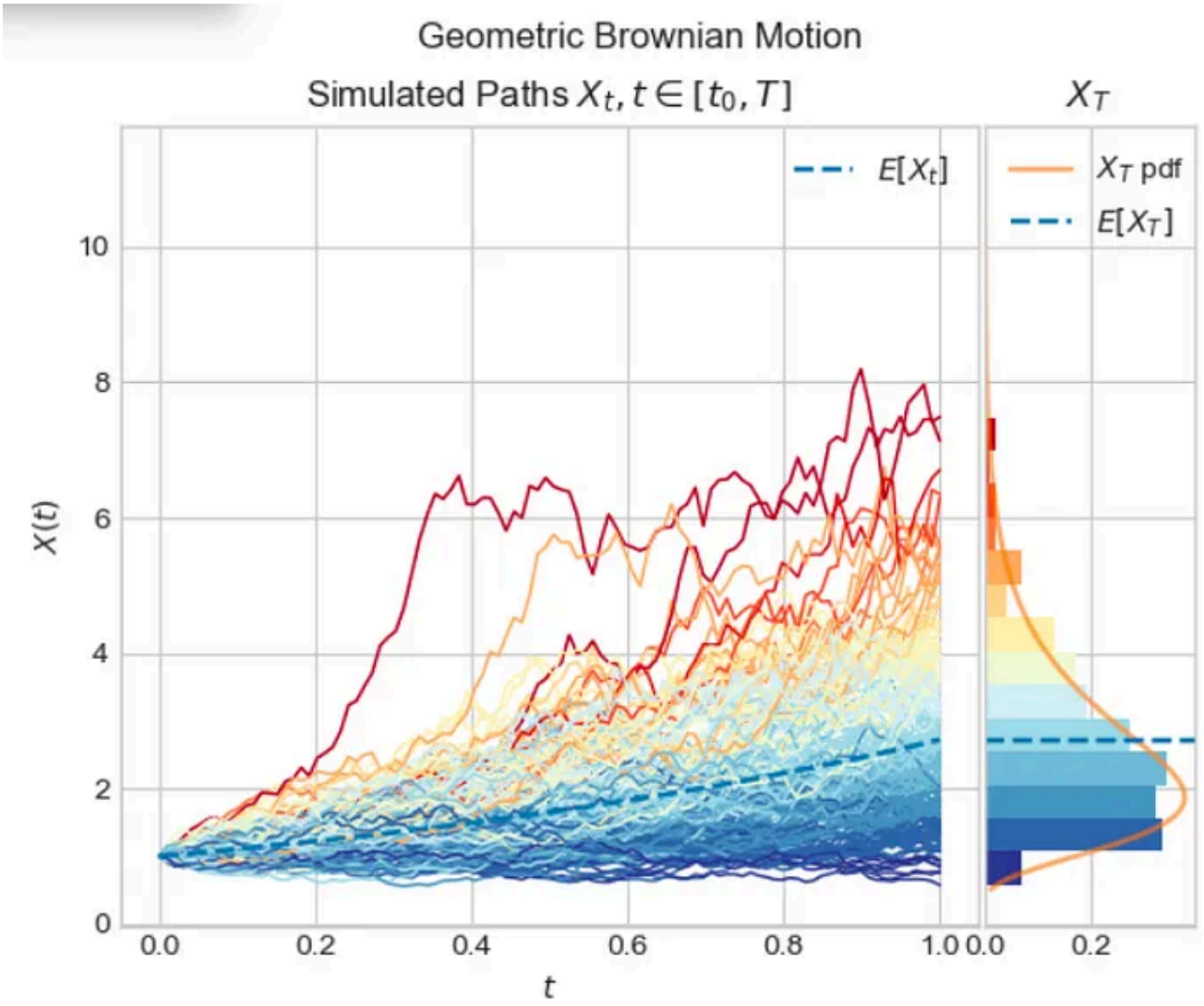
Riding the waves of randomness: using Brownian motion for stock price simulations in Python

Dimitrios Koulialias PhD · Following



Introduction

As an ubiquitous phenomenon in nature, **Brownian motion** refers to the random motion of particles within a medium, usually a gas or a liquid. Having been discovered by the botanist Robert Brown in 1827 through a microscopy analysis on pollen seeds immersed in water, it took about further eighty years to establish a profound quantitative understanding of this phenomenon. Among other scientists, noteworthy are hereby the contributions by Albert Einstein that provided a theoretical foundation of Brownian motion associated with diffusion [1, 2].

