UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática

Desarrollo de un sistema para la gestión de carteras de activos de alta volatilidad: Aplicación a criptomonedas

Development of a System for High Volatility Asset Portfolio Management: Application to Cryptocurrencies

Realizado por
Alejandro Rodríguez Moreno

Tutorizado por
Francisco de Asís Fernández Navarro

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Graduado en Ingeniería Informática

Desarrollo de un sistema para la gestión de carteras de activos de alta volatilidad: Aplicación a criptomonedas

Development of a System for High Volatility Asset Portfolio Management: Application to Cryptocurrencies

Realizado por
Alejandro Rodríguez Moreno

Tutorizado por
Francisco de Asís Fernández Navarro

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2025

Abstract

Portfolio management has traditionally been applied to established assets, where stability and low volatility are assumed. However, the role of more volatile assets, such as cryptocurrencies, has introduced a new investment paradigm. These assets offer a much greater profit opportunity in exchange for higher risk.

Although there are previous approaches to portfolio optimization, these have primarily been studied and applied to traditional portfolios. This work aims to test various computable models adapted to a more dynamic and volatile environment, such as that of cryptocurrencies. Additionally, a new perspective is introduced through the application of neural networks to predict asset movements. All the proposed models will be thoroughly evaluated, analyzing their advantages and disadvantages, with the goal of providing a framework that facilitates the development and further exploration of future research by the community.

Keywords: Portfolio management, cryptocurrencies, optimization, neural networks.

Resumen

La gestión de carteras ha sido tradicionalmente aplicada a activos consolidados, en los que se asume una estabilidad y una baja volatilidad. No obstante, el papel de activos con más volatilidad como lo son las criptomonedas ha introducido un nuevo paradigma de inversión. Estos activos ofrecen una oportunidad de beneficio mucho mayor a cambio de un riesgo elevado.

A pesar de que existen enfoques previos en la optimización de carteras, estos han sido mayoritariamente estudiados y aplicados en carteras tradicionales. En este trabajo, se propone poner a prueba diversos modelos computables adaptados a un entorno más dinámico y volátil, como el de las criptomonedas. Además, se introduce una nueva perspectiva mediante la aplicación de redes neuronales para la predicción de los movimientos de activos. Todos los modelos propuestos serán evaluados de manera exhaustiva, analizando sus ventajas y desventajas, con el objetivo de proporcionar un marco de estudio que facilite el desarrollo y la profundización de futuras investigaciones por parte de la comunidad.

Palabras clave: Gestión de carteras, criptomonedas, optimización, redes neuronales.

Índice

1 Introducción

1.1. Motivación

La gestión de carteras ha sido un área central en el ámbito financiero durante décadas, siendo tradicionalmente aplicada a activos consolidados y estables, donde la volatilidad se encuentra dentro de niveles previsibles y relativamente bajos. Sin embargo, con la aparición de las criptomonedas y otros activos digitales, el panorama de las inversiones ha cambiado drásticamente. Estos activos, debido a su naturaleza descentralizada y altamente especulativa, se caracterizan por presentar una volatilidad mucho más elevada, lo que introduce tanto nuevos riesgos como nuevas oportunidades para los inversores.

Este fenómeno ha puesto de manifiesto la necesidad de replantear las metodologías clásicas de optimización de carteras, que fueron diseñadas bajo la premisa de mercados más estables. Si bien décadas atrás avances como el modelo matemático de Markowitz, el cual hablaremos más adelante, fue un modelo revolucionario para la asignación óptima de activos, la aplicabilidad de estos enfoques en mercados volátiles como los de las criptomonedas sigue siendo objeto de debate. Los modelos tradicionales pueden subestimar la complejidad y los patrones de comportamiento característicos de estos activos, lo que hace imprescindible el desarrollo de nuevos enfoques que puedan capturar estas dinámicas de forma más precisa.

Este TFG tiene como objetivo principal adaptar los modelos tradicionales a un nuevo contexto, integrando los avances en informática, análisis de datos y teoría financiera. De esta manera, se busca revitalizar dichos modelos, facilitando su computación mediante programación, y además incorporar herramientas de machine learning para mejorar la precisión de las predicciones en mercados tan dinámicos como el de las criptomonedas. La aplicación de técnicas avanzadas como las redes neuronales podría proporcionar una solución innovadora, capaz de predecir y optimizar el comportamiento de los portafolios de inversión en criptomonedas,

aprovechando su capacidad para aprender patrones complejos en grandes volúmenes de datos. El desafío radica en integrar estos enfoques innovadores de manera que proporcionen a los inversores una herramienta útil y escalable para gestionar sus activos en un entorno altamente volátil y con un gran potencial de retorno, pero también de riesgos significativos.

1.2. Objetivos

El objetivo principal de este trabajo es desarrollar y evaluar un modelo de gestión de carteras que permita equilibrar riesgo y rentabilidad en activos altamente volátiles como las criptomonedas. Para ello, se abordarán los siguientes puntos:

- Extracción de datos de los distintos activos de estudio, con corrección de errores y datos faltantes.
- Desarrollo de estrategias de diversificación y asignación óptima de activos.
- Implementación de algoritmos de optimización de carteras.
- Evaluación del desempeño del modelo utilizando datos históricos y simulaciones.
- Creación de una herramienta software alojada en un repositorio GitHub, que permita a los inversores gestionar sus carteras con base en el modelo propuesto.

1.3. Estructura del documento

Este documento se estructura de la siguiente manera:

- En este capítulo, el **Capítulo 1** ,se presenta la introducción al trabajo, la motivación detrás de la investigación, los objetivos y la estructura general del documento, además de las tecnologías utilizadas.
- En el **Capítulo 2** se detallan el desarrollo teórico sobre la gestión de carteras, los modelos tradicionales y el uso de redes neuronales en la predicción financiera.
- El **Capítulo 3** describe la metodología utilizada de una manera práctica, incluyendo la extracción de datos, la implementación de algoritmos y el uso de redes neuronales.

- En el **Capítulo 4** se presentan los resultados obtenidos, la evaluación del modelo y las simulaciones realizadas.

- Finalmente, el **Capítulo 5** concluye el trabajo, discutiendo los resultados, las limitaciones y las posibles líneas de investigación futura.

1.4. Tecnologías usadas

Para la implementación de este trabajo, se han utilizado las siguientes tecnologías y herramientas:

1.4.1. NumPy

NumPy, abreviatura de ”Numerical Python”, es una biblioteca de Python diseñada para manejar vectores y matrices, facilitando cálculos numéricos y análisis de grandes volúmenes de datos.[1]

La funcionalidad principal de NumPy es su estructura de datos ”ndarray”, para una matriz de n dimensiones. Estas matrices son vistas escalonadas de la memoria. A diferencia de la estructura de datos de lista incorporada de Python, estas matrices se escriben de forma homogénea: todos los elementos de una única matriz deben ser del mismo tipo.[1]

Se destaca su esencialidad para la realización de cálculos, ahondando aún más en proyectos de análisis de datos y machine learning, siendo utilizada como base para otras bibliotecas avanzadas como Pandas y TensorFlow.

1.4.2. Pandas

Pandas es una librería de Python especializada en la manipulación y el análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales, es como el Excel de Python. Incluye un nuevo tipo de dato llamado ’DataFrame’ que permite la indexación integrada [2].

Es una herramienta clave para trabajar con grandes volúmenes de datos, necesarios para este proyecto, todo ello por su capacidad de importación desde múltiples fuentes y la integración con otras herramientas como puede ser NumPy.

1.4.3. cvxopt

cvxopt es una librería de Python especializada en la resolución de problemas de optimización convexa, especialmente en el ámbito de la optimización matemática aplicada a las finanzas. Proporciona herramientas eficientes para resolver problemas de programación lineal y cuadrática, lo que la hace ideal para la optimización de carteras de inversión. cvxopt permite formular y resolver problemas de optimización de manera sencilla, utilizando algoritmos avanzados y soportando grandes cantidades de datos, lo que la convierte en una librería clave para el desarrollo de modelos de optimización en mercados financieros.

1.4.4. Scipy

La librería **Scipy** proporciona herramientas avanzadas para la computación y, más importante para el uso previsto en este trabajo, de optimización. Específicamente, se utilizará la función **scipy.optimize.minimize**, la cual es una herramienta ampliamente utilizada en la optimización numérica, permitiendo resolver problemas de optimización no lineal con restricciones.

1.4.5. Pyplot de Matplotlib

La librería **Pyplot** nos permite realizar gráficos de datos, es intuitiva y nos permitirá representar de manera visual el crecimiento de la cartera con el paso del tiempo, y poder comparar las distintas soluciones para la optimización de dicha cartera.

1.4.6. Darts

Darts es una librería de código abierto desarrollada por Unit8 que permite trabajar con series temporales de forma eficiente y flexible. Darts ofrece una interfaz unificada para una amplia variedad de modelos de predicción, incluyendo métodos clásicos (como ARIMA o Exponential Smoothing) y modelos avanzados basados en aprendizaje profundo, como RNNs, TCNs, Transformers o N-BEATS, que es el utilizado en este estudio.

Dentro de este TFG, la librería Darts se utilizará principalmente para implementar y entrenar el modelo N-BEATS, una arquitectura de redes neuronales diseñada específicamente para la predicción de series temporales de forma interpretable, además del escalado de datos

(preparación de los datos) y métricas que pueden ser importantes para comprender mejor el funcionamiento del algoritmo.

1.4.7. Pyside6

Esta librería nos permite realizar una interfaz gráfica al programa de tal manera que, dicho programa sea más intuitivo de usar, facilitando así la muestra de los resultados finales, sin que el usuario tenga que tocar código.

2

Propuesta teórica de modelos de cartera

2.1. Introducción teoría de carteras

El fin de este documento es el de exponer, calcular y enfrentar modelos para la correcta gestión óptima de una cartera. Antes, introduciremos una breve teoría de carteras en el que se fundamentarán dichos modelos.

2.1.1. Activo

Un activo es un recurso o derecho con valor económico que puede comprarse y venderse, aunque su precio no es fijo. Puede adquirirse por una cantidad inicial p_0, que puede variar desde unos pocos céntimos hasta miles de euros o cualquier otra cifra. En una fecha específica, el activo es comprado y, al momento de su venta, su valor será p_1, el cual puede haber cambiado tanto positivamente, p_1 es mayor que p_0 y su venta nos reportará un beneficio, como negativamente, p_1 es menor que p_0 y su venta nos reportará una pérdida, o como última opción, mantenerse igual respecto al precio de compra p_0.

Retorno de un activo

Para intentar optimizar la cartera,a la cual contendrá estos activos, deberemos calcular el retorno. Denotamos el retorno de la criptomoneda i para el precio j como:

$$r_{ij} = \frac{p_{ij}}{p_{ij-1}} \tag{1}$$

2.1.2. Cartera (portfolio)

Las carteras de valores, o portfolios, están compuestas por un conjunto de activos. Como se ha mencionado, un activo puede ser un bien o un derecho con valor económico que puede ser objeto de compra y venta. Si un individuo posee un único activo, su cartera estará constituida únicamente por ese activo, aunque puede ampliarse incorporando nuevos activos con el tiempo.

Cada cartera agrupa las características de los activos que la componen. Supongamos que una cartera está formada por n activos, adquiridos con un presupuesto inicial x_0, el cual se distribuye entre todos los activos. El monto asignado a cada activo i dentro de la cartera cumple la relación:

$$x_{i_0} = \omega_i \cdot x_0$$

donde ω_i representa el peso del activo i en la cartera. Estos pesos deben cumplir la condición:

$$\sum_{i=1}^{Q} \omega_i = 1$$

Así, el valor total de la cartera puede expresarse como:

$$\text{Valor de la cartera} = \sum_{i=1}^{Q} \omega_i \cdot p_{i_0} = x_0 \cdot \sum_{i=1}^{Q} \omega_i = x_0$$

donde:

- ω_i es el peso del activo i.
- p_{i_0} es el valor inicial (de compra) del activo i.
- Q es el número total de activos en los que se planea invertir. Este número es un valor constante que refleja la cantidad de activos que se consideran en el portafolio.

Dado que la cartera agrupa múltiples activos, su retorno refleja el comportamiento global de la inversión. Si r_i representa el retorno de un activo i, el valor de la cartera en un instante posterior se calcula como la suma ponderada del retorno, el valor de compra y el peso de cada activo:

$$x_1 = \sum_{i=1}^{Q} r_i \cdot \omega_i \cdot p_{i_0} = x_0 \cdot \sum_{i=1}^{Q} r_i \cdot \omega_i$$

donde:

- r_i es el retorno del activo i.
- ω_i es el peso del activo i.
- p_{i_0} es el valor inicial (de compra) del activo i.
- x_0 es el valor total de compra de la cartera.
- x_1 es el valor de la cartera en el instante posterior.

2.2. Redes neuronales

Modelos Profundos de Redes Neuronales (ANN)

Una **Red Neuronal Artificial** (ANN por sus siglas en inglés, Artificial Neural Network) es un modelo computacional inspirado en la estructura y funcionamiento del cerebro humano, ya que, mientras que el reconocimiento de patrones es muy complicado, parece ser una tarea sencilla para los humanos [3]. Consiste en una red de nodos interconectados, llamados neuronas artificiales, que procesan información de entrada y generan una salida. Cada nodo en una capa realiza una operación matemática, y las salidas de una capa se usan como entradas para la siguiente. Estos modelos pueden aprender representaciones complejas a partir de los datos a través de un proceso denominado **entrenamiento**, donde se ajustan los pesos de las conexiones entre nodos para minimizar el error en la predicción.

Nos centraremos en las redes neuronales profundas, o denominado en inglés como **Deep Neural Networks (DNN)**, son una clase de redes neuronales que contienen múltiples capas ocultas entre la capa de entrada y la capa de salida. El uso de varias capas permite que el modelo aprenda relaciones más complejas y haga mejores predicciones, a partir de los ejemplos de entrenamiento.

La estructura de una red neuronal profunda se puede expresar como:

$$y = f(\mu_L \cdot f(\mu_{L-1} \cdot \ldots f(\mu_1 \cdot x + b_1) + b_L)) \tag{2}$$

donde:

- **x** es la entrada.
- $\mathbf{W}_i$ son los pesos de las conexiones en cada capa, el cual miden que tan importante es cada entrada en la activación de la siguiente capa. Son ajustados durante el entrenamiento.
- $\mathbf{b}_i$ son los sesgos. Son valores que se añaden para permitir mayor flexibilidad a la red y ajustarse mejor a los datos.
- $f(\cdot)$ es la función de activación no lineal. Introduce la no linealidad, lo que permite el aprendizaje de patrones complejos en vez de solo relaciones lineales.

El propósito de una **ANN** es aprender una representación detallada de los datos de entrada, donde cada capa ayuda a generar características cada vez más complejas.

Redes Neuronales Recurrentes (RNN)

Las **Redes Neuronales Recurrentes (RNN)** es una red neuronal profunda que se entrena con datos secuenciales o de series temporales para crear un modelo de machine learning (ML) que pueda hacer predicciones o conclusiones secuenciales basándose en entradas secuenciales[4] , son especialmente útiles para la predicción de series temporales. A diferencia de las Redes Neuronales Artificiales (ANN), donde cada entrada es independientemente procesada, las RNN poseen **conexiones recurrentes** que permiten mantener un estado interno que captura información de las entradas anteriores.

Su estructura permite que la red almacene información sobre secuencias de datos y la reutilice en predicciones futuras.

Concepto del gradiente

A pesar de las ventajas de estas redes, frente a las tradicionales ANN, también presentan varias limitaciones que dificultan su aplicación en problemas con dependencias a largo plazo.

Para entender algunas de las limitaciones de las RNN, es importante conocer el concepto de gradiente. En términos matemáticos, el gradiente de una función escalar $f(x)$ respecto a

un conjunto de parámetros x es un vector que apunta en la dirección de mayor crecimiento de la función:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

El gradiente, hablando de un contexto del aprendizaje automático, indica cómo deben ajustarse los parámetros de un modelo para que se **minimice** la función de pérdida. En las RNN, estos gradientes se calculan mediante "retropropagación a través del tiempo" (BPTT, por sus siglas en inglés).

Limitaciones de las RNN

- **Problema del Desvanecimiento del Gradiente**: Durante el entrenamiento con *back-propagation through time* (BPTT), los gradientes pueden volverse **extremadamente pequeños** al propagarse hacia atrás a través de muchos pasos de tiempo. Esto impide que la red aprenda relaciones de largo plazo en la secuencia.
- **Explosión del Gradiente**: En ciertos casos, los gradientes pueden **crecer exponencialmente**, lo que provoca inestabilidad en el entrenamiento. Este problema se mitiga con técnicas como el *gradient clipping*, el cual limita el valor máximo del gradiente, evitando así este problema.
- **Dificultad para Capturar Dependencias a Largo Plazo**: Aunque también sea un efecto consecuencai del desvanecimiento del gradiente (hecho que ocurre durante el entrenamiento) podemos observar que debido a la propia arquitectura de las RNN con la actualización en cada nuevo paso de la memoria de la red (también denominado más técnicamente como *estado oculto*), las RNN tienden a priorizar información reciente y olvidar detalles importantes de estados anteriores en secuencias largas.

2.2.1. Redes LSTM

Las **Redes Neuronales de Memoria a Largo Plazo (LSTM, Long Short Term Memory)** son un tipo de red neuronal recurrente (RNN) que se desarrolló para mejorar las carencias de las RNN tradicionales, utilizando puertas de memoria y limitando la influencia de los pesos, para tener la capacidad de retener información relevante a lo largo de grandes períodos

de tiempo y solucionar el problema del desvanecimiento del gradiente. Por ello, son capaces de capturar dependencias temporales complejas, lo que hace que sea especialmente efectivas para aprender de secuencias de datos temporales. [5]

Una red LSTM consta de una serie de celdas que contienen tres puertas:

- **Puerta de entrada**: Determina qué nueva información se añadirá a la memoria.
- **Puerta de olvido**: Decide qué información de la memoria tiene que ser descartada.
- **Puerta de salida**: Controla qué parte de la memoria se utiliza para generar la salida de la red.

La fórmula básica que define la evolución de una celda LSTM es[6]:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{5}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tau(W_c x_t + U_c h_{t-1} + b_c) \tag{6}$$

$$h_t = o_t \circ \tau(c_t) \tag{7}$$

Donde:

- n: tamaño de la entrada
- m: tamaño de la celda y salida
- x_t: vector de entrada en el tiempo t, tamaño $n \times 1$
- f_t: vector de la puerta de olvido, tamaño $m \times 1$
- i_t: vector de la puerta de entrada, tamaño $m \times 1$
- o_t: vector de la puerta de salida, tamaño $m \times 1$
- h_t: vector de salida, tamaño $m \times 1$
- c_t: vector de estado de la celda, tamaño $m \times 1$

- W_f, W_i, W_o, W_c: matrices de pesos de entrada, tamaño $m \times n$
- U_f, U_i, U_o, U_c: matrices de pesos de salida, tamaño $m \times m$
- b_f, b_i, b_o, b_c: vectores de sesgo, tamaño $m \times 1$
- σ: función de activación sigmoide logística
- τ: función de activación tangente hiperbólica

Este diseño permite que las redes LSTM capturen dependencias de largo plazo en datos secuenciales, lo que las hace particularmente útiles para tareas de predicción temporal, como la predicción de series temporales en mercados financieros.

2.3. Modelo de Media-Varianza de Markowitz

El modelo de varianza media, o también conocido como el modelo de Markowitz en referencia a su autor Harry Markowitz en 1952[7], se presenta como un modelo de optimización de carteras. Su objetivo está en seleccionar la cartera más eficiente en base a la relación riesgo-beneficio.

2.3.1. Varianza

La varianza representa la media de las desviaciones cuadráticas de una variable aleatoria, referidas al valor medio de esta[8]. En otras palabras, pretende medir la relación de los datos con su media.

Si la varianza es muy grande, encontraremos que los datos están muy dispersos, y la media aritmética será menos representativa a la hora de intentar agrupar el comportamiento de dichos datos.

2.3.2. Covarianza

Otra medida de dispersión que es necesaria para la utilización de este modelo será la covarianza. La covarianza no es más que una medida de la relación que tienen dos variables distintas respecto a sus referencias. La covarianza podrá tomar tres tipos de valores: positivos, negativos o nulos. Cuando la covarianza de dos variables toma valor nulo, implica que entre

ellas no hay una relación, por lo que el aumento o disminución de una no tendrá impacto en el comportamiento de la otra. Si dicha covarianza es mayor que cero, significará que las variables están relacionadas positivamente y si una aumenta, la otra también lo hará. Por el contrario, si la covarianza resulta negativa, refleja que las variables guardan una relación negativa y la disminución de una provocará también la disminución de la otra.

2.3.3. Principios

Para ello, primeramente, Markowitz sentó unos principios para poder postular dicho modelo, como son:

El riesgo de la cartera es medido por la volatilidad y la covarianza de los rendimientos de los activos que componen la cartera, y matemáticamente queda definido como:

$$\text{Riesgo} = \mathbf{w}^T \mathbf{\Sigma} \mathbf{w} \tag{8}$$

$$(1 \times Q)(Q \times 1) = (1 \times 1)$$

donde:

- $\mathbf{w} = (w_1, w_2, \ldots, w_Q)$ es el vector de pesos de los activos en el portafolio, y $\mathbf{w}^T$ representa su transpuesta. ($\mathbf{w} \in R^{Q \times 1}$)
- $\mathbf{\Sigma}$ es la matriz de **varianza-covarianza** de los retornos. ($\mathbf{\Sigma} \in R^{Q \times Q}$)

Por su parte, la rentabilidad de dicha cartera se estima como la media ponderada de los retornos esperados de los activos financieros que la componen, y queda definida como:

$$\text{Rentabilidad} = \boldsymbol{\mu}^T \mathbf{w} \tag{9}$$

$$(1 \times Q)(Q \times 1) = (1 \times 1)$$

donde:

- $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_Q)$ es el vector de rentabilidad esperada de los activos, y $\boldsymbol{\mu}^T$ representa su transpuesta. ($\boldsymbol{\mu} \in R^{Q \times 1}$)

2.3.4. ¿Cómo funciona el modelo?

Una vez presentadas las bases, y los principios teóricos y estadísticos en los que se basa este modelo, podremos concretar que dicho modelo de Markowitz se basa en el criterio de varianza-media; en otras palabras, pretende maximizar la rentabilidad minimizando el riesgo. Recordemos que la varianza, la información que nos da es la dispersión de los datos en relación a su media, por lo que, si buscamos que dicha dispersión sea lo menor posible, estaremos ante un activo más estable y reduciremos el riesgo.

El modelo de **Markowitz** optimiza un portafolio buscando un balance entre **rentabilidad esperada** y **riesgo**, ajustado por la tolerancia del inversor.

Función Objetivo

La función objetivo auna las dos partes previamente explicadas y añade el coeficiente λ, relacionado con la aversión del riesgo:

$$\min_{\mathbf{w}} \quad \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} - \lambda \, \boldsymbol{\mu}^T \mathbf{w} \tag{10}$$

donde:

- λ es el coeficiente de aversión al riesgo, determinado por el perfil del inversor:
  - λ bajo $\Rightarrow$ inversor conservador.
  - λ alto $\Rightarrow$ inversor agresivo.

El resultado de dicha función será un número natural, que nos indicará cuanto es el peso asociado para el activo **i** de todos los activos, denominando dicho conjunto de todos los activos que queramos tener en nuestra cartera como **Q**.

2.3.5. Restricciones

La optimización está sujeta a las siguientes restricciones:

$$\sum_{i=1}^{Q} w_i = 1 \tag{11}$$

Esta ecuación garantiza que todo el capital esté completamente invertido, evitando que se tenga dinero sin asignar.

$$w_i \geq 0, \quad \forall i = 1, \ldots, Q \tag{12}$$

Esta condición impide la venta en corto, asegurando que los pesos w_i sean siempre no negativos.

2.4. Cálculo del riesgo con CVAR

Este modelo de optimización utiliza CVaR (Valor en Riesgo Condicional) como medida de evaluación del riesgo en vez de utilizar la varianza, como es en el caso del modelo de Markowitz. Para explicar el Valor en Riesgo Condicional (CVaR), primeramente debemos explicar que es el Valor en Riesgo (VaR).

2.4.1. Valor en Riesgo (VaR):

El **Valor en Riesgo**, o VaR, es una medida de riesgo ampliamente utilizada del riesgo de mercado en una cartera de inversiones de activos financieros pretende estimar la **máxima pérdida** posible durante un tiempo después de excluir todos los peores resultados, según la probabilidad máxima que hayamos indicado.[9]

Formalmente, el VaR al nivel de confianza α se define como el percentil, un valor de la distribución que divide el conjunto de datos en un porcentaje específico, de la distribución de pérdidas en el portafolio. En otras palabras, el percentil nos indica el valor por debajo de cuál se encuentra una cierta probabilidad.

$$\text{VaR}_\alpha = -\text{percentil}_\alpha(\text{Distribución de pérdidas}) \tag{13}$$

Por ejemplo, si se tiene un VaR al 95 %, esto significa que hay un 95 % de probabilidad de que las pérdidas no superen el valor de VaR durante el período de tiempo considerado.

