewer dependencies
while remaining_variables_:
    variable_scores_ = {variable: min_fill_heuristic(model, variable) for␣
 ↪variable in remaining_variables_}
    min_score_variable_ = min(variable_scores_, key=variable_scores_.get)

    # Add "DEATH_EVENT" at the end of the elimination order
    if min_score_variable_ != 'Feeling anxious':
        elimination_order_.append(min_score_variable_)

    remaining_variables_.remove(min_score_variable_)

# Print the calculated variable elimination order
print("Variable Elimination Order:", elimination_order_)

# Perform Variable Elimination with the calculated order
infer = VariableElimination(model)
```

```
WARNING:pgmpy:Replacing existing CPD for Age
WARNING:pgmpy:Replacing existing CPD for Overeating or loss of appetite
WARNING:pgmpy:Replacing existing CPD for Feeling sad or Tearful
WARNING:pgmpy:Replacing existing CPD for Problems of bonding with baby
WARNING:pgmpy:Replacing existing CPD for Feeling of guilt
WARNING:pgmpy:Replacing existing CPD for Irritable towards baby & partner
WARNING:pgmpy:Replacing existing CPD for Feeling anxious
WARNING:pgmpy:Replacing existing CPD for Suicide attempt
WARNING:pgmpy:Replacing existing CPD for Problems concentrating or making
decision
WARNING:pgmpy:Replacing existing CPD for Trouble sleeping at night
```

```
Variable Elimination Order: ['Irritable towards baby & partner', 'Problems of
bonding with baby', 'Problems concentrating or making decision', 'Suicide
attempt', 'Overeating or loss of appetite', 'Trouble sleeping at night', 'Age',
'Feeling of guilt', 'Feeling sad or Tearful']
```

Updating the Conditional Probability Distributions (CPDs) in a Bayesian Network using Maximum Likelihood Estimation (MLE) and then determining an efficient variable elimination order based on the Min-fill heuristic.

The code iterates through each variable in the network, estimates its CPD using MLE, and incorporates the updated CPD into the model. Following this, the Min-fill heuristic is applied to find an optimal order for variable elimination, considering the minimal number of edges to be added.

The resulting elimination order is printed, and Variable Elimination is performed on the Bayesian Network using this order. This process streamlines probabilistic inference by eliminating variables strategically, enhancing computational efficiency.

2.5 Prediction

<p style="padding: 5px;color:white;"><b> Initialize the inference by Variable Elimination metho

[ ]:
```
from pgmpy.models import BayesianNetwork
import pandas as pd
from IPython.display import Image, display
from pgmpy.estimators import HillClimbSearch, BicScore, PC, K2Score
from pgmpy.estimators import BayesianEstimator
import networkx as nx
```

[ ]:
```
test = np.squeeze(data.iloc[:, -1].values)
predict_data = data.iloc[:, :-1]
y_pred = model.predict(predict_data)
print(accuracy_score(np.squeeze(y_pred.values), test))
```

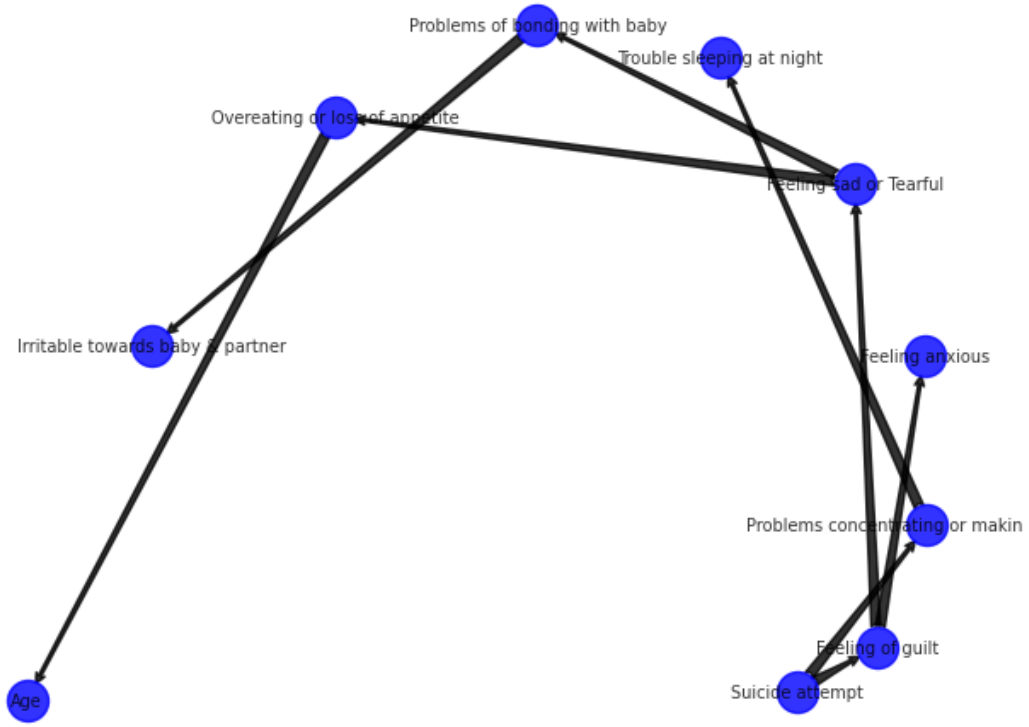
```
  0%|          | 0/319 [00:00<?, ?it/s]
```

