Bayesian Network Modelling of Risk and Prodromal Markers of Parkinson's Disease

Report submitted to SASTRA Deemed to be University
as per the requirement for the course

CSE300 - MINI PROJECT

Submitted by

TEJASWINI KIRTI NARENDAR
(126156165, B.TECH COMPUTER SCIENCE ENGINEERING (AI&DS))
GURUPRIYA K
(126156044, B.TECH COMPUTER SCIENCE ENGINEERING (AI&DS))
AMRITAVARSHINI R V
(126003020, B.TECH COMPUTER SCIENCE ENGINEERING)

May 2025

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA - 613 401

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING
THANJAVUR - 613 401

Bonafide Certificate

This is to certify that the report titled **Bayesian Network Modelling of Risk and Prodromal Markers of Parkinson's Disease** submitted as requirement for the course, CSE300: MINI PROJECT for B.Tech. is a bonafide record of the work done by **Ms. AMRITAVARSHINI R V (126003020, B.TECH COMPUTER SCIENCE ENGINEERING), Ms. GURUPRIYA K (126156044, B.TECH COMPUTER SCIENCE ENGINEERING (AI&DS)), Ms. TEJASWINI KIRTI NARENDAR (126156165, B.TECH COMPUTER SCIENCE ENGINEERING (AI&DS))** during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : **Dr. Ramkumar K,** Associate Dean - Academics, SoC

Date : **04-05-2025**

Mini Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honourable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honourable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Students Welfare.

Our guide **Dr. K. Ramkumar**, Associate Dean, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through project.

List of Figures

Abbreviations

PD	Parkinson's Disease
BBN	Bayesian Belief Networks
PPMI	Parkinson's Progression Markers Initiative
GBA	Glucocerebrosidase
SN	Substantia Nigra
MNAR	Missing Not At Random
RF	Random Forest
BIC	Bayesian Information Criterion
NLL	Negative Log Likelyhood
KLD	Kullback-Leibler Divergence
CPT	Conditional Probabilities Table
AUC	Area Under the Curve
pAUC	Partial Area Under the Curve
UPDRS	Unified Parkinson's Disease Rating Scale
TREND	Tuebinger evaluation of Risk factors for Early detection of NeuroDegeneration
pRBD	Possible REM Sleep Behaviour Disorder
MRI	Magnetic Resonance Imaging
CNN	Convolutional Neural Network

Abstract

Individuals diagnosed with Parkinson's Disease (PD) suffer from a progressive neurodegenerative condition that affects both motor and non-motor functions, leading to significant impairment in their quality of life. Despite the importance of accurate early diagnosis in managing disease, traditional approaches frequently involve subjective clinical assessments and thus result in variable outcomes. The project introduces a Bayesian Belief Networks (BBNs) probabilistic diagnostic framework intended to improve the interpretability and predictive accuracy of clinical decisions.

The data was obtained through the PPMI , which encompasses both longitudinal and static features, and was collected over several visits. Modeling the clinical markers independently is often inadequate due to their interdependence with other markers. To overcome this limitation, the project utilizes a Bayesian Network approach, which uses marker relationships and interdependencies to predict PD.

Learning the Bayesian Network structure through Hill Climb Search and then using non-parametric bootstrapping to refine it allows for the capture of stable, meaningful dependencies among clinical variables.

To determine the generative strength of the learned model, synthetic data generation was carried out, followed by using classification metrics and statistical measures to compare the synthetic data to real data. It offers an interpretable, privacy-preserving and adaptive approach to diagnosing PD, while maintaining consistency and justification in its predictions.

The use of domain knowledge and probabilistic reasoning in this work provides a data-driven approach that can be scaled to support research on PD and early diagnosis. This project report presents these findings.

KEY WORDS: Parkinson’s Disease, Bayesian Belief Networks, Probabilistic Diagnostic framework, Interdependencies, Early diagnosis

Table of Contents

CHAPTER 1
SUMMARY OF THE BASE PAPER

Title	**:**	Bayesian Network Modelling of Risk and Prodromal markers of Parkinson's Disease
Authors	**:**	Sood M, Suenkel U, von Thaler AK, Zacharias HU, Brockmann K, Eschweiler GW, et al.
Journal name	**:**	PLOS ONE
Year	**:**	2023
Indexed in	**:**	SCIE
DOI	**:**	https://doi.org/10.1371/ journal.pone.0280609/

CONTENT

The study tries to capture the interdependencies in the relationships of risk and prodromal features of Parkinson's disease, which enables it to model the data more accurately - picking up on complex and hidden links between the features. This allows for early detection of PD. The dataset was taken from PPMI, with 20 features, including both risk and prodromal such as PD family history, Polygenic risk score, GBA mutation, Pesticide exposure, Solvent exposure, SN hyperechogenicity, Diabetes, Non-physical activity, Non-smoking, Hyposima, Constipation, Dysfunction, Updrs-3, Depression, Cognitive deficits etc. The data records 4 subsequent visits, with an interval of 2 years in between, to check up on the patients. The dataset target has been split into two sub-groups such as no PD and having PD. All the risk markers indicate an increased likelihood of getting PD in future, while the prodromal markers are early warning signs or symptoms of PD that appears before actual diagnosis.

Bayesian networks have been used to link all the features together through conditional probabilities. The interdependencies were captured by the Bayesian network and evaluated using random forest models.

Initially, the dataset had 1237 rows and 20 columns. The participants with invalid IDs were excluded from the analysis. The time-dependent features were then transformed into time-independent features. (from column to [column_visit1, column_visit2, column_visit3, column_visit4]). The number of features was maintained due to their importance to the results.

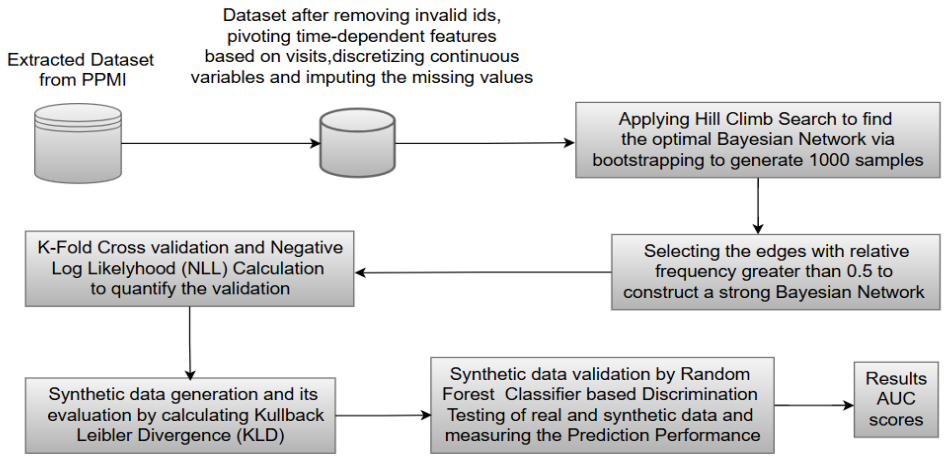
Null values were imputed using MissForest and Most Frequent Label techniques. The features were converted into categorical variables. Auxiliary missing indicators were added for each feature, assuming the missing data to be MNAR (missing not at random) - thus the missing features were used to capture the pattern.

The hill climbing algorithm was applied in order to build the Bayesian network - and it was repeated 1000 times via bootstrapping. This process was done to help the Bayesian network to capture genuine relationships between the features, not the noise generated by particular splits in the data.

Once 1000 Bayesian samples were generated, the strongest edges (edges that appeared in more than 50% of the samples) were selected and used for the construction of the final Bayesian model. Evaluating the performance of a Bayesian Network was done using K-fold cross-validation and calculating the NLL (Negative Log-Likelihood).

The same final Bayesian model was used to generate synthetic data, which was evaluated using the Kullback-Leibler divergence, AUC scores of Random Forest Classifier based Discrimination Testing (real vs synthetic data) and measuring the Prediction performance from data of previous visits.

ARCHITECTURE AND ALGORITHM CORRECTNESS



1. Preprocessing: Discretization and Imputation

The dataset was first partitioned into defined intervals of age and UPDRS scores. By discretizing, the model was able to handle continuous variables more effectively. To address missing data, MissForest was used for time-dependent features which change across visits, while the Most

Frequent Label was utilized for static features. Optimal consistency across dataset was achieved also providing the necessary conditions for Bayesian Network Construction.

2. Structure Learning and Edge Confidence Estimation

Utilizing the Bayesian Network structure and BIC score, the Hill Climb search algorithm was used to construct an optimal network structure. 1000 bootstrap samples were gathered to aid in edge frequency estimation. The final Bayesian Network had edges that were present in over half of the samples (relative frequency >=50%). CPTs were utilized for probability estimation by means of a Dirichlet prior, which simplified the process of handling uncommon feature combinations. The outcome was a model that was both statistically sound and easy to interpret.

