Monte Carlo Simulation for Portfolio Returns

Goal: Simulate thousands of price paths and estimate future portfolio value distribution.

What You'll Learn:

- Geometric Brownian Motion (GBM)
- Drift, volatility & randomness in market outcomes
- Basic Monte Carlo framework
- Value at Risk (VaR) estimation

In [1]:
```
# Install dependencies if needed
%pip install numpy scipy yfinance pandas matplotlib --quiet
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from typing import Tuple, Dict, Optional

print(f"NumPy: {np.__version__}")
print(f"Pandas: {pd.__version__}")
```

```
NumPy: 2.2.5
Pandas: 2.2.3
```

Step 1 – Geometric Brownian Motion (GBM)

The GBM model describes the stochastic evolution of an asset price:

$$dS = \mu S\, dt + \sigma S\, dW$$

Where:

- S : Asset price
- μ : Drift (annualized expected return)
- σ : Annualized volatility
- W : Wiener process (Brownian motion)

Exact Solution:

$$S(t) = S(0) \exp\left[\left(\mu - \frac{\sigma^2}{2}\right) t + \sigma W(t)\right]$$

Discretization for Simulation:

$$S_{t+\Delta t} = S_t \exp\left[\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}\, Z\right]$$

where $Z \sim \mathcal{N}(0,1)$

Step 2 – Download Historical Data

In [3]:
```
# Download S&P 500 data
ticker = "^GSPC"
period = "5y"

print(f"Downloading {ticker} data ({period})...")
data = yf.download(ticker, period=period, progress=False)
print(f"  -> {len(data)} trading days loaded")
print(f"  -> Date range: {data.index[0].strftime('%Y-%m-%d')} to {data.index[-1].st

data.tail()
```

```
Downloading ^GSPC data (5y)...
YF.download() has changed argument auto_adjust default to True
  -> 1256 trading days loaded
  -> Date range: 2020-12-22 to 2025-12-22
```

Out[3]:

Price	Close	High	Low	Open	Volume
Ticker	^GSPC	^GSPC	^GSPC	^GSPC	^GSPC
Date					
2025-12-16	6800.259766	6819.270020	6759.740234	6800.120117	4983180000
2025-12-17	6721.430176	6812.259766	6720.430176	6802.879883	5122120000
2025-12-18	6774.759766	6816.129883	6758.500000	6778.060059	5101190000
2025-12-19	6834.500000	6840.020020	6792.620117	6792.620117	8554470000
2025-12-22	6858.459961	6873.890137	6855.740234	6865.209961	610072344

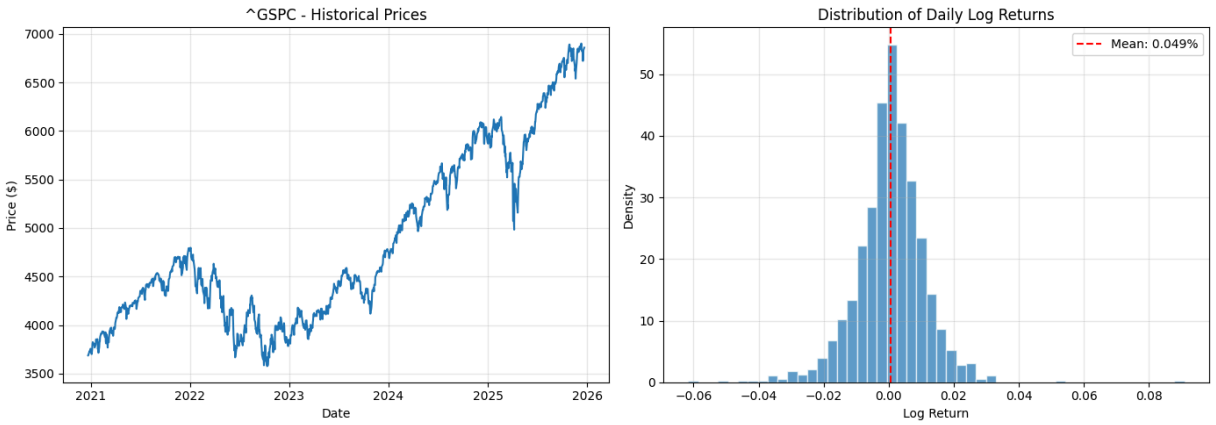
Step 3 – Estimate Parameters (μ and σ)

In [4]:
```
# Calculate log returns
prices = data['Close'].squeeze()  # Force to Series (handles MultiIndex)
log_returns = np.log(prices / prices.shift(1)).dropna()

# Annualize (252 trading days)
daily_mu = float(log_returns.mean())      # Force to float
daily_sigma = float(log_returns.std())    # Force to float

mu = daily_mu * 252
sigma = daily_sigma * np.sqrt(252)
```

```
print("="*50)
print("ESTIMATED PARAMETERS")
print("="*50)
print(f"Daily mean return:        {daily_mu*100:.4f}%")
print(f"Daily volatility:         {daily_sigma*100:.4f}%")
print()
print(f"Annualized return (mu):   {mu*100:.2f}%")
print(f"Annualized volatility (sigma): {sigma*100:.2f}%")
```

```
==================================================
ESTIMATED PARAMETERS
==================================================
Daily mean return:        0.0495%
Daily volatility:         1.0678%

Annualized return (mu):   12.46%
Annualized volatility (sigma): 16.95%
```

In [5]:
```
# Visualize historical returns distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Price history
axes[0].plot(prices.index, prices.values)
axes[0].set_title(f'{ticker} - Historical Prices')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('Price ($)')
axes[0].grid(True, alpha=0.3)

# Returns distribution
axes[1].hist(log_returns, bins=50, density=True, alpha=0.7, edgecolor='white')
axes[1].axvline(daily_mu, color='red', linestyle='--', label=f'Mean: {daily_mu*100:
axes[1].set_title('Distribution of Daily Log Returns')
axes[1].set_xlabel('Log Return')
axes[1].set_ylabel('Density')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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



Step 4 – Simulate Price Paths (GBM)

In [6]:
```python
def simulate_gbm_paths(
    S0: float,
    mu: float,
    sigma: float,
    T: float,
    n_simulations: int,
    n_steps: int,
    seed: int = 42
) -> np.ndarray:
    """
    Simulate price paths using Geometric Brownian Motion.

    Parameters:
        S0: Initial price
        mu: Drift (annualized)
        sigma: Volatility (annualized)
        T: Time horizon in years
        n_simulations: Number of paths to simulate
        n_steps: Number of time steps
        seed: Random seed for reproducibility

    Returns:
        np.ndarray of shape (n_steps+1, n_simulations)
    """
    np.random.seed(seed)

    dt = T / n_steps

    # Initialize price matrix
    prices = np.zeros((n_steps + 1, n_simulations))
    prices[0] = S0

    # Generate random shocks
    Z = np.random.standard_normal((n_steps, n_simulations))

    # Simulate using exact GBM solution
    for t in range(1, n_steps + 1):
        prices[t] = prices[t-1] * np.exp(
            (mu - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z[t-1]
        )

    return prices
```

In [7]:
```python
# Simulation parameters
S0 = 10000          # Initial portfolio value ($)
T = 1.0             # 1 year horizon
n_simulations = 10000
n_steps = 252       # Daily steps (trading days)

print("Running Monte Carlo simulation...")
paths = simulate_gbm_paths(S0, mu, sigma, T, n_simulations, n_steps)
print(f"  -> Simulated {n_simulations:,} paths with {n_steps} steps each")
print(f"  -> Shape: {paths.shape}")
```

```
Running Monte Carlo simulation...
  -> Simulated 10,000 paths with 252 steps each
  -> Shape: (253, 10000)
```

Step 5 – Analyze Results

In [8]:

```python
# Final values
final_values = paths[-1]
returns = (final_values - S0) / S0

# Compute statistics
stats = {
    'mean': np.mean(final_values),
    'median': np.median(final_values),
    'std': np.std(final_values),
    'min': np.min(final_values),
    'max': np.max(final_values),
    'percentile_5': np.percentile(final_values, 5),
    'percentile_25': np.percentile(final_values, 25),
    'percentile_75': np.percentile(final_values, 75),
    'percentile_95': np.percentile(final_values, 95),
    'prob_profit': np.mean(returns > 0),
    'prob_loss_10pct': np.mean(returns < -0.10),
    'mean_return': np.mean(returns),
    'median_return': np.median(returns),
}

print("="*60)
print("MONTE CARLO SIMULATION RESULTS")
print("="*60)
print(f"\nSimulation Parameters:")
print(f"  Initial Investment: ${S0:,.2f}")
print(f"  Time Horizon: {T} year(s)")
print(f"  Number of Simulations: {n_simulations:,}")
print(f"  Estimated mu: {mu*100:.2f}%")
print(f"  Estimated sigma: {sigma*100:.2f}%")

print(f"\n{'Portfolio Value Statistics':^50}")
print("-"*50)
print(f"  Mean:         ${stats['mean']:>12,.2f}")
print(f"  Median:       ${stats['median']:>12,.2f}")
print(f"  Std Dev:      ${stats['std']:>12,.2f}")
print(f"  Min:          ${stats['min']:>12,.2f}")
print(f"  Max:          ${stats['max']:>12,.2f}")

print(f"\n{'Percentiles':^50}")
print("-"*50)
print(f"  5th:          ${stats['percentile_5']:>12,.2f}")
print(f"  25th:         ${stats['percentile_25']:>12,.2f}")
print(f"  75th:         ${stats['percentile_75']:>12,.2f}")
print(f"  95th:         ${stats['percentile_95']:>12,.2f}")

print(f"\n{'Risk Metrics':^50}")
print("-"*50)
print(f"  VaR 95% (5th percentile):  ${stats['percentile_5']:>10,.2f}")
```

```
print(f"  Probability of Profit:     {stats['prob_profit']*100:>10.2f}%")
print(f"  Probability of >10% Loss:  {stats['prob_loss_10pct']*100:>10.2f}%")

print(f"\n{'Expected Returns':^50}")
print("-"*50)
print(f"  Mean Return:   {stats['mean_return']*100:>10.2f}%")
print(f"  Median Return: {stats['median_return']*100:>10.2f}%")
print("="*60)
```

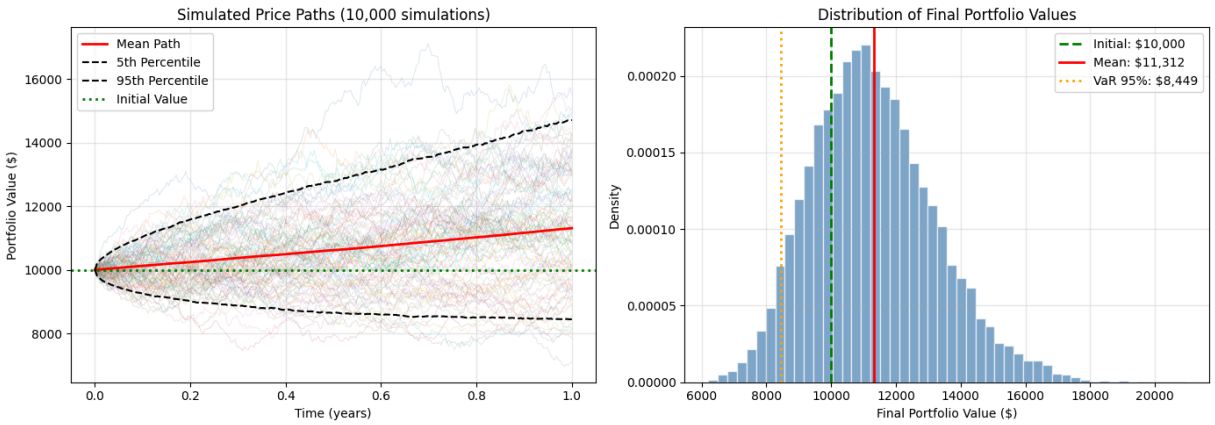
```
============================================================
MONTE CARLO SIMULATION RESULTS
============================================================

Simulation Parameters:
  Initial Investment: $10,000.00
  Time Horizon: 1.0 year(s)
  Number of Simulations: 10,000
  Estimated mu: 12.46%
  Estimated sigma: 16.95%

            Portfolio Value Statistics
--------------------------------------------------
  Mean:         $   11,312.22
  Median:       $   11,159.68
  Std Dev:      $    1,923.58
  Min:          $    6,211.77
  Max:          $   20,940.03

                   Percentiles
--------------------------------------------------
  5th:          $    8,449.47
  25th:         $    9,949.37
  75th:         $   12,504.57
  95th:         $   14,714.79

                   Risk Metrics
--------------------------------------------------
  VaR 95% (5th percentile):  $  8,449.47
  Probability of Profit:          73.92%
  Probability of >10% Loss:       10.40%

                 Expected Returns
--------------------------------------------------
  Mean Return:        13.12%
  Median Return:      11.60%
============================================================
```

Step 6 – Visualize Results

In [9]:

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

time_axis = np.linspace(0, T, n_steps + 1)

# --- Plot 1: Sample paths ---
ax1 = axes[0]
```

```
n_paths_to_show = 100

for i in range(n_paths_to_show):
    ax1.plot(time_axis, paths[:, i], alpha=0.2, linewidth=0.5)

# Mean and percentiles
ax1.plot(time_axis, np.mean(paths, axis=1), 'r-', linewidth=2, label='Mean Path')
ax1.plot(time_axis, np.percentile(paths, 5, axis=1), 'k--', linewidth=1.5, label='5
ax1.plot(time_axis, np.percentile(paths, 95, axis=1), 'k--', linewidth=1.5, label='
ax1.axhline(S0, color='green', linestyle=':', linewidth=2, label='Initial Value')

ax1.set_xlabel('Time (years)')
ax1.set_ylabel('Portfolio Value ($)')
ax1.set_title(f'Simulated Price Paths ({n_simulations:,} simulations)')
ax1.legend(loc='upper left')
ax1.grid(True, alpha=0.3)

# --- Plot 2: Distribution ---
ax2 = axes[1]
ax2.hist(final_values, bins=50, density=True, alpha=0.7, color='steelblue', edgecol

ax2.axvline(S0, color='green', linestyle='--', linewidth=2, label=f'Initial: ${S0:,
ax2.axvline(stats['mean'], color='red', linestyle='-', linewidth=2, label=f'Mean: $
ax2.axvline(stats['percentile_5'], color='orange', linestyle=':', linewidth=2, labe

ax2.set_xlabel('Final Portfolio Value ($)')
ax2.set_ylabel('Density')
ax2.set_title('Distribution of Final Portfolio Values')
ax2.legend(loc='upper right')
ax2.grid(True, alpha=0.3)

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



Interpretation of Results

1. Simulated Price Paths

- The **red line** represents the mean path across all simulations
- The **dashed black lines** show the 90% confidence interval (5th and 95th percentiles)

- Dispersion increases over time — this is the effect of compounding volatility

2. Final Value Distribution

- The distribution is **log-normal** (characteristic of GBM)
- Positive skewness: potential gains are unlimited, while losses are bounded at -100%
- The mean is typically higher than the median due to this asymmetry

3. Value at Risk (VaR)

- VaR at 95% indicates the maximum expected loss in 95% of cases
- It's a standard risk measure in portfolio management
- Example: If VaR 95% = $8,500, there's only a 5% chance of ending below this value

Connection to Black-Scholes

The Black-Scholes model uses exactly the same GBM process:

Approach	Use Case	Method
Black-Scholes	European options	Closed-form analytical solution
Monte Carlo	Complex derivatives, portfolio analysis	Numerical simulation

Both share:

- The same price dynamics: $dS = \mu S\,dt + \sigma S\,dW$
- The same parameters (μ, σ)
- The assumption of log-normal returns

Why use Monte Carlo instead of Black-Scholes?

- Path-dependent options (Asian, barrier, lookback)
- American options (early exercise)
- Multi-asset portfolios
- Complex payoff structures

Model Limitations

1. **Constant volatility assumption**: In reality, volatility is time-varying (see GARCH models)
2. **Log-normal returns**: Markets exhibit fat tails and excess kurtosis
3. **Independence**: Daily returns show some autocorrelation (momentum, mean reversion)
4. **Historical estimation**: Past performance doesn't guarantee future results

These limitations are shared with Black-Scholes — practitioners often use more sophisticated models:

- GARCH for volatility clustering
- Stochastic volatility (Heston model)
- Jump-diffusion processes

As Warren Buffett noted, these models can give "an illusion of precision" — always combine quantitative analysis with fundamental understanding.

Summary

This notebook demonstrated:

1. **Data acquisition**: Downloading historical prices via yfinance
2. **Parameter estimation**: Computing annualized return (μ) and volatility (σ)
3. **GBM simulation**: Generating thousands of possible future price paths
4. **Risk analysis**: VaR, percentiles, probability of profit/loss
5. **Visualization**: Price paths and final value distribution

The Monte Carlo framework is foundational for:

- Portfolio risk management
- Option pricing (when closed-form solutions don't exist)
- Stress testing and scenario analysis