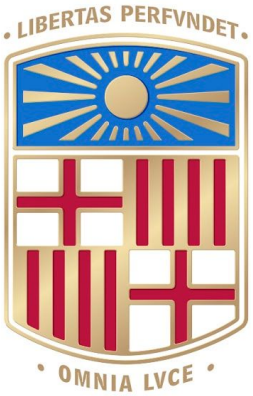
Trabajo Final de Grado



UNIVERSITAT DE BARCELONA

Grado de Ingeniería Informática

Facultad de Matemáticas e Informática

Universitat de Barcelona

“CryptoFy, a web application for cryptocurrencies traders”

Autor: Davis Yoel Armas Ayala

Director: Sergio Escalera

Agradecimientos

En primer lugar me gustaría nombrar a mi madre, mi padre y mi hermano, por haber hecho de mí, la persona que soy hoy. Por infundirme los valores más importantes de la vida. Por enseñarme a afrontar los problemas con la cabeza bien alta y no rendirme. Por apoyarme y ser honestos en todas las acciones y empresas que he tomado en todos mis años. Por esto y más, !Gracias de corazóni

Agradecer también a mi inmenso gran amigo Jesús Juan, que sin quererlo ni esperarlo, se convertiría en una de las personas más importantes durante mi etapa universitaria.

A mi tutor elegido como guía durante la realización del Trabajo de Fin de Grado, Sergio Escalera, por su buen humor y actitud positiva que mostró durante todos esos meses de desarrollo, así como por sus útiles y efectivos 'tips' que me sugería.

Por último pero no menos importante, a todos mis demás amigos, que con su apoyo en los momentos difíciles y sus bromas en los felices, estos 4 años hubieran sido una historia muy diferente. También a todas esas terceras personas, que como pasajeros que se suben y bajan de un tren, aparecieron y se fueron de mi camino, aportándome experiencias y momentos inolvidables.

Gracias a todos...

Resumen

Tras el nacimiento de *BitCoin* allá por el 2008 y su rápido crecimiento de forma globalizada, multitud de otros proyectos parecidos empezaron a florecer animados por el éxito que generaba el fruto de *Satoshi Nakamoto,* creador del BitCoin. Este conjunto de monedas o valores virtuales fue denominado "*AltCoins*".
Gracias a la multitud de criptomonedas que surgían (y surgen), un nuevo tipo de mercado fue establecido en torno a estos efectivos. El más destacado de todos, fue el *Trading*, que como su homólogo del Stock Market, se basa en comprar y vender el determinado valor de una criptomoneda por otra, con el objetivo (generalmente) de obtener beneficios.
Así pues, para ayudar al usuario a obtener los mejores resultados y ganancias, esta aplicación intenta ofrecer servicios vitales y de especial interés, para cumplir este objetivo. Tales servicios se clasifican en análisis en profundidad de los datos históricos de las criptomonedas y de varios exchanges con el fin de averiguar patrones y realizar estudios, así como una parte más interesante de predicción, donde el usuario puede solicitar el valor a futuro de una determinada moneda virtual.

Abstract

After *BitCoin* was born around the 2008 and since its fast and astonishing expansion begun over the globe, many other similar projects started to flourish motivated by the achievements made so far from the *Nakamoto Satoshi's* fruit, the creator of BitCoin. This set or group of virtual coins was denominated as "*AltCoins*".
Thanks to the numerous quantity of cryptocurrencies that were arising (and that still are), a brand new type of market was established around this group of virtual coins. The most outstanding action was the *trading*, that as its counterpart does at the Stock Market, it focuses on buying and selling a determined value of a given cryptocurrency for another one, with the final point (generally) of making a profit out of it.
So for that, to help the user obtain the best and highest results, this application tries to offer vital and of special interest services, to fulfill this objective. Such functionalities can be classified on a deep and smart analysing of the historic cryptocurrencies and exchanges data that aims to find patterns or make studies from it, as for another more powerful service based on the forecasting, meaning that a user can apply to see the future value of a given virtual coin.

índice

Introducción

Antes de comenzar, cabe destacar que este apartado no pretende, ni mucho menos, explicar o narrar de forma detallada la historia de las criptomonedas[1] o incluir en exceso detalles y/o datos sobre éstas, sino poner bajo contexto y ofrecer un mínimo de información al lector, para que su comprensión a lo largo de esta memoria sea más placentera.

Como muchos de ustedes ya sabrán, allá por el año 2008 surgió una idea la cual sugería la creación de un sistema de efectivo electrónico (dinero) basado en una red 'peer-to-peer'[2], la cual fuese descentralizada, es decir, que no estuviera controlada ni gestionada por ningún tipo de entidad o autoridad central, como ocurre por ejemplo con el *Euro* o el *Dólar* (entre casi todos los capitales y sistemas físicos existentes). Esta idea fue bautizada con el título de **BitCoin** y su creador se cree que fue un tal *Satoshi Nakamoto (nunca se ha sabido quién es o fue realmente),* el cual desaparecería tiempo después de que dejara en funcionamiento la red, y tras haber minado (proceso necesario para crear las criptomonedas) cerca de un millón de los correspondientes BitCoins (en principio el código y la cadena de éste está hecho para que no se puedan crear más de 21 millones de unidades, debido a que se quiso que fuera de carácter deflacionario y no perdiese valor con el paso del tiempo). Acto seguido sería *Gavin Andresen*, un desarrollador de software en gráficos 3D y realidad virtual por aquel entonces, el que "tomase las riendas" de este proyecto, convirtiéndose a largo plazo en el desarrollador líder dentro de la comunidad 'anárquica y/o liberal' que corresponde BitCoin.

Pues bien, en tan sólo unos años, esta idea pasaría de ser ser un artículo en una lista de distribución de criptografía, a un proyecto real y vivo en crecimiento constante, hasta consolidarse como el activo digital[3] número uno del mundo y pionero de otra cantidad de aplicaciones relacionadas (sistemas de pagos usando el BitCoin, donaciones, …).

Aunque sus primeros años fueron ligeramente complicados, no impidió que su éxito creciera de forma exponencial y a velocidad medianamente alta, lo cual alentó más temprano que tarde, que otros valientes se lanzaran a intentar emular el servicio y la funcionalidad ofrecida hasta el momento por **BitCoin**, como alternativas a éste (Etherum, Monero, EOS, etc…). Es así como aparece por primera vez el término 'AltCoins'[4] y se consolida un mercado de criptomonedas de una forma equiparable al existente *Mercado de Valores*, donde se registra y juega con el precio de una criptomoneda dada, en función de una serie de elementos clave que no entraremos en este apartado.

Debido a que este mercado es literalmente "nuevo" si lo comparamos con su homólogo ya nombrado *Stock Market*, y que muchas de las AltCoins surgieron hace relativamente muy poco, por no hablar de las nuevas que nacen casi cada semana, hace que se cree todo un abanico de posibilidades para el usuario de

estos servicios/sistemas, de salir beneficiado económicamente hablando (si se sabe cómo actuar) a través de la minería[5], *trading*, apoyo o aportaciones al código fuente de las criptomonedas ya existentes o nuevas, creación de un sistema o red nueva, aplicaciones que las usen, etc…
Así pues, centrándonos sólo en el *trading*, el acto por el cual un usuario cambia una cantidad en un sistema de efectivo (ya séase Dólares, Euros, …u otra Criptomoneda) por una porción equivalente al valor en ese momento, de la moneda virtual deseada, y viceversa (como una compra-venta de acciones en el Mercado de Valores).
Realizando esta acción se puede ganar una cantidad muy elevada de dinero si se efectúa una buena inversión junto con un *trading* inteligente, pero claro, para ello se debe de tener en cuenta muchos, muchos factores, al igual que se debe estar prácticamente atento a todos los movimientos y fluctuaciones de dicha criptomoneda, en todo momento, lo cual, si no se precisa de mucho tiempo, o se carece de información, suele acabar en pérdidas o beneficios bajos, en la mayoría de casos, llegando a perjudicar incluso a largo plazo el valor de dicho efectivo (sobre todo si es relativamente nuevo)…

Es por ello, que la necesidad de crear y generar herramientas que ayuden al usuario a obtener toda la información de la forma más sencilla y concentrada posible, e intentar que sus beneficios sean los mayores en todo momento, sin necesidad de estar las 24 horas del día atento a este mundo, se convierta en algo vital y de urgencia, entre todas aquellas personas que se engloban en este tipo de perfil de usuario.

Contexto y motivación

El creador de este proyecto y escritor de esta memoria, es uno de estos anteriormente nombrados usuarios, que atraído por el nuevo mundo de posibilidades que ofrecen las criptomonedas, un día decidió invertir parte de su dinero en este nuevo mercado, con la idea de incrementar lo invertido, y generar beneficios continuos a través del trading. Sin embargo, poco a poco se fue dando cuenta de que para lograr este objetivo, era necesario invertir una gran cantidad de tiempo controlando gran multitud de factores (de los cuales muchos de ellos eran y son complicados de entender), día tras día. Por desgracia, este tipo de inversión temporal era un lujo que por otros motivos no podía permitirse, y por ente, veía que sus objetivos principales se retrasaban, fallaban o malograba. Por ello, tras gastar incontables horas pensando cómo revertir esa situación o si había alguna solución viable para ello, tanto como buscando y probando numerosas aplicaciones y sugerencias de terceros, llegó a la conclusión, de que lo que necesitaba bajo sus actuales circunstancias, era disponer de una herramienta que le proporcionara la mayor cantidad de información reunida en un único lugar y que fuera fácil de entender y tratar, además de algún tipo de ayuda que le permitiera

saber en qué momentos era más o menos oportuno realizar cierta operación de trading, por ejemplo. Al no encontrar nada que le convenciera y que ofreciera todos sus requisitos y funcionalidades, fue cuando se formuló la pregunta: "¿Por qué no crear mi propia herramienta que me ayude a conseguir mis objetivos, y de paso, ayude a conseguir el de otros?". De ahí nació la idea de este proyecto: Una aplicación Web.

Actuación (resumen de lo que he hecho)

Pues bien, la aplicación web creada intenta reunir todos esos requisitos mencionados anteriormente, dividiéndolos en dos partes.

La primera sección, se centra en adquirir toda la información histórica sobre las criptomonedas existentes, que sea accesible y esté disponible por parte de los principales *Exchange* (mercados virtuales organizados, desarrollados principalmente para el uso del trading, ofreciendo gran cantidad de beneficios: seguridad, contraste, …) más fiables y robustos existentes hasta el momento (Poloniex, CoinBase, BitFinex, entre muchos otros). Estos datos abarcan la cara principal de estas monedas virtuales, como su precio medio, el precio de apertura y cierre por rangos temporales al igual que su precio más bajo y alto en esos mismos rangos, el volumen que mantiene, y muchos más.
Con esta información, se intenta dar libertad y rienda suelta al usuario, para jugar con ella, pudiendo seleccionar variables específicas, rangos temporales, criptomonedas de forma colectiva o individual y hacer lo mismo con los *Exchange*, entre muchas más combinaciones. Gracias a ésto, el usuario puede contrastar datos, ver o descubrir patrones, comparar criptomonedas o exchanges,…en definitiva, ayudarlo a que su trading e inversiones sean lo más seguras y acertadas posible.
Ahora bien, seguramente se esté preguntando cómo le va a ser sencillo entender toda esta cantidad de información donde abundan tantos datos y donde se puede hacer tal cantidad de combinaciones…pues muy sencillo, de una forma visual, es decir, a través de gráficas o *'charts'* dinámicas e interactivas con el usuario, donde el dato numérico deja de ser protagonista visual.

La segunda parte de esta aplicación web, va dirigida a intentar obtener el máximo beneficio posible a la hora de hacer trading. Para ello se lleva a cabo una acción de predicción del valor de una moneda virtual dada a un rango 'x' de tiempo en el futuro. Esto provoca, que si las predicciones son acertadas y fiables, el usuario sepa cuándo es mejor y en qué momento realizar una acción de trading dada (como

comprar o vender). A su vez, también ofrece la posibilidad de observar las predicciones que se han ido haciendo días tras día contrastándolas con el valor real de ese mismo instante, para mostrar la evolución y la fiabilidad de las predicciones.

2. Análisis

La primera parte del desarrollo de la aplicación web consiste en analizar todos los requerimientos y funcionalidades necesarias para obtener el resultado descrito en el apartado de "**Actuación**", e investigar las distintas tecnologías posibles a usar e implementar.

Diagramas de Casos de Uso

Para poder analizar dichos requerimientos y funcionalidades, se presentarán en diagramas y esquemas de casos de uso, que nos permiten un mayor desarrollo y entendimiento de éstos.

Se empezará presentando el diagrama de casos de uso general, que describe las acciones a nivel básico y primario que puede realizar el usuario para acceder a los distintos apartados de la aplicación:

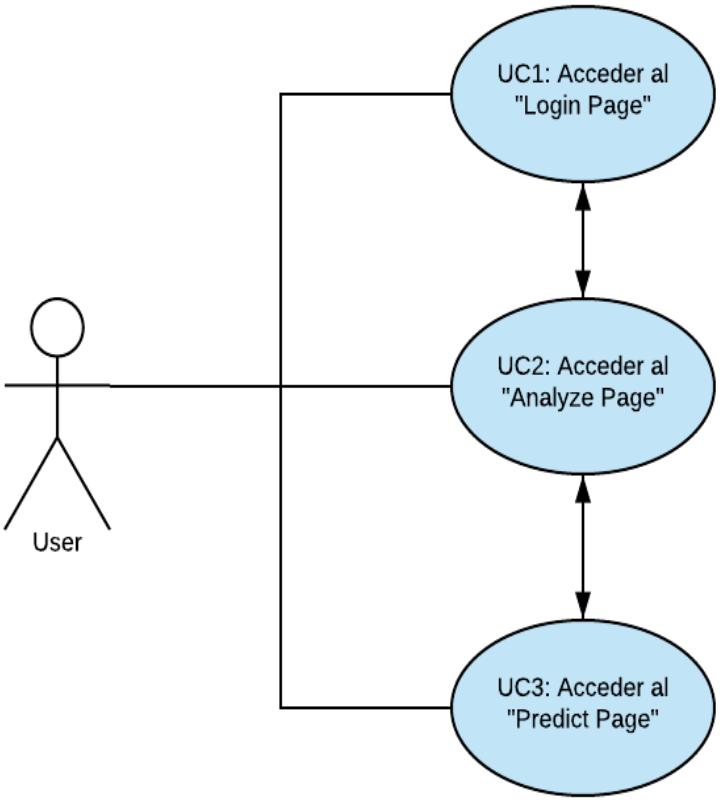


Figura 2.1: Casos de Uso Generales de acceso a la aplicación

Como se puede observar, se describen tres casos de uso, que resumen las posibles acciones de acceso. A continuación, se procede a definirlas en detalle.

UC1	**Acceder al "Index Page"**
Descripción: El usuario accede al Index Page, donde se muestra un breve resumen de la página y sirve como portal a las 2 funcionalidades de ésta.	
Actores: Usuario	
Pre-condiciones: ● Tener el navegador abierto. ● Que la aplicación esté funcionando.	
Flujo básico: 1. El actor introduce la URL específica (Index Page) para acceder a la aplicación (o llega siendo redirigido) 2. El sistema muestra la página solicitada de Inicio. 3. El actor puede interactuar entre las dos acciones posibles que se le ofrece a través de botones: Dirigirse a la página de Análisis, o bien a la de Predicciones.	
Post-condiciones: ● El sistema muestra la nueva página solicitada (el layout cambia)	

UC2:	**Acceder al "Analyze Page"**
Descripción: El actor accede a la página de Análisis, donde puede llevar a cabo una serie de acciones que le permitan analizar datos en detalle. También es capaz de volver a la página de Inicio o a la de Predicción.	
Actores: Usuario.	
Pre-condiciones: ● Haber pulsado el botón de acceso a la página de Análisis en el anterior caso de uso, o haber introducido la URL específica de este apartado en el navegador. ● Que la aplicación esté funcionando correctamente.	
Flujo básico 1:	

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario rellena y selecciona las distintas opciones mostradas en el formulario.
3. El usuario envía su petición al pulsar el botón de envío.
4. El sistema muestra los resultados solicitados.
5. El usuario demanda el formulario de vuelta, pulsando el botón de "Back".
6. El sistema muestra el formulario de nuevo.
7. Se vuelve al punto nº2.

Flujo básico 2:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario pulsa la opción de volver a la página de Inicio.

Flujo básico 3:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario pulsa la opción de ir a la página de Predicción.

Flujo básico 4:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario selecciona cualquiera de las opciones mostradas en el 'Footer' de la página.

Post-condiciones:

- El usuario es redirigido a la nueva sección/página.
- El sistema muestra la nueva página/funcionalidad pedida.

UC3:	**Acceder al "Predict Page"**

Descripcción: El actor accede a la página de Predicción, donde puede realizar una consulta sobre el futuro de precio de una criptomoneda, o analizar datos sobre predicciones. También puede volver a las páginas de Inicio y Análisis.

Actores: Usuario.

Pre-condiciones:

- Haber pulsado el botón de acceso a la página de Predicción en el anterior caso de uso.
- Haber introducido la URL específica de este apartado en el navegador.
- Haber pulsado el botón de acceso a la página de Predicción en la página de Inicio.
- Que la aplicación esté funcionando correctamente.

Flujo básico 1:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario selecciona la opción de mostrar predicción al pulsar el botón de 'Forecast'.
3. El sistema muestra los resultados.

Flujo básico 2:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario selecciona la opción de mostrar análisis de comparación de predicciones al pulsar el botón de 'Examples.
3. El sistema muestra los resultados.

Flujo básico 3:

1. El sistema muestra el 'layout' de la página de Análisis.
2. El usuario selecciona la opción de mostrar predicción al pulsar el botón de 'Forecast'.
3. El sistema muestra los resultados.
4. El usuario selecciona la opción de mostrar análisis de comparación de predicciones al pulsar el botón de 'Examples.
5. El sistema muestra los resultados.

