**Annexe A : Données et Préparation (R)**

Script : prepare prices.R

## Objectif & paramètres

* **Objetif** : préparation des prix ajustés (Yahoo Finance), interpolation des trous internes, purge des colonnes incomplètes, exports prix et mapping index→ticker.
* **Entrées** : Obligatoire pour run le code compositions/eurostoxx50\_composition.csv, compositions/nasdaq100\_composition.csv
* **Sorties**: prices/\*\_prices.csv, data/\*\_prices\_values\_only.csv, data/\*\_ticker\_order.csv

## Code

1 ## ==========Mémoire Fabio Cardoso  
 2 ## Objectif du Code R: préparer PRIX & RENDEMENTS  
 3 ## Etapes :  
 4 ## - Téléchargements Yahoo  
 5 ## - - Interpolation linéaire des trous INTERNES (na.approx)  
 6 ## - - -Suppression des colonnes avec NA restants (début/fin)  
 7 ## - - - - Exports: avec dates (contrôle) + "values-only" (scripts Python)  
 8 ## - - - - - Sauvegarde de l’ordre des tickers (index -> ticker)  
 9 ## ===============================================================  
10   
11 ## ========= 0) Répertoire de travail =========  
12 setwd("C:/Users/rebel/Desktop/Mémoire Recherche") #Changer si run ailleurs  
13   
14 ## ========= 1) Packages =========  
15 install.packages(c("quantmod","xts","zoo"))  
16 library(quantmod)  
17 library(xts)  
18 library(zoo)  
19   
20 ## ========= 2) Dossiers & période =========  
21 dir.create("compositions", showWarnings = FALSE)  
22 dir.create("prices", showWarnings = FALSE)  
23 dir.create("data", showWarnings = FALSE)  
24   
25 start\_date <- as.Date("2019-01-01")  
26 end\_date <- as.Date("2025-07-01")  
27   
28 ## ========= 3) Compositions =========  
29 # Format attendu: 2 colonnes: Nom, Symbol  
30 euro\_comp <- read.csv("compositions/eurostoxx50\_composition.csv", stringsAsFactors = FALSE)  
31 nasdaq\_comp <- read.csv("compositions/nasdaq100\_composition.csv", stringsAsFactors = FALSE)  
32 colnames(euro\_comp) <- c("Nom","Symbol")  
33 colnames(nasdaq\_comp) <- c("Nom","Symbol")  
34 euro\_comp$Symbol <- trimws(euro\_comp$Symbol)  
35 nasdaq\_comp$Symbol <- trimws(nasdaq\_comp$Symbol)  
36   
37 ## ========= 4) Téléchargement PRIX (Yahoo) =========  
38 fetch\_prices <- function(tickers, from, to, pause=0.25){  
39 lst <- list(); keep <- character(0)  
40 for (tk in tickers) {  
41 ok <- tryCatch({  
42 x <- getSymbols(tk, src="yahoo", from=from, to=to, auto.assign=FALSE)  
43 px <- Ad(x); colnames(px) <- tk  
44 lst[[tk]] <- px  
45 TRUE  
46 }, error=function(e) FALSE)  
47 if (ok) keep <- c(keep, tk)  
48 Sys.sleep(pause)  
49 }  
50 if (length(keep)) do.call(merge, lst[keep]) else NULL  
51 }  
52   
53 euro\_prices\_raw <- fetch\_prices(euro\_comp$Symbol, start\_date, end\_date)  
54 nasdaq\_prices\_raw <- fetch\_prices(nasdaq\_comp$Symbol, start\_date, end\_date)  
55   
56 ## ========= 5) Imputation linéaire (trous internes) & purge colonnes NA =========  
57 # na.approx remplit uniquement ENTRE deux points pas les bords.   
58 # On supprime les colonnes avec NA restants (séries incomplètes).  
59   
60 lin\_impute <- function(X){  
61 if (is.null(X) || ncol(X) == 0) return(X)  
62 Xi <- zoo::na.approx(X) # interpole les trous INTERNES  
63 Xi <- Xi[, colSums(is.na(Xi)) == 0] # supprime colonnes encore incomplètes (début/fin)  
64 Xi  
65 }  
66 euro\_prices <- lin\_impute(euro\_prices\_raw)  
67 nasdaq\_prices <- lin\_impute(nasdaq\_prices\_raw)  
68   
69 cat("Conservés après imputation — EURO:", ncol(euro\_prices),  
70 "| NASDAQ:", ncol(nasdaq\_prices), "\n")  
71   
72 ## ========= 6) Exports PRIX =========  
73 # 6.1) Avec dates (contrôle visuel)  
74 write.zoo(euro\_prices, file = "prices/eurostoxx50\_prices.csv", sep = ",")  
75 write.zoo(nasdaq\_prices, file = "prices/nasdaq100\_prices.csv", sep = ",")  
76   
77 # 6.2) "Values-only" (Pour VARLiNGAM & prédiction)  
78 euro\_values <- as.data.frame(zoo::coredata(euro\_prices))  
79 nasdaq\_values <- as.data.frame(zoo::coredata(nasdaq\_prices))  
80 write.csv(euro\_values, "data/eurostoxx50\_prices\_values\_only.csv", row.names = FALSE)  
81 write.csv(nasdaq\_values, "data/nasdaq100\_prices\_values\_only.csv", row.names = FALSE)  
82   
83 # 6.3) Mapping index -> ticker (ordre des colonnes, pour les graphes de causalité)  
84 euro\_map <- data.frame(index = seq\_len(ncol(euro\_values)) - 1, ticker = colnames(euro\_values))  
85 nasdaq\_map <- data.frame(index = seq\_len(ncol(nasdaq\_values)) - 1, ticker = colnames(nasdaq\_values))  
86 write.csv(euro\_map, "data/eurostoxx50\_ticker\_order.csv", row.names = FALSE)  
87 write.csv(nasdaq\_map, "data/nasdaq100\_ticker\_order.csv", row.names = FALSE)

