```
#%%
import pandas as pd
#%%
data_start = 0
data_end = 5000
df = pd.read_csv(
        "/Users/krasimirtrifonov/Documents/GitHub/Transformer-Neural-Network/dataset/EURUSD_Daily_200005300000_202405300000.csv",
        delimiter="\t")

# Extract the closing prices
closing = df["<CLOSE>"].iloc[data_start:data_end]
#%% md
### Augmented Dickey-Fuller (ADF) test using the statsmodels library:
#%%
from statsmodels.tsa.stattools import adfuller

# Assume 'data' is your dataframe and '<CLOSE>' is the column of interest

# Perform Augmented Dickey-Fuller test on the <CLOSE> column
adf_result = adfuller(df['<CLOSE>'])

# Extract the test statistics and p-value
adf_statistic = adf_result[0]
p_value = adf_result[1]
critical_values = adf_result[4]

# Print the results
print(f'ADF Statistic: {adf_statistic:.4f}')
print(f'p-value: {p_value:.4f}')
print('Critical Values:')
for key, value in critical_values.items():
    print(f'   {key}: {value:.4f}')
#%% md
        •   The p-value of 0.253 is higher than the usual significance levels (1%, 5%, or even 10%), meaning we fail to reject the null hypothesis of
```

```
34 the test. This suggests that the data is non-
   stationary.
35         •   The ADF statistic is greater than all
   critical values, further confirming that the series
    likely has a unit root (non-stationary).
36     Let try to make it stationary by Differencing
   and we perform the ADF test
37
38 #%%
39 # Differencing
40 close_diff = df['<CLOSE>'].diff().dropna()
41 #%% md
42 ### *Apply ADF on Difference Close
43 #%%
44 # Perform Augmented Dickey-Fuller test on the <
   CLOSE> column
45 adf_result = adfuller(close_diff)
46
47 # Extract the test statistics and p-value
48 adf_statistic = adf_result[0]
49 p_value = adf_result[1]
50 critical_values = adf_result[4]
51
52 # Print the results
53 print(f'ADF Statistic on Difference Close: {
   adf_statistic:.4f}')
54 print(f'p-value: {p_value:.4f}')
55 print('Critical Values:')
56 for key, value in critical_values.items():
57     print(f'   {key}: {value:.4f}')
58 #%% md
59     •   The ADF statistic is much lower than the
   critical values, and the p-value is 0.0, meaning we
    can now reject the null hypothesis.
60     •   This indicates that the differenced data is
    stationary.
61 #%% md
62 ###    -Now let check for Engle’s ARCH test to
   check for <strong>heteroscedasticity</strong>
63 #%%
64 from statsmodels.stats.diagnostic import het_arch
```

```
65
66 # Perform Engle's ARCH test
67 arch_test_result = het_arch(close_diff)
68
69 # Extract the test statistic and p-value
70 arch_statistic = arch_test_result[0]
71 arch_p_value = arch_test_result[1]
72
73 # Print the results
74 print(f'ARCH Test Statistic on Close Diff: {
   arch_statistic:.4f}')
75 print(f'p-value: {arch_p_value:.4f}')
76 #%% md
77 ##  •   A low p-value (typically < 0.05) suggests
   heteroscedasticity (i.e., variance that changes
   over time).
78 ### Let try on repeating diff
79 #%%
80 close_diff_diff = close_diff.diff().dropna()
81 # Perform Engle's ARCH test on Repeating Close
   Diff
82 arch_test_result = het_arch(close_diff_diff)
83
84 # Extract the test statistic and p-value
85 arch_statistic = arch_test_result[0]
86 arch_p_value = arch_test_result[1]
87
88 # Print the results
89 print(f'ARCH Test Statistic on Repeating Close
   Diff: {arch_statistic:.4f}')
90 print(f'p-value: {arch_p_value:.9f}')
91
92 import matplotlib.pyplot as plt
93
94 # Plot the first 50 points of the second
   differenced data
95 plt.figure(figsize=(10, 6))
96
97 # Plot the second difference of <CLOSE>
98 plt.plot(close_diff_diff.head(50), marker='o',
   linestyle='-', label='Second Difference of <CLOSE
```

```
 98 >')
 99
100 # Add plot title and labels
101 plt.title('First 50 Points of Second Difference <
    CLOSE>')
102 plt.xlabel('Index')
103 plt.ylabel('Second Difference Close')
104 plt.grid(True)  # Adds a grid for better
    readability
105 plt.legend()  # Shows the legend
106
107 # Display the plot
108 plt.show()
109 #%% md
110 ### •   The extremely low p-value suggests that
    even after applying a second differencing, the
    data still exhibits heteroscedasticity (variance
    is not constant over time).
111 ## It is Time to apply GARCH
112 #%%
113 import pandas as pd
114 from arch import arch_model
115
116 # Load your data (ensure the percent change data
    is properly calculated)
117 # Assuming 'data' is the DataFrame with a '
    CLOSE_PERCENT_CHANGE' column
118
119 # Prepare the percent change data for GARCH
    modeling
120 garch_data = close_diff
121
122 # Fit a GARCH(1, 1) model on the percent change
    data
123 garch_model = arch_model(garch_data, vol='Garch',
    p=1, q=1)
124 garch_fitted = garch_model.fit(disp="off")
125
126 # Summarize the model
127 print(garch_fitted.summary())
128
```

```
129 # Forecast future volatility (for the next 5 days
    , for example)
130 forecast = garch_fitted.forecast(horizon=5)
131 print(forecast.variance[-1:])  # This gives the
    predicted variance (volatility) for the next 5
    periods
132 #%% md
133 The results of your GARCH(1,1) model are as
    follows, with some key interpretations and next
    steps:
134
135 GARCH Model Summary:
136
137 Mean Model:
138
139     •   mu: The mean (constant) of the series is
    approximately 0.000024, with a p-value of 0.729,
    which indicates that the mean is not statistically
     significant.
140
141 Volatility (GARCH) Model:
142
143     •   omega: 1.0101 \times 10^{-6} (p-value = 0.
    000) - This is the baseline level of variance, and
     it is statistically significant.
144     •   alpha[1]: 0.0500 (p-value = 0.000) - This
    measures the influence of past shocks (squared
    returns or volatility) on future volatility. The
    significance indicates that past volatility
    influences future volatility.
145     •   beta[1]: 0.9300 (p-value = 0.000) - This
    measures the persistence of volatility. A value
    close to 1 indicates high persistence, meaning
    that volatility tends to remain elevated for
    longer periods.
146
147 Key Observations:
148
149     1.  High persistence of volatility: With a
    beta of 0.93, volatility is persistent, meaning
    that once volatility rises, it tends to remain
```

high for several periods.

2. Statistical significance: Both the alpha and beta coefficients are highly significant, meaning the GARCH model is capturing the volatility structure well.

3. Normal Distribution: The residuals (errors) are assumed to follow a normal distribution, but other distributions (like Student's t) might improve the fit.

Forecast:

The model produces forecasts for the next 5 periods of volatility:

- The forecasted volatility (variance) starts at 0.000021 and increases slightly to 0.000023 by the 5th period.

Convergence Warning:

- The message "Inequality constraints incompatible" suggests that the model did not converge fully. This could mean:
- The starting values or constraints may need adjustment.
- Trying different distributions (e.g., Student's t instead of normal) might help improve model fit.

Next Steps:

1. Refining the Model:
- Try changing the distribution assumption from normal to Student's t to allow for heavier tails.
- Ensure proper convergence by adjusting model parameters or re-running the optimization with different settings.
2. Volatility Interpretation:
- The forecasted volatility is low (around 0

```
171 .000021), indicating stable conditions in the
    short term. If you’re looking for predictive
    insights, this information could be useful in
    understanding future market behavior.
172     3.  Residual Analysis:
173     •   Check the residuals from the model to
    ensure no significant patterns or autocorrelation
    are left.
174 #%%
175 # Rescale the data (multiply by 100)
176 scaled_data = close_diff * 100
177
178 # Fit the GARCH model with rescaled data
179 garch_model = arch_model(scaled_data, vol='Garch'
    , p=1, q=1)
180 garch_rescaled_fitted = garch_model.fit(disp="off"
    )
181
182 # Summarize the model
183 print(garch_rescaled_fitted.summary())
184
185
186 # Forecasting future volatility with rescaled data
187 forecast = garch_rescaled_fitted.forecast(horizon=
    5)
188 forecast_variance_rescaled = forecast.variance[-1
    :] / 100  # Scale back
189 print(forecast_variance_rescaled)
190 #%% md
191 The recommendation to scale rather than normalize
    might seem unusual because in many machine
    learning contexts, normalization or
    standardization is a common preprocessing step.
    However, in the case of GARCH models (and other
    econometric models), there are reasons why scaling
     is preferred over normalization for better
    optimization and interpretation:
192
193 Why Scaling is Recommended in GARCH Models:
194
195     1.  Econometric Models Work with Raw Value
```

```
Ranges:
    •  In time series econometrics, especially with models like GARCH, the parameters are sensitive to the actual magnitude of the data. The variance and returns are calculated directly from the raw data, so the absolute scale of the input values affects how the optimizer estimates the volatility structure.
    •  Scaling by a constant factor (like multiplying by 100) maintains the relationships in the data, while simply normalizing (which changes the range to [0, 1]) may not preserve the underlying structure of volatility or returns in the same way.
    2. Normalization Alters Statistical Properties:
    •  Normalizing the data (scaling it between 0 and 1 or z-scoring) would modify the mean and variance of the data. For models like GARCH, which rely on these properties to model volatility, such changes can disrupt the model's ability to accurately estimate parameters.
    •  Scaling by a constant factor (e.g., multiplying by 100) preserves the mean-to-variance ratio and leaves the overall structure of the time series unchanged.
    3. Convergence Stability:
    •  Optimization algorithms in econometric models often perform better when the data is in a reasonable range (not too small or too large). If the data has very small values (as seen with your original data), the optimizer might struggle to find a solution due to the precision required for floating-point calculations. Rescaling moves the data into a more manageable range, improving convergence.
    4. Maintaining Interpretability:
    •  When you scale the data by a constant (e.g., 100), the output from the GARCH model can still be interpreted in terms of the original scale. You can easily scale the results (e.g., volatility
```

```
204  forecasts) back to the original units.
205      •   In contrast, normalization would make
    interpreting the model's output more difficult, as
     it would no longer be in the original units of
    the data.
206
207 To Clarify:
208
209      •   Scaling is a linear transformation that
    preserves the relationships between the data
    points (i.e., a proportional increase or decrease
    in values).
210      •   Normalization changes the distribution of
    the data, which can affect models that rely on the
     original statistical properties of the series (
    like variance, mean, and residuals).
211
212 Example:
213
214 If your data values are small (e.g., 5.05 \times
    10^{-5}), the optimizer might have trouble
    accurately estimating parameters because the
    changes between points are too subtle. Scaling by
    100 simply stretches the data but keeps the
    relationship between data points intact.
215
216 Summary:
217
218      •   Normalization could alter the statistical
    properties (mean, variance) that are critical for
    GARCH modeling.
219      •   Scaling preserves the data's structure
    while making the optimization easier and improving
     convergence.
220      •   After fitting the model, you can easily
    scale the results back to the original units if
    needed.
221 #%% md
222 Step 1: Plot Conditional Volatility Over Time
223
224 You can plot the conditional variance (volatility
```

```
224 ) over time, which gives insight into how well the
     model captures changing volatility.
225 #%%
226 # Plot the conditional volatility over time (
    variance over time)
227 plt.figure(figsize=(10, 6))
228 # Plot the conditional volatility (square root of
    variance)
229 conditional_volatility = garch_rescaled_fitted.
    conditional_volatility
230 plt.plot(garch_rescaled_fitted.
    conditional_volatility, label='Conditional
    Volatility')
231 plt.title('Conditional Volatility Over Time (GARCH
     Model)')
232 plt.xlabel('Time')
233 plt.ylabel('Conditional Volatility')
234 plt.grid(True)
235 plt.legend()
236 plt.show()
237 #%% md
238 Step 2: Residual Analysis
239
240 After fitting the GARCH model, we can check
    whether the residuals are behaving like white
    noise (uncorrelated and normally distributed).
    This is a key test to verify if the model has
    captured the volatility dynamics adequately.
241 #%%
242 # Plot the standardized residuals
243 plt.figure(figsize=(10, 6))
244 resid = garch_rescaled_fitted.resid
245 plt.plot(garch_rescaled_fitted.resid, label='
    Standardized Residuals')
246 plt.title('Standardized Residuals from GARCH Model
    ')
247 plt.xlabel('Time')
248 plt.ylabel('Residuals')
249 plt.grid(True)
250 plt.legend()
251 plt.show()
```

```
#%%
from statsmodels.stats.diagnostic import acorr_ljungbox
import numpy as np
# Test for autocorrelation in the residuals using the Ljung-Box test
ljung_box_test = acorr_ljungbox(garch_rescaled_fitted.resid, lags=[10], return_df=True)

print(ljung_box_test)
if  np.array(ljung_box_test.lb_pvalue)[0]  > 0.05:
    print('there's no significant autocorrelation in the residuals, suggesting that the GARCH model has adequately captured the volatility dynamics')
else:
     print("GARCH is not adequate!")
#%% md
If the p-values from the Ljung-Box test are greater than 0.05, it means there's no significant autocorrelation in the residuals, suggesting that the GARCH model has adequately captured the volatility dynamics.
#%%
# Use a Q-Q plot to check if residuals are normally distributed
import scipy.stats as stats
stats.probplot(garch_rescaled_fitted.resid, dist="norm", plot=plt)
plt.title('Q-Q Plot of GARCH Model Residuals')
plt.show()
#%% md
The Q-Q plot compares the distribution of the residuals to a theoretical normal distribution. In an ideal case where the residuals are normally distributed, the points should fall along the diagonal line.

In this case, deviations from the line, especially in the tails, suggest that the residuals might not be perfectly normally distributed. This could
```

```
indicate the presence of fat tails (kurtosis) or skewness, which are common in financial time series data.
#%%
# Forecast future volatility for 5 periods
forecast_rescaled = garch_rescaled_fitted.forecast(horizon=5)

# Print forecasted variance
print("Forecasted variance over next 5 periods:")
print(forecast_rescaled.variance[-1:])
#%%
from scipy.stats import kurtosis, skew

# Assuming 'resid_data' contains the residuals from your GARCH model
# Replace 'resid_data['Residuals']' with your own data if necessary

# Calculate Skewness
skewness = skew(resid)

# Calculate Kurtosis
kurt = kurtosis(resid)

# Print the results
print(f'Skewness: {skewness:.4f}')
print(f'Kurtosis: {kurt:.4f}')
#%% md
    Skewness: 0.0247
    •   A skewness value close to zero suggests that the residuals are nearly symmetric. This means there’s no significant skewness in the data.
    Kurtosis: 2.47
    •   Kurtosis less than 3 indicates platykurtic behavior, meaning the residuals have lighter tails than a normal distribution.
        This suggests that extreme events ( outliers) are less frequent compared to what a normal distribution would predict.
    •   The residuals appear to have little
```

```
303 skewness, meaning they are relatively symmetric
    around the mean.
304     •   The kurtosis is slightly lower than 3,
    indicating fewer extreme values (fat tails)
    compared to a normal distribution.
305         This suggests that the normal distribution
     assumption might still hold, although it's on the
     borderline.
306 #%% md
307 Step 1: Fit GARCH Models with Different
    Distributions
308
309 You can fit the GARCH model with various
    distributions to determine the best fit:
310 #%%
311 from arch import arch_model
312
313 # Assuming 'rescaled_data' contains your rescaled
    returns or percent changes
314 # Fit GARCH(1,1) with different distributions
315
316 # Normal distribution (default)
317 garch_normal = arch_model(scaled_data, vol='Garch'
    , p=1, q=1, dist='normal').fit(disp="off")
318
319 # Student's t-distribution
320 garch_t = arch_model(scaled_data, vol='Garch', p=1
    , q=1, dist='t').fit(disp="off")
321
322 # Skewed Student's t-distribution
323 garch_skewt = arch_model(scaled_data, vol='Garch'
    , p=1, q=1, dist='skewt').fit(disp="off")
324
325 # Print model summaries
326 print("Normal Distribution GARCH:")
327 print(garch_normal.summary())
328
329 print("\nStudent's t-Distribution GARCH:")
330 print(garch_t.summary())
331
332 print("\nSkewed t-Distribution GARCH:")
```

```
print(garch_skewt.summary())
#%% md
Step 2: Compare the Distributions Using AIC

The AIC (Akaike Information Criterion) can be used to compare the different models. The model with the lowest AIC is considered the best fit.
#%%
# Compare AIC values for the different models
print(f"Normal AIC: {garch_normal.aic}")
print(f"Student's t AIC: {garch_t.aic}")
print(f"Skewed t AIC: {garch_skewt.aic}")
#%% md
    •   The Student’s t-distribution is likely capturing the fat tails better, which means that your data contains
     more extreme values than a normal distribution would predict.
     Since the Student’s t-distribution GARCH model has the best fit, let’s move forward with a deeper analysis. Here’s a breakdown of what we can do next:

1. Residual Analysis for the Student’s t-Distribution GARCH Model:

    •   We need to ensure that the residuals from the Student’s t-distribution GARCH model behave like white noise (i.e., no autocorrelation and are approximately normally distributed).

2. Forecast Volatility Using the Student’s t-Distribution GARCH Model:

    •   We can also forecast future volatility using this model to see how well it captures future uncertainty.
#%%
# Plot the standardized residuals from the Student's t-distribution GARCH model
plt.figure(figsize=(10, 6))
```

```
plt.plot(garch_t.resid, label='Standardized Residuals (Student\'s t GARCH)')
plt.title('Standardized Residuals from Student\'s t GARCH Model')
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.grid(True)
plt.legend()
plt.show()
#%% md
Step 2: Test for Autocorrelation in the Residuals

We'll use the Ljung-Box test to check if there is any remaining autocorrelation in the residuals. Ideally, there should be no significant autocorrelation left after the GARCH model is applied.
#%%

```