Universidad de Navarra

Proyecto de python : TradingView Bitcoin & Ethereum

Mario Lamas Herrera

ÍNDICE DE CONTENIDOS

1. Introducción

El auge de las criptomonedas, encabezado por Bitcoin y Ethereum, ha transformado la industria financiera y la inversión en los últimos años. La volatilidad y el crecimiento exponencial de estas monedas digitales han despertado el interés de inversores, entusiastas y profesionales en todo el mundo. En este contexto, la capacidad de acceder y analizar datos de cotización en tiempo real se ha convertido en un activo invaluable.

El presente proyecto aborda el desarrollo de una aplicación web para visualizar las cotizaciones de ambas divisas respecto a diferentes monedas (Euro, dolar) así como una respecto de la otra, proporcionando la capacidad de análisis de estas mediante diferentes indicadores. A lo largo de esta memoria, se desarrollan los aspectos clave del diseño, desarrollo y funcionamiento de la aplicación así como las decisiones tomadas en cada etapa y su justificación.

2. Diseño y Desarrollo de la Aplicación

El objetivo principal de la aplicación es proporcionar al usuario final una herramienta para interactuar con las cotizaciones de estas divisas de manera fluida y poder realizar un análisis sobre estas. Para ello, extraemos los datos mediante la API de Kraken con el formato adecuado para posteriormente tratarlos y calcular los diferentes indicadores a mostrar, finalmente, a partir de estos datos postprocesados y de la elección de indicadores por parte del usuario, se crearán los gráficos interactivos correspondientes. Estos gráficos serán mostrados en la aplicación empleando el framework de aplicaciones Streamlit, el cual permite fácilmente mostrar gráficos interactivos e introducir otras funcionalidades como botones de elección múltiple de manera sencilla y eficiente en formato web.
En el apartado a continuación, se exponen detalladamente los procesos seguidos en cada una de las fases previamente descritas.

3. Detalles de Implementación

La aplicación se corresponde con un objeto de tipo clase dentro del cual se encuentran definidos los métodos referentes a cada parte del proceso de ejecución.
El método __init__ se llama automáticamente al crear una instancia de la clase, en este definimos por un lado el objeto para llamar a la API de Kraken y por otro lado, una variable intervalos que relaciona el valor de texto elegido por el usuario con su correspondiente valor numérico, la implementación de este método se observa en la figura a continuación.

```python
class App():

    def __init__(self):
        self.kraken=krakenex.API()
        self.intervalos={'1m':1,'15m':15,'30m':30,'1h':60,'24h':60*24}
```

Figura 1. *Método __init__().*

Dividimos el proceso de ejecución en tres fases: Obtención de los datos, generación de gráficos y visualización con Streamlit.

3.1 Fase 1- Obtención de los datos

Una correcta obtención de los datos es fundamental para el adecuado funcionamiento de la aplicación, el método obt_datos() define este proceso. Este método recibe tres parámetros: el par sobre el que se desea obtener información, el intervalo entre datos (Un intervalo de 1 min devuelve los datos minuto a minuto) y una lista con los indicadores empleados para evitar calcular datos acerca de indicadores no seleccionados.

Inicialmente empleando el método query_public() del objeto de llamada a la API de Kraken, indicamos el tipo de dato en este caso 'OHLC' (Open, High, Low, Close), el par escogido y el intervalo, sin embargo, hay que tener en consideración que la API de Kraken retorna un máximo de 720 datos, en consecuencia para asegurar la fluidez de la aplicación se ha decidido emplear únicamente los datos recibidos en cada petición a la API. La figura 2 muestra la nomenclatura de esta petición.

```
ohlc_data = self.kraken.query_public('OHLC', {'pair': par,'interval':intervalo})
```

Figura 2. *Petición a la API de Kraken.*

Esta petición retorna un diccionario con una clave indicando el tipo de error y en caso de encontrar datos retorna una clave 'result' con la información, para evitar que se detenga la aplicación en caso de no recibir datos emplearemos el flujo try-except para en caso de que aparezca un error en la llamada a la clave 'result' se retorne un Dataframe vacío y se informe al usuario que no se han encontrado los datos solicitados.

