In [1]:

```python
#Import libraries
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import pandas as pd
import numpy as np
import os
import datetime as dt
%matplotlib inline
```

In [3]:

```python
# Load the data.
df = pd.read_table('KO_LEE_READY.txt', sep = " ")
df = df.reset_index()
df = df[['index', 'SYMBOL', 'BID', 'BIDSIZ', 'OFR', 'OFRSIZ', 'PRICE', 'SIZE', 'EX', 'DT']]
names = ['DATE_TIME',  'SYMBOL', 'BID', 'BIDSIZ', 'OFR', 'OFRSIZ', 'PRICE', 'SIZE', 'EX', 'DT']
df.columns = names

df['DATE_TIME'] = df['DATE_TIME'].str.replace('X', '')  # remove the X
df['DATE_TIME'] = df['DATE_TIME'].str[:-3]              # remove nan-seconds to be consistent will all rows.   :(
df['DATE_TIME'] = pd.to_datetime(df['DATE_TIME'], format = '%Y.%m.%d.%H.%M.%S.%f') # save it as a date time variable
df.head()
```

Out[3]:

	DATE_TIME	SYMBOL	BID	BIDSIZ	OFR	OFRSIZ	PRICE	SIZE	EX	DT
0	2022-03-01 00:00:00.020	KO	62.25	1	62.30	1	62.24	0	NYS	-1
1	2022-03-01 14:30:01.244	KO	62.07	2	62.27	2	62.14	277740	NYS	-1
2	2022-03-01 14:30:02.000	KO	62.10	3	62.13	1	62.10	566	NYS	-1
3	2022-03-01 14:30:02.032	KO	62.08	1	62.12	1	62.09	200	NYS	-1
4	2022-03-01 14:30:03.256	KO	62.05	1	62.09	4	62.06	100	NYS	-1

In [4]:

```python
EST = []
n_row, n_col = df.shape

cont = 5
#final_time = given_time - pd.DateOffset(hours=n)

for i in range(0, n_row):
    EST.append(df['DATE_TIME'].iloc[i]  - pd.DateOffset(hours=cont))

df.insert(1, 'EST', EST)
df.head()
```

Out[4]:

	DATE_TIME	EST	SYMBOL	BID	BIDSIZ	OFR	OFRSIZ	PRICE	SIZE	EX	DT
0	2022-03-01 00:00:00.020	2022-02-28 19:00:00.020	KO	62.25	1	62.30	1	62.24	0	NYS	-1
1	2022-03-01 14:30:01.244	2022-03-01 09:30:01.244	KO	62.07	2	62.27	2	62.14	277740	NYS	-1
2	2022-03-01 14:30:02.000	2022-03-01 09:30:02.000	KO	62.10	3	62.13	1	62.10	566	NYS	-1
3	2022-03-01 14:30:02.032	2022-03-01 09:30:02.032	KO	62.08	1	62.12	1	62.09	200	NYS	-1
4	2022-03-01 14:30:03.256	2022-03-01 09:30:03.256	KO	62.05	1	62.09	4	62.06	100	NYS	-1

In [5]:

```python
df['BidAskSpread'] = df['BID']- df['OFR']
df.shape
```

Out[5]: (63390, 12)

In [6]:

```python
df['Return'] = df['PRICE'].pct_change()
df.shape
```

Out[6]: (63390, 13)

In [7]:

```python
EX_NAME = 'ADF'  # please subset the data to only one exchange!!
df1 = df[df['EX'] == EX_NAME].copy()

start_date = pd.to_datetime('2022-03-01 09:30')
end_date = pd.to_datetime('2022-03-01 15:20')
df1.loc[(df1['EST'] > start_date) & (df1['EST'] < end_date)]
```

Out[7]:

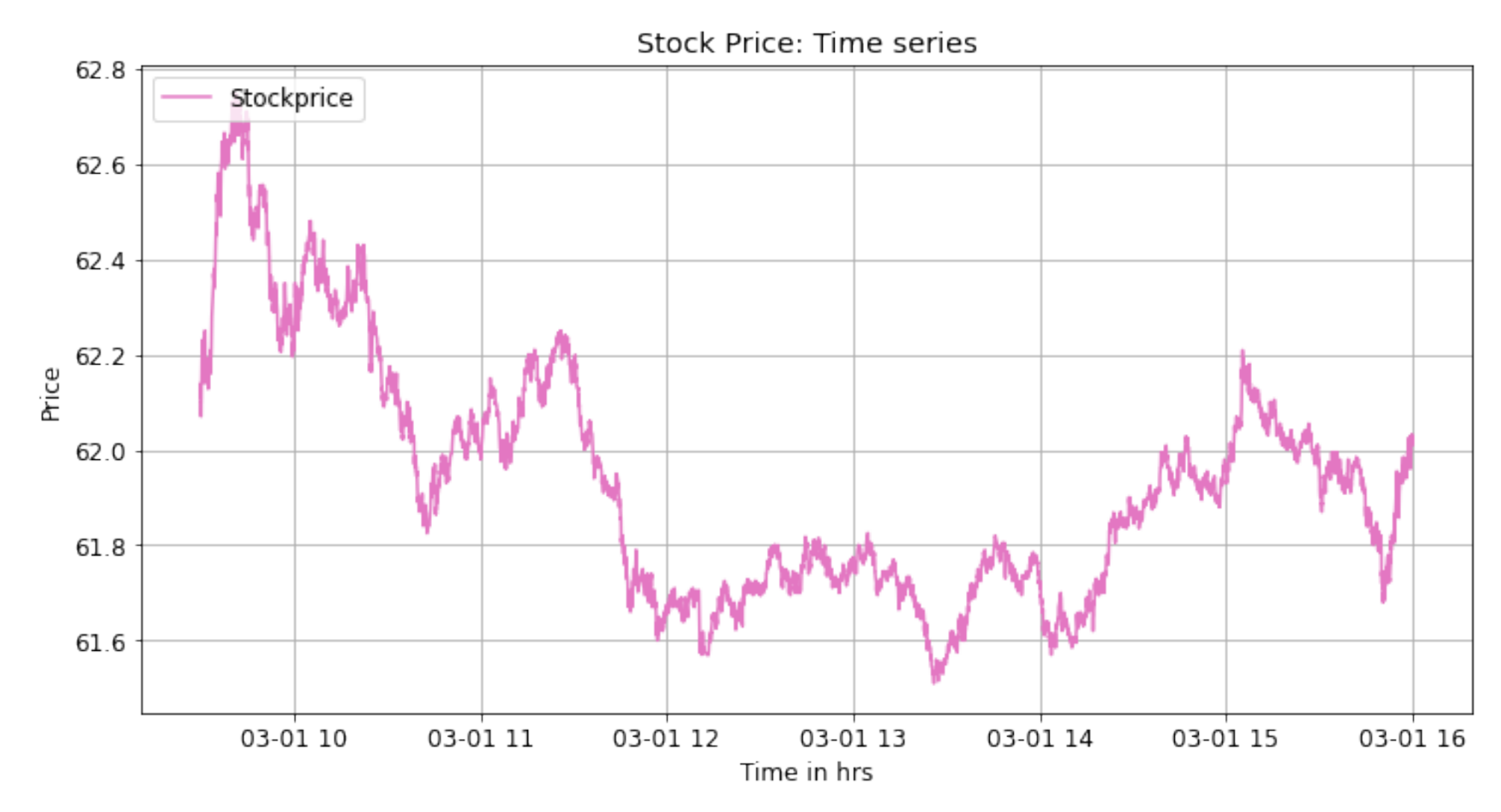
	DATE_TIME	EST	SYMBOL	BID	BIDSIZ	OFR	OFRSIZ	PRICE	SIZE	EX	DT	BidAskSpread	Return
10383	2022-03-01 14:30:01.264	2022-03-01 09:30:01.264	KO	62.11	3	62.15	11	62.1100	100	ADF	-1	-0.04	0.002259
10384	2022-03-01 14:30:01.408	2022-03-01 09:30:01.408	KO	62.10	4	62.14	1	62.1400	100	ADF	1	-0.04	0.000483
10385	2022-03-01 14:30:02.268	2022-03-01 09:30:02.268	KO	62.07	1	62.10	1	62.0700	100	ADF	-1	-0.03	-0.001126
10386	2022-03-01 14:30:04.404	2022-03-01 09:30:04.404	KO	62.10	1	62.11	1	62.0900	100	ADF	-1	-0.01	0.000322
10387	2022-03-01 14:30:04.756	2022-03-01 09:30:04.756	KO	62.09	1	62.11	1	62.0900	490	ADF	-1	-0.02	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19803	2022-03-01 20:19:42.512	2022-03-01 15:19:42.512	KO	62.01	51	62.02	3	62.0150	100	ADF	-1	-0.01	-0.000161
19804	2022-03-01 20:19:48.596	2022-03-01 15:19:48.596	KO	62.00	18	62.01	15	62.0078	1000	ADF	1	-0.01	-0.000116
19805	2022-03-01 20:19:49.852	2022-03-01 15:19:49.852	KO	62.00	18	62.01	18	62.0000	168	ADF	-1	-0.01	-0.000126
19806	2022-03-01 20:19:52.440	2022-03-01 15:19:52.440	KO	62.01	11	62.02	27	62.0100	100	ADF	-1	-0.01	0.000161
19807	2022-03-01 20:19:52.596	2022-03-01 15:19:52.596	KO	62.01	9	62.02	28	62.0150	100	ADF	1	-0.01	0.000081

9425 rows × 13 columns

In [8]:

```python
# Specify area
plt.rc('font', size=12)
fig, ax = plt.subplots(figsize=(12, 6))

# Specify lines
ax.plot(df1.EST, df1.PRICE, color='tab:pink', label='Stockprice')

# Same as above
ax.set_xlabel('Time in hrs')
ax.set_ylabel('Price')
ax.set_title('Stock Price: Time series')
ax.grid(True)
ax.legend(loc='upper left')
```

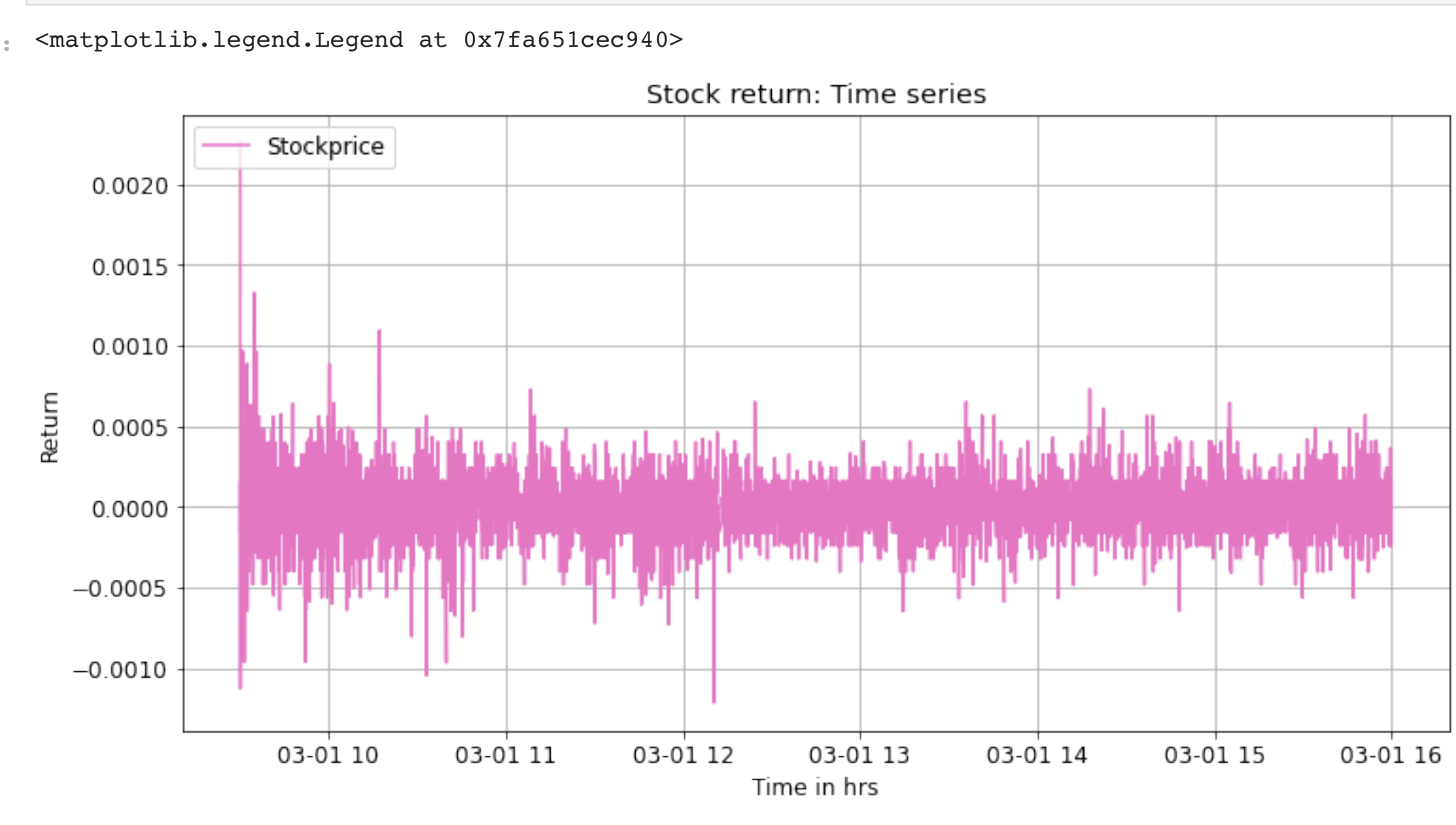
Out[8]: <matplotlib.legend.Legend at 0x7fa671e668e0>



In [9]:

```python
# Specify area
plt.rc('font', size=12)
fig, ax = plt.subplots(figsize=(12, 6))

# Specify lines
ax.plot(df1.EST, df1.Return, color='tab:pink', label='Stockprice')

# Same as above
ax.set_xlabel('Time in hrs')
ax.set_ylabel('Return')
ax.set_title('Stock return: Time series')
ax.grid(True)
ax.legend(loc='upper left')
```

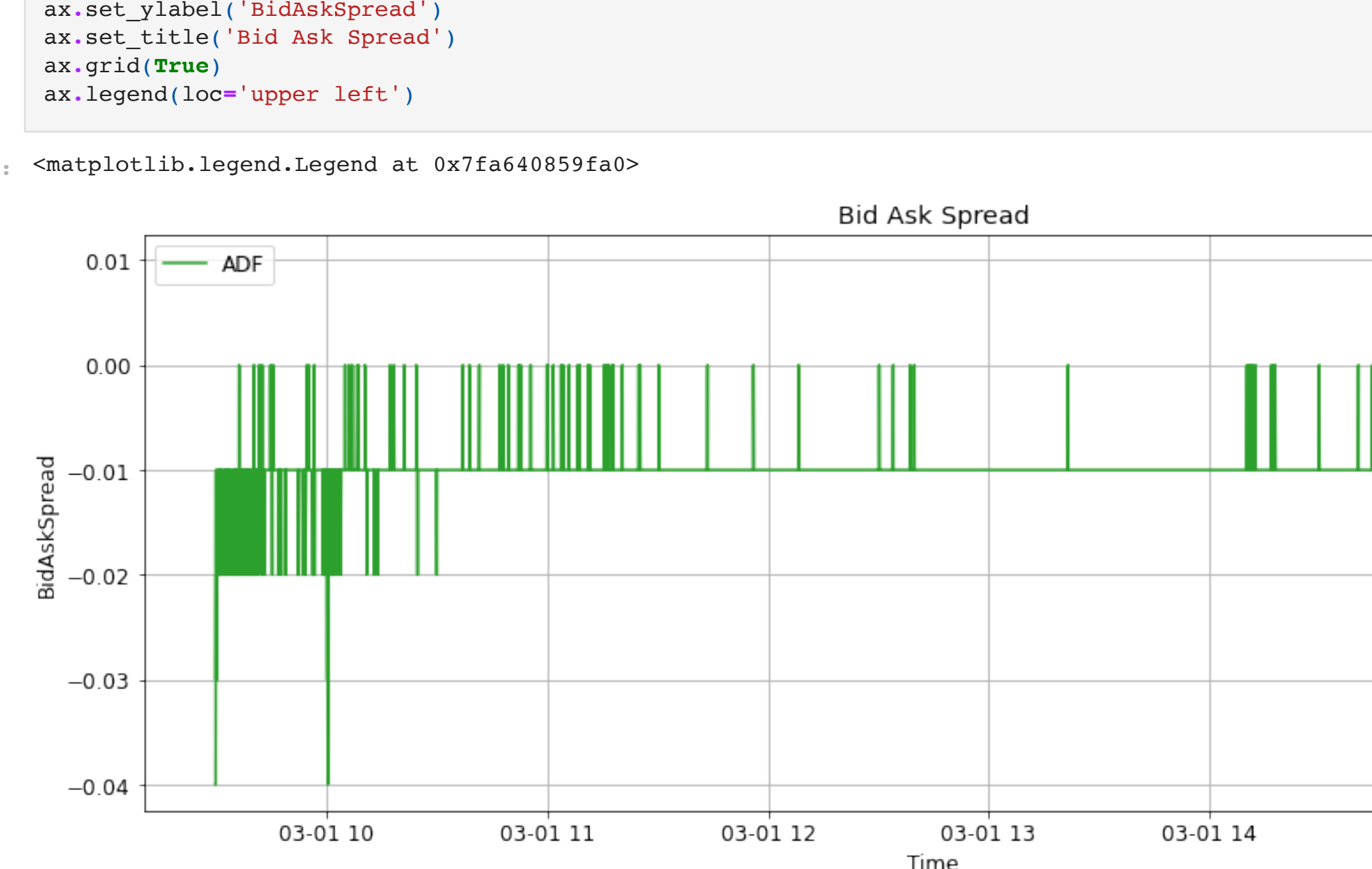
Out[9]: <matplotlib.legend.Legend at 0x7fa651cec940>



In [10]:

```python
# Specify area
plt.rc('font', size=12)
fig, ax = plt.subplots(figsize=(16, 6))

# Specify lines
ax.plot(df1.EST, df1.BidAskSpread, color='tab:green', label='ADF')

# Same as above
ax.set_xlabel('Time')
ax.set_ylabel('BidAskSpread')
ax.set_title('Bid Ask Spread')
ax.grid(True)
ax.legend(loc='upper left')
```

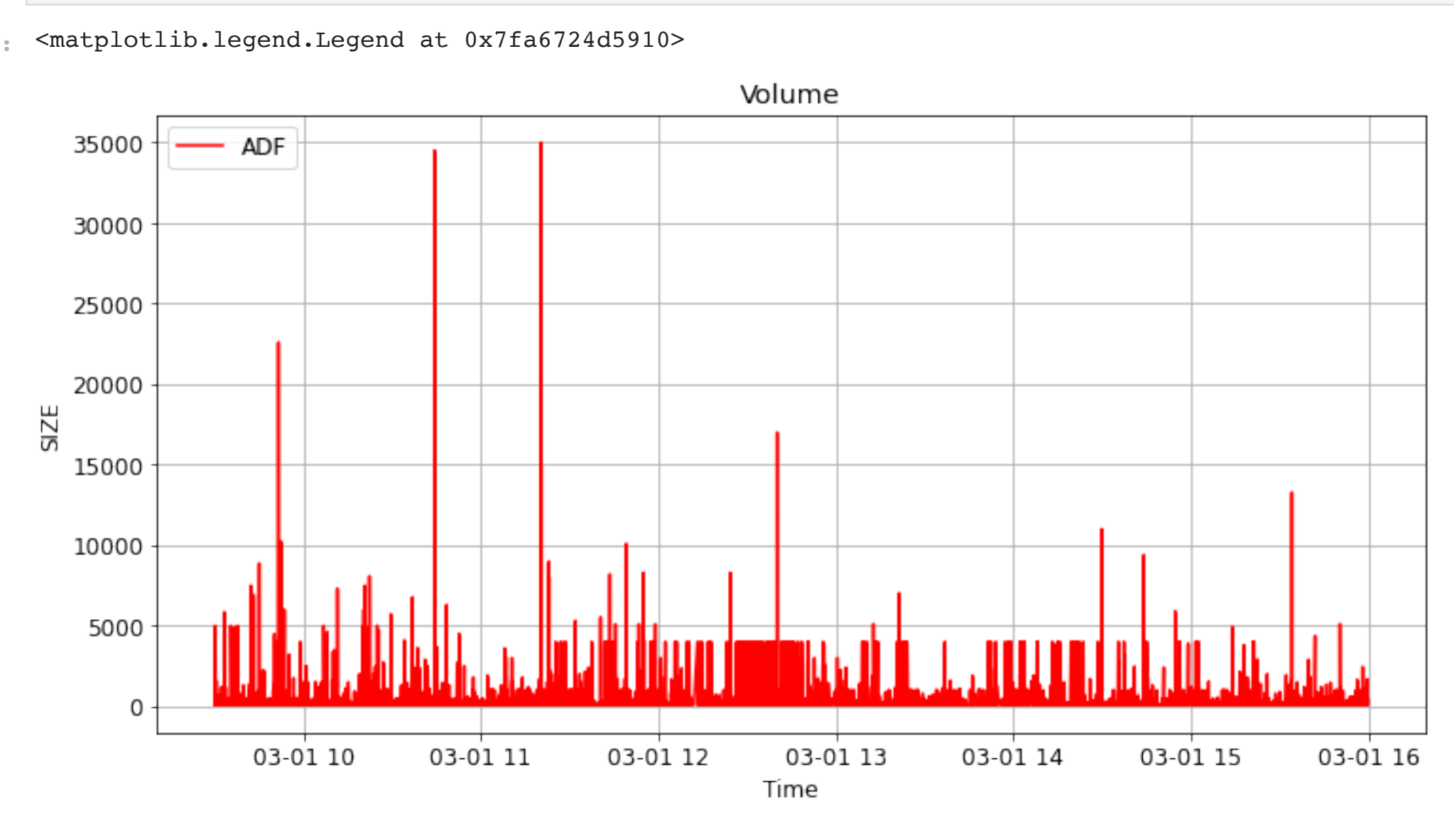
Out[10]: <matplotlib.legend.Legend at 0x7fa640859fa0>



In [11]:

```python
# Specify area
plt.rc('font', size=12)
fig, ax = plt.subplots(figsize=(12, 6))

# Specify lines
ax.plot(df1.EST, df1.SIZE, color='red', label='ADF')

# Same as above
ax.set_xlabel('Time')
ax.set_ylabel('SIZE')
ax.set_title('Volume')
ax.grid(True)
ax.legend(loc='upper left')
```

Out[11]: <matplotlib.legend.Legend at 0x7fa6724d5910>

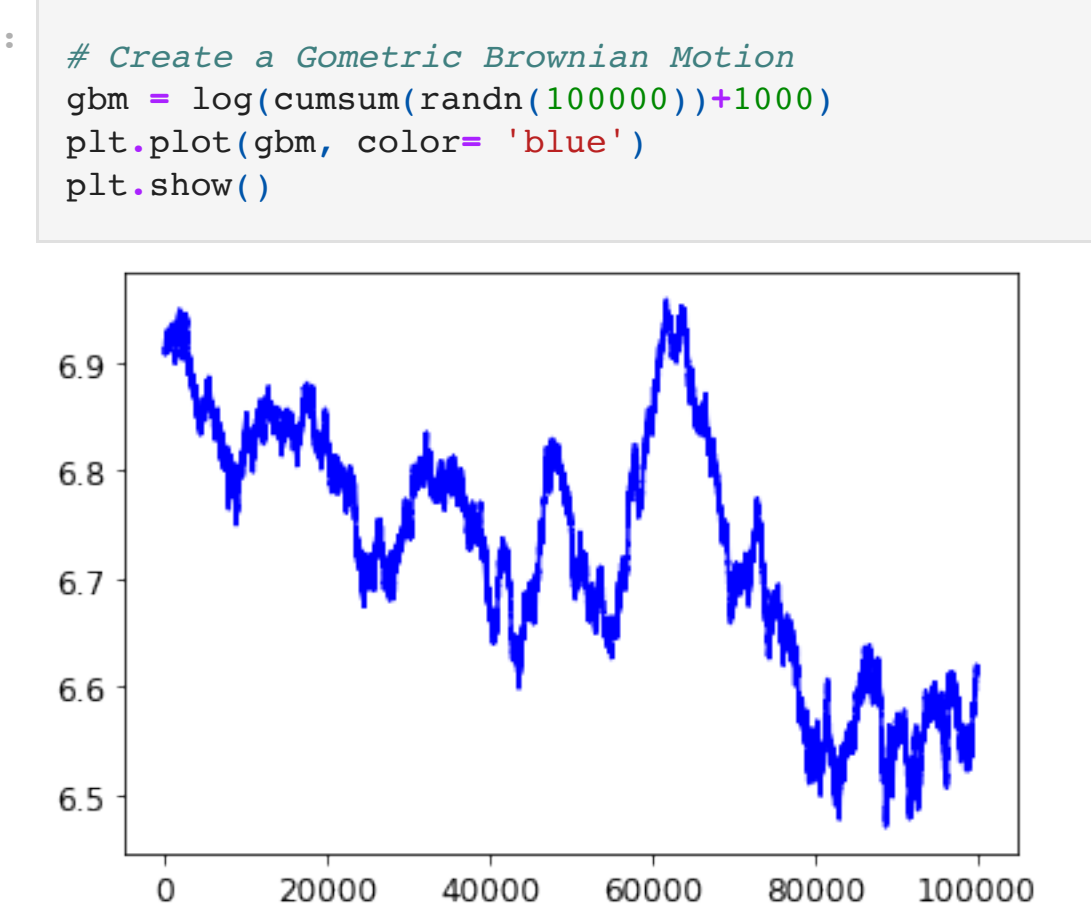


In [14]:

```python
from numpy import cumsum, log, polyfit, sqrt, std, subtract
from numpy.random import randn

def hurst(ts):
    """Returns the Hurst Exponent of the time series vector ts"""
    # Create the range of lag values
    lags = range(2, 100)

    # Calculate the array of the variances of the lagged differences
    tau = [sqrt(std(subtract(ts[lag:], ts[:-lag]))) for lag in lags]

    # Use a linear fit to estimate the Hurst Exponent
    poly = polyfit(log(lags), log(tau), 1)

    # Return the Hurst exponent from the polyfit output
    return poly[0]*2.0
```

In [16]:

```python
# Create a Gometric Brownian Motion
gbm = log(cumsum(randn(100000))+1000)
plt.plot(gbm, color= 'blue')
plt.show()
```



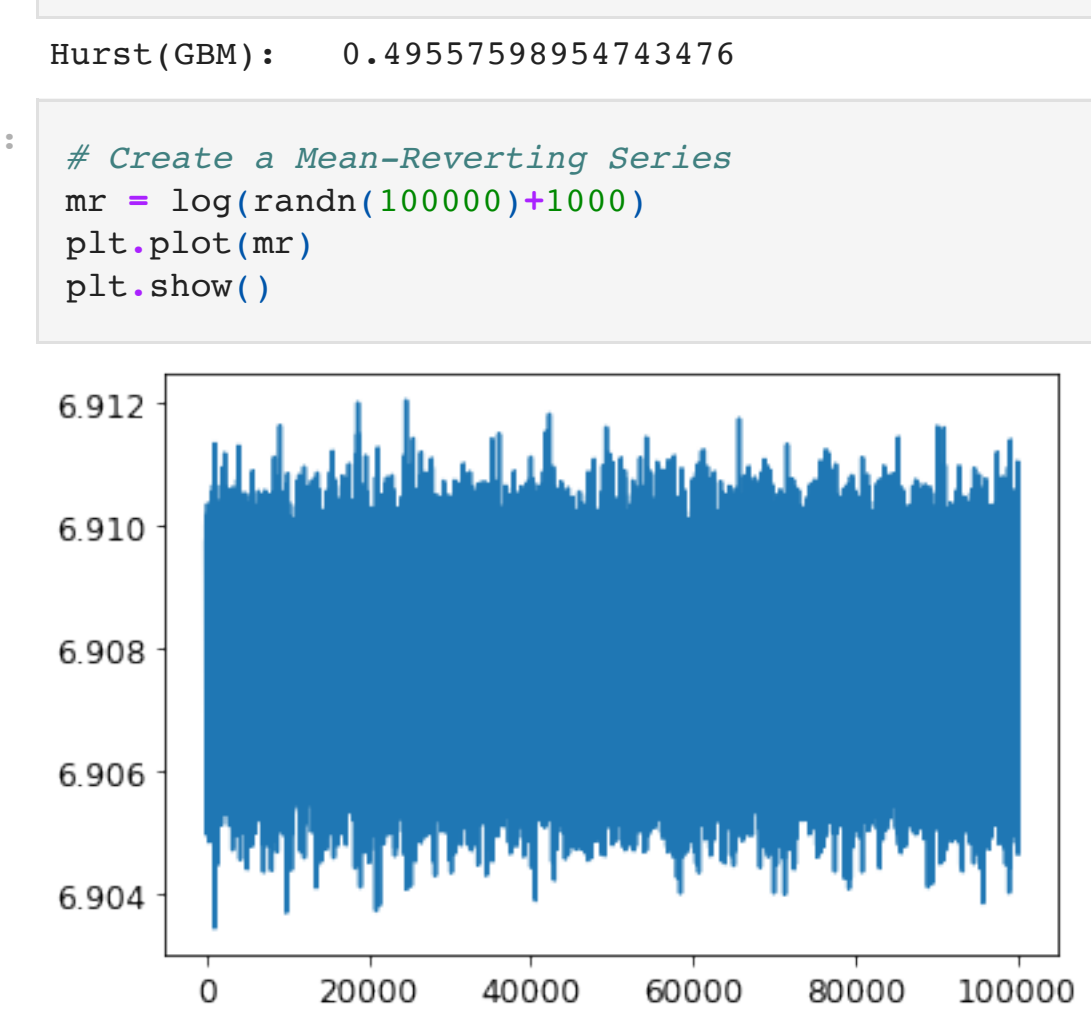
In [17]:

```python
#Calculate Hurst Exponent for GBM
print("Hurst(GBM):   %s" % hurst(gbm))
```

```
Hurst(GBM):   0.49557598954743476
```

In [18]:

```python
# Create a Mean-Reverting Series
mr = log(randn(100000)+1000)
plt.plot(mr)
plt.show()
```



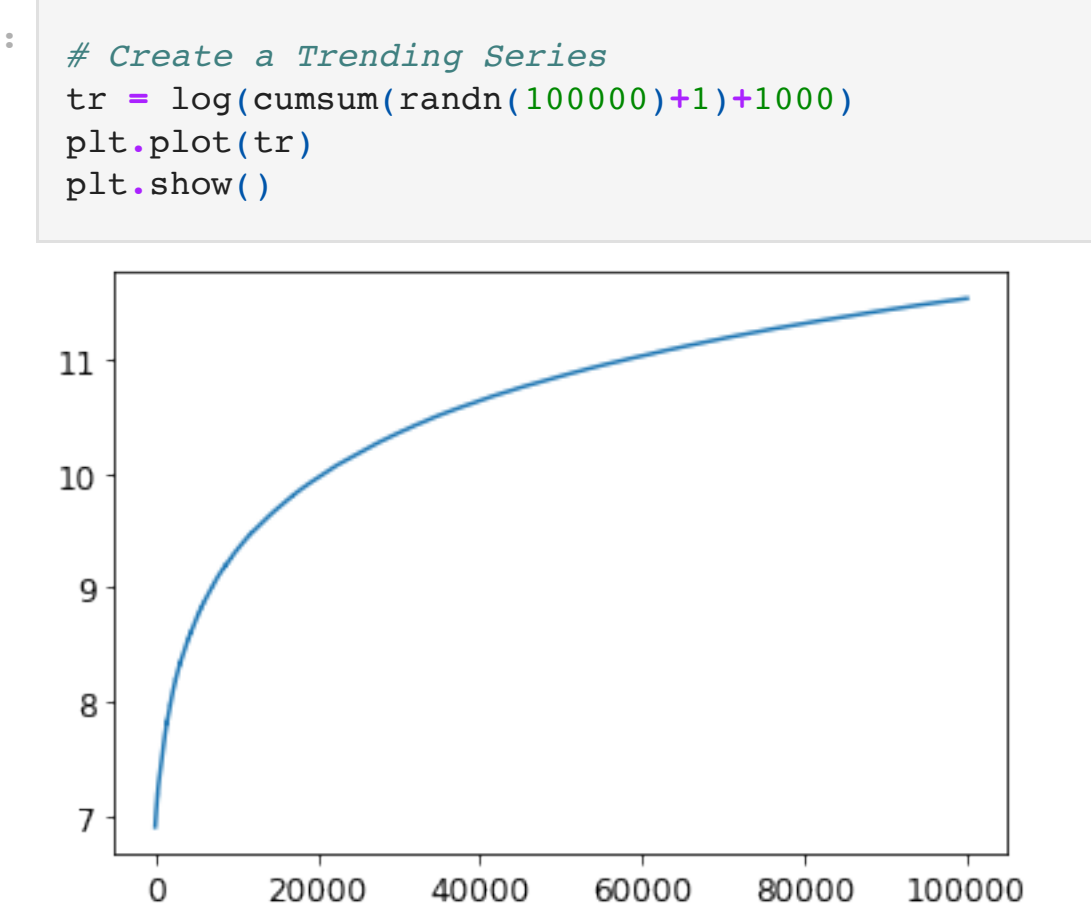
In [19]:

```python
#Calculate Hurst Exponent for MR
print("Hurst(MR):    %s" % hurst(mr))
```

```
Hurst(MR):    -0.00018669427857835027
```

In [20]:

```python
# Create a Trending Series
tr = log(cumsum(randn(100000)+1)+1000)
plt.plot(tr)
plt.show()
```



In [21]:

```python
print("Hurst(TR):    %s" % hurst(tr))
```

```
Hurst(TR):    0.9556007935298709
```

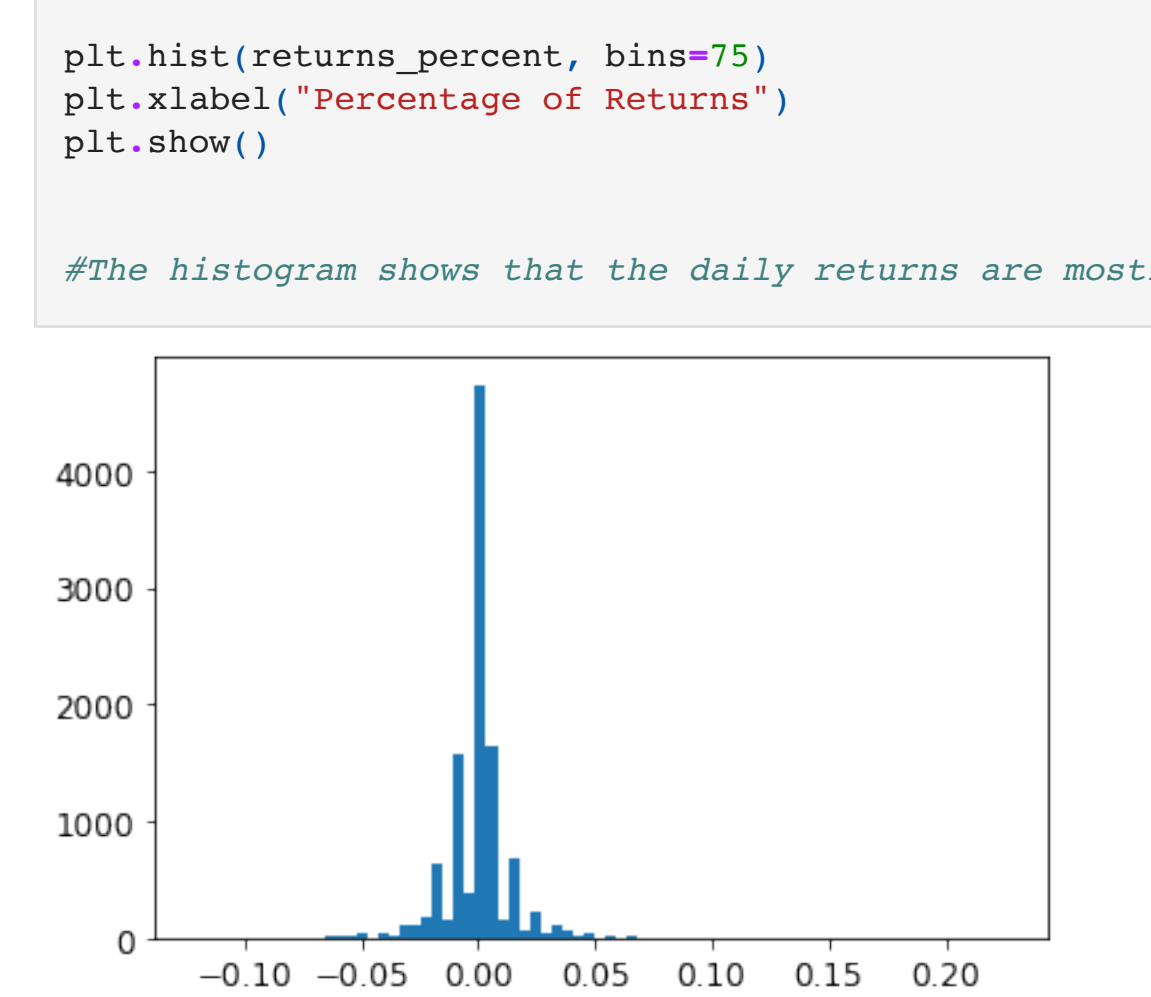
In [22]:

```python
print("Hurst(ADF):    %s" % hurst(df1[['PRICE']]))
```

```
Hurst(ADF):    [0.52607515]
<ipython-input-14-5f50f003dfc8>:10: FutureWarning: Calling a ufunc on non-aligned DataFrames (or DataFrame/Series combinatio
n). Currently, the indices are ignored and the result takes the index/columns of the first DataFrame. In the future , the Da
taFrames/Series will be aligned before applying the ufunc.
Convert one of the arguments to a NumPy array (eg 'ufunc(df1, np.asarray(df2)') to keep the current behaviour, or align manu
ally (eg 'df1, df2 = df1.align(df2)') before passing to the ufunc to obtain the future behaviour and silence this warning.
  tau = [sqrt(std(subtract(ts[lag:], ts[:-lag]))) for lag in lags]
```

In [24]:

```python
#create a histogram to visualize the distribution of the returns.

returns_percent = df1['Return'] * 100
returns_percent.dropna(inplace=True)

plt.hist(returns_percent, bins=75)
plt.xlabel("Percentage of Returns")
plt.show()


#The histogram shows that the daily returns are mostly distributed around -5% to 5%.
```



In [27]:

```python
#calculate the mean of the daily returns and its annualized average returns.

daily_mean_returns = np.mean(df1['Return'])
print("Daily Mean Returns: " , str(daily_mean_returns))

annual_mean_returns = ((1+ daily_mean_returns)**252)-1
print("Annual Mean Returns: " , str(annual_mean_returns))
```

```
Daily Mean Returns:  9.59223046419362e-08
Annual Mean Returns:  2.417271174093294e-05
```

In [29]:

```python
#The variance of returns

daily_std = np.std(df1['Return'])
annual_std = daily_std*np.sqrt(252)
print("Daily Standard Deviation " , str(daily_std))
print("Annual Standard Deviation " , str(annual_std ))

print('\n')

daily_var= daily_std**2
annual_var = annual_std**2
print("Daily Variance " , str(daily_var))
print("Annual Variance " , str(annual_var))
```

```
Daily Standard Deviation  0.00013379220824243692
Annual Standard Deviation  0.0021238854622059253


Daily Variance  1.7900354986387607e-08
Annual Variance  4.510889456569677e-06
```

In [31]:

```python
#Skewness
#Skewness is the measurement of asymmetry of a symmetric probability distribution.
#A normal distribution is the probability distribution with zero skewness.

from scipy.stats import skew
returns = df1['Return'].dropna()
skewness = skew(returns)
print("Skewness: " + str(skewness))
```

```
Skewness: 0.3602776943259701
```

In [32]:

```python
# Kurtosis
#Kurtosis is a measurement of the tail-heaviness of a probability distribution.
#A distribution with a high kurtosis value will show much larger extreme values
#on either side of its tails compared with the tails of a normal distribution.

from scipy.stats import kurtosis
from scipy.stats import shapiro


excess_kurtosis = kurtosis(returns)
print("Excess kurtosis: " + str(excess_kurtosis))

real_kurtosis = excess_kurtosis + 3
print("Real kurtosis: " + str(real_kurtosis))
```

```
Excess kurtosis: 14.913011156180783
Real kurtosis: 17.913011156180783
```

In [ ]:

```python
#The resulting kurtosis value is very high which means an investor of this asset might
#experience a high kurtosis risk
```