o9s4gt6j1

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
```

[1]:
```
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}")
```

```
d:\miniconda\envs\ec5303\Lib\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

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`.

[5]:
```
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())
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/sudalairajkumar
/cryptocurrencypricehistory?dataset_version_number=3…

100%|      | 1.70M/1.70M [00:01<00:00, 1.56MB/s]

Extracting files…
```

	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

[6]:
```
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.

[7]:
```
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

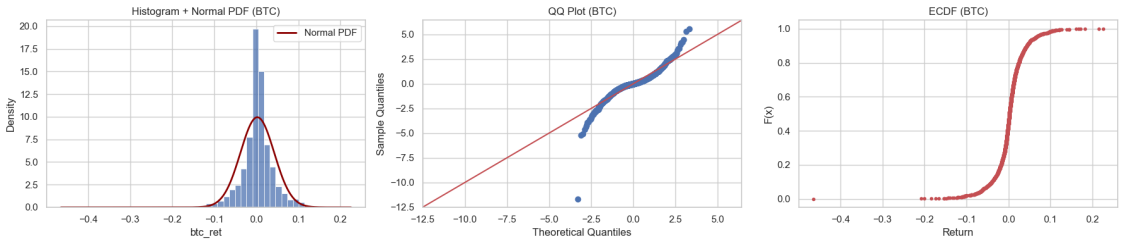
```
Covariance: 0.001356
Correlation: 0.544485
```

[8]:
```
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

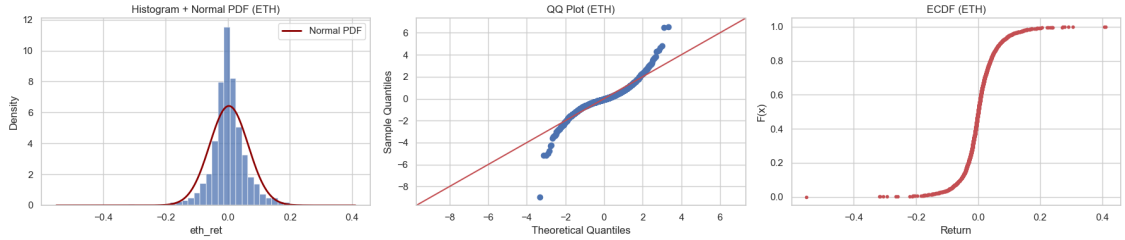
[9]:
```
def plot_distribution(series: pd.Series, title: str):
    mu, sigma = series.mean(), series.std(ddof=1)
    x = np.linspace(series.min(), series.max(), 200)
    normal_pdf = stats.norm.pdf(x, mu, sigma)

