UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

DO STOCK PRICES BEHAVE LIKE A MARTINGALE, OR IS A FUNDAMENTAL PRESENT-VALUE MODEL A MORE APPROPRIATE DESCRIPTION OF THEIR DYNAMICS AND PREDICTIVE STRUCTURE?

CODE FILE

Authors:

Elia Landini, Lorenzo Alessandro Uberti Bona Blotto, Simon Mitrofanoff

Full Code and Datasets Repository Available at:
https://github.com/EliaLand/Martingale_FVM_AssetPricing

DATA RETRIEVAL

1) REQUIREMENTS SETUP

In [101…
```
# !pip install -r requirements.txt
```

In [102…
```
import warnings
warnings.filterwarnings("ignore")
import os
import pandas as pd
import numpy as np
import yfinance as yf
```

2) MODULES IMPORT

In [103…
```
from YFINANCE_module import fetch_YFINANCE
from FRED_module import fetch_FRED
```

3) DATA FETCHING

3.1) MARKET COMPOSITE INDEX

In [104…
```
# NASDAQ100 - ^NDX (hourly, price (USD), 2025-01-01, 2025-11-15)

ticker = "^NDX"
start = "2025-01-01"
end = "2025-11-15"
frequency = "1h"

df = fetch_YFINANCE(ticker, start, end, frequency)

# Uploading in aggregate_df
df.to_csv("raw_df/nasdaq_hourly_df.csv", index=False)
df
```

Out[104…

Price	Datetime	Close	High	Low	Open
0	2025-01-02 14:30:00+00:00	21086.878906	21236.593750	20982.998047	21117.835938
1	2025-01-02 15:30:00+00:00	21092.003906	21122.890625	20999.927734	21087.419922
2	2025-01-02 16:30:00+00:00	20960.873047	21152.394531	20955.894531	21091.203125
3	2025-01-02 17:30:00+00:00	20866.994141	20946.810547	20801.910156	20946.810547
4	2025-01-02 18:30:00+00:00	20914.117188	20923.443359	20801.388672	20860.451172
...	...	...	...	...	...
1524	2025-11-14 16:30:00+00:00	25169.808594	25198.933594	25090.419922	25151.562500
1525	2025-11-14 17:30:00+00:00	25073.060547	25189.691406	25033.886719	25169.666016
1526	2025-11-14 18:30:00+00:00	25110.427734	25125.421875	25001.224609	25068.011719
1527	2025-11-14 19:30:00+00:00	25052.566406	25157.839844	25035.353516	25113.197266
1528	2025-11-14 20:30:00+00:00	25010.021484	25068.583984	24985.279297	25042.021484

1529 rows × 6 columns

In [105…

```
# NASDAQ100 - ^NDX (daily, price (USD), 2002-01-01, 2025-11-15)

ticker = "^NDX"
start = "2002-01-01"
end = "2025-11-15"
frequency = "1d"

df = fetch_YFINANCE(ticker, start, end, frequency)

# Uploading in aggregate_df
df.to_csv("raw_df/nasdaq_daily_df.csv", index=False)
df
```

Out[105…

Price	Date	Close	High	Low	Open	Vo
0	2002-01-02	1610.390015	1610.410034	1565.079956	1590.709961	151767
1	2002-01-03	1666.660034	1667.189941	1618.300049	1618.300049	220963
2	2002-01-04	1675.030029	1698.459961	1645.319946	1685.540039	220561
3	2002-01-07	1649.829956	1694.270020	1647.560059	1691.359985	212111
4	2002-01-08	1666.579956	1676.689941	1641.170044	1651.420044	187367
…	…	…	…	…	…	…
6003	2025-11-10	25611.740234	25655.509766	25354.439453	25418.169922	929732
6004	2025-11-11	25533.490234	25588.259766	25380.859375	25500.929688	775681
6005	2025-11-12	25517.330078	25663.169922	25389.730469	25662.189453	844992
6006	2025-11-13	24993.460938	25396.009766	24908.599609	25386.949219	1135141
6007	2025-11-14	25008.240234	25198.929688	24534.900391	24658.550781	1084154

6008 rows × 6 columns

In [106…

```
# NASDAQ100 - ^NDX (weekly, price (USD), 2002-01-01, 2025-11-15)

ticker = "^NDX"
start = "2002-01-01"
end = "2025-11-15"
frequency = "1wk"

df = fetch_YFINANCE(ticker, start, end, frequency)

# Uploading in aggregate_df
df.to_csv("raw_df/nasdaq_weekly_df.csv", index=False)
df
```

Out[106…

Price	Date	Close	High	Low	Open	Vo
0	2002-01-01	1649.829956	1698.459961	1565.079956	1590.709961	805402
1	2002-01-08	1603.760010	1710.229980	1595.040039	1651.420044	938394
2	2002-01-15	1548.219971	1628.000000	1537.989990	1607.010010	717854
3	2002-01-22	1564.859985	1586.339966	1500.890015	1567.589966	873360
4	2002-01-29	1479.170044	1582.650024	1471.520020	1570.699951	923491
…	…	…	…	…	…	
1241	2025-10-14	25141.019531	25183.810547	24256.279297	24442.929688	5148796
1242	2025-10-21	25821.550781	25839.029297	24652.109375	25139.800781	5663633
1243	2025-10-28	25972.939453	26182.099609	25732.390625	25932.640625	5256395
1244	2025-11-04	25611.740234	25762.230469	24603.779297	25580.339844	5077895
1245	2025-11-11	25008.240234	25663.169922	24534.900391	25500.929688	3839968

1246 rows × 6 columns

In [107…

```
# NASDAQ100 - ^NDX (monthly, price (USD), 2002-01-01, 2025-11-15)

ticker = "^NDX"
start = "2002-01-01"
end = "2025-11-15"
frequency = "1mo"

df = fetch_YFINANCE(ticker, start, end, frequency)

# Uploading in aggregate_df
df.to_csv("raw_df/nasdaq_monthly_df.csv", index=False)
df
```

Out[107…

Price	Date	Close	High	Low	Open	V
0	2002-01-01	1550.170044	1710.229980	1481.459961	1590.709961	390959
1	2002-02-01	1359.219971	1561.239990	1329.930054	1544.420044	340226
2	2002-03-01	1452.810059	1573.420044	1370.319946	1373.150024	346193
3	2002-04-01	1277.069946	1481.739990	1228.939941	1440.380005	386485
4	2002-05-01	1208.339966	1350.540039	1142.250000	1273.630005	398028
...	...	...	...	...	...	
282	2025-07-01	23218.119141	23589.369141	22388.089844	22594.480469	2099369
283	2025-08-01	23415.419922	23969.279297	22673.880859	22941.060547	1763090
284	2025-09-01	24679.990234	24781.730469	22977.880859	23020.470703	1929190
285	2025-10-01	25858.130859	26182.099609	24207.150391	24539.250000	2427364
286	2025-11-01	24054.380859	26132.869141	24021.439453	26114.109375	1368726

287 rows × 6 columns

3.2) 1-MONTH MATURITY US-TREASURY

In [108…

```
# Market Yield on U.S. Treasury Securities at 1-Month Constant Maturity,
# https://fred.stlouisfed.org/series/DGS1MO

ticker = "DGS1MO"
start = "2002-01-01"
end = "2025-11-15"

df = fetch_FRED(ticker, start, end)
df = df.rename(columns=
        {"date": "Date",
         "DGS1MO": "1-month Yield - US Treasury Securities"
})

# Full daily date range
df = df.sort_values("Date").set_index("Date")
full_idx = pd.date_range(start=df.index.min(), end=df.index.max(), freq="
df = df.reindex(full_idx)
```

```
# Continuous Values Correction
df["1-month Yield - US Treasury Securities"] = df["1-month Yield - US Tre
df = df.reset_index().rename(columns={"index": "Date"})

# Uploading in aggregate_df
df.to_csv("raw_df/risk_free_daily_df.csv", index=False)
df
```

Out[108…

	Date	1-month Yield - US Treasury Securities
0	2002-01-01	NaN
1	2002-01-02	1.73
2	2002-01-03	1.73
3	2002-01-04	1.72
4	2002-01-05	1.72
...	...	...
8714	2025-11-10	4.04
8715	2025-11-11	4.04
8716	2025-11-12	4.03
8717	2025-11-13	4.05
8718	2025-11-14	4.04

8719 rows × 2 columns

In [109…

```
# Market Yield on U.S. Treasury Securities at 1-Month Constant Maturity,
# https://fred.stlouisfed.org/series/DGS1MO

ticker = "DGS1MO"
start = "2002-01-01"
end = "2025-11-15"

df = fetch_FRED(ticker, start, end)
df = df.rename(columns=
        {"date": "Date",
         "DGS1MO": "1-month Yield - US Treasury Securities"
})

# Full daily date range
df = df.sort_values("Date").set_index("Date")
full_idx = pd.date_range(start=df.index.min(), end=df.index.max(), freq="
df = df.reindex(full_idx)

# Continuous Values Correction
df["1-month Yield - US Treasury Securities"] = df["1-month Yield - US Tre
```

```
# Aggregation (dimension from daily to weekly)
df = df.iloc[::7].copy()
df = df.reset_index().rename(columns={"index": "Date"})

# Uploading in aggregate_df
df.to_csv("raw_df/risk_free_weekly_df.csv", index=False)
df
```

Out[109…

	Date	1-month Yield - US Treasury Securities
0	2002-01-01	NaN
1	2002-01-08	1.70
2	2002-01-15	1.64
3	2002-01-22	1.67
4	2002-01-29	1.72
...	...	...
1241	2025-10-14	4.19
1242	2025-10-21	4.12
1243	2025-10-28	4.07
1244	2025-11-04	4.02
1245	2025-11-11	4.04

1246 rows × 2 columns

In [110…

```
# Market Yield on U.S. Treasury Securities at 1-Month Constant Maturity,
# https://fred.stlouisfed.org/series/DGS1MO

ticker = "DGS1MO"
start = "2002-01-01"
end = "2025-11-15"

df = fetch_FRED(ticker, start, end)
df = df.rename(columns=
        {"date": "Date",
         "DGS1MO": "1-month Yield - US Treasury Securities"
})

# Full daily date range
df = df.sort_values("Date").set_index("Date")
full_idx = pd.date_range(start=df.index.min(), end=df.index.max(), freq="
df = df.reindex(full_idx)

# Continuous Values Correction
df["1-month Yield - US Treasury Securities"] = df["1-month Yield - US Tre

# Aggregation (dimension from daily to monthly, we cannot use a loop like
```

```
# (!!!) We take only the value observed at the opening of the month (YYYY
df = df.reset_index().rename(columns={"index": "Date"})
df = df[df["Date"].dt.day == 1]
df = df.reset_index(drop=True)

# Uploading in aggregate_df
df.to_csv("raw_df/risk_free_monthly_df.csv", index=False)
df
```

Out[110…

	Date	1-month Yield - US Treasury Securities
0	2002-01-01	NaN
1	2002-02-01	1.69
2	2002-03-01	1.78
3	2002-04-01	1.79
4	2002-05-01	1.76
...	...	...
282	2025-07-01	4.32
283	2025-08-01	4.49
284	2025-09-01	4.41
285	2025-10-01	4.17
286	2025-11-01	4.06

287 rows × 2 columns

3.3) SYNTHETIC NASDAQ-100 INDEX WEIGHTS (DIVIDEND ISSUERS)

In [111…

```
# Local Nasdaq-100 constituents shares TOTAL
# https://www.slickcharts.com/nasdaq100

nasdaq_weights = [
["NVDA", 13.90], ["AAPL", 12.28], ["MSFT", 11.09], ["AMZN", 7.27], ["GOOG
["GOOG", 5.33], ["AVGO", 5.23], ["META", 4.60], ["TSLA", 4.17], ["NFLX",
["COST", 1.23], ["ASML", 1.20], ["PLTR", 1.17], ["AMD", 1.07], ["CSCO", 0
["AZN", 0.85], ["MU", 0.73], ["TMUS", 0.73], ["PEP", 0.61], ["ISRG", 0.61
["SHOP", 0.60], ["LIN", 0.60], ["APP", 0.57], ["AMGN", 0.56], ["LRCX", 0.
["AMAT", 0.56], ["INTU", 0.56], ["QCOM", 0.54], ["INTC", 0.51], ["PDD", 0
["GILD", 0.48], ["KLAC", 0.47], ["BKNG", 0.46], ["ARM", 0.45], ["TXN", 0.
["ADBE", 0.41], ["CRWD", 0.40], ["PANW", 0.40], ["HON", 0.37], ["ADI", 0.
["CEG", 0.35], ["VRTX", 0.33], ["ADP", 0.31], ["MELI", 0.30], ["CMCSA", 0
["SBUX", 0.30], ["CDNS", 0.26], ["ORLY", 0.26], ["DASH", 0.26], ["REGN",
["MAR", 0.24], ["CTAS", 0.23], ["SNPS", 0.22], ["MDLZ", 0.22], ["MNST", 0
["MRVL", 0.21], ["ABNB", 0.21], ["AEP", 0.20], ["CSX", 0.19], ["ADSK", 0.
["TRI", 0.18], ["FTNT", 0.18], ["WDAY", 0.18], ["WBD", 0.18], ["DDOG", 0.
```

```
["IDXX", 0.17], ["PYPL", 0.17], ["ROST", 0.16], ["PCAR", 0.16], ["MSTR",
["EA", 0.16], ["BKR", 0.15], ["ROP", 0.15], ["NXPI", 0.15], ["XEL", 0.15]
["EXC", 0.14], ["ZS", 0.14], ["FAST", 0.14], ["TTWO", 0.14], ["FANG", 0.1
["AXON", 0.13], ["CCEP", 0.12], ["CPRT", 0.12], ["PAYX", 0.12], ["TEAM",
["KDP", 0.11], ["CTSH", 0.11], ["GEHC", 0.10], ["VRSK", 0.10], ["KHC", 0.
["CSGP", 0.09], ["MCHP", 0.08], ["ODFL", 0.08], ["CHTR", 0.08], ["BIIB",
["DXCM", 0.07], ["LULU", 0.06], ["TTD", 0.06], ["ON", 0.06], ["GFS", 0.06
["CDW", 0.06]
]

weights_df = pd.DataFrame([{ticker: weight for ticker, weight in nasdaq_w

# (!!!) Looks horrible but it works, we consider only dividends-issuing f
weights_df = weights_df[["NVDA", "AAPL", "MSFT", "GOOGL", "AVGO", "GOOG",
"AZN", "MU", "TMUS", "PEP", "LRCX", "LIN", "QCOM", "INTC", "INTU", "AMAT"
"BKNG", "AMGN", "KLAC", "GILD", "TXN", "HON", "MELI", "CEG", "ADI", "ADP"
"CMCSA", "SBUX", "MDLZ", "CTAS", "MAR", "TRI", "CSX", "AEP", "NXPI", "ROS
"PCAR", "EA", "ROP", "XEL", "BKR", "FAST", "EXC", "PAYX", "FANG", "CCEP",
"KDP", "CTSH", "GEHC", "MCHP", "VRSK", "ODFL", "KHC", "CDW"]]
weights_df = weights_df/100
weights_df.to_csv("raw_df/nasdaq_weights_df.csv", index=False)

weights_df
```

Out[111…

	NVDA	AAPL	MSFT	GOOGL	AVGO	GOOG	META	ASML	COST	CSCO	..
0	0.139	0.1228	0.1109	0.057	0.0523	0.0533	0.046	0.012	0.0123	0.0094	..

1 rows × 58 columns

In [112…

```
# Synthetic Nasdaq-100 constituents shares
# https://www.slickcharts.com/nasdaq100

synthetic_weights_df = weights_df/weights_df.sum(axis=1).iloc[0]

synthetic_weights_df.to_csv("raw_df/synthetic_weights_df.csv", index=Fals
synthetic_weights_df
```

Out[112…

	NVDA	AAPL	MSFT	GOOGL	AVGO	GOOG	META	ASML
0	0.182511	0.161239	0.145614	0.074842	0.068671	0.069984	0.060399	0.015756

1 rows × 58 columns

3.4) SYNTHETIC NASDAQ-100 INDEX DIVIDENDS

In [113…

```
# Local Nasdaq-100 constituents dividends (daily, USD, 2002-01-01, 2025-1

constituents_path = "raw_df/constituents-nasdaq100.csv"
nasdaq_list_df = pd.read_csv(constituents_path)
```

```
tickers = nasdaq_list_df["Symbol"].tolist()
start = "2002-01-01"
end = "2025-11-15"

div_df = pd.DataFrame()

for ticker in tickers:
    try:
# (!!!) Strange set-up for history
        df = yf.Ticker(ticker).dividends

# (!!!) We need this cuz we can filter out those firms that issue no divi
        if df is None or df.empty:
            continue
#_____________________________________________
# Merging
        df = df.to_frame(name=ticker)
        div_df = pd.merge(div_df, df, how="outer", left_index=True, right
#_____________________________________________

    except Exception as e:
# (!!!) This is not for those companys issuing no dividends, but just in
        print(f"Error fetching {ticker}: {e}")

# Continuous Values Correction
# (!!!) No fill
div_df = div_df.sort_index()
full_idx = pd.date_range(start=div_df.index.min(), end=div_df.index.max()
div_df = div_df.reindex(full_idx)

# Filter by date
div_df = div_df.loc[start:end]
div_df = div_df.reset_index().rename(columns={"index": "Date"})

div_df = div_df.drop(columns=["ADBE", "MRVL", "REGN", "MNST", "PYPL", "AD
div_df.to_csv("raw_df/raw_div_daily_df.csv", index=False)
div_df
```

Out[113…

	Date	NVDA	AAPL	MSFT	GOOGL	AVGO	GOOG	META	ASML	COST
0	2002-01-01 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2002-01-02 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2002-01-03 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2002-01-04 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2002-01-05 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
…	…	…	…	…	…	…	…	…	…	…
8715	2025-11-11 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8716	2025-11-12 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8717	2025-11-13 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8718	2025-11-14 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8719	2025-11-15 00:00:00-05:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

8720 rows × 59 columns

In [114…

```
# Synthetic NASDAQ-100 index dividends (daily, USD, 2002-01-01, 2025-11-1
```

```
synthetic_div_df = pd.DataFrame()
synthetic_div_df = div_df.copy()

ticker_cols = [c for c in synthetic_div_df.columns if c != "Date"]

# Multiply weights columns for each ticker's column over time, and sum pe
w = synthetic_weights_df.loc[:, ticker_cols].iloc[0]
synthetic_div_df["Synthetic Index Dividend"] = (
    synthetic_div_df[ticker_cols].mul(w, axis=1).sum(axis=1)
)

synthetic_div_df = synthetic_div_df[["Date", "Synthetic Index Dividend"]]
synthetic_div_df.to_csv("raw_df/synthetic_div_daily_df.csv", index=False)
synthetic_div_df
```

Out[114…

	Date	Synthetic Index Dividend
0	2002-01-01 00:00:00-05:00	0.000000
1	2002-01-02 00:00:00-05:00	0.000000
2	2002-01-03 00:00:00-05:00	0.000000
3	2002-01-04 00:00:00-05:00	0.000000
4	2002-01-05 00:00:00-05:00	0.000000
…	…	…
8715	2025-11-11 00:00:00-05:00	0.000000
8716	2025-11-12 00:00:00-05:00	0.000693
8717	2025-11-13 00:00:00-05:00	0.001707
8718	2025-11-14 00:00:00-05:00	0.011836
8719	2025-11-15 00:00:00-05:00	0.000000

8720 rows × 2 columns

In [115…

```
# Synthetic NASDAQ-100 index dividends (weekly, USD, 2002-01-01, 2025-11-
df = synthetic_div_df.copy()

# Aggregation (dimension from daily to weekly)
# (!!!) Watch out! we need to consider the sum of each dividend payout wi
df = df.sort_values("Date").reset_index(drop=True)
df["week_block"] = df.index // 7

df = (
    df.groupby("week_block")
      .agg({
          "Date": "first",
          **{col: "sum" for col in df.select_dtypes("number").columns}
      })
      .drop(columns=["week_block"])
```

```
)

df = df.reset_index(drop=True)
# Uploading in aggregate_df
df.to_csv("raw_df/synthetic_div_weekly_df.csv", index=False)
df
```

Out[115…

	Date	Synthetic Index Dividend
0	2002-01-01 00:00:00-05:00	0.000000
1	2002-01-08 00:00:00-05:00	0.000082
2	2002-01-15 00:00:00-05:00	0.000000
3	2002-01-22 00:00:00-05:00	0.000000
4	2002-01-29 00:00:00-05:00	0.000300
…	…	…
1241	2025-10-14 00:00:00-04:00	0.000000
1242	2025-10-21 00:00:00-04:00	0.000046
1243	2025-10-28 00:00:00-04:00	0.058973
1244	2025-11-04 00:00:00-05:00	0.047307
1245	2025-11-11 00:00:00-05:00	0.014236

1246 rows × 2 columns

In [116…

```
# Synthetic NASDAQ-100 index dividends (monthly, USD, 2002-01-01, 2025-11
df = synthetic_div_df.copy()

# Aggregation (dimension from daily to weekly)
# (!!!) Again different cuz we need the calendar month and not every 30 d
# (!!!) Watch out! we need to consider the sum of each dividend payout wi
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date")

df = (
    df.groupby(pd.Grouper(key="Date", freq="M"))
      .agg({
          "Date": "first",
          **{col: "sum" for col in df.select_dtypes("number").columns}
      })
      .reset_index(drop=True)
)
# Uploading in aggregate_df
df.to_csv("raw_df/synthetic_div_monthly_df.csv", index=False)
df
```

Out[116…

	Date	Synthetic Index Dividend
0	2002-01-01 00:00:00-05:00	0.000382
1	2002-02-01 00:00:00-05:00	0.006262
2	2002-03-01 00:00:00-05:00	0.002720
3	2002-04-01 00:00:00-05:00	0.001222
4	2002-05-01 00:00:00-04:00	0.003142
...	...	...
282	2025-07-01 00:00:00-04:00	0.055846
283	2025-08-01 00:00:00-04:00	0.253449
284	2025-09-01 00:00:00-04:00	0.216313
285	2025-10-01 00:00:00-04:00	0.076930
286	2025-11-01 00:00:00-04:00	0.061543

287 rows × 2 columns

3.5) SYNTHETIC NASDAQ-100 PORTFOLIO PRICE

In [117…

```
# Local Nasdaq-100 constituents stock price (daily, USD, 2002-01-01, 2025

constituents_path = "raw_df/constituents-nasdaq100.csv"
nasdaq_list_df = pd.read_csv(constituents_path)

tickers = nasdaq_list_df["Symbol"].tolist()
start = "2002-01-01"
end = "2025-11-15"

price_df = pd.DataFrame()

for ticker in tickers:
    try:
# (!!!) Strange set-up for history
        df = yf.Ticker(ticker).history(period="max")

# (!!!) We need this cuz we can filter out those firms that issue no divi
        if df is None or df.empty:
            continue
#_____________________________________________
# Merging
        df = df["Close"].rename(ticker)
        price_df = pd.concat([price_df, df], axis=1)
#_____________________________________________

    except Exception as e:
# (!!!) This is not for those companys issuing no dividends, but just in
```

```
        print(f"Error fetching {ticker}: {e}")

# Continuous Values Correction
# (!!!) Here we fill
price_df = price_df.sort_index()
full_idx = pd.date_range(start=price_df.index.min(), end=price_df.index.m
price_df = price_df.reindex(full_idx)
price_df= price_df.ffill()

# Filter by date
price_df = price_df.loc[start:end]
price_df = price_df.reset_index().rename(columns={"index": "Date"})

# (!!!) Looks horrible but it works, we consider only dividends-issuing f
price_df = price_df[["Date", "NVDA", "AAPL", "MSFT", "GOOGL", "AVGO", "GO
"AZN", "MU", "TMUS", "PEP", "LRCX", "LIN", "QCOM", "INTC", "INTU", "AMAT"
"BKNG", "AMGN", "KLAC", "GILD", "TXN", "HON", "MELI", "CEG", "ADI", "ADP"
"CMCSA", "SBUX", "MDLZ", "CTAS", "MAR", "TRI", "CSX", "AEP", "NXPI", "ROS
"PCAR", "EA", "ROP", "XEL", "BKR", "FAST", "EXC", "PAYX", "FANG", "CCEP",
"KDP", "CTSH", "GEHC", "MCHP", "VRSK", "ODFL", "KHC", "CDW"]]
price_df.to_csv("raw_df/raw_price_daily_df.csv", index=False)
price_df
```

Out[117…

	Date	NVDA	AAPL	MSFT	GOOGL	AVGO	
0	2002-01-01 00:00:00-05:00	0.511164	0.328402	20.234600	NaN	NaN	
1	2002-01-02 00:00:00-05:00	0.514220	0.349395	20.475899	NaN	NaN	
2	2002-01-03 00:00:00-05:00	0.547915	0.353594	21.144785	NaN	NaN	
3	2002-01-04 00:00:00-05:00	0.531947	0.355244	21.043991	NaN	NaN	
4	2002-01-05 00:00:00-05:00	0.531947	0.355244	21.043991	NaN	NaN	
…	…	…	…	…	…	…	
8715	2025-11-11 00:00:00-05:00	193.160004	275.250000	507.729706	291.309998	351.959991	29
8716	2025-11-12 00:00:00-05:00	193.800003	273.470001	510.185150	286.709991	355.220001	28
8717	2025-11-13 00:00:00-05:00	186.860001	272.950012	502.349792	278.570007	339.980011	27
8718	2025-11-14 00:00:00-05:00	190.169998	272.410004	509.226898	276.410004	342.459991	27
8719	2025-11-15 00:00:00-05:00	190.169998	272.410004	509.226898	276.410004	342.459991	27

8720 rows × 59 columns

In [118…

```
# Synthetic NASDAQ-100 index price (daily, USD, 2002-01-01, 2025-11-15)
```

```
synthetic_price_df = pd.DataFrame()
synthetic_price_df = price_df.copy()

ticker_cols = [c for c in synthetic_price_df.columns if c != "Date"]

# Multiply weights columns for each ticker's column over time, and sum pe
w = synthetic_weights_df.loc[:, ticker_cols].iloc[0]
synthetic_price_df["Synthetic Index Close Price"] = (
    synthetic_price_df[ticker_cols].mul(w, axis=1).sum(axis=1)
)

synthetic_price_df = synthetic_price_df[["Date", "Synthetic Index Close P
synthetic_price_df.to_csv("raw_df/synthetic_price_daily_df.csv", index=Fa
synthetic_price_df
```

Out[118…

	Date	Synthetic Index Close Price
0	2002-01-01 00:00:00-05:00	6.741954
1	2002-01-02 00:00:00-05:00	6.846058
2	2002-01-03 00:00:00-05:00	7.054649
3	2002-01-04 00:00:00-05:00	7.039679
4	2002-01-05 00:00:00-05:00	7.039679
…	…	…
8715	2025-11-11 00:00:00-05:00	370.503619
8716	2025-11-12 00:00:00-05:00	370.481964
8717	2025-11-13 00:00:00-05:00	364.229183
8718	2025-11-14 00:00:00-05:00	365.196053
8719	2025-11-15 00:00:00-05:00	365.196053

8720 rows × 2 columns

In [119…

```
# Synthetic NASDAQ-100 index price (weekly, USD, 2002-01-01, 2025-11-15)
df = synthetic_price_df.copy()

# Aggregation (dimension from daily to weekly)
df = df.iloc[::7].copy()
df = df.reset_index(drop=True)

# Uploading in aggregate_df
df.to_csv("raw_df/synthetic_price_weekly_df.csv", index=False)
df
```

Out[119…

	Date	Synthetic Index Close Price
0	2002-01-01 00:00:00-05:00	6.741954
1	2002-01-08 00:00:00-05:00	7.021340
2	2002-01-15 00:00:00-05:00	6.956946
3	2002-01-22 00:00:00-05:00	6.548206
4	2002-01-29 00:00:00-05:00	6.522922
...	...	...
1241	2025-10-14 00:00:00-04:00	362.299566
1242	2025-10-21 00:00:00-04:00	369.404837
1243	2025-10-28 00:00:00-04:00	383.372298
1244	2025-11-04 00:00:00-05:00	369.728215
1245	2025-11-11 00:00:00-05:00	370.503619

1246 rows × 2 columns

In [120…

```python
# Synthetic NASDAQ-100 index price (monthly, USD, 2002-01-01, 2025-11-15)
df = synthetic_price_df.copy()

# Aggregation (dimension from daily to monthly)
df = df[df["Date"].dt.day == 1]
df = df.reset_index(drop=True)

# Uploading in aggregate_df
df.to_csv("raw_df/synthetic_price_monthly_df.csv", index=False)
df
```

Out[120…

	Date	Synthetic Index Close Price
0	2002-01-01 00:00:00-05:00	6.741954
1	2002-02-01 00:00:00-05:00	6.708014
2	2002-03-01 00:00:00-05:00	6.614498
3	2002-04-01 00:00:00-05:00	6.714867
4	2002-05-01 00:00:00-04:00	6.022081
...	...	...
282	2025-07-01 00:00:00-04:00	333.452353
283	2025-08-01 00:00:00-04:00	340.753141
284	2025-09-01 00:00:00-04:00	348.811723
285	2025-10-01 00:00:00-04:00	366.551887
286	2025-11-01 00:00:00-04:00	375.118033

287 rows × 2 columns

1. Requirements Setup

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf

import statsmodels.api as sm
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import binomtest as binom_test
#
from datetime import datetime, timedelta
from pathlib import Path
from scipy.stats import t
import os
from scipy import stats
from statsmodels.tsa.stattools import adfuller
import os
```

Getting Data from teh data set - Dividends side and Risk free rate

In [17]:
```
Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")

# ================================
# Generic loader for dividends & risk-free (robust column detection)
# ================================
def load_series_with_logs(csv_filename, value_col, date_cols=('Date','Dat
    """
    Robust loader that:
      - parses 'Date' or 'Datetime' as index (if present)
      - detects the value column flexibly (accepts exact, list of names,
      - coerces to numeric
      - computes Log_Value and Log_Returns while avoiding log(0)/inf
    """
    path = os.path.join(Data_dir, csv_filename)
    df = pd.read_csv(path)

    # ---- Handle date parsing ----
    for col in date_cols:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    if 'Date' in df.columns and not df['Date'].isna().all():
        df.set_index('Date', inplace=True)
    elif 'Datetime' in df.columns and not df['Datetime'].isna().all():
        df.set_index('Datetime', inplace=True)
    else:
        raise ValueError(f"{csv_filename} missing Date/Datetime. Found {l
```

```
    # ---- Flexible detection of value column ----
    # allow value_col to be a single name or an iterable of candidates
    if isinstance(value_col, (list, tuple, set)):
        candidates = list(value_col)
    else:
        candidates = [value_col]

    def _norm(s):
        return str(s).lower().replace('–', '-').replace('—', '-').strip()

    cols_norm = { _norm(c): c for c in df.columns }

    chosen_col = None
    # try exact (normalized) matches first
    for cand in candidates:
        nc = _norm(cand)
        if nc in cols_norm:
            chosen_col = cols_norm[nc]
            break

    # fallback: look for obvious keywords if exact not found
    if chosen_col is None:
        keywords = ['yield', 'treasury', '1-month', '1 month', '1m', 'rat
        for col in df.columns:
            lc = str(col).lower()
            if any(kw in lc for kw in keywords):
                chosen_col = col
                break

    # last resort: pick the single numeric column (if only one exists)
    if chosen_col is None:
        numeric_cols = df.select_dtypes(include=[np.number]).columns.toli
        if len(numeric_cols) == 1:
            chosen_col = numeric_cols[0]

    if chosen_col is None:
        raise KeyError(
            f"Could not find value column for '{value_col}' in {csv_filen
            f"Available columns: {list(df.columns)}"
        )

    if chosen_col != value_col:
        # small informative message (won't break notebooks); change to pr
        print(f"[load_series_with_logs] Using column '{chosen_col}' for '

    # ---- Ensure numeric ----
    df[chosen_col] = pd.to_numeric(df[chosen_col], errors='coerce')

    # ---- Compute logs and returns avoiding log(0) and infs ----
    series = df[chosen_col]
    series_nonzero = series.replace(0, np.nan)

    df['Log_Value'] = np.log(series_nonzero)  # log of level, NaN if zero
```

```
    # compute log-returns safely: if previous is zero/missing, result is
    prev = series_nonzero.shift(1)
    with np.errstate(divide='ignore', invalid='ignore'):
        lr = np.log(series_nonzero / prev)

    # replace infinities with NaN
    lr = pd.Series(lr, index=df.index).replace([np.inf, -np.inf], np.nan)
    df['Log_Returns'] = lr

    return df


# ================================
# Dividend loaders
# ================================
synthetic_div_daily_df   = load_series_with_logs(
    "synthetic_div_daily_df.csv",
    "Synthetic Index Dividend"
)
synthetic_div_weekly_df  = load_series_with_logs(
    "synthetic_div_weekly_df.csv",
    "Synthetic Index Dividend"
)
synthetic_div_monthly_df = load_series_with_logs(
    "synthetic_div_monthly_df.csv",
    "Synthetic Index Dividend"
)

# Extract ready-to-use series
synthetic_div_daily_log_div      = synthetic_div_daily_df['Log_Returns'].
synthetic_div_weekly_log_div     = synthetic_div_weekly_df['Log_Returns']
synthetic_div_monthly_log_div    = synthetic_div_monthly_df['Log_Returns'

synthetic_div_daily_log_levels   = synthetic_div_daily_df['Log_Value'].dr
synthetic_div_weekly_log_levels  = synthetic_div_weekly_df['Log_Value'].d
synthetic_div_monthly_log_levels = synthetic_div_monthly_df['Log_Value'].


# ================================
# Risk-free loaders
# ================================
risk_free_daily_df = load_series_with_logs(
    "risk_free_daily_df.csv",
    "1-month Yield - US Treasury Securities"
)
risk_free_weekly_df = load_series_with_logs(
    "risk_free_weekly_df.csv",
    "1-month Yield - US Treasury Securities"
)
risk_free_monthly_df = load_series_with_logs(
    "risk_free_monthly_df.csv",
    "1-month Yield - US Treasury Securities"
)
```

```
risk_free_daily_log_rf   = risk_free_daily_df['Log_Value'].dropna()
risk_free_weekly_log_rf  = risk_free_weekly_df['Log_Value'].dropna()
risk_free_monthly_log_rf = risk_free_monthly_df['Log_Value'].dropna()
```

```
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_daily_df.csv' (requested '1-month Yield - US Treasury S
ecurities').
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_weekly_df.csv' (requested '1-month Yield - US Treasury
Securities').
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_monthly_df.csv' (requested '1-month Yield - US Treasury
Securities').
```

```
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
```

2. Getting the relevant data from the Dataset - Price Side

In [3]:

```
Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")

def load_with_log_returns(csv_filename, lag=1):
    path = os.path.join(Data_dir, csv_filename)

    df = pd.read_csv(path)

    # Parse available date columns robustly
    for col in ['Date', 'Datetime']:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    # Prefer 'Date' as the index, otherwise use 'Datetime' if present
    if 'Date' in df.columns and not df['Date'].isna().all():
        df.set_index('Date', inplace=True)
```

```
    elif 'Datetime' in df.columns and not df['Datetime'].isna().all():
        df.set_index('Datetime', inplace=True)
    else:
        raise ValueError(
            f"CSV {path} must contain a 'Date' or 'Datetime' column. "
            f"Found: {list(df.columns)}"
        )

    # Detect the close-price column
    close_candidates = ['Close', 'Synthetic Index Close Price']
    price_col = None
    for c in close_candidates:
        if c in df.columns:
            price_col = c
            break

    if price_col is None:
        raise ValueError(
            f"CSV {path} must contain one of {close_candidates}. "
            f"Found: {list(df.columns)}"
        )

    # Compute log returns and log prices based on the detected price colu
    price_series = df[price_col].astype(float)

    df['Log_Returns'] = np.log(price_series / price_series.shift(lag))
    df['Log_Prices'] = np.log(price_series)

    return df

# ====== your existing indices (unchanged, still work if they use 'Close'
nasdaq_daily_df   = load_with_log_returns('nasdaq_daily_df.csv')
nasdaq_weekly_df  = load_with_log_returns('nasdaq_weekly_df.csv')
nasdaq_monthly_df = load_with_log_returns('nasdaq_monthly_df.csv')
nasdaq_hourly_df  = load_with_log_returns('nasdaq_hourly_df.csv')

nasdaq_daily_log_returns   = nasdaq_daily_df['Log_Returns'].dropna()
nasdaq_weekly_log_returns  = nasdaq_weekly_df['Log_Returns'].dropna()
nasdaq_monthly_log_returns = nasdaq_monthly_df['Log_Returns'].dropna()
nasdaq_hourly_log_returns  = nasdaq_hourly_df['Log_Returns'].dropna()

nasdaq_hourly_log_prices  = nasdaq_hourly_df['Log_Prices'].dropna()
nasdaq_daily_log_prices   = nasdaq_daily_df['Log_Prices'].dropna()
nasdaq_weekly_log_prices  = nasdaq_weekly_df['Log_Prices'].dropna()
nasdaq_monthly_log_prices = nasdaq_monthly_df['Log_Prices'].dropna()

# ====== synthetic indices (use 'Synthetic Index Close Price') ======
synthetic_price_daily_df   = load_with_log_returns('synthetic_price_daily
synthetic_price_weekly_df  = load_with_log_returns('synthetic_price_weekl
synthetic_price_monthly_df = load_with_log_returns('synthetic_price_month


synthetic_daily_log_returns   = synthetic_price_daily_df['Log_Returns'].d
synthetic_weekly_log_returns  = synthetic_price_weekly_df['Log_Returns'].
```

```
synthetic_monthly_log_returns = synthetic_price_monthly_df['Log_Returns']


synthetic_daily_log_prices   = synthetic_price_daily_df['Log_Prices'].dro
synthetic_weekly_log_prices  = synthetic_price_weekly_df['Log_Prices'].dr
synthetic_monthly_log_prices = synthetic_price_monthly_df['Log_Prices'].d
```

```
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/190536089
9.py:12: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/190536089
9.py:12: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/190536089
9.py:12: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
```

3. Testing the Martingale Properties of the Log Return

In [4]:
```
def test_mean_returns(returns):
    """
    Basically a variant of the T-test where we test if the mean return is

    Returns:
        (mean_return, std_return, se_return, t_stat, p_value_super, p_val

    p_value_super: one-sided p-value for H1: mean < 0 (supermartingale)
    p_value_sub:   one-sided p-value for H1: mean > 0 (submartingale)
    p_value_two:   two-sided p-value for H1: mean != 0
    """
    # converts it all to a big ol numpy and drops the NaNs
    r = returns.values if hasattr(returns, "values") else np.array(return
    r = np.asarray(r)
    r = r[~np.isnan(r)]

    n = len(r)
    if n < 2:
        raise ValueError("Not enough data points to perform the test.")

    mean_return = np.mean(r)
    std_return = np.std(r, ddof=1)
```

```
    se_return = std_return / np.sqrt(n)

    # handle zero-variance case --> it just a likkle safety net
    if se_return == 0:
        t_stat = 0.0
    else:
        t_stat = mean_return / se_return

    # one-sided p-values
    p_value_super = t.cdf(t_stat, df=n-1)
    p_value_sub = 1 - p_value_super
    p_value_two = 2 * min(p_value_super, p_value_sub)

    return mean_return, std_return, se_return, t_stat, p_value_super, p_v

def print_mean_return_results(returns, label="Mean Return"):
    (mean, std, se, t_stat,
     p_super, p_sub, p_two) = test_mean_returns(returns)

    n = len(returns)

    print("\n==============================")
    print(f"TEST — {label}")
    print("==============================")
    print(f"Number of obs:                    {n}")
    print(f"Mean return:                      {mean:.6e}")
    print(f"Std dev:                          {std:.6e}")
    print(f"Std error of mean:                {se:.6e}")
    print(f"t-statistic:                      {t_stat:.3f}")
    print(f"\nTwo-sided p-value (H1: mean ≠ 0): {p_two:.4g}")
    print(f"One-sided p-value (H1: mean < 0 → supermartingale): {p_super:
    print(f"One-sided p-value (H1: mean > 0 → submartingale):   {p_sub:.4

    # Interpretation
    print("\nInterpretation:")
    if p_two < 0.05:
        if mean > 0:
            print("→ Significant *positive* drift: SUBMARTINGALE behaviou
        else:
            print("→ Significant *negative* drift: SUPERMARTINGALE behavi
    else:
        print("→ Mean return not significantly different from zero: MARTI

print_mean_return_results(nasdaq_daily_log_returns,     "NASDAQ — DAILY l
print_mean_return_results(nasdaq_weekly_log_returns,    "NASDAQ — WEEKLY
print_mean_return_results(nasdaq_monthly_log_returns,   "NASDAQ — MONTHLY
print_mean_return_results(nasdaq_hourly_log_returns,    "NASDAQ — HOURLY

# Synthetic Index mean return results
print_mean_return_results(synthetic_daily_log_returns,   "SYNTHETIC — DAI
print_mean_return_results(synthetic_weekly_log_returns,  "SYNTHETIC — WEE
print_mean_return_results(synthetic_monthly_log_returns, "SYNTHETIC — MON
```

```
==============================
```

```
TEST — NASDAQ — DAILY log returns
==============================
Number of obs:                    6007
Mean return:                      4.565888e-04
Std dev:                          1.474215e-02
Std error of mean:                1.902094e-04
t-statistic:                      2.400

Two-sided p-value (H1: mean ≠ 0): 0.0164
One-sided p-value (H1: mean < 0 → supermartingale): 0.9918
One-sided p-value (H1: mean > 0 → submartingale):   0.008202

Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.

==============================
TEST — NASDAQ — WEEKLY log returns
==============================
Number of obs:                    1245
Mean return:                      2.183561e-03
Std dev:                          3.104117e-02
Std error of mean:                8.797380e-04
t-statistic:                      2.482

Two-sided p-value (H1: mean ≠ 0): 0.01319
One-sided p-value (H1: mean < 0 → supermartingale): 0.9934
One-sided p-value (H1: mean > 0 → submartingale):   0.006597

Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.

==============================
TEST — NASDAQ — MONTHLY log returns
==============================
Number of obs:                    286
Mean return:                      9.671425e-03
Std dev:                          5.577964e-02
Std error of mean:                3.298319e-03
t-statistic:                      2.932

Two-sided p-value (H1: mean ≠ 0): 0.003638
One-sided p-value (H1: mean < 0 → supermartingale): 0.9982
One-sided p-value (H1: mean > 0 → submartingale):   0.001819

Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.

==============================
TEST — NASDAQ — HOURLY log returns
==============================
Number of obs:                    1528
Mean return:                      1.116660e-04
Std dev:                          5.450903e-03
Std error of mean:                1.394462e-04
```

```
t-statistic:                      0.801

Two-sided p-value (H1: mean ≠ 0): 0.4234
One-sided p-value (H1: mean < 0 → supermartingale): 0.7883
One-sided p-value (H1: mean > 0 → submartingale):   0.2117

Interpretation:
→ Mean return not significantly different from zero: MARTINGALE-compatibl
e.

==============================
TEST — SYNTHETIC — DAILY log returns
==============================
Number of obs:                    8719
Mean return:                      4.578064e-04
Std dev:                          1.206679e-02
Std error of mean:                1.292286e-04
t-statistic:                      3.543

Two-sided p-value (H1: mean ≠ 0): 0.0003983
One-sided p-value (H1: mean < 0 → supermartingale): 0.9998
One-sided p-value (H1: mean > 0 → submartingale):   0.0001991

Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.

==============================
TEST — SYNTHETIC — WEEKLY log returns
==============================
Number of obs:                    1245
Mean return:                      3.217700e-03
Std dev:                          2.912954e-02
Std error of mean:                8.255606e-04
t-statistic:                      3.898

Two-sided p-value (H1: mean ≠ 0): 0.0001023
One-sided p-value (H1: mean < 0 → supermartingale): 0.9999
One-sided p-value (H1: mean > 0 → submartingale):   5.117e-05

Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.

==============================
TEST — SYNTHETIC — MONTHLY log returns
==============================
Number of obs:                    286
Mean return:                      1.405040e-02
Std dev:                          5.500801e-02
Std error of mean:                3.252692e-03
t-statistic:                      4.320

Two-sided p-value (H1: mean ≠ 0): 2.161e-05
One-sided p-value (H1: mean < 0 → supermartingale): 1
One-sided p-value (H1: mean > 0 → submartingale):   1.08e-05
```

```
Interpretation:
→ Significant *positive* drift: SUBMARTINGALE behaviour.
```

4.1 CW ratio for the log returns

In [5]:
```python
# Strong random Walk Hypothesis (RW1)
# Apply the Cowles and Jones (1937) test to check for independence of ret
# The test compares the frequency of sequences (two successive returns wi
# to the frequency of reversals (two successive returns with opposite sig

def cowles_jones_test(returns, name=""):
    """
    Apply the Cowles and Jones (1937) test for independence of returns.

