



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

FAT

**Financial Analysis Tool.
Herramienta de análisis
financiero.**



Presentado por Rodrigo Merino Tovar
en Universidad de Burgos — 8 de junio de 2024
Tutores: Dra. Virginia Ahedo García y
Dr. José Ignacio Santos Martín



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



Dña. Virginia Ahedo García y D. José Ignacio Santos Martín, profesores del departamento de Ingeniería de Organización, área de Organización de Empresas.

Exponen:

Que el alumno D. Rodrigo Merino Tovar, con DNI 71286910C, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "FAT: Financial Analysis Tool. Herramienta de análisis financiero."

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de junio de 2024

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

Dña. Virginia Ahedo García

D. José Ignacio Santos Martín

Resumen

Invertir nuestro capital para obtener una rentabilidad a cambio puede ser un proceso costoso si no se dispone de un acceso sencillo a la información necesaria. Además, deberemos de tener claro cuáles son el objetivo y el horizonte de nuestra inversión, así como el riesgo que estamos dispuestos a asumir y si la información de la que disponemos es suficiente.

Para abordar algunos de estos retos, este trabajo propone una herramienta digital que recopila información técnica de empresas y sus productos cotizados, así como noticias relacionadas con algunos de los índices de referencia más relevantes. Esta herramienta presenta datos de forma agregada y permite realizar la comparación evolutiva - de precios de cierre de mercado - con el sector de referencia de una empresa cotizada o entre diferentes valores de distintos mercados.

Adicionalmente, en este trabajo se aporta una visión diferente a las webs de mercados financieros, permitiendo hacer uso de algunas utilidades poco comunes, como son la opción de hacer un análisis gráfico de rentabilidad-riesgo para los valores que tengamos en cartera, la experimentación con predicción automática sobre series temporales a través de modelos *ARIMA* o herramientas de *trading* algorítmico basado en técnicas de cruce de medias y en *machine learning*.

Por otro lado, esta herramienta permite llevar el control de una cartera de inversión - realizando cambio de divisa automático a euros en los casos necesarios - y disponer de una lista de seguimiento, con precios reales de cierre diario.

La herramienta se encuentra disponible en <http://takeiteasy.pythonanywhere.com/> o, para ser utilizada de forma local, en <https://github.com/rmt0009alu/FAT>.

Descriptores

Django, SQLite, análisis financiero, diversificación de cartera, *forecasting* de series temporales, *trading* algorítmico.

Abstract

Investing our capital to obtain a return can be a costly process if we don't have easy access to the necessary information. Additionally, we must be clear about the target and horizon of our investment, as well as the risk we are willing to assume and whether the information we have is sufficient.

To address some of these challenges, this work proposes a digital tool that collects technical and fundamental information about companies and their listed products, as well as news related to some of the most relevant indices. This tool presents data in an aggregated manner and allows for comparative analysis - using market closing prices - with the reference sector of a listed company or between different stocks from different markets.

Additionally, this work offers a different perspective on financial market websites, allowing the use of some uncommon utilities, such as the option to perform a graphical analysis of risk-return for securities in our portfolio, experimentation with automatic prediction on time series through *ARIMA* models or algorithmic *trading* tools based on mean crossing techniques and *machine learning*.

On the other hand, this tool allows for the control of an investment portfolio - including automatic currency conversion to euros when necessary - and a watchlist with real closing prices on a daily basis.

The tool is available at <http://takeiteasy.pythonanywhere.com/> or, to be used locally, in <https://github.com/rmt0009alu/FAT>.

Keywords

Django, SQLite, financial analysis, portfolio diversification, time series *forecasting*, algorithmic *trading*.

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
1. Introducción	1
1.1. Estructura de la memoria	2
1.2. Materiales adjuntos	3
2. Objetivos del proyecto	4
2.1. Objetivos generales	4
2.2. Objetivos de carácter técnico	4
2.3. Objetivos personales	5
3. Conceptos teóricos	6
3.1. Diversificación de una cartera de valores cotizados	6
3.2. Análisis de series temporales con modelos ARIMA	18
3.3. Trading algorítmico	33
4. Técnicas y herramientas	43
4.1. Técnicas metodológicas	43
4.2. Patrones de diseño	44
4.3. Control de versiones	45
4.4. Alojamiento del repositorio	45
4.5. Gestión del proyecto	46
4.6. Comunicación	46

4.7. Entorno de desarrollo integrado (IDE)	46
4.8. Documentación de la memoria	47
4.9. Documentación del código	47
4.10. Integración y despliegue continuos (CI/CD)	48
4.11. Calidad y consistencia de código	48
4.12. Cobertura de código	48
4.13. Framework web	49
4.14. Sistema gestor de bases de datos	49
4.15. Bibliotecas y librerías relevantes	49
4.16. Desarrollo web	52
4.17. Otras herramientas	53
5. Aspectos relevantes del desarrollo del proyecto	54
5.1. Inicio del proyecto	54
5.2. Metodologías	55
5.3. Formación	56
5.4. Obtención y procesamiento de datos	58
5.5. División por aplicaciones	62
5.6. Fuentes de noticias	63
5.7. Securizar claves de APIs	63
5.8. Desarrollo backend y frontend de <i>DashBoard</i> de usuario	64
5.9. Desarrollo backend y frontend del <i>Lab</i>	65
5.10. Desarrollo de formularios para interactuar con el usuario	69
5.11. Testing, log de tests y algunos estadísticos relevantes	70
6. Trabajos relacionados	72
7. Conclusiones y Líneas de trabajo futuras	74
7.1. Conclusiones	74
7.2. Líneas de trabajo futuras	75
Bibliografía	77

Índice de figuras

3.1. Distribuciones de retornos de diferentes valores. Fuente: elaboración propia	7
3.2. Simulación de 10.000 posibles portfolios (covarianza calculada con datos de mayo-2023 a mayo-2024). Fuente: elaboración propia	12
3.3. Frontera eficiente. Fuente: [57]	15
3.4. Simulación de Montecarlo con frontera eficiente y rentabilidad de cartera con valores <i>RED.MC</i> , <i>EOAN.DE</i> y <i>CSCO</i> , permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024) Fuente: Elaboración propia	16
3.5. Simulación de Montecarlo con frontera eficiente y Sharpe ratio con los valores <i>RED.MC</i> , <i>EOAN.DE</i> y <i>CSCO</i> , permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024) Fuente: Elaboración propia	18
3.6. Evolución de precios de cierre de <i>ITX.MC</i> en los últimos dos años y los mismos datos con diferenciación de orden 1. Fuente: elaboración propia	22
3.7. Serie no estacionaria por variación de media. Fuente: elaboración propia	23
3.8. Serie no estacionaria por variación de varianza. Fuente: elaboración propia	23
3.9. Serie no estacionaria, p-value >0.5. Fuente: elaboración propia .	25
3.10. Serie estacionaria, con p-value <0.5. Fuente: elaboración propia	26
3.11. Serie estacionaria, con p-value <0.5. Fuente: elaboración propia	27
3.12. ACF con diferenciación logarítmica de orden 1. Fuente: elaboración propia	29
3.13. PACF con diferenciación logarítmica de orden 1. Fuente: elaboración propia	30

3.14. Comparación de ARIMA con una estrategia naíf. Fuente: elaboración propia	32
3.15. Comparación entre ARIMA, con walk-forward anchored, y una estrategia naíf. Fuente: elaboración propia	32
3.16. Conjuntos de entrenamiento y test en estrategia basada en ML. Fuente: elaboración propia	38
4.1. Patrón MVT. Fuente: realización propia	45
5.1. Número de tests realizados. Fuente: elaboración propia	70

Índice de tablas

3.1. Cómo almacenar señales para algoritmo de seguimiento de tendencias	33
3.2. Cómo almacenar retornos para algoritmo de seguimiento de tendencias	35
3.3. Medias móviles simples frecuentes	35
3.4. Valores seleccionados de cada índice	37
3.5. Datos con estrategia basada en ML	38
3.6. Matriz de confusión	41
5.1. Estructura de aplicaciones	62
5.2. Cobertura de código	71
5.3. Número de líneas de código <i>Python</i>	71
5.4. Número de líneas de lenguajes de marcas y <i>JavaScript</i>	71

1. Introducción

Los mercados financieros juegan un papel fundamental en la economía global. Las empresas y Administraciones Públicas que necesitan financiarse acuden a estos mercados en busca de capital proveniente de ahorradores, que esperan obtener un rendimiento sobre el dinero aportado.

Los ahorradores prestan su dinero, depositando su confianza en una empresa, a través de la compra de acciones, bonos, pagarés y obligaciones - o productos derivados, así como materias primas -. Y para estos inversores, la toma de decisiones informada debe de ser la base principal de su estrategia de negocio.

Siguiendo las valiosas recomendaciones de Benjamin Graham [39] podremos invertir de forma sensata, realizando un análisis minucioso, basado en unos principios subyacentes que no se van a modificar sustancialmente con el paso del tiempo, pero que sí requieren de una constante actualización de la información sobre el entorno de las empresas y los mercados en los que cotizan. Por ello, en este trabajo se hace uso de la información disponible para ayudar a los inversores a formar una cartera bien diversificada, teniendo en cuenta diferentes divisas y mercados; y se aportan algunas herramientas de análisis poco frecuentes en otras plataformas web [9, 45, 56], como puede ser el análisis visual de correlaciones entre valores y la comparación gráfica con sectores de referencia, entre otros.

Por otro lado, según el Plan de Educación Financiera 2022-2025 [22] de la CNMV [23] y del Banco de España [20], existe un consenso generalizado sobre la necesidad de mejorar el nivel de cultura financiera, independientemente del país y las circunstancias de los ciudadanos, por lo que, de manera experimental, se introduce al usuario en la utilización de modelos para análisis de series temporales, con la intención de aportar herramientas

adicionales a su *backup* financiero. Concretamente, se da acceso al uso de modelos ARIMA [103] y de estrategias basadas en *machine learning*. Además, se ofrece una herramienta de control de carteras para que el usuario pueda hacer su propio análisis de rentabilidad-riesgo y se aportan soluciones que favorecen la diversificación.

1.1. Estructura de la memoria

La memoria sigue la siguiente estructura:

- **Introducción:** breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.
- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para gestión y desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Trabajos relacionados:** estado del arte en las aplicaciones y sitios web de bolsa y finanzas.
- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software:** se describe la fase de análisis; los objetivos generales, el catálogo de requisitos del sistema y la especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño:** se describe la fase de diseño; el ámbito del software, el diseño de datos, el diseño procedimental y el diseño arquitectónico.
- **Manual del programador:** recoge los aspectos más relevantes relacionados con el código fuente (estructura, compilación, instalación, ejecución, pruebas, etc.).

- **Manual de usuario:** guía de usuario para el correcto manejo de la aplicación.

1.2. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- **Herramienta web FAT:** Financial Analysis Tool[79].
- **Vídeos de demostración.**

Además, los siguientes recursos están accesibles a través de internet:

- **Repositorio** del proyecto [80].
- **Documentación del código** del proyecto [58].

2. Objetivos del proyecto

A continuación se detallan los objetivos que se persiguen con la realización de este proyecto:

2.1. Objetivos generales

- Desarrollar una aplicación *web* que permita a un usuario la composición de una cartera de valores cotizados bien diversificada.
- Ofrecer información agregada sobre la evolución de un valor y su sector de referencia.
- Permitir la comparación relativa entre valores cotizados.
- Aportar valor añadido a través del análisis de correlaciones entre valores.
- Facilitar la interpretación de los datos recogidos mediante representaciones gráficas.
- Dar acceso a información extra mediante el análisis de series temporales con modelos especializados.
- Introducir a los usuarios en la experimentación con técnicas de *trading* algorítmico.

2.2. Objetivos de carácter técnico

- Desarrollar una plataforma *web* con *Django* que permita realizar el seguimiento de valores cotizados en algunos de los principales índices de referencia mundiales.

- Crear bases de datos *SQLite* cuya actualización sea automática a través de un administrador de procesos como *cron* en un servidor *web* remoto o de forma semiautomática en un servidor local.
- Aplicar la arquitectura MVC (*Model-View-Controller*), más conocida en *Django* como MVT (*Model-View-Template*).
- Diseñar formularios que permitan la interacción con el usuario para realizar operaciones *CRUD* en la base de datos principal y operaciones de lectura en las bases de datos de los valores cotizados.
- Utilizar Git como sistema de control de versiones distribuido junto con la plataforma GitHub.
- Hacer uso de herramientas CI/CD integradas en el repositorio con *GitHub actions*. Por ejemplo, utilizar *pyllint* como herramienta de control de calidad de código, o *coverage* para testear de forma continuada el desarrollo del proyecto.
- Aplicar la metodología ágil Scrum junto con TDD (*Test Driven Development*) en los apartados que sea posible a lo largo del desarrollo del software.
- Cubrir todo el código con tests, para mejorar la calidad del producto final.
- Utilizar Zube como herramienta de gestión de proyectos.
- Utilizar un sistema de documentación como Sphinx con el estilo de Read The Docs y con la posibilidad de subir la documentación de forma continua.

2.3. Objetivos personales

- Crear una herramienta sencilla - y no habitual - para el público general en el ecosistema de las *webs* de los mercados de valores.
- Abarcar el máximo número posible de conocimientos adquiridos durante el grado.
- Explorar metodologías, herramientas y estándares utilizados en el mercado laboral.
- Introducirme en el mundo del análisis y el *forecasting* de series temporales de datos.

3. Conceptos teóricos

Los conceptos teóricos más destacables de este proyecto residen en el estudio del modelo de Markowitz para la formación de una cartera bien diversificada, el análisis de series temporales con modelos ARIMA y en técnicas básicas de trading algorítmico.

3.1. Diversificación de una cartera de valores cotizados

Esta sección puede empezar con la idea básica de que el riesgo en las inversiones es perjudicial y tener una cartera diversificada reduce el riesgo, por lo tanto, diversificar una cartera es una buena idea.

La diversificación de una cartera de valores cotizados es una estrategia fundamental para reducir el riesgo y mejorar la rentabilidad a largo plazo. Esta estrategia consiste en distribuir el capital entre diferentes activos, como acciones, bonos y activos de diferentes sectores y regiones geográficas. Al hacerlo, se reduce la dependencia del rendimiento de una sola empresa o sector, lo que protege a la cartera de las fluctuaciones del mercado y minimiza las pérdidas potenciales.

Una manera de caracterizar una cartera es a través del retorno medio de los activos que la componen y su varianza. En esta sección se verá cómo se realiza una optimización de varianza-media, o más conocida como *Modern Portfolio Theory (MPT)* [11]. Es decir, se va a demostrar cómo se busca una cartera con la mejor media y la mejor varianza posibles dados unos valores en dicha cartera y la ponderación de esos valores en la misma.

Disminuir la varianza para minorar el riesgo

El retorno logarítmico de un valor viene dado por:

$$R = \ln\left(\frac{P_t}{P_{t-1}}\right) \quad (3.1)$$

En este trabajo, en algunas ocasiones y por simplicidad de las explicaciones, se hablará de rentabilidad¹:

$$R_{simple} = \frac{P_{t-1} - P_t}{P_t} * 100 \quad (3.2)$$

Donde P_t es el último precio de cierre de mercado disponible y P_{t-1} es el precio de cierre previo.

Los retornos de un valor son aleatorios y podemos asumir, de forma general, que hay una distribución normal subyacente en ellos:

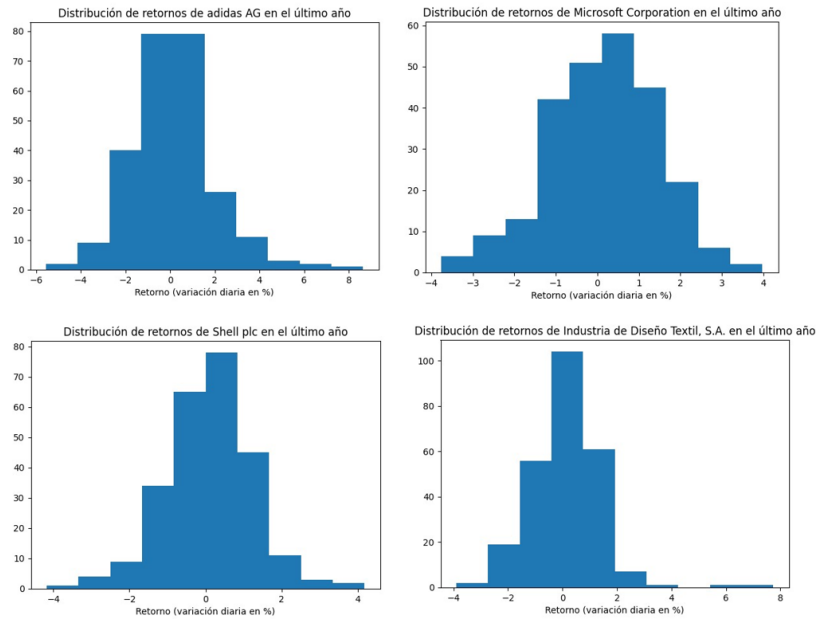


Figura 3.1: Distribuciones de retornos de diferentes valores. Fuente: elaboración propia

¹En los casos en los que sea conveniente el uso del retorno logarítmico, se indicará expresamente haciendo uso de [3.1](#)

Haciendo esta asunción de distribución normal en los retornos (algo que no siempre se cumple) podemos pensar en una cartera con dos valores, A y B, cuyas distribuciones de retornos son iguales:

$$A : R_1 \sim \mathcal{N}(\mu, \sigma^2) \quad (3.3)$$

$$B : R_2 \sim \mathcal{N}(\mu, \sigma^2) \quad (3.4)$$

Entonces, el retorno esperado será el mismo, μ , tengamos el 100 % de A en cartera, el 100 % de B o con diferentes ponderaciones. Sin embargo, la varianza sí es distinta, porque si calculamos la desviación estándar de los retornos de A y B tenemos lo siguiente:

$$sd(R_1) = sd(R_2) = \sigma \quad (3.5)$$

Es decir, si invertimos todo en A o todo en B, tendremos la misma varianza, pero si hacemos un reparto de, por ejemplo, 50 %/50 %, veremos que la varianza es menor.

Para ello, dadas las distribuciones de 3.3, asumiremos - por ahora - que son independientes. Además, supongamos que existe una variable Y con 1/2 de R_1 y 1/2 de R_2 :

$$Y = 1/2R_1 + 1/2R_2 \quad (3.6)$$

Entonces, lo que hay que calcular es la varianza de Y:

$$var(Y) = var(1/2R_1 + 1/2R_2) \quad (3.7)$$

Una de las maneras de calcularlo es teniendo en cuenta lo siguiente:

$$var(cX) = c^2 var(X) \quad (3.8)$$

$$var(A + B) = var(A) + var(B) \quad (3.9)$$

Y, por tanto:

$$var(1/2R_1 + 1/2R_2) = (1/2)^2\sigma^2 + (1/2)^2\sigma^2 = 1/2\sigma^2 \quad (3.10)$$

Es decir:

$$\text{var}(1/2R_1 + 1/2R_2) = 1/2\sigma^2 \rightarrow \text{sd} = 1/\sqrt{2}\sigma \quad (3.11)$$

Esto nos lleva a pensar que podemos obtener los mismos retornos medios pero disminuyendo la varianza - y la desviación estándar -, es decir, asumiendo menos riesgos en nuestras inversiones porque tendremos menor volatilidad.

Retorno esperado y varianza de una cartera (*portfolio*)

En este apartado se tratará de describir una cartera de valores de forma matemática, con el objetivo de buscar una manera de optimizarla. Para ello, empezaré con una serie de definiciones estadísticas.

Los valores de una cartera tienen unos pesos en la misma. A esos pesos se les puede caracterizar como un vector w :

$$w = \text{vector de longitud } D$$

Donde D es la cantidad de valores que tenemos en cartera². Así, la ponderación de un único valor en cartera vendrá representada por w_i , donde $i = 1, \dots, D$.

Los pesos tendrán algunas restricciones relevantes, como que la suma de todos ellos debe ser 1:

$$\sum_{i=1}^D w_i = 1 \quad (3.12)$$

Además, podríamos tener otras restricciones como que los pesos deben ser positivos, lo que limitaría el uso de posiciones cortas [105] en cartera. En estos casos, se puede limitar indicando la condición de que $w_i \geq 0$.

Por otro lado, hay que tener en cuenta algunas definiciones que se utilizan de forma habitual, como:

$$R_i = \text{retorno del valor } i$$

²En mi código, en lugar de D utilizo `num_valores` para hacerlo más intuitivo y para cumplir con las reglas de estilo de *Python*

El retorno medio es el valor esperado de R_i :

$$E(R_i) = \mu_i \text{ (forma vectorial de todos los } \mu_i : \mu \text{)} \quad (3.13)$$

Hay que considerar que es posible que exista una correlación entre los retornos de los valores y, por tanto, necesitaremos hacer uso de la matriz de covarianza:

$$E\{(R_i - \mu_i)(R_j - \mu_j)\} = \sum_{ij} \text{(forma matricial : } \sum_{DxD} \text{)} \quad (3.14)$$

Si tenemos en cuenta lo anterior, ya es posible definir dos conceptos fundamentales que son el retorno medio esperado y la varianza del retorno de una cartera (*portfolio*):

$$\mu_p = E(R_p) \quad (3.15)$$

$$\sigma_p^2 = \text{var}(R_p) \quad (3.16)$$

Para entender cómo se calculan, voy a empezar por el caso más sencillo, en el supuesto de tener sólo dos valores en cartera:

$$R_p = wR_1 + (1 - w)R_2 \quad (3.17)$$

R_p es una función de variables aleatorias y, por tanto, tendrá una distribución en términos de media y varianza. La media de R_p es su valor esperado y recordando que E es un operador lineal podemos operar. Además, podemos sustituir por [3.15](#):

$$E(R_p) = E(wR_1 + (1 - w)R_2) = wE(R_1) + (1 - w)E(R_2) \quad (3.18)$$

$$\mu_p = w\mu_1 + (1 - w)\mu_2 \quad (3.19)$$

La varianza de R_p requiere de más cálculos; no podemos hacer la suma directa de las dos varianzas porque puede existir correlación entre R_1 y R_2 . Entonces, lo más sencillo es sustituir en la definición de varianza por [3.17](#) y [3.19](#):

$$\begin{aligned}
\text{var}(R_p) &= E\{(R_p - \mu_p)^2\} \\
&= E\{[wR_1 + (1-w)R_2 - w\mu_1 - (1-w)\mu_2]^2\} \\
&= E\{[w(R_1 - \mu_1) + (1-w)(R_2 - \mu_2)]^2\} \\
&= E\{w^2(R_1 - \mu_1)^2\} + E\{(1-w)^2(R_2 - \mu_2)^2\} \\
&\quad + 2E\{w(1-w)(R_1 - \mu_1)(R_2 - \mu_2)\} \\
&= w^2\text{var}(R_1) + (1-w)^2\text{var}(R_2) + 2w(1-w)\text{cov}(R_1, R_2)
\end{aligned} \tag{3.20}$$

Lo visto es 3.20 se puede escribir en términos de la correlación en lugar de la covarianza, $\text{corr}_{12} = \rho_{12} = \sigma_{12}/(\sigma_1\sigma_2)$ [91]:

$$\sigma_p^2 = w_2\sigma_1^2 + (1-w)^2\sigma_2^2 + 2w(1-w)\sigma_{12} \tag{3.21}$$

$$\sigma_p^2 = w_2\sigma_1^2 + (1-w)^2\sigma_2^2 + 2w(1-w)\rho_{12}\sigma_1\sigma_2 \tag{3.22}$$

Cualquiera de estas dos fórmulas puede ser utilizada para calcular la varianza de una cartera de valores.

Correlación entre valores

Inicialmente asumía que los retornos de los valores eran totalmente independientes para demostrar que la diversificación disminuye la varianza y, por tanto, el riesgo. Sin embargo, aquí vemos que los retornos no tienen por qué ser independientes.

Entonces, analizando detenidamente 3.21 y 3.22 vemos que, si $0 < w < 1$ y R_1 y R_2 están positivamente correlados, la varianza de la cartera aumenta. Y si R_1 y R_2 están negativamente correlados disminuimos la varianza del portfolio y, por tanto, tendremos menor riesgo.

Ahora, para poder adecuar a código de *Python* estas fórmulas, voy a pasarlas a la notación de producto escalar y despejar la matriz de covarianza de R , $w^T \Sigma w$:

$$\begin{aligned}
\text{var}(R_p) &= E\{(R_p - \mu_p^2)\} \\
&= E\{(R^T w - \mu^T w)^2\} \\
&= E\{(R^T w - \mu^T w)^T (R^T w - \mu^T w)\} \\
&= E\{w^T (R^T - \mu^T)^T (R^T - \mu^T) w\} \\
&= w^T E\{(R - \mu)(R - \mu)^T\} w \\
&= w^T \Sigma w
\end{aligned} \tag{3.23}$$

Simulación de Montecarlo

Es habitual representar las carteras de valores en términos de la relación rentabilidad/riesgo. Para mantener una idea matemática de los conceptos de este trabajo, al riesgo lo voy a denominar volatilidad:

$$\text{volatilidad cartera} = \text{volatilidad}_p = \sqrt{\text{var}(R_p)} \tag{3.24}$$

Una vez tenemos las fórmulas definidas se puede realizar una simulación de Montecarlo [55] con múltiples posibles carteras de inversión y ver la relación rentabilidad/riesgo (retorno esperado/varianza):

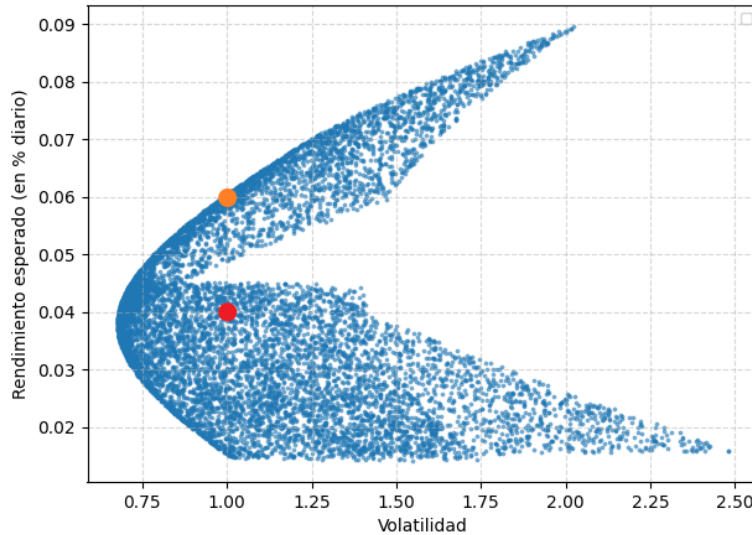


Figura 3.2: Simulación de 10.000 posibles portfolios (covarianza calculada con datos de mayo-2023 a mayo-2024). Fuente: elaboración propia

De esta gráfica podemos deducir que es posible obtener una mejor rentabilidad sin aumentar el riesgo, i.e., puede mejorarse el rendimiento esperado de nuestra cartera manteniendo la misma volatilidad. Por ejemplo, el punto naranja indica una cartera eficiente, para la que dada una volatilidad, obtenemos el máximo rendimiento esperado posible. Mientras que la cartera representada con un punto rojo obtienen un rendimiento menor para la misma volatilidad.

Además, podemos intuir que según aumentamos el riesgo que estamos dispuestos a asumir podemos esperar mayores retornos.

Retornos máximo y mínimo posibles

Para calcular el retorno de una cartera podemos hacer también la representación de producto escalar:

$$\mu_p = \mu^T w \quad (3.25)$$

Si utilizo la intuición rápida de maximizar el retorno por sí sólo, estaré cometiendo un error en los cálculos, ya que como es de esperar el retorno no puede crecer indefinidamente (el máximo de la ecuación previa es ∞). Entonces, hay que añadir al menos dos restricciones, que son la de que los pesos de los valores en cartera deben sumar 1 y que los pesos deben ser positivos. Por tanto, una representación más adecuada sería:

$$\begin{aligned} & \max_w \mu^T w \\ & \text{suje}to \ a : 1_D^T w = 1 \\ & \quad w_i \geq 0 \end{aligned} \quad (3.26)$$

Es decir, estamos ante un problema de optimización con restricciones³. Y, más concretamente, se trata de un problema de programación lineal (LP)[83].

De manera similar se puede calcular el retorno mínimo:

$$\begin{aligned} & \min_w \mu^T w \\ & \text{suje}to \ a : 1_D^T w = 1 \\ & \quad w_i \geq 0 \end{aligned} \quad (3.27)$$

³Es habitual añadir otras restricciones como que, por ejemplo, ningún valor tenga un peso mayor al 50 %, i.e., $w_i \leq 0,5$ pero en mi caso no utilizaré esta limitación.

En *Python* estas funciones pueden representarse a través de la librería *Scipy*, concretamente, con `scipy.optimize.linprog`⁴

Optimización en términos de retorno y riesgo simultáneamente

En el apartado anterior se ha visto cómo optimizar los retornos, pero sin tener en cuenta el riesgo (o volatilidad). En este apartado se añade el concepto de riesgo para realizar una optimización simultánea.

Ya hemos visto, de forma intuitiva, que según aumenta el retorno esperado también aumenta el riesgo. La idea de la optimización de carteras es que no tomemos más riesgos de los necesarios.

El riesgo lo podemos medir con la desviación estándar. Como la minimización de la varianza también implica la minimización de la desviación estándar (la raíz cuadrada es una función monótona creciente), usaré la varianza calculada en 3.23 por comodidad en los cálculos.

Si suponemos que queremos un determinado retorno r , podríamos representar la minimización del riesgo de la siguiente manera:

$$\begin{aligned} \min_w w^T \Sigma w \\ \text{sujeto a : } \mu^T w &= r \\ 1_D^T w &= 1 \\ w_i &\geq 0 \end{aligned} \tag{3.28}$$

Como vemos, estamos ante un problema de programación cuadrática (QP), porque la función objetivo es cuadrática en lugar de lineal, aunque las restricciones sí siguen siendo lineales.

Para simular la optimización de una función cuadrática en *Scipy* hay que utilizar una función genérica llamada `minimize()`⁵. Hay otras librerías más específicas, pero podemos adecuar *Scipy* para problemas QP.

⁴En este proyecto se puede ver cómo se aplica `scipy.optimize.linprog` en `DashBoard.views.py`, en el método `_rendimientos_min_y_max(retornos_df)`.

⁵En este proyecto se puede ver cómo se aplica `minimize()` en `DashBoard.views._frontera_eficiente_por_optimizacion()` como un problema de QP y en `DashBoard.views._mejores_pesos_por_optmizacion()` como un problema que no es QP ni LP

Frontera eficiente

Una vez hemos conocidos los retornos mínimo y máximo posibles, con la función que queremos optimizar, 3.28, podemos hacer que el retorno, r , sea una sucesión de puntos entre el mínimo y el máximo e ir calculando la varianza mínima con esos retornos objetivos. Esto nos dará el mejor nivel de riesgo posible para cada retorno entre el mínimo y el máximo posibles.

Esto nos dará una representación en forma de hipérbola que se conoce como frontera eficiente [95] y que Harry Markowitz representó en [57] con su forma parabólica de la siguiente manera:

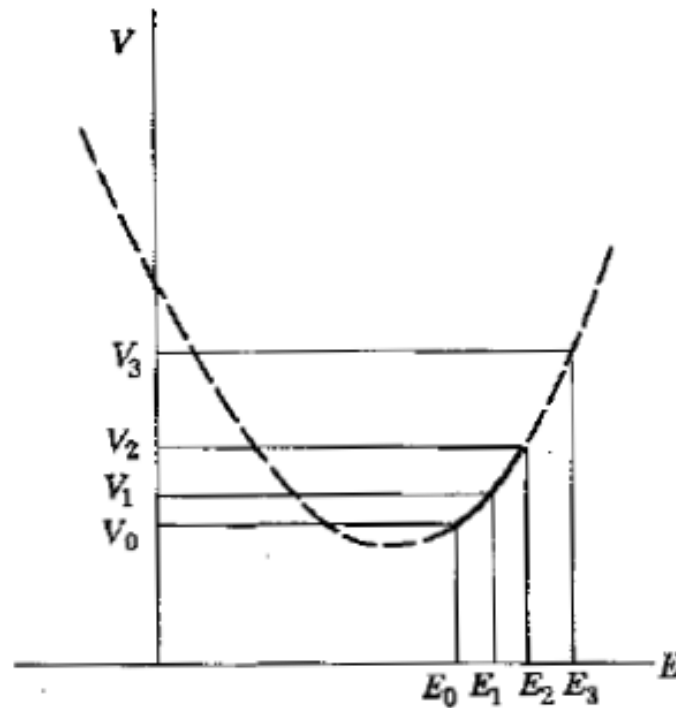


Figure 16. V and E along a critical line.

Figura 3.3: Frontera eficiente. Fuente: [57]

Si aplicamos estos conceptos a la simulación de Montecarlo que se realizaba previamente, se obtiene lo siguiente:

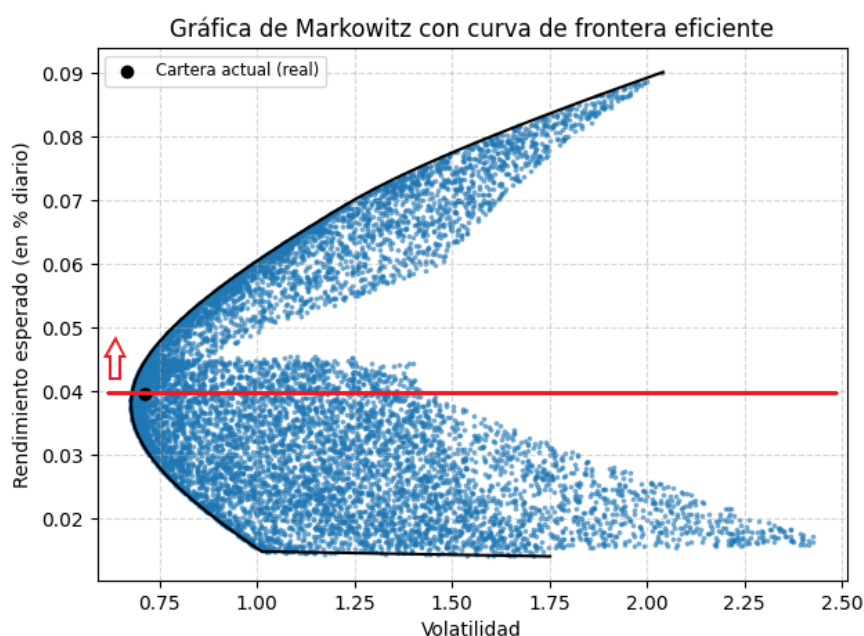


Figura 3.4: Simulación de Montecarlo con frontera eficiente y rentabilidad de cartera con valores *RED.MC*, *EOAN.DE* y *CSCO*, permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024) Fuente: Elaboración propia

La parte superior ⁶ de la línea curva negra que rodea la nube de posibles carteras, obtenidas con la simulación de Montecarlo, es lo que se conoce como frontera eficiente. Lo interesante de esta curva es que cualquier punto que seleccionemos de ella nos indica que no hay otra posible cartera con menor riesgo para el mismo retorno - o que no podemos encontrar un retorno esperado mejor para un determinado nivel de riesgo -.

Sharpe ratio

Hasta ahora se ha visto que podemos encontrar diferentes carteras localmente óptimas - distintas distribuciones de pesos de los mismos valores cotizados - a lo largo de la frontera eficiente, pero cabe preguntarse cómo podemos comparar dos carteras, i.e., cuál es mejor si las dos están en la frontera eficiente.

⁶Aunque se obtiene toda la curva, por el propio proceso de optimización, la parte inferior no se puede considerar eficiente porque sólo tenemos que proyectar hacia la parte superior de la misma para ver retornos mejores.

En principio, podemos asumir que el perfil del inversor influirá en una mayor o menor aversión al riesgo, pero lo ideal es hacer un ratio entre el retorno esperado y la volatilidad para tener una medida objetiva. Ese ratio se conoce como *Sharpe ratio* [108]:

$$SR = \frac{E(R_p) - r_f}{\sigma_p} \quad (3.29)$$

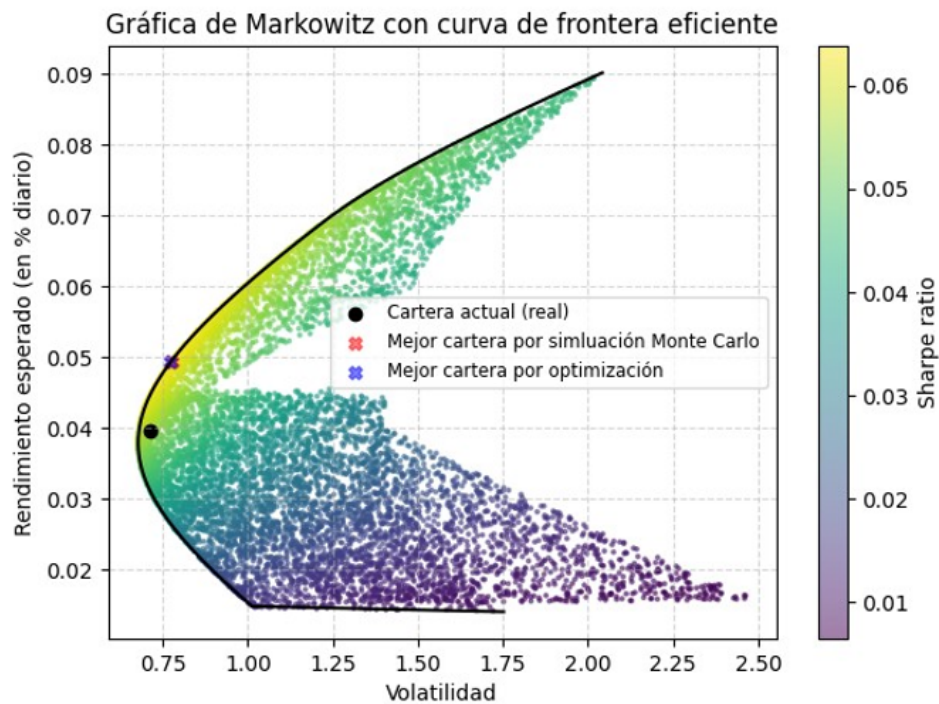
Donde r_f representa la tasa libre de riesgo⁷, que es un retorno garantizado que pueden tener determinados activos como depósitos, bonos, letras o similares.

La obtención del *Sharpe ratio* se puede realizar de dos formas diferentes. Por un lado, podemos optimizar la función del *Sharpe ratio*⁸ pero también podemos aprovechar las múltiples carteras de la simulación de Montecarlo y buscar la de mejor ratio entre todas ellas. En cualquier caso, si se han simulado suficientes carteras, los resultados deben de ser similares - no iguales porque en la simulación puede que no se haya creado la cartera óptima global -.

Para facilitar la comprensión al usuario se puede realizar una gráfica con toda la información necesaria y acompañarlo de información adicional sobre la distribución de pesos de cada caso:

⁷En este proyecto se considera una tasa libre de riesgo de 0, porque sólo se utilizan acciones y, aunque tienen rentabilidades por dividendos, éstos no están garantizados. Otra perspectiva podría ser tomar las rentabilidades de los bonos del estado como referencia para la tasa libre de riesgo, pero he preferido limitar los cálculos al mercado de acciones cotizadas.

⁸En este trabajo se puede ver cómo se optimiza en `DashBoard.views._mejores_pesos_por_optimizacion()`



	Distribución de pesos actual (%)	Mejor distribución según simulación de Monte Carlo (%)	Mejor distribución según optimización de sharpe ratio (%)
RED.MC	19.83	6.46	6.76
EOAN.DE	31.82	63.97	64.37
CSCO	48.35	29.57	28.87
Sharpe ratio	0.05569	0.06383	0.06383

Figura 3.5: Simulación de Montecarlo con frontera eficiente y Sharpe ratio con los valores *RED.MC*, *EOAN.DE* y *CSCO*, permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024) Fuente: Elaboración propia

3.2. Análisis de series temporales con modelos ARIMA

Cuando hablamos de *forecasting* (previsión o predicción) de series temporales con productos financieros hay que tener en cuenta que estamos ante

un proceso especialmente complicado y que los resultados obtenidos, en muchas ocasiones, son erróneos o decepcionantes. Por ello, en este trabajo se propone un laboratorio virtual para analizar los datos, que le permita al usuario experimentar con algunas herramientas relativas al análisis de series temporales y que sea esa persona la que saque sus propias conclusiones a través de la información obtenida.

Ya adelanto que "no existe una bola de cristal que nos diga cuál es el precio futuro de un activo". De hecho, al final de esta sección se intenta demostrar que un modelo *naïve forecast* obtiene mejores resultados que un modelo ARIMA cuando hablamos de datos de valores cotizados.

Modelos autorregresivos ($AR_{(p)}$)

En estadística, un modelo autorregresivo (AR) es una representación de un proceso aleatorio en el que la variable de interés depende de sus observaciones pasadas. De forma general podemos decir que estos modelos son, básicamente, modelos de regresión lineal donde los predictores son datos pasados en la serie temporal. En su versión más sencilla lo podemos representar por:

$$\hat{y} = mx + b \quad (3.30)$$

Donde \hat{y} es el estimador, x son los datos de entrada y m y b son parámetros que se encuentran a través de la minimización del error de las predicciones - error entendido como diferencia entre real y estimado -.

Si tenemos más de una entrada de datos podemos representarlo de la siguiente manera:

$$\hat{y} = w_1x_1 + w_2x_2 + b \quad (3.31)$$

Donde, de nuevo, w_1 , w_2 y b se encuentran a través de un proceso de minimización del error.

Un modelo autorregresivo es un modelo de regresión lineal múltiple donde las entradas son los valores ya pasados en la serie temporal. Un modelo autorregresivo de orden p (AR_p) se puede representar de la siguiente manera:

$$\hat{y}_t = b + \phi_1y_{t-1} + \phi_2y_{t-2} + \dots + \phi_py_{t-p} \quad (3.32)$$

Y se puede denotar para un momento temporal concreto como:

$$y_t = b + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t \quad (3.33)$$

Donde, ϵ_t es el *ruido* en ese instante t :

$$\epsilon_t = \mathcal{N}(0, \sigma^2) \quad (3.34)$$

Es decir, se mide y_t como un modelo lineal de las entradas más un *ruido*. Esto nos lleva a pensar - asumiendo que el valor esperado del *ruido* es 0 - que:

$$\hat{y}_t = E(y_t) \quad (3.35)$$

Modelos de media móvil ($MA_{(q)}$)

Es importante empezar destacando que no hay que confundir el nombre de estos modelos con la *media móvil simple* ni la *media móvil exponencialmente ponderada* (EWMA, o MMEP en castellano).

Este modelo es similar al autorregresivo en el sentido de que es una función lineal, pero en este caso es una función en términos de errores pasados. Es decir, depende de los errores previos, no de los datos anteriores de la serie temporal. Se representa por:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q} \quad (3.36)$$

Si seguimos tratando los errores como una distribución normal, $\epsilon_t = \mathcal{N}(0, \sigma^2)$, cuando calculamos el valor esperado veremos que todos esos errores son 0 y, por tanto, el valor esperado sólo dependerá de c :

$$\begin{aligned} E(y_t) &= E(c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}) \\ &= c \end{aligned} \quad (3.37)$$

Entonces, podemos entender el término sesgado c como el valor medio, y los errores como fluctuaciones que hacen que y_t vaya arriba o abajo alrededor de c .

Combinación de modelos: ($ARMA_{(p,q)}$)

Usaremos este modelo cuando creamos que nuestros datos están linealmente correlados tanto con datos pasados en la serie temporal como con errores pasados del modelo.

$$y_t = b + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q} + \epsilon_t \quad (3.38)$$

Cuando entrenamos un modelo ARMA nos interesa que los datos estén cercanos a la estacionariedad [106]. La estacionariedad es muy útil en procesos de modelado, porque implica que varios estadísticos, como la media, la varianza, la autocorrelación, etc. se mantendrán constantes con el tiempo. Y en un modelo ARMA estamos tomando ventanas temporales previas para entrenar el modelo, lo cual nos dará buenos resultados si los estadísticos se mantienen a lo largo de otras ventanas temporales.

Datos estacionarios. $I_{(d)}$ y $ARIMA_{(p,d,q)}$

Los datos de los valores cotizados, normalmente, no son estacionarios. Por ello, es necesario realizar un proceso de diferenciación para intentar aproximarlos a la estacionariedad. Aquí es donde entra en juego la I de $ARIMA$.

Un proceso $I_{(d)}$ es un proceso estacionario después de haber diferenciado d veces. Es decir, $I_{(d)}$ es un proceso integrado en orden d . Normalmente necesitaremos diferenciar una o dos veces, pero no es aconsejable hacerlo más de dos veces.

De hecho, es posible ver que con una sola diferenciación podemos eliminar la tendencia de los datos:

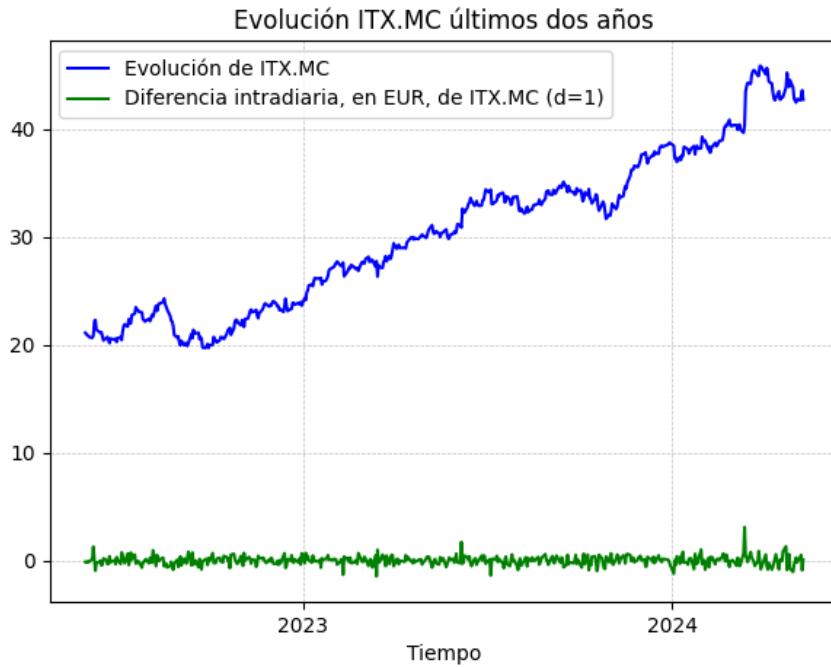


Figura 3.6: Evolución de precios de cierre de *ITX.MC* en los últimos dos años y los mismos datos con diferenciación de orden 1. Fuente: elaboración propia

Pero esto sólo nos da una idea intuitiva de cómo son los datos y, para poder hacer un buen análisis de los mismos, necesitaremos demostrar de alguna forma si nuestros datos son realmente estacionarios.

En la sección previa se daba una idea informal de qué son datos no estacionarios, haciendo referencia a la tendencia y a la variación de determinados estadísticos a lo largo de una serie temporal pero, para tener una idea más clara, se pueden ver algunos ejemplos de datos no estacionarios:



Figura 3.7: Serie no estacionaria por variación de media. Fuente: elaboración propia

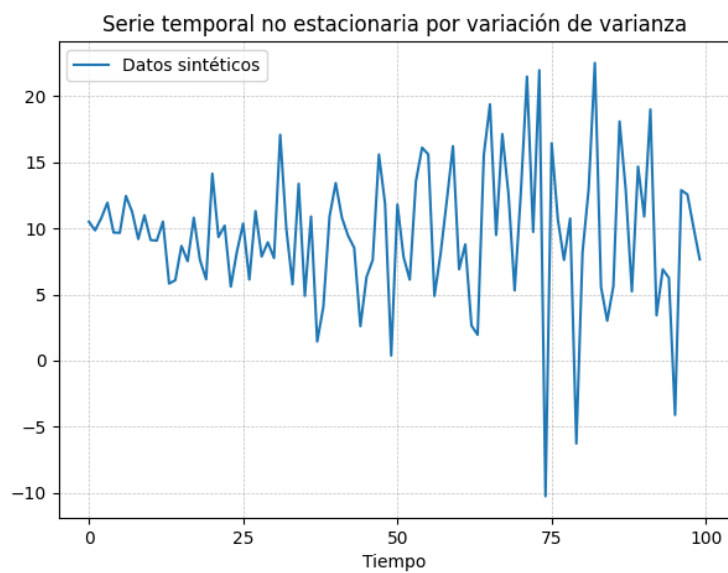


Figura 3.8: Serie no estacionaria por variación de varianza. Fuente: elaboración propia

De nuevo, esto sólo nos da una idea intuitiva. Pero existe un método para testear si una serie temporal es estacionaria o no: la prueba de Dickey-Fuller aumentada (test ADF, por sus siglas en inglés) [107].

En esta prueba tenemos una hipótesis nula - la serie temporal es no estacionaria - y una hipótesis alternativa - la serie temporal es estacionaria -. Introduciremos la serie temporal de datos y obtendremos un resultado con un p -value. Entonces, podré comprobar si el p -value está por encima o por debajo de un umbral significativo, habitualmente 1 % ó 5 %. Si está por debajo del umbral establecido puedo rechazar la hipótesis nula.

Esto en ARIMA tiene bastante utilidad, porque puedo diferenciar d veces los datos hasta conseguir una serie estacionaria, comprobando con un test ADF. Una vez los datos son estacionarios puedo aplicar un modelo $ARMA_{(p,q)}$.

De manera formal se puede hablar de estacionariedad fuerte o débil (SSS o WSS respectivamente, por sus siglas en inglés)[110]. Una estacionariedad fuerte significa que la distribución de las variables aleatorias de un proceso estocástico no varía con el tiempo, es decir, no varía en diferentes momentos τ de la serie temporal:

$$F_Y(y_{t_1+\tau}, y_{t_2+\tau}, \dots, y_{t_n+\tau}) = F_Y(y_{t_1}, y_{t_2}, \dots, y_{t_n}) \quad \forall \tau, t_1, t_2, \dots, t_n \quad (3.39)$$

Una estacionariedad débil hace uso de estadísticos de primer y segundo orden, la media y la varianza (o la auto-covarianza). Por tanto, se va a parecer más a la definición informal que había dado previamente. Se puede representa por:

$$\mu_Y(t) = \mu_Y(t + \tau) \quad \forall \tau \quad (3.40)$$

$$K_{YY}(t_1, t_2) = K_{YY}(t_1 - t_2, 0) \quad \forall t_1, t_2 \quad (3.41)$$

Lo que significa que no importa en qué momento τ de la serie temporal miremos, porque tendremos la misma media y que la auto-covarianza no varía con el tiempo.

Entonces, $ARIMA_{(p,d,q)}$ es un modelo en el que hemos diferenciado d veces antes de aplicar un modelo $ARMA_{(p,q)}$. Y hemos visto que $I_{(d)}$ es una operación que hacemos sobre los datos, mientras que en las partes del modelo autorregresivo y de la media móvil tenemos unas fórmulas que nos permiten hacer predicciones.

***p-value* de un test ADF sobre precios de cierre y sobre retornos**

Dada la importancia que tiene la estacionariedad de los datos, voy a mostrar cómo un test ADF nos arroja valores muy diferentes de *p-value* dependiendo del tipo de datos que estemos utilizando. En todos los ejemplos voy a utilizar un umbral del 5% para el *p-value* a través de una función que me diga si los datos son estacionarios, o no, en base a ese umbral, algo similar a:

```
from statsmodels.tsa.stattools import adfuller
def comprobar_serie_temporal(x):
    res = adfuller(x)
    p_value = res[1]
    if res[1] < 0.05:
        return p_value, "Serie temporal estacionaria"
    return p_value, "Serie temporal no estacionaria"
```

Para esta demostración se usan los datos de precios de cierre de los dos últimos años (mayo 2023 a mayo 2024) de la empresa Inditex. En esos datos se puede ver una clara tendencia ascendente y, por tanto, no será una serie estacionaria.

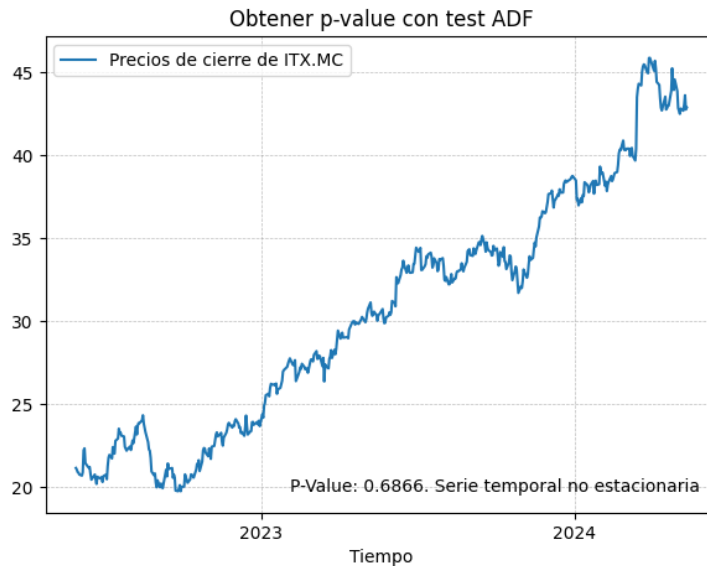


Figura 3.9: Serie no estacionaria, $p\text{-value} > 0.5$. Fuente: elaboración propia

Además, voy a aplicar un test ADF sobre los retornos - variaciones diarias en % - de Inditex en el mismo período de tiempo, para ver que el resultado mejora. Y, por último, voy a utilizar `diff()` para analizar la evolución intradía no porcentual ⁹, que es el tipo de diferenciación que hacen funciones como `auto.arima` de R o `pmdarima.arima.auto_arima` para Python.

De las figuras siguientes podemos deducir que es posible obtener series temporales no estacionarias a partir de precios de cierre de un valor cotizado. Esto permitirá trabajar con esos datos en un modelo ARIMA.

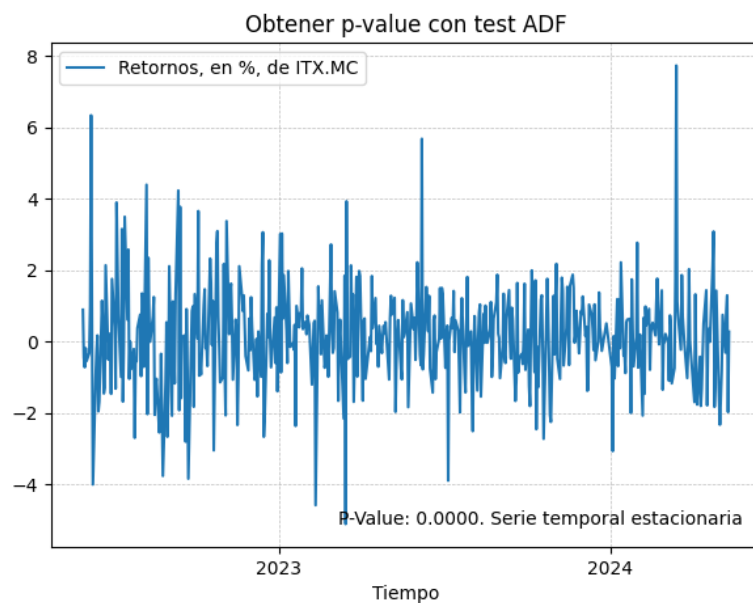


Figura 3.10: Serie estacionaria, con p-value < 0.5. Fuente: elaboración propia

⁹Es habitual aplicar una función logarítmica a `diff()` para suavizar los resultados, pero a efectos gráficos, en este caso, no aporta mejoras significativas.

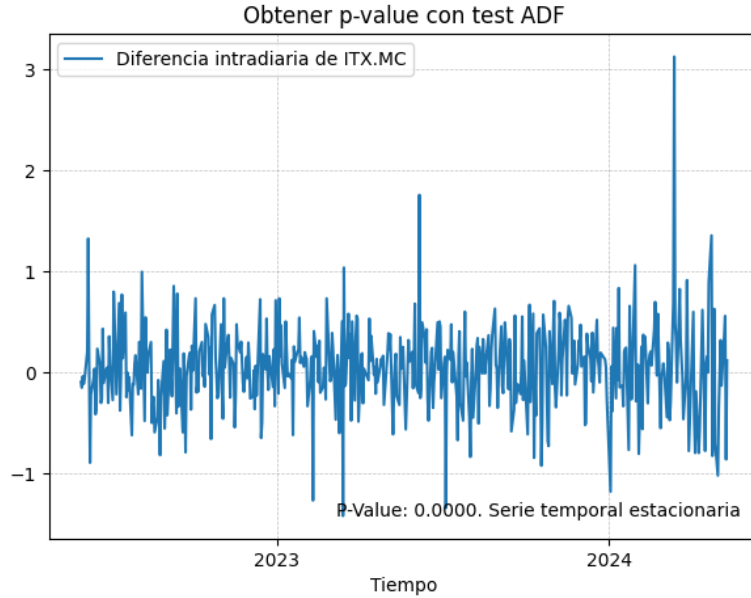


Figura 3.11: Serie estacionaria, con p-value < 0.5. Fuente: elaboración propia

Selección de hiperparámetros q y p . ACF y PACF

Aunque existen funciones automáticas que obtienen los mejores hiperparámetros (p , d , q) de ARIMA, conviene saber de dónde se sacan éstos. Ya hemos visto cómo deducir d a través de los resultados de un test ADF. Ahora voy a mostrar cómo buscar los mejores valores de q y p a través de funciones que pueden ser útiles para el análisis de los datos.

Función de autocorrelación (ACF, por sus siglas en inglés) para obtener q

Sabemos que la covarianza se define como el valor esperado entre dos variables aleatorias cualquiera:

$$\text{cov}(X, Y) = E\{(X - \mu_X)(Y - \mu_Y)\} \quad (3.42)$$

Y la autocovarianza es la covarianza entre dos variables aleatorias específicas seleccionadas de dos puntos diferentes en una misma serie temporal.

$$\text{autocovarianza} = \text{cov}(Y_{t_1}, Y_{t_2}) \quad (3.43)$$

Entonces, en una relación similar a la que hay entre la covarianza y la correlación, podemos establecer la autocorrelación como:

$$\text{autocorrelación} = \frac{\text{cov}(Y_{t_1}, Y_{t_2})}{\sigma_Y(t_1)\sigma_Y(t_2)} \quad (3.44)$$

La ACF está definida como si los datos de la serie temporal fueran estacionarios, i.e., que la autocorrelación depende sólo de una diferencia temporal. Esa diferencia entre dos momentos de la serie temporal se puede denotar por $\rho(\tau)$:

$$\rho(Y_{t_1}, Y_{t_2}) = \rho(t_1, t_2) = \rho(\tau) \quad (3.45)$$

El porqué del estimador de $\hat{\rho}(\tau)$ escapa a los objetivos de explicación de este proyecto, pero si asumimos que la autocorrelación se mantiene constante en el tiempo, podemos usar todas las muestras de la serie para calcular la autocorrelación para cualquier τ y la fórmula para calcularlo es:

$$\hat{\rho}(\tau) = \frac{1}{(T - \tau)\hat{\sigma}^2} \sum_{t=1}^{T-\tau} (y_t - \hat{\mu})(y_{t+\tau} - \hat{\mu}) \quad (3.46)$$

La ventaja de establecerlo como una fórmula es que nos permite mostrar una gráfica de la autocorrelación con $\hat{\rho}$ en el eje vertical y τ en el horizontal. Pero la parte interesante está en que también se muestran los intervalos de confianza, que se pueden entender como un umbral. Cuando veamos un τ que supere el umbral de los intervalos de confianza podremos decir que $\hat{\rho}$ no se iguala a 0 en ese punto. Aunque habrá que tener en cuenta que si tenemos un α de, por ejemplo, el 5 %, significa que habrá un 5 % de los datos fuera de los intervalos de manera aleatoria - pero suelen ser datos cercanos al umbral límite -.

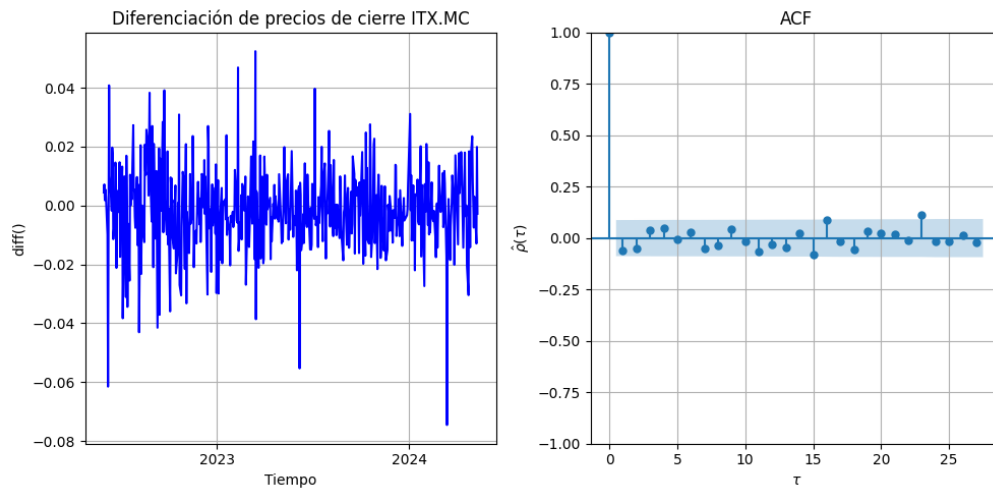


Figura 3.12: ACF con diferenciación logarítmica de orden 1. Fuente: elaboración propia

En estas gráficas, normalmente se verá que cuando τ es 0, la autocorrelación es 1. Esto es porque cuando τ es 0, hacemos $\frac{\text{varianza}}{\text{varianza}}$ y será 1.

Si miramos una gráfica de autocorrelación y vemos que en τ , la ACF $\hat{\rho}$ está fuera del intervalo de confianza, pero justo después de ese τ resulta que $\hat{\rho}$ no está fuera del intervalo, entonces interpretamos que el primer τ es la q adecuada para el modelo $MA_{(q)}$.

Función de autocorrelación parcial (PACF, por sus siglas en inglés) para obtener p

En esta función, de manera similar a como se hacía en ACF, se utiliza una representación gráfica de la propia función en el eje vertical y una especie de retraso temporal (como antes τ) en el eje horizontal. También se muestran los intervalos de confianza y su uso es igual que para los modelos $MA_{(q)}$.

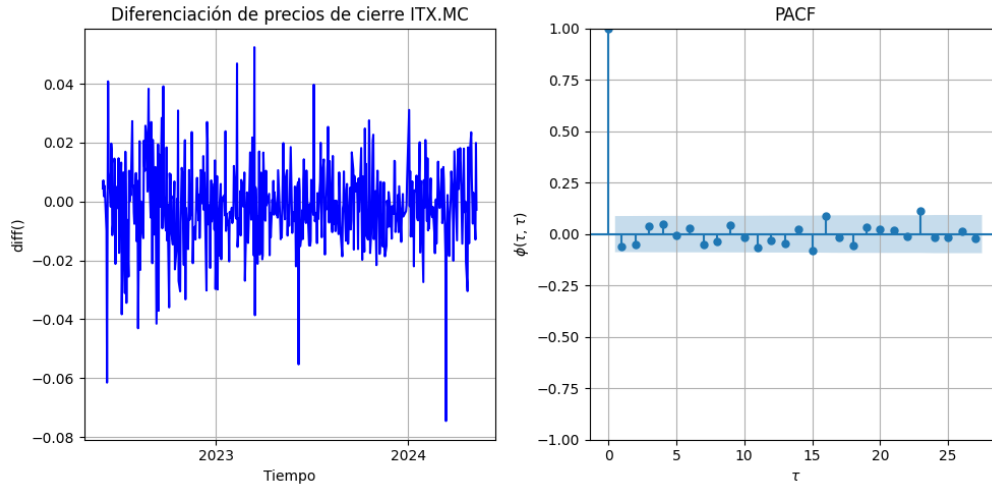


Figura 3.13: PACF con diferenciación logarítmica de orden 1. Fuente: elaboración propia

La definición de la fórmula también está fuera de los objetivos de este trabajo, pero de manera general se podría decir que la PACF en τ es la autocorrelación entre $Y(t)$ y $Y(t + \tau)$ pero condicionada a $Y(t + 1), Y(t + 2), \dots, Y(t + \tau - 1)$.

Se puede representar por:

$$\begin{aligned} \phi(\tau, \tau) &= PACF \text{ en } \tau \\ \phi(\tau, \tau) &= corr(Y_{t+\tau} - \hat{Y}_{t+\tau}, Y_t - \hat{Y}_t) \end{aligned} \quad (3.47)$$

Por ejemplo, $\phi(0, 0) = 1$ porque no hay valores entre $Y(t)$ e $Y(t)$.

Casos especiales de ARIMA

Algunos casos especiales de ARIMA son:

$$\begin{aligned} ARIMA(p, 0, 0) &= ARMA(p, 0) = AR(p) \\ ARIMA(0, 0, q) &= ARMA(0, q) = MA(q) \\ ARIMA(0, d, 0) &= I(d) \end{aligned}$$

Otro caso especial interesante es $ARIMA(0, 1, 0) = I(1)$ que no es más que un paseo aleatorio (RW o random walk)[89] en el que lo único que

tenemos es un modelo representado por *ruido*. Este modelo tan particular se puede denotar por:

$$\begin{aligned}\Delta y_t &= \epsilon_t \\ y_t - y_{t-1} &= \epsilon_t \\ y_t &= y_{t-1} + \epsilon_t\end{aligned}\tag{3.48}$$

Por ejemplo, suponiendo que tenemos un conjunto de datos a modelar formado por retornos logarítmicos de un valor cotizado:

$$r_t = p_t - p_{t-1} = \epsilon_t \sim \mathcal{N}(\mu, \sigma^2)\tag{3.49}$$

Si entrenamos un modelo ARIMA con dichos datos y los mejores parámetros son $p = 0, d = 1, q = 0$, significa que los retornos no son predecibles a través de los valores ni errores previos en la serie temporal. Esto es interesante porque podemos obtener información relevante aunque no estemos haciendo una predicción de la evolución de los precios - o de los retornos -.

Comprobar resultados de ARIMA

Una vez obtenidos los parámetros deseados de ARIMA y tras realizar un análisis con el modelo, tendremos que comparar nuestros resultados con algún método base de referencia. En este trabajo se propone utilizar como modelo base una predicción naïf, que consistirá en estimar que el precio de cierre de la siguiente sesión será igual al de la sesión actual.

A priori, se sabe que esto ocurrirá en muy pocas ocasiones, pero como línea de base comparativa puede ser útil, ya que en la mayoría de los casos los usuarios podrán comprobar - por comparación de RMSE - que los resultados de la estrategia naïf son mejores que los de ARIMA.

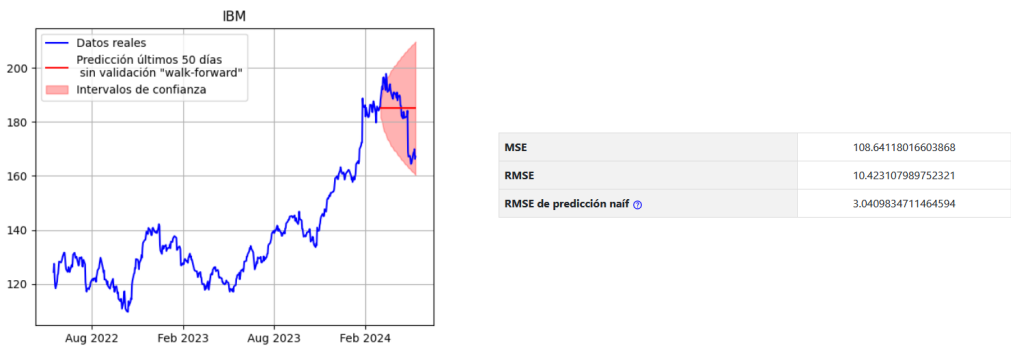


Figura 3.14: Comparación de ARIMA con una estrategia naïf. Fuente: elaboración propia

Otra posible estrategia es utilizar una validación *walk-forward anchored*[14] para evaluar el MSE y, posteriormente, compararlo de nuevo con una predicción naïf. En este caso, los usuarios podrán ver que los resultados de ARIMA son similares a los de la estrategia naïf y, en muchos casos, peores.

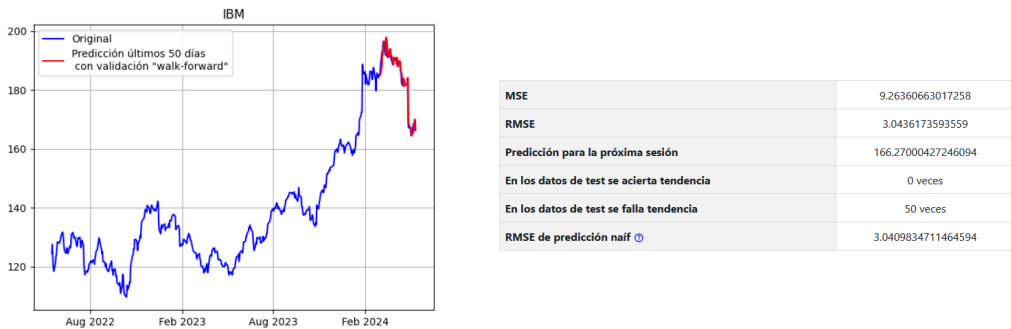


Figura 3.15: Comparación entre ARIMA, con walk-forward anchored, y una estrategia naïf. Fuente: elaboración propia

De esto se puede deducir que ARIMA sirve para entender los datos pero, en la gran mayoría de los casos, no funciona bien para predecir precios de valores cotizados¹⁰.

¹⁰Estas comprobaciones se pueden hacer con el uso de la web generada en este trabajo, en cualquiera de los apartados de ARIMA(p,d,q) el usuario puede experimentar y sacar sus propias conclusiones a través de resultados similares a los mostrados en esta sección

3.3. Trading algorítmico

Estrategia de cruce de medias (o seguimiento de tendencias)

La idea básica de esta estrategia es crear dos medias móviles simples sobre los precios de cierre de valores cotizados. Una será la media móvil *lenta* y la otra será la media móvil *rápida*. De forma intuitiva podemos deducir que si incluimos más datos en una media móvil, ésta se ajustará de manera más lenta, porque cada dato en la media móvil tendrá menor influencia en el resultado. Es decir, una media móvil de 50 días será más lenta que una media móvil de 20 días.

La clave de este algoritmo reside en los puntos en los que la media móvil *rápida* cruza la media móvil *lenta*. Si la media móvil *rápida* cruza desde abajo, entonces consideramos que es un momento de compra; y si la cruza desde arriba, será una señal de venta. Mientras no haya cruce de medias, se asume que la posición se mantiene en cartera sin variación.

En este algoritmo se utilizan las medias ya comentadas pero, además, se implementan señales de compra y venta que sirven para indicar en qué sentido se está realizando el cruce de medias: si la señal previa es **False** y la señal actual es **True** significa que la media *rápida* cruza desde abajo; si la señal previa es **True** y la señal actual es **False** significa que la media *rápida* cruza desde arriba.

Aprovechando la información de las señales se guarda una información en otras columnas para saber en qué momentos comprar.

MMS_ráp	MMS_len	señal	señal_prev	comprar	vender
..	..	1
..	..	0	1	0	1
..	..	0	0	0	0
..	..	1	0	1	0

Tabla 3.1: Cómo almacenar señales para algoritmo de seguimiento de tendencias

El pseudocódigo, suponiendo que se utiliza *Python* y *pandas* es¹¹:

```
def algoritmo(df):
    df["señal"] = df["mms_rápida"] > df["mms_lenta"]
    df["señal_previa"] = df["señal"].shift(1)
    df["comprar"] = (df["señal_previa"]==0) and (df["señal"]==1)
    df["vender"] = (df["señal_previa"]==1) and (df["señal"]==0)
    ...
    return
```

En la implementación de este algoritmo es habitual asumir que no se permiten posiciones cortas en la cartera. Además, cada vez que se obtiene una señal de compra o venta, el algoritmo entiende que se usa todo el dinero disponible para comprar o que se liquida toda la posición. Esto tiene sentido, porque las señales no se basan en el precio, sino en los momentos de compra y venta, es decir, en el tiempo.

Adicionalmente, en la tabla de datos habrá una columna *invertido* que indicará si el usuario tiene el valor comprado o no. Esto nos permitirá medir si nuestra inversión se ve afectada por los retornos. Es decir, si estamos invertidos en el valor, nuestro patrimonio se verá afectado por el retorno del día y si no estamos invertidos nuestro capital se mantendrá constante. Esta idea se puede implementar de diversas formas, pero en este trabajo se sigue el siguiente enfoque:

```
invertido = False

for fila in df:
    if invertido and fila["vender"]:
        invertido = False
    if not invertido and fila["comprar"]:
        invertido = True
    fila["invertido"] = invertido
```

Con esta idea, la columna `df['invertido']` va estar retrasada un día, esto es así porque no podemos comprar al final del día - cuando ya ha cerrado el mercado -, i.e., no podemos estar invertidos hasta el siguiente día,

¹¹Para calcular los retornos y medias se utilizan precios de cierre, i.e., se entiende que no se pueden hacer compras ni ventas en la apertura del mercado y, por tanto, hay que hacer un `shift(1)` en las señales, para que tenga sentido

que es cuando calculo el retorno. Pero ese retorno ya le habré cambiado con un `shift()`, lo cual, desde mi punto de vista, es más conveniente.

Una vez que sabemos si se está invertido en un valor, o no, lo que hay que hacer es calcular el retorno del algoritmo¹². Como en *Python* `True` es 1 y `False` es 0, se pueden multiplicar las columnas de *invertido* y *retorno_shift*, por ejemplo:

```
df["retorno_algoritmo"] = df["invertido"] * df["retorno_shift"]
```

invertido	retorno_shift	retorno_algoritmo
0	1	0
1	-1	-1
1	2	2

Tabla 3.2: Cómo almacenar retornos para algoritmo de seguimiento de tendencias

Selección apropiada de medias móviles

Uno de los problemas de este algoritmo puede residir en la correcta selección de las medias móviles simples que se aplicarán. En este trabajo se realiza una búsqueda por rejilla entre algunas de las medias más frecuentes:

MMS_lenta	MMS_rápida
5	30
10	50
15	200
20	-

Tabla 3.3: Medias móviles simples frecuentes

¹²El retorno se calcula habitualmente como se indica en 3.1 pero en este apartado del trabajo, en la web, se mostrará al usuario información de rentabilidad que le pueda ser de fácil comprensión

La selección de las medias se realiza por comparación entre la suma de retornos totales. El mejor retorno arrojado por el algoritmo será el de la medias escogidas¹³.

Comparar con otras estrategias

Para determinar cómo de bueno es este algoritmo de cruce de medias, se comparan los resultados con una estrategia básica, comúnmente conocida como *buy and hold* (comprar y aguantar)[88].

Estrategia basada en *Machine Learning*

En la estrategia anterior no se usan modelos estadísticos ni hay un *aprendizaje* que se pueda aplicar posteriormente. Sin embargo, sí es posible aplicar ideas de regresión o clasificación para ayudarnos a realizar mejores operaciones¹⁴.

Como ya se ha visto en la sección de ARIMA, predecir la evolución de un valor para varios días es realmente complicado. Por ello, en este apartado se seguirá una idea mucho menos ambiciosa, que es la de intentar predecir la evolución para un sólo día.

Entonces, se intentará entrenar un modelo con todos los datos disponibles hasta la fecha y estimar únicamente la evolución en la fecha siguiente. Con este tipo de estrategias, la decisión de vender, comprar o no hacer nada es algo que se tiene que actualizar diariamente.

La dificultad de esta aproximación radica en la decisión de cómo formar los datos y qué se va a predecir. En mi caso, seguiré la lógica de que hay valores altamente ponderados en un índice y que estos tienen la capacidad de *mover* ese índice *por sí solos* (o, al menos, con mayor influencia que el resto de componentes). Teniendo en cuenta los retornos de dichos valores estimaré si al día siguiente la tendencia será alcista o bajista, de forma general, para el índice de referencia.

Este enfoque se basa en seleccionar los retornos de valores de mayor capitalización de mercado [90] como si fueran atributos de un modelo de regresión o clasificación. Y la media de los retornos se considerará la clase objetivo. De esta forma, basándome en la hipótesis del aprendizaje inductivo,

¹³En `Lab.views._algoritmo_cruce_medias_automaticas()` se puede consultar más información

¹⁴En este trabajo se utilizará la librería `scikit-learn` con algunos de sus modelos

supondré que es posible que un modelo que aproxime bien una función en los datos de entrenamiento, también lo hará bien en los datos desconocidos.

La selección de los valores que *representarán* a cada índice serán:

DJ30	IBEX35	FTSE100	DAX40
Microsoft (MSFT)	Inditex (ITX.MC)	SHELL (SHEL.L)	SAP (SAP.DE)
Apple (AAPL)	Iberdrola (IBE.MC)	AstraZeneca (AZN.L)	Siemens AG (SIE.DE)
United Health (UNH)	Banco Santander (SAN.MC)	HSBC (HS-BA.L)	Allianz (ALV.DE)
Johnson and Johnson (JNJ)	BBVA (BBVA.MC)	Unilever (ULVR.L)	Airbus (AIR.DE)
JP Morgan Chase (JPM)	Caixabank (CABK.MC)	BP (BP.L)	Deutsche Telekom

Tabla 3.4: Valores seleccionados de cada índice

Cabe preguntarse por qué no se usan los datos de todos los componentes de cada índice y la respuesta es que, en esos casos, los modelos tienden a sobreajustar y no nos interesa. También es adecuado pensar en el porqué de usar retornos en lugar de precios de cierre; esto lo haré así porque los retornos - como la se ha explicado en la sección de ARIMA - siguen, habitualmente, una distribución normal, con una varianza similar a lo largo del tiempo. Entonces, como los modelos de *Machine Learning* no son buenos extrapolando, si los entrenara con precios de cierre y luego en los test aparecieran datos por encima o por debajo, el modelo *no sabría qué hacer*. Sin embargo, con los retornos esto es menos probable que ocurra, porque se comportan como series estacionarias en la mayoría de los casos.

Preprocesado de datos

Para poder utilizar esta estrategia es necesario adaptar los datos para poder trabajar con ellos de manera adecuada.

MSFT	AAPL	UNH	JNJ	JPM	DJ30
1	1	1	1	1	2
2	2	2	2	2	3
3	3	3	3	3	4
..

Tabla 3.5: Datos con estrategia basada en ML

La idea es utilizar los retornos actuales de los valores como entradas del modelo para predecir lo que ocurrirá el día siguiente, por tanto, es adecuado que la columna objetivo esté *adelantada* un día - de nuevo, con un `shift()` -. A la hora de trabajar con los datos será mucho más sencillo tener toda la información en una sola fila. Así, los datos para usar con los modelos estarán distribuidos de la siguiente manera:

	MSFT	AAPL	UNH	JNJ	JPM	DJ30	
Xtrain	1	1	1	1	1	2	Ytrain
	2	2	2	2	2	3	
Xtest	3	3	3	3	3	4	Ytest
	

Figura 3.16: Conjuntos de entrenamiento y test en estrategia basada en ML.
Fuente: elaboración propia

Algoritmo

Al utilizar modelos de clasificación o de regresión tendremos algo similar a `predicción = model.predict(X)`. Entonces, puedo utilizar la predicción para guardar información de si un usuario está invertido en ese momento, o no, con un código similar a este:

```
predicción = model.predict(X)
...
if predicción > 0:
    df["invertido"] = 1
else:
    df["invertido"] = 0
```

Es decir, si la predicción es mayor que 0, entonces se genera una señal de compra o, simplemente, se mantiene la inversión si ya estuviera activa; y al contrario para predicciones menores que 0. Esto permite conocer, con cada predicción, si mantenemos la inversión o no. Y una vez que se conoce el estado de invertido/no invertido se realiza el mismo proceso que en la tabla 3.2, multiplicando:

```
df["retorno_algoritmo"] = df["invertido"] * df["retorno_shift"]
```

Además, esta técnica permite sumar todos los retornos del algoritmo y compararlo con otras estrategias como, por ejemplo, la denominada *buy and hold*, ya mencionada cuando se explicaba el algoritmo de cruce de medias.

Modelo de regresión lineal

La regresión lineal es una técnica estadística utilizada para modelar la relación entre una variable dependiente continua y una o más variables independientes. Esta técnica asume que existe una relación lineal entre las variables. Es uno de los modelos más simples y más utilizados en análisis predictivo.

El objetivo de este modelo es predecir un valor continuo basado en uno o más valores predictivos. Cuando sólo tenemos una variable independiente la ecuación que define al modelo es:

$$y = \beta_0 + \beta_1 x \quad (3.50)$$

Donde β_0 es la constante de la recta de regresión y β_1 la pendiente de dicha recta. Las fórmulas para calcular estos coeficientes son:

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \quad (3.51)$$

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (3.52)$$

Si tuviéramos múltiples variables independientes, la fórmula se puede generalizar a:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon \quad (3.53)$$

Donde, y es la variable dependiente, x_i es la variable independiente i , β_0 es la constante de la ecuación de la regresión lineal múltiple, β_i es el coeficiente de regresión asociado a la variable x_i y ϵ es el error o residuo.

Para poder usar este modelo con ciertas garantías se deben de cumplir una serie de supuestos:

- Tiene que haber una relación lineal entre la variable dependiente y las variables independientes.
- Las observaciones deben de ser independientes entre sí.
- La varianza de los errores tiene que ser constante a lo largo de todas las observaciones, es decir, tiene que existir homocedasticidad[96].
- Los errores del modelo seguirán una distribución normal.

Este modelo, habitualmente, se evalúa con el coeficiente de determinación R^2 (proporción de varianza en la variable dependiente que puede ser explicada por las variables independientes) o a través de los errores: MSE (Error Cuadrático Medio) y RMSE (Raíz del Error Cuadrático Medio).

En este trabajo, la aplicación de este modelo se realiza sobre retornos logarítmicos, que tienen ciertas ventajas como la aditividad (pueden sumarse sobre varios periodos), la tendencia a seguir una distribución normal y que tienen cierta simetría. Sin embargo, este tipo de retorno no es ideal porque los errores no suelen tener una varianza constante - aunque sí se mantiene en el tiempo en unos rangos -, es decir, no son del todo homocedásticos y, además, pueden tener autocorrelación, lo que también afectaría al modelo.

Modelo de regresión logística

La regresión logística es un modelo estadístico utilizado para realizar clasificación binaria, es decir, para predecir la probabilidad de que una observación pertenezca a una de dos posibles categorías. Aunque el nombre puede llevar a engaños, se trata de un modelo de clasificación.

La ecuación (a veces representada como $\text{logit}(p)$) de la regresión logística es:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n \quad (3.54)$$

Entonces, la probabilidad de que la variable dependiente tome un valor 1¹⁵ se calcula de la siguiente manera:

¹⁵En el código de este trabajo tomará valores `True`, que es asimilable a 1.

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}} \quad (3.55)$$

De nuevo, para poder utilizar este modelo de forma adecuada se deben de cumplir una serie de condiciones:

- Tiene que haber independencia de los errores, es decir, las observaciones tienen que ser independientes entre sí.
- La relación entre las variables independientes y el $\ln(\frac{p}{1-p})$ de la variable dependiente debe de ser lineal.
- Las variables independientes no deben estar altamente correlacionadas entre sí.

Los modelos de clasificación se suelen evaluar a través de una matriz de confusión, que nos permite deducir la precisión, el *recall*, la sensibilidad o la especificidad, entre otros:

		Predicción	
		True	False
Real	True	TP	FN
	False	FP	TN

Tabla 3.6: Matriz de confusión

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$Especificidad (True negative rate) = TN / (TN + FP)$$

$$Sensibilidad (True positive rate) = TP / (TP + FN)$$

Sin embargo, en este trabajo se ha optado por una comparación más práctica a efectos de interpretación de un usuario. Por ello, aunque sería posible obtener una matriz de confusión sin demasiada complicación, se ha preferido hacer una comparación con una estrategia *buy and hold* para el mismo período de tiempo en el que trabaje el modelo. De esta manera, el usuario podrá determinar si es mejor una estrategia u otra¹⁶.

¹⁶Ver el método `Lab.vies._clasificacion_regresion_logistica` para más información

Los mismos problemas relativos a los retornos logarítmicos, comentados en la subsección previa, se pueden considerar para este modelo¹⁷.

¹⁷Se recomienda al usuario la realización de múltiples pruebas, para recabar sus propias conclusiones sobre la idoneidad de un modelo u otro

4. Técnicas y herramientas

4.1. Técnicas metodológicas

Scrum

Scrum [109] es un marco de trabajo relativamente estructurado y con roles específicos dentro de la metodología Agile (roles principales: *Product Owner*, *Scrum Master* y desarrollador) . Se puede utilizar tanto para la gestión de proyectos como para el desarrollo de productos, especialmente en el despliegue de *software*.

Con *Scrum* los proyectos se dividen en iteraciones cortas llamadas *sprints*. Al final de cada *sprint* se debe presentar un producto mínimo viable y evaluar lo que se ha hecho bien y lo que se puede mejorar.

Se ha optado por esta metodología, frente a otras como *Waterfall* [93] , porque ofrece una alta adaptabilidad y genera entrega temprana de valor, con productos viables y valorables por el usuario final desde las primeras fases.

Test-Driven Development (TDD)

TDD [94] es una metodología de desarrollo de software que se enfoca en escribir una batería de tests automatizados antes de iniciar la implementación del código fuente del propio software. Posteriormente, se hace un proceso de refactorización para mejorar o solucionar los defectos encontrados.

Mis conocimientos previos de *Django* y *SQLite* no me han permitido utilizar de forma integral esta metodología, pero sí que se ha seguido en

diferentes etapas del desarrollo, mejorando notablemente la calidad del código final.

Behavior-Driven Development (BDD)

BDD [86] es una metodología que se basa en el comportamiento del software y me ha resultado útil en aquellas fases del proyecto en las que no tenía una idea preconcebida del cómo trabajar con *Django* pero sí que conocía el resultado final esperado.

La ventaja de este enfoque es que las pruebas se escriben en un lenguaje natural y es sencillo extrapolarlas a un gestor de tareas con un sistema Kanban.

Kanban

Kanban [100] es un método visual de gestión de proyectos a través de la utilización de un tablero, en el que se disponen una serie de tarjetas con las tareas pendientes, en curso o finalizadas. Esto permite crear un flujo de trabajo que prioriza aquellas tareas más urgentes o que aportan antes valor a un producto.

4.2. Patrones de diseño

Model-View-Template (MVT)

Es el patrón de diseño de *Django*, Modelo-Vista-Plantilla [28, 62], que es similar al Modelo-Vista-Controlador (MVC) [104]. En *Django*, el Modelo representa la estructura de los datos, la Vista maneja la lógica de la aplicación (el controlador en MVC) y la Plantilla se encarga de la presentación de los datos (la vista en MVC).

Una de las ventajas de *Django* es que este modelo está plenamente integrado y promueve un acoplamiento débil, lo que facilita el mantenimiento y la escalabilidad de una aplicación.

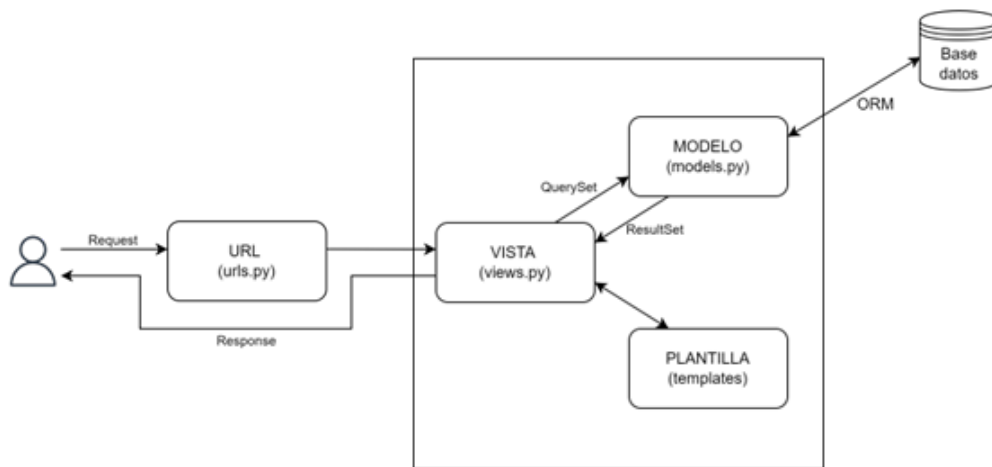


Figura 4.1: Patrón MVT. Fuente: realización propia

4.3. Control de versiones

- Herramientas consideradas: Git [34], Apache Subversion [1] y Mercurial [17].
- Herramienta elegida: Git.

Git y Mercurial son sistemas de control de versiones distribuidos (DVCS), mientras que Subversion - o SVN - es centralizado (VCS).

Una de las ventajas de Git es que permite a cada desarrollador tener una copia en local del repositorio completo y, aunque es menos eficiente para proyectos muy grandes, es más sencillo de utilizar para proyectos pequeños. Además, el sistema de ramificación de Git es más intuitivo y facilita la tarea de los desarrolladores.

4.4. Alojamiento del repositorio

- Herramientas consideradas: GitHub [36], GitLab [38] y Gitea [35].
- Herramienta elegida: GitHub.

Me he decantado por GitHub porque ya lo conocía, porque se utiliza en algunas asignaturas del Grado de Ingeniería Informática y porque es muy popular, lo que facilita la resolución de problemas gracias a su mayor comunidad.

GitHub puede ofrecer menor control sobre proyectos grandes - Gitea y GitLab permiten auto hospedaje con la configuración que más nos interese -, pero en proyectos medios o pequeños es una herramienta práctica y sencilla de utilizar, con diferentes integraciones y que facilita el uso de flujos de trabajo CI/CD.

4.5. Gestión del proyecto

- Herramientas consideradas: Zube [44], ZenHub [116], Trello [6] y Jira [5].
- Herramienta elegida: Zube.

Zube es una plataforma de gestión de proyectos que se integra muy bien con GitHub. Además, permite la sincronización en tiempo real con el repositorio de referencia que se esté utilizando y ofrece una interfaz fácil de utilizar con posibilidad de seguimiento a través de *burndowns*, *burnups* y *throughput* del equipo de desarrollo o de los desarrolladores de forma individual.

Frente a las alternativas valoradas, Zube ha sido la más intuitiva, permitiendo hacer seguimiento y planificación del proyecto en pocos pasos.

4.6. Comunicación

- Herramientas consideradas: email, GitHub y Microsoft Teams [59].
- Herramientas elegidas: todas las anteriores.

La comunicación en tiempo real, con llamadas o vídeo llamadas a través de Teams, aporta soluciones rápidas por el continuo flujo de preguntas-respuestas. Pero no siempre se pueden utilizar estos medios y es preferible hacer uso de email o de *requests* de *GitHub*. Además, recientemente, se ha dado la posibilidad de integrar MS Teams con GitHub [61] para enviar notificaciones a un grupo de trabajo.

4.7. Entorno de desarrollo integrado (IDE)

- Herramientas consideradas: Spyder IDE [42], Visual Studio Code [60].
- Herramienta elegida: Visual Studio Code.

A pesar de que ambos entornos tienen plugins de alta calidad, VS Code ofrece mayor personalización. Además, VS Code es integrable con *GitHub* de forma sencilla y ofrece aplicaciones de terceros que facilitan tanto la implementación de código como las labores de testeo y control de calidad.

VS Code se ha utilizado en este trabajo para desarrollo de *Django*, como editor *CSS* y *HTML*, para *JavaScript* y para *Markdown*, así como medio de integración con *GitHub*, entre otros.

4.8. Documentación de la memoria

- Herramientas consideradas: Texmaker [12] y TeXstudio [81].
- Herramienta elegida: Texmaker.

Texmaker es un editor de texto gratuito, multiplataforma y que integra diversas herramientas necesarias para desarrollar documentos \LaTeX . *Texmaker* incluye soporte *Unicode*, corrección ortográfica, auto-completado y un visor de PDF incorporado que es realmente útil.

Adicionalmente, cabe destacar que la integración de *Texmaker* con la distribución de \TeX / \LaTeX *MikTeX* [102] es menos problemática que con *TeXstudio*.

4.9. Documentación del código

- Herramientas consideradas: Sphinx [7], Read the Docs [78] y pdoc [65].
- Herramientas elegidas: Sphinx y Read the Docs.

Sphinx es la herramienta más extendida en la comunidad *Python* para documentar código, es compatible con varios formatos y estilos de *docstrings* - *PyDoc*, *Google* o *Numpy* entre otros - y, además, puede generar documentación de manera automática a partir de los *docstrings*.

En este trabajo, se ha escogido el formato de *Numpy* para la documentación de código y el estilo de *Read The Docs* para las plantillas de la documentación HTML.

La documentación del código puede ser consultada en fat.readthedocs.io o en fat.rtfld.io

4.10. Integración y despliegue continuos (CI/CD)

- Herramientas consideradas: GitHub actions [37], Jenkins [47] y CircleCI [16].
- Herramienta elegida: GitHub actions.

GitHub actions es una plataforma de CI/CD que permite automatizar el proceso de compilación, pruebas y despliegue de software. En este trabajo, GitHub actions se ha utilizado para tetear y comprobar la calidad del código a través de flujo de trabajo que se activan con cada evento de *push* al *branch main* del repositorio de GitHub.

4.11. Calidad y consistencia de código

- Herramientas consideradas: Pylint [53] y Flake8 [18].
- Herramienta elegida: Pylint.

Pylint es una herramienta de análisis de código estático para *Python*, diseñada para detectar errores y mejorar la calidad del código. Este analizador de código se utilizar para verificar sintaxis, semántica y obliga a seguir las convenciones de estilo de *Python*.

Se ha escogido *Pylint* frente a *Flake8* porque, adicionalmente, permite medir la calidad del código en términos de complejidad y legibilidad, lo que favorece un mantenimiento posterior. Además, con *Pylint* podemos hacer informes que señalan todos los fallos, - aumentando la productividad - y es posible integrarlo con *GitHub actions*, como se ha hecho en este proyecto.

4.12. Cobertura de código

- Herramientas consideradas: Coverage [8] y Pytest-cov [67].
- Herramienta elegida: Coverage.

Coverage es una herramienta que permite medir la cobertura de código en *Python*. Tiene la ventaja de que está bien integrada con proyectos de *Django* y permite reconocer los *django.test.TestCase*. Además, se puede incorporar a *GitHub actions*, tal y como se ha realizado en este proyecto, a través de un documento *YAML*.

Uno de los aspectos relevantes de *coverage* es que, con pocos comandos, permite generar un informe HTML muy intuitivo que guía al desarrollador hacia los fallos detectados.

4.13. Framework web

- Herramientas consideradas: Django [33] y Flask [64].
- Herramienta elegida: Django.

Django es un *framework web* muy completo que incluye diversas características por defecto. También adopta un elevado nivel de seguridad y es altamente escalable. Es menos ligero que *Flask* pero, a cambio, ofrece mayor nivel de personalización y control sobre el sitio web, así como una mejor estructuración general de un proyecto.

4.14. Sistema gestor de bases de datos

- Herramientas consideradas: SQLite [75] y PostgreSQL [40].
- Herramienta elegida: SQLite.

SQLite es una librería de código escrita en lenguaje C, que implementa un motor de bases de datos pequeño y rápido. Se califican a sí mismos como un sistema gestor de bases de datos ligero y multiplataforma. Tiene la ventaja de estar muy extendido y utilizan un único archivo en el sistema de almacenamiento, lo que favorece su distribución y uso.

Por su parte, *PostgreSQL* tiene opciones más avanzadas que *SQLite* y está pensado para soportar alta concurrencia y bases de datos grandes. Pero *SQLite* está perfectamente integrado con *Django* y no requiere de configuración adicional como sí requeriría *PostgreSQL*. Además, *SQLite* es suficiente para las expectativas de este trabajo - la migración a *PostgreSQL* sería recomendable en caso de escalar el proyecto -.

4.15. Bibliotecas y librerías relevantes

En este apartado se indican las bibliotecas y librerías más relevantes dentro del proyecto. Hay otras muchas que forman parte del conjunto de dependencias y que se pueden consultar en el archivo de *requirements*.

Pandas

Pandas [43] es una biblioteca de código abierto para *Python* especializada en análisis de datos. Es útil para cargar datos desde diversas fuentes - como una base de datos o una API -, permite tratar los datos y transformarlos o incluso crear visualizaciones.

Plotly

Plotly [66] es una biblioteca de código abierto que se utiliza para crear visualizaciones de datos interactivas en *Python*. Permite crear gráficos de barras o líneas, entre otros, y permite dar una alta personalización a la información que recibe el usuario final.

Matplotlib

Matplotlib [27] es una librería para crear visualizaciones estáticas, animadas e interactivas en *Python*. Es una herramienta de código abierto, multiplataforma y está orientada a objetos.

La curva de aprendizaje de *Matplotlib* puede ser más compleja que en otras herramientas similares, pero ofrece mayor control sobre los datos y la forma de representarlos, especialmente cuando se utiliza para gráficos estadísticos.

yFinance

yFinance [4] es una biblioteca de código abierto para *Python* que permite el acceso y procesamiento de datos financieros. Permite la recuperación de datos históricos y en tiempo real de cotizaciones de acciones, índices bursátiles, divisas, criptomonedas y otros instrumentos financieros. En este trabajo se utiliza para recuperar datos de valores cotizados y sus divisas de referencia - siempre con precios de cierre de mercado -.

Entre las ventajas que ofrece es que su uso está bastante extendido y tiene una comunidad que facilita la resolución de problemas. Además, es fácil de utilizar y dispone de múltiples ejemplos en su documentación que permiten que la curva de aprendizaje sea progresiva.

News API

News API [2] es una biblioteca de código abierto, de *Python*, que facilita el acceso y la integración de noticias en una aplicación. Es un servicio web

que proporciona una amplia variedad de artículos y noticias actualizadas, ordenadas por diferentes categorías, países y fuentes de información.

Feedparser

Feedparser [114] es una librería que permite analizar y procesar *feeds RSS* y *Atom*. En este trabajo se utiliza para obtener información relativa a índices bursátiles desde enlaces *RSS*. Los *feeds* de los datos se proveen con archivos en formato *XML* y proceden de fuentes relacionadas con mercados financieros nacionales e internacionales (en todos los casos se utilizan fuentes conocidas y contrastadas).

NetworkX

NetworkX [26] es una biblioteca que se utiliza para el estudio y análisis de redes complejas. Entre sus funciones principales se pueden encontrar las de crear, manipular y analizar grafos. Estos grafos son estructuras matemáticas que modelan relaciones entre objetos y, en el caso de este trabajo, se utiliza para visualizar las mayores correlaciones - positiva y negativa - entre las cotizaciones de todos los valores de los índices estudiados.

Statsmodels

Statsmodels [49] es una biblioteca de *Python* que se utiliza para el análisis y modelado estadístico de datos. Permite explorar, estimar y evaluar modelos estadísticos complejos; y ofrece herramientas para el análisis de series temporales. En este proyecto se utiliza para configurar y aplicar modelos ARIMA.

Scikit-learn

Scikit-learn [72] es una biblioteca de código abierto para el aprendizaje automático en *Python*. Se utiliza frecuentemente para clasificación de datos, regresión, agrupación y reducción de dimensionalidad. Sin embargo, en este trabajo se utiliza sólo para calcular el *MSE* (Mean Squared Error) entre datos de test y predicciones de modelos y para normalizar datos en una escala concreta.

Scipy

Scipy [82] SciPy es una biblioteca de código abierto de *Python* que se utiliza comúnmente en cálculo científico y ciencia de datos. Proporciona módulos para resolver problemas matemáticos, científicos, de ingeniería y técnicos. En este trabajo se utiliza para resolver problemas de programación cuadrática y lineal.

Keras

Keras [113] es una biblioteca de código abierto para aprendizaje profundo escrita en *Python* - en este proyecto se ejecuta sobre *TensorFlow* [76] -. *Keras* sirve para crear y entrenar redes neuronales y para experimentar con diferentes arquitecturas de estas redes. Su modularidad permite crear redes muy complejas y, además, es muy eficiente tanto en entrenamiento como en inferencia. En este trabajo se utiliza para crear redes *LSTM* y realizar análisis de series temporales.

4.16. Desarrollo web

HTML

HTML [97] es el lenguaje de marcado de hipertexto estándar para crear páginas web. Es un lenguaje que utiliza etiquetas para definir la estructura y el contenido de una página web.

CSS

CSS [92] es un lenguaje de hojas de estilo que se utiliza para dar un aspecto y diseño agradables a una página web. Se utiliza junto con *HTML*.

Bootstrap

Bootstrap [10] es un *framework* de código abierto para el desarrollo web *front-end*. Proporciona un conjunto de herramientas y componentes predefinidos que permiten a los desarrolladores crear plantillas e interfaces atractivas y responsivas.

JavaScript

JavaScript [99] es un lenguaje de programación interpretado, orientado a objetos y débilmente tipado. En este trabajo se utiliza para desarrollo web *front-end* para agregar interactividad y dinamismo a la página web.

4.17. Otras herramientas

python-dotenv

python-dotenv [112] es una biblioteca que permite gestionar variables de entorno para aplicaciones *Python*. Se puede utilizar, entre otros, para securizar las *API keys* de *Django* y *News API*.

Draw.io

Draw.io [54] es una herramienta de diseño de diagramas y esquemas que incluye una amplia variedad de formas predefinidas. Permite exportar el resultado en varios formatos, lo que facilita la integración con diferentes editores de texto.

Mendeley

Mendeley [31] es una herramienta de gestión de referencias bibliográficas y colaboración académica. Entre sus características cabe destacar la capacidad de organizar las referencias y exportarlas a un archivo que se puede utilizar desde L^AT_EX.

5. Aspectos relevantes del desarrollo del proyecto

5.1. Inicio del proyecto

Este proyecto surge con el afán de aprender más sobre la obtención, el almacenamiento, el tratamiento y el análisis de datos financieros. Inicialmente me planteé otras posibilidades que podían cubrir algunas de estas ideas de manejo de información, como el análisis de datos climatológicos, pero la obtención de la información no era sencilla y, casi al mismo tiempo, descubrí que era posible conseguir los datos de valores cotizados a través de diferentes *APIs*[85] y que podría trabajar con ellos en mi entorno de trabajo local.

Por otro lado, había leído sobre la aplicación de modelos estadísticos para estimar posibles tendencias y quería comprobar el recorrido que tendrían esas estrategias. Tal vez, de fondo estuviera esa idea ingenua de que algún día nos haremos ricos con algún tipo de algoritmo mágico; pero lo cierto es que algunas de las técnicas empleadas en este trabajo han resultado interesantes y pueden ayudar a formar una cartera bien diversificada, así como a propiciar inversiones con menor riesgo del necesario. Incluso el enfoque de trading algorítmico planteado en este trabajo puede ser útil para realizar pronósticos a muy corto plazo - se realizan estimaciones de un día -.

Finalmente, al plantear mis ideas al Dr. José Ignacio Santos Martín y a la Dra. Virginia Ahedo García, ya me avisaron de que había alguna de mis ideas, como la aplicación de modelos ARIMA, que no iban a funcionar y así ha sido. Pero quería aprovechar la oportunidad para ofrecer herramientas que ayudaran a los usuarios a entender por qué este modelo no funciona, sobre todo basándome en el manejo de los datos, y qué podemos hacer los

inversores para utilizar la información a nuestro alcance con otros modelos o métodos de inversión.

5.2. Metodologías

De forma global se ha intentado seguir una metodología ágil a lo largo de todo el proyecto, concretamente, *Scrum*. El inconveniente más evidente de seguir esta filosofía en un proyecto educativo es la falta de un equipo personas que cubran los diferentes roles necesarios. Sin embargo, he intentado ponerme en el papel de cada componente del equipo, haciéndome preguntas constantemente sobre el tiempo disponible, el producto final requerido en cada sprint, qué esperaríamos un potencial cliente y cómo se comprobaría el código en un entorno laboral.

Algunos de los procesos más relevantes llevados a cabo han sido:

- Realización de iteraciones, *sprints*, de forma constante y con diferentes temporalidades dependiendo del producto esperado al final de cada periodo. Aquí he intentado realizar un esfuerzo extra en cuanto a lo esperable en las reuniones diarias y, aunque han sido discusiones conmigo mismo, he de reconocer que el resultado en general es satisfactorio, porque me ha ayudado a plantearme diferentes cuestiones que han contribuido a mejorar mi manera de trabajar y que, desde mi punto de vista, han mejorado los incrementos al final de cada *sprint*.
- Disposición de tareas, conocidas como *Issues*[98], asignadas como si se tratara de un proyecto con un equipo multidisciplinar. Estas tareas han sido creadas con un gestor de proyectos llamado *Zube*, que ha facilitado la consecución de objetivos y que ofrece una elevada integración con repositorios de *GitHub*, lo que evita tener que desplegar *issues* en diferentes entornos.
- Utilización de un panel *Kanban* integrado en *Zube*. Este tablero de tarjetas ha facilitado el seguimiento de las tareas pendientes. Además, es una forma visual rápida de detectar todo el trabajo que queda por hacer, el tiempo disponible para ello y ayuda a centrar los esfuerzos en las *issues* más urgentes.
- Implementación de un flujo de trabajo ágil, dirigido por las tareas dispuestas en el panel *Kanban*, con diferentes estados para las mismas: *inbox*, *backlog*, *ready*, *in progress*, *in review* y *done*. Los estados más utilizados han sido los *inbox* para nuevas tareas que se me iban ocurriendo según avanzaba el proyecto, la de *ready* con todas las tareas

preparadas para ser realizadas y la de *in progress* para tener claro lo que se estaba realizando en cada momento. El resto también han tenido relevancia, pero en menor grado.

Adicionalmente, se han medido la cantidad de trabajo realizado y las tareas pendientes con gráficos *burndown*[87]. Ha sido frecuente no cumplir a la perfección con los plazos esperados durante los *sprint* y esto se ha debido a la incorporación de pequeñas mejoras en los períodos ya definidos - algo que no se debería hacer -. Sin embargo, la visualización de los *burndown* me ha ayudado a dedicar esfuerzos adicionales para llegar a cumplir con prácticamente todos los objetivos al final de cada iteración.

Ensayo y error en fases tempranas del proyecto

Antes de la realización de este trabajo había utilizado *Python* de forma extensiva, en diferentes asignaturas y en el ámbito personal, sin embargo, nunca había usado *Django* y esto me obligó a realizar múltiples pruebas inicialmente.

Buena parte del código que implementé inicialmente se basó en ensayo y error, buscando guías de ayuda y utilizando la documentación de *Django*[24]. Este proceso de pruebas iniciales me ha favorecido en la fase final del trabajo, porque me ha permitido realizar los últimos *sprints* de forma más eficiente.

Diseño dirigido por pruebas. TDD

En la medida de lo posible se han intentado desarrollar tests previos a la implementación de código pero, en múltiples ocasiones, no ha sido posible. Esto se ha debido al desconocimiento previo del autor sobre el funcionamiento de *Django* y a que muchas de las pruebas iban dirigidas hacia el entorno *web* y la comprobación de los resultados esperados en los métodos.

Sin embargo, el mero hecho de haber intentado implantar este tipo de desarrollo me ayudó en fases iniciales a detectar diferentes fallos y a fortalecer la estructura de pruebas que tenía implementada.

5.3. Formación

Este trabajo ha requerido de algunos conocimientos que previamente no se tenían, especialmente en lo referente a *Django*, a la formación de carteras

diversificadas y al *trading* algorítmico. Pero ya se tenían conocimientos de modelos estadísticos, de bases de datos y de desarrollo web que han facilitado algunas de las fases del proyecto.

Las fuentes fundamentales en las que se ha adquirido el conocimiento necesario sobre *Django* han sido:

- La documentación de *Django*[24].
- Vídeos de *YouTube* de canales especializados:
 - Curso de *Django* para principiantes[32].
 - *Django full course*, del que se ha obtenido buena información, especialmente para la implementación de *routers* para el uso de diferentes bases de datos[19].
- La documentación de *Jinja*[63].

En cuanto a la parte más financiera del trabajo, además de las lecturas realizadas durante años previos sobre diferentes métodos de inversión, debo destacar los siguientes - sobre todo por la nueva visión que me han aportado en cuanto a la creación de una cartera bien diversificada y a la implementación de técnicas de *trading* algorítmico -:

- Parte de la teoría de Harry M. Markowitz, especialmente, con su libro *Portfolio selection. Efficient diversification of investments*[57].
- Curso de *Udemy* de *Financial Engineering and Artificial Intelligence in Python*. Un curso muy completo que no he terminado todavía, pero que tiene información especializada y formal que ha contribuido al desarrollo teórico de este proyecto y a la comprensión de varios conceptos. Lo imparte *The Lazy Programmer Team* y es altamente recomendable para cualquier interesado en la materia[77].
- Vídeos de *YouTube* de fuentes especializadas:
 - Serie de vídeos sobre cómo implementar la frontera eficiente en *Python*[68]¹⁸.
 - Clase del Dr. Peter Kempthorne, del MIT, sobre *Portfolio Theory*[50].
- Artículo *Modern Portfolio Theory with Python* con algunas ideas interesantes[15].

Han habido otra muchas fuentes y recursos consultadas pero, sin duda, las más relevantes son las expuestas en esta sección.

¹⁸Aunque no se ha seguido esta técnica exactamente, por el propio proceso de obtención de datos, sí que se han conseguido resultados similares.

5.4. Obtención y procesamiento de datos

La obtención de la información se planteó como algo fundamental en las primeras fases del proyecto y, hasta las fases finales se ha mantenido la estrategia que se decidió adoptar del uso de una API. Para conseguir información de mercados financieros hay diferentes APIs, (*Alpha Vantage*, *Finage* e *IEX CCloud*, entre otras) pero muchas de ellas son de pago o permiten un acceso restringido a pocos datos. Sin embargo, hay una API que está bastante extendida que se llama *yfinance*[4]. Esta API, probablemente, sea la mejor ahora mismo para obtener datos de valores cotizados de los principales mercados financieros de forma gratuita. Hace uso de la información de *Yahoo Finance* y es muy utilizada en entornos en los que no es necesario un uso intensivo de datos en tiempo real, como es el caso de este trabajo. Su popularidad en *GitHub* es merecida[3].

Teniendo claro cómo obtener los datos y que se iban a almacenar en una base de datos SQLite3, decidí crear unos modelos en *Django* (cada valor cotizado tenía su propio modelo y su correspondiente tabla, creados de forma manual) que permitieran almacenar la información necesaria. Inicialmente sólo tenía una base de datos para el índice DJ30 y en ella se mezclaba información sobre otros índices. Pero rápido se detectó que la estructura no era adecuada. Entonces, la división en diferentes bases de datos, con múltiples modelos creados de forma semiautomática, han resultado ser de gran utilidad y han permitido extender a posteriori la información disponible sin demasiadas complicaciones.

Paso 1. Creación de modelos

Tras tener claro de dónde obtener los datos empecé a trabajar con algunos modelos muy básicos, que se creaban a mano y se migraban a una única base de datos. Esta forma de trabajar no era práctica e invertí una buena cantidad de tiempo en descubrir cómo generar modelos de forma automática a partir de una lista de *tickers*. La idea fue generar listas con los *tickers* disponibles de cada índice, lo que cual es sencillo y, a partir de esas listas ir creando modelos automáticos para diferentes bases de datos.

La creación de modelos fue uno de los mayores problemas encontrados al inicio del trabajo, sobre todo porque quería que fuera escalable a más índices y valores cotizados (hasta el punto de que ahora mismo es viable añadir una nueva base de datos en relativamente pocos pasos). La estrategia seguida fue la de crear un modelo base común e ir modificando el nombre de

la tabla de manera dinámica. Así, cada valor cotizado tiene su propia tabla en la base de datos de su índice. La forma de conseguirlo fue la siguiente¹⁹:

Crear una clase base con una serie de atributos (id, precios, moneda, etc.)

Crear un diccionario de clases vacío

Recorrer una lista con todos los tickers de todos los índices:
 Por cada ticker generar una clase dinámica del tipo de la clase base

 Asignar a la clase dinámica el nombre de la tabla como el del ticker

 Crear una nueva entrada en el diccionario con la nueva clase dinámica

Hacer el diccionario accesible a los scripts de creación de bases de datos

Paso 2. Enrutamiento a la base de datos adecuada

Como se iban a utilizar diferentes bases de datos y tenía que redirigir la información de manera adecuada configuré un *router* siguiendo los pasos indicados en la documentación de *Django*[29]²⁰

Paso 3. Realizar las migraciones

Para poder migrar la información a las bases de datos es necesario configurar los motores en *Django*:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'databases/db.sqlite3',  
    },  
}
```

¹⁹Consultar `Analysis.models.py` para más información.

²⁰Se puede consultar más información sobre el multi routing en el código del proyecto, en `FAT.routers.router_bases_datos.py`

```
'dj30': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'databases/dj30.sqlite3',
},

'ibex35': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'databases/ibex35.sqlite3',
},

'ftse100': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'databases/ftse100.sqlite3',
},

'dax40': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'databases/dax40.sqlite3',
},
}
```

Tras haber configurado de manera adecuada las bases de datos ya se pueden hacer las migraciones con los comandos `makemigrations` y `migrate` que proporciona el *framework*. Esto es especialmente interesante porque nos evita el tener que realizar las migraciones a mano o tener que generar las consultas SQL necesarias. Además, gestiona el guardado de la información.

Paso 4. Almacenamiento de los datos

Cada índice tiene su propia base de datos y, adicionalmente, hay otra base de datos común a todas ellas, que tiene información sobre los usuarios, sobre las carteras de los usuarios, sobre divisas (para realizar cálculos adecuados en las carteras de los usuarios) y sobre los sectores de los valores (para agilizar las búsquedas y poder realizar comparaciones)²¹.

²¹La estructura de las tablas de las bases de datos puede verse en los anexos de este trabajo

El pseudocódigo para la obtención de la información y el volcado inicial a las bases de datos es el siguiente²²

```
Inicio de la función crear_bds(índice, bd, logger):
    Registrar información en el logger
    Establecer conexión a la base de datos
    Intentar:
        Iniciar transacción con la conexión (asegurando
        atomicidad)

    Para cada ticker en el índice:
        Obtener datos históricos del stock
        (stock = yfinance.Ticker(ticker))

        Ajustar formato de fechas según la base de
        datos

        Agregar columnas relevantes a los datos históricos

        Calcular medias móviles y añadir información de
        la compañía

        Verificar existencia de la tabla en la base de datos
        Si la tabla existe:
            Guardar los datos en la base de datos
            Registrar éxito en el logger
        Si no:
            Registrar error en el logger

    Capturar excepciones de SQLite y otras
    Finalmente:
        Cerrar conexión a la base de datos
        Registrar finalización del proceso en el logger
    Retornar
```

²²Para ver cómo se rellenan las bases de datos se puede consultar el script `util.CrearBDs.py`

5.5. División por aplicaciones

Cuando se trabaja con *Django* es habitual implementar distintas aplicaciones que aporten una solución, a cada problema planteado, de forma individual. Esta estructura y la filosofía subyacente en ella me han permitido ir ampliando los recursos disponibles en la web.

En la primera fase sólo tenía como objetivo almacenar la información en bases de datos, recuperar esa información y mostrarla en una web. Pero poco a poco fui descubriendo el potencial de *Django* y vi que podía añadir nuevas funcionalidades que resultaran útiles.

Todas esas funcionalidades, que en cierta medida esperaba poder desarrollar desde el principio, tenían que seguir un orden lógico, así que decidí crear mi propia estructura de aplicaciones internas:

Aplicación	Funcionalidad
FAT	Aplicación raíz del proyecto Gestión y configuración de otras aplicaciones Control de migraciones de bases de datos Control de enrutamiento a bases de datos
Analysis	Registro y login de usuarios Mostrar datos de componentes de índices bursátiles Gráficas interactivas de valores cotizados Comparación de un valor con su sector de referencia Comparación entre diferentes valores Grafo de correlaciones entre valores Noticias RSS
DashBoard	Control de valores en cartera Información sobre evolución de una cartera Gráfica de Markowitz y ratio de Sharpe Control de valores en seguimiento Diagramas sobre distribución en divisas y sectores
Lab	Forecasting de series temporales con ARIMA Trading algorítmico con cruce de medias Predicción con estrategias basadas en ML
News	Noticias de portada Control de gráficos de portada

Tabla 5.1: Estructura de aplicaciones

Además, existen otras rutas relevantes dentro del proyecto, en las que se encuentran archivos estáticos, *scripts* de utilidad (con fuentes RSS y métodos para manejar los *tickers*), tests y un log que permite controlar, de manera interna, cómo han funcionado los tests²³.

Por otro lado se pueden encontrar las carpetas de documentación, *docs*, del trabajo y la ruta con el índice para revisar la cobertura del código, *htmlcov*, generada con la herramienta *coverage*²⁴.

5.6. Fuentes de noticias

Además de datos de cotizaciones, en la web se pueden consultar diversas noticias tanto en la página principal como en la página general de cada índice. De forma casi experimental, en la página principal de la aplicación se utiliza una API adicional que es *NewsAPI*. Esta fuente de noticias tiene la ventaja de que provee imágenes asociadas a la información y que se puede hacer un proceso de filtrado sobre diferentes campos y en distintos idiomas. Sin embargo, tras múltiples pruebas, he detectado que su información suele centrarse en noticias de la India y EEUU, y que es muy complicado obtener noticias de España. Pero, en cualquier caso, he decidido dejar esta herramienta habilitada para que la página principal resulte más atractiva.

Por otro lado, se hace uso de fuentes RSS con noticias relacionadas a cada índice bursátil. Las fuentes son públicas y los *feeds* se parsean gracias al módulo *feedparser*[114].

5.7. Securizar claves de APIs

Cuando se utiliza una API suele necesitarse una clave para conectarse a ella. El problema de estas claves es que no se deben de hacer públicas y, por tanto, hay que protegerlas. Con la API de *yFinance* no es necesaria una clave, pero con *NewsAPI* sí. Por su parte *Django*, además, genera su propia clave (denominada *SECRET_KEY*).

Sin estas claves no funciona el proyecto, así que tuve que buscar una manera de favorecer el funcionamiento pero sin comprometer la seguridad. Por supuesto, ya había algo desarrollado que me facilitaría el proceso y ese algo era *dotenv* y las variables de entorno.

²³Ver los directorios *static*, *util*, *tests* y *log* respectivamente.

²⁴Ver la estructura completa de los directorios en los anexos

Tanto en el repositorio de *GitHub* como en la web publicada se han eliminado las claves y se han sustituido por variables de entorno que permiten la ejecución del código sin problema. En el repositorio se crean variables de entorno para poder utilizar las *GitHub actions*, concretamente, la que permite comprobar la cobertura del código con *coverage*²⁵. Y en la web se han creado variables de entorno con una consola en el servidor.

Dentro de este proyecto, en `FAT/.env.example`, se ha dejado un archivo de ejemplo, para la configuración de las variables de entorno. Cada usuario que descargue el repositorio deberá utilizar sus propias claves, que son fáciles de conseguir en NewsAPI y al al crear nuestro propio proyecto de *Django*²⁶. Una vez sustituidas las claves de `FAT/.env.example` sólo es necesario cambiar el nombre de ese mismo archivo a `FAT/.env` y todo debería de funcionar con normalidad.

5.8. Desarrollo backend y frontend de *DashBoard* de usuario

Una de las fases del trabajo en las que más tiempo se ha invertido ha sido en el desarrollo de una interfaz que permitiera al usuario ver toda la información relativa a su cartera y que pudiera entender, con facilidad, qué posibilidades existían de mejora, haciendo uso de información sobre una gráfica de Markowitz y unas tablas con información sobre el reparto de pesos de los distintos valores cotizados dentro de la propia cartera.

Inicialmente, la información se mostraba con tablas muy básicas que no permitían entender los resultados obtenidos con claridad. Tras realizar algunas mejoras, como la inclusión de funciones de *JavaScript*, la comprensión de los resultados fue mejorando.

En este apartado he tenido que realizar modificaciones continuamente, porque iba detectando *bugs* o posibles mejoras que resultaran útiles. En una primera aproximación sólo tenía tablas de valores en cartera, posteriormente añadí información sobre valores en seguimiento, luego detecté que se debía mejorar la contabilidad utilizando una única divisa (me decanté por el euro, por motivos evidentes) y así sucesivamente.

Entre las mejoras más interesantes realizadas a lo largo del tiempo, destacaría los siguientes:

²⁵Ver el código de `/.github/workflows/coverage.yml` para ampliar información

²⁶Ver la guía de usuario de los anexos para mayor información

- Creación de un *donut* para obtener información agregada sobre los valores en cartera (con cambio a EUR automático para que el usuario tenga una referencia clara).
- Inclusión de un diagrama de barras que permite ver el reparto de pesos de los valores según el sector al que pertenecen.
- Añadir información de valores en el mismo sector que aquellos que el usuario tenga en seguimiento.

Todas estas modificaciones iban en el mismo sentido: tratar de que los usuarios tuvieran una herramienta para buscar los mejores valores con los que diversificar su cartera.

Finalmente, en las últimas modificaciones conseguí añadir una gráfica de Markowitz junto con el ratio de Sharpe, así como una tabla que permite ver el reparto de pesos actual y lo que sería idóneo según la *Modern Portfolio Theory*. En el backend²⁷ de esta gráfica hay una serie de métodos de minimización que son muy interesantes, pero que me costó implementar, especialmente por la inclusión de restricciones adicionales a los problemas (problemas LP y QP, como se ha explicado en la sección de teoría de esta memoria).

Todos estos cambios implicaron la creación de modelos adicionales para guardar información sobre los valores en cartera y en seguimiento de un usuario - `StockComprado` y `StockSeguimiento`²⁸ - y sobre el cambio de divisas - `CambioMoneda`²⁹ -. Según iba creando nuevos modelos iba realizando mejoras incrementales sobre la creación de las bases de datos.

5.9. Desarrollo backend y frontend del *Lab*

Implementación de herramientas para modelo ARIMA

Los primeros pasos para interactuar con modelos ARIMA fueron los más sencillos posibles. Inicialmente sólo intentaba mostrar un informe ARIMA/SARIMAX con `auto_arima` de `pmdarima` he iba realizando comprobaciones de las métricas obtenidas.

²⁷Ver el método `DashBoard.views.mostrar_markowitz_frontera_y_mejores()` y todos sus métodos auxiliares

²⁸Ver `DashBoard.models.py`

²⁹Ver `Analysis.models.py`

Posteriormente fui añadiendo formularios para que el usuario pudiera interactuar con el modelo en la búsqueda de los mejores parámetros (p , d , q) posibles. En esta fase me tuve que plantear muchas preguntas:

- Un usuario experimentado sabe que este modelo no va a funcionar ¿qué puedo ofrecer que le resulte interesante?
- ¿Una persona que no haya utilizado ARIMA con anterioridad puede manejarlo de alguna manera muy sencilla?
- ¿Qué le gustaría ver a una persona que sólo va a analizar los datos?

Tratando de contestar a todas estas preguntas fui formando conjuntos de datos fácilmente entendibles y gráficas que resumían las estimaciones. Además, decidí añadir unas tablas que complementan la información de las gráficas, mostrando los errores cometidos, para prevenir al usuario final de utilizar este modelo a través de la comparación con una estrategia naïf - contrastar con un modelo que estima que el día siguiente el precio será el mismo que el día actual -.

Por otro lado, intenté desarrollar un método de validación *walk forward anchored* que permitiera hacer comprobaciones diarias. Mi idea se basó en los códigos del PhD Jason Brownlee[13], pero realicé pequeñas modificaciones que me permitían comprobar el acierto en la tendencia, ya que el precio no me interesaba. El pseudocódigo es el siguiente:

```
función validación_walk_forward():
    Inicializar modelo, predicciones y aciertos_tendencia

    Dividir datos en conjuntos de entrenamiento y test

    Inicializar conjunto_total

    Para cada paso t en el conjunto de test:
        Ajustar modelo ARIMA con orden y conjunto_total
        Predicción del valor siguiente
        Agregar predicción a predicciones
        Agregar valor real a conjunto_total

    Si t > 0, obtener valor anterior del conjunto de test;
    si no, del conjunto de entrenamiento
```

```
Determinar si la predicción fue correcta según valores  
anterior y real
```

```
Registrar la precisión en aciertos_tendencia
```

```
Devolver modelo, aciertos_tendencia y predicciones
```

Por último, integré las gráficas de las funciones ACF y PACF, que permiten a los usuarios más avanzados extraer los mejores parámetros (p, d, q) por sí mismos.

Implementación de algoritmos de trading

En este apartado trabajé de forma casi exploratoria, porque no tenía conocimientos previos sobre cómo aplicar de estas técnicas, aunque conocía que se podían implementar.

Estrategia de cruce de medias

Inicialmente, generé los datos necesarios para la estrategia de cruce de medias móviles, algo que con `pandas` es casi trivial:

```
# Ejemplo para una MM50  
datos['MM50'] = datos["precio_cierre"].rolling(50)
```

Posteriormente, generé el algoritmo como se explica en el apartado teórico de esta memoria. Cabe destacar que lo más complicado es entender el porqué de las señales de este algoritmo y cómo se retrasan los valores que marcan si se está invertido, o no, para obtener los resultados deseables.

El objetivo con la estrategia de cruce de medias era claro: buscar las medias móviles simples que ofrecieran la mayor rentabilidad en un período concreto, para que el usuario dedujera si merecía la pena seguir ese mismo enfoque en sesiones futuras. Es decir, no se trata de predecir un valor futuro, sino de descubrir si una estrategia ha funcionado bien en el pasado y cómo de bueno sería aplicarlo a posteriori.

De nuevo llegué a un punto en el que necesitaba comparar los resultados para saber cómo de bien se comportaba este método y la idea fue la de comparar contra una estrategia *buy and hold* en la misma ventana temporal

en la que se utiliza el cruce de medias. Tras múltiples pruebas, puedo intuir que la estrategia *buy and hold* suele obtener mejores rentabilidades, pero hay cierto nivel de dependencia con las temporalidades y el valor seleccionado³⁰.

Estrategia basada en *machine learning*

El enfoque para utilizar esta estrategia es distinto al utilizado para el método de cruce de medias en el sentido de que aquí sí se busca una predicción para la próxima sesión. Es decir, lo que se quería saber desde el principio era si un modelo de regresión o de clasificación podía estimar si la sesión futura sería alcista o bajista, sin importar el precio final.

Para poder utilizar estos modelos era esencial disponer de la información bien estructurada y que tanto los valores de entrenamiento como los de test tuvieran cierto sentido. Empecé por probar los modelos sobre un sólo valor, pero los resultados no tenían demasiado sentido, así que decidí seguir un enfoque típico de minería de datos, formando un conjunto de valores cotizados como si fueran atributos y una clase a predecir que sería el índice bursátil de referencia de esos mismos valores cotizados. La selección de los valores se podía haber realizado de múltiples formas (por ejemplo, con un `Pipeline` con elección de atributos a través de un modelo `RandomForestClassifier()`) pero decidí que lo más adecuado era darle especial relevancia a los valores que tienen mayor peso en los propios índices. Esta decisión se basa en la capitalización de mercado de cada valor y en la capacidad que tienen algunos valores para *mover* por sí solos la cotización de un índice, precisamente, por tener mayor peso en ellos.

En la fase preliminar de este apartado generé pruebas externas a la plataforma web. Extraje información de mis bases de datos y realicé adaptaciones para ver si era viable utilizar estos modelos en términos de tiempos de ejecución. Los resultados no pudieron ser más satisfactorios, porque se tardaba relativamente poco tiempo en generar una salida y mostrar la información deseada.

Una vez comprobado que los modelos funcionaban de manera adecuada, empecé a integrar estas soluciones en el proyecto. De nuevo, la técnica para la generación del algoritmo se detalla en en la sección teórica de esta memoria, porque resultará más práctico para el lector. Pero se puede destacar - de manera similar al método del cruce de medias - que lo más relevante es

³⁰Los resultados son tan variables que es recomendable que el usuario saque sus propias conclusiones con los resultados mostrados en la web.

entender que se tienen que retrasar los precios de cierre de los índices un día, para que la estimación que hagan los modelos tenga sentido.

Tras tomar la decisión de qué valores debían representar a cada índice y con los datos ordenados, sólo quedaba aplicar los modelos y mostrar los resultados. En esta ocasión se decide no disponer los resultados con una gráfica, porque no aportan valor añadido a la interpretación - de hecho, podrían llegar a confundir al usuario final -. Así que la presentación de la salida se realiza en una tabla que compara los aciertos de tendencia tanto en entrenamiento como en test y coteja los retornos logarítmicos contra los que se podrían haber obtenido con una estrategia *buy and hold*.

5.10. Desarrollo de formularios para interactuar con el usuario

En prácticamente todos los *sprints* he tenido que trabajar con HTML, CSS, Bootstrap y JavaScript. En algunos momentos de forma intensiva para conseguir un resultado visualmente aceptable y entendible.

En la parte más dinámica de interacción con el usuario, los formularios, he realizado un esfuerzo importante en cuanto a manejo de datos y control de errores. Todos los formularios son formularios de *Django*, es decir, están en el lado del servidor y no del cliente. Esto permite, entre otras ventajas, realizar validación automática de usuarios y de modelos asociados a ellos. Además, al utilizar formularios de *Django*, se ha mejorado la gestión de posibles fallos a la hora de introducir los datos necesarios.

La filosofía seguida para el desarrollo de los formularios ha mantenido dos ideas básicas:

- El usuario tiene que saber la información que debe introducir y, a ser posible, con una interfaz amigable.
- El usuario no tiene que poder introducir valores erróneos (intencionada o no intencionadamente).

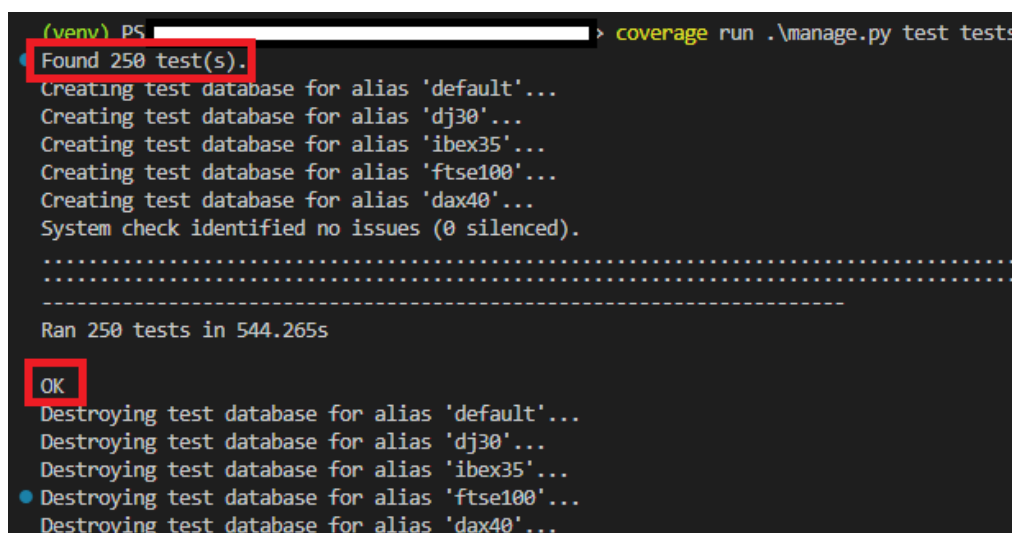
En lo relativo al *DashBoard* hay formularios disponibles para agregar nuevos valores en cartera y para añadir nuevos valores en seguimiento. Y en cuanto al *Lab* hay disponibles formularios para interactuar con los modelos ARIMA y para los algoritmos basados en ML. En todos los casos los métodos de control de errores se han diseñado para que sólo se permitan aquellos

datos estrictamente correctos y se rechacen, informando al usuario, aquellos datos que no cumplan con las restricciones necesarias³¹.

5.11. Testing, log de tests y algunos estadísticos relevantes

Desde el inicio del trabajo tenía claro que una de las facetas a cubrir era el testeo de las aplicaciones de forma ordenada. Cada aplicación, así como las utilidades, tienen sus propios tests por separado. Todas las pruebas se han estructurado de la misma manera en la que están distribuidos los directorios de las aplicaciones. Esto permite llevar un mejor control e ir incrementando las pruebas de manera casi natural.

Hay 250 tests, que dan una cobertura de código casi completa:



```
(venv) PS > coverage run .\manage.py test tests
Found 250 test(s).
Creating test database for alias 'default'...
Creating test database for alias 'dj30'...
Creating test database for alias 'ibex35'...
Creating test database for alias 'ftse100'...
Creating test database for alias 'dax40'...
System check identified no issues (0 silenced).
.....
-----
Ran 250 tests in 544.265s
OK
Destroying test database for alias 'default'...
Destroying test database for alias 'dj30'...
Destroying test database for alias 'ibex35'...
Destroying test database for alias 'ftse100'...
Destroying test database for alias 'dax40'...
```

Figura 5.1: Número de tests realizados. Fuente: elaboración propia

Las comprobaciones se han realizado con **coverage** y es posible encontrar una serie de plantillas HTML con información adicional. Dentro del directorio **htmlcov** se puede encontrar un archivo **index.html** que al ejecutarlo nos abrirá un informe con datos relevante sobre la cobertura de todo el proyecto. Aquí se muestra el resultado final de dicho informe:

³¹Ver los métodos `DashBoard.views._hay_errores()` y `Lab.views._comprobar_formularios()`

Module	statements	missing	excluded	coverage
Total	4094	7	0	100 %

Tabla 5.2: Cobertura de código

Además, se puede comprobar un *log*, que fui realizando para un mejor control y que guarda un comentario de todas aquellas pruebas que se van pasando. La estructura del *log* se controla con un *logger* y se pueden comprobar estos datos en el directorio **log**.

Otras métricas relevantes de este trabajo son³²:

App/Directorio	Archivos	Líneas código	Líneas comentarios
FAT	5	86	135
News	3	139	104
Analysis	6	5013	523
DashBoard	7	521	686
Lab	6	906	743
util	6	428	812
tests	16	2309	307

Tabla 5.3: Número de líneas de código *Python*

App/Directorio	Archivos	Líneas código	Líneas comentarios
News	2	258	23
Analysis	16	449	76
DashBoard	14	698	73
Lab	10	830	77
static/css	273	5	

Tabla 5.4: Número de líneas de lenguajes de marcas y *JavaScript*

³²Información extraída con `pygount` sin contar los archivos estáticos de *Django* ni el directorio del entorno virtual

6. Trabajos relacionados

Publicaciones sobre inversión y control de la diversificación de las carteras hay muchas, pero por el conocimiento que me han aportado tengo que destacar *El inversor inteligente* [39] y uno de los libros de Andrea Redondo, *Inversión: Claves para alcanzar la libertad financiera* [70].

Por otro lado, intentar predecir la evolución de los precios de valores cotizados no parece algo nuevo y, de hecho, en los últimos años se han ido publicando múltiples *papers*, artículos y libros relacionados con algunos de los conceptos tratados en este trabajo. No sólo podemos encontrar artículos sencillos con porciones básicas de código, sino que existe una cantidad importante de estudios serios especialmente relacionados con el *forecasting* de series temporales y con la aplicación de técnicas de *machine learning*. Algunos de los más relevantes se listan a continuación:

- *Machine Learning for Asset Managers* [21]
- *Financial Time Series Forecasting with Deep Learning: A Systematic Literature Review: 2005-2019* [71]. Aquí se pueden encontrar multitud de referencias a trabajos especializados en este campo.
- *Deep Learning for Finance: Deep Portfolios* [52]
- *An Empirical Study on Predicting Stock Prices in the Indian Stock Market Using Machine Learning Techniques* [69]
- *Technical Analysis and Machine Learning: A Systematic Review* [30]

Además de la aplicación de técnicas de *trading* algorítmico, también hay una parte importante de la comunidad analista-inversora que ha dedicado históricamente esfuerzos a aplicar redes neuronales al mundo financiero:

- *Forecasting Stock Prices with Neural Networks* [51]

- *A Survey on LSTM Neural Networks for Time Series Forecasting* [41]

Y otros investigadores han continuado en esa misma línea:

- *Neural Networks for Algorithmic Trading* [46]
- *Deep learning for financial applications: A survey* [73]
- *Deep Reinforcement Learning for Trading* [115]
- *Stock Price Prediction Using Convolutional Neural Networks on a Gridded Time Representation* [11]

También es casi obligatorio mencionar que es frecuente ver publicaciones sobre la programación de *bots* para realizar *trading* de forma automática. Si bien es cierto que no se suele utilizar *Python* para la programación de estos *bots*, sino lenguajes especializados como MQL4 o MQL5, podemos asumir que hay conceptos extrapolables que resultan de interés:

- *Automated Trading with Python: Designing and Developing Automated Trading Systems in Python* [74]
- *Building Machine Learning Powered Trading Bots* [48]

Por último, si utilizamos las webs más populares sobre bolsa e inversión, veremos que empieza a ser habitual que integren algún tipo de sección - normalmente de pago - con predicción automática de precios o análisis de señales de compra-venta. Incluso hay entidades que están ofertando a sus clientes herramientas de formación automática de carteras, lo cual lleva a pensar que este tipo de servicios se están popularizando³³.

³³No parece adecuado convertir este trabajo en un mecanismo publicitario para webs o entidades y por ello no indico referencias; confío en que cualquiera que esté interesado en este mundo conocerá las opciones *pro* de múltiples webs o las herramientas de tipo *roboadvisor*.

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

El análisis técnico de datos de valores cotizados es un asunto especialmente complejo, como ha quedado patente en este trabajo con el uso (y fracaso) de modelos ARIMA para realizar *forecasting* de series temporales. Sin embargo, cada vez se van viendo mayor número de artículos - más o menos acertados - que tratan de avanzar en el campo del *trading* automatizado o que buscan incorporar nuevas técnicas de análisis para favorecer a sus inversiones. Hoy día no es raro encontrar *bots* que hacen operaciones de compra y venta semiautomatizadas o algoritmos que tratan de competir por obtener las mejores rentabilidades. Pero el objetivo de este trabajo no era obtener un muy buen resultado una única vez - que es lo que suele ocurrir en los casos mencionados previamente -, sino tratar de crear herramientas poco comunes que permitan hacer inversiones rentables a largo plazo y que tengan detrás una base de conocimiento justificable, como lo propuesto en el algoritmo del cruce de medias o en las técnicas de *trading* basado en *machine learning*.

De entre todas las webs más famosas sobre bolsa e inversión, en ninguna de ellas he encontrado una herramienta que permita hacer un análisis de rentabilidad-riesgo como el que se propone en este trabajo. Y tampoco está disponible el uso de técnicas basadas en *machine learning* para hacer predicciones de tendencias. Seguramente se deba a que los esfuerzos de las grandes plataformas están concentrados en la presentación agradable de datos en tiempo real y en mantener al inversor entretenido navegando por

las noticias de esos portales; así que en este sentido mi propuesta podría llegar a tener una muy pequeña ventaja competitiva.

En cuanto al desarrollo técnico, cabe destacar que el proceso de aprendizaje de *Django* puede ser complicado, pero a cambio se obtiene un elevado control sobre la información y se dispone de una serie de herramientas ya integradas que facilitan el despliegue en pocos pasos, sobre todo, en lo que concierne a la gestión de las bases de datos.

He de reconocer que este trabajo ha supuesto todo un reto, especialmente por las técnicas utilizadas y por el elevado número de nuevas utilidades que he añadido a mi *caja de herramientas* personal. Hace sólo unos meses no había trabajado con *Django* ni con \LaTeX , tampoco había utilizado *Github actions* ni había generado documentación automática con *Read the docs* y *Sphinx*. Sin duda guardaré buen recuerdo de todas las herramientas utilizadas, porque me han servido para salir de mi zona de confort y fortalecer algunas ideas.

7.2. Líneas de trabajo futuras

Además de los contenidos comentados a lo largo de esta memoria, queda pendiente una mención al uso de redes LSTM[101] para *forecasting* de series temporales. Uno de mis propósitos iniciales era utilizar este tipo de redes para realizar predicciones, pero tras varias implementaciones detecté que los resultados no eran del todo satisfactorios.

En algunas fases del trabajo se desarrolló toda la estructura de formularios y dependencias necesarias para poder trabajar con redes LSTM. Esta parte del código está prácticamente completa - con tests incluidos - pero mi falta de confianza en los resultados obtenidos me llevó a tomar la decisión de no incorporarlo al proyecto final. En la página web no está habilitado este apartado, pero en el repositorio de *Git*Hub se ha dejado disponible para que se vea cómo se podría utilizar, como una herramienta adicional dentro del *Lab*. Es más, aunque la presentación de resultados no está depurada, es posible realizar un *forecasting* con una red que disponga de una capa oculta de cinco neuronas. La dinámica de interacción con el usuario es la misma que en otras aplicaciones de la web y no resultará extraña la interpretación de los datos de salida.

Una de las ventajas de cómo se planteó inicialmente la aplicación del *Lab* es que permitiría ir acoplando nuevas utilidades según evolucionaran los requerimientos de los usuarios o clientes. Es decir, se pueden añadir nuevas

funcionalidades como la de las redes LSTM, pero también, por ejemplo, se podría utilizar *fbprophet*[25] u otras técnicas de *forecasting* que pudieran resultar interesantes.

Por otro lado, hay un conjunto importante de datos que me habría gustado recabar, que son todos los relativos a análisis fundamental[84]. Para la inclusión de este tipo de información también se puede utilizar la API de *yfinance*, aunque requeriría de nuevas bases de datos o, en el mejor de los casos, de una severa adaptación de las existentes. La inclusión de dichos datos podría ofrecer una visión mucho más completa a un posible inversor y facilitaría la toma de decisiones.

Otra característica relevante que hay que señalar es que a lo largo de este trabajo se han utilizado bases de datos *SQLite* por la fácil integración que hay con *Django* pero, seguramente, en un entorno más profesional sería conveniente hacer una migración hacia un sistema gestor de bases de datos como *PostgreSQL*.

Finalmente, no puedo dejar pasar la ocasión de comentar que la página web está alojada en un servidor de *pythonanywhere* y, tal vez, éste no sea el mejor *hosting* posible. Si un día se empezara a tener un elevado número de usuarios es más que probable que el rendimiento cayera considerablemente. Por tanto, en caso de realizar mejoras, sería altamente recomendable ampliar los servicios de alojamiento y habría que comprar un nombre de dominio adecuado.

Bibliografía

- [1] Apache. *Apache Subversion*. <https://subversion.apache.org/>, 2024. Online; Accedido el 20-Abr-2024.
- [2] News API. *News API*. <https://newsapi.org/docs/client-libraries/python>, 2024. Online; Accedido el 22-Abr-2024.
- [3] Ran Aroussi. *yFinance*. <https://github.com/ranaroussi/yfinance>, 2024. Online; Accedido el 31-May-2024.
- [4] Ran Aroussi. *yfinance*. <https://pypi.org/project/yfinance/>, 2024. Online; accedido el 22-Abr-2024.
- [5] Atlassian. *Jira*. <https://www.atlassian.com/es/software/jira>, 2024. Online; Accedido el 22-Abr-2024.
- [6] Atlassian. *Trello*. <https://trello.com/es>, 2024. Online; Accedido el 22-Abr-2024.
- [7] Sphinx Authors. *Sphinx*. <https://www.sphinx-doc.org/en/master/>, 2024. Online; Accedido el 22-Abr-2024.
- [8] Ned Batchelder. *Coverage*. <https://coverage.readthedocs.io/en/7.4.4/>, 2024. Online; Accedido el 22-Abr-2024.
- [9] Bloomberg. *Global media company that covers business, finance, markets, politics and more*. <https://www.bloomberg.com/>, 2024. Online; Accedido el 10-Abr-2024.
- [10] Bootstrap. *Bootstrap*. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>, 2024. Online; Accedido el 22-Abr-2024.

- [11] S. Borovkova and I. Tsiamas. Stock price prediction using convolutional neural networks on a gridded time representation. *Journal of Financial Markets*, 42:197–218, 2019.
- [12] Pascal Brachet. *Texmaker*. <https://www.xmlmath.net/texmaker/>, 2024. Online; Accedido el 22-Abr-2024.
- [13] Jason Bronwlee. *How to Create an ARIMA Model for Time Series Forecasting in Python*. <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>, 2024. Online; Accedido el 01-Jun-2024.
- [14] Marti Castany. *El método walk-forward: la guía definitiva*. <https://quantdemy.com/metodo-walk-forward/>, 2023. Online; Accedido el 23-May-2024.
- [15] Julian Chang. *Modern Portfolio Theory with Python*. <https://medium.com/@changjulian17/modern-portfolio-theory-with-python-f33c9f517cd4>, 2021. Online; Accedido el 31-May-2024.
- [16] CircleCI. *CircleCI*. <https://circleci.com/docs/language-python/>, 2024. Online; Accedido el 22-Abr-2024.
- [17] Mercurial community. *Mercurial*. <https://www.mercurial-scm.org/>, 2024. Online; Accedido el 20-Abr-2024.
- [18] Ian Stapleton Cordasco. *Flake8*. <https://flake8.pycqa.org/en/latest/>, 2024. Online; Accedido el 22-Abr-2024.
- [19] Django Full Course. *Django Full Course*. <https://www.youtube.com/@djangofullcourse1171>, 2023. Online; Accedido el 31-May-2024.
- [20] Banco de España. *Página oficial del Banco de España*. <https://www.bde.es/wbe/es/>, 2023. Online; Accedido el 06-Nov-2023.
- [21] Marcos López de Prado. Machine learning for asset managers. *Quantitative Finance*, 2018.
- [22] Comisión Nacional del Mercado de Valores. *Plan de Educación Financiera 2022-2025*. https://www.cnmv.es/docportal/publicaciones/planeducacion/planeducacionfinanciera_22_25es.pdf, 2022. Online; Accedido el 06-Nov-2023.

- [23] Comisión Nacional del Mercado de Valores. *Página oficial de la Comisión Nacional del Mercado de Valores*. <https://www.cnmv.es/portal/home.aspx>, 2023. Online; Accedido el 06-Nov-2023.
- [24] Django developers. *Django documentation*. <https://docs.djangoproject.com/en/5.0/>, 2024. Online; Accedido el 31-May-2024.
- [25] Mera developers. *Prophet*. <https://facebook.github.io/prophet/>, 2024. Online; Accedido el 01-Jun-2024.
- [26] NetworkX developers. *NetworkX*. <https://networkx.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [27] The Matplotlib development team. *Matplotlib*. <https://matplotlib.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [28] Django. *Django documentation*. <https://docs.djangoproject.com/en/5.0/faq/general/>, 2023. Online; Accedido el 20-Abr-2024.
- [29] Django. *Django multi routing*. <https://docs.djangoproject.com/en/5.0/topics/db/multi-db/#topics-db-multi-db-routing>, 2024. Online; Accedido el 31-May-2024.
- [30] Christian L. Dunis, Apostolos Karathanasopolous, George Sermpinis, Konstantinos Theofilatos, and John Keane. Technical analysis and machine learning: A systematic review. *European Journal of Operational Research*, pages 229–241, 2016.
- [31] Elsevier. *Mendeley*. <https://www.mendeley.com/>, 2024. Online; Accedido el 22-Abr-2024.
- [32] Fazt. *Django, Curso de Django para Principiantes*. <https://www.youtube.com/watch?v=TlintZyhXDU>, 2023. Online; Accedido el 31-May-2024.
- [33] Django Software Foundation. *Django*. <https://www.djangoproject.com/>, 2024. Online; Accedido el 22-Abr-2024.
- [34] Git. *Git*. <https://git-scm.com/>, 2024. Online; Accedido el 20-Abr-2024.
- [35] Gitea. *Gitea*. <https://about.gitea.com/>, 2024. Online; Accedido el 20-Abr-2024.

- [36] GitHub. *GitHub*. <https://github.com/>, 2024. Online; Accedido el 20-Abr-2024.
- [37] GitHub. *GitHub actions*. <https://docs.github.com/en/actions>, 2024. Online; Accedido el 22-Abr-2024.
- [38] GitLab. *GitLab*. <https://about.gitlab.com/>, 2024. Online; Accedido el 20-Abr-2024.
- [39] Benjamin Graham. *El inversor inteligente*. Deusto, Grupo Planeta, 10th edition, September 2007.
- [40] The PostgreSQL Global Development Group. *PostgreSQL*. <https://www.postgresql.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [41] Sepp Hochreiter and Jürgen Schmidhuber. A survey on lstm neural networks for time series forecasting. *Neural Computation*, pages 1735–1780, 1997.
- [42] Spyder IDE. *Spyder IDE*. <https://www.spyder-ide.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [43] Num FOCUS Inc. *Pandas*. <https://pandas.pydata.org/docs/>, 2024. Online; Accedido el 22-Abr-2024.
- [44] Pivit Inc. *Zube*. <https://zube.io/>, 2024. Online; Accedido el 22-Abr-2024.
- [45] Investing. *Website that provides real-time data, news, analysis and tools for global financial markets*. <https://www.investing.com/>, 2024. Online; Accedido el 10-Abr-2024.
- [46] Tomasz Janeczko. Neural networks for algorithmic trading. *MetaQuotes Software Corp.*, 2018.
- [47] Jenkins. *Jenkins*. <https://www.jenkins.io/solutions/python/>, 2024. Online; Accedido el 22-Abr-2024.
- [48] Michael Jones. Building machine learning powered trading bots. *Trading Bot Journal*, pages 55–72, 2021.
- [49] Jonathan Taylor y statsmodels-developers Josef Perktold, Skipper Seabold. *statsmodels*. <https://www.statsmodels.org/stable/index.html>, 2024. Online; Accedido el 22-Abr-2024.

- [50] Dr. Peter Kempthorne. *Lecture 14: Portfolio Theory*. <https://ocw.mit.edu/courses/18-s096-topics-in-mathematics-with-applications-in-finance-fall-2013/resources/lecture-14-portfolio-theory/>, 2021. Online; Accedido el 31-May-2024.
- [51] Takashi Kimoto, Kazuo Asakawa, Masakaz Yoda, and Masakaz Takeoka. Forecasting stock prices with neural networks. *IEEE*, pages 1–7, 1990.
- [52] David Koch, Liang Zhang, and Jürgen Gall. Deep learning for finance: Deep portfolios. *Journal of Financial Data Science*, pages 42–56, 2019.
- [53] Pylint Logilab. *Pylint*. <https://www.pylint.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [54] JGraph Ltd. *Draw.io*. <https://www.drawio.com/>, 2024. Online; Accedido el 22-Abr-2024.
- [55] José Francisco López. *Simulación de Montecarlo*. <https://economipedia.com/definiciones/simulacion-de-montecarlo.html>, 2020. Online; Accedido el 16-May-2024.
- [56] MarketScreener. *Financial information for investors and traders*. <https://www.marketscreener.com/>, 2024. Online; Accedido el 10-Abr-2024.
- [57] Harry Markowitz. *Portfolio selection. Efficient diversification of investments*. Cowles Foundation, 1st edition, 1959.
- [58] Rodrigo Merino. *FAT: Financial Analysis Tool's documentation*. <https://fat.readthedocs.io/es/latest/intro.html#>, 2024. Online; Accedido el 01-Jun-2024.
- [59] Microsoft. *MS Teams*. <https://www.microsoft.com/es-es/microsoft-teams/group-chat-software>, 2024. Online; Accedido el 20-Abr-2024.
- [60] Microsoft. *Visual Studio Code*. <https://code.visualstudio.com/>, 2024. Online; Accedido el 22-Abr-2024.
- [61] microsoft teams. *Integrar GitHub con MS Teams*. <https://github.com/integrations/microsoft-teams>, 2023. Online; Accedido el 20-Abr-2024.

- [62] Vinayak Nishant. *Django MVT Architecture: A Fresh Take on Classic MVC*. <https://www.askpython.com/django/django-mvt-architecture>, 2020. Online; Accedido el 20-Abr-2024.
- [63] Pallets. *Jinja*. <https://jinja.palletsprojects.com/en/3.0.x/>, 2023. Online; Accedido el 31-May-2024.
- [64] Pallets. *Flask*. <https://flask.palletsprojects.com/en/3.0.x/>, 2024. Online; Accedido el 22-Abr-2024.
- [65] pdoc Developers. *pdoc*. <https://pdoc.dev/>, 2024. Online; Accedido el 22-Abr-2024.
- [66] plotly. *plotly*. <https://plotly.com/python/>, 2024. Online; Accedido el 22-Abr-2024.
- [67] pytest-cov contributors. *Pytest-cov*. <https://pytest-cov.readthedocs.io/en/latest/readme.html>, 2024. Online; Accedido el 22-Abr-2024.
- [68] QuantPy. *Efficient Frontier*. <https://www.youtube.com/playlist?list=PLqpCwow11-0ooQGB3vuiCdsRLQ5i-6AEH>, 2021. Online; Accedido el 31-May-2024.
- [69] Vijayalakshmi Ravi and P. A. Vijaya. An empirical study on predicting stock prices in the indian stock market using machine learning techniques. *Procedia Computer Science*, pages 1155–1160, 2019.
- [70] Andrea Redondo. *Inversión: Claves para alcanzar la libertad financiera*. El club inversor, 1st edition, December 2020.
- [71] M. Roondiwala, H. Patel, and K. Varma. Financial time series forecasting with deep learning: A systematic literature review: 2005-2019. *arXiv*, 2019.
- [72] scikit-learn developers. *scikit-learn*. <https://scikit-learn.org/stable/index.html>, 2024. Online; Accedido el 22-Abr-2024.
- [73] O.B. Sezer, M.U. Gudelek, and A.M. Ozbayoglu. Deep learning for financial applications: A survey. *Applied Soft Computing*, 93:106384, 2020.
- [74] John Smith. *Automated trading with python: Designing and developing automated trading systems in python*. *Python Software Foundation*, 2020.

- [75] SQLite. *SQLite*. <https://www.sqlite.org/index.html>, 2024. Online; Accedido el 22-Abr-2024.
- [76] Google Brain Team. *TensorFlow*. <https://www.tensorflow.org/?hl=es>, 2024. Online; Accedido el 22-Abr-2024.
- [77] The Lazy Programmer Team. *Financial Engineering and Artificial Intelligence in Python*. <https://www.udemy.com/course/ai-finance/>, 2024. Online; Accedido el 31-May-2024.
- [78] Read the Docs Inc and contributors. *Read the Docs*. <https://docs.readthedocs.io/en/stable/>, 2024. Online; Accedido el 22-Abr-2024.
- [79] Rodrigo Merino Tovar. *Herramienta web Financial Analysis Tool*. <http://takeiteasy.pythonanywhere.com/>, 2024. Online; Accedido el 10-Abr-2024.
- [80] Rodrigo Merino Tovar. *Repositorio de GitHub del Trabajo de Fin de Grado Financial Analysis Tool*. <https://github.com/rmt0009alu/FAT>, 2024. Online; Accedido el 10-Abr-2024.
- [81] Benito van der Zander. *Textstudio*. <https://www.textstudio.org/>, 2024. Online; Accedido el 22-Abr-2024.
- [82] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [83] Wikipedia. *Programación lineal*. https://es.wikipedia.org/wiki/Programaci%C3%B3n_lineal, 2020. Online; Accedido el 16-May-2024.
- [84] Wikipedia. *Análisis fundamental*. https://es.wikipedia.org/wiki/An%C3%A1lisis_fundamental, 2024. Online; Accedido el 01-Jun-2024.
- [85] Wikipedia. *API*. <https://es.wikipedia.org/wiki/API>, 2024. Online; Accedido el 31-May-2024.

- [86] Wikipedia. *Behavior-driven development*. https://en.wikipedia.org/wiki/Behavior-driven_development, 2024. Online; Accedido el 20-Abr-2024.
- [87] Wikipedia. *Burndown chart*. https://en.wikipedia.org/wiki/Burndown_chart, 2024. Online; Accedido el 31-May-2024.
- [88] Wikipedia. *Buy and hold*. https://en.wikipedia.org/wiki/Buy_and_hold, 2024. Online; Accedido el 24-May-2024.
- [89] Wikipedia. *Camino aleatorio*. https://es.wikipedia.org/wiki/Camino_aleatorio, 2024. Online; Accedido el 18-May-2024.
- [90] Wikipedia. *Capitalización de mercado*. https://es.wikipedia.org/wiki/Capitalizaci%C3%B3n_de_mercado, 2024. Online; Accedido el 24-May-2024.
- [91] Wikipedia. *Covariance and correlation*. https://en.wikipedia.org/wiki/Covariance_and_correlation, 2024. Online; Accedido el 16-May-2024.
- [92] Wikipedia. *CSS*. <https://es.wikipedia.org/wiki/CSS>, 2024. Online; Accedido el 22-Abr-2024.
- [93] Wikipedia. *Desarrollo en cascada*. https://es.wikipedia.org/wiki/Desarrollo_en_cascada, 2024. Online; Accedido el 22-Abr-2024.
- [94] Wikipedia. *Desarrollo guiado por pruebas*. https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas, 2024. Online; Accedido el 20-Abr-2024.
- [95] Wikipedia. *Efficient frontier*. https://en.wikipedia.org/wiki/Efficient_frontier, 2024. Online; Accedido el 17-May-2024.
- [96] Wikipedia. *Homocedasticidad*. <https://es.wikipedia.org/wiki/Homocedasticidad>, 2024. Online; Accedido el 24-May-2024.
- [97] Wikipedia. *HTML*. <https://es.wikipedia.org/wiki/HTML>, 2024. Online; Accedido el 22-Abr-2024.
- [98] Wikipedia. *Issue*. <https://es.wikipedia.org/wiki/Issue>, 2024. Online; Accedido el 31-May-2024.
- [99] Wikipedia. *JavaScript*. <https://es.wikipedia.org/wiki/JavaScript>, 2024. Online; Accedido el 22-Abr-2024.

- [100] Wikipedia. *Kanban (desarrollo)*. [https://es.wikipedia.org/wiki/Kanban_\(desarrollo\)](https://es.wikipedia.org/wiki/Kanban_(desarrollo)), 2024. Online; Accedido el 20-Abr-2024.
- [101] Wikipedia. *Long short-term memory*. https://en.wikipedia.org/wiki/Long_short-term_memory, 2024. Online; Accedido el 31-May-2024.
- [102] Wikipedia. *MiKTeX*. <https://es.wikipedia.org/wiki/MiKTeX>, 2024. Online; Accedido el 22-Abr-2024.
- [103] Wikipedia. *Modelo autorregresivo integrado de media móvil*. https://es.wikipedia.org/wiki/Modelo_autorregresivo_integrado_de_media_m%C3%B3vil, 2024. Online; Accedido el 10-Abr-2024.
- [104] Wikipedia. *Modelo-vista-controlador*. <https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador>, 2024. Online; Accedido el 20-Abr-2024.
- [105] Wikipedia. *Posición corta*. https://es.wikipedia.org/wiki/Posici%C3%B3n_corta, 2024. Online; Accedido el 15-May-2024.
- [106] Wikipedia. *Proceso estacionario*. https://es.wikipedia.org/wiki/Proceso_estacionario, 2024. Online; Accedido el 19-May-2024.
- [107] Wikipedia. *Prueba de Dickey-Fuller aumentada*. https://es.wikipedia.org/wiki/Prueba_de_Dickey-Fuller_aumentada, 2024. Online; Accedido el 18-May-2024.
- [108] Wikipedia. *Ratio de Sharpe*. https://es.wikipedia.org/wiki/Ratio_de_Sharpe, 2024. Online; Accedido el 17-May-2024.
- [109] Wikipedia. *Scrum (desarrollo de software)*. [https://es.wikipedia.org/wiki/Scrum_\(desarrollo_de_software\)](https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)), 2024. Online; Accedido el 20-Abr-2024.
- [110] Wikipedia. *Stationary process*. https://en.wikipedia.org/wiki/Stationary_process, 2024. Online; Accedido el 19-May-2024.
- [111] Wikipedia. *Teoría del portafolio moderna*. https://es.wikipedia.org/wiki/Teor%C3%ADa_del_portafolio_moderna, 2024. Online; Accedido el 15-May-2024.
- [112] Saurabh Kumar y Bertrand Bonnefoy-Claudet. *python-dotenv*. <https://pypi.org/project/python-dotenv/>, 2024. Online; Accedido el 22-Abr-2024.

- [113] François Chollet y desarrolladores de Google. *Keras*. <https://keras.io/>, 2024. Online; Accedido el 22-Abr-2024.
- [114] Kurt McKee y Mark Pilgrim. *Feedparser*. <https://feedparser.readthedocs.io/en/latest/introduction.html>, 2024. Online; Accedido el 22-Abr-2024.
- [115] X. Yang, Y. Li, and P. Deng. Deep reinforcement learning for trading. *Journal of Financial Data Science*, 2(2):25–40, 2020.
- [116] ZenHub. *ZenHub*. <https://www.zenhub.com/>, 2024. Online; Accedido el 22-Abr-2024.



Esta obra está bajo una licencia Creative Commons
Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional
(**CC BY-NC-SA 4.0 DEED**).