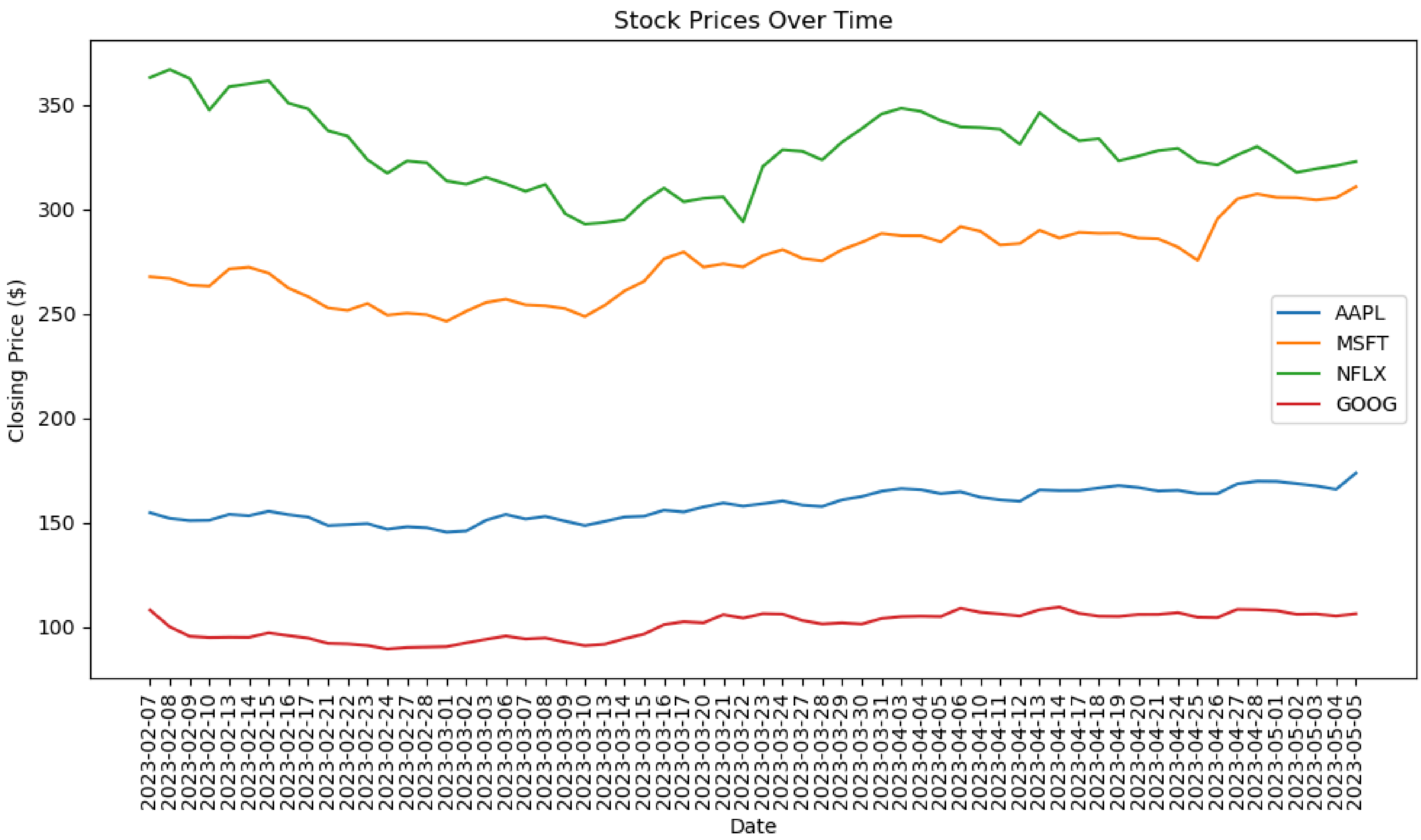
In [2]:
```
import pandas as pd

# Load the dataset
df = pd.read_csv('stocks.csv')

# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())

# Drop any rows with missing values (if any)
df = df.dropna()

# Display the first few rows to confirm it loaded correctly
print("\nFirst 5 rows of the dataset:")
print(df.head())
```

```
Missing values in each column:
Ticker       0
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64

First 5 rows of the dataset:
  Ticker        Date        Open        High         Low       Close  \
0   AAPL  2023-02-07  150.639999  155.229996  150.639999  154.649994
1   AAPL  2023-02-08  153.880005  154.580002  151.169998  151.919998
2   AAPL  2023-02-09  153.779999  154.330002  150.419998  150.869995
3   AAPL  2023-02-10  149.460007  151.339996  149.220001  151.009995
4   AAPL  2023-02-13  150.949997  154.259995  150.919998  153.850006

    Adj Close    Volume
0  154.414230  83322600
1  151.688400  64120100
2  150.639999  56007100
3  151.009995  57450700
4  153.850006  62199000
```