**Annexe B : Découverte Causale**

**Script :** causal\_discovery\_varlingam.py

## Objectif & paramètres

* **Objectif :** Estimer le graphe causal VARLiNGAM et l’écrire en adjacency list.
* **Entrées :** eurostoxx50\_prices\_values\_only et nasdaq100\_prices\_values\_only
* **Sorties:** causal\_graphs/graphs/(market)\_graph\_(ALGO)\_lag\_(L).txt

## Code

## 1. causal\_discovery\_varlingam.py

Chemin : C:\Users\rebel\Desktop\Mémoire Recherche\causal\_discovery\_varlingam.py

Dernière modification : 2025-09-18 20:45

1

2 """

3 Découverte causale

4

5 """

6 from \_\_future\_\_ import annotations

7 import os

8 import sys

9 import numpy as np

10 import networkx as nx

11 import lingam

12

13 # 1)

14

15 def load\_prices\_matrix(path: str) -> np.ndarray:

16

17 data = np.genfromtxt(path, delimiter=",", skip\_header=1)

18 if data.ndim != 2:

19 raise ValueError(f"CSV mal formé (pas 2D) : {path}")

20 return data

21

22

23

24 # 2) VARLiNGAM

25

26 def fit\_varlingam(prices: np.ndarray, lag: int) -> np.ndarray:

27

28 model = lingam.VARLiNGAM(lags=lag)

29 model.fit(prices)

30 A\_all = np.asarray(model.adjacency\_matrices\_) # (K, N, N)

31 # Diagnostic rapide :

32 nnz\_per\_slice = [int((np.abs(A\_all[k]) > 0).sum()) for k in range(A\_all.shape[0])]

33 print(f"[INFO] Slices K={A\_all.shape[0]} (attendu {lag} ou {lag}+1) | nonzeros/lag: {nnz\_per\_slice}")

34 return A\_all

35

36

37 def build\_summary\_matrix(A\_all: np.ndarray, lag: int) -> np.ndarray:

