# at0lpevnh

November 27, 2025

# 1 EC5303 Team Project: BTC & ETH Dependence and Regression

Full pipeline: load Kaggle data, clean, compute returns, run distribution tests, fit copulas, and estimate regressions as specified in README.

[ ]:
```
# Run once in a fresh environment to install all dependencies
# !pip install -r requirements.txt
```

[120]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from pathlib import Path
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
import statsmodels.api as sm
import kagglehub
import pyvinecopulib as pv

sns.set_theme(style="whitegrid")
pd.set_option("display.float_format", lambda x: f"{x:,.6f}")
```

## 1.1 Section A — Data Loading and Cleaning

- Uses `kagglehub.dataset_load` to retrieve BTC and ETH price history (Kaggle dataset `sudalairajkumar/cryptocurrencypricehistory`).
- Cleans columns, converts dates, sorts ascending, and drops duplicates.
- Computes daily log returns and merges into `df_ret` with `btc_ret` and `eth_ret`.

[121]:
```
DATASET_ID = "sudalairajkumar/cryptocurrencypricehistory"
BTC_FILE = "coin_Bitcoin.csv"
ETH_FILE = "coin_Ethereum.csv"

def load_kaggle_csv(dataset_id: str, filename: str) -> pd.DataFrame:
    """Download dataset locally and read a specific CSV file."""
    dataset_dir = kagglehub.dataset_download(dataset_id)
```

```
    csv_path = Path(dataset_dir) / filename
    return pd.read_csv(csv_path)

df_btc_raw = load_kaggle_csv(DATASET_ID, BTC_FILE)
df_eth_raw = load_kaggle_csv(DATASET_ID, ETH_FILE)

display(df_btc_raw.head())
display(df_eth_raw.head())
```

```
   SNo     Name Symbol                 Date        High         Low        Open  \
0    1  Bitcoin    BTC  2013-04-29 23:59:59  147.488007  134.000000  134.444000
1    2  Bitcoin    BTC  2013-04-30 23:59:59  146.929993  134.050003  144.000000
2    3  Bitcoin    BTC  2013-05-01 23:59:59  139.889999  107.720001  139.000000
3    4  Bitcoin    BTC  2013-05-02 23:59:59  125.599998   92.281898  116.379997
4    5  Bitcoin    BTC  2013-05-03 23:59:59  108.127998   79.099998  106.250000

       Close   Volume              Marketcap
0 144.539993 0.000000 1,603,768,864.500000
1 139.000000 0.000000 1,542,813,125.000000
2 116.989998 0.000000 1,298,954,593.750000
3 105.209999 0.000000 1,168,517,495.250000
4  97.750000 0.000000 1,085,995,168.750000
```

```
   SNo      Name Symbol                 Date      High       Low      Open  \
0    1  Ethereum    ETH  2015-08-08 23:59:59  2.798810  0.714725  2.793760
1    2  Ethereum    ETH  2015-08-09 23:59:59  0.879810  0.629191  0.706136
2    3  Ethereum    ETH  2015-08-10 23:59:59  0.729854  0.636546  0.713989
3    4  Ethereum    ETH  2015-08-11 23:59:59  1.131410  0.663235  0.708087
4    5  Ethereum    ETH  2015-08-12 23:59:59  1.289940  0.883608  1.058750

     Close           Volume          Marketcap
0 0.753325   674,188.000000 45,486,894.240800
1 0.701897   532,170.000000 42,399,573.499100
2 0.708448   405,283.000000 42,818,364.394500
3 1.067860 1,463,100.000000 64,569,288.432800
4 1.217440 2,150,620.000000 73,645,010.986300
```

[122]:
```
KEEP_COLS = ["Date", "Open", "High", "Low", "Close", "Volume", "Marketcap"]

def clean_crypto_df(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    df["Date"] = pd.to_datetime(df["Date"])
    df = df.sort_values("Date").drop_duplicates(subset="Date")
    df = df.drop(columns=[c for c in ["SNo", "Name", "Symbol"] if c in df.
 ↪columns], errors="ignore")
    cols = [c for c in KEEP_COLS if c in df.columns]
    return df[cols]
```

```
df_btc = clean_crypto_df(df_btc_raw)
df_eth = clean_crypto_df(df_eth_raw)

df_btc["btc_ret"] = np.log(df_btc["Close"]).diff()
df_eth["eth_ret"] = np.log(df_eth["Close"]).diff()

df_ret = (
    df_btc[["Date", "btc_ret", "Volume"]]
    .merge(df_eth[["Date", "eth_ret"]], on="Date", how="inner")
    .dropna(subset=["btc_ret", "eth_ret"])
)
df_ret["vol_btc"] = np.log(df_ret["Volume"])

display(df_ret.head())
```

| | Date | btc_ret | Volume | eth_ret | vol_btc |
|---|---|---|---|---|---|
| 1 | 2015-08-09 23:59:59 | 0.015534 | 23,789,600.000000 | -0.070710 | 16.984759 |
| 2 | 2015-08-10 23:59:59 | -0.002315 | 20,979,400.000000 | 0.009290 | 16.859052 |
| 3 | 2015-08-11 23:59:59 | 0.022123 | 25,433,900.000000 | 0.410335 | 17.051593 |
| 4 | 2015-08-12 23:59:59 | -0.014942 | 26,815,400.000000 | 0.131094 | 17.104487 |
| 5 | 2015-08-13 23:59:59 | -0.008657 | 27,685,500.000000 | 0.406292 | 17.136419 |

## 1.2 Section B — Session 1: Distribution & Copula Analysis

All computations below use `btc_ret` and `eth_ret` as required.

[123]:
```
def descriptive_stats(series: pd.Series) -> pd.Series:
    return pd.Series({
        "mean": series.mean(),
        "variance": series.var(ddof=1),
        "std": series.std(ddof=1),
        "skewness": series.skew(),
        "kurtosis": series.kurtosis(),
    })

desc = pd.concat(
    {
        "btc_ret": descriptive_stats(df_ret["btc_ret"]),
        "eth_ret": descriptive_stats(df_ret["eth_ret"]),
    },
    axis=1,
)
cov = df_ret["btc_ret"].cov(df_ret["eth_ret"])
corr = df_ret["btc_ret"].corr(df_ret["eth_ret"])

display(desc)
print(f"Covariance: {cov:.6f}")
print(f"Correlation: {corr:.6f}")
```

```
          btc_ret  eth_ret
mean      0.002259 0.003721
variance  0.001608 0.003856
std       0.040103 0.062099
skewness -0.821522 0.005249
kurtosis 11.912273 7.667930

Covariance: 0.001356
Correlation: 0.544485
```

[124]:

```python
def normality_tests(series: pd.Series, label: str):
    mu, sigma = series.mean(), series.std(ddof=1)
    shapiro_stat, shapiro_p = stats.shapiro(series)
    ks_stat, ks_p = stats.kstest(series, "norm", args=(mu, sigma))
    jb_stat, jb_p = stats.jarque_bera(series)
    return {
        "Series": label,
        "Shapiro_W": shapiro_stat,
        "Shapiro_p": shapiro_p,
        "KS_stat": ks_stat,
        "KS_p": ks_p,
        "JB_stat": jb_stat,
        "JB_p": jb_p,
    }