3. Model Evaluation via Cross-Validation

To evaluate the model's generalization performance, cross-validation was applied in a 10-fold manner. Evaluation was based on the Negative Log-Likelihood (NLL) score, which was obtained as –9.257865200241754e–05. The score indicated that there was no overfitting and the model and data were in good agreement.

4. Development of synthetic data and statistical analysis

The Synthetic data was generated, with only the features linked through high-confidence edges being preserved (relative frequency >=50%). To evaluate the synthetic data, Kullback-Leibler Divergence (KLD) was used. The KLD value for the target variable was 0.03, and values for other features were near zero. The synthetic data was found to be able to maintain the statistical properties of the original dataset with high precision.

5. Synthetic Data validation through Discrimination Testing

A Random Forest classifier was trained to distinguish between real and synthetic subjects. 10 times 10-fold cross-validation was the testing approach. The partial AUC (pAUC) was 22.80% when real samples had a true positive rate of 99%. The synthetic data was statistically valid and realistic due to the difficulty the model faced in distinguishing between real and synthetic data.

6. Synthetic data Validation by measuring Prediction Performance

Prodromal indicators, including UPDRS category, Constipation, Dysfunction, and Cognitive Deficits, were predicted using Random Forest models. Repeated cross-validation was applied to both real and synthetic datasets. The results were notable: the synthetic data enabled predictive performance that closely matched the real data.

CHAPTER 2
MERITS AND DEMERITS OF THE BASE PAPER

MERITS:

1. **Data Driven Probabilistic Modeling:** Using Bayesian Networks enables modelling of interdependencies between the features, going beyond the assumptions of interdependence which are prevalent in models.
2. **Longitudinal, real-world data:** Dataset contains a longitudinal study of prodromal and risk markers, making the findings robust and clinically relevant.
3. **Generating synthetic subjects:** The generative capabilities of the Bayesian Networks allow it to generate synthetic data- which is suitable in scenarios where data privacy is a constraint.
4. **Insight into markers :** We are able to uncover unexpected and novel relationships between the markers, which has the potential to provide new insights into the disease and how it is identified.
5. **Transparency :** The study provides transparency regarding the methodology employed, including the preprocessing of the dataset, choice of algorithms, and evaluation metrics used, which enhances the reproducibility of the study and allows for validation by other researchers.

DEMERITS :

1. **Dataset is not open-source :** The TREND dataset used in the study is not publicly available, limiting reproducibility and verification of our results.
2. **Ignorance of some factors :** A few markers such as pRBD (Possible REM Sleep Behaviour Disorder) were assessed using self-report questionnaires, reducing the robustness of the data.
3. **No external validation:** The data used is only from a singular source; without testing on an independent dataset, there is a risk of overfitting or the model having limited generalizability.

4. Assumptions of Causality : Although the Bayesian network models the conditional dependencies between the features, they do not establish true causality or the reasoning behind the established relationships. The relationships may be misleading if they are not properly studied and addressed.

CHAPTER 3
SOURCE CODE

```
pip install pgmpy
import pandas as pd
import numpy as np
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, KBinsDiscretizer

import os

file_path = r"C:\Users\ADMIN\Downloads\Dataset.xlsx"

if os.path.exists(file_path):
    data = pd.read_excel(file_path)
    print("File loaded successfully!")
else:
    print("File not found. Check the path.")

import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno

# Display first few rows
print("\n First 5 Rows of the Dataset:")
print(data.head())

# Dataset Info
print("\n Dataset Info:")
print(data.info())

# Check Data Types
print("\n Column Data Types:")
print(data.dtypes)

# Manually Select Numeric Columns (Exclude ID & Categorical Columns)
numeric_cols = [
    'Age', 'updrs-3', 'Target'  # Add more numeric columns if applicable
]
```

```
plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_cols[:9]):  # Limit to 9 subplots for better visualization
    plt.subplot(3, 3, i+1)
    sns.histplot(data[col], bins=20, kde=True, color='blue')
    plt.title(col)
plt.tight_layout()
plt.show()

# Compute correlation matrix
correlation_matrix = data[numeric_cols].corr()

# Plot Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Matrix (Selected Numeric Variables)")
plt.show()

data.columns = data.columns.str.strip()

print(" Updated Column Names:", data.columns)
```

Discretization

```
# Discretize age into categories
data['Age_Category'] = pd.cut(data['Age'], bins=[0, 65, 110], labels=["≤65", ">65"])

data = data.drop(columns=['Age'])

print("\n First 5 Rows After Discretizing Age:")
print(data.head())

print("\n Value Counts for Age Categories:")
print(data['Age_Category'].value_counts())

import pandas as pd


data['UPDRS_Category'] = pd.cut(data['updrs-3'],
                          bins=[0, 3, 6, float('inf')],
                          labels=[0, 1, 2],
                          right=False)
#Drop the updrs-3 category
data=data.drop(["updrs-3"], axis=1)
```

```
print(data['UPDRS_Category'].value_counts())

data.isna().sum()

categorical_cols = ['Sex', 'PD family history', 'Polygenic risk score', 'GBA mutation',
                    'SN hyperechogenicity', 'pesticide exposure', 'solvent exposure',
                    'diabetes', 'non-physical activity', 'non-smoking', 'hyposima',
                    'constipation', 'dysfunction', 'depression', 'cognitive deficits']

for col in categorical_cols:
    print(f"\n Value Counts for {col}:")
    print(data[col].value_counts())

import numpy as np

cols_with_nil = ['Polygenic risk score', 'SN hyperechogenicity', 'diabetes',
                 'non-smoking', 'hyposima', 'constipation', 'dysfunction', 'depression']

data[cols_with_nil] = data[cols_with_nil].replace("Nil", np.nan)

data['dysfunction'] = data['dysfunction'].replace({"no": "No"})
data['cognitive deficits'] = data['cognitive deficits'].replace({"yes": "Yes", "no": "No"})

data['dysfunction'] = data['dysfunction'].replace("", np.nan)
data['Age_Category'] = data['Age_Category'].replace("", np.nan)

print("\n Missing Values After Replacing 'Nil' with NaN:")
print(data.isnull().sum())

categorical_cols = ['Sex', 'PD family history', 'Polygenic risk score', 'GBA mutation',
                    'SN hyperechogenicity', 'pesticide exposure', 'solvent exposure',
                    'diabetes', 'non-physical activity', 'non-smoking', 'hyposima',
                    'constipation', 'dysfunction', 'depression', 'cognitive deficits']

for col in categorical_cols:
    print(f"\n Value Counts for {col}:")
    print(data[col].value_counts())

# Install Missforest
!pip install missingno scikit-learn

print(data.columns.tolist())
```

```
data.head()

df=data.copy()

# Check for duplicate
duplicates = df[df.duplicated(subset=["ID", "Visit"], keep=False)]
if not duplicates.empty:
    print("ERROR: Duplicate (ID, Visit) pairs found! Listing them now:")
    print(duplicates.sort_values(by=["ID", "Visit"]))
    print(" Unique Visit values:", df["Visit"].unique())
    print("Total duplicate rows:", len(duplicates))

duplicate_ids = df[df.duplicated(subset=["ID", "Visit"], keep=False)]["ID"].unique()

print("\n Duplicate IDs That Appear More Than Once for the Same Visit:")
print(duplicate_ids)

df = df[~df["ID"].isin(duplicate_ids)]

duplicates = df[df.duplicated(subset=["ID", "Visit"], keep=False)]
if not duplicates.empty:
    print(" ERROR: Duplicate (ID, Visit) pairs found! Listing them now:")
    print(duplicates.sort_values(by=["ID", "Visit"]))
    print(" Unique Visit values:", df["Visit"].unique())
    print(" Total duplicate rows:", len(duplicates))
else:
    print("No duplicates")

non_varying_columns = []

exclude_columns = ['ID', 'Visit']

for col in df.columns:
    if col in exclude_columns:
        continue

    max_unique_per_id = df.groupby("ID")[col].nunique().max()

    if max_unique_per_id == 1:
        non_varying_columns.append(col)
print(" Columns that are constant across visits (V01–V04) for every patient:")
print(non_varying_columns)
```

