Mean-Variance portfolio optimization

Santiago Fada,Valentin Diaz

November 2022

.

Índice

1. Introducción

1.1. Resumen

Este trabajo tiene como objetivo el análisis del modelo Mean-Variance como representación de la realidad y su utilización para simular una estrategia de inversión. El análisis se tratará de explicar de manera gráfica y numérica aquellos parámetros que son utilizados en el modelo y las soluciones que el modelo nos propone.

1.2. Introducción

El modelo Mean-Variance como optimización de portfolio fue introducido por el economista estadounidense Harry Markowitz. Se basa en un modelo de optimización convexa que busca maximizar el retorno y minimizar el riesgo. Para esto se utilizan el promedio de variación de datos históricos y la covarianza de variaciones entre 2 acciones como parámetros de retorno y riesgo respectivamente. La solución de este modelo nos indicará como distribuir nuestro presupuesto en las acciones de interés.

2. Desarrollo

2.1. Modelización matemática

Se trata de un modelo de programación cuadrática en el que se busca maximizar el retorno y minimizar el riesgo.Para esto se calcula la variación media de los retornos históricos de los de los precios de las acciones y se utiliza la covarianza entre dichas variaciones como parámetro para medir la volatilidad. Para esto de definen:

- $x \in R^n$ donde cada coordenada sera el porcentaje a invertir en una determinada acción.
- $\mu \in R^n$ que contiene el retorno promedio de cada acción.
- $V \in R^{n \times n}$ es la matriz de covarianza, que nos indica el grado de independencia entre los diferentes activos.

Modelo clásico y principal

$$\begin{aligned} \max_x \ & \mu^T x - \gamma x^T V x \\ \text{s.t} \quad & 1^T x = 1 \end{aligned}$$

fijando un retorno mínimo $\hat{\mu}$

$$\begin{aligned} \min_x \ & x^T V x \\ \text{s.t} \ & \mu^T x \geq \hat{\mu} \\ & 1^T x = 1 \end{aligned}$$

fijando un riesgo máximo $\hat{\sigma}$

$$\begin{aligned} \max_x \ & \mu^T x \\ \text{s.t} \quad & \mathrm{x}^T V x \leq \hat{\sigma} \\ & 1^T x = 1 \end{aligned}$$

2.2. Programación cuadrática

Un problema de programación cuadrática (PC) es un problema de optimización,donde se busca encontrar el máximo/mínimo de una función cuadrática de varias variables, sujeto a un conjunto de restricciones lineales de dichas variables

Todo problema de programación cuadrática puede escribirse en forma estándar de la siguiente manera

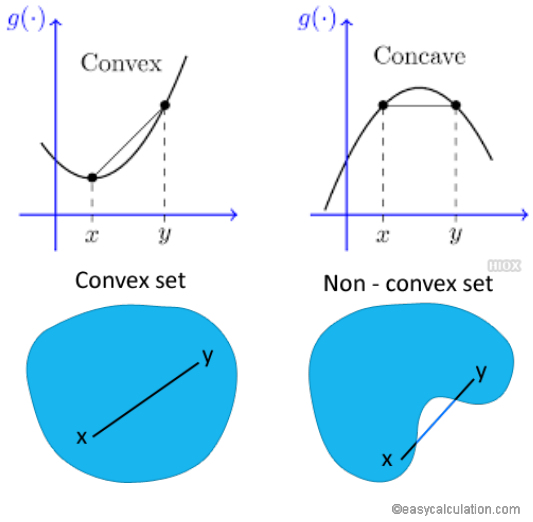
$$\begin{aligned} \max_x \ & \tfrac{1}{2}\, x^T Q x + c^T x \\ \text{s.t} \ & \text{A x} = \text{b} \\ & \text{x} \geq 0 \end{aligned}$$

Si Q es simétrica semi definida positiva entonces el la función es convexa y como las restricciones también lo son. El problema sera de optimización convexa.