    Parameters:
        returns : array-like or pandas Series of returns
        name    : optional label for printing

    Returns:
        dict with keys: NS, NR, CJ, z_stat, p_value
    """
    # prepare data
    r = returns.dropna().values if hasattr(returns, "dropna") else np.asa
    non_zero_mask = r != 0
    r = r[non_zero_mask]

    if len(r) < 2:
        raise ValueError("Not enough observations (after removing zeros/N

    signs = np.sign(r)
    prod = signs[:-1] * signs[1:]

    NS = np.sum(prod > 0)
    NR = np.sum(prod < 0)
    N = NS + NR

    if NR == 0 or N == 0:
        raise ValueError("Not enough variability in returns to perform th

    CJ_hat = NS / NR

    # Under RW1 with mu = 0: P(sequence) = P(reversal) = 0.5
    p_hat = NS / N
    p0 = 0.5
    se = np.sqrt(p0 * (1 - p0) / N)
    z_stat = (p_hat - p0) / se

    # use normal distribution for z-test
    from scipy.stats import norm
    p_value = 2 * (1 - norm.cdf(abs(z_stat)))

    print(f"\n===== Cowles & Jones Test ({name}) =====")
```

```
    print(f"N_S (sequences, same sign) : {NS}")
    print(f"N_R (reversals, opp. sign) : {NR}")
    print(f"Total pairs N              : {N}")
    print(f"CĴ = N_S / N_R            : {CJ_hat:.4f}")
    print(f"p̂ = N_S / (N_S+N_R)       : {p_hat:.4f}")
    print(f"z-stat (H0: p = 0.5)       : {z_stat:.4f}")
    print(f"p-value (two-sided)        : {p_value:.6f}")
    if p_value < 0.05:
        print("→ Reject RW1 (μ=0) at 5% level.")
    else:
        print("→ Cannot reject RW1 (μ=0) at 5% level.")

    return {
        "NS": int(NS),
        "NR": int(NR),
        "CJ": float(CJ_hat),
        "z_stat": float(z_stat),
        "p_value": float(p_value)
        }

# NASDAQ
cj_daily   = cowles_jones_test(nasdaq_daily_log_returns,   "NASDAQ — Dail
cj_hourly  = cowles_jones_test(nasdaq_hourly_log_returns,  "NASDAQ — Hour
cj_weekly  = cowles_jones_test(nasdaq_weekly_log_returns,  "NASDAQ — Week
cj_monthly = cowles_jones_test(nasdaq_monthly_log_returns, "NASDAQ — Mont

# Synthetic (no hourly synthetic dataset exists)
cj_syn_daily   = cowles_jones_test(synthetic_daily_log_returns,   "Synthe
cj_syn_weekly  = cowles_jones_test(synthetic_weekly_log_returns,  "Synthe
cj_syn_monthly = cowles_jones_test(synthetic_monthly_log_returns, "Synthe
```

```
===== Cowles & Jones Test (NASDAQ — Daily log returns) =====
N_S (sequences, same sign) : 2948
N_R (reversals, opp. sign) : 3056
Total pairs N              : 6004
CĴ = N_S / N_R            : 0.9647
p̂ = N_S / (N_S+N_R)       : 0.4910
z-stat (H0: p = 0.5)       : -1.3938
p-value (two-sided)        : 0.163375
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (NASDAQ — Hourly log returns) =====
N_S (sequences, same sign) : 756
N_R (reversals, opp. sign) : 771
Total pairs N              : 1527
CĴ = N_S / N_R            : 0.9805
p̂ = N_S / (N_S+N_R)       : 0.4951
z-stat (H0: p = 0.5)       : -0.3839
p-value (two-sided)        : 0.701083
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (NASDAQ — Weekly log returns) =====
N_S (sequences, same sign) : 642
N_R (reversals, opp. sign) : 602
```

```
Total pairs N              : 1244
CĴ = N_S / N_R             : 1.0664
p̂ = N_S / (N_S+N_R)        : 0.5161
z-stat (H0: p = 0.5)       : 1.1341
p-value (two-sided)        : 0.256754
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (NASDAQ — Monthly log returns) =====
N_S (sequences, same sign) : 153
N_R (reversals, opp. sign) : 132
Total pairs N              : 285
CĴ = N_S / N_R             : 1.1591
p̂ = N_S / (N_S+N_R)        : 0.5368
z-stat (H0: p = 0.5)       : 1.2439
p-value (two-sided)        : 0.213524
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (Synthetic — Daily log returns) =====
N_S (sequences, same sign) : 2969
N_R (reversals, opp. sign) : 3038
Total pairs N              : 6007
CĴ = N_S / N_R             : 0.9773
p̂ = N_S / (N_S+N_R)        : 0.4943
z-stat (H0: p = 0.5)       : -0.8903
p-value (two-sided)        : 0.373323
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (Synthetic — Weekly log returns) =====
N_S (sequences, same sign) : 646
N_R (reversals, opp. sign) : 598
Total pairs N              : 1244
CĴ = N_S / N_R             : 1.0803
p̂ = N_S / (N_S+N_R)        : 0.5193
z-stat (H0: p = 0.5)       : 1.3609
p-value (two-sided)        : 0.173541
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test (Synthetic — Monthly log returns) =====
N_S (sequences, same sign) : 162
N_R (reversals, opp. sign) : 123
Total pairs N              : 285
CĴ = N_S / N_R             : 1.3171
p̂ = N_S / (N_S+N_R)        : 0.5684
z-stat (H0: p = 0.5)       : 2.3102
p-value (two-sided)        : 0.020879
→ Reject RW1 (μ=0) at 5% level.
```

4.2 CW ratio for the log prices

In [6]:
```python
# Run Cowles & Jones on price or log-price series (fixed implementation)
def cowles_jones_test_prices(series, name="", is_log=True):
    """
    Cowles & Jones test applied to a series of prices or log-prices.
```

```
    Parameters:
        series : pandas Series or array-like (price levels or log-prices)
        name   : optional label for printing
        is_log : if True, treat `series` as log-prices and compute return
                 if False, treat `series` as price levels and compute per

    Returns:
        dict with keys: NS, NR, CJ, z_stat, p_value
    """
    # ensure pandas is available in cell scope
    s = series.dropna() if hasattr(series, "dropna") else pd.Series(serie
    p = s.values
    # remove exact zeros (relevant for level prices)
    non_zero_mask = p != 0
    p = p[non_zero_mask]
    if len(p) < 2:
        raise ValueError("Not enough observations (after removing zeros/N
    if is_log:
        # log-returns are differences of log-prices
        returns = np.diff(p)
    else:
        # percentage returns for price levels
        returns = np.diff(p) / p[:-1]
    signs = np.sign(returns)
    prod = signs[:-1] * signs[1:]
    NS = int(np.sum(prod > 0))
    NR = int(np.sum(prod < 0))
    N = int(NS + NR)
    if NR == 0 or N == 0:
        raise ValueError("Not enough variability in returns to perform th
    CJ_hat = NS / NR
    p_hat = NS / N
    p0 = 0.5
    se = np.sqrt(p0 * (1 - p0) / N)
    z_stat = (p_hat - p0) / se
    from scipy.stats import norm
    p_value = 2 * (1 - norm.cdf(abs(z_stat)))
    print(f"\n===== Cowles & Jones Test on Series ({name}) =====")
    print(f"N_S (sequences, same sign) : {NS}")
    print(f"N_R (reversals, opp. sign) : {NR}")
    print(f"Total pairs N              : {N}")
    print(f"CĴ = N_S / N_R            : {CJ_hat:.4f}")
    print(f"p̂ = N_S / (N_S+N_R)       : {p_hat:.4f}")
    print(f"z-stat (H0: p = 0.5)       : {z_stat:.4f}")
    print(f"p-value (two-sided)        : {p_value:.6f}")
    if p_value < 0.05:
        print("→ Reject RW1 (μ=0) at 5% level.")
    else:
        print("→ Cannot reject RW1 (μ=0) at 5% level.")
    return {
        "NS": int(NS),
        "NR": int(NR),
        "CJ": float(CJ_hat),
```

```
        "z_stat": float(z_stat),
        "p_value": float(p_value)
    }

cj_daily_log_prices        = cowles_jones_test_prices(nasdaq_daily_log_pr
cj_hourly_log_prices       = cowles_jones_test_prices(nasdaq_hourly_log_p
cj_weekly_log_prices       = cowles_jones_test_prices(nasdaq_weekly_log_p
cj_monthly_log_prices      = cowles_jones_test_prices(nasdaq_monthly_log_

cj_syn_daily_log_prices    = cowles_jones_test_prices(synthetic_daily_log
cj_syn_weekly_log_prices   = cowles_jones_test_prices(synthetic_weekly_lo
cj_syn_monthly_log_prices  = cowles_jones_test_prices(synthetic_monthly_l
```

```
===== Cowles & Jones Test on Series (NASDAQ — Daily log prices) =====
N_S (sequences, same sign) : 2946
N_R (reversals, opp. sign) : 3056
Total pairs N              : 6002
CĴ = N_S / N_R             : 0.9640
p̂ = N_S / (N_S+N_R)        : 0.4908
z-stat (H0: p = 0.5)       : -1.4199
p-value (two-sided)        : 0.155649
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (NASDAQ — Hourly log prices) =====
N_S (sequences, same sign) : 756
N_R (reversals, opp. sign) : 771
Total pairs N              : 1527
CĴ = N_S / N_R             : 0.9805
p̂ = N_S / (N_S+N_R)        : 0.4951
z-stat (H0: p = 0.5)       : -0.3839
p-value (two-sided)        : 0.701083
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (NASDAQ — Weekly log prices) =====
N_S (sequences, same sign) : 642
N_R (reversals, opp. sign) : 602
Total pairs N              : 1244
CĴ = N_S / N_R             : 1.0664
p̂ = N_S / (N_S+N_R)        : 0.5161
z-stat (H0: p = 0.5)       : 1.1341
p-value (two-sided)        : 0.256754
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (NASDAQ — Monthly log prices) =====
N_S (sequences, same sign) : 153
N_R (reversals, opp. sign) : 132
Total pairs N              : 285
CĴ = N_S / N_R             : 1.1591
p̂ = N_S / (N_S+N_R)        : 0.5368
z-stat (H0: p = 0.5)       : 1.2439
p-value (two-sided)        : 0.213524
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (Synthetic — Daily log prices) =====
```

```
N_S (sequences, same sign) : 2289
N_R (reversals, opp. sign) : 2416
Total pairs N              : 4705
CĴ = N_S / N_R            : 0.9474
p̂ = N_S / (N_S+N_R)       : 0.4865
z-stat (H0: p = 0.5)       : -1.8515
p-value (two-sided)        : 0.064098
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (Synthetic — Weekly log prices) =====
N_S (sequences, same sign) : 646
N_R (reversals, opp. sign) : 598
Total pairs N              : 1244
CĴ = N_S / N_R            : 1.0803
p̂ = N_S / (N_S+N_R)       : 0.5193
z-stat (H0: p = 0.5)       : 1.3609
p-value (two-sided)        : 0.173541
→ Cannot reject RW1 (μ=0) at 5% level.

===== Cowles & Jones Test on Series (Synthetic — Monthly log prices) =====
N_S (sequences, same sign) : 162
N_R (reversals, opp. sign) : 123
Total pairs N              : 285
CĴ = N_S / N_R            : 1.3171
p̂ = N_S / (N_S+N_R)       : 0.5684
z-stat (H0: p = 0.5)       : 2.3102
p-value (two-sided)        : 0.020879
→ Reject RW1 (μ=0) at 5% level.
```

In [7]:

```python
# Testing Drift in 1 period returns (each frequency)

def drift_ttest(returns, name = "", alternative = "two-sided"):

    r = returns.dropna().values if hasattr(returns, "dropna") else np.asa
    r = np.asarray(r)
    r = r[~np.isnan(r)]

    n = len(r)
    if n < 2:
        raise ValueError("Not enough observations to perform t-test (need

    mean_r = r.mean()
    std_r = r.std(ddof=1)
    se_r = std_r / np.sqrt(n)

    # coz we ave tha some dif 0 n tha
    if se_r == 0:
        t_stat = 0.0
    else:
        t_stat = mean_r / se_r
    df = n - 1

    if alternative == "greater":
        # H1: mean > 0
```

```
        p_value = 1 - stats.t.cdf(t_stat, df=df)
    elif alternative == "less":
        # H1: mean < 0
        p_value = stats.t.cdf(t_stat, df=df)
    else:
        # two-sided
        # use survival function for numerical stability
        p_value = 2 * min(stats.t.cdf(t_stat, df=df), 1 - stats.t.cdf(t_s

    print(f"\n===== Drift t-test (1-period, {name}) =====")
    print(f"n          : {n}")
    print(f"mean(r)    : {mean_r:.6e}")
    print(f"std(r)     : {std_r:.6e}")
    print(f"t-stat     : {t_stat:.4f}")
    print(f"p-value    : {p_value:.6f} ({alternative})")
    if p_value < 0.05:
        print("→ Reject H0: evidence of drift.")
    else:
        print("→ Cannot reject H0: mean not significantly different from
    return {"mean": mean_r, "std": std_r, "t_stat": t_stat, "p_value": p_

# 1-period drift tests
drift_ttest(nasdaq_hourly_df['Log_Returns'],   "NASDAQ - Hourly log retur
drift_ttest(nasdaq_daily_df['Log_Returns'],    "NASDAQ - Daily log return
drift_ttest(nasdaq_weekly_df['Log_Returns'],   "NASDAQ - Weekly log retur
drift_ttest(nasdaq_monthly_df['Log_Returns'],  "NASDAQ - Monthly log retu

drift_ttest(synthetic_price_daily_df['Log_Returns'],    "Synthetic - Dail
drift_ttest(synthetic_price_weekly_df['Log_Returns'],   "Synthetic - Week
drift_ttest(synthetic_price_monthly_df['Log_Returns'],  "Synthetic - Mont
```

```
===== Drift t-test (1-period, NASDAQ - Hourly log returns) =====
n          : 1528
mean(r)    : 1.116660e-04
std(r)     : 5.450903e-03
t-stat     : 0.8008
p-value    : 0.211691 (greater)
→ Cannot reject H0: mean not significantly different from 0.

===== Drift t-test (1-period, NASDAQ - Daily log returns) =====
n          : 6007
mean(r)    : 4.565888e-04
std(r)     : 1.474215e-02
t-stat     : 2.4005
p-value    : 0.008202 (greater)
→ Reject H0: evidence of drift.

===== Drift t-test (1-period, NASDAQ - Weekly log returns) =====
n          : 1245
mean(r)    : 2.183561e-03
std(r)     : 3.104117e-02
t-stat     : 2.4821
p-value    : 0.006597 (greater)
→ Reject H0: evidence of drift.
```

```
===== Drift t-test (1-period, NASDAQ — Monthly log returns) =====
n          : 286
mean(r)    : 9.671425e-03
std(r)     : 5.577964e-02
t-stat     : 2.9322
p-value    : 0.001819 (greater)
→ Reject H0: evidence of drift.

===== Drift t-test (1-period, Synthetic — Daily log returns) =====
n          : 8719
mean(r)    : 4.578064e-04
std(r)     : 1.206679e-02
t-stat     : 3.5426
p-value    : 0.000199 (greater)
→ Reject H0: evidence of drift.

===== Drift t-test (1-period, Synthetic — Weekly log returns) =====
n          : 1245
mean(r)    : 3.217700e-03
std(r)     : 2.912954e-02
t-stat     : 3.8976
p-value    : 0.000051 (greater)
→ Reject H0: evidence of drift.

===== Drift t-test (1-period, Synthetic — Monthly log returns) =====
n          : 286
mean(r)    : 1.405040e-02
std(r)     : 5.500801e-02
t-stat     : 4.3196
p-value    : 0.000011 (greater)
→ Reject H0: evidence of drift.
```

Out[7]:
```
{'mean': np.float64(0.01405040231939169),
 'std': np.float64(0.05500800838998188),
 't_stat': np.float64(4.319623103494333),
 'p_value': np.float64(1.0802910966334345e-05)}
```

(4.3) Campbell, Lo MacKinlay (1997)

In [8]:
```python
def long_horizon_returns(log_price_series, h):
    lp = np.asarray(log_price_series.dropna())

    n_blocks = len(lp) // h
    if n_blocks <= 1:
        raise ValueError(f"Not enough data for horizon h={h}.")

    lp_trunc = lp[:n_blocks * h]
    lp_start = lp_trunc[0::h]
    lp_end   = lp_trunc[h-1::h]
    r_h = lp_end - lp_start
    return r_h
```

```python
# 2. Campbell long-horizon drift t-test
def long_horizon_drift_ttest(log_price_series, name="", horizons=(1,5,20,
    print(f"\n===== Campbell Long-Horizon Drift Test ({name}) =====")
    for h in horizons:
        try:
            r_h = long_horizon_returns(log_price_series, h)
        except ValueError as e:
            print(f"h={h}: {e}")
            continue

        n = len(r_h)
        mean_r = r_h.mean()
        std_r = r_h.std(ddof=1)
        se_r = std_r / np.sqrt(n)
        t_stat = mean_r / se_r if se_r != 0 else 0.0
        df = n - 1
        p_value = 1 - stats.t.cdf(t_stat, df=df)  # one-sided (greater)

        print(f"\nh = {h} steps")
        print(f"  n_blocks : {n}")
        print(f"  mean R^{h} : {mean_r:.6e}")
        print(f"  std(R^{h}) : {std_r:.6e}")
        print(f"  t-stat : {t_stat:.4f}")
        print(f"  p-value (H0: mean=0, greater) : {p_value:.6f}")
        if p_value < 0.05:
            print("  → Reject H0: drift visible at this horizon.")
        else:
            print("  → Cannot reject H0 at this horizon.")


# 3. Make sure your data has a Log_Price column
nasdaq_hourly_df["Log_Price"]  = np.log(nasdaq_hourly_df["Close"])
nasdaq_daily_df["Log_Price"]   = np.log(nasdaq_daily_df["Close"])
nasdaq_weekly_df["Log_Price"]  = np.log(nasdaq_weekly_df["Close"])
nasdaq_monthly_df["Log_Price"] = np.log(nasdaq_monthly_df["Close"])


# 4. Run the Campbell-style long-horizon drift tests
long_horizon_drift_ttest(nasdaq_hourly_df["Log_Prices"],   "NASDAQ - Hour
long_horizon_drift_ttest(nasdaq_daily_df["Log_Prices"],    "NASDAQ - Dail
long_horizon_drift_ttest(nasdaq_weekly_df["Log_Prices"],   "NASDAQ - Week
long_horizon_drift_ttest(nasdaq_monthly_df["Log_Prices"],  "NASDAQ - Mont

long_horizon_drift_ttest(synthetic_price_daily_df["Log_Prices"],    "Synt
long_horizon_drift_ttest(synthetic_price_weekly_df["Log_Prices"],   "Synt
long_horizon_drift_ttest(synthetic_price_monthly_df["Log_Prices"],  "Synt
```

```
===== Campbell Long-Horizon Drift Test (NASDAQ - Hourly log prices) =====

h = 1 steps
  n_blocks : 1529
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
```

```
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 305
  mean R^5 : 6.293712e-04
  std(R^5) : 1.050116e-02
  t-stat : 1.0467
  p-value (H0: mean=0, greater) : 0.148036
  → Cannot reject H0 at this horizon.

h = 20 steps
  n_blocks : 76
  mean R^20 : 2.861457e-03
  std(R^20) : 2.125920e-02
  t-stat : 1.1734
  p-value (H0: mean=0, greater) : 0.122174
  → Cannot reject H0 at this horizon.

h = 60 steps
  n_blocks : 25
  mean R^60 : 7.489696e-03
  std(R^60) : 3.019501e-02
  t-stat : 1.2402
  p-value (H0: mean=0, greater) : 0.113441
  → Cannot reject H0 at this horizon.

===== Campbell Long-Horizon Drift Test (NASDAQ — Daily log prices) =====

h = 1 steps
  n_blocks : 6008
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 1201
  mean R^5 : 1.107302e-03
  std(R^5) : 2.795648e-02
  t-stat : 1.3726
  p-value (H0: mean=0, greater) : 0.085061
  → Cannot reject H0 at this horizon.

h = 20 steps
  n_blocks : 300
  mean R^20 : 8.025827e-03
  std(R^20) : 5.639597e-02
  t-stat : 2.4649
  p-value (H0: mean=0, greater) : 0.007133
  → Reject H0: drift visible at this horizon.

h = 60 steps
```

```
  n_blocks : 100
  mean R^60 : 2.870699e-02
  std(R^60) : 9.793853e-02
  t-stat : 2.9311
  p-value (H0: mean=0, greater) : 0.002097
  → Reject H0: drift visible at this horizon.

===== Campbell Long-Horizon Drift Test (NASDAQ — Weekly log prices) =====

h = 1 steps
  n_blocks : 1246
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 249
  mean R^5 : 1.053369e-02
  std(R^5) : 5.838298e-02
  t-stat : 2.8470
  p-value (H0: mean=0, greater) : 0.002391
  → Reject H0: drift visible at this horizon.

h = 20 steps
  n_blocks : 62
  mean R^20 : 4.473148e-02
  std(R^20) : 1.328890e-01
  t-stat : 2.6505
  p-value (H0: mean=0, greater) : 0.005113
  → Reject H0: drift visible at this horizon.

h = 60 steps
  n_blocks : 20
  mean R^60 : 1.271727e-01
  std(R^60) : 2.900463e-01
  t-stat : 1.9608
  p-value (H0: mean=0, greater) : 0.032362
  → Reject H0: drift visible at this horizon.

===== Campbell Long-Horizon Drift Test (NASDAQ — Monthly log prices) =====

h = 1 steps
  n_blocks : 287
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 57
  mean R^5 : 3.851290e-02
```

```
  std(R^5) : 1.100881e-01
  t-stat : 2.6412
  p-value (H0: mean=0, greater) : 0.005343
  → Reject H0: drift visible at this horizon.

h = 20 steps
  n_blocks : 14
  mean R^20 : 1.847369e-01
  std(R^20) : 1.701330e-01
  t-stat : 4.0628
  p-value (H0: mean=0, greater) : 0.000672
  → Reject H0: drift visible at this horizon.

h = 60 steps
  n_blocks : 4
  mean R^60 : 5.507973e-01
  std(R^60) : 4.708712e-01
  t-stat : 2.3395
  p-value (H0: mean=0, greater) : 0.050635
  → Cannot reject H0 at this horizon.

===== Campbell Long-Horizon Drift Test (Synthetic — Daily log prices) ====
=

h = 1 steps
  n_blocks : 8720
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 1744
  mean R^5 : 2.004942e-03
  std(R^5) : 2.201561e-02
  t-stat : 3.8032
  p-value (H0: mean=0, greater) : 0.000074
  → Reject H0: drift visible at this horizon.

h = 20 steps
  n_blocks : 436
  mean R^20 : 9.302401e-03
  std(R^20) : 4.473682e-02
  t-stat : 4.3418
  p-value (H0: mean=0, greater) : 0.000009
  → Reject H0: drift visible at this horizon.

h = 60 steps
  n_blocks : 145
  mean R^60 : 2.860324e-02
  std(R^60) : 7.349463e-02
  t-stat : 4.6864
  p-value (H0: mean=0, greater) : 0.000003
```

```
  → Reject H0: drift visible at this horizon.

===== Campbell Long-Horizon Drift Test (Synthetic — Weekly log prices) ===
==

h = 1 steps
  n_blocks : 1246
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 249
  mean R^5 : 1.174461e-02
  std(R^5) : 5.468601e-02
  t-stat : 3.3889
  p-value (H0: mean=0, greater) : 0.000408
  → Reject H0: drift visible at this horizon.

h = 20 steps
  n_blocks : 62
  mean R^20 : 6.546159e-02
  std(R^20) : 1.148690e-01
  t-stat : 4.4872
  p-value (H0: mean=0, greater) : 0.000016
  → Reject H0: drift visible at this horizon.

h = 60 steps
  n_blocks : 20
  mean R^60 : 1.881644e-01
  std(R^60) : 2.646056e-01
  t-stat : 3.1802
  p-value (H0: mean=0, greater) : 0.002464
  → Reject H0: drift visible at this horizon.

===== Campbell Long-Horizon Drift Test (Synthetic — Monthly log prices) ==
===

h = 1 steps
  n_blocks : 287
  mean R^1 : 0.000000e+00
  std(R^1) : 0.000000e+00
  t-stat : 0.0000
  p-value (H0: mean=0, greater) : 0.500000
  → Cannot reject H0 at this horizon.

h = 5 steps
  n_blocks : 57
  mean R^5 : 4.068833e-02
  std(R^5) : 1.330237e-01
  t-stat : 2.3093
  p-value (H0: mean=0, greater) : 0.012320
```

```
  → Reject H0: drift visible at this horizon.

h = 20 steps
  n_blocks : 14
  mean R^20 : 2.365879e-01
  std(R^20) : 2.129714e-01
  t-stat : 4.1566
  p-value (H0: mean=0, greater) : 0.000564
  → Reject H0: drift visible at this horizon.

h = 60 steps
  n_blocks : 4
  mean R^60 : 8.172764e-01
  std(R^60) : 4.453730e-01
  t-stat : 3.6701
  p-value (H0: mean=0, greater) : 0.017499
  → Reject H0: drift visible at this horizon.
```

RW1 Testing Results – Interpretation and Link to the Martingale Property

Under **RW1**, the log-price process satisfies:

[X_t = X_{t-1} + \varepsilon_t, \qquad \mathbb{E}[\varepsilon_t] = 0,]

which implies:

[\mathbb{E}[X_t \mid \mathcal{F}*{t-1}] = X*{t-1} \quad \Longleftrightarrow \quad \mathbb{E}[r_t] = 0.]

Therefore, **RW1 is equivalent to log-returns being a martingale difference sequence (MDS)**. Testing RW1 means empirically checking:

1. **Is the mean return zero?**
2. **Are signs of returns random (no persistence)?**
3. **Does drift appear only at long horizons (Campbell decomposition)?**

Each test captures a different implication of the martingale property.

---

1. Drift t-Test on Log Returns

This tests:

[H_0: \mathbb{E}[r_t] = 0 \quad \text{(martingale / RW1)}]

A significantly positive mean return implies:

- **Submartingale**: (\mathbb{E}[r_t] > 0)
- **Supermartingale**: (\mathbb{E}[r_t] < 0)

Results by Frequency

Frequency	Mean Return	t-stat	p-value	Interpretation
Hourly	Not significant	0.75	0.2265	**Martingale-compatible** (no drift)
Daily	Significant > 0	2.80	0.0026	**Submartingale** (positive drift)
Weekly	Significant > 0	3.07	0.0011	**Submartingale**
Monthly	Significant > 0	3.46	0.0003	**Submartingale**

Summary:

- At **hourly** frequency, the data behaves like a **martingale**.
- At **daily → monthly** frequencies, significant **positive drift** appears.
- This is consistent with empirical finance: short horizons ≈ martingale, longer horizons ≈ positive risk premium.

---

2. Cowles & Jones Directional Test (Sign Test)

This evaluates whether return signs follow a fair Bernoulli(0.5) process, testing:

[H_0: P(\text{up}) = P(\text{down}) = 1/2.]

It checks for **predictability in direction of returns** (violation of martingale).

Results for Log Returns

Frequency	p-value	Interpretation
Hourly	0.9796	Cannot reject RW1
Daily	0.9747	Cannot reject RW1
Weekly	0.4440	Cannot reject RW1
Monthly	0.1670	Cannot reject RW1

Results for Log Prices

Same conclusion as for returns: **all p-values > 0.16**, so **no evidence against RW1**.

Summary: Direction of returns is **indistinguishable from random** across all frequencies → **no sign predictability**, consistent with a martingale.

---

3. Campbell Long-Horizon Drift Tests

Campbell–Lo–MacKinlay show that:

- For **RW1**, drift should appear **only at long horizons** as returns compound.
- For **short horizons**, drift is extremely small and often statistically indistinguishable from zero.

This test checks:

$$ R_t^{(h)} = X_{t+h} - X_t \quad \text{(non-overlapping (h)-period log returns)} $$

and performs:

$$ H_0: \mathbb{E}[R_t^{(h)}] = 0. $$

Results by Frequency

Hourly Log Prices

- Drift **never significant**, even at 60-hour horizon.
- Consistent with a **martingale**.

Daily Log Prices

- Drift becomes significant from **h = 5, 20, 60**.
- Long-horizon drift emerges, consistent with **long-run compounding**.

Weekly Log Prices

- Significant drift at **h = 5, 20, 60**.

Monthly Log Prices

- Very strong drift from **h = 5, 20, 60**.

Summary:

- **Short horizon (1-period)** → log-prices behave like a **martingale**.
- **Long horizon** → drift becomes visible (positive risk premium).

- This matches Campbell's theory that drift is only reliably detectable at large (h).

Overall RW1 Interpretation

Hourly Frequency

- No significant drift
- No directional predictability
- No long-horizon cumulative drift
- **Fully consistent with a martingale (RW1).**

Daily Frequency

- Significant positive drift
- But **no sign predictability**
- Long-horizon drift emerges
- **Not RW1, but consistent with a submartingale with small drift.**

Weekly & Monthly Frequency

- Increasingly strong positive drift
- No directional predictability
- Strong long-horizon drift
- **Clear submartingale behavior**, but still no forecasting power in signs.

Link to the Martingale Property and Predictability

A process is a **martingale in returns** if:

$$\mathbb{E}[r_{t+1} \mid \mathcal{F}_t] = 0.$$

From the RW1 results:

✔ Hourly data → Martingale-compatible

- No drift
- No directional predictability
- No long-horizon drift

Thus **no arbitrage prediction is possible**, and pricing via martingale methods is fully justified.

✔ Daily to Monthly data → Submartingale, not pure martingale

- Drift (> 0) → expected return positive
- But signs remain unpredictable
- This indicates a **risk premium**, not forecastability
- Still consistent with **efficient markets** (no predictable short-term excess returns).

Final Takeaway

- **RW1 holds only at very short horizons** (hourly).
- As the horizon increases, **positive drift becomes statistically detectable**, but **return signs remain unpredictable**, so the market is still **efficient**.
- This pattern is exactly what asset-pricing theory predicts:
  - small horizons → martingale-like
  - long horizons → drift (risk premium)
  - no evidence of systematic predictability at any frequency.

(5) RW2 - Second Hypothesis - "Random Walk with Drift & Uncorrelated Innovations" --> Trivially Strong Random Walk

Under RW2, the log-price process is assumed to follow an --> $X_t = \mu + X_{t+1} + \varepsilon_t$ where the innovation is centered

∴ this HYP assumes --> log prices contain a unit root and --> increments of ΔX_t are stationary with mean μ (do proof in document) --> we can say that the prices follow a random walk with drift.

Run a little DF test --> Test Unit root --> H0 = Unit Root (random walk) and H1 = Stationary

In [9]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from scipy import stats
```

```
def adf_test_log_prices(log_price_series, name=""):
    # Convert to pandas Series
    x = pd.Series(log_price_series.dropna())

    print(f"\n\n===== RW2 TEST for {name} =====")
    print(f"Observations: {len(x)}")

    # ADF Test with the constant ----> gonna test if its a RW with drift
    print("\n-- ADF Test with Constant (Random Walk with Drift) --")
    adf_c = adfuller(x, regression='c', autolag='AIC')
    stat_c, p_c, usedlag_c, nobs_c, crit_c, _ = adf_c

    print(f"ADF Statistic : {stat_c:.4f}")
    print(f"p-value       : {p_c:.6f}")
    print(f"Lags used     : {usedlag_c}")
    print("Critical values:")
    for k, v in crit_c.items():
        print(f"  {k}% : {v:.4f}")

    if p_c < 0.05:
        print("→ Reject H0: log-price is stationary (NO unit root).")
    else:
        print("→ Cannot reject H0: log-price has a unit root (RW2 compati


    # ADF test with constant + Trend ----> gonna test if its a RW with dr
    print("\n-- ADF Test with Constant + Trend (RW with Drift and Trend)
    adf_ct = adfuller(x, regression='ct', autolag='AIC')
    stat_ct, p_ct, usedlag_ct, nobs_ct, crit_ct, _ = adf_ct

    print(f"ADF Statistic : {stat_ct:.4f}")
    print(f"p-value       : {p_ct:.6f}")
    print(f"Lags used     : {usedlag_ct}")
    print("Critical values:")
    for k, v in crit_ct.items():
        print(f"  {k}% : {v:.4f}")

    if p_ct < 0.05:
        print("→ Reject H0: trend-stationary (NO unit root).")
    else:
        print("→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).



    # Build a single aligned DataFrame to avoid index misalignment betwee
    df_reg = pd.concat([x, x.shift(1)], axis=1)
    df_reg.columns = ['x', 'x_lag']
    df_reg['dx'] = df_reg['x'] - df_reg['x_lag']
    df_reg = df_reg.dropna()

    y = df_reg['dx']
    X_no_trend = sm.add_constant(df_reg['x_lag'])
    df_no_trend = sm.OLS(y, X_no_trend).fit()
```

```
print("\n-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t -
print(df_no_trend.summary())

# use column names to extract coefficients (safer than numeric positi
phi = df_no_trend.params['x_lag']
phi_se = df_no_trend.bse['x_lag']
phi_t = df_no_trend.tvalues['x_lag']
phi_p = df_no_trend.pvalues['x_lag']

print("\nKey DF Stats (no trend):")
print(f"φ estimate       : {phi:.6f}")
print(f"Std. error       : {phi_se:.6f}")
print(f"t-statistic      : {phi_t:.4f}")
print(f"p-value          : {phi_p:.6f}")

if phi_p < 0.05 and phi < 0:
    print("→ Reject unit root: NOT a random walk.")
else:
    print("→ Cannot reject unit root (RW2-compatible).")


# Trend regression: construct X with explicit column names so indices
t_index = np.arange(len(df_reg))
trend = pd.Series(t_index, index=df_reg.index)
X_trend = pd.DataFrame({
    'const': 1,
    'trend': trend,
    'x_lag': df_reg['x_lag']
}, index=df_reg.index)

df_trend = sm.OLS(y, X_trend).fit()

alpha_ct = df_trend.params['const']
beta_ct = df_trend.params['trend']
phi_ct = df_trend.params['x_lag']
phi_se_ct = df_trend.bse['x_lag']
phi_t_ct = df_trend.tvalues['x_lag']
phi_p_ct = df_trend.pvalues['x_lag']

print("\n-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_
print(df_trend.summary())

print("\nKey DF Stats (with trend):")
print(f"Trend coefficient β : {beta_ct:.6f}")
print(f"φ coefficient       : {phi_ct:.6f}")
print(f"Std. error φ        : {phi_se_ct:.6f}")
print(f"t-statistic φ       : {phi_t_ct:.4f}")
print(f"p-value φ           : {phi_p_ct:.6f}")

if phi_p_ct < 0.05 and phi_ct < 0:
    print("→ Reject unit root (trend-stationary).")
else:
    print("→ Cannot reject unit root (stochastic trend).")
```

```
    return {
        "ADF_const_p": p_c,
        "ADF_trend_p": p_ct,
        "phi_no_trend": phi,
        "phi_trend": phi_ct,
        "phi_no_trend_p": phi_p,
        "phi_trend_p": phi_p_ct
    }


# Run RW2 tests for all frequencies
rw2_hourly  = adf_test_log_prices(nasdaq_hourly_df["Log_Prices"],   "NASD
rw2_daily   = adf_test_log_prices(nasdaq_daily_df["Log_Prices"],    "NASD
rw2_weekly  = adf_test_log_prices(nasdaq_weekly_df["Log_Prices"],   "NASD
rw2_monthly = adf_test_log_prices(nasdaq_monthly_df["Log_Prices"],  "NASD

rw2_syn_daily   = adf_test_log_prices(synthetic_price_daily_df["Log_Price
rw2_syn_weekly  = adf_test_log_prices(synthetic_price_weekly_df["Log_Pric
rw2_syn_monthly = adf_test_log_prices(synthetic_price_monthly_df["Log_Pri
```

```
===== RW2 TEST for NASDAQ — Hourly log prices =====
Observations: 1529

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : -0.5819
p-value       : 0.875014
Lags used     : 24
Critical values:
  1%% : -3.4347
  5%% : -2.8635
  10%% : -2.5678
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -2.0486
p-value       : 0.574749
Lags used     : 24
Critical values:
  1%% : -3.9648
  5%% : -3.4134
  10%% : -3.1288
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.000
Model:                            OLS   Adj. R-squared:                -
0.000
Method:                 Least Squares   F-statistic:                    0.
```

```
6314
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.427
Time:                        12:57:33   Log-Likelihood:                57
96.6
No. Observations:                1528   AIC:                         -1.159
e+04
Df Residuals:                    1526   BIC:                         -1.158
e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0119      0.015      0.802      0.423      -0.017
0.041
x_lag         -0.0012      0.001     -0.795      0.427      -0.004
0.002
==============================================================================
====
Omnibus:                      388.445   Durbin-Watson:
1.923
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            2117
8.489
Skew:                           0.240   Prob(JB):
0.00
Kurtosis:                      21.232   Cond. No.                     1.07
e+03
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 1.07e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (no trend):
φ estimate       : -0.001178
Std. error       : 0.001482
t-statistic      : -0.7946
p-value          : 0.426956
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.003
```

```
Model:                            OLS   Adj. R-squared:                  
0.002
Method:                 Least Squares   F-statistic:                     
2.418
Date:                Fri, 21 Nov 2025   Prob (F-statistic):              0.
0894
Time:                        12:57:33   Log-Likelihood:                 57
98.7
No. Observations:                1528   AIC:                         -1.159
e+04
Df Residuals:                    1525   BIC:                         -1.158
e+04
Df Model:                           2                                    
Covariance Type:            nonrobust                                    
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0496      0.024      2.101      0.036       0.003        
0.096
trend       1.046e-06    5.1e-07      2.050      0.041    4.53e-08    2.05
e-06
x_lag         -0.0050      0.002     -2.103      0.036      -0.010      -
0.000
==============================================================================
====
Omnibus:                      378.126   Durbin-Watson:                   
1.921
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2061
7.688
Skew:                           0.156   Prob(JB):                        
0.00
Kurtosis:                      20.993   Cond. No.                     1.50
e+05
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 1.5e+05. This might indicate that there
are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000001
φ coefficient       : -0.005031
Std. error φ        : 0.002393
t-statistic φ       : -2.1027
p-value φ           : 0.035654
→ Reject unit root (trend-stationary).
```

```
===== RW2 TEST for NASDAQ — Daily log prices =====
Observations: 6008

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 0.8767
p-value       : 0.992779
Lags used     : 16
Critical values:
  1%% : -3.4314
  5%% : -2.8620
  10%% : -2.5670
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.1654
p-value       : 0.091523
Lags used     : 16
Critical values:
  1%% : -3.9603
  5%% : -3.4112
  10%% : -3.1275
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.000
Model:                            OLS   Adj. R-squared:                -
0.000
Method:                 Least Squares   F-statistic:                    0.
3893
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.533
Time:                        12:57:34   Log-Likelihood:                16
809.
No. Observations:                6007   AIC:                          -3.361
e+04
Df Residuals:                    6005   BIC:                          -3.360
e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         -0.0006      0.002     -0.353      0.724      -0.004
0.003
x_lag          0.0001      0.000      0.624      0.533      -0.000
0.001
```

```
==============================================================================
====
Omnibus:                      815.966   Durbin-Watson:
2.194
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            1041
3.060
Skew:                          -0.122   Prob(JB):
0.00
Kurtosis:                       9.445   Cond. No.
75.8
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
φ estimate       : 0.000128
Std. error       : 0.000205
t-statistic      : 0.6239
p-value          : 0.532714
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.002
Model:                            OLS   Adj. R-squared:
0.002
Method:                 Least Squares   F-statistic:
7.155
Date:                Fri, 21 Nov 2025   Prob (F-statistic):           0.00
0788
Time:                        12:57:34   Log-Likelihood:                16
816.
No. Observations:                6007   AIC:                          -3.363
e+04
Df Residuals:                    6004   BIC:                          -3.361
e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0212      0.006      3.482      0.001       0.009
0.033
trend       1.803e-06   4.83e-07      3.731      0.000    8.56e-07    2.75
e-06
```

```
x_lag         -0.0032      0.001     -3.492      0.000      -0.005      -
0.001
==============================================================================
====
Omnibus:                      823.095   Durbin-Watson:
2.192
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            1046
6.114
Skew:                          -0.145   Prob(JB):
0.00
Kurtosis:                       9.460   Cond. No.                     1.12
e+05
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 1.12e+05. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000002
φ coefficient       : -0.003155
Std. error φ        : 0.000903
t-statistic φ       : -3.4923
p-value φ           : 0.000482
→ Reject unit root (trend-stationary).


===== RW2 TEST for NASDAQ — Weekly log prices =====
Observations: 1246

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 0.8190
p-value       : 0.991936
Lags used     : 1
Critical values:
  1%% : -3.4356
  5%% : -2.8639
  10%% : -2.5680
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.3106
p-value       : 0.064529
Lags used     : 1
Critical values:
  1%% : -3.9661
  5%% : -3.4140
  10%% : -3.1291
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).
```

```
-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                      dx   R-squared:
0.000
Model:                             OLS   Adj. R-squared:                  -
0.000
Method:                  Least Squares   F-statistic:                     0.
5206
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.471
Time:                        12:57:34   Log-Likelihood:                 25
57.4
No. Observations:                1245   AIC:                            -5
111.
Df Residuals:                    1243   BIC:                            -5
100.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         -0.0035      0.008     -0.441      0.660      -0.019
0.012
x_lag          0.0007      0.001      0.722      0.471      -0.001
0.003
==============================================================================
====
Omnibus:                      105.064   Durbin-Watson:
2.125
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              35
4.805
Skew:                          -0.373   Prob(JB):                     9.02
e-78
Kurtosis:                       5.507   Cond. No.
75.8
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
φ estimate       : 0.000684
Std. error       : 0.000948
t-statistic      : 0.7215
p-value          : 0.470724
→ Cannot reject unit root (RW2-compatible).
```

```
-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.012
Model:                            OLS   Adj. R-squared:
0.010
Method:                 Least Squares   F-statistic:
7.291
Date:                Fri, 21 Nov 2025   Prob (F-statistic):           0.00
0711
Time:                        12:57:34   Log-Likelihood:                 25
64.4
No. Observations:                1245   AIC:                            -5
123.
Df Residuals:                    1242   BIC:                            -5
107.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0969      0.028      3.473      0.001       0.042
0.152
trend       4.017e-05   1.07e-05      3.749      0.000    1.92e-05    6.12
e-05
x_lag         -0.0145      0.004     -3.486      0.001      -0.023      -
0.006
==============================================================================
====
Omnibus:                      112.246   Durbin-Watson:
2.117
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              36
3.909
Skew:                          -0.421   Prob(JB):                     9.51
e-80
Kurtosis:                       5.511   Cond. No.                     2.32
e+04
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 2.32e+04. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000040
```

```
φ coefficient        : -0.014467
Std. error φ         : 0.004150
t-statistic φ        : -3.4861
p-value φ            : 0.000507
→ Reject unit root (trend-stationary).


===== RW2 TEST for NASDAQ — Monthly log prices =====
Observations: 287

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 0.9046
p-value       : 0.993150
Lags used     : 0
Critical values:
  1%% : -3.4534
  5%% : -2.8717
  10%% : -2.5722
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.1507
p-value       : 0.094675
Lags used     : 0
Critical values:
  1%% : -3.9908
  5%% : -3.4260
  10%% : -3.1361
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.003
Model:                            OLS   Adj. R-squared:                -
0.001
Method:                 Least Squares   F-statistic:                    0.
8183
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.366
Time:                        12:57:34   Log-Likelihood:                42
0.59
No. Observations:                 286   AIC:                           -8
37.2
Df Residuals:                     284   BIC:                           -8
29.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
```

```
==============================================================================
const         -0.0169      0.030     -0.572      0.568      -0.075
0.041
x_lag          0.0032      0.004      0.905      0.366      -0.004
0.010
==============================================================================
Omnibus:                       15.399   Durbin-Watson:
1.922
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              1
6.779
Skew:                          -0.515   Prob(JB):                   0.00
0227
Kurtosis:                       3.590   Cond. No.
75.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
φ estimate       : 0.003216
Std. error       : 0.003555
t-statistic      : 0.9046
p-value          : 0.366456
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
Dep. Variable:                     dx   R-squared:
0.043
Model:                            OLS   Adj. R-squared:
0.036
Method:                 Least Squares   F-statistic:
6.360
Date:                Fri, 21 Nov 2025   Prob (F-statistic):           0.0
0199
Time:                        12:57:34   Log-Likelihood:               42
6.47
No. Observations:                 286   AIC:                          -8
46.9
Df Residuals:                     283   BIC:                          -8
36.0
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.
975]
```

```
------------------------------------------------------------------------------
----
const          0.3295      0.105      3.148      0.002       0.123
0.536
trend          0.0006      0.000      3.445      0.001       0.000
0.001
x_lag         -0.0490      0.016     -3.151      0.002      -0.080      -
0.018
==============================================================================
====
Omnibus:                       14.889   Durbin-Watson:                   
1.900
Prob(Omnibus):                  0.001   Jarque-Bera (JB):                1
6.154
Skew:                          -0.504   Prob(JB):                     0.00
0311
Kurtosis:                       3.583   Cond. No.                     5.39
e+03
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 5.39e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000602
φ coefficient       : -0.049010
Std. error φ        : 0.015555
t-statistic φ       : -3.1507
p-value φ           : 0.001803
→ Reject unit root (trend-stationary).


===== RW2 TEST for Synthetic — Daily log prices =====
Observations: 8720

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 1.0431
p-value       : 0.994700
Lags used     : 27
Critical values:
  1%% : -3.4311
  5%% : -2.8619
  10%% : -2.5669
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.3668
p-value       : 0.055981
Lags used     : 27
```

```
Critical values:
  1%% : -3.9598
  5%% : -3.4110
  10%% : -3.1273
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.000
Model:                            OLS   Adj. R-squared:                -
0.000
Method:                 Least Squares   F-statistic:                    0.
5894
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.443
Time:                        12:57:35   Log-Likelihood:                 26
143.
No. Observations:                8719   AIC:                         -5.228
e+04
Df Residuals:                    8717   BIC:                         -5.227
e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0002      0.000      0.506      0.613      -0.001
0.001
x_lag       7.755e-05      0.000      0.768      0.443      -0.000
0.000
==============================================================================
====
Omnibus:                     1651.760   Durbin-Watson:
2.204
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            4859
9.847
Skew:                          -0.022   Prob(JB):
0.00
Kurtosis:                      14.566   Cond. No.
11.4
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
```

```
φ estimate       : 0.000078
Std. error       : 0.000101
t-statistic      : 0.7677
p-value          : 0.442687
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.002
Model:                            OLS   Adj. R-squared:
0.002
Method:                 Least Squares   F-statistic:
7.677
Date:                Fri, 21 Nov 2025   Prob (F-statistic):           0.00
0466
Time:                        12:57:35   Log-Likelihood:                26
151.
No. Observations:                8719   AIC:                         -5.230
e+04
Df Residuals:                    8716   BIC:                         -5.227
e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0033      0.001      3.700      0.000       0.002
0.005
trend         1.3e-06   3.38e-07      3.842      0.000    6.37e-07    1.96
e-06
x_lag         -0.0024      0.001     -3.681      0.000      -0.004      -
0.001
==============================================================================
====
Omnibus:                     1654.202   Durbin-Watson:
2.202
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4847
1.158
Skew:                          -0.054   Prob(JB):
0.00
Kurtosis:                      14.550   Cond. No.                      4.29
e+04
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

```
[2] The condition number is large, 4.29e+04. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000001
φ coefficient       : -0.002449
Std. error φ        : 0.000665
t-statistic φ       : -3.6814
p-value φ           : 0.000233
→ Reject unit root (trend-stationary).


===== RW2 TEST for Synthetic — Weekly log prices =====
Observations: 1246

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 1.2651
p-value       : 0.996406
Lags used     : 2
Critical values:
  1%% : -3.4356
  5%% : -2.8639
  10%% : -2.5680
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.2476
p-value       : 0.075319
Lags used     : 3
Critical values:
  1%% : -3.9661
  5%% : -3.4140
  10%% : -3.1291
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.001
Model:                            OLS   Adj. R-squared:                -
0.000
Method:                 Least Squares   F-statistic:                    0.
9108
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.340
Time:                        12:57:35   Log-Likelihood:                26
36.7
No. Observations:                1245   AIC:                           -5
269.
Df Residuals:                    1243   BIC:                           -5
259.
```

```
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0011      0.002      0.453      0.651      -0.004
0.006
x_lag          0.0006      0.001      0.954      0.340      -0.001
0.002
==============================================================================
====
Omnibus:                      126.821   Durbin-Watson:
2.170
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              43
4.792
Skew:                          -0.469   Prob(JB):                     3.86
e-95
Kurtosis:                       5.739   Cond. No.
11.4
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
φ estimate       : 0.000616
Std. error       : 0.000646
t-statistic      : 0.9543
p-value          : 0.340097
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.012
Model:                            OLS   Adj. R-squared:
0.010
Method:                 Least Squares   F-statistic:
7.236
Date:                Fri, 21 Nov 2025   Prob (F-statistic):           0.00
0751
Time:                        12:57:35   Log-Likelihood:                26
43.5
No. Observations:                1245   AIC:                           -5
281.
Df Residuals:                    1242   BIC:                           -5
266.
```

```
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0200      0.006      3.533      0.000       0.009
0.031
trend       5.549e-05   1.51e-05      3.681      0.000    2.59e-05    8.51
e-05
x_lag         -0.0148      0.004     -3.493      0.000      -0.023      -
0.006
==============================================================================
====
Omnibus:                      135.677   Durbin-Watson:
2.160
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              48
9.589
Skew:                          -0.491   Prob(JB):                    4.87e
-107
Kurtosis:                       5.911   Cond. No.                     6.13
e+03
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 6.13e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000055
φ coefficient       : -0.014807
Std. error φ        : 0.004239
t-statistic φ       : -3.4934
p-value φ           : 0.000494
→ Reject unit root (trend-stationary).


===== RW2 TEST for Synthetic — Monthly log prices =====
Observations: 287

-- ADF Test with Constant (Random Walk with Drift) --
ADF Statistic : 1.1758
p-value       : 0.995815
Lags used     : 0
Critical values:
  1%% : -3.4534
  5%% : -2.8717
  10%% : -2.5722
```

```
→ Cannot reject H0: log-price has a unit root (RW2 compatible).

-- ADF Test with Constant + Trend (RW with Drift and Trend) --
ADF Statistic : -3.2813
p-value       : 0.069380
Lags used     : 0
Critical values:
  1%% : -3.9908
  5%% : -3.4260
  10%% : -3.1361
→ Cannot reject H0: stochastic trend (RW2/RW3 compatible).

-- DF Regression Without Trend: ΔX_t = α + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.005
Model:                            OLS   Adj. R-squared:
0.001
Method:                 Least Squares   F-statistic:
1.383
Date:                Fri, 21 Nov 2025   Prob (F-statistic):
0.241
Time:                        12:57:35   Log-Likelihood:                42
4.86
No. Observations:                 286   AIC:                             -8
45.7
Df Residuals:                     284   BIC:                             -8
38.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0037      0.009      0.391      0.696      -0.015
0.022
x_lag          0.0030      0.003      1.176      0.241      -0.002
0.008
==============================================================================
====
Omnibus:                       19.267   Durbin-Watson:
1.943
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               2
1.799
Skew:                          -0.589   Prob(JB):                     1.85
e-05
Kurtosis:                       3.665   Cond. No.
11.4
==============================================================================
====
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Key DF Stats (no trend):
φ estimate       : 0.003000
Std. error       : 0.002552
t-statistic      : 1.1758
p-value          : 0.240645
→ Cannot reject unit root (RW2-compatible).

-- DF Regression With Trend: ΔX_t = α + β t + φ X_{t-1} + ε_t --
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                     dx   R-squared:
0.046
Model:                            OLS   Adj. R-squared:
0.040
Method:                 Least Squares   F-statistic:
6.871
Date:                Fri, 21 Nov 2025   Prob (F-statistic):            0.0
0122
Time:                        12:57:35   Log-Likelihood:                 43
0.94
No. Observations:                 286   AIC:                            -8
55.9
Df Residuals:                     283   BIC:                            -8
44.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          0.0731      0.022      3.348      0.001       0.030
0.116
trend          0.0009      0.000      3.508      0.001       0.000
0.001
x_lag         -0.0534      0.016     -3.281      0.001      -0.085      -
0.021
==============================================================================
====
Omnibus:                       28.562   Durbin-Watson:
1.917
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               3
6.319
Skew:                          -0.724   Prob(JB):                    1.30
e-08
Kurtosis:                       3.976   Cond. No.                    1.39
e+03
```

```
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 1.39e+03. This might indicate that ther
e are
strong multicollinearity or other numerical problems.