Transforming time-dependent features into time-independent features

```
static_features = ["Sex","PD family history"]
target_feature = "Target"

# Expand into V1, V2, V3, V4
time_features = [col for col in df.columns if col not in ["ID", "Visit"] + static_features + [target_feature]]
df_static = df[df["Visit"] == "V01"][["ID"] + static_features]

df_target = df[df["Visit"] == "V04"][["ID", target_feature]]

df_pivot = df.pivot(index="ID", columns="Visit", values=time_features)

df_pivot.columns = [f"{col[0]}_{col[1]}" for col in df_pivot.columns]

df_pivot = df_pivot.reset_index()
df_final = df_static.merge(df_target, on="ID", how="left").merge(df_pivot, on="ID", how="left")

print(" Final Transformed DataFrame (Static Features Kept from Visit 1, Target from Visit 4 ):")
print(df_final.head())

data = df_final.copy()

data.head()

df_cleaned = data.drop(columns=[col for col in data.columns if "_missing" in col])

print("\nSuccessfully Removed '_missing' and '_missing_missing' Columns!")
print(" Remaining Columns:", df_cleaned.columns)

df_cleaned.head()
```

Creating Auxiliary Missing Indicators and Imputing NaN values using MissForest and Most Frequent Label

```
# Create auxiliary missing indicators for MNAR
for col in df_cleaned.columns:
    df_cleaned[col + "_missing"] = df_cleaned[col].isnull().astype(int)

print("\n Created Auxiliary Missing Indicators!")
df_cleaned.head()
```

```
data=df_cleaned.copy()

from sklearn.preprocessing import LabelEncoder

df_encoded = data.copy()

categorical_cols = df_encoded.select_dtypes(include=["object"]).columns.tolist()

df_encoded[categorical_cols] = df_encoded[categorical_cols].astype(str).apply(lambda x: x.str.strip())

le_dict = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])
    le_dict[col] = le

print("\n Successfully Converted ALL Categorical Features to Numeric!")

import pandas as pd

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

print("\n Checking Data Types After Encoding:")
print(df_encoded.dtypes)

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn.ensemble import RandomForestClassifier
#define feature
static_features = ["Sex", "PD family history"]
time_dependent_features = [col for col in df_encoded.columns if "_V" in col]  # Only V1, V2, V3, V4 features

#impute statistic feature
most_frequent_imputer = SimpleImputer(strategy="most_frequent")
df_encoded[static_features] = most_frequent_imputer.fit_transform(df_encoded[static_features])

print("\n Imputed Static Features Using Most Frequent Label!")

# apply missforest
```

```
missforest_imputer = IterativeImputer(
   estimator=RandomForestClassifier(n_estimators=100, random_state=42),
   max_iter=10,
   random_state=42
)
df_encoded[time_dependent_features] =
missforest_imputer.fit_transform(df_encoded[time_dependent_features])

df_encoded[time_dependent_features] =
df_encoded[time_dependent_features].round().astype(int)

print("\n Successfully Imputed Longitudinal Features Using MissForest with Classifier!")


print("\n Final Imputed Dataset Ready for Bayesian Network Learning!")

#trying to clip extra values created from missforest imputation
features_with_2_categories = [
    'Sex', 'PD family history', 'GBA mutation_V01', 'GBA mutation_V02','GBA mutation_V03',
'GBA mutation_V04',
   'SN hyperechogenicity_V01', 'SN hyperechogenicity_V02', 'SN hyperechogenicity_V03', 'SN
hyperechogenicity_V04',
     'pesticide exposure_V01', 'pesticide exposure_V02', 'pesticide exposure_V03', 'pesticide
exposure_V04',
      'solvent exposure_V01', 'solvent exposure_V02', 'solvent exposure_V03', 'solvent
exposure_V04',
   'diabetes_V01', 'diabetes_V02', 'diabetes_V03', 'diabetes_V04',
    'non-physical activity_V01', 'non-physical activity_V02', 'non-physical activity_V03', 'non-
physical activity_V04',
   'non-smoking_V01', 'non-smoking_V02', 'non-smoking_V03', 'non-smoking_V04',
   'depression_V01', 'depression_V02', 'depression_V03', 'depression_V04',
     'cognitive deficits_V01', 'cognitive deficits_V02', 'cognitive deficits_V03', 'cognitive
deficits_V04',
   'Age_Category_V01', 'Age_Category_V02', 'Age_Category_V03', 'Age_Category_V04'
]

#  Features with 3 valid classes
features_with_3_categories = [
   'Polygenic risk score_V01', 'Polygenic risk score_V02', 'Polygenic risk score_V03', 'Polygenic
risk score_V04',
   'hyposima_V01', 'hyposima_V02', 'hyposima_V03', 'hyposima_V04',
   'constipation_V01', 'constipation_V02', 'constipation_V03', 'constipation_V04',
   'dysfunction_V01', 'dysfunction_V02', 'dysfunction_V03', 'dysfunction_V04',
```

```
            'UPDRS_Category_V01',      'UPDRS_Category_V02',      'UPDRS_Category_V03',
'UPDRS_Category_V04'
]

# Combine all
all_categorical_features = features_with_2_categories + features_with_3_categories

#  convert to int
df_encoded[all_categorical_features] = df_encoded[all_categorical_features].round().astype(int)

# post-process values to valid ranges
for col in features_with_2_categories:
    valid_classes = [0, 1]
    df_encoded[col] = df_encoded[col].apply(
        lambda x: x if x in valid_classes else min(valid_classes, key=lambda v: abs(v - x))
    )

for col in features_with_3_categories:
    valid_classes = [0, 1, 2]
    df_encoded[col] = df_encoded[col].apply(
        lambda x: x if x in valid_classes else min(valid_classes, key=lambda v: abs(v - x))
    )
df_final_imputed = df_encoded.copy()

import pandas as pd

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
print(df_final_imputed.isnull().sum())

data=df_final_imputed.copy()

print("\n Total Rows in Dataset:", df_final_imputed.shape[0])

# common "Missing Data Flag"
data['Missing_Data_Flag'] = data.isnull().any(axis=1).astype(int)

print("\n Created Common Missing Feature for BN!")
print(data['Missing_Data_Flag'].value_counts())
```

Applying Hill Climbing algorithm and Bootstrapping

```
# Install bnlearn
!pip install bnlearn

import pandas as pd
import numpy as np
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import HillClimbSearch, BicScore, MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
from sklearn.model_selection import KFold

data = data.copy()

data.drop(columns=[col for col in data.columns if '_missing' in col], inplace=True)

data.drop(columns=['Missing_Data_Flag'], inplace=True)

data.head()

hc = HillClimbSearch(data)
bn_learned = hc.estimate(scoring_method=BicScore(data), max_indegree=3) # Constrain indegree

bootstrap_samples = 1000
edges_count = {}
all_edges = []
edges_above_50 = []  # to store edges with frequency >0.5

# bootstrap sampling
for i in range(bootstrap_samples):
    data_sample = data.sample(frac=1, replace=True)
    bn_sample = hc.estimate(scoring_method=BicScore(data_sample))

    for edge in bn_sample.edges():
        all_edges.append(edge)
        edges_count[edge] = edges_count.get(edge, 0) + 1
```

Bayesian Network Visualization

```
import networkx as nx
import matplotlib.pyplot as plt
```

```
# --- Thresholds ---
solid_threshold = 0.5
dashed_threshold = 0.3

# --- Categorize edges ---
edges_solid = [edge for edge, count in edges_count.items() if count / bootstrap_samples > solid_threshold]
edges_dashed = [edge for edge, count in edges_count.items() if dashed_threshold <= count / bootstrap_samples <= solid_threshold]

# --- Build graph ---
G = nx.DiGraph()
G.add_edges_from(edges_solid + edges_dashed)

# --- Layout ---
pos = nx.kamada_kawai_layout(G)

# --- Plot ---
plt.figure(figsize=(16, 12))
nx.draw_networkx_nodes(G, pos, node_color='lightgreen', node_size=500)
nx.draw_networkx_labels(G, pos, font_size=6)

# Solid edges: > 0.5
nx.draw_networkx_edges(G, pos, edgelist=edges_solid, edge_color='blue', width=2)

# Dashed edges: 0.3 to 0.5
nx.draw_networkx_edges(G, pos, edgelist=edges_dashed, edge_color='red', style='dashed', width=0.8)

plt.title("Bayesian Network: Blue Solid (>0.5), Red Dashed (0.3–0.5)", fontsize=12)
plt.axis('off')
plt.tight_layout()
plt.show()

# Filter edges that appear in at least 50% of bootstrap samples
final_edges = [edge for edge, count in edges_count.items() if count / bootstrap_samples >= 0.5]
bn_final = BayesianNetwork(final_edges)

print("Nodes in Bayesian Network:", bn_final.nodes())

print("Edges in Bayesian Network:", bn_final.edges())

bn_final.fit(data, estimator=MaximumLikelihoodEstimator)
```

```
kf = KFold(n_splits=10, shuffle=True, random_state=42)
log_likelihoods = []
```

