Untitled11

December 15, 2025

1 Time Series Forecasting of NFLX Stock Prices

This project performs exploratory data analysis, stationarity testing, ARIMA modeling, GARCH volatility modeling, and combined ARIMA+GARCH forecasting on Netflix stock prices.

[17]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
sns.set_style("whitegrid")
%matplotlib inline
```

[2]:
```
file_path = r"C:/Users/USER/Music/Self Projects/Time Series forecasting of␣
↪Stock Prices/NFLX_stocks.csv"
df = pd.read_csv(file_path)

if 'unnamed: 0' in df.columns:
    df.rename(columns={'unnamed: 0': 'date'}, inplace=True)

df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)
df.sort_index(inplace=True)

df['close'] = pd.to_numeric(df['close'], errors='coerce')
df = df.dropna(subset=['close'])

df.head()
```

[2]:
```
                          open                high                 low  \
date
2002-05-23  1.1564290523529053  1.2428569793701172  1.1457140445709229
2002-05-24   1.214285969734192   1.225000023841858  1.1971429586410522
2002-05-28  1.2135709524154663  1.2321430444717407   1.157142996788025
2002-05-29  1.1642860174179077  1.1642860174179077  1.0857139825820923
```

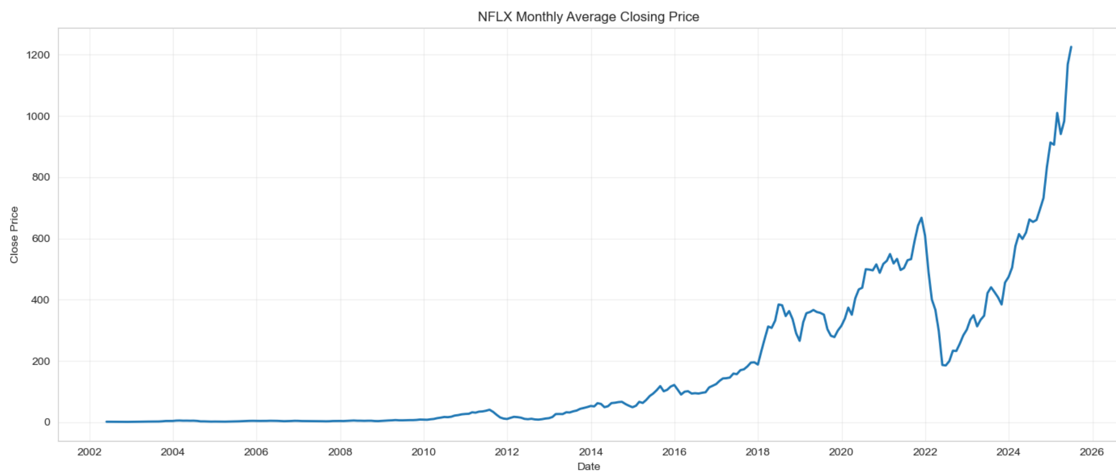
```
2002-05-30  1.1078569889068604  1.1078569889068604  1.0714290142059326