tests = pd.DataFrame([
    normality_tests(df_ret["btc_ret"], "btc_ret"),
    normality_tests(df_ret["eth_ret"], "eth_ret"),
])
display(tests)
```

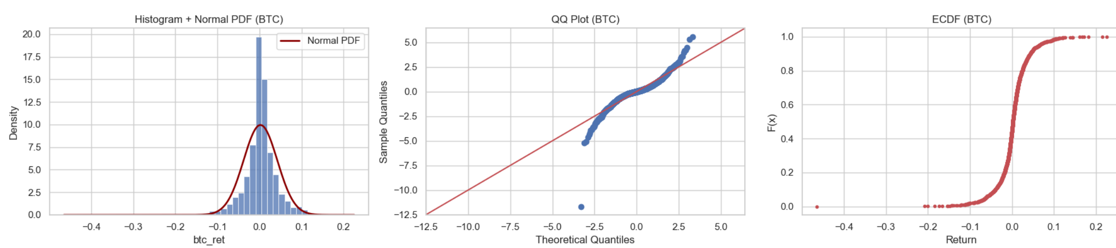
| | Series | Shapiro_W | Shapiro_p | KS_stat | KS_p | JB_stat | JB_p |
|---|---|---|---|---|---|---|---|
| 0 | btc_ret | 0.898178 | 0.000000 | 0.113167 | 0.000000 | 12,942.845177 | 0.000000 |
| 1 | eth_ret | 0.912023 | 0.000000 | 0.092244 | 0.000000 | 5,261.026821 | 0.000000 |

[125]:

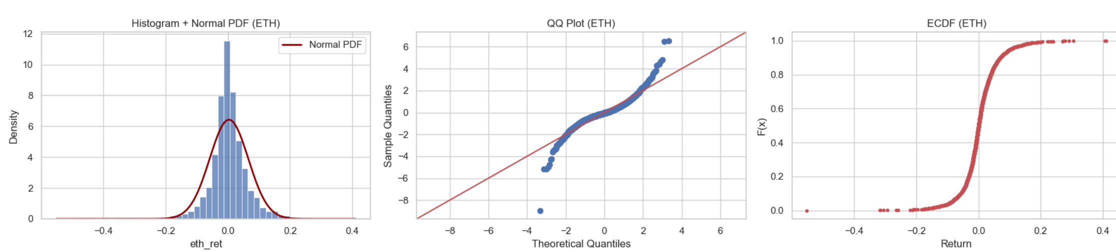
```python
def plot_distribution(series: pd.Series, title: str):
    mu, sigma = series.mean(), series.std(ddof=1)
    x = np.linspace(series.min(), series.max(), 200)
    normal_pdf = stats.norm.pdf(x, mu, sigma)

    fig, axes = plt.subplots(1, 3, figsize=(18, 4))
    sns.histplot(series, bins=50, stat="density", kde=False, ax=axes[0],␣
↪color="#4C72B0")
    axes[0].plot(x, normal_pdf, color="darkred", lw=2, label="Normal PDF")
    axes[0].set_title(f"Histogram + Normal PDF ({title})")
    axes[0].legend()

    sm.ProbPlot(series, fit=True).qqplot(line="45", ax=axes[1], color="#55A868")
```

```
    axes[1].set_title(f"QQ Plot ({title})")

    ecdf = ECDF(series)
    axes[2].plot(ecdf.x, ecdf.y, marker=".", linestyle="none", color="#C44E52")
    axes[2].set_title(f"ECDF ({title})")
    axes[2].set_xlabel("Return")
    axes[2].set_ylabel("F(x)")
    plt.tight_layout()
    plt.show()

plot_distribution(df_ret["btc_ret"], "BTC")
plot_distribution(df_ret["eth_ret"], "ETH")
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\graphics\gofplots.py:1041: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (->
color=(0.2980392156862745, 0.4470588235294118, 0.6901960784313725, 1)). The
keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



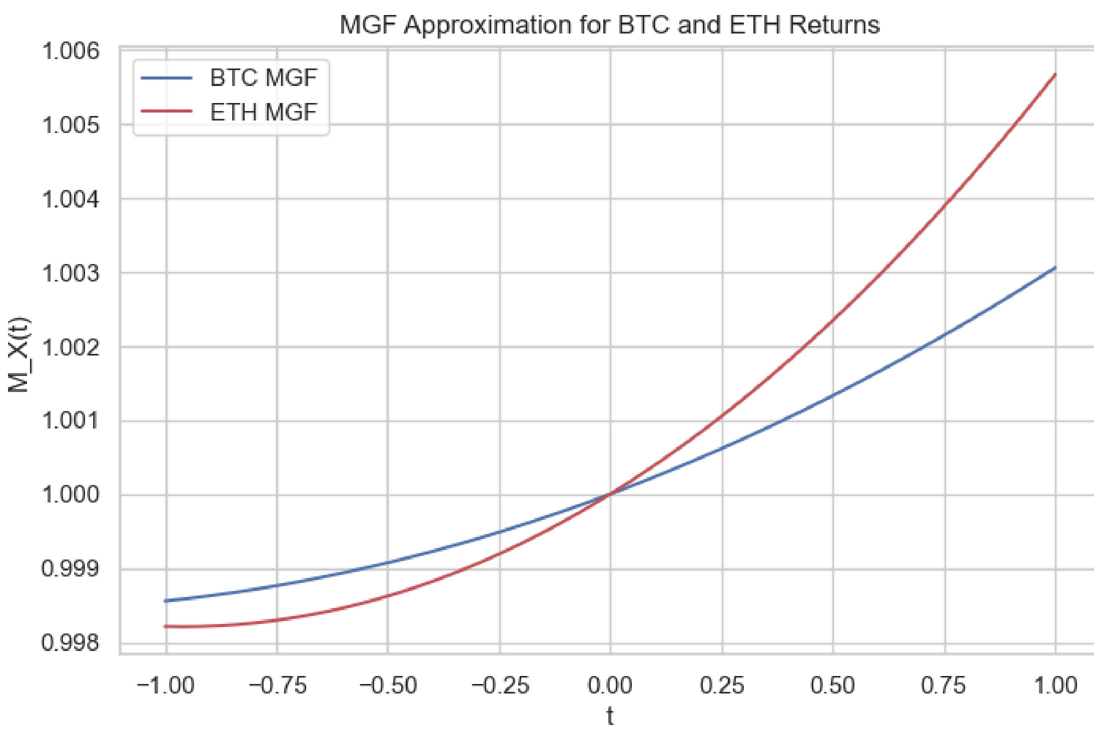

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\graphics\gofplots.py:1041: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (->
color=(0.2980392156862745, 0.4470588235294118, 0.6901960784313725, 1)). The
keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



### 1.2.1 Moment Generating Function (MGF)

Approximate MGF for t [-1, 1]: M_X(t) = E[exp(tX)].

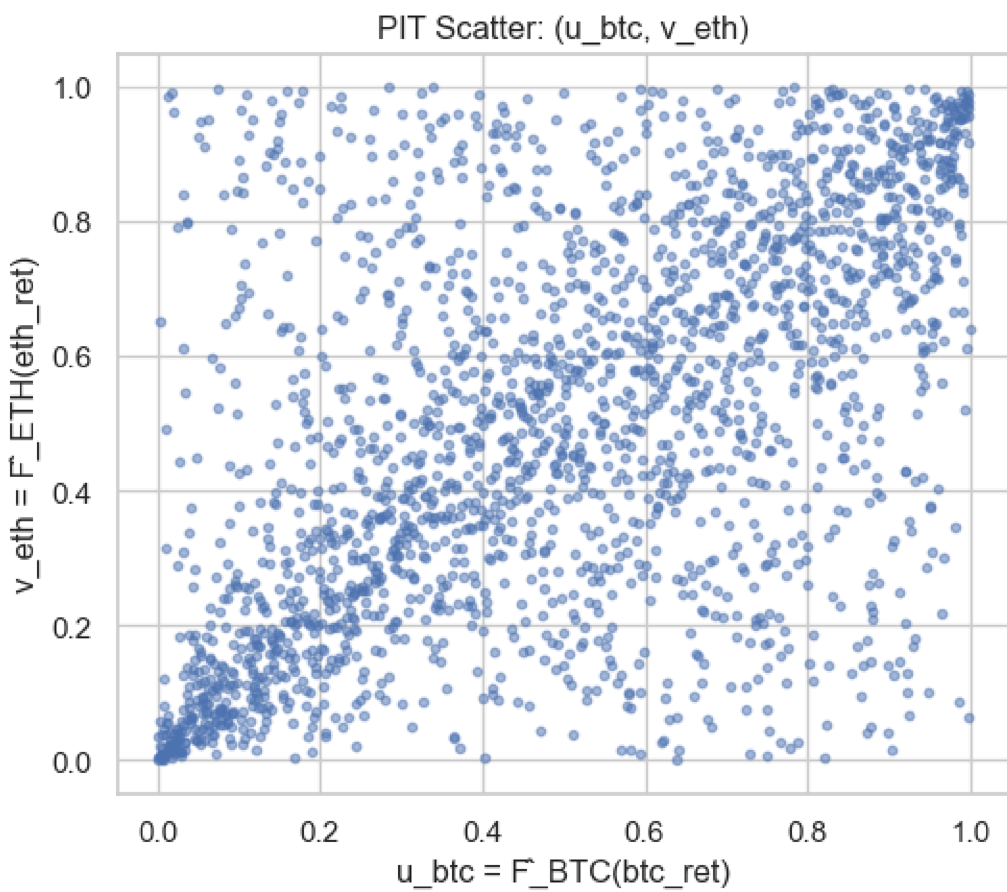
[126]:
```
def mgf_curve(series: pd.Series, t_values: np.ndarray) -> np.ndarray:
    return np.array([np.exp(t * series).mean() for t in t_values])

t_grid = np.linspace(-1, 1, 200)
mgf_btc = mgf_curve(df_ret["btc_ret"], t_grid)
mgf_eth = mgf_curve(df_ret["eth_ret"], t_grid)

plt.figure(figsize=(8, 5))
plt.plot(t_grid, mgf_btc, label="BTC MGF", color="#4C72B0")
plt.plot(t_grid, mgf_eth, label="ETH MGF", color="#C44E52")
plt.xlabel("t")
plt.ylabel("M_X(t)")
plt.title("MGF Approximation for BTC and ETH Returns")
plt.legend()
plt.grid(True)
plt.show()
```




### 1.2.2 Probability Integral Transform (PIT)

Empirical CDFs F_BTC and F_ETH applied to returns to get u_btc and v_eth in (0, 1).