2.3. Optimización Convexa

• Una función $g : X \mapsto R$ es convexa si para x,y $\in X$,$\lambda \in [0, 1]$ vale que $g(\lambda x + (1 - \lambda)y) \leq \lambda g(x) + (1 - \lambda)g(y)$

•Un conjunto C es convexo si $\forall x, y \in C, \lambda \in [0, 1]$ vale que $\lambda x + (1 - \lambda)y \in C$



Interpretación gráfica de funciones y conjuntos convexos

Nos interesa estar tratando con un problema convexo puesto que bajo estas condiciones todo óptimo local, va a ser también un punto óptimo global por lo

que en caso de encontrar una solución óptima sabemos que va a ser la solución óptima del problema por lo tanto tenemos un criterios de parada.

Esto es muy bueno puesto que por como se conforma la Matriz de Covarianza en el modelo de Markowitz, esta siempre sera simétrica semi definida positiva, por lo que el modelo de Mean-Variance sera un problema sera convexo.

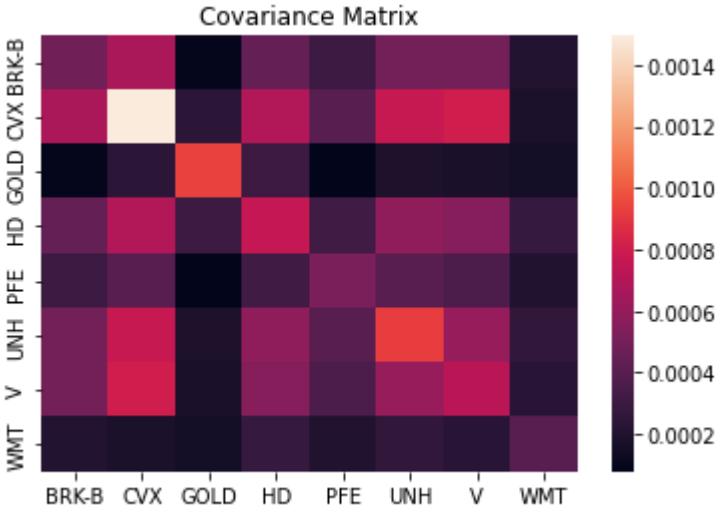
3. Análisis

El análisis será hecho con datos del año 2020 y un portfolio de 8 acciones: Oro, Berkshire Hathaway, Walmart, UnitedHealth, Visa, Chevron, Breville Group y Pfizer.

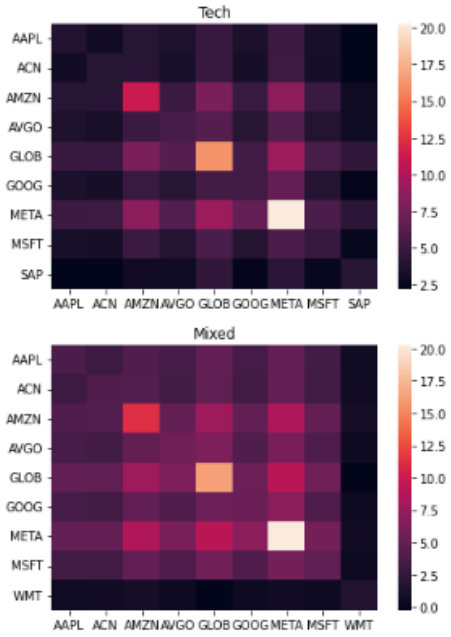
```python
import numpy as np
import yfinance as yf
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sc

def getReturns(stocks, start, end): #descargamos los datos
    data = yf.download(stocks, start, end)['Close']
    return data

stocks="GOLD BRK-B WMT UNH V CVX HD PFE"
start="2020-01-01"
end="2020-12-30"
data= getReturns(stocks, start, end)

tr_days=data.shape[0]  #cantidad de dias
returns= data.pct_change() #Variacion porcentual del precio diario de la acción
mean_returns= returns.mean() #Vector mu
Cov=returns.cov()  #Matriz de covarianza V
```

Dado que la matriz de covarianza son números, uno puede asignar colores a los números y visualizarla de manera gráfica:

```python
def Cov_mtrx(Cov):
    ax = sns.heatmap(Cov, cbar=True)
    plt.title("Covariance Matrix")
    plt.show()
Cov_mtrx(Cov)
```



Para entenderlo mejor, graficamos la matriz de covarianza de un portfolio armado por empresas de tecnología y de un portfolio idéntico al cual le reemplazamos una de esas empresas por Walmart que es del ámbito del retail:



Vemos como en la primera imagen la matriz tiene cierta distribución uniforme del color, pero en la segunda imagen, la última fila y la última columna tienen un color muy oscuro comparado al resto. Esa fila y columna corresponde a la covarianza de Walmart y el resto de acciones. Por lo tanto, en un principio se podría concordar en que la covarianza mide lo que esperamos.