Flujo básico 4:

1. El sistema muestra el 'layout' de la página de Predicción.
2. El usuario pulsa la opción de volver a la página de Inicio.

Flujo básico 5:

1. El sistema muestra el 'layout' de la página de Predicción.
2. El usuario pulsa la opción de volver a la página de Análisis.

Flujo básico 6:

1. El sistema muestra el 'layout' de la página de Predicción.
2. El usuario selecciona cualquiera de las opciones mostradas en el 'Footer' de la página.

Post-condiciones:

- El usuario es redirigido a la nueva sección/página
- El sistema muestra la nueva página/funcionalidad pedida.

Una vez descritos los casos de usos generales, se procede a analizar y detallar los casos de uso que se pueden encontrar en la página de Análisis:

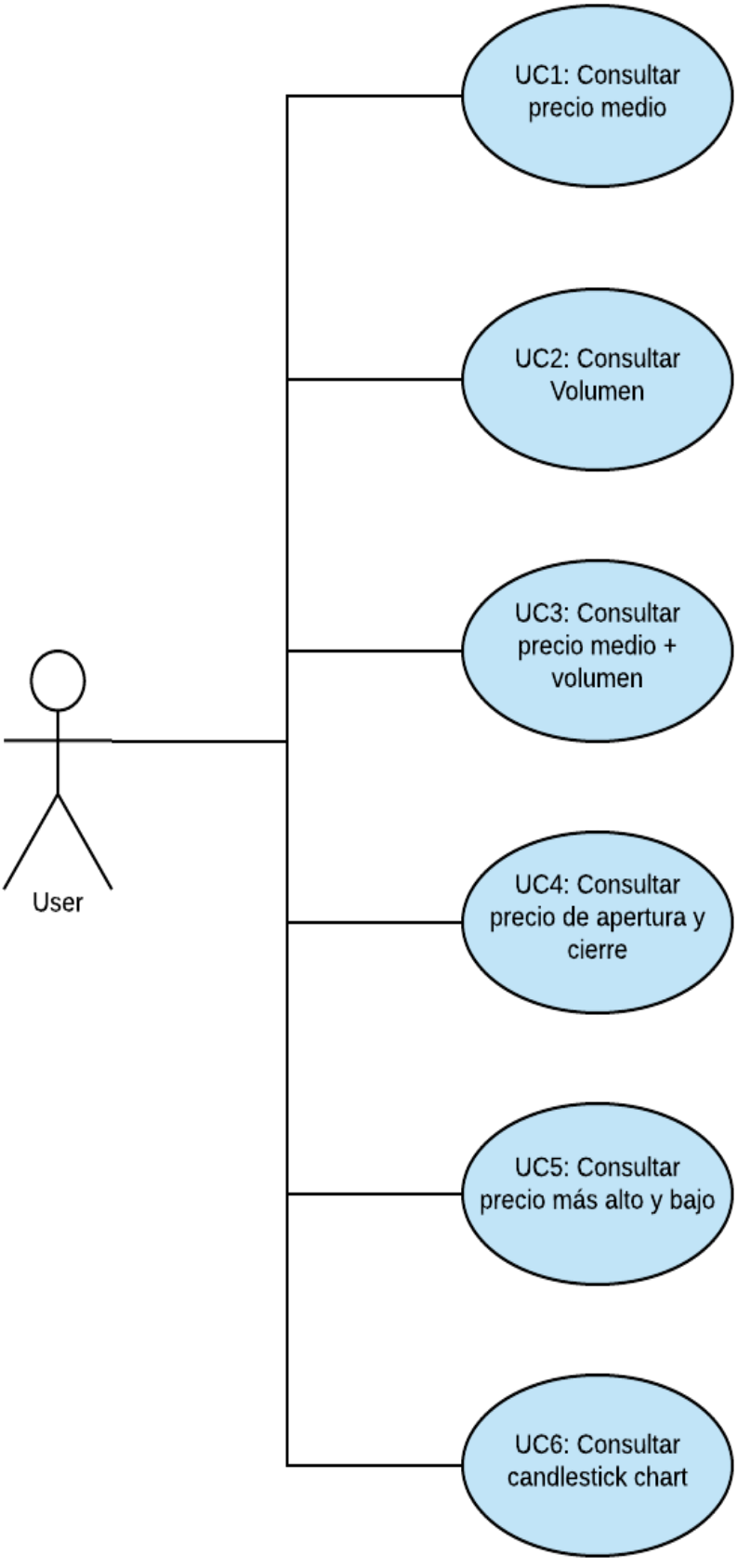


Figura 2.2: Casos de Uso encontrados en la página de Análisis

Tal y como se realizó en el caso anterior, se procederá ahora a explicar cada uno en detalle.

UC1:	**Consultar Precio Medio**
Descripción: Funcionalidad que permite mostrar los datos sobre el precio medio de una o varias criptomoneda/s para su análisis en detalle.	
Actores: Usuario.	
Pre-condiciones: ● Haber accedido a la página de Análisis. ● El sistema renderiza el 'layout' de la página. ● El formulario de petición esté presente.	
Flujo básico: 1. El usuario selecciona la criptomoneda a analizar. 2. El usuario selecciona el 'Exchange' del cual quiere saber los datos. Puede ser uno en específico, o todos de forma colectiva. 3. El usuario selecciona un rango temporal de inicio para la consulta de datos. (Opcional) 4. El usuario selecciona un rango temporal de finalización para la consulta de datos. (Opcional) 5. El usuario selecciona la opción de "Weighted Average Price". 6. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar. 7. El usuario envía la petición pulsando el botón de "Let's Go".	
Post-condiciones: ● El sistema muestra los datos solicitados por el actor. ● El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.	

UC2:	**Consultar Volumen**
Descripción: Funcionalidad que permite mostrar los datos sobre el volumen de una o varias criptomoneda/s para su análisis en detalle.	
Actores: Usuario.	
Pre-condiciones:	

- Haber accedido a la página de Análisis.
- El sistema renderiza el 'layout' de la página.
- El formulario de petición esté presente.

Flujo básico:

1. Pasos idénticos al Caso de Uso nº1 desde el punto 1 al 4.
2. El usuario selecciona la opción de "Volume".
3. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar.
4. El usuario envía la petición pulsando el botón de "Let's Go".

Post-condiciones:

- El sistema muestra los datos solicitados por el actor.
- El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.

UC3:	**Consultar Volumen + Precio Medio**

Descripción: Funcionalidad que permite mostrar los datos sobre el volumen junto con el precio medio de una o varias criptomoneda/s para su análisis en detalle.

Actores: Usuario.

Pre-condiciones:

- Haber accedido a la página de Análisis.
- El sistema renderiza el 'layout' de la página.
- El formulario de petición esté presente.

Flujo básico:

1. Pasos idénticos al Caso de Uso nº1 desde el punto 1 al 4.
2. El usuario selecciona la opción de "Weighted Average Price + Volume".
3. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar.
4. El usuario envía la petición pulsando el botón de "Let's Go".

Post-condiciones:

- El sistema muestra los datos solicitados por el actor.
- El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.

UC4:	**Consultar Precio de Apertura y Cierre**

Descripción: Funcionalidad que permite mostrar los datos sobre el precio de apertura al comienzo de uno de los intervalos de trading y el precio de cierre de ese período temporal de trading, de una o varias criptomoneda/s para su análisis en detalle.

Actores: Usuario.

Pre-condiciones:
- Haber accedido a la página de Análisis.
- El sistema renderiza el 'layout' de la página.
- El formulario de petición esté presente.

Flujo básico:
1. Pasos idénticos al Caso de Uso nº1 desde el punto 1 al 4.
2. El usuario selecciona la opción de "Open Price + Close Price".
3. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar.
4. El usuario envía la petición pulsando el botón de "Let's Go".

Post-condiciones:
- El sistema muestra los datos solicitados por el actor.
- El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.

UC5: Consultar Precio más Alto y Bajo

Descripción: Funcionalidad que permite mostrar los datos sobre el precio más alto y el más bajo dado durante los intervalos temporales de trading, de una o varias criptomoneda/s para su análisis en detalle.

Actores: Usuario.

Pre-condiciones:
- Haber accedido a la página de Análisis.
- El sistema renderiza el 'layout' de la página.
- El formulario de petición esté presente.

Flujo básico:
1. Pasos idénticos al Caso de Uso nº1 desde el punto 1 al 4.
2. El usuario selecciona la opción de "Low Price + High Price".
3. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar.
4. El usuario envía la petición pulsando el botón de "Let's Go".

Post-condiciones: ● El sistema muestra los datos solicitados por el actor. ● El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.

UC6:	**Consultar Candlestick Chart**
Descripción: Funcionalidad que permite mostrar los datos sobre el precio de apertura y de cierre, junto con el más alto y el más bajo dado durante los intervalos temporales de trading, de una o varias criptomoneda/s para su análisis en detalle. Estos intervalos se agrupan formando una especie de candelabro o vela (de ahí el nombre 'candlestick'). Además, presenta un simple "Moving Average" (una sucesión de medias derivadas de sucesivos segmentos).	
Actores: Usuario.	
Pre-condiciones: ● Haber accedido a la página de Análisis. ● El sistema renderiza el 'layout' de la página. ● El formulario de petición esté presente. ● Haber seleccionado una criptomoneda en particular. La opción colectiva no es válida.	
Flujo básico: 1. Pasos idénticos al Caso de Uso nº1 desde el punto 1 al 4. 2. El usuario selecciona la opción de "Candlestick Chart". 3. El usuario selecciona el intervalo temporal entre los datos, si ha seleccionado el BitCoin como criptomoneda a analizar. 4. El usuario envía la petición pulsando el botón de "Let's Go".	
Post-condiciones: ● El sistema muestra los datos solicitados por el actor. ● El sistema no muestra nada si los datos no han sido introducidos o son incorrectos.	

Seguidamente y por último, se muestra el diagrama de casos de uso analizados en la parte de la aplicación referida a la página de "Predict", y se detallan con tranquilidad:

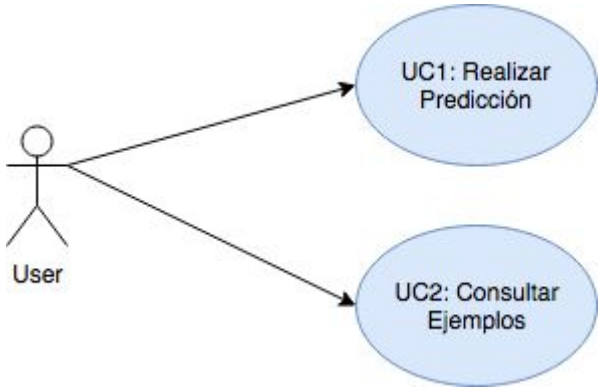


Figura 2.3: Casos de Uso encontrados en la página de Predicción

UC1:	**Realizar Predicción**
Descripción: Esta funcionalidad permite al usuario poder realizar una predicción sobre el valor de una criptomoneda a futuro, usando un algoritmo *machine-learning.*	
Actores: Usuario.	
Pre-condiciones: • Haber accedido a la página de Predict. • La base de datos haya sido actualizada con los últimos datos. • El algoritmo *machine-learning* funcione correctamente.	
Flujo básico: 1. El usuario presiona el botón de "Forecast". 2. La petición se envía al servidor. 3. El nuevo dato (ya calculado) se extrae de la base de datos. 4. El dato se devuelve al cliente.	
Post-condiciones: • El usuario debe ver el nuevo dato predicho en pantalla.	

UC2:	**Consultar Ejemplos**
Descripción: Funcionalidad que permite consultar una secuencia de resultados temporales predichos sobre el valor de una criptomoneda dada, junto a su valor real de forma homóloga, en una gráfica.	
Actores: Usuario.	
Pre-condiciones:	

● Haber accedido a la página de Predict. ● La base de datos haya sido actualizada con los últimos datos.
Flujo básico: 1. El usuario presiona el botón de "See Example". 2. La petición se envía al servidor. 3. Se extraen todos los datos referentes a la petición de la base de datos. 4. Los datos se usan para generar una gráfica. 5. Esta se devuelve al cliente.
Post-condiciones: ● El sistema abre una nueva pestaña en el navegador, donde muestra la gráfica creada con los datos.

Para finalizar este subapartado, cabe destacar dos aspectos.

El primero de ellos, es indicar que para cada uno de los 6 casos de uso de la página de análisis a excepción del último "Consultar Candlestick Chart", puede ser consultados de forma colectiva, es decir, si se toma por ejemplo la funcionalidad de "Consultar Volumen" ésta puede ser consultada para todas las criptomonedas, en vez de una en específico.

La segunda, hace referencia a la omisión de los posibles casos de uso de la página de "Index", debido a que por su sencillez, se considera cubiertas en los casos de uso generales.

Herramientas y Tecnologías usadas

Una vez explicada la parte lógica y funcional de la aplicación, es decir, los casos de usos, toca nombrar y argumentar las herramientas y tecnologías que se han usado para desarrollar y llevar a cabo todo lo anteriormente dicho en los apartados anteriores y que en conjunto, forman el proyecto descrito. Esto es necesario para poder entender lo que vendrá a continuación, y poner bajo contexto al lector de esta memoria, tal y como sucede al comienzo de la *Introducción.*

Pues bien, para evitar crear confusiones y mantener el flujo de lectura ordenados, se procede a comentar y estructurar las tecnologías de forma jerárquica o de árbol, empezando por las raíces, o en este caso, los elementos fundamentales por los cuales sin ellos no sería posible de ningún modo recrear el proyecto, hasta las ojas, que se traduce a los pequeños detalles y retoques de la aplicación.

Como ya se sabe, hoy en día existen una cantidad realmente extensa de lenguajes de programación que han ido surgiendo para cubrir la gran demanda que crea el gran avance y creación exponencial de nuevas tecnologías y usos que ofrece, si se compara con los existentes hace por ejemplo 20 años atrás. Muchos de ellos nacen para desarrollar o cubrir un campo u objetivo específico (siendo realmente potentes para ello)[SQL, HTML, IOS, ...]; otros se crean como versiones mejoradas y actualizadas de predecesores más antiguos (C++, Java, HTML5, ...); etc.
En definitiva, sin los lenguajes de programación no se podría construir el software que nos permite tener y disfrutar de la gran mayoría de herramientas y servicios que disponemos en la actualidad, y para la realización de este proyecto, no queda como una excepción.

Cuando se plantea la realización de una aplicación web, o funcionalidades de este tipo, se suele hacer un análisis en profundidad de todos aquellos lenguajes de programación que mejor se adapten para este campo de actuación. Éstos se pueden dividir entre **FrontEnd** y **BackEnd**, que vendrían a ser si ponemos por ejemplo un coche, la carrocería junto con el diseño que el usuario ve (FrontEnd) y el motor junto con todos sus mecanismos y engranajes que hacen que el coche funcione correctamente (BackEnd).

BackEnd

Pues bien, para este último, se ha utilizado el lenguaje **Python** que aunque no se haya creado especialmente para la realización de aplicaciones web, a diferencia de otros lenguajes como **PHP**, **PERL**, **RUBY**, ...ofrece una funcionalidad más que aceptable de este campo, además de otros factores que brinda, como el gigantesco apoyo y comunidad que hay detrás, permitiendo la realización casi que de cualquier tarea que se necesite llevar a cabo gracias a las miles de librerías ya creadas e implementadas o a la posibilidad de crearlas por ti mismo, su facilidad de lectura y escritura junto a la destacada reusabilidad de código, ...entre muchas otras cosas.

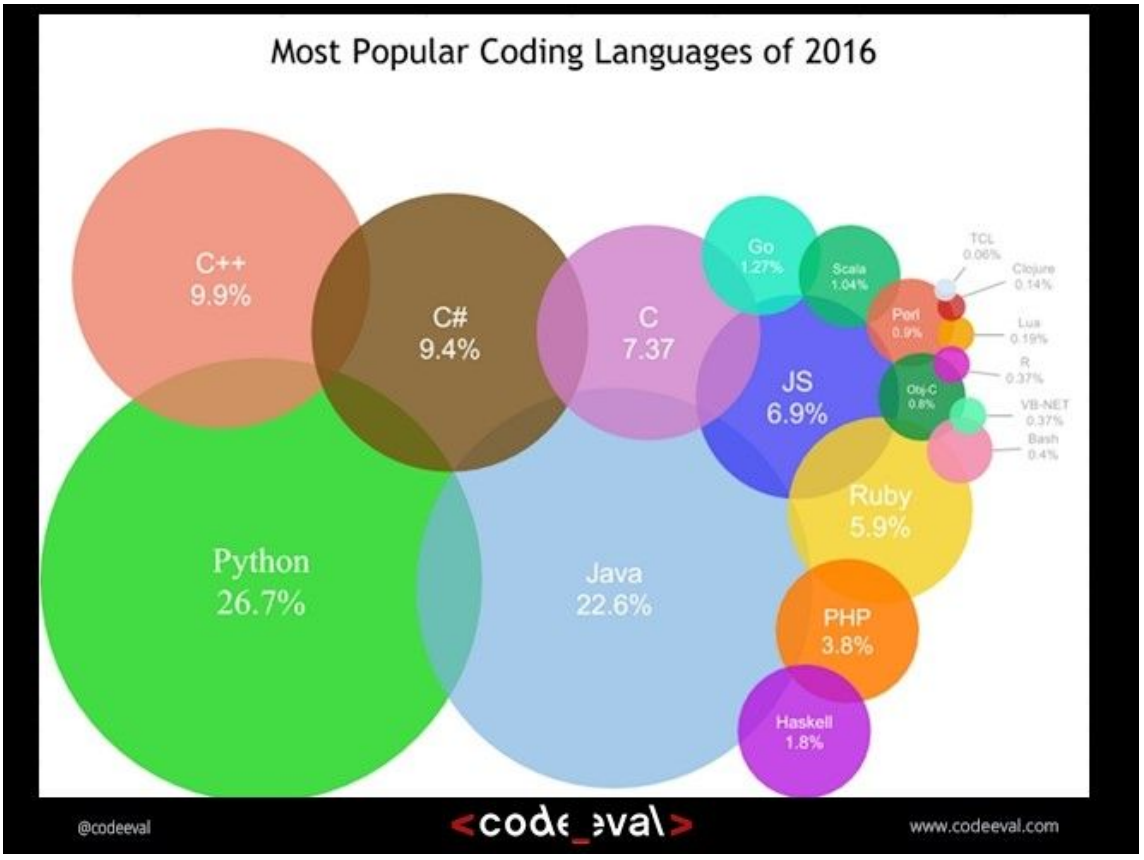


Figura 2.4: Referencia a algunos de los lenguajes de programación más populares y utilizados (del año 2016).

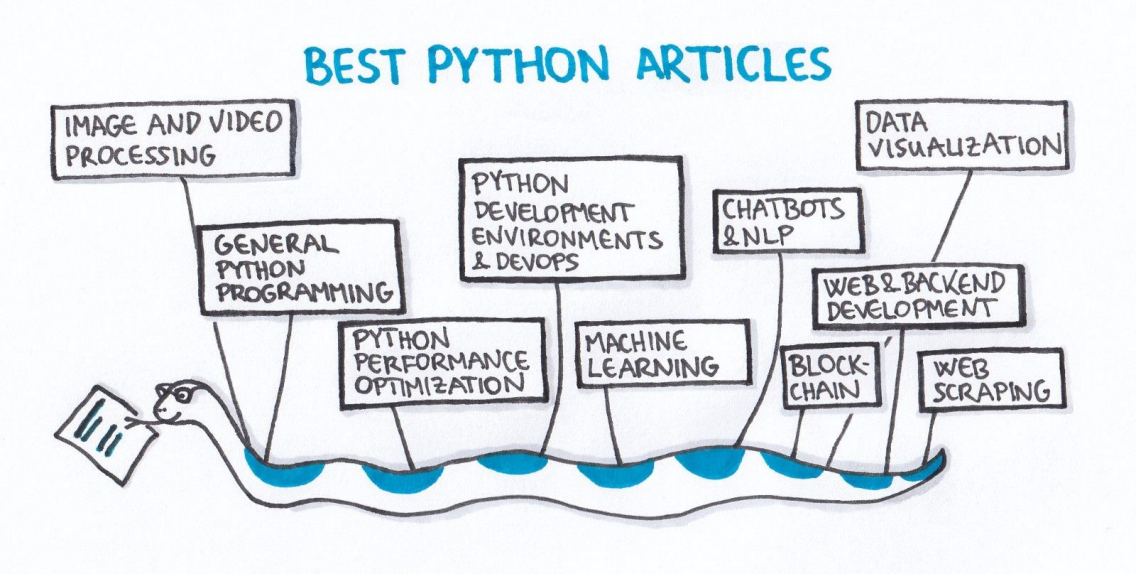


Figura 2.5: Algunas de las mejores funcionalidades que ofrece el lenguaje de programación **Python**

En cuanto al almacenamiento y uso de los datos, surgió el siguiente dilema: ¿Base de Datos o Sistema de Ficheros? Esta pregunta es realmente compleja y puede llevar a numerosos quebraderos de cabeza con muchas más preguntas, de las que no se entrará en detalle (para más información, consulte este enlace). De aquí solo cabe decir, que ambos sistemas son buenos, pero para determinados casos de uso. En el caso de esta aplicación, los datos (grandes cantidades) son utilizados numerosas veces (lecturas) con o sin condiciones previas (es decir, la posibilidad de acceder a determinadas zonas o trozos de esos datos que cumplan una condición dada), se añaden nuevos datos (escrituras) que pueden o no necesitar que los ya existentes sean reordenados de alguna forma, la inclusión de patrones o herramientas para acceder a los datos de la forma más rápida posible, etc...hizo que se decantase por usar una *Base de Datos* y un correspondiente lenguaje **SQL**, que para este particular caso sería **MySQL,** debido a que se ajusta mucho mejor a las características que se necesitan y nos ofrece herramientas específicas para ello. La decisión de usar esta última (y no Postgres o Oracle por ejemplo), fue sencillamente por tener conocimientos previos de su uso, y la facilidad que ofrece para gestionar y controlar los datos, así como la ayuda de librerías de apoyo en el lenguaje principal **Python**.

Con estos dos lenguajes, se podría construir ya una aplicación web desde cero, sin embargo, ésto sería un trabajo extensamente complejo y tedioso, y por ente implicaría un gasto de tiempo innecesario, más cuando existen numerosas herramientas que encapsulan todo este trabajo para que el usuario se centre sólo en moldear y desarrollar el software más externo. Estas herramientas

son muchas veces llamadas *Frameworks*, y Python nos ofrece algunos bastante interesantes para el desarrollo de aplicaciones web.
De entre ellos cabe destacar por encima los tres más conocidos y/o usados hoy en día:

- **RubyOnRails**, es uno de los más extensos y que ofrece multitud de funcionalidades. Tiene una poderosa base construida (y en constante mantenimiento) para poder realizar casi que cualquier tipo de aplicación que se desee, tanto a pequeña como a gran escala. Es ideal para realizar proyectos profesionales, sin embargo, requiere dedicarle bastante tiempo, y debido a su extensa amplitud de características y funcionalidades, puede llegar a ser bastante confuso de usar.
- **Flask** es un micro-framework ideal para la realización de proyectos a corta y media escala debido a su sencilla estructura que se centra específicamente en ello, brindando al usuario de todo su potencial para sacar el máximo provecho. No obstante, si se requiere un proyecto de gran escala, este framework empieza a quedar bastante anticuado y obsoleto.
- **Django**, es posiblemente uno de los más usado hoy en día debido a que gracias a su implementación permite el desarrollo de proyectos a cualquier tipo de escala, con un actuación casi impecable. Rapidez, sencillo de aprender y usar, simplificación, optimización de recursos, ...hacen de este completo framework una de las mejores apuestas para realizar una aplicación web.

Existen muchos otros frameworks orientados al desarrollo de aplicaciones web, que no se mencionarán en esta memoria, pero que pueden ser consultados si se desea, en esta lista.
Pues bien, debido a lo ya mencionado en la breve descripción anterior, y por el hecho de tener también conocimientos previos sobre él, el framework elegido para el desarrollo de este trabajo fue **Django.**
Este sigue el patrón Model-View-Controller o más específicamente un derivado suyo llamado Model-View-Template.
La parte de modelo consiste en clases Python que mediante un ORM interviene entre éstas y las múltiples tablas de la base de datos.
La parte de vista consiste en funciones Python que procesan peticiones HTTP y devuelven respuestas HTTP.
La parte de controlador es el URL dispatcher, es decir un fichero que contiene todas las URL de la plataforma y relaciona cada una con la función de la vista correspondiente.
La parte de template consiste en definir cómo se muestra la información al usuario.

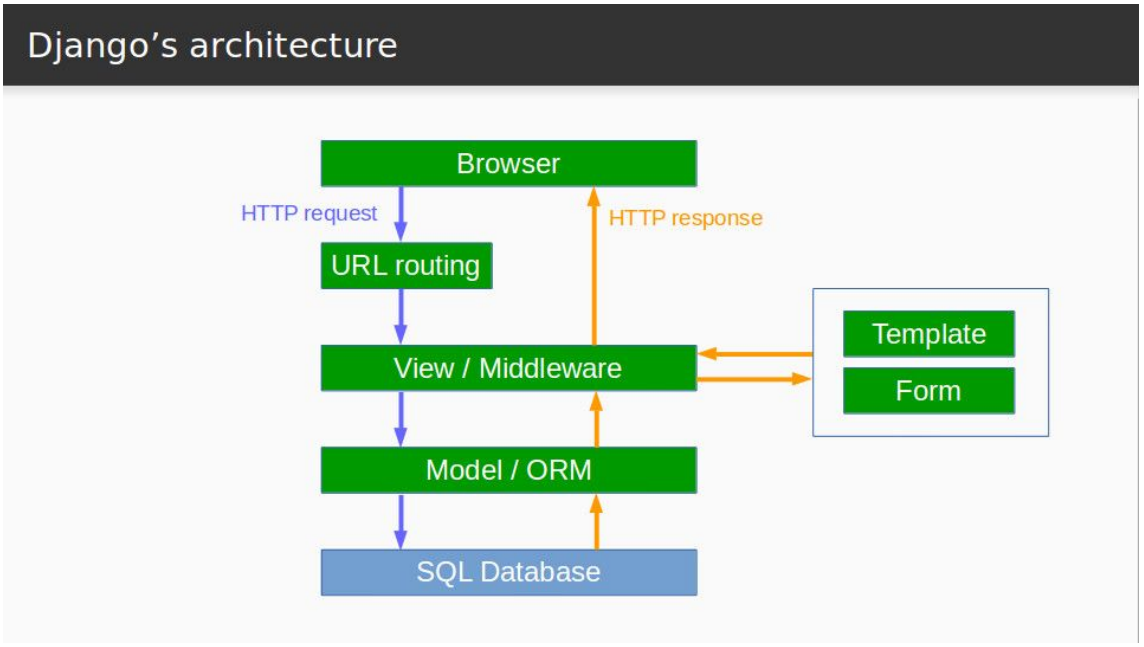


Figura 2.6: Arquitectura y funcionamiento básico del Framework Django.

FrontEnd

Todos los lenguajes y funcionalidades nombrados hasta ahora han sido parte del motor y los engranajes del proyecto, es decir, el software que hace que funcione todo como es debido y que se lleven a cabo todos los objetivos especificados. Sin embargo siguiendo con el ejemplo del coche, esto sólo sería como ver un amasijo de hierro y tubos sin ruedas, chasis, asientos, etc. El aspecto visual falta por completo, y por ente, el producto no está acabado. Aquí se mencionarán los lenguajes de programación usados específicamente para conseguir ese aspecto visual que se le muestra al usuario para que pueda interactuar con la aplicación.

Cabe destacar, que la gran diferencia con respecto a los lenguajes de BackEnd, es que aquí hay predominancia de un único lenguaje para cada cosa en particular y la competencia o alternativas, es nula.

Así pues, para no extender más esta breve introducción, se presentan:

- **HTML** (**H**yper **T**ext **M**arkup **L**anguage), para la creación y estructura de una página web (documento que lee e interpreta el navegador). Es la base principal.
- **CSS** (**C**ascading **S**tyle **S**heets), para dar forma, estilo y customización, a los elementos creados en HTML. Este lenguaje describe cómo los elementos de HTML deben verse en una pantalla o cualquier otro dispositivo.
- **JavaScript**, para añadir dinamismo y funcionalidad del lado del cliente o pre-backend. Se puede considerar como un lenguaje de backend integrado en el lado del front end y del navegador (cliente).

Predict-Análisis

Hasta este punto sólo se han nombrado y comentado lenguajes de programación y herramientas, justificando su uso. Sin embargo, no se puede dejar de lado el análisis y búsqueda de información y tecnologías a usar y/o desarrollar para poder llevar a cabo con éxito el algoritmo de predicción.

Machine-learning es una aplicación o rama de la Inteligencia Artificial*, que provee a un sistema la habilidad de aprender automáticamente y mejorar gracias a la experiencia sin estar explícitamente programada. Este proceso de aprendizaje consiste en realizar observaciones sobre conjuntos de datos (por línea general suelen ser conjuntos masivos de datos) tales como ejemplos, experiencia directa o instrucciones, con el objetivo de encontrar patrones en esos conjuntos para así poder realizar decisiones más acertadas en el futuro, basadas en ejemplos que se proveen.
Los algoritmos que caen en esta rama de la Inteligencia Artificial, suelen ser categorizados en cuatro categorías distintas:

- ***Supervised Machine Learning Algorithms***. Son aquellos que aplican lo que han aprendido en el pasado a nuevos conjuntos de datos, usando ejemplos calificados para predecir futuros eventos.
- ***Unsupervised Machine Learning Algorithms***. Son aquellos que se usan cuando la información usada para entrenar (training dataset) no está clasificada o determinada. Estos algoritmos estudian como sistemas pueden inferir una función para describir estructuras ocultas en datos no clasificados.
- ***Semi-supervised Machine Learning Algorithms***. Son aquellos que caen en el medio de los dos anteriores comentados, ya que usan tanto datos clasificados como no clasificados.
- ***Reinforcement Machine Learning Algorithms***. Son aquellos que interactúan con el entorno, realizando acciones que pueden resultar en errores o recompensas. A través de prueba y error junto con el feedback que reciben (penalización o recompensa), estos algoritmos aprenden y desarrollan eventualmente las mejores decisiones o comportamientos para conseguir un objetivo.

Pues bien, tras analizar estas categorías, se observó que la funcionalidad de realizar un algoritmo que predijera el valor futuro de una criptomoneda basado en sus datos históricos concordaba y asemejaba bastante con el tipo de ***Supervised Machine Learning Algorithms***.

Una vez identificada la categoría, se procedió a investigarla un poco más a fondo, descubriendo que los algoritmos podían ser subclasificados en otros dos apartados:

- De ***Clasificación***. Aquellos donde la variable de salida es una categoría, como "rojo" o "azul" o "limpio" y "no limpio", por ejemplo.
- De ***Regresión***. Aquellos donde la variable de salida es un valor real, como "dolares" o "peso", entre otros.

Este último subapartado encajaba a la perfección con el objetivo descrito, así que habiendo encontrado el campo y procedimiento idóneos, y descartando todas las demás categorías vistas, sólo quedaba encontrar modelos que siguieran este patrón o pauta.

Tras una última investigación, se descubrió que los modelos estándar ARIMA eran los más considerados y usados en este dominio.

ARIMA (**A**uto**R**egressive **I**ntegrated **M**oving **A**verage) es una de las herramientas más potentes usadas para la predicción de valores sobre conjuntos de datos temporales. Las iniciales de este modelo, se traducen en que trabaja con tres tipos de parámetros que deben ser introducidos al usarlo (p, d, q):

- **p**, es el número de términos autorregresivos, es decir, el número de valores previos los cuales el algoritmo mirará para realizar la predicción.
- **d**, es el número de diferencias (restas) que se necesitan para hacer el conjunto de datos "estacionario" (se comentará más adelante).
- **q**, representa el "moving average" de errores de predicciones previamente cometidos por este modelo.

Otros

Por último, cabe nombrar otros tipos de programas que aunque tengan menor peso de aportación respecto al proyecto o carezcan de la misma importancia, han sido de gran ayuda y que han facilitado el desarrollo:

- ➔ **PyCharm**, como IDE (**I**ntegrated **D**evelopment **E**nvironment) de *Python* y a modo de editor de texto, donde se ha desarrollado y escrito toda la aplicación. Ideal debido a la gran cantidad de funcionalidades que ofrece como análisis de código, un '*debugger*' gráfico, ...y destacado, apoyo al desarrollo web con *Django.*
- ➔ **Mozilla FirefoxDeveloperEdition**, como navegador para testear la aplicación, debido a las poderosas herramientas que éste facilita para el desarrollo web.
- ➔ **DataGrip**, como IDE para Bases de Datos. Permite la construcción de eficientes sentencias SQL de forma estática y supervisada, como el fácil control sobre aspectos de Bases de Datos relacionales.

3. Planificación

Se calcula que hubo un tiempo de dedicación estimado de entre dos y tres horas diarias de lunes a viernes, y de entre cuatro y seis horas algunos fines de semana (no notos incluídos), con una duración total comprendida entre veintidós y treinta semanas de trabajo.

La planificación del proyecto se dividió en diversas fases.
El primer paso fue el análisis del problema y las tecnologíias a utilizar en la primera parte de la aplicación (1 semana aprox.).
La segunda fase fue el aprendizaje de estas tecnologías junto con el diseño de la base de datos (4 semanas aprox.).
Seguidamente la implementación, donde se desarrolló todo lo referido sobre escritura del código y la creación del aspecto visual de la aplicación (7 semanas aprox.).
La última fase consistió en la realización de tests y pruebas con la finalidad de encontrar posibles errores (1 semana aprox.).
Las anteriores cuatro etapas, se repiten de nuevo para la segunda parte de la aplicación, sumando un total de 26 semanas.

Para finalizar, hay que añadir otras 3 semanas que se ajustan al desarrollo y escritura de la memoria.

Valoración económica

En este subapartado se realiza una valoración económica del proyecto, enfocado a nivel empresarial pero solo valorando el coste superficial que podría tener la plataforma.
Por tanto, si se supone un precio estándar de 10€ por hora el cual cobrase un programador *Full Stack*, un total de horas estimadas de alrededor de 450 (18 créditos x 25 horas por crédito), supondría un coste de 4500€ aproximadamente.
Faltaría añadir costes adicionales como el alquiler de un servidor donde alojar la aplicación, que podría suponer 20€ mensuales, más la obtención de un dominio público para la aplicación con posible coste de 10€ por mes también, entre otros.

4. Diseño e implementación

Como claramente el nombre de este apartado indica, presenta por fases la idea, construcción y ensamblaje de todo el proyecto, al igual que los métodos utilizados para ello, que de alguna forma rompen con lo acostumbrado hasta ahora.

Pues bien, como paso previo a cualquier tipo de diseño o implementación en código, lo primero que se hizo fue crear un entorno virtual separado, donde se llevara a cabo el proyecto, sin miedo o temor de que en algún momento éste fallase por cualquier tipo de dependencia que las librerías instaladas por necesidad produjeran debido a que las instala a nivel de ese directorio en una especie de copia del intérprete global de *Python* que se seleccione, pudiendo tener una especie de "*Roll Back*" del proyecto. Además también facilita la exportación y su uso a cualquier otra máquina que posea el mismo intérprete de *Python* y tenga esta capacidad de entorno virtual instalada, la cual se denomina '***virtualenv***' (para más información, consulte aquí).

Nota: A partir de ahora, todo lo que se comente en torno a la implementación, se entenderá que está creado dentro de este entorno virtual.

El paso posterior, fue la creación de un proyecto por defecto en *Django*, es decir, una base o "template" que ofrece este framework para poder empezar a desarrollar y crear. La excepción aquí, fue que se decidió omitir el uso de la propia base de datos integrada que ofrece (SQLite), para poder hacer uso de una ya existente (MySQL). Hasta este punto, no fue necesario el planteamiento de ningún diseño de la aplicación, puesto que no se requería, tan solo se creó la base o raíz del programa que sin la cual, no se podría haber llevado a cabo, o al menos del modo planeado.

Una vez realizadas estas dos cosas, sí fue momento de plantear un diseño esquemático de directorios, packages y módulos (como se le denomina a un fichero en python) tanto como de funcionalidad en cuanto a cómo relacionar las fuentes entre sí para conseguir los objetivos establecidos, y de estética para lograr el "*Look and Feel*" deseado.

Así pues, se decidió dividir y diferenciar el trabajo en dos principales partes. Primero desarrollar y testear la página de *Índex* y *Analyze*, y posteriormente la de *Predict*, tal y como se aprecia en el apartado de Análisis.

índex & Analyze (Front-end)

Para esta primera parte no se creó ningún diseño previo en plataformas como ***Balsamiq*** (o cualquier otra del estilo) ni se sabía la estructura de antemano que estas páginas debían poseer. Por contra, sí se conocían ya las funcionalidades que debían tener, por lo que se fueron creando diseños evolutivos, es decir, se empezaba por realizar un modelo básico, el cual se probaba para ver si complacía y se ajustaba a dichas funcionalidades, y se mejoraba a cada paso. Ésto se puede observar en el formulario de la página de *Analyze*, el cual empezó conteniendo un par de campos, que fueron creciendo para poder ajustarse a todas las opciones y variaciones necesarias para cumplimentar el objetivo de ofrecer al usuario el mayor juego de combinaciones posible y libertad de datos (sin mencionar pequeños detalles que ésto implicó implementar, como tooltips, textos, etc..).
Se crearon inicialmente los ficheros *HTML* (.html) junto con su *StyleSheets* (CSS) y funcionalidad extra escrita en *Javascript* dentro de los documentos *HTML* o como ficheros separados. Estas 'templates' se conectaron a funciones en un módulo de *Django* llamado "**views**" las cuales renderizan los archivos *HTML* y que actúan como *Callbacks* cuando un método HTTP/S es recibido a través de una URL.

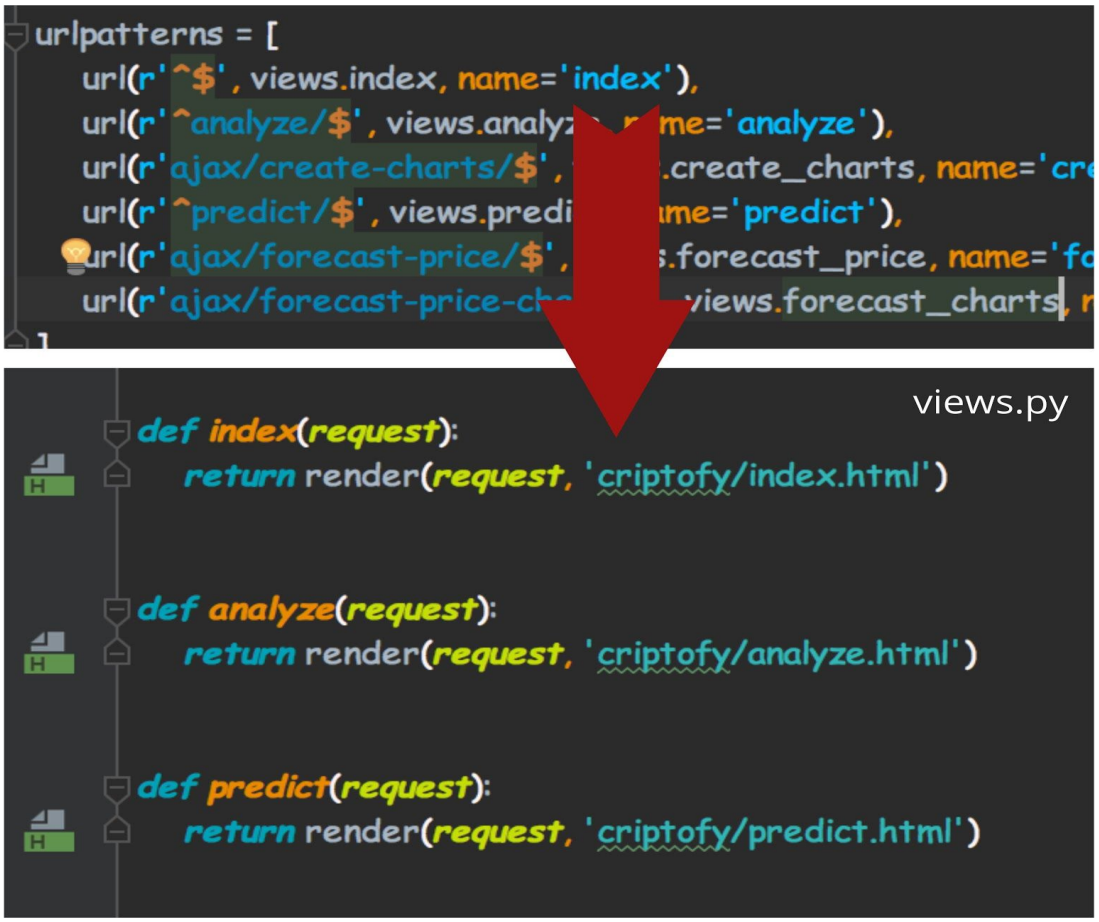


Figura 4.1: Imagen que explica lo que pasa cuando introducimos una URL, y cómo el framework de Django la traduce, llamando a una "view" específica que se haya indicado, la cual en este caso, muestra un archivo HTML.

Dentro de los archivos *HTML* se usó a su vez un framework de Front-End denominado **Bootstrap** siendo uno de los más populares y usados en todo el mundo, para conseguir principalmente que la página esté bien estructurada y sea "responsive" (que se adapte a todos los tipos de pantalla y sus cambios). Para ello se incluye un tag que apunta a este framework *online,* o bien se puede descargar y apuntar a él de forma local.
También se usó las sentencias lógicas que ofrece *Django* para incluir elementos o datos que necesariamente se encuentran fuera del ámbito de estos ficheros. Ésto es necesario para poder incluir imágenes o links externos, entre muchas otras cosas.

```
{% if today_is_weekend %}
    <p>Welcome to the weekend!</p>
{% endif %}
```

An `{% else %}` tag is optional:

```
{% if today_is_weekend %}
    <p>Welcome to the weekend!</p>
{% else %}
    <p>Get back to work.</p>
{% endif %}
```

Figura 4.2: Ejemplo de la utilización de los tags lógicos usados en las templates de Django

Para *Javascript* se usó, en numerosas funciones y declaraciones, otra librería bastante famosa y popular denominada **JQuery** que simplifica bastante el lenguaje de *Javascript* facilitando su uso.

Por regla general *Django* establece un orden de directorios preestablecidos para determinados aspectos, sobre todo para los elementos de *Front-end* dividiendo los archivos HTML y todos los elementos de estilo, así como de Javascript. Sin embargo, también deja cierto grado de libertad para la organización y división. Así, finalmente la organización de la parte visual quedó de tal forma como se puede apreciar en la siguiente figura.

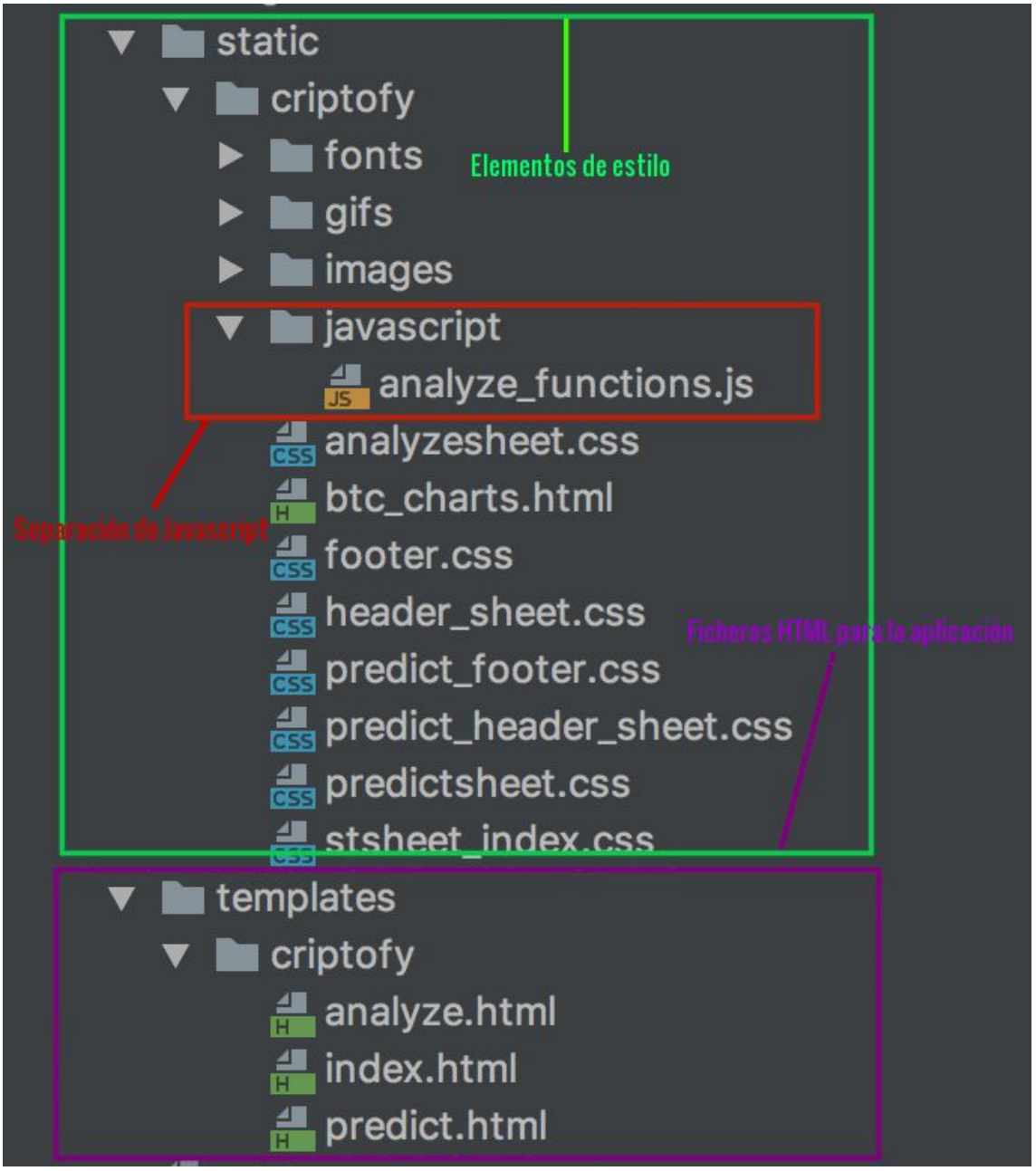


Figura 4.3: Identificación de la estructura de los elementos y ficheros usados para el Front-end de la aplicación, en Django.

índex & Analyze (Back-end)

Una vez creado el aspecto visual, se procedió al desarrollo y entrega de funcionalidad.
Para ello, lo primero que se realizó fue obtener el conjunto de datos con los que trabajar, ya que son la pieza fundamental de esta primera parte de la aplicación (recuerde que aquí se le da al usuario un amplio análisis sobre información específica de criptomonedas).
Se creó un directorio separado al que contiene el proyecto de *Django* con la intención de llevar a cabo todo tipo de acciones que se supone son realizadas por el sistema que sirve de anfitrión para la página web, ajenas a ella a modo de 'background'. Tales acciones incluyen: scripts de pruebas (test de funciones que se vayan a añadir a la aplicación o mejoras); control, manejo y actualización de datos; mantenimiento del sistema; entre otras...En este nuevo directorio, es donde se procedió a la creación de los datos, generando peticiones a las APIs (**A**pplication **P**rogram **I**nterface) de varias *webs de Exchange de criptomonedas*, por las cuales devuelve un conjunto de datos en función de parámetros que se incluyan en la URL (en este caso solicitó el histórico existente). Estos datos a continuación se guardaron en tablas acordes en una Base de Datos, para mantener su persistencia y utilizarlos en cualquier momento. Todo el proceso de forma detallada fue el siguiente:

- Se detalló la estructura de las tablas en la Base de Datos de MySQL. Como los campos devueltos son los mismos para las diferentes plataformas web, la estructura entre las tablas fue la misma (ver figura 3 para estructura en detalle de las tablas).
- Utilizando la librería **requests** de *Python*, se generó la petición a la API de cada plataforma.
- El resultado devuelto en dicha petición, un JSON (**J**ava**S**cript **O**bject **N**otation, es un formato que se usa para intercambiar datos. Se suele usar mucho en las aplicaciones web), es extraído y guardado en un *dataframe* o estructura de datos potente y adecuada para este tipo de volúmenes de información, facilitada por la librería **pandas** (estas estructuras han sido usadas para almacenar y usar la información a lo largo de toda la aplicación).
- Finalmente, estos datos se trasladan de dicho contenedor, a la tabla correspondiente.

Nota: Para actualizar las tablas (añadir nueva información) se realiza prácticamente el mismo procedimiento, con la diferencia de que se solicita únicamente el nuevo dato, y se inserta al final de la tabla.

Nota2: En algunos casos, otros campos fueron creados para algunas tablas, como producto entre combinaciones de determinados datos.

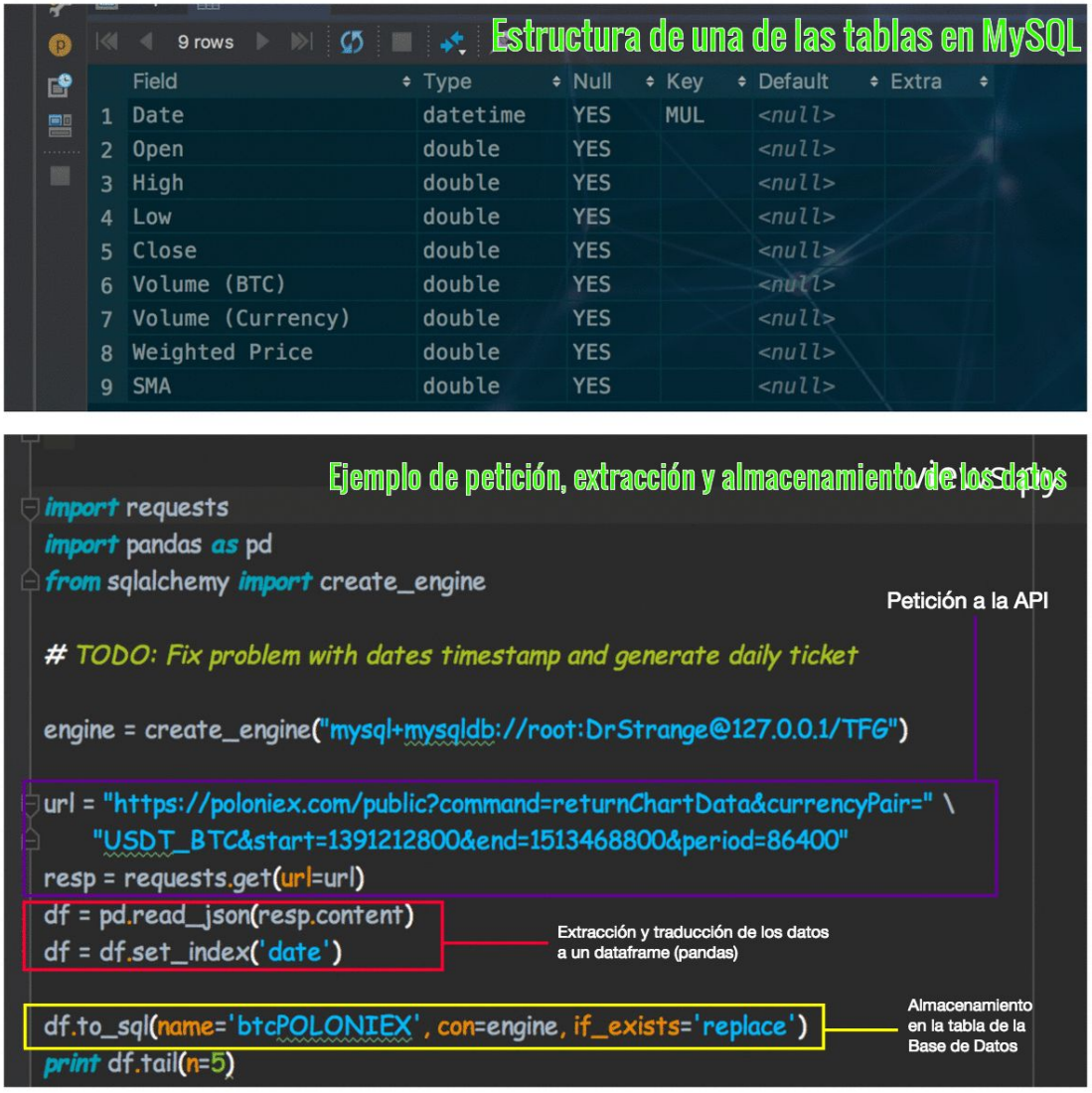


Figura 4.4: Ejemplos de una estructura de tabla MySQL y del proceso de adquisición de datos.

Una vez generado y almacenado el conjunto de datos (el histórico), el siguiente paso fue realizar un conjunto de pruebas para poder mostrar dicha información en tablas dinámicas e interactivas, para posteriormente trasladarlas a la aplicación. Sin salir de este directorio de pruebas y funciones extra, se crearon otra serie de archivos donde se procedió a extraer diversos tipos de conjuntos de información de la Base de Datos y mostrar a su vez numerosas combinaciones de éstos en gráficas observables en una pestaña completa del navegador, hasta conseguir el resultado deseado. El proceso al completo es como sigue:

- ➢ Extracción de los datos de cualquiera de las tablas existentes, y su almacenamiento en un dataframe o pandas.
- ➢ A través de una librería de visualización de datos o librería gráfica denominada **Plotly**, se generan las figuras a mostrar, utilizando la información de los dataframes y sus potentes funcionalidades.

- ➢ Se usan estas figuras junto con otros parámetros (visuales principalmente) para crear la gráfica que se muestra en el navegador.

Nota3: El uso de esta librería (*Plotly*) queda justificado debido a su simplicidad para incrustar las gráficas en aplicaciones web y a sus potentes funcionalidades de completa interacción.

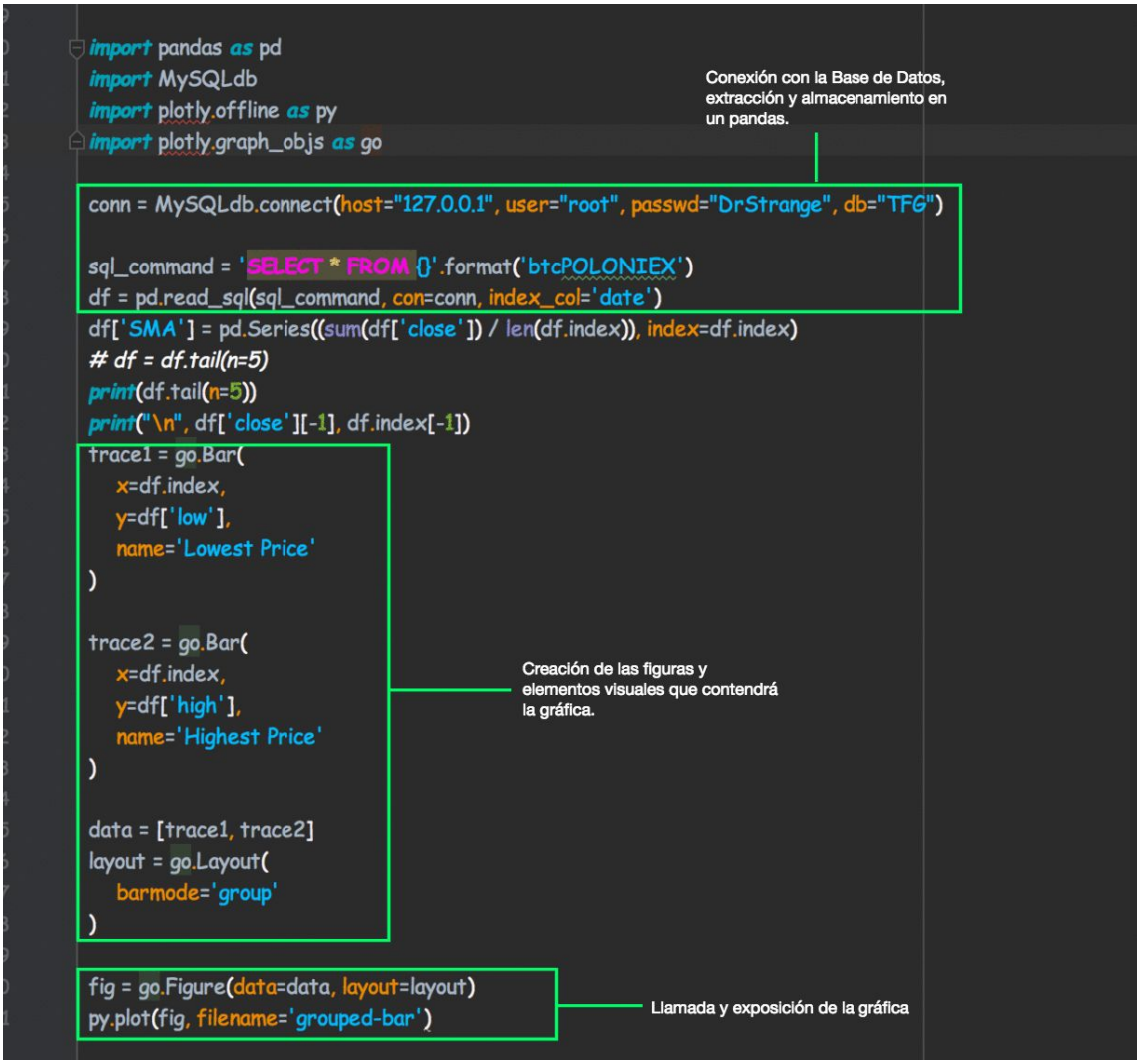


Figura 4.5: Proceso de prueba para la creación de gráficas dinámicas e interactivas.

Tras realizar este conjunto de procesos y test de manera exitosa, fue el momento de trasladar y adecuar esas pruebas para dotar de funcionalidad a los elementos del *Front-end*, que hasta ahora carecían de alguna. De todo este procedimiento se destaca la acción del formulario, junto al resultado que éste genera al ser enviado.

Dicho 'form' (de su traducción al inglés) presenta una serie de opciones por defecto para que el usuario elija, que se traducen en componentes *HTML* de tipo **<select>** (denominado 'select' desde ahora) con sus correspondientes subelementos **<option>**. Sin

embargo, algunos nuevos campos 'select' necesitan ser creados y mostrados en función de qué opciones dentro de los campos ya existentes por defecto el usuario haya elegido, es decir, se supone por ejemplo un formulario en el que hay un campo "bebida" y como opciones hay "agua" y "refresco". Si se selecciona "refresco", un nuevo campo "sabor" es mostrado con sus correspondientes opciones, mientras que si se escoge "agua", el campo anterior de "sabor" no es mostrado.
Para conseguir ésto, se usó el lenguaje de *Javascript* y la ayuda de su librería *JQuery* para programar este tipo de acciones. Mediante funciones donde se comprueba un flag (normalmente la opción seleccionada por ese mismo 'select' u otro), que genera dinámicamente el nuevo contenido a mostrar, o se destruye o esconde al usuario. Estas funciones se activan a modo de *CallBack* cuando un determinado suceso ocurre en ese elemento *HTML*. Los detonantes o "triggers" de estas funciones fueron en este caso `onchange` (se activa cuando cuando el valor del elemento o él en sí cambia) y `onload` (se activa cuando el elemento se carga en el documento).
Todo este proceso se convierte en una parte esencial, para poder mantener el formulario lo más simple posible generando el contenido en función de la demanda que el usuario solicite, tanto para controlar otros aspectos o variables de la información que se pretende mostrar, como lo es por ejemplo el tiempo de inicio desde donde se puede empezar a visualizar los datos de una criptomoneda (para cada exchange es distinto, debido a la fundación en años y/o meses diferentes) y que son manejados por estas funciones *Javascript*.
Todas estas opciones (por defecto y adicionales) se traducen en condiciones y flags que se envían al al lado del servidor para seleccionar distintos tipos y trozos de información de todo el conjunto de datos almacenado que se tiene.

Al pulsar el botón de envío del formulario, se crea una petición HTTP/S al servidor (normalmente con el método POST) con los datos/opciones especificadas en el 'form', solicitando un nuevo documento con la respuesta solicitada. Esta es la manera tradicional y una de las más usadas por su antigüedad, sin embargo es lenta y utiliza muchos más recursos. Como solución o alternativa, se creó la tecnología **AJAX**, que permite el intercambio de datos con el servidor 'detrás de la escena' es decir, sin que el hilo principal se rompa y por tanto, manteniendo el mismo documento y a la vez ahorrando muchos recursos (para más información sobre esta tendencia, consulte *este enlace*). Como era de esperar, fue esta última tecnología la que se decidió usar para el envío de los datos al servidor y recibir la respuesta en forma de gráfica. Todo el procedimiento queda detallado de la siguiente forma:

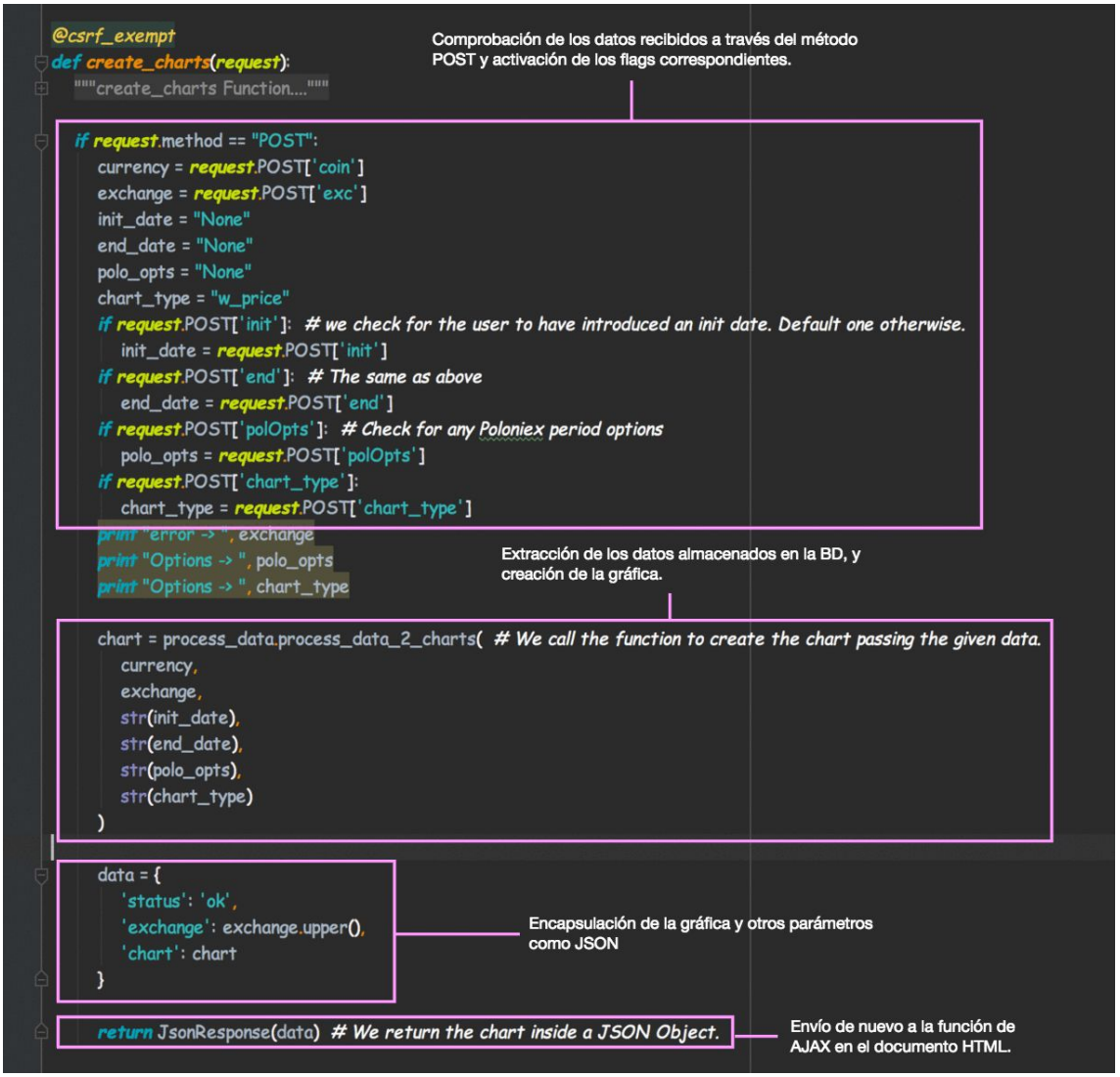
- Al pulsar el botón de envío del formulario, la función de `submit()` es activada usando JQuery. Dentro de esta función, se extraen todas las opciones elegidas por el usuario y se almacenan como un objeto *JSON* (key-value pairs). Esta librería nos facilita el

uso de la tecnología *Ajax* que de por sí suele ser algo compleja, por lo que con su ayuda se crea el objeto a ser enviado que contiene el tipo de petición *HTTP/S* que en este caso es POST, la URL de destino donde se deben enviar los datos, el *JSON* donde se encuentran almacenados los datos y su tipo, y algunos parámetros más. También se desarrolla una función de éxito (`success: function (data)`) en caso de respuesta por parte del servidor, la cual entrará en juego en el último paso de este proceso.

- El objeto *JSON* es enviado a la URL especificada a través del método POST, la cual es mapeada dentro del archivo "urls.py" del proyecto "criptofy" que llama su correspondiente función dentro de "views.py" (se recuerda que éstas actúan como CallBacks en caso de que se mapee la URL).
- En la función seleccionada dentro del archivo "views.py", se procede a extraer los datos encapsulados en el método POST recibido. Se procede a realizar las comprobaciones necesarias sobre estas variables recibidas, en función de activar los flags correspondientes para la extracción de la información guardada en la Base de Datos.
- Se extrae la información de la Base de Datos y se genera la gráfica a mostrar. Este paso se comentará posteriormente con más detenimiento.
- La gráfica junto con otros parámetros es encapsulada de nuevo en un objeto *JSON* y es devuelto al documento HTML de partida, donde aquella función de éxito nombrada en el primer paso, entra el juego.
- Esta función recibe el objeto con toda la información. Elimina el formulario, y lo reemplaza con un nuevo contenedor donde inserta la gráfica y algunos elementos más. Todo usando la librería de *JQuery.*

Figura 4.6: Tecnología AJAX usada en la aplicación para la página de *Analyze*.

Figura 4.7: Mapeo de la URL usada en la anterior figura, que llama a su correspondiente "view".




```python
@csrf_exempt
def create_charts(request):
    """create_charts Function...."""

    if request.method == "POST":
        currency = request.POST['coin']
        exchange = request.POST['exc']
        init_date = "None"
        end_date = "None"
        polo_opts = "None"
        chart_type = "w_price"
        if request.POST['init']:  # we check for the user to have introduced an init date. Default one otherwise.
            init_date = request.POST['init']
        if request.POST['end']:  # The same as above
            end_date = request.POST['end']
        if request.POST['polOpts']:  # Check for any Poloniex period options
            polo_opts = request.POST['polOpts']
        if request.POST['chart_type']:
            chart_type = request.POST['chart_type']
        print "error -> ", exchange
        print "Options -> ", polo_opts
        print "Options -> ", chart_type

        chart = process_data.process_data_2_charts(  # We call the function to create the chart passing the given data.
            currency,
            exchange,
            str(init_date),
            str(end_date),
            str(polo_opts),
            str(chart_type)
        )

        data = {
            'status': 'ok',
            'exchange': exchange.upper(),
            'chart': chart
        }

        return JsonResponse(data)  # We return the chart inside a JSON Object.
```

Figura 4.8: Función que actúa como CallBack de la URL mostrada en la anterior figura.

Antes de finalizar con esta primera parte de la aplicación, falta comentar el proceso por el cual se crea la gráfica que se muestra en el último paso de la anterior descripción. No obstante, esta parte es bastante compleja, por lo que para mantener el flujo de comprensión lo simple posible, se adaptará a un nivel más abstracto pero que englobe toda la funcionalidad.

Así pues, en el momento que se reciben los datos enviados por el formulario en la "view" designada (función de *CallBack*), éstos se pasan a otro fichero llamado ***process_data.py*** que se encarga de procesar dicha información, es decir, comprobar qué parámetros (dentro de unos estándares) se reciben y enviarlos a otra función localizada en el archivo ***creation_btc_charts.py*** el cual es el encargado de extraer los datos de la Base de Datos en función de los parámetros recibidos y de la creación de la gráfica con dicha información extraída.

Centrando la vista en este último fichero, se puede percatar un ligero inconveniente. Como ya se sabe, cada proceso es ejecutado (a no ser

que se indique lo contrario) por un único hilo principal de ejecución denominado la gran mayoría de veces *"Main Thread"*. Ésto no supone problema alguno en procesos donde la carga computacional de las tareas a ejecutar es baja o incluso mediana (hoy en día las CPUs y memorias de las computadoras son bastante potentes y rápidas, lo cual permite una actuación muy eficaz en procesos que antaño hubieran necesitado la división de sus tareas en multiprocesos o hilos), sin embargo el rendimiento se puede ver muy afectado cuando se topa con alguna o algunas tareas que sí tienen una carga computacional bastante elevada, ralentizando por ente, la ejecución de todas las demás labores. En este caso, esas tareas son la de extraer la gran cantidad de información histórica (gran volumen de datos) de la Base de Datos guardandola en un dataframe (contenedor), y el procesamiento de esta información para la creación de la gráfica. Para intentar rebajar esta carga computacional tan intensa, se decidió usar la metodología de "*Multi Threading*" y "*Multi Processing*" que aunque poseen ciertas diferencias ambas apuntan a la división de una tarea en subpartes para su ejecución en paralelo (ver ejemplo en la figura nº 9).
Una vez solventado este inconveniente, se genera la gráfica usando parte de los conceptos aprendidos en las pruebas hechas con la librería **Plotly** de *Python* previamente. La gráfica va contenida como un conjunto de elementos *HTML* dentro de un string, el cual es pasado de vuelta por todos los ficheros anteriormente nombrados de forma inversa, hasta llegar al archivo **"analyze.html"** donde es insertada.
Todo este conjunto de procesos y pasos se puede ver como una especie de patrón 'Model-View-Controller' (*MVC*), donde la vista se puede identificar como el archivo **"analyze.html",** el controlador como el conjunto de **"views.py"** y **"process_data.py"**, y el modelo como **"creation_btc_charts.py"** junto con otros módulos auxiliares.

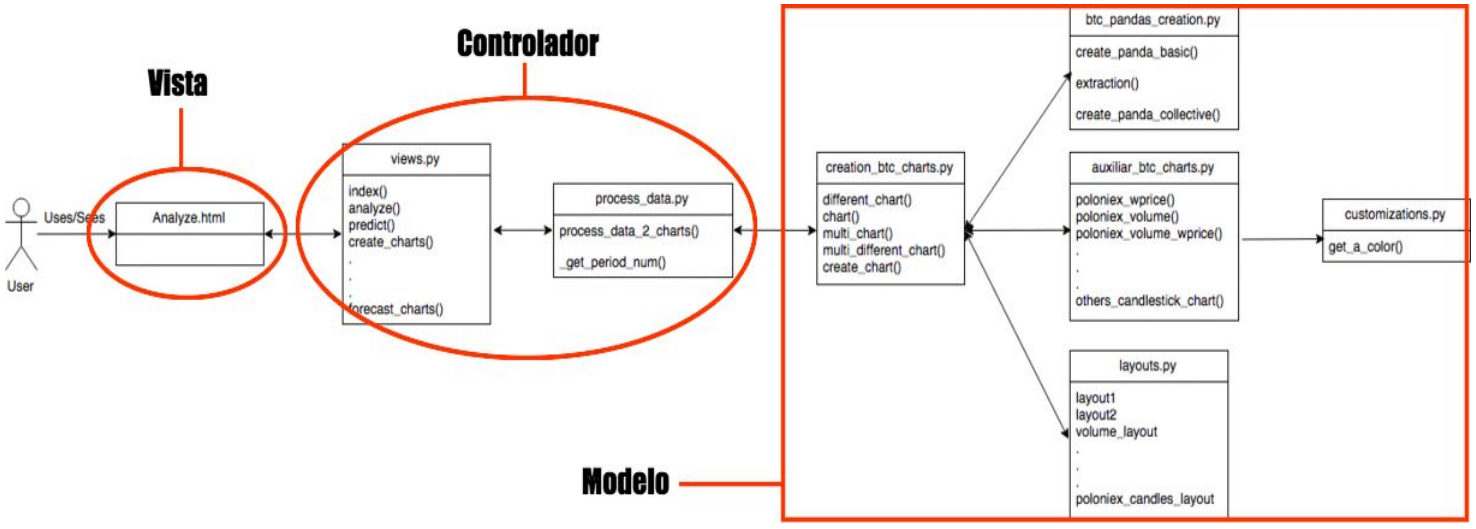


Figura 4.9: Aproximación y simulación de las clases usadas para la extracción de datos y creación de la gráfica, bajo el patrón **MVC**.

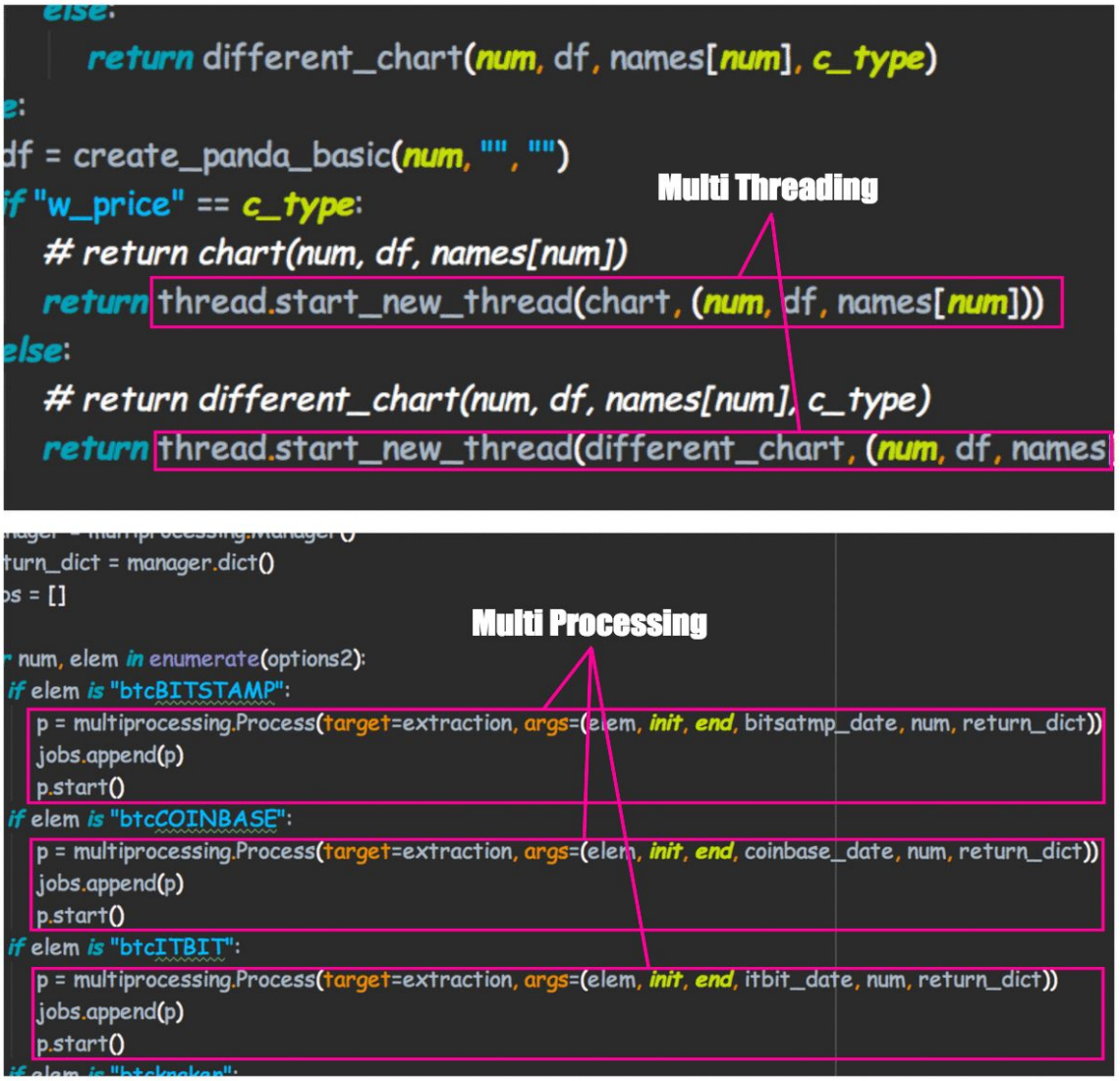


Figura 4.10: Ejemplo de uso de la metodología *Multi Threading* y *Multi Processing*.

Predict (Front-end)

Para el apartado visual de esta segunda parte de la aplicación, se reutilizó gran cantidad de código de la anterior sección (concretamente de la página "Analyze") ya que se quería mantener un patrón estético entre todas las partes.
La complejidad aquí disminuyó debido a que no se necesitó tantos elementos gráficos, así como la poca dotación de funcionalidad excesiva a los elementos existentes.
La creación y utilización de código Javascript fue prácticamente inexistente salvo en contadas ocasiones como el uso de la tecnología AJAX (se explica en el siguiente punto) de nuevo, para realizar peticiones al servidor, y algún que otro elemento dinámico como una barra de carga.
La organización y estructuración de directorios se mantiene incluída en la ya vista del punto homólogo a éste, más arriba.

Predict (Back-end)

Como se recordará, la funcionalidad principal era preveer, es decir, dar un dato a futuro, del valor de una criptomoneda (forecasting).
Para lograr este objetivo se investigó qué librerías mejor se adaptaban al desarrollo de algoritmos *"machine-learning"* (procedimiento por el cual un programa o sistema aprende y mejora con la experiencia, generalmente gracias a un conjunto de datos de entrenamiento). El resultado concluyó con el uso de la librería **statsmodels**, debido a que ofrece una gran variedad de clases y funciones para la estimación de diversos modelos estadísticos, desarrollo de pruebas y tests, y exploración de datos estadísticos entre otras cosas (más información *aquí*).

Con esta información obtenida, se procedió a realizar las primeras implementaciones y pruebas de este algoritmo, bajo el mismo directorio de testing donde se realizó en su momento las pruebas de creación de gráficas, pero en una carpeta distinta.
Los pasos fueron los siguientes:

- Se extrajeron los datos completos de la Base de datos de la misma forma que en anteriores puntos, es decir, almacenándola en un pandas. La diferencia fue que se extrajo sólo una de las columnas de esta estructura, generando así una serie. Cada entrada representa un valor cada 4 horas, que en este caso es el precio de la criptomoneda para ese momento espacial.

- ➢ Al ser la primera prueba, se decidió generar un dataset reducido de unos 20 sets del total de datos. Este dataset se formó añadiendo 6 valores cada vez desde el inicio de los datos, desplazándose un paso a cada iteración. El motivo de ésto, es debido a que con la repetición de valores, el rango de fallo se acorta, pues no hay tanta variedad o alteración numérica.
- ➢ Se alteró este dataset como una lista seguida de valores uno detrás del otro.
- ➢ Se creó el modelo ARIMA y se aplicó sobre la lista de valores creada en el paso anterior. Para los parámetros que necesita este modelo, se usó la **p** y la **d** con valores de 6 (ya que cada 6 valores, implica un día y ese es el valor que se quiere predecir) y de 1 (realizar la primera diferencia entre los valores previos).
- ➢ Finalmente, con el modelo aplicado y aprendido sobre el conjunto de datos, se ordenó al algoritmo calcular el sexto valor de cada conjunto de seis datos, y se comparó con el valor real.

Nota3: El resto de pruebas consiguientes incluyendo la realización final de la funcionalidad de predicción, se llevó a cabo con el mismo conjunto de datos (tabla criptomoneda Poloniex, donde cada entrada es un valor cada 4 horas).

```
if __name__ == '__main__':
    conn = MySQLdb.connect(host="127.0.0.1", user="root", passwd="DrStrange", db="TFG")
    sql_command = 'SELECT `date`, weightedAverage FROM btcPOLONIEX_14400'
    df = pd.read_sql(sql_command, con=conn, index_col='date')  # Construction of the Pandas
    new = pd.Series(data=df['weightedAverage'], index=df.index)  # Create a time series from a single df column

    values = new.values  # Takes just the values (prices)
    datasets = list()

    # Construction of the data set. 20 examples of 6 values each
    for i in xrange(0, 20):
        datasets.append(values[i:i+6])

    # Joining of all examples, one after another. We need this to apply the ARIMA model
    the_prices = [value for elem in datasets for value in elem]

    # >>> Fitting of ARIMA model into the prices data
    model = ARIMA(the_prices, order=(6, 1, 0))  # Takes the previous 6 observations to make the prediction. Also takes
    # the difference from the previous value
    model_fit = model.fit(disp=0)
    print(model_fit.summary())

    print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~PREDICTIONS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    for i in xrange(5, len(the_prices), 6):
        predictions = model_fit.predict(i, i)  # Predicts always the 6th value
        print("> The real value is '{}' and the predicted is '{}'".format(the_prices[i], predictions[0]))
```

Figura 4.11: Prueba primera del algoritmo *Machine-learning*.

Desafortunadamente, los resultados obtenidos de la predicción quedaban a años luz de acercarse a los valores reales. Algo se estaba

realizando mal, o no se estaba teniendo en cuenta a la hora de aplicar el modelo.
Tras una investigación larga y tendida, se descubrió un aspecto sumamente importante a la hora de hacer predicciones sobre conjuntos de datos establecidos en el tiempo. Éstos deben ser "estacionarios". Esto quiere decir, que para que un modelo de predicción sea medianamente certero, necesita que los datos que usa sean lo más homogéneos posible (sin grandes alteraciones y/o contrastes), sin embargo muchos conjuntos de datos pueden contener trazos de tendencias o estacionalidad (alteraciones) que los convierten en demasiado heterogéneos para el modelo predictivo. A este tipo de conjuntos de datos se les denomina "no estacionarios".

Por lo tanto, la siguiente prueba que se realizó fue comprobar si el conjunto que se estaba usando era estacionario o no.

Para averiguar ésto existen numerosas formas, pero la que se eligió usar es considerada como la más certera y segura de saber si un conjunto de datos es estacionario o no, las Pruebas Estadísticas a través del test *"Dickey-Fuller".* Para resumir, este método consta de 2 hipótesis: una nula, h0 (los datos no son estacionarios); y h1 (los datos son estacionarios). Éste calcula y devuelve un "p-value" el cual se contrasta con un valor de umbral que normalmente suele ser de 5% ó 1%. Si dicho "p-value" queda por debajo del umbral, la hipótesis nula no se puede rechazar, por lo que indica que los datos no son estacionarios. Todo lo contrario si queda por encima del umbral.
Así pues, se procedió a ejecutar el método de Dickey-Fuller.

Averiguado que el conjunto de datos no era estacionario, era el momento de alterarlo para eliminar el componente de "trend" (se recuerda que los datos son del histórico de las criptomonedas, por lo tanto los datos son de *trading*)y homogeneizarlos.
Para ello, se aplicaron 2 métodos. El primero fue calcular el logaritmo natural del conjunto de los datos, y posteriormente como segundo método, realizar la primera diferencia entre los datos, es decir, el nuevo valor en ese preciso instante de tiempo es calculado como la resta entre el valor original y el valor anterior a él.
Así, se ejecutó de nuevo el modelo ARIMA sobre ambos conjuntos de datos (no estacionario y estacionario), para comparar los resultados.
Posteriormente, una vez calculada la predicción, se le aplicó el exponencial para reconvertirlo a la escala en la que están los datos.

```
if __name__ == '__main__':
    conn = MySQLdb.connect(host="127.0.0.1", user="root", passwd="DrStrange", db="TFG")
    sql_command = 'SELECT `date`, weightedAverage FROM btcPOLONIEX_14400'
    df = pd.read_sql(sql_command, con=conn, index_col='date')  # Construction of the Pandas
    new = pd.Series(data=df['weightedAverage'], index=df.index)  # Create a time series from a single df column

    values = new.values  # Takes just the values (prices)
    datasets = list()

    # Construction of the data set. 20 examples of 6 values each
    for i in xrange(0, 20):
        datasets.append(values[i:i+6])

    # Joining of all examples, one after another. We need this to apply the ARIMA model
    the_prices = [value for elem in datasets for value in elem]
    print to_matrix_format(datasets)

    # >>> First fitting of ARIMA model into the raw prices (non stationary)
    model = ARIMA(the_prices, order=(6, 1, 0))  # Takes the previous 6 observations to make the prediction. Also takes
    # the difference from the previous value
    model_fit = model.fit(disp=0)
    print(model_fit.summary())

    print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~PREDICTIONS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    for i in xrange(5, len(the_prices), 6):
        predictions = model_fit.predict(i, i)  # Predicts always the 6th value
        print("> The real value is '{}' and the predicted is '{}'".format(the_prices[i], predictions[0]))


    # >>> Second fitting of ARIMA model into the transformed prices (stationary)
    log_prices = np.log(the_prices)  # We apply a normal log in order to subtract trend
    model = ARIMA(log_prices, order=(6, 1, 0))  # Same as above
    model_fit = model.fit(disp=0)
    print(model_fit.summary())

    print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS 2 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    for i in xrange(5, len(log_prices), 6):
        predictions = np.exp(model_fit.predict(i, i, typ='levels'))  # Same as above, but transforms the data back into
        # normal (exponential)
        print("> The real value is '{}' and the predicted is '{}'".format(the_prices[i], predictions[0]))
```

Figura 4.12: Cálculo y comparación del modelo ARIMA entre los datos no estacionarios y estacionarios.

Esta vez, observando los resultados de la segunda predicción, se pudo decir que los valores se acercaban bastante a los de la realidad.

Quedaba sin embargo una última prueba que realizar. Hasta ahora se había probado el modelo ARIMA con un conjunto pequeño del total de los datos. Era hora de probarlo con el conjunto de training al completo y de realizar las predicciones sobre otro conjunto de datos que el modelo no hubiera visto nunca, llamado conjunto de test.
Cabe aclarar que el set de training correspondía al 90% del total de datos, mientras que el de test al restante 10%.
Los pasos que se ejecutaron para llevar a cabo esta prueba fueron los siguientes:

- Como en las demás pruebas, este punto es idéntico. Extracción y almacenamiento de los datos en un dataframe.
- Se separó los datos entre conjunto de training y de test.

- Se creó y aplicó el modelo ARIMA sobre los datos estacionarios del conjunto de training al completo.
- Se predijo el primer valor fuera del conjunto aprendido, que coincide con el primer valor del conjunto de test. El valor predicho es almacenado en una lista, mientras que el real, se añade al conjunto de training. Se vuelve a aplicar el modelo ARIMA sobre el conjunto, y se repite toda la operación pero cada vez con el nuevo valor real, hasta que se hayan predicho todos los valores del conjunto de test.
- Por último, los valores predichos y los reales se muestran en una gráfica para comparar.

```
20
21      conn = MySQLdb.connect(host="127.0.0.1", user="root", passwd="DrStrange", db="TFG")
22      sql_command = 'SELECT `date`, weightedAverage FROM btcPOLONIEX_14400'
23      df = pd.read_sql(sql_command, con=conn, index_col='date')  # Construction of the Pandas
24      new = pd.Series(data=df['weightedAverage'], index=df.index)  # Create a time series from a single df column
25      the_dates_ = pd.Series(data=df.index)  # Create a time series from a single df column
26
27      # 6324 90% ~~ 701 10%
28
29      values = new.values  # Takes just the values (prices)
30      training, test, the_dates = values[:6324], values[6324:], the_dates_.values[6324:]
31
32
33      # >>> Fitting of ARIMA model into the transformed prices (stationary)
34      log_prices = np.log(training)  # We apply a normal log in order to subtract trend
35      model = ARIMA(log_prices, order=(6, 1, 0))  # Same as above
36      model_fit = model.fit(disp=0)
37      print(model_fit.summary())
38
39      predictions = list()
40
41      for elem in test:
42          predicted = np.exp(model_fit.forecast()[0])[0]
43          predictions.append(predicted)
44          training = np.append(training, elem)
45          log_prices = np.log(training)  # We apply a normal log in order to subtract trend
46          model = ARIMA(log_prices, order=(6, 1, 0))  # Same as above
47          model_fit = model.fit(disp=0)
48
49      trace1 = go.Scatter(
50          x=len(test),
51          y=test,
52          name='Real Values'
53      )
54
55      trace2 = go.Scatter(
56          x=len(test),
57          y=predictions,
58          name='Predicted Values'
59      )
60
61      py.plot([trace1, trace2], filename='test1')
```

Figura 4.13: Prueba final de predicción sobre el conjunto de test.

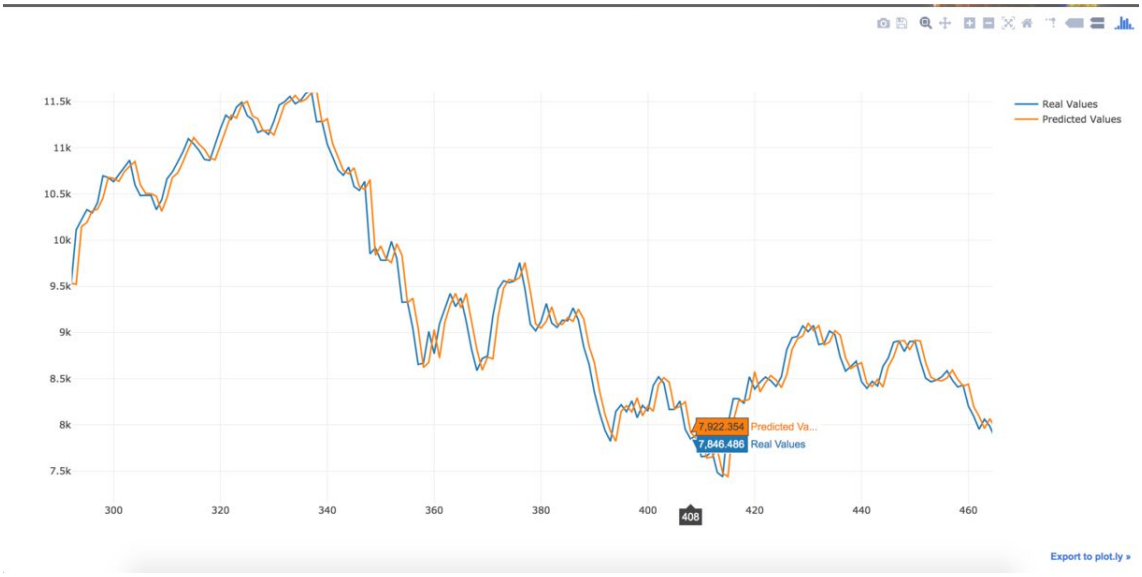


Figura 4.14: Resultado de la gráfica de la última prueba.

Para finalizar, faltaba dotar a la página con la funcionalidad testeada en las últimas pruebas.
Se creó una nueva tabla en la Base de Datos para que contuviera el precio real y el predicho junto con el valor temporal de ese momento. Así, por cada período de 4 horas, el sistema calcularía el valor por predicción de ese momento y añadiría tanto éste como el valor real de ese preciso instante, para poder seguir haciendo predicciones a futuro.
Se ejecutó de nuevo el mismo procedimiento que con la anterior prueba, con la diferencia de que los datos en vez de mostrarse en una gráfica, se guardaron en la nueva tabla creada.
Así, volviendo al apartado visual, al presionar el botón de "forecast", se solicita al servidor el último dato predicho. Esta petición se gestiona a través de la tecnología AJAX de nuevo, pero usando esta vez el método GET, ya que ningún dato se está enviando al lado del servidor. El valor predicho es devuelto como un elemento HTML dentro de un string, que es incluido dentro del fichero "predict.html".
Ocurre prácticamente lo mismo al presionar el botón de "see examples", con la diferencia de que genera una gráfica que se muestra en otra pestaña del navegador.

Nota4: En todo este apartado, se han omitido gran cantidad de elementos, procesos y términos concretos, para evitar que el lector pierda el hilo y deje de comprender lo que se intenta exponer. Para un mayor análisis en detenimiento, se recomienda efusivamente consultar el código de este proyecto.

5. Resultados

En este apartado, se presentarán los resultados visuales obtenidos tras la implementación y desarrollo descritos en el anterior apartado, así como los numéricos, fruto de las pruebas realizadas.

Resultado Visual

Una vez se introduce la *URL* de la página principal (índex), se nos muestra esta pantalla:

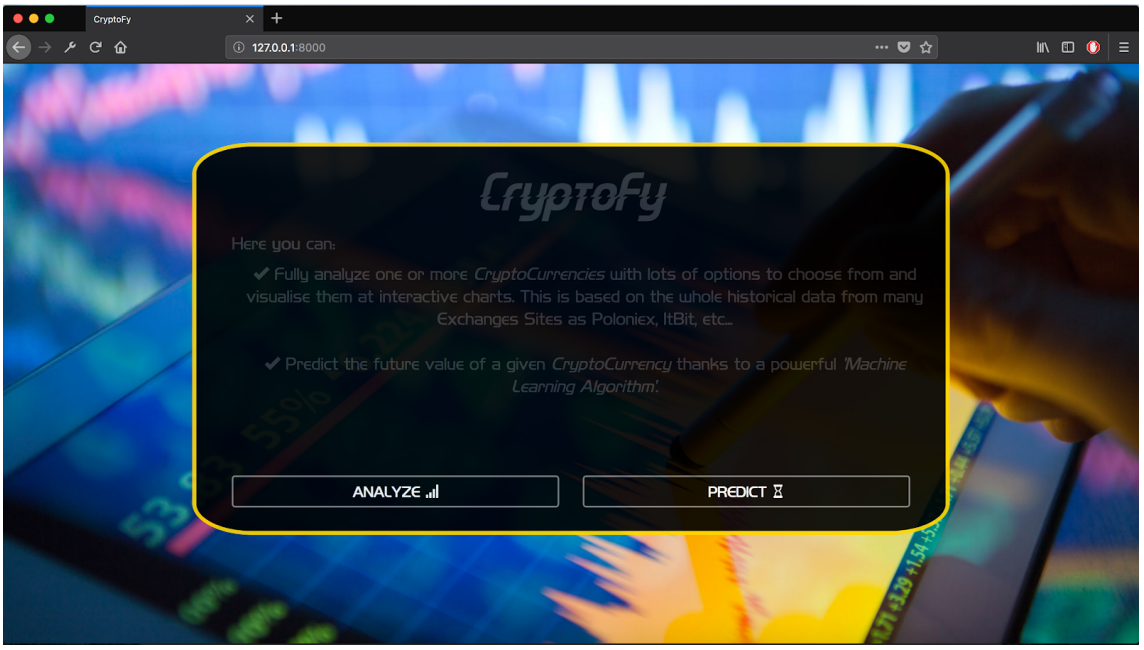
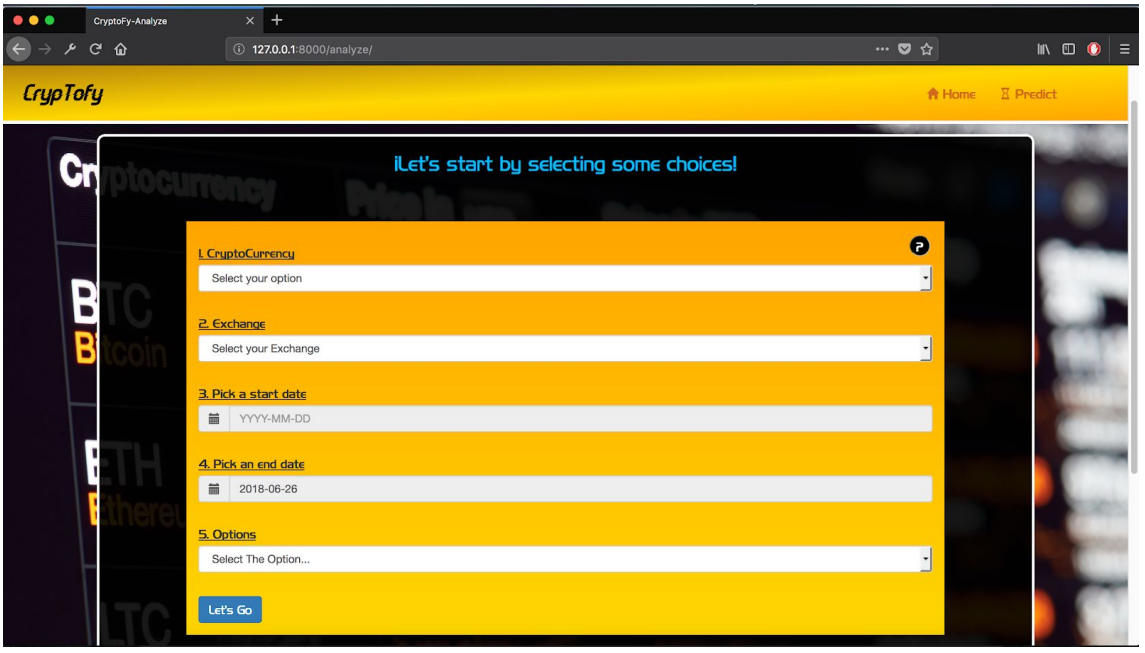
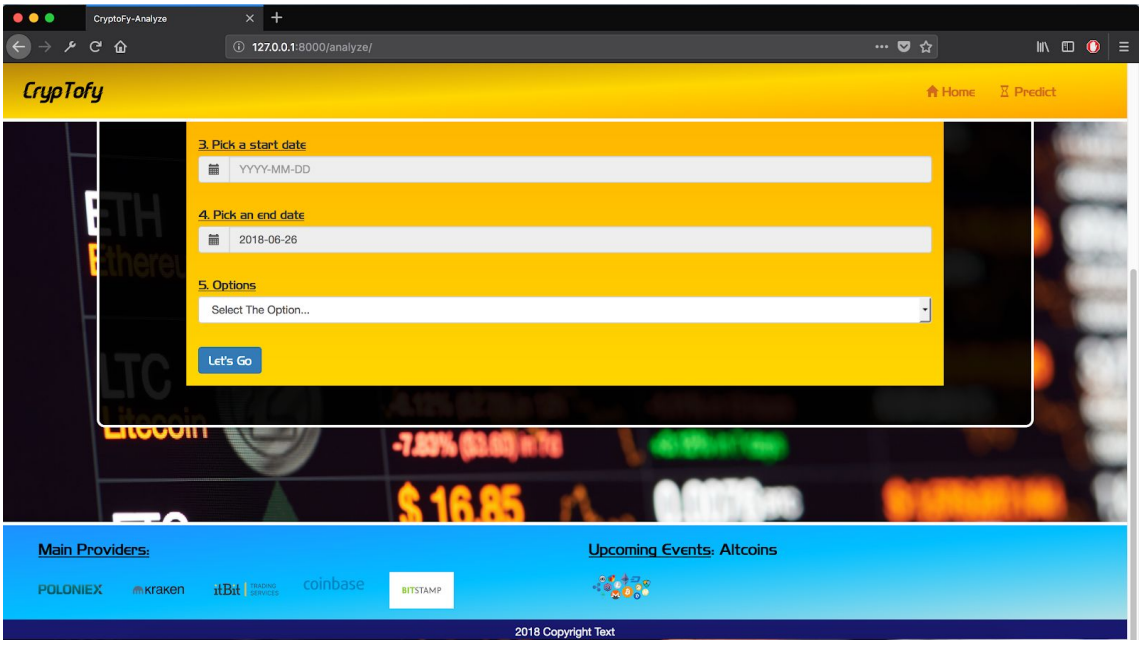


Figura 5.1: Imagen de la página de Index.

Desde aquí, se puede acceder a la página de *Analyze* (análisis de los datos. Primera parte del proyecto), como a la de *Predict* (predicción de valor futuro de una criptomoneda. Segunda parte de la aplicación).

Al pulsar el botón de Analyze (o introducir su URL), se muestra esta otra pantalla:





Figuras 5.2 y 5.3: Imágenes de la página de Analyze.

donde tras rellenar el formulario y presionar el botón de “Let’s Go”, nos muestra (en la misma página) el resultado de nuestra petición en formato de gráfica dinámica, con la opción de volver atrás y rellenar el formulario.

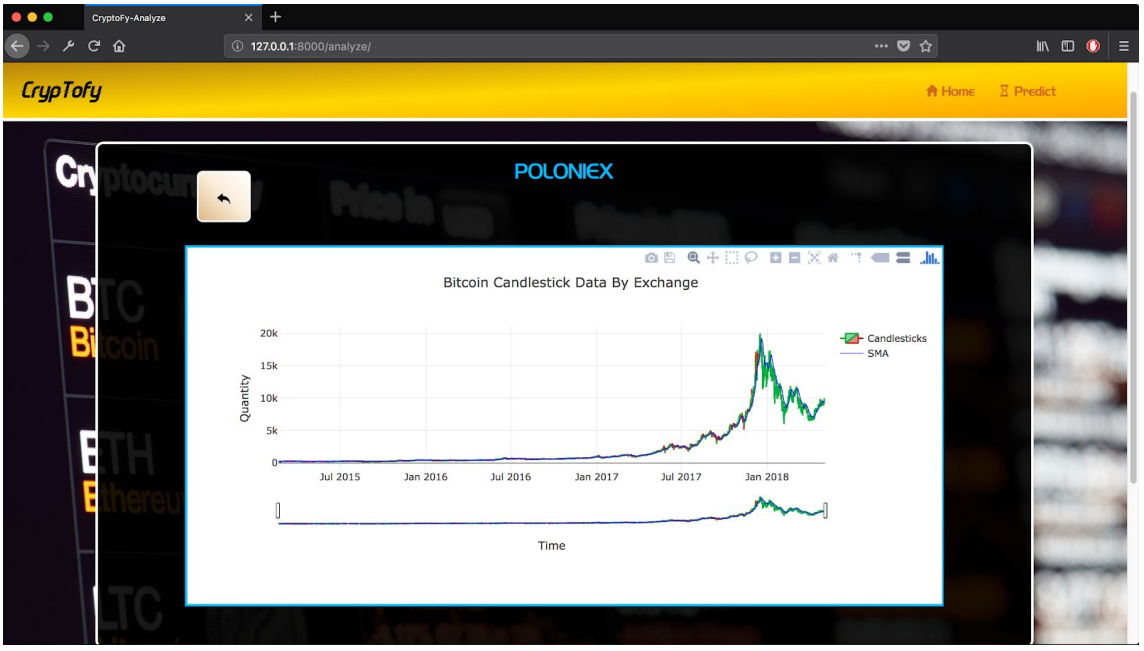
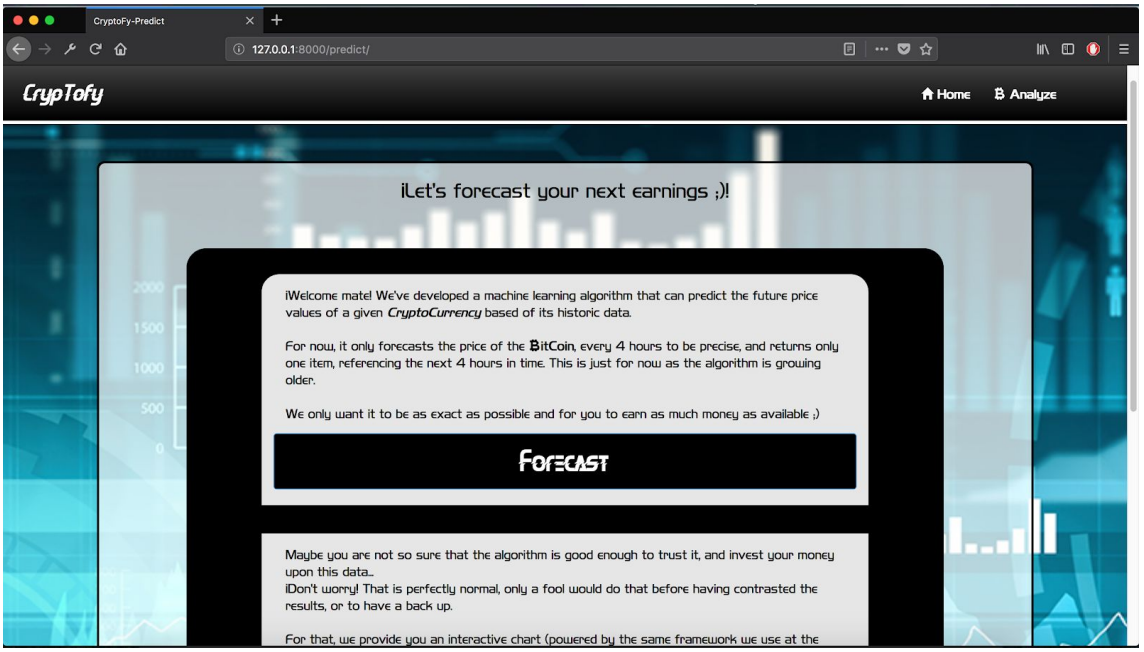
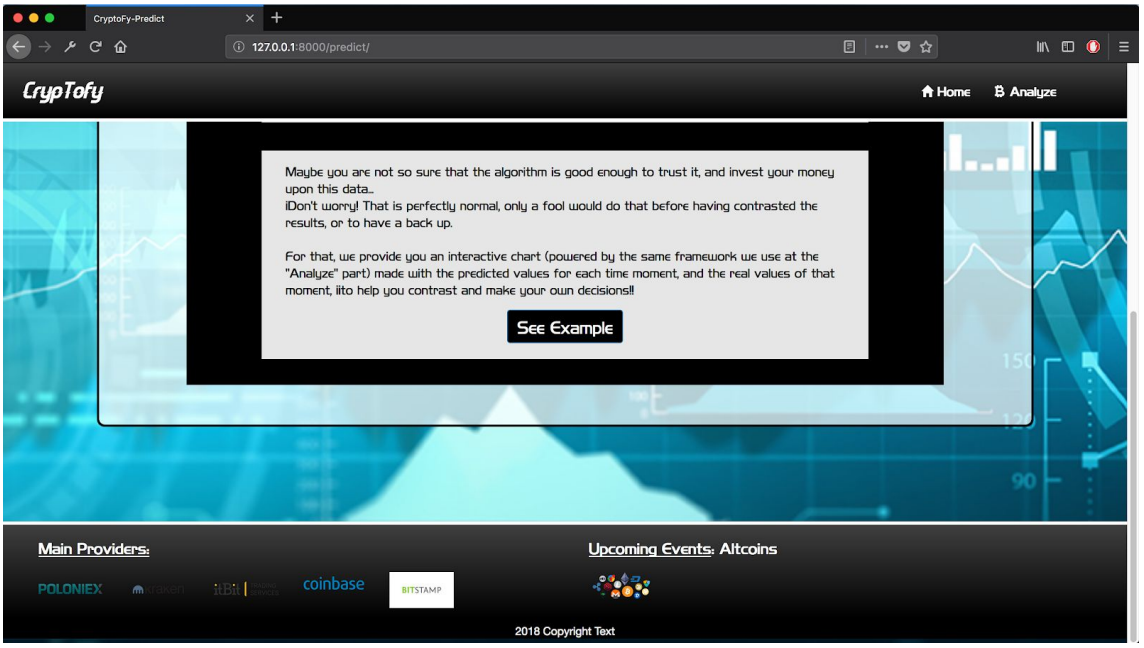


Figura 5.4: Imagen de la página de Analyze.

A partir de aquí o desde la anterior página (índex) o introduciendo la URL correcta, se puede llegar a la página de *Predict.*





Figuras 5.5 y 5.6: Imágenes de la página de Predict.

pudiendo seleccionar la opción de predecir un nuevo valor

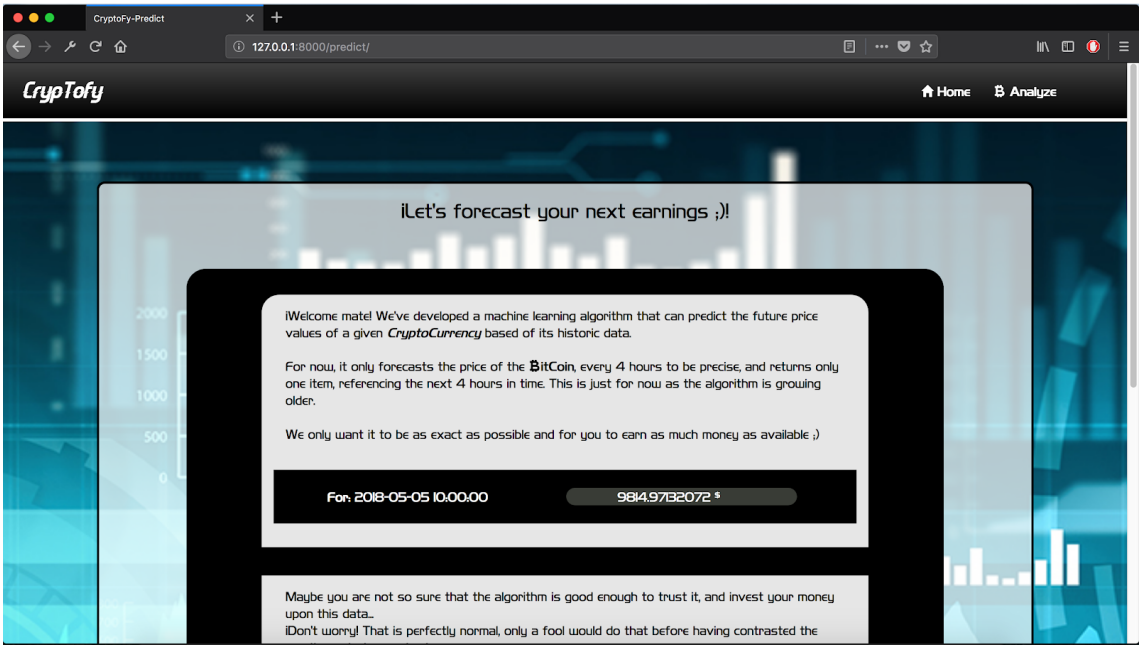


Figura 5.7: Imagen de la página de Predict.

o visualizar el conjunto histórico de valores predichos en una gráfica dinámica mostrada en una nueva pestaña.

Figura 5.8: Resultado de la página de Predict.

Cabe destacar un último aspecto, que no se comentó en ninguno de los anteriores apartados. Cada parte de la aplicación, al ser finalizada, se probó con diversos usuarios con el objetivo de encontrar errores, o cambiar aspectos visuales.

Resultado Numérico

Esta parte corresponde concretamente a los efectos generados en las pruebas realizadas durante la segunda parte del proyecto que corresponde a la aplicación del modelo ARIMA sobre el conjunto de datos.

La primera imagen que se muestra, hace referencia a la primera prueba realizada por este modelo, aplicado a un conjunto de datos no estacionarios. En ella se puede observar el gran fallo de aproximación respecto a los valores reales.

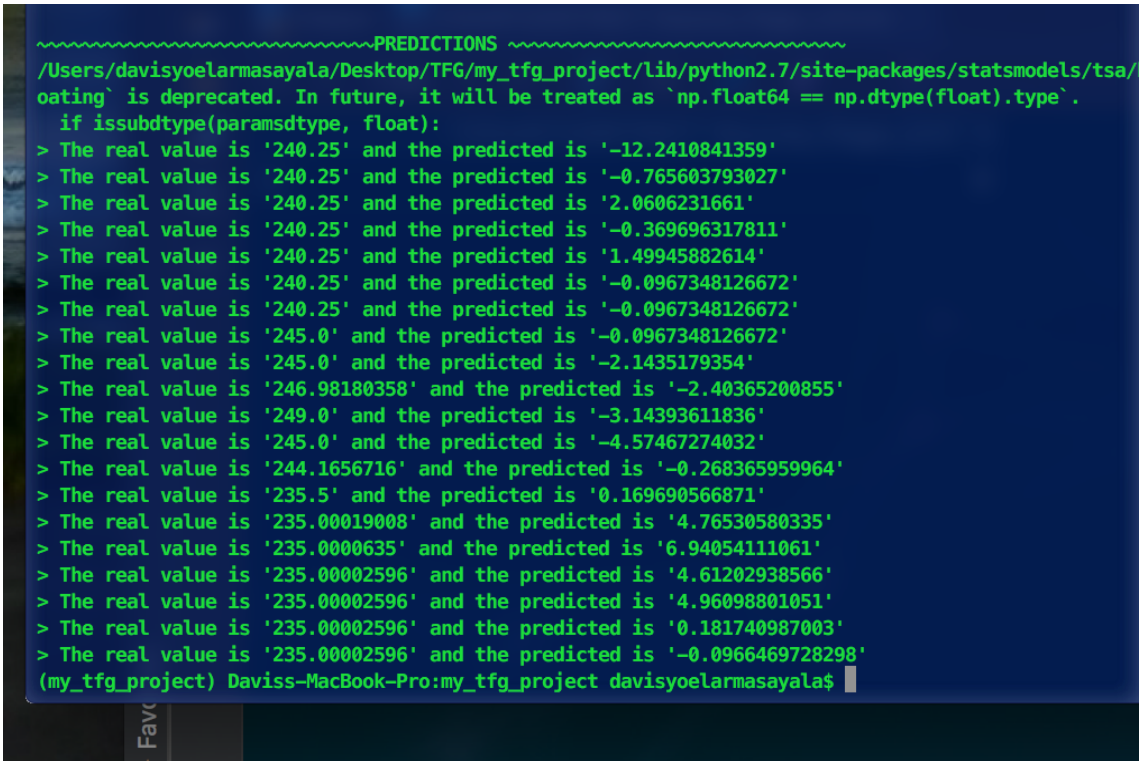


Figura 5.9: Resultado de la primera prueba con el modelo ARIMA.

La segunda fotografía que se incluye, muestra los resultado de aplicar el método de Dickey-Fuller al conjunto de datos, para determinar si es o no estacionario. Como se puede observar, en ese caso era no estacionario debido a que el "p-value" era mayor a '0.05', por lo cual la hipótesis nula (h0) no se puede rechazar e indica que el conjunto de datos no es estacionario.

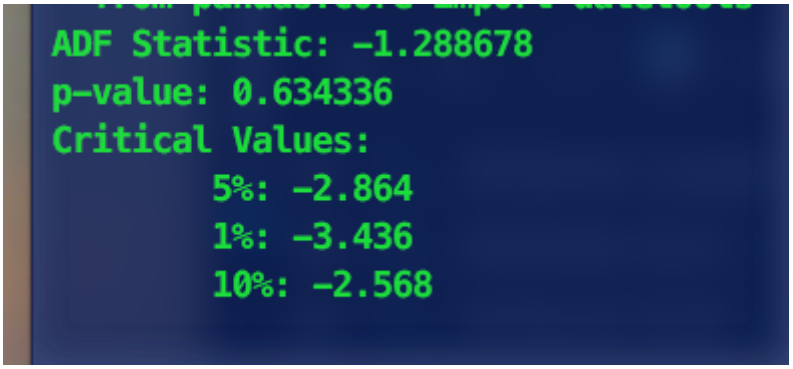


Figura 5.10: Resultado tras ejecutar el método Dickey-Fuller sobre el conjunto de datos.

Por último, la tercera imagen indica los resultados de la prueba realizada sobre datos estacionarios y no estacionarios, comparándolos entre ambos. Se puede ver claramente como en la segunda predicción (datos estacionarios) los resultados predichos se acercan bastante a los reales, con una cota de error de un 2% aproximadamente.

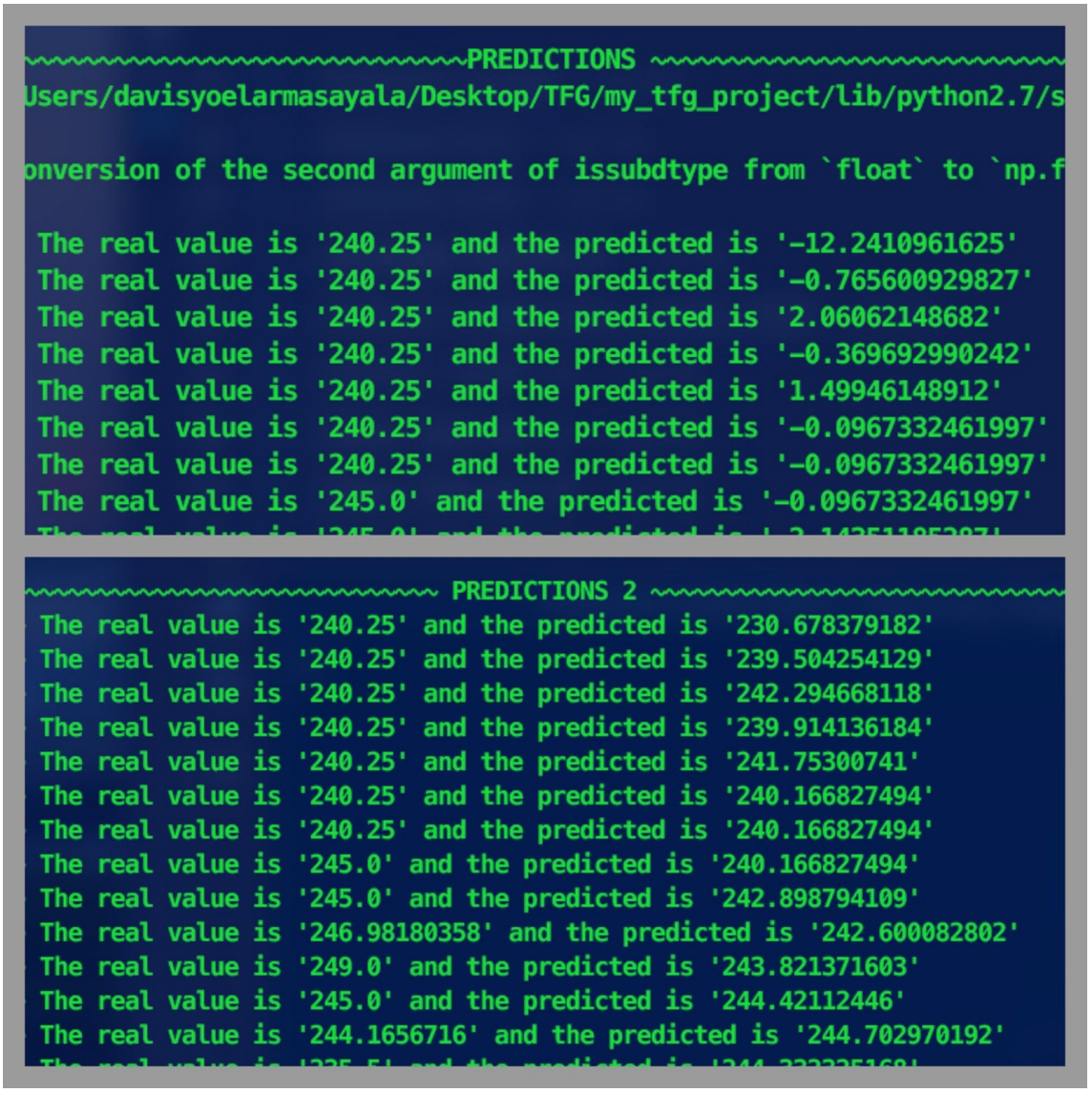


Figura 5.11: Resultado de la segunda prueba con el modelo ARIMA.

6. Conclusiones

Durante los dos primeros meses, se podría decir que me dediqué exclusivamente a tareas de investigación, análisis y autoaprendizaje, debido a la gran cantidad de tecnologías que desconocía y por querer utilizar aquellas herramientas o métodos que pudiesen hacer de mi aplicación lo más robusta, fiable y eficiente posible.
Tras esos meses, me sumergí en una etapa de implementación donde la metodología de prueba y error dominaba, generando alternativas, deshaciendo ideas, o proyectando nuevas. Cuando el resultado parecía encajar con lo demandado, procedía a implementarlo como funcionalidad dentro de la aplicación. Así, paso por paso, hasta que estuvo acabada.

Personalmente, el resultado final de la aplicación no me ha disgustado, pero entre esa peculiaridad que cada uno tiene de autocrítica con lo que hace, y mi actitud perfeccionista, hacen que ni de lejos me satisfaga este resultado debido a que hay multitud de cosas que se podrían mejorar y/o cambiar, al igual que añadir, como también numerosos errores y bugs por solucionar, que desafortunadamente por falta de tiempo, no pude lograr.

Sin embargo, retomando la idea por la cual este proyecto fue llevado a cabo, mi plan es continuar desarrollando y mejorando esta aplicación, para que llegue a un nivel más que aceptable en todos los aspectos, tanto visuales como funcionales, ya que la intención es generar ingresos económicos con ella. Se ampliaría el aspecto de análisis a un mayor número de criptomonedas y exchanges, así como añadiendo más dinamismo e interacción con las gráficas. Se mojoraría el algoritmo machine-learning para que la cota de error fuera lo menor posible y su fiabilidad robusta, dejando un período de prueba al usuario, el cual tendría que renovar con una cuota.

7. Referencias

[1] Explicación Criptomonedas
https://www.ig.com/es/invertir-en-criptomonedas/que-son-las-criptomonedas

[2] Explicación Peer-to-peer
https://en.wikipedia.org/wiki/Peer-to-peer

[3] Explicación Activo Digital
http://sakisgonzalez.com/2014/02/06/que-son-los-activos-digitales/

[4] Explicación AltCoins
https://www.oroyfinanzas.com/2014/10/que-altcoins-criptomoneda/

[5] Explicación Minería (ámbito de las criptomonedas)
https://criptotendencia.com/2017/06/01/que-es-la-mineria-de-criptomonedas/

[6] Diagrama de Casos de Usos
https://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

[7] Documentación de Python
https://www.python.org/doc/

[8] Documentación de MySQL
https://dev.mysql.com/doc/

[9] Documentación de Django
https://docs.djangoproject.com/en/2.0/

[10] Documentación de HTML
https://developer.mozilla.org/en-US/docs/Web/HTML

[11] Documentación de CSS
http://devdocs.io/css/

[12] Documentación de Javascript
https://developer.mozilla.org/bm/docs/Web/JavaScript

[13] Explicación de Algoritmos Machine-Learning
https://machinelearningmastery.com/start-here/

[14] Explicación de modelo ARIMA
https://www.quora.com/What-is-ARIMA

[16] Documentación de Virtualenv
https://virtualenv.pypa.io/en/stable/

[17] Documentación de Bootstrap
https://getbootstrap.com/docs/4.1/getting-started/contents/

[18] Documentación JQuery
https://api.jquery.com/

[19] Explicación de API
https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82

[20] Documentación de JSON
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

[21] Documentación de Plotly
https://plot.ly/api/

[21] Documentación de AJAX (JQuery)
http://api.jquery.com/jquery.ajax/

[22] Explicación de MVC
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[23] Explicación método de Dickey-Fuller
https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test