Name: Seshasai.PB Reg\_N0: 21MIA1005

Econometrics

ARIMA

**ARIMA Model Overview**

**ARIMA** (AutoRegressive Integrated Moving Average) is a statistical model used for forecasting time series data. It is particularly effective when data shows signs of non-stationarity, meaning that the mean, variance, and covariance of the data change over time. ARIMA models combine three components:

1. **AutoRegressive (AR)**: The AR part of the model involves regressing the variable on its own lagged (previous) values. The p parameter in ARIMA denotes the number of lag observations included in the model.
2. **Integrated (I)**: This component deals with differencing the raw observations to make the time series stationary. The d parameter represents the number of differencing steps needed to achieve stationarity.
3. **Moving Average (MA)**: The MA part models the error term as a linear combination of previous error terms. The q parameter refers to the number of lagged forecast errors in the prediction equation.

ARIMA models are denoted as **ARIMA(p, d, q)**, where:

* p: Number of lag observations (AR terms).
* d: Number of differencing operations to make the data stationary.
* q: Number of lagged forecast errors (MA terms).

**Code Overview**

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

import matplotlib.pyplot as plt

# Model Evaluation

def evaluate\_model(fitted\_model, actual\_data):

    # Predict over the entire range of actual data

    predictions = fitted\_model.predict(start=0, end=len(actual\_data)-1, dynamic=False)

    # Calculate MAE and RMSE

    mae = mean\_absolute\_error(actual\_data, predictions)

    mse = mean\_squared\_error(actual\_data, predictions)

    rmse = np.sqrt(mse)

    return predictions, mae, rmse

# Evaluate the ARIMA Model

predicted\_prices, mae\_val, rmse\_val = evaluate\_model(model\_fitted, closing\_prices)

print(f"Mean Absolute Error: {mae\_val}")

print(f"Root Mean Squared Error: {rmse\_val}")

# Forecast Future Values

def forecast\_future(model, steps=10):

    future\_forecast = model.forecast(steps=steps)

    return future\_forecast

future\_predictions = forecast\_future(model\_fitted)

# Plot Actual vs Predicted

plt.figure(figsize=(12, 6))

plt.plot(closing\_prices, label='Actual Prices', color='blue')

plt.plot(predicted\_prices, label='Predicted Prices', color='red', linestyle='--')

plt.title('Actual vs Predicted Prices')

plt.xlabel('Index')

plt.ylabel('Price')

plt.legend()

plt.show()

# Plot Forecasted Values

plt.figure(figsize=(12, 6))

plt.plot(closing\_prices, label='Actual Prices', color='blue')

plt.plot(np.arange(len(closing\_prices), len(closing\_prices) + len(future\_predictions)),

         future\_predictions, label='Forecasted Prices', color='green', linestyle='--')

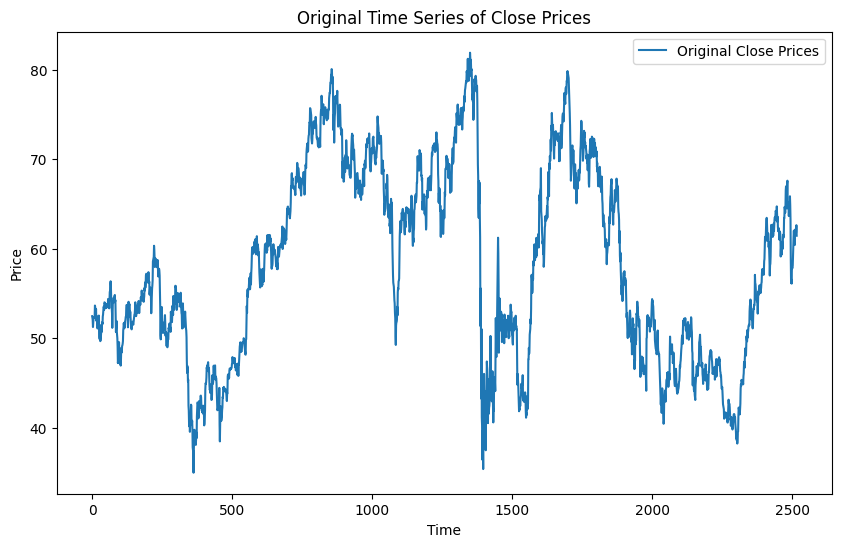
plt.title('Forecasted Prices')

plt.xlabel('Index')

plt.ylabel('Price')

plt.legend()

