Monte Carlo Simulation

In [1]:
```python
# Initial imports
import requests
import pandas as pd
import numpy as np
!pip install alpaca_trade_api
import alpaca_trade_api as tradeapi

%matplotlib inline
```

In [2]:
```python
# Set Alpaca API key and secret
alpaca_api_key = "your_api_key_here"
alpaca_secret_key = "your_secret_key_here"

api = tradeapi.REST(alpaca_api_key, alpaca_secret_key, api_version='v2')
```

Fetch Tickers Data

In [3]:
```python
def get_ticker_prices(ticker):

    # Set timeframe to '1D'
    timeframe = '1D'

    # Make the API cal and store in DataFrame
    data_df = api.get_barset(
        ticker,
        timeframe,
        limit=None,
        after=None,
        until=None,
    ).df

    # Clean DataFrame to show only close prices
    df = pd.DataFrame({'close_'+ticker.lower():data_df[ticker]['close']})
    return df
```

In [4]:

```
spy_data = get_ticker_prices("SPY")
agg_data = get_ticker_prices("AGG")
tickers_data = spy_data.join(agg_data)
tickers_data.head()
```

Out[4]:

	close_spy	close_agg
2020-01-10 00:00:00-05:00	325.70	112.99
2020-01-13 00:00:00-05:00	327.94	112.89
2020-01-14 00:00:00-05:00	327.43	113.00
2020-01-15 00:00:00-05:00	328.17	113.19
2020-01-16 00:00:00-05:00	330.91	113.15

Monte Carlo Simulation Code

In [5]:

```
# 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
2020-01-10 00:00:00-05:00	NaN	NaN
2020-01-13 00:00:00-05:00	0.006877	-0.000885
2020-01-14 00:00:00-05:00	-0.001555	0.000974
2020-01-15 00:00:00-05:00	0.002260	0.001681
2020-01-16 00:00:00-05:00	0.008349	-0.000353

In [6]:
```
# Setup the Monte Carlo Parameters
number_simulations = 10
number_records = 252 * 30  # Years to retirement
monte_carlo = pd.DataFrame()


```

In [7]:

```python
# 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
    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_v
        )
        agg_prices.append(
            agg_prices[-1]
            * (1 + np.random.normal(daily_returns.mean()["close_agg"], agg_v
        )

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

    # 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...
```

In [8]:

```
# Check that the simulation ran successfully
monte_carlo.head()


```

Out[8]:

	0	1	2	3	4	5	6	7	8	
0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00
1	1.001941	1.018772	0.991946	1.033178	0.999727	1.031796	0.996773	1.018177	0.991689	1.00
2	0.998508	1.005907	1.010379	1.026256	0.999801	1.004254	1.015160	1.016009	0.987007	1.02
3	1.011969	0.998476	1.009407	1.015949	1.017484	1.018086	1.007551	1.056503	1.028288	1.03
4	1.005709	1.016011	1.020407	1.014806	1.021678	1.037358	1.025767	1.035644	1.016543	1.06

In [9]:

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

```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1f906662b38>