38

39 K, N, \_ = A\_all.shape

40

41 if K == lag + 1:

42 A\_lags = A\_all[1:] # k = 1..L

43 else:

44 A\_lags = A\_all # déjà uniquement les lags retardés

45

46 summary = np.sum(np.abs(A\_lags), axis=0).copy() # (N, N)

47 np.fill\_diagonal(summary, 0.0) # no self-loop

48 return summary

49

50

51

52 # 3) Graphe orienté

53

54 def summary\_graph\_from\_matrix(summary: np.ndarray) -> nx.DiGraph:

55

56 G = nx.from\_numpy\_array(summary.T, create\_using=nx.DiGraph)

57

58 for \_, \_, d in G.edges(data=True):

59 d.clear()

60 return G

61

62

63 def print\_graph\_diagnostics(G: nx.DiGraph) -> None:

64 N = G.number\_of\_nodes()

65 E = G.number\_of\_edges()

66 density = E / (N \* (N - 1)) if N > 1 else 0.0

67 indeg = [d for \_, d in G.in\_degree()]

68 outdeg = [d for \_, d in G.out\_degree()]

69 q = lambda a, p: sorted(a)[int(p \* (len(a) - 1))] if a else 0

70 print(f"[DIAG] nodes={N} | edges={E} | densité≈{density:.3f}")

71 print(f" in-degree : médiane={q(indeg,0.5)} | P90={q(indeg,0.9)} | max={max(indeg) if indeg else 0}")

72 print(f" out-degree : médiane={q(outdeg,0.5)} | P90={q(outdeg,0.9)} | max={max(outdeg) if outdeg else 0}")

73

74

75

76 # 4) Main

77

78 def main():

79

80 if len(sys.argv) < 5:

81 print("Usage: python causal\_discovery\_varlingam.py <data\_file> <num\_lags> <market\_name> <algorithm>")

82 sys.exit(1)

83

84 data\_file = sys.argv[1]

85 num\_lags = int(sys.argv[2])

86 market\_name = sys.argv[3]

87 algorithm = sys.argv[4]

88

89 out\_dir = "./causal\_graphs/graphs"

90 os.makedirs(out\_dir, exist\_ok=True)

91

92

93 prices = load\_prices\_matrix(data\_file)

94 print(f"[RUN] market={market\_name} | lag={num\_lags} | data={prices.shape} | algo={algorithm}")

95

96

97 A\_all = fit\_varlingam(prices, num\_lags)

98 summary = build\_summary\_matrix(A\_all, num\_lags)

99

100

101 G = summary\_graph\_from\_matrix(summary)

102 print\_graph\_diagnostics(G)

103

104

105 out\_path = os.path.join(out\_dir, f"{market\_name}\_graph\_{algorithm}\_lag\_{num\_lags}.txt")

106 nx.write\_edgelist(G, out\_path, data=False, delimiter=" ")

107 print(f"[SAVE] {out\_path}")

108

109

110 out\_csv = os.path.join(out\_dir, f"{market\_name}\_graph\_{algorithm}\_lag\_{num\_lags}.csv")

111 edges = np.array(list(G.edges()), dtype=int)

112 np.savetxt(out\_csv, edges, delimiter=",", fmt="%d", header="src,dst", comments="")

113 print(f"[SAVE] {out\_csv}")

114

115 if \_\_name\_\_ == "\_\_main\_\_":

116 main()

**Annexe C : Prédiction**

Script : predict.py

## Objectif & paramètres :

* **Objectif :** Produire les prédictions one-step-ahead par titre à partir des parents causaux et la baseline Self
* **Entrées :** eurostoxx50\_prices\_values\_only et nasdaq100\_prices\_values\_only, causal\_graphs/graphs/(market)\_graph\_(ALGO)\_lag\_(L).txt
* **Sorties:** predictions(market)predictions\_(ALGO)\_lag\_(L).csv et predictions(market)\_predictions\_self\_(ALGO)\_lag\_(L).csv

