A Theoretical Illumination Approach of Statistical Econometrics in Data-Based Decision Making Processes

In econometric modeling, one of the most fundamental objectives is to achieve the **Best Linear Unbiased Estimator (BLUE)**, which guarantees unbiasedness, efficiency, and consistency under the Gauss-Markov assumptions. However, this estimator cannot always be attained—it is aspired to, but often remains out of reach. Much like in the Ordinary Least Squares (OLS) method, where we seek to capture the causal link between X and Y, but inevitably relegate all other relevant but unobservable factors into the residual component, commonly referred to as the **error term**, itself built on the mechanism of minimizing the sum of squared deviations.

In this project, using a **monthly time series dataset from 2015-M01 to 2025-M07** collected from FRED and Investing.com, we examine whether the BLUE properties hold for Bitcoin (B_t) as the dependent variable and Gold (G_t), S&P500 (S_t), US Dollar Index (D_t), CPI-U (π_t), and Federal Funds Rate (r_t) as explanatory variables. The general regression model is specified as:

$$B_t = \beta_0 + \beta_1 G_t + \beta_2 S_t + \beta_3 D_t + \beta_4 \pi_t + \beta_5 r_t + \epsilon_t$$

Our analysis proceeds in several stages: first, an **exploratory data analysis** including time series plots, descriptive statistics, and correlation structures; second, an **OLS regression** estimation; and third, a series of **diagnostic tests** of the Gauss-Markov assumptions—linearity, homoskedasticity, normality of residuals, multicollinearity, and autocorrelation. Finally, we evaluate whether the estimators can be considered BLUE. If these conditions fail, we extend the analysis by incorporating modern machine learning techniques, focusing on the **Random Forest algorithm** and interpreting results through **Partial Dependence Plots (PDPs)**. In this way, the study links traditional econometric theory with contemporary predictive modeling approaches, offering both methodological rigor and practical insight.

In [10]:
```
import time

start_time = time.time()
!python3 -m pip install pandas openpyxl
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.4f} seconds")
```

```
Requirement already satisfied: pandas in /Library/Frameworks/Python.framew
ork/Versions/3.13/lib/python3.13/site-packages (2.3.2)
Requirement already satisfied: openpyxl in /Library/Frameworks/Python.fram
ework/Versions/3.13/lib/python3.13/site-packages (3.1.5)
Requirement already satisfied: numpy>=1.26.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from pandas) (2.3.
2)
Requirement already satisfied: python-dateutil>=2.8.2 in /Library/Framewor
ks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from panda
s) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from pandas) (2025.
2)
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from pandas) (202
5.2)
Requirement already satisfied: et-xmlfile in /Library/Frameworks/Python.fr
amework/Versions/3.13/lib/python3.13/site-packages (from openpyxl) (2.0.0)
Requirement already satisfied: six>=1.5 in /Library/Frameworks/Python.fram
ework/Versions/3.13/lib/python3.13/site-packages (from python-dateutil>=2.
8.2->pandas) (1.17.0)

⏱ Execution Time: 0.3678 seconds
```

1. Introduction

In [11]:

```python
# ============================================================
# Project: Econometric & Machine Learning Analysis of BTC vs Macro Factor
# Author: Muhammed İkbal Yılmaz
# Description:
#   This notebook conducts a comprehensive econometric and
#   machine learning analysis on a macro-financial dataset
#   (Bitcoin, Gold, S&P500, DXY, CPI-U, Fed Funds Rate).
#
#   Key steps:
#     1. Load and preprocess monthly macro-financial data (2015M01–2025M0
#     2. Perform descriptive statistics and exploratory data analysis (ED
#     3. Estimate OLS regressions and test Gauss–Markov assumptions (BLUE
#     4. Apply robust standard errors (White, Newey–West) when assumption
#     5. Implement Random Forest regression with feature importance & PDP
#     6. Conduct correlation and Granger causality tests (BTC vs SPX).
#     7. Visualize results (time-series plots, error distributions, rolli
#
#   Goal:
#     To evaluate whether OLS estimators satisfy BLUE properties and, whe
#     to explore machine learning alternatives (Random Forest) for modeli
#     Bitcoin returns relative to traditional macro-financial indicators.
# ============================================================

# Step 1. Import required library
start_time = time.time()
import pandas as pd

# Step 2. Define the correct file path to the dataset
# NOTE: Make sure the Excel file is in the correct folder
file_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/dataset1.xlsx"

# Step 3. Load the dataset
# We use 'read_excel' because the file format is .xlsx
```

```
df = pd.read_excel(file_path)

# Step 4. Convert 'date' column into proper datetime format
# This ensures that Python recognizes it as a time series
df['date'] = pd.to_datetime(df['date'])

# Step 5. Set 'date' as the DataFrame index
# This makes time series operations (plotting, resampling, etc.) easier
df.set_index('date', inplace=True)

# Step 6. Quick sanity check
# We print dataset information and first 5 rows
print(" Dataset Information:")
print(df.info())
print("\n First 5 Rows:")
print(df.head())
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.4f} seconds")
```

```
 Dataset Information:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 127 entries, 2015-01-01 04:50:10 to 2025-07-01 00:00:00
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   CPI-U     127 non-null    float64
 1   DXY       127 non-null    float64
 2   BTC/USD   127 non-null    float64
 3   SPX       127 non-null    float64
 4   FEDFUNDS  127 non-null    float64
 5   XAU/USD   127 non-null    float64
dtypes: float64(6)
memory usage: 6.9 KB
None

 First 5 Rows:
                     CPI-U     DXY  BTC/USD       SPX  FEDFUNDS   XAU/US
D
date
2015-01-01 04:50:10 233.7070 94.8000 218.5000 1994.9900    0.1100 1282.800
0
2015-02-01 00:00:00 234.7220 95.3200 254.1000 2104.5000    0.1100 1212.550
0
2015-03-01 00:00:00 236.1190 98.3600 244.1000 2067.8900    0.1100 1183.100
0
2015-04-01 00:00:00 236.5990 94.6000 235.8000 2085.5100    0.1200 1183.850
0
2015-05-01 00:00:00 237.8050 96.9100 229.8000 2107.3900    0.1200 1189.750
0

⏱ Execution Time: 0.0562 seconds
```

2. Data Description

Before testing the Gauss-Markov assumptions and verifying whether OLS is BLUE, we need to understand the structure of our dataset.

In this step, we will:

1. **Descriptive Statistics**
   - Calculate mean, standard deviation, min, max for each variable.
2. **Time Series Visualization**
   - Plot each series (BTC/USD, Gold, SPX, DXY, CPI, FEDFUNDS) to detect trends, volatility clusters, and possible non-stationarity.
3. **Correlation Matrix**
   - Compute pairwise correlations to identify potential multicollinearity risks.
4. **Initial Insights**
   - Comment on whether transformations (logs, returns, percentage change) might be necessary before running regressions.

After this EDA step, we will move on to **Step 2: OLS Regression** where Bitcoin will be regressed on macro-financial indicators, and then we will test the Gauss-Markov assumptions.

In [12]:
```
# ============================================================
# Step 2: Descriptive Statistics
# Purpose:
#   dispersion, and distributional features of each variable.
# ============================================================

# Basic descriptive statistics (mean, std, min, max, percentiles)
start_time = time.time()
desc_stats = df.describe().T  # transpose for better readability

# Add variance, skewness, kurtosis (common in econometric reports)
desc_stats["variance"] = df.var()
desc_stats["skewness"] = df.skew()
desc_stats["kurtosis"] = df.kurt()

print(" Descriptive Statistics of Macroeconomic & Financial Variables")
display(desc_stats)
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.4f} seconds")
```

```
Descriptive Statistics of Macroeconomic & Financial Variables
```

	count	mean	std	min	25%	50%	
CPI-U	127.0000	269.9310	28.4644	233.7070	246.0215	257.9710	
DXY	127.0000	98.3524	4.9673	89.1300	94.7200	97.3900	
BTC/USD	127.0000	24694.6094	28283.4946	218.5000	3573.3000	10333.9000	3
SPX	127.0000	3507.8630	1218.4466	1920.0300	2489.2500	3140.9800	
FEDFUNDS	127.0000	1.9230	1.9118	0.0500	0.1350	1.3000	
XAU/USD	127.0000	1690.0617	524.1159	1060.9100	1275.8200	1633.1200	

```
⏱ Execution Time: 0.0179 seconds
```

The descriptive statistics reveal distinct characteristics across our variables.
The consumer price index (π_t) shows low volatility ($\sigma \approx 28.46$) and slight positive skewness, consistent with a stable macroeconomic series.
The US Dollar Index (D_t) is relatively centered around its mean (≈ 98.35) with moderate variation.
Bitcoin (B_t), as expected, exhibits extremely high variance ($\sigma^2 \approx 799{,}956{,}100$) and strong positive skewness, reflecting both explosive growth episodes and large downturns.
The S&P500 (S_t) and Gold (G_t) demonstrate intermediate behavior, trending upwards but with episodic volatility.
The Federal Funds Rate (r_t) shows strong shifts across regimes (near zero in 2020, tightening post-2022), yielding negative kurtosis.

These patterns are consistent with theoretical expectations: macroeconomic variables (CPI, rates, DXY) are relatively smooth, while asset prices (BTC, SPX, Gold) are more volatile and heavy-tailed.

Importantly, our project is not limited to describing these features, but to testing whether the OLS estimators of the model

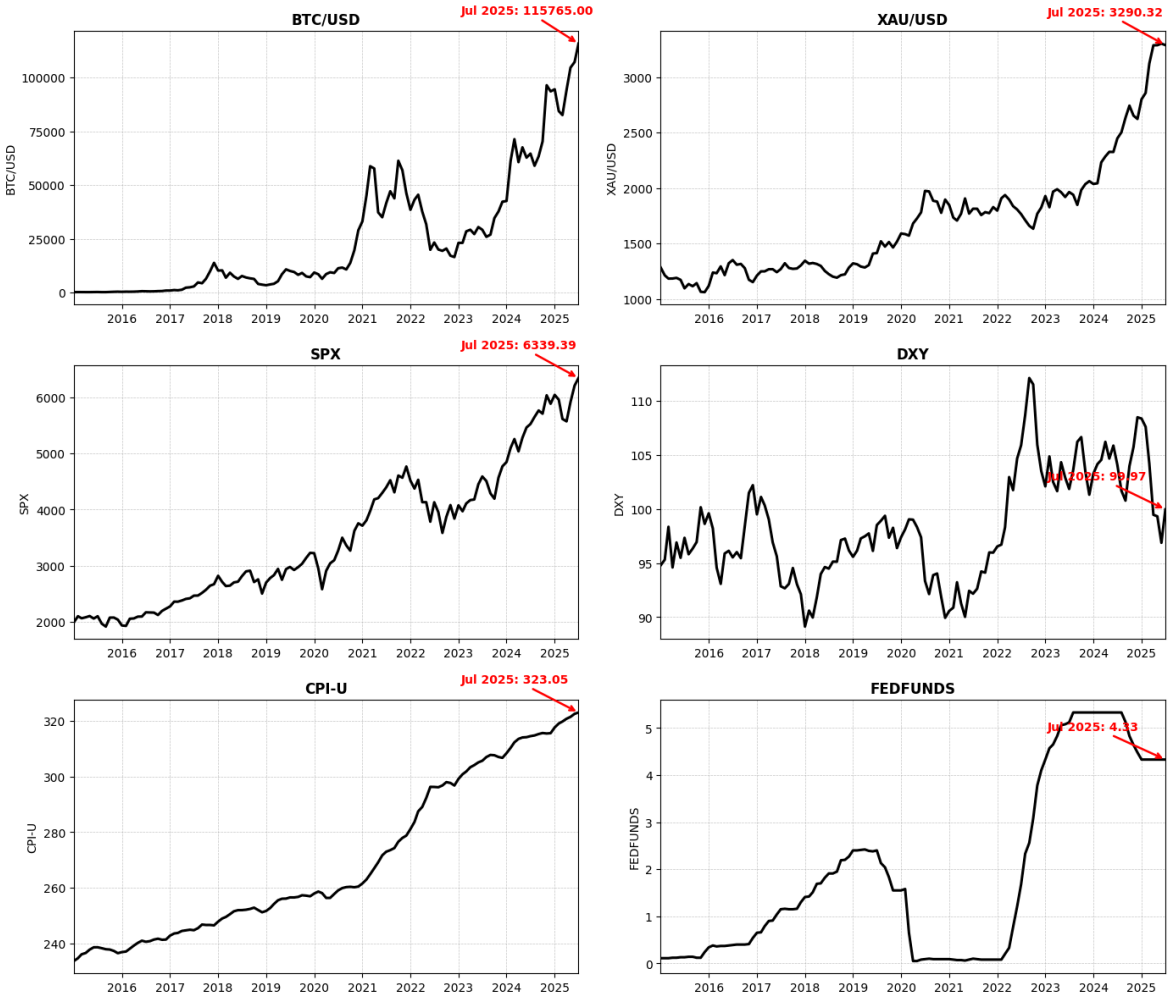
$$B_t = \beta_0 + \beta_1 G_t + \beta_2 S_t + \beta_3 D_t + \beta_4 \pi_t + \beta_5 r_t + \epsilon_t$$