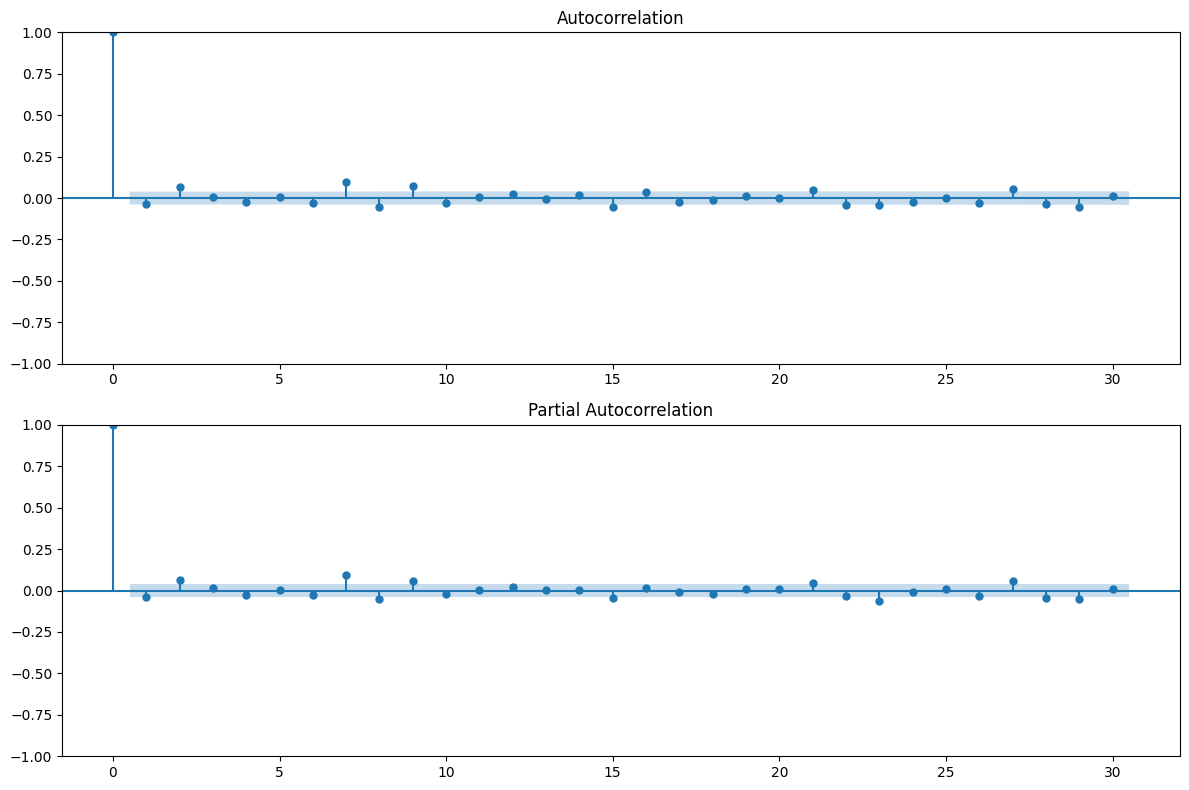
plt.show()



**Inference**: This plot shows the historical stock prices over time. The data demonstrates some trends, with noticeable periods of upward and downward movements. There is a visible spike around the middle of the time series (index ~1500) where the prices drop sharply and recover. The non-stationary nature of this time series is apparent due to the trends and volatility.