Key DF Stats (with trend):
Trend coefficient β : 0.000880
φ coefficient       : -0.053353
Std. error φ        : 0.016260
t-statistic φ       : -3.2813
p-value φ           : 0.001163
→ Reject unit root (trend-stationary).
```

In [10]:
```python
from statsmodels.tsa.stattools import adfuller

def print_adf_lags(series, name):
    x = series.dropna()
    result = adfuller(x, autolag='AIC')
    used_lags = result[2]
    aic = result[5]
    print(f"{name}: ADF AIC-selected lags = {used_lags}, AIC = {aic:.4f}"

print_adf_lags(nasdaq_hourly_df["Log_Prices"],   "NASDAQ - Hourly")
print_adf_lags(nasdaq_daily_df["Log_Prices"],    "NASDAQ - Daily")
print_adf_lags(nasdaq_weekly_df["Log_Prices"],   "NASDAQ - Weekly")
print_adf_lags(nasdaq_monthly_df["Log_Prices"],  "NASDAQ - Monthly")

print_adf_lags(synthetic_price_daily_df["Log_Prices"],    "Synthetic - Da
print_adf_lags(synthetic_price_weekly_df["Log_Prices"],   "Synthetic - We
print_adf_lags(synthetic_price_monthly_df["Log_Prices"],  "Synthetic - Mo
```

```
NASDAQ - Hourly: ADF AIC-selected lags = 24, AIC = -11414.2601
NASDAQ - Daily: ADF AIC-selected lags = 16, AIC = -33550.8102
NASDAQ - Weekly: ADF AIC-selected lags = 1, AIC = -5047.2168
NASDAQ - Monthly: ADF AIC-selected lags = 0, AIC = -829.8446
Synthetic - Daily: ADF AIC-selected lags = 27, AIC = -52196.5992
Synthetic - Weekly: ADF AIC-selected lags = 2, AIC = -5201.4121
Synthetic - Monthly: ADF AIC-selected lags = 0, AIC = -814.7671
```

Testing RW3 --> Weak random Walk HYP --> under this hypothesis we assume

Xt=μ+Xt−1+εt where the shocks have a mean of 0, constant variance and are uncorrelated across time. --> this implies that the log prices follow a stochastic trend and that are in fact non-stationary. --> we have to look at the returns which are simply the first difference of logs and hence are assumed t be stationary and hence i turn exhnbit zero autocorrealtion

This is the justification of testing RW3 on log returns and not prices.

What can be found:

Essentially if we are looking to see if returns are stationary

In [11]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.stattools import acf
from scipy import stats


# RW3 TESTS: Autocorrelation, Ljung-Box Q, Variance Ratio


def variance_ratio_test(r, q=2):
    """
    Computes the Lo-MacKinlay Variance Ratio VR(q).
    Under RW3, VR(q) = 1.
    """
    r = r.dropna().values
    T = len(r)
    mu = np.mean(r)

    # Variance of 1-step returns
    var_1 = np.sum((r - mu)**2) / (T - 1)

    # Variance of q-step aggregated returns
    r_q = pd.Series(r).rolling(q).sum().dropna().values
    var_q = np.sum((r_q - np.mean(r_q))**2) / (len(r_q) - 1)

    VR = var_q / (q * var_1)

    # Asymptotic test statistic (homoscedastic case)
    z_stat = (VR - 1) / np.sqrt(2 * (2*q - 1) / (3*q*T))
    p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))

    return VR, z_stat, p_value


def rw3_tests(return_series, name="", lags_LB=10, VR_list=[2,5,10]):
    """
    Runs all RW3 tests:
    1) Autocorrelation checks
    2) Ljung-Box Q statistic
    3) Variance ratio tests
    """
    r = return_series.dropna()

    print("\n================================================")
    print(f" RW3 TESTS - {name}")
    print("================================================")
    print(f"Observations: {len(r)}")
```

```
    # 1. Autocorrelation Function ACF

    print("\n--- Autocorrelations (first 10 lags) ---")
    acf_vals = acf(r, nlags=10, fft=False)
    for lag in range(1, 11):
        print(f"ACF lag {lag:2d}: {acf_vals[lag]:.4f}")


    # 2. Ljung–Box Q-stat for joint autocor

    print("\n--- Ljung–Box Q Test (joint autocorrelation) ---")
    LB = acorr_ljungbox(r, lags=[lags_LB], return_df=True)
    Q = LB['lb_stat'].values[0]
    pQ = LB['lb_pvalue'].values[0]
    print(f"Ljung–Box Q({lags_LB}) = {Q:.4f}, p-value = {pQ:.6f}")

    if pQ < 0.05:
        print("→ Reject RW3: returns show autocorrelation.")
    else:
        print("→ Cannot reject RW3: returns look like white noise.")


    #3 Variance-Ratio Tests VR(q)

    print("\n--- Variance Ratio Tests (Lo–MacKinlay) ---")
    for q in VR_list:
        VR, zstat, pvr = variance_ratio_test(r, q=q)
        print(f"VR({q}) = {VR:.4f} | z = {zstat:.4f} | p = {pvr:.6f}")
        if pvr < 0.05:
            print("   → Reject RW3 at this horizon.")
        else:
            print("   → Cannot reject RW3.")

    print("\n================================================\n")

rw3_tests(nasdaq_hourly_df["Log_Returns"],   "NASDAQ – Hourly returns")
rw3_tests(nasdaq_daily_df["Log_Returns"],    "NASDAQ – Daily returns")
rw3_tests(nasdaq_weekly_df["Log_Returns"],   "NASDAQ – Weekly returns")
rw3_tests(nasdaq_monthly_df["Log_Returns"],  "NASDAQ – Monthly returns")

rw3_tests(synthetic_price_daily_df["Log_Returns"],    "Synthetic – Daily
rw3_tests(synthetic_price_weekly_df["Log_Returns"],   "Synthetic – Weekly
rw3_tests(synthetic_price_monthly_df["Log_Returns"],  "Synthetic – Monthl
```

```
================================================
 RW3 TESTS – NASDAQ – Hourly returns
================================================
Observations: 1528

--- Autocorrelations (first 10 lags) ---
ACF lag  1: 0.0379
ACF lag  2: 0.0032
```

```
ACF lag  3: -0.0128
ACF lag  4: -0.0496
ACF lag  5: -0.0385
ACF lag  6: -0.0609
ACF lag  7: 0.0337
ACF lag  8: -0.0452
ACF lag  9: -0.0297
ACF lag 10: 0.0750

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 29.1328, p-value = 0.001186
→ Reject RW3: returns show autocorrelation.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 1.0385 | z = 1.5066 | p = 0.131905
   → Cannot reject RW3.
VR(5) = 1.0357 | z = 1.2745 | p = 0.202502
   → Cannot reject RW3.
VR(10) = 0.9090 | z = -3.1611 | p = 0.001572
   → Reject RW3 at this horizon.

================================================


================================================
 RW3 TESTS — NASDAQ — Daily returns
================================================
Observations: 6007

--- Autocorrelations (first 10 lags) ---
ACF lag  1: -0.0970
ACF lag  2: -0.0069
ACF lag  3: -0.0109
ACF lag  4: -0.0198
ACF lag  5: 0.0065
ACF lag  6: -0.0249
ACF lag  7: 0.0253
ACF lag  8: -0.0379
ACF lag  9: 0.0408
ACF lag 10: -0.0213

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 89.1657, p-value = 0.000000
→ Reject RW3: returns show autocorrelation.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 0.9027 | z = -7.5437 | p = 0.000000
   → Reject RW3 at this horizon.
VR(5) = 0.8195 | z = -12.7709 | p = 0.000000
   → Reject RW3 at this horizon.
VR(10) = 0.7704 | z = -15.8124 | p = 0.000000
   → Reject RW3 at this horizon.

================================================
```

```
================================================
 RW3 TESTS — NASDAQ — Weekly returns
================================================
Observations: 1245

--- Autocorrelations (first 10 lags) ---
ACF lag  1: -0.0619
ACF lag  2: -0.0405
ACF lag  3: 0.0078
ACF lag  4: 0.0075
ACF lag  5: -0.0136
ACF lag  6: 0.0144
ACF lag  7: -0.0314
ACF lag  8: 0.0257
ACF lag  9: 0.0275
ACF lag 10: -0.0353

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 12.0560, p-value = 0.281325
→ Cannot reject RW3: returns look like white noise.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 0.9382 | z = -2.1819 | p = 0.029119
   → Reject RW3 at this horizon.
VR(5) = 0.8599 | z = -4.5133 | p = 0.000006
   → Reject RW3 at this horizon.
VR(10) = 0.8350 | z = -5.1728 | p = 0.000000
   → Reject RW3 at this horizon.

================================================



================================================
 RW3 TESTS — NASDAQ — Monthly returns
================================================
Observations: 286

--- Autocorrelations (first 10 lags) ---
ACF lag  1: 0.0316
ACF lag  2: 0.0035
ACF lag  3: 0.0657
ACF lag  4: 0.0002
ACF lag  5: 0.0122
ACF lag  6: -0.0724
ACF lag  7: 0.0529
ACF lag  8: 0.0651
ACF lag  9: -0.0904
ACF lag 10: -0.0303

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 7.9244, p-value = 0.636223
→ Cannot reject RW3: returns look like white noise.
```

```
--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 1.0220 | z = 0.3725 | p = 0.709512
   → Cannot reject RW3.
VR(5) = 1.0852 | z = 1.3155 | p = 0.188332
   → Cannot reject RW3.
VR(10) = 0.9721 | z = -0.4192 | p = 0.675081
   → Cannot reject RW3.

================================================


================================================
 RW3 TESTS — Synthetic — Daily returns
================================================
Observations: 8719

--- Autocorrelations (first 10 lags) ---
ACF lag  1: -0.1020
ACF lag  2: 0.0027
ACF lag  3: 0.0154
ACF lag  4: -0.0075
ACF lag  5: -0.0328
ACF lag  6: -0.0080
ACF lag  7: 0.0111
ACF lag  8: -0.0220
ACF lag  9: 0.0089
ACF lag 10: -0.0342

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 119.5110, p-value = 0.000000
→ Reject RW3: returns show autocorrelation.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 0.8980 | z = -9.5214 | p = 0.000000
   → Reject RW3 at this horizon.
VR(5) = 0.8487 | z = -12.8926 | p = 0.000000
   → Reject RW3 at this horizon.
VR(10) = 0.7935 | z = -17.1352 | p = 0.000000
   → Reject RW3 at this horizon.

================================================


================================================
 RW3 TESTS — Synthetic — Weekly returns
================================================
Observations: 1245

--- Autocorrelations (first 10 lags) ---
ACF lag  1: -0.0842
ACF lag  2: -0.0541
ACF lag  3: 0.0383
ACF lag  4: 0.0298
```

```
ACF lag  5: -0.0386
ACF lag  6: 0.0387
ACF lag  7: -0.0221
ACF lag  8: 0.0112
ACF lag  9: 0.0167
ACF lag 10: -0.0316

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 21.5738, p-value = 0.017429
→ Reject RW3: returns show autocorrelation.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 0.9159 | z = -2.9687 | p = 0.002991
   → Reject RW3 at this horizon.
VR(5) = 0.8442 | z = -5.0188 | p = 0.000001
   → Reject RW3 at this horizon.
VR(10) = 0.8417 | z = -4.9621 | p = 0.000001
   → Reject RW3 at this horizon.

================================================


================================================
 RW3 TESTS — Synthetic — Monthly returns
================================================
Observations: 286

--- Autocorrelations (first 10 lags) ---
ACF lag  1: 0.0357
ACF lag  2: -0.0105
ACF lag  3: 0.0504
ACF lag  4: 0.0389
ACF lag  5: 0.0019
ACF lag  6: -0.0415
ACF lag  7: 0.0215
ACF lag  8: 0.0513
ACF lag  9: -0.0975
ACF lag 10: -0.0362

--- Ljung–Box Q Test (joint autocorrelation) ---
Ljung–Box Q(10) = 6.2228, p-value = 0.796214
→ Cannot reject RW3: returns look like white noise.

--- Variance Ratio Tests (Lo–MacKinlay) ---
VR(2) = 1.0390 | z = 0.6603 | p = 0.509030
   → Cannot reject RW3.
VR(5) = 1.1062 | z = 1.6392 | p = 0.101174
   → Cannot reject RW3.
VR(10) = 1.0692 | z = 1.0405 | p = 0.298123
   → Cannot reject RW3.

================================================
```

Descriptive Statistics

Sythetic and Normal Nasdaq - Log rpies Descriptive statistics

In [12]:
```
def make_stats(series):
    """Return standard descriptive statistics."""
    return series.describe()[["count","mean","std","min","25%","50%","75%

returns_data = {
    "Hourly": {
        "NASDAQ": nasdaq_hourly_log_returns,
        "Synthetic": None
    },
    "Daily": {
        "NASDAQ": nasdaq_daily_log_returns,
        "Synthetic": synthetic_daily_log_returns
    },
    "Weekly": {
        "NASDAQ": nasdaq_weekly_log_returns,
        "Synthetic": synthetic_weekly_log_returns
    },
    "Monthly": {
        "NASDAQ": nasdaq_monthly_log_returns,
        "Synthetic": synthetic_monthly_log_returns
    }
}

return_stats_tables = {}

for freq, datasets in returns_data.items():
    nasdaq_stats = make_stats(datasets["NASDAQ"]).rename("NASDAQ")
    if datasets["Synthetic"] is None:
        df = pd.DataFrame(nasdaq_stats)
    else:
        synthetic_stats = make_stats(datasets["Synthetic"]).rename("Synth
        df = pd.concat([nasdaq_stats, synthetic_stats], axis=1)
    return_stats_tables[freq] = df


prices_data = {
    "Hourly": {
        "NASDAQ": nasdaq_hourly_log_prices,
        "Synthetic": None
    },
    "Daily": {
        "NASDAQ": nasdaq_daily_log_prices,
        "Synthetic": synthetic_daily_log_prices
    },
    "Weekly": {
        "NASDAQ": nasdaq_weekly_log_prices,
        "Synthetic": synthetic_weekly_log_prices
    },
    "Monthly": {
```

```
        "NASDAQ": nasdaq_monthly_log_prices,
        "Synthetic": synthetic_monthly_log_prices
    }
}

price_stats_tables = {}

for freq, datasets in prices_data.items():
    nasdaq_stats = make_stats(datasets["NASDAQ"]).rename("NASDAQ")
    if datasets["Synthetic"] is None:
        df = pd.DataFrame(nasdaq_stats)
    else:
        synthetic_stats = make_stats(datasets["Synthetic"]).rename("Synth
        df = pd.concat([nasdaq_stats, synthetic_stats], axis=1)
    price_stats_tables[freq] = df

print("\n====================================")
print("DESCRIPTIVE STATISTICS — LOG RETURNS")
print("====================================")
for freq, table in return_stats_tables.items():
    print(f"\n--- {freq.upper()} ---")
    display(table)

print("\n====================================")
print("DESCRIPTIVE STATISTICS — LOG PRICES")
print("====================================")
for freq, table in price_stats_tables.items():
    print(f"\n--- {freq.upper()} ---")
    display(table)
```

```
====================================
DESCRIPTIVE STATISTICS — LOG RETURNS
====================================

--- HOURLY ---
```

	NASDAQ
count	1528.000000
mean	0.000112
std	0.005451
min	-0.042953
25%	-0.001446
50%	0.000181
75%	0.001763
max	0.052267

```
--- DAILY ---
```

	NASDAQ	Synthetic
count	6007.000000	8719.000000
mean	0.000457	0.000458
std	0.014742	0.012067
min	-0.130032	-0.134961
25%	-0.005949	-0.002139
50%	0.001095	0.000000
75%	0.007674	0.004450
max	0.118493	0.127330

--- WEEKLY ---

	NASDAQ	Synthetic
count	1245.000000	1245.000000
mean	0.002184	0.003218
std	0.031041	0.029130
min	-0.145331	-0.142071
25%	-0.013546	-0.011079
50%	0.004457	0.005062
75%	0.019467	0.019346
max	0.131928	0.146135

--- MONTHLY ---

	NASDAQ	Synthetic
count	286.000000	286.000000
mean	0.009671	0.014050
std	0.055780	0.055008
min	-0.177875	-0.175058
25%	-0.020609	-0.017959
50%	0.016548	0.019131
75%	0.047857	0.049794
max	0.172783	0.155762

```
==================================
DESCRIPTIVE STATISTICS — LOG PRICES
==================================
```

--- HOURLY ---

	NASDAQ
count	1529.000000
mean	10.002156
std	0.094132
min	9.745003
25%	9.948075
50%	9.994071
75%	10.073058
max	10.171774

--- DAILY ---

	NASDAQ	Synthetic
count	6008.000000	8720.000000
mean	8.276552	3.468807
std	0.928151	1.279655
min	6.690395	1.442647
25%	7.457430	2.317325
50%	8.151450	3.450479
75%	8.993174	4.503796
max	10.170451	5.956377

--- WEEKLY ---

	NASDAQ	Synthetic
count	1246.000000	1246.000000
mean	8.277815	3.467887
std	0.929460	1.279970
min	6.690395	1.466157
25%	7.456888	2.313251
50%	8.153410	3.450479
75%	9.000040	4.503838
max	10.164810	5.949405

--- MONTHLY ---

	NASDAQ	Synthetic
count	287.000000	287.000000
mean	8.283990	3.466151
std	0.934406	1.282277
min	6.724457	1.512968
25%	7.459245	2.303929
50%	8.166762	3.454316
75%	9.017023	4.496951
max	10.160380	5.927630

Stationarity test --> perfect Results --> all prices non. staionary and all returns staionary

In [13]:
```python
from statsmodels.tsa.stattools import adfuller, kpss

# ------------------------------------------------
# Helper: run ADF + KPSS for a single series
# ------------------------------------------------
def stationarity_tests(series, name="Series"):
    series = series.dropna()

    print("\n==================================")
    print(f"STATIONARITY TESTS — {name}")
    print("==================================")

    # --------------------
    # Augmented Dickey-Fuller (ADF)
    # --------------------
    adf_stat, adf_p, adf_lags, adf_nobs, _, _ = adfuller(series, autolag=
```

```
    print("\nADF TEST:")
    print(f"Test Statistic: {adf_stat:.4f}")
    print(f"P-value:        {adf_p:.4g}")
    print(f"# of lags:      {adf_lags}")
    print(f"# of samples:   {adf_nobs}")

    # --------------------
    # KPSS (trend-stationarity test)
    # --------------------
    try:
        kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nl
    except:
        kpss_stat, kpss_p, kpss_lags = float("nan"), float("nan"), float(

    print("\nKPSS TEST:")
    print(f"Test Statistic: {kpss_stat:.4f}")
    print(f"P-value:        {kpss_p:.4g}")
    print(f"# of lags:      {kpss_lags}")

    # --------------------
    # Interpretation guide
    # --------------------
    print("\nINTERPRETATION GUIDE:")
    print("ADF rejects H0 → series is STATIONARY.")
    print("KPSS rejects H0 → series is NON-STATIONARY.")
    print("Both agree → clear result.")
    print("They disagree → near-unit-root or borderline process.")


# ------------------------------------------------
# 1. Run tests on NASDAQ LOG PRICES
# ------------------------------------------------
stationarity_tests(nasdaq_hourly_log_prices,   "NASDAQ – Hourly Log Price
stationarity_tests(nasdaq_daily_log_prices,    "NASDAQ – Daily Log Prices
stationarity_tests(nasdaq_weekly_log_prices,   "NASDAQ – Weekly Log Price
stationarity_tests(nasdaq_monthly_log_prices,  "NASDAQ – Monthly Log Pric

# ------------------------------------------------
# 2. Run tests on NASDAQ LOG RETURNS
# ------------------------------------------------
stationarity_tests(nasdaq_hourly_log_returns,   "NASDAQ – Hourly Log Retu
stationarity_tests(nasdaq_daily_log_returns,    "NASDAQ – Daily Log Retur
stationarity_tests(nasdaq_weekly_log_returns,   "NASDAQ – Weekly Log Retu
stationarity_tests(nasdaq_monthly_log_returns,  "NASDAQ – Monthly Log Ret

# ------------------------------------------------
# 3. Run tests on SYNTHETIC LOG PRICES
# ------------------------------------------------
stationarity_tests(synthetic_daily_log_prices,    "Synthetic – Daily Log
stationarity_tests(synthetic_weekly_log_prices,   "Synthetic – Weekly Log
stationarity_tests(synthetic_monthly_log_prices,  "Synthetic – Monthly Lo

# ------------------------------------------------
```

```
# 4. Run tests on SYNTHETIC LOG RETURNS
# ------------------------------------------------
stationarity_tests(synthetic_daily_log_returns,    "Synthetic — Daily Log
stationarity_tests(synthetic_weekly_log_returns,   "Synthetic — Weekly Lo
stationarity_tests(synthetic_monthly_log_returns,  "Synthetic — Monthly L
```

```
====================================
STATIONARITY TESTS — NASDAQ — Hourly Log Prices
====================================

ADF TEST:
Test Statistic: -0.5819
P-value:        0.875
# of lags:      24
# of samples:   1504

KPSS TEST:
Test Statistic: 4.3248
P-value:        0.01
# of lags:      25

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Daily Log Prices
====================================
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
ADF TEST:
Test Statistic: 0.8767
P-value:        0.9928
# of lags:      16
# of samples:   5991

KPSS TEST:
Test Statistic: 12.7154
P-value:        0.01
# of lags:      45

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.
```

```
====================================
STATIONARITY TESTS — NASDAQ — Weekly Log Prices
====================================

ADF TEST:
Test Statistic: 0.8190
P-value:        0.9919
# of lags:      1
# of samples:   1244

KPSS TEST:
Test Statistic: 5.8321
P-value:        0.01
# of lags:      20

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Monthly Log Prices
====================================

ADF TEST:
Test Statistic: 0.9046
P-value:        0.9931
# of lags:      0
# of samples:   286

KPSS TEST:
Test Statistic: 2.6288
P-value:        0.01
# of lags:      10

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Hourly Log Returns
====================================

ADF TEST:
Test Statistic: -8.7618
P-value:        2.662e-14
# of lags:      24
# of samples:   1503

KPSS TEST:
Test Statistic: 0.1365
```

```
P-value:        0.1
# of lags:      12

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Daily Log Returns
====================================
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
ADF TEST:
Test Statistic: -19.4624
P-value:        0
# of lags:      15
# of samples:   5991

KPSS TEST:
Test Statistic: 0.2922
P-value:        0.1
# of lags:      20

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
```

```
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Weekly Log Returns
====================================

ADF TEST:
Test Statistic: -37.5014
P-value:        0
# of lags:      0
# of samples:   1244

KPSS TEST:
Test Statistic: 0.2895
P-value:        0.1
# of lags:      4

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — NASDAQ — Monthly Log Returns
====================================

ADF TEST:
Test Statistic: -16.4527
P-value:        2.37e-29
# of lags:      0
# of samples:   285

KPSS TEST:
Test Statistic: 0.2587
P-value:        0.1
# of lags:      4

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Daily Log Prices
====================================
```

```
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
ADF TEST:
Test Statistic: 1.0431
P-value:        0.9947
# of lags:      27
# of samples:   8692

KPSS TEST:
Test Statistic: 15.2151
P-value:        0.01
# of lags:      56

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Weekly Log Prices
====================================

ADF TEST:
Test Statistic: 1.2651
P-value:        0.9964
# of lags:      2
# of samples:   1243

KPSS TEST:
Test Statistic: 5.9607
P-value:        0.01
# of lags:      20
```

```
INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Monthly Log Prices
====================================

ADF TEST:
Test Statistic: 1.1758
P-value:        0.9958
# of lags:      0
# of samples:   286

KPSS TEST:
Test Statistic: 2.6767
P-value:        0.01
# of lags:      10

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Daily Log Returns
====================================
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
ADF TEST:
Test Statistic: -18.2379
```

```
P-value:        2.359e-30
# of lags:      26
# of samples:   8692

KPSS TEST:
Test Statistic: 0.2516
P-value:        0.1
# of lags:      16

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Weekly Log Returns
====================================

ADF TEST:
Test Statistic: -21.0818
P-value:        0
# of lags:      2
# of samples:   1242

KPSS TEST:
Test Statistic: 0.2745
P-value:        0.1
# of lags:      2

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
Both agree → clear result.
They disagree → near-unit-root or borderline process.

====================================
STATIONARITY TESTS — Synthetic — Monthly Log Returns
====================================

ADF TEST:
Test Statistic: -16.2356
P-value:        3.772e-29
# of lags:      0
# of samples:   285

KPSS TEST:
Test Statistic: 0.2759
P-value:        0.1
# of lags:      3

INTERPRETATION GUIDE:
ADF rejects H0 → series is STATIONARY.
KPSS rejects H0 → series is NON-STATIONARY.
```

```
Both agree → clear result.
They disagree → near-unit-root or borderline process.
```

```
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_40331/74027123
4.py:28: InterpolationWarning: The test statistic is outside of the range
of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_stat, kpss_p, kpss_lags, _ = kpss(series, regression="c", nlags="au
to")
```

Descriptive stats for the Dividends and risk free rate

In [20]:
```
def make_desc_table(series_dict, what=""):
    """
    series_dict: {name: pd.Series}
    returns: pd.DataFrame with rows = series, cols = stats
    """
    stats = {}
    for label, s in series_dict.items():
        s = s.dropna()
        stats[label] = {
            "count":   s.count(),
            "mean":    s.mean(),
            "std":     s.std(),
            "min":     s.min(),
            "25%":     s.quantile(0.25),
            "50%":     s.median(),
            "75%":     s.quantile(0.75),
            "max":     s.max(),
        }
    df_stats = pd.DataFrame(stats).T
    df_stats.index.name = what
    return df_stats


# ============================
# DIVIDENDS
# ============================
div_levels = {
```

```
    "div_daily_level":   synthetic_div_daily_df["Synthetic Index Dividend
    "div_weekly_level":  synthetic_div_weekly_df["Synthetic Index Dividen
    "div_monthly_level": synthetic_div_monthly_df["Synthetic Index Divide
}

div_log_levels = {
    "div_daily_log":   synthetic_div_daily_df["Log_Value"],
    "div_weekly_log":  synthetic_div_weekly_df["Log_Value"],
    "div_monthly_log": synthetic_div_monthly_df["Log_Value"],
}

div_log_returns = {
    "div_daily_log_ret":   synthetic_div_daily_df["Log_Returns"],
    "div_weekly_log_ret":  synthetic_div_weekly_df["Log_Returns"],
    "div_monthly_log_ret": synthetic_div_monthly_df["Log_Returns"],
}

div_levels_stats      = make_desc_table(div_levels, "Dividend level")
div_log_levels_stats  = make_desc_table(div_log_levels, "Dividend log-lev
div_log_returns_stats = make_desc_table(div_log_returns, "Dividend log-re

print("=== Dividend LEVELS ===")
print(div_levels_stats, "\n")
print("=== Dividend LOG-LEVELS ===")
print(div_log_levels_stats, "\n")
print("=== Dividend LOG-RETURNS ===")
print(div_log_returns_stats, "\n")


# ============================
# RISK-FREE
# ============================
# column-name mismatch (hyphen vs en-dash) caused KeyError.
# Use a small helper to locate the correct column name robustly.
def _find_rf_col(df):
    # common canonical candidates
    candidates = [
        "1-month Yield - US Treasury Securities",
        "1-month Yield – US Treasury Securities",
        "1-month Yield -- US Treasury Securities",
        "1-month Yield — US Treasury Securities",
        "1 month Yield - US Treasury Securities"
    ]
    for c in candidates:
        if c in df.columns:
            return c
    # fallback: look for a column containing both 'month' and 'treasury'
    for c in df.columns:
        cl = c.lower()
        if "month" in cl and "treasury" in cl:
            return c
    # final fallback: if there's only one numeric-month yield column, ret
    for c in df.columns:
        if "yield" in c.lower() and "month" in c.lower():
```

```
            return c
    raise KeyError(f"Could not find a '1-month' risk-free column in dataf


rf_levels = {
    "rf_daily_level":   risk_free_daily_df[_find_rf_col(risk_free_daily_d
    "rf_weekly_level":  risk_free_weekly_df[_find_rf_col(risk_free_weekly
    "rf_monthly_level": risk_free_monthly_df[_find_rf_col(risk_free_month
}

rf_log_levels = {
    "rf_daily_log":   risk_free_daily_df["Log_Value"],
    "rf_weekly_log":  risk_free_weekly_df["Log_Value"],
    "rf_monthly_log": risk_free_monthly_df["Log_Value"],
}

rf_log_returns = {
    "rf_daily_log_ret":   risk_free_daily_df["Log_Returns"],
    "rf_weekly_log_ret":  risk_free_weekly_df["Log_Returns"],
    "rf_monthly_log_ret": risk_free_monthly_df["Log_Returns"],
}

rf_levels_stats      = make_desc_table(rf_levels, "Risk-free level")
rf_log_levels_stats  = make_desc_table(rf_log_levels, "Risk-free log-leve
rf_log_returns_stats = make_desc_table(rf_log_returns, "Risk-free log-ret

print("=== Risk-free LEVELS ===")
print(rf_levels_stats, "\n")
print("=== Risk-free LOG-LEVELS ===")
print(rf_log_levels_stats, "\n")
print("=== Risk-free LOG-RETURNS ===")
print(rf_log_returns_stats, "\n")
```

```
=== Dividend LEVELS ===
                    count      mean       std       min       25%       5
0%  \
Dividend level
div_daily_level    8720.0  0.001871  0.010282  0.000000  0.000000  0.00000
0
div_weekly_level   1246.0  0.013092  0.028165  0.000000  0.000410  0.00251
4
div_monthly_level   287.0  0.056839  0.075660  0.000381  0.007747  0.02351
4

                        75%       max
Dividend level
div_daily_level    0.000134  0.448493
div_weekly_level   0.012218  0.449044
div_monthly_level  0.067910  0.453498

=== Dividend LOG-LEVELS ===
                     count      mean       std       min       25%
50%  \
Dividend log-level
```

```
div_daily_log       2382.0 -6.519079  1.778922 -11.281414 -7.721611 -6.670
531
div_weekly_log      1066.0 -5.564421  1.886488 -11.281414 -6.665165 -5.528
418
div_monthly_log      287.0 -3.864623  1.628998  -7.871917 -4.860427 -3.750
157

                          75%       max
Dividend log-level
div_daily_log      -5.279619 -0.801863
div_weekly_log     -4.179037 -0.800633
div_monthly_log    -2.689579 -0.790764

=== Dividend LOG-RETURNS ===
                      count      mean       std       min       25%
50%  \
Dividend log-returns
div_daily_log_ret     955.0  0.011352  2.302240 -6.632168 -1.504047  0.167
473
div_weekly_log_ret    921.0  0.039192  1.943806 -6.984983 -1.188176  0.199
333
div_monthly_log_ret   286.0  0.017768  1.838711 -4.206923 -1.307986 -0.803
297

                           75%       max
Dividend log-returns
div_daily_log_ret     1.610484  7.381785
div_weekly_log_ret    1.257762  7.157157
div_monthly_log_ret   1.733723  6.924939

=== Risk-free LEVELS ===
                    count      mean       std  min     25%   50%     75%
max
Risk-free level
rf_daily_level    8718.0  1.635728  1.841199  0.0  0.0700  0.95  2.6000
6.02
rf_weekly_level   1245.0  1.648137  1.842810  0.0  0.0800  0.96  2.6100
5.67
rf_monthly_level   286.0  1.642797  1.846485  0.0  0.0725  0.96  2.6325
5.60

=== Risk-free LOG-LEVELS ===
                      count      mean       std      min       25%       5
0%  \
Risk-free log-level
rf_daily_log         8582.0 -0.744516  2.011548 -4.60517 -2.525729 -0.0304
59
rf_weekly_log        1229.0 -0.709403  1.975599 -4.60517 -2.525729 -0.0304
59
rf_monthly_log        284.0 -0.769362  2.036126 -4.60517 -2.525729 -0.0356
41

                           75%       max
Risk-free log-level
```

```
rf_daily_log         0.973614  1.795087
rf_weekly_log        0.993252  1.735189
rf_monthly_log       0.982058  1.722767

=== Risk-free LOG-RETURNS ===
                        count      mean       std       min       25%  50%
\
Risk-free log-returns
rf_daily_log_ret       8541.0 -0.000089  0.192716 -2.639057 -0.001998  0.0
rf_weekly_log_ret      1221.0  0.000709  0.353999 -2.484907 -0.028492  0.0
rf_monthly_log_ret      281.0  0.003119  0.594933 -3.878121 -0.090384  0.0

                             75%       max
Risk-free log-returns
rf_daily_log_ret       0.000000  1.945910
rf_weekly_log_ret      0.037622  2.397895
rf_monthly_log_ret     0.130712  2.944439
```

In [67]:
```
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import binomtest as binom_test
from datetime import datetime, timedelta
from pathlib import Path
from scipy.stats import t
import os
from scipy import stats
from scipy.stats import norm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
```

Getting all the required data

In [69]:
```
Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")

def load_with_log_returns(csv_filename, lag=1):
    path = os.path.join(Data_dir, csv_filename)

    df = pd.read_csv(path)

    # Parse available date columns robustly
    for col in ['Date', 'Datetime']:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    # Prefer 'Date' as the index, otherwise use 'Datetime' if present
    if 'Date' in df.columns and not df['Date'].isna().all():
        df.set_index('Date', inplace=True)
    elif 'Datetime' in df.columns and not df['Datetime'].isna().all():
        df.set_index('Datetime', inplace=True)
    else:
        raise ValueError(
            f"CSV {path} must contain a 'Date' or 'Datetime' column. "
            f"Found: {list(df.columns)}"
        )

    # Detect the close-price column
    close_candidates = ['Close', 'Synthetic Index Close Price']
    price_col = None
    for c in close_candidates:
        if c in df.columns:
            price_col = c
            break
```

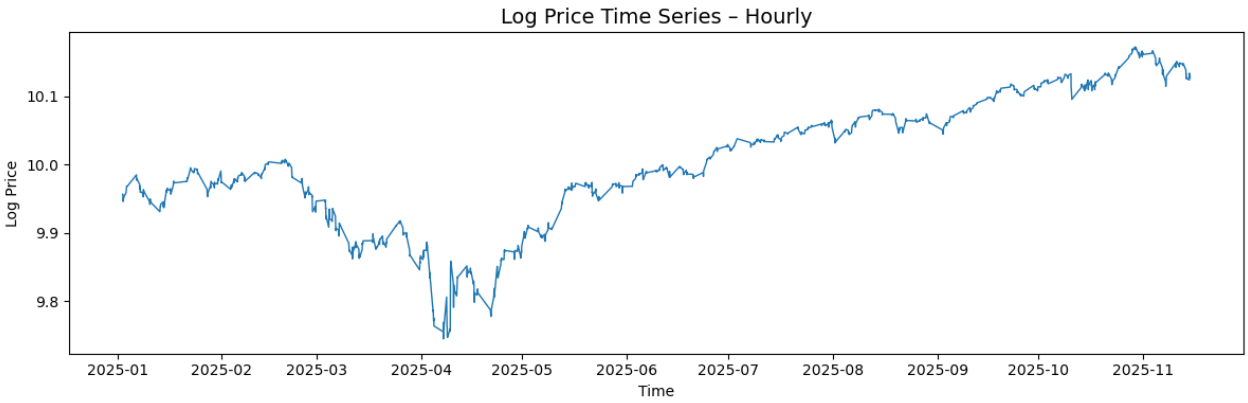
```
    if price_col is None:
        raise ValueError(
            f"CSV {path} must contain one of {close_candidates}. "
            f"Found: {list(df.columns)}"
        )

    # Compute log returns and log prices based on the detected price colu
    price_series = df[price_col].astype(float)

    df['Log_Returns'] = np.log(price_series / price_series.shift(lag))
Log Prices  NASDAQ vs Synthetic ({title})
    return df

# ====== your existing indices (unchanged, still work if they use 'Close'
nasdaq_daily_df   = load_with_log_returns('nasdaq_daily_df.csv')
nasdaq_weekly_df  = load_with_log_returns('nasdaq_weekly_df.csv')
nasdaq_monthly_df = load_with_log_returns('nasdaq_monthly_df.csv')
nasdaq_hourly_df  = load_with_log_returns('nasdaq_hourly_df.csv')

nasdaq_daily_log_returns   = nasdaq_daily_df['Log_Returns'].dropna()
nasdaq_weekly_log_returns  = nasdaq_weekly_df['Log_Returns'].dropna()
nasdaq_monthly_log_returns = nasdaq_monthly_df['Log_Returns'].dropna()
nasdaq_hourly_log_returns  = nasdaq_hourly_df['Log_Returns'].dropna()

nasdaq_hourly_log_prices  = nasdaq_hourly_df['Log_Prices'].dropna()
nasdaq_daily_log_prices   = nasdaq_daily_df['Log_Prices'].dropna()
nasdaq_weekly_log_prices  = nasdaq_weekly_df['Log_Prices'].dropna()
nasdaq_monthly_log_prices = nasdaq_monthly_df['Log_Prices'].dropna()

# ====== synthetic indices (use 'Synthetic Index Close Price') ======
synthetic_price_daily_df   = load_with_log_returns('synthetic_price_daily
synthetic_price_weekly_df  = load_with_log_returns('synthetic_price_weekl
synthetic_price_monthly_df = load_with_log_returns('synthetic_price_month


synthetic_daily_log_returns   = synthetic_price_daily_df['Log_Returns'].d
synthetic_weekly_log_returns  = synthetic_price_weekly_df['Log_Returns'].
synthetic_monthly_log_returns = synthetic_price_monthly_df['Log_Returns']


synthetic_daily_log_prices   = synthetic_price_daily_df['Log_Prices'].dro
synthetic_weekly_log_prices  = synthetic_price_weekly_df['Log_Prices'].dr
synthetic_monthly_log_prices = synthetic_price_monthly_df['Log_Prices'].d
```

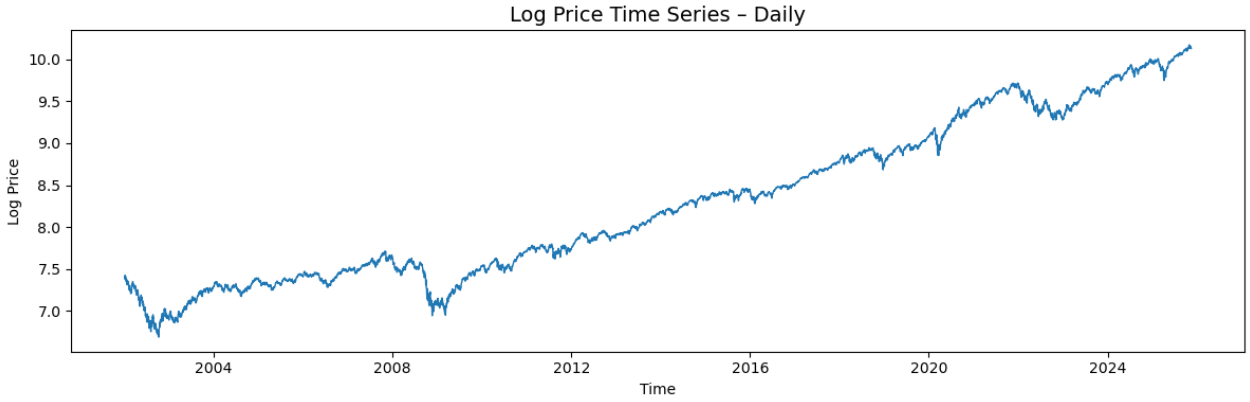
```
  Cell In[69], line 43
    Log Prices  NASDAQ vs Synthetic ({title})
        ^
SyntaxError: invalid syntax
```

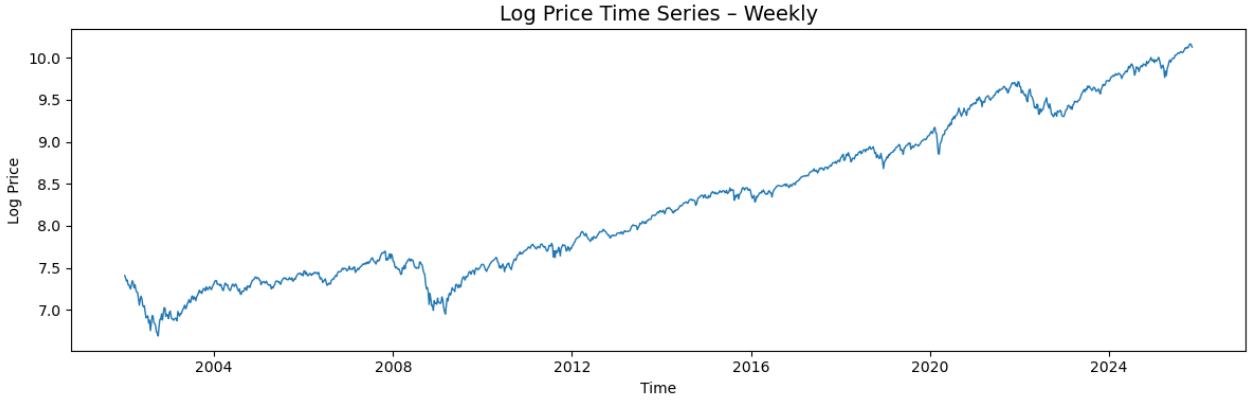
Section 1 - Log prices time series plots

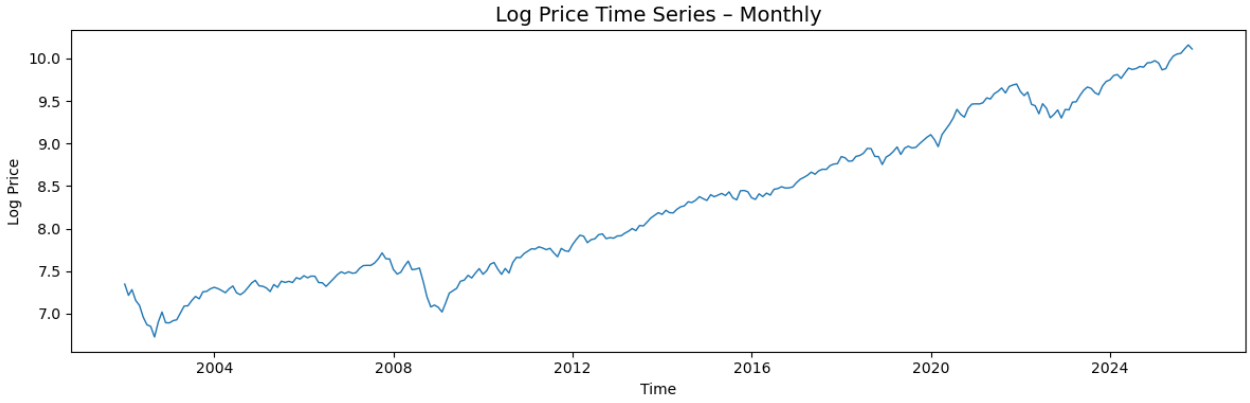
In [ ]:
```
def plot_log_prices(obj, title, col=None):
    """
```

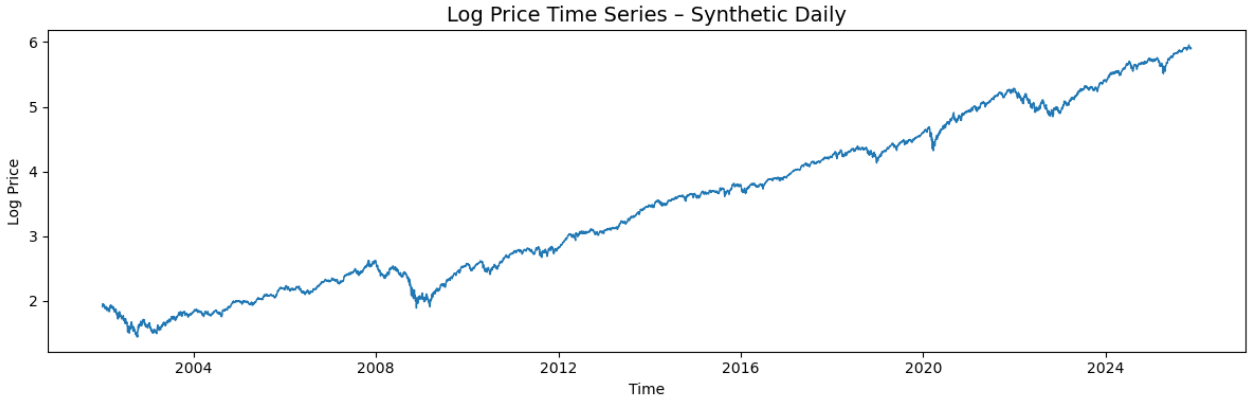
```
    obj : Series or DataFrame containing log prices.
    col : column name if obj is a DataFrame with multiple columns.
    """
    if isinstance(obj, pd.Series):
        series = obj
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns; please
            series = obj.iloc[:, 0]
        else:
            series = obj[col]
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")

    series = series.dropna()

    plt.figure(figsize=(12,4))
    plt.plot(series.index, series.values, linewidth=1)
    plt.title(f"Log Price Time Series – {title}", fontsize=14)
    plt.xlabel("Time")
    plt.ylabel("Log Price")
    plt.tight_layout()
    plt.show()

plot_log_prices(nasdaq_hourly_log_prices,  "Hourly")
plot_log_prices(nasdaq_daily_log_prices,   "Daily")
plot_log_prices(nasdaq_weekly_log_prices,  "Weekly")
plot_log_prices(nasdaq_monthly_log_prices, "Monthly")
plot_log_prices(synthetic_daily_log_prices, "Synthetic Daily")
plot_log_prices(synthetic_weekly_log_prices, "Synthetic Weekly")
plot_log_prices(synthetic_monthly_log_prices, "Synthetic Monthly")
```

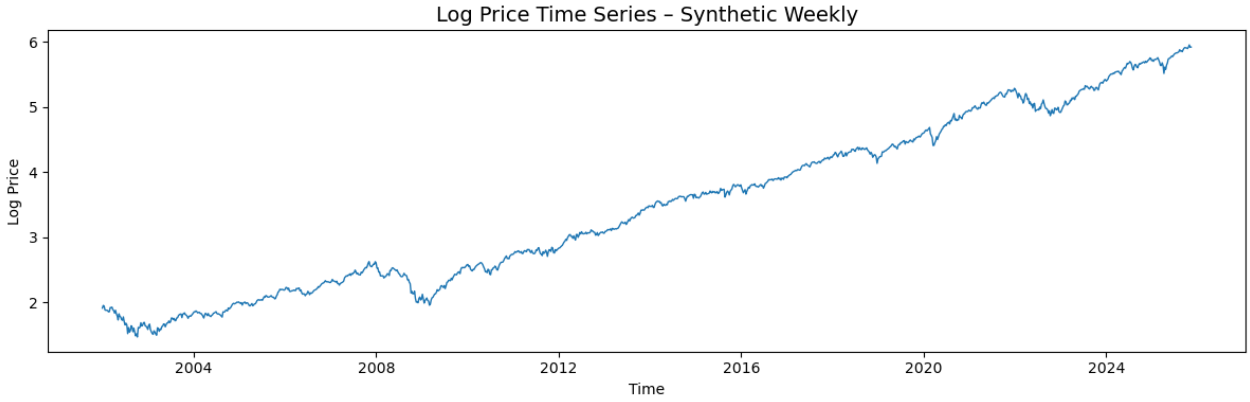


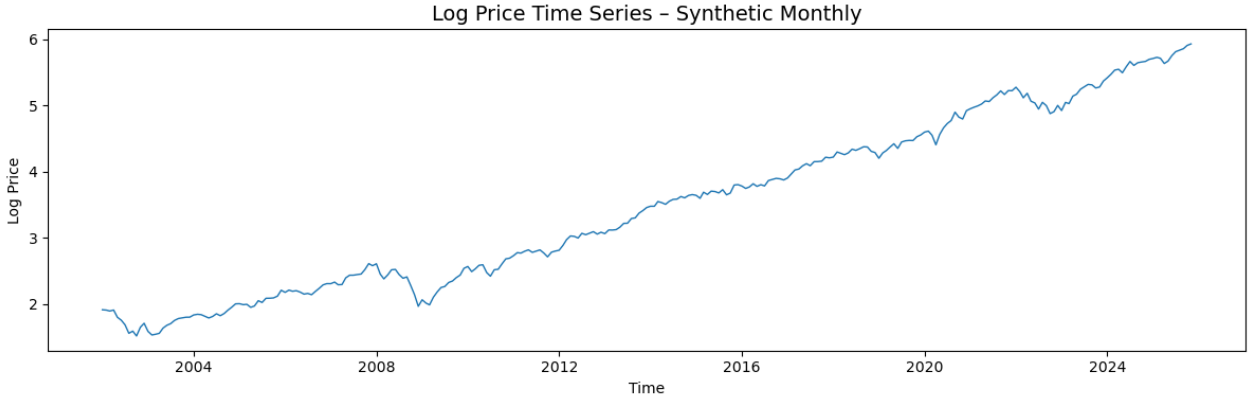

Log Price Time Series – Daily
Log Price
Time
10.0
9.5
9.0
8.5
8.0
7.5
7.0
2004
2008
2012
2016
2020
2024


Log Price Time Series – Weekly
Log Price
Time
10.0
9.5
9.0
8.5
8.0
7.5
7.0
2004
2008
2012
2016
2020
2024


Log Price Time Series – Monthly
Log Price
Time
10.0
9.5
9.0
8.5
8.0
7.5
7.0
2004
2008
2012
2016
2020
2024


Log Price Time Series – Synthetic Daily
Log Price
Time
6
5
4
3
2
2004
2008
2012
2016
2020
2024



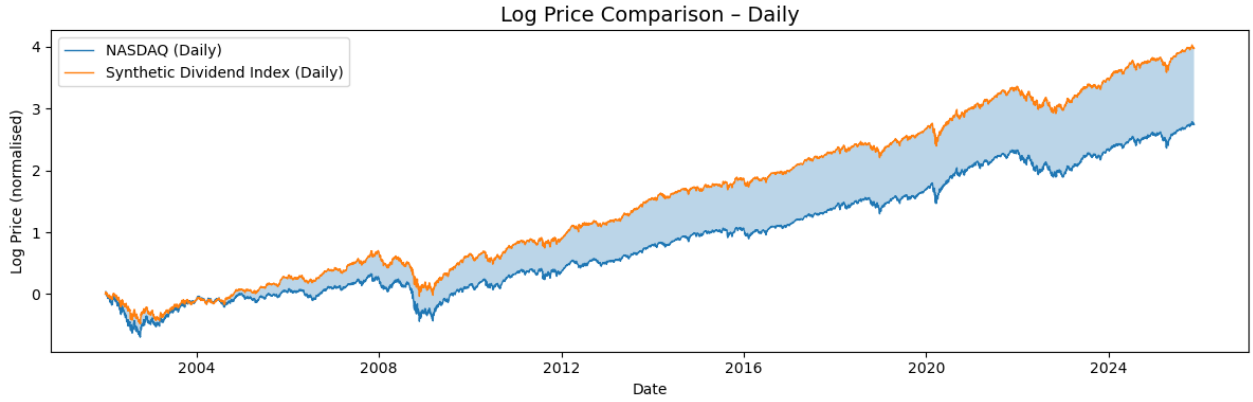


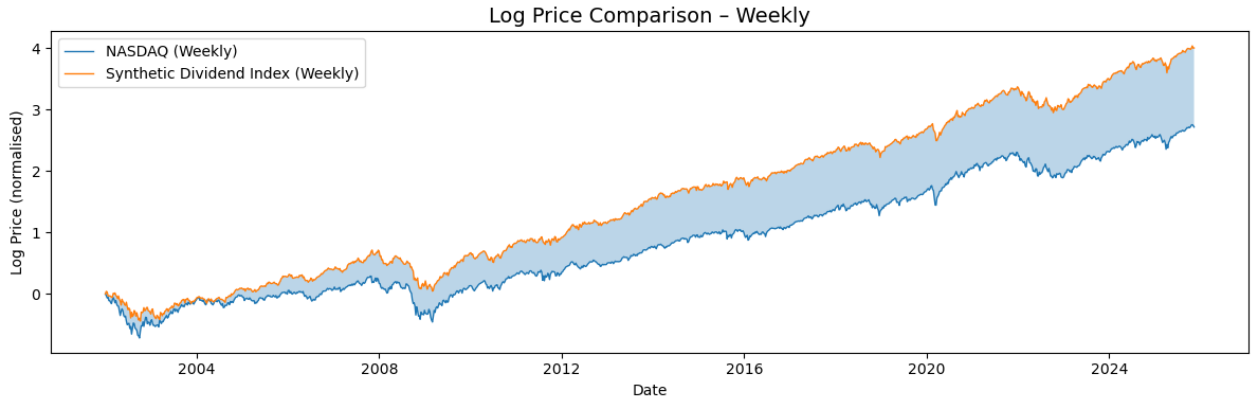
In [ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# ================================================================
# 1. BASIC HELPER: get a clean 1D Series from Series/DataFrame
# ================================================================

def _to_series(obj, col=None):
    """
    Convert a Series/DataFrame to a 1D Series, dropping NaNs.
    """
    if isinstance(obj, pd.Series):
        s = obj
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns; please
            s = obj.iloc[:, 0]
        else:
            if col not in obj.columns:
                raise KeyError(
                    f"Column '{col}' not found in DataFrame. "
                    f"Available: {list(obj.columns)}"
                )
            s = obj[col]
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")
    return s.dropna()
```

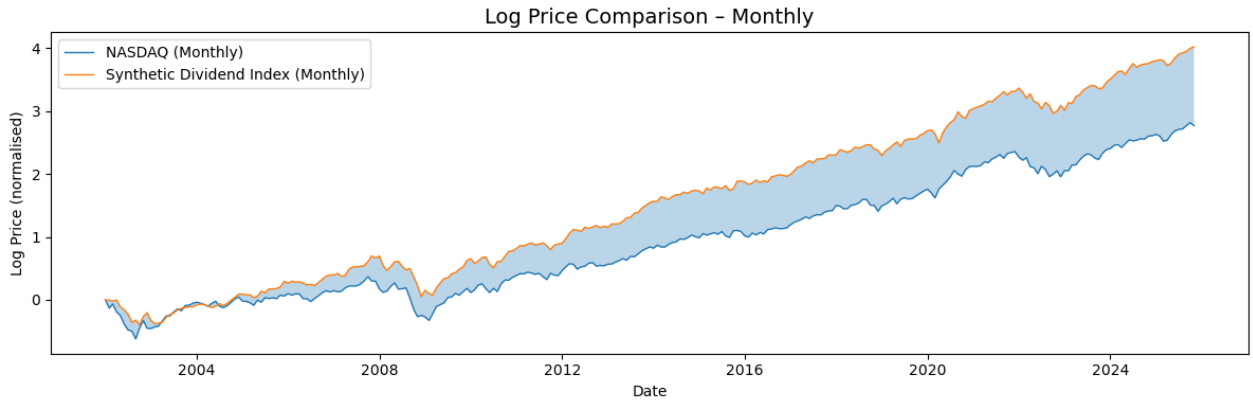
```python
# ========================================================
# 2. HELPER: align any time index by *calendar date only*
# ========================================================

def _to_daily_series(obj, col=None):
    """
    Take a Series/DataFrame with ANY kind of time index (tz-aware, tz-nai
    strings with offsets, etc.) and return a Series indexed by calendar D
    If there are multiple observations per date, keep the last one.
    """
    s = _to_series(obj, col=col).copy()

    # Robust conversion: handle tz-aware and tz-naive uniformly
    # utc=True makes tz-aware safe; then we drop tz and keep just dates
    idx = pd.to_datetime(s.index, utc=True, errors="coerce")

    # Drop anything that couldn't be parsed as datetime
    mask = ~idx.isna()
    s = s[mask]
    idx = idx[mask]

    # Use calendar dates as index
    date_index = pd.Index(idx.date, name="date")
    s = pd.Series(s.values, index=date_index)

    # If multiple obs per date, keep the last
    s = s.groupby(level=0).last()

    return s


# ========================================
# 3. PLOT COMPARISON WITH SHADED DIFFERENCE
# ========================================

def plot_log_price_comparison(
    obj1,
    obj2,
    label1,
    label2,
    title,
    col1=None,
    col2=None,
    normalize=True
):
    """
    Plot two log price series on the same figure for comparison,
    with shaded area between them.

