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 — 20 de mayo 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, 20 de mayo 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 y fundamental 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 correlación entre las cotizaciones de cuatro bolsas mundiales o la experimentación con predicción automática sobre series temporales, a través de *ARIMA* o redes *LSTM*.

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, noticias y RSS, seguimiento de cartera, *forecasting* de series temporales.

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 graphical correlation analysis between the stock quotes from four world stock exchanges, or experimentation with automatic prediction on time series, through ARIMA or LSTM networks.

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, for local use, at https://github.com/rmt0009alu/FAT.

Keywords

Django, SQLite, financial analysis, news and RSS, investment portfolio, time series *forecasting*.

Índice general

Índice de figuras

Índice de tablas

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 [27] 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 [8, 33, 38], como puede ser el análisis visual de correlaciones entre valores o la comparación gráfica con el sector de referencia, entre otros.

Por otro lado, según el Plan de Educación Financiera 2022-2025 [15] de la CNMV [16] y del Banco de España [14], 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 [68] y de redes LSTM [29].

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.
- *Dataset* de **vídeos de prueba**.

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

- **Página web** del proyecto [52].
- **Repositorio** del proyecto [53].

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 y redes neuronales.

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-Controler*), 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.
- Realizar test unitarios, de integración y de interfaz.
- 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 - así como la correlación entre valores cotizados - y en el análisis (*forecasting*) de series temporales con modelos ARIMA y con redes LSTM.

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)* [73]. 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 - o variación diaria porcentual - de un valor viene dado por:

$$R = P_t/P_{t-1} - 1 = (P_t - P_{t-1})/P_t \tag{3.1}$$

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:

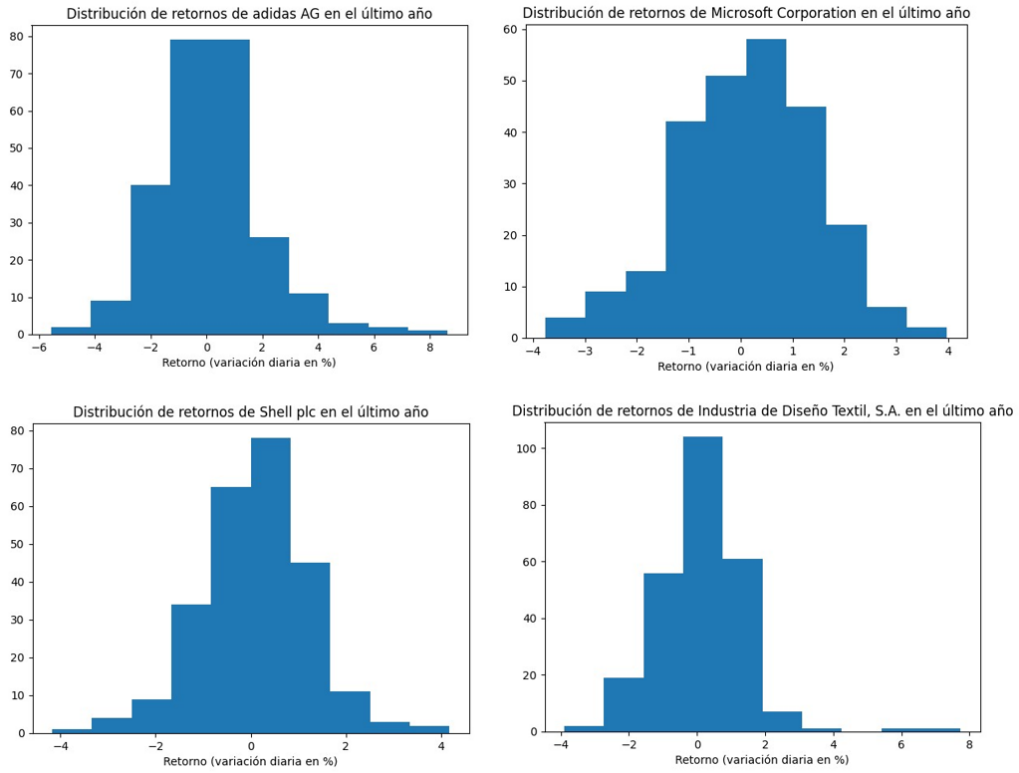


Figura 3.1: Distribuciones de retornos de diferentes valores. Fuente: elaboración propia

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:

$$A : R_1 \sim \mathcal{N}(\mu, \sigma^2) \tag{3.2}$$
$$B : R_2 \sim \mathcal{N}(\mu, \sigma^2) \tag{3.3}$$

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:

$$sd(R_1) = sd(R_2) = \sigma \tag{3.4}$$

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 3.2, 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:

$$Y = 1/2R_1 + 1/2R_2 \tag{3.5}$$

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

$$var(Y) = var(1/2R_1 + 1/2R_2) \tag{3.6}$$

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

$$var(cX) = c^2 var(X) \tag{3.7}$$
$$var(A + B) = var(A) + var(B) \tag{3.8}$$

Y, por tanto:

$$var(1/2R_1 + 1/2R_2) = (1/2)^2\sigma^2 + (1/2)^2\sigma^2 = 1/2\sigma^2 \tag{3.9}$$

Es decir:

$$var(1/2R_1 + 1/2R_2) = 1/2\sigma^2 \rightarrow sd = 1/\sqrt{2}\sigma \tag{3.10}$$