               close           adj_close     volume
date
2002-05-23  1.196429  1.1964290142059326  104790000
2002-05-24  1.210000  1.2100000381469727   11104800
2002-05-28  1.157143   1.157142996788025    6609400
2002-05-29  1.103571  1.1035710573196411    6757800
2002-05-30  1.071429  1.0714290142059326   10154200
```

[3]:
```python
df['close'].describe()
```

[3]:
```
count    5802.000000
mean      172.807773
std       234.885388
min         0.372857
25%         4.307143
50%        47.187143
75%       317.935005
max      1250.520020
Name: close, dtype: float64
```

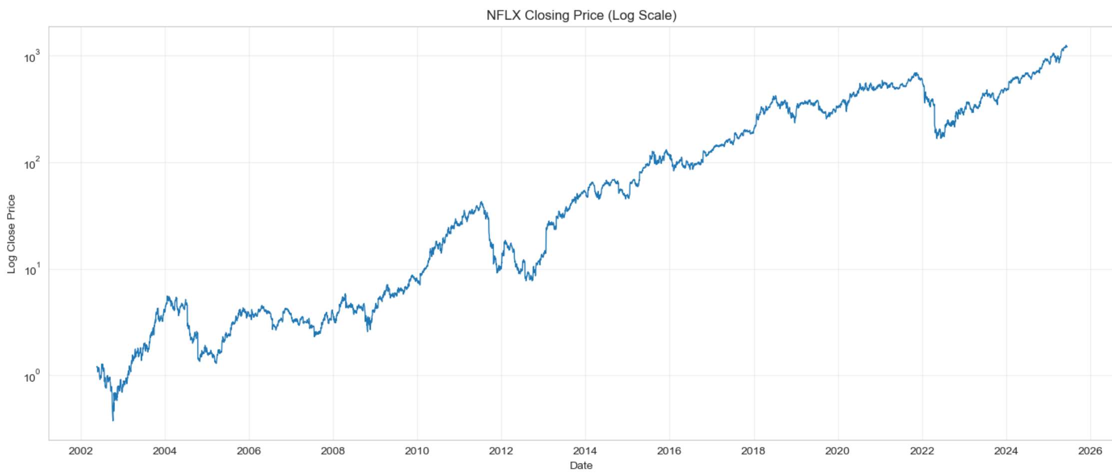
[4]:
```python
plt.figure(figsize=(14,6))
plt.plot(df.index, df['close'], linewidth=1)

plt.title("NFLX Closing Price Time Series")
plt.xlabel("Date")
plt.ylabel("Close Price")

plt.gca().xaxis.set_major_locator(mdates.YearLocator(2))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

[5]:
```python
monthly_close = df['close'].resample('M').mean()

plt.figure(figsize=(14,6))
plt.plot(monthly_close, linewidth=2)

plt.title("NFLX Monthly Average Closing Price")
plt.xlabel("Date")
plt.ylabel("Close Price")

plt.gca().xaxis.set_major_locator(mdates.YearLocator(2))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



[6]:

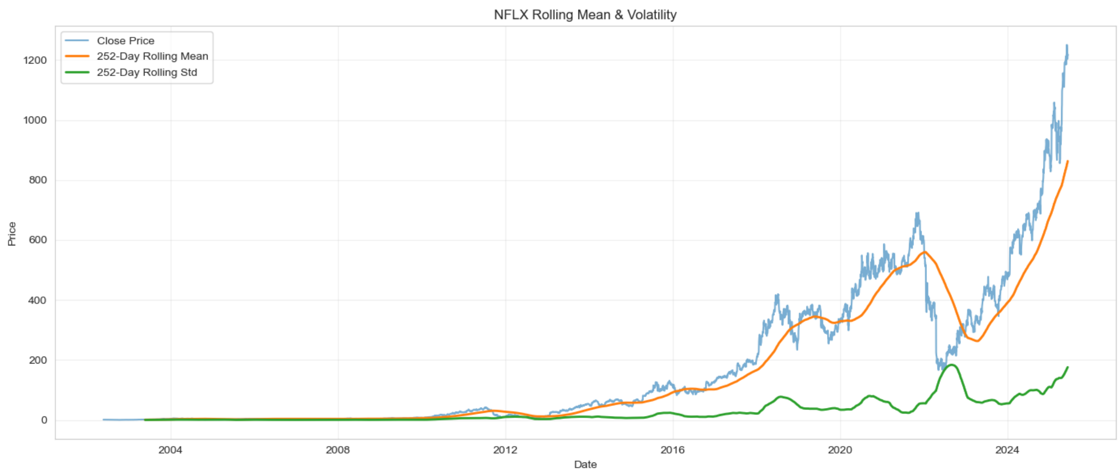
```
plt.figure(figsize=(14,6))
plt.plot(df.index, df['close'], linewidth=1)

plt.yscale('log')
plt.title("NFLX Closing Price (Log Scale)")
plt.xlabel("Date")
plt.ylabel("Log Close Price")

plt.gca().xaxis.set_major_locator(mdates.YearLocator(2))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