2.4.2. Valor en Riesgo Condicional (CVaR):

El CVaR, también conocido como **Expected Shortfall**, es una medida que estima la **pérdida promedio** en el caso de que las pérdidas superen el VaR. Es decir, mide el riesgo de los

peores escenarios. Formalmente, el CVaR se calcula como el valor esperado de las pérdidas, condicionado a que las pérdidas sean mayores que el VaR:

$$\text{CVaR}_\alpha = E\left[\text{Pérdida} \mid \text{Pérdida} \geq \text{VaR}_\alpha\right] \tag{14}$$

2.4.3. Diferencias entre CVaR y VaR

- **VaR** mide la **pérdida máxima esperada** dado un nivel de confianza, pero no tiene en cuenta la magnitud de las pérdidas más allá de ese percentil. Es decir, el VaR no nos dice nada sobre las pérdidas en los peores casos (en el 5 % de los escenarios más adversos si se tiene un VaR al 95 %).
- **CVaR**, por otro lado, mide el **promedio de las pérdidas** en el **5 % peor de los escenarios** (si se tiene un VaR al 95 %), lo cual lo convierte en una medida más conservadora y robusta para evaluar el riesgo en situaciones extremas.
- En resumen:
  - El VaR solo te dice cuánto podrías perder con un nivel de confianza determinado (por ejemplo, 95 %).
  - El CVaR te dice cuánto perderías, en promedio, si las pérdidas superan el VaR (en el peor caso).
- Robustez: El CVaR es generalmente preferido cuando se evalúan **riesgos extremos**, ya que considera el comportamiento del portafolio en los peores casos, mientras que el VaR no proporciona información sobre la magnitud de esas pérdidas extremas. Precisamente por este motivo, es clave para el uso de este caso, que manejamos activos de alta volatilidad.

2.4.4. Modelo con CVaR

El modelo de optimización de portafolio utilizando el **Conditional Value at Risk** (CVaR) en vez de la varianza se formula de la siguiente manera:

$$\min_{\mathbf{w}} \quad \text{CVaR}_\alpha(\mathbf{w}) - \lambda \boldsymbol{\mu}^T \mathbf{w} \tag{15}$$

donde:

- $\mathbf{w}$ y $\boldsymbol{\mu}$ son análogas a las vistas anteriormente.

- $\text{CVaR}_\alpha(\mathbf{w})$ es el valor en riesgo condicional del portafolio, que mide la pérdida esperada en el peor caso, condicionado a un nivel de probabilidad.

Restricciones

La optimización está sujeta a las restricciones homólogas a los demás modelos explicadas anteriormente. Invertimos el capital completo e impedimos ventas en corto.

2.5. Modelo de portafolio igualmente ponderado (Equally Weighted)

Este modelo, también conocido en inglés como *Equally weighted*, funciona sorprendentemente bien respecto a la sencillez que conlleva su cálculo, y nos refiere la importancia que conlleva en un portfolio la diversificación de los activos. Su uso ha sido destacable para la inversión en empresas pequeñas que contienen gran potencial, lo que podría ser un cierto equivalente a estos activos de alto riesgo que estamos estudiando.

$$\mathbf{w_q} = \frac{1}{Q} \tag{16}$$

- $\mathbf{w_q}$: Es el **peso** de un activo específico en el portafolio. El peso representa la proporción del capital total invertido en ese activo. En este caso, la fórmula indica que todos los activos tienen el mismo peso.

- $\frac{\mathbf{1}}{\mathbf{Q}}$: El **peso de cada activo** se determina dividiendo 1 entre el número total de activos Q, lo que significa que si estamos invirtiendo en Q activos, el capital se distribuye de manera equitativa entre ellos.

2.6. Modelo de Media-Varianza con Diversificación (Diversified Markowitz)

Función Objetivo

$$\min_{\mathbf{w}} \quad \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} - \lambda \boldsymbol{\mu}^T \mathbf{w} + \delta \mathbf{w}^T \mathbf{w} \tag{17}$$

La ecuación anterior define la **función objetivo** de un modelo de optimización de portafolios, donde se busca maximizar la rentabilidad ajustada por el riesgo, con un factor de diversificación añadido. Esta función está descompuesta en tres términos:

- $\delta \mathbf{w}^T \mathbf{w}$: Este término fomenta la **diversificación**, evitando que la inversión se concentre demasiado en pocos activos. El parámetro δ regula la penalización sobre distribuciones que estén desbalanceadas, lo que incentivan una asignación de capital más equitativa entre los distintos activos del portafolio.

Al analizar el comportamiento del modelo a medida que el parámetro δ aumenta, se observa que si δ tiende a infinito, el término de diversificación $\mathbf{w}^T \mathbf{w}$ dominará la función objetivo. En este escenario, el resultado que arrojará la función será un portafolio con pesos iguales para todos los activos, es decir, un modelo **equally weighted**. Esto ocurre porque, con δ muy grande, la maximización de la diversificación se vuelve más importante que la rentabilidad y el riesgo, y la solución natural para cumplir con dicha condición es la asignación del mismo peso para cada activo en el portafolio.

Este comportamiento pone de manifiesto cómo el parámetro δ permite ajustar el grado de diversificación en el portafolio, desde una distribución completamente balanceada de los activos hasta una optimización más tradicional centrada en el riesgo y la rentabilidad.

Restricciones

La optimización está sujeta a las restricciones homólogas a los demás modelos, explicadas anteriormente. Invertimos el capital completo e impedimos ventas en corto.

2.7. N-BEATS

El modelo **Neural Basis Expansion Analysis for Time Series Forecasting (N-BEATS)** es un modelo nde redes neuronales basado en el perceptrón multicapa con enlaces residuales, y utiliza una técnica llamada expansión de bases (del inglés, basis expansion) para descomponer la serie temporal original en componentes (más) básicos[10]. Se destaca por su capacidad de generalización sin necesidad de realizar ajustes específicos en los datos de entrada. N-BEATS se basa en una arquitectura de capas totalmente conectadas con conexiones residuales hacia adelante y hacia atrás.

Importancia en los Mercados Financieros

La predicción de series temporales es crucial en los mercados financieros para identificar patrones y tendencias en activos como acciones, criptomonedas y divisas. N-BEATS ofrece ventajas significativas sobre otros modelos tradicionales debido a:

- Su capacidad para modelar relaciones no lineales en los datos financieros.
- Su flexibilidad para adaptarse a distintos tipos de activos sin necesidad de preprocesamiento complejo.
- Su eficacia demostrada en múltiples benchmarks, como en la competencia M4 de predicción de series temporales.

Funcionamiento del Modelo

N-BEATS está compuesto por una arquitectura de pila de bloques residuales. Esta estructura iterativa permite que cada bloque contribuya secuencialmente a la precisión del modelo, pudiendo corregir errores de bloques anteriores gracias al mecanismo de residuales.

- **Reconstrucción del historial (backcast)**: la salida, representada como $\hat{x}_\ell$, tiene el objetivo de capturar aquella parte del historial que el bloque es capaz de modelar (puede explicar). Este backcast se resta del residuo actual, y el resultado es utilizado como entrada para el siguiente bloque. Esto permite que cada bloque elimine patrones ya explicados, forzando a los bloques posteriores a enfocarse en patrones que aún no han sido explicados.

- **Predicción futura (forecast)**: salida denotada como $\hat{y}_\ell$, representa la contribución del bloque a la predicción del horizonte de predicción. Las predicciones individuales producidas por todos los bloques que hay dentro de una pila se combinadas mediante una suma para obtener la predicción total generada por dicha pila.

Ambas salidas se generan mediante funciones de base:

$$\hat{y}_\ell = g_\ell^f(\theta_\ell^f) \tag{18}$$

$$\hat{x}_\ell = g_\ell^b(\theta_\ell^b) \tag{19}$$

Donde:

- $\hat{y}_\ell$ es la predicción del bloque ℓ (forecast).
- $\hat{x}_\ell$ es la reconstrucción del historial (backcast).
- g_ℓ^f y g_ℓ^b son funciones de base que transforman los coeficientes en predicciones.
- θ_ℓ^f y θ_ℓ^b son coeficientes de expansión aprendidos por la red.

Finalmente, las predicciones generadas por cada pila se suman para producir la predicción global del modelo:

$$\hat{y} = \sum_{m=1}^{M} \sum_{\ell=1}^{K} \hat{y}_\ell^{(m)}$$

donde M es el número total de pilas (stacks), y cada pila contiene K bloques.

Podemos **resumir el funcionamiento de N-BEATS** en:

- El modelo recibe como entrada una ventana de observación (lookback window) de longitud nH, donde H es el horizonte de predicción y n es el número de bloques que cubren ese horizonte.
- Esta entrada se propaga secuencialmente a través de múltiples **stacks** (pilas), cada uno formado por varios **blocks** (bloques).
- En cada bloque, la entrada se descompone en backcast y forecast. El backcast se resta del input (residuo), mientras que el forecast se acumula.

- El residuo resultante pasa al siguiente bloque dentro de la misma pila, y posteriormente a la siguiente pila.
- Al finalizar, el modelo produce una predicción global (global forecast) de longitud H, correspondiente al horizonte de predicción.

Este diseño modular permite al modelo explicar de manera progresiva la información de la serie, separando patrones temporales de distinta complejidad en diferentes bloques y pilas, mejorando así las predicciones.

3

Desarrollo práctico

3.1. Datos de estudio

3.1.1. ¿Cómo conseguimos los datos?

Primeramente, debemos obtener los datos financieros de los activos que queremos estudiar para incluirlos en nuestra cartera. Para conseguir dichos datos, he utilizado el exchange [11] de KuCoin. Esta plataforma cuenta con una API (Interfaz de Programación de Aplicaciones) con la que podemos hacer peticiones HTTP para conseguir los datos que requerimos.

Estos servidores utilizan lo que se llama timestamp, o en español, Tiempo Unix. Este es un sistema para la descripción de instantes de tiempo: se define como la cantidad de segundos transcurridos desde la medianoche UTC del 1 de enero de 1970, sin contar segundos intercalares[12].

Por lo que, para el manejo de los datos, previamente deberemos controlar todo en esta medida de tiempo, y para ello tenemos que hacer uso de medidas en Python como datetime y timestamp para hacer la conversión en ambos sentidos, pudiendo trabajar fácilmente con los rangos que queramos.

3.1.2. Activos de estudio

Para este estudio, he utilizado los valores del precio diario del año 2024, la naturaleza de estas criptomonedas es variada para enriquecer las posibilidades, desde activos más consolidados como BTC (Bitcoin) a activos aún más volátiles como puede ser SHIB. Se han seleccionado las siguientes criptomonedas:

- **BTC-USDT**: El par BTC-USDT representa el intercambio entre Bitcoin (BTC), una de las criptomonedas más populares.

- **ETH-USDT**: El par ETH-USDT representa el intercambio entre Ethereum (ETH), la segunda criptomoneda más grande por capitalización.
- **XRP-USDT**: XRP es la criptomoneda de Ripple, utilizada para pagos rápidos y bajos costos de transacción.
- **ADA-USDT**: Cardano (ADA) es una criptomoneda diseñada para ser más escalable y sostenible que otras.
- **SOL-USDT**: Solana (SOL) es conocida por su alta velocidad de transacción y bajo costo, dirigida a contratos inteligentes.
- **BNB-USDT**: Binance Coin (BNB) es la criptomoneda nativa de la plataforma de intercambio Binance.
- **DOT-USDT**: Polkadot (DOT) facilita la interoperabilidad entre diferentes cadenas de bloques.
- **AVAX-USDT**: Avalanche (AVAX) es una plataforma blockchain de alto rendimiento para aplicaciones descentralizadas.
- **DOGE-USDT**: Dogecoin (DOGE) comenzó como una criptomoneda satírica pero ha ganado popularidad debido a su comunidad y el apoyo de figuras como Elon Musk.
- **SHIB-USDT**: Shiba Inu (SHIB) es un token basado en Ethereum que se ha vuelto popular por su insignia.

Se da un pequeño contexto de cada criptomoneda para situarla en el mercado. Además, se designan con el par USDT, activo de la empresa Tether, el cual se considera una moneda estable porque originalmente se diseñó para que valiera siempre $1.00 [13].

3.2. Obtención de datos

3.2.1. Método base

```python

def getData(coin, interval, start, end, shared_data=None, writeOnExcel=False):
    """Get the data from the KuCoin API.
    Due to limitations in KuCoin API it only can retrieve 1500 elements.

    Args:
        coin (str): coin pair (e.g. BTC-USDT)
        interval (str): interval (1min, 5min, 15min, 1hour, 4hour, 8hour, 1day, 1week, 1month)
        start (int): start timestamp
        end (int): end timestamp
        writeOnExcel (bool): write the data to an Excel file (default: False)

    Returns:
        list: data
    """
    #print(f"Getting data for {coin} from {start} to {end}")
    data = client.get_kline(coin, interval, startAt=start, endAt=end)
    file = f"{coin}.{interval}.{start}.{end}.xlsx"

    if writeOnExcel:
        write_to_excel(data, file)

    if shared_data is not None:
        shared_data.extend(data)

    return data
```

Listing 1: Método base para la obtención de los datos

Este enfoque, en el que se utiliza el parámetro shared_data es para mejorar el rendimiento, permitiendo la programación multi-hilo, para el estudio de valores masivos y uso de valores

con una temporalidad más reducida, como el uso de horas o minutos.

