# Durbin watson result is 0.035

A Durbin-Watson statistic of 0.035 suggests significant positive autocorrelation in the residuals of your model. In other words, the residuals are not independent and exhibit a strong correlation with their past values. This indicates that the model has not fully captured the underlying patterns in the data, leaving significant autocorrelation unaccounted for. Here’s how to interpret this and what you might do next:

Interpretation

Positive Autocorrelation:

Value of 0.035: The Durbin-Watson statistic ranges from 0 to 4. Values near 0 indicate strong positive autocorrelation, values around 2 indicate no autocorrelation, and values near 4 indicate strong negative autocorrelation. A value of 0.035 is extremely close to 0, indicating that the residuals from your model are strongly positively autocorrelated.

Implication: This suggests that the residuals from your LSTM + GARCH model are not independent and tend to follow a pattern, meaning the model might not be capturing some underlying time series structure or dependencies effectively.

Potential Actions

1. Model Diagnostics and Refinement:

* LSTM Tuning: Consider revisiting your LSTM model parameters, such as the number of layers, units, look-back period, or even adding dropout for regularization. Ensure that the LSTM model captures long-term dependencies effectively.
* GARCH Refinement: Ensure the GARCH model is correctly specified for the residuals. You might need to try different GARCH variations (e.g., EGARCH, TGARCH) or adjust the GARCH order parameters.

1. Consider Additional Features:

* Lagged Features: Incorporate lagged values of the series as additional features for the LSTM model to capture more patterns.
* Exogenous Variables: Include other relevant external variables if available, which might influence the time series.

Hybrid Modeling:

* Additional Modeling: Use other hybrid approaches like combining with ARIMA models, or using models like SARIMA that might handle autocorrelation better.
* Ensemble Methods: Consider using ensemble learning methods to combine multiple models' outputs more effectively.

Residual Analysis:

* Residual Plots: Plot the residuals to visually inspect the autocorrelation. Use autocorrelation plots (ACF) to identify specific lags where autocorrelation is significant.
* White Noise Check: Ensure the residuals resemble white noise, indicating no predictable patterns left.

Re-evaluate the Overall Model Structure:

* Model Complexity: Ensure that your model is neither too simple (underfitting) nor too complex (overfitting).
* Incorporate Seasonality: Check if seasonal components or other trends have been properly incorporated.

Example of Addressing Residual Autocorrelation:

* Adding Lagged Features

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from arch import arch\_model

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot\_acf

# Assume vix\_data is your DataFrame containing the VIX data with columns 'DATE' and 'VIX'

# Normalize the data

scaler = StandardScaler()

vix\_scaled = scaler.fit\_transform(vix\_data['VIX'].values.reshape(-1, 1))

# Create sequences with lagged features

look\_back = 10

def create\_sequences(data, look\_back):

X, Y = [], []

for i in range(len(data) - look\_back):

X.append(data[i:(i + look\_back)])

Y.append(data[i + look\_back])

return np.array(X), np.array(Y)

X, Y = create\_sequences(vix\_scaled, look\_back)

# Split the data into training and testing sets

train\_size = int(len(X) \* 0.8)

X\_train, Y\_train = X[:train\_size], Y[:train\_size]

X\_test, Y\_test = X[train\_size:], Y[train\_size:]

# Define and train the LSTM model

model = Sequential()

model.add(LSTM(units=50, input\_shape=(look\_back, 1)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

history = model.fit(X\_train, Y\_train, epochs=100, batch\_size=1, validation\_split=0.2, verbose=0)

# Predict using the LSTM model

lstm\_predictions = model.predict(X\_test)

# Inverse transform the predictions and actual values

lstm\_predictions\_rescaled = lstm\_predictions \* scaler.scale\_[0] + scaler.mean\_[0]

Y\_test\_rescaled = Y\_test \* scaler.scale\_[0] + scaler.mean\_[0]

# Calculate residuals

residuals = Y\_test\_rescaled.flatten() - lstm\_predictions\_rescaled.flatten()

# Fit GARCH model on residuals

garch\_model = arch\_model(residuals, vol='Garch', p=1, q=1)

garch\_fit = garch\_model.fit(disp='off')

# Forecast with GARCH model

garch\_forecast = garch\_fit.forecast(horizon=len(residuals))

garch\_volatility = garch\_forecast.variance.values[-1, :]

# Adjust LSTM predictions using GARCH volatility

adjusted\_predictions = lstm\_predictions\_rescaled.flatten() + garch\_volatility

# Calculate MAPE

mape\_value = mean\_absolute\_percentage\_error(Y\_test\_rescaled.flatten(), adjusted\_predictions)

print(f"MAPE: {mape\_value}")

# Plot residuals to check for autocorrelation

plot\_acf(residuals, lags=40)

plt.show()

# Plot actual vs predicted

plt.plot(Y\_test\_rescaled, label='Actual', color='blue')

plt.plot(adjusted\_predictions, label='Predicted', color='red')

plt.title('VIX Forecast')

plt.ylabel('VIX')

plt.xlabel('Time')

plt.legend()

plt.show()

* Consider Adding ARIMA

from statsmodels.tsa.arima.model import ARIMA

# Fit ARIMA model on residuals

arima\_model = ARIMA(residuals, order=(1, 0, 1))

arima\_fit = arima\_model.fit()

# Predict using ARIMA model

arima\_predictions = arima\_fit.predict(start=0, end=len(residuals) - 1)

# Combine ARIMA predictions with LSTM predictions

combined\_predictions = lstm\_predictions\_rescaled.flatten() + arima\_predictions

# Calculate MAPE

mape\_value = mean\_absolute\_percentage\_error(Y\_test\_rescaled.flatten(), combined\_predictions)

print(f"MAPE: {mape\_value}")

# Plot actual vs predicted

plt.plot(Y\_test\_rescaled, label='Actual', color='blue')

plt.plot(combined\_predictions, label='Predicted', color='red')

plt.title('VIX Forecast')

plt.ylabel('VIX')

plt.xlabel('Time')

plt.legend()

plt.show()

By addressing the autocorrelation effectively, you can improve the model's forecasting ability and reduce the dependency in the residuals, leading to more accurate predictions.