```
0.6150234741784038
```

2.6 Start node: Suicide Attempt:

[ ]:
```
est = TreeSearch(data, root_node='Suicide attempt')
dag = est.estimate(estimator_type='chow-liu')
pos = nx.spiral_layout(dag)
nx.draw(dag, pos=pos, with_labels=True,node_color='b', font_size=7,␣
 ↪arrowstyle='fancy', alpha=0.8)
```

```
Building tree:   0%|          | 0/45.0 [00:00<?, ?it/s]
```



[ ]:
```
model = BayesianNetwork(dag)
model.fit(data)
model.get_cpds()
```

[ ]:
```
[<TabularCPD representing P(Suicide attempt:3) at 0x795d4dadd7b0>,
 <TabularCPD representing P(Feeling of guilt:3 | Suicide attempt:3) at
0x795d4dadc190>,
 <TabularCPD representing P(Problems concentrating or making decision:3 |
Suicide attempt:3) at 0x795d4dadc4c0>,
 <TabularCPD representing P(Feeling anxious:2 | Feeling of guilt:3) at
0x795d4dadc910>,
 <TabularCPD representing P(Feeling sad or Tearful:3 | Feeling of guilt:3) at
0x795d4dadd2d0>,
 <TabularCPD representing P(Trouble sleeping at night:3 | Problems concentrating
or making decision:3) at 0x795d4dadd3f0>,
 <TabularCPD representing P(Problems of bonding with baby:3 | Feeling sad or
Tearful:3) at 0x795d4dadd480>,
 <TabularCPD representing P(Overeating or loss of appetite:3 | Feeling sad or
Tearful:3) at 0x795d4dadd390>,
 <TabularCPD representing P(Irritable towards baby & partner:3 | Problems of
```

```
bonding with baby:3) at 0x795d4dadc550>,
 <TabularCPD representing P(Age:5 | Overeating or loss of appetite:3) at
0x795d4dade0e0>]
```

[ ]:
```
infer = VariableElimination(model)
q = infer.query(variables=['Feeling anxious'], evidence={"Overeating or loss of␣
 ↪appetite": 1.}, joint=False)
print(q['Feeling anxious'])
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.3530 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.6470 |
+--------------------+------------------------+
```

[ ]:
```
from pgmpy.estimators import MaximumLikelihoodEstimator
for variable in model.nodes:
    cpd = MaximumLikelihoodEstimator(model, data).estimate_cpd(variable)
    model.add_cpds(cpd)

# Function to calculate Min-fill heuristic for a variable in a graph
def min_fill_heuristic(graph, variable):
    neighbors = set(graph.neighbors(variable))
    fill_edges = set()

    for neighbor1 in neighbors:
        for neighbor2 in neighbors:
            if neighbor1 != neighbor2 and not graph.has_edge(neighbor1,␣
 ↪neighbor2):
                fill_edges.add((neighbor1, neighbor2))

    return len(fill_edges)

# Find the variable elimination order using Min-fill heuristic
elimination_order = []
remaining_variables = set(model.nodes)

# Reverse the order to start eliminating from variables with fewer dependencies
while remaining_variables:
    variable_scores = {variable: min_fill_heuristic(model, variable) for␣
 ↪variable in remaining_variables}
    min_score_variable = min(variable_scores, key=variable_scores.get)

    # Add "DEATH_EVENT" at the end of the elimination order
    if min_score_variable != 'Feeling anxious':
```

```
        elimination_order.append(min_score_variable)

    remaining_variables.remove(min_score_variable)

# Print the calculated variable elimination order
print("Variable Elimination Order:", elimination_order)

# Perform Variable Elimination with the calculated order
infer = VariableElimination(model)
```

```
WARNING:pgmpy:Replacing existing CPD for Suicide attempt
WARNING:pgmpy:Replacing existing CPD for Feeling of guilt
WARNING:pgmpy:Replacing existing CPD for Problems concentrating or making
decision
WARNING:pgmpy:Replacing existing CPD for Feeling anxious
WARNING:pgmpy:Replacing existing CPD for Feeling sad or Tearful
WARNING:pgmpy:Replacing existing CPD for Trouble sleeping at night
WARNING:pgmpy:Replacing existing CPD for Problems of bonding with baby
WARNING:pgmpy:Replacing existing CPD for Overeating or loss of appetite
WARNING:pgmpy:Replacing existing CPD for Irritable towards baby & partner
WARNING:pgmpy:Replacing existing CPD for Age

Variable Elimination Order: ['Irritable towards baby & partner', 'Problems of
bonding with baby', 'Problems concentrating or making decision', 'Overeating or
loss of appetite', 'Trouble sleeping at night', 'Age', 'Suicide attempt',
'Feeling of guilt', 'Feeling sad or Tearful']
```

Updates Conditional Probability Distributions (CPDs) in a Bayesian Network using Maximum Likelihood Estimation (MLE) and then determines an optimal variable elimination order based on the Min-fill heuristic.

The loop iterates through each variable in the model, estimates its CPD using MLE, and adds the updated CPD to the model. The Min-fill heuristic is then applied to find a strategic variable elimination order, considering the minimal number of edges to be added.

The calculated elimination order is printed, and Variable Elimination is performed on the Bayesian Network using this order, enhancing computational efficiency for probabilistic inference.

2.6.1 Junction Tree

```
<p style="padding: 5px;color:white;"><b>  Using belief propagation </b></p>
```

[ ]:
```
bp = BeliefPropagation(model)
bp.calibrate()
bp.get_clique_beliefs()
```

[ ]:
```
{('Feeling of guilt',
  'Suicide attempt'): <DiscreteFactor representing phi(Suicide attempt:3,
Feeling of guilt:3) at 0x795d4dadff10>,
 ('Feeling of guilt',
```

```
  'Feeling anxious'): <DiscreteFactor representing phi(Feeling anxious:2,
Feeling of guilt:3) at 0x795d4dade5c0>,
 ('Feeling of guilt',
  'Feeling sad or Tearful'): <DiscreteFactor representing phi(Feeling of
guilt:3, Feeling sad or Tearful:3) at 0x795d4dadd540>,
 ('Problems concentrating or making decision',
  'Suicide attempt'): <DiscreteFactor representing phi(Suicide attempt:3,
Problems concentrating or making decision:3) at 0x795d4dade2f0>,
 ('Problems of bonding with baby',
  'Feeling sad or Tearful'): <DiscreteFactor representing phi(Problems of
bonding with baby:3, Feeling sad or Tearful:3) at 0x795d4daddff0>,
 ('Overeating or loss of appetite',
  'Feeling sad or Tearful'): <DiscreteFactor representing phi(Overeating or loss
of appetite:3, Feeling sad or Tearful:3) at 0x795d4dade350>,
 ('Irritable towards baby & partner',
  'Problems of bonding with baby'): <DiscreteFactor representing phi(Irritable
towards baby & partner:3, Problems of bonding with baby:3) at 0x795d4dadf880>,
 ('Problems concentrating or making decision',
  'Trouble sleeping at night'): <DiscreteFactor representing phi(Trouble
sleeping at night:3, Problems concentrating or making decision:3) at
0x795d4dadfd30>,
 ('Overeating or loss of appetite',
  'Age'): <DiscreteFactor representing phi(Overeating or loss of appetite:3,
Age:5) at 0x795d4dade320>}
```

2.7 Prediction

```
<p style="padding: 5px;color:white;"><b> Initialize the inference by Variable Elimination metho
```

[ ]:
```
from pgmpy.models import BayesianNetwork
import pandas as pd
from IPython.display import Image, display
from pgmpy.estimators import HillClimbSearch, BicScore, PC, K2Score
from pgmpy.estimators import BayesianEstimator
import networkx as nx
```

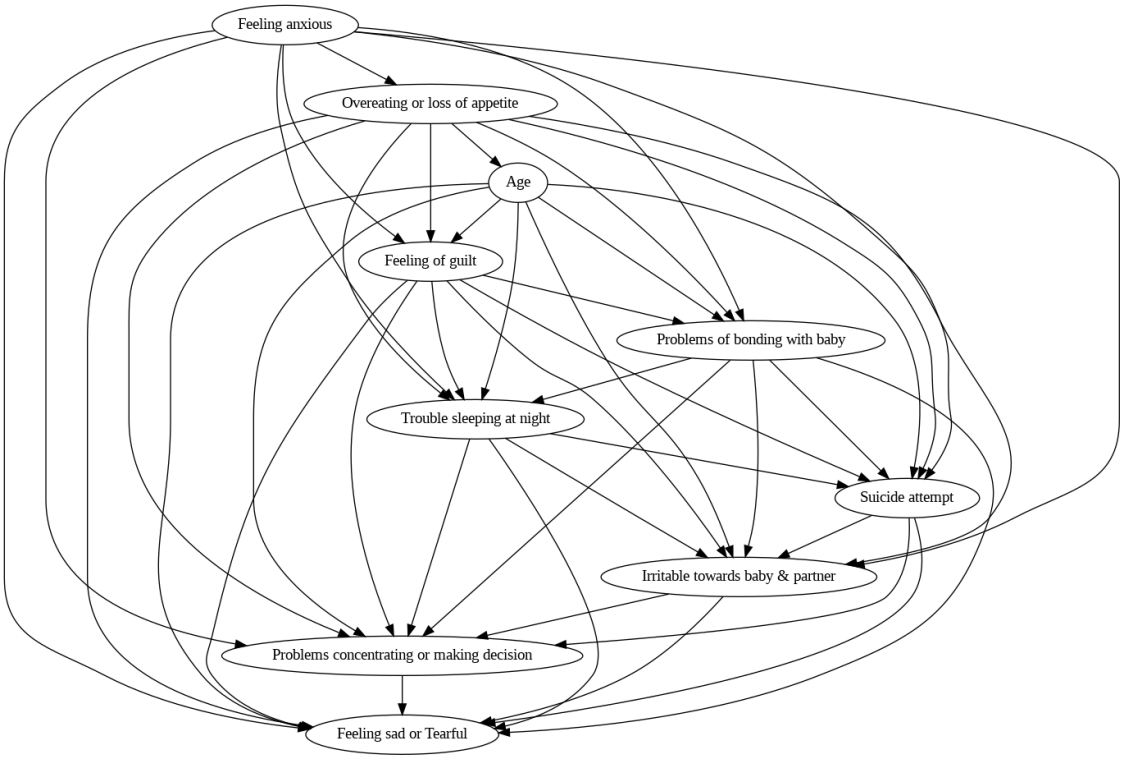
[ ]:
```
test = np.squeeze(data.iloc[:, -1].values)
predict_data = data.iloc[:, :-1]
y_pred = model.predict(predict_data)
print(accuracy_score(np.squeeze(y_pred.values), test))
```

```
  0%|          | 0/319 [00:00<?, ?it/s]
```

```
0.6150234741784038
```

[ ]:
```
hc1 = HillClimbSearch(df, HillClimbSearch(df))
est_model1 = hc1.estimate()
display(Image((nx.drawing.nx_pydot.to_pydot(est_model1)).create_png()))
```

```
0%|          | 0/1000000 [00:00<?, ?it/s]
```



Like a detective trying to figure out the best way to represent relationships in a dataset. It uses a method called "hill climb search" twice to explore and discover the most likely connections between different pieces of information in the dataset (df).

The result is a visual representation of these relationships, kind of like a map of connections, which is then displayed for easier understanding.

So, it's essentially a tool for uncovering and showing the important links within the data.

[ ]:
```
hc = HillClimbSearch(df)
best_model = hc.estimate()
edges = list(best_model.edges())
model = BayesianNetwork(edges)
```

```
0%|          | 0/1000000 [00:00<?, ?it/s]
```

[ ]:
```
# Fitting the data to the model using Maximum Likelihood Estimator
model.fit(df, estimator=MaximumLikelihoodEstimator)