Ahora empezaremos a crear las funciones que nos solucionen el problema, pero primero debemos hablar de dos portfolios destacados. Dentro de todas las soluciones posibles existen un único portfolio el cual tiene el mínimo riesgo y un único portfolio que maximiza el Sharpe-Ratio.

Se define el Sharpe Ratio como:

$$\mathrm{S} = \frac{R(x) - R_f(x)}{\sigma(x)}$$

Donde R es el retorno, sigma la desviación estándar del portfolio R_f es el risk-free asset el cual se puede considerar 0, que es lo que nosotros haremos. Este Ratio nos indica la cantidad de retorno por unidad de riesgo, y por esto uno quisiera maximizarlo.

```python
#Cuando encontremos x esta función nos indicará el retorno y el riesgo del portfolio
def PortfolioPerformance(x, mean_returns, Cov, tr_days):
    returns= (mean_returns@x)*tr_days
    var=((x.T@Cov)@x)*tr_days
    std=np.sqrt(var)
    return returns, std


#Queremos maximizar el sharpe ratio, por lo tanto minimizaremos el -sharp ratio
def SharpeRatio(x, mean_returns, Cov, tr_days):
    #Asumimos risk free rate=0
    returns, std= PortfolioPerformance(x, mean_returns, Cov, tr_days)
    return -returns/std

def max_SR(mean_returns, Cov, tr_days):
    num_assets= len(mean_returns)
    eq_cons= ({'type':'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds=tuple((0,1) for j in range(num_assets))
    arg= (mean_returns, Cov, tr_days)
    sol = sc.optimize.minimize(fun=SharpeRatio, x0=np.ones(num_assets), args=arg,
                               method='SLSQP', bounds=bounds, constraints=eq_cons)
    return -sol['fun'], sol['x']


#Minimizar la desviacion es lo mismo q minimizar la varianza
def PortfolioVar(x, mean_returns, Cov, tr_days):
    var= PortfolioPerformance(x, mean_returns, Cov, tr_days)[1]
    return var

def MinVar(mean_returns, Cov, tr_days):
    num_assets= len(mean_returns)
    arg= (mean_returns, Cov, tr_days)
    eq_cons= ({'type':'eq', 'fun': lambda x: np.sum(x) - 1})
    bounds=tuple((0,1) for j in range(num_assets))
    sol = sc.optimize.minimize(fun=PortfolioVar, x0=np.ones(num_assets), args=arg,
                               method='SLSQP', bounds=bounds, constraints=eq_cons)
    return sol['fun'], sol['x']
```

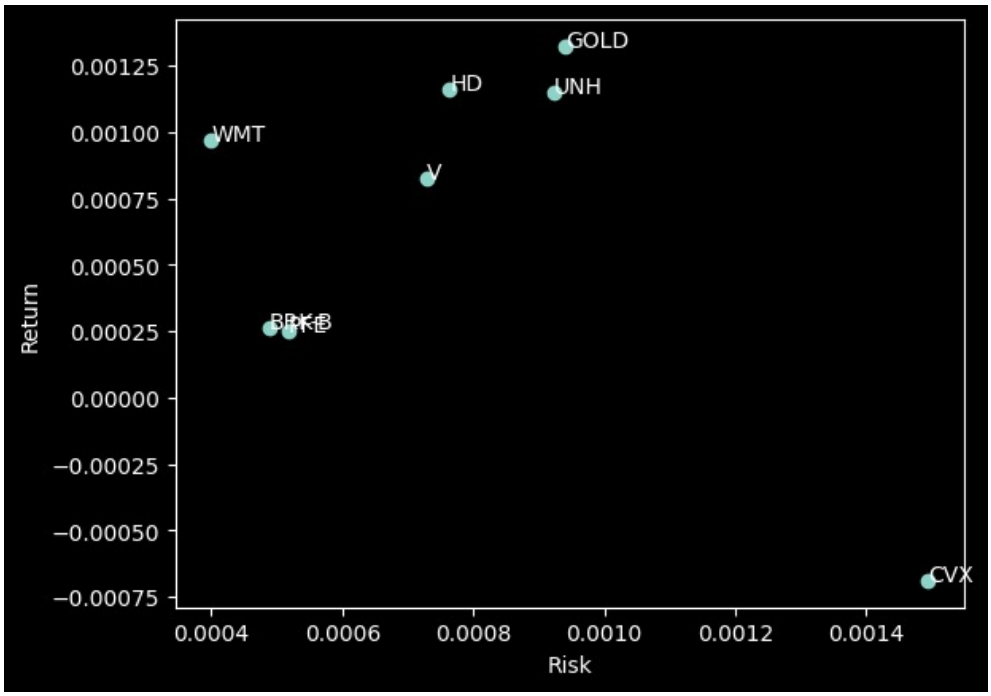
Luego, la función que resuelve el modelo de Markowitz para un retorno dado:

```python
#Minimizar la varianza dado target return
def returnConstr(x, mean_returns, Cov, tr_days):
    return PortfolioPerformance(x, mean_returns, Cov, tr_days)[0]

def Mark_port(mean_returns, Cov, tr_days, target):
    num_assets= len(mean_returns)
    arg= (mean_returns, Cov, tr_days)
    eq_cons= ({'type':'eq', 'fun': lambda x: np.sum(x) - 1},
            {'type':'eq', 'fun': lambda x: returnConstr(x, mean_returns, Cov, tr_days) - target})
    bounds=tuple((0,1) for j in range(num_assets))
    sol = sc.optimize.minimize(fun=PortfolioVar, x0=np.ones(num_assets), args=arg, method='SLSQP', bounds=bounds, constraints=eq_cons)
    return sol['fun'], sol['x']
```

Ahora veamos como estas 3 funciones nos recomiendan distribuir el presupuesto:

```python
def maxSR_data(mean_returns, Cov, tr_days, graph):
    SR= list(max_SR(mean_returns, Cov, tr_days))
    returns, std= PortfolioPerformance(SR[1], mean_returns, Cov, tr_days)
    SR[1]=np.round(SR[1], decimals=2)
    SR_df= pd.DataFrame(SR[1], index=mean_returns.index, columns=["pct"])
    if graph==True:
        fig, ax=plt.subplots()
        plt.style.use('dark_background')
        ax.set_xlabel("Risk")
        ax.set_ylabel("Return")
        tickers=mean_returns.index.tolist()
        plt.scatter(np.diag(Cov), mean_returns.values)
        for j in range(len(tickers)):
            plt.annotate(tickers[j], (np.diag(Cov)[j], mean_returns.values[j]))
        plt.show()
    return returns, std, SR_df

def MinVar_data(mean_returns, Cov, tr_days, graph):
    MV= list(MinVar(mean_returns, Cov, tr_days))
    returns, std= PortfolioPerformance(MV[1], mean_returns, Cov, tr_days)
    MV[1]=np.round(MV[1], decimals=2)
    SR_df= pd.DataFrame(MV[1], index=mean_returns.index, columns=["pct"])

    if graph==True:
        fig, ax=plt.subplots()
        plt.style.use('dark_background')
        ax.set_xlabel("Risk")
        ax.set_ylabel("Return")
        tickers=mean_returns.index.tolist()
        plt.scatter(np.diag(Cov), mean_returns.values)
        for j in range(len(tickers)):
            plt.annotate(tickers[j], (np.diag(Cov)[j], mean_returns.values[j]))
        plt.show()
    return returns, std, SR_df

def MarkPort_data(mean_returns, Cov, tr_days, target, graph): #Hacer que graph sea una parametro opcional
    Mk_P= list(Mark_port(mean_returns, Cov, tr_days, target))
    returns, std= PortfolioPerformance(Mk_P[1], mean_returns, Cov, tr_days)
    Mk_P[1]=np.round(Mk_P[1], decimals=2)
    SR_df= pd.DataFrame(Mk_P[1], index=mean_returns.index, columns=["pct"])
    if graph==True:
        fig, ax=plt.subplots()
        plt.style.use('dark_background')
        ax.set_xlabel("Risk")
        ax.set_ylabel("Return")
        tickers=mean_returns.index.tolist()
        plt.scatter(np.diag(Cov), mean_returns.values)
        for j in range(len(tickers)):
            plt.annotate(tickers[j], (np.diag(Cov)[j], mean_returns.values[j]))
        plt.show()
    return returns, std, SR_df


bar1=maxSR_data(mean_returns, Cov, tr_days, graph=False)[2].plot.bar(title="Assets allocation by Max Sharpe Ratio")
bar2=MinVar_data(mean_returns, Cov, tr_days, graph=False)[2].plot.bar(title="Assets allocation by Min Volatility")
bar3=MarkPort_data(mean_returns, Cov, tr_days, target=0.27964, graph=False)[2].plot.bar(title="Assets allocation by given target")
```

