Autoarima

RD

2023-10-01

Stock Price Forecasting using Autoregressive Integrated Moving Average (ARIMA) in R Language: A Case Study on NVIDIA Corporation (NVDA)

ABOUT THE STOCK

The focus of this study is to forecast stock prices using the Autoregressive Integrated Moving Average (ARIMA) model. The stock in question is NVIDIA Corporation, commonly known as NVDA, a leading technology company known for its Graphics Processing Units (GPUs) for gaming and professional markets.

Data Source

The data used in this study is sourced from historical NVDA stock prices, which can be easily imported from Yahoo Finance using the Quantmod package in R. The dataset comprises various metrics such as Open, High, Low, and Close prices (OHLC). However, for the sake of simplicity, this study will focus solely on the closing prices, resulting in a univariate time series model.

Time Frame

The dataset spans from January 22, 1999, to the most recent available date, encompassing a broad range of trading days, which allows for a comprehensive time series analysis.

DATA PREPROCESSING

```
# Load necessary libraries
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.2     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflic
ts to become errors
```

```
library(quantmod)
```

```
## Loading required package: xts
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
##
## ######################### Warning from 'xts' package ##########################
## #                                                                             #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                             #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                             #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## ###############################################################################
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
##
## Loading required package: TTR
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(tseries)

# Avoid conflict between dplyr and other packages
conflictRules('dplyr', exclude = 'lag')

# Get stock data for NVDA
getSymbols("NVDA", from = "1999-01-22")
```

```
## [1] "NVDA"
```

```
# Create and arrange df
df <- data.frame(Date=index(NVDA), coredata(NVDA))
df$Date <- as.Date(df$Date, format = "%Y-%m-%d")  # As.Date from base package
df <- df %>% arrange(Date)
```

STATIONARITY TEST

```
# Initial ADF test
adf_test <- adf.test(df$NVDA.Close, alternative = "stationary")
```

```
## Warning in adf.test(df$NVDA.Close, alternative = "stationary"): p-value greater
## than printed p-value
```

```
adf_test
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$NVDA.Close
## Dickey-Fuller = 2.3077, Lag order = 18, p-value = 0.99
## alternative hypothesis: stationary
```

```
# Applying differencing
diff_series <- diff(df$NVDA.Close)
diff_series <- na.omit(diff_series)  # Corrected object name here

# Differencing ADF test
adf_test2 <- adf.test(diff_series, alternative = "stationary")
```

```
## Warning in adf.test(diff_series, alternative = "stationary"): p-value smaller
## than printed p-value
```

```
adf_test2
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff_series
## Dickey-Fuller = -18.386, Lag order = 18, p-value = 0.01
## alternative hypothesis: stationary
```

Given by the inital series we were not able to pass the ADF test. However, after a 1st differencing we were able to achieve stationary.

AUTO ARIMA with d =1

From the previous, we were able to achieve that d = 1 for the following (p,d,q) fom used in our auto arima.

```
library(forecast)
#auto arima
auto_arima_model <- auto.arima(df$NVDA.Close, d=1)
auto_arima_model
```

```
## Series: df$NVDA.Close
## ARIMA(0,1,0) with drift
##
## Coefficients:
##        drift
##       0.0720
## s.e.  0.0356
##
## sigma^2 = 7.877:  log likelihood = -15227
## AIC=30458   AICc=30458   BIC=30471.47
```

```
# seasonality - auto arima
auto_arima_model_season <- auto.arima(df$NVDA.Close, d=1, seasonal=TRUE)
auto_arima_model_season
```

```
## Series: df$NVDA.Close
## ARIMA(0,1,0) with drift
##
## Coefficients:
##        drift
##       0.0720
## s.e.  0.0356
##
## sigma^2 = 7.877:  log likelihood = -15227
## AIC=30458   AICc=30458   BIC=30471.47
```