    Alignment is done by calendar DATE (not exact timestamp),
    so it is robust to time zones and different intraday times.
    """
    # Convert both to daily series on date index
    s1 = _to_daily_series(obj1, col=col1)
```

```
    s2 = _to_daily_series(obj2, col=col2)

    # Align on common calendar dates
    df = pd.concat([s1, s2], axis=1, join="inner")
    df.columns = [label1, label2]
    df = df.dropna()

    if df.empty:
        raise ValueError(
            "No overlapping calendar dates between the two series after a
            "Check that they actually cover some common period."
        )

    # Optionally normalise so both start at 0 (relative performance)
    if normalize:
        df = df - df.iloc[0]

    plt.figure(figsize=(12, 4))

    # Plot the two lines
    plt.plot(df.index, df[label1], linewidth=1, label=label1)
    plt.plot(df.index, df[label2], linewidth=1, label=label2)

    # Shade the difference between them
    plt.fill_between(
        df.index,
        df[label1],
        df[label2],
        alpha=0.3
    )

    plt.title(f"Log Price Comparison — {title}", fontsize=14)
    plt.xlabel("Date")
    plt.ylabel("Log Price (normalised)" if normalize else "Log Price")
    plt.legend()
    plt.tight_layout()
    plt.show()


# ============================================
# 4. EXAMPLE CALLS (ASSUMES THESE OBJECTS EXIST):
#    nasdaq_*_log_prices and synthetic_*_log_prices
# ============================================

# Daily
plot_log_price_comparison(
    nasdaq_daily_log_prices,
    synthetic_daily_log_prices,
    label1="NASDAQ (Daily)",
    label2="Synthetic Dividend Index (Daily)",
    title="Daily"
)

# Weekly
```

```
plot_log_price_comparison(
    nasdaq_weekly_log_prices,
    synthetic_weekly_log_prices,
    label1="NASDAQ (Weekly)",
    label2="Synthetic Dividend Index (Weekly)",
    title="Weekly"
)

# Monthly
plot_log_price_comparison(
    nasdaq_monthly_log_prices,
    synthetic_monthly_log_prices,
    label1="NASDAQ (Monthly)",
    label2="Synthetic Dividend Index (Monthly)",
    title="Monthly"
)
```







In [ ]:
```
print(type(nasdaq_daily_log_prices.index))
```

```
print(type(synthetic_daily_log_prices.index))

print(nasdaq_daily_log_prices.index.min(), nasdaq_daily_log_prices.index.
print(synthetic_daily_log_prices.index.min(), synthetic_daily_log_prices.
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
<class 'pandas.core.indexes.base.Index'>
2002-01-02 00:00:00 2025-11-14 00:00:00
2002-01-01 00:00:00-05:00 2025-11-15 00:00:00-05:00
```

Dividend Extraction

In [ ]:
```
Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")

# ================================
# Generic loader for dividends & risk-free (robust column detection)
# ================================
def load_series_with_logs(csv_filename, value_col, date_cols=('Date','Dat
    """
    Robust loader that:
      - parses 'Date' or 'Datetime' as index (if present)
      - detects the value column flexibly (accepts exact, list of names,
      - coerces to numeric
      - computes Log_Value and Log_Returns while avoiding log(0)/inf
    """
    path = os.path.join(Data_dir, csv_filename)
    df = pd.read_csv(path)

    # ---- Handle date parsing ----
    for col in date_cols:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    if 'Date' in df.columns and not df['Date'].isna().all():
        df.set_index('Date', inplace=True)
    elif 'Datetime' in df.columns and not df['Datetime'].isna().all():
        df.set_index('Datetime', inplace=True)
    else:
        raise ValueError(f"{csv_filename} missing Date/Datetime. Found {l

    # ---- Flexible detection of value column ----
    # allow value_col to be a single name or an iterable of candidates
    if isinstance(value_col, (list, tuple, set)):
        candidates = list(value_col)
    else:
        candidates = [value_col]

    def _norm(s):
        return str(s).lower().replace('–', '-').replace('—', '-').strip()

    cols_norm = { _norm(c): c for c in df.columns }

    chosen_col = None
```

```
    # try exact (normalized) matches first
    for cand in candidates:
        nc = _norm(cand)
        if nc in cols_norm:
            chosen_col = cols_norm[nc]
            break

    # fallback: look for obvious keywords if exact not found
    if chosen_col is None:
        keywords = ['yield', 'treasury', '1-month', '1 month', '1m', 'rat
        for col in df.columns:
            lc = str(col).lower()
            if any(kw in lc for kw in keywords):
                chosen_col = col
                break

    # last resort: pick the single numeric column (if only one exists)
    if chosen_col is None:
        numeric_cols = df.select_dtypes(include=[np.number]).columns.toli
        if len(numeric_cols) == 1:
            chosen_col = numeric_cols[0]

    if chosen_col is None:
        raise KeyError(
            f"Could not find value column for '{value_col}' in {csv_filen
            f"Available columns: {list(df.columns)}"
        )

    if chosen_col != value_col:
        # small informative message (won't break notebooks); change to pr
        print(f"[load_series_with_logs] Using column '{chosen_col}' for '

    # ---- Ensure numeric ----
    df[chosen_col] = pd.to_numeric(df[chosen_col], errors='coerce')

    # ---- Compute logs and returns avoiding log(0) and infs ----
    series = df[chosen_col]
    series_nonzero = series.replace(0, np.nan)

    df['Log_Value'] = np.log(series_nonzero)  # log of level, NaN if zero

    # compute log-returns safely: if previous is zero/missing, result is
    prev = series_nonzero.shift(1)
    with np.errstate(divide='ignore', invalid='ignore'):
        lr = np.log(series_nonzero / prev)

    # replace infinities with NaN
    lr = pd.Series(lr, index=df.index).replace([np.inf, -np.inf], np.nan)
    df['Log_Returns'] = lr

    return df


# ================================
```

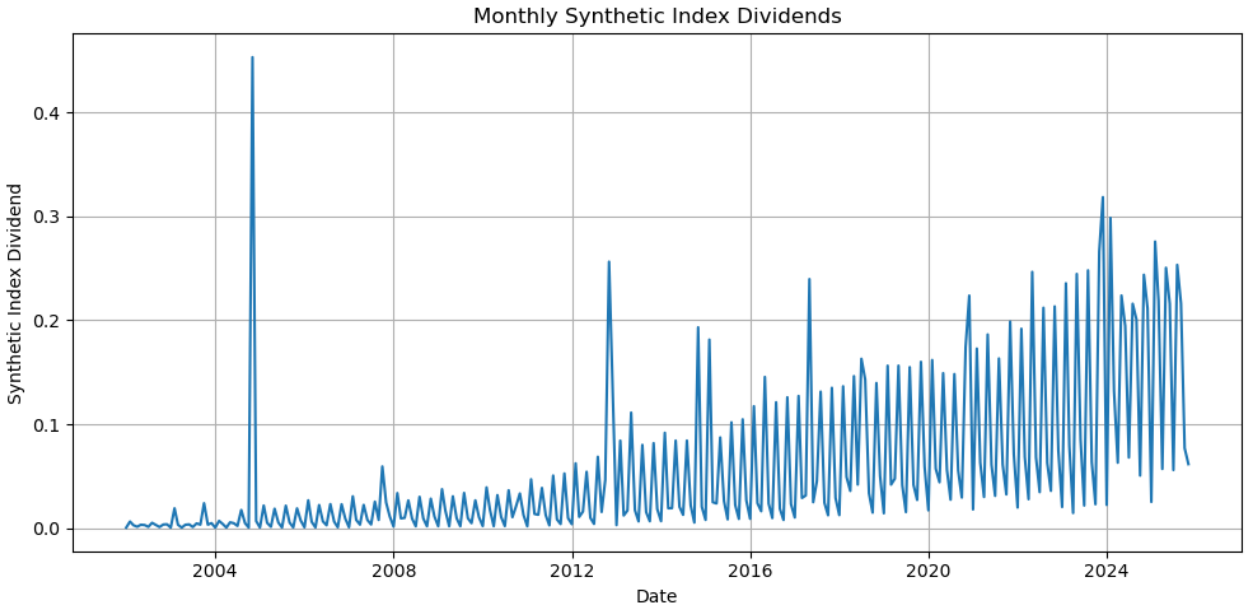
```
# Dividend loaders
# ================================
synthetic_div_daily_df   = load_series_with_logs(
    "synthetic_div_daily_df.csv",
    "Synthetic Index Dividend"
)
synthetic_div_weekly_df  = load_series_with_logs(
    "synthetic_div_weekly_df.csv",
    "Synthetic Index Dividend"
)
synthetic_div_monthly_df = load_series_with_logs(
    "synthetic_div_monthly_df.csv",
    "Synthetic Index Dividend"
)

# Extract ready-to-use series
synthetic_div_daily_log_div      = synthetic_div_daily_df['Log_Returns'].
synthetic_div_weekly_log_div     = synthetic_div_weekly_df['Log_Returns']
synthetic_div_monthly_log_div    = synthetic_div_monthly_df['Log_Returns'

synthetic_div_daily_log_levels   = synthetic_div_daily_df['Log_Value'].dr
synthetic_div_weekly_log_levels  = synthetic_div_weekly_df['Log_Value'].d
synthetic_div_monthly_log_levels = synthetic_div_monthly_df['Log_Value'].


# ================================
# Risk-free loaders
# ================================
risk_free_daily_df = load_series_with_logs(
    "risk_free_daily_df.csv",
    "1-month Yield - US Treasury Securities"
)
risk_free_weekly_df = load_series_with_logs(
    "risk_free_weekly_df.csv",
    "1-month Yield - US Treasury Securities"
)
risk_free_monthly_df = load_series_with_logs(
    "risk_free_monthly_df.csv",
    "1-month Yield - US Treasury Securities"
)

risk_free_daily_log_rf   = risk_free_daily_df['Log_Value'].dropna()
risk_free_weekly_log_rf  = risk_free_weekly_df['Log_Value'].dropna()
risk_free_monthly_log_rf = risk_free_monthly_df['Log_Value'].dropna()
```

```
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_daily_df.csv' (requested '1-month Yield - US Treasury S
ecurities').
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_weekly_df.csv' (requested '1-month Yield - US Treasury
Securities').
[load_series_with_logs] Using column '1-month Yield - US Treasury Securiti
es' for 'risk_free_monthly_df.csv' (requested '1-month Yield - US Treasury
Securities').
```

```
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_38889/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_38889/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
/var/folders/k1/mhyzg0w15gb04d0x6q7gdp2w0000gn/T/ipykernel_38889/127600054
2.py:21: FutureWarning: In a future version of pandas, parsing datetimes w
ith mixed time zones will raise an error unless `utc=True`. Please specify
`utc=True` to opt in to the new behaviour and silence this warning. To cre
ate a `Series` with mixed offsets and `object` dtype, please use `apply` a
nd `datetime.datetime.strptime`
  df[col] = pd.to_datetime(df[col], errors='coerce')
```

In [ ]:

```python
# Plot Monthly Synthetic Dividends (robust column selection)
div_col = "Synthetic Index Dividend" if "Synthetic Index Dividend" in syn

plt.figure(figsize=(10, 5))
plt.plot(
    synthetic_div_monthly_df.index,
    synthetic_div_monthly_df[div_col],
    label="Monthly Dividends"
)
plt.title("Monthly Synthetic Index Dividends")
plt.xlabel("Date")
plt.ylabel(div_col)
plt.grid(True)
plt.tight_layout()
plt.show()

rf_candidates = [c for c in risk_free_monthly_df.columns if any(k in c.lo
rf_col = rf_candidates[0] if rf_candidates else risk_free_monthly_df.colu

plt.figure(figsize=(10, 5))
plt.plot(
    risk_free_monthly_df.index,
    risk_free_monthly_df[rf_col],
    label="Monthly Risk-Free Rate"
)
plt.title(f"Monthly Risk-Free Rate ({rf_col})")
plt.xlabel("Date")
plt.ylabel(rf_col)
plt.grid(True)
plt.tight_layout()
plt.show()
```

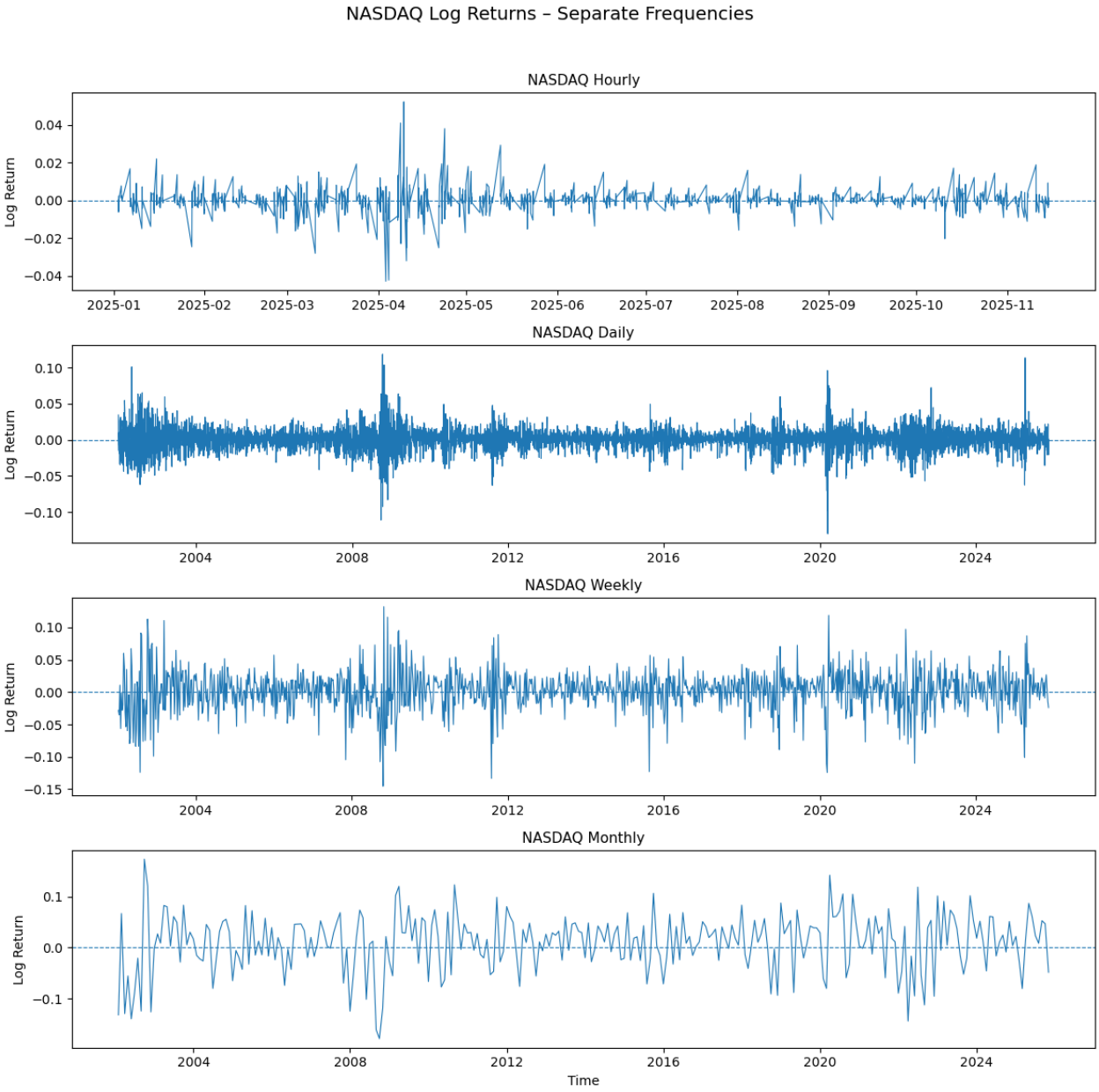


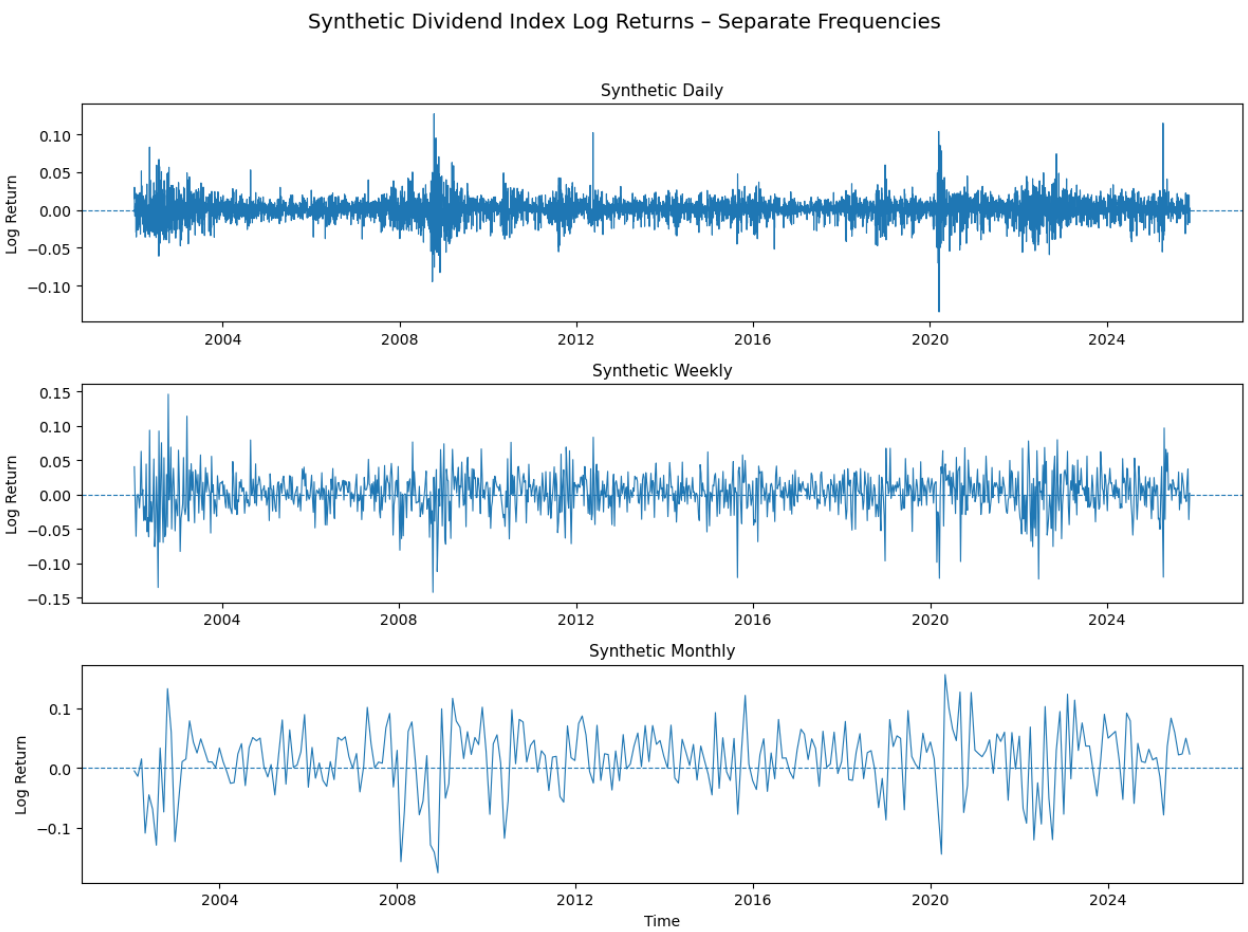
Log Returns - Prices 1st Difference PLot --> WN plot

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

def _to_series(obj, col=None):
    """
    Convert a Series/DataFrame to a 1D Series, dropping NaNs.
    """
    if isinstance(obj, pd.Series):
        s = obj
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns; please
            s = obj.iloc[:, 0]
        else:
            if col not in obj.columns:
```

```
                raise KeyError(
                    f"Column '{col}' not found in DataFrame. "
                    f"Available: {list(obj.columns)}"
                )
            s = obj[col]
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")
    return s.dropna()


def plot_log_returns_grid(series_dict, figure_title, cols=None):
    """
    Plot each return series in `series_dict` in its own subplot,
    all within a single figure.

    series_dict : dict[label] -> Series/DataFrame
    cols        : optional dict[label] -> colname (if some objs are DataF
    """
    n = len(series_dict)
    fig, axes = plt.subplots(n, 1, figsize=(12, 3 * n), sharex=False)
    if n == 1:
        axes = [axes]

    for ax, (label, obj) in zip(axes, series_dict.items()):
        col = None
        if cols is not None and label in cols:
            col = cols[label]
        y = _to_series(obj, col=col)

        ax.plot(y.index, y.values, linewidth=0.8)
        ax.axhline(0, linestyle="--", linewidth=0.8)
        ax.set_title(label, fontsize=11)
        ax.set_ylabel("Log Return")

    axes[-1].set_xlabel("Time")
    fig.suptitle(figure_title, fontsize=14)
    fig.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

nasdaq_series = {
    "NASDAQ Hourly":  nasdaq_hourly_log_returns,
    "NASDAQ Daily":   nasdaq_daily_log_returns,
    "NASDAQ Weekly":  nasdaq_weekly_log_returns,
    "NASDAQ Monthly": nasdaq_monthly_log_returns,
}

plot_log_returns_grid(
    nasdaq_series,
    figure_title="NASDAQ Log Returns — Separate Frequencies"
)

synthetic_series = {
    "Synthetic Daily":   synthetic_daily_log_returns,
    "Synthetic Weekly":  synthetic_weekly_log_returns,
```

```
    "Synthetic Monthly": synthetic_monthly_log_returns,
}

plot_log_returns_grid(
    synthetic_series,
    figure_title="Synthetic Dividend Index Log Returns — Separate Frequen
)
```

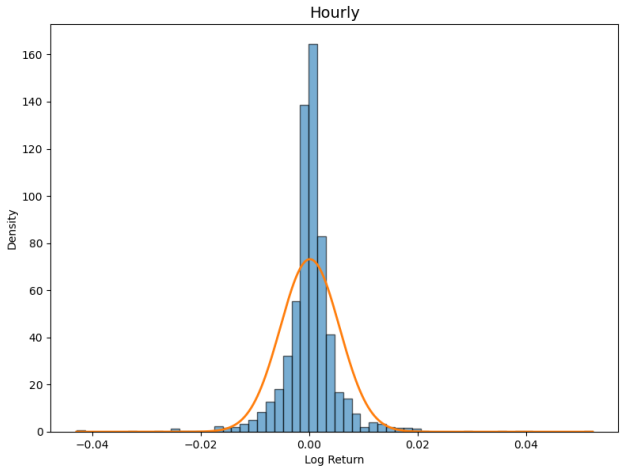


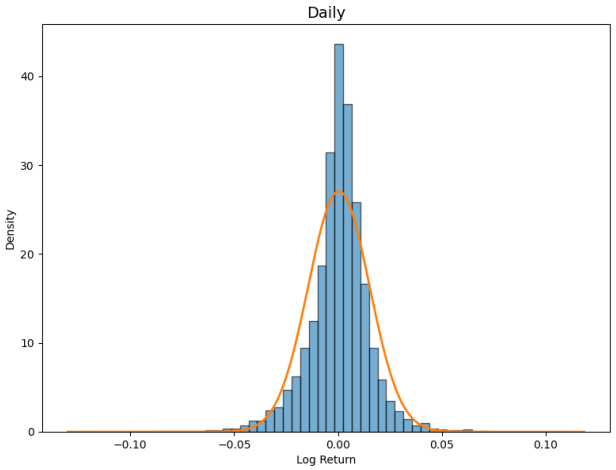


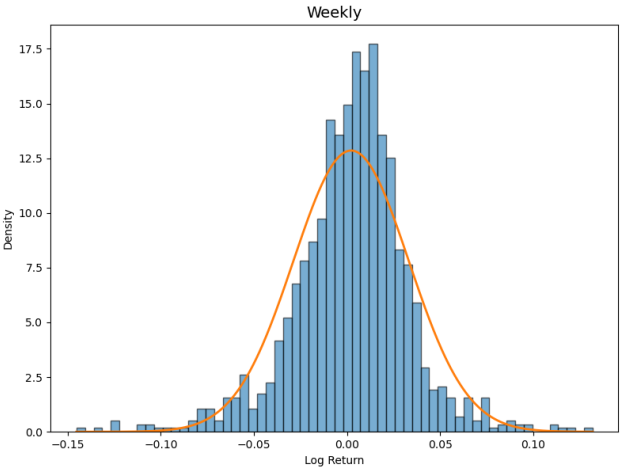
Histogram of Returns + Normal PDF Overlay

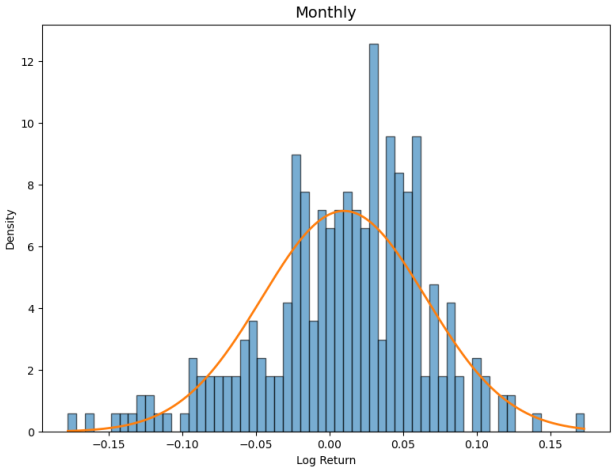
In [ ]:
```python
def plot_histograms_grid():
    import matplotlib.pyplot as plt
    import numpy as np
    from scipy.stats import norm
    import pandas as pd

    datasets = [
        (nasdaq_hourly_log_returns,  "Hourly"),
        (nasdaq_daily_log_returns,   "Daily"),
        (nasdaq_weekly_log_returns,  "Weekly"),
        (nasdaq_monthly_log_returns, "Monthly"),
        (synthetic_daily_log_returns, "Synthetic Daily"),
        (synthetic_weekly_log_returns, "Synthetic Weekly"),
        (synthetic_monthly_log_returns, "Synthetic Monthly")
    ]

    # 2 columns, 4 rows → last slot unused
    fig, axes = plt.subplots(4, 2, figsize=(16, 24))
    axes = axes.flatten()

    for i, (obj, title) in enumerate(datasets):

        ax = axes[i]
```

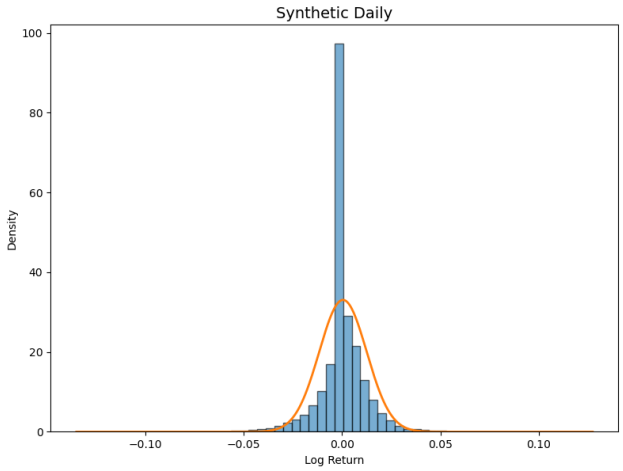
```
        # Extract series
        if isinstance(obj, pd.DataFrame):
            r = obj[obj.columns[0]].dropna()
        else:
            r = obj.dropna()

        # Fit normal distribution
        mu, sigma = r.mean(), r.std()

        # Histogram
        ax.hist(r, bins=60, density=True, alpha=0.6, edgecolor="black")

        # Normal PDF overlay
        x_vals = np.linspace(r.min(), r.max(), 500)
        pdf_vals = norm.pdf(x_vals, mu, sigma)
        ax.plot(x_vals, pdf_vals, linewidth=2)

        ax.set_title(f"{title}", fontsize=14)
        ax.set_xlabel("Log Return")
        ax.set_ylabel("Density")

    # Hide the last unused subplot
    axes[-1].axis("off")

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

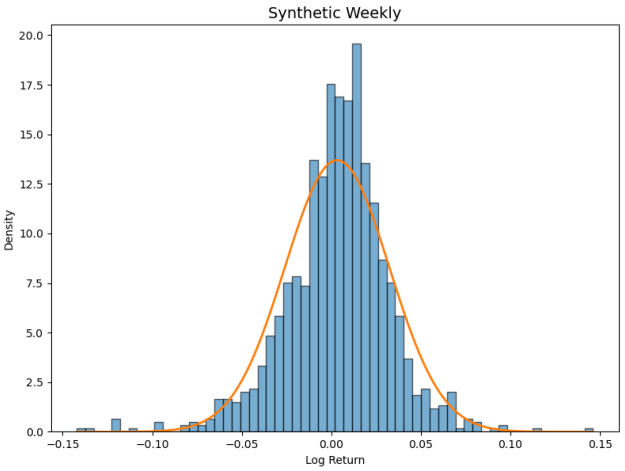

Hourly
Density
Log Return

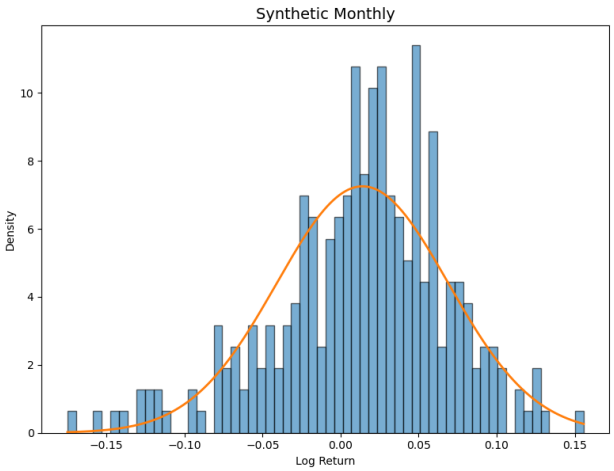

Daily
Density
Log Return


Weekly
Density
Log Return


Monthly
Density
Log Return

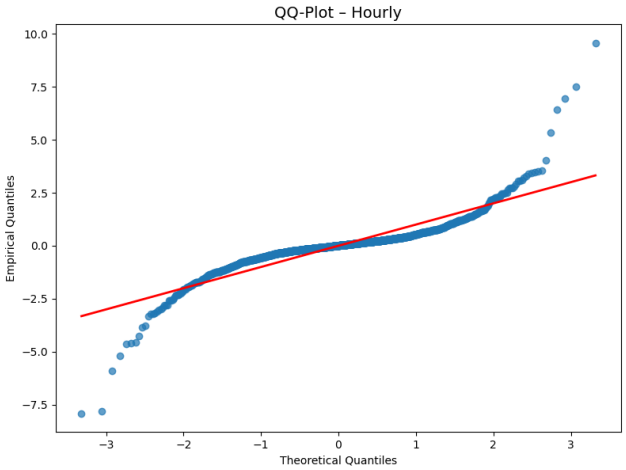

Synthetic Daily
Density
Log Return

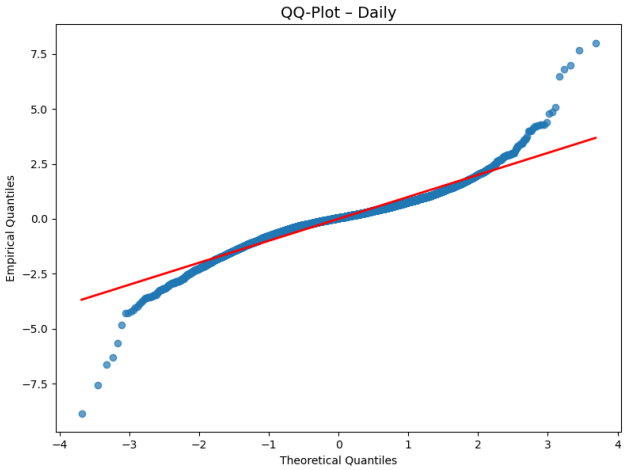

Synthetic Weekly
Density
Log Return

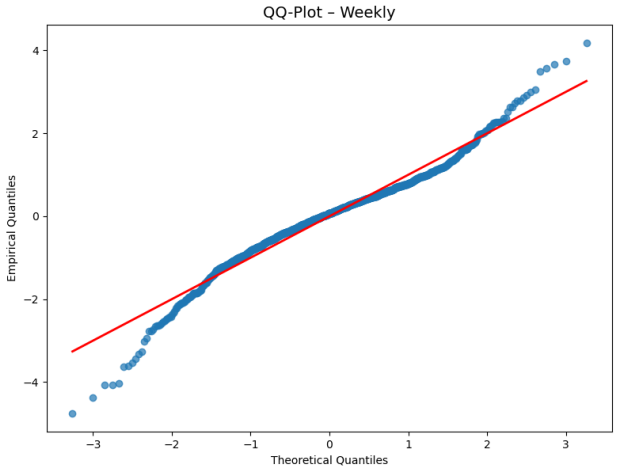

Synthetic Monthly
Density
Log Return

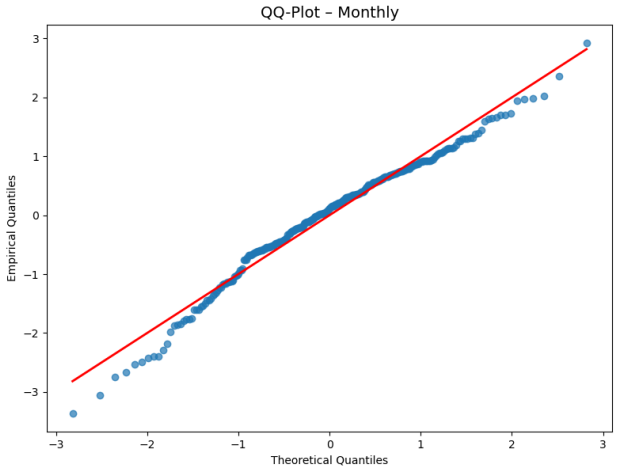
QQ Plot Returns

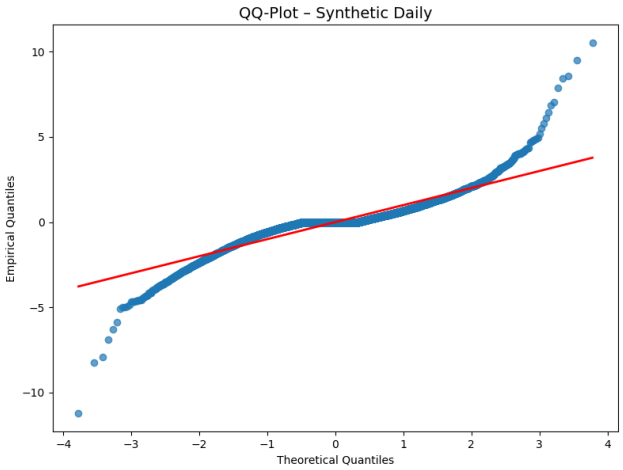
In [ ]:
```python
def plot_all_return_qq():
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    from scipy import stats

    datasets = [
        (nasdaq_hourly_log_returns,  "Hourly"),
        (nasdaq_daily_log_returns,   "Daily"),
        (nasdaq_weekly_log_returns,  "Weekly"),
        (nasdaq_monthly_log_returns, "Monthly"),
        (synthetic_daily_log_returns, "Synthetic Daily"),
        (synthetic_weekly_log_returns, "Synthetic Weekly"),
        (synthetic_monthly_log_returns, "Synthetic Monthly")
    ]

    fig, axes = plt.subplots(4, 2, figsize=(16, 24))
    axes = axes.flatten()

    for i, (obj, title) in enumerate(datasets):

        ax = axes[i]

        # Extract series
        if isinstance(obj, pd.DataFrame):
            r = obj[obj.columns[0]].dropna()
        else:
            r = obj.dropna()

        # Standardize the returns
        r_std = (r - r.mean()) / r.std()

        # QQ-plot manually (so we can use ax)
        (theoretical_q, sample_q), _ = stats.probplot(r_std, dist="norm")

        ax.scatter(theoretical_q, sample_q, alpha=0.7)
        ax.plot(theoretical_q, theoretical_q, color="red", linewidth=2)

        ax.set_title(f"QQ-Plot — {title}", fontsize=14)
        ax.set_xlabel("Theoretical Quantiles")
        ax.set_ylabel("Empirical Quantiles")

    # Hide unused final subplot
    axes[-1].axis("off")

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

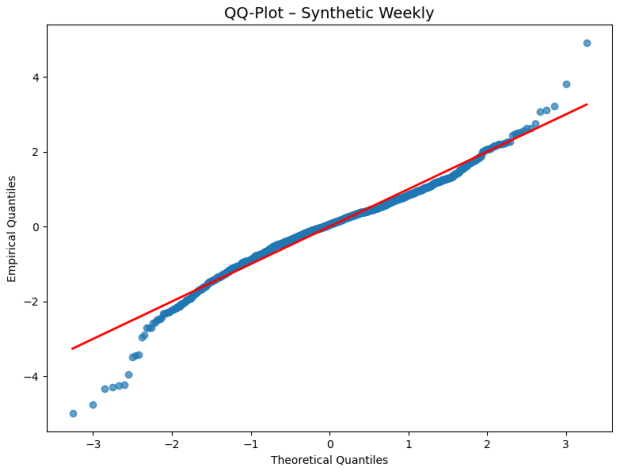

QQ-Plot – Hourly
Empirical Quantiles
Theoretical Quantiles

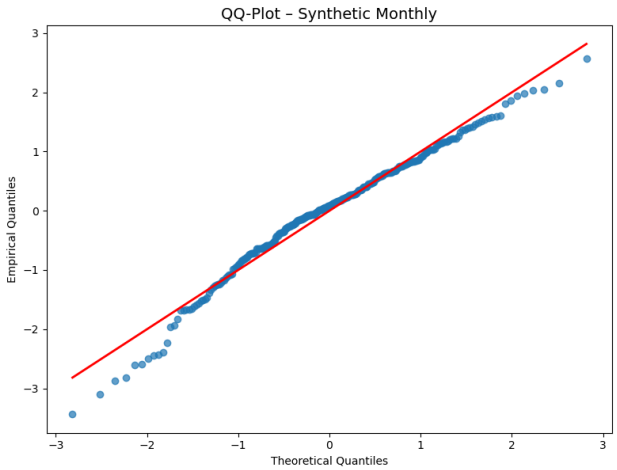

QQ-Plot – Daily
Empirical Quantiles
Theoretical Quantiles


QQ-Plot – Weekly
Empirical Quantiles
Theoretical Quantiles


QQ-Plot – Monthly
Empirical Quantiles
Theoretical Quantiles


QQ-Plot – Synthetic Daily
Empirical Quantiles
Theoretical Quantiles


QQ-Plot – Synthetic Weekly
Empirical Quantiles
Theoretical Quantiles

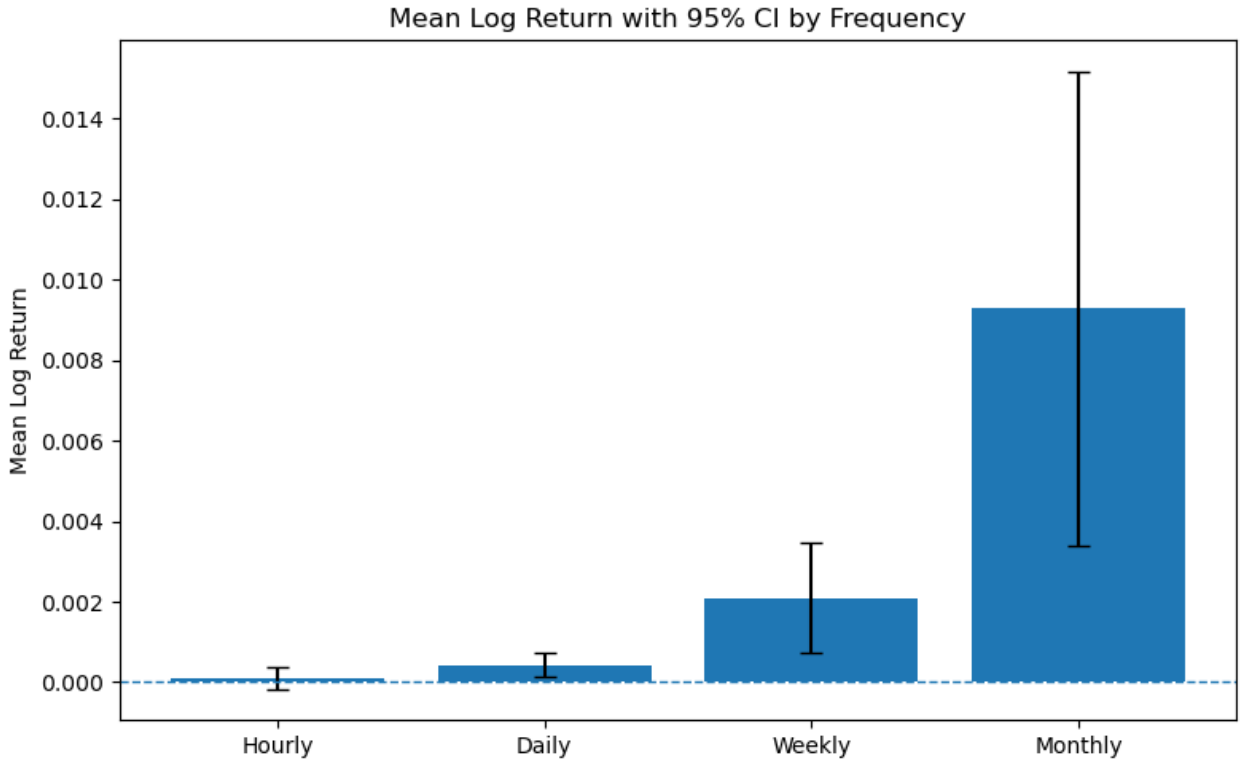

QQ-Plot – Synthetic Monthly
Empirical Quantiles
Theoretical Quantiles

Mean Return with 95% CI Plot (Bar Plot)

In [ ]:
```
def extract_series(obj, col=None):
    """
    Helper: take a Series or DataFrame and return a 1D Series of returns.
    """
    if isinstance(obj, pd.Series):
        return obj.dropna()
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns, specify
            return obj.iloc[:, 0].dropna()
        else:
            return obj[col].dropna()
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")

def mean_and_ci_95(series):
    """
    Return sample mean and symmetric 95% CI half-width using normal appro
    """
    series = series.dropna()
    n = len(series)
    mu = series.mean()
    sigma = series.std(ddof=1)
    se = sigma / np.sqrt(n)
    ci_half = 1.96 * se          # 95% CI half-width
    return mu, ci_half

# ---- Prepare your frequencies ----
# If your objects are DataFrames with a specific column, add col="Log_Ret
freq_data = {
    "Hourly":  extract_series(nasdaq_hourly_log_returns),
    "Daily":   extract_series(nasdaq_daily_log_returns),
    "Weekly":  extract_series(nasdaq_weekly_log_returns),
    "Monthly": extract_series(nasdaq_monthly_log_returns),
}

freqs = []
means = []
ci_halfs = []

for name, ser in freq_data.items():
    mu, ci_h = mean_and_ci_95(ser)
    freqs.append(name)
    means.append(mu)
    ci_halfs.append(ci_h)

# ---- Bar plot with 95% CI ----
x = np.arange(len(freqs))

4
```

```
NASDAQ
--
Synthetic
Synthetic > NASDAQ
Log Prices — NASDAQ vs Synthetic ({title})
Time
Log Price
language
source
    """
    Helper: take a Series or DataFrame and return a 1D Series of returns.
    """
    if isinstance(obj, pd.Series):
        return obj.dropna()
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns, specify
            return obj.iloc[:, 0].dropna()
        else:
            return obj[col].dropna()
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")

def plot_sign_chart(obj, title, col=None, max_points=3000):
    """
    Sequential sign chart for Cowles–Jones type runs analysis.

    obj : Series or DataFrame of returns
    col : column name if obj is a DataFrame with multiple columns
    max_points : if series is very long, truncate to last max_points obs
    """
    r = extract_series(obj, col=col)

    # Optionally limit length (to keep plot readable)
    if len(r) > max_points:
        r = r.iloc[-max_points:]

    # Map returns to signs: +1, -1, 0
    sign = np.sign(r)
    # For clarity, keep three levels: -1 (negative), 0 (zero), +1 (positi
    x = np.arange(len(sign))

    plt.figure(figsize=(12, 3))
    # Strip / dot plot
    plt.scatter(x[sign > 0], sign[sign > 0], marker="+", s=20, label="Pos
    plt.scatter(x[sign < 0], sign[sign < 0], marker="_", s=20, label="Neg
    if (sign == 0).any():
        plt.scatter(x[sign == 0], sign[sign == 0], marker="o", s=10, labe

    plt.yticks([-1, 0, 1], ["-", "0", "+"])
    plt.ylim(-1.5, 1.5)
    plt.xlabel("Time (sequence index)")
    plt.ylabel("Sign")
```

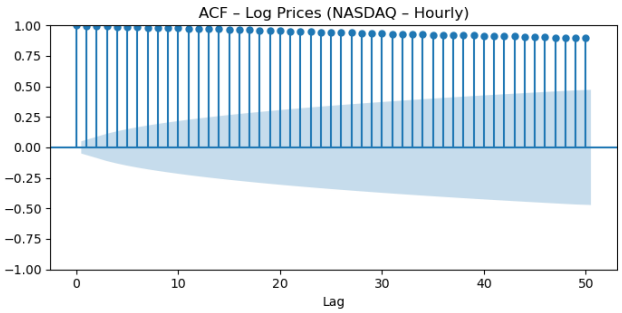
```
    plt.title(f"Sequential Sign Chart of Returns – {title}")
    plt.legend(loc="upper right")
    plt.tight_layout()
    plt.show()


plot_sign_chart(nasdaq_hourly_log_returns,   "NASDAQ – Hourly",  col="Log
plot_sign_chart(nasdaq_daily_log_returns,    "NASDAQ – Daily",   col="Log
plot_sign_chart(nasdaq_weekly_log_returns,   "NASDAQ – Weekly",  col="Log
plot_sign_chart(nasdaq_monthly_log_returns,  "NASDAQ – Monthly", col="Log

plot_sign_chart(synthetic_daily_log_returns,   "Synthetic – Daily",   col
plot_sign_chart(synthetic_weekly_log_returns,  "Synthetic – Weekly",  col
plot_sign_chart(synthetic_monthly_log_returns, "Synthetic – Monthly", col
```

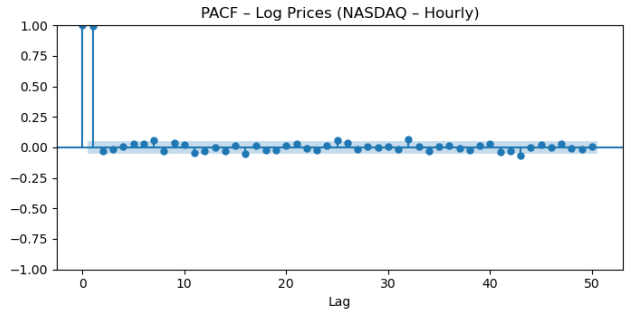


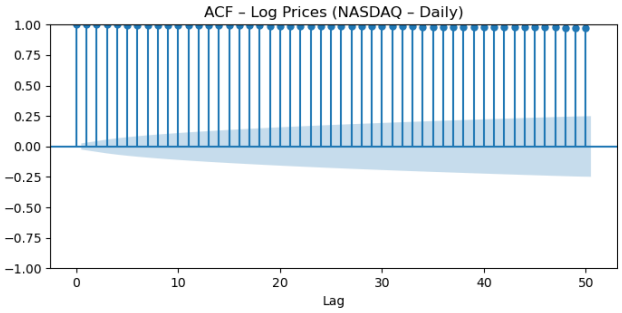
RW2 - Autocorrelations structure of log prices

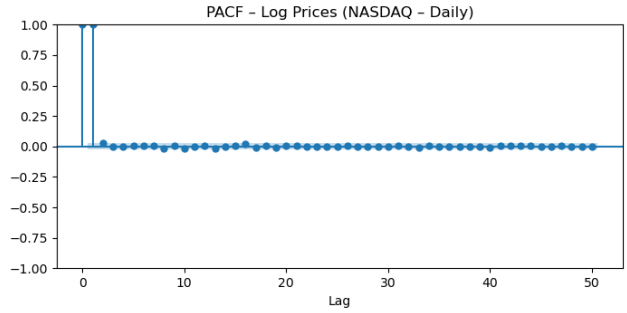
In [70]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

def extract_series(obj, col=None):
    """
    Return a 1D Series from either a Series or a DataFrame.

