```python
# First, make sure to import the yfinance library
import yfinance as yf

# Create a Ticker object for Nifty 50 (the correct symbol might be "^NSEI")
nifty50 = yf.Ticker("^NSEI")  # Use the correct ticker symbol for Nifty 50

# Now you can use the history method
nifty50_data = nifty50.history(period="max")

nifty50_data
```

```
                                  Open          High           Low  \
Date
2007-09-17 00:00:00+05:30   4518.450195   4549.049805   4482.850098
2007-09-18 00:00:00+05:30   4494.100098   4551.799805   4481.549805
2007-09-19 00:00:00+05:30   4550.250000   4739.000000   4550.250000
2007-09-20 00:00:00+05:30   4734.850098   4760.850098   4721.149902
2007-09-21 00:00:00+05:30   4752.950195   4855.700195   4733.700195
...                                 ...           ...           ...
2025-12-08 00:00:00+05:30  26159.800781  26178.699219  25892.250000
2025-12-09 00:00:00+05:30  25867.099609  25923.650391  25728.000000
2025-12-10 00:00:00+05:30  25864.050781  25947.650391  25734.550781
2025-12-11 00:00:00+05:30  25771.400391  25922.800781  25693.250000
2025-12-12 00:00:00+05:30  25971.199219  26057.599609  25938.449219

                                 Close  Volume  Dividends  Stock
Splits
Date

2007-09-17 00:00:00+05:30   4494.649902       0        0.0
0.0
2007-09-18 00:00:00+05:30   4546.200195       0        0.0
0.0
2007-09-19 00:00:00+05:30   4732.350098       0        0.0
0.0
2007-09-20 00:00:00+05:30   4747.549805       0        0.0
0.0
2007-09-21 00:00:00+05:30   4837.549805       0        0.0
0.0
...                                 ...     ...        ...           .
..
2025-12-08 00:00:00+05:30  25960.550781  320700        0.0
0.0
2025-12-09 00:00:00+05:30  25839.650391  276000        0.0
0.0
2025-12-10 00:00:00+05:30  25758.000000  207900        0.0
0.0
2025-12-11 00:00:00+05:30  25898.550781  206100        0.0
```

```
0.0
2025-12-12 00:00:00+05:30  26046.949219       0         0.0
0.0

[4475 rows x 7 columns]
```

```
nifty50_data.index
```

```
DatetimeIndex(['2007-09-17 00:00:00+05:30', '2007-09-18
00:00:00+05:30',
               '2007-09-19 00:00:00+05:30', '2007-09-20
00:00:00+05:30',
               '2007-09-21 00:00:00+05:30', '2007-09-24
00:00:00+05:30',
               '2007-09-25 00:00:00+05:30', '2007-09-26
00:00:00+05:30',
               '2007-09-27 00:00:00+05:30', '2007-09-28
00:00:00+05:30',
               ...
               '2025-12-01 00:00:00+05:30', '2025-12-02
00:00:00+05:30',
               '2025-12-03 00:00:00+05:30', '2025-12-04
00:00:00+05:30',
               '2025-12-05 00:00:00+05:30', '2025-12-08
00:00:00+05:30',
               '2025-12-09 00:00:00+05:30', '2025-12-10
00:00:00+05:30',
               '2025-12-11 00:00:00+05:30', '2025-12-12
00:00:00+05:30'],
              dtype='datetime64[ns, Asia/Kolkata]', name='Date',
length=4475, freq=None)
```

```
 # cleaning and visualizing our stock market
nifty50_data.plot.line(y="Close",use_index=True)
```

```
<Axes: xlabel='Date'>
```


```
# Check if columns exist before deleting them
if "Stock Splits" in nifty50_data.columns:
    del nifty50_data["Stock Splits"]

if "Dividends" in nifty50_data.columns:
    del nifty50_data["Dividends"]

# Alternative approach using drop method with errors='ignore'
# nifty50_data = nifty50_data.drop(["Stock Splits", "Dividends"],
axis=1, errors='ignore')

nifty50_data
```