Generation of Conditional Probability Tables using Dirichlet Prior

```
from pgmpy.estimators import BayesianEstimator

for train_index, test_index in kf.split(data):
    data_train, data_test = data.iloc[train_index], data.iloc[test_index]

    bn_final.remove_cpds(*bn_final.get_cpds())

    # train BN
            bn_final.fit(data_train, estimator=BayesianEstimator, prior_type="dirichlet",
pseudo_counts=1)

    # print (CPDs)
    for cpd in bn_final.get_cpds():
        print(f"\n CPD for {cpd.variable}:\n{cpd}")

import warnings
import logging
import matplotlib

warnings.filterwarnings("ignore", module='matplotlib.category')
logging.getLogger('matplotlib.category').setLevel(logging.ERROR)

for cpd in bn_final.get_cpds():
    print(f"{cpd.variable} states: {cpd.state_names[cpd.variable]}")

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import warnings

warnings.filterwarnings("ignore", category=UserWarning, module='matplotlib.category')

for cpd in bn_final.get_cpds():
    variable = cpd.variable
    state_names = cpd.state_names
    values = cpd.get_values()
```

```
    print(f"\n CPD of {variable} (shape: {values.shape}):")

    # no parents (marginal)
    if len(cpd.variables) == 1:
        probs = values.flatten()
        x_labels = [str(i) for i in range(len(probs))]
        plt.bar(x_labels, probs, color='skyblue')
        plt.title(f'Marginal Distribution of {variable}')
        plt.xlabel(f'{variable} states')
        plt.ylabel('Probability')
        plt.tight_layout()
        plt.show()

    # has parents (conditional)
    else:
        parents = cpd.variables[1:]
        parent_cardinalities = [cpd.cardinality[cpd.variables.index(p)] for p in parents]
                parent_combinations = np.array(np.meshgrid(*[range(card) for card in
parent_cardinalities])).T.reshape(-1, len(parents))

        for idx, parent_vals in enumerate(parent_combinations):
            prob_vector = values[:, idx]
            label = ', '.join([f'{p}={cpd.state_names[p][v]}' for p, v in zip(parents, parent_vals)])
            x_labels = [str(i) for i in range(len(prob_vector))]
            plt.bar(x_labels, prob_vector, label=label)

        plt.title(f'CPT of {variable}')
        plt.xlabel(f'{variable} states')
        plt.ylabel('Probability')
        plt.legend(title='Given', fontsize='small', bbox_to_anchor=(1.05, 1), loc='upper left')
        plt.tight_layout()
        plt.show()

for var in bn_final.nodes():
    print(f"\n Valid States for {var} in BN:")
    print(bn_final.get_cpds(var))
```

Bayesian Network Validation by NLL Calculation

```
from pgmpy.inference import VariableElimination

infer = VariableElimination(bn_final)
```

```
for _, row in data_test.iterrows():
    evidence = row.drop(["Target", "ID"]).to_dict()
    valid_evidence = {k: v for k, v in evidence.items() if k in bn_final.nodes()}

    # perform inference
    query_result = infer.query(variables=["Target"], evidence=valid_evidence)

    target_value = row["Target"]
    if target_value in query_result.state_names["Target"]:
        target_index = query_result.state_names["Target"].index(target_value)
        prob = query_result.values[target_index]
    else:
        print(f" Target value {target_value} not found in state names!")
        prob = 0  # to avoid indexing error

    #  log probability
    log_likelihood = np.log(prob) if prob > 0 else float('-inf')
    log_likelihoods.append(log_likelihood)

print("\n Cross-Validation Negative Log-Likelihood Scores:")
print(log_likelihoods)
print("\n Average Negative Log-Likelihood:", np.mean(log_likelihoods))
```

Generation and Evaluation of Synthetic Data using KLD

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import RepeatedKFold
from pgmpy.sampling import BayesianModelSampling

# generate synthetic data
sampler = BayesianModelSampling(bn_final)
synthetic_data = sampler.forward_sample(size=len(data), show_progress=False)

print(synthetic_data.head())

synthetic_cols = list(synthetic_data.columns)
real_data_subset = data[synthetic_cols].copy()

print(synthetic_cols)

from scipy.stats import entropy
```

```
def kl_divergence(p, q):
    p = p / p.sum()
    q = q / q.sum()
    return entropy(p, q)

#comparison for 'Target'
real_dist = real_data_subset['Target'].value_counts(normalize=True).sort_index()
synth_dist = synthetic_data['Target'].value_counts(normalize=True).sort_index()

synth_dist = synth_dist.reindex(real_dist.index, fill_value=1e-6)

kl = kl_divergence(real_dist, synth_dist)
print(f"KL Divergence for 'Target': {kl:.4f}")

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import entropy

def kl_divergence(p, q):
    p = p / p.sum()
    q = q / q.sum()
    return entropy(p, q)

for column in synthetic_data.columns:
    if column in real_data_subset.columns:
        real_dist = real_data_subset[column].value_counts(normalize=True).sort_index()
        synth_dist = synthetic_data[column].value_counts(normalize=True).sort_index()
        all_categories = sorted(set(real_dist.index).union(set(synth_dist.index)))
        real_dist = real_dist.reindex(all_categories, fill_value=0)
        synth_dist = synth_dist.reindex(all_categories, fill_value=0)

        # KL Divergence
        kl = kl_divergence(real_dist, synth_dist)

        # Plot
        x = np.arange(len(all_categories))
        width = 0.35

        plt.figure(figsize=(8, 4))
        plt.bar(x - width/2, real_dist, width, label='Real', color='green')
        plt.bar(x + width/2, synth_dist, width, label='Synthetic', color='red')
        plt.xticks(x, all_categories, rotation=45, ha='right')
```

```
    plt.title(f"{column} (KL Divergence = {kl:.4f})")
    plt.ylabel('Probability')
    plt.legend()
    plt.tight_layout()
    plt.grid(axis='y', linestyle='--', alpha=0.5)
    plt.show()

