Summary of "Machine Learning Methods Economists Should Know About" by Susan Athey and Guido W. Imbens

Abstract: The paper explores the integration of machine learning (ML) methods into economics and econometrics, highlighting the differences between ML and traditional econometrics. It discusses the importance of supervised and unsupervised learning methods, matrix completion methods, and new techniques at the intersection of ML and econometrics for empirical research in economics.

Introduction:

- **Leo Breiman's Perspective:** The distinction between model-based (traditional econometrics) and algorithmic (ML) approaches to statistics is emphasized. Breiman advocates for a diverse set of tools beyond traditional data models, a sentiment that Athey and Imbens echo for econometrics.
- **Adoption in Economics:** While slower than in statistics, ML methods are becoming more common in economics, particularly in "big data" settings where traditional methods may fall short.

Challenges in Adoption:

- **Cultural Differences:** Economics emphasizes formal properties like consistency and efficiency, while ML focuses on algorithm performance and error rates.
- **Confidence Intervals:** Traditional econometrics prioritizes constructing valid confidence intervals, whereas ML often does not. However, some ML methods are now being adapted to provide these intervals.

Adaptation of ML Techniques:

- **Problem-Specific Tuning:** Effective use of ML in economics often requires tuning algorithms to exploit the structure of economic problems, such as causal relationships and endogeneity.
- **New Methods:** Techniques like sample splitting and orthogonalization can improve ML estimators, offering properties like asymptotic normality.

Key ML Methods for Economists:

1. **Nonparametric Regression (Supervised Learning):** Useful for regression problems, focusing on predicting outcomes based on features.
2. **Classification (Supervised Learning):** Applying nonparametric regression techniques to discrete response models.
3. **Unsupervised Learning:** Includes clustering analysis and density estimation.
4. **Heterogeneous Treatment Effects:** Estimating effects that vary across individuals and optimal policy mapping.
5. **Experimental Design:** Using bandit approaches to improve experimentation.
6. **Matrix Completion:** Relevant for causal panel data models and consumer choice modeling.
7. **Text Analysis:** Employing ML methods for analyzing textual data in economics.

Comparison with Traditional Econometrics:

- **Goals:** Traditional econometrics focuses on estimating parameters with known properties, while ML prioritizes prediction accuracy.
- **Model Validation:** ML emphasizes out-of-sample validation to avoid overfitting, unlike traditional econometrics which relies more on theoretical model selection.
- **Regularization:** ML uses regularization to balance model complexity and predictive accuracy, a concept less emphasized in traditional econometrics.

Conclusion: The authors argue for the inclusion of ML methods in the core econometrics curriculum, emphasizing their potential to solve complex economic problems more effectively than traditional methods alone. They highlight the necessity for economists to adapt and tune ML techniques to the specific needs and structures of economic data and problems.

Summary of "An ARIMA-LSTM model for predicting volatile agricultural price series with random forest technique" by Soumic Ray et al., published in 2023 in the *Applied Soft Computing Journal*

Highlights

1. **Hybrid Model**: The paper introduces a hybrid ARIMA-LSTM model, utilizing the random forest algorithm to select suitable input lags for the LSTM component.
2. **Volatile Data Handling**: This model is capable of dealing with volatile datasets that exhibit skewedness.
3. **Application and Results**: The model was applied to three agricultural price series, yielding superior results compared to traditional models.
4. **Superiority of Hybrid Models**: The study demonstrates the effectiveness of hybrid machine learning models over conventional statistical models.

Introduction

- **Importance of Price Forecasting**: Accurate price forecasting is crucial for the efficient management of agricultural commodities, aiding stakeholders and policymakers.
- **Volatility in Agricultural Prices**: Agricultural price data are often volatile due to factors like market fluctuations, demand-supply gaps, and weather conditions.
- **Conventional Models**: Traditional models like ARIMA and GARCH have been used to handle such data, but they have limitations.
- **Machine Learning**: Recent advances in machine learning, particularly deep learning models like LSTM, have shown promise in forecasting complex time series.

Methodology

- **ARIMA Model**: Used to estimate the mean effect of the time series data.
- **GARCH Model**: Employed to capture the volatility in the residuals from the ARIMA model.
- **LSTM Model**: Applied to normalized training data to predict future values.
- **Hybrid ARIMA-LSTM Model**: Combines the strengths of ARIMA for linear modeling and LSTM for capturing non-linear patterns. Random forest is used for selecting input lags for the LSTM.

Results

- **Performance Metrics**: The hybrid model was evaluated using RMSE, MAPE, and MASE metrics. The proposed model showed improvements of 8-25% in RMSE, 2-28% in MAPE, and 2-29% in MASE compared to conventional models.
- **Application**: The model was applied to monthly price indices of gram, moong, and urad pulses, demonstrating its effectiveness.

Conclusion

The paper concludes that the hybrid ARIMA-LSTM model, with random forest-based lag selection, outperforms traditional models in forecasting volatile agricultural price series. The authors suggest that this approach could be extended to other volatile datasets, potentially enriching the literature on machine learning applications in time series forecasting.

Key Points

- **Volatility Modeling**: The study focuses on handling volatility, a common characteristic in agricultural price series.
- **Machine Learning Integration**: Integrating machine learning techniques like LSTM with traditional statistical models enhances forecasting accuracy.
- **Random Forest for Lag Selection**: Using random forest to determine the appropriate lags for the LSTM component is a novel approach that improves model performance.

This summary encapsulates the essence of the paper, highlighting the innovative hybrid model and its successful application to agricultural price forecasting.

CODE-

For Reliance Industries closing prices-
Applied ARIMA and LSTM Model

```
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)

import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

import pandas as pd
import numpy as np
from nsetools import Nse
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Fetch data from NSE (assuming historical data is available)
nse = Nse()
symbol = 'RELIANCE'

# Placeholder for historical data
# For demonstration, generate synthetic data
dates = pd.date_range(start='1/1/2020', periods=365, freq='D')
np.random.seed(0)
close_prices = np.random.normal(loc=2000, scale=50, size=len(dates))
df = pd.DataFrame(data={'Date': dates, 'Close': close_prices})

# Convert date to datetime and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df.index.freq = 'D'  # Explicitly set the frequency

# Plot original data
plt.figure(figsize=(10, 5))
plt.plot(df['Close'], label='Original Data')
plt.title('Closing Prices of Reliance Industries')
plt.xlabel('Date')
```

```
plt.ylabel('Close')
plt.legend()
plt.show()

# Fit ARIMA model
arima_model = ARIMA(df['Close'], order=(5,1,0))
arima_result = arima_model.fit()

# Get ARIMA predictions
df['ARIMA_Prediction'] = arima_result.predict(start=0, end=len(df)-1,
dynamic=False)

# Calculate residuals
df['Residuals'] = df['Close'] - df['ARIMA_Prediction']

# Plot ARIMA results
plt.figure(figsize=(10, 5))
plt.plot(df['Close'], label='Original Data')
plt.plot(df['ARIMA_Prediction'], label='ARIMA Prediction')
plt.title('ARIMA Model Prediction')
plt.xlabel('Date')
plt.ylabel('Close')
plt.legend()
plt.show()

# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data =
scaler.fit_transform(df['Residuals'].values.reshape(-1,1))

# Prepare dataset for LSTM
def create_dataset(data, time_step=1):
    X, Y = [], []
    for i in range(len(data)-time_step-1):
        a = data[i:(i+time_step), 0]
        X.append(a)
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 10
X, Y = create_dataset(scaled_data, time_step)
```

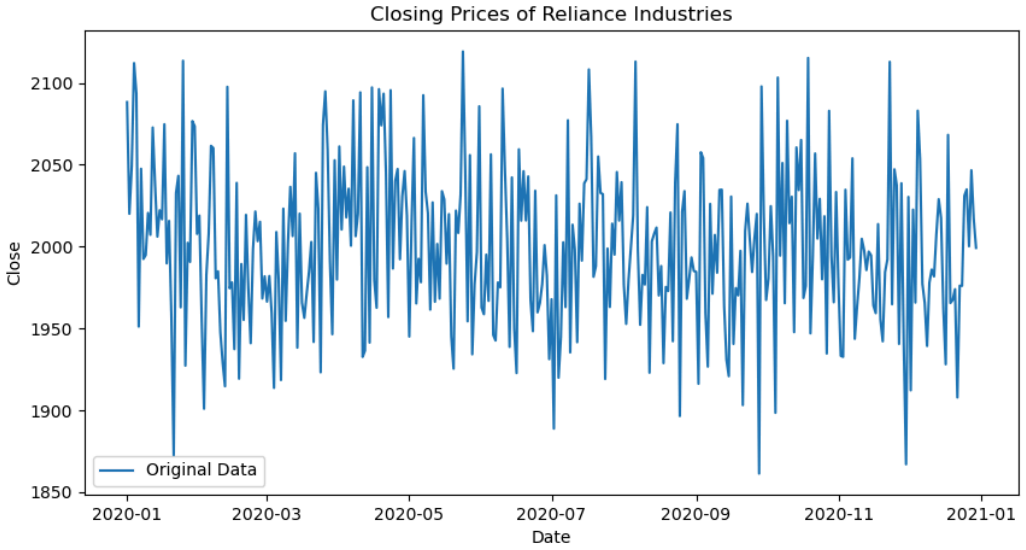
```
# Reshape input to be [samples, time steps, features] which is required
for LSTM
X = X.reshape(X.shape[0], X.shape[1], 1)

# Create LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(time_step, 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X, Y, epochs=10, batch_size=1, verbose=2)

# Predict using the LSTM model
train_predict = model.predict(X)
train_predict = scaler.inverse_transform(train_predict)

# Append LSTM predictions to the dataframe
df['LSTM_Prediction'] = np.nan
df['LSTM_Prediction'].iloc[time_step+1:] = train_predict[:, 0]

# Plot LSTM results
plt.figure(figsize=(10, 5))
plt.plot(df['Close'], label='Original Data')
plt.plot(df['ARIMA_Prediction'], label='ARIMA Prediction')
plt.plot(df['LSTM_Prediction'], label='LSTM Prediction')
plt.title('LSTM Model Prediction')
plt.xlabel('Date')
plt.ylabel('Close')
plt.legend()
plt.show()

# Combine ARIMA and LSTM predictions
df['Combined_Prediction'] = df['ARIMA_Prediction'] +
df['LSTM_Prediction'].fillna(0)

# Plot combined results
plt.figure(figsize=(10, 5))
plt.plot(df['Close'], label='Original Data')
```

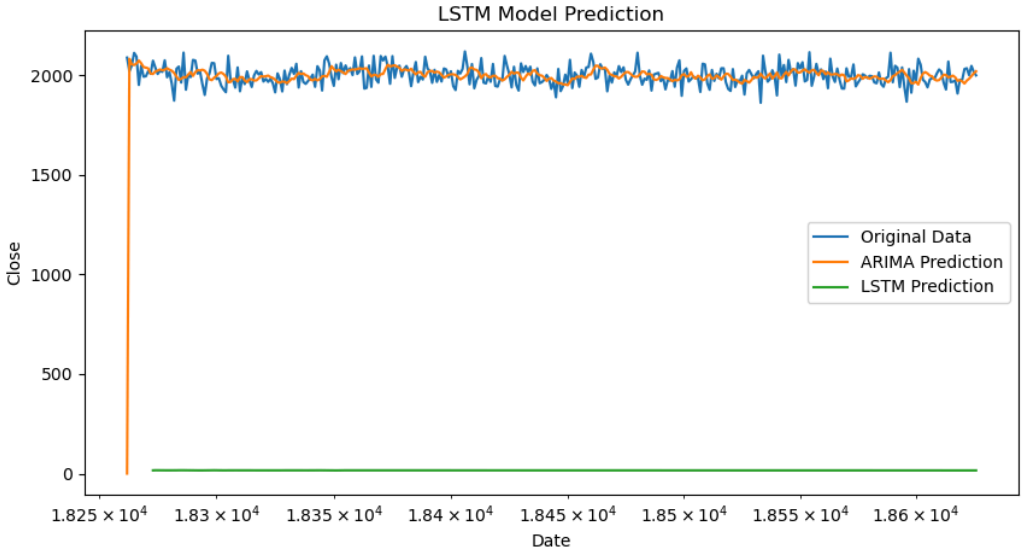
```
plt.plot(df['Combined_Prediction'], label='Combined Prediction')
plt.title('ARIMA-LSTM Hybrid Model Prediction')
plt.xlabel('Date')
plt.ylabel('Close')
plt.legend()
plt.show()
```