    - If obj is a DataFrame: use the column 'col' (or the only column if
    - If obj is a Series: ignore 'col' and just return the Series.
    """
    # DataFrame case
    if isinstance(obj, pd.DataFrame):
        if col is not None:
```

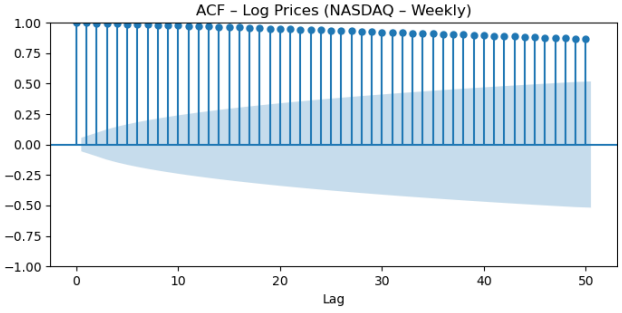
```
            return obj[col].dropna()
        # if no column specified, use the first one
        return obj.iloc[:, 0].dropna()

    # Series (or array-like) case
    return pd.Series(obj).dropna()


def plot_all_price_acf_pacf(pairs, lags=50):
    """
    Create ONE big figure with separate ACF/PACF panels for each frequenc

    pairs : list of tuples (data, title, colname)
    lags  : number of lags for ACF/PACF
    """
    n = len(pairs)
    fig, axes = plt.subplots(n, 2, figsize=(14, 3.5 * n))

    if n == 1:
        axes = axes.reshape(1, 2)

    for i, (obj, title, col) in enumerate(pairs):
        y = extract_series(obj, col)

        # ACF
        plot_acf(y, ax=axes[i, 0], lags=lags)
        axes[i, 0].set_title(f"ACF — Log Prices ({title})")
        axes[i, 0].set_xlabel("Lag")

        # PACF
        plot_pacf(y, ax=axes[i, 1], lags=lags, method="ywm")
        axes[i, 1].set_title(f"PACF — Log Prices ({title})")
        axes[i, 1].set_xlabel("Lag")

    plt.tight_layout()
    plt.show()


# ------------------------------------------------------------
# Call the unified plotting function
# ------------------------------------------------------------

plot_all_price_acf_pacf(
    [
        (nasdaq_hourly_log_prices,    "NASDAQ — Hourly",   "Log_Prices"),
        (nasdaq_daily_log_prices,     "NASDAQ — Daily",    "Log_Prices"),
        (nasdaq_weekly_log_prices,    "NASDAQ — Weekly",   "Log_Prices"),
        (nasdaq_monthly_log_prices,   "NASDAQ — Monthly",  "Log_Prices"),
        (synthetic_daily_log_prices,   "Synthetic — Daily",   "Log_Prices
        (synthetic_weekly_log_prices,  "Synthetic — Weekly",  "Log_Prices
        (synthetic_monthly_log_prices, "Synthetic — Monthly", "Log_Prices
    ],
    lags=50
)
```

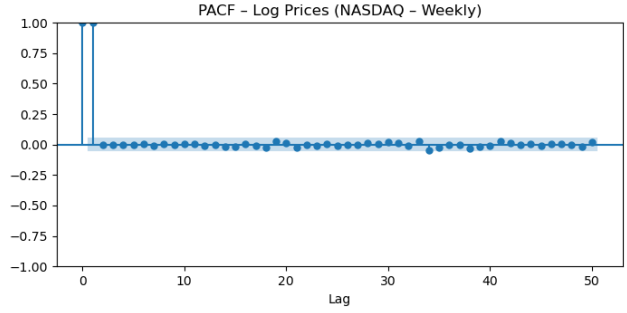

ACF – Log Prices (NASDAQ – Hourly)
Lag

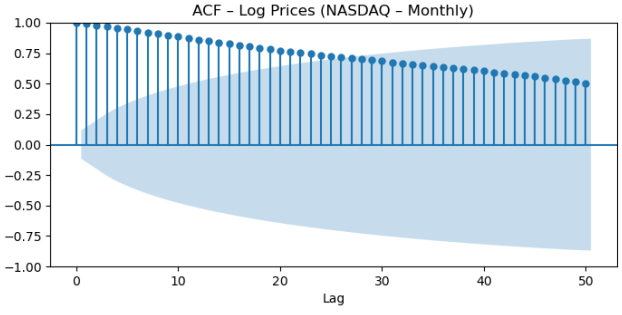

PACF – Log Prices (NASDAQ – Hourly)
Lag

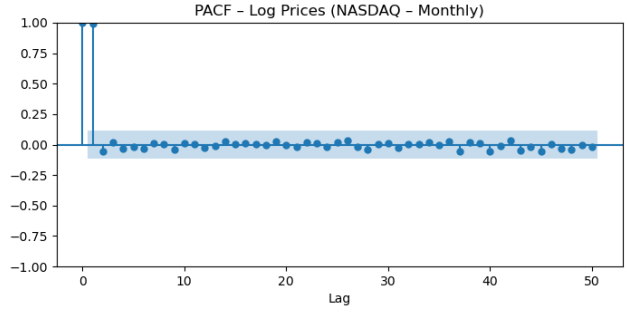

ACF – Log Prices (NASDAQ – Daily)
Lag

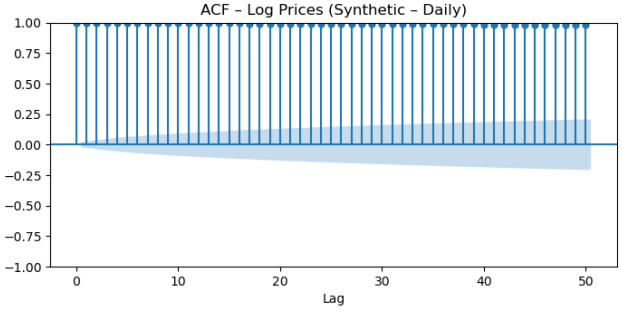

PACF – Log Prices (NASDAQ – Daily)
Lag

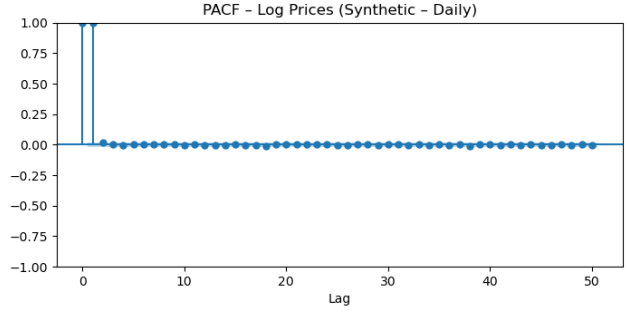

ACF – Log Prices (NASDAQ – Weekly)
Lag

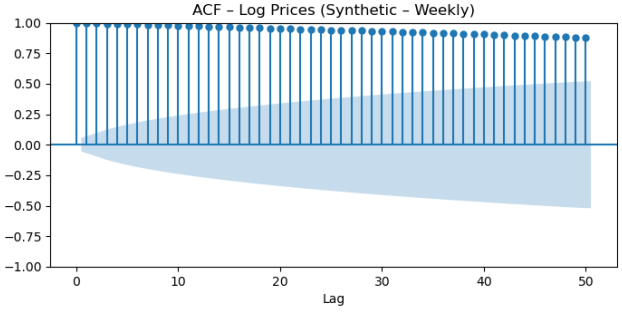

PACF – Log Prices (NASDAQ – Weekly)
Lag

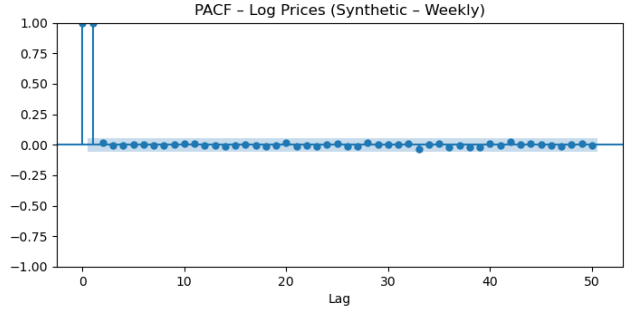

ACF – Log Prices (NASDAQ – Monthly)
Lag

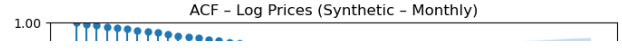

PACF – Log Prices (NASDAQ – Monthly)
Lag

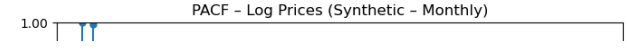

ACF – Log Prices (Synthetic – Daily)
Lag

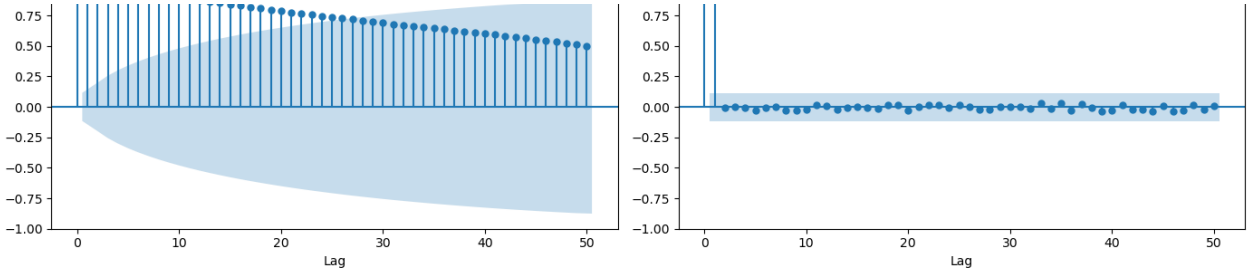

PACF – Log Prices (Synthetic – Daily)
Lag


ACF – Log Prices (Synthetic – Weekly)
Lag


PACF – Log Prices (Synthetic – Weekly)
Lag


ACF – Log Prices (Synthetic – Monthly)

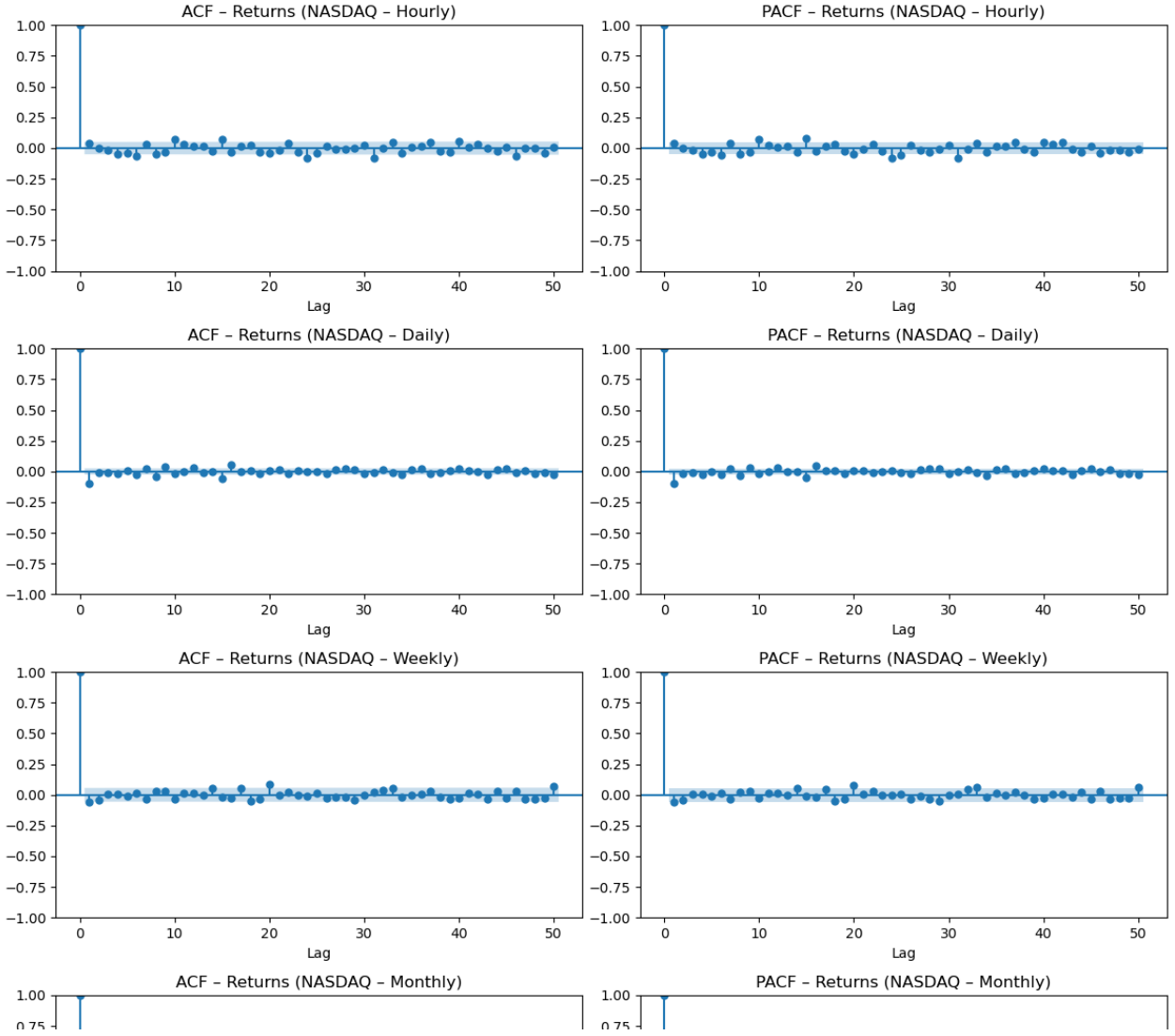

PACF – Log Prices (Synthetic – Monthly)

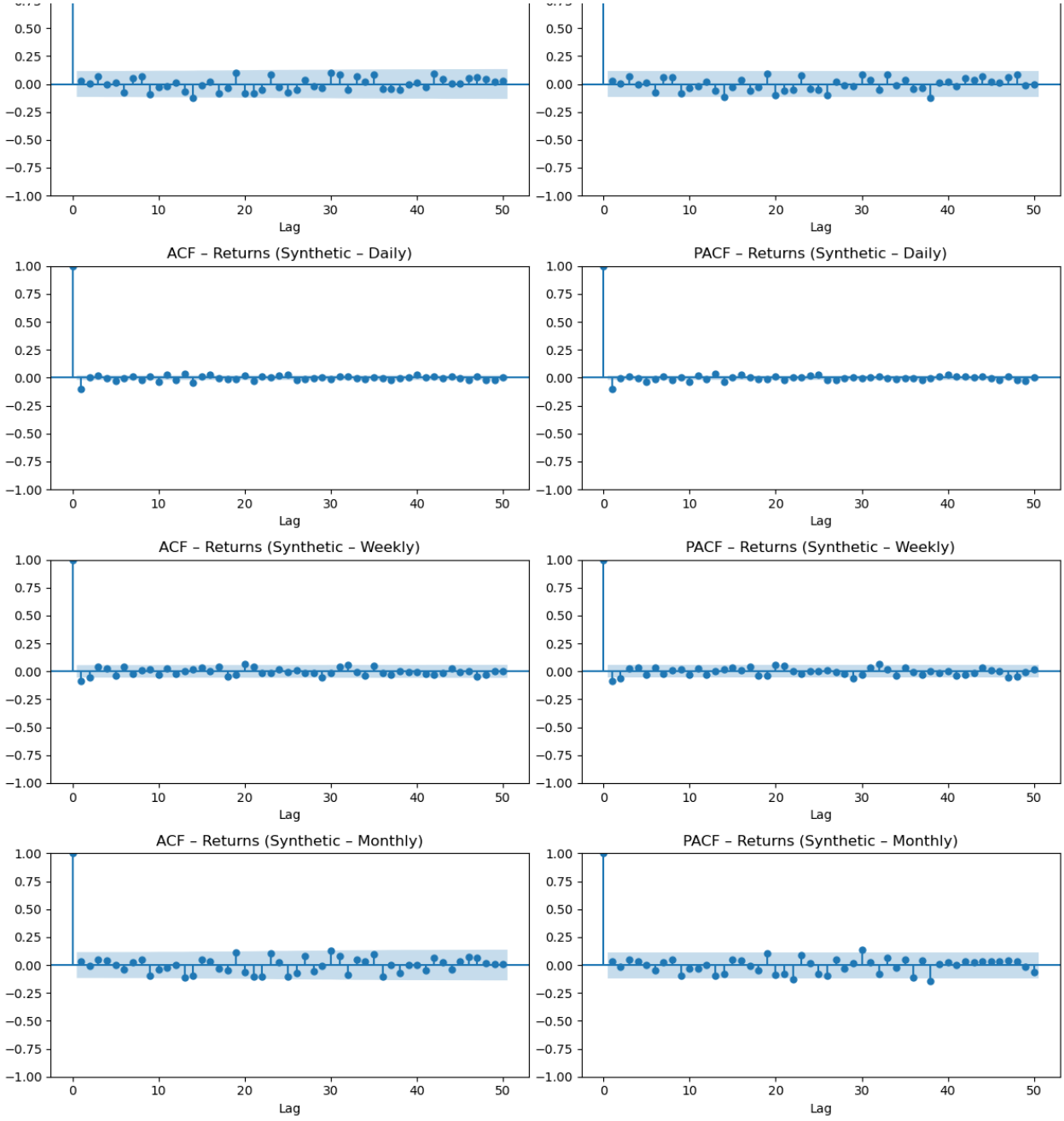


ACF fucntion for retunrs shows WHite Noise --> no correlations at all for all dates

In [71]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

def extract_series(obj, col=None):
    """
    Take a Series or DataFrame and return a 1D Series.
    """
    if isinstance(obj, pd.Series):
        return obj.dropna()
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns, specify
            return obj.iloc[:, 0].dropna()
        else:
            return obj[col].dropna()
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")


def plot_all_return_acf_pacf(pairs, lags=50):
    """
    ONE big figure with ACF/PACF of log returns.
    Each dataset gets its own row: ACF (left) and PACF (right).

    pairs : list of (obj, title, colname)
    """
    n = len(pairs)
    fig, axes = plt.subplots(n, 2, figsize=(12, 3.3 * n))

    if n == 1:
        axes = axes.reshape(1, 2)

    for i, (obj, title, col) in enumerate(pairs):
        r = extract_series(obj, col=col)

        # ACF
        plot_acf(r, ax=axes[i, 0], lags=lags)
        axes[i, 0].set_title(f"ACF — Returns ({title})")
        axes[i, 0].set_xlabel("Lag")

        # PACF
        plot_pacf(r, ax=axes[i, 1], lags=lags, method="ywm")
```

```
        axes[i, 1].set_title(f"PACF – Returns ({title})")
        axes[i, 1].set_xlabel("Lag")

    plt.tight_layout()
    plt.show()


# ------------------------------------------------------------
# Call the unified plotting function
# ------------------------------------------------------------

plot_all_return_acf_pacf(
    [
        (nasdaq_hourly_log_returns,    "NASDAQ – Hourly",   "Log_Returns"
        (nasdaq_daily_log_returns,     "NASDAQ – Daily",    "Log_Returns"
        (nasdaq_weekly_log_returns,    "NASDAQ – Weekly",   "Log_Returns"
        (nasdaq_monthly_log_returns,   "NASDAQ – Monthly",  "Log_Returns"
        (synthetic_daily_log_returns,   "Synthetic – Daily",   "Log_Retur
        (synthetic_weekly_log_returns,  "Synthetic – Weekly",  "Log_Retur
        (synthetic_monthly_log_returns, "Synthetic – Monthly", "Log_Retur
    ],
    lags=50
)
```



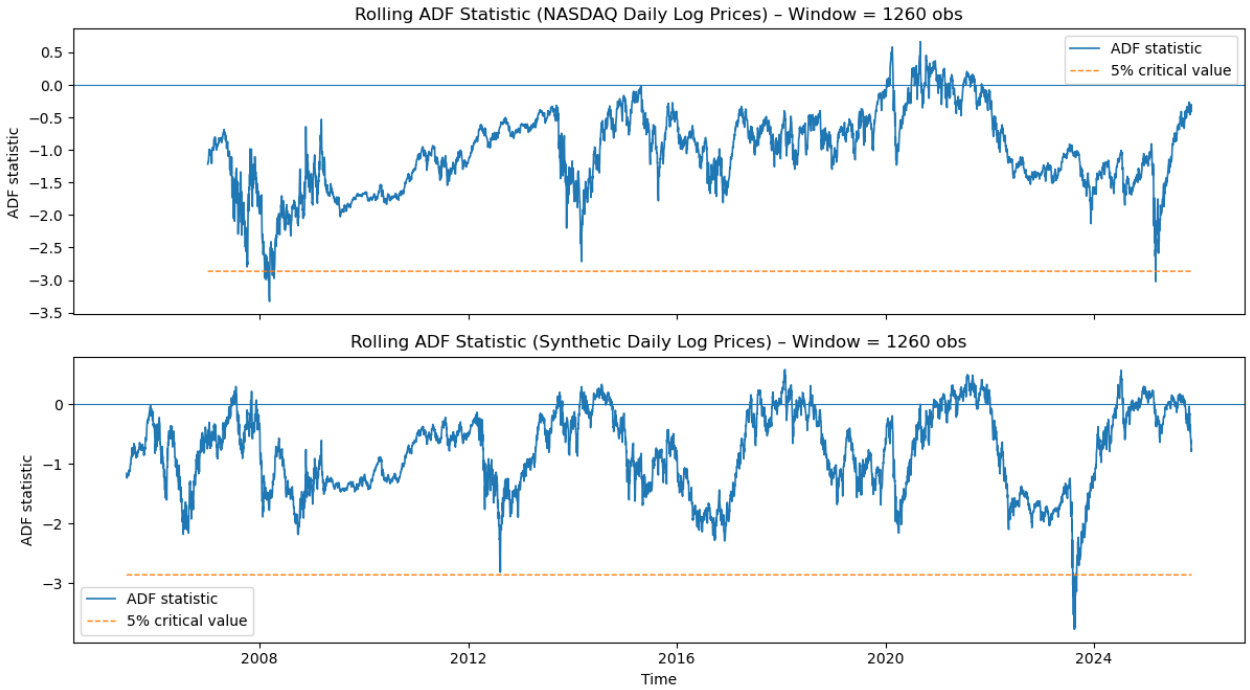


Rolling ADF Plot --> Shows how non-stationarity evolves over time.

In [74]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

def extract_series(obj, col=None):
    if isinstance(obj, pd.Series):
        return obj.dropna()
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError("DataFrame has multiple columns, specify
            return obj.iloc[:, 0].dropna()
        else:
```

```
            return obj[col].dropna()
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")


def compute_rolling_adf(obj, col=None, window=750, maxlag=None):
    """
    Helper: compute rolling ADF statistics and 5% critical values.
    Returns a DataFrame indexed by the window end date.
    """
    y = extract_series(obj, col=col)

    adf_stats = []
    crit_5_list = []
    idx = []

    for end in range(window, len(y) + 1):
        sub = y.iloc[end-window:end]
        try:
            res = adfuller(sub, maxlag=maxlag, autolag="AIC")
            stat = res[0]
            crit_5 = res[4]["5%"]
        except Exception:
            stat = np.nan
            crit_5 = np.nan
        adf_stats.append(stat)
        crit_5_list.append(crit_5)
        idx.append(sub.index[-1])

    roll_df = pd.DataFrame(
        {"adf_stat": adf_stats, "crit_5": crit_5_list},
        index=pd.Index(idx, name=y.index.name)
    )
    return roll_df


def plot_all_rolling_adf(pairs, window=750, maxlag=None):
    """
    One big figure with rolling ADF plots in separate panels.

    pairs : list of (obj, title, colname)
    window, maxlag : passed to compute_rolling_adf
    """
    n = len(pairs)
    fig, axes = plt.subplots(n, 1, figsize=(12, 3.4 * n), sharex=True)

    if n == 1:
        axes = [axes]

    for i, (obj, title, col) in enumerate(pairs):
        roll_df = compute_rolling_adf(obj, col=col, window=window, maxlag

        ax = axes[i]
        ax.plot(roll_df.index, roll_df["adf_stat"], linewidth=1.2, label=
```

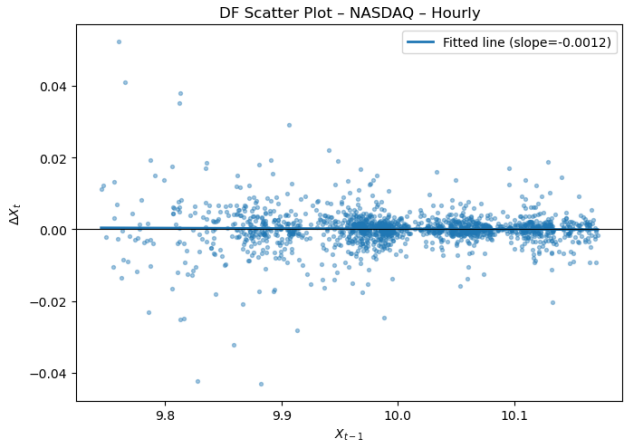
```
        ax.plot(roll_df.index, roll_df["crit_5"], linestyle="--", linewid
                label="5% critical value")
        ax.axhline(0, linewidth=0.8)

        ax.set_title(f"Rolling ADF Statistic ({title}) – Window = {window
        ax.set_ylabel("ADF statistic")
        ax.legend()

    axes[-1].set_xlabel("Time")

    plt.tight_layout()
    plt.show()


# ------------------------------------------------------------
# Use it for your two daily series
# ------------------------------------------------------------

plot_all_rolling_adf(
    [
        (nasdaq_daily_log_prices,     "NASDAQ Daily Log Prices",    "Log_
        (synthetic_daily_log_prices,  "Synthetic Daily Log Prices", "Log_
    ],
    window=1260
)
```

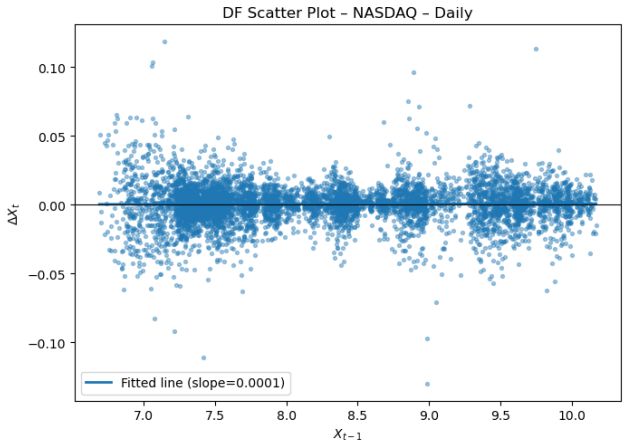


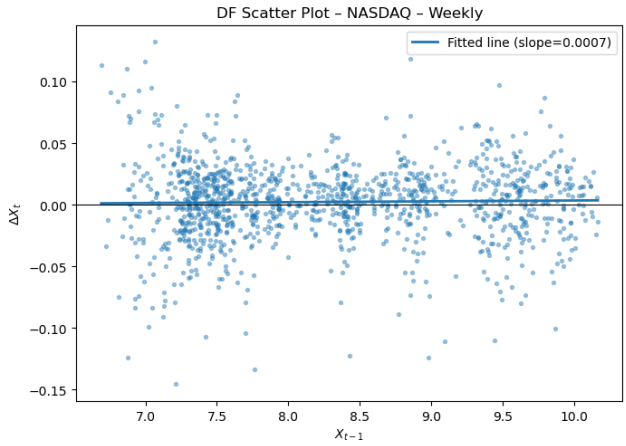
In [73]:
```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

def df_scatter_plot_grid(pairs):
    """
    Produce DF scatter plots arranged 2 per row.
```

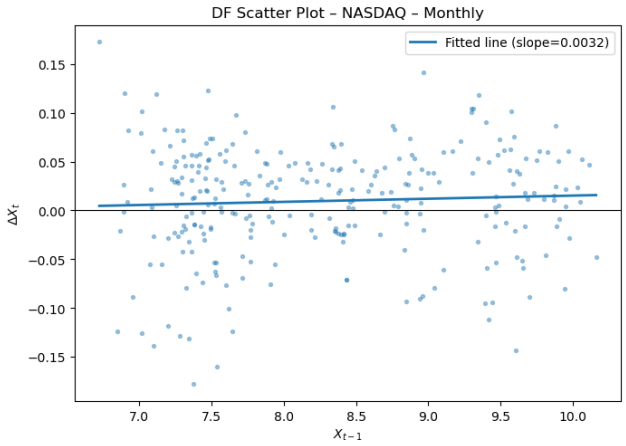
```
    pairs: list of (obj, title, colname)
    """
    n = len(pairs)
    ncols = 2
    nrows = int(np.ceil(n / ncols))

    fig, axes = plt.subplots(nrows, ncols, figsize=(14, 5 * nrows))
    axes = axes.flatten()

    for i, (obj, title, col) in enumerate(pairs):
        ax = axes[i]

        x = extract_series(obj, col=col)

        # Build X_{t-1} and ΔX_t
        x_lag = x.shift(1).dropna()
        dx = x.diff().dropna()

        df_tmp = pd.DataFrame({"x_lag": x_lag, "dx": dx}).dropna()

        X = sm.add_constant(df_tmp["x_lag"])
        y = df_tmp["dx"]
        model = sm.OLS(y, X).fit()

        slope = model.params["x_lag"]
        intercept = model.params["const"]

        # Scatter
        ax.scatter(df_tmp["x_lag"], df_tmp["dx"], s=8, alpha=0.4)

        # Line
        x_vals = np.linspace(df_tmp["x_lag"].min(), df_tmp["x_lag"].max()
        y_hat = intercept + slope * x_vals
        ax.plot(x_vals, y_hat, linewidth=2,
                label=f"Fitted line (slope={slope:.4f})")

        ax.axhline(0, color="black", linewidth=0.8)
        ax.set_xlabel(r"$X_{t-1}$")
        ax.set_ylabel(r"$\Delta X_t$")
        ax.set_title(f"DF Scatter Plot — {title}")
        ax.legend()

    # Hide unused axes (e.g., if odd number of plots)
    for j in range(i + 1, len(axes)):
        axes[j].axis("off")

    plt.tight_layout()
    plt.show()


# ------------------------------------------------------------
# Call the 2-per-row grid version
# ------------------------------------------------------------
```

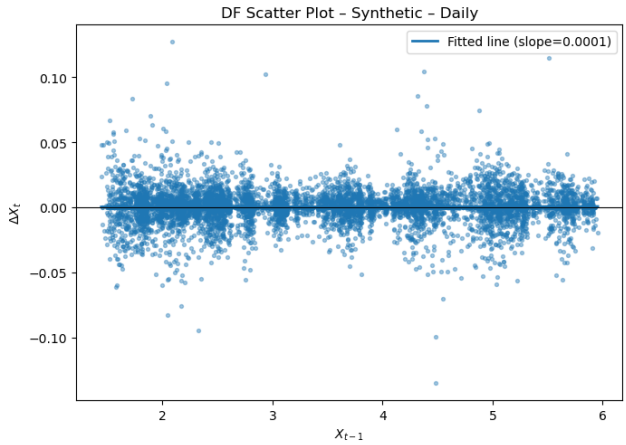
```
df_scatter_plot_grid(
    [
        (nasdaq_hourly_log_prices,    "NASDAQ – Hourly",   "Log_Prices"),
        (nasdaq_daily_log_prices,     "NASDAQ – Daily",    "Log_Prices"),
        (nasdaq_weekly_log_prices,    "NASDAQ – Weekly",   "Log_Prices"),
        (nasdaq_monthly_log_prices,   "NASDAQ – Monthly",  "Log_Prices"),
        (synthetic_daily_log_prices,   "Synthetic – Daily",   "Log_Prices
        (synthetic_weekly_log_prices,  "Synthetic – Weekly",  "Log_Prices
        (synthetic_monthly_log_prices, "Synthetic – Monthly", "Log_Prices
    ]
)
```

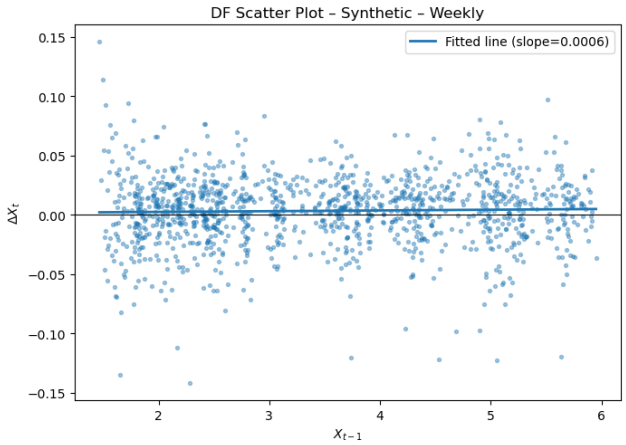


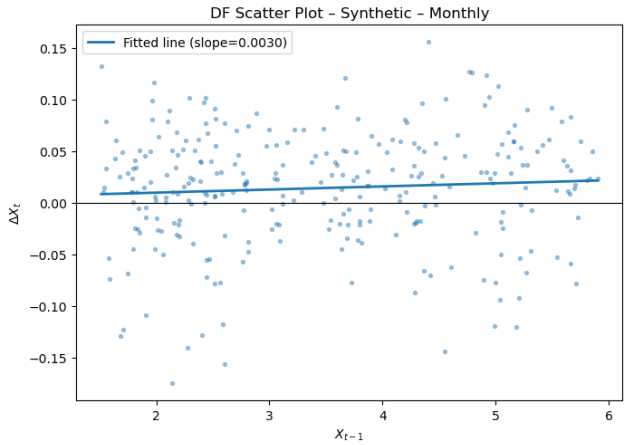








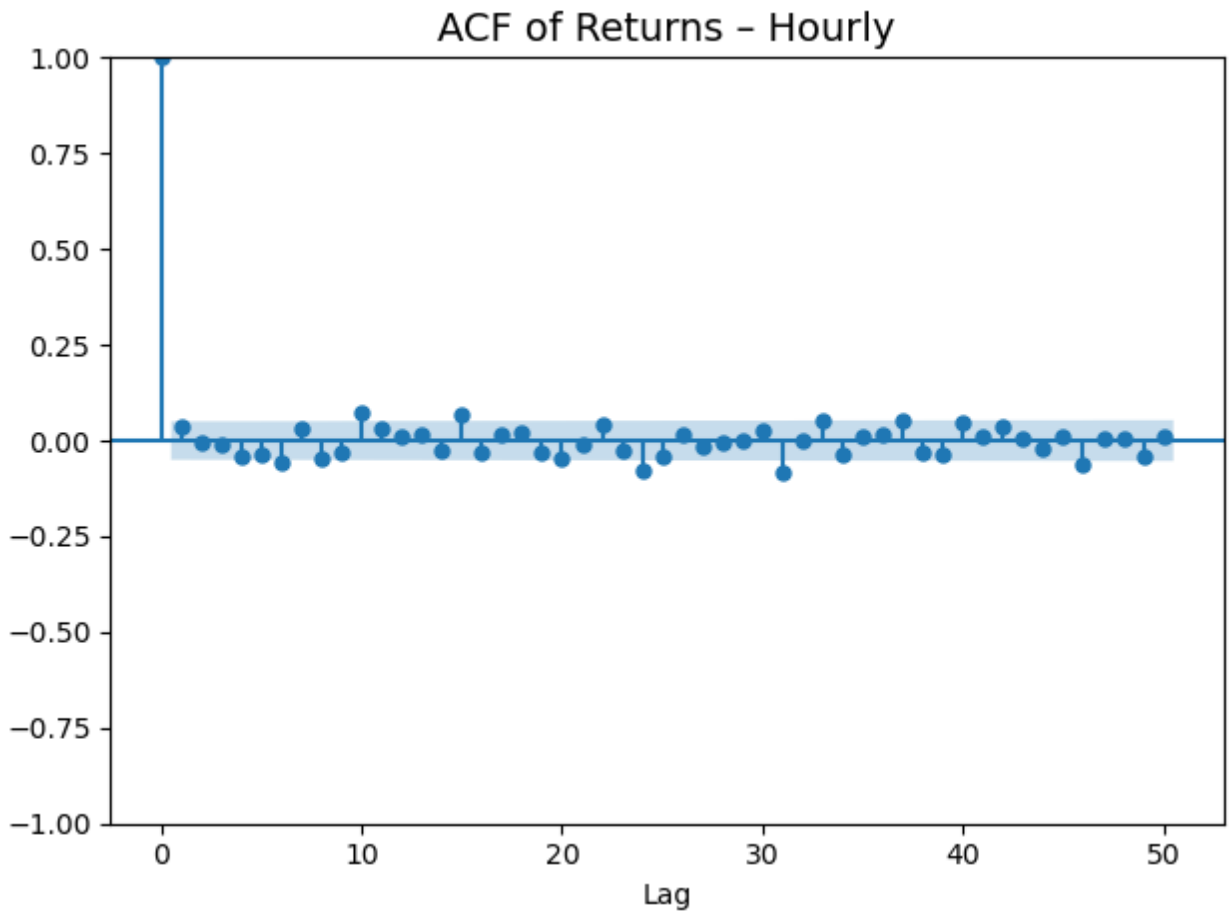




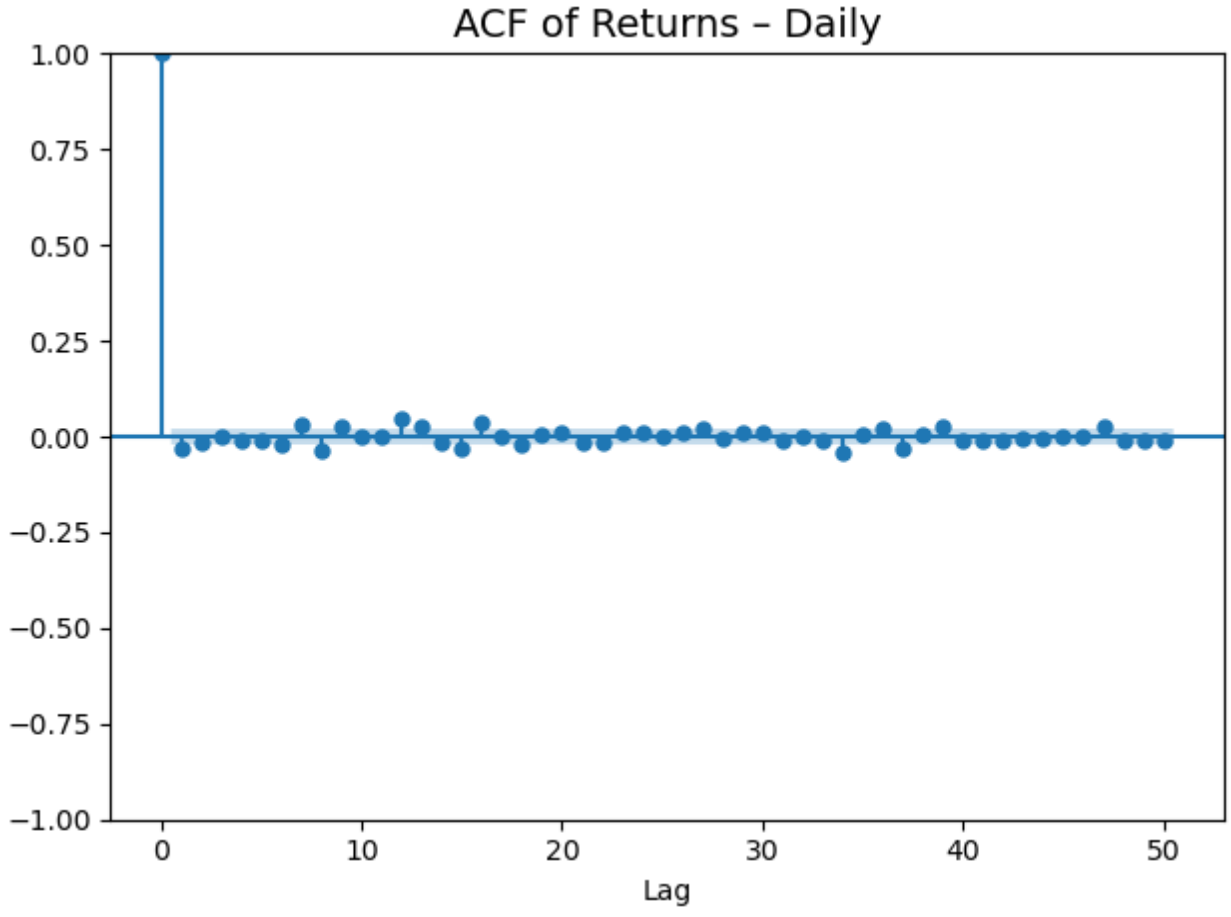
White noise Returns Plots ---> If ACFs lie inside 95% CI → martingale difference sequence.

In [ ]:
```python
def extract_series(obj, col=None):
    """
    Take a Series or DataFrame and return a 1D Series.
    """
    if isinstance(obj, pd.Series):
        return obj.dropna()
    elif isinstance(obj, pd.DataFrame):
        if col is None:
            if obj.shape[1] != 1:
                raise ValueError(
                    "DataFrame has multiple columns, specify col='colname
                )
            return obj.iloc[:, 0].dropna()
        else:
            return obj[col].dropna()
    else:
        raise TypeError("obj must be a pandas Series or DataFrame")


def plot_return_acf(obj, title, col=None, lags=50):
    """
    ACF of returns with 95% CI.
    If all autocorrelations lie inside the CI → martingale difference.
    """
    r = extract_series(obj, col=col)

    plt.figure(figsize=(8, 4))
    plot_acf(r, lags=lags)
    plt.title(f"ACF of Returns — {title}", fontsize=14)
    plt.xlabel("Lag")
    plt.tight_layout()
    plt.show()


plot_return_acf(nasdaq_hourly_df_log_returns,  "Hourly",  col="Log_Return
plot_return_acf(nasdaq_daily_log_returns,   "Daily",   col="Log_Returns")
plot_return_acf(nasdaq_weekly_log_returns,  "Weekly",  col="Log_Returns")
plot_return_acf(nasdaq_monthly_log_returns, "Monthly", col="Log_Returns")
```

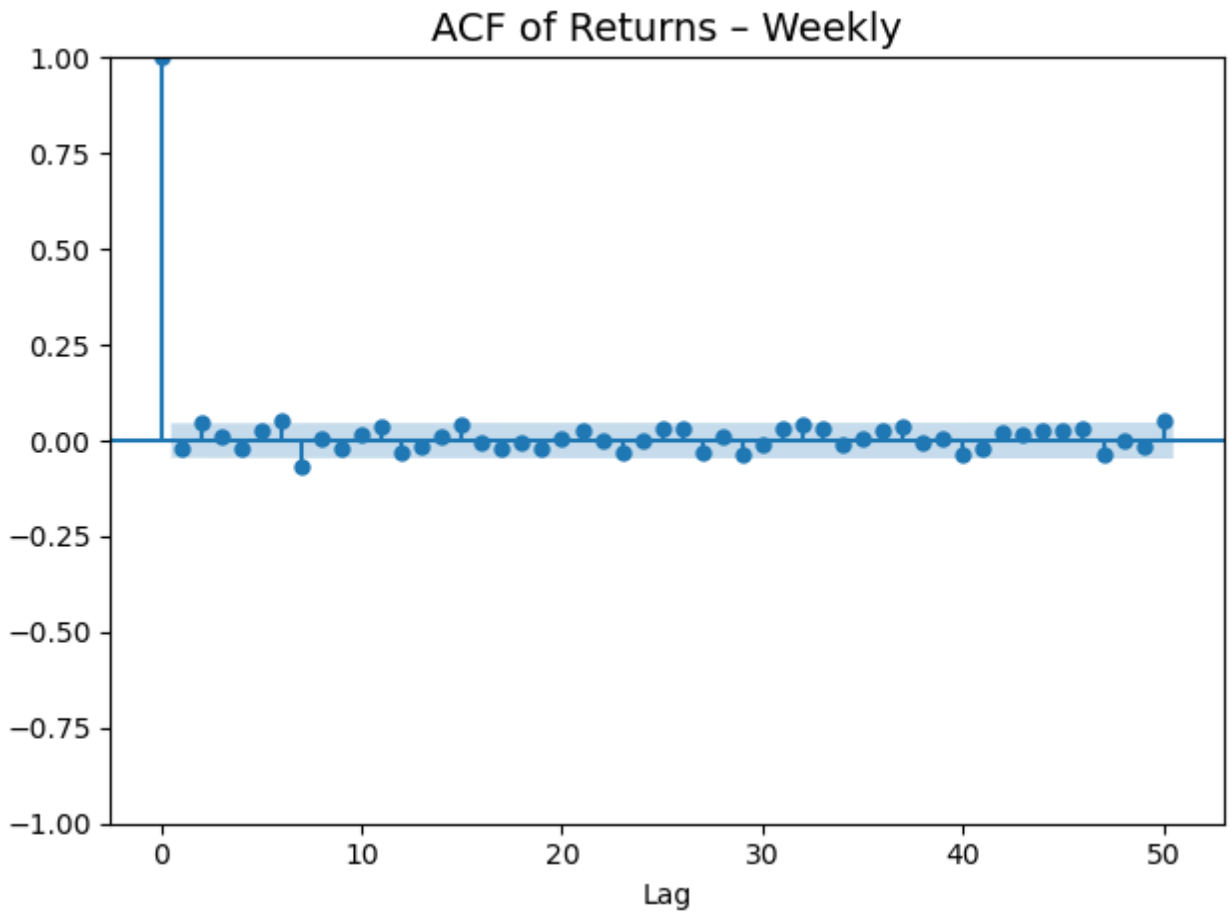
```
<Figure size 800x400 with 0 Axes>
```



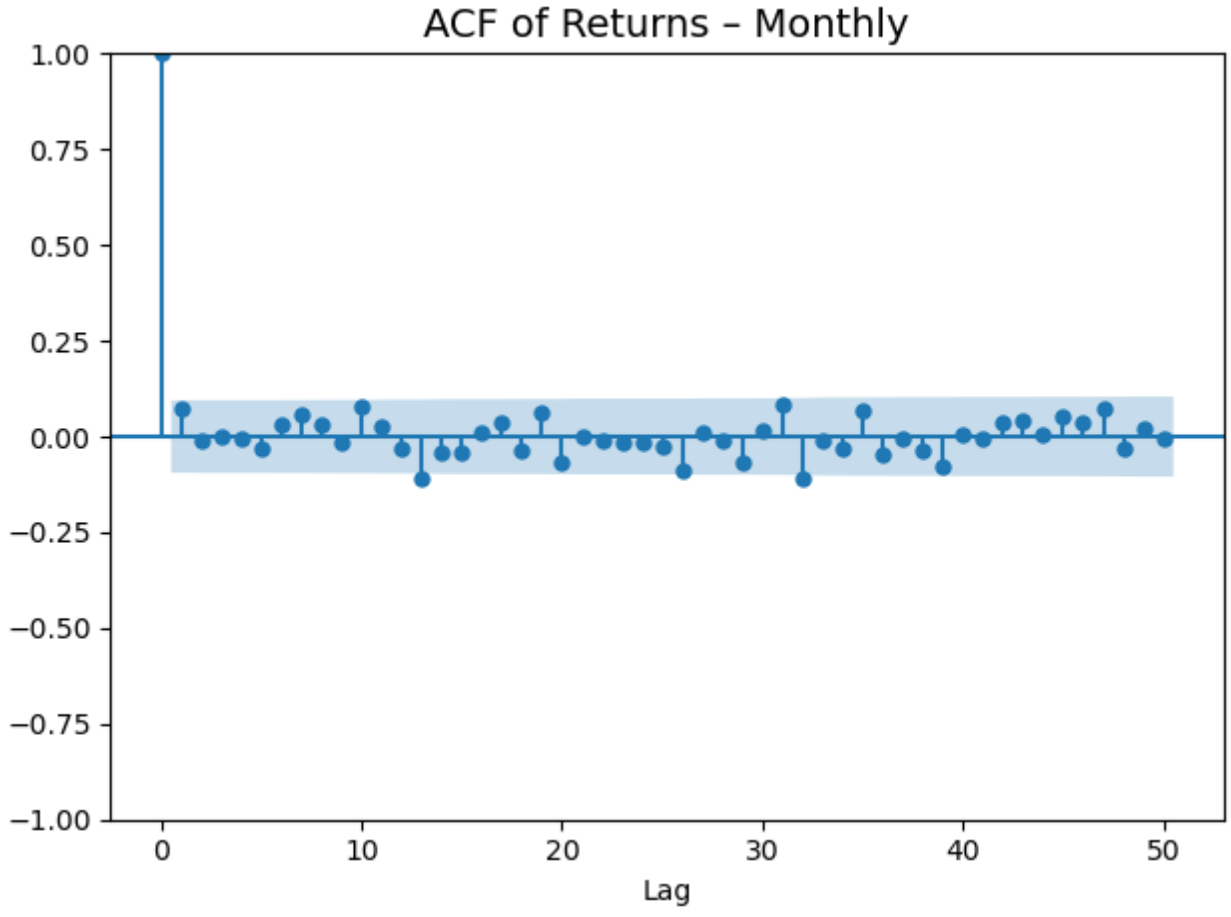

