Optimización de Portafolio con Clustering Jerárquico y Markowitz

Rommel Lopez Garcia

Problema de Negocio

El problema de negocio se centra en la necesidad de **gestionar portafolios de inversión** de forma eficiente, maximizando el retorno ajustado por riesgo y minimizando el riesgo general mediante una correcta diversificación. El uso de **clustering** y el **modelo de Markowitz**, con la incorporación de técnicas de regularización, permite abordar de manera efectiva la selección de activos y la construcción de un portafolio optimizado, ayudando a los inversores a tomar decisiones informadas y estratégicas en un entorno financiero dinámico y a menudo incierto.

1. **Clustering Jerárquico**: Se utiliza el algoritmo de clustering jerárquico para agrupar activos con patrones similares en sus retornos, basados en una matriz de correlación. Esto permite a los inversores identificar activos que podrían formar un grupo coherente desde el punto de vista de su comportamiento de mercado.

2. **Optimización de Portafolio de Markowitz**: Posteriormente, el problema de negocio se resuelve utilizando la **teoría de Markowitz** para optimizar los pesos de los activos seleccionados, maximizando el índice Sharpe, que mide la rentabilidad ajustada por riesgo.

Definición de Tickers

```
TICKERS = [
    # Tecnología
    'AAPL', 'MSFT', 'GOOGL', 'NVDA','AMZN', 'META', 'INTC', 'AMD', 'CSCO',

    # Salud
    'JNJ', 'PFE', 'MRK', 'UNH', 'ABBV', 'BMY', 'GILD', 'REGN', 'VRTX', 'LLY',

    # Energía (Energy)
    'XOM', 'CVX', 'SLB', 'BP', 'COP', 'OXY', 'EOG', 'KMI', 'PSX', 'HAL',

    # Finanzas
    'JPM', 'GS', 'BRK-B', 'C', 'BAC', 'WFC', 'MS', 'AXP', 'BLK', 'SCHW',

    # Consumo Básico
    'KO', 'PEP', 'PG', 'WMT', 'MO', 'CL', 'KMB', 'EL', 'STZ', 'COST',

    # Consumo Discrecional
    'DIS', 'TSLA', 'NFLX', 'HD', 'NKE', 'SBUX', 'MCD', 'TGT', 'LOW',

    # Industriales
    'BA', 'CAT', 'UPS', 'LMT', 'MMM', 'GE', 'HON', 'DE', 'UNP', 'CSX',

    # Telecomunicaciones
    'VZ', 'T', 'TMUS', 'CHT', 'CMCSA', 'EA', 'NOK', 'ERIC', 'CCI', 'SBAC',

    # ETFs
    'SPY', 'QQQ', 'VTI', 'EWW', 'IEMG', 'EFA', 'VWO', 'IXC', 'XLE', 'XLF',
]
```

Descripción del Dataset

El dataset utilizado consta de precios históricos de cierre de diversos activos financieros en diferentes sectores económicos, buscando así una correcta diversificación, estos datos son obtenidos de Yahoo Finance. Los activos están identificados por sus símbolos ticker (por ejemplo, AAPL para Apple, MSFT para Microsoft).

- Periodo de los datos: Desde enero de 2014 hasta diciembre de 2024.
- Características: Precios de cierre de los activos seleccionados.

Análisis Exploratorio de Datos (EDA):

Retornos Logarítmicos: Los precios de cierre son transformados en **retornos logarítmicos** para medir la variación de precios a lo largo del tiempo. Este análisis es crucial porque los retornos logarítmicos son más precisos para la modelización financiera.

Matriz de Correlación: Una parte importante de EDA es visualizar la **correlación** entre los activos. Esto nos ayuda a entender las relaciones entre diferentes activos y cómo pueden agruparse de acuerdo a sus comportamientos.

