Libraries used:

- pandas: primary data loading, cleaning, resampling, indexing, feature creation, and DataFrame operations.

- numpy: numerical operations and NaN handling.

- pathlib: path handling for filenames (used when building file paths).

- matplotlib.pyplot: plotting time series, moving averages, volume charts, and model results.

- sqlalchemy.create_engine and urllib.parse.quote_plus: build a MySQL connection URL and load the DataFrame to a MySQL table via to_sql.

- pymysql (used implicitly by SQLAlchemy engine string): MySQL driver used by SQLAlchemy.

- sklearn.ensemble.RandomForestClassifier: classification model to predict next-day positive return.

- sklearn.metrics (accuracy_score, classification_report, confusion_matrix): evaluate classifier performance.

- sklearn.model_selection.TimeSeriesSplit: (imported) useful for time-series-aware cross-validation.

- statsmodels.tsa.arima.model.ARIMA: classical ARIMA time-series forecasting.

- warnings: to suppress non-critical warnings during modeling.

Data loading & initial preparation

1. Read CSV: Google_Stock_Train (2010-2023).csv using pandas.read_csv.

2. Parse 'Date' column into datetime with format '%d-%m-%Y'.

3. Sort by 'Date', reset index, and set 'Date' as the DataFrame index for time-series operations.

Basic feature creation and cleaning

1. Daily_Return: fractional day-to-day change computed with pct_change().

2. High_Low_Range: difference between 'High' and 'Low'.

3. Price_Momentum_7d: difference between current Close and Close shifted by 7 days.

4. Rolling moving average 'MA_30' created as Close.rolling(30).mean().

Resampling

- Weekly ('W') and monthly ('ME') aggregations were created with common aggregations:

  Open:first, High:max, Low:min, Close:last, Adj Close:last, Volume:sum,

  Daily_Return:mean, High_Low_Range:mean, Price_Momentum_7d:last.

- dropna() applied to remove incomplete groups.

Exploratory plots

- Full Close price trend plotted over time.

- Close price plotted with 30-day moving average overlay.

- Volume bar chart with spikes highlighted where Volume > mean + 2*std.

Feature engineering for classification

1. Create 'Return' = Close.pct_change() and lag features return_lag_1 .. return_lag_7.

2. Create additional features:

   - ma_7, ma_30: rolling means of Close.

   - vol_7, vol_30: rolling std of Return as volatility proxies.

   - momentum_7: Close - Close.shift(7).

   - volume_change: Volume.pct_change().

   - dayofweek, month: calendar features extracted from the datetime index.

Target variable

- next_close = Close.shift(-1)

- next_pos = 1 if next_close > Close else 0 → the classification target (next-day positive return).

Prepare modeling DataFrame

- Select feature columns (lag returns, moving averages, vols, momentum, volume_change, dayofweek, month).

- Drop rows with NaNs in features/target using dropna(subset=...).

Train-test splitting (time-aware)

- Primary split: training data up to 2021-12-31, test data from 2022-01-01 onward.

- If test set is empty, fallback to last 20% as test set.

RandomForest classification

- Train RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1).

- Predict on test set and compute:

    - accuracy_score

    - classification_report (precision, recall, f1)

    - confusion_matrix

- Compute and display feature importances as a sorted Series.

ARIMA forecasting

- Train ARIMA on Close series up to 2022-12-31 (fallback to full series if too short).

- Chosen order: (5,1,0) as a simple default.

- Fit ARIMA model and forecast the next 60 calendar days (freq='D'), producing forecast values and a date index for plotting.

- Record model AIC for summary.

Outputs and visualizations

- Save results dictionary:

    classifier_accuracy, classification_report, confusion_matrix (as list), top_features (top 10), arima_summary_aic.

- Plot top 10 feature importances.

- Plot ARIMA forecast vs recent actuals (last ~200 days).

- Display confusion matrix in tabular form and a sample of the ARIMA forecast.

Database save (optional)

- Uses create_engine with a URL that encodes special characters in the password via quote_plus.

- df.to_sql(table_name, con=engine, if_exists='replace', index=False) to load the cleaned DataFrame into MySQL (table name: 'stock').

Notes, caveats, and suggestions

- Ensure date parsing format matches the CSV; current code uses '%d-%m-%Y'.

- ARIMA forecasting with daily freq on trading data is an approximation; better to forecast based on trading-day frequency or use market calendars.

- RandomForest is used for classification (directional). For probabilistic trading signals consider using predict_proba and backtesting returns, with transaction costs and slippage.

- Dropna removes rows needed by lag/rolling features; ensure enough historical rows remain for modeling.

- Consider hyperparameter tuning and time-series-aware cross-validation before trusting model performance.

- For reproducibility, set random_state where applicable and record package versions if needed.