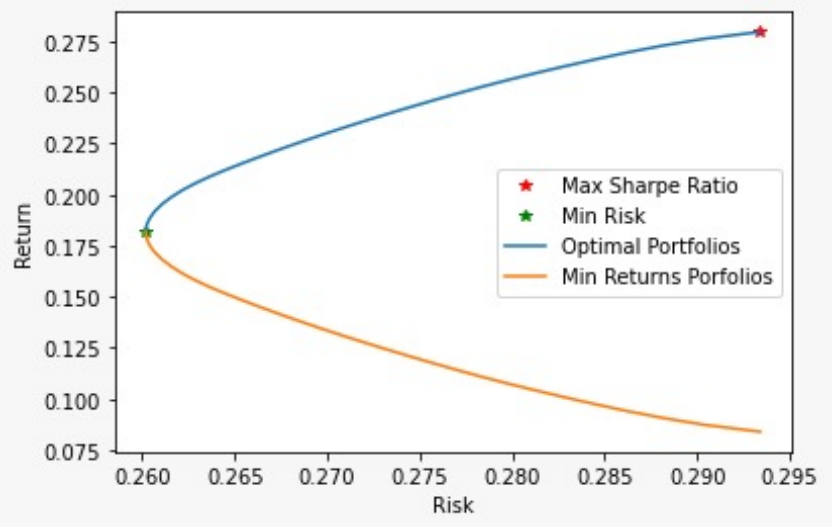
Notar que el grafico del given target es el mismo que el del Sharpe Ratio (SR), esto es así porque el retorno objetivo que usamos en la función es el retorno del SR.
Si graficamos las acciones con respecto a sus riesgos y retornos nos dará una mejor idea de cómo los elige:



Efectivamente el portfolio de mínima varianza elige las 3 acciones con menor riesgo y suma al oro el cual tiene un gran retorno. Por otro lado, el máx. SR elige 3 acciones que tienen alto nivel de retorno, pero un riesgo no tan alto.

Por último graficaremos lo que se conoce como frontera eficiente. En esta viven todas las soluciones al problema cuadrático (soluciones óptimas) dado un retorno buscado. Esta frontera nos ayuda a definir un espacio en donde todos los portfolios posibles (incluyendo los no óptimos) viven entre la frontera eficiente y su reflexión:

```python
def efficient_frontier(mean_returns, Cov, tr_days):
    fig, ax=plt.subplots()
    y_targets=np.linspace(MinVar_data(mean_returns, Cov, tr_days, graph=False)[0]
                          , maxSR_data(mean_returns, Cov, tr_days, graph=False)[0],30)
    y2_targets=[]
    x_stds=[]
    x2_stds=[]
    for j in range(len(y_targets)):
        x_std = MarkPort_data(mean_returns, Cov, tr_days, y_targets[j], graph=False)[1]
        x_stds.append(x_std)

        y2_targets.append(-y_targets[j]+2*y_targets[0])
        x2_std = MarkPort_data(mean_returns, Cov, tr_days, y_targets[j], graph=False)[1]
        x2_stds.append(x_std)

    plt.plot(x_stds[-1],y_targets[-1] ,'*', color="white",label="Max Sharpe Ratio")
    plt.plot(x_stds[0],y_targets[0] ,'*', color="white",label="Min Risk")
    plt.plot(x_stds, y_targets, label="Optimal Portfolios", color="green")
    plt.plot(x2_stds, y2_targets, label="Min Returns Porfolios", color="red")
    ax.set_xlabel("Risk")
    ax.set_ylabel("Return")
    plt.legend()
    plt.show()
```



4. Simulación de Inversión

Al utilizar el modelo Mean-Variance explicado anteriormente, la mayoría de las veces los retornos eran positivos, pero notamos que la cantidad de veces en que los retornos eran negativos era considerablemente alta. Entonces para mitigar las pérdidas, en vez de trabajar con un solo portfolio, trabajaremos con 8 portfolios de diferentes rubros.

Utilizaremos los datos históricos de 20 días en el pasado para predecir un día

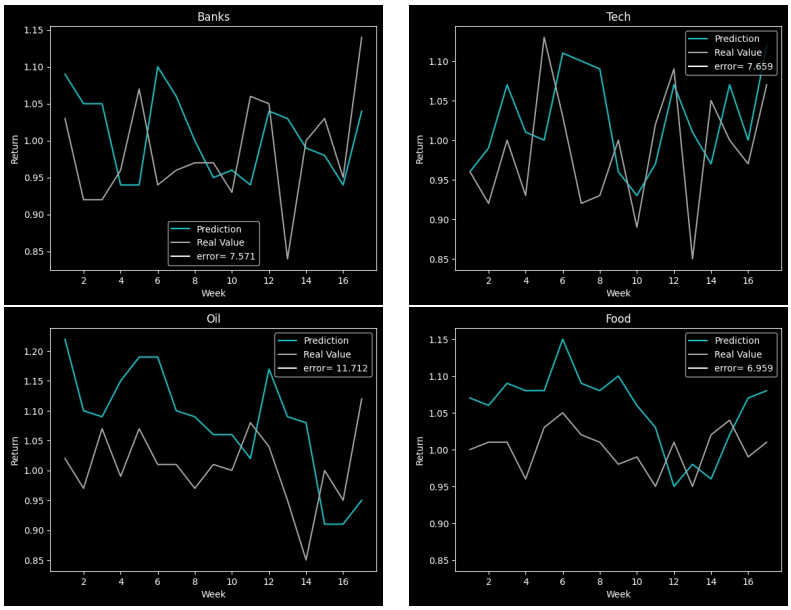
en el futuro, esto lo haremos una vez por semana durante 140 días de apertura de mercado (6 meses y medio aprox).

Si bien el modelo de Markowitz no predice los precios, al principio supondremos que el mercado tiene un comportamiento parecido al del máx. SR y luego ajustaremos la decisión de compra o no de acciones a partir de la performance de la predicción.

```
assets={"Tech":["AAPL MSFT GOOG AMZN META AVGO SAP ACN GLOB"],
        "Banks":["JPM BAC WFC SCHW MS GS C USB PNC"],
        "Mixed":["AAPL MSFT GOOG AMZN META AVGO ACN GLOB WMT"],
        "Otros1":["TSLA UNH XOM JNJ V NVDA LLY PG MA"],
        "Otros2":["HD BAC ABBV PFE KO MRK PEP COST ORCL"],
        "Otros3":["TMO MCD CSCO DHR TMUS ABT WFC NEE NKE"],
        "Oil":["SHEL CVX XOM COP TTE EQNR EPD HES"],
        "Food":["MCD MDLZ ADM GIS KHC HSY CMG HRL"]
        }

start="2022-01-01" #140 tr days. 20 semanas. 7 bloques de 20 dias
end="2022-07-26"

def results(data):
    pred_returns=[]
    real_returns=[]

    for j in range(21,134,7):
       #Este for permite indexar al primer dia de la prediccion tomando 20 anteriores y hasta el dia 140
        returns= data.pct_change()

        #datos reales

        weekly_returns= returns.iloc[j:j+7,:].mean()
         #Array con las medias de todas las acciones

        #Prediccion
        mean_returns=returns.iloc[j-20:j,:].mean()
        Cov=returns.iloc[j-20:j,:].cov()
        allocation=max_SR(mean_returns, Cov, tr_days=20)[1]
        #Cálculo total de performance
        pred_return_pct=PortfolioPerformance(allocation, mean_returns, Cov, 20)[0]+1
        real_return_pct=np.prod(weekly_returns+1)

        pred_returns.append(pred_return_pct)
        real_returns.append(real_return_pct)

    return pred_returns, real_returns

#Separamos los días en semanas
def week_split(data):
    data1=data.T
    dates=[]
    days=[]
    for j in range(len(data1.columns)): #Creo lista con todas las fechas
        dates.append(str(data1.columns[j])[:10])
        #Hay q agarrar hasta :10 porque después tenía la hora
    for j in range(21, len(dates), 7):
        days.append(dates[j])
    return days

```

```
#Grafico del rendimiento de cada portfolio...
#... comparado con el valor real

def grafico(data, industry):
    y1, y2=results(data)[0], results(data)[1]
    y1=np.array(y1)
    y2=np.array(y2)

    errores=[]
    for j in range(len(y1)):
        errores.append(abs(round(y1[j]-y2[j],3)*100))
    error=sum(errores)/len(errores)

    y1=np.round(y1,2)
    y2=np.round(y2,2)
    xdates=week_split(data)
    x1=np.linspace(1,len(xdates),len(xdates))

    fig, ax=plt.subplots()
    plt.style.use('dark_background')
    plt.plot(x1,y1, color="darkturquoise",label="Prediction")
    plt.plot(x1,y2, color="darkgray",label="Real Value")
    plt.plot(2,0.85, color="white",label="error= "+str(round(error,3)))
    plt.title(industry)
    ax.set_xlabel("Week")
    ax.set_ylabel("Return")
    plt.legend()
    plt.show()

#Se descarga toda la data
def portfolios():
    FullData=[]
    for industry in list(assets.keys()):
        data= getReturns(assets[industry][0], start, end)
        FullData.append(data)
    return FullData

#Gráfico de todos los portfolios
def ejecucion(FullData):
    k=0
    for industry in list(assets.keys()):
        grafico(FullData[k], industry)
        k+=1

FullData=portfolios()
ejecucion(FullData)
```

Por razones de espacio solo mostraremos el rendimiento de 4 porfolios



Vemos como el máx. SR es una aproximación considerablemente buena, y por lo general aproxima el retorno por encima del valor real.

Haciendo esto vimos que por lo general el error de la predicción rondaba el 8 % y algunas veces aisladas superaban el 10 %.

Este análisis lo utilizamos para ajustar la decisión de compra del modelo. Si la predicción del precio de la acción superaba el 10 %, entonces las acciones de esa semana eran compradas, y si el rendimiento estaba por debajo no se compraban (pues no se desean retornos negativos).