3.3. Limpieza de datos

Utilizaremos como apoyo archivos excel (.xlsx), para guardar los precios y los retornos de los distintos activos, y de ellos poder ejecutar operaciones y los modelos computables que se han mostrado en el desarrollo teórico.

Primeramente, antes de escribir los datos en estos archivos, debemos asegurarnos de que tienen valores correctos, como únicamente nos interesa quedarnos con la marca de tiempo, y el precio de cierre diario del activo, haremos operaciones sobre la columna *Close* para asegurarnos que la columna es de tipo float, forzaremos que los valores nulos sean inválidos, e interpolaremos dichas filas con valores nulos utilizando el método lineal. Posteriormente, si siguen habiendo valores nulos después de dichas operaciones, se eliminarán las filas que los contengan.

```
    # Convert 'Close' to float, force NaN values to be invalid values
    df = df.apply(pd.to_numeric, errors='coerce')

    # Interpolate null values in 'Close'
    df['Close'] = df['Close'].interpolate(method="linear")

    # Delete rows where 'Close' is NaN after the interpolation
    df.dropna(subset=['Close'], inplace=True)
```

Listing 2: Limpieza de datos

3.4. Pasos previos para la computación de los modelos

Para la resolución de los modelos vistos en la sección anterior, habremos generado un excel con los precios de las distintas criptomonedas a incluir en la cartera, y además haber calculado los retornos diarios de cada una de ellas.

	BTC-USDT	ETH-USDT	XRP-USDT	ADA-USDT	SOL-USDT	BNB-USDT	DOT-USDT	AVAX-USDT	DOGE-USDT	SHIB-USDT
Precios										
1	44175.2	2351.6	0.62943	0.623208	109.936	313.524	8.5944	41.94	0.09198	0.000010679
2	44940.6	2355.42	0.6243	0.604917	106.729	312.163	8.4105	40.609	0.09109	0.000010557
3	42840.9	2209.49	0.58225	0.556923	98.53	315.896	7.6296	36.786	0.08199	0.000009564
4	44149.2	2266.99	0.58679	0.569874	104.924	323.513	7.8757	38.689	0.08399	0.00000999
5	44147.9	2268.23	0.57579	0.541239	99.933	317.475	7.4462	36.456	0.08266	0.000009822
6	43970.8	2240.49	0.56804	0.522845	93.799	307.491	7.133	34.63	0.08059	0.000009604
7	43926.6	2221.25	0.55139	0.49449	89.4	302.404	6.8972	33.638	0.07824	0.00000902
8	46949.2	2330.71	0.5775	0.541044	97.931	303.711	7.4856	36.078	0.08132	0.000009578
9	46116.8	2344.3	0.56694	0.51215	99.364	301.212	7.1233	34.651	0.07922	0.000009661
10	46644.8	2584.58	0.60057	0.566097	102.001	305.636	7.9892	38.53	0.08306	0.000009978
11	46340.1	2617.56	0.60196	0.581862	99.89	308.33	8.1472	39.293	0.08457	0.00001015
12	42780.2	2522.33	0.56982	0.547768	92.107	296.653	7.5656	35.911	0.08007	0.000009633
13	42842.7	2578.24	0.57471	0.548999	95.844	302.1	7.618	36.487	0.08092	0.00000988
14	41727.2	2473.61	0.57623	0.525022	93.821	299.518	7.3188	35.292	0.08003	0.000009569

Cuadro 1: Precios de criptomonedas en pares USDT (Primeras 14 filas)

	BTC-USDT	ETH-USDT	XRP-USDT	ADA-USDT	SOL-USDT	BNB-USDT	DOT-USDT	AVAX-USDT	DOGE-USDT	SHIB-USDT
Retornos										
1	0	0	0	0	0	0	0	0	0	0
2	0.01718	0.00162	-0.00818	-0.02979	-0.02961	-0.00435	-0.02163	-0.03225	-0.00972	-0.01149
3	-0.04785	-0.06396	-0.06973	-0.08266	-0.07993	0.01189	-0.09745	-0.09887	-0.10525	-0.09878
4	0.03008	0.02569	0.00777	0.02299	0.06288	0.02383	0.03175	0.05044	0.02410	0.04358
5	-0.00003	0.00055	-0.01892	-0.05155	-0.04874	-0.01884	-0.05608	-0.05945	-0.01596	-0.01696
6	-0.00402	-0.01231	-0.01355	-0.03458	-0.06335	-0.03195	-0.04297	-0.05139	-0.02536	-0.02245
7	-0.00101	-0.00862	-0.02975	-0.05576	-0.04803	-0.01668	-0.03362	-0.02906	-0.02959	-0.06274
8	0.06655	0.04810	0.04627	0.08997	0.09114	0.00431	0.08187	0.07003	0.03861	0.06002
9	-0.01789	0.00581	-0.01845	-0.05488	0.01453	-0.00826	-0.04961	-0.04036	-0.02616	0.00863
10	0.01138	0.09758	0.05763	0.10015	0.02619	0.01458	0.11472	0.10611	0.04733	0.03229
11	-0.00655	0.01268	0.00231	0.02747	-0.02091	0.00878	0.01958	0.01961	0.01802	0.01709
12	-0.07993	-0.03706	-0.05487	-0.06038	-0.08112	-0.03861	-0.07406	-0.09000	-0.05468	-0.05228
13	0.00146	0.02192	0.00855	0.00224	0.03977	0.01819	0.00690	0.01591	0.01056	0.02532
14	-0.02638	-0.04143	0.00264	-0.04466	-0.02133	-0.00858	-0.04007	-0.03330	-0.01106	-0.03198

Cuadro 2: Retornos de criptomonedas en pares USDT (Primeras 14 filas)

Para el desarrollo del código que compute todas las soluciones y modelos propuestos en el desarrollo práctico, se ha utilizado un enfoque de orientación a objetos, teniendo Asset (activo) y Portfolio (la cartera a optimizar) que contendrá los distintos activos.

3.5. Modelo de Media-Varianza de Markowitz

Para resolver y poder computar este modelo, deberemos utilizar el resolvedor de funciones cuadráticas (QP, *Quadratic Programming*), que posee la librería cvxopt. Esto es debido a que el problema que pretende resolver este modelo se basa en una función objetivo que contiene un producto de vectores.

- **Función objetivo cuadrática**: La función $w^T \Sigma w$ es cuadrática, ya que involucra un término de segundo grado ($w^T \Sigma w$, donde w es un vector de pesos y Σ es la matriz de covarianzas).

- **Restricciones lineales**: Las restricciones, como $\sum w_i = 1$ y $w^T \mu \geq r$, son lineales. Las restricciones sobre los pesos son lineales, y por lo tanto no afectan la naturaleza cuadrática del problema.

3.5.1. Código

En este fragmento de código, designamos el valor lambda, que será el valor que indique cuanto quiere arriesgarse o no un inversor, obtenemos la covarianza, la pasamos a formato matriz de la librería cvxopt ya que es el dato requerido por el resolvedor y además, calculamos la rentabilidad esperada y le añadimos el riesgo.

```
1        l_riesgo = 3
2        cov = df_return.cov()
3        cov_matrix = matrix(cov.values)
4
5        r = df_return.mean()
6        r_riesgo = matrix(-l_riesgo * r.values)
```

Listing 3: Lambda, covarianza y rentabilidad según riesgo

Restricción de Igualdad: Suma de Pesos Igual a 1

Se debe garantizar que la suma de todos los pesos sea igual a 1, o 100 % si nos referimos al valor porcentual. Recordamos que matemáticamente, esta restricción se expresa como:

$$\sum_{i=1}^{n} w_i = 1$$

donde:

- w_i representa los pesos asignados a cada activo
- n es el número total de activos.

En `cvxopt`, esta restricción se modela con la ecuación:

$$Aw = b$$

donde:

- A es una matriz fila de unos:

$$A = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$$

- b es un escalar con valor 1.

En código:

```
A = matrix(np.ones(len(self.assets)).reshape(1, -1))
b = matrix(1.0)  # Escalar 1
```

Listing 4: Restricción suma de pesos igual a 1

Restricción de No Negatividad: $w_i \geq 0$

En muchos problemas, no se permite que los pesos sean negativos (por ejemplo, en inversión sin ventas en corto) como es en este caso. Esta restricción se modela como:

$$w_i \geq 0, \quad \forall i \in \{1, \dots, n\}$$

En términos de programación lineal, esto se expresa como:

$$Gw \leq h$$

donde:

- G es la matriz identidad negativa, para representar $-w_i \leq 0$, es decir, $w_i \geq 0$.
- h es un vector de ceros.

En código:

```
G = matrix(-np.eye(len(self.assets))) # Matriz identidad en negativo
h = matrix(np.zeros(len(self.assets))) # Vector de ceros (pesos iguales o
    mayores a 0)
```

Listing 5: Restricción suma de pesos igual a 1

Por último:

```
sol = solvers.qp(P=cov_matrix, q=r_riesgo, G=G, h=h, A=A, b=b)

w_opt = np.round(np.array(sol['x']).flatten(), 4)

print("(Markowitz) Optimal weights:", w_opt)
print("Sum of weights:", w_opt.sum())
```

A continuación, se explican los elementos clave del código.

Explicación de `sol = solvers.qp(P, q, G, h, A, b)`

La función `solvers.qp` resuelve el problema de optimización cuadrática, sujeto a las restricciones:

$$Gw \leq h, \quad Aw = b$$

donde:

- $P =$ `cov_matrix` representa la matriz de covarianza de los activos.

- $q =$ `r_riesgo` es el vector de penalización del riesgo.
- G y h establecen las restricciones de no negatividad ($w_i \geq 0$).
- A y b imponen la restricción de que la suma de los pesos sea igual a 1.

Cálculo de los Pesos Óptimos

Después de resolver el problema de optimización, se extraen los pesos óptimos:

```
w_opt = np.round(np.array(sol['x']).flatten(), 4)
```

Aquí:

- `sol['x']` contiene la solución óptima w.
- Se convierte a un array de NumPy con `np.array(sol['x'])`.
- `flatten()` transforma la matriz columna en un vector unidimensional.
- `np.round(..., 4)` redondea los valores a 4 decimales.

3.6. Modelo CVAR

Entre los métodos disponibles, **SLSQP** (Sequential Least Squares Programming) es particularmente útil para problemas de optimización convexa con restricciones de igualdad y desigualdad.

SLSQP se basa en la aproximación secuencial de mínimos cuadrados, combinando métodos de programación cuadrática con técnicas de optimización restringida. En el contexto de optimización de portafolios financieros, permite encontrar la asignación óptima de pesos considerando restricciones como la suma de los pesos igual a 1 y la no negatividad. Este método es eficiente para resolver problemas donde la función objetivo es diferenciable y donde se requiere manejar múltiples restricciones de forma flexible.

El uso de **SLSQP** en la optimización de carteras proporciona una alternativa robusta a métodos basados en programación cuadrática, como los implementados en **cvxopt**, y es especialmente útil cuando se busca minimizar medidas de riesgo como el Valor en Riesgo Condicional (CVaR) en lugar de la varianza del portafolio.

3.7. Modelo de portafolio igualmente ponderado

La implementación de este modelo es sencilla, requiere dividir de manera equitativa los pesos o porcentajes a invertir para cada criptomoneda, asignando así el mismo valor para cada una, sumando entre todas la cartera completa.

```
df_result = pd.DataFrame([1/len(self.assets)] * len(self.assets
    ), columns=["Equally Weighted"])
```

El resto de la implementación es muy similar a los otros modelos y, al igual que todos ellos, puede verse en el archivo *Portfolio.py* en la clase Portfolio.

3.8. Modelo de Media-Varianza con Diversificación (Diversified Markowitz)

En este modelo, a parte del uso del valor lambda que gestiona la importancia del riesgo/-beneficio, contamos además con otro valor δ que representará el grado de diversificación que queramos.

Cuanto mayor sea el valor de δ más será la diversificación del portfolio. De hecho, en el caso que añadamos un valor muy alto, estaremos ante un portfolio equivalente al igualmente ponderado, debido a la gran importancia que dará el modelo a la diversificación.

3.9. N-BEATS

3.9.1. Integración de N-BEATS en la optimización de carteras

La clave del enfoque basado en **N-BEATS** es que, a diferencia de los modelos clásicos utilizados en finanzas, no predice directamente rentabilidades o porcentajes, sino que genera **predicciones explícitas de precios futuros**, por tanto, no podremos usarlo de manera directa para componer nuestro portfolio.

Esta característica, lejos de ser una limitación, se convierte en una oportunidad al permitir una integración innovadora con los modelos de **optimización de carteras**. En lugar de utilizar medias históricas o suposiciones simplificadas sobre el comportamiento del mercado,

podemos emplear las predicciones generadas por N-BEATS para **estimar un vector dinámico de rendimientos esperados**, el cual sí que podemos usar como entrada para modelos como el de Markowitz. De esta forma, se combinan el *potencial predictivo* de las redes neuronales profundas con el *rigor cuantitativo* de los modelos financieros clásicos.

