ENGIN 604 Introducción a Python para las Finanzas — Otoño 2021

Tarea 4 - Pauta

Entrega: 11:59pm, Sábado, Abril 17
Enviar a: engin604assignments@gmail.com
Límite máximo de páginas: 5 páginas

Ver políticas de tareas en `https://docenciaweb.fen.uchile.cl`

Open in Colab

El archivo ETFs.xlsx contiene el OHCL (*Open*, *High*, *Close* y *Low*) de los siguientes *Exchange Traded Funds*[1] con frecuencia diaria:

a. **SPY** (S&P500 ETF), desde 2012-11-26 hasta 2020-12-30
b. **EFA** (a non-US equities ETF), desde 2012-12-31 hasta 2020-12-30
c. **IJS** (a small-cap value ETF), desde 2012-10-31 hasta 2020-12-30
d. **EEM** (an emerging-markets ETF), desde 2012-04-25 hasta 2020-12-30
e. **AGG** (A bond ETF), desde 2012-12-31 hasta 2020-12-30

Todos los datos fueron extraido desde *Yahoo Finance* utilizando la libería `YFinance`.

1. Cargue el archivo ETFs.xlsx a su espacio de trabajo de manera que cada hoja sea almacenada en un diccionario.

```
import pandas as pd # se carga pandas
import numpy as np # se carga numpy
from datetime import datetime # se carga datetime

# lee etfs en diccionario
etfs = pd.read_excel('ETFs.xlsx', sheet_name=None, index_col=0)
```

2. Genere un DataFrame que contenga el precio al cierre de cada ETF.
   a. La columna (`columns`) debe tener el nombre del ETF y en el índice (`index`) la fecha.
   b. Solo trabaje con los datos desde 2012-12-31 hasta 2020-12-30.

   Explique al menos dos maneras de realizar (2.b).

```
# lista vacía
etfs_list = []
```

[1] Son fondos de inversión colectiva cuya política de inversión consiste en reproducir un índice.

```
# guarda en una lista los df con el precio al cierre
for etf_name, etf_price in etfs.items():
    etfs_list.append(etf_price.loc[:,['Close']].rename(columns={'Close':etf_name}))

# se concatena la lista
etfs_daily = pd.concat(etfs_list, axis=1)
```

3. Transforme el DataFrame creado en (2) a frecuencia mensual. Lo anterior se realiza asumiendo que el último precio disponible del mes corresponde al precio mensual.

```
# se extrae el año de la fecha en el índice
etfs_daily['Year'] = etfs_daily.index.year

# se extrae el mes de la fecha en el índice
etfs_daily['Month'] = etfs_daily.index.month

# se extrae el día de la fecha en el índice
etfs_daily['Day'] = etfs_daily.index.day

# remueve la hora de la fecha en el índice
etfs_daily.index = etfs_daily.index.date

# elimina los na para que el df comience desde 2012-12-31
etfs_daily.dropna(inplace=True)

# agrupa y extrae el último precio como mensual
etfs_monthly = etfs_daily.groupby(['Year','Month']).tail(1)
```

4. Utilizando una función `lambda` (anónima) genere el retorno logarítmico mensual para cada ETF. No olvide eliminar la fila de `NAs`.

$$R_{j,t} = ln\left(\frac{P_{j,t}}{P_{j,t-1}}\right) \quad j = 1,..,5 \quad t = 1,..,97$$

Hint: NumPy tiene la función `np.log()` para calcular el logaritmo natural.

```
# se define la función
def logret(x):
  return(np.log(x/x.shift(1)))

# se calcula el retorno logarítmico
etfs_monthly_log_return = etfs_monthly.loc[:,'SPY':'AGG'].apply(lambda x: logret(x))

# elimina la fila con na
etfs_monthly_log_return.dropna(inplace=True)
```

5. Genere una función que permita calcular una breve estadística descriptiva que contenga: promedio aritmético, desvación estandar, *kurtosis*, *skewness* y ratio de Sharpe (asuma tasa libre de riesgo igual a 0). Para la función tener en consideración lo siguiente:

   - El *input* debe ser un Pandas con estructura Series.
   - El *output* debe ser un Pandas con estructura Series donde:

- El atributo `name` contenga el nombre del *input*.
- El atributo `index` contenga el nombre de cada estadística descriptiva realizada.