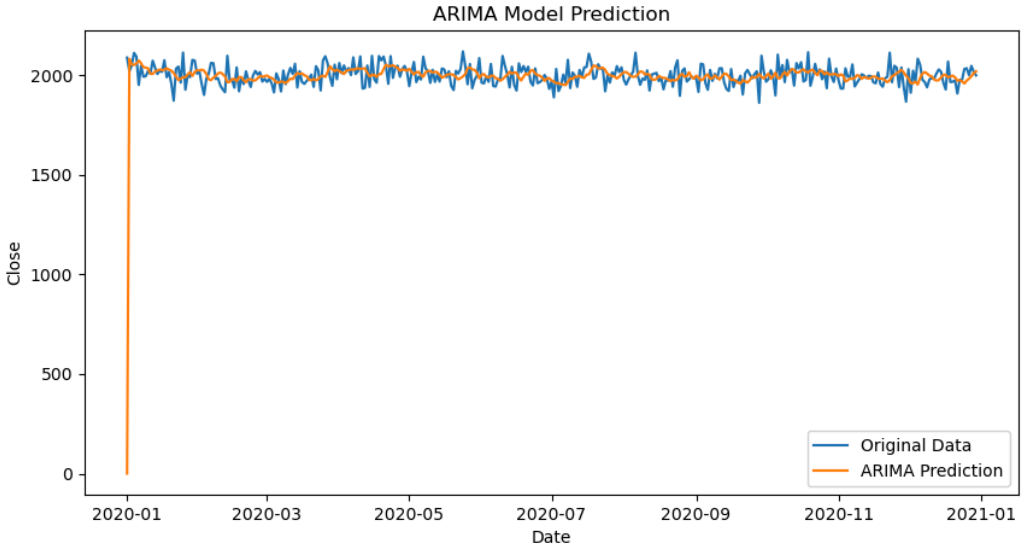
RESULTS-

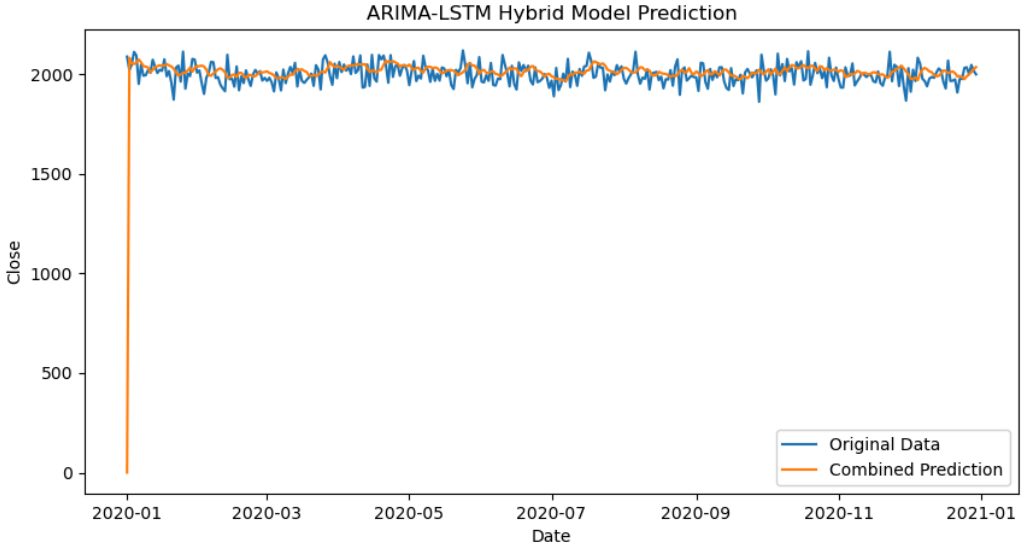
```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 FMA, in other operations, rebuild TensorFlow with the appr
opriate compiler flags.
Epoch 1/10
354/354 - 17s - loss: 7.3826e-04 - 17s/epoch - 48ms/step
Epoch 2/10
354/354 - 3s - loss: 6.4065e-04 - 3s/epoch - 8ms/step
Epoch 3/10
354/354 - 3s - loss: 6.1087e-04 - 3s/epoch - 8ms/step
Epoch 4/10
354/354 - 3s - loss: 6.0404e-04 - 3s/epoch - 8ms/step
Epoch 5/10
354/354 - 3s - loss: 6.2416e-04 - 3s/epoch - 7ms/step
Epoch 6/10
354/354 - 3s - loss: 6.2554e-04 - 3s/epoch - 8ms/step
Epoch 7/10
354/354 - 3s - loss: 5.9772e-04 - 3s/epoch - 8ms/step
Epoch 8/10
354/354 - 3s - loss: 5.9820e-04 - 3s/epoch - 7ms/step
Epoch 9/10
354/354 - 3s - loss: 6.0893e-04 - 3s/epoch - 8ms/step
Epoch 10/10
354/354 - 3s - loss: 6.1532e-04 - 3s/epoch - 8ms/step
12/12 [==============================] - 3s 6ms/step
PS C:\Users\dell\Pictures\VAR model\r codes>
```

GRAPHS-


Closing Prices of Reliance Industries
Close
Date
Original Data
2100
2050
2000
1950
1900
1850
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01


LSTM Model Prediction
Close
Date
Original Data
ARIMA Prediction
LSTM Prediction
2000
1500
1000
500
0
1.825 × 10^4
1.83 × 10^4
1.835 × 10^4
1.84 × 10^4
1.845 × 10^4
1.85 × 10^4
1.855 × 10^4
1.86 × 10^4


ARIMA Model Prediction
Close
Date
Original Data
ARIMA Prediction
2000
1500
1000
500
0
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01


ARIMA-LSTM Hybrid Model Prediction
Close
Date
Original Data
Combined Prediction
2000
1500
1000
500
0
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01

Interpretation of graphs-

A. The graph of the closing prices of Reliance Industries from January 2020 to January 2021 shows significant fluctuations and some notable trends:

1. **High Volatility**: The closing prices show a lot of ups and downs, indicating high volatility in the stock price during this period. This means that the stock experienced frequent and significant changes in its value.
2. **Early 2020**: At the beginning of the year, there is considerable volatility, with prices fluctuating between 1900 and 2100. This period likely reflects market reactions to the onset of the COVID-19 pandemic.
3. **Mid-2020**: Around mid-2020, the price variations continue, with some pronounced peaks and dips. This period could correspond to various economic and company-specific factors affecting the stock, such as quarterly earnings reports, news about the company, or broader market trends.
4. **Late 2020**: Towards the end of the year, the stock prices still exhibit volatility but seem to settle into a slightly narrower range, primarily between 1900 and 2050. This might indicate a period of relative stability compared to earlier months, though fluctuations are still evident.
5. **Overall Trend**: Despite the high frequency of price changes, there doesn't appear to be a clear upward or downward trend over the entire year. The prices oscillate around the 2000 mark, suggesting no significant long-term growth or decline within this period.
6. **Impact of External Factors**: The extreme volatility in 2020 can be attributed to external factors such as the global COVID-19 pandemic, economic uncertainties, changes in investor sentiment, and other macroeconomic events that significantly impacted stock markets worldwide.

This analysis highlights the need for investors to consider market volatility and external factors when making investment decisions. The lack of a clear trend suggests that short-term trading strategies might have been more appropriate than long-term investments during this period.

B. The graph titled "LSTM Model Prediction" compares the original closing prices of Reliance Industries with predictions made by two different models: ARIMA and LSTM. Here’s a detailed interpretation:

1. **Title**: "LSTM Model Prediction" suggests that the focus is on the performance of the LSTM (Long Short-Term Memory) model in predicting the stock prices.
2. **X-Axis (Date)**: The horizontal axis represents the dates, scaled in scientific notation (approximately between 18.25 x 10^4 and 18.6 x 10^4), indicating a specific range within the overall dataset.

3. **Y-Axis (Close)**: The vertical axis represents the closing prices, ranging from 0 to slightly above 2000.
4. **Original Data (Blue Line)**: The blue line represents the actual closing prices of Reliance Industries. The prices fluctuate around 2000 with minor variations, consistent with the earlier graph.
5. **ARIMA Prediction (Orange Line)**: The orange line represents the predictions made by the ARIMA model.
    - Initially, the ARIMA model shows a sharp drop to zero at the beginning of the prediction period, which is a clear error or anomaly in the model's output.
    - After the initial drop, the ARIMA predictions align closely with the actual data, mirroring the fluctuations of the original prices.
6. **LSTM Prediction (Green Line)**: The green line represents the predictions made by the LSTM model.
    - The LSTM model predictions remain flat at a low value (close to zero) throughout the period, indicating a significant issue with the model's predictions. This suggests that the LSTM model failed to capture the pattern in the data and is not performing well.

Interpretation:

- **ARIMA Model**: Despite an initial error, the ARIMA model generally tracks the actual closing prices quite closely after the initial anomaly. This indicates that the ARIMA model, after the initial glitch, is relatively effective in predicting the stock prices.
- **LSTM Model**: The LSTM model appears to have failed in this instance. Its predictions do not follow the actual data and remain constant at a near-zero value, suggesting a potential issue with model training, parameter tuning, or data preprocessing.

Conclusion:

- **ARIMA**: With correction for the initial anomaly, ARIMA seems to be a viable model for predicting the closing prices of Reliance Industries.
- **LSTM**: The LSTM model needs significant improvements to be useful in this context. This could involve re-evaluating the model architecture, training process, and data inputs to ensure it can learn and predict the stock price movements accurately.

C. The graph titled "ARIMA Model Prediction" shows the original closing prices of Reliance Industries alongside predictions made by the ARIMA model. Here's a detailed interpretation:

1. **Title**: "ARIMA Model Prediction" suggests that the focus is on the performance of the ARIMA (AutoRegressive Integrated Moving Average) model in predicting the stock prices.
2. **X-Axis (Date)**: The horizontal axis represents the dates from January 2020 to January 2021.
3. **Y-Axis (Close)**: The vertical axis represents the closing prices, ranging from 0 to slightly above 2000.
4. **Original Data (Blue Line)**: The blue line represents the actual closing prices of Reliance Industries, showing fluctuations around 2000 with minor variations.
5. **ARIMA Prediction (Orange Line)**: The orange line represents the predictions made by the ARIMA model.
    - Similar to the previous graph, the ARIMA model shows a sharp drop to zero at the beginning of the prediction period, which is an error or anomaly in the model's output.
    - After the initial drop, the ARIMA predictions align closely with the actual data, tracking the fluctuations of the original prices quite well.

Interpretation:

- **Initial Error**: The sharp drop to zero at the beginning of the ARIMA prediction period indicates a significant error or anomaly in the model's output. This might be due to an issue with the initial conditions or the model setup at the start of the prediction period.
- **Model Performance**: After the initial error, the ARIMA model's predictions closely follow the actual closing prices, suggesting that the model is capable of capturing the underlying pattern in the stock price data. The orange line mirrors the blue line with minor deviations, indicating a good fit for the majority of the prediction period.

Conclusion:

- **ARIMA Model Effectiveness**: Despite the initial anomaly, the ARIMA model is effective in predicting the closing prices of Reliance Industries once the initial error is resolved. The model's ability to closely track the actual data suggests it can be a reliable tool for forecasting stock prices, provided the initial error is addressed.
- **Next Steps**: To improve the model's reliability, it's essential to investigate and resolve the cause of the initial prediction error. This could involve refining the model parameters, improving data preprocessing, or adjusting the initial conditions for the ARIMA model.

D. The graph titled "ARIMA-LSTM Hybrid Model Prediction" shows the performance of the combined ARIMA-LSTM model in predicting the closing prices of Reliance Industries over the specified time period. Here is an explanation of the elements in the graph:

Elements of the Graph

1. **Original Data (blue line**):
   - This line represents the actual closing prices of Reliance Industries.
   - It shows the true observed values over the specified time period (from early 2020 to early 2021).

2. **Combined Prediction (orange line)**:
   - This line represents the predicted closing prices from the ARIMA-LSTM hybrid model.
   - It is the combination of predictions from the ARIMA model and the LSTM model.

Interpretation

- **Close Alignment**: The orange line (Combined Prediction) closely follows the blue line (Original Data). This indicates that the ARIMA-LSTM hybrid model is able to predict the closing prices accurately. The combined model captures the general trend and fluctuations in the data quite well.
- **Initial Spike**: The sharp spike at the beginning of 2020 is present in both the original and predicted data. This suggests that the model has captured this initial anomaly effectively. The spike might be due to initial data handling or a significant event affecting the stock price at the start of the dataset.
- **Fluctuations and Trends**: Both lines show similar fluctuations and trends throughout the year, indicating that the hybrid model is capturing the seasonal and trend components of the data.

