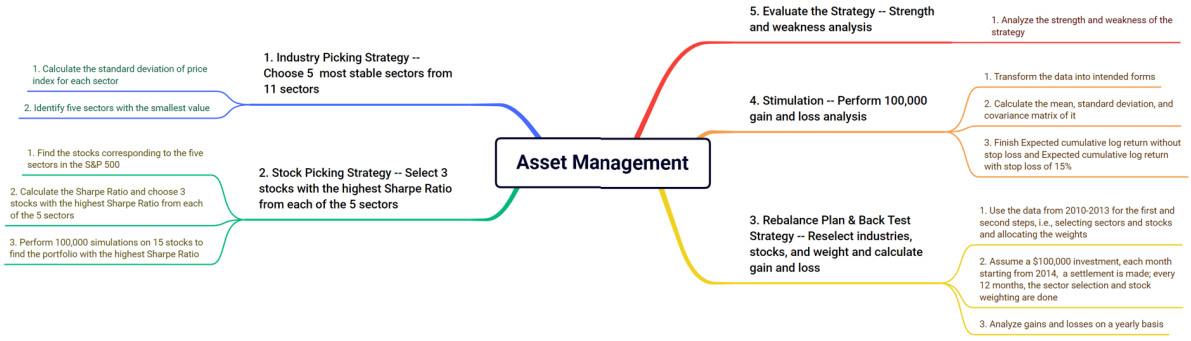
Asset Management

0. Overview



1. Introduction

- **Core Idea:** Constructed the portfolio using the top-down approach --- **1.Current macroeconomic environment; 2.Most stable industries; 3.Stocks that can yield the most profit**.
- **Background:** The hawkish remarks of Fed officials put pressure on the stock index, and the three major US stock indexes closed across the board. As the 2-year US bond yield continues to rise, fears of a recession are growing in the market. The economic situation is unstable.
- **Role: A risk-averse investor**. Maximize investment returns while ensuring soundness.

- **Assumption:**
  1. Ignore transaction fees regardless of the number of transactions made.
  2. Adjust trading strategies before the next quote appears.
  3. Assume that the price of the U.S. treasury bill stays the same, i.e., it has an interest rate of 1.00%.

2. Industry Picking Strategy

In this section, we completed the selection of investment sectors：

> **1.** Use the **Global Industry Classification Standard (GICS)** to classify stocks in the S&P500 into 11 sectors.

> **2.** Download monthly price index data of the 11 sectors from January 2010 to December 2013 and calculate their **standard deviation**.
>
> **3. Select 5 sectors whose standard deviation** is the smallest from the total 11 sectors, convincing that they are the most stable 5 industries. In order of standard deviation from smallest to largest, these five sectors are **Utility Industry, Communication Services Industry, Health Care Industry, Information Technology Industry, and Industrials Industry**.

In [1]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import openpyxl as p
import yfinance as yf
from statsmodels.formula.api import ols
import statsmodels.formula.api as smf
```

Code Explanation:

- Define the function to get the development of sectors for a certain period, then calculate and sort **the standard deviation** of each sector.
  (Data Source: https://www.spglobal.com/ratings/en/)

In [2]:
```
def get_stable_sectors(start_time, end_time):
    indus = pd.read_csv('11 sectors.csv')
    indus = indus.set_index('Date')
    indus_3y = indus.loc[start_time:end_time]
    x = indus_3y.std()
    x = x.sort_values()

    return x.index[0:5]
```

Code Explanation:

- Find the **five smallest standard deviation sectors**, which are the most stable, and print them out.

In [3]:
```
stocks_number = 15
indus_start = '201001'
indus_end = '201312'

most_stable_indus = get_stable_sectors(indus_start, indus_end)

print('The 1st stable industry from Jan. 2010 to Dec. 2013 is ' + most_stable_
print('The 2nd stable industry from Jan. 2010 to Dec. 2013 is ' + most_stable_
print('The 3rd stable industry from Jan. 2010 to Dec. 2013 is ' + most_stable_
print('The 4th stable industry from Jan. 2010 to Dec. 2013 is ' + most_stable_
print('The 5th stable industry from Jan. 2010 to Dec. 2013 is ' + most_stable_
```

```
The 1st stable industry from Jan. 2010 to Dec. 2013 is Utilities Industry.
The 2nd stable industry from Jan. 2010 to Dec. 2013 is Communication Services
Industry.
The 3rd stable industry from Jan. 2010 to Dec. 2013 is Health Care Industry.
The 4th stable industry from Jan. 2010 to Dec. 2013 is Information Technology
Industry.
The 5th stable industry from Jan. 2010 to Dec. 2013 is Industrials Industry.
```

2. Stock Picking Strategy

In this section, we completed the selection of investment stocks and the assignment of stocks' weights:

> **1.** We download the stocks in the S&P500 corresponding to the 5 selected sectors from **yahoo finance** for the period from January 2010 to December 2013, using the adj closed price of each stock on the last day of each month as **its monthly price**.
>
> **2.** We converted the prices to log form to calculate the monthly return for each stock and then calculated the average return and volatility for each stock and **the Sharpe ratio**. From each industry, we selected the top 3 stocks in the sectors in terms of the Sharpe ratio. The codes of the 15 stocks we selected are: **'SO', 'ED', 'DUK', 'T', 'VZ', 'LUMN', 'JNJ', 'BDX', 'ABT', 'ADP', 'PAYX', 'IBM', 'LMT', 'WM', and 'UPS'**.
>
> **3.** We used the code to perform **100000 weight simulations** and draw scatter plots of returns and standard deviation for 10 stocks with 100000 different weight combinations and found the weight combined with the largest Sharpe ratio.
>
> **4.** As shown in the figure, we found the **tangent point of the CML with the efficient frontier**, which is the optimal portfolio point we are looking for. Our portfolio has **a volatility of 9.30%, an expected return of 16.75%, and a Sharpe Ratio of 1.694**.

Code Explanation:

- Get S&P500 stock symbols from Wikipedia and set column 'Symbol' as tickers.
- Use for loop to adjust the format of the stock code in the 'Symbol' column.

In [4]:
```
# Get S&P500 stocks symbols from Wikipedia and set column 'Symbol' as tickers.
sp500url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
data_table = pd.read_html(sp500url)
tickers = data_table[0]

# Use for loop to adjust the format of the stock code in the 'Symbol' column.
for i in range(len(tickers)):
    if tickers['Symbol'][i] == 'BRK.B':
        tickers['Symbol'][i] == 'BRK-B'
    elif tickers['Symbol'][i] == 'BF.B':
```

```
        tickers['Symbol'][i] == 'BF-B'

