IMF Business School

MÁSTER EN DATA SCIENCE
AND BUSINESS ANALYTICS

PREDICCIÓN DE PRECIOS DE BITCOIN USANDO ANALISIS DE SENTIMENTOS DE NOTICAS FINANCIERAS

Autora: Violeta Chamosa Fondevila

- Madrid, 2023 -

ÍNDICE

INDICE DE FIGURAS

1. ABSTRACT

The increasing importance of the cryptocurrency market in the modern world has reached the curiosity of some people to develop an accurate price forecasting model in order to get some insights regarding the price movement. Predicting crypto currency prices is complex because of their highly volatile nature and the dependency on various social, political and economic factors.

Many studies have relied solely on either historical price data or textual data, which has proven to be insufficient. It's been found by recent sentimental analysis studies that the correlation between stock price movement and financial news articles is very strong. The resulting accuracy of these studies is rather low as the amount of data collected has not been sufficient. The accuracy obtained with the use of machine learning models depends to a large extent on the amount of data with which the model is trained.

In this project, we have collected historical price data, along with technical indicators and sentimental analysis scores over the last 5 years, studying the impact of using these new features in the accuracy of the Bitcoin price forecasting model. Traditional time series predictive models as well as machine learning models have been tested on 2 different datasets: the historical price data alone and the data with the new features.

2. INTRODUCCIÓN

En los últimos años han surgido muchas criptomonedas, irrumpiendo en el mercado como nuevos activos digitales, cambiando el paradigma del mercado financiero global. Bitcoin fue la primera en aparecer en 2009 como un sistema de efectivo electrónico. A partir de ese momento, una gran variedad de monedas digitales se introdujeron alcanzando una capitalización de mercado de 2.65 billones de dólares en 2021.

Las criptomonedas tienen un gran potencial de cambiar nuestras vidas como han hecho, por ejemplo, internet, abriendo una ventana de conocimiento compartido y accesible por todos fácilmente, y, también, los teléfonos móviles, permitiendo a las personas estar comunicadas a kilómetros de distancia. En el plano económico, las monedas digitales podrían llegar a sustituir a los bancos centrales. En el caso de Bitcoin, cada moneda es única y está asegurada criptográficamente, por lo que no puede ser hackeada o replicada. Por otro lado, cada transacción es verificada a través de algoritmos descentralizados, distribuidos en nodos alrededor del mundo. Por último, no es necesario intermediarios para su producción y distribución. Estos factores tienen la capacidad de cambiar el funcionamiento de la economía global tal y como la conocemos.

Desde su nacimiento, Bitcoin ha experimentado un crecimiento meteórico, generando gran atención por parte de inversores y agentes financieros. Por otro lado, su alta volatilidad presenta un gran reto en términos de predicción de movimientos del precio de manera precisa. Por ello, la necesidad de desarrollar un modelo de predicción robusto es más que evidente.

Este proyecto, aborda esa necesidad explorando la integración del análisis de sentimientos sobre noticias financieras relacionadas con la criptomoneda, con el pronóstico del precio de la misma, a través de datos de precios históricos e indicadores técnicos. Los artículos acerca de las criptomonedas en los periódicos y las publicaciones en redes sociales han demostrado tener una gran influencia en los mercados de monedas digitales, impulsando rápidas fluctuaciones en el precio. Por ello, entender el impacto de estas en los precios es una tarea esencial.

2.1. Motivación

Bitcoin o, a menudo conocido como el ‘oro digital’, ha atraído tanto inversores a nivel individual como institucional, posicionándolo como activo crítico en las carteras de inversión modernas. Por ende, un modelo de predicción preciso es crucial en la toma de decisiones en el mercado de criptomonedas.

El análisis de sentimientos juega un papel fundamental en el entendimiento de la percepción de los activos financieros. Así, las noticias, publicaciones en redes sociales y debates acerca del mismo, pueden influir en el sentimiento de los agentes financieros y, consecuentemente, afectar al precio de Bitcoin. Este proyecto tiene como objetivo el aprovechamiento de este análisis para mejorar la precisión a la hora de pronosticar los movimientos del precio de Bitcoin.

La motivación del proyecto viene dada por la creciente demanda de herramientas fiables para evaluar el mercado de criptomonedas de manera efectiva. Las monedas digitales presentan retos únicos y diferentes de los activos financieros tradicionales, debido principalmente, a su descentralización, su rápida fluctuación y su susceptibilidad a las noticias. El proyecto está enfocado en proporcionar a los agentes financieros un marco de trabajo mejorado para el pronóstico del precio de Bitcoin.

2.2. Objetivo del Proyecto

2.2.1. Objetivo general

El objetivo principal del trabajo es construir y evaluar un modelo robusto y preciso para predecir el precio de la criptomoneda Bitcoin (BTC), introduciendo el análisis de sentimientos de noticias financieras. El proyecto se enfoca en aprovechar las técnicas de procesamiento del lenguaje natural (NLP) y los algoritmos de aprendizaje automático para analizar la influencia de los artículos periodísticos en el mercado de criptomonedas, y por consiguiente, analizar, por un lado, la influencia del análisis de sentimientos a la hora de pronosticar los precios de Bitcoin y, también, la mejora de la capacidad de los modelos de pronosticar el precio una vez introducido este análisis.

2.2.2. Objetivo específico

- Recolectar el histórico de precios de Bitcoin y artículos relacionados con Bitcoin. Limpiar los datos y preprocesarlos.
- Usar herramientas de obtención de datos como el web scraping o el uso de API.
- Aplicar feature engineering para crear características en relación al sentimiento acerca de la criptomoneda, empleando técnicas de análisis de sentimientos en los artículos recogidos para extraer el sentimiento de los mismos. Esto incluye la polaridad, si es positivo, negativo o neutro, y la subjetividad. También, añadir indicadores técnicos relevantes.
- Desarrollar, entrenar y evaluar modelos de predicción, tanto tradicionales como de aprendizaje automático como, por ejemplo, ARIMA y LSTM.
- Evaluar los modelos usando métricas como el error absoluto medio (MAE), el error cuadrado medio (MSE) y la precisión.
- Comparar y analizar los resultados una vez añadidas las nuevas características.

2.3. Metodología del proyecto

Para el desarrollo de este proyecto, se han seguido una serie de pasos o etapas las cuales me han proporcionado un conocimiento más profundo acerca de técnicas de *feature engineering* y de los modelos *machine learning* y sus diferentes aplicaciones.

La primera de estas fases es la de documentación, en la cual he recolectado una gran cantidad de información a través de no solo documentos escritos, si no también, videográficos. El análisis y prueba de los diferentes procesos llevados a cabo para recolectar los datos, junto con la aplicación de una variedad de modelos predictivos, es la segunda etapa del proyecto. Por último, y una vez obtenidos resultados acordes con la finalidad del proyecto, he empezado el periodo final del proyecto haciendo la redacción del mismo. A continuación, vamos a explicar más en profundidad cada una de las fases.

2.3.1. Documentación

Las herramientas que se han usado en esta etapa son principalmente Google, más en concreto, Google Scholar, y YouTube. Empezando con el uso de YouTube para poder tener una primera idea de cómo se llevan a cabo proyectos de este estilo y qué conocimientos y herramientas son necesarios para su ejecución. Tras ello, y una vez tenido una idea borrador en mente, he procedido a documentarme más en profundidad acerca de los modelos de machine learning usados en casos de predicción de series temporales y, más específicamente, en el mundo de las criptomonedas.

También, y gracias a las lecturas de estos *papers* y revisiones sistemáticas, he encaminado la búsqueda de información hacia el *feature engineering* para poder complementar el *dataset* y darle más información al modelo para el pronóstico. Para ello, he tenido que formarme con conocimiento de inversión.

Por último, he recabado información acerca del análisis de sentimientos de criptomonedas y de cómo se puede hacer uso de noticias y comentarios en plataformas de redes sociales, para obtener un sentimiento de los mismos y ser capaz de clasificarlos para ayudar a la predicción del modelo de machine learning.

Por otro lado, la lectura de otros Trabajos de fin de Máster de alumnos de diferentes universidades y escuelas de negocios, me ha aportado una idea de la estructura de mi proyecto.

2.3.2. Análisis y pruebas

Una vez aclarados los conceptos y seleccionado unas cuantas herramientas y modelos para el desarrollo del proyecto, empieza la fase de prueba de los mismos.

En este apartado he hecho uso de páginas web como Github y Kaggle, entre otras. Estas me han ayudado a ver las diferentes maneras de aplicación y las diferentes librerías a mi disposición para llevar a cabo la escritura del código de los modelos, el cual se ha realizado en Python. Otra de las páginas web usadas es Stackoverflow, como una ayuda para la resolución de problemas como, por ejemplo, cuando el resultado no era el deseado o la aparición de un error inesperado. También, he utilizado la propia documentación de las distintas librerías que he decidido usar.

2.3.3. Redacción del proyecto

Por último, la redacción del proyecto. En esta fase, he revisado de nuevo la documentación previamente leída y los diferentes *notebooks* elaborados en la fase anterior para ir redactando el contenido de esta memoria. También, me he documentado con nuevos *papers* que están, no tan enfocados a la práctica, si no a la parte teórica del proyecto. Y, por supuesto, he usado la documentación aportada por el máster en los diferentes campos en los que este proyecto aplica.

3. ESTRUCTURA DEL PROYECTO

Previamente al desarrollo del proyecto paso a paso, vamos a hacer una pequeña introducción a la estructura seguida respecto al desarrollo del trabajo. La finalidad es poder tener una visión superficial de las distintas técnicas usadas para el desarrollo del modelo de predicción de precios de Bitcoin combinado con el análisis de sentimientos de noticias financieras relacionadas con la criptomoneda. En el siguiente apartado, examinaremos las distintas etapas llevadas a cabo con más detalle.

Recolección de los datos

Por un lado, se recogen los datos históricos de los precios de Bitcoin obtenidos a través de la API (*Application Programming Interface*) de Yahoo Finance. Por otro lado, para el análisis de sentimientos, usando *web scraping* en Coindesk se extraen las noticias referentes a Bitcoin usando Visual Studio Code. Ambos *sets* de datos se guardan en dos archivos separados por comas o CSV para ser procesados posteriormente en Python.

Preproceso de los datos

Se limpian los datos y se procesan valores faltantes o nulos en el dataset de precios anteriormente exportado. También se revisan los distintos artículos recolectados, viendo si hay caracteres extraños o texto faltante. Se homogenizan las tablas de datos para poder trabajar de una manera más limpia y ordenada.

Feature engineering

Se crean nuevas características relevantes a Bitcoin, como son, en este caso, indicadores técnicos como medias móviles o RSI. También, se añaden los resultados del análisis de sentimientos como nuevos rasgos.

Correlación

Se hace un estudio de la correlación entre las nuevas características y el precio de cierre, el cual es nuestra variable dependiente en el modelo. Una vez obtenidos los niveles de explicación de cada variable se guarda un nuevo dataset con las variables relevantes en un archivo CSV el cual se usará para la aplicación de los modelos.

Análisis de sentimientos

Se aplican técnicas de análisis de sentimientos para extraer las métricas del sentimiento relativo a los distintos artículos relacionados con Bitcoin.

Selección del modelo

Se escogen varios modelos para realizar la predicción. Para ello, contamos con modelos tradicionales como modelos de aprendizaje automático.

Entrenamiento del modelo

Tras dividirse los datos en dos grupos: datos de entrenamiento, el cuál es un 90% de los datos, y datos para testear el modelo, siendo el restante 10%, se entrena el modelo con el primer conjunto de datos, afinando los hiperparámetros para una optimización del modelo predictivo.

Fine-tuning

Se refinan los parámetros del modelo en función del resultado obtenido. Se prueban distintos algoritmos para ver si el rendimiento del modelo mejora.

Testear

Se prueban los modelos finales con el grupo de datos de prueba para obtener una estimación del rendimiento de las predicciones del modelo.

Evaluación del modelo

Se evalúa la actuación de cada modelo usando diferentes métricas como, por ejemplo, el error absoluto medio (MAE) y otras métricas usadas en regresión.

Interpretación y conclusiones

Se interpretan los resultados obtenidos y se extraen conclusiones finales en conjunto con posibles estudios futuros. También se añaden las dificultades encontradas en el proyecto y se describen posibles soluciones para futuros proyectos.

4. MARCO TEÓRICO

4.1. Bitcoin

En 2008, Satoshi Nakamo publicó un *white paper* llamado "*Bitcoin: A peer-to-peer electronic cash system*". En él, se expone la teoría y diseño de un sistema de moneda digital que no depende del control de una organización o gobierno, lo que él identifica como el problema principal de las monedas fíat, la necesidad de confiar en el sistema para que funcione.

"The root problem with conventional currencies is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust."

Satoshi Nakamo, 2008.

En vez del uso de instituciones, el funcionamiento de Bitcoin se basa en la tecnología peer-to-peer, de igual a igual, y en la criptografía. Expliquemos estos dos componentes.

Todas las transacciones se registran en servidores alrededor del mundo. Cualquier persona que tenga un ordenador puede establecer uno de estos servidores o nodos. Cada transacción es publicada en la red y compartida por los nodos, creando colecciones de las mismas, conocidos como bloques, los cuales son

añadidos permanentemente a la cadena de bloques o *blockchain*. Básicamente es el Libro Mayor de Bitcoin.

Por otro lado, del mismo modo que las monedas tradicionales son guardadas en carteras, las monedas digitales se almacenan en carteras digitales. Pero, en verdad, no existe nada parecido a bitcoin o una cartera en sí, sino, un acuerdo entre los servidores de la red acerca de quien posee la propiedad de cada moneda, alcanzado criptográficamente a través de los nodos. La criptografía es, básicamente, una práctica de ocultar información, manteniéndola secreta y segura de terceras personas, sustituyendo así la necesidad de una institución central en la que confiar esa seguridad. En palabras de Satoshi Nakamoto:

> "What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party."
>
> Satoshi Nakamo, 2008.

La criptografía detrás de Bitcoin fue diseñada por la Agencia de Seguridad Nacional de los Estados Unidos y está basada en el algoritmo SHA-256. Hackear este algoritmo es imposible ya que hay más claves privadas posibles que átomos en el universo.

El software detrás de Bitcoin fue publicado y lanzado a principios del 2009. Hoy en día, el software es libre por lo que cualquiera puede verlo, usarlo o contribuir en el código del mismo.

4.2. Predicción de los precios de Bitcoin

Bitcoin es la primera y más importante criptomoneda. En el mundo de la tecnología financiera, se han desarrollado numerosos modelos matemáticos para predecir el precio futuro de Bitcoin, los cuales pueden proveer a los inversores conocimiento acerca de las inversiones.

Una de las mayores diferencias entre los valores de mercado y Bitcoin es el tiempo. Mientras que los valores tradicionales suelen operar en un horario establecido y solo durante la semana, las operaciones con Bitcoin están disponibles todo el año a cualquier hora del día. Esto hace que los inversores puedan comprar o vender en cualquier momento, haciendo que el precio fluctúe a horas impredecibles. Por ello, el desarrollo de un modelo de predicción del precio de Bitcoin puede ayudar a los inversores a llevar a cabo decisiones estratégicas sobre sus inversiones de manera más segura.

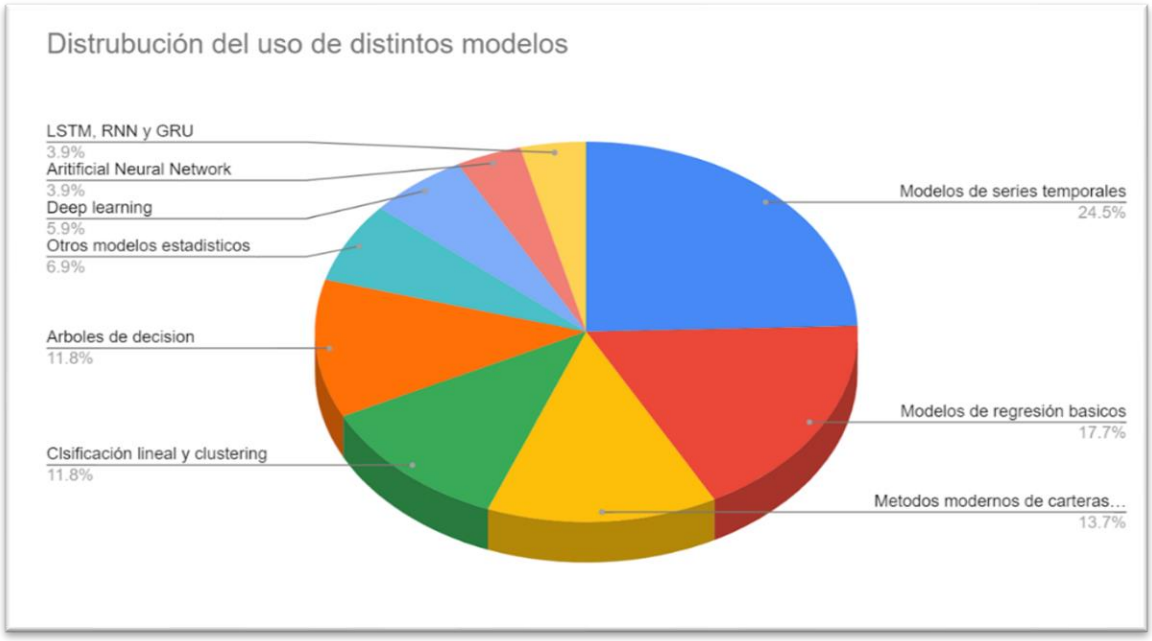
Para abordar el problema de series temporales de los precios de Bitcoin, hay dos principales tipos de estrategias desarrolladas en múltiples trabajos previos: modelos de series temporales tradicionales y modelos de *machine learning*.

En los modelos tradicionales se emplean una combinación de teorías estadísticas y económicas para estimar los precios futuros en función de distintas variables independientes. Esto es conocido como econometría. En estos estudios los investigadores hacen uso de modelos estadísticos lineales como GARCH (*Generalized Autoregressive Conditional Heteroskedasticity*, en español, heterocedasticidad condicional autorregresiva generalizada) o ARIMA (*Autoregressive integrated moving average* o modelo de media móvil integrada autorregresiva) y sus variantes.

Por otro lado, los modelos de *machine learning* son herramientas que permiten inferenciar las relaciones entre los datos que normalmente un humano no es capaz de observar. ML se basa en la definición de dos componentes principales: las características de entrada y la función objetiva. Algunos ejemplos de modelos ampliamente utilizados son random forest, redes neuronales recurrentes (RNN), long short-term memory (LSTM) y gated recurrent units (GRU).

Un análisis de la distribución del uso de los distintos modelos, publicado en febrero del 2022 en un artículo de Financial Innovation, revela que solo el 14% de los *papers* examinados hacen uso de aprendizaje automático, mientras que el otro 86% emplean métodos estadísticos tradicionales, como son regresiones básicas, análisis de series temporales o árboles de decisión, entre otros. En el siguiente gráfico, podemos ver la distribución de los distintos modelos según su uso en los distintos *papers* que se han estudiado.

Figura 1: Distribución del uso de distintas tecnologías en cryptocurrency trading



Fuente: Cryptocurrency trading: a comprehensive survey. Financial Innovation volume 8, Article: 13. 2022

Respecto a los datos más ampliamente usados en los diferentes estudios, podemos dividirlos en 3 grandes grupos: datos sobre el histórico de precios, donde se incluyen el precio de apertura, cierre, máximo, mínimo y volumen de trading; indicadores técnicos basados en los precios y estadísticos técnicos del mercado, como pueden ser el MACD (*moving average convergence/divergence*), el índice de fuerza relativa o RSI o, el OBV (*on-balance volume*) siendo el resultado acumulativo del volumen de trading y, por último, indicadores de sentimientos calculados a través de técnicas de procesamiento del lenguaje natural relativos a noticias y publicaciones en redes sociales.

La predicción de precios de los valores de mercado, especialmente de las criptomonedas, ha ganado mucho interés en los últimos años. Las criptomonedas, específicamente, están siendo aceptadas en el mundo de las finanzas debido a su gran popularidad, lo que las convierte en sujetos de influencia pública. La opinión pública y los precios de las monedas digitales están fuertemente relacionadas. Trabajos recientes en esta materia, muestran que las redes sociales pueden influir cambios en los precios presentándose como indicadores de las tendencia de la economía.

4.3. Aprendizaje automático

El aprendizaje automático o *machine learning* es una rama de la Inteligencia artificial (IA) la cual usa datos y algoritmos tratando de imitar el comportamiento del humano a la hora de aprender, mejorando gradualmente. Los sistemas de inteligencia artificial son usados para llevar a cabo tareas complejas de una manera similar a como los humanos resuelven problemas. Fue definido por primera vez en los años 50 por Arthur Samuel como el campo de estudio que provee a los ordenadores la habilidad de aprender sin haber sido programados para hacerlo.

> *"the field of study that gives computers the ability to learn without explicitly being programmed."*
>
> Arthur Samuel, 1950's

ML es una parte muy importante del campo de la ciencia de datos. A través del uso de métodos estadísticos, se entrenan algoritmos de computación para hacer predicciones y descubrir ideas clave o *key insights* en los datos, ayudándonos a tomar decisiones más precisas. Todo empieza con la recolección de datos, ya sean números, texto, fotos, series de datos temporales, etc. Una vez obtenidos, se preparan para ser usados en el entrenamiento del modelo de aprendizaje automático. Posteriormente, se suministran los datos al modelo computacional para entrenarlo y encontrar patrones o hacer predicciones acerca de los mismos.

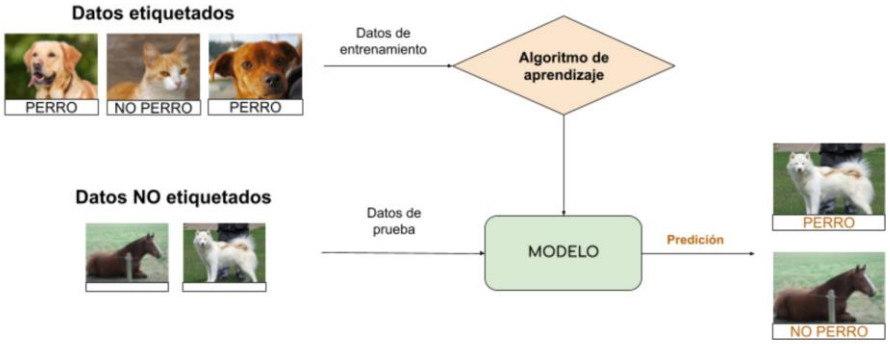
La función de los sistemas de machine learning pueden ser descriptiva, es decir, usa los datos para intentar explicar lo que ha pasado; predictiva, los datos se explotan con el fin de predecir qué va a pasar o prescriptiva, los datos son utilizados para hacer sugerencias acerca de qué acciones tomar.

Existen diferentes tipos de aprendizaje automático dependiendo de la finalidad del proyecto y los datos utilizados para alcanzar el objetivo. Estos son aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. También es posible el uso de modelos híbridos que son combinaciones de los 3 tipos de aprendizaje mencionados anteriormente.

4.3.1. Aprendizaje supervisado

El aprendizaje supervisado es el más usado debido a su sencillez. Estos modelos son entrenados con datos que están etiquetados, es decir, hay un conocimiento a priori de los mismos. En ellos, hay unas variables llamadas características (X) y unas etiquetas (Y). Estos datos son aprendidos por el modelo

en el entrenamiento para, posteriormente, ser capaz de predecir la variable Y o *target.* Por ejemplo, tenemos una serie de fotos de perros y de otros animales, las cuales han sido clasificadas por un humano con las etiquetas de 'perro' y 'no perro'. Estas imágenes son utilizadas para entrenar el modelo, el cual va a aprender patrones existentes en las mismas posibilitando la identificación de fotos de perros por sí solo. Una vez entrenado, le podemos pasar nuevas imágenes. El modelo va a ser capaz de predecir si se trata de un perro o no, gracias al entrenamiento con datos etiquetados anteriormente hecho.



Dependiendo de la variable a predecir, Y, se pueden subclasificar estos modelos en dos tipos: modelos de clasificación o modelos de regresión. La variable a inferir en los modelos de clasificación es cualitativa, es decir, que pertenece a una clase, por ejemplo, perro / no perro. En los de regresión, la *target* es cuantitativa, por ejemplo, predecir ventas futuras.

En este proyecto se va a hacer uso de este tipo de modelo de aprendizaje automático, donde la variable a predecir será el precio de cierre de Bitcoin. Es decir, usaremos modelos de regresión de aprendizaje supervisado.

4.3.2. Aprendizaje no supervisado

En este caso los datos no están etiquetados, es decir, no hay conocimiento a priori. El modelo busca patrones o tendencias en los datos. Por lo tanto, el objetivo no es predecir una variable, si no modelizar la distribución de los datos. Por ejemplo, un modelo de aprendizaje automático no supervisado podría ver en los datos de ventas online un patrón que identifique diferentes tipos de clientes, los cuales no son conocidos a priori.

En este tipo de ML también hay dos subcategorías: clustering y reducción dimensional. El primero de ellos trata de agrupar los datos en conjuntos de similares características, mientras que, el segundo se basa en reducir las variables expresando la misma información.

4.3.3. Aprendizaje por refuerzo

Este modelo es el más complejo. Se entrena a través de prueba y error para encontrar la mejor acción estableciendo un sistema de recompensas. Por ejemplo, se pueden entrenar modelos para jugar a juegos, como el ajedrez, o entrenar coches autónomos para conducir por sí solos.

Adicionalmente, destacar la diferencia entre aprendizaje automático, redes neuronales y aprendizaje profundo o deep learning. Como he mencionado anteriormente, el aprendizaje automático es una rama de la inteligencia artificial. Dentro de él, encontramos las redes neuronales. Estas son algoritmos ampliamente utilizados en machine learning, las cuales son modeladas imitando el funcionamiento del cerebro humano, en donde, cientos o miles de nodos (neuronas) organizados en capas están interconectados creando una red. Por último, deep learning es un tipo de red neuronal con la característica de que posee muchas capas y, por consiguiente, muchas más neuronas o nodos, de ahí lo de profunda.

4.4. Redes neuronales

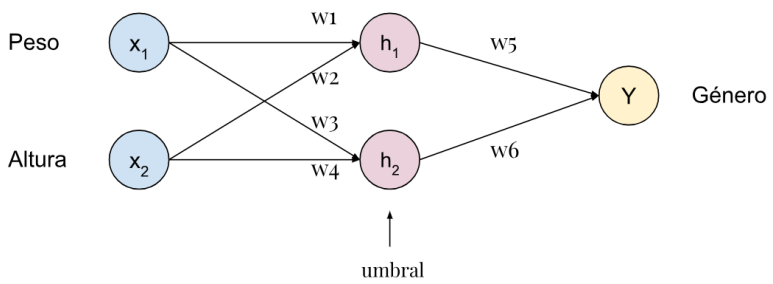
Aunque el término de redes neuronales suena muy moderno, las redes neuronales llevan con nosotros más de 70 años. Fueron introducidas por primera vez en el año 1944 por Warren McCullough y Walter Pitts, dos investigadores de la Universidad de Chicago. La investigación de las redes neuronales fue para la neurociencia y las ciencias de la computación una de las más grandes áreas de investigación hasta los años 70, cuando cayó en desuso, volviendo a resurgir en los años 80 para desaparecer de nuevo hasta los primeros años del nuevo siglo acompañado del aumento del poder de procesamiento de los chips gráficos.

Las redes neuronales son un grupo de algoritmos inspirados en el cerebro humano, diseñados para reconocer patrones a través de la interpretación de datos etiquetados o clusterizados. Lo que el humano

percibe como imágenes, voz o texto, es reconocido por el ordenador a través de números contenidos en vectores.

La configuración de las redes neuronales se basa en cientos, miles o, incluso, millones de nodos simples de procesamiento que están densamente interconectados. La gran mayoría de las redes neuronales actuales están organizadas en capas de nodos donde los datos se mueven entre las neuronas en una sola dirección, conocido por *feed-forward*. Cada nodo está conectado a muchos otros nodos de la capa superior, a través de los cuales recibe información, y otros muchos nodos en la capa inferior, a los cuales les envía los datos. Cada neurona asigna un peso a cada nodo de los que recibe información. Una vez que este nodo recibe datos desde sus nodos previos los multiplicará por el peso que ha asignado a cada uno de ellos, resultando en un único número. Si este valor es inferior o superior a un umbral, el nodo enviará, o no, la información a la siguiente capa. Estos pesos y umbrales iniciales son configurados aleatoriamente y, una vez que se alimenta al modelo con los datos de entrenamiento, estos serán ajustados por el modelo hasta que se obtengan resultados consistentes.

Veamos el siguiente ejemplo gráfico muy simple en el que la red neuronal intenta predecir el género de una persona basándose en su peso y altura. La estructura es la siguiente:



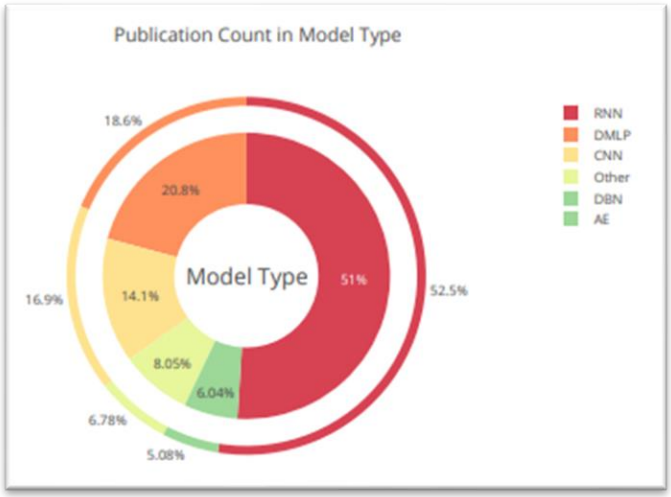
En un principio, los umbrales y los pesos (w_i) son asignados aleatoriamente, una vez que el proceso de entrenamiento empieza, estos valores se van ajustando hasta obtener unos valores finales consistentes con los datos usados para su entrenamiento. Estos datos son datos etiquetados, es decir, cada conjunto de peso + altura tiene un valor establecido para el género, por lo que la Y es conocida. Una vez entrenado el modelo y hallado los valores óptimos de los parámetros de umbral y peso, el modelo está listo para, a través de nuevos datos no etiquetados, predecir la variable Y, en este caso el género.

Hay muchos tipos de redes neuronales como, por ejemplo, las redes neuronales convolucionales, las recurrentes, artificiales… Cada una de ellas es óptima para un tipo de problemas específicos, por

ejemplo, las redes convolucionales son usadas en el campo de reconocimiento de objetos o patrones en imágenes y videos.

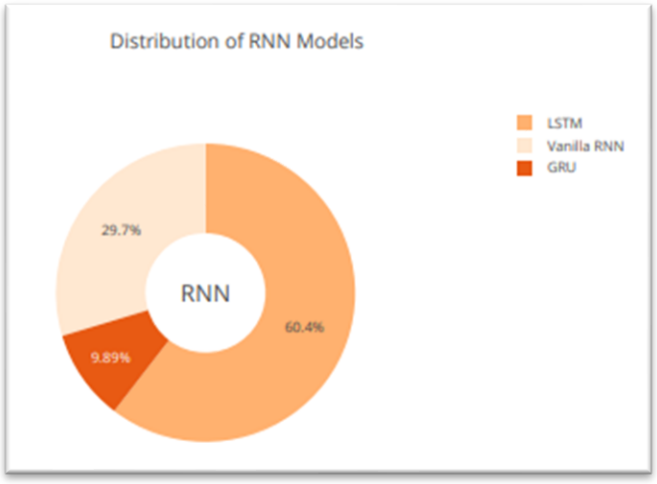
En una revisión sistemática acerca de la predicción de series de datos financieros temporales con el uso del aprendizaje automático, se concluye que, en los estudios revisados más del 50% de los mismos usan modelos de redes neuronales recurrentes, dentro de este grupo se encuentra el modelo LSTM o *Long Short Term Memory*, Vanilla RNN y GRU o *Gated Recurrent Units*.

Figura 2: Gráfico del número de publicaciones por cada tipo de modelo



Fuente: Financial time series forecasting with Deep learning: a systematic literatura review (p. 43). Omer Berat Sezera, M. Ugur Gudeleka y Ahmet Murat Ozbayoglu, 2019.

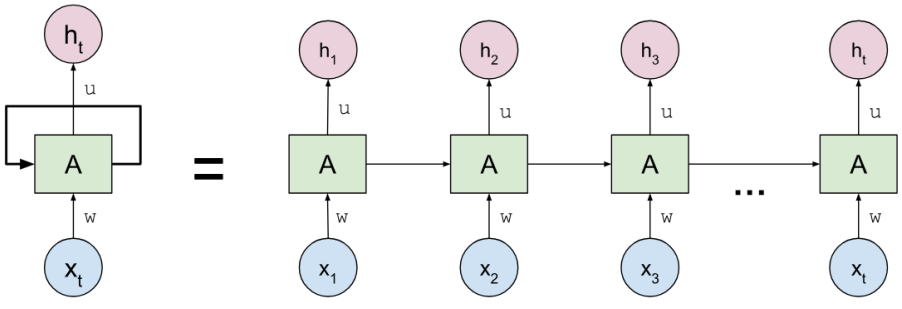
Figura 3: Distribución de los modelos de RNN



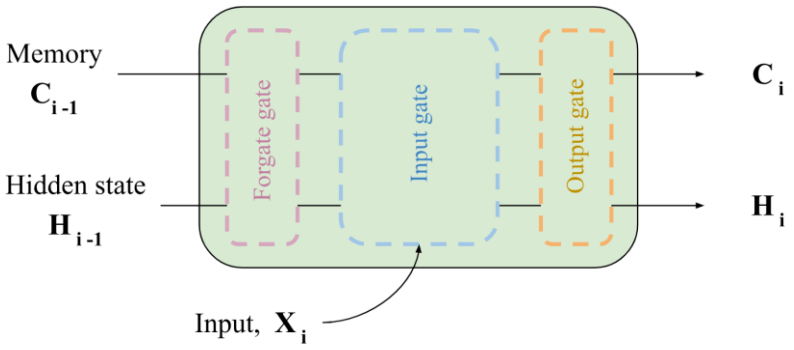
Fuente: Financial time series forecasting with Deep learning: a systematic literatura review (p. 43). Omer Berat Sezera, M. Ugur Gudeleka y Ahmet Murat Ozbayoglu, 2019.

El uso de redes neuronales recurrentes (RNN) son las más adecuadas para nuestro caso ya que están especializadas en procesar secuencias de valores o datos en series temporales. También han revolucionado ámbitos como el procesamiento del lenguaje natural, otra materia de nuestro interés en este proyecto. En nuestro caso, haremos uso del modelo LSTM y GRU.

Las redes neuronales recurrentes (*Recurrent Neural Networks*, RNN) son un tipo de red neuronal artificial caracterizada por el uso de datos secuenciales o datos de series temporales. Utilizan datos de entrenamiento para aprender, pero, a diferencia de otras redes *feed-forward*, estas utilizan la información de nodos previos para influir en la salida del nodo actual, por lo que, el resultado de un nodo es dependiente de los elementos previos a esa secuencia. Esta es una de las características principales de RNN, lo que se conoce por *Hidden State*, el cual recuerda cierta información sobre las secuencias pasadas y usa los mismos parámetros (`u, w`) para cada entrada para producir los distintos inputs.



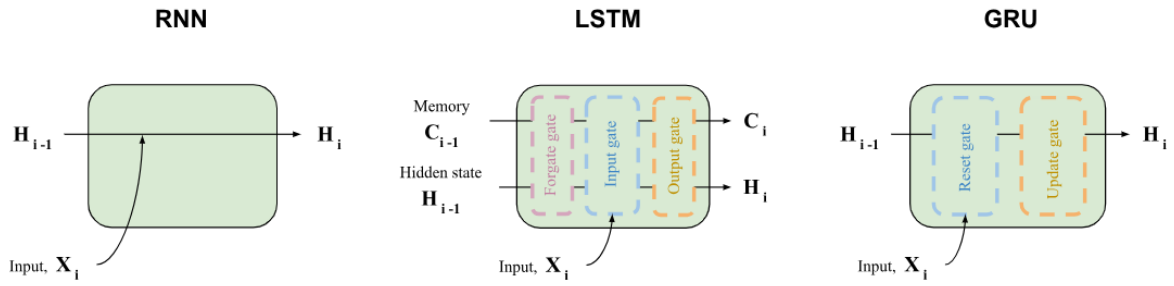
El primero de los modelos que vamos tratar es el LSTM (*Long Short-Term Memory*). Este modelo es una red neuronal recurrente caracterizada por encontrar patrones a largo plazo en secuencias de datos, siendo muy efectiva a lo hora de entender y predecir patrones en datos de tipo series temporales. Incorpora conexiones de retroalimentación, permitiendo procesar secuencias de datos. A diferencia de las células de una red neuronal recurrente simple, el proceso dentro de las mismas es mucho más complejo. Mientras que en los modelos simples la entrada proveniente del nodo anterior pasa por la función de activación (A) para obtener la salida, en las redes LSTM las entradas provienen de 3 estados diferentes: la entrada actual o input (misma entrada que en las redes simples, x_i), la memoria a corto plazo de las células previas (C_{i-1}) y la memoria a largo plazo (*Hidden State*, H_{i-1}). El flujo de datos dentro de cada neurona es controlado a través de 3 gates o puertas: *forget gate*, la cual controla la entrada de información de la memoria a corto y largo plazo; *input gate*, la cual controla la entrada del input y, *output gate*, que controla la salida.



Por otro lado, las redes GRU o *Gated Recurrent Units* son muy parecidas a la red LSTM con pequeñas diferencias. La principal es que las redes GRU usan menos memoria ya que solo usan dos puertas de entrada de información para controlar el flujo de la misma, en vez de 3, como las redes LSTM. Estas gates o puertas son la *reset gate* y la *update gate*. Básicamente, es una simplificación de las redes LSTM, haciéndola más sencilla de entrenar y más rápida de ejecutar.

El modelo GRU se trata de un modelo bastante reciente por lo que no hay estudios suficientes para asegurar cuál de las dos es más conveniente. Lo que sí se sabe es que, si la extensión temporal de los datos es pequeña, es decir, la cantidad de datos es relativamente baja, las redes GRU funcionan mejor que las LSTM.

Veamos a continuación el esquema del funcionamiento de las neuronas en los tres modelos: el modelo simple, el modelo LSTM y el modelo GRU.



4.5. Procesamiento del Lenguaje Natural

El procesamiento del lenguaje natural o NLP (*Natural Language Processing*) es otra de las ramas de la Inteligencia Artificial la cual provee a los ordenadores con la habilidad de entender palabras, tanto en textos como habladas, de una manera similar a los humanos. NLP combina modelos del lenguaje humano basados en reglas con modelos estadísticos y de aprendizaje automático, lo que permite a los ordenadores entender el significado de textos o de la voz.

Hoy en día, la aplicación de NLP está integrada en nuestro día a día. Los asistentes de voz como Alexa o Siri, los *chatbots* en las áreas de servicio al cliente tanto telefónicos como online o, por ejemplo, chat GPT. También es usado en redes sociales para detectar publicaciones no convenientes o en el buscador de Google para mejorar los resultados de las búsquedas.

Como podemos ver, las utilidades del Procesamiento del Lenguaje Natural son muy amplias. En este proyecto vamos a hacer uso de esta herramienta para realizar un análisis de sentimientos sobre noticias relacionadas con Bitcoin. El análisis de sentimientos es el proceso de clasificación de textos según la emoción que refleja, el cual va a ser tratado más en detalle en el siguiente apartado.

4.6. Importancia del análisis de sentimientos

Hay muchos factores que influyen en los precios de los valores de mercado. Uno de ellos es la opinión de los inversores acerca de las noticias financieras y los eventos que suceden en el día a día relacionados con estos mercados. En la actualidad, la disponibilidad de noticias se ha incrementado drásticamente, haciendo muy difícil para los inversores tener una visión clara de la tendencia del mercado. El desarrollo de un modelo que permita recolectar las noticias, relacionadas con el tópico de interés, y el análisis de las mismas resulta de mucha utilidad.

Los analistas del mercado financiero necesitan monitorear y evaluar constantemente las noticias financieras ya que estas son uno de los principales conductores de cambios abruptos en los precios de los valores. Pero, el tiempo necesario para leer y analizar dichas noticias puede ser un factor crucial, pudiendo llegar a costar millones debido a una decisión tardía. Otro factor es la cantidad de noticias disponibles, provenientes de cientos de fuentes diferentes. Resolver estos dos problemas desde una visión tradicional supone o bien reducir la cantidad de artículos a leer o reducir el tiempo de lectura de cada uno de ellos. Una posible solución a esta dicotomía es usar algoritmos computacionales para analizar los artículos empleando técnicas de Procesamiento del Lenguaje Natural (NLP).

La mayoría de trabajos acerca de la predicción del precio de un valor de mercado se han construido basándose solamente en análisis técnicos y fundamentales de los valores, pero en trabajos más recientes, se ha evidenciado una fuerte correlación entre el sentimiento de artículos periodísticos y los movimientos del precio de los valores.

En un estudio relacionado con este tema, llevado a cabo por Joshiand Rao, indica que el uso del análisis de sentimientos de noticias puede ser un predictor de la tendencia del precio de un valor cuando se combina con algoritmos de aprendizaje automático. En este *paper*, se recolectaron noticias de diferentes fuentes y se determinó el sentimiento de los diferentes artículos. Seguido se aplican diferentes algoritmos, incluyendo *Support Vector Machine* (SVM), *Random Forest* y *Naïve Bayes* para predecir la tendencia en el precio del valor de mercado analizado. El uso del análisis de sentimientos incorporado en el estudio mejoró la precisión del modelo, especialmente en el caso del modelo de *random forest*.

El análisis de sentimientos es el proceso del uso del procesamiento del lenguaje natural para identificar y extraer información subjetiva de un texto. Es decir, es capaz de extraer el sentimiento de un texto y clasificarlo en positivo, negativo o neutro. Esta herramienta es usada para analizar actitudes, opiniones

y emociones expresadas en un texto ayudando a mejorar el entendimiento sobre cómo la gente se siente acerca de un tema en concreto. Para llevar a cabo este análisis, se han usado las siguientes técnicas:

4.6.1. TextBlob

Es una librería de Python que incluye el análisis de sentimientos. TextBlob nos proporciona dos salidas principales:

- Polaridad: indica la medida en que un texto es positivo o negativo
- Subjetividad: describe en qué medida el texto es subjetivo o objetivo

TextBlob usa un método basado en reglas, es decir, usa un léxico predefinido (diccionario predefinido agrupado en dos bolsas de palabras: positivas y negativas) y unas condiciones en las que se basa para decidir si un texto es positivo, negativo o neutro. Las noticias, como las publicaciones en redes sociales, pueden llegar a ser bastante informales, haciendo uso del sarcasmo o la ironía. Una de las limitaciones de esta librería es que no es capaz de entender estas informalidades por lo que puede clasificar el texto de manera incorrecta.

4.6.2. VADER (Valence Aware Dictionary for Sentiment Reasoning)

También utiliza un léxico predefinido y unas reglas. La diferencia con textBlob es que usa una combinación de características léxicas etiquetadas para clasificar nuevas palabras según el sentimiento. Por lo que, el sentimiento resultado del texto, es una composición con diferentes pesos según la importancia de cada palabra, normalizado entre -1 y 1, siendo -1 negativo y 1 positivo. Y, a pesar de ser predefinido, el léxico que usa está entrenado en publicaciones de redes sociales y es capaz de entender los emoticonos, lo que lo hace más conveniente a la hora de analizar los sentimientos de un tema en redes sociales.

4.6.3. Fin-BERT

BERT (*Bi-directional Encoder Representation for Transformers*) es uno de los modelos top usado para tareas de procesamiento del lenguaje natural, incluyendo análisis de sentimientos. Fue desarrollada por Google en 2018. Esta librería ha demostrado ser una de las más precisas, debido a que tiene una mejor habilidad a la hora de entender el lenguaje y aprender los diferentes patrones ya que ha sido entrenada

en textos de gran extensión. FinBERT, es el primer modelo de lenguaje pre entrenado específicamente en datos financieros, derivado del modelo BERT. Fue originalmente diseñado para realizar un análisis de sentimientos en textos financieros, más en concreto artículos y noticias financieras. Este modelo clasifica los textos según el sentimiento positivo, negativo o neutro de los mismos.

4.7. Aplicaciones de NLP y machine learning

El uso extensivo de Internet y, más en concreto de redes sociales, hace que en la actualidad la información pueda ser compartida entre usuarios a una gran alta velocidad. También, todos los usuarios son libres de exponer y publicar sus opiniones o investigaciones acerca de cualquier tema, lo que hace que el volumen de noticias y la velocidad a la que las recibimos haya aumentado considerablemente. La disponibilidad de información es abundante. Con ello, muchos de los trabajos que antes eran necesarios ya no lo son, debido a lo que se conoce como la era del DIY o hazlo tú mismo, que gracias al uso de nuevas herramientas y el acceso a la información son plausibles de hacer por uno mismo. Un ejemplo de esto se puede ver en el área de los analistas financieros. El campo de los analistas financieros está llevándose a cabo por los propios inversores, prestando atención a los múltiples foros de inversión, periódicos online y redes sociales en busca de nuevas pistas sobre el movimiento futuro de los valores. La evidencia empírica sugiere que el sentimiento de los inversores puede ser un factor crítico a la hora de explicar las fluctuaciones en los precios de los activos.

Hay varios estudios que combinan datos en formato texto provenientes de redes sociales y datos de los precios históricos del valor en concreto a analizar. La gran mayoría se focalizan en analizar el movimiento de forma binaria, es decir, como una tarea de clasificación (subida o bajada en el precio). Algunos otros, presentan modelos de predicción basados solamente en el análisis de series temporales, los cuales usan las noticias como única fuente de análisis del sentimiento de los inversores.

En este proyecto, se propone un modelo híbrido para salvar estas limitaciones. En él, modelos de análisis del mercado de valores y predicción de precios a través del aprendizaje automático se combinan con el análisis de sentimientos.

4.8. Métricas de evaluación

Las métricas de evaluación a la hora de valorar un modelo de predicción son muy importantes, ya que son las que nos van a orientar a la hora de decidir en qué medida un modelo se ajusta a la realidad.

Evaluar la cantidad de datos con los que se está entrenando el modelo también es importante ya que esto puede conducirnos a dos fenómenos no deseados. El primero de ellos es lo que se conoce por *overfitting*, que significa que hay datos que no son relevantes para el modelo, provocando una disminución de la capacidad predictiva del modelo. Por otro lado, existe lo contrario, llamado *underfitting*, cuando no hay características relevantes suficientes para el aprendizaje del modelo. Una elección adecuada de las características o variables independientes es imprescindible para evitar los dos fenómenos mencionados anteriormente.

El proceso de verificación de los modelos es, básicamente, medir el grado de diferencia de los datos reales y las predicciones obtenidas con el modelo. Por lo general, después de entrenar el modelo, se obtienen las métricas sobre el conjunto de prueba para evaluar el rendimiento del modelo. Existen diferentes tipos de métricas dependiendo del tipo de problema que nos planteemos. A continuación, vamos a explicar los tipos de métricas existentes, haciendo hincapié en aquellas que se van a utilizar en este proyecto.

4.8.1. Métricas de clasificación binaria

En este caso, las métricas evalúan el grado de coincidencia del valor real con el valor predicho. Estas son usadas en modelos de clasificación, es decir, aquellos que predicen una categoría. Existen un gran número de métricas para medir el grado en que las predicciones obtenidas por los modelos y los datos reales se ajustan entre ellos. Las métricas más usadas son la exactitud, la precisión, la tasa F1 o *F-score*, entre otras.

4.8.2. Métricas de clasificación múltiple

La clasificación múltiple difiere de la binaria en que el número de clases es superior a dos, es decir, hay 3 o más clases. En estos casos, el modelo intenta predecir a qué clase de las existentes pertenece cada

caso. Las métricas utilizadas en este caso son las mismas que las de clasificación binaria. También existen métricas específicas a esta categoría como la pérdida logarítmica o entropía cruzada.

4.8.3. Métricas de regresión

El objetivo de los modelos de regresión es predecir una variable dependiente, en nuestro caso el precio de cierre de Bitcoin, en función de otras variables llamadas independientes. Cuanto más se aproxime la previsión con el valor real, mejor será el modelo. Estas métricas son usadas en modelos de regresión, es decir, los que predicen variables cuantitativas. Para ello, se usan diferentes métricas que veremos a continuación. Por lo general, estas métricas utilizadas se miden individualmente en cada caso, para luego, hacer una media de las mismas.

Media del error absoluto (MAE)

Se hace una media entre las diferencias de los valores predichos y los valores reales.

Media del error cuadrático (MSE)

Es la media de los errores cuadráticos, es decir, la media del cuadrado de la diferencia entre el valor predicho y el valor real.

Raíz cuadrada de la media del error cuadrático (RMSE)

Como el nombre indica viene siendo la raíz cuadrada de la media del error cuadrático o MSE, anteriormente mencionada.

Media del valor absoluto de los errores relativos (MAPE)

Es la media de los valores absolutos de los errores relativos. El error relativo es la diferencia del valor real con el pronosticado dividido por el valor real, es decir, es el porcentaje de desviación del valor predicho y real.

4.9. Herramientas usadas

El lenguaje de programación usado para el desarrollo del proyecto ha sido Python. Es uno de los lenguajes más populares y, en el mundo de la ciencia de datos, el más utilizado. Fue creado en los años 90 por Guido Van Rossum, más en concreto, fue publicado en el año 1991.

Python es un lenguaje orientado a objetos, con una sintaxis simple y elegante. Es un software libre donde los usuarios pueden contribuir en el código y en la documentación, permitiendo corregir errores y añadir nuevas características, por ejemplo, a través de librerías. Es decir, es un código hecho por y para la comunidad.

Las características principales son:

- Programa orientado a objetos (OOP) a través del uso de objetos y clases
- Tipado dinámico: no es necesario asignar el tipo de objeto (string, entero…)
- Lenguaje interpretado a través del script que ejecuta el intérprete de Python
- Válido en diferentes plataformas, ya sea Windows, Linux, Mac OS…, siempre que cuente con un intérprete de Python.

La elección de Python se debe a su gran utilidad en el mundo de la ciencia de datos y la inteligencia artificial posibilitando el uso de potentes librerías como son Pandas, Numpy, Keras, etc.

Hemos hablado acerca de la necesidad de un intérprete de Python. En este caso, se ha escogido el entorno de programación Jupyter Notebook ya que es muy sencillo de usar y fácilmente visualizable debido a su estructura. En él, podemos incluir comentarios sobre el código y mostrar los resultados obtenidos de forma interactiva, como si de una libreta de notas se tratase.

Como se ha mencionado el uso de librerías simplifica el código y proporcionan funciones muy útiles sin la necesidad de tener que definirlas uno mismo. En este trabajo, he hecho uso de librerías como Pandas y Numpy (manejo de datos), Matplotlib y Seaborn (visualización gráfica), Sklearn, Tensorflow y keras (aprendizaje automático), entre otras. A continuación, vamos a introducir las diferentes librerías utilizadas, para posteriormente, en el apartado del desarrollo del proyecto, explicar las funcionalidades de cada una aplicadas en el mismo en más detalle.

Pandas

Pandas proviene del término econométrico de panel data o agrupaciones de datos. Empezó a ser desarrollado por Wes McKinney en 2007 y fue lanzado al público en 2008.

Es una librería creada para la manipulación y análisis de datos tabulares en Dataframes. Algunas de las funciones principales son la carga y guardado de datos en formatos como CSV o JSON, la manipulación de datos como uniones entre tablas o tratamiento de datos duplicados o faltantes y, filtrado de datos, por mencionar algunas.

Numpy

Numpy fue creado en 2005 por Travis Oliphant, como una mejora de la librería Numeric. De hecho, el nombre de numpy proviene de Numeric Python.

Es una herramienta *open-source* para la manipulación y limpieza de datos multidimensionales, en concreto, arrays y matrices, que son el equivalente a las listas en Python, pero su procesamiento es mucho más rápido, por ello, es muy usada por los científicos de datos. Entre las funciones incluidas se encuentran las operaciones lógicas y matemáticas de alto nivel entre objetos, generadores de números aleatorios, entre muchas otras.

Matplotlib

Es una librería sencilla de utilizar enfocada en la visualización de datos en Python. Con ella podemos representar datos con pocas líneas de código creando gráficos estáticos, animados o, incluso, interactivos. Permite la modificación del estilo, fuente, ejes…También es posible exportar dichas visualizaciones en ficheros individuales.

Seaborn

Seaborn proporciona una API sobre Matplotlib para la visualización de datos. Las principales funcionalidades de esta librería son, por ejemplo, la de encontrar relaciones entre variables en un grupo de datos, la comparación de distintas distribuciones, entre otras.

Tensorflow

TensorFlow fue desarrollado por Google y lanzado al público en 2015. Después de crear DistBelief en 2011, Google decidió asignar un grupo de científicos de la computación a su mejora, con el propósito de mejorar la velocidad y la exactitud de los pronósticos. Fue así como nació TensorFlow.

Es una librería dedicada al aprendizaje automático y la inteligencia artificial. Puede llevar a cabo numerosas tareas como la construcción, el entrenamiento y la inferencia de las redes neuronales, como tareas de procesamiento del lenguaje natural ya que posee la característica del reconocimiento de texto, imágenes y habla.

Keras

Nació como parte del proyecto ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*) dirigido por un ingeniero de Google, François Chollet. Fue publicada, también, en el año 2015.

Keras es una librería dedicada al aprendizaje profundo y las redes neuronales. Usa las funcionalidades de otras librerías como Tensorflow con el objetivo de implementar redes neuronales de forma más sencilla y modulable. Contiene numerosas implementaciones relacionadas con el desarrollo de redes neuronales como el manejo de capas, objetivos, funciones de activación (como relu o sigma), optimizadores (Adam)…

Scikit-learn

Esta librería empezó como un proyecto de código de verano de Google en Francia. El nombre proviene de Scipy toolkit, scikit, ya que fue diseñada como una extensión de la librería Scipy. Este código fue revisado por un grupo de desarrolladores, los cuales publicaron lo que hoy se conoce como Scikit-learn en 2010.

Scikit-learn es una librería para el aprendizaje automático, incluyendo algoritmos de clasificación, clusterización, regresión y reducción dimensional. Es una librería fácil de usar y se centra únicamente en los modelos. Por lo que, para llevar a cabo tareas de manipulación o visualización de datos, Scikit-learn se puede integrar fácilmente con otras librerías como Pandas o Matplotlib.

pmdarina

Es una librería estadística diseñada para el análisis de series temporales. Su interfaz es muy parecida a la de scikit-learn. Pmdarina trajo lo que en R se conoce como auto.arima. Esta librería generaliza todos los modelos de ARIMA en una clase única y busca los mejores parámetros en cada caso.

Xgboost

Extreme gradient boosting o XGBoost es una librería que nos proporciona un algoritmo de alto rendimiento, el cual usa modelos ensamblados, es decir, utiliza algoritmos simples secuencialmente, así, va aprendiendo de los errores de los modelos anteriores.

Math

Es un módulo que proporciona funciones matemáticas comunes, como trigonometría.

Textgblob

Es una librería para el procesamiento de datos en formato texto. Proporciona un API simple para tareas de procesamiento del lenguaje natural (NLP) como el análisis de sentimientos, clasificación de textos o traductor.

VaderSentiment

Vader o *valence aware dictionary and sentiment reasoner* es una librería dedicada al análisis de sentimientos. Proporciona un modelo simple de léxico basado en reglas usado en tareas de análisis de sentimiento en textos.

Por otro lado, también he hecho uso de la herramienta de Visual Studio Code o VS Code. Es un editor de código desarrollado Microsoft y publicado en 2015. Este editor puede ser usado en diferentes lenguajes, como, en nuestro caso, Python.

En este programa se ha llevado a cabo la extracción de noticias relacionadas con la criptomoneda Bitcoin de la página web Coindesk en un Dataframe, el cual se ha guardado en un documento de tipo CSV para su posterior uso en el análisis de sentimientos. El proceso será explicado más detalladamente

en el apartado del desarrollo del proyecto, donde, como ya hemos indicado, haremos un análisis profundo de las distintas herramientas usadas.

5. DESARROLLO DEL MODELO DE PREDICCIÓN

En este apartado de la memoria, vamos a explicar más en detalle los pasos seguidos para desarrollar el proyecto, los cuales han sido comentados brevemente en el apartado tercero de la misma. Para ello, se van a proveer partes del código, así como tablas de datos y gráficas obtenidas durante su desarrollo, para facilitar el entendimiento del proceso seguido a la hora de desenvolver el modelo de predicción de precios de Bitcoin complementado con el análisis de sentimientos de noticias financieras relacionadas con la criptomoneda.

El proceso que se va a explicar a continuación podría ser válido para el estudio del movimiento del precio de otros activos financieros en la bolsa, simplemente habría que cambiar el nombre del ticker o valor de interés en el apartado de la extracción de los datos.

5.1. Proceso de recolección de los datos

El proceso de recolección de los datos se divide en dos categorías. La primera de ellas, la extracción de los datos de tipo de series temporales, en este caso, el histórico de precios de Bitcoin, desarrollada en Jupiter Notebooks haciendo uso de una API pública y, para la extracción de datos en forma de texto, es decir, las noticias financieras relativas a Bitcoin, la segunda de estas categorías, se ha usado la plataforma Visual Studio Code y la técnica conocida por web scraping.

5.1.1. Time series data

El primer dataframe está compuesto por 17 columnas y 3205 filas. Las columnas son los históricos de precios y algunos indicadores técnicos calculados a partir de los datos de los precios.

Para obtener este dataframe se han usado las librerías de *pandas* para la carga y manipulación del dataset; *pandas_ta* para el cálculo de indicadores técnicos de momentum, volumen, volatilidad y tendencia; *datetime* para la manipulación de la columna de las fechas y, por último y más importante, *yfinance*, a través de la cual se han extraído los datos.

Yahoo Finance o yfinance es una librería gratuita desarrollada por Ran Aroussi que nos permite el acceso a los datos financieros disponibles en la página web de Yahoo Finance. En otras palabras, es una API no oficial que nos da acceso de una manera sencilla a todos los datos de la plataforma de Yahoo Finance.

Las APIs son interfaces de programación de aplicaciones ofrecidas por diferentes proveedores de servicios que permiten que sistemas externos interactúen entre ellos. Es decir, es una forma en que dos programas se comuniquen entre ellos compartiendo datos o funcionalidades.

Con el uso de esta API obtenemos un primer conjunto de datos del histórico de precios de Bitcoin desde el 17 de septiembre de 2014 hasta la fecha actual. Programamos la columna de las fechas como índice de la tabla y lo guardamos para usos posteriores.

Date	Open	High	Low	Close	Volume
2014-09-17	465.864014	468.174011	452.421997	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	36863600
2014-09-21	408.084991	412.425995	393.181000	398.821014	26580100
...	...	...	...	...	...
2023-10-06	27412.123047	28252.537109	27215.552734	27946.597656	13492391599
2023-10-07	27946.781250	28028.091797	27870.423828	27968.839844	6553044316
2023-10-08	27971.677734	28102.169922	27740.662109	27935.089844	7916875290
2023-10-09	27934.472656	27989.470703	27302.562500	27583.677734	12007668568
2023-10-10	27599.025391	27715.847656	27410.812500	27431.419922	9899147264

3311 rows × 5 columns

El siguiente paso es el cálculo de indicadores técnicos, llevado a cabo con la librería de pandas_ta. En este apartado se añaden 12 nuevas columnas al dataset anterior. Este proceso es conocido por *feature engineering* el cual explicaremos en más detalle en el siguiente punto.

Una vez añadidas estas nuevas columnas obtenemos nuestro dataframe en crudo el cual procesaremos en el siguiente apartado.

Date	Open	High	Low	Close	Volume	SMA_14	EMA_26	EMA_12	
2015-01-01	320.434998	320.434998	314.002991	314.248993	8036550	321.108355	331.283191	320.549943	-
2015-01-02	314.079010	315.838989	313.565002	315.032013	7860650	320.907571	330.079400	319.701031	-
2015-01-03	314.846008	315.149994	281.082001	281.082001	33054400	317.416571	326.449963	313.759642	-
2015-01-04	281.145996	287.230011	257.612000	264.195007	55629100	313.370287	321.838485	306.134313	-
2015-01-05	265.084015	278.341003	265.084015	274.473999	43962800	309.269431	318.330005	301.263496	-
...	...	...	...	...	...	...	...	...	
2023-10-06	27412.123047	28252.537109	27215.552734	27946.597656	13492391599	27050.876116	26993.947216	27303.725579	3
2023-10-07	27946.781250	28028.091797	27870.423828	27968.839844	6553044316	27150.122489	27066.161485	27406.050850	3
2023-10-08	27971.677734	28102.169922	27740.662109	27935.089844	7916875290	27269.998465	27130.526549	27487.441465	3
2023-10-09	27934.472656	27989.470703	27302.562500	27583.677734	12007668568	27361.798270	27164.093303	27502.247045	3
2023-10-10	27599.025391	27715.847656	27370.615234	27558.812500	10336539648	27457.624163	27193.331762	27510.949422	

3205 rows × 17 columns

5.1.2. Text data

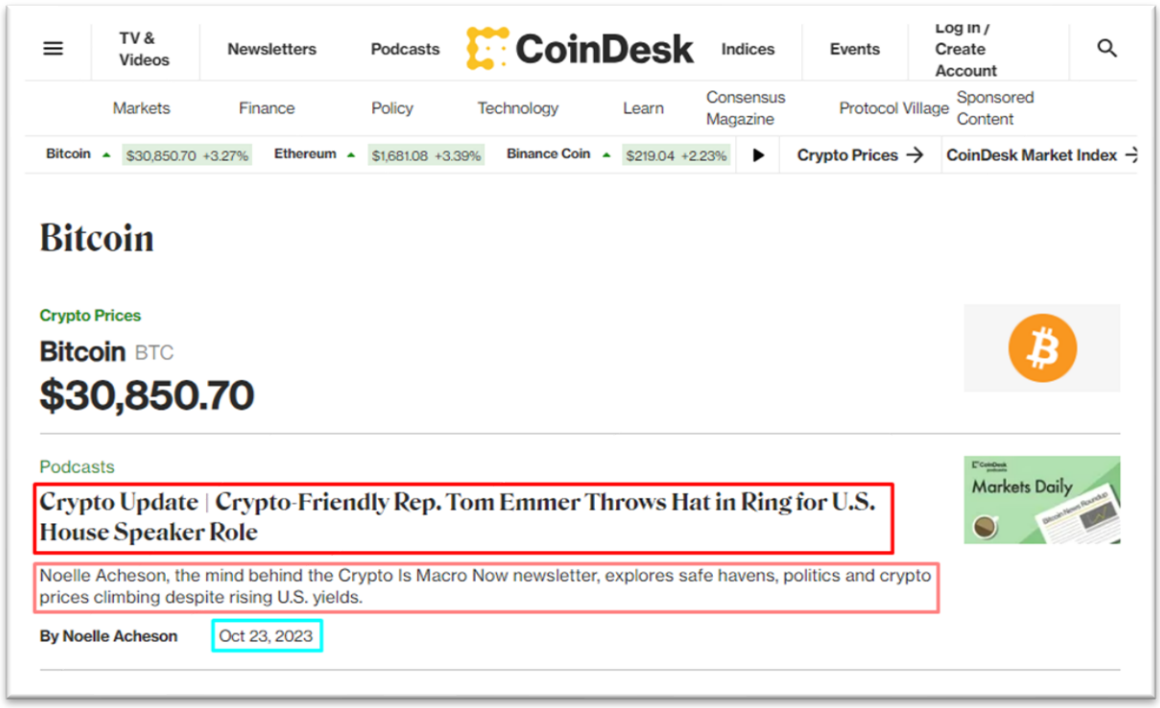
Para realizar un análisis de sentimientos sobre noticias financieras, necesitamos un dataset con los artículos organizados por fechas. Para ello, hemos usado la técnica de *web scraping* en la página web de Coindesk. El código ha sido escrito en la aplicación de Visual Studio Code. La librería que hemos usado es la de *scrapy* que es de uso libre y muy sencilla de utilizar.

El *web scraping* es el acceso automático a los datos dentro de una página web para transformarlos en conjuntos de datos con el fin de poder hacer un uso analítico de los mismos. Requiere de programación específica ya que depende del formato en que la página web esté escrita.

Como vemos en la siguiente imagen, se han obtenido los campos encuadrados de cada artículo de la citada página web: en rojo tenemos el título del artículo, en rosa, el subtítulo y en azul, la fecha en la

que ha sido publicado. Estos campos se han extraído a un archivo CSV para poder trabajar con él en el análisis de sentimientos.

Figura 4: Partes extraídas de los artículos de CoinDesk



Fuente: Pagina web CoinDesk, noticias sobre Bitcoin. (https://www.coindesk.com/tag/bitcoin/1/)

Una vez exportados los datos a un archivo CSV, los cargamos en Jupiter Notebooks para procesarlos previamente a su uso en el análisis de sentimientos. Programamos las fechas como índices y obtenemos el dataframe en bruto. Esta tabla cuenta con 2 columnas, el título y subtítulo del artículo, las cuales agregamos en una sola columna llamada artículos y 1826 filas, correspondientes a los artículos publicados desde finales de febrero de 2018 hasta finales de febrero de 2023. Veamos el dataset a continuación.

Date	articles
2018-02-25	Original Pizza Day Purchaser Does It Again W...
2018-02-26	Bitcoin Pizza Day 2 How A Lightning Payment...
2018-02-27	Rapper 50 Cent Who Bragged About Owning Bit...
2018-02-28	This Is Who Controls Bitcoin British Man ...
2018-03-01	Bitcoin makes inroads in LA s residential re...
...	...
2023-02-20	Bitcoin regains 25K amid hope record China ...
2023-02-21	Bitcoin active addresses concern analyst d...
2023-02-22	Bitcoin Ethereum Technical Analysis BTC Fa...
2023-02-23	Bitcoin bears attempt to pin BTC price under...
2023-02-24	Bitcoin on chain data highlights key similar...

1826 rows × 1 columns

Tras recolectar los datos necesarios podemos pasar al siguiente paso, donde analizaremos los datos y los prepararemos para su posterior modelización, usando diferentes técnicas.

5.2. Métodos y técnicas empleadas

Una vez obtenidos los datos en bruto, vamos a tratarlos para poder trabajar con ellos. En esta sección, vamos a hablar acerca de los procesos del preprocesamiento de los datos en donde, con el uso de la librería pandas, manipulamos los dataframes con el objetivo de obtener unos datos claros y limpios con los que poder trabajar; el *feature engineering*, llevado a cabo en ambos datasets, tanto en el de precios como el de los artículos financieros; y el estudio de las importancia de las variables independientes con la dependiente, en este caso, el precio de cierre diario de Bitcoin.

5.2.1. Preproceso de datos

Una vez recolectados los datos necesarios, es imprescindible hacer una limpieza y preprocesado de los mismos para obtener unos datos de calidad. Este factor es determinante a lo hora de modelar y desarrollar un modelo que nos aporte predicciones precisas y fiables. El preprocesamiento de datos se

entiende como la transformación de los datos en bruto, incluyendo la estandarización de formatos, tratamiento de valores nulos y valores atípicos, entre otras.

En primer lugar, vamos a tratar los datos de series temporales, es decir, los precios históricos. Para ello, vamos a ver si los datos extraídos de Yahoo Finance con ayuda de la API contienen valores faltantes o nulos. Como podemos observar en la siguiente imagen no se obtiene ningún valor nulo.

```
1 data.isnull().sum()
```

```
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

Este dataset no solo contiene los históricos de los precios, si no, como hemos explicado en el punto 5.1, hemos incluido indicadores técnicos calculados a través de los históricos (este proceso es explicado en el siguiente apartado, *feature engineering*). Por lo que, una vez añadidas dichas características, veamos si hay valores nulos en el conjunto final.

```
1 data.isnull().sum()
```

```
Open       0
High       0
Low        0
Close      0
Volume     0
SMA_14    13
EMA_26     0
EMA_12     0
MACD       0
MACDh      0
MACDs     25
CCI        0
ADX        0
RSI        0
ATR        0
OBV        0
CMF        0
dtype: int64
```

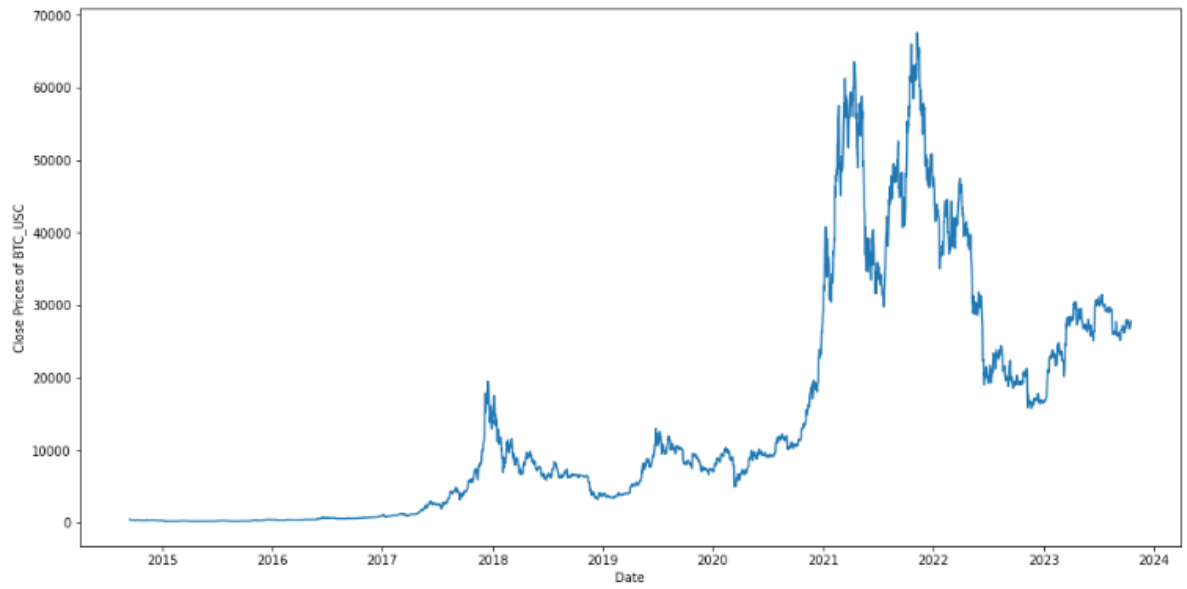
Si nos fijamos en la imagen anterior, podemos ver que hay dos columnas que contienen valores nulos. Haciendo un análisis de dichas columnas podemos ver que se tratan de indicadores calculados a través de medias móviles en n periodos, por lo que, hasta que se obtengan n entradas el cálculo no puede llevarse a cabo. Fijémonos en el primero de ellos la media móvil simple de 14 periodos. En ella podemos ver 13 valores nulos, es decir, hasta tener 14 datos de entrada, la media móvil no puede ser calculada resultando en un valor nulo. Pero, sabiendo como los indicadores han sido calculados, podemos intuir

que tendría que haber más valores nulos, ya que no solo son dos los indicadores que usan medias móviles para su cálculo. Haciendo una impresión de la cabecera del dataframe podemos ver que efectivamente hay más valores nulos los cuales son etiquetados con 'False', como se observa en la siguiente imagen.

SMA_14	EMA_26	EMA_12	MACD	MACDh	MACDs	CCI	ADX	RSI	ATR
NaN	457.334015	457.334015	False	False	NaN	False	False	False	False
NaN	454.897421	452.273398	False	False	NaN	False	False	False	False
NaN	450.445463	443.430719	False	False	NaN	False	False	False	False
NaN	447.368317	438.118915	False	False	NaN	False	False	False	False
NaN	443.772221	432.073084	False	False	NaN	False	False	False	False

Los datos recolectados son desde el 17 de septiembre de 2014, por lo que, para solucionar este problema de una manera simple y sin afectar la calidad de los datos, vamos a eliminar todos los datos previos a 2015, que como vemos en el siguiente gráfico, su eliminación no afectará al modelo.

```
1 fig = plt.subplots(figsize=(16,8))
2 data['Close'].plot()
3 plt.ylabel("Close Prices of BTC_USC")
4 plt.show()
```



Una vez llevada a cabo dicha limpieza analizamos los resultados obtenidos en busca de valores nulos. Observado que no hay existencia de valores nulos hagamos un análisis de la repercusión de la eliminación de los datos del año 2014. En la siguiente figura podemos ver los descriptivos de los datos en bruto, con los valores nulos, en la parte superior y, en la inferior, se muestran los descriptivos una vez eliminados. Se observa que la diferencia entre ambos conjuntos de datos es despreciable.

```
1 data.describe()
```

	Open	High	Low	Close	Volume	SMA_14	EMA_26	EMA_12
count	3317.000000	3317.000000	3317.000000	3317.000000	3.317000e+03	3304.000000	3317.000000	3317.000000
mean	14036.009370	14367.564562	13673.390784	14042.919757	1.644849e+10	14043.696990	13942.348495	13998.476852
std	15960.783671	16350.433659	15516.416618	15957.734805	1.925557e+10	15921.882226	15823.786165	15899.492603
min	176.897003	211.731003	171.509995	178.102997	5.914570e+06	222.873070	230.699121	224.309373
25%	818.142029	834.281006	799.405029	821.797974	1.389800e+08	867.027068	799.249811	833.185520
50%	8039.069824	8218.359375	7837.610840	8043.951172	1.068064e+10	8105.696673	7960.593034	8021.243142
75%	22428.322266	22974.001953	21708.050781	22435.513672	2.705580e+10	22482.250488	22138.778086	22249.030977
max	67549.734375	68789.625000	66382.062500	67566.828125	3.509679e+11	63983.060826	62406.255181	64182.504548

```
1 df.describe()
```

	Open	High	Low	Close	Volume	SMA_14	EMA_26	EMA_12
count	3211.000000	3211.000000	3211.000000	3211.000000	3.211000e+03	3211.000000	3211.000000	3211.000000
mean	14487.308287	14829.537689	14113.021313	14494.491862	1.699069e+10	14439.926956	14390.115195	14448.349499
std	16024.463616	16415.970753	15577.532742	16021.085707	1.933438e+10	15977.218348	15886.618552	15962.657676
min	176.897003	211.731003	171.509995	178.102997	7.860650e+06	222.873070	230.699121	224.309373
25%	1129.975037	1171.159973	1108.289978	1138.530029	2.814350e+08	1098.014531	1097.373689	1110.561697
50%	8320.286133	8514.666992	8141.180176	8319.472656	1.212860e+10	8328.775565	8375.353008	8406.072536
75%	23094.459961	23472.067383	22701.054688	23127.910156	2.764339e+10	22909.110421	22618.321260	22866.703847
max	67549.734375	68789.625000	66382.062500	67566.828125	3.509679e+11	63983.060826	62406.255181	64182.504548

El siguiente paso en el preprocesamiento de los datos de tipo series temporales consiste en comprobar su homogeneidad en términos de la tipología de los datos. Analizamos en la siguiente imagen la existencia de valores que son tratados como objetos, pero en verdad, son datos numéricos, por lo que procedemos a su transformación en tipos *float* para homogeneizarlos. En el caso del volumen, podemos ver que es de tipo entero ya que no es posible la venta o compra de una parte de un activo.

```
1 df.dtypes
```

```
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
SMA_14    float64
EMA_26    float64
EMA_12    float64
MACD       object
MACDh      object
MACDs      object
CCI        object
ADX        object
RSI        object
ATR        object
OBV       float64
CMF        object
dtype: object
```

```
1 #df.index= pd.to_datetime(df.index)
```

```
1 columns = df.select_dtypes(include='object').columns
2 df[columns] = df[columns].astype("float")
```

```
1 df.dtypes
```

```
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
SMA_14    float64
EMA_26    float64
EMA_12    float64
MACD      float64
MACDh     float64
MACDs     float64
CCI       float64
ADX       float64
RSI       float64
ATR       float64
OBV       float64
CMF       float64
dtype: object
```

Una vez realizadas estas transformaciones obtenemos un dataset libre de valores nulos y con un formato homogéneo con el que poder trabajar. Como estamos tratando con series de datos temporales, vamos a ver si existen valores faltantes dentro del rango temporal de datos. En la siguiente imagen se puede comprobar que no obtenemos ningún dato faltante. Por lo que, tenemos un dataset limpio y procesado listo para ser utilizado.

```
1 #Check any missing value
2
3 pd.date_range(start = '2015-01-01', end = '2023-10-16' ).difference(df.index)
```

```
DatetimeIndex([], dtype='datetime64[ns]', freq='D')
```

En segundo lugar, analicemos los datos de texto, es decir, el dataset con los artículos financieros. Al tratarse de texto, podemos encontrar caracteres que no son reconocibles por los modelos de procesamiento del lenguaje natural, para ello vamos a homogeneizar cada artículo con caracteres entendibles por los modelos y, a su vez, castearlos a tipo *string*.

```
df.articles =  df.articles.str.replace('[^0-9a-zA-Z\s]', ' ').astype('string')
```

Veamos si los tipos de datos en las distintas columnas concuerdan con el contenido de las mismas. Como observamos a continuación, tenemos datos temporales y datos en formato texto.

```
df.dtypes
```

```
date        datetime64[ns]
articles            string
dtype: object
```

Una vez realizado este proceso, nos queda analizar la existencia de valores nulos o valores faltantes dentro del rango temporal de los datos recolectados.

```
# Check any null value

df.isnull().sum()
```

```
articles    0
dtype: int64
```

```
#Check any missing value

pd.date_range(start = '2018-02-25', end = '2023-02-24' ).difference(df.index)
```

```
DatetimeIndex([], dtype='datetime64[ns]', freq='D')
```

Como observamos en la imagen anterior, no existen valores nulos en nuestro dataset y, para el rango de fechas en los que tenemos datos, no hay ningún valor faltante.

Una vez analizada la calidad de los datasets y llevado a cabo el proceso de preprocesamiento de los mismos, hemos obtenido unos dataframes limpios y listos para ser explotados y modelizados.

5.2.2. Feature engineering

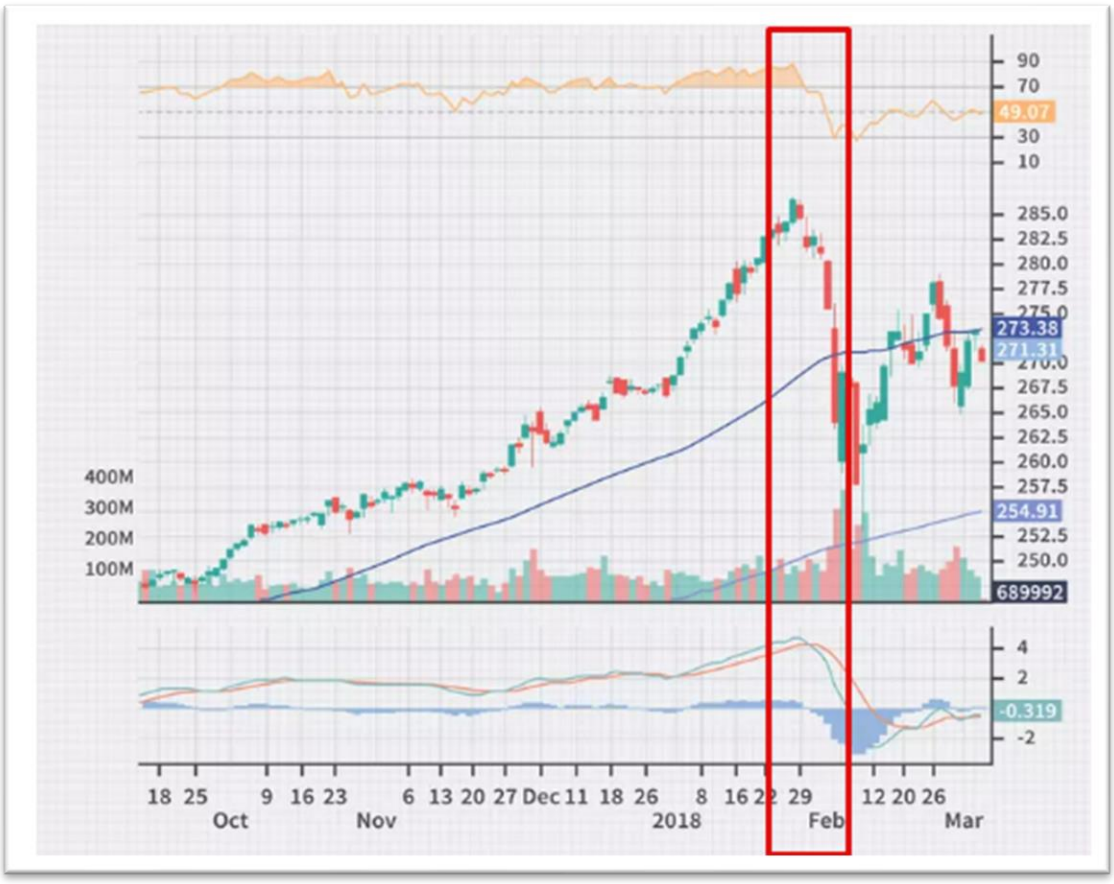
Feature engineering es un proceso de creación de nuevas variables basadas en los datos en bruto existentes para facilitar el análisis y modelado del proyecto. Con esta técnica no se añade nueva información si no que se aumenta la utilidad de los datos que ya teníamos. Es una tarea creativa y contextual, es decir, dependiendo del tipo de problema que nos estemos planteando y del objetivo final del mismo pueden crearse diferentes características relevantes para el mismo.

En primer lugar, hemos aplicado esta técnica a los datos de los precios para crear diferentes indicadores técnicos que nos proporcionen más información acerca de la volatilidad y la tendencia del precio de Bitcoin.

Los indicadores técnicos son cálculos matemáticos basados en el precio y el volumen de un valor de mercado usados por inversores para realizar un análisis técnico de dicho valor, es decir, examinar las inversiones e identificar oportunidades de compra o venta. Se centran en patrones del movimiento del precio y nos proporcionan señales de aumento o de descenso del precio.

En el siguiente gráfico podemos ver 3 de los indicadores más comunes: RSI o índice de fuerza relativa (gráfico amarillo en la parte superior), medias móviles a 50 y 200 días (líneas azules por encima del gráfico de los precios) y media móvil convergencia/divergencia o MACD (líneas roja y verde en el gráfico del volumen en la parte inferior).

Figura 5: Demostración del funcionamiento de algunos indicadores técnicos



Fuente: Technical Indicator: Definition, analyst uses, types and examples. James Chen. 2021. (https://www.investopedia.com/terms/t/technicalindicator.asp)

Sin entrar en mucho detalle y con el fin de mostrar cómo los indicadores nos aportan información acerca del movimiento del precio, veamos en el anterior gráfico un simple ejemplo. Como vemos, el indicador RSI es superior a 70, lo que nos indica que el volumen de compra es mucho más alto que el de venta, por lo que el precio va a aumentar. En el momento que empieza a descender, vemos que el precio desciende también, por lo que ambas gráficas se mueven simultáneamente en la misma dirección. Lo mismo observamos en el indicador MACD, la tendencia es muy parecida a la de los precios.

Esta explicación es muy sencilla y hay muchas más señales que estamos ignorando con el propósito de enseñar de una manera sencilla cómo los indicadores técnicos nos aportan información acerca de las fluctuaciones del precio del valor de mercado de interés.

Volviendo a nuestro caso específico, los indicadores técnicos escogidos son los más usados dentro de las 4 categorías de momento, tendencia, volatilidad y volumen.

Indicadores de momentum

Los indicadores de momento miden la fuerza o debilidad del precio del valor de mercado. Es decir, miden la velocidad o la ratio al que el precio varía. Los indicadores más relevantes son la media móvil simple, la media móvil exponencial y RSI.

SMA o media móvil simple calcula la media de precios de una franja de tiempo. Nos indica si el precio de un valor va a continuar creciendo o va a cambiar de tendencia alcista a bajista. Esta media puede ser mejorada como una media móvil exponencial o EMA en la cual los precios más cercanos a la fecha presente tienen un mayor peso a la hora de calcular la media. Ambas medias son similares y por ello son interpretadas de la misma manera.

El índice de fuerza relativa o RSI mide la magnitud a la que el precio actual de un valor varía, evaluando si dicho precio está sobrevalorado o infravalorado. Su valor fluctúa entre 0 y 100. Normalmente se establecen barreras entre 30 y 70, entre estos valores no hay señales claras para asegurar que el precio del activo está sobreestimado o infraestimado. Cuando el valor supera los 70 puntos, significa que el volumen de compra del activo es muy alto, por lo que puede inducir a un precio más alto de lo que debería, la mayor demanda del activo hace que los precios suban. Lo contrario pasa cuando es menor de 30 puntos, el volumen de venta es alto por lo que su valor decrece más de lo que debería.

Indicadores de tendencia

Los de tendencia nos sirven, como el nombre indica, para identificar y seguir tendencias del mercado. Es bastante parecido a los indicadores de momento y algunos de estos indicadores son clasificados indistintamente en ambas categorías. En este caso, hemos escogido MACD, CCI y ADX.

Moving average convergence divergence o MACD nos muestra cambios en la dirección de precio del valor de mercado, se calcula como la diferencia entre la media móvil de los 12 y de los 26 anteriores días.

El Índice de la Confianza del Consumidor o CCI es una encuesta realizada por *The Conference Board* que mide cómo de optimistas o pesimistas son los consumidores respecto a su situación financiera. Si

un consumidor es optimista significa que gastará más dinero y estimulará la economía, el caso contrario ocurre cuando un usuario es pesimista.

Por último, ADX o *average directional index* es la media de la intensidad de la tendencia y su cálculo se basa en las medias móviles de 14 periodos. Es un indicador que se mueve entre 0 y 100, es decir, cuando el indicador es cercano a 0 significa que la intensidad de crecimiento o decrecimiento del precio es pequeña, mientras que, si es cercana 100, la fluctuación alcista o bajista del precio es grande.

Indicadores de volatilidad

Los indicadores de volatilidad permiten medir y entender los periodos de alta o baja volatilidad ayudando a determinar el riesgo asociado a una inversión. El indicador usado es el ATR o rango verdadero promedio que mide la volatilidad del valor de mercado derivado de la media móvil simple de, normalmente, 14 periodos. Muestra la media del rango de movimiento de precios en el periodo seleccionado.

Indicadores de volumen

Los indicadores de volumen miden la percepción de los inversores sobre un valor de mercado midiendo el número de usuarios interesados en comprar y vender en un momento concreto. Si la percepción es negativa, el precio del activo podría disminuir, mientras que si es positiva tenderá a aumentar. Básicamente estos indicadores se basan en la teoría de la oferta y la demanda. Los indicadores seleccionados son el OBV y CMF.

On-Balance Volume o OBV usa el movimiento en el volumen de compra y venta para predecir cambios en el precio del valor de mercado.

Por otro lado, el *Chaikin Money Flow* o CMF mide la acumulación o distribución de un valor de mercado en un periodo establecido, usualmente de 21 periodos. Varía entre -1 y 1. Cada vez que cruza la barrera de 0 nos indica que el flujo de dinero está cambiando, es decir, si hay más ventas que compras o viceversa.

Una vez introducidos los indicadores técnicos usados en el análisis del precio de Bitcoin, vamos a hacer uso de la librería de pandas_ta para calcularlos e introducirlos en el dataframe de series temporales. Se puede observar en el gráfico siguiente el código usado para dicha inserción en Jupiter Notebook.

```
macd = data.ta.macd(close= data.Close, window_slow= 26, window_fast= 12, window_sign= 9, fillna = False)
```

```
cci = data.ta.cci(high= data.High, low=data.Low, close=data.Close, window=20, constant=0.015, fillna=False)
```

```
adx = data.ta.adx(high= data.High, low=data.Low, close=data.Close, window=14, fillna=False)
```

```
rsi = ta.momentum.rsi(close= data['Close'], window = 14, fillna = False)
```

```
atr = data.ta.atr(high= data.High, low=data.Low, close=data.Close, window = 14, fillna= False)
```

```
obv = data.ta.obv(close=data.Close, volume=data.Volume, fillna = False)
```

```
cmf = data.ta.cmf(high= data.High, low=data.Low, close=data.Close, volume=data.Volume, window = 20, fillna = False)
```

```
data = data.assign(
    SMA_14 = data['Close'].rolling(14).mean(),
    EMA_26 = data['Close'].ewm(span=26, adjust=False).mean(),
    EMA_12 = data['Close'].ewm(span=12, adjust=False).mean(),
    MACD = macd.MACD_12_26_9,
    MACDh = macd.MACDh_12_26_9,
    MACDs = macd.MACDs_12_26_9,
    CCI = data.index.map(cci),
    ADX = data.index.map(adx.ADX_14),
    RSI = data.index.map(rsi),
    ATR = data.index.map(atr),
    OBV = data.index.map(obv),
    CMF = data.index.map(cmf),
                )

```

En segundo lugar, aplicamos la técnica de feature engineering para analizar el sentimiento de los artículos financieros relacionados con la criptomoneda de Bitcoin. Para ello, vamos a utilizar dos librerías diferentes para analizar, por un lado, la subjetividad y polaridad y, por otro lado, el sentimiento positivo, negativo o neutro de dichos artículos.

La primera de estas librerías es textblob. Es una API que nos da acceso a diferentes funciones del procesamiento del lenguaje natural como el análisis de sentimientos. Esta librería nos proporciona dos salidas:

- Polaridad: mide el sentimiento del texto aportado. Toma valores entre -1 y 1 indicando si es positivo (1) o negativo (-1).
- Subjetividad: mide la cantidad de opinión personal dentro de un texto. Fluctúa entre 0 y 1. Si el texto contiene una mayor cantidad de opinión personal en vez de simples hechos el valor se acercará a 1, mientras que si en el texto se describen hechos el valor tenderá a 0.

En la siguiente imagen se puede observar el código para recolectar ambas salidas anteriormente descritas y para añadir dichos resultados como nuevas columnas del dataframe de los artículos

```
#Get subjectivity

def Subjectivity(text):
    return TextBlob(text).sentiment.subjectivity

#Get polarity

def Polarity(text):
    return TextBlob(text).sentiment.polarity
```

```
#Add coluns to the dataset

df['Subjectivity'] = df['articles'].apply(Subjectivity)

df['Polarity'] = df['articles'].apply(Polarity)
```

La segunda de las librerías es la de vaderSentiment. VADER o *Valence Aware Dictionary and Sentiment Reasoner* funciona de una manera muy parecida a text blob con la gran diferencia de que esta, no solo nos dice es qué grado el léxico usado es positivo negativo o neutro, sino que también nos da una puntuación del conjunto de palabras que el texto. Este último parámetro se calcula a través de la normalización de las anteriores 3 calificaciones entre -1 y 1. Por lo que las salidas del uso de esta librería sobre los artículos financieros son 4, como vemos en la siguiente imagen donde se muestra el código escrito en Python para su obtención y posterior inserción en el dataframe.

```
#Get the sentiment score

def Sentiment(text):
    sia = SentimentIntensityAnalyzer()
    sentiment = sia.polarity_scores(text)
    return sentiment
```

```
compound = []
neg = []
pos = []
neu = []


for i in range(len(df.articles)):
    SIA = Sentiment(df.articles[i])
    compound.append(SIA['compound'])
    neg.append(SIA['neg'])
    pos.append(SIA['pos'])
    neu.append(SIA['neu'])
```

```
df['sentiment'] =  compound
df['negative'] =  neg
df['positive'] =  pos
df['neutral'] =  neu
```

Una vez añadidas estas nuevas características obtenemos el siguiente dataframe

date	articles	Subjectivity	Polarity	sentiment	negative	positive	neutral
2018-02-25	Original Pizza Day Purchaser Does It Again W...	0.441667	0.220833	0.7788	0.021	0.087	0.892
2018-02-26	Bitcoin Pizza Day 2 How A Lightning Payment...	0.446667	0.010000	-0.6597	0.080	0.037	0.883
2018-02-27	Rapper 50 Cent Who Bragged About Owning Bit...	0.518506	0.001623	-0.6705	0.073	0.034	0.892
2018-02-28	This Is Who Controls Bitcoin British Man ...	0.459091	0.039394	0.4939	0.038	0.093	0.869
2018-03-01	Bitcoin makes inroads in LA s residential re...	0.335000	-0.083333	-0.1543	0.097	0.081	0.822
...	...	...	...	...	...	...	...
2023-02-20	Bitcoin regains 25K amid hope record China ...	0.575510	0.102041	0.1952	0.113	0.136	0.751
2023-02-21	Bitcoin active addresses concern analyst d...	0.350000	0.083333	0.8225	0.035	0.124	0.841
2023-02-22	Bitcoin Ethereum Technical Analysis BTC Fa...	0.283333	0.166667	-0.1027	0.052	0.040	0.908
2023-02-23	Bitcoin bears attempt to pin BTC price under...	0.334407	0.057197	-0.2382	0.072	0.073	0.855
2023-02-24	Bitcoin on chain data highlights key similar...	0.733333	0.250000	0.7960	0.000	0.083	0.917

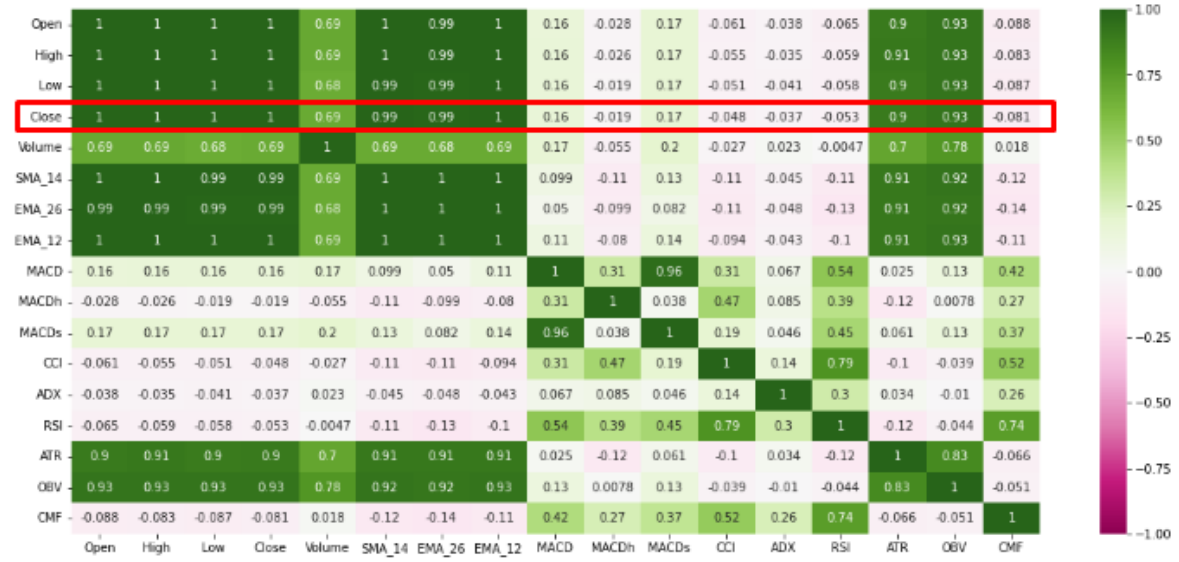
1826 rows × 7 columns

5.2.3. Estudio de las variables

Tras la generación de nuevas características en ambos datasets, es necesario hacer un estudio del nivel de explicación de las variables independientes respecto de la variable dependiente. Para ello, vamos a realizar un análisis de la correlación de variables respecto a nuestra variable dependiente o target, es decir, el precio de cierre.

```
fig = plt.subplots(figsize=(18,8))
sns.heatmap(df.corr(), cmap="PiYG", vmin=-1.0, vmax=1.0, annot=True)

```

```
<AxesSubplot:>
```



Con el fin de minimizar el ruido en el dataset, vamos a eliminar las variables que tienen una correlación más cercana a 0. Una correlación cercana al valor nulo significa que la variable independiente no explica en prácticamente ninguna medida la variabilidad de la variable dependiente, mientras que una correlación cerca al 1 positivo o negativo, nos dice que la variable independiente tiene una alta relevancia para la explicación del movimiento de la dependiente. En el caso de ser negativo la relación es inversa, es decir, cuando la variable independiente crece, la dependiente decrece, lo contrario ocurre cuando es uno positivo, es decir, si la independiente crece, la dependiente crecerá también.

Por lo que, nuestro nuevo dataset tendrá las características más relevantes a la hora de explicar las fluctuaciones de la variable dependiente o target, es decir, el precio de cierre. Vamos a continuación, en la siguiente imagen dichas características.

```
DF = df[['Open','High','Low','Close', 'Volume', 'SMA_14', 'EMA_26','EMA_12', 'MACD','MACDs', 'ATR', 'OBV']]

DF.to_csv('DF_BTC.csv', encoding='utf-8')
```

Una vez recolectados los datos en bruto, analizada su calidad, limpiados y preprocesados y guardados en archivos separados por comas o CSV, tenemos como resultado dos datasets listos para ser modelados.

En este proyecto vamos a trabajar con un modelo híbrido, por lo que, una vez obtenidos los datasets anteriormente mencionados, tenemos que proceder a su unión. Como vemos en la siguiente imagen, obtenemos un nuevo dataset con las siguientes columnas: 'Open', 'High', 'Low', 'Close', 'Volume', 'SMA_14', 'EMA_26', 'EMA_12', 'MACD', 'MACDs', 'ATR', 'OBV', 'Subjectivity', 'Polarity', 'sentiment', 'negative', 'positive' y 'neutral', un total de 18 columnas con 1826 filas de datos.

```
1 df = pd.merge(df_prices_TI, df_sentiment, left_index=True, right_index=True)
```

```
1 df.columns
```

```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'SMA_14', 'EMA_26', 'EMA_12',
       'MACD', 'MACDs', 'ATR', 'OBV', 'Subjectivity', 'Polarity', 'sentiment',
       'negative', 'positive', 'neutral'],
      dtype='object')
```

5.3. Aplicación de los modelos

En el apartado 4.4 se han mencionado los modelos escogidos para este proyecto tras la investigación realizada sobre los diferentes estudios publicados hasta el momento.

En este apartado, vamos a profundizar en cada uno de ellos mostrando su aplicación con los datos recolectados.

Para aportar más información acerca de los beneficios de llevar a cabo un modelo híbrido donde se combinan los datos de los históricos de precios de Bitcoin con indicadores técnicos e indicadores extraídos del análisis de sentimientos realizado, hemos aplicado los diferentes modelos al dataset híbrido y, también, al dataset guardado que contienen solamente el histórico de precios, con la finalidad comparativa de los resultados obtenidos por ambos.

5.3.1. ARIMA

La predicción de series temporales es tradicionalmente usada en econometría empleando el modelo ARIMA, el cual ha sido el método estándar utilizado para este tipo de problemas por mucho tiempo. Aunque esté muy extendido en el modelado de series de datos temporales en campos como el económico o financiero, presenta una serie de limitaciones, como, por ejemplo, la dificultad para modelizar variables con relaciones no lineales. A pesar de ello, vamos a estudiar su comportamiento y analizar los resultados obtenidos con los otros modelos.

Modelo de medias móviles autorregresivas integradas o ARIMA es un modelo ampliamente usado para la predicción de series de datos temporales. La terminología usada en ARIMA se puede descomponer en 3 elementos clave:

- AR: autorregresiva. Es un modelo que muestra una variable cambiante que hace regresión sobre los valores previos (p), es decir, un modelo que predice el comportamiento de la variable target basándose en el comportamiento pasado de los datos.
- I: integrada. Haciendo estacionaria la serie de datos temporal midiendo las diferencias entre las observaciones en diferentes momentos (d).
- MA: media móvil. Incorporando la dependencia entre una observación y el error residual cuando se utiliza un modelo de medias móviles en las observaciones pasadas (q).

Cada uno de estos elementos funciona como un parámetro dentro del modelo, los que podemos definir de la siguiente manera:

- p: número de observaciones pasadas en el modelo.
- d: número de veces que los datos han sido sustraídos por valores pasados.
- q: el tamaño de la ventana en la que se mueve la media móvil.

La función matemática del modelo es la siguiente:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + ..+ \phi_q \epsilon_{t-q}$$

Una vez descrito el modelo en términos matemáticos vamos a ver cómo se implementa. Para la implementación del modelo vamos a seguir una serie de pasos:

I. Importar las librerías necesarias

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
```

```
from pmdarima.arima import auto_arima
```

La librería de pandas se utiliza para la lectura del fichero CSV que contiene el dataset con los precios históricos, los indicadores técnicos y los indicadores del análisis de sentimientos.

Las librerías de matplotlib y seaborn son empleadas para la representación gráfica.

El uso de sklearn y math es necesario para el cálculo de las métricas que nos ayudarán a comparar los resultados de cada modelo.

Por último, la librería de pmdarina nos permite seleccionar el modelo para la predicción de series de datos temporales.

II. Dividir los datos en conjunto de entrenamiento y prueba

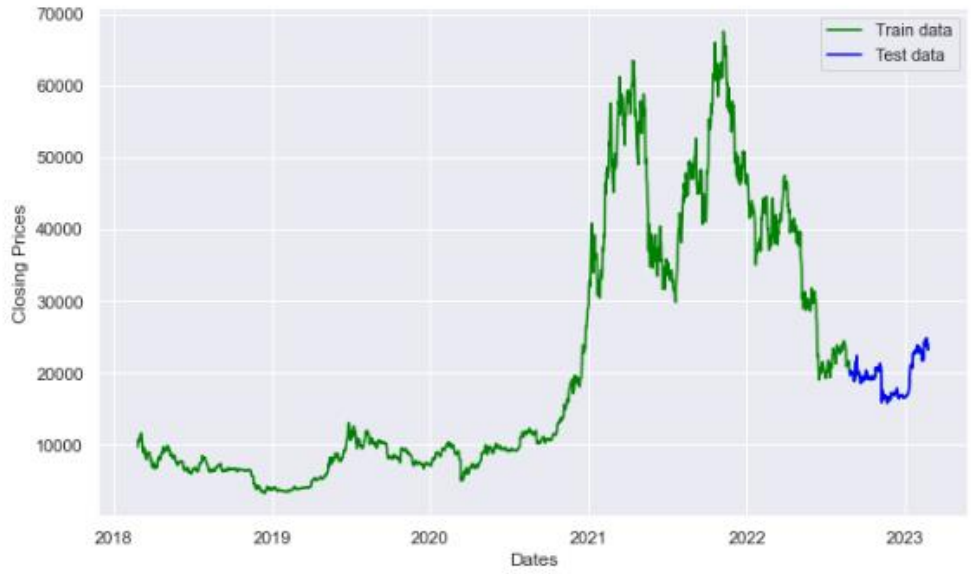
En este caso, vamos a dividir los datos en dos conjuntos: entrenamiento y prueba. Debido a la baja cantidad de datos, hemos decidido dividir los grupos en una proporción de 90/10.

```
#Split the dataset into trian and test

train_data_1, test_data_1 = df_1[:int(len(df_1)*0.9)], df_1[int(len(df_1)*0.9):]
```

Al tratarse de series temporales, dichos conjuntos no pueden crearse con la randomización de valores, es decir, seleccionando valores aleatorios dentro de la serie temporal, por lo que hemos creado el primero de los grupos, el de entrenamiento, cogiendo el 90% de los datos desde el comienzo de eje temporal, mientras que el grupo de prueba es el último 10%. Veámoslo gráficamente a continuación.

```
plt.figure(figsize=(10,6))
plt.xlabel('Dates')
plt.ylabel('Closing Prices')
plt.plot(df_1.Close, 'green', label='Train data')
plt.plot(test_data_1.Close, 'blue', label='Test data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x22a49bece20>
```



III. Selección del modelo ARIMA

Usando la función *auto_arima* de la librería de pmdarima se seleccionan las variables p, d y q más óptimas para nuestro caso, maximizando el AIC. El AIC es un método matemático para la evaluación del ajuste de un modelo a los datos para los que es generado. Este modelo resultante lo almacenamos en la variable arima_model.

```
arima_model = auto_arima(train_data_1['Close'], error_action='ignore',
                         suppress_warnign=True, stepwise=False,
                         approximation=False, seasonal=False)
```

```
arima_model.summary()
```

SARIMAX Results

Dep. Variable:	y	No. Observations:	1643
Model:	SARIMAX(2, 1, 2)	Log Likelihood	-13730.782
Date:	Fri, 08 Sep 2023	AIC	27473.563
Time:	14:24:45	BIC	27505.985
Sample:	02-25-2018	HQIC	27485.587
	- 08-25-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	5.6345	27.020	0.209	0.835	-47.324	58.593
ar.L1	0.9261	0.012	77.310	0.000	0.903	0.950
ar.L2	-0.9497	0.012	-79.013	0.000	-0.973	-0.926
ma.L1	-0.9537	0.009	-108.871	0.000	-0.971	-0.936
ma.L2	0.9776	0.009	112.352	0.000	0.961	0.995
sigma2	1.087e+06	1.69e+04	64.487	0.000	1.05e+06	1.12e+06

Ljung-Box (L1) (Q):	0.14	Jarque-Bera (JB):	5628.04
Prob(Q):	0.71	Prob(JB):	0.00
Heteroskedasticity (H):	26.24	Skew:	-0.23
Prob(H) (two-sided):	0.00	Kurtosis:	12.06

Como vemos, el modelo que mejor se ajusta a nuestros datos es el modelo ARIMA (2, 1, 2). En la tabla inferior, podemos ver los coeficientes de la función matemática descrita al principio.

IV. Usar el modelo para predecir los valores en el conjunto de prueba

En este paso usamos el modelo creado en el apartado 3 y almacenado en la variable llamada 'arima_model' para predecir la variable target, es decir, el precio de cierre, en un grupo de datos de prueba.

```
# Predecir los valores en el conjutno de prueba
```

```
n_future = len(test_data_1)
```

```
arima_pred = arima_model.predict(n_periods=n_future, return_conf_int=True, alpha=0.05)
```

```
arima_pred = [pd.DataFrame(arima_pred[0], columns=['prediction']),
              pd.DataFrame(arima_pred[1], columns=['low_95', 'up_95'],
              index=arima_pred[0].index)]


```

```
arima_pred = pd.concat(arima_pred, axis=1)
```

Con este modelo obtenemos 3 salidas, el valor predicho para el precio de cierre y el rango superior e inferior para un intervalo de confianza del 95%, establecido en la función como alpha = 0.5. Esto significa que, con una confianza del 95%, podemos asegurar que el valor de la predicción del precio de cierre se encuentra dentro del rango de valores establecidos por el límite inferior y superior.

```
arima_pred
```

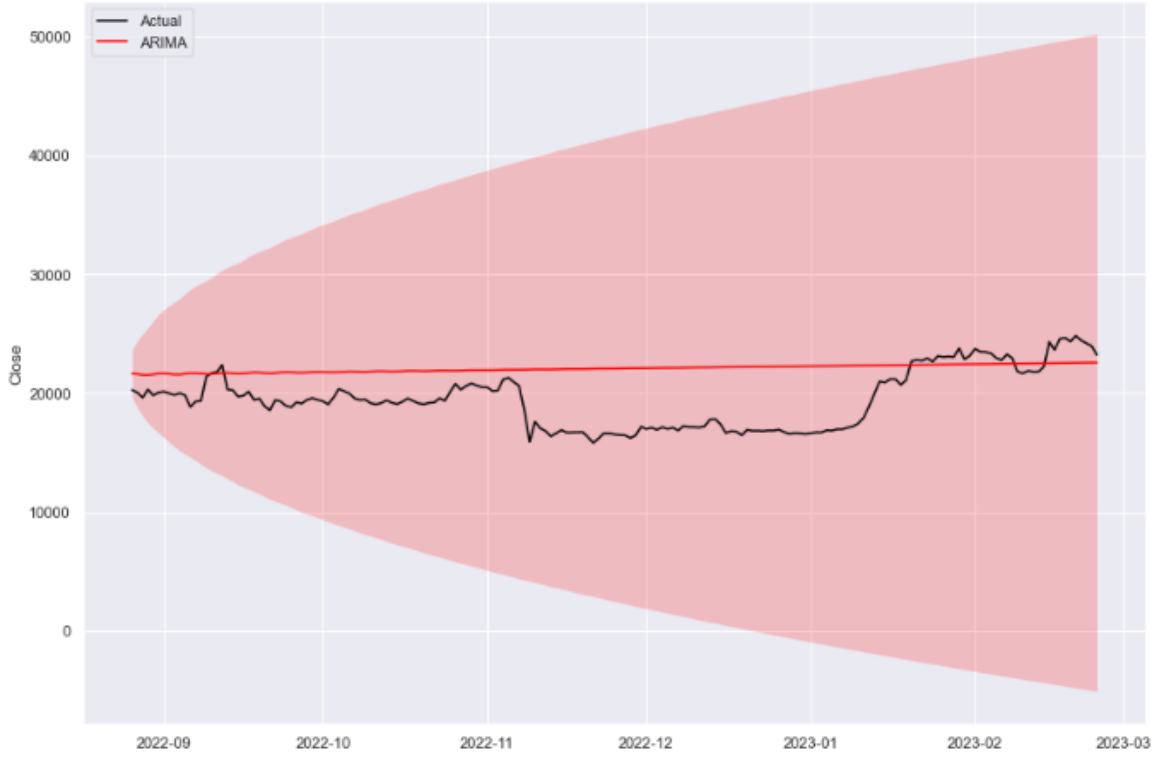
	prediction	low_95	up_95
2022-08-26	21634.909130	19591.457327	23678.360933
2022-08-27	21590.991408	18740.623280	24441.359537
2022-08-28	21523.657371	18046.144589	25001.170154
2022-08-29	21508.643090	17471.808144	25545.478036
2022-08-30	21564.322326	17014.328226	26114.316426
...	...	...	...
2023-02-20	22539.220219	-4806.882804	49885.323241
2023-02-21	22545.119351	-4877.320918	49967.559619
2023-02-22	22551.379413	-4947.190823	50049.949650
2023-02-23	22557.208865	-5017.241403	50131.659133
2023-02-24	22562.296731	-5087.784254	50212.377715

183 rows × 3 columns

Para tener una imagen más clara veamos a continuación una representación gráfica de estas 3 salidas.

V. Imprimir los resultados

```
#Print results

fig, ax = plt.subplots(1, figsize=(12, 8))

ax = sns.lineplot(data=test_data_1['Close'], color='black', label= 'Actual')

ax.plot(arima_pred.prediction, color='red', label= 'ARIMA')

ax.fill_between(arima_pred.index, arima_pred.low_95, arima_pred.up_95, alpha=0.2,
                facecolor='red')

ax.legend(loc='upper left')

plt.tight_layout()
plt.show()
```



La línea negra representa los valores del precio de cierre reales, mientras que la línea roja representa los predichos por el modelo. La banda de color rojizo representa el intervalo de confianza. Podemos ver en el gráfico que los precios de cierre están dentro de la banda de confianza, lo que puede sugerir que puede ser un buen modelo predictivo.

También, se puede observar que los errores cometidos por el modelo son pequeños pero considerables, por ello, vamos a evaluar su ejecución en el siguiente paso con una serie de métricas estadísticas

VI. Uso de métricas para evaluar los resultados

Para evaluar el comportamiento del modelo vamos a usar 4 métricas regresivas, las cuales hemos explicado en profundidad en el apartado 4.7.

```
# report performance
mse = mean_squared_error(test_data_1.Close, arima_pred.prediction)
print('MSE: '+str(mse))
mae = mean_absolute_error(test_data_1.Close, arima_pred.prediction)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_data_1.Close, arima_pred.prediction))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(arima_pred.prediction - test_data_1.Close)/np.abs(test_data_1.Close))
print('MAPE: '+str(mape))
```

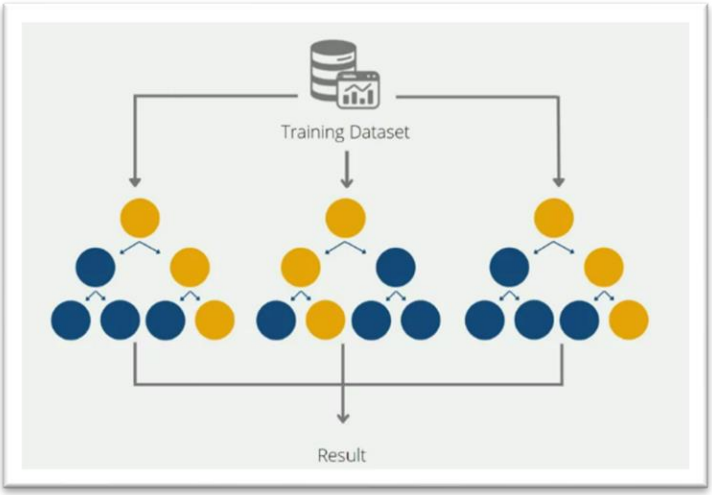
```
MSE: 12106086.819315514
MAE: 2923.815705898013
RMSE: 3479.380234943504
MAPE: 0.16341654684169693
```

Fijémonos en la métrica MAPE. El valor obtenido es un 16.34%. Eso quiere decir que de media el valor predicho se desvía del valor real un 16%. Un valor entre el 10 y el 25%, es aceptable, pero tiene una baja exactitud. En otras palabras, este modelo tiene una precisión del 83% prediciendo el precio de cierre de la criptomoneda Bitcoin. Vamos a ver si podemos encontrar otro modelo donde este valor sea menor.

5.3.2. XGBoost

XGBoost o *extreme gradient boosting* es un algoritmo de aprendizaje automático. Es un ensamblaje de una colección de árboles de decisión que se usan para hacer predicciones. Este tipo de algoritmos boosting utilizan modelos simples de manera secuencial aprovechando las predicciones de cada modelo simple, las cuales se combinan para producir la predicción final. La velocidad de ejecución es alta y su rendimiento es uno de los mejores, por ello, es uno de los modelos preferidos entre los científicos de datos.

Figura 6: Esquema del funcionamiento del modelo XGBoost.



Fuente: What is XGBoost? None autor, 2023. (https://databasecamp.de/en/ml/xgboost-en)

La librería XGboost es de software libre y su objetivo es proveer una librería de *gradient boosting* escalable, portable y distribuida.

Como en el apartado anterior, vamos a explicar su implementación siguiendo una serie de pasos.

I. Importar las librerías necesarias

```
import xgboost as xgb
from sklearn.model_selection import  GridSearchCV
from sklearn.model_selection import train_test_split
```

En este caso, las librerías usadas son la librería de xgboost para el desarrollo del modelo y la librería sklearn desde la cual importamos dos funcionalidades: GridSearchCV y train_test_split.

GridSearchCV es una técnica de validación cruzada empleada para buscar los parámetros óptimos dentro de un conjunto de parámetros aportados. La validación cruzada evalúa el modelo en distintos subconjuntos de datos, en este caso las distintas combinaciones de parámetros.

La funcionalidad de train_test_split, como el nombre indica, nos ayuda a separar los datos de entrada en los conjuntos de entrenamiento y prueba.

II. Separar la variable dependiente de las independientes y dividir ambos conjuntos en entrenamiento y prueba

```
# Split the data in target and independant variables

x = df_1.drop(['Close'], axis = 1)

y = df_1['Close']
```

```
# Splitting to training and testing data

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, shuffle=False)
```

Como en el anterior modelo, nuestra variable target o dependiente es el precio de cierre.

A diferencia del anterior caso, donde usamos ARIMA, los grupos de entrenamiento y prueba se dividen siguiendo una distribución de 80/20, ya que tras hacer una serie de pruebas encontramos esta combinación la más óptima.

III. Búsqueda de los parámetros óptimos con GridSearchCV

```
%%time

parameters = {
    'n_estimators': [100, 200, 300, 400],
    'learning_rate': [0.001, 0.005, 0.01, 0.05],
    'max_depth': [8, 10, 12, 15],
    'gamma': [0.001, 0.005, 0.01, 0.02],
    'random_state': [42]
}

model = xgb.XGBRegressor( objective='reg:squarederror')
clf = GridSearchCV(model, parameters)

clf.fit(x_train, y_train)

print(f'Best params: {clf.best_params_}')
print(f'Best validation score = {clf.best_score_}')
```

```
Best params: {'gamma': 0.005, 'learning_rate': 0.05, 'max_depth': 10, 'n_estimators': 400,
'random_state': 42}
Best validation score = 0.9802578428069765
Wall time: 7min 23s
```

La combinación de parámetros óptimos se puede ver en la salida de esta celda de código. También se imprime cuál ha sido la puntuación de la validación cruzada para este conjunto de datos y, por último, el tiempo que ha tardado en ejecutar la búsqueda, en este caso, 7 minutos.

IV. Definición y entrenamiento del modelo con los parámetros óptimos

El siguiente paso es definir el modelo de XGBoost usando los parámetros encontrados en la búsqueda del paso anterior y entrenarlo con el grupo de datos de entrenamiento.

```
%%time

model_xgb = xgb.XGBRegressor(**clf.best_params_, objective='reg:squarederror')
model_xgb.fit(x_train, y_train)
```

```
Wall time: 1.44 s

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.02, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=15, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=400, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, ...)
```

V. Predicción de la variable target en el conjunto de prueba

Utilizamos el modelo creado para predecir la variable target en el conjunto de datos de prueba y lo almacenamos en la variable ‘y_pred’.

```
y_pred = model_xgb.predict(x_test)
```

VI. Representación de los resultados

Para la representación de los resultados, vamos a almacenar en un nuevo dataframe llamado 'prediction', por un lado, los valores del precio de cierre reales para el horizonte temporal del conjunto de prueba y, por otro, las predicciones calculadas en el paso anterior.

```
# Create dataframe

prediction = y_test.to_frame().copy()
```

```
# Model prediction on test data

prediction['y_predr'] = y_pred
```

```
figure(figsize=(15, 6), dpi=80)

plt.plot(prediction['Close'], label='y_test')
plt.plot(prediction['y_predr'], label='y_predr')
plt.legend()
plt.show()
```

En el gráfico tenemos los valores reales del precio de cierre representados en color azul y los valores predichos por el modelo en color naranja. Podemos observar que las estimaciones son muy buenas. Para asegurarnos de ello, vamos a hacer uso de las métricas estadísticas usadas en el anterior modelo.

VII. Evaluación del modelo

Para terminar, veamos las métricas obtenidas por este modelo.

```
# report performance
mse = mean_squared_error(prediction['Close'], prediction['y_predr'])
print('MSE: '+str(mse))
mae = mean_absolute_error(prediction['Close'], prediction['y_predr'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(prediction['Close'],prediction['y_predr']))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(prediction['y_predr'] - prediction['Close'])/np.abs(prediction['Close']))
print('MAPE: '+str(mape))
```

```
MSE: 1179253.6455659554
MAE: 883.043891948429
RMSE: 1085.9344573066808
MAPE: 0.039803559099890085
```

Como vemos, claramente, el rendimiento de este modelo, en comparación con el modelo ARIMA, es mucho mejor. Obtenemos una desviación porcentual del 3.98, es decir, el modelo tiene una alta precisión, para ser más precisos una precisión del 96%.

Hasta el momento hemos estudiado el rendimiento de los dos modelos tradicionales más usados. A continuación, vamos a estudiar el uso de redes neurales. Más en concreto de redes neuronales recurrentes, para la predicción de series de datos temporales y evaluar su rendimiento en comparación con los métodos tradicionales.

5.3.3.LSTM

Las redes neuronales LSTM son un tipo de redes recurrentes las cuales utilizan una célula de memoria diseñada para recordar información a largo plazo. Este flujo de información es controlado por unas puertas o gates las cuales, a su vez, están controladas por funciones de activación, como por ejemplo tangente, sigmoidal, ReLu o Softmax. La función por defecto es la tangente.

La función de activación decide si una neurona va a ser activada o no, es decir, si alcanza el umbral necesario para pasar la información a las siguientes neuronas. Como hemos explicado en el apartado

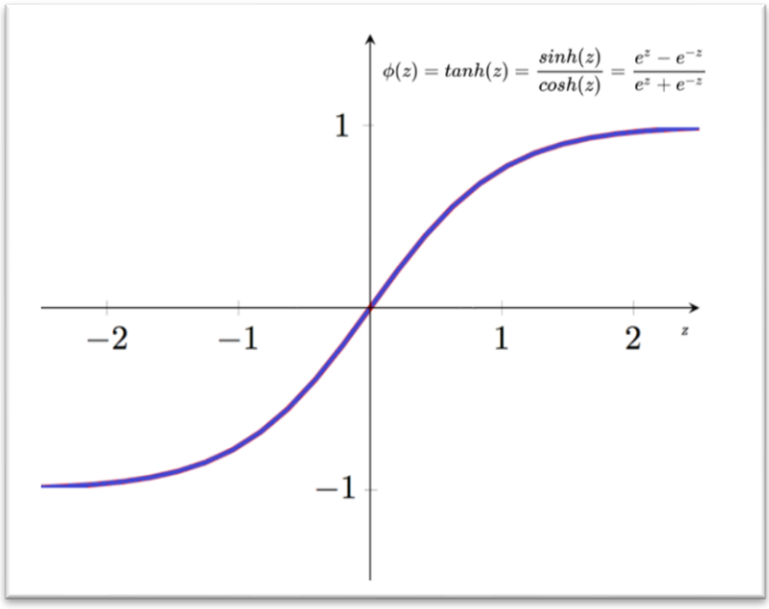
4.4, el valor de una neurona se calcula teniendo en cuenta el valor de entrada (x_i) y un peso (w_i) como se ve a continuación.

$$Y = \sum_{i=1}^{n} (x_i \; w_i) + b$$

Una vez calculado el valor de salida de la neurona, se aplica la función de activación y se decide si la neurona estará activa o no.

La función de activación tangente hiperbólica o *tanh* se mueve entre los valores -1 y 1. Se define matemáticamente de la siguiente manera:

Figura 7: Representación gráfica de la función de activación *tanh*



Fuente: Explicación de las funciones de activación en RN y practica con Python. Alberto Rubiales, 2020.

Veamos los pasos seguidos al aplicar este modelo a nuestro set de datos.

I. Importación de librerías

```
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU,LSTM, Dropout
```

En primer lugar, tenemos de la librería sklearn la funcionalidad de *MinMaxScaler* usada para escalar los datos entre un rango fijado, en nuestro caso 0 - 1, es decir, cambia en rango de valores sin cambiar su distribución. El uso de esta funcionalidad se debe a que los algoritmos de aprendizaje automático funcionan mejor cuando los valores tienen la misma escala y se distribuyen de manera similar a la normal.

Por otro lado, tenemos la librería *TensorFlow* que es una de las mejores librerías para el aprendizaje automático. También usamos *keras*, la cual usa funcionalidades de *TensorFlow* y nos ayuda con la creación de capas dentro de los modelos. Como podemos ver las funcionalidades usadas son *Sequential*, la cual nos ayuda a apilar las capas dentro del modelo. También usamos las funcionalidades de *Dense*, GRU (para el siguiente caso) y LSTM, que son tipologías de capas que se pueden crear dentro de una red. Y, por último, *Dropout* es también una capa que nos ayuda a prevenir el sobreajuste desactivando algunas neuronas de forma aleatoria.

II. Escalar los datos

Escalamos todos los datos del dataset usando la función de *MinMaxScaler*. Como vemos, nos devuelve un array de 2 dimensiones con todos los datos escalados entre 0 y 1.

```
#Scale train data

scaler = MinMaxScaler()
scaler = scaler.fit(df_1)
scaled_data = scaler.transform(df_1)
```

```
scaled_data
```

```
array([[0.10200268, 0.1014717 , 0.09836495, ..., 0.08015267, 0.3625    ,
        0.69405099],
       [0.10002813, 0.10989399, 0.09986313, ..., 0.30534351, 0.15416667,
        0.66855524],
       [0.11129281, 0.11605296, 0.11164285, ..., 0.27862595, 0.14166667,
        0.69405099],
       ...,
       [0.32965329, 0.32354736, 0.32367098, ..., 0.19847328, 0.16666667,
        0.73937677],
       [0.3258174 , 0.32506993, 0.32445593, ..., 0.27480916, 0.30416667,
        0.58923513],
       [0.32201243, 0.31792058, 0.3135865 , ..., 0.        , 0.34583333,
        0.76487252]])
```

III. Dividir las variables independientes de la dependiente

Como nuestro modelo tiene en consideración los datos del pasado, creamos una función donde se utilizan una ventana de 14 últimos datos para obtener el siguiente. Hacemos este proceso tanto para la columna *target*, que es la 10 y para el resto de columnas, osea las variables independientes.

```
# Ya que el modelo LSTM alamcena memoria a largo plazo, creamos una extructura de 14 periodos.
# Por lo que, para cada elemento de grupo de entrenamiento, se usan los 14 anteriores datos.
# Basicamente cogemos los primeros 14 periodos para calcular el 15.

def X_y (df, window_size):

    X=[]
    y=[]

    for i in range(len(scaled_data)-window_size):
        row = [a for a in scaled_data[i:i+window_size]]
        X.append(row)
        label = scaled_data[i+window_size][9]       # Columna del precio de cierre
        y.append(label)

    return np.array(X), np.array(y)
```

```
X, y = X_y(scaled_data, window_size=14)

X.shape, y.shape
```

```
((1812, 14, 18), (1812,))
```

Obtenemos dos grupos de datos con las variables independiente (X) y la variable dependiente o *target* (y). Observamos que para el primer grupo tenemos un total de 1812 observaciones, para una ventana de 14 días y con 18 características o variables diferentes. Y, para el segundo, tenemos las 1812 observaciones para 1 sola característica.

IV. Dividir ambos conjuntos en grupos de entrenamiento, validación y prueba

El siguiente paso es dividir ambos grupos de datos en datos de entrenamiento, de validación y de prueba. En este caso, la división se hace siguiendo una distribución de 90/5/5.

```
X_train, y_train = X[:int(X.shape[0]*0.9)], y[:int(X.shape[0]*0.9)]
X_val, y_val = X[int(X.shape[0]*0.9):int(X.shape[0]*0.95)], y[int(X.shape[0]*0.9):int(X.shape[0]*0.95)]
X_test, y_test = X[int(X.shape[0]*0.95):], y[int(X.shape[0]*0.95):]
```

V. Definir y compilar el modelo y sus capas

Definimos un modelo con 7 capas: 3 de ellas son capas LSTM las cuales tienen 50 neuronas, otras 3 son capas *dropout*, para controlar el sobreajuste y, la última, es una capa densa con una sola neurona, que será nuestra salida del precio de cierre.

La elección de estos parámetros se ha decidido a través de prueba y error, probando diferentes combinaciones y viendo los resultados. En este caso se podría utilizar, como hemos hecho anteriormente, la función de GridSearchCV, pero el coste computacional y de tiempo hace que no merezca la pena su uso y se haga a través de pruebas.

```
model2 = Sequential()

model2.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],X_train.shape[2])))
model2.add(Dropout(0.2))
model2.add(LSTM(units=50, return_sequences=True))
model2.add(Dropout(0.2))
model2.add(LSTM(units=50))
model2.add(Dropout(0.2))
model2.add(Dense(units=1))

model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 14, 50)            13800

 dropout_3 (Dropout)         (None, 14, 50)            0

 lstm_1 (LSTM)               (None, 14, 50)            20200

 dropout_4 (Dropout)         (None, 14, 50)            0

 lstm_2 (LSTM)               (None, 50)                20200

 dropout_5 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 54,251
Trainable params: 54,251
Non-trainable params: 0
_________________________________________________________________
```

```
model2.compile(tf.keras.optimizers.Adam(learning_rate=0.01),
               loss=tf.keras.losses.MeanSquaredError())
```

Una vez definido el modelo hay que compilarlo. Esto quiere decir que se va a configurar el proceso de aprendizaje definiendo el optimizador utilizado y la función de pérdida.

Los optimizadores son métodos usados para cambiar los atributos del modelo como los pesos (w_i) y la ratio de aprendizaje con el objetivo de reducir la función de pérdida. En nuestro caso usamos el optimizador Adam (Optimizador adaptativo de estimación de momentos) que es uno de los más usados debido a su sencillez y velocidad de ejecución. Este actualizará los pesos para reducir la función de pérdida, que en nuestro caso es el error cuadrado medio.

```
model2.compile(tf.keras.optimizers.Adam(learning_rate=0.01),
               loss=tf.keras.losses.MeanSquaredError())
```

VI. Entrenar el modelo

Una vez definido y compilado nuestro modelo es hora de entrenarlo.

Definimos las variables de entrada como x_train y y_train, y el número de épocas como 10, es decir, cuantas veces se tienen en cuenta las muestras.

```
1 history2 = model2.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val) )
```

```
Epoch 1/10
51/51 [==============================] - 6s 29ms/step - loss: 0.0276 - val_loss: 2.1147e-04
Epoch 2/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0050 - val_loss: 0.0025
Epoch 3/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0042 - val_loss: 1.9957e-04
Epoch 4/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0024 - val_loss: 7.3192e-04
Epoch 5/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0018 - val_loss: 7.4799e-04
Epoch 6/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0016 - val_loss: 5.8914e-04
Epoch 7/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0013 - val_loss: 2.9749e-04
Epoch 8/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0012 - val_loss: 3.8983e-04
Epoch 9/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0010 - val_loss: 1.8980e-04
Epoch 10/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0011 - val_loss: 5.0653e-04
```

VII. Predicción del modelo en el grupo de prueba

Utilizamos el grupo de prueba para ver las predicciones obtenidas por el modelo.

Como se observa en la tabla inferior de la siguiente imagen, los valores predichos y los reales son muy parecidos, lo que nos da una primera idea de que el modelo tiene un alto rendimiento, pero veámoslo gráficamente a continuación en el siguiente paso.

```
predictions2 = model2.predict(X_test).flatten()
df2 = pd.DataFrame(data={'Predictions':predictions2, 'Real': Y_test})
df2
```

```
3/3 [==============================] - 1s 8ms/step
```

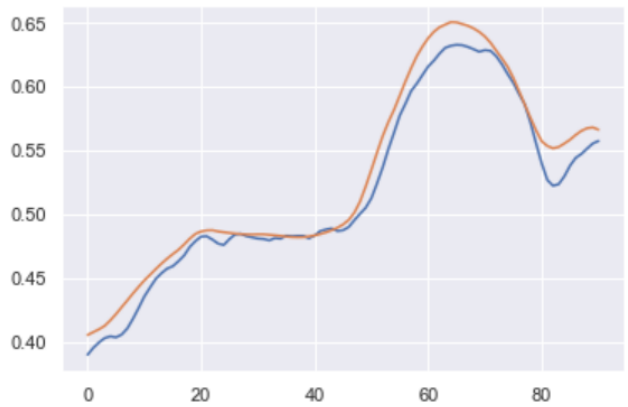
	Predictions	Real
0	0.389654	0.405175
1	0.395090	0.407416
2	0.399326	0.409628
3	0.402726	0.412307
4	0.404128	0.416618
...	...	...
86	0.544120	0.562142
87	0.547204	0.565312
88	0.551367	0.567353
89	0.555386	0.567986
90	0.557329	0.566204

91 rows × 2 columns

VIII. Representación gráfica de las predicciones

```
start = 0
end = X_test.shape[0]

plt.plot(df2.Predictions[start:end])
plt.plot(df2.Real[start:end])
```

```
[<matplotlib.lines.Line2D at 0x26cd3ef0970>]
```



Efectivamente vemos que el comportamiento del modelo se ajusta a la realidad, veamos este rendimiento con números.

IX. Evaluación con el uso de métricas

```
# report performance
mse = mean_squared_error(df2['Real'], df2['Predictions'])
print('MSE: '+str(mse))
mae = mean_absolute_error(df2['Real'], df2['Predictions'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(df2['Real'],df2['Predictions']))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(df2['Predictions'] - df2['Real'])/np.abs(df2['Real']))
print('MAPE: '+str(mape))
```

```
MSE: 0.000190875228551257
MAE: 0.011243721651645654
RMSE: 0.013815760151046956
MAPE: 0.021021895383987858
```

Como vemos, los errores se han minimizado considerablemente comparado con los dos modelos anteriores. Obtenemos un MAPE del 2.1%, es decir, el modelo tiene una precisión del 97%.

El rendimiento de este modelo es muy bueno, pero vamos a probar un último modelo, el modelo GRU que es una simplificación del modelo LSTM y es más adecuado para la cantidad de datos con los que trabajamos.

5.3.4. Gated Recurrent Networks

Como ya explicamos en el apartado 4.4, GRU o unidad recurrente cerrada es un tipo de red neuronal recurrente desarrollada en el 2014. Se caracteriza por su simplicidad en comparación con otras ya que cada unidad de memoria solo contiene dos puertas, las cuales controlan el flujo de información dentro

y fuera de ella. Las redes GRU han demostrado un mejor rendimiento en conjunto de datos pequeños, al contrario que las redes LSTM, por lo que se ajustará mejor a nuestro caso de estudio.

A continuación, mostraremos los pasos seguidos en la implementación de este modelo. Los pasos de la importación de librerías y el agrupamiento de los datos, son compartidos por el modelo LSTM por lo que vamos a obviar esos pasos e ir directamente a la definición del modelo.

I. Importación de librerías

Como ya hemos comentado anteriormente, las redes neuronales LSTM y las GRU son muy parecidas con algunas pequeñas diferencias, por lo que su implementación también es parecida. Las librerías necesarias en este caso son las mismas que en el apartado anterior, por lo que una vez que están importadas no es necesario volverlas a importar.

II. Dividir los datos en datos de entrenamiento, validación y prueba

De nuevo, los datos usados para este modelo son los mismos que hemos usado para el modelo anterior, por lo que no es necesario reescribir el código.

III. Definición y compilación del modelo

La estructura de esta red neuronal también está compuesta por 7 capas. La única diferencia con el modelo LSTM es que, en este caso, usamos las capas GRU.

Como se observa en la imagen anterior la cantidad de parámetros totales en este modelo en comparación con el anterior es menor, siendo en este un total de 41.151, mientras que en el modelo de LSTM

obtuvimos un total de 54.251. Esto se debe a que los modelo GRU utilizan menos memoria que los modelo LSTM y como veremos más adelante el tiempo de ejecución es menor.

```
model1 = Sequential()

model1.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1],X_train.shape[2])))
model1.add(Dropout(0.2))
model1.add(GRU(units=50, return_sequences=True))
model1.add(Dropout(0.2))
model1.add(GRU(units=50))
model1.add(Dropout(0.2))
model1.add(Dense(units=1))

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 14, 50)            10500

 dropout (Dropout)           (None, 14, 50)            0

 gru_1 (GRU)                 (None, 14, 50)            15300

 dropout_1 (Dropout)         (None, 14, 50)            0

 gru_2 (GRU)                 (None, 50)                15300

 dropout_2 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 41,151
Trainable params: 41,151
Non-trainable params: 0
_________________________________________________________________
```

Para compilar usamos los mismos parámetros que en el anterior modelo: el optimizador Adam y una función de pérdida basada en el error cuadrado medio.

```
model1.compile(tf.keras.optimizers.Adam(learning_rate=0.01),
               loss=tf.keras.losses.MeanSquaredError())
```

IV. Entrenamiento del modelo

```
history1 = model1.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val) )
```

```
Epoch 1/10
51/51 [==============================] - 6s 25ms/step - loss: 0.0742 - val_loss: 4.3613e-04
Epoch 2/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0054 - val_loss: 8.8485e-04
Epoch 3/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0038 - val_loss: 1.2462e-04
Epoch 4/10
51/51 [==============================] - 1s 10ms/step - loss: 0.0026 - val_loss: 2.6028e-04
Epoch 5/10
51/51 [==============================] - 0s 10ms/step - loss: 0.0029 - val_loss: 3.4222e-04
Epoch 6/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0021 - val_loss: 0.0016
Epoch 7/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0021 - val_loss: 2.1364e-04
Epoch 8/10
51/51 [==============================] - 0s 9ms/step - loss: 0.0016 - val_loss: 0.0015
Epoch 9/10
51/51 [==============================] - 0s 10ms/step - loss: 0.0015 - val_loss: 5.9821e-04
Epoch 10/10
51/51 [==============================] - 0s 10ms/step - loss: 0.0013 - val_loss: 2.2546e-05
```

V. Predicción del modelo

```
predictions = model1.predict(X_test).flatten()
df = pd.DataFrame(data={'Predictions':predictions, 'Real': Y_test})
df
```

```
3/3 [==============================] - 1s 12ms/step
```

	Predictions	Real
0	0.406745	0.405175
1	0.410891	0.407416
2	0.410669	0.409628
3	0.413972	0.412307
4	0.415315	0.416618
...	...	...
86	0.556297	0.562142
87	0.560013	0.565312
88	0.566787	0.567353
89	0.568698	0.567986
90	0.571316	0.566204

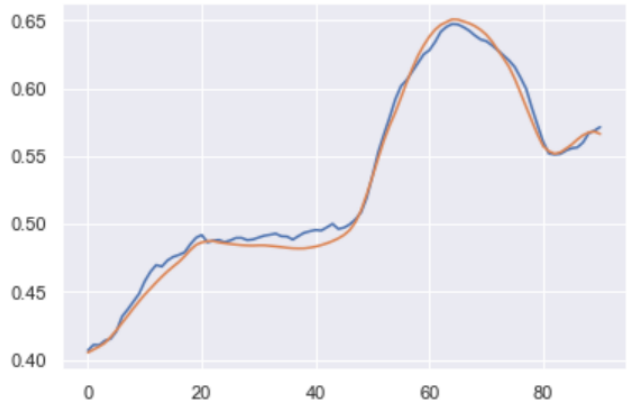
91 rows × 2 columns

A diferencia del modelo LSTM, en este caso el tiempo de ejecución es de 3 milisegundos, mientras que en el anterior fue superior a 1 segundo. Por lo que este modelo es más veloz.

VI. Representación de los resultados

```
start = 0
end = X_test.shape[0]

plt.plot(df.Predictions[start:end])
plt.plot(df.Real[start:end])
```

```
[<matplotlib.lines.Line2D at 0x26cd530c4f0>]
```



En este modelo se observa claramente que las predicciones son las mejores, por lo que el rendimiento de este modelo es prácticamente perfecto.

VII. Cálculo de métricas

```
# report performance
mse = mean_squared_error(df['Real'], df['Predictions'])
print('MSE: '+str(mse))
mae = mean_absolute_error(df['Real'], df['Predictions'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(df['Real'],df['Predictions']))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(df['Predictions'] - df['Real'])/np.abs(df['Real']))
print('MAPE: '+str(mape))
```

```
MSE: 4.13155072080609e-05
MAE: 0.005396014756681487
RMSE: 0.006427713995508893
MAPE: 0.01041329080423181
```

Los errores son prácticamente nulos. Obtenemos un error de 0.1, es decir una precisión del 99%.

6. RESULTADOS

Comparemos los 4 modelos anteriormente desarrollados en el grupo de datos en el que se ha aplicado feature engineering y en el grupo que contiene solamente el histórico de precios de Bitcoin.

Figura 8: Resumen de los resultados obtenidos

MODEL / Dataset	ARIMA	XGBOOST	LSTM	GRU
Dataset con indicadores técnicos y análisis de sentimientos	MSE: 12106086.8 MAE: 2923.815 RMSE: 3479.380 **MAPE: 0.163**	MSE: 1179253.645 MAE: 883.043 RMSE: 1085.934 **MAPE: 0.039**	MSE: 0.00019 MAE: 0.01124 RMSE: 0.0138 **MAPE: 0.021**	MSE: 4.131e-05 MAE: 0.0053 RMSE: 0.0064 **MAPE: 0.010**
Dataset del histórico de precios	MSE: 73105534.9 MAE: 7511.206 RMSE: 8550.177 **MAPE: 0.276**	MSE: 14679766.98 MAE: 3539.652 RMSE: 3831.4184 **MAPE: 0.126**	MSE: 0.00024 MAE: 0.01343 RMSE: 0.0157 **MAPE: 0.032**	MSE: 0.00017 MAE: 0.0107 RMSE: 0.013 **MAPE: 0.026**

Fuente: Notebooks desarrollados en Python

Como observamos añadir las nuevas características creadas a través de la técnica de feature engineering, se mejora considerablemente la precisión de los modelos. Las mejoras en el rendimiento de los modelos son de entre el 20%, en el caso del modelo ARIMA, y el 70%, en el caso del modelo GRU.

7. CONCLUSIONES

El mercado de las criptomonedas es conocido por sus altas ganancias y pérdidas de valor en el corto plazo, es decir, por su naturaleza volátil. También es sabido el gran impacto que el sentimiento de los distintos agentes económicos y financieros tiene sobre este mercado.

La predicción del precio de valores de mercado es un campo en el que la investigación es constante. En este proyecto estudiamos el impacto del análisis de sentimientos de noticias financieras sobre las fluctuaciones de los precios en la criptomoneda Bitcoin. Se recolectaron noticias relacionadas con la criptomoneda de la página web Coindesk y se compararon los resultados de los modelos de predicción con y sin estas nuevas características. Los modelos analizados han sido ARIMA, *Random Forest*, LSTM (*Long Short-Term Memory*) y GRU (*Gated Recurrent Units*).

Las principales contribuciones del estudio son las siguientes. En primer lugar, los precios de Bitcoin fueron predichos usando dos grupos de datos que se diferencian por la presencia o no de las características añadidas con la técnica *feature engineering*. Como resultado obtenemos una reducción de las métricas MSE, MAE, RMSE y MAPE en los modelos en los que estas características están incluidas, en comparación con los modelos en los que no están. Con ello, ha sido posible confirmar que el análisis de sentimientos de diferentes artículos relacionados con la criptomoneda afecta la predicción del precio de la misma.

En segundo lugar, gracias al estudio de los distintos modelos seleccionados, se ha demostrado que los modelos que tradicionalmente han sido utilizados para la predicción del precio de valores de mercado, véase ARIMA o *Random Forests*, tienen una menor precisión que los modelos más novedosos en este campo como son los modelos de redes neuronales recurrentes, en nuestro caso, LSTM y GRU.

En tercer lugar, el uso de una gran cantidad de noticias extraídas en un horizonte temporal de cinco años ha incrementado el rendimiento obtenido por los modelos de redes neuronales recurrentes, siendo muy superior a estudios previamente analizados los cuales no contaban con el mismo abanico de artículos.

Este proyecto revela que la incorporación de los resultados del análisis de sentimientos extraídos de las noticias financieras al modelo de predicción mejora la precisión de los mismos, implicando una relación entre los movimientos de precios de la criptomoneda y el sentimiento de los agentes financieros. Por otro lado, este estudio ofrece la posibilidad de investigar más en profundidad el impacto del análisis de

sentimientos en otros dominios como, por ejemplo, en redes sociales como Twitter (X) o eToro. También, posibilita a las corporaciones e inversores particulares identificar la relación entre los valores de mercado y el sentimiento de noticias financieras y gestionar sus carteras de inversión acorde con ello.

En resumen, este estudio demuestra que el análisis de sentimientos de noticias financieras tiene un gran impacto en la predicción de precios de la criptomoneda Bitcoin, mejorando notablemente la precisión de las proyecciones. Esto no solo confirma la relevancia del sentimiento de los agentes financieros en el mercado de las criptomonedas, sino que también resalta la pertinencia de adoptar enfoques más avanzados en su estudio, como son las redes neuronales recurrentes. Este proyecto marca un paso significativo hacia una mejor comprensión de la dinámica del mercado de criptomonedas y sus impulsores.

8. REFERENCIAS

- **A Beginner's Guide to Neural Networks and Deep Learning**. Chris V. Nicholson. Pathmind. https://wiki.pathmind.com/neural-network#define

- **A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction.** (15 de septiembre, 2021). Nan Jing, Zhao Wu y Hefei Wang. Expert Systems with Applications, Volume 178. https://www.sciencedirect.com/science/article/abs/pii/S0957417421004607

- **An Intuitive Explanation of LSTM**. (21 de febrero, 2022). Ottavio Calzone. Medium. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

- **A Sentiment and Emotion Annotated Dataset for Bitcoin Price Forecasting Based on Reddit Posts.** (2022). Pavlo Seroyizhko et al. DISI, University of Bologna. https://aclanthology.org/2022.finnlp-1.27.pdf

- **A Stacking Ensemble Deep Learning Model for Bitcoin Price Prediction Using Twitter Comments on Bitcoin.** (14 de abril, 2022). Zi Ye, Yinxu Wu, Hui Chen, Yi Pan y Qingshan Jiang. Mathematics. https://www.mdpi.com/2227-7390/10/8/1307

- **BERT for Stock Market Sentiment Analysis**. Matheus Gomes de Sousa et al. FACOM/UFMS. https://www.researchgate.net/publication/339286476_BERT_for_Stock_Market_Sentiment_Analysis

- **Bitcoin: A Peer-to-Peer Electronic Cash System**. Satoshi Nakamoto. Bitcoin.org. https://www.bitcoin.com/bitcoin.pdf

- **Can Bitcoin Kill Central Banks?** (30 de mayo, 2023). James McWhinney. Investopedia. https://www.investopedia.com/articles/investing/050715/can-bitcoin-kill-central-banks.asp

- **Cryptocurrency trading: a comprehensive survey.** (7 de febrero, 2022). Fan Fang et al. Financial Innovation 8, articulo 13. https://jfin-swufe.springeropen.com/articles/10.1186/s40854-021-00321-6

- **Documentación de librerías de Python**. Pypi.org. https://pypi.org/

- **Evaluating and understanding text-based stock price prediction models**. (31 de enero, 2014). Junqué de Fortuny, De Smedt, Martens y Daelemans. Infromation processing and management, volumen 50, issue 2, paginas 426-441. https://www.sciencedirect.com/science/article/abs/pii/S0306457313001143

- **Explained: Neural networks**. (14 de abril, 2017). Larry Hardesty. MIT News. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

- **Explicación de las Funciones de activación en Redes Neuronales y práctica con Python**. (16 de octubre, 2020). Rubiales Alberto. Medium. https://rubialesalberto.medium.com/explicaci%C3%B3n-funciones-de-activaci%C3%B3n-y-pr%C3%A1ctica-con-python-5807085c6ed3

- **Financial Sentiment Analysis using FinBERT with application in prediction stock movement.** (3 de junio 2023) Qingyun Zeng y Tingsong Jiang. https://arxiv.org/pdf/2306.02136.pdf

- **Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019.** (29 de noviembre, 2019) Omer Berat Sezera , M. Ugur Gudeleka y Ahmet Murat Ozbayoglua. Department of Computer Engineering, TOBB University of Economics and Technology. https://arxiv.org/pdf/1911.13288.pdf

- **Forecasting economic and financial time series: arima vs. Lstm**. (15 de marzo, 2018). Sima Siami Namin y Akbar Siami Namin. Texas Tech University. https://arxiv.org/ftp/arxiv/papers/1803/1803.06386.pdf

- **How can cryptocurrency reshape the global economy?** (21 de marzo, 2022). International finance. https://internationalfinance.com/magazine/banking-and-finance-magazine/how-cryptocurrency-reshape-global-economy/

- **Machine learning, explained**. (24 abril, 2021). Sara Brown. MITSloan. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

- **Natural Language Processing**. (11 de enero, 2023). DeepLearning.AI. https://www.deeplearning.ai/resources/natural-language-processing/

- **Predicting Stock Price and Market Direction using XGBoost Machine Learning Algorithm** (4 de abril, 2023). Rajandran R. MarketCalls. https://www.marketcalls.in/machine-learning/predicting-stock-price-and-market-direction-using-xgboost-machine-learning-algorithm.html

- **Quick way to find p, d and q values for ARIMA**. (24 de mayo, 2022) Yugest Verma. Mystery Vault. https://analyticsindiamag.com/quick-way-to-find-p-d-and-q-values-for-arima/

- **Sentiment-driven cryptocurrency price prediction: a machine learning approach utilizing historical data and social media sentiment análisis**. (Septiembre, 2023). Saachin Bhatt, Mustansar Ghazanfar y Mohammad Hossein Amirhosseini. MLAIJ. https://aircconline.com/abstract/mlaij/10323mlaij01.html

- S**tock market forecasting/ARIMA** (2020). Nagesh Singh Chauhan. Kaggle. https://www.kaggle.com/code/nageshsingh/stock-market-forecasting-arima/notebook

- **Stock price movement prediction based on Stocktwits investor sentiment using FinBERT and ensemble SVM**. (7 de junio, 2023) (Liu J, Leu J, Holst S). PeerJ Computer Science 9. https://peerj.com/articles/cs-1403/

- **Stock Price Prediction Using News Sentiment Analysis**. (2019). Saloni Mohan, Sahitya Mullapudi, Sudheer Sammeta, Parag Vijayvergia y David C. Anastasiu. IEEE. http://davidanastasiu.net/pdf/papers/2019-MohanMSVA-BDS-stock.pdf

- **Technical Indicator: Definition, Analyst Uses, Types and Examples**. (29 de septiembre, 2021) James Chen. Investopidia. https://www.investopedia.com/terms/t/technicalindicator.asp

- **The Ultimate Showdown: RNN vs LSTM vs GRU. Which is the Best?** (22 de mayo, 2023). Atul Harsha. Shiksha. https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/

- **Time-Series Forecasting: Predicting Stock Prices Using An ARIMA Model.** (14 de juio, 2021) Serafeim Loukas. Medium. https://medium.com/mlearning-ai/time-series-forecasting-predicting-stock-prices-using-an-arima-model-627db94590e6

- **Time Series Part 3 - Stock Price prediction using ARIMA model with Python**. (14 de mayo, 2022). Divyant Agarwal. LinkedIn. https://www.linkedin.com/pulse/time-series-part-3-stock-price-prediction-using-arima-agarwal/

- **What is bitcoin and how does it work?** Matthew Sparkes. NewScientist. https://www.newscientist.com/definition/bitcoin/

- **What is natural language processing?** IBM. https://www.ibm.com/topics/natural-language-processing

- **XGBoost for stock trend & prices prediction.** (2020). Kaggle. https://www.kaggle.com/code/mtszkw/xgboost-for-stock-trend-prices-prediction