```
# %% [markdown]
# Originated by: Sandy G. Cabanes
# # This generates a synthetic dataset from the model with the lowest ess that
achieves no floating nodes.

# %%
import time
import numpy as np
import pandas as pd
import pgmpy
import pickle

#from pgmpy.estimators import HillClimbSearch, BDeu,
BayesianEstimator,MaximumLikelihoodEstimator
#from pgmpy.models import BayesianNetwork
from pgmpy.sampling import BayesianModelSampling

# %%

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

# %%
# Use saved df_mod, convert to category, ready for modeling
df_mod = pd.read_csv("df_mod.csv")
df_mod = df_mod.astype('category')
df = df_mod.copy()
# df.head(1)

# %%
# Load saved Bayesian Estimation fitted model
chosen_model_filepath = "ess_models_fitted/model_be_fitted.pkl"
with open(chosen_model_filepath, "rb") as f:
    loaded_model = pickle.load(f)

# %%
# Suppress Warnings about probabilities difference ~1e-16
import warnings

# Suppress only the specific pgmpy warning about probability sum adjustment
warnings.filterwarnings(
    "ignore",
    message="Probability values don't exactly sum to 1. Differ by:*",
```

```
    module="pgmpy"
)

# %%
# Single pass: syn_df creation

n_samples = df.shape[0]
model = loaded_model
log(f"Sampling synthetic data: {n_samples} rows...")
sampler = BayesianModelSampling(model)
syn_df = sampler.forward_sample(
    size=n_samples,
    seed=42
)

for c in syn_df.columns:
    syn_df[c] = syn_df[c].astype('category')

syn_df.to_csv("syn_df0.csv", index = False)
log(f"Synthetic dataset shape: {syn_df.shape}")
log("Preview of synthetic data (first 5 rows):")
print(syn_df.head())

log("Pipeline complete.")
# syn_df0.csv holds the Single pass synthetic dataset

# %% [markdown]
# # Loop to detect duplicates

# %%
# Define the target number of duplicates
target_duplicates = 20

# Set a counter to prevent infinite loops
run_count = 0
max_runs = 5

# Initialize duplicated_real, min_duplicates
duplicated_real = np.inf
min_duplicates = np.inf
best_run_data = {}
run_data = {}

# %%
# Suppress Warnings about probabilities difference ~1e-16
```

```
import warnings

# Suppress only the specific pgmpy warning about probability sum adjustment
warnings.filterwarnings(
    "ignore",
    message="Probability values don't exactly sum to 1. Differ by:*",
    module="pgmpy"
)


# %%
# Start a while loop that runs until the target is met or max_runs is reached
while run_count < max_runs:

    # Increment the run counter
    run_count += 1
    print(f"Attempt number: {run_count}")

    # Setting a seed for reproducibility.
    # Set the seed using the run count.
    np.random.seed(run_count)

    # Generate new synthetic data in each loop iteration
    n_samples = df.shape[0]
    model = loaded_model
    log(f"Sampling synthetic data: {n_samples} rows...")