[127]:
```python
ecdf_btc = ECDF(df_ret["btc_ret"])
ecdf_eth = ECDF(df_ret["eth_ret"])

df_ret["u_btc"] = ecdf_btc(df_ret["btc_ret"])
df_ret["v_eth"] = ecdf_eth(df_ret["eth_ret"])
df_ret[["u_btc", "v_eth"]] = df_ret[["u_btc", "v_eth"]].clip(1e-6, 1 - 1e-6)

plt.figure(figsize=(6, 5))
plt.scatter(df_ret["u_btc"], df_ret["v_eth"], alpha=0.5, s=10, color="#4C72B0")
plt.xlabel("u_btc = F_BTC(btc_ret)")
plt.ylabel("v_eth = F_ETH(eth_ret)")
plt.title("PIT Scatter: (u_btc, v_eth)")
plt.grid(True)
plt.show()
```




### 1.2.3 Copula Model Fitting

Fits Gaussian, Student t, Clayton, Gumbel, and Frank copulas on (u_btc, v_eth); computes parameters, log-likelihood, AIC, and BIC.

[128]:

```
uv = np.column_stack([df_ret["u_btc"], df_ret["v_eth"]])
n_obs = uv.shape[0]

def resolve_family(candidates, label):
    # Pick the first available BicopFamily attribute from candidates.
    for cand in candidates:
        if hasattr(pv.BicopFamily, cand):
            return getattr(pv.BicopFamily, cand)
    available = [a for a in dir(pv.BicopFamily) if not a.startswith("_")]
    raise AttributeError(
        f"{label} not found in pyvinecopulib.BicopFamily. "
        f"Tried {candidates}. Available: {available}"
    )

families = {
    "Gaussian Copula": resolve_family(["gaussian"], "Gaussian"),
    "Student t Copula": resolve_family(["t", "student_t", "student"], "Student t"),
    "Clayton Copula": resolve_family(["clayton"], "Clayton"),
    "Gumbel Copula": resolve_family(["gumbel"], "Gumbel"),
    "Frank Copula": resolve_family(["frank"], "Frank"),
}

copula_rows = []
for name, family in families.items():
    model = pv.Bicop(family)
    model.fit(uv)
    #      get_parameters   parameters        []
    params = getattr(model, "get_parameters", lambda: getattr(model, "parameters", []))()
    loglik = model.loglik(uv)
    k = len(params)
    aic = 2 * k - 2 * loglik
    bic = np.log(n_obs) * k - 2 * loglik
    copula_rows.append(
        {"Copula": name, "Parameters": params, "LogLik": loglik, "AIC": aic, "BIC": bic}
    )

copula_results = pd.DataFrame(copula_rows).sort_values("AIC").reset_index(drop=True)
best_by_aic = copula_results.loc[copula_results["AIC"].idxmin(), "Copula"]
best_by_bic = copula_results.loc[copula_results["BIC"].idxmin(), "Copula"]

display(copula_results)
print(f"Best copula by AIC: {best_by_aic}")
print(f"Best copula by BIC: {best_by_bic}")
```

```
             Copula                                         Parameters     LogLik  \
0  Student t Copula  [[0.6050088855164852], [2.5998837070963345]] 537.411298
1    Clayton Copula                            [[1.184432884575223]] 461.923295
2      Frank Copula                            [[4.446769639282415]] 426.707511
3   Gaussian Copula                           [[0.5506744917239277]] 391.948074
4     Gumbel Copula                           [[1.5772114867724165]] 372.887469

            AIC           BIC
0 -1,070.822597 -1,059.467796
1   -921.846591   -916.169190
2   -851.415023   -845.737623
3   -781.896149   -776.218748
4   -743.774938   -738.097538

Best copula by AIC: Student t Copula
Best copula by BIC: Student t Copula
```

### 1.3 Section C — Session 2: Regression Analysis

Dependent variable: `eth_ret`. Independent variables: `btc_ret` and `vol_btc = ln(Volume_BTC)`. Model 1: ETH on BTC. Model 2: ETH on BTC + BTC volume.

[129]:
```python
def regression_diagnostics(model, resid_label: str):
    resid = model.resid
    fitted = model.fittedvalues
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))
    sns.scatterplot(x=fitted, y=resid, ax=axes[0], s=12, color="#4C72B0")
    axes[0].axhline(0, color="red", lw=1)
    axes[0].set_title(f"Residuals vs Fitted ({resid_label})")
    sm.qqplot(resid, line="45", ax=axes[1], color="#55A868")
    axes[1].set_title(f"QQ Plot of Residuals ({resid_label})")
    plt.tight_layout()
    plt.show()

reg_data = df_ret.dropna(subset=["btc_ret", "eth_ret", "vol_btc"]).copy()

X1 = sm.add_constant(reg_data[["btc_ret"]])
model1 = sm.OLS(reg_data["eth_ret"], X1).fit()
display(model1.summary())
regression_diagnostics(model1, "Model 1: ETH ~ BTC")
```

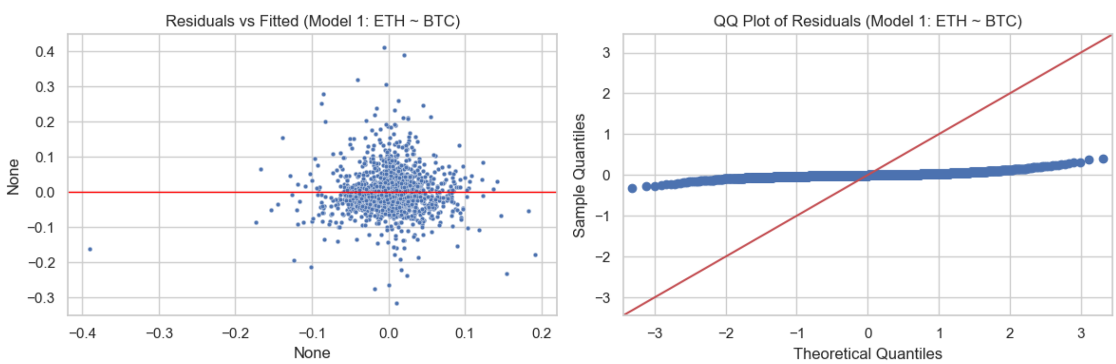
| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **R-squared:** | 0.296 |
| **Model:** | OLS | **Adj. R-squared:** | 0.296 |
| **Method:** | Least Squares | **F-statistic:** | 908.9 |
| **Date:** | Fri, 28 Nov 2025 | **Prob (F-statistic):** | 6.27e-167 |
| **Time:** | 00:46:58 | **Log-Likelihood:** | 3316.5 |
| **No. Observations:** | 2159 | **AIC:** | -6629. |
| **Df Residuals:** | 2157 | **BIC:** | -6618. |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0018 | 0.001 | 1.618 | 0.106 | -0.000 | 0.004 |
| **btc_ret** | 0.8431 | 0.028 | 30.149 | 0.000 | 0.788 | 0.898 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 619.252 | **Durbin-Watson:** | 1.843 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 7703.257 |
| **Skew:** | 0.990 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 12.039 | **Cond. No.** | 24.9 |

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

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\graphics\gofplots.py:1041: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (->
color=(0.2980392156862745, 0.4470588235294118, 0.6901960784313725, 1)). The
keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```




