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

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

```
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_107_BBI as erk
```

# The Efficient Frontier - Part I

This week, we are going to learn how to compute the efficient frontier when we have a set of expec
variances) and correlations (or covariances). It's a fair question as to how we can get these numbe
assume that historic returns are a reasonable estimate. In future sections, we'll learn how to impro

Let's start by importing a new dataset. This is the Ken French dataset of the returns of 30 different

This datafile has a number of minor problems that we'll sort through as we go:

```
import pandas as pd
ind = pd.read_csv(DATAPATH + "/ind30_m_vw_rets.csv", header=0, index_col=0)/100
ind.index = pd.to_datetime(ind.index, format="%Y%m").to_period('M')
```

```
ind.head()
```

```
ind.columns
```

Note that the column names have embedded spaces. We can strip out the leading and trailing spac
`.str.strip` method.

```
ind.columns = ind.columns.str.strip()
```

```
ind.shape
```

This looks good, so let's add the following code to our module for future use:

```
def get_ind_returns():
    """
    Load and format the Ken French 30 Industry Portfolios Value Weighted Monthly Returns
    """
    ind = pd.read_csv("data/ind30_m_vw_rets.csv", header=0, index_col=0)/100
    ind.index = pd.to_datetime(ind.index, format="%Y%m").to_period('M')
    ind.columns = ind.columns.str.strip()
    return ind
```

and then test it by loading the module as usual.

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

import edhec_risk_kit_107_BBI as erk
ind = erk.get_ind_returns(DATAPATH)
ind.shape
```

```
erk.drawdown(ind["Food"])["Drawdown"].plot.line(figsize = (12, 6))
```

```
erk.var_gaussian(ind[["Food", "Beer", "Smoke"]], modified=True)
```

```
erk.var_gaussian(ind).sort_values().plot.bar(figsize = (12, 6))
```

Let's use this as an opportunity to write functions for annualized returns, volatility and sharpe ratio
`edhec_risk_kit.py` file:

```python
def annualize_rets(r, periods_per_year):
    """
    Annualizes a set of returns
    We should infer the periods per year
    but that is currently left as an exercise
    to the reader :-)
    """
    compounded_growth = (1+r).prod()
    n_periods = r.shape[0]
    return compounded_growth**(periods_per_year/n_periods)-1

def annualize_vol(r, periods_per_year):
    """
    Annualizes the vol of a set of returns
    We should infer the periods per year
    but that is currently left as an exercise
    to the reader :-)
    """
    return r.std()*(periods_per_year**0.5)

def sharpe_ratio(r, riskfree_rate, periods_per_year):
    """
    Computes the annualized sharpe ratio of a set of returns
    """
    # convert the annual riskfree rate to per period
    rf_per_period = (1+riskfree_rate)**(1/periods_per_year)-1
    excess_ret = r - rf_per_period
    ann_ex_ret = annualize_rets(excess_ret, periods_per_year)
    ann_vol = annualize_vol(r, periods_per_year)
    return ann_ex_ret/ann_vol
```

```python
erk.sharpe_ratio(ind, 0.03, 12).sort_values()
```

```
erk.sharpe_ratio(ind, 0.03, 12).sort_values().plot.bar(title="Industry Sharpe Ra
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_

```
Enter your authorization code:
··········
Mounted at /content/drive
```

```
(ind["2000":], 0.03, 12).sort_values().plot.bar(title='Industry Sharpe Ratios si
```

## Expected Returns and the Covariance Matrix

Generating the efficient frontier requires a set of expected returns and a covariance matrix. For nov
these simply by looking back in time and naively assuming they will hold in the future. Clearly, they
time to dig into that in future lectures. For the moment, assume that our naive method of estimatin

We can generate an estimate of expected returns using the `annualize_rets()` function, that retur
instance, let's generate the set of expected returns based on historic returns from the 5 year perioc

```
er = erk.annualize_rets(ind["1995":"2000"], 12)
```

```
er.sort_values().plot.bar(figsize = (12,6))
```

Finally, let's generate the covariance matrix. Fortunately, this is easy enough to do using the `.cov`

```
cov = ind["1995":"2000"].cov()
cov.shape
```

In the next lab session, we'll take the expected returns vector and the covariance matrix we've cons
efficient frontier!

```
cov
```

| | Food | Beer | Smoke | Games | Books | Hshld | Clths | Hlth | Chems | Txtls |
|---|---|---|---|---|---|---|---|---|---|---|
| **1926-07** | 0.0056 | -0.0519 | 0.0129 | 0.0293 | 0.1097 | -0.0048 | 0.0808 | 0.0177 | 0.0814 | 0.0039 |
| **1926-08** | 0.0259 | 0.2703 | 0.0650 | 0.0055 | 0.1001 | -0.0358 | -0.0251 | 0.0425 | 0.0550 | 0.0814 |
| **1926-09** | 0.0116 | 0.0402 | 0.0126 | 0.0658 | -0.0099 | 0.0073 | -0.0051 | 0.0069 | 0.0533 | 0.0231 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **1926-10** | -0.0306 | -0.0331 | 0.0106 | -0.0476 | 0.0947 | -0.0468 | 0.0012 | -0.0057 | -0.0476 | 0.0100 |
| **1926-11** | 0.0635 | 0.0729 | 0.0455 | 0.0166 | -0.0580 | -0.0054 | 0.0187 | 0.0542 | 0.0520 | 0.0311 |

