

System requirements:

Colab:

```
!pip install pgmpy networkx pandas numpy scipy matplotlib tqdm joblib bnlearn requests  
statsmodels lightgbm scikit-learn graphviz pydot
```

```
import os
```

```
!python "/content/drive/MyDrive/causal-discovery-project/run_simulation.py"
```

Windows:

Setting up:

Click twice on setup_windows.bat file.

Or go to your project folder in your command line and run

```
.\setup_windows.bat.
```

Running:

1) Via command line:

Go to your project folder in your command line and run

```
venv\Scripts\activate
```

```
python run_simulation.py
```

```
deactivate
```

2) Visual Studio Code:

Open project folder.

Press Ctrl + Shift + P – choose Select Interpreter – our virtual environment.

Execute run_simulation.py.

Linux:

Setting up:

Go to your project folder in your command line and run

```
bash setup_linux.sh.
```

Running:

Go to your project folder in your command line and run

```
source venv/bin/activate
```

```
python run_simulation.py
```

```
deactivate
```

Purpose:

Causal diagrams are oriented graphs meant for representing relationships between variables in datasets (some of variables become causes and some – effects). Each rib in such a diagram represents association between two variables, direction of the rib shows which one is the cause and which one is the effect. For example, we have a dataset on many people with only two variables: Sleep_quality and Coffee_consumption. Our diagram will consist of two nodes (marked accordingly) with an arrow from Sleep_quality to Coffee_consumption (people who sleep bad tend to drink more coffee, let's assume coffee doesn't make your sleep worse). The purpose of my code is automatically constructing a causal diagram using two kinds of data: observational (that is – our original dataset) and experimental (usually it is being collected in the real world, but we will use an artificial approximation instead). For testing the workflow I use diagrams from bnlearn library to see how well my code reconstructs them. The code is meant for **discrete** data.

Stage 1:

I generate data based on a bnlearn diagram (getting an observational dataset), then run PC algorithm out of the box on this data. It gives me a skeleton of the diagram – some ribs are not oriented, only association is evident (it is clear that Sleep_quality and Coffee_consumption are associated, but how – the machine doesn't yet know).

Stage 2:

Bnlearn diagrams represent relationships between variables by conditional probabilities of one variable over a set of others (parents in the diagram). I run quasi-experiments merely by

changing distribution of one variable and generating data again. For example, we change distribution of Sleep_quality (in the real world it would be providing therapy for people with bad sleep), generate data and see that Coffee_consumption went down. We conclude that Sleep_quality is the cause and Coffee_consumption – the effect.

Quasi-experiments and usual experiments:

Usual experiment: we change some variable in any way we want – for example, if we chose to change Coffee_consumption we could forbid people buy coffee it or we could it so cheap people would almost certainly buy it.

Quasi-experiment: we change some variable only to a certain extent – for example, a doctor cannot make patients take a certain drug, he can only encourage or discourage them to do so.

I use quasi-experiments because they are more general (usual experiments are a special case of them).

Strategies:

Orienting our diagram consists of repeating the following steps:

- 1) choosing a variable whose distribution will change (on which we will run a quasi-experiment);
- 2) generating data (observational and experimental);
- 3) trying to orient all adjacent ribs (not oriented so far) using statistical tests;
- 4) orienting some edges to avoid invalid dags (loops and v-structures).

There is a question: what variable should we choose next? After all, in the real world running experiments is costly. There are several strategies:

- 1) greedy – choosing a variable with most adjacent edges (not oriented so far);
- 2) entropy – choosing a variable that, on average, makes future orienting more efficient;
- 3) minimax – choosing a variable that, in the worst case, makes future orienting more efficient.

Statistical tests:

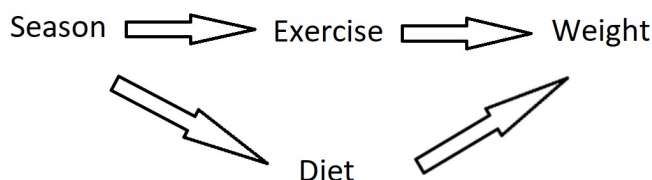
- 1) Marginal:

If, when we modify variable's distribution (Sleep_quality), other variable's distribution (Coffee_consumption) also changes, we conclude that first variable is the cause.

- 2) Conditional:

If, when we modify variable's distribution (Coffee_consumption), relationship itself with the second variable changes (Sleep_quality), we conclude that second variable is the cause. (We made coffee cheap, so now even people with good sleep drink it, we no longer can say that if a man drinks a lot of coffee he must be sleeping badly).

Note: we have to take into account all possible parents of the second variable. For example, we want to orient an edge between Exercise and Weight. We changed Exercise's distribution (released an app that prompts users to exercise in winter).



Before app release			
Season	Exercise	Diet	Weight
Winter	Low	Bad	Large
Spring	Medium	Average	Slightly overweight

Summer	High	Good	Normal
Autumn	Medium	Average	Slightly overweight
After app release			
Season	Exercise	Diet	Weight
Winter	High	Bad	Slightly overweight
Spring	Medium	Average	Slightly overweight
Summer	High	Good	Normal
Autumn	Medium	Average	Slightly overweight

If we take into account only Exercise (not looking at Diet) we will conclude that the relationship changed (now we can't say that if a man exercises a lot his weight must be normal). Having in mind that Diet in winter is different from Diet in summer we will not make such a mistake.

Running unittests (for development):

Colab:

```
os.chdir('/content/drive/MyDrive/causal-discovery-project')
```

```
!python -m unittest tests.test_graph_utils
```

```
!python -m unittest tests.test_intervention
```

```
!python -m unittest tests.test_orienting_alg
```

```
!python -m unittest tests.test_statistical_tests
```

Windows & Linux:

Go to your project folder in command line and:

Activate virtual environment.

```
python -m unittest tests.test_graph_utils
```

```
python -m unittest tests.test_intervention
```

```
python -m unittest tests.test_orienting_alg
```

```
python -m unittest tests.test_statistical_tests
```