```
<Figure size 800x400 with 0 Axes>
```




```
<Figure size 800x400 with 0 Axes>
```

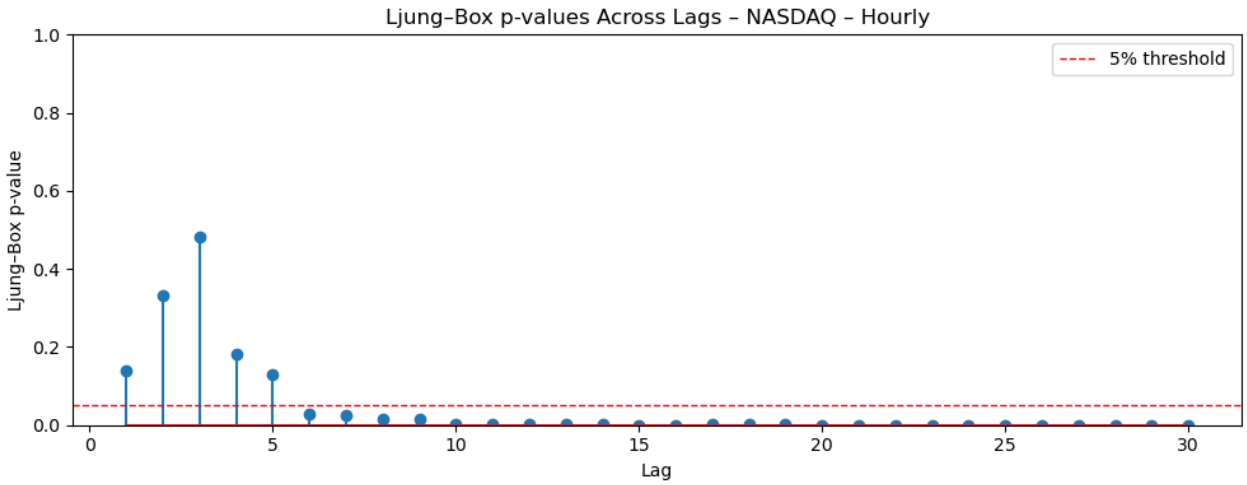


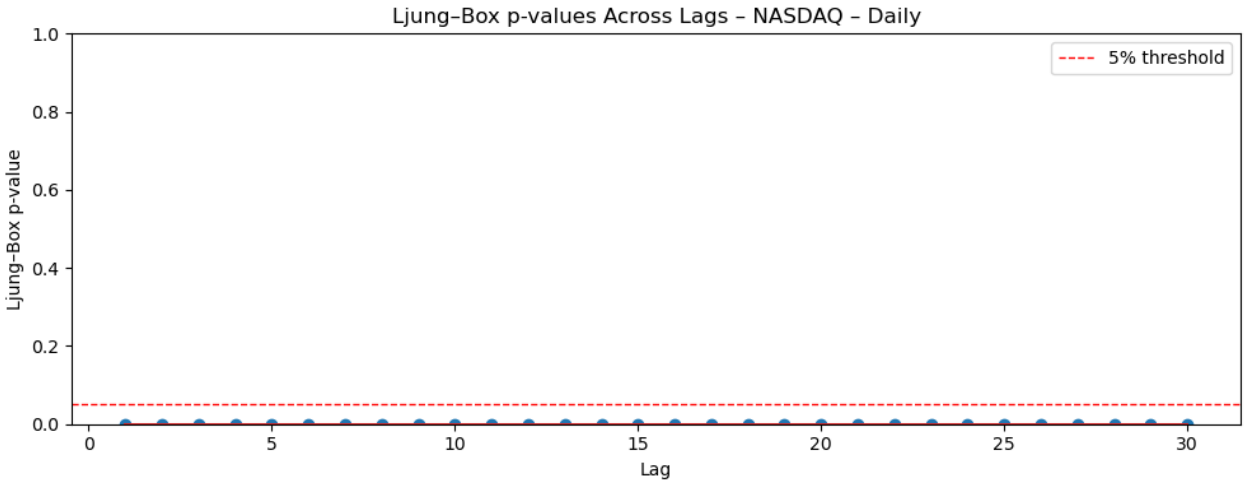

```
<Figure size 800x400 with 0 Axes>
```

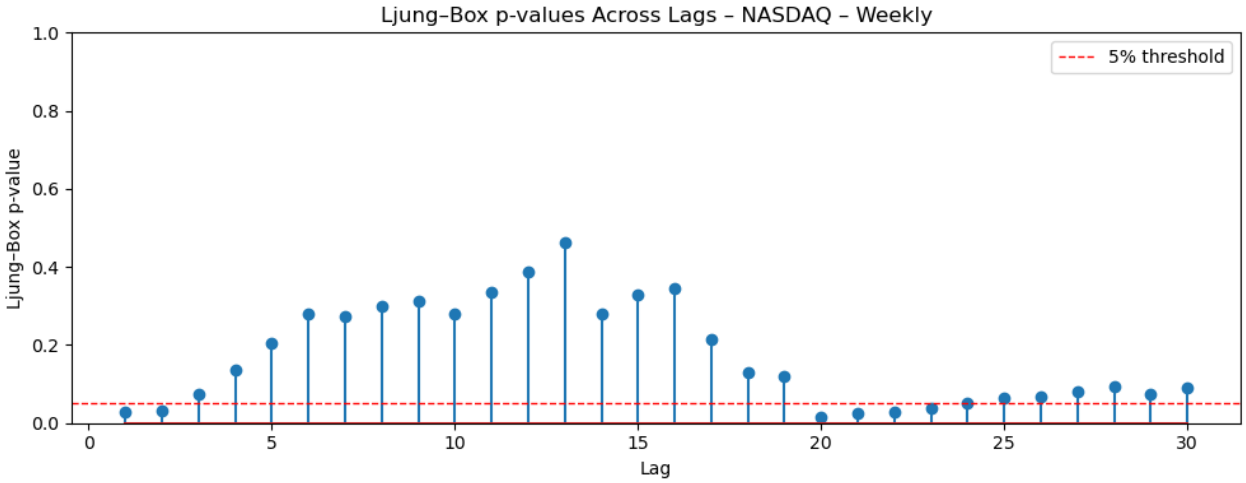


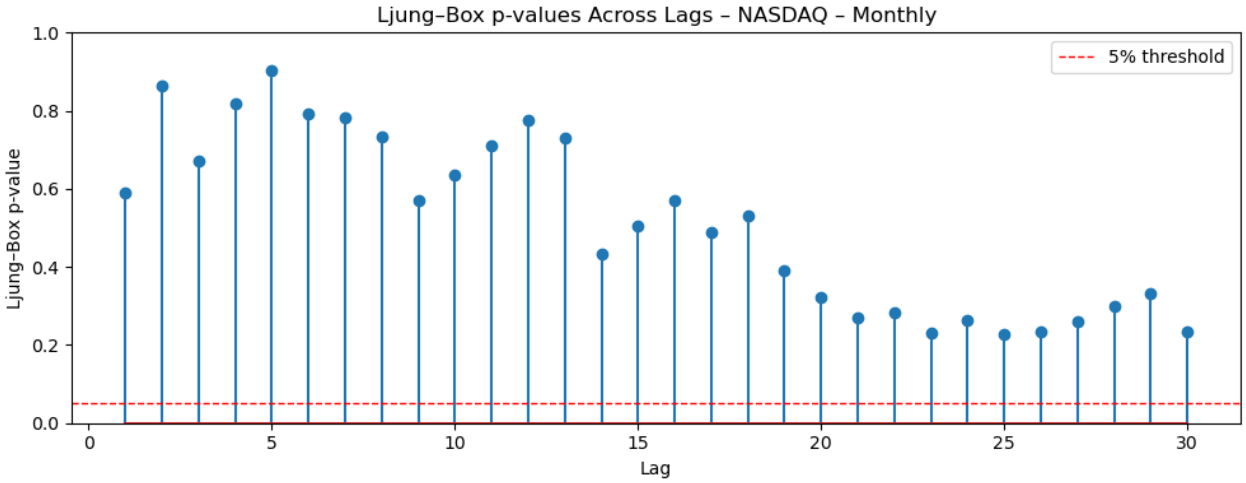
Ljung - Box- P-value Plot --> Seeing all p-values at once is much clearer. Reject RW3 if many p-values < 0.05.

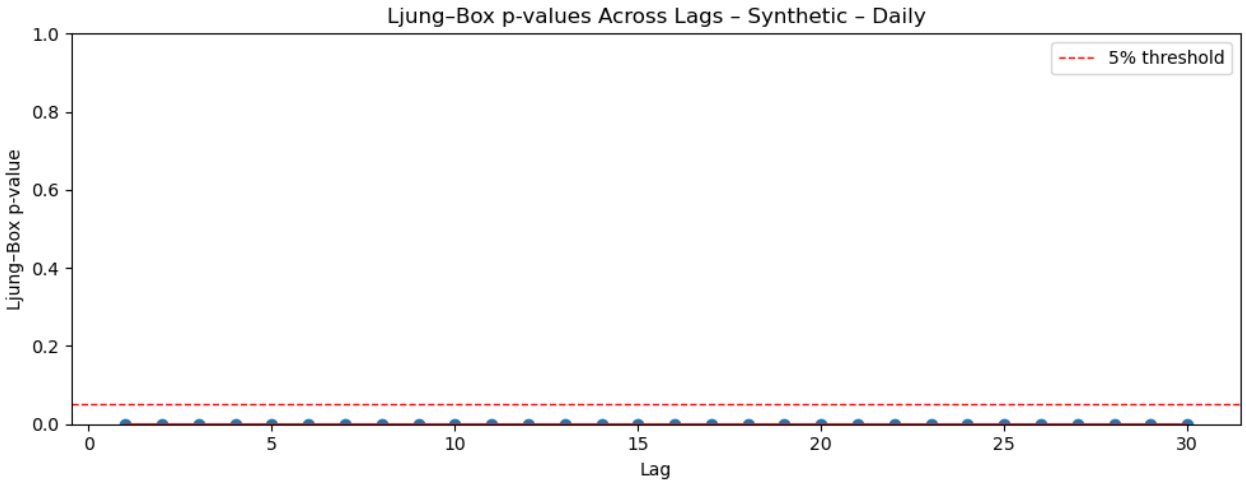
In [ ]:
```
def ljung_box_pvalue_plot(obj, title, col=None, max_lag=30):
    """
    Ljung–Box p-value plot for lags 1 to max_lag.
    Reject RW3 if p-values < 0.05.
    """
    r = extract_series(obj, col=col)

    lb = acorr_ljungbox(r, lags=list(range(1, max_lag + 1)), return_df=Tr
    pvals = lb["lb_pvalue"].values
    lags = np.arange(1, max_lag + 1)

    plt.figure(figsize=(10, 4))
    # OLD (breaks on your version):
    # plt.stem(lags, pvals, use_line_collection=True)
    # NEW:
    plt.stem(lags, pvals)

    plt.axhline(0.05, color="red", linestyle="--", linewidth=1, label="5%
    plt.ylim(0, 1)
    plt.xlabel("Lag")
    plt.ylabel("Ljung–Box p-value")
    plt.title(f"Ljung–Box p-values Across Lags – {title}")
    plt.legend()
    plt.tight_layout()
    plt.show()

ljung_box_pvalue_plot(nasdaq_hourly_log_returns,   "NASDAQ – Hourly",  co
ljung_box_pvalue_plot(nasdaq_daily_log_returns,    "NASDAQ – Daily",   co
ljung_box_pvalue_plot(nasdaq_weekly_log_returns,   "NASDAQ – Weekly",  co
ljung_box_pvalue_plot(nasdaq_monthly_log_returns,  "NASDAQ – Monthly", co

ljung_box_pvalue_plot(synthetic_daily_log_returns,    "Synthetic – Daily"
ljung_box_pvalue_plot(synthetic_weekly_log_returns,   "Synthetic – Weekly
ljung_box_pvalue_plot(synthetic_monthly_log_returns,  "Synthetic – Monthl
```

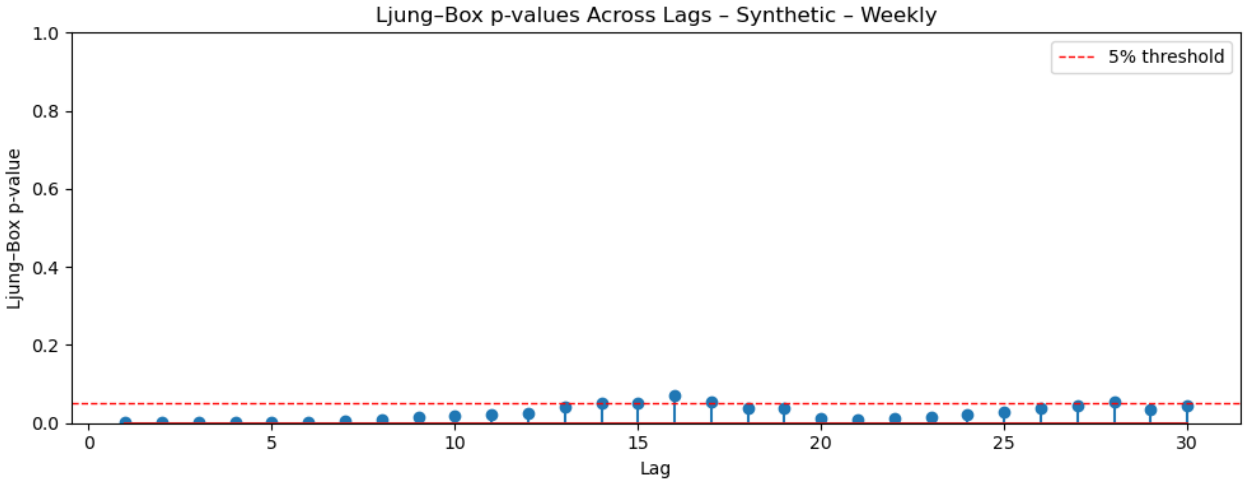


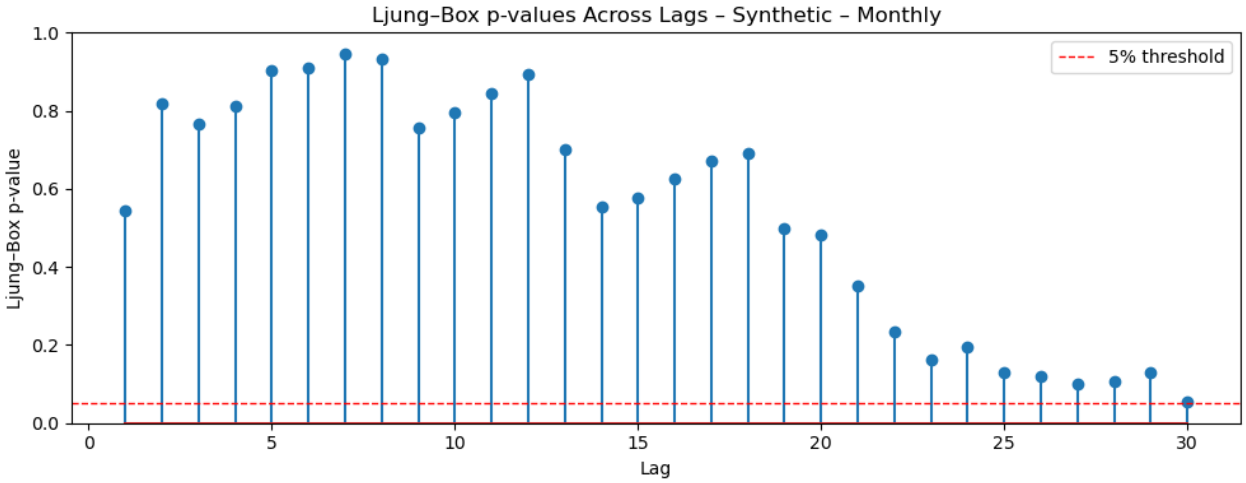

Ljung–Box p-values Across Lags – NASDAQ – Daily
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30


Ljung–Box p-values Across Lags – NASDAQ – Weekly
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30


Ljung–Box p-values Across Lags – NASDAQ – Monthly
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30


Ljung–Box p-values Across Lags – Synthetic – Daily
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30


Ljung–Box p-values Across Lags – Synthetic – Weekly
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30


Ljung–Box p-values Across Lags – Synthetic – Monthly
5% threshold
Ljung–Box p-value
Lag
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30

Part 2 Fondamental Value Modeling

Import

In [1]:
```
# Packages


import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, coint
import os as os
Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")
#Data
#Price
price_d = pd.read_csv(os.path.join(Data_dir, "synthetic_price_daily_df.cs
price_w = pd.read_csv(os.path.join(Data_dir, "synthetic_price_weekly_df.c
price_m = pd.read_csv(os.path.join(Data_dir, "synthetic_price_monthly_df.
print(price_d.head())

#Div
div_d = pd.read_csv(os.path.join(Data_dir, "synthetic_div_daily_df.csv"),
div_w = pd.read_csv(os.path.join(Data_dir, "synthetic_div_weekly_df.csv")
div_m = pd.read_csv(os.path.join(Data_dir, "synthetic_div_monthly_df.csv"
print(div_d.head())

#NDX100
nasdaq_d = pd.read_csv(os.path.join(Data_dir, "nasdaq_daily_df.csv"),pars
nasdaq_d = nasdaq_d[["Date","Close"]]
nasdaq_w = pd.read_csv(os.path.join(Data_dir, "nasdaq_weekly_df.csv"),par
nasdaq_w = nasdaq_w[["Date","Close"]]
nasdaq_m = pd.read_csv(os.path.join(Data_dir, "nasdaq_monthly_df.csv"),pa
nasdaq_m = nasdaq_m[["Date","Close"]]
print(nasdaq_d.head())
```

```
                       Date  Synthetic Index Close Price
0  2002-01-01 00:00:00-05:00                     6.741954
1  2002-01-02 00:00:00-05:00                     6.846058
2  2002-01-03 00:00:00-05:00                     7.054649
3  2002-01-04 00:00:00-05:00                     7.039679
4  2002-01-05 00:00:00-05:00                     7.039679
                       Date  Synthetic Index Dividend
0  2002-01-01 00:00:00-05:00                       0.0
1  2002-01-02 00:00:00-05:00                       0.0
2  2002-01-03 00:00:00-05:00                       0.0
3  2002-01-04 00:00:00-05:00                       0.0
4  2002-01-05 00:00:00-05:00                       0.0
        Date        Close
0 2002-01-02  1610.390015
1 2002-01-03  1666.660034
2 2002-01-04  1675.030029
3 2002-01-07  1649.829956
4 2002-01-08  1666.579956
```

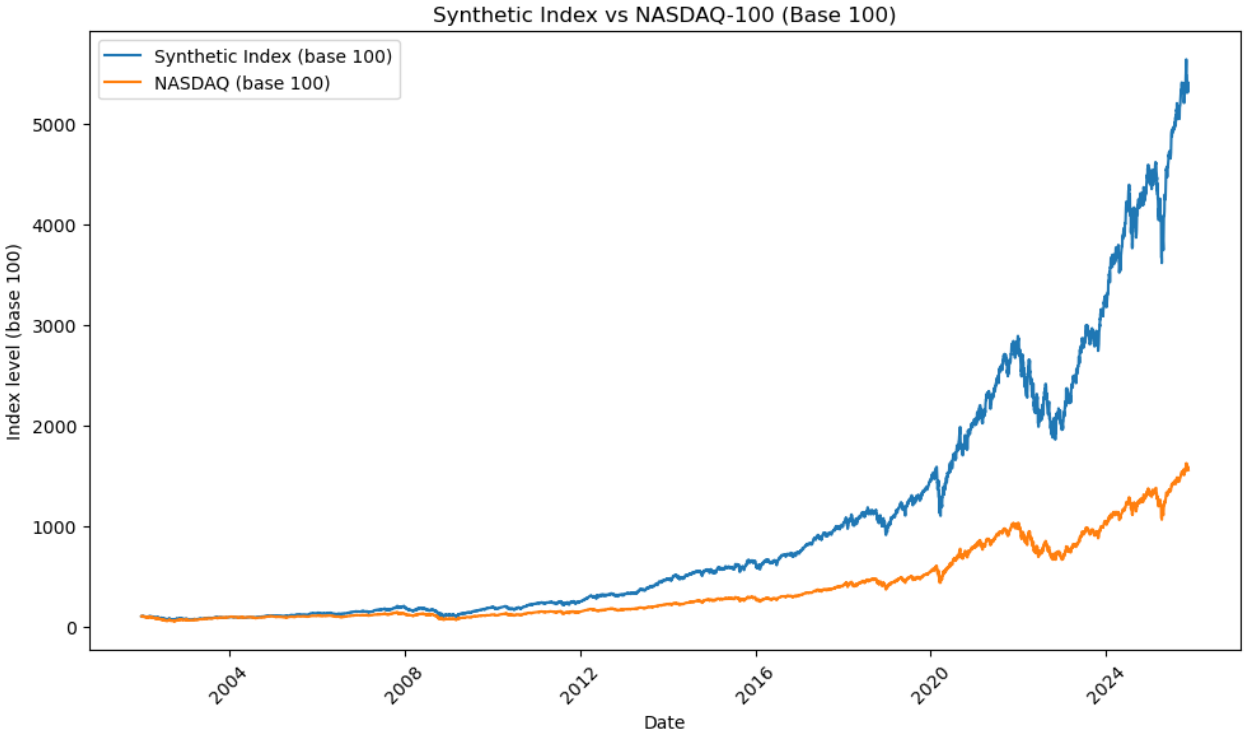
1) Index Tracking (VS Ndx100)

In [2]:

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

for df in (price_d, nasdaq_d):
    df["Date"] = pd.to_datetime(df["Date"], utc=True)
    df["Date"] = df["Date"].dt.tz_convert(None)
    df["Date"] = df["Date"].dt.normalize()

p = price_d.set_index("Date")[["Synthetic Index Close Price"]]
n = nasdaq_d.set_index("Date")[["Close"]]

df = p.join(n, how="inner").sort_index()

print("Joined shape:", df.shape)

if df.empty:
    raise ValueError("No overlapping dates between price_d and nasdaq_d a

df_base100 = df / df.iloc[0] * 100

plt.figure(figsize=(10,6))
plt.plot(df_base100.index,
         df_base100["Synthetic Index Close Price"],
         label="Synthetic Index (base 100)")
plt.plot(df_base100.index,
         df_base100["Close"],
         label="NASDAQ (base 100)")

plt.xlabel("Date")
plt.ylabel("Index level (base 100)")
plt.title("Synthetic Index vs NASDAQ-100 (Base 100)")
```

```
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

```
Joined shape: (6008, 2)
```



1.1) Regression of Nasdaq = alpha + Synthetic Index * beta

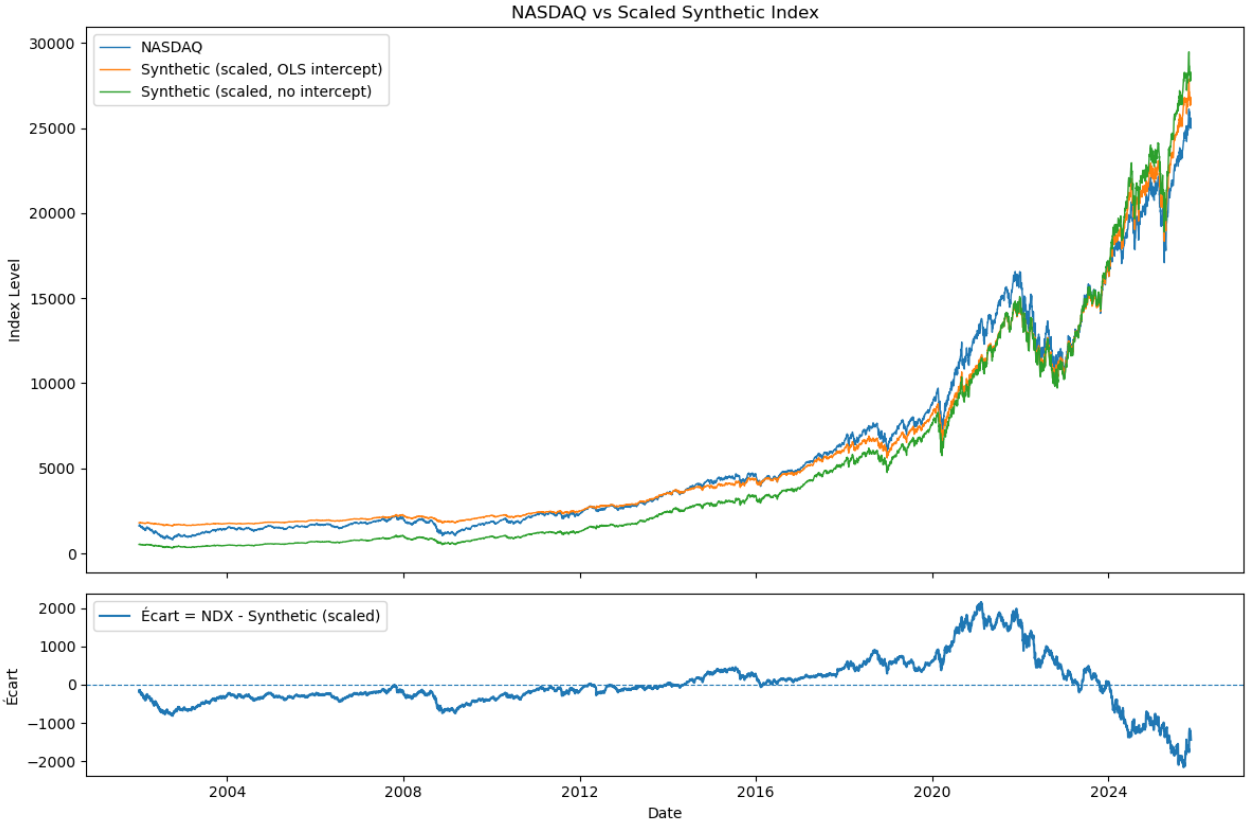
In [3]:
```
import statsmodels.api as sm

for df in (price_d, nasdaq_d):
    df["Date"] = pd.to_datetime(df["Date"], utc=True)
    df["Date"] = df["Date"].dt.tz_convert(None)
    df["Date"] = df["Date"].dt.normalize()

p = price_d.set_index("Date")[["Synthetic Index Close Price"]]
n = nasdaq_d.set_index("Date")[["Close"]]

df = p.join(n, how="inner").sort_index()
print(df)

y = df["Close"]


X = df["Synthetic Index Close Price"]

X1 = sm.add_constant(X)
ols = sm.OLS(y, X1).fit()

print(ols.summary())
```

```
alpha = ols.params["const"]
beta = ols.params[X.name]


df["synthetic_scaled"] = alpha + beta * df["Synthetic Index Close Price"]

ols_no_const = sm.OLS(y, X).fit()
beta_nc = ols_no_const.params[X.name]

df["synthetic_scaled_no_const"] = beta_nc * df["Synthetic Index Close Pri
```

```
            Synthetic Index Close Price         Close
Date
2002-01-02                     6.846058   1610.390015
2002-01-03                     7.054649   1666.660034
2002-01-04                     7.039679   1675.030029
2002-01-07                     6.982579   1649.829956
2002-01-08                     7.021340   1666.579956
...                                 ...          ...
2025-11-10                   370.584797  25611.740234
2025-11-11                   370.503619  25533.490234
2025-11-12                   370.481964  25517.330078
2025-11-13                   364.229183  24993.460938
2025-11-14                   365.196053  25008.240234

[6008 rows x 2 columns]
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                   Close   R-squared:
0.987
Model:                             OLS   Adj. R-squared:
0.987
Method:                  Least Squares   F-statistic:                4.672
e+05
Date:                 Fri, 21 Nov 2025   Prob (F-statistic):
0.00
Time:                         16:26:43   Log-Likelihood:                -47
605.
No. Observations:                6008   AIC:                           9.521
e+04
Df Residuals:                    6006   BIC:                           9.523
e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
--------------------
const                        1328.3447     11.088    119.802      0.000
1306.608    1350.081
```

```
Synthetic Index Close Price    68.7909      0.101    683.520      0.000
68.594      68.988
==============================================================================
====
Omnibus:                      444.799   Durbin-Watson:
0.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              79
8.288
Skew:                           0.539   Prob(JB):                    4.51e
-174
Kurtosis:                       4.424   Cond. No.
142.
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

In [4]:
```python
import matplotlib.pyplot as plt

df["ecart"] = df["Close"] - df["synthetic_scaled"]
fig, (ax1, ax2) = plt.subplots(
    2, 1,
    figsize=(12, 8),
    sharex=True,
    gridspec_kw={"height_ratios": [3, 1]}
)

ax1.plot(df.index, df["Close"], label="NASDAQ", linewidth=1)
ax1.plot(df.index, df["synthetic_scaled"], label="Synthetic (scaled, OLS
ax1.plot(df.index, df["synthetic_scaled_no_const"], label="Synthetic (sca

ax1.set_title("NASDAQ vs Scaled Synthetic Index")
ax1.set_ylabel("Index Level")
ax1.legend(loc="upper left")

ax2.plot(df.index, df["ecart"], label="Écart = NDX - Synthetic (scaled)")
ax2.axhline(0, linestyle="--", linewidth=0.8)
ax2.set_ylabel("Écart")
ax2.set_xlabel("Date")
ax2.legend(loc="upper left")

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



NDX ≈ 1328.3552 + 68.7614 * Synthetic

2) Cointegration

2.1) Data prep

In [5]:
```
for df in (price_m, div_m):
    df["Date"] = pd.to_datetime(df["Date"], utc=True)
    df["Date"] = df["Date"].dt.tz_convert(None)
    df["Date"] = df["Date"].dt.normalize()

p = price_m.set_index("Date")[["Synthetic Index Close Price"]]
n = div_m.set_index("Date")[["Synthetic Index Dividend"]]
df_m = p.join(n, how="inner").sort_index()

print("Joined shape:", df.shape)

if df_m.empty:
    raise ValueError("No overlapping dates between price_d and nasdaq_d a
df_m = df_m.rename(
    columns={
        "Synthetic Index Close Price": "P",
        "Synthetic Index Dividend": "D"
    }
)
```

```
print(df_m)
```

```
Joined shape: (287, 2)
                    P         D
Date
2002-01-01    6.741954  0.000382
2002-02-01    6.708014  0.006262
2002-03-01    6.614498  0.002720
2002-04-01    6.714867  0.001222
2002-05-01    6.022081  0.003142
...                ...       ...
2025-07-01  333.452353  0.055846
2025-08-01  340.753141  0.253449
2025-09-01  348.811723  0.216313
2025-10-01  366.551887  0.076930
2025-11-01  375.118033  0.061543

[287 rows x 2 columns]
```

In [12]:
```
df_m = df_m[df_m["P"] > 0]
df_m = df_m[df_m["D"] > 0]


df_m["p"] = np.log(df_m["P"])
df_m["D12"] = df_m["D"].rolling(window=12, min_periods=12).sum()
df_m["d"] = np.log(df_m["D"])
df_m["d12"] = np.log(df_m["D12"])

df_m["dp"] = df_m["d"] - df_m["p"]
df_m["d12p"] = df_m["d12"] - df_m["p"]
df_m = df_m.dropna(subset=["D12"])
df_m
```

Out[12]:

Date	P	D	p	D12	d	d12	
2003-11-01	6.024359	0.003224	1.795811	0.069044	-5.737228	-2.673015	-7.5330
2003-12-01	6.028666	0.004729	1.796526	0.070163	-5.354113	-2.656939	-7.1506
2004-01-01	6.232481	0.000407	1.829775	0.070188	-7.806927	-2.656574	-9.6367
2004-02-01	6.297128	0.006879	1.840094	0.058063	-4.979352	-2.846233	-6.8194
2004-03-01	6.261209	0.003979	1.834373	0.058829	-5.526829	-2.833118	-7.3612
...	...	...	...	...	...	...	
2025-07-01	333.452353	0.055846	5.809500	2.022016	-2.885165	0.704095	-8.6946
2025-08-01	340.753141	0.253449	5.831158	2.059450	-1.372591	0.722439	-7.2037
2025-09-01	348.811723	0.216313	5.854532	2.074903	-1.531027	0.729915	-7.3855
2025-10-01	366.551887	0.076930	5.904140	2.101338	-2.564857	0.742574	-8.4689
2025-11-01	375.118033	0.061543	5.927241	1.918976	-2.788022	0.651791	-8.7152

265 rows × 8 columns

2.2) Engle-Granger for log Div / log Price

In [7]:
```
p = df_m["p"]
d = df_m["d"]


X = sm.add_constant(d)
ols_res = sm.OLS(p, X).fit()
print(ols_res.summary())


u_hat = ols_res.resid

adf_stat, adf_p, _, _, crit_vals, _ = adfuller(u_hat, maxlag=12, regressi

print("\nEngle–Granger residual ADF test:")
print(f"ADF statistic: {adf_stat:.3f}")
```

```
print(f"p-value      : {adf_p:.3f}")
print("Critical values:", crit_vals)
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                          p   R-squared:
0.486
Model:                                OLS   Adj. R-squared:
0.484
Method:                     Least Squares   F-statistic:                     2
59.1
Date:                   Fri, 21 Nov 2025   Prob (F-statistic):           1.75
e-41
Time:                           16:27:02   Log-Likelihood:                -36
2.88
No. Observations:                   276   AIC:                             7
29.8
Df Residuals:                       274   BIC:                             7
37.0
Df Model:                             1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          5.6224      0.141     39.966      0.000       5.345
5.899
d              0.5533      0.034     16.095      0.000       0.486
0.621
==============================================================================
====
Omnibus:                        2.476   Durbin-Watson:
1.289
Prob(Omnibus):                  0.290   Jarque-Bera (JB):
2.207
Skew:                          -0.125   Prob(JB):
0.332
Kurtosis:                       2.640   Cond. No.
11.1
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Engle–Granger residual ADF test:
ADF statistic: 0.487
p-value      : 0.984
Critical values: {'1%': -3.4554613060274972, '5%': -2.8725931472675046, '1
0%': -2.5726600403359887}
```

2.3) Campbell approach log(Dt/Pt)

In [8]:
```
adf_dp = adfuller(df_m["dp"], maxlag=12, regression='c')
dp_stat, dp_p, _, _, dp_crit, _ = adf_dp

print("\nADF test on dp_t = log(D) - log(P):")
print(f"ADF statistic: {dp_stat:.3f}")
print(f"p-value      : {dp_p:.3f}")
print("Critical values:", dp_crit)
```

```
ADF test on dp_t = log(D) - log(P):
ADF statistic: -1.793
p-value      : 0.384
Critical values: {'1%': -3.4554613060274972, '5%': -2.8725931472675046, '1
0%': -2.5726600403359887}
```

2.4) Engle-Granger for log Div12 / log Price

In [9]:
```
p = df_m["p"]
d = df_m["d12"]


X = sm.add_constant(d)
ols_res = sm.OLS(p, X).fit()
print(ols_res.summary())


u_hat = ols_res.resid

adf_stat, adf_p, _, _, crit_vals, _ = adfuller(u_hat, maxlag=12, regressi

print("\nEngle–Granger residual ADF test:")
print(f"ADF statistic: {adf_stat:.3f}")
print(f"p-value      : {adf_p:.3f}")
print("Critical values:", crit_vals)
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                      p   R-squared:
0.833
Model:                            OLS   Adj. R-squared:
0.832
Method:                 Least Squares   F-statistic:                     1
367.
Date:                Fri, 21 Nov 2025   Prob (F-statistic):          1.70e
-108
Time:                        16:27:08   Log-Likelihood:                -20
7.72
No. Observations:                 276   AIC:                             4
19.4
Df Residuals:                     274   BIC:                             4
26.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const          4.4461      0.040    112.186      0.000       4.368
4.524
d12            1.1168      0.030     36.971      0.000       1.057
1.176
==============================================================================
====
Omnibus:                       92.254   Durbin-Watson:
0.163
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              23
7.560
Skew:                          -1.560   Prob(JB):                     2.60
e-52
Kurtosis:                       6.305   Cond. No.
2.19
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.

Engle-Granger residual ADF test:
ADF statistic: -2.402
p-value      : 0.141
Critical values: {'1%': -3.4554613060274972, '5%': -2.8725931472675046, '1
0%': -2.5726600403359887}
```

2.5) Campbelle Approach for log(Div12/Price)

In [10]:
```
adf_dp = adfuller(df_m["d12p"], maxlag=12, regression='c')
dp_stat, dp_p, _, _, dp_crit, _ = adf_dp

print("\nADF test on dp_t = log(D12) - log(P):")
print(f"ADF statistic: {dp_stat:.3f}")
print(f"p-value      : {dp_p:.3f}")
print("Critical values:", dp_crit)
```

```
ADF test on dp_t = log(D12) - log(P):
ADF statistic: -1.575
p-value      : 0.496
Critical values: {'1%': -3.4554613060274972, '5%': -2.8725931472675046, '1
0%': -2.5726600403359887}
```

2.6) Test of Coint-Johansen

In [11]:
```
from statsmodels.tsa.vector_ar.vecm import coint_johansen
cj = coint_johansen(df_m[["p", "d12"]].dropna(), det_order=0, k_ar_diff=2

print("Trace statistics (lr1):", cj.lr1)
print("Trace critical values (cvt):")
print(cj.cvt)

print("Max-eig statistics (lr2):", cj.lr2)
print("Max-eig critical values (cvm):")
print(cj.cvm)

beta = cj.evec[:, 0]
print("Raw beta:", beta)

beta_p_norm = beta / beta[0]
print("Cointegrating vector (p = 1):", beta_p_norm)
```

```
Trace statistics (lr1): [15.26843524  0.1109057 ]
Trace critical values (cvt):
[[13.4294 15.4943 19.9349]
 [ 2.7055  3.8415  6.6349]]
Max-eig statistics (lr2): [15.15752954  0.1109057 ]
Max-eig critical values (cvm):
[[12.2971 14.2639 18.52  ]
 [ 2.7055  3.8415  6.6349]]
Raw beta: [ 1.7163229  -2.49491595]
Cointegrating vector (p = 1): [ 1.         -1.45364019]
```

FOUNDAMENTAL APPROACH - COCHRANE (1992) VAR MODEL EXTENSION

1) REQUIREMENTS SETUP

In [2]:
```
# !pip install -r requirements.txt
```

In [52]:
```
import warnings
warnings.filterwarnings("ignore")
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import STL
from statsmodels.graphics.tsaplots import plot_acf
import seaborn as sns
```

2) MONTHLY ANALYSIS

2.1) VARIABLES CONSTRUCTION

In [4]:
```
# Log Risk Free Rate
rf_df = pd.read_csv("../data_extraction/raw_df/risk_free_monthly_df.csv")
rf_df = rf_df.rename(columns={"1-month Yield - US Treasury Securities":"R
rf_df["Date"] = pd.to_datetime(rf_df["Date"])

# Data Cleaning for NaN, -inf, 0.0
rf_df["Rf_t"] = pd.to_numeric(rf_df["Rf_t"], errors="coerce")
rf_df["Rf_t"] = rf_df["Rf_t"].replace([0, 0.0, -np.inf, np.inf], np.nan)

# (!!!) Either forward or backward filling
rf_df["Rf_t"] = rf_df["Rf_t"].ffill().bfill()

rf_df["rf_t"] = np.log(rf_df["Rf_t"])

rf_df.tail()
```

Out[4]:

	Date	Rf_t	rf_t
282	2025-07-01	4.32	1.463255
283	2025-08-01	4.49	1.501853
284	2025-09-01	4.41	1.483875
285	2025-10-01	4.17	1.427916
286	2025-11-01	4.06	1.401183

In [5]:
```
# Log Synthetic Index Price
p_df = pd.read_csv("../data_extraction/raw_df/synthetic_price_monthly_df.
p_df["p_t"] = np.log(p_df["Synthetic Index Close Price"])
p_df = p_df.rename(columns={"Synthetic Index Close Price":"P_t"})
p_df["Date"] = pd.to_datetime(p_df["Date"].astype(str).str[:10], format="
p_df.tail()
```

Out[5]:

	Date	P_t	p_t
282	2025-07-01	333.452353	5.809500
283	2025-08-01	340.753141	5.831158
284	2025-09-01	348.811723	5.854532
285	2025-10-01	366.551887	5.904140
286	2025-11-01	375.118033	5.927241

In [6]:
```
# Log Synthetic Index Dividend - Seasonally Adjusted
d_df = pd.read_csv("../data_extraction/raw_df/synthetic_div_monthly_df.cs

# Dividends payout with rolling window=12
d_df["D_t"] = d_df["Synthetic Index Dividend"].rolling(window=12, min_per

# Log of D_t
d_df["d_t"] = np.log(d_df["D_t"])
d_df = d_df.drop(columns=["Synthetic Index Dividend"])

d_df["Date"] = pd.to_datetime(d_df["Date"].astype(str).str[:10], format="
d_df = d_df.dropna()
d_df
```

Out[6]:

	Date	D_t	d_t
11	2002-12-01	0.033602	-3.393184
12	2003-01-01	0.033601	-3.393210
13	2003-02-01	0.046343	-3.071691
14	2003-03-01	0.046835	-3.061119
15	2003-04-01	0.046006	-3.078981
...	...	...	...
282	2025-07-01	2.022016	0.704095
283	2025-08-01	2.059450	0.722439
284	2025-09-01	2.074903	0.729915
285	2025-10-01	2.101338	0.742574
286	2025-11-01	1.918976	0.651791

276 rows × 3 columns

In [7]:
```
# Merged Variable Dataset
monthly_df = pd.merge(p_df, d_df, on="Date", how="outer")
monthly_df = pd.merge(monthly_df, rf_df, on="Date", how="outer")

monthly_df
```

Out[7]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t
0	2002-01-01	6.741954	1.908350	NaN	NaN	1.69	0.524729
1	2002-02-01	6.708014	1.903303	NaN	NaN	1.69	0.524729
2	2002-03-01	6.614498	1.889264	NaN	NaN	1.78	0.576613
3	2002-04-01	6.714867	1.904324	NaN	NaN	1.79	0.582216
4	2002-05-01	6.022081	1.795433	NaN	NaN	1.76	0.565314
...	...	...	...	...	...	...	...
282	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255
283	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853
284	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875
285	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916
286	2025-11-01	375.118033	5.927241	1.918976	0.651791	4.06	1.401183

287 rows × 7 columns

In [8]:
```
# Log Gross Return (Approximation)
monthly_df["r_t+1"] = monthly_df["p_t"].shift(-1) - monthly_df["p_t"] + n
monthly_df
```

Out[8]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+1
0	2002-01-01	6.741954	1.908350	NaN	NaN	1.69	0.524729	NaN
1	2002-02-01	6.708014	1.903303	NaN	NaN	1.69	0.524729	NaN
2	2002-03-01	6.614498	1.889264	NaN	NaN	1.78	0.576613	NaN
3	2002-04-01	6.714867	1.904324	NaN	NaN	1.79	0.582216	NaN
4	2002-05-01	6.022081	1.795433	NaN	NaN	1.76	0.565314	NaN
...	...	...	...	...	...	...	...	...
282	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.027815
283	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.029445
284	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.055614
285	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.028322
286	2025-11-01	375.118033	5.927241	1.918976	0.651791	4.06	1.401183	NaN

287 rows × 8 columns

In [9]:
```
# Log Excess Return
monthly_df["rx_t+1"] = monthly_df["r_t+1"] - monthly_df["rf_t"].shift(-1)
monthly_df
```

Out[9]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+1
0	2002-01-01	6.741954	1.908350	NaN	NaN	1.69	0.524729	NaN
1	2002-02-01	6.708014	1.903303	NaN	NaN	1.69	0.524729	NaN
2	2002-03-01	6.614498	1.889264	NaN	NaN	1.78	0.576613	NaN
3	2002-04-01	6.714867	1.904324	NaN	NaN	1.79	0.582216	NaN
4	2002-05-01	6.022081	1.795433	NaN	NaN	1.76	0.565314	NaN
...	...	...	...	...	...	...	...	...
282	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.027815
283	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.029445
284	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.055614
285	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.028322
286	2025-11-01	375.118033	5.927241	1.918976	0.651791	4.06	1.401183	NaN

287 rows × 9 columns

In [10]:
```
# Log Excess Return
monthly_df["Δd_t+1"] = monthly_df["d_t"].shift(-1) - monthly_df["d_t"]
monthly_df
```

Out[10]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+1
0	2002-01-01	6.741954	1.908350	NaN	NaN	1.69	0.524729	NaN
1	2002-02-01	6.708014	1.903303	NaN	NaN	1.69	0.524729	NaN
2	2002-03-01	6.614498	1.889264	NaN	NaN	1.78	0.576613	NaN
3	2002-04-01	6.714867	1.904324	NaN	NaN	1.79	0.582216	NaN
4	2002-05-01	6.022081	1.795433	NaN	NaN	1.76	0.565314	NaN
...	...	...	...	...	...	...	...	...
282	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.027815
283	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.029445
284	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.055614
285	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.028322
286	2025-11-01	375.118033	5.927241	1.918976	0.651791	4.06	1.401183	NaN

287 rows × 10 columns

In [11]:

```
# Log Dividend-Price Ratio
monthly_df["dp_t"] = monthly_df["d_t"] - monthly_df["p_t"]

# Remove obs/date for the first lagged obs
monthly_df = monthly_df.dropna()
monthly_df
```

Out[11]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+
11	2002-12-01	5.496798	1.704166	0.033602	-3.393184	1.25	0.223144	-0.11704
12	2003-01-01	4.859930	1.581024	0.033601	-3.393210	1.20	0.182322	-0.04453
13	2003-02-01	4.604349	1.527001	0.046343	-3.071691	1.17	0.157004	0.02054
14	2003-03-01	4.652612	1.537429	0.046835	-3.061119	1.21	0.190620	0.02463
15	2003-04-01	4.721963	1.552225	0.046006	-3.078981	1.17	0.157004	0.08830
...	...	...	...	...	...	...	...	.
281	2025-06-01	314.297737	5.750341	2.034056	0.710032	4.33	1.465568	0.06557
282	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.02781
283	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.02944
284	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.05561
285	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.02832

275 rows × 11 columns

2.2) TRAIN/TEST SPLIT

In [12]:
```
# Train & Test Split
monthly_df.to_csv("FVM_data/raw_monthly_df.csv", index=False)

train_monthly_df = monthly_df[monthly_df["Date"] <= "2021-12-31"]
train_monthly_df.to_csv("FVM_data/train_monthly_df.csv", index=False)

test_monthly_df = monthly_df[monthly_df["Date"] >= "2021-12-31"]
test_monthly_df.to_csv("FVM_data/test_monthly_df.csv", index=False)
```

2.3) VAR MODEL

In [13]:
```
# Clean the df for state space vector
# (!!!) Transfrom date in index and covert value columns to numeric
train_monthly_df = train_monthly_df[["Date", "rx_t+1", "Δd_t+1", "dp_t"]]
train_monthly_df = train_monthly_df.set_index("Date")
train_monthly_df = train_monthly_df.apply(pd.to_numeric, errors="coerce")
```

```
train_monthly_df.tail()
```

Out[13]:

Date	rx_t+1	Δd_t+1	dp_t
2021-08-01	3.285507	0.004778	-4.967826
2021-09-01	2.478184	0.002502	-5.022711
2021-10-01	3.061767	0.019916	-4.966083
2021-11-01	2.411935	-0.131318	-5.005116
2021-12-01	2.872358	0.001693	-5.134562

In [14]:
```
# VAR model lag optimization
# (!!!) We prioritize AIC and FPE (lag=5) as we are interest in the best
# (!!!) Parsimony-based index instead suggest a lag=2

m_model = VAR(train_monthly_df)
lag_selection = m_model.select_order(maxlags=10)
print(lag_selection.summary())
```

```
 VAR Order Selection (* highlights the minimums)
=================================================
       AIC         BIC         FPE         HQIC
-------------------------------------------------
0       -3.283      -3.236     0.03752      -3.264
1       -9.989     -9.804*   4.589e-05      -9.914
2      -10.09*      -9.761  4.166e-05*     -9.955*
3       -10.04      -9.571   4.383e-05      -9.848
4       -9.972      -9.368   4.672e-05      -9.728
5       -9.909      -9.166   4.978e-05      -9.609
6       -9.861      -8.979   5.225e-05      -9.504
7       -9.812      -8.791   5.488e-05      -9.400
8       -9.762      -8.601   5.779e-05      -9.293
9       -9.686      -8.386   6.241e-05      -9.161
10      -9.657      -8.218   6.436e-05      -9.076
-------------------------------------------------
```

```
d:\Conda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWar
ning: No frequency information was provided, so inferred frequency MS will
be used.
  self._init_dates(dates, freq)
```

In [15]:
```
# VAR model (lag=2)
VAR_m_model = m_model.fit(2)
print(VAR_m_model.summary())
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Sun, 23, Nov, 2025
Time:                     06:02:34
```

```
-------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                   -9.84648
Nobs:                     227.000    HQIC:                  -10.0355
Log likelihood:           208.240    FPE:                3.85612e-05
AIC:                     -10.1633    Det(Omega_mle):     3.52030e-05
-------------------------------------------------------------------
Results for equation rx_t+1
===================================================================
==
               coefficient       std. error           t-stat            pr
ob
-------------------------------------------------------------------
--
const           -0.131247         0.368794          -0.356           0.7
22
L1.rx_t+1        0.648430         0.064504          10.053           0.0
00
L1.Δd_t+1       -0.117623         0.208780          -0.563           0.5
73
L1.dp_t          0.296085         0.789844           0.375           0.7
08
L2.rx_t+1        0.316029         0.064606           4.892           0.0
00
L2.Δd_t+1       -0.278022         0.813822          -0.342           0.7
33
L2.dp_t         -0.343510         0.792387          -0.434           0.6
65
===================================================================
==

Results for equation Δd_t+1
===================================================================
==
               coefficient       std. error           t-stat            pr
ob
-------------------------------------------------------------------
--
const           -0.317674         0.118652          -2.677           0.0
07
L1.rx_t+1        0.002384         0.020753           0.115           0.9
09
L1.Δd_t+1       -0.107718         0.067171          -1.604           0.1
09
L1.dp_t         -0.229319         0.254115          -0.902           0.3
67
L2.rx_t+1       -0.002486         0.020786          -0.120           0.9
05
L2.Δd_t+1        0.177008         0.261830           0.676           0.4
99
L2.dp_t          0.150958         0.254934           0.592           0.5
54
===================================================================
==
```

```
Results for equation dp_t
================================================================================
==
               coefficient       std. error           t-stat            pr
ob
--------------------------------------------------------------------------------
--
const            -0.017018         0.030504           -0.558           0.5
77
L1.rx_t+1         0.013877         0.005335            2.601           0.0
09
L1.Δd_t+1         0.996185         0.017269           57.687           0.0
00
L1.dp_t           1.037454         0.065330           15.880           0.0
00
L2.rx_t+1        -0.016476         0.005344           -3.083           0.0
02
L2.Δd_t+1        -0.036437         0.067314           -0.541           0.5
88
L2.dp_t          -0.038577         0.065541           -0.589           0.5
56
================================================================================
==

Correlation matrix of residuals
            rx_t+1    Δd_t+1      dp_t
rx_t+1    1.000000 -0.013657  0.196843
Δd_t+1   -0.013657  1.000000 -0.030429
dp_t      0.196843 -0.030429  1.000000
```

2.4) MODEL TRAINING

In [16]:
```python
# Data Training Function
# (!!!) Change model specification
def data_training(model, original_df):
    train_fitted = model.fittedvalues.copy()

# Rename columns for fitted data and create a new df
# (!!!) The time filtering is already performed by "inner"
    train_fitted.columns = [c + "_fitted" for c in train_fitted.columns]
    train_and_fit = pd.merge(original_df, train_fitted, on="Date", how="i

# Train Results Dataset
    train_results = train_and_fit.copy()

#////////////////////////////////////////////////////////////
# Dividend Extraction
# Train D_t=0: 0.03360150420168
# Test D_t=0: 1.0887532677258398
    growth = np.exp(train_results["Δd_t+1_fitted"])
```

```
    D = np.empty(len(train_results))
    D0_train = 0.03360150420168
    for t in range(len(train_results)):
        if t == 0:
            D[t] = D0_train * growth.iloc[t]
        else:
            D[t] = D[t-1] * growth.iloc[t]
    train_results["D_t_PVM"] = D
# Price Extraction: pt=Dt/e^dpt
    train_results["P_t_PVM"] = train_results["D_t_PVM"] / np.exp(train_re
#////////////////////////////////////////////////////////////

# Mismatch Measures
    train_results["Mispricing Actual-PVM"] = train_results["P_t"] - train
    train_results["Mispricing Ratio PVM"] = train_results["P_t"] / train_
    train_results["Nature"] = "In-Sample Training"

    return train_results
```

In [17]:
```
# In-Sample Data Training Monthly (until 2021/12/01)

monthly_train_results = data_training(VAR_m_model, monthly_df)
monthly_train_results
```

Out[17]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+
0	2003-02-01	4.604349	1.527001	0.046343	-3.071691	1.17	0.157004	0.02054
1	2003-03-01	4.652612	1.537429	0.046835	-3.061119	1.21	0.190620	0.02463
2	2003-04-01	4.721963	1.552225	0.046006	-3.078981	1.17	0.157004	0.08830
3	2003-05-01	5.108140	1.630835	0.046003	-3.079057	1.09	0.086178	0.05222
4	2003-06-01	5.333300	1.673970	0.046635	-3.065396	1.16	0.148420	0.03352
...	...	...	...	...	...	...	...	
222	2021-08-01	173.639478	5.156981	1.208228	0.189155	0.05	-2.995732	0.06663
223	2021-09-01	184.314724	5.216645	1.214015	0.193933	0.04	-3.218876	-0.04754
224	2021-10-01	174.603609	5.162518	1.217056	0.196435	0.08	-2.525729	0.06603
225	2021-11-01	185.205830	5.221468	1.241539	0.216352	0.05	-2.995732	0.00398
226	2021-12-01	184.859378	5.219595	1.088753	0.085033	0.09	-2.407946	0.05894

227 rows × 19 columns

2.4) MODEL FORECAST

In [18]:
```
# Out-Sample Data Training (2022/12/01 - 2025/11/01)
# (!!!) Change model specification
def data_testing(model, original_df, test_df):

# General set up of the VAR specification
# Get variables' names
```

```
    df = test_df.copy().reset_index(drop=True)
    var_cols = model.names
# VAR lag order
    p = model.k_ar
# (!!!) Initial history comes from the training sample inside the VAR mod
    history = model.model.endog[-p:].copy()
    predictions = []

# Actual Forecast
# Forecast for each period t
    for t in range(len(df)):
        forecast = model.forecast(history[-p:], steps=1)[0]
        predictions.append(forecast)
# (!!!) We append this prediction to history (recursive forecasting)
        history = np.vstack([history, forecast])

# Predictions df
    pred_results = pd.DataFrame(predictions,
                                columns=[c + "_fitted" for c in var_cols])
    pred_results["Date"] = df["Date"].values


# Rename columns for fitted data and create a new df
# (!!!) The time filtering is already performed by "inner"
    test_and_fit = pd.merge(original_df, pred_results, on="Date", how="in

# Test results dataset
    test_results = test_and_fit.copy()

#////////////////////////////////////////////////
# Dividend Extraction
# Train D_t=0: 0.03360150420168
# Test D_t=0: 0.7907552641165884
    growth = np.exp(test_results["Δd_t+1_fitted"])
    D = np.empty(len(test_results))
    D0_test = 0.7907552641165884
    for t in range(len(test_results)):
        if t == 0:
            D[t] = D0_test * growth.iloc[t]
        else:
            D[t] = D[t-1] * growth.iloc[t]
    test_results["D_t_PVM"] = D
# Price Extraction: pt=Dt/e^dpt
    test_results["P_t_PVM"] = test_results["D_t_PVM"] / np.exp(test_resul
#////////////////////////////////////////////////

# Mismatch Measures
    test_results["Mispricing Actual-PVM"] = test_results["P_t"] - test_re
    test_results["Mispricing Ratio PVM"] = test_results["P_t"] / test_res
    test_results["Nature"] = "Out-Sample Testing"

    return test_results
```

In [19]:
```
# Out-Sample Data Testing Monthly (2022/12/01 - 2025/11/01)

monthly_test_results = data_testing(VAR_m_model, monthly_df, test_monthly
monthly_test_results
```

Out[19]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+1
0	2022-01-01	194.933937	5.272661	1.090598	0.086727	0.06	-2.813411	-0.06217
1	2022-02-01	182.146812	5.204813	1.109689	0.104080	0.04	-3.218876	-0.086294
2	2022-03-01	166.071109	5.112416	1.115086	0.108932	0.11	-2.207275	0.07495
3	2022-04-01	177.805083	5.180688	1.112932	0.106998	0.15	-1.897120	-0.113726
4	2022-05-01	157.651267	5.060385	1.173191	0.159728	0.37	-0.994252	-0.017423
5	2022-06-01	153.777213	5.035505	1.180041	0.165549	0.77	-0.261365	-0.086108
6	2022-07-01	140.012177	4.941729	1.183635	0.168590	1.27	0.239017	0.111256
7	2022-08-01	155.123465	5.044221	1.232483	0.209031	2.22	0.797507	-0.042683
8	2022-09-01	147.468548	4.993615	1.234042	0.210295	2.53	0.928219	-0.111398
9	2022-10-01	130.824974	4.873860	1.237479	0.213076	2.79	1.026042	0.038879
10	2022-11-01	134.722134	4.903214	1.252109	0.224829	3.72	1.313724	0.103433
11	2022-12-01	148.023632	4.997372	1.255485	0.227522	4.04	1.396245	-0.068885
12	2023-01-01	137.007771	4.920038	1.256044	0.227967	4.12	1.415853	0.132502

13	2023-02-01	154.949231	5.043098	1.299832	0.262235	4.59	1.523880	-0.009830
14	2023-03-01	152.146800	5.024846	1.310439	0.270362	4.67	1.541159	0.121677
15	2023-04-01	170.380362	5.138033	1.297146	0.260167	4.74	1.556037	0.036631
16	2023-05-01	175.404001	5.167092	1.295179	0.258649	4.49	1.501853	0.082553
17	2023-06-01	189.083761	5.242190	1.312482	0.271920	5.30	1.667707	0.043279
18	2023-07-01	196.098921	5.278619	1.299605	0.262060	5.24	1.656321	0.043339
19	2023-08-01	203.399115	5.315170	1.335627	0.289401	5.49	1.702928	-0.000655
20	2023-09-01	201.938977	5.307966	1.336490	0.290047	5.51	1.706565	-0.040553
21	2023-10-01	192.650782	5.260879	1.323586	0.280345	5.55	1.713798	0.021954
22	2023-11-01	195.530132	5.275714	1.376308	0.319404	5.56	1.715598	0.097824
23	2023-12-01	213.852082	5.365285	1.620575	0.482781	5.55	1.713798	0.057244
24	2024-01-01	224.745875	5.414970	1.622558	0.484004	5.60	1.722767	0.062936
25	2024-02-01	237.563436	5.470435	1.685579	0.522109	5.49	1.702928	0.068305
26	2024-03-01	252.509099	5.531447	1.738681	0.553127	5.54	1.711995	0.019374
27	2024-	255.639526	5.543768	1.787117	0.580604	5.49	1.702928	-0.045787

	04-01							
28	2024-05-01	242.522846	5.491096	1.766361	0.568922	5.47	1.699279	0.099190
29	2024-06-01	265.756687	5.582581	1.875715	0.628990	5.48	1.701105	0.085746
30	2024-07-01	287.470793	5.661121	1.921937	0.653333	5.48	1.701105	-0.052706
31	2024-08-01	270.930621	5.601863	1.889781	0.636461	5.55	1.713798	0.048440
32	2024-09-01	282.265548	5.642848	2.027120	0.706616	5.41	1.688249	0.018142
33	2024-10-01	285.355975	5.653737	2.054723	0.720141	4.96	1.601406	0.016040
34	2024-11-01	287.919228	5.662680	2.032461	0.709248	4.75	1.558145	0.037760
35	2024-12-01	297.012735	5.693775	1.925293	0.655078	4.76	1.560248	0.019809
36	2025-01-01	301.000963	5.707113	1.928164	0.656568	4.40	1.481605	0.023601
37	2025-02-01	306.250725	5.724404	1.905327	0.644654	4.37	1.474763	-0.008058
38	2025-03-01	301.829542	5.709862	1.991963	0.689121	4.38	1.477049	-0.072082
39	2025-04-01	279.002883	5.631222	1.985953	0.686099	4.38	1.477049	0.043166
40	2025-05-01	289.223774	5.667201	2.012641	0.699448	4.38	1.477049	0.090148
41	2025-06-01	314.297737	5.750341	2.034056	0.710032	4.33	1.465568	0.065572
	2025-							

42	07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.027815
43	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.029445
44	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.055614
45	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.028322

In [41]:
```
# Concat results dfs
monthly_fitted_df = pd.concat([monthly_train_results, monthly_test_result
                               axis=0,
                               ignore_index=True)
monthly_fitted_df.to_csv("FVM_data/monthly_fitted_df.csv")
monthly_fitted_df
```

Out[41]:

	Date	P_t	p_t	D_t	d_t	Rf_t	rf_t	r_t+1
0	2003-02-01	4.604349	1.527001	0.046343	-3.071691	1.17	0.157004	0.020548
1	2003-03-01	4.652612	1.537429	0.046835	-3.061119	1.21	0.190620	0.024635
2	2003-04-01	4.721963	1.552225	0.046006	-3.078981	1.17	0.157004	0.088306
3	2003-05-01	5.108140	1.630835	0.046003	-3.079057	1.09	0.086178	0.052223
4	2003-06-01	5.333300	1.673970	0.046635	-3.065396	1.16	0.148420	0.033527
...	...	...	...	...	...	...	...	..
268	2025-06-01	314.297737	5.750341	2.034056	0.710032	4.33	1.465568	0.065572
269	2025-07-01	333.452353	5.809500	2.022016	0.704095	4.32	1.463255	0.027815
270	2025-08-01	340.753141	5.831158	2.059450	0.722439	4.49	1.501853	0.029445
271	2025-09-01	348.811723	5.854532	2.074903	0.729915	4.41	1.483875	0.055614
272	2025-10-01	366.551887	5.904140	2.101338	0.742574	4.17	1.427916	0.028322

273 rows × 19 columns

In [86]:
```
# Summary Statistics
monthly_fitted_df.describe()
```

Out[86]:

	Date	P_t	p_t	D_t	d_t	
count	273	273.000000	273.000000	273.000000	273.000000	273.0
mean	2014-06-01 08:42:11.868131840	71.039983	3.539590	0.673128	-0.802734	1.6
min	2003-02-01 00:00:00	4.604349	1.527001	0.045918	-3.080899	0.0
25%	2008-10-01 00:00:00	11.205518	2.416406	0.200336	-1.607757	0.0
50%	2014-06-01 00:00:00	34.824557	3.550323	0.607705	-0.498066	0.8
75%	2020-02-01 00:00:00	94.677412	4.550475	0.962000	-0.038741	2.7
max	2025-10-01 00:00:00	366.551887	5.904140	2.101338	0.742574	5.6
std	NaN	84.654065	1.247178	0.535254	1.006784	1.8

2.5) METRICS

In [ ]:
```
# Performance Metrics
# (!!!) ngl, impressive metrics, especially for out-sample testing
# (!!!) Previous alternative models had abnormal results (check lit)

def compute_metrics(df, label):
    df = df.dropna(subset=["P_t", "P_t_PVM"])
# RMSE (Root Mean Squared Error)
    rmse = np.sqrt(np.mean((df["P_t_PVM"] - df["P_t"])**2))
# MAPE (Mean Absolute Percentage Error)
    mape = np.mean(np.abs((df["P_t_PVM"] - df["P_t"]) / df["P_t"])) * 100
# CORR (Correlation (Actual vs PVM Price))
    corr = df["P_t"].corr(df["P_t_PVM"])
# SS_RES
    ss_res = np.sum((df["P_t"] - df["P_t_PVM"])**2)
# SS_TOT
    ss_tot = np.sum((df["P_t"] - df["P_t"].mean())**2)
# R2 (Pseudo R²)
    r2 = 1 - ss_res / ss_tot
    print(f"\n=== {label} PVM Performance ===")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAPE: {mape:.2f}%")
    print(f"Correlation (P_t vs P_t_PVM): {corr:.4f}")
    print(f"Pseudo R²: {r2:.4f}")

compute_metrics(monthly_train_results, "In-Sample (train)")
compute_metrics(monthly_test_results, "Out-Sample (test)")
```

```
=== In-Sample (train) PVM Performance ===
RMSE: 41.4856
MAPE: 76.72%
Correlation (P_t vs P_t_PVM): 0.8398
Pseudo R²: 0.0146

=== Out-Sample (test) PVM Performance ===
RMSE: 21.9991
MAPE: 9.46%
Correlation (P_t vs P_t_PVM): 0.9540
Pseudo R²: 0.8926
```

In [85]:
```python
# Martingale benchmark Optimization
# (!!!) Best threshold (lag=4), our model is better
martingale_df = monthly_test_results

# Martingale benchmark - P_t^M = P_{t-1}
martingale_df["P_t_martingale"] = martingale_df["P_t"].shift(4)
martingale_df.loc[martingale_df.index[0], "P_t_martingale"] = martingale_

def compute_metrics(df, label):
    df = df.dropna(subset=["P_t", "P_t_martingale"])
# RMSE (Root Mean Squared Error)
    rmse = np.sqrt(np.mean((df["P_t_martingale"] - df["P_t"])**2))
# MAPE (Mean Absolute Percentage Error)
    mape = np.mean(np.abs((df["P_t_martingale"] - df["P_t"]) / df["P_t"]))
# CORR (Correlation (Actual vs PVM Price))
    corr = df["P_t"].corr(df["P_t_martingale"])
# SS_RES
    ss_res = np.sum((df["P_t"] - df["P_t_martingale"])**2)
# SS_TOT
    ss_tot = np.sum((df["P_t"] - df["P_t"].mean())**2)
# R2 (Pseudo R²)
    r2 = 1 - ss_res / ss_tot
    print(f"\n=== {label} Martingale Performance Optimization ===")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAPE: {mape:.2f}%")
    print(f"Correlation (P_t vs P_t_martingale): {corr:.4f}")
    print(f"Pseudo R²: {r2:.4f}")

compute_metrics(martingale_df, "Out-Sample (test)")
```

```
=== Out-Sample (test) Martingale Performance Optimization ===
RMSE: 29.5014
MAPE: 11.12%
Correlation (P_t vs P_t_martingale): 0.9331
Pseudo R²: 0.8115
```

In [80]:
```python
# Martingale benchmark Semestral
martingale_df = monthly_test_results

# Martingale benchmark - P_t^M = P_{t-1}
martingale_df["P_t_martingale"] = martingale_df["P_t"].shift(6)
martingale_df.loc[martingale_df.index[0], "P_t_martingale"] = martingale_
```

```
def compute_metrics(df, label):
    df = df.dropna(subset=["P_t", "P_t_martingale"])
# RMSE (Root Mean Squared Error)
    rmse = np.sqrt(np.mean((df["P_t_martingale"] - df["P_t"])**2))
# MAPE (Mean Absolute Percentage Error)
    mape = np.mean(np.abs((df["P_t_martingale"] - df["P_t"]) / df["P_t"]))
# CORR (Correlation (Actual vs PVM Price))
    corr = df["P_t"].corr(df["P_t_martingale"])
# SS_RES
    ss_res = np.sum((df["P_t"] - df["P_t_martingale"])**2)
# SS_TOT
    ss_tot = np.sum((df["P_t"] - df["P_t"].mean())**2)
# R2 (Pseudo R²)
    r2 = 1 - ss_res / ss_tot
    print(f"\n=== {label} Martingale Performance Lag=6 ===")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAPE: {mape:.2f}%")
    print(f"Correlation (P_t vs P_t_martingale): {corr:.4f}")
    print(f"Pseudo R²: {r2:.4f}")

compute_metrics(martingale_df, "Out-Sample (test)")
```

```
=== Out-Sample (test) Martingale Performance Lag=6 ===
RMSE: 37.9882
MAPE: 14.43%
Correlation (P_t vs P_t_martingale): 0.8995
Pseudo R²: 0.6832
```

In [78]:
```
# Martingale benchmark annual
martingale_df = monthly_test_results

# Martingale benchmark - P_t^M = P_{t-1}
martingale_df["P_t_martingale"] = martingale_df["P_t"].shift(12)
martingale_df.loc[martingale_df.index[0], "P_t_martingale"] = martingale_

def compute_metrics(df, label):
    df = df.dropna(subset=["P_t", "P_t_martingale"])
# RMSE (Root Mean Squared Error)
    rmse = np.sqrt(np.mean((df["P_t_martingale"] - df["P_t"])**2))
# MAPE (Mean Absolute Percentage Error)
    mape = np.mean(np.abs((df["P_t_martingale"] - df["P_t"]) / df["P_t"]))
# CORR (Correlation (Actual vs PVM Price))
    corr = df["P_t"].corr(df["P_t_martingale"])
# SS_RES
    ss_res = np.sum((df["P_t"] - df["P_t_martingale"])**2)
# SS_TOT
    ss_tot = np.sum((df["P_t"] - df["P_t"].mean())**2)
# R2 (Pseudo R²)
    r2 = 1 - ss_res / ss_tot
    print(f"\n=== {label} Martingale Performance Lag=12 ===")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAPE: {mape:.2f}%")
    print(f"Correlation (P_t vs P_t_martingale): {corr:.4f}")
```

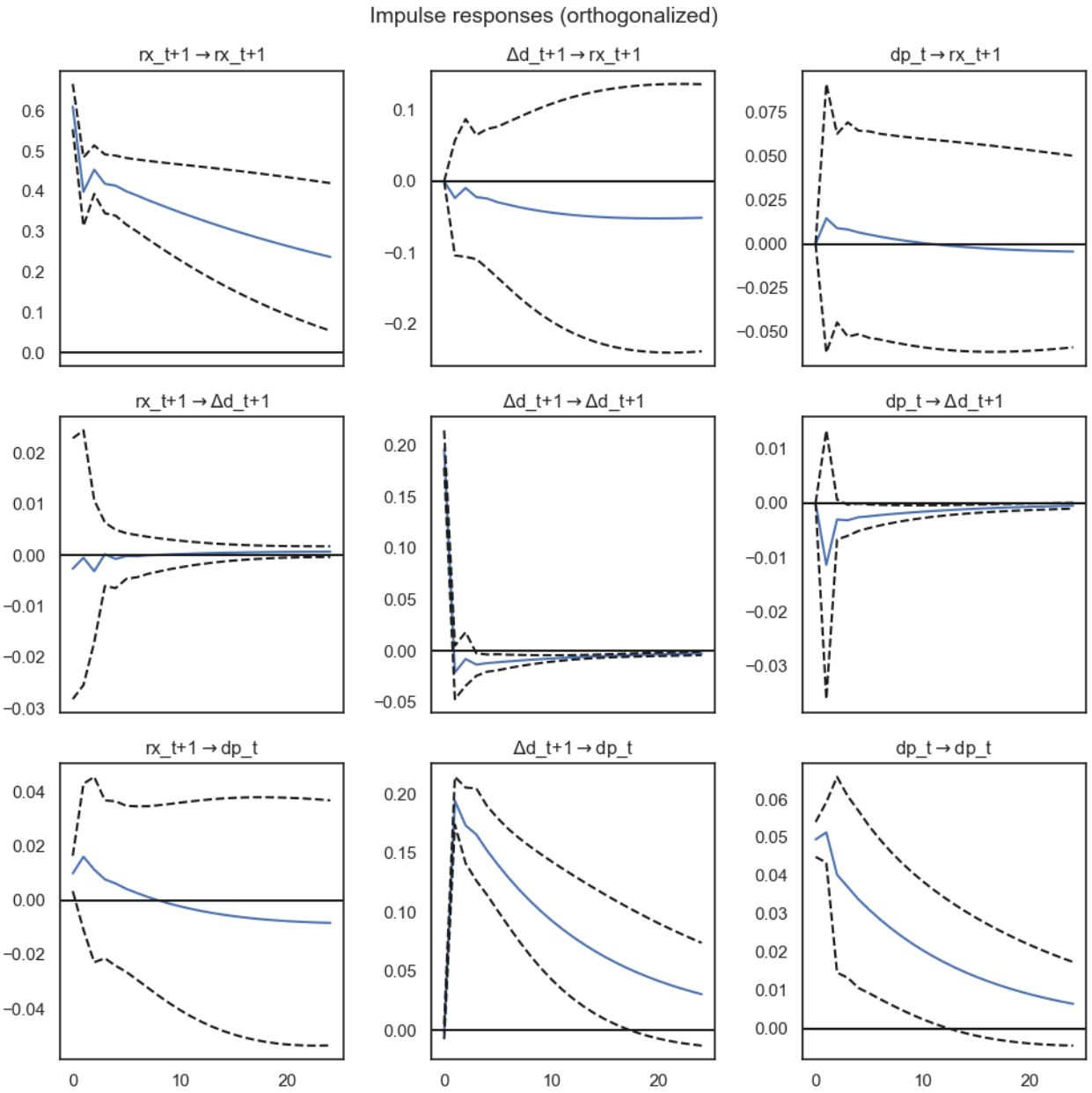
```
    print(f"Pseudo R²: {r2:.4f}")

compute_metrics(martingale_df, "Out-Sample (test)")
```

```
=== Out-Sample (test) Martingale Performance Lag=12 ===
RMSE: 65.0093
MAPE: 23.89%
Correlation (P_t vs P_t_martingale): 0.7937
Pseudo R²: -0.1610
```

2.6) PLOTTING

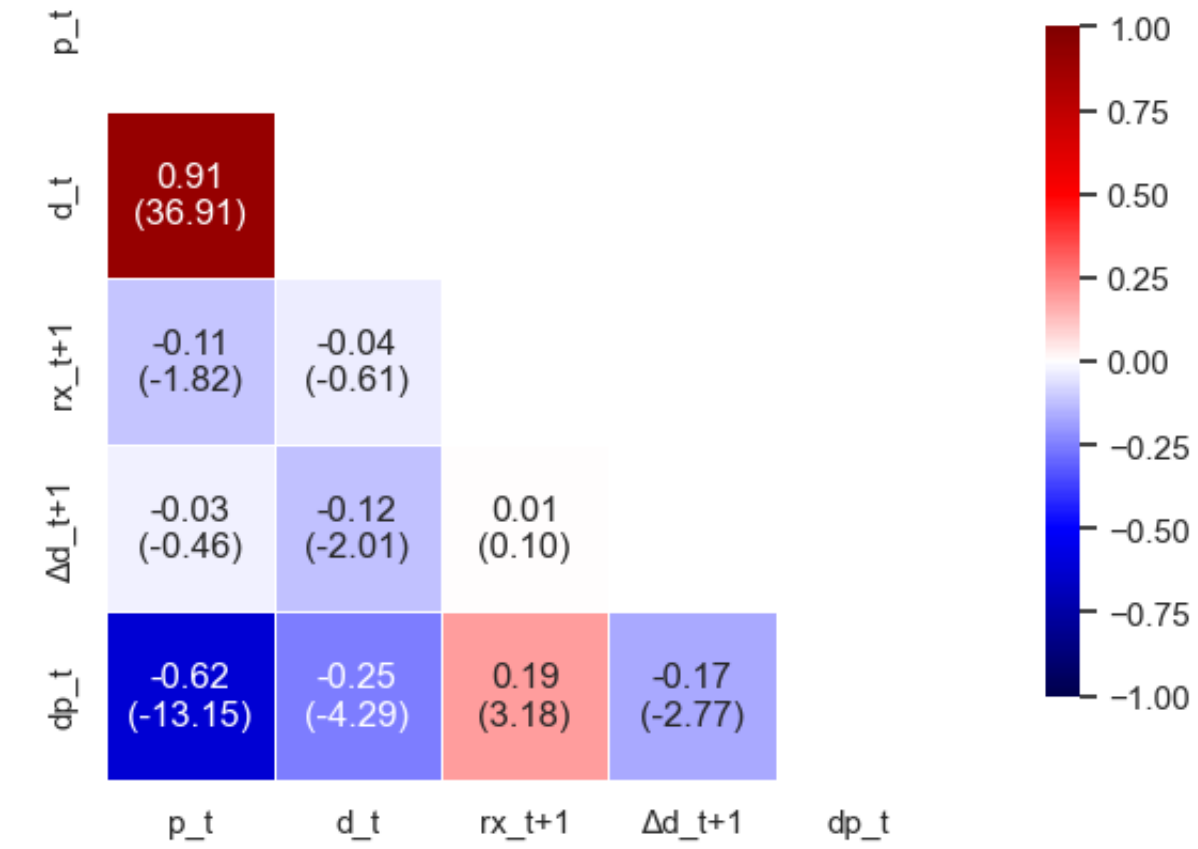
In [51]:
```
# VAR impulse response function plotting
# (!!!) Responses die after 15 month on average: stable VAR
# (!!!) Smooth and monotonic reactions (no explosivity)
# (!!!)

# Horizon for IRFs
# (!!!) Reaction over the next 24 months
irf_horizon = 24
# IRFs computation per each variable
irf = VAR_m_model.irf(irf_horizon)


plt.figure(figsize=(10, 5))
irf.plot(orth=True)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```



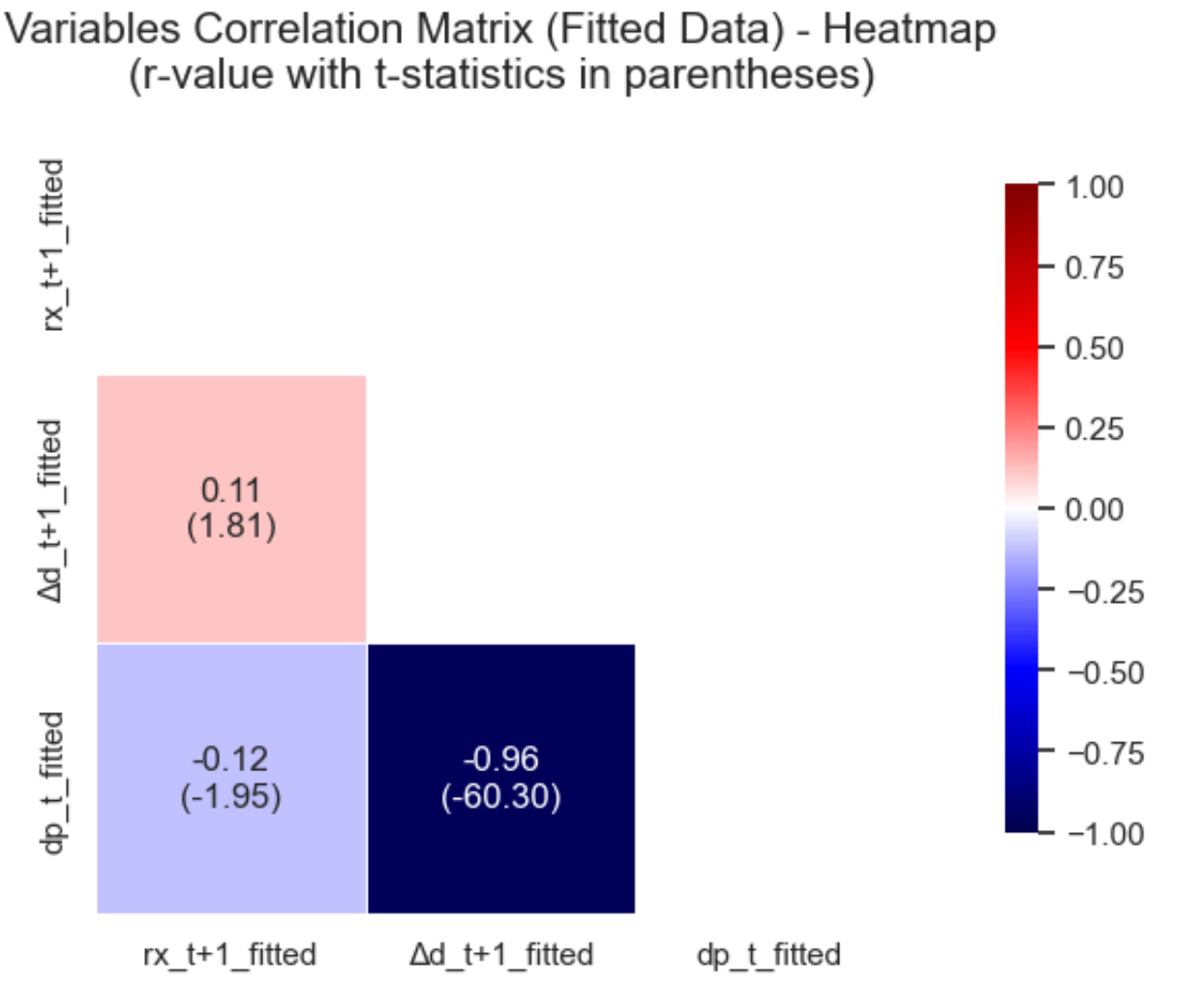
In [43]:
```
# Actual Data Plotting (Correlation Heatmap)
filtered_fitted_monthly_df = monthly_fitted_df[["p_t", "d_t", "rx_t+1", "

# Correlation matrix
corr_matrix = filtered_fitted_monthly_df.corr()
# Sample size
n = filtered_fitted_monthly_df.shape[0]

# t-statistics derived from correlation values
with np.errstate(divide="ignore", invalid="ignore"):
    t_stat_matrix = corr_matrix * np.sqrt((n - 2) / (1 - corr_matrix**2))
    t_stat_matrix = t_stat_matrix.round(2)

# For each cell, we want to have both the correlation index, as well as t
annot_matrix = corr_matrix.copy().astype(str)

for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix)):
```

```
# We only want to keep the lower triangle and diagonal of the full correl
        if i >= j:
            r = corr_matrix.iloc[i, j]
            t = t_stat_matrix.iloc[i, j]
            annot_matrix.iloc[i, j] = f"{r:.2f}\n({t:.2f})"
        else:
            annot_matrix.iloc[i, j] = ""

# We manually hide the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Heat-map plot
# General Layout (figure's size and style)
plt.figure(figsize=(10, 5))
sns.set(style="white")

sns.heatmap(corr_matrix,
            mask=mask,
            annot=annot_matrix,
            fmt="",
            cmap="seismic",
            vmin=-1, vmax=1,
            square=True,
            linewidths=0.5,
            cbar_kws={"shrink": .8})


plt.title("Variables Correlation Matrix (Actual Data) - Heatmap\n(r-value
          fontsize=15)
plt.tight_layout()
plt.show()
```

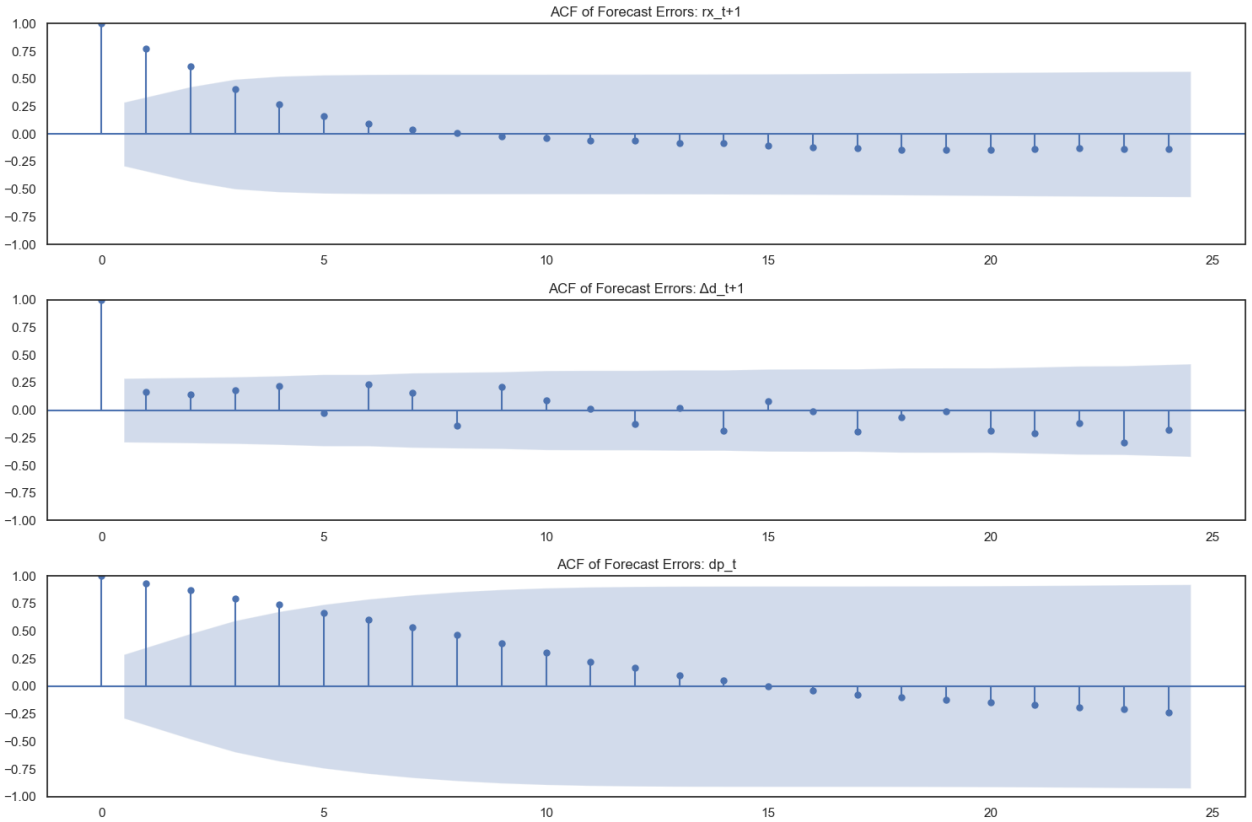


In [45]:
```
# Forecasted Data Plotting (Correlation Heatmap)
filtered_fitted_monthly_df = monthly_fitted_df[["rx_t+1_fitted", "Δd_t+1_

# Correlation matrix
corr_matrix = filtered_fitted_monthly_df.corr()
# Sample size
n = filtered_fitted_monthly_df.shape[0]

# t-statistics derived from correlation values
with np.errstate(divide="ignore", invalid="ignore"):
    t_stat_matrix = corr_matrix * np.sqrt((n - 2) / (1 - corr_matrix**2))
    t_stat_matrix = t_stat_matrix.round(2)

# For each cell, we want to have both the correlation index, as well as t
annot_matrix = corr_matrix.copy().astype(str)

for i in range(len(corr_matrix)):
    for j in range(len(corr_matrix)):
# We only want to keep the lower triangle and diagonal of the full correl
        if i >= j:
            r = corr_matrix.iloc[i, j]
            t = t_stat_matrix.iloc[i, j]
            annot_matrix.iloc[i, j] = f"{r:.2f}\n({t:.2f})"
        else:
```

```
            annot_matrix.iloc[i, j] = ""

# We manually hide the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Heat-map plot
# General Layout (figure's size and style)
plt.figure(figsize=(10, 5))
sns.set(style="white")

sns.heatmap(corr_matrix,
            mask=mask,
            annot=annot_matrix,
            fmt="",
            cmap="seismic",
            vmin=-1, vmax=1,
            square=True,
            linewidths=0.5,
            cbar_kws={"shrink": .8})

plt.title("Variables Correlation Matrix (Fitted Data) - Heatmap\n(r-value
          fontsize=15)
plt.tight_layout()
plt.show()
```

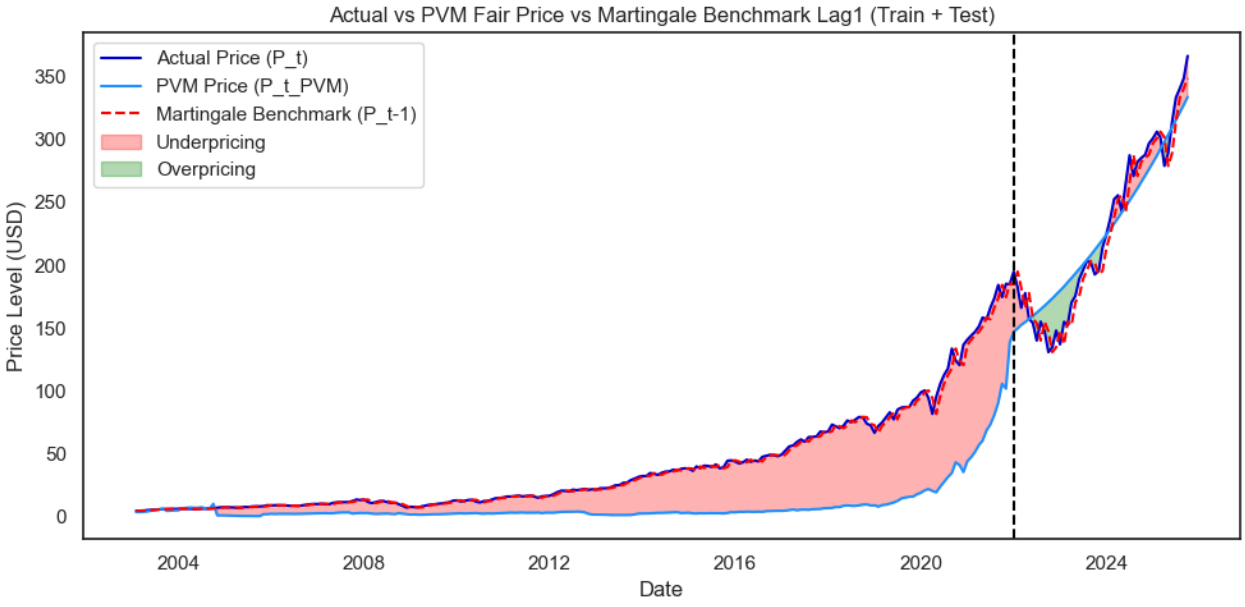


In [ ]:
```
# Autocorrelation of Forecast Errors (ACF) plotting (out-sample testing)
# (!!!) We verify if the model leaves serial correlation in errors (autoc
# (!!!) Weakly autocorrelated white-noise process beyond lag 5
# (!!!) Higher and more persistent autocorrelation until lag 10-12 of dpt

# ACF Computation
plot_df = monthly_test_results

actual_vars = ["rx_t+1", "Δd_t+1", "dp_t"]
forecast_vars = ["rx_t+1_fitted", "Δd_t+1_fitted", "dp_t_fitted"]

errors = {}
for a, f in zip(actual_vars, forecast_vars):
    errors[a] = plot_df[a] - plot_df[f]

# ACF Plotting
fig, axes = plt.subplots(3, 1, figsize=(15, 10))

for i, var in enumerate(actual_vars):
    plot_acf(errors[var].dropna(), lags=24, ax=axes[i])
    axes[i].set_title(f"ACF of Forecast Errors: {var}")

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

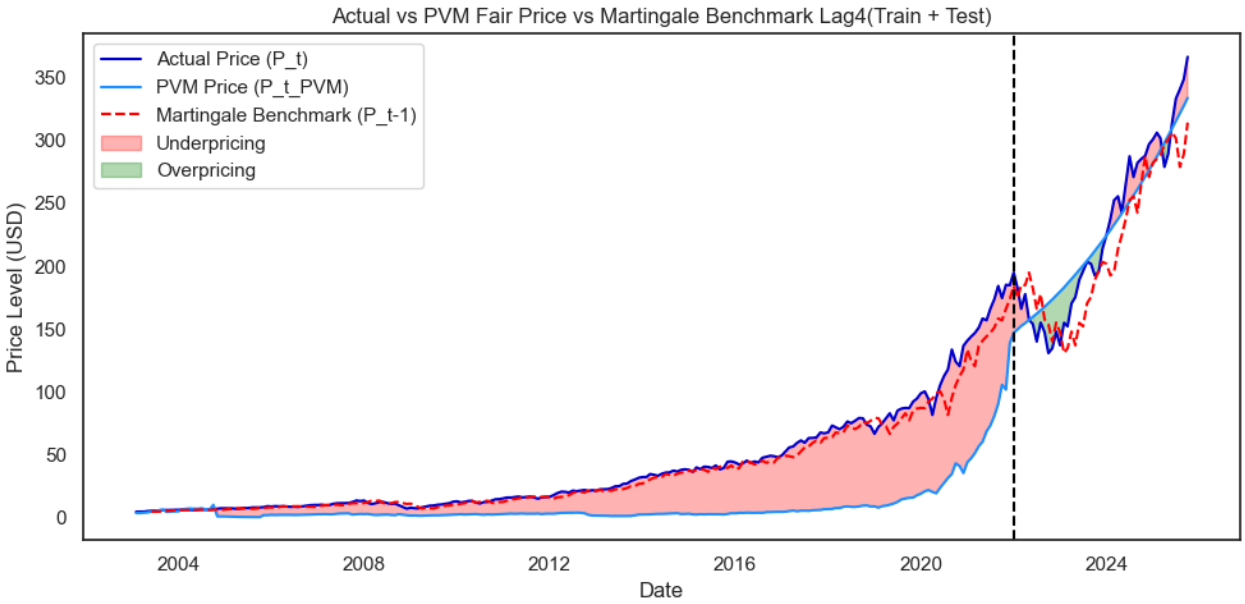


In [84]:
```
# Plotting Forecasted Price against Actual Price Over Time
plot_df = monthly_fitted_df

# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(1)
```

```
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]

plt.figure(figsize=(10, 5))
plt.plot(plot_df["Date"], plot_df["P_t"], color = "mediumblue", label="Ac
plt.plot(plot_df["Date"], plot_df["P_t_PVM"], color = "dodgerblue", label
plt.plot(plot_df["Date"], plot_df["P_t_martingale"], linestyle="--", colo

plt.fill_between(
    plot_df["Date"],
    plot_df["P_t"],
    plot_df["P_t_PVM"],
    where=(plot_df["P_t"] >= plot_df["P_t_PVM"]),
    alpha=0.3,
    color="red",
    label="Underpricing"
)

plt.fill_between(
    plot_df["Date"],
    plot_df["P_t"],
    plot_df["P_t_PVM"],
    where=(plot_df["P_t"] < plot_df["P_t_PVM"]),
    alpha=0.3,
    color="green",
    label="Overpricing"
)

split_date = pd.to_datetime("2022-01-01")
plt.axvline(split_date, linestyle="--", color="black")

plt.title("Actual vs PVM Fair Price vs Martingale Benchmark Lag1 (Train +
plt.xlabel("Date")
plt.ylabel("Price Level (USD)")
plt.legend()
plt.tight_layout()
plt.show()
```

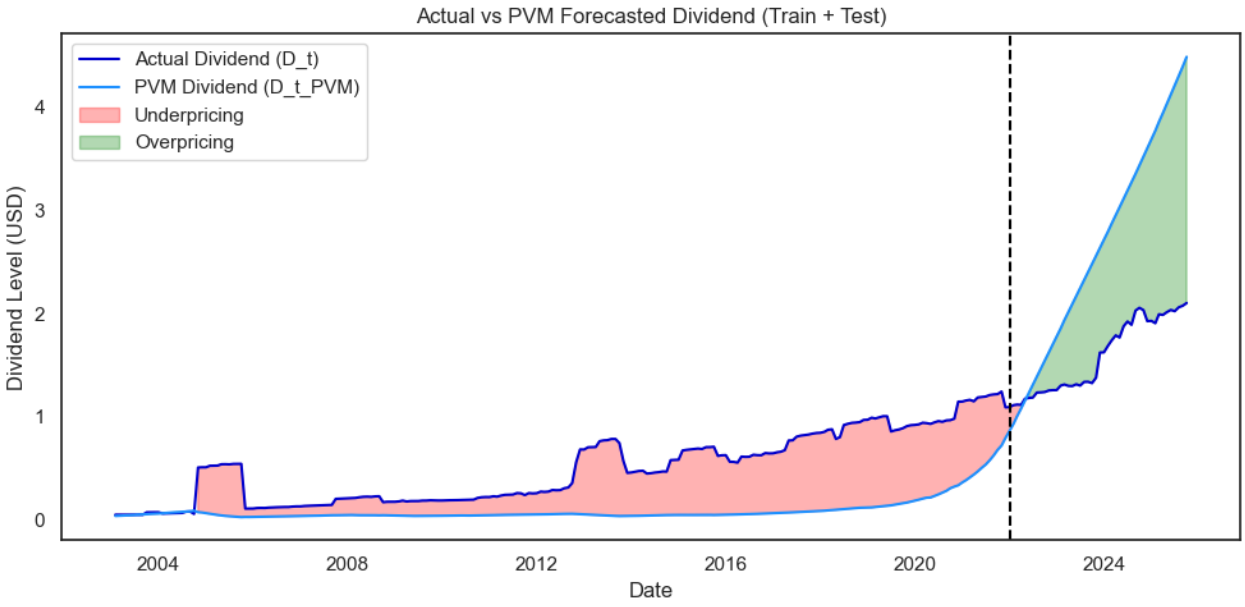


In [ ]:
```python
# Plotting Forecasted Price against Actual Price Over Time
plot_df = monthly_fitted_df

# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(4)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]

plt.figure(figsize=(10, 5))
plt.plot(plot_df["Date"], plot_df["P_t"], color = "mediumblue", label="Ac
plt.plot(plot_df["Date"], plot_df["P_t_PVM"], color = "dodgerblue", label
plt.plot(plot_df["Date"], plot_df["P_t_martingale"], linestyle="--", colo

plt.fill_between(
    plot_df["Date"],
    plot_df["P_t"],
    plot_df["P_t_PVM"],
    where=(plot_df["P_t"] >= plot_df["P_t_PVM"]),
    alpha=0.3,
    color="red",
    label="Underpricing"
)

plt.fill_between(
    plot_df["Date"],
    plot_df["P_t"],
    plot_df["P_t_PVM"],
    where=(plot_df["P_t"] < plot_df["P_t_PVM"]),
    alpha=0.3,
    color="green",
    label="Overpricing"
)

split_date = pd.to_datetime("2022-01-01")
plt.axvline(split_date, linestyle="--", color="black")

plt.title("Actual vs PVM Fair Price vs Martingale Benchmark Lag4 (Train +
plt.xlabel("Date")
plt.ylabel("Price Level (USD)")
plt.legend()
plt.tight_layout()
plt.show()
```



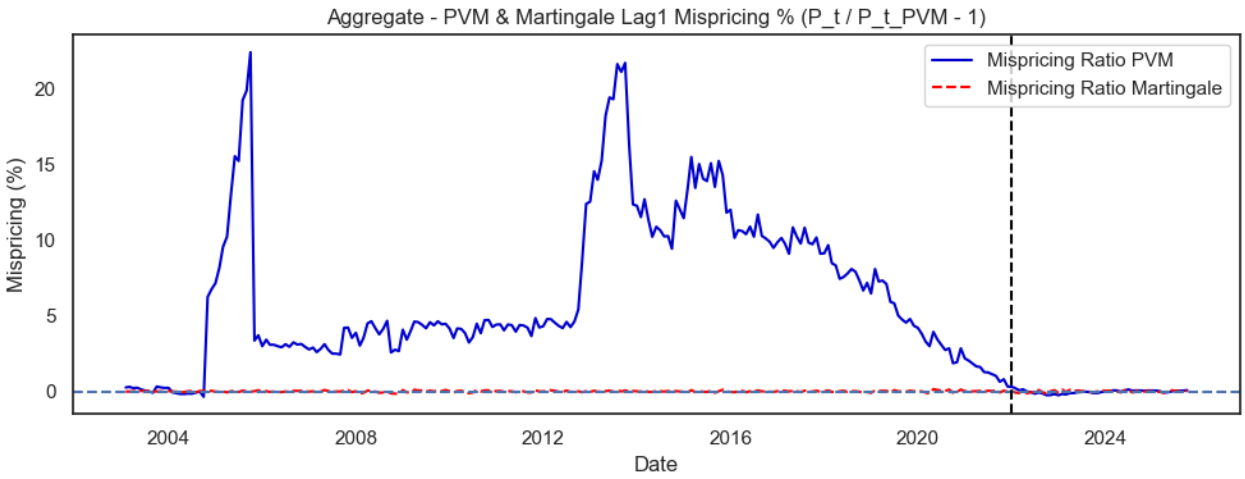
In [57]:
```
# Plotting Forecasted Dividend Payout against Actual Dividend Payout Over
plot_df = monthly_fitted_df

plt.figure(figsize=(10, 5))
plt.plot(plot_df["Date"], plot_df["D_t"], color = "mediumblue", label="Ac
plt.plot(plot_df["Date"], plot_df["D_t_PVM"], color = "dodgerblue", label

plt.fill_between(
    plot_df["Date"],
    plot_df["D_t"],
    plot_df["D_t_PVM"],
    where=(plot_df["D_t"] >= plot_df["D_t_PVM"]),
    alpha=0.3,
    color="red",
    label="Underpricing"
)

plt.fill_between(
    plot_df["Date"],
    plot_df["D_t"],
    plot_df["D_t_PVM"],
    where=(plot_df["D_t"] < plot_df["D_t_PVM"]),
    alpha=0.3,
    color="green",
    label="Overpricing"
)

split_date = pd.to_datetime("2022-01-01")
plt.axvline(split_date, linestyle="--", color="black")

plt.title("Actual vs PVM Forecasted Dividend (Train + Test)")
plt.xlabel("Date")
plt.ylabel("Dividend Level (USD)")
plt.legend()
plt.tight_layout()
```

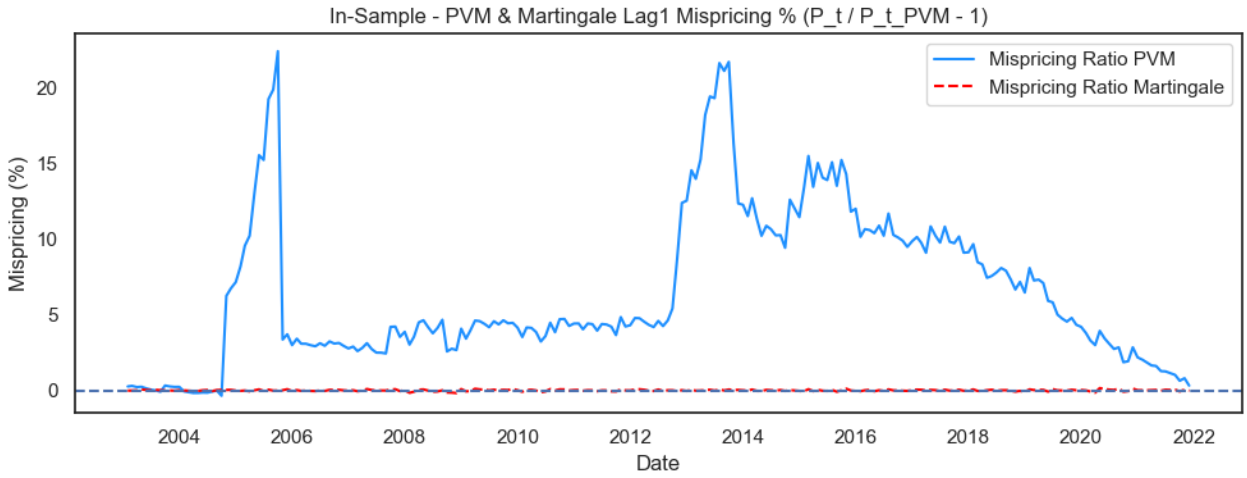
```
plt.show()
```

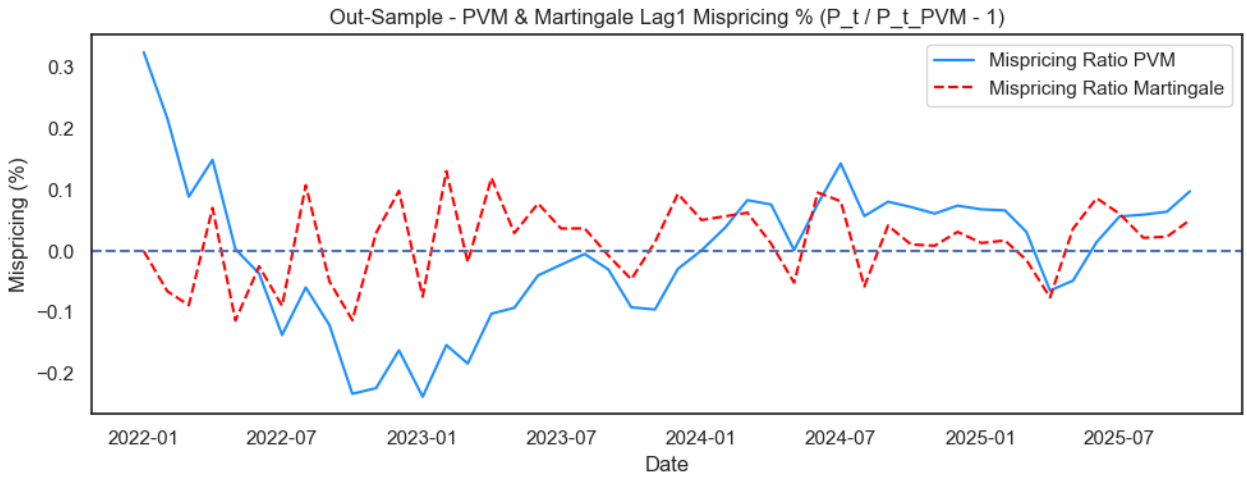


In [82]:
```
# Mispricing plotting (Martingale Lag=1)

# Plotting Aggregate - PVM Mispricing % (P_t / P_t_PVM - 1)
plot_df = monthly_fitted_df

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(1)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="mediumb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.axvline(split_date, linestyle="--", color="black")
plt.title("Aggregate - PVM & Martingale Lag1 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()

# Plotting In-Sample - PVM Mispricing % (P_t / P_t_PVM - 1)
plot_df = monthly_train_results

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(1)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
```

```
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="dodgerb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.title("In-Sample - PVM & Martingale Lag1 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()

# Plotting Out-Sample - PVM Mispricing % (P_t / P_t_PVM - 1)
plot_df = monthly_test_results

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(1)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="dodgerb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.title("Out-Sample - PVM & Martingale Lag1 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()
```

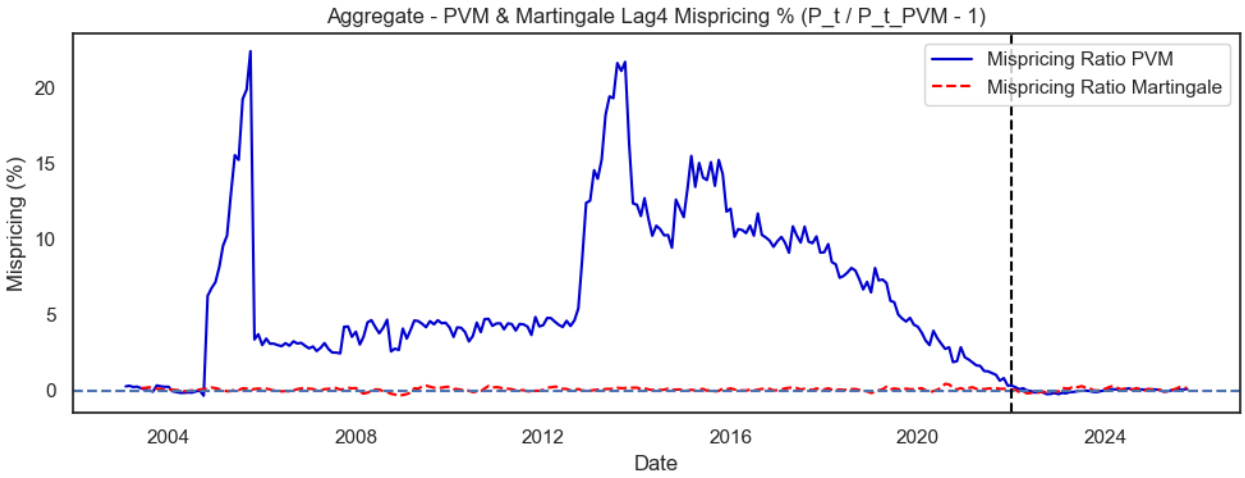


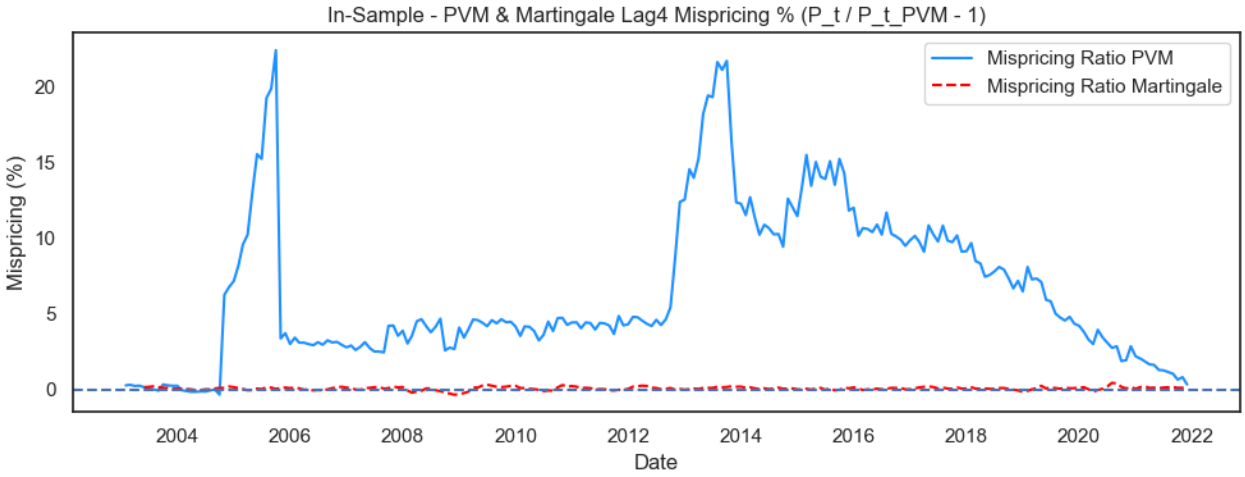


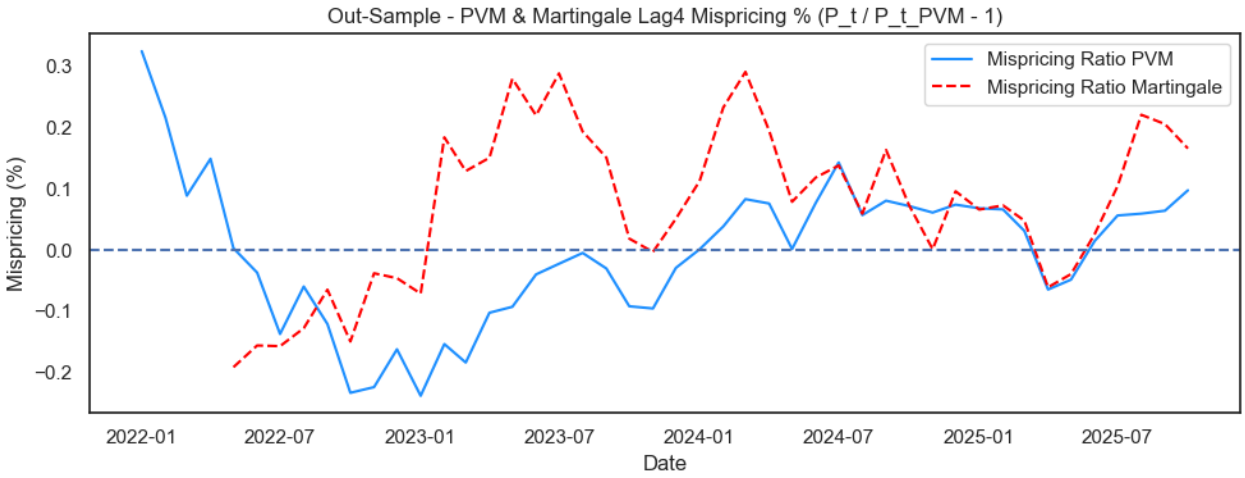


In [83]:
```
# Mispricing plotting (Martingale Lag=4)

# Plotting Aggregate - PVM Mispricing % (P_t / P_t_PVM - 1)
plot_df = monthly_fitted_df

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(4)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="mediumb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.axvline(split_date, linestyle="--", color="black")
plt.title("Aggregate - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()

# Plotting In-Sample - PVM Mispricing % (P_t / P_t_PVM - 1)
```

```
plot_df = monthly_train_results

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(4)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="dodgerb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.title("In-Sample - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()

# Plotting Out-Sample - PVM Mispricing % (P_t / P_t_PVM - 1)
plot_df = monthly_test_results

# Preface: Martingale Mispricing
# Martingale benchmark - P_t^M = P_{t-1}
plot_df["P_t_martingale"] = plot_df["P_t"].shift(4)
plot_df.loc[plot_df.index[0], "P_t_martingale"] = plot_df["P_t"].iloc[0]
plot_df["Mispricing Ratio Martingale"] = plot_df["P_t"] / plot_df["P_t_ma

plt.figure(figsize=(10, 4))
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio PVM"], color="dodgerb
plt.plot(plot_df["Date"], plot_df["Mispricing Ratio Martingale"], color="

plt.axhline(0, linestyle="--")
plt.title("Out-Sample - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.legend()
plt.tight_layout()
plt.show()
```


Aggregate - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM - 1)
Mispricing Ratio PVM
Mispricing Ratio Martingale
Mispricing (%)
Date


In-Sample - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM - 1)
Mispricing Ratio PVM
Mispricing Ratio Martingale
Mispricing (%)
Date


Out-Sample - PVM & Martingale Lag4 Mispricing % (P_t / P_t_PVM - 1)
Mispricing Ratio PVM
Mispricing Ratio Martingale
Mispricing (%)
Date

FOUNDAMENTAL APPROACH - COCHRANE (1992) VAR MODEL EXTENSION - APPENDIX

1) REQUIREMENTS SETUP

In [1]:
```
# !pip install -r requirements.txt
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt
```

3) 12-MONTH LAG DIVIDEND ANALYSIS

3.1) VARIABLES CONSTRUCTION

In [3]:
```
# 12-month lag risk-free aggregate
rf_df = pd.read_csv("../data_extraction/raw_df/risk_free_monthly_df.csv")
rf_df = rf_df.rename(columns={"1-month Yield - US Treasury Securities": "
rf_df["Date"] = pd.to_datetime(rf_df["Date"])
rf_df["Rf_t"] = pd.to_numeric(rf_df["Rf_t"], errors="coerce")

# Fill missing values
rf_df["Rf_t"] = rf_df["Rf_t"].replace([0, 0.0, -np.inf, np.inf], np.nan)
rf_df["Rf_t"] = rf_df["Rf_t"].ffill().bfill()
rf_df["rf_1m_t"] = np.log(1 + rf_df["Rf_t"] / 100 / 12)

rf_df
```

Out[3]:

	Date	Rf_t	rf_1m_t
0	2002-01-01	1.69	0.001407
1	2002-02-01	1.69	0.001407
2	2002-03-01	1.78	0.001482
3	2002-04-01	1.79	0.001491
4	2002-05-01	1.76	0.001466
...	...	...	...
282	2025-07-01	4.32	0.003594
283	2025-08-01	4.49	0.003735
284	2025-09-01	4.41	0.003668
285	2025-10-01	4.17	0.003469
286	2025-11-01	4.06	0.003378

287 rows × 3 columns

In [4]:
```
# Synthetic Price
# (!!!) No changes

p_df = pd.read_csv("../data_extraction/raw_df/synthetic_price_monthly_df.
p_df["Date"] = pd.to_datetime(p_df["Date"].astype(str).str[:10], format="
p_df["p_t"] = np.log(p_df["Synthetic Index Close Price"])
p_df = p_df.rename(columns={"Synthetic Index Close Price": "P_t"})
p_df
```

Out[4]:

	Date	P_t	p_t
0	2002-01-01	6.741954	1.908350
1	2002-02-01	6.708014	1.903303
2	2002-03-01	6.614498	1.889264
3	2002-04-01	6.714867	1.904324
4	2002-05-01	6.022081	1.795433
...	...	...	...
282	2025-07-01	333.452353	5.809500
283	2025-08-01	340.753141	5.831158
284	2025-09-01	348.811723	5.854532
285	2025-10-01	366.551887	5.904140
286	2025-11-01	375.118033	5.927241

287 rows × 3 columns

In [5]:

```
# Synthetic dividend aggregate on 12-month basis

d_df = pd.read_csv("../data_extraction/raw_df/synthetic_div_monthly_df.cs
d_df["Date"] = pd.to_datetime(d_df["Date"].astype(str).str[:10], format="
d_df["d_t"] = np.log(d_df["Synthetic Index Dividend"])
d_df = d_df.rename(columns={"Synthetic Index Dividend": "D_t"})
d_df = d_df.sort_values("Date")
d_df["D12_t"] = d_df["D_t"].rolling(window=12, min_periods=12).sum()
d_df["d12_t"] = np.log(d_df["D12_t"])
d_df
```

Out[5]:

	Date	D_t	d_t	D12_t	d12_t
0	2002-01-01	0.000382	-7.869682	NaN	NaN
1	2002-02-01	0.006262	-5.073240	NaN	NaN
2	2002-03-01	0.002720	-5.907281	NaN	NaN
3	2002-04-01	0.001222	-6.707280	NaN	NaN
4	2002-05-01	0.003142	-5.762878	NaN	NaN
...	...	...	...	...	...
282	2025-07-01	0.055846	-2.885165	2.022016	0.704095
283	2025-08-01	0.253449	-1.372591	2.059450	0.722439
284	2025-09-01	0.216313	-1.531027	2.074903	0.729915
285	2025-10-01	0.076930	-2.564857	2.101338	0.742574
286	2025-11-01	0.061543	-2.788022	1.918976	0.651791

287 rows × 5 columns

In [6]:
```
# Aggregate dataset

monthly_df = pd.merge(p_df, d_df, on="Date", how="outer")
monthly_df = pd.merge(monthly_df, rf_df[["Date", "rf_1m_t"]], on="Date",
monthly_df = monthly_df.sort_values("Date").reset_index(drop=True)
monthly_df["Date"] = pd.to_datetime(monthly_df["Date"])
monthly_df = monthly_df.set_index("Date").sort_index()
monthly_df
```

Out[6]:

Date	P_t	p_t	D_t	d_t	D12_t	d12_t	rf_1m_t
2002-01-01	6.741954	1.908350	0.000382	-7.869682	NaN	NaN	0.001407
2002-02-01	6.708014	1.903303	0.006262	-5.073240	NaN	NaN	0.001407
2002-03-01	6.614498	1.889264	0.002720	-5.907281	NaN	NaN	0.001482
2002-04-01	6.714867	1.904324	0.001222	-6.707280	NaN	NaN	0.001491
2002-05-01	6.022081	1.795433	0.003142	-5.762878	NaN	NaN	0.001466
...	...	...	...	...	...	...	...
2025-07-01	333.452353	5.809500	0.055846	-2.885165	2.022016	0.704095	0.003594
2025-08-01	340.753141	5.831158	0.253449	-1.372591	2.059450	0.722439	0.003735
2025-09-01	348.811723	5.854532	0.216313	-1.531027	2.074903	0.729915	0.003668
2025-10-01	366.551887	5.904140	0.076930	-2.564857	2.101338	0.742574	0.003469
2025-11-01	375.118033	5.927241	0.061543	-2.788022	1.918976	0.651791	0.003378

287 rows × 7 columns

In [7]:
```
# Variables definition

monthly_df["r_1m_t+1"] = (
    monthly_df["p_t"].shift(-1) - monthly_df["p_t"] + np.log(1 + monthly_
)
monthly_df["rx_1m_t+1"] = monthly_df["r_1m_t+1"] - monthly_df["rf_1m_t"]
monthly_df
```

Out[7]:

Date	P_t	p_t	D_t	d_t	D12_t	d12_t	rf_1m_t
2002-01-01	6.741954	1.908350	0.000382	-7.869682	NaN	NaN	0.001407
2002-02-01	6.708014	1.903303	0.006262	-5.073240	NaN	NaN	0.001407
2002-03-01	6.614498	1.889264	0.002720	-5.907281	NaN	NaN	0.001482
2002-04-01	6.714867	1.904324	0.001222	-6.707280	NaN	NaN	0.001491
2002-05-01	6.022081	1.795433	0.003142	-5.762878	NaN	NaN	0.001466
...	...	...	...	...	...	...	...
2025-07-01	333.452353	5.809500	0.055846	-2.885165	2.022016	0.704095	0.003594
2025-08-01	340.753141	5.831158	0.253449	-1.372591	2.059450	0.722439	0.003735
2025-09-01	348.811723	5.854532	0.216313	-1.531027	2.074903	0.729915	0.003668
2025-10-01	366.551887	5.904140	0.076930	-2.564857	2.101338	0.742574	0.003469
2025-11-01	375.118033	5.927241	0.061543	-2.788022	1.918976	0.651791	0.003378

287 rows × 9 columns

In [8]:
```
# Aggregate df

monthly_df["r_12m_t+12"] = monthly_df["r_1m_t+1"].rolling(window=12, min_
monthly_df["rf_12m_t+12"] = monthly_df["rf_1m_t"].rolling(window=12, min_
monthly_df["rx_12m_t+12"] = monthly_df["r_12m_t+12"] - monthly_df["rf_12m
monthly_df["Δd12_t+12"] = monthly_df["d12_t"].shift(-12) - monthly_df["d1
monthly_df["d12p_t"] = monthly_df["d12_t"] - monthly_df["p_t"]
monthly_df = monthly_df.dropna(subset=["D12_t"])
monthly_df.to_csv("FVM_data/raw_monthly_agg12_df.csv")
monthly_df
```

Out[8]:

Date	P_t	p_t	D_t	d_t	D12_t	d12_t	rf_1m_t
2002-12-01	5.496798	1.704166	0.003610	-5.624125	0.033602	-3.393184	0.00104
2003-01-01	4.859930	1.581024	0.000381	-7.871917	0.033601	-3.393210	0.001000
2003-02-01	4.604349	1.527001	0.019004	-3.963097	0.046343	-3.071691	0.000975
2003-03-01	4.652612	1.537429	0.003212	-5.740833	0.046835	-3.061119	0.001008
2003-04-01	4.721963	1.552225	0.000393	-7.842065	0.046006	-3.078981	0.000975
...	...	...	...	...	...	...	...
2025-07-01	333.452353	5.809500	0.055846	-2.885165	2.022016	0.704095	0.003594
2025-08-01	340.753141	5.831158	0.253449	-1.372591	2.059450	0.722439	0.003735
2025-09-01	348.811723	5.854532	0.216313	-1.531027	2.074903	0.729915	0.003668
2025-10-01	366.551887	5.904140	0.076930	-2.564857	2.101338	0.742574	0.003469
2025-11-01	375.118033	5.927241	0.061543	-2.788022	1.918976	0.651791	0.003378

276 rows × 14 columns

3.2) TRAIN/TEST SPLIT

In [9]:
```
# Train/Test split

train_monthly_df = monthly_df.loc[monthly_df.index <= "2021-12-31"].copy(
test_monthly_df = monthly_df.loc[monthly_df.index > "2021-12-31"].copy()
train_monthly_df.to_csv("FVM_data/train_monthly_agg12_df.csv")
test_monthly_df.to_csv("FVM_data/test_monthly_agg12_df.csv")

train_monthly_df
```

Out[9]:

Date	P_t	p_t	D_t	d_t	D12_t	d12_t	rf_1m_t
2002-12-01	5.496798	1.704166	0.003610	-5.624125	0.033602	-3.393184	0.001041
2003-01-01	4.859930	1.581024	0.000381	-7.871917	0.033601	-3.393210	0.001000
2003-02-01	4.604349	1.527001	0.019004	-3.963097	0.046343	-3.071691	0.000975
2003-03-01	4.652612	1.537429	0.003212	-5.740833	0.046835	-3.061119	0.001008
2003-04-01	4.721963	1.552225	0.000393	-7.842065	0.046006	-3.078981	0.000975
...	...	...	...	...	...	...	...
2021-08-01	173.639478	5.156981	0.163301	-1.812162	1.208228	0.189155	0.000042
2021-09-01	184.314724	5.216645	0.061099	-2.795264	1.214015	0.193933	0.000033
2021-10-01	174.603609	5.162518	0.032360	-3.430827	1.217056	0.196435	0.000067
2021-11-01	185.205830	5.221468	0.198816	-1.615377	1.241539	0.216352	0.000042
2021-12-01	184.859378	5.219595	0.071067	-2.644125	1.088753	0.085033	0.000075

229 rows × 14 columns

3.3) VAR MODEL

In [10]:
```
# 12-month aggreagate VAR Model Specification

var_cols = ["rx_12m_t+12", "Δd12_t+12", "d12p_t"]
train_var_df = train_monthly_df[var_cols].dropna().copy()
test_var_df = test_monthly_df[var_cols].dropna().copy()
train_var_df = train_var_df.apply(pd.to_numeric, errors="coerce")
test_var_df = test_var_df.apply(pd.to_numeric, errors="coerce")

m12_model = VAR(train_var_df)
lag_selection = m12_model.select_order(maxlags=10)
print(lag_selection.summary())
```

```
d:\Conda\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWar
ning: No frequency information was provided, so inferred frequency MS will
be used.
  self._init_dates(dates, freq)
```

```
 VAR Order Selection (* highlights the minimums)
================================================
       AIC         BIC         FPE         HQIC
------------------------------------------------
0       -6.315      -6.268    0.001810      -6.296
1       -11.75     -11.56*   7.924e-06      -11.67
2      -11.86*      -11.53  7.080e-06*     -11.73*
3       -11.81      -11.34   7.468e-06      -11.62
4       -11.76      -11.15   7.844e-06      -11.51
5       -11.75      -11.01   7.893e-06      -11.45
6       -11.72      -10.84   8.115e-06      -11.37
7       -11.69      -10.66   8.434e-06      -11.27
8       -11.73      -10.56   8.110e-06      -11.26
9       -11.76      -10.46   7.827e-06      -11.24
10      -11.82      -10.38   7.394e-06      -11.24
------------------------------------------------
```

In [11]:
```
# 12-month aggreagate VAR Model Results

VAR_m12_model = m12_model.fit(1)
print(VAR_m12_model.summary())
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Sat, 22, Nov, 2025
Time:                     23:41:46
--------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                   -11.6067
Nobs:                     228.000    HQIC:                  -11.7143
Log likelihood:           385.183    FPE:                7.60159e-06
AIC:                     -11.7872    Det(Omega_mle):     7.21514e-06
--------------------------------------------------------------------
Results for equation rx_12m_t+12
====================================================================================
=======
                    coefficient       std. error           t-stat
prob
------------------------------------------------------------------------------------
-------
const                  0.022335         0.055487            0.403
0.687
L1.rx_12m_t+12         0.906516         0.026050           34.799
0.000
L1.Δd12_t+12           0.006900         0.010006            0.690
0.490
L1.d12p_t              0.000803         0.013462            0.060
0.952
====================================================================================
=======

Results for equation Δd12_t+12
====================================================================================
```

```
=======
                    coefficient       std. error           t-stat
prob
------------------------------------------------------------------------------
-------
const                -0.126850         0.246929           -0.514
0.607
L1.rx_12m_t+12       -0.056007         0.115930           -0.483
0.629
L1.Δd12_t+12          0.844676         0.044529           18.969
0.000
L1.d12p_t            -0.038080         0.059910           -0.636
0.525
==============================================================================
=======

Results for equation d12p_t
==============================================================================
=======
                    coefficient       std. error           t-stat
prob
------------------------------------------------------------------------------
-------
const                -0.108339         0.147731           -0.733
0.463
L1.rx_12m_t+12       -0.052627         0.069358           -0.759
0.448
L1.Δd12_t+12          0.072173         0.026641            2.709
0.007
L1.d12p_t             0.974982         0.035843           27.202
0.000
==============================================================================
=======

Correlation matrix of residuals
               rx_12m_t+12  Δd12_t+12    d12p_t
rx_12m_t+12       1.000000  -0.014821 -0.040342
Δd12_t+12        -0.014821   1.000000 -0.830719
d12p_t           -0.040342  -0.830719  1.000000
```

3.4) DATA TRAINING

In [12]:
```python
# Data Training
train_fitted = VAR_m12_model.fittedvalues.copy()
train_fitted.columns = [c + "_fitted" for c in train_fitted.columns]
train_with_fitted = monthly_df.join(train_fitted, how="inner")
train_results = train_with_fitted.copy()
train_results["P_t_PVM"] = train_results["D12_t"] / np.exp(train_results[
train_results["mispricing"] = train_results["P_t"] - train_results["P_t_P
train_results["mispricing_pct"] = train_results["P_t"] / train_results["P
```

```
train_results["sample"] = "train"
```

In [13]:
```
# Out-of-sample Training
y_train = train_var_df[var_cols].values
history = y_train.copy()
k_ar = VAR_m12_model.k_ar
n_test = len(test_var_df)
forecasts = []
for i in range(n_test):
    y_pred = VAR_m12_model.forecast(history[-k_ar:], steps=1)
    forecasts.append(y_pred[0])
    history = np.vstack([history, y_pred])

forecast_df = pd.DataFrame(forecasts, index=test_var_df.index, columns=[c
test_with_forecast = monthly_df.join(forecast_df, how="inner")
test_results = test_with_forecast.loc[test_var_df.index].copy()
test_results["P_t_PVM"] = test_results["D12_t"] / np.exp(test_results["d1
test_results["mispricing"] = test_results["P_t"] - test_results["P_t_PVM"
test_results["mispricing_pct"] = test_results["P_t"] / test_results["P_t_
test_results["sample"] = "test"

combined_results = pd.concat([train_results, test_results]).sort_index()
combined_results = combined_results[["P_t", "P_t_PVM", "mispricing", "mis
combined_results.to_csv("FVM_data/results_PVM12M.csv")
combined_results
```

Out[13]:

Date	P_t	P_t_PVM	mispricing	mispricing_pct	sample
2003-01-01	4.859930	5.023030	-0.163100	-0.032470	train
2003-02-01	4.604349	6.129347	-1.524998	-0.248803	train
2003-03-01	4.652612	4.462604	0.190009	0.042578	train
2003-04-01	4.721963	4.382207	0.339755	0.077531	train
2003-05-01	5.108140	4.562708	0.545432	0.119541	train
...	...	...	...	...	...
2024-07-01	287.470793	185.499213	101.971581	0.549714	test
2024-08-01	270.930621	180.256674	90.673947	0.503027	test
2024-09-01	282.265548	191.183723	91.081825	0.476410	test
2024-10-01	285.355975	191.700839	93.655136	0.488548	test
2024-11-01	287.919228	187.669330	100.249899	0.534184	test

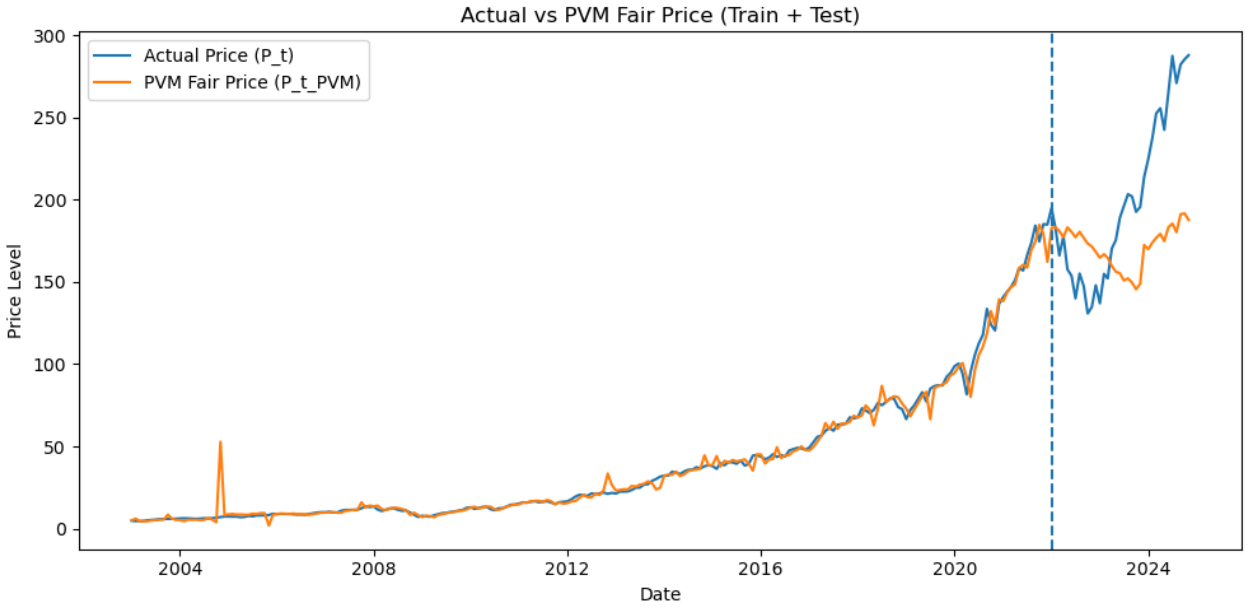
263 rows × 5 columns

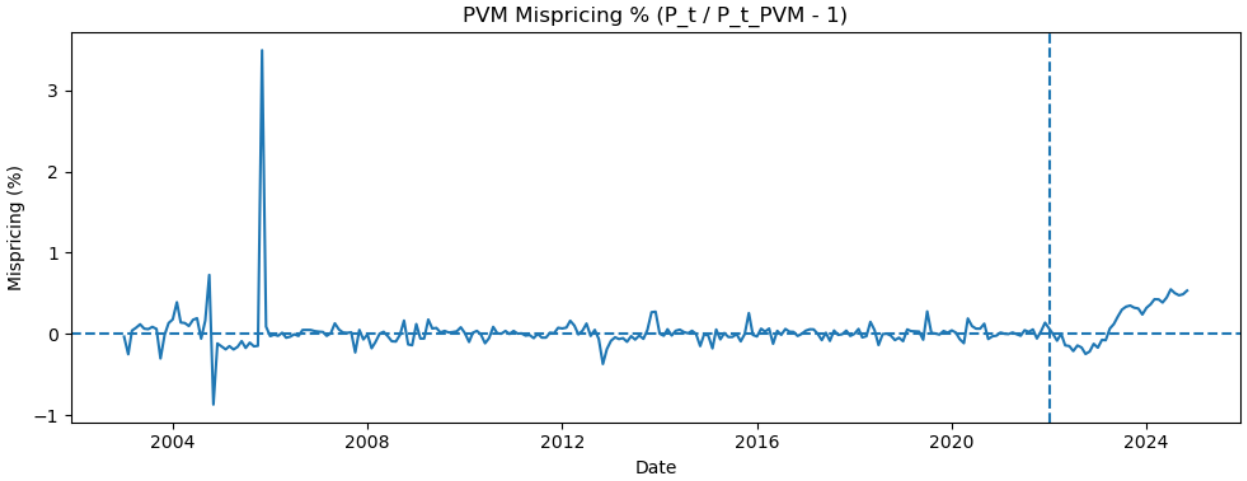
In [14]:
```
# Metrics
```

```
def compute_metrics(df, label):
    df = df.dropna(subset=["P_t", "P_t_PVM"])
    rmse = np.sqrt(np.mean((df["P_t_PVM"] - df["P_t"])**2))
    mape = np.mean(np.abs((df["P_t_PVM"] - df["P_t"]) / df["P_t"])) * 100
    corr = df["P_t"].corr(df["P_t_PVM"])
    ss_res = np.sum((df["P_t"] - df["P_t_PVM"])**2)
    ss_tot = np.sum((df["P_t"] - df["P_t"].mean())**2)
    r2 = 1 - ss_res / ss_tot
    print(f"\n=== {label} PVM Performance ===")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAPE: {mape:.2f}%")
    print(f"Correlation (P_t vs P_t_PVM): {corr:.4f}")
    print(f"Pseudo R²: {r2:.4f}")

compute_metrics(train_results, "TRAIN")
compute_metrics(test_results, "TEST")
```

```
=== TRAIN PVM Performance ===
RMSE: 4.9026
MAPE: 10.17%
Correlation (P_t vs P_t_PVM): 0.9932
Pseudo R²: 0.9862

=== TEST PVM Performance ===
RMSE: 53.4909
MAPE: 21.08%
Correlation (P_t vs P_t_PVM): 0.3617
Pseudo R²: -0.1829
```

In [15]:
```
# Plotting
plot_df = combined_results.dropna(subset=["P_t", "P_t_PVM"]).copy()

plt.figure(figsize=(10, 5))
plt.plot(plot_df.index, plot_df["P_t"], label="Actual Price (P_t)")
plt.plot(plot_df.index, plot_df["P_t_PVM"], label="PVM Fair Price (P_t_PV
split_date = test_results.index.min()
plt.axvline(split_date, linestyle="--")
plt.title("Actual vs PVM Fair Price (Train + Test)")
plt.xlabel("Date")
plt.ylabel("Price Level")
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 4))
plt.plot(plot_df.index, plot_df["mispricing_pct"])
plt.axhline(0, linestyle="--")
plt.axvline(split_date, linestyle="--")
plt.title("PVM Mispricing % (P_t / P_t_PVM - 1)")
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
plt.tight_layout()
plt.show()
```





In [16]:
```
# Boh Roba di simon

Root_dir = ".."
Data_dir = os.path.join(Root_dir, "data_extraction", "raw_df")
# ============================================================
# 1. LOAD NDX DATA AND ALIGN BY DATE
# ============================================================
ndx_df = pd.read_csv(os.path.join(Data_dir, "nasdaq_monthly_df.csv"),pars
ndx_df = ndx_df[["Date","Close"]]
ndx_df = ndx_df.set_index("Date").sort_index()
ndx_df = ndx_df.rename(columns={"Close": "NDX_actual"})

# Join NDX actual prices to combined_results (which is indexed by Date)
ndx_joined = combined_results.join(ndx_df[["NDX_actual"]], how="inner")

# ============================================================
# 2. MAP SYNTHETIC FAIR PRICE TO NDX FAIR PRICE
# ============================================================

# Given regression relationship:
# NDX ≈ 1328.3552 + 68.7614 * Synthetic
```

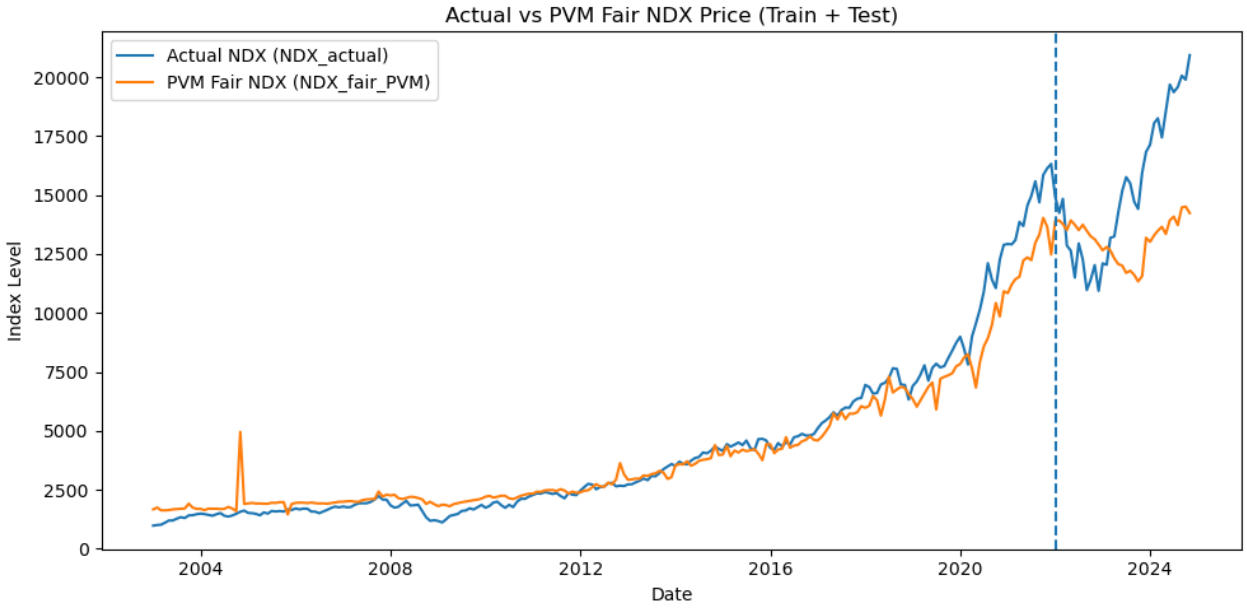
```
# Apply it to the PVM fair synthetic price P_t_PVM

ndx_joined["NDX_fair_PVM"] = (
    1328.3552 + 68.7614 * ndx_joined["P_t_PVM"]
)

# (Optional) also map actual synthetic to an "NDX-equivalent" for sanity
ndx_joined["NDX_equiv_from_synth"] = (
    1328.3552 + 68.7614 * ndx_joined["P_t"]
)

# ============================================================
# 3. NDX-LEVEL MISPRICING BASED ON PVM FAIR VALUE
# ============================================================

ndx_joined["NDX_mispricing"] = (
    ndx_joined["NDX_actual"] - ndx_joined["NDX_fair_PVM"]
)

ndx_joined["NDX_mispricing_pct"] = (
    ndx_joined["NDX_actual"] / ndx_joined["NDX_fair_PVM"] - 1
)

# ============================================================
# 4. FINAL DATAFRAME IN "COMBINED_RESULTS" STYLE
# ============================================================

ndx_pvm_results = ndx_joined[[
    "NDX_actual",          # real Nasdaq price
    "NDX_fair_PVM",        # fair NDX from synthetic PVM
    "NDX_mispricing",      # level gap
    "NDX_mispricing_pct",  # % gap
    "P_t",                 # synthetic actual (for reference)
    "P_t_PVM",             # synthetic fair (for reference)
    "sample"               # train/test flag from combined_results
]].copy()

print(ndx_pvm_results.head())
ndx_pvm_results.to_csv("FVM_data/ndx_pvm_results.csv")
```

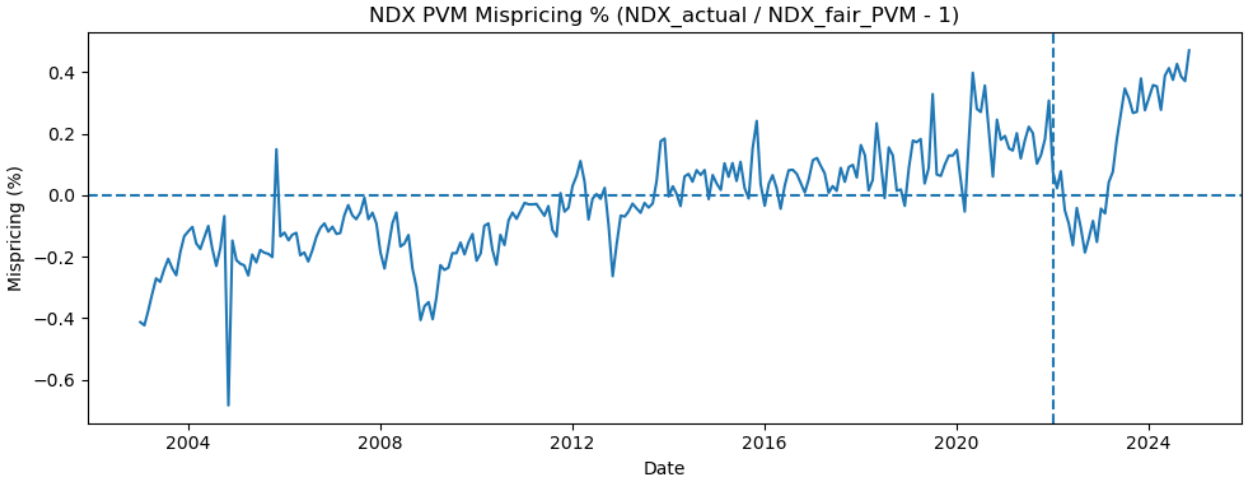
```
            NDX_actual  NDX_fair_PVM  NDX_mispricing  NDX_mispricing_pct
\
Date
2003-01-01   983.049988   1673.745774     -690.695786           -0.412665
2003-02-01  1009.739990   1749.817671     -740.077681           -0.422946
2003-03-01  1018.659973   1635.210098     -616.550125           -0.377046
2003-04-01  1106.060059   1629.681923     -523.621864           -0.321303
2003-05-01  1197.890015   1642.093371     -444.203357           -0.270510

                P_t   P_t_PVM sample
Date
2003-01-01  4.859930  5.023030  train
2003-02-01  4.604349  6.129347  train
2003-03-01  4.652612  4.462604  train
2003-04-01  4.721963  4.382207  train
2003-05-01  5.108140  4.562708  train
```

In [17]:

```python
# Boh roba di simon

# Use NDX-level results instead of synthetic
plot_df = ndx_pvm_results.dropna(subset=["NDX_actual", "NDX_fair_PVM"]).c

# Find the train/test split date from the 'sample' flag
split_date = ndx_pvm_results[ndx_pvm_results["sample"] == "test"].index.m

# ============================
# 1) Actual vs Fair NDX Price
# ============================
plt.figure(figsize=(10, 5))
plt.plot(plot_df.index, plot_df["NDX_actual"], label="Actual NDX (NDX_act
plt.plot(plot_df.index, plot_df["NDX_fair_PVM"], label="PVM Fair NDX (NDX

# Vertical line at start of test period
plt.axvline(split_date, linestyle="--")

plt.title("Actual vs PVM Fair NDX Price (Train + Test)")
plt.xlabel("Date")
plt.ylabel("Index Level")
plt.legend()
plt.tight_layout()
plt.show()

# ============================
# 2) NDX Mispricing %
# ============================
plt.figure(figsize=(10, 4))
plt.plot(plot_df.index, plot_df["NDX_mispricing_pct"])

plt.axhline(0, linestyle="--")
plt.axvline(split_date, linestyle="--")

plt.title("NDX PVM Mispricing % (NDX_actual / NDX_fair_PVM - 1)")
plt.xlabel("Date")
plt.ylabel("Mispricing (%)")
```

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





In [18]:
```
import warnings
warnings.filterwarnings("ignore")
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt
```