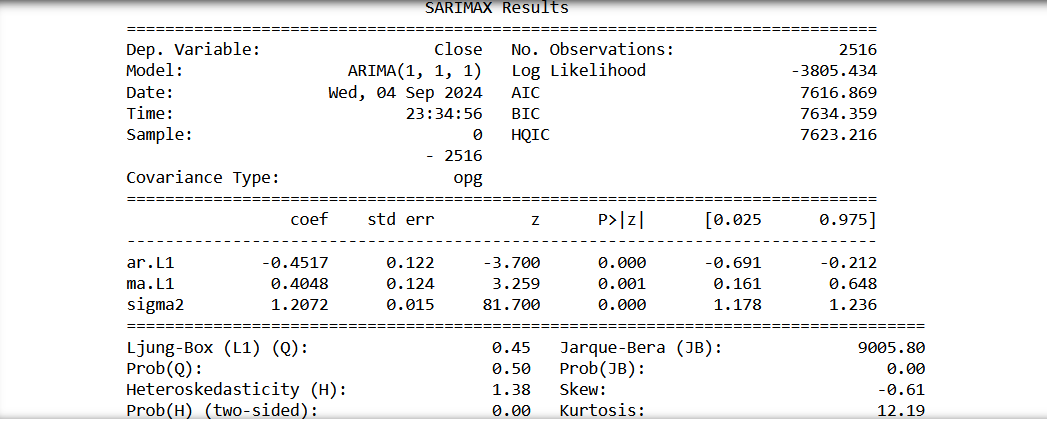
**Inference**: After differencing, the time series appears to fluctuate around a constant mean (approximately 0), indicating that it is now stationary. Differencing has removed the trends, making the series ready for ARIMA modeling. The spikes indicate large changes in price during certain periods, which are especially prominent around index 1500.



 **Autocorrelation Function (ACF)**: The ACF plot shows significant autocorrelations for the first few lags and then quickly diminishes, indicating that there is some correlation structure in the data but no strong periodic behavior.

 **Partial Autocorrelation Function (PACF)**: The PACF plot also shows a significant lag at the first point, suggesting that an AR(1) term could be appropriate in the ARIMA model.

 **Inference**: These plots suggest that an ARIMA(1,1,1) model is suitable, where 1 AR term and 1 MA term are included with first differencing.

****

 **Coefficients**:

* **AR(1) Coefficient**: -0.4517 (significant as p-value < 0.05)
* **MA(1) Coefficient**: 0.4048 (significant as p-value < 0.05)
* **Sigma^2**: 1.2072, representing the variance of the residuals.

 **Model Quality**:

* **AIC (Akaike Information Criterion)**: 7616.869. A lower AIC indicates a better-fitting model. This value is reasonable for the complexity of the model.
* **Log Likelihood**: -3805.434, which reflects the goodness of fit (higher is better).
* **Diagnostics**: The model passes basic statistical tests like Ljung-Box (p-value > 0.05) and shows no significant autocorrelation in residuals.

 **Inference**: The ARIMA(1,1,1) model fits the data well, with both AR(1) and MA(1) terms being statistically significant.

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

import matplotlib.pyplot as plt

# Model Evaluation

def evaluate\_model(fitted\_model, actual\_data):

    # Predict over the entire range of actual data

    predictions = fitted\_model.predict(start=0, end=len(actual\_data)-1, dynamic=False)

    # Calculate MAE and RMSE

    mae = mean\_absolute\_error(actual\_data, predictions)

    mse = mean\_squared\_error(actual\_data, predictions)

    rmse = np.sqrt(mse)

    return predictions, mae, rmse

# Evaluate the ARIMA Model

predicted\_prices, mae\_val, rmse\_val = evaluate\_model(model\_fitted, closing\_prices)

print(f"Mean Absolute Error: {mae\_val}")

print(f"Root Mean Squared Error: {rmse\_val}")

# Forecast Future Values

def forecast\_future(model, steps=10):

    future\_forecast = model.forecast(steps=steps)

    return future\_forecast

future\_predictions = forecast\_future(model\_fitted)

# Plot Actual vs Predicted

plt.figure(figsize=(12, 6))

plt.plot(closing\_prices, label='Actual Prices', color='blue')

plt.plot(predicted\_prices, label='Predicted Prices', color='red', linestyle='--')

plt.title('Actual vs Predicted Prices')

plt.xlabel('Index')

plt.ylabel('Price')

plt.legend()

plt.show()

# Plot Forecasted Values

plt.figure(figsize=(12, 6))

plt.plot(closing\_prices, label='Actual Prices', color='blue')

plt.plot(np.arange(len(closing\_prices), len(closing\_prices) + len(future\_predictions)),

         future\_predictions, label='Forecasted Prices', color='green', linestyle='--')