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:

$$w = vector\, de\, longitud\, D$$

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

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

$$\sum_{i=1}^{D} w_i = 1 \tag{3.11}$$

Además, podríamos tener otras restricciones como que los pesos deben ser positivos, lo que limitaría el uso de posiciones cortas [70] 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:

$$R_i = retorno\, del\, valor\, i$$

El retorno medio es el valor esperado de R_i:

$$E(R_i) = \mu_i\ (forma\, vectorial\, de\, todos\, los\, \mu_i\ :\ \mu) \tag{3.12}$$

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

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:

$$E\{(R_i - \mu_i)(R_j - \mu_j)\} = \sum_{ij} (forma\, matricial\ :\ \sum_{DxD}) \tag{3.13}$$

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*):

$$\mu_p = E(R_p) \tag{3.14}$$
$$\sigma_p^2 = var(R_p) \tag{3.15}$$

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

$$R_p = wR_1 + (1-w)R_2 \tag{3.16}$$

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 3.14:

$$E(R_p) = E(wR_1 + (1-w)R_2) = wE(R_1) + (1-w)E(R_2) \tag{3.17}$$
$$\mu_p = w\mu_1 + (1-w)\mu_2 \tag{3.18}$$

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, los más sencillo es sustituir en la definición de varianza por 3.16 y 3.18:

$$\begin{aligned}
\text{var}(R_p) &= E\{(R_p - \mu_p)^2\} \\
&= E\{[wR_1 + (1-w)R_2 - w\mu_1 - (1-w)\mu_2]^2\} \\
&= E\{[w(R_1 - \mu_1) + (1-w)(R_2 - \mu_2)]^2\} \\
&= E\{w^2(R_1 - \mu_1)^2\} + E\{(1-w)^2(R_2 - \mu_2)^2\} \\
&\quad + 2E\{w(1-w)(R_1 - \mu_1)(R_2 - \mu_2)\} \\
&= w^2\text{var}(R_1) + (1-w)^2\text{var}(R_2) + 2w(1-w)\text{cov}(R_1, R_2)
\end{aligned} \tag{3.19}$$

Lo visto es 3.19 se puede escribir en términos de la correlación en lugar de la covarianza, $corr_{12} = \rho_{12} = \sigma_{12}/(\sigma_1\sigma_2)$ [58]:

$$\sigma_p^2 = w_2\sigma_1^2 + (1-w)^2\sigma_2^2 + 2w(1-w)\sigma_{12} \tag{3.20}$$

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

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 3.20 y 3.21 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, $w^T\Sigma w$:

$$\begin{aligned} var(R_p) &= E\{(R_p - \mu_p^2) \\ &= E\{(R^T w - \mu^T w)^2\} \\ &= E(R^T w - \mu^T w)^T (R^T w - \mu^T w) \\ &= E\{w^T (R^T - \mu^T)^T (R^T - \mu^T) w\} \\ &= w^T E\{(R-\mu)(R-\mu)^T\} w \\ &= w^T \Sigma w \end{aligned} \tag{3.22}$$

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:

$$volatilidad\,cartera = volatilidad_p = \sqrt{(var(R_p))} \tag{3.23}$$

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

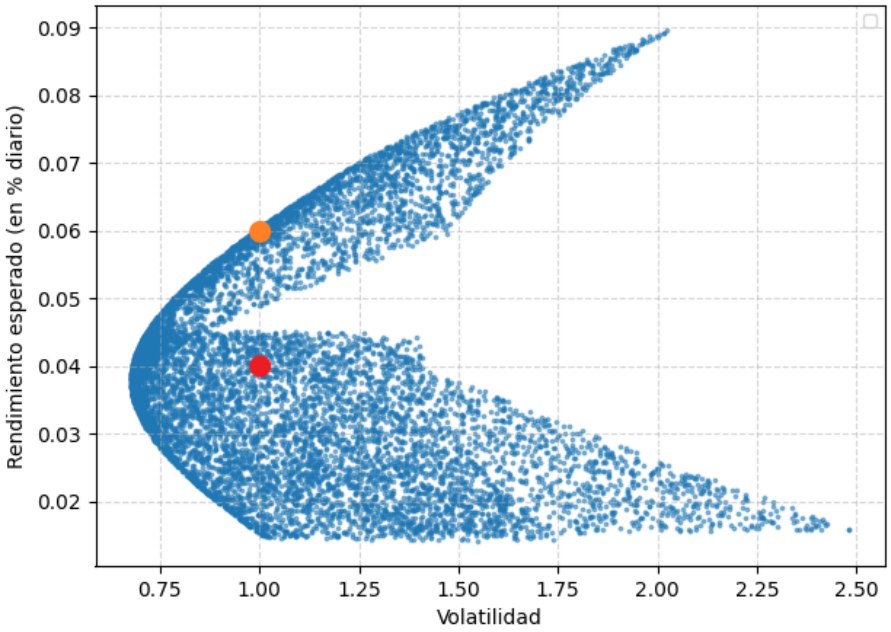


Figura 3.2: Simulación de 10.000 posibles portfolios con los valores *RED.MC*, *EOAN.DE* y *CSCO*, permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024). Fuente: elaboración propia

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:

$$\mu_p = \mu^T w \tag{3.24}$$

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:

$$\begin{aligned} \max_w \mu^T w \\ sujeto\, a : 1_D^T w = 1 \\ w_i \geq 0 \end{aligned} \tag{3.25}$$

Es decir, estamos ante un problema de optimización con restricciones[2] Y, más concretamente, se trata de un problema de programación lineal (LP)[**?**].

De manera similar se puede calcular el retorno mínimo:

$$\begin{aligned} \min_w \mu^T w \\ sujeto\, a : 1_D^T w = 1 \\ w_i \geq 0 \end{aligned} \tag{3.26}$$

En *Python* estas funciones pueden representarse a través de la librería *Scipy*, concretamente, con `scipy.optimize.linprog` [3]

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.

[2]Es 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.

[3]En este proyecto se puede ver cómo se aplica `scipy.optmize.linprog` en `DashBoard.views.py`, en el método `_rendimientos_min_y_max(retornos_df)`.

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 3.22 por comodidad en los cálculos.

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

$$\begin{aligned} min_w w^T \Sigma w \\ sujeto a : \mu^T w = r \\ 1_D^T w = 1 \\ w_i \geq 0 \end{aligned} \tag{3.27}$$

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()` [4]. Hay otras librerías más específicas, pero podemos adecuar *Scipy* para problemas QP.

Frontera eficiente

Una vez hemos conocidos los retornos mínimo y máximo posibles, con la función que queremos optimizar, 3.27, 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 [62] y que Harry Markowitz representó en [39] con su forma parabólica de la siguiente manera:

[4] En 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

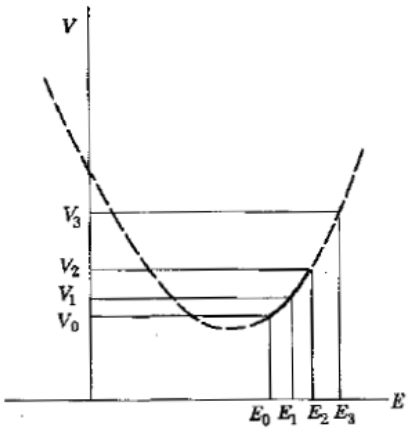


Figure 16. V and E along a critical line.

Figura 3.3: Frontera eficiente. Fuente: [39]

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

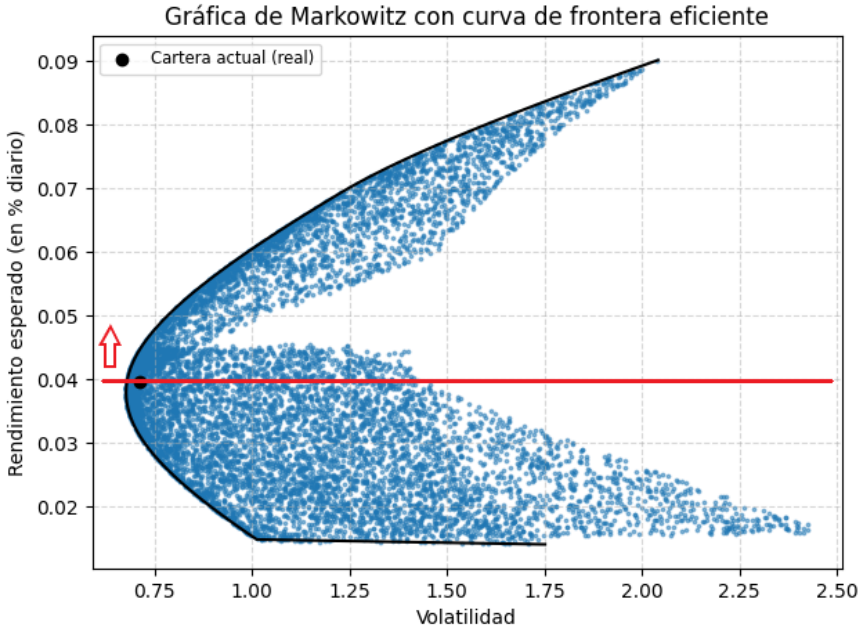


Figura 3.4: Simulación de Montecarlo con frontera eficiente y rentabilidad de cartera con valores *RED.MC*, *EOAN.DE* y *CSCO*, permitiendo posiciones cortas (covarianza calculada con datos de mayo-2023 a mayo-2024) Fuente: Elaboración propia

La parte superior [5] 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.

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* [71]:

$$SR = \frac{E(R_p) - r_f}{\sigma_p} \tag{3.28}$$

Donde r_f representa la tasa libre de riesgo [6], 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* [7] 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

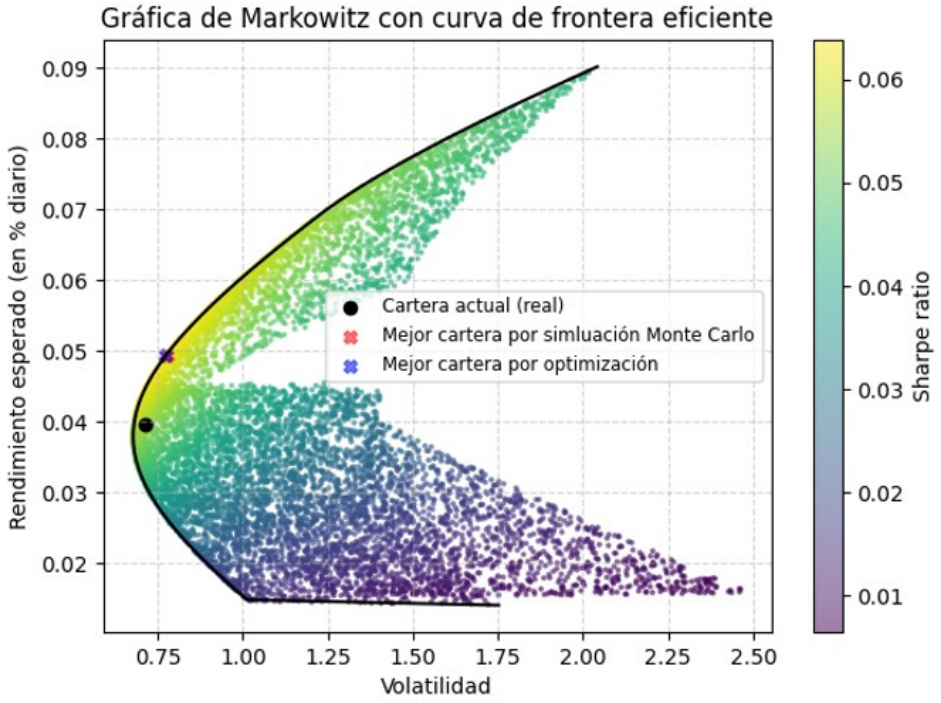
[5] Aunque se obtiene toda la curva, por el propio proceso de optimización, la parte inferior no se puede considerar eficiente porque sólo tenemos que proyectar hacia la parte superior de la misma para ver retornos mejores.