1. For Garch model the data needs to be stationary but the LSTM Model residuals are non-stationary.

ChatGPT

If the residuals from the LSTM model are non-stationary, this poses a problem for fitting a GARCH model, which requires stationary data. To address this, you must first make the residuals stationary before applying the GARCH model. Here’s a step-by-step guide on how to handle this issue:

Steps to Address Non-Stationary Residuals

Check for Stationarity:

* Use statistical tests such as the Augmented Dickey-Fuller (ADF) test or the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test to check if the residuals are stationary.

Transform Residuals:

* Differencing: Apply differencing to the residuals to remove trends and achieve stationarity.
* Log Transform: Apply log transformation if the data shows exponential growth or volatility clustering.
* Deseasonalize: Remove any seasonal components if identified.

Fit GARCH Model:

* Once the residuals are made stationary, fit the GARCH model to the transformed residuals.

Reconstruct Forecast:

* Use the GARCH model's output to adjust the LSTM predictions, considering the transformations applied.

Example: Making Residuals Stationary and Applying GARCH

1. Generate Residuals and Check Stationarity:

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from arch import arch\_model

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller, kpss

# Assume vix\_data is your DataFrame containing the VIX data with columns 'DATE' and 'VIX'

# Normalize the data

scaler = StandardScaler()

vix\_scaled = scaler.fit\_transform(vix\_data['VIX'].values.reshape(-1, 1))

# Create sequences for LSTM

look\_back = 10

def create\_sequences(data, look\_back):

X, Y = [], []

for i in range(len(data) - look\_back):

X.append(data[i:(i + look\_back)])

Y.append(data[i + look\_back])

return np.array(X), np.array(Y)

X, Y = create\_sequences(vix\_scaled, look\_back)

# Split the data into training and testing sets

train\_size = int(len(X) \* 0.8)

X\_train, Y\_train = X[:train\_size], Y[:train\_size]

X\_test, Y\_test = X[train\_size:], Y[train\_size:]

# Define and train the LSTM model

model = Sequential()

model.add(LSTM(units=50, input\_shape=(look\_back, 1)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

history = model.fit(X\_train, Y\_train, epochs=100, batch\_size=1, validation\_split=0.2, verbose=0)

# Predict using the LSTM model

lstm\_predictions = model.predict(X\_test)

# Inverse transform the predictions and actual values

lstm\_predictions\_rescaled = lstm\_predictions \* scaler.scale\_[0] + scaler.mean\_[0]

Y\_test\_rescaled = Y\_test \* scaler.scale\_[0] + scaler.mean\_[0]

# Calculate residuals

residuals = Y\_test\_rescaled.flatten() - lstm\_predictions\_rescaled.flatten()

# Check stationarity using ADF and KPSS tests

def check\_stationarity(data):

adf\_test = adfuller(data)

kpss\_test = kpss(data, nlags='auto')

print(f"ADF Statistic: {adf\_test[0]}, p-value: {adf\_test[1]}")

print(f"KPSS Statistic: {kpss\_test[0]}, p-value: {kpss\_test[1]}")

return adf\_test[1] < 0.05 and kpss\_test[1] > 0.05

stationarity = check\_stationarity(residuals)

print(f"Are residuals stationary? {'Yes' if stationarity else 'No'}")

1. Transform Residuals (Differencing):

if not stationarity:

diff\_residuals = np.diff(residuals)

stationarity = check\_stationarity(diff\_residuals)

print(f"Are differenced residuals stationary? {'Yes' if stationarity else 'No'}")

else:

diff\_residuals = residuals

1. Fit GARCH Model on Stationary Residuals:

garch\_model = arch\_model(diff\_residuals, vol='Garch', p=1, q=1)

garch\_fit = garch\_model.fit(disp='off')

# Forecast with GARCH model

garch\_forecast = garch\_fit.forecast(horizon=len(diff\_residuals))

garch\_volatility = garch\_forecast.variance.values[-1, :]

# Adjust LSTM predictions using GARCH volatility

adjusted\_predictions = lstm\_predictions\_rescaled.flatten()[1:] + garch\_volatility

# Re-adjust differencing

adjusted\_predictions = np.insert(adjusted\_predictions, 0, lstm\_predictions\_rescaled.flatten()[0])

# Calculate MAPE

from sklearn.metrics import mean\_absolute\_percentage\_error

mape\_value = mean\_absolute\_percentage\_error(Y\_test\_rescaled.flatten(), adjusted\_predictions)

print(f"MAPE: {mape\_value}")

# Plot actual vs predicted

plt.plot(Y\_test\_rescaled, label='Actual', color='blue')

plt.plot(adjusted\_predictions, label='Predicted', color='red')

plt.title('VIX Forecast')

plt.ylabel('VIX')

plt.xlabel('Time')

plt.legend()

plt.show()

Key Points to Remember:

Stationarity: Making the residuals stationary is crucial for the GARCH model. Non-stationary residuals will lead to invalid results from the GARCH model.

Transformations: Choose the appropriate transformation (differencing, log transformation) based on the nature of the residuals.

Reconstruction: Ensure that any transformations applied are reversed correctly when combining predictions.

By ensuring that residuals are stationary, you can effectively apply the GARCH model to capture volatility and improve the overall forecast quality.