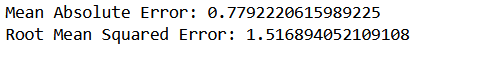
plt.title('Forecasted Prices')

plt.xlabel('Index')

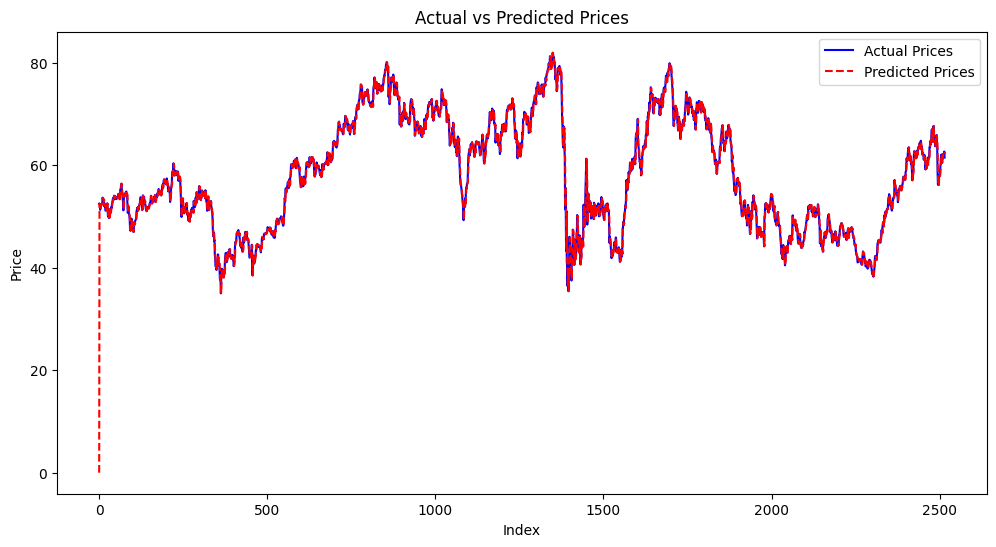
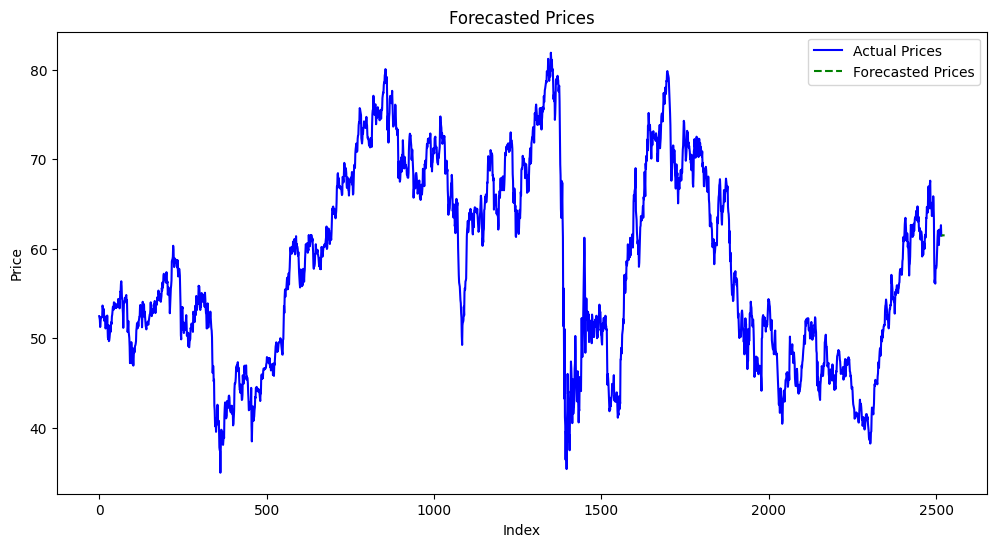
plt.ylabel('Price')

plt.legend()

plt.show()

****

The low values for MAE and RMSE indicate that the model has a relatively good predictive power. The errors are not too large, suggesting that the model captures the underlying patterns in the stock prices effectively.

**Inference**: The predicted values (dashed red line) closely follow the actual stock prices (solid blue line), indicating a good fit of the ARIMA model. The overlap between the two lines suggests that the model is capturing the major trends and fluctuations in the data accurately, even with the variability in the stock prices.

