```
# %% [markdown]
# Originated by: Sandy G. Cabanes
# September 9, 2025
# This script generates the Bayesian Network model using HillClimbSearch, BDeu
scoring and fits the model with Bayesian Estimator and Maximum Likelihood
Estimator

# %%
# Import packages
import time
import numpy as np
import pandas as pd
import pgmpy
import pickle
import os

from pgmpy.estimators import HillClimbSearch, BDeu, BayesianEstimator,
MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork, DiscreteBayesianNetwork #For Naive
Bayes seed
from pgmpy.sampling import BayesianModelSampling

# %%
# Create log function
def log(msg):
    ts = time.strftime("%H:%M:%S")
    print(f"[{ts}] {msg}")

# %%
# Pre-processing data (skip if df_mod final)
# ---------------------------
log("Loading and preprocessing dataset...")

# Load the dataset
df_raw = pd.read_csv('dataset_full.csv', encoding='latin1')

# Dropping unnecessary columns like previous R code
df_full = df_raw.drop(columns=['Timestamp', 'resp_id', 'bestproject', 'agegrp'])

# Discretizing the 'age' variable into 'age_grp'
bins = [-np.inf, 19, 25, 30, 35, 40, 45, 50, 55, np.inf]
labels = ["<19", "20 to 24", "25 to 29", "30 to 34", "35 to 39", "40 to 44", "45
to 49", "50 to 54", "55+"]
df_full['age_grp'] = pd.cut(df_full['age'], bins=bins, labels=labels,
right=False, include_lowest=True)
```

```
df_full = df_full.drop(columns=['age'])
df_full.to_csv('df_full.csv', index=False)

# Convert string to pre-process blanks
df_str = df_full.astype('string')

# Preprocessing: Fill in missing values
df_str = df_str.replace('', np.nan)
df_str = df_str.fillna('CODEASBLANK')

# Convert all columns to the 'category' data type
df_mod = df_str.astype('category')
df_mod.to_csv('df_mod.csv', index=False)

# Printing summary and structure to verify preprocessing
print("\nDataFrame summary:")
print(df_mod.describe(include='category'))
print("\nDataFrame structure:")
df_mod.info()

df = df_mod.copy()
log(f"Preprocessing done. Shape: {df.shape[0]} rows x {df.shape[1]} columns")


# %%
# Use saved df_mod, convert to category, for start_dag
# Also copied to main loop
df_mod = pd.read_csv("df_mod.csv")
df_mod = df_mod.astype('category')
df = df_mod.copy()
# df.head(1)

# %%
# Global: Build Naive Bayes edges as seed DAG
# nb_model is global input to estimate_model function
target = "educstat"
features = [col for col in df.columns if col != target]
edges = [(target, feat) for feat in features]
nb_model = DiscreteBayesianNetwork(edges)


# %%
# Check nodes if matching

nodes_startdag = sorted(nb_model.nodes())
```

```
print("start_dag nodes:", nodes_startdag)
nodes_cols = sorted(df.columns.tolist())
print("df columns:", sorted(df.columns.tolist()))
extra_nodes = [ x for x in nodes_startdag if x not in nodes_cols]
print(extra_nodes)

# %%
# Global: Expert knowledge required edges

from pgmpy.estimators import ExpertKnowledge

# Defining the required and forbidden edges
required_edges = (
    [('city', 'country')] +
    [('age_grp', 'salary')]
)

# Applying the constraints into ek as config parameter
ek = ExpertKnowledge(required_edges=required_edges)


# %% [markdown]
# # Optimization loop
# - Loop through ess range to get zero isolated nodes

# %%
# estimate_model function: HillClimbSearch and BDeu scoring method -> model
# start_dag fixed as nb_model
def estimate_model(df, ess):
    hc = HillClimbSearch(df)
    model = hc.estimate(
        scoring_method=BDeu(df, equivalent_sample_size=ess),
        start_dag=nb_model,
        max_indegree=None,
        expert_knowledge= ek
    )
    return model

# %%
# save_model_and_metadata using pickle and json
import pickle
import json
import os
def save_model_and_metadata(model, ess, is_best=False):
```

```
    # Create directory for ess runs -----------------------------------
    ess_models_dir = "ess_models_dir"
    os.makedirs(ess_models_dir, exist_ok=True)

    model_name = f"model_ess{ess}"

    # Saving pickle file of model -------------------------------------
    log(f"Saving model for ess={ess}...")
    pickle_filename = os.path.join(ess_models_dir, f"{model_name}.pkl")
    with open(pickle_filename, "wb") as f:
        pickle.dump(model, f)

    # Saving json file of model---------------------------------------
    model_data = {
        "nodes": list(model.nodes()),
        "edges": list(model.edges()),
        "ess": ess,
        "arcs": len(model.edges())
    }
    json_filename = os.path.join(ess_models_dir, f"{model_name}.json")
    with open(json_filename, "w") as f:
        json.dump(model_data, f, indent=4)

    status = "best " if is_best else ""
    print(f"Saved {status}model to '{pickle_filename}' and '{json_filename}'.")

# %%
# run_hill_climb_for_ess_range function looping through ess range
# calls estimate_model(df,ess) -> returns model
# calls save_model_and_metadata(model, ess, is_best=False) -> returns pkl json
is_best

import numpy as np
import random

np.random.seed(42)
random.seed(42)

# Main function looping both build naive bayes dag and estimate model
def run_hill_climb_for_ess_range(df, ess_range, seed=42):

    np.random.seed(seed)  # For reproducibility
    random.seed(seed)
    log(f"Starting hill climb search with seed={seed} ")
```

```
    max_arcs = 40
    best_model = None
    best_ess = None

    start_dag = nb_model
    arcs_data = []

    for ess in ess_range:
        log(f"Running HillClimb for equivalent_sample_size = {ess}")

        # Pass the seed into the estimator
        model = estimate_model(df, ess)

        arcs = len(model.edges())
        log(f"Model for ess={ess} has {arcs} arcs.")

        arcs_data.append({
            'model_name': f'model_ess{ess}',
            'len_edges': arcs
        })

        save_model_and_metadata(model, ess)

        if arcs > max_arcs:
            max_arcs = arcs
            best_model = model
            best_ess = ess
            log(f"New best model found with {max_arcs} arcs at ess={best_ess}")

    arcs_df = pd.DataFrame(arcs_data)
    arcs_df.to_csv("edges_df.csv", index=False)
    print("\nSaved 'edges_df.csv' with all arc counts.")

    if best_model:
        save_model_and_metadata(best_model, best_ess, is_best=True)
        print(f"Best model has {max_arcs} arcs and was found with
ess={best_ess}.")
    else:
        log("No best model estimated.")



# %%
# Run main function
import pickle
```

```
if __name__ == "__main__":
    # Skipping the import and pre-processing, use saved df_mod
    try:
        df_mod = pd.read_csv("df_mod.csv")
        df_mod = df_mod.astype('category')
        df = df_mod.copy()
        log("Dataset loaded and ready.")
    except FileNotFoundError:
        log("df_mod.csv not found. Please run the data preprocessing steps
first.")
        df = None

    if df is not None:
        # ess start at n100 because highest nlevels is 438
        run_hill_climb_for_ess_range(df, range(500,3000, 100))

# %%
# audit_connectivity: if there are isolated nodes
import json
import os
import pandas as pd

model_folder = "ess_models_dir"
files = [f for f in os.listdir(model_folder) if f.lower().endswith(".json")]

def audit_connectivity(model_path):
    with open(model_path, "r") as f:
        model = json.load(f)

    all_nodes = set(model["nodes"])
    edges = model["edges"]

    connected_nodes = set()
    for edge in edges:
        connected_nodes.update(edge)

    isolated_nodes = all_nodes - connected_nodes
    coverage_ratio = len(connected_nodes) / len(all_nodes)

    return {
        "model": os.path.basename(model_path),
        "ess": model.get("ess", None),
        "total_nodes": len(all_nodes),
        "connected_nodes": len(connected_nodes),
        "isolated_nodes": len(isolated_nodes),
```

```
        "coverage": coverage_ratio
    }

# Build audit summary DataFrame
audit_rows = []
for file in files:
    file_path = os.path.join(model_folder, file)
    audit_rows.append(audit_connectivity(file_path))

audit_summary = pd.DataFrame(audit_rows)
audit_summary = audit_summary.sort_values(by = 'ess', ascending = True)
audit_summary.to_csv('audit_summary.csv', index = False)

# Print minimum ESS with full connectivity
no_isolates = audit_summary[audit_summary["isolated_nodes"] == 0]
if not no_isolates.empty:
    min_ess = no_isolates["ess"].min()
    print(f"Minimum ESS with no isolated nodes: {min_ess}")
    print(f"Best model pickle filename: model_ess{min_ess}.pkl")
else:
    print("No model achieved full node connectivity.")


# %%
# Tabulate edges per ESS value in loop
import os
import pandas as pd
import json

model_folder = "ess_models_dir"
files = [f for f in os.listdir(model_folder) if f.lower().endswith(".json")]
rows = []

for f in files:
    file_path = os.path.join(model_folder, f)
    with open(file_path, 'r') as infile:
        dfjson = json.load(infile)

    ess = dfjson.get('ess')
    arcs = dfjson.get('arcs')

    if ess is not None and arcs is not None:
        rows.append({'ess': ess, 'arcs': arcs})

arcs_df = pd.DataFrame(rows)
```

```
arcs_df = arcs_df.sort_values(by = 'ess', ascending = True)
arcs_df.to_csv("arcs_summary.csv", index=False)
min_arcs = arcs_df[arcs_df['ess'] == min_ess]
print(f"min_ess = {min_ess}")
print(f"{min_arcs}")



# %% [markdown]
# # Fit CPDs using Bayesian Estimator

# %%
# Fit CPDs with Bayesian Estimator, ess = 3 -> model_be_fitted.pkl
# loaded_model as the best option so far
import pickle
from pgmpy.estimators import BayesianEstimator

# Choose model to plot -> model_ess1500.pkl
model_filename_be = input("Paste best model pkl filename:")
# Use as loaded_model
chosen_model_filepath = os.path.join("ess_models_dir", model_filename_be)
with open(chosen_model_filepath, "rb") as f:
    loaded_model_be_fit = pickle.load(f)

# Fit CPDs
loaded_model_be_fit.fit(df, estimator=BayesianEstimator,
               prior_type="BDeu",
               equivalent_sample_size=3)

# Save fitted version
os.makedirs("ess_models_fitted", exist_ok=True)
fitted_be_pkl_filename = os.path.join("ess_models_fitted", "model_be_fitted.pkl")
with open(fitted_be_pkl_filename, "wb") as f:
    pickle.dump(loaded_model_be_fit, f)

print(f"model_be_fitted.pkl created.")

# %% [markdown]
# # Fit CPDs using MLE

# %%
# Fit CPDs with MLE -> model_mle_fitted.pkl
# loaded_model as the best option so far
import pickle
from pgmpy.estimators import MaximumLikelihoodEstimator
```

```
# Best model pickle filename: model_ess1500.pkl
# Choose model to plot
model_filename_mle = input("Paste best model pkl filename:")
# Use as loaded_model
chosen_model_filepath = os.path.join("ess_models_dir", model_filename_mle)
with open(chosen_model_filepath, "rb") as f:
    loaded_model_mle_fit = pickle.load(f)

# Fit CPDs
loaded_model_mle_fit.fit(df, estimator=MaximumLikelihoodEstimator)

# Save fitted version
os.makedirs("ess_models_fitted", exist_ok=True)
fitted_mle_pkl_filename = os.path.join("ess_models_fitted",
"model_mle_fitted.pkl")
with open(fitted_mle_pkl_filename, "wb") as f:
    pickle.dump(loaded_model_mle_fit, f)

print(f"model_mle_fitted.pkl created.")

# %% [markdown]
# # Extract metadata from best model

# %%
# Extract metadata -> as "_metadata.txt"
import os
import pickle

def export_model_metadata(model, output_path):
    with open(output_path, "w") as f:
        f.write("Nodes:\n")
        f.write(", ".join(model.nodes()) + "\n\n")

        f.write("Edges:\n")
        for edge in model.edges():
            f.write(f"{edge[0]} -> {edge[1]}\n")
        f.write("\n")

        f.write("Cardinalities:\n")
        for var, card in model.get_cardinality().items():
            f.write(f"{var}: {card}\n")
        f.write("\n")

# Prompt for base filename (without extension)
```

```
model_base_name = input("Paste best FITTED model pkl filename (without .pkl):
").strip()

# Construct full paths
model_path = os.path.join("ess_models_fitted", f"{model_base_name}.pkl")
metadata_path = os.path.join("ess_models_fitted",
f"{model_base_name}_metadata.txt")

# Load model
with open(model_path, "rb") as f:
    model = pickle.load(f)

# Export metadata
export_model_metadata(model, metadata_path)


# %% [markdown]
# # Plots of DAGs

# %%
# Plot loaded_model using pyvis and view in browser to bypass issues
import os
import pickle
from pyvis.network import Network

# Choose model to plot -> model_ess1500.pkl
model_filename = input("Paste best model pkl filename:")
# Use as loaded_model
chosen_model_filepath = os.path.join("ess_models_dir", model_filename)
with open(chosen_model_filepath, "rb") as f:
    loaded_model = pickle.load(f)


net = Network(notebook=True, directed=True)
net.add_nodes(list(loaded_model.nodes()))
net.add_edges(list(loaded_model.edges()))

net.show("loaded_model.html")


# %%
# Plot loaded_model using nx

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

```
# Model to plot
model_filename = input("Paste best model pkl filename:")
chosen_model_filepath = os.path.join("ess_models_dir", model_filename)
# Use as loaded_model
with open(chosen_model_filepath, "rb") as f:
    loaded_model = pickle.load(f)

# Convert pgmpy model to a NetworkX DiGraph
G = nx.DiGraph(loaded_model.edges())

# Choose a layout (spring_layout is a good default)
pos = nx.spring_layout(G, seed=42)  # seed for reproducibility

# Draw nodes, edges, and labels
nx.draw(G, pos, with_labels=False, node_size=2000, node_color="lightblue",
arrowsize=20)
nx.draw_networkx_labels(G, pos, font_size=10, font_color="black")

plt.title("Bayesian Network Structure using Naive Bayes as Seed")
plt.axis("off")
plt.savefig(f"{chosen_model_filepath}.png")
plt.show()
```