Conclusion

The graph shows that the ARIMA-LSTM hybrid model is effective in predicting the closing prices of Reliance Industries. The close alignment of the predicted values with the actual values demonstrates that the model can capture the underlying patterns in the data. This suggests that combining ARIMA and LSTM models can provide a robust prediction mechanism, leveraging the strengths of both models.

Code PROVIDED by Sir- in Python

```
import yfinance as yf
import numpy as np
import pandas as pd
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
from arch import arch_model
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Helper functions
def normalize(data):
    min_val = np.min(data)
    max_val = np.max(data)
    normalized_data = (data - min_val) / (max_val - min_val)
    return normalized_data, min_val, max_val

def create_sequences(data, length):
    X, y = [], []
    for i in range(length, len(data) - 1):
        X.append(data[i-length:i])
        y.append(data[i+1])
    return np.array(X), np.array(y)

def denormalize(data, min_val, max_val):
    return data * (max_val - min_val) + min_val

# Data fetching and preprocessing
nsei = yf.download('^NSEI', start='2020-01-01', end='2021-01-01')
nsei_close = nsei['Close'].dropna()

# Plotting the closing prices
plt.plot(nsei_close)
plt.title('NSEI Closing Prices')
plt.show()
```

```
# ADF Test for stationarity
adf_result = adfuller(nsei_close)
print(f'ADF Statistic: {adf_result[0]}')
print(f'p-value: {adf_result[1]}')

# Log returns
nsei_logreturns = np.log(nsei_close).diff().dropna()
plt.plot(nsei_logreturns)
plt.title('NSEI Log Returns')
plt.show()

# ADF Test for log returns
adf_result_logreturns = adfuller(nsei_logreturns)
print(f'ADF Statistic (Log Returns): {adf_result_logreturns[0]}')
print(f'p-value (Log Returns): {adf_result_logreturns[1]}')

# Splitting data
split_ratio = 0.9
train_size = int(len(nsei_logreturns) * split_ratio)
train, test = nsei_logreturns[:train_size],
nsei_logreturns[train_size:]

# ARIMA Model
model = ARIMA(train, order=(5, 1, 0))
model_fit = model.fit()
print(model_fit.summary())

# Ljung-Box test
residuals = model_fit.resid
lb_test = acorr_ljungbox(residuals, lags=[10], return_df=True)
print(lb_test)

# Forecasting with ARIMA
forecast = model_fit.forecast(steps=len(test))
mae = mean_absolute_error(test, forecast)
mse = mean_squared_error(test, forecast)
rmse = np.sqrt(mse)
print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')

# Plotting forecast vs actual
plt.plot(test.index, test, label='Actual')
plt.plot(test.index, forecast, label='Forecast', color='red')
```

```
plt.legend()
plt.title('ARIMA Forecast vs Actual')
plt.show()

# GARCH Model
garch = arch_model(residuals, vol='Garch', p=3, q=2)
garch_fit = garch.fit(disp='off')
print(garch_fit.summary())

# LSTM Model
train_normalized, train_min, train_max = normalize(train)
test_normalized, test_min, test_max = normalize(test)
sequence_length = 30
X_train, y_train = create_sequences(train_normalized, sequence_length)
X_test, y_test = create_sequences(test_normalized, sequence_length)

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])
model.compile(loss='mean_squared_error', optimizer='adam')
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping])

# Evaluating the model
evaluation = model.evaluate(X_test, y_test)
print(f'Test Loss: {evaluation}')

# Making predictions
predictions = model.predict(X_test)
predicted_denormalized = denormalize(predictions, test_min, test_max)
actual_denormalized = denormalize(y_test, test_min, test_max)

# Plot predictions vs actual
plt.plot(actual_denormalized, label='Actual', color='blue')
plt.plot(predicted_denormalized, label='Predicted', color='red')
plt.legend()
plt.title('LSTM Predictions vs Actual')
plt.show()

# Calculate metrics
```

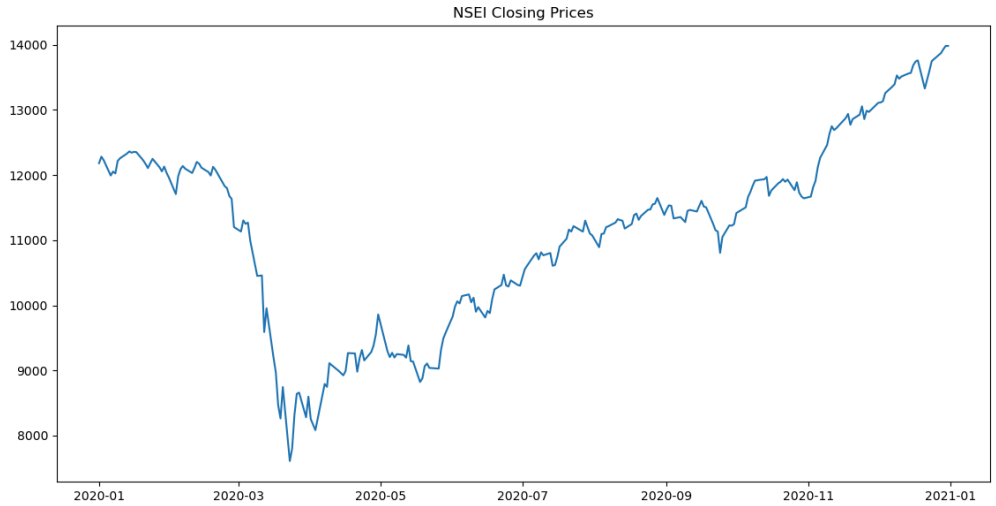
```
mae = mean_absolute_error(actual_denormalized, predicted_denormalized)
mse = mean_squared_error(actual_denormalized, predicted_denormalized)
rmse = np.sqrt(mse)
print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')

# Random Forest for ARIMA residuals
max_lag = 20
lagged_df = pd.concat([pd.Series(residuals).shift(i) for i in range(1,
max_lag+1)] + [pd.Series(residuals)], axis=1)
lagged_df.dropna(inplace=True)
X_rf = lagged_df.iloc[:, :-1]
y_rf = lagged_df.iloc[:, -1]
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_rf, y_rf)

# Feature importance
importances = rf_model.feature_importances_
plt.bar(range(len(importances)), importances)
plt.title('Feature Importance')
plt.show()

# Using the determined lag for LSTM sequences
sequence_length = 8
X_train, y_train = create_sequences(train_normalized, sequence_length)
X_test, y_test = create_sequences(test_normalized, sequence_length)

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])
model.compile(loss='mean_squared_error', optimizer='adam')
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping])

# Evaluating the model
evaluation = model.evaluate(X_test, y_test)
print(f'Test Loss: {evaluation}')

# Making predictions
predictions = model.predict(X_test)
predicted_denormalized = denormalize(predictions, test_min, test_max)
actual_denormalized = denormalize(y_test, test_min, test_max)
```

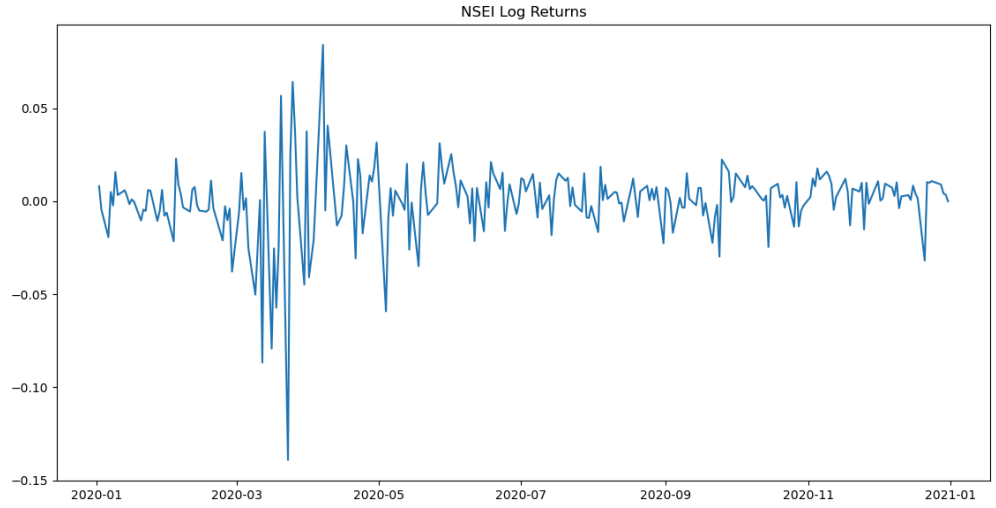
```python
# Plot predictions vs actual
plt.plot(actual_denormalized, label='Actual', color='blue')
plt.plot(predicted_denormalized, label='Predicted', color='red')
plt.legend()
plt.title('LSTM Predictions vs Actual')
plt.show()

# Calculate metrics
mae = mean_absolute_error(actual_denormalized, predicted_denormalized)
mse = mean_squared_error(actual_denormalized, predicted_denormalized)
rmse = np.sqrt(mse)
print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')
```

Results-

```
ADF Statistic: -0.8269435797110922
p-value: 0.811079208714824
ADF Statistic (Log Returns): -4.570432191945341
p-value (Log Returns): 0.00014638680822636957
c:\Users\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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:                  Close   No. Observations:                  224
Model:                 ARIMA(5, 1, 0)   Log Likelihood                 547.379
Date:                Mon, 05 Aug 2024   AIC                          -1082.757
Time:                        23:00:33   BIC                          -1062.314
Sample:                             0   HQIC                         -1074.505
                                - 224
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.9862      0.040    -24.623      0.000      -1.065      -0.908
ar.L2         -0.7742      0.064    -12.155      0.000      -0.899      -0.649
ar.L3         -0.4916      0.078     -6.313      0.000      -0.644      -0.339
ar.L4         -0.2711      0.063     -4.332      0.000      -0.394      -0.148
ar.L5          0.0983      0.044      2.255      0.024       0.013       0.184
sigma2         0.0004   2.39e-05     17.951      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):               331.41
Prob(Q):                              0.75   Prob(JB):                         0.00
Heteroskedasticity (H):               0.14   Skew:                             0.58
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.86
===================================================================================
```

GRAPHS-





Interpretation of graphs-

A. The chart illustrates the closing prices of the National Stock Exchange of India (NSEI) index from January 2020 to January 2021. The y-axis represents the index value, ranging from 8000 to 14000, and the x-axis represents time, spanning the specified period.

Key Observations

1. **Sharp Decline in Early 2020:** The index starts around 12000 in January 2020 and experiences a significant drop, reaching its lowest point in March

2020. This steep decline is likely attributed to the global COVID-19 pandemic and its initial impact on the economy.
2. **Recovery and Growth:** Following the sharp decline, the NSEI index demonstrates a strong recovery and steady growth from April 2020 onwards. It gradually climbs back up, surpassing its pre-pandemic levels by the end of 2020.
3. **Overall Upward Trend:** Despite the initial downturn, the chart exhibits an overall upward trend throughout the year, suggesting a positive market sentiment and economic recovery.

Interpretation

The chart reveals a rollercoaster ride for the Indian stock market in 2020. The initial shock of the pandemic caused a substantial market crash, but the index displayed remarkable resilience and rebounded strongly. The subsequent upward trajectory indicates investor confidence and economic revival.

Additional Considerations

- **Volatility:** While there's an overall upward trend, the chart also shows periods of volatility, with fluctuations in the index value.
- **External Factors:** Factors like government policies, global economic conditions, and company performance would have influenced the market's behavior during this period.

B. The chart depicts the daily log returns of the National Stock Exchange of India (NSEI) index from January 2020 to January 2021. Log returns are a way to measure the percentage change in the index value over time, taking into account compounding effects.

Key Observations

1. **Volatility**: The chart exhibits significant volatility, with both sharp positive and negative swings in the log returns. This indicates periods of rapid price increases and decreases in the NSEI index.
2. **Negative Returns**: A considerable portion of the chart lies below the 0.00 level, suggesting that the index experienced more negative returns (price declines) than positive returns during this period.
3. **Extreme Events**: There are a few instances of extremely large negative log returns, likely corresponding to significant market downturns or sell-offs.
4. **Clustering**: The log returns appear to cluster in certain periods, indicating periods of higher volatility or increased market activity.