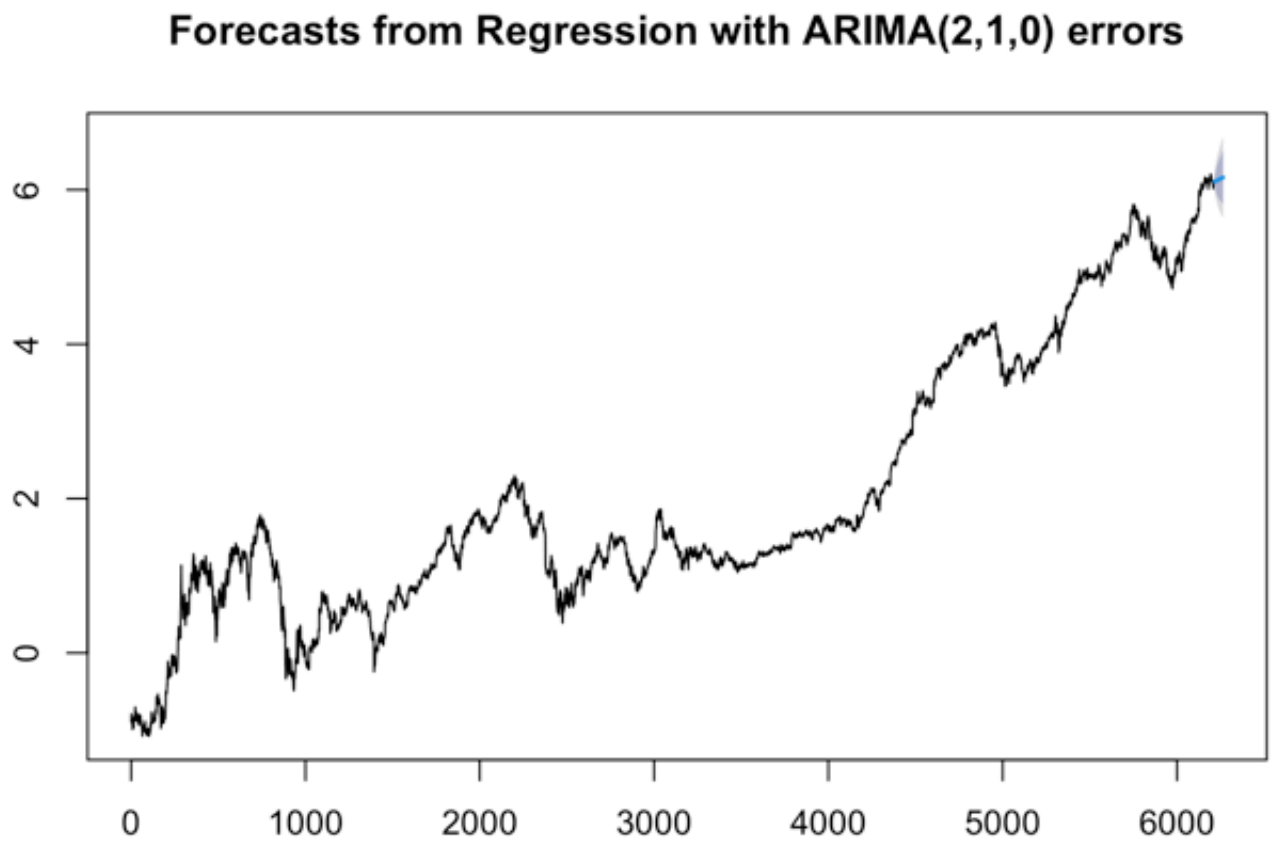
```
#stepwise auto arima
auto_arima_model2 <- auto.arima(df$NVDA.Close, d=1, stepwise=FALSE, approximation=FAL
SE)
auto_arima_model2
```

```
## Series: df$NVDA.Close
## ARIMA(0,1,5) with drift
##
## Coefficients:
##           ma1      ma2      ma3      ma4     ma5   drift
##       -0.0172  -0.0290  -0.0138  -0.0208  0.0856  0.0720
## s.e.   0.0127   0.0128   0.0128   0.0133  0.0128  0.0356
##
## sigma^2 = 7.816:  log likelihood = -15200.53
## AIC=30415.06   AICc=30415.08   BIC=30462.2
```

```
# Number of trading days in your NVDA dataset. Replace nrow(df) with the actual numbe
r of rows in your NVDA data.
n <- nrow(df)

# Initialize the seed for reproducibility
set.seed(123)

# Generate initial random interest rates within bounds
simulated_interest_rate <- runif(n, min = 0.25, max = 3.5)

# Introduce some autocorrelation to make the series more realistic
# Assume a lag-1 autocorrelation of 0.9
for(i in 2:n) {
  simulated_interest_rate[i] <- 0.9 * simulated_interest_rate[i - 1] + (1 - 0.9) * ru
nif(1, min = 0.25, max = 3.5)
}

# Clip values to stay within bounds
simulated_interest_rate <- pmin(pmax(simulated_interest_rate, 0.25), 3.5)

# Add it to your existing NVDA data frame
df$simulated_interest_rate <- simulated_interest_rate

# Now you can use this simulated data in your auto.arima model
library(forecast)
auto_arima_model_exo <- auto.arima(log(df$NVDA.Close), d=1, D=1, xreg=df$simulated_in
terest_rate)

auto_arima_model_exo
```

```
## Series: log(df$NVDA.Close)
## Regression with ARIMA(2,1,0) errors
##
## Coefficients:
##          ar1      ar2   drift    xreg
##       0.0006  -0.0067  0.0011  -3e-04
## s.e.  0.0127   0.0127  0.0005   5e-03
##
## sigma^2 = 0.001439:  log likelihood = 11514.87
## AIC=-23019.74   AICc=-23019.73   BIC=-22986.07
```

```
# forecasting and plotting
# Generate future values for the simulated_interest_rate
set.seed(456)  # For reproducibility

# Initialize the future values with the last value from the previous simulation
future_simulated_interest_rate <- numeric(50)
future_simulated_interest_rate[1] <- tail(simulated_interest_rate, n=1)

# Generate the rest
for(i in 2:50) {
  future_simulated_interest_rate[i] <- 0.9 * future_simulated_interest_rate[i - 1] +
(1 - 0.9) * runif(1, min = 0.25, max = 3.5)
}

# Clip values to stay within bounds
future_simulated_interest_rate <- pmin(pmax(future_simulated_interest_rate, 0.25), 3.
5)

# Forecasting
forecasts <- forecast(auto_arima_model_exo, h=50, xreg=future_simulated_interest_rat
e)

# Diagnostic plots
plot(forecasts)
```



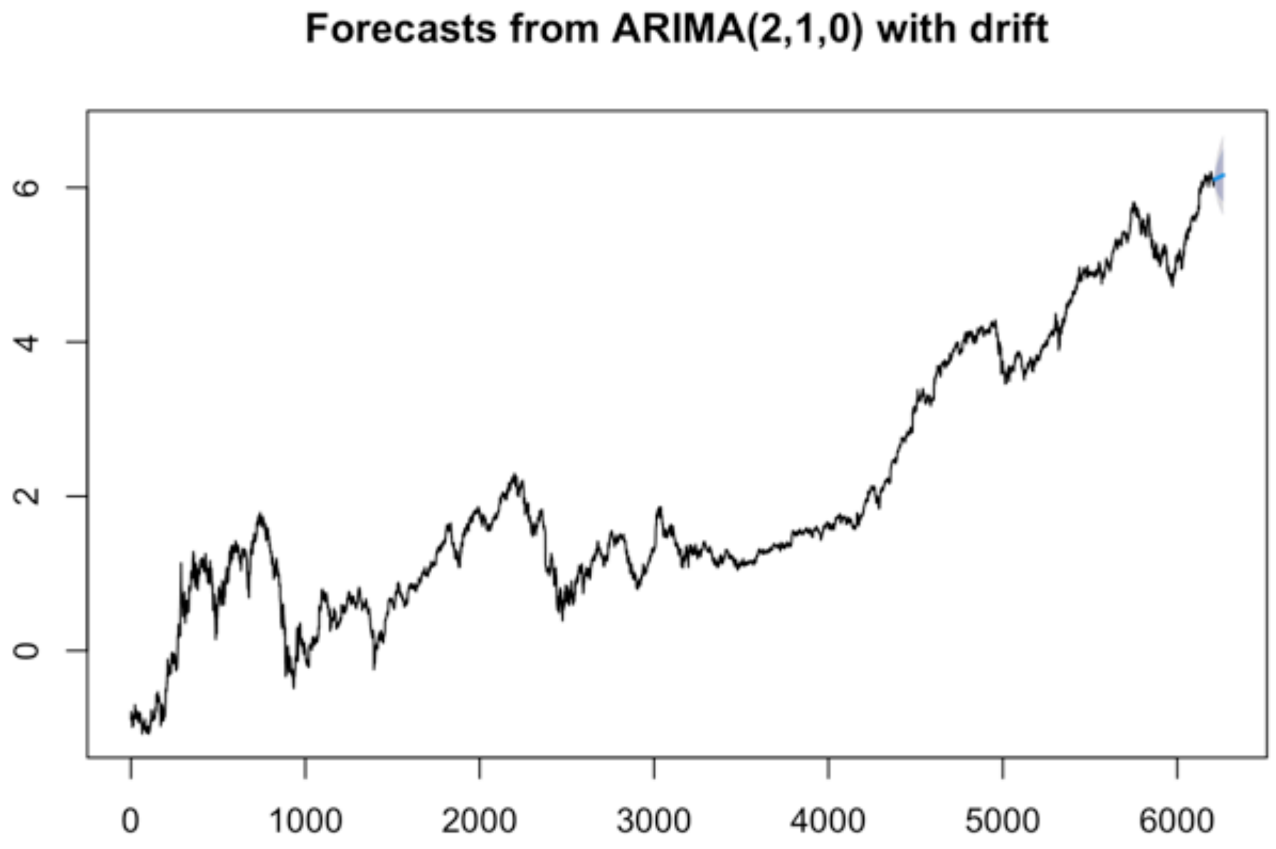
```
# Ljung-Box test on residuals
Box.test(auto_arima_model_exo$residuals, lag=log(length(auto_arima_model_exo$residual
s)))
```

```
##
##  Box-Pierce test
##
## data:  auto_arima_model_exo$residuals
## X-squared = 19.094, df = 8.7346, p-value = 0.02133
```

```
# differencing and transfomration
auto_arima_model3 <- auto.arima(log(df$NVDA.Close), d=1, D=1)
auto_arima_model3
```

```
## Series: log(df$NVDA.Close)
## ARIMA(2,1,0) with drift
##
## Coefficients:
##          ar1      ar2   drift
##       0.0006  -0.0067  0.0011
## s.e.  0.0127   0.0127  0.0005
##
## sigma^2 = 0.001439:  log likelihood = 11514.87
## AIC=-23021.74   AICc=-23021.73   BIC=-22994.8
```

```
# Diagnostic plots
plot(forecast(auto_arima_model3, h=50))
```



```
# Ljung-Box test
Box.test(auto_arima_model3$residuals, lag=log(length(auto_arima_model3$residuals)))
```

```
##
##  Box-Pierce test
##
## data:  auto_arima_model3$residuals
## X-squared = 19.093, df = 8.7346, p-value = 0.02134
```

Monte Carlo Methods

The mathematical foundation of Monte Carlo methods lies in probability theory and statistics. Specifically, they are often used to approximate the expectation $E[f(X)]$ of some function $f(X)$ with respect to the probability distribution $P(X)$:

$$E[f(X)] = \int f(x)P(x)dx$$

where the Monte Carlo simulation is the approximated integral by taking the average of $f(x)$ over N samples for $\{x_1, x_2, \ldots, x_N\}$ drawn from the fact that $P(x)$:

$$E[f(X)] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

as N approaches infinity, the approximation converges to the actual expectation by the Law of Large Numbers.

NVDA stock price

```
# Get stock data for NVDA
getSymbols("NVDA", from = "1999-01-22", to = "2023-10-03", auto.assign = TRUE, warnin
gs= FALSE)
```

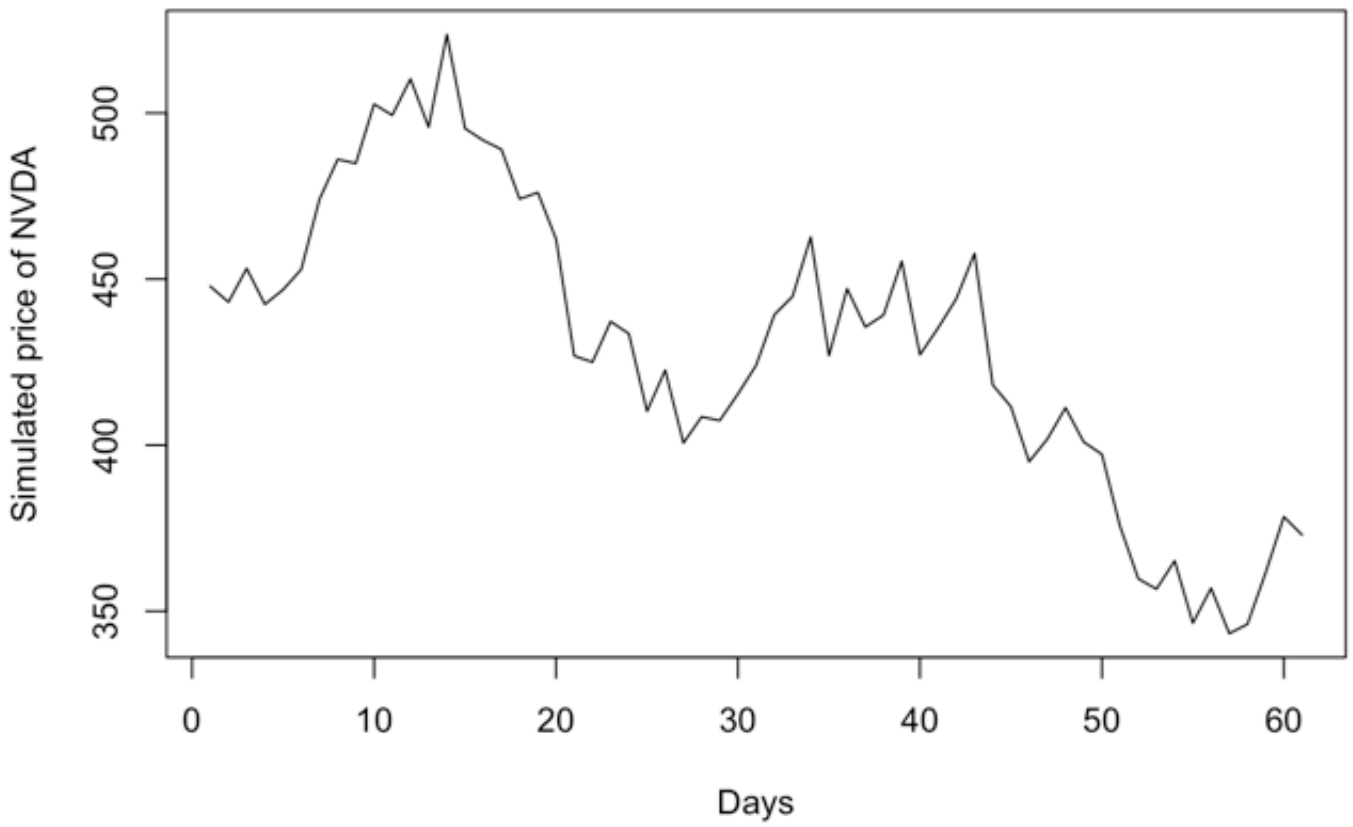
```
## [1] "NVDA"
```

```
#getSymbols("SPY", from = '2011-09-08', to = "2021-09-08", auto.assign = TRUE, warnin
gs = FALSE)
```

```
daily_mean <- mean(dailyReturn(NVDA)) #mean
daily_std_dev <- sd(dailyReturn(NVDA)) #standard deviation
```

```
no_of_days <- 60 # Set variable to 60 days
starting_price <- last(NVDA$NVDA.Close)[[1]] #Closing price from 434.95 - last item i
n our frame
```

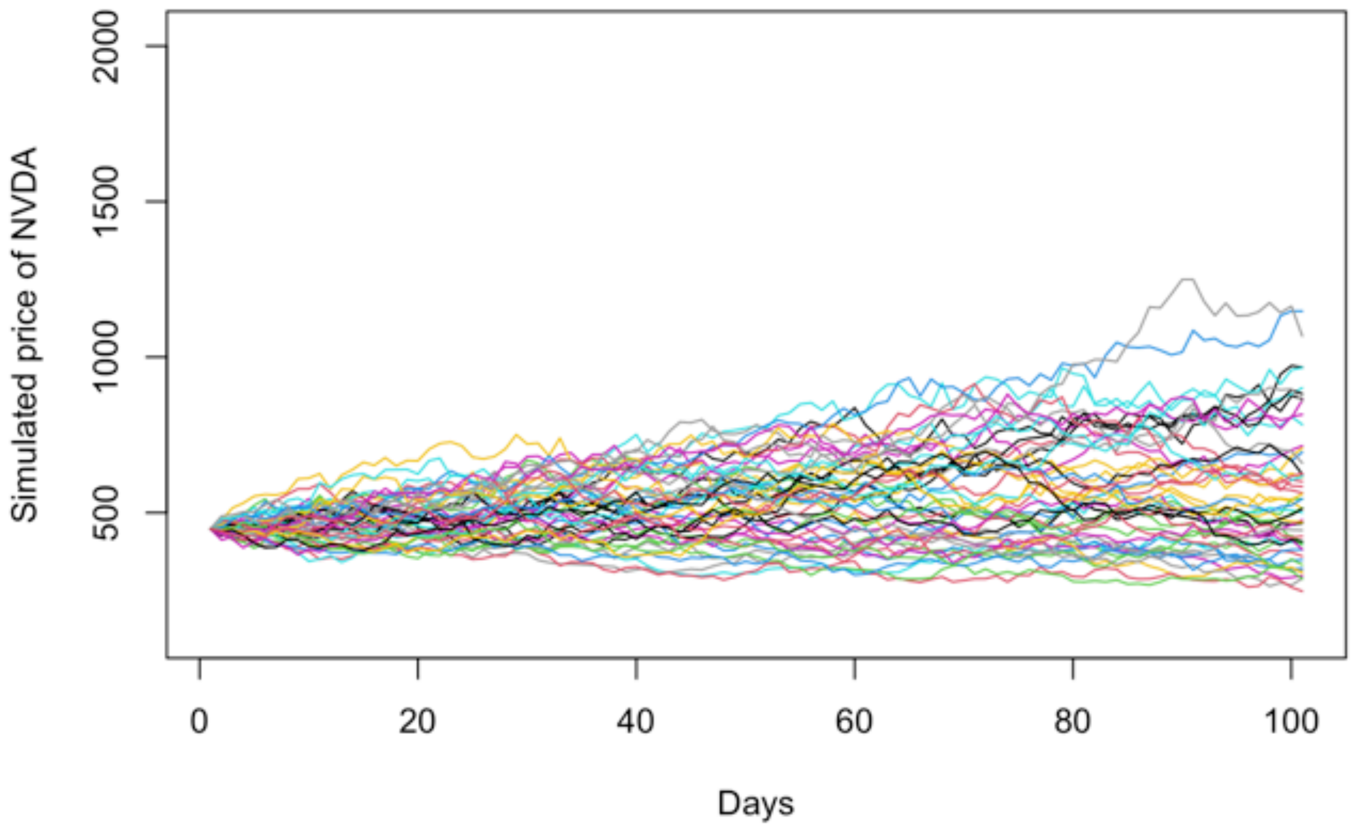
```
set.seed(101) #Set seed for reproducibility of the random numbers
returns <- 1+rnorm(no_of_days, mean=daily_mean, sd=daily_std_dev) #Generate random va
riables
prices <- cumprod(c(starting_price, returns)) #Calculate cumulative product

plot(prices, type='l', ylab="Simulated price of NVDA", xlab="Days")
```



60 day simulated price chart for NVDA.

Monte Carlo 100 Simulations - NVDA

```
# Existing setup code
no_of_sims <- 4000 # times to run the loop used in the for loop
no_of_days <- 100  # assuming you have this variable somewhere
starting_price <- last(NVDA$NVDA.Close)[[1]] #Closing price from 434.95 - last item i
n our frame
daily_mean <- mean(dailyReturn(NVDA)) #mean
daily_std_dev <- sd(dailyReturn(NVDA)) #standard deviation

returns_list <- matrix(0, nrow = no_of_sims, ncol = no_of_days) #define matrices
prices_list <- matrix(0, nrow = no_of_sims, ncol = no_of_days+1)

for(i in 1:no_of_sims) {
  returns_list[i,] <- rnorm(no_of_days, mean=daily_mean, sd=daily_std_dev)
  prices_list[i,] <- cumprod(c(starting_price, 1+returns_list[i,]))
}

# Calculate dynamic y-limits
y_min <- min(prices_list)
y_max <- max(prices_list)

# Plotting
plot(prices_list[1,], type='l', ylab="Simulated price of NVDA", xlab="Days", ylim=c(y
_min, y_max))
for(i in 1:50) {
  lines(prices_list[i, ], type = 'l', col=i)
}
```



As we can see, the results of different simulations vary greatly. Some end up below the starting point and some increase very rapidly to over $800. We can also notice, just by looking at the graphs, that most simulations tend to end up with a positive return. As this is only 50 of 4000 simulations we can only assume that's the case for all simulations.

ASML stock price

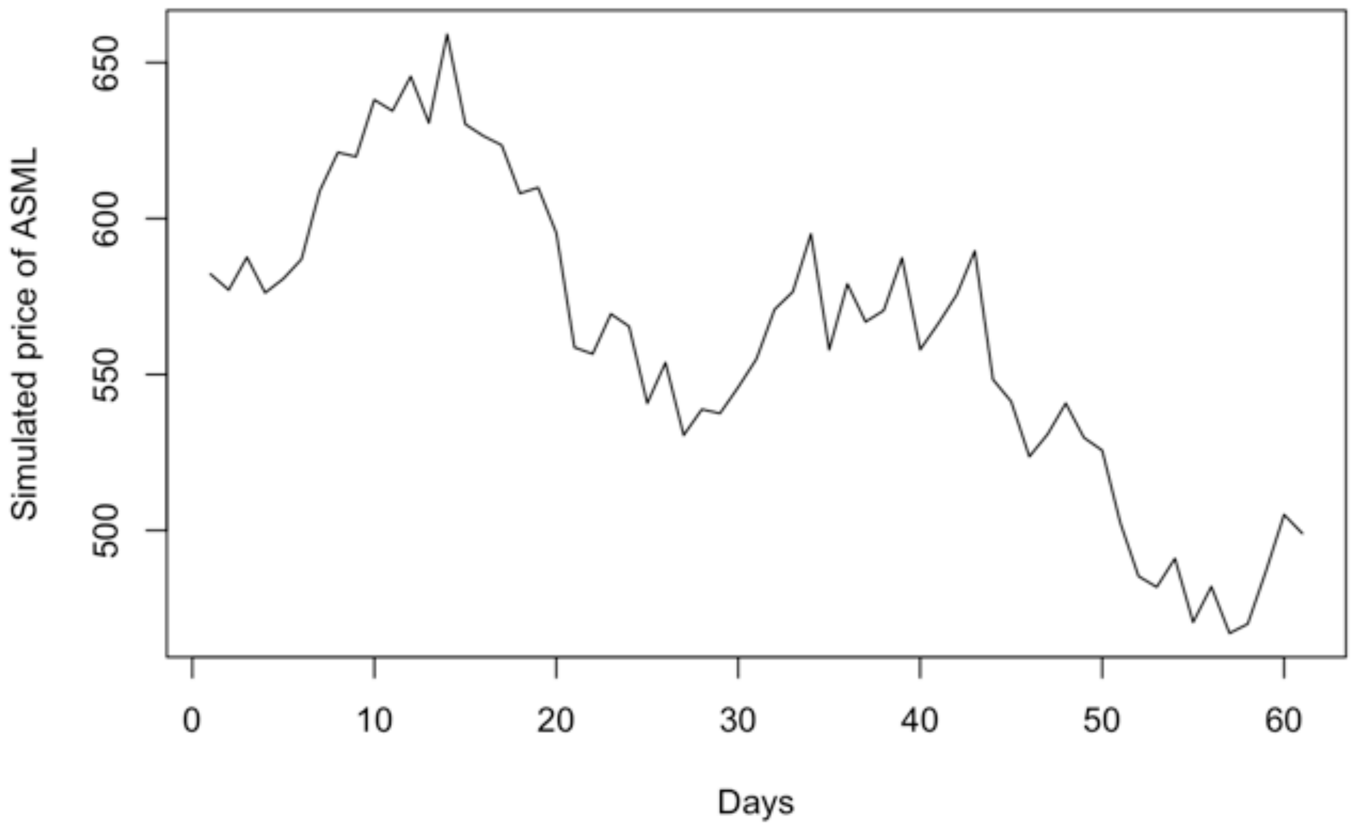
```
# Get stock data for NVDA
getSymbols("ASML", from = "1995-03-14", to = "2023-10-03", auto.assign = TRUE, warnin
gs= FALSE)
```

```
## [1] "ASML"
```

```
daily_mean1 <- mean(dailyReturn(ASML)) #mean
daily_std_dev1 <- sd(dailyReturn(ASML)) #standard deviation
```

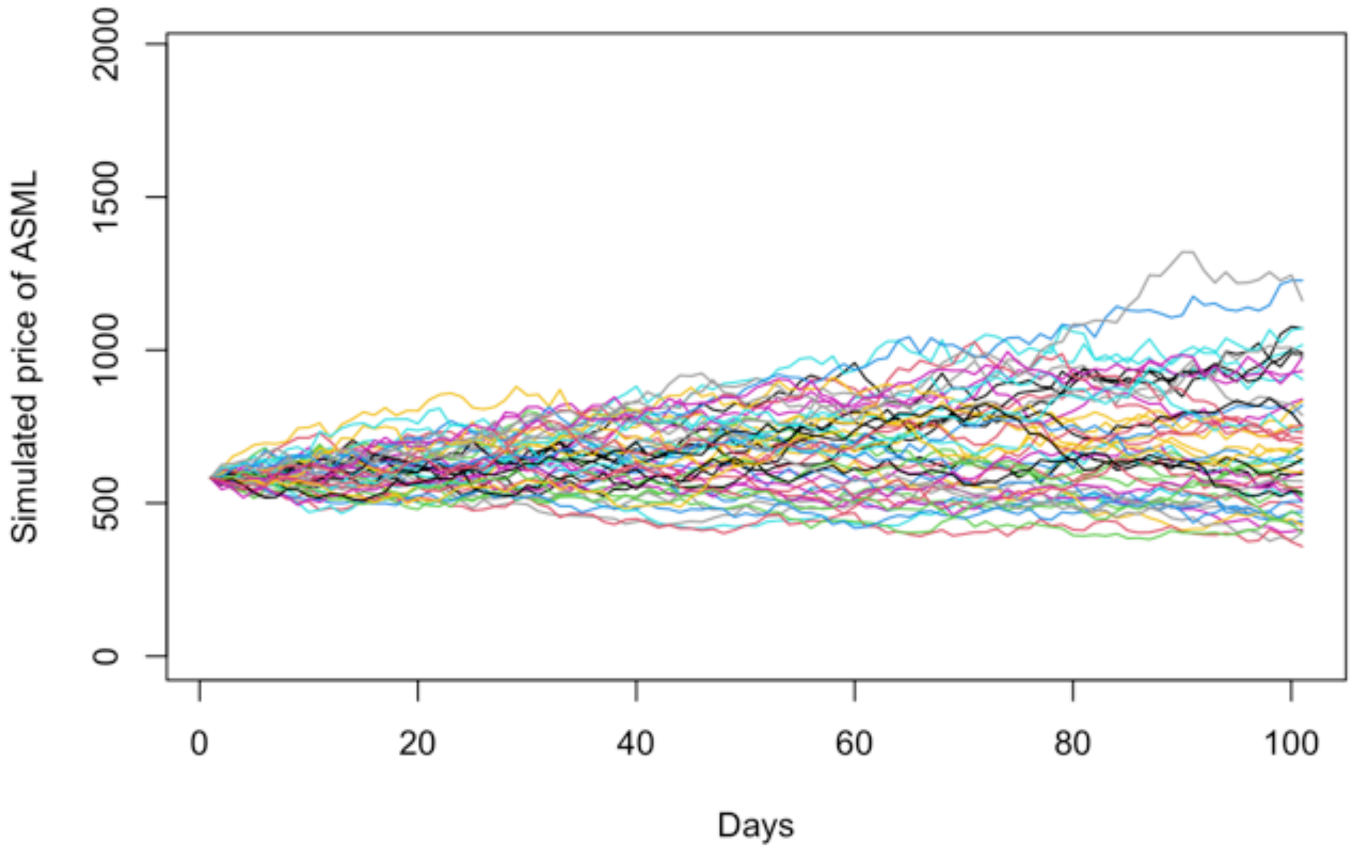
```
no_of_days1 <- 60 # Set variable to 60 days
starting_price1 <- last(ASML$ASML.Close)[[1]] #Closing price from 434.95 - last item
in our frame
```

```
set.seed(101) #Set seed for reproducibility of the random numbers
returns <- 1+rnorm(no_of_days1, mean=daily_mean1, sd=daily_std_dev1) #Generate random
variables
prices <- cumprod(c(starting_price1, returns)) #Calculate cumulative product

plot(prices, type='l', ylab="Simulated price of ASML", xlab="Days")
```



Monte Carlo 100 Simulations - ASML

```r
# Existing setup code
no_of_sims1 <- 9000 # times to run the loop used in the for loop
no_of_days1 <- 100  # assuming you have this variable somewhere
starting_price1 <- last(ASML$ASML.Close)[[1]] #Closing price
daily_mean1 <- mean(dailyReturn(ASML)) #mean
daily_std_dev1 <- sd(dailyReturn(ASML)) #standard deviation

returns_list <- matrix(0, nrow = no_of_sims1, ncol = no_of_days1) #define matrices
prices_list <- matrix(0, nrow = no_of_sims1, ncol = no_of_days1+1)

for(i in 1:no_of_sims) {
  returns_list[i,] <- rnorm(no_of_days1, mean=daily_mean1, sd=daily_std_dev1)
  prices_list[i,] <- cumprod(c(starting_price1, 1+returns_list[i,]))
}

# Calculate dynamic y-limits
y_min <- min(prices_list)
y_max <- max(prices_list)

# Plotting
plot(prices_list[1,], type='l', ylab="Simulated price of ASML", xlab="Days", ylim=c(y
_min, y_max))
for(i in 1:50) {
  lines(prices_list[i, ], type = 'l', col=i)
}
```



Simulated price of ASML
0
500
1000
1500
2000
0
20
40
60
80
100
Days