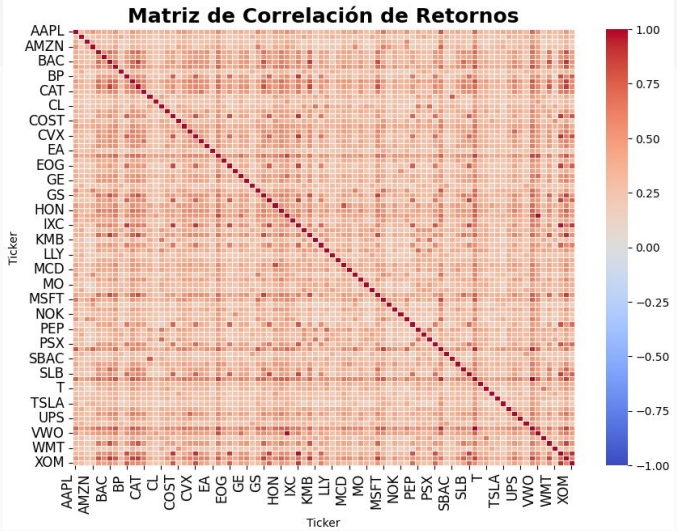
El análisis de esta matriz permite identificar activos que son altamente correlacionados y que podrían ser incluidos en el mismo grupo para diversificar el riesgo.

```
datos = yf.download(TICKERS, start="2014-01-01", end="2024-12-31")['Close']
```

```
retornos = np.log(datos / datos.shift(1)).dropna()  # Eliminar NaN resultantes del cálculo
```

```
matriz_correlacion = retornos.corr()
```

```
plt.figure(figsize=(10,7))

sns.heatmap(matriz_correlacion, annot=False, cmap='coolwarm',  # Eliminamos los valores
            vmin=-1, vmax=1,  # Limitar el rango de la barra de colores
            linewidths=0.5,  # Líneas entre celdas
            linecolor='white')  # Cambiar el color de las líneas de las celdas

plt.title("Matriz de Correlación de Retornos", fontsize=18, fontweight='bold')
plt.xticks(rotation=90, ha='right', fontsize=12)  # Aumentar el tamaño de las etiquetas de las columnas
plt.yticks(rotation=0, fontsize=12)  # Aumentar el tamaño de las etiquetas de las filas

plt.show()
```