Este enfoque híbrido aporta las ventajas de ambos mundos:

- **Capacidad del modelo N-BEATS** para capturar patrones temporales complejos, relaciones no lineales y dinámicas de mercado que los modelos clásicos no logran identificar.
- **Robustez teórica y control del riesgo** que ofrecen los modelos de optimización matemática como el de Markowitz, especialmente útil para tomar decisiones de asignación de capital.

3.9.2. Implementación de N-BEATS en la optimización de carteras

De entre todos los modelos matemáticos que se proponen, dada la complejidad ya añadida de la inteligencia artificial que captura de manera correcta las tendencias, se usará el modelo de Markowitz para la composición de la cartera a partir de los retornos de las predicciones.

Preprocesamiento de datos

La implementación de N-BEATS comienza con un preprocesado de datos, para poder dejar dichos datos preparados para los cálculos. Esta implementación cuenta con escalar los datos para que se encuentren todos en el mismo grado y puedan ser comparables.

Entrenamiento

Posteriormente, podremos entrenar el modelo, dando una ventana de entrada de datos, una ventana de salida de datos, el número de épocas *(número de veces que se alimenta al modelo)*, un número que actuará como semilla (se refiere a un número que se toma como referencia de partida para permitir que cada ejecución comience con los mismos datos dando resultados reproducibles, y por ello, haciendo que el modelo sea determinista) y el *pl_trainer_kwargs*, para aplicar técnicas y permitir el uso de la tarjeta gráfica **si se tiene**.

Como pequeño inciso, para hacer uso de la tarjeta gráfica **no será válido únicamente tenerla** en nuestro dispositivo, los requisitos son que dicha tarjeta tenga **soporte CUDA** (las

tarjetas NVIDIA comúnmente contienen soporte CUDA, las tarjetas gráficas de AMD no están soportadas directamente, al menos no de forma estable), PyTorch (la librería de Python que se utiliza para la ejecución del modelo) con soporte CUDA, y de manera recomendable CUDA Toolkit para asegurar la correcta detección por parte de PyTorch de la GPU.

A continuación se muestra el código de entrenamiento del modelo:

```
def train_model(self, input_chunk_length=60,
   output_chunk_length=30, epochs=100, val_size=90):

    # Si no se han preparado los datos para el algoritmo de
   inteligencia artificial
    if self.series is None:
        raise ValueError(f"Series for {self.name} has not been
   prepared.")

    # Si no hay datos suficientes para los datos de
   entrenamiento y validación
    if len(self.series) < input_chunk_length +
   output_chunk_length:
        raise ValueError(f"Series for {self.name} is too short
   ({len(self.series)} values) to train any model.")

    # Si hay suficientes datos para separar validación
    if len(self.series) > val_size + input_chunk_length:
        train, val = self.series[:-val_size], self.series[-
   val_size:]
        use_val = True
    else:
        train = self.series
        val = None
```

```
            use_val = False
            print(f"[{self.name}] Not enough data for validation.
Training with full dataset.")

        # Metrica Early Stopping (previene sobreentrenamiento, se
explica después)
        early_stopper = EarlyStopping(
            monitor="val_loss",
            patience=10,
            mode="min"
        )

        # Entrenamiento del modelo
        print(f"Training model for {self.name}...")
        self.model = NBEATSModel(
            input_chunk_length=input_chunk_length,
            output_chunk_length=output_chunk_length,
            n_epochs=epochs,
            random_state=42,
            pl_trainer_kwargs={
                "accelerator": "gpu" if torch.cuda.is_available()
else "cpu",
                "callbacks": [early_stopper],
                "enable_progress_bar": True,
            }
        )

        print(f"Training model for {self.name}...")

        if use_val:
            self.model.fit(train, val_series=val)
```

```
    else:
        self.model.fit(train)
    print(f"Training completed for {self.name}.")

    # Guardamos el modelo de cada moneda
    os.makedirs("./models", exist_ok=True)  # Crea el
   directorio si no existe

    self.model.save(f"./models/{self.name}_nbeats_model.pth")
    print(f"Model saved for {self.name}.")
```

Predicción de los precios

Una vez procesados los datos, y entrenado el modelo, podremos realizar la predicción de los precios futuros.

```
def predict(self, n_future=365):
    if self.model is None:
        raise ValueError(f"The model for {self.name} has not
   been trained.")
    forecast = self.model.predict(n_future)
    forecast = self.scaler.inverse_transform(forecast)
    return forecast
```

Gracias a este proceso, podremos construir una estructura de datos que contenga los distintos precios que ha predecido el modelo para cada activo.

```
def predict_and_export_to_excel(self, n_future=365, filename="
   predicted_prices.xlsx"):
    self.predict_all()

    # Crear un DataFrame para almacenar las predicciones
```

```
    df_predictions = pd.DataFrame()

    for asset in self.assets:
        forecast = asset.predict(n_future)
        forecast_df = forecast.pd_dataframe()
        df_predictions[asset.name] = forecast_df[asset.name]

    df_predictions.to_excel(filename, index=False)
    print(f"Predicciones guardadas en {filename}")
    return df_predictions
```

Asignamos estos precios a una variable del objeto Portfolio:

```
self.df_prices_nbeats = df_predictions
```

Por último, calculamos los retornos de los precios que el modelo ha precedido, construyendo un vector dinámico de rendimientos esperados. Ahora que tenemos dicho vector de rendimientos, sí que podemos alimentar a un modelo matemático como los vistos anteriormente con esta información, combinando los modelos matemáticos clásicos con un enfoque novedoso de inteligencia artificial, permitiendo así capturar tendencias y poder ofrecer un mejor rendimiento.

```
# Ahora calculamos los retornos de las predicciones de N-BEATS
df_returns_nbeats = self.get_returns_from_prices(self.
   df_prices_nbeats)

# Ahora aplicamos Markowitz a las predicciones de N-BEATS
self.df_nbeats = self.markowitz_function(df_returns_nbeats,
   l_riesgo)
```

3.9.3. Parámetros clave en el entrenamiento del modelo N-BEATS

El modelo **N-BEATS** (*Neural Basis Expansion Analysis for Time Series Forecasting*) requiere definir una serie de **parámetros clave** que afectan directamente al rendimiento y a la capaci-

dad de generalización del modelo. A continuación se explican los más relevantes:

- **Ventana de entrada (input window):** define cuántos valores pasados de la serie temporal se utilizan como entrada del modelo para predecir el futuro. Por ejemplo, si se establece una ventana de entrada de 30, el modelo usará los últimos 30 valores históricos para generar la predicción. Una ventana pequeña puede impedir capturar patrones relevantes, mientras que una ventana demasiado grande puede introducir ruido y aumentar la complejidad del modelo.

- **Ventana de salida (forecast horizon):** indica cuántos pasos en el futuro se desea predecir. Por ejemplo, si se establece en 5, el modelo tratará de prever los próximos 5 valores. Esta variable define el *horizonte temporal de la predicción*, y debe elegirse en función de los objetivos del estudio (por ejemplo, predicción diaria, semanal, etc.).

  Estas **ventanas deslizantes** (sliding windows) usadas durante el entrenamiento significan que, a lo largo de un año de datos (por ejemplo, todos los precios de 2023), se generan múltiples pares entrada-salida recorriendo los datos día a día:

  - *Ejemplo 1:* entrada con días 1 a 60, salida con días 61 a 90.
  - *Ejemplo 2:* entrada con días 2 a 61, salida con días 62 a 91.
  - ...
  - *Ejemplo n:* entrada con días 275 a 334, salida con días 335 a 364.

- **Validación:** durante el entrenamiento, se divide el conjunto de datos en un conjunto de *entrenamiento* y otro de *validación*. Este último no se utiliza para ajustar los pesos del modelo, sino para evaluar su rendimiento en datos no vistos. Esto permite controlar si el modelo está aprendiendo patrones generales o simplemente memorizando los datos.

```
"""Si hay datos suficientes para la validacion"""
if len(self.series) > val_size + input_chunk_length:
    train, val = self.series[:-val_size], self.series[-
   val_size:]
    use_val = True
else:
```

```
    train = self.series
    val = None
    use_val = False
    print(f"[{self.name}] Not enough data for validation.
  Training with full dataset.")
```

- **Épocas (epochs):** una época representa una pasada completa por todo el conjunto de entrenamiento. El modelo se entrena durante múltiples épocas con el fin de minimizar el error de predicción. No obstante, un número elevado de épocas puede provocar que el modelo se sobreajuste a los datos de entrenamiento, en otras palabras, el algoritmo memorizalos datos de entrenamiento en vez de encontrar patrones en los datos.

- **Early stopping (detención temprana):** es una técnica que interrumpe el entrenamiento si el rendimiento en el conjunto de validación deja de mejorar durante un número determinado de épocas consecutivas. Esta práctica tiene varias ventajas:

  - Previene el *sobreajuste* del modelo.
  - Ahorra *tiempo de cómputo* al evitar continuar con el entrenamiento innecesariamente.
  - Mejora la *capacidad de generalización* del modelo en datos futuros.

```
"""
monitor="val_loss": Indica que se observará la métrica
   llamada val_loss (métrica de error).
patience=10: El entrenamiento se detendrá si val_loss no
   mejora después de 10 validaciones consecutivas.
mode="min": El entrenamiento busca minimizar la métrica
   val_loss (es decir, detenerse cuando deje de disminuir).
"""

early_stopper = EarlyStopping(
```

```
        monitor="val_loss",
        patience=10,
        mode="min"
    )
```

La correcta configuración de estos parámetros es esencial para lograr predicciones precisas y útiles que sirvan de base en la posterior optimización de carteras.

4
Resultados

4.1. Distribución de pesos

Una vez ejecutado el código, este nos devolverá la cantidad de pesos que deberíamos de haber invertido para cada activo. Para el modelo de Markowitz, CVaR Markowitz y Diversified Markowitz se ha utilizado unos valores de lambda y delta intermedios, que reflejan un índice de riesgo y diversificación medio, aunque se mostrará una tabla donde compararemos los distintos valores de estos.

En el caso del cálculo de CVaR para los posteriores resultados que se muestran, podremos observar activos que tienen un valor casi residual por su cercanía tal al 0. El optimizador distribuye el peso según los retornos esperados y el riesgo, y si hay activos que no aportan valor al objetivo, simplemente no se les asigna prácticamente nada. Se siguen cumpliendo las restricciones de no haber ventas en corto (pesos negativos significativos) y se respeta la restricción de suma de pesos igual a 1. Concluyendo que son residuos típicos del cálculo numérico.

4.2. Resultados de retorno por modelo

Una vez que obtenemos cuál es el porcentaje a invertir en cada activo, supondremos un capital inicial de 1000.

Este capital de 1000 se distribuirá tal y como dicta cada modelo y se enfrentarán los resultados de los retornos para ver cuál habría sido la mejor opción.

Cogemos el último valor diario de cierre del año 2023, año el cual hemos utilizado para el cómputo de los modelos, como precio de compra para la inversión en cada activo.

BTC-USDT	**ETH-USDT**	**XRP-USDT**	**ADA-USDT**	**SOL-USDT**
16616.5	1200.54	0.33876	0.24995	9.986
BNB-USDT	**DOT-USDT**	**AVAX-USDT**	**DOGE-USDT**	**SHIB-USDT**
244.4	4.372	10.862	0.07022	0.00000813

Cuadro 3: Precios de cierre del año de 2023 de las criptomonedas

Recordamos que cuanto mayor sea λ mayor será la rentabilidad que busquemos, se busca la rentabilidad por encima del riesgo, incluso llegando a eliminar el riesgo por completo. Esta faceta la caracterizamos como un inversor más agresivo cuanto más aumenta el valor λ. Por otra parte, cuanto menor sea dicho valor, estaremos priorizando una inversión con menos riesgo; si es muy pequeña o cero, se priorizará menor riesgo antes que rentabilidad.

