monte_carlo

September 3, 2019

1 Monte Carlo Simulation

[3]:
```
# Initial imports
import requests
import pandas as pd
import numpy as np

iex_key = "your_key_goes_here"

%matplotlib inline
```

1.1 Fetch Tickers Data

[4]:
```
def get_ticker_prices(ticker):
    """
    Retrieves one year old historical prices for the ticket based on the␣
 ↪current date.
    """

    # Define API endpoint
    url = "https://cloud.iexapis.com/stable/stock/{}/chart/1y?token={}".format(
        ticker, iex_key
    )

    # Retrieve historical prices data
    data = requests.get(url).json()

    # Create response DataFrame taking only the date and the close price
    df = pd.DataFrame(
        [{"date": price["date"], "close_" + ticker: price["close"]} for price␣
 ↪in data]
    )

    # Define DataFrame's index
    df.set_index("date", inplace=True)
```

```python
    return df
```

[5]:
```python
spy_data = get_ticker_prices("spy")
agg_data = get_ticker_prices("agg")
tickers_data = pd.merge(spy_data, agg_data, on="date")
tickers_data.head()
```

[5]:
```
            close_spy  close_agg
date
2018-09-04     289.81     106.06
2018-09-05     289.03     106.01
2018-09-06     288.16     106.19
2018-09-07     287.60     105.81
2018-09-10     288.10     105.91
```

1.2 Monte Carlo Simulation Code

[6]:
```python
# Calculate the daily roi for the stocks
daily_returns = tickers_data.pct_change()
print("*" * 100)
print("Daily ROI")
print("*" * 100)
display(daily_returns.head())

# volatility
daily_volatility = daily_returns.std()
spy_volatility = daily_volatility["close_spy"]
agg_volatility = daily_volatility["close_agg"]

# Save the last day's closing price
spy_last_price = tickers_data["close_spy"][-1]
agg_last_price = tickers_data["close_agg"][-1]
```

```
********************************************************************************
********************
Daily ROI
********************************************************************************
********************

            close_spy  close_agg
date
2018-09-04        NaN        NaN
2018-09-05  -0.002691  -0.000471
2018-09-06  -0.003010   0.001698
2018-09-07  -0.001943  -0.003578
2018-09-10   0.001739   0.000945
```

[7]:
```
# Setup the Monte Carlo Parameters
number_simulations = 10
number_records = 252 * 30  # Years to retirement
monte_carlo = pd.DataFrame()
```

[8]:
```
# Run the Monte Carlo Simulation
for x in range(number_simulations):

    print(f"Running Simulation {x}...")

    # Create the initial simulated prices array seeded with the last closing␣
 ↪price
    spy_prices = [spy_last_price]
    agg_prices = [agg_last_price]

    # Simulate the returns for 20 years
    for iteration in range(number_records):
        spy_prices.append(
            spy_prices[-1]
            * (1 + np.random.normal(daily_returns.mean()["close_spy"],␣
 ↪spy_volatility))
        )
        agg_prices.append(
            agg_prices[-1]
            * (1 + np.random.normal(daily_returns.mean()["close_agg"],␣
 ↪agg_volatility))
        )

    # Create a DataFrame of the simulated prices
    portfolio = pd.DataFrame(
        {"SPY Simulated Prices": spy_prices, "AGG Simulated Prices": agg_prices}
    )

    # Calculate the Portfolio Daily Returns
    portfolio_returns = portfolio.pct_change()

    # Set the Portfolio Weights (Assume a 60/40 stocks to bonds ratio)
    stocks_weight = 0.60
    bonds_weight = 0.40

    # Calculate the weighted portfolio return:
    portfolio_returns = (
        stocks_weight * portfolio_returns["SPY Simulated Prices"]
        + bonds_weight * portfolio_returns["AGG Simulated Prices"]
    )

    # Calculate the normalized, cumulative return series
```

```
    monte_carlo[x] = (1 + portfolio_returns.fillna(0)).cumprod()
```

```
Running Simulation 0...
Running Simulation 1...
Running Simulation 2...
Running Simulation 3...
Running Simulation 4...
Running Simulation 5...
Running Simulation 6...
Running Simulation 7...
Running Simulation 8...
Running Simulation 9...
```

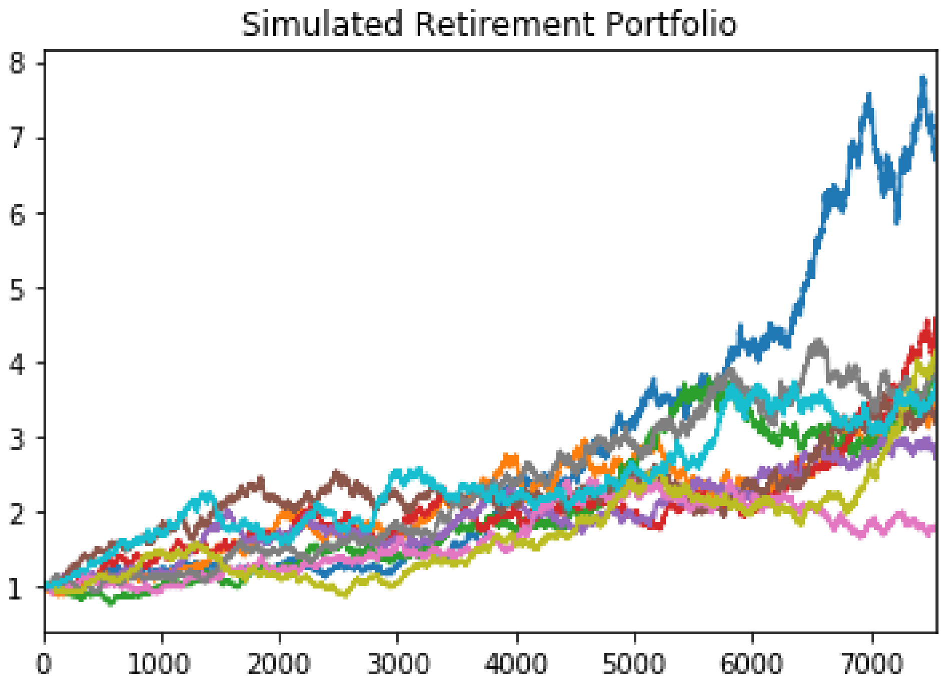
[9]:
```
# Check that the simulation ran successfully
monte_carlo.head()
```

[9]:
```
          0         1         2         3         4         5         6  \
0  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
1  1.005482  1.002578  1.000634  0.995945  0.998586  0.997353  1.003083
2  1.002088  1.006828  1.013192  1.003073  1.001271  0.992538  1.002229
3  0.997989  1.007777  1.006345  1.004643  1.000379  0.991181  0.999913
4  1.006052  1.001195  1.007911  0.998880  0.999869  0.993946  0.996089

          7         8         9
0  1.000000  1.000000  1.000000
1  1.000698  0.996601  1.001807
2  0.997392  0.998590  0.991981
3  0.992141  1.003347  0.996694
4  0.994661  1.002287  0.999100
```

[10]:
```
# Visualize the Simulation
monte_carlo.plot(legend=None, title="Simulated Retirement Portfolio")
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x121f178d0>
```


Simulated Retirement Portfolio
8
7
6
5
4
3
2
1
0
1000
2000
3000
4000
5000
6000
7000