## Code

## 1. predict.py

Chemin : C:\Users\rebel\Desktop\Mémoire Recherche\predict.py

Dernière modification : 2025-09-18 20:49

1

2 """

3 PREDICT

4

5 """

6

7 import os

8 import sys

9 import numpy as np

10 import pandas as pd

11 import networkx as nx

12 from sklearn.linear\_model import LinearRegression

13

14 # 1) Prédiction

15

16 def predict\_single(data, lag, G, stock\_index, train\_frac=0.8):

17

18 predicted = []

19

20

21 causes\_name = list(G.predecessors(str(stock\_index)))

22 causes\_index = [int(i) for i in causes\_name]

23

24

25 causes = data[:, causes\_index] if len(causes\_index) > 0 else np.empty((data.shape[0], 0))

26 target = data[:, stock\_index]

27

28

29 Y = target[lag:]

30

31

32 X = np.empty((0, lag \* max(1, len(causes\_index))))

33 for t in range(lag, data.shape[0]):

34 if causes.size > 0:

35

36 lagged = causes[(t - lag):t, :]

37 features = np.concatenate(lagged)

38 else:

39

40 features = np.zeros(lag)

41 X = np.vstack([X, features])

42

43

44 train\_length = int(len(Y) \* train\_frac)

45

46

47 for t in range(train\_length, len(Y)):

48 model = LinearRegression()

49 model.fit(X[:t, :], Y[:t])

50 y\_hat = model.predict(X[t, :].reshape(1, -1))

51 predicted.append(float(y\_hat[0]))

52

53 return predicted

54

55

56

57 # 2) Prédictions pour TOUTES les colonnes (j=0..N-1)

58

59 def predict\_batch(data, lag, G, train\_frac=0.8):

60

61 predictions = np.empty((0, 0))

62 for j in range(data.shape[1]):

63 print(f"[PRED] actif {j}")

64 col\_pred = predict\_single(data, lag, G, j, train\_frac)

65 col\_pred = np.asarray(col\_pred).reshape(-1, 1)

66 predictions = np.hstack((predictions, col\_pred)) if predictions.size else col\_pred

67 return predictions

68

69

70

71 # 3) train\_frac

72

73 def \_parse\_train\_frac(argv, default=0.8):

74

75 if "--train\_frac" in argv:

76 i = argv.index("--train\_frac")

77 if i + 1 < len(argv):

78 try:

79 return float(argv[i + 1])

80 except ValueError:

81 pass

82 return default

83

84

85

86 # 4) Main

87

88 if \_\_name\_\_ == "\_\_main\_\_":

89

90 if len(sys.argv) < 6:

91 print("Usage: python predict.py data.csv graph.txt L market algo [--train\_frac 0.8]")

92 sys.exit(1)

93

94

95 output\_directory = "./predictions"

96 os.makedirs(output\_directory, exist\_ok=True)

97

98

99 data\_filename = sys.argv[1]

100 causal\_graph\_filename= sys.argv[2]

101 num\_lags = int(sys.argv[3])

102 market\_name = sys.argv[4]

103 algorithm = sys.argv[5]

104

105

106 train\_frac = \_parse\_train\_frac(sys.argv, default=0.8)

107 print(f"[CONFIG] market={market\_name} | lag={num\_lags} | algo={algorithm} | train\_frac={train\_frac:.2f}")

108

109 df = pd.read\_csv(data\_filename, delimiter=",", header=0)

110 tickers = df.columns.tolist()

111 data = df.to\_numpy()

112

113

114 G = nx.read\_adjlist(causal\_graph\_filename, create\_using=nx.DiGraph)

115

116

117 # Prédictions "causal"

118

119 print("[RUN] Making Predictions (causal)")

120 preds = predict\_batch(data, num\_lags, G, train\_frac=train\_frac)

121 out\_path = os.path.join(output\_directory, f"{market\_name}\_predictions\_{algorithm}\_lag\_{num\_lags}.csv")