[130]:
```
X2 = sm.add_constant(reg_data[["btc_ret", "vol_btc"]])
model2 = sm.OLS(reg_data["eth_ret"], X2).fit()
display(model2.summary())
regression_diagnostics(model2, "Model 2: ETH ~ BTC + log(Volume_BTC)")

# Coefficient interpretation
beta1 = model2.params.get("btc_ret", np.nan)
```

```python
beta2 = model2.params.get("vol_btc", np.nan)
print(f"beta1 (BTC return): {beta1:.6f} -> ETH moves beta1 per unit BTC return (ceteris paribus).")
print(f"beta2 (log BTC volume): {beta2:.6f} -> ETH return change per unit log volume shock.")
```

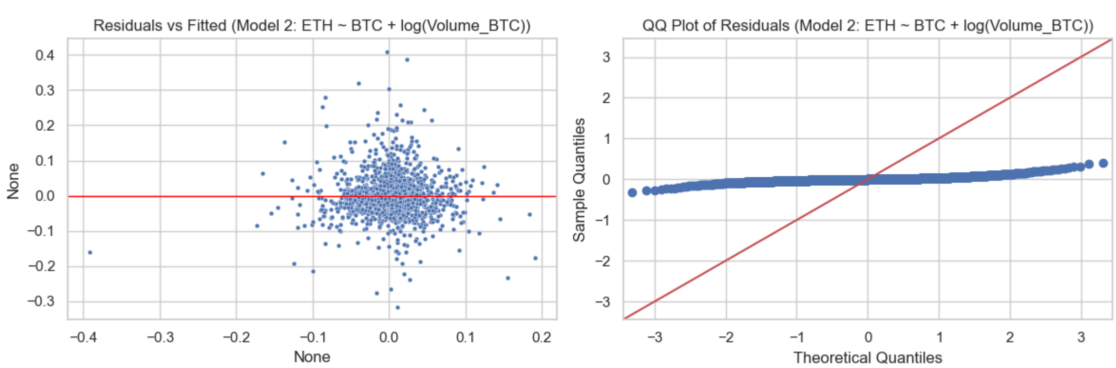
| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **R-squared:** | 0.297 |
| **Model:** | OLS | **Adj. R-squared:** | 0.296 |
| **Method:** | Least Squares | **F-statistic:** | 455.1 |
| **Date:** | Fri, 28 Nov 2025 | **Prob (F-statistic):** | 1.30e-165 |
| **Time:** | 00:46:59 | **Log-Likelihood:** | 3317.1 |
| **No. Observations:** | 2159 | **AIC:** | -6628. |
| **Df Residuals:** | 2156 | **BIC:** | -6611. |
| **Df Model:** | 2 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P> \|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0124 | 0.010 | 1.276 | 0.202 | -0.007 | 0.031 |
| **btc_ret** | 0.8428 | 0.028 | 30.138 | 0.000 | 0.788 | 0.898 |
| **vol_btc** | -0.0005 | 0.000 | -1.096 | 0.273 | -0.001 | 0.000 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 600.178 | **Durbin-Watson:** | 1.844 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 7527.830 |
| **Skew:** | 0.946 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 11.950 | **Cond. No.** | 547. |

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

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\graphics\gofplots.py:1041: UserWarning: color is
redundantly defined by the 'color' keyword argument and the fmt string "b" (->
color=(0.2980392156862745, 0.4470588235294118, 0.6901960784313725, 1)). The
keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



```
beta1 (BTC return): 0.842828 -> ETH moves beta1 per unit BTC return (ceteris
paribus).
beta2 (log BTC volume): -0.000484 -> ETH return change per unit log volume
shock.
```

### 1.3.1 Optional Extra Credit

Uncomment and extend below to try nonlinear terms (e.g., btc_ret^2) or ARCH/GARCH volatility models using `arch`.

[131]:
```python
# Example sketch for nonlinear model (not executed by default)
reg_data["btc_ret_sq"] = reg_data["btc_ret"] ** 2
X3 = sm.add_constant(reg_data[["btc_ret", "btc_ret_sq", "vol_btc"]])
model3 = sm.OLS(reg_data["eth_ret"], X3).fit()
display(model3.summary())
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **R-squared:** | 0.306 |
| **Model:** | OLS | **Adj. R-squared:** | 0.305 |
| **Method:** | Least Squares | **F-statistic:** | 316.4 |
| **Date:** | Fri, 28 Nov 2025 | **Prob (F-statistic):** | 3.29e-170 |
| **Time:** | 00:46:59 | **Log-Likelihood:** | 3330.9 |
| **No. Observations:** | 2159 | **AIC:** | -6654. |
| **Df Residuals:** | 2155 | **BIC:** | -6631. |
| **Df Model:** | 3 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P> \|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0091 | 0.010 | 0.945 | 0.345 | -0.010 | 0.028 |
| **btc_ret** | 0.8143 | 0.028 | 28.757 | 0.000 | 0.759 | 0.870 |
| **btc_ret_sq** | -1.0095 | 0.192 | -5.267 | 0.000 | -1.385 | -0.634 |
| **vol_btc** | -0.0003 | 0.000 | -0.583 | 0.560 | -0.001 | 0.001 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 648.525 | **Durbin-Watson:** | 1.847 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 7837.661 |
| **Skew:** | 1.062 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 12.089 | **Cond. No.** | 3.77e+03 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.77e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

[ ]:

## 1.4 Section D — Time Series Models (AR, MA, ARMA, ARCH/GARCH)

- Uses daily log returns `btc_ret` and `eth_ret`.
- Fits AR, MA, ARMA for short-term dynamics and compares AIC/BIC.
- Fits ARCH(1) and GARCH(1,1) to capture volatility clustering.

[132]:
```python
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model

ts_data = df_ret[["Date", "btc_ret", "eth_ret"]].dropna()

def arima_grid(series, orders):
    rows = []
    for label, order in orders.items():
        res = ARIMA(series, order=order).fit()
        rows.append({
            "Model": label,
            "order": order,
            "AIC": res.aic,
            "BIC": res.bic,
        })
        print(f"{label} summary:")
        display(res.summary())
    return pd.DataFrame(rows).sort_values("AIC").reset_index(drop=True)

orders = {"AR(1)": (1, 0, 0), "MA(1)": (0, 0, 1), "ARMA(1,1)": (1, 0, 1)}

print("BTC return AR/MA/ARMA")
arima_btc = arima_grid(ts_data["btc_ret"], orders)
display(arima_btc)

print("ETH return AR/MA/ARMA")
arima_eth = arima_grid(ts_data["eth_ret"], orders)
display(arima_eth)
```

```
BTC return AR/MA/ARMA
AR(1) summary:
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | btc_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(1, 0, 0) | **Log Likelihood** | 3882.322 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -7758.643 |
| **Time:** | 00:47:00 | **BIC** | -7741.611 |
| **Sample:** | 0<br>- 2159 | **HQIC** | -7752.413 |
| **Covariance Type:** | opg | | |

| | **coef** | **std err** | **z** | **P>\|z\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0023 | 0.001 | 2.634 | 0.008 | 0.001 | 0.004 |
| **ar.L1** | -0.0347 | 0.015 | -2.347 | 0.019 | -0.064 | -0.006 |
| **sigma2** | 0.0016 | 1.9e-05 | 84.373 | 0.000 | 0.002 | 0.002 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.00 | **Jarque-Bera (JB):** | 13159.76 |
| **Prob(Q):** | 0.97 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.45 | **Skew:** | -0.84 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 14.98 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

```
MA(1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | btc_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(0, 0, 1) | **Log Likelihood** | 3882.252 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -7758.505 |
| **Time:** | 00:47:00 | **BIC** | -7741.473 |
| **Sample:** | 0<br>- 2159 | **HQIC** | -7752.275 |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P> \|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 0.0023 | 0.001 | 2.634 | 0.008 | 0.001 | 0.004 |
| **ma.L1** | -0.0338 | 0.015 | -2.290 | 0.022 | -0.063 | -0.005 |
| **sigma2** | 0.0016 | 1.9e-05 | 84.335 | 0.000 | 0.002 | 0.002 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.00 | **Jarque-Bera (JB):** | 13155.31 |
| **Prob(Q):** | 1.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.45 | **Skew:** | -0.84 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 14.98 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
ARMA(1,1) summary:
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | btc_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(1, 0, 1) | **Log Likelihood** | 3883.034 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -7758.067 |
| **Time:** | 00:47:01 | **BIC** | -7735.358 |
| **Sample:** | 0 | **HQIC** | -7749.761 |
| | - 2159 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P> \|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 0.0023 | 0.001 | 2.600 | 0.009 | 0.001 | 0.004 |
| **ar.L1** | -0.4908 | 0.350 | -1.403 | 0.161 | -1.177 | 0.195 |
| **ma.L1** | 0.4570 | 0.355 | 1.287 | 0.198 | -0.239 | 1.153 |
| **sigma2** | 0.0016 | 1.91e-05 | 84.134 | 0.000 | 0.002 | 0.002 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.00 | **Jarque-Bera (JB):** | 13150.59 |
| **Prob(Q):** | 0.99 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.44 | **Skew:** | -0.83 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 14.98 |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
       Model      order           AIC           BIC
0      AR(1)  (1, 0, 0) -7,758.643119 -7,741.610918
1      MA(1)  (0, 0, 1) -7,758.504793 -7,741.472592
2  ARMA(1,1)  (1, 0, 1) -7,758.067201 -7,735.357599
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

```
ETH return AR/MA/ARMA
AR(1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(1, 0, 0) | **Log Likelihood** | 2937.157 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -5868.314 |
| **Time:** | 00:47:01 | **BIC** | -5851.282 |
| **Sample:** | 0 - 2159 | **HQIC** | -5862.084 |
| **Covariance Type:** | opg | | |

| | **coef** | **std err** | **z** | **P>\|z\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0037 | 0.001 | 2.743 | 0.006 | 0.001 | 0.006 |
| **ar.L1** | 0.0141 | 0.014 | 1.004 | 0.315 | -0.013 | 0.042 |
| **sigma2** | 0.0039 | 5.44e-05 | 70.834 | 0.000 | 0.004 | 0.004 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.00 | **Jarque-Bera (JB):** | 5196.18 |
| **Prob(Q):** | 0.97 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 0.50 | **Skew:** | 0.01 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 10.60 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
```

```
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

```
MA(1) summary:
```

```
d:\miniconda\envs\ec5303\Lib\site-packages\statsmodels\base\model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(0, 0, 1) | **Log Likelihood** | 2937.137 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -5868.274 |
| **Time:** | 00:47:01 | **BIC** | -5851.242 |
| **Sample:** | 0 - 2159 | **HQIC** | -5862.044 |
| **Covariance Type:** | opg | | |

| | **coef** | **std err** | **z** | **P> \|z\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0037 | 0.001 | 2.749 | 0.006 | 0.001 | 0.006 |
| **ma.L1** | 0.0119 | 0.014 | 0.842 | 0.400 | -0.016 | 0.040 |
| **sigma2** | 0.0039 | 5.44e-05 | 70.860 | 0.000 | 0.004 | 0.004 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.01 | **Jarque-Bera (JB):** | 5206.82 |
| **Prob(Q):** | 0.94 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 0.50 | **Skew:** | 0.01 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 10.61 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
ARMA(1,1) summary:
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
```

```
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **No. Observations:** | 2159 |
| **Model:** | ARIMA(1, 0, 1) | **Log Likelihood** | 2939.441 |
| **Date:** | Fri, 28 Nov 2025 | **AIC** | -5870.883 |
| **Time:** | 00:47:01 | **BIC** | -5848.173 |
| **Sample:** | 0<br>- 2159 | **HQIC** | -5862.577 |
| **Covariance Type:** | opg | | |

| | **coef** | **std err** | **z** | **P> \|z\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | 0.0037 | 0.002 | 2.409 | 0.016 | 0.001 | 0.007 |
| **ar.L1** | 0.7968 | 0.119 | 6.690 | 0.000 | 0.563 | 1.030 |
| **ma.L1** | -0.7680 | 0.126 | -6.102 | 0.000 | -1.015 | -0.521 |
| **sigma2** | 0.0038 | 5.45e-05 | 70.534 | 0.000 | 0.004 | 0.004 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.63 | **Jarque-Bera (JB):** | 5070.23 |
| **Prob(Q):** | 0.43 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 0.50 | **Skew:** | -0.02 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 10.51 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

| | Model | order | AIC | BIC |
|---|---|---|---|---|
| 0 | ARMA(1,1) | (1, 0, 1) | -5,870.882929 | -5,848.173328 |
| 1 | AR(1) | (1, 0, 0) | -5,868.314132 | -5,851.281931 |
| 2 | MA(1) | (0, 0, 1) | -5,868.273991 | -5,851.241790 |

[133]:
```
def fit_vol_models(series, label):
    scaled = series.dropna() * 100  # scale to percentage
    specs = {
        "ARCH(1)": {"vol": "ARCH", "p": 1, "q": 0},
        "GARCH(1,1)": {"vol": "GARCH", "p": 1, "q": 1},
    }
    rows = []
    for name, spec in specs.items():
        am = arch_model(scaled, mean="Constant", rescale=False, **spec)
        res = am.fit(disp="off")
        rows.append({"Model": name, "AIC": res.aic, "BIC": res.bic})
        print(f"{label} {name} summary:")
        display(res.summary())
    return pd.DataFrame(rows).sort_values("AIC").reset_index(drop=True)
```

```python
print("BTC volatility models")
arch_btc = fit_vol_models(ts_data["btc_ret"], "BTC")
display(arch_btc)

print("ETH volatility models")
arch_eth = fit_vol_models(ts_data["eth_ret"], "ETH")
display(arch_eth)
```

```
BTC volatility models
BTC ARCH(1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | btc_ret | **R-squared:** | 0.000 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | 0.000 |
| **Vol Model:** | ARCH | **Log-Likelihood:** | -6024.01 |
| **Distribution:** | Normal | **AIC:** | 12054.0 |
| **Method:** | Maximum Likelihood | **BIC:** | 12071.0 |
| | | **No. Observations:** | 2159 |
| **Date:** | Fri, Nov 28 2025 | **Df Residuals:** | 2158 |
| **Time:** | 00:47:01 | **Df Model:** | 1 |

| | **coef** | **std err** | **t** | **P> \|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **mu** | 0.2418 | 8.361e-02 | 2.892 | 3.825e-03 | [7.795e-02, 0.406] |

| | **coef** | **std err** | **t** | **P> \|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **omega** | 13.8373 | 1.636 | 8.460 | 2.667e-17 | [ 10.632, 17.043] |
| **alpha[1]** | 0.1421 | 4.819e-02 | 2.948 | 3.199e-03 | [4.761e-02, 0.237] |

Covariance estimator: robust

```
BTC GARCH(1,1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | btc_ret | **R-squared:** | 0.000 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | 0.000 |
| **Vol Model:** | GARCH | **Log-Likelihood:** | -5866.70 |
| **Distribution:** | Normal | **AIC:** | 11741.4 |
| **Method:** | Maximum Likelihood | **BIC:** | 11764.1 |
| | | **No. Observations:** | 2159 |
| **Date:** | Fri, Nov 28 2025 | **Df Residuals:** | 2158 |
| **Time:** | 00:47:01 | **Df Model:** | 1 |

| | **coef** | **std err** | **t** | **P> \|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **mu** | 0.2537 | 7.724e-02 | 3.284 | 1.023e-03 | [ 0.102, 0.405] |

| | **coef** | **std err** | **t** | **P> \|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **omega** | 0.6151 | 0.265 | 2.318 | 2.043e-02 | [9.509e-02, 1.135] |
| **alpha[1]** | 0.1369 | 4.888e-02 | 2.800 | 5.106e-03 | [4.107e-02, 0.233] |
| **beta[1]** | 0.8419 | 3.756e-02 | 22.413 | 2.937e-111 | [ 0.768, 0.916] |

Covariance estimator: robust

| Model | AIC | BIC |
|---|---|---|

```
0  GARCH(1,1) 11,741.390430 11,764.100032
1     ARCH(1) 12,054.015625 12,071.047826
```

```
ETH volatility models
ETH ARCH(1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **R-squared:** | 0.000 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | 0.000 |
| **Vol Model:** | ARCH | **Log-Likelihood:** | -6931.96 |
| **Distribution:** | Normal | **AIC:** | 13869.9 |
| **Method:** | Maximum Likelihood | **BIC:** | 13886.9 |
| | | **No. Observations:** | 2159 |
| **Date:** | Fri, Nov 28 2025 | **Df Residuals:** | 2158 |
| **Time:** | 00:47:01 | **Df Model:** | 1 |

| | **coef** | **std err** | **t** | **P>\|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **mu** | 0.2432 | 0.121 | 2.007 | 4.471e-02 | [5.741e-03, 0.481] |

| | **coef** | **std err** | **t** | **P>\|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **omega** | 29.1013 | 2.833 | 10.271 | 9.504e-25 | [ 23.548, 34.654] |
| **alpha[1]** | 0.2765 | 6.898e-02 | 4.008 | 6.118e-05 | [ 0.141, 0.412] |

Covariance estimator: robust

```
ETH GARCH(1,1) summary:
```

| | | | |
|---|---|---|---|
| **Dep. Variable:** | eth_ret | **R-squared:** | 0.000 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | 0.000 |
| **Vol Model:** | GARCH | **Log-Likelihood:** | -6771.52 |
| **Distribution:** | Normal | **AIC:** | 13551.0 |
| **Method:** | Maximum Likelihood | **BIC:** | 13573.8 |
| | | **No. Observations:** | 2159 |
| **Date:** | Fri, Nov 28 2025 | **Df Residuals:** | 2158 |
| **Time:** | 00:47:01 | **Df Model:** | 1 |

| | **coef** | **std err** | **t** | **P>\|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **mu** | 0.2192 | 0.105 | 2.089 | 3.667e-02 | [1.358e-02, 0.425] |

| | **coef** | **std err** | **t** | **P>\|t\|** | **95.0% Conf. Int.** |
|---|---|---|---|---|---|
| **omega** | 2.6548 | 0.824 | 3.222 | 1.272e-03 | [ 1.040, 4.270] |
| **alpha[1]** | 0.1696 | 3.723e-02 | 4.556 | 5.214e-06 | [9.664e-02, 0.243] |
| **beta[1]** | 0.7699 | 4.251e-02 | 18.112 | 2.579e-73 | [ 0.687, 0.853] |

Covariance estimator: robust

```
        Model           AIC           BIC
0  GARCH(1,1) 13,551.040665 13,573.750267
1     ARCH(1) 13,869.915575 13,886.947776
```

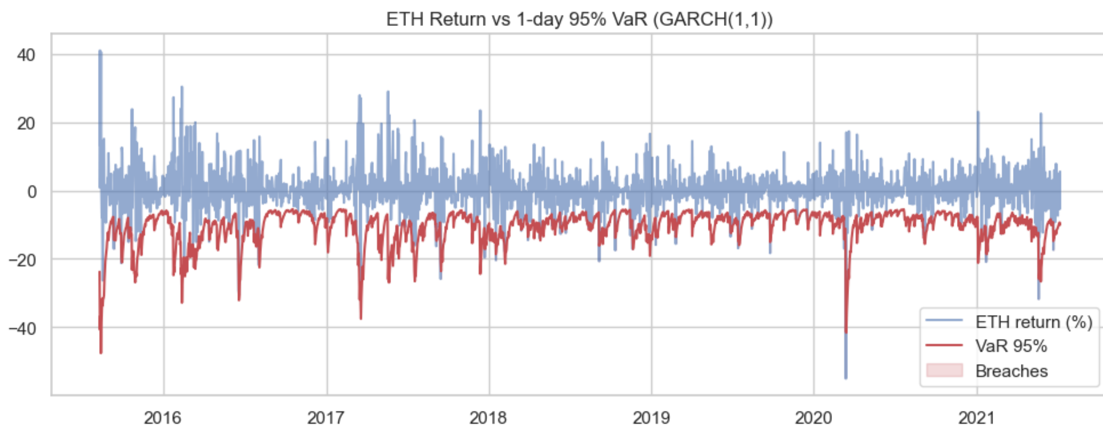
### 1.4.1 GARCH(1,1) Volatility Forecast + 1-day VaR Backtest

- Fit GARCH(1,1) on ETH returns (mean AR(1), t innovations) to model conditional volatility.

- Generate 1-step-ahead conditional mean/vol and compute 95% one-day VaR: VaR = mu + z0.05 * sigma.
- Backtest breaches (actual return below VaR) and plot VaR vs returns.

[134]:
```python
from arch import arch_model
from scipy.stats import norm
import numpy as np

eth_series = df_ret.set_index("Date")["eth_ret"].dropna() * 100

# Fit GARCH(1,1) with AR(1) mean and t innovations
garch = arch_model(eth_series, mean="AR", lags=1, vol="GARCH", p=1, q=1,␣
 ↪dist="t")
garch_res = garch.fit(disp="off")
print(garch_res.summary())

# In-sample 1-step forecasts for all points (start=0)
fcast = garch_res.forecast(horizon=1, start=0, reindex=True)
mu_fc = fcast.mean["h.1"]
sigma_fc = np.sqrt(fcast.variance["h.1"])

z_005 = norm.ppf(0.05)  # left tail 5%
var_95 = mu_fc + z_005 * sigma_fc

backtest = (
    pd.DataFrame({"eth_ret_pct": eth_series})
    .join(var_95.rename("VaR_95"), how="inner")
    .dropna()
)
print(f"Forecast entries: mu={mu_fc.notna().sum()}, var={sigma_fc.notna().
 ↪sum()}, backtest rows={len(backtest)}")

backtest["breach"] = backtest["eth_ret_pct"] < backtest["VaR_95"]
hit_rate = backtest["breach"].mean()
print(f"VaR 95% hit rate: {hit_rate:.4f} (expected ~5%)")

plt.figure(figsize=(10, 4))
plt.plot(backtest.index, backtest["eth_ret_pct"], label="ETH return (%)",␣
 ↪color="#4C72B0", alpha=0.6)
plt.plot(backtest.index, backtest["VaR_95"], label="VaR 95%", color="#C44E52",␣
 ↪lw=1.5)
plt.fill_between(backtest.index, backtest["VaR_95"], backtest["eth_ret_pct"],␣
 ↪where=backtest["breach"], color="#C44E52", alpha=0.2, label="Breaches")
plt.title("ETH Return vs 1-day 95% VaR (GARCH(1,1))")
plt.legend()
plt.tight_layout()
plt.show()
```

```
                          AR - GARCH Model Results
================================================================================
====
Dep. Variable:                    eth_ret   R-squared:
-0.007
Mean Model:                            AR   Adj. R-squared:
-0.008
Vol Model:                          GARCH   Log-Likelihood:
-6564.07
Distribution:      Standardized Student's t   AIC:
13140.1
Method:                  Maximum Likelihood   BIC:
13174.2
                                            No. Observations:
2158
Date:                      Fri, Nov 28 2025   Df Residuals:
2156
Time:                              00:47:01   Df Model:
2
                                Mean Model
===========================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
---------------------------------------------------------------------------
Const          0.1199  7.925e-02      1.513      0.130 [-3.540e-02,  0.275]
eth_ret[1]    -0.0600  2.154e-02     -2.786  5.329e-03 [ -0.102,-1.780e-02]
                           Volatility Model
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
omega          2.3160      0.686      3.378  7.292e-04 [  0.972,  3.660]
alpha[1]       0.2267  3.661e-02      6.193  5.919e-10 [  0.155,  0.298]
beta[1]        0.7733  3.393e-02     22.790 5.701e-115 [  0.707,  0.840]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             3.1151      0.208     14.974  1.085e-50 [  2.707,  3.523]
========================================================================

Covariance estimator: robust
Forecast entries: mu=2159, var=2158, backtest rows=2158
VaR 95% hit rate: 0.0171 (expected ~5%)
```




### 1.4.2 VAR for BTC & ETH returns

- Fit VAR(p) on [btc_ret, eth_ret] with lag order chosen by AIC/BIC.
- Produce 1-step forecasts and compare forecast errors (RMSE/MAE) against univariate AR on ETH.
- (Optional) Granger causality: test if BTC returns Granger-cause ETH returns.

[135]:
```
from statsmodels.tsa.api import VAR
from statsmodels.tsa.ar_model import AutoReg

var_data = df_ret[["btc_ret", "eth_ret"]].dropna()

# Select lag order by BIC (p>=1)
sel = VAR(var_data).select_order(maxlags=5)
p = max(int(sel.bic), 1)
print("Selected lag p =", p)

var_res = VAR(var_data).fit(p)
print(var_res.summary())

# Rolling 1-step forecasts for both BTC and ETH
fc_list = []
for i in range(p, len(var_data)):
    past = var_data.values[i - p : i]
    fc = var_res.forecast(y=past, steps=1)[0]
    fc_list.append(fc)

pred_index = var_data.index[p:]
var_pred_df = pd.DataFrame(fc_list, index=pred_index, columns=["btc_fc",␣
 ↪"eth_fc"])

# Univariate AR baselines for each asset
```

```python
ar_btc = AutoReg(var_data["btc_ret"], lags=p, old_names=False).fit()
ar_eth = AutoReg(var_data["eth_ret"], lags=p, old_names=False).fit()
ar_btc_pred = ar_btc.predict(start=p, end=len(var_data)-1)
ar_eth_pred = ar_eth.predict(start=p, end=len(var_data)-1)

# Align
comp = pd.concat([
    var_data.iloc[p:][["btc_ret", "eth_ret"]],
    var_pred_df,
    ar_btc_pred.rename("btc_ar_fc"),
    ar_eth_pred.rename("eth_ar_fc"),
], axis=1)

# Error helper
def err(y, yhat):
    return {
        "RMSE": np.sqrt(((y - yhat) ** 2).mean()),
        "MAE": (y - yhat).abs().mean(),
    }

records = []
records.append({"Asset": "BTC", "Model": "VAR", **err(comp["btc_ret"],
 ↪comp["btc_fc"])})
records.append({"Asset": "BTC", "Model": "AR", **err(comp["btc_ret"],
 ↪comp["btc_ar_fc"])})
records.append({"Asset": "ETH", "Model": "VAR", **err(comp["eth_ret"],
 ↪comp["eth_fc"])})
records.append({"Asset": "ETH", "Model": "AR", **err(comp["eth_ret"],
 ↪comp["eth_ar_fc"])})

err_df = pd.DataFrame(records)
display(err_df)
```

```
Selected lag p = 1
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 28, Nov, 2025
Time:                     00:47:02
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                   -12.3324
Nobs:                     2158.00    HQIC:                  -12.3424
Log likelihood:           7205.54    FPE:                4.33768e-06
AIC:                     -12.3482    Det(Omega_mle):     4.32565e-06
--------------------------------------------------------------------
Results for equation btc_ret
```

```
==============================================================================
                 coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const               0.002402         0.000864            2.779           0.005
L1.btc_ret         -0.001521         0.025633           -0.059           0.953
L1.eth_ret         -0.039380         0.016556           -2.379           0.017
==============================================================================

Results for equation eth_ret
==============================================================================
                 coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const               0.003786         0.001339            2.828           0.005
L1.btc_ret         -0.086130         0.039708           -2.169           0.030
L1.eth_ret          0.044381         0.025647            1.730           0.084
==============================================================================

Correlation matrix of residuals
            btc_ret   eth_ret
btc_ret    1.000000  0.547372
eth_ret    0.547372  1.000000
```

```
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
d:\miniconda\envs\ec5303\Lib\site-
packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: An unsupported
index was provided. As a result, forecasts cannot be generated. To use the model
for forecasting, use one of the supported classes of index.
  self._init_dates(dates, freq)
```

```
  Asset Model      RMSE       MAE
0   BTC   VAR 0.040025 0.025748
1   BTC    AR 0.041503 0.026722
```

```
2   ETH   VAR 0.062004 0.041208
3   ETH    AR 0.061207 0.040574
```

[136]:
```python
# Optional: Granger causality BTC -> ETH
from statsmodels.tsa.stattools import grangercausalitytests

print("Granger causality test: does BTC_ret Granger-cause ETH_ret?")
gc_res = grangercausalitytests(var_data[["eth_ret", "btc_ret"]], maxlag=p,␣
 ↪verbose=False)
# summarize p-values
pvals = {lag: res[0]["ssr_chi2test"][1] for lag, res in gc_res.items()}
display(pd.Series(pvals, name="p-value"))
```

```
Granger causality test: does BTC_ret Granger-cause ETH_ret?
```

```
d:\miniconda\envs\ec5303\Lib\site-packages\statsmodels\tsa\stattools.py:1556:
FutureWarning: verbose is deprecated since functions should not print results
  warnings.warn(
```

```
1   0.029963
Name: p-value, dtype: float64
```

### 1.4.3 t-Copula Joint Predictive Simulation (BTC & ETH)

- Use fitted t-copula parameters + GARCH(1,1) margins to simulate next-day joint returns.
- Compare marginal forecasts vs VAR and compute joint risk metrics (e.g., joint VaR, co-move probability).

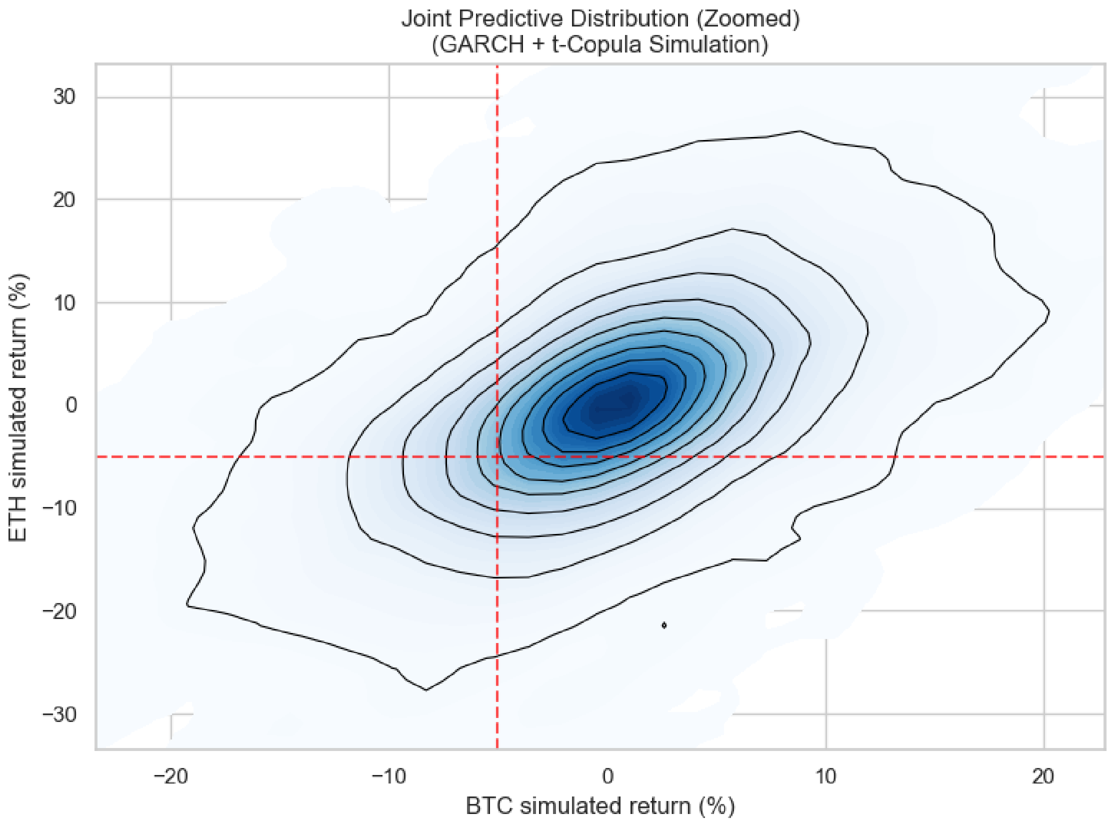
[137]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import t
from arch import arch_model

# ===============================
# 1. Prepare data (BTC/ETH returns)
# ===============================
btc = df_ret.set_index("Date")["btc_ret"].dropna() * 100
eth = df_ret.set_index("Date")["eth_ret"].dropna() * 100

# ===============================
# 2. Fit GARCH(1,1) with t innovations
# ===============================
def fit_garch(series):
    model = arch_model(series, vol="GARCH", p=1, q=1,
                       mean="AR", lags=1, dist="t")
    res = model.fit(disp="off")
```

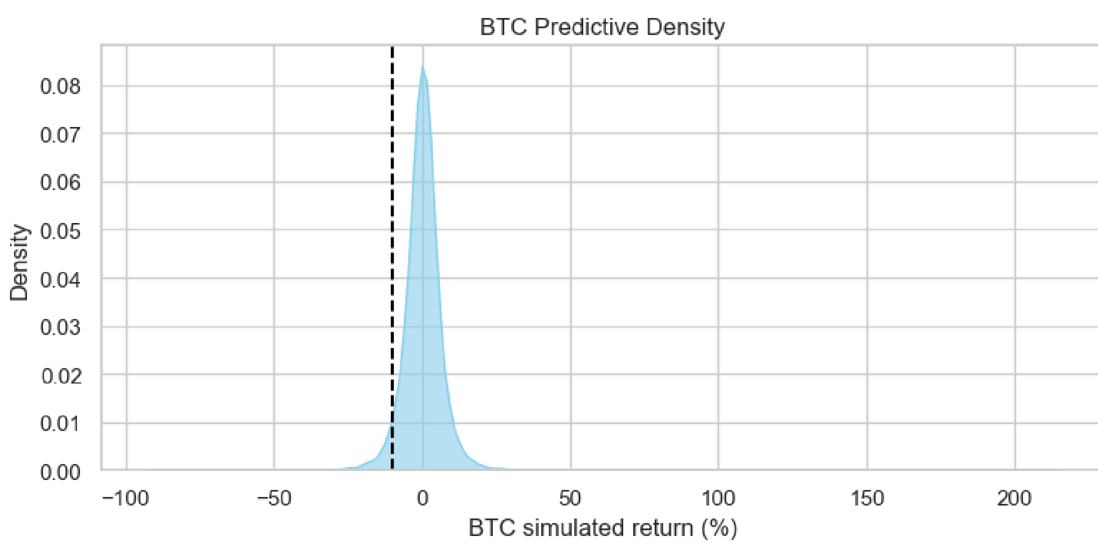
```
    # next-day forecast
    f = res.forecast(horizon=1, reindex=True)
    mu = f.mean.iloc[-1, 0]
    sigma = np.sqrt(f.variance.iloc[-1, 0])
    df = res.params.get("nu", 10)
    return mu, sigma, df

mu_btc, sig_btc, df_btc = fit_garch(btc)
mu_eth, sig_eth, df_eth = fit_garch(eth)

print("GARCH forecasts:")
print("BTC  ,  , df =", mu_btc, sig_btc, df_btc)
print("ETH  ,  , df =", mu_eth, sig_eth, df_eth)

# ================================
# 3. Hand-made t-Copula simulation
# ================================
n_sim = 10000

# Estimate copula rho from PIT uniforms u_btc, v_eth
rho = np.corrcoef(df_ret["u_btc"], df_ret["v_eth"])[0, 1]
nu_cop = 5   # copula DOF (can be tuned)

print("Using t-Copula with rho=", rho, "df=", nu_cop)

# Correlation matrix
R = np.array([[1, rho],
              [rho, 1]])

# Cholesky
L = np.linalg.cholesky(R)

# Step 1: Generate correlated normals
Z = np.random.randn(n_sim, 2)
Z_corr = Z @ L.T

# Step 2: Convert to t-copula draws
chi = np.sqrt(nu_cop / np.random.chisquare(nu_cop, size=(n_sim, 1)))
T_cop = Z_corr * chi

# Step 3: Map to uniforms
u = t.cdf(T_cop[:, 0], df=nu_cop)
v = t.cdf(T_cop[:, 1], df=nu_cop)

# Step 4: Apply marginal inverse CDF from GARCH
sim_btc_eps = t.ppf(u, df_btc)
sim_eth_eps = t.ppf(v, df_eth)
```

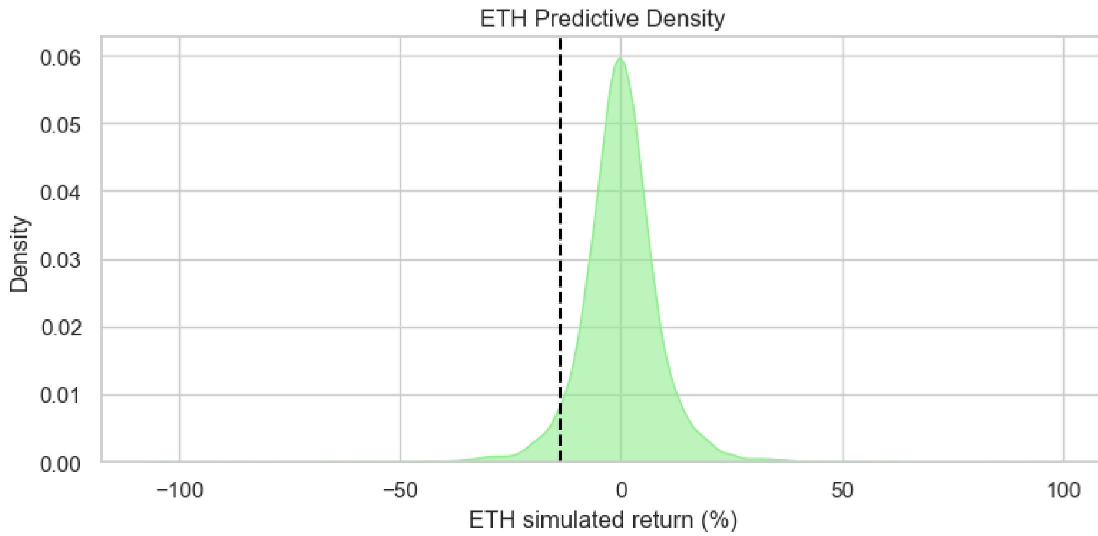
```
sim_btc = mu_btc + sig_btc * sim_btc_eps
sim_eth = mu_eth + sig_eth * sim_eth_eps

# ===============================
# 4. Risk metrics
# ===============================
eth_var_95 = np.percentile(sim_eth, 5)
btc_var_95 = np.percentile(sim_btc, 5)
joint_port = 0.5 * sim_btc + 0.5 * sim_eth
joint_var_95 = np.percentile(joint_port, 5)
co_crash_prob = ((sim_btc < -5) & (sim_eth < -5)).mean()

print("\n=== Monte Carlo (t-Copula + GARCH) ===")
print(f"BTC 95% VaR: {btc_var_95:.3f}%")
print(f"ETH 95% VaR: {eth_var_95:.3f}%")
print(f"Portfolio 95% VaR: {joint_var_95:.3f}%")
print(f"P(BTC<-5% & ETH<-5%): {co_crash_prob:.4f}")

# ===============================
# 5. Beautiful KDE + Contour (Joint Density)
# ===============================
x_low, x_high = np.percentile(sim_btc, [0.5, 99.5])
y_low, y_high = np.percentile(sim_eth, [0.5, 99.5])

plt.figure(figsize=(8,6))
sns.kdeplot(x=sim_btc, y=sim_eth, fill=True, cmap="Blues", levels=40, thresh=0.
 ↪01)
sns.kdeplot(x=sim_btc, y=sim_eth, levels=10, color="black", linewidths=0.8)

plt.xlim(x_low, x_high)
plt.ylim(y_low, y_high)

plt.axvline(-5, color="red", linestyle="--", alpha=0.7)
plt.axhline(-5, color="red", linestyle="--", alpha=0.7)

plt.title("Joint Predictive Distribution (Zoomed)\n(GARCH + t-Copula␣
 ↪Simulation)")
plt.xlabel("BTC simulated return (%)")
plt.ylabel("ETH simulated return (%)")
plt.tight_layout()
plt.show()


# ===============================
# 6. (Optional) Density curves for BTC/ETH
# ===============================
```

```python
plt.figure(figsize=(8,4))
sns.kdeplot(sim_btc, fill=True, color="skyblue", alpha=0.6)
plt.axvline(btc_var_95, color="black", linestyle="--", label="BTC 95% VaR")
plt.title("BTC Predictive Density")
plt.xlabel("BTC simulated return (%)")
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,4))
sns.kdeplot(sim_eth, fill=True, color="lightgreen", alpha=0.6)
plt.axvline(eth_var_95, color="black", linestyle="--", label="ETH 95% VaR")
plt.title("ETH Predictive Density")
plt.xlabel("ETH simulated return (%)")
plt.tight_layout()
plt.show()
```

```
GARCH forecasts:
BTC  ,  , df = 0.13968032525228286 4.143795284847009 3.180525441923541
ETH  ,  , df = -0.21483816664024347 5.962322232696654 3.1150937531509992
Using t-Copula with rho= 0.5465860324628669 df= 5

=== Monte Carlo (t-Copula + GARCH) ===
BTC 95% VaR: -9.847%
ETH 95% VaR: -13.876%
Portfolio 95% VaR: -10.432%
P(BTC<-5% & ETH<-5%): 0.0913
```


Joint Predictive Distribution (Zoomed)
(GARCH + t-Copula Simulation)
ETH simulated return (%)
BTC simulated return (%)
30
20
10
0
−10
−20
−30
−20
−10
0
10
20


BTC Predictive Density
Density
BTC simulated return (%)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−100
−50
0
50
100
150
200


ETH Predictive Density
Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−100
−50
0
50
100
ETH simulated return (%)

```
[ ]:
```