    sampler = BayesianModelSampling(model)
    syn_df = sampler.forward_sample(
             size=n_samples,
             seed=42
    )

    syn_df.to_csv(f"syn_df_runcount{run_count}.csv", index = False)

    # Add unique, non-overlapping row_ids to identify rows to drop from synthetic
    # Add tags so we can identify the source
    df_tagged = df.copy()
    df_tagged['row_id'] = range(10001, 10001 + len(df_tagged))
    df_tagged['source'] = 'real'

    syn_df_tagged = syn_df.copy()
    syn_df_tagged['row_id'] = range(20001, 20001 + len(syn_df_tagged))
    syn_df_tagged['source'] = 'synthetic'

    # Combine datasets
```

```
    concat_df = pd.concat([df_tagged, syn_df_tagged], ignore_index=True)

    # Count duplicates using composite factors
    # This combines factors to identify full-row duplicates across both datasets
    columns_to_combine = [col for col in concat_df.columns if col not in
['row_id', 'source']]
    concat_df['combinedfactors'] = concat_df[columns_to_combine].apply(lambda
row: '|'.join(row.values.astype(str)), axis=1)

    dup_counts =
concat_df.groupby('combinedfactors')['source'].value_counts().unstack(fill_value=
0)
    dup_counts = dup_counts.reset_index()
    dup_counts.columns = ['combinedfactors', 'count_real', 'count_synthetic']
    dup_counts.to_csv("dup_counts.csv", index = False)

    dup_check = dup_counts[(dup_counts['count_real'] > 0) &
(dup_counts['count_synthetic'] > 0)]
    dup_check['count'] = dup_check['count_real'] + dup_check['count_synthetic']
    dup_check['sources'] = 'real,synthetic'
    dup_check.to_csv("dup_check.csv", index = False)

    # This part finds the specific row IDs for the duplicates
    dup_check_with_ids = dup_check.merge(concat_df[['combinedfactors', 'row_id',
'source']], on='combinedfactors', how='left')

    dup_check_with_ids['row_id_real'] = dup_check_with_ids.apply(
        lambda row: ','.join(map(str, row['row_id'][row['source'] == 'real'])),
axis=1
    )
    dup_check_with_ids['row_id_syn'] = dup_check_with_ids.apply(
        lambda row: ','.join(map(str, row['row_id'][row['source'] ==
'synthetic'])), axis=1
    )


    # Update the duplicated_real value, must be <20
    duplicated_real = sum(dup_check_with_ids['count_real'])
    run_data[run_count] = duplicated_real

    print(f"run_count {run_count} : duplicated_real {duplicated_real}")

    # Check if current run has less duplicates, if yes update min_duplicates
    if run_count == 1 or duplicated_real < min_duplicates or duplicated_real ==
0:
```

```
        min_duplicates = duplicated_real
        best_run_data = {
            'dup_check_with_ids': dup_check_with_ids,
            'syn_df': syn_df,
            'syn_df_tagged': syn_df_tagged,
            'run_count': run_count
        }
        print(f"New minimum duplicates found: {min_duplicates}")

# Final outputs after the loop
if duplicated_real <= target_duplicates:
    print("\nTarget number of duplicates reached!")
else:
    print(f"\nCould not reach the target number of duplicates within the maximum
number of runs. The best result was {min_duplicates} duplicates with seed
{best_run_data['run_count']}")

# Export the best results found
best_run_data['dup_check_with_ids'].to_csv("dup_check_with_ids_bestrun.csv",
index=False)
best_run_data['syn_df'].to_csv("syn_df_bestrun.csv", index=False)
best_run_data['syn_df_tagged'].to_csv("syn_df_tagged_bestrun.csv", index=False)


# %% [markdown]
# # Check distributions and splits

# %%
# Check distributions
# Long df
df_tagged = df.copy()
df_tagged['row_id'] = range(10001, 10001 + len(df_tagged))
df_tagged['source'] = 'Original'

syn_df_tagged = pd.read_csv("syn_df_bestrun.csv")
syn_df_tagged['row_id'] = range(20001, 20001 + len(syn_df_tagged))
syn_df_tagged['source'] = 'Synthetic'


# %%
# Bar plot clustered Original vs Synthetic
import matplotlib.pyplot as plt
import seaborn as sns
df_combined = pd.concat([df_tagged, syn_df_tagged], axis = 0)
```

```
plotcluster = sns.histplot(data = df_combined, x = 'salary', hue = 'source',
multiple='dodge',
    shrink=0.8)
plotcluster.tick_params(axis='x', labelrotation=90)
plt.show()

# %%
# Quick check of salary vs. educstat
import matplotlib.pyplot as plt
import seaborn as sns
sns.color_palette('pastel')
salary_order = sorted(syn_df['salary'].unique())
syn_df['salary'] = pd.Categorical(syn_df['salary'], categories=salary_order,
ordered=True)

plot = sns.histplot(data = syn_df, x = 'salary', hue = 'educstat', multiple =
'stack')
plot.tick_params(axis='x', labelrotation=90)
plt.show()


# %%
# Plot of salary by educstat - df
sns.color_palette('pastel')
plot = sns.histplot(data = df, x = 'salary', hue = 'educstat', multiple =
'stack')
plot.tick_params(axis='x', labelrotation=90)
plt.show()


# %% [markdown]
# # Double check duplicates

# %%
# Count duplicates using composite factors
# Recall: df_combined = pd.concat([df_tagged, syn_df_tagged], axis = 0)
# This combines factors to identify full-row duplicates across both datasets
columns_to_combine = [col for col in df_combined.columns if col not in ['row_id',
'source']]
df_combined['combinedfactors'] = df_combined[columns_to_combine].apply(lambda
row: '|'.join(row.values.astype(str)), axis=1)

dup_counts =
df_combined.groupby('combinedfactors')['source'].value_counts().unstack(fill_valu
e=0)
```

```
dup_counts = dup_counts.reset_index()
dup_counts.columns = ['combinedfactors', 'count_real', 'count_synthetic']
dup_counts.to_csv("dup_counts.csv")

dup_check = dup_counts[(dup_counts['count_real'] > 0) &
(dup_counts['count_synthetic'] > 0)]
dup_check['count'] = dup_check['count_real'] + dup_check['count_synthetic']
dup_check['sources'] = 'real,synthetic'
dup_check.to_csv("dup_check.csv")
dup_check.head()


# %%
# This part finds the specific row IDs for the duplicates
dup_check_with_ids = dup_check.merge(df_combined[['combinedfactors', 'row_id',
'source']], on='combinedfactors', how='left')

dup_check_with_ids['row_id_real'] = dup_check_with_ids.apply(
        lambda row: ','.join(map(str, row['row_id'][row['source'] == 'real'])),
axis=1
    )
dup_check_with_ids['row_id_syn'] = dup_check_with_ids.apply(
        lambda row: ','.join(map(str, row['row_id'][row['source'] ==
'synthetic'])), axis=1
    )
dup_check_with_ids.to_csv("dup_check_with_ids.csv")
dup_check_with_ids

# %%
import os
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Create directory for plots
plots_dir = "plots_dir"
os.makedirs(plots_dir, exist_ok=True)

# Get timestamp for filenames
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

# %%
# Generate and save plots
colnames = syn_df_tagged.columns
```

```
df_combined = pd.concat([df_tagged, syn_df_tagged], axis=0)

saved_files = []

for col in colnames:
    print(f"Generating plot for: {col}")
    plt.figure(figsize=(8, 4))
    plotcluster = sns.histplot(
        data=df_combined,
        x=col,
        hue='source',
        multiple='dodge',
        shrink=0.8
    )
    plotcluster.tick_params(axis='x', labelrotation=90)

    # Truncate labels for readability
    plotcluster.set_xticklabels([
        str(label.get_text())[:25] for label in plotcluster.get_xticklabels()
    ])

    plt.title(f"Distribution of {col} by source")
    plt.tight_layout()

    # Save plot
    filename = f"{col}_{timestamp}.png"
    filepath = os.path.join(plots_dir, filename)
    plt.savefig(filepath, dpi=150)
    plt.close()

    saved_files.append(filename)

# %%
# Output as html
html_path = os.path.join(plots_dir, f"_summary_{timestamp}.html")

with open(html_path, "w") as f:
    f.write("<html><head><title>Cluster Plots Summary</title></head><body>\n")
    f.write("<h1>Distribution of Original vs. Synthetic Dataset</h1>\n")
    for filename in saved_files:
        f.write(f"<h2>{filename.split('_')[0]}</h2>\n")
        f.write(f'<img src="{filename}" style="width:800px;"><br><br>\n')
    f.write("</body></html>")
```