**Inference**: The green dashed line shows the forecasted prices for future time periods. The forecast continues the recent trend observed in the actual data, with the forecast suggesting that the prices will remain stable or slightly increase in the near future. This forecast provides a reasonable continuation based on past behavior, but given the variability in stock prices, long-term predictions should be interpreted cautiously.

**Code Explanation**

**Applying ARIMA to the Dataset**

**Step 1: Stationarity Check**

The first step when using an ARIMA model is to check if the data is **stationary**. Stationarity implies that the statistical properties of the time series (such as mean and variance) remain constant over time. To assess stationarity, the **Augmented Dickey-Fuller (ADF)** test is used. A p-value less than 0.05 typically suggests that the series is stationary.

In the code:

is\_stationary = check\_stationarity(closing\_prices)

The function check\_stationarity() checks if the dataset is stationary using the ADF test. If the data is non-stationary, we apply **differencing** (subtracting previous values) to make it stationary.

**Step 2: Differencing the Data**

If the data is non-stationary (p-value > 0.05), differencing is applied:

if not is\_stationary:

    stock\_data['Diff\_Close'] = closing\_prices.diff().dropna()

    diff\_prices = stock\_data['Diff\_Close']

This operation subtracts the previous value from the current one to eliminate trends or seasonality, making the time series stationary, which is necessary for ARIMA models to perform accurately.

**Step 3: ACF and PACF Analysis**

**ACF (Auto-Correlation Function)** and **PACF (Partial Auto-Correlation Function)** plots are used to determine the order of the AR and MA terms in the ARIMA model. The ACF measures the correlation between the time series and its lagged versions, while PACF helps isolate the relationship with only immediate past observations. These plots help decide the values of p (AR term) and q (MA term).

plot\_acf\_pacf(diff\_prices)

This function plots the ACF and PACF to visualize the dependencies between different time points in the series.

**Step 4: Fitting the ARIMA Model**

After determining the necessary values for p, d, and q, the ARIMA model is fitted:

model\_fitted = fit\_arima\_model(closing\_prices, 1, 1, 1)

Here, the ARIMA(1,1,1) model is being applied to the differenced data. The values of p=1, d=1, and q=1 indicate:

* **1 lag** of autoregressive terms,
* **1 differencing step** to make the data stationary,
* **1 lag** of the moving average term.

This model is then used for forecasting future values and making predictions.

**Model Evaluation**

To assess how well the ARIMA model fits the data, **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)** are used. These are common error metrics in forecasting to measure the average prediction error:

predictions, mae, rmse = evaluate\_model(model\_fitted, closing\_prices)

print(f"Mean Absolute Error: {mae}")

print(f"Root Mean Squared Error: {rmse}")

* **MAE** represents the average absolute error between actual and predicted values.
* **RMSE** is the square root of the average squared errors and penalizes large errors more than MAE.

Both metrics provide insight into how well the model's predictions align with the actual data.

**Forecasting Future Values**

Once the model is fitted, it can be used to predict future values. This is done using the ARIMA model's .forecast() method:

future\_predictions = forecast\_future(model\_fitted, steps=10)

This generates forecasts for the next 10 time points based on the trained ARIMA model.

**Visualization**

The code generates two types of plots:

1. **Actual vs. Predicted Prices**: This compares the actual stock prices with the prices predicted by the ARIMA model over the historical data.

plt.plot(closing\_prices, label='Actual Prices')

plt.plot(predicted\_prices, label='Predicted Prices')

1. **Forecasted Prices**: This plot extends the existing data to include future predicted values.

plt.plot(np.arange(len(closing\_prices), len(closing\_prices) + len(future\_predictions)),

         future\_predictions, label='Forecasted Prices')

These plots help visualize the model's performance and future forecasts, allowing for better interpretation of trends in the data.

**Conclusion**

This ARIMA model analyzes the **closing stock prices** from the dataset, checks for stationarity, fits an appropriate ARIMA model, and makes predictions about future prices. By using ADF tests for stationarity, differencing for non-stationary data, and error metrics like MAE and RMSE, the model effectively captures trends and forecasts the future trajectory of stock prices.