Date	Open	High	Low	\
2007-09-17 00:00:00+05:30	4518.450195	4549.049805	4482.850098	
2007-09-18 00:00:00+05:30	4494.100098	4551.799805	4481.549805	
2007-09-19 00:00:00+05:30	4550.250000	4739.000000	4550.250000	
2007-09-20 00:00:00+05:30	4734.850098	4760.850098	4721.149902	
2007-09-21 00:00:00+05:30	4752.950195	4855.700195	4733.700195	
...	...	...	...	
2025-12-08 00:00:00+05:30	26159.800781	26178.699219	25892.250000	
2025-12-09 00:00:00+05:30	25867.099609	25923.650391	25728.000000	
2025-12-10 00:00:00+05:30	25864.050781	25947.650391	25734.550781	
2025-12-11 00:00:00+05:30	25771.400391	25922.800781	25693.250000	
2025-12-12 00:00:00+05:30	25971.199219	26057.599609	25938.449219	

```
                                  Close  Volume
Date
2007-09-17 00:00:00+05:30   4494.649902       0
2007-09-18 00:00:00+05:30   4546.200195       0
2007-09-19 00:00:00+05:30   4732.350098       0
2007-09-20 00:00:00+05:30   4747.549805       0
2007-09-21 00:00:00+05:30   4837.549805       0
...                                 ...     ...
2025-12-08 00:00:00+05:30  25960.550781  320700
2025-12-09 00:00:00+05:30  25839.650391  276000
2025-12-10 00:00:00+05:30  25758.000000  207900
2025-12-11 00:00:00+05:30  25898.550781  206100
2025-12-12 00:00:00+05:30  26046.949219       0

[4475 rows x 5 columns]
```

```python
import numpy as np
import pandas as pd

df = nifty50_data.copy()

# Daily returns
df["Return"] = df["Close"].pct_change()

# Rolling volatility
df["Volatility"] = df["Return"].rolling(5).std()

# Moving averages
df["MA10"] = df["Close"].rolling(10).mean()
df["MA20"] = df["Close"].rolling(20).mean()

df.dropna(inplace=True)
df.head()
```

```
                                  Open         High          Low
Close  \
Date

2007-10-15 00:00:00+05:30  5428.350098  5682.649902  5419.899902
5670.399902
2007-10-16 00:00:00+05:30  5670.649902  5708.350098  5578.450195
5668.049805
2007-10-17 00:00:00+05:30  5658.899902  5658.899902  5107.299805
5559.299805
2007-10-18 00:00:00+05:30  5551.100098  5736.799805  5269.649902
5351.000000
2007-10-19 00:00:00+05:30  5360.350098  5390.850098  5101.750000
5215.299805

                           Volume    Return  Volatility
```

```
MA10  \
Date

2007-10-15 00:00:00+05:30       0  0.044609    0.026320  5315.155029

2007-10-16 00:00:00+05:30       0 -0.000414    0.023370  5375.064990

2007-10-17 00:00:00+05:30       0 -0.019186    0.026431  5409.914990

2007-10-18 00:00:00+05:30       0 -0.037469    0.031179  5424.150000

2007-10-19 00:00:00+05:30       0 -0.025360    0.032091  5427.094971


                                      MA20
Date
2007-10-15 00:00:00+05:30  5067.165015
2007-10-16 00:00:00+05:30  5125.835010
2007-10-17 00:00:00+05:30  5176.489990
2007-10-18 00:00:00+05:30  5207.422485
2007-10-19 00:00:00+05:30  5230.809985
```

```
# Median volatility as threshold
vol_threshold = df["Volatility"].median()

df["Target"] = (df["Volatility"].shift(-1) >
vol_threshold).astype(int)

df.dropna(inplace=True)

df[["Volatility", "Target"]]
```

```
                           Volatility  Target
Date
2007-10-15 00:00:00+05:30    0.026320       1
2007-10-16 00:00:00+05:30    0.023370       1
2007-10-17 00:00:00+05:30    0.026431       1
2007-10-18 00:00:00+05:30    0.031179       1
2007-10-19 00:00:00+05:30    0.032091       1
...                                ...     ...
2025-12-08 00:00:00+05:30    0.005745       0
2025-12-09 00:00:00+05:30    0.005617       0
2025-12-10 00:00:00+05:30    0.005670       0
2025-12-11 00:00:00+05:30    0.006423       0
2025-12-12 00:00:00+05:30    0.006387       0

[4456 rows x 2 columns]
```

```
features = ["Return", "MA10", "MA20", "Volatility"]
```

```
X = df[features]
y = df["Target"]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    n_estimators=200,
    max_depth=5,
    random_state=42
)

model.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=5, n_estimators=200, random_state=42)
```