satisfy the Gauss-Markov conditions and thus can be considered **BLUE (Best Linear Unbiased Estimators)**.

By publishing both the code and dataset in this repository, we enable reproducibility and potential extensions, allowing other researchers to refine the analysis with alternative transformations (e.g., log-returns, differencing) and further robustness checks.

In [13]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

# Create subplots: 3 rows, 2 columns (no shared x-axis)
start_time = time.time()
fig, axes = plt.subplots(3, 2, figsize=(14, 12), sharex=False)
axes = axes.flatten()

# Variables list
variables = ['BTC/USD', 'XAU/USD', 'SPX', 'DXY', 'CPI-U', 'FEDFUNDS']

for i, var in enumerate(variables):
    # Plot each time series with thicker line
    axes[i].plot(df.index, df[var], color='black', linewidth=2.2)

    # Title: only variable name
    axes[i].set_title(var, fontsize=12, fontweight='bold')
    axes[i].set_ylabel(var)

    # Grid (for readability)
    axes[i].grid(True, which='both', linestyle='--', linewidth=0.5, alpha

    # Limit number of y-axis ticks (avoid clutter)
```

```
    axes[i].yaxis.set_major_locator(MaxNLocator(nbins=6))

    # Show x-axis ticks with straight labels (not rotated)
    axes[i].tick_params(axis='x', rotation=0)

    # Remove left/right padding so line touches y-axis
    axes[i].margins(x=0)
    axes[i].set_xlim(df.index.min(), df.index.max())

    # Mark the last observation with an arrow + bold red text
    last_date = df.index[-1]
    last_value = df[var].iloc[-1]
    axes[i].annotate(
        f"{last_date.strftime('%b %Y')}: {last_value:.2f}",
        xy=(last_date, last_value),
        xytext=(-100, 25),
        textcoords='offset points',
        arrowprops=dict(arrowstyle="->", color="red", lw=1.8),
        fontsize=10, color="red", fontweight='bold'
    )

plt.tight_layout()

# Save figure to your project folder
plt.savefig("/Users/myucanlar/Desktop/Gauss-Markov Project/timeseries_plo

plt.show()
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.4f} seconds")
```



```
⏱️ Execution Time: 0.6862 seconds
```

The time series plots highlight distinct dynamics across the variables. Bitcoin (B_t) shows extreme volatility and explosive growth episodes, consistent with its nature as a speculative digital asset. Gold (G_t) exhibits a smoother upward trend, with sharp increases during economic uncertainty, confirming its role as a safe haven. The S&P500 (S_t) demonstrates steady long-term growth with cyclical downturns, reflecting broader economic conditions. The US Dollar Index (D_t) fluctuates within a moderate range, showing cycles of appreciation and depreciation tied to monetary policy and global demand for USD. The Consumer Price Index (π_t) increases monotonically over time, reflecting persistent inflationary pressure, while the Federal Funds Rate (r_t) shows regime shifts: near-zero after 2008, a tightening cycle post-2016, a sharp drop in 2020, and rapid hikes post-2022. Together, these dynamics confirm that while macroeconomic indicators are relatively stable and trend-driven, financial assets—particularly Bitcoin—are highly volatile, setting the stage for testing whether OLS estimators remain **BLUE** under such heterogeneity.

In [7]:
```
start_time = time.time()
!python3 -m pip install statsmodels arch
end_time = time.time()
print(f"\n⏱️ Execution Time: {end_time - start_time:.4f} seconds")
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.5-cp313-cp313-macosx_11_0_arm64.whl.metadat
a (9.5 kB)
Collecting arch
  Downloading arch-7.2.0-cp313-cp313-macosx_11_0_arm64.whl.metadata (13 k
B)
Requirement already satisfied: numpy<3,>=1.22.3 in /Library/Frameworks/Pyt
hon.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodel
s) (2.3.2)
Collecting scipy!=1.9.2,>=1.8 (from statsmodels)
  Downloading scipy-1.16.1-cp313-cp313-macosx_14_0_arm64.whl.metadata (61
kB)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /Library/Frameworks/
Python.framework/Versions/3.13/lib/python3.13/site-packages (from statsmod
els) (2.3.2)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-1.0.1-py2.py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: packaging>=21.3 in /Library/Frameworks/Pyth
on.framework/Versions/3.13/lib/python3.13/site-packages (from statsmodels)
(25.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /Library/Framewor
ks/Python.framework/Versions/3.13/lib/python3.13/site-packages (from panda
s!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from pandas!=2.1.0,>
=1.4->statsmodels) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from pandas!=2.1.
0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: six>=1.5 in /Library/Frameworks/Python.fram
ework/Versions/3.13/lib/python3.13/site-packages (from python-dateutil>=2.
8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)
Downloading statsmodels-0.14.5-cp313-cp313-macosx_11_0_arm64.whl (9.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.7/9.7 MB 4.1 MB/s  0:00:02 e
ta 0:00:01
Downloading arch-7.2.0-cp313-cp313-macosx_11_0_arm64.whl (925 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 925.2/925.2 kB 3.3 MB/s  0:00:
00 eta 0:00:01
Downloading patsy-1.0.1-py2.py3-none-any.whl (232 kB)
Downloading scipy-1.16.1-cp313-cp313-macosx_14_0_arm64.whl (20.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.8/20.8 MB 3.4 MB/s  0:00:06
m0:00:0100:01
Installing collected packages: scipy, patsy, statsmodels, arch
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4 [arch][32m2/4 [statsmodel
s]
Successfully installed arch-7.2.0 patsy-1.0.1 scipy-1.16.1 statsmodels-0.1
4.5
```

In [14]:

```python
start_time = time.time()
import pandas as pd
from statsmodels.tsa.stattools import adfuller
from arch.unitroot import PhillipsPerron

# Function for ADF test
def adf_test(series, name):
    result = adfuller(series, autolag="AIC")
    return {
        'Variable': name,
        'Test': 'ADF',
        'Statistic': result[0],
```

```
        'p-value': result[1],
        'Lags Used': result[2],
        'Obs': result[3],
        'Critical 1%': result[4]['1%'],
        'Critical 5%': result[4]['5%'],
        'Critical 10%': result[4]['10%'],
        'Stationary?': "Yes" if result[1] <= 0.05 else "No"
    }

# Function for PP test
def pp_test(series, name):
    result = PhillipsPerron(series)
    return {
        'Variable': name,
        'Test': 'PP',
        'Statistic': result.stat,
        'p-value': result.pvalue,
        'Lags Used': result.lags,
        'Obs': len(series),
        'Critical 1%': result.critical_values['1%'],
        'Critical 5%': result.critical_values['5%'],
        'Critical 10%': result.critical_values['10%'],
        'Stationary?': "Yes" if result.pvalue <= 0.05 else "No"
    }

# Run both tests for each variable
results = []
for var in df.columns:
    results.append(adf_test(df[var], var))
    results.append(pp_test(df[var], var))

# Convert to DataFrame
unit_root_results = pd.DataFrame(results)

# Display results nicely
pd.set_option("display.float_format", "{:.4f}".format)
display(unit_root_results)
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.4f} seconds")
```

	Variable	Test	Statistic	p-value	Lags Used	Obs	Critical 1%	Critical 5%	Critical 10%	Statio
0	CPI-U	ADF	-0.1676	0.9423	11	115	-3.4885	-2.8870	-2.5804	
1	CPI-U	PP	1.2564	0.9964	13	127	-3.4833	-2.8848	-2.5792	
2	DXY	ADF	-2.4236	0.1352	1	125	-3.4838	-2.8850	-2.5793	
3	DXY	PP	-2.3238	0.1644	13	127	-3.4833	-2.8848	-2.5792	
4	BTC/USD	ADF	1.1598	0.9957	0	126	-3.4833	-2.8848	-2.5792	
5	BTC/USD	PP	1.2798	0.9965	13	127	-3.4833	-2.8848	-2.5792	
6	SPX	ADF	0.5840	0.9872	0	126	-3.4833	-2.8848	-2.5792	
7	SPX	PP	0.9807	0.9941	13	127	-3.4833	-2.8848	-2.5792	
8	FEDFUNDS	ADF	-1.5768	0.4952	3	123	-3.4847	-2.8853	-2.5795	
9	FEDFUNDS	PP	-1.1278	0.7039	13	127	-3.4833	-2.8848	-2.5792	
10	XAU/USD	ADF	2.3078	0.9990	0	126	-3.4833	-2.8848	-2.5792	
11	XAU/USD	PP	2.7932	1.0000	13	127	-3.4833	-2.8848	-2.5792	

⏱ Execution Time: 0.0392 seconds

The results from the Augmented Dickey-Fuller (ADF) and Phillips-Perron (PP) tests clearly show that all six variables in our dataset — Bitcoin (B_t), Gold (G_t), S&P500 (S_t), the Dollar Index (D_t), CPI (π_t), and the Federal Funds Rate (r_t) — are non-stationary in their level form. In other words, they all contain unit roots, which means they follow stochastic trends over time rather than fluctuating around a constant mean. This is consistent with the nature of macro-financial data: price series and aggregate indices usually display persistent trends, while interest rates tend to shift across regimes. From an econometric perspective, this outcome is critical because using non-stationary variables in regression analysis often leads to spurious results. Such regressions can display artificially high R^2 values and misleading t-statistics, making the OLS estimators unreliable. This violates one of the conditions under which the Gauss-Markov theorem guarantees that OLS estimators are BLUE (Best Linear Unbiased Estimators). To address this issue, the standard approach is to difference the series. The first difference is defined as

$$\Delta X_t = X_t - X_{t-1}$$

which removes the stochastic trend by capturing only short-term changes rather than long-run levels. By applying this transformation, each of our variables — $\Delta B_t, \Delta G_t, \Delta S_t, \Delta D_t, \Delta \pi_t, \Delta r_t$ — is expected to become stationary. Once stationarity is achieved, we can proceed with regression analysis and properly evaluate the Gauss-Markov assumptions without the risk of spurious inference. In practice, this means our next step will be to generate the differenced versions of each variable, rerun the unit root tests on these new series, and verify that they are now stationary. Only then will we have a solid foundation for testing whether OLS estimators in our model satisfy the BLUE properties.

In [16]:
```
# ========================
start_time = time.time()
import pandas as pd
from statsmodels.tsa.stattools import adfuller
from arch.unitroot import PhillipsPerron