[7]:

```
rolling_mean = df['close'].rolling(window=252).mean()
rolling_std = df['close'].rolling(window=252).std()

plt.figure(figsize=(14,6))
plt.plot(df['close'], label='Close Price', alpha=0.6)
plt.plot(rolling_mean, label='252-Day Rolling Mean', linewidth=2)
plt.plot(rolling_std, label='252-Day Rolling Std', linewidth=2)

plt.title("NFLX Rolling Mean & Volatility")
plt.xlabel("Date")
plt.ylabel("Price")

plt.legend()
plt.grid(True, alpha=0.3)
```

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

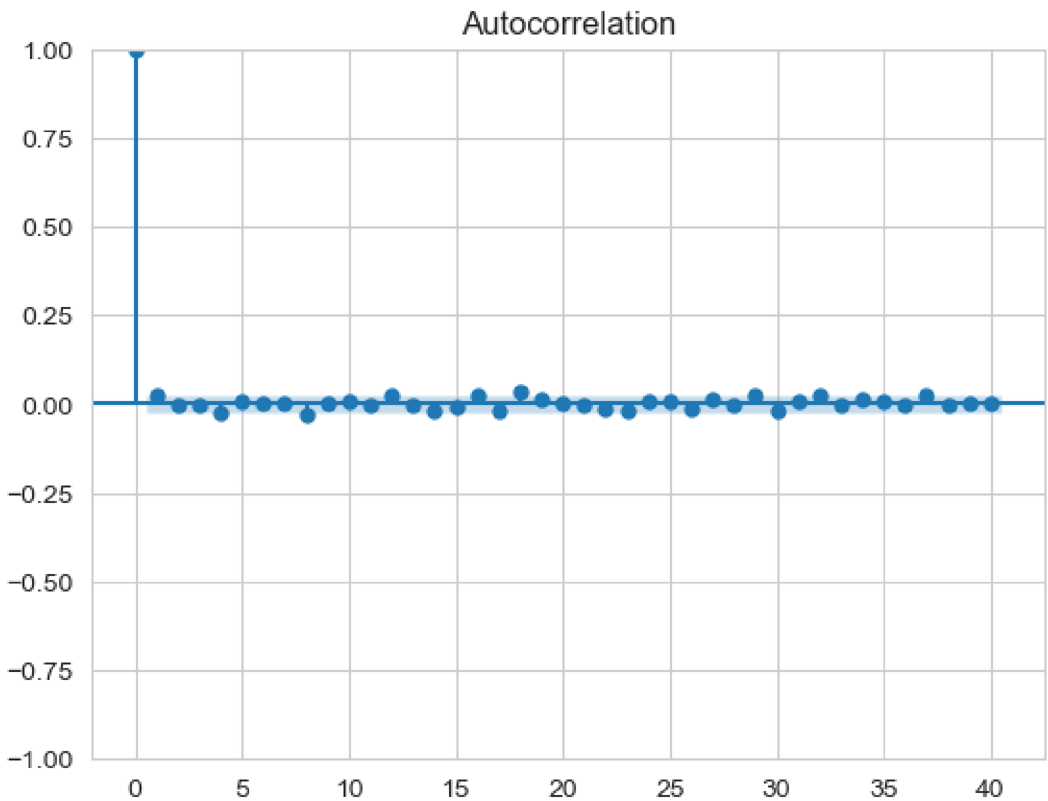


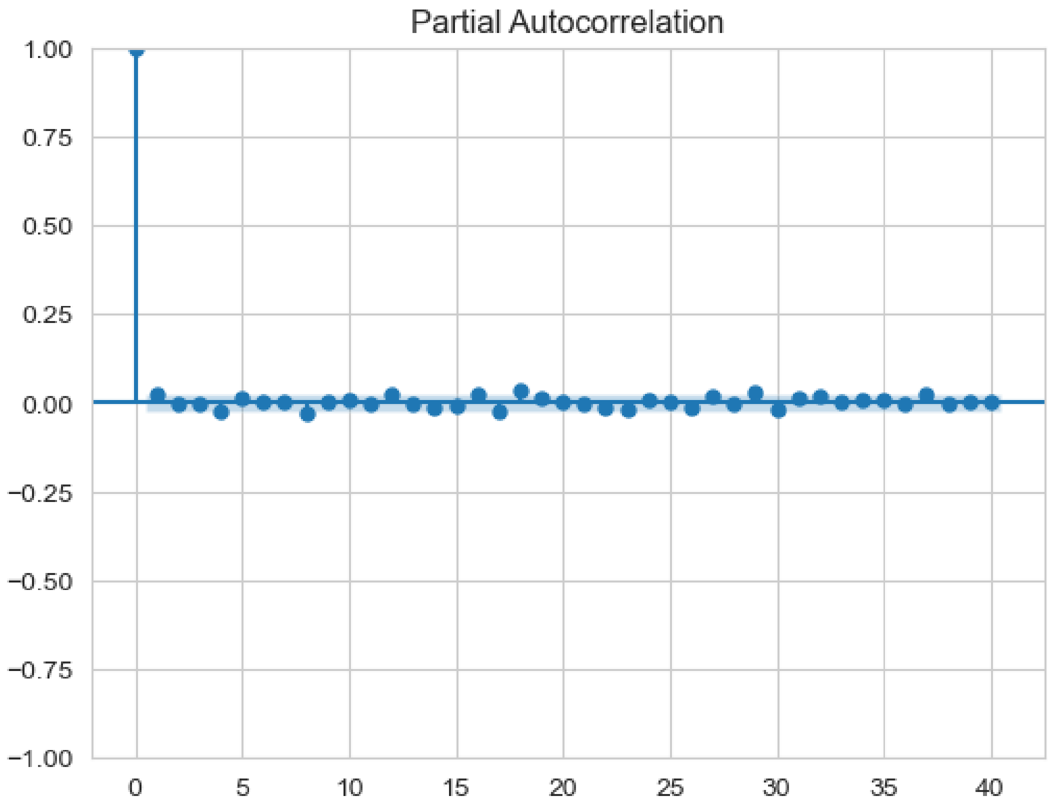
[8]:
```
from statsmodels.tsa.stattools import adfuller, kpss

def adf_test(series):
    return adfuller(series, autolag='AIC')[1]

def kpss_test(series):
    return kpss(series, regression='c', nlags="auto")[1]

print("ADF (level):", adf_test(df['close']))
print("KPSS (level):", kpss_test(df['close']))

df['log_close'] = np.log(df['close'])
df['diff_log_close'] = df['log_close'].diff()

print("ADF (diff log):", adf_test(df['diff_log_close'].dropna()))
print("KPSS (diff log):", kpss_test(df['diff_log_close'].dropna()))
```

```
ADF (level): 1.0
KPSS (level): 0.01

C:\Users\USER\AppData\Local\Temp\ipykernel_53620\1044478671.py:7:
InterpolationWarning: The test statistic is outside of the range of p-values
available in the
look-up table. The actual p-value is smaller than the p-value returned.

  return kpss(series, regression='c', nlags="auto")[1]

ADF (diff log): 0.0
KPSS (diff log): 0.1
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_53620\1044478671.py:7:
InterpolationWarning: The test statistic is outside of the range of p-values
available in the
look-up table. The actual p-value is greater than the p-value returned.

  return kpss(series, regression='c', nlags="auto")[1]
```

[9]:
```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(df['diff_log_close'].dropna(), lags=40)
plt.show()

plot_pacf(df['diff_log_close'].dropna(), lags=40)
plt.show()
```