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
```

```
Accuracy: 0.8778026905829597

Confusion Matrix:
 [[599  62]
 [ 47 184]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       661
           1       0.75      0.80      0.77       231

    accuracy                           0.88       892
   macro avg       0.84      0.85      0.84       892
weighted avg       0.88      0.88      0.88       892
```

```
latest_data = X.iloc[-1:].values
prediction = model.predict(latest_data)[0]

if prediction == 1:
```

```
    print("⚠ High volatility expected tomorrow")
else:
    print("✅ Low volatility expected tomorrow")
```

```
✅ Low volatility expected tomorrow
```

```
C:\Users\chait\AppData\Local\Programs\Orange\Lib\site-packages\
sklearn\base.py:465: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

```
df = nifty50_data.copy()

df["Target"] = df["Close"].shift(-1)

df.dropna(inplace=True)

df[["Close", "Target"]]
```

Date	Close	Target
2007-09-17 00:00:00+05:30	4494.649902	4546.200195
2007-09-18 00:00:00+05:30	4546.200195	4732.350098
2007-09-19 00:00:00+05:30	4732.350098	4747.549805
2007-09-20 00:00:00+05:30	4747.549805	4837.549805
2007-09-21 00:00:00+05:30	4837.549805	4932.200195
...	...	...
2025-12-05 00:00:00+05:30	26186.449219	25960.550781
2025-12-08 00:00:00+05:30	25960.550781	25839.650391
2025-12-09 00:00:00+05:30	25839.650391	25758.000000
2025-12-10 00:00:00+05:30	25758.000000	25898.550781
2025-12-11 00:00:00+05:30	25898.550781	26046.949219

```
[4474 rows x 2 columns]
```

```
# Returns
df["Return"] = df["Close"].pct_change()

# Moving averages
df["MA5"] = df["Close"].rolling(5).mean()
df["MA10"] = df["Close"].rolling(10).mean()

# Volatility
df["Volatility"] = df["Return"].rolling(5).std()

df.dropna(inplace=True)

features = ["Close", "Return", "MA5", "MA10", "Volatility"]

X = df[features]
y = df["Target"]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(
    n_estimators=300,
    max_depth=6,
    random_state=42
)

model.fit(X_train, y_train)
```

```
RandomForestRegressor(max_depth=6, n_estimators=300, random_state=42)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print("MAE:", mae)
print("RMSE:", rmse)
```

```
MAE: 3405.4683136570684
RMSE: 4405.556024129921
```

```
latest_features = X.iloc[-1:].values
tomorrow_close = model.predict(latest_features)[0]

print("□ Predicted NIFTY 50 Close Tomorrow:", round(tomorrow_close,
2))
```

```
□ Predicted NIFTY 50 Close Tomorrow: 18196.16
```

```
C:\Users\chait\AppData\Local\Programs\Orange\Lib\site-packages\
sklearn\base.py:465: UserWarning: X does not have valid feature names,
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

```
df = nifty50_data.copy()

df["Return"] = df["Close"].pct_change()
df["Target"] = df["Return"].shift(-1)

df.dropna(inplace=True)
```

```
df["MA5"] = df["Close"].rolling(5).mean()
df["MA10"] = df["Close"].rolling(10).mean()
df["Volatility"] = df["Return"].rolling(5).std()

df.dropna(inplace=True)

features = ["Return", "MA5", "MA10", "Volatility"]

X = df[features]
y = df["Target"]

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, shuffle=False
)

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(
    n_estimators=300,
    max_depth=6,
    random_state=42
)

model.fit(X_train, y_train)

RandomForestRegressor(max_depth=6, n_estimators=300, random_state=42)

from sklearn.metrics import mean_absolute_error, mean_squared_error

y_pred = model.predict(X_test)

print("MAE (Return):", mean_absolute_error(y_test, y_pred))
print("RMSE (Return):", mean_squared_error(y_test, y_pred,
squared=False))

MAE (Return): 0.005811790142321782
RMSE (Return): 0.007928511346090988

latest_return_pred = model.predict(X_scaled[-1:].reshape(1, -1))[0]

last_close = df["Close"].iloc[-1]
predicted_close = last_close * (1 + latest_return_pred)

print("□ Predicted NIFTY Close Tomorrow:", round(predicted_close, 2))
```

```
 Predicted NIFTY Close Tomorrow: 25900.51
```

```
#On average, your model’s next-day return prediction error is less
than 1%.

# Predict next-day return
predicted_return = model.predict(X_scaled[-1:].reshape(1, -1))[0]

# Convert return to price
last_close = df["Close"].iloc[-1]
predicted_close = last_close * (1 + predicted_return)

print("Last Close:", round(last_close, 2))
print("Predicted Return:", round(predicted_return * 100, 3), "%")
print(" Predicted NIFTY Close Tomorrow:", round(predicted_close, 2))
```

```
Last Close: 25898.55
Predicted Return: 0.019 %
 Predicted NIFTY Close Tomorrow: 25903.51
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)  # FIT ONLY ON TRAIN
X_test_scaled = scaler.transform(X_test)        # TRANSFORM TEST

model.fit(X_train_scaled, y_train)

RandomForestRegressor(max_depth=6, n_estimators=300, random_state=42)

y_pred = model.predict(X_test_scaled)

print("MAE:", mean_absolute_error(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
```

```
MAE: 0.005811790142321782
RMSE: 0.007928511346090988
```

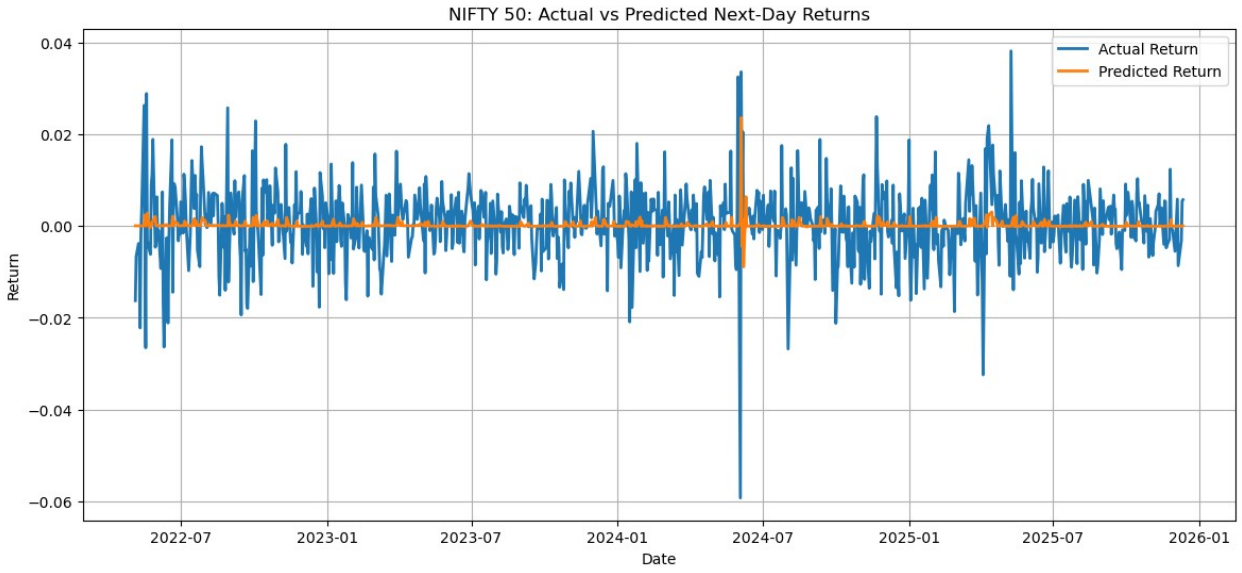
```
# Scale test data correctly
X_test_scaled = scaler.transform(X_test)