# Doing exact inference using Variable Elimination
infer = VariableElimination(model)
```

[ ]:
```
model.get_cpds
```

[ ]:
```
<bound method BayesianNetwork.get_cpds of
<pgmpy.models.BayesianNetwork.BayesianNetwork object at 0x795d4db5b0a0>>
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Feeling sad or Tearful', 'Irritable towards baby & partner',
       'Trouble sleeping at night',
       'Problems concentrating or making decision',
       'Overeating or loss of appetite', 'Feeling anxious', 'Feeling of guilt',
       'Problems of bonding with baby', 'Suicide attempt'],
      dtype='object')
```

[ ]:
```
print(infer.query(variables=['Feeling anxious'], evidence={'Age':1}))
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.3659 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.6341 |
+--------------------+------------------------+
```

[ ]:
```
print(infer.query(variables=['Feeling anxious'], evidence={'Overeating or loss␣
↪of appetite':1}))
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.2627 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.7373 |
+--------------------+------------------------+
```

[ ]:
```
print(infer.query(variables=['Feeling anxious'], evidence={'Age':0,'Feeling of␣
↪guilt':1}))
```

```
+--------------------+------------------------+
| Feeling anxious    |   phi(Feeling anxious) |
+====================+========================+
| Feeling anxious(0) |                 0.1759 |
+--------------------+------------------------+
| Feeling anxious(1) |                 0.8241 |
+--------------------+------------------------+
```

[ ]:
```
from pgmpy.estimators import MaximumLikelihoodEstimator
for variable in model.nodes:
    cpd = MaximumLikelihoodEstimator(model, data).estimate_cpd(variable)
    model.add_cpds(cpd)
```

```
# Function to calculate Min-fill heuristic for a variable in a graph
def min_fill_heuristic(graph, variable):
    neighbors = set(graph.neighbors(variable))
    fill_edges = set()

    for neighbor1 in neighbors:
        for neighbor2 in neighbors:
            if neighbor1 != neighbor2 and not graph.has_edge(neighbor1,␣
 ↪neighbor2):
                fill_edges.add((neighbor1, neighbor2))

    return len(fill_edges)

# Find the variable elimination order using Min-fill heuristic
elimination_order = []
remaining_variables = set(model.nodes)

# Reverse the order to start eliminating from variables with fewer dependencies
while remaining_variables:
    variable_scores = {variable: min_fill_heuristic(model, variable) for␣
 ↪variable in remaining_variables}
    min_score_variable = min(variable_scores, key=variable_scores.get)

    # Add "DEATH_EVENT" at the end of the elimination order
    if min_score_variable != 'Feeling anxious':
        elimination_order.append(min_score_variable)

    remaining_variables.remove(min_score_variable)

# Print the calculated variable elimination order
print("Variable Elimination Order:", elimination_order)

# Perform Variable Elimination with the calculated order
infer = VariableElimination(model)
```

```
WARNING:pgmpy:Replacing existing CPD for Age
WARNING:pgmpy:Replacing existing CPD for Problems concentrating or making
decision
WARNING:pgmpy:Replacing existing CPD for Feeling sad or Tearful
WARNING:pgmpy:Replacing existing CPD for Suicide attempt
WARNING:pgmpy:Replacing existing CPD for Feeling of guilt
WARNING:pgmpy:Replacing existing CPD for Problems of bonding with baby
WARNING:pgmpy:Replacing existing CPD for Irritable towards baby & partner
WARNING:pgmpy:Replacing existing CPD for Trouble sleeping at night
WARNING:pgmpy:Replacing existing CPD for Overeating or loss of appetite
WARNING:pgmpy:Replacing existing CPD for Feeling anxious
```

```
Variable Elimination Order: ['Problems concentrating or making decision',
'Feeling sad or Tearful', 'Irritable towards baby & partner', 'Suicide attempt',
'Trouble sleeping at night', 'Problems of bonding with baby', 'Feeling of
guilt', 'Age', 'Overeating or loss of appetite']
```

Updates a Bayesian Network by estimating Conditional Probability Distributions (CPDs) using Maximum Likelihood Estimation (MLE) for each variable. It then determines an efficient variable elimination order based on the Min-fill heuristic, a method to minimize computational complexity.

The calculated elimination order is printed, and Variable Elimination is performed on the Bayesian Network using this order, streamlining the process of obtaining probabilities for different variables.

The code is essentially refining and optimizing the Bayesian Network to make probabilistic inference more efficient.

[ ]:
```
test = np.squeeze(data.iloc[:, -1].values)
predict_data = data.iloc[:, :-1]
y_pred = model.predict(predict_data)
print(accuracy_score(np.squeeze(y_pred.values), test))
```

```
  0%|          | 0/319 [00:00<?, ?it/s]
```

```
0.9959758551307847
```

2.8 Conclusion

In this work, we considered a medical dataset to apply the Bayesian Network to compute the conditional probabilities based on the estimated Directed Acyclic Graph.

Different approaches were added also such as Markov Blanket, which could be useful in other purposes such as variable eliminations.