[6] En 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.

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

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:



	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:

$$\hat{y} = mx + b \tag{3.29}$$

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:

$$\hat{y} = w_1 x_1 + w_2 x_2 + b \tag{3.30}$$

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:

$$\hat{y}_t = b + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} \tag{3.31}$$

Y se puede denotar para un momento temporal concreto como:

$$y_t = b + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{3.32}$$

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

$$\epsilon_t = \mathcal{N}(0,\, \sigma^2) \tag{3.33}$$

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:

$$\hat{y}_t = E(y_t) \tag{3.34}$$

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:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} \tag{3.35}$$

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:

$$\begin{aligned} E(y_t) &= E(c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}) \\ &= c \end{aligned} \tag{3.36}$$

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.

$$y_t = b + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t \quad (3.37)$$

Cuando entrenamos un modelo ARMA nos interesa que los datos estén cercanos a la estacionariedad [**?**]. 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:

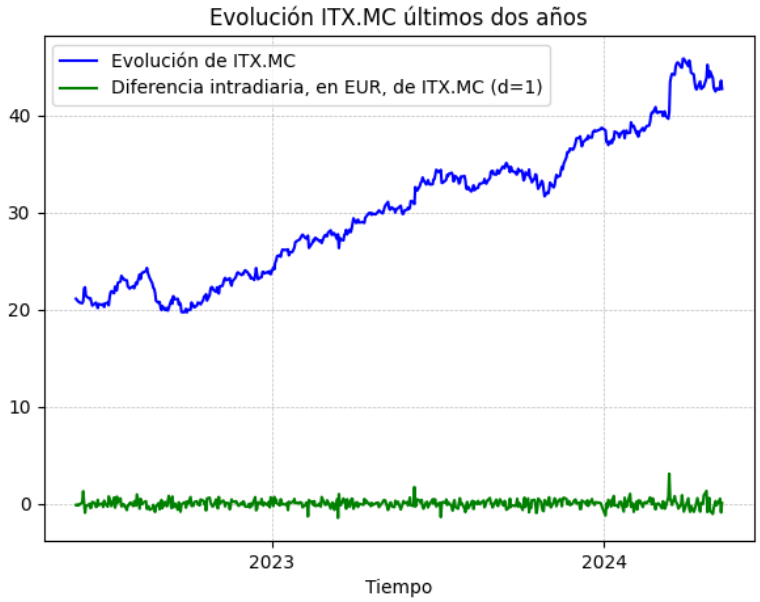


Figura 3.6: Evolución de precios de cierre de *ITX.MC* en los últimos dos años y los mismos datos con diferenciación de orden 1. Fuente: elaboración propia

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:

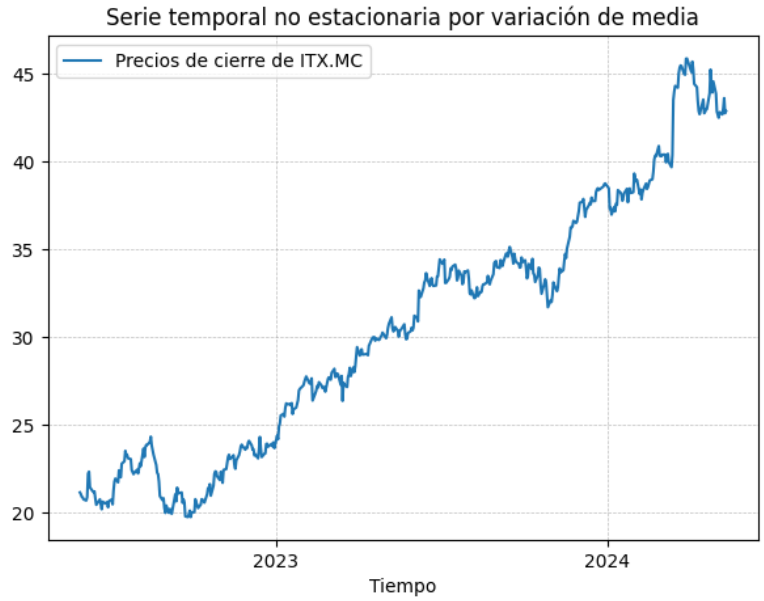


Figura 3.7: Serie no estacionaria por variación de media. Fuente: elaboración propia

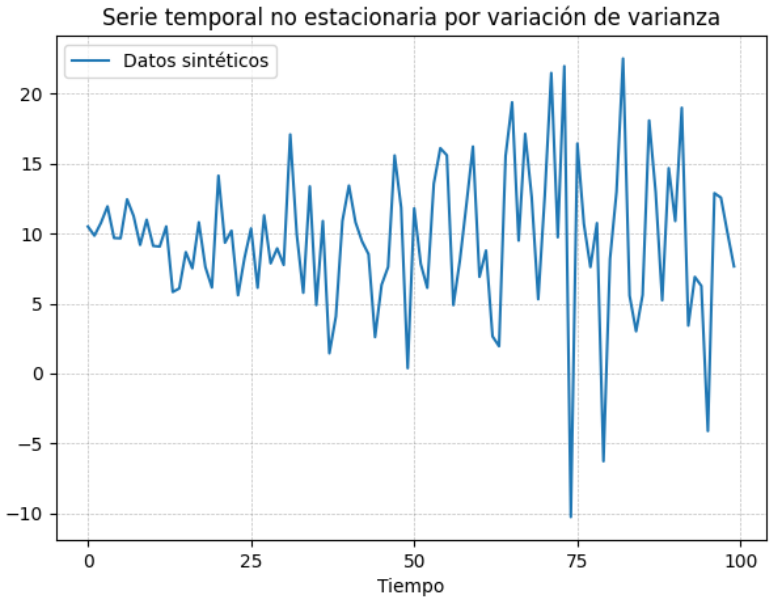


Figura 3.8: Serie no estacionaria por variación de varianza. Fuente: elaboración propia

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) [**?**].

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)[**?**]. Una estacionariedad fuerte significa que la distribución de las variables aleatorias de un proceso estocástico no varia con el tiempo, es decir, no varía en diferentes momentos τ de la serie temporal:

$$F_Y(y_{t_1+\tau}, y_{t_2+\tau}, \ldots, y_{t_n+\tau}) = F_Y(y_{t_1}, y_{t_2}, \ldots, y_{t_n}) \quad \forall \tau, t_1, t_2, \ldots, t_n \quad (3.38)$$

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:

$$\mu Y(t) = \mu_Y(t+\tau) \quad \forall \tau \quad (3.39)$$

$$K_{YY}(t_1, t_2) = K_{YY}(t_1 - t_2, 0) \quad \forall t_1, t_2 \quad (3.40)$$

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.

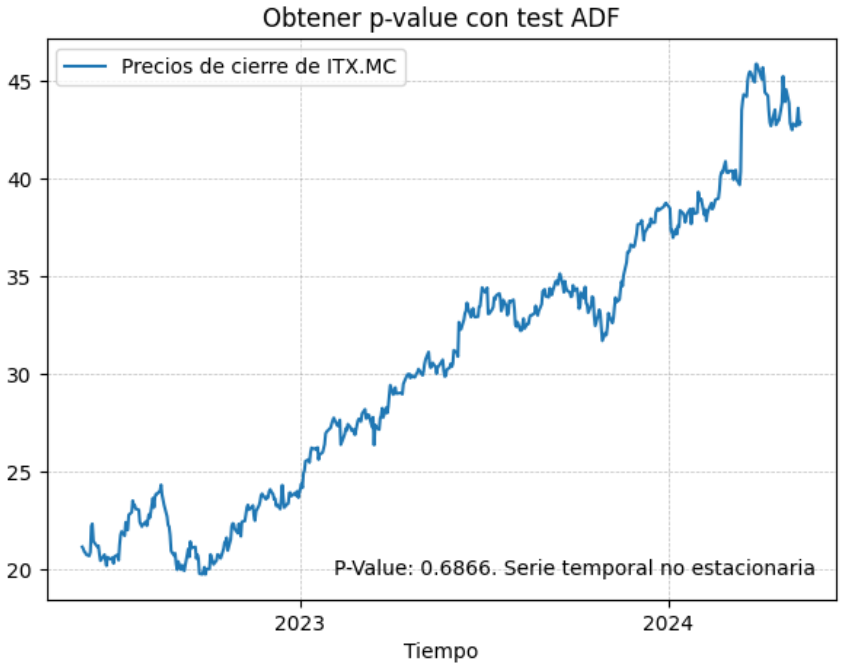


Figura 3.9: Serie no estacionaria, con p-value > 0.5. Fuente: elaboración propia

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 [8], 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.

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

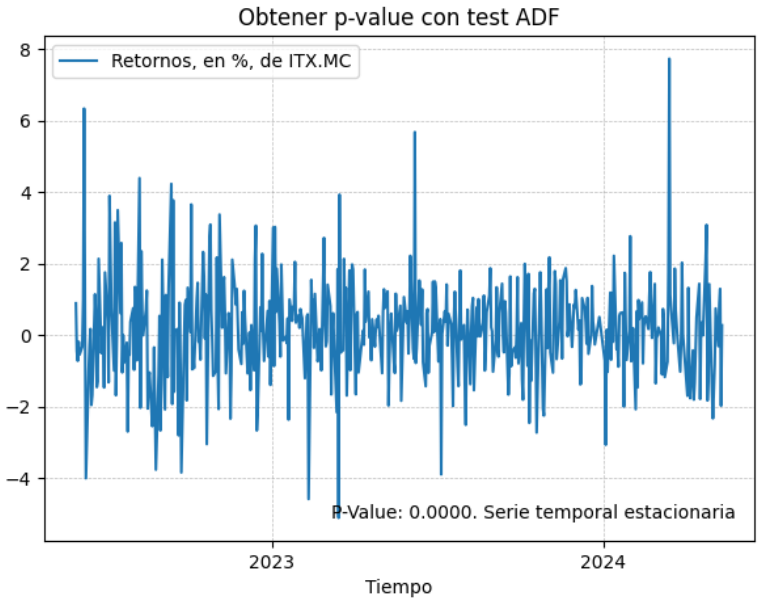


Figura 3.10: Serie estacionaria, con p-value < 0.5. Fuente: elaboración propia

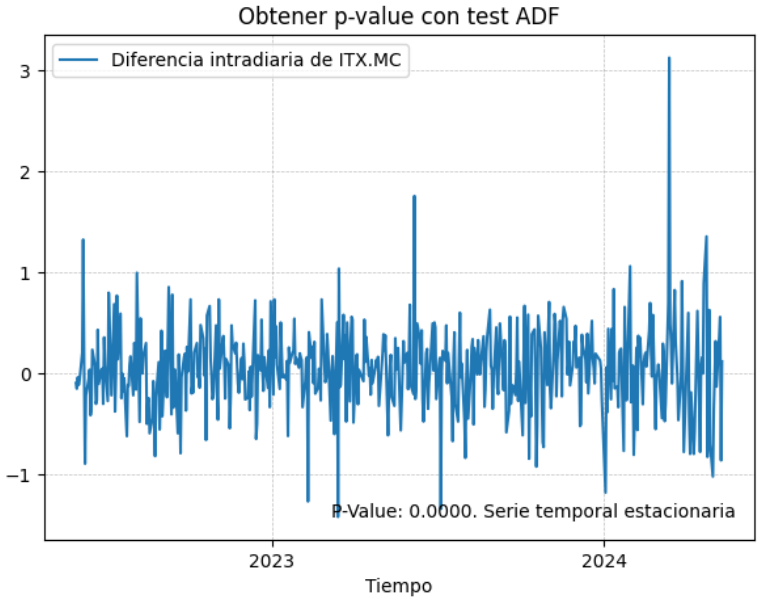


Figura 3.11: Serie estacionaria, con p-value < 0.5. Fuente: elaboración propia

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 p y q a través de funciones que pueden ser útiles para el análisis de los datos.

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

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

$$cov(X, Y) = E\{(X - \mu_X)(Y - \mu_Y)\} \tag{3.41}$$

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

$$autocovarianza = cov(Y_{t_1}, Y_{t_2}) \tag{3.42}$$

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

$$autocorrelation = \frac{cov(Y_{t_1}, Y_{t_2})}{\sigma_Y(t_1)\sigma_Y(t_2)} \tag{3.43}$$

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)[57] en el que lo único que tenemos es un modelo representado por *ruido*. Este modelo tan particular se puede denotar por:

$$\begin{aligned} \Delta y_t &= \epsilon_t \\ y_t - t_{t-1} &= \epsilon_t \\ y_t &= y_{t-1} + \epsilon_t \end{aligned} \tag{3.44}$$

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

$$r_t = p_t - p_{t-1} = \epsilon_t \sim \mathcal{N}(\mu,\, \sigma^2) \tag{3.45}$$

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 -.

4. Técnicas y herramientas

4.1. Técnicas metodológicas

Scrum

Scrum [72] 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* [60] , 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 [61] 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 [56] 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 [66] 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 [19, 43], que es similar al Modelo-Vista-Controlador (MVC) [69]. 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.

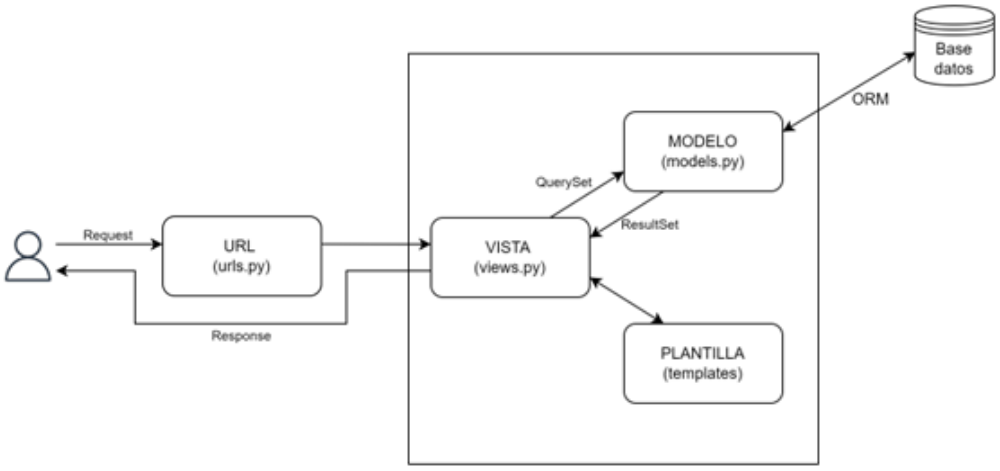


Figura 4.1: Patrón MVT. Fuente: realización propia

4.3. Control de versiones

- Herramientas consideradas: Git [22], Apache Subversion [1] y Mercurial [12].
- 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 [24], GitLab [26] y Gitea [23].
- 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 [32], ZenHub [77], Trello [5] y Jira [4].
- 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 *burndonws* , *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 [40].
- 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 [42] para enviar notificaciones a un grupo de trabajo.

4.7. Entorno de desarrollo integrado (IDE)

- Herramientas consideradas: Spyder IDE [30], Visual Studio Code [41].
- 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 integracción con *GitHub*, entre otros.

4.8. Documentación de la memoria

- Herramientas consideradas: Texmaker [10] y TeXstudio [54].
- 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* [67] es menos problemática que con *TeXstudio*.

4.9. Documentación del código

- Herramientas consideradas: Sphinx [6], Read the Docs [51] y pdoc [45].
- 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.rtfd.io

4.10. Integración y despliegue continuos (CI/CD)

- Herramientas consideradas: GitHub actions [25], Jenkins [34] y CircleCI [11].
- 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 [36] y Flake8 [13].
- 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 analizaor 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 [7] y Pytest-cov [47].
- 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 [21] y Flask [44].
- 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 [49] y PotgreSQL [28].
- 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 [31] 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 [46] 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 [18] 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 [3] es una biblioteca de código abierto para *Python* que permite el acceso y procesado 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 [76] 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 [17] 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 [35] 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 [48] es una bilioteca 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 [55] 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 [75] es una biblioteca de código abierto para aprendizaje profundo escrita en *Python* - en este proyecto se ejecuta sobre *TensorFlow* [50] -. *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 [64] es el lenguaje de marcado de hipertexto estándar para crear páginas web. Es un lenguaje que utiliza etiquetas para definir la esrtructura y el contenido de una página web.

CSS

CSS [59] 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 [9] es un *framewrok* de código abierto para el desarrollo web *front-end*. Proporciona un conjunto de herramientas y componentes pre-definidos que permiten a los desarrolladores crear plantillas e interfaces atractivas y responsivas.

JavaScript

JavaScript [65] 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 [74] 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 [37] 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 [20] es una herramienta de gestión de referecnias 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 LaTeX.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

6. Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

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://pypi.org/project/yfinance/, 2024. Online; Accedido el 22-Abr-2024.

[4] Atlassian. *Jira*. https://www.atlassian.com/es/software/jira, 2024. Online; Accedido el 22-Abr-2024.

[5] Atlassian. *Trello*. https://trello.com/es, 2024. Online; Accedido el 22-Abr-2024.

[6] Sphinx Authors. *Sphinx*. https://www.sphinx-doc.org/en/master/, 2024. Online; Accedido el 22-Abr-2024.

[7] Ned Batchelder. *Coverage*. https://coverage.readthedocs.io/en/7.4.4/, 2024. Online; Accedido el 22-Abr-2024.

[8] Bloomberg. *Global media company that covers business, finance, markets, politics and more*. https://www.bloomberg.com/, 2024. Online; Accedido el 10-Abr-2024.

[9] Bootstrap. *Bootstrap*. https://getbootstrap.com/docs/5.3/getting-started/introduction/, 2024. Online; Accedido el 22-Abr-2024.

[10] Pascal Brachet. *Texmaker*. https://www.xm1math.net/texmaker/, 2024. Online; Accedido el 22-Abr-2024.

[11] CircleCI. *CircleCI*. https://circleci.com/docs/language-python/, 2024. Online; Accedido el 22-Abr-2024.

[12] Mercurial community. *Mercurial*. https://www.mercurial-scm.org/, 2024. Online; Accedido el 20-Abr-2024.

[13] Ian Stapleton Cordasco. *Flake8*. https://flake8.pycqa.org/en/latest/, 2024. Online; Accedido el 22-Abr-2024.

[14] Banco de España. *Página oficial del Banco de España*. https://www.bde.es/wbe/es/, 2023. Online; Accedido el 06-Nov-2023.

[15] 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.

[16] 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.

[17] NetworkX developers. *NetworkX*. https://networkx.org/, 2024. Online; Accedido el 22-Abr-2024.

[18] The Matplotlib development team. *Matplotlib*. https://matplotlib.org/, 2024. Online; Accedido el 22-Abr-2024.

[19] Django. *Django documentation*. https://docs.djangoproject.com/en/5.0/faq/general/, 2023. Online; Accedido el 20-Abr-2024.

[20] Elsevier. *Mendeley*. https://www.mendeley.com/, 2024. Online; Accedido el 22-Abr-2024.

[21] Django Software Foundation. *Django*. https://www.djangoproject.com/, 2024. Online; Accedido el 22-Abr-2024.

[22] Git. *Git*. https://git-scm.com/, 2024. Online; Accedido el 20-Abr-2024.

[23] Gitea. *Gitea*. https://about.gitea.com/, 2024. Online; Accedido el 20-Abr-2024.

[24] GitHub. *GitHub*. https://github.com/, 2024. Online; Accedido el 20-Abr-2024.

[25] GitHub. *GitHub actions*. https://docs.github.com/en/actions, 2024. Online; Accedido el 22-Abr-2024.

[26] GitLab. *GitLab*. https://about.gitlab.com/, 2024. Online; Accedido el 20-Abr-2024.

[27] Benjamin Graham. *El inversor inteligente*. Deusto, Grupo Planeta, 10th edition, 2007.

[28] The PostgreSQL Global Development Group. *PostgreSQL*. https://www.postgresql.org/, 2024. Online; Accedido el 22-Abr-2024.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[30] Spyder IDE. *Spyder IDE*. https://www.spyder-ide.org/, 2024. Online; Accedido el 22-Abr-2024.

[31] Num FOCUS Inc. *Pandas*. https://pandas.pydata.org/docs/, 2024. Online; Accedido el 22-Abr-2024.

[32] Pivit Inc. *Zube*. https://zube.io/, 2024. Online; Accedido el 22-Abr-2024.

[33] 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.

[34] Jenkins. *Jenkins*. https://www.jenkins.io/solutions/python/, 2024. Online; Accedido el 22-Abr-2024.

[35] Jonathan Taylor y statsmodels-developers Josef Perktold, Skipper Seabold. *statsmodels*. https://www.statsmodels.org/stable/index.html, 2024. Online; Accedido el 22-Abr-2024.

[36] Pylint Logilab. *Pylint*. https://www.pylint.org/, 2024. Online; Accedido el 22-Abr-2024.

[37] JGraph Ltd. *Draw.io*. https://www.drawio.com/, 2024. Online; Accedido el 22-Abr-2024.

[38] MarketScreener. *Financial information for investors and traders*. https://www.marketscreener.com/, 2024. Online; Accedido el 10-Abr-2024.

[39] Harry Markowitz. *Portfolio selection. Efficient diversification of investments*. Cowles Foundation, 1st edition, 1959.

[40] Microsoft. *MS Teams*. https://www.microsoft.com/es-es/microsoft-teams/group-chat-software, 2024. Online; Accedido el 20-Abr-2024.

[41] Microsoft. *Visual Studio Code*. https://code.visualstudio.com/, 2024. Online; Accedido el 22-Abr-2024.

[42] microsoft teams. *Integrar GitHub con MS Teams*. https://github.com/integrations/microsoft-teams, 2023. Online; Accedido el 20-Abr-2024.

[43] 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.

[44] Pallets. *Flask*. https://flask.palletsprojects.com/en/3.0.x/, 2024. Online; Accedido el 22-Abr-2024.

[45] pdoc Developers. *pdoc*. https://pdoc.dev/, 2024. Online; Accedido el 22-Abr-2024.

[46] plotly. *plotly*. https://plotly.com/python/, 2024. Online; Accedido el 22-Abr-2024.

[47] pytest-cov contributors. *Pytest-cov*. https://pytest-cov.readthedocs.io/en/latest/readme.html, 2024. Online; Accedido el 22-Abr-2024.

[48] scikit-learn developers. *scikit-learn*. https://scikit-learn.org/stable/index.html, 2024. Online; Accedido el 22-Abr-2024.

[49] SQLite. *SQLite*. https://www.sqlite.org/index.html, 2024. Online; Accedido el 22-Abr-2024.

[50] Google Brain Team. *TensorFlow*. https://www.tensorflow.org/?hl=es, 2024. Online; Accedido el 22-Abr-2024.

[51] Read the Docs Inc and contributors. *Read the Docs*. https://docs.readthedocs.io/en/stable/, 2024. Online; Accedido el 22-Abr-2024.

[52] Rodrigo Merino Tovar. *Herramienta web Financial Analysis Tool*. http://takeiteasy.pythonanywhere.com/, 2024. Online; Accedido el 10-Abr-2024.

[53] 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.

[54] Benito van der Zander. *Texstudio*. https://www.texstudio.org/, 2024. Online; Accedido el 22-Abr-2024.

[55] 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.

[56] Wikipedia. *Behavior-driven development*. https://en.wikipedia.org/wiki/Behavior-driven_development, 2024. Online; Accedido el 20-Abr-2024.

[57] Wikipedia. *Camino aleatorio*. https://es.wikipedia.org/wiki/Camino_aleatorio, 2024. Online; Accedido el 18-May-2024.

[58] Wikipedia. *Covariance and correlation*. https://en.wikipedia.org/wiki/Covariance_and_correlation, 2024. Online; Accedido el 16-May-2024.

[59] Wikipedia. *CSS*. https://es.wikipedia.org/wiki/CSS, 2024. Online; Accedido el 22-Abr-2024.

[60] Wikipedia. *Desarrollo en cascada*. https://es.wikipedia.org/wiki/Desarrollo_en_cascada, 2024. Online; Accedido el 22-Abr-2024.

[61] Wikipedia. *Desarrollo guiado por pruebas*. https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas, 2024. Online; Accedido el 20-Abr-2024.

[62] Wikipedia. *Efficient frontier*. https://en.wikipedia.org/wiki/Efficient_frontier, 2024. Online; Accedido el 17-May-2024.

[63] Wikipedia. *Estacionalidad*. https://es.wikipedia.org/wiki/Estacionalidad, 2024. Online; Accedido el 18-May-2024.

[64] Wikipedia. *HTML*. https://es.wikipedia.org/wiki/HTML, 2024. Online; Accedido el 22-Abr-2024.

[65] Wikipedia. *JavaScript*. https://es.wikipedia.org/wiki/JavaScript, 2024. Online; Accedido el 22-Abr-2024.

[66] Wikipedia. *Kanban (desarrollo)*. https://es.wikipedia.org/wiki/Kanban_(desarrollo), 2024. Online; Accedido el 20-Abr-2024.

[67] Wikipedia. *MiKTeX*. https://es.wikipedia.org/wiki/MiKTeX, 2024. Online; Accedido el 22-Abr-2024.

[68] 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.

[69] Wikipedia. *Modelo–vista–controlador*. https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador, 2024. Online; Accedido el 20-Abr-2024.

[70] Wikipedia. *Posición corta*. https://es.wikipedia.org/wiki/Posici%C3%B3n_corta, 2024. Online; Accedido el 15-May-2024.

[71] Wikipedia. *Ratio de Sharpe*. https://es.wikipedia.org/wiki/Ratio_de_Sharpe, 2024. Online; Accedido el 17-May-2024.

[72] Wikipedia. *Scrum (desarrollo de software)*. https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software), 2024. Online; Accedido el 20-Abr-2024.

[73] Wikipedia. *Teoría del portafolio moderna*. https://es.wikipedia.org/wiki/Teor%C3%ADa_del_portafolio_moderna, 2024. Online; Accedido el 15-May-2024.

[74] Saurabh Kumar y Bertrand Bonnefoy-Claudet. *python-dotenv*. https://pypi.org/project/python-dotenv/, 2024. Online; Accedido el 22-Abr-2024.

[75] François Chollet y desarrolladores de Google. *Keras*. https://keras.io/, 2024. Online; Accedido el 22-Abr-2024.

[76] Kurt McKee y Mark Pilgrim. *Feedparser*. https://feedparser.readthedocs.io/en/latest/introduction.html, 2024. Online; Accedido el 22-Abr-2024.

[77] ZenHub. *ZenHub*. https://www.zenhub.com/, 2024. Online; Accedido el 22-Abr-2024.