    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.
```

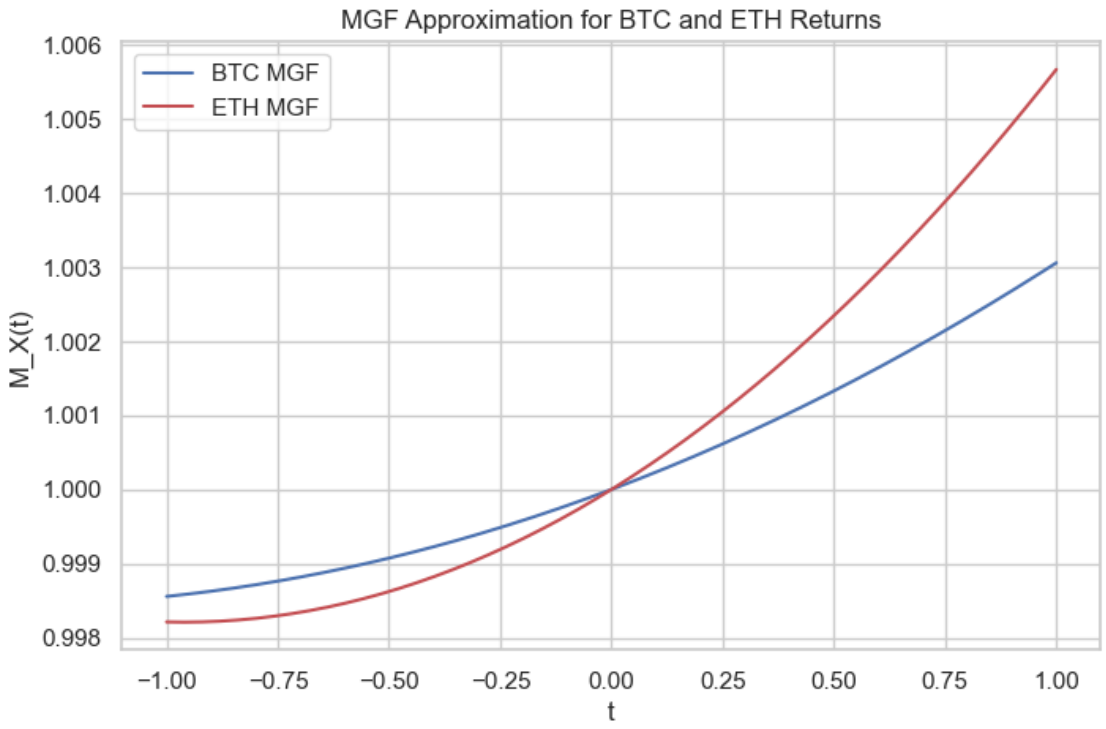
```
ax.plot(x, y, fmt, **plot_style)
```



1.2.1 Moment Generating Function (MGF)

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

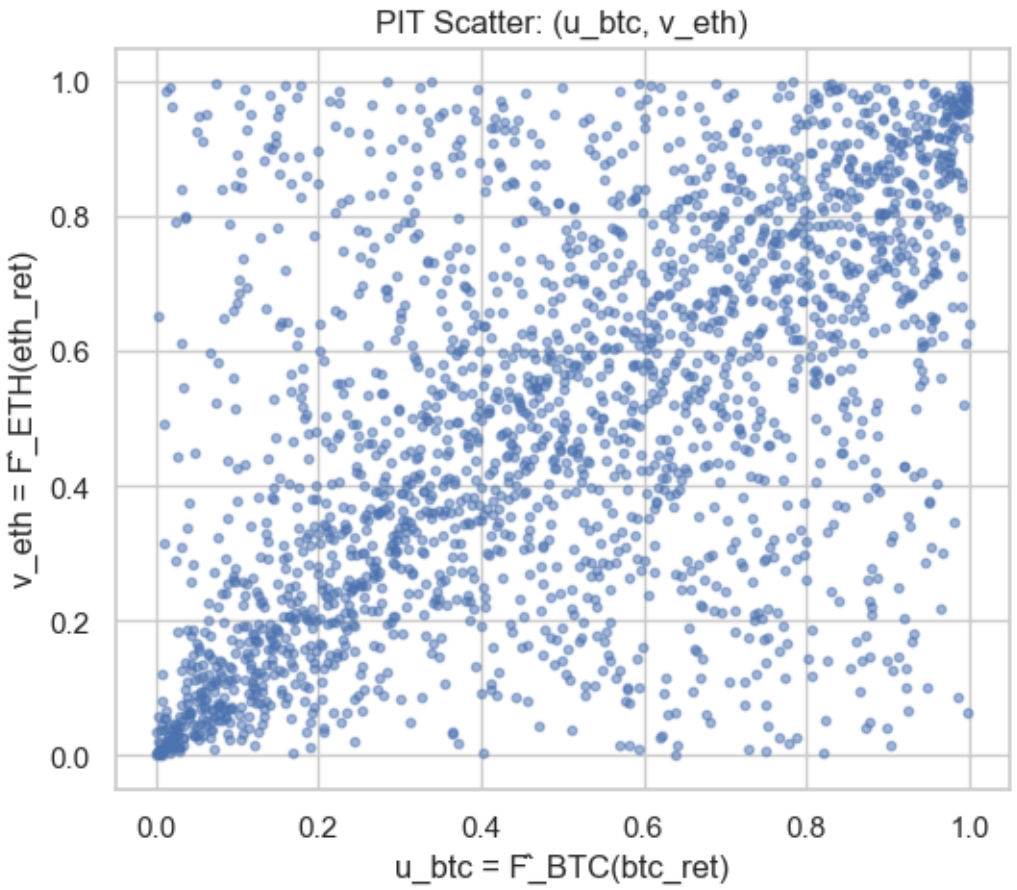
[10]:
```
def mgf_curve(series: pd.Series, t_values: np.ndarray) -> np.ndarray:
    return np.array([np.exp(t * series).mean() for t in t_values])

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).