In simple terms, the idea behind Brownian motion is that a trajectory follows a random and unpredictable path. This idea has been widely utilized

across several scientific and application-oriented disciplines, among which also includes **quantitative finance** for describing the random behavior of **stock price movements.** In this story, we are going to use Python in order to carry out stock price simulations based on Brownian motion.

> *Disclaimer: This article is only for educational purposes and shall not be taken as an investment, financial, or other professional advice.*

Standard Brownian motion (Wiener process)

The stochastic nature of stock prices relies on the empirical observation that daily returns nearly follow a **normal distribution.** The driving force behind this part is the **standard Brownian motion**, also known as **Wiener process.** Specifically, for an increment $\{W(t) : t \geq 0\}$ accounting for a change in stock price at time t, the Wiener process exhibits the following characteristics:

1. It starts at zero, i.e., $W(0) = 0$
2. For any timestep dt, the difference between the subsequent and current increment follows a **normal distribution:** $W(t+dt) - W(t) \sim N(0, dt)$
3. The evolution of $W(t)$ is continuous

An example for implementing $W(t)$ in Python is given in the script below, along with one plot of a possible evolution

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
```

```
#Function for calculating the temporal evolution of W
#for n=1000 timesteps
def wiener_process(dt=0.1, x0=0, n=1000):

    # Initialize W(t)
    W = np.zeros(n+1)

    # Create n+1 timesteps: t=0,1,2,3...n
    t = np.linspace(x0, n, n+1)

    # Use the cumulative sum for calculating W at every timestep
    W[1:n+1] = np.cumsum(npr.normal(0, np.sqrt(dt), n))

    return t, W

# Function for plotting W
def plot_process(t, W):
    plt.plot(t, W)
    plt.xlabel('Time(t)')
    plt.ylabel('W(t)')
    plt.title('Wiener process')
    plt.show()

[t, W] = wiener_process()