!pip install scikit-learn numpy pandas matplotlib –quiet
```

Synthetic Data Validation using Random Forest Classifier and computing AUC scores

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve, auc

real_data_subset['label'] = 1
synthetic_data['label'] = 0

# Combine datasets
combined_data = pd.concat([real_data_subset, synthetic_data], ignore_index=True)
X = combined_data.drop(columns=['label'])
y = combined_data['label']

import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve, auc

#parameters
detection_rate_threshold = 0.99
n_splits = 10
n_repeats = 10
random_state = 42

rf = RandomForestClassifier(n_estimators=100, random_state=random_state)
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=random_state)

pauc_scores = []
```

```
all_true = []
all_proba = []

for repeat in range(n_repeats):
    fold_pauc_scores = []

    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        rf.fit(X_train, y_train)
        y_prob = rf.predict_proba(X_test)[:, 1]

        all_true.extend(y_test)
        all_proba.extend(y_prob)

        fpr, tpr, _ = roc_curve(y_test, y_prob)

        # pAUC region where TPR ≥ 99%
        mask = tpr >= detection_rate_threshold
        fpr_high = fpr[mask]
        tpr_high = tpr[mask]

        if len(fpr_high) >= 2:
            pAUC = auc(fpr_high, tpr_high)
        else:
            pAUC = 0

        fold_pauc_scores.append(pAUC)

    pauc_scores.append(fold_pauc_scores)

# convert to percentage
mean_pAUC = np.mean([np.mean(r) for r in pauc_scores]) * 100
print(f"Mean pAUC (TPR ≥ 99%): {mean_pAUC:.2f}%")

# ROC Curve
fpr, tpr, _ = roc_curve(all_true, all_proba)
roc_auc = auc(fpr, tpr)

# pAUC region
mask = tpr >= detection_rate_threshold
fpr_pauc = fpr[mask]
```

```
tpr_pauc = tpr[mask]
pauc_combined = auc(fpr_pauc, tpr_pauc) if len(fpr_pauc) >= 2 else 0

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.fill_between(fpr_pauc, tpr_pauc, alpha=0.3, color='orange',
                 label=f'pAUC (TPR ≥ 0.99) = {pauc_combined:.2f}')
plt.plot([0, 1], [0, 1], 'k--', label='Chance')

plt.title('ROC Curve with pAUC Region (TPR ≥ 99%)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Detection Rate)')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

#Horizontal Boxplot
all_pauc_scores = [score for repeat in pauc_scores for score in repeat]

plt.figure(figsize=(8, 4))
plt.boxplot(all_pauc_scores, vert=False, patch_artist=True,
            boxprops=dict(facecolor='lightblue', color='black'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            medianprops=dict(color='red'))

plt.title('Random Forest Classifier: pAUC Distribution\n(TPR ≥ 99%, Combined 10x10 CV)')
plt.xlabel('pAUC')
plt.yticks([1], [''])
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()

print("\nWe used the partial area under ROC curve (pAUC) at a pre-specified true positive rate of 99% for real subjects as a measure of the prediction performance.")
print("The area under the ROC curve at which the detection rate for real subjects was between 99% and 100% served as an indicator of the validity of the synthetic TREND participants.")

from sklearn.metrics import roc_auc_score

results = []
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import roc_auc_score
import numpy as np
import pandas as pd

real_df = real_data_subset.copy()
synth_df = synthetic_data.copy()

markers = [
    'UPDRS_Category_V01',
    'UPDRS_Category_V02',
    'UPDRS_Category_V03',
    'constipation_V02',
    'dysfunction_V01',
    'cognitive deficits_V03',
    'cognitive deficits_V04'
]

def get_visit(col_name):
    if '_V0' in col_name:
        return int(col_name.split('_V0')[1])
    return None

for target in markers:
    visit = get_visit(target)
    if visit is None or visit == 1:
        continue

    prev_visit = visit - 1
    prev_visit_str = f'_V0{prev_visit}'

    input_features = [col for col in real_df.columns if prev_visit_str in col and col != target]
    if not input_features:
        print(f"Skipping {target}: no previous visit features found.")
        continue

    print(f"\n Predicting {target} using features from visit {prev_visit}:")

    X_real = real_df[input_features].copy()
    y_real = real_df[target].copy()

    X_synth = synth_df[input_features].copy()
```

```
    y_synth = real_df[target].copy()

    combined = pd.concat([X_real, y_real], axis=1).dropna()
    X_real = combined[input_features]
    y_real = combined[target]

    combined_synth = pd.concat([X_synth, y_synth], axis=1).dropna()
    X_synth = combined_synth[input_features]
    y_synth = combined_synth[target]

    class_counts = y_real.value_counts()
    n_classes = len(class_counts)
    min_class_size = class_counts.min()

    if n_classes < 2 or min_class_size < 10:
        print(f" Skipping {target}: Not enough samples in one of the classes (min count = {min_class_size})")
        continue

    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state=42)

    auc_scores = []

    for train_idx, test_idx in cv.split(X_real, y_real):
        try:
            clf.fit(X_real.iloc[train_idx], y_real.iloc[train_idx])
            probas = clf.predict_proba(X_real.iloc[test_idx])
            if n_classes == 2:
                auc = roc_auc_score(y_real.iloc[test_idx], probas[:, 1])
            else:
                auc = roc_auc_score(y_real.iloc[test_idx], probas, multi_class='ovr', average='macro')
            auc_scores.append(auc)
        except:
            continue

    if auc_scores:
        print(" Real→Real AUC:", round(np.mean(auc_scores), 3))
    else:
        print(" Real→Real AUC: Not computable (possibly not enough data)")

    # train on synthetic, test on real
    try:
```

```
        clf.fit(X_synth, y_synth)
        probas = clf.predict_proba(X_real)
        if n_classes == 2:
            auc_synth_to_real = roc_auc_score(y_real, probas[:, 1])
        else:
            auc_synth_to_real = roc_auc_score(y_real, probas, multi_class='ovr', average='macro')
        print("Synthetic→Real AUC:", round(auc_synth_to_real, 3))
    except:
        print("Synthetic→Real AUC: Could not compute")

pip install prince

import pandas as pd
import prince
import matplotlib.pyplot as plt

binary_cols = [
    'Age_Category_V01', 'Age_Category_V02', 'Age_Category_V03', 'Age_Category_V04',
    'cognitive deficits_V03', 'cognitive deficits_V04',
    'non-physical activity_V01', 'non-physical activity_V02',
    'non-physical activity_V03', 'non-physical activity_V04',
    'solvent exposure_V04', 'diabetes_V02'
]

ternary_cols = [
    'UPDRS_Category_V01', 'UPDRS_Category_V02', 'UPDRS_Category_V03',
    'dysfunction_V01', 'constipation_V02'
]

target_col = ['Target']

# mappings
binary_map = {0: 'class_0', 1: 'class_1'}
ternary_map = {0: 'class_0', 1: 'class_1', 2: 'class_2'}
target_map = {0: 'class_0', 2: 'class_2'}

real_data_copy = real_data_subset.copy()
synthetic_data_copy = synthetic_data.copy()

for df in [real_data_copy, synthetic_data_copy]:
    for col in binary_cols:
        df[col] = df[col].map(binary_map)
    for col in ternary_cols:
```

```
        df[col] = df[col].map(ternary_map)
    for col in target_col:
        df[col] = df[col].map(target_map)

real_data_copy['source'] = 'real'
synthetic_data_copy['source'] = 'synthetic'

combined_df = pd.concat([real_data_copy, synthetic_data_copy], ignore_index=True)

# apply MCA
mca = prince.MCA(n_components=2, random_state=42)
mca_fit = mca.fit(combined_df.drop(columns=['source']))
mca_coords = mca_fit.transform(combined_df.drop(columns=['source']))

# Plot
plt.figure(figsize=(8, 6))
for label, color in zip(['real', 'synthetic'], ['steelblue', 'darkorange']):
    subset = mca_coords[combined_df['source'] == label]
    plt.scatter(subset[0], subset[1], label=label, alpha=0.6, c=color)

plt.title('MCA Projection of Real vs Synthetic Subjects')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Spearman correlations
real_corr = real_data_subset.corr(method='spearman')
synth_corr = synthetic_data.corr(method='spearman')
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
sns.heatmap(real_corr, cmap='RdBu_r', center=0, annot=False, fmt=".2f", square=True,
cbar=True)
plt.title("A. Spearman Correlation (Real Subjects)", fontsize=14)
plt.subplot(1, 2, 2)
sns.heatmap(synth_corr, cmap='RdBu_r', center=0, annot=False, fmt=".2f", square=True,
cbar=True)
```

```
plt.title("B. Spearman Correlation (Synthetic Subjects)", fontsize=14)

plt.tight_layout()
plt.show()
```

CHAPTER 4
SNAPSHOTS

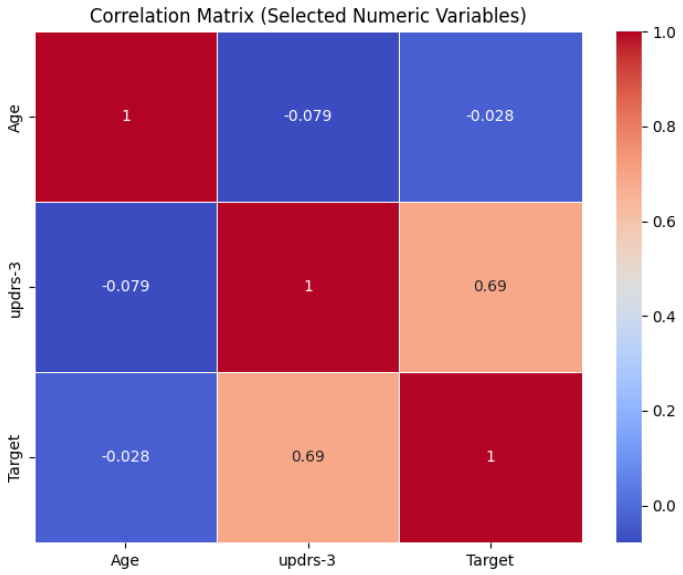
Fig.1 Exploratory Data Analysis of features

1a Dataset Head

First 5 Rows of the Dataset:

	ID	Visit	Age	Sex	PD family history	Polygenic risk score	GBA mutation	SN hyperechogenicity	pesticide exposure	solvent exposure	diabetes	non-physical activity	non-smoking	hyposima	constipation	dysfunction	updrs-3	depression	cognitive deficits	Target
0	3518	V01	69.1	Male	Yes	Mid	No	Positive	Yes	No	No	Yes	No	Mid	Mid	No	2	No	No	0
1	3656	V01	67.0	Male	Yes	No	Yes	Negative	No	Yes	Yes	No	Yes	Mid	Nil	No	0	Yes	No	0
2	3271	V01	50.7	Male	No	Mid	No	Nil	Yes	No	Nil	No	Nil	Nil	Nil	No	0	No	Yes	0
3	3270	V01	79.4	Male	No	Yes	Yes	Positive	No	Yes	No	Yes	No	Nil	Mid	Nil	2	Nil	No	0
4	3269	V01	84.7	Male	Yes	Nil	No	Negative	Yes	No	Mid	No	Yes	Mid	Mid	Yes	1	Yes	Yes	0