Una vez obtenidos los datos, creamos un Dataframe con la información, convertimos las columnas a su tipo de dato correspondiente y creamos una columna nueva para indicar el color de las velas que serán verdes en caso de que la cotización se cierre por encima de su apertura y rojas en caso contrario. En la figura 3 se muestra este proceso.

```
try:
    ohlc_list = ohlc_data['result']
    df=pd.DataFrame(ohlc_list[par],columns=['timestamp', 'open', 'high','low','close','_','volume','conteo'])
    df['Date']=df['timestamp'].map(lambda x: datetime.datetime.fromtimestamp(x))
    df['Date_str']=df['Date'].astype(str)
    df['color']=df[['close','open']].apply(lambda x: 'red' if x.open>x.close else 'green',axis=1 )
    df['open']=df['open'].astype(float)
    df['close']=df['close'].astype(float)
    df['high']=df['high'].astype(float)
    df['low']=df['low'].astype(float)
    df['volume']=df['volume'].astype(float)
```

Figura 3. *Preprocesamiento de los datos.*

Finalmente, en función de la elección de indicadores por parte del usuario, se calculan los correspondientes indicadores como se puede observar en la figura 4, una vez calculados los indicadores se finaliza la fase de obtención de los datos para dar lugar a la generación de los gráficos.

```
if 'VWAP' in ind:
    df['Cum_Vol'] = df.iloc[::-1]['volume'].cumsum()
    df['m']=(df['close'])*df['volume']
    df['m_cum']=df.iloc[::-1]['m'].cumsum()
    df['VWAP'] = df['m_cum'] / df['Cum_Vol']

if 'STOCH' in ind:
    df = self.calc_stoch(df,14,3,3)
if 'BANDS' in ind:
    df = self.calc_bbands(df)
return df
```

```
def calc_stoch(self,df,n,smooth_k,smooth_d):
    # Calcular el %K
    low_min = df['low'].rolling(n).min()
    high_max = df['high'].rolling(n).max()
    df['%K'] = ((df['close'] - low_min) / (high_max - low_min)) * 100

    # Aplicar un suavizado a %K
    df['%K_Smooth'] = df['%K'].rolling(smooth_k).mean()

    # Calcular %D
    df['%D'] = df['%K_Smooth'].rolling(smooth_d).mean()
    return df

def calc_bbands(self,df, window=20, num_std_dev=2):
    # Calcular la media móvil simple (banda central)
    df['SMA'] = df['close'].rolling(window=window).mean()

    # Calcular la desviación estándar (volatilidad)
    df['STD'] = df['close'].rolling(window=window).std()

    # Calcular las bandas de Bollinger
    df['banda_sup'] = df['SMA'] + (df['STD'] * num_std_dev)
    df['banda_inf'] = df['SMA'] - (df['STD'] * num_std_dev)

    return df
```

Figura 4. *Cálculo de indicadores.*

Una vez hemos procesado la información, procedemos a la creación de gráficos cuyo proceso se expone a continuación.

3.2 Fase 2- Creación de los gráficos

Para la creación de gráficos empleamos la librería bokeh que tiene una amplia gama de gráficos disponibles y otras funcionalidades que facilitan la creación e interacción con estos.

Debido a que contamos con 720 datos, definimos unos límites en el rango del eje x correspondiente al eje de tiempo para observar un máximo de 46 datos al mismo tiempo. Para ello seguimos el proceso mostrado en la figura 5, donde inicialmente se invierte el dataframe ya que nos interesa mostrar las cotizaciones de más recientes a más antiguas y posteriormente empleando los últimos 46 registros definimos los valores inicial y final que se asignan posteriormente a los rangos del eje X.

```
def crear_graf(self,df,ind,vol,div):
    df=df[::-1]
    df_reduce=df[:46].copy()
    start=df_reduce['Date'].values.min()
    end=df_reduce['Date'].values.max()
```

Figura 5. *Definición del rango inicial.*

El primer gráfico a generar es el gráfico de velas que muestra el valor de las cotizaciones, para ello dentro de la figura de bokeh por cada dato se generan dos segmentos, uno hace referencia a el color y su tamaño viene definido por el valor de apertura y de cierre de la cotización mientras que el otro viene definido por los valores máximo y mínimo.

Una vez definidos los segmentos, se configuran algunos parámetros del gráfico como los rangos y la visualización del título. Finalmente, creamos una lista llamada children donde se almacenan los diferentes gráficos a mostrar en función de las elecciones del usuario.

```
candle=figure(x_axis_type='datetime',height=300,x_range=(df['Date'].values[-1],df['Date'].values[0]),
              tooltips=[('Date','@Date_str'),('open','@open'),('close','@close'),('high','@high'),('low','@low'),
                        ('color','@color')],y_axis_label='Precio',toolbar_location=None,
                        title=f'{div}')

candle.segment('Date','low','Date','high',color='black',line_width=.5,source=df)
candle.segment('Date','open','Date','close',color='color',line_width=6,source=df)
candle.title.align='center'
candle.title.text_font_size='16pt'
candle.x_range.start= start
candle.x_range.end= end
candle.y_range.start=start_y
candle.y_range.end = end_y
children=[candle]
```

Figura 6. *Creación del gráfico de velas.*

En caso de haber seleccionado el indicador VWAP se grafica la línea sobre las cotizaciones.

```
if 'VWAP' in ind:
    candle.line('Date', 'VWAP', line_color="blue", line_width=.5, legend_label="VWAP-",source=df)
```

