<!--Herramientas de Programación-->

Proyecto Final {

<Por="Josy Nocua,Luis Camilo Araujo, Camilo Giraldo, Alejandro Rodríguez"/>

}

Contenidos

Introducción {

Este proyecto desarrolla una herramienta para optimizar portafolios de inversión. De este modo, mediante el modelo Black-Litterman para la asignación de activos y simulaciones de Montecarlo para evaluar riesgos futuros, nuestro objetivo es ayudar a construir carteras más eficientes y tomar decisiones de inversión fundamentadas, de modo que se maximizen retornos y gestionen el riesgo de forma inteligente.

}

Librerías {

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, widgets
from IPython.display import clear_output
from scipy.optimize import minimize
import warnings
warnings.filterwarnings("ignore")
```

Las librerías proveen las herramientas clave: yfinance para datos de acciones, pandas y numpy para manejo y cálculo, matplotlib para visualización, ipywidgets para interactividad, y scipy.optimize para la optimización del portafolio.

}

Carga de datos {

```python
def cargar_datos_global(tickers):
    print(f"\nCargando {len(tickers)} tickers desde Yahoo Finance...")
    tickers_obj = yf.Tickers(' '.join(tickers))

    # Como en tu imagen: group_by='ticker'
    data = yf.download(tickers, period="5y", auto_adjust=True, group_by='ticker', progress=False)

    precios_dict = {}
    for t in tickers:
        if t in data and 'Close' in data[t].columns:
            precios_dict[t] = data[t]['Close'].dropna()
        else:
            print(f"{t}: No hay datos. Se omitirá.")

    precios_df = pd.DataFrame(precios_dict).dropna(how='all')
    tickers_validos = precios_df.columns.tolist()
    print(f"Tickers válidos: {tickers_validos}")
    return tickers_obj, precios_df, tickers_validos
```

Esta sección descarga precios históricos de acciones de Yahoo Finance para los últimos 5 años, asegurando la selección de datos válidos y completos para un análisis robusto del portafolio, y se centra en los precios de cierre para cada activo.

}

Clase Activo {

```python
class Activo:
    def __init__(self, ticker):
        self.ticker = ticker.upper()
        self.precios = pd.Series()
        self.retornos = pd.Series()
        self.retorno_esperado = 0.0
        self.riesgo = 0.0
        self.beta = 0.0
        self.var_95 = 0.0
```

```python
def cargar(self, tickers_obj, precios_df):
    try:
        if self.ticker not in precios_df.columns:
            raise ValueError("No en DataFrame")
        self.precios = precios_df[self.ticker].dropna()
        if len(self.precios) < 100:
            raise ValueError("Datos insuficientes")
        self.retornos = self.precios.pct_change().dropna()
        self.retorno_esperado = self.retornos.mean() * 252
        self.riesgo = self.retornos.std() * np.sqrt(252)
        self.beta = tickers_obj.tickers[self.ticker].info.get('beta', 1.0)
        self.var_95 = -np.percentile(self.retornos, 5) * np.sqrt(252)
        print(f"{self.ticker}: OK | Ret: {self.retorno_esperado:.1%} | Beta: {self.beta:.2f} | VaR: {self.var_95:.1%}")
    except Exception as e:
        print(f"{self.ticker}: ERROR → {e}")
        self.retornos = pd.Series()
```

La clase Activo modela cada acción individual, calculando sus métricas financieras clave (retorno esperado, riesgo, Beta, VaR) anualmente a partir de sus precios de cierre, validando los datos para su uso en el portafolio.

}

CLASE PORTAFOLIO {

```python
class Portafolio:
    def __init__(self, tickers, tasa_libre_riesgo=0.01):
        self.tickers_input = [t.upper() for t in tickers if t.strip()]
        self.tasa_libre_riesgo = tasa_libre_riesgo
        self.tickers_obj, self.precios_df, self.tickers = cargar_datos_global(self.tickers_input)
        self.activos = []
        for t in self.tickers:
            a = Activo(t)
            a.cargar(self.tickers_obj, self.precios_df)
            if len(a.retornos) > 0:
                self.activos.append(a)
        self.tickers = [a.ticker for a in self.activos]
        self.pesos = np.ones(len(self.activos)) / len(self.activos) if self.activos else None
        self.cov_matrix = None
        self.mu_bl = None
        self.retorno_opt = self.riesgo_opt = self.sharpe_opt = 0.0
        self.backtest = pd.Series()
        self.frontera_eficiente = pd.DataFrame() # Renamed to avoid conflict

    def calcular_cov(self):
        if len(self.activos) < 2: return False
        df = pd.concat([a.retornos for a in self.activos], axis=1).dropna()
        df.columns = self.tickers
        self.cov_matrix = df.cov() * 252
        return True
```

Propósito y Construcción del Portafolio:

- Gestor Central: Organiza y optimiza un portafolio de múltiples Activo.
- Inicialización: Recibe tickers y una tasa_libre_riesgo.
- Carga de Datos: Procesa datos de mercado para cada Activo.
- Matriz de Covarianza: Calcula cómo los activos se mueven entre sí para el riesgo.
- Pesos Iniciales: Define la distribución inicial de la inversión.

continuación...

Optimización y Análisis Avanzado:

- Black-Litterman: Ajusta retornos esperados, mejorando la robustez.
- Optimización Sharpe: Encuentra pesos que maximicen el Ratio de Sharpe (retorno/riesgo).
- Frontera Eficiente: Simula portafolios para visualizar opciones riesgo-retorno eficientes.
- Resumen Detallado: Presenta métricas clave del portafolio óptimo.

```python
def black_litterman(self):
    if not self.calcular_cov(): return
    n = len(self.activos)
    P = np.eye(n)
    Q = np.array([a.retorno_esperado * 0.8 for a in self.activos])
    Omega = np.diag([0.01] * n)
    w_mkt = np.ones(n) / n
    pi = 2.5 * self.cov_matrix @ w_mkt
    tau = 0.05
    try:
        inv_tau = np.linalg.inv(tau * self.cov_matrix)
        A = inv_tau + P.T @ np.linalg.inv(Omega) @ P
        b = inv_tau @ pi + P.T @ np.linalg.inv(Omega) @ Q
        self.mu_bl = np.linalg.solve(A, b)
    except:
        self.mu_bl = np.array([a.retorno_esperado for a in self.activos])

def optimizar(self):
    if self.mu_bl is None: return
    n = len(self.activos)
    def neg_sharpe(w):
        ret = w @ self.mu_bl
        risk = np.sqrt(w.T @ self.cov_matrix @ w)
        return -(ret - 0.01) / risk if risk > 0 else 1e9
    cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1},)
    bounds = [(0, 1) for _ in range(n)]
    res = minimize(neg_sharpe, np.ones(n)/n, method='SLSQP', bounds=bounds, constraints=cons)
    self.pesos = res.x if res.success else np.ones(n)/n
    self.retorno_opt = self.pesos @ self.mu_bl
    self.riesgo_opt = np.sqrt(self.pesos.T @ self.cov_matrix @ self.pesos)
    self.sharpe_opt = (self.retorno_opt - 0.01) / self.riesgo_opt
    df = pd.concat([a.retornos for a in self.activos], axis=1).dropna()
    self.backtest = (df @ self.pesos).dropna()
```

}

continuación...

```python
def generar_frontera(self, n=100): # Renamed method
    if self.mu_bl is None: return
    res = []
    for _ in range(n):
        w = np.random.random(len(self.activos)); w /= w.sum()
        ret = w @ self.mu_bl
        risk = np.sqrt(w.T @ self.cov_matrix @ w)
        if risk > 0:
            res.append([ret, risk, (ret-0.01)/risk])
    self.frontera_eficiente = pd.DataFrame(res, columns=['Retorno', 'Riesgo', 'Sharpe']) # Assign to correct attribute


def resumen(self):
    print(f"\n{'='*60}")
    print(f"PORTAFOLIO ÓPTIMO ({len(self.tickers)} activos)")
    print(f"Retorno: {self.retorno_opt:.4f} ({self.retorno_opt*100:+.2f}%)")
    print(f"Riesgo:  {self.riesgo_opt:.4f} ({self.riesgo_opt*100:.2f}%)")
    print(f"Sharpe:  {self.sharpe_opt:.4f}")
    print(f"Backtest: {self.backtest.mean()*252:.4f}")
    print(f"VaR 95% promedio: {np.mean([a.var_95 for a in self.activos]):.1%}")
    print(f"\nPESOS ÓPTIMOS:")
    for t, w in zip(self.tickers, self.pesos):
        print(f"  {t}: {w:.4f} ({w*100:5.1f}%)")
    print(f"{'='*60}\n")
```

- Optimización Estratégica: La clase aplica el modelo Black-Litterman para ajustar las expectativas de retorno y optimiza los pesos de los activos para maximizar el Ratio de Sharpe, buscando el mejor balance riesgo-retorno.
- Visualización de Eficiencia: Utiliza generar_frontera para simular y trazar la frontera eficiente, permitiendo una visualización clara de las opciones de inversión óptimas en términos de riesgo y retorno.
- Resumen Conclusivo: Finalmente, consolida y presenta un resumen detallado del portafolio óptimo, incluyendo sus métricas clave y los pesos finales, para una toma de decisiones informada.

}

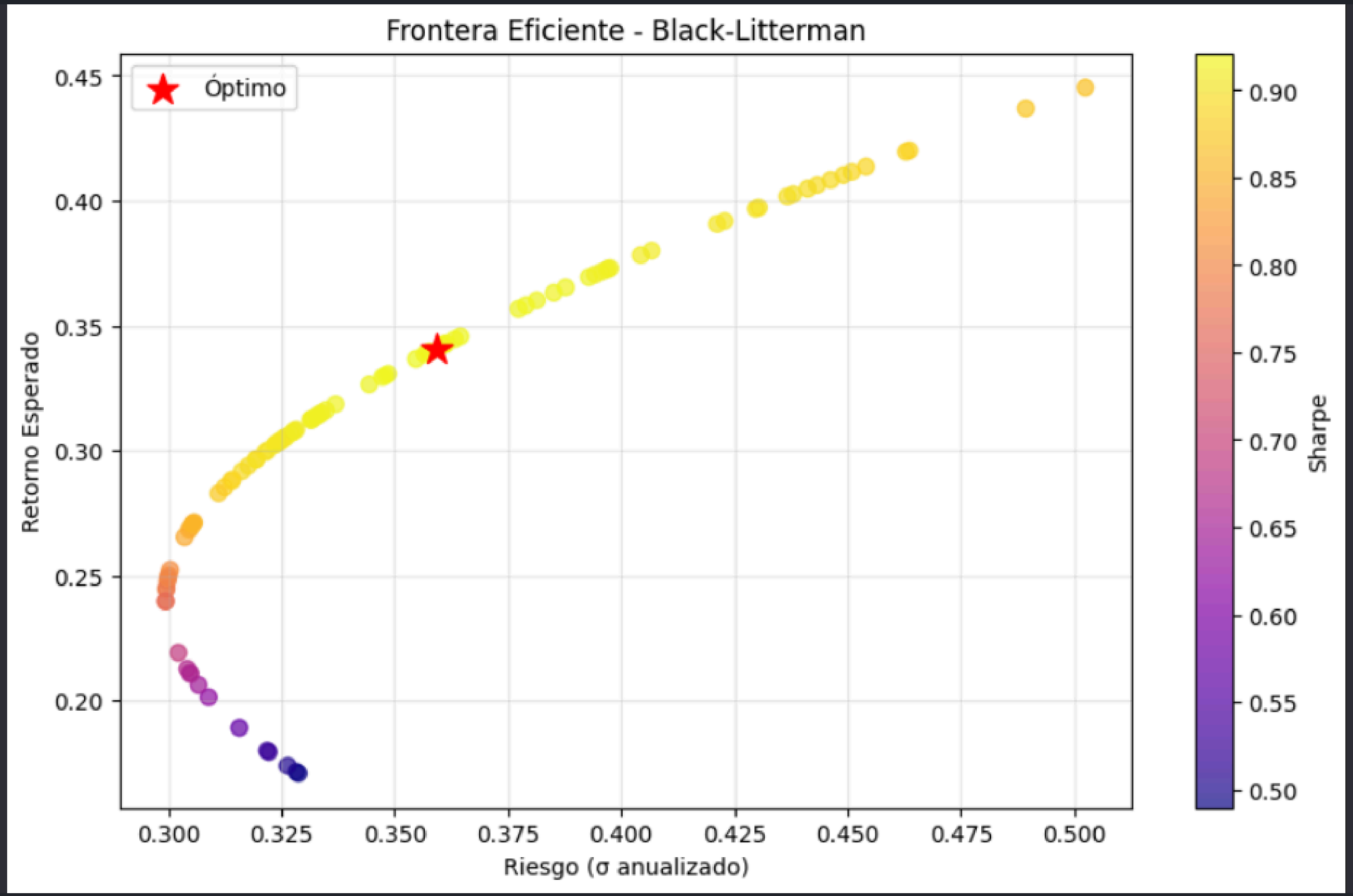
Ejecución{

```
def obtener_tickers():
    print("Ingresa tickers (máx 30, 'done' para terminar):")
    t = []
    while len(t) < 30:
        x = input(f"[{len(t)+1}/30]: ").strip().upper()
        if x == 'DONE': break
        if x and x not in t: t.append(x)
    return t or ['NVDA', 'BABA', 'OXY']

# === EJECUTAR TODO ===
tickers = obtener_tickers()
port = Portafolio(tickers)
port.black_litterman()
port.optimizar()
port.generar_frontera() # Call the renamed method
port.resumen()
graficar(port)
metricas(port)
```

La sección 'Ejecución' organiza el proceso: obtiene tickers, construye el Portafolio, aplica Black-Litterman y optimización. Finalmente, usa generar_frontera y graficar para calcular y visualizar la Frontera Eficiente, mostrando el resultado clave del análisis.

}

Gráfico de frontera eficiente



}

Gráficas {

- Frontera Eficiente (graficar): Genera un gráfico interactivo (matplotlib) de la Frontera Eficiente, resaltando el portafolio óptimo por su Ratio de Sharpe y permitiendo ajustar el tamaño de los puntos.
- Métricas del Portafolio (metricas): Muestra un resumen interactivo de las métricas clave del portafolio óptimo (Retorno, Riesgo, Sharpe), con una casilla para ver los detalles individuales de cada activo (Retorno, Beta, VaR).

```python
def graficar(port):
    if port.frontera_eficiente.empty: # Check correct attribute
        print("No hay frontera eficiente.")
        return
    def plot(tam=50):
        clear_output(wait=True)
        plt.figure(figsize=(10,6))
        s = plt.scatter(port.frontera_eficiente['Riesgo'], port.frontera_eficiente['Retorno'], # Use correct attribute
                        c=port.frontera_eficiente['Sharpe'], cmap='plasma', s=tam, alpha=0.7) # Use correct attribute
        plt.scatter(port.riesgo_opt, port.retorno_opt, color='red', s=200, marker='*', label='Óptimo')
        plt.xlabel('Riesgo (σ anualizado)')
        plt.ylabel('Retorno Esperado')
        plt.title('Frontera Eficiente - Black-Litterman')
        plt.colorbar(s, label='Sharpe')
        plt.legend()
        plt.grid(True, alpha=0.3)
        plt.show()
    interact(plot, tam=widgets.IntSlider(min=20, max=100, step=10, value=50, description='Tamaño:'))

def metricas(port):
    def show(det=False):
        clear_output(wait=True)
        print(f"Ret: {port.retorno_opt:.2%} | Riesgo: {port.riesgo_opt:.2%} | Sharpe: {port.sharpe_opt:.3f}")
        if det:
            for a in port.activos:
                print(f"  {a.ticker}: Ret {a.retorno_esperado:.1%}, Beta {a.beta:.2f}, VaR {a.var_95:.1%}")
    interact(show, det=widgets.Checkbox(description='Detalles'))
```

}

Gráfico pastel

```python
# FRAGMENTO 5: Gráfico de pastel profesional
def grafico_pesos_profesional(port, titulo="COMPOSICIÓN DEL PORTAFOLIO ÓPTIMO"):
    """
    Gráfico de donut profesional:
    - Etiquetas afuera con líneas
    - Leyenda con colores y porcentajes
    - Título centrado arriba
    - Colores suaves y profesionales
    - Exporta PNG y PDF automáticamente
    """
    if port.pesos is None or len(port.pesos) == 0:
        print("No hay pesos para mostrar.")
        return
```

Crear un gráfico de dona estético y profesional que muestre cómo se distribuye la inversión (los "pesos") entre los diferentes activos del portafolio.

}

Cargar Datos{

```python
# Datos
tickers = port.tickers
pesos = port.pesos
porcentajes = [f"{w*100:.1f}%" for w in pesos]
etiquetas_con_porcentaje = [f"{t} {p}" for t, p in zip(tickers, porcentajes)]
```

Se extraen dos listas de datos fundamentales del objeto port (el portafolio). y usa TIKERS como simbolos bursatiles y pesos con las proporciones del activo.

}

Base y estética técnica {

```python
# Colores profesionales (como en tu imagen: suaves y distintos)
colores = [
    '#4C72B0',  # Azul (NVDA)
    '#55A868',  # Verde (OXY)
    '#C44E52',  # Rojo (JPM)
    '#8172B2',  # Morado (RBLX)
    '#CCB974',  # Amarillo (IBIT)
    '#64B5CD',  # Cian (C)
    '#8C564B',  # Marrón (KO)
][:len(pesos)]  # Ajustar si hay más o menos activos

# Crear figura
fig, ax = plt.subplots(figsize=(11, 9), facecolor='white')
```

- Datos Claros: Convierte pesos decimales en etiquetas legibles (Ticker + Porcentaje a 1 decimal: "AAPL 30.0%").
- Estilo Adaptable: Usa una paleta de colores profesionales que se ajusta dinámicamente al tamaño del portafolio.
- Alta Definición (HD): Crea el lienzo gráfico en gran formato ((11, 9)) para asegurar la máxima calidad y nitidez en la exportación final a PNG y PDF.

}

Gráfico de anillo {

```
# Donut chart
wedges, texts = ax.pie(
    pesos,
    labels=None,
    startangle=90,
    colors=colores,
    wedgeprops=dict(width=0.35, edgecolor='white', linewidth=2.5),
    counterclock=False
)

# Etiquetas afuera con líneas (como en tu imagen)
for i, (wedge, label) in enumerate(zip(wedges, etiquetas_con_porcentaje)
    ang1 = wedge.theta1
    ang2 = wedge.theta2
```

Este código crea un Donut Chart, calculando con precisión la ubicación de cada etiqueta de texto fuera del anillo usando el ángulo central de cada porción.

}

dibuja la línea de conexión y coloca la etiqueta de texto final{

```
# Línea
ax.plot([0.8 * np.cos(ang * np.pi / 180), x],
        [0.8 * np.sin(ang * np.pi / 180), y],
        color='gray', linewidth=1.2, alpha=0.8)

# Etiqueta
ha = 'left' if x > 0 else 'right'
ax.text(x, y, label, ha=ha, va='center', fontsize=11, fontweight='bold', color='black')
```

Se dibuja una línea de conexión desde el borde del anillo hasta un punto exterior, y se coloca una etiqueta de texto que se alinea automáticamente (izquierda o derecha) para proyectarse hacia afuera del gráfico.

}

Claridad y presentación profesional {

La sección de configuración de la leyenda posiciona la leyenda a la derecha del gráfico con el título "Activos". Se personaliza el marco con fondo blanco y borde gris para mejorar la legibilidad de las etiquetas con porcentajes, separándolas del área de trazado.

```
# Leyenda (como en tu imagen: a la derecha, con co
legend = ax.legend(
    wedges,
    etiquetas_con_porcentaje,
    title="Activos",
    loc="center left",
    bbox_to_anchor=(1.05, 0.5),
    fontsize=11,
    title_fontsize=13,
    frameon=True,
    fancybox=True,
    shadow=False,
    labelspacing=1.2
)
legend.get_frame().set_facecolor('white')
legend.get_frame().set_edgecolor('gray')
```

}

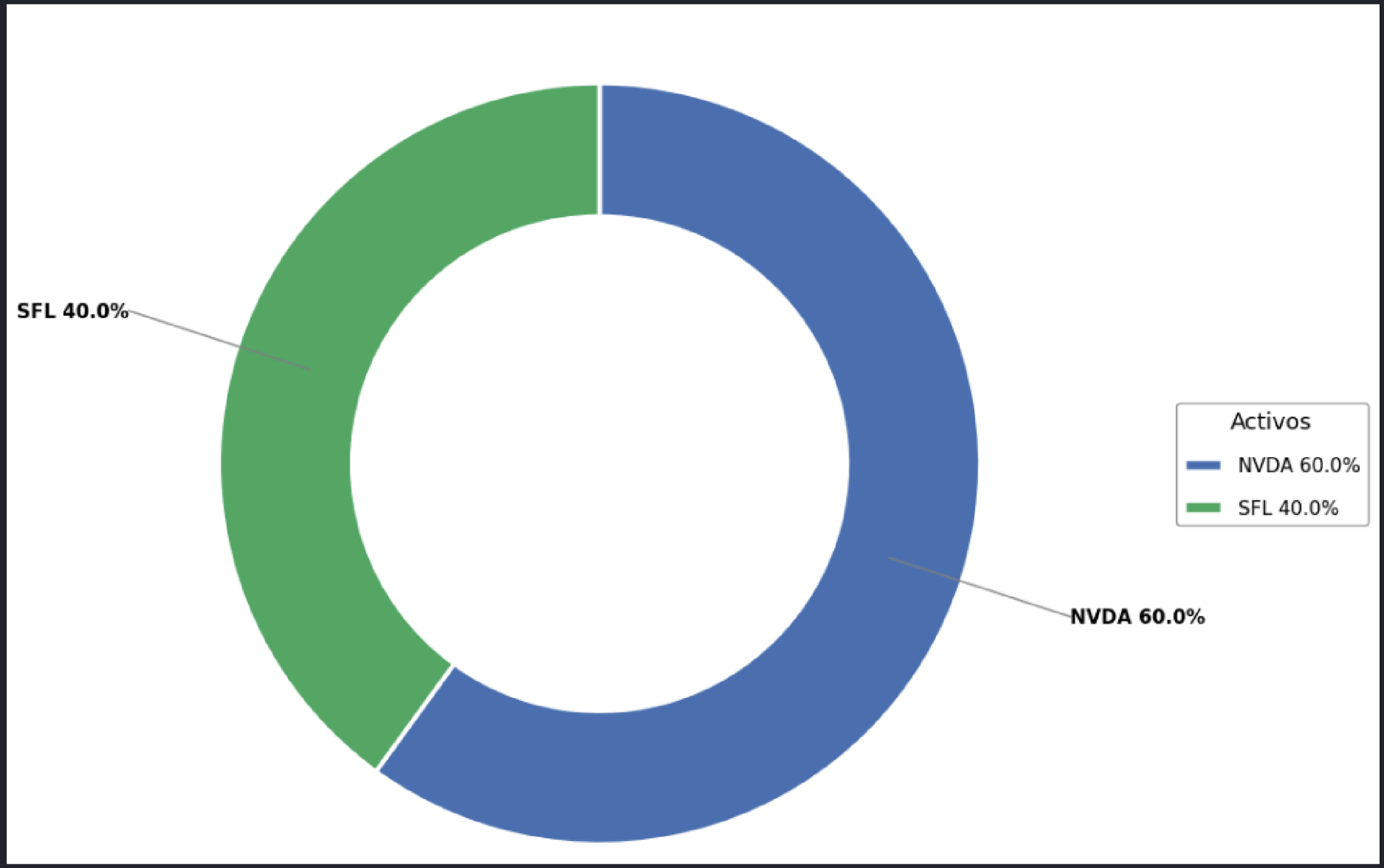
Finalización y exportación {

```python
# Título centrado arriba
plt.title(titulo, fontsize=18, fontweight='bold', pad=40, loc='center')

# Fondo y estilo
ax.axis('equal')
plt.tight_layout()

# EXPORTAR AUTOMÁTICO
plt.savefig('grafico_pesos.png', dpi=300, bbox_inches='tight', facecolor='white')
plt.savefig('grafico_pesos.pdf', bbox_inches='tight', facecolor='white')
print("Gráfico guardado: grafico_pesos.png y grafico_pesos.pdf")

plt.show()
```

Finalización y Exportación: Se añaden los últimos detalles de diseño, se exporta automáticamente en alta resolución como PNG y como PDF, y se muestra en pantalla.

}

Composición del portafolio óptimo



}

Función de Montecarlo {

```
=== AÑADIR ESTO AL FINAL DE TU taller_final.py ===
ef montecarlo_simulacion(port, simulaciones=10000, horizonte=1)
    """
    Simulación Montecarlo del portafolio óptimo
    - Usa retornos normales: μ_BL, Σ
    - Horizonte en años
    """
    if port.mu_bl is None or port.cov_matrix is None:
        print("Faltan μ_BL o Σ para Montecarlo.")
        return

    mu_anual = port.mu_bl
    sigma_anual = port.cov_matrix
    w = port.pesos
```

La función inicia una Simulación de Montecarlo para tu portafolio óptimo, proyectando 10,000 posibles resultados de inversión en un tiempo específico. Utiliza retornos esperados ajustados y una matriz de covarianza para modelar la volatilidad y la interdependencia entre activos, ofreciendo una visión probabilística del riesgo más allá de un cálculo teórico.

}

Función de Montecarlo {

```
ret_port = w @ mu_anual
risk_port = np.sqrt(w.T @ sigma_anual @ w)

# Simulación
ret_sim = np.random.normal(ret_port, risk_port, simulaciones)
```

Cálculo y Modelado de Escenarios: El código calcula el Retorno y Riesgo Anualizado del portafolio usando fórmulas matriciales. Con estos valores como media (μ) y desviación estándar (σ)) , se generan 10,000 retornos anuales aleatorios (ret_sim) bajo la suposición de una distribución normal.

Función de Montecarlo {

```
plt.figure(figsize=(10, 6))
plt.hist(ret_sim, bins=50, alpha=0.7, color='skyblue', edgecolor='black')
plt.axvline(ret_port, color='red', linestyle='--', linewidth=2, label=f'Retorno Óptimo: {ret_p
plt.axvline(np.percentile(ret_sim, 5), color='orange', linestyle='--', linewidth=2, label='VaR
plt.title('SIMULACIÓN MONTECARLO - DISTRIBUCIÓN DE RETORNOS', fontsize=14, fontweight='bold')
plt.xlabel('Retorno Anualizado')
plt.ylabel('Frecuencia')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Guardar
plt.savefig('montecarlo.png', dpi=300, bbox_inches='tight')
plt.savefig('montecarlo.pdf', bbox_inches='tight')
plt.show()
```

Cálculo y Modelado de Escenarios: El código calcula el Retorno y Riesgo Anualizado del portafolio usando fórmulas matriciales. Con estos valores como media (μ) y desviación estándar (σ)) , se generan 10,000 retornos anuales aleatorios (ret_sim) bajo la suposición de una distribución normal.

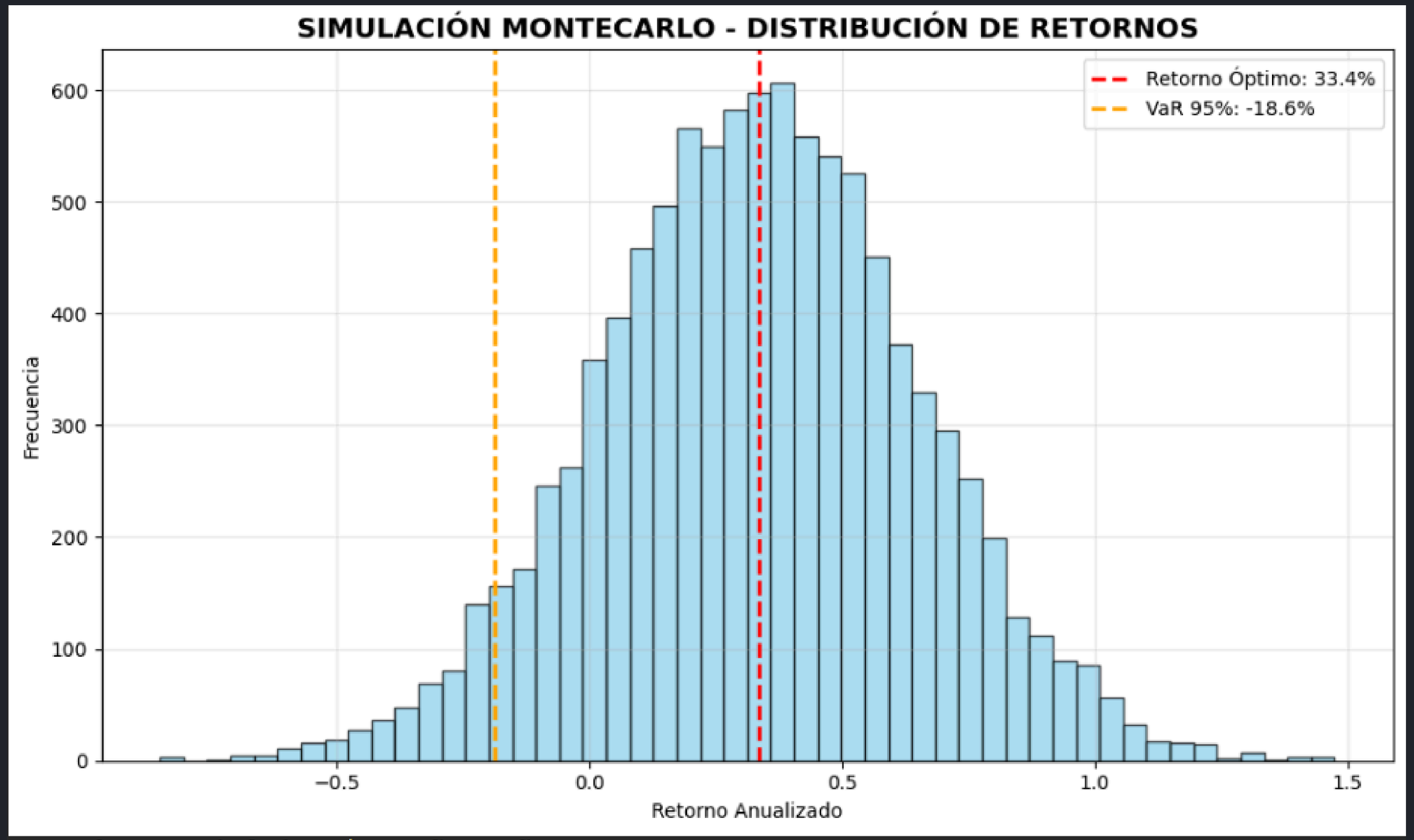
Análisis de resultados {

```
# Gráfico
plt.figure(figsize=(10, 6))
plt.hist(ret_sim, bins=50, alpha=0.7, color='skyblue', edgecolor='black')
plt.axvline(ret_port, color='red', linestyle='--', linewidth=2, label=f'Retorno Óptimo: {ret_por
plt.axvline(np.percentile(ret_sim, 5), color='orange', linestyle='--', linewidth=2, label='VaR 9
plt.title('SIMULACIÓN MONTECARLO - DISTRIBUCIÓN DE RETORNOS', fontsize=14, fontweight='bold')
plt.xlabel('Retorno Anualizado')
plt.ylabel('Frecuencia')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Guardar
plt.savefig('montecarlo.png', dpi=300, bbox_inches='tight')
plt.savefig('montecarlo.pdf', bbox_inches='tight')
plt.show()
```

Para concluir el Análisis de Resultados de Montecarlo (VaR), este código presenta una visualización de los 10,000 retornos simulados a través de un histograma, que ilustra la distribución de los resultados. El análisis se centra en dos líneas fundamentales: el Retorno Óptimo, representado por la línea roja, y, de manera crucial, el Valor en Riesgo (VaR) al 95%, indicado por la línea naranja. Este último señala la pérdida máxima esperada con un 95% de confianza.

}

Simulación MONTECARLO



Montecarlo: 10000 simulaciones | VaR 95%: -18.6%

}

Gracias {

```
<Por="Josy Nocua,Luis Camilo Araujo, Camilo Giraldo, Alejandro Rodríguez"/>
```

}