plot_process(t, W)
```

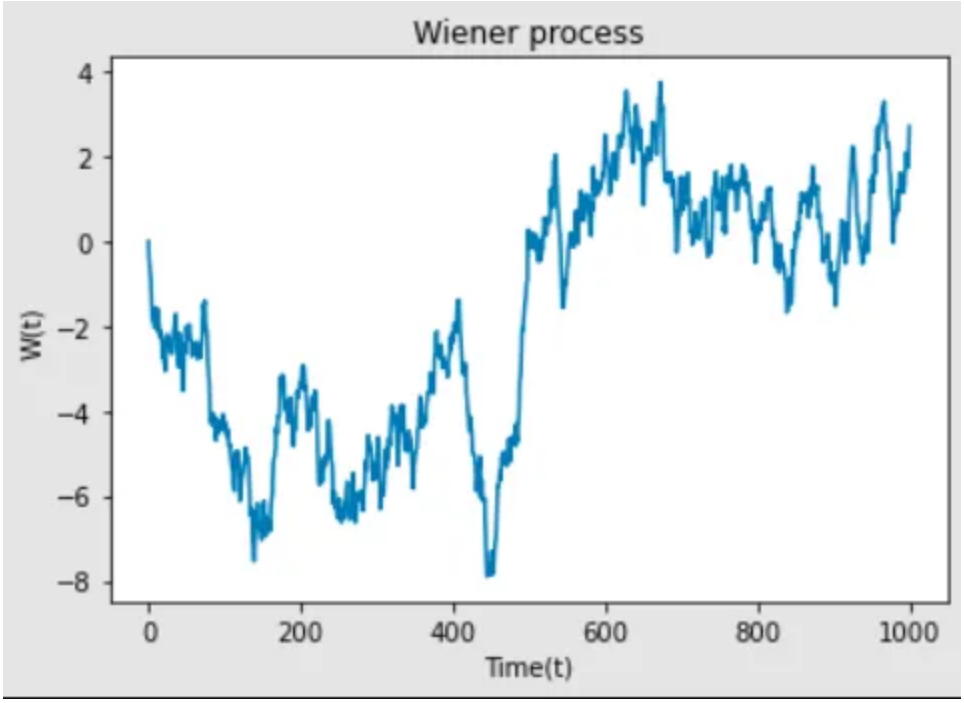


Figure 1: Possible trajectory W(t) as calculated from the above Python script.

Stock price behavior described by arithmetic Brownian motion

Given the characteristics of the Wiener process stated above, it can theoretically represent the daily stock price fluctuations without considering any bullish or bearish behavior (i.e., growing or declining trend in price). Because, however, such case does not occur for actual stocks, we also need to take into account a **deterministic** component, i.e., a **drift**, that is positive (negative) in case of a bullish (bearish) trend.

With this in mind, the stock price, denoted *S(t)* at time *t,* experiences a change dS_t during a timestep *dt* that is given by

$$dS_t = \mu dt + \sigma dW_t \quad (1)$$

which represents an **arithmetic Brownian motion.** From eq. (1), we see that two fundamental quantities enter in the expressions of the deterministic and stochastic components:

- The average **return** μ that is responsible for the drift
- The **risk/volatility** σ that accounts for the stochastic behavior associated with the Wiener process

A qualitative representation of *S(t)* given by eq. (1) is shown in Figure 2

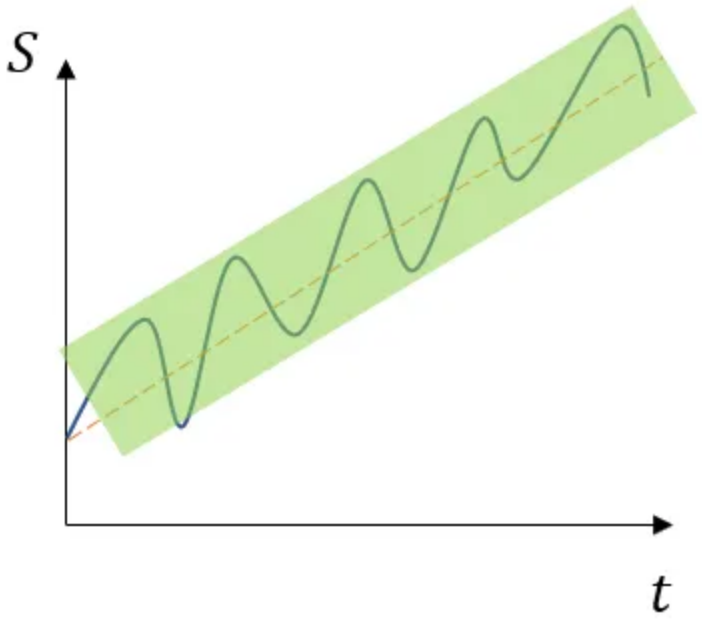


Figure 2: Schematic delineation of a stock price evolution S(t) (solid line) under an arithmetic Brownian motion. The linear component refers to the drift term (dashed line), whereas the stochastic component stems from the Wiener process that accounts for the price fuctuations (green highlited area).

A drawback with eq. (1), however, is that stock prices within this model can generally attain negative values. Because in reality this never happens, we need to encounter a different version of eq. (1), in which the stock is not subjected to a normal, but a log-normal distribution instead. This is achieved using **geometric Brownian motion.**

Stock price simulation with Geometric Brownian motion

The geometric Brownian motion is characterized by

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad (2)$$

that has the following solution:

$$S_t = S_0 \exp\left((\mu - \frac{\sigma^2}{2})t + \sigma W_t \right) \quad (3)$$

In this form, since *S(t)* follows a log-normal distribution, the non-negativity of the stock price is always met. We now can make use of eq. (3) to simulate the evolution of a stock price with the following parameters

- $S(0) = S_0 = 10$ $
- $\mu = 0{,}1$
- $\sigma = 0{,}05$

within the timespan [0, *T*] (using $T = 1$ by default), based on the below .py script:

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt

def simulate_geometric_brownian_motion(S0, T=1, N=1000, mu=0.1, sigma=0.05):

    #timestep dt
    dt = T/N

    # Initialize W(t)
    W = np.zeros(N+1)

    # Create N timesteps
    t = np.linspace(0, T, N+1)

    # Use the cumulative sum for calculating W at every timestep
    W[1:N+1] = np.cumsum(npr.normal(0, np.sqrt(dt), N))

    #Calculate S(t)
    S = S0 * np.exp((mu - 0.5 * sigma ** 2) * t + sigma * W)

    return t, S
```

```
def plot_simulation(t, S):
    plt.plot(t, S)
    plt.xlabel('Time (t)')
    plt.ylabel('Stock Price S(t)')
    plt.title('Geometric Brownian Motion')
    plt.show()

#Calculate S with an initial price S0 = 10 $
[t, S] = simulate_geometric_brownian_motion(10)

plot_simulation(t, S)
```

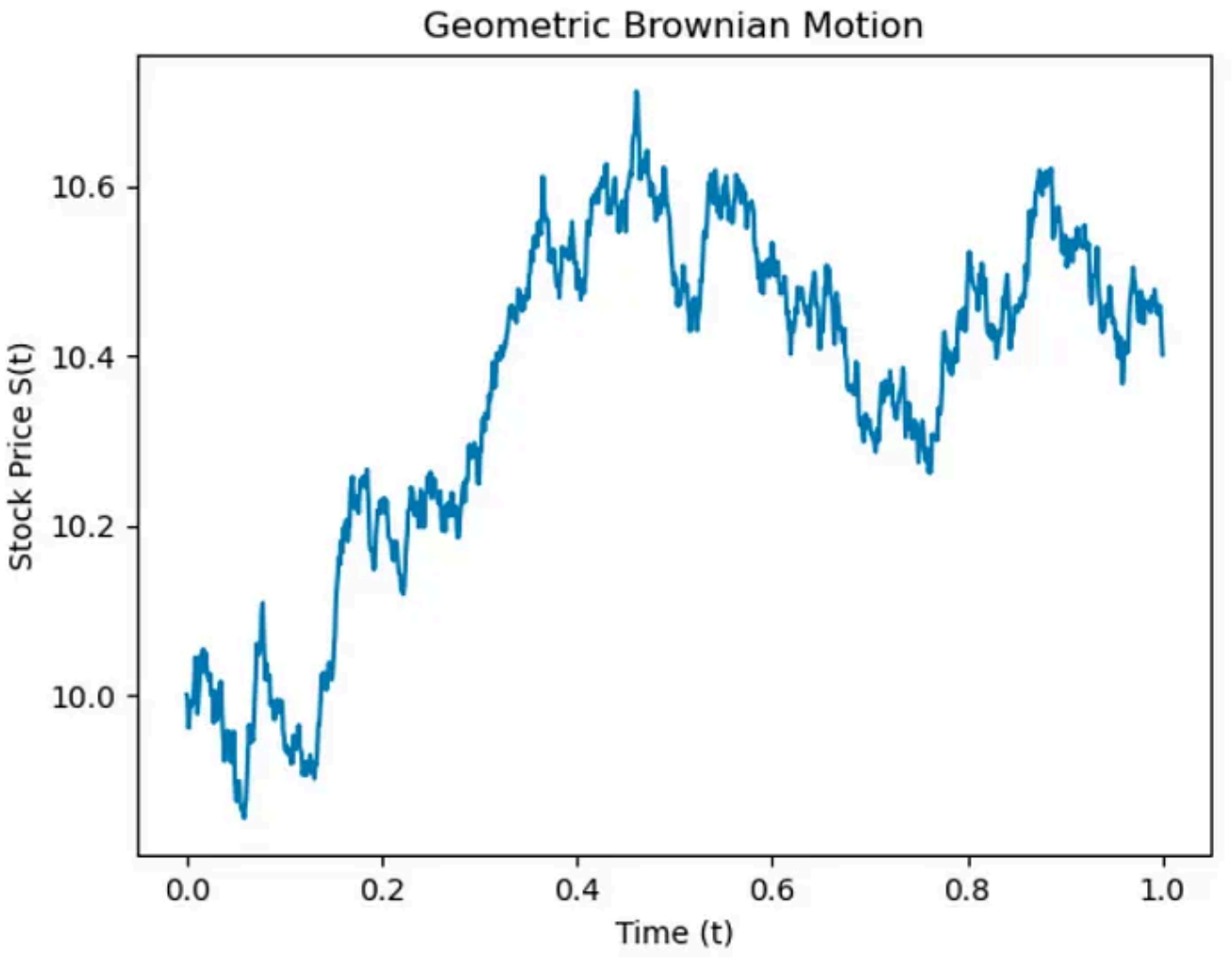


Figure 3: Simulated stock price path using eq. (3) with the parameters S0 = 10 $, μ = 0.1, and σ = 0.05

In order to infer whether $S(t)$ truly follows a log-normal distribution, one would need to conduct many trajectories of geometric Brownian motions, and then plot the distributions of $S(t)$. This can be verified using **Aleatory**, one powerful Python library for simulating and visualising stochastic processes [3].

Simulating Geometric Brownian motion with Aleatory

The simulation is pretty straightforward, since all the process conducted in the above .py script, is implemented in Aleatory within a class called **GBM**. Using any development environment, e.g., Jupyter Notebook, an instance of GBM can be created as

```
# Install aleatory, if necessary
pip install aleatory

# Import the geometric Brownian motion class GBM
from aleatory.processes import GBM

# Create an instance of GBM
geometric_brownian_motion = GBM(drift=1, volatility=0.5, initial=1.0, T=1.0, rng
```

with the following parameters (using again $T = 1$ by default)

- $S_0 = 1$ \$
- $\mu = 1$
- $\sigma = 0.5$

The last entry refers to the type of random number generator, which is given as None by default. Given this, the plot can be drawn by using the draw method,

```
geometric_brownian_motion.draw(n=100, N=200)
```

where n corresponds to the number of steps in each trajectory, and N to the number of trajectories to simulate.

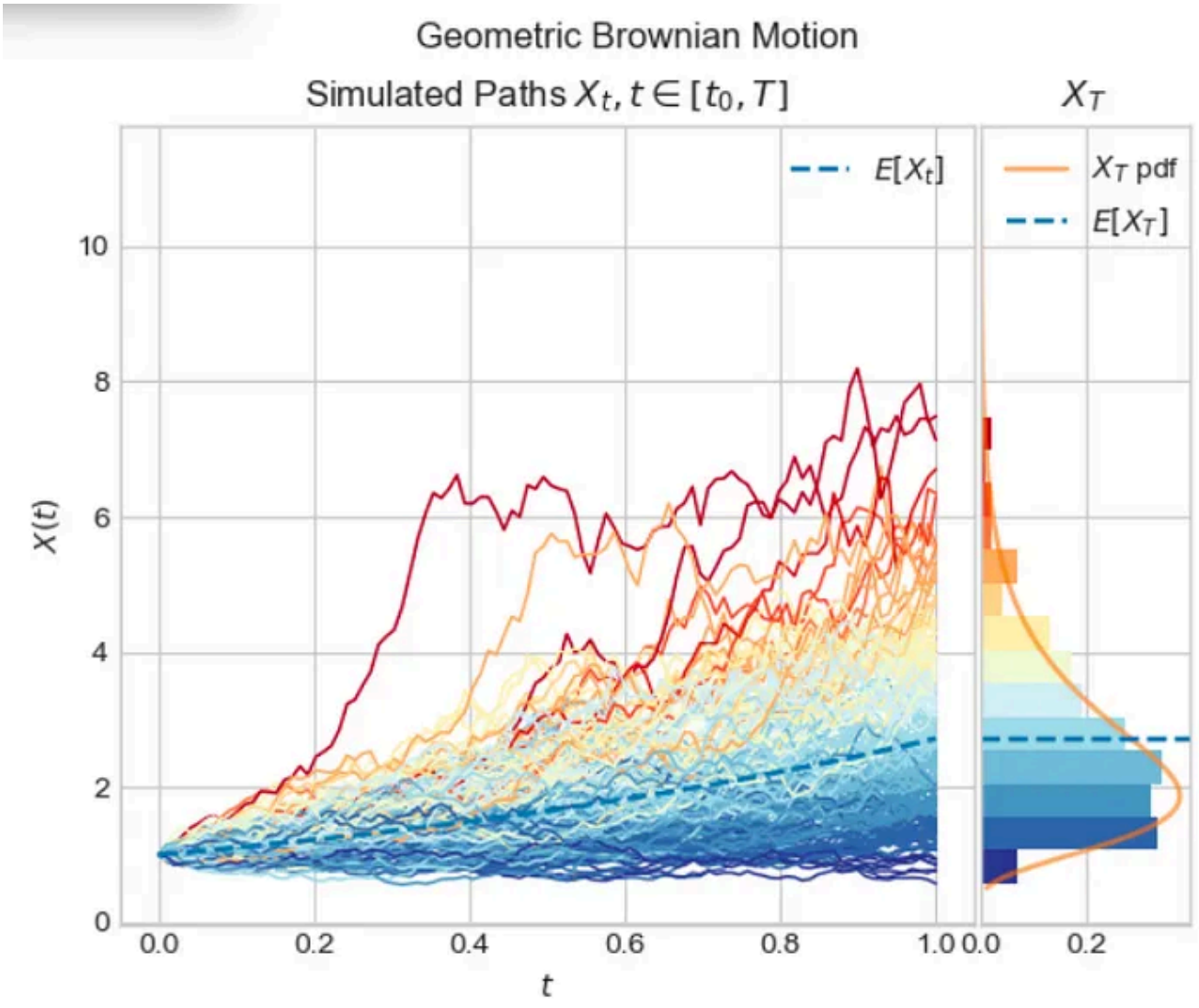


Figure 4: Simulated stock price trajectories with the Aleatory package (main plot), along with their distribution at T = 1 (right plot)

As shown in Figure 4, the simulation yields several stock price trajectories, with a log-normal distribution, as indicated by its distinct skewness.

Conclusion

In summary, using Brownian motion to simulate stock prices can provide a deeper insight into market dynamics, as well as into potential future trends of stocks. While Python is equipped with the extensive Aleatory library to carry out a variety of such simulations, it is also important to note that Brownian motion offers a simplified representation of stock movements, and that it does not capture other factors that can have a critical impact under real world market conditions (economic events, interest rates, etc.). Nevertheless, when complemented with further analyses and empirical data, Brownian motion is the tool of choice for informed decision-making in the financial market.

Sources

[1] A. Einstein, *Ann. Phys.* **17**, 549 (1905).

[2] A. Einstein, *Zeitschrift fuer Elektrochemie und angewandte physikalische Chemie* **13**, 41 (1907).

[3] aleatory 0.4.0 documentation

Stocks Python Quantitative Finance Financial Modelling

Published in Nerd For Tech

11.9K Followers · Last published 13 hours ago

Follow

NFT is an Educational Media House. Our mission is to bring the invaluable knowledge and experiences of experts from all over the world to the novice. To know more about us, visit https://www.nerdfortech.org/.

Written by Dimitrios Koulialias PhD

375 Followers · 613 Following

Following

Physicist | Data and Finance enthusiast | Lifelong learner| PhD from Swiss Federal Institute of Technology (ETH Zurich)

Responses (2)

Alex Mylnikov

What are your thoughts?

NeuralNode

Aug 31, 2024

Love the visuals the GBM tool creates.

6 1 reply Reply

Godwin Feranmi
Sep 16, 2024

Hello coach, are you available for a brief chat concerning your Udemy course. i have a quick review

https://www.udemy.com/course/web-scraping-and-data-analysis-with-python/

Reply

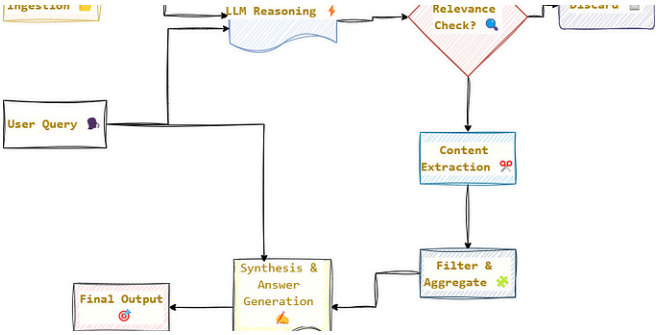
More from Dimitrios Koulialias PhD and Nerd For Tech

In The Pythoneers by Dimitrios Koulialias PhD

Tech-savvy saving: how to build a budget tracker with a few lines of...

Part 2—Implementation

Mar 14, 2024 56



In Nerd For Tech by Plaban Nayak

Fixing RAG with Reasoning Augmented Generation

🛠️ Why RAG is Broken—And How ReAG Fixes It

Feb 16 296 5

In Nerd For Tech by Dilka Sithari

How to Resolve the “‘mvn’ is not recognized as an internal or...

Maven (mvn) commands are widely used to automate tasks such as managing...

Oct 7, 2024 33

Dimitrios Koulialias PhD

A Python-based analysis on crude oil price data extracted from Yaho...

Introduction

Jan 31, 2024 21 2

See all from Dimitrios Koulialias PhD

See all from Nerd For Tech

Recommended from Medium

In Coding Nexus by Algo Insights

Cracking Multi-Timeframe Analysis with VectorBT Pro

Okay, let's talk multi-timeframe (MTF) analysis—a trading trick that sounds like...

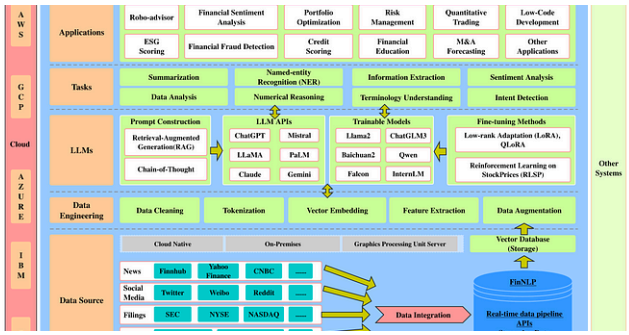
Mar 10 5 1

Martin Bauer

A Dynamic ETF Rotation Strategy (Part I)

Choosing between growth and value stocks is one of the most challenging decisions...

Oct 27, 2024



In AI monks.io by JIN

FinGPT: The Future of Financial Analysis—Revolutionizing Marke...

Discover how FinGPT is disrupting traditional financial tools like Bloomberg Terminal,...

Feb 16 650 10

In InsiderFinance Wire by Manish Peshwani

Reverse DCF: Valuing Stocks with Precision using Python

Discounted Cash Flow (DCF) analysis is a cornerstone of stock valuation, enabling...

Dec 31, 2024 158 1

Navnoor Bawa

Statistical Arbitrage with Deep RL: Solving the Sharpe Ratio Paradox

Implementation Results & The Sharpe Ratio Challenge

In Coinm... by MicroBioscopicData (by Alexandros...

Introduction to Portfolio Management using Python (5)

Reduce Portofolio Risk by Diversification

Mar 17 5 ••• Sep 27, 2024 110 •••

See more recommendations