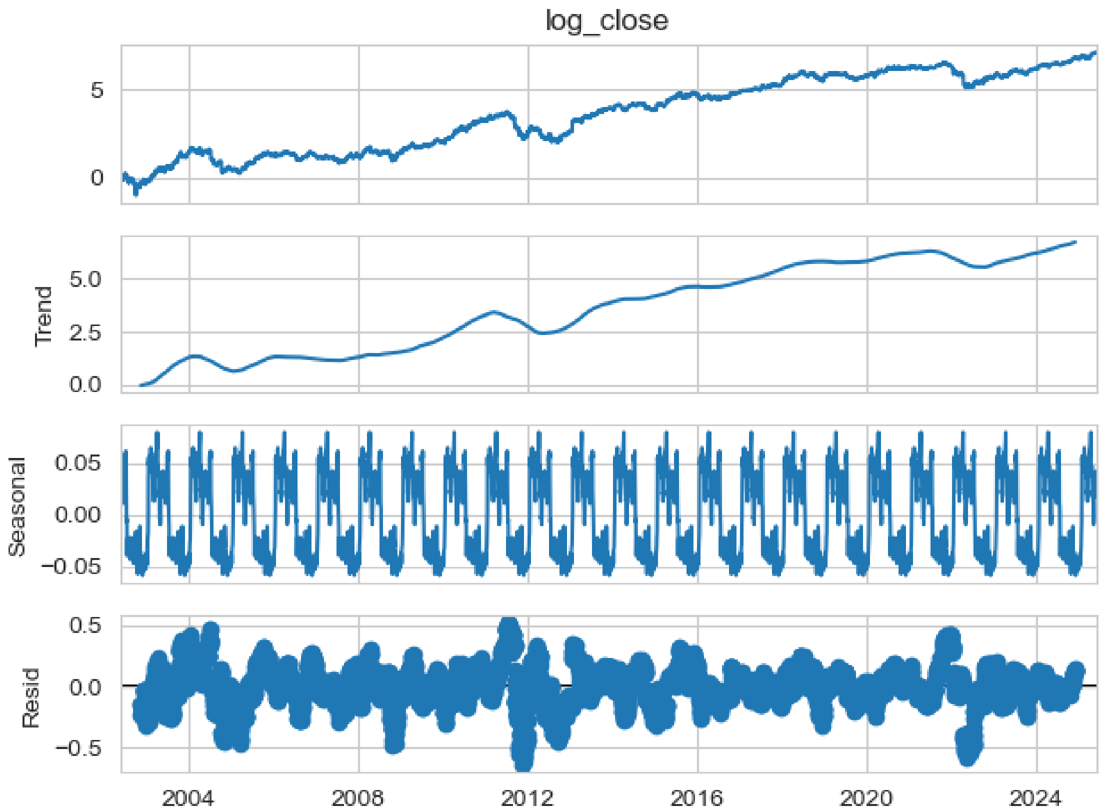


[10]:
```
from statsmodels.stats.diagnostic import acorr_ljungbox

acorr_ljungbox(df['diff_log_close'].dropna(), lags=[10], return_df=True)
```

