Harry Markowitz founded the Modern Portfolio Theory (MPT), in which the most fundamental aspect is to **maximize returns** while **minimizing risks**.

MPT means that investors can increase their returns, while minimizing or having no additional risk, by investing in different asset classes instead of just one.

Having a combination of securities that lack correlation with each other, allows investors to increase or optimize their returns without increasing the risk of their portfolio.

Encourages **diversification**.

also known as **mean-variance analysis**.

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
plt.style.use('fivethirtyeight')
import datetime
```

The goal is to plot the **Efficient Frontier**:

Efficient frontier is a graph with ‘returns’ on the Y-axis and ‘volatility’ on the X-axis. It shows the set of optimal portfolios that offer the **highest expected return** **for a given risk level or the** ***lowest risk** for a given level of expected return.

Let us start by getting tickers from Wikipedia:

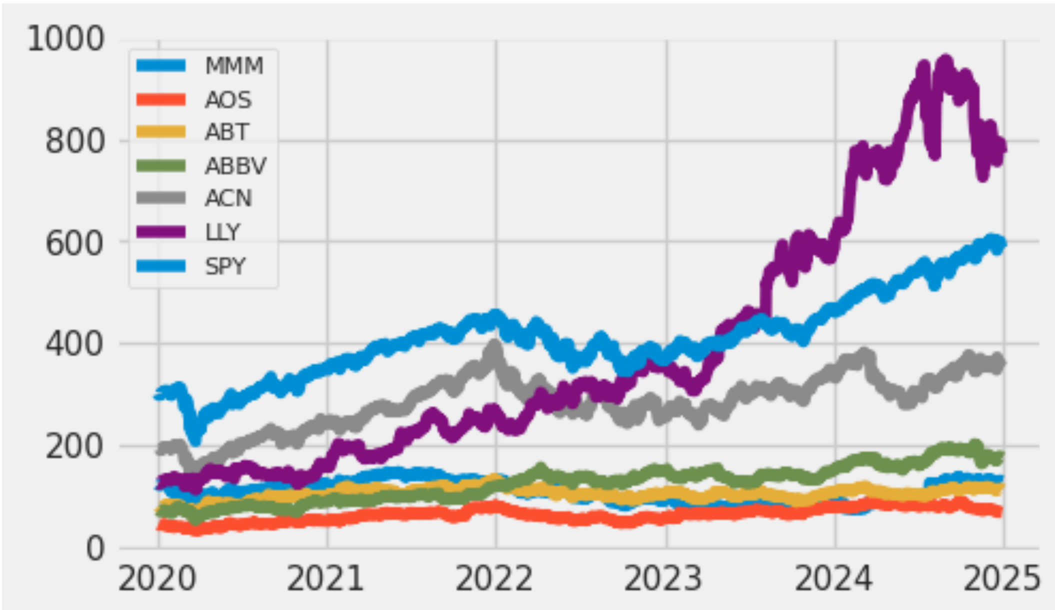
```
stockInfo = pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_compani
tickers_np = stockInfo['Symbol'].to_numpy()
```

```
tickers = ['MMM', 'AOS', 'ABT', 'ABBV', 'ACN','LLY','SPY']
for ticker in tickers:
    globals()[ticker] = yf.Ticker(ticker)
    globals()[ticker] = globals()[ticker].history(start = "2020-01-01", end= "202

for ticker in tickers:
  globals()[ticker] = globals()[ticker].Close

df = pd.DataFrame()
for ticker in tickers:
    df[ticker] = globals()[ticker]

df
```

Date	MMM	AOS	ABT	ABBV	ACN	LLY	S
2020-01-02 00:00:00-05:00	122.357445	43.422443	79.273933	71.589790	195.263611	123.702332	300.291
2020-01-03 00:00:00-05:00	121.303810	43.040657	78.307533	70.910255	194.938354	123.290604	298.017
2020-01-06 00:00:00-05:00	121.419373	43.313366	78.717789	71.469872	193.665421	123.749046	299.154
2020-01-07 00:00:00-05:00	120.929947	43.022495	78.280182	71.062141	189.484161	123.982964	298.313
2020-01-08 00:00:00-	122.785683	42.958862	78.599266	71.565788	189.855881	125.105797	299.903

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(5, 3))
plt.plot(df, label=df.columns)
plt.legend(loc='upper left',fontsize=8)
plt.show()
```



Log returns of the 7 assets:

```
data = np.log(df/df.shift(1))
data.iloc[0] = 0
data.head(5)
```

Date	MMM	AOS	ABT	ABBV	ACN	LLY	SPY
2020-01-02 00:00:00-05:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2020-01-03 00:00:00-05:00	-0.008648	-0.008831	-0.012266	-0.009537	-0.001667	-0.003334	-0.007601

Next steps: Generate code with data | View recommended plots | New interactive sheet

Now, create annualized covariance and correlation matrices.

Covariance measures the **directional relationship** between the returns on two assets.

```
data.mean()*250
cov_matrix = data.cov() * 250
cov_matrix
```

	MMM	AOS	ABT	ABBV	ACN	LLY	SPY
MMM	0.084735	0.044532	0.029603	0.022298	0.039522	0.019445	0.034763
AOS	0.044532	0.090048	0.027495	0.019606	0.041046	0.020330	0.036544
ABT	0.029603	0.027495	0.067746	0.028886	0.039750	0.030728	0.033485
ABBV	0.022298	0.019606	0.028886	0.061874	0.026031	0.031755	0.024058
ACN	0.039522	0.041046	0.039750	0.026031	0.085552	0.029804	0.047499
LLY	0.019445	0.020330	0.030728	0.031755	0.029804	0.102445	0.028939
SPY	0.034763	0.036544	0.033485	0.024058	0.047499	0.028939	0.044097

Next steps: Generate code with `cov_matrix` View recommended plots New interactive sheet

Correlation measures the **degree** to which two securities move in relation to each other.

```
corr_matrix =data.corr() * 250
corr_matrix
```

	MMM	AOS	ABT	ABBV	ACN	LLY	SP
MMM	250.000000	127.451220	97.678375	76.986711	116.045144	52.176159	142.17501
AOS	127.451220	250.000000	88.007044	65.666605	116.913028	52.917429	144.98259
ABT	97.678375	88.007044	250.000000	111.538735	130.531580	92.211078	153.15865
ABBV	76.986711	65.666605	111.538735	250.000000	89.446041	99.711763	115.14636
ACN	116.045144	116.913028	130.531580	89.446041	250.000000	79.588283	193.33471
LLY	52.176159	52.917429	92.211078	99.711763	79.588283	250.000000	107.64021
SPY	142.175011	144.982593	153.158653	115.146368	193.334713	107.640210	250.00000

Next steps: Generate code with corr_matrix | View recommended plots | New interactive sheet

```
portfolio1 = data[['MMM','LLY']]
portfolio1.corr() * 250
portfolio1.cov() * 250
# portfolio1
```

	MMM	LLY
MMM	0.084735	0.019445
LLY	0.019445	0.102445

Let us assign weights to each investment in the portfolio randomly, and find the variance of this portfolio.

```
w = {'MMM': 0.1, 'LLY': 0.2, 'AOS': 0.2, 'ABT': 0.1, 'ABBV': 0.2, 'ACN': 0.05, 'S
# sum of all elements in w
sum(w.values())

port_var = cov_matrix.mul(w, axis=0).mul(w, axis=1).sum().sum()
port_var
```

```
np.float64(0.03703096589673029)
```

To optimize the portfolio, we **cannot assign the weights**. We need exact weights that will maximize expected return for a given risk.

So, let us get the yearly returns for each company using the package resample.

```
ind_er = df.resample('Y').last().pct_change().mean()
ind_er
```

```
<ipython-input-11-fbdefa6d49e6>:1: FutureWarning: 'Y' is deprecated and will k
  ind_er = df.resample('Y').last().pct_change().mean()
```

	0
MMM	0.045165
AOS	0.144306
ABT	0.041581
ABBV	0.185137
ACN	0.153931
LLY	0.487153
SPY	0.155179

dtype: float64

Now, the portfolio returns: individual returns multiplied by weights in the portfolio.

```
weights = list(w.values())
port_er = (weights*ind_er).sum()
port_er
```

```
np.float64(0.13862813355218426)
```

Calculate the volatility, or the annualized standard deviation.

```
ann_sd = df.pct_change().apply(lambda x: np.log(1+x)).std().apply(lambda x: x*np.
ann_sd
```

	0
MMM	0.291209
AOS	0.300200
ABT	0.260385
ABBV	0.248844
ACN	0.292609
LLY	0.320197
SPY	0.210075

dtype: float64

```
data.std()*np.sqrt(250)
```

	0
MMM	0.291093
AOS	0.300080
ABT	0.260281
ABBV	0.248745
ACN	0.292493
LLY	0.320070
SPY	0.209992

dtype: float64

Create a table for returns and volatility of assets.

```
assets = pd.concat([ind_er, ann_sd], axis=1)
assets.columns = ['Returns', 'Volatility']
assets
```

	Returns	Volatility
MMM	0.045165	0.291209
AOS	0.144306	0.300200
ABT	0.041581	0.260385
ABBV	0.185137	0.248844
ACN	0.153931	0.292609
LLY	0.487153	0.320197
SPY	0.155179	0.210075

Next steps: Generate code with assets | View recommended plots | New interactive sheet

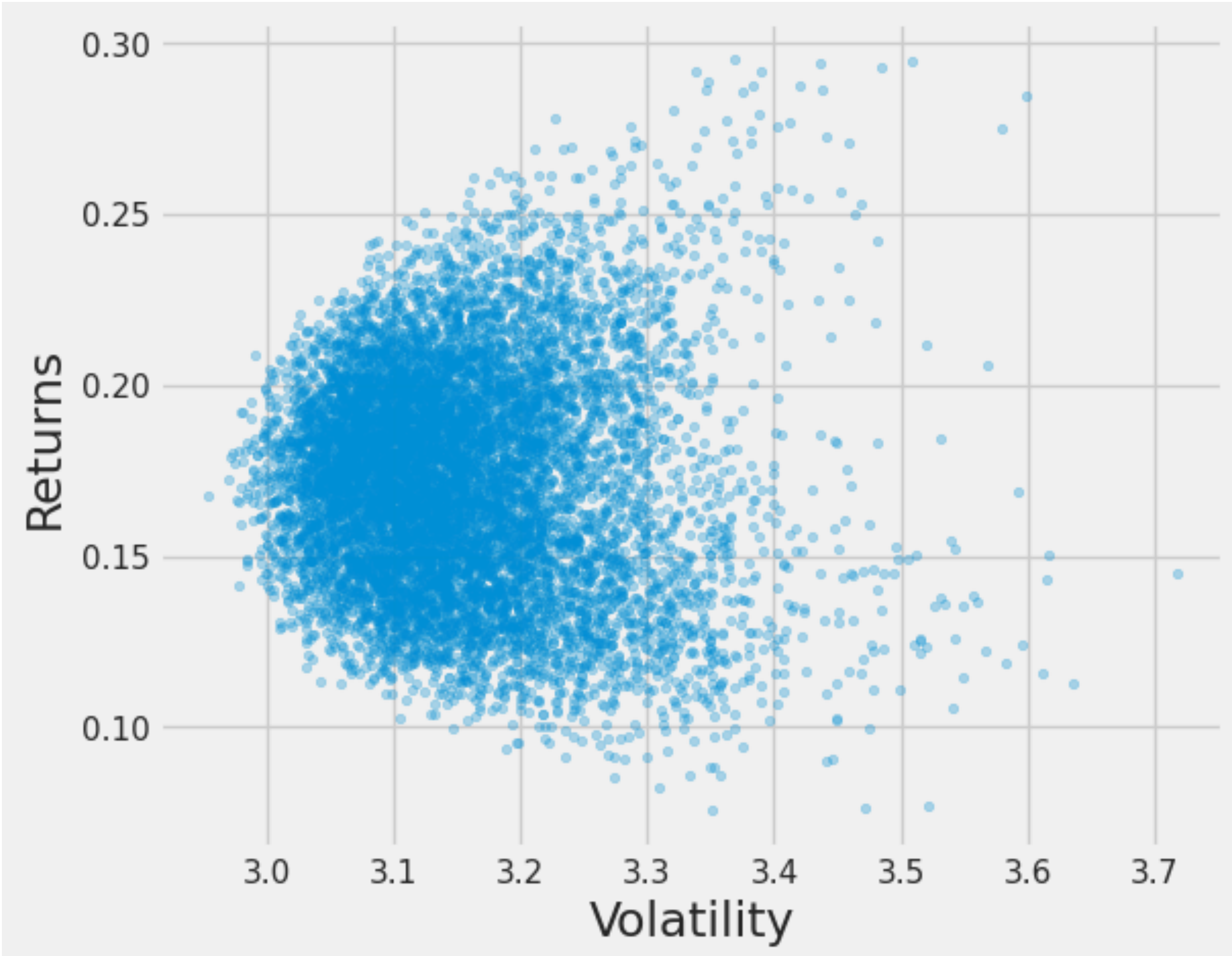
```
p_ret = []
p_vol = []
p_weights = [] # Define an empty array for asset weights

num_assets = len(df.columns)
num_portfolios = 10000


for portfolio in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights = weights/np.sum(weights)
    p_weights.append(weights)
    returns = np.dot(weights, ind_er) # Returns are the product of individual exp
                                      # weights
    p_ret.append(returns)
    var = cov_matrix.mul(weights, axis=0).mul(weights, axis=1).sum().sum()# Portf
    sd = np.sqrt(var) # Daily standard deviation
    ann_sd = sd*np.sqrt(250) # Annual standard deviation = volatility
    p_vol.append(ann_sd)


data = {'Returns':p_ret, 'Volatility':p_vol}

for counter, symbol in enumerate(df.columns.tolist()):
    #print(counter, symbol)
    data[symbol+' weight'] = [w[counter] for w in p_weights]


portfolios  = pd.DataFrame(data)
portfolios.head() # Dataframe of the 10000 portfolios created
```