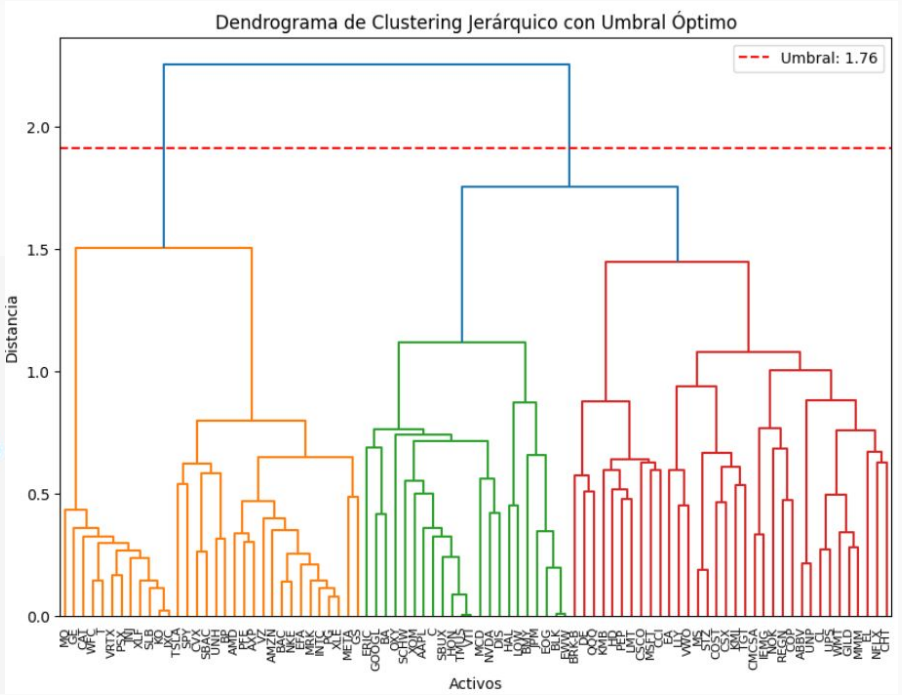


Clustering Jerárquico

```python
escalado = StandardScaler()
retornos_escalados = escalado.fit_transform(retornos.T)
```

```python
matriz_distancia = 1 - matriz_correlacion
# Convertir la matriz de distancia completa a una forma condensada
matriz_distancia_condensada = squareform(matriz_distancia)
```

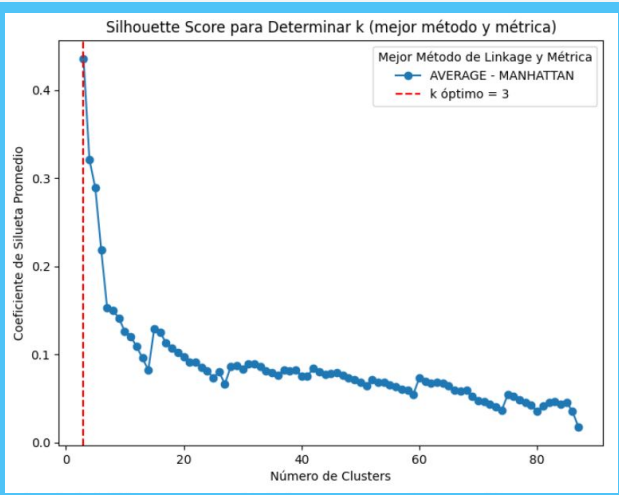
```python
Z = linkage(matriz_distancia_condensada, method='ward')

# Calcular el número óptimo de clusters con el Dendrograma (usando el umbral óptimo)
distancias = Z[:, 2]  # Extraemos las distancias
diferencia_distancias = np.diff(distancias)  # Diferencia entre distancias consecutivas
indice_umbral = np.argmax(diferencia_distancias)  # Mayor salto en la distancia
umbral_optimo = distancias[indice_umbral]  # Umbral óptimo

# Usamos fcluster para obtener los clusters según el umbral
clusters = fcluster(Z, umbral_optimo, criterion='distance')

# Obtener el número óptimo de clusters del dendrograma
k_optimo_dendograma = len(set(clusters))

# Graficar el Dendrograma con el umbral óptimo
plt.figure(figsize=(10, 7))
dendrogram(Z, labels=TICKERS, leaf_rotation=90, leaf_font_size=8)
separacion = umbral_optimo * 1.09
plt.axhline(y=separacion, color='r', linestyle='--', label=f'Umbral: {umbral_optimo:.2f}')
plt.title("Dendrograma de Clustering Jerárquico con Umbral Óptimo")
plt.xlabel("Activos")
plt.ylabel("Distancia")
plt.legend()
plt.show()

# Imprimir el número óptimo de clusters del Dendrograma
print(f"Número óptimo de clusters (Dendograma): {k_optimo_dendograma}")
print(f"Umbral Óptimo: {umbral_optimo:.2f}")
```



Evaluación de la Calidad del Clustering

```python
def evaluar_silueta(k, metodo_conexion, metrica, datos):
    try:
        # Realizar el agrupamiento y calcular el Silhouette Score
        agrupamiento = AgglomerativeClustering(n_clusters=k, linkage=metodo_conexion)
        etiquetas = agrupamiento.fit_predict(datos)
        return silhouette_score(datos, etiquetas, metric=metrica)
    except Exception as e:
        print(f"Error evaluando k={k} con {metodo_conexion} y métrica {metrica}: {e}")
        return -1  # Retorna un valor inválido si hay un error

# Inicializamos las variables de evaluación
mejor_score = -1
k_optimo_silueta = 0

# Diccionario para almacenar los resultados solo para el mejor método de linkage y métrica
scores_silueta = {}

# El rango de k va desde un valor mínimo hasta un valor máximo dinámico
min_k = 3  # El mínimo de k, puedes ajustarlo según sea necesario
max_k = len(TICKERS)  # El máximo de k será el número total de activos
metricas = ['euclidean', 'manhattan', 'cosine']
metodos = ['ward', 'complete', 'average', 'single']

# Evaluación para los diferentes valores de k, método de linkage y métricas
for k in range(min_k, max_k):  # Usamos el rango dinámico
    for metodo_conexion in metodos:
        for metrica in metricas:
            score = evaluar_silueta(k, metodo_conexion, metrica, retornos_escalados)
            if score != -1:  # Solo almacenamos resultados válidos
                if score > mejor_score:
                    mejor_score = score
                    mejor_conexion = metodo_conexion
                    mejor_metrica = metrica
                    k_optimo_silueta = k # Actualizamos el k óptimo cuando encontramos un mejor score
```

```python
# Graficar el Silhouette Score vs. Número de Clusters para el mejor método y métrica
# Sólo tomamos los valores que corresponden al mejor método y métrica
plt.figure(figsize=(8, 6))

# Evaluamos de nuevo el Silhouette Score para los valores de k con el mejor método y métrica
scores_mejor_metodo = []
for k in range(min_k, max_k):
    score = evaluar_silueta(k, mejor_conexion, mejor_metrica, retornos_escalados)
    if score != -1:
        scores_mejor_metodo.append((k, score))

# Convertimos en DataFrame para facilitar la visualización
df_scores = pd.DataFrame(scores_mejor_metodo, columns=["k", "score"])

# Graficar el resultado
plt.plot(df_scores['k'], df_scores['score'], marker='o', label=f'{mejor_conexion.upper()} - {mejor_metrica.upper()}')

# Agregar una línea vertical para el k óptimo
plt.axvline(x=k_optimo_silueta, color='red', linestyle='--', label=f'k óptimo = {k_optimo_silueta}')

# Títulos y etiquetas
plt.title("Silhouette Score para Determinar k (mejor método y métrica)")
plt.xlabel("Número de Clusters")
plt.ylabel("Coeficiente de Silueta Promedio")
plt.legend(title="Mejor Método de Linkage y Métrica")
plt.show()
```



Aplicación del Clustering Óptimo

- **Asignación de Etiquetas de Cluster:**

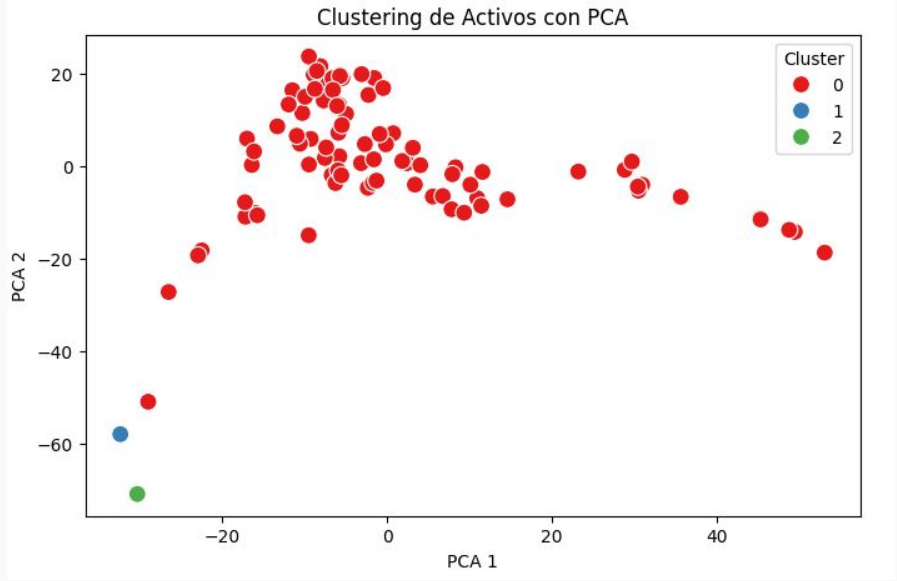
Cada acción se asigna a un cluster específico. Esto se hace usando el número de clusters óptimo determinado anteriormente.

- **Visualización con PCA:**

Se reduce la dimensionalidad con **PCA** esto ayuda a representar los clusters en un espacio 2D. Esta visualización facilita la comprensión de cómo se agrupan los activos.

```
# Aplicar Clustering Jerárquico con el número óptimo de clusters
agrupamiento_aglomerativo = AgglomerativeClustering(n_clusters=k_optimo_silueta, linkage=mejor_conexion)
etiquetas = agrupamiento_aglomerativo.fit_predict(retornos_escalados)
```

```
# Visualizar Clusters con PCA
pca = PCA(n_components=2)
resultado_pca = pca.fit_transform(retornos_escalados)

plt.figure(figsize=(8, 5))
sns.scatterplot(x=resultado_pca[:, 0], y=resultado_pca[:, 1], hue=etiquetas, palette='Set1', s=100)
plt.title("Clustering de Activos con PCA")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.legend(title="Cluster")
plt.show()
```



Selección de Activos Representativos

```python
# Seleccionar un Activo Representante de cada Cluster basándonos en el retorno promedio
df_cluster = pd.DataFrame({'Ticker': TICKERS, 'Cluster': etiquetas})

tasa_libre_riesgo = 0.03 / 252  # Convertido a retornos diarios

def seleccionar_activo_optimo(activos_en_cluster):
    activos_validos = set(activos_en_cluster) & set(retornos.columns)  # Intersección entre ambos conjuntos
    if not activos_validos:
        return None

    # Convertir la intersección de vuelta a una lista
    activos_validos = list(activos_validos)

    retornos_promedio = retornos[activos_validos].mean()
    volatilidades = retornos[activos_validos].std() + 1e-8  # Se suma un valor pequeño para evitar división por cero
    razones_sharpe = (retornos_promedio - tasa_libre_riesgo) / volatilidades

    return razones_sharpe.idxmax()  # Retornar el ticker con la mejor razón de Sharpe

# Seleccionamos los activos óptimos de cada cluster
activos_seleccionados = df_cluster.groupby('Cluster')['Ticker'].apply(seleccionar_activo_optimo).dropna().values
```

```
Activos Seleccionados para la Optimización de Portafolio: ['NVDA', 'ERIC', 'GOOGL']
```

Índice de Sharpe:

Selección de activos: Dentro de cada cluster, seleccionamos el activo con el mejor **Índice de Sharpe**, ya que esto representa el activo que ha logrado el mejor rendimiento ajustado por riesgo.

El activo con mayor Sharpe es el que ha demostrado la mejor relación entre rentabilidad y volatilidad, lo que lo convierte en el más atractivo para el portafolio.

Optimización del Portafolio

```python
# Obtener los retornos y la matriz de covarianza de los activos seleccionados
retornos_seleccionados = retornos[activos_seleccionados]
rendimientos_promedio = retornos_seleccionados.mean()
matriz_covarianza = retornos_seleccionados.cov()

# Función de optimización para encontrar la mejor combinación de activos
def ratio_sharpe_regularizado(pesos, retornos_promedio, matriz_covarianza, lambda_ridge=0.05):
    retorno = np.sum(pesos * retornos_promedio)  # Retorno esperado del portafolio
    volatilidad = np.sqrt(np.dot(pesos.T, np.dot(matriz_covarianza, pesos)))  # Volatilidad del portafolio
    regularizacion = lambda_ridge * np.sum(pesos**2)  # Término de regularización (Ridge)
    return -((retorno - tasa_libre_riesgo) / volatilidad) + regularizacion  # Minimización del Sharpe Ratio regularizado

# Optimización de la frontera eficiente
# Parámetros necesarios
num_activos = len(activos_seleccionados)
restricciones = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})  # Restricción de que la suma de los pesos sea 1
volatilidades = np.sqrt(np.diag(matriz_covarianza))  # Volatilidad de cada activo
limites = tuple((0.01, min(0.50, 1 / vol)) for vol in volatilidades)  # Limitar pesos entre 0.01 y 0.50 o 1/volatilidad

# Pesos iniciales aleatorios
pesos_iniciales = np.random.dirichlet(np.ones(num_activos), size=1)[0]

# Realizar la optimización
resultado_optimizacion = minimize(ratio_sharpe_regularizado, pesos_iniciales, args=(rendimientos_promedio, matriz_covarianza
                         method='SLSQP', bounds=limites, constraints=restricciones)
pesos_optimos = resultado_optimizacion.x  # Pesos óptimos encontrados
```