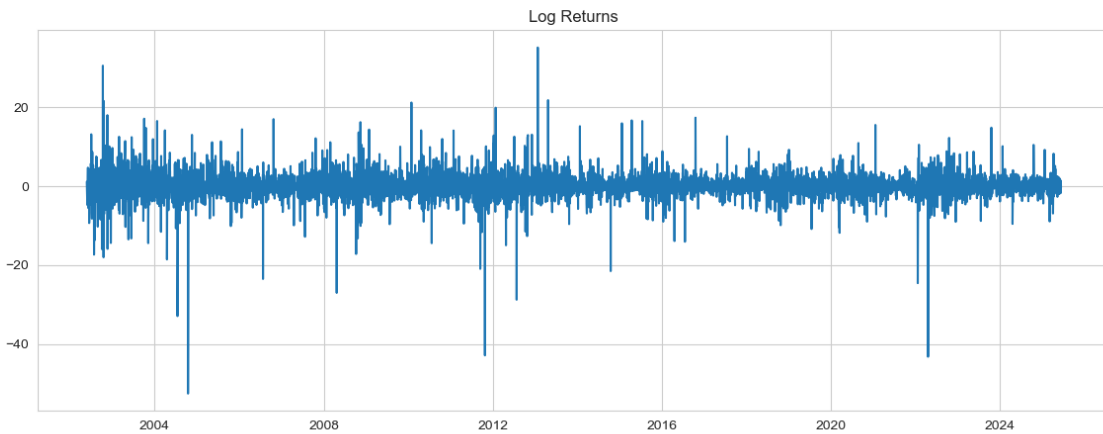
[10]:

	lb_stat	lb_pvalue
10	13.191168	0.213179

[11]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

seasonal_decompose(df['log_close'], model='additive', period=252).plot()
plt.show()
```



[12]:
```
df['returns'] = 100 * df['log_close'].diff()

plt.figure(figsize=(14,5))
plt.plot(df['returns'])
plt.title("Log Returns")
plt.show()

from statsmodels.stats.diagnostic import het_arch
het_arch(df['returns'].dropna())
```



[12]:

```
(33.15552477022573,
 0.0002564674388313528,
 3.3283103424612555,
 0.00025105222316720497)
```

[13]:

```python
from arch import arch_model

garch = arch_model(df['returns'].dropna(),
                   mean='zero',
                   vol='Garch',
                   p=1, q=1)

garch_fit = garch.fit(disp='off')
print(garch_fit.summary())
```

```
                       Zero Mean - GARCH Model Results
==============================================================================
Dep. Variable:                returns   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -15165.7
Distribution:                  Normal   AIC:                           30337.4
Method:            Maximum Likelihood   BIC:                           30357.4
                                        No. Observations:                 5801
Date:                Mon, Dec 15 2025   Df Residuals:                     5801
Time:                        20:48:52   Df Model:                            0
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0480  2.761e-02      1.740  8.188e-02  [-6.076e-03,  0.102]
alpha[1]       0.0136  4.255e-03      3.205  1.350e-03 [5.298e-03,2.198e-02]
beta[1]        0.9825  4.845e-03    202.781      0.000     [  0.973,  0.992]
```

```
================================================================================

Covariance estimator: robust
```

[14]:
```
from statsmodels.tsa.arima.model import ARIMA

arima = ARIMA(df['log_close'].dropna(), order=(1,1,1))
arima_fit = arima.fit()
print(arima_fit.summary())
```

```
C:\Users\USER\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\USER\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
C:\Users\USER\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)

                               SARIMAX Results
==============================================================================
Dep. Variable:              log_close   No. Observations:                 5802
Model:                 ARIMA(1, 1, 1)   Log Likelihood               11202.714
Date:                Mon, 15 Dec 2025   AIC                         -22399.428
Time:                        20:48:53   BIC                         -22379.431
Sample:                             0   HQIC                        -22392.472
                               - 5802
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.1104      0.362     -0.305      0.760      -0.819       0.599
ma.L1          0.1382      0.360      0.383      0.701      -0.568       0.845
sigma2         0.0012   6.11e-06    201.325      0.000       0.001       0.001
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
169623.70
Prob(Q):                              0.95   Prob(JB):
0.00
Heteroskedasticity (H):               0.45   Skew:
-1.12
Prob(H) (two-sided):                  0.00   Kurtosis:
29.40
================================================================================
```

