```python
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```python
import matplotlib.pyplot as plt
#%matplotlib plt.rcParams['figure.figsize'] = (12.0, 6.0)
%matplotlib plt.rc('figure', figsize=(20.0, 10.0))
```

```python
import os, sys

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/data'
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")

MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio/nb'
print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")

sys.path.append(MODULEPATH)
print(f"sys.path:{sys.path}")

import numpy as np
import pandas as pd

import edhec_risk_kit_109_BBI as erk
```

## Efficient Frontier - Part III - Running the Optimizer

In order to plot the frontier for portfolios with more than 2 assets, we need to find the weights of th

We start by creating the same sort of function that we already created for the 2 asset case:

```
def plot_ef(n_points, er, cov):
    """
    Plots the multi-asset efficient frontier
    """
    weights = ???? # we need to implement: optimal_weights(n_points, er, cov)
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns", style='.-')
```

But let's start by loading up the data as usual:

```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import edhec_risk_kit_109_BBI as erk

ind = erk.get_ind_returns(DATAPATH)
er = erk.annualize_rets(ind["1996":"2000"], 12)
cov = ind["1996":"2000"].cov()
```

In order to find the optimal weights, we need a function that will minimize the volatility for a given l

```
from scipy.optimize import minimize


def minimize_vol(target_return, er, cov):
    """
    Returns the optimal weights that achieve the target return
    given a set of expected returns and a covariance matrix
    """
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples!
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er: target_return - erk.portfolio_return(
    }
    weights = minimize(erk.portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,return_is_target),
                       bounds=bounds)
    return weights.x
```

Let's use this to recreate the result we got from frontier for the 2-Asset optimization between "Gam
frontier as we did last time:

```
l = ["Games", "Fin"]
erk.plot_ef2(20, er[l], cov.loc[l,l])
```

The Efficient Frontier for the protfolio that has a target return of 0.15 is approx 0.056, so let's see if

```
weights_15 = erk.minimize_vol(0.15, er[l], cov.loc[l,l])
vol_15 = erk.portfolio_vol(weights_15, cov.loc[l,l])
vol_15
```

Perfect!

Now that we can find the weights to minimize the vol given a target return, we can plot the efficient

from the highest to the lowest possible return into a grid, and finding the portfolio that targets the
targeted rate of return.

Add these:

```
def optimal_weights(n_points, er, cov):
    """
    """
    target_rs = np.linspace(er.min(), er.max(), n_points)
    weights = [minimize_vol(target_return, er, cov) for target_return in target_rs]
    return weights

def plot_ef(n_points, er, cov):
    """
    Plots the multi-asset efficient frontier
    """
    weights = optimal_weights(n_points, er, cov) # not yet implemented!
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns", style='.-')
```

```
l = ["Smoke", "Fin", "Games", "Coal"]
erk.plot_ef(50, er[l], cov.loc[l,l])
```

```
%load_ext autoreload
%autoreload 2
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

```
import matplotlib.pyplot as plt
#%matplotlib plt.rcParams['figure.figsize'] = (12.0, 6.0)
%matplotlib plt.rc('figure', figsize=(20.0, 10.0))
```

```
import os, sys

DATAPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfolio
```

```
print(f"DATAPATH:{DATAPATH} contents:{os.listdir(DATAPATH)}")

MODULEPATH = '/content/drive/My Drive/Coursera/EDHEC/investment-portfol
print(f"MODULEPATH:{MODULEPATH} contents:{os.listdir(MODULEPATH)}")

sys.path.append(MODULEPATH)
print(f"sys.path:{sys.path}")

import numpy as np
import pandas as pd

import edhec_risk_kit_109_BBI as erk
```

## Efficient Frontier - Part III - Running the Optimizer

In order to plot the frontier for portfolios with more than 2 assets, we need to find the weights of th
on the efficient frontier.

We start by creating the same sort of function that we already created for the 2 asset case:

```
def plot_ef(n_points, er, cov):
    """
    Plots the multi-asset efficient frontier
    """
    weights = ???? # we need to implement: optimal_weights(n_points, er, cov)
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns", style='.-')
```

But let's start by loading up the data as usual:

```
%load_ext autoreload
%autoreload 2
%matplotlib inline
import edhec_risk_kit_109_BBI as erk

ind = erk.get_ind_returns(DATAPATH)
er = erk.annualize_rets(ind["1996":"2000"], 12)
cov = ind["1996":"2000"].cov()
```

In order to find the optimal weights, we need a function that will minimize the volatility for a given l
return.

```
from scipy.optimize import minimize

def minimize_vol(target_return, er, cov):
    """
    Returns the optimal weights that achieve the target return
    given a set of expected returns and a covariance matrix
    """
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples!
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er: target_return - erk.portfolio_return(
    }
    weights = minimize(erk.portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
                       constraints=(weights_sum_to_1,return_is_target),
                       bounds=bounds)
    return weights.x
```

Let's use this to recreate the result we got from frontier for the 2-Asset optimization between "Gam
"Fin". Let's plot that efficient frontier as we did last time:

```
l = ["Games", "Fin"]
erk.plot_ef2(20, er[l], cov.loc[l,l])
```

The Efficient Frontier for the protfolio that has a target return of 0.15 is approx 0.056, so let's see if
optimizer is able to locate it.

```
weights_15 = erk.minimize_vol(0.15, er[l], cov.loc[l,l])
vol_15 = erk.portfolio_vol(weights_15, cov.loc[l,l])
vol_15
```

Perfect!

Now that we can find the weights to minimize the vol given a target return, we can plot the efficient

dividing up the range from the highest to the lowest possible return into a grid, and finding the port
targets the minimum volatility given a particular targeted rate of return.

Add these:

```
def optimal_weights(n_points, er, cov):
    """
    """
    target_rs = np.linspace(er.min(), er.max(), n_points)
    weights = [minimize_vol(target_return, er, cov) for target_return in target_rs]
    return weights

def plot_ef(n_points, er, cov):
    """
    Plots the multi-asset efficient frontier
    """
    weights = optimal_weights(n_points, er, cov) # not yet implemented!
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns", style='.-')
```

```
l = ["Smoke", "Fin", "Games", "Coal"]
erk.plot_ef(50, er[l], cov.loc[l,l])
```