1b Correlation matrix of Numeric attributes



1c Distributions of data in Numeric Attributes

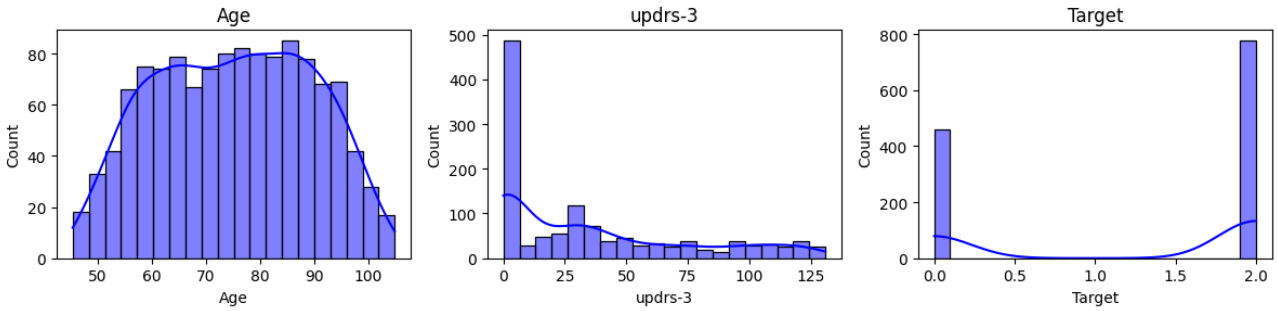


Fig.2 Bayesian Network constructed by Hill Climbing and Bootstrapping (blue solid lines represent edges captured by >=50% of the Bootstrap samples ; red dashed lines represent edges captured by >=30% of the Bootstrap samples.)