# Predict returns
y_test_pred = model.predict(X_test_scaled)
```

```
results = pd.DataFrame({
    "Actual_Return": y_test.values,
    "Predicted_Return": y_test_pred
}, index=y_test.index)

results
```

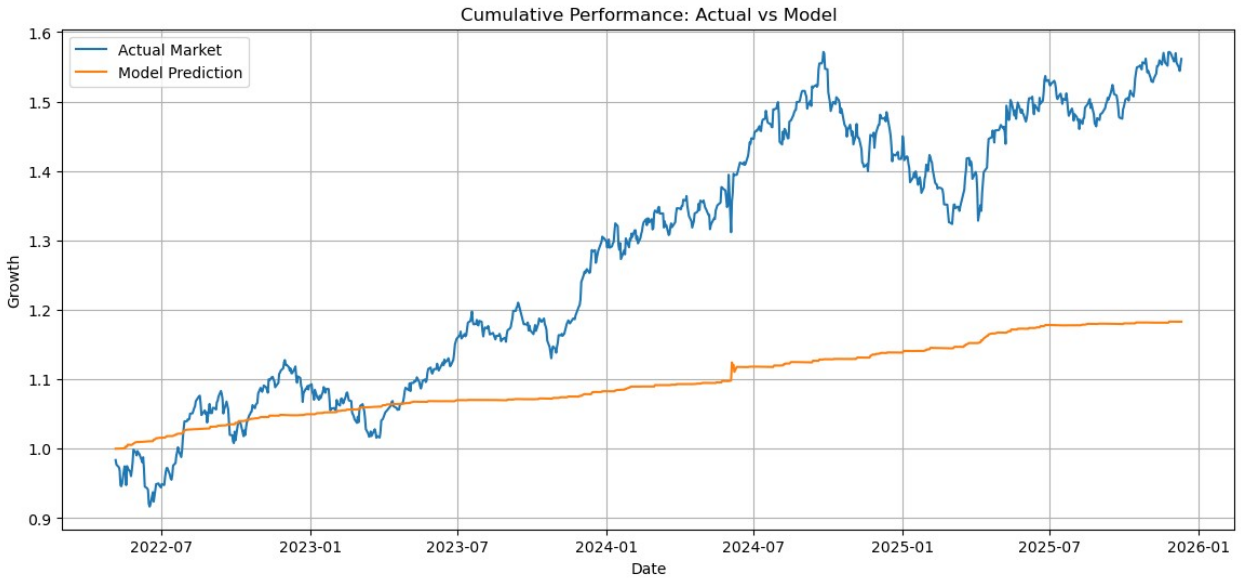
Date	Actual_Return	Predicted_Return
2022-05-05 00:00:00+05:30	-0.016268	0.000061
2022-05-06 00:00:00+05:30	-0.006666	0.000049
2022-05-09 00:00:00+05:30	-0.003791	0.000049
2022-05-10 00:00:00+05:30	-0.004492	0.000049
2022-05-11 00:00:00+05:30	-0.022212	0.000049
...	...	...
2025-12-05 00:00:00+05:30	-0.008627	0.000094
2025-12-08 00:00:00+05:30	-0.004657	-0.000049
2025-12-09 00:00:00+05:30	-0.003160	-0.000049
2025-12-10 00:00:00+05:30	0.005457	-0.000049
2025-12-11 00:00:00+05:30	0.005730	0.000076

[893 rows x 2 columns]