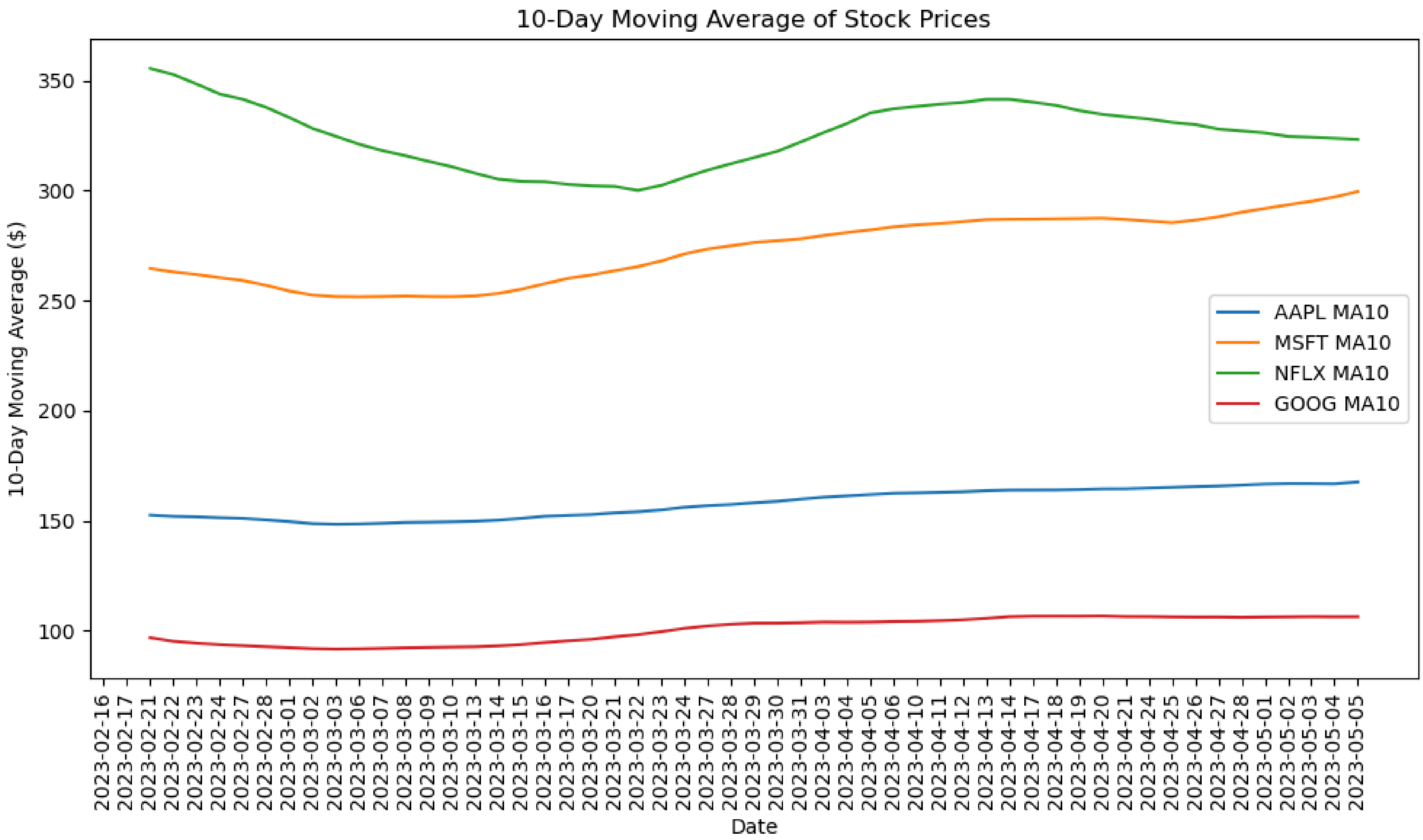
In [4]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('stocks.csv')

# Get unique company tickers
companies = df['Ticker'].unique()

# Plot closing prices for each company
plt.figure(figsize=(10, 6))
for company in companies:
    company_data = df[df['Ticker'] == company]
    plt.plot(company_data['Date'], company_data['Close'], label=company)
```