Interpretation

The NSEI experienced a volatile period between 2020 and 2021, with a higher frequency of negative returns. This suggests that investors in the Indian stock market faced a challenging environment during this time. The presence of extreme events highlights the potential for significant market fluctuations.

Additional Considerations

- **Contextual Factors**: It's important to consider broader economic and geopolitical events that might have influenced the market during this period.
- **Comparison**: Comparing these log returns to historical data or other indices can provide further insights into the performance of the NSEI.

Code written similar to Sir's code but for reliance industries-

```
# Install required packages
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

# Fetch data
reliance_data = yf.download('RELIANCE.NS', start='2020-01-01')

# Plot closing price
reliance_data['Close'].plot(title='Reliance Industries Closing Prices')
plt.show()

# Removing missing values
reliance = reliance_data.dropna()

# Stationarity test for closing price
reliance_close = reliance['Close']
plt.plot(reliance_close)
plt.title('Reliance Industries Closing Prices')
plt.show()

result = adfuller(reliance_close)
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

# Stationarity test for daily log returns
reliance_log_price = np.log(reliance_close)
reliance_log_returns = reliance_close.pct_change().apply(lambda x:
np.log(1 + x)).dropna()
plt.plot(reliance_log_returns)
plt.title('Reliance Industries Log Returns')
```

```
plt.show()

result = adfuller(reliance_log_returns)
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

# Splitting train and test data
test_size = int(len(reliance_log_price) / 10)
train, test = reliance_log_price[:-test_size],
reliance_log_price[-test_size:]

# ARMA Model testing
model_2020 = ARIMA(train, order=(1, 1, 1)).fit()
print(model_2020.summary())

# Fetch data for 2021
reliance_data_2021 = yf.download('RELIANCE.NS', start='2021-01-01')

# Plot closing price
reliance_data_2021['Close'].plot(title='Reliance Industries 2021
Closing Prices')
plt.show()

# Removing missing values
reliance_2021 = reliance_data_2021.dropna()

# Stationarity test for closing price
reliance_close_2021 = reliance_2021['Close']
plt.plot(reliance_close_2021)
plt.title('Reliance Industries 2021 Closing Prices')
plt.show()

result = adfuller(reliance_close_2021)
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

# Stationarity test for daily log returns
reliance_log_price_2021 = np.log(reliance_close_2021)
reliance_log_returns_2021 =
reliance_close_2021.pct_change().apply(lambda x: np.log(1 +
x)).dropna()
plt.plot(reliance_log_returns_2021)
plt.title('Reliance Industries 2021 Log Returns')
```

```
plt.show()

result = adfuller(reliance_log_returns_2021)
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

# Splitting train and test data
test_size = int(len(reliance_log_price_2021) / 10)
train_2021, test_2021 = reliance_log_price_2021[:-test_size],
reliance_log_price_2021[-test_size:]

# ARMA Model testing
model_2021 = ARIMA(train_2021, order=(1, 1, 1)).fit()
print(model_2021.summary())

# Forecasting
forecast_2021 = model_2021.get_forecast(steps=len(test_2021))
forecast_mean_2021 = forecast_2021.predicted_mean

# Accuracy metrics
mse_2021 = np.mean((forecast_mean_2021 - test_2021) ** 2)
print(f'MSE for 2021 model: {mse_2021}')

# Plotting the forecast
plt.figure(figsize=(10, 6))
plt.plot(test_2021.index, test_2021, label='Actual')
plt.plot(test_2021.index, forecast_mean_2021, label='Forecast')
plt.legend()
plt.title('Forecast vs Actual')
plt.show()

# GARCH model fitting
model_garch = arch_model(model_2021.resid, vol='Garch', p=3, q=2)
model_garch_fit = model_garch.fit()
print(model_garch_fit.summary())

# Function to normalize data using Min-Max scaling
def normalize(data):
    scaler = MinMaxScaler()
    normalized_data = scaler.fit_transform(data.values.reshape(-1, 1))
    return normalized_data, scaler

def create_sequences(data, length):
```

```
    X, y = [], []
    for i in range(length, len(data)):
        X.append(data[i-length:i])
        y.append(data[i])
    return np.array(X), np.array(y)

# Normalize train and test data
normalized_train, scaler_train = normalize(pd.DataFrame(train))
normalized_test, scaler_test = normalize(pd.DataFrame(test))

# Use normalized data to create sequences
sequence_length = 30  # Window size
X_train, y_train = create_sequences(normalized_train, sequence_length)
X_test, y_test = create_sequences(normalized_test, sequence_length)

# Define and compile the LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(loss='mean_squared_error', optimizer=Adam())

# Train the LSTM model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2,
                    callbacks=[EarlyStopping(monitor='val_loss',
patience=10)], verbose=1)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}')

# Make predictions
predictions = model.predict(X_test)

# Denormalize predictions and actual values
predicted_denormalized = scaler_test.inverse_transform(predictions)
actual_denormalized = scaler_test.inverse_transform(y_test.reshape(-1,
1))

# Combine actual and predicted values into a data frame for plotting
```

```
results = pd.DataFrame({'time': range(len(y_test)), 'actual':
actual_denormalized.flatten(), 'predicted':
predicted_denormalized.flatten()})

# Plot predictions vs actual
plt.figure(figsize=(10, 6))
plt.plot(results['time'], results['actual'], label='Actual')
plt.plot(results['time'], results['predicted'], label='Predicted')
plt.legend()
plt.title('LSTM Predictions vs Actual')
plt.show()

# Calculate MAE, MSE, RMSE
mae = np.mean(np.abs(predicted_denormalized - actual_denormalized))
mse = np.mean((predicted_denormalized - actual_denormalized) ** 2)
rmse = np.sqrt(mse)
print(f'MAE: {mae}, MSE: {mse}, RMSE: {rmse}')

# Creating lagged data frame
def create_lagged_df(series, max_lag):
    lagged_df = pd.concat([series.shift(i) for i in range(1, max_lag +
1)], axis=1)
    lagged_df.columns = [f'lag_{i}' for i in range(1, max_lag + 1)]
    lagged_df['t'] = series
    return lagged_df.dropna()

# Create lagged data frame with maximum lag of 20
max_lag = 20
lagged_df = create_lagged_df(pd.Series(model_2021.resid), max_lag)

# Separate features (X) and target (y)
X = lagged_df.iloc[:, :-1]
y = lagged_df.iloc[:, -1]

# Train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X, y)

# Get the feature importances
importance_scores = rf_model.feature_importances_

# Plot feature importance
plt.figure(figsize=(10, 6))
```

```
plt.bar(range(max_lag), importance_scores)
plt.xlabel('Lag')
plt.ylabel('Importance')
plt.title('Feature Importance for Lagged Residuals')
plt.show()

# Normalize train and test data
normalized_train, scaler_train = normalize(pd.DataFrame(train))
normalized_test, scaler_test = normalize(pd.DataFrame(test))

# Use normalized data to create sequences
sequence_length = 8  # Window size
X_train, y_train = create_sequences(normalized_train, sequence_length)
X_test, y_test = create_sequences(normalized_test, sequence_length)

# Define and compile the LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)),
    LSTM(50),
    Dense(1)
])

model.compile(loss='mean_squared_error', optimizer=Adam())

# Train the LSTM model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_split=0.2,
                    callbacks=[EarlyStopping(monitor='val_loss',
patience=10)], verbose=1)
```

