UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Análisis y Visualización de Datos Masivos

Sistema de recogida, almacenamiento y predicción de valores de criptomonedas

Trabajo Fin de Máster

Tipo de trabajo: Desarrollo Software

Presentado por: Gil Calvo, Javier

Director/a: Calle Cordón, Álvaro

Resumen

En los últimos años, gracias a los avances en las ciencias de la computación, tanto en el apartado relativo a la mejora de hardware, como en el uso de nuevas técnicas para computar y manejar grandes cantidades de datos, numerosos ámbitos se han visto beneficiados, apareciendo aplicaciones novedosas a la hora de manejar los flujos de información generados por todo tipo de actividades productivas. Uno de estos campos que se ha visto beneficiado ha sido el de la inversión, en concreto, con la aparición de las monedas virtuales, más conocidas como criptomonedas, ha surgido la necesidad de diseñar nuevas soluciones de software que ayuden a la hora de comprar y vender estos activos, cuya demanda ha crecido enormemente en los últimos años. El objetivo de este proyecto es por tanto el diseño y construcción de un sistema que recoja, almacene y prediga el valor futuro de una criptomoneda, con el fin de servir de apoyo en la tarea de compra-venta de estos activos. Para ello, se han utilizado tecnologías big data especializadas en tratar grandes conjuntos de datos de una manera eficiente.

Palabras clave: Big data, Inteligencia Artificial, Predicción, Criptomonedas

Abstract

In recent years, thanks to advances in computer science, both in the area of hardware and in the use of new techniques for computing and handling large amounts of data, many fields have benefited, emerging many innovative applications for handling the information flows generated by all kinds of productive activities. One of these fields that has benefited has been that of investment, specifically, with the emergence of virtual currencies, better known as cryptocurrencies, the need has arisen to design new software solutions that help when buying and selling these assets, whose demand has grown enormously in recent years. The objective of this project is therefore the design and construction of a system that collects, stores and predicts the future value of a cryptocurrency, in order to support the task of buying and selling these assets. For this purpose, big data technologies specialized in processing large datasets in an efficient way have been used and implemented.

Keywords: Big data, Artificial Intelligence, Prediction, Cryptocurrencies

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

Con el aumento de las necesidades y la variedad de los problemas que encontramos tanto a nivel de usuario como a nivel empresarial, cada vez más y más proyectos se están desarrollando para solventar estas carencias presentes en el mercado. Debido a esto, la necesidad de financiación tanto a nivel privado como a nivel público se ha disparado, obligando en muchas ocasiones a los bancos centrales a producir moneda alcanzando máximos históricos en 2021. Este sistema monetario definido por la autoridad de un banco central hace también obligatoria la intermediación de este en las transacciones realizadas con dinero fiduciario.

Como alternativa, a lo largo de los años hemos visto numerosas propuestas como, por ejemplo, el sistema criptográfico monetario electrónico llamado eCash concebido por el criptógrafo David Chaum (1983)**,** o el sistema de criptodivisa descrito por la propia NSA (1996). No es hasta 2009 cuando la primera criptomoneda, que Wei Dai (Wei Dai, 1998, como se citó en BP, s.f.) define como *"dinero descentralizado que usa la criptografía como medio de control"*, es creada por el desarrollador Satoshi Nakamoto, conocida como Bitcoin.

A partir de esta, numerosas criptomonedas han sido desarrolladas como medio para evitar la intermediación de instituciones financieras y para dar soporte a infinidad de proyectos haciendo uso de la tecnología *blockchain*, que consiste en una estructura de datos formada por bloques de información, cuya finalidad es similar a la de una base de datos (Conway, 2021). Además, paralelamente se ha constituido un mercado de criptomonedas distribuido similar al mercado de valores, aumentando considerablemente el intercambio de estas y llegando a suponer un valor de mercado de 2 billones de dólares en abril de 2021. Lógicamente, este mercado presenta las mismas dificultades que el mercado tradicional, ya sea complejidad, incertidumbre o volatilidad.

Por otro lado, debido al considerable aumento en la generación de datos y a la mejora en la capacidad de computación, ámbitos como la Inteligencia Artificial o el Big Data se han visto profundamente beneficiados, presenciando un desarrollo considerable los últimos años. La utilidad de estos ámbitos recae principalmente en el procesamiento automático o semiautomático de grandes cantidades de datos, lo que sería imposible o muy costoso para el ser humano utilizando las tecnologías tradicionales.

Uno de los primeros que ha aprovechado la aparición de estas innovadoras soluciones ha sido el campo de la inversión, siendo objeto de desarrollo de todo tipo de proyectos

relacionados con la automatización de la compra-venta de acciones o en su caso, criptomonedas.

1.1 Motivación

La motivación para el desarrollo de este proyecto recae en las nuevas exigencias del mercado de valores. Como se exponía anteriormente, el mercado de criptomonedas ha llegado a representar una capitalización de mercado de 2 billones de dólares a mediados del año 2021, por otro lado, en 2021, las 20 mayores bolsas de valores por capitalización de mercado del mundo supusieron entorno a los 110 billones de dólares (Statista Research Department, 2021). Lógicamente, unido a esta capitalización bursátil, va unido un enorme número de operaciones y, por lo tanto, un gran número de agentes con diversos intereses entorno al valor de los activos presentes en las bolsas.

Debido al crecimiento en el número de operaciones realizadas, así como el número de inversores implicados en el devenir del valor de los activos financieros, la competitividad dentro del ámbito de la inversión ha sufrido también un crecimiento considerable, lo que crea la necesidad de nuevas maneras de invertir o analizar valores en bolsa. Estas nuevas maneras de invertir o analizar activos están directamente relacionadas con los nuevos avances en materia de manipulación y análisis de datos, así como en lo referente a las ciencias de la computación.

Como consecuencia, el desarrollo de soluciones de software relacionadas con la recogida, almacenamiento y predicción de valores se presenta como un tema importante dentro del mundo de la inversión, por lo que este proyecto tiene relevancia debido al contexto en el que nos encontramos.

1.2 Planteamiento del trabajo

Dado el contexto y problema encontrados, podemos sostener que una solución automatizada para la gestión de los datos y la realización de una tarea tan compleja como la compra-venta de valores, o en este caso criptomonedas, sería una solución plausible y necesaria. Es por ello que se plantea la siguiente propuesta: El diseño y desarrollo de un sistema que recoja y almacene datos del mercado de criptomonedas utilizando tecnologías

Big Data y la implementación de un modelo que prediga en tiempo real el valor de las mismas, realizando a su vez un sistema para la visualización de los resultados.

1.3 Estructura de la memoria

Para comprender el trabajo realizado, se desarrollan varios puntos en esta memoria que explican de manera detallada el proyecto en cuestión:

En primer lugar, se describe el contexto de aplicación, aportando un resumen del conocimiento que ya existe en el campo de los problemas identificados y presentando algunos de los autores de referencia en la temática del trabajo, lo que nos permite comprender mejor la investigación a tratar.

A continuación, se desarrollan los objetivos y la metodología de trabajo, lo que nos presenta un puente entre el estudio del dominio y la contribución realizada. Para ello trataremos los objetivos generales de la investigación, así como los objetivos específicos del desarrollo. Como último punto del apartado, plasmaremos la metodología del trabajo describiendo detalladamente los pasos a seguir.

Más adelante, se detalla el desarrollo de la contribución, explicando la arquitectura del sistema, describiendo en detalle las tecnologías utilizadas y la motivación de su uso, además del proceso completo de obtención, manipulación y almacenamiento del dato, el diseño e implementación del modelo predictor con lo que ello conlleva y la realización del sistema de visualización de los resultados. Posteriormente, se evalúa la usabilidad de la herramienta, así como su aplicabilidad para resolver el problema planteado.

Finalmente, se resume el proyecto realizado en una conclusión donde se valora el alcance y la relevancia de la aportación. Además, también se plantean algunas líneas de trabajo futuras que podrían dar un valor añadido al trabajo.

2. Contexto y estado del arte

Tanto el procesado de datos de criptomonedas como la predicción de valores del mercado de criptodivisas son problemas en los que se han desarrollado numerosos estudios. Debido a la presencia de diferentes requisitos y tecnologías dentro del proyecto, se han desarrollado tres apartados: plataformas big data, modelo predictor y visualización de datos. Estos tres apartados engloban las tres necesidades principales dentro de la solución de software implementada.

2.1. Plataformas Big Data

En lo relativo al uso de plataformas Big Data para la obtención y manipulación de la información en este ámbito, encontramos diversas aportaciones, por ejemplo, Horvat et al. (2020), proponen una arquitectura basada en la *Arquitectura Lambda* para el procesado y análisis en tiempo real de datos de criptomonedas. La arquitectura Lambda consiste en una estructura en la que se dispone de dos módulos, uno con información histórica almacenada y otro que recibe información en tiempo real para realizar análisis en vivo de los datos entrantes; este concepto se desarrolla más adelante cuando se trata la arquitectura de la aplicación desarrollada. Además, un aspecto a tener en cuenta sobre este estudio es el uso no solo de datos financieros de las criptomonedas analizadas, sino también de datos relacionados con eventos en redes sociales y medios de comunicación sobre las criptomonedas en cuestión. También, Rose (2018) diseña e implementa una plataforma de streaming haciendo uso de las herramientas de mensajería como Kafka, también de herramientas para el procesado de datos en tiempo real como es Spark Streaming, así como software para la visualización de datos, más en concreto Kibana. Por otro lado, Mohapatra et al. (2019), proponen un sistema de predicción en tiempo real llamado KryptoOracle, que consiste en una plataforma usando técnicas de análisis de sentimiento de tweets con una arquitectura basada en Spark que manipula un gran volumen de datos en vivo con tolerancia a los fallos. Además, el modelo que proponen hace uso de online Learning, es decir, se adapta a medida que los datos de entrada van cambiando.

2.2. Modelo predictor

En cuanto al desarrollo del modelo predictor, también existen numerosas investigaciones**.** McNally analizó el uso de dos modelos de aprendizaje profundo, una Red Neuronal Recurrente utilizando optimización bayesiana y una Red LSTM, para predecir el valor del Bitcoin. Como conclusión, la Red LSTM mostró mayor accuracy en la predicción, superando

a otros modelos analizados. Por otro lado, Shah et al. (2014) desarrollaron la regresión bayesiana como método para predecir la variación de precio del Bitcoin, que junto a una estrategia de inversión les permitió doblar la inversión inicial en menos de 60 días.

Observando los estudios presentados sobre el uso de tecnología Big Data, podemos comprobar que estos no solamente recogen información relativa al precio o volumen del mercado, sino que también a los eventos en los medios de comunicación. Además, la modularidad de las soluciones es un aspecto destacado dentro de los proyectos descritos, facilitando, según los autores, el análisis de las criptodivisas. Pese a desarrollar soluciones teniendo en cuenta eventos en medios de comunicación o redes sociales, en este proyecto solamente se tendrán en cuenta datos relativos al mercado o derivados del mismo.

Por otro lado, gran parte de los modelos utilizados para predecir valores en bolsa, o en su caso valores del mercado de las criptomonedas, se basan en Redes Neuronales Recurrentes con arquitecturas como la LSTM. Aun así, podemos encontrar otros desarrollos merecedores de consideración, como la regresión bayesiana planteada por Shah et al. (2014) o también modelos fundamentados en el Aprendizaje Reforzado. En el caso de este proyecto, debido a los datos de entrada propuestos (datos referentes al mercado), como la salida esperada (valor de la criptomoneda), se hace uso del modelo desarrollado por Facebook conocido como Prophet (Taylor S.J., 2017), que consiste en un modelo regresor que nos permite procesar secuencias completas de datos para realizar predicciones en series temporales, es decir, en la evolución de los precios de las criptodivisas. Además, el proyecto Prophet está diseñado de tal manera que el modelo pueda ser ajustado de manera intuitiva y flexible, lo que facilita en gran medida la tarea de moldear el algoritmo utilizado. Los detalles del modelo serán explicados más en profundidad en posteriores apartados de cara a entender el porqué de determinadas decisiones.

2.3. Visualización de datos

En lo relativo a la visualización de los resultados, nos podemos remitir a los proyectos mencionados con anterioridad. En el primer caso, Horvat et al. (2020), diseñan un módulo aplicación cliente que hace uso de un REST API junto a una aplicación fron-end desarrollada en javascript con un dashboard que contiene diferentes tipos de diagramas analíticos. En el paper se mencionan diferentes tipos de visualizaciones como gráficos con series temporales, histogramas o diagramas con comparaciones de precios, aunque, no se especifica ninguna herramienta en concreto. Pese a esto, es necesario resaltar el uso de javascript para la implementación de gráficos interactivos, ya que esta se presenta como una solución razonable dentro del contexto planteado. Por otra parte, Rose (2018), hace uso

de la plataforma para visualización de datos de código abierto conocida como Kibana. Para utilizar esta en su proyecto, hace uso de un contenedor de Docker, a través del que construye un entorno virtual.

3. Objetivos concretos y metodología de trabajo

Este capítulo es el puente entre el estudio del dominio y la contribución a realizar. Según el tipo concreto de trabajo, el bloque se puede organizar de distintas formas, pero los siguientes elementos deberían estar presentes con mayor o menor detalle.

A continuación, se describen los objetivos tanto generales como específicos y la metodología de trabajo llevada a cabo para desarrollar el proyecto.

3.1. Objetivo general

El objetivo del estudio consiste en comprobar la utilidad de técnicas de inteligencia artificial para el desarrollo de modelos que sirvan como apoyo a la decisión al trading en un mercado tan volátil como el de las criptomonedas, así como la influencia de las tecnologías Big Data a la hora de facilitar y sofisticar el proceso de obtención y manipulación de los datos necesarios para hacer uso de estos algoritmos.

3.2. Objetivos específicos

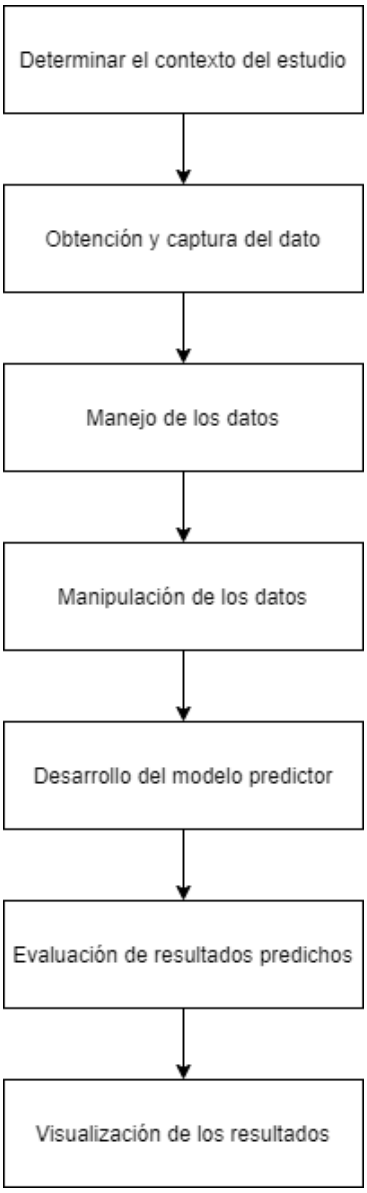
Los objetivos específicos se centran en los diferentes apartados técnicos a desarrollar para realizar el estudio en cuestión:

- Diseñar e implementar un sistema de recogida y manipulación de datos del mercado de criptomonedas.
- Construir un sistema de almacenamiento en plataforma Big Data
- Desarrollar un modelo que permita predecir el valor de una criptodivisa.
- Evaluar y recoger los resultados del modelo desarrollado.
- Implementar un sistema de visualización de los resultados generados.

3.3. Metodología del trabajo

En lo que a la metodología del trabajo se refiere, se puede identificar un flujo de trabajo lineal, que viene dado por la naturaleza tanto del proyecto como del dato capturado. Los pasos llevados a cabo en el desarrollo de la aplicación se centran en los diferentes usos que se quieren dar al dato en cada estadio de la solución.

Figura 1: *Flujo de Trabajo*



(Fuente: Elaborado por el autor)

Determinar el contexto del estudio: En este paso, se trata de identificar tanto el problema a resolver, como las herramientas o técnicas a utilizar. Como se ha mencionado en varias ocasiones, el objetivo es desarrollar una herramienta que haciendo uso de técnicas de

inteligencia artificial y Big data sirva como punto de apoyo a la hora de invertir en criptomonedas. Sabiendo esto, se puede identificar tanto los datos necesarios, las herramientas a utilizar y las técnicas necesarias para evaluar el sistema. En esta ocasión, se ha seleccionado la criptodivisa conocida como ADA, respaldada por la plataforma de blockchain Cardano, de la que se quiere predecir el valor futuro.

Obtención y captura del dato: Para ello, hay dos opciones principales: la obtención de un dataset ya confeccionado o la creación de un dataset que sirva para entrenar el modelo. Se ha considerado que construir un conjunto de datos desde cero puede servir no solo para asegurarnos de que la distribución de la que proviene la información es la misma durante el entrenamiento que durante la ejecución, sino también para elaborar los mecanismos necesarios para obtenerlo en el proceso de ejecución del sistema.

Como origen de los datos se ha optado por la API de la conocida plataforma de intercambio de criptomonedas Binance.

Por otro lado, también es necesario determinar las características de estos registros. Se valora la evolución del precio, el volumen y el número de operaciones realizadas.

Manejo de los datos: Para comunicar los diferentes módulos presentes en la aplicación, es necesaria la implementación de un sistema que permita tanto el envío como la lectura de mensajes. Además, hay que tener en cuenta que este flujo de mensajes puede ser escalable, es decir, puede requerir una tecnología diseñada para el tratamiento de información a gran escala. Por este motivo, se ha escogido el uso de la herramienta conocida como Kafka, diseñada expresamente para este cometido.

Manipulación de los datos: Una vez se dispone de un sistema que pueda comunicar los diferentes módulos desarrollados o a desarrollar, se procede a implementar el flujo de datos en tiempo real, que además permitirá manipular los datos para que tengan el formato necesario para poder ser interpretados por el modelo. Para ello, se ha escogido utilizar Apache Spark, que haciendo uso de su módulo Spark Structured Streaming, permite capturar flujos de datos en tiempo real, además, también se manipula y se transforma la información recibida en dicho flujo haciendo uso de la herramienta.

Desarrollo del modelo predictor: Teniendo en cuenta las características del problema, se ha optado por el modelo de Facebook conocido como Prophet, que permite analizar información secuencial, lo que es esencial para el contexto planteado. Para atajar este apartado, no solo se requiere de un entrenamiento, sino que también es indispensable realizar un ajuste de los hiperparámetros del proceso de entrenamiento. Posteriormente,

también se evalúa la capacidad de dicho algoritmo para predecir los valores de la criptomoneda en cuestión durante el entrenamiento.

Evaluación de los resultados: Para evaluar los resultados se trata de realizar una evaluación directa del modelo mediante una métrica como el RMSE (raíz de la media de los cuadrados del error). Esta medida permite dar mayor importancia a los errores graves, lo que resulta indispensable en la tarea de un inversor.

Visualización de los resultados: Para visualizar los resultados en tiempo real se presentan diversas posibilidades, se pueden utilizar herramientas como Tableau o PowerBI, también se puede optar por bibliotecas propias del lenguaje Python, que es el utilizado para desarrollar la aplicación. Debido a la facilidad de integración, así como la facilidad de uso, se ha escogido la biblioteca Matplotlib, que permite realizar visualizaciones en tiempo real de diferentes tipos, lo que supone una solución a las necesidades del proyecto.

4. Desarrollo específico de la contribución

En este capítulo se explica en detalle la contribución aportada, tanto los requisitos de la solución de software desarrollada, como las tecnologías utilizadas, así como de los diferentes estadios por los que ha pasado la herramienta en cuestión.

4.1. Identificación de requisitos

La identificación o captura de requisitos es el proceso por el que se especifica las actividades realizadas por el software a desarrollar, los elementos que deben componer el sistema y las restricciones a las que debe atender el mismo. Esta fase del desarrollo de software es considerada una de las más importantes a la hora de completar un proyecto con éxito, ya que establece los puntos en común entre el cliente y el desarrollador.

Para la identificación de requisitos se debe diferenciar entre requisitos funcionales y requisitos no funcionales. Los requisitos funcionales son los que describen cualquier tarea que el sistema deba realizar, por otro lado, los requisitos no funcionales se encargan de describir las características generales del sistema, así como otras limitaciones asociadas a su desarrollo.

En lo relativo a los requisitos funcionales se identifica:

- El sistema recogerá en tiempo real datos sobre la cotización de la criptomoneda conocida como ADA respecto del valor de la criptomoneda destinada a ser reflejo del dólar conocida como USDT.
- Los datos recogidos se compondrán de: valor de apertura, valor de cierre (o actual), precio máximo, precio mínimo y volumen.
- Cuando el usuario indique por la línea de comandos, los datos obtenidos en tiempo real no solo se almacenarán, si no que estos se utilizarán para predecir, también en tiempo real, el futuro valor del par ADA/USDT mediante un modelo predictor.
- Cuando el modelo prediga los futuros valores del par analizado, estos serán plasmados de manera gráfica.
- La efectividad del modelo también se podrá visualizar en tiempo real, comparando el valor predicho con el valor real.

En lo que respecta a los requisitos no funcionales se identifica:

- El software debe ser utilizable a través de un entorno virtual para asegurar la posible reproducción de los resultados y el funcionamiento en diversos sistemas.
- La herramienta desarrollada debe ser implementable en un servicio web en la nube.
- Se debe desarrollar una guía de usuario tanto para el proceso de instalación como para el uso de la herramienta en cuestión.
- El desarrollo debe ser compatible con el SO Windows.
- El software debe ser desarrollado para funcionar en PC.
- El software debe utilizar tecnologías Big Data para el manejo y gestión de los datos.

4.2. Tecnologías utilizadas

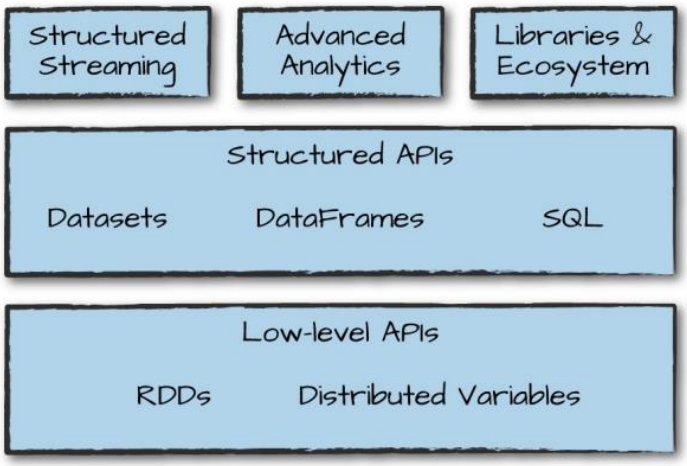
De cara a comprender las características del sistema desarrollado, así como de los motivos para la elección de las diferentes tecnologías, en este apartado se explica las diferentes herramientas utilizadas para el desarrollo de la solución de software propuesta.

4.2.1. Apache Spark

La herramienta más utilizada en el proyecto tanto para la manipulación de los datos como para el procesamiento de los mismos es Apache Spark. Este motor unificado de cálculo, junto a sus bibliotecas para el procesamiento en paralelo y distribuido de datos, nos permite realizar todas las operaciones necesarias para manejar la información con diferentes propósitos de una manera satisfactoria.

Spark está compuesto por diversos módulos con funcionalidades diferentes. El módulo *Spark Core*, que es el más importante, es el centro del framework y donde se encuentran las estructuras de datos fundamentales como los RDD, que son las únicas sobre las que sabe ejecutarse el motor. Por otro lado, encontramos los módulos inferiores que se encargan de gestionar los recursos de los clústeres sobre los que se ejecuta Spark. Finalmente, el resto de módulos realizan diversas tareas como el manejo de tablas de datos distribuidas (Spark SQL), la gestión de datos en tiempo real (Spark Structured Streaming), la implementación de algoritmos de Machine Learning (Spark MLlib) o el procesamiento de grafos (Spark GraphX).

Figura 2: *Módulos de Apache Spark*



(Fuente: Chambers et al. (2018))

En lo relativo a este proyecto, se utilizan dos módulos: Spark SQL y Spark Structured Streaming, que permiten manejar tablas de datos distribuidas, en concreto *dataframes*, además de ejecutar sentencias SQL sobre las mismas utilizando *Spark SQL* y operar de manera de manera distribuida sobre datos en tiempo real con Spark Structured Streaming.

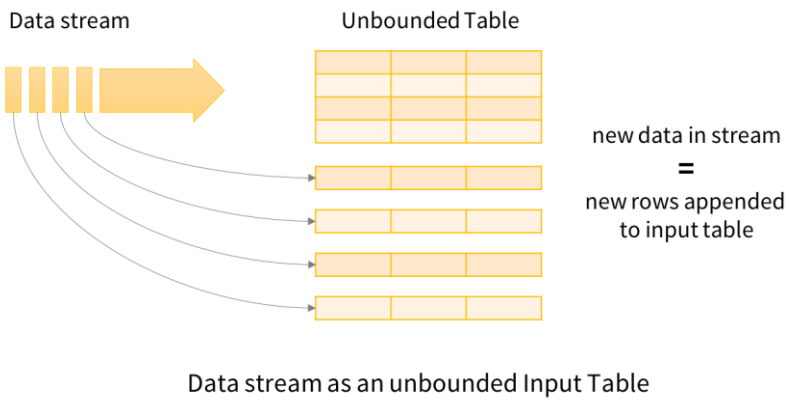
Actualmente, pese a encontrar otras opciones, Apache Spark se presenta como la alternativa más viable debido a que se trata de una plataforma de código abierto, además de su velocidad y sus múltiples módulos que permiten tratar los datos de maneras diversas sin tener que utilizar otros motores. También, otro factor a tener en cuenta ha sido la presencia de su *API* nativa para el lenguaje de programación **Python**, que es el usado en la totalidad del proyecto.

En concreto, el módulo más utilizado y quizás el más necesario en el funcionamiento del software es Spark Structured Streaming. Este módulo, que corresponde a una versión mejorada de Spark Streaming, se asegura de que haya consistencia entre los datos procesando los mismos de un modo secuencial, es decir, procesando los datos producidos uno detrás de otro. Además, también tiene una gran tolerancia a fallos y permite manejar datos antiguos, o lo que es lo mismo, datos que se han producido mucho antes de haberlos recibido, lo que en muchas ocasiones puede generar problemas dentro del sistema.

Spark Structured Streaming trata a los datos de entrada como si fueran una tabla a la que se le van añadiendo filas continuamente. En concreto, se usa una estructura de datos llamada *streaming dataframes* que funcionan como un dataframe tradicional, pero con la peculiaridad mencionada anteriormente.

Sistema de recogida, almacenamiento y predicción de valores de bolsa

Figura 3: *Streaming Dataframe*

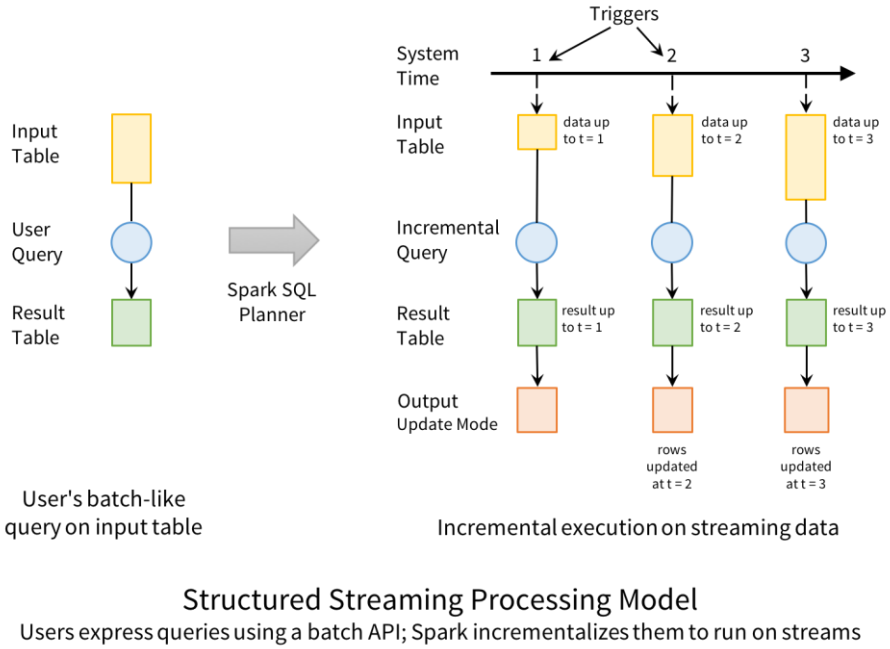


(Fuente: Zaharia et al. (2016))

Este detalle permite al sistema trabajar a mayor velocidad al hacer un cálculo a partir de los datos de entrada y no realizándolo desde cero. El sistema recibe datos en tiempo real que se van añadiendo al streaming dataframe, después, esos datos sufren una serie de transformaciones o filtros que han sido implementados previamente por el desarrollador y, por último, el resultado de estos pasos previos deriva en una tabla que, a medida que llegan datos, se va actualizando. Finalmente, es necesario guardar los resultados, ya sea en una base de datos o un sistema de almacenamiento en la nube. Para ello, el módulo proporciona una serie de posibles salidas:

- **Actualizar (Update):** solamente cambia las filas que han sido modificadas la última vez que se han obtenido datos.
- **Adjuntar (Append):** solamente adjunta al sistema de almacenamiento externo las nuevas filas añadidas al streaming dataframe.
- **Completar (Complete):** toda la tabla resultante se escribe en el sistema de almacenamiento externo.

Figura 4: *Proceso en Spark Structured Streaming*



(Fuente: Zaharia et al. (2016))

4.2.2. Apache Kafka

Para interconectar los diferentes módulos diseñados para conformar el sistema, se utiliza Kafka. Esta herramienta consiste en un bus de datos, es decir, una cola de mensajes distribuida y replicada, basada en el paradigma publicación-suscripción. Esto nos permite que las diferentes aplicaciones, en este caso módulos, publiquen o escriban mensajes relativos a los diferentes pasos por los que van pasando los datos y que, además, lean los mensajes que les son relevantes leer.

Para el uso de Kafka debemos conocer y entender algunos conceptos:

Producer: un productor consiste es un programa que, haciendo uso de la API de Kafka, envía mensajes en algún *topic* concreto.

Consumer: un consumidor es un programa que, haciendo uso de la API de Kafka, lee mensajes de los *topic* que considera relevantes.

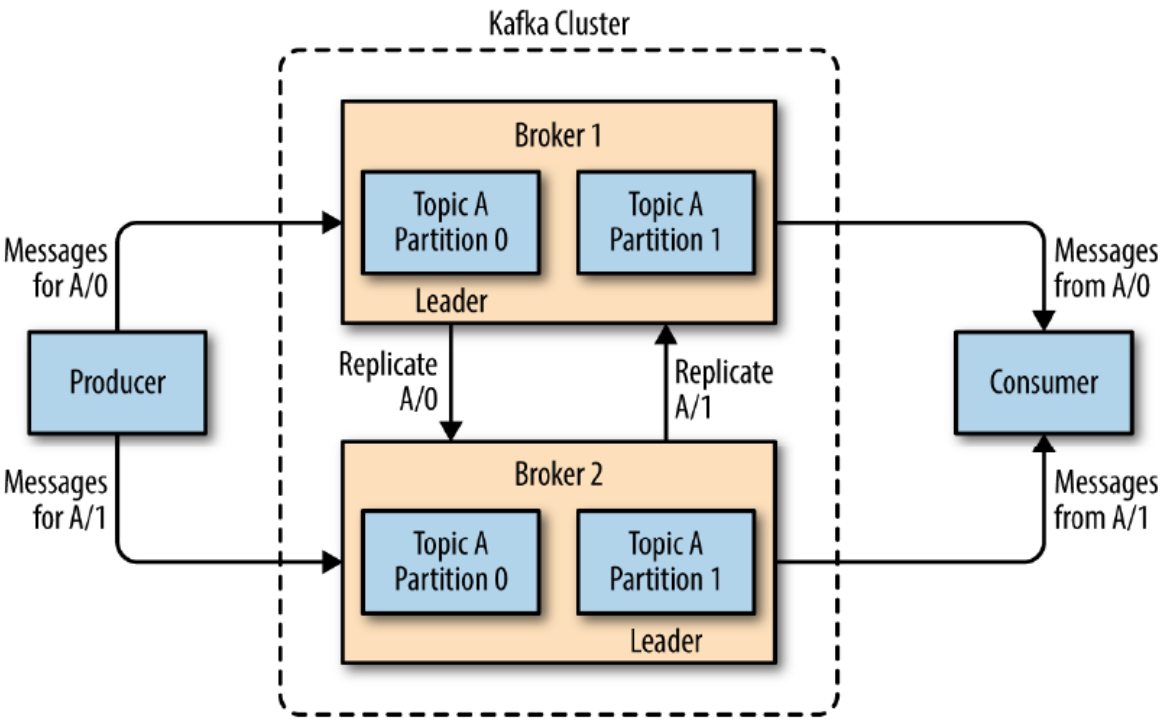
Broker: un *broker* es un nodo que conforma la arquitectura del sistema cuyo papel es el de intermediario en la lectura o escritura de mensajes.

Clúster: un *clúster* es un conjunto de *brokers*.

Topic: un *topic* es una agrupación de mensajes los cuales tienen la misma estructura y pueden o deben entenderse de la misma manera.

Partición: las particiones son las distribuciones físicas de los datos, es decir, cómo están fragmentados los *topic* en un *broker*. Cada partición puede encontrarse en uno o más *brokers* diferentes.

Figura 5: *Arquitectura Kafka*



(Fuente: Narkhede et al. (2017))

Esta herramienta nos proporciona una serie de ventajas:

- **Baja latencia:** la baja latencia es determinante en el trading algorítmico para obtener unos resultados favorables. Además, de cara al desarrollo de un sistema en tiempo real, esta característica también debe ser considerada. Esto se debe a diversos factores como las operaciones en copia cero, es decir, que no son copiadas de un área de la memoria a otra, el no acceso aleatorio de los datos o la escritura secuencial en disco.

- **Sistema distribuido:** la división del sistema en varios *brokers* que trabajan conjuntamente en un clúster aumenta la escalabilidad, la capacidad de computación y disminuye el impacto de posibles fallos.

- **Escalabilidad:** debido a que se trata de un sistema distribuido, el escalado del sistema en términos de añadir nuevos nodos no supone un problema.
- **Replicabilidad:** Kafka nos permite replicar la información que gestiona, siendo esta característica extremadamente útil para aumentar la disponibilidad, así como la tolerancia a errores.

Por estos motivos, Kafka se presenta como una buena alternativa en el proyecto a desarrollar y es la herramienta que conforma el apartado de transmisión de información entre los módulos presentes.

4.2.3. Binance API

En lo relativo a la obtención de los datos, se ha optado por la API del conocido Exchange de criptomonedas Binance. Esta interfaz de programación de aplicaciones nos permite recoger el valor de cualquier activo presente en la plataforma personalizando la consulta con los diferentes parámetros, ya sea el intervalo temporal, el par de activos concretos a valorar, el tipo de dato a consultar o incluso realizar operaciones de compra-venta. La decisión de utilizar la API de Binance es el constante mantenimiento y actualización del que goza la misma, ya que estamos hablando del Exchange más grande del mundo atendiendo al volumen de comercio (CoinMarketCap, s.f.).

Debido a la necesidad de obtener la información en tiempo real, se ha usado un websocket como herramienta complementaria. Un websocket es una tecnología que establece un canal de comunicación a través de una conexión TCP, en este caso, el objetivo es conectarse a la API para que proporcione datos actualizados sobre el par que es objeto de estudio (ADA/USDT). Además, esta conexión nos permite mantener un flujo constante de información desde la plataforma Binance hasta nuestro sistema.

4.2.4. Facebook Prophet

Como modelo predictor se han valorado diversas alternativas. Los factores que han influido en la decisión han sido la facilidad de implementación y la flexibilidad en términos de adaptación del modelo. En primer lugar, se optó por el uso de Redes Neuronales Recurrentes basadas en una arquitectura LSTM, que nos permiten analizar información secuencial, lo que se presentaba como una posibilidad a tener en cuenta, pero a pesar de ello, se descartaron por ser más inflexibles y difíciles de personalizar que otros modelos. Por

otro lado, los modelos autorregresivos integrados de promedio móvil o ARIMA estuvieron presentes en la elección inicial, pero fueron descartados por no tener en cuenta de una manera eficiente la estacionalidad, es decir, los patrones periódicos que se repiten en los datos en un periodo de tiempo determinado, que requieren los problemas relacionados con series temporales. Ambos tipos de modelos mencionados requieren de un profundo conocimiento del funcionamiento de los modelos relacionados con el análisis de series temporales. Finalmente, debido a su sencillez y a su facilidad de cara a la personalización, el modelo predictor elegido es el conocido como Prophet, desarrollado por Facebook (Taylor S.J. 2017).

Prophet es un software desarrollado para la predicción de series temporales basado en un modelo donde las tendencias no-lineales de los datos se ajustan con la estacionalidad y los efectos de otros eventos. Además, nos avisa de los posibles problemas relacionados con un mal rendimiento, lo que es bastante útil en contextos basados en el paso del tiempo y la información en tiempo real.

El modelo Prophet es un modelo diseñado a partir de unos parámetros intuitivos que pueden ser ajustados sin conocer en detalle las características internas del mismo, lo que facilita la tarea del analista o del desarrollador.

Para valorar los factores mencionados anteriormente (tendencias no-lineales, estacionalidad y otros eventos), el equipo de Facebook usa un modelo de series temporales con tres componentes: tendencia, estacionalidad y vacaciones, combinándolos de la siguiente manera:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \qquad (1)$$

Donde:

- $\boldsymbol{g(t)}$**:** la función que modela los cambios no periódicos en el valor de la serie temporal.
- $\boldsymbol{s(t)}$**:** función que modela los cambios periódicos, como por ejemplo la estacionalidad semanal o anual.
- $\boldsymbol{h(t)}$**:** función que representa los efectos de las vacaciones u otros eventos que ocurren de una manera potencialmente irregular en uno o más días.

- ϵ_t**:** cualquier cambio que no pueda ser percibido por el modelo.

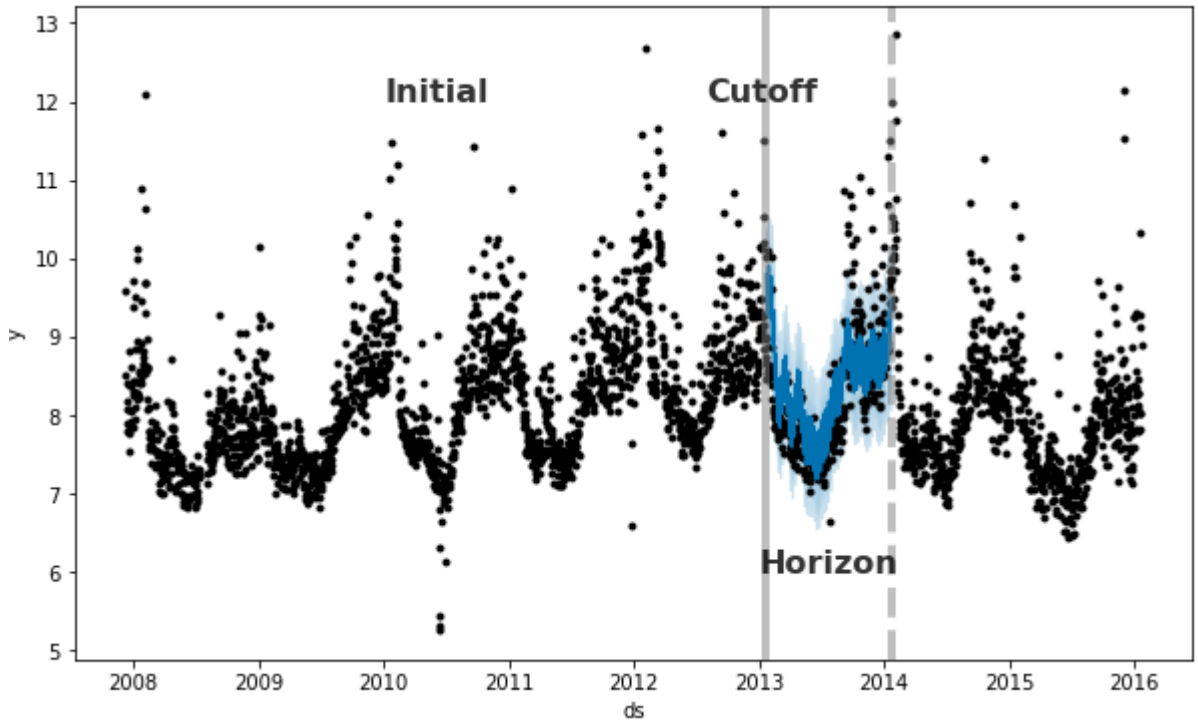
Se usa (1) para calcular el valor de la criptomoneda en cuestión en un momento t , por lo que ajustar la ecuación correctamente es determinante para obtener un resultado adecuado.

Además, el modelo presenta ciertas ventajas en lo relativo al manejo del mismo y los requerimientos que deben cumplir los datos utilizados:

- Los datos no tienen que estar espaciados regularmente y no se necesita interpolar los valores faltantes.
- El entrenamiento es rápido, lo que permite analizar diferentes parámetros en el modelo de una manera más ágil
- Los parámetros son fácilmente interpretables lo que da una mayor flexibilidad de cara a la personalización del modelo.

En lo relativo a la evaluación del modelo, no es posible utilizar métodos como el cross-validation debido a que los datos no se pueden mezclar aleatoriamente ya que se está tratando de una serie temporal y, por lo tanto, el orden de los mismos es relevante. Para solventar esto, se usa un método llamado simulated historical forecasts (SHFs) que consiste en producir K predicciones simuladas en varios intervalos de la serie temporal, formados por un origen (*cutoff*) y el tamaño del intervalo a evaluar (*horizon*). Este método se basa en la técnica conocida como *Rolling origin*, que consiste en actualizar el origen de la predicción de manera consecutiva, así como las predicciones resultantes de estos orígenes (Tashman 2000). Aplicando dicho procedimiento, se ahorra en lo relativo a la computación a la vez que se obtienen resultados con poca correlación entre sí sobre la precisión del modelo, de hecho, algunos de los riesgos derivados del uso de este procedimiento tienen que ver con la evaluación de un número alto de predicciones simuladas, lo que puede dar lugar a resultados con una alta correlación entre sí. Por el contrario, si se evalúa sobre una cantidad excesivamente pequeña de intervalos, no se obtendrán unos resultados fiables. Además, la longitud de los intervalos utilizados también es relevante, ya que, si estos son demasiado largos, la precisión obtenida será peor.

Figura 6: *Simulated Historical Forecasts (SHF)*



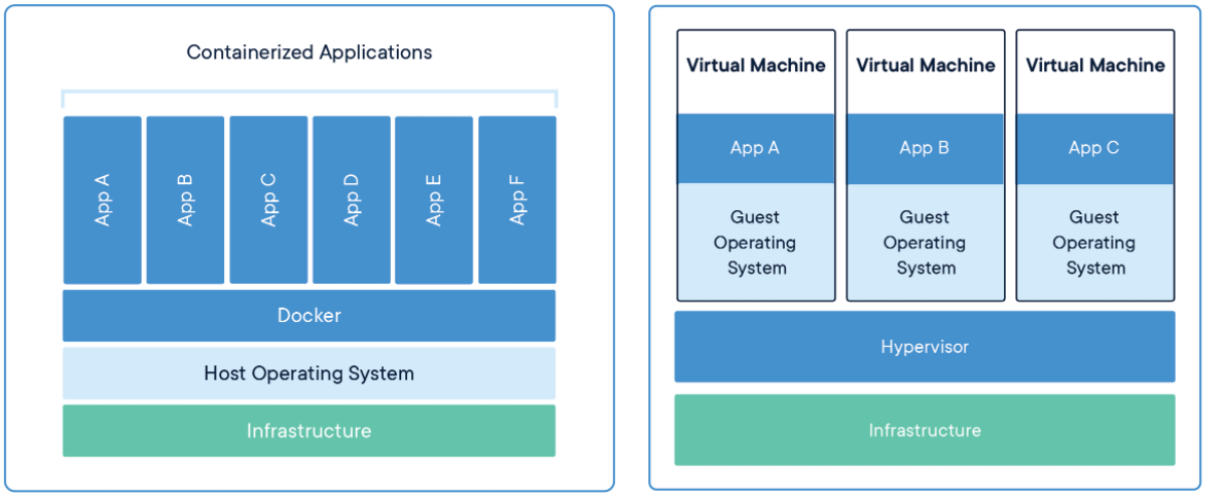
(Fuente: Facebook Prophet, s.f.)

4.2.5. Docker

Para automatizar el despliegue de la aplicación se hace uso del software de código abierto conocido como Docker. Esta herramienta permite compartir el desarrollo empaquetado en contenedores que son unidades de software que contiene el código y todas las dependencias necesarias para ejecutarlo.

A diferencia de las máquinas virtuales, los contenedores de Docker comparten el sistema operativo, lo que proporciona un ahorro sustancial en el uso de recursos.

Figura 7: *Arquitectura de Docker*



(Fuente: Docker, s.f.)

El principal motivo para utilizar esta tecnología es la necesidad de reproducir la solución desarrollada en otras máquinas, además de su facilidad de implementación, ya que tan solo requiere el motor de Docker para su ejecución, la aplicación a ejecutar y el fichero de configuración de la imagen a reproducir en el contenedor en cuestión (entorno virtual). Otra ventaja es lo ligera que es la solución, consumiendo únicamente os recursos necesarios para hacer viable el funcionamiento de la herramienta.

4.2.6. Matplotlib

En lo referente a la visualización de los resultados se hace uso de la biblioteca para la generación de gráficos para Python, Matplotlib. Su facilidad de uso y su integración con Python son los principales factores para elegir esta herramienta como la adecuada para la

representación gráfica de los resultados obtenidos con el modelo implementado. Otro aspecto a valorar es la necesidad de mostrar los datos en tiempo real, ya que, pese a haber otras soluciones que pueden ser también válidas para este propósito, esta es la que menos obstáculos presenta al no necesitar una configuración concreta en el contenedor de Docker más allá de la instalación de la biblioteca en cuestión y al existir una funcionalidad concreta en la librería para esta tarea en concreto.

Por otro lado, existen otras opciones como Tableau o PowerBI, que pueden llegar a ser utilizadas recibiendo un flujo constante de información, presentan dificultades al requerir ser reproducidas en otros equipos y no tener la posibilidad, en este caso, de desplegar el proyecto en la nube.

4.3. Descripción de la herramienta

En este apartado se describen detalladamente las características principales de los diferentes aspectos de la herramienta implementada, así como los diferentes prototipos desplegados y los problemas encontrados durante la construcción de la solución de software expuesta. Todos los pasos llevados a cabo están motivados por las necesidades intrínsecas del proyecto y, sobre todo, de los recursos de los que se dispone para desarrollar el mismo.

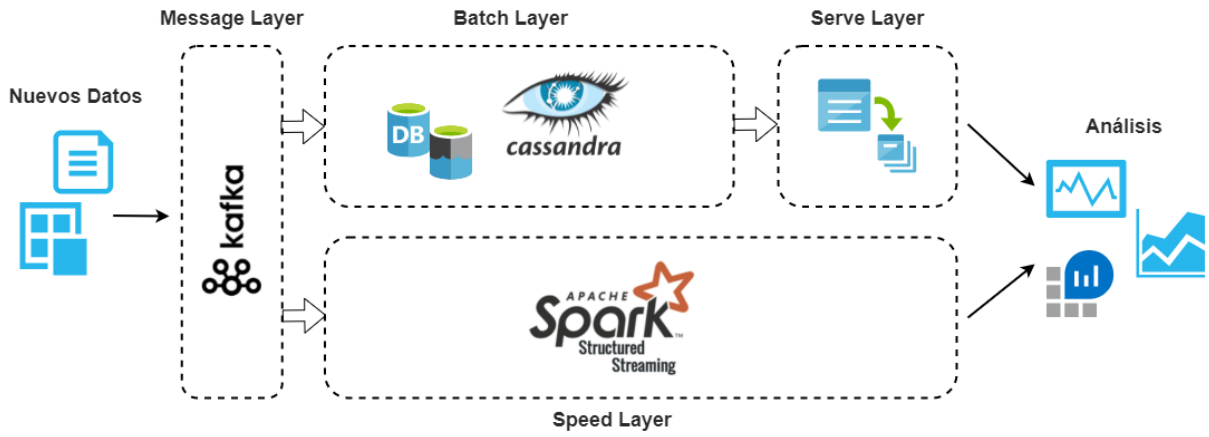
4.3.1. Arquitectura del sistema

Para explicar la arquitectura del sistema en primer lugar hay que entender las alternativas y las necesidades del proyecto.

Debido al aumento en la producción de datos, como de los nuevos casos de uso a los que se tienen que enfrentar las empresas para analizar y sacar provecho de la información generada, se han diseñado diversas arquitecturas que solventan de forma más o menos satisfactoria los obstáculos que estos nuevos contextos presentan.

La primera opción valorada es la arquitectura propuesta por Nathan Marz (2011) conocida como *Arquitectura Lambda*. La arquitectura lambda consiste en cuatro capas: la capa de mensajería, la *capa batch*, la capa de streaming o *speed layer* y la *serving layer*.

Figura 8: *Arquitectura Lambda*



(Fuente: Elaborado por el autor adaptado de Marz, N. (2011))

La capa de mensajería es la encargada en recoger la información que se presenta en un formato desestructurado o semiestructurado y transmitirla a las siguientes capas. Una tecnología muy común en esta capa es Kafka. Por otro lado, nos encontramos con la capa de streaming, que dispone de herramientas relacionadas con el procesamiento de información en tiempo real, como Spark Structured Streaming, ya que se encarga de recibir datos en streaming, procesarlos y transferirlos a la siguiente capa. A su vez, se dispone del batch layer, que es la capa en la que se almacena información a medio o largo plazo que será utilizada más adelante para realizar los análisis pertinentes. Finalmente, la serving layer se encarga de indexar la información presente en el batch layer para que pueda ser accedida en baja latencia.

La principal desventaja de esta arquitectura como el alto coste que supone manejar la información por duplicado. Pese a esto, también presenta una ventaja notable en lo referente al cambio de código o procesamiento de los datos, conocido como *reprocessing*, ya sea porque se quiere devolver nuevos campos que antes no se tenían en cuenta o porque se necesita corregir algún error. Debido a la estructura de la arquitectura lambda, estos cambios se pueden llevar a cabo de una manera relativamente sencilla.

Otra alternativa es la conocida como *Arquitectura Kappa*, cuyo término fue acuñado por Jay Kreps (2014). Esta consiste en una variación de la arquitectura lambda en la que se prescinde de la batch layer y que realiza el siguiente proceso descrito por Kreps (2014):

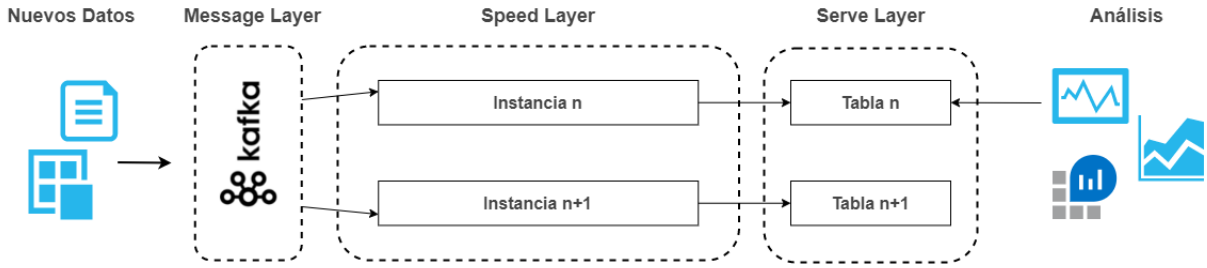
1. Para guardar la información que queremos reprocesar se utiliza Kafka u otro sistema que te permita realizar esta tarea y además permita comunicarse con múltiples suscriptores, guardando los datos durante el tiempo que necesitemos reprocesar.

2. Cuando se quiera realizar esta reprocessing, se inicia una segunda instancia del servicio de streaming que comienza a procesar la información desde el principio de los datos guardados pero que vuelque estos datos a una nueva tabla.

3. Cuando esta nueva instancia alcance temporalmente a la instancia anterior, se debe cambiar la lectura a la nueva tabla.

4. Se para la instancia antigua y se elimina la antigua tabla.

Figura 9: *Arquitectura Kappa*



(Fuente: Elaborado por el autor adaptado de Kreps, J. (2014))

Para tomar una decisión conociendo las alternativas disponibles es necesario analizar las necesidades del sistema a desarrollar, en este caso, requerimos tanto de un flujo de datos constante para realizar la predicción en tiempo real como de un histórico del valor del par ADA/USDT para reentrenar el modelo y así poder mejorar su desempeño. Además, es importante tener en cuenta el carácter del proyecto, ya que el sistema, por motivos obvios, no está activo y, por lo tanto, no es posible guardar mensajes que no han sido enviados.

Presentadas las alternativas y las necesidades del sistema, se ha realizado una adaptación del primer modelo, es decir de la arquitectura Lambda, en el que guardamos un histórico del valor del par ADA/USDT y lo actualizamos cuando se requiere entrenar el modelo. Por otro lado, también existe una capa de streaming cuyo cometido consiste en procesar la información de entrada para que el modelo pueda posteriormente realizar una predicción adecuada.

4.3.2. Prototipo 1: Binance API para la obtención de los datos.

El primer prototipo desarrollado es el relativo a la obtención de los datos desde la API de Binance. Para esta fase se tienen que diferenciar dos objetivos que son la obtención de datos en tiempo real y la obtención de datos históricos, ya que para cada tarea se requiere utilizar la API de una manera distinta.

En el caso de la obtención de datos históricos es importante conocer el funcionamiento de la interfaz. Para realizar la tarea de recopilación de información hay diversas opciones, Binance permite leer datos sobre las velas y también leer el precio medio de un símbolo, en este caso se ha optado por utilizar las velas, ya que aportan más información y, además, hay una mayor posibilidad de personalizar la consulta mediante los parámetros introducidos:

- **Símbolo:** el símbolo corresponde al par de tokens que vamos a analizar, en este caso, se analiza el par ADA/USDT.
- **Intervalo temporal:** el intervalo temporal de las velas, es decir, cuánto tiempo abarca cada vela, se ha optado por un intervalo de una hora.
- **Fecha inicial:** la fecha inicial desde la que recogemos los datos.
- **Fecha final:** la fecha final hasta la que recogemos los datos.

Figura 10: *Función de la API para obtener las velas en un periodo determinado*

```
symbol = "ADAUSDT"
start_point = "1 Sep, 2018"
end_point = "27 Aug, 2021"
interval = Client.KLINE_INTERVAL_1HOUR

klines = client.get_historical_klines(symbol, interval, start_point, end_point)
```

(Fuente: elaborado por el autor)

Por otro lado, también es relevante mencionar el formato de la información devuelta por la función en cuestión. Como resultado de la consulta, se obtiene una lista de elementos con el siguiente formato:

Figura 11: *Formato de consulta API Binance*

```
candle = [
    1620633600000,  # Open Time
    '1.78180000',  # Open
    '1.79910000',  # High
    '1.76450000',  # Low
    '1.79650000',  # Close
    '17828264.00000000',  # Volume
    1620637199999,  # Close Time
    '31809245.98575100',  # Quote asset volume
    39736,  # Number of trades
    '9140632.65000000',  # Taker buy base asset volume
    '16311599.78697900',  # Taker buy quote asset volume
    '0'  # Ignore
]
```

(Fuente: elaborado por el autor)

A continuación, a partir de esas velas obtenidas, se conforma un *DataFrame* que se define como “una estructura de datos etiquetada de dos dimensiones con columnas de diferentes tipos” (Pandas, s.f.), o lo que es lo mismo, una estructura similar a una tabla. Dicho DataFrame está compuesto por las columnas referentes a la hora de apertura de la vela en formato *Tiempo Unix*, es decir, en milisegundos desde la medianoche del 1 de enero de 1970, precio de apertura, valor máximo alcanzado, valor mínimo alcanzado, precio de cierre, hora de cierre en formato Tiempo Unix y el volumen de las operaciones realizadas en ese intervalo, el resto de campos se desechan.

Finalmente, toda la información recopilada se almacena en formato *CSV*, que se trata de un formato manejable y ordenado para guardar los datos, debido a se trata de texto plano, lo que hace a su vez más ligeros a los archivos en cuestión.

En lo relativo a la obtención de datos en tiempo real, la API de Binance nos da la posibilidad de utilizar websockets para la captura de datos en vivo. Para ello se ha optado por la instalación de una biblioteca para el lenguaje Python llamada websocket, que permite la creación de conexiones facilitando tanto el host como el puerto al que conectarse. También, se debe indicar el intervalo temporal que se quiere recoger, o lo que es lo mismo, cuánto tiempo abarca cada vela leída. El resultado de la operación es un flujo de datos correspondiente a la vela que en ese momento está teniendo lugar, con un formato tal que así:

Figura 12: *Formato websocket API Binance*

```
candle = {
    "e": "kline",  # Event type
    "E": 123456789,  # Event time
    "s": "BNBBTC",  # Symbol
    "k": {
        "t": 123400000,  # Kline start time
        "T": 123460000,  # Kline close time
        "s": "BNBBTC",  # Symbol
        "i": "1m",  # Interval
        "f": 100,  # First trade ID
        "L": 200,  # Last trade ID
        "o": "0.0010",  # Open price
        "c": "0.0020",  # Close price
        "h": "0.0025",  # High price
        "l": "0.0015",  # Low price
        "v": "1000",  # Base asset volume
        "n": 100,  # Number of trades
        "x": False,  # Is this kline closed?
        "q": "1.0000",  # Quote asset volume
        "V": "500",  # Taker buy base asset volume
        "Q": "0.500",  # Taker buy quote asset volume
        "B": "123456"  # Ignore
    }
}
```

(Fuente: elaborado por el autor)

A medida que se van recibiendo los datos, hay que tratarlos para darles un formato que sea más fácilmente manejable, desechando la información innecesaria y manteniendo la esencial. Como consecuencia, debido a que ya conocemos el símbolo a tratar y disponemos de la información de apertura y cierre de la vela, se recoge únicamente la parte

correspondiente al campo “k”, es decir, las llaves internas de la estructura de datos, de la que también se seleccionan únicamente los campos referentes al tiempo y precio de cierre y apertura, al precio máximo, al precio mínimo y al volumen.

Como resultado del prototipo, se obtienen dos procesos paralelos, en uno se recopila la información histórica cuando sea necesario reentrenar el modelo y en otro se recopilan los datos en tiempo real para realizar las predicciones pertinentes y así poder también mostrar los resultados obtenidos.

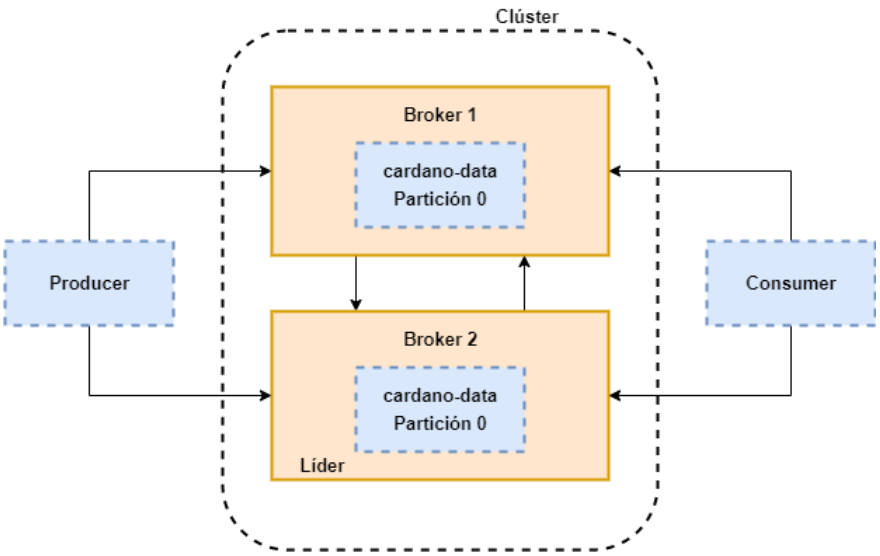
4.3.3. Prototipo 2: Comunicación con Apache Kafka

Una vez recopilados los datos y automatizada la recogida de información en tiempo real, se procede a añadir la cola de mensajes distribuida y replicada usando Kafka.

Primero, se tiene que proceder a la instalación de la herramienta, lo que se puede llevar a cabo de diferentes maneras, aunque la elegida finalmente fue desplegar un contenedor en Docker con Kafka implementado al que poder conectarse cuando se requiera publicar o leer mensajes, ya que el desarrollo se ha llevado en Windows 10 y este modo de instalación es incluso más rápido y sencillo que realizar la instalación estándar en Windows.

Partiendo de una plataforma ya desplegada, el siguiente paso es el diseño de la arquitectura que va a seguir el sistema de mensajería, por lo que hay tener en cuenta las capacidades de las que se disponen, limitaciones presentes y las necesidades el proyecto. Teniendo en cuenta estos factores, se ha decidido proceder con un esquema con dos brokers, es decir, un factor de replicación igual a dos y una sola partición por broker.

Figura 13: *Arquitectura topic creado*



(Fuente: elaborado por el autor)

Como se ha mencionado, se quiere construir el entorno Kafka utilizando Docker y, por lo tanto, es necesario diseñar un archivo docker-compose con las características que se quiere que tenga el contenedor:

Figura 14: *Archivo docker-compose*

```
version: '3'
services:
  zookeeper-1:
    image: confluentinc/cp-zookeeper:5.5.0
    hostname: zookeeper-1
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_LOG4J_ROOT_LOGLEVEL: INFO
      # ZOOKEEPER_LOG4J_LOGGERS: "INFO,CONSOLE,ROLLINGFILE" -- No Include
    ports:
      - "2181:2181"

  kafka-1:
    image: confluentinc/cp-kafka:5.5.0
    hostname: kafka-1
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181"
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka-1:29092,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_LOG4J_ROOT_LOGLEVEL: INFO
      # KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"
      - "29092:29092"
    depends_on:
      - zookeeper-1
  kafka-2:
    image: confluentinc/cp-kafka:5.5.0
    hostname: kafka-2
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181"
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka-2:29093,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_LOG4J_ROOT_LOGLEVEL: INFO
      # KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9093:9093"
      - "29093:29093"
    depends_on:
      - zookeeper-1
```

(Fuente: elaborado por el autor)

A continuación, se ejecuta el comando docker-compose que construye un contenedor a partir de un archivo de configuración como el mostrado previamente:

Figura 15: *Ejecución de docker-compose*

```
PS D:\Master2020UNIR\TFM\PROYECTO_PYCHARM> docker-compose -f docker-compose.yml up -d
```

(Fuente: elaborado por el autor)

Una vez preparado el entorno, se procede a probar la plataforma y a crear el topic en cuestión. Para ello, se debe utilizar el comando *kafka-topic* indicando el host y puerto para acceder al mismo, el factor de replicación, el número de particiones y el nombre del topic en cuestión:

Figura 16: *Creación del topic*

```
# kafka-topics --create --bootstrap-server localhost:29092 --replication-factor 2 --partitions 1 --topic cardano-data
Created topic cardano-data.
```

(Fuente: elaborado por el autor)

Se puede comprobar qué topics hay creados para verificar que se ha hecho correctamente:

Figura 17: *Verificación de topics*

```
# kafka-topics --list --bootstrap-server localhost:29092
__confluent.support.metrics
cardano-data
```

(Fuente: elaborado por el autor)

Además, se pueden comprobar los detalles del mismo:

Figura 18: *Detalles del topic*

```
# kafka-topics --describe --bootstrap-server localhost:29092 --topic cardano-data
Topic: cardano-data     PartitionCount: 1       ReplicationFactor: 2    Configs:
        Topic: cardano-data     Partition: 0    Leader: 2       Replicas: 2,1   Isr: 2,1
```

(Fuente: elaborado por el autor)

En este caso, se pueden ver los siguientes detalles:

- El nombre del topic (cardano-data), que tiene solamente una partición y un factor de replicación igual a 2.
- Que el id del broker líder es igual a 1 y el de la partición es igual a 0.
- Por otro lado, los brokers que disponen de las réplicas del topic son los que tienen el id igual a 2 y a 1.

en la aplicación haciendo uso de la biblioteca kafka-python para Python, que posibilita el uso de Kafka de manera programática, es decir, mediante código.

La primera implementación llevada a cabo corresponde con la publicación de los datos recibidos por el websocket a través de un Producer y recibida a través de un Consumer. Como prueba, simplemente se devuelven por pantalla estos valores para comprobar que el

sistema funciona correctamente y que posteriormente va a poder ser utilizado por Spark Structured Streaming.

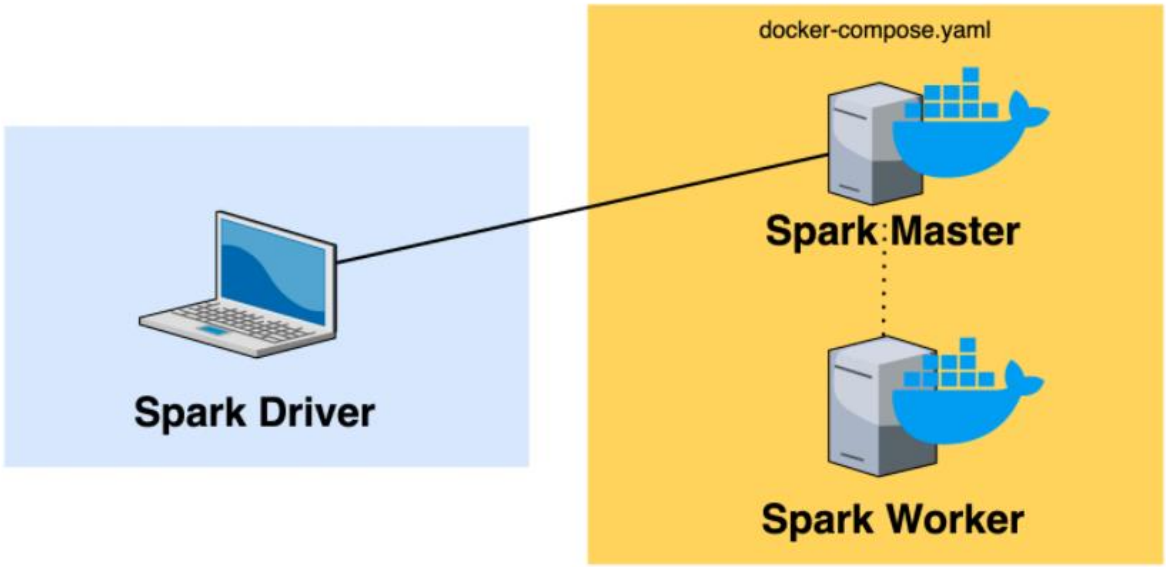
4.3.4. Prototipo 3: Datos en streaming

A continuación, se debe implementar el flujo de datos en streaming usando la herramienta expuesta con anterioridad, Spark Structured Streaming, que está exclusivamente diseñada para dicho propósito.

Para poder proceder, al igual que Kafka, es necesario realizar una instalación y configuración previa, en este caso, se ha escogido instalar el framework en Windows, ya que, pese a existir la posibilidad de ser implementado en Docker, para realizar pruebas con el mismo es más cómodo instalarlo directamente en el equipo. Este proceso, pese a parecer un paso sencillo en un principio, presenta ciertas dificultades debido a la carencia de multitud de los requisitos necesarios para hacer funcionar Spark Structured Streaming en la API de Python desde la página oficial de Apache Spark, de hecho, la instalación de la herramienta ha supuesto un obstáculo en el proyecto, ya que la propia fundación no proporciona algunas de las dependencias necesarias en Windows para utilizarla. Como es lógico, la principal pretensión de la organización Apache es fomentar el software de licencia abierta, por lo que el soporte y facilidad de instalación en Linux es lógicamente mayor en comparación al del sistema operativo de Microsoft.

En lo relativo a la relación de los diferentes nodos Spark, se define en el fichero de configuración docker-compose, que determina las características de los contenedores ejecutados. Para ello, se definen tres nodos: el *Spark Driver*, el *Spark Master* y el *Spark Worker*.

Figura 19: *Arquitectura Spark*



(Fuente: Synkov D. (2021))

Spark Driver: El *Driver* es el proceso en el que se ejecuta el código del programa, que se ejecuta en una máquina concreta (en este caso el equipo local) y que se conecta al Cluster Manager (Spark Master).

Spark Master: Es el gestor del clúster de Spark, que controla a los *workers* que realizan las tareas necesarias. Al crear un *SparkSession* se requiere indicar la dirección IP y el puerto del clúster configurado.

Spark Worker: Estos nodos se encargan de realizar las tareas y el trabajo del clúster en cuestión.

Todas las características mencionadas (master, driver…) deben ser indicadas en el SparkSession, que establece una comunicación entre el driver y el gestor del clúster:

Figura 20: *Creación del SparkSession*

```
def get_spark_context(app_name: str) -> SparkSession:
    conf = SparkConf()

    conf.setAll(
        [
            (
                "spark.master",
                os.environ.get("SPARK_MASTER_URL", "spark://spark-master:7077"),
            ),
            ("spark.driver.host", "localhost"),
            ("spark.submit.deployMode", "client"),
            ('spark.ui.showConsoleProgress', 'true'),
            ("spark.driver.bindAddress", "0.0.0.0"),
            ("spark.app.name", app_name)
        ]
    )
    print(conf.getAll())
    spark_session = SparkSession.builder.config(conf=conf).getOrCreate()
    spark_session.sql("set spark.sql.caseSensitive=true")
    spark_session.sql("set spark.sql.codegen.wholeStage= false")
    return spark_session
```

(Fuente: elaborado por el autor)

Con la herramienta instalada, hay que realizar también la instalación de la biblioteca de Python para poder implementar en código el funcionamiento de esta, en concreto se ha instalado la biblioteca PySpark para el uso de Spark en el lenguaje mencionado.

Posteriormente, para unir los dos anteriores prototipos con el desarrollado a continuación se aprovecha una opción presente en Spark Streaming que consiste en recoger directamente la información usando como fuente la cola de mensajes de Kafka, es decir, se conecta el módulo de mensajería con el encargado de recoger los datos en tiempo real y procesar dicha información. Para ello, se crea un objeto stream reader, cuya función consiste en leer los datos desde los parámetros indicados:

Figura 21: *Stream Reader*

```
candle_streaming = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:29092") \
    .option("subscribe", "cardano-data") \
    .option("startingOffsets", "latest") \
    .load()
```

(Fuente: elaborado por el autor)

Posteriormente, también es necesario indicar cuál es la estructura de los datos, es decir, qué esquema siguen y de qué tipo son. Para eso, se crea un objeto *StructType* con los campos en cuestión y su tipo:

Figura 22: *Esquema de los datos*

```
esquema = StructType([
    StructField("t", StringType()),
    StructField("o", StringType()),
    StructField("h", StringType()),
    StructField("l", StringType()),
    StructField("c", StringType()),
    StructField("T", StringType()),
    StructField("v", StringType()),
    StructField("n", IntegerType())
])
```

(Fuente: elaborado por el autor)

Después, con los datos recogidos, se realizan una serie de transformaciones relativas a la creación de las nuevas columnas, el casting de los datos que se están recogiendo y la selección de las columnas a mostrar:

Figura 23: *Transformación datos de entrada*

```
parsedDF = candle_streaming \
    .select("value") \
    .withColumn("value", F.col("value").cast(StringType())) \
    .withColumn("value", F.translate(F.col("value"), "\\", "")) \
    .withColumn("vela", F.from_json(F.col("value"), esquema)) \
    .withColumn("OpenTime", F.col("vela.t")) \
    .withColumn("Open", F.col("vela.o").cast(DoubleType())) \
    .withColumn("High", F.col("vela.h").cast(DoubleType())) \
    .withColumn("Low", F.col("vela.l").cast(DoubleType())) \
    .withColumn("Close", F.col("vela.c").cast(DoubleType())) \
    .withColumn("CloseTime", F.col("vela.T")) \
    .withColumn("Volume", F.col("vela.v").cast(DoubleType())) \
    .withColumn("NumberOfTrades", F.col("vela.n").cast(IntegerType())) \
    .select("OpenTime", "Open", "High", "Low", "Close", "Volume", "CloseTime", "NumberOfTrades")
```

(Fuente: elaborado por el autor)

Una vez implementadas las transformaciones intermedias que sufren los datos, hay que indicar cómo se va a guardar o qué salida se les va a dar a los datos recibidos y transformados. Para este cometido hay diversas opciones, se puede devolver el resultado en un csv, se puede enviar por Kafka o se puede tratar programáticamente como decida el desarrollador, que es la alternativa por la que se ha optado, ya que es necesario tratar los datos de una manera diferente con la predicción en tiempo real:

Figura 24: *Salida del streaming pipeline*

```
consoleOutput = parsedDF \
    .writeStream \
    .foreachBatch(foreach_batch_function) \
    .start() \
    .awaitTermination()
```

(Fuente: elaborado por el autor)

Para agilizar el proceso, simplemente se almacena en una variable la última vela guardada y la vela que actualmente se está analizando, es decir, el último batch devuelto por spark streaming. Luego, se compara el dato obtenido con la última vela devuelta por el stream y con la última vela guardada, así se puede saber si la vela se ha cerrado y estamos tratando con una vela nueva, para poder guardar los datos en el csv:

```
def foreach_batch_function(df, epoch_id):
    global last_historical_data_row
    global last_current_data_row

    if len(df.tail(1)) > 0:

        current_time = df.tail(1)[0]['OpenTime']
        last_current_time = last_current_data_row['OpenTime']
        historical_time = last_historical_data_row['OpenTime']

        if current_time != last_current_time:
            if historical_time != last_current_time:
                # DEBEMOS GUARDAR EL MODELO!!!
                print("Guardamos en el csv y reentrenamos el modelo")
                last_historical_data_row = last_current_data_row

            last_current_data_row = df.tail(1)[0]
        else:
            last_current_data_row = df.tail(1)[0]
```

Figura 25: Tratamiento de los datos (Fuente: elaborado por el autor)

4.3.5. Prototipo 4: Implementación del modelo predictor

Con el flujo de datos ya disponible y el histórico del precio del par ADA/USDT almacenado, se debe proceder a implementar el modelo predictor que servirá de apoyo al experto en trading. Este proceso consta de cuatro pasos: ajuste de hiperparámetros, entrenamiento, evaluación y almacenamiento del modelo.

En lo relativo al ajuste de hiperparámetros, en primer lugar, hay que definir qué son los hiperparámetros y cuáles son los más relevantes en el entrenamiento del modelo que se está tratando. Los hiperparámetros son todos los parámetros que determinan el progreso del proceso de entrenamiento del modelo. Como mencionan Marc Claesen y Bart De Moor (2015), normalmente se tratan de valores continuos o enteros, lo que da lugar a problemas de optimización distintos. La búsqueda de los hiperparámetros idóneos es un tema candente en el ámbito del aprendizaje automático, ya que el entrenamiento y, por lo tanto, el desempeño de los modelos desarrollados, cambia completamente dependiendo de qué valores se han seleccionado previamente.

Para ajustar los hiperparámetros de manera que la función de pérdida utilizada tenga un valor mínimo, se ha utilizado el método de evaluación conocido como cross-validation, que

también se usará posteriormente para valorar la eficacia del modelo. La función de error utilizada como referencia es la correspondiente a la Raíz del Error Cuadrático Medio, en adelante, RMSE por sus siglas en inglés:

$$\boldsymbol{RMSE} = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \frac{(\hat{\boldsymbol{y}}\boldsymbol{i}-\boldsymbol{y_i})\boldsymbol{2}}{\boldsymbol{n}}} \quad (2)$$

En la que:

- Yi: al valor real.
- Yi: valor predicho por el modelo.
- n: número de elementos, es decir, número de velas predichas por el modelo.

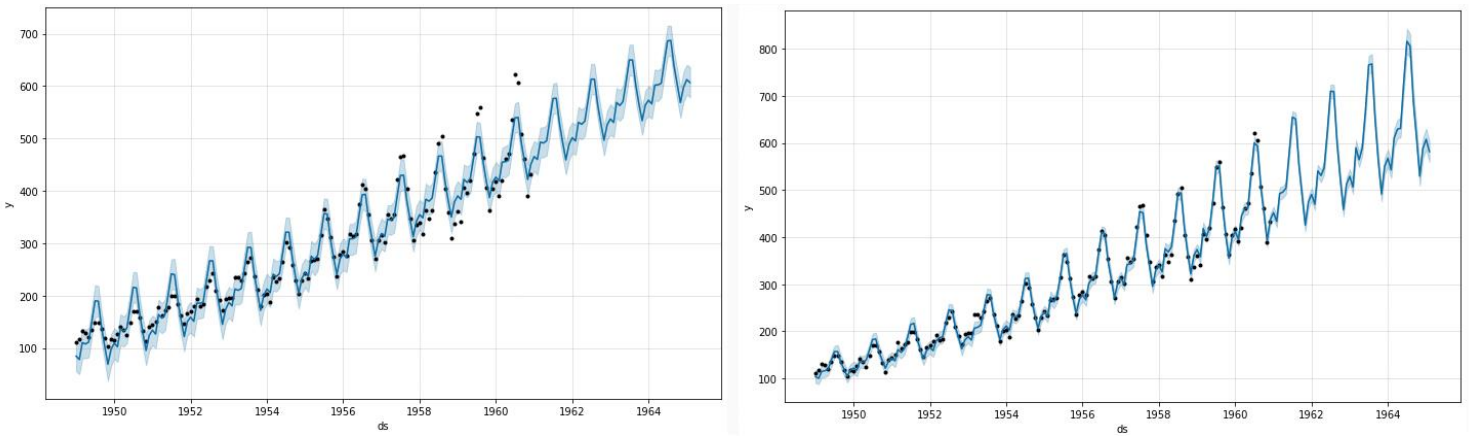
El motivo para elegir (2) es que esta da un mayor valor a los errores de mayor tamaño, ya que se trata de la raíz de la media de los errores cuadrados, por lo que es más sensible a outliers. Este enfoque está relacionado con el hecho de que un valor predicho muy alejado de la realidad puede ser extremadamente perjudicial de cara a una toma de decisiones, así que, dando una mayor importancia a valores desproporcionados aun limpiando el dataset, puede dar mayor fiabilidad al modelo implementado.

Para englobar todos los hiperparámetros sobre los que se quiere moldear el proceso de entrenamiento y, en consecuencia, el desempeño del modelo, se construye un diccionario, que consiste en un conjunto de pares clave-valor o, en este caso, nombre del hiperparámetro y valores posibles del hiperparámetro. Lógicamente, es importante seleccionar los hiperparámetros adecuados para este proceso, ya que no todos tienen la misma utilidad, ni incidencia en el entrenamiento del algoritmo. Los seleccionados para este proceso son:

- **Changepoint prior scale:** determina la flexibilidad de la tendencia y cuánto cambia esta en los puntos de cambio, es decir, en puntos en los que el elemento analizado ha sufrido un cambio relevante.
- **Seasonality prior scale:** determina la flexibilidad de la estacionalidad. Un valor alto permite a la estacionalidad ajustarse a fluctuaciones largas y, por el contrario, un valor pequeño disminuye la dimensión de la estacionalidad. El valor por defecto es 10, pero se recomienda que esté en el rango [0.01, 10].

- **Seasonality mode:** corresponde al tipo de estacionalidad que tienen los datos analizados. Puede tomar dos valores: 'additive' y 'multiplicative'. Una serie temporal con una estacionalidad aditiva es la que sus componentes se suman, es decir, que la estacionalidad se suma a la tendencia. Por otro lado, una serie temporal con una estacionalidad multiplicativa es la que la magnitud de patrón encontrado en la estacionalidad depende de la magnitud de la serie temporal. Por ejemplo:

Figura 26: *Series temporales aditivas y multiplicativas*



(Fuente: Facebook Prophet (s.f.))

En esta imagen, se puede ver cómo el modelo Prophet asume una estacionalidad aditiva en la gráfica de la izquierda, el patrón estacional no se ve afectado por la magnitud de la serie temporal, pero no se ajusta a la realidad de los datos. Por otro lado, a la derecha se puede ver cómo a medida que aumenta el tamaño de la serie temporal, también aumenta el tamaño del patrón estacional, o lo que es lo mismo, Prophet está asumiendo una estacionalidad multiplicativa.

- **Daily/Weekly/Yearly seasonality:** Estos hiperparámetros indican al modelo si existe un patrón estacional en el intervalo diario, semanal o anual. Los posibles valores son Verdadero o Falso.

A partir de esta estructura de datos mencionada previamente se genera una lista con todas las posibles combinaciones de los hiperparámetros a valorar y, a continuación, se entrena y evalúa (mediante cross-validation) al modelo tantas veces como combinaciones se encuentren. Una vez terminado el proceso, se selecciona la combinación que haya obtenido un menor valor en la RMSE, es decir, se selecciona la combinación que genera un modelo más efectivo. Los hiperparámetros que han resultado en un mejor rendimiento del algoritmo son:

Tabla 1. *Resultados del ajuste de hiperparámetros*

CPS (changepoint_prior_scale)	**SPS (seasonality_prior_scale)**	**seasonality_mode**	**D. Seas. (daily)**	**W. Seas. (weekly)**	**Y. Seas. (yearly)**	**RMSE**
0.001111	11.111	'multiplicative'	40	40	40	0.350703
0.001111	11.111	'multiplicative'	False	1	True	0.351542
0.001111	11.111	'multiplicative'	5	15	True	0.352794
0.001111	11.111	'multiplicative'	True	True	True	0.353561
0.001111	11.111	'multiplicative'	15	10	True	0.353675

(Fuente: elaborado por el autor)

Conociendo los parámetros adecuados para el entrenamiento, se debe proceder a entrenar el modelo en cuestión, lo que presenta otras dificultades relacionadas con el *overfitting* que Webb (2011) define de la siguiente manera: "Un modelo sobreentrena respecto de los datos de entrenamiento cuando describe atributos que aparecen desde el ruido o la varianza de los datos, más que de la distribución inherente de los mismos".

El overfitting es un fenómeno que aparece en la fase de entrenamiento y que da lugar a la pérdida de precisión del modelo en muestras que están fuera del conjunto de entrenamiento. Además, este fenómeno puede dar lugar a una falsa sensación de que nuestro modelo rinde adecuadamente, ya que un síntoma claro de que se está incurriendo en el sobreentrenamiento es que el rendimiento del algoritmo para los datos de entrenamiento es muy bueno, pero cuando se le muestra nuevos datos, este obtiene unos resultados mucho peores. Existen diversos motivos por los que este fenómeno puede aparecer, algunos de los más comunes son la excesiva iteración sobre los datos de entrenamiento o el uso de modelos sobredimensionados en problemas que presentan una menor exigencia.

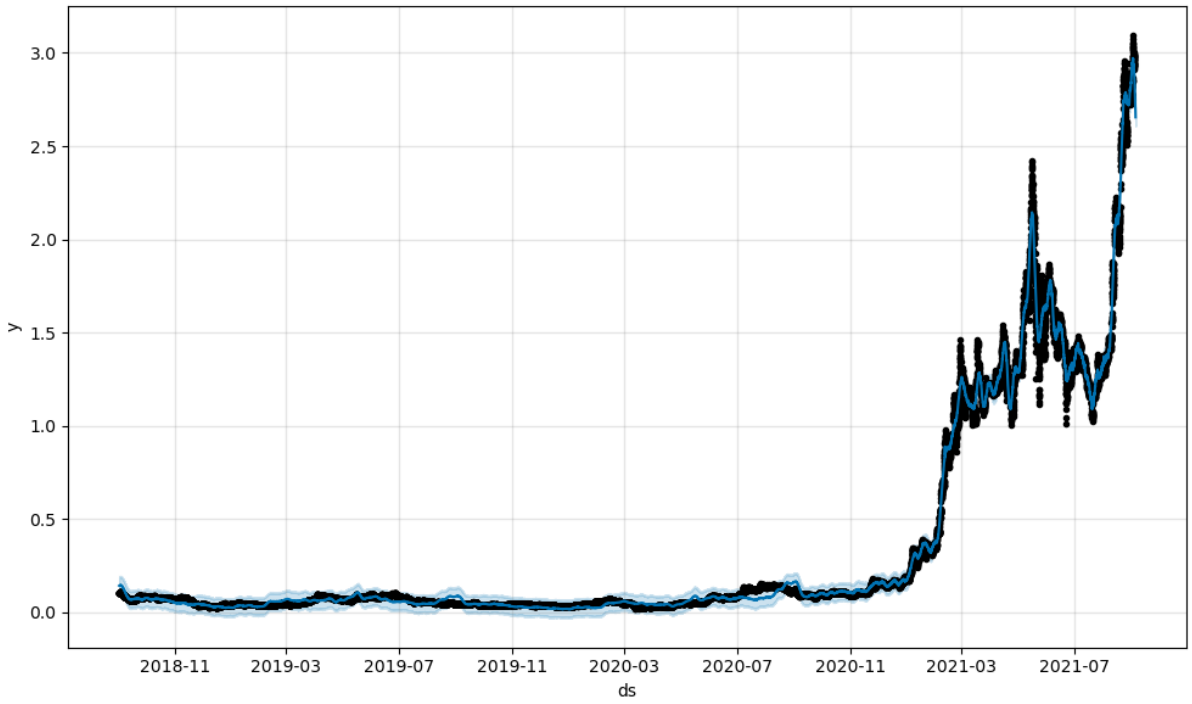
Una vez realizado y optimizado el proceso de entrenamiento, el siguiente paso consiste en conocer la efectividad del modelo mediante la evaluación del mismo. Como se ha mencionado con anterioridad, el método utilizado para comprobar la habilidad del predictor es el conocido como *cross-validation*, que consiste en un método estadístico para evaluar modelos fragmentando los datos en segmentos y realizando tantos procesos de entrenamiento y evaluación como fragmentos haya, de modo que, en cada iteración un fragmento distinto es utilizado para evaluar mientras que los restantes se usan para entrenar

y finalmente se realiza una media entre los resultados obtenidos (Refaeilzadeh P., Tang L., Liu H., 2009).

Este proceso no solamente da una idea de cuán bueno es el modelo, sino que también ayuda a destapar otros problemas como el del sobreentrenamiento. La importancia de conocer la efectividad del algoritmo recae en la necesidad del experto de saber con qué confianza puede tener en cuenta el valor predicho y, por lo tanto, tomar una decisión con mayor o menor fundamento.

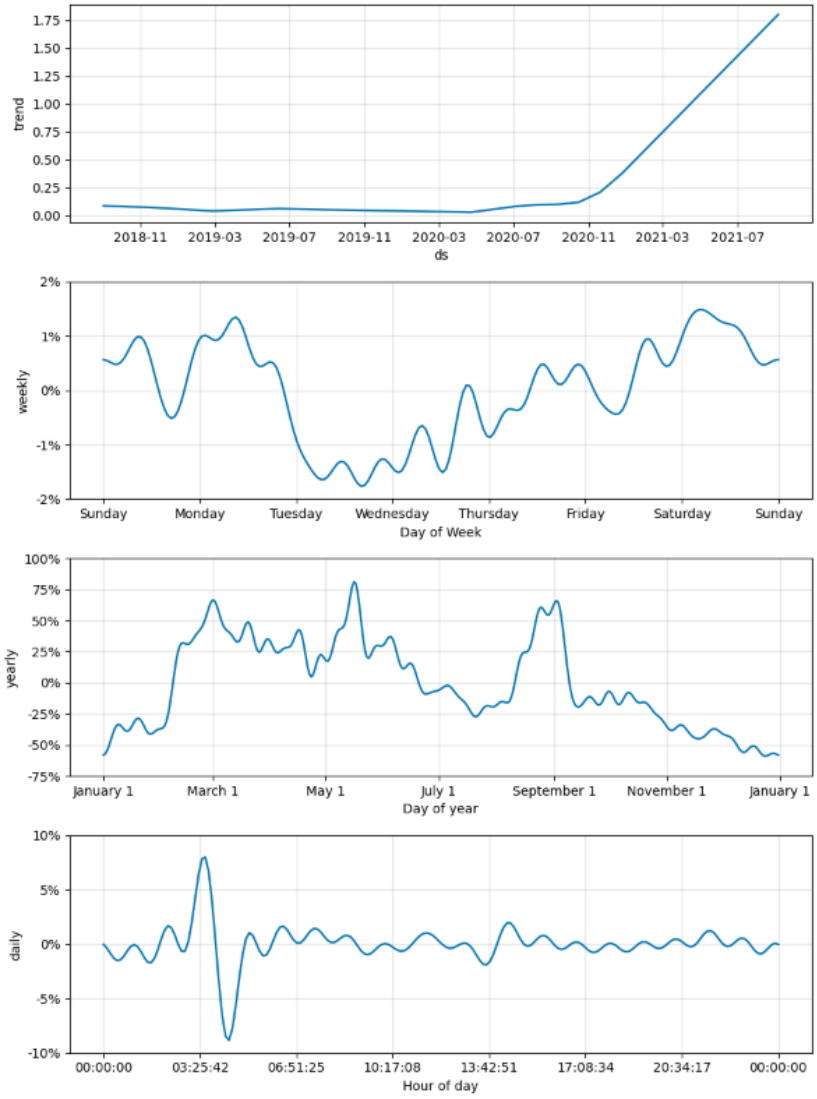
Durante la evaluación del modelo se han obtenido los siguientes resultados:

Figura 27: *Resultado del entrenamiento*



(Fuente: elaborado por el autor)

Figura 28: *Estacionalidades encontradas en el entrenamiento*



(Fuente: elaborado por el autor)

Finalmente, se debe almacenar el modelo entrenado para poder usarlo más adelante. Lógicamente, Facebook Prophet dispone de un modo para salvar los algoritmos desarrollados, que difiere del método estándar utilizado para esta tarea. Normalmente, la manera más rápida de almacenar un algoritmo de machine Learning es haciendo uso de la biblioteca conocida como *Pickle*, que consiste en un modelo cuya finalidad es serializar y decodificar objetos en Python. En este caso, se utilizan los métodos propios disponibles en la biblioteca Prophet para esta misma tarea, serializando el modelo en un objeto json, ya que el uso de Pickle puede dar lugar a errores en ciertas versiones de Python.

Una vez guardado el predictor, se debe integrar este mismo en el sistema ya implementado. Para ello, simplemente se debe cargar el algoritmo antes de utilizarlo, a través de las funciones disponibles en el software de Facebook para realizar el proceso contrario a la serialización. Después, simplemente se debe transmitir una fecha que tenga el mismo formato que los datos de entrenamiento y el modelo devolverá el valor de cierre de los siguientes veinticuatro periodos, es decir, de las siguientes veinticuatro horas, junto con el intervalo de posibles valores que puede tomar el activo en cuestión:

Figura 29: *Resultado predicción*

	ds	yhat	yhat_lower	yhat_upper
2021-09-05	07:59:59.999	2.884483	2.844309	2.925108
2021-09-05	08:59:59.999	2.874365	2.836635	2.916985
2021-09-05	09:59:59.999	2.872622	2.832930	2.915311
2021-09-05	10:59:59.999	2.868908	2.823384	2.911505
2021-09-05	11:59:59.999	2.861377	2.817413	2.902871

(Fuente: elaborado por el autor)

4.3.6. Prototipo 5: Visualización de los resultados

Se ha implementado el proceso referente a la obtención los datos utilizando la API de Binance, también se ha conectado esta con Kafka para gestionar el envío y lectura de información de una manera más selectiva y distribuida. Más tarde, utilizando Spark Structured Streaming se ha conseguido que los datos provenientes de Binance, que a su vez coordina Kafka, sean recogidos y gestionados, realizando transformaciones en la información para su posterior manipulación y permitiendo construir un flujo de datos en tiempo real. Además, se ha desplegado un modelo predictor utilizando el software Facebook Prophet, diseñado exclusivamente para el análisis de series temporales, que predice con relativa efectividad la tendencia de los datos. Finalmente, en este apartado se explican los detalles de la implementación del módulo encargado de gestionar la visualización de los resultados en vivo.

La principal tecnología utilizada para el propósito a exponer es la biblioteca para el lenguaje de programación Python conocida como Matplotlib. Esta biblioteca permite la creación de visualizaciones tanto estáticas como animadas, así como la posibilidad de interactuar con las mismas.

En primer lugar, es necesario analizar qué información se quiere mostrar y con qué intención, es decir, hay que tener en cuenta cuál es el objetivo de la herramienta y en qué

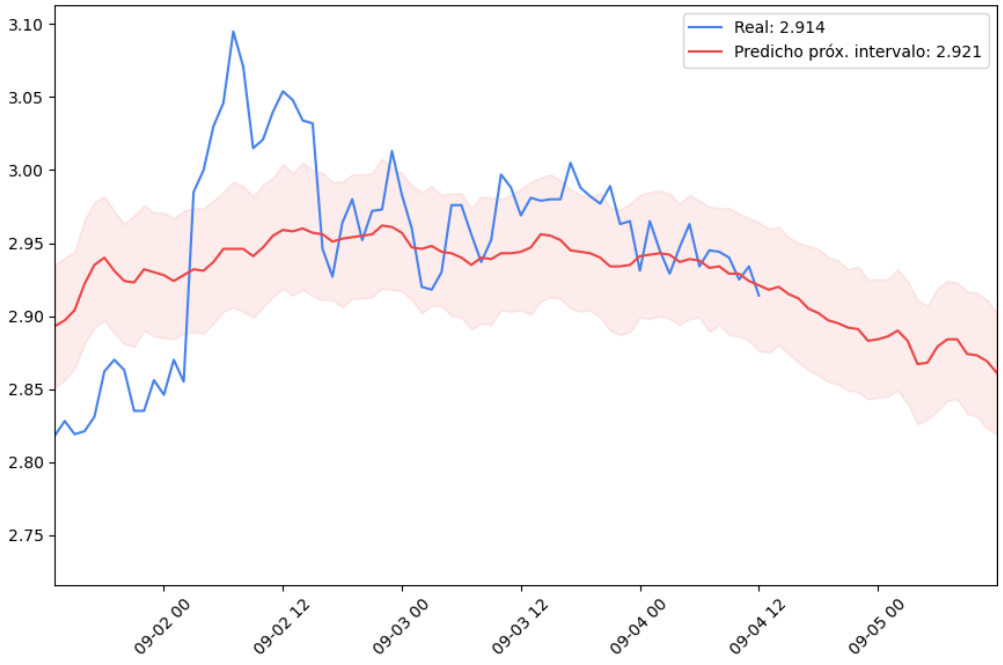
contexto puede ser utilizada, ya que la visualización es una consecuencia directa del propósito del sistema, no sería lo mismo mostrar un gráfico con los valores predichos por el modelo, que un gráfico que resalte la comparativa entre lo predicho y lo real u otro que muestre el error a lo largo del tiempo.

Por otro lado, un aspecto a valorar es también el requisito de ser una visualización en tiempo real, es decir, que se actualice a lo largo del tiempo mostrando datos en vivo. Como consecuencia, se actualiza la representación mostrada, dando información en tiempo real sobre el activo, así como la predicción en cuestión.

En este proyecto, debido a que se está tratando un sistema que sirva de apoyo a la toma de decisión en el proceso de trading, la gráfica principal a mostrar será la relacionada con la predicción realizada por el modelo. Adicionalmente, para que el experto pueda interpretar el gráfico y tomar sus propias decisiones, también es interesante mostrar el progreso completo del precio del par analizado.

La representación escogida consiste en una gráfica en la que se puede comprobar el valor real y el valor predicho, además del posible error en la predicción realizada:

Figura 30: *Visualización de resultados*



(Fuente: elaborado por el autor)

También, se ha añadido una etiqueta con el valor predicho para el próximo intervalo, así como el valor real actual que toma el activo analizado.

4.3.7. Prototipo 6: Despliegue de la aplicación

De cara a disponer de una herramienta flexible y portable que pueda ser utilizada en diferentes sistemas, es importante enfocarse en el despliegue de la aplicación. El despliegue de aplicaciones consiste en, como menciona Pressman(1982), presentar al cliente un incremento de software, o una solución de software completa, haciendo que el sistema esté disponible para ser usado, dándole apoyo al mismo y recibiendo retroalimentación del cliente. Plasmando esta idea en la herramienta desarrollada en este proyecto, se debe conseguir que el sistema de recogida, almacenamiento y predicción de valores de criptomonedas sea utilizable por el experto de una manera sencilla.
Para conseguir este objetivo, se ha optado por usar la herramienta Docker, que como se mencionaba con anterioridad, se trata de una herramienta para automatizar el proceso de despliegue de aplicaciones.
Para utilizar Docker, también hay diversas alternativas, pero se ha optado por usar Docker Desktop, que es una versión de escritorio de la herramienta, lo que facilita su uso. Esta es una opción recomendable en el caso de disponer de una máquina cuyo sistema operativo sea Windows, ahora bien, no es estrictamente obligatoria en caso de estar en disposición de un equipo con Linux, ya que en este se puede instalar Docker simplemente a través de la línea de comandos.

Después, se debe configurar la imagen a usar, es decir, las librerías y requisitos que debe tener el contenedor sobre el que se va a ejecutar la herramienta. En lo relativo a las imágenes, se dispone de la plataforma oficial de Docker conocida como Docker Hub, que consiste en un servicio para descargar y compartir imágenes de contenedores, y en la que se puede encontrar algunas imágenes oficiales de diversas tecnologías. Desde Docker Hub se descargan tanto la imagen de Spark (que incluye la API para Python), como la imagen de Kafka.

La imagen de Spark está desarrollada por Bitrock, empresa de consultoría y desarrollo de software que presenta Bitnami, una biblioteca de instaladores y paquetes de software, en este caso la imagen en Docker de Spark que se utiliza en este proyecto. Dicha imagen está desarrollada sobre el sistema operativo Debian, es decir, sobre una distribución de Linux, que dispone de todos los requisitos necesarios para poder utilizar la tecnología en cuestión.

En lo relativo a la imagen utilizada para Kafka, se ha utilizado la proporcionada por Confluent, empresa creada por algunos de los ingenieros que fundaron el proyecto Kafka,

que se desarrolla para ser utilizada a través de la Confluent Platform, es decir, a través de una plataforma formada por una aplicación que proporciona una interfaz gráfica para facilitar el uso de la cola de mensajes Kafka, aunque no es necesario, de hecho, en este proyecto se ha prescindido de la misma.

El diseño del contenedor utilizado se consigue a través de un fichero de configuración, el docker-compose, que consiste en un conjunto de parámetros de configuración a través de los que se establecen las condiciones concretas del entorno.

Figura 31: *Fichero docker-compose*

```
version: '3'
services:
  spark-master:
    image: docker.io/bitnami/spark:2
    environment:
      - SPARK_MODE=master
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - type: bind
        source: ./conf/log4j.properties
        target: /opt/bitnami/spark/conf/log4j.properties
    ports:
      - '8080:8080'
      - '7077:7077'
    networks:
      - spark
  spark-worker-1:
    image: docker.io/bitnami/spark:2
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://localhost:7077
      - SPARK_WORKER_MEMORY=1G
      - SPARK_WORKER_CORES=1
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - type: bind
        source: ./conf/log4j.properties
        target: /opt/bitnami/spark/conf/log4j.properties
    ports:
      - '8081:8081'
    networks:
      - spark
    depends_on:
      - spark-master
  zookeeper-1:
    image: confluentinc/cp-zookeeper:5.5.0
    hostname: zookeeper-1
    environment:
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
      ZOOKEEPER_LOG4J_ROOT_LOGLEVEL: INFO
      # ZOOKEEPER_LOG4J_LOGGERS: "INFO,CONSOLE,ROLLINGFILE" -- No Include
    ports:
      - "2181:2181"

  kafka-1:
    image: confluentinc/cp-kafka:5.5.0
    hostname: kafka-1
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181"
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka-1:29092,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_LOG4J_ROOT_LOGLEVEL: INFO
      # KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"
      - "29092:29092"
    depends_on:
      - zookeeper-1
  kafka-2:
    image: confluentinc/cp-kafka:5.5.0
    hostname: kafka-2
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper-1:2181"
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka-2:29093,LISTENER_DOCKER_EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9093
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_LOG4J_ROOT_LOGLEVEL: INFO
      # KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9093:9093"
      - "29093:29093"
    depends_on:
      - zookeeper-1
  app:
    build:
      context: ./
    ports:
      - 5000:5000
networks:
  spark:
    driver: bridge
```

(Fuente: elaborado por el autor)

Además, otro aspecto a valorar es la instalación de todas las bibliotecas de Python usadas durante el proceso de implementación de la herramienta. Esto se consigue usando un fichero Dockerfile, que se utilizan para coordinar la ejecución de la imagen diseñada

previamente. Este permite instalar bibliotecas de Python, ejecutar scripts en concreto o también, utilizar parámetros concretos durante la ejecución.

Figura 32: *Dockerfile*

```
FROM python:3.8

WORKDIR /app
COPY . /app

RUN pip install -r requirements.txt
```

(Fuente: elaborado por el autor)

4.3. Evaluación

En este apartado, se evalúa tanto la usabilidad de la herramienta, así como su aplicabilidad para resolver el problema propuesto en este proyecto.

En primer lugar, para analizar la usabilidad de la aplicación desarrollada es necesario plantear qué clase de persona la va a utilizar, ya que, la dificultad de la herramienta para ser usada, dependerá de cómo de familiarizada está el usuario en cuestión con las tecnologías implementadas. En este caso, el usuario no requiere de conocimientos en las herramientas big data utilizadas, simplemente debe tener conocimientos muy básicos del funcionamiento de Docker, para poder instalar la herramienta y ejecutarla. Por estos motivos, se considera que la usabilidad de la herramienta es relativamente correcta, ya que no requiere de amplios conocimientos ni tampoco requiere de la realización de muchos pasos intermedios para hacerla funcionar.

Por otro lado, también es importante evaluar la aplicabilidad de la misma para resolver el problema que se ha planteado. El objetivo de la herramienta, como se formuló previamente, era servir de apoyo a la decisión durante la tarea de trading que puede realizar un broker, es decir, el cometido de la herramienta no es en ningún momento servir como bot de inversión y es por ello que, pese a buscar un modelo lo más preciso posible, este no necesita tener tanta precisión como debería en caso de ser utilizado directamente para invertir. Por esto, se puede sostener que la aplicabilidad de la herramienta es relativamente correcta en lo que se refiere a servir como apoyo en una posible toma de decisiones.

5. Conclusión y trabajo futuro

A continuación, se explican las conclusiones resultantes del proyecto desarrollado y el las posibles líneas de trabajo futuro.

5.1. Conclusión

Como conclusión, dado el contexto planteado, la importancia creciente del mercado de las criptomonedas, reflejado en la capitalización de mercado que representan las operaciones realizadas en exchanges, así como el desarrollo de nuevas tecnologías para el manejo de grandes volúmenes de datos y la capacidad de computación, el desarrollo de una herramienta que sirva para analizar activos como las monedas virtuales se presenta como una necesidad de cara a mantener la competitividad dentro del ecosistema de compra-venta de acciones o, en este caso, criptomonedas. Por este motivo, se ha desarrollado una herramienta capaz de almacenar y predecir el valor futuro de una moneda virtual. Para ello, se han utilizado tecnologías como interfaces de programación de aplicaciones (API), websockets para la creación de conexiones, un sistema de comunicación entre los diferentes módulos desarrollados (Kafka), Spark para la manipulación y transformación de los datos, además del despliegue de un flujo de datos constante y finalmente, una predicción del valor futuro de la criptomoneda en cuestión usando el software desarrollado por Facebook conocido como Prophet. Como evaluación, se puede sostener que la herramienta se presenta como una opción con una relativa usabilidad por parte del experto en cuestión (broker) y también una aplicabilidad correcta dado el objetivo inicial, servir como apoyo a la toma de decisiones en la tarea de compra-venta de criptomonedas.

Por otro lado, para evaluar si se han conseguido los objetivos se presentan dos puntos de vista, el objetivo inicial del proyecto y la captura de requisitos planteada. En lo relativo al objetivo inicial del proyecto, se ha desarrollado un sistema que recoge y almacena datos del mercado de criptomonedas utilizando tecnologías Big Data e implementa un modelo que predice en tiempo real el valor de las mismas, realizando a su vez un sistema para la visualización de los resultados obtenidos, por lo que, se puede sostener que se ha cumplido el objetivo general, ahora bien, en lo relativo a los objetivos específicos, se han cumplido de manera parcial, ya que, siendo implementadas de manera satisfactoria las herramientas big data para la recogida, almacenaje y manipulación de datos, y también el modelo predictor, el sistema de visualización de datos no muestra la información con la precisión que podría ser necesaria, ya que esta se actualiza con cada intervalo, lo que puede no ser suficiente

dependiendo del tipo de análisis que se quiera realizar. En lo referente a la captura de requisitos, se han cumplido los siguientes tanto funcionales como no-funcionales:

- El sistema recoge en tiempo real datos sobre la cotización de la criptomoneda conocida como ADA respecto del valor de la criptomoneda destinada a ser reflejo del dólar conocida como USDT.
- Los datos recogidos se componen de: valor de apertura, valor de cierre (o actual), precio máximo, precio mínimo y volumen.
- Cuando el usuario indica por la línea de comandos, los datos obtenidos en tiempo real no solo se almacenan, si no que estos se utilizan para predecir, también en tiempo real, el futuro valor del par ADA/USDT mediante un modelo predictor.
- Cuando el modelo predice los futuros valores del par analizado, estos son plasmados de manera gráfica.
- El software es utilizable a través de un entorno virtual para asegurar la posible reproducción de los resultados y el funcionamiento en diversos sistemas.
- Se ha desarrollado una guía de usuario tanto para el proceso de instalación como para el uso de la herramienta en cuestión.
- El desarrollo debe es compatible con el SO Windows.
- El software debe funciona en PC.
- El software utiliza tecnologías Big Data para el manejo y gestión de los datos.

Por el contrario, no se han cumplido los siguientes requisitos:

- La efectividad del modelo también no se puede visualizar en tiempo real, comparando el valor predicho con el valor real.
- La herramienta desarrollada no es implementable en un servicio web en la nube.

Como resumen, la mayoría de los requisitos se cumplen a excepción de los relativos a la visualización de la efectividad del modelo y de la capacidad para que la herramienta sea implementada en un servicio web en la nube.

5.2. Trabajo futuro

Para hablar de las líneas de trabajo futuro, se puede empezar por la mejora del sistema de visualización, así como los puntos faltantes de la captura de requisitos. Para mejorar el sistema de visualización se presentan diversas opciones como, por ejemplo, ajustar la tecnología usada actualmente para dar unos resultados satisfactorios o también, utilizar otras tecnologías que puedan ofrecer un servicio similar al buscado. Estas tecnologías pueden ser herramientas como Tableau o PowerBI, aunque la alternativa más plausible es Kibana, que consiste en un software para la representación de visualizaciones en formato dashboard, es decir, cuadro de mando. Kibana se puede implementar a través de Docker, lo que hace que se ajuste adecuadamente a las necesidades del proyecto.

También, una limitación que ha condicionado el proyecto ha sido la imposibilidad de implementar un servicio en la nube. Lógicamente, muchas decisiones tomadas a lo largo del proyecto se han visto influidas por el hecho de no poder mantener un servicio en constante funcionamiento, lo que hubiera cambiado algunos aspectos muy relevantes como la arquitectura, diseñando seguramente una arquitectura Kappa. Es por esto que el implementar un servicio web, realizando las adaptaciones necesarias, se presenta como una línea de trabajo futuro bastante necesaria.

En lo relativo al modelo, ajustar el algoritmo para obtener un resultado más fiable puede ser también un aspecto a tener en cuenta en futuras aportaciones o mejoras. También, se podrían utilizar otras fuentes de datos como noticias, tweets o entradas en blogs relacionados con la criptomoneda analizada, para realizar un análisis de sentimiento, que consiste en hacer uso de técnicas de procesamiento del lenguaje natural para extraer valoraciones subjetivas de los datos. Como consecuencia, habría que implementar otro modelo que puede servir como apoyo para el ya implementado, obteniendo unos resultados más fiables o con más fundamento.

Además, uno de los aspectos más relevantes en el devenir de un modelo predictor es el desempeño a lo largo del tiempo, ya que estos algoritmos necesitan ser reentrenados con frecuencia para que consigan generalizar los nuevos patrones presentados en los datos. Por este motivo, una línea de trabajo bastante prometedora sería la monitorización del modelo con alguna herramienta como MLflow, que consiste en un software para controlar el ciclo de vida de los modelos desarrollados.

Finalmente, para ayudar lo máximo posible al experto, se podría valorar el desarrollo de una estrategia de inversión que acompañe a las predicciones realizadas por el modelo, aunque, lógicamente, esta dependerá del experto en cuestión y de cómo afronta la tarea de compra-venta de criptomonedas.

6. Bibliografía

Agencia de Seguridad Nacional. (1997). *How to make a mint: The cryptography of Anonymous electronic cash* (no. 4). Recuperado de https://digitalcommons.wcl.american.edu/cgi/viewcontent.cgi?article=1389&context=aulr

Bitcoin Project. (s.f.). *Preguntas más frecuentes*. Recuperado de https://bitcoin.org/es/faq#general

Chambers, B., Zaharia, M., (2018) *Spark: The Definitive Guide*. O'Reilly Media, Inc.

Chaum, David (1983). Blind signatures for untraceable payments. En Chaum, David. (Ed.), *Advances in Cryptology Proceedings of Crypto* (pp. 199-203). Springer, Boston, MA.

Claesen, M., De Moor, B. (2015). Hyperparameter Search in Machine Learning. *CoRR* abs/1502.02127

CoinMarketCap. (s.f.). *Top Cryptocurrency Spot Exchanges).* Recuperado el 15 de agosto de 2021 de https://coinmarketcap.com/rankings/exchanges/

Conway, L. (1 de junio de 2021). *Blockchain Explained*. Investopedia. Recuperado de https://www.investopedia.com/terms/b/blockchain.asp

Docker. (s.f.) *What is a Container?* https://www.docker.com/resources/what-container

Federal Reserve Bank of St.Louis (2021) *M2 Money Stock (DISCONTINUED).* Federal Reserve Bank of St.Louis. Recuperado de https://fred.stlouisfed.org/series/M2

Horvat, N., Ivković, V., Todorović, N., Ivančević, V., Gajic, D., Luković, I. (2020). Big Data Architecture for Cryptocurrency Real-time Data Processing. Ivković (Presidencia), *International Conference on Information Society and Technology.* Conferencia llevada a cabo en el X ICIST, Kopaonik, Serbia.

Kharpal, A. (6 de abril de 2021). Cryptocurrency market value tops $2 trillion for the first time as ethereum hits record high. *CNBC*. Recuperado de https://www.cnbc.com/2021/04/06/cryptocurrency-market-cap-tops-2-trillion-for-the-first-time.html

Kreps, J. (2 de julio de 2014) Questioning the Lambda Architecture. *Oreilly*. Recuperado de https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Liu, L., Özsu, M.T. (2009) *Encyclopedia of Database Systems*. Springer. doi: https://doi.org/10.1007/978-0-387-39940-9_565

Marz, N. (13 de octubre de 2011) *How to beat the CAP theorem.* Nathanmarz. Recuperado de http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

McNally, S. (2016). *Predicting the price of Bitcoin using Machine Learning.* (MSc Research Project). School of Computing National College of Ireland, Irlanda.

Mohapatra, S., Ahmed, N., Alencar, P., (2019) KryptoOracle: A Real-Time Cryptocurrency Price Prediction Platform Using Twitter Sentiments, *IEEE International Conference on BigData (Big Data)*, 2019, pp. 5544-5551, doi: 10.1109/BigData47090.2019.9006554.

Narkhede, N., Shapira, G., Palino, T. (2017) *Kafka: The Definitive Guide*. O'Reilly Media, Inc.

Pandas. (s.f.) *User Guide*. Recuperado de: https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe

Pressman, R.S. (1982) *Ingeniería del software: Un enfoque práctico*. McGraw-Hill.

Rose, B. (2018) *Real-Time Crypto: A big data approach to analyzing & automating cryptocurrency trading.*

Shah, D., Zhang, K. (2014). Bayesian regression and Bitcoin. *Communication, Control, and Computing*. Conferencia llevada a cabo en el LII Annual Allerton Conference, Illinois, USA.

Statista Research Department (6 de septiembre de 2021). *Largest stock Exchange operators worldwide in 2021, by market capitalization of listed companies*.

Recuperado de https://www.statista.com/statistics/270126/largest-stock-exchange-operators-by-market-capitalization-of-listed-companies/

Synkov, D., (29 de marzo de 2021). s*Spark-livy-on-airflow-workspace.* Github.

Recuperado de https://github.com/dsynkov/spark-livy-on-airflow-workspace

Tashman, Leonard J. 2000. Out-of-sample tests of forecasting accuracy: An analysis and review*. International Journal of Forecasting* 16 (4): 437–50. https://doi.org/10.1016/S0169-2070(00)00065-0

Taylor, S.J., Letham, B., (2017) Forecasting at scale. *The American Statistician*, 72:1, 37-45

https://doi.org/10.1080/00031305.2017.1380080

Trading Economics (2021) *Euro Area Money Supply M2.* Trading Economics. Recuperado de https://tradingeconomics.com/euro-area/money-supply-m2

Webb G.I., Sammut, C. *(2011) Encyclopedia of Machine Learning.* Springer.

doi: https://doi.org/10.1007/978-0-387-30164-8_623

Zaharia, M., Tathagata, D., Armbrust, M., Xin, R., (28 de julio de 2016). *Spark Structured Streaming.* Databricks. Recuperado de https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

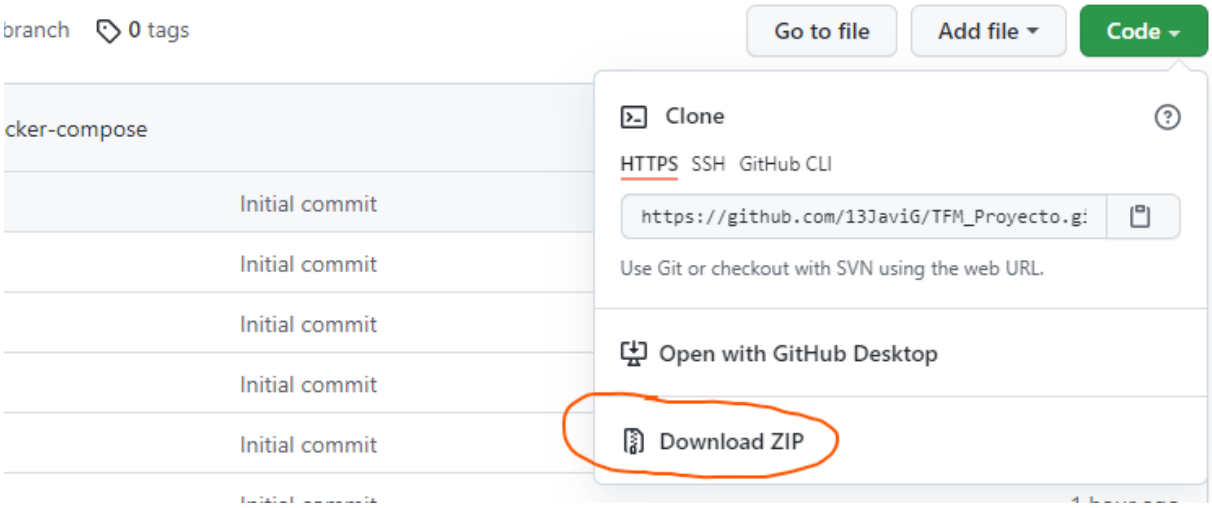
7. Anexos

7.1. Anexo. Guía de usuario

En primer lugar, es necesario instalar Docker, se aconseja instalar la versión Desktop para poder manipular los contenedores a través de una interfaz. Para ello se aconseja seguir los pasos indicados en la página oficial: https://www.docker.com/get-started

Una vez instalada la herramienta se debe descargar el repositorio con todos los archivos de la aplicación. Como se trata de un proyecto privado y personal, para tener acceso al mismo se debe pedir autorización a través de mi correo personal: javier.gil102@comunidadunir.net

Después, para obtener el código en cuestión, desde el directorio principal en la pestaña "Code" se descarga en formato ZIP el proyecto. Una vez descargade, se debe descomprimir para poder ejecutarlo.



Después, utilizando el fichero de configuración docker-compose, se construyen los contenedores que dan soporte al proyecto. Para este cometido, se debe ejecutar el siguiente comando:

```
PS D:\Master2020UNIR\TFM\PROYECTO_PYCHARM> docker-compose up -d
```

Este comando instalará todas las dependencias necesarias para que la aplicación se pueda ejecutar.

Para ejecutar definitivamente la aplicación en el contenedor ya activo, se debe ejecutar:

```
PS D:\Master2020UNIR\TFM\PROYECTO_PYCHARM> docker exec <nombreContenedor> python main.py <args>
```

En caso de no conocer el nombre del contenedor, se puede ejecutar el comando `docker ps` que permite comprobar los detalles de todos los contenedores creados previamente.

Como argumentos (<args>), existen dos opciones: entrenar y predecir. En caso de querer entrenar se debería ejecutar el siguiente comando:

```
PS D:\Master2020UNIR\TFM\PROYECTO_PYCHARM> docker exec <nombreContenedor> python main.py -o train
```

En cambio, si se quiere predecir:

```
PS D:\Master2020UNIR\TFM\PROYECTO_PYCHARM> docker exec <nombreContenedor> python main.py -o predict
```

Si es la primera vez que se ejecuta la aplicación, se recomienda entrenar y, una vez se haya entrenado un modelo (proceso que puede tardar bastante tiempo), se puede predecir. Pese a esto, ya existe un modelo entrenado en el proyecto descargado para que no sea necesario realizar este proceso tan tedioso.

Para ejemplificar un proceso de predicción completo se mostrarán los pasos que realiza el sistema. En primer lugar, se actualizará el Dataset, ya que, como se ha indicado en la memoria, el sistema no puede estar operativo constantemente y por tanto, no se actualiza el catálogo de datos a lo largo del tiempo. A continuación, se comprueba si el topic de Kafka ya existe y, en caso de que no, se crea. Después, se construye el stream reader de Spark y el stream writer que sirven para procesar los datos entrantes.

```
Actualizando datos...
Datos Actualizados!
---------------------------------
Creando topic...
El topic ya existe.
---------------------------------
Iniciando datos en streaming...
```

Finalmente, se activa el websocket que obtiene los datos desde la API de Binance y tanto la predicción como los datos reales se muestran en una visualización en tiempo real.