Figura 7. *Graficación del indicador VWAP.*

En caso de estar seleccionado el checkbox del volumen se crea un gráfico en la zona inferior del gráfico de cotizaciones donde se muestra el volumen en el mismo intervalo temporal que las cotizaciones y se añade a la lista children.

```
if vol:      mariol-lamas, 3 weeks ago • Add files via upload
    volumen=figure(x_axis_type='datetime',height=120,x_range=(df['Date'].values[-1],df['Date'].values[0]),y_axis_label='Volumen',
                   toolbar_location=None)
    volumen.segment('Date',0,'Date','volume',color='color',line_width=6,source=df)
    volumen.yaxis.formatter = NumeralTickFormatter(format="0")
    volumen.x_range=candle.x_range
    children.append(volumen)
```

Figura 8. *Creación del gráfico de volumen.*

Si el usuario ha seleccionado el indicador estocástico, generamos un gráfico donde mostramos los datos referentes a este indicador y añadimos el gráfico a la lista children.

```
if 'STOCH' in ind:
    estocast=figure(x_axis_type='datetime',height=100,x_range=(df['Date'].values[-1],df['Date'].values[0]),y_axis_label='STOCH',
                    toolbar_location=None)
    estocast.line('Date','%K_Smooth',line_color='blue',line_width=.5,legend_label='slowk',source=df)
    estocast.line('Date','%D',line_color='orange',line_width=.5,legend_label='slowd',source=df)
    estocast.x_range=candle.x_range
    children.append(estocast)
```

Figura 9. *Creación del gráfico del indicador estocástico.*

Este proceso será idéntico para cualquier indicador, en caso de que se trate de un indicador que deba mostrarse sobre el gráfico de cotizaciones se añadirá a este y en caso contrario se generará un nuevo gráfico y se añadirá a la lista children.

Una vez generados los gráficos necesarios, se retornan apilados verticalmente en el orden en que han sido introducidos en la lista children, finalmente, este gráfico de columna se retorna para ser mostrado en Streamlit.

```
return column(children=children,sizing_mode='scale_width')
```

Figura 10. *Retorno del gráfico de columna.*

3.3 Fase 3- Creación visualización con Streamlit.

La visualización se divide en dos bloques, a la izquierda se encuentra el bloque de opciones donde se seleccionan las divisas a graficar, los intervalos de los datos, la elección de indicadores, etc.

Esta estructura se logra empleando elementos interactivos de Streamlit como checkbox, multi selectores y otros elementos cuya definición se muestra en la figura 11, finalmente, retorna los elementos seleccionados por el usuario.

```
def barra_lat(self):

    with st.sidebar:
        st.write('Opciones del gráfico')
        divisa=st.selectbox(
            'Par de divisas',
            ('BTC/EUR','BTC/USD','ETH/EUR','ETH/USD')
        )
        intervalo = st.selectbox(
                'Intervalo',
                ('1m', '15m', '30m','1h','24h'),
                index=0)
        indicadores = st.multiselect(
                'Indicadores',
                ['VWAP', 'STOCH', 'BANDS'],
                [],)

        vol=st.checkbox('Mostrar volumen')
        self.test=st.checkbox('Realizar test')
    return divisa,intervalo,indicadores, vol
```

Figura 11. *Definición de la barra lateral.*

Por otro lado, el resto de la aplicación la ocupa el bloque central donde se muestran los distintos gráficos junto al título de la aplicación.Streamlit cuenta con elementos predefinidos para la muestra de gráficos de la librerías bokeh como se observa en la figura a continuación.

```
def centro(self,fig):
    st.header("INTERACTIVE TRADING VIEW     BTC & ETH", divider='rainbow')
    st.bokeh_chart(fig,use_container_width=True)
```

Figura 12. *Definición del bloque central.*

Tras estas tres fases, quedan definidos los elementos necesarios para un adecuado funcionamiento de la aplicación, sin embargo para tener un mayor control del flujo de ejecución de estas fases, la aplicación cuenta con un método llamado run cuyo funcionamiento se explica en la próxima fase.

3.4 Fase 4- Flujo de ejecución.

El método run(), inicialmente crea la configuracion de la aplicación, a continuación, llamando al método barra_lat() obtenemos la selección del usuario que posteriormente empleamos para obtener los datos llamado al método obt_datos(), en caso de que la obtención de datos sea correcta y el dataframe contenga información, se crean los gráficos correspondientes empleando el método crear_graf() cuya información se emplea para llamar al método centro() encargado de dibujar los gráficos con Streamlit, esta implementación se muestra en la figura 13.

```
def run(self):
    st.set_page_config(page_title='INTERACTIVE TRADING VIEW')
    divisa, intervalo, indicadores, vol=self.barra_lat()
    dataframe=self.obt_datos(divisa,self.intervalos[intervalo],indicadores)
    if not dataframe.empty:
        fig=self.crear_graf(dataframe,indicadores,vol,divisa)
        self.centro(fig)
```

Figura 13. *Definición del método run().*

3.5 Testeo y cobertura.

Para testear el funcionamiento adecuado de la aplicación, se han diseñado diversos testeos para comprobar la funcionalidad crítica, en este caso la obtención de datos.