```
import matplotlib.pyplot as plt

plt.figure(figsize=(14,6))
plt.plot(results.index, results["Actual_Return"], label="Actual
Return", linewidth=2)
plt.plot(results.index, results["Predicted_Return"], label="Predicted
Return", linewidth=2)

plt.title("NIFTY 50: Actual vs Predicted Next-Day Returns")
plt.xlabel("Date")
plt.ylabel("Return")
plt.legend()
plt.grid(True)

plt.show()
```



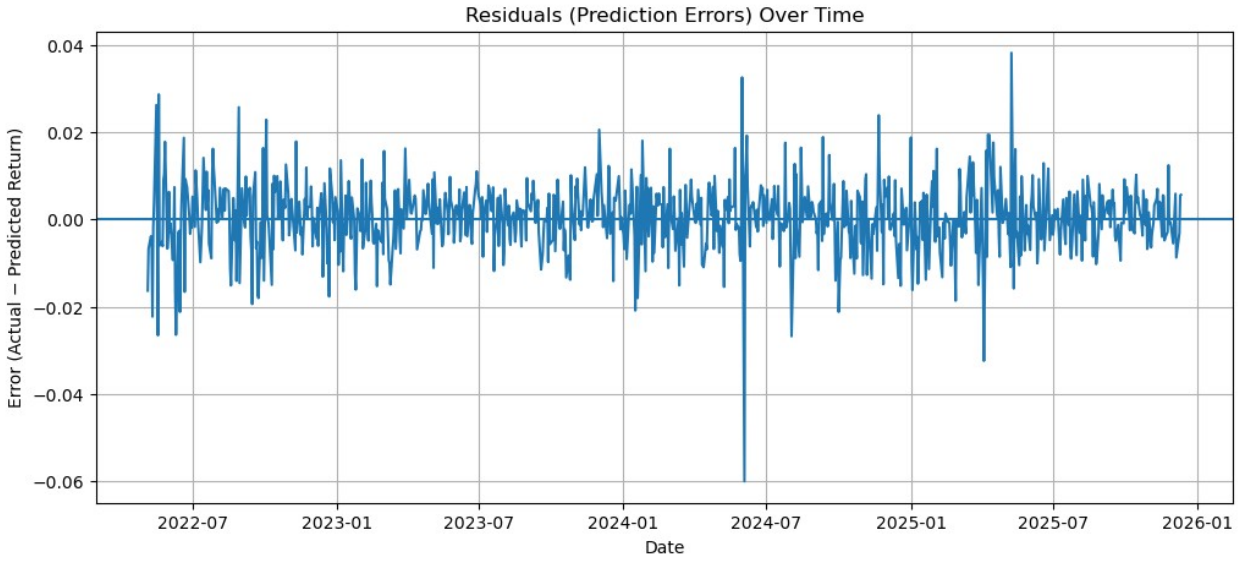

```
results["Actual_Cumulative"] = (1 +
results["Actual_Return"]).cumprod()
results["Predicted_Cumulative"] = (1 +
results["Predicted_Return"]).cumprod()

plt.figure(figsize=(14,6))
plt.plot(results.index, results["Actual_Cumulative"], label="Actual
Market")
plt.plot(results.index, results["Predicted_Cumulative"], label="Model
Prediction")

plt.title("Cumulative Performance: Actual vs Model")
plt.xlabel("Date")
plt.ylabel("Growth")
plt.legend()
plt.grid(True)