```
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

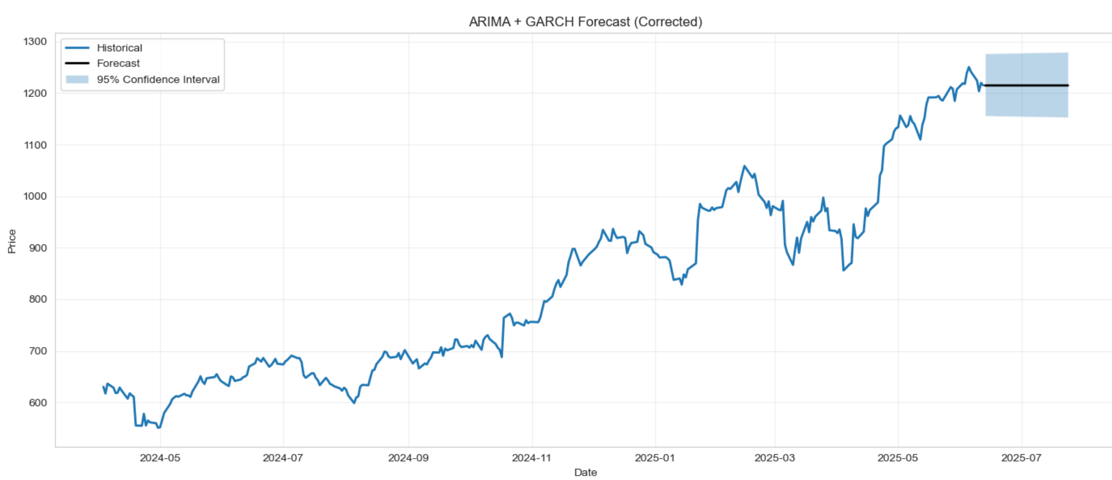
[15]:

```python
# ================================
# FIXED ARIMA + GARCH FORECAST
# ================================

n_forecast = 30

# ---------- ARIMA forecast ----------
arima_forecast = arima_fit.get_forecast(steps=n_forecast)
mean_log_forecast = arima_forecast.predicted_mean

# Create proper future date index
last_date = df.index[-1]
forecast_index = pd.date_range(
    start=last_date + pd.Timedelta(days=1),
    periods=n_forecast,
    freq='B'  # business days
)

mean_log_forecast.index = forecast_index

# ---------- GARCH forecast ----------
garch_forecast = garch_fit.forecast(horizon=n_forecast)

# Volatility of RETURNS
volatility = np.sqrt(garch_forecast.variance.iloc[-1].values)

# Scale volatility correctly (returns are %)
volatility = volatility / 100

# ---------- Combine correctly ----------
forecast_df = pd.DataFrame(index=forecast_index)
forecast_df['mean_log'] = mean_log_forecast.values
forecast_df['upper_log'] = forecast_df['mean_log'] + 1.96 * volatility
forecast_df['lower_log'] = forecast_df['mean_log'] - 1.96 * volatility

# Convert to price
forecast_df['mean_price'] = np.exp(forecast_df['mean_log'])
forecast_df['upper_price'] = np.exp(forecast_df['upper_log'])
forecast_df['lower_price'] = np.exp(forecast_df['lower_log'])
```

```
C:\Users\USER\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
```

```
with an integer index beginning at `start`.
  return get_prediction_index(
```

[16]:
```python
plt.figure(figsize=(14,6))

# Plot recent history only (important)
plt.plot(df.index[-300:], df['close'][-300:], label='Historical', linewidth=2)

# Plot forecast
plt.plot(forecast_df.index, forecast_df['mean_price'],
         label='Forecast', linewidth=2, color='black')

# Confidence bands
plt.fill_between(
    forecast_df.index,
    forecast_df['lower_price'],
    forecast_df['upper_price'],
    alpha=0.3,
    label='95% Confidence Interval'
)

plt.title("ARIMA + GARCH Forecast (Corrected)")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



1.1 Conclusion

Stock prices are non-stationary, while returns are stationary with strong volatility clustering. ARIMA effectively models the conditional mean, and GARCH captures time-varying volatility, making the combined ARIMA + GARCH framework optimal for financial time series forecasting.

[ ]:

```

```