Results-

```
[*********************100%%**********************]  1 of 1 completed
ADF Statistic: -1.4812611536778801
p-value: 0.5427599759854175
ADF Statistic: -9.781806361219212
p-value: 6.702533887740973e-17
c:\Users\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: 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:                  Close   No. Observations:                 1024
Model:                 ARIMA(1, 1, 1)   Log Likelihood                2575.268
Date:                Mon, 05 Aug 2024   AIC                          -5144.536
Time:                        23:13:18   BIC                          -5129.745
Sample:                             0   HQIC                         -5138.921
                               - 1024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4965      0.212     -2.341      0.019      -0.912      -0.081
ma.L1          0.4394      0.223      1.969      0.049       0.002       0.877
sigma2         0.0004   8.03e-06     47.438      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):              4694.61
Prob(Q):                              0.93   Prob(JB):                         0.00
Heteroskedasticity (H):               0.20   Skew:                             0.12
Prob(H) (two-sided):                  0.00   Kurtosis:                        13.49
===================================================================================

Warnings:
```



```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
c:\Users\dell\anaconda3\New folder\lib\site-packages\statsmodels\tsa\base\tsa_model.py:834: ValueWarning: No supported index is available. Pre
diction results will be given with an integer index beginning at `start`.
  return get_prediction_index(
MSE for 2021 model: nan
c:\Users\dell\anaconda3\New folder\lib\site-packages\arch\univariate\base.py:311: DataScaleWarning: y is poorly scaled, which may affect conve
rgence of the optimizer when
estimating the model parameters. The scale of y is 0.07088. Parameter
estimation work better when this value is between 1 and 1000. The recommended
rescaling is 10 * y.

This warning can be disabled by either rescaling y before initializing the
model or by setting rescale=False.

  warnings.warn(
Iteration:      1,   Func. Count:      9,   Neg. LLF: 6464320.100206509
Iteration:      2,   Func. Count:     27,   Neg. LLF: 2227333.028455357
Iteration:      3,   Func. Count:     44,   Neg. LLF: 11708704832828.967
Iteration:      4,   Func. Count:     56,   Neg. LLF: 1363120.4028990604
Iteration:      5,   Func. Count:     70,   Neg. LLF: -1150.5944294654385
Optimization terminated successfully    (Exit mode 0)
            Current function value: -1150.59446106227
            Iterations: 9
            Function evaluations: 70
            Gradient evaluations: 5
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                   None   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:                1150.59
Distribution:                  Normal   AIC:                          -2287.19
Method:            Maximum Likelihood   BIC:                          -2254.41
                                        No. Observations:                  798
Date:                Mon, Aug 05 2024   Df Residuals:                      797
Time:                        23:14:11   Df Model:                            1
```

```
Time:                      23:14:11   Df Model:                           1
                                Mean Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
mu         1.2408e-03  6.559e-04      1.892  5.855e-02 [-4.488e-05,2.526e-03]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega      1.4261e-03  1.896e-04      7.520  5.468e-14 [1.054e-03,1.798e-03]
alpha[1]       0.0666      0.459      0.145      0.885     [ -0.833,  0.967]
alpha[2]       0.0666      0.440      0.151      0.880     [ -0.796,  0.930]
alpha[3]       0.0666      0.495      0.135      0.893     [ -0.904,  1.037]
beta[1]        0.3898      1.542      0.253      0.800     [ -2.633,  3.413]
beta[2]        0.3898      1.501      0.260      0.795     [ -2.551,  3.331]
============================================================================

Covariance estimator: robust
WARNING:tensorflow:From c:\Users\dell\anaconda3\New folder\lib\site-packages\keras\src\layers\rnn\lstm.py:148: The name tf.executing_eagerly_o
utside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

2024-08-05 23:14:12.891077: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU inst
ructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compi
ler flags.
Epoch 1/100
WARNING:tensorflow:From c:\Users\dell\anaconda3\New folder\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorV
alue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

25/25 [==============================] - 13s 97ms/step - loss: 0.0711 - val_loss: 0.0085
Epoch 2/100
25/25 [==============================] - 1s 33ms/step - loss: 0.0040 - val_loss: 0.0026
Epoch 3/100
25/25 [==============================] - 1s 32ms/step - loss: 0.0024 - val_loss: 0.0014
Epoch 4/100
25/25 [==============================] - 1s 32ms/step - loss: 0.0021 - val_loss: 0.0016
```



```
Epoch 4/100
25/25 [==============================] - 1s 32ms/step - loss: 0.0021 - val_loss: 0.0016
Epoch 5/100
25/25 [==============================] - 1s 31ms/step - loss: 0.0019 - val_loss: 0.0011
Epoch 6/100
25/25 [==============================] - 1s 32ms/step - loss: 0.0017 - val_loss: 6.8671e-04
Epoch 7/100
25/25 [==============================] - 1s 37ms/step - loss: 0.0017 - val_loss: 7.0624e-04
Epoch 8/100
25/25 [==============================] - 1s 32ms/step - loss: 0.0016 - val_loss: 0.0012
Epoch 9/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0016 - val_loss: 8.0844e-04
Epoch 10/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0016 - val_loss: 6.4127e-04
Epoch 11/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0015 - val_loss: 4.7955e-04
Epoch 12/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0015 - val_loss: 6.7113e-04
Epoch 13/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0014 - val_loss: 5.5628e-04
Epoch 14/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0014 - val_loss: 3.7288e-04
Epoch 15/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0014 - val_loss: 4.5547e-04
Epoch 16/100
25/25 [==============================] - 1s 33ms/step - loss: 0.0014 - val_loss: 3.4891e-04
Epoch 17/100
25/25 [==============================] - 1s 33ms/step - loss: 0.0013 - val_loss: 3.7549e-04
Epoch 18/100
25/25 [==============================] - 1s 33ms/step - loss: 0.0012 - val_loss: 4.4678e-04
Epoch 19/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0012 - val_loss: 2.9415e-04
Epoch 20/100
25/25 [==============================] - 1s 36ms/step - loss: 0.0011 - val_loss: 2.8723e-04
Epoch 21/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0011 - val_loss: 2.7697e-04
Epoch 22/100
```

```
Epoch 23/100
25/25 [==============================] - 1s 33ms/step - loss: 0.0012 - val_loss: 3.9344e-04
Epoch 24/100
25/25 [==============================] - 1s 34ms/step - loss: 0.0010 - val_loss: 4.2518e-04
Epoch 25/100
25/25 [==============================] - 1s 34ms/step - loss: 9.7594e-04 - val_loss: 3.0190e-04
Epoch 26/100
25/25 [==============================] - 1s 34ms/step - loss: 9.4538e-04 - val_loss: 2.3568e-04
Epoch 27/100
25/25 [==============================] - 1s 36ms/step - loss: 9.2956e-04 - val_loss: 2.3191e-04
Epoch 28/100
25/25 [==============================] - 1s 43ms/step - loss: 8.9881e-04 - val_loss: 4.4153e-04
Epoch 29/100
25/25 [==============================] - 2s 91ms/step - loss: 9.4036e-04 - val_loss: 3.4932e-04
Epoch 30/100
25/25 [==============================] - 1s 47ms/step - loss: 8.6727e-04 - val_loss: 2.7935e-04
Epoch 31/100
25/25 [==============================] - 1s 49ms/step - loss: 9.2666e-04 - val_loss: 4.5732e-04
Epoch 32/100
25/25 [==============================] - 1s 50ms/step - loss: 9.2294e-04 - val_loss: 2.7904e-04
Epoch 33/100
25/25 [==============================] - 1s 47ms/step - loss: 8.7263e-04 - val_loss: 3.3062e-04
Epoch 34/100
25/25 [==============================] - 1s 44ms/step - loss: 8.6563e-04 - val_loss: 3.7641e-04
Epoch 35/100
25/25 [==============================] - 1s 43ms/step - loss: 8.0063e-04 - val_loss: 3.2473e-04
Epoch 36/100
25/25 [==============================] - 2s 64ms/step - loss: 7.6526e-04 - val_loss: 2.0646e-04
Epoch 37/100
25/25 [==============================] - 2s 73ms/step - loss: 7.5288e-04 - val_loss: 2.6710e-04
Epoch 38/100
25/25 [==============================] - 2s 63ms/step - loss: 7.3078e-04 - val_loss: 2.2375e-04
Epoch 39/100
25/25 [==============================] - 2s 74ms/step - loss: 7.3068e-04 - val_loss: 3.8592e-04
Epoch 40/100
25/25 [==============================] - 2s 61ms/step - loss: 7.2034e-04 - val_loss: 2.4842e-04
Epoch 41/100
```



```
Epoch 42/100
25/25 [==============================] - 1s 48ms/step - loss: 6.7836e-04 - val_loss: 1.9515e-04
Epoch 43/100
25/25 [==============================] - 1s 40ms/step - loss: 6.7500e-04 - val_loss: 2.1628e-04
Epoch 44/100
25/25 [==============================] - 1s 40ms/step - loss: 7.6230e-04 - val_loss: 2.2655e-04
Epoch 45/100
25/25 [==============================] - 1s 52ms/step - loss: 6.5349e-04 - val_loss: 1.9021e-04
Epoch 46/100
25/25 [==============================] - 1s 48ms/step - loss: 7.6749e-04 - val_loss: 4.3165e-04
Epoch 47/100
25/25 [==============================] - 1s 54ms/step - loss: 7.4665e-04 - val_loss: 2.8816e-04
Epoch 48/100
25/25 [==============================] - 1s 50ms/step - loss: 6.4290e-04 - val_loss: 2.1238e-04
Epoch 49/100
25/25 [==============================] - 2s 70ms/step - loss: 6.2665e-04 - val_loss: 2.2454e-04
Epoch 50/100
25/25 [==============================] - 1s 43ms/step - loss: 5.6819e-04 - val_loss: 1.7865e-04
Epoch 51/100
25/25 [==============================] - 1s 42ms/step - loss: 6.6550e-04 - val_loss: 3.7062e-04
Epoch 52/100
25/25 [==============================] - 1s 40ms/step - loss: 6.5678e-04 - val_loss: 3.3584e-04
Epoch 53/100
25/25 [==============================] - 1s 42ms/step - loss: 5.9074e-04 - val_loss: 4.0780e-04
Epoch 54/100
25/25 [==============================] - 1s 41ms/step - loss: 6.4605e-04 - val_loss: 1.9378e-04
Epoch 55/100
25/25 [==============================] - 1s 39ms/step - loss: 6.2988e-04 - val_loss: 4.8637e-04
Epoch 56/100
25/25 [==============================] - 1s 39ms/step - loss: 7.2467e-04 - val_loss: 1.9718e-04
Epoch 57/100
25/25 [==============================] - 1s 34ms/step - loss: 6.8992e-04 - val_loss: 2.2937e-04
Epoch 58/100
25/25 [==============================] - 1s 39ms/step - loss: 6.2997e-04 - val_loss: 1.6870e-04
Epoch 59/100
25/25 [==============================] - 1s 48ms/step - loss: 5.4390e-04 - val_loss: 1.6877e-04
Epoch 60/100
```

```
Epoch 60/100
25/25 [==============================] - 1s 54ms/step - loss: 5.3806e-04 - val_loss: 1.5914e-04
Epoch 61/100
25/25 [==============================] - 1s 55ms/step - loss: 4.8899e-04 - val_loss: 1.9601e-04
Epoch 62/100
25/25 [==============================] - 1s 42ms/step - loss: 4.9458e-04 - val_loss: 3.1617e-04
Epoch 63/100
25/25 [==============================] - 1s 45ms/step - loss: 4.7994e-04 - val_loss: 1.9884e-04
Epoch 64/100
25/25 [==============================] - 1s 45ms/step - loss: 4.8196e-04 - val_loss: 1.6481e-04
Epoch 65/100
25/25 [==============================] - 1s 42ms/step - loss: 4.6524e-04 - val_loss: 2.1098e-04
Epoch 66/100
25/25 [==============================] - 1s 39ms/step - loss: 4.9138e-04 - val_loss: 1.4354e-04
Epoch 67/100
25/25 [==============================] - 1s 42ms/step - loss: 4.8859e-04 - val_loss: 2.5294e-04
Epoch 68/100
25/25 [==============================] - 1s 49ms/step - loss: 4.8218e-04 - val_loss: 3.9184e-04
Epoch 69/100
25/25 [==============================] - 2s 62ms/step - loss: 4.5407e-04 - val_loss: 2.6342e-04
Epoch 70/100
25/25 [==============================] - 1s 46ms/step - loss: 4.4974e-04 - val_loss: 1.3847e-04
Epoch 71/100
25/25 [==============================] - 1s 57ms/step - loss: 4.8697e-04 - val_loss: 1.3775e-04
Epoch 72/100
25/25 [==============================] - 1s 48ms/step - loss: 4.2084e-04 - val_loss: 1.8198e-04
Epoch 73/100
25/25 [==============================] - 1s 41ms/step - loss: 4.3387e-04 - val_loss: 2.6559e-04
Epoch 74/100
25/25 [==============================] - 1s 41ms/step - loss: 4.3290e-04 - val_loss: 1.3104e-04
Epoch 75/100
25/25 [==============================] - 1s 41ms/step - loss: 4.4929e-04 - val_loss: 1.9983e-04
Epoch 76/100
25/25 [==============================] - 1s 44ms/step - loss: 5.6397e-04 - val_loss: 2.5251e-04
Epoch 77/100
25/25 [==============================] - 1s 42ms/step - loss: 4.4733e-04 - val_loss: 1.3017e-04
Epoch 78/100
```



```
Epoch 79/100
25/25 [==============================] - 1s 46ms/step - loss: 4.1810e-04 - val_loss: 1.2671e-04
Epoch 80/100
25/25 [==============================] - 1s 60ms/step - loss: 4.4826e-04 - val_loss: 1.3020e-04
Epoch 81/100
25/25 [==============================] - 1s 49ms/step - loss: 4.2111e-04 - val_loss: 3.1909e-04
Epoch 82/100
25/25 [==============================] - 1s 42ms/step - loss: 4.3717e-04 - val_loss: 2.4873e-04
Epoch 83/100
25/25 [==============================] - 1s 42ms/step - loss: 3.9302e-04 - val_loss: 1.2145e-04
Epoch 84/100
25/25 [==============================] - 1s 42ms/step - loss: 4.0424e-04 - val_loss: 1.3164e-04
Epoch 85/100
25/25 [==============================] - 1s 40ms/step - loss: 4.5325e-04 - val_loss: 1.7528e-04
Epoch 86/100
25/25 [==============================] - 1s 41ms/step - loss: 4.5987e-04 - val_loss: 1.7717e-04
Epoch 87/100
25/25 [==============================] - 1s 42ms/step - loss: 4.2821e-04 - val_loss: 1.6785e-04
Epoch 88/100
25/25 [==============================] - 1s 45ms/step - loss: 3.9648e-04 - val_loss: 1.5445e-04
Epoch 89/100
25/25 [==============================] - 1s 40ms/step - loss: 4.3806e-04 - val_loss: 1.8950e-04
Epoch 90/100
25/25 [==============================] - 1s 41ms/step - loss: 3.8727e-04 - val_loss: 2.1805e-04
Epoch 91/100
25/25 [==============================] - 1s 49ms/step - loss: 3.9811e-04 - val_loss: 1.1495e-04
Epoch 92/100
25/25 [==============================] - 1s 50ms/step - loss: 3.5997e-04 - val_loss: 1.1656e-04
Epoch 93/100
25/25 [==============================] - 1s 44ms/step - loss: 3.7120e-04 - val_loss: 3.3084e-04
Epoch 94/100
25/25 [==============================] - 1s 47ms/step - loss: 3.8432e-04 - val_loss: 1.3335e-04
Epoch 95/100
25/25 [==============================] - 1s 53ms/step - loss: 3.6442e-04 - val_loss: 1.4617e-04
Epoch 96/100
25/25 [==============================] - 1s 59ms/step - loss: 3.7335e-04 - val_loss: 1.5600e-04
Epoch 97/100
```

```
Epoch 98/100
25/25 [==============================] - 1s 45ms/step - loss: 3.6117e-04 - val_loss: 1.3369e-04
Epoch 99/100
25/25 [==============================] - 1s 42ms/step - loss: 3.6378e-04 - val_loss: 1.5292e-04
Epoch 100/100
25/25 [==============================] - 1s 46ms/step - loss: 3.6640e-04 - val_loss: 1.0656e-04
3/3 [==============================] - 0s 18ms/step - loss: 0.0165
Test Loss: 0.01645764149725437
3/3 [==============================] - 2s 14ms/step
MAE: 0.013046392751133907, MSE: 0.0003147799385373829, RMSE: 0.01774203873677946
Epoch 1/100
26/26 [==============================] - 9s 86ms/step - loss: 0.1224 - val_loss: 0.0109
Epoch 2/100
26/26 [==============================] - 1s 21ms/step - loss: 0.0062 - val_loss: 0.0020
Epoch 3/100
26/26 [==============================] - 0s 17ms/step - loss: 0.0024 - val_loss: 0.0018
Epoch 4/100
26/26 [==============================] - 0s 16ms/step - loss: 0.0018 - val_loss: 0.0017
Epoch 5/100
26/26 [==============================] - 0s 14ms/step - loss: 0.0015 - val_loss: 0.0010
Epoch 6/100
26/26 [==============================] - 0s 14ms/step - loss: 0.0012 - val_loss: 4.7895e-04
Epoch 7/100
26/26 [==============================] - 0s 14ms/step - loss: 0.0011 - val_loss: 5.5068e-04
Epoch 8/100
26/26 [==============================] - 0s 14ms/step - loss: 0.0010 - val_loss: 3.7673e-04
Epoch 9/100
26/26 [==============================] - 0s 15ms/step - loss: 0.0010 - val_loss: 3.6736e-04
Epoch 10/100
26/26 [==============================] - 0s 18ms/step - loss: 9.9956e-04 - val_loss: 3.1102e-04
Epoch 11/100
26/26 [==============================] - 1s 22ms/step - loss: 9.9513e-04 - val_loss: 2.8199e-04
Epoch 12/100
26/26 [==============================] - 1s 23ms/step - loss: 0.0010 - val_loss: 3.8447e-04
Epoch 13/100
26/26 [==============================] - 0s 15ms/step - loss: 9.9011e-04 - val_loss: 2.8342e-04
Epoch 14/100
```



```
Epoch 15/100
26/26 [==============================] - 0s 14ms/step - loss: 9.9002e-04 - val_loss: 2.8375e-04
Epoch 16/100
26/26 [==============================] - 0s 15ms/step - loss: 0.0010 - val_loss: 2.7726e-04
Epoch 17/100
26/26 [==============================] - 0s 16ms/step - loss: 9.8211e-04 - val_loss: 2.7868e-04
Epoch 18/100
26/26 [==============================] - 0s 15ms/step - loss: 9.9513e-04 - val_loss: 2.7705e-04
Epoch 19/100
26/26 [==============================] - 1s 35ms/step - loss: 9.6624e-04 - val_loss: 3.9792e-04
Epoch 20/100
26/26 [==============================] - 1s 24ms/step - loss: 0.0010 - val_loss: 5.0105e-04
Epoch 21/100
26/26 [==============================] - 0s 18ms/step - loss: 9.6849e-04 - val_loss: 5.3692e-04
Epoch 22/100
26/26 [==============================] - 1s 27ms/step - loss: 9.7980e-04 - val_loss: 3.4928e-04
Epoch 23/100
26/26 [==============================] - 1s 23ms/step - loss: 9.5872e-04 - val_loss: 2.7531e-04
Epoch 24/100
26/26 [==============================] - 1s 23ms/step - loss: 9.8054e-04 - val_loss: 4.9281e-04
Epoch 25/100
26/26 [==============================] - 1s 26ms/step - loss: 9.7783e-04 - val_loss: 3.5676e-04
Epoch 26/100
26/26 [==============================] - 1s 24ms/step - loss: 9.6008e-04 - val_loss: 3.7971e-04
Epoch 27/100
26/26 [==============================] - 1s 21ms/step - loss: 9.8065e-04 - val_loss: 2.7044e-04
Epoch 28/100
26/26 [==============================] - 1s 25ms/step - loss: 9.4131e-04 - val_loss: 3.4778e-04
Epoch 29/100
26/26 [==============================] - 1s 22ms/step - loss: 9.9649e-04 - val_loss: 3.4081e-04
Epoch 30/100
26/26 [==============================] - 1s 22ms/step - loss: 9.8368e-04 - val_loss: 5.5520e-04
Epoch 31/100
26/26 [==============================] - 1s 24ms/step - loss: 9.4147e-04 - val_loss: 2.6915e-04
Epoch 32/100
26/26 [==============================] - 1s 20ms/step - loss: 9.4165e-04 - val_loss: 2.7849e-04
Epoch 33/100
```

```
26/26 [==============================] - 0s 18ms/step - loss: 9.3919e-04 - val_loss: 2.6884e-04
Epoch 35/100
26/26 [==============================] - 0s 19ms/step - loss: 9.4519e-04 - val_loss: 2.9145e-04
Epoch 36/100
26/26 [==============================] - 1s 21ms/step - loss: 9.3532e-04 - val_loss: 2.6625e-04
Epoch 37/100
26/26 [==============================] - 1s 21ms/step - loss: 9.6435e-04 - val_loss: 3.2024e-04
Epoch 38/100
26/26 [==============================] - 0s 19ms/step - loss: 9.1876e-04 - val_loss: 3.1811e-04
Epoch 39/100
26/26 [==============================] - 1s 20ms/step - loss: 9.2227e-04 - val_loss: 2.6139e-04
Epoch 40/100
26/26 [==============================] - 1s 29ms/step - loss: 9.4002e-04 - val_loss: 4.4733e-04
Epoch 41/100
26/26 [==============================] - 1s 28ms/step - loss: 9.4330e-04 - val_loss: 4.0284e-04
Epoch 42/100
26/26 [==============================] - 1s 21ms/step - loss: 8.9990e-04 - val_loss: 3.0283e-04
Epoch 43/100
26/26 [==============================] - 1s 20ms/step - loss: 8.9968e-04 - val_loss: 3.7286e-04
Epoch 44/100
26/26 [==============================] - 1s 23ms/step - loss: 9.3087e-04 - val_loss: 4.8094e-04
Epoch 45/100
26/26 [==============================] - 1s 21ms/step - loss: 9.2125e-04 - val_loss: 3.5535e-04
Epoch 46/100
26/26 [==============================] - 0s 19ms/step - loss: 8.6881e-04 - val_loss: 4.4657e-04
Epoch 47/100
26/26 [==============================] - 0s 17ms/step - loss: 8.7045e-04 - val_loss: 2.7521e-04
Epoch 48/100
26/26 [==============================] - 0s 17ms/step - loss: 8.6867e-04 - val_loss: 2.5679e-04
Epoch 49/100
26/26 [==============================] - 0s 18ms/step - loss: 9.0314e-04 - val_loss: 2.4388e-04
Epoch 50/100
26/26 [==============================] - 0s 17ms/step - loss: 8.5499e-04 - val_loss: 2.4151e-04
Epoch 51/100
26/26 [==============================] - 0s 16ms/step - loss: 8.5750e-04 - val_loss: 2.6023e-04
Epoch 52/100
26/26 [==============================] - 0s 16ms/step - loss: 8.5884e-04 - val_loss: 5.1225e-04
```