Regularización Ridge: La regularización se emplea para evitar la sobreajuste, penalizando grandes valores en los pesos del portafolio. Esto garantiza que el modelo no asigne demasiado peso a activos que podrían ser demasiado volátiles o poco representativos.

Optimización: Utilizando técnicas como **`scipy.optimize`**, ajustamos los pesos de los activos para encontrar la combinación que maximiza el índice de Sharpe.

Modelo de Markowitz:

La **optimización de Markowitz** se centra en crear un portafolio eficiente en el que el **riesgo** (volatilidad) sea minimizado para un nivel dado de **retorno** esperado, o el retorno se maximice para un nivel dado de riesgo.

Frontera eficiente: Todos los portafolios que se encuentran en esta frontera representan combinaciones óptimas de riesgo y retorno.

Retorno Esperado y Volatilidad:

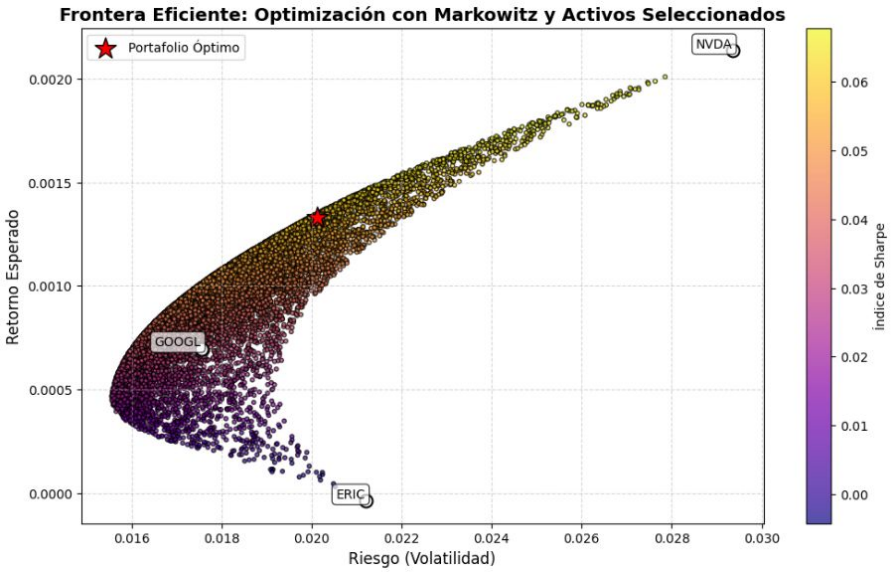
Calculamos el retorno y la volatilidad de cada posible portafolio utilizando los rendimientos históricos de los activos seleccionados, ponderados según los pesos asignados.

Generación de la Frontera Eficiente:

Realizamos diferentes combinaciones aleatorias de pesos y calculamos su rendimiento y riesgo, lo que nos permite construir una frontera eficiente.

```
# Cálculo del retorno y riesgo del portafolio óptimo
retorno_portafolio = np.sum(pesos_optimos * rendimientos_promedio)  # Retorno esperado del portafolio
volatilidad_portafolio = np.sqrt(np.dot(pesos_optimos.T, np.dot(matriz_covarianza, pesos_optimos)))
sharpe_optimo = (retorno_portafolio - tasa_libre_riesgo) / (volatilidad_portafolio + 1e-8)

# Generación de portafolios aleatorios para la frontera eficiente
num_portafolios = 10000  # Número de portafolios a generar
resultados = np.zeros((3, num_portafolios))  # Retorno, Riesgo, Sharpe Ratio
pesos_registro = np.zeros((num_portafolios, num_activos))  # Registro de pesos de cada portafolio

for idx_portafolio in range(num_portafolios):
    pesos = np.random.uniform(0.01, 0.50, num_activos)  # Pesos entre 0.01 y 0.50 para cada activo
    pesos /= np.sum(pesos)  # Normalizamos para que sumen 1 (portafolio completamente invertido)

# Guardamos los pesos generados para este portafolio
pesos_registro[idx_portafolio, :] = pesos
# Cálculo del retorno esperado del portafolio
retorno = np.dot(pesos, rendimientos_promedio)  # Usamos np.dot para una multiplicación eficiente
# Cálculo de la volatilidad (riesgo) del portafolio usando la matriz de covarianza
volatilidad = np.sqrt(np.dot(pesos.T, np.dot(matriz_covarianza, pesos)))  # Calculamos la volatilidad
# Cálculo del índice de Sharpe para este portafolio (ajustado por tasa libre de riesgo)
sharpe = (retorno - tasa_libre_riesgo) / volatilidad if volatilidad != 0 else 0
# Guardamos los resultados del portafolio: retorno, volatilidad, Sharpe
resultados[:, idx_portafolio] = [retorno, volatilidad, sharpe]
```

Cálculo del Retorno y Riesgo del Portafolio




```
Pesos del portafolio óptimo basado en clustering jerárquico y optimización de Markowitz:
NVDA: 50.00%
ERIC: 11.95%
GOOGL: 38.05%
Retorno Esperado del Portafolio Óptimo: 0.13%
Volatilidad del Portafolio Óptimo: 2.01%
Índice Sharpe del Portafolio Óptimo: 0.0601
```

```
Rendimiento ponderado del portafolio: -0.16%
Precio acumulado del portafolio en el último período: 2247.60
```

Resultados Finales

1. Número Óptimo de Clústeres (Dendrograma):

- Número óptimo de clústeres: **2**
- Identificación basada en la visualización del dendrograma.

2. Umbral Óptimo:

- **Umbral de corte: 1.76**
- Marca el punto de corte en la jerarquía de agrupamiento.

3. Mejor Método de Enlace:

- Método: **Promedio**
- Métrica: **Manhattan**
- **Silhouette Score**: **0.44** (calidad moderada del agrupamiento).

Activo	Peso (%)
NVDA (NVIDIA)	**50.00**
ERIC (Ericsson)	**11.87**
GOOGL (Google)	**38.13**

Métrica	Valor
Rendimiento Esperado (Retorno modesto)	**0.13%**
Volatilidad (Riesgo bajo)	**2.01%**
Índice de Sharpe (Baja relación retorno-riesgo)	**0.0601**
Rendimiento Ponderado (Pequeña pérdida)	**-0.16%**

7. Valor Final del Portafolio:

- **Valor Acumulado del Portafolio: 2245.67**
- Crecimiento positivo desde el valor inicial de **100**.

Conclusiones

Aunque el modelo de optimización basado en clustering jerárquico y Markowitz ya ofrece una solución sólida para la creación de portafolios eficientes, existen áreas clave de mejora. La combinación de nuevas técnicas de clustering, optimización multiobjetivo, incorporación de datos adicionales y simulaciones avanzadas puede incrementar aún más la eficiencia del modelo y mejorar la toma de decisiones en escenarios más complejos. Implementando estas mejoras, es posible optimizar el retorno ajustado por riesgo de manera más precisa y adaptada a un entorno financiero cambiante.