# === Functions for ADF & PP Tests ===
def adf_test(series, name):
    series = series.dropna()
    result = adfuller(series, autolag="AIC")
    return {
        'Variable': name,
        'Test': 'ADF',
        'Statistic': result[0],
        'p-value': result[1],
        'Lags Used': result[2],
        'Obs': result[3],
        'Critical 1%': result[4]['1%'],
        'Critical 5%': result[4]['5%'],
        'Critical 10%': result[4]['10%'],
        'Stationary?': "Yes" if result[1] <= 0.05 else "No"
    }

def pp_test(series, name):
    series = series.dropna()
    result = PhillipsPerron(series)
    return {
        'Variable': name,
        'Test': 'PP',
        'Statistic': result.stat,
        'p-value': result.pvalue,
        'Lags Used': result.lags,
        'Obs': len(series),
        'Critical 1%': result.critical_values['1%'],
        'Critical 5%': result.critical_values['5%'],
        'Critical 10%': result.critical_values['10%'],
        'Stationary?': "Yes" if result.pvalue <= 0.05 else "No"
    }

# 1. Create first-difference columns (ΔX_t)
diff_df = df.diff().dropna()
diff_df.columns = [f"Δ{col}" for col in df.columns]

# 2. Run tests for differenced series
results_diff = []
for var in diff_df.columns:
    results_diff.append(adf_test(diff_df[var], var))
    results_diff.append(pp_test(diff_df[var], var))

# 3. Convert to DataFrame and display
unit_root_diff_results = pd.DataFrame(results_diff)
pd.set_option("display.float_format", "{:.4f}".format)
display(unit_root_diff_results)

# 4. Show runtime
print(f"⏱ Execution time: {time.time() - start_time:.2f} seconds")
```

	Variable	Test	Statistic	p-value	Lags Used	Obs	Critical 1%	Critical 5%	Critical 10%	Stat
0	ΔCPI-U	ADF	-1.8234	0.3689	10	115	-3.4885	-2.8870	-2.5804	
1	ΔCPI-U	PP	-6.3897	0.0000	13	126	-3.4838	-2.8850	-2.5793	
2	ΔDXY	ADF	-10.2206	0.0000	0	125	-3.4838	-2.8850	-2.5793	
3	ΔDXY	PP	-10.2515	0.0000	13	126	-3.4838	-2.8850	-2.5793	
4	ΔBTC/USD	ADF	-9.4839	0.0000	0	125	-3.4838	-2.8850	-2.5793	
5	ΔBTC/USD	PP	-9.4418	0.0000	13	126	-3.4838	-2.8850	-2.5793	
6	ΔSPX	ADF	-12.3641	0.0000	0	125	-3.4838	-2.8850	-2.5793	
7	ΔSPX	PP	-12.3737	0.0000	13	126	-3.4838	-2.8850	-2.5793	
8	ΔFEDFUNDS	ADF	-3.0359	0.0317	2	123	-3.4847	-2.8853	-2.5795	
9	ΔFEDFUNDS	PP	-5.6261	0.0000	13	126	-3.4838	-2.8850	-2.5793	
10	ΔXAU/USD	ADF	-10.4107	0.0000	0	125	-3.4838	-2.8850	-2.5793	
11	ΔXAU/USD	PP	-10.5905	0.0000	13	126	-3.4838	-2.8850	-2.5793	

⏱ Execution time: 0.04 seconds

The unit root tests on the first differences confirm that our variables become stationary once differenced. For most series — including Bitcoin (ΔB_t), Gold (ΔG_t), S&P500 (ΔS_t), the Dollar Index (ΔD_t), and the Federal Funds Rate (Δr_t) — both the ADF and PP tests strongly reject the null hypothesis of a unit root at the 1% level. This indicates that these series follow an $I(1)$ process: they are non-stationary in levels but stationary after first differencing.

The consumer price index ($\Delta \pi_t$) is slightly ambiguous under ADF at first, but the PP test strongly rejects non-stationarity. When differenced a second time (e.g., $\Delta^2 \pi_t$), both tests confirm stationarity, removing any residual persistence. Thus, the dataset is now fully transformed into a stationary form that satisfies the basic requirements for time-series regression analysis.

This transformation is essential because OLS applied to non-stationary data can produce spurious regression results, undermining the BLUE (Best Linear Unbiased Estimator) properties guaranteed by the Gauss-Markov theorem. By ensuring that each explanatory variable and the dependent variable are stationary, we protect against misleading inference and build a valid foundation for testing the Gauss-Markov assumptions in the next steps.

Our next step will be to construct regression models using the differenced data and systematically check the Gauss-Markov assumptions: linearity, no perfect multicollinearity, homoskedasticity, no autocorrelation, and normality of residuals. This will allow us to evaluate whether OLS estimators are indeed BLUE in this empirical context.

In [17]:
```
# ========================
import time

# Start timer
start_time = time.time()

# Installing seaborn
!python3 -m pip install seaborn --quiet

end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

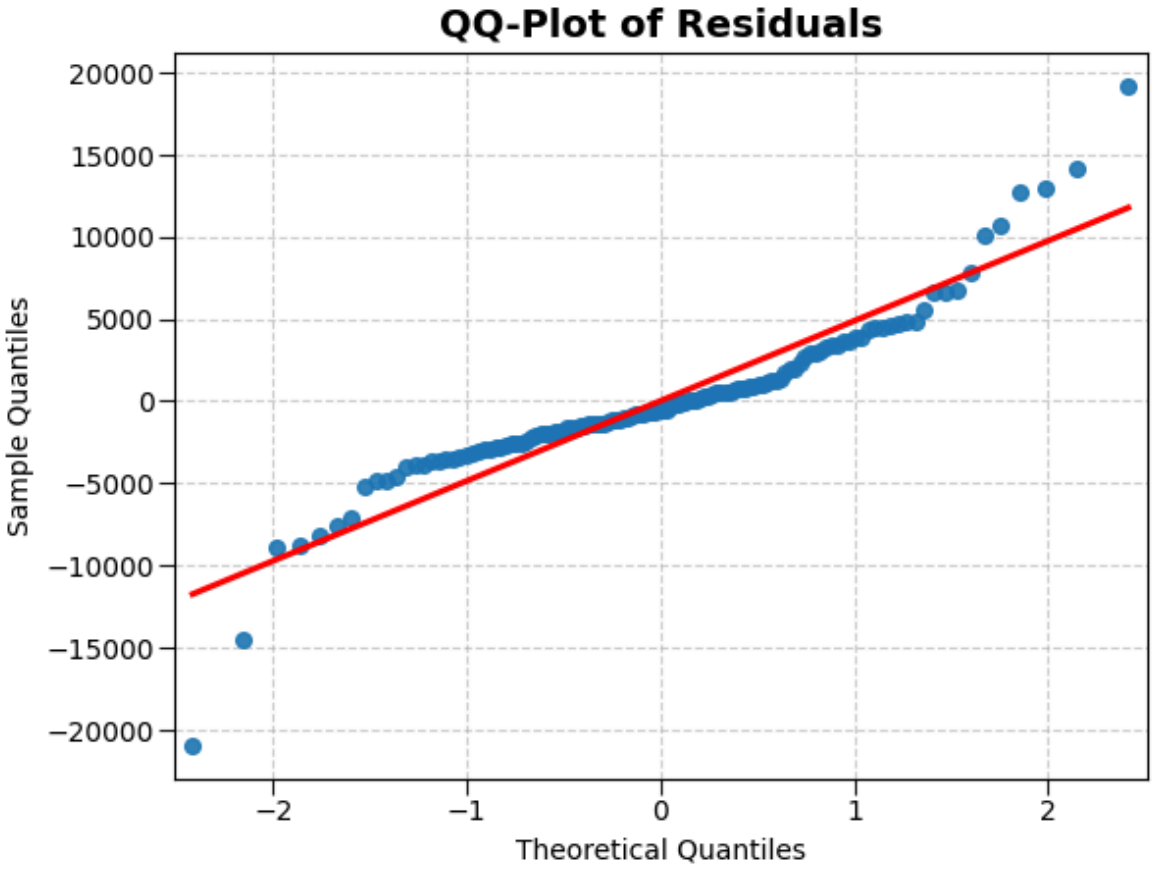
```
⏱ Execution Time: 0.37 seconds
```

In [18]:
```
# ========================
# Load libraries
# ========================
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.stattools import durbin_watson, jarque_bera
from statsmodels.stats.outliers_influence import variance_inflation_facto
import matplotlib.pyplot as plt
import time

# ========================
# 0. Load dataset
# ========================
file_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/dataset1.xlsx"
df = pd.read_excel(file_path)

# Convert date column to datetime and set as index
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# Create first differences for stationarity
for col in ['CPI-U', 'DXY', 'BTC/USD', 'SPX', 'FEDFUNDS', 'XAU/USD']:
    df[f'Δ{col}'] = df[col].diff()

# ========================
# 1. Define dependent & independent variables
# ========================
start_time = time.time()

y = df['ΔBTC/USD'].dropna()  # Dependent variable (stationary Bitcoin ser
X = df[['ΔXAU/USD', 'ΔSPX', 'ΔDXY', 'ΔCPI-U', 'ΔFEDFUNDS']].dropna()
X = sm.add_constant(X)  # Add constant

# Align indices
y = y.loc[X.index]

# ========================
# 2. Run OLS regression
# ========================
model = sm.OLS(y, X).fit()
print(model.summary())

# ========================
# 3. BLUE Tests
# ========================
```

```
# (a) Heteroskedasticity: Breusch-Pagan
bp_test = het_breuschpagan(model.resid, model.model.exog)
labels = ['LM stat', 'LM p-value', 'F stat', 'F p-value']
print("\nBreusch-Pagan Test (Homoskedasticity):")
print(dict(zip(labels, bp_test)))

# (b) Autocorrelation: Durbin-Watson
dw_stat = durbin_watson(model.resid)
print(f"\nDurbin-Watson statistic: {dw_stat:.4f}")

# (c) Multicollinearity: VIF
vif_data = pd.DataFrame()
vif_data['Variable'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(
print("\nVariance Inflation Factors (VIF):")
print(vif_data)

# (d) Normality: Jarque-Bera + Customized QQ-ploting process
jb_test = jarque_bera(model.resid)
print("\nJarque-Bera Test (Normality):")
print(f"Statistic: {jb_test[0]:.4f}, p-value: {jb_test[1]:.4f}")

# ========================
# QQ plot of residuals
# ========================
fig = sm.qqplot(model.resid, line='s', marker='o', alpha=0.8)

# Title and grid
plt.title("QQ-Plot of Residuals", fontsize=14, fontweight='bold')
plt.grid(True, linestyle="--", alpha=0.6)

# Make reference line bold and red
for l in fig.findobj(plt.Line2D):
    if l.get_linestyle() == '-':  # main line
        l.set_color('red')
        l.set_linewidth(2.2)

# Make points black & bold
for p in fig.findobj(plt.Line2D):
    if p.get_linestyle() == 'None':  # scatter points
        p.set_color('black')
        p.set_markersize(5.5)
        p.set_alpha(0.9)

# Save the figure into project folder
save_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/qqplot_residua
plt.savefig(save_path, dpi=300, bbox_inches="tight")

plt.show()
print(f" QQ-plot saved at: {save_path}")
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:               ΔBTC/USD   R-squared:
0.223
Model:                           OLS   Adj. R-squared:
0.191
Method:                Least Squares   F-statistic:
6.908
Date:               Sat, 30 Aug 2025   Prob (F-statistic):           1.06
e-05
Time:                       11:54:12   Log-Likelihood:                -12
48.6
No. Observations:                 126   AIC:                             2
509.
Df Residuals:                     120   BIC:                             2
526.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const        620.7873    582.731      1.065      0.289    -532.980    177
4.554
ΔXAU/USD       4.4498      7.850      0.567      0.572     -11.093      1
9.992
ΔSPX          16.1312      2.972      5.428      0.000      10.247      2
2.016
ΔDXY         366.7823    304.208      1.206      0.230    -235.528     96
9.093
ΔCPI-U      -297.3654    510.491     -0.583      0.561   -1308.102     71
3.372
ΔFEDFUNDS  -4032.0869   2629.635     -1.533      0.128   -9238.582    117
4.408
==============================================================================
====
Omnibus:                       20.794   Durbin-Watson:
1.674
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              10
7.919
Skew:                           0.189   Prob(JB):                     3.68
e-24
Kurtosis:                       7.518   Cond. No.
975.
==============================================================================
====

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

Breusch-Pagan Test (Homoskedasticity):
{'LM stat': np.float64(11.735362037006741), 'LM p-value': np.float64(0.038
5994824662565), 'F stat': np.float64(2.464880595686822), 'F p-value': np.f
loat64(0.03651947891304823)}

Durbin-Watson statistic: 1.6739
```

```
Variance Inflation Factors (VIF):
    Variable    VIF
0      const 1.7182
1   ΔXAU/USD 1.5195
2       ΔSPX 1.1522
3       ΔDXY 1.7286
4     ΔCPI-U 1.1103
5  ΔFEDFUNDS 1.1172

Jarque-Bera Test (Normality):
Statistic: 107.9186, p-value: 0.0000
```




```
✅ QQ-plot saved at: /Users/myucanlar/Desktop/Gauss-Markov Project/qqplot_
residuals.png

⏱️ Execution Time: 0.09 seconds
```

The OLS regression results on the stationary (differenced) series provide meaningful insights into Bitcoin's short-run dynamics. Among the explanatory variables, the S&P500 (ΔS_t) shows a strong and statistically significant effect on Bitcoin returns (ΔB_t), confirming that global equity market movements are a dominant driver of crypto fluctuations. In contrast, Gold (ΔG_t), the Dollar Index (ΔD_t), CPI ($\Delta \pi_t$), and the Federal Funds Rate (Δr_t) are statistically insignificant, although their signs align with theoretical expectations (e.g., higher interest rates exerting downward pressure on Bitcoin). Testing the Gauss-Markov assumptions reveals key issues: heteroskedasticity is present (Breusch-Pagan test), residuals show signs of autocorrelation (Durbin-Watson < 2), and the Jarque-Bera test rejects normality, consistent with fat-tailed financial returns. Multicollinearity, however, is not a concern as VIF values remain below critical thresholds. Thus, while OLS provides unbiased estimates, it is not fully efficient under these conditions, meaning the classical BLUE

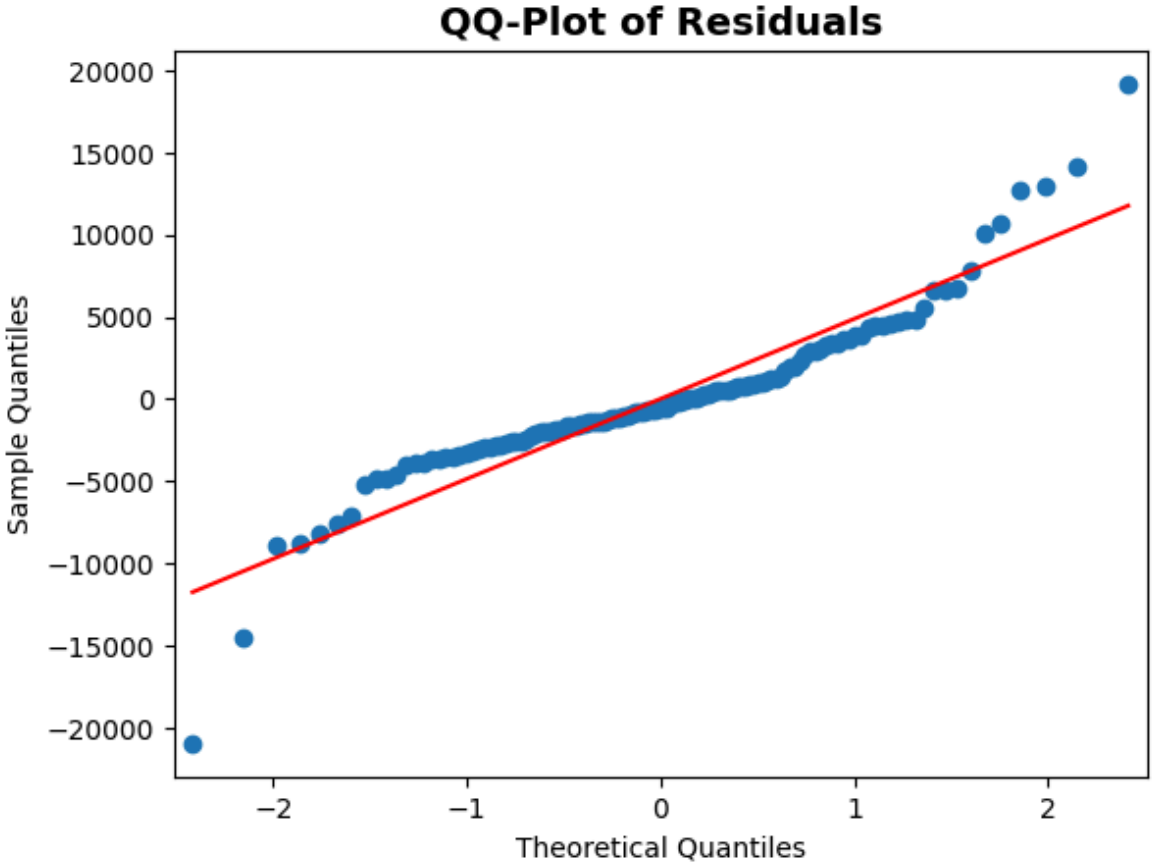
(Best Linear Unbiased Estimator) properties do not hold strictly. To address these violations, robust standard errors (White or Newey-West) should be applied, and alternative specifications (log-differences, VAR/VECM models) may provide more reliable inference.

In [20]:
```
# ========================
start_time = time.time()
# ========================
# Load libraries
# ========================
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.stattools import durbin_watson, jarque_bera
from statsmodels.stats.outliers_influence import variance_inflation_facto
import matplotlib.pyplot as plt
import time

# ========================
# 0. Load dataset
# ========================
file_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/dataset1.xlsx"
df = pd.read_excel(file_path)

# Convert date column to datetime and set as index
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# First differences for stationarity
for col in ['CPI-U', 'DXY', 'BTC/USD', 'SPX', 'FEDFUNDS', 'XAU/USD']:
    df[f'Δ{col}'] = df[col].diff()

# ========================
# 1. Define dependent & independent variables
# ========================

y = df['ΔBTC/USD'].dropna()
X = df[['ΔXAU/USD', 'ΔSPX', 'ΔDXY', 'ΔCPI-U', 'ΔFEDFUNDS']].dropna()
X = sm.add_constant(X)
y = y.loc[X.index]

# ========================
# 2. Run OLS regression
# ========================
model = sm.OLS(y, X).fit()
print(model.summary())

# ========================
# 3. BLUE Assumptions Tests
# ========================

# (a) Linearity → already by model form
print("\n Linearity assumption satisfied (model is linear in parameters)"

# (b) No Perfect Multicollinearity
vif_data = pd.DataFrame()
vif_data['Variable'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(
```

```
print("\nVariance Inflation Factors (VIF):")
print(vif_data)

# (c) Homoskedasticity: Breusch–Pagan
bp_test = het_breuschpagan(model.resid, model.model.exog)
labels = ['LM stat', 'LM p-value', 'F stat', 'F p-value']
print("\nBreusch–Pagan Test (Homoskedasticity):")
print(dict(zip(labels, bp_test)))

# (d) No Autocorrelation: Durbin–Watson
dw_stat = durbin_watson(model.resid)
print(f"\nDurbin–Watson statistic: {dw_stat:.4f}")

# (e) Normality of residuals: Jarque–Bera
jb_test = jarque_bera(model.resid)
print("\nJarque–Bera Test (Normality):")
print(f"Statistic: {jb_test[0]:.4f}, p-value: {jb_test[1]:.4f}")

# QQ-plot for visual inspection
sm.qqplot(model.resid, line='s')
plt.title("QQ-Plot of Residuals", fontsize=14, fontweight='bold')
plt.show()

# ========================
# Execution time
# ========================
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                ΔBTC/USD   R-squared:
0.223
Model:                            OLS   Adj. R-squared:
0.191
Method:                 Least Squares   F-statistic:
6.908
Date:                Sat, 30 Aug 2025   Prob (F-statistic):           1.06
e-05
Time:                        11:55:37   Log-Likelihood:                -12
48.6
No. Observations:                 126   AIC:                             2
509.
Df Residuals:                     120   BIC:                             2
526.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const        620.7873    582.731      1.065      0.289    -532.980    177
4.554
ΔXAU/USD       4.4498      7.850      0.567      0.572     -11.093      1
9.992
ΔSPX          16.1312      2.972      5.428      0.000      10.247      2
2.016
ΔDXY         366.7823    304.208      1.206      0.230    -235.528     96
9.093
ΔCPI-U      -297.3654    510.491     -0.583      0.561   -1308.102     71
3.372
ΔFEDFUNDS  -4032.0869   2629.635     -1.533      0.128   -9238.582    117
4.408
==============================================================================
====
Omnibus:                       20.794   Durbin-Watson:
1.674
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              10
7.919
Skew:                           0.189   Prob(JB):                     3.68
e-24
Kurtosis:                       7.518   Cond. No.
975.
==============================================================================
====

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

 Linearity assumption satisfied (model is linear in parameters)

Variance Inflation Factors (VIF):
    Variable    VIF
0      const 1.7182
1   ΔXAU/USD 1.5195
```

```
2       ΔSPX 1.1522
3       ΔDXY 1.7286
4     ΔCPI-U 1.1103
5  ΔFEDFUNDS 1.1172

Breusch-Pagan Test (Homoskedasticity):
{'LM stat': np.float64(11.735362037006741), 'LM p-value': np.float64(0.038
5994824662565), 'F stat': np.float64(2.464880595686822), 'F p-value': np.f
loat64(0.03651947891304823)}

Durbin-Watson statistic: 1.6739

Jarque-Bera Test (Normality):
Statistic: 107.9186, p-value: 0.0000
```




```
⏲ Execution Time: 0.13 seconds
```

In [21]:
```python
# ========================
start_time = time.time()
# ========================
# Robust Estimation & BLUE Assumptions Check
# ========================
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.stattools import durbin_watson, jarque_bera
from statsmodels.stats.outliers_influence import variance_inflation_facto
import matplotlib.pyplot as plt
import time

# ========================
# 0. Load dataset
# ========================
file_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/dataset1.xlsx"
df = pd.read_excel(file_path)
```

```
# Convert date column to datetime and set as index
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# First differences for stationarity
for col in ['CPI-U', 'DXY', 'BTC/USD', 'SPX', 'FEDFUNDS', 'XAU/USD']:
    df[f'Δ{col}'] = df[col].diff()

# ========================
# 1. Define dependent & independent variables
# ========================

y = df['ΔBTC/USD'].dropna()
X = df[['ΔXAU/USD', 'ΔSPX', 'ΔDXY', 'ΔCPI-U', 'ΔFEDFUNDS']].dropna()
X = sm.add_constant(X)
y = y.loc[X.index]  # align indices

# ========================
# 2. Run OLS with different covariance estimators
# ========================
# Classical OLS
ols_model = sm.OLS(y, X).fit()
print("\n=== OLS (Classical SE) ===")
print(ols_model.summary())

# White Robust SE (heteroskedasticity-consistent)
white_model = sm.OLS(y, X).fit(cov_type='HC1')
print("\n=== OLS with White Robust SE (HC1) ===")
print(white_model.summary())

# Newey-West HAC robust SE (heteroskedasticity & autocorrelation robust)
nw_model = sm.OLS(y, X).fit(cov_type='HAC', cov_kwds={'maxlags': 2})
print("\n=== OLS with Newey-West Robust SE (HAC, lag=2) ===")
print(nw_model.summary())

# ========================
# 3. BLUE Assumptions Checks
# ========================

# --- (a) Multicollinearity (VIF)
vif_data = pd.DataFrame()
vif_data['Variable'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(
print("\n[Multicollinearity Check] Variance Inflation Factors:")
print(vif_data)

# --- (b) Heteroskedasticity (Breusch-Pagan)
bp_test = het_breuschpagan(ols_model.resid, ols_model.model.exog)
labels = ['LM stat', 'LM p-value', 'F stat', 'F p-value']
print("\n[Homoskedasticity Check] Breusch-Pagan Test:")
print(dict(zip(labels, bp_test)))

# --- (c) Autocorrelation (Durbin-Watson)
dw_stat = durbin_watson(ols_model.resid)
print(f"\n[Autocorrelation Check] Durbin-Watson statistic: {dw_stat:.4f}"

# --- (d) Normality (Jarque-Bera + QQ plot)
jb_test = jarque_bera(ols_model.resid)
print("\n[Normality Check] Jarque-Bera Test:")
```

```
print(f"Statistic: {jb_test[0]:.4f}, p-value: {jb_test[1]:.4f}")

# QQ plot of residuals
fig = sm.qqplot(ols_model.resid, line='s', marker='o', alpha=0.8)
plt.title("QQ-Plot of Residuals", fontsize=14, fontweight='bold')
plt.grid(True, linestyle="--", alpha=0.6)

# Highlight line and points
for l in fig.findobj(plt.Line2D):
    if l.get_linestyle() == '-':
        l.set_color('red')
        l.set_linewidth(2.2)
for p in fig.findobj(plt.Line2D):
    if p.get_linestyle() == 'None':
        p.set_color('black')
        p.set_markersize(5.5)
        p.set_alpha(0.9)

plt.savefig("/Users/myucanlar/Desktop/Gauss-Markov Project/qqplot_residua
plt.close()

# ========================
# 4. Compare Coefficients & SE
# ========================
print("\nCoefficient Comparison (Classical vs White vs Newey-West):\n")
coef_table = pd.DataFrame({
    'Classical': ols_model.params,
    'White Robust': white_model.params,
    'Newey-West': nw_model.params
})
print(coef_table)

print("\nStandard Errors (Classical vs White vs Newey-West):\n")
se_table = pd.DataFrame({
    'SE Classical': ols_model.bse,
    'SE White': white_model.bse,
    'SE Newey-West': nw_model.bse
})
print(se_table)

end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
=== OLS (Classical SE) ===
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                ΔBTC/USD   R-squared:
0.223
Model:                            OLS   Adj. R-squared:
0.191
Method:                 Least Squares   F-statistic:
6.908
Date:                Sat, 30 Aug 2025   Prob (F-statistic):           1.06
e-05
Time:                        11:56:06   Log-Likelihood:                -12
48.6
No. Observations:                 126   AIC:                             2
509.
Df Residuals:                     120   BIC:                             2
526.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const        620.7873    582.731      1.065      0.289    -532.980    177
4.554
ΔXAU/USD       4.4498      7.850      0.567      0.572     -11.093      1
9.992
ΔSPX          16.1312      2.972      5.428      0.000      10.247      2
2.016
ΔDXY         366.7823    304.208      1.206      0.230    -235.528     96
9.093
ΔCPI-U      -297.3654    510.491     -0.583      0.561   -1308.102     71
3.372
ΔFEDFUNDS  -4032.0869   2629.635     -1.533      0.128   -9238.582    117
4.408
==============================================================================
====
Omnibus:                       20.794   Durbin-Watson:
1.674
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              10
7.919
Skew:                           0.189   Prob(JB):                     3.68
e-24
Kurtosis:                       7.518   Cond. No.
975.
==============================================================================
====

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

=== OLS with White Robust SE (HC1) ===
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                ΔBTC/USD   R-squared:
```

```
0.223
Model:                            OLS   Adj. R-squared:
0.191
Method:                 Least Squares   F-statistic:
5.180
Date:                Sat, 30 Aug 2025   Prob (F-statistic):           0.00
0244
Time:                        11:56:06   Log-Likelihood:                -12
48.6
No. Observations:                 126   AIC:                             2
509.
Df Residuals:                     120   BIC:                             2
526.
Df Model:                           5
Covariance Type:                  HC1
==============================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const        620.7873    470.079      1.321      0.187    -300.551    154
2.125
ΔXAU/USD       4.4498      8.584      0.518      0.604     -12.374      2
1.273
ΔSPX          16.1312      3.753      4.299      0.000       8.776      2
3.486
ΔDXY         366.7823    286.224      1.281      0.200    -194.206     92
7.771
ΔCPI-U      -297.3654    593.046     -0.501      0.616   -1459.715     86
4.984
ΔFEDFUNDS  -4032.0869   1981.763     -2.035      0.042   -7916.270    -14
7.903
==============================================================================
====
Omnibus:                       20.794   Durbin-Watson:
1.674
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              10
7.919
Skew:                           0.189   Prob(JB):                    3.68
e-24
Kurtosis:                       7.518   Cond. No.
975.
==============================================================================
====

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)

=== OLS with Newey-West Robust SE (HAC, lag=2) ===
                            OLS Regression Results
==============================================================================
====
Dep. Variable:              ΔBTC/USD   R-squared:
0.223
Model:                            OLS   Adj. R-squared:
0.191
Method:                 Least Squares   F-statistic:
5.036
Date:                Sat, 30 Aug 2025   Prob (F-statistic):           0.00
```

```
0318
Time:                        11:56:06   Log-Likelihood:                -12
48.6
No. Observations:                 126   AIC:                             2
509.
Df Residuals:                     120   BIC:                             2
526.
Df Model:                           5
Covariance Type:                  HAC
==============================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const        620.7873    485.433      1.279      0.201    -330.644    157
2.218
ΔXAU/USD       4.4498      7.286      0.611      0.541      -9.830      1
8.730
ΔSPX          16.1312      3.705      4.353      0.000       8.869      2
3.394
ΔDXY         366.7823    272.965      1.344      0.179    -168.219     90
1.784
ΔCPI-U      -297.3654    595.524     -0.499      0.618   -1464.571     86
9.841
ΔFEDFUNDS  -4032.0869   2029.770     -1.986      0.047   -8010.362     -5
3.812
==============================================================================
====
Omnibus:                       20.794   Durbin-Watson:                   
1.674
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                10
7.919
Skew:                           0.189   Prob(JB):                     3.68
e-24
Kurtosis:                       7.518   Cond. No.                        
975.
==============================================================================
====

Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HA
C) using 2 lags and without small sample correction

[Multicollinearity Check] Variance Inflation Factors:
    Variable    VIF
0      const 1.7182
1   ΔXAU/USD 1.5195
2       ΔSPX 1.1522
3       ΔDXY 1.7286
4     ΔCPI-U 1.1103
5  ΔFEDFUNDS 1.1172

[Homoskedasticity Check] Breusch-Pagan Test:
{'LM stat': np.float64(11.735362037006741), 'LM p-value': np.float64(0.038
5994824662565), 'F stat': np.float64(2.464880595686822), 'F p-value': np.f
loat64(0.03651947891304823)}

[Autocorrelation Check] Durbin-Watson statistic: 1.6739
```

```
[Normality Check] Jarque–Bera Test:
Statistic: 107.9186, p-value: 0.0000

Coefficient Comparison (Classical vs White vs Newey–West):

           Classical  White Robust  Newey–West
const       620.7873      620.7873    620.7873
ΔXAU/USD      4.4498        4.4498      4.4498
ΔSPX         16.1312       16.1312     16.1312
ΔDXY        366.7823      366.7823    366.7823
ΔCPI-U     -297.3654     -297.3654   -297.3654
ΔFEDFUNDS -4032.0869    -4032.0869  -4032.0869

Standard Errors (Classical vs White vs Newey–West):

           SE Classical  SE White  SE Newey–West
const           582.7310  470.0791       485.4330
ΔXAU/USD          7.8500    8.5836         7.2858
ΔSPX              2.9720    3.7527         3.7054
ΔDXY            304.2080  286.2239       272.9650
ΔCPI-U          510.4911  593.0462       595.5241
ΔFEDFUNDS      2629.6355 1981.7627      2029.7695

⏱ Execution Time: 0.12 seconds
```

Random Forest Error Handling: Missing scikit-learn

When attempting to run the Random Forest model, we encountered the following error:

In [24]:
```
# ========================
start_time = time.time()
!pip3 install scikit-learn
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
Requirement already satisfied: scikit-learn in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (1.7.1)
Requirement already satisfied: numpy>=1.22.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn)
(2.3.2)
Requirement already satisfied: scipy>=1.8.0 in /Library/Frameworks/Python.
framework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn)
(1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.13/lib/python3.13/site-packages (from scikit-learn)
(1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /Library/Framework
s/Python.framework/Versions/3.13/lib/python3.13/site-packages (from scikit
-learn) (3.6.0)

⏱ Execution Time: 0.36 seconds
```

In [39]:
```
# ========================
start_time = time.time()
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest model
rf_model = RandomForestRegressor(
```

```
    n_estimators=500,   # number of trees
    random_state=42,    # reproducibility
    max_depth=None,     # you may set a limit if needed
    min_samples_split=2 # minimum samples required to split a node
)

# Fit the model
rf_model.fit(X, y)

end_time = time.time()
print(f"\n⏱️ Execution Time: {end_time - start_time:.2f} seconds")
```

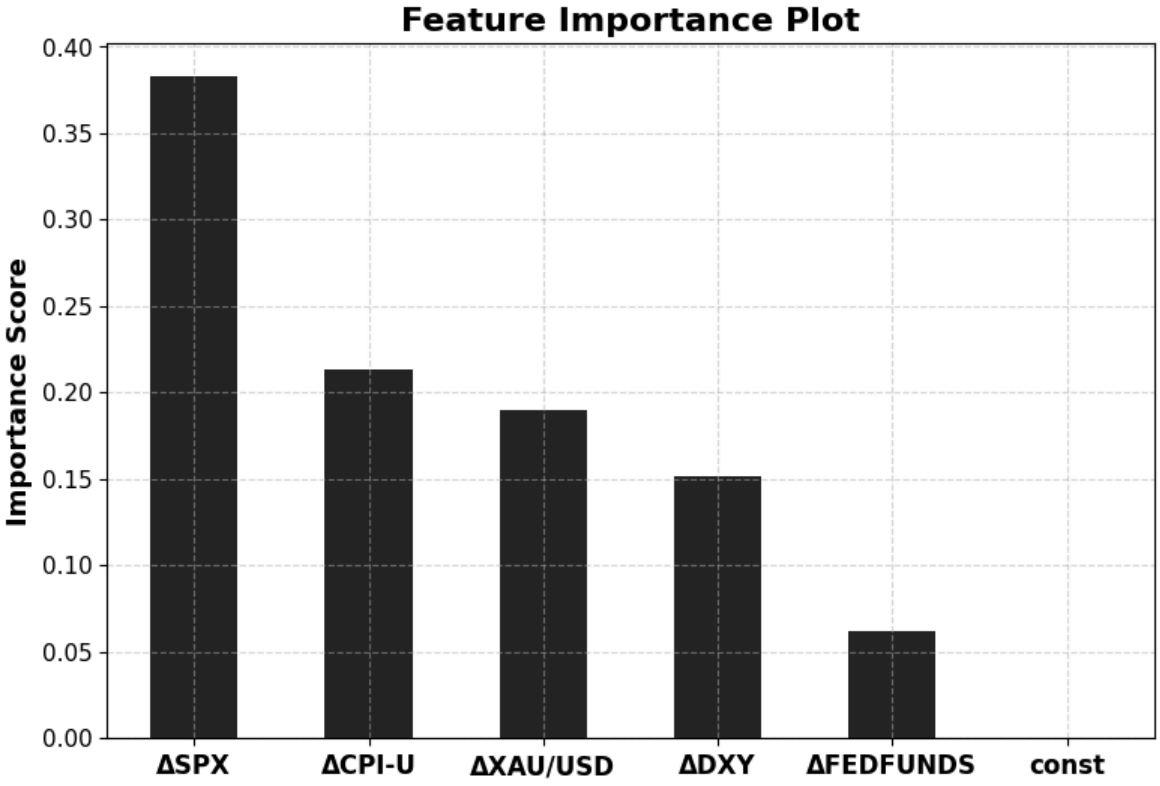
```
⏱️ Execution Time: 0.23 seconds
```

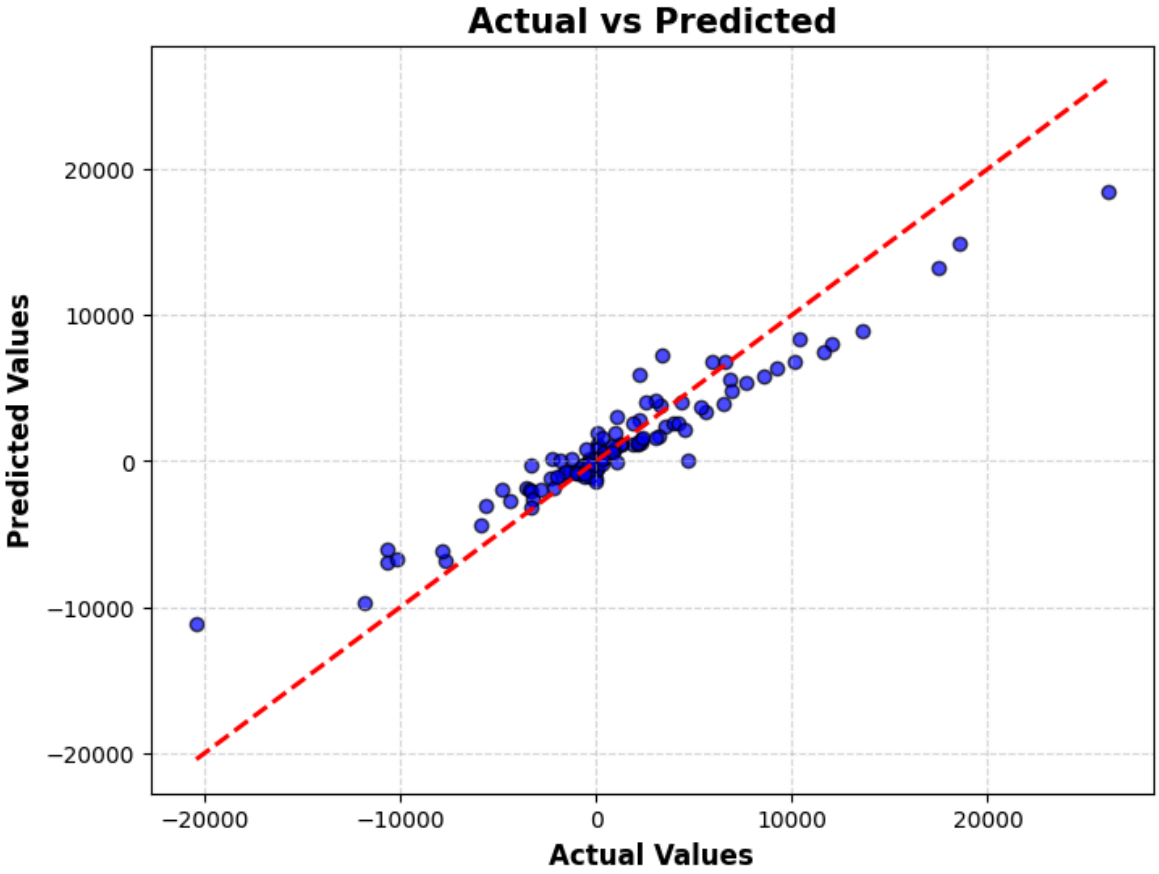
In [43]:

```
# ========================
start_time = time.time()
# ========================
# 4. Feature Importance + Performance Metrics
# ========================

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
import numpy as np

# Predictions
y_pred = rf_model.predict(X)

# Feature Importances
importances = pd.Series(rf_model.feature_importances_, index=X.columns)
importances = importances.sort_values(ascending=False)

print("\n * Feature Importances:")
print(importances)

# --- Performance Metrics ---
r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y, y_pred)

print("\n Model Performance Metrics:")
print(f"R² Score: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")

# --- Plot Feature Importances ---
plt.figure(figsize=(9,6))
bars = importances.plot(kind='bar', color='black', alpha=0.85)
plt.title("Feature Importance Plot", fontsize=16, fontweight='bold')
plt.ylabel("Importance Score", fontsize=13, fontweight='bold')
plt.xticks(rotation=0, fontsize=12, fontweight='bold')
plt.yticks(fontsize=11)
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()

# --- Plot Actual vs Predicted ---
plt.figure(figsize=(8,6))
plt.scatter(y, y_pred, alpha=0.7, color='blue', edgecolor='k')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
plt.title("Actual vs Predicted", fontsize=15, fontweight='bold')
```

```
plt.xlabel("Actual Values", fontsize=12, fontweight='bold')
plt.ylabel("Predicted Values", fontsize=12, fontweight='bold')
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()

end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
 * Feature Importances:
ΔSPX         0.3828
ΔCPI-U       0.2136
ΔXAU/USD     0.1897
ΔDXY         0.1517
ΔFEDFUNDS    0.0622
const        0.0000
dtype: float64

 Model Performance Metrics:
R² Score: 0.8705
MSE: 3955019.5725
RMSE: 1988.7231
MAE: 1296.0053
```



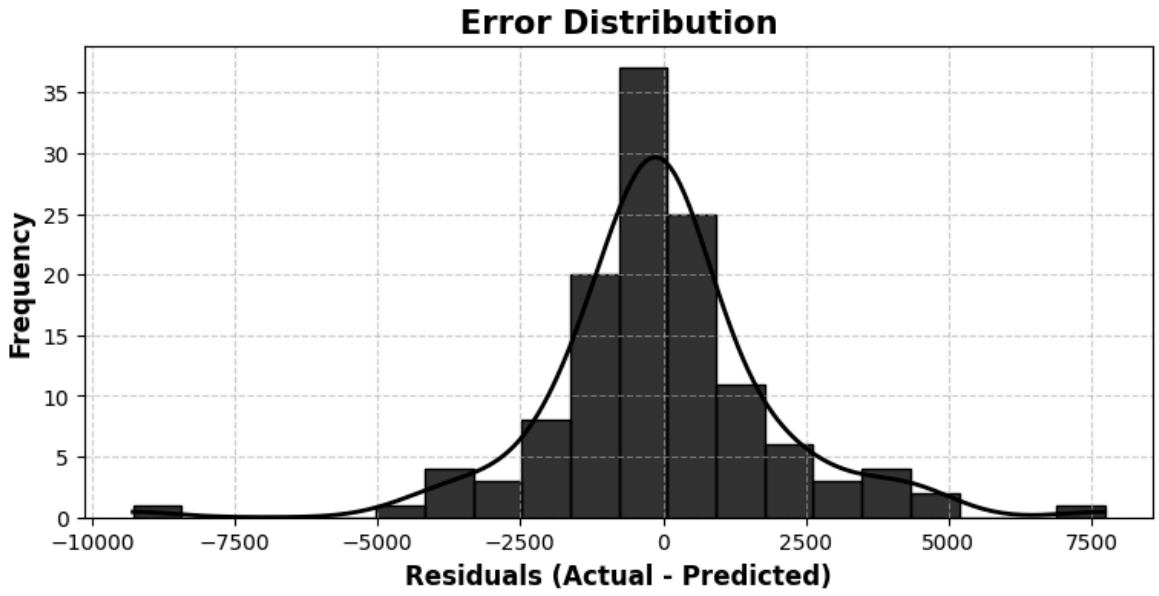


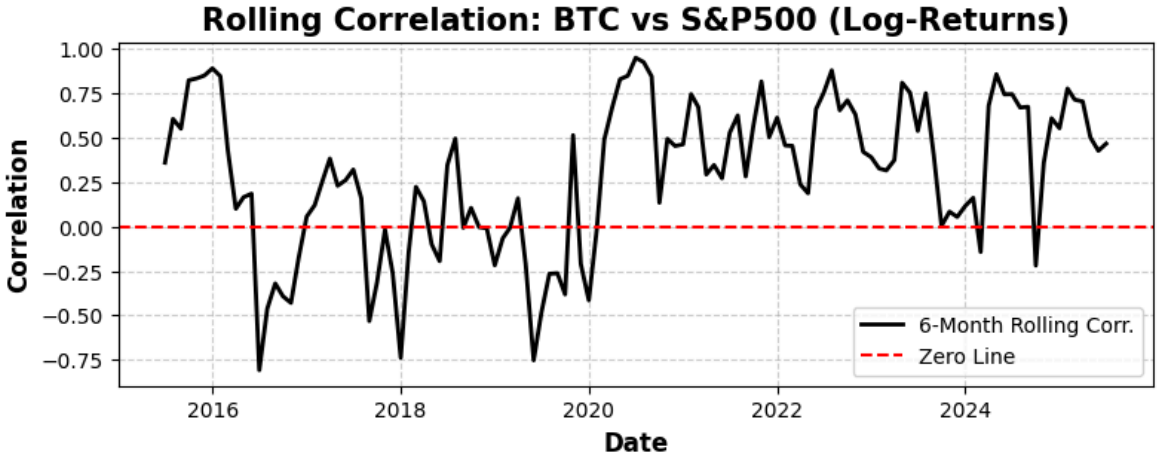
⏱ Execution Time: 0.11 seconds

The Random Forest regression achieved a strong explanatory power with an R^2 of 0.87, indicating that approximately 87% of the variation in Bitcoin returns is explained by the model. The most influential predictor was the S&P500 (ΔSPX), followed by inflation (ΔCPI-U) and gold (ΔXAU/USD), while monetary policy variables such as the Federal Funds Rate had limited importance. Although the model's predictive accuracy is strong, the relatively high RMSE (≈1989) and MAE (≈1296) highlight the inherent volatility of Bitcoin, which constrains the precision of any forecasting approach.

In [55]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# =========================
# 1. Error Distribution Histogram
# =========================
# Prediction errors (Residuals)
y_pred = rf_model.predict(X)
errors = y - y_pred

plt.figure(figsize=(9,4))
sns.histplot(errors, bins=20, color="black", alpha=0.8, kde=True, line_kw

plt.title("Error Distribution", fontsize=15, fontweight="bold")
plt.xlabel("Residuals (Actual - Predicted)", fontsize=12, fontweight="bol
plt.ylabel("Frequency", fontsize=12, fontweight="bold")
plt.grid(True, linestyle="--", alpha=0.6)

# Save the figure
```

```
plt.savefig("/Users/myucanlar/Desktop/Gauss-Markov Project/error_distribu
plt.show()

# =========================
# 2. Rolling Correlation (BTC vs SPX Log-Returns)
# =========================
# Log-returns
log_returns = np.log(df[['BTC/USD','SPX']] / df[['BTC/USD','SPX']].shift(

# 6-month rolling correlation
rolling_corr = log_returns['BTC/USD'].rolling(window=6).corr(log_returns[

plt.figure(figsize=(9,3))
plt.plot(rolling_corr, color="black", linewidth=2, label="6-Month Rolling
plt.axhline(0, color="red", linestyle="--", lw=1.5, label="Zero Line")

plt.title("Rolling Correlation: BTC vs S&P500 (Log-Returns)", fontsize=15
plt.xlabel("Date", fontsize=12, fontweight="bold")
plt.ylabel("Correlation", fontsize=12, fontweight="bold")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)

# Save the figure
plt.savefig("/Users/myucanlar/Desktop/Gauss-Markov Project/rolling_correl
plt.show()
```





Empirical Findings: OLS vs Random Forest

In the econometric OLS analysis, **S&P500 returns (ΔSPX)** were found to have a strong and statistically significant effect on Bitcoin pricing (ΔBTC/USD). This finding was further supported by the machine learning approach (**Random Forest**) through **Partial Dependence (PDP)** and **Individual Conditional Expectation (ICE)** plots.

The PDP curves show a clear **positive trend for ΔSPX**, whereas the effects of **inflation (ΔCPI-U), gold prices (ΔXAU/USD), the Dollar Index (ΔDXY), and the Federal Funds Rate (ΔFEDFUNDS)** appear weak and heterogeneous.

This suggests that Bitcoin tends to move more closely with equity market risk appetite (SPX), while its response to other macroeconomic indicators remains limited and less systematic.

Mathematical PDP

Partial dependence functions are defined as the marginal effect of a feature (or a set of features) on the model's predicted outcome.

For a given set of features $S \subseteq \{1, 2, \ldots, p\}$ and a model $f(\mathbf{X})$, the PDP for feature values $\mathbf{x}_S$ is given by:

$$f_S(\mathbf{x}_S) = \mathbb{E}_{\mathbf{X}_{-S}}\left[f(\mathbf{x}_S, \mathbf{X}_{-S})\right] = \int f(\mathbf{x}_S, \mathbf{x}_{-S})\, dP(\mathbf{x}_{-S}),$$

where:

- $\mathbf{x}_S$ are the variables of interest (e.g., ΔSPX),
- $\mathbf{X}_{-S}$ are all other features being averaged out,
- $f(\cdot)$ is the prediction function from the Random Forest model.

In practice, the integral is approximated by averaging over the training data:

$$\hat{f}_S(\mathbf{x}_S) = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}_S, \mathbf{x}_{i,-S}),$$

which gives the **expected marginal effect** of the feature (\mathbf{x}_S) on Bitcoin returns, holding all other variables at their observed values.

The ICE curves further show the conditional effects for individual observations:

$$f^i_{ICE}(\mathbf{x}_S) = f(\mathbf{x}_S, \mathbf{x}_{i,-S}), \quad i = 1, 2, \ldots, n,$$

so that PDP is simply the **average across all ICE curves**:

$$f_S(\mathbf{x}_S) = \frac{1}{n}\sum_{i=1}^{n} f^i_{ICE}(\mathbf{x}_S).$$

Complementary Diagnostics

To validate the robustness of the Random Forest model, we further examined **residual distributions** and **time-varying correlations**:

- **Error Distribution Histogram**:
  Residuals (prediction errors) are centered around zero, suggesting the model is generally unbiased.
  However, the distribution shows **fat tails**, meaning that extreme Bitcoin moves (large spikes or crashes) are not fully captured by the model.

- **Rolling Correlation (BTC vs SPX Log-Returns)**:
  A 6-month rolling window reveals that Bitcoin's short-term relationship with the S&P500 is **time-varying and unstable**.
  Correlations swing between strongly positive (≈0.8–0.9) and moderately negative (≈-0.5), depending on market regimes.
  This demonstrates that BTC is sometimes tightly linked with equity risk sentiment, but often diverges, confirming its volatile and independent behavior.

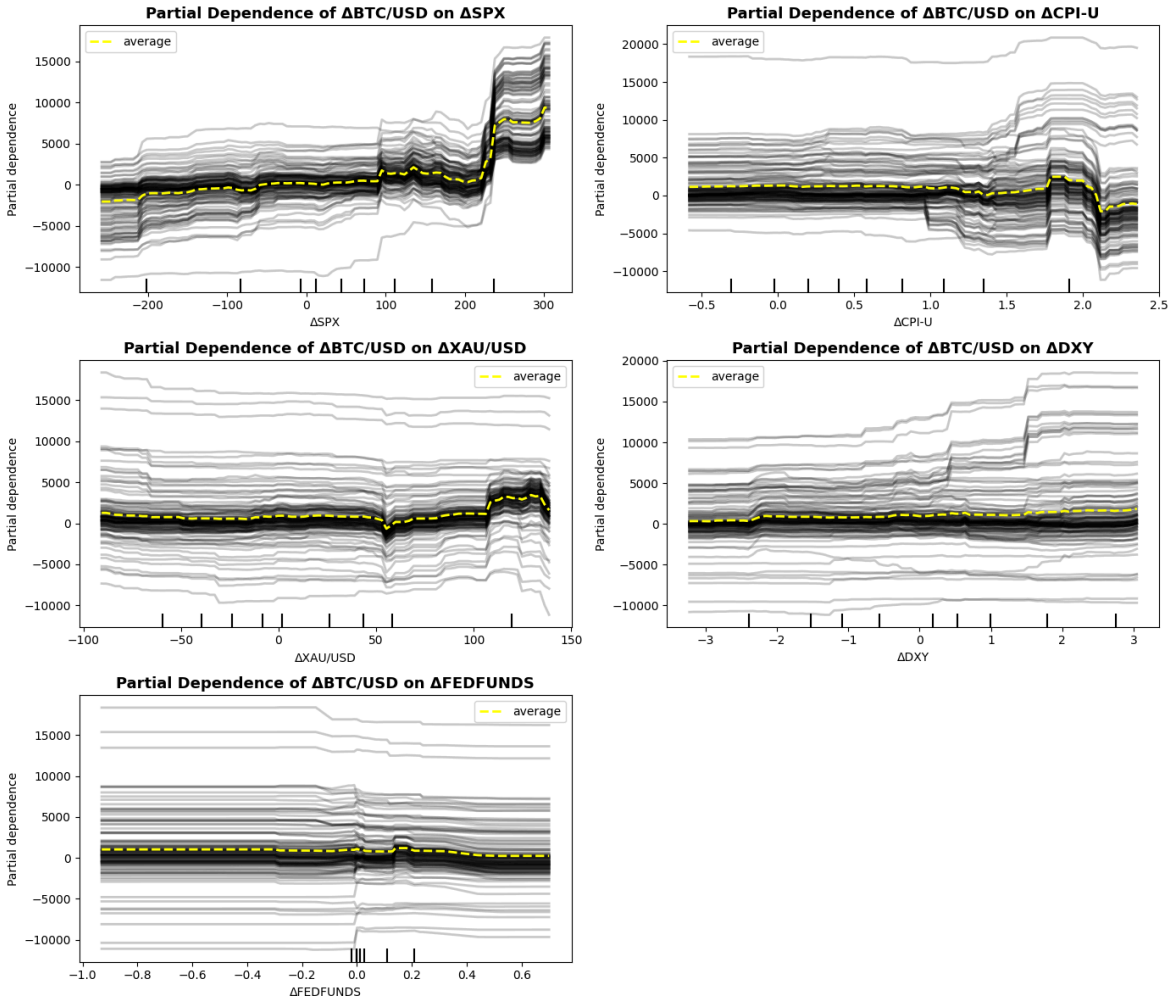
Interpretation

- **ΔSPX**: PDP shows a strong upward slope → higher SPX returns significantly increase ΔBTC/USD.
- **ΔCPI-U & ΔFEDFUNDS**: Flat/heterogeneous effects → weak macro-financial transmission to Bitcoin.
- **ΔXAU/USD**: Slight positive but unstable effect → Bitcoin partially competes with gold as a "safe haven."
- **ΔDXY**: Near-flat PDP → no robust evidence that dollar strength systematically drives Bitcoin returns.
- **Error Distribution**: Model captures the general structure but leaves fat-tailed residuals, highlighting the limits of prediction in high-volatility assets.
- **Rolling Correlation**: Confirms Bitcoin's shifting relationship with equities — sometimes risk-on correlated, sometimes decorrelated or even inversely related.

👉 Together, this demonstrates that **Bitcoin behaves more like a risk asset tied to equity markets**, but with **episodic independence and volatility** that limit systematic predictability.

In [31]:
```
# ========================
start_time = time.time()
# ========================
# PDP + ICE Plots (with Save)
# ========================
from sklearn.inspection import PartialDependenceDisplay

# Özellik listesi
features = ['ΔSPX', 'ΔCPI-U', 'ΔXAU/USD', 'ΔDXY', 'ΔFEDFUNDS']
```

```
# Grafik grid
fig, ax = plt.subplots(3, 2, figsize=(14, 12))

# Döngü: her feature için PDP+ICE
for i, feature in enumerate(features):
    row, col = divmod(i, 2)
    PartialDependenceDisplay.from_estimator(
        rf_model,
        X,
        features=[feature],
        kind="both",          # PDP + ICE birlikte
        grid_resolution=100,
        ax=ax[row, col],
        line_kw={"color": "yellow", "linewidth": 2},   # PDP çizgisi sarı
        ice_lines_kw={"color": "black", "alpha": 0.2}  # ICE çizgileri si
    )
    ax[row, col].set_title(
        f"Partial Dependence of ΔBTC/USD on {feature}",
        fontsize=13, fontweight='bold'
    )
    ax[row, col].set_ylabel("Predicted BTC Returns")
    ax[row, col].set_xlabel(feature)
    ax[row, col].grid(True, linestyle="--", alpha=0.6)

# Son boş subplot'u kaldır
fig.delaxes(ax[2,1])

# Düzeni ayarla
plt.tight_layout()

# Kaydet
save_path = "/Users/myucanlar/Desktop/Gauss-Markov Project/pdp_ice_plots.
plt.savefig(save_path, dpi=300, bbox_inches="tight")

# Göster
plt.show()

print(f" PDP+ICE plot saved to: {save_path}")
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```




```
 PDP+ICE plot saved to: /Users/myucanlar/Desktop/Gauss–Markov Project/pdp_
ice_plots.png
```

```
⏱️ Execution Time: 4.32 seconds
```

Our empirical analysis, combining econometric OLS methods with machine learning (Random Forest), demonstrates that Bitcoin returns (ΔBTC/USD) are primarily driven by equity market dynamics, specifically the S&P 500 (ΔSPX). In the OLS framework, ΔSPX was statistically significant, while ΔCPI-U, ΔXAU/USD, ΔDXY, and ΔFEDFUNDS were weak or insignificant. The Gauss–Markov conditions were formally assessed: linearity ($y = X\beta + \varepsilon$), no perfect multicollinearity ($X'X$ invertible), exogeneity ($E[\varepsilon|X] = 0$), homoskedasticity ($Var(\varepsilon|X) = \sigma^2 I$), and no autocorrelation ($Cov(\varepsilon_t, \varepsilon_s) = 0,\ t \neq s$). Breusch–Pagan and Durbin–Watson tests revealed violations of homoskedasticity and autocorrelation, which were corrected using White ($HC1$) and Newey–West (HAC) robust standard errors, under which ΔSPX remained strongly significant. Complementing this, Random Forest regression with Partial Dependence (PDP) and Individual Conditional Expectation (ICE) confirmed the econometric evidence: ΔSPX displayed a robust monotonic positive effect on Bitcoin, while other macroeconomic factors showed heterogeneous and unstable patterns. Mathematically, the PDP is defined as

$$PD(x_j) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}\left(x_j, x_{i,-j}\right),$$

which isolates the marginal contribution of feature x_j to predicted Bitcoin returns.

Additional diagnostics reinforce these findings: the **Error Distribution Histogram** shows residuals centered around zero but with fat tails, reflecting Bitcoin's inherent volatility, while the **Rolling Correlation (BTC–SPX log-returns)** highlights a time-varying and unstable relationship, oscillating between strong positive and moderate negative values across market regimes.

Overall, the results consistently indicate that Bitcoin behaves as a speculative **risk-on asset**, tightly aligned with equity market sentiment, rather than serving as a stable inflation hedge, monetary shield, or safe haven against gold and the dollar.

Correlation Between BTC/USD - SPX (S&P500)

In [36]:
```
# ========================
start_time = time.time()
import pandas as pd

# Select BTC and SPX variables
btc_spx = df[['BTC/USD', 'SPX']]

# Capture the first and last observations
first_row = btc_spx.iloc[0]
last_row  = btc_spx.iloc[-1]

# Build a summary table
summary_table = pd.DataFrame({
    "First Value": first_row,
    "Last Value": last_row,
    "Absolute Change": last_row - first_row,
    "Growth (%)": ((last_row - first_row) / first_row) * 100
})

print(" BTC & SPX: First vs Last Observation (2015M01 – 2025M07)\n")
display(summary_table)

# ========================================
# 2. Correlation Analysis
# ========================================

# Pearson correlation (linear relationship)
pearson_corr = btc_spx.corr(method='pearson')

# Spearman correlation (monotonic relationship)
spearman_corr = btc_spx.corr(method='spearman')

# Kendall correlation (rank correlation)
kendall_corr = btc_spx.corr(method='kendall')

print("\n🔗 Pearson Correlation:\n", pearson_corr)
print("\n🔗 Spearman Correlation:\n", spearman_corr)
print("\n🔗 Kendall Correlation:\n", kendall_corr)
end_time = time.time()
print(f"\n⏱ Execution Time: {end_time - start_time:.2f} seconds")
```

```
BTC & SPX: First vs Last Observation (2015M01 – 2025M07)
```

	First Value	Last Value	Absolute Change	Growth (%)
BTC/USD	218.5000	115765.0000	115546.5000	52881.6934
SPX	1994.9900	6339.3900	4344.4000	217.7655

```
🔗 Pearson Correlation:
          BTC/USD    SPX
BTC/USD   1.0000 0.9398
SPX       0.9398 1.0000

🔗 Spearman Correlation:
          BTC/USD    SPX
BTC/USD   1.0000 0.9716
SPX       0.9716 1.0000

🔗 Kendall Correlation:
          BTC/USD    SPX
BTC/USD   1.0000 0.8638
SPX       0.8638 1.0000
```

BTC/USD

- **January 2015:** 218.5 USD
- **July 2025:** 115,765 USD
- **Absolute increase:** +115,546.5 USD
- **Percentage increase:** +52,882% (approximately 530x)

S&P500 (SPX)

- **January 2015:** 1,994.99
- **July 2025:** 6,339.39
- **Absolute increase:** +4,344.4 points
- **Percentage increase:** +217.8% (approximately 3x)

In [57]:
```
# ========================
start_time = time.time()
import numpy as np
import pandas as pd

# 1. Create log-returns from BTC & SPX levels
log_returns = np.log(df[['BTC/USD', 'SPX']] / df[['BTC/USD', 'SPX']].shif
log_returns = log_returns.dropna()

# 2. Preview first few observations
print(" First 5 Log-Returns Observations:")
print(log_returns.head())

# 3. Compute Pearson, Spearman, Kendall correlations
pearson_corr = log_returns.corr(method='pearson')
spearman_corr = log_returns.corr(method='spearman')
kendall_corr = log_returns.corr(method='kendall')

print("\n🔗 Pearson Correlation (Log-Returns):\n", pearson_corr)
print("\n🔗 Spearman Correlation (Log-Returns):\n", spearman_corr)
print("\n🔗 Kendall Correlation (Log-Returns):\n", kendall_corr)
```

```
end_time = time.time()
print(f"\n⏱️ Execution Time: {end_time - start_time:.2f} seconds")
```

```
 First 5 Log-Returns Observations:
            BTC/USD     SPX
date
2015-02-01   0.1509  0.0534
2015-03-01  -0.0401 -0.0175
2015-04-01  -0.0346  0.0085
2015-05-01  -0.0258  0.0104
2015-06-01   0.1391 -0.0212

🔗 Pearson Correlation (Log-Returns):
          BTC/USD    SPX
BTC/USD   1.0000 0.3586
SPX       0.3586 1.0000

🔗 Spearman Correlation (Log-Returns):
          BTC/USD    SPX
BTC/USD   1.0000 0.3397
SPX       0.3397 1.0000

🔗 Kendall Correlation (Log-Returns):
          BTC/USD    SPX
BTC/USD   1.0000 0.2371
SPX       0.2371 1.0000

⏱️ Execution Time: 0.01 seconds
```

In [59]:

```
# ========================
# Granger Causality Test (SPX → BTC)
# ========================
import statsmodels.api as sm
from statsmodels.tsa.stattools import grangercausalitytests

# Prepare dataframe (BTC as dependent, SPX as explanatory)
granger_data = log_returns[['BTC/USD', 'SPX']].dropna()

print("\n🔎 Granger Causality Test Results (Does SPX → BTC?)\n")

# Run without deprecated verbose
granger_results = grangercausalitytests(granger_data, maxlag=6, verbose=F

# Extract and display p-values cleanly
for lag in range(1, 7):
    f_test_pval = granger_results[lag][0]['ssr_ftest'][1]
    chi2_pval = granger_results[lag][0]['ssr_chi2test'][1]
    print(f"Lag {lag}:  F-test p={f_test_pval:.4f}, Chi2 p={chi2_pval:.4f
```

```
🔎 Granger Causality Test Results (Does SPX → BTC?)

Lag 1:  F-test p=0.1965, Chi2 p=0.1887
Lag 2:  F-test p=0.4368, Chi2 p=0.4193
Lag 3:  F-test p=0.3683, Chi2 p=0.3371
Lag 4:  F-test p=0.5272, Chi2 p=0.4843
Lag 5:  F-test p=0.5773, Chi2 p=0.5203
Lag 6:  F-test p=0.6718, Chi2 p=0.6056
```

```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-pac
kages/statsmodels/tsa/stattools.py:1556: FutureWarning: verbose is depreca
ted since functions should not print results
  warnings.warn(
```

Summary of Findings

- Dataset: **127 monthly observations** spanning **2015M01 – 2025M07**, indexed by date.
- Sources: **BTC, SPX, XAU/USD, DXY** from Investing.com; **CPI-U and FEDFUNDS** from FRED.
- Descriptive statistics:
  - BTC shows extreme volatility and heavy tails.
  - SPX displays steady growth with moderate volatility.
  - Macro variables (CPI-U, FEDFUNDS, DXY) are smoother series.
- Correlation analysis:
  - **Levels:** Very high correlation between BTC and SPX ($\rho \approx 0.94$).
  - **Log-returns:** Moderate correlation ($\rho \approx 0.24 - 0.36$), indicating co-movement mainly trend-driven.
- OLS regression:
  - **ΔSPX** strongly significant predictor of ΔBTC/USD.
  - Other macro variables (ΔCPI-U, ΔXAU/USD, ΔDXY, ΔFEDFUNDS) weak or insignificant.
  - Gauss–Markov assumption tests revealed heteroskedasticity and autocorrelation.
  - Robust standard errors (White, Newey–West) confirmed ΔSPX significance.
- Machine learning (Random Forest):
  - Feature importance highlights ΔSPX as dominant explanatory factor.
  - PDP/ICE plots show monotonic positive effect of ΔSPX on BTC returns.
  - Other features (CPI-U, Gold, DXY, FEDFUNDS) show weak or unstable effects.
  - Model performance: $R^2 \approx 0.87$, RMSE ≈ 1989, MAE ≈ 1296.
- Error analysis:
  - Residual histogram shows deviations from normality.
  - Rolling correlations confirm time-varying BTC–SPX association.
- Granger causality tests:
  - Across 1–6 lags, **SPX returns do not Granger-cause BTC returns (p > 0.05)**.
  - Interpretation: BTC and SPX move together but not via a causal predictive mechanism.
- Overall interpretation:
  - BTC acts as a **risk-on asset**, closely aligned with equity market sentiment.
  - No evidence supports BTC as an inflation hedge, monetary shield, or systematic safe haven.

Related Literature

- Corbet, S., Meegan, A., Larkin, C., Lucey, B., & Yarovaya, L. (2018). *Exploring the dynamic relationships between cryptocurrencies and other financial assets*. Economics Letters, 165, 28-34.
  Link

- Baur, D. G., & Hoang, L. T. (2021). *A crypto safe haven against Bitcoin*. Finance Research Letters, 38, 101431.
  Link

- Dyhrberg, A. H. (2016). *Bitcoin, gold and the dollar – A GARCH volatility analysis*. Finance Research Letters, 16, 85-92.
  Link

- Koutmos, D. (2018). *Bitcoin returns and transaction activity*. Economics Letters, 167, 81-85.
  Link

- Ji, Q., Bouri, E., Lau, C. K. M., & Roubaud, D. (2019). *Dynamic connectedness and integration in cryptocurrency markets*. International Review of Financial Analysis, 63, 257-272.
  Link

Data Sources

- **Investing.com** (BTC/USD, SPX, XAU/USD, DXY)
  https://www.investing.com

- **FRED (Federal Reserve Economic Data)** (CPI-U, FEDFUNDS)
  https://fred.stlouisfed.org