plt.show()
```



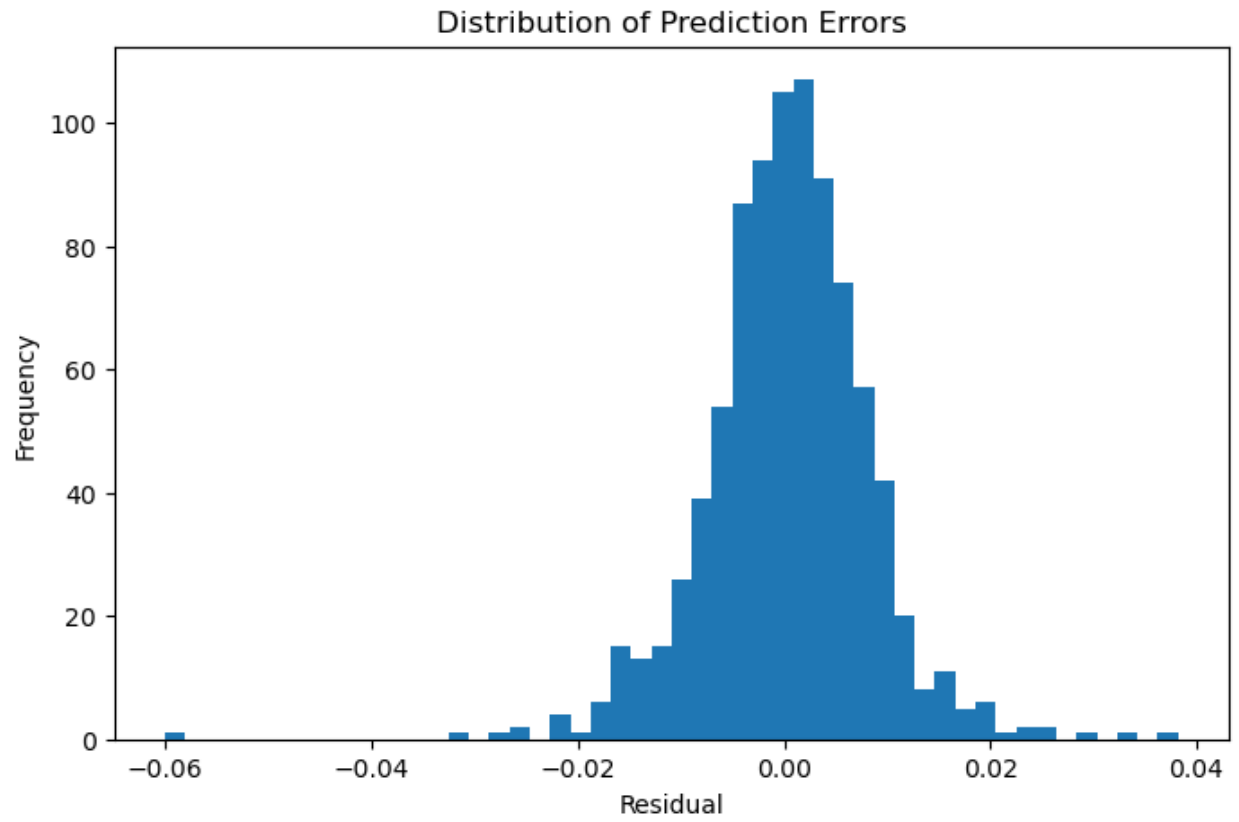

```
# Direction: 1 = Up, 0 = Down
results["Actual_Direction"] = (results["Actual_Return"] >
0).astype(int)
results["Predicted_Direction"] = (results["Predicted_Return"] >
0).astype(int)

direction_accuracy = (
    results["Actual_Direction"] == results["Predicted_Direction"]
).mean()

print("Directional Accuracy:", round(direction_accuracy * 100, 2),
"%")
```

```
Directional Accuracy: 50.06 %
```

```
# random

results["Residual"] = results["Actual_Return"] -
results["Predicted_Return"]

plt.figure(figsize=(12,5))
plt.plot(results.index, results["Residual"])
plt.axhline(0)

plt.title("Residuals (Prediction Errors) Over Time")
plt.xlabel("Date")
plt.ylabel("Error (Actual - Predicted Return)")
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(8,5))
plt.hist(results["Residual"], bins=50)
plt.title("Distribution of Prediction Errors")
plt.xlabel("Residual")
plt.ylabel("Frequency")
plt.show()
```



```
import pandas as pd

importance = pd.Series(
    model.feature_importances_,
    index=X.columns
).sort_values(ascending=False)

importance
```

```
Volatility    0.360744
Return        0.268574
MA10          0.199126
MA5           0.171556
dtype: float64
```

```
importance.plot(kind="bar", figsize=(8,4), title="Feature Importance")
plt.show()
```