In [ ]:
```
1. Find out which probability distribution function best fits Bitcoin’s returns for trading data every minute, from :
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")


#Dataset from 2012 to 2025
df = pd.read_csv("btcusd_1-min_data.csv")
df['Timestamp'] = pd.to_datetime(df['Timestamp'], unit='s')
df = df.sort_values('Timestamp')




# Compute for the log returns
df['log_return'] = np.log(df['Close']).diff()

# Removing NAs
returns = df['log_return'].dropna()
returns = returns[np.isfinite(returns)]

# Sample 1% of the returns for faster distribution fitting
sample_returns = returns.sample(frac=0.01, random_state=42).sort_values()

print("First 10 entries from 2012:")
print(df[df['Timestamp'].dt.year == 2012].head(10))

print("\nFirst 10 entries from 2025:")
print(df[df['Timestamp'].dt.year == 2025].head(10))



# === Histogram with  Kernel Density Estimation ===
plt.figure(figsize=(10, 5))
sns.histplot(sample_returns, bins=100, kde=True, stat="density", color="skyblue")
```

```python
plt.title("Sampled Bitcoin Minute-by-Minute Returns (2012–2025)")
plt.xlabel("Log Return")
plt.ylabel("Density")
plt.grid(True)
plt.tight_layout()
plt.show()


# === Fit Distributions ===
norm_params = stats.norm.fit(sample_returns)
laplace_params = stats.laplace.fit(sample_returns)
t_params = stats.t.fit(sample_returns)

# === Q-Q Plots ===
# Normal Q-Q
sm.qqplot(sample_returns, line='s', dist=stats.norm, loc=norm_params[0], scale=norm_params[1])
plt.title("Q-Q Plot: Normal")
plt.grid(True)
plt.tight_layout()
plt.show()

# Laplace Q-Q
theoretical_laplace = stats.laplace.ppf(np.linspace(0.01, 0.99, len(sample_returns)), *laplace_params)
plt.figure()
plt.scatter(np.sort(theoretical_laplace), sample_returns, alpha=0.3)
plt.plot(sample_returns, sample_returns, 'b-')
plt.title("Q-Q Plot: Laplace")
plt.xlabel("Theoretical")
plt.ylabel("Sample")
plt.grid(True)
plt.tight_layout()
plt.show()

# Student's t Q-Q
theoretical_t = stats.t.ppf(np.linspace(0.01, 0.99, len(sample_returns)), *t_params)
plt.figure()
plt.scatter(np.sort(theoretical_t), sample_returns, alpha=0.3)
plt.plot(sample_returns, sample_returns, 'b-')
plt.title("Q-Q Plot: Student's t")
plt.xlabel("Theoretical")
plt.ylabel("Sample")
plt.grid(True)
```

```python
plt.tight_layout()
plt.show()

# === Kolmogorov-Smirnov Tests on Full Data ===

ks_norm = stats.kstest(returns, 'norm', norm_params)
ks_laplace = stats.kstest(returns, 'laplace', laplace_params)
ks_t = stats.kstest(returns, lambda x: stats.t.cdf(x, *t_params))

# === Output Results ===
print("\n--- Goodness of Fit Results (Fitted on Sample, KS on Full Data) ---")

print("\nNormal Distribution:")
print(f"Parameters: mean = {norm_params[0]:.6f}, std = {norm_params[1]:.6f}")
print(f"KS Statistic = {ks_norm.statistic:.6f}, p-value = {ks_norm.pvalue:.6f}")

print("\nLaplace Distribution:")
print(f"Parameters: location = {laplace_params[0]:.6f}, scale = {laplace_params[1]:.6f}")
print(f"KS Statistic = {ks_laplace.statistic:.6f}, p-value = {ks_laplace.pvalue:.6f}")

print("\nStudent's t Distribution:")
print(f"Parameters: df = {t_params[0]:.2f}, loc = {t_params[1]:.6f}, scale = {t_params[2]:.6f}")
print(f"KS Statistic = {ks_t.statistic:.6f}, p-value = {ks_t.pvalue:.6f}")
```

Interpretation/s:

Minute-by-minute returns of Bitcoin exhibit heavy tails and non-normality. Additionally, Normal, Laplace, and Student's t distribution fits indicate that the heavy-tailed t distribution usually provides the best fit to the data. However, Q-Q plots and goodness-of-fit tests suggest that there are more extreme returns than a normal model would suggest. In general, Bitcoin returns are high in variability and with large swings happening often, which emphasizes the necessity of employing heavy-tailed models when assessing risk.

2. Test using Shapiro-Wilk normality test the Ethereum returns for trading data every five minutes, from August 7, 2015 to April 15, 2025.

In [10]:
```python
import pandas as pd
import numpy as np
from scipy.stats import shapiro
```

```python
# Load first dataset (2015 to 2020)
file_path_1 = r"ETHUSDT_2015_to_2020.csv"
df1 = pd.read_csv(file_path_1)
df1.rename(columns={'Date': 'date', 'Close': 'close'}, inplace=True)
df1['date'] = pd.to_datetime(df1['date'])

# Load second dataset (2020 to 2025)
file_path_2 = r"ETHUSDT_2020_to_2025.csv"
df2 = pd.read_csv(file_path_2)
df2.rename(columns={'date': 'date', 'close': 'close'}, inplace=True)  # just for clarity
df2['date'] = pd.to_datetime(df2['date'])

# Combine datasets
combined_df = pd.concat([df1, df2], ignore_index=True)

# Sort by date
combined_df = combined_df.sort_values('date')

# Set 'date' as index
combined_df.set_index('date', inplace=True)

# Calculate log returns
combined_df['log_return'] = np.log(combined_df['close'] / combined_df['close'].shift(1))
eth_returns = combined_df['log_return'].dropna()

# Sample 5000 returns for the Shapiro-Wilk test (or less if not enough data)
sample_size = min(5000, len(eth_returns))
sample_returns = eth_returns.sample(n=sample_size, random_state=42)

# Perform Shapiro-Wilk normality test
statistic, p_value = shapiro(sample_returns)

# Print results
print("📊 Shapiro-Wilk Normality Test on ETH/USDT 2015-2025 5-min Returns")
print(f"Test Statistic: {statistic}")
print(f"P-value: {p_value}")

if p_value > 0.05:
    print("✅ Returns appear normally distributed (fail to reject H₀).")
else:
    print("❌ Returns are not normally distributed (reject H₀).")
```

```
📊 Shapiro-Wilk Normality Test on ETH/USDT 2015-2025 5-min Returns
Test Statistic: 0.5958860022712331
P-value: 1.2980180286413712e-75
❌ Returns are not normally distributed (reject H₀).
```