122 np.savetxt(out\_path, preds, delimiter=",", header=",".join(tickers), comments="")

123 print("[SAVE]", out\_path)

124

125

126 # Baseline "self-cause only"

127 summary\_matrix = np.eye(data.shape[1])

128 G\_self = nx.from\_numpy\_array(summary\_matrix.T, create\_using=nx.DiGraph)

129

130 for \_, \_, d in G\_self.edges(data=True):

131 d.pop("weight", None)

132 G\_self = nx.relabel\_nodes(G\_self, lambda x: str(x))

133

134 print("[RUN] Making Predictions (self-cause)")

135 preds\_self = predict\_batch(data, num\_lags, G\_self, train\_frac=train\_frac)

136 out\_self = os.path.join(output\_directory, f"{market\_name}\_predictions\_self\_{algorithm}\_lag\_{num\_lags}.csv")

137 np.savetxt(out\_self, preds\_self, delimiter=",", header=",".join(tickers), comments="")

138 print("[SAVE]", out\_self)

**Annexe D: Backtest**

Script: backtest.py

## Objectif & paramètres :

* **Objectif :** Construire le portefeuille long/short à partir des prédictions, calculer AR net de coûts et exporter la grille AR(k), les élections journalières winners/losers pour un k et les PnL quotidien.
* **Entrées :** eurostoxx50\_prices\_values\_only et nasdaq100\_prices\_values\_only, predictions(market)predictions\_(ALGO)\_lag\_(L).csv et predictions(market)\_predictions\_self\_(ALGO)\_lag\_(L).csv
* **Sorties:** 
  + Grille: (market)\_backtest\_returns\_(ALGO)\_lag\_(L).csv
  + Sélections : (market)\_(k)\_(winners|losers)\_(ALGO)\_lag\_(L).csv
  + PnL quotidien: (market)\_daily\_portfolio\_returns\_(k)\_(ALGO)\_lag\_(L).csv.

## Code

## 1. backtest.py

Chemin : C:\Users\rebel\Desktop\Mémoire Recherche\backtest.py

Dernière modification : 2025-09-18 20:49

1

2 """

3 BACKTEST — Long/Short

4

5 """

6

7 import os

8 import sys

9 import csv

10 import numpy as np

11 import matplotlib.pyplot as plt

12

13 # 1) Cœur du backtest

14

15 def calculate\_annualized\_portfolio\_returns(data: np.ndarray,

16 predictions: np.ndarray,

17 num\_winners: int,

18 cost\_per\_day: float = 0.001):

19

20

21 data\_backtest = data[-(predictions.shape[0] + 1):]

22 predicted\_returns = (predictions - data\_backtest[:-1]) / data\_backtest[:-1]

23 real\_returns = (data\_backtest[1:] - data\_backtest[:-1]) / data\_backtest[:-1]

24 winners = np.argpartition(predicted\_returns, -num\_winners, axis=1)[:, -num\_winners:]

25 losers = np.argpartition(predicted\_returns, num\_winners, axis=1)[:, :num\_winners]

26 portfolio\_returns = []

27 for t in range(predictions.shape[0]):

28 winner\_return = np.mean(real\_returns[t, winners[t]])

29 loser\_return = np.mean(real\_returns[t, losers[t]])

30 daily\_pnl = winner\_return - loser\_return - cost\_per\_day

31 portfolio\_returns.append(daily\_pnl)

32

33 cumulative\_portfolio\_return = np.exp(np.sum(np.log(np.array(portfolio\_returns) + 1.0))) - 1.0

34 annualized\_return = (1.0 + cumulative\_portfolio\_return) \*\* (252.0 / predictions.shape[0]) - 1.0

35

36 return annualized\_return, winners, losers, portfolio\_returns

37

38

39 # 2) Main

40

41 if \_\_name\_\_ == "\_\_main\_\_":

42

43 if len(sys.argv) < 7:

44 print("Usage: data\_file prediction\_file num\_lags market\_name algorithm test\_winner\_num")

45 sys.exit(1)

46 output\_directory = "./backtesting"

47 os.makedirs(output\_directory, exist\_ok=True)

48

49 data\_filename = sys.argv[1]

50 predictions\_filename = sys.argv[2]

51 num\_lags = int(sys.argv[3])

52 market\_name = sys.argv[4]

53 algorithm = sys.argv[5]

54 test\_winner\_num = int(sys.argv[6])

55

56 data = np.genfromtxt(data\_filename, delimiter=",", skip\_header=1)

57 predictions = np.genfromtxt(predictions\_filename, delimiter=",", skip\_header=1)

58 predictions\_self\_filename = predictions\_filename.replace("\_predictions\_", "\_predictions\_self\_")

59 predictions\_self = np.genfromtxt(predictions\_self\_filename, delimiter=",", skip\_header=1)

60

61 print(f"[RUN] Backtesting market={market\_name} | lag={num\_lags} | algo={algorithm}")

62

63

64 # 2.1) Balayage de k

65

66 n\_assets = data.shape[1]

67 n\_winners\_range = np.arange(1, max(int(0.2 \* n\_assets), 5))

68

69 ar = [calculate\_annualized\_portfolio\_returns(data, predictions, k)[0]

70 for k in n\_winners\_range]

71 ar\_self = [calculate\_annualized\_portfolio\_returns(data, predictions\_self, k)[0]

72 for k in n\_winners\_range]

73

74

75 backtest\_output\_filename = os.path.join(

76 output\_directory,

77 f"{market\_name}\_backtest\_returns\_{algorithm}\_lag\_{num\_lags}.csv"

78 )

79 backtest\_returns = np.column\_stack((n\_winners\_range, ar, ar\_self))

80 with open(backtest\_output\_filename, mode="w", newline="") as file:

81 writer = csv.writer(file)

82 writer.writerow(["winner\_num", "ar", "ar\_self"])

83 writer.writerows(backtest\_returns)

84 print("[SAVE]", backtest\_output\_filename)

85

86 plt.figure(figsize=(7, 5))

87 plt.plot(n\_winners\_range, ar, label="Causal discovery", linewidth=2, color="C0")

88 plt.plot(n\_winners\_range, ar\_self, label="Self-cause only", linewidth=2, color="C1", linestyle="--")

89 plt.xlabel("Number of Winners/Losers (k)")

90 plt.ylabel("Annualized return")

91 plt.legend()

92 plt.tight\_layout()

93 backtest\_plot\_filename = os.path.join(

94 output\_directory,

95 f"{market\_name}\_backtest\_returns\_plot\_{algorithm}\_lag\_{num\_lags}.png"

96 )

97 plt.savefig(backtest\_plot\_filename, dpi=160, bbox\_inches="tight")

98 plt.close()

99 print("[SAVE]", backtest\_plot\_filename)

100

101

102 # 2.2) Sauvegardes

103

104 winner\_filename = os.path.join(

105 output\_directory,

106 f"{market\_name}\_{test\_winner\_num}\_winners\_{algorithm}\_lag\_{num\_lags}.csv"

107 )

108 loser\_filename = os.path.join(

109 output\_directory,

110 f"{market\_name}\_{test\_winner\_num}\_losers\_{algorithm}\_lag\_{num\_lags}.csv"

111 )

112 daily\_port\_return\_filename = os.path.join(

113 output\_directory,

114 f"{market\_name}\_daily\_portfolio\_returns\_{test\_winner\_num}\_{algorithm}\_lag\_{num\_lags}.csv"

115 )

116

117 \_, winners, losers, daily\_port\_returns = calculate\_annualized\_portfolio\_returns(

118 data, predictions, test\_winner\_num

119 )

120

121 with open(winner\_filename, "w", newline="") as file:

122 writer = csv.writer(file)

123 writer.writerows(winners)

124

125 with open(loser\_filename, "w", newline="") as file:

126 writer = csv.writer(file)

127 writer.writerows(losers)