4.3. Simulación de resultados

Para enfrentar el rendimiento de los distintos modelos, dichos modelos son calculados para el año 2023, y puestos a prueba con datos de 2024. Para ello, en el código, la función *simulate_portfolio_growth* permite la simulación del rendimiento individualmente, mostrando un gráfico del valor de la cartera diario por cada modelo usado y exportando dicha imagen como archivo, bajo el nombre de *portfolio_evolution* con formato PNG. Además, se devuelve el valor final de la cartera para cada modelo en un archivo Excel, llamado *final_portfolio_values.xlsx*.

```
def simulate_portfolio_growth(self, initial_capital=1000):
    if self.df_prices_test.empty:
```

```
        raise ValueError("df_prices_test está vacío. Ejecuta
primero get_prices().")

    model_dfs = {
        'Markowitz': self.df_markowitz,
        'CVaR Markowitz': self.df_cvar_markowitz,
        'Equally Weighted': self.df_ew,
        'Diversified Markowitz': self.df_diversified_markowitz,
        'N-BEATS': self.df_nbeats
    }

    portfolio_values = {}
    final_values = {}

    for model_name, df_weights in model_dfs.items():
        if df_weights.empty:
            continue

        # Indice como los nombres de los activos
        df_weights.index = list(self.df_prices_test.columns)
        weights = df_weights.iloc[:, 0]

        prices = self.df_prices_test[weights.index]

        initial_prices = prices.iloc[0]
        quantities = (weights * initial_capital) /
initial_prices

        # Calculamos el valor de la cartera en cada momento
        portfolio_value = prices.multiply(quantities, axis=1).
sum(axis=1)
```

```
        portfolio_values[model_name] = portfolio_value

        final_values[model_name] = portfolio_value.iloc[-1]

    # Mostrar valores finales por consola
    print("Valor final de cada cartera:")
    for model_name, final_val in final_values.items():
        print(f"{model_name}: {final_val:.2f} USDT")

    # Guardar a Excel
    df_final = pd.DataFrame.from_dict(final_values, orient='
index', columns=['Final Portfolio Value'])
    df_final.to_excel("final_portfolio_values.xlsx")

    # Graficamos
    plt.figure(figsize=(12, 6))
    for model_name, values in portfolio_values.items():
        plt.plot(values.index, values.values, label=model_name)

    plt.title("Evolución del valor de la cartera según el
modelo")
    plt.xlabel("Fecha")
    plt.ylabel("Valor de la cartera")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("portfolio_evolution.png", dpi=300)
    plt.show()
```

Ejemplo de un valor alto de $\lambda = 5$:

Como adelanto a las alternativas y casos siguientes, esta es la que mejores resultados da, también cabe destacar que hemos asumido mayor riesgo por el uso de un valor alto de λ. Considero interesante mostrar el comportamiento de dichos modelos con un valor de λ elevado, puesto que un valor tan alto de λ, priorizará la rentabilidad por encima de todo. Al ejecutar las simulaciones, el modelo devuelve estos porcentajes:

Coins	**Markowitz**	**CVaR Markowitz**	**Equally Weighted**	**Diversified Markowitz**	**N-BEATS**
BTC-USDT	0	4.87053E-16	0.1	0	0
ETH-USDT	0	2.48437E-16	0.1	0	0.3356
XRP-USDT	0	1.05005E-16	0.1	0	0.4772
ADA-USDT	0	4.14364E-16	0.1	0	0
SOL-USDT	1	1	0.1	1	0.1872
BNB-USDT	0	8.22537E-17	0.1	0	0
DOT-USDT	0	3.62395E-16	0.1	0	0
AVAX-USDT	0	1.66533E-16	0.1	0	0
DOGE-USDT	0	1.0202E-16	0.1	0	0
SHIB-USDT	0	8.28959E-17	0.1	0	0

Cuadro 4: Pesos de cada activo en la cartera por cada estrategia (λ alto)

Recalcar que no debemos preocuparnos por los resultados con valores tan pequeños, ya que son residuos del cómputo de los modelos y no poseen peso alguno. La siguiente tabla corresponde al resultado final, una vez terminado el año 2024, si hubiéramos hecho dicha inversión conforme a los porcentajes que hemos visto y un capital inicial de 1000$.

	Final Portfolio Value (USD)
N-BEATS	2378.56
Equally Weighted	1920.992109
Markowitz	1721.974603
CVaR Markowitz	1721.974603
Diversified Markowitz	1721.974603

Cuadro 5: Valores finales de la cartera por cada estrategia

Como podemos observar, este resultado es el mismo para los modelos de Markowitz, CVAR Markowitz y Diversified Markowitz, ya que el valor de λ es tan grande que se prioriza por encima de otras variables que pueda tener el modelo.

Si lo graficáramos, podremos apreciar que tenemos únicamente a la vista los modelos de Equally Weighted porque tiene valores únicos, y el Diversified Markowitz, que a pesar de tener los mismos valores que Markowitz y CVaR Markowitz, este se representa puesto que es el último modelo de estos en ser representado.

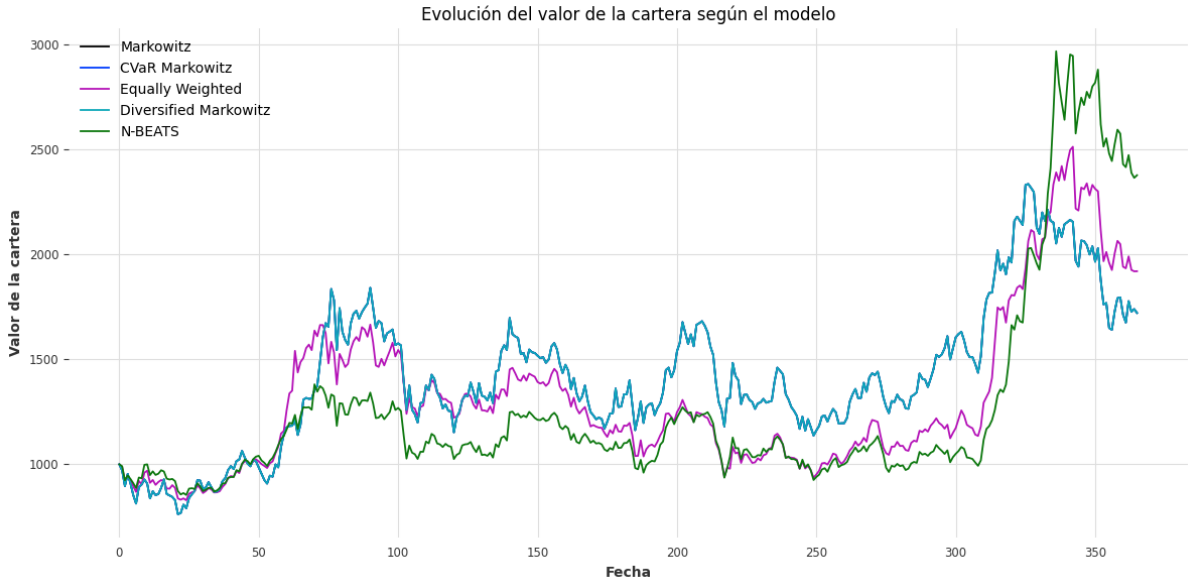

Figura 1: Visualización de crecimiento de Portfolio para un valor alto de lambda

Ejemplo de un valor bajo de $\lambda = 0{,}1$:

Para este caso, podemos apreciar que el valor de lambda del riesgo es tan bajo que por ejemplo, el CVaR se comportará como un Equally Weighted. Usamos $\delta = 0.01$.

Coins	Markowitz	CVaR Markowitz	Equally Weighted	Diversified Markowitz	N-BEATS
BTC-USDT	0.8816	0.1	0.1	0.1250	0.3531
ETH-USDT	0.0001	0.1	0.1	0.1130	0.2126
XRP-USDT	0.0003	0.1	0.1	0.0933	0.1048
ADA-USDT	0	0.1	0.1	0.0939	0.0154
SOL-USDT	0.1165	0.1	0.1	0.1057	0.1224
BNB-USDT	0.0014	0.1	0.1	0.1108	0.0330
DOT-USDT	0	0.1	0.1	0.0901	0
AVAX-USDT	0.0001	0.1	0.1	0.0922	0
DOGE-USDT	0	0.1	0.1	0.0886	0.0432
SHIB-USDT	0	0.1	0.1	0.0874	0.1156

Cuadro 6: Pesos del portfolio para cada moneda y modelo (Valor bajo de λ)

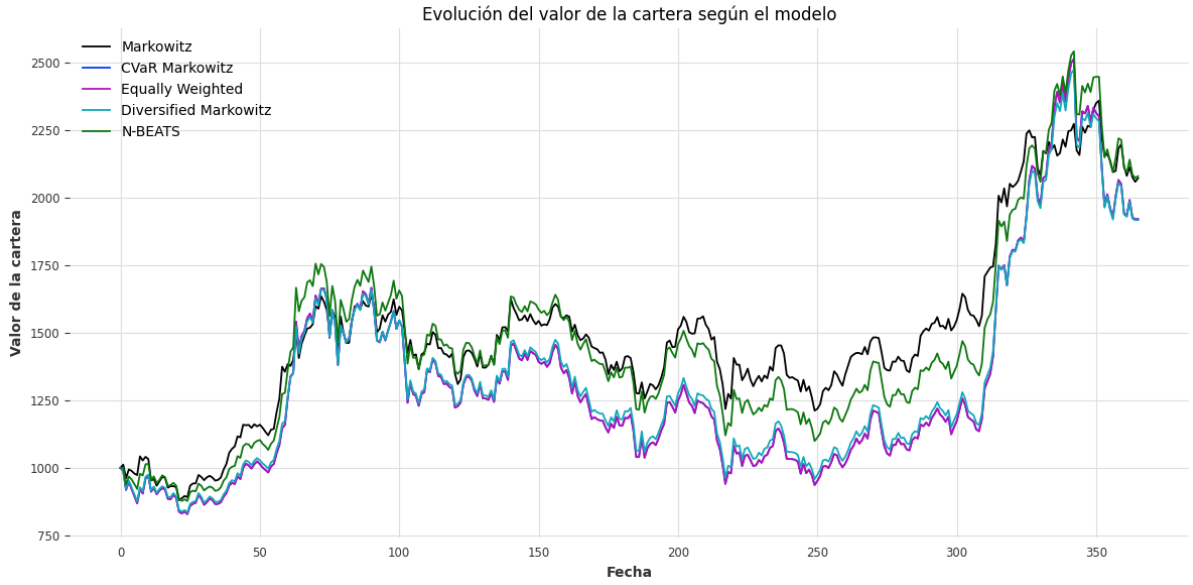

Figura 2: Visualización de crecimiento de Portfolio para un valor bajo de lambda

Esto ocurre por cómo se gestionan la importancia de las variables en cada modelo matemático, aún más si hablamos de diferencias tan significativas.

4.3.1. Distintos valores de delta para la gestión del riesgo

Recordamos que cuanto mayor sea δ mayor será la importancia de la diversificación del capital entre todos los activos que tenemos, por contrario, cuanto más baja, menor será la importancia de dicha diversificación. Veremos resultados prácticos, sin embargo, en la teoría ya podemos dar por hecho que valores extremos positivos estaremos ante un símil del modelo Equally Weighted.

Ejemplo de un valor alto de $\delta = 3$:

Para un valor intermedio como $\lambda = 0{,}25$, y un valor alto de δ, tenemos:

Coins	Markowitz	CVaR Markowitz	Equally Weighted	Diversified Markowitz	N-BEATS
BTC-USDT	0.5287	1.63064E-16	0.1	0.1001	0.368
ETH-USDT	0	1.10697E-16	0.1	0.1	0.2266
XRP-USDT	0	0	0.1	0.0999	0.114
ADA-USDT	0	1.95156E-17	0.1	0.1	0.0002
SOL-USDT	0.4711	0.807051417	0.1	0.1002	0.1386
BNB-USDT	0	0	0.1	0.1	0.0397
DOT-USDT	0	0	0.1	0.0999	0
AVAX-USDT	0.0001	0.192948583	0.1	0.1	0
DOGE-USDT	0	0	0.1	0.0999	0.0121
SHIB-USDT	0	0	0.1	0.0999	0.1009

Cuadro 7: Pesos del portfolio para cada moneda y modelo (Valor alto de λ)

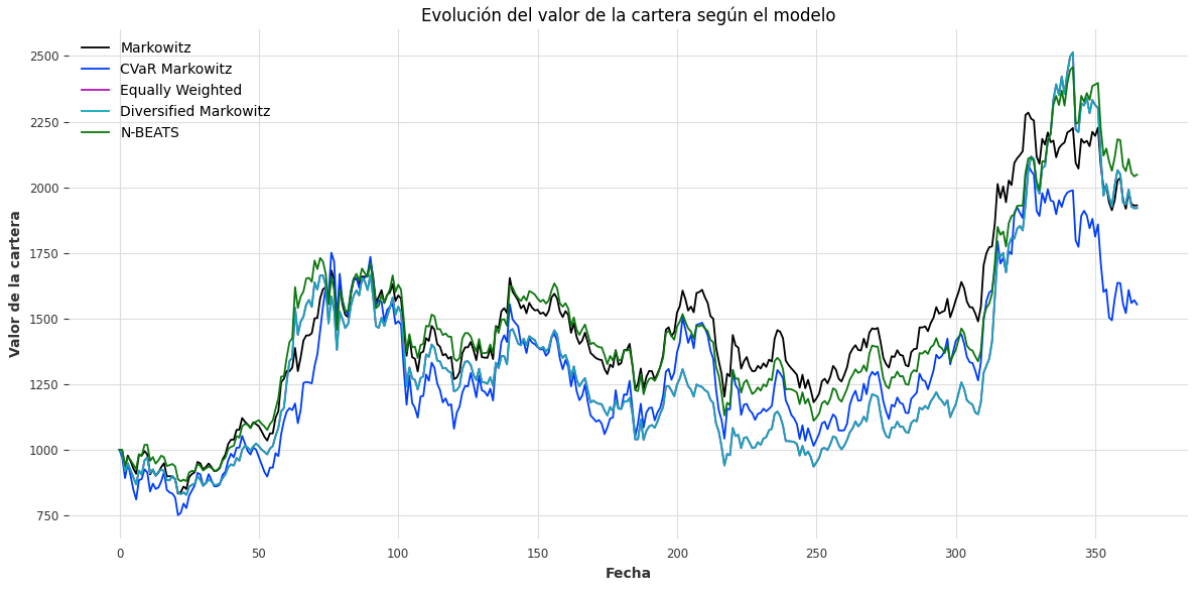

Figura 3: Portfolio allocation visualization for high delta value

Como se puede observar, el valor δ es mucho mayor respecto al valor λ, por lo que este dictamina casi en su totalidad el resultado.

