Stock Price Forecasting Using ARIMA Model in Python

Table of Contents

1. Introduction
2. Learning Objectives
3. Prerequisites
4. Project Overview
5. Module 1: Understanding Time Series Data and the ARIMA Model
    - 5.1 What is a Time Series?
    - 5.2 Introduction to the ARIMA Model
    - 5.3 Components of Time Series Data
6. Module 2: Data Acquisition
    - 6.1 Installing Required Libraries
    - 6.2 Fetching Historical Stock Data
7. Module 3: Data Preprocessing
    - 7.1 Understanding the Data Structure
    - 7.2 Visualizing the Stock's Daily Closing Price
    - 7.3 Checking for Missing Values
    - 7.4 Data Transformation
8. Module 4: Exploratory Data Analysis (EDA)
    - 8.1 Statistical Summary
    - 8.2 Distribution of Closing Prices
    - 8.3 Decomposing Time Series
9. Module 5: Stationarity Testing
    - 9.1 Understanding Stationarity
    - 9.2 Augmented Dickey-Fuller Test
    - 9.3 Differencing to Achieve Stationarity
10. Module 6: Determining ARIMA Parameters
    - 10.1 Autocorrelation and Partial Autocorrelation Plots
    - 10.2 Using Auto ARIMA to Find Optimal Parameters
11. Module 7: Model Building
    - 11.1 Splitting the Data
    - 11.2 Fitting the ARIMA Model
12. Module 8: Model Diagnostics
    - 12.1 Residual Analysis
13. Module 9: Forecasting Future Stock Prices
    - 13.1 Generating Forecasts
    - 13.2 Visualizing Forecasts

14. Module 10: Model Evaluation
    - 14.1 Evaluating Forecast Accuracy
    - 14.2 Interpreting Evaluation Metrics
15. Conclusion
16. Key Takeaways
17. References

---

1. Introduction

The stock market plays a pivotal role in the global economy by enabling the exchange of corporate stocks and facilitating investment. Accurately forecasting stock prices can significantly impact investment strategies, risk management, and economic planning. In this module, we will develop a forecasting model using the **Autoregressive Integrated Moving Average (ARIMA)** model to predict future stock prices based on historical data.

This project is an introductory sample so that you can get an idea of the steps that go into working on a project. You'll notice we don't hanker much on results here — what matters is the process.

2. Learning Objectives

- **Understand** the fundamentals of time series analysis and the ARIMA model.
- **Acquire and preprocess** historical stock price data.
- **Perform** exploratory data analysis to identify trends and patterns.
- **Test and achieve** stationarity in time series data.
- **Determine** optimal ARIMA model parameters using statistical methods.
- **Build, validate, and evaluate** the ARIMA model.
- **Forecast** future stock prices and interpret the results.
- **Visualize** data and model outputs effectively.

3. Prerequisites

- Proficiency in Python programming.
- Familiarity with concepts in statistics and time series analysis.
- Understanding of financial markets and stock price data.
- Libraries: `pandas`, `numpy`, `matplotlib`, `seaborn`, `statsmodels`, `scikit-learn`, `pmdarima`, `yfinance`.

4. Project Overview

We will:

- **Select** a publicly traded company (e.g., Apple Inc., ticker symbol: AAPL).
- **Fetch** historical stock price data.
- **Conduct** data cleaning and preprocessing.
- **Perform** exploratory data analysis.
- **Test for** stationarity and transform data if necessary.
- **Determine** the ARIMA model parameters (p, d, q).
- **Build** and **diagnose** the ARIMA model.
- **Forecast** future stock prices.
- **Evaluate** the model's performance using statistical metrics.
- **Visualize** and **interpret** the results.

5. Module 1: Understanding Time Series Data and the ARIMA Model

5.1 What is a Time Series?

A **time series** is a sequence of data points collected or recorded at successive points in time, usually at uniform intervals. Time series analysis involves understanding the underlying structures and functions that produce the observations.

5.2 Introduction to the ARIMA Model

The **Autoregressive Integrated Moving Average (ARIMA)** model is a popular statistical method for time series forecasting. It combines three components:

- **Autoregression (AR)**: Involves regressing the variable on its own lagged (prior) values.
- **Integrated (I)**: Involves differencing the data to achieve stationarity.
- **Moving Average (MA)**: Involves modeling the error term as a linear combination of error terms occurring contemporaneously and at various times in the past.

5.3 Components of Time Series Data

Time series data can typically be decomposed into four components:

1. **Trend**: The long-term movement in the data.
2. **Seasonality**: The repeating short-term cycle in the series.
3. **Cyclic**: Long-term oscillations around the trend.
4. **Irregular (Residual/Noise)**: Random variations that cannot be explained by the model.

6. Module 2: Data Acquisition

6.1 Installing Required Libraries

Ensure all necessary libraries are installed.

python
```
!pip install yfinance pandas numpy matplotlib seaborn statsmodels
scikit-learn pmdarima
```

6.2 Fetching Historical Stock Data

We will use the `yfinance` library to fetch historical stock data for Apple Inc. (AAPL).

```
import yfinance as yf
import pandas as pd

# Define the ticker symbol
ticker = 'AAPL'

# Fetch data
stock_data = yf.download(ticker, start='2015-01-01', end='2023-12-31')

# Display the first few rows
stock_data.head()
```

Output:

Copy code
```
[**********************100%**********************]  1 of 1 completed
```

7. Module 3: Data Preprocessing

7.1 Understanding the Data Structure

Let's examine the data types and summary statistics.

```
# Check data types
stock_data.info()

# Summary statistics
stock_data.describe()
```

7.2 Visualizing the Stock's Daily Closing Price

Plot the closing price to get an initial sense of the data.

```
import matplotlib.pyplot as plt

# Plot closing price
plt.figure(figsize=(12,6))
plt.plot(stock_data['Close'])
plt.title('Apple Inc. Closing Price')
plt.xlabel('Date')
plt.ylabel('Closing Price ($)')
plt.grid(True)
plt.show()
```

7.3 Checking for Missing Values

```
# Check for missing values
print(stock_data.isnull().sum())

# Fill missing values if any
stock_data.fillna(method='ffill', inplace=True)
```

7.4 Data Transformation

We will focus on the 'Close' price and apply log transformation to stabilize variance.

```
import numpy as np

# Extract 'Close' price
df_close = stock_data['Close']

# Apply log transformation
df_log = np.log(df_close)

# Plot the transformed data
plt.figure(figsize=(12,6))
plt.plot(df_log)
plt.title('Log Transformed Closing Price of Apple Inc.')
plt.xlabel('Date')
plt.ylabel('Log of Closing Price')
plt.grid(True)
plt.show()
```

8. Module 4: Exploratory Data Analysis (EDA)

8.1 Statistical Summary

Get a statistical summary of the closing prices.

```
df_close.describe()
```

8.2 Distribution of Closing Prices

Plot the distribution to understand the skewness and kurtosis.

```
# Density plot
plt.figure(figsize=(12,6))
```

```
df_close.plot(kind='kde')
plt.title('Density Plot of Closing Prices')
plt.xlabel('Closing Price ($)')
plt.grid(True)
plt.show()
```

8.3 Decomposing Time Series

Decompose the time series to observe trend and seasonality.

```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df_log, model='multiplicative',
period=252)  # Approximate trading days in a year

# Plot decomposition
result.plot()
plt.show()
```

9. Module 5: Stationarity Testing

9.1 Understanding Stationarity

A stationary time series has constant mean and variance over time. Most time series models assume stationarity.

9.2 Augmented Dickey-Fuller Test

Use the ADF test to check for stationarity.

```
from statsmodels.tsa.stattools import adfuller
```

```
def test_stationarity(timeseries):
    # Rolling statistics
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()

    # Plot rolling statistics
    plt.figure(figsize=(12,6))
    plt.plot(timeseries, label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label='Rolling Std')
    plt.legend()
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    # Perform ADF test
    print('Results of Augmented Dickey-Fuller Test:')
    adf_test = adfuller(timeseries, autolag='AIC')
    adf_output = pd.Series(adf_test[0:4], index=['Test
Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key, value in adf_test[4].items():
        adf_output['Critical Value (%s)' % key] = value
    print(adf_output)

# Test stationarity
test_stationarity(df_log)
```

9.3 Differencing to Achieve Stationarity

If the series is non-stationary, apply differencing.

```
# Differencing
df_log_diff = df_log.diff().dropna()

# Test stationarity again
test_stationarity(df_log_diff)
```

10. Module 6: Determining ARIMA Parameters

10.1 Autocorrelation and Partial Autocorrelation Plots

Plot ACF and PACF to identify potential values of p and q.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# ACF and PACF plots
fig, axes = plt.subplots(1, 2, figsize=(16,4))
plot_acf(df_log_diff, lags=50, ax=axes[0])
plot_pacf(df_log_diff, lags=50, ax=axes[1])
plt.show()
```

10.2 Using Auto ARIMA to Find Optimal Parameters

Use `pmdarima`'s `auto_arima` to automatically determine the optimal (p, d, q) parameters.

```
import pmdarima as pm

# Auto ARIMA
model_autoARIMA = pm.auto_arima(df_log, start_p=1, start_q=1,
                                test='adf',       # use adf test to
find optimal 'd'
                                max_p=5, max_q=5, # maximum p and q
                                m=1,              # frequency of
series
                                d=None,           # let model
determine 'd'
                                seasonal=False,   # No Seasonality
                                start_P=0,
                                D=0,
                                trace=True,
                                error_action='ignore',
```

```
                                suppress_warnings=True,
                                stepwise=True)

print(model_autoARIMA.summary())

# Plot diagnostics
model_autoARIMA.plot_diagnostics(figsize=(15,8))
plt.show()
```

11. Module 7: Model Building

11.1 Splitting the Data

Divide the data into training and testing sets.

```
# Split data
train_size = int(len(df_log) * 0.9)
train_data, test_data = df_log[:train_size], df_log[train_size:]

# Plot training and testing data
plt.figure(figsize=(12,6))
plt.plot(train_data, label='Training Data')
plt.plot(test_data, label='Test Data')
plt.title('Train/Test Split')
plt.xlabel('Date')
plt.ylabel('Log of Closing Price')
plt.legend()
plt.show()
```

11.2 Fitting the ARIMA Model

Fit the ARIMA model using the optimal parameters from `auto_arima`.

```
from statsmodels.tsa.arima.model import ARIMA

# Build and fit the model
```

```
model = ARIMA(train_data, order=model_autoARIMA.order)
model_fit = model.fit()
print(model_fit.summary())
```

12. Module 8: Model Diagnostics

12.1 Residual Analysis

Analyze residuals to check the adequacy of the model.

```
# Residuals plot
residuals = model_fit.resid

# Plot residuals
plt.figure(figsize=(12,6))
plt.plot(residuals)
plt.title('Residuals')
plt.xlabel('Date')
plt.ylabel('Residual')
plt.show()

# Density plot of residuals
plt.figure(figsize=(12,6))
residuals.plot(kind='kde')
plt.title('Density Plot of Residuals')
plt.show()

# QQ plot
import scipy.stats as stats
import pylab

stats.probplot(residuals, dist="norm", plot=pylab)
pylab.show()
```

13. Module 9: Forecasting Future Stock Prices

13.1 Generating Forecasts

Forecast the closing prices for the test data period.

```
# Forecast
forecast_steps = len(test_data)
forecast_result = model_fit.get_forecast(steps=forecast_steps)
forecast = forecast_result.predicted_mean
conf_int = forecast_result.conf_int()

# Convert forecast to original scale
forecast_series = np.exp(forecast)
lower_series = np.exp(conf_int.iloc[:, 0])
upper_series = np.exp(conf_int.iloc[:, 1])
```

13.2 Visualizing Forecasts

Plot the forecasted values against the actual values.

```
# Actual test data in original scale
test_data_actual = np.exp(test_data)

# Plot
plt.figure(figsize=(12,6))
plt.plot(df_close, label='Actual Stock Price')
plt.plot(forecast_series, color='red', label='Forecasted Stock Price')
plt.fill_between(lower_series.index, lower_series, upper_series,
color='pink', alpha=0.5)
plt.title('Forecast vs Actuals')
plt.xlabel('Date')
plt.ylabel('Stock Price ($)')
plt.legend()
plt.show()
```

14. Module 10: Model Evaluation

14.1 Evaluating Forecast Accuracy

Calculate evaluation metrics.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Calculate metrics
mse = mean_squared_error(test_data_actual, forecast_series)
mae = mean_absolute_error(test_data_actual, forecast_series)
rmse = np.sqrt(mse)
mape = np.mean(np.abs(forecast_series -
test_data_actual)/np.abs(test_data_actual))*100

print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'Mean Absolute Percentage Error (MAPE): {mape:.2f}%')
```

14.2 Interpreting Evaluation Metrics

- **MAE**: Average absolute difference between forecasted and actual values.
- **MSE**: Average squared difference between forecasted and actual values.
- **RMSE**: Square root of MSE; interpretable in the same units as the data.
- **MAPE**: Average absolute percentage error; a relative measure of prediction accuracy.

15. Conclusion

In this project, we developed an ARIMA model to forecast the stock prices of Apple Inc. We started by understanding the fundamentals of time series data and the ARIMA model. After acquiring and preprocessing the data, we performed exploratory data analysis to identify trends and patterns. We tested for stationarity and made the series stationary through differencing. Using statistical methods and `auto_arima`, we determined the optimal parameters for the ARIMA model. We then built the model, performed diagnostics, and generated forecasts. Finally, we evaluated the model's performance using various statistical metrics.

16. Key Takeaways

- **ARIMA Models**: Effective for forecasting time series data by capturing autocorrelations in the data.
- **Stationarity**: Essential for time series modeling; non-stationary data can lead to unreliable forecasts.
- **Model Evaluation**: Using metrics like MAE, MSE, RMSE, and MAPE helps in assessing the model's predictive performance.
- **Data Preprocessing**: Critical step to handle missing values, transform data, and achieve stationarity.

17. References

- **YFinance Documentation**: https://pypi.org/project/yfinance/
- **Statsmodels Documentation**: https://www.statsmodels.org/
- **Pmdarima Documentation**: http://alkaline-ml.com/pmdarima/
- **Time Series Analysis Resources**:
    - Hyndman, R.J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. OTexts.
    - Chatfield, C. (2003). *The Analysis of Time Series: An Introduction*. Chapman and Hall/CRC.