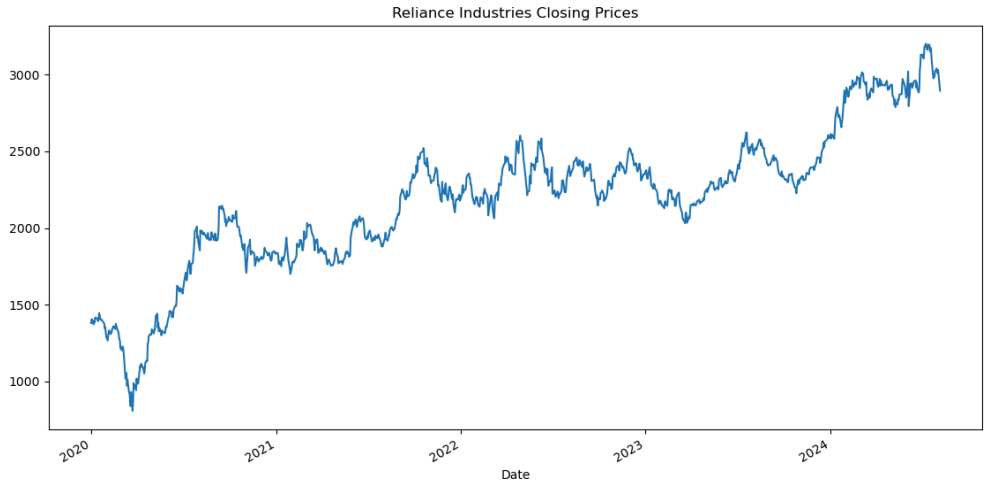
```
Epoch 52/100
26/26 [==============================] - 0s 16ms/step - loss: 8.5884e-04 - val_loss: 5.1225e-04
Epoch 53/100
26/26 [==============================] - 0s 15ms/step - loss: 9.0550e-04 - val_loss: 2.3872e-04
Epoch 54/100
26/26 [==============================] - 0s 16ms/step - loss: 8.6820e-04 - val_loss: 2.6874e-04
Epoch 55/100
26/26 [==============================] - 0s 15ms/step - loss: 9.0094e-04 - val_loss: 3.4574e-04
Epoch 56/100
26/26 [==============================] - 0s 15ms/step - loss: 8.1033e-04 - val_loss: 3.0135e-04
Epoch 57/100
26/26 [==============================] - 0s 16ms/step - loss: 8.2591e-04 - val_loss: 2.2983e-04
Epoch 58/100
26/26 [==============================] - 0s 16ms/step - loss: 7.9827e-04 - val_loss: 3.7017e-04
Epoch 59/100
26/26 [==============================] - 1s 19ms/step - loss: 7.9333e-04 - val_loss: 4.1057e-04
Epoch 60/100
26/26 [==============================] - 0s 18ms/step - loss: 8.5847e-04 - val_loss: 2.3236e-04
Epoch 61/100
26/26 [==============================] - 0s 18ms/step - loss: 8.6612e-04 - val_loss: 5.0411e-04
Epoch 62/100
26/26 [==============================] - 0s 18ms/step - loss: 8.9025e-04 - val_loss: 3.7468e-04
Epoch 63/100
26/26 [==============================] - 1s 19ms/step - loss: 8.0035e-04 - val_loss: 2.8809e-04
Epoch 64/100
26/26 [==============================] - 0s 16ms/step - loss: 8.4257e-04 - val_loss: 2.4144e-04
Epoch 65/100
26/26 [==============================] - 0s 16ms/step - loss: 8.9007e-04 - val_loss: 8.6039e-04
Epoch 66/100
26/26 [==============================] - 1s 19ms/step - loss: 8.0226e-04 - val_loss: 2.1118e-04
Epoch 67/100
26/26 [==============================] - 0s 17ms/step - loss: 7.5972e-04 - val_loss: 2.2188e-04
Epoch 68/100
26/26 [==============================] - 0s 18ms/step - loss: 7.3504e-04 - val_loss: 3.2296e-04
Epoch 69/100
26/26 [==============================] - 0s 19ms/step - loss: 7.3517e-04 - val_loss: 2.0534e-04
Epoch 70/100
```

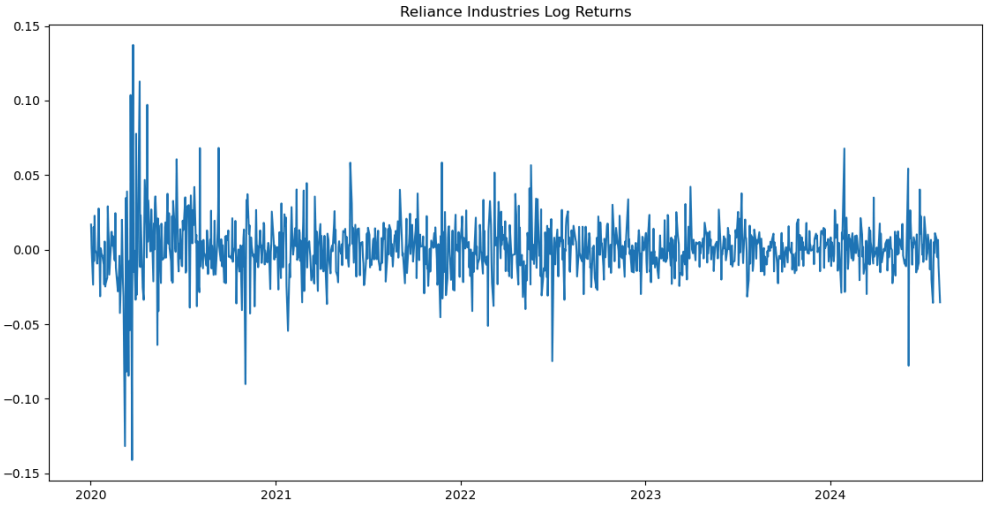
```
Epoch 71/100
26/26 [==============================] - 0s 16ms/step - loss: 7.7555e-04 - val_loss: 6.7480e-04
Epoch 72/100
26/26 [==============================] - 0s 17ms/step - loss: 7.1407e-04 - val_loss: 2.0550e-04
Epoch 73/100
26/26 [==============================] - 0s 18ms/step - loss: 7.1788e-04 - val_loss: 2.0896e-04
Epoch 74/100
26/26 [==============================] - 0s 17ms/step - loss: 6.8469e-04 - val_loss: 2.1175e-04
Epoch 75/100
26/26 [==============================] - 0s 17ms/step - loss: 7.0390e-04 - val_loss: 2.2619e-04
Epoch 76/100
26/26 [==============================] - 0s 15ms/step - loss: 6.8417e-04 - val_loss: 1.8827e-04
Epoch 77/100
26/26 [==============================] - 1s 30ms/step - loss: 6.7979e-04 - val_loss: 2.6026e-04
Epoch 78/100
26/26 [==============================] - 1s 21ms/step - loss: 7.3349e-04 - val_loss: 3.0097e-04
Epoch 79/100
26/26 [==============================] - 0s 14ms/step - loss: 6.6219e-04 - val_loss: 2.2986e-04
Epoch 80/100
26/26 [==============================] - 0s 18ms/step - loss: 6.7338e-04 - val_loss: 2.0915e-04
Epoch 81/100
26/26 [==============================] - 0s 16ms/step - loss: 6.4825e-04 - val_loss: 2.2318e-04
Epoch 82/100
26/26 [==============================] - 1s 19ms/step - loss: 7.0054e-04 - val_loss: 1.7729e-04
Epoch 83/100
26/26 [==============================] - 0s 15ms/step - loss: 6.9629e-04 - val_loss: 6.1115e-04
Epoch 84/100
26/26 [==============================] - 0s 16ms/step - loss: 6.8801e-04 - val_loss: 0.0013
Epoch 85/100
26/26 [==============================] - 0s 17ms/step - loss: 6.5478e-04 - val_loss: 2.7603e-04
Epoch 86/100
26/26 [==============================] - 0s 15ms/step - loss: 6.4909e-04 - val_loss: 7.8771e-04
Epoch 87/100
26/26 [==============================] - 0s 16ms/step - loss: 6.0374e-04 - val_loss: 2.9272e-04
Epoch 88/100
26/26 [==============================] - 1s 21ms/step - loss: 7.1813e-04 - val_loss: 0.0010
Epoch 89/100
```