Para ello se han definido tests unitarios empleando la librería de python llamada unittest, donde definiremos los test para cada divisa en los diferentes intervalos de tiempo disponibles, para ello, creamos una clase llamada Test que hereda de la clase TestCase de la librería unittest. Dentro de esta definimos un método setUp donde realizamos la preparación para posteriormente realizar los test donde definimos las monedas y los intervalos sobre los que realizaremos los tests.

```
class Test(unittest.TestCase):
    def setUp(self):
        self.divisas=('BTC/EUR','BTC/USD','ETH/EUR','ETH/USD','ETH/BTC')
        self.intervalos=(1,15,30,60,60*24)
        self.columnas=['timestamp', 'open', 'high','low','close','_','volume','conteo']
        self.app=App()
```

Figura 14. *Definición del método setUp().*

Para definir los tests creamos un método llamado test_obtención_datos dentro del cual comprobamos por cada tipo de cotización y cada intervalo disponible si llamando al método obt_datos de la aplicación retorna valores para los parámetros indicados y si se han creado adecuadamente las columnas correspondientes. Esta definición en python se realiza de la siguiente manera:

```python
def test_obtencion_datos(self):
    for divisa in self.divisas:
        for interval in self.intervalos:
            df=self.app.obt_datos(divisa,interval,[])
            self.assertEqual(df.empty,False,f'No se han recibido datos para la divisa {divisa} ')
            for col in self.columnas:
                self.assertIn(col,df.columns,f'La columna {col} no existe en el DataFrame')
```

Figura 15. *Definición del método test_obtencion_datos().*

Este método se ejecutará al seleccionar el checkbox de realizar test, de esta manera evitamos que Streamlit ejecute el testeo en cada modificación del usuario, para ello, en el método run descrito anteriormente definimos el siguiente proceso:

```python
if self.test:
    try:
        test=Test()
        test.setUp()
        test.test_obtencion_datos()
        st.write('Todo ha ido correcto')
    except AssertionError:
        st.write('\n Han ocurrido errores durante el testeo')
```

Figura 16. *Ejecución de los métodos.*

4.Ejecución y utilización del código

Para acceder a la aplicación hay dos métodos disponibles, puede accederse mediante la web en la siguiente url: https://proyecto-mario.streamlit.app/ , en caso de no disponer de conexión a la red, realizaremos el siguiente procedimiento:

Paso 1.- Instalación de GIT.

En el siguiente enlace se explica el proceso de instalación de la herramienta para los distintos sistemas operativos:

https://www.hostinger.es/tutoriales/instalar-git-en-distintos-sistemas-operativos .

Paso 2.- Instalación de anaconda.

El proceso de instalación de anaconda se muestra en el siguiente enlace: https://www.aprendemachinelearning.com/instalar-ambiente-de-desarrollo-python-anaconda-para-aprendizaje-automatico/ .

Paso 3.- Creación del entorno y clonación del repositorio.

Inicialmente creamos un entorno virtual con la versión 3.11 de Python ejecutando el siguiente comando en la terminal o el CMD.

conda create -n nombre_entorno python=3.11

Una vez creado el entorno, creamos una carpeta en el escritorio, abrimos la terminal en dicha carpeta y ejecutamos los siguientes comandos:

-Activación del entorno -> **conda activate nombre_entorno**

-Clonación del repositorio -> **git clone https://github.com/mariol-lamas/work.git**

-Directorio de trabajo -> **cd work**

-Instalación de paquetes -> **pip install -r requirements.txt**

-Ejecución del programa (Windows)-> **streamlit run src\Trade_view_MLA\proyecto_final.py**

-Ejecución del programa (Linux y MacOS) -> **streamlit run src/Trade_view_MLA/proyecto_final.py**

Tras ejecutar el programa se abrirá en el navegador una pestaña donde se podrá interactuar con la aplicación.

5. Conclusiones

En conclusión, este proyecto me ha permitido poner a prueba mis conocimientos en el lenguaje de programación python tratando diversos temas como la creación de clases y funciones, la interacción con APIs, el preprocesamiento de información con pandas y numpy, la creación de gráficos, el diseño de tests para asegurar el adecuado funcionamiento y la distribución del código mediante github y en este caso Streamlit. Para más información acerca del código correspondiente, está disponible en el siguiente repositorio de github: https://github.com/mariol-lamas/work.git.