display(tickers)
```

	Symbol	Security	SEC filings	GICS Sector	GICS Sub-Industry	Headquarters Location	Date first added	
0	MMM	3M	reports	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1976-08-09	66
1	AOS	A. O. Smith	reports	Industrials	Building Products	Milwaukee, Wisconsin	2017-07-26	91
2	ABT	Abbott	reports	Health Care	Health Care Equipment	North Chicago, Illinois	1964-03-31	1
3	ABBV	AbbVie	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	2012-12-31	1551
4	ABMD	Abiomed	reports	Health Care	Health Care Equipment	Danvers, Massachusetts	2018-05-31	815
...	...	...	...	...	...	...	...	...
498	YUM	Yum! Brands	reports	Consumer Discretionary	Restaurants	Louisville, Kentucky	1997-10-06	1041
499	ZBRA	Zebra Technologies	reports	Information Technology	Electronic Equipment & Instruments	Lincolnshire, Illinois	2019-12-23	877
500	ZBH	Zimmer Biomet	reports	Health Care	Health Care Equipment	Warsaw, Indiana	2001-08-07	1136
501	ZION	Zions Bancorporation	reports	Financials	Regional Banks	Salt Lake City, Utah	2001-06-22	109
502	ZTS	Zoetis	reports	Health Care	Pharmaceuticals	Parsippany, New Jersey	2013-06-21	1555

503 rows × 9 columns

Code Explanation:

- Define the function to convert daily data to monthly data and take the value of the last day.

In [5]:
```
def get_monthly_prices(stocks_data):
    stocks_data.index = pd.to_datetime(stocks_data.index)
    stocks_data['year'] = stocks_data.index.year
    stocks_data['month'] = stocks_data.index.month
    stocks_data = stocks_data.set_index([stocks_data['year'], stocks_data['mon
    stocks_data = stocks_data.groupby([stocks_data['year'], stocks_data['month
```

Code Explanation:

- Define the function to choose the three stocks from chosen industries:

- Download the stocks corresponding to the sector from yahoo finance.
  (Data Source: https://finance.yahoo.com/)
- Use function mentioned above to adjusted closing prices.
- Calculate the return of the stock in log form.
- Calculate chosen stock's **mean & standard deviation & Sharp ratio**.
- Define the function is to **choose the three stocks from each stable industries**.

In [6]:
```
def choose_stocks(stable_indus, chosen_stock1):
    # Download the stocks corresponding to the sector from yahoo finance.
    for j in range(len(stable_indus)):
        sector = tickers.loc[tickers['GICS Sector'] == stable_indus[int('{}'.f
        prices = yf.download(sector, start = start_time, end = end_time)['Adj
        prices = prices.dropna(axis = 1)
        sector = prices.columns.values

        # Use function mentioned above to adjusted closing prices.
        get_monthly_prices(prices)

        # Calculate the return of the stock in log form.
        for i in sector:
            prices['ret_{}'.format(i)] = np.log(prices['{}'.format(i)]/prices[
            prices = prices.drop(columns = ['{}'.format(i)])

        # Calculate chosen stock's mean & standard deviation & Sharp ratio.
        stock_mean = prices.mean()
        stock_vol = prices.std()
        stock_sharpe_ratio = (stock_mean - rfrate) / stock_vol
        stock_sharpe_ratio = stock_sharpe_ratio.sort_values(ascending = False)

        # Define the function is to choose the three stocks from each stable i
        chosen_stock1.append(stock_sharpe_ratio.index[-1].split('_')[1])
        chosen_stock1.append(stock_sharpe_ratio.index[-2].split('_')[1])
        chosen_stock1.append(stock_sharpe_ratio.index[-3].split('_')[1])
```

Code Explanation:

- Define the function to get the return of stock in form of log:
  - Download the chosen stocks from yahoo finance.
    (Data Source: https://finance.yahoo.com/)
  - Use function mentioned above to adjusted closing prices.
  - Calculate the **return of the stock in log form**.

In [7]:
```
def get_stock_return(choosed_stocks, start_time1, end_time1):
    # Download the chosen stocks from yahoo finance.
    down_stocks = yf.download(choosed_stocks, start = start_time1, end = end_t
    down_stocks = down_stocks.dropna(axis = 1)

    #Use function mentioned above to adjusted closing prices.
    down_stocks.index = pd.to_datetime(down_stocks.index)
    down_stocks['year'] = down_stocks.index.year
    down_stocks['month'] = down_stocks.index.month
    down_stocks = down_stocks.set_index([down_stocks['year'], down_stocks['mon
    down_stocks = down_stocks.groupby([down_stocks['year'], down_stocks['month
```

```
    # Calculate the return of the stock in log form.
    for i in choosed_stocks:
        down_stocks['ret_{}'.format(i)] = np.log(down_stocks['{}'.format(i)]/d
        down_stocks = down_stocks.drop(columns = ['{}'.format(i)])

    down_stocks.dropna(axis = 0, inplace = True)
    return down_stocks
```

Code Explanation:

- Arrange the returns of the 15 selected stocks into a Dataframe in the order of year and month.

In [8]:
```
start_time = '2010-01-01'
end_time = '2013-12-31'
rfrate = np.log(1+0.01)

chosen_stock = []
choose_stocks(most_stable_indus, chosen_stock)
deal_stocks = get_stock_return(chosen_stock, start_time, end_time)

display(deal_stocks)
```

```
[**********************100%**********************]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  70 of 70 completed

5 Failed downloads:
- HWM: No data found for this date range, symbol may be delisted
- OTIS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CARR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FTV: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- IR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  15 of 15 completed
```

year	month	ret_SO	ret_ED	ret_DUK	ret_T	ret_VZ	ret_LUMN	ret_JNJ
2010	2	-0.007213	-0.009015	0.003650	-0.021926	-0.016796	0.007616	0.009847
	3	0.042822	0.041019	-0.001837	0.040677	0.069753	0.055593	0.034325
	4	0.054341	0.014709	0.027797	0.029840	-0.055449	-0.038815	-0.013900
	5	-0.055322	-0.045968	-0.035833	-0.069925	-0.048929	0.006429	-0.088865
	6	0.017581	0.011903	0.002503	-0.004537	0.018005	-0.009188	0.012952
	7	0.072289	0.067724	0.066489	0.092891	0.118580	0.067050	-0.016560
	8	0.037772	0.042707	0.018614	0.041161	0.016044	0.015046	-0.009205
	9	0.014880	0.014413	0.030383	0.056460	0.098599	0.107270	0.083087
	10	0.028726	0.030633	0.027841	0.016321	0.011262	0.047510	0.028323
	11	-0.003969	-0.015493	-0.023534	-0.025929	-0.014576	0.038170	-0.026393
	12	0.013431	0.024506	0.014706	0.055638	0.111341	0.087985	0.004862
2011	1	-0.016085	0.006836	0.003923	-0.046229	0.008557	-0.065564	-0.034208
	2	0.025110	0.013450	0.019739	0.030772	0.035846	-0.032134	0.036561
	3	0.000000	0.014697	0.008854	0.075642	0.042943	0.008945	-0.036296
	4	0.036340	0.027228	0.027176	0.035243	-0.007120	-0.018706	0.103638
	5	0.026291	0.029023	0.018004	0.014040	-0.022755	0.057417	0.032244
	6	0.007457	0.003386	0.004257	-0.004764	0.008090	-0.048904	-0.011509
	7	-0.009186	-0.012093	-0.012290	-0.052742	-0.040528	-0.085686	-0.026352
	8	0.045002	0.077608	0.030626	-0.027019	0.024630	-0.026209	0.024338
	9	0.024126	0.014308	0.055541	0.001404	0.017267	-0.067281	-0.032592
	10	0.019400	0.014796	0.021283	0.047638	0.018711	0.062612	0.010931
	11	0.027304	0.036543	0.033002	-0.011323	0.020078	0.062125	0.014350
	12	0.052784	0.042987	0.053689	0.042560	0.061419	0.011262	0.013200
2012	1	-0.015896	-0.050759	-0.031866	-0.008481	-0.050352	-0.004581	0.005019
	2	-0.020054	-0.004410	-0.006775	0.039339	0.011878	0.083382	-0.003878
	3	0.016607	0.005493	0.004293	0.020706	0.003143	-0.022226	0.013431
	4	0.022230	0.017478	0.019793	0.071023	0.067824	-0.002331	-0.013124
	5	0.010080	0.025358	0.036940	0.037572	0.030727	0.016971	-0.032191
	6	0.008459	0.029867	0.047967	0.042682	0.065079	0.025519	0.078970
	7	0.039179	0.036472	-0.020445	0.077869	0.026830	0.050610	0.024274
	8	-0.050067	-0.052523	-0.033990	-0.034338	-0.049965	0.017184	-0.017154
	9	0.016627	-0.012115	0.000154	0.028519	0.059446	-0.027987	0.021711
	10	0.016141	0.008149	0.013795	-0.070784	-0.009662	-0.051293	0.027340
	11	-0.062312	-0.068282	-0.016467	-0.013388	-0.011717	0.011914	-0.006707

year	month	ret_SO	ret_ED	ret_DUK	ret_T	ret_VZ	ret_LUMN	ret_JNJ
	12	-0.017138	-0.004491	-0.000314	-0.012382	-0.019453	0.026030	0.005292
2013	1	0.043696	0.023840	0.074578	0.048514	0.019417	0.033433	0.053061
	2	0.017482	0.047413	0.018460	0.031686	0.064811	-0.154192	0.037163
	3	0.041560	0.033828	0.047104	0.021488	0.054780	0.028466	0.068791
	4	0.027537	0.042034	0.035324	0.036565	0.102885	0.067150	0.044383
	5	-0.083413	-0.099082	-0.105834	-0.068212	-0.106165	-0.080401	-0.004907
	6	0.005226	0.021495	0.008481	0.011649	0.037649	0.034536	0.019761
	7	0.015961	0.026903	0.050552	0.013089	-0.007142	0.014046	0.085243
	8	-0.063148	-0.052975	-0.068029	-0.041685	-0.043368	-0.079206	-0.071352
	9	-0.010628	-0.019575	0.017828	-0.000295	-0.015099	-0.037422	0.003235
	10	0.005595	0.054353	0.071505	0.085685	0.090341	0.076064	0.066059
	11	-0.006868	-0.042526	-0.014050	-0.027729	-0.017778	-0.080845	0.028851
	12	0.009065	0.000362	-0.012225	-0.000284	-0.009517	0.040223	-0.025247

Code Explanation:

- Define two functions to calculate the portfolio mean and standard deviation separately.

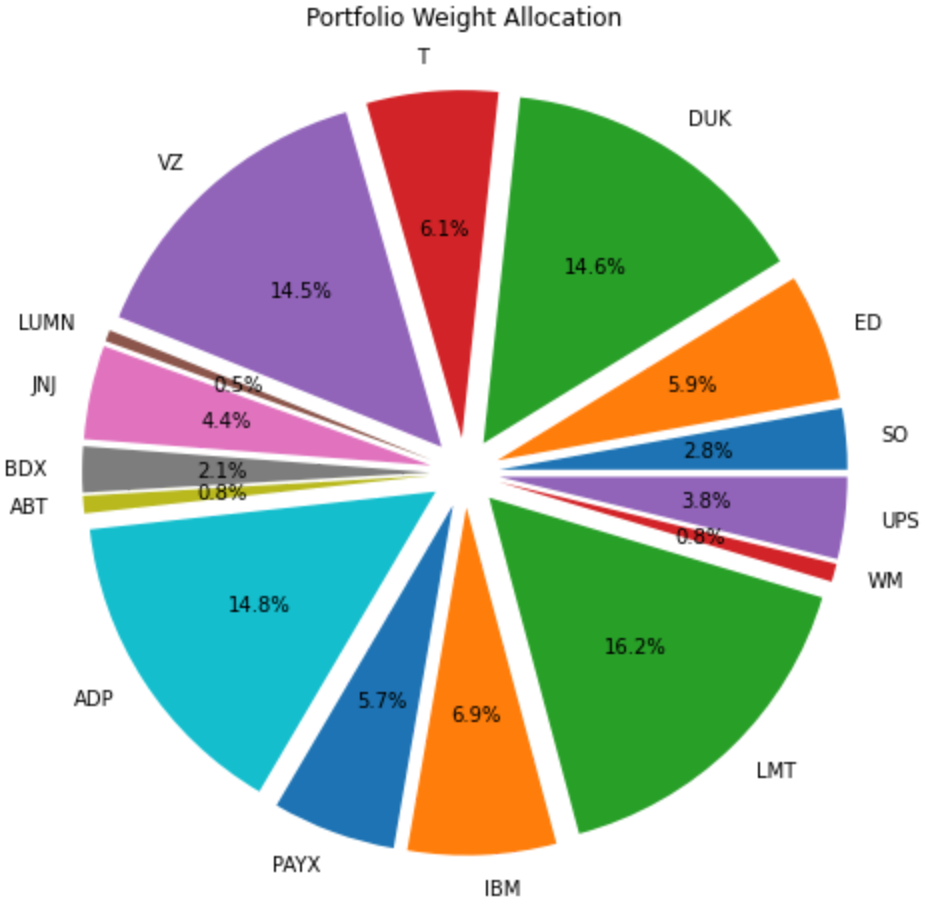
In [9]:
```
# Define the function to calculate the mean and standard deviation separately
def total_return(weights, calculated_stocks):
    return np.sum((weights * calculated_stocks.mean()) * 12)

def total_vol(weights, calculated_stocks):
    return math.sqrt(np.dot(weights.T, np.dot(calculated_stocks.cov() * 12, we
```

Code Explanation:

- Perform 100,000 simulations on 15 stocks to find the weight of each stock that the portfolio has **the highest Sharpe Ratio**:
  - Generate 15 weights for 15 stocks randomly.
  - Calculate the expected return and standard deviation of the portfolio.
  - Calculate **the Sharpe ratio of each stock** and find **the highest portfolio**.
  - Plot the pie charts.
  - Print out the final result.

In [10]:
```
best_weight = []
best_sharpe = 0
stock_return = []
stock_vol = []

for j in range(0,100000):
    # Generate 15 weights for 15 stocks randomly
    weight = np.random.random(stocks_number)
```

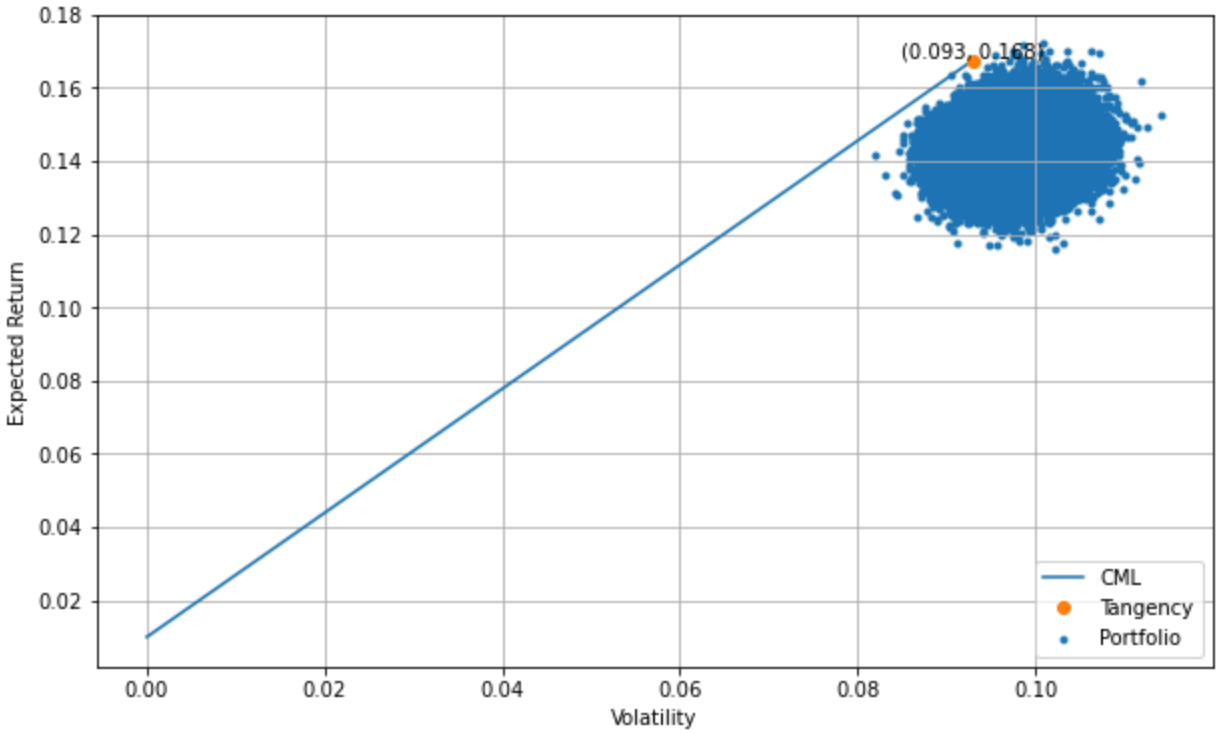
```
    weight /= np.sum(weight)

    # Calculate the expected return and standard deviation of the portfolio
    stock_return.append(total_return(weight, deal_stocks))
    stock_vol.append(total_vol(weight, deal_stocks))

    # Calculate the sharpe ratio of each stock and find the highest portfolio
    sharpe_ratio = (total_return(weight, deal_stocks) - rfrate) / total_vol(we
    if sharpe_ratio >= best_sharpe:
        best_sharpe = sharpe_ratio
        best_weight = weight

best_return = total_return(best_weight, deal_stocks)
best_vol = total_vol(best_weight, deal_stocks)
best_sharpe_ratio = (best_return - rfrate) / best_vol

# Plot the pie charts
plt.rcParams['figure.figsize'] = [12,8]
plt.pie(best_weight, explode = np.ones(15)*0.1, labels = chosen_stock, autopct
plt.title('Portfolio Weight Allocation')
plt.show()

# Print out the final result
print(f'The tangency portfolio invests {best_weight[0]:.1%} into {chosen_stock
```




```
The tangency portfolio invests 2.8% into SO, 5.9% into ED, 14.6% into DUK, 6.
1% into T, 14.5% into VZ, 0.5% into LUMN, 4.4% into JNJ, 2.1% into BDX, 0.8% i
nto ABT, 14.8% into ADP, 5.7% into PAYX, 6.9% into IBM, 16.2% into LMT, 0.8% i
nto WM, and 3.8% into UPS achieving a volatility 9.30% and expected return of
16.75% for a Sharpe Ratio of 1.694.
```

In [11]:
```
result_shown = pd.DataFrame([[best_return, best_vol, best_sharpe_ratio]],columns
result_shown.index = ['Result']
display(result_shown)

plt.rcParams['figure.figsize'] = [10,6]
plt.scatter(stock_vol , stock_return, marker = '.', label='Portfolio')
plt.plot(np.array([0, best_vol]), np.array([rfrate, best_return]), label='CML'
plt.plot(best_vol,best_return,'o',label='Tangency')
plt.text(best_vol,best_return,(round(best_vol,3),round(best_return,3)), ha = '
plt.grid('on')
plt.legend(loc='lower right')
plt.xlabel('Volatility')
plt.ylabel('Expected Return')
plt.show()
```

	Return	Volatility	Sharpe Ratio
Result	0.167513	0.093017	1.693908



3. Rebalance Plan & Back Test Strategy

In this section, we completed the backtesting of our investment strategy:

> **1.** In the first two sections, we have selected the sectors and stocks to invest in and obtained the weights of each stock in the optimal portfolio. We proceeded with the same weights for January to December of 2014 and calculated monthly profits or losses. Our assumptions are as follows: we **invest 100,000 dollars per month**, allocating the investment size of 10 stocks according to their weights of them. We assume we will **purchase stocks at the beginning of the month and sell them at the end**. The

> difference in the total value of the stocks at the time of sale with the initial capital of 100,000 dollars is our profit or loss.
>
> **2.** For each year after that until the end of 2021, we **first repeated the previous two sections**, re-performing the selection of sectors and stocks and assigning selected stocks' weights. Then we repeated the operations of each month in 2014 as above, calculating the monthly profits and losses.
>
> **3.** We added up the profits or losses for each stock for twelve months of one year to get the annualized data. As shown in the figure, we finally calculated the returns for **the eight years from 2014 to 2021 as 23.13%, 0.45%, 18.30%, 12.85%, 4.30%, 32.43%, 6.15%, and 21.49%**.

Code Explanation:

- **Backtest gains and losses from 2014-2021** using trading strategies developed from 2010-2013:
  - Use loops to obtain information about the year and download stock data for the set time
  - From January 2014 to December 2021, calculate the monthly losses and gains and add the final results to total_output
  - Obtain information on the development of sectors in **the last three years and reselect the five most stable sectors**
  - Download the stocks corresponding to the most stable sectors and reselect the three stocks with **the highest Sharpe Ratio** from each sector
  - Use the function to re-simulate the release of weight and find the combination with **the largest Sharpe ratio**.
  - Record sector selection, stock selection, and weight allocation for each year from 2014 to 2021

In [12]:
```
total_output = []
weight_2014to2021 = []
stock_2014to2021 = []
indus_2014to2021 = []

# Use loops to obtain information about the year and download stock data for t
for j in np.arange(2014,2022,1):
    years = [j, j]
    months = [1, 12]
    days = [1, 31]
    date = pd.to_datetime({'years': years, 'month': months, 'day': days})
    used_stocks = yf.download(chosen_stock, start = date[0], end = date[1])['A

    # From January 2014 to December 2021, calculate the monthly losses and gai
    for k in np.arange(1,13):
        used_stocks['month'] = used_stocks.index.month
        used_data = used_stocks.loc[used_stocks['month'] == k]
        used_data.reset_index(inplace = True)
        used_data.drop(columns = ['month','Date'], inplace = True)
```

```
        sum_output = np.sum(best_weight * 100000 / np.array(used_data[0:1])[0]
        total_output.append(round(sum_output,2))

    # Obtain information on the development of sectors in the last three years
    year = [j-2, j]
    date = pd.to_datetime({'years': years, 'month': months, 'day': days})

    loc_start = '{}01'.format(year[0])
    loc_end =  '{}31'.format(year[1])
    most_stable_indus = get_stable_sectors(loc_start, loc_end)

    # Download the stocks corresponding to the most stable sectors and reselec
    chosen_stock = []
    choose_stocks(most_stable_indus, chosen_stock)
    chosen_stock_return = get_stock_return(chosen_stock, date[0], date[1])

    # Use the function to re-simulate the release of weight and find the combi
    new_sharpe_ratio = 0
    for l in range(0,100000):
        weight = np.random.random(stocks_number)
        weight /= np.sum(weight)

        sharpe_ratio = (total_return(weight, chosen_stock_return) - rfrate) / 
        if sharpe_ratio >= new_sharpe_ratio:
            new_sharpe_ratio = sharpe_ratio
            best_weight = weight

    # Record sector selection, stock selection and weight allocation for each 
    weight_2014to2021.append(best_weight)
    stock_2014to2021.append(chosen_stock)
    indus_2014to2021.append(most_stable_indus)
```

```
[**********************100%**********************]  15 of 15 completed
```

```
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********100%**********]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  70 of 70 completed

5 Failed downloads:
- HWM: No data found for this date range, symbol may be delisted
- OTIS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CARR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FTV: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- IR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  28 of 28 completed

3 Failed downloads:
- WRK: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTVA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- DOW: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  15 of 15 completed
[**********100%**********]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********100%**********]  76 of 76 completed

8 Failed downloads:
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  70 of 70 completed

5 Failed downloads:
- HWM: No data found for this date range, symbol may be delisted
- OTIS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CARR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FTV: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- IR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  15 of 15 completed
[**********100%**********]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********100%**********]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  70 of 70 completed

5 Failed downloads:
- HWM: No data found for this date range, symbol may be delisted
- OTIS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CARR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FTV: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- IR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  15 of 15 completed
[**********100%**********]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********100%**********]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  70 of 70 completed

5 Failed downloads:
- HWM: No data found for this date range, symbol may be delisted
- OTIS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CARR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FTV: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- IR: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  15 of 15 completed
[**********100%**********]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********100%**********]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  28 of 28 completed

3 Failed downloads:
- WRK: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTVA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- DOW: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********100%**********]  15 of 15 completed
[**********100%**********]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********************100%**********************]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  67 of 67 completed

3 Failed downloads:
- BRK.B: No data found for this date range, symbol may be delisted
- CFG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SYF: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  15 of 15 completed
[**********************100%**********************]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********************100%**********************]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  67 of 67 completed

3 Failed downloads:
- BRK.B: No data found for this date range, symbol may be delisted
- CFG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SYF: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  15 of 15 completed
[**********************100%**********************]  15 of 15 completed
D:\Anaconda3-2021.11-Windows-x86_64\Anaconda\lib\site-packages\pandas\core\fra
me.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
[**********************100%**********************]  30 of 30 completed

1 Failed download:
- CEG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  25 of 25 completed

2 Failed downloads:
- FOX: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- FOXA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  64 of 64 completed

3 Failed downloads:
- OGN: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- MRNA: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- CTLT: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  76 of 76 completed

8 Failed downloads:
- CDAY: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- HPE: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PAYC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- QRVO: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- ANET: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- SEDG: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- KEYS: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- PYPL: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  33 of 33 completed

3 Failed downloads:
- BF.B: No data found for this date range, symbol may be delisted
- KHC: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
- LW: Data doesn't exist for startDate = 1262322000, endDate = 1388466000
[**********************100%**********************]  15 of 15 completed
```

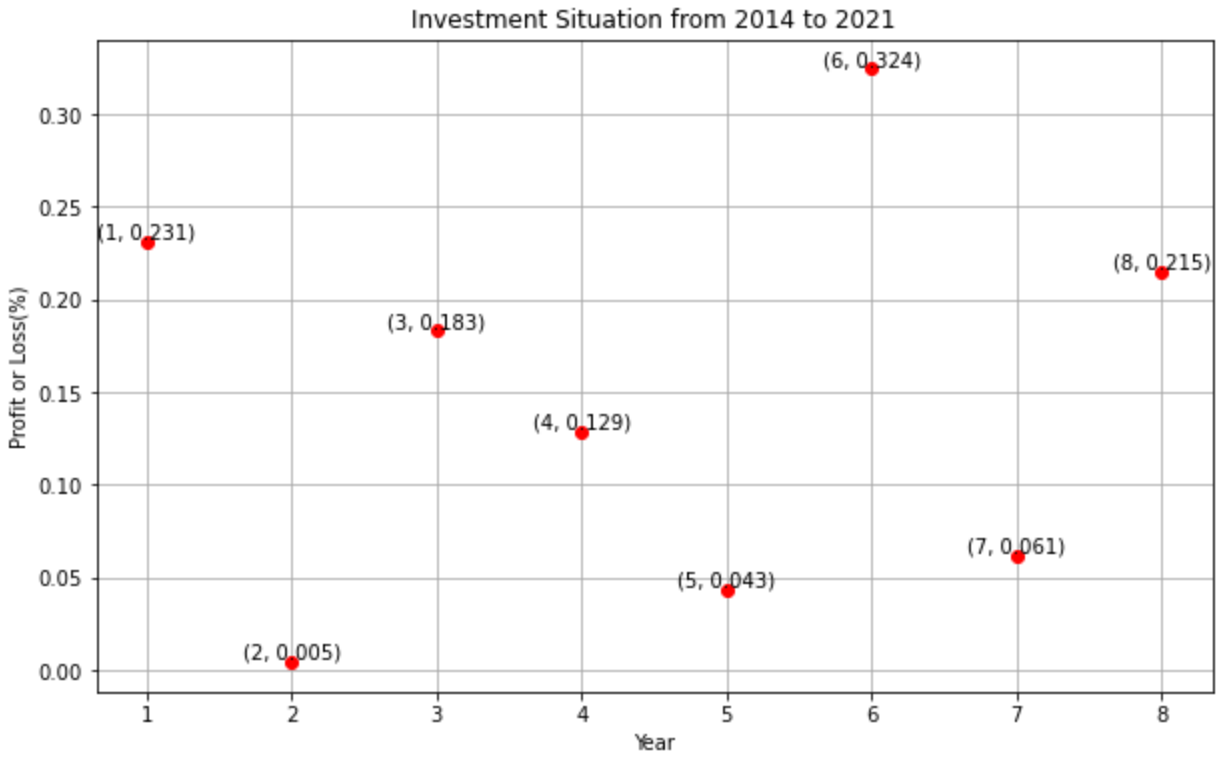
Code Explanation:

- Calculation of gains and losses on an annual basis.

In [13]:
```
profit_or_loss = np.array(total_output)/100000
yearly_situation = []

for i in range(0, len(profit_or_loss), 12):
    yearly_situation.append(np.sum(profit_or_loss[i:i+12])-12)

print(f'The return of 1st year is {yearly_situation[0]:.2%}.')
print(f'The return of 2nd year is {yearly_situation[1]:.2%}.')
print(f'The return of 3rd year is {yearly_situation[2]:.2%}.')
print(f'The return of 4th year is {yearly_situation[3]:.2%}.')
print(f'The return of 5th year is {yearly_situation[4]:.2%}.')
print(f'The return of 6th year is {yearly_situation[5]:.2%}.')
print(f'The return of 7th year is {yearly_situation[6]:.2%}.')
print(f'The return of 8th year is {yearly_situation[7]:.2%}.')
```

```
The return of 1st year is 23.13%.
The return of 2nd year is 0.45%.
The return of 3rd year is 18.30%.
The return of 4th year is 12.85%.
The return of 5th year is 4.30%.
The return of 6th year is 32.43%.
The return of 7th year is 6.15%.
The return of 8th year is 21.49%.
```

In [14]:
```
plt.plot(np.arange(1,9), yearly_situation,'ro')
for i,j in zip(np.arange(1,9), yearly_situation):
    plt.text(i, j, (i,round(j,3)), ha = 'center', va = 'bottom', fontsize = 10
plt.title('Investment Situation from 2014 to 2021')
plt.xlabel('Year')
plt.ylabel('Profit or Loss(%)')
plt.grid('on')
plt.show()
```



4. Simulation

In this section, we completed the simulation of our investment strategy:

> **1.** We collated **the monthly historical data** and investment weight allocation results of the selected stocks of backtest, read the data **from 2017 to 2021** into the notebook, transformed the simple return of each stock into log return for each month, calculated returns' mean and standard deviation, and the covariance between stocks' mean return. Displayed the data at the end of December 2021.
>
> **2.** We simulated the new price and return of each stock using the previously calculated mean and covariance of each stock and the year-end data of 2021.

> **3. 10,000 simulations calculate** the cumulative returns of each stock, and the histograms of the cumulative returns and losses of the portfolio for twelve months with and without stop loss levels set are plotted separately. As shown in the figure, **the expected cumulative log return (12 months) without a stop loss is 23.14%, and the expected cumulative log return with a stop loss of -15.00% is 21.72**.

Code Explanation:

- Adjust the data form:
  - Adjust the form of the Dataframe and put the weight of the matching year. (Data Source: https://wrds-www.wharton.upenn.edu/)
  - Sort and reset the index
  - Calculate log returns from the simple returns.

In [15]:
```
stockdata = pd.read_excel('fifteen_choosen_stocks.xlsx',parse_dates=['date'])

# Adjust the form of the dataframe and put the weight of the matching year.
stockdata['year'] = stockdata['date'].dt.year
stockdata.sort_values(['TICKER','year'],inplace=True)
stockdata.set_index(['TICKER','year'],inplace=True)
stockdata['weight'] = 1

for i in np.arange(2,8):
    for j in np.arange(0,15):
        stockdata.loc[(stock_2014to2021[i][j], 'weight')] = weight_2014to2021[

# Sort and reset the index
stockdata.reset_index(inplace=True)
stockdata.drop(columns = ['year'])
stockdata.sort_values(['TICKER','date'],inplace=True)
stockdata.set_index(['TICKER','date'],inplace=True)

# Calculate log returns from the simple returns
stockdata['log_ret'] = np.log(1+stockdata['RET'])
display(stockdata)
```

TICKER	date	year	PRC	RET	weight	log_ret
ABT	2016-01-29	2016	37.85	-0.151414	0.034108	-0.164184
	2016-02-29	2016	38.74	0.023514	0.034108	0.023242
	2016-03-31	2016	41.83	0.079763	0.034108	0.076742
	2016-04-29	2016	38.90	-0.063830	0.034108	-0.065958
	2016-05-31	2016	39.63	0.018766	0.034108	0.018592
...	...	...	...	...	...	...
WRB	2020-08-31	2020	62.05	0.004858	0.013944	0.004846
	2020-09-30	2020	61.15	-0.012570	0.013944	-0.012650
	2020-10-30	2020	60.12	-0.016844	0.013944	-0.016987
	2020-11-30	2020	65.13	0.083333	0.013944	0.080042
	2020-12-31	2020	66.42	0.021649	0.013944	0.021418

1080 rows × 5 columns

Code Explanation:

- Calculate means and standard deviation of stock returns and build correlation matrix:
  - Get a DataFrame that is only the log return column
  - Unstack into a wide matrix, so each column is one TICKER
  - Calculate the covariance matrix
  - Get the last entry for each stock used to the start point for our simulation

In [16]:
```
# Get a DataFrame that is only the log return column
rets_long = stockdata[['log_ret']]
sum_stats = rets_long[['log_ret']].groupby('TICKER').agg(['mean','std'])
print()
print('******Summary Stats for monthly log returns(%):********')
display(100*sum_stats)

# Unstack into a wide matrix so each column is one TICKER
rets_wide = stockdata['log_ret'].unstack('TICKER')

# Calculate the covariance matrix
cov_mat = rets_wide.cov()
cov_mat = cov_mat.fillna(0)
print()
print('****************Covariance matrix:****************')
display(cov_mat)

# Get the last entry for each stock uesd to start point for our simulation
last_stock_data = stockdata.groupby("TICKER").last()
last_stock_data['time'] = 0
last_stock_data.reset_index(inplace=True)
last_stock_data.set_index(['time','TICKER'],inplace=True)
last_stock_data = last_stock_data[['PRC','RET','weight']]
```

```
print('****************Last Date Data:****************')
display(last_stock_data)
```

******Summary Stats for monthly log returns(%):********

	log_ret	
	mean	std
TICKER		
ABT	1.741478	5.774377
ACGL	1.250199	10.360056
ADP	1.662842	5.586118
BALL	-1.404164	10.252809
BDX	0.795880	5.834388
DUK	0.888242	4.828171
ECL	0.877189	4.741696
ED	0.703716	5.048302
IBM	0.390621	7.160256
JNJ	0.534144	4.616698
KMB	0.761969	4.634531
L	0.002041	8.569244
LIN	-1.035865	5.569461
LMT	1.852657	3.358458
LUMN	-0.190274	9.581359
PAYX	1.571092	5.855202
PEP	1.548951	5.669533
PG	1.555245	5.463064
SO	0.911734	5.186719
T	0.036536	5.475443
UPS	1.136278	3.516288
VZ	0.931984	4.701235
WM	2.200509	2.996059
WRB	1.381866	7.065112

****************Covariance matrix:****************

TICKER	ABT	ACGL	ADP	BALL	BDX	DUK	ECL	ED
TICKER								
ABT	0.003334	0.000038	0.000596	0.003290	0.001680	0.000635	0.001039	0.000540
ACGL	0.000038	0.010733	0.002429	0.000000	0.001553	0.002199	0.000000	0.000653
ADP	0.000596	0.002429	0.003120	0.004370	0.000646	0.000544	0.001990	0.000566
BALL	0.003290	0.000000	0.004370	0.010512	0.006184	0.001360	0.003537	0.002225
BDX	0.001680	0.001553	0.000646	0.006184	0.003404	0.000938	0.002662	0.001101
DUK	0.000635	0.002199	0.000544	0.001360	0.000938	0.002331	0.000690	0.001962
ECL	0.001039	0.000000	0.001990	0.003537	0.002662	0.000690	0.002248	0.001148
ED	0.000540	0.000653	0.000566	0.002225	0.001101	0.001962	0.001148	0.002549
IBM	0.001940	0.003583	0.001383	0.004981	0.002101	0.000820	0.002102	0.000745
JNJ	0.000563	0.000225	0.001000	0.003761	0.000824	0.000981	0.001652	0.001126
KMB	0.000822	0.000000	0.001009	0.000000	0.000918	0.001685	0.000000	0.001744
L	0.001052	0.006094	0.002594	0.000000	0.002360	0.001570	0.000000	0.000820
LIN	0.001641	0.000000	0.000898	0.002954	0.002064	0.000461	0.000386	0.000512
LMT	0.000445	0.000000	0.000074	0.000000	0.000329	0.000067	0.000000	-0.000043
LUMN	0.001292	0.002855	0.001437	0.004534	0.000896	0.000443	0.002870	0.000253
PAYX	0.001128	0.004159	0.002381	0.003743	0.000933	0.000876	0.001311	0.000672
PEP	0.000749	0.000000	0.002284	0.000000	0.000812	0.002612	0.000000	0.002589
PG	0.000682	0.000000	0.001999	0.000000	0.000915	0.002231	0.000000	0.002548
SO	0.000690	0.003150	0.000836	0.001741	0.001115	0.001981	0.000555	0.001966
T	0.000826	0.003961	0.001086	0.001587	0.000979	0.000985	0.000985	0.000730
UPS	0.000341	0.000000	0.000315	0.000000	0.000144	-0.000152	0.000000	-0.000137
VZ	0.001432	0.000713	0.000773	0.003682	0.001365	0.001101	0.001644	0.001022
WM	0.000279	0.000000	0.000660	0.000000	-0.000093	0.000407	0.000000	0.000368
WRB	0.001375	0.005993	0.002113	0.000000	0.001871	0.002385	0.000000	0.001344

24 rows × 24 columns

```
**********Last Date Data:**********
```

time	TICKER	PRC	RET	weight
0	ABT	140.74001	0.119027	0.034108
	ACGL	36.07000	0.120360	0.012373
	ADP	246.58000	0.072459	0.186837
	BALL	97.53000	-0.154046	0.195171
	BDX	251.48000	0.064139	0.005179
	DUK	104.90000	0.081332	0.012076
	ECL	147.35001	-0.079008	0.005511
	ED	85.32000	0.098918	0.055206
	IBM	133.66000	0.141418	0.008973
	JNJ	51.96000	0.033618	0.000823
	KMB	142.92000	0.105518	0.114426
	L	45.02000	0.074207	0.004952
	LIN	26.72000	-0.066387	0.087932
	LMT	321.04999	0.006048	0.152054
	LUMN	12.55000	0.017018	0.026864
	PAYX	136.50000	0.145134	0.175231
	PEP	173.71001	0.093910	0.121907
	PG	163.58000	0.131415	0.173848
	SO	68.58000	0.122422	0.048768
	T	24.60000	0.077530	0.025007
	UPS	119.15000	-0.018938	0.171677
	VZ	171.07001	0.097095	0.010747
	WM	86.30000	0.049240	0.126254
	WRB	66.42000	0.021649	0.013944

Code Explanation:

- **Define two functions** to generate CRSP-style data with a (time, TICKER) MultiIndex:
  - **Generate num_periods *simple* returns for each stock in the sum_stats/cov_mat dataframes**:
    - Simulate log returns
    - Unlock them into the simple returns and put them into a nice DataFrame, setting the time index to 1:num_periods
  - **Generates CRSP-style data with a (time, TICKER) MultiIndex**:
    - Call the function above to generate num_periods simple returns for each stock.
    - Calculate the simulated prices from the simulated returns

- Generates the prices at period 1:num_periods, starting from the last prices in the data
- Take the wide RET and PRC data, and stack them into a MultiIndex with time and TICKER
- Merge the RET, PRC, and weight columns for 1:num_periods
- Concatenate the last observation from each stock with our simulated data

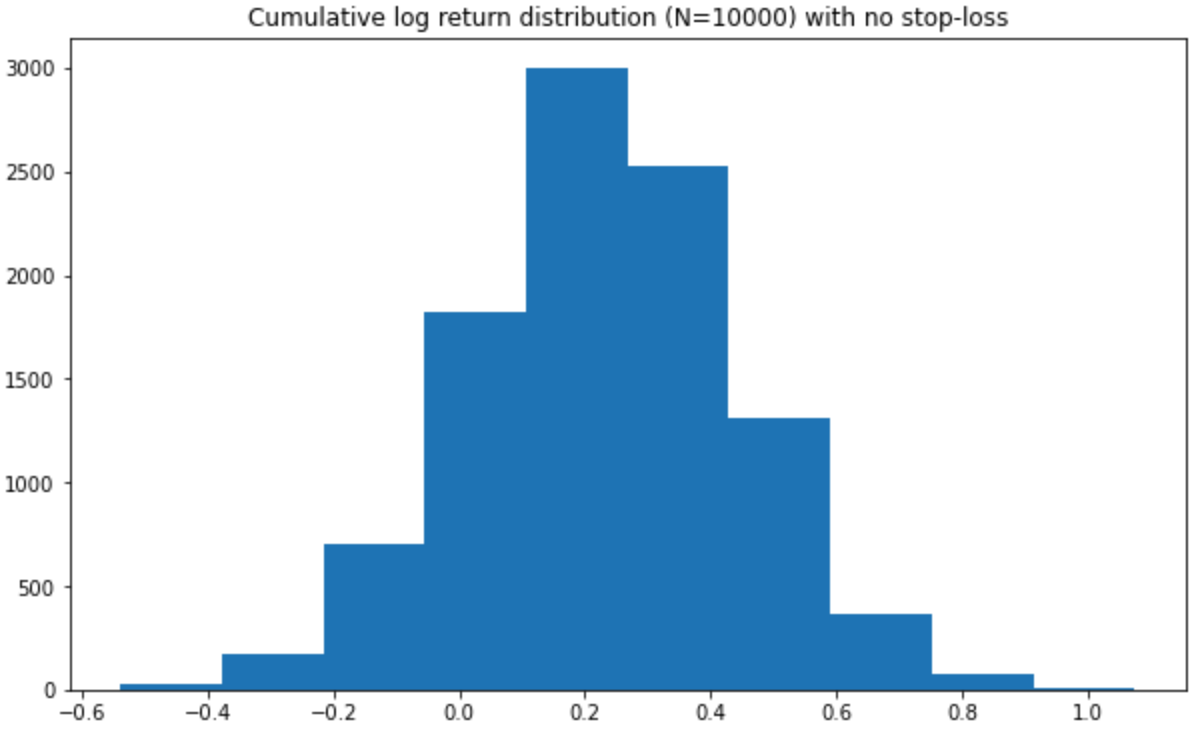
In [17]:
```
rng = np.random.default_rng()

# Generate num_periods *simple* returns for each stock in the sum_stats/cov_ma
def sim_period_returns(num_periods):
    # Simulate log returns
    sim_log_rets = rng.multivariate_normal(sum_stats[('log_ret','mean')],cov_m
    # Unlog them into the simple returns and put into a nice DataFrame, setting
    sim_simple_rets = pd.DataFrame(np.exp(sim_log_rets)-1,columns=cov_mat.colu
                                                    index=pd.Index(np.arange(1,nu

    return sim_simple_rets

# Generates CRSP-style data, with a (time,TICKER) MultiIndex
def gen_sim_data(num_periods):
    # Call the function above to generate num_periods simple returns for each
    sim_simple_rets = sim_period_returns(num_periods)

    # Calculate the simulated prices from the simulated returns
    lsd = last_stock_data.reset_index()
    last_prices = lsd.pivot('time','TICKER','PRC')

    # Generates the prices at period 1:num_periods, starting from the last pri
    sim_prices = np.array(last_prices)*(1+sim_simple_rets).cumprod(axis=0)

    # Take the wide RET and PRC data, and stack them into a MultiIndex with ti
    sim_simple_rets = pd.DataFrame(sim_simple_rets.stack(),columns=["RET"])
    sim_prices = pd.DataFrame(sim_prices.stack(),columns=["PRC"])

    # Merge the RET, PRC, and weight columns for 1:num_periods
    all_sim_data = sim_prices.merge(sim_simple_rets,left_index=True,right_inde
    all_sim_data.reset_index(inplace=True)
    all_sim_data = all_sim_data.merge(last_stock_data[['weight']],left_on='TIC

    # Concatenate the last observation from each stock with our simulated data
    all_sim_data = pd.concat([lsd,all_sim_data],axis=0)
    all_sim_data.sort_values(['time','TICKER'],inplace=True)
    all_sim_data.set_index(['time','TICKER'],inplace=True)
    return all_sim_data
```

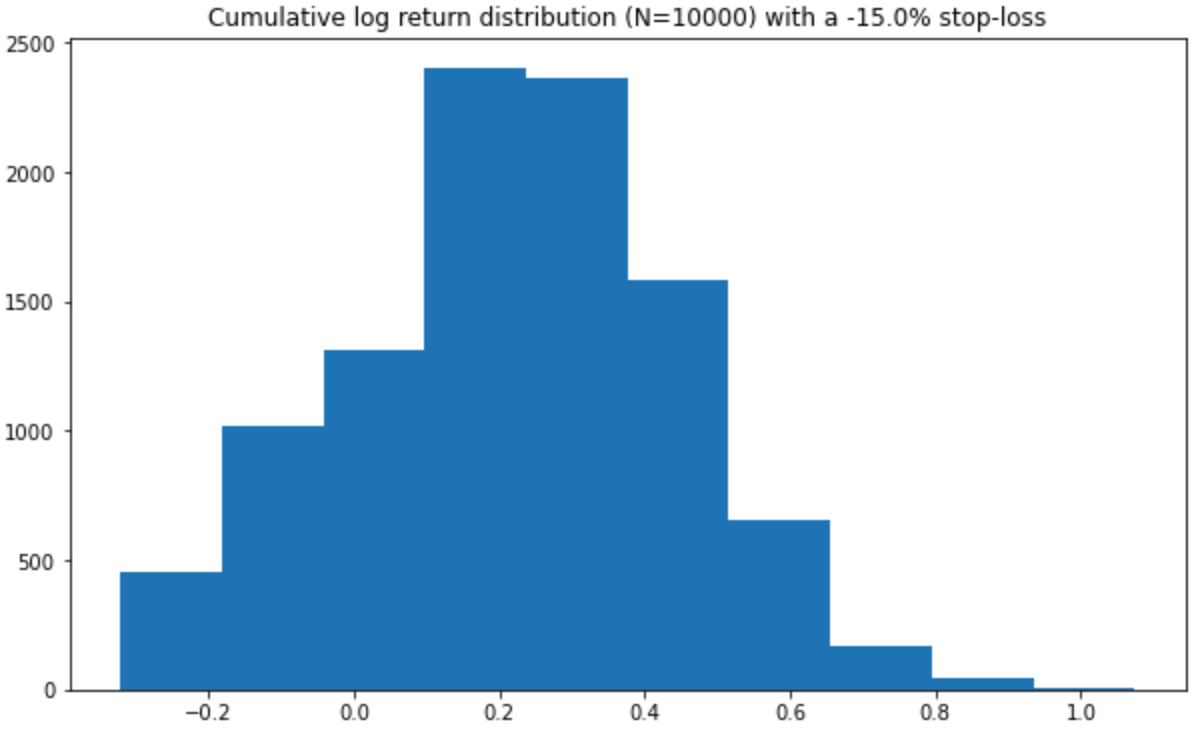
Code Explanation:

- Perform 10,000 simulations to find the gains and losses of the release:
  - Generate the simulated data and calculate weights
  - Store the last cumulative return (the 12-month return):
    - Hit the stop-loss level, so find the first one and store it
    - Didn't hit our loss level, so use the cumulative return at month 12
  - Plot histogram

In [18]:
```
stop_loss = -0.15
num_periods = 12
N_sims = 10_000
all_cum_log_returns = np.zeros(N_sims)
cum_log_returns_stop = np.zeros(N_sims)
for sim_num in np.arange(N_sims):
    # Generate the simulated data
    sim_data = gen_sim_data(num_periods)
    sim_data.drop(0,inplace=True)

    # Calculate weights
    sim_data['weight_ret'] = sim_data.RET * sim_data.weight
    port_log_returns = np.log(1+sim_data.groupby('time')['weight_ret'].sum())
    cum_log_returns = port_log_returns.cumsum()

    # Store the last cumulative return (the 12-month return)
    all_cum_log_returns[sim_num] = cum_log_returns[cum_log_returns.size]

    if cum_log_returns.lt(stop_loss).any():
        # Hit the stop-loss level, so find the first one and store it
        cum_log_returns_stop[sim_num] = cum_log_returns[cum_log_returns.lt(stop
    else:
        # Didn't hit our loss level, so use the cumulative return at month 12
        cum_log_returns_stop[sim_num] = cum_log_returns[cum_log_returns.size]

# Plot histogram
plt.figure()
plt.hist(all_cum_log_returns)
plt.title(f'Cumulative log return distribution (N={N_sims}) with no stop-loss'
plt.show()

plt.figure()
plt.hist(cum_log_returns_stop)
plt.title(f'Cumulative log return distribution (N={N_sims}) with a {stop_loss:
plt.show()

print(f'Expected cumulative log return ({num_periods} months) without stop loss
print(f'Expected cumulative log return with stop loss of {stop_loss:.2%} = {cu
```

```
C:\Users\PC\AppData\Local\Temp/ipykernel_12224/1343334264.py:6: RuntimeWarnin
g: covariance is not positive-semidefinite.
  sim_log_rets = rng.multivariate_normal(sum_stats[('log_ret','mean')],cov_ma
t,size=num_periods)
```






```
Expected cumulative log return (12 months) without stop loss = 23.14%
Expected cumulative log return with stop loss of -15.00% = 21.72%
```

Code Explanation:

- Use the dataframe form to convert the final result into the form of Percentiles

In [19]:
```
percentiles = np.array([1,5,10,1/3*100,50,2/3*100,90,95,99])/100
all_pct = np.quantile(all_cum_log_returns,percentiles)
stop_pc = np.quantile(cum_log_returns_stop,percentiles)
print("Percentiles:")
pct_comparison = pd.DataFrame([all_pct, stop_pc],columns=np.round(100*percenti
```

```
pct_comparison.index=['No Stop Loss','Stop Loss 15%']
display(pct_comparison*100)
```

Percentiles:

	1.0	5.0	10.0	33.3	50.0	66.7	90.0	95
No Stop Loss	-27.286447	-11.512688	-4.263802	14.143395	23.194923	32.086584	50.014638	58.1969
Stop Loss 15%	-22.348155	-17.887134	-15.446615	13.237175	22.764171	31.925002	49.990970	58.1920

5. Evaluate the Strategy

Code Explanation:

- Display the result of chosen industry, stock, and weight.

In [20]:
```
print('**********Chosen Industries from 2014 to 2021:**********')
year_number = ['1st Year','2nd Year','3rd Year','4th Year','5th Year','6th Yea
indus_shown = pd.DataFrame(indus_2014to2021, columns = ['1st Stable', '2nd Stal
indus_shown.index = year_number
display(indus_shown)

print()
print('**********Chosen Stocks and Their Weights from 2014 to 2021:******
stock_shown = pd.DataFrame(stock_2014to2021)
stock_shown.index = year_number

weight_shown = pd.DataFrame(weight_2014to2021)
weight_shown.index = year_number

stock_and_weight = pd.DataFrame()
for i in range(0,15):
    stock_and_weight = pd.concat([stock_and_weight, stock_shown[i]],axis = 1)
    stock_and_weight = pd.concat([stock_and_weight, weight_shown[i]],axis = 1)

display(stock_and_weight)
```

```
**********Chosen Industries from 2014 to 2021:**********
```

	1st Stable	2nd Stable	3rd Stable	4th Stable	5th Stable
1st Year	Communication Services	Industrials	Information Technology	Utilities	Materials
2nd Year	Information Technology	Utilities	Communication Services	Industrials	Health Care
3rd Year	Utilities	Communication Services	Health Care	Information Technology	Industrials
4th Year	Utilities	Health Care	Communication Services	Information Technology	Industrials
5th Year	Utilities	Communication Services	Information Technology	Health Care	Materials
6th Year	Utilities	Communication Services	Financials	Information Technology	Health Care
7th Year	Utilities	Communication Services	Information Technology	Health Care	Financials
8th Year	Utilities	Communication Services	Health Care	Information Technology	Consumer Staples

**********Chosen Stocks and Their Weights from 2014 to 2021:**********

	0	0	1	1	2	2	3	3	4	4	...	
1st Year	T	0.067544	VZ	0.007691	LUMN	0.117670	LMT	0.033973	WM	0.139352	...	
2nd Year	ADP	0.009036	PAYX	0.254144	IBM	0.140182	SO	0.040641	ED	0.067353	...	W
3rd Year	SO	0.070136	ED	0.005111	DUK	0.049411	T	0.024078	VZ	0.000022	...	PA
4th Year	SO	0.038594	ED	0.012111	DUK	0.029048	JNJ	0.146209	BDX	0.053730	...	PA
5th Year	SO	0.192734	ED	0.055391	DUK	0.059741	T	0.000695	VZ	0.001581	...	B
6th Year	SO	0.188386	ED	0.025144	DUK	0.035929	T	0.153435	VZ	0.013836	...	PA
7th Year	SO	0.009314	ED	0.013672	DUK	0.166926	T	0.012074	VZ	0.167910	...	B
8th Year	SO	0.048768	ED	0.055206	DUK	0.012076	T	0.025007	VZ	0.010747	...	PA

8 rows × 30 columns

Strength:

> **1.** we have a **clear stock selection strategy** that first ensures the stability of the investment and, on this basis, **selects stocks with higher returns**.
>
> **2.** To ensure the stability of our investment, we **combined data analysis and analysis of the general economic environment** when selecting sectors. We

> selected the least volatile sectors and the stocks with **the highest Sharpe ratio**, making our investment relatively stable in the sector and single stock selection.
>
> **3.** We conducted **100,000 weight assignment simulations** to find the optimal portfolio among them, which allows our portfolio to meet our investment requirements to a greater extent with less coincidence.

Weakness & Adjustment:

> **1. The 5 most stable sectors from 2010 to 2021 change little**, while the stocks with the highest sharpe ratios in each sector did not change either.
> **Adjustment: extend the rebalancing time**, for example, once every 3-5 years.
>
> **2.** The **maximum possible loss exceeds 20%**, indicating that our strategy is not stable as we imagined.
> **Adjustment: Select more sectors** and stocks to diversify the riSharpe>
>
> **3. The randomness of stocks corresponding to weight is large**; for example, the 15 stocks selected in January 2014 and December 2013 are the same, but the weight difference is large, which may impact the return on the investment.
> **Adjustment: Increase the number of simulations** and try to simulate the distribution of weights as much as possible.