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

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

```
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_107_BBI as erk
```

# The Efficient Frontier - Part I

This week, we are going to learn how to compute the efficient frontier when we have a set of expected returns, volatilities (or variances) and correlations (or covariances). It's a fair question as to how we can get these numbers for the future, but for now, we'll assume that historic returns are a reasonable estimate. In future sections, we'll learn how to improve on it.

Let's start by importing a new dataset. This is the Ken French dataset of the returns of 30 different industry portfolios.

This datafile has a number of minor problems that we'll sort through as we go:

```
import pandas as pd
ind = pd.read_csv(DATAPATH + "/ind30_m_vw_rets.csv", header=0, index_co
ind.index = pd.to_datetime(ind.index, format="%Y%m").to_period('M')
```

```
ind.head()
```

```
ind.columns
```

Note that the column names have embedded spaces. We can strip out the leading and trailing spaces in the Series by using the `.str.strip` method.

```
ind.columns = ind.columns.str.strip()
```

```
ind.shape
```

This looks good, so let's add the following code to our module for future use:

```
def get_ind_returns():
    """
    Load and format the Ken French 30 Industry Portfolios Value Weighted Monthly Returns
    """
    ind = pd.read_csv("data/ind30_m_vw_rets.csv", header=0, index_col=0)/100
    ind.index = pd.to_datetime(ind.index, format="%Y%m").to_period('M')
    ind.columns = ind.columns.str.strip()
    return ind
```

and then test it by loading the module as usual.

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

import edhec_risk_kit_107_BBI as erk
ind = erk.get_ind_returns(DATAPATH)
ind.shape
```

```
erk.drawdown(ind["Food"])["Drawdown"].plot.line(figsize = (12, 6))
```

```
erk.var_gaussian(ind[["Food", "Beer", "Smoke"]], modified=True)
```

```
erk.var_gaussian(ind).sort_values().plot.bar(figsize = (12, 6))
```

Let's use this as an opportunity to write functions for annualized returns, volatility and sharpe ratios. Add the following to the `edhec_risk_kit.py` file:

```python
def annualize_rets(r, periods_per_year):
    """
    Annualizes a set of returns
    We should infer the periods per year
    but that is currently left as an exercise
    to the reader :-)
    """
    compounded_growth = (1+r).prod()
    n_periods = r.shape[0]
    return compounded_growth**(periods_per_year/n_periods)-1

def annualize_vol(r, periods_per_year):
    """
    Annualizes the vol of a set of returns
    We should infer the periods per year
    but that is currently left as an exercise
    to the reader :-)
    """
    return r.std()*(periods_per_year**0.5)

def sharpe_ratio(r, riskfree_rate, periods_per_year):
    """
    Computes the annualized sharpe ratio of a set of returns
    """
    # convert the annual riskfree rate to per period
    rf_per_period = (1+riskfree_rate)**(1/periods_per_year)-1
    excess_ret = r - rf_per_period
    ann_ex_ret = annualize_rets(excess_ret, periods_per_year)
    ann_vol = annualize_vol(r, periods_per_year)
    return ann_ex_ret/ann_vol
```

```python
erk.sharpe_ratio(ind, 0.03, 12).sort_values()
```

```
erk.sharpe_ratio(ind, 0.03, 12).sort_values().plot.bar(title="Industry
```

```
1 erk.sharpe_ratio(ind["2000":], 0.03, 12).sort_values().plot.bar(title='
```

## Expected Returns and the Covariance Matrix

Generating the efficient frontier requires a set of expected returns and a covariance matrix. For now, let's assume that we can estiamte these simply by looking back in time and naively assuming they will hold in the future. Clearly, they will not, but we will have plenty of time to dig into that in future lectures. For the moment, assume that our naive method of estimating these parameters wi suffice.

We can generate an estimate of expected returns using the `annualize_rets()` function, that returns a vector of expected returns. For instance, let's generate the set of expected returns based on historic returns from the 5 year period from 1996 through 2000:

```
er = erk.annualize_rets(ind["1995":"2000"], 12)
```

```
er.sort_values().plot.bar(figsize = (12,6))
```

Finally, let's generate the covariance matrix. Fortunately, this is easy enough to do using the `.cov` method:

```
cov = ind["1995":"2000"].cov()
cov.shape
```

In the next lab session, we'll take the expected returns vector and the covariance matrix we've constructed and start to plot the efficient frontier!

```
cov
```