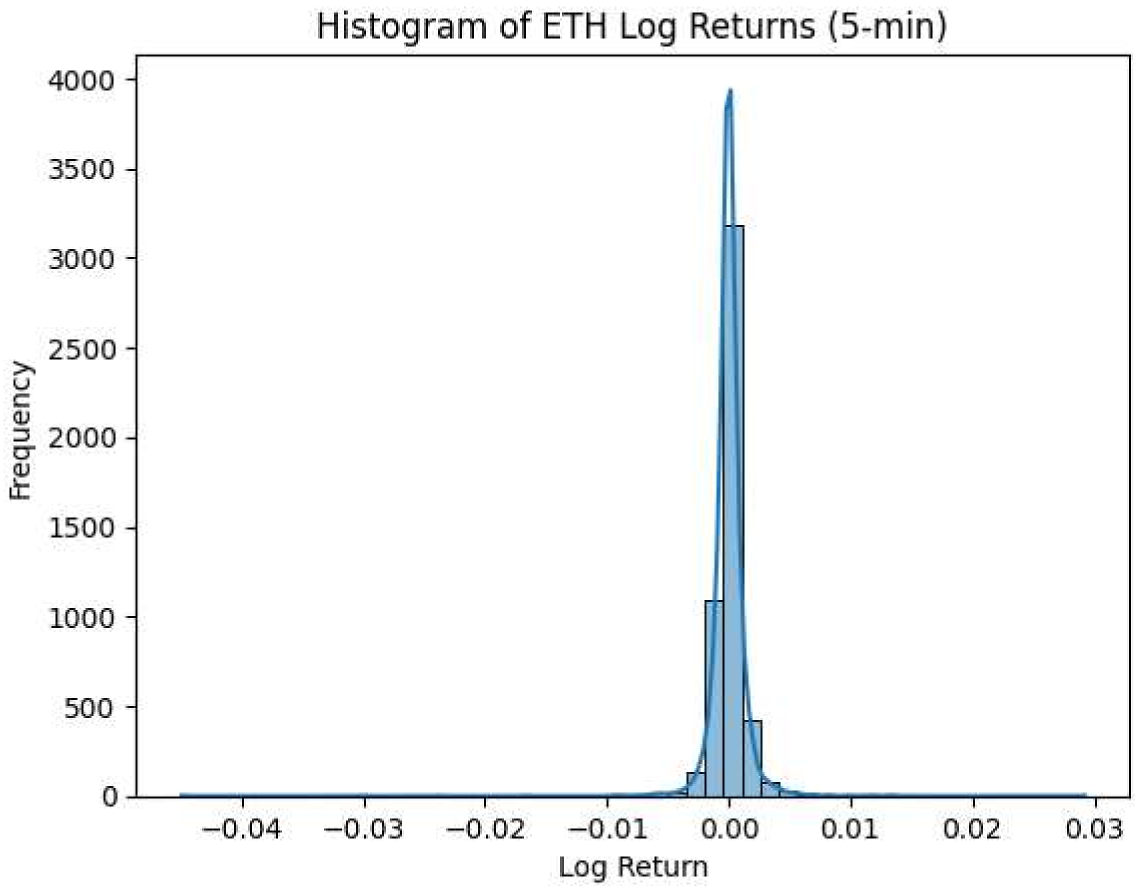
Plotting Histogram:

In [11]:
```
!pip install matplotlib seaborn

import matplotlib.pyplot as plt
import seaborn as sns

sns.histplot(sample_returns, kde=True, bins=50)
plt.title("Histogram of ETH Log Returns (5-min)")
plt.xlabel("Log Return")
plt.ylabel("Frequency")
plt.show()
```

```
Requirement already satisfied: matplotlib in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (3.1
0.3)
Requirement already satisfied: seaborn in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages
(from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (fr
om matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-package
s (from matplotlib) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-package
s (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (fro
m matplotlib) (2.2.5)
Requirement already satisfied: packaging>=20.0 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages
(from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (from
matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages
(from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-pack
ages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: pandas>=1.2 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (fro
m seaborn) (2.2.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (fr
om pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages
(from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\consuelo b. mercado\downloads\sa2\.venv\lib\site-packages (from p
ython-dateutil>=2.7->matplotlib) (1.17.0)
```



The normality of Ethereum 5-minute log returns from August 7, 2015 to April 15, 2025 was assessed using two statistical tests:

Shapiro-Wilk Test (on a random sample of 5,000 returns):

- Test Statistic: 0.5958860022712331
- P-value: 1.2980180286413712e-75
- **Interpretation**: The very low p-value indicates a strong rejection of the null hypothesis. This means the sampled Ethereum returns wass not normally distributed.