```python
plt.xlabel('Date')
plt.ylabel('Closing Price ($)')
plt.title('Stock Prices Over Time')
plt.legend()
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig('stock_prices.png')
```



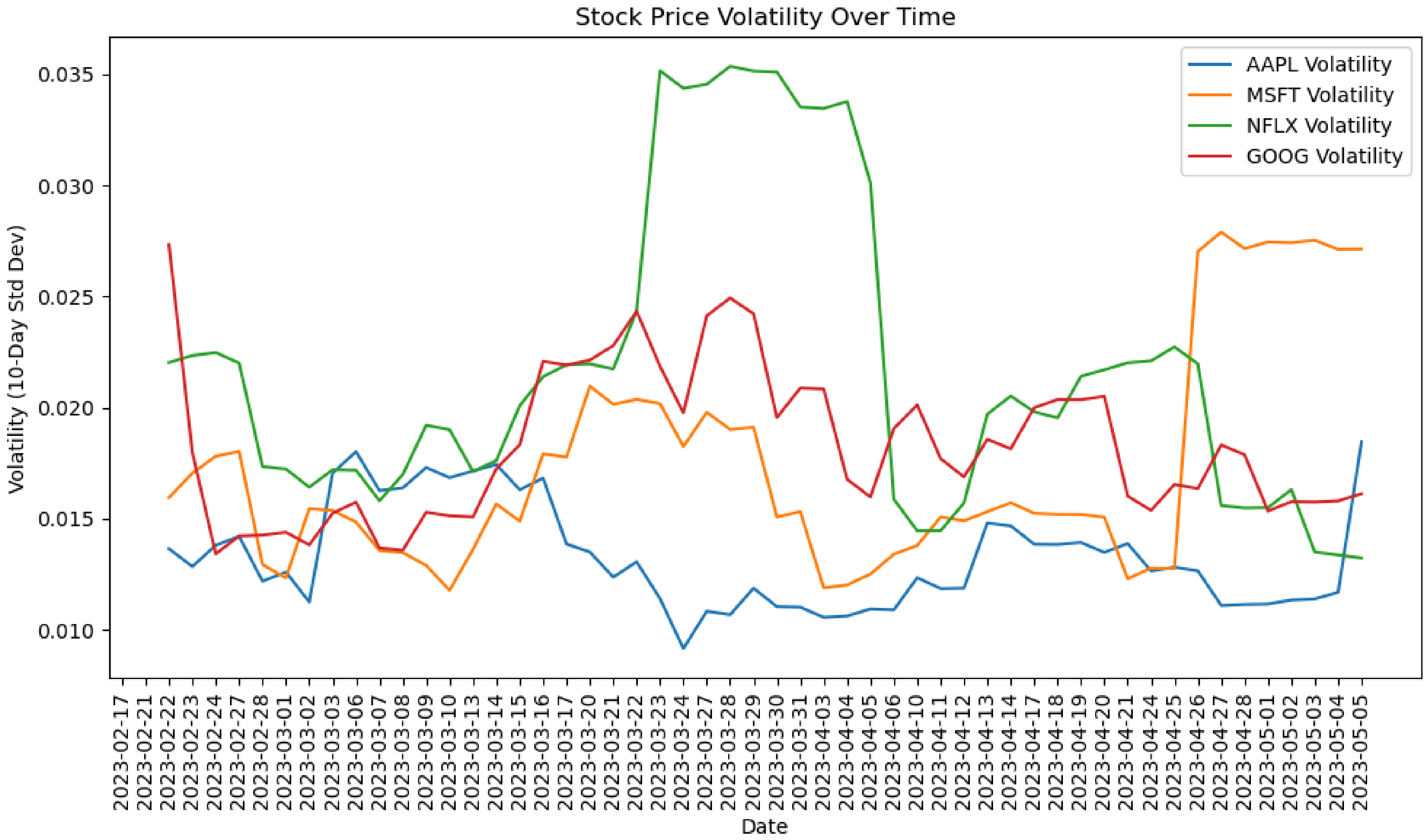
In [6]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('stocks.csv')

# Get unique company tickers
companies = df['Ticker'].unique()

# Plot 10-day moving averages
plt.figure(figsize=(10, 6))
for company in companies:
    company_data = df[df['Ticker'] == company].copy()
    company_data['MA10'] = company_data['Close'].rolling(window=10).mean()
    plt.plot(company_data['Date'], company_data['MA10'], label=f'{company} MA10')

plt.xlabel('Date')
plt.ylabel('10-Day Moving Average ($)')
plt.title('10-Day Moving Average of Stock Prices')
plt.legend()
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig('moving_averages.png')
```



In [8]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('stocks.csv')

# Calculate daily returns
```

```python
df['Return'] = df.groupby('Ticker')['Close'].pct_change()

# Get unique company tickers
companies = df['Ticker'].unique()

# Plot 10-day volatility
plt.figure(figsize=(10, 6))
for company in companies:
    company_data = df[df['Ticker'] == company].copy()
    company_data['Volatility'] = company_data['Return'].rolling(window=10).std()
    plt.plot(company_data['Date'], company_data['Volatility'], label=f'{company} Volatility')

plt.xlabel('Date')
plt.ylabel('Volatility (10-Day Std Dev)')
plt.title('Stock Price Volatility Over Time')
plt.legend()
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig('volatility.png')
```



In [9]:
```
import pandas as pd

# Load the dataset
df = pd.read_csv('stocks.csv')

# Pivot the data (dates as rows, tickers as columns)
pivot_df = df.pivot(index='Date', columns='Ticker', values='Close')
```

```
# Calculate correlation matrix
correlation_matrix = pivot_df.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
Ticker      AAPL      GOOG      MSFT      NFLX
Ticker
AAPL    1.000000  0.901662  0.953037  0.154418
GOOG    0.901662  1.000000  0.884527  0.201046
MSFT    0.953037  0.884527  1.000000  0.191273
NFLX    0.154418  0.201046  0.191273  1.000000
```

In [10]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv('stocks.csv')

# Function to create features (past 5 days) and target (next day)
def create_features(data, N):
    X, y = [], []
    for i in range(N, len(data)):
        X.append(data[i-N:i])
        y.append(data[i])
    return np.array(X), np.array(y)

# Select AAPL data
aapl_data = df[df['Ticker'] == 'AAPL']['Close'].values

# Create features and target
X, y = create_features(aapl_data, 5)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
```

```
model.fit(X_train, y_train)

# Evaluate the model
score = model.score(X_test, y_test)
print(f'AAPL Model Score: {score}')
```

```
AAPL Model Score: 0.8543249947858749
```

In [13]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv('stocks.csv')

# Function to create features (past 5 days) and target (next day)
def create_features(data, N):
    X, y = [], []
    for i in range(N, len(data)):
        X.append(data[i-N:i])
        y.append(data[i])
    return np.array(X), np.array(y)

# Select AAPL data
aapl_data = df[df['Ticker'] == 'AAPL']['Close'].values

# Create features and target
X, y = create_features(aapl_data, 5)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
lr_model = LinearRegression()
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train models
lr_model.fit(X_train, y_train)
```

```
rf_model.fit(X_train, y_train)

# Make predictions
lr_pred = lr_model.predict(X_test)
rf_pred = rf_model.predict(X_test)

# Evaluate models
print("Linear Regression Performance:")
print(f"Mean Squared Error: {mean_squared_error(y_test, lr_pred):.2f}")
print(f"R² Score: {r2_score(y_test, lr_pred):.2f}")

print("\nRandom Forest Performance:")
print(f"Mean Squared Error: {mean_squared_error(y_test, rf_pred):.2f}")
print(f"R² Score: {r2_score(y_test, rf_pred):.2f}")
```

```
Linear Regression Performance:
Mean Squared Error: 5.73
R² Score: 0.85

Random Forest Performance:
Mean Squared Error: 8.55
R² Score: 0.78
```

In [14]:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_csv('stocks.csv')

# Select AAPL data and add features
aapl_df = df[df['Ticker'] == 'AAPL'].copy()
aapl_df['MA10'] = aapl_df['Close'].rolling(window=10).mean()
aapl_df['Return'] = aapl_df['Close'].pct_change()
aapl_df = aapl_df.dropna()  # Drop rows with NaN values

# Create features (past 5 days of Close, MA10, Return) and target
def create_features(data, N):
    X, y = [], []
```

```
    for i in range(N, len(data)):
        X.append(np.concatenate([
            data['Close'].values[i-N:i],
            data['MA10'].values[i-N:i],
            data['Return'].values[i-N:i]
        ]))
        y.append(data['Close'].values[i])
    return np.array(X), np.array(y)

# Prepare data
X, y = create_features(aapl_df, 5)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Initialize Random Forest
rf_model = RandomForestRegressor(random_state=42)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model
best_rf = grid_search.best_estimator_

# Make predictions
y_pred = best_rf.predict(X_test)

# Evaluate the tuned model
print("Tuned Random Forest Performance:")
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")
print(f"R² Score: {r2_score(y_test, y_pred):.2f}")
```

```
Tuned Random Forest Performance:
Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
Mean Squared Error: 6.43
R² Score: 0.71
```

In [ ]:

```

```