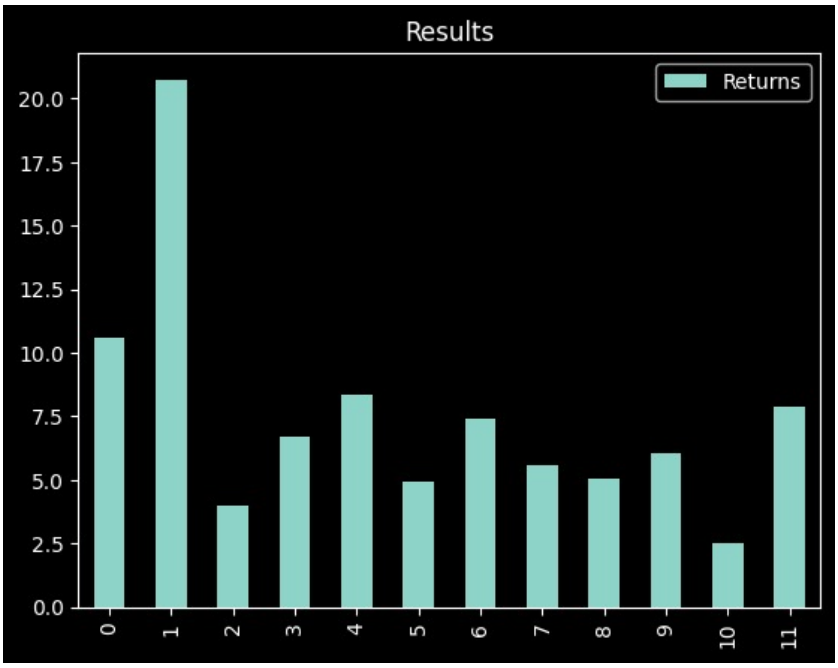
Utilizamos finalmente esta estrategia con los tiempos y portfolios aclarados al principio en las siguientes 12 ventanas de fechas:

```
["2020-01-01","2020-07-23"]
["2020-03-01","2020-09-18"]
["2020-05-01","2020-11-18"]
["2020-07-01","2021-01-21"]
["2020-09-01","2021-03-24"]
["2020-11-01","2021-05-25"]
["2021-01-01","2021-07-26"]
["2021-03-01","2021-09-17"]
["2021-05-01","2021-11-18"]
["2021-07-01","2022-01-20"]
["2021-09-01","2022-03-23"]
["2021-11-01","2022-05-23"]
["2022-01-01","2022-07-26"]
```

```python
#Simulación de compra de acciones
def review(FullData):
    benefits=[]
    for j in range(len(FullData)):
        preds, real=results(FullData[j])[0], results(FullData[j])[1]
        errores=[]
        for j in range(len(preds)):
            errores.append(abs(round(preds[j]-real[j],3)))
        error=sum(errores)/len(errores)

        prod1, prod2=1, 1
        for j in range(len(preds)):
            if preds[j]>=1.1:  #Si la predición es un retorno +10% comprar
                prod1=preds[j]*prod1
                prod2=real[j]*prod2
        prod1=round((prod1-1)*100,2)
        prod2=round((prod2-1)*100,2)
        #Imprimo el retorno esperado y el real de cada portfolio.
        print("Expected return: {:.2f}% \nReal Return: {:.2f}%\n".format(prod1, prod2))
        benefits.append(prod2)

    return benefits

#El retorno esperado tiene un error muy grande por la propagacion de errores de cada semana.
#Si el retorno esperado y el real es 0% indica que la predicción nunca superó el 10%.

print("Total Return: {}%".format(sum(review(FullData))/8))
```

Y los resultados fueron:

```python
resultados=[10.6,20.76,3.96,6.71,8.34,4.9,7.41,5.58,7.25 ,5.02,6.02,2.48,7.85]
results= pd.DataFrame(resultados, index=None, columns=["Returns"])
plt.style.use('dark_background')
bar= results.plot.bar(title="Results")
```



5. Conclusión

Durante el desarrollo del trabajo, pudimos ver como el modelado matemático del problema tiene gran correspondencia con la realidad con un cierto grado de error. Fue importante en un primer lugar el análisis del modelo para luego poder crear una estrategia de inversión que se ajuste a lo observado para generar una mayor ganancia.

Si bien las ganancias generadas fueron considerables, uno podría esperar una mejora de estas a través de aspectos que el modelo no tiene en cuenta que se podrían considerar como falencias. Por ejemplo, la utilización otros tipos de riesgos o de noticias periodísticas (que estén relacionadas con las acciones) para complementar el parámetro de la variación de los precios.

De todos modos, este modelo es un gran primer paso para poder generar una estrategia de inversión más robusta que permita una mayor ganancia y un menor riesgo de lo conseguido.

6. Referencias

[1]Gérard Cornuéjols, Javier Peña, Reha Tütüncü. Optimization Methods in Finance, University of Cambridge, 2018.