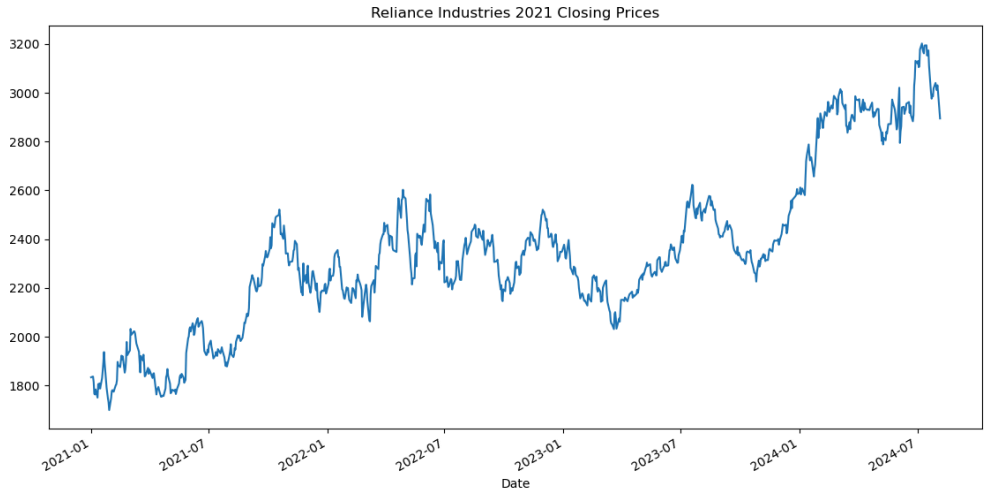


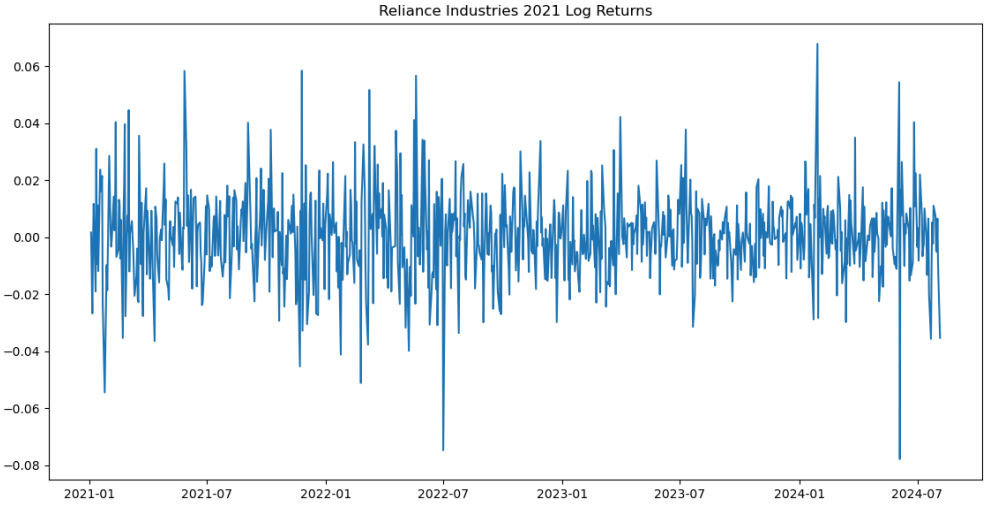
```
Epoch 83/100
26/26 [==============================] - 0s 15ms/step - loss: 6.9629e-04 - val_loss: 6.1115e-04
Epoch 84/100
26/26 [==============================] - 0s 16ms/step - loss: 6.8801e-04 - val_loss: 0.0013
Epoch 85/100
26/26 [==============================] - 0s 17ms/step - loss: 6.5478e-04 - val_loss: 2.7603e-04
Epoch 86/100
26/26 [==============================] - 0s 15ms/step - loss: 6.4909e-04 - val_loss: 7.8771e-04
Epoch 87/100
26/26 [==============================] - 0s 16ms/step - loss: 6.0374e-04 - val_loss: 2.9272e-04
Epoch 88/100
26/26 [==============================] - 1s 21ms/step - loss: 7.1813e-04 - val_loss: 0.0010
Epoch 89/100
26/26 [==============================] - 0s 15ms/step - loss: 7.9257e-04 - val_loss: 3.3027e-04
Epoch 90/100
26/26 [==============================] - 0s 14ms/step - loss: 5.8574e-04 - val_loss: 1.6055e-04
Epoch 91/100
26/26 [==============================] - 0s 15ms/step - loss: 5.6705e-04 - val_loss: 3.3008e-04
Epoch 92/100
26/26 [==============================] - 0s 16ms/step - loss: 5.6614e-04 - val_loss: 1.6610e-04
Epoch 93/100
26/26 [==============================] - 0s 14ms/step - loss: 5.8688e-04 - val_loss: 2.0786e-04
Epoch 94/100
26/26 [==============================] - 0s 15ms/step - loss: 5.4337e-04 - val_loss: 2.6669e-04
Epoch 95/100
26/26 [==============================] - 0s 15ms/step - loss: 6.4472e-04 - val_loss: 6.6005e-04
Epoch 96/100
26/26 [==============================] - 0s 18ms/step - loss: 6.6528e-04 - val_loss: 3.6064e-04
Epoch 97/100
26/26 [==============================] - 0s 18ms/step - loss: 5.7607e-04 - val_loss: 1.7115e-04
Epoch 98/100
26/26 [==============================] - 0s 15ms/step - loss: 6.1177e-04 - val_loss: 3.5430e-04
Epoch 99/100
26/26 [==============================] - 0s 18ms/step - loss: 5.5118e-04 - val_loss: 1.5534e-04
Epoch 100/100
26/26 [==============================] - 0s 19ms/step - loss: 5.7160e-04 - val_loss: 0.0010
PS C:\Users\dell\Pictures\VAR model\r codes>
```

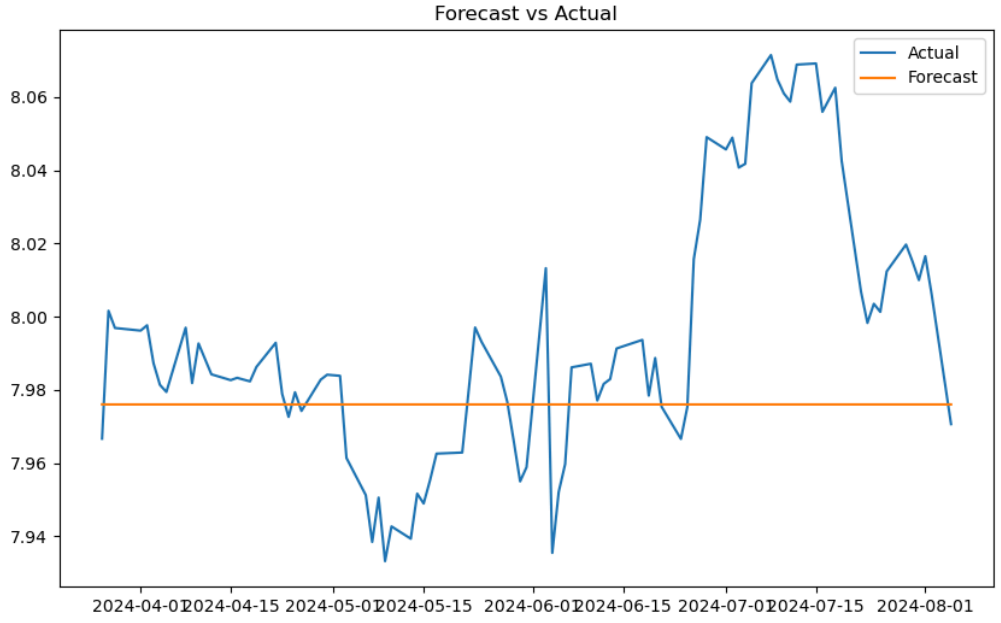
GRAPHS-

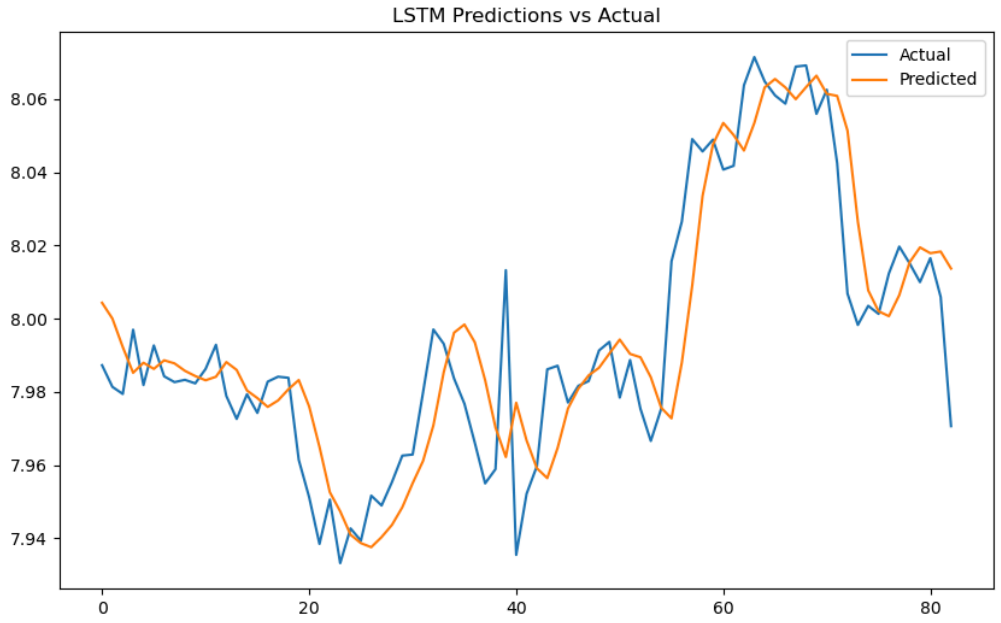

Reliance Industries Closing Prices
3000
2500
2000
1500
1000
2020
2021
2022
2023
2024
Date