[11]:
```
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.

[20]:
```
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.

[14]:
```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:	Thu, 27 Nov 2025	**Prob (F-statistic):**	6.27e-167
Time:	15:57:35	**Log-Likelihood:**	3316.5
No. Observations:	2159	**AIC:**	-6629.
Df Residuals:	2157	**BIC:**	-6618.
Df Model:	1		
Covariance Type:	nonrobust		

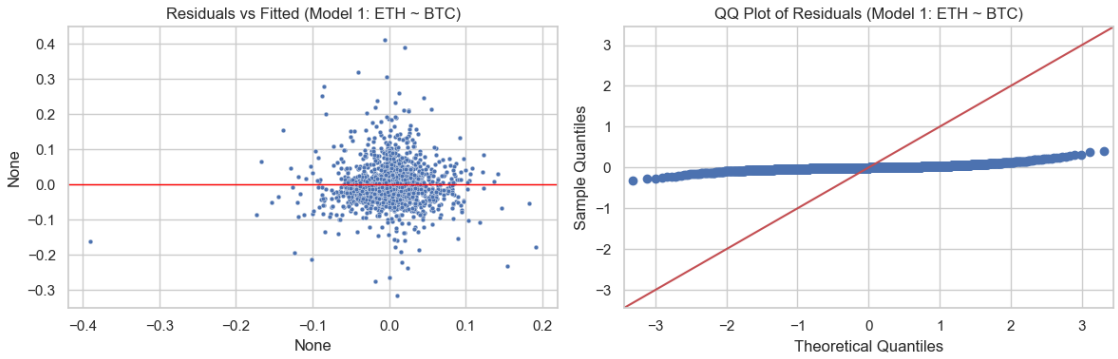
	coef	std err	t	P> |t|	[0.025	0.975]
const	0.0018	0.001	1.618	0.106	-0.000	0.004
btc_ret	0.8431	0.028	30.149	0.000	0.788	0.898

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)
```



[15]:
```
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)
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.")
```

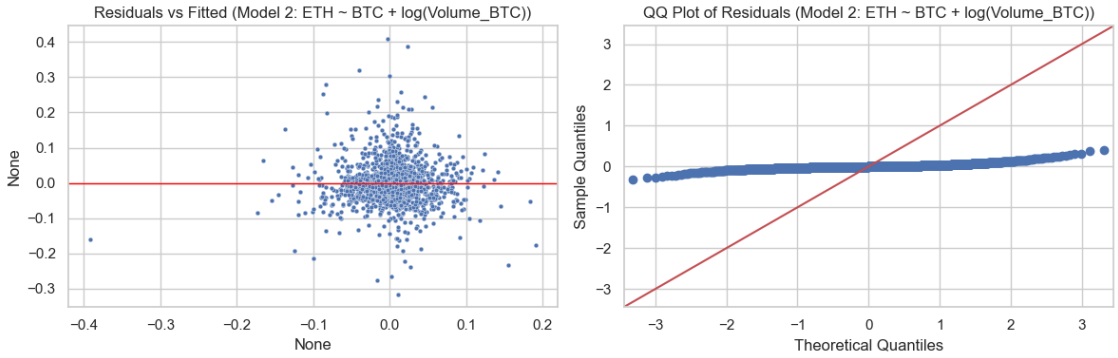
Dep. Variable:	eth_ret	**R-squared:**	0.297
Model:	OLS	**Adj. R-squared:**	0.296
Method:	Least Squares	**F-statistic:**	455.1
Date:	Thu, 27 Nov 2025	**Prob (F-statistic):**	1.30e-165
Time:	15:57:48	**Log-Likelihood:**	3317.1
No. Observations:	2159	**AIC:**	-6628.
Df Residuals:	2156	**BIC:**	-6611.
Df 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`.

[16]:
```
# 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:	Thu, 27 Nov 2025	**Prob (F-statistic):**	3.29e-170
Time:	15:58:21	**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.

[ ]: