ANÁLISIS DE PARES DE CRIPTOMONEDAS

Paula Sanjuan Campos e Inés Hernández Pastor

Universidad de Navarra

Python para el análisis de datos

Máster en Big Data Science
Universidad de Navarra

Índice

1. Introducción

Este proyecto ha sido desarrollado por Inés Hernández Pastor y Paula Sanjuan Campos para la asignatura de Python del Máster en *Big Data Science* de la Universidad de Navarra. En esta aplicación web, exploramos las cotizaciones de diferentes criptomonedas, centrándonos en los pares de divisas.

1.1. Objetivos

El objetivo es plasmar en formato web una interfaz de usuario de modo que se pueda interactuar y ver información de uno de los seis pares de criptomonedas previamente seleccionados. Para ello hemos utilizado herramientas de programación en Python.

1.2. Herramientas utilizadas

- Python: versión 3.12.1
- KrakenAPI: la API de Kraken se utiliza para la descarga de datos de cotizaciones actualizadas.
- Entorno virtual: hemos configurado un entorno virtual específico utilizando "virtualenv". Esto asegura que las dependencias y versiones de las bibliotecas utilizadas sean coherentes y evita posibles conflictos con otros proyectos. La versión de las librerías empleadas queda reflejada en "requirements.txt".
- Plotly: librería de Python escogida para la representación de gráficos.
- Streamlit: biblioteca de Python que se utiliza para la creación de la aplicación web interactiva.

2. Información auxiliar

Antes de ejecutar el código se debe adecuar el entorno para una correcta ejecución. Además, una vez ejecutado, aparecerá una ventana del navegador cuyo funcionamiento se introduce en este apartado.

2.1. Configuración del entorno de desarrollo

El proyecto se encuentra en la carpeta comprimida de la entrega, donde previamente se ha configurado todo. Simplemente hay que acceder desde el terminal a la dirección de la carpeta del proyecto, activar el entorno virtual y ejecutar nuestra aplicación en el fichero "main.py". Para ello se ejecutan los siguientes comandos:

```
# activación entorno virtual
# Para MacOs:
source venv/bin/activate
# Para Windows:
venv/Scripts/activate

# ejecución aplicación
streamlit run main.py
```

Además, el proyecto se puede descargar desde GitHub en el siguiente enlace de dos maneras diferentes:

- Clonando el repositorio desde el terminal

  ```
  # Clonar el repositorio
  git clone
  https://github.com/ihernandezp1/Proyecto_final_python_PaulaSanjuan_InesHernandez
  ```

- Descargando el proyecto como un zip desde GitHub

  Se extraen los archivos contenidos en la carpeta comprimida llamada 'Proyecto_final_python_PaulaSanjuan_InesHernandez-main'. Se entra a la carpeta del proyecto desde el terminal y se configura el entorno virtual.

Llegado a este punto, simplemente quedaría configurar y activar el entorno virtual, y ejecutar la aplicación como se ha especificado anteriormente.

2.2. Interfaz de usuario

En este apartado se realiza una breve descripción de la interfaz de usuario resultante de nuestro programa. Más adelante se especifica el funcionamiento detalladamente, junto con capturas de pantalla del resultado.

En la parte izquierda de la página que se abre en el navegador, se encuentra el Menú Principal y dentro de este, el índice en el que se selecciona la pestaña que se quiere visualizar: “Introducción“, “Cotizaciones“ o “Indicadores“.

- Descarga de cotizaciones: el proyecto permite descargar las cotizaciones actualizadas de la plataforma Kraken para el par de monedas seleccionado por el usuario. La información descargada será la base para el análisis posterior.
  En la vista de “Cotizaciones“ se selecciona el par a representar.

- Gráficas de movimiento: una vez se hayan descargado las cotizaciones, se podrá visualizar directamente el movimiento histórico del par de monedas mediante un gráfico de velas. Esta representación gráfica facilitará la identificación de patrones y tendencias en el comportamiento de las cotizaciones.
  Por lo tanto, en la misma vista de “Cotizaciones“, una vez se selecciona el par, se puede observar directamente el gráfico de velas y justo debajo el *DataFrame* en el que se basa la representación.

- Oscilador estocástico: adicionalmente, el proyecto incluirá la generación de gráficas del oscilador estocástico, un indicador técnico ampliamente utilizado en el análisis bursátil. Esta herramienta proporcionará señales sobre la posible sobrecompra o sobreventa del par de monedas, ayudándonos a tomar decisiones informadas.

En la vista de “Indicadores“ se selecciona de nuevo el par de monedas y se puede observar la cotización del par (precio de cierre en el tiempo), el indicador estocástico y la media móvil del estocástico. Además, se realiza una gráfica combinada de los anteriores para visualizarlos de forma simultánea. Finalmente, se representa la cotización sobre su media móvil.

2.3. Archivos de la carpeta comprimida.

Además de los ficheros que contienen el código principal, en la carpeta se pueden observar otros archivos auxiliares. Entre ellos, se destaca “.gitignore”, donde se especifican los archivos que no se suben a GitHub al ejecutar el comando “*push*".
Por otro lado, se ha desarrollado un “README.md”, en formato *MarkDown*, de tal manera que queden explicadas las funcionalidades de la interfaz de usuario, así como la puesta en marcha del programa. Igualmente, en esta memoria también se desarrollan más extensamente estos puntos.

3. Descripción del código

En este apartado se explica de forma detallada la estructura del código, que está organizado en 3 ficheros: “data.py”, “indicators.py” y “main.py”.

- “data.py”: se describe la clase *Data*
- “indicators.py”: se describe la clase *Indicators*
- “main.py”: se ejecuta para abrir la aplicación web

Estos tres están relacionados, de modo que al ejecutar el fichero “main.py” se hacen llamadas a los demás. Posteriormente, se explican las características de cada una de las clases creadas.

3.1. Librerías importadas

Para la correcta ejecución del proyecto es necesario importar las librerías especificadas a continuación. Entre estas cabe destacar “krakenex” y “KrakenAPI”, esta última es muy importante para poder conectarnos a la API de Kraken ya que proporciona los permisos necesarios para la descarga de datos. Por otro lado, empleamos “Pandas” para la manipulación de los datos descargados en formato *DataFrame*, “Plotly” para la representación gráfica y “Streamlit” para la configuración de la interfaz de usuario.

- Data.py:

```
import pandas as pd
import krakenex
from pykrakenapi import KrakenAPI
import plotly.graph_objects as go
```

- Indicators.py:

```
import plotly.graph_objects as go
```

- Main.py:

```
import streamlit as st
from data import Data
from indicators import Indicators
```

3.2. Descarga de datos

Para la obtención de los datos, desde el fichero “data.py”, se configura la clase *Data*. En la siguiente imagen se puede observar el constructor, de modo que los objetos de tipo *Data* tengan como atributos: la API, que se inicializa a través del módulo de “krakenex”; el *DataFrame* de los datos descargados; otro *DataFrame* de los precios de cierre; y por último el par de criptomonedas correspondiente. Estos últimos atributos se inicializan a *None*, ya que se configuran más adelante.

```
7   class Data:
8       def __init__(self):
9           self.api = krakenex.API()
10          self.data = None  # DataFrame con los datos descargados
11          self.data_close = None  # DataFrame con los precios de cierre
12          self.pair_type = None  # Tipo de par representado
```

En primer lugar, en esta clase se encuentra la función *set_pair_type(),* que sirve para inicializar el atributo *pair_type* de la clase *Data*. Para ello se toma el par seleccionado por el usuario desde la interfaz gráfica (*pair*).

```
14      def set_pair_type(self, pair):
15          pair1 = pair.split('/')[0]
16          pair2 = pair.split('/')[1]
17          self.pair_type =  pair1+pair2# Ejemplo: 'XBT/USD' -> 'USD'
```

En segundo lugar, encontramos la función *get_data().* Mediante esta se descargan los datos a través de la función *get_ohlc_data()* de la API de Kraken y se establecen los parámetros del tipo de moneda que se quiere descargar, el intervalo de tiempo y el orden. Establecemos el valor 1.440 en el intervalo, que indica los minutos que pasan entre una salida de datos y la siguiente. Por lo tanto, obtenemos una base de datos con 8 columnas (“*time”, “open”, ”high”, ”low”, ”close”, ”vwap”, ”volumn”, ”count”*) y 720 registros.

Se comprueba que la descarga ha sido realizada a través de una excepción, pues debemos asegurarnos de que la carga de datos se ha realizado de manera correcta. Se ha añadido en este apartado porque lo consideramos crucial para el correcto funcionamiento del código. Se comprueba que la descarga de datos ha sido correcta y, en caso de no ser así, se levanta una excepción.

Los datos descargados son de tipo tupla, por lo tanto, para facilitar su manejo se transforman a *DataFrame*. Además del *DataFrame* global, como en el ámbito bursátil se trabaja con los precios de cierre, hemos considerado relevante crear un nuevo *DataFrame "*data_close*"* que los contenga. De esta manera, ya se han inicializado todos los atributos de la clase *Data*.

```
19      def get_data(self,pair):
20          try:
21              k = KrakenAPI(self.api)
22              ohlc_result = k.get_ohlc_data(self.pair_type, interval = 1440, ascending = True)
23              if ohlc_result==None:
24                  raise Exception(f"Error en la solicitud OHLC: {ohlc_result['error']}")
25
26              self.data = pd.DataFrame(ohlc_result[0])
27              self.data_close = self.data['close']
28          except Exception as e:
29              print(f"Error al obtener datos OHLC: {e}")
```

Una vez obtenidos los datos, el siguiente paso es comprenderlos y ver si es necesario llevar a cabo alguna transformación. Bajo nuestro criterio las columnas *"vwap"* y *"count"* pueden ser eliminadas ya que no aportan información relevante. Seguidamente, se comprueba la presencia de valores ausentes con la función *isnull()* y, en caso de existir, estos se rellenan con el valor del registro anterior mediante *ffill()*.

```
35      def data_clean(self):
36          self.data.drop('vwap', axis=1, inplace=True)
37          self.data.drop('count', axis=1, inplace=True)
38          if self.data.isnull().sum().sum() != 0:
39              print(self.data.isnull().sum())
40              self.data.ffill() # toma el valor anterior de la columna
41          return self.data
```

3.3. Visualización de las cotizaciones

Como introducción, para una primera idea del comportamiento del par seleccionado, hemos realizado el gráfico de velas mediante *Candlestick()* de la librería "Plotly". En este se observa de un vistazo información muy relevante para el análisis.

```
43      def candle_graph(self):
44          # gráfica
45          fig = go.Figure(data = [go.Candlestick(x=self.data.index,
46                                                 open=self.data['open'],
47                                                 high=self.data['high'],
48                                                 low=self.data['low'],
49                                                 close=self.data['close'])])
50          # fig.show()
51          return fig
```

3.4. Obtención y visualización de indicadores técnicos

Para la obtención de los indicadores técnicos, desde el fichero “indicators.py” se configura la clase *Indicators*. En su constructor se inicializa el único atributo de esta clase, que es el *DataFrame* global. En la siguiente imagen podemos observar que primeramente se inicializa a *None*, pero que con la función *set_indicators()*, se actualiza a través del *input* “data”, que es el *DataFrame* procesado en el punto anterior con los métodos de la clase *Data*.

```
4   class Indicators:
5       def __init__(self):
6           self.dfdata = None
7
8       def set_indicators(self, data):
9           self.dfdata = data
```

En el primer gráfico, mediante la función *line_graph()* se extrae la cotización (precio de cierre) del par seleccionado y se representa mediante un gráfico de líneas.

```
11      def line_graph(self):
12          fig = go.Figure()
13          fig.add_trace(go.Scatter(x = self.dfdata.index,
14                                   y = self.dfdata["close"], mode='lines'))
15          return fig
```

En el segundo gráfico con la función *stochastic_graph()* se grafica el estocástico, calculado de la siguiente forma:

$$estocástico = \frac{(close\ -low)}{(high-low)}\ x\ 100$$

```
18      def stochastic_graph(self):
19          # indicadores estocásticos
20          # cálculo de estocástico => estocástico =  (close - low)/(high-low)
21          close = self.dfdata["close"]
22          low = self.dfdata["low"]
23          high = self.dfdata["high"]
24          self.dfdata["Estocastico"] = 100*(close-low)/(high-low)
25
26          fig = go.Figure()
27          fig.add_trace(go.Scatter(x = self.dfdata.index,
28                                   y = self.dfdata["Estocastico"], mode='lines'))
29          return fig
```

En tercer lugar, con la función *avg_graph(),* se calcula la media móvil del estocástico para suavizar el comportamiento del gráfico del indicador y así poder extraer conclusiones más eficientes.

La media móvil se obtiene a través de un *rolling* con un período de 30, ya que se ha considerado interesante resaltar tendencias y patrones mensuales.

```
31      # grafico de media movil del precio de cierre
32      def avg_graph(self):
33          period = 30
34          self.dfdata['media_movil'] = self.dfdata['Estocastico'].rolling(window=period).mean()
35          self.dfdata['media_movil_close'] = self.dfdata['close'].rolling(window=period).mean()
36          fig = go.Figure()
37          fig.add_trace(go.Scatter(x = self.dfdata.index,
38                                   y = self.dfdata["media_movil"], mode='lines'))
39          return fig
```

Por último, se obtienen gráficos combinados para ver la información de forma simultánea y contrastarla. Por un lado, con la función *mixed_graph()* combinamos el estocástico con su media móvil y, por otro lado, con la función *avg_close()* combinamos la cotización con su media móvil.

```
42      # grafico conjunto de media movil y estocastico
43      def mixed_graph(self):
44          fig = go.Figure()
45          fig.add_trace(go.Scatter(x = self.dfdata.index,
46                                   y = self.dfdata["Estocastico"],
47                                   mode='lines',
48                                   name = 'Estocástico',
49                                   line=dict(color='#0077cc')))
50          fig.add_trace(go.Scatter(x = self.dfdata.index,
51                                   y = self.dfdata["media_movil"],
52                                   mode='lines',
53                                   name = 'Media móvil',
54                                   line=dict(color='red', width=2)))
55
56          return fig
57
58      # grafico conjunto de media movil y precios de cierre
59      def avg_close(self):
60          fig = go.Figure()
61          fig.add_trace(go.Scatter(x = self.dfdata.index,
62                                   y = self.dfdata["close"],
63                                   mode='lines',
64                                   name= 'Precio de cierre',
65                                   line=dict(color='#0077cc')))
66          fig.add_trace(go.Scatter(x = self.dfdata.index,
67                                   y = self.dfdata["media_movil_close"],
68                                   mode='lines',
69                                   name = 'Media móvil',
70                                   line=dict(color='red', width=2)))
71
72          return fig
```

Además, como se indica en los requerimientos del enunciado del proyecto, se obtiene el gráfico combinado de las cotizaciones con la media móvil del estocástico. Este no lo consideramos significativo ya que no aporta información relevante. Decidimos prescindir de él por este motivo, pero el código para su ejecución es el siguiente:

```
    # def sthoc_close(self):
    #     fig = go.Figure()
    #     fig.add_trace(go.Scatter(x = self.dfdata.index,
    #                              y = self.dfdata["close"],
    #                              mode='lines',
    #                              name= 'Precio de cierre',
    #                              line=dict(color='#0077cc')))
    #     fig.add_trace(go.Scatter(x = self.dfdata.index,
    #                              y = self.dfdata["Estocástico"],
    #                              mode='lines',
    #                              name = 'Estocástico',
    #                              line=dict(color='red', width=2)))
```

3.5. Configuración de la interfaz de usuario con Streamlit

En el fichero “main.py”, se crea la clase *App*, la cual tiene como atributos un objeto de la clase *Data* y otro de la clase *Indicators*.

```
class App:
    def __init__(self):
        self.data = Data()
        self.indicators = Indicators()
```

En esta clase hay tres funciones que configuran las diferentes pestañas de la interfaz. Este apartado no profundiza en la explicación del código, sino que se centra en el resultado final de la interfaz. Por lo tanto, el código de este fichero consiste en llamadas a los métodos o funciones anteriormente explicados de las clases *Data* e *Indicators*, además del uso de la librería “Streamlit”.

En el código que vemos a continuación se escriben las cabeceras y el texto que aparece en cada vista.

```
    def intro_page(self):
        st.header('Introducción')
        st.write("¡Hola! Nos complace darte la bienvenida a nuestro proyecto de la asignatura de
        st.write("En esta aplicación web, exploramos las cotizaciones de diferentes criptomonedas
        st.write("Nuestro proyecto tiene como objetivo analizar y visualizar la relación entre di
        st.write("Este proyecto ha sido desarrollado por Inés Hernández y Paula Sanjuan.")

    def cot_page(self):
        st.header("Cotizaciones")
        st.write('''En esta segunda página se puede ver el gráfico de velas del par seleccionado,

    def ind_page(self):
        st.header("Indicadores")
        st.write("En esta tercera ventana podemos ver la representación gráfica de la cotización
```

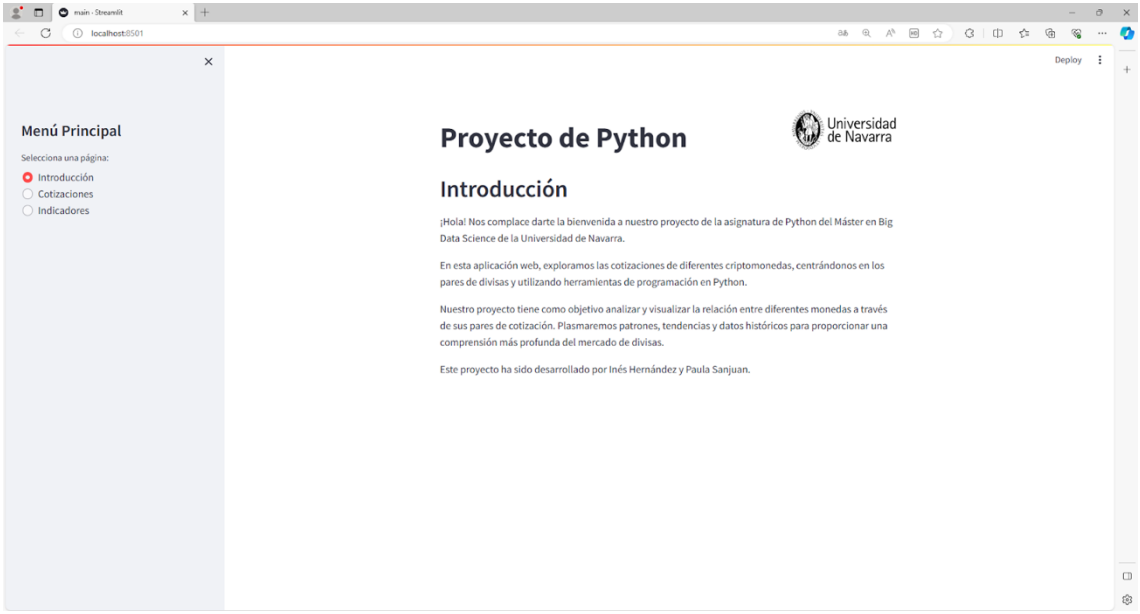
Justo después, se crea la función de ejecución *run()*. En esta se establece el formato de página, empezando por la cabecera, el título y el logo. Para estructurar la página se decide dividir el espacio en 2 columnas, de modo que la primera ocupa 3/4 de la anchura y la segunda 1/4. Se realiza mediante el comando ".columns()" y con la finalidad de mejorar la estética de la página. Por otro lado, con el comando ".sidebar()" se implementa el navegador del Menú Principal. El contenido principal de la página se implementa en un contenedor obtenido mediante la función *container()* de "Streamlit".

```
27      def run(self):
28          col1, col2 = st.columns([3, 1])  # La primera columna ocupa 3/4 del ancho, la segunda 1/4
29          with col1:
30              st.title("Proyecto de Python")
31          with col2:
32              # Imagen en la esquina superior derecha
33              st.image('logo_unav.png', use_column_width=True)
34
35          st.sidebar.title("Menú Principal")
36          page = st.sidebar.radio("Selecciona una página: ", ["Introducción", "Cotizaciones", "Indicadores"])
37          content = st.container()
```

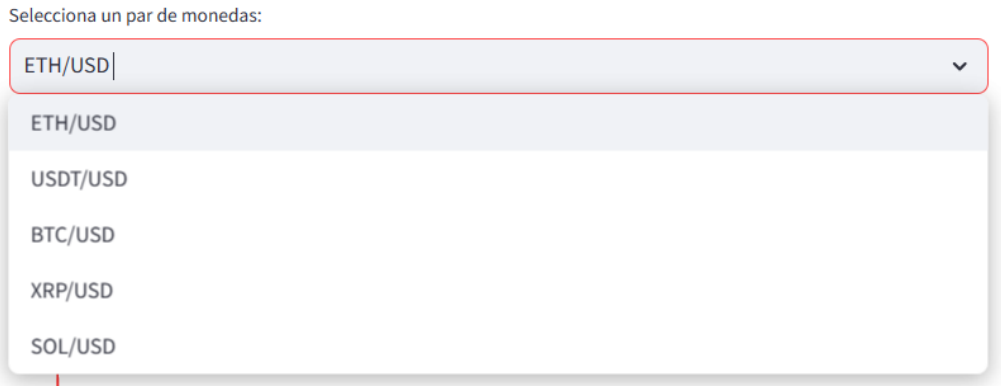
En caso de seleccionar la pestaña "Introducción" se ejecuta la función *intro_page()* que contiene la información ya citada.

```
38          with content:
39              if page == "Introducción":
40                  self.intro_page()
```

A continuación, se adjunta el resultado de la pestaña de "Introducción".



Si se selecciona la pestaña “Cotizaciones”, se debe elegir un par del desplegable.



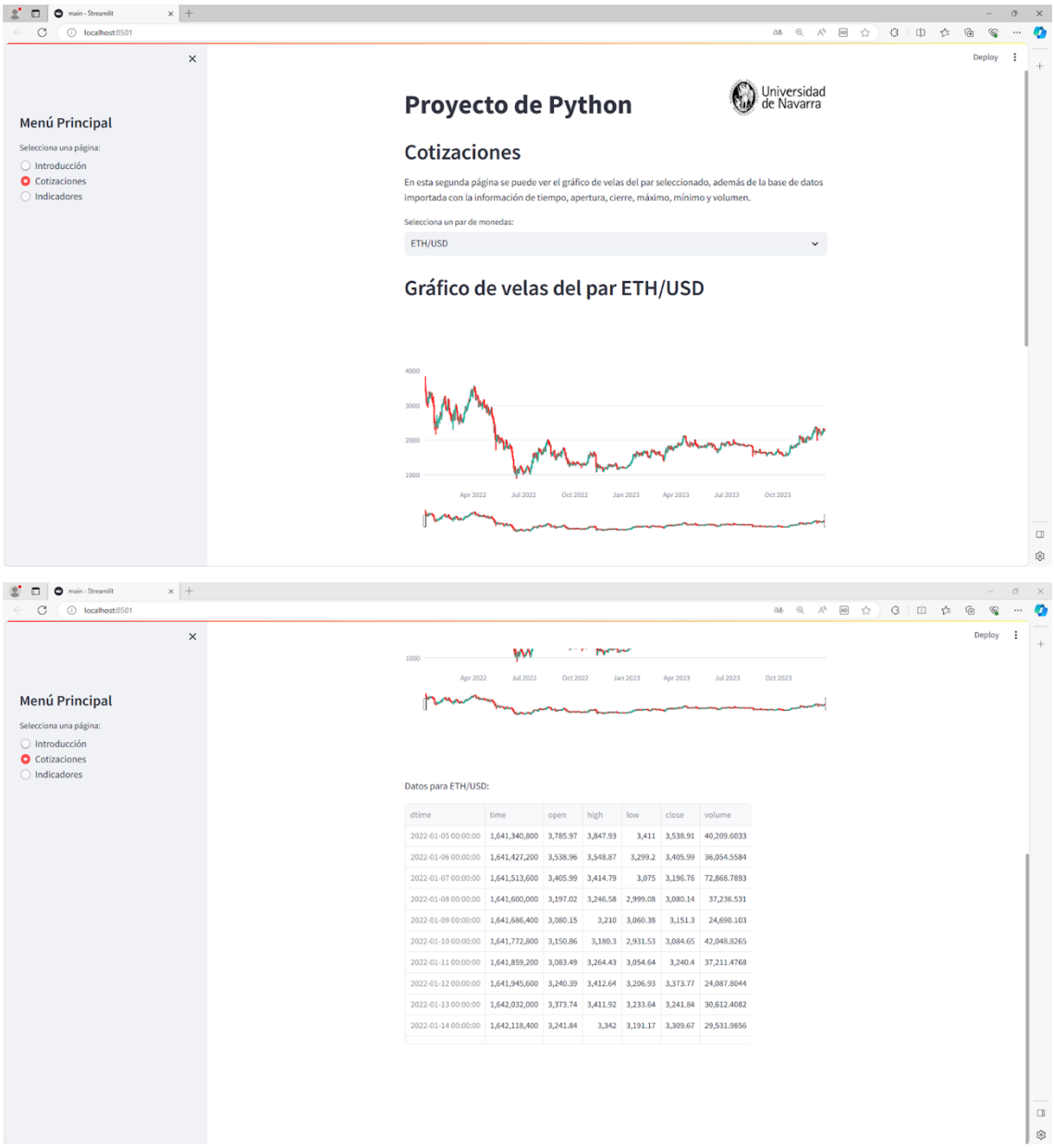
Como consecuencia, se visualizará tanto el gráfico de velas como la tabla de datos correspondiente. Para plasmar el desplegable en la interfaz en el código se ha utilizado la función *selectbox()*, donde simplemente se indica el título del desplegable y las diferentes opciones.

Para la representación de los gráficos se utiliza la función *plotly_chart()*, que permite representar objetos de tipo *Figure*. Estos son devueltos por los métodos de las clases anteriormente explicadas.

En el siguiente código se pueden ver las llamadas a las funciones.

```
43            elif page == "Cotizaciones":
44                self.cot_page()
45
46                criptos = ('ETH/USD','USDT/USD','BTC/USD','XRP/USD','SOL/USD')
47                selected_pair = st.selectbox('Selecciona un par de monedas: ', criptos)
48                self.data.set_pair_type(selected_pair)
49
50                self.data.get_data(selected_pair)
51                data = self.data.data_clean()
52
53                fig = self.data.candle_graph()
54                st.header(f'Gráfico de velas del par {selected_pair}')
55                st.plotly_chart(fig)
56
57                st.write(f'Datos para {selected_pair}: ')
58                st.write(self.data.data)
```

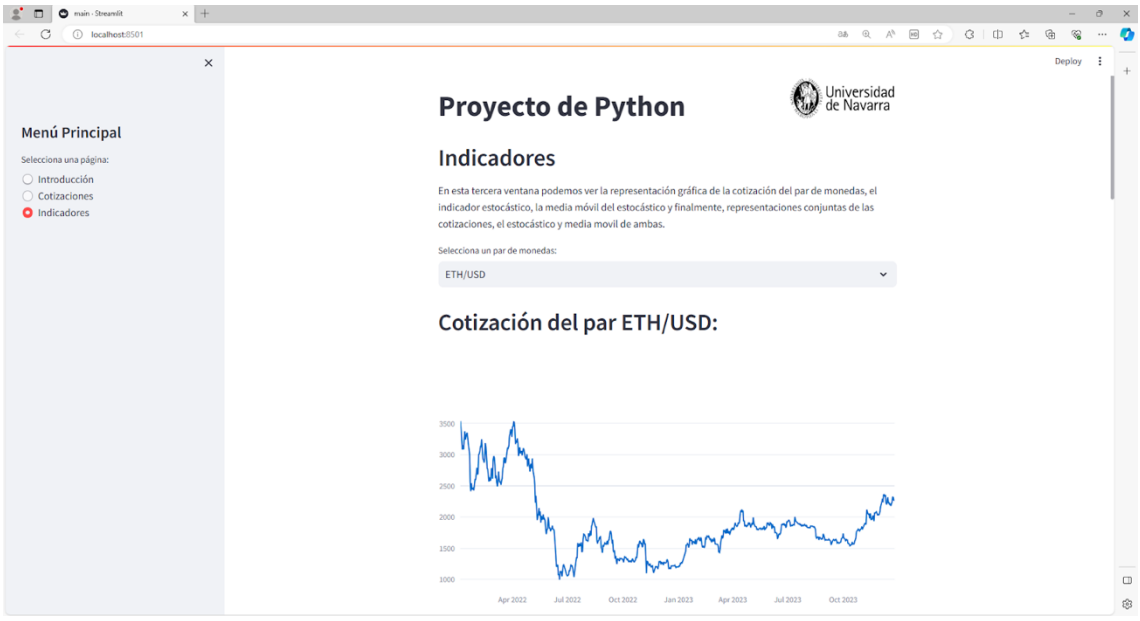
Una vez más, se muestra la salida por pantalla de la pestaña completa.



Por último, si se selecciona “Indicadores”, se elegirá el par y como resultado se mostrarán los indicadores estocásticos pertinentes.

```python
        elif page == "Indicadores":
            self.ind_page()
            criptos = ('ETH/USD','USDT/USD','BTC/USD','XRP/USD','SOL/USD')
            selected_pair = st.selectbox('Selecciona un par de monedas: ', criptos)
            self.data.set_pair_type(selected_pair)

            self.data.get_data(selected_pair)
            data = self.data.data_clean()

            self.indicators.set_indicators(data)

            st.header(f'Cotización del par {selected_pair}: ')
            fig = self.indicators.line_graph()
            st.plotly_chart(fig)

            st.header(f'Estocástico del par {selected_pair}: ')
            fig2 = self.indicators.stochastic_graph()
            st.plotly_chart(fig2)

            st.header(f'Media móvil del estocástico del par {selected_pair}: ')
            fig3 = self.indicators.avg_graph()
            st.plotly_chart(fig3)

            st.header(f'Gráfico conjunto de la Media móvil del estocástico y el Estocástico del par {selected_pair}: ')
            fig4 = self.indicators.mixed_graph()
            st.plotly_chart(fig4)

            st.header(f'Gráfico conjunto de la Media móvil del precio de cierre y Cotización del par {selected_pair}: ')
            fig5 = self.indicators.avg_close()
            st.plotly_chart(fig5)
```

Se muestra la salida por pantalla de la última pestaña.



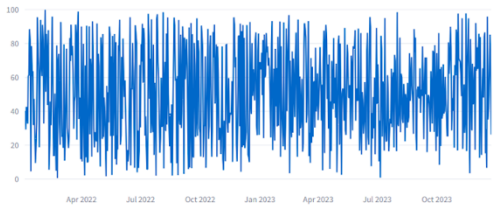
Deploy

Menú Principal

Selecciona una página:

- Introducción
- Cotizaciones
- Indicadores

Estocástico del par ETH/USD:



Menú Principal

Selecciona una página:

- Introducción
- Cotizaciones
- Indicadores

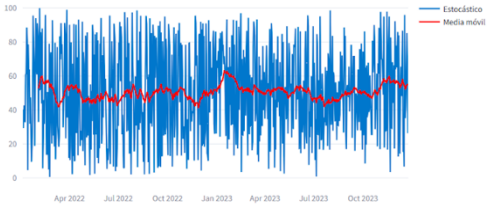
Media móvil del estocástico del par ETH/USD:

Menú Principal

Selecciona una página:

- Introducción
- Cotizaciones
- Indicadores

Gráfico conjunto de la Media móvil del estocástico y el Estocástico del par ETH/USD:



Menú Principal

Selecciona una página:

- Introducción
- Cotizaciones
- Indicadores

Gráfico conjunto de la Media móvil del precio de cierre y Cotización del par ETH/USD:

4. Conclusión

Para la ejecución del proyecto, hemos aplicado los conocimientos obtenidos en la asignatura del máster, complementándolos con información extraída de documentación. Aunque ya habíamos trabajado previamente con herramientas de Python, queremos resaltar la adquisición de habilidades con la librería Streamlit, la cual consideramos sumamente útil, visual y de fácil implementación. Además, destacamos el enriquecimiento de nuestros conocimientos financieros gracias a la utilización de Kraken y al manejo de los datos del proyecto.