	Returns	Volatility	MMM weight	AOS weight	ABT weight	ABBV weight	ACN weight	LLY weight	w
0	0.184120	2.990137	0.098255	0.085260	0.078278	0.278835	0.086509	0.124480	0.2
1	0.146693	3.084897	0.056530	0.184114	0.183245	0.195162	0.157993	0.044887	0.1
2	0.154251	3.041383	0.178461	0.069902	0.044149	0.205334	0.107312	0.055615	0.3
3	0.193989	3.201704	0.058319	0.169130	0.141449	0.058243	0.230657	0.185788	0.1
4	0.151988	3.346869	0.208677	0.007433	0.075381	0.182465	0.408299	0.070650	0.0

Next steps: Generate code with `portfolios` View recommended plots New interactive sheet

```
portfolios.plot.scatter(x='Volatility', y='Returns', marker='o', s=10, alpha=0.3,
```

```
<Axes: xlabel='Volatility', ylabel='Returns'>
```

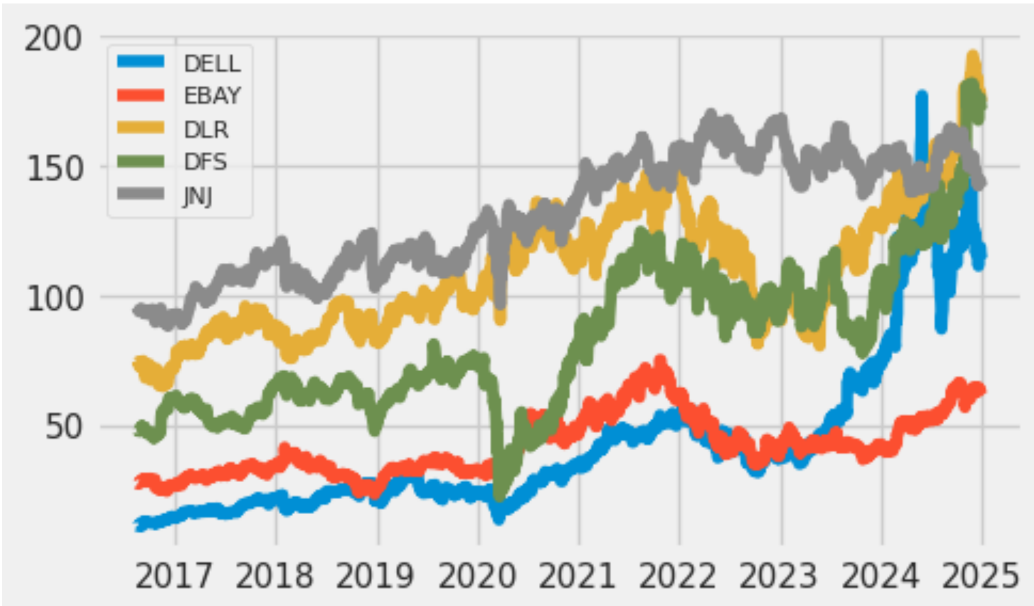


```
tickers = ['DELL', 'EBAY', 'DLR', 'DFS', 'JNJ']
for ticker in tickers:
    globals()[ticker] = yf.Ticker(ticker)
    globals()[ticker] = globals()[ticker].history(start = "2014-01-01", end= "202

for ticker in tickers:
  globals()[ticker] = globals()[ticker].Close

df = pd.DataFrame()
for ticker in tickers:
    df[ticker] = globals()[ticker]

df
```

	DELL	EBAY	DLR	DFS	JNJ
Date					
2016-08-17 00:00:00-04:00	11.218981	27.533337	74.847343	48.104401	95.384834
2016-08-18 00:00:00-04:00	11.153753	27.452387	74.435524	48.137520	94.920929
2016-08-19 00:00:00-04:00	11.349432	27.551329	74.288437	47.930466	94.920937
2016-08-22 00:00:00-04:00	11.388571	27.542337	74.722305	47.814514	94.295631
2016-08-23 00:00:00-04:00	11.740795	27.587313	73.280968	48.096123	94.541000
...	...	...	...	...	...
2024-12-23 00:00:00-05:00	118.346001	63.342800	176.978058	173.908981	144.116409
2024-12-24 00:00:00-05:00	118.465523	63.492119	178.891922	175.971863	144.691803

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(5, 3))
plt.plot(df, label=df.columns)
plt.legend(loc='upper left',fontsize=8)
plt.show()
```




```
data = np.log(df/df.shift(1))
data.iloc[0] = 0
data.head(5)
```

Date	DELL	EBAY	DLR	DFS	JNJ
2016-08-17 00:00:00-04:00	0.000000	0.000000	0.000000	0.000000	0.000000e+00
2016-08-18 00:00:00-04:00	-0.005831	-0.002944	-0.005517	0.000688	-4.875378e-03
2016-08-19 00:00:00-04:00	0.017392	0.003598	-0.001978	-0.004311	8.037631e-08
2016-08-22 00:00:00-04:00	0.003443	-0.000326	0.005823	-0.002422	-6.609437e-03
2016-08-23 00:00:00-04:00	0.030459	0.001632	-0.019478	0.005872	2.598745e-03

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.mean()*250
cov_matrix = data.cov() * 250
cov_matrix
```

	DELL	EBAY	DLR	DFS	JNJ
DELL	0.153337	0.033362	0.027001	0.062578	0.012910
EBAY	0.033362	0.089249	0.025446	0.039138	0.015230
DLR	0.027001	0.025446	0.081246	0.027634	0.016360
DFS	0.062578	0.039138	0.027634	0.169918	0.021194
JNJ	0.012910	0.015230	0.016360	0.021194	0.034150

Next steps: Generate code with cov_matrix | View recommended plots | New interactive sheet

```
ortfolio1 = data[['DELL','EBAY']]
portfolio1.corr() * 250
portfolio1.cov() * 250
# portfolio1
```

	MMM	LLY
MMM	0.084735	0.019445
LLY	0.019445	0.102445

```
w = {'DELL': 0.1, 'EBAY': 0.2, 'DLR': 0.2, 'DFS': 0.1, 'JNJ': 0.2}
# sum of all elements in w
sum(w.values())

port_var = cov_matrix.mul(w, axis=0).mul(w, axis=1).sum().sum()
port_var
```

```
np.float64(0.02368233240980641)
```

```
ind_er = df.resample('Y').last().pct_change().mean()
ind_er
```

```
<ipython-input-30-fbdefa6d49e6>:1: FutureWarning: 'Y' is deprecated and will k
  ind_er = df.resample('Y').last().pct_change().mean()
```

	0
DELL	0.346330
EBAY	0.154088
DLR	0.147914
DFS	0.169038
JNJ	0.061898

dtype: float64

```
weights = list(w.values())
port_er = (weights*ind_er).sum()
port_er
```

```
np.float64(0.12431671325943416)
```

```
ann_sd = df.pct_change().apply(lambda x: np.log(1+x)).std().apply(lambda x: x*np.
ann_sd
```

	0
DELL	0.391676
EBAY	0.298816
DLR	0.285104
DFS	0.412309
JNJ	0.184841

dtype: float64

```
data.std()*np.sqrt(250)
```

	0
DELL	0.391583
EBAY	0.298745
DLR	0.285037
DFS	0.412211
JNJ	0.184797

dtype: float64

```
assets = pd.concat([ind_er, ann_sd], axis=1)
assets.columns = ['Returns', 'Volatility']
assets
```

	Returns	Volatility
DELL	0.346330	0.391676
EBAY	0.154088	0.298816
DLR	0.147914	0.285104
DFS	0.169038	0.412309
JNJ	0.061898	0.184841

Next steps: Generate code with assets View recommended plots New interactive sheet

```
p_ret = []
p_vol = []
p_weights = [] # Define an empty array for asset weights

num_assets = len(df.columns)
num_portfolios = 10000
```

```
for portfolio in range(num_portfolios):
    weights = np.random.random(num_assets)
    weights = weights/np.sum(weights)
    p_weights.append(weights)
    returns = np.dot(weights, ind_er) # Returns are the product of individual exp
                                      # weights
    p_ret.append(returns)
    var = cov_matrix.mul(weights, axis=0).mul(weights, axis=1).sum().sum()# Portf
    sd = np.sqrt(var) # Daily standard deviation
    ann_sd = sd*np.sqrt(250) # Annual standard deviation = volatility
    p_vol.append(ann_sd)


data = {'Returns':p_ret, 'Volatility':p_vol}

for counter, symbol in enumerate(df.columns.tolist()):
    #print(counter, symbol)
    data[symbol+' weight'] = [w[counter] for w in p_weights]


portfolios  = pd.DataFrame(data)
portfolios.head() # Dataframe of the 10000 portfolios created
```

	Returns	Volatility	DELL weight	EBAY weight	DLR weight	DFS weight	JNJ weight
0	0.180588	3.452669	0.228999	0.131722	0.162601	0.255977	0.220701
1	0.140342	3.114543	0.115137	0.104037	0.101910	0.255169	0.423747
2	0.176459	3.181484	0.208778	0.247008	0.255580	0.097281	0.191352
3	0.191194	3.697577	0.226682	0.118624	0.258157	0.295676	0.100861
4	0.168228	3.408696	0.179581	0.044232	0.247051	0.279296	0.249841

Next steps: Generate code with portfolios | View recommended plots | New interactive sheet

```
portfolios.plot.scatter(x='Volatility', y='Returns', marker='o', s=10, alpha=0.3, g
```

<Axes: xlabel='Volatility', ylabel='Returns'>