Aplique la función al DataFrame creado en (4) mediante una función `lambda` (anónima).

```
# se define función para estadística descriptiva
def desc_stat(x):

    mean = sum(x)/len(x)
    std = (sum((x-mean) ** 2)/(len(x)-1)) ** (1/2)
    kurtosis = (sum((x-mean) ** 4)/len(x))/((sum((x-mean) ** 2)/len(x)) ** 2)
    skewness = (sum((x-mean) ** 3)/len(x))/((sum((x-mean) ** 2)/len(x)) ** (1.5))

    output = pd.Series(data=[mean, std, kurtosis, skewness],
                       index=['mean', 'std', 'kurtosis', 'skewness'])

    return(output)

# se aplica la función
summary = etfs_monthly_log_return.apply(lambda x: desc_stat(x))
```

6. Mediante un gráfico de linea grafique el retorno logarítmico acumulado de cada ETF.

```
# se grafíca el retorno acumulado de cada etf
etfs_monthly_log_return.loc[:,'SPY':'AGG'].apply(lambda x: x.cumsum()).plot()
```

7. El portafolio tangente es aquel maximiza el ratio de Sharpe. El problema de optimización expresado es su forma matricial es:

$$\max_{\mathbf{t}} \frac{\mathbf{t}'\mu - r_f}{(\mathbf{t}'\Sigma\mathbf{t})^{\frac{1}{2}}} = \frac{\mu_{p,t} - r_f}{\sigma_{p,t}} \quad s.t \quad \mathbf{t}'\mathbf{1} = 1$$

Cuya solución (también matricial) es:

$$\mathbf{t} = \frac{\Sigma^{-1}(\mu - r_f \cdot \mathbf{1})}{\mathbf{1}'\Sigma^{-1}(\mu - r_f \cdot \mathbf{1})}$$

Donde $\mathbf{t}$ es el vector de pesos óptimos del portafolio tangente, $\mathbf{1}$ un vector de unos cuya dimensión es $j \times 1$ siendo j el total de activos y Σ^{-1} el inverso de la matriz de varianza-covarianza ($j \times j$).

Utilizando la solución matricial para $\mathbf{t}$, genere una función cuyos *inputs* sean: el vector de retornos promedio, la matriz de varianza-covarianza de los activos (ETFs) que conformarían el portafolio tangente y la tasa libre de riesgo (escalar).

El *output* de la función debe ser los pesos $\mathbf{t}$, el retorno esperado y desviación estandar del portafolio tangente.

- Retorno esperado del portafolio tangente es:

$$R_{p,t} = \mathbf{t}' \cdot \mu$$

Donde μ es el vector de retornos promedio de dimensión $j \times 1$.

- La desviación estandar del portafolio de mínima varianza es:

$$\sigma_{p,t} = \sqrt{\mathbf{t}'\Sigma\mathbf{t}}$$

Pruebe la función utilizando los retornos promedios creados en (5) y genere la matriz de varianza-covarianza utilizando el DataFrame generado en (4). Explique linea por linea su función. Asuma una tasa libre de riesgo igual a 0.005.

Hint: Para transformar filas o columnas de un DataFrame a NumPy se utiliza luego de la selección (`loc` o `iloc`) el atributo `.to_numpy()`.

```
# se extrae la fila de promedio creado con la función y se pasa a numpy array
Er_mat = summary.loc[['mean'],:].to_numpy()

# calculo de la matriz de varianza-covarianza
Cov_mat = np.cov(etfs_monthly_log_return, rowvar=False)

# se define la función que calcula el portafolio tangente
def tangency_port(mu, cov, rf):

    m,n = mu.shape

    one_vec = np.ones((n,1))
    sigma_inv_mat = np.linalg.inv(cov)

    mu_minus_rf = mu - rf

    top_mat = sigma_inv_mat.dot(mu_minus_rf.T)
    bottom_escalar = one_vec.T.dot(top_mat)

    wi = top_mat / bottom_escalar
    er = mu.dot(wi)
    sigma2 = (wi.T.dot(cov).dot(wi)) ** (1/2)

    return([wi, er, sigma2])

# tasa libre de riesgo
rf = 0.005

# se calcula el portafolio tangente
tangency_port(Er_mat, Cov_mat, rf)
```