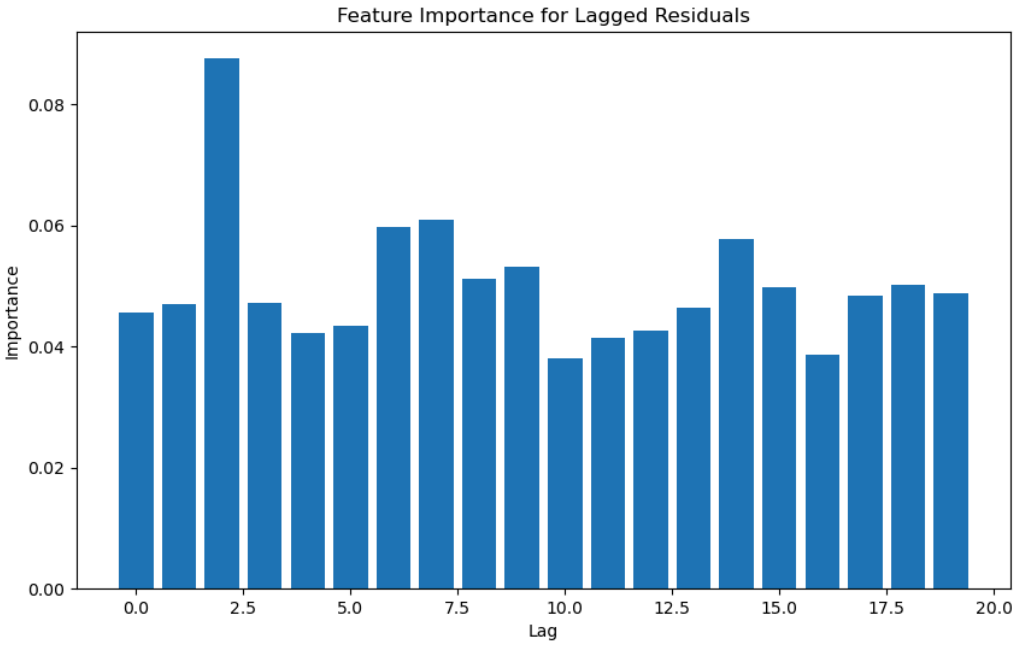

Reliance Industries Log Returns
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
2020
2021
2022
2023
2024


Reliance Industries 2021 Closing Prices
3200
3000
2800
2600
2400
2200
2000
1800
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date


Reliance Industries 2021 Log Returns
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07


Forecast vs Actual
Actual
Forecast
8.06
8.04
8.02
8.00
7.98
7.96
7.94
2024-04-01
2024-04-15
2024-05-01
2024-05-15
2024-06-01
2024-06-15
2024-07-01
2024-07-15
2024-08-01


LSTM Predictions vs Actual
Actual
Predicted
8.06
8.04
8.02
8.00
7.98
7.96
7.94
0
20
40
60
80



ABOUT CODE-

Data Acquisition and Visualization:
Fetches Reliance Industries stock data from Yahoo Finance.
Plots closing prices for 2020 and 2021.

Stationarity Tests:
Conducts stationarity tests on closing prices and log returns using the ADF test.

ARIMA Model:
Splits data into training and testing sets.
Fits an ARIMA model to the training data.
Forecasts and evaluates the model's performance using MSE.

GARCH Model:
Fits a GARCH model to the residuals of the ARIMA model.

Normalization and LSTM Model:
Normalizes data using Min-Max scaling.
Creates sequences for LSTM model training.
Defines, compiles, and trains an LSTM model.
Evaluates the LSTM model and plots predictions vs actual values.
Computes MAE, MSE, and RMSE.

Random Forest Model:

Creates a lagged data frame and trains a Random Forest model on the residuals from the ARIMA model.
Plots feature importance for lagged residuals.

Additional LSTM Model Training:
Normalizes data again with a different window size and trains a second LSTM model.

WHY GARCH Model?

The GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model is included here for a few reasons:

1.***Volatility Modeling***:
- GARCH models are specifically designed to model and forecast the volatility of financial time series data. Since financial markets often exhibit volatility clustering (periods of high volatility followed by periods of low volatility), GARCH can capture these dynamics more effectively than simple ARIMA models.

2. ***Residual Analysis***:
- After fitting the ARIMA model, you obtain residuals that are assumed to be white noise. However, if these residuals exhibit time-varying volatility, a GARCH model can help model this volatility. This can improve forecasting by providing a more accurate representation of the underlying uncertainty in the data.

3.***Enhancing Forecasting***:
- By combining ARIMA for modeling the mean of the time series and GARCH for modeling the volatility, you can potentially improve forecasting accuracy, especially in financial applications where understanding and predicting volatility is crucial.

4. ***Feature Engineering for Machine Learning Models***:
- The GARCH model's output, such as the conditional variance, can be used as a feature in machine learning models to enhance predictions. In your case, the feature importances from the Random Forest model could provide insights into how lagged residuals and volatility affect the forecasts.

In summary, incorporating a GARCH model allows for a more nuanced understanding of the time series data by accounting for volatility, which is particularly relevant for financial time series forecasting.

Interpretation of graphs-

A.The chart illustrates the closing prices of Reliance Industries stock from January 2020 to January 2024. The y-axis represents the stock price, and the x-axis represents time.

Key Observations

1. **Initial Volatility:** The stock price starts around 1500 in early 2020 and exhibits significant volatility in the first half of the year, with both upward and downward swings.
2. **Upward Trend:** From the second half of 2020, there's a clear upward trend in the stock price. It gradually increases, with a few periods of consolidation, reaching its peak around the end of 2023.
3. **Recent Decline:** From its peak in late 2023, the stock price has shown a downward trend, indicating a potential correction or change in market sentiment.

Interpretation

Reliance Industries stock demonstrated a positive performance from mid-2020 to late 2023, with a notable upward trend. However, the recent decline suggests a potential shift in market dynamics. The stock has experienced periods of volatility throughout the analyzed period, indicating market fluctuations and investor sentiment changes.

Additional Considerations

- **External Factors:** Economic conditions, industry trends, company performance, and global events likely influenced the stock price during this period.
- **Comparison:** Comparing the stock price to market indices or other industry peers can provide additional insights into Reliance Industries' performance.

B. The chart illustrates the daily log returns of Reliance Industries stock from January 2020 to January 2024. Log returns are a way to measure the percentage change in the stock price over time, taking into account compounding effects.

Key Observations

1. **Volatility:** The chart exhibits significant volatility, with both sharp positive and negative spikes in the log returns. This indicates periods of rapid price increases and decreases in the Reliance Industries stock.

2. **Clustering:** The log returns appear to cluster in certain periods, indicating periods of higher volatility or increased market activity.
3. **Positive and Negative Returns:** There is a mix of positive and negative log returns throughout the period, suggesting that the stock price has both increased and decreased during this timeframe.
4. **No Clear Trend:** Unlike the closing price chart, there isn't a clear upward or downward trend in the log returns. This suggests that the stock price has fluctuated significantly without a consistent direction.

Interpretation

The Reliance Industries stock has experienced a volatile period between 2020 and 2024, with frequent and significant price swings. While there have been periods of both gains and losses, the overall trend is not clear-cut. Investors in Reliance Industries would have experienced a rollercoaster ride during this period.

Additional Considerations

- **Comparison:** Comparing these log returns to the overall market index or other stocks can provide insights into Reliance Industries' performance relative to the broader market.
- **External Factors:** Economic conditions, industry trends, company performance, and global events likely influenced the stock's volatility during this period.

C. The chart illustrates the closing prices of Reliance Industries stock from January 2021 to August 2024. The y-axis represents the stock price, ranging from 1800 to 3200, and the x-axis represents time, spanning the specified period.

Key Observations

1. **Initial Volatility:** The stock price starts around 1800 in early 2021 and exhibits significant volatility in the first half of the year, with both upward and downward swings.
2. **Upward Trend:** From the second half of 2021, there's a clear upward trend in the stock price. It gradually increases, with a few periods of consolidation, reaching its peak around the end of 2023.
3. **Recent Decline:** From its peak in late 2023, the stock price has shown a downward trend, indicating a potential correction or change in market sentiment.

Interpretation

Reliance Industries stock demonstrated a positive performance from mid-2021 to late 2023, with a notable upward trend. However, the recent decline suggests a potential shift in market dynamics. The stock has experienced periods of volatility throughout the analyzed period, indicating market fluctuations and investor sentiment changes.

Additional Considerations

- **External Factors:** Economic conditions, industry trends, company performance, and global events likely influenced the stock price during this period.
- **Comparison:** Comparing the stock price to market indices or other industry peers can provide additional insights into Reliance Industries' performance.

D. The chart depicts the daily log returns of Reliance Industries stock from January 2021 to July 2024. Log returns are a way to measure the percentage change in the stock price over time, taking into account compounding effects.

Key Observations

1. **High Volatility:** The chart exhibits extreme volatility, with frequent and large swings in both positive and negative directions. This indicates periods of rapid and significant price changes in the Reliance Industries stock.
2. **No Clear Trend:** There is no apparent upward or downward trend in the log returns. The stock price has fluctuated significantly without a consistent direction, suggesting a highly volatile market environment.
3. **Extreme Values:** The chart shows several extreme values, both positive and negative, indicating days with exceptionally large price movements.

Interpretation

Reliance Industries stock experienced a highly volatile period between 2021 and 2024. The frequent and large price swings indicate a challenging market environment for investors. The absence of a clear trend suggests that predicting the stock price movement would have been difficult during this period.

Additional Considerations

- **Comparison:** Comparing these log returns to the overall market index or other stocks can provide insights into Reliance Industries' performance relative to the broader market.
- **External Factors:** Economic conditions, industry trends, company performance, and global events likely contributed to the stock's volatility during this period.

E. The chart compares the actual values of a dataset (represented by the blue line) with the predictions made by a forecasting model (represented by the orange line). The x-axis indicates the time period from April 2024 to August 2024, while the y-axis represents the values of the dataset.

Key Observations

1. **Constant Forecast:** The orange forecast line is relatively flat, indicating that the model is predicting a constant value of approximately 7.96 for the entire period.
2. **Fluctuating Actuals:** The blue actual line exhibits significant fluctuations, with values ranging from around 7.94 to 8.06. This suggests that the underlying data is highly volatile and unpredictable.
3. **Poor Predictive Power:** The large discrepancy between the forecast line and the actual line demonstrates that the forecasting model is not effective in capturing the underlying patterns and trends in the data.

Interpretation

The forecasting model used in this chart has limited predictive power for this dataset. The constant forecast line indicates that the model is unable to adapt to the dynamic nature of the data, which exhibits significant fluctuations.

Additional Considerations

- **Error Metrics:** To quantitatively assess the model's performance, it would be helpful to calculate error metrics like Mean Squared Error (MSE) or Mean Absolute Error (MAE).
- **Model Complexity:** The complexity of the forecasting model (e.g., simple moving average, exponential smoothing, ARIMA) can impact its predictive accuracy.
- **Data Characteristics:** The nature of the data (e.g., stationary vs. non-stationary, presence of trends or seasonality) can influence the model's performance.

F.The chart compares the actual values of a dataset (represented by the blue line) with the predictions made by an LSTM model (represented by the orange line). The x-axis indicates the data points, while the y-axis represents the values.

Key Observations

1. **Close Tracking:** The orange predicted line follows the blue actual line closely for most of the data points. This suggests that the LSTM model is capturing the underlying pattern in the data reasonably well.
2. **Some Deviations:** There are instances where the predicted line deviates slightly from the actual line, indicating areas where the model's predictions are less accurate.
3. **Overall Fit:** The overall fit between the predicted and actual values appears to be good, suggesting that the LSTM model is performing well in capturing the trends and patterns in the data.

Interpretation

The LSTM model has demonstrated a strong ability to predict the values in the dataset. While there are some areas where the predictions deviate from the actual values, the overall performance of the model is commendable.

Additional Considerations

- **Error Metrics:** To quantitatively assess the model's performance, it would be helpful to calculate error metrics like Mean Squared Error (MSE) or Mean Absolute Error (MAE).
- **Model Complexity:** The complexity of the LSTM model (number of layers, neurons) and the size of the training dataset can influence its predictive accuracy.
- **Data Characteristics:** The nature of the data (e.g., stationary vs. non-stationary, presence of trends or seasonality) can impact the model's performance.

G. The chart illustrates the importance of different lags (time periods) in predicting the residuals of a model. The x-axis represents the lag values, ranging from 0 to 20, and the y-axis represents the importance score, indicating the contribution of each lag to the prediction.

Key Observations

1. **Varying Importance:** The importance scores for different lags vary significantly. Some lags have higher importance scores, suggesting they are more influential in predicting the residuals, while others have lower scores, indicating less impact.
2. **No Clear Pattern:** There is no apparent pattern in the importance scores across the lags. The importance seems to fluctuate without a consistent trend.

3. **Highest Importance:** The highest importance score is observed for lag 2.5, indicating that this lag has the most significant contribution to predicting the residuals.

Interpretation

The chart suggests that the residuals of the model are influenced by multiple lags, with lag 2.5 being the most important factor. However, the importance of other lags cannot be disregarded as they also contribute to the prediction. The absence of a clear pattern in the importance scores indicates that the relationship between the residuals and the lags is complex and not easily explained by a simple trend.

Additional Considerations

- **Model Type:** The specific model used to generate the residuals and calculate the importance scores would provide additional context for interpreting the results.
- **Data Characteristics:** The nature of the data (e.g., time series, cross-sectional) and the presence of any trends or seasonality can influence the importance of different lags.