Ejemplo de un valor bajo de $\delta = 0{,}001$:

Mantenemos el valor de $\lambda = 0{,}25$, con $\delta = 0{,}001$. Los resultados son:

Coins	Markowitz	CVaR Markowitz	Equally Weighted	Diversified Markowitz	N-BEATS
BTC-USDT	0.5287	1.63064E-16	0.1	0.3356	0.368
ETH-USDT	0	1.10697E-16	0.1	0.1181	0.2266
XRP-USDT	0	0	0.1	0.0129	0.114
ADA-USDT	0	1.95156E-17	0.1	0.0478	0.0002
SOL-USDT	0.4711	0.807051417	0.1	0.3814	0.1386
BNB-USDT	0	0	0.1	0.0008	0.0397
DOT-USDT	0	0	0.1	0	0
AVAX-USDT	0.0001	0.192948583	0.1	0.1033	0
DOGE-USDT	0	0	0.1	0	0.0121
SHIB-USDT	0	0	0.1	0	0.1009

Cuadro 8: Pesos del portfolio para cada moneda y modelo (Valor bajo de δ)

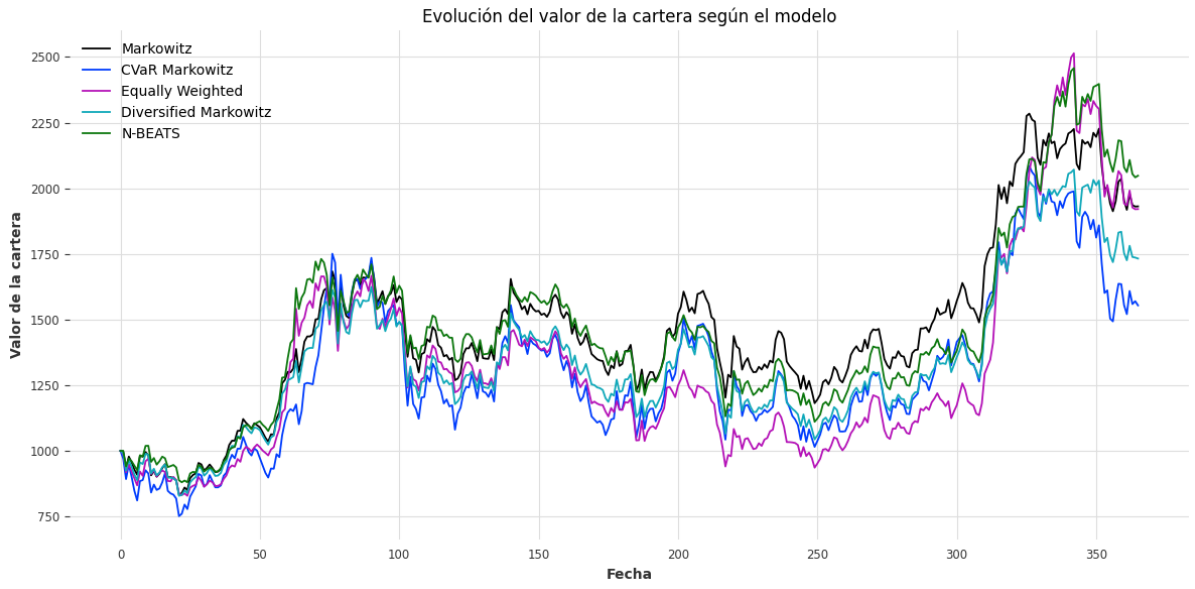

Figura 4: Visualización de crecimiento de Portfolio para un valor bajo de delta

El valor delta es tan pequeño que no posee casi relevancia en el cómputo matemático.

4.3.2. Resultados finales

Una vez concluido, y estudiado cómo los distintos valores de las variables en los extremos pueden dar comportamientos anómalos, para valores muy altos de δ la diversificación toma el control por encima de λ, y para valores muy altos de δ viceversa.

Continuando con la filosofía de valores medios que escribimos al principio de este apartado de *Resultados*, mostremos en la tabla 9 los pesos que hemos elegido finalmente para cada uno de los activos y modelos. Además, una vez detallados estos porcentajes para nuestra cartera, observaremos como evolucionan en el año de prueba en la figura 5, y el rendimiento final de la cartera para cada modelo:

Coins	**Markowitz**	**CVaR Markowitz**	**Equally Weighted**	**Diversified Markowitz**	**N-BEATS**
BTC-USDT	0.5287	1.63064E-16	0.1	0.1316	0.368
ETH-USDT	0	1.10697E-16	0.1	0.1078	0.2266
XRP-USDT	0	0	0.1	0.0865	0.114
ADA-USDT	0	1.95156E-17	0.1	0.0968	0.0002
SOL-USDT	0.4711	0.807051417	0.1	0.1608	0.1386
BNB-USDT	0	0	0.1	0.0901	0.0397
DOT-USDT	0	0	0.1	0.0827	0
AVAX-USDT	0.0001	0.192948583	0.1	0.1089	0
DOGE-USDT	0	0	0.1	0.068	0.0121
SHIB-USDT	0	0	0.1	0.0668	0.1009

Cuadro 9: Distribución de pesos en diferentes modelos de inversión

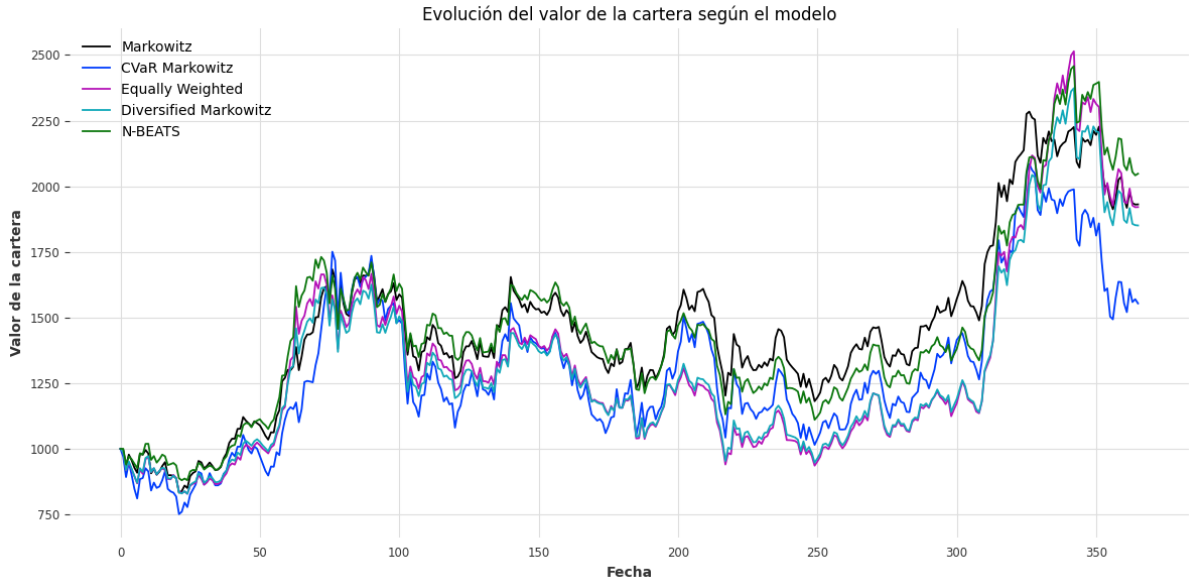

Figura 5: Visualización del crecimiento del portfolio

A nivel gráfico, vemos un crecimiento en su mayoría lineal sin demasiados mínimos y máximos anómalos, y a su vez un buen beneficio, pudiendo concluir que aborda buenos resultados. Comenzando con una inversión inicial de 1000$, obtenemos en el peor caso, que se retorna finalmente la cantidad de 1553.97$ tomando el modelo CVaR Markowitz, y en los mejores casos el modelo de N-BEATS y Markowitz con un retorno total de 2048.08$ y 1931.23$ respectivamente.

Estrategia	**Valor final (USD)**	**Beneficio (USD)**	**Beneficio (%)**
N-BEATS	2048.08	1048.08	104.81 %
Markowitz	1931.23	931.23	93.12 %
Equally Weighted	1920.99	920.99	92.10 %
Diversified Markowitz	1850.90	850.90	85.09 %
CVaR Markowitz	1553.97	553.97	55.40 %

Cuadro 10: Valor final, beneficio y rentabilidad porcentual con inversión inicial de 1000 USD

Con estos resultados queda concluido cómo los modelos matemáticos son capaces de adaptarse a entornos volátiles como en estos activos de estudio, las criptomonedas, a parte de poder ser una herramienta muy potente combinada con algoritmos de inteligencia artificial como en este caso, el uso del algoritmo N-BEATS.

5

Conclusions and Futures Lines of Research

5.1. Conclusions

Throughout this work, various mathematical models have been explored that successfully address the problem of optimal asset allocation in an investment portfolio, aiming to maximize expected return under different levels of risk and diversification. The models analyzed include the classical **Markowitz** model, its variant based on **CVaR** (*Conditional Value at Risk*), a diversified version of the former, and an **Equally Weighted** strategy as a baseline to assess the importance of diversification.

It has been observed how the parameters λ (related to risk) and δ (related to diversification) directly influence the final composition of the portfolio. High values of λ prioritize profitability over risk, often causing the model to concentrate the entire investment in a single asset (frequently the most historically profitable), while low values result in a more conservative capital distribution. On the other hand, a high δ value **enforces** greater diversification among assets, sometimes leading to an almost equal-weight distribution.

In addition to evaluating weight allocation, the **N-BEATS** model (*Neural basis expansion analysis for interpretable time series forecasting*) has been incorporated. This model is based on deep neural networks designed for time series forecasting and allows anticipating the future behavior of cryptocurrency prices. This approach enables portfolio allocation not only based on historical data but also by incorporating a **forecast of future price trends**.

The inclusion of **N-BEATS** has proven advantageous by enabling the use of forecasted prices as inputs for the portfolio optimization models. This has shown adequate performan-

ce and opens an interesting avenue for combining these mathematical models with artificial intelligence in portfolio management.

As a final result, it has been demonstrated that the return of an initial investment of **1000** USD can vary significantly depending on the chosen model and parameter configuration. More aggressive models showed higher returns in contexts of strong growth of specific assets, whereas diversified strategies offered greater stability and risk control.

Additionally, throughout the development of the project, several key technical skills have been acquired and put into practice for the implementation of the described models. These include the use of data structures such as **dataframes**, which are fundamental for efficiently handling large volumes of structured information; **Excel file management** for exporting and analyzing results; **automatic graph generation**, which has enabled clear visualization of portfolio behavior and predictions; and **data extraction and cleaning**, essential tasks for ensuring the quality and consistency of the datasets used.

Furthermore, work has been done on **timestamp management** to properly organize the temporal information of asset prices; **quadratic problem** solving as the basis for optimizing the Markowitz model; and a deep understanding of **neural network theory**, which is crucial for leveraging the full potential of **N-BEATS** in financial time series forecasting.

Finally, a **graphical user interface (GUI)** has been developed to connect all the implemented code with an intuitive and accessible visual environment. This interface not only facilitates interaction with the various models and parameters but also enables clear visualization of results, export of such data into images and Excel files, and exploration of different investment scenarios without requiring advanced technical knowledge. The integration of the GUI is considered a key milestone in the creation of a practical and user-friendly tool.

This set of technical competencies has been essential for coherently integrating the different components of the project and achieving solid, reproducible results.

5.2. Future lines of Research

Based on the work developed, multiple avenues for further research and development arise, which could enrich and expand the results obtained:

- **Exploration of other forecasting models**: Although N-BEATS has shown good per-

formance, it would be interesting to compare its effectiveness with other advanced time series models, such as convolution-based models like TCN (Temporal Convolutional Networks). This would help evaluate which model better fits the specific dynamics of the cryptocurrency market.

- **Incorporation of alternative data**: Including unstructured data such as news, social media, sentiment analysis, or macroeconomic indicators could enhance the predictive capabilities of the machine learning models used, providing a more comprehensive market view.