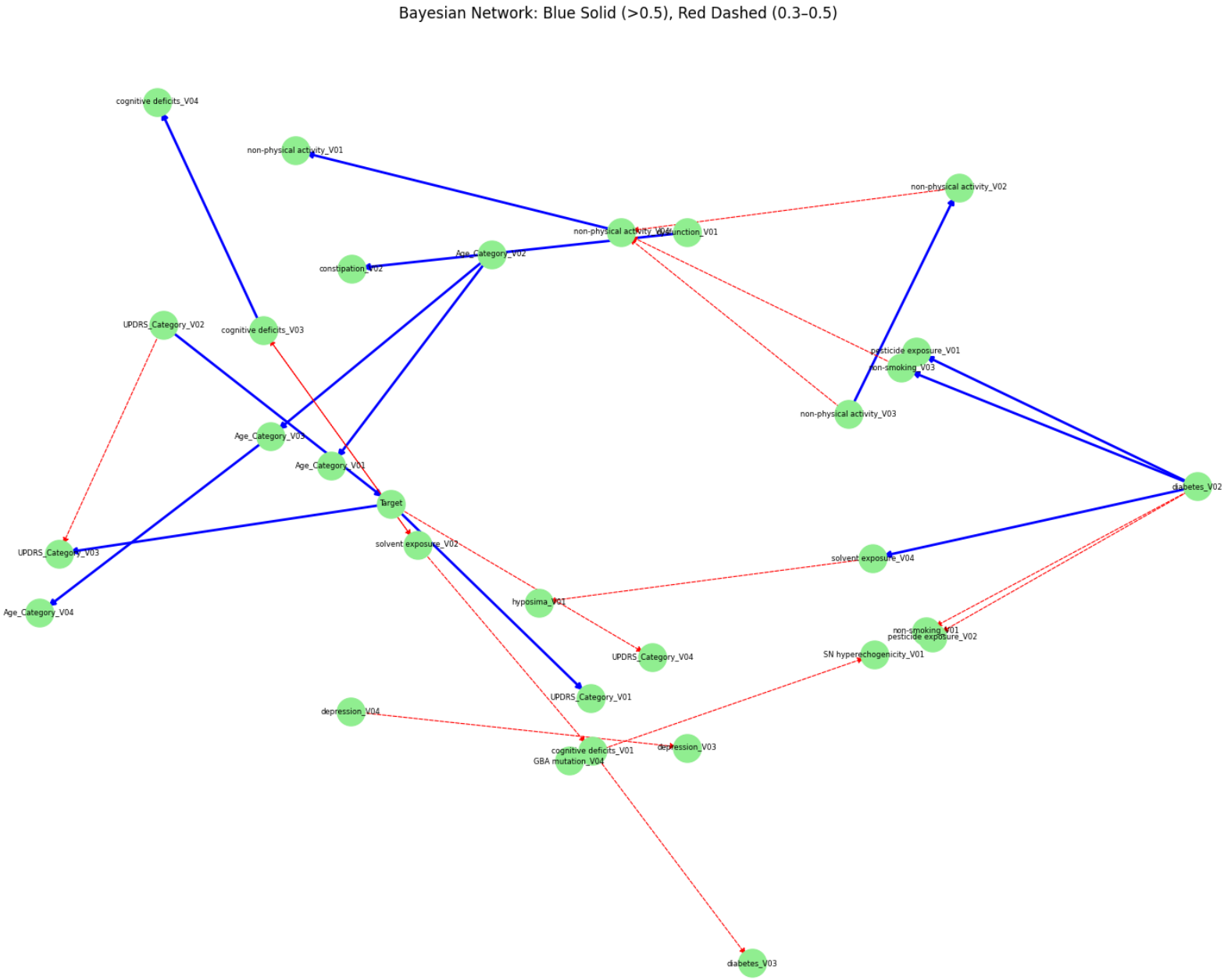


Fig.3 Conditional probability table of the features

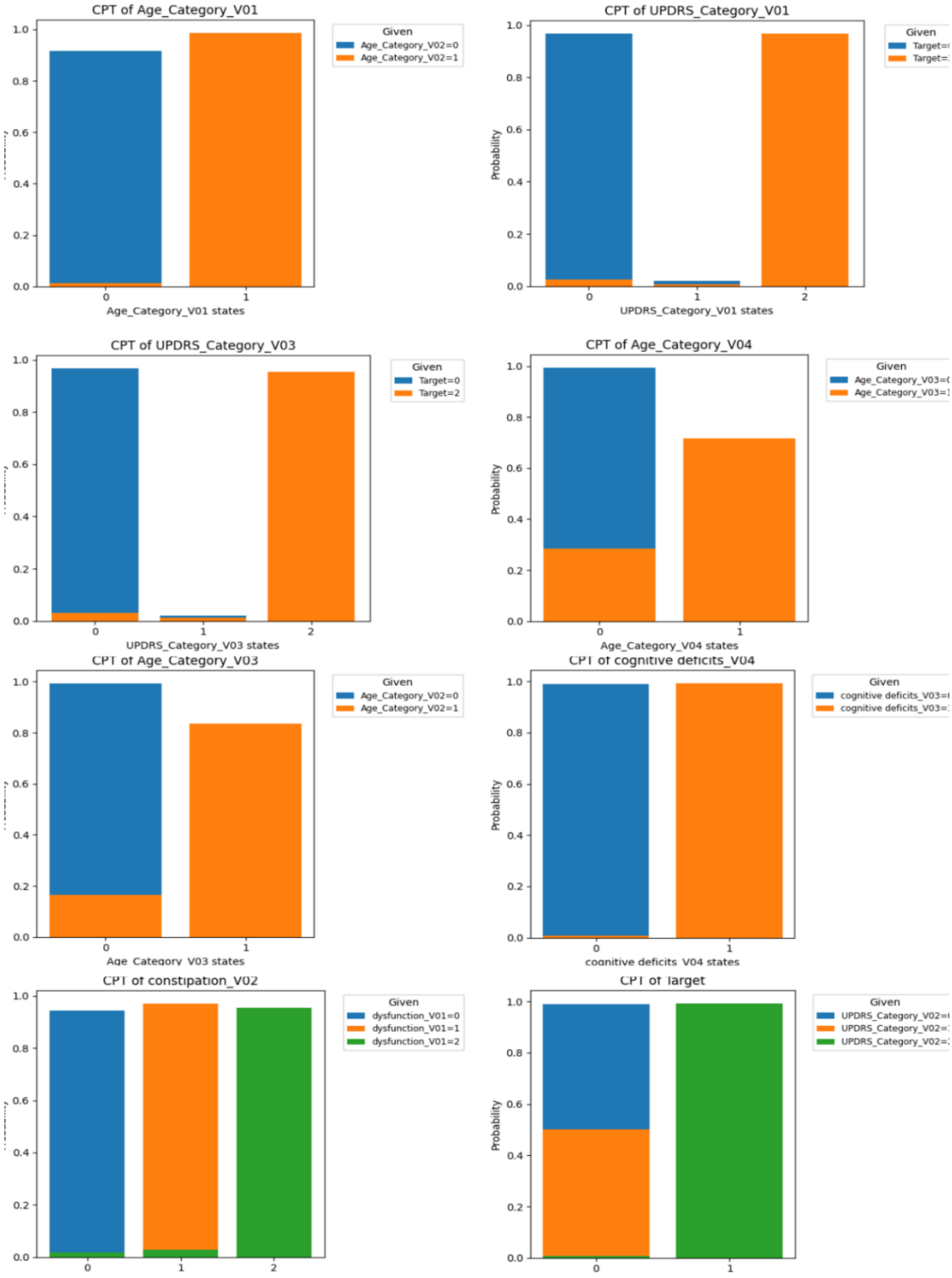


Fig.4 Kullback-Leibler Divergence

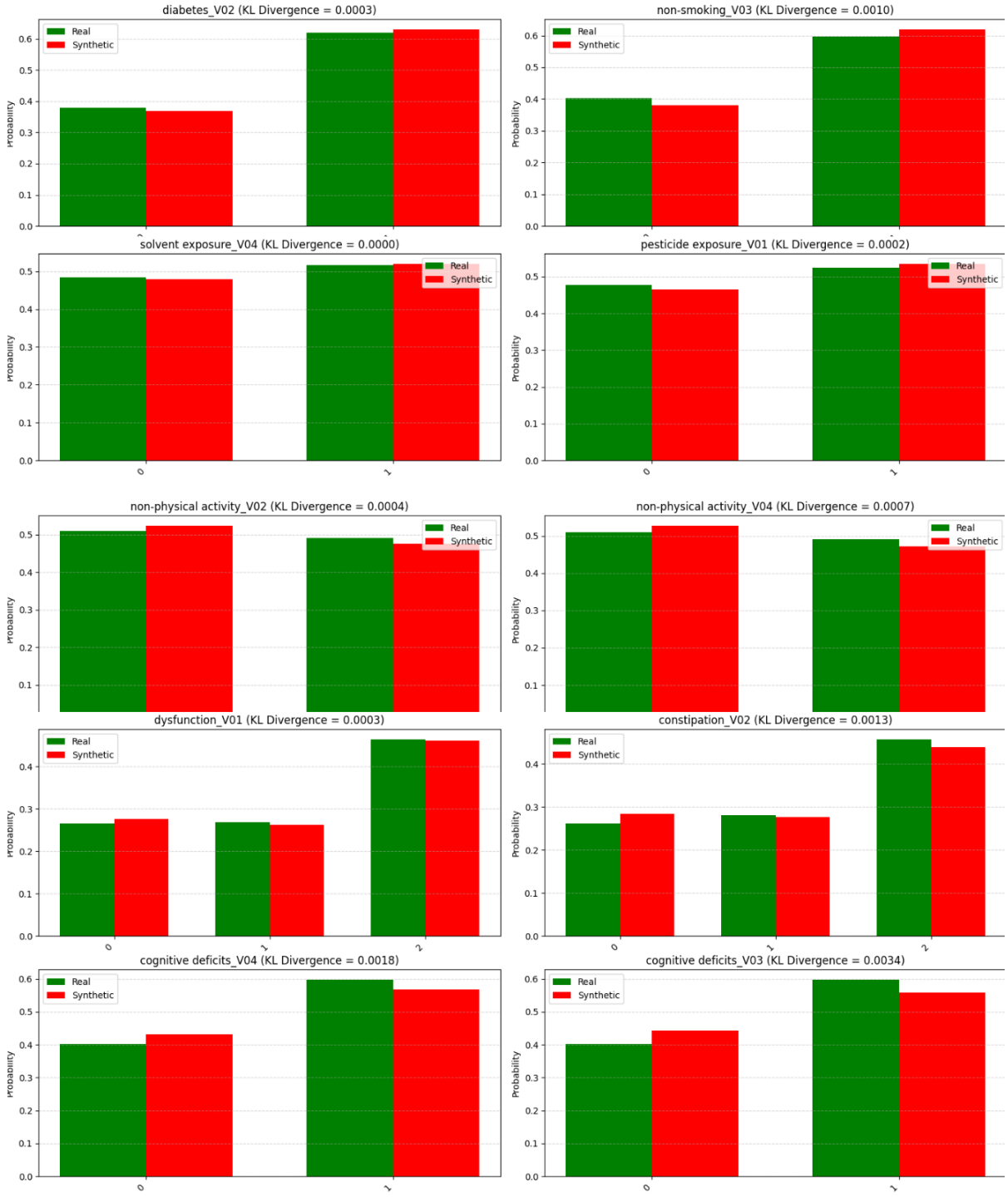


Fig. 5 ROC curve of pAUC Region of high confidence, in RandomForest classifier. (classifies synthetic and real data)

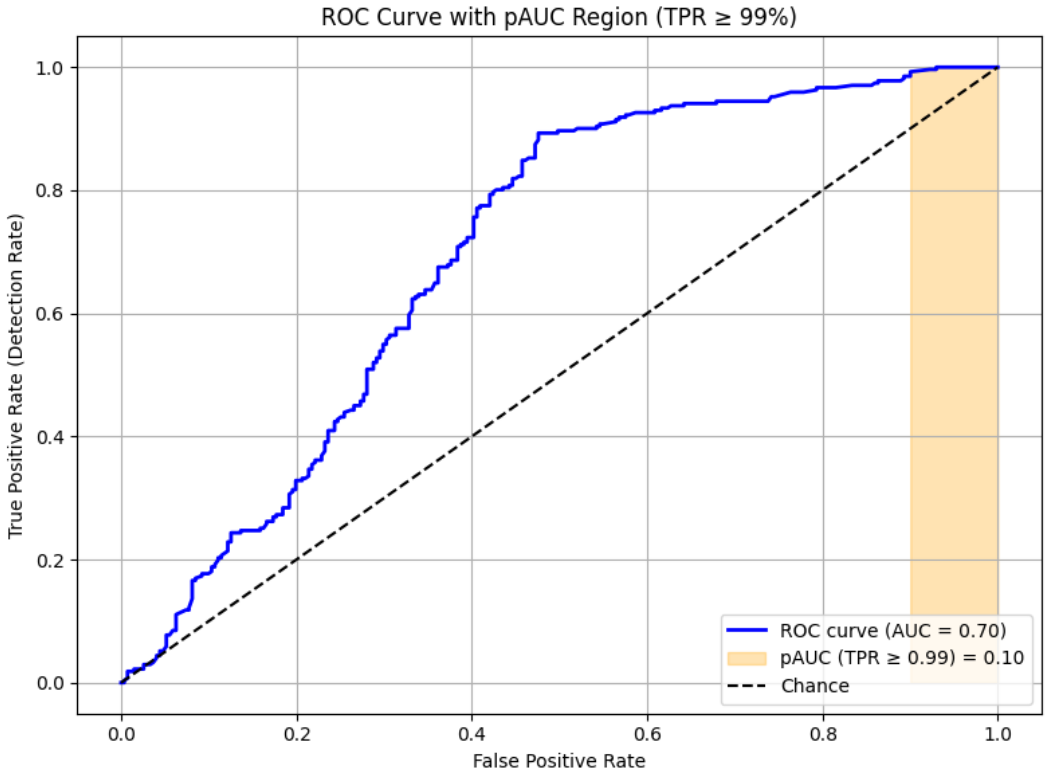


Fig. 6 Distribution of pAUC scores as boxplot.

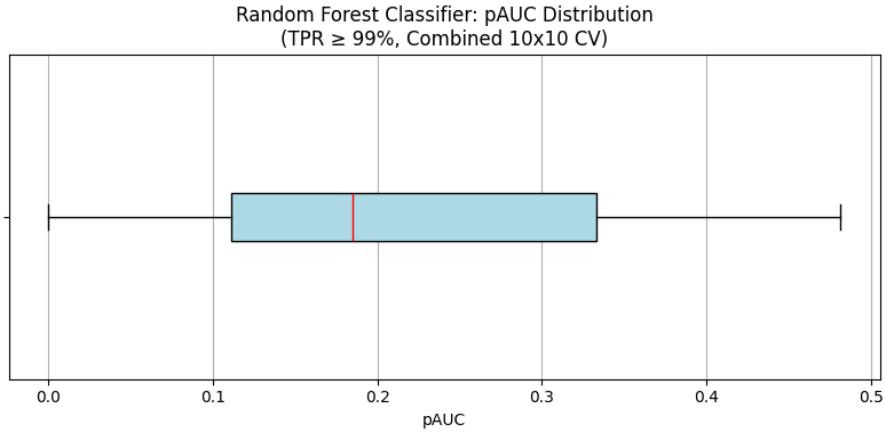


Fig. 7 Plotting synthetic and real data using Multiple Correspondence Plot

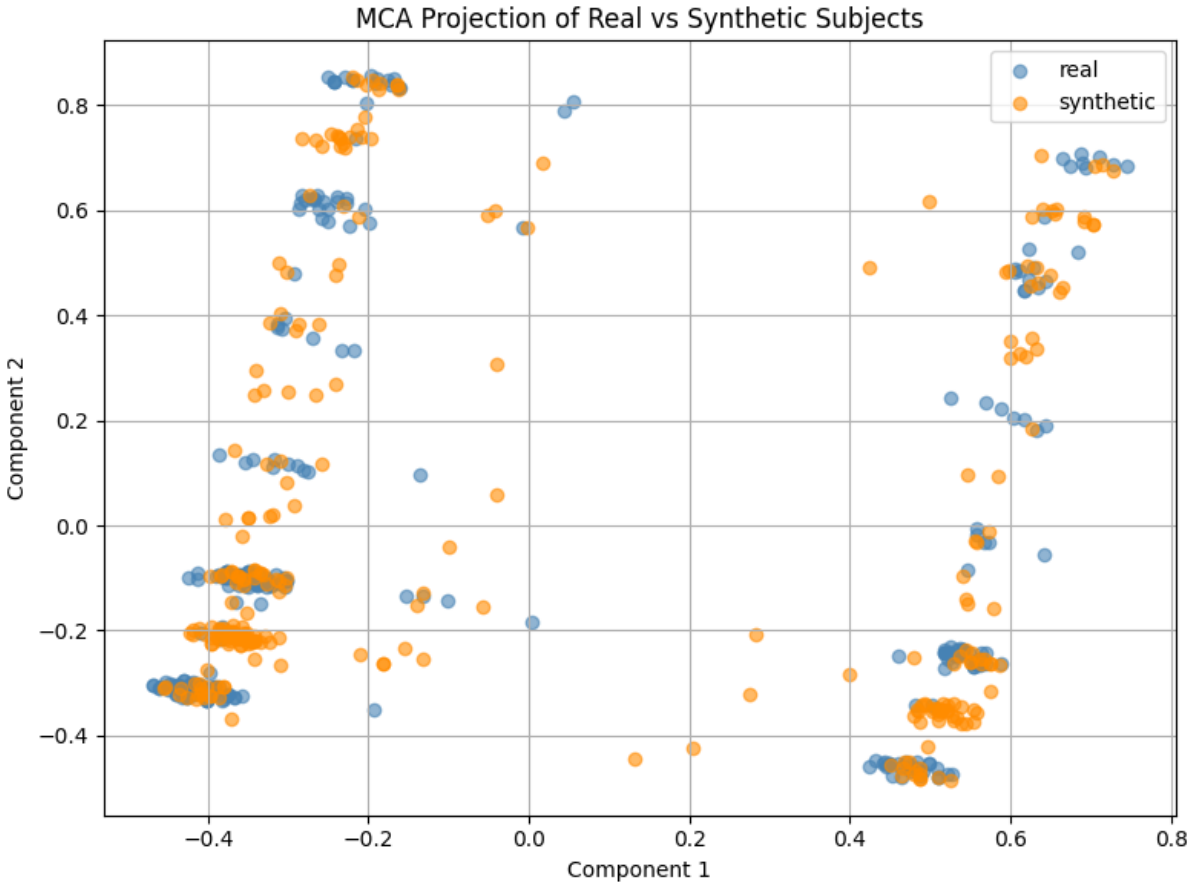
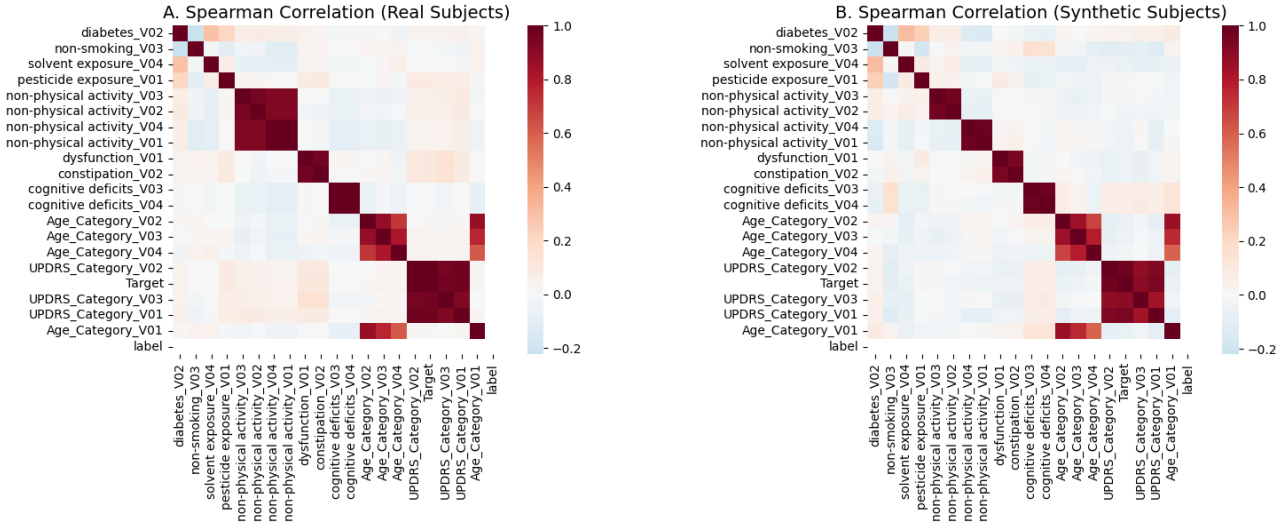


Fig.8 Spearman correlation of real and synthetic data



CHAPTER 5
CONCLUSION AND FUTURE PLANS

This study focused on identifying the hidden and complex interdependencies between the risk and prodromal markers of the participants using a Bayesian network and then generating synthetic data using the network. The markers were all structured and tabular and did not include multi-modal inputs such as imaging data (such as MRI scans). Thus this study has limitations on the types of data used and incorporated into the study. Integrating more systems and diverse data modalities could enhance the robustness and improve early prediction of PD. Including rule-based systems, guided by clinical protocols, in future work can enhance the diagnostic adaptability and interpretability. In the same way, utilizing convolutional neural networks (CNNs) for processing MRI scans alongside the Bayesian model could result in a comprehensive, hybrid diagnostic framework.

CHAPTER 6
REFERENCES

1. Youjia Qiu, Jane Smith, Robert Taylor (2023) "Motor Function Improvement and Acceptability of Non-Invasive Brain Stimulation in Patients with Parkinson's Disease: A Bayesian Network Analysis" Frontiers in Neuroscience, Volume 17, Article 1212640. Sep 2023.
2. Simuni T., Caspell-Garcia C., Coffey C.S., Weintraub D., Mollenhauer B., Lasch S., Tanner C.M., Jennings D., Kieburtz K., Chahine L.M., Marek K. (2018) "Baseline prevalence and longitudinal evolution of non-motor symptoms in early Parkinson's disease: the PPMI cohort." Journal of Neurology, Neurosurgery & Psychiatry, Volume 89, Pages 78–88, BMJ Publishing Group.
3. 3. Ren J., Xie H., Weng Y., Ge Y., Yao R., Jiang Z., Zhang J., Zhu Y., Fu X., Wang J., Liu Z., Zhang S., Zhang T., Chen G., Yang D. (2024) "Longitudinal decline in DAT binding in Parkinson's disease: connections with sleep disturbances." BMC Medicine, Volume 22, Article 550, Springer Nature.
4. 4. Fengler S., Kessler J., Timmermann L., Zapf A., Elben S., Wojtecki L., Tucha O., Kalbe E. (2016) "Screening for Cognitive Impairment in Parkinson's Disease: Improving the Diagnostic Utility of the MoCA through Subtest Weighting." PLOS ONE, Volume 11, Issue 7, Article e015931.

CHAPTER 7
APPENDIX - BASE PAPER

TITLE:

Bayesian network modeling of risk and prodromal markers of Parkinson's disease

CITATION:

Sood M, Suenkel U, von Thaler AK, Zacharias HU, Brockmann K, Eschweiler GW, et al. Bayesian network modeling of risk and prodromal markers of Parkinson's disease.

PLOS ONE. 2023;18(2):e0280609.

doi: 10.1371/journal.pone.0280