y96jeami9

August 21, 2023

Name: Goktug Akca

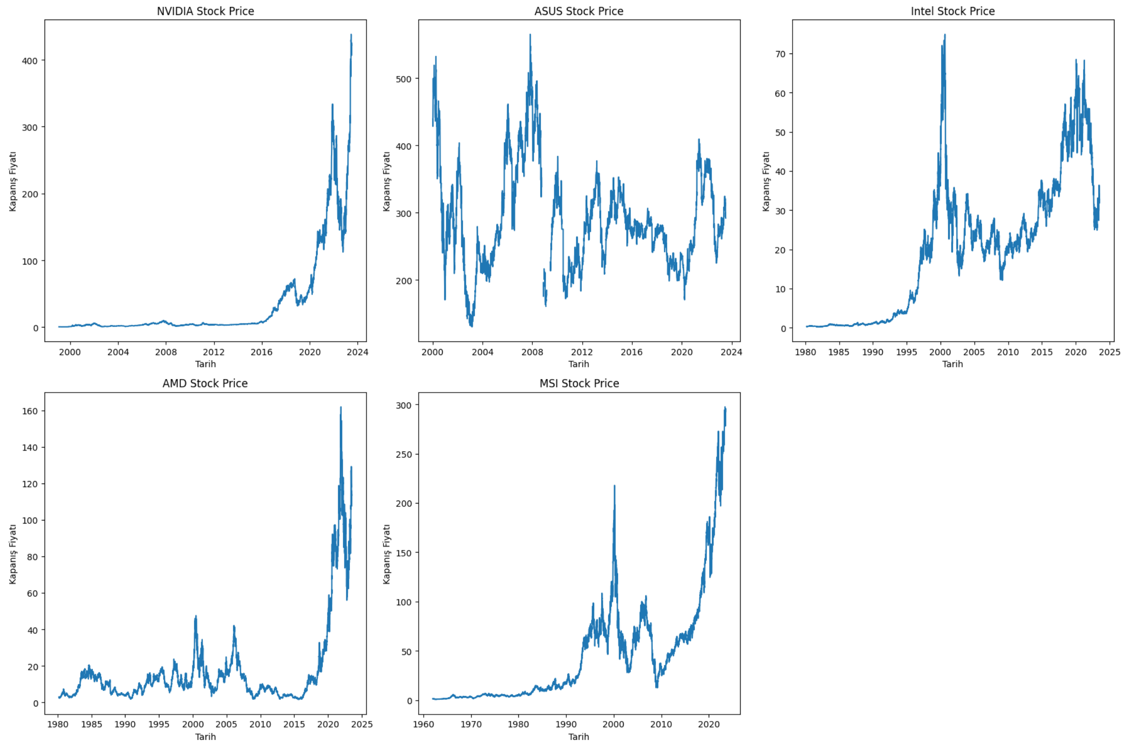
ID: 191101073

Course: BIL570 /BIL470

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import train_test_split
from math import sqrt
```

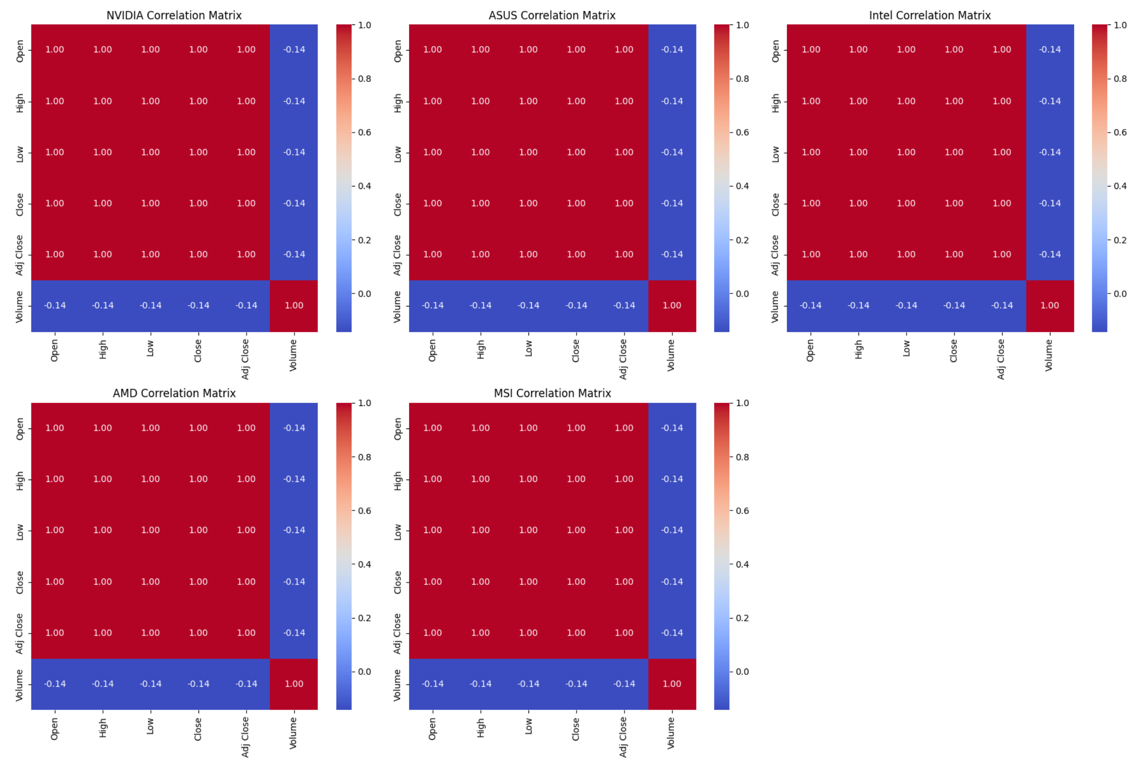
[2]:
```python
nvidia=pd.read_csv("NVIDIA (1999 -11.07.2023).csv")
asus=pd.read_csv("ASUS (2000 - 11.07.2023).csv")
intel=pd.read_csv("INTEL (1980 - 11.07.2023).csv")
amd=pd.read_csv("AMD (1980 -11.07.2023).csv")
msi=pd.read_csv("Motorola Solutions (MSI) (1962 -11.07.2023).csv")

# Date sütununu date time formatına çevirdim
nvidia['Date'] = pd.to_datetime(nvidia['Date'])
asus['Date'] = pd.to_datetime(asus['Date'])
intel['Date'] = pd.to_datetime(intel['Date'])
amd['Date'] = pd.to_datetime(amd['Date'])
msi['Date'] = pd.to_datetime(msi['Date'])

#her bir şirketin tarihe göre close'unu pyplot ile grafikleştridim

plt.figure(figsize=(18, 12))

# NVIDIA Grafik
plt.subplot(231)  # 2x3'lük bir alt-çizimde 1. grafik
plt.plot(nvidia['Date'], nvidia['Close'])
plt.title('NVIDIA Stock Price')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')

# ASUS Grafik
plt.subplot(232)  # 2x3'lük bir alt-çizimde 2. grafik
```

```python
plt.plot(asus['Date'], asus['Close'])
plt.title('ASUS Stock Price')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')

# Intel Grafik
plt.subplot(233)  # 2x3'lük bir alt-çizimde 3. grafik
plt.plot(intel['Date'], intel['Close'])
plt.title('Intel Stock Price')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')

# AMD Grafik
plt.subplot(234)  # 2x3'lük bir alt-çizimde 4. grafik
plt.plot(amd['Date'], amd['Close'])
plt.title('AMD Stock Price')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')

# MSI Grafik
plt.subplot(235)  # 2x3'lük bir alt-çizimde 5. grafik
plt.plot(msi['Date'], msi['Close'])
plt.title('MSI Stock Price')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')

# Grafikleri ayarlayın ve gösterin
plt.tight_layout()  # Grafiklerin sıkışık olmasını önlemek için kullanılır
plt.show()
```



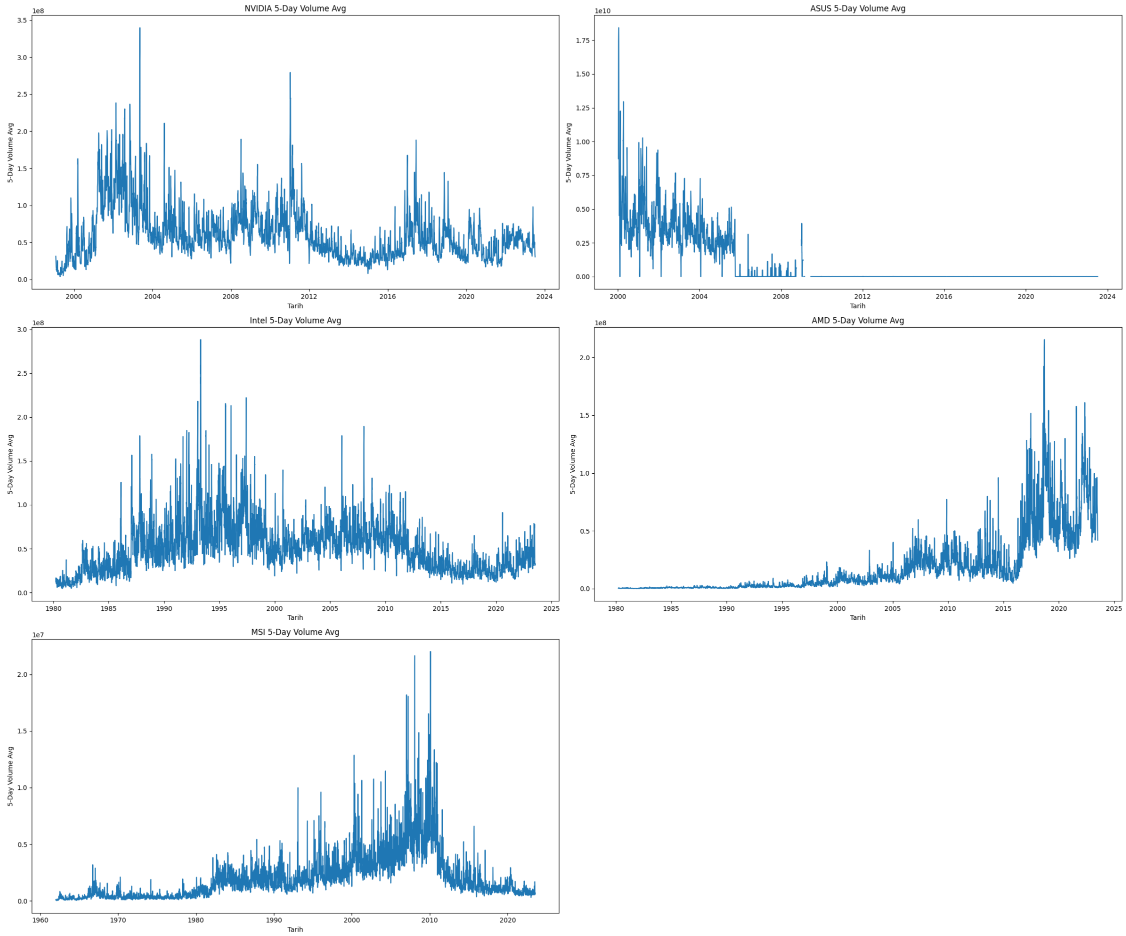
[3]:
```python
import seaborn as sns
plt.figure(figsize=(18, 12))
nvidia_x = nvidia.drop(columns=["Date"])
# NVIDIA Grafik
plt.subplot(231)  # 2x3'lük bir alt-çizimde 1. grafik
correlation_matrix = nvidia_x.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('NVIDIA Correlation Matrix')

# ASUS Grafik
asus_x = nvidia.drop(columns=["Date"])
plt.subplot(232)  # 2x3'lük bir alt-çizimde 2. grafik
correlation_matrix = asus_x.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('ASUS Correlation Matrix')

# Intel Grafik
intel_x = nvidia.drop(columns=["Date"])
plt.subplot(233)  # 2x3'lük bir alt-çizimde 3. grafik
correlation_matrix = intel_x.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('Intel Correlation Matrix')
```

```
# AMD Grafik
amd_X = nvidia.drop(columns=["Date"])
plt.subplot(234)  # 2x3'lük bir alt-çizimde 4. grafik
correlation_matrix = amd_X.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('AMD Correlation Matrix')

# MSI Grafik
msi_x = nvidia.drop(columns=["Date"])
plt.subplot(235)  # 2x3'lük bir alt-çizimde 5. grafik
correlation_matrix = msi_x.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('MSI Correlation Matrix')

# Grafikleri ayarlayın ve gösterin
plt.tight_layout()  # Grafiklerin sıkışık olmasını önlemek için kullanılır
plt.show()
```



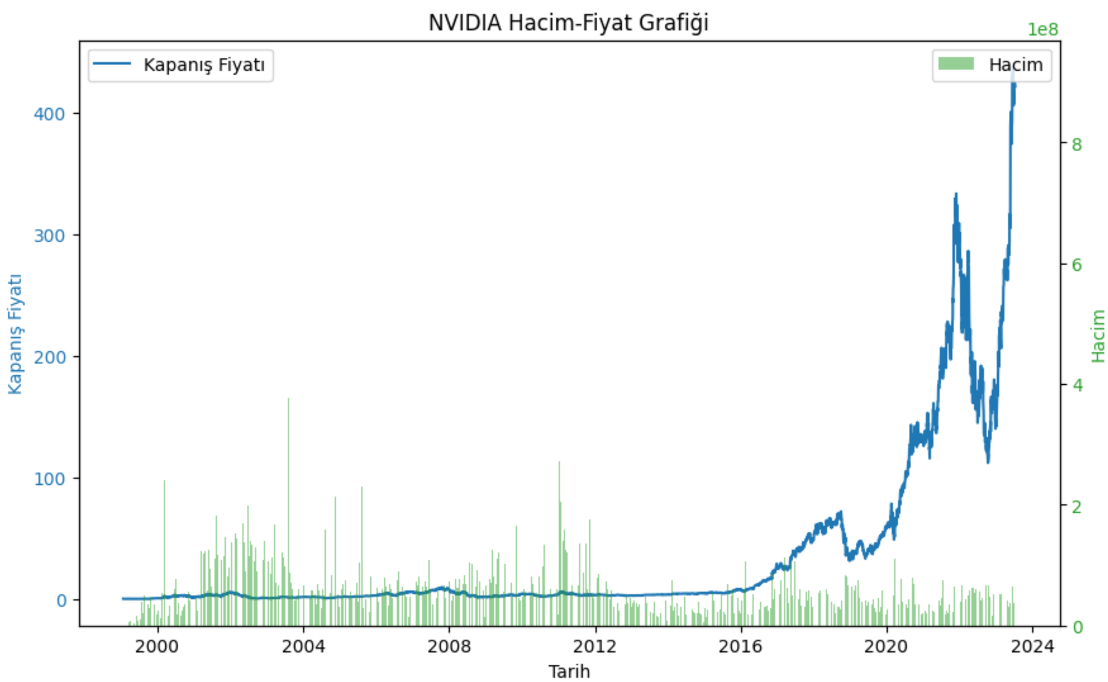
[4]:
```
#5 günlük hacim performansı diye yeni bir özellik üretmenin hacmin geçmiş performanslarına
#dayalı gelecekteki fiyat hareketlerini anlamaya yardımcı olacağını düşündüm
```

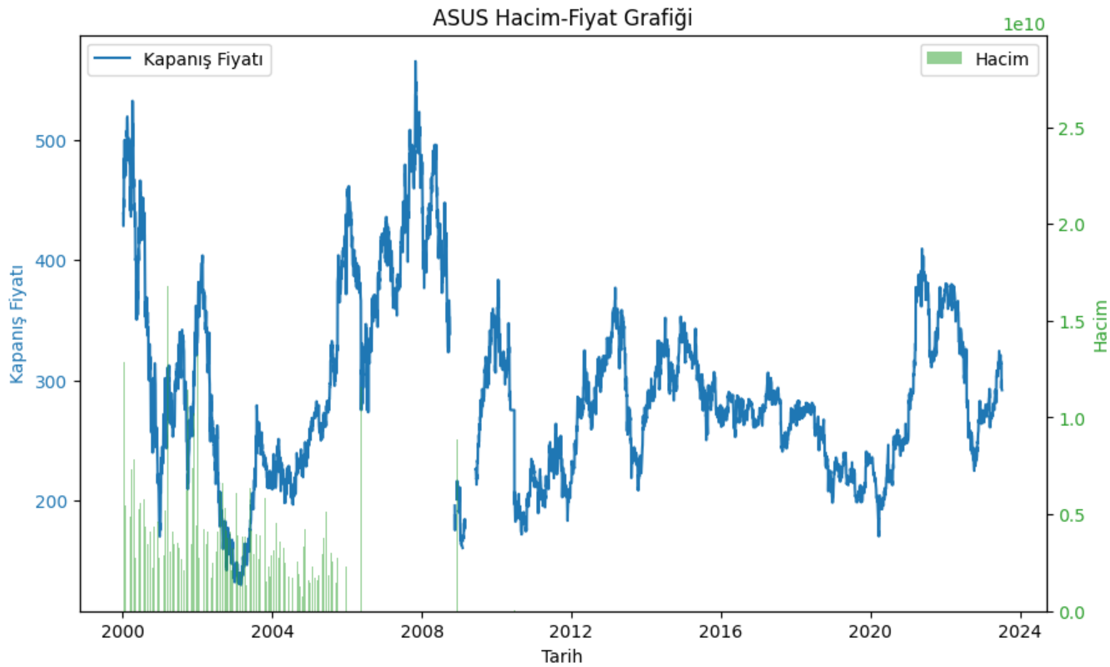
```python
plt.figure(figsize=(24, 20))
# 5 günlük ortalama hacmi hesaplayın
nvidia['5-Day Volume Avg'] = nvidia['Volume'].rolling(window=5).mean()
asus['5-Day Volume Avg'] = asus['Volume'].rolling(window=5).mean()
intel['5-Day Volume Avg'] = intel['Volume'].rolling(window=5).mean()
amd['5-Day Volume Avg'] = amd['Volume'].rolling(window=5).mean()
msi['5-Day Volume Avg'] = msi['Volume'].rolling(window=5).mean()

# NVIDIA Grafik
plt.subplot(321) # 1.grafik
plt.plot(nvidia['Date'], nvidia['5-Day Volume Avg'])
plt.title('NVIDIA 5-Day Volume Avg')
plt.xlabel('Tarih')
plt.ylabel('5-Day Volume Avg')

# ASUS Grafik
plt.subplot(322)  # 2. grafik
plt.plot(asus['Date'], asus['5-Day Volume Avg'])
plt.title('ASUS 5-Day Volume Avg')
plt.xlabel('Tarih')
plt.ylabel('5-Day Volume Avg')

# Intel Grafik
plt.subplot(323)  # 3. grafik
plt.plot(intel['Date'], intel['5-Day Volume Avg'])
plt.title('Intel 5-Day Volume Avg')
plt.xlabel('Tarih')
plt.ylabel('5-Day Volume Avg')

# AMD Grafik
plt.subplot(324)  # 4. grafik
plt.plot(amd['Date'], amd['5-Day Volume Avg'])
plt.title('AMD 5-Day Volume Avg')
plt.xlabel('Tarih')
plt.ylabel('5-Day Volume Avg')

# MSI Grafik
plt.subplot(325)  # 5. grafik
plt.plot(msi['Date'], msi['5-Day Volume Avg'])
plt.title('MSI 5-Day Volume Avg')
plt.xlabel('Tarih')
plt.ylabel('5-Day Volume Avg')

plt.tight_layout()
plt.show()
```

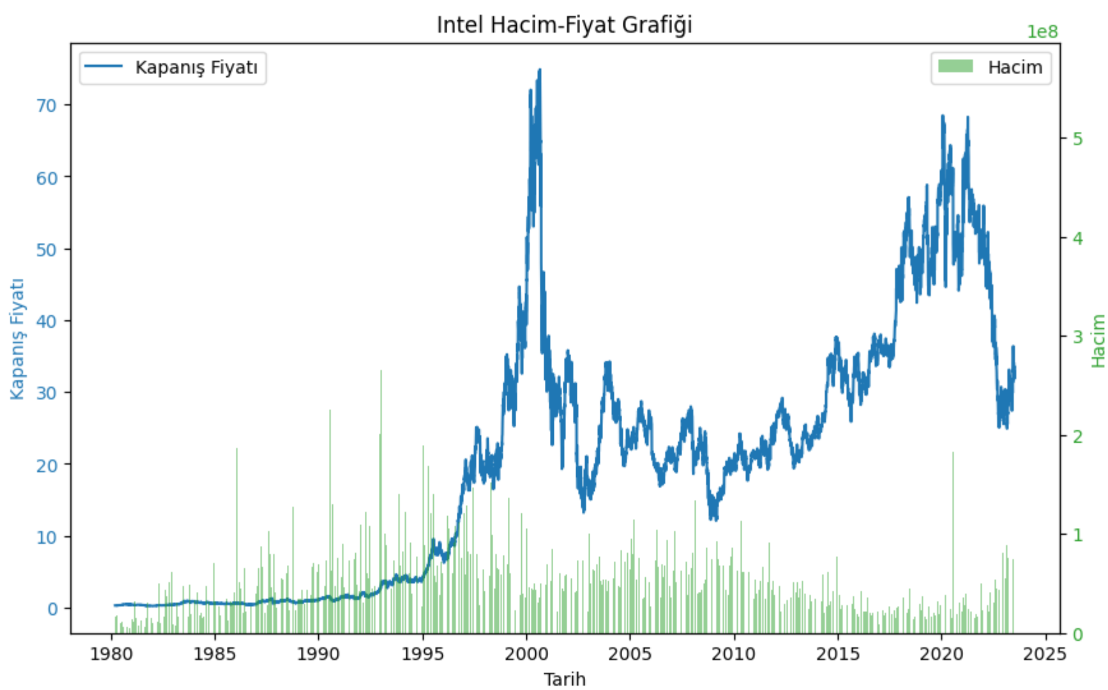


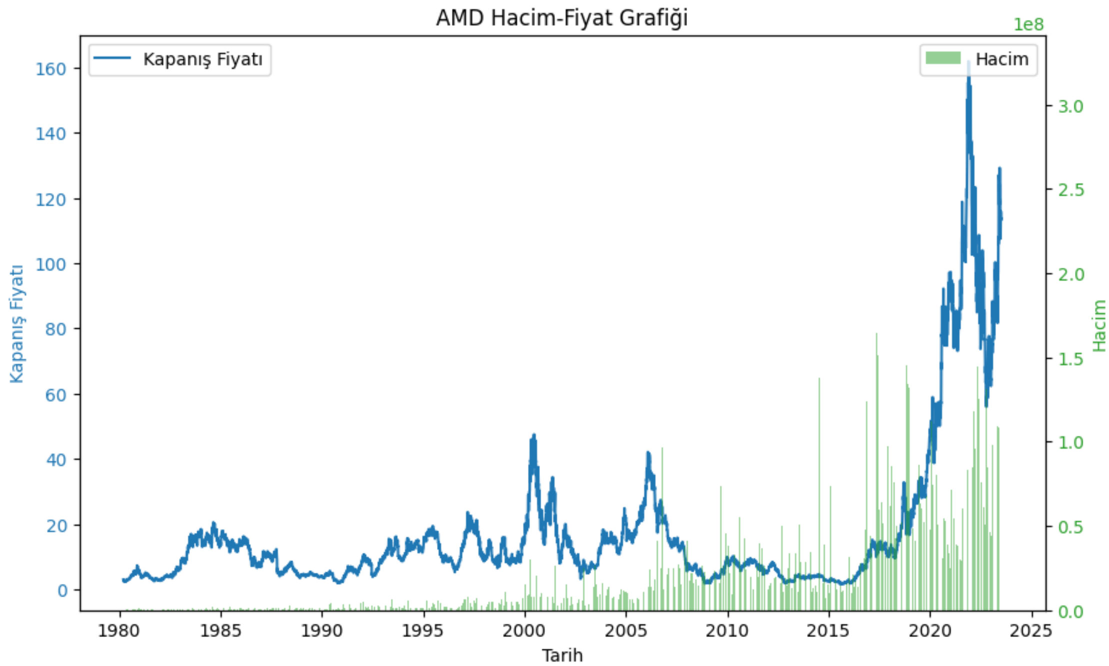
[5]:
```python
def plot_dual_axis(df, title):
    fig, ax1 = plt.subplots(figsize=(10, 6))

    # Birinci eksen (sol): Kapanış Fiyatı
    color = 'tab:blue'
    ax1.set_xlabel('Tarih')
    ax1.set_ylabel('Kapanış Fiyatı', color=color)
    ax1.plot(df['Date'], df['Close'], color=color, label='Kapanış Fiyatı')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.legend(loc='upper left')

    # İkinci eksen (sağ): Hacim
    ax2 = ax1.twinx()  # Aynı x ekseni kullan
    color = 'tab:green'
    ax2.set_ylabel('Hacim', color=color)
    ax2.bar(df['Date'], df['Volume'], color=color, alpha=0.5, label='Hacim')
    ax2.tick_params(axis='y', labelcolor=color)
    ax2.legend(loc='upper right')
```

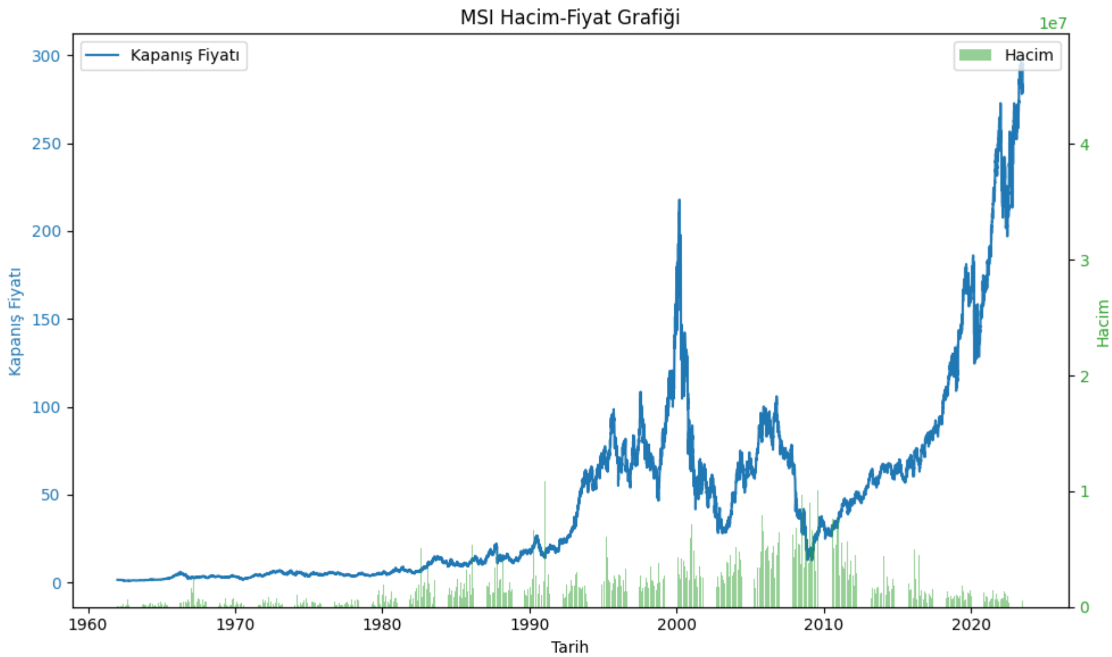
```python
    plt.title(title)
    plt.xticks(rotation=45)

# NVIDIA
plot_dual_axis(nvidia, 'NVIDIA Hacim-Fiyat Grafiği')

# ASUS
plot_dual_axis(asus, 'ASUS Hacim-Fiyat Grafiği')

# Intel
plot_dual_axis(intel, 'Intel Hacim-Fiyat Grafiği')

# AMD
plot_dual_axis(amd, 'AMD Hacim-Fiyat Grafiği')

# MSI
plot_dual_axis(msi, 'MSI Hacim-Fiyat Grafiği')

# Grafikleri ayarlayın ve gösterin
plt.tight_layout()
plt.show()
```




ASUS Hacim-Fiyat Grafiği
Kapanış Fiyatı
Hacim
1e10
500
400
300
200
2.5
2.0
1.5
1.0
0.5
0.0
2000
2004
2008
2012
2016
2020
2024
Tarih


Intel Hacim-Fiyat Grafiği
Kapanış Fiyatı
Hacim
1e8
70
60
50
40
30
20
10
0
5
4
3
2
1
0
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
Tarih

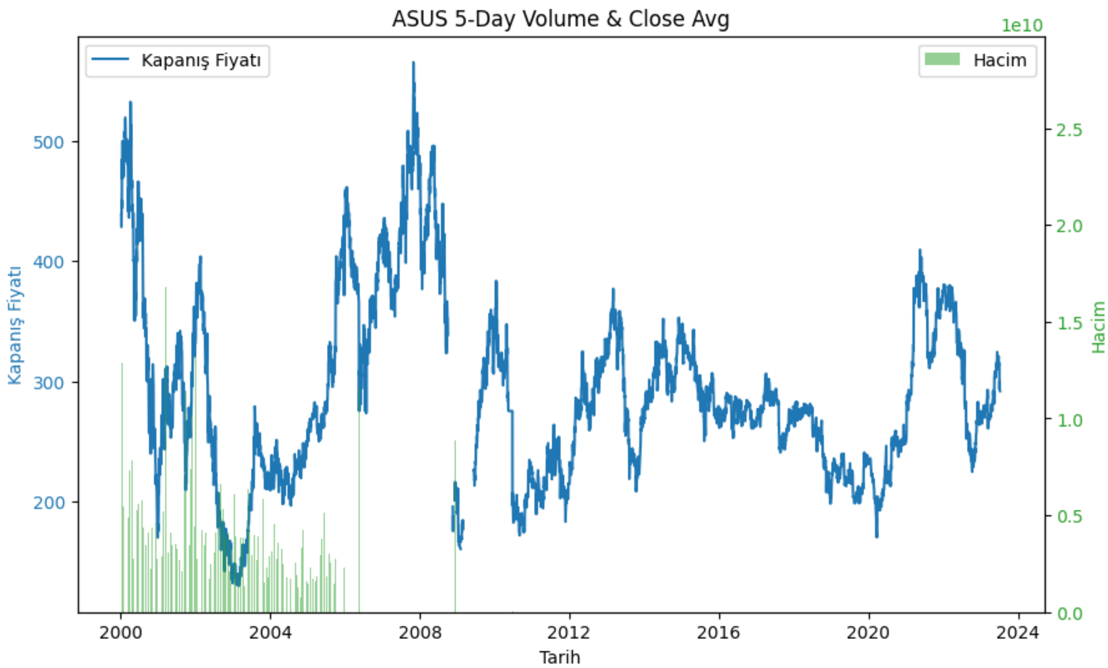




[6]:
```
# ASUS
ax = plt.subplot(322)  # 3x2'lik bir alt-çizimde 2. grafik
plot_dual_axis(asus, 'ASUS 5-Day Volume & Close Avg')
```

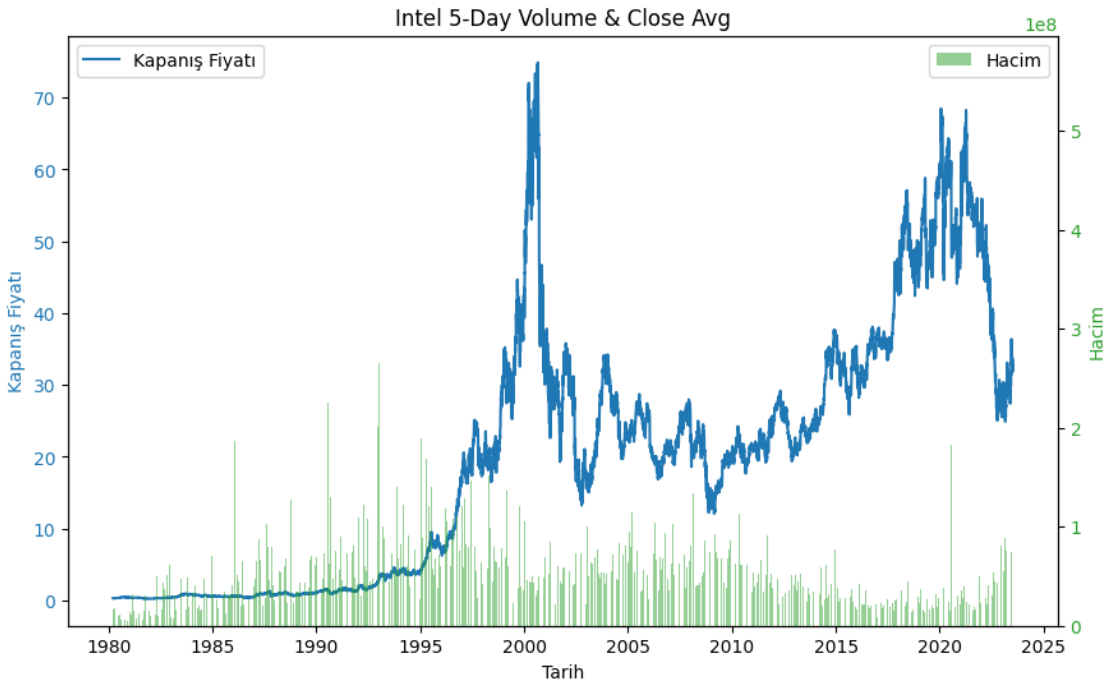
```
ax.remove()  # Üst üste binme uyarısını kaldır
```

<Figure size 640x480 with 0 Axes>



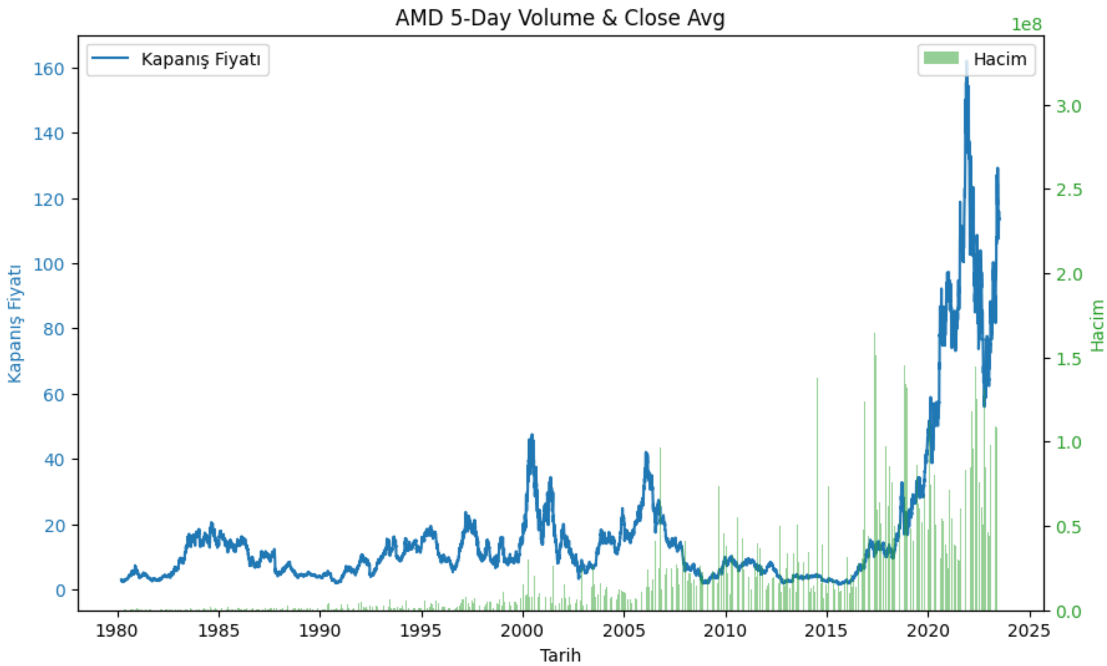
[7]:
```
# Intel
ax = plt.subplot(323)  # 3x2'lik bir alt-çizimde 3. grafik
plot_dual_axis(intel, 'Intel 5-Day Volume & Close Avg')
ax.remove()  # Üst üste binme uyarısını kaldır
```

<Figure size 640x480 with 0 Axes>



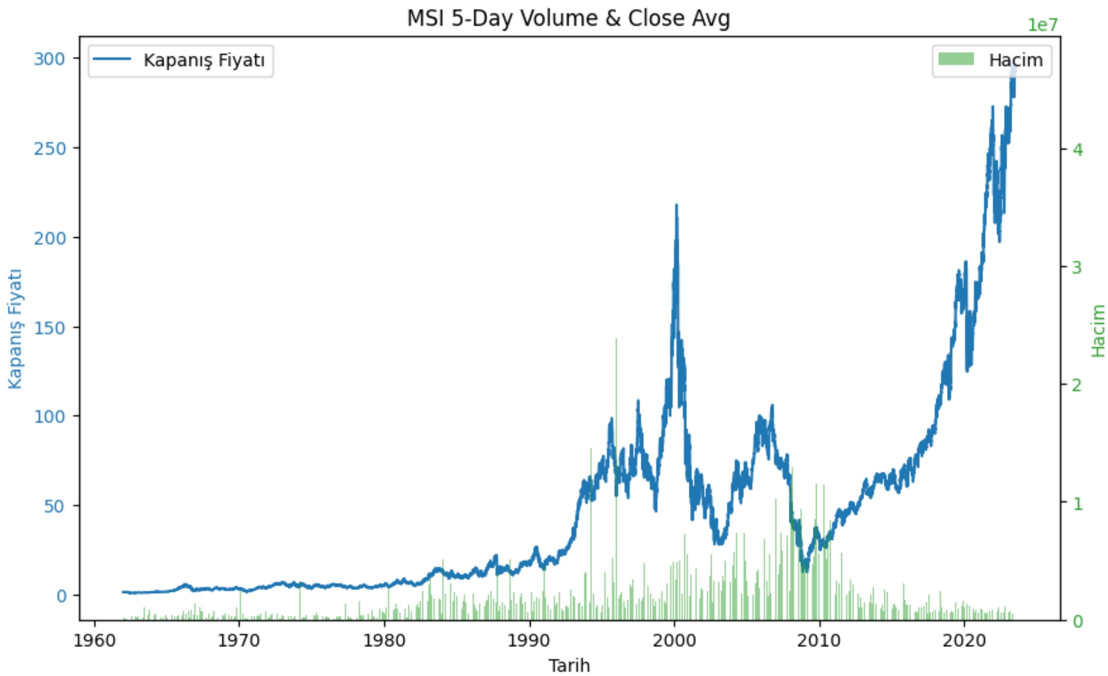
[8]:
```
# AMD
ax = plt.subplot(324)  # 3x2'lik bir alt-çizimde 4. grafik
plot_dual_axis(amd, 'AMD 5-Day Volume & Close Avg')
ax.remove()  # Üst üste binme uyarısını kaldır
```

```
<Figure size 640x480 with 0 Axes>
```



[9]:
```
# MSI
ax = plt.subplot(325)  # 3x2'lik bir alt-çizimde 5. grafik
plot_dual_axis(msi, 'MSI 5-Day Volume & Close Avg')
ax.remove()  # Üst üste binme uyarısını kaldır
```

```
<Figure size 640x480 with 0 Axes>
```

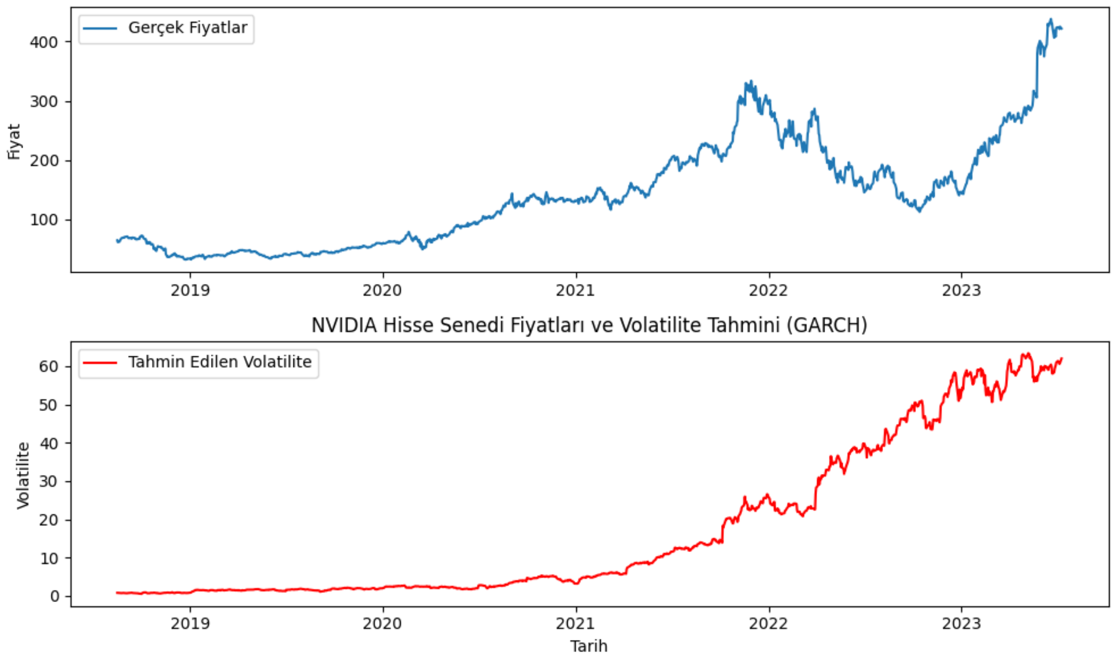


[10]:

```
from arch import arch_model

# Verileri yükleyin veya oluşturun
data = pd.read_csv("NVIDIA (1999 -11.07.2023).csv")  # NVIDIA hisse senedi␣
 ↪fiyatları örneği

# Verileri zaman serisi olarak indeksleyin
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)

# Veriyi eğitim ve test setlerine bölelim
split_index = int(0.8 * len(data))
train_data = data[:split_index]
test_data = data[split_index:]

# GARCH modelini otomatik olarak uyum sağlayacak şekilde eğitin
model = arch_model(train_data['Close'], vol='Garch')  # Otomatik p ve q seçimi
model_fit = model.fit()

# Volatilite tahminlerini yapın (sadece test verileri boyunca)
forecast_volatility = model_fit.conditional_volatility[-len(test_data):]

# Tahminleri ve gerçek değerleri görselleştirin
plt.figure(figsize=(10, 6))
```

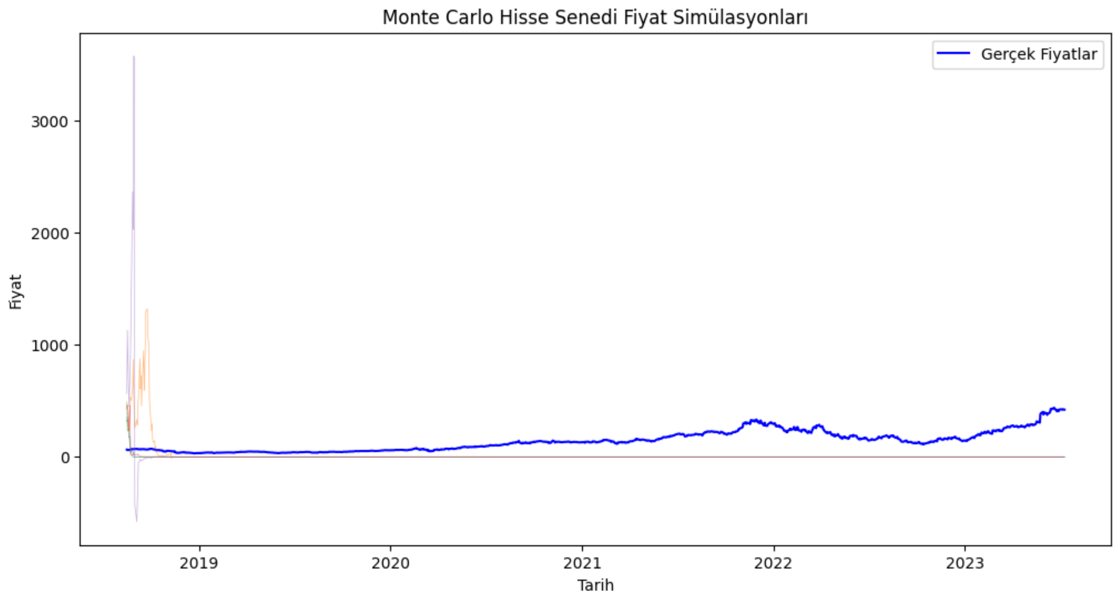
```
# Gerçek fiyatlar
plt.subplot(2, 1, 1)
plt.plot(test_data['Close'], label='Gerçek Fiyatlar')
plt.ylabel('Fiyat')
plt.legend()

# Volatilite tahminleri
plt.subplot(2, 1, 2)
plt.plot(test_data.index, forecast_volatility, color='red', label='Tahmin␣
 ↪Edilen Volatilite')
plt.xlabel('Tarih')
plt.ylabel('Volatilite')
plt.legend()

plt.title('NVIDIA Hisse Senedi Fiyatları ve Volatilite Tahmini (GARCH)')
plt.tight_layout()
plt.show()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 397098996.74810183
Iteration:      2,   Func. Count:     15,   Neg. LLF: 1409720814.83946
Iteration:      3,   Func. Count:     21,   Neg. LLF: 9057.504107863535
Iteration:      4,   Func. Count:     26,   Neg. LLF: 150536.73638871574
Iteration:      5,   Func. Count:     40,   Neg. LLF: 9911.394132105495
Iteration:      6,   Func. Count:     46,   Neg. LLF: 8926.451476723254
Iteration:      7,   Func. Count:     52,   Neg. LLF: 8916.503843830962
Iteration:      8,   Func. Count:     58,   Neg. LLF: 8910.382066384187
Iteration:      9,   Func. Count:     63,   Neg. LLF: 8907.180690977882
Iteration:     10,   Func. Count:     68,   Neg. LLF: 8905.615694128017
Iteration:     11,   Func. Count:     73,   Neg. LLF: 8905.319579202089
Iteration:     12,   Func. Count:     78,   Neg. LLF: 8905.307629895768
Iteration:     13,   Func. Count:     83,   Neg. LLF: 8905.307502554937
Iteration:     14,   Func. Count:     88,   Neg. LLF: 8905.30749070184
Iteration:     15,   Func. Count:     92,   Neg. LLF: 8905.307490343152
Optimization terminated successfully    (Exit mode 0)
            Current function value: 8905.30749070184
            Iterations: 15
            Function evaluations: 92
            Gradient evaluations: 15
```

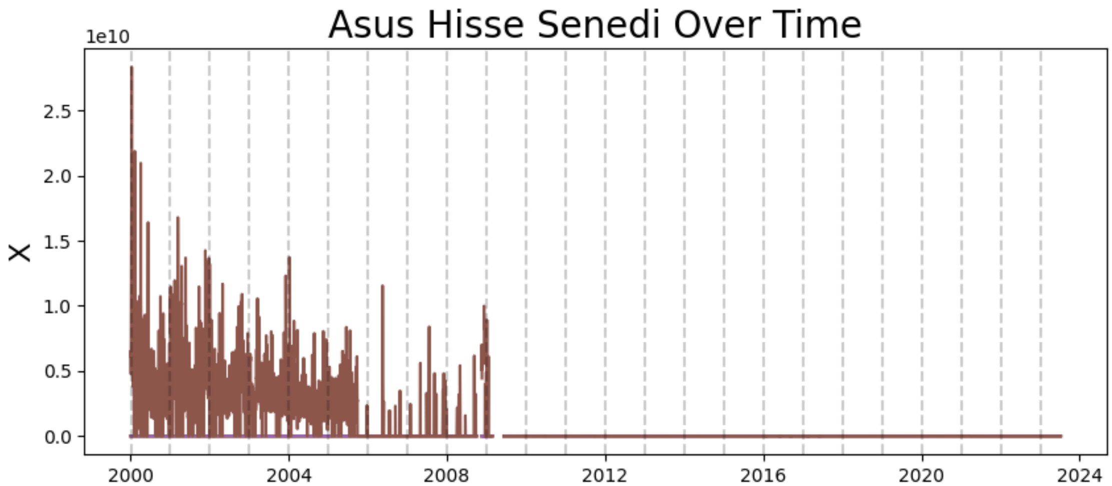


[11]:
```python
volatilities = np.clip(forecast_volatility.values.flatten(), 0.01, 0.5)

# Simülasyon parametreleri
num_simulations = 5  # Simülasyon sayısı
num_days = len(test_data)  # Test verileriyle aynı sayıda gün kullanın
simulation_results = np.empty((num_simulations, num_days))

for i in range(num_simulations):
    initial_price = test_data['Close'].iloc[-1]  # Test verilerinin son gününün kapanış fiyatıyla başlayın
    daily_returns = np.random.normal(0, volatilities, num_days)
    price_path = initial_price * np.cumprod(1 + daily_returns)
    simulation_results[i, :] = price_path

# Fiyat tahminlerini içeren DataFrame'i oluşturun
price_forecast = pd.DataFrame(simulation_results.T, columns=[f'Simulation {i+1}' for i in range(num_simulations)], index=test_data.index)

# Gerçek fiyatlar ve fiyat tahminlerini görselleştirin
plt.figure(figsize=(12, 6))
plt.plot(test_data['Close'], label='Gerçek Fiyatlar', color='blue')
for i in range(num_simulations):
    plt.plot(price_forecast.index, price_forecast[f'Simulation {i+1}'], lw=0.5, alpha=0.5)
```

```
plt.xlabel('Tarih')
plt.ylabel('Fiyat')
plt.legend()
plt.title('Monte Carlo Hisse Senedi Fiyat Simülasyonları')
plt.show()
```

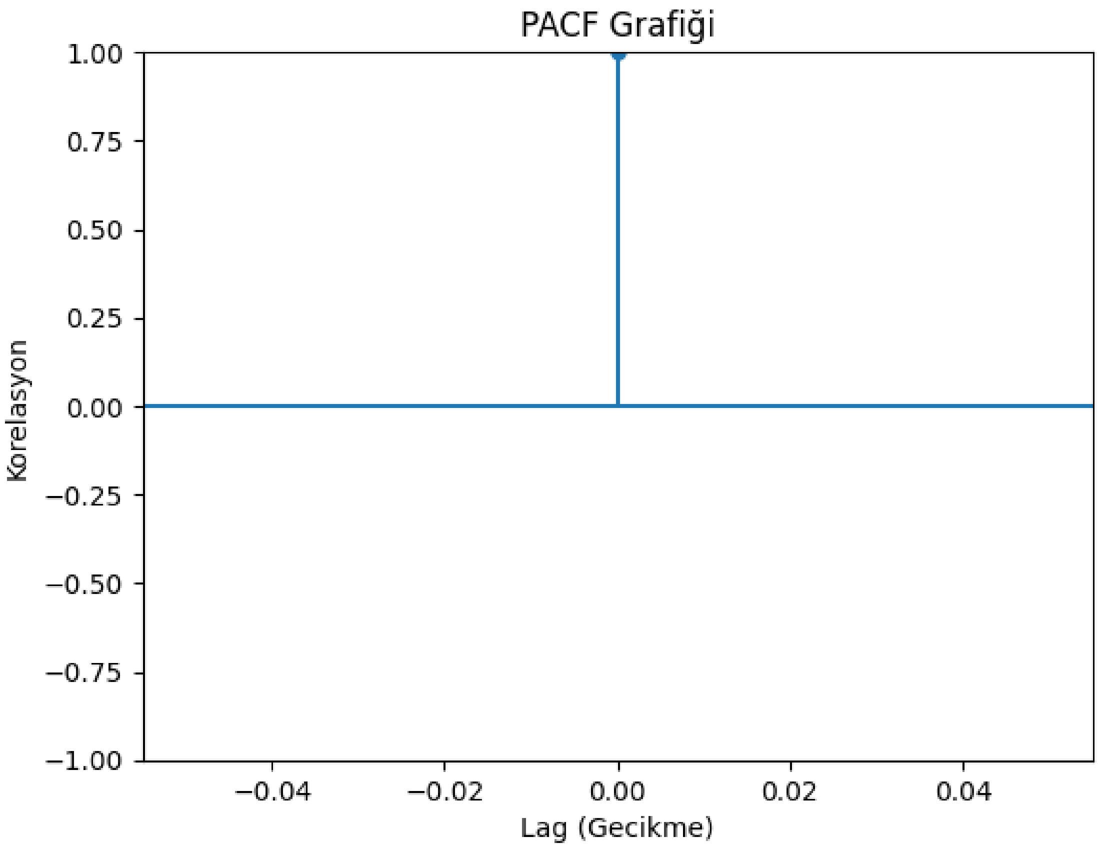


[12]:
```
asus = pd.read_csv("ASUS (2000 - 11.07.2023).csv")
asus['Date'] = pd.to_datetime(asus['Date'])
asus.set_index('Date', inplace=True)
plt.figure(figsize=(10, 4))
plt.plot(asus)
plt.title('Asus Hisse Senedi Over Time', fontsize=20)
plt.ylabel('X', fontsize=16)

# Veri çerçevesinin indeksini kullanarak yılları çizin
for year in range(asus.index.year.min(), asus.index.year.max() + 1):
    plt.axvline(asus.index[asus.index.year == year][0], color='k',␣
 ↪linestyle='--', alpha=0.2)

plt.show()
```



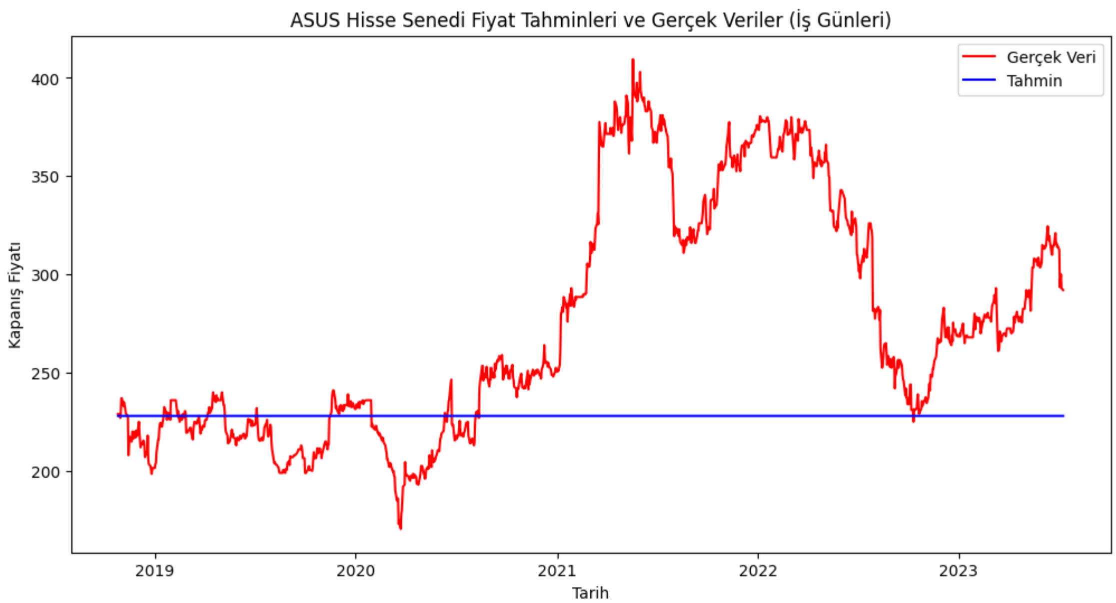
[13]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# ACF Grafiği
asus = pd.read_csv("ASUS (2000 - 11.07.2023).csv")
asus['Date'] = pd.to_datetime(asus['Date'])
asus.set_index('Date', inplace=True)
asus = asus.asfreq('B', method='ffill')
plt.figure(figsize=(12, 6))
plot_pacf(asus['Close'], lags=1)
plt.title('PACF Grafiği')
plt.xlabel('Lag (Gecikme)')
plt.ylabel('Korelasyon')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



[14]:
```
asus = pd.read_csv("ASUS (2000 - 11.07.2023).csv")
asus['Date'] = pd.to_datetime(asus['Date'])
asus.set_index('Date', inplace=True)
asus = asus.asfreq('B', method='ffill')

# Veriyi %80 eğitim ve %20 test olarak bölelim
split_ratio = 0.8
split_index = int(len(asus) * split_ratio)
train_data = asus.iloc[:split_index]
test_data = asus.iloc[split_index:]

# ARIMA modelini oluşturun ve eğitin
p, d, q = 0, 1, 0  # Örnek parametre değerleri, bu değerleri veriye göre␣
↪ayarlamalısınız
model = ARIMA(train_data['Close'], order=(p, d, q))
model_fit = model.fit()

# Test verileri üzerinde tahmin yapın
forecast_steps = len(test_data)
forecast = model_fit.forecast(steps=forecast_steps)
```

```python
# Tahminleri ve gerçek verileri aynı grafikte görselleştirin
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, test_data['Close'], color='red', label='Gerçek Veri')
plt.plot(test_data.index, forecast, color='blue', label='Tahmin')
plt.title('ASUS Hisse Senedi Fiyat Tahminleri ve Gerçek Veriler (İş Günleri)')
plt.xlabel('Tarih')
plt.ylabel('Kapanış Fiyatı')
plt.legend()
plt.show()
```



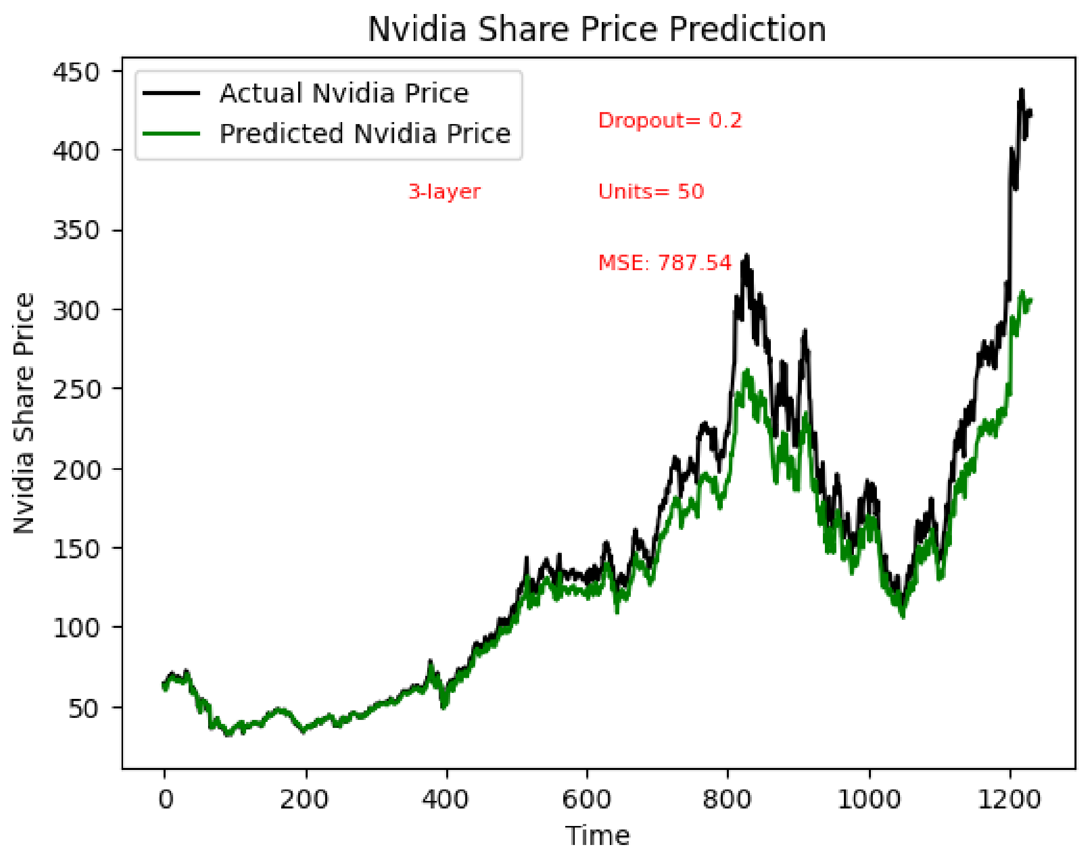
[15]:
```python
import datetime as dt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import mean_squared_error

# Veriyi yükleyin (örnek bir CSV dosyası kullanılıyor)
df = pd.read_csv("NVIDIA (1999 -11.07.2023).csv", parse_dates=['Date'])

# Tarih aralıklarını belirtin
START_DATE = df['Date'].min()
END_DATE = df['Date'].max()
```

```
# Eğitim ve test verilerini ayırmak için oranı belirtin
SPLIT_RATIO = 0.8



def load_data(start, end):
    dataframe = df.copy()
    dataframe = dataframe.loc[(dataframe['Date'] >= start) & (dataframe['Date']␣
 ↪<= end), :]
    dataframe = dataframe.rename(columns={'Closing_Price': 'Close'})
    return dataframe

data = load_data(start=START_DATE, end=END_DATE)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

# Geliştirilmiş LSTM modelini oluşturun
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    return model

# Eğitim ve test verilerini bölmek için indeksi hesaplayın
split_index = int(len(scaled_data) * SPLIT_RATIO)

# Eğitim verileri
x_train = scaled_data[:split_index - 1]
y_train = scaled_data[1:split_index]

# Test verileri
x_test = scaled_data[split_index - 1:-1]
y_test = scaled_data[split_index:]

# Modeli oluşturun ve derleyin
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)
```

```python
# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()
mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 50", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.2", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 2ms/step - loss: 6.2086e-04
Epoch 2/25
154/154 [==============================] - 0s 2ms/step - loss: 2.2891e-05
Epoch 3/25
154/154 [==============================] - 0s 2ms/step - loss: 1.8244e-05
Epoch 4/25
154/154 [==============================] - 0s 2ms/step - loss: 2.1108e-05
Epoch 5/25
154/154 [==============================] - 0s 2ms/step - loss: 2.1125e-05
Epoch 6/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6271e-05
Epoch 7/25
154/154 [==============================] - 0s 2ms/step - loss: 2.1772e-05
Epoch 8/25
154/154 [==============================] - 0s 2ms/step - loss: 1.7478e-05
Epoch 9/25
```

```
154/154 [==============================] - 0s 2ms/step - loss: 2.0010e-05
Epoch 10/25
154/154 [==============================] - 0s 2ms/step - loss: 1.9979e-05
Epoch 11/25
154/154 [==============================] - 0s 2ms/step - loss: 2.0442e-05
Epoch 12/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6860e-05
Epoch 13/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6605e-05
Epoch 14/25
154/154 [==============================] - 0s 2ms/step - loss: 1.8006e-05
Epoch 15/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6638e-05
Epoch 16/25
154/154 [==============================] - 0s 2ms/step - loss: 1.7526e-05
Epoch 17/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5617e-05
Epoch 18/25
154/154 [==============================] - 0s 2ms/step - loss: 1.9358e-05
Epoch 19/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5951e-05
Epoch 20/25
154/154 [==============================] - 0s 2ms/step - loss: 1.7856e-05
Epoch 21/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4945e-05
Epoch 22/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4443e-05
Epoch 23/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3328e-05
Epoch 24/25
154/154 [==============================] - 0s 2ms/step - loss: 1.7234e-05
Epoch 25/25
154/154 [==============================] - 0s 2ms/step - loss: 1.7346e-05
39/39 [==============================] - 1s 958us/step
```



[16]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 787.5365281906815
```

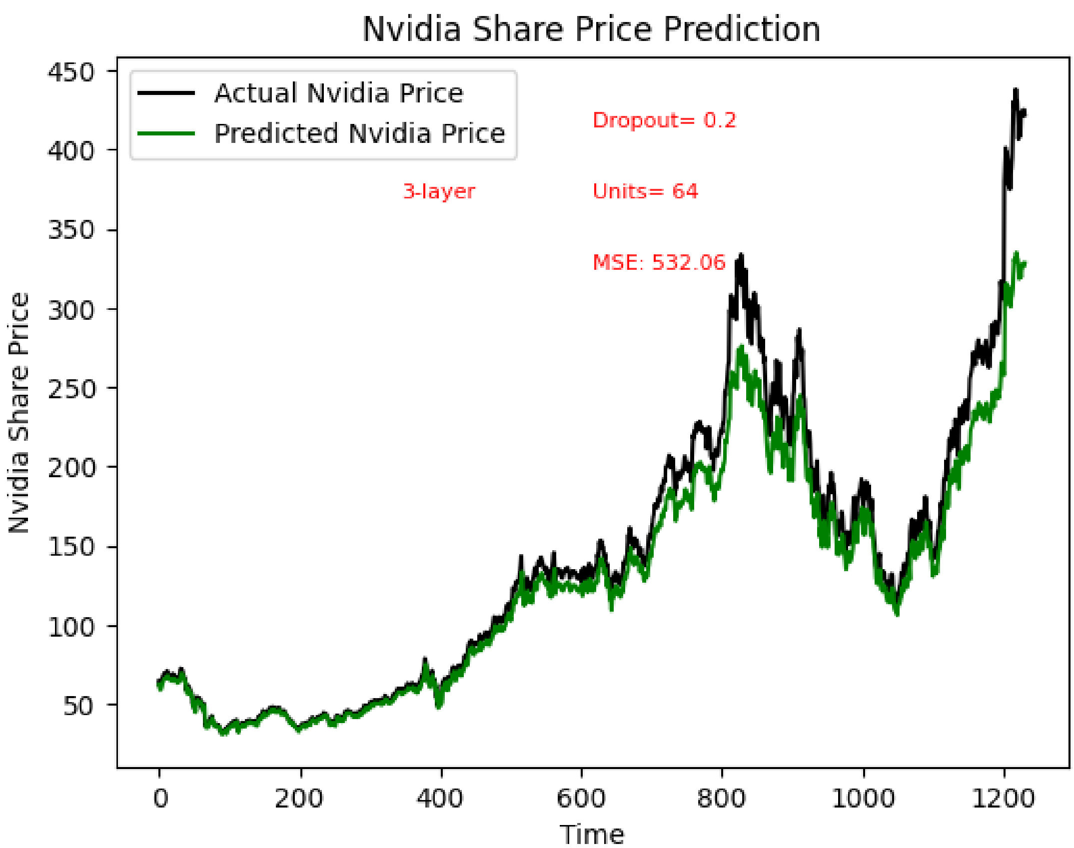
[17]:
```
import datetime as dt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import mean_squared_error


# Tarih aralıklarını belirtin
START_DATE = df['Date'].min()
END_DATE = df['Date'].max()
```

```python
# Eğitim ve test verilerini ayırmak için oranı belirtin
SPLIT_RATIO = 0.8

# Veriyi yükleyin (örnek bir CSV dosyası kullanılıyor)
df = pd.read_csv("NVIDIA (1999 -11.07.2023).csv", parse_dates=['Date'])

def load_data(start, end):
    dataframe = df.copy()
    dataframe = dataframe.loc[(dataframe['Date'] >= start) & (dataframe['Date']␣
↪<= end), :]
    dataframe = dataframe.rename(columns={'Closing_Price': 'Close'})
    return dataframe

data = load_data(start=START_DATE, end=END_DATE)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

# Geliştirilmiş LSTM modelini oluşturun
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=64, return_sequences=True, input_shape=(x_train.
↪shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=64, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=64))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    return model

# Eğitim ve test verilerini bölmek için indeksi hesaplayın
split_index = int(len(scaled_data) * SPLIT_RATIO)

# Eğitim verileri
x_train = scaled_data[:split_index - 1]
y_train = scaled_data[1:split_index]

# Test verileri
x_test = scaled_data[split_index - 1:-1]
y_test = scaled_data[split_index:]

# Modeli oluşturun ve derleyin
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
```

```
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()
mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 64", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.2", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')

plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 2ms/step - loss: 6.2045e-04
Epoch 2/25
154/154 [==============================] - 0s 2ms/step - loss: 2.0940e-05
Epoch 3/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6271e-05
Epoch 4/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5017e-05
Epoch 5/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5018e-05
Epoch 6/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4930e-05
Epoch 7/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5223e-05
Epoch 8/25
```

```
154/154 [==============================] - 0s 2ms/step - loss: 1.3817e-05
Epoch 9/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5914e-05
Epoch 10/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4813e-05
Epoch 11/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3481e-05
Epoch 12/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5236e-05
Epoch 13/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1706e-05
Epoch 14/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3535e-05
Epoch 15/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2448e-05
Epoch 16/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4932e-05
Epoch 17/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5601e-05
Epoch 18/25
154/154 [==============================] - 0s 2ms/step - loss: 1.5277e-05
Epoch 19/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4471e-05
Epoch 20/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4775e-05
Epoch 21/25
154/154 [==============================] - 0s 2ms/step - loss: 1.4298e-05
Epoch 22/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3467e-05
Epoch 23/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2703e-05
Epoch 24/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2135e-05
Epoch 25/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2797e-05
39/39 [==============================] - 1s 910us/step
```



[18]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 532.0589874131769
```

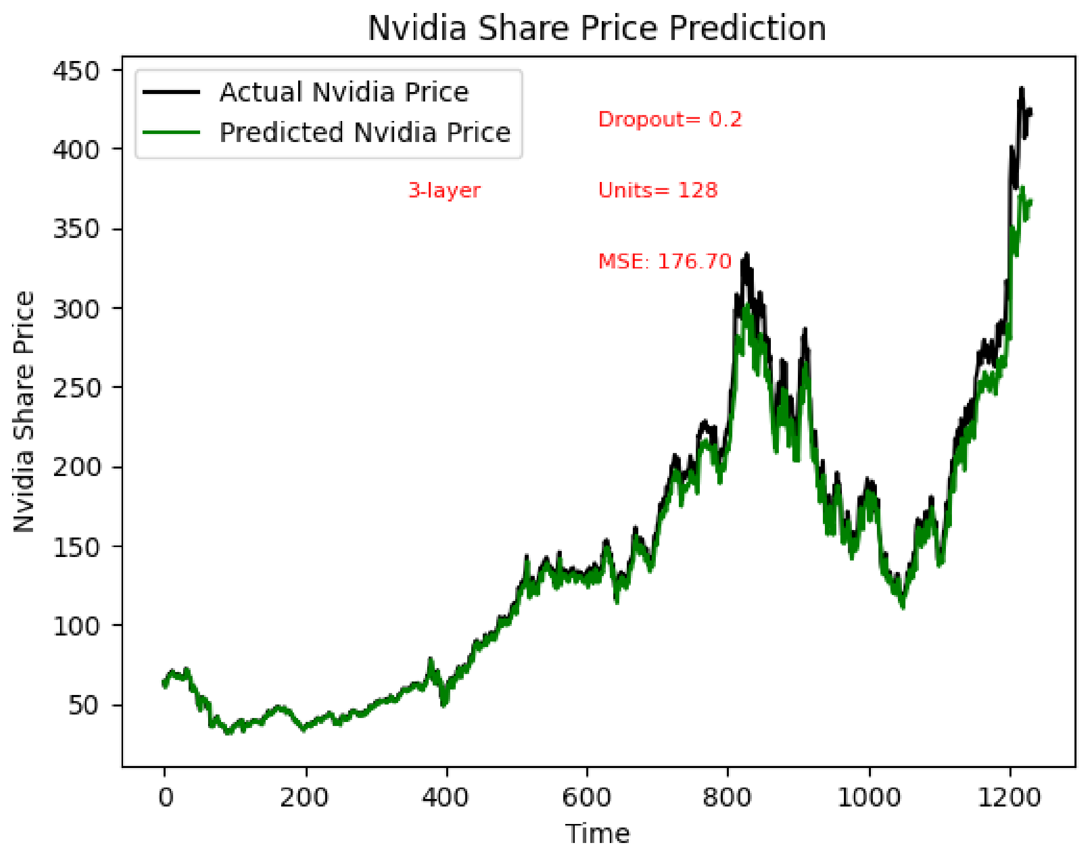
[19]:
```
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=128, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=128, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=128))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    return model

# Eğitim ve test verilerini bölmek için indeksi hesaplayın
split_index = int(len(scaled_data) * SPLIT_RATIO)
```

```python
# Eğitim verileri
x_train = scaled_data[:split_index - 1]
y_train = scaled_data[1:split_index]

# Test verileri
x_test = scaled_data[split_index - 1:-1]
y_test = scaled_data[split_index:]

# Modeli oluşturun ve derleyin
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()
mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 128", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.2", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 4ms/step - loss: 4.8822e-04
```

```
Epoch 2/25
154/154 [==============================] - 1s 4ms/step - loss: 1.1130e-05
Epoch 3/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0161e-05
Epoch 4/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0116e-05
Epoch 5/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0475e-05
Epoch 6/25
154/154 [==============================] - 1s 4ms/step - loss: 1.1421e-05
Epoch 7/25
154/154 [==============================] - 1s 4ms/step - loss: 9.9819e-06
Epoch 8/25
154/154 [==============================] - 1s 4ms/step - loss: 9.9882e-06
Epoch 9/25
154/154 [==============================] - 1s 4ms/step - loss: 1.1302e-05
Epoch 10/25
154/154 [==============================] - 1s 4ms/step - loss: 9.5709e-06
Epoch 11/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0328e-05
Epoch 12/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0287e-05
Epoch 13/25
154/154 [==============================] - 1s 4ms/step - loss: 9.4283e-06
Epoch 14/25
154/154 [==============================] - 1s 4ms/step - loss: 8.9177e-06
Epoch 15/25
154/154 [==============================] - 1s 4ms/step - loss: 9.2117e-06
Epoch 16/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0579e-05
Epoch 17/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0794e-05
Epoch 18/25
154/154 [==============================] - 1s 4ms/step - loss: 8.4702e-06
Epoch 19/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0374e-05
Epoch 20/25
154/154 [==============================] - 1s 4ms/step - loss: 8.8536e-06
Epoch 21/25
154/154 [==============================] - 1s 4ms/step - loss: 9.8921e-06
Epoch 22/25
154/154 [==============================] - 1s 4ms/step - loss: 8.9486e-06
Epoch 23/25
154/154 [==============================] - 1s 4ms/step - loss: 9.1725e-06
Epoch 24/25
154/154 [==============================] - 1s 4ms/step - loss: 1.0604e-05
Epoch 25/25
154/154 [==============================] - 1s 4ms/step - loss: 7.9411e-06
```

```
39/39 [==============================] - 1s 2ms/step
```



[20]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

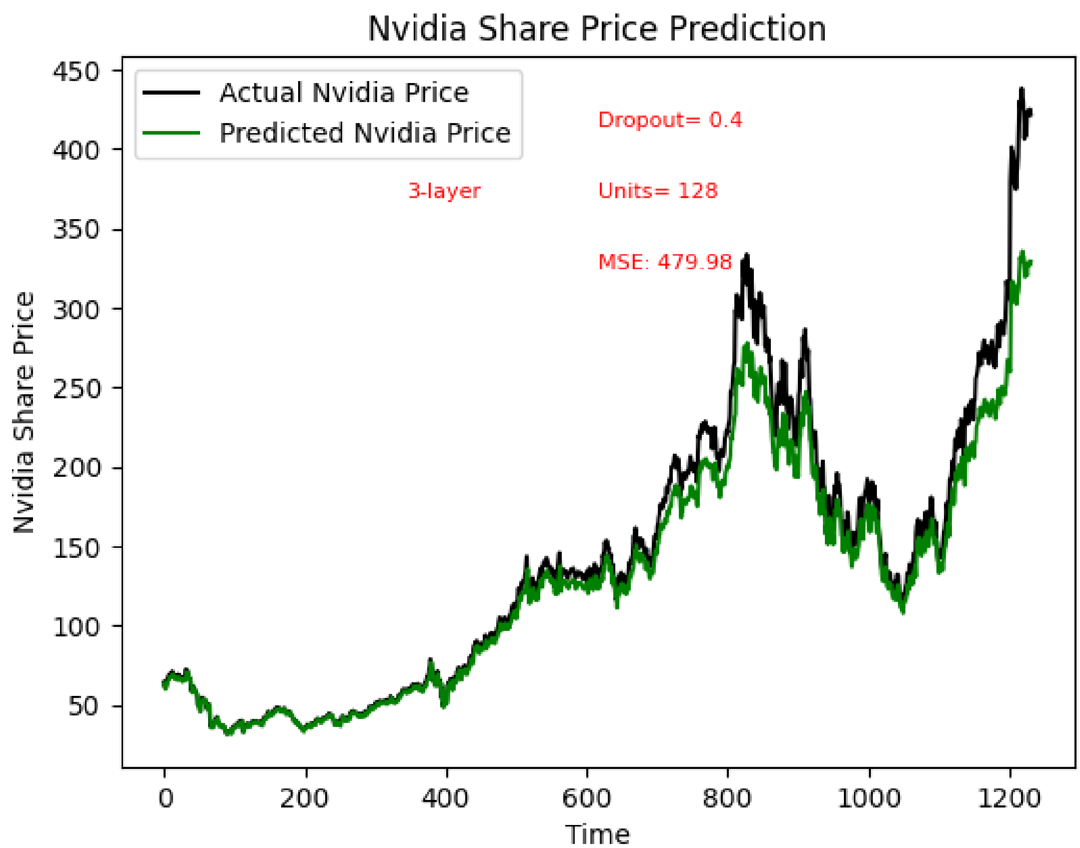
```
Mean Squared Error (MSE): 176.69940851857223
```

[21]:
```
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=128, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.4))
    model.add(LSTM(units=128, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(units=128))
    model.add(Dropout(0.4))
    model.add(Dense(units=1))
    return model
```

```python
# Eğitim ve test verilerini bölmek için indeksi hesaplayın
split_index = int(len(scaled_data) * SPLIT_RATIO)

# Eğitim verileri
x_train = scaled_data[:split_index - 1]
y_train = scaled_data[1:split_index]

# Test verileri
x_test = scaled_data[split_index - 1:-1]
y_test = scaled_data[split_index:]

# Modeli oluşturun ve derleyin
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()

mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 128", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.4", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 4ms/step - loss: 5.6626e-04
Epoch 2/25
154/154 [==============================] - 1s 4ms/step - loss: 2.6172e-05
Epoch 3/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3236e-05
Epoch 4/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2845e-05
Epoch 5/25
154/154 [==============================] - 1s 4ms/step - loss: 2.5389e-05
Epoch 6/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1740e-05
Epoch 7/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3985e-05
Epoch 8/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1636e-05
Epoch 9/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1466e-05
Epoch 10/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3620e-05
Epoch 11/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2782e-05
Epoch 12/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0646e-05
Epoch 13/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3445e-05
Epoch 14/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1061e-05
Epoch 15/25
154/154 [==============================] - 1s 4ms/step - loss: 1.7417e-05
Epoch 16/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3817e-05
Epoch 17/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0025e-05
Epoch 18/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0813e-05
Epoch 19/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9851e-05
Epoch 20/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0956e-05
Epoch 21/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9007e-05
Epoch 22/25
154/154 [==============================] - 1s 4ms/step - loss: 1.7750e-05
Epoch 23/25
154/154 [==============================] - 1s 4ms/step - loss: 1.7258e-05
Epoch 24/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2899e-05
```

```
Epoch 25/25
154/154 [==============================] - 1s 4ms/step - loss: 1.7682e-05
39/39 [==============================] - 1s 1ms/step
```



[22]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 479.9817491142463
```

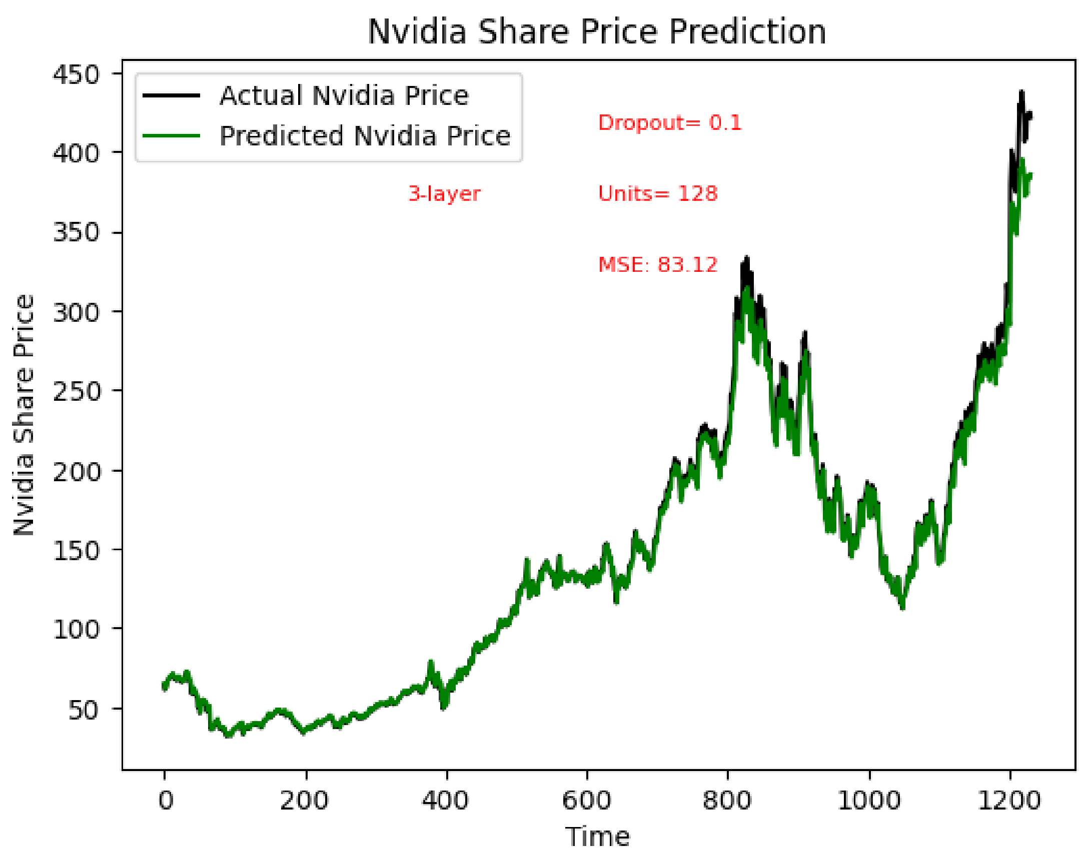
[23]:
```
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=128, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.1))
    model.add(LSTM(units=128, return_sequences=True))
    model.add(Dropout(0.1))
    model.add(LSTM(units=128))
    model.add(Dropout(0.1))
    model.add(Dense(units=1))
```

```python
    return model

# Eğitim ve test verilerini bölmek için indeksi hesaplayın
split_index = int(len(scaled_data) * SPLIT_RATIO)

# Eğitim verileri
x_train = scaled_data[:split_index - 1]
y_train = scaled_data[1:split_index]

# Test verileri
x_test = scaled_data[split_index - 1:-1]
y_test = scaled_data[split_index:]

# Modeli oluşturun ve derleyin
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()

mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 128", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.1", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
```

```
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 3ms/step - loss: 5.0598e-04
Epoch 2/25
154/154 [==============================] - 1s 3ms/step - loss: 6.3900e-06
Epoch 3/25
154/154 [==============================] - 1s 3ms/step - loss: 5.6522e-06
Epoch 4/25
154/154 [==============================] - 1s 3ms/step - loss: 5.5789e-06
Epoch 5/25
154/154 [==============================] - 1s 3ms/step - loss: 5.8530e-06
Epoch 6/25
154/154 [==============================] - 1s 3ms/step - loss: 5.4431e-06
Epoch 7/25
154/154 [==============================] - 1s 4ms/step - loss: 5.6239e-06
Epoch 8/25
154/154 [==============================] - 1s 3ms/step - loss: 4.7236e-06
Epoch 9/25
154/154 [==============================] - 1s 3ms/step - loss: 6.2903e-06
Epoch 10/25
154/154 [==============================] - 1s 3ms/step - loss: 5.0847e-06
Epoch 11/25
154/154 [==============================] - 1s 3ms/step - loss: 5.8901e-06
Epoch 12/25
154/154 [==============================] - 1s 3ms/step - loss: 4.8336e-06
Epoch 13/25
154/154 [==============================] - 1s 3ms/step - loss: 5.4850e-06
Epoch 14/25
154/154 [==============================] - 1s 4ms/step - loss: 5.9329e-06
Epoch 15/25
154/154 [==============================] - 1s 3ms/step - loss: 6.8447e-06
Epoch 16/25
154/154 [==============================] - 1s 3ms/step - loss: 5.5172e-06
Epoch 17/25
154/154 [==============================] - 1s 3ms/step - loss: 5.8198e-06
Epoch 18/25
154/154 [==============================] - 1s 3ms/step - loss: 5.5254e-06
Epoch 19/25
154/154 [==============================] - 1s 3ms/step - loss: 5.6909e-06
Epoch 20/25
154/154 [==============================] - 1s 3ms/step - loss: 4.6821e-06
Epoch 21/25
154/154 [==============================] - 1s 3ms/step - loss: 5.3744e-06
Epoch 22/25
```

```
154/154 [==============================] - 1s 3ms/step - loss: 5.3913e-06
Epoch 23/25
154/154 [==============================] - 1s 3ms/step - loss: 4.8251e-06
Epoch 24/25
154/154 [==============================] - 1s 4ms/step - loss: 4.8303e-06
Epoch 25/25
154/154 [==============================] - 1s 3ms/step - loss: 5.8441e-06
39/39 [==============================] - 1s 1ms/step
```



[24]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

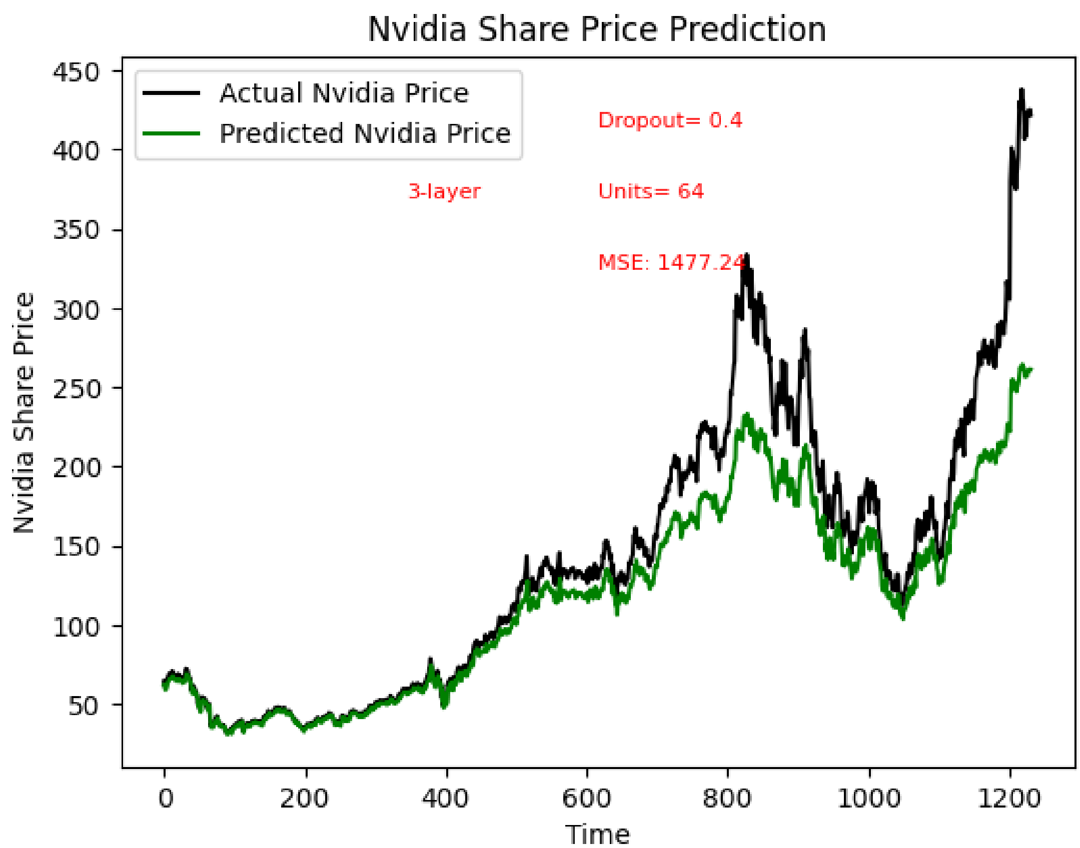
```
Mean Squared Error (MSE): 83.11820082386609
```

[25]:
```
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=64, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.4))
```

```
    model.add(LSTM(units=64, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(units=64))
    model.add(Dropout(0.4))
    model.add(Dense(units=1))
    return model
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()
mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 64", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.4", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 2ms/step - loss: 7.1881e-04
Epoch 2/25
154/154 [==============================] - 0s 2ms/step - loss: 5.0407e-05
Epoch 3/25
154/154 [==============================] - 0s 2ms/step - loss: 4.1591e-05
```

```
Epoch 4/25
154/154 [==============================] - 0s 2ms/step - loss: 4.0773e-05
Epoch 5/25
154/154 [==============================] - 0s 2ms/step - loss: 3.5253e-05
Epoch 6/25
154/154 [==============================] - 0s 2ms/step - loss: 3.6724e-05
Epoch 7/25
154/154 [==============================] - 0s 2ms/step - loss: 4.0297e-05
Epoch 8/25
154/154 [==============================] - 0s 2ms/step - loss: 3.4335e-05
Epoch 9/25
154/154 [==============================] - 0s 2ms/step - loss: 3.5918e-05
Epoch 10/25
154/154 [==============================] - 0s 2ms/step - loss: 3.7006e-05
Epoch 11/25
154/154 [==============================] - 0s 2ms/step - loss: 3.3759e-05
Epoch 12/25
154/154 [==============================] - 0s 2ms/step - loss: 3.4994e-05
Epoch 13/25
154/154 [==============================] - 0s 2ms/step - loss: 2.8393e-05
Epoch 14/25
154/154 [==============================] - 0s 2ms/step - loss: 3.3080e-05
Epoch 15/25
154/154 [==============================] - 0s 2ms/step - loss: 3.2964e-05
Epoch 16/25
154/154 [==============================] - 0s 2ms/step - loss: 3.0619e-05
Epoch 17/25
154/154 [==============================] - 0s 2ms/step - loss: 3.4378e-05
Epoch 18/25
154/154 [==============================] - 0s 2ms/step - loss: 3.5331e-05
Epoch 19/25
154/154 [==============================] - 0s 2ms/step - loss: 3.6655e-05
Epoch 20/25
154/154 [==============================] - 0s 2ms/step - loss: 3.5779e-05
Epoch 21/25
154/154 [==============================] - 0s 2ms/step - loss: 2.9153e-05
Epoch 22/25
154/154 [==============================] - 0s 2ms/step - loss: 3.1412e-05
Epoch 23/25
154/154 [==============================] - 0s 2ms/step - loss: 3.2810e-05
Epoch 24/25
154/154 [==============================] - 0s 2ms/step - loss: 3.0012e-05
Epoch 25/25
154/154 [==============================] - 0s 2ms/step - loss: 3.0443e-05
39/39 [==============================] - 1s 985us/step
```



[26]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

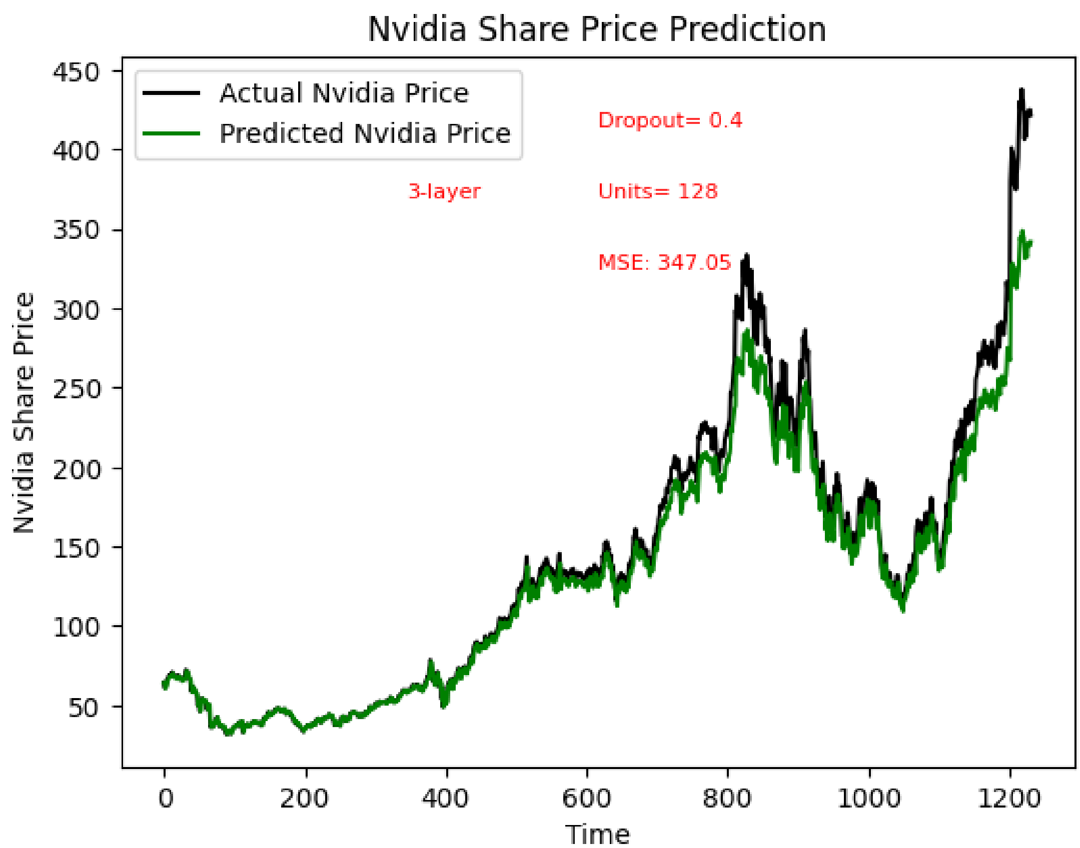
```
Mean Squared Error (MSE): 1477.2380223036457
```

[27]:
```
def LSTM_model():
    model = Sequential()
    model.add(LSTM(units=128, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.4))
    model.add(LSTM(units=128, return_sequences=True))
    model.add(Dropout(0.4))
    model.add(LSTM(units=128))
    model.add(Dropout(0.4))
    model.add(Dense(units=1))
    return model
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')
```

```python
# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)



# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()

mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 128", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.4", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"3-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 3s 4ms/step - loss: 5.3482e-04
Epoch 2/25
154/154 [==============================] - 1s 4ms/step - loss: 3.0666e-05
Epoch 3/25
154/154 [==============================] - 1s 4ms/step - loss: 2.4716e-05
Epoch 4/25
154/154 [==============================] - 1s 4ms/step - loss: 2.4856e-05
Epoch 5/25
154/154 [==============================] - 1s 4ms/step - loss: 2.4092e-05
Epoch 6/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2066e-05
Epoch 7/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3344e-05
```

```
Epoch 8/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2385e-05
Epoch 9/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2033e-05
Epoch 10/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3090e-05
Epoch 11/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3538e-05
Epoch 12/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9242e-05
Epoch 13/25
154/154 [==============================] - 1s 4ms/step - loss: 2.2482e-05
Epoch 14/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0747e-05
Epoch 15/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1739e-05
Epoch 16/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9726e-05
Epoch 17/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0678e-05
Epoch 18/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0735e-05
Epoch 19/25
154/154 [==============================] - 1s 4ms/step - loss: 2.0926e-05
Epoch 20/25
154/154 [==============================] - 1s 4ms/step - loss: 2.3508e-05
Epoch 21/25
154/154 [==============================] - 1s 4ms/step - loss: 1.7626e-05
Epoch 22/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9418e-05
Epoch 23/25
154/154 [==============================] - 1s 4ms/step - loss: 1.9435e-05
Epoch 24/25
154/154 [==============================] - 1s 4ms/step - loss: 2.1153e-05
Epoch 25/25
154/154 [==============================] - 1s 4ms/step - loss: 1.5693e-05
39/39 [==============================] - 1s 1ms/step
```



[28]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

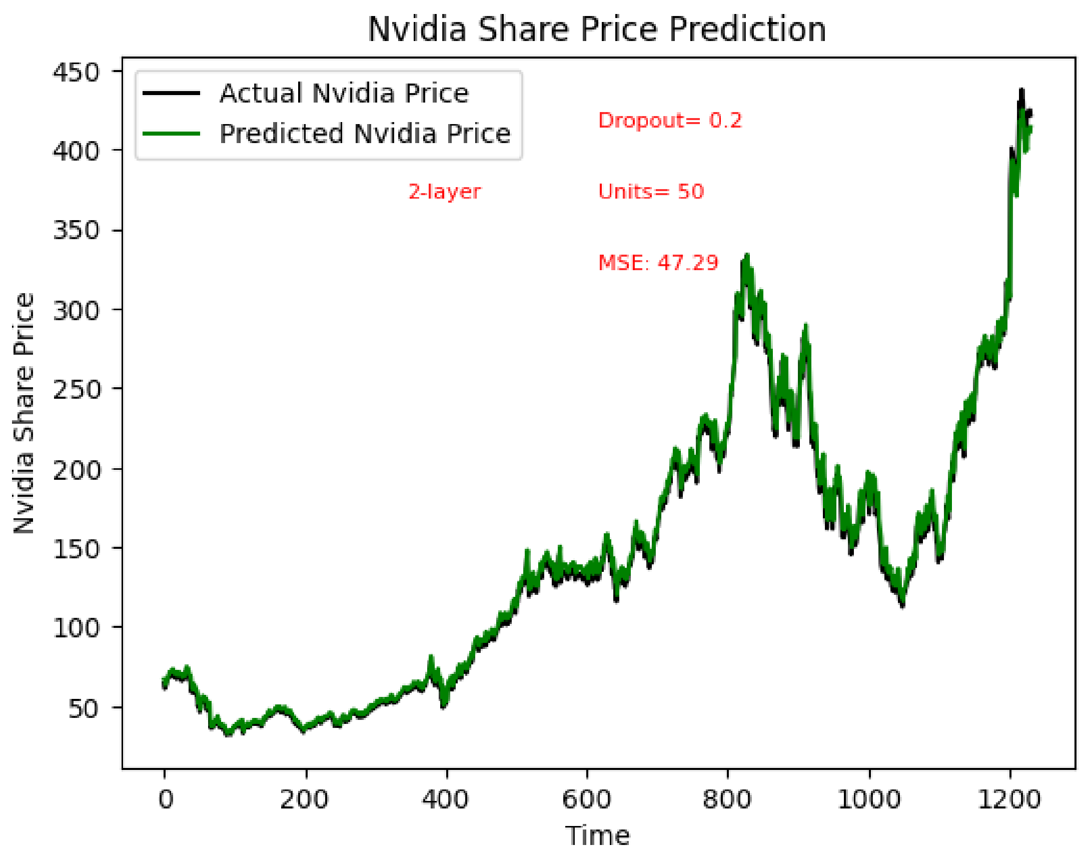
```
Mean Squared Error (MSE): 347.05141535963264
```

[29]:
```
def LSTM_model():
    model = Sequential()

    # İlk LSTM katmanı
    model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.2))

    # İkinci LSTM katmanı
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))

    # Son birim (Dense) katman
    model.add(Dense(units=1))
```

```python
    return model

model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')

# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Modeli kullanarak tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri yeniden şekillendirin
predicted_prices = predicted_prices.reshape(-1, 1)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)


# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()
mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 50", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.2", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"2-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')

plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 2s 2ms/step - loss: 5.3315e-04
Epoch 2/25
154/154 [==============================] - 0s 2ms/step - loss: 1.6828e-05
Epoch 3/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2549e-05
```

```
Epoch 4/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2219e-05
Epoch 5/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1600e-05
Epoch 6/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3710e-05
Epoch 7/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0456e-05
Epoch 8/25
154/154 [==============================] - 0s 2ms/step - loss: 1.3178e-05
Epoch 9/25
154/154 [==============================] - 0s 1ms/step - loss: 1.2175e-05
Epoch 10/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0726e-05
Epoch 11/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2960e-05
Epoch 12/25
154/154 [==============================] - 0s 1ms/step - loss: 1.1836e-05
Epoch 13/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1669e-05
Epoch 14/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0391e-05
Epoch 15/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1976e-05
Epoch 16/25
154/154 [==============================] - 0s 1ms/step - loss: 1.0208e-05
Epoch 17/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0861e-05
Epoch 18/25
154/154 [==============================] - 0s 1ms/step - loss: 1.1147e-05
Epoch 19/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1545e-05
Epoch 20/25
154/154 [==============================] - 0s 2ms/step - loss: 9.8217e-06
Epoch 21/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0524e-05
Epoch 22/25
154/154 [==============================] - 0s 1ms/step - loss: 1.0880e-05
Epoch 23/25
154/154 [==============================] - 0s 2ms/step - loss: 1.2156e-05
Epoch 24/25
154/154 [==============================] - 0s 2ms/step - loss: 1.1233e-05
Epoch 25/25
154/154 [==============================] - 0s 2ms/step - loss: 1.0920e-05
39/39 [==============================] - 0s 786us/step
```



[30]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

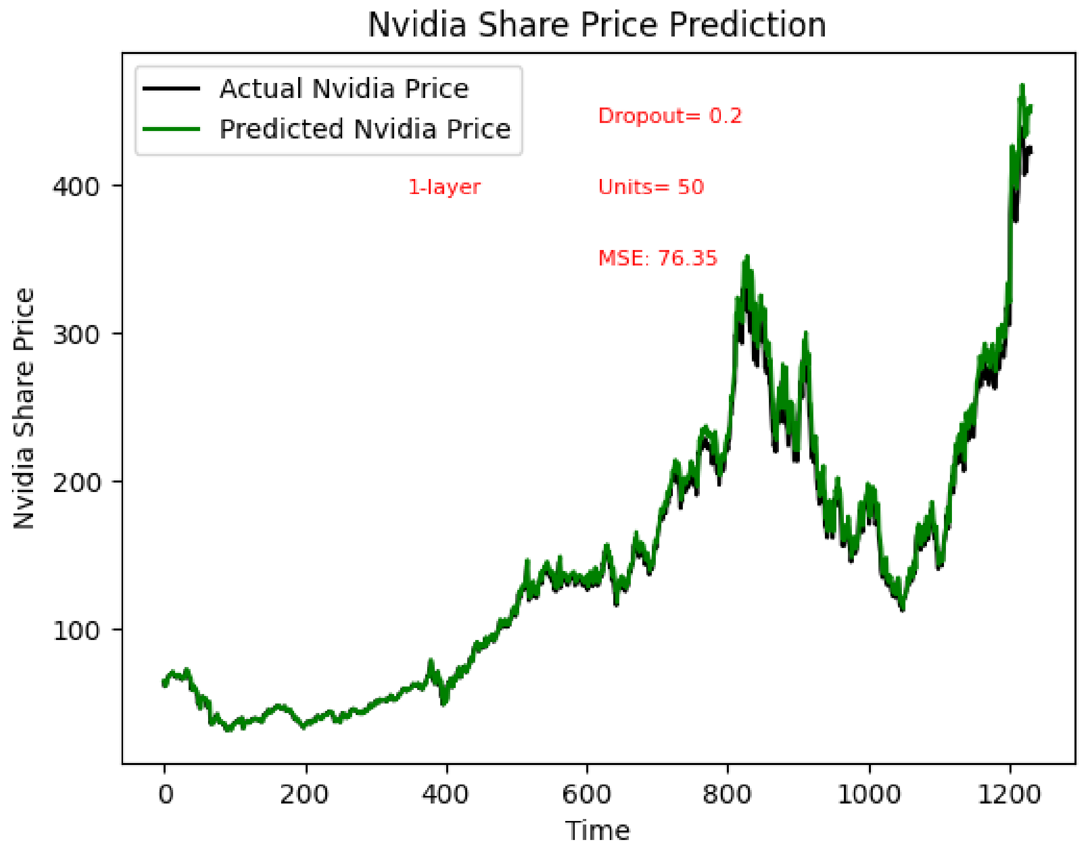
```
Mean Squared Error (MSE): 47.288123063864084
```

[31]:
```
def LSTM_model():
    model = Sequential()

    # Tek bir LSTM katmanı
    model.add(LSTM(units=50, return_sequences=False, input_shape=(x_train.
 ↪shape[1], 1)))
    model.add(Dropout(0.2))

    # Çıkış katmanı
    model.add(Dense(units=1))

    return model
model = LSTM_model()
model.compile(optimizer='adam', loss='mean_squared_error')
```

```python
# Modeli eğitin
model.fit(x_train, y_train, epochs=25, batch_size=32)

# Modeli kullanarak tahminler yapın
predicted_prices = model.predict(x_test)

# Tahminleri yeniden şekillendirin
predicted_prices = predicted_prices.reshape(-1, 1)

# Tahminleri ters ölçekleme ve gerçek değerlerle karşılaştırma
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y_test)


# Gerçek ve tahmin edilen fiyatları çizdirin
plt.plot(actual_prices, color='black', label="Actual Nvidia Price")
plt.plot(predicted_prices, color='green', label="Predicted Nvidia Price")
plt.title("Nvidia Share Price Prediction")
plt.xlabel("Time")
plt.ylabel("Nvidia Share Price")
plt.legend()

mse = mean_squared_error(actual_prices, predicted_prices)

# MSE değerini grafiğe ekleyin
plt.text(0.5, 0.7, f"MSE: {mse:.2f}", transform=plt.gca().transAxes,␣
 ↪fontsize=8, color='red')
plt.text(0.5, 0.8, f"Units= 50", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.5, 0.9, f"Dropout= 0.2", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.text(0.3, 0.8, f"1-layer", transform=plt.gca().transAxes, fontsize=8,␣
 ↪color='red')
plt.show()
```

```
Epoch 1/25
154/154 [==============================] - 1s 1ms/step - loss: 6.5612e-04
Epoch 2/25
154/154 [==============================] - 0s 1ms/step - loss: 1.2760e-04
Epoch 3/25
154/154 [==============================] - 0s 1ms/step - loss: 9.6171e-06
Epoch 4/25
154/154 [==============================] - 0s 1ms/step - loss: 8.5890e-06
Epoch 5/25
154/154 [==============================] - 0s 1ms/step - loss: 7.9295e-06
Epoch 6/25
```

```
154/154 [==============================] - 0s 1ms/step - loss: 9.4993e-06
Epoch 7/25
154/154 [==============================] - 0s 1ms/step - loss: 8.6128e-06
Epoch 8/25
154/154 [==============================] - 0s 1ms/step - loss: 7.3936e-06
Epoch 9/25
154/154 [==============================] - 0s 1ms/step - loss: 8.6098e-06
Epoch 10/25
154/154 [==============================] - 0s 1ms/step - loss: 8.1299e-06
Epoch 11/25
154/154 [==============================] - 0s 1ms/step - loss: 8.9122e-06
Epoch 12/25
154/154 [==============================] - 0s 1ms/step - loss: 8.1850e-06
Epoch 13/25
154/154 [==============================] - 0s 1ms/step - loss: 7.7460e-06
Epoch 14/25
154/154 [==============================] - 0s 1ms/step - loss: 8.0564e-06
Epoch 15/25
154/154 [==============================] - 0s 1ms/step - loss: 7.8877e-06
Epoch 16/25
154/154 [==============================] - 0s 1ms/step - loss: 7.2792e-06
Epoch 17/25
154/154 [==============================] - 0s 1ms/step - loss: 7.4622e-06
Epoch 18/25
154/154 [==============================] - 0s 1ms/step - loss: 7.5061e-06
Epoch 19/25
154/154 [==============================] - 0s 1ms/step - loss: 7.6588e-06
Epoch 20/25
154/154 [==============================] - 0s 1ms/step - loss: 7.2399e-06
Epoch 21/25
154/154 [==============================] - 0s 1ms/step - loss: 7.1464e-06
Epoch 22/25
154/154 [==============================] - 0s 1ms/step - loss: 7.5747e-06
Epoch 23/25
154/154 [==============================] - 0s 1ms/step - loss: 7.1520e-06
Epoch 24/25
154/154 [==============================] - 0s 1ms/step - loss: 7.2345e-06
Epoch 25/25
154/154 [==============================] - 0s 1ms/step - loss: 6.5758e-06
39/39 [==============================] - 0s 654us/step
```



[32]:
```
# Hata hesaplama
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 76.35201162790251
```