128

129

130 with open(daily\_port\_return\_filename, mode="w", newline="") as file:

131 writer = csv.writer(file)

132 writer.writerow(["daily\_portfolio\_return"])

133 for x in daily\_port\_returns:

134 writer.writerow([x])

135

136 print("[SAVE] winners/losers & daily returns for k =", test\_winner\_num)

**Annexe E: Commandes**

RUN PIPELINE — Euro STOXX 50 & NASDAQ-100

# PRÉREQUIS — fichiers d'entrée

data\eurostoxx50\_prices\_values\_only.csv

data\nasdaq100\_prices\_values\_only.csv

**# ÉTAPE 1 — CAUSAL DISCOVERY**

# EURO STOXX 50

python causal\_discovery\_varlingam.py data\eurostoxx50\_prices\_values\_only.csv 1 eurostoxx50 VARLiNGAM

python causal\_discovery\_varlingam.py data\eurostoxx50\_prices\_values\_only.csv 2 eurostoxx50 VARLiNGAM

# NASDAQ-100

python causal\_discovery\_varlingam.py data\nasdaq100\_prices\_values\_only.csv 1 nasdaq100 VARLiNGAM

python causal\_discovery\_varlingam.py data\nasdaq100\_prices\_values\_only.csv 2 nasdaq100 VARLiNGAM

**# ÉTAPE 2 — PREDICT**

# EURO STOXX 50 LAG = 1

python predict.py "data\eurostoxx50\_prices\_values\_only.csv" "causal\_graphs\graphs\eurostoxx50\_graph\_VARLiNGAM\_lag\_1.txt" 1 eurostoxx50 VARLiNGAM --train\_frac 0.80

# EURO STOXX 50 LAG = 2

python predict.py "data\eurostoxx50\_prices\_values\_only.csv" "causal\_graphs\graphs\eurostoxx50\_graph\_VARLiNGAM\_lag\_2.txt" 2 eurostoxx50 VARLiNGAM --train\_frac 0.80

# NASDAQ-100 LAG = 1

python predict.py "data\nasdaq100\_prices\_values\_only.csv" "causal\_graphs\graphs\nasdaq100\_graph\_VARLiNGAM\_lag\_1.txt" 1 nasdaq100 VARLiNGAM --train\_frac 0.80

# NASDAQ-100 LAG = 2

python predict.py "data\nasdaq100\_prices\_values\_only.csv" "causal\_graphs\graphs\nasdaq100\_graph\_VARLiNGAM\_lag\_2.txt" 2 nasdaq100 VARLiNGAM --train\_frac 0.80

**# ÉTAPE 3 — BACKTEST**

# EURO STOXX 50 LAG = 1 k = 3

python backtest.py "data\eurostoxx50\_prices\_values\_only.csv" "predictions\eurostoxx50\_predictions\_VARLiNGAM\_lag\_1.csv" 1 eurostoxx50 VARLiNGAM 3

# EURO STOXX 50 LAG = 2 k = 3

python backtest.py "data\eurostoxx50\_prices\_values\_only.csv" "predictions\eurostoxx50\_predictions\_VARLiNGAM\_lag\_2.csv" 2 eurostoxx50 VARLiNGAM 3

# NASDAQ-100 LAG = 1 k = 6

python backtest.py "data\nasdaq100\_prices\_values\_only.csv" "predictions\nasdaq100\_predictions\_VARLiNGAM\_lag\_1.csv" 1 nasdaq100 VARLiNGAM 6

# NASDAQ-100 LAG = 2 k = 6

python backtest.py "data\nasdaq100\_prices\_values\_only.csv" "predictions\nasdaq100\_predictions\_VARLiNGAM\_lag\_2.csv" 2 nasdaq100 VARLiNGAM 6

**# ÉTAPE 4 — FIGURES PROPRES AR vs k**

python make\_clean\_graph.py --root backtesting --outdir clean\_graphs --markets eurostoxx50 nasdaq100 --lags 1 2