- **Simulation of adverse scenarios**: Including mechanisms for financial stress testing or crisis simulations would allow assessing the robustness of the portfolios in extreme contexts and improve the models based on resilience or crisis response criteria.

- **Real-time implementation**: A practical future application could involve developing a tool for automatically updating predictions and re-optimizing portfolios in real time, integrating live data sources and automating decision-making. This could be further enhanced with the integration of market sentiment data such as news and commercial agreements, allowing for more accurate coverage of potential reversals or price surges.

These research directions represent opportunities to further develop increasingly sophisticated investment strategies that combine the best of quantitative analysis, AI-based forecasting, and comprehensive risk management.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

A lo largo de este trabajo se han explorado diferentes modelos matemáticos que abordan de manera satisfactoria el problema de la asignación óptima de activos en una cartera de inversión, maximizando el retorno esperado bajo distintos niveles de riesgo y diversificación. Entre los modelos analizados se encuentran el clásico modelo de **Markowitz**, su variante basada en el **CVaR** (*Conditional Value at Risk*), una versión diversificada del primero, y una estrategia de pesos iguales (**Equally Weighted**) como referencia para probar la importancia de la diversificación.

Se ha observado cómo los parámetros λ (*relativo al riesgo*) y δ (*relativo a la diversificación*) influyen de manera directa en la composición final de la cartera. Valores altos de λ priorizan la rentabilidad sobre el riesgo, provocando en muchos casos que el modelo concentre toda la inversión en un solo activo (*a menudo el más rentable históricamente*), mientras que valores bajos distribuyen el capital de una forma más conservadora. Por otro lado, un δ alto, **fuerza** una diversificación más marcada entre activos, conllevando en casos extremos a una diversificación ecuánime.

Además de evaluar la asignación de pesos, se ha incorporado el modelo **N-BEATS** (*Neural basis expansion analysis for interpretable time series forecasting*), una arquitectura basada en redes neuronales profundas diseñada para la predicción de series temporales, con el fin de anticipar el comportamiento futuro de los precios de las criptomonedas. Este enfoque ha permitido no solo basar la asignación de pesos en los datos históricos que podemos obtener, sino además, incorporar una **estimación de la evolución de los precios a futuro**.

La inclusión de **N-BEATS** ha supuesto una ventaja al permitir el uso de precios previstos como entrada para los modelos de optimización de cartera. Se puede observar un rendimiento adecuado y plantea una vía interesante para utilizar tanto dichos modelos matemáticos para la gestión de una cartera, como el uso de la inteligencia artificial.

Como resultado final, se ha evidenciado que el rendimiento de una inversión inicial de **1000** dólares puede variar de manera significativa dependiendo del modelo escogido y los parámetros utilizados. Modelos más agresivos han mostrado mayores retornos en contextos de fuerte crecimiento de algunos activos, mientras que estrategias diversificadas han aportado mayor estabilidad y control del riesgo.

Adicionalmente, a lo largo del desarrollo del proyecto se han adquirido y puesto en práctica diversas habilidades técnicas clave para la implementación de los modelos descritos. Entre ellas se incluyen el uso de estructuras de datos como **dataframes**, fundamentales para el manejo eficiente y estructurado de grandes volúmenes de información; el **manejo de archivos Excel** para la exportación y análisis de resultados; la **generación automática de gráficos**, lo cual ha permitido visualizar de manera clara el comportamiento de las carteras y predicciones; y la **extracción y limpieza de datos**, tareas esenciales para asegurar la calidad y coherencia de los conjuntos de datos utilizados.

Asimismo, se ha trabajado con el **manejo de estampas de tiempo** para organizar correctamente la información temporal de los precios; la **resolución de problemas cuadráticos** como base para la optimización del modelo de Markowitz; y un **profundo aprendizaje de la teoría de redes neuronales**, indispensable para explotar el potencial de N-BEATS en la predicción de series temporales financieras.

Finalmente, se ha desarrollado una **interfaz gráfica de usuario (GUI)** que permite conectar todo el código implementado con un entorno visual intuitivo y accesible. Esta interfaz no solo facilita la interacción con los distintos modelos y parámetros, sino que también permite visualizar los resultados de forma clara, la exportación de dichos datos en imágenes y archivos Excel, explorando distintos escenarios de inversión sin necesidad de conocimientos técnicos avanzados. Se considera que la incorporación de la GUI es un pilar fundamental hacia la creación de una herramienta práctica y usable.

Este conjunto de competencias técnicas ha sido esencial para integrar de forma coherente los distintos componentes del proyecto y obtener resultados sólidos y replicables.

6.2. Líneas Futuras

A partir del trabajo desarrollado, se abren múltiples líneas de investigación y desarrollo que pueden enriquecer y ampliar los resultados obtenidos:

- **Exploración de otros modelos de predicción**: aunque N-BEATS ha mostrado un buen rendimiento, sería interesante comparar su eficacia con otros modelos avanzados de series temporales o modelos basados en convoluciones como TCN (Temporal Convolutional Networks). Esto permitiría evaluar cuál se adapta mejor a las dinámicas específicas del mercado de criptomonedas.

- **Incorporación de datos alternativos**: la inclusión de datos no estructurados como noticias, redes sociales, análisis de sentimiento o indicadores macroeconómicos podría mejorar la capacidad predictiva de los modelos de machine learning utilizados, proporcionando una visión más completa del mercado.

- **Simulación de escenarios adversos**: incluir mecanismos de estrés financiero o simulación de crisis (stress testing) permitiría comprobar la robustez de las carteras en contextos extremos y mejorar los modelos bajo criterios de resiliencia o crisis.

- **Implementación en tiempo real**: una futura aplicación práctica consistiría en desarrollar una herramienta que permita la actualización automática de predicciones y la reoptimización de carteras en tiempo real, integrando fuentes de datos en vivo y ejecutando decisiones de forma automatizada. Este punto podría ir de la mano con la inclusión de datos anexos al sentimiento de mercado como son las noticias, acuerdos comerciales entre otros, pudiendo abarcar de manera más precisa posibles reversiones o repuntes de los precios.

Estas líneas representan una oportunidad para seguir desarrollando estrategias de inversión cada vez más sofisticadas, que combinen lo mejor del análisis cuantitativo, la predicción mediante inteligencia artificial y la gestión del riesgo de forma integral.

Referencias

[1] Travis E Oliphant et al. *Guide to numpy*. Vol. 1. Trelgol Publishing USA, 2006.

[2] Wes McKinney et al. "pandas: a foundational Python library for data analysis and statistics". En: *Python for high performance and scientific computing* 14.9 (2011), págs. 1-9.

[3] Amit D.Kachare A.D.Dongare R.R.Kharde. "Introduction to Artificial Neural Network". En: *International Journal of Engineering and Innovative Technology (IJEIT)* (2012), pág. 1. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b6952a4f0c7203577afc9476c2fcab2cba06.

[4] L.R Medsker, L. Jain. *Recurrent Neuronal Networks Design and Applications*. The CRC Press International Series on Computational Intelligence, 2001. URL: https://d1wqtxts1xzle7.cloudfront.net/31279335/___Recurrent_Neural_Networks_Design_And_Applicatio%28BookFi.org%29-libre.pdf?1390940527=&response-content-disposition=inline%3B+filename%3DEffects_of_Liquid_and_Encapsulated_Lacti.pdf&Expires=1749375396&Signature=gsXtVxQcM27Yet70yRebkLE559vMTwDenb0y1h1HFy88vDU5lJxtApaA3RCo~LeICNVBGBvIeO2MXdJhVrX2aXTAv1a7mgSCF3yrfb10SlOGFv6lqX7DRI1yqWHkS8WtL6zk9GfPQxISAXDuoj7fp2jWew~k6vLJWj89Y4~tozqNYo1uEBRMckWMuMwl~Njt2emVDv~3YVaXn-vjcviW8PRrX-vhm0A0BNueF1OSFQBILXRSCdKsWyLXQjaKWBEz3I9YriHudTXMNscjSz0GjmB1GDiUwD-h2X-UgFp52H-Yeuj-Xok7w6uXjzs4-Cg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

[5] Yong Yu et al. "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures". En: *Neural Computation* 31.7 (jul. de 2019), págs. 1235-1270. ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01199.

[6] James McCaffrey. *Understanding LSTM Cells Using C#*. Accedido el 3 de marzo de 2025. 2018. URL: https://learn.microsoft.com/es-es/archive/msdn-magazine/2018/april/test-run-understanding-lstm-cells-using-csharp.

[7] Harry M. Markowitz. "Portfolio selection". En: *The Journal of Finance* (1952), págs. 77-91. URL: https://www.math.hkust.edu.hk/~maykwok/courses/ma362/07F/markowitz_JF.pdf.

[8] Francisco J. Tejedor Tejedor. *Análisis de varianza : introducción conceptual y diseños básicos*. spa. Cuadernos de estadística ; 3. Madrid: La Muralla, 1999. ISBN: 847635388x.

[9] Luis Ceferino Franco Arbeláez y Luis Eduardo Franco Ceballos. "El valor en riesgo condicional CVaR como medida coherente de riesgo". En: *Revista Ingenierías Universidad de Medellín* 4.6 (2005), págs. 43-54.

[10] R. González-Domínguez, D. Chushig-Muzo y C. Soguero-Ruíz. "Modelos basados en redes neuronales artificiales para la generación y predicción de series temporales de glucosa". En: *XLI Congreso Anual de la Sociedad Española de Ingeniería Biomédica* (2023), págs. 182-185. URL: https://repositorio.upct.es/server/api/core/bitstreams/10505aa4-3345-45c1-8e27-4740371a4955/content.

[11] Cambridge. *Cambridge Dictionary*. Cambridge, 2025. URL: https://dictionary.cambridge.org/es/diccionario/ingles/exchange.

[12] Cambridge. URL: https://dictionary.cambridge.org/dictionary/english/timestamp.

[13] Tether. URL: https://tether.to/en/.

Apéndice A
Manual de Instalación

A.1. Instalación de Python

Para la instalación, primeramente necesitaremos en el sistema tener instalado Python. Se puede descargar desde https://www.python.org/downloads/, en las pruebas se está utilizando la versión 3.12.4, sin embargo, en principio, se puede utilizar cualquier versión de Python 3.

Nota: en caso de ser usuario de Linux, posiblemente tenga instalado en su sistema Python, para comprobarlo ejecute un terminal y escriba "python", en cuyo caso no lo tenga instalado, ejecute "sudo apt updatez "sudo apt install python3"

Una vez tengamos instalado en nuestro sistema Python, podremos comprobarlo accediendo a *Símbolo del sistema* en caso de Windows, o *Terminal* en caso de usuario de Mac o Linux, una vez dentro escribiremos *python* y debería salirnos la versión y una entrada de comandos del intérprete de Python.

A.2. Instalación de librerías

Para poder ejecutar el programa, solo quedaría la instalación de las librerías correspondientes, para su fácil instalación, se aporta un archivo requirements que contendrá la versión y librerías a instalar, que de sencilla manera podrán ser instaladas a través del comando *"pip install -r requirements.txt"* en la ruta del proyecto.

A.3. Raíz del proyecto

- **Marketdata.py:** se ocupa de controlar las estampas de tiempo y las peticiones para obtener los precios de las monedas, además de operaciones con hojas de cálculo, como

almacenando dicha información en ellos.

- **Portfolio.py:** clase principal que se ocupa de la optimización del portfolio con todos los modelos matemáticos explicados y el entrenamiento y predicción de los precios con el algoritmo N-BEATS.

- **GUI.py:** se encarga de proporcionar una interfaz gráfica donde poder de manera intuitiva realizar la gestión de carteras.

- **Carpeta results:** esta carpeta, una vez ejecutada alguna simulación a través de la interfaz gráfica, incluirá en su nombre el riesgo y el año de la simulación. Dentro de ella:

  - *crypto_(año)_portfolio* contendrá los porcentajes para invertir por cada activo según el modelo.
  - *final_portfolio_values.xlsx* con la información del valor final de la cartera si hubiéramos realizado la inversión.
  - *portfolio_evolution.png* una imagen que representa un gráfico de crecimiento al probar cada modelo matemático.
  - *predicted_prices.png* los precios predichos por N-BEATS.
  - *rentabilidad_por_modelo.png* representa una gráfica de barras mostrando la rentabilidad de cada modelo.
  - Carpeta models: dicha carpeta contiene los modelos de entrenamiento de la red neuronal, para cada moneda.

A.4. Ejecución del programa

Para poder ejecutar el programa, abriendo una terminal en la misma ruta del proyecto, ejecutaremos "*python gui.py*". Nos saldrá una ventana con la interfaz gráfica del programa.

Como alternativa a la interfaz gráfica, se puede ejecutar "*python Portfolio.py*".

A.4.1. Ejemplo ejecución interfaz gráfica

Figura 6: Resultado de selección de preferencias y ejecución con interfaz gráfica (gui.py)

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA